COP8™ Microcontroller

COP888 Feature Family User's Manual

Literature Number 620897-003
September 1996

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
A	03/89	First Release COP888 User's Manual NSC Publication Number 420411060-001A
B	07/94	Reformatted and updated manual to include information on new devices added to the COP888 Feature Family.
-001	08/95	Modified to incorporate information on the COP888EK microcontroller.
-002	01/96	Updated Chapter 11 to include information on new devices.
-003	09/96	Updated manual to include new devices

National Semiconductor

READER'S COMMENT FORM

In the interest of improving our documentation, National Semiconductor invites your comments on this manual.

Please restrict your comments to the documentation. Technical Support may be contacted at:

U.S. and Canada	(800) 272-9959
Europe Deutsch	+49 (0) 180-530 85 85
English	+49 (0) 180-532 78 32
Français	+49 (0) 180-532 93 58
Italian	+49 (0) 180-534 16 80
Japan	+81-043-299-2309
Australia	(+61) 3-9558-9999
India	(+91) 80-559-9467

S. E. Asia	
Beijing	(+86) 10-6856-8601
Shanghai	(+86) 21-6415-4092
Hong Kong	(+852) 2737-1600
Korea	(+82) 2-3771-6909
Malaysia	(+60-4) 644-9061
Singapore	(+65) 255-2226
Taiwan	+886-2-521-3288

Please rate this document according to the following categories. Include your comments below.

	EXCELLENT	GOOD	ADEQUATE	FAIR	POOR
Readability (style)	❒	❒	❒	❒	❒
Technical Accuracy	❒	❒	❒	❒	❒
Fulfills Needs	❒	❒	❒	❒	❒
Organization	❒	❒	❒	❒	❒
Presentation (format)	❒	❒	❒	❒	❒
Depth of Coverage	❒	❒	❒	❒	❒
Overall Quality	❒	❒	❒	❒	❒

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

Do you require a response? ❒Yes ❒No PHONE _____

Comments: _____

_____

COP888 Feature Family User's Manual

FOLD, STAPLE, AND MAIL

Literature Number 620897-003

Please return this to:

In U.S.A.
(Including Canada and South America):

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Attn: Technical Publications, M/S E215

In South East Asia
(Including Australia, New Zealand and India):

National Semiconductor H.K. Ltd.
15th Floor, Straight Block
Ocean Centre
5 Canton Road
Tsim Sha Tsui, Kowloon
Hong Kong
Attn: Business Center Marketing

In Europe
(Including Israel):

National Semiconductor GmbH
Livry-Garge-Stra. #10
D-82256 Fuerstenfeldbruck
Germany
Attn: Business Center Marketing

In Japan

National Semiconductor Japan Ltd.
Sumitomo Chemical Engineering Building, 7F
1-7-1, Naskase, Mihama-Ku
Chiba-City, Ciba Prefecture 261
Japan
Attn: Business Center Marketing

(fold)

Place Stamp Here

National Semiconductor

PREFACE

The COP8™ Feature Family of 8-bit microcontrollers is ideally suited to embedded controller applications such as keyboard interfaces, electronic telephones, home appliances, and ABS systems. The design of this family takes advantage of National Semiconductor's M^2CMOS™ manufacturing technology, providing a useful combination of high performance, low power consumption, and reasonable cost. The rich instruction set and flexible addressing modes of the COP888 controllers contribute to their high performance and code efficiency.

This manual describes the features, architecture, instruction set, and usage of the COP888 microcontrollers. The beginning chapters describe the general features found in all family members. Later chapters describe the individual family members and their specific features.

Chapter 1, OVERVIEW, provides an overview of the COP888 family and compares the features of different family members.

Chapter 2, ARCHITECTURE, describes the overall architecture of the COP888 microcontroller, including the CPU core, registers, memory organization, reset operation, and clock options.

Chapter 3, INTERRUPTS, describes the device interrupts and how they are used. The types of interrupts vary from one family member to another.

Chapter 4, TIMERS, describes the on-chip timers and their operating modes. The number and types of timers vary from one family member to another.

Chapter 5, MICROWIRE/PLUS, describes the microcontroller's MICROWIRE/PLUS serial interface and its operating modes.

Chapter 6, POWER SAVE MODES, describes the special operating modes in which the microcontroller is shut down, reducing power consumption to a very low value while maintaining the processor status and all register contents. All family members have a HALT mode, and some also have an IDLE mode that maintains real time while the processor is shut down.

Chapter 7, INPUT/OUTPUT, describes the input/output ports of the microcontroller and how they are used. The number and types of ports vary from one family member to another.

Chapter 8, WATCHDOG AND CLOCK MONITOR, describes an internal circuit available in some COP888 devices that monitors the operation of the microcontroller and reports an abnormal condition by issuing a signal on an output pin.

Chapter 9, UART, describes the Universal Asynchronous Receiver/Transmitter unit available in some COP888 devices. A UART is provided to support serial data communications.

Chapter 10, ANALOG-TO-DIGITAL CONVERTER, describes the multi-channel Analog-to-Digital Converter unit available in some COP888 devices. An A/D Converter converts an analog signal into an 8-bit digital value.

Chapter 11, CAN INTERFACE, describes the Controller Area Network (CAN) interface provided in some COP888 devices. CAN is a communications network standard often used in automotive and industrial control applications.

Chapter 12, INSTRUCTION SET, describes the instruction set of the COP888 microcontrollers, including detailed descriptions of each instruction.

The remaining chapters describe the specific features of different COP888 Feature Family members. The chapters are in alphabetical order by COP888 device name: COP888BC, COP888CG, and so on. Only device-specific information is provided in these chapters. Features common to several or all COP888 devices are described in the earlier chapters.

Appendix A, APPLICATION HINTS, provides additional information that may be useful in implementing a design.

Appendix B, ELECTRICAL CHARACTERIZATION DATA, shows the general electrical characteristics of COP888 devices such as power consumption, source current, and sink current. For additional information, consult the device-specific data sheets.

The information contained in this manual is for reference only and is subject to change without notice.

No part of this document may be reproduced in any form or by any means without the prior written consent of National Semiconductor Corporation.

COP8, M^2CMOS are trademarks of National Semiconductor Corporation.

CONTENTS

Chapter 1 OVERVIEW

1.1 INTRODUCTION 1-1
1.2 BASIC FEATURES 1-1
1.3 DEVICE-SPECIFIC FEATURES 1-2

Chapter 2 ARCHITECTURE

2.1 INTRODUCTION 2-1
2.2 BLOCK DIAGRAM 2-1
2.3 MEMORY ORGANIZATION 2-2
2.3.1 Program Memory 2-3
2.3.2 Data Memory 2-3
2.3.3 Memory-Mapped I/O Registers 2-7
2.4 CORE REGISTERS 2-8
2.4.1 Accumulator 2-8
2.4.2 Program Counter 2-9
2.4.3 Control Registers 2-9
2.4.4 Data Registers 2-10
2.4.5 MICROWIRE/PLUS Register 2-12
2.4.6 Timer Registers 2-12
2.5 CPU OPERATION 2-12
2.5.1 Memory Fetches 2-14
2.5.2 Instruction Decoding and Execution 2-15
2.5.3 Interrupt and Error Handling 2-20
2.6 RESET 2-21
2.7 CLOCK OPTIONS 2-22
2.7.1 Crystal Oscillator 2-22
2.7.2 RC Oscillator 2-23

Chapter 3 INTERRUPTS

3.1 INTRODUCTION 3-1
3.2 VIS INSTRUCTION AND VECTOR TABLE 3-2
3.3 CONTEXT SWITCHING 3-4
3.4 MASKABLE INTERRUPTS 3-5
3.5 NON-MASKABLE INTERRUPTS 3-7
3.5.1 Non-Maskable Interrupt Pending Flags 3-7
3.5.2 Software Trap 3-8
3.5.3 NMI 3-9
3.5.4 Software Trap and NMI Interaction 3-9
3.6 INTERRUPT SUMMARY 3-10

Chapter 4 TIMERS

4.1 INTRODUCTION 4-1
4.2 TIMER/COUNTER BLOCK 4-1
4.3 TIMER CONTROL BITS 4-2
4.4 TIMER OPERATING MODES 4-3

4.4.1 PWM Mode 4-4
4.4.2 External Event Counter Mode 4-5
4.4.3 Input Capture Mode 4-7
4.5 ADDITIONAL GENERAL-PURPOSE TIMERS 4-10
4.6 IDLE TIMER 4-11

Chapter 5 MICROWIRE/PLUS

5.1 INTRODUCTION 5-13
5.2 THEORY OF OPERATION 5-14
5.2.1 Timing 5-14
5.2.2 Port G Configuration 5-16
5.2.3 SK Clock Frequency 5-16
5.2.4 Busy Flag and Interrupt 5-17
5.3 MASTER MODE OPERATION EXAMPLE 5-18
5.4 SLAVE MODE OPERATION EXAMPLE 5-19

Chapter 6 POWER SAVE MODES

6.1 INTRODUCTION 6-1
6.2 ENTERING THE HALT MODE 6-1
6.2.1 Clock-Stopping Method 6-1
6.2.2 Port G Method 6-2
6.3 EXITING THE HALT MODE 6-2
6.3.1 HALT Exit Using Reset 6-2
6.3.2 HALT Exit Using Multi-Input Wakeup 6-2
6.3.3 HALT Exit Using G7 Pin 6-3
6.4 IDLE MODE 6-3
6.5 HALT/IDLE AND WATCHDOG OPERATION 6-4
6.6 NMI EXIT FROM HALT/IDLE 6-4

Chapter 7 INPUT/OUTPUT

7.1 INTRODUCTION 7-1
7.2 PORT C 7-2
7.3 PORT D 7-2
7.4 PORT G 7-3
7.5 PORT I 7-3
7.6 PORT L 7-3
7.7 ALTERNATE FUNCTIONS 7-4
7.7.1 Port G Alternate Functions 7-4
7.7.2 Multi-Input Wakeup/Interrupt 7-4

Chapter 8 WATCHDOG AND CLOCK MONITOR

8.1 INTRODUCTION 8-1
8.2 WATCHDOG OPERATION 8-1
8.3 CLOCK MONITOR OPERATION 8-3
8.4 CONFIGURATION 8-3
8.5 ERROR REPORT ON WDOUT 8-5

Chapter 9 UART

9.1 UART 9-1

9.1.1 UART Operation Overview 9-1
9.1.2 UART Registers 9-2
9.1.3 UART Interface 9-6
9.1.4 Asynchronous Mode 9-6
9.1.5 Synchronous Mode 9-8
9.1.6 Framing Formats 9-8
9.1.7 Reset Initialization 9-10
9.1.8 HALT/IDLE Mode Reinitialization 9-11
9.1.9 Baud Clock Generation 9-12
9.1.10 UART Interrupts 9-18
9.1.11 UART Error Flags 9-18
9.1.12 Diagnostic Testing 9-19
9.1.13 Attention Mode 9-19
9.1.14 Break Generation and Detection 9-20

Chapter 10 ANALOG-TO-DIGITAL CONVERTER

10.1 INTRODUCTION 10-1
10.2 A/D OPERATION 10-1
10.3 A/D CONVERTER REGISTERS 10-3
10.3.1 Busy Bit 10-3
10.3.2 Prescaler Selection 10-4
10.3.3 Single Conversion or Continuous Mode 10-4
10.3.4 Channel Selection 10-5
10.4 MULTI-CHANNEL CONVERSION 10-5
10.5 SPEED, ACCURACY, AND HARDWARE CONSIDERATIONS 10-7

Chapter 11 CAN INTERFACE

11.1 INTRODUCTION 11-1
11.2 BASIC CAN CONCEPTS 11-1
11.2.1 Physical Layer 11-2
11.2.2 Modules and Frames 11-2
11.2.3 Contention-Based Arbitration 11-3
11.2.4 Error Detection and Error Handling 11-4
11.3 FRAMES 11-4
11.3.1 Frame Fields 11-5
11.3.2 Frame Coding 11-7
11.3.3 Data Frames and Remote Frames 11-8
11.3.4 Interframe Spaces 11-10
11.3.5 Error Frames 11-11
11.3.6 Overload Frames 11-13
11.3.7 Frame Validation 11-14
11.4 FRAME ARBITRATION AND PRIORITY 11-14
11.5 ACCEPTANCE FILTERING 11-14
11.6 ERROR MANAGEMENT AND DETECTION 11-14
11.6.1 Error Types 11-15
11.6.2 Error Counters 11-16
11.6.3 Module Error States 11-17
11.7 SYNCHRONIZATION 11-19
11.7.1 Hard Synchronization 11-20

11.7.2 Soft Synchronization 11-20
11.8 COP888 CAN Interface Block 11-22
11.8.1 Interface Management Logic 11-22
11.8.2 Bit Stream Processor 11-22
11.8.3 Transceiver Logic 11-22
11.8.4 Error Management Logic 11-22
11.8.5 CRC Generator and Register 11-24
11.8.6 Receive/Transmit (Rx/Tx) Register 11-24
11.8.7 Output Drivers and Input Comparators 11-24
11.8.8 Clock Generator 11-25
11.8.9 CAN Register Block 11-28
11.9 EXTENDED FRAMES 11-40

Chapter 12 INSTRUCTION SET

12.1 INTRODUCTION 12-1
12.2 INSTRUCTION FEATURES 12-1
12.3 ADDRESSING MODES 12-1
12.3.1 Operand Addressing Modes 12-2
12.3.2 Transfer-of-Control Addressing Modes 12-5
12.4 INSTRUCTION TYPES 12-7
12.5 DIFFERENCES BETWEEN COP800 AND COP888 12-9
12.6 DETAILED FUNCTIONAL DESCRIPTIONS OF INSTRUCTIONS . 12-10
12.6.1 ADC— Add with Carry 12-13
12.6.2 ADD — Add 12-14
12.6.3 AND — And 12-15
12.6.4 ANDSZ — And, Skip if Zero 12-16
12.6.5 CLR — Clear Accumulator 12-17
12.6.6 DCOR — Decimal Correct 12-18
12.6.7 DEC — Decrement Accumulator 12-19
12.6.8 DRSZ REG# — Decrement Register and Skip if Result is Zero 12-20
12.6.9 IFBIT — Test Bit 12-21
12.6.10 IFBNE # — If B Pointer Not Equal 12-22
12.6.11 IFC — Test if Carry 12-23
12.6.12 IFEQ — Test if Equal 12-24
12.6.13 IFGT — Test if Greater Than 12-25
12.6.14 IFNC — Test If No Carry 12-26
12.6.15 IFNE — Test If Not Equal 12-27
12.6.16 INC — Increment Accumulator 12-28
12.6.17 INTR — Interrupt (Software Trap) 12-29
12.6.18 JID — Jump Indirect 12-30
12.6.19 JMP — Jump Absolute 12-31
12.6.20 JMPL — Jump Absolute Long 12-32
12.6.21 JP — Jump Relative 12-33
12.6.22 JSR — Jump Subroutine 12-34
12.6.23 JSRL — Jump Subroutine Long 12-35
12.6.24 LAID — Load Accumulator Indirect 12-36
12.6.25 LD — Load Accumulator 12-37
12.6.26 LD — Load B Pointer 12-39
12.6.27 LD — Load Memory 12-40

12.6.28 LD — Load Register 12-41
12.6.29 NOP — No Operation 12-42
12.6.30 OR — Or 12-43
12.6.31 POP — Pop Stack 12-44
12.6.32 PUSH — Push Stack 12-45
12.6.33 RBIT — Reset Memory Bit 12-46
12.6.34 RC — Reset Carry 12-47
12.6.35 RET — Return from Subroutine 12-48
12.6.36 RETI — Return from Interrupt 12-49
12.6.37 RETSK — Return and Skip 12-50
12.6.38 RLC — Rotate Accumulator Left Through Carry 12-51
12.6.39 RPND — Reset Pending 12-52
12.6.40 RRC — Rotate Accumulator Right Through Carry 12-53
12.6.41 SBIT — Set Memory Bit 12-54
12.6.42 SC — Set Carry 12-55
12.6.43 SUBC — Subtract with Carry 12-56
12.6.44 SWAP — Swap Nibbles of Accumulator 12-58
12.6.45 VIS — Vector Interrupt Select 12-59
12.6.46 X — Exchange Memory with Accumulator 12-60
12.6.47 XOR — Exclusive Or 12-62
12.7 INSTRUCTION OPERATIONS SUMMARY 12-63
12.8 INSTRUCTION BYTES AND CYCLES 12-64

Chapter 13 COP884BC

13.1 INTRODUCTION 13-1
13.2 BLOCK DIAGRAM 13-1
13.3 DEVICE PINOUTS/PACKAGES 13-2
13.4 PIN DESCRIPTIONS 13-3
13.5 INPUT/OUTPUT PORTS 13-4
13.6 PROGRAM MEMORY 13-5
13.7 DATA MEMORY 13-5
13.8 REGISTER BIT MAPS 13-5
13.9 MEMORY MAP 13-10
13.10 RESET 13-12
13.11 ON-CHIP POWER-ON RESET 13-13
13.12 INTERRUPTS 13-14
13.13 COMPARATORS 13-14
13.13.1 COMPARATOR CONTROL REGISTER (CMPSL) (00D3) . 13-16
13.14 PWM TIMER 13-17
13.14.1 PWM Mode 13-19
13.14.2 Frequency Monitor Mode 13-21
13.14.3 PWM Registers 13-23
13.15 CAN INTERFACE 13-25
13.16 MASK OPTIONS 13-25

Chapter 14 COP888CS/CG/EG/GG/HG/KG

14.1 INTRODUCTION 14-1
14.2 BLOCK DIAGRAMS 14-2
14.3 DEVICE PINOUTS/PACKAGES 14-2

14.4 PIN DESCRIPTIONS 14-3
14.5 INPUT/OUTPUT PORTS 14-6
14.6 PROGRAM MEMORY 14-7
14.7 DATA MEMORY 14-8
14.8 REGISTER BIT MAPS 14-8
14.9 MEMORY MAP 14-13
14.10 RESET 14-15
14.11 INTERRUPTS 14-17
14.12 COMPARATOR 14-17
14.13 UART 14-19
14.14 MASK OPTIONS 14-20

Chapter 15 COP888CF

15.1 INTRODUCTION 15-1
15.2 BLOCK DIAGRAM 15-1
15.3 DEVICE PINOUTS/PACKAGES 15-2
15.4 PIN DESCRIPTIONS 15-2
15.5 INPUT/OUTPUT PORTS 15-5
15.6 PROGRAM MEMORY 15-7
15.7 DATA MEMORY 15-7
15.8 REGISTER BIT MAPS 15-7
15.9 MEMORY MAP 15-9
15.10 RESET 15-12
15.11 INTERRUPTS 15-13
15.12 ANALOG-TO-DIGITAL CONVERTER 15-14
15.12.1 A/D Operation 15-14
15.12.2 A/D Converter Registers 15-15
15.12.3 Prescaler Selection 15-16
15.12.4 Single Conversion or Continuous Mode 15-17
15.12.5 Channel Selection 15-17
15.12.6 Multi-Channel Conversion 15-18
15.12.7 Speed, Accuracy, and Hardware Considerations 15-19
15.13 MASK OPTIONS 15-21

Chapter 16 COP888CL

16.1 INTRODUCTION 16-1
16.2 BLOCK DIAGRAM 16-1
16.3 DEVICE PINOUTS/PACKAGES 16-1
16.4 PIN DESCRIPTIONS 16-2
16.5 INPUT/OUTPUT PORTS 16-5
16.6 PROGRAM MEMORY 16-7
16.7 DATA MEMORY 16-7
16.8 REGISTER BIT MAPS 16-7
16.9 MEMORY MAP 16-9
16.10 RESET 16-11
16.11 INTERRUPTS 16-12
16.12 MASK OPTIONS 16-13

Chapter 17 COP888EB

17.1 INTRODUCTION 17-1
17.2 BLOCK DIAGRAM 17-1
17.3 DEVICE PINOUTS/PACKAGES 17-2
17.4 PIN DESCRIPTIONS 17-4
17.5 INPUT/OUTPUT PORTS 17-7
17.6 PROGRAM MEMORY 17-8
17.7 DATA MEMORY 17-8
17.8 REGISTER BIT MAPS 17-8
17.9 MEMORY MAP 17-16
17.10 RESET 17-20
17.11 INTERRUPTS 17-21
17.12 UART 17-21
17.13 A/D CONVERTER 17-23
17.14 CAN INTERFACE 17-23
17.14.1 HALT/IDLE and the CAN Interface 17-23
17.15 SPI INTERFACE 17-24
17.15.1 Control Register Format 17-26
17.15.2 Status Register Format 17-27
17.15.3 SS-Expander 17-28
17.15.4 SPI Synchronization 17-30
17.15.5 Transmission Start in Master Mode 17-30
17.15.6 SPI Receive Data Register (SPIRXD) 17-30
17.15.7 SPI Transmit Data Register (SPITXD) 17-30
17.15.8 SPI Receive FIFO 17-30
17.15.9 SPI Transmit FIFO 17-31
17.15.10 Pointer Operation with SPI Disabled 17-32
17.15.11 Loop Back Mode 17-32
17.15.12 HALT/IDLE Mode 17-32
17.16 MASK OPTIONS 17-33

Chapter 18 COP888EK

18.1 INTRODUCTION 18-1
18.2 BLOCK DIAGRAM 18-1
18.3 DEVICE PINOUTS/PACKAGES 18-2
18.4 PIN DESCRIPTIONS 18-2
18.5 INPUT/OUTPUT PORTS 18-5
18.6 PROGRAM MEMORY 18-7
18.7 DATA MEMORY 18-7
18.8 REGISTER BIT MAPS 18-7
18.9 ANALOG FUNCTION BLOCK 18-11
18.10 MEMORY MAP 18-14
18.11 RESET 18-16
18.12 INTERRUPTS 18-17
18.13 MASK OPTIONS 18-18

Chapter 19 COP888FH

19.1 INTRODUCTION 19-1
19.2 BLOCK DIAGRAM 19-1

19.3 DEVICE PINOUTS/PACKAGES 19-2
19.4 PIN DESCRIPTIONS 19-2
19.5 INPUT/OUTPUT PORTS 19-5
19.6 PROGRAM MEMORY 19-7
19.7 DATA MEMORY 19-7
19.8 REGISTER BIT MAPS 19-7
19.9 MEMORY MAP 19-12
19.10 RESET 19-15
19.11 INTERRUPTS 19-17
19.12 COMPARATOR 19-17
19.13 UART 19-19
19.14 HARDWARE MULTIPLY/DIVIDE 19-20
19.14.1 Multiply/Divide Data Registers 19-20
19.14.2 Multiply/Divide Control Register 19-21
19.14.3 Multiply Operation 19-21
19.14.4 Divide Operation 19-21
19.14.5 Reset State 19-22
19.15 MASK OPTIONS 19-22

Chapter 20 COP888GW

20.1 INTRODUCTION 20-1
20.2 BLOCK DIAGRAM 20-1
20.3 DEVICE PINOUTS/PACKAGES 20-2
20.4 PIN DESCRIPTIONS 20-2
20.5 INPUT/OUTPUT PORTS 20-5
20.6 PROGRAM MEMORY 20-7
20.7 DATA MEMORY 20-7
20.8 REGISTER BIT MAPS 20-7
20.9 MEMORY MAP 20-14
20.10 RESET 20-17
20.11 INTERRUPTS 20-19
20.12 UART 20-19
20.13 HARDWARE MULTIPLY/DIVIDE 20-20
20.13.1 Multiply/Divide Data Registers 20-21
20.13.2 Multiply/Divide Control Register 20-21
20.13.3 Multiply Operation 20-22
20.13.4 Divide Operation 20-22
20.13.5 Reset State 20-22
20.14 CAPTURE TIMERS 20-23
20.15 CAPTURE TIMER REGISTERS 20-24
20.16 CAPTURE TIMER OPERATION 20-25
20.16.1 Initialization 20-25
20.16.2 Port Configuration 20-26
20.16.3 Interrupts 20-26
20.16.4 Starting the Timer 20-26
20.16.5 Edge Transitions and Underflow 20-27
20.16.6 Stopping the Capture Timer 20-27
20.17 RESET STATE 20-28
20.18 PULSE TRAIN GENERATORS 20-28
20.19 PULSE TRAIN GENERATOR REGISTERS 20-29

20.20 PULSE TRAIN GENERATOR OPERATION 20-31
20.20.1 Initialization 20-32
20.20.2 Port Configuration 20-33
20.20.3 Interrupts 20-33
20.20.4 Starting the Pulse Train 20-33
20.20.5 Stopping the Pulse Train Generator 20-33
20.21 Reset State 20-34
20.22 MASK OPTIONS 20-34

Appendix A APPLICATION HINTS

A.1 INTRODUCTION A-1
A.2 MICROWIRE/PLUS INTERFACE A-1
A.2.1 MICROWIRE/PLUS Master/Slave Protocol A-1
A.2.2 MICROWIRE/PLUS Continuous Mode A-3
A.2.3 MICROWIRE/PLUS Fast Burst Output A-4
A.2.4 NMC93C06-COP888CL Interface A-5
A.3 TIMER APPLICATIONS A-9
A.3.1 Timer Capture Example A-9
A.3.2 External Event Counter Example A-10
A.4 TRIAC CONTROL A-11
A.5 ANALOG-TO-DIGITAL CONVERSION USING ON-CHIP COMPARATOR. A-14
A.6 BATTERY-POWERED WEIGHT MEASUREMENT. A-16
A.7 ZERO CROSS DETECTION A-18
A.8 INDUSTRIAL TIMER A-19
A.9 PROGRAMMING EXAMPLES A-19
A.9.1 Clear RAM A-19
A.9.2 Binary/BCD Arithmetic Operations A-21
A.9.3 Binary Multiplication A-24
A.9.4 Binary Division A-25
A.10 EXTERNAL POWER WAKEUP CIRCUIT. A-27
A.11 WATCHDOG RESET CIRCUIT A-30
A.12 INPUT PROTECTION ON COP888 PINS A-30
A.13 ELECTROMAGNETIC INTERFERENCE (EMI) CONSIDERATIONS. A-33
A.13.1 Introduction A-33
A.13.2 Emission Predictions A-33
A.13.3 Board Layout A-35
A.13.4 Decoupling A-35
A.13.5 Output Series Resistance A-36
A.13.6 Oscillator Control A-37
A.13.7 Mechanical Shielding A-37
A.13.8 Conclusion A-37
A.14 DAIL-A-HELPER SERVICE A-38

Appendix B ELECTRICAL CHARACTERIZATION DATA

INDEX

LIST OF FIGURES

Figure 2-1 COP888 Block Diagram 2-2
Figure 2-2 A Basic Memory Map 2-4
Figure 2-1 COP888 Block Diagram 2-2
Figure 2-2 A Basic Memory Map 2-4
Figure 2-3 Memory Map with Data Segment Extension 2-6
Figure 2-4 Control Logic and ALU Interface 2-13
Figure 2-5 Crystal Oscillator Circuit 2-22
Figure 2-6 RC Oscillator Circuit 2-23
Figure 3-1 COP888 Interrupt Block Diagram 3-2
Figure 4-1 Timer in PWM Mode 4-4
Figure 4-2 Timer in External Event Counter Mode 4-6
Figure 4-3 Timer in Input Capture Mode 4-8
Figure 4-4 IDLE Timer (Timer T0) Block Diagram 4-11
Figure 5-1 MICROWIRE/PLUS Example 5-13
Figure 5-2 MICROWIRE/PLUS Circuit Block Diagram 5-14
Figure 5-3 MICROWIRE/PLUS Interface Timing, Standard SK Mode 5-15
Figure 5-4 MICROWIRE/PLUS Interface Timing, Alternate SK Mode 5-15
Figure 7-1 COP888 Port Structure 7-1
Figure 7-2 Multi-Input Wakeup/Interrupt Logic 7-5
Figure 8-1 Watchdog Logic Block Diagram 8-2
Figure 8-2 Watchdog Service Register (WDSVR) Format 8-4
Figure 9-1 UART Block Diagram 9-2
Figure 9-2 UART Transmitter Timing, Asynchronous Mode 9-7
Figure 9-3 UART Receiver Bit Sampling, Asynchronous Mode 9-7
Figure 9-4 UART Receiver Timing, Asynchronous Mode 9-7
Figure 9-5 UART Synchronous Mode Timing. 9-8
Figure 9-6 UART Framing Formats 9-9
Figure 9-7 UART Baud Clock Generation Block Diagram 9-12
Figure 9-8 UART Baud Clock Divisor Registers 9-14
Figure 9-9 UART Diagnostic Mode Loopback Connection 9-20
Figure 10-1 COP888 A/D Converter Block Diagram 10-2
Figure 10-2 A/D Conversion Routine 10-6
Figure 10-3 Analog Input Pin Internal Operation 10-7
Figure 11-1 CAN Message Arbitration 11-3
Figure 11-2 Data Frame and Remote Frame Formats. 11-6
Figure 11-3 Interframe Space Format 11-7
Figure 11-4 Error Frame and Overload Frame Format. 11-7
Figure 11-5 Normal Interframe Space Format. 11-10
Figure 11-6 Interframe Space Format for Error-Passive Transmitter 11-11
Figure 11-7 Error Frame from Error-active Transmitter 11-12
Figure 11-8 Error Frame from Error-passive Transmitter 11-13
Figure 11-9 CANModule Error States 11-18
Figure 11-10 Bit Timing. 11-20
Figure 11-11 Resynchronization by Lengthening Bit Time. 11-21
Figure 11-12 Resynchronization by Shortening Bit Time 11-21
Figure 11-13 CAN Interface Block Diagram. 11-23
Figure 11-14 External Transceiver Connections (ISO High Speed) 11-25
Figure 11-15 Bit Rate Generator. 11-26

Figure 11-16 Transmit Buffer Empty Timing 11-27
Figure 11-17 Receiver Status Bit Waveforms. 11-39
Figure 13-1 COP884BC Block Diagram 13-2
Figure 13-2 Device Package Pinouts. 13-2
Figure 13-3 Comparator 1 Block Diagram 13-15
Figure 13-4 Comparator 2 Block Diagram 13-16
Figure 13-5 PWM Timer Block Diagram (PWM Mode) 13-18
Figure 13-6 PWM Timer Block Diagram (Capture Mode). 13-19
Figure 13-7 PWM Mode Operation. 13-20
Figure 13-8 Frequency Monitor Mode Operation. 13-22
Figure 14-1 COP888CG Block Diagram 14-2
Figure 14-2 COP888CS Block Diagram 14-3
Figure 14-3 Device Package Pinouts. 14-4
Figure 15-1 COP888CF Block Diagram 15-2
Figure 15-2 Device Package Pinouts. 15-3
Figure 15-3 COP888CF A/D Converter Block Diagram. 15-14
Figure 15-4 A/D Conversion Routine 15-18
Figure 15-5 Analog Input Pin Internal Operation 15-20
Figure 16-1 COP888CL Block Diagram 16-2
Figure 16-2 Device Package Pinouts. 16-3
Figure 17-1 COP888EB Block Diagram 17-2
Figure 17-2 Device Package Pinouts. 17-3
Figure 17-3 SPI Frame Timing 17-24
Figure 17-4 SPI Block Diagram. 17-25
Figure 17-5 SS-Expander Logic. 17-29
Figure 17-6 SS-Expander Programming Timing 17-29
Figure 17-7 SPI Interface in Loop Back Mode 17-32
Figure 18-1 COP888EK Block Diagram 18-2
Figure 18-2 Device Package Pinouts. 18-3
Figure 18-3 COP888EK Analog Function Block. 18-11
Figure 19-1 COP888FH Block Diagram 19-2
Figure 19-2 Device Package Pinouts. 19-3
Figure 20-1 COP888GW Block Diagram. 20-2
Figure 20-2 Device Package Pinout. 20-3
Figure 20-3 Capture Timer 1 Block Diagram. 20-23
Figure 20-4 Pulse Train Generator 1 Block Diagram 20-29
Figure A-1 MICROWIRE/PLUS Sample Protocol Timing A-2
Figure A-2 MICROWIRE/PLUS Fast Burt Timing A-5
Figure A-3 NMC93C06-COP888CL Interface. A-5
Figure A-4 Timer Capture Application A-9
Figure A-5 A/D Conversion Using On-board Comparator and Timer T1 . . . . . A-15
Figure A-6 Battery-powered Weight Measurement A-17
Figure A-7 Industrial Timer Application. A-20
Figure A-8 Power Wakeup Using An NPN Transistor. A-28
Figure A-9 Power Wakeup Using Diodes And Resistors A-29
Figure A-10 Watchdog Reset Circuit. A-30
Figure A-11 Ports L/C/G Input Protection (Except G6) A-31
Figure A-12 Port I Input Protection. A-31
Figure A-13 Diode Equivalent of Input Protection. A-32
Figure A-14 External Protection of Inputs A-33

LIST OF TABLES

Table 2-1 Data Memory Map 2-5
Table 2-2 I/O Port Configuration 2-8
Table 2-3 PSW Register Bits 2-9
Table 2-4 CNTRL Register Bits 2-10
Table 2-5 ICNTRL Register Bits 2-10
Table 3-1 Interrupt Vector Table 3-4
Table 4-1 Timer Control Bits 4-2
Table 4-2 Timer Mode Control Bits 4-3
Table 5-1 Port G Configuration Register Bits 5-16
Table 5-2 Master Mode Clock Select Bits 5-17
Table 9-1 ENU, UART Control and Status Register (Address xxBA) 9-3
Table 9-2 ENUR, UART Receive Control and Status Register (Address xxBB) . . 9-4
Table 9-3 ENUI, UART Interrupt and Clock Source Register (Address xxBC) . . 9-4
Table 9-4 BAUD, UART Baud Register (Address xxBD) 9-5
Table 9-5 PSR, UART Prescaler Select Register (Address xxBE) 9-5
Table 9-6 UART Clock Sources (Asynchronous Mode) 9-13
Table 9-7 UART Clock Sources (Synchronous Mode) 9-13
Table 9-8 UART Prescaler Factors 9-15
Table 9-9 UART Baud Rate Divisors, 1.8432 MHz Prescaler Output 9-16
Table 10-1 ENAD, A/D Converter Control Register 10-3
Table 10-2 A/D Prescaler Options 10-4
Table 10-3 A/D Channel Selection 10-5
Table 11-1 Transmit Error Counter Handling 11-16
Table 11-2 Receive Error Counter Handling 11-17
Table 11-3 CAN Timing (CKI = 10 MHz t_C = 1µS) 11-27
Table 11-4 CAN Interface Registers 11-28
Table 11-5 Transmit Data Length Code and Identifier Low register (TDLC) . . 11-29
Table 11-6 Transmit Data Length Codes 11-29
Table 11-7 Transmit Identifier High (TID) 11-30
Table 11-8 Receive Data Length Code and Identifier Low register (RIDL) (00B6) 11-31
Table 11-9 Transmit Data Length Codes 11-31
Table 11-10 Receive Identifier High 11-32
Table 11-11 CAN Prescaler Register 11-32
Table 11-12 CAN Bus Timing Register (CTIM) 11-33
Table 11-13 CAN Bus Control Register (CBUS) 11-33
Table 11-14 Output Drivers 11-34
Table 11-15 Transmit Control/Status 11-35
Table 11-16 Node Status 11-35
Table 11-17 Receive/Transmit Status (RTSTAT) 11-37
Table 11-18 Transmit Error Counter Register (TEC) 11-40
Table 11-19 Receive Error Counter Register (REC) 11-40
Table 12-1 Instructions Using A and C 12-64
Table 12-2 Transfer of Control Instructions 12-64
Table 12-3 Memory Transfer Instructions 12-65
Table 12-4 Arithmetic and Logic Instructions 12-65

Table 12-5 Opcodes 12-66
Table 13-1 COP884BC Pinouts 13-3
Table 13-2 PWMCON, PWM Timer Control Register (Address 00A2) 13-6
Table 13-3 CTIM, CAN Bus Timing Register (Address 00B9) 13-6
Table 13-4 CBUS, CAN Bus Control Register (Address 00BA) 13-6
Table 13-5 TCNTL, CAN Bus Transmit Control/Status Register (Address 00BB) 13-7
Table 13-6 RTSTAT, CAN Bus Receive/Transmit Status Register (Address 00BC) 13-7
Table 13-7 CMPSL, Comparator Control Register (Address 00D3) 13-8
Table 13-8 ICNTRL, Interrupt Control Register (Address 00E8) 13-8
Table 13-9 CNTRL, Control Register (Address 00EE) 13-9
Table 13-10 PSW, Processor Status Word Register (Address 00EF) 13-9
Table 13-11 Data Memory Map 13-10
Table 13-12 COP884BC Interrupt Rank and Vector Addresses 13-14
Table 13-13 Comparator Control Register (CMPSL) (00D3) 13-16
Table 13-14 PWM Control register Bits 13-24
Table 14-1 COP888CS/CG/EG/GG/HG/KG Pinouts 14-5
Table 14-2 T3CNTRL, Timer T3 Control Register (Address xxB6)* 14-8
Table 14-3 CMPSL, Comparator Select Register (Address xxB7) 14-9
Table 14-4 ENU, UART Control and Status Register (Address xxBA) 14-9
Table 14-5 ENUR, UART Receive Control and Status Register (Addres xxBB) . 14-10
Table 14-6 ENUI, UART Interrupt and Clock Source Register (Address xxBC) 14-10
Table 14-7 BAUD, UART Baud Register (Address xxBD) 14-10
Table 14-8 PSR, UART Prescaler Select Register (Address xxBE) 14-11
Table 14-9 T2CNTRL, Timer T2 Control Register (Address xxC6) 14-11
Table 14-10 WDSVR, Watchdog Service Register (Address xxC7) 14-11
Table 14-11 ICNTRL, Interrupt Control Register (Address xxE8) 14-12
Table 14-12 CNTRL, Control Register (Address xxEE) 14-12
Table 14-13 PSW, Processor Status Word Register (Address xxEF) 14-12
Table 14-14 COP888CS/CG/EG/GG/HG/KG Data Memory Map 14-13
Table 14-15 COP888CS/CG/EG/GG/HG/KG Interrupt Rank and Vector Addresses 14-18
Table 14-16 Port I Alternate Functions 14-18
Table 14-17 CMPSL, Comparator Select Register (Address xxB7) 14-19
Table 15-1 COP888CF Pinouts 15-4
Table 15-2 T2CNTRL, Timer T2 Control Register (Address xxC6) 15-7
Table 15-3 WDSVR, WATCHDOG Service Register (Address xxC7) 15-8
Table 15-4 ENAD, A/D Converter Control Register (Address xxCB) 15-8
Table 15-5 ICNTRL, Interrupt Control Register (Address xxE8) 15-8
Table 15-6 CNTRL, Control Register (Address xxEE) 15-9
Table 15-7 PSW, Processor Status Word Register (Address xxEF) 15-9
Table 15-8 COP888CF Data Memory Map 15-9
Table 15-9 COP888CF Interrupt Rank and Vector Addresses 15-13
Table 15-10 ENAD, A/D Converter Control Register (Address xxCB) 15-16
Table 15-11 A/D Prescaler Options 15-17
Table 15-12 A/D Channel Selection 15-18
Table 16-1 COP888CL Pinouts 16-4
Table 16-2 T2CNTRL, Timer T2 Control Register (Address xxC6) 16-7
Table 16-3 WDSVR, WATCHDOG Service Register (Address xxC7) 16-8

Table 16-4 ICNTRL, Interrupt Control Register (Address xxE8) 16-8
Table 16-5 CNTRL, Control Register (Address xxEE) 16-8
Table 16-6 PSW, Processor Status Word Register (Address xxEF) 16-9
Table 16-7 COP888CL Data Memory Map 16-9
Table 16-8 COP888CL Interrupt Rank and Vector Addresses 16-13
Table 17-1 COP888EB Pinouts 17-4
Table 17-2 SPICNTL, SPI Control Register (Address xx98) 17-9
Table 17-3 SPISTAT, SPI Status Register (Address xx99) 17-9
Table 17-4 CTIM, CAN Bus Timing Register (Address xxA9) 17-9
Table 17-5 CBUS, CAN Bus Control Register (Address xxAA) 17-10
Table 17-6 TCNTL, CAN Bus Transmit Control/Status Register (Address xxAB) 17-10
Table 17-7 RTSTAT, CAN Bus Receive/Transmit Status Register (Address xxAC) 17-11
Table 17-8 ENU, UART Control and Status Register (Address xxBA) 17-11
Table 17-9 ENUR, UART Receive Control and Status Register (Address xxBB) 17-12
Table 17-10 ENUI, UART Interrupt and Clock Source Register (Address xxBC) 17-12
Table 17-11 BAUD, UART Baud Register (Address xxBD) 17-12
Table 17-12 PSR, UART Prescaler Select Register (Address xxBE) 17-13
Table 17-13 T2CNTRL, Timer T2 Control Register (Address xxC6) 17-13
Table 17-14 WDSVR, WATCHDOG Service Register (Address xxC7) 17-13
Table 17-15 ENAD, A/D Converter Control Register (Address xxCB) 17-14
Table 17-16 ITMR, IDLE Timer Control Register (Address xxCF) 17-14
Table 17-17 ICNTRL, Interrupt Control Register (Address xxE8) 17-14
Table 17-18 CNTRL, Control Register (Address xxEE) 17-15
Table 17-19 PSW, Processor Status Word Register (Address xxEF) 17-15
Table 17-20 COP888EB Data Memory Map 17-16
Table 17-21 COP888EB Interrupt Rank and Vector Addresses 17-22
Table 17-22 SPI Control (SPICNTL) register 17-26
Table 17-23 SPI Status (SPISTAT) register 17-27
Table 18-1 COP888EK Pinouts 18-4
Table 18-2 T2CNTRL, Timer T2 Control Register (Address xxC6) 18-7
Table 18-3 T3CNTRL, Timer T3 Control Register (Address xxB6) 18-8
Table 18-4 WDSVR, Watchdog Service Register (Address xxC7) 18-8
Table 18-5 ICNTRL, Interrupt Control Register (Address xxE8) 18-8
Table 18-6 CNTRL, Control Register (Address xxEE) 18-9
Table 18-7 PSW, Processor Status Word Register (Address xxEF) 18-9
Table 18-8 CMPSL, Comparator Register (Address xxB7) 18-10
Table 18-9 Comparator Input Selection 18-12
Table 18-10 Analog Function Block Electrical Characteristics (Vcc=5V) 18-13
Table 18-11 Data Memory Map 18-14
Table 18-12 COP888EK Interrupt Rank and Vector Address 18-17
Table 19-1 COP888FH Pinouts 19-4
Table 19-2 Hardware Multipler/Divider Control Register, MDCR (Address xx9D) 19-7
Table 19-3 T3CNTRL, Timer T3 Control Register (Address xxB6)* 19-8
Table 19-4 CMPSL, Comparator Select Register (Address xxB7) 19-8
Table 19-5 ENU, UART Control and Status Register (Address xxBA) 19-9
Table 19-6 ENUR, UART Receive Control and Status Register (Address xxBB). 19-9

Table 19-7 ENUI, UART Interrupt and Clock Source Register (Address xxBC) 19-10
Table 19-8 BAUD, UART Baud Register (Address xxBD) 19-10
Table 19-9 PSR, UART Prescaler Select Register (Address xxBE) 19-10
Table 19-10 T2CNTRL, Timer T2 Control Register (Address xxC6) 19-11
Table 19-11 WDSVR, WATCHDOG Service Register (Address xxC7) 19-11
Table 19-12 ICNTRL, Interrupt Control Register (Address xxE8) 19-11
Table 19-13 CNTRL, Control Register (Address xxEE) 19-12
Table 19-14 PSW, Processor Status Word Register (Address xxEF) 19-12
Table 19-15 COP888FH Data Memory Map 19-12
Table 19-16 COP888FH Interrupt Rank and Vector Addresses 19-17
Table 19-17 Port I Alternate Functions 19-18
Table 19-18 CMPSL, Comparator Select Register (Address xxB7) 19-19
Table 19-19 Multiply/Divide Data Registers. 19-20
Table 19-20 Multiply/Divide Control Register 19-21
Table 20-1 COP888GW Pinouts. 20-3
Table 20-2 Hardware Multipler/Divider Control Register, MDCR (Address xx9D) 20-8
Table 20-3 Counter Control Register 1, CCR1 (Address xx9E) 20-8
Table 20-4 Counter Control Register 2, CCR2 (Address xx9F) 20-9
Table 20-5 Capture Timer 1 Control Register, CCMR1 (Address xxB6) 20-9
Table 20-6 Capture Timer 2 Control Register, CCMR2 (Address xxB7) 20-10
Table 20-7 ENU, UART Control and Status Register (Address xxBA) 20-10
Table 20-8 ENUR, UART Receive Control and Status Register (Address xxBB) 20-11
Table 20-9 ENUI, UART Interrupt and Clock Source Register (Address xxBC) 20-11
Table 20-10 BAUD, UART Baud Register (Address xxBD) 20-12
Table 20-11 PSR, UART Prescaler Select Register (Address xxBE) 20-12
Table 20-12 T2CNTRL, Timer T2 Control Register (Address xxC6) 20-12
Table 20-13 ICNTRL, Interrupt Control Register (Address xxE8) 20-13
Table 20-14 CNTRL, Control Register (Address xxEE) 20-13
Table 20-15 PSW, Processor Status Word Register (Address xxEF) 20-13
Table 20-16 COP888GW Data Memory Map 20-14
Table 20-17 COP888GW Interrupt Rank and Vector Addresses 20-20
Table 20-18 Multiply/Divide Data Registers. 20-21
Table 20-19 Multiply/Divide Control Register 20-21
Table 20-20 Capture Timer Registers 20-24
Table 20-21 CCMR1 [xxB6] Register Bits 20-24
Table 20-22 CCMR2 [xxB7] Register Bits 20-25
Table 20-23 Pulse Train Generator Registers. 20-29
Table 20-24 CCR1 Register Bits (Address xx9E Hex) 20-30
Table 20-25 CCR2 Register Bits (Address xx9F Hex) 20-31
Table A-1 Electric Field Calculation Results. A-34

Chapter 1

OVERVIEW

1.1 INTRODUCTION

The COP888 Feature Family 8-bit microcontrollers provide high-performance, low-cost solutions for embedded control applications. COP888 devices are fabricated with National Semiconductor's M^2CMOS™ technology for low current drain and a wide operating voltage range. Most instructions are single-byte and have an execution time of one instruction cycle, allowing high throughput. Multiple addressing modes and a rich instruction set further enhance throughput efficiency and reduce program size. Other COP888 features such as reconfigurable inputs and outputs, multi-mode general-purpose timers, and the MICROWIRE/PLUS™ serial interface provide the flexibility needed to construct single-chip solutions for a wide variety of applications.

All COP888 Feature Family members share the set of features listed in Section 1.2. Many individual family members also contain other features such as additional timers, an A-to-D converter, or a UART. The device-specific features are described in Section 1.3.

1.2 BASIC FEATURES

Each member of the COP888 family of microcontrollers offers the following features:

- 8-bit core processor.
- CMOS technology for low power, fully static operation.
- HALT mode for very low standby power.
- Memory mapped architecture. All RAM, I/O ports, and registers (except A and PC) are mapped into the data memory address space.
- On-chip data memory and program memory.
- Flexible, reconfigurable I/O.
- MICROWIRE/PLUS serial interface, a 3-wire serial data communication system that allows the microcontroller to be programmed for either master or slave mode operation.
- Extremely versatile 16-bit timer with two associated autoload/capture registers, which can operate in any of three modes: PWM (Pulse Width Modulation), external event counter, or input capture register.
- Non-maskable Software Trap interrupt.
- Maskable interrupts (number and type depending on family member).
- Two 8-bit "Register Indirect" data memory pointers.

- 8-bit Stack Pointer (SP) for stack in data memory RAM.
- Choice of clock types: crystal oscillator, or R/C oscillator.

1.3 DEVICE-SPECIFIC FEATURES

In addition to the core features, non-core features are provided by specific COP888 devices. These features are:

- 8-bit Data Segment Address Register (S Register), used for addressing data memory beyond the first 128 bytes of RAM
- One or more additional 16-bit general-purpose timers
- IDLE mode for very low standby power while maintaining real time with associated IDLE Timer
- Multi-Input Wakeup/Interrupt feature, which provides additional inputs for interrupts or to exit the HALT or IDLE mode
- Watchdog and clock monitor
- NMI non-maskable interrupt
- Full-duplex, double-buffered UART (Universal Asynchronous Receiver/Transmitter) for serial communication
- Analog-to-Digital (A/D) Converter with eight single-ended or four differential-pair channels
- Controller Area Network (CAN) interface, a communications network standard often used in automotive and industrial control applications.
- Serial Peripheral Interface (SPI), a synchronous bidirectional serial communications interface used to transfer data between modules in a system.
- Hardware multiplier/divider that supports 1-byte by 2-byte multiplication in one instruction cycle and 3-byte by 2-byte division in two instruction cycles.
- Reduced electromagnetic radiated emissions using internal power supply filters, a low-current crystal oscillator, and gradual turn-on of output drivers.

Chapter 2

ARCHITECTURE

2.1 INTRODUCTION

The COP888 microcontroller contains all program memory and data memory internally. In addition, it contains on-chip configurable I/Os, an on-chip timer, and a built-in MICROWIRE/PLUS interface. The presence of on-chip memory and peripherals allows the COP888 microcontroller to provide a single-chip solution for many applications.

The COP888 memory organization is based on the “Harvard” architecture, in which the program memory is distinct from the data memory. Each of these two types of memory has its own physical memory space, and uses its own internal address bus. The advantage of this type of organization is that accesses to program memory and data memory can take place concurrently, reducing overall execution time. By contrast, in the “Von Neumann” architecture, program memory and data memory share the same address bus, and concurrent accesses cannot occur.

Except for the Accumulator (A) and Program Counter (PC), all registers, I/O ports, and RAM are memory mapped in the data memory address space. Among these registers are the B Register, X Register, Stack Pointer (SP), and I/O port registers. All such registers can be accessed by reading or writing their memory addresses.

The COP888 architecture provides one enhancement to the Harvard architecture: an instruction called Load Accumulator Indirect (LAID), which allows access to data tables stored in program memory. A conventional Harvard architecture does not allow this.

The COP888 device communicates with other devices through several configurable I/O ports or through the MICROWIRE/PLUS serial I/O interface. The I/O ports are designated by letter names, such as Port C, Port D, Port G, Port I, and Port L.

A 16-bit general-purpose timer is provided in all COP888 microcontrollers, together with two associated 16-bit autoload/capture registers. The timer can be configured to operate in any of three modes: Pulse Width Modulation (PWM), external event counter, or input capture mode.

A maximum of sixteen different interrupts are available in the COP888: two non-maskable interrupts and 14 maskable interrupts. All interrupts cause a branch to the same location in program memory. A special instruction (VIS) may be placed at this location to force an automatic branch to the highest priority-interrupt service routine.

2.2 BLOCK DIAGRAM

A block diagram of the COP888 Family architecture is shown in Figure 2-1. All COP888 Family devices contain the elements pictured in the block diagram. These elements include: the Arithmetic Logic Unit (ALU), Data Memory, Program Memory, Timer 1, MICROWIRE/PLUS, Port I/Os, and Interrupt Logic. Functional blocks not common to all

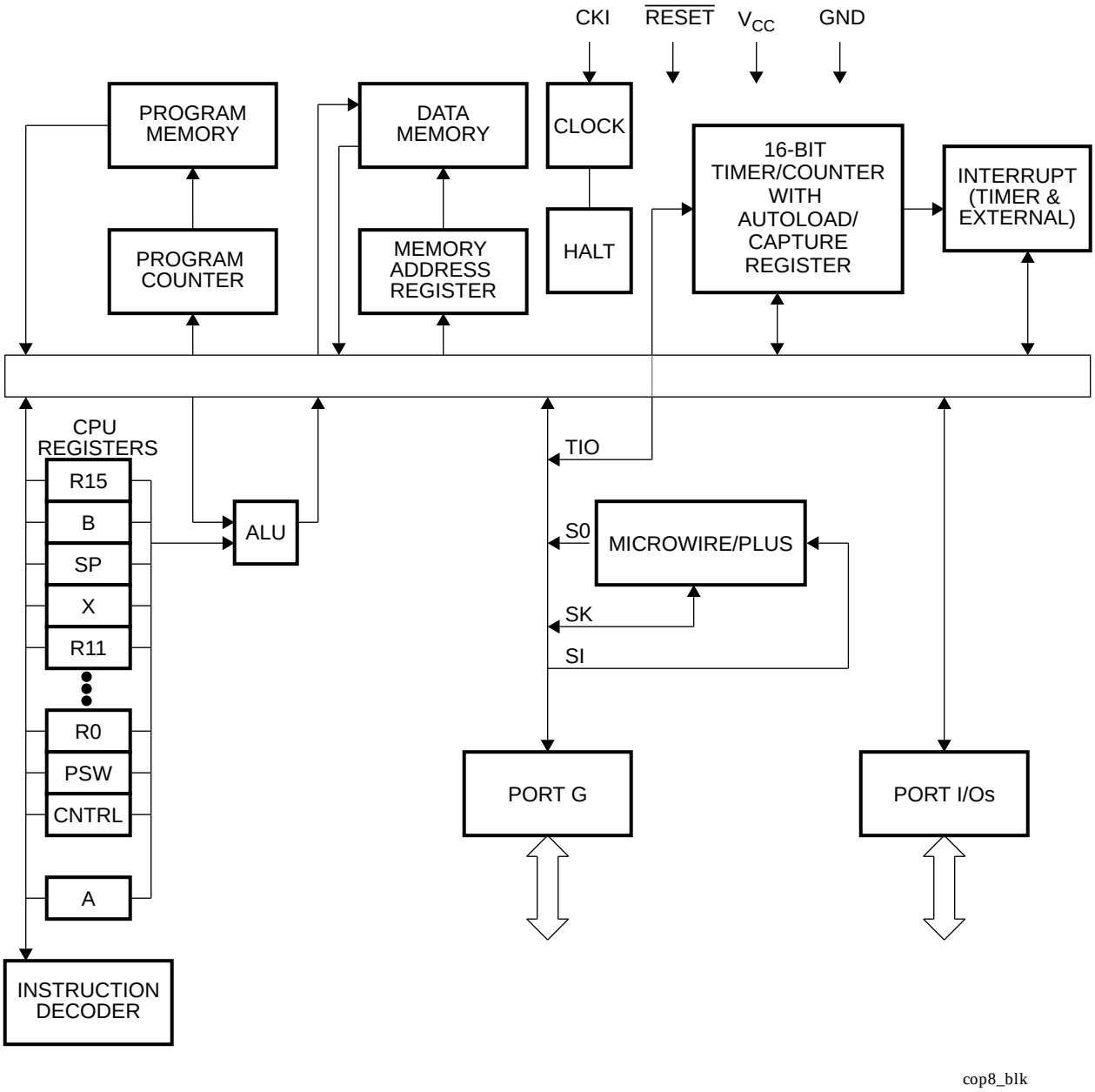


Figure 2-1 COP888 Block Diagram

COP888 Family members are not shown in Figure 2-1. Block diagrams of individual devices are shown in the device-specific chapters of this manual.

2.3 MEMORY ORGANIZATION

The COP888 microcontrollers are based on a modified Harvard-style architecture. This type of architecture separates the control program memory from the data memory. Each memory type has its own address space, address bus and data bus. The following sections describe the memory structure.

2.3.1 Program Memory

The COP888 program memory is a block of byte-wide non-volatile ROM or EPROM memory. The program memory addressing range is 32 Kbytes. A 15-bit Program Counter (PC) is used to address the program memory, which is subdivided into 4-Kbyte segments with respect to certain instructions. The program memory may hold program instructions or constant data.

The 4-Kbyte segment divisions within the program memory affect the economical 2-byte Jump Absolute (JMP) and Jump Subroutine (JSR) instructions. These instructions cause the lower 12-bits of the PC to be replaced by the value specified in the instruction while the upper three bits remain unchanged. Thus, these instructions branch only within the currently addressed 4-Kbyte program memory segment.

The indirect instructions, Jump Indirect (JID) and Load Accumulator Indirect (LAID), operate only within a program memory block of 256 bytes. This restriction exists because only the lower eight bits of the PC (PCL) are replaced during program memory table lookups. The upper seven bits of the PC (PCU) remain unchanged. Replacing only the PCL minimizes the execution time of this instruction. Programmers must ensure that LAID/JID instructions and their associated tables do not cross the 256-byte program memory boundaries.

The very economical Jump Relative Short (JP) instruction is completely independent of all program memory block and memory segment boundaries. This single-byte JP instruction allows a branch forward of up to 32 locations or backwards of up to 31 locations relative to the current contents of the program counter. A branch forward of 1 is not allowed, since this may be implemented with a NOP.

2.3.2 Data Memory

All COP888 family members have read/write data memory. Some sections of the data memory space are reserved for the CPU registers, I/O registers, and control registers, all of which are memory mapped. Other sections of the data memory contain RAM and/or EEPROM, which can be used by the application program. The amount of available RAM or EEPROM memory varies from one family member to another. For information on the quantity and type of data memory, see the device-specific chapters later in this manual.

Data memory can be accessed either directly by an address specified in the instruction, or indirectly using the X, SP, or B pointer registers. In all cases, the data memory location is specified as a single byte. Thus, one of 256 memory locations is specified for a data memory access. In the most basic COP888 devices, only the first 256 bytes of data memory are used. In other COP888 devices, a data segment extension register (S register) extends the data memory address range to 32 Kbytes. Data memory extension using the S register is explained in the next section of this manual, Data Segment Extension.

In basic devices that do not use data segment extension, there is a single 256-byte segment of data memory, divided into smaller segments as shown in Figure 2-2. There are 128 bytes of RAM, occupying two non-contiguous address spaces: 112 bytes of lower memory, from 00 to 6F Hex, and 16 bytes at the top of the memory, from F0 to FF Hex. Most of the remaining upper part of this memory, from B0 through EF Hex, is used for

the I/O and control registers required by the timers, ports, MICROWIRE interface, CPU core, and optional COP888 peripherals (UART, comparator, etc.). The remaining parts of the 256-byte memory segment (70-7F Hex) are not used.

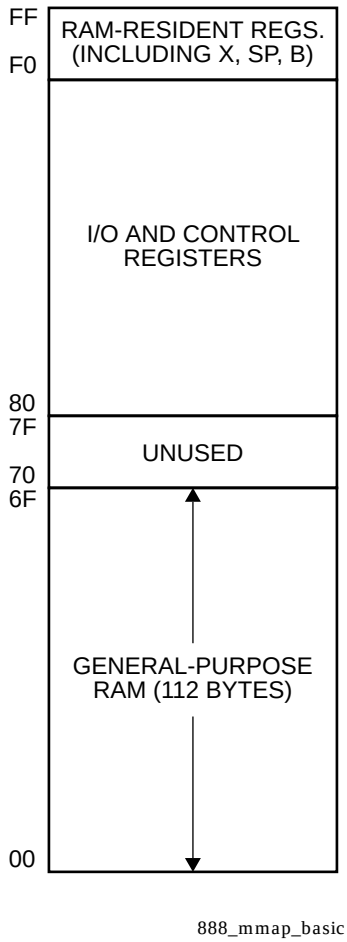


Figure 2-2 A Basic Memory Map

The lower, 112-byte segment of RAM is general-purpose read/write memory that is available to the application program. Upon reset, the stack pointer is initialized to the top of this segment (6F Hex), and the stack grows downward from that address as items are pushed onto the stack. Memory from 00 Hex up to the stack can be used for any purpose by the application program. The first 16 addresses (00-0F Hex) have special significance when used with certain instructions (such as LD B,#), because then the instructions are single byte and take only one instruction cycle to execute, rather than two bytes and two instruction cycles.

The upper, 16-byte segment of RAM at the top of memory is used for the RAM-resident registers. The X, SP, and B pointer registers are mapped into memory locations FC, FD, and FE Hex, respectively. The S register, if used, is mapped into address FF Hex. The remaining 12 register locations are available to the application program for any purpose. Certain COP888 instructions (such as DRSZ) work only with this 16-byte segment of memory. Certain other instructions (such as LD MD,#) are more efficient when used with this 16-byte segment than with other RAM memory locations.

There is at least one segment of unused data: 16 bytes from 70 to 7F Hex. Reading from this unused segment returns FF Hex. Reading from other unused segments returns unknown data.

All RAM, I/O ports, and registers (except A and PC) are mapped into the data memory address space. Table 2-1 shows a basic Register Memory Map for all COP888 devices. Refer to the device-specific chapters for complete memory maps of individual devices.

Table 2-1 Data Memory Map

Address	Contents
00–6F	On-chip RAM Address Space
70–BF	On-chip Data Memory Address Space
C0–CF	I/O and Register Address Space
D0	Port L Data Register
D1	Port L Configuration Register
D2	Port L Input Pins (read only)
D3	Reserved for Port L
D4	Port G Data Register
D5	Port G Configuration Register
D6	Port G Input Pins (read only)
D7	Port I Input Pins (read only)
D8	Port C Data Register
D9	Port C Configuration Register
DA	Port C Input Pins (read only)
DB	Reserved for Port C
DC	Port D Data Register
DD-DF	Reserved for Port D
E0–E5	Reserved
E6	Timer T1 Autoload Register T1RB Lower Byte
E7	Timer T1 Autoload Reigster T1RB Upper Byte
E8	1CNTRL Register
E9	MICROWIRE/PLUS Shift Register
EA	Timer T1 Lower Byte
EB	Timer T1 Upper Byte
EC	Timer T1 Autoload Register T1RA Lower Byte
ED	Timer T1 Autoload Register T1RA Upper Byte
EE	CNTRL Control Register
EF	PSW Register
F0–FB	On-chip RAM mapped as Registers
FC	X Register
FD	SP Register
FE	B Register
FF	S Register

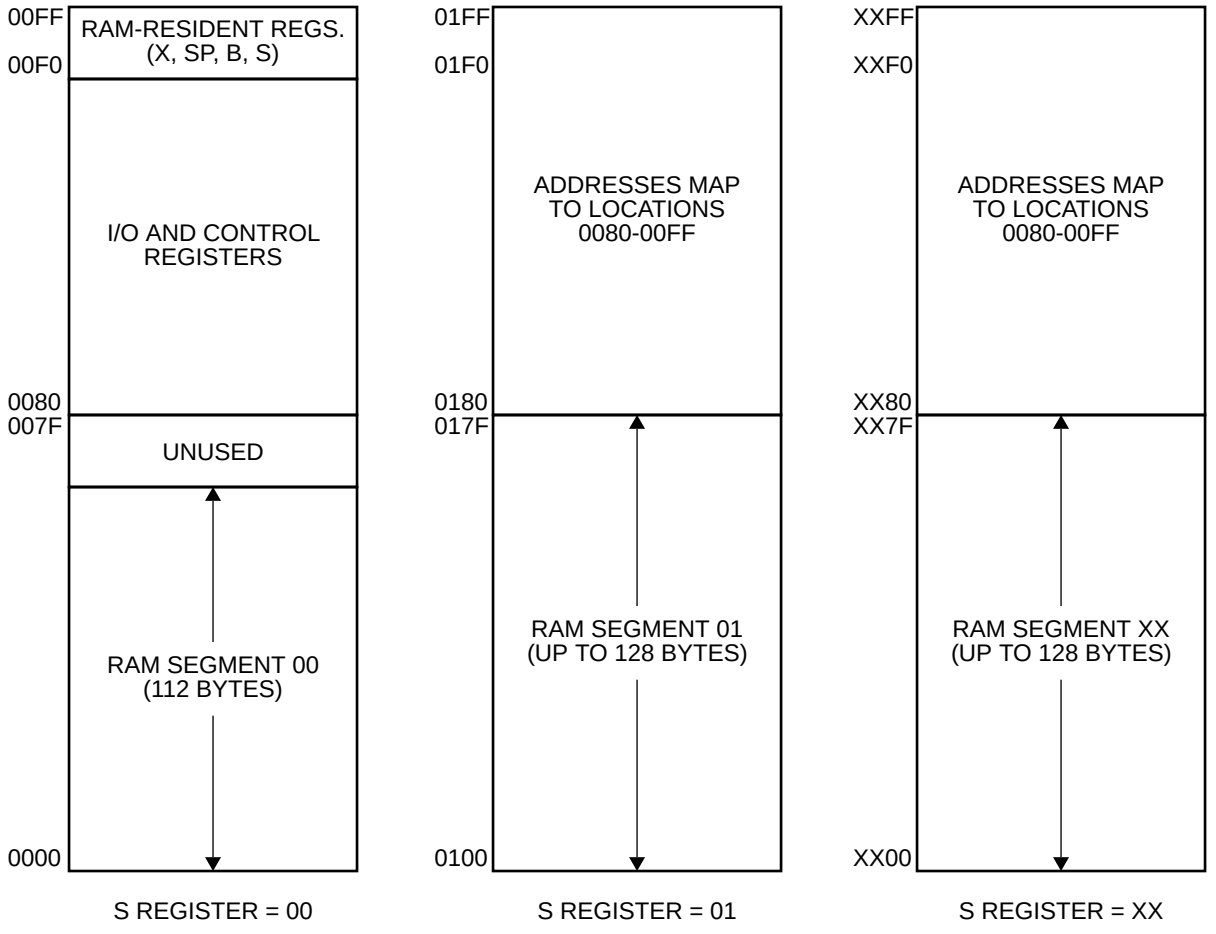
Data Segment Extension

In the most basic COP888 devices, there are 128 bytes of RAM residing in two segments within the 256-byte data memory, as described in the previous section. In COP888 devices having more than 128 bytes of RAM, some of the RAM occupies memory above the first 256 addresses (above address FF Hex).

To allow the program to access the RAM residing above this address, the data segment extension register, or "S register," is used. A 16-bit address is made by combining the 8-bit S register with the normal 8-bit memory address specified by the program instruction. The S register is the high-order byte, and the normal 8-bit memory address is the low-order byte. The S register can be accessed via address FF Hex, and can be read or written like any other register.

The data segment extension feature only works for the lower 128 bytes of each 256-byte memory segment. Therefore, only the lower 128 bytes are used for general-purpose RAM in each 256-byte memory page. The upper 128 bytes are occupied by the same set of registers in all 256-byte memory pages. This concept is more easily understood by looking at the memory map for devices with data segment extension, shown in Figure 2-3.

Upon reset, the S register is cleared to zero, and the memory map is the same as for a device without the data segment extension feature. However, if the value 01 is written to



888_mmap_data_seg

Figure 2-3 Memory Map with Data Segment Extension

the S register, a data memory access instruction specifying an address between 00 and 7F Hex will access the RAM residing at 0100-017F Hex, also called RAM Segment 01. Similarly, if the value 02 is written to the S register, a data memory access between 00 and 7F Hex will access RAM at 0200-027F Hex (RAM Segment 02), and so on. An access to the upper half of any 256-byte memory segment (from 80 to FF Hex), regardless of the S register contents, will always access the same set of registers as a device that does not use data segment extension.

Additional memory beyond the minimum of 128 bytes, if available, resides in the additional RAM segments, starting with Segment 01. For example, the COP888CG has 192 bytes of RAM, or 50% more than the minimum of 128 bytes. The additional 64 bytes of RAM reside in the address range of 0100-013F Hex, or the bottom half of RAM Segment 01. The COP888EG, which has 256 bytes of RAM, has an additional 128 bytes residing in the address range of 0100-017F Hex, thus filling all of RAM Segment 01. A COP888 device having more than 256 bytes of RAM begins to fill Segment 02, an so on, up to the theoretical limit of 32K bytes in 256 segments. Note that each additional 128 bytes of memory fill a contiguous 128-byte segment, unlike the first 128 bytes of RAM in Segment 00.

The software can write to the S register (address xxFF Hex) at any time to change from one memory address segment to another. All addressing modes are available, no matter which segment is chosen. Note that the S register does not need to be changed in order to access registers residing between 80 and FF Hex, because the S register is ignored in that range. Also, note that the Stack Pointer (SP) always points to memory in Segment 00 (starting with address 6F upon reset), regardless of what is contained in the S register. Therefore, the stack must always be stored in Segment 00.

2.3.3 Memory-Mapped I/O Registers

The COP888 devices have three different types of ports: reconfigurable input/output, dedicated output, and dedicated input. Every I/O port has specific memory mapped I/O registers and/or addresses associated with it, depending on the port type. The following sections describe the I/O port register structure for each port type.

NOTE: All port registers and pins are memory mapped in the data memory address space. Therefore, instructions which operate on data memory also operate on port registers and pins. This includes instructions used to set, reset and test individual bits. The I/O register addresses for specific ports are listed in the memory map shown in Table 2-1.

Reconfigurable Input/Outputs

Reconfigurable input/output ports have two associated port registers: a port configuration register and a port data register. These two memory mapped registers allow the port pins to be individually configured as either inputs or outputs, and to be individually changed back and forth in software.

The configuration register is used to configure the pins as inputs or outputs. A pin may be configured as an input by writing a 0 or as an output by writing a 1 to its associated configuration register bit. If a pin is configured as an output, the associated data register bit represents the state of the pin (1 = logic high, 0 = logic low). If the pin is configured as

an input, the associated data register bit selects whether the pin is a weak pull-up or Hi-Z input. Table 2-2 shows the port configuration options. The port configuration and data registers are all read/write registers.

A third data memory address is assigned to each I/O port. Reading this memory address returns the value of the port pins regardless of how the pins are configured.

Table 2-2 I/O Port Configuration

Configuration Bit	Data Bit	Port Pin Setup
0	0	Hi-Z input (TRI-STATE output)
0	1	Input with pull-up (weak one output)
1	0	Push-pull zero output
1	1	Push-pull one output

Dedicated Outputs

Dedicated output ports have one associated port register. This memory mapped output data register is used to set the port pins to a logic high or low. A port pin may be individually configured logic high or low by writing a one or zero, respectively, to its associated data register bit. Port data registers may be read or written.

Dedicated Inputs

Dedicated input ports have no associated port registers. However, a data memory address is assigned to the port pins for reading of the port input. Port pin addresses are read-only memory locations.

2.4 CORE REGISTERS

All COP888 microcontrollers share a common block of logic referred to as the COP888 Core. This core includes the COP888 Central Processing Unit (CPU), the Timer 1 Block, the MICROWIRE/PLUS block, and the Interrupt Block. The registers contained within these blocks are the core registers. The registers include: a 15-bit program counter (PC), an 8-bit accumulator (A), a processor status word (PSW), two core control registers (CNTRL, ICNTRL), sixteen 8-bit data memory registers, one 16-bit timer, two 16-bit autoload capture registers, and one 8-bit shift register. All core registers are memory mapped into the data memory address space except for the program counter (PC) and accumulator. The following sections describe in detail the COP888 core registers.

2.4.1 Accumulator

All COP888 family parts have a single 8-bit accumulator. The accumulator is used in all arithmetic and logical operations, such as ADD and XOR. In addition, it is used with the exchange, JID and LAID instructions. The arithmetic and logical instructions use the

accumulator as both an operand and result register. A second operand register, if required, is either the instruction register (IR), which contains immediate data or a register in data memory.

2.4.2 Program Counter

The CPU contains a 15-bit program counter used in addressing the byte-wide program memory. The PC is initialized to zero at reset and is incremented once for each byte of an instruction opcode. Jumps, jump subroutines, interrupts, and the JID instruction cause some or all of the PC bits to be replaced. Transfer-of-control instructions that replace only some of the PC bits have a limited jumping range.

2.4.3 Control Registers

The COP888 core contains three 8-bit control registers (PSW, CNTRL and ICNTRL). The following paragraphs and tables show the bits contained in each register. The functions of these bits are described in later chapters.

PSW Register (Address xxEF Hex)

The Processor Status Word (PSW) register contains eight different flag bits. The PSW register bits are assigned as follows:

GIE	Global Interrupt Enable
EXEN	External Interrupt Enable
BUSY	MICROWIRE busy shifting flag
EXPND	External Interrupt Pending
T1ENA	Timer T1A Interrupt Enable
T1PNDA	Timer T1A Interrupt Pending
C	Carry Flag
HC	Half-Carry Flag

Table 2-3 PSW Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

CNTRL Register (Address xxEE Hex)

The control register (CNTRL) contains the MICROWIRE/PLUS, External interrupt, and Timer 1 control flags. The CNTRL register bits are assigned as follows:

SL1 & SL0	Select the MICROWIRE clock divide-by (00=2,01=4,1x=8)
IEDG	External interrupt edge polarity (0 = rising edge, 1 = falling edge)
MSEL	Selects G5 and G4 as MICROWIRE signals SK and SO, respectively
T1C0	Timer T1 Start/Stop Control in Timer Modes 1 and 2
T1C1	Timer T1 Mode Control Bit

T1C2 Timer T1 Mode Control Bit
T1C3 Timer T1 Mode Control Bit

Table 2-4 CNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

ICNTRL Register (Address xxE8 Hex)

The interrupt control register (ICNTRL) contains the MICROWIRE/PLUS Interrupt enable and pending flags, the Timer 1 Source B Interrupt enable and pending flags, the general purpose Peripheral Interrupt enable and pending flags, and two spare bits. In most COP888 family members, the Peripheral Interrupt enable and pending flags are used for the Idle Timer interrupt enable and pending flags, T0EN and T0PND, respectively. If a family member does not have an Idle Timer, these flags are used for other purposes. In COP888 devices that support the Multi-Input Wakeup feature on Port L, one of the spare flags (FLAG2) is used for the Port L Interrupt Enable flag. Many COP888 devices do not use the second spare flag (FLAG1). In these devices, FLAG1 is reserved for future use and should not be set by the user program. Refer to the device specific chapters for information on the assignment of the ICNTRL bits for individual COP888 devices. The ICNTRL register bits are assigned as follows:

T1ENB Timer T1B Interrupt Enable
T1PNDB Timer T1B Interrupt Pending
uWEN MICROWIRE/PLUS Interrupt Enable
uWPND MICROWIRE/PLUS Interrupt Pending
PEN General Purpose Peripheral Interrupt Enable (frequently used for T0EN; Idle Timer interrupt enable)
PPND General Purpose Peripheral Interrupt Pending (frequently used for T0PND; Idle Timer interrupt pending)
FLAG2 General Purpose Flag (frequently used for LPEN; Enable Port L interrupts)
FLAG1 General Purpose Flag (reserved)

Table 2-5 ICNTRL Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FLAG1	FLAG2 (LPEN)	PPND (T0PND)	PPEN (T0EN)	uWPND	uWEN	T1PNDB	T1ENB

2.4.4 Data Registers

The COP888 contains sixteen 8-bit data registers, located in data memory from address 00F0 to 00FF Hex. Four of these registers, 00FC through 00FF Hex, have special functions. Locations 00FC and 00FE Hex contain the 8-bit data memory pointers X and B respectively. Location 00FD contains the 8-bit stack pointer (SP) for data memory. Location 00FF is reserved for the data segment extension register. This register is used

in some COP888 devices to extend data memory beyond 128 bytes. In devices which contain 128 or fewer bytes of data memory, this register is reserved. The remaining twelve registers, 00F0 through 00FB, are always available for general-purpose use.

Certain COP888 instructions differentiate data registers from other data memory locations, such as the DRSZ (decrement register skip if zero) instruction, DRSZ subtracts one from a specified data register and skips the following instruction if the result of the decrement is zero. This instruction is extremely useful in constructing code loops, and makes the data registers ideal choices for loop counters. Other instructions like the "load memory with immediate data" are more efficient when used with the register memory than when used with the general data memory.

Stack Pointer

The stack pointer (SP) is memory mapped at data memory location 00FD Hex. The stack pointer is automatically initialized during Reset to point to location 06F Hex.

Pushing addresses onto the stack causes the stack to grow downward in data memory toward address zero. Popping addresses off the stack causes the stack to shrink upward. If the stack pointer is initialized to the top of the base segment of memory (06F Hex), over-popping the stack causes a Software Trap error interrupt. The lower limit of the stack is address 0000 Hex. Over-pushing the stack causes the stack to wrap around to addresses 00FF and 00FE Hex (subroutine calls and interrupts cause a double-byte push). This should be avoided because it interferes with the B pointer, which is memory mapped at location 00FE Hex.

The user program may initialize the stack pointer anywhere in the base segment of memory. The stack still grows down toward address zero, but the stack no longer has the Software Trap interrupt over-pop protection. Initializing the stack pointer to one of the upper base segment data register addresses (00F0 to 00FB Hex) is potentially very hazardous. The available stack memory is severely limited, and if the stack pushes downward beyond address location 00F0, interference occurs with the PSW and CNTRL control registers, which are memory mapped at address locations 00EF and 00EE Hex, respectively.

Data Memory Pointers (Index Registers)

The COP888 contains two special registers, X and B, which may be used as pointers. These registers allow indirect addressing of all locations mapped in the data memory address space. In addition, these registers may be automatically incremented or decremented by certain instructions that use register indirect addressing. The auto-incrementing and auto-decrementing features allow the user program to easily step through data memory locations (i.e., tables).

Data Segment Extension Register

In COP888 family members that have more than 128 bytes of RAM, the data memory register located at FF Hex is used as the Data Segment Extension Register (S). Refer to Section 2.3.2 for more information on this register.

2.4.5 MICROWIRE/PLUS Register

The MICROWIRE/PLUS three-wire serial communication system contains an 8-bit memory-mapped serial shift register (SIOR). The serial data input and output signals to the SIOR register are supplied by SI and SO, respectively. The shift register is clocked by the signal SK. Data is shifted through the SIOR from the low-order end to the high-order end on the falling edge of the SK clock signal.

2.4.6 Timer Registers

The COP888 core contains one timer block. The timer block consists of a 16-bit timer/ counter with two associated 16-bit autoload/capture registers. The 16-bit registers and timer are each organized as two 8-bit memory mapped registers. The upper and lower byte addresses for the memory mapped timer and autoload/capture registers are shown in the data memory address map (Table 2-1).

2.5 CPU OPERATION

This section describes the operation of the Central Processing Unit (CPU). A brief description of the control logic and Arithmetic Logic Unit is given at the beginning of this section. The remainder of this section describes how the microcontroller performs memory fetches, executes instructions, and handles interrupt/error conditions. A block diagram of the main elements that interface with the control logic and ALU is shown in Figure 2-4.

Control Logic

The control logic handles virtually all operations within the device. It includes the program counter, the memory address register, the processor status word register, and the instruction register for storing information. It also includes logic for directing memory fetches, instruction decoding and execution, and interrupt/error handling. It receives inputs from the ALU and on-chip peripherals including the timer(s) and the MICROWIRE/PLUS interface, and generates control signals for these and other parts of the device.

Arithmetic Logic Unit (ALU)

The ALU performs all logical and arithmetic operations. Inputs to the ALU are provided by the accumulator, several hard-wired data constants, the carry/half-carry bits and the memory data register (MDR). The ALU inputs for a given instruction are specified in the instruction opcode. The accumulator functions as both a source and destination for the ALU, and is used in **all** logical and arithmetic instructions. It always contains the result of the last executed logical, arithmetic, or load/exchange accumulator instruction. The hard-wired data constants, which include 0000, 0001, and 00FF Hex are used in instructions like CLR A, INC A, and DEC A. These instructions have an implicit addressing mode. The carry (C) and half-carry (HC) bits are used in instructions like ADC and SUBC. All arithmetic and logical instructions with two operands use the MDR as one input to the ALU. The MDR may be loaded with operands from data memory or

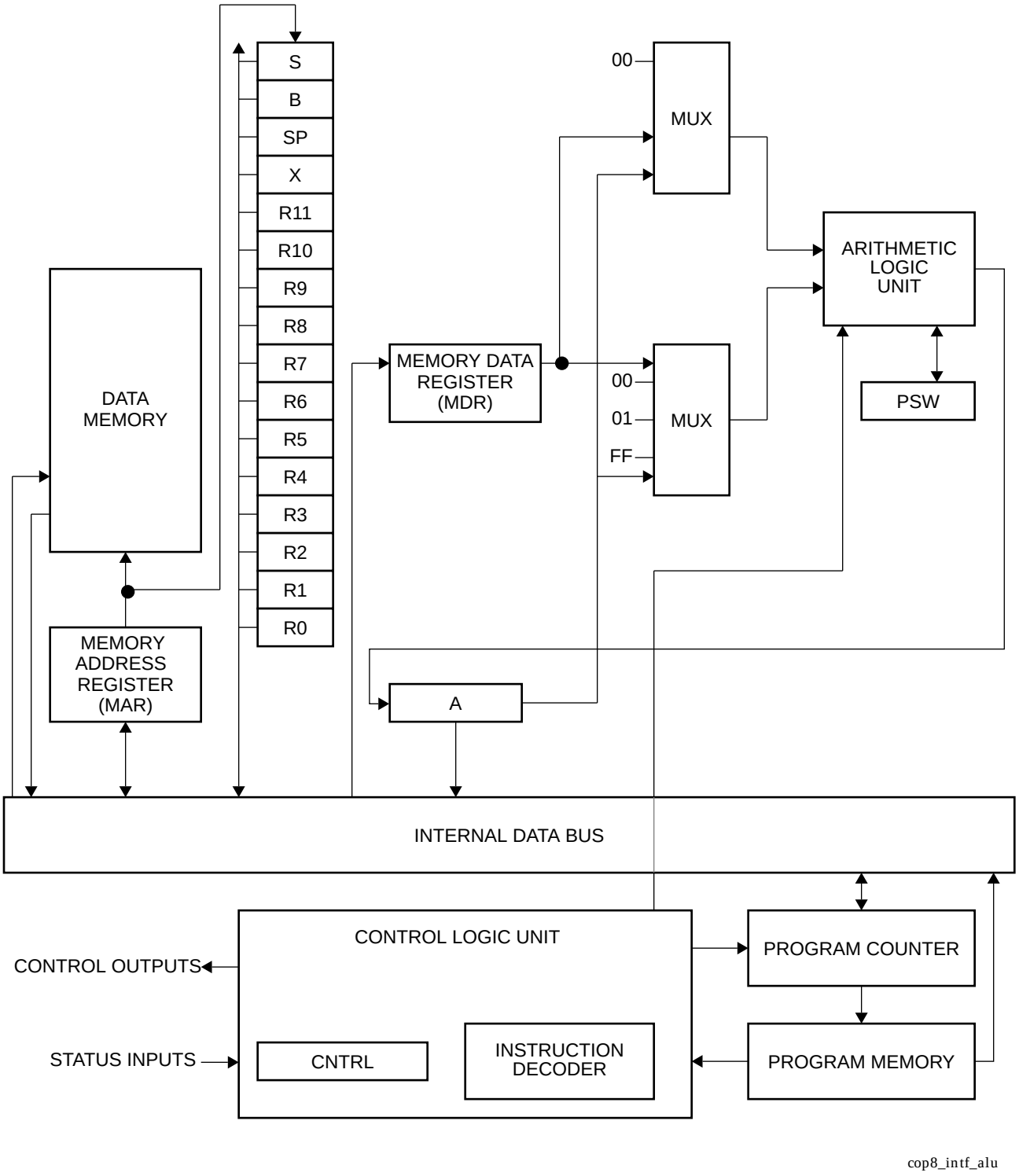


Figure 2-4 Control Logic and ALU Interface

the instruction register (immediate data specified in an instruction opcode). Since only one MDR exists, arithmetic and logical instructions can not be performed directly on two operands from data and/or program memory. Such operations require one operand from memory to be loaded into the accumulator prior to execution.

2.5.1 Memory Fetches

The following two sections describe the manner in which the COP888 accesses data and program memory. Memory access time greatly affects total instruction execution time, and is therefore an important element in understanding the COP888 microcontroller timing.

Data Memory Fetches

All data memory accesses are performed using the internal memory address register (MAR). The contents of MAR selects the location within the data memory address space to be read/written by the current instruction. It should be noted that Memory Direct to Memory Direct data transfers and operations are not supported by the instruction set.

The MAR is loaded with the contents of the B pointer during the last instruction cycle of all instructions. Therefore, instructions which use the Register B Indirect mode of addressing are extremely efficient. This is because the address of the memory location to be accessed during an instruction is already present in the MAR at the start of the instruction. Instructions which use Memory Direct addressing or Register X Indirect addressing to access data memory require an extra one or two instruction cycles to fetch and load the desired memory address into the MAR before the actual instruction is executed.

Some instructions which use Memory Direct addressing are more efficient when addressing the data registers located between 00F0 and 00FF Hex. This is because the complete memory address of the register is contained in the first byte of the instruction opcode. This allows the MAR to be loaded with the new address in the first instruction cycle of the instruction. Instructions which do not access data memory do not affect the MAR. During the execution of instructions which use the ALU and an operand from data memory, the contents of the memory location addressed by the MAR is loaded into the memory data register (MDR) before being fed into the ALU.

Program Memory Fetches

All program memory accesses are performed using the 15-bit program counter (PC). This includes accesses to program memory for table lookups. At any given time, the PC addresses one byte within program memory. This byte is loaded into the instruction register for decoding, or used as immediate or memory address data. All data/opcode fetches cause the PC to be incremented automatically, so that the PC typically points to one program memory location ahead of the current instruction byte being executed. This allows pre-fetching of opcodes. This is also the reason why table lookup instructions (LAID, JID) located at the last byte within a 256-byte program memory page cause fetches from program memory locations in the following 256-byte page. (The JID and LAID instructions replace the lower eight bits of the PC, and rely on the current upper seven bits of the PC to form the complete address for table lookups. However, the upper seven bits of the PC change when the PC is automatically incremented over a page boundary.)

2.5.2 Instruction Decoding and Execution

All instruction decoding is performed by the CPU control logic. Single-byte opcodes require a single memory fetch. Therefore, many single-byte opcodes are single-cycle. Multiple-byte opcodes require more than one program memory fetch; the first byte is decoded to determine the number of program fetches needed to complete the instruction. Only one program memory fetch can be performed during a single instruction cycle. Therefore, an instruction always requires at least as many instruction cycles to execute as number of opcode bytes.

NOTE: Data and program memory fetches may be performed in the same instruction cycle due to the Harvard-style architecture of the COP888 Family.

The instruction cycle clock (t_C) always equals one-tenth the frequency of the clock signal at the CKI pin. All instructions are executed in multiples of the instruction cycle clock period.

During the last cycle of an instruction, the next instruction's first byte is always fetched from program memory. In addition, the PC is always incremented. This means that at the start of the first cycle of an instruction, the opcode for that instruction is already in the IR and the PC is pointing to the next instruction byte. In order to generate skips (non-execution of an instruction), the microcontroller Skip Logic is activated. This prevents an instruction (already located in the IR) from being executed by the microcontroller. Skipped instructions require X number of cycles to be skipped, where X equals the number of bytes in the skipped instruction's opcode.

The exact number of instruction cycles required for an instruction to execute can be found in Section 12.6. As noted previously, memory fetches (and therefore addressing modes) greatly influence instruction execution time. In order to optimize instruction execution time, the programmer should pay special attention to these items when developing code.

The following sections explain the steps performed by the control logic when executing different instructions.

One-Cycle Instructions

During the single cycle of one of these instructions, the following steps are performed:

1. The instruction is decoded and executed. (The instruction opcode is already in the IR at the start of the instruction cycle due to pre-fetching).
2. The next instruction is fetched from program memory.
3. The PC is incremented.

Two-Cycle Instructions

A two-cycle instruction has either a one-byte or two-byte opcode. These instructions each fall into one of five instruction categories: logical, arithmetic, conditional, exchange or load. The operations performed during two-cycle instructions are given below.

The logical, arithmetic and conditional instructions that use the Immediate addressing mode each have the following steps:

Cycle 1: Decode the opcode for the instruction. Fetch the immediate data from program memory. Execute the instruction. Activate Skip Logic if necessary. Increment the PC. (The logical/arithmetic or conditional instruction is complete at the end of this instruction cycle.)

Cycle 2: Fetch the first byte of the next instruction. Increment the PC.

Two-cycle load and exchange accumulator instructions, and load memory indirect using the B pointer have these steps:

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 2: If necessary, increment or decrement the B pointer. Load the contents of the B pointer into MAR. Fetch the first byte of the next instruction. Increment the PC.

Three-Cycle Instructions

The device has nine three-cycle load and exchange instructions. A generic overview of the sequence of steps performed by the COP888 in executing these instructions is given below.

Cycle 1: Decode the opcode for the instruction. If necessary, fetch the memory direct address from program memory and increment the PC. Load the MAR with the address of the data memory location to be accessed (either the address fetched from program memory or the contents of the X pointer depending on the instruction). If necessary, increment or decrement the X pointer.

Cycle 2: If necessary, fetch the immediate data from program memory and increment the PC. Execute the instruction. (The load or exchange is complete at the end of this instruction cycle.)

Cycle 3: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The remaining three-cycle instructions are all unique. Therefore, the sequence of events is given separately for each.

JP Instruction

Cycle 1: At the beginning of this instruction cycle, the PC is one count ahead of the address of the JP instruction. Decode the instruction opcode. Add the lower six bits of the contents of the IR (the JP opcode) to the lower byte of the PC.

Cycle 2: If the offset contained in the JP opcode was positive and the add performed in Cycle 1 had a carry out (overflow), increment the upper byte of the PC. If the offset was negative and no carry out was produced by the add in Cycle 1(underflow), decrement the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

JMP Instruction

Cycle 1: Decode the instruction opcode. Fetch the lower byte of the branch address from program memory. Load the lower byte of the PC with the fetched address.

Cycle 2: Load the four least significant bits of the JMP opcode stored in the IR into the four least significant bits of the upper byte of the PC.

Cycle 3: Fetch the first byte of the next instruction (instruction located at the branch address). Increment the PC.

LAID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the accumulator back to the lower-byte of the PC. Store the fetched byte in the accumulator.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

JID Instruction

Cycle 1: Decode the instruction opcode. Exchange the lower byte of the PC with the contents of the accumulator.

Cycle 2: Fetch the byte from program memory addressed by the PC. Transfer the contents of the PC back to the accumulator (restore the contents of the ACC). Store the fetched byte in the lower-byte of the PC.

Cycle 3: Fetch the first byte of the next instruction. Increment the PC.

DRSZ Instruction

Cycle 1: Decode the opcode of the instruction. Load the MAR with the address of the register being decremented.

Cycle 2: Decrement the contents of the register addressed by the MAR. If the result is zero, activate the Skip Logic.

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

PUSH Instruction

Cycle 1: Decode the opcode of the instruction. Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 2: Load the memory location addressed by MAR (the first available stack location) with the contents of the accumulator.

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

POP Instruction

Cycle 1: Decode the opcode of the instruction. Increment the Stack Pointer to point to the last entry in stack. Load the MAR with the address contained in the Stack Pointer.

Cycle 2: Load the accumulator with the contents of the memory location addressed by the MAR (last stack entry).

Cycle 3: Load the MAR with the contents of the B pointer. Fetch the first byte of the next instruction. Increment the PC.

Four-Cycle Instructions

All four-cycle instructions except JMPL use the Memory Direct addressing mode. The following steps outline the general sequence of events performed during the execution of these memory direct instructions.

Cycle 1: Decode the Memory Direct mode opcode prefix. Fetch the memory direct address from program memory and store it in the MAR. Increment the PC.

Cycle 2: Fetch the actual opcode from program memory and store it in the IR.

Cycle 3: Execute the instruction. (The bit manipulation, conditional test, or logical/arithmetic operation is complete at the end of this instruction cycle.)

Cycle 4: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

A JMPL has the following steps:

Cycle 1: Decode the JMPL opcode. Fetch the second byte of the instruction (the high-order byte of the branch address) and store it in IR. Increment the PC.

Cycle 2: Fetch the third byte of the instruction (the low-order byte of the branch address) and load it into the lower byte of the PC.

Cycle 3: Load the high-order byte of the branch address from the IR into the upper byte of the PC.

Cycle 4: Fetch the next instruction (located at the branch address). Increment the PC.

Five-Cycle Instructions

The COP888 has six five-cycle instructions; JSR, JSRL, VIS, RET, RETI and RETSK. All of these instructions force program branches.

The COP888 performs the following steps during the JSR and JSRL instructions:

Cycle 1: Decode the opcode for the instruction. Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location. If JSRL, fetch the next byte of the instruction and increment the PC.

Cycle 2: Increment the PC. Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Fetch the next byte of the instruction. Load the low-order byte of the subroutine address (addressed by the MAR) into the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 4: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). If JSR, load the four bits of the high-order byte of the subroutine address stored in the IR into the PC. If JSRL, load the seven bits of the high-order byte of the subroutine address stored in the IR into the PC.

Cycle 5: Load the contents of the B pointer into the MAR. Fetch the first byte of the next instruction. Increment the PC.

The COP888 performs the following steps during the VIS instruction:

Cycle 1: Decode the opcode for the instruction. Load the low-order byte of the PC with the low-order byte of the address of the location of the vector which corresponds the highest priority pending flag. The high-order byte of the PC already contains the high-order byte of the address of the location of the vector since the vector table must reside within the same 256-byte block as the VIS instruction. (The VIS instruction may reside in the last location of the 256-byte block located above the vector table. In this case, the incrementing of the PC at the end of the previous instruction increments the high-order byte of

the PC so that it is pointing to the correct block at the time the VIS instruction is actually executed.)

Cycle 2: (At the start of this cycle, the PC is pointing to the high-order byte of the interrupt vector.) Fetch the high-order byte of the interrupt vector from program memory and load it into the instruction register (IR). Increment the PC to point to the low-order byte of the interrupt vector.

Cycle 3: Fetch the low-order byte of the interrupt vector from program memory and load it into the low-order byte of the PC.

Cycle 4: Transfer the contents of the IR to the high-order byte of the PC.

Cycle 5: Fetch the first byte of the next instruction. Increment the PC.

The COP888 performs the following steps during the RET, RETSK, and RETI instructions:

Cycle 1: Decode the opcode for the instruction. Increment the stack pointer to point to the last entry on the stack. Load the MAR with the address of the last entry in the stack (the address in the updated SP).

Cycle 2: Pop the high byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the upper byte of the PC with the high byte of the return address.

Cycle 3: Increment the stack pointer to point to the next byte of data on the stack. Load the MAR with the address of the last entry in the stack (the address in the updated SP).

Cycle 4: Pop the low-byte of the return address off the stack (the contents of the memory location addressed to by the MAR). Load the lower byte of the PC with the low byte of the return address.

Cycle 5: Load the contents of the B pointer into the MAR. If RETI, set the GIE bit. If RETSK, activate skip logic to skip the instruction at the return address. Fetch the first byte of the instruction at the return address. Increment the PC.

Seven-Cycle Instructions

The Software Trap is the only instruction which requires seven cycles to execute. This instruction is performed when a 00 opcode (INTR) is loaded into the Instruction Register. The execution of this instruction when an interrupt is not pending is considered an error. Refer to Section 2.5.3 for information on the execution of this instruction.

2.5.3 Interrupt and Error Handling

The COP888 microcontrollers have a maximum of 16 interrupt sources. All interrupts force a jump to location 00FF Hex in program memory. Therefore, all interrupt and error handling routines/branches should be located at 00FF Hex.

The CPU forces a jump to 00FF Hex by jamming the INTR opcode (00) into the IR upon detecting an interrupt. The detection of an error (Software Trap) is the result of the INTR opcode being loaded into the IR as a part of the normal program sequence. An interrupt that occurs while an instruction is being executed is not acknowledged until the end of the current instruction. If the instruction following the current instruction is to be skipped, the next instruction is skipped before the pending interrupt is acknowledged and the INTR opcode is jammed into the IR. The COP888 requires seven cycles to execute the INTR instruction.

Cycle 1: Decode 00 opcode. If not a Software Trap, reset the GIE bit. Decrement the lower byte of the PC. (Note: The address of the instruction that was ready to be executed is the return address, this needs to be saved on the stack. However, the PC is one count ahead of the current instruction, and therefore must be decremented before being saved on the stack.)

Cycle 2: If the decrementing of the lower byte of the PC caused a borrow, decrement the upper byte of the PC.

Cycle 3: Load the MAR with the address of the first available stack location (the address currently in SP). Increment the stack pointer to point to the next byte of data on the stack.

Cycle 4: Push the low-order byte of the return address onto the stack (store at the location addressed by MAR). Load the low-order byte of the PC with 0FF Hex.

Cycle 5: Load the MAR with the address of the first available stack location (the address currently in SP). Decrement the stack pointer to point to the next available stack location.

Cycle 6: Push the high-order byte of the return address onto the stack (store at the location addressed by MAR). Load the upper byte of the PC with 00 Hex.

Cycle 7: Load the contents of the B pointer into the MAR. Fetch the first byte of the instruction located at 00FF Hex. Increment the PC.

Once a branch to location 00FF Hex occurs, the programmer can use the VIS instruction or poll the available pending flags to determine the source of the interrupt. Refer to Chapter 3 for more information on interrupts and the Software Trap.

2.6 RESET

The COP888 enters a reset state immediately upon detecting a logic low on the $\overline{\text{RESET}}$ pin. When the $\overline{\text{RESET}}$ pin is pulled to a logic high, the device begins code execution within two instruction cycles. The $\overline{\text{RESET}}$ pin must be held low for a minimum of one instruction cycle to guarantee a valid reset. During power-up initialization, the external circuitry must ensure that the $\overline{\text{RESET}}$ pin is held low until the COP888 is within the specified V_{CC} voltage range. Additionally, if a crystal oscillator or resonator is used, the designer must ensure that the oscillator has had time to stabilize before the $\overline{\text{RESET}}$ pin is pulled high. An R/C delay circuit on the $\overline{\text{RESET}}$ pin with a delay 5 times greater than the power supply rise time is recommended.

All COP888 microcontrollers contain logic to initialize their internal circuitry during the reset state. The following initializations are performed at reset:

- The Program Counter is loaded with 0000 Hex.
- All bits of the PSW, CNTRL and ICNTRL registers are reset. This disables all interrupts, stops Timer 1, and disables MICROWIRE/PLUS.
- The SP is initialized to 6F Hex.

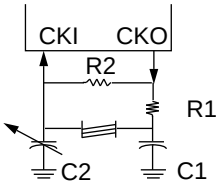
The accumulator, all data memory, and registers (including the B and X pointers) are not initialized during Reset. Refer to the device-specific chapters for details on the reset initialization of registers not found in the COP888 microcontroller core.

2.7 CLOCK OPTIONS

Most COP888 parts support two clock options: crystal oscillator and RC oscillator. Depending on the device type, the clock option is either selected via a mask option or programmed into the device by the user. Selection of a specific clock option affects the operating frequency, clocking accuracy, and power consumption of a particular device. Refer to the device-specific data sheets to obtain accurate information on frequency ranges, power consumption, and component values for the different oscillator circuits.

2.7.1 Crystal Oscillator

The dedicated CKI (clock input) pin and G7 (CKO) on the COP888 devices can be connected to make a crystal controlled oscillator as shown in Figure 2-5. If G7 is used as the CKO pin, it is not available for general-purpose use.



cop8_cry_osci

Figure 2-5 Crystal Oscillator Circuit

2.7.2 RC Oscillator

The dedicated CKI pin can be used to construct an RC oscillator as shown in Figure 2-6. With this option, G7 is available as a general-purpose input pin.

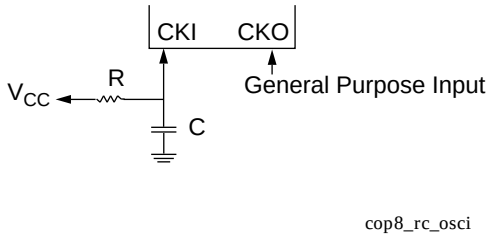


Figure 2-6 RC Oscillator Circuit

Chapter 3

INTERRUPTS

3.1 INTRODUCTION

An interrupt is an event that temporarily stops the normal flow of program execution and causes a separate interrupt service routine to be executed. After the interrupt has been serviced, execution continues with the next instruction in the program that would normally have been executed following the point of interruption.

The architecture of the COP888 family supports up to 16 different interrupts. The actual number of interrupts in a device depends on the individual device type. For example, devices that have a UART also have interrupts associated with the UART, while other devices lack that type of interrupt. The interrupts are vectored, meaning that a set of vectors (memory location pointers) stored in program memory directs the processor to one of several interrupt service routines. A special instruction called VIS (Vector Interrupt Select) directs the processor to the vector location based on the cause of the interrupt.

The various types of interrupts can be divided into two categories, maskable and non-maskable. A maskable interrupt can be enabled or disabled by the software, allowing the software to determine whether or not an occurrence of the event will trigger an interrupt. A non-maskable interrupt cannot be masked, and will always trigger an interrupt, except in certain cases when a non-maskable interrupt is already being serviced.

Figure 3-1 is a block diagram that illustrates the internal interrupt logic of a hypothetical COP888 device. The interrupt logic of actual COP888 devices is very similar to what is shown in the figure, except that the source of interrupts differ from one device type to another. The names along the left side of the diagram represent the events that can cause an interrupt. Some of these events originate from on-chip functions, while others come from off-chip devices connected to the COP888.

An interrupt is serviced when the interrupt signal is activated (the output of the OR gate on the right-hand side of the illustration). The occurrence of any one event can trigger an interrupt. The event can occur at any time: before, during, or after an instruction cycle. An occurrence immediately latches a flag bit called the pending flag. The chip hardware checks the interrupt logic at the start of each instruction, and a pending interrupt is acknowledged and handled at that time if the interrupt is enabled.

The first two interrupts, Software Trap and NMI, are both non-maskable and are always enabled. (All COP888 devices have a Software Trap, but not all have the NMI interrupt.) The remaining interrupts are all maskable. In order for a maskable interrupt event to trigger an interrupt, its own individual interrupt enable bit must be set, and the GIE (Global Interrupt Enable) bit must also be set in the PSW register. All of the maskable interrupts can be disabled together by clearing the GIE bit.

The different types of interrupts are organized by rank. If two or more interrupts are detected at the same time, the interrupt arbitration logic of the device determines which

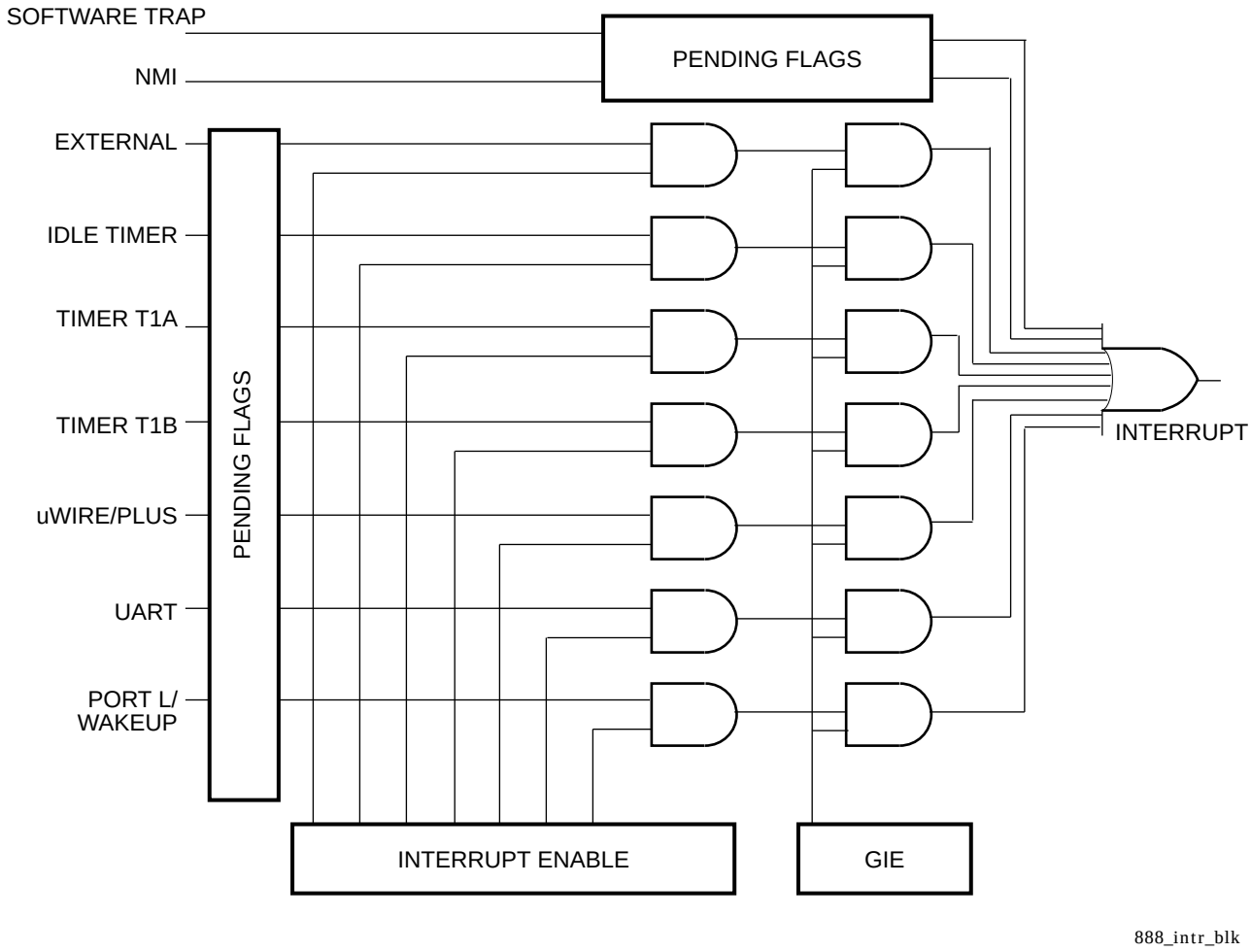


Figure 3-1 COP888 Interrupt Block Diagram

interrupt has the highest priority, and that interrupt is serviced first. The interrupts in Figure 3-1 are shown in order of rank, starting from the highest-priority interrupt, Software Trap.

3.2 VIS INSTRUCTION AND VECTOR TABLE

The general interrupt service routine, which starts at address 00FF Hex, must be capable of handling all types of interrupts. The VIS instruction, together with an interrupted vector table, directs the microcontroller to the specific interrupt handling routine based on the cause of the interrupt.

VIS is a single-byte instruction, typically used at the very beginning of the general interrupt service routine at address 00FF Hex, or shortly after that point, just after the code used for context switching (as described in Section 3.3). The VIS instruction determines which enabled and pending interrupt has the highest priority, and causes an indirect jump to the address corresponding to that interrupt source. The jump addresses (“vectors”) for all possible interrupt sources are stored in a vector table.

The vector table is 32 bytes long and resides at the top of the 256-byte block containing the VIS instruction. However, if the VIS instruction is at the very top of a 256-byte block (such as at 00FF Hex), the vector table resides at the top of the next 256-byte block. Thus, if the VIS instruction is located somewhere between 00FF and 01DF Hex (the usual case), the vector table is located between addresses 01E0 and 01FF hex. If the VIS instruction is located between 01FF and 02DF Hex, then the vector table is located between addresses 02E0 and 02FF Hex, and so on.

Each vector is 15 bits long and points to the beginning of a specific interrupt service routine somewhere in the 32-Kbyte memory space. Each vector occupies two bytes of the vector table, with the higher-order byte at the lower address. The vectors are arranged in order of interrupt priority. The vector of the maskable interrupt with the lowest rank is located at 0yE0 (higher-order byte) and 0yE1 (lower-order byte). The next priority interrupt is located at 0yE2 and 0yE3, and so forth in increasing rank. The NMI has the second highest rank and its vector is always located at 0yFC and 0yFD. The Software Trap has the highest rank and its vector is always located at 0yFE and 0yFF.

Table 3-1 shows the source of interrupts in a hypothetical COP888 device, the interrupt arbitration ranking, and the locations of the corresponding vectors in the vector table. A similar table can be found in the device-specific chapters for each of the COP888 devices. Those tables are similar to this one, except for slight differences in the list of interrupt types.

The vector table should be filled by the user with the memory locations of the specific interrupt service routines. For example, if the Software Trap routine is located at 0310 Hex, then the vector location 01FE–01FF should contain the data 03 and 10 Hex; the high address byte is stored at the lower address. When a Software Trap interrupt occurs and the VIS instruction is executed, the program jumps to the address specified in the vector table, 0310 Hex.

The interrupt sources shown in Table 3-1 are listed in order of rank, from highest to lowest priority. If two or more enabled and pending interrupts are detected at the same time, the one with the highest priority is serviced first. Upon return from the interrupt service routine, the next highest-level pending interrupt is serviced. Interrupts labelled “Reserved” in the table are not used in the device, but are reserved for future use should another feature be added that can generate an interrupt.

If the VIS instruction is executed, but no interrupts are enabled and pending, the lowest-priority interrupt vector is used, and a jump is made to the corresponding address in the vector table. This is an unusual occurrence, and probably the result of an error. It can result from a change in the enable bits or pending flags prior to using the VIS instruction, or from inadvertent execution of the VIS command outside of the context of an interrupt. It is a good idea to make this vector point to the Software Trap interrupt service routine or some other error handling routine. A normal RETI instruction should not be used in any such routine because the stack might not contain a valid return address.

To ensure reliable operation, the user should always use the VIS instruction to determine the source of an interrupt. Although it is possible to poll the pending bits to detect the source of an interrupt, this practice is not recommended. The use of polling allows the standard arbitration ranking to be altered, but the reliability of the interrupt system is compromised. The polling routine must individually test the enable and pending bits of each maskable interrupt. If a Software Trap interrupt should occur, it will be service last,

Table 3-1 Interrupt Vector Table

Arbitration Rank	Interrupt Description	Vector Address[a]
1	Software Trap	01FE–01FF
2	NMI	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	(Reserved)	01EE–01EF
10	(Reserved)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	(Reserved)	01E6–01E7
14	(Reserved)	01E4–01E5
15	Port L/Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1

a. The location of the vector table depends on the location of the VIS instruction. The vector addresses shown in the table assume a VIS instruction between 00FF and 01DF Hex.

even though it should have the highest priority. Under certain conditions, a Software Trap could be triggered but not serviced, resulting in a an inadvertent “locking out” of all maskable interrupts by the Software Trap pending flag. Problems such as this can be avoided by using the VIS instruction.

3.3 CONTEXT SWITCHING

In some applications, the first action that should be carried out in the interrupt service routine is to save the contents of the registers so that the “context” of the microcontroller can be restored just before returning from the interrupt. The register values can be saved on the stack. When interrupt servicing is finished, the register values can be popped from the stack and restored to the registers, allowing program execution to proceed with the original register values.

