TEXAS INSTRUMENTS

MSP430x3xx Family

User's Guide

July 2000

Mixed Signal Products

SLAU012

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Preface

Read This First

About This Manual

The MSP430x3xx User's Guide is intended to assist the development of MSP430x3xx family products by assembling together and presenting hardware and software information in a manner that is easy for engineers and programmers to use.

This manual discusses modules and peripherals of the MSP430x3xx family of devices. Each discussion presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals are present on all devices. In addition, modules or peripherals may differ in their exact implementation between device families, or may not be fully implemented on an individual device or device family. Therefore, a user must always consult the data sheet of any device of interest to determine what peripherals and modules are implemented, and exactly how they are implemented on that particular device.

How to Use This Manual

This document contains the following chapters and appendixes:

Chapter 1. Introduction

Chapter 2. Architectural Overview

Chapter 3. System Resets, Interrupts, and Operating Modes

Chapter 4. Memory

Chapter 5. 16-Bit CPU

Chapter 6. Hardware Multiplier

Chapter 7. FLL Clock Module

Chapter 8. Digital I/O Configuration

Chapter 9. Universal Timer/Port Module

Chapter 10. Timers

Chapter 11. Timer_A

Chapter 12. USART Peripheral Interface, UART Mode

Chapter 13. USART Peripheral Interface, SPI Mode

Chapter 14. Liquid Crystal Display Drive

Chapter 15. ADC12+2 A-to-D Converter

Appendix A. Peripheral File Map

Appendix B. Instruction Set Description

Appendix C. EPROM Programming

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's.

  Here is a sample program listing:

```
0011  0005  0001          .field    1, 2
0012  0005  0003          .field    3, 4
0013  0005  0006          .field    6, 3
0014  0006                .even
```

Related Documentation From Texas Instruments

For related documentation see the web site http://www.ti.com/sc/msp430.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Contents

1 Introduction 1-1
1.1 Features and Capabilities 1-2
1.2 31x Devices 1-3
1.3 32x Devices 1-3
1.4 33x Devices 1-4

2 Architectural Overview 2-1
2.1 Introduction 2-2
2.2 Central Processing Unit 2-2
2.3 Program Memory 2-3
2.4 Data Memory 2-3
2.5 Operation Control 2-3
2.6 Peripherals 2-4
2.7 Oscillator and Clock Generator 2-4

3 System Resets, Interrupts, and Operating Modes 3-1
3.1 System Reset and Initialization 3-2
3.1.1 Introduction 3-2
3.1.2 Device Initialization after System Reset 3-4
3.2 Global Interrupt Structure 3-5
3.3 MSP430 Interrupt-Priority Scheme 3-6
3.3.1 Operation of Global Interrupt—Reset/NMI 3-8
3.3.2 Operation of Global Interrupt—Oscillator Fault Control 3-8
3.4 Interrupt Processing 3-9
3.4.1 Interrupt Control Bits in Special-Function Registers (SFRs) 3-11
3.4.2 Interrupt Vector Addresses 3-14
3.5 Operating Modes 3-14
3.5.1 Low-Power Modes 0 and 1 (LPM0 and LPM1) 3-18
3.5.2 Low-Power Modes 2 and 3 (LPM2 and LPM3) 3-19
3.5.3 Low-Power Mode 4 (LPM4) 3-19
3.6 Basic Hints for Low-Power Applications 3-20

4 Memory 4-1
4.1 Introduction 4-2
4.2 Data in the Memory 4-3
4.3 Internal ROM Organization 4-4
4.3.1 Processing of ROM Tables 4-4
4.3.2 Computed Branches and Calls 4-5
4.4 RAM and Peripheral Organization 4-6

4.4.1 Random Access Memory 4-6
4.4.2 Peripheral Modules—Address Allocation 4-8
4.4.3 Peripheral Modules—Special Function Registers (SFRs) 4-10

5 16-Bit CPU 5-1
5.1 CPU Registers 5-2
5.1.1 The Program Counter (PC) 5-2
5.1.2 The System Stack Pointer (SP) 5-2
5.1.3 The Status Register (SR) 5-4
5.1.4 The Constant Generator Registers CG1 and CG2 5-5
5.2 Addressing Modes 5-7
5.2.1 Register Mode 5-8
5.2.2 Indexed Mode 5-9
5.2.3 Symbolic Mode 5-10
5.2.4 Absolute Mode 5-11
5.2.5 Indirect Mode 5-12
5.2.6 Indirect Autoincrement Mode 5-13
5.2.7 Immediate Mode 5-14
5.2.8 Clock Cycles, Length of Instruction 5-15
5.3 Instruction Set Overview 5-17
5.3.1 Double-Operand Instructions 5-18
5.3.2 Single-Operand Instructions 5-19
5.3.3 Conditional Jumps 5-20
5.3.4 Short Form of Emulated Instructions 5-21
5.3.5 Miscellaneous 5-22
5.4 Instruction Map 5-23

6 Hardware Multiplier 6-1
6.1 Hardware Multiplier Module Support 6-2
6.2 Hardware Multiplier Operation 6-3
6.2.1 Multiply Unsigned, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit 6-5
6.2.2 Multiply Signed, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit 6-6
6.2.3 Multiply Unsigned and Accumulate, 16×16bit, 16×8bit, 8×16bit, 8×8bit 6-7
6.2.4 Multiply Signed and Accumulate, 16×16bit, 16×8bit, 8×16bit, 8×8bit 6-8
6.3 Hardware Multiplier Registers 6-9
6.4 Hardware Multiplier Special Function Bits 6-10
6.5 Hardware Multiplier Software Restrictions 6-10
6.5.1 Hardware Multiplier Software Restrictions—Address Mode 6-10
6.5.2 Hardware Multiplier Software Restrictions—Interrupt Routines 6-11
6.5.3 Hardware Multiplier Software Restrictions—MACS 6-12

7 FLL Clock Module 7-1
7.1 The FLL Clock Module 7-2
7.2 Crystal Oscillator 7-3
7.3 Digitally-Controlled Oscillator (DCO) and Frequency-Locked Loop 7-4
7.3.1 FLL Operation 7-4
7.3.2 Modulator Operation 7-5
7.3.3 DCO Frequency Range 7-5
7.3.4 Disabling the FLL 7-6
7.3.5 MCLK Stability 7-6
7.3.6 Oscillator Fault Detection 7-6

7.4 FLL Operating Modes 7-7
7.4.1 Starting From Power Up Clear (PUC) 7-7
7.4.2 Adjusting the FLL Frequency 7-7
7.4.3 FLL Features for Low-Power Applications 7-7
7.5 Buffered Clock Output 7-8
7.6 FLL Module Control Registers 7-9
7.6.1 MCLK Frequency Control 7-9
7.6.2 Special-Function Register Bits 7-10

8 Digital I/O Configuration 8-1
8.1 Introduction 8-2
8.2 Port P0 8-3
8.2.1 Port P0 Control Registers 8-3
8.2.2 Port P0 Schematic 8-6
8.2.3 Port P0 Interrupt Control Functions 8-9
8.3 Ports P1, P2 8-11
8.3.1 Port P1, Port P2 Control Registers 8-12
8.3.2 Port P1, Port P2 Schematic 8-15
8.3.3 Port P1, P2 Interrupt Control Functions 8-16
8.4 Ports P3, P4 8-17
8.4.1 Port P3, P4 Control Registers 8-17
8.4.2 Port P3, P4 Schematic 8-19

9 Universal Timer/Port Module 9-1
9.1 Timer/Port Configuration 9-2
9.2 Timer/Port Module Operation 9-3
9.2.1 Timer/Port Counter TPCNT1, 8-Bit Operation 9-3
9.2.2 Timer/Port Counter TPCNT2, 8-Bit Operation 9-4
9.2.3 Timer/Port Counter, 16-Bit Operation 9-4
9.2.4 Enable Control 9-6
9.2.5 Comparator Input 9-6
9.3 Timer/Port Registers 9-7
9.4 Timer/Port Interrupts 9-11
9.5 Timer/Port in an ADC Application 9-12

10 Timers 10-1
10.1 Basic Timer1 10-2
10.1.1 Basic Timer1 Registers 10-3
10.1.2 Special Function Register Bits 10-5
10.1.3 Basic Timer1 Operation 10-5
10.1.4 Basic Timer1 Operation: Signal fLCD 10-6
10.2 8-Bit Interval Timer/Counter 10-7
10.2.1 Operation of 8-Bit Timer/Counter 10-8
10.2.2 8-Bit Timer/Counter Registers 10-9
10.2.3 Special Function Register Bits, 8-Bit Timer/Counter Related 10-11
10.2.4 Implementing a UART With the 8-Bit Timer/Counter 10-11
10.3 The Watchdog Timer 10-13
10.3.1 Watchdog Timer Register 10-14
10.3.2 Watchdog Timer Interrupt Control Functions 10-16
10.3.3 Watchdog Timer Operation 10-16

11 Timer_A 11-1
11.1 Introduction 11-2

11.2 Timer_A Operation 11-4
11.2.1 Timer Mode Control 11-4
11.2.2 Clock Source Select and Divider 11-5
11.2.3 Starting the Timer 11-6
11.3 Timer Modes 11-6
11.3.1 Timer – Stop Mode 11-6
11.3.2 Timer – Up Mode 11-6
11.3.3 Timer – Continuous Mode 11-9
11.3.4 Timer – Up/Down Mode 11-10
11.4 Capture/Compare Blocks 11-13
11.4.1 Capture/Compare Block – Capture Mode 11-14
11.4.2 Capture/Compare Block – Compare Mode 11-18
11.5 The Output Unit 11-19
11.5.1 Output Unit – Output Modes 11-20
11.5.2 Output Control Block 11-21
11.5.3 Output Examples 11-22
11.6 Timer_A Registers 11-25
11.6.1 Timer_A Control Register TACTL 11-25
11.6.2 Timer_A Register TAR 11-27
11.6.3 Capture/Compare Control Register CCTLx 11-27
11.6.4 Timer_A Interrupt Vector Register 11-30
11.7 Timer_A UART 11-34

12 USART Peripheral Interface, UART Mode 12-1
12.1 USART Peripheral Interface 12-2
12.2 USART Peripheral Interface, UART Mode 12-3
12.2.1 UART Serial Asynchronous Communication Features 12-3
12.3 Asynchronous Operation 12-4
12.3.1 Asynchronous Frame Format 12-4
12.3.2 Baud Rate Generation in Asynchronous Communication Format 12-5
12.3.3 Asynchronous Communication Formats 12-7
12.3.4 Idle-Line Multiprocessor Format 12-7
12.3.5 Address-Bit Multiprocessor Format 12-9
12.4 Interrupt and Enable Functions 12-11
12.4.1 USART Receive Enable Bit 12-11
12.4.2 USART Transmit Enable Bit 12-12
12.4.3 USART Receive Interrupt Operation 12-13
12.4.4 USART Transmit Interrupt Operation 12-14
12.5 Control and Status Registers 12-15
12.5.1 USART Control Register UCTL 12-15
12.5.2 Transmit Control Register UTCTL 12-17
12.5.3 Receiver Control Register URCTL 12-18
12.5.4 Baud Rate Select and Modulation Control Registers 12-20
12.5.5 Receive-Data Buffer URXBUF 12-21
12.5.6 Transmit Data Buffer UTXBUF 12-22
12.6 Utilizing Features of Low-Power Modes 12-23
12.6.1 Receive-Start Operation From UART Frame 12-23
12.6.2 Maximum Utilization of Clock Frequency vs Baud Rate UART Mode 12-25
12.6.3 Support of Multiprocessor Modes for Reduced Use of MSP430 Resources 12-26
12.7 Baud Rate Considerations 12-26

12.7.1 Bit Timing in Transmit Operation 12-27
12.7.2 Typical Baud Rates and Errors 12-29
12.7.3 Synchronization Error 12-30

13 USART Peripheral Interface, SPI Mode 13-1
13.1 USART Peripheral Interface 13-2
13.2 USART Peripheral Interface, SPI Mode 13-3
13.2.1 SPI Mode Features 13-3
13.3 Synchronous Operation 13-4
13.3.1 Master SPI Mode 13-7
13.3.2 Slave SPI Mode 13-8
13.4 Interrupt and Control Functions 13-9
13.4.1 USART Receive/Transmit Enable Bit, Receive Operation 13-9
13.4.2 USART Receive/Transmit Enable Bit, Transmit Operation 13-11
13.4.3 USART Receive-Interrupt Operation 13-13
13.4.4 Transmit-Interrupt Operation 13-14
13.5 Control and Status Registers 13-15
13.5.1 USART Control Register 13-15
13.5.2 Transmit Control Register UTCTL 13-16
13.5.3 Receive Control Register URCTL 13-18
13.5.4 Baud Rate Select and Modulation Control Registers 13-18
13.5.5 Receive Data Buffer URXBUF 13-19
13.5.6 Transmit Data Buffer UTXBUF 13-19

14 Liquid Crystal Display Drive 14-1
14.1 LCD Drive Basics 14-2
14.2 LCD Controller/Driver 14-7
14.2.1 LCD Controller/Driver Features 14-8
14.2.2 LCD Timing Generation 14-8
14.2.3 LCD Voltage Generation 14-9
14.2.4 LCD Outputs 14-10
14.2.5 LCD Control Register 14-14
14.2.6 LCD Memory 14-16
14.3 Code Examples 14-21
14.3.1 Example Code for Static LCD 14-21
14.3.2 Example Code for Two MUX, 1/2-Bias LCD 14-22
14.3.3 Example Code for Three MUX, 1/3-Bias LCD 14-23
14.3.4 Example Code for Four MUX, 1/3-Bias LCD 14-24

15 ADC12+2 A-to-D Converter 15-1
15.1 Introduction 15-2
15.2 Analog-to-Digital Operation 15-4
15.2.1 A/D Conversion 15-4
15.2.2 A/D Interrupt 15-7
15.2.3 A/D Ranges 15-7
15.2.4 A/D Current Source 15-8
15.2.5 Analog Inputs and Multiplexer 15-9
15.2.6 A/D Grounding and Noise Considerations 15-10
15.2.7 A/D Converter Input and Output Pins 15-12
15.3 ADC12+2 Control Registers 15-13
15.3.1 Input Register AIN 15-13