For example, if registers A, X, and B are used within the interrupt service routines, the following assembly code can be used to save the contents of those registers.

```
          .=00FF          ;Start at interrupt address
SAVE:     PUSH      A     ;Push Accumulator contents onto stack
          LD        A,B
          PUSH      A     ;Push B pointer onto stack
          LD        A,X
          PUSH      A     ;Push X pointer onto stack
          VIS
```

Upon completion of interrupt servicing, the following routine can be used to restore the context to the registers and return from the interrupt.

```
RESTOR:   POP       A     ;Pop X pointer from stack
          X         A,X   ;Restore X pointer
          POP       A     :Pop B pointer from stack
          X         A,B   ;Restore B pointer
          POP       A     ;Restore Accumulator contents
          RETI            ;Return from interrupt
```

In the example above, the SAVE routine resides at 00FF Hex, the interrupt address, so it is executed at the beginning for any type of interrupt. The VIS instruction causes a jump to a specific interrupt service routine, depending on the interrupt source. Each specific interrupt service routine (except the Software Trap) should end with a jump to the RESTOR routine.

3.4 MASKABLE INTERRUPTS

All interrupts other than the Software Trap and NMI are maskable. Each maskable interrupt has an associated enable bit and pending flag bit. The pending bit is latched to 1 when the interrupt condition occurs. The state of the interrupt's enable bit, combined with the GIE bit, determines whether an active pending flag actually triggers an interrupt. All of the maskable interrupt pending and enable bits belong to memory-mapped control registers, and thus can be controlled by the software.

A maskable interrupt condition triggers an interrupt under the following conditions:

1. The enable bit associated with that interrupt is set.
2. The GIE bit is set.
3. The microcontroller is not processing a non-makable interrupt. (If a non-maskable interrupt is being serviced, a maskable interrupt must wait until that service routine is completed.)

An interrupt is triggered only when all of these conditions are met at the beginning of an instruction. If different maskable interrupts meet these conditions simultaneously, the highest-priority interrupt will be serviced first, and the other pending interrupts must wait.

Upon Reset, all pending bits, individual enable bits, and the GIE bit are reset to zero. Thus, a maskable interrupt condition cannot trigger an interrupt until the program enables it by setting both the GIE bit and the individual enable bit. When enabling an interrupt, the user should consider whether or not a previously activated (set) pending bit should be acknowledged. If, at the time an interrupt is enabled, any previous occurrences of the interrupt should be ignored, the associated pending bit must be reset to zero prior to enabling the interrupt. Otherwise, the interrupt may be simply enabled; if the pending bit is already set, it will immediately trigger an interrupt. A maskable interrupt is active if its associated enable and pending bits are set.

At the start of interrupt acknowledgment, the control logic halts normal program execution by jamming an INTR (00) opcode into the instruction register. As a result, the following actions occur:

1. The GIE bit is automatically reset to zero, preventing any subsequent maskable interrupt from interrupting the current service routine. This feature prevents one maskable interrupt from interrupting another one being serviced.

2. The address of the instruction about to be executed is pushed onto the stack.

3. The program counter is loaded with 00FF Hex, causing a jump to that program memory location. A general interrupt service routine must start at that address.

The COP888 requires seven instruction cycles to perform the actions listed above.

The interrupt service routine stored at location 00FF Hex should use the VIS instruction (described in Section 3.2) to determine the cause of the interrupt, and jump to the interrupt handling routine corresponding to the highest-priority enabled and active interrupt. Alternately, you may choose to poll all interrupt pending and enable bits to determine the source(s) of the interrupt. If more than one interrupt is active, the program must decide which interrupt to service.

It is possible for one interrupt source to trigger an interrupt, and a different interrupt source to be serviced. For example, a low-priority maskable interrupt occurs, setting its pending flag and triggering an interrupt. While the interrupt is being acknowledged (jump to 00FF Hex) or context saving is being performed by the general interrupt service routine, another (higher-priority) maskable event occurs, setting its own pending flag. When the service routine reaches the VIS instruction, the higher-priority event is serviced first, and the lower-priority event that triggered the interrupt must wait. Assuming that no more interrupt events occur, the lower-priority event will be serviced upon return from the interrupt routine of the higher-priority event.

If you wish to allow nested interrupts, the interrupt service routine may set the GIE bit to 1 by writing to the PSW register, and thus allow other maskable interrupts to interrupt the current service routine. If nested interrupts are allowed, caution must be exercised. You must write the program in such a way as to prevent stack overflow, loss of saved context information, and other unwanted conditions.

Within a specific interrupt service routine, the associated pending bit should be cleared. This is typically done as early as possible in the service routine in order to avoid missing the next occurrence of the same type of interrupt event. Thus, if the same event occurs a

second time, even while the first occurrence is still being serviced, the second occurrence will be serviced immediately upon return from the current interrupt routine.

An interrupt service routine typically ends with an RETI instruction. This instruction sets the GIE bit back to 1, pops the address stored on the stack, and restores that address to the program counter. Program execution then proceeds with the next instruction that would have been executed had there been no interrupt. If there are any valid interrupts pending, the highest-priority interrupt is serviced immediately upon return from the previous interrupt.

3.5 NON-MASKABLE INTERRUPTS

The architecture of the COP888 family supports two non-maskable interrupts, known as the Software Trap (INTR instruction) and NMI (Non-Maskable Interrupt pin). All COP888 devices have the Software Trap interrupt, but some devices do not have an NMI pin, and therefore do not allow the use of the NMI interrupt. The device-specific chapters later in this manual indicate whether the NMI interrupt is available.

The non-maskable interrupts do not have individual enable bits, and they are not affected by (and do not affect) the GIE bit. Thus, they cannot be masked out by software. However, they each have a pending flag. While a non-maskable interrupt is being serviced (when its pending flag is set), maskable interrupts are prevented from being acknowledged, but their pending bits can still be set. Pending maskable interrupts are acknowledged upon return from the non-maskable interrupt.

A non-maskable interrupt is acknowledged in essentially the same manner as a maskable interrupt. If the NMI pending flag is found to be set at the beginning of an instruction, the address the of the next instruction that would normally be executed is saved on the stack. If the Software Trap pending flag is found to be set at the beginning of an instruction, the address of the INTR instruction which triggered the Software Trap is saved on the stack rather than the next normally executed instruction. In either case, the program then jumps to address 00FF Hex. The VIS instruction directs the microcontroller to the appropriate interrupt service routine. Note that with non-maskable interrupts, the GIE bit neither has an effect nor is affected by the interrupt; the GIE bit is used only with maskable interrupts.

To return from a non-maskable interrupt routine, the RET (Return from Subroutine) instruction or RETSK (Return from Subroutine and Skip) rather than the RETI (Return from Interrupt) instruction should be used. Doing so preserves the status of the GIE bit. The RET instruction pops the address stored on the stack and places it in the program counter, and execution of the program continues from that point. The RETI instruction does the same thing, but it also sets the GIE bit to re-enable maskable interrupts; it should be used only to return from a maskable interrupt routine.

3.5.1 Non-Maskable Interrupt Pending Flags

There is a pending flag bit associated with each non-maskable interrupt, called STPND for the Software Trap and NMIPND for the NMI interrupt. These pending flags are not memory-mapped and cannot be accessed directly by the software.

The pending flags are reset to zero when a chip Reset occurs. When a non-maskable interrupt event occurs, the associated pending bit is latched to one. When either flag is set, maskable interrupts are inhibited. The interrupt service routine should contain an RPND instruction to reset the pending flag to zero.

The RPND instruction always resets the STPND flag, but resets the NMIPND flag only if the NMI has been acknowledged and the STPND flag is already reset to zero. The reason for this behavior is because of the interaction between the Software Trap and NMI interrupts. A Software Trap can interrupt an NMI routine, but a NMI cannot interrupt a Software Trap routine. Thus, if both the STPND and NMIPND flags are set, it must be because an NMI routine was in progress and was interrupted by a Software Trap. Under these conditions, an RPND instruction in the Software Trap routine resets the STPND flag, thus ending the Software Trap pending condition, but not the NMIPND flag. Upon return from interrupt (if so programmed), a second RPND in the NMI routine will reset the NMIPND flag.

3.5.2 Software Trap

The Software Trap is a special kind of non-maskable interrupt which occurs when the INTR instruction (used to acknowledge interrupts) is fetched from program memory and placed in the instruction register. This can happen in a variety of ways, usually because of an error condition. Some examples of causes are listed below.

- If the program counter incorrectly points to a memory location beyond the available program memory space, the non-existent memory locations are interpreted as the INTR instruction.
- If the INTR instruction is deliberately placed in unused sections of the program memory, and an unexpected condition occurs which causes the code in these sections to be executed, a Software Trap will occur.
- If the stack is popped beyond the allowed limit (address 006F Hex), a Software Trap is triggered.
- A Software Trap can be triggered by a temporary hardware condition such as a brownout or power supply glitch.

The Software Trap has the highest priority of all interrupts. When a Software Trap occurs, the STPND bit is set, which inhibits all other interrupts. Nothing can interrupt a Software Trap service routine except for another Software Trap. The STPND can be reset only by the RPND instruction or a chip Reset.

The Software Trap indicates an unusual or unknown error condition. Generally, returning to normal execution at the point where the Software Trap occurred cannot be done reliably. Therefore, the Software Trap service routine should re-initialize the stack pointer and perform a recovery procedure that re-starts the software at some known point, similar to a chip Reset, but not necessarily performing all the same functions as a chip Reset. The routine must also execute the RPND instruction to reset the STPND flag. Otherwise, all other interrupts will be locked out. To the extent possible, the Software Trap routine should record or indicate the context of the microcontroller so that the cause of the Software Trap can be determined.

If you wish to return to normal execution from the point at which the Software Trap was triggered, the program must first execute RPND, followed by RETSK (Return from Subroutine and Skip) rather than RETI (Return from Interrupt) or RET (Return from Subroutine). This is because the return address stored on the stack is the address of the INTR instruction that triggered the interrupt. You need to skip that instruction in order to proceed with the next one. Otherwise, an infinite loop of Software Traps and returns will occur.

Programming a return to normal execution requires careful consideration. If the Software Trap routine is interrupted by another Software Trap, the second RPND will reset the STPND flag; upon return to the first Software Trap routine, the STPND flag will have the wrong state. To avoid problems such as this, program the Software Trap routine to perform a recovery procedure rather than a return to normal execution.

Under normal conditions, the STPND flag is reset by a RPND instruction in the Software Trap service routine. If a programming error or hardware condition (brownout, power supply glitch, etc.) sets the STPND flag without providing a way for it to be cleared, all other interrupts will be locked out. To alleviate this condition, the software should contain extra RPND instructions in the main program and in the Watchdog service routine (if present). There is no harm in executing extra RPND instructions in these parts of the program.

3.5.3 NMI

The NMI interrupt is a non-maskable interrupt triggered by a positive-going edge on the NMI input pin. (Some COP888 devices have do not have an NMI pin, and therefore do not allow the NMI interrupt to be used.) A positive-going edge latches the NMIPND flag to a 1, and causes subsequent edges on the NMI pin to be ignored. At the beginning of the next instruction, the NMI request is acknowledged unless a Software Trap interrupt is being serviced.

The NMI interrupt has the second-highest priority among all interrupts. An NMI service routine can be interrupted only by a Software Trap and nothing else. If a second NMI request is received on the NMI pin during an NMI routine with the NMIPND flag still set, that second request is ignored and lost.

The NMI service routine should end with an RPND instruction to reset the NMIPND flag, immediately followed by a RET (Return from Subroutine) instruction to return from the interrupt. (RET should be used instead of RETI to preserve the status of the GIE bit.) This two-instruction sequence (RPND-RET) allows any enabled and pending maskable interrupts to be serviced after execution of the RET instruction. The actual clearing of the NMIPND bit is delayed one instruction cycle from the execution of the RPND bit (the RPND is executed during the first instruction cycle of the RET instruction). This allows the RET to be executed first, before allowing any pending interrupt to be acknowledged.

3.5.4 Software Trap and NMI Interaction

In systems where both the Software Trap and NMI can be used, it is important to understand what happens when both types of interrupts occur very close to each other in time, and to understand the behavior of the RPND instruction. RPND always clears the

STPND flag, but clears the NMIPND flag only if the NMI has been acknowledged and the STPND flag is already cleared. The flag clearing performed by the RPND instruction is delayed one instruction cycle from the execution of the instruction.

First consider the case where an NMI service routine is in progress and a Software Trap occurs. The Software Trap is acknowledged, the program jumps to 00FF hex, and the VIS instruction causes a jump to the Software Trap routine, which has the highest priority. At the end of this routine, an RPND instruction clears the STPND flag, but not the NMIPND flag because the user might want to return to the NMI service routine in order to complete it. If the Software Trap routine re-starts the software at a known point (ignore the return address stored on the stack), a second RPND instruction should be used in the Software Trap service routine to clear the NMIPND flag. (Two RPND instructions in a row will first clear the STPND flag and then the NMIPND flag.) The software routine does not know (and cannot find out) whether the NMIPND flag is set, but it is not a problem to have an extra RPND instruction in the routine.

Now consider the case where a Software Trap routine is in progress and an NMI interrupt request is received. The NMI is not acknowledged and must wait until the STPND flag has been cleared. If the Software Trap service routine ends with a RPND followed by a RETSK, the pending NMI will interrupt the CPU right after the RETSK. If the Software Trap routine ends with two RPND instructions as explained in the previous paragraph, the NMI will be acknowledged after the second RPND instruction. The second RPND instruction does not clear the NMIPND flag because at that point, the NMI has not yet been acknowledged.

In summary, the RPND instruction always resets the STPND bit and resets the NMI pending bit only if the following two conditions are satisfied:

1. The ST pending bit is not set.
2. The NMI has already interrupted the CPU (actually caused an interrupt).

3.6 INTERRUPT SUMMARY

The COP888 uses the following types of interrupts, listed below in order of priority:

1. The Software Trap non-maskable interrupt, triggered by the INTR (00 op-code) instruction. The Software Trap is acknowledged immediately. This interrupt service routine can be interrupted only by another Software Trap. The Software Trap should end with two RPND instructions followed by a re-start procedure.
2. NMI non-maskable interrupt, triggered by a positive-edge transition on the NMI input pin (not available on some COP888 devices). An NMI request will interrupt any lower-level interrupt (i.e. any maskable interrupt) in progress. An NMI service routine can be interrupted only by a Software Trap. If another NMI request is received during an NMI service routine, the second NMI request is ignored and lost. The NMI routine should end with the instructions RPND and RET or RPND followed by a re-start procedure.

3. Maskable interrupts, triggered by an on-chip peripheral block or an external device connected to the COP888. The list of maskable interrupts and their priority ranking vary from one device type to another. Under ordinary conditions, a maskable interrupt will not interrupt any other interrupt routine in progress. A maskable interrupt routine in progress can be interrupted by any non-maskable interrupt request (NMI or Software Trap). A maskable interrupt routine should end with an RETI instruction.

Chapter 4

TIMERS

4.1 INTRODUCTION

The COP888 device contains a versatile 16-bit timer/counter that can satisfy a wide range of application requirements. The timer can be configured to operate in any of three modes:

- Processor-independent Pulse Width Modulation (PWM) mode, which generates pulses of a specified width and duty cycle
- External event counter mode, which counts occurrences of an external event
- Input capture mode, which measures the elapsed time between occurrences of an external event

The 16-bit timer/counter is designated Timer T1. Many COP888 devices have more than one of this type of timer, each timer having its own set of registers and I/O pins. Additional timers of this type are designated T2, T3, and so on. See the device-specific chapters for information on the number of timers contained in any particular device.

Many devices also have an IDLE Timer, designated T0, which is used for different purposes: for timing the duration of the IDLE mode, for timing the Watchdog service window, and for timing the start-up delay when exiting the HALT mode. The structure of the IDLE Timer is described in this chapter. Usage of the IDLE timer in different contexts is covered in other chapters of this manual.

4.2 TIMER/COUNTER BLOCK

The section of the device containing the timer circuitry is called the timer/counter block. This block contains a 16-bit counter/timer register, designated T1, and two associated 16-bit autoload/capture registers, designated R1A and R1B. Each 16-bit register is organized as a pair of 8-bit memory-mapped register bytes. The register bytes reside at the following data memory addresses:

Register	Address
T1:	00EA-00EB
R1A:	00EC-00ED
R1B:	00E6-00E7

In each case, the lower byte resides at the lower memory address.

The timer/counter block uses two I/O pins, designated T1A and T1B, which are alternate functions of G3 and G2 (Port G, bits 3 and 2), respectively.

The timer can be started or stopped under program control. When running, the timer counts down (decrements). Depending on the operating mode, the timer counts either instruction clock cycles or transitions on the T1A pin. Occurrences of timer underflows (transitions from 0000 to FFFF) can either generate an interrupt and/or toggle the T1A pin, also depending on the operating mode.

There are two interrupts associated with the timer, designated the T1A interrupt and the T1B interrupt. When timer interrupts are enabled, the source of the interrupt depends on the timer operating mode: either a timer underflow, a transfer of data to or from the R1A or R1B register, or an input signal received on the T1B pin.

4.3 TIMER CONTROL BITS

Timer T1 is controlled by reading and writing eight bits contained within three registers of the CPU core: the PSW (Processor Status Word), CNTRL (Control), and ICNTRL (Interrupt Control) registers. By programming these control bits, the user can enable or disable the timer interrupts, set the operating mode, and start or stop the timer. The control bits operate as described in Tables 4-1 and 4-2.

Table 4-1 Timer Control Bits

Register/Bit	Name	Function
PSW/Bit 5	T1PNDA	T1A interrupt pending flag: 1 = T1A interrupt pending, 0 = T1A interrupt not pending
PSW/Bit 4	T1ENA	T1A interrupt enable bit: 1 = T1A interrupt enabled, 0 = T1A interrupt disabled
ICNTRL/Bit 1	T1PNDB	T1B interrupt pending flag: 1 = T1B interrupt pending, 0 = T1B interrupt not pending
ICNTRL/Bit 0	T1ENB	T1B interrupt enable bit: 1 = T1B interrupt enabled, 0 = T1B interrupt disabled
CNTRL/Bit 7	T1C3	Timer T1 control bit 3 (see Table 4-2)
CNTRL/Bit 6	T1C2	Timer T1 control bit 2 (see Table 4-2)
CNTRL/Bit 5	T1C1	Timer T1 control bit 1 (see Table 4-2)
CNTRL/Bit 4	T1C0	Timer T1 run: 1 = Start timer, 0 = Stop timer; or Timer T1 underflow interrupt pending flag in input capture mode

Table 4-2 Timer Mode Control Bits

T1C3-T1C2-T1C1	Operating Mode	T1A Interrupt	T1B Interrupt	Timer Counts On
0-0-0	External event counter with auto-load register	Timer underflow	T1B positive edge	T1A positive edge
0-0-1	External event counter with auto-load register	Timer underflow	T1B positive edge	T1A negative edge
0-1-0	Timer with input capture registers	T1A positive edge or timer underflow	T1B positive edge	Instruction clock
0-1-1	Timer with input capture registers	T1A negative edge or timer underflow	T1B positive edge	Instruction clock
1-0-0	PWM timer with autoload registers; without T1A toggle	Autoload R1A	Autoload R1B	Instruction clock
1-0-1	PWM timer with autoload registers; toggle T1A output	Autoload R1A	Autoload R1B	Instruction clock
1-1-0	Timer with input capture register	T1A positive edge or timer underflow	T1B negative edge	Instruction clock
1-1-1	Timer with input capture register	T1A negative edge or timer underflow	T1B negative edge	Instruction clock

4.4 TIMER OPERATING MODES

The timer can be configured to operate in any one of three modes. Within each mode, there are options related to the use of the T1A and T1B I/O pins.

The Pulse Width Modulation (PWM) mode can be used to generate precise pulses of known width and duty cycle on the T1A pin (configured as an output). The timer is clocked by the instruction clock. An underflow causes the timer register to be reloaded alternately from the R1A and R1B registers, and optionally causes the T1A output to toggle. Thus, the values stored in the R1A and R1B registers control the width and duty cycle of the signal produced on T1A.

The external event counter mode can be used to count occurrences of an external event. The timer is clocked by the signal appearing on the T1A pin (configured as an input). An underflow causes the timer register to be reloaded alternately from the R1A and R1B registers.

The input capture mode can be used to precisely measure the frequency of an external clock that is slower than the instruction clock, or to measure the elapsed time between external events. The timer is clocked by the instruction clock. A transition received on the T1A or T1B pin (with the pins configured as inputs) causes a transfer of the timer contents to the R1A or R1B register, respectively.

4.4.1 PWM Mode

In the Pulse Width Modulation (PWM) mode, the timer counts down at the instruction clock rate. When an underflow occurs, the timer register is reloaded alternately from the R1A and R1B registers, and counting proceeds downward from the loaded value. At the first underflow, the timer is loaded from R1A, the second time from R1B, the third time from R1A, and so on.

The timer can be configured to toggle the T1A output bit upon underflow. This results in the generation of a clock signal on T1A with the width and duty cycle controlled by the values stored in the R1A and R1B registers. If the same values are stored in R1A and R1B, the resulting signal will have a 50% duty cycle; different values will produce a clock signal with other percentage duty cycles. This is a “processor-independent” PWM clock because once the timer is set up, no more action is required from the CPU. In cases where a clock with a duty cycle other than 50% is desired, no interrupt processing is necessary to change the reload values. By contrast, a microcontroller that has only a single reload register requires an interrupt routine to update the reload value (on-time, off-time) on each underflow.

A block diagram of the timer operating in the PWM mode is shown in Figure 4-1.

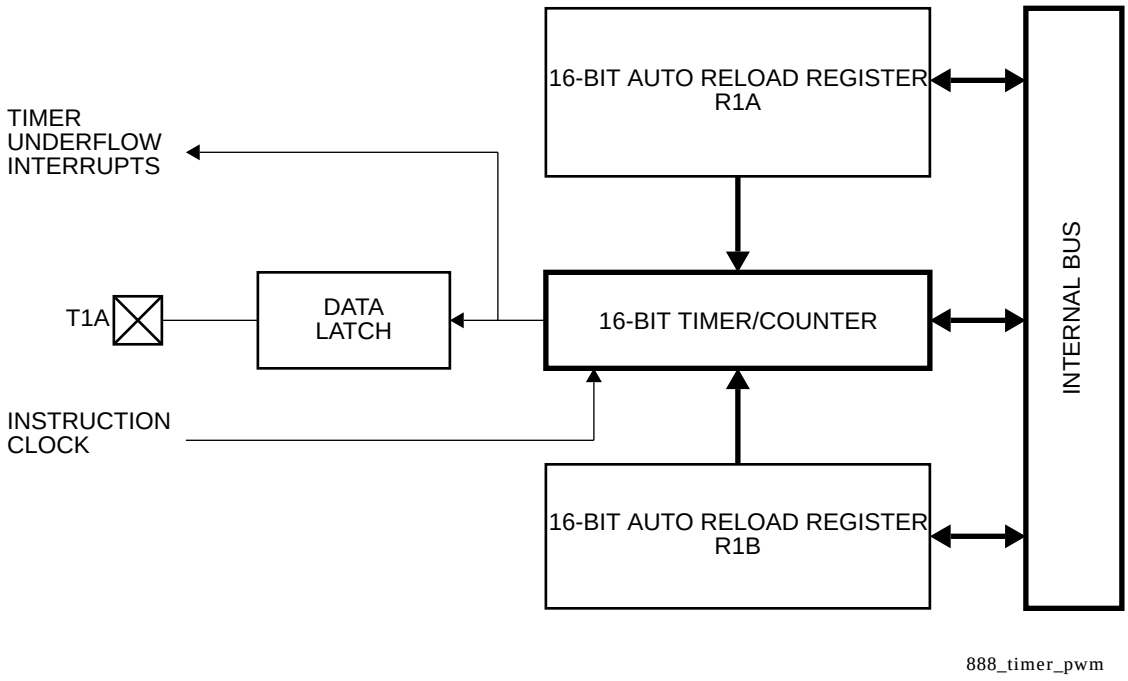


Figure 4-1 Timer in PWM Mode

There are two interrupts associated with the timer, designated T1A and T1B. The two interrupts are individually maskable by the enable bits T1ENA and T1ENB. Thus, the user can generate an interrupt on the rising edge, on the falling edge, or on both edges of the PWM output (or not at all).

When an underflow occurs that causes a timer reload from R1A, the interrupt pending flag bit T1PNDA is set. Similarly, when an underflow occurs that causes a reload from R1B, the interrupt pending flag bit T1PNDB is set. A CPU interrupt occurs if the corresponding enable bit is set and the GIE (Global Interrupt Enable) bit is also set. The interrupt service routine should reset the pending bit and perform whatever processing is necessary at the interrupt point.

The following steps can be used to operate the timer in the PWM mode. In this example, the T1A output pin is toggled with every timer underflow, and the “high” and “low” times for the T1A output are set to different values. The T1A output can start out either high or low; the instructions below are for starting with the T1A output high. (Follow the instructions in parentheses to start it low.)

1. Configure the T1A pin as an output by setting bit 3 in the Port G configuration register.
2. Initialize the T1A pin value to 1 (or 0) by setting (or clearing) the bit 3 in the Port G data register.
3. Load the PWM “high” (or “low”) time into the both the timer register and the R1B register.
4. Load the PWM “low” (or “high”) time into the R1A register.
5. Write the appropriate value to the timer control bits T1C3-T1C2-T1C1 of the CNTRL register to select the PWM mode, and to toggle the T1A output with every timer underflow (see Table 4-2).
6. Set the T1C0 bit in the CNTRL register to start the timer.

If the user wishes to generate an interrupt on timer output transitions, reset the pending flags and then enable the desired type(s) of interrupts using T1ENA and/or T1ENB. The GIE bit must also be set. The interrupt service routine should reset the pending flag and perform whatever processing is desired.

The selectable range for the PWM “high” and “low” times is 1 to 65,536 clock cycles. For a 1 MHz clock, this corresponds to a time range of 1 microsecond to 65.5 milliseconds. Thus, the pulse period (“high” plus “low” times) can range from 2 microseconds to 131 milliseconds.

4.4.2 External Event Counter Mode

The external event counter mode is similar to the PWM mode, except that instead of counting instruction clock pulses, the timer counts transitions received on the T1A pin (configured as an input). The T1A pin should be connected to an external device that generates a pulse for each event to be counted.

The timer can be configured to sense either positive-going or negative-going transitions on the T1A pin. The maximum frequency at which transitions can be sensed is one-half the frequency of the instruction clock.

As with the PWM mode, when an underflow occurs, the timer register is reloaded alternately from the R1A and R1B registers, and counting proceeds downward from the loaded value.

Interrupt generation in the external event counter mode is different from the PWM mode. If the T1A interrupt is enabled by T1ENA, an interrupt is generated with every underflow regardless of whether the timer register is reloaded from R1A or R1B. If the T1B interrupt is enabled with T1ENB, an interrupt is generated when a positive-going transition is detected on the T1B pin (configured as an input).

A block diagram of the timer operating in the external event counter mode is shown in Figure 4-2.

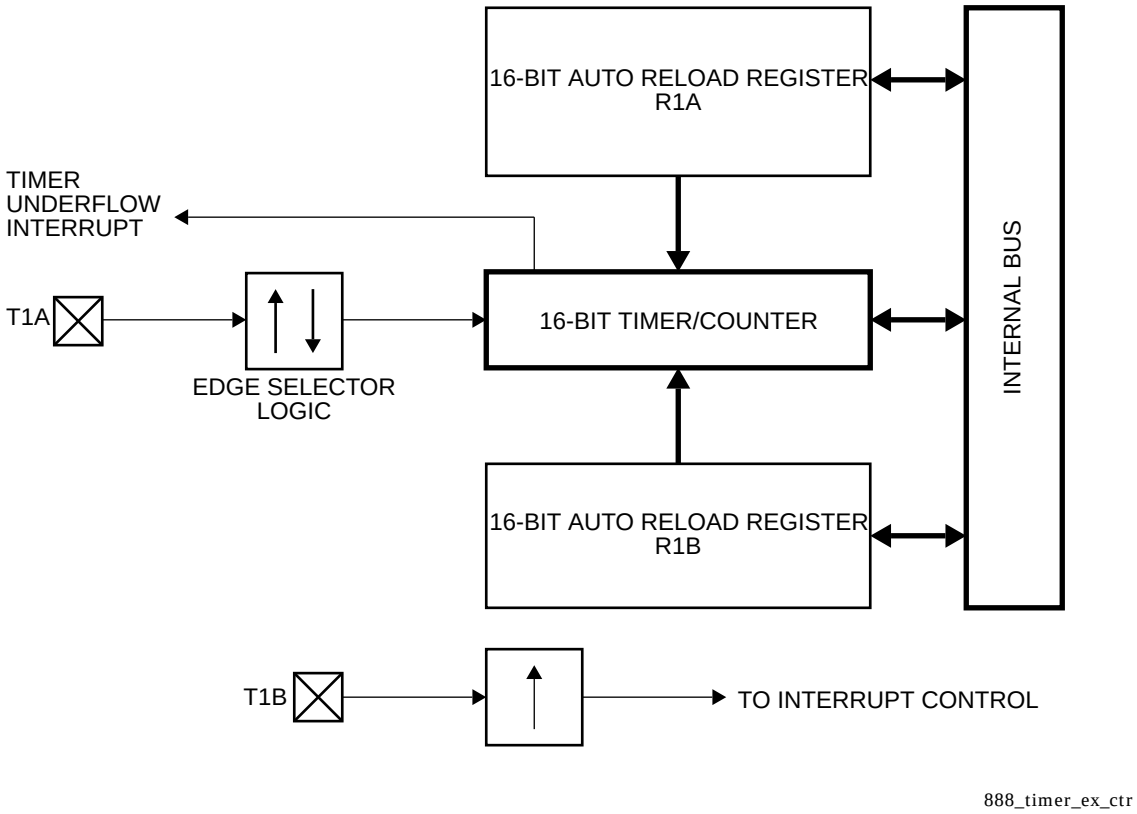


Figure 4-2 Timer in External Event Counter Mode

The following steps can be used to operate the timer in the external event counter mode.

1. Configure the T1A and T1B pins as an inputs by clearing bits 3 and 2 in the Port G configuration register.
2. Load the initial count into the timer register, the R1A register, and the R1B register. When this number of external events is detected, the counter will reach zero, however, it will not underflow until the next event is detected. To count N pulses, load the value N-1 into the registers. If it is only necessary to

count the number of occurrences and no action needs to be taken at a particular count, load the value FFFF into the registers.

3. In order to generate an interrupt each time the timer underflows, clear the T1PNDA pending flag and then enable the interrupt by setting the T1ENA bit. In order to enable interrupts on the T1B pin, clear the T1PNDB pending flag and then set the T1ENB bit. The GIE bit must also be set.

4. Write the appropriate value to the timer control bits T1C3-T1C2- T1C1 of the CNTRL register to select the external event counter mode, and to select the type of transition to be sensed on the T1A pin (positive-going or negative-going; see Table 4-2).

5. Set the T1C0 bit in the CNTRL register to start the timer.

If interrupts are being used, the T1A interrupt service routine should clear the T1PNDA flag and take whatever action is required when the timer underflows. If the user wishes to merely count the number of occurrences of an event, and anticipates that the number of events may exceed 65,536, the interrupt service routine should record the number of underflows by incrementing a counter in memory. On each underflow, the counter/timer register is reloaded with FFFF Hex from the R1A or R1B register for the next 65,536 occurrences.

The T1B pin and the corresponding interrupt can be used for any purpose. For example, an external device can request the value of the current count by generating a positive transition on the T1B pin. In that case, the T1B interrupt service routine should reset the T1PNDB pending bit, read the timer/counter register, and complement the value to convert a down-count from FFFF Hex into a positive count value.

4.4.3 Input Capture Mode

In the input capture mode, the T1A and T1B pins are configured as inputs. The timer counts down at the instruction clock rate. A transition received on the T1A pin causes a transfer of the timer contents to the R1A register. Similarly, a transition received on the T1B pin causes a transfer of the timer contents to the R1B register. The values captured in the R1A register at different times reflect the elapsed time between transitions on the T1A pin. The same is true for the R1B register and R1B pin. Each input pin can be configured to sense either positive-going or negative-going transitions.

A block diagram of the timer operating in the input capture mode is shown in Figure 4-3.

There are three interrupt events associated with the input capture mode: input capture in R1A, input capture in R1B, and timer underflow. If interrupts are enabled, a T1A interrupt is triggered by either an input capture in R1A or a timer underflow. A T1B interrupt is triggered by an input capture in R1B.

In this operating mode, the T1C0 control bit serves as the timer underflow interrupt pending flag. The T1A interrupt service routine can look at this flag and the T1PNDA flag to determine what caused the interrupt. A set T1C0 flag means that a timer underflow occurred, whereas a set T1PNDA flag means that an input capture occurred in R1A. It is possible that both flags will be found set, meaning that both events occurred at the same time. The interrupt routine should take this possibility into consideration.

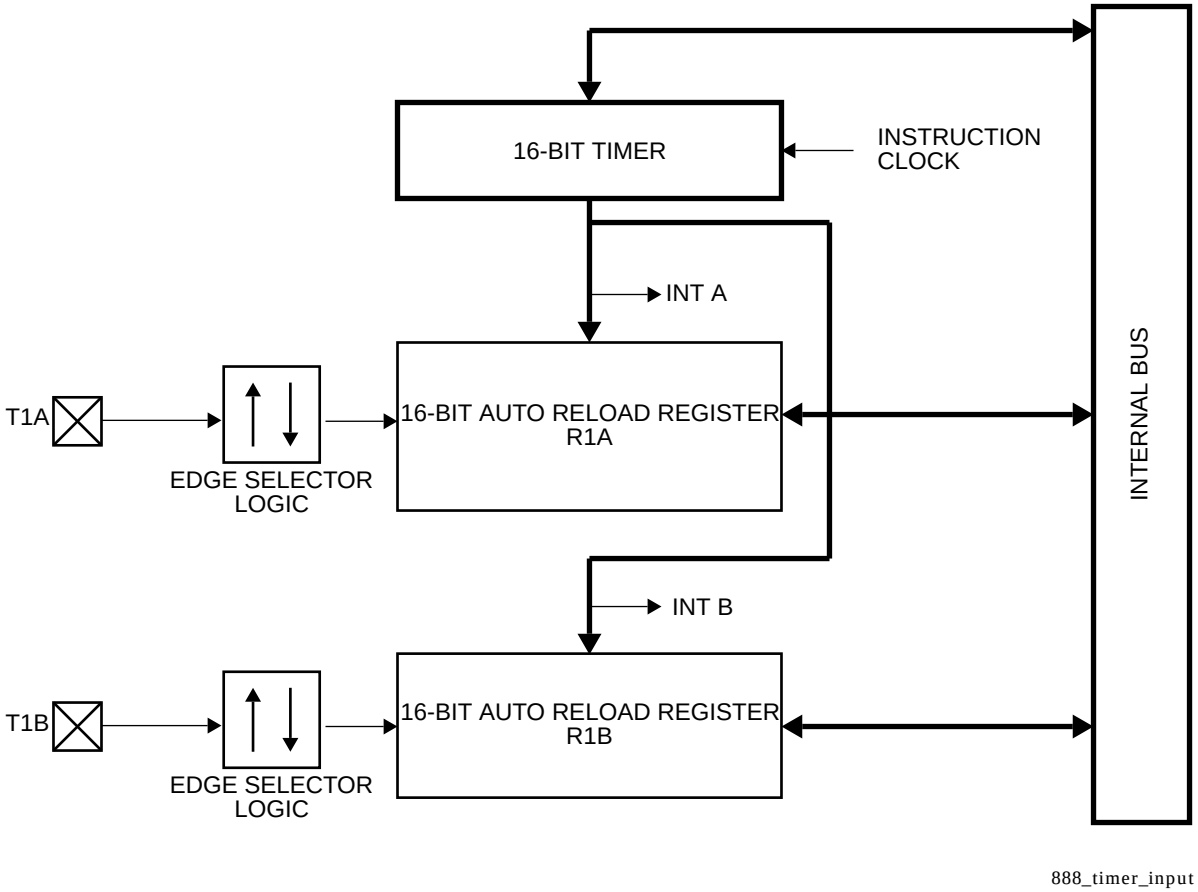


Figure 4-3 Timer in Input Capture Mode

Because the T1C0 bit is used as the underflow interrupt pending flag, it is not available for use as a start/stop bit as in the other modes. The timer/counter register counts down continuously at the instruction clock rate, starting from the time that the input capture mode is selected with bits T1C3-T1C2-T1C1. To stop the timer from running, you must change from the input capture mode to the PWM or external event counter mode and reset the T1C0 bit.

Each of the two input pins can be independently configured to sense positive-going or negative-going transitions, resulting in four possible input capture mode configurations. The edge sensitivity of pin T1A is controlled by bit T1C1, and the edge sensitivity of pin T1B is controlled by bit T1C3, as indicated in Table 4-2.

The edge sensitivity of a pin can be changed without leaving the input capture mode by setting or clearing the appropriate control bit (T1C1 or T1C3), even while the timer is running. This feature allows you to measure the width of a pulse received on an input pin. For example, the T1B pin can be programmed to be sensitive to a positive-going edge. When the positive edge is sensed, the timer contents are transferred to the R1B register, and a T1B interrupt is generated. The T1B interrupt service routine records the contents of the R1B register and also reprograms the input capture mode, changing the T1B pin from positive to negative edge sensitivity. When the negative-going edge appears on the T1B pin, another T1B interrupt is generated. The interrupt service routine reads the

R1B register again. The difference between the previous reading and the current reading reflects the elapsed time between the positive edge and negative edge on the T1B input pin, i.e., the width of the positive pulse.

The T1A pin and R1A register can be used in a similar manner, perhaps to measure the interval between some different events. Remember that the T1A interrupt service routine must test the T1C0 and T1PNDA flags to determine what caused the interrupt.

The software that measures elapsed time must take into account the possibility that an underflow occurred between the first and second readings. This can be managed by using the interrupt triggered by each underflow. The T1A interrupt service routine, after determining that an underflow caused the interrupt, should record the occurrence of an underflow by incrementing a counter in memory, or by some other means. The software that calculates the elapsed time should check the status of the underflow counter and take it into account in making the calculation.

The following steps can be used to operate the timer in the external event counter mode.

1. Configure the T1A and T1B pins as inputs by clearing bits 3 and 2 in the Port G configuration register.

2. With the timer/counter configured to operate in the PWM or external event counter mode (T1C2 equal to 0), reset the T1C0 bit. This stops the timer register from counting.

3. Load the initial count into the timer register, typically the value FFFF to allow the maximum possible number of counts before underflow.

4. Clear the T1PNDA and T1PNDB interrupt pending flags, then set the T1ENA and T1ENB interrupt enable bits. The GIE enable bit should also be set. The interrupts are now enabled.

5. Write the appropriate value to the timer control bits T1C3-T1C2-T1C1 of the CNTRL register to select the input capture mode, and to select the types of transitions to be sensed on the T1A and T1B pins (positive-going or negative-going; see Table 4-2). As soon as the input capture mode is enabled, the timer starts counting.

6. When the programmed type of edge is sensed on the T1B pin, the R1B register is loaded and a T1B interrupt is triggered. The interrupt service routine resets the T1PNDB pending bit and performs the required task such as recording the R1B register contents.

7. When the programmed type of edge is sensed on the T1A pin, the R1A register is loaded and a T1A interrupt is triggered. A T1A interrupt is also triggered when an underflow occurs in the timer register. The interrupt service routine tests both the T1PNDA and T1C0 flags to determine the cause of the interrupt, resets the pending bit, and performs the required task, such as recording the R1A register contents or incrementing an underflow counter.

4.5 ADDITIONAL GENERAL-PURPOSE TIMERS

Some COP888 devices have additional timers that operate in exactly the same manner as the T1 timer. Additional timers of this type are designated T2, T3, and so on. The device-specific chapters indicate the number of timers available in each device.

Like the T1 timer, each additional timer has its own set of three 16- bit memory-mapped registers and two I/O pins that are alternate functions of port pins. The eight control bits for each additional timer are grouped together into a single memory-mapped control byte, rather than being distributed between the PSW, CNTRL, and ICNTRL registers as in the case of Timer T1.

The register addresses and other information for the basic timer T1 and additional timers T2 and T3 are listed below.

Timer T1

T1:	Address 00EA-00EB
R1A:	Address 00EC-00ED
R1B:	Address 00E6-00E7
Control bits in CNTRL, PSW, ICNTRL:	Addresses 00EE, 00EF, 00E8
T1C3-T1C2-T1C1-T1C0:	CNTRL bits 7-6-5-4
T1PNDA-T1ENA:	PSW bits 5-4
T1PNDB-T1ENB:	ICNTRL bits 1-0
T1A pin:	Alternate function of G3
T1B pin:	Alternate function of G2

Timer T2 (If present)

T2:	Address 00C0-00C1
R1A:	Address 00C2-00C3
R1B:	Address 00C4-00C5
Control bits in T2CNTRL:	Address 00C6
T2C3-T2C2-T2C1-T2C0:	T2CNTRL bits 7-6-5-4
T2PNDA-T2ENA:	T2CNTRL bits 3-2
T2PNDB-T2ENB:	T2CNTRL bits 1-0
T2A pin:	Alternate function of L4
T2B pin:	Alternate function of L5

Timer T3 (If present)

T3:	Address 00B0-00B1
R1A:	Address 00B2-00B3
R1B:	Address 00B4-00B5
Control bits in T3CNTRL:	Address 00B6
T3C3-T3C2-T3C1-T3C0:	T3CNTRL bits 7-6-5-4
T3PNDA-T3ENA:	T3CNTRL bits 3-2
T3PNDB-T3ENB:	T3CNTRL bits 1-0
T3A pin:	Alternate function of L6
T3B pin:	Alternate function of L7

4.6 IDLE TIMER

Many COP888 devices have a special-purpose timer called the IDLE Timer, also known as Timer T0 to distinguish it from the general-purpose timers T1, T2, etc. The IDLE timer, if present, can be used for the following purposes:

- Timing the duration of the IDLE mode
- Timing a start-up delay when exiting the HALT mode (under certain conditions)
- Timing the Watchdog service window (if present)

The IDLE Timer uses a 16-bit count-down timer register. It runs continuously at the instruction clock rate whenever the device is operating, including during the IDLE mode. (The instruction clock runs at one-tenth the frequency of the chip clock.) The timer register is not memory mapped, so it cannot be read or written by the software, and it cannot be stopped from running. Its contents upon Reset are unknown. The state of the timer can be determined only by using the IDLE Timer interrupt or by polling the interrupt pending bit, described below.

Figure 4-4 is a functional block diagram showing the structure of the IDLE Timer and its associated interrupt logic.

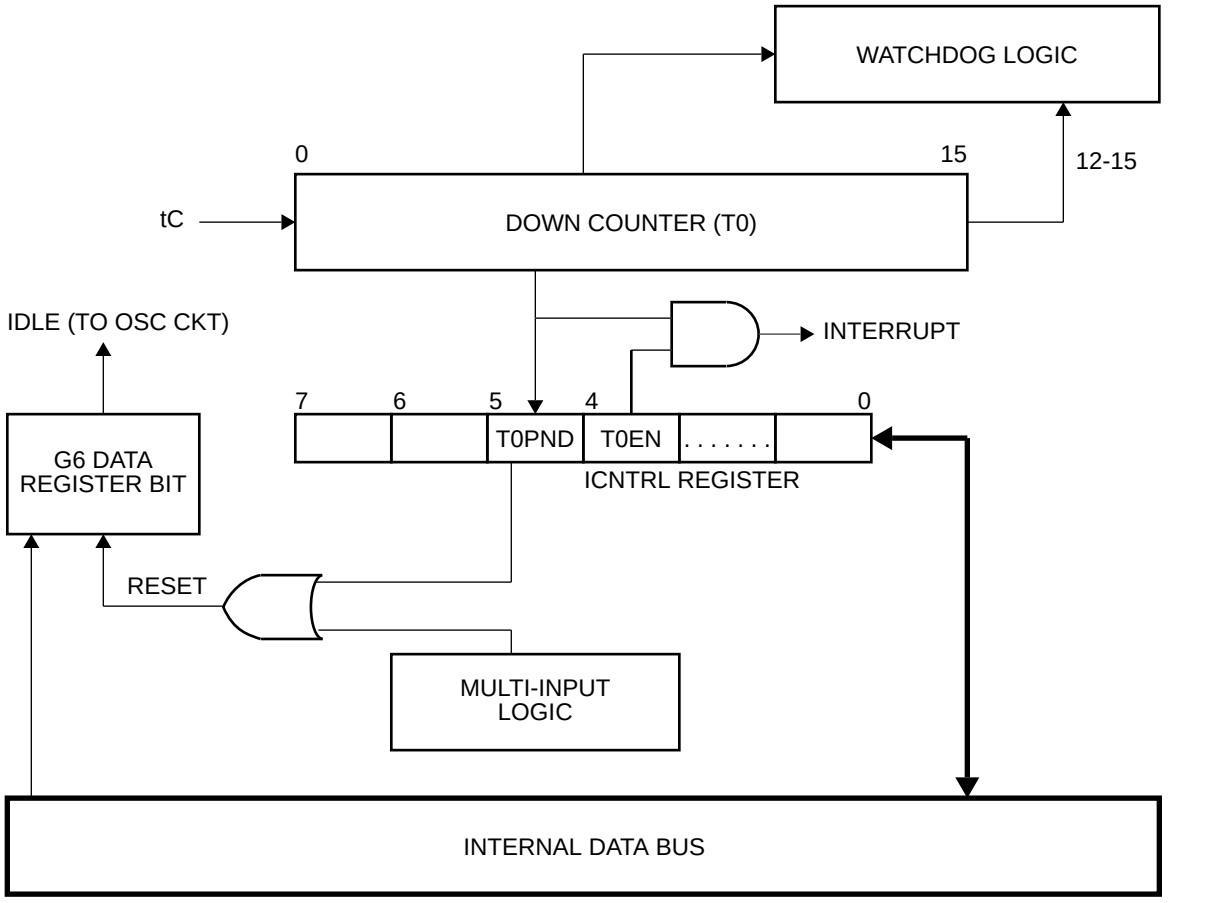


Figure 4-4 IDLE Timer (Timer T0) Block Diagram

Bit 12 of the timer register is used for triggering the IDLE Timer interrupt. Each time bit 12 toggles (every 4,096 instruction cycles), the IDLE Timer interrupt pending bit T0PND is set, thus generating an interrupt (if enabled), and bit 6 of the Port G data register is reset, thus causing an exit from the IDLE mode if the device is in that mode.

In order for an interrupt to be generated, the IDLE Timer interrupt enable bit T0EN must be set, and the GIE (Global Interrupt Enable) bit must also be set. The T0PND flag and T0EN bit are bits 5 and 4 of the ICNTRL register, respectively. The interrupt can be used for any purpose. Typically, it is used to perform a task upon exit from the IDLE mode. For more information on the IDLE mode, see Chapter 6.

The higher-order bits of the IDLE timer are used only by the Watchdog circuit (if present). The timer count establishes a “time window” in which the software must periodically write a specific value to the Watchdog service register. The Watchdog feature is described in Chapter 8.

Under certain conditions, a start-up delay is used when exiting the HALT mode, which allows the clock to reach the proper amplitude and operating frequency before program execution resumes. If a start-up delay is used, the chip hardware loads the IDLE timer register with the value 256 upon exit from the HALT mode. Program execution is delayed until the counter counts down to zero. For more information on this feature, see Chapter 6.

Chapter 5

MICROWIRE/PLUS

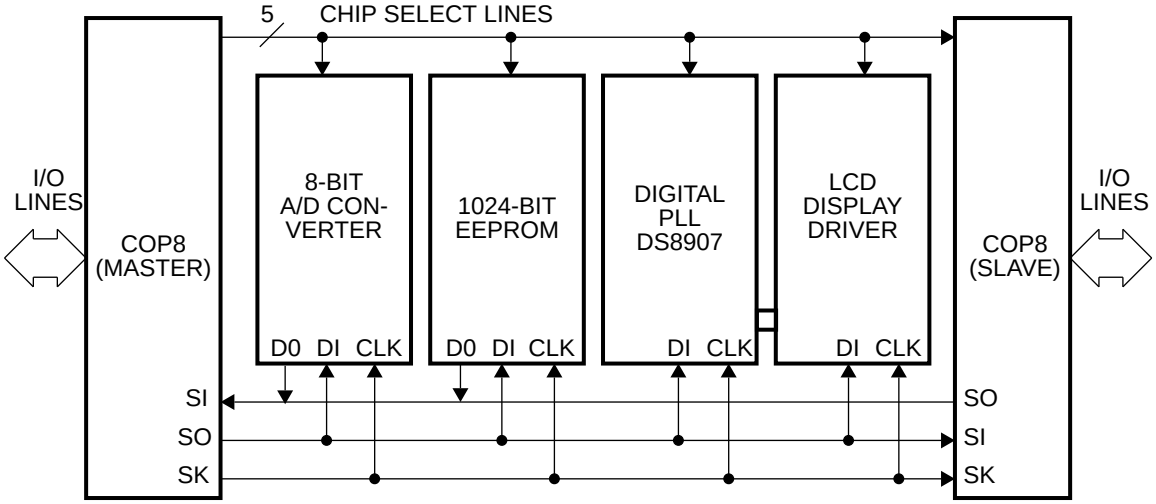
5.1 INTRODUCTION

MICROWIRE/PLUS is a synchronous serial communication system that allows the COP888 microcontroller to communicate with any other device that also supports the MICROWIRE/PLUS system. Examples of such devices include A/D converters, comparators, EEPROMs, display drivers, telecommunications devices, and other processors (e.g., HPC and COP400 processors). The MICROWIRE/PLUS serial interface uses a simple and economical 3-wire connection between devices.

Several MICROWIRE/PLUS devices can be connected to the same 3-wire system. One of these devices, operating in what is called the master mode, supplies the synchronous clock for the serial interface and initiates data transfers. Another device, operating in what is called the slave mode, responds by sending (or receiving) the requested data. The slave device uses the master's clock for serially shifting data out (or in), while the master device shifts the data in (or out).

On the COP888 device, the three interface signals are called SI (Serial Input), SO (Serial Output), and SK (Serial Clock). To the master, SO and SK are outputs (connected to slave inputs), and SI is an input (connected to slave outputs).

The COP888 can operate either as a master or a slave, depending on how it is configured by the software. Figure 5-1 shows an example of how several devices can be connected together using the MICROWIRE/PLUS system, with the COP888 on the left operating as the master, and other devices operating as slaves. The protocol for selecting and enabling slave devices is determined by the system designer.



DD11208-23
cop8_uwirep

Figure 5-1 MICROWIRE/PLUS Example

5.2 THEORY OF OPERATION

Figure 5-2 is a block diagram illustrating the internal operation of the MICROWIRE/PLUS circuit of the COP888.

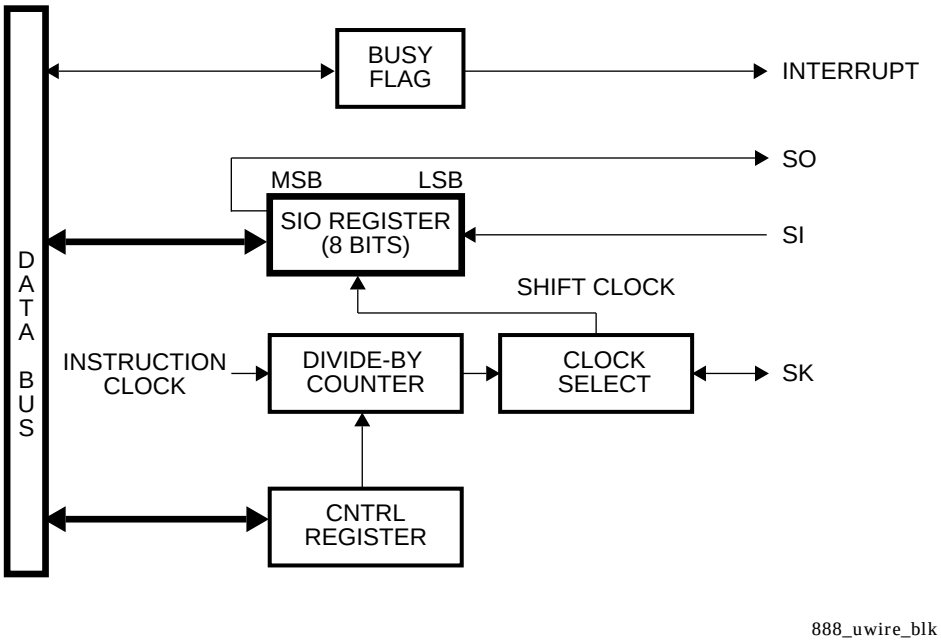


Figure 5-2 MICROWIRE/PLUS Circuit Block Diagram

An 8-bit shift register, called the SIO (Serial Input/Output) register, is used for both sending and receiving data. In either type of data transfer, bits are shifted left through the register. When a data byte is being sent, bits are shifted out through the SO output, most significant bit first. When a data byte is being received, bits are shifted in through the SI input, most significant bit first also.

The SIO register is memory-mapped in the microcontroller's data memory space, allowing the software to write a data byte to be sent, or to read a full data byte that has been received. The Busy flag in the PSW register indicates whether the SIO register is ready to be read or written. Interrupts or polling can be used to synchronize the reading or writing of the SIO register to completion of each 8-bit shift operation, or a carefully timed software loop can be programmed for this purpose.

The software should write the SIO register only when the SK clock is low. A data byte is generally written at the end of an 8-bit shifting cycle, when the SK clock is low anyway. If the software inadvertently writes to the register when SK is high, unknown data may be placed in the register.

5.2.1 Timing

The SK clock signal is generated by the master device. The timing of the MICROWIRE/PLUS interface is synchronized to this signal.

There are two operating modes for the interface, called the standard SK mode and the alternate SK mode. In the standard SK mode, output data on SO is clocked out on the falling edge of the SK clock, and input data on SI is sampled on the rising edge of the SK

clock. In the alternate SK mode, the SK clock edge functions are reversed: output data on SO is clocked out on the rising edge of the SK clock, and the input data on SI is sampled on the falling edge of the SK clock.

The timing for SK, SO, and SI interface signals is shown in Figure 5-3 for the standard SK mode, and in Figure 5-4 for the alternate SK mode. The solid arrows indicate the SK clock edges that cause the output data to be clocked out on the SO pin, and the dotted arrows indicate the clock edges that cause the input data on the SI pin to be sampled. In either mode, shifting occurs in the SIO register on the falling edge of the SK clock.

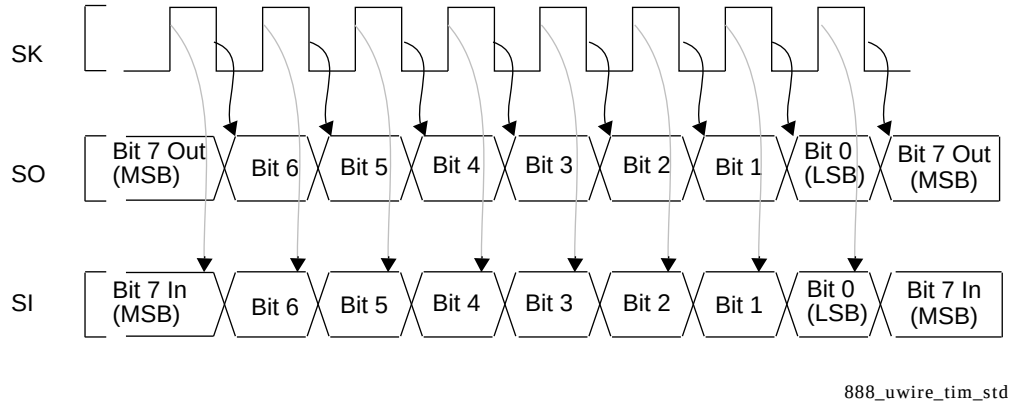


Figure 5-3 MICROWIRE/PLUS Interface Timing, Standard SK Mode

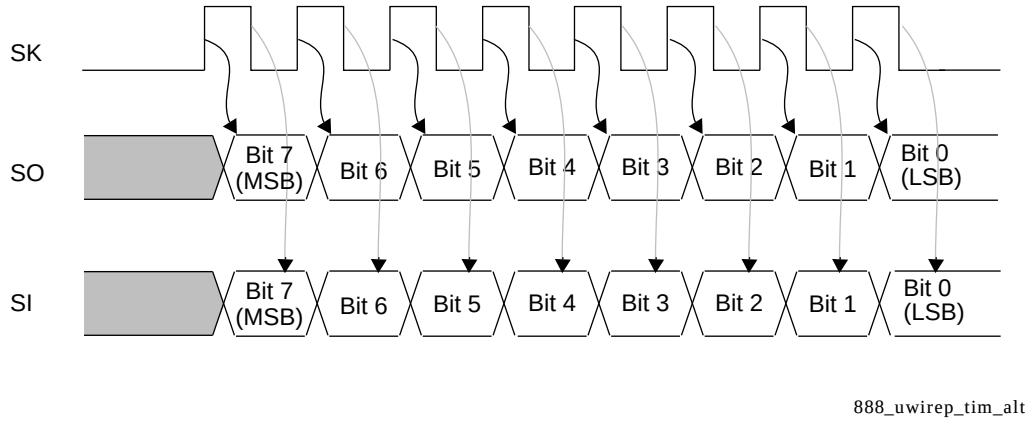


Figure 5-4 MICROWIRE/PLUS Interface Timing, Alternate SK Mode

In the standard SK mode, the most significant bit becomes valid immediately after the SIO register of the transmitting device is loaded. Thus, the data is clocked into the receiving device on the initial rising edge of the SK clock. In the alternate SK mode, the most significant bit is clocked out on the initial rising edge of the SK clock, and then clocked into the receiving device on the falling edge of that same clock pulse. The choice of which mode to use depends on the implementation of the MICROWIRE/PLUS protocol.

The SK clock mode is determined by the SKSEL bit (bit 6) in the Port G configuration register. Upon reset, this bit is reset to 0, resulting in the standard SK mode. The

software can change the operation of the COP888 to the alternate SK mode by setting the SKSEL bit.

5.2.2 Port G Configuration

The three MICROWIRE/PLUS signals SO, SK, and SI are alternate functions of Port G pins G4, G5, and G6, respectively. To enable the use of these pins for the MICROWIRE/PLUS interface, the MSEL (MICROWIRE Select) bit of the CNTRL register must be set to 1. (The SL1 and SL0 bits, also in the CNTRL register, are used to control the SK clock speed in master mode, as described below.)

Port G must be properly configured for operation of the interface. This is accomplished by writing certain bit values to the Port G configuration register. Pin G4 (SO) should be configured as an output for sending data. It can be placed in the TRI-STATE (high-impedance) mode for receiving data, thereby preventing unknown data from being placed on the SO output pin. Pin G5 (SK) should be configured as an output in master mode, or as an input in slave mode. G6 (SI) serves only as an input, so it need not be specifically configured as such. The Port G configuration register programming options are summarized in Table 5-1.

Table 5-1 Port G Configuration Register Bits

Port G Config. Reg. Bits G5-G4	MICROWIRE Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-0	Slave, data in	TRI-STATE (unused)	SK Input	SI Input
0-1	Slave, data out and data in	SO Output	SK Input	SI Input
1-0	Master, data in	TRI-STATE (unused)	SK Output	SI Input
1-1	Master, data out and data in	SO Output	SK Output	SI Input

5.2.3 SK Clock Frequency

When the COP888 operates in master mode, it generates the SK clock signal. A divide-by counter lowers the frequency of the instruction clock, producing an SK clock period that is 2, 4, or 8 times the period of the instruction clock. The divide-by factor is programmed by writing two bits to the CNTRL register, designated SL1 and SL0 (Select 1 and Select 0 bits), as indicated in Table 5-2.

The internal divide-by counter is reset when the MICROWIRE Busy flag (described below) goes to 1. Because of this, the divide-by counter always starts from 0 at the beginning of an 8-bit shift cycle, ensuring uniform SK clock pulses.

Table 5-2 Master Mode Clock Select Bits

SL1 (CNTRL Bit 1)	SL0 (CNTRL Bit 0)	SK Clock Period
0	0	2 times instruction clock period
0	1	4 times instruction clock period
1	X	8 times instruction clock period

When the COP888 microcontroller operates in slave mode, the SK clock is generated by the external master device. In this case, SK is an input, and the SK clock-generating circuit of the COP888 is inactive.

5.2.4 Busy Flag and Interrupt

A flag bit in the PSW (Processor Status Word) register indicates the status of the SIO shift register. To initiate an 8-bit shifting operation, the software sets this bit to 1. Shifting then starts and continues automatically at the SK clock rate. With each shift, the high-order bit of the register is shifted out on SO (if enabled), and simultaneously, the low-order bit of the register is shifted in from SI.

When the 8-bit shifting operation is finished, the Busy flag is automatically reset to 0 by the hardware, and a MICROWIRE/PLUS interrupt is generated (if enabled). When this occurs, the software should write the next byte to be sent (and/or read the full byte just received) and then set the Busy flag to initiate transfer of the next byte. Either polling or interrupts can be used to determine when this needs to be done.

If polling is used, the CPU runs in a continuous loop in which the status of the Busy flag is read. When the flag is found to be 0, the software reads or writes the SIO register, sets the Busy flag for the next transfer, and then returns to the polling loop.

If interrupts are used, the CPU can perform other tasks while the data byte is being shifted in or out of the SIO register. The MICROWIRE interrupt service routine resets the MICROWIRE pending flag (uWPND), reads or writes the SIO register, sets the Busy flag to one for the next transfer, and then returns from the interrupt. To enable the MICROWIRE interrupt, set the uWEN (MICROWIRE interrupt enable) bit. The GIE (Global Interrupt Enable) bit must also be set. The uWPND flag and uWEN enable bit are bits 3 and 2 of the ICNTRL register, respectively.

The software can control the timing of the transfer by setting and clearing the Busy flag directly. If the Busy bit is cleared directly by the software, shifting stops immediately. In this case an interrupt is not generated.

The handshaking protocol between the master and slave should ensure that the slave device is given enough time to respond after being enabled by the master. An example of a MICROWIRE/PLUS master/slave protocol is provided in the applications chapter.

It is possible to eliminate the need for polling or interrupts, thereby speeding up the transfer. This is accomplished by writing a software loop that executes in the exact amount of time necessary to allow an 8-bit shift operation. At the end of the loop, the

software initiates the next 8-bit transfer immediately, without checking the Busy bit. This is called the MICROWIRE “fast burst” mode. An example of this type of program is presented in the applications chapter.

Some external devices may require a continuous bit stream, without any pauses between bytes. This mode, called the MICROWIRE “continuous” mode, is also accomplished by writing a software loop that executes in a specific number of cycles. The clock divide-by factor must be 8. An example of this type of program is presented in the applications chapter.

When the COP888 operates in slave mode, the Busy flag should be set only when the SK clock signal (an input) is low. This is because the Busy bit is ANDed internally with the SK signal to produce the clock-shifting signal. If the Busy flag is set while SK is already high, the current SK pulse is gated-in immediately, resulting in a clock pulse with an unknown width (perhaps very narrow), causing unreliable shifting.

5.3 MASTER MODE OPERATION EXAMPLE

When the COP888 operates in master mode, it generates the SK clock and initiates the transfer. The application software can perform a data transfer using the numbered steps shown below.

1. Write the proper value to the CNTRL register. To enable use of the Port G pins, set the MSEL bit. To set the divide-by factor for the SK clock, write the desired 2-bit value to the SL1 and SL0 bits (Table 5-2).
2. Write the proper value to the Port G configuration register, bits G5 and G4, to make the G5 (SK) pin an output and the G4 (SO) pin either TRI-STATE or an output, depending on whether the COP888 is transmitting (Table 5-1). Write the proper value to the Port G configuration register bit 6 to select the Standard or Alternate SK Clocking mode.
3. If necessary, enable the desired slave device.
4. If sending data, write the data byte to the SIO register.
5. Set the Busy flag in the PSW register to initiate the transfer. Shifting proceeds automatically at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.
6. Run in a loop and test the Busy flag for completion of the 8-bit transfer.
7. If receiving data, read the data byte in the SIO register.
8. Repeat steps 4 through 7 until all data bytes are transferred.

Polling is used in the example above. If interrupts are to be used instead, the software should reset the uWPND pending flag and set the uWEN enable bit at the beginning. The GIE bit must also be set. The MICROWIRE interrupt service routine should be programmed to reset the pending flag, read or write the SIO register, and set the Busy flag to initiate the next transfer, thus replacing steps 4 through 8 above.

5.4 SLAVE MODE OPERATION EXAMPLE

When the COP888 operates in slave mode, the external master device generates the SK clock and initiates the transfer; SK is an input to the COP888. The application software can set up the COP888 device to allow a data transfer using the numbered steps shown below.

1. To enable use of the Port G pins, set the MSEL bit of the CNTRL register.
2. Write the proper value to the Port G configuration register to make the G5 (SK) pin an input and the G4 (SO) pin either TRI-STATE or an output, depending on whether the COP888 is transmitting (Table 5-1). Write the proper value to the Port G configuration register bit 6 to select the Standard or Alternate SK Clocking mode.
3. If sending data, write the data byte to the SIO register.
4. Set the Busy flag in the PSW register to allow the transfer. This should be done only when the SK signal is low. The handshaking protocol between the master and slave should ensure that the COP888 is given enough time to set the Busy flag before the data transfer starts. Once started, shifting proceeds at the SK clock rate. The Busy flag is automatically reset upon completion of the 8-bit transfer.
5. Run in a loop and test the Busy flag for completion of the 8-bit transfer.
6. If receiving data, read the data byte in the SIO register.
7. Repeat steps 3 through 6 until all data bytes are transferred.

Again, polling is used in the example above. Interrupts can be used instead as described under the master mode example.

Chapter 6

POWER SAVE MODES

6.1 INTRODUCTION

The COP888 supports a power-save mode of operation called the HALT mode. In this mode of operation, all processor activity stops and power consumption is reduced to a very low level. The processor can be forced to exit the HALT mode and resume normal operation at any time. The HALT feature is a mask option.

The fully static architecture of the COP888 allows the state of the microcontroller to be frozen. This is accomplished by stopping the internal clock of the device during the HALT mode. If an R-C or crystal type clock is used, the controller also stops the CKI pin from oscillating during the HALT mode.

During normal operation, typical power consumption is in the range of 1 to 10 milliamps. The actual power consumption depends heavily on the clock speed and operating voltage used in an application. In the HALT mode, the device only draws a small leakage current, plus any current necessary for driving the outputs. Typically, power consumption is reduced to less than 1 microamp. Since total power consumption is affected by the amount of current required to drive the outputs, all I/Os should be configured to draw minimal current prior to entering the HALT mode, if possible. In order to reduce power consumption even further, the power supply (V_{CC}) can be reduced to a very low level during the HALT mode, just high enough to guarantee retention of data stored in RAM. The allowed lower voltage level (Vr) is specified in the device data sheet.

Some COP888 devices also support another power-save mode called the IDLE mode. The IDLE mode is similar to the HALT mode, except that certain sections of the device continue to operate, allowing real time to be maintained: the on-board oscillator, the IDLE Timer (Timer T0), and the Clock Monitor. The device-specific chapters indicate whether the IDLE mode is available.

6.2 ENTERING THE HALT MODE

There are two ways to enter the HALT mode. One method is to simply stop the processor clock (if the hardware implementation allows it). The other method is to set bit 7 of the Port G data register.

6.2.1 Clock-Stopping Method

The clock-stopping method of entering the HALT mode can be used only if the hardware implementation of the processor clock allows it. If an R-C circuit or crystal is connected between the CKI and CKO inputs of the device, there is no practical way to stop the clock, and this method cannot be used. However, if the clock signal is generated externally and supplied to the CKI input, the external clock circuit can simply stop the clock at any time.

The clock signal at CKI should be stopped in the low state. When the clock stops, the COP888 stops running but maintains all register and RAM contents. Power consumption is reduced to a very low level. When the clock starts running again, the processor begins running again from the point at which it was stopped.

6.2.2 Port G Method

Using the Port G method, the device enters the HALT mode under software control when the Port G data register bit 7 is set to 1. All processor action stops immediately, and power consumption is reduced to a very low level.

6.3 EXITING THE HALT MODE

If the HALT mode was entered by stopping the processor clock, it is exited by re-starting the clock. The processor begins running again from the point at which it was stopped.

If the HALT mode was entered by setting bit 7 of the Port G data register, there is a choice of methods for exiting the HALT mode: a chip Reset using the $\overline{\text{RESET}}$ pin, a Multi-Input Wakeup or a low-to-high transition on the G7 pin of Port G. The Reset method can be used with any COP888 device, but the availability of the other two methods depends on the specific device type.

6.3.1 HALT Exit Using Reset

A device Reset, which is invoked by a low-level signal on the $\overline{\text{RESET}}$ input pin, takes the device out of the HALT mode and starts execution from address 0000H. The initialization software should determine what special action is needed, if any, upon start-up of the device. The initialization of all registers following a $\overline{\text{RESET}}$ exit from HALT is described in Section 2.6 and in the device-specific chapters.

6.3.2 HALT Exit Using Multi-Input Wakeup

In devices having the Multi-Input Wakeup/Interrupt feature, the device can be brought out of the HALT mode by a transition received on one of the available Wakeup pins. The pins used and the types of transitions sensed on the port pins are software programmable. For information on programming and using the Multi-Input Wakeup feature, see the description of Multi-Input Wakeup in Chapter 7.

A start-up delay may be required between the device wakeup and the execution of program instructions, depending on the type of chip clock. If a single-pin clock is used (R-C or external clock signal), the start-up delay is optional. It can be invoked under software control by setting the CLKDLY bit, which is bit 7 of the Port G configuration register. (Upon Reset, this bit is cleared, resulting in no start-up delay.)

If a two-pin, closed-loop crystal oscillator is used, the start-up delay is mandatory, and is implemented whether or not the CLKDLY bit is set. This is because all crystal oscillators and resonators require some time to reach a stable frequency and full operating amplitude. In this case, the CLKDLY flag can be used by the software for any purpose.

If the start-up delay is used, the IDLE Timer (Timer T0) provides a fixed delay from the time the clock is enabled to the time the program execution begins. Upon exit from the HALT mode, the IDLE Timer is enabled with a starting value of 256 and is decremented with each instruction cycle. (The instruction clock runs at one-tenth the frequency of the oscillator clock.) An internal Schmitt trigger connected to the on-chip CKI inverter ensures that the IDLE Timer is clocked only when the oscillator has a large enough amplitude. (The Schmitt trigger is not part of the oscillator closed loop.) When the IDLE Timer underflows, the clock signals are enabled on the chip, allowing program execution to proceed. Thus, the delay is equal to 256 instruction cycles.

To ensure accurate operation on start-up of the device using Multi-Input Wakeup, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

6.3.3 HALT Exit Using G7 Pin

Using the G7 input pin is possible only if an external clock signal is supplied to CKI or an R-C circuit is being used. If a crystal is being used, the G7/CKO pin is used as CKO, and is therefore unavailable for use as the HALT/Restart pin.

If the G7 pin is available, a low-to-high transition on the pin takes the processor out of the HALT mode, and program execution resumes from the point at which it stopped. To ensure accurate operation upon start-up of the device using the G7 pin, the instruction in the application program used for entering the HALT mode should be followed by two consecutive NOP (no-operation) instructions.

6.4 IDLE MODE

Some COP888 devices support another power-save mode called the IDLE mode. The device-specific chapters indicate whether the IDLE mode is available. In the IDLE mode, program execution stops and power consumption is reduced to a very low level as with the HALT mode. However, the on-board oscillator, IDLE Timer (Timer T0), and Clock Monitor continue to operate, allowing real time to be maintained. The device remains idle for up to 4,096 instruction cycles, or 4.096 milliseconds with a 1 MHz instruction clock frequency, and then automatically exits the IDLE mode and returns to normal program execution.

The device is placed in the IDLE mode under software control by setting the IDLE bit (bit 6 of the Port G data register).

The IDLE mode uses the on-chip IDLE Timer (Timer T0) to keep track of elapsed time in the IDLE state. The IDLE Timer runs continuously at the instruction clock rate, whether or not the device is in the IDLE mode. Each time bit 12 of the timer toggles (every 4,096 instruction cycles), the T0PND bit is set, an interrupt is generated (if enabled), and the device exits the IDLE mode if in that mode. If the IDLE timer interrupt is enabled, the interrupt is serviced before execution of the main program resumes.

The IDLE timer cannot be started or stopped under software control, and it is not memory mapped, so it cannot be read or written by the software. Its state upon Reset is

unknown. Therefore, if the device is put into the IDLE mode at an arbitrary time, it will stay in the IDLE mode for somewhere between 1 and 4,096 instruction cycles.

In order to precisely time the duration of the IDLE state, entry into the IDLE mode must be synchronized to the state of the IDLE Timer. The best way to do this is to use the IDLE Timer interrupt, which occurs on every toggle of bit 12 of the IDLE Timer. Another method is to poll the state of the IDLE Timer pending bit T0PND, which is set on the same occurrence.

For more information on the IDLE Timer and its associated interrupt, see the description in Chapter 4.

6.5 HALT/IDLE AND WATCHDOG OPERATION

Some COP888 family members have a Watchdog and Clock Monitor circuit, which requires the software to perform a specific action periodically within a specified time window. The Watchdog feature is described fully in Chapter 8 of this manual.

Operation of the Watchdog is inhibited during the HALT mode. If a single-pin clock is used and the start-up delay has not been enabled, the Watchdog circuit resumes normal operation upon exit from the HALT mode, as if the HALT had not occurred.

If the start-up delay is used, then the IDLE counter is reset at the end of the start-up delay and an IDLE counter interrupt is generated (if enabled). The Watchdog count, which also uses the IDLE timer, is re-started using the previously programmed Watchdog and Clock Monitor settings. Upon resuming operation with the start-up delay, the software should reset the IDLE counter interrupt pending flag (T0PND).

Operation of the Watchdog is also inhibited by the IDLE mode. As in the case of the start-up delay with the HALT mode, the Watchdog count is re-started using the previously programmed Watchdog and Clock Monitor settings.

The software should take into account the re-starting of the Watchdog count upon exit from the HALT mode (with a start-up delay) or IDLE mode. Otherwise, the program could perform its Watchdog servicing routine too early, triggering a Watchdog error.

6.6 NMI EXIT FROM HALT/IDLE

Some COP888 devices have an NMI (non-maskable interrupt) pin. A low-to-high transition on this pin triggers a non-maskable interrupt. If the device is in the HALT or IDLE mode, the NMI brings the device out of that mode. After the NMI interrupt is serviced, normal execution resumes with the instruction following the instruction that placed the device in the HALT or IDLE mode.

Chapter 7

INPUT/OUTPUT

7.1 INTRODUCTION

All COP888 family devices have at least four dedicated input pins ($\overline{\text{RESET}}$, V_{CC}, GND, CKI) and at least one 8-bit I/O port, designated Port G. Additional bidirectional I/O ports, dedicated input ports, and/or dedicated output ports are available on all deices. Refer to the device-specific chapters for information on available ports, packages, and pinouts.

Figure 7-1 is a block diagram illustrating the general structure of bidirectional, dedicated input, and dedicated output pins. A bidirectional I/O pin can be configured by the software to operate as a high-impedance input, an input with a weak pull-up, or as a push-pull output. The operating state is determined by the contents of the corresponding bits in the data register and configuration register. Each bidirectional I/O pin can be used for general-purpose I/O, or in some cases, for a specific alternate function determined by the on-chip hardware. Similarly, a dedicated input pin or dedicated output pin can be used for general-purpose input or output, or for a specific alternate function. (The internal interfaces to alternate functions are not shown in the Figure 7-1.)

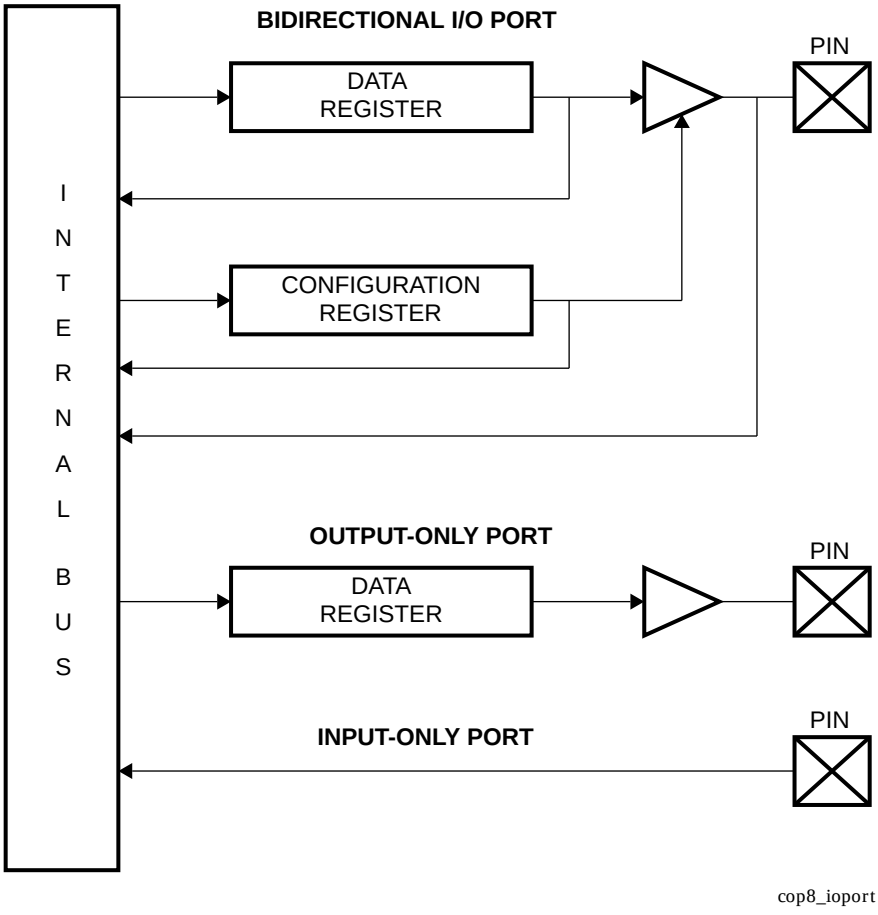


Figure 7-1 COP888 Port Structure

The following sections of this chapter describe each of the individual ports used in COP888 devices. In general, each port contains eight bits. There is usually a memory-mapped data register associated with each port that holds the bit values for the port pins. For bidirectional pins, there is also a configuration register, that specifies whether an I/O pin operates as an input or as an output. Use of the data and configuration registers is described in Section 2.3.3. In some cases, a third memory location is available for reading the state of the port pins directly.

Many pins can be used either for general-purpose I/O or for a specific alternate function. Alternate functions vary from one COP888 device to another, so these functions are described in the device-specific chapters.

Note that not all ports are available in all devices.

7.2 PORT C

Port C is a general-purpose, bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

Any device package that has a Port C, but has fewer than eight Port C pins, contains unbonded, floating pads internally on the chip. For these types of devices, the software should write a 1 to the configuration register bits corresponding to the non-existent port pins. This configures the port bits as outputs, thereby reducing leakage current of the device.

7.3 PORT D

Port D is a general-purpose, dedicated output port. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits.

Port D output pins have high-sink drive capability. Refer to the COP888 data sheets for information on the electrical specifications.

Port D is preset high when the RESET signal goes active (low). If the D2 pin is held low by the external circuit during the Reset state, the microcontroller enters a special mode of operation upon exiting the Reset state. This special mode is used for testing purposes. To avoid entering this mode, the hardware design should ensure that D2 is not pulled low during a Reset operation.

7.4 PORT G

Port G is a bidirectional I/O port that is present in all COP888 devices. The number of pins available in this port depends on the device type and package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs.

Pins G0, G2, G3, G4, and G5 are general-purpose I/O pins, that support alternate functions such as the timer interface control, external interrupt, and MICROWIRE/ PLUS interface. The alternate pin functions are listed in Section 7.7, and described in detail in the chapters devoted to specific COP888 features. For general purposes, these pins are programmed as described in Section 2.3.3.

Pin G1 serves as a dedicated output, WDOUT, if the device has a Watchdog and Clock Monitor. Pin G6, if present, is a dedicated input pin. Pin G7 is either a dedicated input or dedicated output, depending on the oscillator mask option selected. With the R-C oscillator or external clock mask option, G7 can be used as a general-purpose input pin, or as the HALT/Restart input pin as described in Chapter 6. With the crystal oscillator mask option, G7 is the clock output pin, CKO.

7.5 PORT I

Port I is a dedicated input port. The Port I pins are used for general input or for alternate functions such as inputs to a comparator or A/D converter. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high- impedance inputs, that must be pulled to a logic high or low by the external hardware.

Any device package that has a Port I, but has fewer than eight Port I pins, contains unbonded, floating pads internally on the chip. The binary value read from these bits is undetermined. The application software should mask out these unknown bits when reading the Port I register, or use only bit-access program instructions when reading Port I. The unconnected bits draw power only when they are addressed (i.e., in brief spikes).

7.6 PORT L

Port L is a bidirectional I/O port that can be used for general-purpose I/O or for alternate functions such as Multi-Input Wakeup/Interrupt and/or timer I/O. The number of pins available in this port depends on the device type and package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly.

In devices that support the Multi-Input Wakeup/Interrupt feature, the Port L inputs have Schmitt triggers. Refer to the COP888 data sheets for information on the electrical specifications.

7.7 ALTERNATE FUNCTIONS

Many general-purpose I/O pins have alternate functions. The software can program each I/O pin to be used either for general-purpose or for a specific alternate function. The chip hardware determines which of the I/O pins have alternate functions. Alternate functions vary from one COP888 device type to another, and from one package type to another.

Port G is present in all COP888 devices. The alternate functions of this port are described below. The Multi-Input Wakeup/Interrupt feature, an alternate function of some ports, is also described below. This feature can be used for waking up from (exiting) the HALT or IDLE mode, and also for providing an additional eight maskable interrupts. For information on the alternate functions of other ports, refer to the device-specific chapters and the chapters describing individual COP888 features.

7.7.1 Port G Alternate Functions

The pins of Port G have the alternate functions listed below.

- G0 INTR (External Interrupt Input)
- G1 Watchdog Output (on some COP888 devices)
- G2 T1B (Timer T1B input)
- G3 T1A (Timer T1A input/output)
- G4 SO (MICROWIRE Serial Output)
- G5 SK (MICROWIRE Serial Clock)
- G6 SI (MICROWIRE Serial Input)
- G7 If crystal oscillator mask option: Dedicated CKO (Clock Output)

  If R-C/ external oscillator mask option: HALT/Restart input

For more information on interrupts, timers, MICROWIRE/PLUS, and HALT mode, see the separate chapters covering these subjects.

7.7.2 Multi-Input Wakeup/Interrupt

The Multi-Input Wakeup/Interrupt feature is an alternate function of some COP888 ports. This feature can be used for either of two purposes: to provide separate inputs for waking up (exiting) from the HALT or IDLE mode, or to provide separate edge-triggered maskable interrupts. Figure 7-2 is a block diagram showing the internal logic of the Multi-Input Wakeup/Interrupt circuit.

There are three memory-mapped registers associated with this circuit: WKEDG (Wakeup Edge), WKEN (Wakeup Enable), and WKPND (Wakeup Pending). Each register has eight bits, with each bit corresponding to one of the eight input pins shown in Figure 7-2. All three registers are initialized to zero with a Reset.

The WKEDG register establishes the edge sensitivity for each of the port input pins: either positive-going edges (0) or negative-going edges (1).

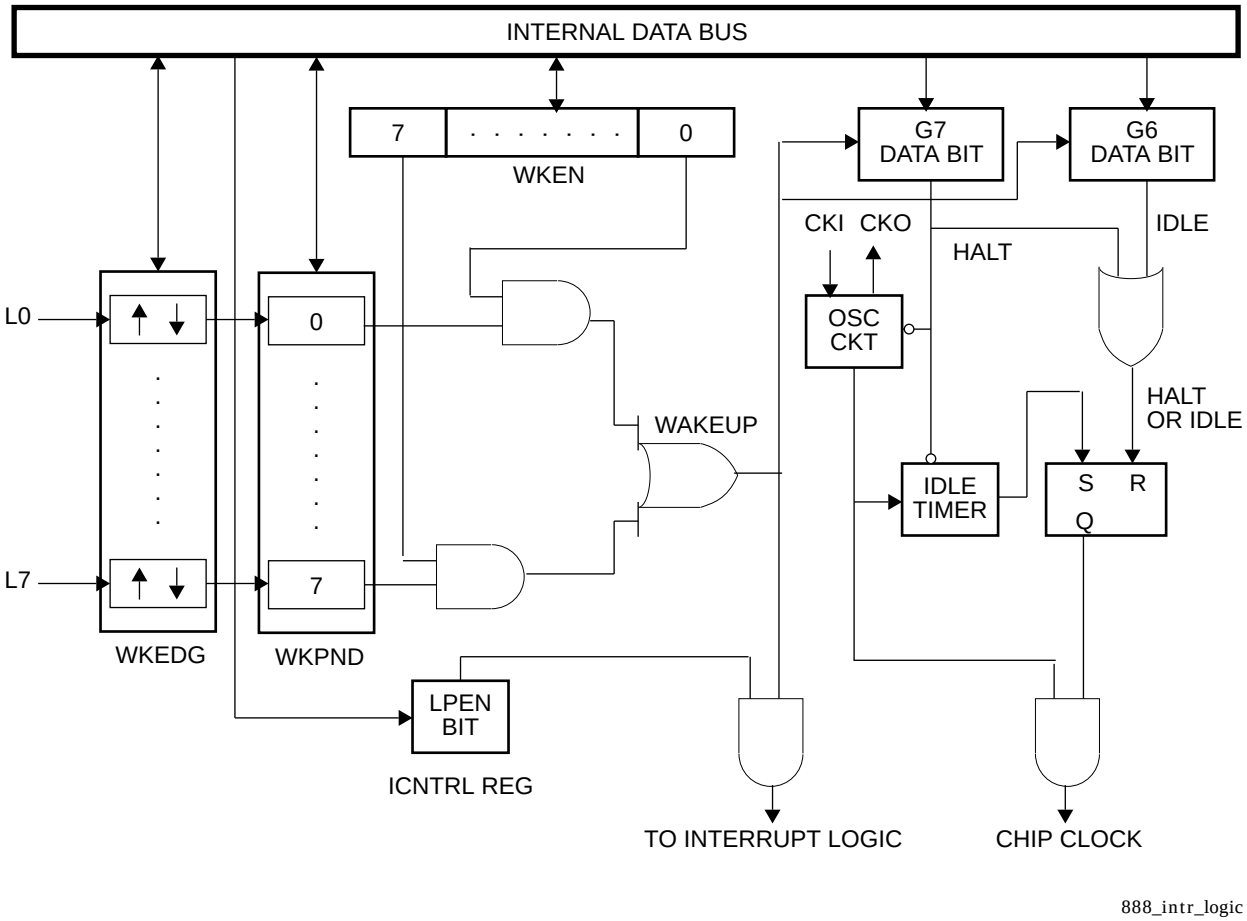


Figure 7-2 Multi-Input Wakeup/Interrupt Logic

The WKEN register enables (1) or disables (0) each of the port pins for the Wakeup/ Interrupt function. Any pin that is to used for the Wakeup/Interrupt function must also be configured as an input pin in its associated configuration register.

The WKPND register contains the pending flags corresponding to each of the port pins (1 for wakeup/interrupt pending, 0 for wakeup/interrupt not pending).

To use the Multi-Input Wakeup/Interrupt circuit, perform the steps listed below. Performing the steps in the order shown will prevent false triggering of a Wakeup/ Interrupt condition. This same procedure should be used following a Reset because the Wakeup inputs are left floating as a result of a Reset, resulting in unknown data on the Port L inputs.

1. If necessary, write to the port configuration register to change the desired port pins from outputs to inputs.
2. Write the WKEDG register to select the desired type of edge sensitivity for each of the pins used.
3. Clear the WKPND register to cancel any pending bits.
4. Set the WKEN bits associated with the pins to be used, thus enabling those pins for the Wakeup/Interrupt function.

Once the Multi-Input Wakeup/Interrupt function has been set up, a transition sensed on any of the enabled pins will set the corresponding bit in the WKPND register. This brings the device out of the HALT or IDLE mode (if in that mode), and also triggers a port maskable interrupt if that interrupt is enabled. To enable the port interrupt, set the enable bit in the appropriate control register (refer to the register bit map sections of the device-specific chapters) and the GIE bit in the PSW register. If the port interrupt is enabled, the microcontroller is vectored to the same port interrupt service routine, regardless of which port pin sensed the interrupt condition. The interrupt service routine can read the WKPND register to determine which pin sensed the interrupt.

The interrupt service routine or other software should clear the pending bit. The COP888 will not enter the HALT or IDLE mode as long as any WKPND pending bit is pending and enabled.

Upon Reset, the WKEDG register is configured to select positive-going edge sensitivity for all Wakeup inputs. If the user wishes to change the edge sensitivity of a port pin, use the following procedure to avoid false triggering of a Wakeup/Interrupt condition.

1. Clear the WKEN bit associated with the pin to disable that pin.
2. Write the WKEDG register to select the new type of edge sensitivity for the pin.
3. Clear the WKPND bit associated with the pin.
4. Set the WKEN bit associated with the pin to re-enable it.

The following example illustrates the program steps for changing the edge sensitivity of port bit 5 from positive to negative edges.

```
RBIT    5,WKEN    ;Disable Port bit 5 for Wakeup/Interrupt
SBIT    5,WKEDG   ;Select negative-going edge sensitivity
RBIT    5,WKPND   ;Clear the pending bit
SBIT    5,WKEN    ;Re-enable the bit for Wakeup/Interrupt
```

For more information on interrupts or the HALT/IDLE power-save modes, see the respective chapters on those subjects.

Chapter 8

WATCHDOG AND CLOCK MONITOR

8.1 INTRODUCTION

Some COP888 devices have a circuit called the Watchdog and Clock Monitor. This circuit monitors the operation of the device and reports an abnormal condition by issuing a signal on the G1 output pin, WDOUT. The device-specific chapters indicate whether the Watchdog and Clock Monitor is present in each COP888 family member.

The Watchdog is a circuit that monitors the number of instruction cycles being executed and detects certain types of program execution errors. The application program must periodically write a specific value to the Watchdog Service Register, within specific time intervals. The Watchdog reports any failure of the software to perform this function. Thus, a runaway program or infinite program loop will result in a Watchdog error. The output signal resulting from an error can be used to reset the chip or to perform any other function.

The Clock Monitor is a circuit that detects the absence of a clock signal, or a clock that is running too slowly. A clock error is reported on the same output pin as a Watchdog error.

Using the Watchdog and Clock Monitor functions is optional. The Clock Monitor function can be disabled by the software following a Reset. The Watchdog function always operates, but the user application can simply ignore the output signal generated on the WDOUT pin.

8.2 WATCHDOG OPERATION

The Watchdog circuit looks at the contents of the IDLE Timer (Timer T0) to keep track of the number of instruction cycles being executed. The IDLE Timer is a 16-bit register that counts down continuously at the instruction clock rate. (The instruction clock runs at one-tenth the frequency of the chip clock.) The IDLE Timer is not memory-mapped, and cannot be read or written by the software.

The application software must write a specific value at periodic intervals into the Watchdog Service Register (WDSVR). This must happen within a “time window”: no more often than once every 2048 instruction cycles, but at least as often as once every 65536 cycles initially. The lower limit is fixed at 2048 instruction cycles, but the upper limit can be programmed to be either 8192, 16384, 32768 or 65536 instruction cycles. Programming of the upper limit can be performed only once following a chip Reset.

Figure 8-1 is a block diagram showing the logic of the Watchdog circuit.

The Watchdog Service Register, WDSVR, is an 8-bit memory-mapped register. The software “services” this register at periodic intervals by writing a specific value to it. If the written value is correct, the “Valid Service” signal is asserted; if the value is not correct, it is considered an “Invalid Service,” which triggers a Watchdog error.

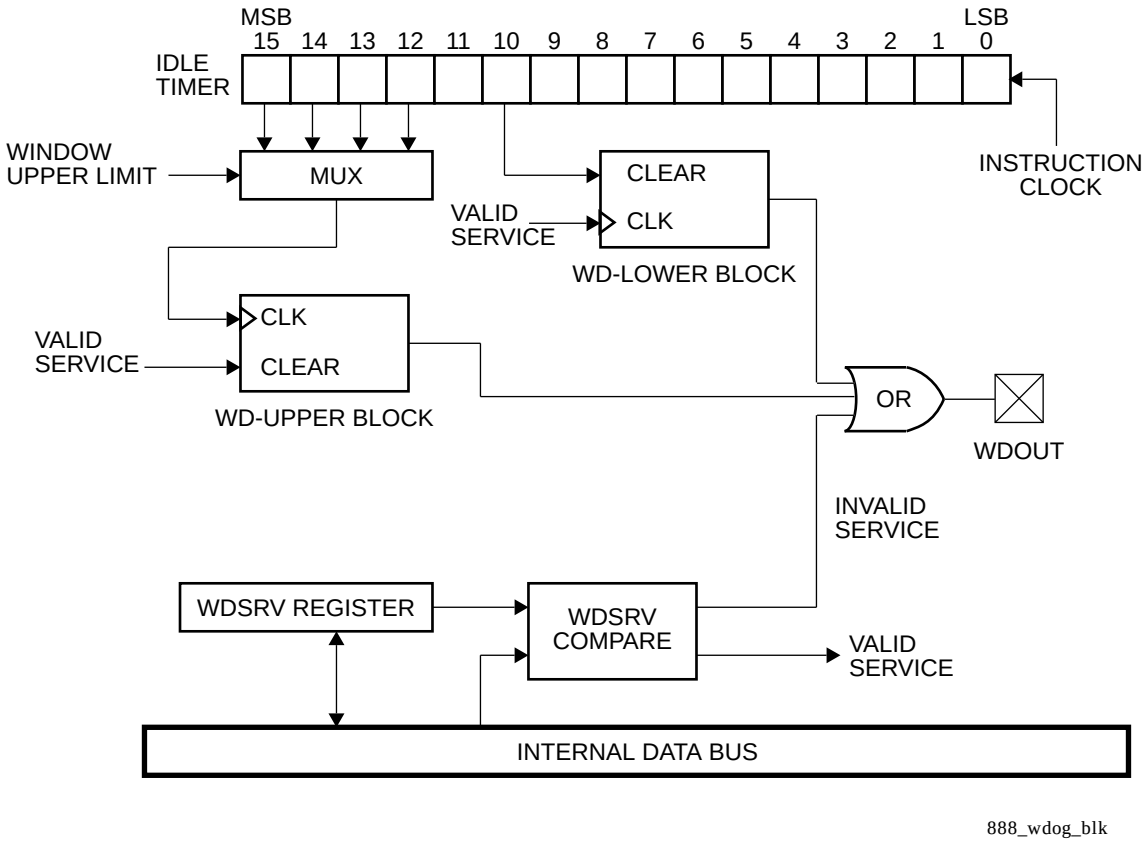


Figure 8-1 Watchdog Logic Block Diagram

The Watchdog logic contains two internal logic blocks called WD-Upper and WD-Lower. The WD-Upper logic block establishes the upper limit of the time window, and the WD-Lower logic block establishes the lower limit. The WD-Upper block can be programmed to use any of the four most significant bits of the IDLE Timer register to clock its counter, thus establishing the upper limit at 8192, 16384, 32768, or 65536 instruction cycles. The WD-Lower logic block uses bit 10 of the IDLE Timer register to clear its counter, thus establishing a fixed lower limit of 2048 instruction cycles.

Each of the two logic blocks contains a 1-bit counter. If either counter overflows (is clocked twice without being cleared), a Watchdog error is reported on WDOUT. In the WD-Upper block, the counter is clocked by an IDLE Timer bit at intervals of 8192, 16384, 32768, or 65536 instruction cycles. The counter is cleared by a valid servicing of WDSVR. Thus, the Watchdog Service Register must be serviced at least as often as the programmed upper limit. In the WD-Lower block, the counter is clocked by a valid servicing of WDSVR, and cleared by bit 10 of the IDLE Timer register every 2048 instruction cycles. Thus, the Watchdog Service Register must be serviced no more than once every 2048 instruction cycles. If WDSVR is serviced not often enough for the WD-Upper block or too often for the WD-Lower block, an overflow occurs and an error is reported.

The specific value that must be written periodically to the WDSVR register is explained in Section 8.4.

Operation of the Watchdog circuit is inhibited during the HALT and IDLE modes. The contents of the IDLE timer are affected by the HALT mode under certain conditions (i.e., when a start-up delay is used) and by the IDLE mode. Upon exit from the HALT or IDLE mode in these cases, the on-chip hardware services the Watchdog circuit automatically. Therefore, upon exit from the HALT or IDLE mode, the application software should service the Watchdog Service Register within the established upper limit of 8192, 16384, 32768, or 65536 instruction cycles as usual, but not before 2048 instruction cycles. Otherwise, a Watchdog error may be triggered.

For the HALT mode, the re-starting of the Watchdog count only occurs if a start-up delay is used. If a start-up delay is not used, the Watchdog service window resumes counting normally upon exit from the HALT mode, as if the HALT had never occurred. For more information on the IDLE mode, HALT mode, and HALT mode start-up delay, see Chapter 6. For more information on the IDLE Timer, see Chapter 4.

8.3 CLOCK MONITOR OPERATION

The Clock Monitor constantly checks the chip clock and issues an error signal on the WDOUT pin if the clock is not running, or is running very slowly. Upon Reset, the microcontroller starts out with the Clock Monitor enabled. If the clock is not operating properly at that time, an error signal is issued on the WDOUT pin. If the clock fails to operate correctly at a later time, an error signal is issued when the error is detected. The Clock Monitor works even in the HALT and IDLE modes. Therefore, any entry into the HALT mode is reported as a clock error.

The Clock Monitor circuit monitors the instruction clock, which runs at one-tenth the frequency of the chip clock. An instruction clock running at 10 KHz or higher is interpreted as normal operation. An instruction clock running at 10 Hz or slower is considered too slow, and is reported as an error. The pass/fail threshold frequency can vary within the range of 10 Hz and 10 KHz, so an instruction clock running in that range may or may not trigger an error.

Following a Reset, the Clock Monitor function can be disabled by the software. Configuration of the Watchdog and Clock Monitor functions can be performed only once following a Reset, as explained below.

8.4 CONFIGURATION

The Watchdog and Clock Monitor circuits are inhibited during a Reset (with the $\overline{\text{RESET}}$ signal active), and begin operating as soon as the microcontroller comes out of a Reset. Initially, the Watchdog operates with the service window set to the maximum upper limit of 65,536 instruction cycles. The Clock Monitor will detect a non-functioning clock immediately following the Reset, so the active $\overline{\text{RESET}}$ signal should be made long enough to allow the clock circuit to reach its full amplitude and a stable operating frequency.

The Watchdog and Clock Monitor functions can be configured only once following a chip Reset. Once configured, they cannot be configured again unless there is another Reset. They are configured by writing a data byte to the Watchdog Service Register, WDSVR, a

read/write register residing at address C7 Hex. The format of the data byte is shown in Figure 8-2.

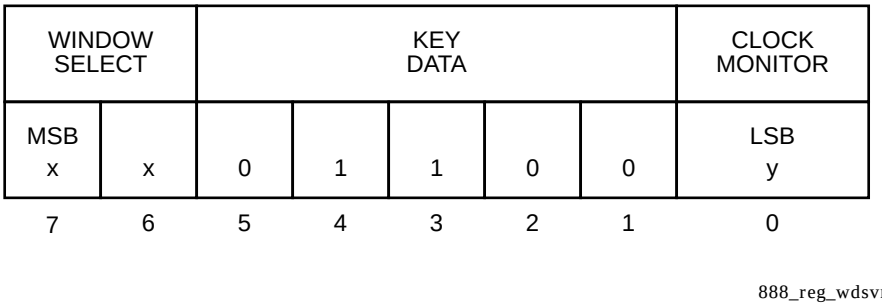


Figure 8-2 Watchdog Service Register (WDSVR) Format

The data byte written to WDSVR for the first time following a Reset configures the Watchdog circuit and the Clock Monitor. To avoid a Watchdog error, this must be done no later than 65536 instruction cycles after the $\overline{\text{RESET}}$ signal goes inactive. However, it may be done within the first 2048 instruction cycles without triggering a Watchdog error. Immediately following Reset is the recommended time.

The first time WDSVR is written, the two most significant bits are used to select the upper limit of the Watchdog service window, as indicated in the table below.

WDCNT		**Upper Limit of Service Window**
Bit 7	**Bit 6**	
0	0	8,192 Instruction Cycles
0	1	16,384 Instruction Cycles
1	0	32,768 Instruction Cycles
1	1	65,536 Instruction Cycles

Bits 5-4-3-2-1 of WDSVR must always be written with the binary value 01100. This is a fixed code that cannot be changed. If any other value is written at any time, a Watchdog error is triggered.

The first time WDSVR is written, the least significant bit controls the Clock Monitor feature. If a 1 is written, the Clock Monitor remains enabled. If a 0 is written, the Clock Monitor is disabled.

The Watchdog and Clock Monitor can be programmed only once following a Reset. The exact same 8-bit value used for programming must be written to the WDSVR register at each Watchdog service. Any attempt to write a different value to WDSVR after the first time will trigger a Watchdog error, and will not affect the Watchdog or Clock Monitor configuration.

The WDSVR register can be read as well as written. The values read back from bits 7, 6, and 0 reflect the values that were written to that register at the time of programming (the first time following Reset). However, bits 5 through 1 always read back as all zeros. The code bits written to the register (01100) cannot be read back.

8.5 ERROR REPORT ON WDOUT

An abnormal condition detected by the Watchdog or Clock Monitor is reported by issuing a signal on the G1 output pin, WDOUT. In devices that have the Watchdog and Clock Monitor, the G1 pin operates independently from the Port G configuration and data registers, allowing bit 1 of those registers to be used as independent software flags.

Under normal conditions, the WDOUT pin remains in the high-impedance state. When a Watchdog error is detected, the pin is pulled low for a period ranging from 16 to 32 instruction cycles, after which it returns to the high-impedance state. During this time that WDOUT is low, any Watchdog service (writing to WDSVR) is ignored. After a Watchdog error, the Watchdog service window restarts when the WDOUT pin returns to the high-impedance state.

In the case of a Clock Monitor error, the WDOUT pin remains in the low state as long as the clock is not operating properly. When the clock returns to normal operation, the WDOUT pin returns to the high-impedance state after a delay of 16 to 32 instruction cycles.

The WDOUT pin can be used in a variety of ways. It can be simply ignored, in which case there is no need to service the Watchdog Service Register. If WDOUT is used, a pullup resistor is needed to pull the pin high during normal operation. When a Watchdog or Clock Monitor error is detected, the pin goes low. This signal can be used to reset the microcontroller or the whole system. In that case, the circuit hardware should ensure that the $\overline{\text{RESET}}$ signal generated from WDOUT going low meets the system requirements. Note that the WDOUT signal itself will go low for only 16 to 32 instruction cycles for a Watchdog error.

The state of the WDOUT pin is unknown immediately following a Reset. If it starts out active-low, it will return to the high-impedance state within 16 to 32 instruction cycles if there is no Watchdog or Clock Monitor error.

Chapter 9

UART

9.1 UART

Some COP888 devices have an on-chip Universal Asynchronous Receiver Transmitter (UART), which can be used for transmitting and receiving data serially, either synchronously or asynchronously. This full-duplex, double-buffered UART is fully programmable, and can be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (7/8, 1, 1-7/8, or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)
- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability
- Error detection circuitry and diagnostic self-test capability

9.1.1 UART Operation Overview

Figure 9-1 is a block diagram of the COP888 UART circuit. There are three major sections to the circuit: the receiver, transmitter, and control sections.

The receiver section receives serial data on the RDX pin. Serial data bits are shifted into the Receive Shift register, low-order bit first. When a full byte is received, it is transferred to the receive buffer (RBUF), a memory-mapped register, and the receive buffer full flag (RBFL) is set. If the UART receiver interrupt is enabled, an interrupt is also generated. The software routine reads and processes the byte in RBUF.

The transmitter section sends out serial data on the TDX pin. When the transmit buffer register (TBUF) is empty, the transmit buffer empty flag (TBMT) is set. If the UART transmitter interrupt is enabled, an interrupt is also generated on the buffer empty condition. The software then writes the next byte to be transmitted into the transmit buffer register (TBUF), a memory-mapped register. At the appropriate time, the data

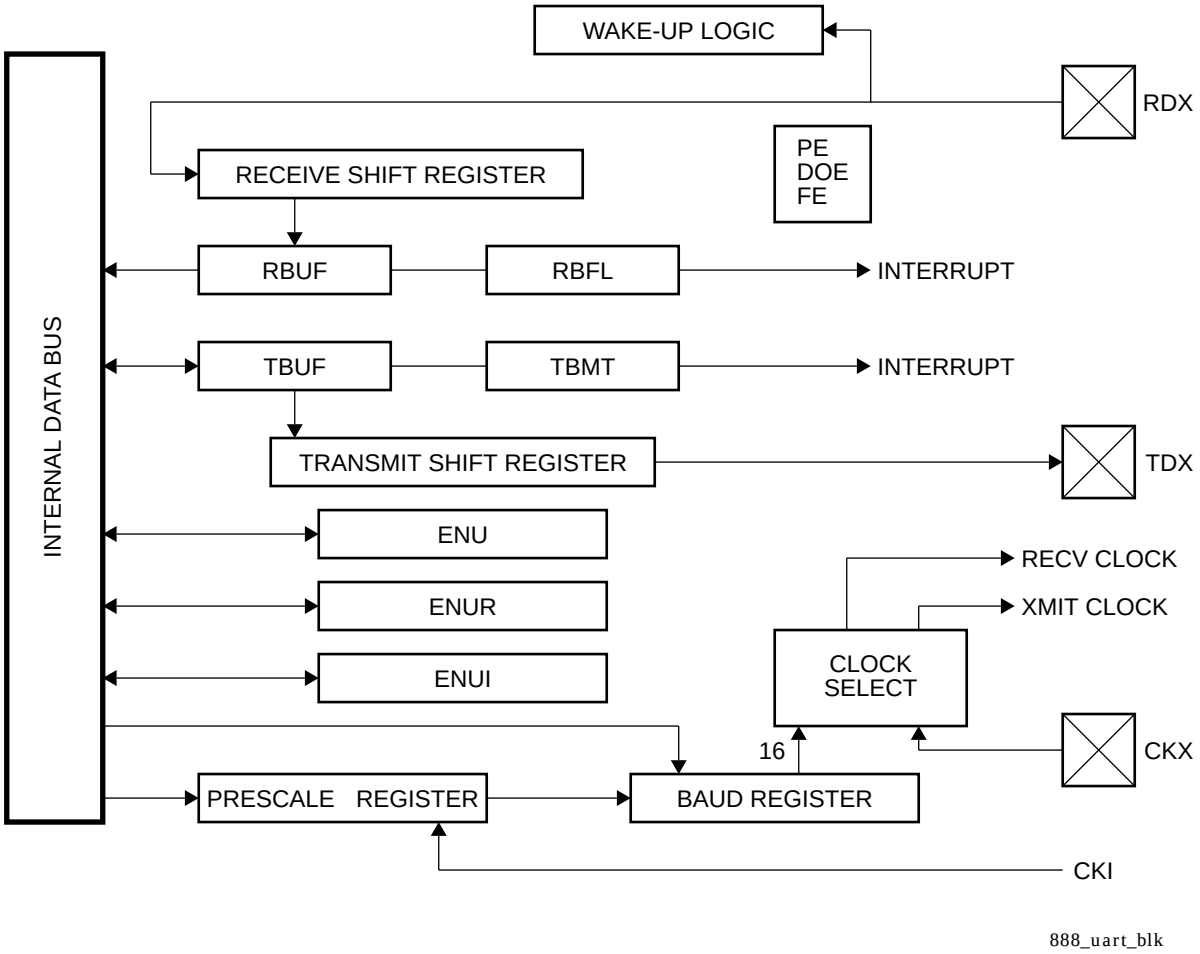


Figure 9-1 UART Block Diagram

byte is transferred from TBUF into the Transmit Shift register. Serial data bits are shifted out of the Transmit Shift register on the TDX pin, low-order bit first.

The software controls the UART using three memory-mapped registers: the Control and Status Register (ENU), the Receive Control and Status Register (ENUR), and the Interrupt and Clock Source Register (ENUI). Some bits in these registers are control bits, while others are status bits. Two more memory-mapped registers are used to select the basic UART baud clock rate: the Prescaler (PSR) and Baud (BAUD) registers. The clock source for generating the baud clock can be either the chip clock (CKI) or an external clock signal received on the CKX pin.

9.1.2 UART Registers

The UART contains seven memory-mapped registers, located in data memory at addresses xxB8 through xxBE Hex. Three registers (ENU, ENUR, and ENUI) are used for configuring the UART and two registers (PSR and BAUD) are used for setting the baud rate. The UART transmitter section contains two linked registers, the Transmit Buffer (TBUF) and Transmit Shift registers. The UART receiver section also contains two linked registers, the Receive Shift and Receive Buffer (RBUF) registers. The two shift registers are not memory mapped, and cannot be accessed directly by the software.

The bit maps for the control and status registers ENU, ENUR, and ENUI, and the baud selection registers PSR and BAUD, are shown in Tables 9-1 to 9-5.

Table 9-1 ENU, UART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN:	Parity enable bit (0 = Parity disabled, 1 = Parity Enabled
PSEL1:	Parity selection bit (with PSEL0), as indicated below:
	PSEL1-PSEL0 = 00 Odd parity, if parity is enabled
	PSEL1-PSEL0 = 01 Even parity, if parity is enabled
	PSEL1-PSEL0 = 10 Mark (1), if parity is enabled
	PSEL1-PSEL0 = 11 Space (0), if parity is enabled
XBIT9/PSEL0:	For nine data bits per frame, this bit is XBIT9, or Transmit Bit 9, the ninth data bit transmitted. For seven or eight data bits per frame, this bit serves as PSEL0 (see PSEL1 above).
CHL1-CHL0:	Frame format selection bits (number of data bits):
	CHL1-CHL0 = 00 Eight data bits per frame.
	CHL1-CHL0 = 01 Seven data bits per frame.
	CHL1-CHL0 = 10 Nine data bits per frame (using XBIT9, RBIT9).
	CHL1-CHL0 = 11 Diagnostic Loopback Mode. Transmitter output is internally looped back to receiver input. Nine-bit framing is used.
ERR:	Error flag. Set upon occurrence of any DOE, FE, or PE error.
RBFL:	Receive buffer full flag. This bit is set when the UART has received a complete byte and has copied it into the RBUF register. The bit is reset automatically when the software reads from the RBUF register.
TBMT:	Transmit buffer empty flag. This bit is set when the UART transfers a byte of data from the TBUF register into the Transmit Shift register for transmission. The bit is reset automatically when the software writes into the TBUF register.

Table 9-2 ENUR, UART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	Reserved	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag; set by a Data Overrun error, cleared by reading the ENUR register
FE:	Framing Error flag; set by a Framing error, cleared by reading the ENUR register
PE:	Parity Error flag; set by a Parity error, cleared by reading the ENUR register
RBIT9:	Receive bit 9; ninth data bit received when UART operates with nine bits per frame
ATTN:	Attention mode enable bit, cleared automatically upon receiving a character with data bit nine set
XMTG:	UART transmitting; set to indicate the transmitter is transmitting, reset at end of last frame
RCVG:	UART receiving error; set to indicate a framing error, reset when RDX goes high

Table 9-3 ENUI, UART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	STP78	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2:	Number of Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)
STP78:	Last Stop bit length (0 = 1 bit, 1 = 7/8 bit)
ETDX:	Enable TDX transmit pin, an alternate function of Port L pin L2 (0 = TDX pin, 1 = Port L pin).
SSEL:	Synchronous select bit (0 = asynchronous, 1 = synchronous)
XRCLK:	External Receive Clock; selects clock source for receiver section (0 = internal baud rate generator clock, 1 = external CKX clock)
XTCLK:	External Transmit Clock; selects clock source for transmitter section (0 = internal baud rate generator clock, 1 = external CKX clock)
ERI:	Enable Receive Interrupt (0 = disable, 1 = enable interrupts from the receiver section)
ETI:	Enable Transmit Interrupt (0 = disable, 1 = enable interrupts from the transmitter section)

Table 9-4 BAUD, UART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0
BRD7–BRD0: Baud rate divisor (with PSR register)							

Table 9-5 PSR, UART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8
PSR4-PSR0: Baud rate prescaler							
BRDA-BRD8: Baud rate divisor (with BAUD register)							

Data Registers

The transmit section contains a pair of linked registers TBUF (Transmit Buffer) and TSFT (Transmit Shift). The TBUF and TSFT registers double-buffer data for transmission, with data shifting out low-order bit first from TSFT to the TDX output. The next byte for transmission is loaded into TBUF by the software while the previous byte is shifting out of TSFT. The next byte from TBUF is automatically loaded into TSFT once the previous byte has shifted out. TSFT can not be read or written by the software.

The receive section contains a pair of linked registers RSFT (Receive Shift) and RBUF (Receive Buffer). The RSFT and RBUF registers double-buffer data being received, with data shifting in low-order bit first from RDX to the RSFT input. While the next byte is shifting into RSFT, the previous byte received is read from RBUF by the software. The next byte from RSFT is automatically loaded into RBUF once the byte has finished shifting into RSFT. Note that RBUF is a read only register. RSFT can not be read or written by the software.

Prescaler and Baud Select Registers

The UART baud clock selection is programmed through the two registers PSR (Prescaler Select) and BAUD (Baud Select). The Prescaler factor is selected by the upper five bits of the PSR register, while the baud rate divisor is determined by the lower three bits of the PSR register in conjunction with the eight bits of the BAUD register. This allows an 11-bit baud rate divisor, ranging from 1 to 2048. Note that the programmed value is equal to the baud rate divisor minus one, which ranges from 0 to 2047. An all zero prescaler value is reserved for selecting NO CLOCK.

Control and Status Registers

The operation of the UART is programmed through the three registers ENU, ENUR and ENUI. All bits in these registers are cleared as a result of a reset with the exception of the TBMT (Transmit Buffer Empty) bit in the ENU register which is initialized high.

Several of the bits in the ENU and ENUR registers are read-only and cannot be written by software. These read-only bits include ERR, RBFL and TBMT in the ENU register and RBIT9, XMTG, RCVG, DOE, FE and PE in the ENUR register. The DOE, FE and PE bits of the ENUR register are cleared automatically when the ENUR register is read. Note that bit manipulation instructions (test bit, set bit, reset bit) or comparison instructions (if equal, if not equal, if greater than) with the ENUR register do not clear the error flags. Moreover, a reset bit instruction on the error flag itself does not clear the error flag, since these error flags are read-only bits.

9.1.3 UART Interface

Port L pins L1, L2, and L3 are used for the UART interface. These three pins are used for CKX (clock), TDX (transmit), and RDX (receive), respectively. RDX is an inherent function associated with the L3 pin and requires no setup following reset, except for ensuring that the associated L3 data register and configuration register bits both remain reset in order to select L3 as an input. The TDX function is assigned to the L2 pin by setting the ETDX bit in the ENUI control register and setting the associated L2 configuration bit in order to configure L2 as an output pin. Once the ETDX bit is set, the associated L2 data register bit may be used as a software flag, since the TDX output from the Transmit Shift register is connected to the L2 pin directly.

The baud rate clock for the UART can be generated on-chip, or can be selected from an external source. The L1 pin is used as the external clock I/O pin (CKX) if either the XTCLK or XRCLK bits in the ENUI control register are set. If neither of these control bits is set, then L1 serves as a normal I/O pin. The CKX pin can be used as either an input or output, as determined by the associated L1 configuration register bit. As an input, CKX represents an external clock input, which may be selected to drive the UART transmitter and/or receiver. As an output, CKX represents the internal baud rate generator clock output.

9.1.4 Asynchronous Mode

The asynchronous mode is selected by resetting the SSEL bit to zero in the ENUI register. The input frequency to the UART is 16 times the baud rate. The Transmit Shift and TBUF registers double-buffer data for transmission. While the Transmit Shift register is shifting out the current character on the TDX pin, the TBUF register may be loaded by software with the next byte to be transmitted. When the Transmit Shift register finishes transmitting the current character, the contents of TBUF are transferred to the Transmit Shift register and the Transmit Buffer Empty flag (TBMT in the ENU register) is set. The TBMT flag is automatically reset by the UART when software loads a new character into the TBUF register. There is also the XMTG bit, which is set to indicate that the UART is transmitting. This bit is reset at the end of the last frame (end of last Stop bit). Figure 9-2 shows the transmitter timing diagram.

The Receive Shift and RBUF registers double-buffer data being received. The UART receiver continually monitors the signal on the RDX pin for a low level to detect the beginning of a Start bit. Upon sensing this low level, it waits for half a bit time and samples again. If the RDX pin is still low, the receiver considers this to be a valid Start bit, and the remaining bits in the character frame are each sampled a single time, at the mid-bit position. Serial data received on the RDX pin is shifted into the Receive Shift

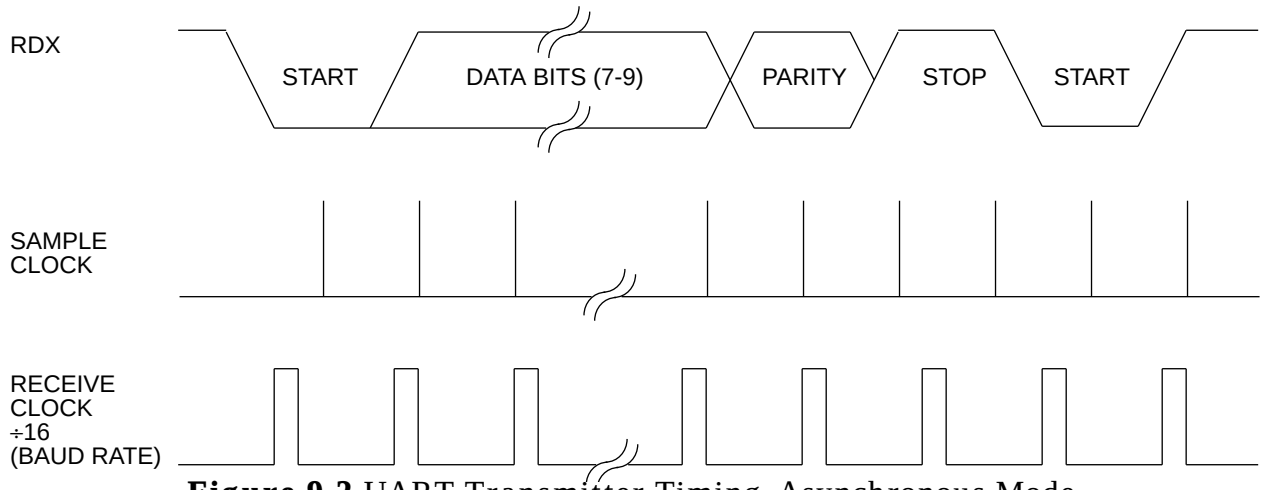
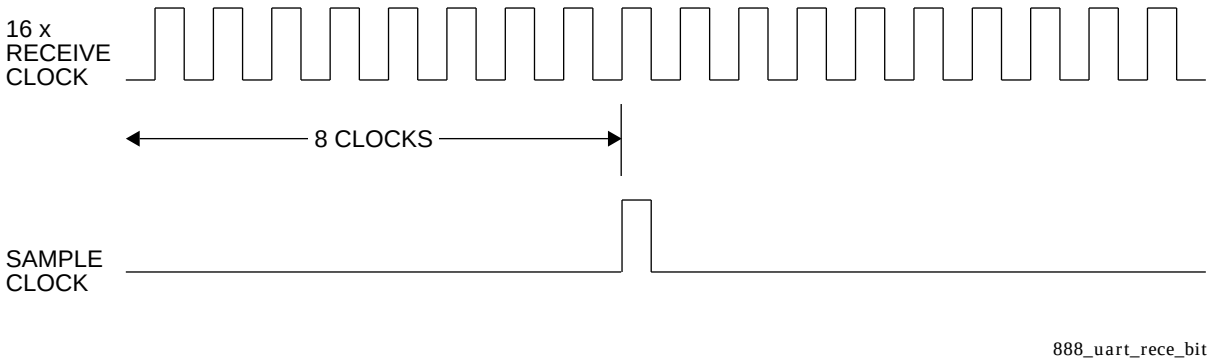


Figure 9-2 UART Transmitter Timing, Asynchronous Mode

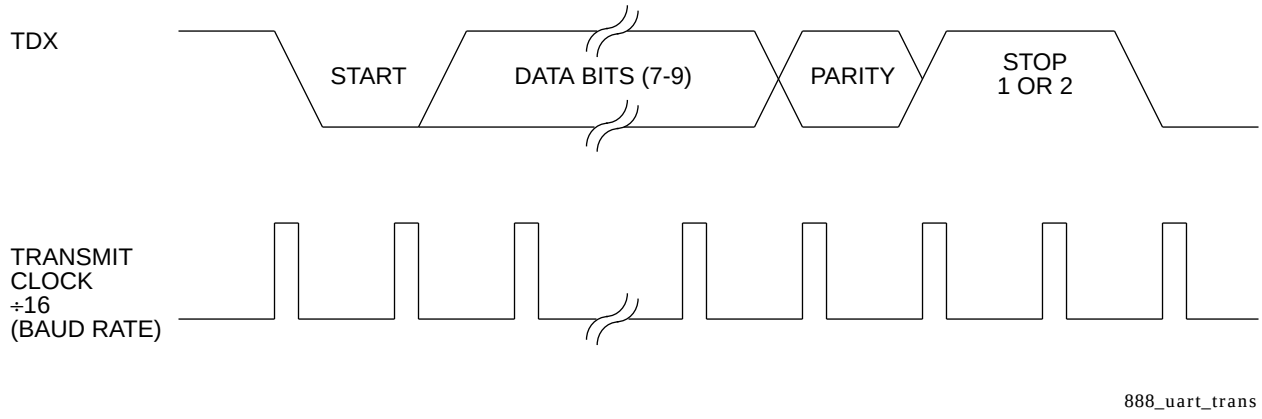
888_uart_rece_tim

register. Upon receiving the complete character, the contents of the Receive Shift register are copied into the RBUF register and the Receive Buffer Full flag (RBFL) is set. RBFL is automatically reset when software reads the character from the RBUF register. RBUF is a read-only register. There is also the RCVG bit, which is set high whenever a framing error occurs. The RCVG bit is reset whenever RDX goes high. TBMT, XMTG, RBFL and RCVG are read-only bits. Figures 9-3 and 9-4 show the timing diagrams for the receiver.



888_uart_rece_bit

Figure 9-3 UART Receiver Bit Sampling, Asynchronous Mode



888_uart_trans

Figure 9-4 UART Receiver Timing, Asynchronous Mode

The clock source for the transmitter and/or receiver can be selected to come from an external source (at the CKX pin) or from the internal baud rate generator. If the internal baud rate generator is used, the internal clock can be output to the CKX pin.

9.1.5 Synchronous Mode

The synchronous mode is selected by setting the SSEL, XTCLK, and XRCLK bits to all ones in the ENUI register. In this mode, data is transmitted on the rising edge and received on the falling edge of the synchronous clock on the CKX pin.

The input frequency to the UART is the same as the baud rate on the CKX pin. If data transmit and receive are selected with the CKX pin as the clock output, the microcontroller generates the synchronous clock output at the CKX pin. The internal baud rate generator is used to produce the synchronous clock. Data transmit and receive are performed synchronously with this clock through the TDX and RDX pins. Note that if CKX is selected as an output pin, then the selected clock from the internal Baud Rate Generator is divided by 2 before being output to the CKX pin. This results in a 50 percent duty cycle on the CKX clock. Figure 9-5 shows the timing diagram for the synchronous mode.

When an external clock input is selected at the CKX pin, data transmit and receive are performed synchronously with this clock through the TDX and RDX pins.

9.1.6 Framing Formats

The UART supports several framing formats as shown in Figure 9-6. The format is selected by writing certain control bits in the ENU and ENUI registers. A framing format consists of a Start bit followed by seven, eight, or nine data bits (excluding parity), followed by an optional parity bit, followed by 7/8, 1, 1-7/8, or 2 Stop bits. In applications using parity, the parity bit is generated and verified by hardware. The optional parity bit may be selected as either odd or even parity, a mark, or a space. No parity is possible with the nine-bit format. Note that the 7/8 option, which selects the last Stop bit to be 7/8 of a bit in length, is independent of whether one or two Stop bits are selected.

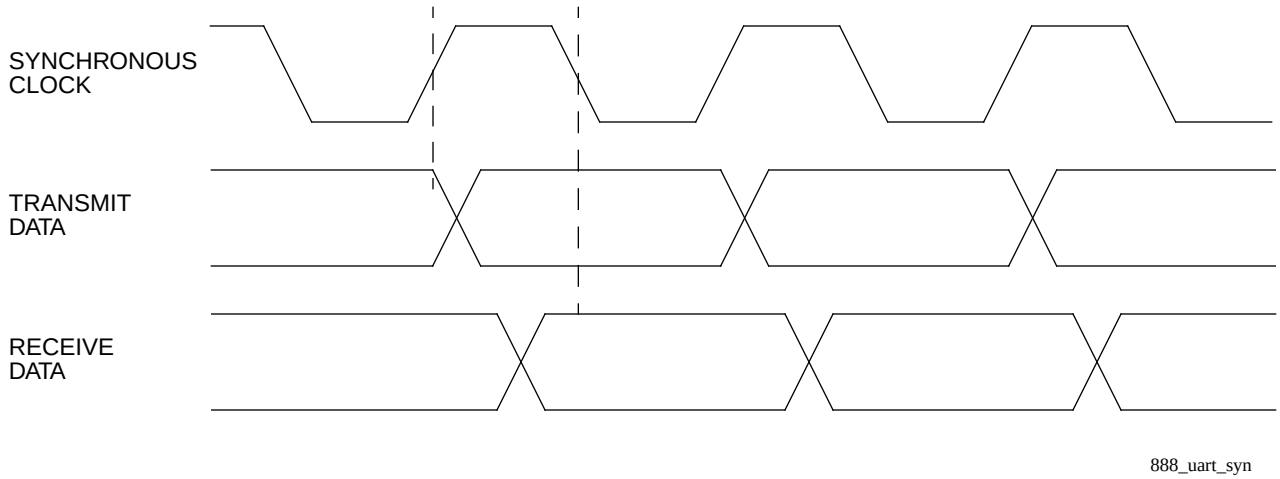


Figure 9-5 UART Synchronous Mode Timing

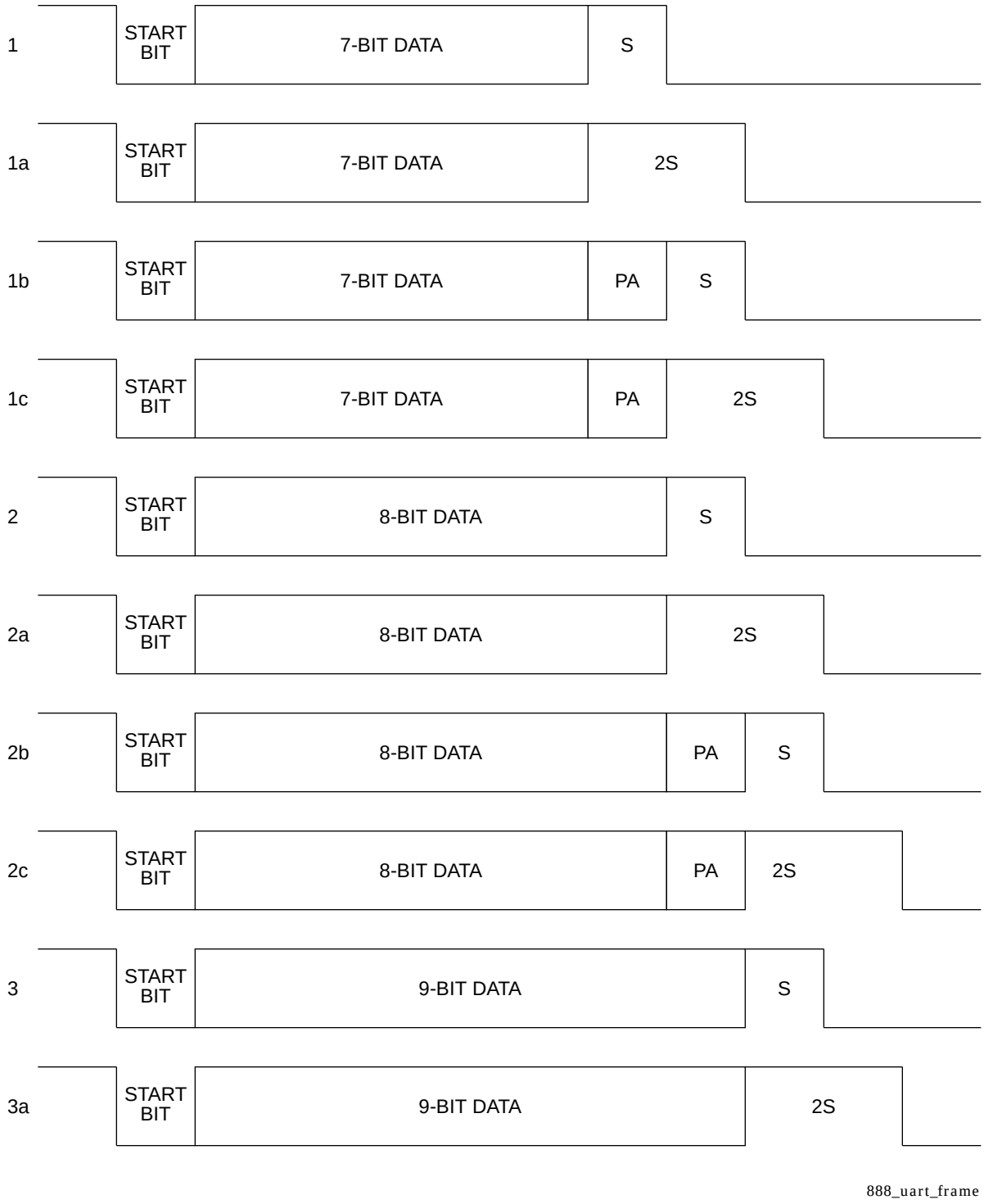


Figure 9-6 UART Framing Formats

The number of data bits (7, 8, or 9) is selected with the CHL0 and CHL1 bits in the ENU register. The first data transmission format shown in Figure 9-6 (1, 1a, 1b, 1c) consists of a Start bit, seven data bits, an optional parity bit, and 7/8, 1, 1-7/8, or 2 Stop bits. This format is selected with CHL1 = 0, CHL0 = 1.

The second data transmission format (2, 2a, 2b, 2c) consists of a Start bit, eight data bits, an optional parity bit, and 7/8, 1, 1-7/8, or 2 Stop bits. This format is selected with CHL1 = 0, CHL0 = 0. The optional parity bit with these first two formats is generated and verified by hardware. The parity bit is enabled or disabled by the PEN (Parity Enable) bit in the ENU register. If parity is enabled (PEN = 1), the parity selection is made with the PSEL0 and PSEL1 bits of the ENU register. The parity selections include odd parity (PSEL1 = 0, PSEL0 = 0), even parity (PSEL1 = 0, PSEL0 = 1), mark (PSEL1 = 1, PSEL0 = 0), and space (PSEL1 = 1, PSEL0 = 1). The mark consists of a 1 bit, while the space consists of a 0 bit.

The third data transmission format (3, 3a) consists of a Start bit, nine data bits, and 7/8, 1, 1-7/8, or 2 Stop bits. This format is selected with CHL1 = 1, CHL0 = 0. Parity is not generated or verified with this format. The UART Attention mode feature is supported by this 9-data-bit format. When operating in this format, all eight bits of TBUF and RBUF are used for data. The ninth data bit is transmitted and received using the two bits XBIT9 and RBIT9, located in the ENU and ENUR registers, respectively. Note that RBIT9 is a read-only bit.

The last Stop bit can be programmed to be 7/8 of a bit in length with any of the transmission formats. If two Stop bits are selected with the 7/8 bit also selected, then the second Stop bit is 7/8 of a bit in length. The 7/8-bit select (STP78) and the two-Stop-bit select (STP2) control bits are both located in the ENUI control register.

Note that the XBIT9/PSEL0 bit located in the ENU register serves two mutually exclusive functions. This bit programs the ninth bit for transmission when the UART is operating with the nine-data-bit framing format selected. There is no parity selection with this framing format. For the other framing formats, XBIT9 is not needed, so this same bit is defined instead as the PSEL0 bit, which is used in conjunction with the PSEL1 bit to select the type of parity (odd, even, mark, space) if parity generation is enabled.

The framing formats for the receiver only differ from those for the transmitter in the number of Stop bits required. The receiver requires only one Stop bit in a frame, regardless of the setting of the Stop-bit selection bits in the ENUI control register. Note that an implicit assumption is made for full-duplex UART operation that the framing formats are the same for the transmitter and receiver.

9.1.7 Reset Initialization

All bits in the UART Control and Status registers ENU, ENUR, and ENUI are cleared as a result of RESET, with the exception of the TBMT (Transmit Buffer Empty) bit in the ENU register, which is initialized high.

The PSR register is cleared as a result of RESET, while the Transmit Shift register is initialized high to all ones. The reason for initializing the Transmit Shift register high is to provide a high-to-low transition on the TDX output to signify a start bit once the UART

has started. Note that the Transmit Shift and Receive Shift registers are non-addressable registers, while the RBUF register is a read-only register.

The TBUF, RBUF, Receive Shift, and BAUD registers are not initialized with RESET.

9.1.8 HALT/IDLE Mode Reinitialization

The UART is initialized as a result of reset and is initialized again whenever the microcontroller enters the HALT or IDLE mode. Note that a HALT or IDLE mode reinitialization is a subset of the RESET initialization, in that much of the UART control selection (baud rate, framing format, etc.) is left unchanged following HALT or IDLE.

The HALT/IDLE mode reinitialization is the same as the reset initialization described above, with the exception of the three UART control and status registers ENU, ENUR, and ENUI. All of the read/write bits in these three registers are left unchanged as a result of HALT/IDLE mode reinitialization. The read-only bits ERR, RBFL, DOE, FE, PE, RBIT9, XMTG, and RCVG are cleared as a result of the HALT/IDLE mode reinitialization, while the TBMT (Transmit Buffer Empty) read-only bit is initialized high.

In summary, the HALT/IDLE mode reinitialization is identical to the reset initialization, with the exception of the read/write bits in the ENU, ENUR, and ENUI registers. These read/write bits are cleared with reset initialization, but are left unchanged with HALT/IDLE mode reinitialization.

The microcontroller may exit from the HALT/IDLE mode when the Start bit of a character is detected at the RDX input (pin L3 of Port L). This feature is activated by using the Multi-Input Wakeup selected for pin L3. Note that the COP888 microcontroller also exits the IDLE mode whenever the thirteenth bit of the IDLE counter (T0) toggles.

The application program must set up the RDX Start bit wakeup on pin L3 before entering the HALT/IDLE mode. This section of the program consists of the following steps:

```
RBIT 3,   WKEN

SBIT 3,   WKEDG

RBIT 3,   WKPND

SBIT 3,   WKEN
```

The Wakeup trigger condition for an RDX start bit is programmed as a high-to-low transition by setting bit 3 of the WKEDG register. Bit 3 of the WKPND register is cleared to eliminate any previous RDX transition still pending, followed by setting bit 3 of the WKEN register to enable the Wakeup.

If a crystal or resonator closed loop oscillator is used with the microcontroller, the IDLE timer (T0) is used to generate a fixed delay following the HALT mode. Crystals and resonators require a certain amount of startup time to reach full amplitude and frequency stability. The fixed delay from the IDLE timer following HALT is necessary to ensure that the oscillator has indeed stabilized before the microcontroller is allowed to execute program code. The program must consider this fixed delay when a UART data transfer is expected immediately following the HALT mode.

9.1.9 Baud Clock Generation

The clock inputs to the transmitter and receiver sections of the UART can be individually selected to come from either an external source on the CKX pin (pin L1 of the Port L) or a source selected by dividing the CKI clock by the Baud Rate Generator. The Baud Rate Generator consists of the PSR (prescaler) and BAUD registers along with the associated selection circuitry. The clock input selection for the UART Transmitter and Receiver sections is shown in Figure 9-7. The selection of CKX versus BRG (Baud Rate Generator) is programmed using the XTCLK and XRCLK bits in the ENUI control register. Tables 9-6 and 9-7 show the UART clock source selection relative to the XTCLK and XRCLK bits for the Asynchronous and Synchronous modes, respectively. The PSR and BAUD registers are memory mapped at data memory address locations xxBE and xxBD Hex, respectively.

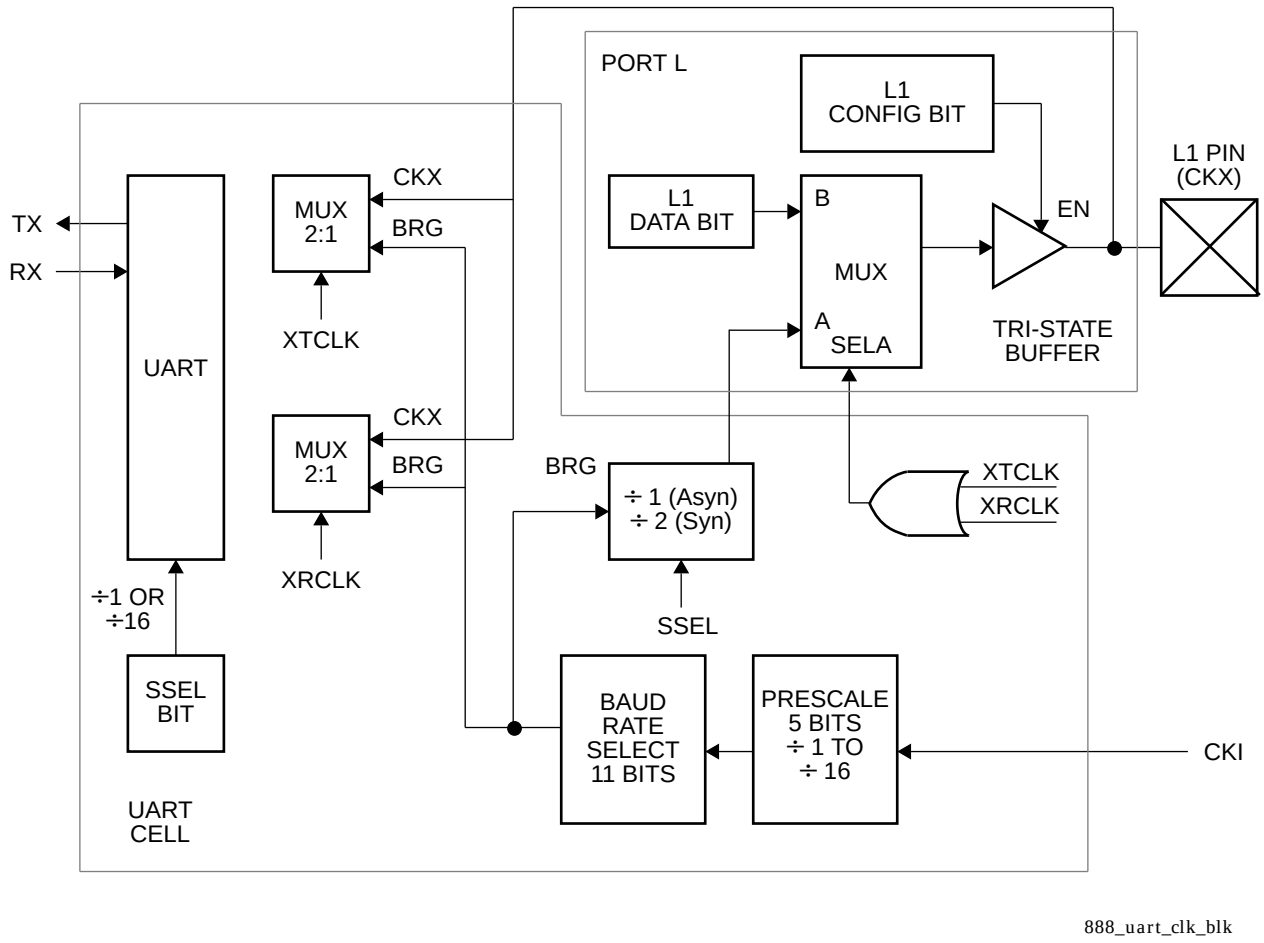


Figure 9-7 UART Baud Clock Generation Block Diagram

Table 9-6 UART Clock Sources (Asynchronous Mode)

XTCLK	XRCLK	TRANSMIT CLOCK	RECEIVE CLOCK	L1 PIN
0	0	BRG	BRG	Normal I/O
0	1	BRG	CKX	CKX I/O
1	0	CKX	BRG	CKX I/O
1	1	CKX	CKX	CKX I/O

Table 9-7 UART Clock Sources (Synchronous Mode)

XTCLK	XRCLK	TRANSMIT CLOCK	RECEIVE CLOCK	L1 PIN
1	1	CKX	CKX	CKX I/O

CKX I/O in Tables 9-6 and 9-7 means that the L1 pin can be configured as either an input or an output. With L1 configured as an input, an external clock can be used for the transmitter and/or receiver. With L1 configured as an output, the BRG clock is routed to the L1 pin, from where the BRG clock can be selected as the CKX clock for the transmitter and/or receiver. This is the data path used in the Synchronous mode to route the BRG clock (divided by 2) to the UART.

The selected internal Baud Rate Generator clock (BRG) can be output to the CKX pin in both the Asynchronous and Synchronous modes. In the Synchronous mode, the selected internal BRG clock is divided by two before it is output to the CKX pin. The division by two is necessary to obtain an output CKX clock with a 50 percent duty cycle. Note also that integer prescaler values (no fractional values such as 2.5) are necessary in the Synchronous mode to produce a 50 percent duty cycle output CKX clock. The CKX pin can also be selected as an external clock input for both the Synchronous and Asynchronous modes. In the Synchronous mode, both the XTCLK and XRCLK clock select bits should be set high to a “1” as shown in Table 9-7.

As an example of UART clock selection, consider the Asynchronous mode with XTCLK = 0 and XRCLK = 1 as selected from Table 9-6. The transmitter is clocked from the internal Baud Rate Generator (BRG), while the receiver is clocked from the CKX pin, which may be sourced from either an external clock input or from the BRG. Pin L1 is dedicated for the CKX clock whenever the CKX clock is selected. The pin L1 configuration bit determines whether a selected CKX clock is an input from an external source or an output from the BRG. This configuration bit is set to select the CKX pin L1 as an output and is reset to select CKX as an external input. Note that both the transmitter and receiver are clocked from BRG when the CKX pin is selected as an output.

In the Asynchronous mode, the UART divides the clock by 16, regardless of whether the clock is selected from BRG or CKX. In the Synchronous mode, the UART divides the CKX clock by 1. Note, however, that if the CKX clock is selected as an output clock in the

Synchronous mode, then the CKX clock will result from the selected BRG clock divided by 2 as shown in Figure 9-7. Consequently, a selected BRG clock in the Synchronous mode is divided by 2 before it reaches the UART by way of CKX.

Internally, the basic baud clock is created from the CKI input oscillator frequency through a two-section divider chain. This divider chain consists of a five-bit prescaler ranging from 1 to 16 in increments of 0.5, coupled with an 11-bit binary counter. The division factors are specified through two registers, PSR (Prescaler Select) register and BAUD (Baud Select) register, as shown in Figure 9-8. Note that the 11-bit Baud Rate Divisor spills over from the BAUD register into the PSR Prescaler Select register.

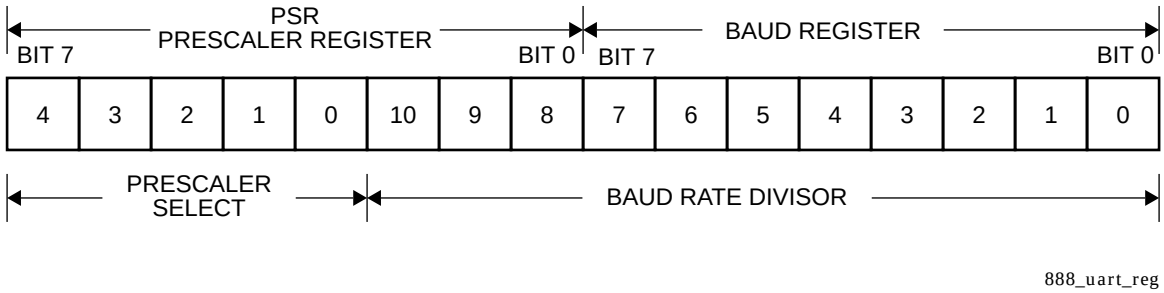


Figure 9-8 UART Baud Clock Divisor Registers

The Prescaler factor is selected by the upper five bits of PSR, while the baud rate divisor is determined by the lower three bits of PSR in conjunction with the eight bits of BAUD. This allows an eleven-bit baud rate divisor, ranging from 1 to 2048. Note that the programmed value is equal to the baud rate divisor minus one, which ranges from 0 to 2047. An all-zero prescaler value is reserved for selecting NO CLOCK. The NO CLOCK condition is the UART Power Down mode where the UART clock is turned off to save power. The application program must also turn off the UART clock when selecting a different baud rate.

Note that many features of the UART are disabled when the NO CLOCK option is selected. Consequently, even when an external CKX clock is being used for both the UART transmitter and receiver, the NO CLOCK option should not be used when any of the parity, 7/8 stop bit, Synchronous mode, 7 data bit, or Loopback Diagnostic mode options are selected.

The correspondence between the 5-bit Prescaler Select and the Prescaler factors is shown in Table 9-8. There are many different ways to calculate the two division factors (Prescaler and Baud Rate Divisor). One effective method is to achieve a 1.8432-MHz frequency coming out of the Prescaler. One possible method of producing this 1.8432-MHz frequency is to select a 9.216 MHz CKI oscillator clock rate coupled with a prescaler selection of a divide-by-five factor. The 1.8432 MHz prescaler output is then used to drive the software programmable baud rate counter to produce an X16 clock for the following baud rates: 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200, and 38400. These baud rates, together with their associated Baud Rate Division factors, are shown in Table 9-9. Note that Table 9-9 actually contains the baud rate divisor minus one (N-1) associated with each baud rate, where N is the baud rate divisor. The value (N-1) from Table 9-9 is the value to be programmed in the 11-bit baud rate counter (lower 3 bits of PSR and 8 bits of BAUD). The X16 clock is then divided by 16 in the UART

(Asynchronous mode only) to provide the clocks for the serial shift registers of the UART transmitter and receiver.

Table 9-8 UART Prescaler Factors (Sheet 1 of 2)

Prescaler Select	Prescaler Factor
00000	NO CLOCK
00001	1
00010	1.5
00011	2
00100	2.5
00101	3
00110	3.5
00111	4
01000	4.5
01001	5
01010	5.5
01011	6
01100	6.5
01101	7
01110	7.5
01111	8
10000	8.5
10001	9
10010	9.5
10011	10
10100	10.5
10101	11
10110	11.5
10111	12
11000	12.5
11001	13
11010	13.5
11011	14

Table 9-8 UART Prescaler Factors (Sheet 2 of 2)

Prescaler Select	Prescaler Factor
11100	14.5
11101	15
11110	15.5
11111	16

Table 9-9 UART Baud Rate Divisors, 1.8432 MHz Prescaler Output

Baud Rate	Baud Rate Divisor Minus One (N-1)
110 (110.03)	1046
134.5 (134.58)	855
150	767
300	383
600	191
1200	95
1800	63
2400	47
3600	31
4800	23
7200	15
9600	11
19200	5
38400	2

The entries in Table 9-9 assume a prescaler output of 1.8432 MHz.

Example 1:

This example further clarifies the usage of Tables 9-8 and 9-9 in selecting a desired baud rate. Consider a CKI clock frequency of 4.608 MHz with the UART Asynchronous mode selected, and suppose a baud rate of 19200 is desired. If a prescaler output frequency of 1.8432 MHz is selected in order to use Table 9-9, then 4.608 divided by 1.8432 yields a prescaler factor of 2.5 which is an available entry in Table 9-8. For a baud rate of 19200, the corresponding entry in Table 9-9 is 5. This means a value of 5 should be programmed as the value in the 11-bit counter (lower 3 bits of PSR and 8 bits of BAUD) to select a baud

rate divisor of 6. Consequently, N = 6 where N is the baud rate divisor corresponding to 19200 baud, and N-1 = 5 is the value from Table 9-9 to be programmed in the 11-bit counter. A value of 00100 would be programmed as the prescaler value in the upper 5 bits of PSR to match the prescaler factor of 2.5 from Table 9-8. The baud rate of 19200 can be calculated as the prescaler output frequency (1.8432 MHz) divided by 16 times the baud rate divisor of 6. The divide by 16 results from the UART input frequency being 16 times the baud rate in the Asynchronous mode. Note that in the Synchronous mode, this division factor of 16 is replaced by a division factor of 2.

The actual baud rate (BR) may be calculated as follows:

$$BR = CKIF/(16 \times P \times N)$$

where CKIF is the CKI input oscillator frequency, P is the prescaler division factor, and N is the baud rate divisor. This calculation is for Asynchronous mode UART operation, as indicated by the division factor of 16. Using Example 1 produces a calculation as follows:

$$BR = CKIF/(16 \times P \times N)$$

$$= (4.608 \times 10^6)/(16 \times 2.5 \times 6)$$

$$= 19200$$

Note that in the Synchronous mode this division factor of 16 is replaced by a division factor of 2. Consequently, baud rate may be calculated for the Synchronous mode as follows:

$$BR = CKIF/CKXF$$

$$= CKIF/(2 \times P \times N)$$

where CKXF is the CKX output frequency. Note that the division factor of 2 results from the BRG frequency being divided by 2 to produce a 50 percent duty cycle for the output CKX frequency.

Example 2:

As a further example, consider a CKI crystal frequency of 5 MHz with the UART in Asynchronous mode, where the desired baud rate is 9600. Using the asynchronous BR equation yields:

$$P \times N = CKIF/(16 \times BR)$$

$$= (5 \times 10^6)/(16 \times 9600)$$

$$= 32.552$$

The value 32.552 is now divided by each prescaler factor in Table 9-8 to obtain a value closest to an integer. A prescaler factor of 6.5 yields a result very close to an integer as follows:

$$N = 32.552/P$$

$$= 32.552/6.5$$

$$= 5.008$$

Consequently, N = 5 approximately, and a value of N-1 = 4 should be programmed in the 11-bit counter for the baud rate divisor value.

The calculated values of P and N are now used to calculate the actual baud rate that is produced:

$$BR = CKIF/(16 \times P \times N)$$

$$= (5 \times 10^6)/(16 \times 6.5 \times 5)$$

$$= 9615.385$$

The percentage error of the Baud Rate produced is:

$$\% \text{ ERROR} = (9615.385 - 9600)/9600$$

$$= 0.0016$$

$$= 0.16\%$$

9.1.10 UART Interrupts

The UART is capable of generating interrupts as result of Receive Buffer Full and Transmit Buffer Empty conditions. Both interrupts have individual interrupt vectors. Two bytes of program memory space are reserved for each interrupt vector. The two vectors are located at addresses xxEE to xxF1 Hex in the program memory space, with the high-order address byte depending on the location of the VIS instruction. The interrupts can be individually enabled or disabled using the Enable Transmit Interrupt (ETI) and Enable Receive Interrupt (ERI) bits in the ENUI register.

The interrupt from the Transmitter is set pending, and remains pending, as long as both the TBMT and ETI bits are set. To remove this interrupt, software must either clear the ETI bit or write to the TBUF register (thus clearing the TBMT bit).

The interrupt from the receiver is set pending, and remains pending, as long as both the RBFL and ERI bits are set. To remove this interrupt, software must either clear the ERI bit or read from the RBUF register (thus clearing the RBFL bit).

9.1.11 UART Error Flags

Error conditions detected by the UART receiver are brought to the software's attention by setting three error flags in the ENUR register. The flag bit FE (Framing Error) is set when the receiver fails to see a valid Stop bit at the end of the frame. The flag bit DOE (Data Overrun Error) is set if a new character is received in RBUF while it is still full (before the software has read the previous character from RBUF). The selected parity is verified by hardware, and the flag bit PE (Parity Error) is set if a parity error is detected. A global flag ERR (in the ENU register) is set if any of the three error conditions (FE, DOE, PE) occur.

If another character comes into the Receive Buffer (RBUF) before the software has read the previous character from RBUF, any errors associated with the new character will augment any previous errors already present. The receipt of a new character in RBUF

can cause any of the three error bits to go from a 0 to a 1, but not from a 1 to a 0. In other words, the receipt of a new character can set new errors but cannot reset previous errors.

The four error flags (ERR, FE, DOE, PE) are all read-only bits and cannot be written by the software. These four error bits are cleared with RESET. They are also cleared as a result of the HALT/IDLE mode. The three error flags FE, DOE, and PE are reset whenever they are read by software. Consequently, there are two considerations in accessing the ENUR register where the FE, DOE, and PE flags are located:

1. When reading the ENUR register, always check for receiver errors in the copy of the ENUR register read into the accumulator (or save the copy for error checking later).

2. Never perform a read type operation (other than LD or X) involving the accumulator on the ENUR register (examples: ADD, ADC, SUBC, AND, OR, XOR) since this clears the error flags without saving them for later test. Bit manipulation instructions (test bit, set bit, reset bit) or comparison instructions (including those on the accumulator) with the ENUR register do not clear the error flags. Moreover, a reset bit instruction on the error flag itself does not clear the error flag, since these error flags are read-only bits.

9.1.12 Diagnostic Testing

A loopback feature is provided for diagnostic testing of the UART. The CHL0 and CHL1 bits in the ENU register are used to select the loopback mode. When these bits are both set to one, the following internal connections are made:

1. The receiver input pin RDX is internally connected to the transmitter output pin TDX.

2. The output of the Transmitter Shift register is “looped back” into the Receiver Shift register input.

In this diagnostic mode, data that is transmitted is immediately received. This diagnostic feature allows the microcontroller to verify the transmit and receive data paths of the UART.

Note that the framing format for the diagnostic mode is the 9-bit format, consisting of one Start bit, nine data bits, and 7/8, 1, 1-7/8 or 2 Stop bits. Parity is not generated or verified in the Diagnostic mode.

A block diagram illustrating the Diagnostic mode “loopback” connection is shown in Figure 9-9.

9.1.13 Attention Mode

The UART receiver section supports an alternate mode of operation, called the Attention mode. This mode of operation is selected by the ATTN bit in the ENUR register. The framing format for transmission in the Attention mode is the 9-bit format, consisting of one Start bit, nine data bits, and 7/8, 1, 1-7/8, or 2 Stop bits. Parity is not generated or verified in the Attention mode.

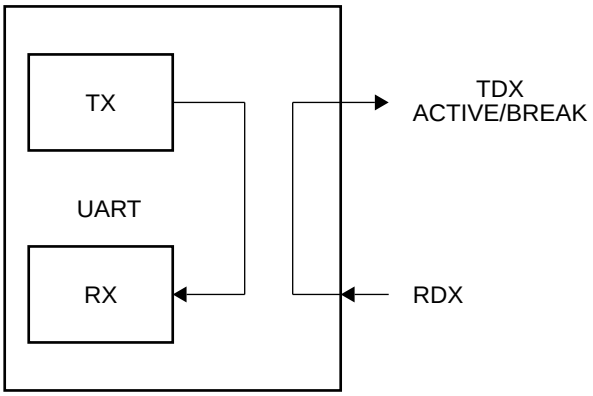


Figure 9-9 UART Diagnostic Mode Loopback Connection

The Attention mode of operation is intended for use in networking the microcontroller with other processors. Typically, in such environments, the messages consist of device addresses and data, with the device address indicating which of several destinations should receive the data. The device addresses are flagged in the messages by having the ninth bit of the data field set to a one. If the ninth bit of the data field is a zero, then the data field contains a data byte.

While in Attention mode, the UART monitors the communication flow but ignores all characters until an address character (a character whose ninth bit is set to 1) is received. Upon receiving an address character, the UART signals that the character is ready by setting the RBFL flag. This interrupts the microcontroller if the UART receiver interrupt is enabled. The ATTN bit is cleared automatically at this point, resulting in data characters as well as address characters being received. The software examines the contents of the RBUF register and responds by deciding either to accept the subsequent data stream (by leaving the ATTN bit reset) or to wait until the next address character is received (by setting the ATTN bit again).

The operation of the UART transmitter is not affected by selection of the Attention mode. The value of the ninth data bit to be transmitted is programmed by setting the XBIT9 bit appropriately. The value of the ninth data bit received is obtained by reading the RBIT9 bit. This ninth data bit in the Attention mode is used to distinguish between address and data characters.

Since RBIT9 is located in the ENUR register where the three error flags (FE, DOE, PE) reside, care should be taken not to reset the error flags while reading RBIT9. To avoid clearing the error flags, use a test bit instruction to read RBIT9.

9.1.14 Break Generation and Detection

To emulate a line Break, reset the ETDX bit in the ENUI register to change the TDX pin into a general-purpose Port L pin, and output a logic 0 on the same pin using the Port L data and configuration registers, bit 2.

Break detection is not directly supported by the UART, but can be programmed in software by using the RCVG flag in the ENUR register. This flag is set high when a framing error occurs, and is reset when RDX goes high.

Chapter 10

ANALOG-TO-DIGITAL CONVERTER

10.1 INTRODUCTION

Some COP888 devices have a multi-channel, multiplexed-input, ratiometric Analog-to-Digital (A/D) Converter. The A/D Converter receives an analog voltage signal on an input pin (or a pair of input pins), and converts the analog signal into a 8-bit digital value using successive approximation. The digital value can then be read by the software from a memory-mapped register. The input voltage range is defined by the V_{CC} and GND voltages.

NOTE: The COP888EB has a separate V_{CC} reference voltage input pin called A/D V_{REF} used for the A/D Converter.

The A/D Converter is useful in applications where the software needs to read a value (for example, temperature or speed) from an analog sensor. Up to eight different single-ended or four different differential inputs can be handled by a single COP888 device through the multiplexed input channels.

NOTE: The COP888CF has an A/D Converter that is similar to the one described in this chapter. However, that A/D Converter has a different format for the ENAD register (A/D Converter Control Register) and has separate voltage reference pins for specifying the analog input range. Because of these differences, the COP888CF A/D Converter is described in the COP888CF chapter (Chapter 15).

10.2 A/D OPERATION

Figure 10-1 is a block diagram showing the structure of the COP888 A/D Converter. There are two memory-mapped registers in the A/D Converter circuit: the A/D Converter control register (ENAD), used for configuring and enabling the A/D Converter, and the A/D Result register (ADRSLT), a read-only register containing the results of the conversion the A/D Converter supports ratiometric measurements. The input voltage range is determined by two fixed reference voltages supplied to the device through pins GND (ground) and V_{CC} (power supply voltage). GND is the lower limit and V_{CC} is the upper limit of the input voltage range.

Analog voltages are received on pins ACH0 through ACH7 (Analog Channels 0 through 7), which are alternate functions of the Port I pins. A multiplexer selects one channel for a single-ended conversion, or a pair of channels for differential-pair conversion, based on the value programmed into the ENAD register. The input voltage (or for a differential pair, the difference between the two input voltages) is sent to a comparator. The comparator compares the input voltage with a voltage generated by an internal digital-to-analog converter.

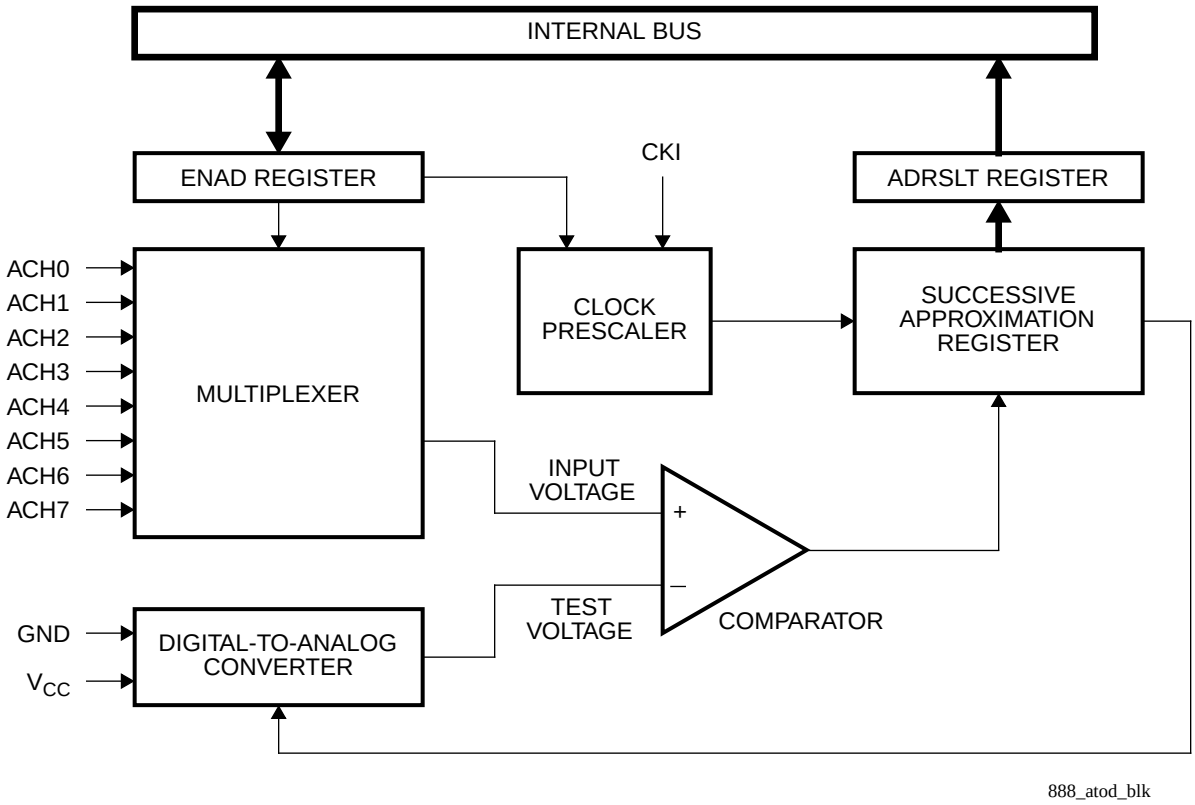


Figure 10-1 COP888 A/D Converter Block Diagram

The A/D Converter uses successive approximation to determine the analog input voltage. The internal digital-to-analog converter generates a test voltage, which is initially set to the middle of the possible voltage range, halfway between the voltages on GND and V_{CC}. After the comparison is made, another comparison is made with the test voltage set to the middle of the new possible range (either 1/4 or 3/4 between GND and V_{CC}). This is repeated until the input voltage is determined to a precision of 1/256 of the full input voltage range (eight iterations).

Each time a comparison is made, a bit is set or cleared in the Successive Approximation Register, depending on whether the input voltage is higher or lower than the test voltage. The bit values in the Successive Approximation Register are determined from most to least significant bit, generating an 8-bit digital value that is proportional to the analog input voltage. The value 00 Hex represents the lowest voltage (GND), and a value of FF Hex represents the highest voltage (V_{CC}). At the end of the conversion, the 8-bit value is transferred to the ADRSLT register, allowing the final result to be read by the software. The Successive Approximation Register cannot be accessed directly by the software.

The A/D Converter uses its own clock, which is generated by scaling down the CKI clock to a lower frequency. The clock prescaler circuit allows a choice of four different divide-by factors for generating the A/D clock. The choice is made by programming a set of bits in the ENAD register. This feature allows a certain amount of control over the trade-off between speed and accuracy of the A/D Converter without changing the chip clock speed, as explained later in this chapter.

A single conversion takes 17 A/D clock cycles: one for A/D reset, seven for sampling the input voltage, eight for conversion, and one for transferring the result to the ADRSLT

register. The total conversion time depends on the chip clock speed and the divide-by factor used for generating the A/D clock. If analog voltages on different channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

The A/D Converter circuit is powered down when the device enters the HALT or IDLE mode. If the A/D Converter is running when this happens, the conversion is cancelled, and the conversion is restarted, without resampling, upon exit from the HALT or IDLE mode. The application program should always terminate and restart any pending conversions due to possible corruption of the sampled voltage upon exiting the HALT or IDLE mode.

10.3 A/D CONVERTER REGISTERS

Two memory-mapped registers are used with the A/D Converter: the ENAD register and the ADRSLT register The ENAD register is a read-write memory location used for controlling the operation of the A/D Converter. The ADRSLT register is a read-only register that contains the most recent A/D conversion result.

The A/D Converter is controlled by writing a byte to the ENAD register. The data byte written to this register enables (or disables) the A/D Converter clock, sets the divide-by factor for generating the clock, selects the operating mode (single-conversion or continuous mode, single-ended or differential inputs), and selects the channel or channel pair that is to receive the analog input. The register bit map for the ENAD register is shown in Table 10-1.

Table 10-1 ENAD, A/D Converter Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CHS2	CHS1	CHS0	DIFF	CONT	PRSC1	PRSC0	ADBSY
CHS2–CHS0:	Channel Selection bits						
DIFF:	Differential Mode (0 = single-ended, 1 = differential)						
CONT:	Continuous Mode (0 = single conversion, 1 = continuous)						
PRSC1–PRSC0:	A/D clock prescaler selection bits						
ADBSY:	A/D enable and Busy bit						

The ENAD register is cleared upon reset. This inhibits operation of the A/D clock and disables the A/D converter. To begin using the A/D converter, it is only necessary to write the proper bit values to the ENAD register, as described in detail below.

The ADRSLT register contents are unknown following a reset.

10.3.1 Busy Bit

The ADBSY bit of the ENAD register is used to start and stop the A/D conversion. When ADBSY is cleared, the prescale logic is disabled and the A/D clock is turned off. Setting the ADBSY bit starts the A/D clock and initiates a conversion based on the mode select

value currently in the ENAD register. Normal completion of an A/D conversion clears the ADBSY bit and turns off the A/D converter.

The ADBSY bit remains a one during continuous conversion. The application program can stop continuous conversion by writing a zero to the ADBSY bit.

To stop and restart a conversion already in progress, first write a zero to the ADBSY bit to stop the current conversion and then write a one to ADBSY to start a new conversion. This can be done in two consecutive instructions.

10.3.2 Prescaler Selection

The A/D clock is generated by dividing the CKI clock. The resulting clock period is an integer multiple of the CKI clock. The divide-by factor can be set to 2, 4, 6, or 12. However, the resulting A/D clock frequency must be between 100 KHz and 1.67 MHz for proper operation of the A/D circuit.

With a CKI clock running at 10 MHz (1 MHz instruction cycle), a divide-by factor of 6 results in an A/D clock frequency of 1.67 MHz, the maximum allowed A/D clock speed. Therefore, with the CKI clock running at 10 MHz, the selected divide-by factor must be 6 or 12. A lower divide-by factor can be used only if the CKI clock is running slower than 10 MHz. In any case, the resulting clock frequency must be between 100 KHz and 1.67 MHz.

The prescaler divide-by factor is programmed by writing bits 2 and 1 in the ENAD register. Table 10-2 shows the binary values used for programming this part of the register, and the resulting A/D clock for each binary value.

Table 10-2 A/D Prescaler Options

ENAD Bits 2-1	A/D Clock
00	Divide CKI by 2
01	Divide CKI by 4
10	Divide CKI by 6
11	Divide CKI by 12

10.3.3 Single Conversion or Continuous Mode

The A/D Converter can be programmed to perform a single conversion, or to perform conversions continuously using the same single-ended or differential-pair input. The single-conversion mode can be used to obtain a single analog value, or used sequentially to obtain values from different channels. The continuous mode can be used to monitor a single channel or one differential-pair input over a period of time. The operating mode is controlled by bit 3 of the ENAD register: 0 for the single-conversion mode, or 1 for the continuous mode.

In the single-conversion mode, the conversion is started as soon as the ADBSY bit of the ENAD register is set by the software. After 17 A/D clock cycles, the result is transferred

to the ADRSLT register and ADBSY is cleared. The software can read the result any time after that.

In the continuous mode, the conversion is performed in the same manner as the single-conversion mode, but conversions are repeated continuously using the same analog input. A new result is transferred to the ADRSLT register every 17 A/D clock cycles, overwriting the previous result. The software can read the most recent result from the ADRSLT register at any time. Conversions are performed repeatedly until the software clears the ADBSY bit of the ENAD Register.

10.3.4 Channel Selection

Analog voltages are received on pins ACH0 through ACH7, which are alternate functions of Port I pins I0 through I7.

The A/D Converter uses one channel for a single-ended conversion, or a pair of channels for differential-pair conversion. The single-ended or differential mode is selected with bit 4 of the ENAD register: 0 for single-ended, or 1 for differential. The channel number (or set of two channel numbers) is selected with bits 7, 6, and 5 of the same register, as indicated Table 10-3.

When the A/C Converter is used in differential mode, only the positive input to the A/D converter is sampled and held. The negative input is constantly connected and should be held stable for the duration of the conversion. Failure to maintain a stable negative input will result in incorrect conversion.

Table 10-3 A/D Channel Selection

ENAD Bits 7-6-5	Single-Ended Mode	Differential Pair Mode (+, -)
000	ACH0	ACH0, ACH1
001	ACH1	ACH1, ACH0
010	ACH2	ACH2, ACH3
011	ACH3	ACH3, ACH2
100	ACH4	ACH4, ACH5
101	ACH5	ACH5, ACH4
110	ACH6	ACH6, ACH7
111	ACH7	ACH7, ACH6

10.4 MULTI-CHANNEL CONVERSION

The A/D Converter allows the use of up to eight single-ended or four differential-pair inputs. However, there is only one A/D converter, with only one result register. If multiple channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

Figure 10-2 shows an example of an assembly language program that performs this function. The program makes a request for an analog-to-digital conversion from each of eight single-ended channels in sequence, reads the eight result bytes, and writes the results into a block of data memory. Operation of the program is explained below.

```
                              ;A/D, convert 8 channels     Bytes/Cycles
ADC8:     RC                  ;Reset carry bit                    1/1
          LD      ENAD,#5     ;Select ACH0 conversion and PSC=6   3/3
                              ;Start the conversion
          NOP                 ;Needed for timing on first         1/1
          LD      B,#ENAD     ;B pointer to ENAD register         2/2
          LD      X,#DEST     ;X pointer to result destination    2/3
LOOP:     LD      A,[B]       ;Get previous ENAD contents         1/1
          ADC     A,#021      ;Change to next input channel       2/2
          X       A,[B+]      ;Request next conversion            1/2
          LD      A,[B-]      ;Get previous result from ADRSLT    1/2
          X       A,[X+]      ;Store result and incr. pointer     1/3
          IFNC                ;Test ADC instr. for overflow       1/1
          JP      LOOP        ;Loop back for next channel         1/3
```

Figure 10-2 A/D Conversion Routine

The program was written with execution speed as the primary consideration. The comment text in the program listing shows the purpose of each instruction, the number of bytes of program memory used by the instruction, and the number of instruction clock cycles required to execute the instruction.

At the start of the program, the first conversion is initiated, specifying channel ACH0 with a clock divide-by factor of 6. Both ENAD and ADRSLT will be accessed with the B pointer, so the B register is initialized to ENAD. The results will be stored in a block of data memory occupying eight bytes starting at DEST. The data memory will be accessed with the X pointer, so the X register is initialized to DEST.

The program loop first reads the ENAD register, changes the control byte to specify the next single-ended channel (ACH1 the first time through the loop), and uses the X (exchange) instruction to request the next conversion. The X instruction also increments the B register, setting up the next instruction to access the ADRSLT register. (The ADRSLT register immediately follows the ENAD register in data memory.) The result byte from the previous conversion is loaded into the Accumulator from ADRSLT, and at the same time, the B pointer is decremented in preparation for the next pass through the loop. The result byte is written into data memory using the X pointer. The IF instruction tests for overflow from the ADC instruction, which will happen after all eight result bytes have been written (eight passes through the loop).

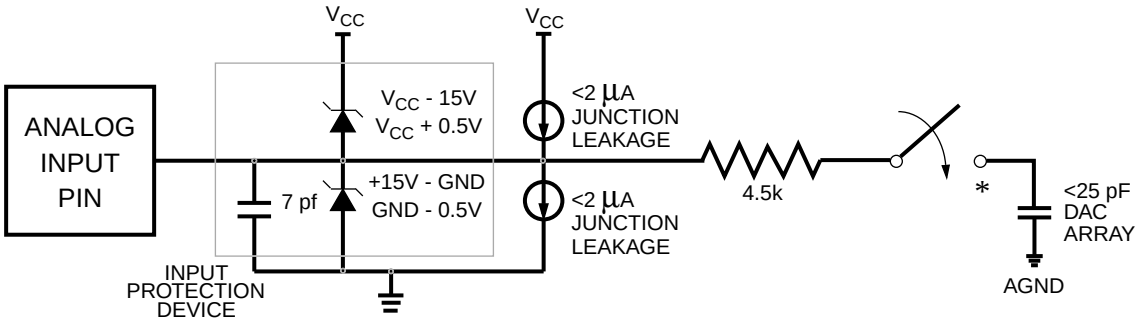
In each iteration of the loop, the program requests a new conversion and quickly reads the results of the previous conversion. The old result remains in the ADRSLT register until overwritten with the new result, which takes 17 A/D clock cycles, or 102 CKI clock cycles. The loop section of the program takes 14 instruction clock cycles to execute, or 140 CKI clock cycles. Thus, the new result becomes available before it is needed in the next iteration of the loop.

The program code uses 17 bytes of program memory. For execution, the whole routine takes 120 instructions cycles: 10 cycles for initial setup, 14 cycles for the first seven loops, and 12 cycles for the last iteration of the loop (1 cycle to skip over the JP instruction). With a 10 MHz CKI clock, the routine takes 120 microseconds.

10.5 SPEED, ACCURACY, AND HARDWARE CONSIDERATIONS

The maximum allowed A/D clock speed of 1.67 MHz results in a clock period of 600 ns. The time required for one A/D conversion is 17 A/D clock cycles, or 10.2 microseconds. This is the shortest possible conversion time for the COP888GD A/D converter. In some cases, a longer conversion time may be necessary or desirable in order to ensure accurate conversions. This is because it takes some time for the analog input signal to charge the capacitive load on the A/D input.

Figure 10-3 shows the internal operation of an analog input pin in the single-ended mode, together with the capacitive loads and input protection components on the pin. The analog switch is kept open most of the time. For an A/D conversion, the switch is closed for the duration of seven A/D clock cycles, and the voltage is sampled during this period.



*The analog switch is closed only during the sample time.

Figure 10-3 Analog Input Pin Internal Operation

The A/D clock period should be made long enough to allow the capacitor to charge up to the full voltage supplied to the analog input pin. Whether the minimum A/D clock period of 600 ns is sufficient depends on the output impedance of the analog source and the accuracy requirements of the application. In general, the benefits and costs of using a slower A/D clock should be considered when the analog source impedance exceeds 3 KΩ. As a rule of thumb, you should increase the A/D clock period in proportion to the source impedance. For example, for a source impedance of 6 KΩ, use twice the minimum clock period of 600 ns, or 1200 ns (833 KHz). The A/D clock can be slowed down either by increasing the programmed divide-by factor or by reducing the CKI clock frequency, or a combination of both. The A/D clock frequency should not be reduced below 100 KHz.

Both electrical noise and digital clock coupling to the analog inputs can cause inaccurate conversions. For the most accurate possible results, design the circuit board to minimize noise on these inputs. Keep the leads to the analog input pins as short as possible, and as far away as possible from clock leads.

Chapter 11

CAN INTERFACE

11.1 INTRODUCTION

Some COP888 devices have a built-in Controller Area Network (CAN) interface. CAN is a communications network standard often used in automotive and industrial control applications.

In a CAN system, multiple modules share a common serial communications bus. Modules communicate with one another by sending and receiving messages in data packets called "frames." A message on the bus operates globally and is received by all other modules, allowing every module in the system to access every message, if required. Each message has a single source and can be used by any number of modules on the bus. A message not needed by a particular module is acknowledged and checked for errors but otherwise ignored.

Each message has an identifier number to distinguish it from other messages and to resolve conflicts between messages. When multiple messages are placed on the bus simultaneously, highest-priority message is allowed to continue. The other messages are canceled and delayed until the frame is completed. The feature is known as Carrier Sense Multiple Access - Collision Avoidance (CSMA-CA).

COP888 devices with a built-in CAN interface support the CAN Specification Revision 2.0 Part B, or "Passive" type operation, at data rates of up to 1 Mbit/second using standard message identifiers. This interface is suitable for lower-speed (CAN class A and B) applications. For example, in automotive applications, it is suitable for control of instrument gauges, door locks, headlights, and audio systems.

11.2 BASIC CAN CONCEPTS

This section is a general overview of Controller Area Network (CAN) principles as described in Chapter 4 of the ISO/DIS11519-1 specification.

The CAN system is a message-based protocol that allows a total of 2032 (2^{11}-16) different messages in the "standard frame" format or 512 million (2^{29}-16) different messages in the "extended frame" format. A COP888 device with a built-in CAN interface fully supports the standard frame format but not the extended frame format. The COP888 device acknowledges extended frames but discards the data in them. Thus, the device cannot use extended frames, but can be used in a network containing other modules that use extended frames.

The descriptions of frame formats in this document are generally restricted to the standard frame format.

11.2.1 Physical Layer

The physical implementation of the bus is not specified in the CAN specification. You can choose to either connect these bus input/output pins of the COP888 device directly to the CAN bus or connect them to an external bus driver interface, depending on the system requirements.

The COP888 device has five pins dedicated to CAN usage:

- Vref: On-chip reference voltage with a value of Vcc/2
- Tx0: Transmit data output pin 0
- Tx1: Transmit data output pin 1
- Rx0: Receive data input pin 0
- Rx1: Receive data input pin 1

If you choose to connect the device input/output pins directly to the CAN bus, the bus is implemented as two serial data lines, Tx0/Rx0 and Tx1/Rx1.

There are two possible bit values, called "dominant" and "recessive." The “dominant” value corresponds to a zero in the COP888 CAN register. For transmitting a dominant bit, output pin Tx0 drives the Tx0 line low and output pin Tx1 drives the Tx1 line high. For transmitting a recessive bit or for receiving, the device places both the Tx0 and Tx1 output pins in the TRISTATE (high impedance) mode, allowing the serial bus lines to be driven by an external module. The device never drives the Tx0 line high or the Tx1 line low.

11.2.2 Modules and Frames

When a module sends a message, it places a frame on the CAN bus. Every module receives every frame placed on the bus. Each frame contains a message identifier, allowing each module to filter out frames that it does not need. For example, in an automotive application, when the dashboard module sends out a request to switch on the headlights, only the module that controls the headlights responds to the message. The module that controls the brake lights receives and acknowledges the message but ignores the information in the message.

A module can issue a request for data from another module by setting a bit in the transmitted frame. This bit is called the Remote Transmission Request (RTR) bit. A frame with this bit set is called a "remote frame." A remote frame causes another module to transmit a "data frame" after transmission of the remote frame has been completed.

If a module needs more time to process a message, it can place an "overload frame" on the bus. This is a signal to other modules to delay transmission of the next message. Upon completion of the overload frame, modules on the bus are again allowed to transmit messages.

Similarly, a module can place an "error frame" on the bus to signal an error condition to other modules. An error frame can start at any time, including the middle of another frame.

Additional modules can be added to an existing network without a configuration change. An added module can either perform a completely new function requiring new data (for example, an automatic window opener) or process existing data to perform a new function (for example, oil pressure measurement).

11.2.3 Contention-Based Arbitration

The CAN protocol allows any module to start a transmission when the bus is idle. This means that multiple modules can attempt to start a transmission simultaneously. Simultaneous transmission attempts are resolved non-destructively according to the ordered priority of the modules in the network.

When any module starts transmission of a frame, it monitors the bus line to determine whether its message is being overwritten by another message with a higher priority. As soon as the module detects another module with a higher priority accessing the bus, it stops transmitting its own frame and switches to the receive mode.

Figure 11-1 illustrates this process. Module A and Module B both attempt to transmit a frame at the same time. At the dashed line, Module A detects a higher-priority module (Module B) accessing the bus, so it stops transmission at that point.

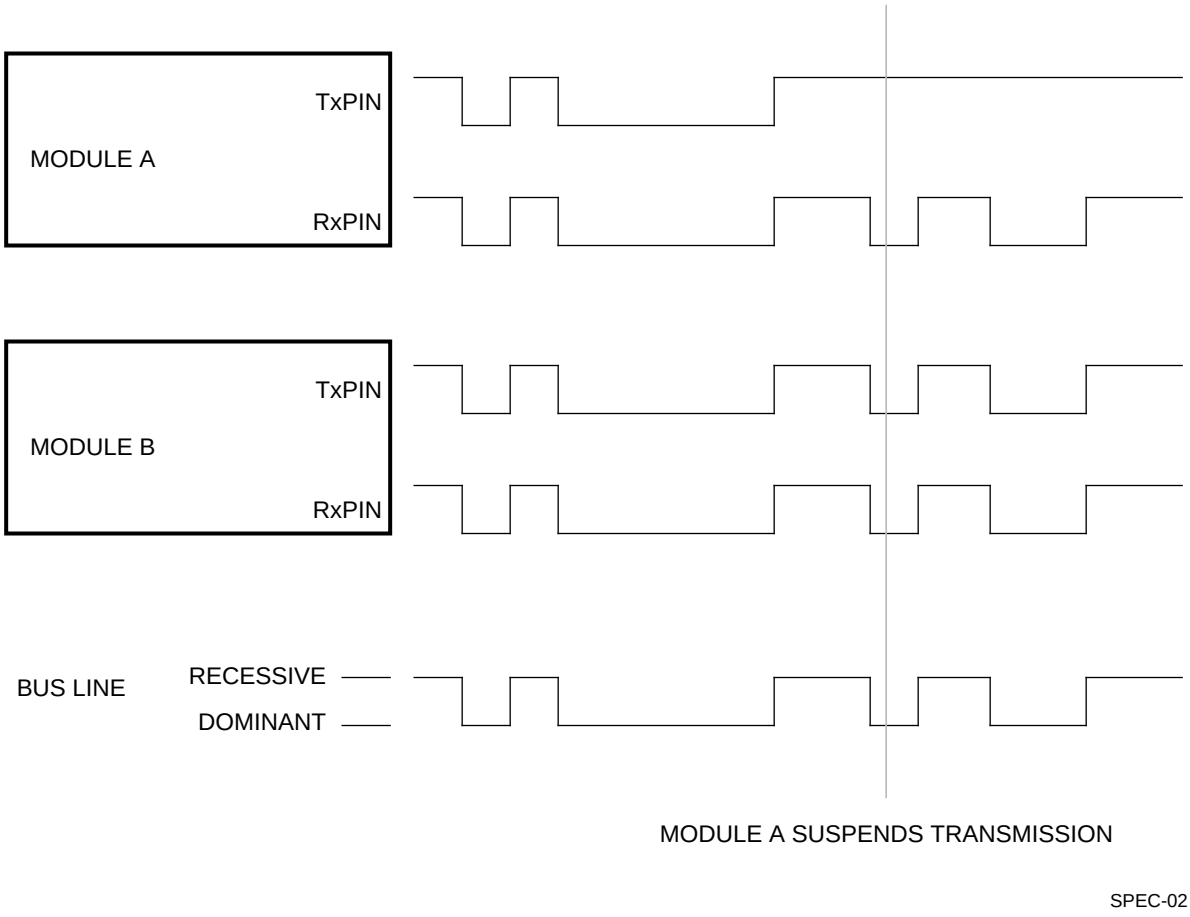


Figure 11-1 CAN Message Arbitration

Later, when the bus is idle, the module that was forced to stop tries again to transmit the same frame. The COP888 hardware automatically handles retransmission of up to two data bytes. A message longer than two data bytes requires the application software to update the transmit registers with the required data.

11.2.4 Error Detection and Error Handling

Every module in the network checks and acknowledges every message placed on the bus, including messages intended for other modules. If any module detects an error in any message, it starts the transmission of an "error frame."

Each module has two error counters: one for counting transmitted data errors and one for counting received data errors. If the current count in either counter exceeds a specific value, the module enters an error state. There are three possible error states for a module:

- Error-active. The module can still participate in bus communications. If it detects another error, it sends an active (dominant) error flag.
- Error-passive. The module can still participate in bus communications. If it detects another error, it sends a passive (recessive) error flag.
- Bus-off. The module has its output drivers disabled and therefore does not participate in bus communications.

Errors and module error states are explained further in Section 11.6.

11.3 FRAMES

A frame is a packet of data transmitted over the CAN bus. There are two general types of frames: data transmission frames and control frames. Data transmission frames are used to transmit data, whereas control frames are used to transmit error information.

In the data transmission frame category, there are two types of frames: data frames and remote frames. A data frame carries data from one module to another. A remote frame is a request for data transmission from another module in the network.

In the control frame category, there are two types of frames: error frames and overload frames. A module transmits an error frame when it detects an error condition. A module transmits an overload frame when it is unable to process data quickly enough to respond to the next possible message on the bus.

Another type of frame called an "interframe space" precedes every data frame and remote frame.

11.3.1 Frame Fields

Each frame consists of a sequence of fields, and each field consists of a sequence of bits. The types of fields and the number of bits in each field varies, depending on the frame type. The fields and bits defined for each frame type are shown in Figure 11-2 through 11-4:

- Figure 11-2 shows the formats for data frames and remote frames.
- Figure 11-3 shows the format for interframe spaces.
- Figure 11-4 shows the frame formats for error frames and overload frames.

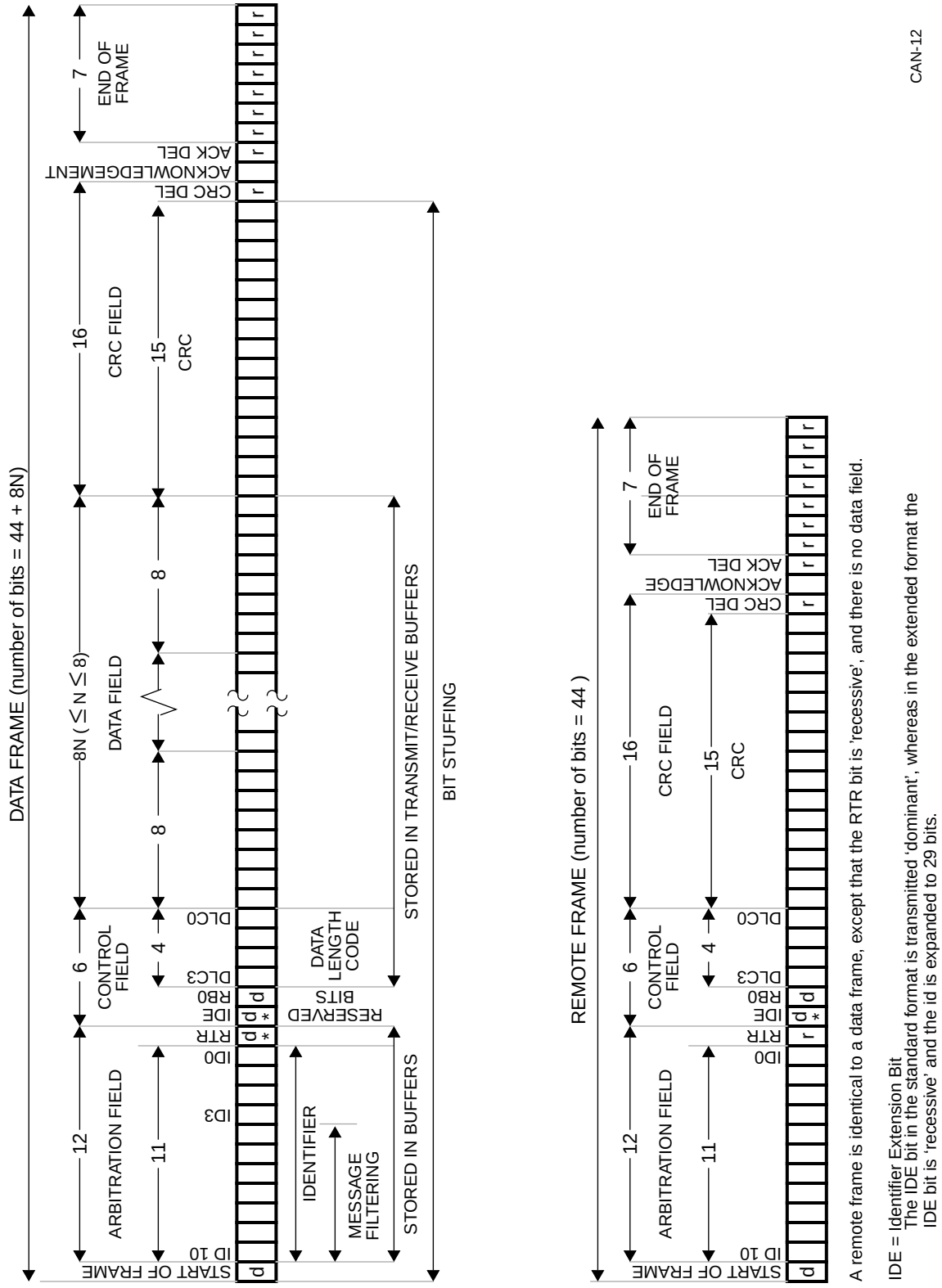


Figure 11-2 Data Frame and Remote Frame Formats

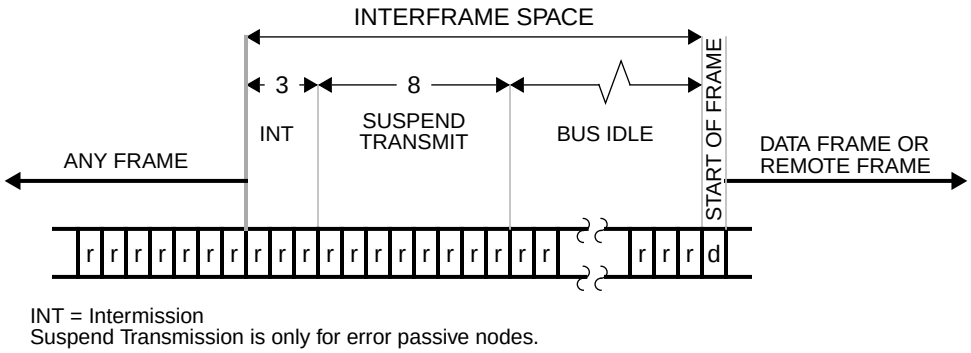
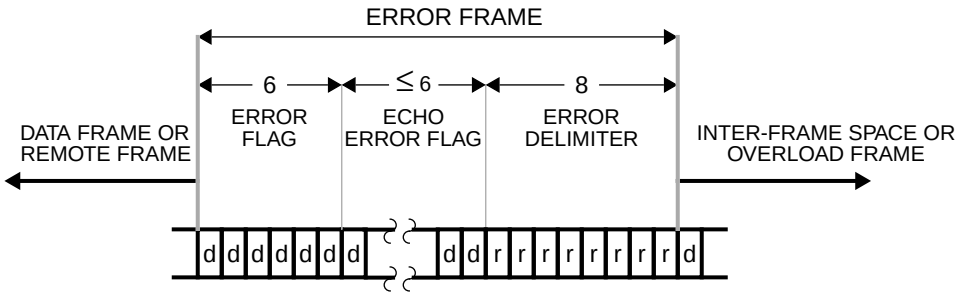


Figure 11-3 Interframe Space Format



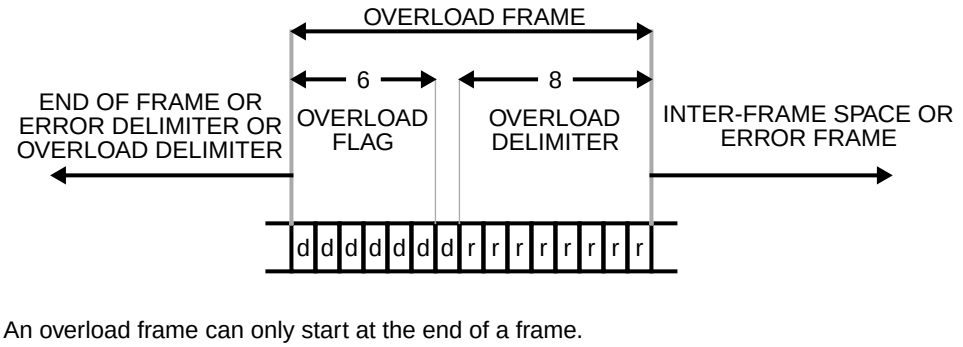


Figure 11-4 Error Frame and Overload Frame Format

11.3.2 Frame Coding

Each transmitted frame starts with a Start of Frame (SOF) bit followed by the remaining fields in sequential order. Bits are transmitted in order, from most to least significant bit.

Bits are Non-Return to Zero (NRZ) coded on the serial line. A logic 1 is coded as a "recessive" bit on the bus (the high-impedance state). A logic 0 is coded as a "dominant" bit on the bus (the bus line is pulled low). When two or more modules write conflicting values on the bus, a dominant bit overwrites all recessive bits.

Within data frames and remote frames, "bit stuffing" is used to increase the reliability of the transmission. After five consecutive bits of the same value (00000 or 11111), a "stuff bit" of the opposite value is inserted by the transmitter. At the receiver end, the stuff bit is deleted from the bit stream to restore the original data. For example, the bit stream 100000X is transmitted as 1000001X. At the receiver end, the stuff bit "1" is removed, thus restoring the original bit stream, 100000X.

Bit stuffing is only used in data frames and remote frames, and only in the fields that hold information for the interface: the start of frame, arbitration, control, data, and CRC fields.

11.3.3 Data Frames and Remote Frames

A data frame is used to transmit data from one module to another. A standard data frame consists of seven fields:

1. Start of Frame (SOF)
2. Arbitration field
3. Control field
4. Data field
5. CRC field
6. ACK field
7. End of Frame (EOF)

A remote frame is a request for data transmission from another module in the network. The format of a remote frame is similar to that of a data frame, except that there is no data field within the frame and there is one bit in the arbitration field that is recessive rather than dominant.

Start of Frame (SOF)

The Start of Frame (SOF) field is a single dominant bit that indicates the beginning of a data frame or remote frame.

A module is allowed to start transmission of a frame only when the bus is idle. When this happens, all modules in the network must synchronize to the leading edge of the SOF bit, which is the first edge that occurs on the bus after the idle state.

Arbitration Field

The arbitration field consists of 12 bits. The first 11 bits form an identifier field that identifies the message as a data frame or a remote frame.the data frame or remote frame. The last bit in the arbitration field is the Remote Transmission Request (RTR) bit, which is a dominant bit in a data frame or recessive bit in a remote frame.

Control Field

The control field consists of six bits. The first bit is the Identifier Extension (IDE) bit. A dominant bit indicates a regular ID number and a recessive bit indicates an extended ID number. An extended ID number uses an additional bits after the RTR bit (in that case called the "substitute" RTR bit). The second bit in the control field is called the RB0 bits, which is reserved for future expansion.

With respect to the IDE and RB0 bits, modules implemented with the COP888 device always transmit zeros (dominant bits) and accept any combination of bit values. The IDE bit, if set by an external module, indicates an extended frame with 29 identifier bits. The COP888 cannot process data in extended frames, but accepts extended frames without generating an error. This allows extended frames to be used by other modules in an existing network without interference from the COP888- based module.

The remaining four bits of the control field are the Data Length Code (DLC) bits, which specify the number of bytes in the data field. The number of data bytes can be any value from zero to eight.

Data Field

The data field contains the data being transmitted in a data frame. Only data frames (not remote frames) have this field.

The data field consists of zero to eight whole bytes of data, with eight bits per byte. Within each byte, bits are transmitted in order from most to least significant bit.

Note: The CAN specification does not define the order of data bytes within the data field.

CRC Field

The Cyclic Redundancy Checking (CRC) field consists of a 15-bit CRC sequence followed by the CRC delimiter bit, a single recessive bit. The CRC sequence provides a way for the receiver to check for and correct transmission errors.

The transmitter derives the CRC sequence by modulo 2 division of the preceding bit fields by the generator polynomial:

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

The CRC is generated from all the bits leading up to the CRC field, starting with the SOF bit and ending with the last bit of the data field (excluding any stuff bits inserted into the bit stream). The remainder of the division is the 15-bit CRC sequence transmitted over the bus.

On the receiver side, the module divides the whole sequence of bits by the generator polynomial, starting from the SOF bit and ending with the last bit of the CRC sequence (excluding any stuff bits). A nonzero result indicates a transmission error.

ACK Field

The acknowledge (ACK) field contains two bits: the ACK slot bit and the ACK delimiter bit.

The transmitter fills the ACK slot with a recessive bit. Every receiver that receives a correct CRC sequence overwrites this same slot with a dominant bit.

The transmitter transmits a recessive bit for the second ACK field bit, the ACK delimiter.

The ACK slot is preceded by the CRC delimiter bit and is followed by the ACK delimiter bit. These two bits surrounding the ACK slot are both recessive bits in a valid data frame or remote frame.

End of Frame (EOF) Field

The End of Frame (EOF) field consists of seven recessive bits. It indicates the end of a data frame or remote frame.

11.3.4 Interframe Spaces

Each data frame and remote frame is separated from the preceding frame (a data, remote, error, or overload frame) by an interframe space. The format of an interframe space depends on the error status of the module that transmitted the previous frame.

Figure 11-5 shows the usual format of an interframe space. Following the end of the previous frame, there is an "intermission" field containing three recessive bits followed by a "bus idle" field containing any number of recessive bits. The bus idle state continues until a module in the network starts a new transmission.

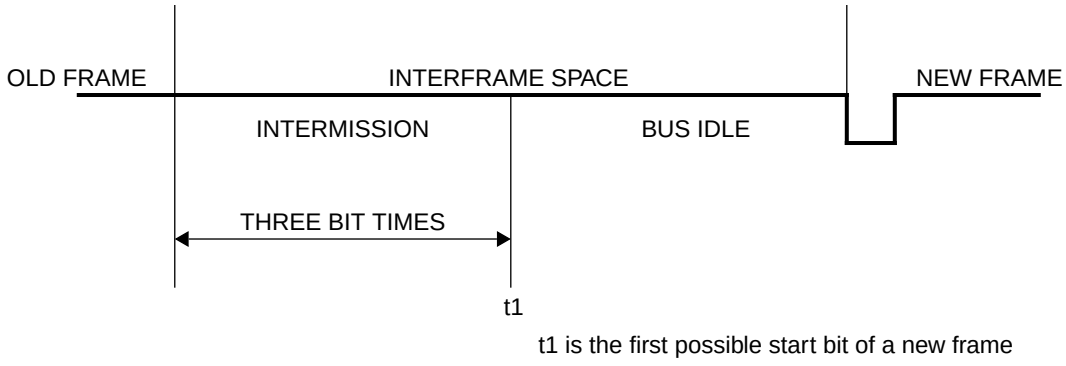


Figure 11-5 Normal Interframe Space Format

While the intermission field is active, no node is allowed to start transmission of a data frame or remote frame. The only action that can be taken on the bus at this time is the signaling of an overload condition. This means that any error in the intermission field is interpreted as an overload condition.

Figure 11-6 shows the format of an interframe space generated by a module in the "error-passive" state that was also the transmitter of the previous frame. The format is the same as for the usual case, except that there is an additional field called the "suspend transmission" field between the intermission and bus idle fields. The suspend transmission field contains eight recessive bits.

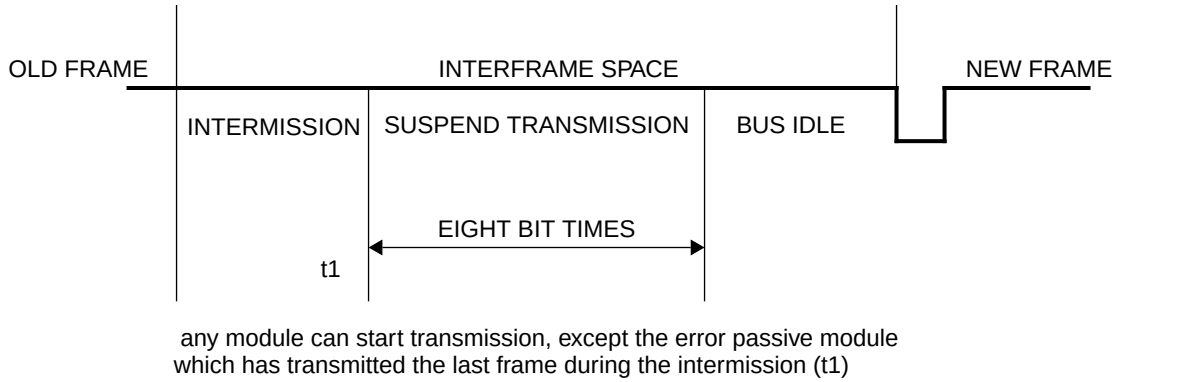


Figure 11-6 Interframe Space Format for Error-Passive Transmitter

The module that generates the suspend transmission field cannot start a new transmission during the suspend transmission period. It must wait for the bus idle state. However, any other module can start a new transmission during the suspend transmission time. The dominant start bit the other module overwrites the suspend transmission bit generated by the error-passive module. In other words, other modules treat the suspend transmission time the same as bus idle.

11.3.5 Error Frames

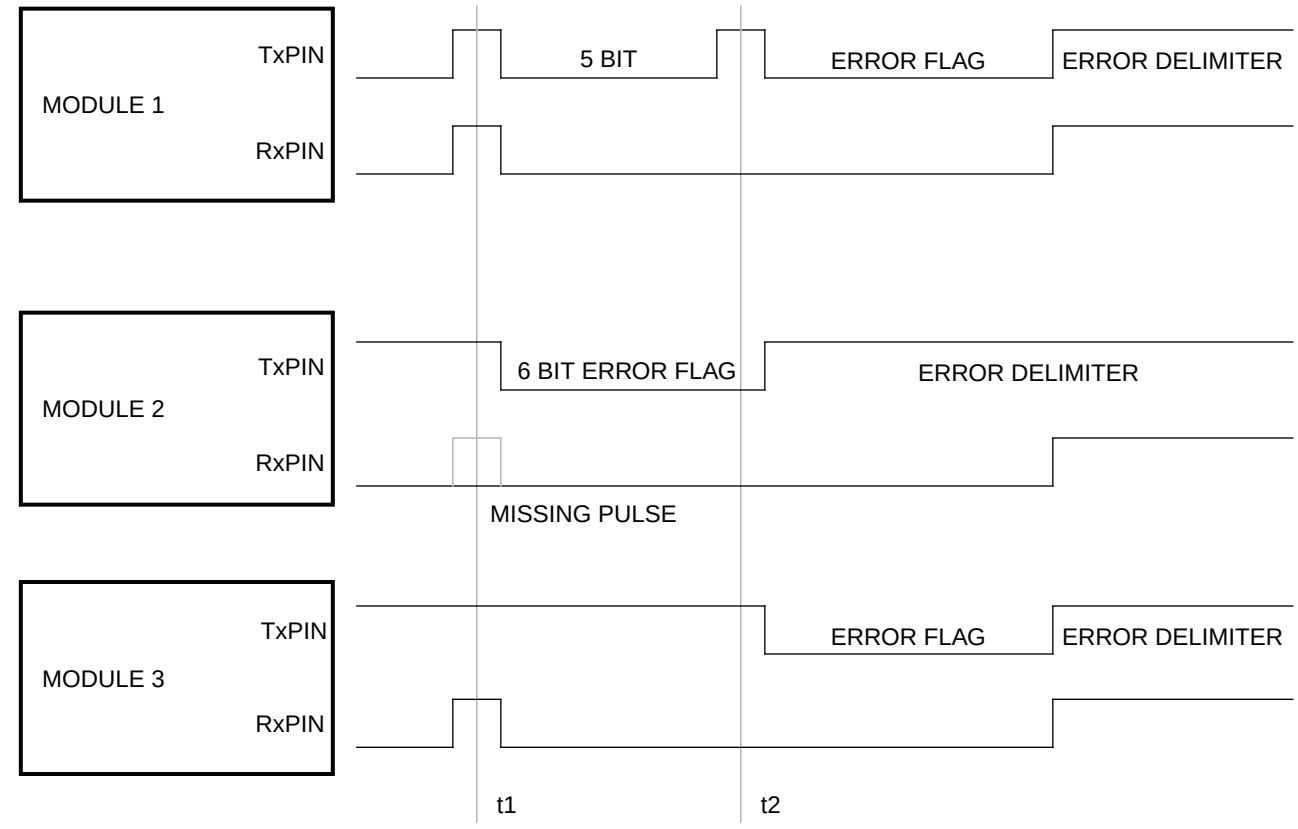
An error frame is generated when a module detects an error. This can occur at any time, including the middle of a data frame or the middle of another error frame.

An error frame consists of two fields: an error flag and an error delimiter. The error flag varies in length from six to twelve bits, depending on the error conditions. The error delimiter consists of eight recessive bits. After a module transmits its error flag, it waits for completion of the error delimiter before it continues.

When a module detects a bit error, stuff error, form error, or acknowledgement error, it starts transmission of the error flag at the very next bit. When a module detects a CRC error in the absence of any other type of error, it starts transmission of the error flag at the bit following the acknowledge delimiter.

Figure 11-7 shows how a local fault at one module (Module 2 in the figure) leads to a 12-bit error frame on the bus. Interruption of the transmitted frame triggers additional error frames at the receiver modules.

The bus level for an error flag can be dominant or recessive, depending on the error state of the module that detects the error.



module 1 = error active transmitter detects bit error at t2
module 2 = error active receiver with a local fault at t1
module 3 = error active receiver detects stuff error at t2

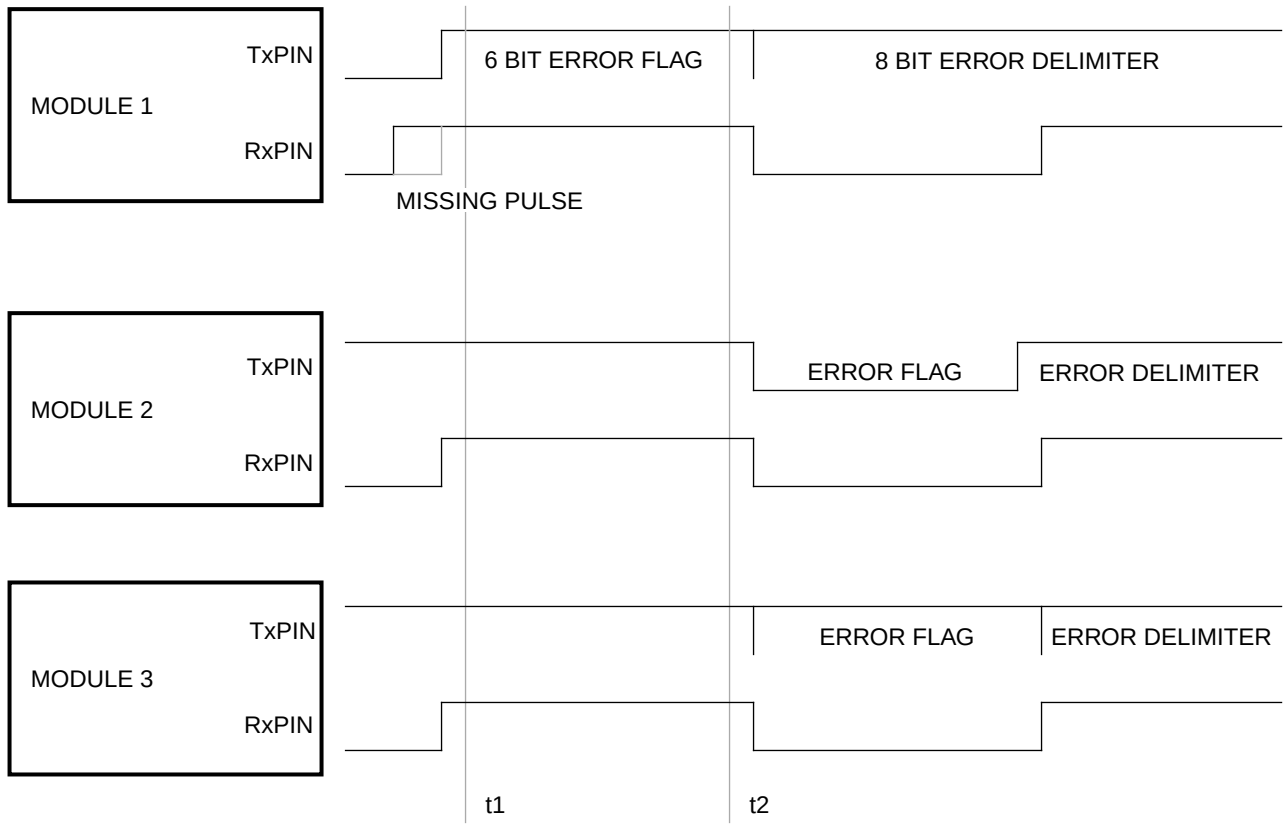
SPEC-06

Figure 11-7 Error Frame from Error-active Transmitter

An error-active module transmits an active error flag consisting of six dominant bits, which destroys the actual data frame placed on the bus. Other modules detect this condition as an error, either because of a bit-stuffing violation or destruction of a fixed-bit field. As a result, all other nodes start transmission of their own error flags. The error sequence that occurs on the bus has a maximum length of 12 bits: six from the module where the error occurred first plus six from other modules affected by the error.

An error-passive module transmits an error flag consisting of six recessive bits. If the message was originally transmitted by another module, the error frame does not destroy the message and is not detected by the other modules. However, if the module detecting the error is also the module that transmitted the original message, the error frame triggers errors in the other modules, either because of a bit-stuffing violation or destruction of a fixed-bit field.

Figure 11-8 shows how an error frame generated by an error-passive transmitter (Module 1 in the figure) is detected by receiver modules.



module 1 = error active receiver with a local fault at t1
module 2 = error passive transmitter detects bit error at t2
module 3 = error passive receiver detects stuff error at t2

SPEC-07

Figure 11-8 Error Frame from Error-passive Transmitter

11.3.6 Overload Frames

A module generates an overload frame when it is unable to complete a required operation quickly enough to respond to the next possible message on the bus. The overload frame is a signal to all modules on the bus to delay transmission of the next message.

The format of an overload frame is similar to that of an error frame. It begins with an overload flag and ends with an overload delimiter. The overload flag consists of six dominant bits and overload delimiter consists of eight recessive bits. Between the overload flag and the overload delimiter can be additional bits, the number depending on whether other modules superimpose overload frames over the first one. There can be a maximum of two consecutive overload frames on the bus.

An overload frame can only be transmitted immediately after the end of another frame. Therefore, it destroys the fixed form of the intermission field of the interframe space that normally separates successive data transmission frames.

11.3.7 Frame Validation

All frames are checked for errors by all modules in the network. The point at which a frame is considered valid depends on whether the frame is being transmitted or received.

For a transmitter, a frame is considered valid if no error detected through the end of the frame, including the last bit of the End of Frame field. For a receiver, a frame is considered valid if no error is detected through the second-to-last bit in the End of Frame field.

11.4 FRAME ARBITRATION AND PRIORITY

Each module is allowed to start transmission of a frame after the intermission field of an interframe space (except for the case of an error-passive module that transmitted the most recent frame). There is nothing to prevent two or more modules from starting transmission at the same time.

To prevent a module from destroying another module's data transmission frame, each module monitors the bus during transmission of the arbitration field. If a module detects a dominant bit on the bus while it is transmitting a recessive bit, it recognizes this condition as a higher-priority message being transmitted on the bus. As a result, it immediately releases the bus, stops transmission, and starts receiving the higher-priority message.

The 11 identifier bits in the arbitration field identify the message contained in the frame. The lower the identifier value, the higher the priority of the message with respect to other message in the network. Identifier values can range from 0x000 (highest priority) to 0x7EF (lowest priority).

11.5 ACCEPTANCE FILTERING

Every module performs "acceptance filtering" on the identifier field of each data frame and each remote frame transmitted on the bus. In other words, each module examines the identifier field to determine whether it needs the message being transmitted in the frame. A module stores transmitted data in its data buffers only if the frame identifier matches the acceptance filter.

If the frame identifier does not match a module's acceptance filter, the module ignores the message. However, every module acknowledges every frame received with a correct CRC sequence (except for modules in the bus-off error state), whether or not the message is used.

11.6 ERROR MANAGEMENT AND DETECTION

The CAN protocol has multiple mechanisms to detect errors and to prevent a malfunctioning module from disabling activity on the bus. A malfunctioning module enters one of several error states. The error state depends on the number and type of errors detected by a module.

11.6.1 Error Types

A module can detect the following types of errors:

- Bit Error
- Stuff Error
- Form Error
- Bit CRC Error
- Acknowledgment Error

Bit Error

A transmitting CAN device monitors the bus while it transmits the frame. If a bit value monitored on the bus is different from the bit value sent, it triggers a bit error.

The reception of a dominant bit instead of a recessive bit is not interpreted as a bit error if it occurs during the transmission of a passive error flag, the stuffed bit stream of the arbitration flag, or the acknowledge slot.

Stuff Error

A stuff error is triggered by six consecutive identical bit values in a message field that should be coded with bit stuffing. Bit stuffing is used in data frames and remote frames in the fields that hold information for the interface: the start of frame, arbitration, control, data, and CRC fields.

Form Error

A form error is triggered when a fixed bit value in a frame is not the expected value (for example, the recessive CRC delimiter bit or the ACK delimiter bit).

For a receiver, a dominant bit received at the last bit of the End of Frame field (normally a recessive bit) does not trigger an error.

Bit CRC Error

A CRC error is triggered when a receiver performs the CRC polynomial division on the received CRC value and obtains a nonzero result.

Acknowledgment Error

An acknowledgment error is triggered when a transmitting module does not get an acknowledgment from any other node. The transmitter fills the ACK slot of the transmitted frame with a recessive bit and expects at least one other module to overwrite this bit with a dominant bit. The lack of a dominant bit on the bus in this slot is an acknowledgment error.

11.6.2 Error Counters

Each module has two error counters: one for counting transmitted data errors called the Transmit Error Counter (TEC) and one for counting received data errors called the Receive Error Counter (REC). The values in these two counters determine the error state of the module.

Each time the module detects a transmission error, it increments the Transmit Error Counter by one or by eight, depending on the type of error, until the count reaches the maximum value of 255. Each time the module successfully transmits or receives a valid frame, it decrements the same counter by one until the count is back to zero.

Similarly, each time the module detects an error in receiving data, it increments the Receive Error Counter by one or by eight, depending on the type of error, until the count reaches the value 127. Each time the module successfully transmits or receives a valid frame, it decrements the same counter by one until the count is back to zero. If the count exceeds 127, the count value in the register is frozen.

Table 11-1 shows the conditions that cause the Transmit Error Counter to be incremented or decremented and the increment or decrement size for each condition. Similarly, Table 11-2 shows conditions that cause the Receive Error Counter to be incremented or decremented.

Table 11-1 Transmit Error Counter Handling

Condition	Transmit Error Counter
A transmitter detects a Bit Error during sending an active error flag	increment by 8
After detecting the 14th consecutive 'dominant' bit following an active error flag or overload flag or after detecting the 8th consecutive 'dominant' bit following a passive error flag. After each sequence of additional 8 consecutive 'dominant' bits.	increment by 8
Any other error condition (stuff, frame, CRC, Ack)	increment by 8
A valid reception or transmission	decrement by 1 unless counter is already 0

Table 11-2 Receive Error Counter Handling

Condition	Receive Error Counter
A receiver detects a Bit Error during sending an active error flag.	increment by 8
A receiver detects a 'dominant' bit as the first bit after sending an error flag	increment by 8
After detecting the 14th consecutive 'dominant' bit following an active error flag or overload flag or after detecting the 8th consecutive 'dominant' bit following a passive error flag. After each sequence of additional 8 consecutive 'dominant' bits.	increment by 8
Any other error condition (stuff, frame, CRC, Ack)	increment by 1
A valid reception or transmission	decrement by 1 unless counter is already 0

The following rules apply to the operation of the Transmit Error Counter:

- A bit-stuffing error occurs during arbitration when a transmitted recessive bit is received as a dominant bit. However, this condition does not cause the Transmit Error Counter to be incremented.
- An acknowledgment error occurs when an error-passive module sends a passive error flag and no dominant bits are detected on the bus. However, this condition does not cause the Transmit Error Counter to be incremented.
- If only one module is active on the bus and this module transmits a message, it will not receive an acknowledgment. This condition causes an acknowledgment error. As a result, the module attempts the same transmission repeatedly. The Transmit Error Counter is incremented with each unsuccessful attempt. After the module enters the error-passive mode, subsequent acknowledgment errors no longer cause the Transmit Error Counter to be incremented. This prevents the last active module on the bus from entering the bus-off state.

11.6.3 Module Error States

Depending on the numbers held in the error counters, a module can operate in any one of the following states:

- Normal (no error)
- Error-Active
- Error-Passive
- Bus-Off

When both counts are zero, the module operates normally. When one or both counts are between 1 and 127, the module operates in the error-active state. When REC overflows the maximum of 127 or when TEC is between 128 and 255, the module operates in the error-passive state. When the REC counter overflows, it has an effective value of 128 and remains at that level until it is decremented back to 127. When TEC overflows the maximum of 255, the module operates in the bus-off state. When the TEC counter overflows, it has an effective value of 256 and stays at that level until it is decremented back to 255.

The error states and the error counts that determine those states are illustrated in Figure 11-9.

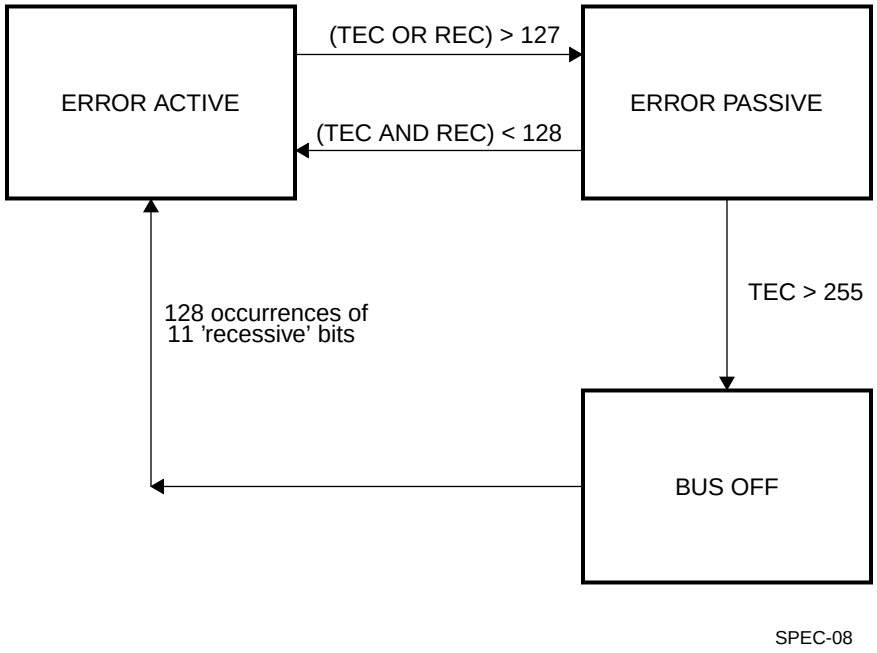


Figure 11-9 CANModule Error States

In the error-active state, a module can still participate in bus communications. If it detects another error while in this state, it sends an active (dominant) error flag.

A module operates in the error-active state when either of its two error counters is nonzero, and both error counts are less than 128.

Error-Passive State

In the error-passive state, a module can still participate in bus communications. However, if it detects another error while in this state, it sends a passive (recessive) error flag rather than an active (dominant) one.

A module operates in the error-passive state when the Receive Error Counter overflows the maximum value of 127 or when the Transmit Error Counter is between 128 and 255. A module sends an active error flag when it first enters the error-passive state.

Bus-Off State

In the bus-off state, a module has its output drivers disabled and therefore does not participate in bus communications. A module operates in this state when the Transmit Error Counter overflows the maximum value of 255.

Once a module enters the bus-off state, it can change back to the error-active state in one of two ways, depending on the fault confinement mode programmed by the application program. If the fault confinement mode selection bit (FMOD) in the CAN Bus Control Register (CBUS) is cleared to zero (the default mode upon power-up), the bus-off module operates in the "standard" fault confinement mode. If the FMOD bit is set to one instead, the bus-off module operates in the "enhanced" fault confinement mode.

In the standard fault confinement mode, the module monitors the bus and changes from the bus-off state to the error-active state after it detects 128 occurrences of 11 consecutive recessive bits, including recessive bits while the bus is idle.

In the enhanced fault confinement mode, the module monitors the bus and changes from the bus-off state to the error-active state after it detects 128 occurrences of "good" messages on the bus. A "good" message is a string of 11 consecutive recessive bits that includes an End of Frame bit (in other words, a successfully received data frame or remote frame).

The enhanced fault confinement mode ensures that a module in the bus-off state (a device with a serious fault) interferes with bus operations no more than once per 128 messages successfully transmitted on the bus. This is better than the standard fault confinement mode because when there is a lot of bus idle time, a bus-off module can frequently interfere with messages sent by other modules, thus slowing down operation of the network. However, in some cases it is necessary to use the standard fault confinement mode for compatibility with existing CAN systems.