15.3.2 Input Enable Register AEN 15-14
15.3.3 ADC12+2 Data Register ADAT 15-14
15.3.4 ADC12+2 Control Register ACTL 15-15

A Peripheral File Map A-1
A.1 Overview A-2
A.2 Special Function Register of MSP430x3xx Family, Byte Access A-2
A.3 Digital I/O, Byte Access A-3
A.4 LCD Registers, Byte Access A-5
A.5 8-Bit Timer/Counter, Basic Timer, Timer/Port, Byte Access A-6
A.6 FLL Registers, Byte Access A-6
A.7 EPROM Control Register and Crystal Buffer, Byte Access A-7
A.8 USART, UART Mode (Sync=0), Byte Access A-7
A.9 USART, SPI Mode (Sync=1), Byte Access A-8
A.10 ADC12+2, Word Access A-9
A.11 Watchdog/Timer, Word Access A-10
A.12 Hardware Multiplier, Word Access A-10
A.13 Timer_A Registers, Word Access A-11

B Instruction Set Description B-1
B.1 Instruction Set Overview B-2
B.1.1 Instruction Formats B-4
B.1.2 Conditional and Unconditional Jumps (Core Instructions) B-5
B.1.3 Emulated Instructions B-6
B.2 Instruction Set Description B-8

C EPROM Programming C-1
C.1 EPROM Operation C-2
C.1.1 Erasure C-2
C.1.2 Programming Methods C-2
C.1.3 EPROM Control Register EPCTL C-3
C.1.4 EPROM Protect C-4
C.2 FAST Programming Algorithm C-4
C.3 Programming an EPROM Module Through a Serial Data Link Using the JTAG Feature C-5
C.4 Programming an EPROM Module With Controller’s Software C-6
C.5 Code C-8

Figures

2–1 MSP430 System Configuration 2-2
2–2 Bus Connection of Modules/Peripherals 2-4
3–1 Power-on Reset and Power-Up Clear Schematic 3-2
3–2 Power-On Reset Timing on Fast V_{CC} Rise Time 3-3
3–3 Power-on Reset Timing on Slow V_{CC} Rise Time 3-3
3–4 Interrupt Priority Scheme 3-6
3–5 Block Diagram of NMI Interrupt Sources 3-7
3–6 $\overline{\text{RST}}$/NMI Mode Selection 3-7
3–7 Interrupt Processing 3-9
3–8 Return from Interrupt 3-10
3–9 Status Register (SR) 3-10
3–10 MSP430x3xx Family Operating Modes 3-17
3–11 Typical Current Consumption vs Operating Modes 3-18
4–1 Memory Map of Basic Address Space 4-2
4–2 Memory Data Bus 4-2
4–3 Bits, Bytes, and Words in a Byte-Organized Memory 4-3
4–4 ROM Organization 4-4
4–5 Byte and Word Operation 4-6
4–6 Register-Byte/Byte-Register Operations 4-7
4–7 Example of RAM/Peripheral Organization 4-8
5–1 Program Counter 5-2
5–2 System Stack Pointer 5-2
5–3 Stack Usage 5-3
5–4 PUSH SP and POP SP 5-3
5–5 Status Register Bits 5-4
5–6 Operand Fetch Operation 5-13
5–7 Double Operand Instruction Format 5-18
5–8 Single Operand Instruction Format 5-19
5–9 Conditional-Jump Instruction Format 5-20
5–10 Core Instruction Map 5-23
6–1 Connection of the Hardware Multiplier Module to the Bus System 6-2
6–2 Block Diagram of the MSP430 16×16-Bit Hardware Multiplier 6-3
6–3 Registers of the Hardware Multiplier 6-9
7–1 Frequency-Locked Loop 7-2
7–2 Crystal Oscillator Schematic 7-3
7–3 Fractional Tap Frequency Required 7-4
7–4 Modulator Hop Patterns 7-5
7–5 Schematic of Clock Buffer 7-8
7–6 SCFQCTL Register 7-9

7–7 SCFI0 and SCFI1 Registers 7-9
7–8 Crystal Buffer Control Register 7-10
8–1 Port P0 Configuration 8-3
8–2 Interrupt Flags Register 8-5
8–3 Interrupt Enable Register 8-6
8–4 Schematic of Bits P07 to P03 8-7
8–5 Schematic of Bit P02 8-7
8–6 Schematic of Bit P01 8-8
8–7 Schematic of Bit P00 8-8
8–8 Port P1, Port P2 Configuration 8-11
8–9 Schematic of One Bit in Port P1, P2 8-15
8–10 Ports P3, P4 Configuration 8-17
8–11 Schematic of Bits Pnx 8-19
9–1 Timer/Port Configuration 9-2
9–2 Timer/Port Counter, 16-Bit Operation 9-5
9–3 Timer/Port Comparator Input 9-6
9–4 Timer/Port Control Register 9-7
9–5 Timer/Port Counter Registers 9-8
9–6 Timer/Port Data Register 9-9
9–7 Timer/Port Enable Register 9-9
9–8 Timer/Port Interrupt Scheme 9-11
10–1 Basic Timer1 Configuration 10-2
10–2 Basic Timer1 Control Register 10-3
10–3 Basic Timer1 Control Register Function 10-4
10–4 Basic Timer1 Counter BTCNT1 10-4
10–5 Basic Timer1 Counter BTCNT2 10-5
10–6 8-Bit Timer/Counter 10-7
10–7 8-Bit Counter Example 10-8
10–8 8-Bit Timer/Counter Control Register 10-9
10–9 Start Bit Detection 10-12
10–10 Data Latching 10-12
10–11 Schematic of Watchdog Timer 10-13
10–12 Watchdog Timer Control Register 10-14
10–13 Reading WDTCTL 10-15
10–14 Writing to WDTCTL 10-15
11–1 Timer_A Block Diagram 11-3
11–2 Mode Control 11-4
11–3 Schematic of 16-Bit Timer 11-5
11–4 Schematic of Clock Source Select and Input Divider 11-5
11–5 Timer Up Mode 11-7
11–6 Up Mode Flag Setting 11-7
11–7 New Period > Old Period 11-8
11–8 New Period < Old Period 11-8
11–9 Timer Continuous Mode 11-9
11–10 Continuous Mode Flag Setting 11-9
11–11 Output Unit in Continuous Mode for Time Intervals 11-10
11–12 Timer Up/Down Mode 11-10
11–13 Output Unit in Up/Down Mode (II) 11-11
11–14 Timer Up/Down Direction Control 11-11
11–15 Up/Down Mode Flag Setting 11-12

11–16 Altering CCR0 – Timer in Up/Down Mode 11-12
11–17 Capture/Compare Blocks 11-13
11–18 Capture Logic Input Signal 11-14
11–19 Capture Signal 11-15
11–20 Capture Cycle 11-16
11–21 Software Capture Example 11-17
11–22 Output Unit 11-19
11–23 Output Control Block 11-21
11–24 Output Examples – Timer in Up Mode 11-23
11–25 Output Examples – Timer in Continuous Mode 11-23
11–26 Output Examples – Timer in Up/Down Mode (I) 11-24
11–27 Timer_A Control Register TACTL 11-25
11–28 TAR Register 11-27
11–29 Capture/Compare Control Register CCTLx 11-27
11–30 Capture/Compare Interrupt Flag 11-30
11–31 Schematic of Capture/Compare Interrupt Vector Word 11-31
11–32 Vector Word Register 11-31
11–33 UART Implementation 11-35
11–34 Timer_A UART Timing 11-36
12–1 Block Diagram of USART 12-2
12–2 Block Diagram of USART – UART Mode 12-3
12–3 Asynchronous Frame Format 12-4
12–4 Asynchronous Bit Format Example for n or n + 1 Clock Periods 12-4
12–5 Typical Baud-Rate Generation Other Than MSP430 12-5
12–6 MSP430 Baud Rate Generation Example for n or n + 1 Clock Periods 12-6
12–7 Idle-Line Multiprocessor Format 12-7
12–8 USART Receiver Idle Detect 12-8
12–9 Double-Buffered WUT and TX Shift Register 12-8
12–10 USART Transmitter Idle Generation 12-9
12–11 Address-Bit Multiprocessor Format 12-10
12–12 State Diagram of Receiver Enable 12-11
12–13 State Diagram of Transmitter Enable 12-12
12–14 Receive Interrupt Operation 12-13
12–15 Transmit Interrupt Operation 12-14
12–16 USART Control Register UCTL 12-15
12–17 Transmitter Control Register UTCTL 12-17
12–18 Receiver-Control Register URCTL 12-18
12–19 USART Baud Rate Select Register 12-20
12–20 USART Modulation Control Register 12-21
12–21 USART Receive Data Buffer URXBUF 12-21
12–22 Transmit Data Buffer UTXBUF 12-22
12–23 Receive-Start Conditions 12-23
12–24 Receive-Start Timing Using URXS Flag, Start Bit Accepted 12-24
12–25 Receive Start Timing Using URXS Flag, Start Bit Not Accepted 12-24
12–26 Receive Start Timing Using URXS Flag, Glitch Suppression 12-24
12–27 MSP430 Transmit Bit Timing 12-27
12–28 MSP430 Transmit Bit Timing Errors 12-27
12–29 Synchronization Error 12-30
13–1 Block Diagram of USART 13-2
13–2 Block Diagram of USART—SPI Mode 13-3

13–3 MSP430 USART as Master, External Device With SPI as Slave 13-5
13–4 Serial Synchronous Data Transfer 13-6
13–5 Data Transfer Cycle 13-6
13–6 MSP430 USART as Slave in Three-Pin or Four-Pin Configuration 13-7
13–7 State Diagram of Receiver Enable Operation—MSP430 as Master 13-10
13–8 State Diagram of Receive/Transmit Enable—MSP430 as Slave, Three-Pin Mode 13-10
13–9 State Diagram of Receive Enable—MSP430 as Slave, Four-Pin Mode 13-11
13–10 State Diagram of Transmit Enable—MSP430 as Master 13-11
13–11 State Diagram of Transmit Enable—MSP430 as Slave 13-12
13–12 Receive Interrupt Operation 13-13
13–13 Receive Interrupt State Diagram 13-13
13–14 Transmit-Interrupt Operation 13-14
13–15 USART Control Register 13-15
13–16 Transmit Control Register UTCTL 13-16
13–17 USART Clock Phase and Polarity 13-17
13–18 Receive Control Register URCTL 13-18
13–19 USART Baud-Rate Select Register 13-18
13–20 USART Modulation Control Register 13-19
13–21 Receive Data Buffer URXBUF 13-19
13–22 Transmit Data Buffer UTXBUF 13-19
14–1 Static Wave-Form Drive 14-3
14–2 Two-MUX Wave-Form Drive 14-4
14–3 Three-MUX Wave-Form Drive 14-5
14–4 Four-MUX Wave-Form Drive 14-6
14–5 LCD Controller/Driver Block Diagram 14-7
14–6 External LCD Module Analog Voltage 14-9
14–7 Schematic of LCD Output 14-10
14–8 Segment Line or Output Line 14-11
14–9 Mixed LCD and Port Mode Application 14-12
14–10 Schematic of LCD Pin – Timer/Port Comparator 14-13
14–11 LCD Control and Mode Register 14-14
14–12 Information Control 14-15
14–13 Display Memory Bits Attached to Segment Lines 14-16
14–14 Example With the Static Drive Mode 14-17
14–15 Example With the Two-MUX Mode 14-18
14–16 Example With the 3-MUX Mode 14-19
14–17 Example With the Four-MUX Mode 14-20
15–1 ADC12+2 Module Configuration 15-2
15–2 ADC12+2 Schematic 15-5
15–3 ADC12+2 Timing, 12-Bit Conversion 15-6
15–4 ADC12+2 Timing, 12+2-Bit Conversion 15-6
15–5 ADC, Input Sampling Timing 15-7
15–6 A/D Current Source 15-9
15–7 Analog Multiplexer 15-10
15–8 A/D Grounding and Noise Considerations 15-11
15–9 ADC12+2 Input Register, Input Enable Register 15-12
15–10 Input Register AIN 15-13
15–11 Input Enable Register AEN 15-14
15–12 ADC12+2 Data Register ADAT 15-14
15–13 ADC12+2 Control Register ACTL 15-15

B–1 Double-Operand Instructions B-4
B–2 Single-Operand Instructions B-5
B–3 Conditional and Unconditional Jump Instructions B-5
B–4 Decrement Overlap B-26
B–5 Main Program Interrupt B-46
B–6 Destination Operand—Arithmetic Shift Left B-47
B–7 Destination Operand—Carry Left Shift B-48
B–8 Destination Operand—Arithmetic Right Shift B-49
B–9 Destination Operand—Carry Right Shift B-51
B–10 Destination Operand Byte Swap B-58
B–11 Destination Operand Sign Extension B-59
C–1 EPROM Control Register EPCTL C-3
C–2 EPROM Programming With Serial Data Link C-5
C–3 EPROM Programming With Controller’s Software C-6

Tables

3–1 Interrupt Control Bits in SFRs 3-11
3–2 Interrupt Enable Registers 1 and 2 3-12
3–3 Interrupt Flag Register 1 and 2 3-13
3–4 Module Enable Registers 1 and 2 3-13
3–5 Interrupt Sources, Flags, and Vectors of 3xx Configurations 3-14
3–6 Low-Power Mode Logic Chart 3-17
4–1 Peripheral File Address Map—Word Modules 4-9
4–2 Peripheral File Address Map—Byte Modules 4-10
4–3 Special Function Register Address Map 4-11
5–1 Register by Functions 5-2
5–2 Description of Status Register Bits 5-4
5–3 Values of Constant Generators CG1, CG2 5-5
5–4 Source/Destination Operand Addressing Modes 5-7
5–5 Register Mode Description 5-8
5–6 Indexed Mode Description 5-9
5–7 Symbolic Mode Description 5-10
5–8 Absolute Mode Description 5-11
5–9 Indirect Mode Description 5-12
5–10 Indirect Autoincrement Mode Description 5-13
5–11 Immediate Mode Description 5-14
5–12 Instruction Format I and Addressing Modes 5-15
5–13 Instruction Format-II and Addressing Modes 5-16
5–14 Miscellaneous Instructions or Operations 5-16
5–15 Double Operand Instruction Format Results 5-18
5–16 Single Operand Instruction Format Results 5-19
5–17 ConditIonal-Jump Instructions 5-20
5–18 Emulated Instructions 5-21
6–1 Sum Extension Register Contents 6-4
6–2 Hardware Multiplier Registers 6-9
7–1 The DCO Range Control Bits 7-5
8–1 Port P0 Control Registers 8-4
8–2 Port P1 Registers 8-12
8–3 Port P2 Registers 8-12
8–4 Port P3 P4 Registers 8-18
9–1 Timer/Port Counter Signals, 16-Bit Operation 9-6
9–2 Timer/Port Registers 9-7
9–3 Bit EN1 Level/Signal 9-8
9–4 Timer/Port Clock Source Selection 9-8
9–5 Counter TPCNT2 Clock Sources 9-10