With either type of fault confinement mode, when a bus-off module returns to the error-active state, both error counters are automatically reset to zero.

11.7 SYNCHRONIZATION

When the bus is in the idle state, any module can begin transmitting a frame at any time. The falling edge of the Start of Frame (SOF) bit on the bus is the signal to other modules that a frame transmission is starting. Each receiver module must synchronize its operation to that of the transmitter.

Each module has a prescaler clock that drives the logic of the module. The period of the prescaler clock is considered one "time quantum" (plural: "time quanta").

The time used to transmit each bit consists of a certain number of time quanta that depends on the design of the network and module. The time period of each bit is divided into four sections:

- Synchronization Segment: 1 time quantum
- Propagation Segment: 1 to 8 time quanta
- Phase Segment 1: 1 to 8 time quanta

- Phase Segment 2: 1 to 8 time quanta

The sections of a bit period are illustrated in Figure 11-10.

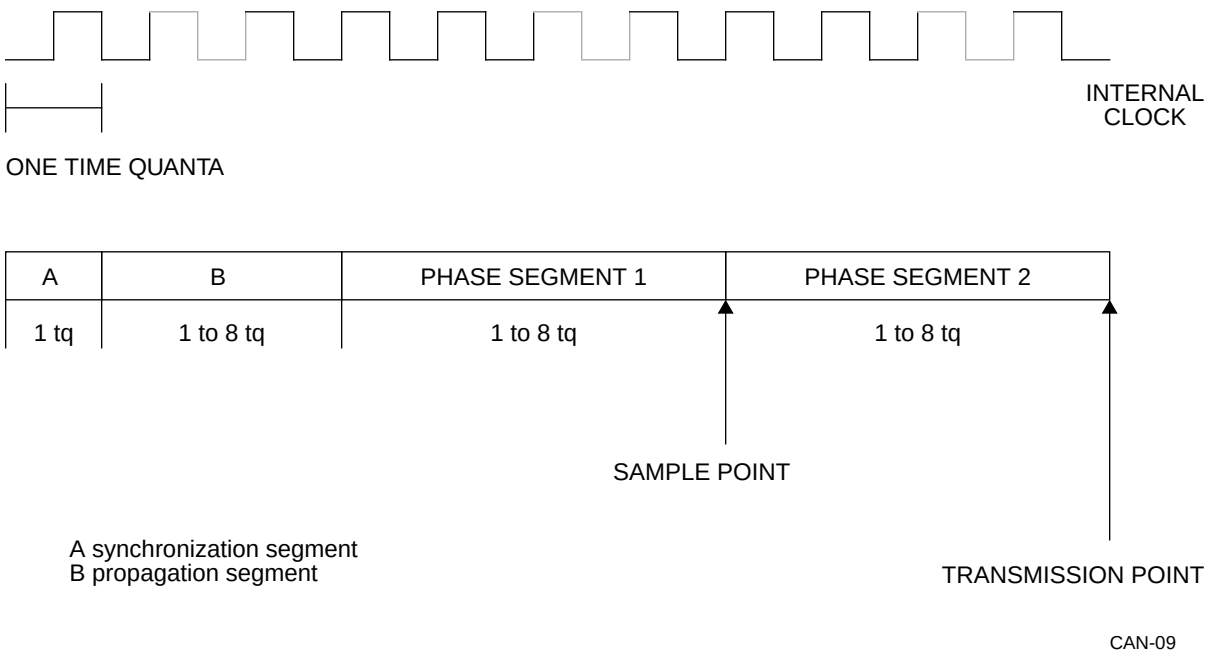


Figure 11-10 Bit Timing

The point at which a transmitter transmits a bit is at the beginning of the synchronization segment. The point at which a receiver samples the data is between phase segment 1 and phase segment 2.

When a receiver detects a falling edge on the bus for a Start of Frame bit, that edge should occur in the receiver's synchronization segment. There are two ways to synchronize a receiver to a transmitter, called "hard" and "soft" synchronization.

11.7.1 Hard Synchronization

Hard synchronization is based on hardware operation of the receiver. When a falling edge occurs on the bus when the bus is idle, the receiver interprets it as a Start of Frame being transmitted by another module. This event restarts the internal logic of the receiver, starting with a synchronization segment.

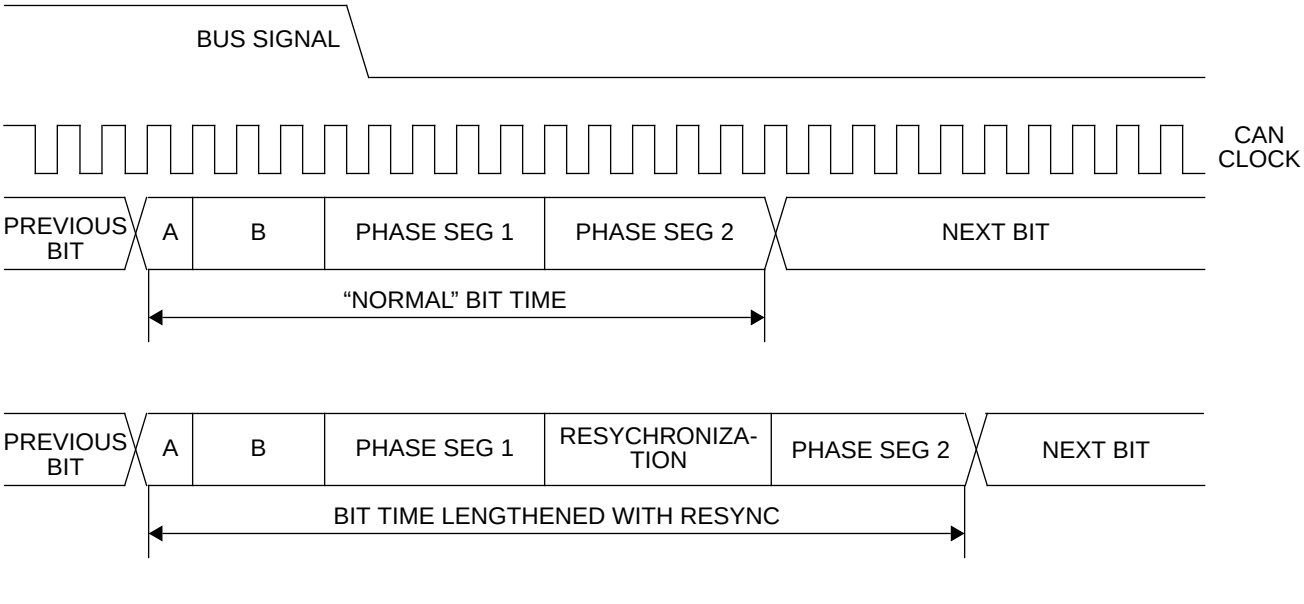
11.7.2 Soft Synchronization

Soft synchronization is based on resynchronizing the receiver under software control. The receiver adjusts its timing by a specified amount. Each such adjustment tends to make the next falling edge from the transmitter occur closer to the synchronization segment of the receiver. Only one resynchronization adjustment is allowed in any given bit period.

Whenever a falling edge is detected in the propagation segment or in phase segment 1, additional time is inserted between phase segment 1 and phase segment 2, thus

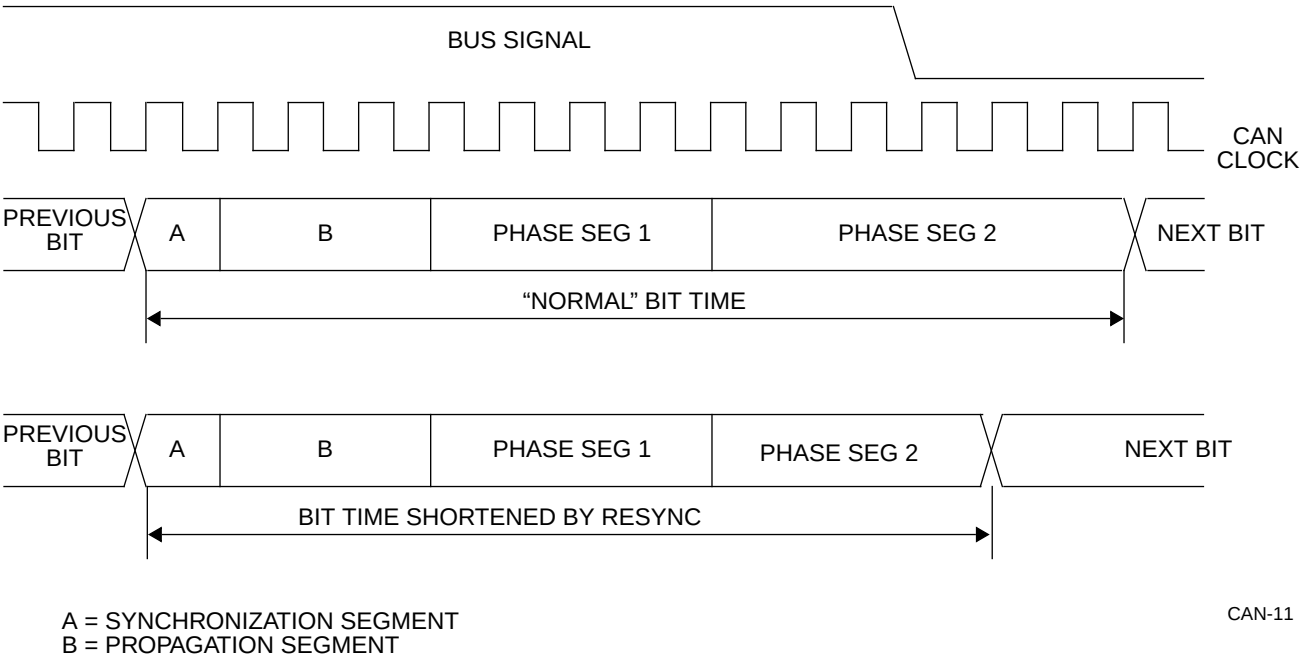
lengthening the time period for that bit. The amount of time added is called the resynchronization jump width. Figure 11-11 shows how this type of resynchronization is done.

Whenever a falling edge is detected in phase segment 2, then phase segment 2 is shortened by the resynchronization jump width, thus shortening the time period for that bit. Figure 11-12 shows how this type of resynchronization is done.



A = SYNCHRONIZATION SEGMENT
B = PROPAGATION SEGMENT

Figure 11-11 Resynchronization by Lengthening Bit Time



A = SYNCHRONIZATION SEGMENT
B = PROPAGATION SEGMENT

Figure 11-12 Resynchronization by Shortening Bit Time

11.8 COP888 CAN Interface Block

A COP888 device with a built-in CAN interface (for example, the COP888BE and COP884BC) contains a CAN serial bus interface that is optimized for low-speed (class A and B) applications as described in the CAN Specification Revision 2.0 part B. The COP888 device can serve as a bus module.

The COP888 device supports fully automatic transmission for messages up to two bytes. Messages longer than two bytes require software intervention. In that case, the application program must transfer the data between memory and the Tx/Rx registers. The software must either poll the Transmit Buffer Empty (TBE) and Receive Buffer Full (RBF) bits or enable their respective interrupts to determine when to transfer the data.

The COP888 device cannot receive or transmit extended frames. However, the device correctly acknowledges extended frames transmitted between other modules on the bus.

Figure 11-13 is a block diagram showing the functional blocks of the COP888 device's CAN interface hardware.

11.8.1 Interface Management Logic

The Interface Management Logic is a state machine that executes the CPU's transmission and reception commands and controls the transfer of data between the CPU and the Rx/Tx register. It provides the CAN interface with Rx/Tx data from the memory-mapped Register Block. It also sets and resets the CAN status information and generates CAN interrupts to the CPU.

11.8.2 Bit Stream Processor

The Bit Stream Processor is a sequencer that controls the data stream between the Interface Management Logic (parallel data) and the bus line (serial data). The Bit Stream Processor controls reception and arbitration of the Transceiver Logic and creates error signals according to the CAN bus specification.

11.8.3 Transceiver Logic

The Transceiver Logic is a state machine that controls the bit-stuff logic, output drivers, CRC logic, and Rx/Tx shift register. It also controls synchronization of the module to the bus using the bit rate clock from the Clock Generator.

11.8.4 Error Management Logic

The Error Management Logic is responsible for fault confinement according to the CAN protocol. It changes the error counters, sets the appropriate error counter flag bits and interrupts, and changes the error state of the module.

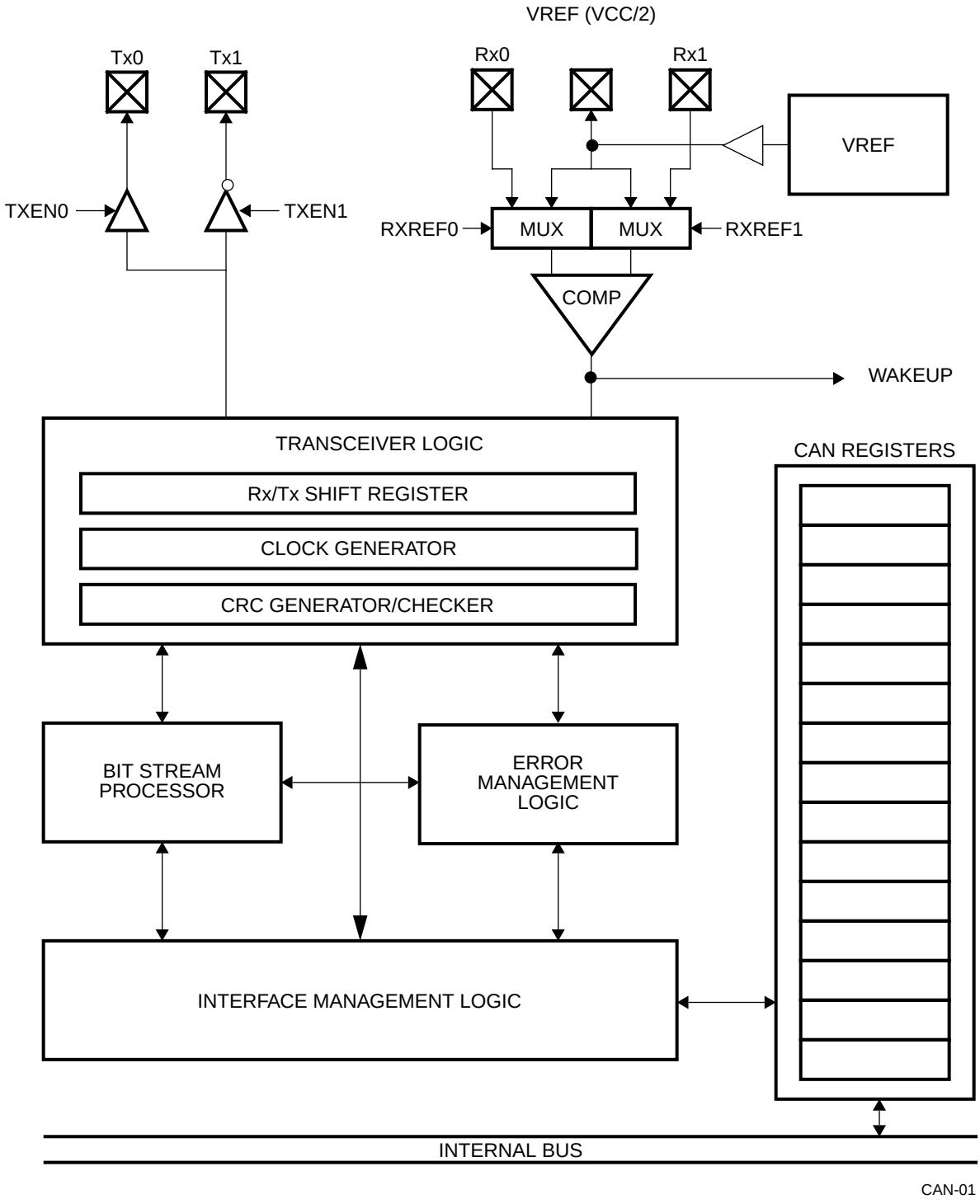


Figure 11-13 CAN Interface Block Diagram

11.8.5 CRC Generator and Register

The CRC Generator consists of a 15-bit shift register and the logic required to generate the checksum of the bit stream (without stuff bits). It generates the CRC bits for transmitting a frame and generates a checksum upon receiving a frame. It tells the Error Management Logic the result of a receiver checksum.

11.8.6 Receive/Transmit (Rx/Tx) Register

The Rx/Tx register is an 8-bit shift register controlled by the Transceiver Logic. It holds the data to be transmitted or the data that has been received. It is loaded or read by the Interface Management Logic.

11.8.7 Output Drivers and Input Comparators

A set of output drivers and input comparators serve as the physical interface to the bus. You can choose to either connect these bus input/output pins directly to the CAN bus or connect them to an external bus driver interface, depending on the system requirements.

Bus Connections

If you choose to connect the bus input/output pins directly to the CAN bus, the bus is implemented as two serial data lines, Tx0 and Tx1. For transmitting a dominant bit, output pin Tx0 drives the Tx0 line low and output pin Tx1 drives the Tx1 line high. For transmitting a recessive bit or for receiving, the device places both the Tx0 and Tx1 output pins in the TRISTATE (high impedance) mode, allowing the serial bus lines to be driven by an external module. The device never drives the Tx0 line high or the Tx1 line low.

In order to monitor the serial data lines and to receive data from other modules, the Rx0 input can be connected to the Tx0 data line and the Rx1 input can be connected to the Tx1 data line. At the point in time when the input data is read, the hardware compares the voltage on the data line against a reference voltage, VREF, which is one-half of the power supply voltage, VCC.

For the Tx0/Rx0 line, a voltage less than VREF indicates a dominant bit on the bus; a voltage greater than VREF indicates a recessive bit on the bus. For the Tx1/Rx1 line, a voltage greater than VREF indicates a dominant bit on the bus; a voltage less than VREF indicates a recessive bit on the bus.

Instead of connecting the bus input/output pins directly to the CAN bus, you can use an external transceiver chip between the COP888 device and the CAN bus lines as shown in Figure 11-14. In that case, the bus runs in differential mode while the COP888 CAN Interface operates in single wire mode, using TX0 and RX0 only. For this hardware configuration, the CBUS register must be configured for single-ended mode for TX0 and RX0 (TXEN1=0, TXEN0=1, RXREF1=1, RXREF0=0). RX1 must be connected to GND with a pull-down resistor, and for high-speed applications, TX0 needs a pullup resistor, as shown in the figure.

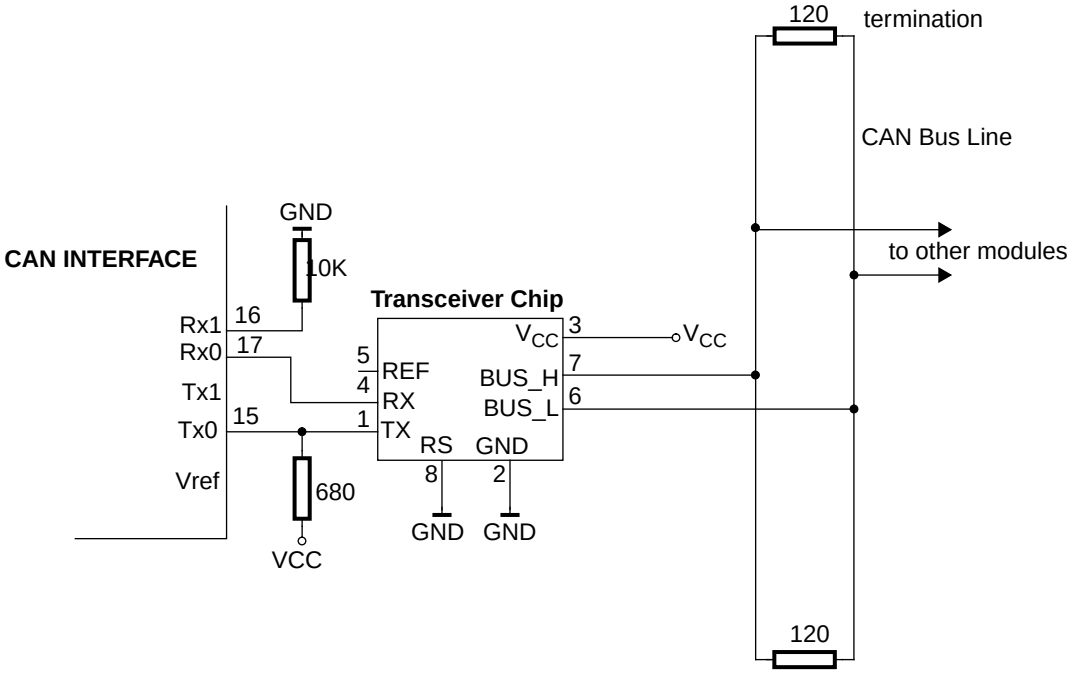


Figure 11-14 External Transceiver Connections (ISO High Speed)

On-Chip Voltage Reference

The on-chip voltage reference is a ratiometric reference. For electrical characteristics of the voltage reference, see the device data sheet.

Analog Switches

Analog switches are used for connecting the receive comparator inputs to either Rx0 or V_{REF} and either Rx1 or V_{REF}.

11.8.8 Clock Generator

The Clock Generator creates two clock signals: an internal CAN clock and a bit rate clock. The internal CAN clock is used by all CAN blocks in the COP888 device. The bit rate clock controls the rate at which bits are transmitted or received on the CAN network. Figure 11-15 shows how the two clocks are generated from the CKI clock.

The internal CAN clock is generated by dividing the CKI chip clock by a prescaler factor. The prescaler factor is 1 plus the value programmed into the CSCAL register.

The bit rate clock is generated by dividing the internal CAN clock. The bit transmission period consists of four sections:

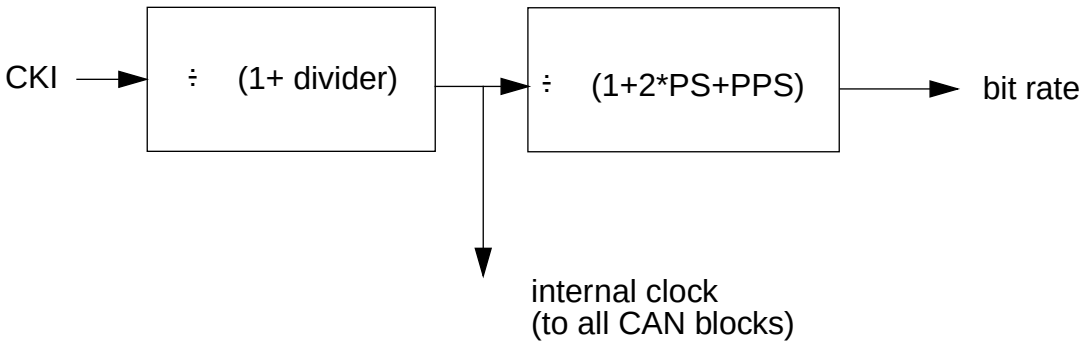


Figure 11-15 Bit Rate Generator

- Synchronization Segment: 1 internal CAN clock period
- Propagation Segment: 1 to 8 internal CAN clock periods
- Phase Segment 1: 1 to 8 internal CAN clock periods
- Phase Segment 2: 1 to 8 internal CAN clock periods

The duration of the propagation segment is programmed in the PPS2-PPS0 field of the CAN Bus Timing (CTIM) register. The duration of both phase segment 1 and phase segment 2 is programmed into the PS2-PS0 field of the same register. Each of these two fields, PPS and PS, can be set to any value from 0 to 7. The number of internal CAN clock periods used for each segment is 1 plus the 3-bit value written in the register field, which is a value from 1 to 8.

The total number of internal CAN clock periods per bit is:

$1 + (1+PPS) + (1+PS) + (1+PS) = 4+PPS+2*PS$

NOTE: For soft resynchronization of a receiver, the number of internal CAN clock periods can be adjusted upward or downward. See the description of the CTIM register for details.

The frequency of the bit rate clock is:

$f_{CAN} = f_{CKI} / [(1+CSCAL)*(1+PPS+2*PS)]$

NOTE: The frequency of the bit rate clock (f_{CAN}) should not exceed 1.0 MHz.

Bus Timing Considerations

The internal architecture of the CAN interface is optimized to allow fast response times within messages of more than two data bytes. The software can send data to the transmit buffer two bytes at a time.

The hardware sets the Transmit Buffer Empty (TBE) bit in the Receive/Transmit Status (RTSTAT) register at the end of every odd data byte when the internal sample strobe is high. This event can be used to trigger an interrupt. The interrupt service routine should

write the next two data bytes to be transmitted. Figure 11-16 shows the timing of the Transmit Buffer Empty bit.

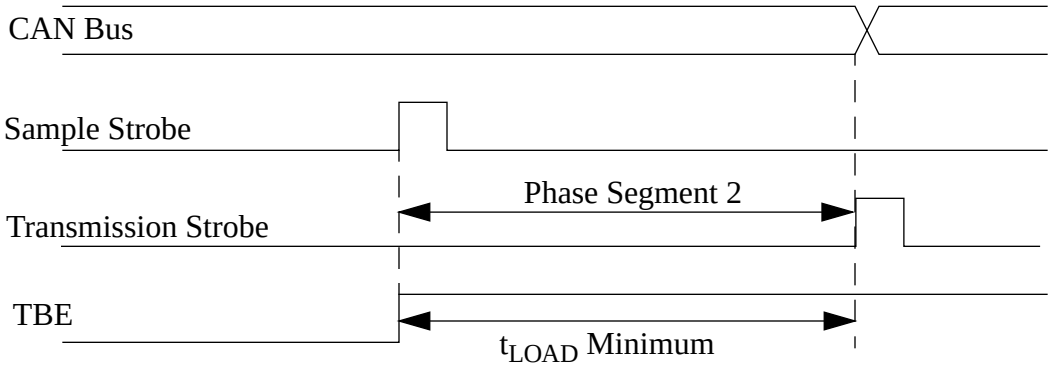


Figure 11-16 Transmit Buffer Empty Timing

The application software must be written so that the time from setting the TBE bit to the reloading of the Transmit Data Register 2 (TXD2) is longer than the length of phase segment 2:

$$t_{LOAD} > \frac{(PS + 1) \times (CSCAL + 1)}{10} t_c \quad = \text{ absolute length of PS2}$$

Otherwise, the register will be overwritten before the data can be transmitted.

Table 11-3 shows some examples of the minimum required t_{LOAD} time for a CKI clock frequency of 10 MHz, a PS setting of 4, and various settings of CSCAL. The CAN bit rates shown in the table are based on a PPS setting of 1.

Table 11-3 CAN Timing (CKI = 10 MHz t_C = 1µS)

PS	CSCAL	CAN bit rate (kbit/s)	Minimum t_{LOAD} (µS)
4	0	1000	0.5
4	1	500	1.0
4	3	250	2.0
4	9	100	5.0
4	15	62	8.0
4	24	40	12.5
4	39	25	20
4	99	10	50
4	199	5	100

You can choose to use interrupts or polling to control the timing of loading the transmit data registers. An interrupt-driven program requires more time to process the data than a program that polls the TBE flag. However, programs that process data at lower baud rates have more time available for interrupt response.

In the case of an interrupt-driven CAN interface, the actual t_{LOAD} time can be calculated as follows:

```
INT:              ;Interrupt latency = 7tC = 7µS
     PUSH A       ;3tC = 3 µS
     LD   A,B     ;2tC = 2 µS
     PUSH A       ;3tC = 3 µS
     VIS          ;5tC = 5 µS
CANTX:            ;20tC = 20 µS to this point
     .            ;additional time for instructions which check
     .            ;status prior to reloading the transmit data
     .            ;registers with subsequent data bytes.
     LD   TXD2,DATA
     .
     .
     .
```

11.8.9 CAN Register Block

The register block consists of 15 8-bit registers used with the CAN interface. These registers are accessible to the CPU in the address range of X'00B0 through X'00BF in the COP888BC or . X'00A0 through X'00AF in the COP888EB (see device-specific chapters for exact address assignments.)

The registers are listed in Table 11-4 and described in the following subsections.

Table 11-4 CAN Interface Registers

Register Name	Abbreviation
Transmit Data Register 1	(TXD1)
Transmit Data Register 2	(TXD2)
Transmit Data Length Code and Identifier Low Register	(TDLC)
Transmit Identifier High Register	(TID)
Receive Data Register 1	(RXD1)
Receive Data Register 2	(RXD2)
Receive Data Length Code and Identifier Low Register	(RIDL)
Receive Identifier High	(RID)
CAN Prescaler Register	(CSCAL)

Table 11-4 CAN Interface Registers

Register Name	Abbreviation
CAN Bus Timing Register	(CTIM)
CAN Bus Control Register	(CBUS)
Transmit Control/Status Register	(TCNTL)
Receive/Transmit Status Register	(RTSTAT)
Transmit Error Counter	(TEC)
Receive Error Counter	(REC)

Transmit Data Register 1 (TXD1)

The Transmit Data Register 1 contains the first data byte to be transmitted and then successive odd bytes of data to be transmitted in a frame (in other words, bytes 1, 3, 5, 7).

Transmit Data Register 2 (TXD2)

The Transmit Data Register 2 contains the second data byte to be transmitted and then successive even bytes of data to be transmitted in a frame (in other words, bytes 2, 4, 6, 8).

Transmit Data Length Code and Identifier Low Register (TDLC)

Table 11-5 Transmit Data Length Code and Identifier Low register (TDLC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TID3	TID2	TID1	TID0	TDLC3	TDLC2	TDLC1	TDLC0

The Transmit Data Length Code and Identifier Low register is a read/write register used to specify the number of data bytes to be transmitted in a frame and four bits of the transmit identifier.

Bits TID3-TID0 are the lower four bits of the transmit identifier. The remaining seven bits of the identifier are stored in the TID register.

Bits TDLC3-TDLC0 specify the number of data bytes to be transmitted within the frame. Table 11-6 shows the valid bit combinations and the corresponding number of data bytes transmitted for each combination.

Table 11-6 Transmit Data Length Codes

Number of Data Bytes	TDLC3	TDLC2	TDLC1	TDLC0
0	0	0	0	0
1	0	0	0	1

Table 11-6 Transmit Data Length Codes

Number of Data Bytes	TDLC3	TDLC2	TDLC1	TDLC0
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

Data length codes greater than eight are considered invalid in the CAN specification. Transmission of a data length code greater than eight will not cause an error on the bus. However, the number of data bytes that can be transmitted in one frame without error is limited to eight. If the transmitted data length code is greater than eight, the data bytes sent will be invalid.

Transmit Identifier High Register (TID)

Table 11-7 Transmit Identifier High (TID)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRTR	TID10	TID9	TID8	TID7	TID6	TID5	TID4

The Transmit Identifier High register is a read/write register used to specify the upper seven bits of the 11-bit transmit identifier and to specify whether the frame is a data frame or a remote frame.

The highest-order bit, Transmit Remote Frame (TRTR), is set to 1 for a remote frame or to 0 for a data frame.

The remaining bits, TID10-TID4, specify the upper seven bits of the 11-bit transmit identifier. The four lower bits are specified in the TDLC register.

Receive Data Register 1 (RXD1)

The Receive Data Register 1 contains the first data byte received in a frame and then successive odd bytes of data received in the same frame (in other words, bytes 1, 3, 5, 7). This is a read-only register.

Receive Data Register 2 (RXD2)

The Receive Data Register 2 contains the second data byte received in a frame and then successive even bytes of data received in the same frame (in other words, bytes 2, 4, 6, 8). This is a read-only register.

Receive Data Length Code and Identifier Low Register (RIDL)

Table 11-8 Receive Data Length Code and Identifier Low register (RIDL) (00B6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RID3	RID2	RID1	RID0	RDLC3	RDLC2	RDLC1	RDLC0

The Receive Data Length Code and Identifier Low register is a read-only register that specifies the number of data bytes received in a frame and four bits of the receive identifier contained in the frame.

Bits RID3-RID0 are the lower four bits of the receive identifier. The remaining seven bits of the identifier are stored in the RID register.

Bits RDLC3-RDLC0 specify the number of data bytes within the received frame. Table 11-9 shows the valid bit combinations and the corresponding number of data bytes received for each combination.

Table 11-9 Transmit Data Length Codes

Number of Data Bytes	TDLC3	TDLC2	TDLC1	TDLC0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

Data length codes greater than eight are considered invalid in the CAN specification. Reception of a data length code greater than eight will not cause an error on the bus. However, the number of data bytes that can be received in one frame without error is limited to eight. If the received data length code is greater than eight, the data bytes received will not be usable data.

Receive Identifier High (RID)

Table 11-10 Receive Identifier High

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
unused	RID10	RID9	RID8	RID7	RID6	RID5	RID4

The Receive Identifier High register is a read/write register that contains the upper seven bits of the 11-bit receive identifier of the received frame.

The highest-order bit of this register is not used. The remaining bits, RID10-RID4, specify the upper seven bits of the 11-bit receive identifier. The four lower bits are contained in the TDLC register.

This register can be used as a filter to prevent unwanted messages from being accepted by the module. If the Receive Identifier Acceptance Filter bit (RIAF) in the CAN Bus Control (CBUS) register is set to zero (the default mode upon power-up or reset), the hardware compares bits 4 through 10 of the received identifier with the mask bits written to the RID register. If the corresponding bits match, the message is accepted. Otherwise, the message is not accepted and the contents of the receiver-related registers (RXD1, RXD2, RIDL, RID, and RTSTAT) are left unchanged.

Only the seven higher-order bits of the identifier are used for filtering purposes. The application software can read the four lower-order bits in the RIDL register to select between different functions handled by the module (for example, speedometer, clock, and oil pressure display functions implemented in one dashboard CAN controller).

If the RIAF bit in the CBUS register is set to 1, the filter function is disabled and all messages are accepted. In that case, the receiver-related registers are updated by all messages received.

CAN Prescaler Register (CSCAL)

Table 11-11 CAN Prescaler Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CKS7	CKS6	CKS5	CKS4	CKS3	CKS2	CKS1	CKS0

The CAN Prescaler Register is a read/write register that specifies the prescaler factor for generating the internal CAN clock from the CKI hardware clock. The prescaler factor is 1 plus the value written to the register. Thus, the prescaler factor can be any value from 1 to 256. The frequency of the internal CAN clock is equal to the frequency of the CKI clock divided by the prescaler factor.

CAN Bus Timing Register (CTIM)

Table 11-12 CAN Bus Timing Register (CTIM)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PPS2	PPS1	PPS0	PS2	PS1	PS0	Reserved	Reserved

The CAN Bus Timing Register is a read/write register that specifies the lengths of the propagation segment, phase segment 1, and phase segment 2 of the CAN bus clock period. For more information on these segments, see the Section 5.7, Synchronization. For information on the CAN bit rate that you get from programming the CTIM register, see Section 5.8.8, Clock Generator.

The PPS2-PPS0 bits specify the number of internal CAN clock cycles that are used to make the propagation segment. This number can range from 1 to 8, equal to 1 plus the 3-bit value written to the field.

Similarly, the PS2-PS0 bits specify the number of internal CAN clock cycles that are used to make phase segment 1 and phase segment 2. This number can range from 1 to 8, equal to 1 plus the 3-bit value written to the field. Both phase segments are set to the specified value.

The Synchronization Jump Width is the number of internal CAN clock cycles that the hardware uses to lengthen or shorten the CAN bit period when synchronizing the operation of the receiver to a transmitter. The Synchronization Jump Width used by the hardware is either the programmed length of phase segment 1 or the value 4, whichever is smaller.

Setting phase segments 1 and phase segment 2 to one internal CAN clock cycle (setting PS2-PS0 to 000) is not recommended because it limits the range of acceptable baud rates. The result of programming a length of one internal CAN clock cycle allows lengthening of the bit time if the bus edge falls in the propagation segment or in phase segment 1, but does not allow for shortening the bit time when the edge falls in phase segment 2.

CAN Bus Control Register (CBUS)

Table 11-13 CAN Bus Control Register (CBUS)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	RIAF	TXEN1	TXEN0	RXREF1	RXREF0	Reserved	FMOD

The CAN Bus Control register is a read/write register used to control various programming options related to the CAN bus.

Bit 7 is reserved for testing. During normal operation, it can be used as a general-purpose flag bit.

The Receive Identifier Acceptance Filter (RIAF) bit determines whether the receiver performs filtering of messages. If this bit is set to zero (the default mode upon power-up or reset), the hardware compares bits 4 through 10 of the received identifier with the mask bits written to the RID register. If the corresponding bits match, the message is

accepted. Otherwise, the message is not accepted and the contents of the receiver-related registers (RXD1, RXD2, RIDL, RID, and RTSTAT) are left unchanged. If the RIAF bit is set to 1, the filter function is disabled and all messages are accepted.

The TxD Output Driver Enable bits (TXEN0 and TXEN1) are used to enable or disable the respective output driver pins, Tx0 and Tx1. When one of these bits is set to 1, the corresponding output driver pin is enabled. When one of these bits is set to 0, the corresponding output driver pin is disabled by placing the pin in the TRISTATE (high-impedance) mode. The bit functions are summarized in Table 11-14.

Table 11-14 Output Drivers

TXEN1	TXEN0	Output
0	0	Tx0, Tx1 TRI-STATED, CAN input comparator disabled
0	1	Tx0 enabled
1	0	Tx1 enabled
1	1	Tx0 and Tx1 enabled

Setting both TXEN1 and TXEN0 to 0 disables both output drivers and also disables the CAN input comparator, without affecting other CAN-related registers and flags.

If both output drivers are in the disabled state and then one or both are enabled, the device waits until 11 consecutive recessive bits have been received before the device starts transmitting or receiving any frames. This behavior synchronizes operation of the device to the bus.

It is recommended that the software reset the TXEN1 and TXEN0 bits before it switches the device into the HALT mode. Doing this reduces power consumption and ensures proper resynchronization to the bus when the device later exits the HALT mode. The CAN recessive wakeup feature still works with the output drivers disabled.

Note: Entering the bus-off error state causes the two output drivers to be placed in the TRISTATE mode, even if the TXEN1 and TXEN0 bits are set to 1.

The Receiver Reference Voltage control bits (RXREF1 and RXREF0) determine whether the VREF reference voltage is applied to the receiver multiplexers for Rx0 and Rx1. The Receiver Reference Voltage control bits (RXREF1 and RXREF0) determine whether the VREF reference voltage is applied to the receiver multiplexers for Rx0 and Rx1. If the bit is set, VREF is applied to the applicable multiplexer. For differential mode operation, both bits should be cleared. For single-ended operation, the bit corresponding to the input not used for the CAN interface should be set. These bit settings do not affect the CAN wakeup circuit.

Bit 1 is reserved and should always be cleared to zero.

The Fault Confinement Mode (FMOD) selection bit determines the operating mode for the module in the bus-off error state: 0 selects the "standard" mode and 1 selects the "enhanced" mode. In the standard mode, the module monitors the bus and changes from

the bus-off state to the error-active state after it detects 128 occurrences of 11 consecutive recessive bits, including recessive bits while the bus is idle. In the enhanced mode, the module operates in the same way, except that it looks for 128 occurrences of "good" messages. A "good" message is a string of 11 consecutive recessive bits that includes an End of Frame bit (in other words, a successfully received data frame or remote frame).

The enhanced fault confinement mode ensures that a module in the bus-off state (a device with a serious fault) interferes with bus operations no more than once per 128 messages successfully transmitted on the bus. This is better than the standard fault confinement mode because when there is a lot of bus idle time, a bus-off module can frequently interfere with messages sent by other modules, thus slowing down operation of the network. However, it may be necessary to use the standard fault confinement mode for compatibility with existing CAN systems.

Transmit Control/Status Register (TCNTL)

Table 11-15 Transmit Control/Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NS1	NS0	TERR	RERR	CEIE	TIE	RIE	TXSS

The Transmit Control/Status register is used to control the transmitter and to read its status. The two highest-order bits (the Node Status bits) are read-only bits. The remaining bits can be read and written.

The Node Status (NS1-NS0) bits indicate the current error status of the module as indicated in Table 11-16.

Table 11-16 Node Status

NS1	NS0	Status
0	0	Error-active or no error
0	1	Error-passive
1	0	Bus-off (transitional state)
1	1	Bus off

The Transmit Error (TERR) bit is automatically set when an error occurs during transmission of a frame. This event can be programmed to generate an interrupt by setting the CAN Error Interrupt Enable (CEIE) bit. After an error occurs, the software must clear the TERR bit.

After a transmission error, the frame is automatically transmitted again if the frame has two or fewer data bytes. For a frame with more than two data bytes, the application software must handle reloading of the data bytes into the TxD registers for retransmission.

Note: Whether the error is handled automatically by the hardware or by the software, the software must clear the TERR bit if the CEIE bit is set. Otherwise, the TERR bit remains set, resulting in an endless transmit error interrupt loop.

The Receive Error (RERR) bit is automatically set when an error occurs during reception of a frame. This event can be programmed to generate an interrupt by setting the CAN Error Interrupt Enable (CEIE) bit. After an error occurs, the software must clear the RERR bit.

The CAN Error Interrupt Enable (CEIE) bit, if set, enables both the transmit and receive error interrupts. The interrupt pending flags are TERR and RERR.

The Transmit Error Interrupt Enable (TIE) bit, if set, enables the transmit error interrupt. The interrupt pending flag is TBE.

The Receive Interrupt Enable (RIE) bit, if set, enables the receive interrupt. The interrupt pending flag is RBF, RRTR, or RFV.

Resetting an interrupt enable bit (CEIE, TIE, or RIE) with a pending interrupt inhibits the interrupt but does not clear the interrupt flag. If the interrupt enable bit is set to 1 again without first clearing the pending interrupt flag, the interrupt will occur again.

The transmission Start/Stop (TXSS) bit can be set to 1 to start transmission of a frame. Once this bit is set, the transmission is pending, as indicated by the TXPND flag in the RTSTAT register. If one or more data bytes are to be transmitted, the software must load the Transmit Data Register(s) before setting the TXSS bit.

The software can request cancellation of a pending transmission by writing a 0 to the TXSS bit. Cancellation of a pending transmission works under any of the following conditions: the transmission has not yet started and the bus is idle; arbitration is lost and the module is receiving; a transmission error occurs. If the module has already started transmission and won arbitration, the transmission is completed.

When the software writes a 0 to the TXSS bit, it is a request for cancellation of a pending transmission that does not actually clear the flag bit. The TXPND and TXSS flags stay set until the transmission is completed or the transmission is successfully cancelled.

There are only three events that cause the TXSS to be cleared:

- Successful completion of a transmitted message
- Successful cancellation of a pending transmission
- Transition of the module to the bus-off error state

Receive/Transmit Status Register (RTSTAT)

Table 11-17 Receive/Transmit Status (RTSTAT)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TBE	TXPND	RRTR	ROLD	RORN	RFV	RCV	RBF

The Receive/Transmit Status register is a read-only register that indicates the status of the current receive or transmit operation.

The Transmit Buffer Empty (TBE) bit is set when the TXD2 register is copied into the Rx/Tx shift register, or in other words, just after the first byte of a pair of data bytes has been transmitted. It is a signal to reload the TXD1 and TXD2 register. This bit is automatically reset when the TXD2 register is written.

The setting of the TBE bit can be programmed to generate an interrupt by setting the Transmit Interrupt Enable (TIE) bit. The interrupt service routine must write to the TXD2 register, which clears the TBE bit. Otherwise, another interrupt will occur immediately after completion of the interrupt service routine. When transmitting an odd number of bytes or no bytes, the software should write a dummy byte to the TXD2 register.

The TBE bit is set to 1 upon power-up or reset. It is also set upon completion of transmission of a valid message. The total of number of TBE interrupts per frame transmitted depends on the number of bytes in the message. For example:

- A valid 8-byte message generates five interrupts: one upon transfer of each even byte (bytes 2, 4, 6, 8) to the Rx/Tx shift register and one on End of Frame.
- A valid 5-byte message generates three interrupts: one upon transfer of each even byte (bytes 2 and 4) to the Rx/Tx shift register and one on End of Frame.
- A valid 0-byte message generates one interrupt on End of Frame.

The Transmission Pending (TXPND) bit is set by the hardware as soon as the software sets the Transmit Start/Stop (TXSS) bit in the TCNTL register. The TXPND bits stays set until one of the following occurs: the frame is successfully transmitted, the transmission is successfully canceled by the software clearing the TXSS bit, or the module enter the bus-off error state.

The Receive Mode (RCV) bit is set by the hardware after the data length code of a received message passes through the device's acceptance filter. It is automatically reset after the CRC delimiter of the same frame has been received.

The Receiver Overrun (RORN) bit is set by the hardware when there is an overrun of the Receive Data Register, or in other words, when the software fails to maintain the RxDn registers when receiving a frame. It is automatically reset when the software reads the RTSTAT register. The software must read the RTSTAT register to clear the RORN bit before the next frame is received.

The Received Overload Frame (ROLD) bit is set by the hardware when an overload frame is received on the bus. It is automatically reset when the software reads the RTSTAT

register. The software must read the RTSTAT register to clear the ROLD bit before the next frame is received.

The Received Remote Transmission Request (RRTR) bit is set by the hardware when the data length code is received for a remote frame. A remote frame has its Remote Transmission Request (RTR) bit set. The RRTR bit is automatically reset when the software reads the RXD1 register.

To detect a Received Remote Transmission Request, the software can either poll the RRTR flag or enable the Receive Interrupt. If the Receive Interrupt is enabled, the interrupt service routine should check the RRTR flag to determine whether a Received Remote Transmission Request or a Receive Buffer Full condition caused the interrupt. If the RRTR flag is set, the software must clear the flag by reading the RXD1 register before the next frame is received.

The Received Frame Valid (RFV) bit is set by the hardware when the second-to-the last bit of the End of Frame field is received, which indicates a valid frame. The RFV bit is automatically reset when the software reads the RTSTAT register. The software must clear the flag by reading the RTSTAT register before the next frame is received. The Received Frame Valid condition causes a Receive Interrupt if enabled by the RIE bit in the TCNTL register. Be sure to have the software read the last data byte of odd-length messages (1, 3, 5, or 7 data bytes) from RXD1 when a Received Frame Valid occurs. The RFV flag is the only indication that the last byte of the message has been received.

The Receive Buffer Full (RBF) bit is set by the hardware when the second data byte of a pair of data bytes is received. The RBF bit is automatically reset when the software reads the RXD1 register. The Receive Buffer Full condition causes a Receive Interrupt if enabled by the RIE bit in the TCNTL register. The interrupt service routine must clear the RBF bit by loading the RXD1 register. Otherwise, a new interrupt will be generated immediately upon return from the interrupt service routine.

Figure 11-17 shows the state of the receiver status bits during reception of a data frame and a remote frame.

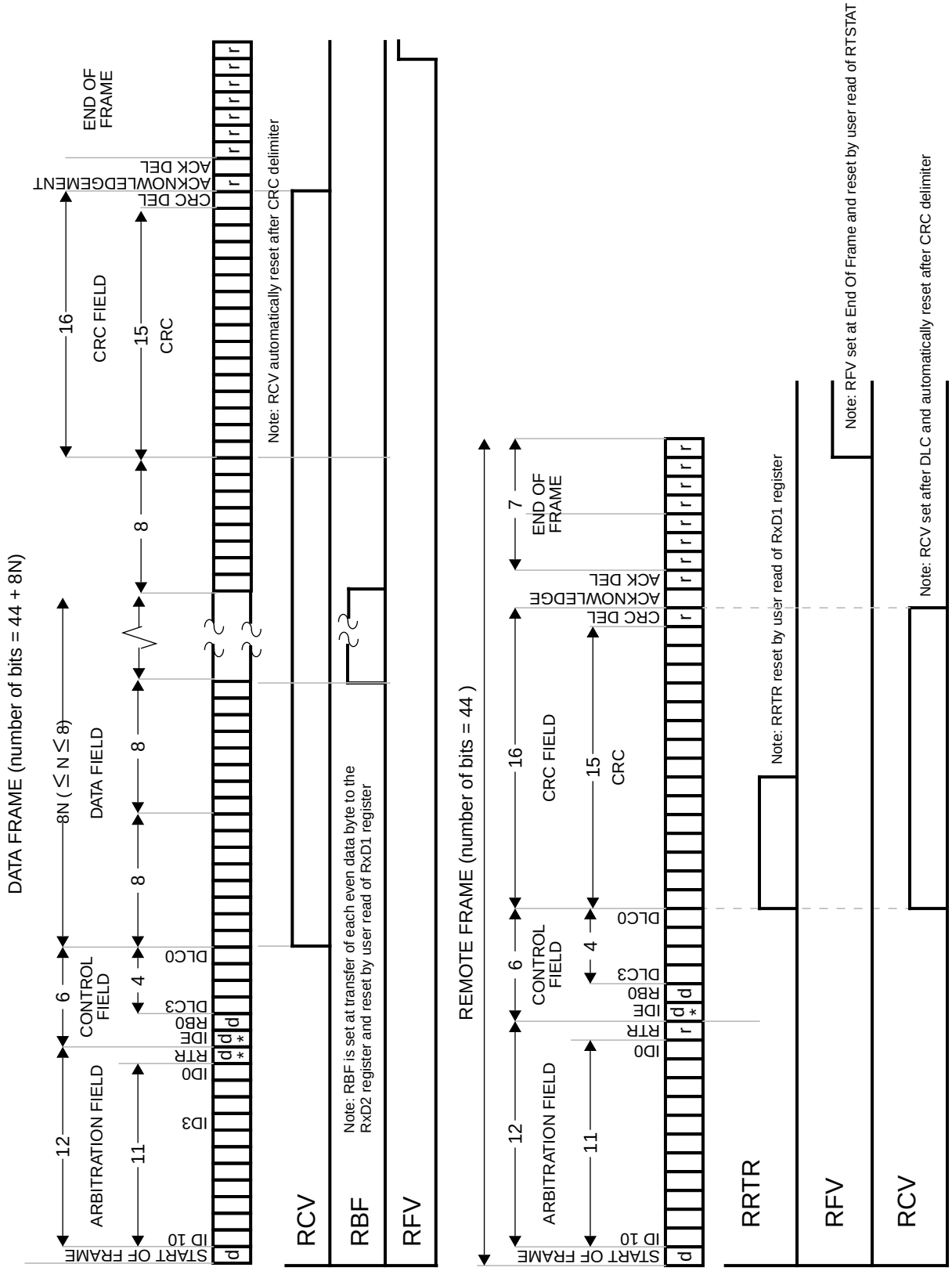


Figure 11-17 Receiver Status Bit Waveforms

Transmit Error Counter (TEC)

Table 11-18 Transmit Error Counter Register (TEC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0

The Transmit Error Counter is a read/write register that automatically keeps track of the number of transmit errors. The value in the counter determines the error state of the device. If this value is 1 or more, the device enters the error-active state. If this value is 128 or more, the device enters the error-passive state. If this value overflows, the device enters the bus-off state.

Writing the value zero to the TEC register resets the CAN interface. If the software does this, it should also write the value zero to the Receive Error Counter.

Note: The software should never modify the contents of the Transmit Error Counter or Receive Error Counter, except to reset the CAN interface. Modifying the counter values interferes with normal operation of CAN fault confinement.

Receive Error Counter (REC)

Table 11-19 Receive Error Counter Register (REC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ROVL	REC6	REC5	REC4	REC3	REC2	REC1	REC0

The Receive Error Counter is a read/write register that automatically keeps track of the number of receive errors. The value in the counter determines the error state of the device. If this value is from 1 or more, the device enters the error-active state. As soon as this value is 128 or more (in other words, when the ROVL bit is set), the counter is frozen and the device enters the error-passive state. In the error-passive state, the device no longer attempts to transmit any active error frames.

Reset State

Upon power-up or reset, all CAN register bits are cleared to zero except for the Transmit Buffer Empty (TBE) bit of the Receive/Transmit Status (RTSTAT) register, which is set to 1.

11.9 EXTENDED FRAMES

A COP888 device with a built-in CAN interface provides only limited support for extended frames received on the bus.

If the device's Receive All Frames (RIAF) bit is set to 1, extended frame messages are acknowledged. However, the data is discarded and the device does not reply to a remote transmission request received in extended frame format.

If the device's RIAF bit is set to 0, the upper seven received ID bits of an extended frame that match the device's receive identifier (RID) acceptance filter bits are stored in the device's RID register. However, this does not change the register contents and the device does not reply to a remote transmission request and any data is discarded. The device only acknowledges the message if it is correct.

The Receive Error Counter is still incremented when an error occurs with an extended frame, and the counter is still decremented when an valid extended frame is acknowledged.

Chapter 12

INSTRUCTION SET

12.1 INTRODUCTION

This chapter defines the instruction set of the COP888 Family members. It contains information about the instruction set features, addressing modes and types. In addition, it contains a detailed description of each COP888 instruction.

12.2 INSTRUCTION FEATURES

The strength of the instruction set is based on the following features:

- Mostly single-byte opcode instructions minimize program size.
- One instruction cycle for the majority of single-byte instructions to minimize program execution time.
- Many single-byte, multiple function instructions such as DRSZ.
- Three memory mapped pointers: two for register indirect addressing, and one for the software stack.
- Sixteen memory mapped registers that allow an optimized implementation of certain instructions.
- Ability to set, reset, and test any individual bit in data memory address space, including the memory-mapped I/O ports and registers.
- Register-Indirect LOAD and EXCHANGE instructions with optional automatic post-incrementing or decrementing of the register pointer. This allows for greater efficiency (both in cycle time and program code) in loading, walking across and processing fields in data memory.
- Unique instructions to optimize program size and throughput efficiency. Some of these instructions are: DRSZ, IFBNE, DCOR, RETSK, VIS and RRC.

12.3 ADDRESSING MODES

The COP888 instruction set offers a variety of methods for specifying memory addresses. Each method is called an addressing mode. These modes are classified into two categories: operand addressing modes and transfer-of-control addressing modes. Operand addressing modes are the various methods of specifying an address for accessing (reading or writing) data. Transfer-of-control addressing modes are used in conjunction with jump instructions to control the execution sequence of the software program.

12.3.1 Operand Addressing Modes

The operand of an instruction specifies what memory location is to be affected by that instruction. Several different operand addressing modes are available, allowing memory locations to be specified in a variety of ways. An instruction can specify an address directly by supplying the specific address, or indirectly by specifying a register pointer. The contents of the register (or in some cases, two registers) point to the desired memory location. In the immediate mode, the data byte to be used is contained in the instruction itself.

Each addressing mode has its own advantages and disadvantages with respect to flexibility, execution speed, and program compactness. Not all modes are available with all instructions. The Load (LD) instruction offers the largest number of addressing modes.

The available addressing modes are:

- Direct
- Register B or X Indirect
- Register B or X Indirect with Post-Incrementing/Decrementing
- Immediate
- Immediate Short
- Indirect from Program Memory

The addressing modes are described below. Each description includes an example of an assembly language instruction using the described addressing mode.

Direct. The memory address is specified directly as a byte in the instruction. In assembly language, the direct address is written as a numerical value (or a label that has been defined elsewhere in the program as a numerical value).

Example: Load Accumulator Memory Direct

LD A,05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	A6 Hex
Memory Location 0005 Hex	A6 Hex	A6 Hex

Register B or X Indirect. The memory address is specified by the contents of the B Register or X register (pointer register). In assembly language, the notation [B] or [X] specifies which register serves as the pointer.

Example: Exchange Memory with Accumulator, B Indirect

X A,[B]

Reg/Data Memory	Contents Before	Contents After
Accumulator	01 Hex	87 Hex
Memory Location 0005 Hex	87 Hex	01 Hex
B Pointer	05 Hex	05 Hex

Register B or X Indirect with Post-Incrementing/Decrementing. The relevant memory address is specified by the contents of the B Register or X register (pointer register). The pointer register is automatically incremented or decremented after execution, allowing easy manipulation of memory blocks with software loops. In assembly language, the notation [B+], [B-], [X+], or [X-] specifies which register serves as the pointer, and whether the pointer is to be incremented or decremented.

Example: Exchange Memory with Accumulator, B Indirect with Post-Increment

X A,[B+]

Reg/Data Memory	Contents Before	Contents After
Accumulator	03 Hex	62 Hex
Memory Location 0005 Hex	62 Hex	03 Hex
B Pointer	05 Hex	06 Hex

Immediate. The data for the operation follows the instruction opcode in program memory. In assembly language, the number sign character (#) indicates an immediate operand.

Example: Load Accumulator Immediate

LD A,#05

Reg/Data Memory	Contents Before	Contents After
Accumulator	XX Hex	05 Hex

Immediate Short. This is a special case of an immediate instruction. In the "Load B immediate" instruction, the 4-bit immediate value in the instruction is loaded into the lower nibble of the B register. The upper nibble of the B register is reset to 0000 binary.

Example: Load B Register Immediate Short

LD B,#7

Reg/Data Memory	Contents Before	Contents After
B Pointer	12 Hex	07 Hex

Indirect from Program Memory. This is a special case of an indirect instruction that allows access to data tables stored in program memory. In the "Load Accumulator Indirect" (LAID) instruction, the upper and lower bytes of the Program Counter (PCU and PCL) are used temporarily as a pointer to program memory. For purposes of accessing program memory, the contents of the Accumulator and PCL are exchanged. The data pointed to by the Program Counter is loaded into the Accumulator, and simultaneously, the original contents of PCL are restored so that the program can resume normal execution.

Example: Load Accumulator Indirect

LAID

Reg/Data Memory	Contents Before	Contents After
PCU	04 Hex	04 Hex
PCL	35 Hex	36 Hex
Accumulator	1F Hex	25 Hex
Memory Location 041F Hex	25 Hex	25 Hex

12.3.2 Transfer-of-Control Addressing Modes

Program instructions are usually executed in sequential order. However, Jump instructions can be used to change the normal execution sequence. Several transfer-of-control addressing modes are available to specify jump addresses.

A change in program flow requires a non-incremental change in the Program Counter contents. The Program Counter consists of two bytes, designated the upper byte (PCU) and lower byte (PCL). The most significant bit of PCU is not used, leaving 15 bits to address the program memory.

Different addressing modes are used to specify the new address for the Program Counter. The choice of addressing mode depends primarily on the distance of the jump. Farther jumps sometimes require more instruction bytes in order to completely specify the new Program Counter contents.

The available transfer-of-control addressing modes are:

- Jump Relative
- Jump Absolute
- Jump Absolute Long
- Jump Indirect

The transfer-of-control addressing modes are described below. Each description includes an example of a Jump instruction using a particular addressing mode, and the effect on the Program Counter bytes of executing that instruction.

Jump Relative. In this 1-byte instruction, six bits of the instruction opcode specify the distance of the jump from the current program memory location. The distance of the jump can range from –31 to +32. A JP+1 instruction is not allow. The programmer should use a NOP instead.

Example: Jump Relative

JP 0A

Reg	Contents Before	Contents After
PCU	02 Hex	02 Hex
PCL	05 Hex	0F Hex

Jump Absolute. In this 2-byte instruction, 12 bits of the instruction opcode specify the new contents of the Program Counter. The upper three bits of the Program Counter remain unchanged, restricting the new Program Counter address to the same 4-Kbyte

address space as the current instruction. (This restriction is relevant only in devices using more than one 4-Kbyte program memory space.)

Example Jump Absolute

JMP 0125

Reg	Contents Before	Contents After
PCU	0C Hex	01 Hex
PCL	77 Hex	25 Hex

Jump Absolute Long. In this 3-byte instruction, 15 bits of the instruction opcode specify the new contents of the Program Counter.

Example: Jump Absolute Long

JMP 03625

Reg/Memory	Contents Before	Contents After
PCU	42 Hex	36 Hex
PCL	36 Hex	25 Hex

Jump Indirect. In this 1-byte instruction, the lower byte of the jump address is obtained from a table stored in program memory, with the Accumulator serving as the low order byte of a pointer into program memory. For purposes of accessing program memory, the contents of the Accumulator are written to PCL (temporarily). The data pointed to by the Program Counter (PCH/PCL) is loaded into PCL, while PCH remains unchanged.

Example: Jump Indirect

JID

Reg/Memory	Contents Before	Contents After
PCU	01 Hex	01 Hex
PCL	C4 Hex	32 Hex
Accumulator	26 Hex	26 Hex
Memory Location 0126 Hex	32 Hex	32 Hex

NOTE: The VIS instruction is a special case of the Indirect Transfer of Control addressing mode, where the double-byte vector associated with the interrupt

is transferred from adjacent addresses in program memory into the Program Counter in order to jump to the associated interrupt service routine.

12.4 INSTRUCTION TYPES

The instruction set contains a wide variety of instructions. The available instructions are listed below, organized into related groups.

Some instructions test a condition and skip the next instruction if the condition is not true. Skipped instructions are executed as no-operation (NOP) instructions.

Arithmetic Instructions

The arithmetic instructions perform binary arithmetic such as addition and subtraction, with or without the Carry bit.

Add (ADD)

Add with Carry (ADC)

Subtract (SUB)

Subtract with Carry (SUBC)

Increment (INC)

Decrement (DEC)

Decimal Correct (DCOR)

Clear Accumulator (CLR)

Set Carry (SC)

Reset Carry (RC)

Transfer-of Control Instructions

The transfer-of-control instructions change the usual sequential program flow by altering the contents of the Program Counter. The Jump to Subroutine instructions save the Program Counter contents on the stack before jumping; the Return instructions pop the top of the stack back into the Program Counter.

Jump Relative (JP)

Jump Absolute (JMP)

Jump Absolute Long (JMPL)

Jump Indirect (JID)

Jump to Subroutine (JSR)

Jump to Subroutine Long (JSRL)

Return from Subroutine (RET)

Return from Subroutine and Skip (RETSK)

Return from Interrupt (RETI)

Software Trap Interrupt (INTR)

Vector Interrupt Select (VIS)

Load and Exchange Instructions

The load and exchange instructions write byte values in registers or memory. The addressing mode determines the source of the data.

Load (LD)

Load Accumulator Indirect (LAID)

Exchange (X)

Logical Instructions

The logical instructions perform the operations AND, OR, and XOR (Exclusive OR). Other logical operations can be performed by combining these basic operations. For example, complementing is accomplished by exclusive-ORing the Accumulator with FF Hex.

Logical AND (AND)

Logical OR (OR)

Exclusive OR (XOR)

Accumulator Bit Manipulation Instructions

The Accumulator bit manipulation instructions allow the user to shift the Accumulator bits and to swap its two nibbles.

Rotate Right Through Carry (RRC)

Rotate Left Through Carry (RLC)

Swap Nibbles of Accumulator (SWAP)

Stack Control Instructions

Push Data onto Stack (PUSH)

Pop Data off of Stack (POP)

Memory Bit Manipulation Instructions

The memory bit manipulation instructions allow the user to set and reset individual bits in memory.

Set Bit (SBIT)

Reset Bit (RBIT)

Reset Pending Bit (RPND)

Conditional Instructions

The conditional instructions test a condition. If the condition is true, the next instruction is executed in the normal manner; if the condition is false, the next instruction is skipped.

If Equal (IFEQ)

If Not Equal (IFNE)

If Greater Than (IFGT)

If Carry (IFC)

If Not Carry (IFNC)

If Bit (IFBIT)

If B Pointer Not Equal (IFBNE)

And Skip if Zero (ANDSZ)

Decrement Register and Skip if Zero (DRSZ)

No-Operation Instruction

The no-operation instruction does nothing, except to occupy space in the program memory and time in execution.

No-Operation (NOP)

12.5 DIFFERENCES BETWEEN COP800 AND COP888

The COP888 instruction set is almost identical to the COP800 instruction set. There exist only nine differences between the two sets. These differences should be taken into account when translating code from COP888 devices to COP800 devices, but are not important when translating code from COP800 devices to COP888 devices.

The COP888 has seven instructions not available on the COP800 family devices. These instructions are:

- Push Data onto Stack (PUSH)
- Pop Data off of Stack (POP)
- And Skip if Zero (ANDSZ)
- Rotate Left through Carry (RLC)
- If Not Equal (IFNE)
- Vector Interrupt Select (VIS)
- Reset Pending Bit (RPND)

The COP888 instruction set contains a faster version of the LD B,# (B>15) instruction than the COP800. In the COP800 devices, this instruction is 2-bytes/3-cycles and has an opcode of DE/Imm# Hex. In the COP888 devices, this instruction is 2-bytes/2-cycles and has an opcode of 9F/Imm# Hex.

The COP888 supports an additional addressing mode for the IFEQ instruction; the memory direct-immediate operand addressing mode (IFEQ MD,#).

All COP888 instructions are described in detail in Section 12.6 of this manual.

12.6 DETAILED FUNCTIONAL DESCRIPTIONS OF INSTRUCTIONS

The instruction set contains 58 different instructions. Most of the arithmetic, comparison, and data transfer (load, exchange) instructions operate with three different addressing modes (register indirect with **B** pointer, memory direct, and immediate). These various addressing modes increase the instruction total to 87. The detailed instruction descriptions contain the following:

- Opcode mnemonic
- Instruction syntax with operand field descriptor
- Full instruction description
- Register level instruction description
- Number of instruction cycles (each cycle equal to one microsecond at full clock speed)
- Number of bytes (1, 2, or 3) in instruction
- Hexadecimal code for the instruction bytes

The following abbreviations represent the nomenclature used in the detailed instruction description and the COP8 cross-assembler:

A	Accumulator.
B	B Pointer, located in RAM register memory location 00FE.
[B]	Contents of RAM data memory location indicated by B pointer.
[B+]	Same as [B], except that B pointer is post-incremented.
[B-]	Same as [B], except that B pointer is post-decremented.
C	Carry flag, located in bit 6 of the PSW register at memory location 00EF Hex.
HC	Half Carry flag, located in bit 7 of the PSW register at memory location 00EF Hex.
MA	8-bit memory address for RAM data store memory.
MD	Memory Direct, which may be represented by an implicit label (B, X, SP), a defined label (TEMP, COUNTER, etc.), or a direct memory address (12, 0EF, 027, etc., where a leading 0 indicates hexadecimal).
PC	Program Counter (15 bits, with a program memory addressing range of 32768).
PCU	Program Counter Upper, which contains the upper 7 bits of PC.
PCL	Program Counter Lower, which contains the lower 8 bits of PC.
PSW	Processor Status Word Register, found at memory location 00EF.
REG	Selected Register (1 of 16) from the RAM data memory at addresses 00F0-00FF.
REG#	# the of memory register to be used (# = 0-F hexadecimal).
#	# symbol is used to indicate an immediate value, with a leading zero (0) indicating hexadecimal. Examples: #045 = immediate value of hexadecimal 45 #45 = immediate value of decimal 45 # may also be used to indicate bit position, where # = 0-7 Example: RBIT #, [B]
SP	Stack Pointer, located in RAM register memory location 00FD.

X	X pointer, located in RAM register memory location 00FC.
[X]	Contents of RAM data memory location indicated by the X pointer.
[X+]	Same as [X], except that the X pointer is post-incremented.
[X-]	Same as [X], except that the X pointer is post-decremented.

12.6.1 ADC— Add with Carry

Syntax: a) ADC A,[B]

b) ADC A,#

c) ADC A,MD

Description: The contents of

a) the data memory location referenced by the **B** pointer

b) the immediate value found in the second byte of the instruction

c) the data memory location referenced by the second byte of the instruction

are added to the contents of the accumulator, and the result is simultaneously incremented if the Carry flag is found previously set. The result is placed back in the accumulator, and the Carry flag is either set or reset, depending on the presence or absence of a carry from the result. Similarly, the Half Carry flag is either set or reset, depending on the presence or absence of a carry from the low-order nibble.

Operation: A <- A + VALUE + C

C <- CARRY; HC <- HALF CARRY

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
ADC A,[B]	Register Indirect (B Pointer)	1	1	80
ADC A,#	Immediate	2	2	90/Imm #
ADC A,MD	Memory Direct	4	3	BD/MA/80

12.6.2 ADD — Add

Syntax: a) ADD A,[B]

b) ADD A,MD

c) ADD A,#

Description: The contents of the data memory location referenced by

a) the **B** pointer

b) the address in the second byte of the instruction

c) the immediate value found in the second byte of the instruction

are added to the contents of the accumulator, and the result is placed back in the accumulator. The Carry and Half Carry flags are not changed.

Operation: A <- A + VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
ADD A,[B]	Register Indirect (B Pointer)	1	1	84
ADD A,MD	Memory Direct	4	3	BD/MA/84
ADD A,#	Immediate	2	2	94/Imm.#

12.6.3 AND — And

Syntax:

a) AND A,[B]

b) AND A,#

c) AND A,MD

Description: An AND operation is performed on corresponding bits of the accumulator and

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **AND** VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
AND A,[B]	Register Indirect (B Pointer)	1	1	85
AND A,#	Immediate	2	2	95/Imm.#
AND A,MD	Memory Direct	4	3	BD/MA/85

12.6.4 ANDSZ — And, Skip if Zero

Syntax: ANDSZ A,#

Description: An AND operation is performed on corresponding bits of the accumulator and the immediate value found in the second byte of the instruction. If the result is zero, the next instruction is skipped. The accumulator remains unchanged. This instruction may be used in testing for the presence of any selected bits in the accumulator. The mask in the second byte is used to select which bits are tested.

Operation: IF (A AND #) = 0, THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
ANDSZ A,#	Immediate	2	2	60/Imm.#

12.6.5 CLR — Clear Accumulator

Syntax: CLR A

Description: The accumulator is cleared to all zeros.

Operation: A <- 0

Instruction	Addressing Mode	Instruction Cycle	Bytes	Hex Op Code
CLR A	Implicit	1	1	64

12.6.6 DCOR — Decimal Correct

Syntax: DCOR A

Description: This instruction when used following an ADC (add with carry) or SUBC (subtract with carry) instruction will decimal correct the result from the binary addition or subtraction. Note that the ADC instruction must be preceded with an ADD A, #066 (add hexadecimal 66) instruction for the decimal addition correction. This instruction assumes that the two operands are in BCD (Binary Coded Decimal) format and produces the result in the same BCD format. The Carry and Half Carry flags remain unchanged.

Operation: A (BCD FORMAT) <- A (BINARY FORMAT)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DCOR A	Implicit	1	1	66

12.6.7 DEC — Decrement Accumulator

Syntax: DECA

Description: This instruction decrements the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DEC A	Implicit	1	1	8B

12.6.8 DRSZ REG# — Decrement Register and Skip if Result is Zero

Syntax: DRSZ REG#

Description: This instruction decrements the contents of the selected memory register (selected by #, where # = 0 to F) and places the result back in the same register. If the result is zero, the next instruction is skipped. This instruction is useful where it is desired to repeat an instruction sequence a given number of times. The desired number of times that the instruction sequence is to be executed is placed in a register, and a DRSZ instruction with that register is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The JP branch-back instruction is executed each time around the instruction sequence loop until the register count is decremented down to zero, at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation: REG <- REG - 1

IF (REG - 1) = 0,

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
DRSZ REG#	Register Direct (Implicit)	3	1	C (REG#)

12.6.9 IFBIT — Test Bit

Syntax:

a) IFBIT #,[B]

b) IFBIT #,MD

c) IFBIT #,A

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) from

a) the memory location reference by the B pointer is tested.

b) the memory location referenced by the address in the second byte of the instruction is tested.

c) the accumulator is tested.

If the selected bit is high (=1), then the next instruction is executed. Otherwise, the next instruction is skipped.

Operation:

IF BIT (#) SELECTED

IS EQUAL TO 0,

THEN SKIP NEXT INSTRUCTION

Instruction	Address Mode	Instruction Cycle	Bytes	Hex Op Code
IFBIT #,[B]	Register Indirect (B Pointer)	1	1	7#
IFBIT #,MD	Memory Direct	4	3	BD/MA/7#
IFBIT #,A	Immediate	2	2	60/2#

NOTE: The IFBIT #,A is a special subset of the more generalized ANDSZ instruction and shares the same opcode of 60. This instruction disassembles into the ANDSZ instruction.

IFBIT 0,A equivalent to ANDSZ A,#1

IFBIT 1,A equivalent to ANDSZ A,#2

IFBIT 2,A equivalent to ANDSZ A,#4

IFBIT 3,A equivalent to ANDSZ A,#8

IFBIT 4,A equivalent to ANDSZ A,#16

IFBIT 5,A equivalent to ANDSZ A,#32

IFBIT 6,A equivalent to ANDSZ A,#64

IFBIT 7,A equivalent to ANDSZ A,#128

12.6.10 IFBNE # — If B Pointer Not Equal

Syntax: IFBNE #

Description: If the low-order nibble of the **B** pointer is not equal to # (where # = 0 to F), then the next instruction is executed. Otherwise, the next instruction is skipped. This instruction is useful where the **B** pointer is walked across a data field as part of a closed loop instruction sequence. The IFBNE instruction is coded at the end of the sequence followed by a JP (Jump Relative) instruction that branches back to the start of the instruction sequence. The # coded with the IFBNE represents the next address beyond the data field. The **B** pointer instruction with post-increment or decrement of the pointer may be used in walking across the data field in either direction. The instruction sequence branches back and repeats until the low-order nibble of the **B** pointer is found equal to the # (representing the next address beyond the data field), at which time the JP instruction is skipped as the program branches (skips) out of the loop.

Operation: IF **B** POINTER LOW-ORDER NIBBLE EQUALS #,

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFBNE #	Implicit	1	1	4#

12.6.11 IFC — Test if Carry

Syntax: IFC

Description: The next Instruction is executed if the Carry flag is found set. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF NO CARRY (C = 0),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFC	Implicit	1	1	88

12.6.12 IFEQ — Test if Equal

Syntax:

a) IFEQ A,[B]

b) IFEQ A,#

c) IFEQ A,MD

d) IFEQ MD,#

a) The contents of the data memory location referenced by the **B** pointer are compared for equality with the contents of the accumulator.

b) The immediate value found in the second byte of the instruction is compared for equality with the contents of the accumulator.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are compared for equality with the contents of the accumulator.

d) The contents of the memory location referenced by the address in the second byte of the instruction are compared for equality with the immediate value found in the third byte of the instruction.

A successful equality comparison results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation:

IF CONTENTS OF SPECIFIED LOCATION ≠ VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFEQ A,[B]	Register Indirect (B Pointer)	1	1	82
IFEQ A,#	Immediate	2	2	92/Imm.#
IFEQ A,MD	Memory Direct	4	3	BD/MA/82
IFEQ MD,#	Memory Direct, Immediate	3	3	A9/MA/Imm.#

12.6.13 IFGT — Test if Greater Than

Syntax:

a) IFGT A,[B]

b) IFGT A,#

c) IFGT A,MD

Description: The contents of the accumulator are tested for being greater than

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

A successful greater than test results in the execution of the next instruction. Otherwise, the next instruction is skipped.

Operation: IF A ≤ VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFGT A,[B]	Register Indirect (B Pointer)	1	1	83
IFGT A,#	Immediate	2	2	93/Imm.#
IFGT A,MD	Memory Direct	4	3	BD/MA/83

12.6.14 IFNC — Test If No Carry

Syntax: IFNC

Description: The next instruction is executed if the Carry flag is found reset. Otherwise, the next instruction is skipped. The Carry flag is left unchanged.

Operation: IF CARRY (C=1),

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFNC	Implicit	1	1	89

12.6.15 IFNE — Test If Not Equal

Syntax:

a) IFNE A,[B]

b) IFNE A,#

c) IFNE A,MD

Description:

a) The contents of the data memory location referenced by the **B** pointer are compared for inequality with the contents of the accumulator.

b) The immediate value found in the second byte of the instruction is compared for inequality with the contents of the accumulator.

c) The contents of the data memory location referenced by the address in the second bye of the instruction are compared for inequality with the contents of the accumulator.

A successful inequality comparison results in the execution of the next instruction; otherwise, the next instruction is skipped.

Operation:

IF A = VALUE

THEN SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
IFNE A,[B]	Register Indirect (B Pointer)	1	1	B9
IFNE A,#	Immediate	2	2	99/Imm.#
IFNE A,MD	Memory Direct	4	3	BD/MA/B9

12.6.16 INC — Increment Accumulator

Syntax: INC A

Description: This instruction increments the contents of the accumulator and places the result back in the accumulator. The Carry and Half Carry flags remain unchanged.

Operation: A <- A + 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INC A	Implicit	1	1	8A

12.6.17 INTR — Interrupt (Software Trap)

Syntax: INTR

Description: This zero opcode software trap instruction first stores its return address in the data memory software stack and then branches to program memory location 00FF. This memory location is the common switching point for all COP888 interrupts, both hardware and software. The program starting at memory location 00FF sorts out the priority of the various interrupts and then vectors to the correct interrupt service routine.

In order to save the return address, the contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented. The contents of PCU (Upper 7 bits of PC) are transferred to the new data memory location referenced by SP. Then SP is again decremented to set up the software stack for the next interrupt or subroutine.

The INTR instruction is not meant to be programmed explicitly, but rather to be automatically invoked when certain error conditions occur. The reading of undefined (non-existent) program memory produces all zeros, which in turn invokes the INTR instruction. A similar software trap can be set up if the subroutine Stack Pointer (SP) is initialized to the data memory location at the top of user RAM space. Then if the software stack is ever overpopped (more subroutine or interrupt returns than calls), all ones will be returned from the undefined (non-existent) RAM. This will cause the program to return to the program address FFFF Hex, which in turn will read all zeros and once again invoke the software trap INTR instruction.

Operation:
[SP] <- PCL
[SP - 1] <- PCU
[SP - 2] : SET UP FOR NEXT STACK REFERENCE
PC <- 0FF

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
INTR	Implicit	7	1	00

12.6.18 JID — Jump Indirect

Syntax: JID

Description: The JID instruction uses the contents of the accumulator to point to an indirect vector table of program addresses. The contents of the accumulator are transferred to PCL (Lower 8 bits of PC), after which the data accessed from the program memory location addressed by PC is transferred to PCL. The program then jumps to the program memory location accessed by PC. It should be observed that PCU (Upper 7 bits of PC) is never changed during the JID instruction, so that the Jump Indirect must jump to a location in the current program memory page of 256 addresses. However, if the JID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary, and both accesses to program memory (vector table and the new instruction) will be fetched from the next page of 256 bytes.

Operation: PCL <- A

PCL <- Program Memory (PCU,A)

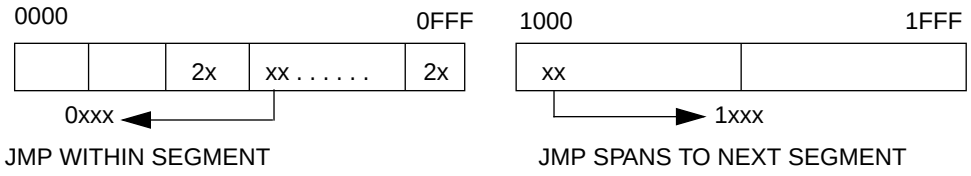
Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JID	Indirect	3	1	A5

12.6.19 JMP — Jump Absolute

Syntax: JMP ADDR

Description: This instruction jumps to the programmed memory address. The value found in the lower nibble (4 bits) of the first byte of the instruction is transferred to the lower nibble of PCU (Upper 7 bits of PC), and then the value found in the second byte of the instruction is transferred to PCL (Lower 8 bits of PC). The program then jumps to the program memory location accessed by PC.

The address range is 15 bits. The JMP instruction address contents is only 12 bits. Resolution of the high 3 bits of the address is to the 4K memory segment containing the **low byte of the instruction**. The following diagram illustrates:



If instruction byte 2 is in 4K segment 0, the jump address is in segment 0.

If instruction byte 2 is in 4K segment 1, the jump address is in segment 1.

Operation: PC11-8 <- HIADDR (HIGH NIBBLE OF SECOND BYTE OF INSTRUCTION, LOW NIBBLE OF FIRST BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMP ADDR	Absolute	3	2	2HIADDR/LOADDR

12.6.20 JMPL — Jump Absolute Long

Syntax: JMPL ADDR

Description: The JMPL instruction allows branching to anywhere in the 32-Kbyte program memory space. The values found in the second and third bytes of the instruction are transferred to PCU (Upper 7 bits of PC) and PCL (Lower 8 bits of PC) respectively. The program then jumps to the program memory location accessed by PC.

Operation: PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JMPL ADDR	Absolute	4	3	AC/HIADDR/LOADDR

12.6.21 JP — Jump Relative

Syntax: JP DISP

Description: The relative displacement value found in the instruction opcode (all 8 bits) is added to the Program Counter (PC). The normal PC incrementation is also performed. The displacement value allows a branch back from 0 to 31 places (with the 0 representing an infinite closed loop branch to itself) and a branch forward from 2 to 32 places. A branch forward of 1 is not allowed, since this zero opcode conflicts with the INTR software trap instruction.

Operation: PC <- PC + DISP + 1 (DISP ≠ 0)

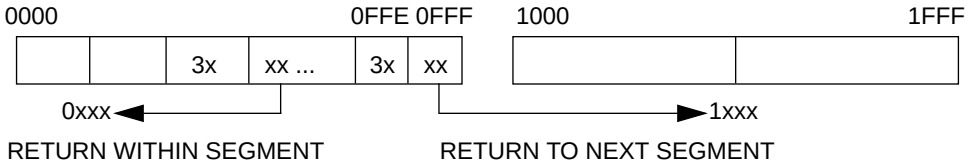
Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JP DISP	Relative	3	1	0, 1, E, F + DISP #

12.6.22 JSR — Jump Subroutine

Syntax: JSR ADDR

Description: This instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address. The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address has now been saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

The address range is 15 bits. The JSR instruction address contents is only 12 bits. Resolution of the high 3 bits of the address is to the 4K memory segment containing the **high byte of the return address**. A JSR contained in the last 2 bytes of a 4K segment will jump to the next segment. The following diagram illustrates:



If the return address is in 4K segment 0, the jump address is in segment 0.

If the return address is in 4K segment 1, the jump address is in segment 1.

Operation: [SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC11-8 <- HIADDR (HIGH NIBBLE OF RETURN ADDRESS, LOW NIBBLE OF FIRST BYTE INSTRUCTION)

PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSR ADDR	Absolute	5	2	3HIADDR/LOADDR

12.6.23 JSRL — Jump Subroutine Long

Syntax: JSRL ADDR

Description: The JSRL instruction allows the subroutine to be located anywhere in the 32-Kbyte program memory space. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address.

The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.

Next, the values found in the second and third bytes of the instruction are transferred to PCU and PCL respectively. The program then jumps to the program memory location accessed by PC.

Operation: [SP] <- PCL

[SP - 1] <- PCU

[SP - 2]: SET UP FOR NEXT STACK REFERENCE

PC14-8 <- HIADDR (SECOND BYTE OF INSTRUCTION)

PC7-0 <- LOADDR (THIRD BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSRL ADDR	Absolute	5	3	AD/HIADDR/LOADDR

12.6.24 LAID — Load Accumulator Indirect

Address Mode: INDIRECT

Description: The LAID instruction uses the contents of the accumulator to point to a fixed data table stored in program memory. The data table usually represents a translation matrix, such as the input from a keyboard or the output to a display.

The contents of the accumulator are exchanged with the contents of PCL (Lower 8 bits of PC). The data accessed from the program memory location addressed by PC is then transferred to the accumulator. Simultaneously, the original contents of PCL are transferred back to PCL from the accumulator. It should be observed that PCU (Upper 7 bits of PC) is not changed during the LAID instruction, so that the load accumulator indirect along with the associated fixed data table must both be located in the current memory page of 256 bytes. However, if the LAID instruction is located at the last address of the page, the PC counter will have already incremented over the page boundary resulting in the operand being fetched from the next page. Consequently, in this instance, the fixed data table must reside in the next page of 256 bytes in the program memory.

Operation: A <- Program Memory (PCU, A)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LAID	Indirect	3	1	A4

12.6.25 LD — Load Accumulator

Syntax:

a) LD A,[B]

b) LD A,[B+]

c) LD A,[B-]

d) LD A,#

e) LD A,MD

f) LD A,[X]

g) LD A,[X+]

h) LD A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are loaded into the accumulator, and then the **B** pointer is post-decremented.

d) The immediate value found in the second byte of the instruction is loaded into the accumulator.

e) The contents of the data memory location referenced by the address in the second byte of the instruction are loaded into the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator.

g) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-incremented.

h) The contents of the data memory location referenced by the **X** pointer are loaded into the accumulator, and then the **X** pointer is post-decremented.

Operation: A <- VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD A,[B]	Register Indirect (B Pointer)	1	1	AE
LD A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	AA
LD A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	AB
LD A,#	Immediate	2	2	98/Imm.#
LD A,MD	Memory Direct	3	2	9D/MA
LD A,[X]	Register Indirect (X Pointer)	3	1	BE
LD A,[X+]	Register Indirect With Post-Incrementing X Pointer	3	1	BA
LD A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	BB

12.6.26 LD — Load B Pointer

Syntax: a) LD B,# (# < 16)

b) LD B,# (# > 15)

Description: a) The one's complement of the value found in the lower nibble (4 bits) of the instruction is transferred to the lower-nibble position of the **B** pointer register, with the upper-nibble position being cleared to all zeros.

b) The immediate value found in the second byte of the instruction is transferred to the **B** pointer register.

Operation: a) B3-B0 <- # and B7-B4 <- 0

b) B <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD B,#	Short Immediate	1	1	5(15-#)
LD B,#	Immediate	2	2	9F/Imm.#

12.6.27 LD — Load Memory

Syntax:

a) LD [B],#

b) LD [B+],#

c) LD [B-],#

d) LD MD,#

Description:

a) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer.

b) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-incremented.

c) The immediate value found in the second byte of the instruction is loaded into the data memory location referenced by the **B** pointer, and then the **B** pointer is post-decremented.

d) The immediate value found in the third byte of the instruction is loaded into the data memory location referenced by the address in the second byte of the instruction.

Operation:

a) [B] <- #

b) [B] <- #; B <- B + 1

c) [B] <- #; B <- B - 1

d) MD <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD [B],#	Register Indirect/Immediate	2	2	9E/Imm.#
LD [B+],#	Register Indirect With Post-Incrementing/Immediate	2	2	9A/Imm.#
LD [B-],#	Register Indirect With Post-Decrementing/Immediate	2	2	9B/Imm.#
LD MD,#	Memory Direct/Immediate	3	3	BC/MA/Imm.#

12.6.28 LD — Load Register

Syntax: LD REG,#

Description: The immediate value found in the second byte of the instruction is loaded into the data memory register referenced by the low-order nibble of the first byte of the instruction.

Operation: REG <- #

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
LD REG,#	Implicit/Immediate	3	2	D(REG#)/Imm.#

12.6.29 NOP — No Operation

Syntax: NOP

Description: No operation is performed by this instruction, so the net result is a delay of one instruction cycle time.

Operation: NO OPERATION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
NOP	Implicit	1	1	B8

12.6.30 OR — Or

Syntax:

a) OR A,[B]

b) OR A,#

c) OR A,MD

Description: An OR operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **OR** VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
OR A,[B]	Register Indirect (B Pointer)	1	1	87
OR A,#	Immediate	2	2	97/Imm.#
OR A,MD	Memory Direct	4	3	BD/MA/87

12.6.31 POP — Pop Stack

Syntax: POP A

Description: The Stack Pointer (SP) is incremented, and then the contents of the data memory location referenced by the SP are transferred to the accumulator.

Operation: SP <- SP + 1

A <- [SP]

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
POP	Implicit	3	1	8C

12.6.32 PUSH — Push Stack

Syntax: PUSH A

Description: The contents of the accumulator are transferred to the data memory location referenced by the Stack Pointer (SP), and then the SP is decremented.

Operation: [SP] <- A

SP <- SP - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
PUSH	Implicit	3	1	67

12.6.33 RBIT — Reset Memory Bit

Syntax: a) RBIT #,[B]

b) RBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) of the data memory location referenced by the

a) **B** pointer is reset to 0.

b) address in the second byte of the instruction is reset to 0.

Operation: [Address:#] <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RBIT #,[B]	Register Indirect (B Pointer)	1	1	6(8 + #)
RBIT #,MD	Memory Direct	4	3	BD/MA/6(8+#)

12.6.34 RC — Reset Carry

Syntax: RC

Description: Both the Carry and Half Carry flags are reset to 0.

Operation: C <- 0

HC <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RC	Implicit	1	1	A0

12.6.35 RET — Return from Subroutine

Syntax: RET

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), after which SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RET	Implicit	5	1	8E

12.6.36 RETI — Return from Interrupt

Syntax: RETI

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to the program memory location accessed by PC. The Global Interrupt Enable flag (GIE) is set to 1.

Operation:

PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

GIE <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETI	Implicit	5	1	8F

12.6.37 RETSK — Return and Skip

Syntax: RETSK

Description: The Stack Pointer (SP) is first incremented. The contents of the data memory location referenced by SP are then transferred to PCU (Upper 7 bits of PC), and SP is again incremented. Next, the contents of the data memory location referenced by SP are transferred to PCL (Lower 8 bits of PC). The return address has now been retrieved from the software stack in data memory RAM. The program now jumps to and then skips the instruction in the program memory location accessed by PC.

Operation: PCU <- [SP + 1]

PCL <- [SP + 2]

[SP + 2] : SET UP FOR NEXT STACK REFERENCE

SKIP NEXT INSTRUCTION

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RETSK	Implicit	5	1	8D

12.6.38 RLC — Rotate Accumulator Left Through Carry

Address Mode: RLC A

Description: The contents of the accumulator and Carry flag are rotated left one bit position, with the Carry flag serving as a ninth bit position linking the ends of the 8-bit accumulator. The previous carry is transferred to the low-order bit position of the accumulator. The high-order accumulator bit (A7) is transferred to the Carry flag. The A3 (high-order bit of the low-order nibble) of the accumulator is transferred into the Half Carry flag (HC) as well as into the A4 bit position.

Operation: C <- A7 <- A6 <- A5 <- A4 <- A3 <- A2 <- A1 <- A0 <- C

HC <- A3

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RLC A	Implicit	1	1	A8

12.6.39 RPND — Reset Pending

Syntax: RPND

Description: The RPND instruction resets the Non-Maskable Interrupt Pending flag (NMIPND) provided that the NMI interrupt has already been acknowledged and the Software Trap Pending flag was not found set. Also, RPND unconditionally resets the Software Trap Pending flag (STPND).

Operation: IF NMI interrupt acknowledged and STPND = 0

THEN NMPND <- 0 and STPND <- 0

ELSE STPND <- 0

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RPND	Implicit	1	1	B5

12.6.40 RRC — Rotate Accumulator Right Through Carry

Address Mode: RRC A

Description: The contents of the accumulator and Carry flag are rotated right one bit position, with the Carry flag serving as a ninth bit position linking the ends of the 8-bit accumulator. The previous carry is transferred to the high-order bit position of the accumulator. The low-order accumulator bit (A0) is transferred to both the Carry flag and the Half Carry flag.

Operation: C -> A7 -> A6 -> A5 -> A4 -> A3 -> A2 -> A1 -> A0 -> C

A0 -> HC

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
RRC A	Implicit	1	1	B0

12.6.41 SBIT — Set Memory Bit

Syntax: a) SBIT #,[B]

b) SBIT #,MD

Description: The selected bit (# = 0 to 7, with 7 being the high-order bit) of the data memory location referenced by the

a) **B** pointer is set to 1.

b) address in the second byte of the instruction is set to 1.

Operation: [Address:#] <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SBIT #,[B]	Register Indirect (B Pointer)	1	1	7(8 + #)
SBIT #,MD	Memory Direct	4	3	BD/MA/7(8+#)

12.6.42 SC — Set Carry

Syntax: SC

Description: Both the Carry and Half Carry flags are set to 1.

Operation: C <- 1

HC <- 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SC	Implicit	1	1	A1

12.6.43 SUBC — Subtract with Carry

Syntax:

a) SUBC A,[B]

b) SUBC A,#

c) SUBC A,MD

Description:

a) The contents of the data memory location referenced by the **B** pointer are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

b) The immediate value found in the second byte of the instruction is subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

c) The contents of the data memory location referenced by the address in the second byte of the instruction are subtracted from the contents of the accumulator, and the result is simultaneously decremented if the Carry flag is found previously reset.

The result is placed back in the accumulator, and the Carry flag is either reset or set, depending on the presence or absence of a borrow from the result. Similarly, the Half Carry flag is either reset or set, depending on the presence or absence of a borrow from the low-order nibble.

This instruction is implemented by adding the one's complement of the subtrahend to the accumulator and then incrementing the result. Consequently, the borrow is the equivalent of the absence of carry and vice versa. Similarly, the half carry is the equivalent of the absence of half borrow and vice versa. A previous borrow (absence of previous carry) will inhibit the incrementation of the result.

Operation:

$A \leftarrow A + \overline{VALUE} + C$

C <- ABSENCE OF BYTE BORROW

HC <- ABSENCE OF LOW NIBBLE HALF BORROW

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SUBC A,[B]	Register Indirect (B Pointer)	1	1	81
SUBC A,#	Immediate	2	2	91/Imm.#
SUBC A,MD	Memory Direct	4	3	BD/MA/81

12.6.44 SWAP — Swap Nibbles of Accumulator

Syntax: SWAP A

Description: The upper and lower nibbles of the accumulator are exchanged.

Operation: A(7-4) <--> A(3 - 0)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
SWAP A	Implicit	1	1	65

12.6.45 VIS — Vector Interrupt Select

Syntax: VIS

Description: The purpose of the VIS instruction is to vector to the interrupt service routine for the interrupt with the highest priority and arbitration ranking that is currently enabled and requesting. The VIS instruction expedites this procedure of vectoring to the interrupt service routine.

All interrupts branch to program memory location 00FF Hex once an interrupt is acknowledged. Thus, any desired context switching (such as storing away the contents of the accumulator or B or X pointer) is normally programmed starting at location 00FF Hex, followed by the VIS instruction. The VIS instruction can be programmed at memory location 00FF Hex if no context switching is desired.

The VIS instruction first jumps to a double-byte vector in a 32-byte interrupt vector program memory table that is located at the top of a program memory block from address xyE0 to xyFF Hex. Note that xy is the block number (usually 01) where the VIS instruction is located (each block of program memory contains 256 bytes). This double-byte vector is transferred to PC (high-order byte first), and then the program jumps to the associated interrupt service routine indicated by the vector. These interrupt service routines can be anywhere in the 32-Kbyte program memory space.

Should the VIS instruction be programmed at the top location of a memory block (such as address 00FF Hex), the associated 32-byte vector table is resident at the top of the next higher block (locations 01E0 to 01FF Hex with the VIS instruction at 00FF Hex).

Operation: PCL <- VA (Interrupt Arbitration Vector generated by hardware)

PCU <- Program Memory (PCU,VA)

PCL <- Program Memory (PCU,VA+1)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
VIS	Implicit	5	1	B4

12.6.46 X — Exchange Memory with Accumulator

Syntax:

a) X A,[B]

b) X A,[B+]

c) X A,[B-]

d) X A,MD

e) X A,[X]

f) X A,[X+]

g) X A,[X-]

Description:

a) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator.

b) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-incremented.

c) The contents of the data memory location referenced by the **B** pointer are exchanged with the contents of the accumulator, and then the **B** pointer is post-decremented.

d) The contents of the data memory location referenced by the address in the second byte of the instruction are exchanged with the contents of the accumulator.

e) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator.

f) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-incremented.

g) The contents of the data memory location referenced by the **X** pointer are exchanged with the contents of the accumulator, and then the **X** pointer is post-decremented.

Operation:

a) A <-> [B]

b) A <-> B; B <- B + 1

c) A <-> B; B <- B - 1

d) A <-> MD

e) A <-> X

f) A <-> X; X <- X + 1

g) A <-> X; X <- X - 1

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
X A,[B]	Register Indirect (B Pointer)	1	1	A6
X A,[B+]	Register Indirect With Post-Incrementing B Pointer	2	1	A2
X A,[B-]	Register Indirect With Post-Decrementing B Pointer	2	1	A3
X A,MD	Memory Direct	3	2	9C/MA
X A,[X]	Register Indirect (X Pointer)	3	1	B6
X A,[X+]	Register Indirect With Post-Incrementing X Pointer	3	1	B2
X A,[X-]	Register Indirect With Post-Decrementing X Pointer	3	1	B3

12.6.47 XOR — Exclusive Or

Syntax:

a) XOR A,[B]

b) XOR A,#

c) XOR A,MD

Description: An XOR (Exclusive OR) operation is performed on corresponding bits of the accumulator with

a) the contents of the data memory location referenced by the **B** pointer.

b) the immediate value found in the second byte of the instruction.

c) the contents of the data memory location referenced by the address in the second byte of the instruction.

The result is placed back in the accumulator.

Operation: A <- A **XOR** VALUE

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
XOR A,[B]	Register Indirect (B Pointer)	1	1	86
XOR A,#	Immediate	2	2	96/Imm.#
XOR A,MD	Memory Direct	4	3	BD/MA/86

12.7 INSTRUCTION OPERATIONS SUMMARY

INSTR		FUNCTION	REGISTER OPERATION
ADD	A, MemI	Add	A <- A + MemI
ADC	A, MemI	Add with carry	A <- A + MemI + C, C <- Carry
SUBC	A, MemI	Subtract with carry	A <- A - MemI + C, C <- Carry
AND	A, MemI	Logical AND	A <- A and MemI
ANDSZ	A, Imm	Logical AND Imm, Skip if Zero	Skip next if (A and Imm) = 0
OR	A, MemI	Logical OR	A <- A or MemI
XOR	A, MemI	Logical Exclusive-OR	A <- A xor MemI
IFEQ	A, MemI	IF equal	Compare A and MemI, Do next if A = MemI
IFEQ	MD, Imm	IF equal	Compare MD and Imm, Do next if MD = Imm
IFNE	A, MemI	IF not equal	Compare A and MemI, Do next if A $\neq$ MemI
IFGT	A, MemI	IF greater than	Compare A and MemI, Do next if A > MemI
IFBNE	#	IF B not equal	Do next if lower 4 bits of B not = Imm
DRSZ	Reg	Decrement Reg, skip if zero	Reg <- Reg - 1, skip if Reg goes to zero
SBIT	#, Mem	Set bit	1 to Mem.bit (bit = 0 to 7 immediate)
RBIT	#, Mem	Reset bit	0 to Mem.bit (bit = 0 to 7 immediate)
IFBIT	#, Mem	If bit	If Mem.bit is true, do next instruction
RPND		Reset Pending Flag	Reset Software Interrupt Pending Flag
X	A, Mem	Exchange A with memory	A <-> Mem
LD	B, Imm	Load B with Immed.	B <- Imm
LD	A, MemI	Load A with memory	A <- MemI
LD	Mem, Imm	Load Direct memory Immed.	Mem <- Imm
LD	Reg, Imm	Load Register memory immed.	Reg <- Imm
X	A, [B$\pm$]	Exchange A with memory [B]	A <-> [B] (B <- B $\pm$ 1)
X	A, [X$\pm$]	Exchange A with memory [X]	A <-> [X] (X <- X $\pm$1)
LD	A, [B$\pm$]	Load A with memory [B]	A <- [B] (B <- B $\pm$1)
LD	A, [X$\pm$]	Load A with memory [X]	A <- [X] (X <- X $\pm$1)
LD	[B$\pm$], Imm	Load memory immediate	[B] <- Imm (B <- B $\pm$1)
CLRA		Clear A	A <- 0
INC	A	Increment A	A <- A + 1
DEC	A	Decrement A	A <- A - 1
LAID		Load A indirect from ROM	A <- ROM(PU, A)
DCOR	A	Decimal correct A	A <- BCD correction (follows ADC, SUBC)
RRC	A	Rotate right through carry	C -> A7 -> .... -> A0 -> C, HC <- A0
RLC	A	Rotate left through carry	C <- A7 <- .... <- A0 <- C, HC <- A3
SWAP	A	Swap nibbles of A	A7...A4 <-> A3...A0
SC		Set C	C <- 1
RC		Reset C	C <- 0
IFC		If C	If C is true, do next instruction
IFNC		If Not C	If C is not true, do next instruction
POP	A	Pop the Stack into A	SP <- SP + 1, A <- [SP]
PUSH	A	Push A onto the Stack	[SP] <- A, SP <- SP - 1
VIS		Vector to Interrupt Service	PCL <- [VL], PCU <- [VU]
JMPL	Addr.	Jump absolute long	PC <- ii (ii = 15 bits, 0 to 32K)
JMP	Addr.	Jump absolute	PC11...PC0 <- i (i = 12 bits) PC15...PC12 remain unchanged
JP	Disp.	Jump relative short	PC <- PC + r (r is -31 to +32, not 1)
JSRL	Addr.	Jump subroutine long	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- ii
JSR	Addr.	Jump subroutine	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC11..PC0 <- ii
JID		Jump indirect	PL <- ROM(PU, A)
RET		Return from subroutine	SP + 2, PL <- [SP], PU <- [SP - 1]
RETSK		Return and skip	SP + 2, PL <- [SP], PU <- [SP - 1], Skip next instr.
RETI		Return from interrupt	SP + 2, PL <- [SP], PU <- [SP - 1], GIE <- 1
INTR		Generate an interrupt	[SP] <- PL, [SP - 1] <- PU, SP - 2, PC <- 0FF
NOP		No operation	PC <- PC + 1

12.8 INSTRUCTION BYTES AND CYCLES

Table 12-1 Instructions Using A and C

INSTR	BYTES/ CYCLES
CLRA	1/1
INCA	1/1
DECA	1/1
LAID	1/3
DCOR	1/1
RRCA	1/1
RLCA	1/1
SWAPA	1/1
SC	1/1
RC	1/1
IFC	1/1
IFNC	1/1
PUSHA	1/3
POPA	1/3
ANDSZ	2/2

Table 12-2 Transfer of Control Instructions

INSTR	BYTES/ CYCLES
JMPL	3/4
JMP	2/3
JP	1/3
JSRL	3/5
JSR	2/5
JID	1/3
VIS	1/5
RET	1/5
RETSK	1/5
RETI	1/5
INTR	1/7
NOP	1/1

Table 12-3 Memory Transfer Instructions

INSTR	REGISTER INDIRECT		DIRECT	IMMEDIATE	REGISTER INDIRECT AUTO INCR & DECR	
	[B]	[X]			[B+, B-]	[X+, X-]
X A,[a]	1/1	1/3	2/3		1/2	1/3
LD A,[a]	1/1	1/3	2/3	2/2	1/2	1/3
LD B, Imm				1/1[b]		
LD B, Imm				2/2[c]		
LD Mem, Imm	2/2		3/3		2/2	
LD Reg, Imm			2/3			

a. Memory location addressed by B or X or directly
b. IF B < 16
c. IF B > 15

Table 12-4 Arithmetic and Logic Instructions

INSTR	[B]	DIRECT	IMMEDIATE
ADD	1/1	3/4	2/2
ADC	1/1	3/4	2/2
SUBC	1/1	3/4	2/2
AND	1/1	3/4	2/2
OR	1/1	3/4	2/2
XOR	1/1	3/4	2/2
IFEQ	1/1	3/4	2/2
IFNE	1/1	3/4	2/2
IFGT	1/1	3/4	2/2
IFBNE	1/1		
DRSZ		1/3	
SBIT	1/1	3/4	
RBIT	1/1	3/4	
IFBIT	1/1	3/4	

RPND	1/1

Table 12-5 Opcodes

UPPER NIBBLE																LOWER NIBBLE
F	**E**	**D**	**C**	**B**	**A**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
JP-15	JP-31	LD 0F0,#I	DRSZ 0F0	RRCA	RC	ADC A,#I	ADC A,[B]	IFBIT 0,[B]	ANDSZ A,#I	LD B,#0F	IFBNE 0	JSR x000-x0FF	JMP x000-x0FF	JP+17	INTR	0
JP-14	JP-30	LD 0F1,#I	DRSZ 0F1	*	SC	SUBC A,#I	SUBC A,[B]	IFBIT 1,[B]	*	LD B,#0E	IFBNE 1	JSR x100-x1FF	JMP x100-x1FF	JP+18	JP+2	1
JP-13	JP-29	LD 0F2,#I	DRSZ 0F2	X A,[X+]	X A,[B+]	IFEQ A,#I	IFEQ A,[B]	IFBIT 2,[B]	*	LD B,#0D	IFBNE 2	JSR x200-x2FF	JMP x200-x2FF	JP+19	JP+3	2
JP-12	JP-28	LD 0F3,#I	DRSZ 0F3	X A,[X-]	X A,[B-]	IFGT A,#I	IFGT A,[B]	IFBIT 3,[B]	*	LD B,#0C	IFBNE 3	JSR x300-x3FF	JMP x300-x3FF	JP+20	JP+4	3
JP-11	JP-27	LD 0F4,#I	DRSZ 0F4	VIS	LAID	ADD A,#I	ADD A,[B]	IFBIT 4,[B]	CLRA	LD B,#0B	IFBNE 4	JSR x400-x4FF	JMP x400-x4FF	JP+21	JP+5	4
JP-10	JP-26	LD 0F5,#I	DRSZ 0F5	RPND	JID	AND A,#I	AND A,[B]	IFBIT 5,[B]	SWAPA	LD B,#0A	IFBNE 5	JSR x500-x5FF	JMP x500-x5FF	JP+22	JP+6	5
JP-9	JP-25	LD 0F6,#I	DRSZ 0F6	X A,[X]	X A,[B]	XOR A,#I	XOR A,[B]	IFBIT 6,[B]	DCORA	LD B,#09	IFBNE 6	JSR x600-x6FF	JMP x600-x6FF	JP+23	JP+7	6
JP-8	JP-24	LD 0F7,#I	DRSZ 0F7	*	*	OR A,#I	OR A,[B]	IFBIT 7,[B]	PUSHA	LD B,#08	IFBNE 7	JSR x700-x7FF	JMP x700-x7FF	JP+24	JP+8	7
JP-7	JP-23	LD 0F8,#I	DRSZ 0F8	NOP	RLCA	LD A,#I	IFC	SBIT 0,[B]	RBIT 0,[B]	LD B,#07	IFBNE 8	JSR x800-x8FF	JMP x800-x8FF	JP+25	JP+9	8
JP-6	JP-22	LD 0F9,#I	DRSZ 0F9	IFNE A,[B]	IFEQ Md,#I	IFNE A,#I	IFNC	SBIT 1,[B]	RBIT 1,[B]	LD B,#06	IFBNE 9	JSR x900-x9FF	JMP x900-x9FF	JP+26	JP+10	9
JP-5	JP-21	LD 0FA,#I	DRSZ 0FA	LD A,[X+]	LD A,[B+]	LD [B+],#I	INCA	SBIT 2,[B]	RBIT 2,[B]	LD B,#05	IFBNE 0A	JSR xA00-xAFF	JMP xA00-xAFF	JP+27	JP+11	A
JP-4	JP-20	LD 0FB,#I	DRSZ 0FB	LD A,[X-]	LD A,[B-]	LD [B-],#I	DECA	SBIT 3,[B]	RBIT 3,[B]	LD B,#04	IFBNE 0B	JSR xB00-xBFF	JMP xB00-xBFF	JP+28	JP+12	B
JP-3	JP-19	LD 0FC,#I	DRSZ 0FC	LD Md,#I	JMPL	X A,Md	POPA	SBIT 4,[B]	RBIT 4,[B]	LD B,#03	IFBNE 0C	JSR xC00-xCFF	JMP xC00-xCFF	JP+29	JP+13	C
JP-2	JP-18	LD 0FD,#I	DRSZ 0FD	DIR	JSRL	LD A,Md	RETSK	SBIT 5,[B]	RBIT 5,[B]	LD B,#02	IFBNE 0D	JSR xD00-xDFF	JMP xD00-xDFF	JP+30	JP+14	D
JP-1	JP-17	LD 0FE,#I	DRSZ 0FE	LD A,[X]	LD A,[B]	LD [B],#I	RET	SBIT 6,[B]	RBIT 6,[B]	LD B,#01	IFBNE 0E	JSR xE00-xEFF	JMP xE00-xEFF	JP+31	JP+15	E
JP-0	JP-16	LD 0FF,#I	DRSZ 0FF	*	*	LD B,#I	RETI	SBIT 7,[B]	RBIT 7,[B]	LD B,#00	IFBNE 0F	JSR xF00-xFFF	JMP xF00-xFFF	JP+32	JP+16	F

where: #I is the immediate data — Md is a directly addressed memory location —* is an unused opcode— NOTE: Opcode 60 Hex is also the opcode for IFBIT #I,A

Chapter 13

COP884BC

13.1 INTRODUCTION

The COP884BC is a member of the COP888 Feature Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications. The COP884BC has a combination of several additional features: two comparators, a High Speed 8-bit PWM Timer, a Controller Area Network (CAN) Interface, and reduced electromagnetic radiated emissions. Emissions are reduced by using internal power supply filters, a low-current crystal oscillator, and gradual turn-on of output drivers.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP884BC.

- Program Memory: Mask-programmed ROM, 2K bytes
- Data Memory: Static RAM, 64 bytes
- 16-Bit Programmable Timers: Timer T1
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- Additional features: two Comparators, a high speed PWM Timer, CAN Interface, and Reduced Electromagnetic Emissions

Several versions of the COP884BC are available that operate over different voltage and temperature ranges. Refer to the COP884BC data sheet for more specific information.

This chapter describes the device-specific features of the COP884BC. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual. Information on the CAN Interface is provided in Chapter 11.

13.2 BLOCK DIAGRAM

Figure 13-1 is a block diagram showing the basic functional blocks of the COP884BC. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

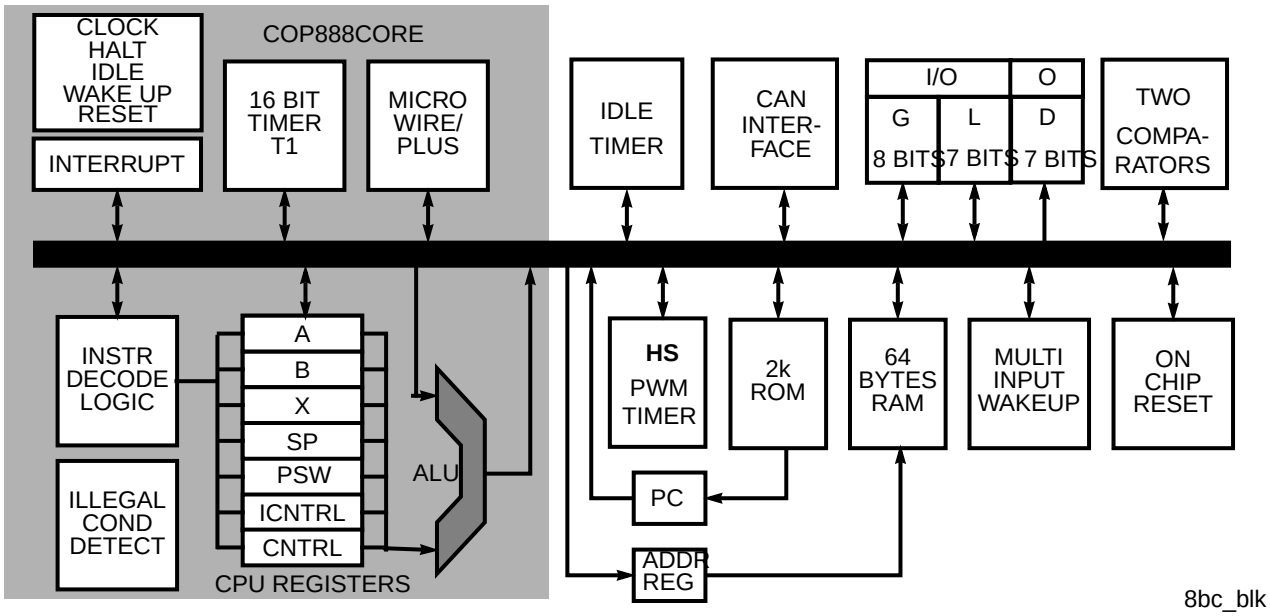


Figure 13-1 COP884BC Block Diagram

13.3 DEVICE PINOUTS/PACKAGES

The COP884BC is available in 28-pin SOIC. Figure 13-2 shows the COP884BC device package pinouts.

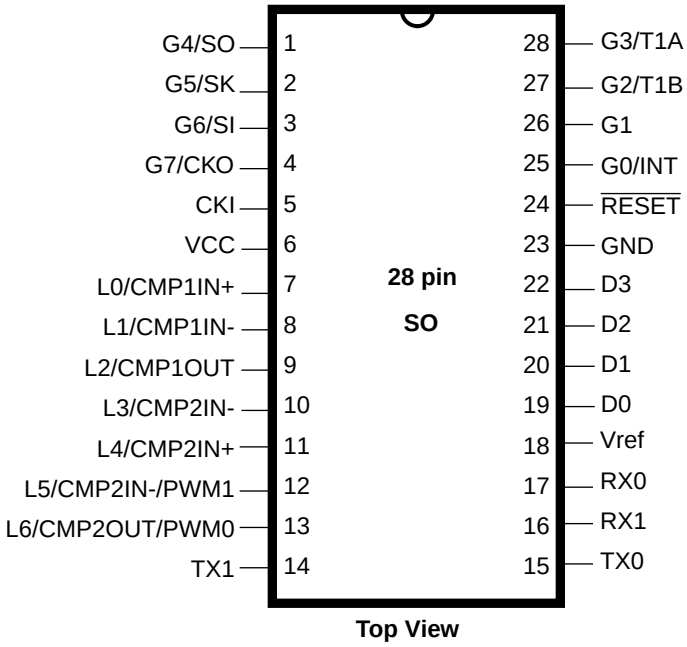


Figure 13-2 Device Package Pinouts

Refer to the COP884BC data sheet for more detailed package information.

13.4 PIN DESCRIPTIONS

The COP884BC has the following dedicated function pins: V_{CC}, GND, CKI, $\overline{\text{RESET}}$, CAN Vref, CAN Tx0, CAN Tx1, CAN Rx0, and CAN Rx1. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. The CAN pins are dedicated to the CAN Interface. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 13-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.

Table 13-1 COP884BC Pinouts

Port Pin	Type	ALT Function	28 PIN SO
G0	I/O	INTR	25
G1	I/O		26
G2	I/O	T1B	27
G3	I/O	T1A	28
G4	I/O	SO	1
G5	I/O	SK	2
G6	I	SI	3
G7	I	CKO	4
L0	I/O	CMP1IN+/MIWU	7
L1	I/O	CMP1IN-/MIWU	8
L2	I/O	CMP1OUT/MIWU	9
L3	I/O	CMP2IN-/MIWU	10
L4	I/O	CMP2IN+/MIWU	11
L5	I/O	CMP2IN-/PWM1/MIWU	12
L6	I/O	CMP2OUT/PWM0/FMONIN/ MIWU	13
D0	O		19
D1	O		20
D2	O		21
D3	O		22
CAN Vref			18
CAN Tx0	O		15
CAN Tx1	O		14
CAN Rx0	I	MIWU[a]	17

Table 13-1 COP884BC Pinouts (Continued)

Port Pin	Type	ALT Function	28 PIN SO
CAN Rx1	I	MIWU	16
Vcc			6
GND			23
CKI	I		5
RESET	I		24

a. The MIWU function for the CAN Interface is internal (see CAN Interface block diagram).

13.5 INPUT/OUTPUT PORTS

Refer to Table 13-1, COP884BC Pinouts, for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP884BC device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate function of the port pin.

Port D, a general-purpose dedicated output port, is present on all COP884BC devices. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or a zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP884BC devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected.

This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port L, a general-purpose bidirectional I/O port, is present in all COP884BC devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for Multi-Input Wakeup/Interrupt, an alternate function. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/ Interrupt feature.

13.6 PROGRAM MEMORY

The COP884BC contains 2K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space from 0000 to 03FF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs.

13.7 DATA MEMORY

The COP884BC contains 64 bytes of static RAM memory, used for temporary data storage. The RAM occupies two address ranges in the lowest 256-byte data memory space: 48 bytes from 00–2F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers.

13.8 REGISTER BIT MAPS

The COP884BC has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 13-2 through 13-10 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4) or Power Save Modes (Chapter 6).

Table 13-2 PWMCON, PWM Timer Control Register (Address 00A2)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	ESEL	PWPND	PWIE	PWMD	PWON	PWEN1	PWEN0

Unused:	This bit is unused
ESEL:	Capture/Frequency Monitor Mode Edge Select bit
PWPND:	PWM Timer Interrupt Pending bit
PWIE:	PWM Timer Interrupt Pending bit
PWMD:	PWM Timer Operating Mode Control bit
PWON:	PWM Timer Start bit
PWEN1-PWEN0:	PWM0, PWM1 Alternate Function Control bits.

Table 13-3 CTIM, CAN Bus Timing Register (Address 00B9)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PPS2	PPS1	PPS0	PS2	PS1	PS0	Unused	

PPS2-PPS0:	Propagation Segment Length Control bits
PS2-PS0:	Phase Segment Length Control bits
Unused:	These bits are unused

Table 13-4 CBUS, CAN Bus Control Register (Address 00BA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	RIAF	TXEN1	TXEN0	RXREF1	RXREF0	Reserved	FMOD

Reserved:	These bits are reserved and should not be used.
RIAF:	Receive ID Acceptance Filter Disable bit (receive all frames)
TXEN1-TXEN0:	Transmit Output Enable/CAN Enable bits
RXREF1-RXREF0:	Receive Input to V_{REF} Control bits
FMOD:	Advanced Fault Confinement Mode Enable bit

Table 13-5 TCNTL, CAN Bus Transmit Control/Status Register (Address 00BB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NS1	NS0	TERR	RERR	CEIE	TIE	RIE	TXSS

NS1-NS0:	Node Status bits
TERR:	Transmit Error Pending flag
RERR:	Receive Error Pending flag
CEIE:	CAN Error (transmit/receive) Interrupt Enable
TIE:	Transmit Interrupt Enable
RIE:	Receive Interrupt Enable
TXSS:	Transmit Start/Stop Control bit

Table 13-6 RTSTAT, CAN Bus Receive/Transmit Status Register (Address 00BC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TBE	TXPND	RRTR	ROLD	RORN	RFV	RCV	RBF

TBE:	Transmit Buffer Empty
TXPND:	Transmission Pending
RRTR:	Received Remote Transmission Request
ROLD:	Received Overload Frame
RORN:	Receiver Overrun
RFV:	Received Frame is Valid
RCV:	Receive Mode Status bit
RBF:	Receive Buffer Full

Table 13-7 CMPSL, Comparator Control Register (Address 00D3)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMP2SEL	CMP2OE	CMP2RD	CMP2EN	CMP1OE	CMP1RD	CMP1EN	Unused

CMP2SEL: Comparator 2 Output Select

CMP2OE: Comparator 2 Output Enable

CMP2RD: Comparator 2 Output Data

CMP2EN: Comparator 2 Enable

CMP1OE: Comparator 1 Output Enable

CMP1RD: Comparator 1 Output Data

CMP1EN: Comparator 1 Enable

Table 13-8 ICNTRL, Interrupt Control Register (Address 00E8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN: Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)

T0PND: Timer T0 (IDLE Timer) interrupt pending flag

T0EN: Timer T0 (IDLE Timer) interrupt enable bit

µWPND: MICROWIRE interrupt pending flag

µWEN: MICROWIRE interrupt enable bit

T1PNDB: Timer T1 interrupt B pending flag

T1ENB: Timer T1 interrupt B enable bit

Table 13-9 CNTRL, Control Register (Address 00EE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0
T1C3-T1C2-T1C1-T1C0: Timer T1 control bits							
MSEL: MICROWIRE Select bit							
IEDG: External Interrupt Edge selection bit							
SL1–SL0: MICROWIRE clock divide-by selection bits							

Table 13-10 PSW, Processor Status Word Register (Address 00EF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE
HC: Half-Carry bit							
C: Carry bit							
T1PNDA: Timer T1 interrupt A pending flag							
T1ENA: Timer T1 interrupt A enable bit							
EXPND: External interrupt pending flag							
BUSY: MICROWIRE Busy flag							
EXEN: External interrupt enable bit							
GIE: Global Interrupt Enable							

13.9 MEMORY MAP

Table 13-11 is a memory map showing the organization of the data memory and the specific memory addresses of the COP884BC registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 13-11 Data Memory Map

Address	Contents
0000 to 002F	On-Chip RAM bytes (48bytes)
0030 to 007F	Unused RAM Address Space (Reads As All Ones)
0080 to 009F	Unused RAM Address Space (Reads Undefined Data)
00A0	PSCAL, PWM timer Prescaler Register
00A1	RLON, PWM timer On-Time Register
00A2	PWMCON, PWM Control Register
00B0	TXD1, Transmit 1 Data
00B1	TXD2, Transmit 2 Data
00B2	TDLC, Transmit Data Length Code and Identifier Low
00B3	TID, Transmit Identifier High
00B4	RXD1, Receive Data 1
00B5	RXD2, Receive Data 2
00B6	RIDL, Receive Data Length Code
00B7	RID, Receive Identify High
00B8	CSCAL, CAN Prescaler
00B9	CTIM, Bus Timing Register
00BA	CBUS, Bus Control Register
00BB	TCNTL, Transmit/Receive Control Register
00BC	RTSTAT Receive/Transmit Status Register
00BD	TEC, Transmit Error Count Register
00BE	REC, Receive Error Count Register
00BF	PLATST, CAN Bit Stream Processor Test Register
00C0 to 00C7	Reserved

Table 13-11 Data Memory Map

Address	Contents
00C8	WKEDG, MIWU Edge Select Register
00C9	WKEN, MIWU Enable Register
00CA	WKPND, MIWU Pending Register
00CB	Reserved
00CC	Reserved
00CD to 00CF	Reserved
00D0	PORTLD, Port L Data Register
00D1	PORTLC, Port L Configuration Register
00D2	PORTLP, Port L Input Pins (Read Only)
00D3	CMPSL, Comparator control register
00D4	PORTGD, Port G Data Register
00D5	PORTGC, Port G Configuration Register
00D6	PORTGP, Port G Input Pins (Read Only)
00D7 to 00DB	Reserved
00DC	PORTD, Port D output register
00DD to 00DF	Reserved for Port D
00E0-00E5	Reserved
00E6	T1RBLO, Timer T1 Autoload Register Lower Byte
00E7	T1RBHI, Timer T1 Autoload Register Upper Byte
00E8	ICNTRL, Interrupt Control Register
00E9	SIOR, MICROWIRE/PLUS Shift Register
00EA	TMR1LO, Timer T1 Lower Byte
00EB	TMR1HI, Timer T1 Upper Byte
00EC	T1RALO, Timer T1 Autoload Register Lower Byte
00ED	T1RAHI, Timer T1 Autoload Register T1RA Upper Byte
00EE	CNTRL, Control Register
00EF	PSW, Processor Status Word Register

Table 13-11 Data Memory Map

Address	Contents
00F0 to 00FB	On-Chip RAM Mapped as Registers
00FC	X Register
00FD	SP Register
00FE	B Register
00FF	Reserved[a]
Reading memory locations 0030H–007FH will return all ones. Reading unused memory locations 0080H–009FH will return undefined data.	

a. In devices with more than 128 bytes of RAM, location 0FF is used as the Segment register to switch between different Segments of RAM memory. In this device location 0FF can be used as a general purpose, on-chip RAM mapped register. However, the user is advised that caution should be taken in porting software utilizing this memory location to a chip with more than 128 bytes of RAM.

13.10 RESET

Upon reset of the COP884BC, the ports and registers are initialized as follows:

Port D data register, PORTD:	FF
Port G data register, PORTGD:	00
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timer T1 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.

Accumulator, A and Timers T1:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
Comparator Control Register, CMPSL:	Cleared
Pulse Width Modulation Control Register, PWMCON:	Cleared
CAN register bits:	All cleared to zero except for the Transmit Buffer Empty (TBE) bit of the Receive/ Transmit Status (RTSTAT) register, which is set to 1; and the RID, RIDL, TID, and TDLC registers, which are unknown.

13.11 ON-CHIP POWER-ON RESET

The on-chip reset circuit is a mask-selectable option. If enabled, it resets the device when the operating voltage (Vcc) goes below the specified on-chip reset voltage level. The device is held in reset when Vcc stays below this voltage level. The device comes out of reset as Vcc rises from a voltage below the on-chip reset voltage level to a voltage above this level. If a two pin crystal/resonator clock option is selected, the on-chip reset will trigger a 256 t_c delay. This delay allows the oscillator to stabilize before the device exits the reset state. The contents of data registers and RAM are unknown following an on-chip reset. The External reset takes priority over the on-chip reset and will deactivate the 256 t_c cycle delay if any is in progress.

13.12 INTERRUPTS

Table 13-12 shows the types of interrupts in the COP884BC, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 13-12 COP884BC Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	CAN Receive Interrupt	01FA–01FB
4	CAN Error Interrupt (transmit/receive)	01F8–01F9
5	CAN Transmit Interrupt	01F6–01F7
6	External Interrupt Pin G0	01F4–01F5
7	IDLE Timer Underflow	01F2–01F3
8	Timer T1A/Underflow	01F0–01F1
9	Timer T1B	01EE–01EF
10	MICROWIRE/PLUS	01EC–01ED
11	PWM Timer	01EA–01EB
12	(Reserved)	01E8–01E9
13	(Reserved)	01E6–01E7
14	(Reserved)	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

13.13 COMPARATORS

The COP884BC has two differential comparators. Port L is used for the comparators. The output of the comparators is multiplexed out to two pins. The following are the Port L assignments:

L0 Comparator 1 positive input

L1 Comparator 1 negative input

L2 Comparator 1 output

L3 Comparator 2 negative input

L4 Comparator 2 positive input

L5 Comparator 2 negative input

L6 Comparator 2 output

The Port L pins can also be used for Multi-Input Wakeup/Interrupt (see Chapter 7) and for the PWM timer (see Section 13.14).

Each comparator output can be connected internally to the L-Port pin of the respective positive input and thereby generate an interrupt using the L-Port interrupt structure. Port L interrupts are controlled by the WKEDG (Wakeup Edge), WKEN (Wakeup Enable), and WKPND (Wakeup Pending) registers.

The application program must clear the associated WKEN bit before enabling the comparator and must clear the associated WKPND bit after enabling the comparator and before setting WKEN. This will prevent unwanted interrupts from being generated by transitions on the comparator outputs during enabling.

Figure 13-3 and 13-4 show block diagrams of the two comparator circuits.

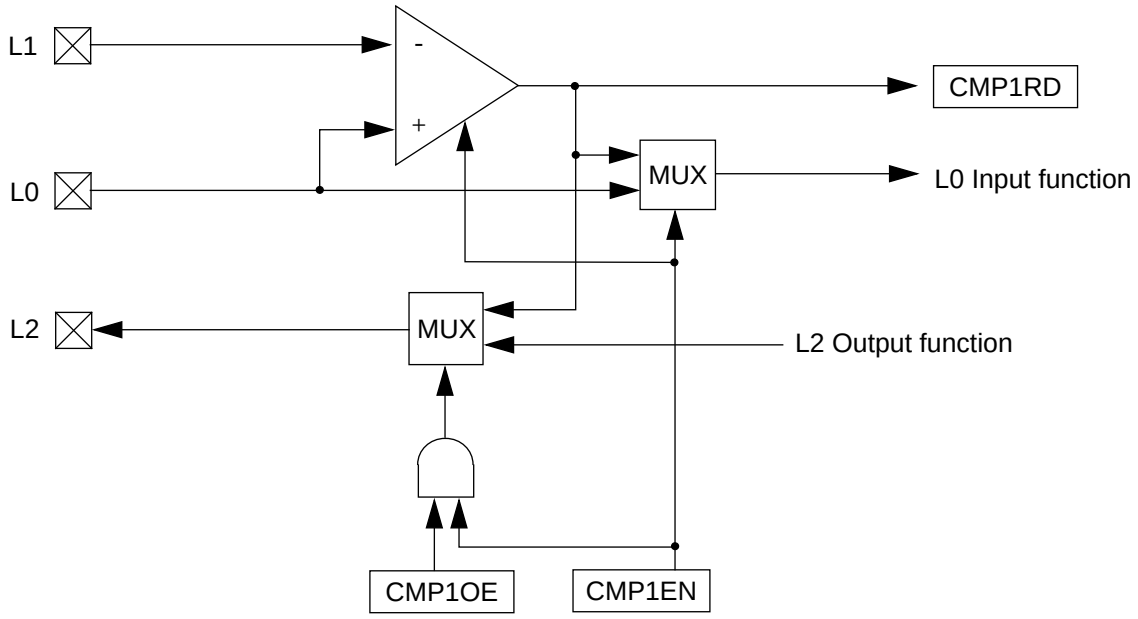
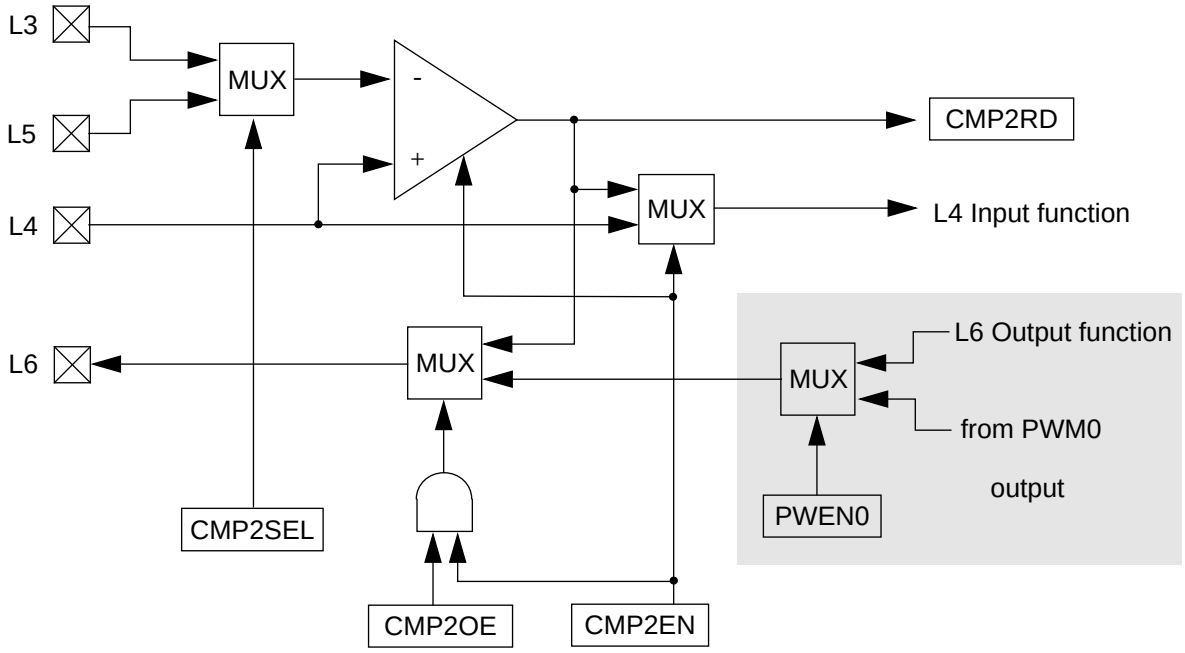


Figure 13-3 Comparator 1 Block Diagram

Figure 13-4 shows that pin L6 has an alternate function of supporting the PWM0 output. The comparator 2 output must be disabled in order to use the PWM0 output on L6.



NOTE: The SHADED area shows logic from PWM Timer. Comparator 2 output (CMP2OE) must be disabled in order to use PWM0 output.

Figure 13-4 Comparator 2 Block Diagram

13.13.1 COMPARATOR CONTROL REGISTER (CMPSL) (00D3)

The comparators are controlled by the Comparator Control Register. The register format is shown in Table 13-13.

Table 13-13 Comparator Control Register (CMPSL) (00D3)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CMP2SEL	CMP2OE	CMP2RD	CMP2EN	CMP1OE	CMP1RD	CMP1EN	UNUSED

The register contains the following bits:

CMP1EN Enables Comparator 1 ("1"=enable). If Comparator 1 is disabled the associated L-pins can be used as standard I/O.

CMP1RD Reads Comparator 1 output internally (CMP1EN=1) Read-only, reads as a "0" if comparator not enabled.

CMP1OE Enables Comparator 1 output ("1"=enable), CMP1EN bit must be set to enable this function.

CMP2EN Enables Comparator 2 ("1"=enable). If Comparator 2 is disabled the associated L-pins can be used as standard I/O.

CMP2RD Reads Comparator 2 output internally (CMP2EN=1) Read-only, reads as a "0" if comparator not enabled.

CMP2OE Enables Comparator 2 output ("1"=enable), CMP2EN bit must be set to enable this function.

CMP2SEL Selects the L port pin to use for Comparator 2 negative input. (CMP2SEL = 0 selects L5; CMP2SEL = 1 selects pin L3)

The Comparator Select/Control bits are cleared on Reset (the comparator is disabled). To save power, the program should also disable the comparator before the device enters the HALT mode.

The Comparator rise and fall times are symmetrical. The application program must set up the Configuration and Data registers of Port L correctly for comparator operation. For the AC and DC characteristics of the comparators, refer to Chapter 4.

13.14 PWM TIMER

The COP884BC has a high-speed, processor-independent Pulse Width Modulation (PWM) timer. This timer is provided in addition to the T1 core timer that can also be configured to operate in a PWM mode.

The COP884BC PWM timer is an 8-bit timer driven by the CKI internal hardware clock, which has a maximum rate of 10 MHz. A slower timer can be implemented by dividing the CKI block by a prescaler factor. The prescaler factor can be set to any value from 1 to 256.

The timer can be configured to operate in either of two modes: the PWM mode or the frequency monitor mode. In the PWM mode, the timer produces a PWM signal at a specified rate and duty cycle. In the frequency monitor mode, the timer determines whether the frequency of an external signal exceeds a specified threshold value.

Figure 13-5 shows a block diagram of the PWM timer configured to operate in the PWM mode. Figure 13-6 shows the same timer configured to operate in the frequency monitor mode.

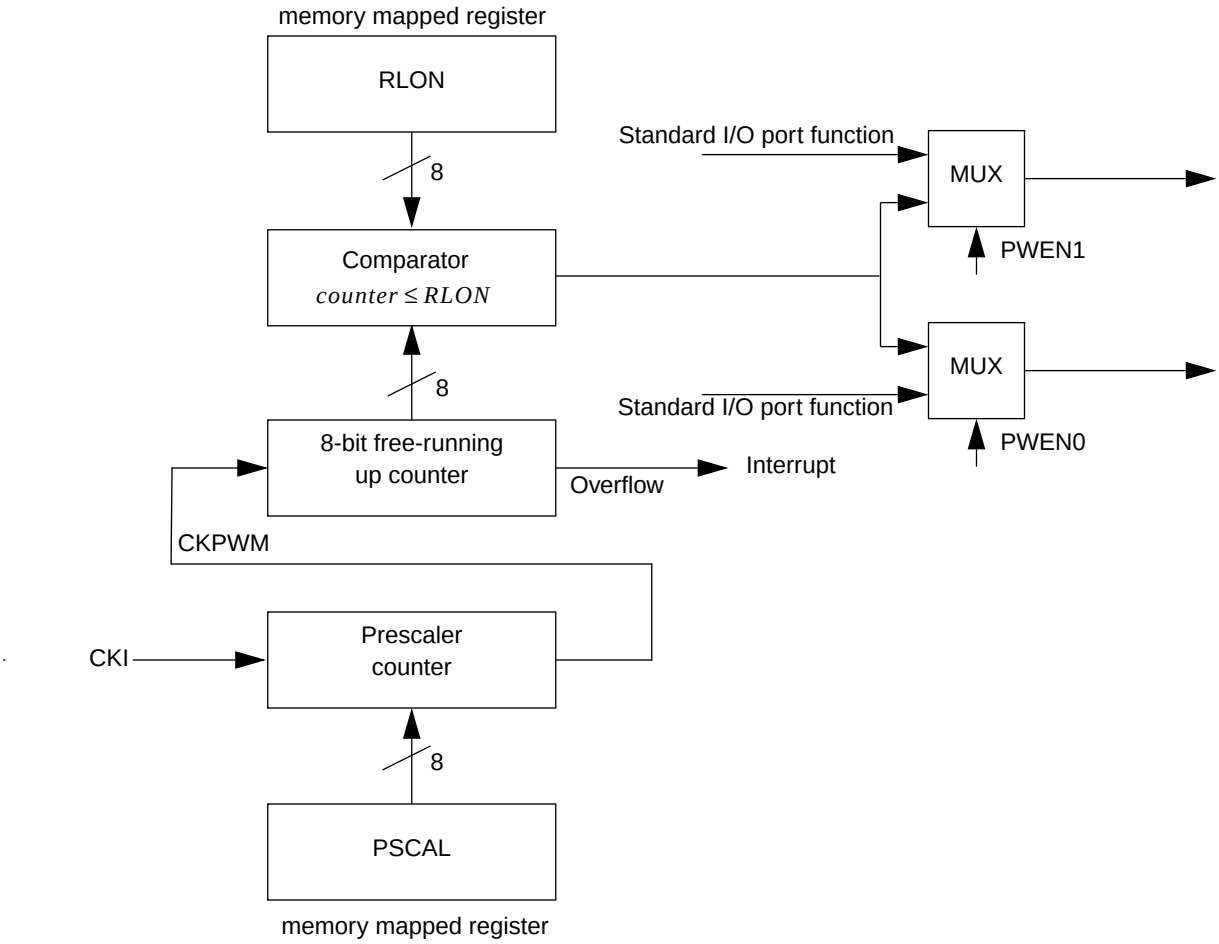


Figure 13-5 PWM Timer Block Diagram (PWM Mode)

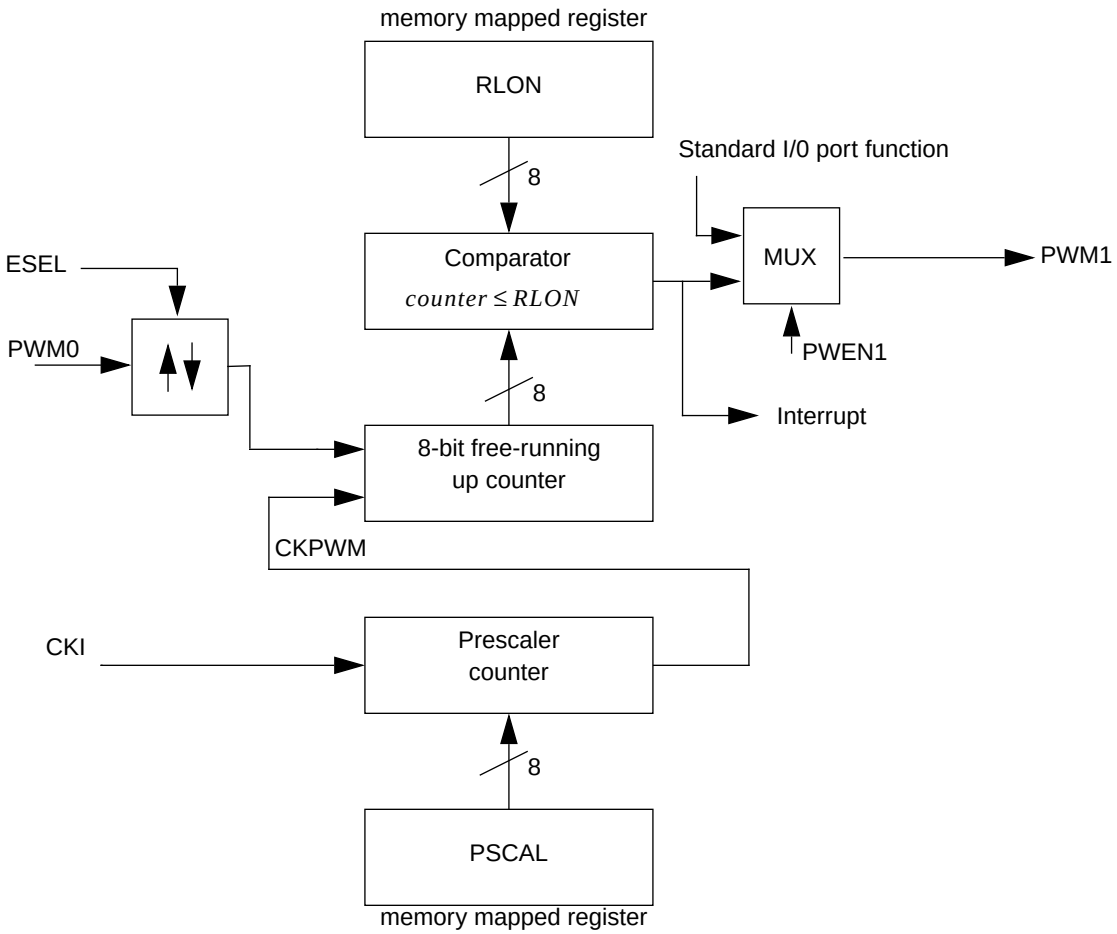


Figure 13-6 PWM Timer Block Diagram (Capture Mode)

The PWM timer is controlled by three registers: the PWM Control Register (PWMCON), the PWM Prescaler Register (PSCAL), and the PWM On-Time Register (RLON).

13.14.1 PWM Mode

The Pulse Width Modulation (PWM) mode can be used to generate precise pulses of known width and duty cycle on the PWM0 and/or PWM1 pin. PWM0 and PWM1 are alternate functions of the L6 and L5 port pins.

The timer is driven by the internal hardware clock, CKI, operating at a maximum rate of 10 MHz. This clock frequency is divided by the prescaler factor to produce the PWM clock. The prescaler factor is 1 plus the value set in the PSCAL register. The register can be set to any value from 0 to 255, producing a prescaler factor of 1 to 256.

The timer generates a PWM signal with a fixed repetition rate of 256 PWM clock cycles. The resulting period of the PWM signal is:

(CLI clock period) X (prescaler factor) X 256

With a 10 MHz CKI clock, you can obtain the fastest possible PWM clock rate of 39 KHz by using a prescaler factor of 1; or you can obtain the slowest possible PWM clock rate of is 153 Hz by using a prescaler factor of 256.

The PWM timer is based on a free-running 8-bit up-counter that counts continuously at the PWM clock rate. When the count is less than the value in the RLON register, the timer output is high. When the count is greater than or equal to the value in the RLON register, the timer output is low. The PWM Enable bits in the PWM Control Register, PWEN0 and PWEN1, determine whether this output signal appears on pin PWM0 or PWM1 (or both).

When the application program sets the PWM On (PWON) bit in the PWM Control Register, the free-running up counter is initialized and generation of the PWM signal begins. The timer operates continuously, independent of any processor activity, until it is stopped by clearing the PWON bit in the PWM Control Register.

The value in the RLON register determines the duty cycle of the PWM signal. A register value of N produces a high output signal for N/256 of the PWM signal period. Thus, the resolution of the duty cycle is 1/256 of the PWM period. For example, a register value of 64 results in a PWM signal that is high for 64/256 of the PWM period and low for the remaining 192/256 of the period.

Figure 13-7 shows how the PWM output is generated for various values programmed into the RLON register. Note that a register value of 0 results in a signal that is always low, and a register value of 256 results in a signal that is always high.

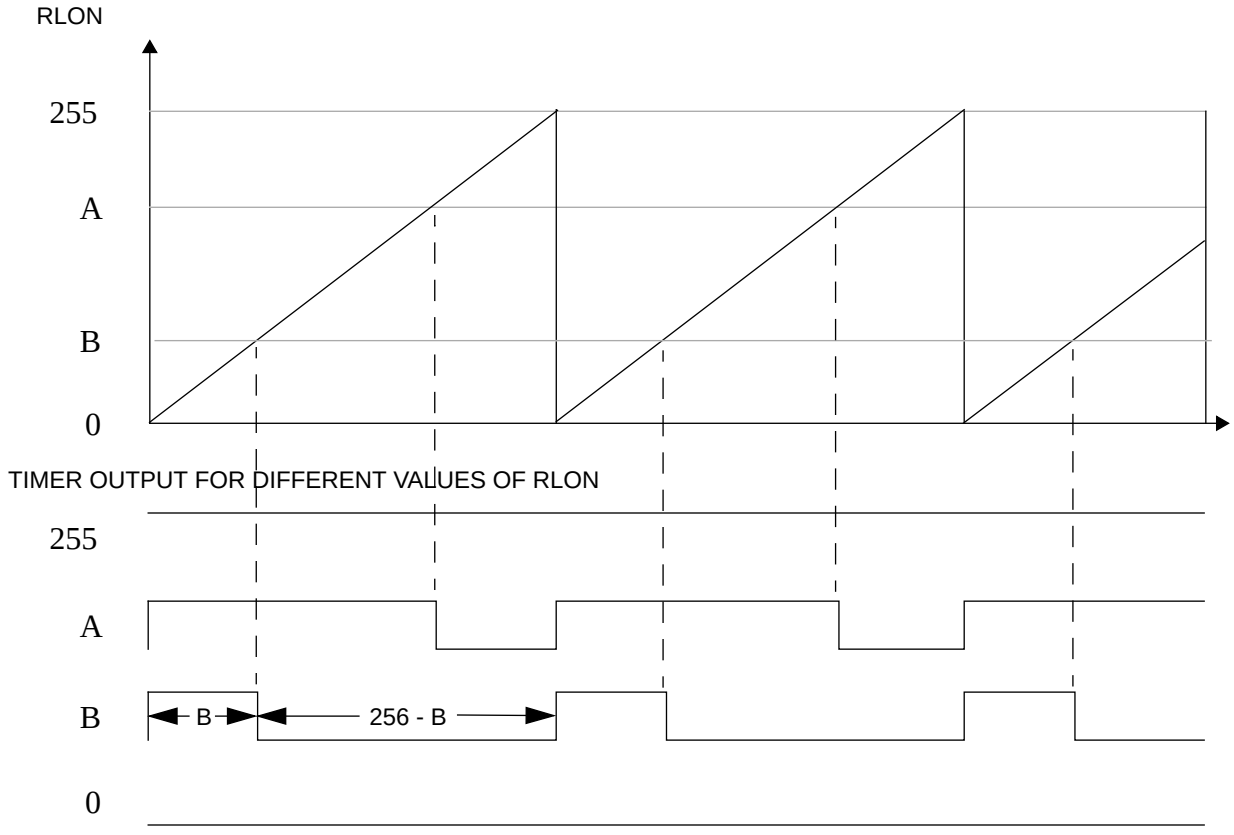


Figure 13-7 PWM Mode Operation.

The application software can modify the RLON register contents while the PWM timer is running. For the PWM mode, the previous RLON register setting remains in effect until the next overflow of the free-running up-counter. When overflow occurs, the new RLON register contents take effect.

Each time the up-counter overflows, the PWM Interrupt Pending Flag (PWPND) in the PWM Control Register is set. If the PWM Interrupt Enable (PWIE) bit is set in the PWM Control Register, the overflow generates an interrupt. The PWPND remains set until cleared by the software.

NOTE: If the software tries to clear the PWPND bit at the same time that the counter overflows, the bit is still set by the hardware. Writing a 1 to the PWPND bit has no effect.

The following steps can be used to operate the timer in the PWM mode:

1. Configure the L5 and/or L6 pin as an output by setting bit 5 and/or 6 in the Port L configuration register.
2. Write the desired prescaler value into the PSCAL register. This controls the frequency of the PWM signal. The prescaler factor is 1 plus the value written to the register.
3. Write the desired PWM On-Time value into the RLON register. This controls the duty cycle of the PWM signal. The signal is high for N/256 of the PWM period, where N is the value written to the register.
4. Write the appropriate value to the PWM Control Register to enable the PWM mode, to enable the desired interrupt(s), to enable the desired output pin(s), and to start the timer.

13.14.2 Frequency Monitor Mode

In the frequency monitor mode, the timer determines whether the frequency of an external signal exceeds a specified threshold value. The timer senses either rising edges or falling edges at the Frequency Monitor Input (FMONIN) and compares the time intervals between successive edges against a value set in a register. It reports the result of this comparison on the PWM1 output. FMONIN is an alternate function of the L6 port pin and PWM1 is an alternate function of the L5 port pin.

The timer is driven by the internal hardware clock, CKI, operating at a maximum rate of 10 MHz. This clock frequency is divided by the prescaler factor to produce the PWM clock. The prescaler factor is 1 plus the value set in the PSCAL register. The register can be set to any value from 0 to 255, producing a prescaler factor of 1 to 256.

With a 10 MHz CKI clock, you can obtain the fastest possible PWM clock rate of 39 KHz by using a prescaler factor of 1; and you can obtain the slowest possible PWM clock rate of is 153 Hz by using a prescaler factor of 256.

The timer is based on a 8-bit up-counter that counts at the PWM clock rate. The count is reset to zero each time an active edge is sensed at the FMONIN input. You can select either rising or falling edges by programming the ESEL bit in the PWM Control Register. If no edge is received at the FMONIN input, the count returns to zero when overflow

occurs. When the count is less than the value in the RLON register, the timer output is low. When the count is greater than or equal to the value in the RLON register, the timer output is high.

The output of the timer indicates whether the frequency of the input signal exceeds a specified threshold. If the frequency of the input signal is high enough, the counter will be reset before it reaches the value set in the RLON register, and the output will always be low. However, if the time interval between active edges is long enough, the output will go high whenever the count exceeds the value set in the RLON register. This effect is illustrated in Figure 13-8.

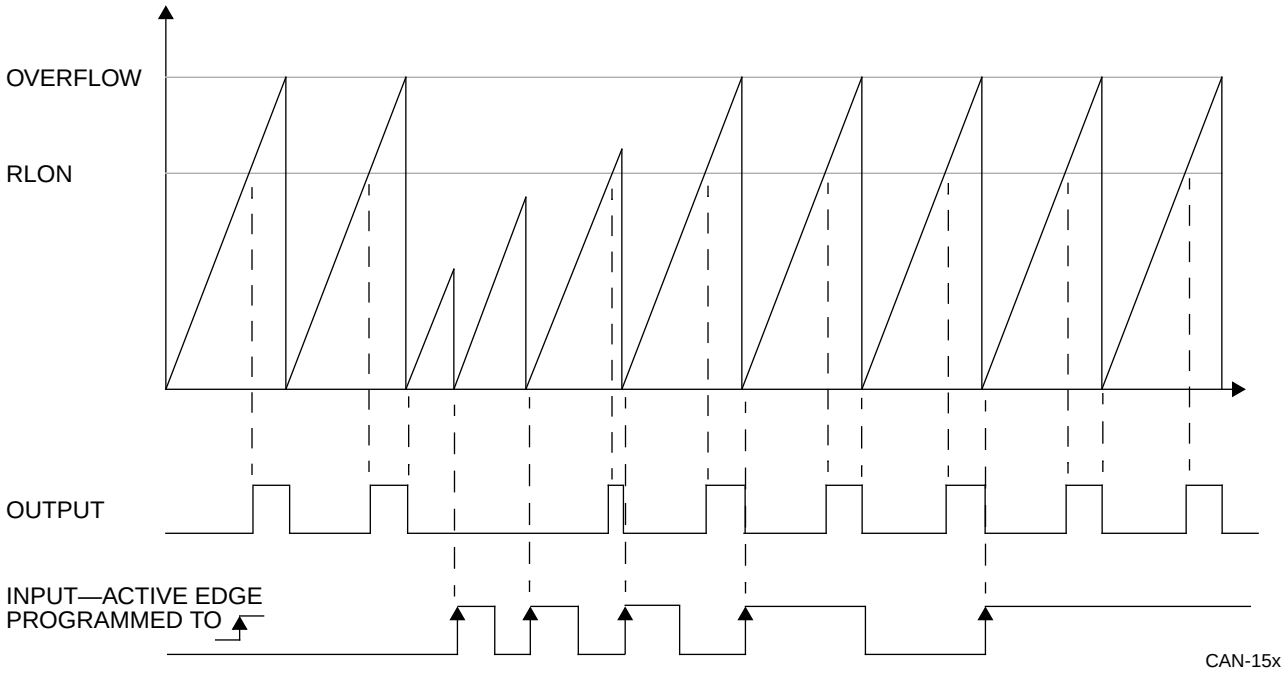


Figure 13-8 Frequency Monitor Mode Operation.

When the application program sets the PWM On (PWON) bit in the PWM Control Register, the counter is initialized and monitoring of the input signal begins. The timer operates continuously until it is stopped by clearing the PWON bit in the PWM Control Register.

The value in the RLON register and the prescaler factor determine the tested time interval. For example, with a CKI rate of 10 MHz, a prescaler factor of 4 (a value of 3 in the PSCAL register), and a value of 64 (40H) in the RLON register, the tested time interval is:

0.1 µsec X 4 X 64 = 25.6 µsec

In this case, if the input is a clock with a frequency of 39.06 KHz or higher, the output of the timer is always low. Otherwise, the output of the timer goes high whenever the elapsed time between input edges exceeds 25.6 µsec. In the absence of any input edges, the counter overflows every 102.4 µsec, resulting in a pulse train with that is low for 25.6 µsec and high for 76.8 µsec.

The application software can modify the RLON register contents while the PWM timer is running. For the frequency monitor mode, the new RLON register contents take effect immediately.

When the count reaches the value set in the RLON register, the PWM Interrupt Pending Flag (PWPND) in the PWM Control Register is set. If the PWM Interrupt Enable (PWIE) bit is set in the PWM Control Register, this event generates an interrupt. The PWPND remains set until cleared by the software.

The following conditions will also cause the PWPDN flag to be set:

- The timer operating mode is changed from the PWM mode to the frequency monitor mode "on the fly" and the timer output changes from 0 to 1.
- The RLON register is changed "on the fly" so that RLON becomes less than the current count in the 8-bit counter.

NOTE: If the software tries to clear the PWPND bit at the same time as a frequency monitor interrupt event, the bit is still set by the hardware. Writing a 1 to the PWPND bit has no effect.

The following steps can be used to operate the timer in the frequency monitor mode:

1. Configure the L5 pin as an output and the L6 pin as an input by setting bit 5 and clearing bit 6 in the Port L configuration register.
2. Write the desired prescaler value into the PSCAL register. This controls the frequency of the clock for the 8-bit counter. The prescaler factor is 1 plus the value written to the register.
3. Write the desired PWM On-Time value into the RLON register. This specifies the time interval threshold tested by the frequency monitor. The time value is:

   (CKI clock period) X (prescaler factor) X (RLON register value)

4. 4. Write the appropriate value to the PWM Control Register to enable the frequency monitor mode, to enable the PWM interrupt (if desired), to enable the PWM1 output pin, to specify the input edge type (rising or falling), and to start the timer.

13.14.3 PWM Registers

The PWM timer is controlled by three registers: the PWM Control Register (PWMCON), the PWM Prescaler Register (PSCAL), and the PWM On-Time Register (RLON).

PWM Prescaler Register, PSCAL (00A0)

The PWM Prescaler Register, PSCAL, is a read/write register that specifies the divide-by factor for generating the PWM clock from the CKI clock. The prescaler factor is 1 plus the value written into the register. For example, write the value zero to use a prescaler factor of 1, which causes the 8-bit up-counter to operate at the maximum rate of 10 MHz; or

write the value 255 to use a prescaler factor of 256, which causes the 8-bit up-counter to operate at the minimum rate of 39 KHz.

The Prescaler Register contents can be modified at any time, even when the timer is running. The change takes effect after a full PWM clock cycle is completed. In other words, the prescaler counter is decremented by the CKI clock and when underflow occurs, the prescaler counter is reloaded from the PSCAL register.

PWM On-Time Register, RLON (00A1)

The PWM On-Time Register, RLON, is a read/write register that specifies the PWM duty cycle in PWM mode or the tested frequency threshold in the frequency monitor mode. The timer output is 0 when the count in the 8-bit counter is less than the value in the RLON register, and 1 when the count equals or exceeds this value.

In the PWM mode, the timer output signal is high for N/256 of the PWM period, where N is the value written to the RLON register. The result of writing a new value to the RLON register occurs after the current PWM pulse is completed.

In the frequency monitor mode, the timer output signal goes high whenever the elapsed time between successive clock edges of the input signal exceeds the number of PWM clock cycles specified in the RLON register. The result of writing a new value to the RLON register occurs immediately.

PWM Control Register, PWMCON (00A2).

Table 13-14 PWM Control register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
unused	ESEL	PWPND	PWIE	PWMD	PWON	PWEN1	PWEN0

The PWM Control Register, PWMCON, controls the high-speed PWM timer. The register bits are assigned as follows:

PWEN0 PWM0 input/output pin function enable

PWEN1 PWM1 input/output pin function enable

PWON PWM on bit: 1 to start PWM generator or frequency monitor; 0 to stop

PWMD PWM mode bit: 1 for PWM mode or 0 for frequency monitor mode

PWIE PWM interrupt enable bit

PWPND PWM interrupt pending flag bit

ESEL Edge selection bit for frequency monitor mode: 1 for falling edge detection or 0 for rising edge detection

NOTE: If you enable the PWM mode without starting the timer (PWMD=1 and PWON=0 in the PWM Control Register), the timer output is set to 1. If you enable the capture mode without starting the timer (PWMD=0 and PWON=0 in the PWM Control Register), the timer output is set to 0.

13.15 CAN INTERFACE

The COP884BC has a built-in Controller Area Network (CAN) interface. CAN is a communications network standard often used in automotive and industrial control applications.

In a CAN system, multiple modules share a common serial communications bus. Modules communicate with one another by sending and receiving packets of data called "frames." Each message has a unique identifier so that messages can be sent and received unambiguously between different modules. The CAN standard specifies the protocol for message transmission, for resolving bus conflicts, and for handling errors.

The COP884BC supports the CAN Specification Revision 2.0 Part B, "Passive" type operation, at data rates of up to 1 Mbits/second using standard message identifiers. This interface is suitable for lower-speed (CAN class A and B) applications. For example, in automotive applications, it is suitable for control of instrument gauges, door locks, headlights, and audio systems.

For complete information on the CAN Interface, see Chapter 11.

13.16 MASK OPTIONS

The COP884BC has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

OPTION 1: Clock Configuration

=1 Crystal Oscillator (CKI/10)

G7 (CKO) is clock generator output to crystal/resonator

CKI is the clock input

OPTION 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

OPTION 3: Bond Out

=1 28-Pin SO

OPTION 4: ON-CHIP RESET

=1 Enable ON-CHIP POWER-ON RESET

=2 Disable ON-CHIP POWER-ON RESET

Chapter 14

COP888CS/CG/EG/GG/HG/KG

14.1 INTRODUCTION

The COP888CS, COP888CG, COP888EG, COP888GG, COP888HG and COP888KG are members of the COP888 Feature Family 8-bit microcontrollers. Like other members of this family, these devices provide high-performance, economical solutions for embedded control applications. Each of these devices contains a Universal Asynchronous Receiver/ Transmitter (UART), at least one comparator, and additional RAM memory beyond the family minimum of 128 bytes. These devices are described together in this chapter because of their similarity, and are together referred to as the COP888CS/CG/EG/GG/ HG/KG. They differ only in the number of comparators, number of timers, amount of RAM, and amount of ROM available.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888CS/CG/EG/GG/HG/KG.

- Program Memory: Mask-programmed ROM, 4K bytes (CS and CG), 8K bytes (EG), 16K bytes (GG), 20K bytes (HG) or 24K bytes (KG)
- Data Memory: Static RAM, 192 bytes (CS and CG), 256 bytes (EG), 512 bytes (GG, HG) or 1K bytes (KG)
- S-Register Data Memory Extension: Yes
- 16-Bit Programmable Timers: Timer T1 , T2 and T3 (CG,EG,GG,HG and KG only, CS has Timer T1 only)
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- WATCHDOG and Clock Monitor: Yes
- Additional features: UART and comparators (one comparator for the CS and two for the other devices)

Several versions of the COP888CS/CG/EG/GG/HG/KG are available that operate over different voltage and temperature ranges. Refer to the appropriate data sheet for more specific information.

This chapter describes the device-specific features of the COP888CS/CG/EG/GG/HG/KG. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

14.2 BLOCK DIAGRAMS

Figure 14-1 is a block diagram showing the basic functional blocks of the COP888CG. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

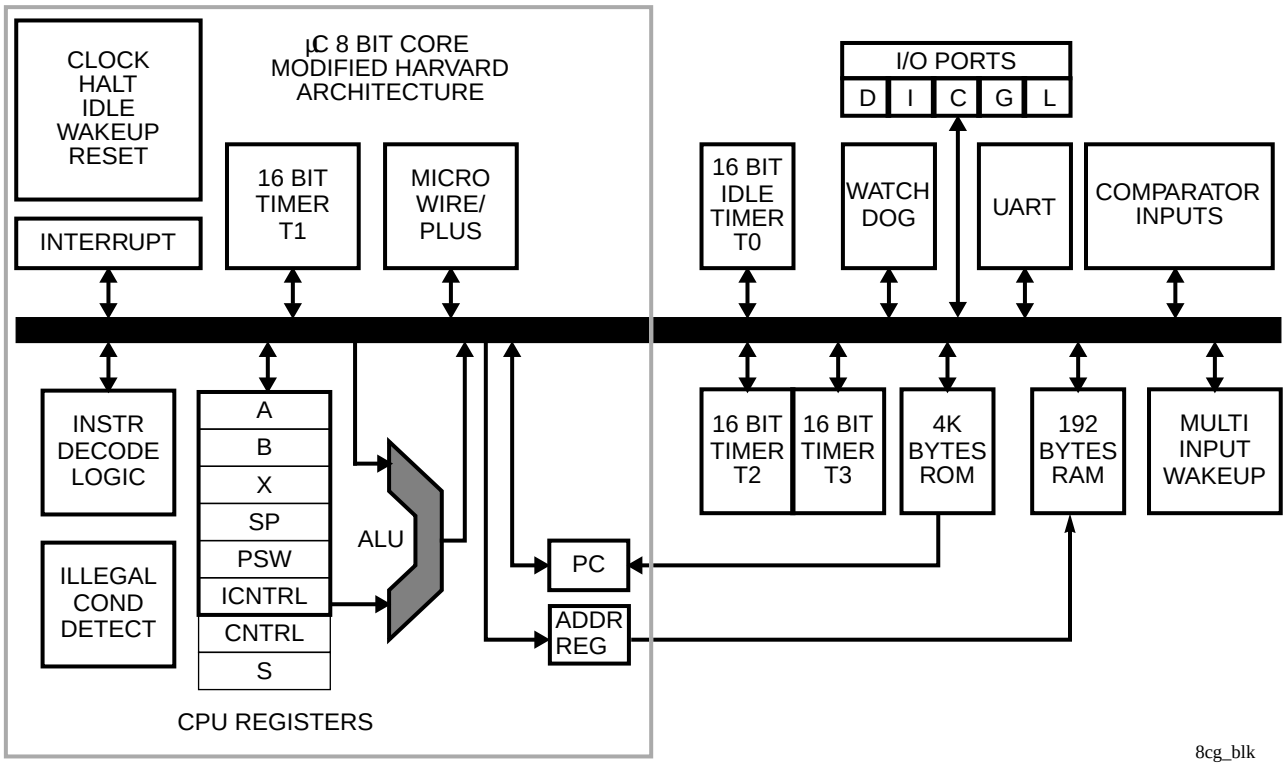


Figure 14-1 COP888CG Block Diagram

This block diagram also applies to the COP888EG/GG/HG/KG with one exception. The COP888EG/GG/HG/KG have more memory (8K bytes ROM and 256 bytes RAM in EG, 16K bytes ROM and 512 bytes RAM in GG, 20K bytes ROM and 512 bytes RAM in HG and 24K bytes ROM and 1K bytes RAM in KG) than the COP888CG.

Figure 14-2 is a block diagram showing the basic functional blocks of the COP888CS. The COP888CS is identical to the COP888CG except it has one general-purpose timer (T1) rather than three, and has one comparator rather than two.

14.3 DEVICE PINOUTS/PACKAGES

The COP888CS/CG/EG are available in 28-pin DIP, 28-pin SO, 40-pin DIP and 44-pin PLCC packages, the COP888GG/HG/KG are available in 40-pin DIP and 44-pin PLCC packages. Figure 14-3 shows the COP888CS/CG/EG/GG/HG/KG device package pinouts.

Refer to the COP888CG, COP888EG, COP888GG, COP888HG, COP888KG and COP888CS data sheets for more detailed package information.

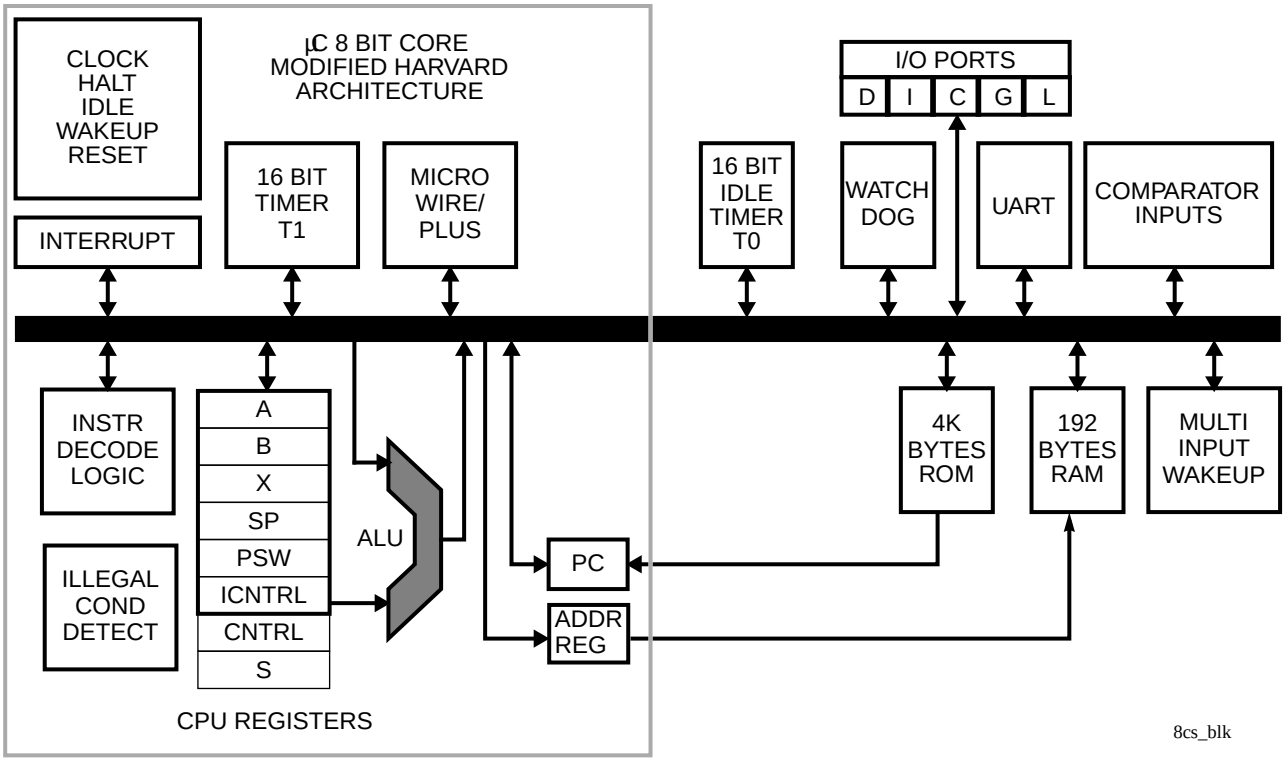


Figure 14-2 COP888CS Block Diagram

14.4 PIN DESCRIPTIONS

The COP888CS/CG/EG/GG/HG/KG has four dedicated function pins: V_{CC}, GND, CKI, and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 14-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types. This table is valid for the COP888CS, COP888CG, COP888EG, COP888GG, COP888HG and COP888KG, except that the alternate functions related to Timer T2, Timer T3, and Comparator 2 do not apply to the COP888CS.

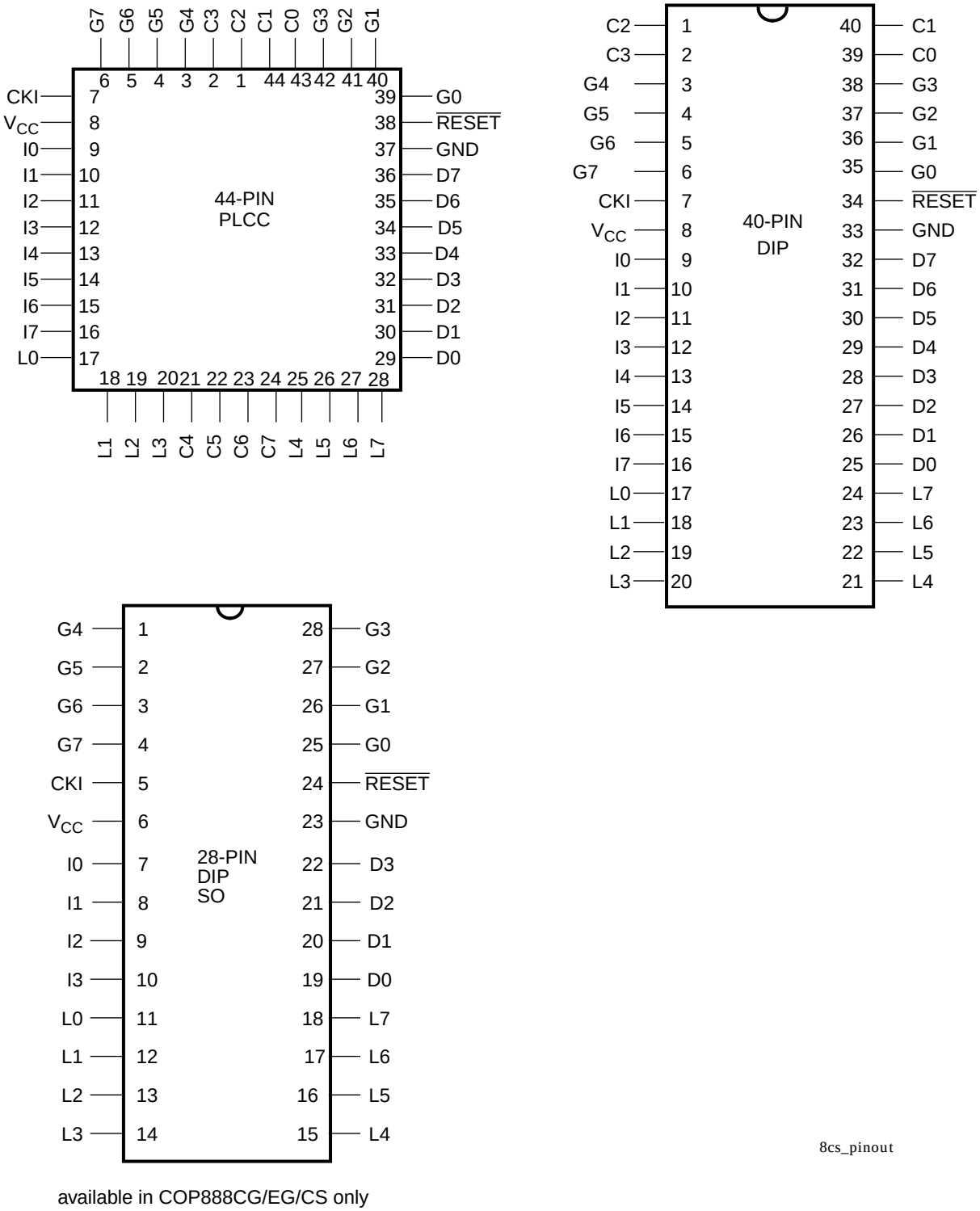


Figure 14-3 Device Package Pinouts

Table 14-1 COP888CS/CG/EG/GG/HG/KG Pinouts

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
L0	I/O	MIWU	11	17	17
L1	I/O	MIWU or CKX	12	18	18
L2	I/O	MIWU or TDX	13	19	19
L3	I/O	MIWU or RDX	14	20	20
L4	I/O	MIWU or T2A*	15	21	25
L5	I/O	MIWU or T2B*	16	22	26
L6	I/O	MIWU or T3A*	17	23	27
L7	I/O	MIWU or T3B*	18	24	28
G0	I/O	INT	25	35	39
G1	WDOUT		26	36	40
G2	I/O	T1B	27	37	41
G3	I/O	T1A	28	38	42
G4	I/O	SO	1	3	3
G5	I/O	SK	2	4	4
G6	I	SI	3	5	5
G7	I/CKO	HALT RESTART	4	6	6
D0	O		19	25	29
D1	O		20	26	30
D2	O		21	27	31
D3	O		22	28	32
I0	I		7	9	9
I1	I	COMP1IN–	8	10	10
I2	I	COMP1IN+	9	11	11
I3	I	COMP1OUT	10	12	12
I4	I	COMP2IN–*		13	13
I5	I	COMP2IN+*		14	14
I6	I	COMP2OUT*		15	15
I7	I			16	16
D4	O			29	33
D5	O			30	34
D6	O			31	35
D7	O			32	36

Table 14-1 COP888CS/CG/EG/GG/HG/KG Pinouts (Continued)

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
C0	I/O			39	43
C1	I/O			40	44
C2	I/O			1	1
C3	I/O			2	2
C4	I/O				21
C5	I/O				22
C6	I/O				23
C7	I/O				24
V_{CC}			6	8	8
GND			23	33	37
CKI			5	7	7
$\overline{\text{RESET}}$			24	34	38
* Not available on COP888CS					

14.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888CS/CG/EG/GG/HG/KG depend on the package type. The 28-pin packages have Port L, Port G, and a subset of Port D and Port I. The 40-pin packages have Port L, Port G, Port D, Port I, and a subset Port C. The 44-pin packages have a complete set of ports, including all bits of Port L, Port G, Port D, Port I, and Port C. Refer to Table 14-1 for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888CS/CG/EG/GG/HG/KG device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate functions of the port pins.

Port C, a general-purpose bidirectional I/O port, is present in COP888CS/CG/EG/GG/HG/KG 44-pin packages and partially available (pins 0-3) in 40-pin packages. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in all COP888CS/CG/EG/GG/HG/KG devices, although the 28-pin package only has pins 0-3 available. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP888CS/CG/EG/GG/HG/KG devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G1 is a dedicated Watchdog output that can be used to reset the device in the event of a Watchdog or Clock Monitor error. This pin can not be used a general purpose input or output, and should be left unconnected if not used to signal an error condition. Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I, a general-purpose dedicated input port, is present in all COP888CS/CG/EG/GG/HG/KG devices, although not all eight pins are available in the 28-pin packages. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. The alternate functions of pins I1 though I6 are the comparator inputs and outputs. Comparator 2 is not available in 28-pin packages because the necessary Port I pins are not present in those packages.

Port L, a general-purpose bidirectional I/O port, is present in all COP888CS/CG/EG/GG/HG/KG devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for several alternate functions: the Multi-Input Wakeup/Interrupt inputs, the UART, Timer T2 (if present) and/or Timer T3 (if present). The port inputs have Schmitt triggers to support the Multi- Input Wakeup/Interrupt feature.

14.6 PROGRAM MEMORY

The COP888CS/CG/EG/GG/HG/KG contains 4K (CS/CG), 8K (EG), 16K (GG), 20K (HG) or 24K (KG) bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space starting at 0000 and ending at 0FFF, 1FFF, 3FFF, 4FFF or 5FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs, as described later in this chapter.

14.7 DATA MEMORY

The COP888CS/CG, EG, GG/HG and KG devices contain 192, 256, 512 and 1024 bytes respectively of static RAM used for data storage. The first 128 bytes of RAM occupy two areas in the lowest 256-byte data memory space: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next 256-byte pages of data memory space, starting at 0x00 Hex and ending at 0x3F or 0x7F Hex. This higher part of RAM is accessed by writing the value 0x (x = 1 thru 7) to the S register, as explained in “Data Segment Extension” on page 6 of Chapter 2.

14.8 REGISTER BIT MAPS

The COP888CS/CG/EG/GG/HG/KG has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 14-2 through 14-13 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), Watchdog and Clock Monitor (Chapter 8), Multi-Input Wakeup/ Interrupt and Timer T0 (Chapter 6), UART (Section 14.13), or Comparator (Section 14.12).

Table 14-2 T3CNTRL, Timer T3 Control Register (Address xxB6)*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T3C3	T3C2	T3C1	T3C0	T3PNDA	T3ENA	T3PNDB	T3ENB
T3C3-T3C2-T3C1-T3C0: Timer T3 control bits							
T3PNDA: Timer T3 interrupt A pending flag							
T3ENA: Timer T3 interrupt A enable bit							
T3PNDB: Timer T3 interrupt B pending flag							
T3ENB: Timer T3 interrupt B enable bit							
*COP888CG and COP888EG/GG/HG/KG only.							

Table 14-3 CMPSL, Comparator Select Register (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	CMP2OE*	CMP2RD*	CMP2EN*	CMP1OE	CMP1RD	CMP1EN	Unused

CMP2OE: Comparator 2 output enable bit (Pin I6)

CMP2RD: Comparator 2 read-only result bit

CMP2EN: Comparator 2 enable bit (Pins I4 and I5)

CMP1OE: Comparator 1 output enable bit (Pin I3)

CMP1RD: Comparator 1 read-only result bit

CMP1EN: Comparator 1 enable bit (Pins I1 and I2)

* COP888CG and COP888EG/GG/HG/KG only. Reserved on COP888CS.

Table 14-4 ENU, UART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN: Parity enable bit

PSEL1: Parity selection bit (with PSEL0)

XBIT9/PSEL0: Transmit bit 9 / Parity selection bit (with PSEL1)

CHL1-CHL0: Frame format selection bits (number of data bits)

ERR: Error flag (data overrun, framing, or parity error)

RBFL: Receive buffer full flag

TBMT: Transmit buffer empty flag

Table 14-5 ENUR, UART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	Reserved	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag
FE:	Framing Error flag
PE:	Parity Error flag
RBIT9:	Receive bit 9 (when 9 bits per frame)
ATTN:	Attention mode enable bit
XMTG:	Transmitting; reset at end of last frame
RCVG:	Receiving; reset when RDX goes high

Table 14-6 ENUI, UART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	STP78	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2:	Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)
STP78:	Last Stop bit length (0 = 1 bit, 1 = 7/8 bit)
ETDX:	Enable TDX transmit pin
SSEL:	Synchronous select bit (0 = asynchronous, 1 = synchronous)
XRCLK:	External Receive Clock (0 = internal, 1 = external clock)
XTCLK:	External Transmit Clock (0 = internal, 1 = external clock)
ERI:	Enable Receive Interrupt
ETI:	Enable Transmit Interrupt

Table 14-7 BAUD, UART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0

BRD7–BRD0: Baud rate divisor (with PSR register)

Table 14-8 PSR, UART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8
PSR4–PSR0:	Baud rate prescaler						
BRD10–BRD8:	Baud rate divisor (with BAUD register)						

Table 14-9 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0:	Timer T2 control bits						
T2PNDA:	Timer T2 interrupt A pending flag						
T2ENA:	Timer T2 interrupt A enable bit						
T2PNDB:	Timer T2 interrupt B pending flag						
T2ENB:	Timer T2 interrupt B enable bit						

Table 14-10 WDSVR, Watchdog Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN
WS1–WS0:	Watchdog Window Select bits						
KEY4–KEY0:	Watchdog Key Data (01100)						
CMEN:	Clock Monitor Enable bit						

Table 14-11 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN:	Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)
T0PND:	Timer T0 (IDLE Timer) interrupt pending flag
T0EN:	Timer T0 (IDLE Timer) interrupt enable bit
µWPND:	MICROWIRE interrupt pending flag
µWEN:	MICROWIRE interrupt enable bit
T1PNDB:	Timer T1 interrupt B pending flag
T1ENB:	Timer T1 interrupt B enable bit

Table 14-12 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits
MSEL:	MICROWIRE Select bit
IEDG:	External Interrupt Edge selection bit
SL1–SL0:	MICROWIRE clock divide-by selection bits

Table 14-13 CNTRL, Control Regoister (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC:	Half-Carry bit
C:	Carry bit
T1PNDA:	Timer T1 interrupt A pending flag
T1ENA:	Timer T1 interrupt A enable bit
EXPND:	External interrupt pending flag
BUSY:	MICROWIRE Busy flag
EXEN:	External interrupt enable bit
GIE:	Global Interrupt Enable

14.9 MEMORY MAP

Table 14-14 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888CS/CG/EG/GG/HG/KG registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated "Reserved" in the table. The COP888CS does not have Timer T2 or T3, so the register addresses associated with those timers are not used, and should be considered "Reserved."

Table 14-14 COP888CS/CG/EG/GG/HG/KG Data Memory Map

ADDRESS	CONTENTS
0000–006F	RAM (112 bytes general-purpose data memory)
0070–007F	Unused address space
xx80–xxAF	Unused address space
xxB0	Timer T3 lower byte
xxB1	Timer T3 upper byte
xxB2	Timer T3 autoload register T3RA lower byte
xxB3	Timer T3 autoload register T3RA upper byte
xxB4	Timer T3 autoload register T3RB lower byte
xxB5	Timer T3 autoload register T3RB upper byte
xxB6	Timer T3 control register, T3CNTRL
xxB7	Comparator select register, CMPSL
xxB8	UART transmit buffer, TBUF
xxB9	UART receive buffer, RBUF
xxBA	UART control and status register, ENU
xxBB	UART receive control and status register, ENUR
xxBC	UART interrupt and clock source register, ENUI
xxBD	UART baud register, BAUD
xxBE	UART prescaler select register, PSR
xxBF	Reserved
xxC0	Timer T2 lower byte
xxC1	Timer T2 upper byte
xxC2	Timer T2 autoload register T2RA lower byte
xxC3	Timer T2 autoload register T2RA upper byte

Table 14-14 COP888CS/CG/EG/GG/HG/KG Data Memory Map (Continued)

ADDRESS	CONTENTS
xxC4	Timer T2 autoload register T2RB lower byte
xxC5	Timer T2 autoload register T2RB upper byte
xxC6	Timer T2 control register, T2CNTRL
xxC7	Watchdog service register, WDSVR
xxC8	Multi-Input Wakeup edge select register, WKEDG
xxC9	Multi-Input Wakeup enable register, WKEN
xxCA	Multi-Input Wakeup interrupt pending register, WKPND
xxCB–xxCF	Reserved
xxD0	Port L data register, PORTLD
xxD1	Port L configuration register, PORTLC
xxD2	Port L input pins (read only), PORTCP
xxD3	Reserved
xxD4	Port G data register, PORTGD
xxD5	Port G configuration register, PORTGC
xxD6	Port G input pins (read only), PORTGP
xxD7	Port I input pins (read only), PORTI
xxD8	Port C data register, PORTCD
xxD9	Port C configuration register, PORTCC
xxDA	Port C input pins (read only), PORTCP
xxDB	Reserved
xxDC	Port D data register, PORTD
xxDD–xxDF	Reserved
xxE0–xxE5	Reserved
xxE6	Timer T1 autoload register T1RB lower byte
xxE7	Timer T1 autoload register T1RB upper byte
xxE8	Interrupt control register, ICNTRL
xxE9	MICROWIRE serial I/O shift register, SIO
xxEA	Timer T1 lower byte
xxEB	Timer T1 upper byte

Table 14-14 COP888CS/CG/EG/GG/HG/KG Data Memory Map (Continued)

ADDRESS	CONTENTS
xxEC	Timer T1 autoload register T1RA lower byte
xxED	Timer T1 autoload register T1RA upper byte
xxEE	Control register, CNTRL
xxEF	Processor Status Word register, PSW
xxF0–xxFB	General-purpose memory-mapped registers
xxFC	X pointer register
xxFD	SP (Stack Pointer) register
xxFE	B pointer register
xxFF	S (data segment extension) register
0100–013F	RAM (COP888CS/CG, additional 64 bytes)
0100–017F	RAM (COP888EG/GG/HG/KG, additional 128 bytes)
0200–027F	RAM (COP888GG/HG/KG, additional 128 bytes)
0300–037F	RAM (COP888GG/HG/KG, additional 128 bytes)
0400–047F	RAM (COP888KG, additional 128 bytes)
0500–057F	RAM (COP888KG, additional 128 bytes)
0600–067F	RAM (COP888KG, additional 128 bytes)
0700–077F	RAM (COP888KG, additional 128 bytes)

The COP888CS/CG/EG/GG/HG/KG contains general-purpose RAM at addresses above 00FF Hex. The S register is used for designating the high-order byte of the RAM address. Registers and memory residing in the top half of the 256-byte memory space (80-FF Hex) can always be accessed, regardless of the value in the S register. For more information on this subject, see “Data Segment Extension” on page 6.

14.10 RESET

Upon reset of the COP888CS/CG/EG/GG/HG/KG, the ports and registers are initialized as follows:

Port C data register, PORTCD: 00

Port C configuration register, PORTCC: 00

Port D data register, PORTD: FF

Port G data register, PORTGD: 00

Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timer T2 control register, T2CNTRL:	00
Timer T3 control register, T3CNTRL:	00
Timer T1, T2 and T3 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator and Timers T1, T2, T3:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
S register:	00
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
Watchdog service register, WDSVR:	D9
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
Comparator select register, CMPSL:	00

UART control and status register, ENU:	01
UART receive control and status register, ENUR:	00
UART interrupt and clock source register, ENUI:	00
UART baud register, BAUD:	Upon power-up reset, unknown. Upon external reset, unchanged.
UART prescaler select register, PSR:	00

14.11 INTERRUPTS

Table 14-15 shows the types of interrupts in the COP888CS/CG/EG/GG/HG/KG, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. The COP888CS does not have Timer T2 or T3, so the interrupts associated with those timers should be considered “Reserved.” For basic information on COP888 interrupts, see Chapter 3.

14.12 COMPARATOR

Each COP888CS/CG/EG/GG/HG/KG device has at least one on-chip comparator. A comparator is a circuit that compares the analog input voltages applied to two input pins, and determines which voltage is higher. A memory mapped-register is used for controlling the comparator and for reading the results. The results can also be placed on a device output, if desired.

Every COP888CS/CG/EG/GG/HG/KG device has one comparator, designated Comparator 1. Each COP888CS/CG/EG/GG/HG/KG device supplied in a 40-pin or 44-pin package has an additional comparator, designated Comparator 2.

Two input pins are associated with each comparator, called the negative and positive inputs. If the voltage on the positive input is higher than the voltage on the negative pin, then the output result is “true” or logic 1. If the voltage on the negative input is higher, then the result is “false” or logic 0. The current result of a comparison is written to the read-only result bit of the comparator select register. If the comparator's output pin is enabled, the current result of the comparison is placed on an output pin as a logic 1 or 0, making the comparison result available to the external circuit.

Once enabled, the comparator operates continuously until stopped by the software. The response time is very fast, typically less than the instruction cycle time. In order to operate properly, the input voltages must be within a specific range, slightly smaller than the range from GND to V_{CC}. See the device data sheet for specific values.

The comparator inputs and outputs are alternate functions of Port I. If the comparator is enabled, the two Port I pins that serve as the comparator inputs will operate in that capacity rather than as general-purpose port pins. If the comparator output is enabled along with the comparator, a third Port I pin operates as an output, indicating the

Table 14-15 COP888CS/CG/EG/GG/HG/KG Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	UART Receive	01EE–01EF
10	UART Transmit	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	Timer T3A/Underflow	01E6–01E7
14	Timer T3B	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

comparison result as a logic 1 (high) or logic 0 (low). The Port I pins and their alternate functions are listed in Table 14-16.

Table 14-16 Port I Alternate Functions

Port I Pin	Alternate Function
I0	None
I1	COMP1IN- (Comparator 1 negative input)
I2	COMP1IN+ (Comparator 1 positive input)
I3	COMP1OUT (Comparator 1 output)

Table 14-16 Port I Alternate Functions

Port I Pin	Alternate Function
I4	COMP2IN- (Comparator 2 negative input)
I5	COMP2IN+ (Comparator 2 positive input)
I6	COMP2OUT (Comparator 2 output)
I7	None

The comparators are controlled by a single memory-mapped register called the CMPSL (Comparator Select) register, located at address xxB7 Hex. Three bits in the register are used for each comparator: a comparator enable bit, a comparator-output enable bit, and a read-only result bit. Upon reset, all bits are cleared and the comparators are disabled. Setting the comparator enable bit enables operation of the comparator and its two inputs. Setting the comparator-output enable bit causes the results of the comparison to be placed on the output pin. Whether or not the output pin is enabled, the comparison results can be read from the result bit, which is a read-only bit.

The CMPSL register bit map is shown in Table 14-17.

Table 14-17 CMPSL, Comparator Select Register (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	CMP2OE	CMP2RD	CMP2EN	CMP1OE	CMP1RD	CMP1EN	Unused

CMP2OE: Comparator 2 output enable bit (Pin I6)

CMP2RD: Comparator 2 read-only result bit

CMP2EN: Comparator 2 enable bit (Pins I4 and I5)

CMP1OE: Comparator 1 output enable bit (Pin I3)

CMP1RD: Comparator 1 read-only result bit

CMP1EN: Comparator 1 enable bit (Pins I1 and I2)

If a comparator is not enabled, the corresponding result bit is read as 0. When writing to the CMPSL register, it does not matter what bit values are written to the two read-only result bit positions. The two “unused” bits can serve as read/write software flags, if desired.

To reduce power consumption, the comparators should be disabled prior to entering the HALT or IDLE mode.

14.13 UART

Each COP888CS/CG/EG/GG/HG/KG device has an on-chip Universal Asynchronous Receiver Transmitter (UART), which can be used for transmitting and receiving data

serially, either synchronously or asynchronously. This full-duplex, double-buffered UART is fully programmable, and can be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (7/8, 1, 1-7/8, or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)
- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability
- Error detection circuitry and diagnostic self-test capability

Information on the UART is provided in <tag-Chapter 9>.

14.14 MASK OPTIONS

The COP888CS/CG/EG/GG/HG/KG has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/IO)
G7 (CK) is the clock generator output to a crystal/resonator
CKI is the clock input

=2 Single pin RC controlled oscillator (CKI/IO)
G7 is available as a HALT/Restart or general-purpose input

Option 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Option 3 differs for the COP888CS and COP888CG/EG/GG/HG/KG as noted below.

Option 3: COP888CS Bonding

=1 44-pin PLCC

=2 40-pin DIP

=3 NA

=4 28-pin DIP

=5 28-pin SO

Option 3: COP888CG/EG/GG/HG/KG Bonding

=1 44-pin PLCC

=2 40-pin DIP

=3 NA

=4 28-pin DIP/SO (available for COP888CG and EG only)

Chapter 15

COP888CF

15.1 INTRODUCTION

The COP888CF is a member of the COP888 Feature Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications. The COP888CF has an on-chip, multi-channel Analog-to-Digital (A/D) converter. The A/D Converter converts an analog voltage into a 8-bit digital value, which can then be read by the software from a memory-mapped register. The input voltage range is defined by two reference voltages provided to the device, AGND and V_{REF}.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888CF.

- Program Memory: Mask-programmed ROM, 4K bytes
- Data Memory: Static RAM, 128 bytes
- S-Register Data Memory Extension: No
- 16-Bit Programmable Timers: Timers T1 and T2
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- WATCHDOG and Clock Monitor: Yes
- Additional feature: A/D Converter

Several versions of the COP888CF are available that operate over different voltage and temperature ranges. Refer to the COP888CF data sheet for more specific information.

This chapter describes the device-specific features of the COP888CF. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

15.2 BLOCK DIAGRAM

Figure 15-1 is a block diagram showing the basic functional blocks of the COP888CF. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

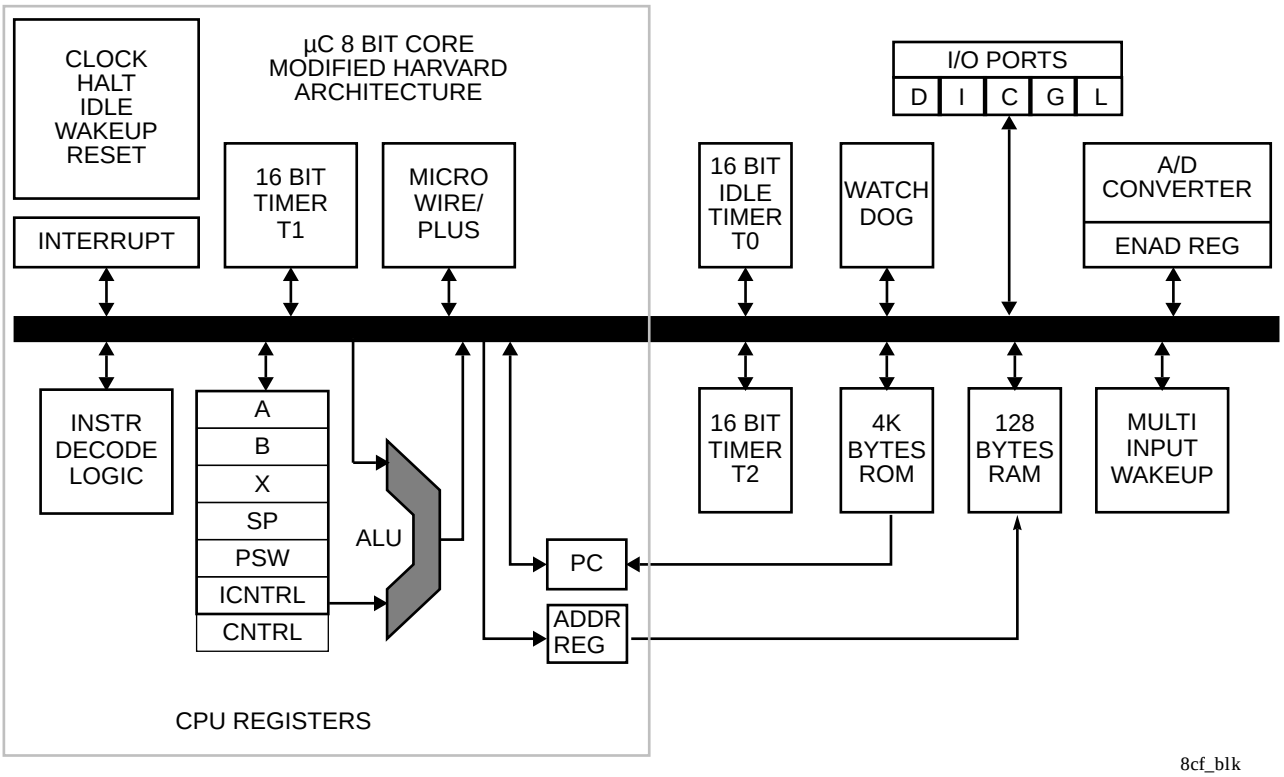


Figure 15-1 COP888CF Block Diagram

15.3 DEVICE PINOUTS/PACKAGES

The COP888CF is available in 28-pin DIP, 28-pin SO, 40-pin DIP and 44-pin PLCC packages. Figure 15-2 shows the COP888CF device package pinouts.

Refer to the COP888CF data sheet for more detailed package information.

15.4 PIN DESCRIPTIONS

The COP888CF has six dedicated function pins: V_{CC}, GND, CKI, $\overline{\text{RESET}}$, AGND, and V_{REF}. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. AGND and V_{REF} are the lower and upper reference voltages that define the input range for the A/D Converter. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 15-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.

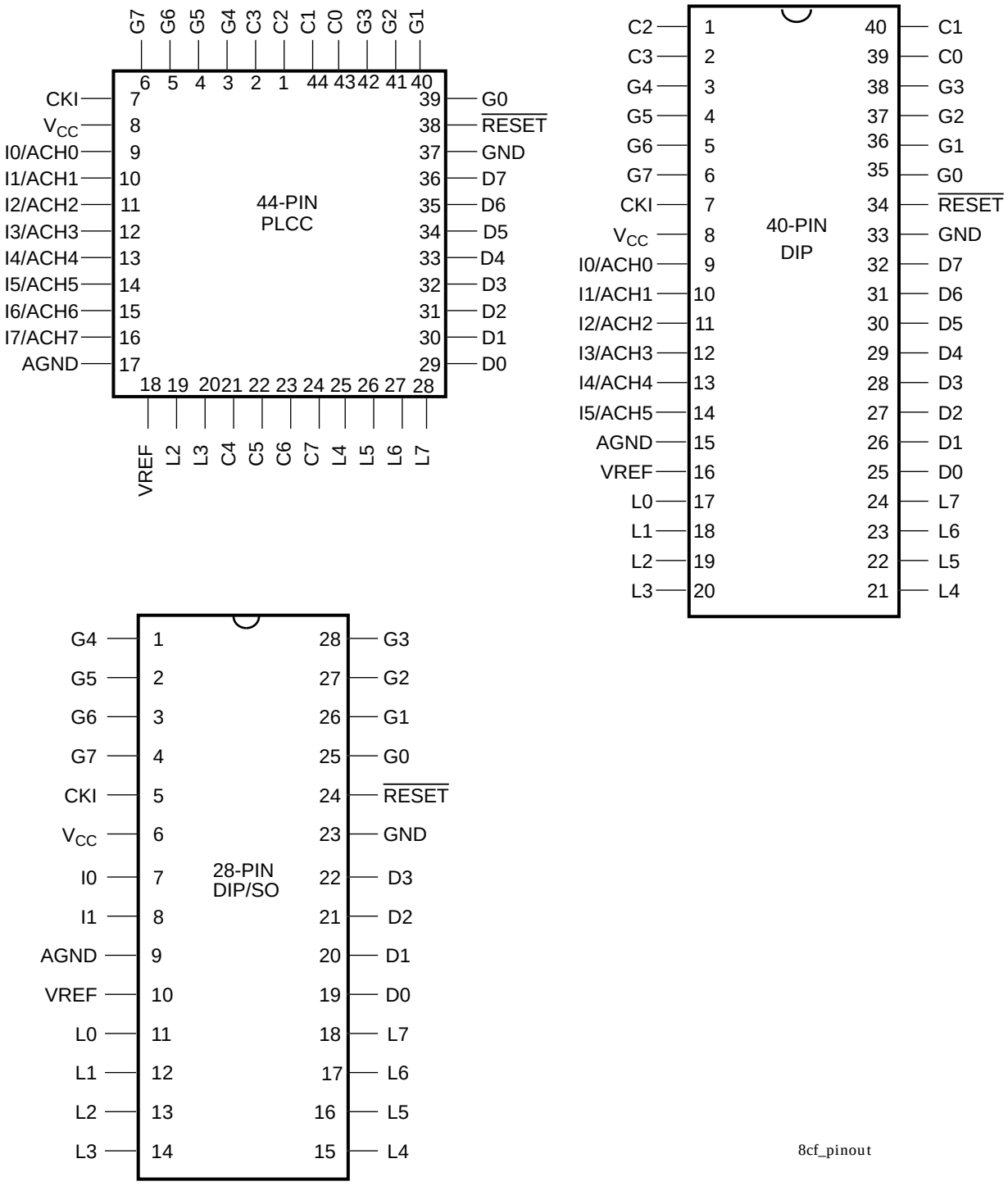


Figure 15-2 Device Package Pinouts

Table 15-1 COP888CF Pinouts

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
L0	I/O	MIWU	11	17	—
L1	I/O	MIWU	12	18	—
L2	I/O	MIWU	13	19	19
L3	I/O	MIWU	14	20	20
L4	I/O	MIWU or T2A	15	21	25
L5	I/O	MIWU or T2B	16	22	26
L6	I/O	MIWU	17	23	27
L7	I/O	MIWU	18	24	28
G0	I/O	INT	25	35	39
G1	WDOUT		26	36	40
G2	I/O	T1B	27	37	41
G3	I/O	T1A	28	38	42
G4	I/O	SO	1	3	3
G5	I/O	SK	2	4	4
G6	I	SI	3	5	5
G7	I/CKO	HALT RESTART	4	6	6
D0	O		19	25	29
D1	O		20	26	30
D2	O		21	27	31
D3	O		22	28	32
I0	I	ACH0	7	9	9
I1	I	ACH1	8	10	10
I2	I	ACH2		11	11
I3	I	ACH3		12	12
I4	I	ACH4		13	13
I5	I	ACH5		14	14
I6	I	ACH6			15
I7	I	ACH7			16

Table 15-1 COP888CF Pinouts (Continued)

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
D4	O			29	33
D5	O			30	34
D6	O			31	35
D7	O			32	36
C0	I/O			39	43
C1	I/O			40	44
C2	I/O			1	1
C3	I/O			2	2
C4	I/O				21
C5	I/O				22
C6	I/O				23
C7	I/O				24
V_{REF}	$+V_{REF}$		10	16	18
AGND	AGND		9	15	17
V_{CC}			6	8	8
GND			23	33	37
CKI			5	7	7
$\overline{\text{RESET}}$			24	34	38

15.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888CF depend on the package type. The 28-pin packages have Port L, Port G, and a subset of Port D and Port I. The 40-pin packages have Port L, Port G, Port D, and a subset of Port I and Port C. The 44-pin packages have a complete set of ports, including all bits of Port L, Port G, Port D, Port I, and Port C. Refer to Table 15-1, COP888CF Pinouts, for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888CF device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate function of the port pin.

Port C, a general-purpose bidirectional I/O port, is present in COP888CF 44-pin packages and partially available (pins 0-3) in 40-pin packages. There are three memory locations

associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in the 40-pin and 44-pin COP888CF devices. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP888CF devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G1 is a dedicated WATCHDOG output that can be used to reset the device in the event of a WATCHDOG or Clock Monitor error. This pin can not be used a general purpose input or output, and should be left unconnected if not used to signal an error condition. Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I, a general-purpose dedicated input port, is present in all COP888CF devices, although not all eight pins are available in the 28-pin and 40-pin packages. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. The alternate functions of the Port I pins are the A/D Converter inputs.

Port L, a general-purpose bidirectional I/O port, is present in all COP888CF devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for Multi-Input Wakeup/Interrupt, an alternate function. In addition, L4 and L5 can be used for Timer T2 I/O. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/Interrupt feature.

15.6 PROGRAM MEMORY

The COP888CF contains 4K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space from 0000 to 0FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs, as described later in this chapter.

15.7 DATA MEMORY

The COP888CF contains 128 bytes of static RAM memory, used for temporary data storage. The RAM occupies two segments in data memory: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers.

15.8 REGISTER BIT MAPS

The COP888CF has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 15-2 through 15-7 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), WATCHDOG and Clock Monitor (Chapter 8), Multi-Input Wakeup/ Interrupt and Timer T0 (Chapter 6), or Analog-to-Digital Converter (Section 15.12).

Table 15-2 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0:			Timer T2 control bits				
T2PNDA:			Timer T2 interrupt A pending flag				
T2ENA:			Timer T2 interrupt A enable bit				
T2PNDB:			Timer T2 interrupt B pending flag				
T2ENB:			Timer T2 interrupt B enable bit				

Table 15-3 WDSVR, WATCHDOG Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN

WS1–WS0: WATCHDOG Window Select bits

KEY4–KEY0: WATCHDOG Key Data (01100)

CMEN: Clock Monitor Enable bit

Table 15-4 ENAD, A/D Converter Control Register (Address xxCB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CHS2	CHS1	CHS0	DIFF	CONT	PRSC2	PRSC1	PRSC0

CHS2–CHS0: Channel Selection bits

DIFF: Differential Mode (0 = single-ended, 1 = differential)

CONT: Continuous Mode (0 = single conversion, 1 = continuous)

PRSC2–PRSC0: A/D clock prescaler selection/enable bits

Table 15-5 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN: Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)

T0PND: Timer T0 (IDLE Timer) interrupt pending flag

T0EN: Timer T0 (IDLE Timer) interrupt enable bit

µWPND: MICROWIRE interrupt pending flag

µWEN: MICROWIRE interrupt enable bit

T1PNDB: Timer T1 interrupt B pending flag

T1ENB: Timer T1 interrupt B enable bit

Table 15-6 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0: Timer T1 control bits

MSEL: MICROWIRE Select bit

IEDG: External Interrupt Edge selection bit

SL1–SL0: MICROWIRE clock divide-by selection bits

Table 15-7 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC: Half-Carry bit

C: Carry bit

T1PNDA: Timer T1 interrupt A pending flag

T1ENA: Timer T1 interrupt A enable bit

EXPND: External interrupt pending flag

BUSY: MICROWIRE Busy flag

EXEN: External interrupt enable bit

GIE: Global Interrupt Enable

15.9 MEMORY MAP

Table 15-8 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888CF registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 15-8 COP888CF Data Memory Map

Address	Contents
00–6F	RAM (112 bytes general-purpose data memory)
70–7F	Unused address space

Table 15-8 COP888CF Data Memory Map (Continued)

Address	Contents
80–AF	Unused address space
B0–BF	Reserved
C0	Timer T2 lower byte
C1	Timer T2 upper byte
C2	Timer T2 autoload register T2RA lower byte
C3	Timer T2 autoload register T2RA upper byte
C4	Timer T2 autoload register T2RB lower byte
C5	Timer T2 autoload register T2RB upper byte
C6	Timer T2 control register, T2CNTRL
C7	WATCHDOG service register, WDSVR
C8	Multi-Input Wakeup edge select register, WKEDG
C9	Multi-Input Wakeup enable register, WKEN
CA	Multi-Input Wakeup interrupt pending register, WKPND
CB	A/D Converter control register, ENAD
CC	A/D Converter result register, ADRSLT
CD–CF	Reserved
D0	Port L data register, PORTLD
D1	Port L configuration register, PORTLC
D2	Port L input pins (read only), PORTLP
D3	Reserved
D4	Port G data register, PORTGD
D5	Port G configuration register, PORTGC
D6	Port G input pins (read only), PORTGP
D7	Port I input pins (read only), PORTI
D8	Port C data register, PORTCD
D9	Port C configuration register, PORTCC
DA	Port C input pins (read only), PORTCP
DB	Reserved
DC	Port D data register, PORTD

Table 15-8 COP888CF Data Memory Map (Continued)

Address	Contents
DD–DF	Reserved
E0–E5	Reserved
E6	Timer T1 autoload register T1RB lower byte
E7	Timer T1 autoload register T1RB upper byte
E8	Interrupt control register, ICNTRL
E9	MICROWIRE serial I/O shift register, SIO
EA	Timer T1 lower byte
EB	Timer T1 upper byte
EC	Timer T1 autoload register T1RA lower byte
ED	Timer T1 autoload register T1RA upper byte
EE	Control register, CNTRL
EF	Processor Status Word register, PSW
F0–FB	General-purpose memory-mapped registers
FC	X pointer register
FD	SP (Stack Pointer) register
FE	B pointer register
FF	Reserved

15.10 RESET

Upon reset of the COP888CF, the ports and registers are initialized as follows:

Port C data register, PORTCD:	00
Port C configuration register, PORTCC:	00
Port D data register, PORTD:	FF
Port G data register, PORTGD:	00
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timer T2 control register, T2CNTRL:	00
Timer T1 and T2 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator, A and Timers T1, T2:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown

WATCHDOG service register, WDSVR:	D9
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
A/D Converter control register, ENAD:	00
A/D Converter result register, ADRSLT:	Unknown

15.11 INTERRUPTS

Table 15-9 shows the types of interrupts in the COP888CF, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 15-9 COP888CF Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	(Reserved)	01EE–01EF
10	(Reserved)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	(Reserved)	01E6–01E7
14	(Reserved)	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1

* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.

15.12 ANALOG-TO-DIGITAL CONVERTER

The COP888CF has a multi-channel, multiplexed-input Analog-to-Digital (A/D) Converter. The A/D Converter receives an analog voltage signal on an input pin (or a pair of input pins), and converts the analog signal into a 8-bit digital value. The digital value can then be read by the software from a memory-mapped register. The input voltage range is defined by two reference voltages provided to the device, AGND and V_{REF}.

The A/D Converter is useful in applications where the software needs to read a quantity (for example, temperature or speed) from an analog sensor. Up to eight different single-ended or four different differential inputs can be handled by a single COP888CF device through the multiplexed input channels.

15.12.1 A/D Operation

Figure 15-3 is a block diagram showing the structure of the COP888CF A/D Converter. There are two memory-mapped registers in the A/D Converter circuit: the A/D Converter control register (ENAD), used for configuring and enabling the A/D Converter, and the A/D Result register (ADRSLT), a read-only register containing the results of the conversion.

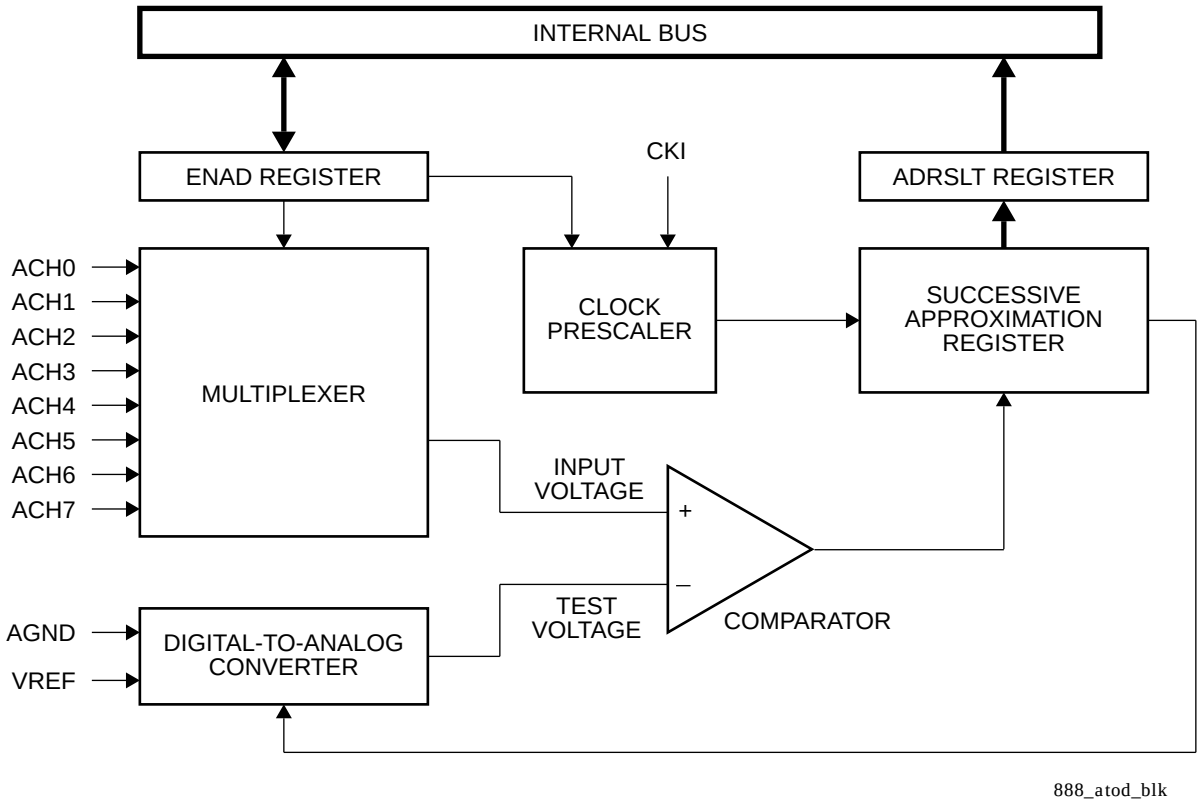


Figure 15-3 COP888CF A/D Converter Block Diagram

The A/D Converter supports ratio-metric measurements. The input voltage range is determined by two fixed reference voltages supplied to the device through pins AGND (analog ground) and V_{REF} (voltage reference). AGND is the lower limit, and V_{REF} is the

upper limit of the input voltage range. The reference voltages are allowed to be set only between certain levels related to GND and V_{CC}, as specified in the device data sheet.

Analog voltages are received on pins ACH0 through ACH7 (Analog Channels 0 through 7), which are alternate functions of the Port I pins. A multiplexer selects one channel for a single-ended conversion, or a pair of channels for differential-pair conversion, based on the value programmed into the ENAD register. The input voltage (or for a differential pair, the difference between the two input voltages) is sent to a comparator. The comparator compares the input voltage with another analog voltage generated by an internal digital-to-analog converter.

The A/D Converter uses successive approximation to determine the analog input voltage. The internal digital-to-analog converter generates a test voltage, which is initially set to the middle of the possible voltage range, halfway between the voltages on AGND and V_{REF}. After the comparison is made, another comparison is made with the test voltage set to the middle of the new possible range (either 1/4 or 3/4 between AGND and V_{REF}). This is repeated until the input voltage is determined to a precision of 1/256 of the full input voltage range (eight iterations).

Each time a comparison is made, a bit is set or cleared in the Successive Approximation Register, depending on whether the input voltage is higher or lower than the test voltage. The bit values in the Successive Approximation Register are determined from most to least significant bit, generating an 8-bit digital value that is proportional to the analog input voltage. The value 00 Hex represents the lowest voltage (AGND), and a value of FF Hex represents the highest voltage (V_{REF}). At the end of the conversion, the 8-bit value is transferred to the ADRSLT register, allowing the final result to be read by the software. The Successive Approximation Register cannot be accessed directly by the software.

The A/D Converter uses its own clock, which is generated by scaling down the CKI clock to a lower frequency. The clock prescaler circuit allows a choice of several different divide-by factors for generating the A/D clock. The choice is made by programming a set of bits in the ENAD register. This feature allows a certain amount of control over the trade-off between speed and accuracy of the A/D Converter without changing the chip clock speed, as explained later in this chapter.

A single conversion takes 12 A/D clock cycles: one for A/D reset, two for sampling the input voltage, eight for conversion, and one for transferring the result to the ADRSLT register. The total conversion time depends on the chip clock speed and the divide-by factor used for generating the A/D clock. If analog voltages on different channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

The A/D Converter circuit is powered down when the device enters the HALT or IDLE mode. If the A/D Converter is running when this happens, the conversion is cancelled, and then started again from the beginning upon exit from the HALT or IDLE mode.

15.12.2 A/D Converter Registers

Two memory-mapped registers are used with the A/D Converter: the ENAD register at address xxCB Hex, and the ADRSLT register at address xxCC Hex. The ENAD register is a read-write memory location used for controlling the operation of the A/D Converter.

The ADRSLT register is a read-only register that contains the most recent A/D conversion result.

The A/D Converter is controlled by writing a byte to the ENAD register. The data byte written to this register enables (or disables) the A/D Converter clock, sets the divide-by factor for generating the clock, selects the operating mode (single-conversion or continuous mode, single-ended or differential inputs), and selects the channel or channel pair that is to receive the analog input. The register bit map for the ENAD register is shown in Table 15-10.

Table 15-10 ENAD, A/D Converter Control Register (Address xxCB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CHS2	CHS1	CHS0	DIFF	CONT	PRSC2	PRSC1	PRSC0
CHS2–CHS0:	Channel Selection bits						
DIFF:	Differential Mode (0 = single-ended, 1 = differential)						
CONT:	Continuous Mode (0 = single conversion, 1 = continuous)						
PRSC2–PRSC0:	A/D clock prescaler selection/enable bits						

The ENAD register is cleared upon reset. This inhibits operation of the A/D clock and disables the A/D converter. To begin using the A/D converter, it is only necessary to write the proper bit values to the ENAD register, as described in detail below.

The ADRSLT register contents are unknown following a reset.

15.12.3 Prescaler Selection

The A/D clock is generated by dividing the CKI clock. The resulting clock period is an integer multiple of the CKI clock. The divide-by factor can be set to 1, 2, 4, 6, 8, 12, or 16. However, the resulting A/D clock frequency must be between 100 KHz and 1.67 MHz for proper operation of the A/D circuit.

With a CKI clock running at 10 MHz (1 MHz instruction cycle), a divide-by factor of 6 results in an A/D clock frequency of 1.67 MHz, the maximum allowed A/D clock speed. Therefore, with the CKI clock running at 10 MHz, the selected divide-by factor must be 6, 8, 12, or 16. A lower divide-by factor can be used only if the CKI clock is running slower than 10 MHz. In any case, the resulting clock frequency must be between 100 KHz and 1.67 MHz.

The prescaler divide-by factor is programmed by writing bits 2, 1, and 0 in the ENAD register. Table 15-11 shows the binary values used for programming this part of the register, and the resulting A/D clock for each binary value.

Table 15-11 A/D Prescaler Options

ENAD Bits 2-1-0	A/D Clock
000	Inhibit A/D clock
001	Divide CKI by 1
010	Divide CKI by 2
011	Divide CKI by 4
100	Divide CKI by 6
101	Divide CKI by 12
110	Divide CKI by 8
111	Divide CKI by 16

If the value 000 is written, the A/D clock is stopped immediately. This can be done to reduce power consumption when the A/D converter is no longer needed.

15.12.4 Single Conversion or Continuous Mode

The A/D Converter can be programmed to perform a single conversion, or to perform conversions continuously using the same single-ended or differential-pair input. The single-conversion mode can be used to obtain a single analog value, or used sequentially to obtain values from different channels. The continuous mode can be used to monitor a single channel or one differential-pair input over a period of time. The operating mode is controlled by bit 3 of the ENAD register: 0 for the single-conversion mode, or 1 for the continuous mode.

In the single-conversion mode, the conversion is started as soon as the ENAD register is written with the proper value. After 12 A/D clock cycles, the result is transferred to the ADRSLT register. The software can read the result any time after that.

In the continuous mode, the conversion is performed in the same manner as the single-conversion mode, but conversions are repeated continuously using the same analog input. A new result is transferred to the ADRSLT register every 12 A/D clock cycles, overwriting the previous result. The software can read the most recent result from the ADRSLT register at any time. Conversions are performed repeatedly until the software writes a new control byte to the ADRSLT register.

15.12.5 Channel Selection

Analog voltages are received on pins ACH0 through ACH7, which are alternate functions of Port I pins I0 through I7. Depending on the package type, some COP888CF devices do not have a complete set of eight Port I pins, and therefore do not have all eight analog channels available.

The A/D Converter uses one channel for a single-ended conversion, or a pair of channels for differential-pair conversion. The single-ended or differential mode is selected with bit 4 of the ENAD register: 0 for single-ended, or 1 for differential. The channel number (or set of two channel numbers) is selected with bits 7, 6, and 5 of the same register, as indicated Table 15-12.

Table 15-12 A/D Channel Selection

ENAD Bits 7-6-5	Single-Ended Mode	Differential Pair Mode (+, -)
000	ACH0	ACH0, ACH1
001	ACH1	ACH1, ACH0
010	ACH2	ACH2, ACH3
011	ACH3	ACH3, ACH2
100	ACH4	ACH4, ACH5
101	ACH5	ACH5, ACH4
110	ACH6	ACH6, ACH7
111	ACH7	ACH7, ACH6

15.12.6 Multi-Channel Conversion

The A/D Converter allows the use of up to eight single-ended or four differential-pair inputs. However, there is only one A/D converter, with only one result register. If multiple channels are to be monitored simultaneously, they must be time-multiplexed by the application software.

Figure 15-4 shows an example of an assembly language program that performs this function. The program makes a request for an analog-to- digital conversion from each of eight single-ended channels in sequence, reads the eight result bytes, and writes the results into a block of data memory. Operation of the program is explained below.

```
                          ;A/D, convert 8 channels      Bytes/Cycles
ADC8:    RC               ;Reset carry bit                  1/1
         LD   ENAD,#4     ;Initiate ACH0 conversion         3/3
         LD   B,#ENAD     ;B pointer to ENAD register       2/2
         LD   X,#DEST     ;X pointer to result destination  2/3
LOOP:    LD   A,[B]       ;Get previous ENAD contents       1/1
         ADC  A,#020      ;Change to next input channel     2/2
         X    A,[B+]      ;Request next conversion          1/2
         LD   A,[B-]      ;Get previous result from ADRSLT  1/2
         X    A,[X+]      ;Store result and incr. pointer   1/3
         IFNC             ;Test ADC instr. for overflow     1/1
         JP   LOOP        ;Loop back for next channel       1/3
```

Figure 15-4 A/D Conversion Routine

The program was written with execution speed as the primary consideration. The comment text in the program listing shows the purpose of each instruction, the number of bytes of program memory used by the instruction, and the number of instruction clock cycles required to execute the instruction.

At the start of the program, the first conversion is initiated, specifying channel ACH0 with a clock divide-by factor of 6. Both ENAD and ADRSLT will be accessed with the B pointer, so the B register is initialized to ENAD. The results will be stored in a block of data memory occupying eight bytes starting at DEST. The data memory will be accessed with the X pointer, so the X register is initialized to DEST.

The program loop first reads the ENAD register, changes the control byte to specify the next single-ended channel (ACH1 the first time through the loop), and uses the X (exchange) instruction to request the next conversion. The X instruction also increments the B register, setting up the next instruction to access the ADRSLT register. (The ADRSLT register immediately follows the ENAD register in data memory.) The result byte from the previous conversion is loaded into the Accumulator from ADRSLT, and at the same time, the B pointer is decremented in preparation for the next pass through the loop. The result byte is written into data memory using the X pointer. The IF instruction tests for overflow from the ADC instruction, which will happen after all eight result bytes have been written (eight passes through the loop).

In each iteration of the loop, the program requests a new conversion and quickly reads the results of the previous conversion. The old result remains in the ADRSLT register until overwritten with the new result, which takes 12 A/D clock cycles, or 48 CKI clock cycles. The loop section of the program takes 14 instruction clock cycles to execute, or 140 CKI clock cycles. Thus, the new result becomes available before it is needed in the next iteration of the loop.