10–1 Basic Timer1 Registers 10-3
10–2 BTCNT2 Input Frequency Sources 10-4
10–3 8-Bit Timer/Counter Registers 10-9
10–4 Clock Input Source 10-10
10–5 WDTCNT Taps 10-14
11–1 Timer Modes 11-4
11–2 State of OUTx at Next Rising Edge of Timer Clock 11-22
11–3 Timer_A Registers 11-25
11–4 Mode Control 11-26
11–5 Input Clock Divider Control Bits 11-26
11–6 Clock Source Selection 11-26
11–7 Capture/Compare Control Register Output Mode 11-29
11–8 Capture/Compare Control Register Capture Mode 11-29
11–9 Vector Register TAIV Description 11-32
12–1 USART Interrupt Control and Enable Bits – UART Mode 12-11
12–2 Control and Status Registers 12-15
12–3 Interrupt Flag Set Conditions 12-19
12–4 Receive Data Buffer Characters 12-22
12–5 Commonly Used Baud Rates, Baud Rate Data, and Errors 12-29
13–1 USART Interrupt Control and Enable Bits – SPI Mode 13-9
13–2 USART Control and Status Registers 13-15
14–1 LCDM Selections 14-15
14–2 LCDM Signal Outputs for Port Functions 14-15
15–1 ADC12+2 Control Registers 15-13
15–2 A/D Input Selection 15-15
15–3 A/D Current Source Selection 15-16
15–4 Range Selection 15-16
15–5 ADCLK Clock Frequency 15-16

Examples

12–1 4800 Baud 12-6
12–2 19,200 Baud 12-6
12–3 Error Example for 2400 Baud 12-28
12–4 Synchronization Error—2400 Baud 12-31
C–1 MSP430 On-Chip Program Memory Format C-3
C–2 Fast Programming Subroutine C-4
C–3 Programming EPROM Module With Controller’s Software C-7
C–4 Subroutine C-7

Notes, Cautions, and Warnings

Word-Byte Operations 4-7
Status Register Bits V, N, Z and C 5-5
Data in Registers 5-8
Instruction Format II Immediate Mode 5-16
Destination Address 5-17
Instructions CMP and SUB 5-18
Writing to the Read-Only Register P0IN 8-4
Port P0 Interrupt Sensitivity 8-6
Writing to Read-Only Registers P1IN, P2IN 8-12
Port P1, Port P2 Interrupt Sensitivity 8-14
Function Select With P1SEL, P2SEL 8-15
Writing to Read-Only Register 8-18
Function Select With PnSEL Registers 8-19
RC1FG and RC2FG When Software Disables the Counter 9-7
Watchdog Timer, Changing the Time Interval 10-17
Capture With Timer Halted 11-16
Changing Timer_A Control Bits 11-27
Modifying Timer A Register TAR 11-27
Simultaneous Capture and Capture Mode Selection 11-30
Writing to Read-Only Register TAIV 11-32
URXE Reenabled, UART Mode 12-11
Writing to UTXBUF, UART Mode 12-12
Write to UTXBUF/Reset of Transmitter, UART Mode 12-12
Mark and Space Definitions 12-17
Receive Status Control Bits 12-20
Break Detect (BRK) Bit With Halted UART Clock 12-25
USART Synchronous Master Mode, Receive Initiation 13-7
USPIIE Re-Enabled, SPI Mode 13-10
Writing to UTXBUF, SPI Mode 13-12
Write to UTXBUF/Reset of Transmitter, SPI Mode 13-12
ADC, Start-of-Conversion 15-3
ADC12+2 Offset Voltage 15-8
Asterisked Instructions B-3
Operations Using the Status Register (SR) for Destination B-4
Conditional and Unconditional Jumps B-6
Disable Interrupt B-28
Enable Interrupt B-29
Emulating No-Operation Instruction B-42
The System Stack Pointer B-43

The System Stack Pointer B-44
RLA Substitution B-47
RLC and RLC.B Emulation B-48
Borrow Is Treated as a .NOT. B-52
Borrow Is Treated as a .NOT. B-56
Borrow Is Treated as a .NOT. Carry B-57
EPROM Exposed to Ambient Light (1) C-2

Chapter 1

Introduction

This chapter outlines the features and capabilities of the Texas Instruments (TI) MSP430x3xx family of microcontrollers.

The MSP430 employs a von-Neumann architecture, therefore, all memory and peripherals are in one address space.

The MSP430 devices constitute a family of ultralow-power, 16-bit RISC microcontrollers with an advanced architecture and rich peripheral set. The architecture uses advanced timing and design features, as well as a highly orthogonal structure to deliver a processor that is both powerful and flexible. The MSP430 consumes less than 400 μA in active mode operating at 1 MHz in a typical 3-V system and can wake up from a <2-μA standby mode to fully synchronized operation in less than 6 μs. These exceptionally low current requirements, combined with the fast wake-up time, enable a user to build a system with minimum current consumption and maximum battery life.

Additionally, the MSP430 family has an abundant mix of peripherals and memory sizes enabling true system-on-a-chip designs. The peripherals include a 14-bit A/D, slope A/D, multiple timers (some with capture/compare registers and PWM output capability), LCD driver, on-chip clock generation, H/W multiplier, USART, Watchdog Timer, GPIO, and others.

See http://www.ti.com for the latest device information and literature for the MSP430 family.

Topic **Page**

1.1 Features and Capabilities 1-2
1.2 31x Devices 1-3
1.3 32x Devices 1-3
1.4 33x Devices 1-4

1.1 Features and Capabilities

The TI MSP430x3xx family of controllers has the following features and capabilities:

- Ultralow-power architecture:
  - 0.1– 400 µA nominal operating current @1 MHz
  - 2.5 – 5.5 V operation available
  - 6 µs wake-up from standby mode
  - Extensive interrupt capability relieves need for polling
- Flexible and powerful processing capabilities:
  - Seven source-address modes
  - Four destination-address modes
  - Only 27 core instructions
  - Prioritized, nested interrupts
  - No interrupt or subroutine level limits
  - Large register file
  - Ram execution capability
  - Efficient table processing
  - Fast hex-to-decimal conversion
- Extensive, memory-mapped peripheral set including:
  - Integrated 14-bit A/D converter
  - Multiple timers and PWM capability
  - Slope A/D conversion (all devices)
  - Integrated USART
  - Integrated LCD driver
  - Watchdog Timer
  - Multiple I/O with extensive interrupt capability
  - Integrated programmable oscillator
  - 32-kHz crystal oscillator (all devices)
- Powerful, easy-to-use development tools including:
  - Simulator (including peripheral and interrupt simulation)
  - C compiler
  - Assembler
  - Linker
  - Emulators (ICE and JTAG)
  - Evaluation kits
  - Device programmer
  - Application notes
  - Example code
- Versatile ultralow-power device options including:
  - Masked ROM
  - OTP (in-system programmable)
  - EPROM (UV-erasable, in-system programmable)
  - –40°C to 85°C operating temperature range
  - Up to 64K addressing space
  - Memory mixes to support all types of applications

1.2 31x Devices

The 31x devices contain the following peripherals:

- FLL clock system (on-chip DCO + crystal oscillator)
- Watchdog Timer/General-Purpose Timer
- Timer/Port (2 8-bit or 1 16-bit timer with analog comparator, 5 outputs, and 1 I/O. Ideal for slope A/D conversion)
- Basic Timer1 (2 8-bit timers or 1 16-bit timer)
- LCD Controller/Driver (up to 92 segments)
- 8-Bit Timer/Counter (8-bit counter with preload. Can be used as UART)
- I/O Port0 (8 I/O's all with interrupt)

Available 31x devices are:

MSP430C311S	2KB ROM, 128B RAM
MSP430C312	4KB ROM, 256B RAM
MSP430C314	12KB ROM, 512B RAM
MSP430C315	16KB ROM, 512B RAM
MSP430P315	16KB OTP, 512B RAM
MSP430P315S	16KB OTP, 512B RAM
PMS430E315	16KB EPROM, 512B RAM

1.3 32x Devices

The 32x devices contain the following peripherals:

- FLL clock system (on-chip DCO + crystal oscillator)
- Watchdog Timer/General-Purpose Timer
- Timer/Port (2 8-bit or 1 16-bit timer with analog comparator, 5 outputs, and 1 I/O. Ideal for slope A/D conversion)
- Basic Timer1 (2 8-bit timers or 1 16-bit timer)
- LCD Controller/Driver (up to 84 segments)
- 8-bit Timer/Counter (8-bit counter with preload. Can be used as UART)
- I/O Port0 (8 I/O's all with interrupt)
- ADC12+2 (14-bit A/D)

Available 32x devices are:

MSP430C323	8KB ROM, 256B RAM
MSP430C325	16KB ROM, 512B RAM
MSP430P325A	16KB OTP, 512B RAM
PMS430E325A	16KB EPROM, 512B RAM

1.4 33x Devices

The 33x devices contain the following peripherals:

- FLL clock system (on-chip DCO + crystal oscillator)
- Watchdog Timer/General-Purpose Timer
- Timer/Port (2 8-bit or 1 16-bit timer with analog comparator, 5 outputs, and 1 I/O. Ideal for slope A/D conversion)
- Basic Timer1 (2 8-bit timers or 1 16-bit timer)
- LCD Controller/Driver (up to 120 segments)
- 8-Bit Timer/Counter (8-bit counter with preload. Can be used as UART)
- I/O Port0 (8 I/O's all with interrupt)
- I/O Port1,2 (8 I/O's each all with interrupt)
- I/O Port3,4 (8 I/O's each)
- Hardware Multiplier (16 × 16-bit)
- Timer_A (16-bit timer with 5 capture/compare registers and PWM output)
- USART

Available 33x devices are:

MSP430C336	24KB ROM, 1KB RAM
MSP430C337	32KB ROM, 1KB RAM
MSP430P337A	32KB OTP, 1KB RAM
PMS430E337A	32KB EPROM, 1KB RAM

Chapter 2

Architectural Overview

This section describes the basic functions of an MSP430-based system.

The MSP430 devices contain the following main elements:

- Central processing unit
- Program memory
- Data memory
- Operation control
- Peripheral modules
- Oscillator and clock generator

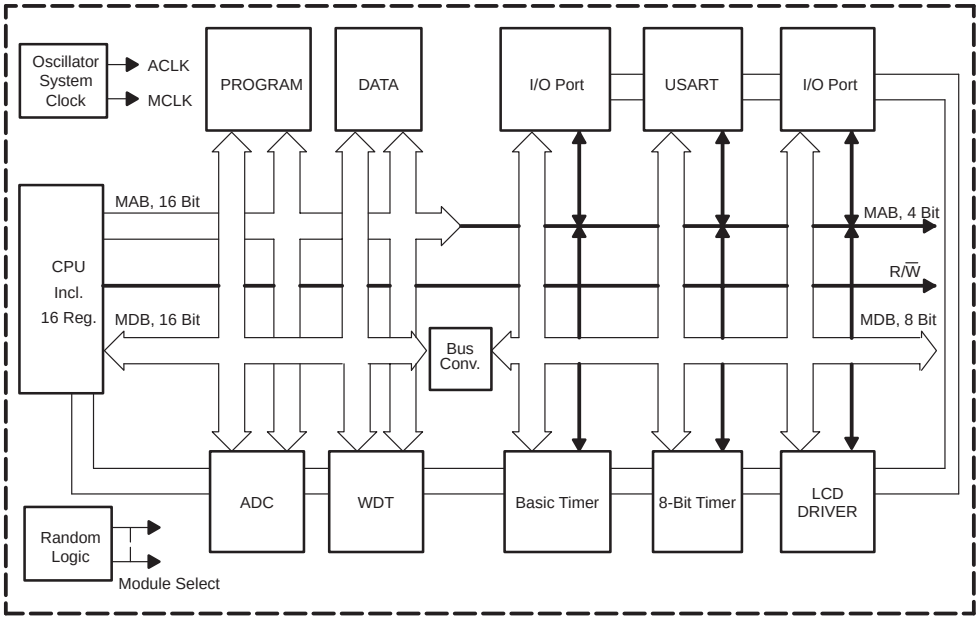
Topic **Page**

2.1	Introduction	2-2
2.2	Central Processing Unit	2-2
2.3	Program Memory	2-3
2.4	Data Memory	2-3
2.5	Operation Control	2-3
2.6	Peripherals	2-4
2.7	Oscillator and Clock Generator	2-4

2.1 Introduction

The architecture of the MSP430 family is based on a memory-to-memory architecture, a common address space for all functional blocks, and a reduced instruction set applicable to all functional blocks as illustrated in Figure 2–1. See specific device data sheets for complete block diagrams of individual devices.

Figure 2–1. MSP430 System Configuration



2.2 Central Processing Unit

The CPU incorporates a reduced and highly transparent instruction set and a highly orthogonal design. It consists of a 16-bit arithmetic logic unit (ALU), 16 registers, and instruction control logic. Four of these registers are used for special purposes. These are the program counter (PC), stack pointer (SP), status register (SR), and constant generator (CGx). All registers, except the constant-generator registers R3/CG2 and part of R2/CG1, can be accessed using the complete instruction set. The constant generator supplies instruction constants, and is not used for data storage. The addressing mode used on CG1 separates the data from the constants.

The CPU control over the program counter, the status register, and the stack pointer (with the reduced instruction set) allows the development of applications with sophisticated addressing modes and software algorithms.

2.3 Program Memory

Instruction fetches from program memory are always 16-bit accesses, whereas data memory can be accessed using word (16-bit) or byte (8-bit) instructions. Any access uses the 16-bit memory data bus (MDB) and as many of the least-significant address lines of the memory address bus (MAB) as required to access the memory locations. Blocks of memory are automatically selected through module-enable signals. This technique reduces overall current consumption. Program memory is integrated as programmable or mask-programmed memory.

In addition to program code, data may also be placed in the ROM section of the memory map and may be accessed using word or byte instructions; this is useful for data tables, for example. This unique feature gives the MSP430 an advantage over other microcontrollers, because the data tables do not have to be copied to RAM for usage.

Sixteen words of memory are reserved for reset and interrupt vectors at the top of the 64-kilobytes address space from 0FFFFh down to 0FFE0h.

2.4 Data Memory

The data memory is connected to the CPU through the same two buses as the program memory (ROM): the memory address bus (MAB) and the memory data bus (MDB). The data memory can be accessed with full (word) data width or with reduced (byte) data width.

Additionally, because the RAM and ROM are connected to the CPU via the same busses, program code can be loaded into and executed from RAM. This is another unique feature of the MSP430 devices, and provides valuable, easy-to-use debugging capability.

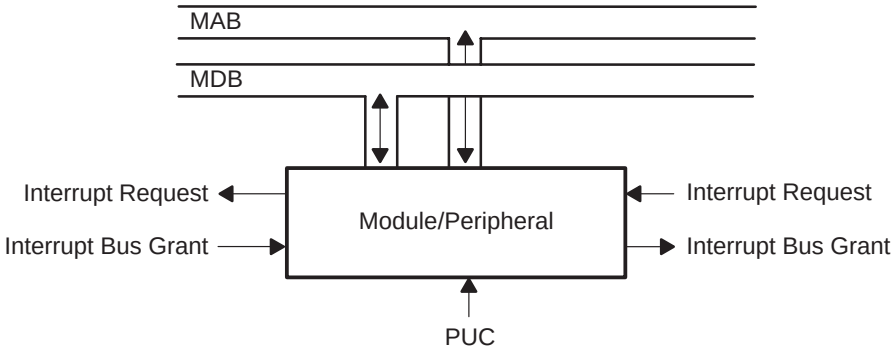
2.5 Operation Control

The operation of the different MSP430 members is controlled mainly by the information stored in the special–function registers (SFRs). The different bits in the SFRs enable interrupts, provide information about the status of interrupt flags, and define the operating modes of the peripherals. By disabling peripherals that are not needed during an operation, total current consumption can be reduced. The individual peripherals are described later in this manual.

2.6 Peripherals

Peripheral modules are connected to the CPU through the MAB, MDB, and interrupt service and request lines. The MAB is usually a 5-bit bus for most of the peripherals. The MDB is an 8-bit or 16-bit bus. Most of the peripherals operate in byte format. Modules with an 8-bit data bus are connected by bus-conversion circuitry to the 16-bit CPU. The data exchange with these modules must be handled with byte instructions. The SFRs are also handled with byte instructions. The operation for 8-bit peripherals follows the order described in Figure 2–2.

Figure 2–2. Bus Connection of Modules/Peripherals



2.7 Oscillator and Clock Generator

The oscillator is designed for the commonly used 32,768 Hz, low-current-consumption clock crystal. All analog components are integrated into the MSP430x3xx; only the crystal needs to be connected with no other external components required.

In addition to the crystal oscillator, all MSP430 devices contain a digitally-controlled RC oscillator (DCO). The DCO is different from RC oscillators found on other microcontrollers because it is digitally controllable and tuneable.

MSP430x3xx devices contain an additional logic block called the frequency locked loop (FLL). The FLL continuously and automatically adjusts the frequency of the DCO relative to the 32768-Hz crystal oscillator to stabilize the DCO over voltage and temperature. This provides an effective, stable, ultralow-power oscillator for the CPU and peripherals.

Clock source selection for peripherals is very flexible. Most peripherals are capable of using the 32768-Hz crystal oscillator clock or the DCO clock. The CPU executes from the DCO clock. See Chapter 7 for details on the clock module.

Chapter 3

System Resets, Interrupts, and Operating Modes

This chapter discusses the MSP430x3xx system resets, interrupts, and operating modes.

Topic **Page**

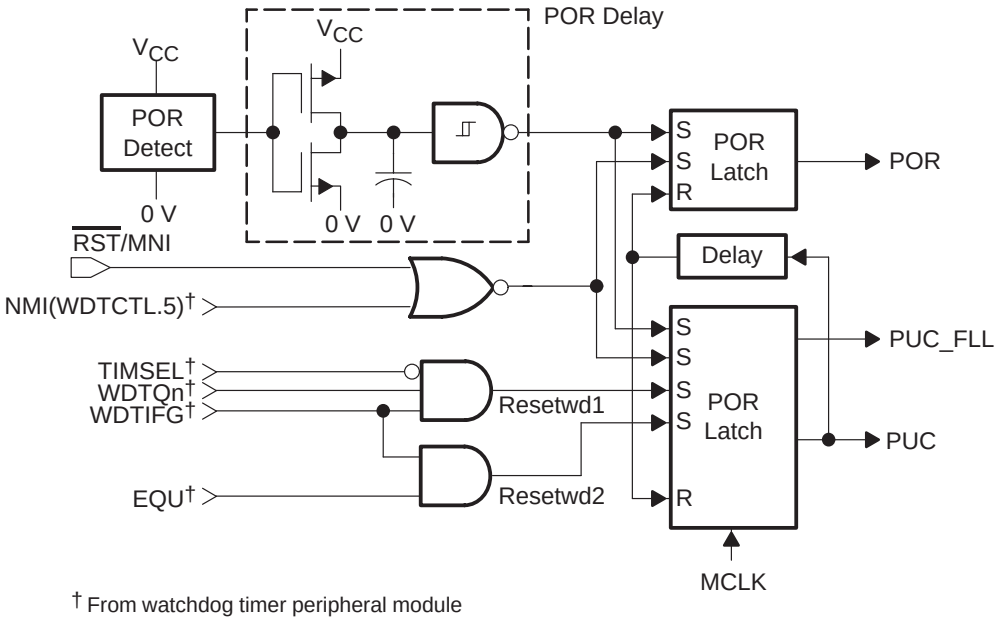
3.1	System Reset and Initialization	3-2
3.2	Global Interrupt Structure	3-5
3.3	MSP430 Interrupt-Priority Scheme	3-6
3.4	Interrupt Processing	3-9
3.5	Operating Modes	3-14
3.6	Basic Hints for Low-Power Applications	3-20

3.1 System Reset and Initialization

3.1.1 Introduction

The MSP430 system reset circuitry (shown in Figure 3–1) sources two internal reset signals: power-on reset (POR) and power-up clear (PUC). Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

Figure 3–1. Power-on Reset and Power-up Clear Schematic



† From watchdog timer peripheral module

A POR is a device reset. It is only generated by the two following events:

- Powering up the device
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog Timer expiration (in watchdog mode only)
- Watchdog Timer security key violation
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the NMI mode

Note:

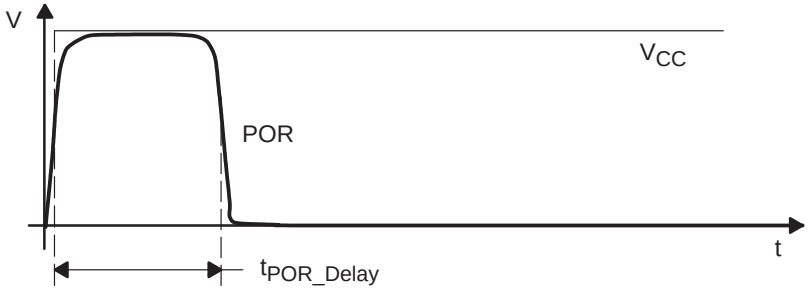
If desired, software can cause a PUC by simply writing to the watchdog timer control register with an incorrect password.

Note:

Generation of the POR/PUC signals does not necessarily generate a system reset interrupt. Anytime a POR is activated, a system reset interrupt is generated. However, when a PUC is activated, a system reset interrupt may or may not be generated. Instead, a lower priority interrupt vector may be generated, depending on what action caused the PUC. Each device data sheet gives a detailed table of what action generates each interrupt. This table should be consulted for the proper handling of all interrupts.

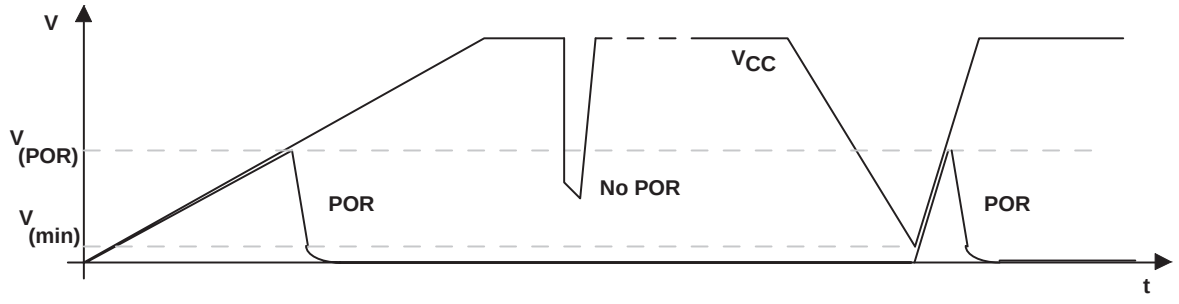
When the V_{CC} supply provides a fast rise time as shown in Figure 3–2, the POR delay provides enough active time on the POR signal to allow the signal to initialize the circuitry correctly after power up. When the V_{CC} rise time is slow, as shown in Figure 3–3, the POR detector holds the POR signal active until V_{cc} has risen above the $V_{(POR)}$ level. This also ensures a correct initialization.

Figure 3–2. Power-On Reset Timing on Fast V_{CC} Rise Time



If power to the chip is cycled, the supply voltage V_{CC} must fall below the $V_{(min)}$ (see Figure 3–3) to ensure that another POR signal occurs when V_{CC} is powered up again. If V_{CC} does not fall below $V_{(min)}$ during a cycle or a glitch, a POR is not generated and power-up conditions do not set correctly.

Figure 3–3. Power-on Reset Timing on Slow V_{CC} Rise Time



3.1.2 Device Initialization after System Reset

After a device reset (POR/PUC combination), the initial system conditions are:

- ❑ I/O pins switched to input mode (see note below).
- ❑ I/O flags are cleared as described in the I/O chapter (see note below).
- ❑ Other peripherals and registers initialized as described in their respective chapters.
- ❑ Status register is reset.
- ❑ Program counter is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address.
- ❑ FLL begins regulation of the DCO.

Note:

I/O pins and flags are only initialized after power up. After the '430 is powered and running, if a reset is generated with $\overline{\text{RST}}$/NMI pin (in reset mode), the I/O pins are unaffected.

After a system reset, the user program can evaluate the various flags to determine the source of the reset and take appropriate action.

The initial state of registers and peripherals is discussed in each applicable section of this manual. Each register is shown with a key indicating the accessibility of the register and the initial condition, for example, *rw–(0)*, or *rw–0*. In these examples, the *r* indicates read, the *w* indicates write, and the value after the dash indicates the initial condition. If the value is in parenthesis, the initial condition takes effect only after a POR – a PUC alone will not effect the bit(s). If the value is not in parenthesis, it takes effect after a PUC alone or after a POR/PUC combination. Some examples follow:

Type	Description
rw–(0)	Read/write, reset with POR
rw–0	Read/write, reset with POR or PUC
r–1	Read only, set with POR or PUC
r	Read only, no initial state
w	Write only, no initial state

3.2 Global Interrupt Structure

There are four types of interrupts:

- System reset
- Maskable
- Nonmaskable
- (Non)maskable

System reset (POR/PUC) is discussed in section 3.1.

Maskable interrupts are caused by:

- A Watchdog-Timer overflow (if timer mode is selected)
- Other modules with interrupt capability

Nonmaskable interrupts are not maskable in any way. No individual interrupt enable bit is implemented for them, and the general interrupt enable bit (GIE) has no effect on them.

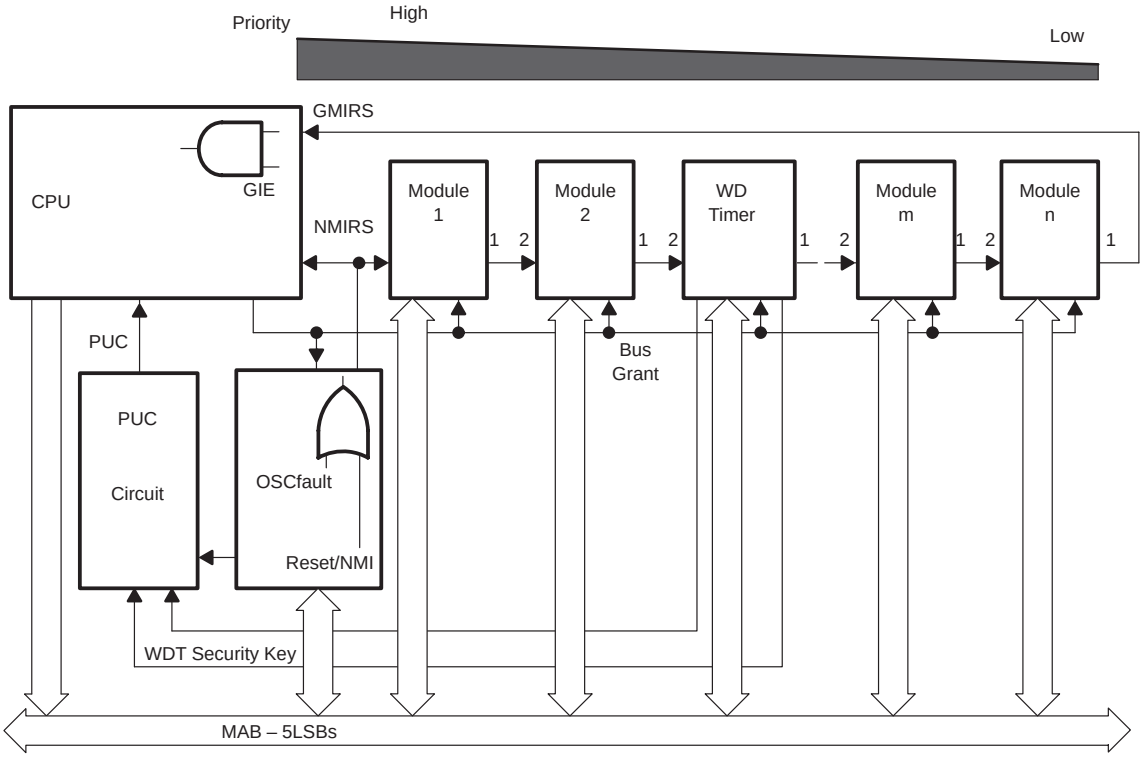
(Non)maskable interrupts are not masked by the general interrupt enable bit (GIE) but are individually enabled or disabled by an individual interrupt enable bit. When a (non)maskable interrupt is accepted, the corresponding interrupt enable bit is automatically reset, therefore disabling the interrupt for execution of the interrupt service routine (ISR). The RETI (return from interrupt) instruction has no effect on the individual enable bits of the (non)maskable interrupts. So the software must set the corresponding interrupt enable bit in the ISR before execution of the RETI instruction for the interrupt to be re-enabled after the ISR.