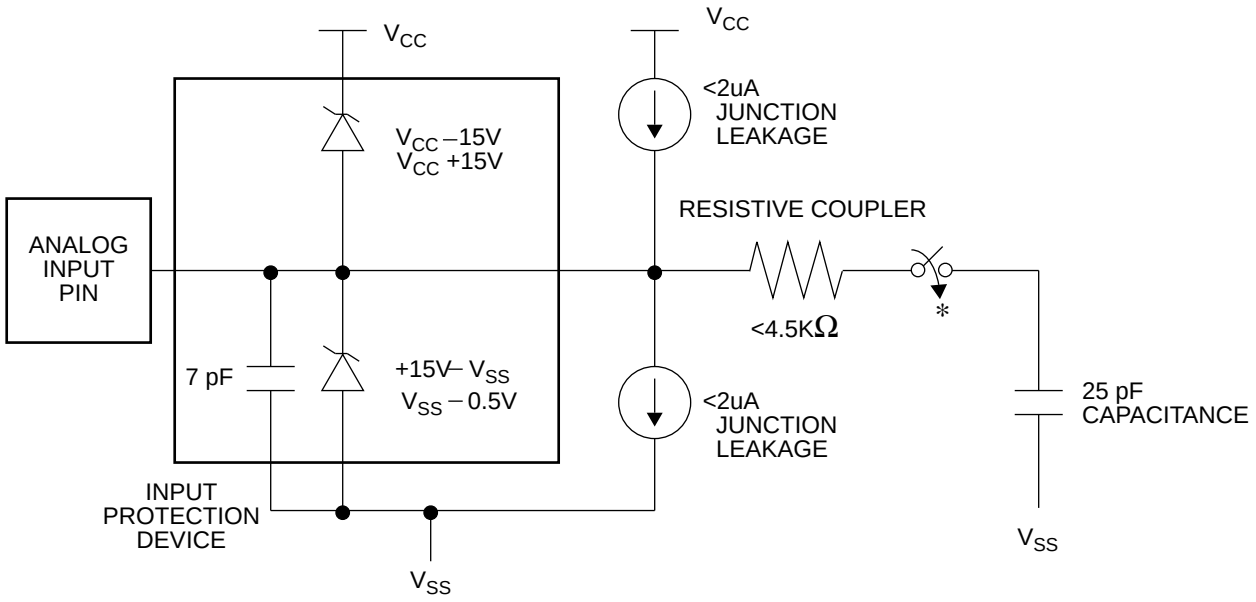
The program code uses 16 bytes of program memory. For execution, the whole routine takes 119 instructions cycles: 9 cycles for initial setup, 14 cycles for the first seven loops, and 12 cycles for the last iteration of the loop (1 cycle to skip over the JP instruction). With a 10 MHz CKI clock, the routine takes 119 microseconds.

15.12.7 Speed, Accuracy, and Hardware Considerations

The maximum allowed A/D clock speed of 1.67 MHz results in a clock period of 600 ns. The time required for one A/D conversion is 12 A/D clock cycles, or 7.2 microseconds. This is the shortest possible conversion time for the COP888CF A/D converter. In some cases, a longer conversion time may be necessary or desirable in order to ensure accurate conversions. This is because it takes some time for the analog input signal to charge the capacitive load on the A/D input.

Figure 15-5 shows the internal operation of an analog input pin in the single-ended mode, together with the capacitive loads and input protection components on the pin. The analog switch is kept open most of the time. For an A/D conversion, the switch is closed for the duration of two A/D clock cycles, and the voltage is sampled in the middle of this period.

The A/D clock period should be made long enough to allow the capacitor to charge up to the full voltage supplied to the analog input pin. Whether the minimum A/D clock period



* The analog switch is closed only during the sample time.

888_atod_input

Figure 15-5 Analog Input Pin Internal Operation

of 600 ns is sufficient depends on the output impedance of the analog source and the amount of inaccuracy that is acceptable for the application. In general, the benefits and costs of using a slower A/D clock should be considered when the analog source impedance exceeds 1 KΩ. As a rule of thumb, you should increase the A/D clock period in proportion to the source impedance. For example, for a source impedance of 2 KΩ, use twice the minimum clock period of 600 ns, or 1200 ns (833 KHz). The A/D clock can be slowed down either by increasing the programmed divide-by factor or by reducing the CKI clock frequency, or a combination of both. The A/D clock frequency should not be reduced below 100 KHz.

Both electrical noise and digital clock coupling to the analog inputs can cause inaccurate conversions. For the most accurate possible results, design the circuit board to minimize noise on these inputs. Keep the leads to the analog input pins as short as possible, and as far away as possible from clock leads.

15.13 MASK OPTIONS

The COP888CF has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/I0)
G7 (CKO) is the clock generator output to a crystal/resonator
CKI is the clock input.

=2 Single-pin RC controlled oscillator
G7 is available as a HALT/Restart and general-purpose input

Option 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Option 3: COP888CF Bonding

=1 44-pin PLCC

=2 40-pin DIP

=3 NA

=4 28-pin DIP

=5 28-pin SO

Chapter 16

COP888CL

16.1 INTRODUCTION

The COP888CL is a member of the COP888 Feature Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications. The COP888CL is a basic member of the family, useful in low-cost applications that do not require additional features or extended data memory.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888CL.

- Program Memory: Mask-programmed ROM, 4K bytes
- Data Memory: Static RAM, 128 bytes
- S-Register Data Memory Extension: No
- 16-Bit Programmable Timers: Timers T1 and T2
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- WATCHDOG and Clock Monitor: Yes

Several versions of the COP888CL are available that operate over different voltage and temperature ranges. Refer to the COP888CL data sheet for more specific information.

This chapter describes the device-specific features of the COP888CL. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

16.2 BLOCK DIAGRAM

Figure 16-1 is a block diagram showing the basic functional blocks of the COP888CL. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

16.3 DEVICE PINOUTS/PACKAGES

The COP888CL is available in 28-pin DIP, 28-pin SO, 40-pin DIP, and 44-pin PLCC packages. Figure 16-2 shows the COP888CL device package pinouts.

Refer to the COP888CL data sheet for more detailed package information.

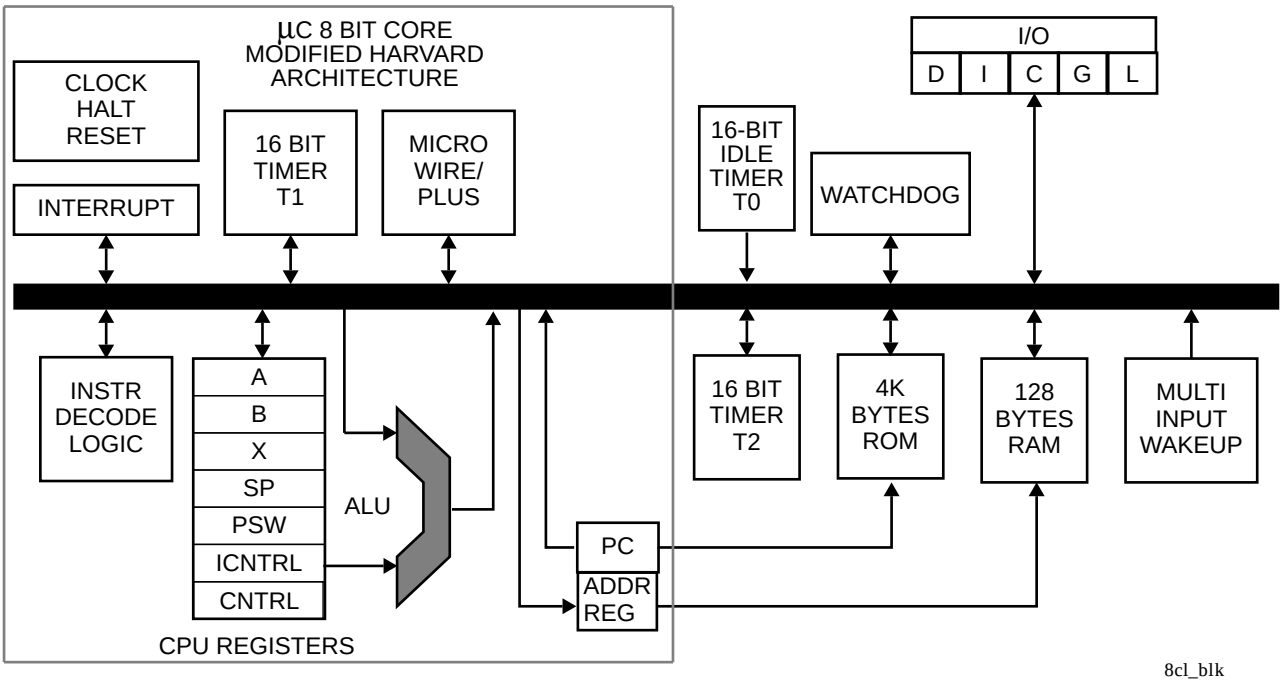


Figure 16-1 COP888CL Block Diagram

16.4 PIN DESCRIPTIONS

The COP888CL has four dedicated function pins: V_{CC}, GND, CKI, and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 16-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.

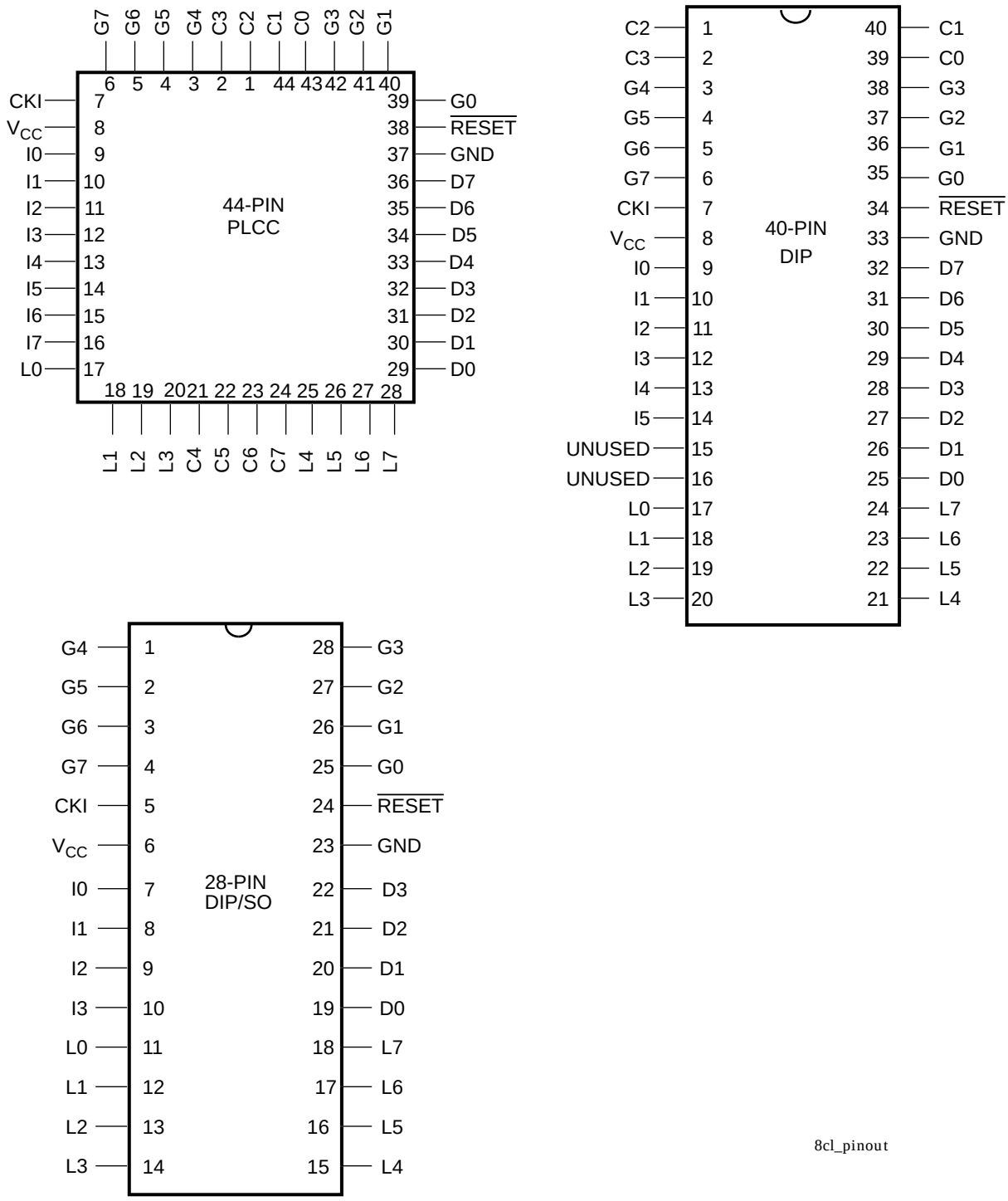


Figure 16-2 Device Package Pinouts

Table 16-1 COP888CL Pinouts

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
L0	I/O	MIWU	11	17	17
L1	I/O	MIWU	12	18	18
L2	I/O	MIWU	13	19	19
L3	I/O	MIWU	14	20	20
L4	I/O	MIWU or T2A	15	21	25
L5	I/O	MIWU or T2B	16	22	26
L6	I/O	MIWU	17	23	27
L7	I/O	MIWU	18	24	28
G0	I/O	INT	25	35	39
G1	WDOUT		26	36	40
G2	I/O	T1B	27	37	41
G3	I/O	T1A	28	38	42
G4	I/O	SO	1	3	3
G5	I/O	SK	2	4	4
G6	I	SI	3	5	5
G7	I/CKO	HALT RESTART	4	6	6
D0	O		19	25	29
D1	O		20	26	30
D2	O		21	27	31
D3	O		22	28	32
I0	I		7	9	9
I1	I		8	10	10
I2	I			11	11
I3	I			12	12
I4	I		9	13	13
I5	I		10	14	14
I6	I				15
I7	I				17

Table 16-1 COP888CL Pinouts (Continued)

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
D4	O			29	33
D5	O			30	34
D6	O			31	35
D7	O			32	36
C0	I/O			39	43
C1	I/O			40	44
C2	I/O			1	1
C3	I/O			2	2
C4	I/O				21
C5	I/O				22
C6	I/O				23
C7	I/O				24
Unused*				16	
Unused				15	
V_{CC}			6	8	8
GND			23	33	37
CKI			5	7	7
$\overline{\text{RESET}}$			24	34	38
* On the 40-pin package Pins 15 and 16 must be connected to GND.					

16.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888CL depend on the package type. The 28-pin packages have Port L, Port G, and a subset of Port D and Port I. The 40-pin packages have Port L, Port G, Port D, and a subset of Port I and Port C. The 44-pin packages have a complete set of ports, including all bits of Port L, Port G, Port D, Port I, and Port C. Refer to Table 16-1 for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888CL device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate function of the port pin.

Port C, a general-purpose bidirectional I/O port, is present in COP888CL 44-pin packages and partially available (pins 0-3) in 40-pin packages. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in all COP888CL devices, although the 28-pin package only has pins 0-3 available. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP888CL devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general-purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G1 is a dedicated WATCHDOG output that can be used to reset the device in the event of a WATCHDOG or Clock Monitor error. This pin can not be used as a general-purpose input or output, and should be left unconnected if not used to signal an error condition. Pin G6 can be used as a general-purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general-purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general-purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I, a general-purpose dedicated input port, is present in all COP888CL devices, although not all eight pins are available in the 28- pin and 40-pin packages. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. There are no alternate functions associated with this port.

Port L, a general-purpose bidirectional I/O port, is present in all COP888CL devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for Multi-Input Wakeup/Interrupt, an alternate function. In addition, L4 and L5 can be used for Timer T2 I/O. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/Interrupt feature.

16.6 PROGRAM MEMORY

The COP888CL contains 4K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space from 0000 to 0FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs, as described later in this chapter.

16.7 DATA MEMORY

The COP888CL contains 128 bytes of static RAM memory, used for temporary data storage. The RAM occupies two segments in data memory: 112 bytes from 00-6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers.

16.8 REGISTER BIT MAPS

The COP888CL has several 8-bit memory-mapped registers used for controlling the CPU, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 16-2 through 16-6 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: Core registers (Chapter 2), Timers (Chapter 4), WATCHDOG and Clock Monitor (Chapter 8), or Multi-Input Wakeup/ Interrupt and Timer T0 (Chapter 6).

Table 16-2 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0:			Timer T2 control bits				
T2PNDA:			Timer T2 interrupt A pending flag				
T2ENA:			Timer T2 interrupt A enable bit				
T2PNDB:			Timer T2 interrupt B pending flag				
T2ENB:			Timer T2 interrupt B enable bit				

Table 16-3 WDSVR, WATCHDOG Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN
WS1–WS0:	WATCHDOG Window Select bits						
KEY4–KEY0:	WATCHDOG Key Data (01100)						
CMEN:	Clock Monitor Enable bit						

Table 16-4 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB
LPEN:	Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)						
T0PND:	Timer T0 (IDLE Timer) interrupt pending flag						
T0EN:	Timer T0 (IDLE Timer) interrupt enable bit						
µWPND:	MICROWIRE interrupt pending flag						
µWEN:	MICROWIRE interrupt enable bit						
T1PNDB:	Timer T1 interrupt B pending flag						
T1ENB:	Timer T1 interrupt B enable bit						

Table 16-5 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0
T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits						
MSEL:	MICROWIRE Select bit						
IEDG:	External Interrupt Edge selection bit						
SL1-SL0:	MICROWIRE clock divide-by selection bits						

Table 16-6 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC: Half-Carry bit

C: Carry bit

T1PNDA: Timer T1 interrupt A pending flag

T1ENA: Timer T1 interrupt A enable bit

EXPND: External interrupt pending flag

BUSY: MICROWIRE Busy flag

EXEN: External interrupt enable bit

GIE: Global Interrupt Enable

16.9 MEMORY MAP

Table 16-7 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888CL registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 16-7 COP888CL Data Memory Map

Address	Contents
00–6F	RAM (112 bytes general-purpose data memory)
70–7F	Unused address space
80–AF	Unused address space
B0–BF	Reserved
C0	Timer T2 lower byte
C1	Timer T2 upper byte
C2	Timer T2 autoload register T2RA lower byte
C3	Timer T2 autoload register T2RA upper byte
C4	Timer T2 autoload register T2RB lower byte
C5	Timer T2 autoload register T2RB upper byte
C6	Timer T2 control register, T2CNTRL

Table 16-7 COP888CL Data Memory Map (Continued)

Address	Contents
C7	WATCHDOG service register, WDSVR
C8	Multi-Input Wakeup edge select register, WKEDG
C9	Multi-Input Wakeup enable register, WKEN
CA	Multi-Input Wakeup interrupt pending register, WKPND
CB–CF	Reserved
D0	Port L data register, PORTLD
D1	Port L configuration register, PORTLC
D2	Port L input pins (read only), PORTLP
D3	Reserved
D4	Port G data register, PORTGD
D5	Port G configuration register, PORTGC
D6	Port G input pins (read only), PORTGP
D7	Port I input pins (read only), PORTI
D8	Port C data register, PORTCD
D9	Port C configuration register, PORTCC
DA	Port C input pins (read only), PORTCP
DB	Reserved
DC	Port D data register, PORTD
DD–DF	Reserved
E0–E5	Reserved
E6	Timer T1 autoload register T1RB lower byte
E7	Timer T1 autoload register T1RB upper byte
E8	Interrupt control register, ICNTRL
E9	MICROWIRE serial I/O shift register, SIO
EA	Timer T1 lower byte
EB	Timer T1 upper byte
EC	Timer T1 autoload register T1RA lower byte
ED	Timer T1 autoload register T1RA upper byte
EE	Control register, CNTRL

Table 16-7 COP888CL Data Memory Map (Continued)

Address	Contents
EF	Processor Status Word register, PSW
F0–FB	General-purpose memory-mapped registers
FC	X pointer register
FD	SP (Stack Pointer) register
FE	B pointer register
FF	Reserved

16.10 RESET

Upon reset of the COP888CL, the ports and registers are initialized as follows:

Port C data register, PORTCD:	00
Port C configuration register, PORTCC:	00
Port D data register, PORTD:	FF
Port G data register, PORTGD:	00
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timer T2 control register, T2CNTRL:	00
Timer T1 and T2 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator A and Timer T1:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option)

Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
WATCHDOG service register, WDSVR:	D9
RAM (other than F0-FC):	Upon power-up reset, unknown. Upon external reset, unchanged.

16.11 INTERRUPTS

Table 16-8 shows the types of interrupts in the COP888CL, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 16-8 COP888CL Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	(Reserved)	01EE–01EF
10	(Reserved)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	(Reserved)	01E6–01E7
14	(Reserved)	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

16.12 MASK OPTIONS

The COP888CL has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/I0)
G7 (CKO) is the clock generator output to a crystal/resonator
CKI is the clock input

=2 Single-end RC controlled oscillator (CKI/10)
G7 is available as a HALT/Restart and general-purpose input

Option 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Option 3: COP888CL Bonding

=1 44-pin PLCC

=2 40-pin DIP

=3 NA

=4 28-pin DIP

=5 28-pin SO

Chapter 17

COP888EB

17.1 INTRODUCTION

The COP888EB is a member of the COP888 Feature Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications. The COP888EB has a combination of several additional features: a Controller Area Network (CAN) Interface, a Universal Asynchronous Receiver/Transmitter (UART), a multi-channel Analog-to-Digital (A/D) converter, a Serial Peripheral Interface (SPI), and reduced electromagnetic radiated emissions achieved by using internal power supply filters, a low-current crystal oscillator, and gradual turn-on of output drivers.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888EB.

- Program Memory: Mask-programmed ROM, 8K bytes
- Data Memory: Static RAM, 192 bytes
- S-Register Data Memory Extension: Yes
- 16-Bit Programmable Timers: Timers T1 and T2
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes (Using Port L, Port M, and CAN Interface)
- WATCHDOG and Clock Monitor: Yes
- Additional features: UART, A/D Converter, CAN Interface, SPI Master/Slave Interface, additional ports (E, F, M, and N), and Reduced Electromagnetic Emissions

Several versions of the COP888EB are available that operate over different voltage and temperature ranges. Refer to the COP888EB data sheet for more specific information.

This chapter describes the device-specific features of the COP888EB. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual. Information on the UART, A/D Converter, and CAN Interface is provided in Chapters 9, 10, and 11.

17.2 BLOCK DIAGRAM

Figure 17-1 is a block diagram showing the basic functional blocks of the COP888EB. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

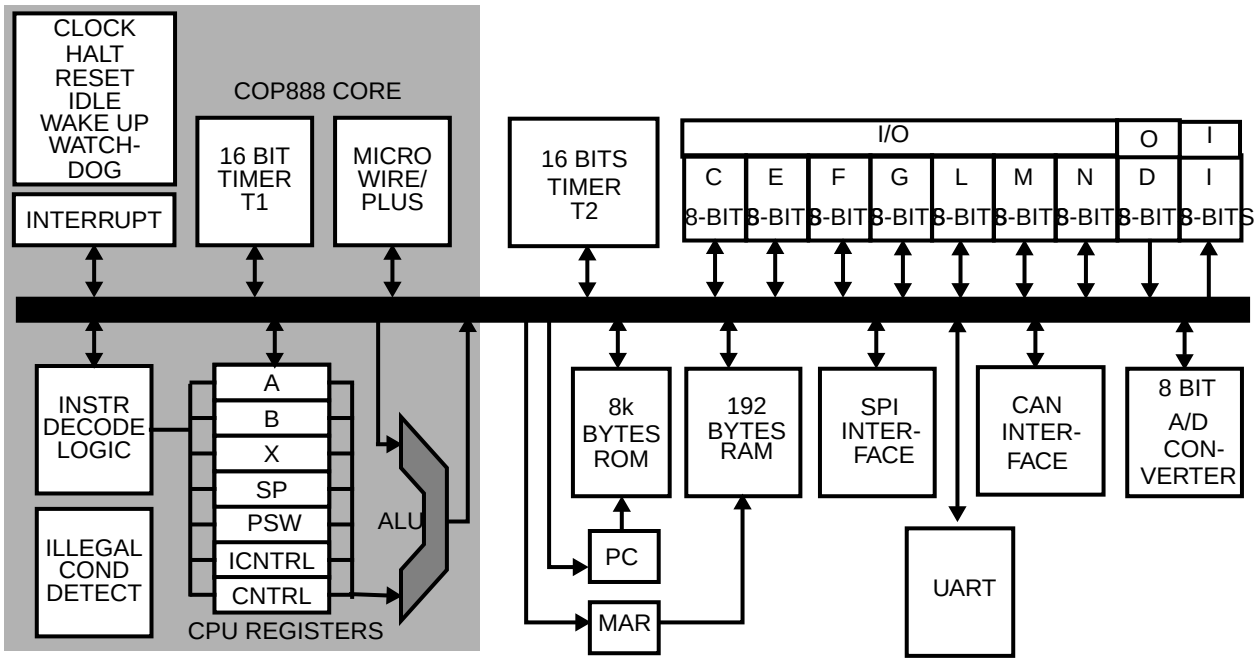
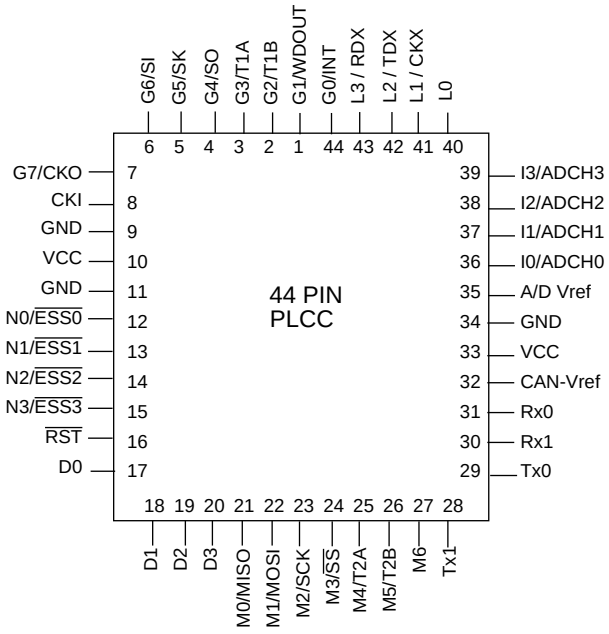


Figure 17-1 COP888EB Block Diagram

17.3 DEVICE PINOUTS/PACKAGES

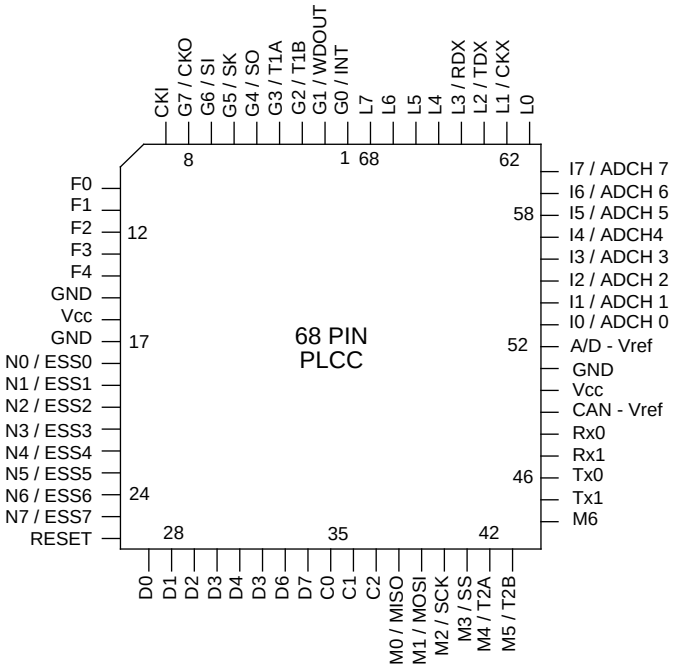
The COP888EB is available in 44-pin and 68-pin PLCC packages. Figure 17-2 shows the COP888EB device package pinouts.

Plastic Chip Carrier



Top View

Plastic Leaded Chip Carrier



Top View

Figure 17-2 Device Package Pinouts

Refer to the COP888EB data sheet for more detailed package information.

17.4 PIN DESCRIPTIONS

The COP888EB has the following dedicated function pins: DVcc, GND, CKI, and $\overline{\text{RESET}}$, A/D Vref, and the CAN Interface pins (RX0, RX1, TX0, TX1, and CAN Vref). DVcc and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. The A/D Vref, which is nominally set to the save voltage as DVcc, used for the A/D Converter reference voltage. The other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 17-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.

Table 17-1 COP888EB Pinouts

Port Pin	Type	ALT Function	44 PIN PLCC	68 PIN PLCC
G0	I/O	INT	44	1
G1	I/O	WDOUT	1	2
G2	I/O	T1B	2	3
G3	I/O	T1A	3	4
G4	I/O	SO	4	5
G5	I/O	SK	5	6
G6	I	SI	6	7
G7	I	CKO	7	8
D0	O		17	27
D1	O		18	28
D2	O		19	29
D3	O		20	30
D4	O			31
D5	O			32
D6	O			33
D7	O			34
I0	I	ADCH0	36	53
I1	I	ADCH1	37	54
I2	I	ADCH2	38	55
I3	I	ADCH3	39	56
I4	I	ADCH4		57
I5	I	ADCH5		58

Table 17-1 COP888EB Pinouts

Port Pin	Type	ALT Function	44 PIN PLCC	68 PIN PLCC
I6	I	ADCH6		59
I7	I	ADCH7		60
L0	I/O	MIWU	40	61
L1	I/O	MIWU; CKX	41	62
L2	I/O	MIWU; TDX	42	63
L3	I/O	MIWU; RDX	43	64
L4	I/O	MIWU		65
L5	I/O	MIWU		66
L6	I/O	MIWU		67
L7	I/O	MIWU		68
E4	I/O			
E5	I/O			
E6	I/O			
E7	I/O			
M0	I/O	MIWU; MISO	21	38
M1	I/O	MIWU; MOSI	22	39
M2	I/O	MIWU; SCK	23	40
M3	I/O	MIWU; $\overline{SS}$	24	41
M4	I/O	MIWU;T2A	25	42
M5	I/O	MIWU;T2B	26	43
M6	I/O	MIWU	27	44
M7	I/O			
N0	I/O	$\overline{ESS0}$	12	18
N1	I/O	$\overline{ESS1}$	13	19
N2	I/O	$\overline{ESS2}$	14	20
N3	I/O	$\overline{ESS3}$	15	21
N4	I/O	$\overline{ESS4}$		22

Table 17-1 COP888EB Pinouts

Port Pin	Type	ALT Function	44 PIN PLCC	68 PIN PLCC
N5	I/O	$\overline{\text{ESS5}}$		23
N6	I/O	$\overline{\text{ESS6}}$		24
N7	I/O	$\overline{\text{ESS7}}$		25
F0	I/O			10
F1	I/O			11
F2	I/O			12
F3	I/O			13
F4	I/O			14
F5	I/O			
F6	I/O			
F7	I/O			
C0	I/O			35
C1	I/O			36
C2	I/O			37
C3	I/O			
C4	I/O			
C5	I/O			
C6	I/O			
RX0	I		31	48
RX1	I		30	47
TX0	O		29	46
TX1	O		28	45
CANVref			32	49
CKI			8	9
$\overline{\text{RESET}}$			16	26
DVcc			10, 33	16, 50
GND			9, 11, 34	15, 17, 51
A/D Vref			35	52

17.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888EB depend on the package type. Refer to Table 17-1, COP888EB Pinouts, for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888EB device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate function of the port pin.

Port C, a general-purpose bidirectional I/O port, is partially available (pins 0-3) in the 68-pin package and is not available in the 44-pin package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in all COP888EB devices, although the 44-pin package only has pins 0-3 available. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port F, a general-purpose bidirectional I/O port, is partially available (pins 0-4) in the 68-pin package and is not available in the 44-pin package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port G, a bidirectional I/O port, is present in all COP888EB devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. The alternate functions of the port pins are described in Chapter 7.

Port I, a general-purpose dedicated input port, is present in all COP888EB devices, although only pins 0-3 are available in the 44-pin package. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. The alternate functions of the Port I pins are the A/D Converter inputs.

Port L, a general-purpose bidirectional I/O port, is present in all COP888EB devices, although only pins 0-3 are available in the 44-pin package. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for Multi-Input Wakeup/Interrupt, an alternate function. In addition, L1 through L3 can be used for another alternate function, UART I/O. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/Interrupt feature.

Port M, a general-purpose bidirectional I/O port, is present in all COP888EB devices. There are three memory locations associated with this port: one each for the data

register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for Multi-Input Wakeup/Interrupt, an alternate function. In addition, M0 through M3 can be used for the SPI Interface; and M4 and M5 can be used for Timer T2 I/O. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/Interrupt feature.

Port N, a general-purpose dedicated input port, is present in all COP888EB devices, although only pins 0-3 are available in the 44-pin package. There are four memory locations associated with this port: one each for the data register, for the configuration register, for reading the port pins directly, and for enabling the alternate functions. This port can be used for general-purpose I/O or for the SPI Slave Select Expander, an alternate function.

A device package that has Port C, E, F, M, or N but fewer than eight pins in the port contains unbonded, floating pads internally on the chip. For these devices, the software should write a 1 to the configuration register bits corresponding to the floating pins. This configures the port bits as outputs, which reduces leakage current.

17.6 PROGRAM MEMORY

The COP888EB contains 8K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space from 0000 to 1FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs.

17.7 DATA MEMORY

The COP888EB contains 192 bytes of static RAM memory, used for temporary data storage. The first 128 bytes of RAM occupy two address ranges in the lowest 256-byte data memory space: 112 bytes from 00-6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next 256-byte page of data memory space, starting at 0100 Hex and ending at 013F Hex.

17.8 REGISTER BIT MAPS

The COP888EB has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Table 17-2 through Table 17-19 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), Multi-Input Wakeup/Interrupt and Timer T0 (Chapter

6), WATCHDOG and Clock Monitor (Chapter 8), UART (Chapter 9), A/D Converter (Chapter 10), CAN Interface (Chapter 11), or SPI Interface (Section Section 17.15).

Table 17-2 SPICNTL, SPI Control Register (Address xx98)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRIE	STIE	SES-SEN	SPIMOD[1:0]		SCE	SPIEN	SLOOP

SRIE:	SPI Receive Interrupt Enable bit
STIE:	SPI Transmit Buffer Interrupt Enable bit
SESSEN:	SPI $\overline{SS}$ Expander Enable bit
SPIMOD[1:0]:	SPI Operation Mode Select bits
SCE:	SPI Clock Edge Select bit
SPIEN:	SPI Enable bit
SLOOP:	SPI Loop Back Mode Enable bit

Table 17-3 SPISTAT, SPI Status Register (Address xx99)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRORN	SRBNE	STBF	STBE	STFL	SESS-DET	Unused	

SRORN:	SPI Receiver Overrun.
SRBNE:	SPI Receive Buffer Not Empty.
STBF:	SPI Transmit Buffer Full.
STBE:	SPI Transmit Buffer Empty.
STFL:	SPI Transmit Buffer Flush.
SESSDET:	SPI $\overline{SS}$ Expander Detection.

Table 17-4 CTIM, CAN Bus Timing Register (Address xxA9)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PPS2	PPS1	PPS0	PS2	PS1	PS0	Unused	

PPS2-PPS0:	Propagation Segment Length Control bits
PS2-PS0:	Phase Segment Length Control bits
Unused:	These bits are unused

Table 17-5 CBUS, CAN Bus Control Register (Address xxAA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	RIAF	TXEN1	TXEN0	RXREF1	RXREF0	Reserved	FMOD

Reserved:	These bits are reserved and should not be used.
RIAF:	Receive Identifier Acceptance Filter Enable bit
TXEN1-TXEN0:	Transmit Output Enable bits
RXREF1-RXREF0:	Receive Input to V_{REF} Control Bits
FMOD:	Advanced Fault Confinement Mode Enable bit

Table 17-6 TCNTL, CAN Bus Transmit Control/Status Register (Address xxAB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NS1	NS0	TERR	RERR	CEIE	TIE	RIE	TXSS

NS1-NS0:	Node status bits
TERR:	Transmit Error Pending flag
RERR:	Receive Error Pending flag
CEIE:	CAN Error (transmit/receive) Interrupt Enable
TIE:	Transmit Interrupt Enable
RIE:	Receive Interrupt Enable
TXSS:	Transmit Start/Stop Control Bit

Table 17-7 RTSTAT, CAN Bus Receive/Transmit Status Register (Address xxAC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TBE	TXPND	RRTR	ROLD	RORN	RFV	RCV	RBF

TBE:	Transmit Buffer Empty
TXPND:	Transmission Pending
RRTR:	Received Remote Transmission Request
ROLD:	Received Overload Frame
RORN:	Receiver Overrun
RFV:	Received Frame is Valid
RCV:	Receive Mode Status bit
RBF:	Receive Buffer Full

Table 17-8 ENU, UART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN:	Parity enable bit
PSEL1:	Parity selection bit (with PSEL0)
XBIT9/PSEL0:	Transmit bit 9 / Parity selection bit (with PSEL1)
CHL1-CHL0:	Frame format selection bits (number of data bits)
ERR:	Error flag (data overrun, framing, or parity error)
RBFL:	Receive buffer full flag
TBMT:	Transmit buffer empty flag

Table 17-9 ENUR, UART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	Reserved	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag
FE:	Framing Error flag
PE:	Parity Error flag
RBIT9:	Receive bit 9 (when 9 bits per frame)
ATTN:	Attention mode enable bit
XMTG:	Transmitting; reset at end of last frame
RCVG:	Receiving; reset when RDX goes high

Table 17-10 ENUI, UART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	STP78	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2:	Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)
STP78:	Last Stop bit length (0 = 1 bit, 1 = 7/8 bit)
ETDX:	Enable TDX transmit pin
SSEL:	Synchronous select bit (0 = asynchronous, 1 = synchronous)
XRCLK:	External Receive Clock (0 = internal, 1 = external clock)
XTCLK:	External Transmit Clock (0 = internal, 1 = external clock)
ERI:	Enable Receive Interrupt
ETI:	Enable Transmit Interrupt

Table 17-11 BAUD, UART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0

BRD7–BRD0: Baud rate divisor (with PSR register)

Table 17-12 PSR, UART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8
PSR4–PSR0:	Baud rate prescaler						
BRD10–BRD8:	Baud rate divisor (with BAUD register)						

Table 17-13 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0:	Timer T2 control bits						
T2PNDA:	Timer T2 interrupt A pending flag						
T2ENA:	Timer T2 interrupt A enable bit						
T2PNDB:	Timer T2 interrupt B pending flag						
T2ENB:	Timer T2 interrupt B enable bit						

Table 17-14 WDSVR, WATCHDOG Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN
WS1–WS0:	WATCHDOG Window Select bits						
KEY4–KEY0:	WATCHDOG Key Data (01100)						
CMEN:	Clock Monitor Enable bit						

Table 17-15 ENAD, A/D Converter Control Register (Address xxCB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CHS2	CHS1	CHS0	DIFF	CONT	PRSC1	PRSC0	ADBSY
CHS2–CHS0:	Channel Selection bits						
DIFF:	Differential Mode (0 = single-ended, 1 = differential)						
CONT:	Continuous Mode (0 = single conversion, 1 = continuous)						
PRSC1–PRSC0:	A/D clock prescaler selection bits						
ADBSY:	A/D enable and Busy bit						

Table 17-16 ITMR, IDLE Timer Control Register (Address xxCF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESERVED					ITSEL2	ITSEL1	ITSEL0
RESERVED:	These bits are Reserved and should be ZERO						
ITSEL2–ITSEL0:	IDLE Timer Period Select Bits						

Table 17-17 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	uWPND	uWEN	T1PNDB	T1ENB
LPEN:	Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)						
T0PND:	Timer T0 (IDLE Timer) interrupt pending flag						
T0EN:	Timer T0 (IDLE Timer) interrupt enable bit						
uWPND:	MICROWIRE interrupt pending flag						
uWEN:	MICROWIRE interrupt enable bit						
T1PNDB:	Timer T1 interrupt B pending flag						
T1ENB:	Timer T1 interrupt B enable bit						

Table 17-18 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits
MSEL:	MICROWIRE Select bit
IEDG:	External Interrupt Edge selection bit
SL1–SL0:	MICROWIRE clock divide-by selection bits

Table 17-19 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC:	Half-Carry bit
C:	Carry bit
T1PNDA:	Timer T1 interrupt A pending flag
T1ENA:	Timer T1 interrupt A enable bit
EXPND:	External interrupt pending flag
BUSY:	MICROWIRE Busy flag
EXEN:	External interrupt enable bit
GIE:	Global Interrupt Enable

17.9 MEMORY MAP

Table 17-20 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888EB registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated "Reserved" in the table.

Table 17-20 COP888EB Data Memory Map (Sheet 1 of 4)

Address	Contents
0000 to 006F	On-Chip RAM bytes (112 bytes)
0070 to 007F	Unused RAM Address Space (Reads As All Ones)
0080	PORTMD, Port M Data Register
0081	PORTMC, Port M Configuration Register
0082	PORTMP, Port M Input Pins (Read Only)
0083	Reserved for Port M
0084	MMIWU Edge Select Register (MWKEDG)
0085	MMIWU Enable Register (MWKEN)
0086	MMIWU Pending Register (MWKPND)
0087	Reserved for MMIWU
0088	PORTND, Port N Data Register
0089	PORTNC, Port N Configuration Register
008A	PORTNP, Port N Input Pins (Read Only)
008B	PORTNX, Port N Alternate function Enable
008C to 008F	Unused RAM Address Space (Reads Undefined Data)
0090	PORTED, Port E Data Register
0091	PORTEC, Port E Configuration Register
0092	PORTEP, Port E Input Pins (Read Only)
0093	Reserved for port E
0094	PORTFD, Port F Data Register
0095	PORTFC, Port F Configuration Register
0096	PORTFP, Port F Input Pins (Read Only)
0097	Reserved for Port F
0098	SPICNTL, SPI Control register
0099	SPISTAT, SPI Status Register

Table 17-20 COP888EB Data Memory Map (Sheet 2 of 4)

Address	Contents
009A	SPIRXD, SPI current receive data (Read Only)
009B	SPITXD, SPI transmit data
009C to 009F	Unused
00A0	TXD1, Transmit 1 Data
00A1	TXD2, Transmit 2 Data
00A2	TDLC, Transmit Data Length Code and Identifier Low
00A3	TID, Transmit Identifier High
00A4	RXD1, Receive Data 1
00A5	RXD2, Receive Data 2
00A6	RIDL, Receive Data Length Code
00A7	RID, Receive Identify High
00A8	CSCAL, CAN Prescaler
00A9	CTIM, Bus Timing Register
00AA	CBUS, Bus Control Register
00AB	TCNTL, Transmit/Receive Control Register
00AC	RTSTAT Receive/Transmit Status Register
00AD	TEC, Transmit Error Count Register
00AE	REC, Receive Error Count Register
00AF	Reserved
xxB0 to xxB7	Reserved
xxB8	UART transmit buffer, TBUF
xxB9	UART receive buffer, RBUF
xxBA	UART control and status register, ENU
xxBB	UART receive control and status register, ENUR
xxBC	UART interrupt and clock source register, ENUI
xxBD	UART baud register, BAUD
xxBE	UART prescaler select register, PSR
xxBF	Reserved
xxC0	Timer T2 lower byte

Table 17-20 COP888EB Data Memory Map (Sheet 3 of 4)

Address	Contents
xxC1	Timer T2 upper byte
xxC2	Timer T2 autoload register T2RA lower byte
xxC3	Timer T2 autoload register T2RA upper byte
xxC4	Timer T2 autoload register T2RB lower byte
xxC5	Timer T2 autoload register T2RB upper byte
xxC6	Timer T2 control register, T2CNTRL
xxC7	WATCHDOG service register, WDSVR
xxC8	Multi-Input Wakeup edge select register, WKEDG
xxC9	Multi-Input Wakeup enable register, WKEN
xxCA	Multi-Input Wakeup interrupt pending register, WKPND
xxCB to xxCF	Reserved
xxD0	Port L data register, PORTLD
xxD1	Port L configuration register, PORTLC
xxD2	Port L input pins (read only), PORTCP
xxD3	Reserved
xxD4	Port G data register, PORTGD
xxD5	Port G configuration register, PORTGC
xxD6	Port G input pins (read only), PORTGP
xxD7	Port I input pins (read only), PORTI
xxD8	Port C data register, PORTCD
xxD9	Port C configuration register, PORTCC
xxDA	Port C input pins (read only), PORTCP
xxDB	Reserved
xxDC	Port D data register, PORTD
xxDD to xxDF	Reserved
xxE0 to xxE5	Reserved
xxE6	Timer T1 autoload register T1RB lower byte
xxE7	Timer T1 autoload register T1RB upper byte
xxE8	Interrupt control register, ICNTRL

Table 17-20 COP888EB Data Memory Map (Sheet 4 of 4)

Address	Contents
xxE9	MICROWIRE serial I/O shift register, SIO
xxEA	Timer T1 lower byte
xxEB	Timer T1 upper byte
xxEC	Timer T1 autoload register T1RA lower byte
xxED	Timer T1 autoload register T1RA upper byte
xxEE	Control register, CNTRL
xxEF	Processor Status Word register, PSW
xxF0 to xxFB	General-purpose memory-mapped registers
xxFC	X pointer register
xxFD	SP (Stack Pointer) register
xxFE	B pointer register
xxFF	S (data segment extension) register
0100 to 013F	RAM, additional 64 bytes
Reading memory locations 0070H–007FH will return all ones. Reading unused memory locations 00xxH–00xxH will return undefined data.	

17.10 RESET

Upon reset of the COP888EB, the ports and registers are initialized as follows:

Port C data register, PORTCD:	00
Port C configuration register, PORTCC:	00
Port D data register, PORTD:	FF
Port G data register, PORTGD:	00
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timers T2 control register, T2CNTRL :	00
Timers T1 and T2 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator, A and Timers T1 and T2:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00

Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
WATCHDOG service register, WDSVR:	D9
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
A/D Converter control register, ENAD:	00
A/D Converter result register, ADRSLT:	Unknown
IDLE Timer Control Register, ITMR	00
UART control and status register, ENU:	01
UART receive control and status register, ENUR:	00
UART interrupt and clock source register, ENUI:	00
UART baud register, BAUD:	Upon power-up reset, unknown. Upon external reset, unchanged.
UART prescaler select register, PSR:	00
CAN register bits:	All cleared to zero except for the Transmit Buffer Empty (TBE) bit of the Receive/Transmit Status (RTSTAT) register, which is set to 1; and the RID, RIDL, TID, and TDLC registers, which are unknown.
SPICNTL, SPI Control Register:	Cleared
SPISTAT, SPI Status Register:	Cleared except for the STBE and STBE bits, which are set to 1.

17.11 INTERRUPTS

Table 17-21 shows the types of interrupts in the COP888EB, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 17-21 COP888EB Interrupt Rank and Vector Addresses.

17.12 UART

The COP888EB has an on-chip Universal Asynchronous Receiver Transmitter (UART), which can be used for transmitting and receiving data serially, either synchronously or asynchronously. This full-duplex, double-buffered UART is fully programmable, and can

Table 17-21 COP888EB Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	CAN Receive (RBF, RFV set)	01FA–01FB
4	CAN Error (transmit/ receive)	01F8–01F9
5	CAN Transmit (TBE set)	01F6–01F7
6	Pin G0 Edge (external)	01F4–01F5
7	MICROWIRE/PLUS (Busy low) or SPI Interface (SRBF or STBE set)	01F2–01F3
8	Timer T0 (timer underflow)	01F0–01F1
9	UART (receive buffer full)	01EE–01EF
10	UART (transmit buffer empty)	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	Timer T1A/Underflow	01E6–01E7
14	Timer T1B	01E4–01E5
15	Port L / Port M / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (7/8, 1, 1-7/8, or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)

- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability
- Error detection circuitry and diagnostic self-test capability

The UART is described in Chapter 9.

17.13 A/D CONVERTER

The COP888EB has a multi-channel, multiplexed-input, ratiometric Analog-to-Digital (A/D) Converter. The A/D Converter receives an analog voltage signal on an input pin (or a pair of input pins), and converts the analog signal into a 8-bit digital value. The digital value can then be read by the software from a memory-mapped register.

The A/D Converter is useful in applications where the software needs to read a quantity (for example, temperature or speed) from an analog sensor. Up to eight different single-ended or four different differential inputs can be handled by a single COP888EB device through the multiplexed input channels.

The COP888EB has a separate Vcc reference voltage input pin called "A/D Vcc" used for the A/D Converter positive reference voltage, unlike some COP888 devices that use the Vcc power supply pin for this purpose.

The A/D Converter is described in Chapter 10.

17.14 CAN INTERFACE

The COP888EB has a built-in Controller Area Network (CAN) interface. It supports the CAN Specification Revision 2.0 Part B, "Passive" type operation, at data rates of up to 1 Mbits/second using standard message identifiers. This interface is suitable for lower-speed (CAN class A and B) applications. For example, in automotive applications, it is suitable for control of instrument gauges, door locks, headlights, and audio systems.

The CAN interface of the COP888EB is described in Chapter 11.

17.14.1 HALT/IDLE and the CAN Interface

In order to reduce the overall power consumption of the COP888EB device in the HALT/IDLE mode, you should prepare the CAN interface before entering the HALT/IDLE mode using the following two steps:

1. Disable the main receive comparator by resetting both the TXEN0 and TXEN1 bits in the CBUS register. This is necessary to allow proper resynchronization to the CAN bus after leaving the HALT/IDLE mode.

2. Disable the CAN wakeup comparator by resetting bit 7 in the Port M Wakeup Enable (MWKEN) register. This is necessary to prevent a transition on the CAN bus from waking up the device from the HALT/IDLE mode.

Note: If both the main receive comparator and wakeup comparator are disabled on the chip as described above, the CAN voltage reference is also disabled and the CAN-Vref output pin (normally Vcc/2) is placed the high-impedance state.

17.15 SPI INTERFACE

The COP888EB has a Serial Peripheral Interface (SPI), a synchronous bidirectional serial communications interface used to transfer data between modules called "masters" and "slaves." The Serial Peripheral Interface is an alternate function of Port M and Port N.

A network contains one master and any number of slaves. The master controls the flow of data to or from a slave. The communications bus has two data lines, called MISO (Master In, Slave Out) and MOSI (Master Out, Slave In). The master also generates a Shift Clock (SCK) signal and a Slave Select ($\overline{SS}$) signal.

Data bytes are transmitted over the bus in units called "frames," with up to 12 bytes in a frame. A frame is defined as the time between a falling edge and a rising edge of the $\overline{SS}$ signal, as shown in Figure 17-3.

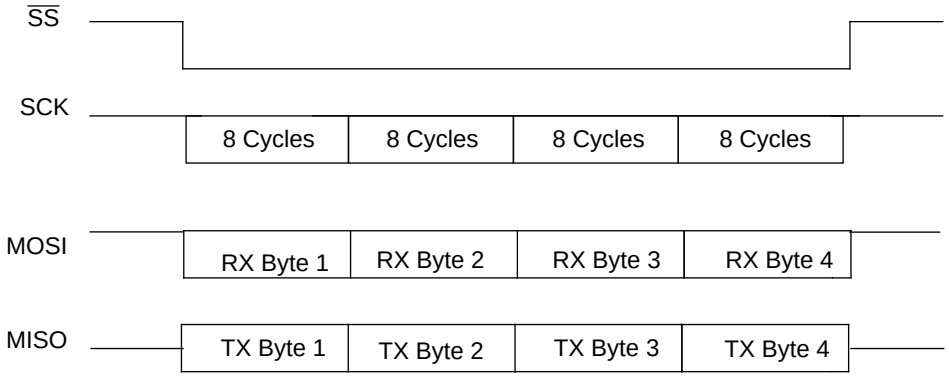


Figure 17-3 SPI Frame Timing

Figure 17-3 is a block diagram illustrating the basic operation of the SPI circuit in the COP888EB. Data bits are transmitted serially in packets of eight bits. The bits are shifted into or out of a shift register, one bit on each active edge of the Shift Clock (SCK) signal. The active edge can be programmed to be either the rising or falling edge, with

data shifted out of the source on one edge and shifted into the destination on the other edge.

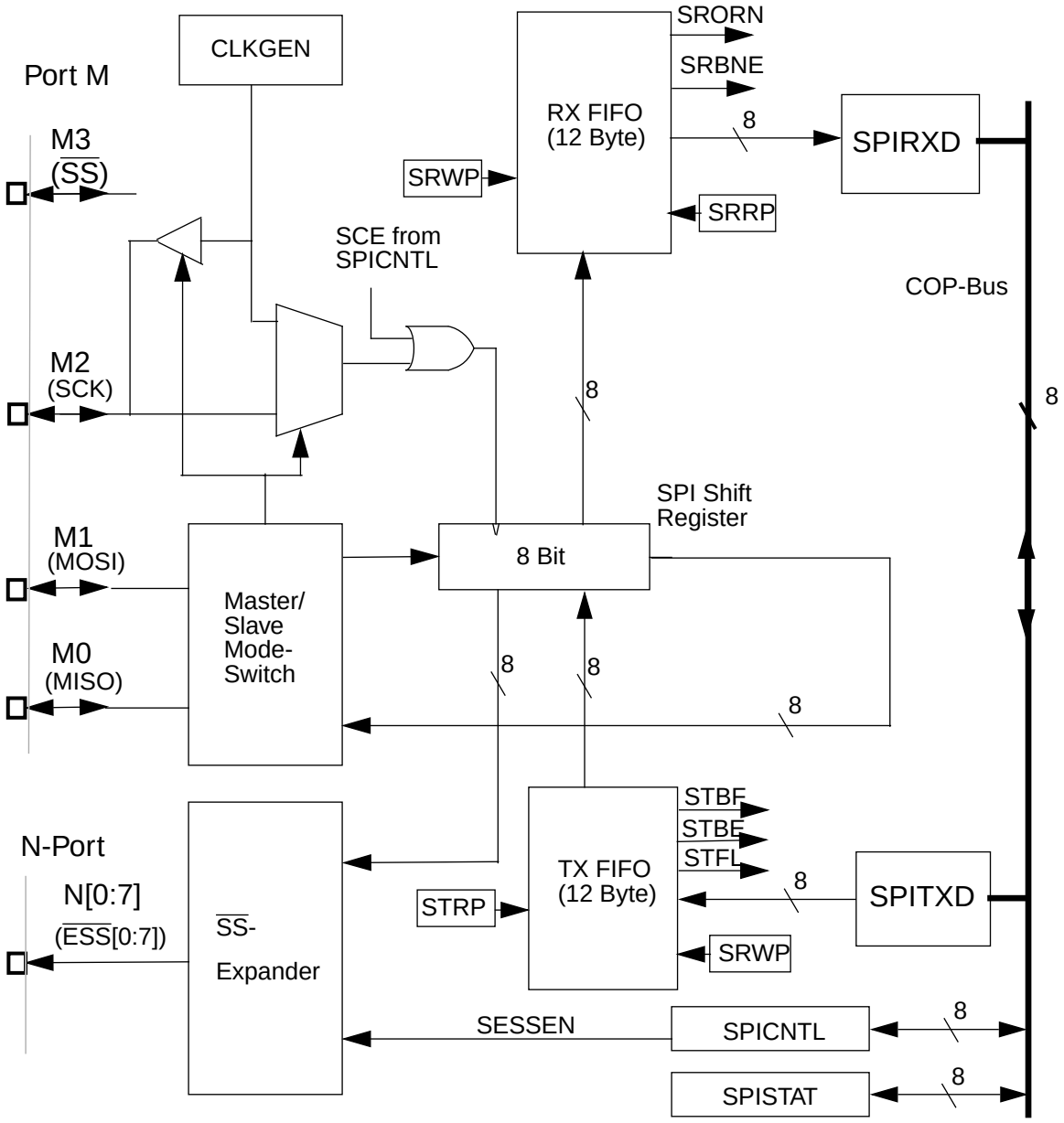


Figure 17-4 SPI Block Diagram

There are two 12-byte FIFO buffers, one to serve as a receive buffer and one to serve as a transmit buffer. They allow a maximum message length of 12 bytes in both the transmit and receive direction without CPU interaction. With CPU interaction, a much larger number of bytes can be received or transmitted. Each FIFO has two pointers: one for the CPU side and one for the SPI side.

The CPU controls the SPI interface using to memory-mapped registers: the SPI Control (SPICNTL) register and the SPW Status (SPISTAT) register. Several different operating modes are possible such as master, slave, and loop back mode.

17.15.1 Control Register Format

Table 17-22 shows the format of the SPI Control (SPICNTL) register. This memory-mapped, read/write register is programmed by the CPU to control the Serial Peripheral Interface. The register resides at address 0098 Hex.

Table 17-22 SPICNTL, SPI Control Register (Address 0098)

Table 17-22 SPI Control (SPICNTL) register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRIE	STIE	SESSEN	SPIMOD[1:0]		SCE	SPIEN	SLOOP

SRIE: SPI Receive Interrupt Enable bit:

0 = disable receive interrupt

1 = enable receive interrupt

STIE: SPI Transmit Interrupt Enable bit:

0 = disable transmit buffer interrupt

1 = enable transmit buffer interrupt

SESSEN: SPI $\overline{SS}$ Expander Enable bit:

0 = disable $\overline{ESS}$ mode

1 = enable $\overline{ESS}$ mode

SPIMOD1-SPIMOD0: SPI Operation Mode Select bits:

00 = slave mode, $f_{SCK} = f_{CKI}/40$ depends on external

01 = master mode, $f_{SCK} = f_{CKI}/10$

10 = master mode, $f_{SCK} = f_{CKI}/40$ (standard master mode)

11 = master mode, $f_{SCK} = f_{CKI}/10$

SCE: SPI Clock Edge Select bit:

0 = data shifted out on falling edge of SCK and shifted in on rising edge of SCK

1 = data shifted out on rising edge of SCK and shifted in on falling edge of SCK

SPIEN: SPI Enable bit:

0 = disable SPI

1 = enable SPI (MISO, MOSI, SCK, and $\overline{SS}$ alternate functions)

SLOOP: SPI Loop Back Mode Enable bit:

0 = disable loop back mode (for normal operation)

1 = enable loop back mode (MISO and MOSI internally connected for testing)

17.15.2 Status Register Format

Table 17-23 shows the format of the SPI Status (SPISTAT) register. This memory-mapped, read-only register shows the current status of the Serial Peripheral Interface. The register resides at address 0099 Hex.

Table 17-23 SPISTAT, SPI Status Register (Address 0099)

Table 17-23 SPI Status (SPISTAT) register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRORN	SRBNE	STBF	STBE	STFL	SESSDET	Unused	

SRORN: SPI Receiver Overrun.

SRBNE: SPI Receive Buffer Not Empty.

STBF: SPI Transmit Buffer Full.

STBE: SPI Transmit Buffer Empty.

STFL: SPI Transmit Buffer Flush.

SESSDET: SPI $\overline{SS}$ Expander Detection.

The SPI Receiver Overrun (SRORN) bit is set when the SPI interface attempts to write data into an RX FIFO location that has not yet been read by the CPU. This event generates a receive interrupt if the interrupt is enabled. The receive interrupt is enabled by setting the SRIE bit in the SPI Control Register.

If an SPI Receiver Overrun occurs, the write operation is cancelled and the data written by the SPI interface is lost. The pointer to the FIFO on the SPI side is frozen and no more data can be written to the FIFO from the SPI side until the SRORN bit is reset.

The SRORN bit, once set, remains set until it is reset by the CPU writing a "dummy" byte to the SPI Status Register. The only effect of writing to the SPI Status Register address is to reset the SRORN bit. The SPI Status Register is a read-only register, so it does not matter what data is written.

The SPI Receive Buffer Not Empty (SRBNE) bit is set when the SPI interface writes to the Receive FIFO and the result is unequal values for the two pointers to the FIFO (from the SPI side and from the CPU side). The SRBNE bit is reset automatically when the CPU reads the SPI Receive Data register and the two pointers are equal again. The setting of the SRBNE bit generates a receive interrupt if this interrupt is enabled with the SRIE bit in the SPI Control Register.

The SPI Transmit Buffer Full (STBF) bit is set when the CPU writes to the Transmit FIFO and the result is equal values for the two pointers to the FIFO (from the SPI side and from the CPU side). The STBF bit is reset automatically when the SPI interface reads the Transmit FIFO and the two pointers are unequal again.

The SPI Transmit Buffer Empty (STBE) bit is set when the SPI interface reads the last bit from the SPI Transmit Data register and the result is equal values for the two pointers to the FIFO (from the SPI side and from the CPU side). This condition generates

an STBE interrupt. The STBE bit is reset automatically when the CPU writes the Transmit FIFO.

Note: When the COP888EB device is reset, the STBE and STFL bits are both set to 1 and the other register bits are cleared.

The SPI Transmit Buffer Flush (STFL) bit is set when the $\overline{SS}$ signal goes high before all data in the Transmit FIFO could be transmitted. In other words, the bit is set if the $\overline{SS}$ signal goes high and the two pointers to the Transmit FIFO (from the SPI side and from the CPU side) are not equal, or they are equal but the current byte has not been completely transmitted from the SPI shift register. If this happens, both pointers are reset to zero.

The SPI $\overline{SS}$ Expander Detection (SESSDET) bit is 1 when the SPI interface operates in the $\overline{SS}$ Expander mode and 0 for normal $\overline{SS}$ operation. It is set when MOSI is low at the falling edge of $\overline{SS}$ immediately after Port N has been programmed (eighth SCK bit, 8 microseconds at SCK = 1 MHz). The SESSDET bit is reset automatically at the rising edge of $\overline{SS}$.

NOTE: The SPI master must hold $\overline{SS}$=0 long enough to allow the device to read SESSDET. Otherwise, the SESSDET information will be lost.

17.15.3 $\overline{SS}$-Expander

The $\overline{SS}$-Expander block optionally generates up to eight output signals on Port N. These signals can be used as Expanded Slave Select signals (ESS7-ESS0) or as host-programmable, general-purpose control signals. The $\overline{SS}$-Expander is enabled by setting the SESSN bit in the SPI Control (SPICNTL) register.

The $\overline{ESS}$ control logic detects the condition MOSI low at the falling edge of $\overline{SS}$. When this occurs, the $\overline{SS}$-Expander is programmed with the contents of the first MOSI byte. In other words, after complete reception of the first byte, the bits of the first byte appear in Port N bits 7-0. If any bytes follow the first MOSI byte, the data in those bytes are ignored. The programmed bit values are held active while $\overline{SS}$ is low. After the rising edge of $\overline{SS}$, all bits of Port N are set to 1.

NOTE: The Port N pins can be individually programmed to operate either as $\overline{SS}$-Expander outputs or as general-purpose input/output port pins.

Figure 17-5 shows the $\overline{SS}$-Expander logic and Figure 17-6 shows the signal timing for using the $\overline{SS}$-Expander.

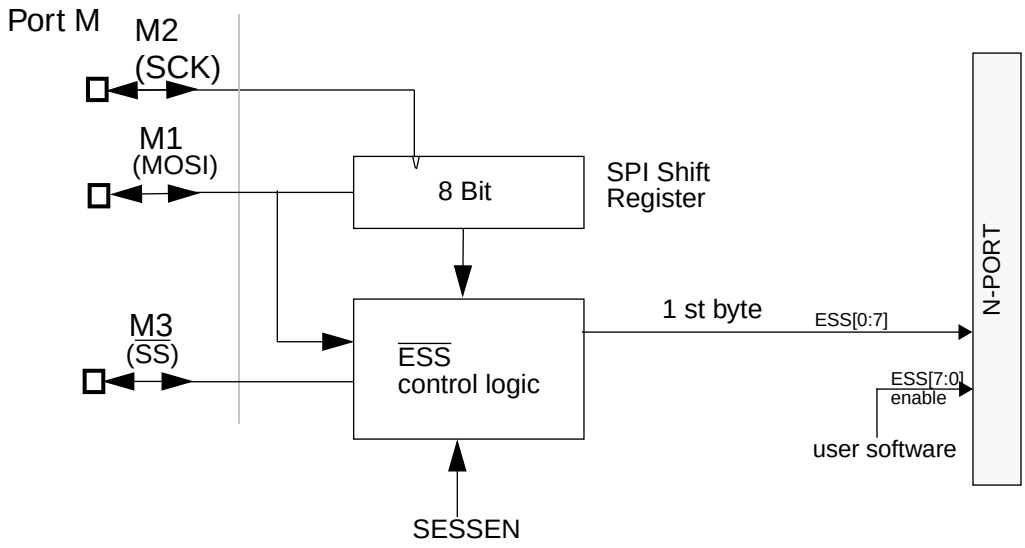
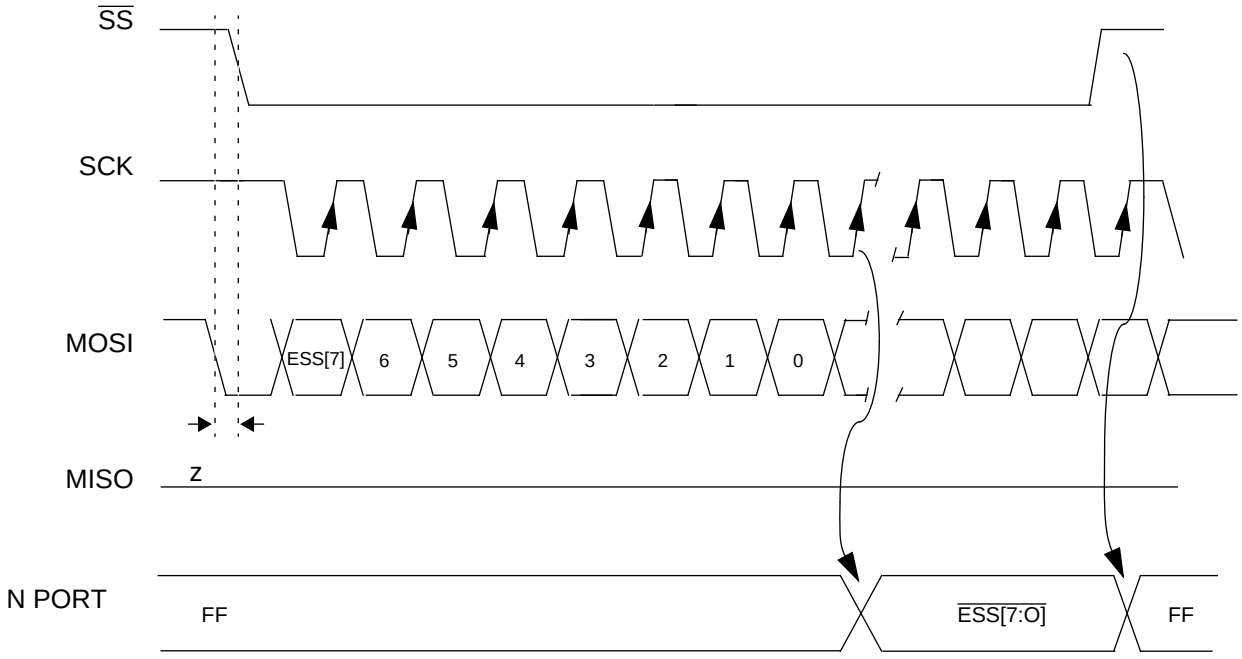


Figure 17-5 $\overline{SS}$-Expander Logic



SESSEN=1, SCE=0. If MOSI=0 at the falling edge of $\overline{SS}$, the $\overline{ESS}$ programming mode is detected and all N-port alternate functions are enabled.

Figure 17-6 $\overline{SS}$-Expander Programming Timing

17.15.4 SPI Synchronization

Operation of the SPI is enabled by setting the SPIEN bit in the SPI Control (SPICNTL) register. When this happens, the SPI internal receive and transmit shift clock is kept disabled until $\overline{SS}$ becomes inactive. If $\overline{SS}$ is active when the SPIEN bit is set, no receive or transmit operation is possible until $\overline{SS}$ becomes inactive.

17.15.5 Transmission Start in Master Mode

Data transmission in the master mode starts when the $\overline{SS}$ signal is switched active. No SCK signal is generated in the master mode, so no data is transmitted as long as the $\overline{SS}$ signal is kept high. In other words, $\overline{SS}$ must be switched low to generate SCK.

Resetting the $\overline{SS}$ signal in the master mode immediately stops the transmission and flushes the transmit FIFO. Thus, you must only reset $\overline{SS}$ if STBDE is set, or if SCK is high with SCE=0, or SCK is low with SCE=1.

17.15.6 SPI Receive Data Register (SPIRXD)

The SPI Receive Data (SPIRXD) register is a read/write register that holds the receive data at the current SPI Receive Read Pointer (SRRP) location. When the CPU reads from the SPIRXD register to the accumulator, it reads from the receive FIFO at the SRRP pointer location and then that pointer is automatically incremented.

When the CPU writes to the SPIRXD register, it writes to the receive FIFO at the SRRP pointer location without changing the SRRP pointer.

NOTE: The SRRP pointer is not incremented during a breakpoint.

A write operation to the SPIRXD register from the SPI interface side writes to the current SPI Receive Write (SRWP) pointer location and then increments that pointer.

17.15.7 SPI Transmit Data Register (SPITXD)

The SPI Transmit Data (SPITXD) register is a read/write register that holds the data at the current SPI Transmit Write Pointer (STWP) location. When the CPU writes the SPIRXD register, it writes to the transmit FIFO at the STWP pointer location and then that pointer is automatically incremented.

When the CPU reads from the SPITXD register, it reads from the receive FIFO at the STWP pointer location without changing the STWP pointer.

NOTE: Do not use any read-modify-write instructions on this register.

A read operation of the SPIRXD register from the SPI interface side reads from the current SPI Transmit Read (STRP) pointer location and then increments that pointer.

17.15.8 SPI Receive FIFO

The SPI RX FIFO is a 12-byte first-in, first-out buffer used for storing data being received through the SPI interface. The CPU reads data in this FIFO by reading the SPIRXD

register. The SPI Receive Read Pointer (SRRP) specifies the current location in the FIFO from which the CPU reads. This pointer is incremented automatically each time the CPU reads data from the FIFO.

Data is written to the RX FIFO by the SPI interface. The write location from the SPI side is determined by the SPI Receive Write Pointer (SRWP). This pointer is incremented each time the SPI interface stores data into the FIFO.

Each FIFO pointer counts from 0 to 11 and then wraps back to 0, and therefore points to successive FIFO locations and then wraps back to the beginning. The pointers are never decremented. Both pointers are cleared by a reset operation.

The SPI Receive Buffer Not Empty (SRBNE) bit in the SPI Status Register indicates the current status of the RX FIFO. This flag is defined as follows:

SRBNE == (SRWP != SRRP) & !SRORN

SRORN (SPI Receiver Overrun) is set when SRWP == SRRP and after a write from the SPI side. It is reset by a write to the SPI Status Register.

While SRORN is set, no writes to the RX FIFO are allowed from the SPI side and the SRWP pointer is frozen. Resetting the SRORN bit (if it was set) clears the SRWP and SRRP pointers. To prevent incorrect clearing of the RX FIFO when entering the HALT/IDLE mode, enable the MIWU or Port M3 ($\overline{SS}$) by setting bit 3 in the MWKEN register.

17.15.9 SPI Transmit FIFO

The SPI TX FIFO is a 12-byte first-in, first-out buffer used for storing data being transmitted through the SPI interface. The CPU writes data to this FIFO by writing the SPITXD register. The SPI Transmit Write Pointer (STWP) specifies the current location in the FIFO to which the CPU writes. This pointer is incremented automatically each time the CPU writes data to the FIFO.

Data is read from the TX FIFO by the SPI interface. The read location from the SPI side is determined by the SPI Transmit Read Pointer (STRP). This pointer is incremented each time the SPI interface reads data from the FIFO.

Each FIFO pointer counts from 0 to 11 and then wraps back to 0, and therefore points to successive FIFO locations and then wraps back to the beginning. The pointers are never decremented. Both pointers are cleared by a reset operation.

The SPI Transmit Buffer Full (STBF) bit in the SPI Status Register indicates the current status of the TX FIFO. This flag is set when STRP == STWP after a write from the CPU. It is reset when ((STRP != STWP) | STBE) after a read from the SPI interface, where SPI Transmit Buffer Empty (STBE) is defined to be STBE == (STRP == STWP) after a read from the SPI interface.

If the $\overline{SS}$ signal becomes high before the last bit of the last byte in the TX FIFO is transmitted, both STRP and STWP are reset to zero. The SPI Transmit Buffer Flush (STFL) and SPI Transmit Buffer Empty (STBE) bits are also set in the SPI Status Register.

17.15.10Pointer Operation with SPI Disabled

If the SPI interface is disabled (SPIEN bit cleared), all SPI FIFO-related pointers are reset to zero. As long as the SPI interface is disabled, the pointers are kept at zero, and read/write operations to SPITXD and SPIRXD do not affect the pointers.

17.15.11Loop Back Mode

The loop back mode is a operating mode in which the MISO and MOSI bus pins are connected together internally in the device for testing purposes. In the master mode, the MOSI output is internally connected to the MISO input. In the slave mode, the MISO output is internally connected to the MOSI input. Data bits from the transmit FIFO are communicated to the receive FIFO through the SPI Shift register, as shown in Figure 17-7.

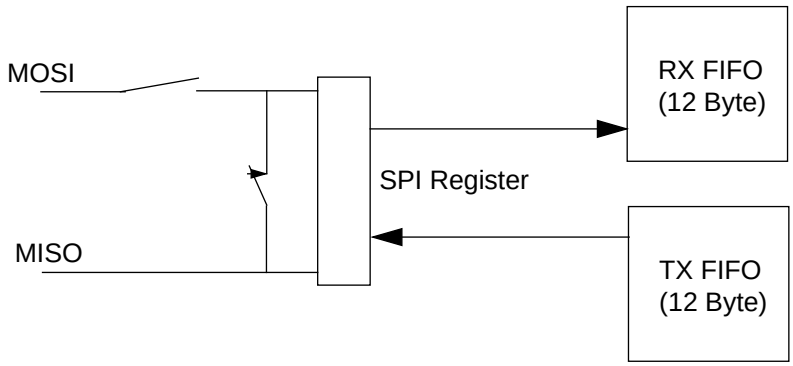


Figure 17-7 SPI Interface in Loop Back Mode

The loop back mode is enabled by setting the SLOOP bit in the SPI Control (SPCNTL) register.

17.15.12HALT/IDLE Mode

If the COP888EB device enters the HALT/IDLE mode, both the receive and transmit FIFOs are flushed (reset to zero). To prevent erroneous clearing of the receive FIFO when entering the HALT/IDLE mode, disable $\overline{SS}$ first by enabling the MIWU or Port M3 function by setting bit 3 in the MWKEN register.

When the device leaves the HALT/IDLE mode, SPI synchronization is done as described in Section 17.15.4. (just like after enabling the SPI by setting the SPIEN bit). The SPI Receive Data (SPIRXD) register and SPI Transmit Data (SPITXD) register are initialized in the same manner as after a reset operation, and the SPISTAT register bits are initialized as follows:

SRORN: unchanged

SRBNE and STBF: 0

STBF and STFL: 1

SESSDET: depends on the $\overline{SS}$ and MOSI line

17.16 MASK OPTIONS

The COP888EB has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options can also be programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/10)
G7 (CKO) is the clock generator output to a crystal/resonator
CKI is the clock input.

Option 2: HALT

=1 Enable HALT mode

Option 3: COP888EB Bonding

=1 44-pin PLCC

=2 68-pin PLCC

Chapter 18

COP888EK

18.1 INTRODUCTION

The COP888EK is a member of the COP888 Feature Family 8-bit microcontrollers. Like all members of this family, it provides high-performance, economical solutions for embedded control applications. The COP888EK is a basic member of the family, useful in low-cost applications that do not require a UART, or high performance A/D converter. Although the device does not have a built-in A/D converter, a low-cost, single-slope, 6-channel A/D converter can be implemented by using the device's comparator and timer.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888EK.

- Program Memory: Mask-programmed ROM, 8K bytes
- Data Memory: Static RAM, 256 bytes
- S-Register Data Memory Extension: Yes
- 16-Bit Programmable Timers: Timers T2 and T3
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- WATCHDOG and Clock Monitor: Yes
- Additional features: Comparator (with input multiplexor and current source) and A/D Converter (implemented with an external current source and the built-in comparator and timer).

Several versions of the COP888EK are available that operate over different voltage and temperature ranges. Refer to the COP888EK data sheet for more specific information.

This chapter describes the device-specific features of the COP888EK. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

18.2 BLOCK DIAGRAM

Figure 18-1 is a block diagram showing the basic functional blocks of the COP888EK. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

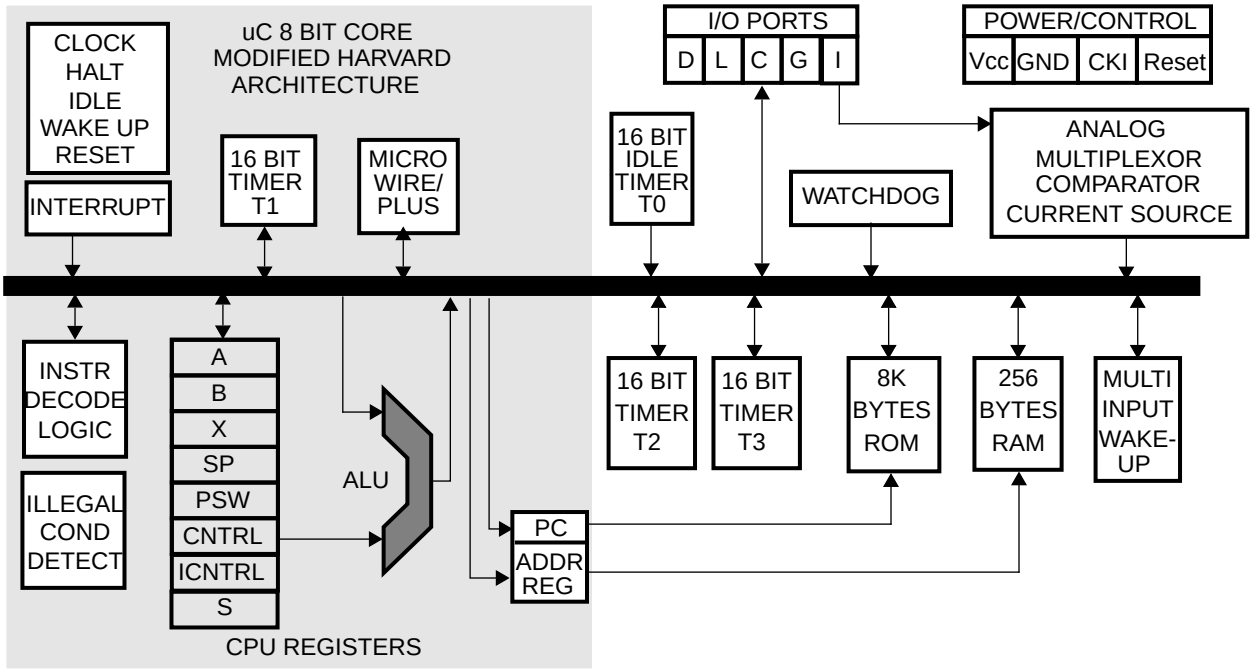


Figure 18-1 COP888EK Block Diagram

18.3 DEVICE PINOUTS/PACKAGES

The COP888EK is available in 28-pin DIP, 28-pin SO, 40-pin DIP, and 44-pin PLCC packages. Figure 18-2 shows the COP888EK device package pinouts.

Refer to the COP888EK data sheet for more detailed package information.

18.4 PIN DESCRIPTIONS

The COP888EK has four dedicated function pins: V_{CC}, GND, CKI, and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 18-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.

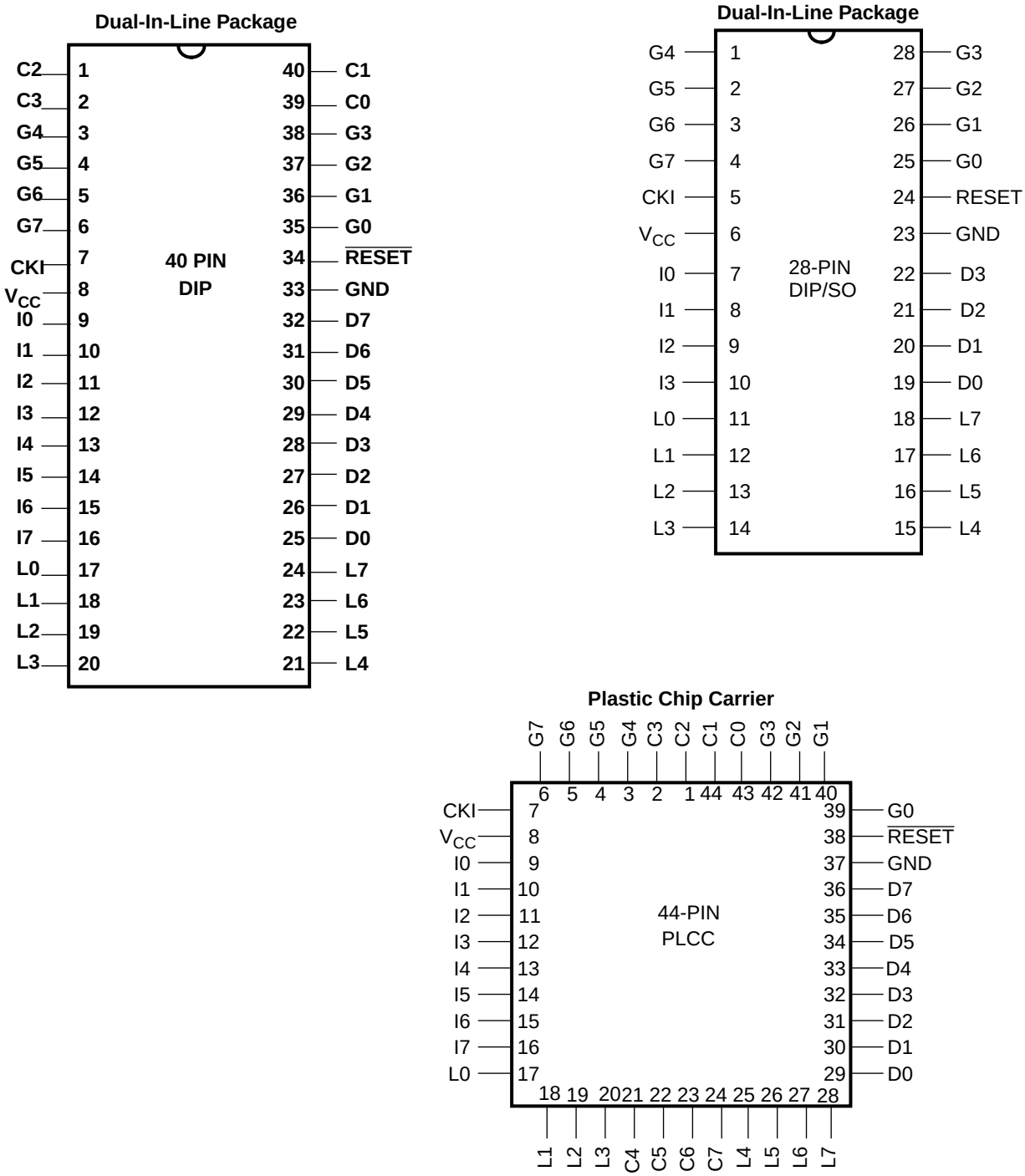


Figure 18-2 Device Package Pinouts

Table 18-1 COP888EK Pinouts

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
L0	I/O	MIWU	11	17	17
L1	I/O	MIWU	12	18	18
L2	I/O	MIWU	13	19	19
L3	I/O	MIWU	14	20	20
L4	I/O	MIWU or T2A	15	21	25
L5	I/O	MIWU or T2B	16	22	26
L6	I/O	MIWU or T3A	17	23	27
L7	I/O	MIWU or T3B	18	24	28
G0	I/O	INT	25	35	39
G1	WDOUT		26	36	40
G2	I/O	T1B	27	37	41
G3	I/O	T1A	28	38	42
G4	I/O	SO	1	3	3
G5	I/O	SK	2	4	4
G6	I	SI	3	5	5
G7	I/CKO	HALT RESTART	4	6	6
D0	O		19	25	29
D1	O		20	26	30
D2	O		21	27	31
D3	O		22	28	32
D4	O			29	33
D5	O			30	34
D6	O			31	35
D7	O			32	36
I0	I	COMPIN1+	7	9	9
I1	I	COMPIN–/Current Source Out	8	10	10
I2	I	COMPIN0+	9	11	11

Table 18-1 COP888EK Pinouts (Continued)

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
I3	I	COMPOUT/ COMPIN2+	10	12	12
I4	I	COMPIN3+		13	13
I5	I	COMPIN4+		14	14
I6	I	COMPIN5+		15	15
I7	I	COMPOUT		16	16
C0	I/O			39	43
C1	I/O			40	44
C2	I/O			1	1
C3	I/O			2	2
C4	I/O				21
C5	I/O				22
C6	I/O				23
C7	I/O				24
V_{CC}			6	8	8
GND			23	33	37
CKI			5	7	7
$\overline{\text{RESET}}$			24	34	38

18.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888EK depend on the package type. The 28-pin packages have Port L, Port G, and a subset of Port D and Port I. The 40-pin packages have Port L, Port G, Port D, Port I, and a subset Port C. The 44-pin packages have a complete set of ports, including all bits of Port L, Port G, Port D, Port I, and Port C. Refer to Table 18-1 for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888EK device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate functions of the port pins.

Port C, a general-purpose bidirectional I/O port, is present in COP888EK 44-pin packages and partially available (pins 0-3) in 40-pin packages. There are three memory

locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in all COP888EK devices, although the 28-pin package only has pins 0-3 available. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP888EK devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G1 is a dedicated Watchdog output that can be used to reset the device in the event of a Watchdog or Clock Monitor error. This pin can not be used a general purpose input or output, and should be left unconnected if not used to signal an error condition. Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I, a general-purpose dedicated input port, is present in all COP888EK devices, although not all eight pins are available in the 28-pin packages. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. The alternate functions of pins I1 though I6 are the comparator inputs and outputs. Comparator 2 is not available in 28-pin packages because the necessary Port I pins are not present in those packages.

Port L, a general-purpose bidirectional I/O port, is present in all COP888EK devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for several alternate functions: the Multi-Input Wakeup/Interrupt inputs, Timer T2 and Timer T3. The port inputs have Schmitt triggers to support the Multi-Input Wakeup/Interrupt feature.

18.6 PROGRAM MEMORY

The COP888EK contains 8K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space from 0000 to 01FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs, as described later in this chapter.

18.7 DATA MEMORY

The COP888EK contains 256 bytes of static RAM memory, used for temporary data storage. The first 128 bytes of RAM occupy two segments in the lowest 256-byte data memory space: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next 256-byte page of data memory space, starting at 0100 Hex and ending at 017F Hex.

18.8 REGISTER BIT MAPS

The COP888EK has several 8-bit memory-mapped registers used for controlling the CPU, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 18-2 through 18-7 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: Core registers (Chapter 2), Timers (Chapter 4), Watchdog and Clock Monitor (Chapter 8), or Multi-Input Wakeup/Interrupt and Timer T0 (Chapter 6).

Table 18-2 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0:			Timer T2 control bits				
T2PNDA:			Timer T2 interrupt A pending flag				
T2ENA:			Timer T2 interrupt A enable bit				
T2PNDB:			Timer T2 interrupt B pending flag				
T2ENB:			Timer T2 interrupt B enable bit				

Table 18-3 T3CNTRL, Timer T3 Control Register (Address xxB6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T3C3	T3C2	T3C1	T3C0	T3PNDA	T3ENA	T3PNDB	T3ENB

T3C3-T3C2-T3C1-T3C0: Timer T3 control bits

T3PNDA: Timer T3 interrupt A pending flag

T3ENA: Timer T3 interrupt A enable bit

T3PNDB: Timer T3 interrupt B pending flag

T3ENB: Timer T3 interrupt B enable bit

Table 18-4 WDSVR, Watchdog Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN

WS1–WS0: Watchdog Window Select bits

KEY4–KEY0: Watchdog Key Data (01100)

CMEN: Clock Monitor Enable bit

Table 18-5 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN: Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)

T0PND: Timer T0 (IDLE Timer) interrupt pending flag

T0EN: Timer T0 (IDLE Timer) interrupt enable bit

µWPND: MICROWIRE interrupt pending flag

µWEN: MICROWIRE interrupt enable bit

T1PNDB: Timer T1 interrupt B pending flag

T1ENB: Timer T1 interrupt B enable bit

Table 18-6 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits
MSEL:	MICROWIRE Select bit
IEDG:	External Interrupt Edge selection bit
SL1-SL0:	MICROWIRE clock divide-by selection bits

Table 18-7 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC:	Half-Carry bit
C:	Carry bit
T1PNDA:	Timer T1 interrupt A pending flag
T1ENA:	Timer T1 interrupt A enable bit
EXPND:	External interrupt pending flag
BUSY:	MICROWIRE Busy flag
EXEN:	External interrupt enable bit
GIE:	Global Interrupt Enable

Table 18-8 CMPSL, Comparator Register (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMPT2B	CMPISEL 2	CMPISEL 1	CMPISEL 0	CMPOE	CMPRD	CMPEN	CMPNEG

CMPT2B Selects the timer T2B input to be driven directly by the comparator output. If the comparator is disabled (CMPEN=0), this function is disabled, i.e. the T2B input is connected to Port L5

CMPISEL0/1/2 Will select one of seven possible sources (I0/I2/I3/I4/I5/I6/internal reference) as a positive input to the comparator (see Table 18-9 for more information.)

CMPOE Enables the comparator output to either pin I3 or pin I7 (“1”=enable) depending on the value of CMPISEL0/1/2.

CSEN Enables the internal constant current source. This current source provides a nominal 20 µA constant current at the I1 pin. This current can be used to ensure a linear charging rate on an external capacitor. This bit has no affect and the current source is disabled if the comparator is not enabled (CMPEN=0).

CMPEN Enable the comparator (“1” = enable)

CMPNEG Will drive I1 to a low level. This bit can be used to discharge an external capacitor. This bit is disabled if the comparator is not enabled (CMPEN=0)

18.9 ANALOG FUNCTION BLOCK

This device contains an analog function block which allows for single slope, low cost, A/D conversion of up to 6 channels. A block diagram is shown below.

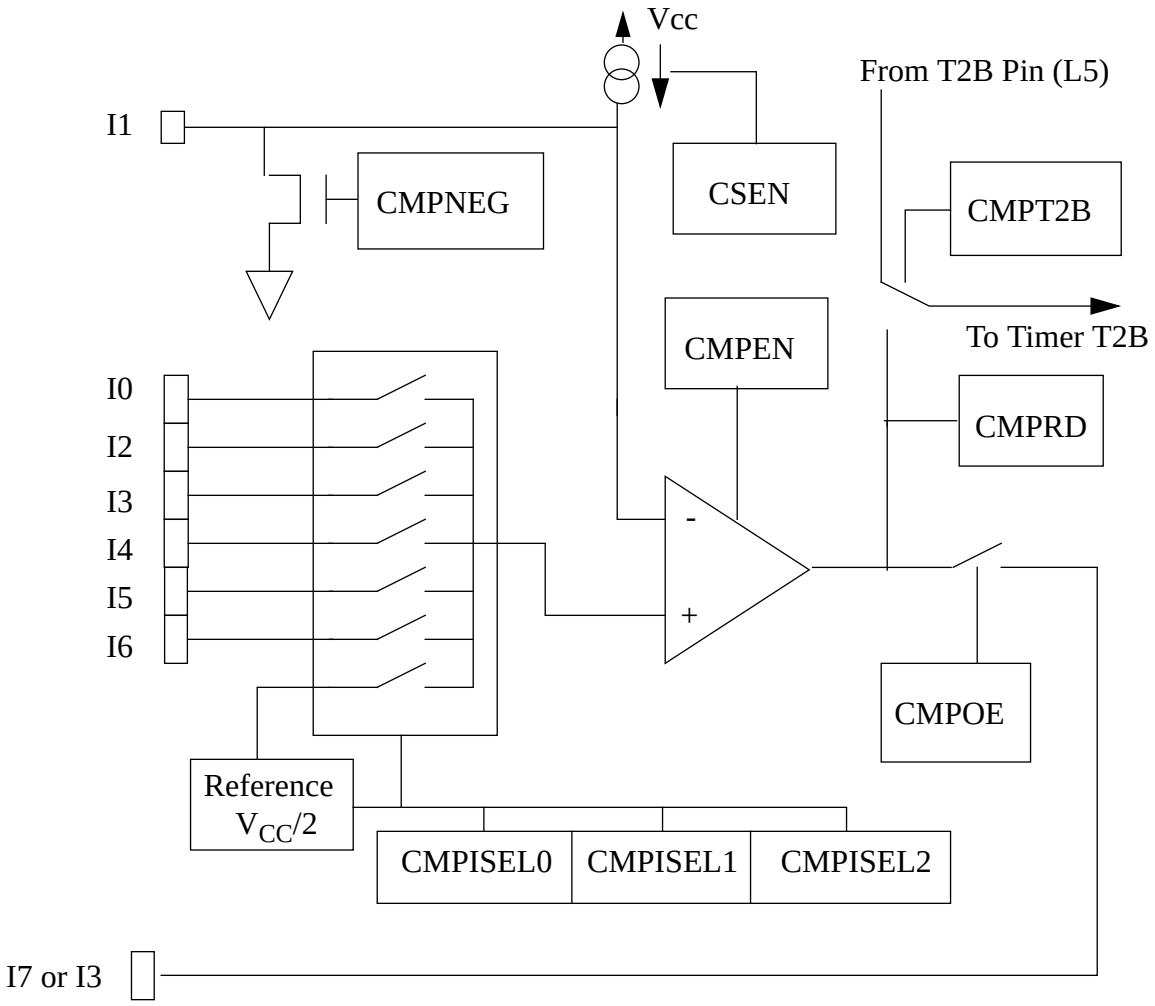


Figure 18-3 COP888EK Analog Function Block

The Comparator Select Register is cleared on RESET (the comparator is disabled). To save power the program should also disable the comparator before the microcontroller enters the HALT/IDLE mode. Disabling the comparator turns off the constant current source and the Vcc/2 reference, disconnects the comparator output from the T2B input and pin I3 or I7, and removes the low on I1 caused by CMPNEG.

It is often useful for the application program to read the result of a comparator operation. Since I1 is always selected to be COMPIN- when the comparator is enabled (CMPEN=1), the comparator output can be read internally by reading bit 1 (CMPRD) of register PORTI (RAM address 0xD7).

The following table lists the comparator inputs and outputs vs. the value of the CMPISEL0/1/2 bits. The output is only driven if the CMPOE bit is set to 1.

Table 18-9 Comparator Input Selection

Control Bit			Comparator Input Source		Comparator Output
CMPISEL 2	**CMPISEL 1**	**CMPISEL 0**	**Neg. Input**	**Pos. Input**	
0	0	0	I1	I2	I3
0	0	1	I1	I2	I7
0	1	0	I1	I3	I7
0	1	1	I1	I0	I7
1	0	0	I1	I4	I7
1	0	1	I1	I5	I7
1	1	0	I1	I6	I7
1	1	1	I1	Vcc/2 Ref.	I7

RESET

The state of the analog function block immediately after RESET is as follows:

a. The CMPSL Register is set to all zeros.

b. The Comparator is disabled.

c. The Constant Current Source is disabled.

d. CMPNEG is turned off.

e. The Port I inputs are electrically isolated from the comparator.

f. The T2B input is selected by the T2CNTRL Register normally.

g. CMPISEL0-CMPISEL2 are set to zero.

h. All Port I inputs are selected to the default digital input mode.

The comparator outputs have the same specification as Ports L and G except that the rise and fall times are symmetrical. The following table shows the DC and AC characteristics for the comparator.

.

Table 18-10 Analog Function Block Electrical Characteristics (Vcc=5V)

Parameter	Conditions	Min	Typ	Max	Units
Input Offset Voltage	0.4 <Vin<Vcc-1.5		10	25	mV
Input Common Mode Voltage Range		0.4		Vcc-1.5	V
Voltage Gain			300k		V/V
Low Level Output Current	Vol=0.4V	1.6			mA
High Level Output Current	Voh=4.6V	1.6			mA
Vcc/2 reference	4.0V<Vcc<6.0V	0.5Vcc-0.04	0.5Vcc	0.5Vcc +0.04	V
DC Supply Current For Comparator (when enabled)	Vcc=6.0V			250	µA
DC Supply Current For Vcc/2 reference (when enabled)	Vcc=6.0V		50	80	µA
DC Supply Current For Constant Current Source (when enabled)	Vcc=6.0V			200	µA
Constant Current Source	4.0V<Vcc<6.0V	10	20	40	µA
Current Source Variation	4.0V<Vcc<6.0V temp = constant			2	µA
Current Source Enable Time			1.5	2	uS
Comparator Response Time	10mV overdrive, 100pF load			1	uS

18.10 MEMORY MAP

Table 18-11 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888EK registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 18-11 Data Memory Map

Address	Contents
0000 to 006F	On-Chip RAM bytes (112 bytes)
0070 to 007F	Unused RAM Address Space (Reads As All Ones)
xx80 to xxAF	Unused RAM Address Space (Reads Undefined Data)
xxB0	Timer T3 Lower Byte
xxB1	Timer T3 Upper Byte
xxB2	Timer T3 Autoload Register T3RA Lower Byte
xxB3	Timer T3 Autoload Register T3RA Upper Byte
xxB4	Timer T3 Autoload Register T3RB Lower Byte
xxB5	Timer T3 Autoload Register T3RB Upper Byte
xxB6	Timer T3 Control Register
xxB7	Comparator Select Register (CMPSL)
xxB8 to xxBF	Reserved
xxC0	Timer T2 Lower Byte
xxC1	Timer T2 Upper Byte
xxC2	Timer T2 Autoload Register T2RA Lower Byte
xxC3	Timer T2 Autoload Register T2RA Upper Byte
xxC4	Timer T2 Autoload Register T2RB Lower Byte
xxC5	Timer T2 Autoload Register T2RB Upper Byte
xxC6	Timer T2 Control Register
xxC7	WATCHDOG Service Register (Reg:WDSVR)
xxC8	MIWU Edge Select Register (Reg:WKEDG)
xxC9	MIWU Enable Register (Reg:WKEN)
xxCA	MIWU Pending Register (Reg:WKPND)
xxCB	Reserved
xxCC	Reserved

Table 18-11 Data Memory Map

Address	Contents
xxCD to xxCF	Reserved
xxD0	Port L Data Register
xxD1	Port L Configuration Register
xxD2	Port L Input Pins (Read Only)
xxD3	Reserved for Port L
xxD4	Port G Data Register
xxD5	Port G Configuration Register
xxD6	Port G Input Pins (Read Only)
xxD7	Port I Input Pins (Read Only)
xxD8	Port C Data Register
xxD9	Port C Configuration Register
xxDA	Port C Input Pins (Read Only)
xxDB	Reserved for Port C
xxDC	Port D
xxDD to DF	Reserved for Port D
xxE0 to xxE5	Reserved for EE Control Registers
xxE6	Timer T1 Autoload Register T1RB Lower Byte
xxE7	Timer T1 Autoload Register T1RB Upper Byte
xxE8	ICNTRL Register
xxE9	MICROWIRE/PLUS Shift Register
xxEA	Timer T1 Lower Byte
xxEB	Timer T1 Upper Byte
xxEC	Timer T1 Autoload Register T1RA Lower Byte
xxED	Timer T1 Autoload Register T1RA Upper Byte
xxEE	CNTRL Control Register
xxEF	PSW Register
xxF0 to FB	On-Chip RAM Mapped as Registers
xxFC	X Register
xxFD	SP Register

Table 18-11 Data Memory Map

Address	Contents
xxFE	B Register
xxFF	S Register
0100--017F	On-Chip 128 RAM Bytes
Reading memory locations 0070H–007FH (Segment 0) will return all ones. Reading unused memory locations 0080H–00AFH (Segment 0) will return undefined data. Reading memory locations from other Segments (i.e., Segment 2, Segment 3, ... etc.) will return all ones.	

18.11 RESET

Upon reset of the COP888EK, the ports and registers are initialized as follows:

Port L: TRISTATE

Port C: TRISTATE

Port G: TRISTATE

Port D: HIGH

PC: CLEARED

PSW, CNTRL and ICNTRL registers: CLEARED

SIOR:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

T1CNTRL, T2CNTRL and T3CNTRL: CLEARED

CMPSL: CLEARED

Accumulator, Timer 1, Timer 2, and Timer3:

RANDOM after RESET with crystal clock option (power already applied)

UNAFFECTED after RESET with RC clock option (power already applied)

RANDOM after RESET at power-on

WKEN, WKEDG: CLEARED

WKPND: RANDOM

S Register: CLEARED

SP (Stack Pointer): Loaded with 6F Hex

B and X Pointers:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

RAM:

UNAFFECTED after RESET with power already applied

RANDOM after RESET at power-on

The device comes out of reset with both the WATCHDOG logic and the Clock Monitor detector armed, with the WATCHDOG service window bits set and the Clock Monitor bit set. The WATCHDOG and Clock Monitor circuits are inhibited during reset. The WATCHDOG service window bits being initialized high default to the maximum WATCHDOG service window of 64k t_C clock cycles. The Clock Monitor bit being initialized high will cause a Clock Monitor error following reset if the clock has not reached the minimum specified frequency at the termination of reset. A Clock Monitor error will cause an active low error output on pin G1. This error output will continue until 16 tC–32 t_C clock cycles following the clock frequency reaching the minimum specified value, at which time the G1 output will enter the TRI-STATE mode.

18.12 INTERRUPTS

Table 18-12 shows the types of interrupts in the COP888EK, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 18-12 COP888EK Interrupt Rank and Vector Address

Arbitration Rank	Interrupt Source	Description	Vector Address[a]
1	Software Trap	INTR Instruction	0yFE - 0yFF
2	Reserved	for Future Use	0yFC - 0yFD
3	Pin G0 Edge	External	0yFA - 0yFB
4	Timer T0	IDLE Timer Underflow	0yF8 - 0yF9
5	Timer T1	Timer T1A/Underflow	0yF6 - 0yF7
6	Timer T1	Timer T1B	0yF4 - 0yF5
7	MICROWIRE/PLUS	BUSY Goes Low	0yF2 - 0yF3
8	Reserved	for Future Use	0yF0-0yF1
9	Reserved	for Future Use	0yEE - 0yEF
10	Reserved	for Future Use	0yEC-0yED
11	Timer T2	T2A/Underflow	0yEA - 0yEB
12	Timer T2	T2B	0yE8 - 0yE9
13	Timer T3	T3A/Underflow	0yE6 - 0yE7
14	Timer T3	T3B	0yE4 - 0yE5

Table 18-12 COP888EK Interrupt Rank and Vector Address

Arbitration Rank	Interrupt Source	Description	Vector Address[a]
15	Port L/Wakeup	Port L Edge	0yE2 - 0yE3
16	Default VIS	VIS Interrupt	0yE0 - 0yE1

a. The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in the table assume a VIS instruction between 00FF and 01DF Hex.

18.13 MASK OPTIONS

The mask options for this device are described below. These options are programmed at the same time as the ROM pattern and therefore must be submitted with the ROM pattern.

OPTION 1: Clock Configuration

=1 Crystal Oscillator (CKI/10)

G7 (CKO) is clock generator output to crystal/resonator

CKI is the clock input

=2 Single-pin R/C Controlled Oscillator

G7 is available as a HALT restart and/or general purpose input

CKI is clock input

OPTION 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Chapter 19

COP888FH

19.1 INTRODUCTION

The COP888FH is a member of the COP888 Feature Family 8-bit microcontrollers. Like other members of this family, this device provides high-performance, economical solutions for embedded control applications. The COP888FH contains a Universal Asynchronous Receiver/Transmitter (UART), one or two comparators, a hardware multiply/divide, 12K bytes of ROM, and 512 bytes of RAM.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888FH.

- Program Memory: Mask-programmed ROM, 12K bytes
- Data Memory: Static RAM, 512 bytes
- S-Register Data Memory Extension: Yes
- 16-Bit Programmable Timers: Timer T1, T2, and T3
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- Hardware Multiply/Divide: Yes
- WATCHDOG and Clock Monitor: Yes
- NMI Interrupt: No
- Additional features: UART, comparator(s) and Reduced Electromagnetic Emissions

Several versions of the COP888FH are available that operate over different voltage and temperature ranges. Refer to the appropriate data sheet for more specific information.

This chapter describes the device-specific features of the COP888FH. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

19.2 BLOCK DIAGRAM

Figure 19-1 is a block diagram showing the basic functional blocks of the COP888FH. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers. Various functional blocks of the COP888 device communicate with the core through an internal bus.

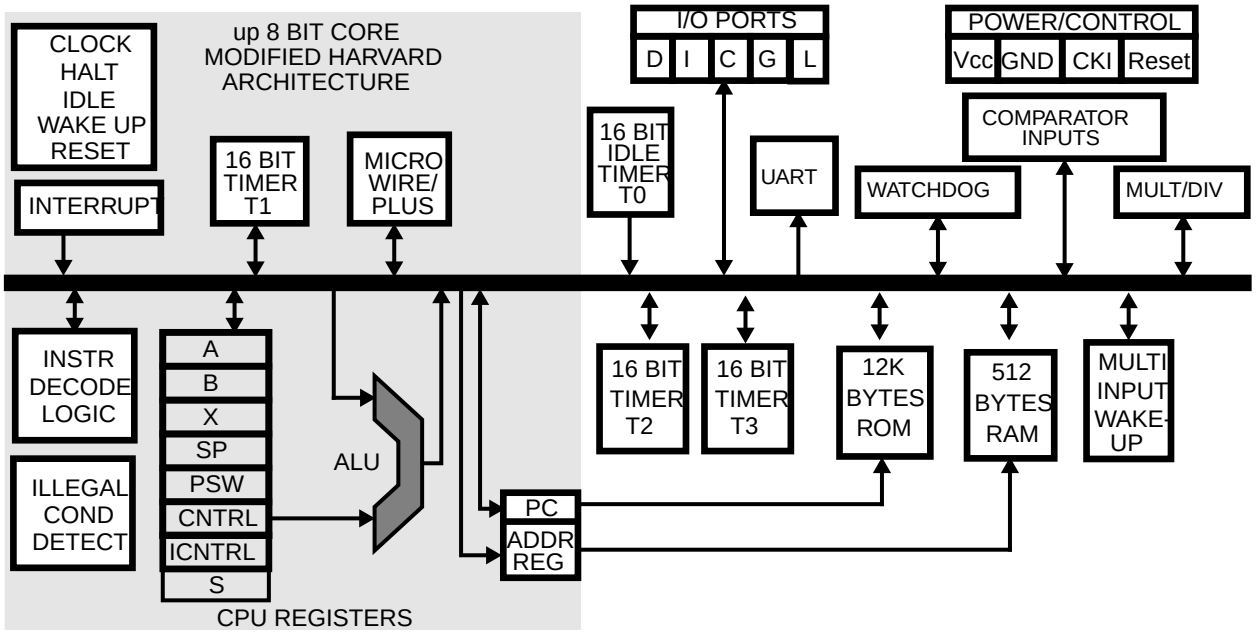


Figure 19-1 COP888FH Block Diagram

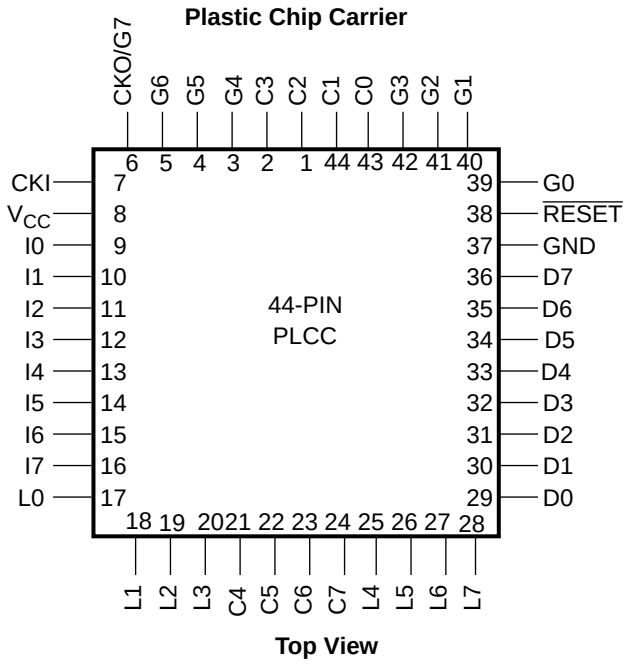
19.3 DEVICE PINOUTS/PACKAGES

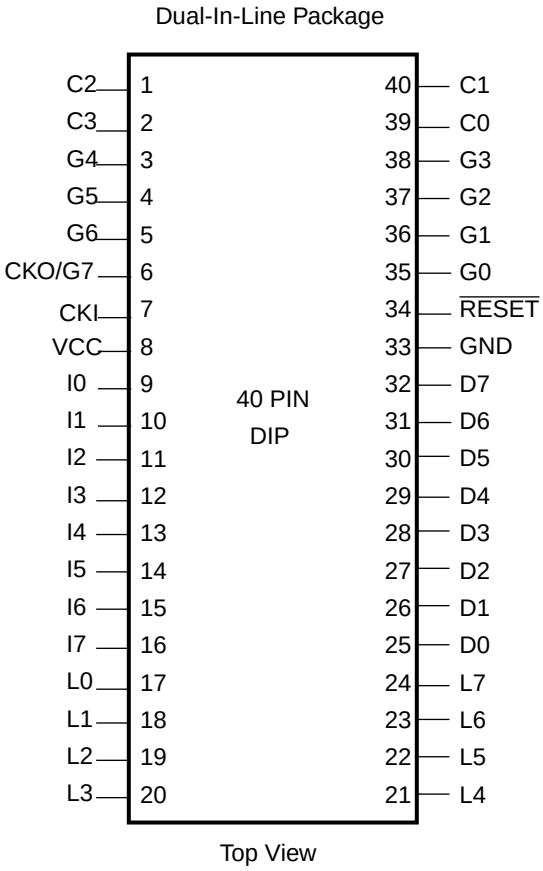
The COP888FH is available in 28-pin DIP, 28-pin SO, 40-pin DIP and 44-pin PLCC packages. Figure 19-2 shows the COP888FH device package pinouts.

Refer to the COP888FH data sheet for more detailed package information.

19.4 PIN DESCRIPTIONS

The COP888FH has four dedicated function pins: V_{CC}, GND, CKI, and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 19-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number for the available package types.