A nonmaskable NMI interrupt can be generated by an edge on the RST/NMI pin if NMI mode is selected. Additionally, a (non)maskable interrupt event can be generated when an oscillator fault occurs, if the oscillator fault interrupt enable bit is set.

3.3 MSP430 Interrupt-Priority Scheme

The interrupt priority of the modules, as shown in Figure 3–4, is defined by the arrangement of the modules in the connection chain: the nearer a module is to the CPU/NMIRS, the higher the priority.

Figure 3–4. Interrupt Priority Scheme



Reset and NMI, as shown in Figure 3–5, can only be used as alternative interrupts because they use the same input pin. The associated control bits are located in the watchdog timer control register shown in Figure 3–6, and are password protected.

Figure 3–5. Block Diagram of NMI Interrupt Sources

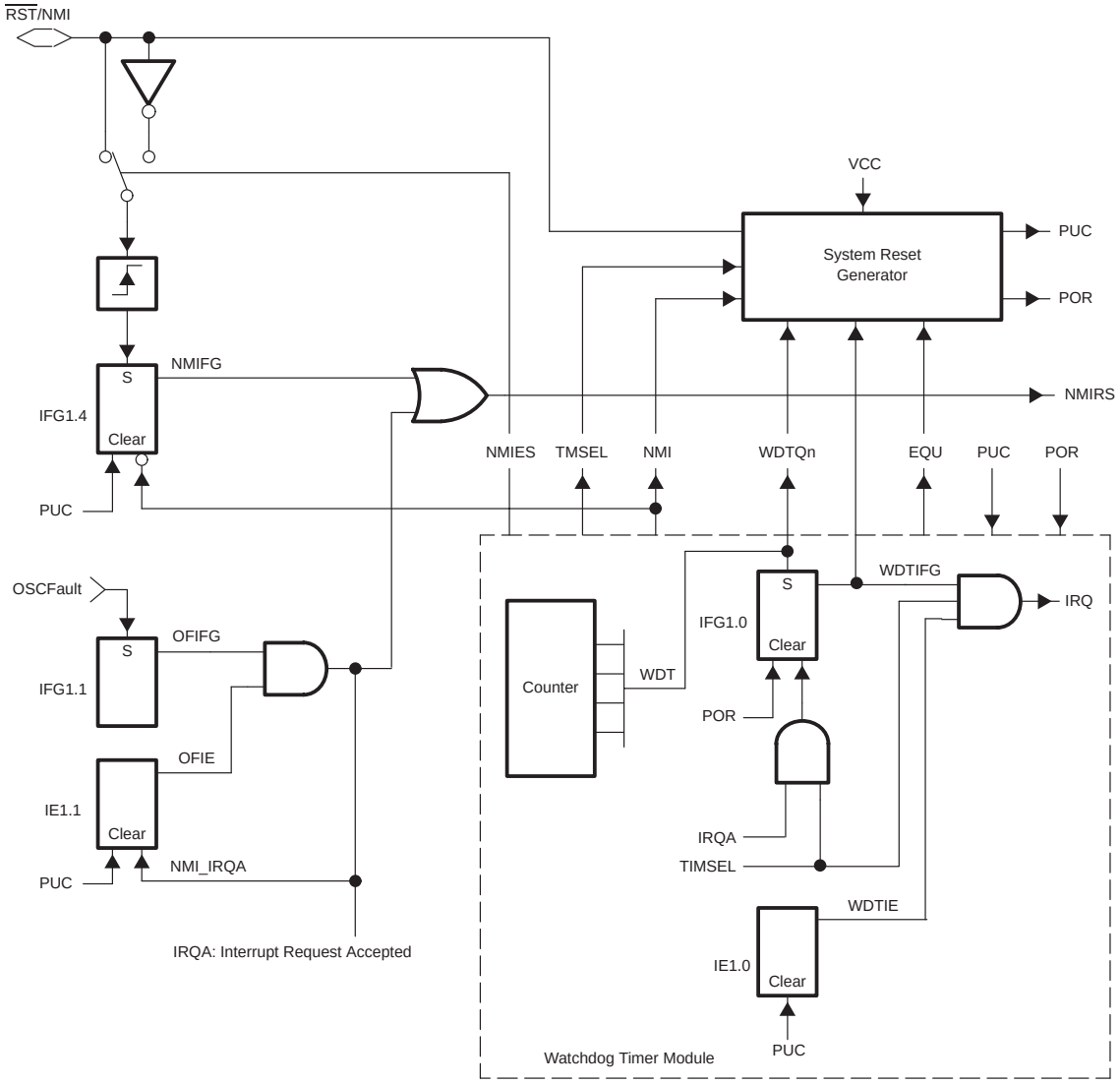


Figure 3–6. $\overline{RST}$/NMI Mode Selection

WDTCTL
0120h

7							0
HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	IS1	IS0
rw-0	rw-0	rw-0	rw-0	(w)-0	rw-0	rw-0	rw-0

BITS 0–4,7 See *Timers* chapter.

BIT 5: The NMI bit selects the function of the $\overline{RST}$/NMI input pin. It is cleared after a PUC signal.

NMI = 0: The $\overline{RST}$/NMI input works as reset input. As long as the $\overline{RST}$/NMI pin is held low, the internal PUC signal is active (level-sensitive).

NMI = 1: The $\overline{RST}$/NMI input works as an edge-sensitive, nonmaskable interrupt input.

BIT 6: This bit selects the activating edge of the $\overline{RST}$/NMI input if the NMI function is selected. It is cleared after a PUC signal.

NMIES = 0: A rising edge triggers an NMI interrupt.

NMIES = 1: A falling edge triggers an NMI interrupt.

3.3.1 Operation of Global Interrupt—Reset/NMI

If the $\overline{\text{RST}}$/NMI pin is set to the reset function, the CPU is held in the reset state as long as the $\overline{\text{RST}}$/NMI pin is held low. After the input changes to a high state, the CPU starts program execution at the word address stored in word location 0FFFEh (reset vector).

If the $\overline{\text{RST}}$/NMI pin is set to the NMI function, a signal edge (selected by the NMIES bit) will generate an unconditional interrupt. When accepted, program execution begins at the address stored in location 0FFFCh. The $\overline{\text{RST}}$/NMI flag in the SFR IFG1.4 is also set.

Note:

When configured in the NMI mode, a signal generating an NMI event should not hold the $\overline{\text{RST}}$/NMI pin low, unless it is intended to hold the processor in reset. When an NMI event occurs on the pin, the PUC signal is activated, thus resetting the bits in the WDTCTL register. This results in the $\overline{\text{RST}}$/NMI pin being configured in the reset mode. If the signal on the $\overline{\text{RST}}$/NMI pin that generated the NMI event remains low, the processor will be held in the reset state.

When NMI mode is selected and the NMI edge select bit is changed, an NMI can be generated, depending on the actual level at $\overline{\text{RST}}$/NMI pin. When the NMI edge select bit is changed before selecting the NMI mode, no NMI is generated.

3.3.2 Operation of Global Interrupt—Oscillator Fault Control

The oscillator fault signal warns of a possible error condition with the crystal oscillator. It is generated by different events in the FLL Clock and Basic Clock systems.

3.3.2.1 Oscillator Fault Control in the FLL Clock System

The oscillator fault signal is triggered if the 5MSB (2^9–2^5) DCO control taps in the SCFI1 register are equal to 0, or greater than or equal to 28h. The oscillator fault signal can be enabled to generate an NMI by bit IE1.1 in the SFRs. The interrupt flag IFG1.1 in the SFRs can then be tested by the interrupt service routine to determine if the NMI was caused by an oscillator fault. See chapter 7 for more details on the operation of the DCO oscillator and the FLL.

3.4 Interrupt Processing

The MSP430 programmable interrupt structure allows flexible on-chip and external interrupt configurations to meet real-time interrupt-driven system requirements. Interrupts may be initiated by the processor's operating conditions such as watchdog overflow; or by peripheral modules or external events. Each interrupt source can be disabled individually by an interrupt enable bit, or all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register.

Whenever an interrupt is requested and the appropriate interrupt enable bit and general interrupt enable (GIE) bit are set, the interrupt service routine becomes active as follows:

1) CPU active: The currently executing instruction is completed.

2) CPU stopped: The low-power modes are terminated.

3) The program counter pointing to the next instruction is pushed onto the stack.

4) The status register is pushed onto the stack.

5) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.

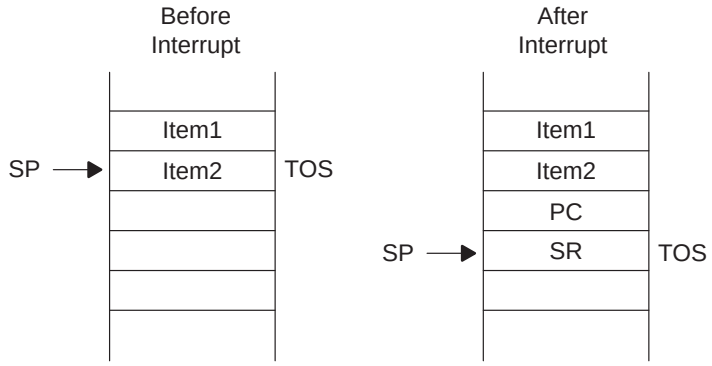
6) The appropriate interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.

7) The GIE bit is reset; the CPUOff bit, the OscOff bit, and the SCG1 bit are cleared; the status bits V, N, Z, and C are reset. SCG0 is left unchanged, and loop control remains in the previous operating condition.

8) The content of the appropriate interrupt vector is loaded into the program counter: the program continues with the interrupt handling routine at that address.

The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the appropriate interrupt-service routine first instruction, as shown in Figure 3–7.

Figure 3–7. Interrupt Processing



The interrupt handling routine terminates with the instruction:

`RETI` (return from an interrupt service routine)

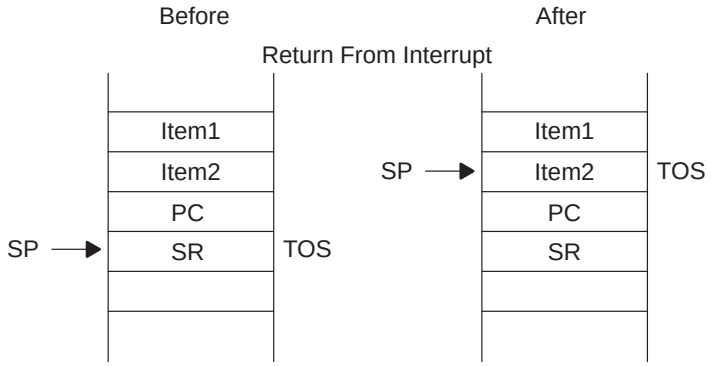
which performs the following actions:

1) The status register with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings utilized during the interrupt service routine.

2) The program counter pops from the stack and begins execution at the point where it was interrupted.

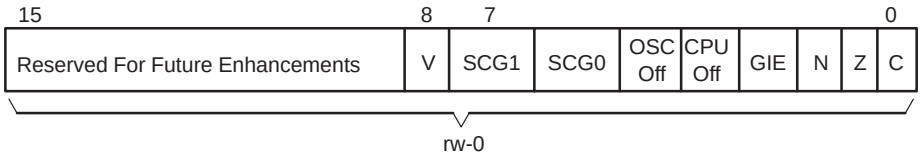
The return from the interrupt is illustrated in Figure 3–8.

Figure 3–8. Return from Interrupt



A RETI instruction takes five cycles. Interrupt nesting is activated if the GIE bit is set inside the interrupt handling routine. The GIE bit is located in status register SR/R2, which is included in the CPU as shown in Figure 3–9.

Figure 3–9. Status Register (SR)



Apart from the GIE bit, other sources of interrupt requests can be enabled/disabled individually or in groups. The interrupt enable flags are located together within two addresses of the special-function registers (SFRs). The program-flow conditions on interrupt requests can be easily adjusted using the interrupt enable masks. The hardware serves the highest priority within the empowered interrupt source.

3.4.1 Interrupt Control Bits in Special-Function Registers (SFRs)

Most of the interrupt control bits, interrupt flags, and interrupt enable bits are collected in SFRs under a few addresses, as shown in Table 3–1. The SFRs are located in the lower address range and are implemented in byte format. SFRs must be accessed using byte instructions.

Table 3–1. Interrupt Control Bits in SFRs

Address	7 0
000Fh	Not yet defined or implemented
000Eh	Not yet defined or implemented
000Dh	Not yet defined or implemented
000Ch	Not yet defined or implemented
000Bh	Not yet defined or implemented
000Ah	Not yet defined or implemented
0009h	Not yet defined or implemented
0008h	Not yet defined or implemented
0007h	Not yet defined or implemented
0006h	Not yet defined or implemented
0005h	Module enable 2 (ME2.x)
0004h	Module enable 1 (ME1.x)
0003h	Interrupt flag reg. 2 (IFG2.x)
0002h	Interrupt flag reg. 1 (IFG1.x)
0001h	Interrupt enable 2 (IE2.x)
0000h	Interrupt enable 1 (IE1.x)

The MSP430 family supports SFRs by applying the correct logic and functions to each individual module. Each module interrupt source can be individually enabled or disable using the bits described in Table 3–2.

The interrupt-flag registers are described in Table 3–3. The module-enable bits are described in Table 3–4.

Table 3–2. Interrupt Enable Registers 1 and 2

Bit Position	Short Form	Initial State†	Comments
IE1.0	WDTIE	Reset	Watchdog Timer enable signal. Inactive if watchdog mode is selected. Active if Watchdog Timer is configured as general-purpose timer.
IE1.1	OFIE	Reset	Oscillator fault interrupt enable
IE1.2	P0IE.0	Reset	Dedicated I/O P0.0 interrupt enable
IE1.3	P0IE.1	Reset	Dedicated I/O P0.1 or 8-Bit Timer/Counter interrupt enable
IE1.4		Reset	Reserved
IE1.5		Reset	Reserved
IE1.6		Reset	Reserved
IE1.7		Reset	Reserved
IE2.0	URXIE	Reset	USART receive interrupt enable (33x devices)
IE2.1	UTXIE	Reset	USART transmit interrupt enable (33x devices)
IE2.2	ADIE/TPIE	Reset	ADC enable (32x devices), Timer/Port enable (31x devices)
IE2.3	TPIE	Reset	Timer/Port (32x, 33x devices)
IE2.4		Reset	Reserved
IE2.5		Reset	Reserved
IE2.6		Reset	Reserved
IE2.7	BTIE	Reset	Basic timer interrupt enable signal

† The initial state is the logical state after the PUC signal.

Table 3–3. Interrupt Flag Register 1 and 2

Bit Position	Short Form	Initial State	Comments
IFG1.0	WDTIFG	Set	Set on Watchdog Timer overflow in watchdog mode or security key violation.
		Or reset	Reset with VCC power-up, or a reset condition at the $\overline{\text{RST}}$/NMI pin in reset mode.
IFG1.1	OFIFG	Set	Flag set on oscillator fault
IFG1.2	P0IFG.0	Reset	Dedicated I/O P0.0
IFG1.3	P0IFG.1	Reset	Dedicated I/O P0.1 or 8-Bit Timer/Counter
IFG1.4	NMIIFG	Reset	Set through the $\overline{\text{RST}}$/NMI pin
IFG1.5			Reserved
IFG1.6			Reserved
IFG1.7			Reserved
IFG2.0	URXIFG	Reset	USART receive flag (33x devices)
IFG2.1	UTXIFG	Set	USART transmitter ready (33x devices)
IFG2.2	ADIFG	Reset	ADC, set on end-of-conversion
IFG2.3			Reserved
IFG2.4			Reserved
IFG2.5			Reserved
IFG2.6			Reserved
IFG2.7	BTIFG	Unchanged	Basic timer flag

Table 3–4. Module Enable Registers 1 and 2

Bit Position	Short Form	Initial State	Comments
ME1.0			Reserved
ME1.1			Reserved
ME1.2			Reserved
ME1.3			Reserved
ME1.4			Reserved
ME1.5			Reserved
ME1.6			Reserved
ME1.7			Reserved
ME2.0	URXE USPIIE	Reset Reset	USART receiver enable (33x devices, UART mode) USART transmit and receive enable (33x devices, SPI mode)
ME2.1	UTXE	Reset	USART transmit enable (33x devices, UART mode)
ME2.2			Reserved
ME2.3			Reserved
ME2.4			Reserved
ME2.5			Reserved
ME2.6			Reserved
ME2.7			Reserved

3.4.2 Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the ROM, using the address range 0FFFFh – 0FFE0h as described in Table 3–5. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence. The interrupt vectors for 3xx devices are shown in Table 3–5 in decreasing order of priority. See device data sheet for interrupt vectors for a specific device.

Table 3–5. Interrupt Sources, Flags, and Vectors of 3xx Configurations

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Power-up, ext. reset, watchdog	WDTIFG	Reset	0FFFEh	15 (highest)
NMI OSC. fault	NMIIFG OFIFG†	See Note (Non)maskable¶	0FFFCh	14
Dedicated I/O	P0IFG.0	Maskable	0FFFAh	13
Dedicated I/O	P0IFG.1	Maskable	0FFF8h	12
		Maskable	0FFF6h	11
Watchdog Timer	WDTIFG	Maskable	0FFF4h	10
Timer_A	CCIFG0	Maskable	0FFF2h	9
Timer_A	TAIFG	Maskable	0FFF0h	8
USART receive	URXIFG	Maskable	0FFEEh	7
USART transmit	UTXIFG	Maskable	0FFECh	6
ADC, Timer/Port‡	ADCIFG	Maskable	0FFEAh	5
Timer/Port§		Maskable	0FFE8h	4
Port P2	P2IFG.07†	Maskable	0FFE6h	3
Port P1	P1IFG.07†	Maskable	0FFE4h	2
Basic timer	BTIFG	Maskable	0FFE2h	1
Port 0	P0IFG.27†	Maskable	0FFE0h	0 (lowest)

† Multiple source flags
‡ Timer/Port vector in ’31x configuration
§ Timer/Port vector in ’32x and ’33x configuration
¶ Interrupt can be disabled with individual interrupt enable bit, but not with the general interrupt enable bit, GIE.

3.4.2.1 External Interrupts

All eight bits of ports P0, P1, and P2 are designed for interrupt processing of external events. All individual I/O bits are independently programmable. Any combinations of inputs, outputs, and interrupt conditions are possible. This allows easy adaptation to different I/O configurations. See Chapter 8 for more details on I/O ports.

3.5 Operating Modes

The MSP430 family was developed for ultra-low power applications and uses different levels of operating modes. The MSP430 operating modes, shown in Figure 3–10, give advanced support to various requirements for ultra-low power and ultra-low energy consumption. This support is combined with an intelligent management of operations during the different module and CPU states. An interrupt event wakes the system from each of the various operating

modes and the RETI instruction returns operation to the mode that was selected before the interrupt event.

The ultra-low power system design which uses complementary metal-oxide semiconductor (CMOS) technology, takes into account three different needs:

- The desire for speed and data throughput despite conflicting needs for ultralow-power
- Minimization of individual current consumption
- Limitation of the activity state to the minimum required by the use of low-power modes

There are four bits that control the CPU and the system clock generator: CPUOff, OscOff, SCG0, and SCG1. These four bits support discontinuous active mode (AM) requests, to limit the time period of the full operating mode, and are located in the status register. The major advantage of including the operating mode bits in the status register is that the present state of the operating condition is saved onto the stack during an interrupt service request. As long as the stored status register information is not altered, the processor continues (after RETI) with the same operating mode as before the interrupt event. Another program flow may be selected by manipulating the data stored on the stack or the stack pointer. Being able to access the stack and stack pointer with the instruction set allows the program structures to be individually optimized, as illustrated in the following program flow:

- Enter interrupt routine

The interrupt routine is entered and processed if an enabled interrupt awakens the MSP430:

  - The SR and PC are stored on the stack, with the content present at the interrupt event.
  - Subsequently, the operation mode control bits OscOff, SCG1, and CPUOff are cleared automatically in the status register.

- Return from interrupt

Two different modes are available to return from the interrupt service routine and continue the flow of operation:

  - Return with low-power mode bits set. When returning from the interrupt, the program counter points to the next instruction. The instruction pointed to is not executed, since the restored low-power mode stops CPU activity.
  - Return with low-power mode bits reset. When returning from the interrupt, the program continues at the address following the instruction that set the OscOff or CPUOff-bit in the status register. To use this mode, the interrupt service routine must reset the OscOff, CPUOff, SCGO, and SCG1 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

The software can configure five operating modes:

- Active mode AM; SCG1=0, SCG0=0, OscOff=0, CPUOff=0:
  CPU clocks are active
- Low-power mode 0 (LPM0); SCG1=0, SCG0=0, OscOff=0, CPUOff=1:
  CPU is disabled
  ACLK and MCLK remain active
  Loop control for MCLK remains active
- Low-power mode 1 (LPM1); SCG1=0, SCG0=1, OscOff=0, CPUOff=1:
  CPU is disabled
  Loop control for MCLK is disabled
  ACLK and MCLK remain active
- Low-power mode 2 (LPM2); SCG1=1, SCG0=0, OscOff=0, CPUOff=1:
  CPU is disabled
  MCLK and loop control for MCLK are disabled
  DCO's dc-generator remains enabled
  ACLK remains active
- Low-power mode 3 (LPM3); SCG1=1, SCG0=1, OscOff=0, CPUOff=1:
  CPU is disabled
  MCLK and loop control for MCLK are disabled
  DCO oscillator is disabled
  DCO's dc-generator is disabled
  ACLK remains active
- Low-power mode 4 (LPM4); SCG1=X, SCG0=X, OscOff=1, CPUOff=1:
  CPU is disabled
  ACLK is disabled
  MCLK and loop control for MCLK are disabled
  DCO oscillator is disabled
  DCO's dc-generator is disabled
  Crystal oscillator is stopped

Note:

Peripheral operation is not halted by CPUOff. Peripherals are controlled by their individual control registers.

Table 3–6. Low-Power Mode Logic Chart

	SCG1	SCG0	OscOff	CPUOff
LPM0	0	0	0	1
LPM1	0	1	0	1
LPM2	1	0	0	1
LPM3	1	1	0	1
LPM4	X	X	1	1

These modes are illustrated in Figure 3–10.

Figure 3–10. MSP430x3xx Family Operating Modes

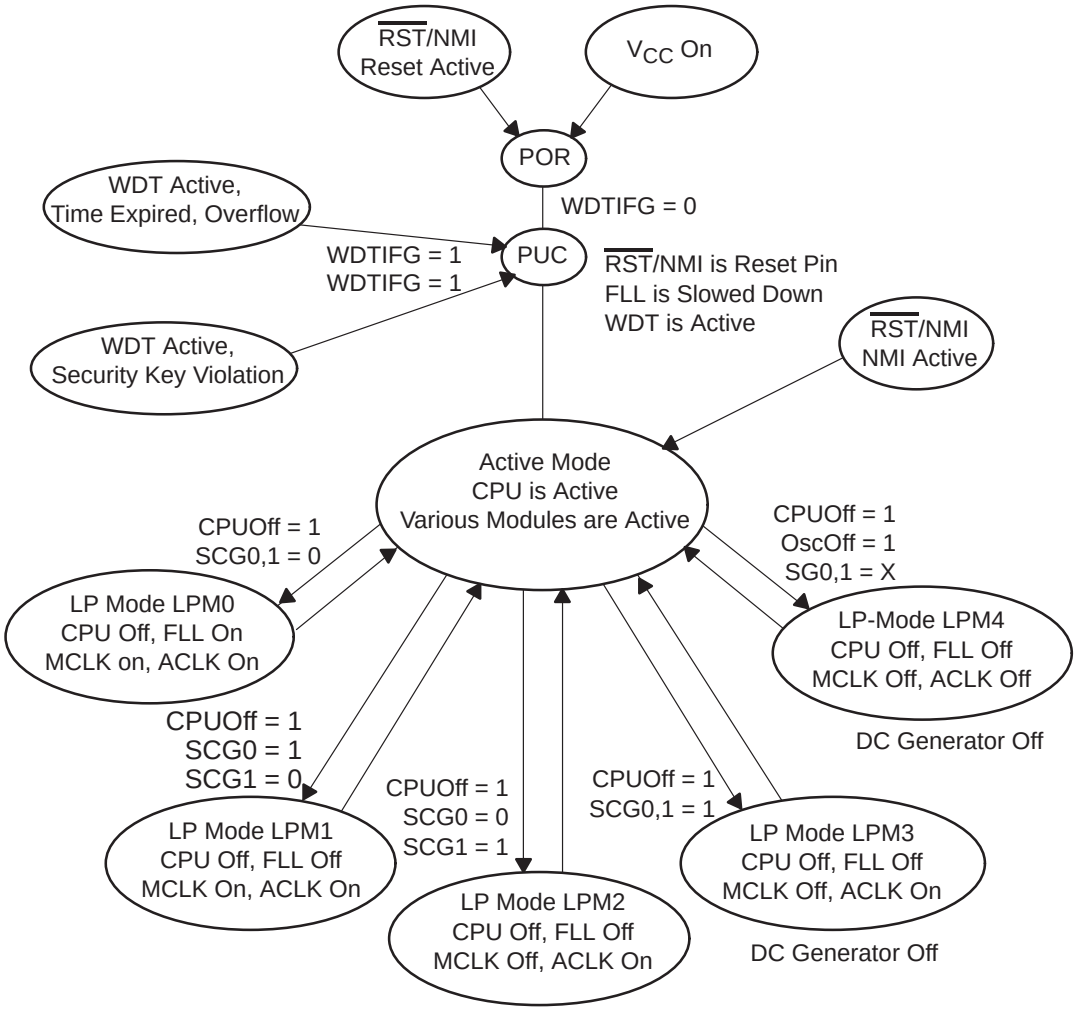
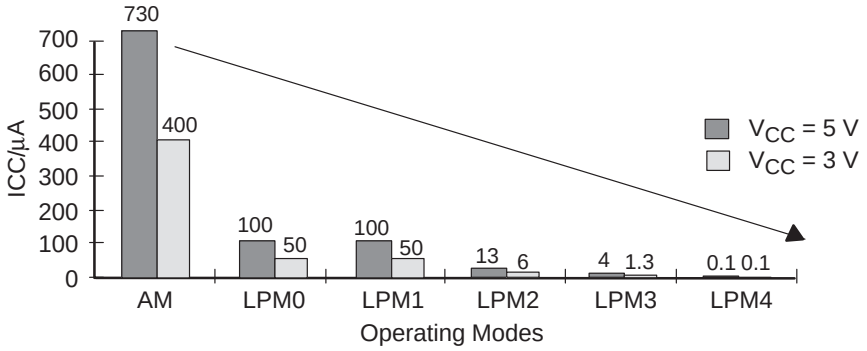


Figure 3–11. Typical Current Consumption vs Operating Modes



The low-power modes 1–4 enable or disable the CPU and the clocks. In addition to the CPU and clocks, enabling or disabling specific peripherals may further reduce total current consumption of the individual modes. The activity state of each peripheral is controlled by the control registers for the individual peripherals. An example is the enable/disable function of the segment lines of the LCD peripheral: they can be turned on or off using a single register bit in the LCD control and mode register. In addition, the SFRs include module enable bits that may be used to enable or disable the operation of specific peripheral modules (see Table 3–4).

3.5.1 Low-Power Modes 0 and 1 (LPM0 and LPM1)

Low-power mode 0 or 1 is selected if bit CPUOff in the status register is set. Immediately after the bit is set the CPU stops operation, and the normal operation of the system core stops. The operation of the CPU halts and all internal bus activities stop until an interrupt request or reset occurs. The system clock generator continues operation, and the clock signals MCLK and ACLK stay active depending on the state of the other three status register bits, SCG0, SCG1, and OscOff.