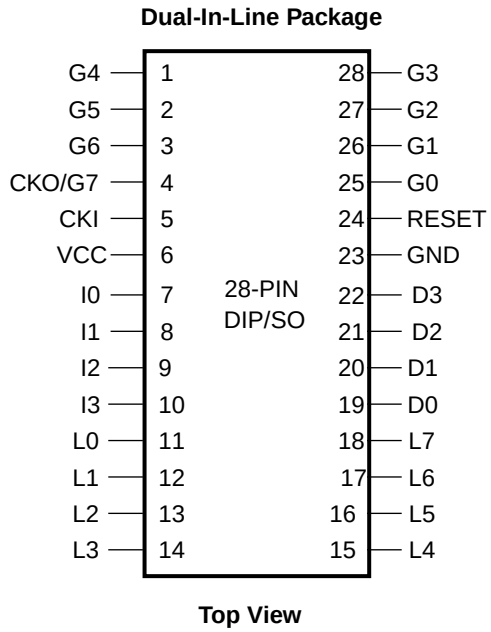


Figure 19-2 Device Package Pinouts

Table 19-1 COP888FH Pinouts

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
L0	I/O	MIWU	11	17	17
L1	I/O	MIWU or CKX	12	18	18
L2	I/O	MIWU or TDX	13	19	19
L3	I/O	MIWU or RDX	14	20	20
L4	I/O	MIWU or T2A	15	21	25
L5	I/O	MIWU or T2B	16	22	26
L6	I/O	MIWU or T3A	17	23	27
L7	I/O	MIWU or T3B	18	24	28
G0	I/O	INT	25	35	39
G1	WDOUT		26	36	40
G2	I/O	T1B	27	37	41
G3	I/O	T1A	28	38	42
G4	I/O	SO	1	3	3
G5	I/O	SK	2	4	4
G6	I	SI	3	5	5
G7	I/CKO	HALT RESTART	4	6	6
I0	I		7	9	9
I1	I	COMP1IN–	8	10	10
I2	I	COMP1IN+	9	11	11
I3	I	COMP1OUT	10	12	12
I4	I	COMP2IN–		13	13
I5	I	COMP2IN+		14	14
I6	I	COMP2OUT		15	15
I7	I			16	16
D0	O		19	25	29
D1	O		20	26	30
D2	O		21	27	31
D3	O		22	28	32

Table 19-1 COP888FH Pinouts (Continued)

Pin Name	Type	Alternate Function	28-Pin DIP/SO	40-Pin DIP	44-Pin PLCC
D4	O			29	33
D5	O			30	34
D6	O			31	35
D7	O			32	36
C0	I/O			39	43
C1	I/O			40	44
C2	I/O			1	1
C3	I/O			2	2
C4	I/O				21
C5	I/O				22
C6	I/O				23
C7	I/O				24
V_{CC}			6	8	8
GND			23	33	37
CKI			5	7	7
$\overline{\text{RESET}}$			24	34	38

19.5 INPUT/OUTPUT PORTS

The number and types of I/O port pins available on the COP888FH depend on the package type. The 28-pin packages have Port L, Port G, and a subset of Port D and Port I. The 40-pin packages have Port L, Port G, Port D, Port I, and a subset Port C. The 44-pin packages have a complete set of ports, including all bits of Port L, Port G, Port D, Port I, and Port C. Refer to Table 19-1 for a detailed list of the available port pins and their alternate functions.

The general I/O port functions are described in Chapter 7. The COP888FH device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate functions of the port pins.

Port C, a general-purpose bidirectional I/O port, is present in COP888FH 44-pin packages and partially available (pins 0-3) in 40-pin packages. There are three memory locations associated with this port: one each for the data register, for the configuration

register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port D, a general-purpose dedicated output port, is present in all COP888FH devices, although the 28-pin package only has pins 0-3 available. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port G, a bidirectional I/O port, is present in all COP888FH devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/So or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G1 is a dedicated WATCHDOG output that can be used to reset the device in the event of a WATCHDOG or Clock Monitor error. This pin can not be used a general purpose input or output, and should be left unconnected if not used to signal an error condition. Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I, a general-purpose dedicated input port, is present in all COP888FH devices, although not all eight pins are available in the 28-pin packages. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware. The alternate functions of pins I1 though I6 are the comparator inputs and outputs. Comparator 2 is not available in 28-pin packages because the necessary Port I pins are not present in those packages.

Port L, a general-purpose bidirectional I/O port, is present in all COP888FH devices. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. This port can be used for general-purpose I/O or for several alternate functions: the Multi-Input Wakeup/Interrupt inputs, the UART, Timer T2, and Timer T3. The port inputs have Schmitt triggers to support the Multi- Input Wakeup/Interrupt feature.

19.6 PROGRAM MEMORY

The COP888FH contains 12K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space starting at 0000 and ending at 2FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs.

19.7 DATA MEMORY

The COP888FH device contains 512 bytes of static RAM used for data storage. The first 128 bytes of RAM occupy two areas in the lowest 256-byte data memory space: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next 256-byte pages of data memory space, starting at 0x00 Hex and ending at 0x3F Hex. This higher part of RAM is accessed by writing the value 0x (x = 1 thru 3) to the S register, as explained in “Data Segment Extension” on page 6.

19.8 REGISTER BIT MAPS

The COP888FH has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 19-3 through 19-14 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), WATCHDOG and Clock Monitor (Chapter 8), Multi-Input Wakeup/ Interrupt and Timer T0 (Chapter 6), UART (Chapter 9), or Comparator (Section 19.12).

Table 19-2 Hardware Multiply/Divide Control Register, MDCR (Address xx9D)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved					DIVOVF	DIV	MULT
DIVOVF:	Start Multiply bit (set bit to 1 to start)						
DIV:	Start Divide bit (set bit to 1 to start)						
MULT:	Division Overflow (1 = 16-bit overflow or divide-by-zero error, 0 = no error)						

.

Table 19-3 T3CNTRL, Timer T3 Control Register (Address xxB6)*

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T3C3	T3C2	T3C1	T3C0	T3PNDA	T3ENA	T3PNDB	T3ENB
T3C3-T3C2-T3C1-T3C0: Timer T3 control bits							
T3PNDA: Timer T3 interrupt A pending flag							
T3ENA: Timer T3 interrupt A enable bit							
T3PNDB: Timer T3 interrupt B pending flag							
T3ENB: Timer T3 interrupt B enable bit							
*COP888CG and COP888EG/GG/HG/KG only.							

Table 19-4 CMPSL, Comparator Select Register (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	CMP2OE*	CMP2RD*	CMP2EN*	CMP1OE	CMP1RD	CMP1EN	Unused
CMP2OE: Comparator 2 output enable bit (Pin I6)							
CMP2RD: Comparator 2 read-only result bit							
CMP2EN: Comparator 2 enable bit (Pins I4 and I5)							
CMP1OE: Comparator 1 output enable bit (Pin I3)							
CMP1RD: Comparator 1 read-only result bit							
CMP1EN: Comparator 1 enable bit (Pins I1 and I2)							
* Reserved bits in 28-pin devices.							

Table 19-5 ENU, UART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN:	Parity enable bit
PSEL1:	Parity selection bit (with PSEL0)
XBIT9/PSEL0:	Transmit bit 9 / Parity selection bit (with PSEL1)
CHL1-CHL0:	Frame format selection bits (number of data bits)
ERR:	Error flag (data overrun, framing, or parity error)
RBFL:	Receive buffer full flag
TBMT:	Transmit buffer empty flag

Table 19-6 ENUR, UART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	Reserved	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag
FE:	Framing Error flag
PE:	Parity Error flag
RBIT9:	Receive bit 9 (when 9 bits per frame)
ATTN:	Attention mode enable bit
XMTG:	Transmitting; reset at end of last frame
RCVG:	Receiving; reset when RDX goes high

Table 19-7 ENUI, UART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	STP78	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2: Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)

STP78: Last Stop bit length (0 = 1 bit, 1 = 7/8 bit)

ETDX: Enable TDX transmit pin

SSEL: Synchronous select bit (0 = asynchronous, 1 = synchronous)

XRCLK: External Receive Clock (0 = internal, 1 = external clock)

XTCLK: External Transmit Clock (0 = internal, 1 = external clock)

ERI: Enable Receive Interrupt

ETI: Enable Transmit Interrupt

Table 19-8 BAUD, UART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0

BRD7–BRD0: Baud rate divisor (with PSR register)

Table 19-9 PSR, UART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8

PSR4–PSR0: Baud rate prescaler

BRD10–BRD8: Baud rate divisor (with BAUD register)

Table 19-10 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB

T2C3-T2C2-T2C1-T2C0: Timer T2 control bits

T2PNDA: Timer T2 interrupt A pending flag

T2ENA: Timer T2 interrupt A enable bit

T2PNDB: Timer T2 interrupt B pending flag

T2ENB: Timer T2 interrupt B enable bit

Table 19-11 WDSVR, WATCHDOG Service Register (Address xxC7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WS1	WS0	KEY4	KEY3	KEY2	KEY1	KEY0	CMEN

WS1–WS0: WATCHDOG Window Select bits

KEY4–KEY0: WATCHDOG Key Data (01100)

CMEN: Clock Monitor Enable bit

Table 19-12 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN: Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)

T0PND: Timer T0 (IDLE Timer) interrupt pending flag

T0EN: Timer T0 (IDLE Timer) interrupt enable bit

µWPND: MICROWIRE interrupt pending flag

µWEN: MICROWIRE interrupt enable bit

T1PNDB: Timer T1 interrupt B pending flag

T1ENB: Timer T1 interrupt B enable bit

Table 19-13 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits
MSEL:	MICROWIRE Select bit
IEDG:	External Interrupt Edge selection bit
SL1–SL0:	MICROWIRE clock divide-by selection bits

Table 19-14 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC:	Half-Carry bit
C:	Carry bit
T1PNDA:	Timer T1 interrupt A pending flag
T1ENA:	Timer T1 interrupt A enable bit
EXPND:	External interrupt pending flag
BUSY:	MICROWIRE Busy flag
EXEN:	External interrupt enable bit
GIE:	Global Interrupt Enable

19.9 MEMORY MAP

Table 19-15 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888FH registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 19-15 COP888FH Data Memory Map

ADDRESS	CONTENTS
0000–006F	RAM (112 bytes general-purpose data memory)
0070–007F	Unused address space
xx80–xx97	Unused address space
xx98	Dividend or Result byte, MDR1
xx99	Dividend/Multiplier Result byte, MDR2

Table 19-15 COP888FH Data Memory Map (Continued)

ADDRESS	CONTENTS
xx9A	Dividend/Result byte, MDR3
xx9B	Divisor/Multiplicand or Result byte, MDR4
xx9C	Divisor or Multiplicand byte, MDR5
xx9D	Multiply/Divide Control Register, MDCR
xx9E–xxAF	Unused address space
xxB0	Timer T3 lower byte
xxB1	Timer T3 upper byte
xxB2	Timer T3 autoload register T3RA lower byte
xxB3	Timer T3 autoload register T3RA upper byte
xxB4	Timer T3 autoload register T3RB lower byte
xxB5	Timer T3 autoload register T3RB upper byte
xxB6	Timer T3 control register, T3CNTRL
xxB7	Comparator select register, CMPSL
xxB8	UART transmit buffer, TBUF
xxB9	UART receive buffer, RBUF
xxBA	UART control and status register, ENU
xxBB	UART receive control and status register, ENUR
xxBC	UART interrupt and clock source register, ENUI
xxBD	UART baud register, BAUD
xxBE	UART prescaler select register, PSR
xxBF	Reserved
xxC0	Timer T2 lower byte
xxC1	Timer T2 upper byte
xxC2	Timer T2 autoload register T2RA lower byte
xxC3	Timer T2 autoload register T2RA upper byte
xxC4	Timer T2 autoload register T2RB lower byte
xxC5	Timer T2 autoload register T2RB upper byte
xxC6	Timer T2 control register, T2CNTRL
xxC7	WATCHDOG service register, WDSVR

Table 19-15 COP888FH Data Memory Map (Continued)

ADDRESS	CONTENTS
xxC8	Multi-Input Wakeup edge select register, WKEDG
xxC9	Multi-Input Wakeup enable register, WKEN
xxCA	Multi-Input Wakeup interrupt pending register, WKPND
xxCB–xxCF	Reserved
xxD0	Port L data register, PORTLD
xxD1	Port L configuration register, PORTLC
xxD2	Port L input pins (read only), PORTCP
xxD3	Reserved
xxD4	Port G data register, PORTGD
xxD5	Port G configuration register, PORTGC
xxD6	Port G input pins (read only), PORTGP
xxD7	Port I input pins (read only), PORTI
xxD8	Port C data register, PORTCD
xxD9	Port C configuration register, PORTCC
xxDA	Port C input pins (read only), PORTCP
xxDB	Reserved
xxDC	Port D data register, PORTD
xxDD–xxDF	Reserved
xxE0–xxE5	Reserved
xxE6	Timer T1 autoload register T1RB lower byte
xxE7	Timer T1 autoload register T1RB upper byte
xxE8	Interrupt control register, ICNTRL
xxE9	MICROWIRE serial I/O shift register, SIO
xxEA	Timer T1 lower byte
xxEB	Timer T1 upper byte
xxEC	Timer T1 autoload register T1RA lower byte
xxED	Timer T1 autoload register T1RA upper byte
xxEE	Control register, CNTRL
xxEF	Processor Status Word register, PSW

Table 19-15 COP888FH Data Memory Map (Continued)

ADDRESS	CONTENTS
xxF0–xxFB	General-purpose memory-mapped registers
xxFC	X pointer register
xxFD	SP (Stack Pointer) register
xxFE	B pointer register
xxFF	S (data segment extension) register
0100–017F	RAM, additional 128 bytes)
0200–027F	RAM, additional 128 bytes)
0300–037F	RAM, additional 128 bytes)

The COP888FH contains general-purpose RAM at addresses above 00FF Hex. The S register is used for designating the high-order byte of the RAM address. Registers and memory residing in the top half of the 256-byte memory space (80-FF Hex) can always be accessed, regardless of the value in the S register. For more information on this subject, see “Data Segment Extension” on page 6.

19.10 RESET

Upon reset of the COP888FH, the ports and registers are initialized as follows:

Port C data register, PORTCD:	00
Port C configuration register, PORTCC:	00
Port D data register, PORTD:	FF
Port G data register, PORTGD:	00
Port G configuration register, PORTGC:	00
Port L data register, PORTLD:	00
Port L configuration register, PORTLC:	00
Processor Status Word, PSW:	00
Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.

Timer T2 control register, T2CNTRL:	00
Timer T3 control register, T3CNTRL:	00
Timer T1, T2 and T3 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator and Timers T1, T2, T3:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
S register:	00
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
WATCHDOG service register, WDSVR:	D9
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
Comparator select register, CMPSL:	00
UART control and status register, ENU:	01
UART receive control and status register, ENUR:	00
UART interrupt and clock source register, ENUI:	00
UART baud register, BAUD:	Upon power-up reset, unknown. Upon external reset, unchanged.
UART prescaler select register, PSR:	00
Multiply/Divide Control Register, MDCR:	00

19.11 INTERRUPTS

Table 19-16 shows the types of interrupts in the COP888FH, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

Table 19-16 COP888FH Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	(Reserved)	01F0–01F1
9	UART Receive	01EE–01EF
10	UART Transmit	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	Timer T3A/Underflow	01E6–01E7
14	Timer T3B	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1

* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.

19.12 COMPARATOR

Each COP888FH device has at least one on-chip comparator. A comparator is a circuit that compares the analog input voltages applied to two input pins, and determines which voltage is higher. A memory mapped-register is used for controlling the comparator and for reading the results. The results can also be placed on a device output, if desired.

Every COP888FH device has one comparator, designated Comparator 1. Each COP888FH device supplied in a 40-pin or 44-pin package has an additional comparator, designated Comparator 2.

Two input pins are associated with each comparator, called the negative and positive inputs. If the voltage on the positive input is higher than the voltage on the negative pin, then the output result is “true” or logic 1. If the voltage on the negative input is higher, then the result is “false” or logic 0. The current result of a comparison is written to the read-only result bit of the comparator select register. If the comparator's output pin is enabled, the current result of the comparison is placed on an output pin as a logic 1 or 0, making the comparison result available to the external circuit.

Once enabled, the comparator operates continuously until stopped by the software. The response time is very fast, typically less than the instruction cycle time. In order to operate properly, the input voltages must be within a specific range, slightly smaller than the range from GND to V_{CC}. See the device data sheet for specific values.

The comparator inputs and outputs are alternate functions of Port I. If the comparator is enabled, the two Port I pins that serve as the comparator inputs will operate in that capacity rather than as general-purpose port pins. If the comparator output is enabled along with the comparator, a third Port I pin operates as an output, indicating the comparison result as a logic 1 (high) or logic 0 (low). The Port I pins and their alternate functions are listed in Table 19-17.

Table 19-17 Port I Alternate Functions

Port I Pin	Alternate Function
I0	None
I1	COMP1IN- (Comparator 1 negative input)
I2	COMP1IN+ (Comparator 1 positive input)
I3	COMP1OUT (Comparator 1 output)
I4	COMP2IN- (Comparator 2 negative input)
I5	COMP2IN+ (Comparator 2 positive input)
I6	COMP2OUT (Comparator 2 output)
I7	None

The comparators are controlled by a single memory-mapped register called the CMPSL (Comparator Select) register, located at address xxB7 Hex. Three bits in the register are used for each comparator: a comparator enable bit, a comparator-output enable bit, and a read-only result bit. Upon reset, all bits are cleared and the comparators are disabled. Setting the comparator enable bit enables operation of the comparator and its two inputs. Setting the comparator-output enable bit causes the results of the comparison to be placed on the output pin. Whether or not the output pin is enabled, the comparison results can be read from the result bit, which is a read-only bit.

The CMPSL register bit map is shown in Table 19-18.

Table 19-18 CMPSL, Comparator Select Register (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	CMP2OE	CMP2RD	CMP2EN	CMP1OE	CMP1RD	CMP1EN	Unused

CMP2OE: Comparator 2 output enable bit (Pin I6)

CMP2RD: Comparator 2 read-only result bit

CMP2EN: Comparator 2 enable bit (Pins I4 and I5)

CMP1OE: Comparator 1 output enable bit (Pin I3)

CMP1RD: Comparator 1 read-only result bit

CMP1EN: Comparator 1 enable bit (Pins I1 and I2)

If a comparator is not enabled, the corresponding result bit is read as 0. When writing to the CMPSL register, it does not matter what bit values are written to the two read-only result bit positions. The two “unused” bits can serve as read/write software flags, if desired.

To reduce power consumption, the comparators should be disabled prior to entering the HALT or IDLE mode.

19.13 UART

Each COP888FH device has an on-chip Universal Asynchronous Receiver Transmitter (UART), which can be used for transmitting and receiving data serially, either synchronously or asynchronously. This full-duplex, double-buffered UART is fully programmable, and can be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (7/8, 1, 1-7/8, or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)
- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability

- Error detection circuitry and diagnostic self-test capability

Information on the UART is provided in <tag-Chapter 9>.

19.14 HARDWARE MULTIPLY/DIVIDE

The COP888GW has a hardware multiply/divide block that supports the following operations:

- Multiplication of a 1-byte value by a 2-byte value, producing a 3-byte result
- Division of a 3-byte value by a 2-byte value, producing a 2-byte result

To carry out a multiply or divide operation, the CPU first places the operands into the appropriate memory-mapped registers, MDR1 through MDR5, and then sets the appropriate control bit in the multiply/divide control register, MDCR. All operands are considered unsigned values.

The operation is carried out very fast, requiring only one instruction cycle for a multiply operation or two instruction cycles for a divide operation. (A division that produces an overflow error or divide-by-zero error requires only one instruction cycle.)

19.14.1 Multiply/Divide Data Registers

The five Hardware Multiply/Divide data registers, MDR1 through MDR5, are described in Table 19-19. For each register, the table shows the register name, the register address, and how the register is used in multiply and divide operations. All operands are considered unsigned values.

Table 19-19 Multiply/Divide Data Registers

Register Name (Address)	Multiplication Assignment		Division Assignment	
	Before Operation	After Operation	Before Operation	After Operation
MDR1 (xx98)	Unused	Unchanged	Low byte of dividend	Low byte of result
MDR2 (xx99)	Multiplier	Low byte of result	Middle byte of dividend	High byte of result
MDR3 (xx9A)		Middle byte of result	High byte of dividend	Undefined
MDR4 (xx9B)	Low byte of multiplicand	High byte of result	Low byte of divisor	Low byte of divisor
MDR5 (xx9C)	High byte of multiplicand	Unchanged	High byte of divisor	High byte of divisor

19.14.2 Multiply/Divide Control Register

The CPU initiates multiply and divide operations by writing to the Multiply/Divide control register, MDCR, at address xx9D. The register format is shown below:

Table 19-20 Multiply/Divide Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved					DIVOVF	DIV	MULT
DIVOVF:	Start Multiply bit (set bit to 1 to start)						
DIV:	Start Divide bit (set bit to 1 to start)						
MULT:	Division Overflow (1 = 16-bit overflow or divide-by-zero error, 0 = no error)						

The MULT and DIV bits are read/write bits and the DIVOVF bit is a read-only bit. Upon completion of any multiply or divide operation, the MULT and DIV bits are both cleared, the DIVOVF is set or cleared (depending on whether an error occurred), and the reserved register bits contain undefined data.

NOTE: While a multiply or divide operation is in progress, registers MDCR and MDR1-MDR5 cannot be accessed. During this time, writing the register has no effect and reading the register returns undefined data.

19.14.3 Multiply Operation

To perform a multiply operation, the CPU loads the 1-byte multiplier into MDR2 and the 2-byte multiplicand into MDR5-MDR4. It then writes the value 01 to the MDCR register to initiate the operation. After one instruction cycle, the 3-byte result is available for reading from MDR5-MDR4-MDR3.

The three bytes of the result are sufficient to handle all combinations of input operands without overflow or loss of precision. The overflow flag (DIVOVF) is always cleared after a multiply operation.

19.14.4 Divide Operation

To perform a divide operation, the CPU loads the 3-byte dividend into MDR3-MDR2-MDR1 and the 2-byte divisor into MDR5-MDR4. It then writes the value 02 to the MDCR register to initiate the operation. After two instruction cycles, the 2-byte result is available for reading from MDR2-MDR1.

NOTE: If the CPU sets both the DIV and MULT bits together, the set MULT bit is ignored and a division is performed. Both bits are cleared upon completion of the division operation.

If the dividend cannot be divided evenly by the divisor, the remainder is discarded. For example, 9 divided by 2 produces the value 4 (in hexadecimal, 000009/0002 = 0004).

If the result of division is larger than 16 bits or if division by zero was attempted, the DIVOVF (division overflow) bit is set in the MDCR register. In that case, the dividend and divisor are left unchanged in the MDR1-MDR5 registers and the operation is completed in one instruction cycle rather than the two normally required.

It is not necessary to read or clear the DIVOVF error flag. A subsequent divide operation overwrites the flag bit with a new result (either 1 or 0). A multiply operation always clears the flag.

19.14.5 Reset State

If a reset signal applied to the device during normal operation, the MDCR register is cleared any operation in progress is stopped. MDR1 through MDR5 are undefined after a reset.

19.15 MASK OPTIONS

The COP888FH has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/10)
G7 (CK) is the clock generator output to a crystal/resonator
CKI is the clock input

=2 External (CKI/10) CKO available as G7 input

=3 Single pin RC controlled oscillator (CKI/10)
G7 is available as a HALT/Restart or general-purpose input

Option 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Option 3: COP888FH Bonding

=1 44-pin PLCC

=2 40-pin DIP

=3 NA

=4 28-pin DIP

=5 28-pin SO

Chapter 20

COP888GW

20.1 INTRODUCTION

The COP888GW is a member of the COP888 Feature Family 8-bit microcontrollers. Like other members of this family, this device provides high-performance, economical solutions for embedded control applications. The COP888GW contains a Universal Asynchronous Receiver/Transmitter (UART), a hardware multiply/divide, two 16-bit input capture modules, four pulse train generators, 16K bytes of ROM, and 512 bytes of RAM.

The types of features available in the COP888 Feature Family are listed below, together with the quantity or availability of each feature in the COP888GW.

- Program Memory: Mask-programmed ROM, 16K bytes
- Data Memory: Static RAM, 512 bytes
- S-Register Data Memory Extension: Yes
- 16-Bit Programmable Timers: Timer T1 and T2
- IDLE Mode and Timer: Yes
- Multi-Input Wakeup/Interrupt: Yes
- Hardware Multiply/Divide: Yes
- Input Capture Modules: Yes
- Pulse Train Generators: Yes
- WATCHDOG and Clock Monitor: No
- Additional features: UART and Reduced Electromagnetic Emissions

The COP888GW is designed to operate at Industrial temperature ranges. Refer to the COP888GW data sheet for more specific information.

This chapter describes the device-specific features of the COP888GW. Information that applies to all COP888 Feature Family members is not provided in this chapter, but is available in the earlier chapters of this manual.

20.2 BLOCK DIAGRAM

Figure 20-1 is a block diagram showing the basic functional blocks of the COP888GW. The CPU core consists of an Arithmetic Logic Unit (ALU) and a set of CPU core registers.

Various functional blocks of the COP888 device communicate with the core through an internal bus.

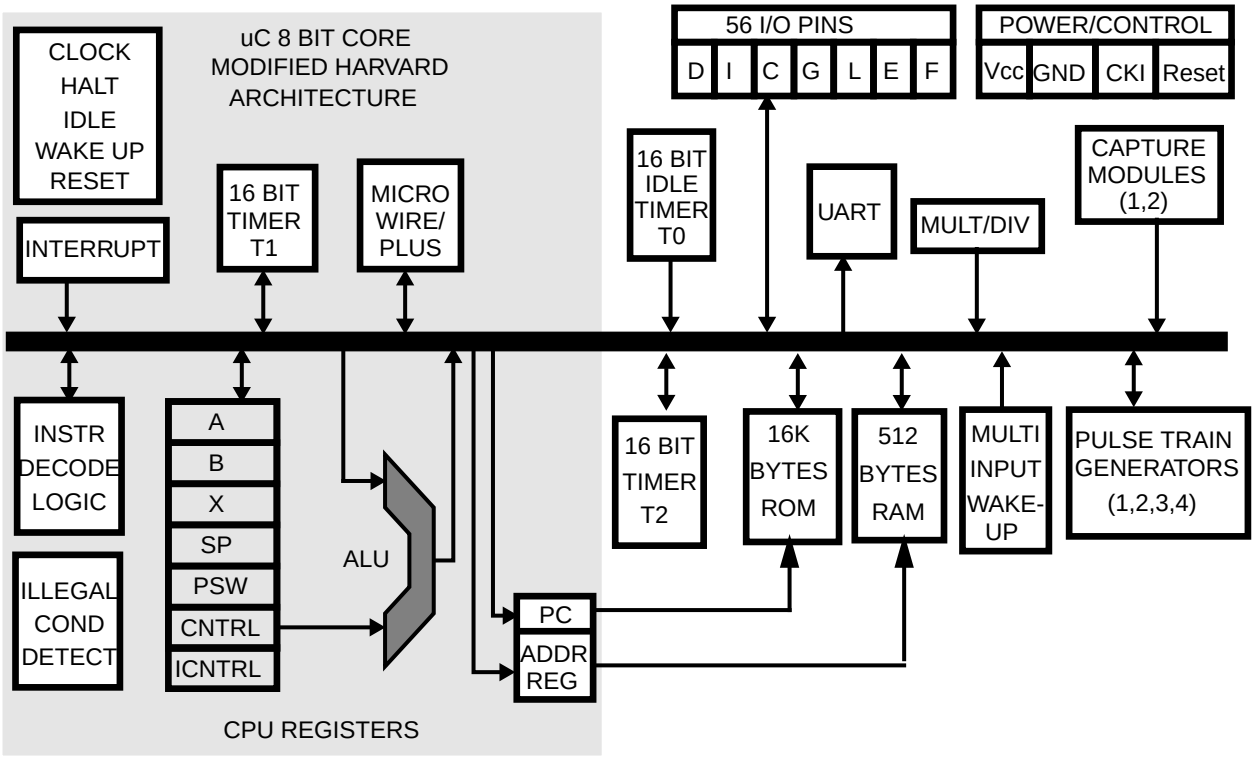


Figure 20-1 COP888GW Block Diagram

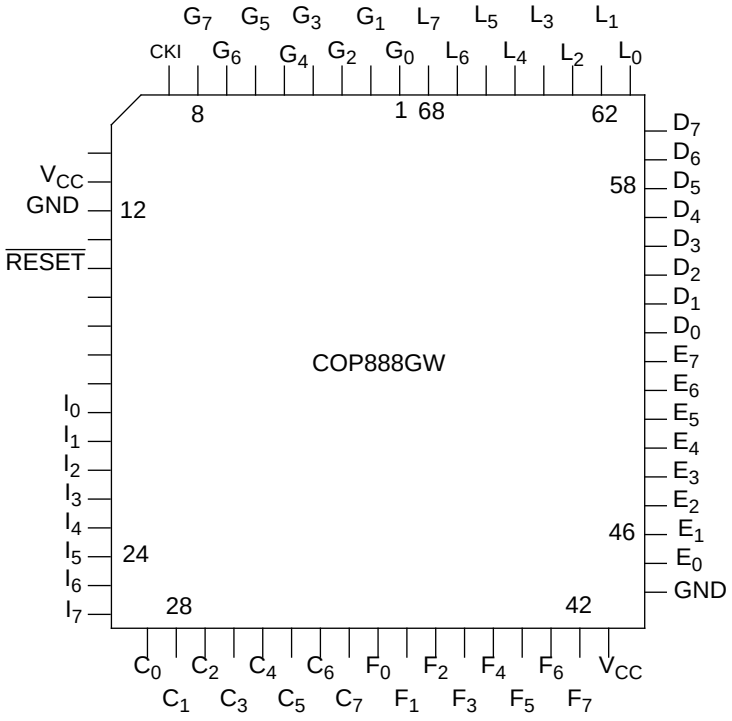
20.3 DEVICE PINOUTS/PACKAGES

The COP888GW is available in a 68-pin PLCC package. Figure 20-2 shows the COP888GW device package pinouts.

Refer to the COP888GW data sheet for more detailed package information.

20.4 PIN DESCRIPTIONS

The COP888GW has dedicated function pins for V_{CC}, GND, CKI, and $\overline{\text{RESET}}$. V_{CC} and GND function as the power supply pins. $\overline{\text{RESET}}$ is used as the master reset input, and CKI is used as a dedicated clock input. All other pins are available as general purpose inputs/outputs or as defined by their alternate functions. For each device pin, Table 20-1 lists the pin name, pin type (Input or Output), alternate function (where applicable), and device pin number.



Top View

Figure 20-2 Device Package Pinout

Table 20-1 COP888GW Pinouts

Pin Name	Type	Alternate Function	66-Pin PLCC
L0	I/O	MIWU	61
L1	I/O	MIWU or CKX	62
L2	I/O	MIWU or TDX	63
L3	I/O	MIWU or RDX	64
L4	I/O	MIWU or T2A	65
L5	I/O	MIWU or T2B	66
L6	I/O	MIWU or CAP1	67
L7	I/O	MIWU or CAP2	68
G0	I/O	INT	1
G1	I/O		2

Table 20-1 COP888GW Pinouts (Continued)

Pin Name	Type	Alternate Function	66-Pin PLCC
G2	I/O	T1B	3
G3	I/O	T1A	4
G4	I/O	SO	5
G5	I/O	SK	6
G6	I	SI	7
G7	I/CKO	HALT RESTART	8
I0	I		19
I1	I		20
I2	I		21
I3	I		22
I4	I		23
I5	I		24
I6	I		25
I7	I		26
D0	O		53
D1	O		54
D2	O		55
D3	O		56
D4	O		57
D5	O		58
D6	O		59
D7	O		60
C0	I/O		27
C1	I/O		28
C2	I/O		29
C3	I/O		30
C4	I/O		31
C5	I/O		32
C6	I/O		33

Table 20-1 COP888GW Pinouts (Continued)

Pin Name	Type	Alternate Function	66-Pin PLCC
C7	I/O		34
F0	I/O		35
F1	I/O		36
F2	I/O		37
F3	I/O		38
F4	I/O		39
F5	I/O		40
F6	I/O		41
F7	I/O		42
E0	I/O	CT1	45
E1	I/O	CT2	46
E2	I/O	CT3	47
E3	I/O	CT4	48
E4	I/O		49
E5	I/O		50
E6	I/O		51
E7	I/O		52
V_{CC}			11, 43*
GND			12, 44*
CKI			9
$\overline{\text{RESET}}$			14

* Both pins must be connected.

20.5 INPUT/OUTPUT PORTS

The general I/O port functions are described in Chapter 7. The COP888GW device-specific port functions and alternate functions are described briefly below. Detailed information on using the port pins can be found in Chapter 7, or in the section describing the specific alternate functions of the port pins.

Port C is a general-purpose bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and

for reading the port pins directly. There are no alternate functions associated with this port.

Port D is a general-purpose dedicated output port. There is one memory location associated with this port, which is used for accessing the port data register. Port D output pins can be individually set to a logic high or low by writing a one or zero, respectively, to the associated data register bits. To avoid placing the device into a special testing mode, the hardware design should ensure that D2 is not pulled low during a reset.

Port E is a general-purpose bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There following alternate functions are associated with this port:

E0 CT1 (Output for Counter 1, Pulse Train Generator)

E1 CT2 (Output for Counter 2, Pulse Train Generator)

E2 CT3 (Output for Counter 3, Pulse Train Generator)

E3 CT4 (Output for Counter 4, Pulse Train Generator)

Port F is a general-purpose bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. There are no alternate functions associated with this port.

Port G is a bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and for reading the port pins directly. All Port G pins have Schmitt triggers on their inputs. Pins G0 and G2 through G5 can be used as general purpose I/Os or for the alternate functions listed below.

G0 INTR (External Interrupt Input)

G2 T1B (Timer T1B Input)

G3 T1A (Timer T1A Input/Output)

G4 SO (MICROWIRE/PLUS Serial Output)

G5 SK (MICROWIRE/PLUS Serial Clock)

Pin G6 can be used as a general purpose input or for the alternate function SI (MICROWIRE/PLUS Serial Input). This pin can not be used as a general purpose output. Pin G7 is a dedicated clock output (CKO) with the crystal oscillator mask option selected. This pin can be used as a general purpose input and HALT/Restart pin with the RC oscillator mask option selected.

Port I is a general-purpose dedicated input port. There is one memory location associated with this port, which is a read-only address used for reading the pin values directly. All Port I pins are high-impedance inputs, which must be pulled to a logic high or low by the external hardware.

Port L is a general-purpose bidirectional I/O port. There are three memory locations associated with this port: one each for the data register, for the configuration register, and

for reading the port pins directly. This port can be used for general-purpose I/O or for several alternate functions: the Multi-Input Wakeup/Interrupt inputs, the UART, and Timer T2 Capture Timer inputs. The port inputs have Schmitt triggers to support the Multi- Input Wakeup/Interrupt feature.

20.6 PROGRAM MEMORY

The COP888GW contains 16K bytes of mask-programmed ROM memory, used for storing application programs and fixed program data. The ROM occupies the program memory address space starting at 0000 and ending at 3FFF hex. It is addressed by the 15-bit Program Counter (PC). Emulation devices are available for developing application programs.

20.7 DATA MEMORY

The COP888GW device contains 512 bytes of static RAM used for data storage. The first 128 bytes of RAM occupy two areas in the lowest 256-byte data memory space: 112 bytes from 00–6F Hex for general-purpose storage, and 16 bytes from F0 to FF Hex for the memory-mapped device registers. The remaining RAM, which is used for general-purpose storage, occupies the bottom of the next 256-byte pages of data memory space, starting at 0x00 Hex and ending at 0x7F Hex. This higher part of RAM is accessed by writing the value 0x (x = 1 thru 3) to the S register, as explained in “Data Segment Extension” on page 6.

20.8 REGISTER BIT MAPS

The COP888GW has several 8-bit memory-mapped registers used for controlling the CPU core, MICROWIRE interface, interrupt interface, timers, and other functions. In some control registers, multiple register bits are grouped together to control a single function, or different control bits within a register are used for unrelated control functions.

Tables 20-2 through 20-15 show the bit maps for these registers. Each bit map shows the name of the register, the register address, the name of each bit in the register, and a brief description of each bit. For detailed information on using individual control bits, refer to the relevant description elsewhere in this manual: CPU core registers (Chapter 2), Timers (Chapter 4), Multi-Input Wakeup/Interrupt and Timer T0 (Chapter 6), or UART (Chapter 9), Capture Timer (Section 20.14), or Pulse Train Generators (Section 20.18).

Table 20-2 Hardware Multiply/Divide Control Register, MDCR (Address xx9D)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved					DIVOVF	DIV	MULT

DIVOVF:	Division Overflow (1 = 16-bit overflow or divide-by-zero error, 0 = no error)
DIV:	Start Divide bit (set bit to 1 to start)
MULT:	Start Multiply bit (set bit to 1 to start)

Table 20-3 Counter Control Register 1, CCR1 (Address xx9E)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C2TM	C2 IPND	C2 IEN	C2 RUN	C1TM	C1 IPND	C1 IEN	C1 RUN

C2TM:	Counter 2 test mode control bit (1 = special test)
C2IPND:	Counter 2 interrupt pending bit (1 = interrupt pending)
C2IEN:	Counter 2 interrupt enable bit (1 = enable)
C2RUN	Counter 2 run bit (1 = start, 0 = stop)
C1TM:	Counter 1 test mode control bit (1 = special test)
C1IPND:	Counter 1 interrupt pending bit (1 = interrupt pending)
C1IEN:	Counter 1 interrupt enable bit (1 = enable)
C1RUN	Counter 1 run bit (1 = start, 0 = stop)

Table 20-4 Counter Control Register 2, CCR2 (Address xx9F)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C4TM	C4 IPND	C4 IEN	C4 RUN	C3TM	C3 IPND	C3 IEN	C3 RUN

C4TM:	Counter 4 test mode control bit (1 = special test)
C4IPND:	Counter 4 interrupt pending bit (1 = interrupt pending)
C4IEN:	Counter 4 interrupt enable bit (1 = enable)
C4RUN	Counter 4 run bit (1 = start, 0 = stop)
C3TM:	Counter 3 test mode control bit (1 = special test)
C3IPND:	Counter 3 interrupt pending bit (1 = interrupt pending)
C3IEN:	Counter 3 interrupt enable bit (1 = enable)
C3RUN	Counter 3 run bit (1 = start, 0 = stop)

Table 20-5 Capture Timer 1 Control Register, CCMR1 (Address xxB6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CM1 TM	unused	unused	CM1 EC	CM1 IP2	CM1 IP1	CM1 IEN	CM1 RUN

CM1TM:	CM1 test mode control bit (1 = special test)
CM1EC:	CM1 active edge control bit (0 = rising, 1 = falling)
CM1IP2:	CM1 interrupt pending bit 2 (1 = CM1 captured)
CM1IP1	CM1 interrupt pending bit 1(1 = CM1 underflowed)
CMI1EN	CM1 interrupt enable bit (1 = enable)
CM1RUN:	CM1 run control bit (1 = start, 0 = stop)

Table 20-6 Capture Timer 2 Control Register, CCMR2 (Address xxB7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CM2 TM	unused	unused	CM2 EC	CM2 IP2	CM2 IP1	CM2 IEN	CM2 RUN

CM2TM:	CM2 test mode control bit (1 = special test)
CM2EC:	CM2 active edge control bit (0 = rising, 1 = falling)
CM2IP2:	CM2 interrupt pending bit 2 (1 = CM2 captured)
CM2IP1	CM2 interrupt pending bit 1(1 = CM2 underflowed)
CM2IEN	CM2 interrupt enable bit (1 = enable)
CM2RUN:	CM2 run control bit (1 = start, 0 = stop)

Table 20-7 ENU, UART Control and Status Register (Address xxBA)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PEN	PSEL1	XBIT9/PSEL0	CHL1	CHL0	ERR	RBFL	TBMT

PEN:	Parity enable bit
PSEL1:	Parity selection bit (with PSEL0)
XBIT9/PSEL0:	Transmit bit 9 / Parity selection bit (with PSEL1)
CHL1-CHL0:	Frame format selection bits (number of data bits)
ERR:	Error flag (data overrun, framing, or parity error)
RBFL:	Receive buffer full flag
TBMT:	Transmit buffer empty flag

Table 20-8 ENUR, UART Receive Control and Status Register (Address xxBB)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOE	FE	PE	Reserved	RBIT9	ATTN	XMTG	RCVG

DOE:	Data Overrun Error flag
FE:	Framing Error flag
PE:	Parity Error flag
RBIT9:	Receive bit 9 (when 9 bits per frame)
ATTN:	Attention mode enable bit
XMTG:	Transmitting; reset at end of last frame
RCVG:	Receiving; reset when RDX goes high

Table 20-9 ENUI, UART Interrupt and Clock Source Register (Address xxBC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STP2	STP78	ETDX	SSEL	XRCLK	XTCLK	ERI	ETI

STP2:	Stop bits (0 = 1 Stop bit, 1 = 2 Stop bits)
STP78:	Last Stop bit length (0 = 1 bit, 1 = 7/8 bit)
ETDX:	Enable TDX transmit pin
SSEL:	Synchronous select bit (0 = asynchronous, 1 = synchronous)
XRCLK:	External Receive Clock (0 = internal, 1 = external clock)
XTCLK:	External Transmit Clock (0 = internal, 1 = external clock)
ERI:	Enable Receive Interrupt
ETI:	Enable Transmit Interrupt

Table 20-10 BAUD, UART Baud Register (Address xxBD)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BRD7	BRD6	BRD5	BRD4	BRD3	BRD2	BRD1	BRD0
BRD7–BRD0: Baud rate divisor (with PSR register)							

Table 20-11 PSR, UART Prescaler Select Register (Address xxBE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PSR4	PSR3	PSR2	PSR1	PSR0	BRD10	BRD9	BRD8
PSR4–PSR0: Baud rate prescaler							
BRD10–BRD8: Baud rate divisor (with BAUD register)							

Table 20-12 T2CNTRL, Timer T2 Control Register (Address xxC6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2C3	T2C2	T2C1	T2C0	T2PNDA	T2ENA	T2PNDB	T2ENB
T2C3-T2C2-T2C1-T2C0: Timer T2 control bits							
T2PNDA: Timer T2 interrupt A pending flag							
T2ENA: Timer T2 interrupt A enable bit							
T2PNDB: Timer T2 interrupt B pending flag							
T2ENB: Timer T2 interrupt B enable bit							

Table 20-13 ICNTRL, Interrupt Control Register (Address xxE8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	LPEN	T0PND	T0EN	µWPND	µWEN	T1PNDB	T1ENB

LPEN:	Port L Interrupt Enable bit (Multi-Input Wakeup/Interrupt)
T0PND:	Timer T0 (IDLE Timer) interrupt pending flag
T0EN:	Timer T0 (IDLE Timer) interrupt enable bit
µWPND:	MICROWIRE interrupt pending flag
µWEN:	MICROWIRE interrupt enable bit
T1PNDB:	Timer T1 interrupt B pending flag
T1ENB:	Timer T1 interrupt B enable bit

Table 20-14 CNTRL, Control Register (Address xxEE)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1C3	T1C2	T1C1	T1C0	MSEL	IEDG	SL1	SL0

T1C3-T1C2-T1C1-T1C0:	Timer T1 control bits
MSEL:	MICROWIRE Select bit
IEDG:	External Interrupt Edge selection bit (0 = rising, 1 = falling)
SL1–SL0:	MICROWIRE clock divide-by selection bits

Table 20-15 PSW, Processor Status Word Register (Address xxEF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HC	C	T1PNDA	T1ENA	EXPND	BUSY	EXEN	GIE

HC:	Half-Carry bit
C:	Carry bit
T1PNDA:	Timer T1 interrupt A pending flag
T1ENA:	Timer T1 interrupt A enable bit
EXPND:	External interrupt pending flag
BUSY:	MICROWIRE Busy flag
EXEN:	External interrupt enable bit
GIE:	Global Interrupt Enable

20.9 MEMORY MAP

Table 20-16 is a memory map showing the organization of the data memory and the specific memory addresses of the COP888GW registers, including all RAM, I/O ports, port registers, and control registers. For purposes of upward compatibility, do not allow the software to access any address that is designated “Reserved” in the table.

Table 20-16 COP888GW Data Memory Map

ADDRESS	CONTENTS
0000–006F	RAM (112 bytes general-purpose data memory)
0070–007F	Unused address space
xx80–xx8F	Unused address space
xx90	Port E data register
xx91	Port E configuration register
xx92	Port E input pins (read only)
xx93	Reserved
xx94	Port F data register
xx95	Port F configuration register
xx96	Port F input pins (read only)
xx97	Reserved
xx98	Dividend or Result byte, MDR1
xx99	Dividend/Multiplier or Result byte, MDR2
xx9A	Dividend/Result byte or undefined, MDR3
xx9B	Divisor/Multiplicand or Result byte, MDR4
xx9C	Divisor or Multiplicand byte, MDR5
xx9D	Multiply/Divide Control Register, MDCR
xx9E	Counter Control Register 1, CCR1
xx9F	Counter Control Register 2, CCR2
xxA0	Counter 1 prescaler lower byte, C1PRL
xxA1	Counter 1 prescaler upper byte, C1PRH
xxA2	Counter 1 count register lower byte, C1CTL
xxA3	Counter 1 count register upper byte, C1CTH
xxA4	Counter 2 prescaler lower byte, C2PRL
xxA5	Counter 2 prescaler upper byte, C2PRH

Table 20-16 COP888GW Data Memory Map (Continued)

ADDRESS	CONTENTS
xxA6	Counter 2 count register lower byte, C2CTL
xxA7	Counter 2 count register upper byte, C2CTH
xxA8	Counter 3 prescaler lower byte, C3PRL
xxA9	Counter 3 prescaler upper byte, C3PRH
xxAA	Counter 3 count register lower byte, C3CTL
xxAB	Counter 3 count register upper byte, C3CTH
xxAC	Counter 4 prescaler lower byte, C4PRL
xxAD	Counter 4 prescaler upper byte, C4PRH
xxAE	Counter 4 count register lower byte, C4CTL
xxAF	Counter 4 count register upper byte, C4CTH
xxB0	Capture Timer 1 prescaler register, CM1PSC
xxB1	Capture Timer 1 lower byte (read-only), CM1CRL
xxB2	Capture Timer 1 upper byte (read-only), CM1CRH
xxB3	Capture Timer 2 prescaler register, CM2PSC
xxB4	Capture Timer 2 lower byte (read-only), CM2CRL
xxB5	Capture Timer 2 upper byte (read-only), CM2CRH
xxB6	Capture Timer 1 Control Register, CCMR1
xxB7	Capture Timer 2 Control Register, CCMR2
xxB8	UART transmit buffer, TBUF
xxB9	UART receive buffer, RBUF
xxBA	UART control and status register, ENU
xxBB	UART receive control and status register, ENUR
xxBC	UART interrupt and clock source register, ENUI
xxBD	UART baud register, BAUD
xxBE	UART prescaler select register, PSR
xxBF	Reserved
xxC0	Timer T2 lower byte
xxC1	Timer T2 upper byte
xxC2	Timer T2 autoload register T2RA lower byte

Table 20-16 COP888GW Data Memory Map (Continued)

ADDRESS	CONTENTS
xxC3	Timer T2 autoload register T2RA upper byte
xxC4	Timer T2 autoload register T2RB lower byte
xxC5	Timer T2 autoload register T2RB upper byte
xxC6	Timer T2 control register, T2CNTRL
xxC7	Reserved
xxC8	Multi-Input Wakeup edge select register, WKEDG
xxC9	Multi-Input Wakeup enable register, WKEN
xxCA	Multi-Input Wakeup interrupt pending register, WKPND
xxCB–xxCF	Reserved
xxD0	Port L data register, PORTLD
xxD1	Port L configuration register, PORTLC
xxD2	Port L input pins (read only), PORTCP
xxD3	Reserved
xxD4	Port G data register, PORTGD
xxD5	Port G configuration register, PORTGC
xxD6	Port G input pins (read only), PORTGP
xxD7	Port I input pins (read only), PORTI
xxD8	Port C data register, PORTCD
xxD9	Port C configuration register, PORTCC
xxDA	Port C input pins (read only), PORTCP
xxDB	Reserved
xxDC	Port D data register, PORTD
xxDD–xxDF	Reserved
xxE0–xxE5	Reserved
xxE6	Timer T1 autoload register T1RB lower byte
xxE7	Timer T1 autoload register T1RB upper byte
xxE8	Interrupt control register, ICNTRL
xxE9	MICROWIRE serial I/O shift register, SIO
xxEA	Timer T1 lower byte

Table 20-16 COP888GW Data Memory Map (Continued)

ADDRESS	CONTENTS
xxEB	Timer T1 upper byte
xxEC	Timer T1 autoload register T1RA lower byte
xxED	Timer T1 autoload register T1RA upper byte
xxEE	Control register, CNTRL
xxEF	Processor Status Word register, PSW
xxF0–xxFB	General-purpose memory-mapped registers
xxFC	X pointer register
xxFD	SP (Stack Pointer) register
xxFE	B pointer register
xxFF	S (data segment extension) register
0100–017F	RAM, additional 128 bytes)

The COP888GW contains general-purpose RAM at addresses above 00FF Hex. The S register is used for designating the high-order byte of the RAM address. Registers and memory residing in the top half of the 256-byte memory space (80-FF Hex) can always be accessed, regardless of the value in the S register.

20.10 RESET

Upon reset of the COP888GW, the ports and registers are initialized as follows:

Port C data register, PORTCD: 00

Port C configuration register, PORTCC: 00

Port D data register, PORTD: FF

Port E data register, PORTED: 00

Port E configuration register, PORTEC: 00

Port F data register, PORTFD: 00

Port F configuration register, PORTFC: 00

Port G data register, PORTGD: 00

Port G configuration register, PORTGC: 00

Port L data register, PORTLD: 00

Port L configuration register, PORTLC: 00

Processor Status Word, PSW: 00

Control register, CNTRL:	00
Interrupt control register, ICNTRL:	00
Global Interrupt Enable flag, GIE:	Cleared
Software Trap interrupt pending flag, STPND:	Cleared
MICROWIRE shift register, SIOR:	Upon power-up reset, unknown. Upon external reset, unchanged.
Timer T2 control register, T2CNTRL:	00
Timers T1 and T2 reload registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Accumulator and Timers T1, T2:	Upon power-up reset, unknown. Upon external reset, unknown (with crystal oscillator option) or unchanged (with RC oscillator option).
Program Counter, PC:	00
SP (Stack Pointer) register:	6F
B pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
X pointer register:	Upon power-up reset, unknown. Upon external reset, unchanged.
S register:	00
Multi-Input Wakeup edge select register, WKEDG:	00
Multi-Input Wakeup enable register, WKEN:	00
Multi-Input Wakeup interrupt pending register, WKPND:	Unknown
RAM (other than FC-FF):	Upon power-up reset, unknown. Upon external reset, unchanged.
Comparator select register, CMPSL:	00
UART control and status register, ENU:	01
UART receive control and status register, ENUR:	00
UART interrupt and clock source register, ENUI:	00
UART baud register, BAUD:	Upon power-up reset, unknown. Upon external reset, unchanged.

UART prescaler select register, PSR:	00
Multiply/Divide Control Register, MDCR:	00
Capture Timer Control Registers, CCMR1, CCMR2:	00
Other Capture Timer Registers:	Upon power-up reset, unknown. Upon external reset, unchanged.
Counter Control Registers, CCR1, CCR2	00
Other Counter Control Registers:	Upon power-up reset, unknown. Upon external reset, unchanged.

20.11 INTERRUPTS

Table 20-17 shows the types of interrupts in the COP888GW, the interrupt arbitration ranking, and the locations of the corresponding interrupt vectors in the vector table. For basic information on COP888 interrupts, see Chapter 3.

20.12 UART

Each COP888GW device has an on-chip Universal Asynchronous Receiver Transmitter (UART), which can be used for transmitting and receiving data serially, either synchronously or asynchronously. This full-duplex, double-buffered UART is fully programmable, and can be configured into a wide variety of operating modes. It offers the following major features:

- Full-duplex operation
- Fully programmable serial interface options, including baud rate, start bit, data length (7, 8, or 9 bits), parity bit (even/odd, mark, space, or none), and stop bits (7/8, 1, 1-7/8, or 2)
- Accurate baud rate generation using the chip clock (no additional crystal necessary)
- Complete status reporting
- Separate interrupt vectors for Receive Buffer Full and Transmit Buffer Empty
- Independent clock inputs for the transmit and receive sections (either internal or external)
- Asynchronous or synchronous operation
- Receiver Attention mode for better networking capability
- Error detection circuitry and diagnostic self-test capability

Information on the UART is provided in Chapter 9.

Table 20-17 COP888GW Interrupt Rank and Vector Addresses

Arbitration Rank	Interrupt Description	Vector Address*
1	Software Trap (INTR)	01FE–01FF
2	(Reserved)	01FC–01FD
3	External Interrupt Pin G0	01FA–01FB
4	IDLE Timer Underflow	01F8–01F9
5	Timer T1A/Underflow	01F6–01F7
6	Timer T1B	01F4–01F5
7	MICROWIRE/PLUS	01F2–01F3
8	Pulse Train Counters	01F0–01F1
9	UART Receive	01EE–01EF
10	UART Transmit	01EC–01ED
11	Timer T2A/Underflow	01EA–01EB
12	Timer T2B	01E8–01E9
13	Capture Timers 1 and 2	01E6–01E7
14	Reserved	01E4–01E5
15	Port L / Wakeup	01E2–01E3
16	Default VIS Interrupt	01E0–01E1
* The location of the vector table depends on the location of the VIS instruction. Vector addresses shown in table assume a VIS location between 00FF Hex and 01DF Hex.		

20.13 HARDWARE MULTIPLY/DIVIDE

The COP888GW has a hardware multiply/divide block that supports the following operations:

- Multiplication of a 1-byte value by a 2-byte value, producing a 3-byte result
- Division of a 3-byte value by a 2-byte value, producing a 2-byte result

To carry out a multiply or divide operation, the CPU first places the operands into the appropriate memory-mapped registers, MDR1 through MDR5, and then sets the appropriate control bit in the multiply/divide control register, MDCR. All operands are considered unsigned values.

The operation is carried out very fast, requiring only one instruction cycle for a multiply operation or two instruction cycles for a divide operation. (A division that produces an overflow error or divide-by-zero error requires only one instruction cycle.)

20.13.1 Multiply/Divide Data Registers

The five Hardware Multiply/Divide data registers, MDR1 through MDR5, are described in Table 20-18. For each register, the table shows the register name, the register address, and how the register is used in multiply and divide operations. All operands are considered unsigned values.

Table 20-18 Multiply/Divide Data Registers

Register Name (Address)	Multiplication Assignment		Division Assignment	
	Before Operation	After Operation	Before Operation	After Operation
MDR1 (xx98)	Unused	Unchanged	Low byte of dividend	Low byte of result
MDR2 (xx99)	Multiplier	Low byte of result	Middle byte of dividend	High byte of result
MDR3 (xx9A)		Middle byte of result	High byte of dividend	Undefined
MDR4 (xx9B)	Low byte of multiplicand	High byte of result	Low byte of divisor	Low byte of divisor
MDR5 (xx9C)	High byte of multiplicand	Unchanged	High byte of divisor	High byte of divisor

20.13.2 Multiply/Divide Control Register

The CPU initiates multiply and divide operations by writing to the Multiply/Divide control register, MDCR, at address xx9D. The register format is shown below:

Table 20-19 Multiply/Divide Control Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved					DIVOVF	DIV	MULT
DIVOVF:	Division Overflow (1 = 16-bit overflow or divide-by-zero error, 0 = no error)						
DIV:	Start Divide bit (set bit to 1 to start)						
MULT:	Start Multiply bit (set bit to 1 to start)						

The MULT and DIV bits are read/write bits and the DIVOVF bit is a read-only bit. Upon completion of any multiply or divide operation, the MULT and DIV bits are both cleared,

the DIVOVF is set or cleared (depending on whether an error occurred), and the reserved register bits contain undefined data.

NOTE: While a multiply or divide operation is in progress, registers MDCR and MDR1-MDR5 cannot be accessed. During this time, writing the register has no effect and reading the register returns undefined data.

20.13.3 Multiply Operation

To perform a multiply operation, the CPU loads the 1-byte multiplier into MDR2 and the 2-byte multiplicand into MDR5-MDR4. It then writes the value 01 to the MDCR register to initiate the operation. After one instruction cycle, the 3-byte result is available for reading from MDR5-MDR4-MDR3.

The three bytes of the result are sufficient to handle all combinations of input operands without overflow or loss of precision. The overflow flag (DIVOVF) is always cleared after a multiply operation.

20.13.4 Divide Operation

To perform a divide operation, the CPU loads the 3-byte dividend into MDR3-MDR2-MDR1 and the 2-byte divisor into MDR5-MDR4. It then writes the value 02 to the MDCR register to initiate the operation. After two instruction cycles, the 2-byte result is available for reading from MDR2-MDR1.

NOTE: If the CPU sets both the DIV and MULT bits together, the set MULT bit is ignored and a division is performed. Both bits are cleared upon completion of the division operation.

If the dividend cannot be divided evenly by the divisor, the remainder is discarded. For example, 9 divided by 2 produces the value 4 (in hexadecimal, 000009/0002 = 0004).

If the result of division is larger than 16 bits or if division by zero was attempted, the DIVOVF (division overflow) bit is set in the MDCR register. In that case, the dividend and divisor are left unchanged in the MDR1-MDR5 registers and the operation is completed in one instruction cycle rather than the two normally required.

It is not necessary to read or clear the DIVOVF error flag. A subsequent divide operation overwrites the flag bit with a new result (either 1 or 0). A multiply operation always clears the flag.

20.13.5 Reset State

If a reset signal applied to the device during normal operation, the MDCR register is cleared any operation in progress is stopped. MDR1 through MDR5 are undefined after a reset.

20.14 CAPTURE TIMERS

The COP888GW contains two independent capture timers, called Capture Timer 1 and Capture Timer 2, which can be used to measure the time intervals between external events. The capture inputs are alternate functions of port pins L6 and L7, respectively. Each capture timer contains an 8-bit programmable prescaler register, a 16-bit down counter, a 16-bit input capture register, and capture edge-sensitive logic.

The 16-bit down counter is clocked at a frequency determined by the value loaded into the prescaler register. An active edge transition (programmed to be either a positive or negative edge) on the capture input causes the contents of the down counter to be latched into the capture register.

The values captured in the register reflect the elapsed time between the two successive transitions on the capture input. The time between a positive and negative edge (a pulse width) can be measured by switching the active edge type from positive to negative after the initial positive edge is captured.

Each capture timer can be stopped and started under software control, and each can be configured to interrupt the CPU when an underflow or input capture occurs.

Figure 20-3 is a block diagram of Capture Timer 1.

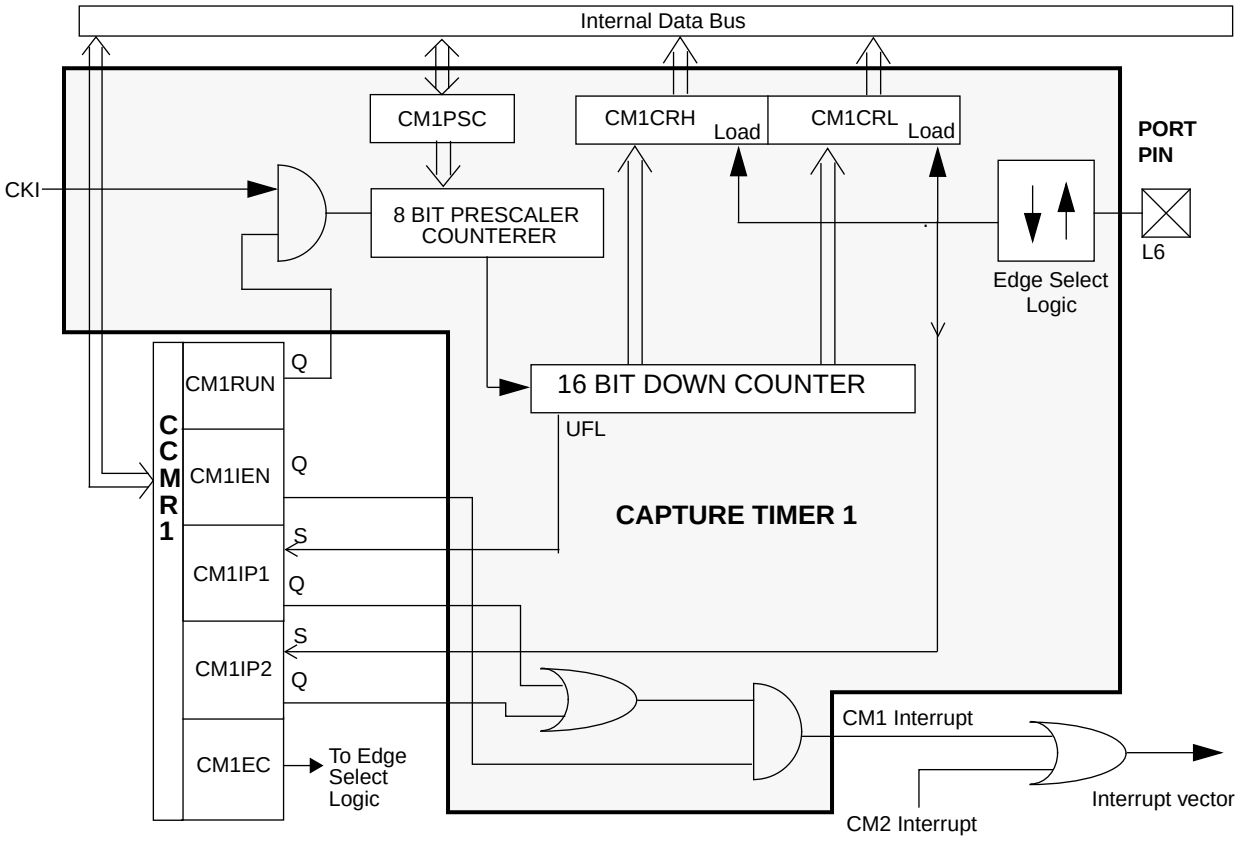


Figure 20-3 Capture Timer 1 Block Diagram

20.15 CAPTURE TIMER REGISTERS

Each capture timer register has four memory-mapped registers: a prescaler register, two capture registers (high and low bytes), and a control register. The prescaler and control registers are read/write registers and the two capture registers are read-only registers.

Table 20-20 lists and describes the capture registers.

Table 20-20 Capture Timer Registers

Register Name	Address	Description
CM1PSC	xxB0	Capture Timer 1 prescaler register
CM1CRL	xxB1	Capture Timer 1 capture register low byte
CM1CRH	xxB2	Capture Timer 1 capture register high byte
CM2PSC	xxB3	Capture Timer 2 prescaler register
CM2CRL	xxB4	Capture Timer 2 capture register low byte
CM2CRH	xxB5	Capture Timer 2 capture register high byte
CCMR1	xxB6	Capture Timer 1 control register
CCMR2	xxB7	Capture Timer 2 control register

Table 20-21 and 20-22 show the function of each bit in the capture timer control registers CCMR1 and CCMR2.

The CCMR1 Register Bits are:

CM1RUNCM1 start/stop control bit (1=start; 0=stop)
CM1IEN CM1 interrupt enable control bit (1= enable IRQ)
CM1IP1 CM1 interrupt pending bit 1 (1=CM1 underflowed)
CM1IP2 CM1 interrupt pending bit 2 (1=CM1 captured)
CM1EC Select the active edge for capture on CM1 (0=rising, 1=falling)
CM1TM CM1 test mode control bit (1=special test path in test mode. This bit is reserved during normal operation, and must never be set to one.)

Table 20-21 CCMR1 [xxB6] Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CM1TM	spare	spare	CM1EC	CM1IP2	CM1IP1	CM1IEN	CM1RUN

All interrupt pending bits must be reset by software.

The CCMR2 Register Bits are:

CM2RUNCM2 start/stop control bit (1=start; 0=stop)
CM2IEN CM2 interrupt enable control bit (1= enable IRQ)
CM2IP1 CM2 interrupt pending bit 1 (1=CM2 underflowed)
CM2IP2 CM2 interrupt pending bit 2 (1=CM2 captured)
CM2EC Select the active edge for capture on CM2 (0=rising, 1=falling)
CM2TM CM2 test mode control bit (1=special test path in test mode. This bit is reserved during normal operation, and must never be set to one.)

Table 20-22 CCMR2 [xxB7] Register Bits

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CM2TM	spare	spare	CM2EC	CM2IP2	CM2IP1	CM2IEN	CM2RUN

All interrupt pending bits must be reset by software.

20.16 CAPTURE TIMER OPERATION

Each capture timer can be used to determine the elapsed time between events, where an event is a edge transition on the capture input (port pin L6 or L7). The resolution and maximum range of the time measurement depend on the frequency at which the down counter is clocked. The CPU sets this frequency by loading an appropriate value into the prescaler register.

There are two cascaded counters in each capture timer: an 8-bit prescaler counter clocked by CKI and a 16-bit down counter clocked on each underflow of the prescaler counter. The prescaler counter divides the CKI clock frequency by a prescaler factor and feeds the resulting clock signal into the 16-bit down counter. The prescaler factor is 1 plus the value written to the CM1PSC or CM2PSC register. You can use the following equation to determine the appropriate value for CM1PSC or CM2PSC:

(down counter clock frequency) = (CKI frequency) / (CMxPSC + 1)

The following description refers to operation of Capture Timer 1. Capture Timer 2 operates in the same manner, independently from Capture Timer 1, using a different set of registers and port pin L7 rather than L6.

20.16.1 Initialization

To perform an input capture operation using Capture Timer 1 and its associated interrupts, the CPU should perform the following steps:

1. Clear the CM1RUN bit in the CCMR1 register.
2. Configure port pin L6 to operate as an input using the Port L configuration and data registers (configuration bit = 0, data bit = 1).
3. Set the CM1EC bit in the CCMR1 register to specify the active edge type (0 for rising edge, 1 for falling edge).

4. Clear the CM1IP1 and CM1IP2 bits in the CCMR1 register (underflow interrupt pending bit and capture interrupt pending bit)
5. Load the CM1PSC register with the appropriate value from 0 to 255 (the desired clock prescaler factor minus 1)
6. Set the CM1IEN bit in the CCMR1 register to enable capture timer interrupts.
7. Set the Global Interrupt Enable (GIE) bit in the PSW register.
8. Set the CM1RUN bit in the CCMR1 register to start the capture timer.

20.16.2 Port Configuration

The Capture Timer 1 input is an alternate function of port pin L6. If you configure the port pin to operate as an input, an active edge received at that input results in a capture. The edge select bit in the capture timer control register determines whether a positive-going or negative-going edge is the active edge type.

If the port pin is configured to operate as an output, toggling the data register bit in the appropriate direction (0 to 1 or 1 to 0) results in a timer capture. This configuration can be used to time events detected by the software.

The rest of the capture timer description assumes that the port pin is configured as an input.

20.16.3 Interrupts

You can choose to use interrupts or polling to communicate information from the capture timer to the CPU. If you choose to use interrupts, there are two capture timer events that can trigger the interrupt: underflow of the 16-bit down counter or an active edge received at the capture input. Each event has its own interrupt pending flag: CM1IP1 for underflow and CM1IP2 for an input capture. These interrupt pending flags must be reset by the software.

Both the capture and the underflow interrupt sources are enabled by the same control bit (CM1IEN in the CCMR1 register). The Global Interrupt Enable (GIE) bit in the PSW register must also be set to enable interrupts.

To avoid the possibility of erroneous interrupts, the CPU should disable the capture timer interrupt prior to setting or clearing the capture edge control bit, CM1EC. In addition, after the CPU sets the active edge type, it should reset the two Capture Timer 1 interrupt pending flags before it enables or re-enables the capture interrupt.

The interrupt signals from both Capture Timer 1 and Capture Timer 2 are gated to a single 16-bit interrupt vector located at 0xE6 and 0xE7.

20.16.4 Starting the Timer

To start Capture Timer 1, the CPU sets the CM1RUN bit in the CCMR1register. Setting this bit has the following effects:

- Starts the timer.
- Enables operation of port pin L6 as the capture input.
- Loads the prescaler counter from CM1PSC.
- Loads the 16-bit down counter with FFFF Hex.

The prescaler counter is clocked by CKI and counts down. Each underflow of the prescaler counter decrements the 16-bit down counter and also reloads the programmed value in CM1PSC into the prescaler counter.

20.16.5 Edge Transitions and Underflow

In order to guarantee recognition of an input transition on L6, the width of the transition pulse must be at least one CKI clock period. A pulse shorter than one CKI clock period may not be recognized.

An active edge received at L6 has the following effects:

- The contents of the 16-bit down counter are immediately latched into the capture register.
- The 16-bit down counter is reloaded with FFFF Hex. (The prescaler counter is not reloaded.)
- The capture input pending flag (CM1P2) is set.
- A CPU interrupt is generated if interrupts are enabled by CM1IEN and GIE.

The prescaler counter and 16-bit down counter continue to operate until a reset condition occurs or the capture timer is stopped. The CPU stops the timer by clearing the CM1RUN bit in the CCMR1 register.

The CPU must process the interrupt before the next edge transition is received on L6. Otherwise, the next input transition may not be detected and erroneous time values may be read from the capture registers.

If the down counter underflows (changes from 0000 to FFFF Hex) before an input capture is detected, the underflow interrupt pending flag (CM1P1) is set, and if interrupts are enabled, a CPU interrupt is generated.

20.16.6 Stopping the Capture Timer

The CPU can stop the capture timer at any time by clearing the CM1RUN bit in the CCMR1 register. The contents of the prescaler counter and 16-bit down counter remain constant while the capture timer is stopped. The capture edge detection logic is disabled and no captures are detected, even if a transition occurs on the input pin.

Due to the asynchronous nature of the input capture signal, it is possible for a capture to occur between time that the software clears the bit and the physical disabling of the capture timer. For reliable operation, the software must be written to handle this possibility.

To properly process a capture that occurs when the timer is stopped, the capture timer interrupts should not be disabled prior to clearing the CM1RUN bit. If you choose to ignore a possible capture occurring at this time, the capture timer interrupt service routine should verify that the timer is still running prior to processing each interrupt.

If interrupts are not being used, the software should poll the two capture timer interrupt pending bits after the timer has been stopped to determine whether a capture has occurred. If you choose to ignore a possible capture occurring at this time, the software should clear the two interrupt pending bits.

The CPU can restart the capture timer by setting the CM1RUN bit in the CCMR1 register. This reloads the prescaler counter from the CM1PSC register and reloads the 16-bit down counter with FFFF Hex.

20.17 RESET STATE

A reset signal applied to the device during normal operation clears the CCMR1 and CCMR2 registers, but does not affect the CM1PSC, CM1CRL, CM1CRH, CM2PSC, CM2CRL, and CM2CRH registers.

Upon power-up, the contents of the CM1PSC, CM1CRL, CM1CRH, CM2PSC, CM2CRL, CM2CRH registers are undefined.

The bidirectional ports pins are initialized to the high-impedance state during a reset. Setting the CM1RUN or CM2RUN connects the L6 or L7 port pin to the corresponding capture timer.

20.18 PULSE TRAIN GENERATORS

The COP888GW contains four independent pulse train generators, designated Counters 1, 2, 3, and 4. The pulse train outputs are made available as alternate functions of port pins E0, E1, E2, and E3, respectively. Each pulse train generator can be configured to produce a specified number of pulses with a 50% duty cycle.

Each pulse train generator contains a 16-bit programmable prescaler register and a 16-bit count register. The value programmed into the prescaler register determines the frequency of the output pulses, while the value programmed into the count register determines the number of pulses produced in a pulse train.

Each pulse train counter can be stopped and started under software control, and each can be configured to interrupt the CPU on whenever a series of pulses is completed.

Figure 20-4 is a block diagram of Pulse Train Generator 1.

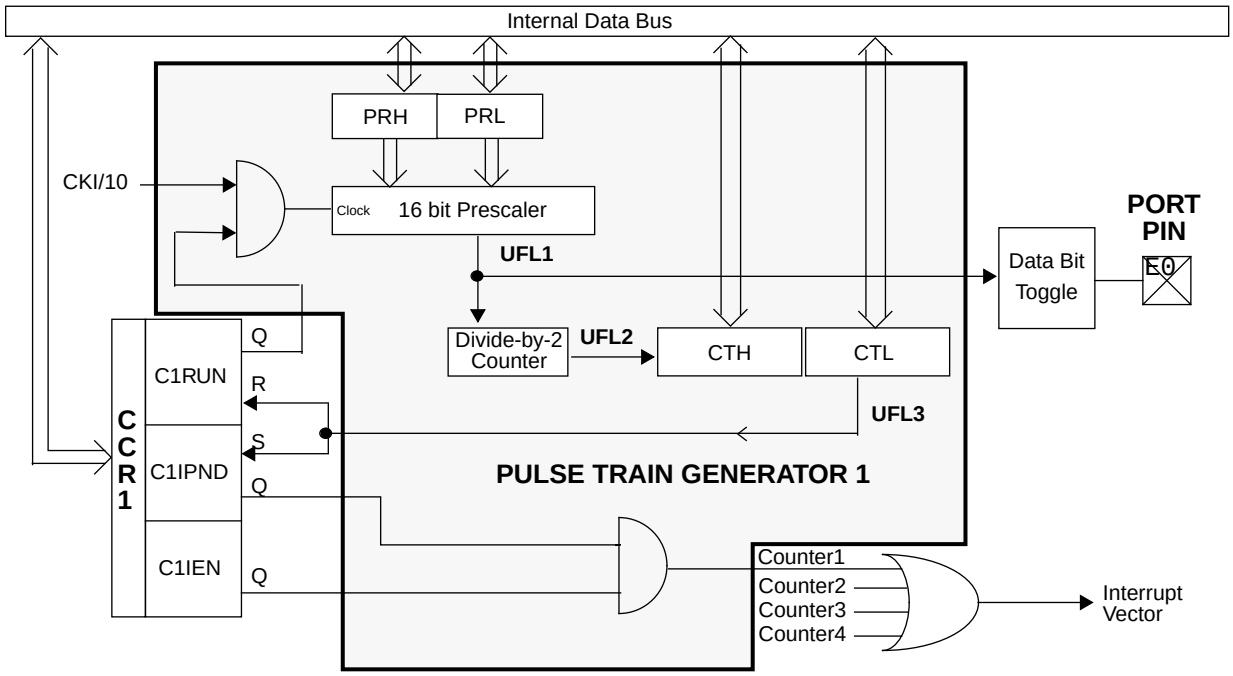


Figure 20-4 Pulse Train Generator 1 Block Diagram

20.19 PULSE TRAIN GENERATOR REGISTERS

Each pulse train generator register has four memory-mapped registers: a two 8-bit prescaler registers for the 16-bit prescaler and two 8-bit count registers for the 16-bit count. There is also one 8-bit control register for controlling both Counter 1 and Counter 2, and another for controlling both Counter 3 and Counter 4. All of these registers are read/write registers.

Table 20-23 lists and describes the pulse train generator registers.

Table 20-23 Pulse Train Generator Registers (Sheet 1 of 2)

Register Name	Address	Description
CCR1	xx9E	Counter control register 1 (for Counters 1 and 2)
CCR2	xx9F	Counter control register 2 (for Counters 3 and 4)
C1PRL	xxA0	Counter 1 prescaler low byte
C1PRH	xxA1	Counter 1 prescaler high byte
C1CTL	xxA2	Counter 1 count register low byte
C1CTH	xxA3	Counter 1 count register high byte

Table 20-23 Pulse Train Generator Registers (Sheet 2 of 2)

Register Name	Address	Description
C2PRL	xxA4	Counter 2 prescaler low byte
C2PRH	xxA5	Counter 2 prescaler high byte
C2CTL	xxA6	Counter 2 count register low byte
C2CTH	xxA7	Counter 2 count register high byte
C3PRL	xxA8	Counter 3 prescaler low byte
C3PRH	xxA9	Counter 3 prescaler high byte
C3CTL	xxAA	Counter 3 count register low byte
C3CTH	xxAB	Counter 3 count register high byte
C4PRL	xxAC	Counter 4 prescaler low byte
C4PRH	xxAD	Counter 4 prescaler high byte
C4CTL	xxAE	Counter 4 count register low byte
C4CTH	xxAF	Counter 4 count register high byte

Table 20-24 and Table 20-25 show the function of each bit in the two control registers, CCR1 and CCR2.

The control bits for Counter1 and Counter2 are contained in the CCR1 register. The CCR1 Register bits are:

C1RUN	COUNTER1 start/stop control bit (1=start; 0=stop)
C1IEN	COUNTER1 interrupt enable control bit (1= enable IRQ)
C1IPND	COUNTER1 interrupt pending bit (1=counter 1 underflowed)
C1TM	COUNTER1 test mode control bit (1=special test path. This bit is re served during normal operation, and must never be set to one.)
C2RUN	COUNTER2 start/stop control bit (1=start; 0=stop)
C2IEN	COUNTER2 interrupt enable control bit (1= enable IRQ)
C2IPND	COUNTER2 interrupt pending bit (1=counter 2 underflowed)
C2TM	COUNTER2 test mode control bit (1=special test path in test mode. This bit is reserved during normal operation, and must never be set to one.)

Table 20-24 CCR1 Register Bits (Address xx9E Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C2TM	C2IPND	C2IEN	C2RUN	C1TM	C1IPND	C1IEN	C1RUN

All interrupt pending bits must be reset by software.

The control bits for Counter3 and Counter4 are contained in the CCR2 register. The CCR2 Register bits are:

C3RUN	COUNTER3 start stop control bit (1=start; 0=stop)
C3IEN	COUNTER3 interrupt enable control bit (1= enable IRQ)
C3IPND	COUNTER3 interrupt pending Bit (1=counter 3 underflowed)
C3TM	COUNTER3 test mode control bit (1=special test path in test mode. This bit is reserved during normal operation, and must never be set to one.)
C4RUN	COUNTER4 start/stop control bit (1=start; 0=stop)
C4IEN	COUNTER4 interrupt enable control bit (1= enable IRQ)
C4IPND	COUNTER4 interrupt pending bit (1=counter 4 underflowed)
C4TM	COUNTER4 test mode control bit (1=special test path in test mode. This bit is reserved during normal operation, and must never be set to one.)

Table 20-25 CCR2 Register Bits (Address xx9F Hex)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C4TM	C4IPND	C4IEN	C4RUN	C3TM	C3IPND	C3IEN	C3RUN

All interrupt pending bits must be reset by software.

20.20 PULSE TRAIN GENERATOR OPERATION

Each pulse train generator can be used to produce a series of output pulses of a given width. The pulse width is determined by the value loaded into the two 8-bit prescaler registers. The number of pulses in a series is determined by the value loaded into the two 8-bit count registers.

There are two counters in each pulse train generator: a 16-bit prescaler counter and a 16-bit down counter. The 16-bit prescaler counter is initially loaded with the programmed prescaler value and then decremented on each rising edge of t_c (CKI/10).

Each time the 16-bit prescaler counter underflows, it generates an internal underflow signal called UFL1 that causes the output pin to toggle from 0 to 1 or from 1 to 0. In addition, the 16-bit prescaler counter is reloaded with the programmed prescaler value.

Every other overflow of the 16-bit prescaler counter generates another internal signal called UFL2. The UFL2 signal operates at half the rate of UFL1. UFL2 decrements the 16-bit count register. As a result, the 16-bit count register is decremented once for each output pulse.

Each underflow of the 16-bit count register generates another internal signal called UFL3. This signal sets the pulse generator interrupt pending bit and stops pulse generation by clearing the control register's RUN bit. It also generates a CPU interrupt if that interrupt is enabled.

To specify the output pulse width t_w, the CPU loads the appropriate value into the 16-bit prescaler register (for example, C1PRH-C1PRL for Counter 1). The high time and low time of the output pulse are each equal to t_w, so the output pulse train has a 50% duty

cycle and a period of $2*t_w$. The appropriate values for the two 8-bit prescaler registers can be determined as follows:

$$t_w = [\ (256*PRH) + PRL + 1\ ] * t_c$$

where PRH and PRL are the 8-bit values loaded into the prescaler registers and t_c is the period of the instruction clock (CKI/10). With a CKI clock of 10 MHz, t_c is one microsecond. In that case, the maximum allowed value of t_w is 65536 microseconds and the minimum allowed value is 1 microsecond.

The specify the number of output pulses generated in a series, the CPU loads the appropriate value into the 16-bit count register (for example, CT1CTH-C1CTL for Counter 1). The number of pulses generated is equal to 1 plus the 16-bit value. The appropriate values for the two 8-bit count registers can be determined as follows:

$$\text{number of pulses} = 256*CTH + CTL + 1$$

where CTH and CTL are the 8-bit values loaded into the count registers. Thus, the number of pulses in a series is can be set to any value from 1 to 65536.

The following description refers to operation of Pulse Train Generator 1. Pulse Train Generators 2, 3, and 4 operate in the same manner, each using a different sets of registers and a different port pin.

20.20.1 Initialization

To generate a pulse train using Pulse Train Generator 1, the CPU should perform the following steps:

1. Load the C1PRL register with the prescaler low byte.
2. Load the C1PRH register with the prescaler high byte. The prescaler factor is 1 plus the loaded 16-bit value.
3. Load the C1CTL register with the count low byte.
4. Load the C1CTH register with the count high byte. The number of pulses in the pulse train is 1 plus the loaded 16-bit value.
5. Clear the C1IPND (interrupt pending) bit in the CCR1 register.
6. Set the C1IEN bit in the CCR1 register to enable pulse train generator interrupts (if interrupts are to be used).
7. Configure port pin E0 to operate as an output and specify its initial output value using the Port E configuration and data registers (configuration bit = 1, data bit = initial output value).
8. Set the Global Interrupt Enable (GIE) bit in the PSW register (if interrupts are to be used).
9. Set the C1RUN bit in the CCR1 register to start the counter and pulse train output.

20.20.2 Port Configuration

The Pulse Train Generator 1 output is an alternate function of port pin E0. You must configure E0 to operate as an output before you can generate pulses on that pin. The initial output value on the pin prior to the start of the first pulse is determined by the initial value of bit 0 in the Port E data register.

20.20.3 Interrupts

You can optionally generate an interrupt to the CPU on each underflow of the 16-bit counter. This counter underflow occurs upon completion of a pulse train series. The interrupt pending bit, if used, must be reset by the software. To enable interrupts of this type, the CPU must set both the CM1IEN bit in the CCMR1 register and the Global Interrupt Enable (GIE) bit in the PSW register.

The interrupt signals from all four pulse train generators (1 through 4) are gated to a single 16-bit interrupt vector located at 0xF0 and 0xF1.

20.20.4 Starting the Pulse Train

To start a pulse train from Pulse Train Generator 1, the CPU sets the C1RUN bit in the CCR1 register. Setting this bit has the following effects:

- Resets the divide-by-2 counter that generates UFL2 from UFL1.
- Reloads the internal 16-bit prescaler counter from C1PRH-C1PRL.
- Loads the 16-bit down counter from C1CTH-C1CTL.
- Enables down counting of the 16-bit prescaler counter on the next rising edge of tc.
- Enables operation of port pin E0 as the pulse train output.

20.20.5 Stopping the Pulse Train Generator

The CPU can stop the pulse train generator at any time by clearing theC1RUN bit in the CCR1 register. The count register and pulse train output pin remain constant while the pulse train generator is stopped.

It is possible for an underflow to occur between time that the software clears the bit and the physical disabling of the pulse train generator. For reliable operation, the software must be written to handle this possibility.

To properly process an underflow that occurs when the pulse train generator is stopped, the pulse train generator interrupts should not be disabled prior to clearing the C1RUN bit. If you choose to ignore a possible underflow occurring at this time, the pulse train generator interrupt service routine should verify that the pulse train generator is still running prior to processing each interrupt.

If interrupts are not being used, the software should poll the pulse train generator interrupt pending bit after the pulse train generator has been stopped to determine

whether an underflow has occurred. If you choose to ignore a possible underflow occurring at this time, the software should clear the interrupt pending bit.

The CPU can restart the pulse train generator by setting the C1RUN bit in the CCR1 register. This reloads the prescaler counter from C1PRH- C1PRL. As a result, a full-width pulse is generated when the pulse generator is restarted. The 16-bit count register is not reloaded, but retains its value, so the remaining number of pulses of the pulse train are generated.

To start with a new value in the count register after the counter has been stopped, the CPU must reload the C1CTH/C1CTL register before restarting it. It is not necessary to reload the counter if you want to continue from the point at which the counter was stopped.While the pulse train generator is stopped, the CPU can control the output value of E0 by setting or clearing bit 0 of the Port E data register.

20.20.6 Reset State

A reset signal applied to the device during normal operation has the following effect on each pulse train generator:

- Counting stops immediately.
- The interrupt enable bit (CxIEN) is cleared.
- The start/stop bit (CxRUN) is cleared.
- The interrupt pending bit (CxIPND) is cleared.
- The test mode control bit (CxTM) is cleared.
- The UFL1 to UFL2 divide-by-2 counter is reset.
- The bidirectional I/O pins are initialized to the high-impedance state.

A reset does not affect the CxPRL, CxPRH, CxCTL, and CxCTH registers.

Upon power-up, the contents of these registers are undefined.

20.21 MASK OPTIONS

The COP888GW has a mask-programmed ROM memory, which contains the object code and the fixed data of the application program. At the same time that the ROM pattern is programmed into the device, certain device options are also programmed. Each option is assigned a number, and each option selection is assigned another number (Option 1 =3, Option 2 =1, etc.). The types of options and the available selections are listed below.

Option 1: Clock Configuration

=1 Crystal Oscillator (CKI/10)
G7 (CK) is the clock generator output to a crystal/resonator
CKI is the clock input

Option 2: HALT

=1 Enable HALT mode

=2 Disable HALT mode

Option 3: COP888GW Bonding

=1 68-pin PLCC

Appendix A

APPLICATION HINTS

A.1 INTRODUCTION

This chapter describes several application examples using the COP888 family of microcontrollers. Design examples often include block diagrams and/or assembly code. Certain hardware design considerations are also presented.

Topics covered in this chapter include the following:

- MICROWIRE/PLUS implementation examples
- Timer application examples
- Triac control example
- Analog to digital conversion technique
- Battery-powered weight measurement example
- Zero Cross Detection
- Industrial timer example
- Programming examples (clear RAM, binary arithmetic)
- External power wakeup circuit
- Watchdog Reset Circuit
- Input protection on COP888 pins
- Electromagnetic interference (EMI) considerations

A.2 MICROWIRE/PLUS INTERFACE

A large number of off-the-shelf devices are directly compatible with the MICROWIRE/PLUS interface. This allows direct interface of the COP888 microcontrollers with a large number of peripheral devices. The following sections provide examples of the MICROWIRE/PLUS interface. These examples include a master/slave mode protocol, code for a continuous mode of operation, code for a fast burst mode of operation, and a COP888CL to an NMC93C06 interface.

A.2.1 MICROWIRE/PLUS Master/Slave Protocol

This section gives a sample MICROWIRE/PLUS master/slave protocol, the slave mode operating procedure for the sample protocol, and a timing illustration of the sample protocol.

Master/Slave Protocol:

1. CS from the master device is connected to G0 of the slave device. An active-low level on the CS line causes the slave to interrupt.
2. From the high-to-low transition on the CS line, there is no data transfer on the MICROWIRE interface until the setup time T has elapsed (see Figure A-1).
3. The master initiates data transfer on the MICROWIRE interface by turning on the SK clock.
4. A series of data transfers takes place between the master and slave devices.
5. The master pulls the CS line high to end the MICROWIRE operation. The slave device returns to normal mode of operation.

Slave Mode Operating Procedure (for the previous protocol):

1. Set the MSEL bit in the CNTRL register to enable MICROWIRE; G0 and G5 are configured as inputs and G4 as an output. Reset bit 6 of the Port G configuration register to select Standard SK Clocking mode.
2. Normal mode of operation until interrupted by CS going low.
3. Set the BUSY flag and load SIOR register with the data to be sent out on S0. (The shift register shifts eight bits of data from S0 at the high-order end of the shift register. Concurrently, eight new bits of data from SI are loaded into the low-order end of the shift register.)
4. Wait for the BUSY flag to be reset. (The BUSY flag automatically resets after 8 bits of data have been shifted.)
5. If data is being read in, the contents of the SIO register are saved.
6. The prearranged set of data transfers are performed.

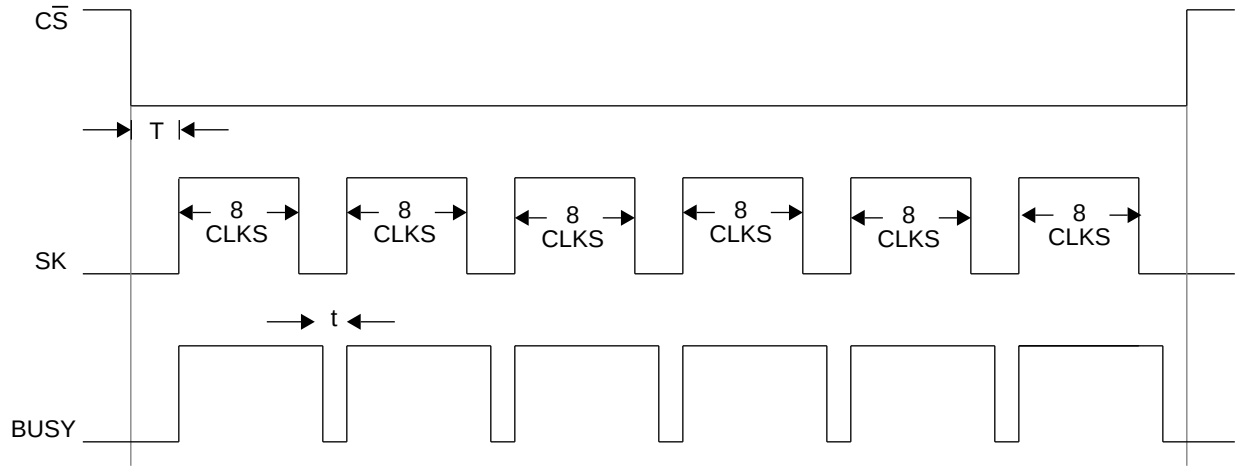


Figure A-1 MICROWIRE/PLUS Sample Protocol Timing

7. Repeat steps 3 through 6. The user must ensure step 3 is performed within *t-time* (refer to Figure A-1) as agreed upon in the protocol.

A.2.2 MICROWIRE/PLUS Continuous Mode

The MICROWIRE/PLUS interface can be used in continuous clock mode with the master mode divide-by-eight clock division factor selected. The maximum data transfer rate for this MICROWIRE/PLUS continuous clock mode is 64 microseconds per byte (equivalent to 125 KHz) for parts operating with a 1 µsec instruction cycle.

The continuous clock mode is achieved by resetting the BUSY bit under program control just before it would automatically be reset with the hardware, and then immediately setting the BUSY bit with the next instruction. The SIO MICROWIRE shift register is then loaded (or read) with the following instruction. This loading of SIO occurs before the SK clock goes high, even though the previous set BUSY bit instruction has started the divide-by-eight (3-stage) counter. The B pointer must be already set up to point at the PSW register where the MICROWIRE BUSY bit is located. This three-instruction sequence is programmed as follows:

Instruction			Bytes/Cycles
RBIT	BUSY,	[B]	1/1
SBIT	BUSY,	[B]	1/1
X	A,	SIOR	2/3

This three-instruction sequence must be embedded in an instruction program loop that is exactly 64 instruction cycles (t_C cycles) in length. This yields a 125-KHz (64 microseconds per byte) data transfer rate at the maximum instruction cycle rate of 1 MHz.

The following program demonstrates the use of the MICROWIRE/PLUS continuous clock mode. The program continually outputs the 256 bytes of the current program memory block on the MICROWIRE S0 output pin (G4). The low-order bit (G0) of Port G is set to cover the transition period of the three-instruction sequence outlined previously, where the SIO register is loaded with a new byte.

```
          .SECT   MW, REG
MWMEM:    .DSB 2
          MWTEMP = MWMEM
          MWCNT  = MWMEM+1

          MARK = 0
          BUSY = 2
          .SECT   CODE, ROM                    CYCLES
MWCONT:   LD      PORTGC, #031
          LD      PORTGD, #0
          LD      CNTRL, #0B
          LD      B, #PSW
          LD      MWCNT, #0
MWLOOP:   LD      A, MWCNT                      3
          INC     A                             1
          X       A, MWCNT                      3
          LAID                                  3
          SBIT    MARK, PORTGD                  4
          RBIT    BUSY, [B]                     1
          SBIT    BUSY, [B]                     1
          X       A, SIOR                       3
          RBIT    MARK, PORTGD                  4
          LD      MWTEMP, #6                    3
```

```
MWLUP:  DRSZ    MWTEMP          3 }
        JP      MWLUP           3 } x6 - 2 = 34
        NOP                                  1
        JP      MWLOOP                       3

        TOTAL CYCLES IN MWLOOP =            64
```

A.2.3 MICROWIRE/PLUS Fast Burst Output

The maximum COP888 MICROWIRE/PLUS master mode burst clock rate (using the divide-by-two clock division factor) is 500 KHz. This assumes that the COP888 microcontroller is running at the maximum instruction cycle frequency of 1 MHz. The equivalent time of one extra master mode SK clock cycle is necessary to set up the next byte (and/or read the previous byte) in SIOR when using the burst mode SK frequency. This yields an equivalent minimum data transfer time of 18 microseconds per byte.

The following program demonstrates the use of the MICROWIRE/PLUS burst clock mode at the maximum data transfer rate (with the divide-by-two master mode clock option selected). The X pointer is initialized to the TOP of a RAM table, where SIZE represents the size of the table. This subroutine outputs the contents of the RAM table on the MICROWIRE S0 output pin (G4).

```
        .SECT   MW, REG
MWCNT:  .DSB 1

        BUSY = 2
        .SECT   MWFB, ROM          INSTRUCTION    CYCLE NUMBER
                                   CYCLES          IN Figure A-2
MWBRST: LD      CNTRL, #8
        LD      B, #PSW
        LD      MWCNT, #SIZE
        LD      X, #TOP
MWLOOP: LD      A, [X-]             3             13, 14, 15
        X       A, SIOR             3             16, 17, 18
        SBIT    BUSY, [B]           1                   1
        NOP*                        1                   2
        NOP*                        1                   3
        LAID*                       3              4, 5, 6
        DRSZ    MWCNT               3              7, 8, 9
        JP      MWLOOP              3             10, 11, 12
        RET

        TOTAL CYCLES IN MWLOOP =   18
```

* Time Delay

The MICROWIRE BUSY bit is allowed to reset automatically with the hardware following the eighth SK clock. The transfer of new data into the SIO register and the transfer of the new input data from SIO to A occurs at the end of the second cycle of the three-cycle “exchange A with SIO” instruction. This exchange instruction is immediately followed by the “set BUSY bit” instruction to initiate another MICROWIRE serial byte transfer. The associated timing for this 18-instruction cycle MICROWIRE loop is shown in Figure A-2.

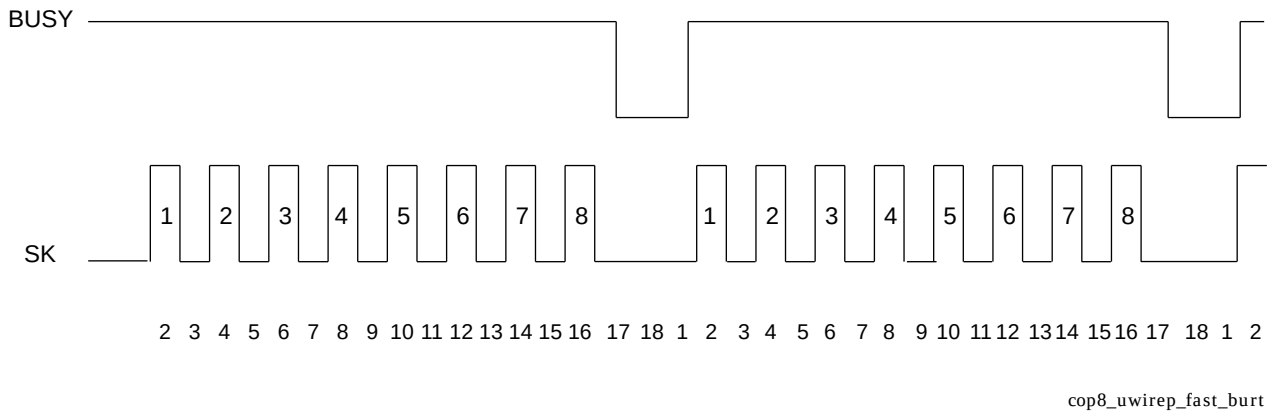


Figure A-2 MICROWIRE/PLUS Fast Burt Timing

A.2.4 NMC93C06-COP888CL Interface

This example shows the COP888CL interface to a NMC93C06, a 256-bit E^2PROM, using the MICROWIRE interface. The pin connection for interfacing a NMC93C06 with the COP888CL microcontroller is shown in Figure A-3. Some notes on the NMC93C06 interface requirements are:

1. The SK clock frequency should be less than 250 KHz. The SK clock should be configured for standard SK mode.
2. The CS low period following an Erase/Write instruction must not exceed 30 ms maximum. It should be set at the typical or minimum specification of 10 ms.

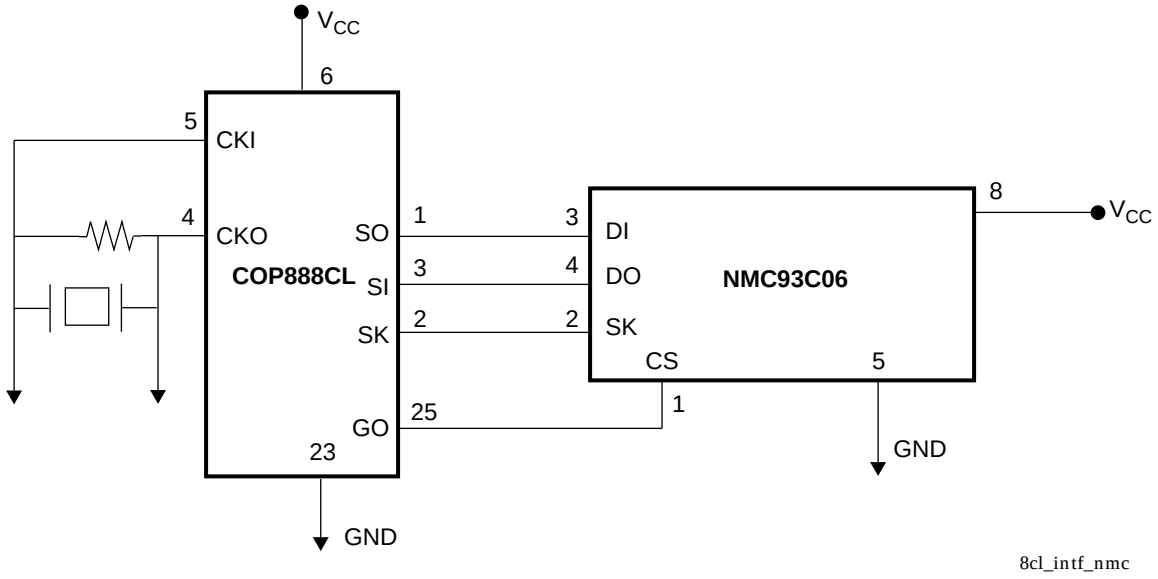


Figure A-3 NMC93C06-COP888CL Interface

3. The start bit on DI must be programmed as a “0” to “1” transition following a CS enable (“0” to “1”) when executing any instruction. One CS enable transition can execute only one instruction.

4. In the read mode, following an instruction and data train, the DI is a “don’t care” while the data is being output for the next 17 bits or clocks. The same is true for other instructions after the instruction and data has been fed in.

5. The data out train starts with a dummy bit 0 and is terminated by chip deselect. Any extra SK cycle after 16 bits is ignored. If CS is held on after all 16 of the data bits have been output, the DO will output the state of DI until another CS low to high transition starts a new instruction cycle.

6. After a read cycle, the CS must be brought low for one SK clock cycle before another instruction cycle starts.

The following table describes the instruction set of the NMC93C06. In the table A3A2A1A0 corresponds to one of the sixteen 16-bit registers.

Commands	Start Bit	Opcode	Address	Comments
READ	1	0000	A3A2A1A0	Read Register 0–15
WRITE	1	1000	A3A2A1A0	Write Register 0–15
ERASE	1	0100	A3A2A1A0	Erase Register 0–15
EWEN	1	1100	0001	Write/Erase Enable
EWDS	1	1100	0010	Write/Erase Disable
WRAL	1	1100	0100	Write All Registers
ERAL	1	1100	0101	Erase All Registers

All commands and data are shifted in/out on the rising edge of the SK clock. All instructions are initiated by a low-to-high transition on CS followed by a low-to-high transition on DI.

A detailed explanation of the NMC93C06 E^2PROM timing, instruction set, and other considerations can be found in the datasheet. A source listing of the software to interface the NMC93C06 with the COP888CL is provided below.

```
;This program provides in the form of subroutines, the ability to erase,
;enable, disable, read and write to the NMC93C06 EEPROM.

.INCLD   COP888.INC
         .SECT    NMC, RAM
NMCMEM:  .DSB 5
         SNDBUF = NMCMEM      ;Contains the command byte to be written NMC93C06
         RDATL  = NMCMEM+1    ;Lower byte of NMC93C06 register data read
         RDATH  = NMCMEM+2    ;Upper byte of NMC93C06 register data read
         WDATL  = NMCMEM+3    ;Lower byte of data to be written to NMC93C06 register
         WDATH  = NMCMEM+4    ;Upper byte of data to be written to NMC93C06 register
ADRESS:  .DSB 1               ;Lower 4-bits of this location contain the address of
                              ;the NMC93C06 register to read/write
FLAGS:   .DSB 1               ;Used for setting up flags
                              ;Flag valueAction
                              ;00Erase, enable, disable, erase all
                              ;01Read contents of NMC93C06 register
```

```
                              ;03Write to NMC93C06 register
                              ;OthersIllegal combination
        .SECT   CNT, REG
DLYH:   .DSB 1                ;Delay counter register
DLYL:   .DSB 1                ;Delay counter register

;The interface between the COP888 and the NMC93C06 (256-bit EEPROM) consists of four
;lines: The G0 (chip select line), G4 (serial out SO), G5 (serial clock SK), and
;G6 (serial in SI).
;
; Initialization
;
        .SECT   NMCODE, ROM
        LD      PORTGC,#031 ;Setup G0, G4, G5 as outputs and
                            ;select standard SK mode
        LD      PORTGD,#00  ;Initialize G data reg to zero
        LD      CNTRL,#08   ;Enable MSEL, select MW rate of 2tc
        LD      B,#PSW
        LD      X,#SIOR


;This routine erases the memory location pointed to by the address contained in the
;location “ADRESS.” The lower nibble of “ADRESS” contains the NMC93C06 register
;address and the upper nibble should be set to zero.
;
ERASE:  LD      A,ADRESS
        OR      A,#0C0
        X       A,SNDBUF
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine enables programming of the NMC93C06. Programming must be preceded
;once by a programming enable (EWEN).
;
EWEN:   LD      SNDBUF,#030
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine disables programming of the NMC93C06
;
EWDS:   LD      SNDBUF,#0
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine erases all registers of the NMC93C06
;
ERAL:   LD      SNDBUF,#020
        LD      FLAGS,#0
        JSR     INIT
        RET

;This routine reads the contents of a NMC93C06 register. The NMC93C06 address is
;specified in the lower nibble of location “ADRESS.” The upper nibble should be set
;to zero. The 16-bit contents of the NMC93C06 register are stored in RDATL and RDATH.
;
READ:   LD      A,ADRESS
        OR      A,#080
        X       A,SNDBUF
        LD      FLAGS,#1
        JSR     INIT
        RET
```

```
;This routine writes a 16-bit value stored in WDATL and WDATH to the NMC93C06 register
;whose address is contained in the lower nibble of the location “ADRESS.” The upper
;nibble of address location should be set to zero.
;
WRITE:  LD      A,ADRESS
        OR      A,#040
        X       A,SNDBUF
        LD      FLAGS,#3
        JSR     INIT
        RET

;This routine sends out the start bit and the command byte. It also deciphers the
;contents of the flag location and takes a decision regarding write, read or return
;to the calling routine.
;
INIT:   SBIT    0,PORTGD    ;Set chip select high
        LD      SIOR,#001   ;Load SIOR with start bit
        SBIT    BUSY,[B]    ;Send out the start bit
PUNT1:  IFBIT   BUSY,[B]
        JP      PUNT1
        LD      A,SNDBUF
        X       A,[X]       ;Load SIOR with command byte
        SBIT    BUSY,[B]    ;Send out command byte
PUNT2:  IFBIT   BUSY,[B]
        JP      PUNT2
        IFBIT   0,FLAGS     ;Any further processing?
        JP      NOTDON      ;Yes
        RBIT    0,PORTGD    ;No, reset CS and return
        RET
;
NOTDON: IFBIT   1,FLAGS     ;Read or write?
        JP      WR494       ;Jump to write routine
        LD      SIOR,#000   ;No, read NMC93C06
        SBIT    BUSY,PSW    ;Dummy clock to read zero
        RBIT    BUSY,[B]
        SBIT    BUSY,[B]
PUNT3:  IFBIT   BUSY,[B]
        JP      PUNT3
        X       A,[X]
        SBIT    BUSY,[B]
        X       A,RDATH
PUNT4:  IFBIT   BUSY,[B]
        JP      PUNT4
        LD      A,[X]
        X       A,RDATL
        RBIT    0,PORTGD
        RET
;
WR494:  LD      A,WDATH
        X       A,[X]
        SBIT    BUSY,[B]
PUNT5:  IFBIT   BUSY,[B]
        JP      PUNT5
        LD      A,WDATL
        X       A,[X]
        SBIT    BUSY,[B]
PUNT6:  IFBIT   BUSY,[B]
        JP      PUNT6
        RBIT    0,PORTGD
        JSR     TOUT
        RET
```

```
;Routine to generate delay for write
;
TOUT:   LD      DLYH,#00A
WAIT:   LD      DLYL,#0FF
WAIT1:  DRSZ    DLYL
        JP      WAIT1
        DRSZ    DLYH
        JP      WAIT
        RET
        .END
```

A.3 TIMER APPLICATIONS

This section describes two applications that use the 16-bit on-chip timer: speed measurement with the Input Capture mode, and an external event counter with the External Event Counter mode.

A.3.1 Timer Capture Example

The Timer Input Capture Mode can be used to measure the time between events. The simple block diagram in Figure A-4 shows how the COP888 can be used to measure motor speed based on the time required for one revolution of the wheel. A magnetic sensor is used to produce a pulse for each revolution of the wheel.

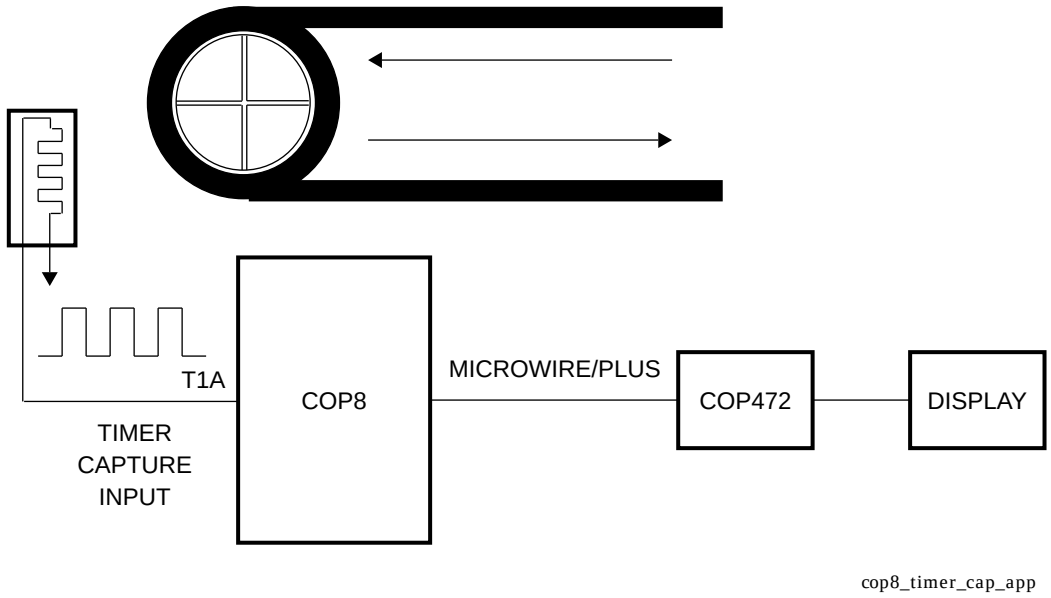


Figure A-4 Timer Capture Application

In the capture mode of operation, the timer counts down at the instruction cycle rate. In this application, the timer T1 is set up to generate an interrupt on a T1A positive edge transition. The timer is initialized to 0FFFF Hex and begins counting down. An edge transition on the T1A input pin of the timer causes the current timer value to be copied into the R1A register. In addition, it sets the timer interrupt pending flag, which causes a program branch to memory location 0FF Hex. The interrupt service routine for the timer is vectored to using the VIS instruction. The interrupt service routine resets the

T1PNDA pending flag. It then reads the contents of R1A and stores it in RAM for later processing. An RETI instruction is used to return to normal program execution and re-enable subsequent interrupts (by setting the GIE bit).

On the next rising edge transition on T1A, the program returns to the interrupt service routine. The value in R1A is read again, and compared with the previously read value. The difference between the two captured values, multiplied by the instruction cycle time, gives the time for one revolution. This is easily converted to a frequency. The frequency may be displayed on an LCD using the COP MICROWIRE/PLUS interface and a COP472-3 LCD Driver.

NOTE: The T1B input may be used simultaneously to measure the time between different events.