The peripherals are enabled or disabled according with their individual control register settings, and with the module enable registers in the SFRs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

The following are examples of entering and exiting LPM0. The method shown is applicable to all low-power modes.

The following example describes entering into low-power mode 0.

```
;===Main program flow with switch to CPUOff Mode==============
;
     BIS #18h,SR   ;Enter LPM0 + enable general interrupt GIE
                   ;(CPUOff=1, GIE=1). The PC is incremented
                   ;during execution of this instruction and
                   ;points to the consecutive program step.
     ......        ;The program continues here if the CPUOff
                   ;bit is reset during the interrupt service
                   ;routine. Otherwise, the PC retains its
                   ;value and the processor returns to LPM0.
```

The following example describes clearing low-power mode 0.

```
;===Interrupt service routine================================
    ......          ;CPU is active while handling interrupts
    BIC #10h,0(SP)  ;Clears the CPUOff bit in the SR contents
                    ;that were stored on the stack.

    RETI            ;RETI restores the CPU to the active state
                    ;because the SR values that are stored on
                    ;the stack were manipulated. This occurs
                    ;because the SR is pushed onto the stack
                    ;upon an interrupt, then restored from the
                    ;stack after the RETI instruction.
```

3.5.2 Low-Power Modes 2 and 3 (LPM2 and LPM3)

Low-power mode 2 or 3 is selected if bits CPUOff and SCG1 in the status register are set. Immediately after the bits are set, CPU, and MCLK operations halt and all internal bus activities stop until an interrupt request or reset occurs.

Peripherals that operate with the MCLK signal are inactive because the clock signal is inactive. Peripherals that operate with the ACLK signal are active or inactive according with the individual control registers and the module enable bits in the SFRs. All I/O port pins and the RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or $\overline{\text{RST}}$/NMI.

3.5.3 Low-Power Mode 4 (LPM4)

In low-power mode 4 all activities cease; only the RAM contents, I/O ports, and registers are maintained. Wake up is only possible by enabled external interrupts.

Before activating LPM4, the software should consider the system conditions during the low-power mode period. The two most important conditions are environmental (that is, temperature effect on the DCO), and the clocked operation conditions.

The environment defines whether the value of the frequency integrator should be held or corrected. A correction should be made when ambient conditions are anticipated to change drastically enough to increase or decrease the system frequency while the device is in LPM4.

3.6 Basic Hints for Low-Power Applications

There are some basic practices to follow when current consumption is a critical part of a system application:

- ❑ Switch off analog circuitry when possible.
- ❑ Select the lowest possible operating frequency for the core and the individual peripheral module.
- ❑ Select the weakest drive capability if an LCD is used or switch the drive off.
- ❑ Use the interrupt driven software; the program starts execution rapidly.
- ❑ Tie all unused inputs to an applicable voltage level. The list below defines the correct termination for all unused pins.

PIN	Potential	Comment
❑ **AV_{CC}:**	DV_{CC}	
❑ **AV_{SS}:**	DV_{SS}	
❑ **SV_{CC}:**	open	May be used as a low impedance output
❑ **A0 to A7:**	open	Switched to analog inputs: AEN.x=0
❑ **Xout:**	open	
❑ **XBUF:**	open	
❑ **CI:**	V_{SS}	May be used as a digital input
❑ **TP0.0 to TP0.5:**	open	TP.5 switched to output direction, others to Hi-Z
❑ **Px.0 to Px.7:**	open	Unused ports switched to output direction
❑ **R03:**	V_{SS}	
❑ **R13:**	V_{SS}	
❑ **R23:**	V_{SS}	
❑ **R33:**	open	
❑ **S0 to S1:**	open	
❑ **S3 to S20:**	open	Switched to output direction
❑ **Com0 to Com3:**	open	
❑ **RST/NMI**:	DV_{CC} resp. V_{CC}	Pullup resistor 100k
❑ **TDO:**		
❑ **TDI:**	Refer to device specific datasheets for the correct termination of these pins.	
❑ **TMS:**		
❑ **TCK:**		

Chapter 4

Memory

MSP430 devices are configured as a von-Neumann architecture. It has code memory, data memory, and peripherals in one address space. As a result, the same instructions are used for code, data, or peripheral accesses. Also, code may be executed from RAM.

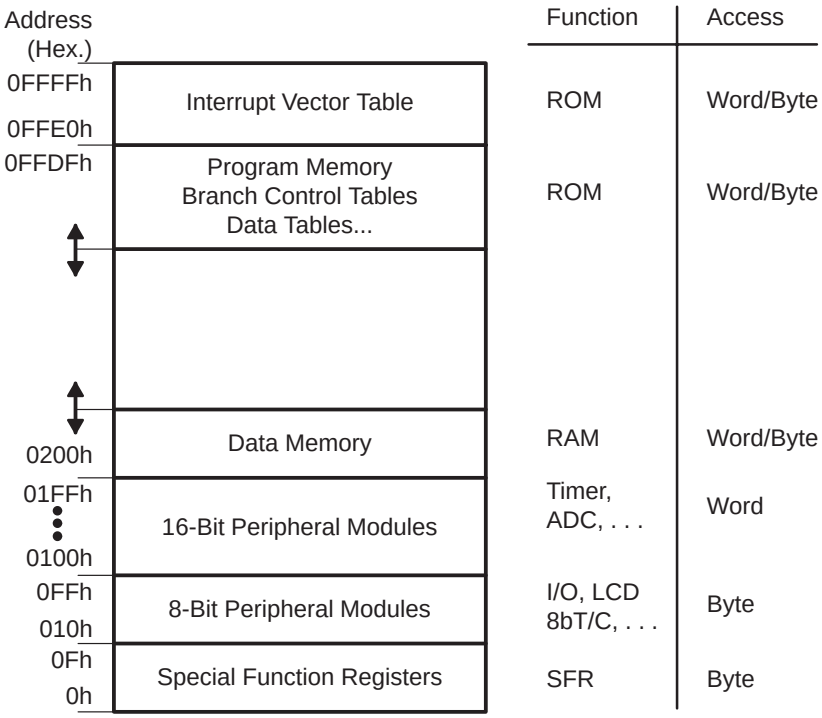
Topic **Page**

4.1	**Introduction**	**4-2**
4.2	**Data in the Memory**	**4-3**
4.3	**Internal ROM Organization**	**4-4**
4.4	**RAM and Peripheral Organization**	**4-6**

4.1 Introduction

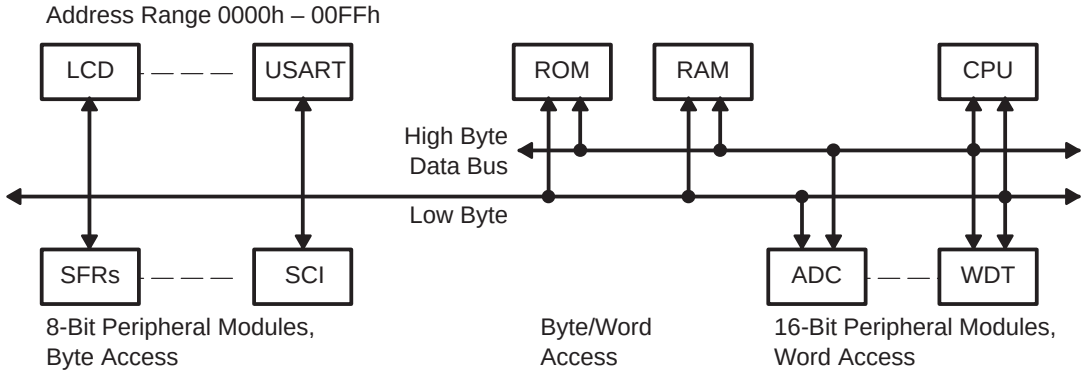
All of the physically separated memory areas (ROM, RAM, SFRs, and peripheral modules) are mapped into the common address space, as shown in Figure 4–1 for the MSP430 family. The addressable memory space is 64KB. Future expansion is possible.

Figure 4–1. Memory Map of Basic Address Space



The memory data bus (MDB) is 16- or 8-bits wide. For those modules that can be accessed with word data the width is always 16 bits. For the other modules, the width is 8 bits, and they must be accessed using byte instructions only. The program memory (ROM) and the data memory (RAM) can be accessed with byte or word instructions.

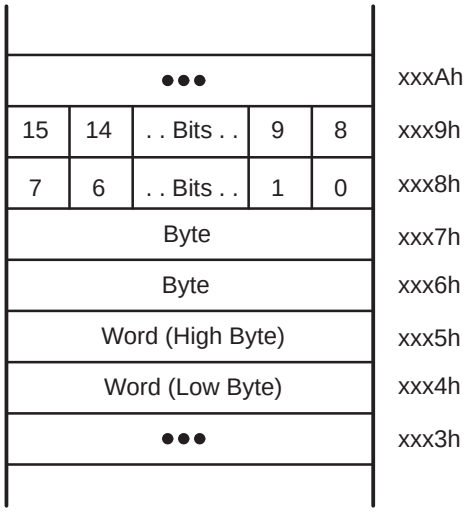
Figure 4–2. Memory Data Bus



4.2 Data in the Memory

Bytes are located at even or odd addresses as shown in Figure 4–3. However, words are only located at even addresses. Therefore, when using word instructions, only even addresses may be used. The low byte of a word is always at an even address. The high byte of a word is at the next odd address after the address of the word. For example, if a data word is located at address xxx2h, then the low byte of that data word is located at address xxx2h, and the high byte of that word is located at address xxx3h.

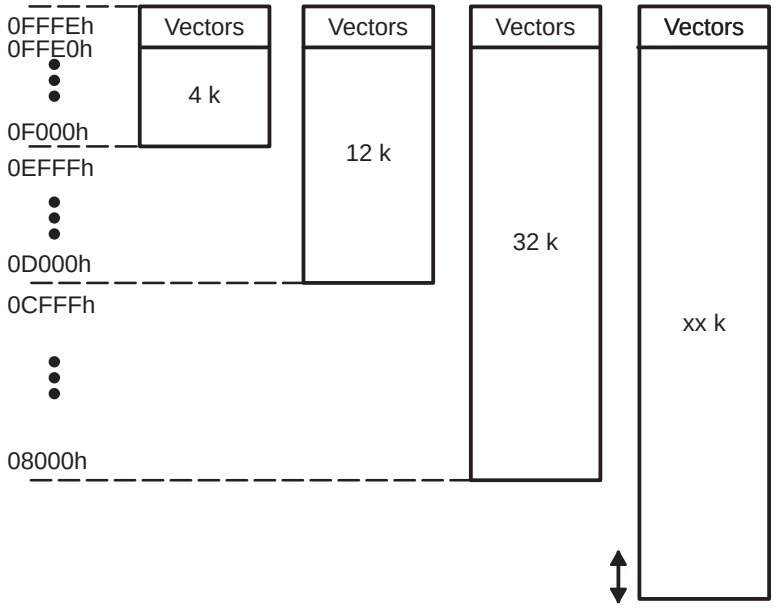
Figure 4–3. Bits, Bytes, and Words in a Byte-Organized Memory



4.3 Internal ROM Organization

Various sizes of ROM (OTP, masked-ROM, or EPROM) are available within the 64-kB address space, as shown in Figure 4–4. The common address space is shared with SFRs, peripheral module registers, data and code memory. The SFRs and peripheral modules are mapped into the address range, starting with 0 and ending with 01FFh. The remaining address space, 0200h to 0FFFFh, is shared by data and code memory. The start address for ROM depends on the amount of ROM present. The interrupt vector table is mapped into the the upper 16 words of ROM address space, with the highest priority interrupt vector at the highest ROM word address (0FFFEh). See the individual data sheets for specific memory maps.

Figure 4–4. ROM Organization



4.3.1 Processing of ROM Tables

The MSP430 architecture allows for the storage and usage of large tables in ROM without the need to copy the tables to RAM before using them. This ROM accessing of tables allows fast and clear programming in applications where data tables are necessary. This offers the flexible advantages listed below, and saves on ROM and RAM requirements. To access these tables, all word and byte instructions can be used.

- ROM storage of an output programmable logic array (OPLA) for display character conversion
- The use of as many OPLA terms as needed (no restriction on n terms)
- OTP version automatically includes OPLA programmability
- Computed table accessibility (for example, for a bar graph display)
- Table-supported program flows

4.3.2 Computed Branches and Calls

Computed branches and subroutine calls are possible using standard instructions. The call and branch instructions use the same addressing modes as the other instructions.

The addressing modes allow indirect-indirect addressing that is ideally suited for computed branches and calls. This programming technique permits a program structure that is different from conventional 8- and 16-bit microcontrollers. Most of the routines can be handled easily by using software status handling instead of flag-type program-flow control.

The computed branch and subroutine calls are valid throughout the entire ROM space.

4.4 RAM and Peripheral Organization

The entire RAM can be accessed with byte or word instructions using the appropriate instruction suffix. The peripheral modules, however, are located in two different address spaces and must be accessed with the appropriate instruction length.

- The SFRs are byte-oriented and mapped into the address space from 0h up to 0Fh.
- Peripheral modules that are byte-oriented are mapped into the address space from 010h up to 0FFh.
- Peripheral modules that are word-oriented are mapped into the address space from 100h up to 01FFh.

4.4.1 Random Access Memory

RAM can be used for both code and data memory. Code accesses are always performed on even byte addresses.