An example of the code that can be used for this application is provided below.

```
          .INCLD   COP888XX.INC
          .SECT    TMR, ROM
                                ;TIMER T1 CONFIGURATION
          LD       PORTGC,#00     ;Configure G3/T1A as input
          LD       PORTGD,#08     ;Weak pull-up on G3
          LD       TMR1L0,#0FF    ;Timer lower byte initialization
          LD       TMR1HI,#0FF    ;Timer upper byte initialization
          LD       ICNTRL,#00     ;Disable T1B interrupt
          LD       CNTRL,#0C0     ;Timer capture mode, positive edge on T1A
          LD       PSW,#011       ;Enable T1A and global interrupts
SELF:     JP       SELF           ;Wait for capture

          .SECT    INTERRUPT,ROM, ABS=0FF  ;TIMER INTERRUPT HANDLING ROUTINE
                                ;Set location counter to 00FF Hex
          VIS                   ;Vector to appropriate interrupt routine

                                ;TIMER SERVICE ROUTINE
T1SERV:   IFBIT    T1C0,CNTRL   ;If T1 underflow
          JP       UNDFLW       ;then handle timer underflow
          RBIT     T1PNDA,PSW   ;else Reset T1PNDA pending flag
          .                     ;(Process Timer Capture)
          .                     ;(Process Timer Capture)
          RETI                  ;Return from interrupt
UNDFLW:   RBIT     T1C0,CNTRL   ;Reset timer underflow pending flag
          .                     ;(Process Timer Underflow)
          RETI                  ;Return from interrupt

                                ;ERROR ROUTINE (NO INTERRUPT PENDING)
ERROR:    RETI                  ;Return from interrupt
          .SECT    INTTABLE, ROM, ABS=01E0  ;VECTOR TABLE
                                ;Set location counter to 01E0 Hex
.ADDRW    ERROR                 ;Store Error routine vector address
          ...                   ;{Store vector addresses 01E2 - 01F5 Hex}
                                ;Set location counter to 01F6 Hex
.ADDRW    T1SERV                ;Store T1PNDA routine vector address
          ...                   ;{Store vector addresses 01F8 - 01FF Hex}
.ENDSECT
```

A.3.2 External Event Counter Example

This mode of operation is very similar to the PWM Mode of operation. The only difference is that the timer is clocked from an external source. This mode provides the ability to perform control of a system based on counting a predetermined number of external

events, such as searching for the nth sector on a disk or testing every nth part on an assembly line. The code for this example is provided below.

```
          .INCLD     COP888XX.INC
          .SECT      EEC, ROM
                                    ;TIMER T1 CONFIGURATION
          RBIT       3,PORTGC       ;Configure G3/T1A as Hi-Z input
          RBIT       3,PORTGD       ;
          LD         CNTRL,#00      ;Select timer T1 as external event counter
          LD         ICNTRL,#00     ;Disable T1B interrupt
          LD         TMR1LO,#COUNT0 ;Timer T1 lower byte
          LD         TMR1HI,#COUNT1 ;Timer T1 upper byte
          LD         T1RALO,#Count0 ;Initialize Auto-reload R1A lower byte
          LD         T1RAHI,#Count1 ;Initialize Auto-reload R1A upper byte
          LD         T1RBLO,#Count0 ;Initialize Auto-reload R1B lower byte
          LD         T1RBHI,#Count1 ;Initialize Auto-reload R1B upper byte
          SBIT       T1C0,CNTRL     ;Start timer
          LD         PSW,#011       ;Enable timer T1A and global interrupts
SELF:     JP         SELF           ;Wait for the n-th count

          .SECT      INTERRUPT, ROM, ABS=0FF  ;TIMER INTERRUPT HANDLING ROUTINE
                                    ;Set location counter to 00FF Hex
          VIS                       ;Vector to appropriate interrupt routine

                                    ;TIMER SERVICE ROUTINE
T1SERV:   RBIT       T1PNDA,PSW     ;Reset T1A pending flag
          RBIT       T1C0,PSW       ;Stop timer
          .                         ;{Process timer interrupt}
          .                         ;{Process timer interrupt}
          SBIT       T1C0,PSW       ;Start timer
          RETI                      ;Return from interrupt

                                    ;ERROR ROUTINE (NO INTERRUPT PENDING)
Error:    RETI                      ;Return from interrupt

          .SECT      INTTABLE, ROM, ABS=01E0  ;VECTOR TABLE
                                    ;Set location counter to 01E0 Hex
.ADDRW    ERROR                     ;Store Error routine vector address
          ...                       ;{Store vector addresses 01E2 - 01F5 Hex}
                                    ;Set location counter to 01F6 Hex
.ADDRW    T1SERV                    ;Store T1PNDA routine vector address
          ...                       ;{Store vector addresses 01F8 - 01FF Hex}
.ENDSECT
```

A.4 TRIAC CONTROL

The COP888 family devices provide the computational ability and speed that is suitable for intelligently managing power control. In order to control a triac on a cyclic basis, an accurate time base must be established. This may be in the form of an AC 60Hz sync pulse generated by a zero voltage detection circuit or a simple real-time clock. The COP888 family is suited to accommodate either of these time base schemes while accomplishing other tasks.

Zero voltage detection is the most useful scheme in AC power control because it affords a real-time clock base as well as a reference point in the AC waveform. With this information it is possible to minimize RFI by initiating power-on operations near the AC line voltage zero crossing. It is also possible to fire the triac only a portion of the cycle, thus utilizing conduction angle manipulation. This is useful in both motor control and light-intensity control.

COP888 software is capable of compensating for noisy or semi-accurate zero voltage detection circuits. This is accomplished by using delays and debounce algorithms in the software. With a given reference point in the AC waveform, it becomes easy to divide the waveform to efficiently allocate processing time.

These techniques are demonstrated in the following code listing. This application example is based on the half cycle approach of AC power for triac light intensity control. The code intensifies and deintensifies a lamp under program control.

This program example is not intended to be a final functional program. It is a general-purpose light intensifying/deintensifying routine which can be modified for a light dimmer application. The delay routines are based on a 10 MHz crystal clock (1 µs instruction cycle). The COP888's 16-bit timer can be used for timing the half cycle of an AC power line, and the timer can be started or stopped under software control. Timer T1 is a read/write memory mapped counter with two associated 16-bit auto-reload registers. In this example, only one reload register is used because the timer is stopped after each timer reload from R1A. Zero crossings of the 60 Hz line are detected and software debounced to initiate each half cycle, so the triac is serviced on every half cycle of the power line. This program divides the half cycle of a 60 Hz AC power line into 16 levels. Intensity is varied by increasing or decreasing the conduction angle by firing the triac at various levels. Each level is a fixed time which is loaded into the timer. Once a true zero cross is detected, the timer starts and the triac is serviced.

A level/sublevel approach is utilized to vary the conduction angle and to provide a prolonged intensifying period. The maximum intensity reached is at the maximum conduction angle (level), and the time required to get to that level is loaded into the timer in increments. Once a level has been specified, the remaining time in the half cycle is then divided into sublevels. The sublevels are increased in steps to the maximum level and the triac is fired 16 times per sublevel, thus creating the intensity time base. For deintensifying, the sublevels are decremented.

```
;This is a general purpose light dimmer program
;it uses a 10 MHz clock (1 us instruction cycle time)

        .INCLD COP888.INC
        .TITLE TIMER, 'TIMER APPLICATION EXAMPLE'

        .SECT   TRIAC, REG      ;INITIALIZATIONS
        TEMP:   .DSB 1          ;Temporary storage location
        LEVEL:  .DSB 1          ;Level storage location
        FIN:    .DSB 1          ;Fire number storage location
        REG1:   .DSB 1          ;Register1 definition
        .SECT   MAIN,ROM
        LD      FIN,#000        ;Set fire number to zero
        LD      LEVEL,#040      ;Set sublevel to 40 Hex
        LD      PORTGC,#000     ;Configure Port G as all inputs
        LD      PORTGD,#004     ;Weak-up on pin G2
        LD      CNTRL,#080      ;Configure Timer T1 in autoreload mode
        LD      PSW,#000        ;Disable all interrupts
        LD      TMR1LO,#07D     ;Initialize T1 and T1RA with 0.5mS delay
        LD      TMR1HI,#000     ;
        LD      T1RALO,#0EB     ;
        LD      T1RAHI,#003     ;

                                ;POWER UP SYNCHRONIZATION OR RESET SYNCH.
BEG:    IFBIT   2,PORTGP        ;If Bit G2 = 1
        JP      HI              ;then re-check bit
        JP      BEG             ;else keep looping to synch up 60Hz
HI:     IFBIT   2,PORTGP        ;If Bit G2 is still 1
```

```
        JP      HI              ;then wait until it is zero
                                ;else test for true zero cross

                                ;TEST FOR TRUE ZERO CROSS (Valid Transition)
                                ;Debounce for zero cross detection
        JSR     DELAY           ;When Bit G2 = 0, perform debounce delay
        IFBIT   2,PORTGP        ;If Bit G2 is high after the delay
        JP      BEG             ;then false alarm, go back to beginning
DOIT:   JMP     INIT            ;else go start program
                                ;Debounce 0 to 1
LO:     IFBIT   2,PORTGP        ;If Bit G2 is high
        JP      D1              ;then go perform debounce delay
        JP      LO              ;else loop back and wait for a 1
D1:     JSR     DELAY           ;Debounce delay (clean transition)
        IFBIT   2,PORTGP        ;If Bit G2 is still 1
        JP      DOIT            ;then go start program
        JP      LO              ;else false alarm, keep debouncing

                                ;MAIN ROUTINE FOR INTENSIFY/DE-INTENSIFY
                                ;A true zero cross has been detected
INIT:   JSR     TIMER           ;Delay for 1ms to get to MAX
        LD      A,FIN           ;Load accumulator with fire number
        IFEQ    A,#015          ;If the fire number equals 15
        JP      THER            ;then finished firing, continue on
BEGG:   INC     A               ;else increment fire number
        X       A,FIN           ;Save new fire number
        JP      FIRE            ;Keep firing
THER:   LD      FIN,#000        ;Reset fire number to zero
        LD      A,LEVEL         ;Load accumulator with sublevel number
        DEC     A               ;Decrement sublevel
        X       A,LEVEL         ;Save new sublevel
        LD      A,LEVEL         ;Load sublevel back into accumulator
        IFEQ    A,#000          ;If sublevel = MAX level
        JP      LP2             ;then go reset level
        JP      LP3             ;else go check level
LP2:    LD      LEVEL, #040     ;Reset level to 40 Hex
        JP      FIRE            ;Go fire (exit)
LP3:    IFBIT   5,LEVEL         ;If current level is greater than 1F Hex
        JSR     ADD             ;then MAX not yet reached, add delay
        JSR     SUB             ;else MAX has been reached, subtract delay
        NOP                     ;No-operation
        NOP                     ;No-operation

                                ;FIRE SUBROUTINE
FIRE:   LD      PORTD,#0FF      ;Set Port D HIGH for 32uSec
        X       A,TEMP          ;Save accumulator in temp location
        CLR     A               ;Clear accumulator
LP6:    INC     A               ;Increment accumulator
        IFEQ    A,#03           ;If accumulator equals three
        JP      LP5             ;then 32uSec done, continue on
        JP      LP6             ;else not done, keep looping
LP5:    CLR     A               ;Clear accumulator
        LD      PORTD,#00       ;Set Port D low
TWO:    X       A,TEMP          ;Restore accumulator
HI1:    IFBIT   2,PORTGP        ;If Bit G2 is high
        JMP     HI              ;then go debounce from High
        JMP     LO              ;else go debounce from Low

                                ;DELAY SUBROUTINE
DELAY:  LD      REG1,#00F       ;Load Reg1 with 0F Hex
LOOP:   DRSZ    REG1            ;Decr Reg1, If Reg1 not equal to 0
        JP      LOOP            ;then keep looping
        RET                     ;else return from delay routine

                                ;DECREMENT THE TIMER BY THE DESIRED DELAY
SUB:    LD      A, T1RALO       ;Load accumulator with value from T1RALO
        SUBC    A,#07D          ;Subtract 7D Hex
        X       A,T1RALO        ;Store result in T1RALO
        LD      A,T1RAHI        ;Load accumulator with value from T1RAHI
```

```
        SUBC    A,#000          ;Subtract zero and borrow (if occurred)
        RC                      ;Reset carry flag
        X       A,T1RAHI        ;Store result in T1RAHI
        RET                     ;Return from subtract routine

                                ;INCREMENT THE TIMER BY THE DESIREd DELAY
ADD:    LD      A, T1RALO       ;Load accumulator with value from T1RALO
        ADC     A,#07D          ;Add 7D Hex
        X       A,T1RALO        ;Store result in T1RALO
        LD      A,T1RAHI        ;Load accumulator with value from T1RAHI
        ADC     A,#000          ;Add zero and carry bit
        RC                      ;Reset carry flag
        X       A,T1RAHI        ;Store result in T1RAHI
        RETSK                   ;Return and skip from add routine

                                ;TIMER Subroutine
TIMER:  SBIT    T1C0,CNTRL      ;Start the timer
LP1:    IFBIT   T1PNDA,PSW      ;If underflow (reload from R1A) occurred
        JP      LP4             ;then go stop the timer
        JP      LP1             ;else keep looping
LP4:    RBIT    T1C0,CNTRL      ;Stop the timer
        RBIT    T1PNDA,PSW      ;Reset the T1 source A pending flag
        RET                     ;Return from timer subroutine
        .END                    ;end of program
```

A.5 ANALOG-TO-DIGITAL CONVERSION USING ON-CHIP COMPARATOR

Some microcontroller applications require a low-cost, but effective way of performing analog-to-digital conversion. A number of techniques for doing this are described in COP NOTE 1: “Analog to Digital Conversion Techniques with COPS Family Microcontrollers” and in Application Note 607: “Pulse Width Modulation A/D Conversion Techniques with COP800 Family Microcontrollers”. These notes may be found in the Embedded Controllers databook. This section explains how the COP888 devices that contain an on-board comparator can be integrated into two of the solutions described in these notes: the single slope A/D conversion technique and the pulse width modulation A/D technique.

Figure A-5 shows the hardware connections for either type of A/D conversion technique. The voltage to be measured, V_{IN}, is connected to the inverting terminal, COMPIN-, of the comparator. The non-inverting terminal, COMPIN+, is connected to an RC network via a current-limiting resistor. For the single slope technique, the comparator output pin, COMPOUT is connected to the Timer T1A input pin. This is not required for the pulse width modulation technique.

The principle of the single slope conversion technique is to measure the time it takes for the RC network to charge up to the voltage level on the inverting terminal, by using Timer T1 in the input capture mode. The cycle count obtained in Timer T1 can be converted into real time if it is scaled by the COP8 clock frequency. If the COP8 is clocked by a crystal, this parameter is known very accurately. Applications connected to the power line using an RC clock can use the line frequency as a reference with which to measure the RC clock. The time measurement is then converted into the voltage, either by direct calculation or by using a suitable approximation.

This very low cost technique is ideally suited to cost-sensitive applications which do not require high accuracy. The pulse width modulation A/D conversion technique will improve the accuracy at the cost of a higher conversion time. Application Note 607 describes this technique in detail.

The accuracy can be improved further by using a low-cost MM74HC4016 to multiplex the analog input voltage with an accurate voltage reference used for calibration. Replacing the resistor in the RC network with a current source will linearize the charging curve, offering better resolution.

The user must ensure that the input voltage supplied to the comparator lies within its input common mode range, which is shown in the characterization curves in the datasheet.

Before the start of conversion, the capacitor must be discharged. The program reconfigures the COMPIN+ pin as an output logic low to perform the discharge. Timer T1 is stopped and configured for input capture mode on a low-to-high transition. The T1A register is cleared and pin T1A set up as a Hi-Z input. The comparator initialization is performed. The conversion begins when timer T1 is started and the COMPIN+ pin is configured back to an input.

The initial value of the comparator is zero. A capture event occurs when the RC voltage rises above the input voltage. If desired, the Timer T1 interrupt can be enabled to produce an interrupt on this capture event. The capture time can then be read and converted into voltage. This measurement technique has a resolution of 8 bits if the value of the timer is scaled to contain 1000 (or more) counts after five RC periods. The accuracy is primarily dependent on the accuracy of the user's estimation of the RC time constant, the offset voltage, and the user's approximation routine.

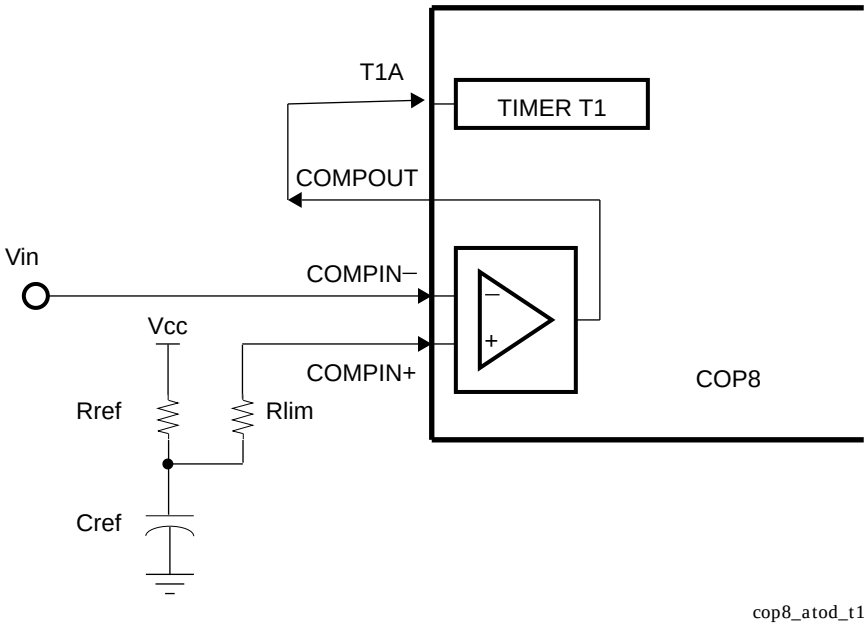


Figure A-5 A/D Conversion Using On-board Comparator and Timer T1

The following code example demonstrates how this technique is implemented in assembly code. In this example, polling the Timer T1A pending flag is used instead of interrupts. The 16-bit timer value is stored in REFHI:REFLO.

```
         ;PORTxC Addr of port config reg assigned comp alt. fun.
         ;PORTxD Addr of port data reg assigned comp alt. fun.
         .INCLD      COP888XX.INC
         .SECT       REGDEC,REG
R0:      .DSB 1
REF:     .DSB 2
         REFLO = REF
         REFHI = REF+1
         .SECT       CONVRT,ROM     ;DISCHARGE THE CAPACITOR
CONVRT:  SBIT        CMPINN,PORTxC  ;Configure COMPIN- as output
         RBIT        CMPINN,PORTxD  ;Set COMPIN- low
DELAY:                              ;Wait for capacitor to discharge
         LD          R0,#020        ;Load register with 20 Hex
DR0:     DRSZ        R0             ;Decrement register, if not equal to zero
         JP          DR0            ;then keep looping, else

                                    ;SET UP THE COMPARATOR AND CONFIGURE T1A
STCMP:   SBIT        CMPOUT,PORTxC  ;Configure COMPOUT pin as an output
         RBIT        CMPINP,PORTxC  ;Configure COMPIN+ pin as an input
         RBIT        CMPINN,PORTxC  ;Configure COMPIN- pin as an input
         LD          PORTxD,#00     ;Configure outputs low and inputs as Hi-Z
         RBIT        T1A,PORTGC     ;Configure T1A pin as an input
         RBIT        T1A,PORTGD     ;Configure T1A pin as Hi-Z

                                    ;PRELOAD TIMER T1
LDTIM:   LD          B,#T1MRLO      ;Load B pointer with T1 address
         LD          [B+],#0FF      ;Load T1 lower byte with FF Hex
         LD          [B],#0FF       ;Load T1 upper byte with FF Hex

                                    ;START TIMER T1 AND CHARGE THE CAPACITOR
STIM:    LD          PSW,#00        ;Disable all interrupts
         LD          CNTRL,#040     ;Configure T1 for input capture mode, positive
                                    T1A

                                    ;WAIT FOR CAPTURE AND SAVE VALUE
WAITC1:  IFBIT       T1PNDA,CNTRL   ;If capture occurred
         JP          STORE1         ;then go store capture value
         JP          WAITC1         ;else keep waiting

STORE1:  RBIT        T1PNDA,CNTRL   ;Reset T1PNDA flag
         LD          A,T1RALO       ;Read T1RA low byte
         X           A,REFLO        ;Save value in REFLO
         LD          A,T1RAHI       ;Read T1RA high byte
         X           A,REFHI        ;Save value in REFHI

                                    ;WAIT FOR 2nd CAPTURE AND COMPUTE ELAPSED TIME
WAITC2:  IFBIT       T1PNDA,CNTRL   ;If capture occurred
         JP          COMP           ;then go compute elapsed time
         JP          WAITC2         ;else keep waiting
COMP:                               ;Compute elapsed time
.ENDSECT                            ;End of example
```

A.6 BATTERY-POWERED WEIGHT MEASUREMENT

Figure A-6 shows the block diagram of a simple weight scale application. This implementation of weight measurement may be used with any COP888 device that has the Multi-input Wakeup feature. The pressure sensor circuit is based on a buffered Wheatstone bridge arrangement. A current source and a capacitor generate the linear ramp for the A/D conversion. A crystal oscillator is required for an accurate time base. A

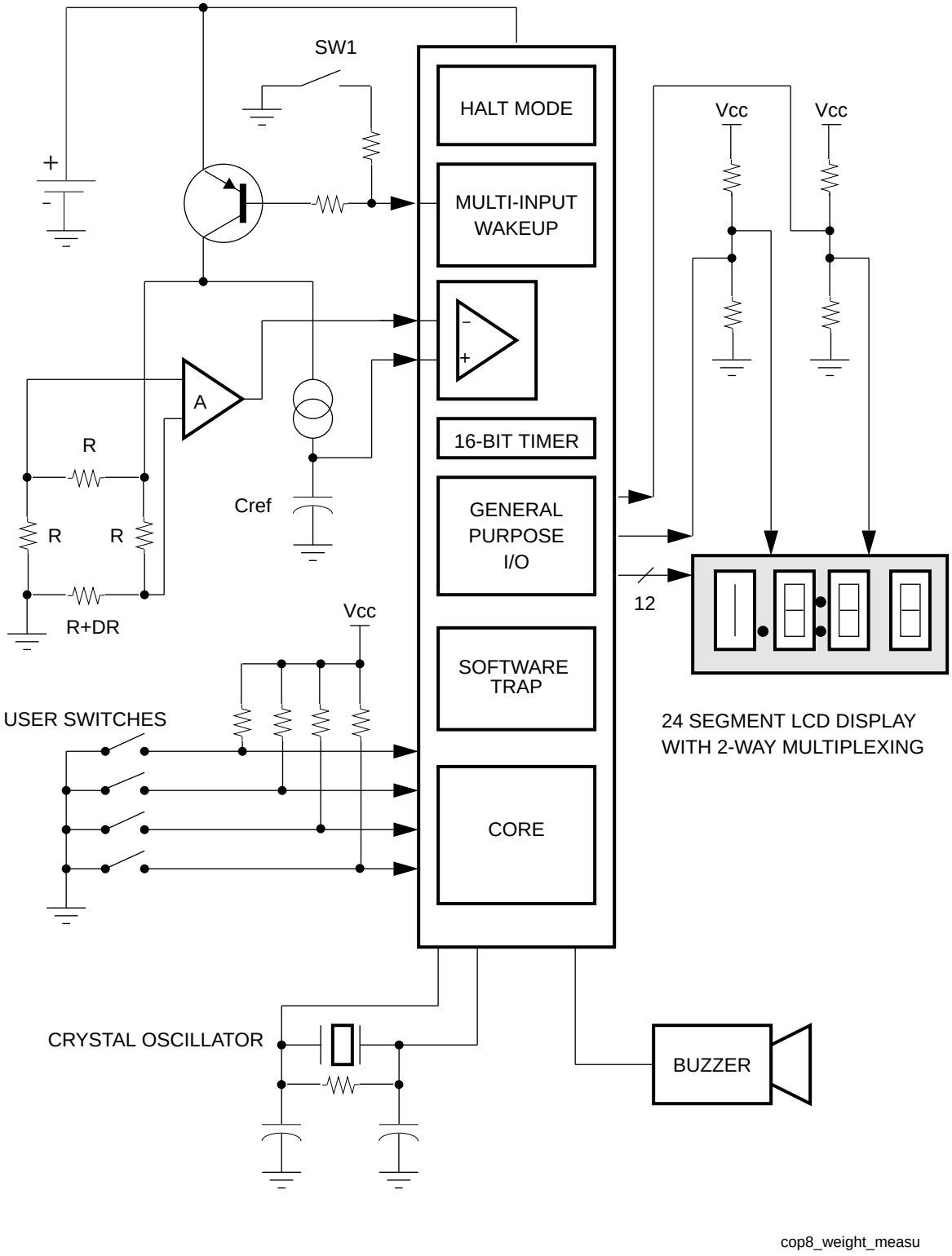


Figure A-6 Battery-powered Weight Measurement

50% duty cycle signal is generated for the audible tone. A 24-segment LCD display indicates the weight to the user. Four inputs are used for configuring the scale.

If the application is not in use, the COP888 is held in HALT mode. As soon as a weight is applied to the system, SW1 closes and the COP exits the HALT mode via a Multi-Input Wakeup pin. The MIWU pin is then reconfigured as an output to power up the sensor circuit, thus power remains even when the switch is open. The measurement and display are then performed. After completing the measurement and display routines, the COP888 reconfigures the sensor power pin as a Wakeup pin, thereby disconnecting power from the sensor circuit. The device then re-enters the HALT mode.

The 16-bit timer can be used to generate the interrupts required to refresh the LCD display. A power-on reset circuit (not shown) is required in this application.

A.7 ZERO CROSS DETECTION

Zero cross detection is often used in appliances connected to the AC power line. The line frequency is a useful time base for applications such as industrial timers or irons which switch off if not used for five minutes. Phase-controlled applications require a consistent timing reference in phase with the line voltage.

The COP888 requires a square wave, magnitude V_{CC}, at the same frequency as the power line voltage, connected to a input port pin for a simple time base. For a phase-control time base, this waveform should preferably be in phase with the line voltage, although control is still possible if there is a predictable, constant phase lag, less than the phase lag introduced by the load. The choice of zero cross detection circuit depends on factors such as cost, the type of power supply used in the appliance, and the expected interference.

The zero cross detection input can either be polled by software or can be connected to the G0 interrupt line. Polling the pin by software is the simplest technique and saves the interrupt for another function, but has the disadvantage that the polling procedure can be interrupted, causing inaccuracies in synchronization. Disabling the interrupt during the polling is not always possible, as the interrupt may be required for the implementation of other features.

Connecting the zero cross detection input to the external interrupt pin guarantees synchronization. It has the additional advantage that a regular interrupt is generated, which could interrupt the processor out of a fault condition. The interrupt routine only needs to test the integrity of the stack to determine whether such a fault has occurred.

The following software example shows how software polling of the zero cross line is implemented.

```
ZCD:
        LD      B,#STATUS       ;Save bytes using the B pointer
        IFBIT   SYNCHRO,[B]     ;If SYNCHRO is 1, wait for a rising edge
        JP      WLOHI           ;otherwise wait for a falling edge.
WHILO:  IFBIT   3,PORTLP        ;Wait for falling edge
        JP      WHILO
        SBIT    SYNCHRO,[B]     ;SYNCHRO = 1, so wait for rising edge
        JP      ENDZCD          ;next time.
WLOHI:  IFBIT   3,PORTLP        ;Wait for a rising edge
        JP      RSYNC
        JP      WLOHI
RSYNC:  RBIT    SYNCHRO,[B]     ;SYNCHRO = 0, so wait for a falling edge
                                ;next time.
ENDZCD:                         ;End of example
```

A.8 INDUSTRIAL TIMER

Figure A-7 shows the block diagram for an industrial timer. The user turns the potentiometer to set the required delay time. When the delay time has elapsed, a load is switched on or off, as selected by the input switches. The time base is derived from the power line using a simple zero cross detection circuit, thereby allowing the use of an inexpensive RC clock instead of a crystal oscillator. There are two indicator diodes and a buzzer.

The A/D conversion routine used by this industrial timer is based on the single slope technique defined in Section A.5, but it has an important difference. Instead of connecting the variable resistor into a voltage divider circuit and measuring the voltage using the single slope technique, the variable resistor forms part of the RC network. The time that the variable RC circuit takes to exceed the fixed reference voltage is directly proportional to the value of the resistor, simplifying the conversion from time into resistance. The circuit as shown can be used to program a time proportional to the angle of the potentiometer setting. The potentiometer can be replaced by a rotary switch connected to a series of resistors, so that each position of the switch generates a different resistance. Here the COP888 can identify the switch positions if the difference in each resistance for each position is greater than the inaccuracy in measuring the absolute resistance.

A.9 PROGRAMMING EXAMPLES

This section is intended to be an overview of programming examples. For more detailed and varied programming examples, refer to the Microcontroller Databook or call the Customer Response Center at 1-800-272-9959 (also see Section A.14).

A.9.1 Clear RAM

The following program clears all RAM locations in the base segment except for the stack pointer. The value of the argument to IFBNE may need to be adjusted, depending on the size of RAM in specific family members.

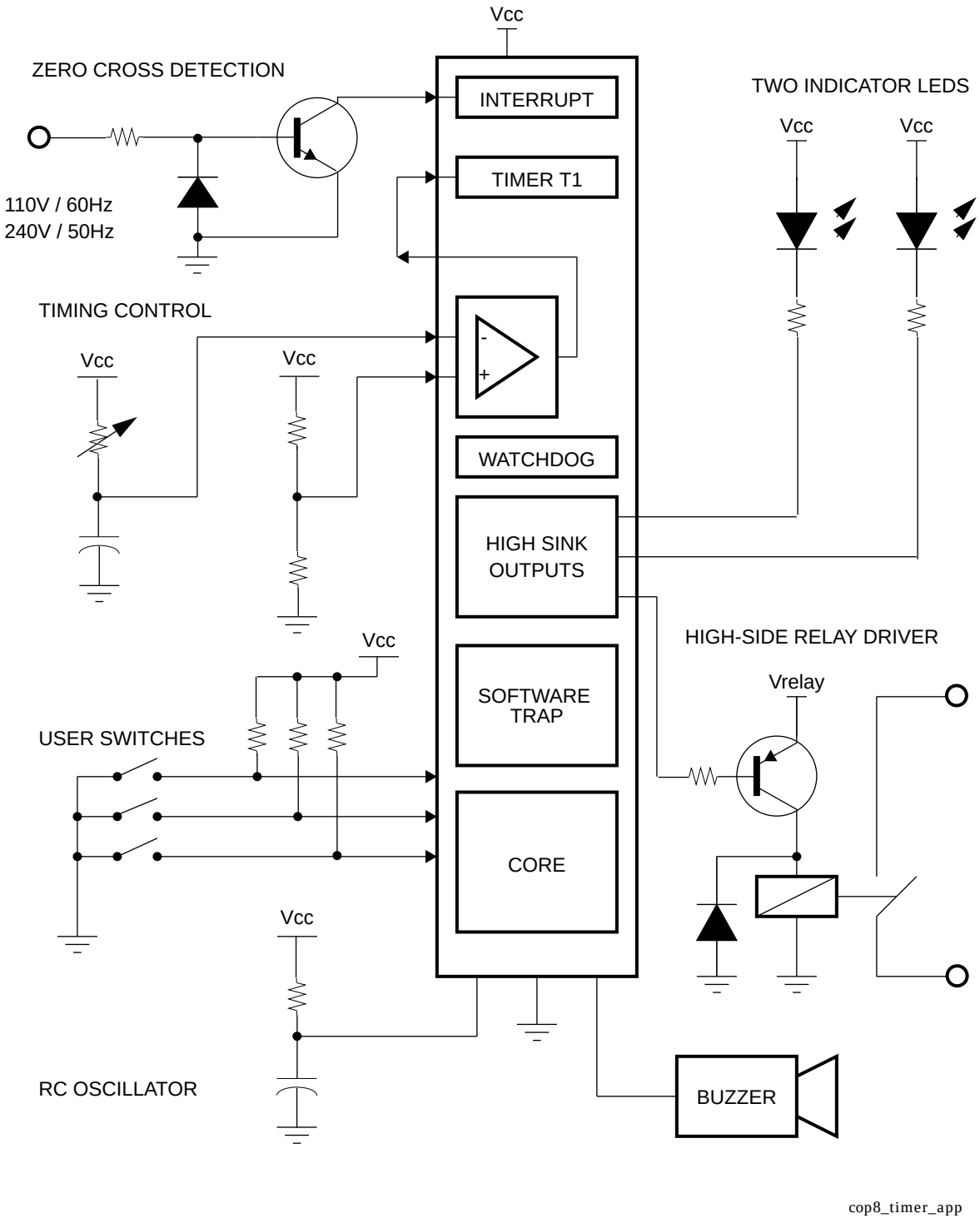


Figure A-7 Industrial Timer Application

```
COP888 PROGRAM TO CLEAR ALL RAM EXCEPT SP

CLRAM:   LD       0FC,#070    ;Define X-pointer as counter
         LD       B,#0        :Initialize B pointer
CLRAM2:  LD       [B+],#0     ;Load mem with 0 and incr B pointer
         DRSZ     0FC         ;Decrement counter
         JP       CLRAM2      ;Skip if lower half RAM is cleared
         LD       B,#0F0      ;Point B to upper half of RAM
CLRAM3:  LD       [B+],#0     ;Load upper RAM half with 0
         IFBNE    #0D         ;until B points to 0FD (=SP)
         JP       CLRAM3      ;Skip if B=0FD
         LD       B,#0        ;Initialize B to 0
```

A.9.2 Binary/BCD Arithmetic Operations

The arithmetic instructions include the Add (ADD), Add with Carry (ADC), Subtract with Carry (SUBC), Increment (INC), Decrement (DEC), Decimal Correct (DCOR), Clear Accumulator (CLR), Set Carry (SC), and Reset Carry (RC). The shift and rotate instructions, which include the Rotate Right through Carry (RRC), the Rotate Left through Carry (RLC), and the Swap accumulator nibbles (SWAP), may also be considered arithmetic instruction variations. The RRC instruction is instrumental in writing a fast multiply routine.

In subtraction, a borrow is represented by the absence of a Carry and vice versa. Consequently, the Carry flag needs to be set (no borrow) before a subtraction, just as the Carry flag is reset (no carry) before an addition. The ADD instruction does not use the Carry flag as an input. It should also be noted that both the Carry and Half Carry flags (Bits 6 and 7, respectively, of the PSW control register) are cleared with RESET and remain unchanged with the ADD, INC, DEC, DCOR, CLR, and SWAP instructions. The DCOR instruction uses both the Carry and Half Carry flags. The SC instruction sets both the Carry and Half Carry flags, while the RC instruction resets both these flags.

The following program examples illustrate additions and subtractions of 4-byte data fields in both binary and BCD (Binary Coded Decimal). The four bytes from data memory locations 24 through 27 are added to or subtracted from the four bytes in data memory locations 16 through 19. The results replace the data in memory locations 24 through 27.

These operations are performed both in binary and BCD. It should be noted that the BCD preconditioning of adding (ADD) the hexadecimal value 66 is necessary only with the BCD addition, not with the BCD subtraction. The binary coded decimal DCOR (Decimal Correct) instruction uses both the Cary and Half Carry flags as inputs but does not change the Carry and Half Carry flags. Also note that the #12 with the IFBNE instruction represents 28 minus 16, since the IFBNE operand is modulo 16 (remainder when divided by 16).

```
BINARY ADDITION
         LD       X,#16    ;No leading zero indicates decimal
         LD       B,#24
         RC
LOOP:    LD       A,[X+]
         ADC      A,[B]
         X        A,[B+]
         IFBNE    #12
         JP       LOOP
         IFC
         JP       OVFLOW   ;OverFLow if C
```

```
BINARY SUBTRACTION
        LD      X,#010  ;Leading zero indicates hex
        LD      B,#018
        SC
LOOP:   LD      A,[X+]
        SUBC    A,[B]
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFNC
        JP      NEGRSLT ;Neg. result if no C (No C = Borrow)

BCD ADDITION
        LD      X,#010  ;Leading zero indicates hex
        LD      B,#018
        RC
LOOP:   LD      A,[X+]
        ADD     A,#066  ;Add hex 66
        ADC     A,[B]
        DCOR    A       ;Decimal correct
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFC
        JP      OVFLOW  ;Overflow if C

BCD SUBTRACTION
        LD      X,#16   ;No leading zero indicates decimal
        LD      B,#24
        SC
LOOP:   LD      A,[X+]
        SUBC    A,[B]
        DCOR    A       ;Decimal correct
        X       A,[B+]
        IFBNE   #12
        JP      LOOP
        IFNC
        JP      NEGRSLT ;Neg. result if no C (No C = Borrow)
```

Note that the previous additions and subtractions are not “adding machine” type arithmetic operations in that the result replaces the second operand rather that the first. The following program examples illustrate “adding machine” type operations where the result replaces the first operand. With subtraction, this entails the result replacing the minuend rather that the subtrahend.

```
BINARY ADDITION
        LD      B,#16
        LD      X,#24
        RC
LOOP:   LD      A,[X+]
        ADC     A,[B]
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFC
        JP      OVFLOW   ;Overflow if C

BINARY SUBTRACTION
        LD      B,#010
        LD      X,#018
        SC
LOOP:   LD      A,[X+]
        X       A,[B]
        SUBC    A,[B]
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFNC
        JP      NEGRSLT  ;Neg. result if no C (No C = Borrow)

BCD ADDITION
        LD      B,#010
        LD      X,#018
        RC
LOOP:   LD      A,[X+]
        ADD     A,#066
        ADC     A,[B]
        DCOR    A
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFC
        JP      OVFLOW   ;Overflow if C

BCD SUBTRACTION
        LD      B,#16
        LD      X,#24
        SC
LOOP:   LD      A,[X+]
        X       A,[B]
        SUBC    A,[B]
        DCOR    A
        X       A,[B+]
        IFBNE   #4
        JP      LOOP
        IFNC
        JP      NEGRSLT  ;Neg. result if no C (No C = Borrow)
```

The following hybrid arithmetic example adds five successive bytes of a data table in program memory to a two-byte SUM, and then subtracts the SUM from a two-byte total TOT. Assume that the table is located starting a program memory address 0401, while SUM and TOT are at RAM data memory locations 1, 0 and 3, 2, respectively. The program is encoded as a subroutine.

```
          .SECT    MATH,RAM
MATHMEM:  .DSB4                ;CONSTANT DECLARATIONS
          SUMLO = MATHMEM      ;Sum lower byte storage location
          SUMHI = MATHMEM+1    ;Sum upper byte storage location
          TOTLO = MATHMEM+2    ;Total lower byte storage location
          TOTHI = MATHMEM+3    ;Total upper byte storage location
          .SECT    CODE, ROM, ABS=0401
                               ;ROM TABLE
          .BYTE    102         ;Store 102
          .BYTE    41          ;Store 41
          .BYTE    31          ;Store 31
          .BYTE    26          ;Store 26
          .BYTE    5           ;Store 5
                               ;PERFORM ADDITION AND SUBTRACTION
ARITH1:   LD       X,#5        ;Set up ROM table pointer
          LD       B,#SUMLO    ;Set up sum pointer
LOOP:     RC                   ;Reset carry flag
          LD       A,X         ;Load ROM pointer into accumulator
          LAID                 ;Read data from ROM
          ADC      A,[B]       ;Add SUMLO to ROM value
          X        A,[B+]      ;Store result in SUMLO, point to SUMHI
          CLR      A           ;Clear accumulator
          ADC      A,[B]       ;Add SUMHI and carry bit to the accumulator
          X        A,[B-]      ;Store result in SUMHI, point to SUMLO
          DRSZ     X           ;Decrement ROM pointer, If not equal to zero
          JP       LOOP        ;then repeat the loop
          SC                   ;else set the carry flag
          LD       B,#2        ;Load B pointer with 2 (point to TOTLO)
LUP:      LD       A,[X+]      ;Load accumulator with subtrahend
          X        A,[B]       ;Reverse operands for subtraction
          SUBC     A,[B]       ;Subtract
          X        A,[B+]      ;Increment minuend pointer
          IFBNE    #4          ;If B pointer not equal to 4
          JP       LUP         ;then repeat the loop
          RET                  ;else return
```

A.9.3 Binary Multiplication

The following program listing shows the code for a 16-by-16-bit binary multiply subroutine. The multiplier starts in the lower 16 bits of the 32-bit result location. As the multiplier is shifted out of the low end of the result location with the RRC instruction, each multiplier bit is tested in the Carry flag. The multiplicand is conditionally added (depending on the multiplier bit) into the high end of the result location, after which the partial product is shifted down one bit position following the multiplier. Note that one additional terminal shift cycle is necessary to align the result.

```
COP888 MULTIPLY (16X16) SUBROUTINE
MULTIPLICAND IN [1,0] MULTIPLIER IN [3,2]
PRODUCT IN [5, 4, 3, 2]

         .SECT   MEMCNT, REG
CNTR:    .DSB 1
         .SECT   CODE, ROM
MULT:    LD      CNTR,#17
         LD      B,#4
         LD      [B+],#0
         LD      [B],#0
         LD      X,#0
         RC
MLOOP:   LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B-]
         LD      A,[B]
         RRC     A
         X       A,[B]
         LD      B,#5
         IFNC
         JP      TEST
         RC
         LD      B,#4
         LD      A,[X+]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[X-]
         ADC     A,[B]
         X       A,[B]
TEST:    DRSZ    CNTR
         JP      MLOOP
         RET
```

A.9.4 Binary Division

The following program shows a subroutine for a 16-by-16-bit binary division. A 16-bit quotient is generated along with a 16-bit remainder. The dividend is left shifted up into an initially-cleared 16-bit test window where the divisor is test-subtracted. If the test subtraction generates no high-order borrow, then the real subtraction is performed with the result stored back in the test window. At the same time, a quotient bit (equal to 1) is inserted into the low end of the dividend window to record that a real subtraction has taken place. The entire dividend and test window is then shifted up (left shifted) one bit position with the quotient following the dividend.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated as straight-line code rather than a loop in order to optimize throughput time.

```
COP888 DIVIDE (16X16)
SUBROUTINE
DIVIDEND IN [3,2]
DIVISOR IN [1,0]
QUOTIENT IN [3,2]
REMAINDER IN [5,4]

         .SECT   MEMCNT, REG
CNTR:    .DSB 1
         .SECT   CODE, ROM
DIV:     LD      CNTR,#16
         LD      B,#5
         LD      [B-],#0
         LD      [B],#0
         LD      X,#4
LSHFT:   RC
         LD      B,#2
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
TSUBT:   SC
         LD      B,#0
         LD      A,[X+]
         SUBC    A,[B]
         LD      B,#1
         LD      A,[X-]
         SUBC    A,[B]
         IFNC
         JP      TEST
SUBT:    LD      B,#0
         LD      A,[X]
         SUBC    A,[B]
         X       A,[X+]
         LD      B,#1
         LD      A,[X]
         SUBC    A,[B]
         X       A,[X-]
         LD      B,#2
         SBIT    0,[B]
TEST:    DRSZ    CNTR
         JMP     LSHFT
         RET
```

With a division where the dividend is larger than the divisor (relative to the number of bytes), an additional test step must be added. This test determines whether a high-order carry is generated from the left shift of the dividend through the test window. When this carry occurs, the program branches directly to the SUBT subtract routine. This carry can occur only if the divisor contains a high-order bit. Moreover, the divisor must also be larger than the shifted dividend when the shift has placed a high-order bit in the test window. When this case occurs, the TSUBT test subtract shows the divisor to be larger than the shifted dividend and no real subtraction occurs. Consequently, the high-order bit of the shifted dividend is again left shifted and results in a high-order carry. This test is illustrated in the following program for a 24-by-8-bit binary division.

Note that the four left shifts (LD, ADC, X) in the LSHFT section of the program are repeated with the JP jump to LUP instruction in order to minimize program size.

```
COP888 DIVIDE (24X8) SUBROUTINE
DIVIDEND IN [2,1,0]
DIVISOR IN [4]
QUOTIENT IN [2,1,0]
REMAINDER IN [3]

         .SECT   MEMCNT, REG
CNTR:    .DSB 1
         .SECT   CODE, ROM
DIV:     LD      CNTR,#24
         LD      B,#3
         LD      [B],#0
LSHFT:   RC
         LD      B,#0
LUP:     LD      A,[B]
         ADC     A,[B]
         X       A,[B+]
         IFBNE   #4
         JP      LUP
         IFC
         JP      SUBT
TSUBT:   SC
         LD      B,#3
         LD      A,[B+]
         SUBC    A,[B]
         IFNC
         JP      TEST
SUBT:    LD      A,[B-]
         X       A,[B]
         SUBC    A,[B]
         X       A,[B]
         LD      B,#0
         SBIT    0,[B]
TEST:    DRSZ    CNTR
         JMP     LSHFT
         RET
```

A.10 EXTERNAL POWER WAKEUP CIRCUIT

Power-on wakeup is a technique used in battery powered applications such as electronic keys or digital scales to save battery power. Instead of using the HALT mode when the application is not in use, the COP device is powered off. If there is only one input switch in the application, the implementation is simple. This switch is put in series with the battery, providing power to the circuit when the switch is closed.

If there is more than one switch, power-on wakeup can be achieved by using an NPN transistor and one resistor per switch as shown in Figure A-8. Here, the circuit ground is connected to the battery negative terminal via the NPN transistor. If the base is floating, it will not conduct. If the base is pulled to V_{CC} via a current-limiting resistor, it will conduct, powering up the circuit.

An alternative technique is shown in Figure A-6. Here the positive terminal of the battery is connected to the V_{CC} line via a switch, a diode and two resistors per line. If a switch is pressed, power is applied to the V_{CC} line. The pull-down resistors pull any ports connected to open switches to ground. If the switch is closed, the voltage on the switch will be V_{CC} plus the diode voltage drop. If this potential were directly applied to the Port

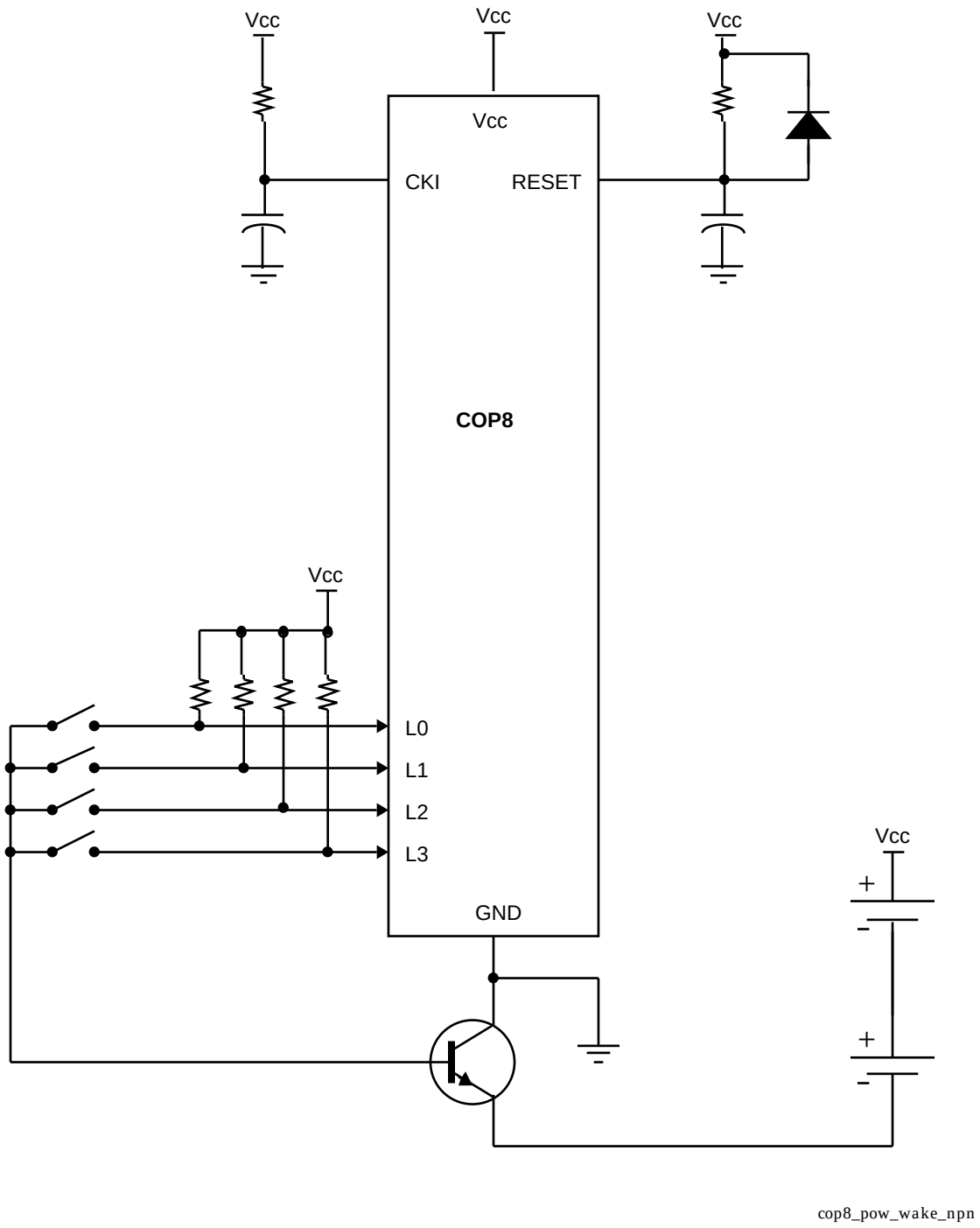


Figure A-8 Power Wakeup Using An NPN Transistor

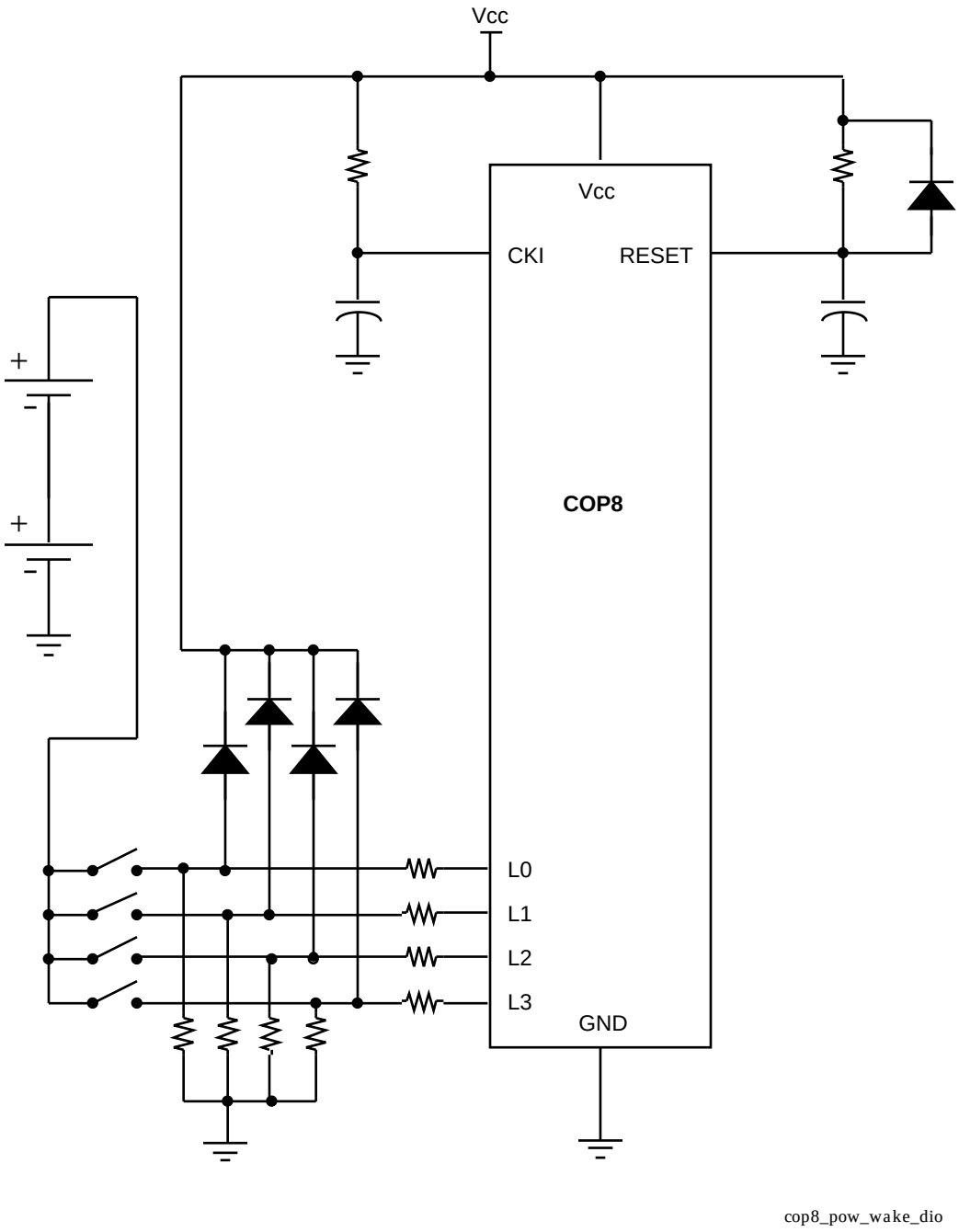


Figure A-9 Power Wakeup Using Diodes And Resistors

L pin, the COP device would be driven outside the operating specification. Therefore, series protection resistors are used on all Port L pins connected to the switches.

A.11 WATCHDOG RESET CIRCUIT

The following circuit is recommended for connecting the $\overline{\text{WDOUT}}$ pin to the $\overline{\text{RESET}}$ pin (Figure A-10). This circuit guarantees that the COP888 receives a valid reset signal as the result of a Watchdog/Clock Monitor error. With this circuit, the $\overline{\text{RESET}}$ is held low a minimum of 16 instruction cycles even if C_1 fails to discharge completely. The 2.5kΩ resistor limits the current into the $\overline{\text{WDOUT}}$ pin. This prevents damage to the $\overline{\text{WDOUT}}$ pin during the discharge of C_1. The components R_1 and C_1 are chosen to ensure a reset risetime of five times the power supply risetime.

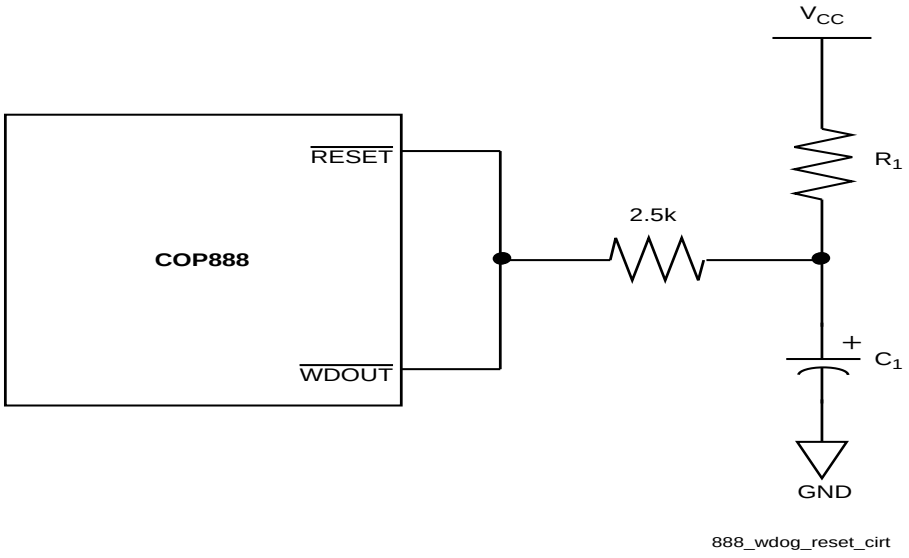


Figure A-10 Watchdog Reset Circuit

A.12 INPUT PROTECTION ON COP888 PINS

The COP888 input pins have internal circuitry for protection from ESD. The internal circuitry is shown in Figures A-11 and A-12.

The input protection circuitry is implemented with the P_channel transistors. The equivalent diode circuit is shown in Figure A-13.

When the inputs are tri-stated and the input voltage on the pin is between GND and V_{CC}, the input protection diodes are off. The only current drawn into or out of the pin is leakage current. If the input is expected to be below GND and/or above V_{CC}, an external series resistor must be used to limit the input current below the maximum allowable current.

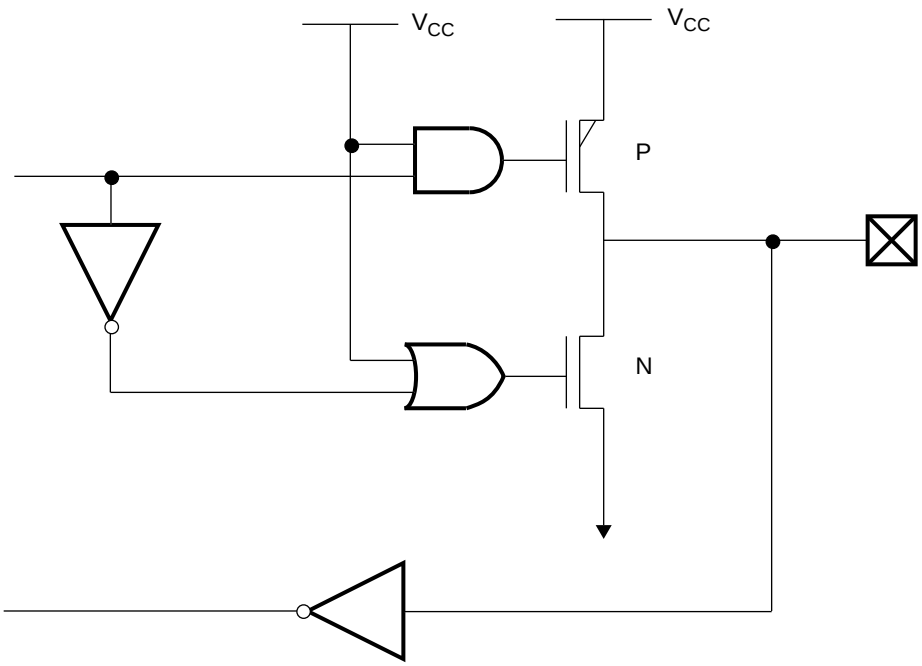


Figure A-11 Ports L/C/G Input Protection (Except G6)

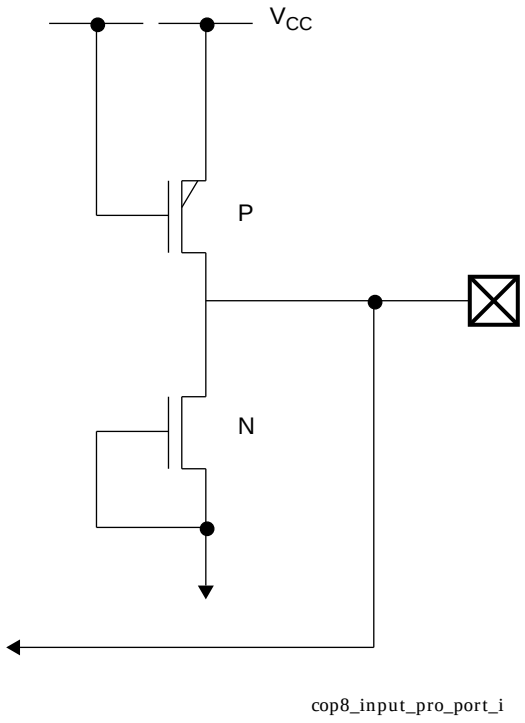


Figure A-12 Port I Input Protection

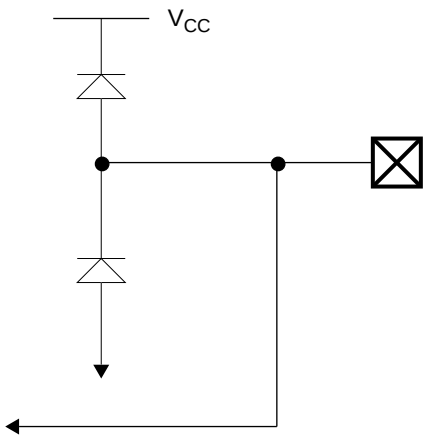


Figure A-13 Diode Equivalent of Input Protection

In addition to limiting the input current to below the maximum latchup spec (specified in the datasheet), the user should also consider the fact that drawing excessive continuous current into the pin, even though below the maximum latchup current, may cause overstress.

A typical example of drawing continuous current is in an automotive application where the ignition signal (battery) is connected to an input pin through a series resistor. Assuming a 100K series resistor with a tolerance of ±10%, the worst case resistor value is 90K. The battery voltage is assumed to be 12V for normal operation and 24V for a "jump start." The high voltage applied to the pin causes the on-chip protection diode to be forward biased, resulting in current into the associated V_{CC} metal trace. Based on a diode threshold voltage of 0.6V, the voltage at the pin will be V_{CC} + 0.6V. Based on a V_{CC} value of 5V, the input current can be calculated as follows:

Normal Operation:

$$\text{Input current} = \frac{[12 - (5 + 0.6)]}{90K} = 71\mu A$$

Jump Start:

$$\text{Input current} = \frac{[24 + (5 + 0.6)]}{90K} = 204\mu A$$

A study of the internal circuitry indicates that the input pin can draw about 200 µA without causing any damage or reliability problem.

Another approach is to use appropriate external circuitry that prevents the input protection diodes from being biased. An example is shown in Figure A-14.

The resistors are required to drop the +12V and the diode prevents the –12V from being applied to the pin.

For V_I = 12V±5% and V_{CC} = 5V ± 5%, the resistor values are calculated to be:

R_1 = 47K +5%

R_2 = 82K ±5%

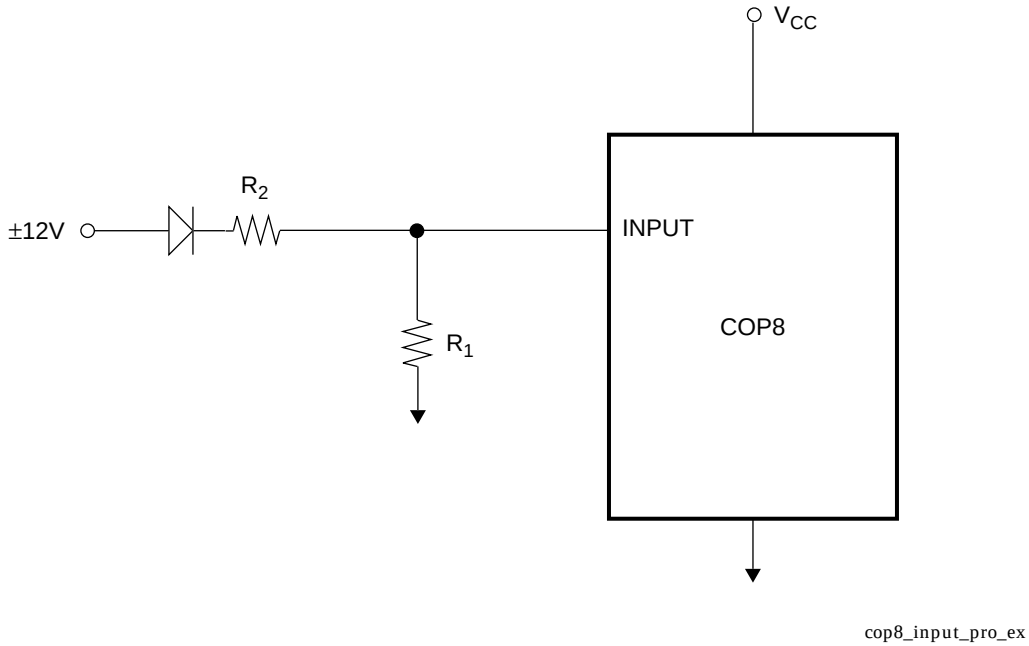


Figure A-14 External Protection of Inputs

This analysis does not apply to G6, $\overline{\text{RESET}}$, and CKI which do not have the protection diodes. Implementation of the above circuit will result in a V_{IH} that is between 0.7 V_{CC} and V_{CC}, and a V_{IL} that is between V_{SS} (0V) and 0.2 V_{CC}.

A.13 ELECTROMAGNETIC INTERFERENCE (EMI) CONSIDERATIONS

A.13.1 Introduction

CMOS has become the technology of choice for the processors used in many embedded systems due to its capability for low standby power consumption. However, CMOS is prone to high current transients on the power supply as the internal logic switches. These transients can easily be the source of high-frequency emissions from the system. The system designer should anticipate and minimize unwanted electromagnetic interference (EMI).

A.13.2 Emission Predictions

"EMI in a typical electronic circuit is generated by a current flowing in a loop configured within the circuit. These paths can be either V_{CC}-to-GND loops or output-to-GND loops. EMI generation is a function of several factors. Transmitted signal frequency, duty cycle, edge rates, and output voltage swings are the major factors of the resultant EMI levels."[1]

1. "FACT™ Advanced CMOS Logic Databook", National Semiconductor, 1993

The formula for predicting the Electric Field emissions from such a loop is as follows:

$$|E|_{MAX} = \frac{1.32 \times 10^{-3} IA(Freq)^2}{D} \times \left(1 + \left(\frac{\lambda}{2\pi D}\right)^2\right)^{1/2}$$

where:

- $|E|_{MAX}$ is the maximum E-field in the plane of the loop in μV/m
- I is the current amplitude in milliamps
- A is the loop area in square cm
- λ is the wavelength at the frequency of interest in meters
- D is the observation distance in meters
- $Freq$ is the frequency in MHz
- and the perimeter of the loop $P << \lambda$.

Applying this equation to a single standard output for a National Semiconductor Microcontroller, and performing a Fourier analysis of the output switching at a frequency of 20 MHz, yields the results shown in Table A-1. These calculations assume a trace length of 5 inches, a board thickness of 0.062 inches and a full ground plane. The load capacitance is 100 pf.

Table A-1 Electric Field Calculation Results

Harmonic (MHz)	Current (mA)	$\|E\|_{Max}$ (μV/M)	$\|E\|_{Max}$ (dBμV/M)
20	37.56	8.3	18.4
40	3.66	0.3	-10.2
60	26.13	44.2	33.0
80	4.44	0.6	-4.4
100	16.82	80.2	38.1
120	4.71	2.0	6.0
140	11.21	104.0	40.4
160	4.86	5.8	15.2
180	7.82	127.4	42.1

Note that the assumption is made that the output is switching at 20 MHz, which is rarely the case for a port output. There is noise, however, on the output at these frequencies due to switching within the device. This is the noise which is coupled to the output through V_{CC} and GND. Another point to keep in mind is that rarely does one single output switch,

but usually several at one time, thus adding the effective magnetic fields from all the outputs which are switching.

Accurate analysis requires characterization of the noise present at the output due to V_{CC} or GND noise which is dependent on many factors, including internal peripherals in use, execution code, and address of memory locations in use.

A.13.3 Board Layout

There are two primary techniques for reducing emissions from within the application. This can be done either by reducing the noise or by controlling the antenna. Control of the antenna is accomplished through careful PC board layout.

General

Standard good PC layout practices will go a long way toward reducing emissions. Traces carrying large AC currents (such as signals with fast transition times, that drive large loads) should be kept as short as possible. Traces that are sensitive to noise should be surrounded by ground to the greatest extent possible. Ground and V_{CC} traces should be kept as short and wide as possible to reduce the supply impedance.

Ground Plane

One of the most effective ways to control emissions through board layout is with a ground plane. The use of a plane can help by providing a return path for fast switching signals, thus reducing loop size for both power and signals.

Multilayer Board

The best way to provide a ground plane is through the use of a multilayer printed circuit board. The large area and the proximity of the V_{CC} and GND planes provide additional decoupling for the power, and provide effective return paths for both power and signals.

The problem with the use of a multilayer board, particularly in consumer related industries, is cost. Due to the volumes involved, an addition of several dollars to the cost of an item may be prohibitive.

A.13.4 Decoupling

Control of the emitted noise can be accomplished by several techniques, including decoupling, reduced power supplies, and limitation of signal strength by the addition of series resistance.

It is important to take the time to properly design the decoupling for CMOS processors. Two decoupling techniques can and should be used to minimize both voltage and current switching noise in the system.

Capacitive Decoupling

Capacitive decoupling is commonly used to control voltage noise on the V_{CC} and GND lines of the board, but if the decoupling is properly designed and is kept as close as possible to the power pins of the device, it can also reduce the effective loop area and thus the antenna efficiency. Capacitive decoupling can prevent high-frequency current transients from being seen by the power supply.

One factor of capacitive decoupling which is often overlooked is the frequency response of the capacitors. Each capacitor, dependent on value, lead length, and dielectric material, possesses a series resonant frequency beyond which the device has inductive characteristics. This inductance inhibits the capacitor from responding quickly to the current needs of the processor and forces the current to use the longer path back to the main power supply.

These inductive characteristics can be countered by the addition of extra capacitors of different values in parallel with the original device. As the value of the capacitor decreases (for capacitors of similar manufacture), the resonant frequency increases.

Placing multiple decoupling capacitors across the power pins of the processor can effectively improve the high frequency performance of the decoupling network. Capacitance values are normally selected which are separated by a decade. However, it is best to check the specifications of the capacitors which are used.

Inductive Decoupling

Another very effective method of decoupling which is rarely used is inductive decoupling. The proper placement of ferrite beads between the decoupling capacitors and the processor can significantly reduce the current noise on the power pins.

The use of inductive decoupling, which will increase the series impedance of the power supply, appears to be contradictory to the effect of capacitive decoupling. However, the purpose of inductive decoupling is to force nodes internal to the processor, which are not switching, into providing the charge for the nodes which are switching.

Ferrite beads are very effective for this type of decoupling due to their lossy nature. Rather than storing the energy and returning it to the circuit later, ferrites will dissipate the energy as a resistor.

One should be aware of potential repercussions from the use of any type of series isolation from the power supply. Due to the reduced V_{CC} which may be present during switching transients, interfacing to other devices in the system may be a problem. Since the V_{CC} should only be reduced for the duration of the switching transient, this should only be a problem if the other devices have especially sensitive and fast-responding inputs.

A.13.5 Output Series Resistance

The addition of resistance in series with outputs can have a significant effect on the emissions caused by the switching of the outputs.

Outputs that drive large capacitive loads can have a lot of current flowing when they switch. While the series resistance may slow the switching speed of the node and thus affect the propagation delay, it can also have a large effect on emissions by reducing the amplitude of the current spike that charges or discharges the load.

A.13.6 Oscillator Control

One very definite source of emissions is the system clock. The oscillator is intended to switch at high speed and therefore will emit some noise. Keeping the circuit loop of the oscillator as small as possible will help considerably.

Ceramic resonators are available with the capacitive load included in a single three terminal package. The use of these devices and placing them right next to the processor can reduce emissions as much as 10 dB.

RC oscillators are particularly troublesome for emissions due to the high transient current when the processor turns on the N-channel device that discharges the capacitor. The transistor is meant to be large and to turn on strongly in order to discharge the capacitor as quickly as possible. This allows simple control over the frequency of oscillation but causes difficulty for the designer of systems for EMI-sensitive applications.

A.13.7 Mechanical Shielding

A last resort for controlling emissions is the addition of mechanical shielding. While shielding can be effective and can be easier from an electrical design standpoint, the implementation and installation of a proper electromagnetic shield can be excessively costly and time consuming.

It is much better to design the system with the control of emissions in mind from the start rather than to apply bandages when it is time to begin production.

A.13.8 Conclusion

While electromagnetic emissions can be a problem for the designer of any electronic system, it is particularly troublesome in the design of high speed CMOS systems. With knowledge of the primary sources of noise, and the ways to combat that noise, it is possible to design and build systems which are electromagnetically quiet.

Very few references to specific values of capacitance, resistance, or inductance have been made in this document. The reason for this is that a value which works well in one application may not be effective in another. The best way to determine the values which will work well for a particular application is by experimentation.

A.14 DAIL-A-HELPER SERVICE

The Dial-A-Helper is a service provided by the Microcontroller Applications group. The Dial-A-Helper is an Electronic Information System that may be accessed as a Bulletin Board System (BBS) via data modem, as an FTP site on the Internet via standard FTP client application or as an FTP site on the Internet using a standard Internet browser such as Netscape or Mosaic.

The Dial-A-Helper system provides access to an automated information storage and retrieval system. The system capabilities include a MESSAGE SECTION (electronic mail, when accessed as a BBS) for communications to and from the Microcontroller Applications Group and a FILE SECTION which consists of several file areas where valuable application software and utilities could be found.

DIAL-A-HELPER BBS via a Standard Modem

Voice:	(800) 272-9959	
Modem:	CANADA/ U.S.:	(800) NSC-MICRO (800) 672-6427
	EUROPE:	(+49) 0-8141-351332
	Baud:	14.4k
	Set-Up:	Length: 8-Bit
		Parity: None
		Stop Bit: 1
	Operation:	24 Hours, 7 Days

DIAL-A-HELPER via FTP

ftp nscmicro.nsc.com	
user:	anonymous
password:	username@yourhost.yoursite.yourdomain

DIAL-A-HELPER via a WorldWide Web Browser

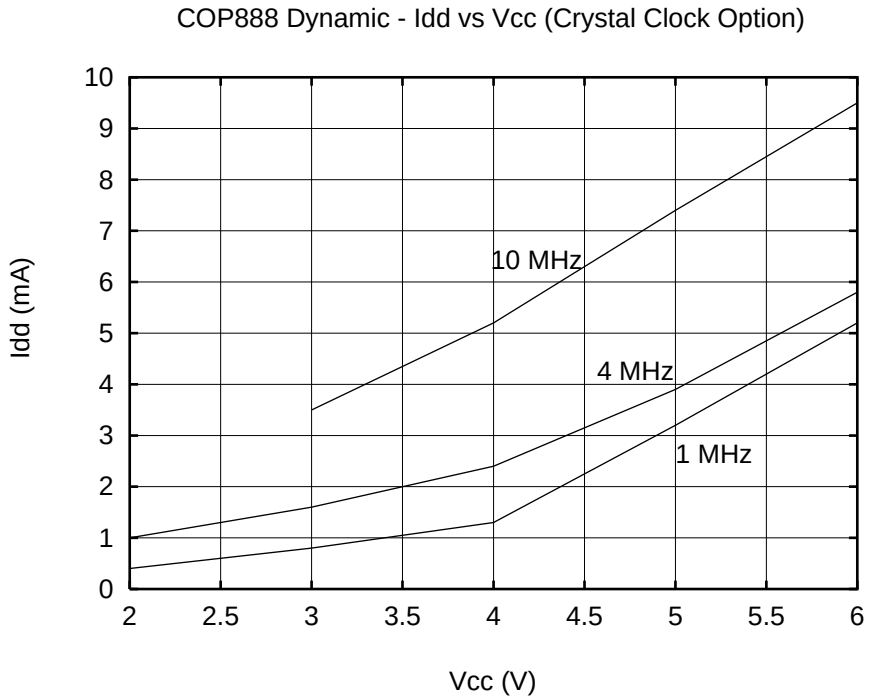
ftp://nscmicro.nsc.com

Appendix B

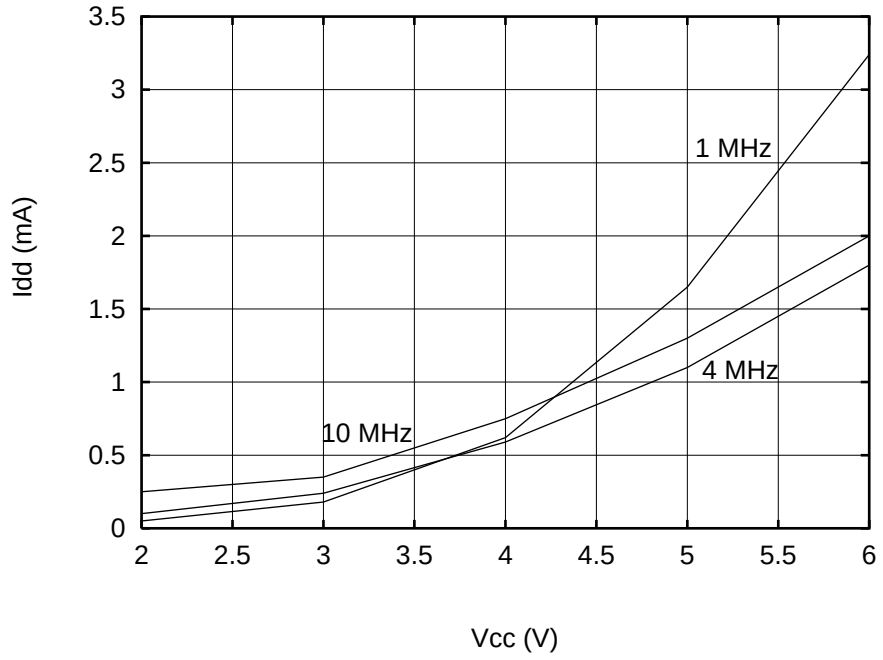
ELECTRICAL CHARACTERIZATION DATA

This appendix presents characterization data for the COP888 Feature Family members. All graphs in this appendix apply to the entire COP888 Feature Family unless otherwise noted.

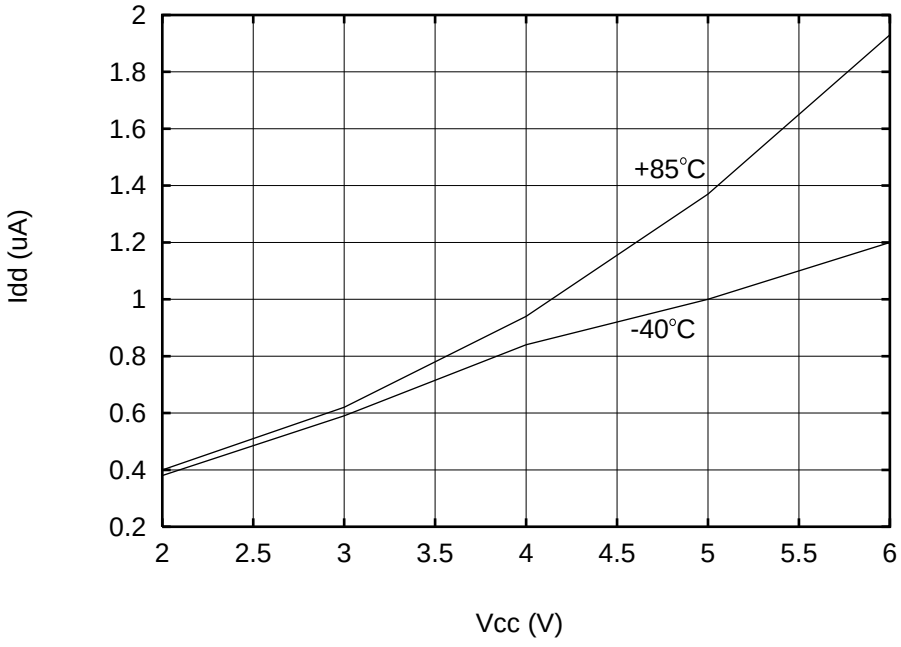
Characterization data is information gained from testing a wide range of sample of parts. Most tests are performed over the full temperature and operating voltage range of the COP888 devices. All information provided in the graphs represents typical values. Most parts will meet these typical values. However, National Semiconductor does not guarantee these values on all parts. Guaranteed numbers are provided in the AC and DC Electrical Characteristics tables found in every datasheet. Guaranteed numbers are tested on every device shipped to our customers.



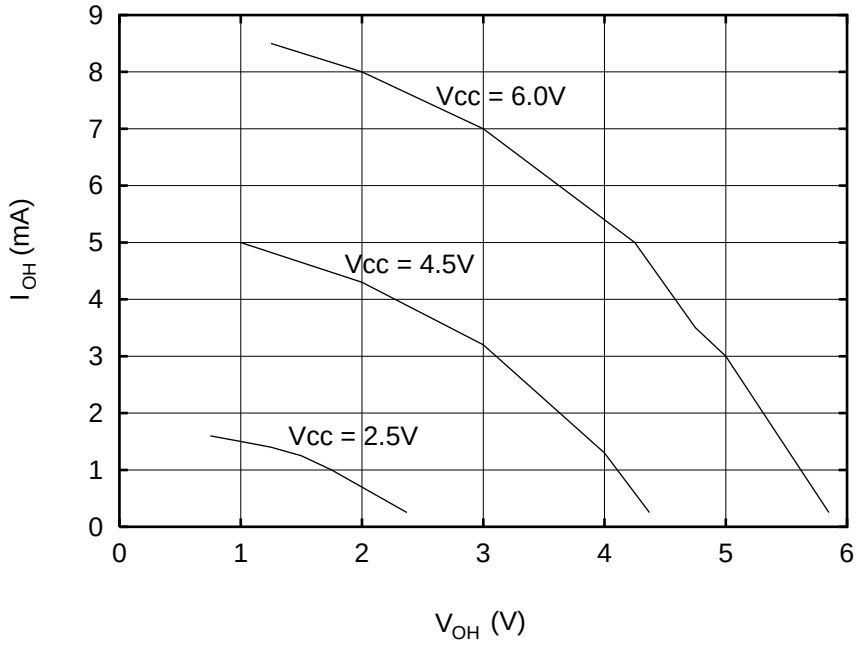
COP888 Idle - Idd vs Vcc (Crystal Clock Option)



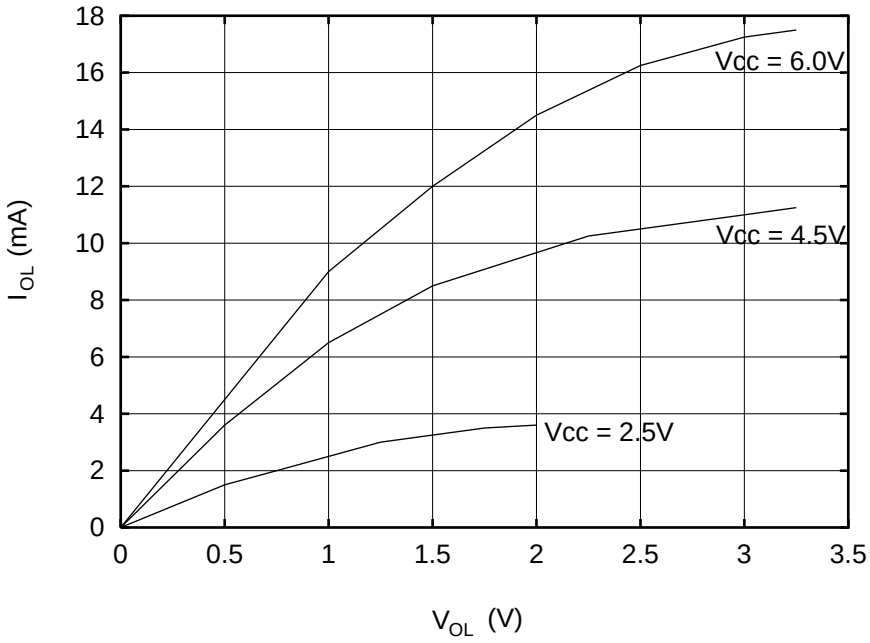
COP888 Halt - Idd vs Vcc



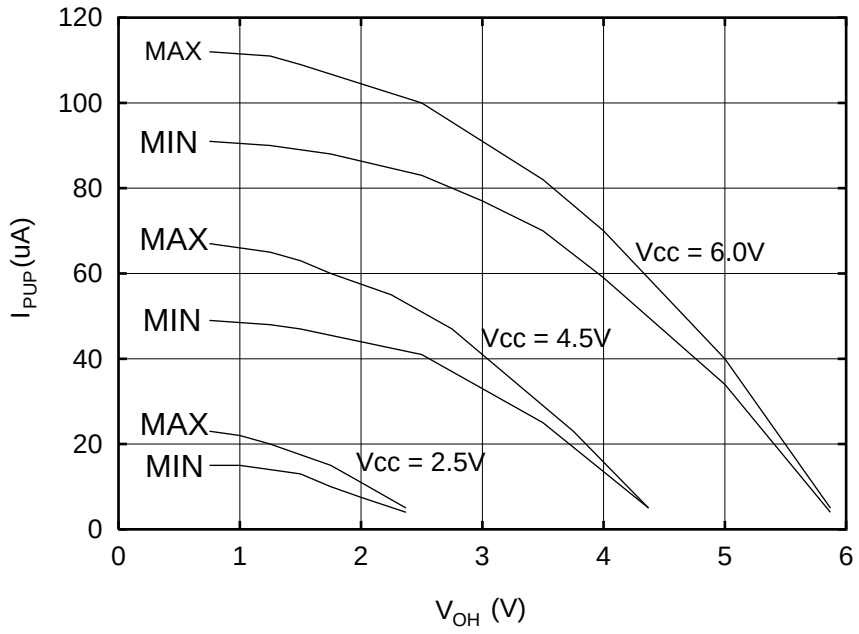
Port L/C/G Push-Pull Source Current



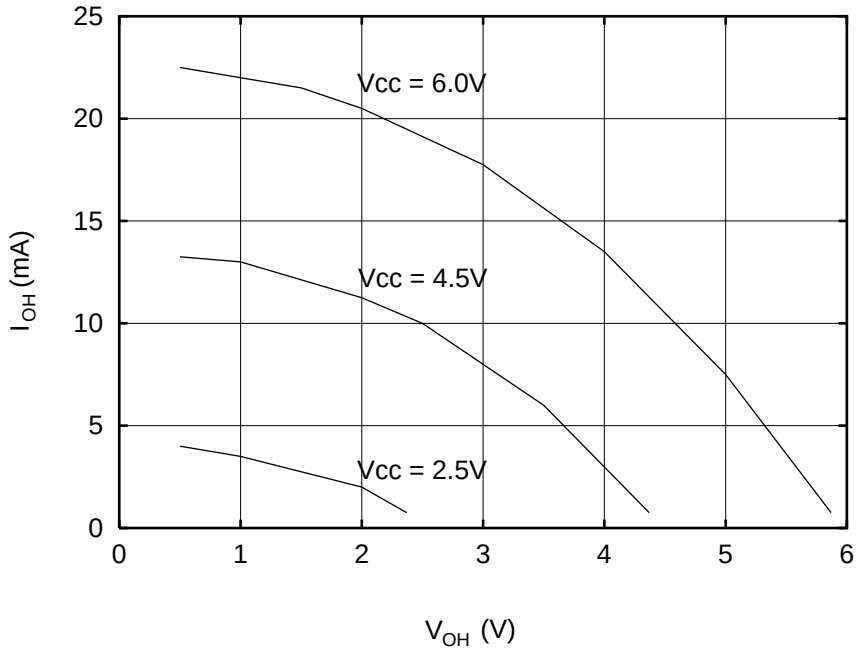
Port L/C/G Push-Pull Sink Current



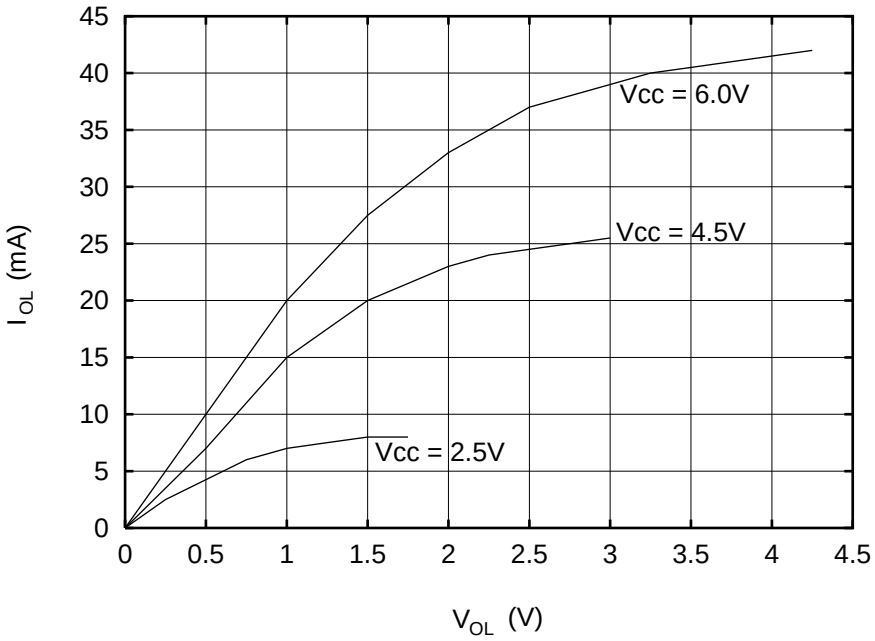
Port L/C/G Pull-up Source Current



Port D Source Current



Port D Sink Current



A

A/D converter registers
  COP888CF 10-3, 15-16
  COP888EB 17-23
AC power control A-14
Acceptance filtering 11-14
Accumulator 2-8
Accumulator Bit Manipulation instructions 12-8
ACK field 11-10
Acknowledgement error 11-15
Add (ADD) 12-12
Add with Carry (ADC) 12-11
Addressing modes 12-1, 12-2
Alternate functions 7-4
  Port G 7-4
ALU 2-12
ALU interface 2-13
Analog switches 11-25
Analog-to-digital conversion A-17
  using on-chip comparator A-17
Analog-to-digital converter
  accuracy 10-7, 15-20
  channel selection 10-5, 15-18
  continuous mode 10-4, 15-18
  COP888CF 15-14
  hardware considerations 10-7, 15-20
  multi-channel conversion 10-5, 15-19
  operation 10-1, 15-14
  prescaler selection 10-3, 15-17
  single conversion 10-4, 15-18
  speed 10-7, 15-20
And (AND) 12-12
And, Skip if Zero (ANDSZ) 12-13
Arbitration field 11-8
Architecture 2-1
Arithmetic instructions 12-6
Arithmetic Logic Unit (ALU) 2-12

B

Battery-powered weight measurement A-20
Binary division A-31
Binary multiplication A-29
Binary/BCD arithmetic operations A-24
Bit CRC error 11-15
Bit error 11-15
Bit Stream Processor 11-22
Bit types
  acknowledgement error 11-15
Block diagram 8-2
  COP888 2-2
  COP888 interrupt 3-2
  COP888BC 13-1
  COP888CF 15-1
  COP888CL 16-1
  COP888CS/CG/EG/GG/HG/KG 14-2
  COP888EB 17-1
  COP888EK 18-1
  COP888FH 19-1
  COP888GW 20-1
  idle timer 4-11
  MICROWIRE/PLUS circuit 5-14
Board layout A-41
Bus connections 11-24
Bus timing considerations 11-26
Bus-off state 11-19
BUSY 2-9
Busy flag and interrupt 5-17

C

C (see Carry Flag)
CAN interface 11-1
  acceptance filtering 11-14
  ACK field 11-10
  arbitration field 11-8
  Bit Stream Processor 11-22
  block 11-22
  Clock Generator 11-25
  contention-based arbitration 11-3
  control field 11-9
  COP888EB 17-23
  CRC field 11-9
  CRC Generator 11-24
  CRC Register 11-24
  data field 11-9
  data frames 11-8
  End of Frame (EOF) 11-10
  error counters 11-16
  error detection 11-14
  error frames 11-11
  error handling 11-4
  error management 11-14
  Error Management Logic 11-22
  error types 11-15
  extended frames 11-40
  frame arbitration 11-14
  frame coding 11-7
  frame fields 11-5
  frame priority 11-14
  frame validation 11-14
  frames 11-2, 11-4
  hard synchronization 11-20
  input comparators 11-24
  Interface Management Logic 11-22
  interframe spaces 11-10
  module error states 11-17
  modules 11-2
  output drivers 11-24
  overload frames 11-13
  physical layer 11-2
  register block 11-28
  remote frames 11-8
  reset state 11-40
  Rx/Tx Register 11-24
  soft synchronization 11-20
  Start of Frame (SOF) 11-8
  synchronization 11-19
  Transceiver Logic 11-22
Capacitive decoupling A-42

Carrier Sense Multiple Access - Collision Avoidance (CSMA-CA) 11-1
Carry Flag 2-9
Characterization data B-1
Clear Accumulator (CLR) 12-13
Clear RAM A-24
Clock Generator 11-25
Clock Monitor 8-1
  configuration 8-3
  operation 8-3
Clock options 2-22
Clock-stopping method 6-1
CMPSL 14-9, 18-9, 19-8
CNTRL register 2-9
Comparator
  COP888CS/CG/EG/GG/HG/KG 14-17
  COP888FH 19-17
  select register 14-9, 18-9, 19-8
Conditional instructions 12-8
Contention-based arbitration
  CAN interface 11-3
Context switching 3-4
Control field 11-9
Control logic 2-12, 2-13
Control register format
  COP888EB 17-26
Control registers 2-9
Controller Area Network (CAN) interface 11-1
  acceptance filtering 11-14
  ACK field 11-10
  arbitration field 11-8
  Bit Stream Processor 11-22
  block 11-22
  Clock Generator 11-25
  contention-based arbitration 11-3
  control field 11-9
  CRC field 11-9
  CRC Generator 11-24
  CRC Register 11-24
  data field 11-9
  data frames 11-8
  End of Frame (EOF) 11-10
  error counters 11-16
  error detection 11-14
  error frames 11-11
  error handling 11-4
  error management 11-14
  Error Management Logic 11-22
  error types 11-15
  extended frames 11-40
  frame arbitration 11-14
  frame coding 11-7
  frame fields 11-5
  frame priority 11-14
  frame validation 11-14
  frames 11-2, 11-4
  hard synchronization 11-20
  input comparators 11-24
  Interface Management Logic 11-22
  interframe spaces 11-10
  module error states 11-17
  modules 11-2
  output drivers 11-24
  overload frames 11-13
  physical layer 11-2
  register block 11-28
  remote frames 11-8
  Rx/Tx Register 11-24
  soft synchronization 11-20
  Start of Frame (SOF) 11-8
  synchronization 11-19
  Transceiver Logic 11-22
COP800 port structure 7-1, 18-11
COP888 block diagram 2-2
COP888BC 13-1
  block diagram 13-1
  data memory 13-5
  emulation devices 13-26
  input/output ports 13-4
  interrupts 13-14
  memory map 13-10
  pin descriptions 13-3
  pinouts/packages 13-2
  program memory 13-5
  register bit maps 13-5
  reset 13-12
COP888CF 10-1, 15-1
  A/D accuracy 10-7, 15-20
  A/D channel selection 10-5, 15-18
  A/D continuous mode 10-4, 15-18
  A/D converter registers 10-3, 15-16
  A/D multi-channel conversion 10-5, 15-19
  A/D operation 10-1, 15-14
  A/D prescaler selection 10-3, 15-17
  A/D single conversion 10-4, 15-18
  A/D speed 10-7, 15-20
  analog-to-digital converter 15-14
  block diagram 15-1
  data memory 15-7
  hardware considerations 10-7, 15-20
  input/output ports 15-5
  interrupts 15-13
  mask options 15-21
  memory map 15-9
  pin descriptions 15-2
  pinouts/packages 15-2
  program memory 15-7
  register bit maps 15-7
  reset 15-12
COP888CG/EG/CS 9-1
  input/output ports 18-5
  UART 9-1
COP888CL 16-1
  block diagram 16-1
  data memory 16-7
  input/output ports 16-5
  interrupts 16-12
  mask options 16-13
  memory map 16-9
  pin descriptions 16-2
  pinouts/packages 16-1
  program memory 16-7

register bit maps 16-7
reset 16-11
COP888CS/CG/EG/GG/HG/KG 14-1
block diagrams 14-2
comparator 14-17
data memory 14-8
input/output ports 14-6
interrupts 14-17
mask options 14-20
memory map 14-13
pin descriptions 14-3
pinouts/packages 14-2
program memory 14-7
register bit maps 14-8
reset 14-15
UART 14-19
COP888EB 17-1
A/D converter 17-23
block diagram 17-1
CAN interface 17-23
control register format 17-26
data memory 17-8
HALT/IDLE 17-23, 17-32
input/output ports 17-7
interrupts 17-21
loop back mode 17-32
mask options 17-33
memory map 17-16
pin descriptions 17-4
pinouts/packages 17-2
program memory 17-8
register bit maps 17-9
reset 17-20
SPI interface 17-24
SPI synchronization 17-30
SS-Expander 17-28
status register format 17-27
UART 17-21
COP888EK 18-1
block diagram 18-1
data memory 18-7
interrupts 18-17
mask options 18-18
memory map 18-13
pin descriptions 18-2
pinouts/packages 18-2
program memory 18-7
register bit maps 18-7
reset 18-16
COP888FH 19-1
block diagrams 19-1
comparator 19-17
data memory 19-7
input/output ports 19-5
interrupts 19-17
mask options 19-22
memory map 19-12
pin descriptions 19-2
pinouts/packages 19-2
program memory 19-7
register bit maps 19-7
reset 19-15
UART 19-19
COP888GW 20-1
block diagrams 20-1
data memory 20-7
input/output ports 20-5
interrupts 20-19
mask options 20-35
memory map 20-14
pin descriptions 20-2
pinouts/packages 20-2
program memory 20-7
register bit maps 20-7
reset 20-17
UART 20-19
Core registers 2-8
CPU operation 2-12
CRC field 11-9
CRC Generator 11-24
CRC Register 11-24
Crystal oscillator 2-22
CSMA-CA 11-1

D

Data field 11-9
Data frames 11-8
Data memory 2-3
COP888BC 13-5
COP888CF 15-7
COP888CL 16-7
COP888CS/CG/EG/GG/HG/KG 14-8
COP888EB 17-8
COP888EK 18-7
COP888FH 19-7
COP888GW 20-7
Data memory fetches 2-14
Data memory map 2-5
Data register 2-10
Data segment extension 2-6
Data segment extension register 2-11
Data store memory pointers 2-11
Decimal Correct (DCOR) 12-14
Decoupling A-42
capacitive A-42
inductive A-42
Decrement Accumulator (DEC) 12-14
Decrement Register and Skip if Zero (DRSZ) 12-15
Dedicated inputs 2-8
Dedicated outputs 2-8
Device-specific features 1-2
Display drivers 5-13
DRSZ instruction 2-18

E

EEPROM 2-3, 5-13
Electric Field emissions A-40
Electrical characterization data B-1
Electromagnetic interference (EMI) considerations A-39
Emulation devices
COP888BC 13-26
End of Frame (EOF) field 11-10
Error counters 11-16
Error detection 11-14
Error frames 11-11
Error handling 2-20
CAN interface 11-4
Error management 11-14

Error Management Logic 11-22
Error report on WDOUT 8-5
Error types 11-15
bit CRC error 11-15
bit error 11-15
form error 11-15
stuff error 11-15
Error-passive state 11-18
Exchange Memory with Accumulator (X) 12-39
Exclusive Or (XOR) 12-41
EXEN 2-9
EXPND 2-9
Extended frames 11-40
External event counter mode 4-5
example A-13
External Interrupt Enable 2-9
External Interrupt Pending 2-9

F

Features
basic 1-1
device-specific 1-2
list 1-3
Fetches
data memory 2-14
program memory 2-14
Fields
ACK 11-10
arbitration 11-8
control 11-9
CRC 11-9
data 11-9
End of Frame (EOF) 11-10
Start of Frame (SOF) 11-8
Five-cycle instructions 2-19
FLAG1 2-10
FLAG2 2-10
Form error 11-15
Four-cycle instructions 2-18
Frame coding 11-7
Frame fields 11-5
Frames
arbitration 11-14
CAN interface 11-2, 11-4
data 11-8
error 11-11
extended 11-40
overload 11-13
priority 11-14
remote 11-8
validation 11-14

G

General-purpose timers 4-10
GIE 2-9
Global Interrupt Enable 2-9

H

Half-Carry Flag 2-9
HALT exit
using G7 pin 6-3
using multi-input wakeup 6-2
using Reset 6-2
Halt mode
entering 6-1
exiting 6-2
HALT/IDLE
COP888EB 17-23, 17-32
HALT/IDLE operation 6-4
Harvard architecture 2-1
HC (see Half-Carry Flag)
Hi-Z input 2-8

I

ICNTRL register 2-10
IDLE mode 6-3
Idle timer 4-11
block diagram 4-11
IEDG 2-9
If B Pointer Not Equal (IFBNE) 12-16
Increment Accumulator (INC) 12-20
Index registers 2-11
Inductive decoupling A-42
Industrial timer A-22
Input capture mode 4-7
Input comparators 11-24
Input protection A-37
Input/output 7-1
Input/output ports
COP888BC 13-4
COP888CF 15-5
COP888CG/EG/CS 18-5
COP888CL 16-5
COP888CS/CG/EG/GG/HG/KG 14-6
COP888EB 17-7
COP888FH 19-5
COP888GW 20-5
Instructions
ADC 12-11
ADD 12-12
AND 12-12
ANDSZ 12-13
bytes and cycles 12-43
CLR 12-13
DCOR 12-14
DEC 12-14
decoding 2-15
DRSZ 2-18, 12-15
execution 2-15
five-cycle 2-19
four-cycle 2-18
IFBIT 12-15
IFBNE 12-16
IFC 12-17
IFEQ 12-17
IFGT 12-18
IFNC 12-19
IFNE 12-19
INC 12-20
INTR 12-21
JID 2-17, 12-21
JMP 2-17, 12-22
JMPL 12-23

JP 2-17, 12-23
JSR 12-24
JSRL 12-25
LAID 2-17, 12-26
LD 12-26, 12-28, 12-29
NOP 12-30
one-cycle 2-15
operations summary 12-42
OR 12-30
POP 2-18, 12-31
PUSH 2-18, 12-31
RBIT 12-31
RC 12-32
RET 12-32
RETI 12-33
RETSK 12-33
RPND 12-34
RRC 12-34, 12-35
SBIT 12-35
SC 12-36
seven-cycle 2-20
single cycle 2-15
SUBC 12-36
SWAP 12-38
three-cycle 2-16
two-cycle 2-15
types 12-6
VIS 12-38
X 12-39
XOR 12-41
Interface Management Logic 11-22
Interframe spaces 11-10
Interrupt Software Trap (INTR) 12-21
Interrupts 3-1
block diagram 3-2
COP888BC 13-14
COP888CF 15-13
COP888CL 16-12
COP888CS/CG/EG/GG/HG/KG 14-17
COP888EB 17-21
COP888EK 18-17
COP888FH 19-17
COP888GW 20-19
handling 2-20
maskable 3-5
non-maskable 3-7
priority 3-3
summary 3-10
vector table 3-4

J

JID instruction 2-17
JMP instruction 2-17
JP instruction 2-17
Jump Absolute (JMP) 12-22
Jump Absolute Long (JMPL) 12-23
Jump Indirect (JID) 12-21
Jump Relative (JP) 12-23
Jump Subroutine (JSR) 12-24
Jump Subroutine Long (JSRL) 12-25

L

LAID instruction 2-17
Load Accumulator (LD) 12-26
Load Accumulator Indirect (LAID) 12-26
Load and Exchange instructions 12-7
Load B Pointer (LD) 12-28
Load Memory (LD) 12-28
Load Register (LD) 12-29
Logical instructions 12-7
Loop back mode
COP888EB 17-32

M

MAR (Memory Address Register) 2-14
Mask options
COP888CF 15-21
COP888CL 16-13
COP888CS/CG/EG/GG/HG/KG 14-20
COP888EB 17-33
COP888EK 18-18
COP888FH 19-22
COP888GW 20-35
Maskable interrupts 3-5
Master mode operation 5-18
MDR (Memory Data Register) 2-12, 2-13, 2-14
Mechanical shielding A-43
Memory
data 2-3
mapped I/O registers 2-7
organization 2-2
program 2-3
Memory Bit Manipulation instructions 12-8
Memory fetches 2-14
Memory map
basic 2-4
COP888BC 13-10
COP888CF 15-9
COP888CL 16-9
COP888CS/CG/EG/GG/HG/KG 14-13
COP888EB 17-16
COP888EK 18-13
COP888FH 19-12
COP888GW 20-14
MICROWIRE busy shifting flag 2-9
MICROWIRE/PLUS 5-13
busy flag and interrupt 5-17
circuit block diagram 5-14
continuous mode A-3
fast burst output A-4
interface A-1
interface timing 5-15
master/slave protocol A-1
Ports G configuration 5-16
register 2-12
theory of operation 5-14
timing 5-14
Module error states 11-17
bus-off state 11-19
error-passive state 11-18
Modules
CAN interface 11-2
MSEL 2-9
Multi-input wakeup/interrupt 7-4

N

NMC93C06 Instruction Set A-6
NMC93C06-COP820C interface A-5

NMI 3-1, 3-9
exit from HALT/IDLE 6-4
NMIPND 3-9, 3-10
No Operation (NOP) 12-30
Non-maskable interrupt pending flags 3-7
Non-maskable interrupts 3-7

O

On-chip voltage reference 11-25
One-cycle instructions 2-15
Opcodes 12-45
Or (OR) 12-30
Oscillator control A-43
Output drivers 11-24
Output series resistance A-43
Overload frames 11-13

P

PEN 2-10
Physical layer
CAN interface 11-2
Pin descriptions
COP888BC 13-3
COP888CF 15-2
COP888CL 16-2
COP888CS/CG/EG/GG/HG/KG 14-3
COP888EB 17-4
COP888EK 18-2
COP888FH 19-2
COP888GW 20-2
Pinouts/packages
COP888BC 13-2
COP888CF 15-2
COP888CL 16-1
COP888CS/CG/EG/GG/HG/KG 14-2
COP888EB 17-2
COP888EK 18-2
COP888FH 19-2
COP888GW 20-2
Pointer Operation with SPI Disabled 17-32
POP instruction 2-18
Pop Stack (POP) 12-31
Ports
C 7-2
D 7-2
G 7-3
alternate functions 7-4
method 6-2
I 7-3
L 7-3
Power save modes 6-1
Power wakeup circuit A-33
PPND 2-10
Program counter 2-9
Program memory 2-3
COP888BC 13-5
COP888CF 15-7
COP888CL 16-7
COP888CS/CG/EG/GG/HG/KG 14-7
COP888EB 17-8
COP888EK 18-7
COP888FH 19-7
COP888GW 20-7
Program memory fetches 2-14
Programming examples A-24
PSW register 2-9
Pulse Width Modulation 4-4
A/D conversion A-17
PUSH instruction 2-18
Push Stack (PUSH) 12-31
Push-pull 2-8
PWM mode (see Pulse Width Modulation)

R

RC oscillator 2-23
Receive/Transmit (Rx/Tx) Register 11-24
Reconfigurable input/output 2-7
Register bit maps
COP888BC 13-5
COP888CF 15-7
COP888CL 16-7
COP888CS/CG/EG/GG/HG/KG 14-8
COP888EB 17-9
COP888EK 18-7
COP888FH 19-7
COP888GW 20-7
Register block 11-28
Registers
CMPSL 14-9, 18-9, 19-8
CNTRL 2-9
control 2-9
data 2-10
data segment extension 2-11
ICNTRL 2-10
MICROWIRE/PLUS 2-12
PSW 2-9
timer 2-12
WDSVR 8-1
WKEGD 7-4
WKEN 7-4
WKPND 7-4
Remote frames 11-8
Reset 2-21
COP888BC 13-12
COP888CF 15-12
COP888CL 16-11
COP888CS/CG/EG/GG/HG/KG 14-15
COP888EB 17-20
COP888EK 18-16
COP888FH 19-15
COP888GW 20-17
Reset Carry (RC) 12-32
Reset Memory Bit (RBIT) 12-31
Reset Pending (RPND) 12-34
Reset stateController Area Network (CAN) interface
reset state 11-40
Return and Skip (RETSK) 12-33
Return from Interrupt (RETI) 12-33
Return from Subroutine (RET) 12-32
Rotate Accumulator Left Through Carry (RLC) 12-34
Rotate Accumulator Right Through Carry (RRC) 12-35

S

S register 2-6
Schmitt triggers 7-3
Set Carry (SC) 12-36
Set Memory Bit (SBIT) 12-35
Seven-cycle instructions 2-20
Single slope A/D conversion A-17
SK clock frequency 5-16
SL0 2-9
SL1 2-9
Slave mode operation 5-19
Software trap 3-1, 3-8, 3-9
SPI interface
  COP888EB 17-24
SPI Receive Data Register (SPIRXD) 17-30
SPI Receive FIFO 17-30
SPI synchronization
  COP888EB 17-30
SPI Transmit Data Register (SPITXD) 17-30
SPI Transmit FIFO 17-31
SS-Expander
  COP888EB 17-28
Stack Control instructions 12-8
Stack pointer 2-11
Start of Frame (SOF) 11-8
Status register format
  COP888EB 17-27
STPND 3-8
Stuff error 11-15
Subtract with Carry (SUBC) 12-36
Swap Nibbles of Accumulator (SWAP) 12-38
Synchronization
  CAN interface 11-19
  hard 11-20
  soft 11-20

T

T1C0 2-9
T1C1 2-9
T1C2 2-10
T1C3 2-10
T1ENA 2-9
T1ENB 2-10
T1PNDA 2-9
T1PNDB 2-10
technical support C-1
Telecommunications 5-13
Test Bit (IFBIT) 12-15
Test if Carry (IFC) 12-17
Test if Equal (IFEQ) 12-17
Test if Greater Than (IFGT) 12-18
Test If No Carry (IFNC) 12-19
Test If Not Equal (IFNE) 12-19
Three-cycle instructions 2-16
Timer capture example A-11
Timer register 2-12
Timer T1A Interrupt Enable 2-9
Timer T1A Interrupt Pending 2-9
Timers 4-1
  applications A-11
  block diagram 4-1
  control bits 4-2
  general-purpose 4-10
  idle 4-11
  operating modes 4-3
  register addresses 4-10
Timing 5-14
Transceiver Logic 11-22
Transfer-of Control instructions 12-7
Transmission start in master mode 17-30
Triac control A-14
Two-cycle instructions 2-15

U

UART
  asynchronous mode 9-6
  attention mode 9-19
  baud clock generation 9-12
  baud select 9-5
  break detection 9-20
  break generation 9-20
  control 9-5
  COP888CG/EG/CS 9-1
  COP888CS/CG/EG/GG/HG/KG 14-19
  COP888EB 17-21
  COP888FH 19-19
  COP888GW 20-19
  data registers 9-5
  diagnostic testing 9-19
  error flags 9-18
  framing formats 9-8, 9-9
  HALT/IDLE mode reinitialization 9-11
  interface 9-6
  interrupts 9-18
  operation overview 9-1
  prescaler 9-5
  prescaler factors 9-15
  registers 9-2
  reset initialization 9-10
  status registers 9-5
  synchronous mode 9-8
Universal Asynchronous Receiver Transmitter (see UART)
uWEN 2-10
uWPND 2-10

V

Vector Interrupt Select (VIS) 12-38
Vector table 3-2
VIS instruction 3-1, 3-2

W

Watchdog 8-1
  block diagram 8-2
  circuit 8-1
  configuration 8-3
  logic 8-2
  operation 6-4, 8-1
  reset circuit A-34
WDOUT
  error report 8-5
WDSVR 8-1
Weak pull-up 2-8
WKEDG 7-4
WKEN 7-4
WKPND 7-4

Z

Zero cross detection A-20