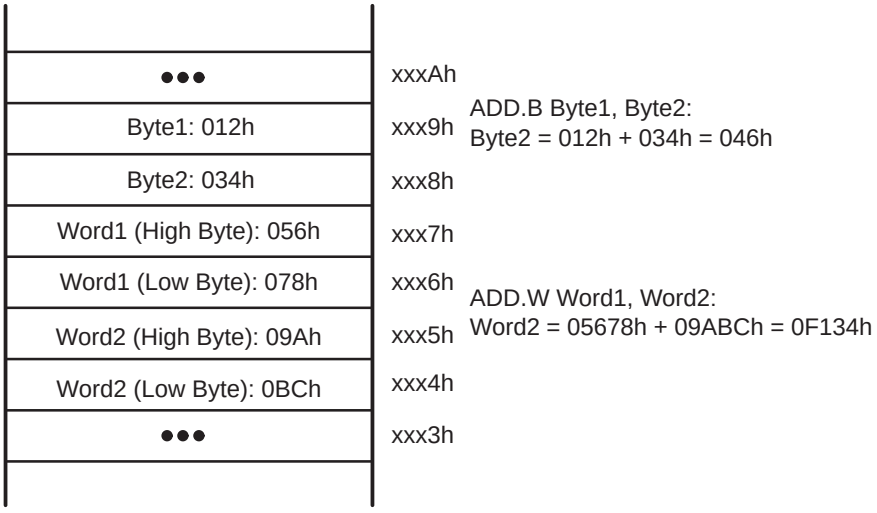
The instruction mnemonic suffix defines the data as being word or byte data.

Example:

```
ADD.B   &TCDATA,TCSUM_L                          ;Byte access
ADDC.B  TCSUM_H                                  ;Byte access
ADD     R5,SUM_A        =   ADD.W    R5,SUM_A    ;Word access
ADDC    SUM_B           =   ADDC.W   SUM_A       ;Word access
```

A word consists of two bytes: a high byte (bit 15 to bit 8), and a low byte (bit 7 to bit 0) as shown in Figure 4–5. It must always align to an even address.

Figure 4–5. Byte and Word Operation



All operations on the stack and PC are word operations and use even-aligned memory addresses.

In the following examples, word-to-word and byte-to-byte operations show the results of the operation and the status bit information.

Example Word-Word Operation	**Example Byte-Byte Operation**
R5 = 0F28Eh	R5 = 0223h
EDE .EQU 0212h	EDE .EQU 0202h
Mem(0F28Eh) = 0FFFEh	Mem(0223h) = 05Fh
Mem(0212h) = 00112h	Mem(0202h) = 043h
ADD @R5,&EDE	ADD.B @R5,&EDE
Mem(0212h) = 00110h	Mem(0202h) = 0A2h
C = 1, Z = 0, N = 0	C = 0, Z = 0, N = 1

Figure 4–6 shows the register-byte and byte-register operations.

Figure 4–6. Register-Byte/Byte-Register Operations

The following examples describe the register-byte and byte-register operations.

Example Register-Byte Operation	**Example Byte-Register Operation**
R5 = 0A28Fh	R5 = 01202h
R6 = 0203h	R6 = 0223h
Mem(0203h) = 012h	Mem(0223h) = 05Fh
ADD.B R5,0(R6)	ADD.B @R6,R5
08Fh + 012h 0A1h	05Fh + 002h ;Low byte of R5 00061h ;–>Store into R5 - ;High byte is 0
Mem (0203h) = 0A1h	R5 = 00061h
C = 0, Z = 0, N = 1	C = 0, Z = 0, N = 0
(Low byte of register) + (Addressed byte) –>(Addressed byte)	(Addressed byte) + (Low byte of register) –>(Low byte of register, zero to High byte)

Note: Word-Byte Operations

Word-byte or byte-word operations on memory data are not supported. Each register-byte or byte-register is performed as a byte operation.

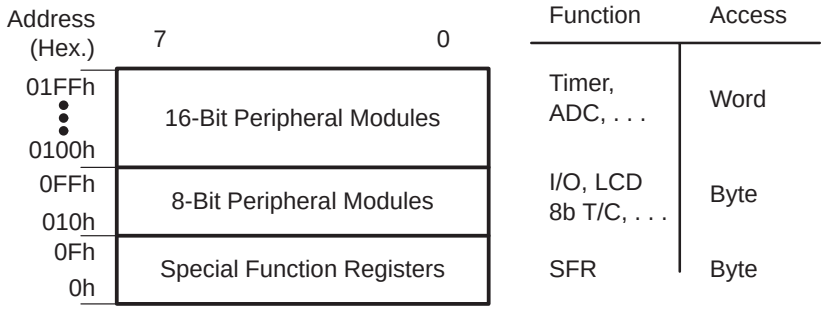
4.4.2 Peripheral Modules—Address Allocation

Some peripheral modules are accessible only with byte instructions, while others are accessible only with word instructions. The address space from 0100 to 01FFh is reserved for word modules, and the address space from 00h to 0FFh is reserved for byte modules.

Peripheral modules that are mapped into the word address space must be accessed using word instructions (for example, MOV R5,&WDTCTL). Peripheral modules that are mapped into the byte address space must be accessed with byte instructions (MOV.B #1,&TCCTL).

The addressing of both is through the absolute addressing mode or the 16-bit working registers using the indexed, indirect, or indirect autoincrement addressing mode. See Figure 4–7 for the RAM/peripheral organization.

Figure 4–7. Example of RAM/Peripheral Organization



4.4.2.1 Word Modules

Word modules are peripherals that are connected to the 16-bit MDB.

Word modules can be accessed with word or byte instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always ’0’.

The peripheral file address space is organized into sixteen frames with each frame representing eight words as described in Table 4–1.

Table 4–1. Peripheral File Address Map—Word Modules

Address	Description
1F0h – 1FFh	Reserved
1E0h – 1EFh	Reserved
1D0h – 1DFH	Reserved
1C0h – 1CFH	Reserved
1B0h – 1BFH	Reserved
1A0h – 1AFH	Reserved
190h – 19FH	Reserved
180h – 18FH	Reserved
170h – 17FH	Timer_A
160h – 16FH	Timer_A
150h – 15FH	Reserved
140h – 14FH	Reserved
130h – 13FH	Multiplier
120h – 12FH	Watchdog Timer
110h – 11FH	Analog-to-Digital Converter
100h – 10FH	Reserved

4.4.2.2 Byte Modules

Byte modules are peripherals that are connected to the reduced (eight LSB) MDB. Access to byte modules is always by byte instructions. The hardware in the peripheral byte modules takes the low byte (the LSBs) during a write operation.

Byte instructions operate on byte modules without any restrictions. Read access to peripheral byte modules using word instructions results in unpredictable data in the high byte. Word data is written into a byte module by writing the low byte to the appropriate peripheral register and ignoring the high byte.

The peripheral file address space is organized into sixteen frames as described in Table 4–2.

Table 4–2. Peripheral File Address Map—Byte Modules

Address	Description
00F0h – 00FFh	Reserved
00E0h – 00EFh	Reserved
00D0h – 00DFh	Reserved
00C0h – 00CFh	Reserved
00B0h – 00BFh	Reserved
00A0h – 00AFh	Reserved
0090h – 009Fh	Reserved
0080h – 008Fh	Reserved
0070h – 007Fh	USART
0060h – 006Fh	Reserved
0050h – 005Fh	System clock generator, EPROM and Crystal Buffer
0040h – 004Fh	Basic timer, 8-Bit Timer/Counter, Timer/Port
0030h – 003Fh	LCD
0020h – 002Fh	Digital I/O port P1 and P2 control
0010h – 001Fh	Digital I/O port P0, P3, and P4 control
0000h – 000Fh	Special function

4.4.3 Peripheral Modules—Special Function Registers (SFRs)

The system configuration and the individual reaction of the peripheral modules to the processor operation is configured in the SFRs as described in Table 4–3. The SFRs are located in the lower address range, and are organized by bytes. SFRs must be accessed using byte instructions only.

Table 4–3. Special Function Register Address Map

Address	Data Bus 7 0
000Fh	Not yet defined or implemented
000Eh	Not yet defined or implemented
000Dh	Not yet defined or implemented
000Ch	Not yet defined or implemented
000Bh	Not yet defined or implemented
000Ah	Not yet defined or implemented
0009h	Not yet defined or implemented
0008h	Not yet defined or implemented
0007h	Not yet defined or implemented
0006h	Not yet defined or implemented
0005h	Module enable 2; ME2.2
0004h	Module enable 1; ME1.1
0003h	Interrupt flag reg. 2; IFG2.x
0002h	Interrupt flag reg.1; IFG1.x
0001h	Interrupt enable 2; IE2.x
0000h	Interrupt enable 1; IE1.x

The system power consumption is influenced by the number of enabled modules and their functions. Disabling a module from the actual operation mode reduces power consumption while other parts of the controller remain fully active (unused pins must be tied appropriately or power consumption will increase; see *Basic Hints for Low Power Applications* in section 3.6.

Chapter 5

16-Bit CPU

The MSP430 von-Neumann architecture has RAM, ROM, and peripherals in one address space, both using a single address and data bus. This allows using the same instruction to access either RAM, ROM, or peripherals and also allows code execution from RAM.

Topic **Page**

5.1	CPU Registers	5-2
5.2	Addressing Modes	5-7
5.3	Instruction Set Overview	5-17
5.4	Instruction Map	5-23

5.1 CPU Registers

Sixteen 16-bit registers (R0, R1, and R4 to R15) are used for data and addresses and are implemented in the CPU. They can address up to 64 Kbytes (ROM, RAM, peripherals, etc.) without any segmentation. The complete CPU-register set is described in Table 5–1. Registers R0, R1, R2, and R3 have dedicated functions, which are described in detail later.

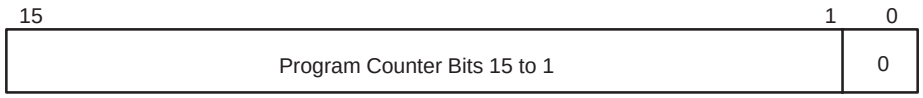
Table 5–1. Register by Functions

Program counter (PC)	R0
Stack pointer (SP)	R1
Status register (SR)	R2
Constant generator (CG1)	
Constant generator (CG2)	R3
Working register R4	R4
Working register R5	R5
: :	: :
Working register R13	R13
Working register R14	R14
Working register R15	R15

5.1.1 The Program Counter (PC)

The 16-bit program counter points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, or six), and the program counter is incremented accordingly. Instruction accesses are performed on word boundaries, and the program counter is aligned to even addresses. Figure 5–1 shows the program counter bits.

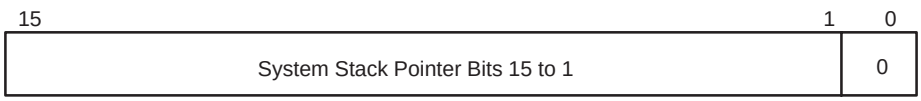
Figure 5–1. Program Counter



5.1.2 The System Stack Pointer (SP)

The system stack pointer must always be aligned to even addresses because the stack is accessed with word data during an interrupt request service. The system SP is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. The advantage of this scheme is that the item on the top of the stack is available. The SP can be used by the user software (PUSH and POP instructions), but the user should remember that the CPU also uses the SP. Figure 5–2 shows the system SP bits.

Figure 5–2. System Stack Pointer

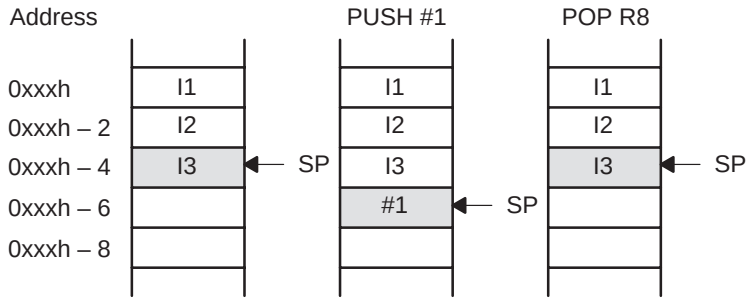


5.1.2.1 Examples for System SP Addressing (Refer to Figure 5–4)

```
MOV     SP,R4       ; SP –> R4
MOV     @SP,R5      ; Item I3 (TOS) –> R5
MOV     2(SP),R6    ; Item I2 –> R6
MOV     R7,0(SP)    ; Overwrite TOS with R7
MOV     R8,4(SP)    ; Modify item I1
PUSH    R12         ; Store R12 in address 0xxxh – 6; SP points to same address
POP     R12         ; Restore R12 from address 0xxxh – 6; SP points to
                      0xxxh – 4
MOV     @SP+,R5     ; Item I3 –> R5 (popped from stack); same as POP
                      instruction
```

Figure 5–3 shows stack usage.

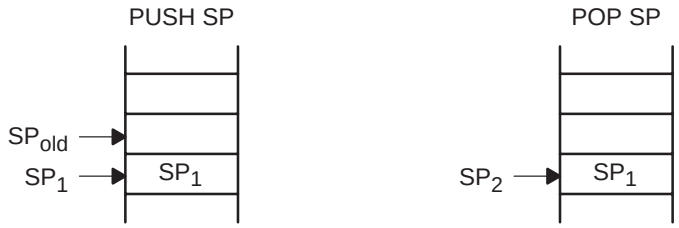
Figure 5–3. Stack Usage



5.1.2.2 Special Cases—PUSH SP and POP SP

The special cases of using the SP as an argument to the PUSH and POP instructions are described below.

Figure 5–4. PUSH SP and POP SP



The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction.

After the sequence

```
PUSH SP     ; SP1 is stack pointer after this instruction
  I
  I
POP SP      ; SP2 is stack pointer after this instruction
```

The stack pointer is two bytes lower than before this sequence.

5.1.3 The Status Register (SR)

The status register SR contains the following CPU status bits:

- ❑ V — Overflow bit
- ❑ SCG1 — System clock generator control bit 1
- ❑ SCG0 — System clock generator control bit 0
- ❑ OscOff — Crystal oscillator off bit
- ❑ CPUOff — CPU off bit
- ❑ GIE — General interrupt enable bit
- ❑ N — Negative bit
- ❑ Z — Zero bit
- ❑ C — Carry bit

Figure 5–5 shows the SR bits.

Figure 5–5. Status Register Bits

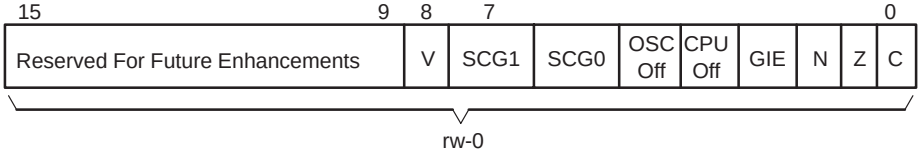


Table 5–2 describes the status register bits.

Table 5–2. Description of Status Register Bits

Bit	Description	
V	Overflow bit. Set if the result of an arithmetic operation overflows the signed-variable range. The bit is valid for both data formats, byte and word:	
	ADD(.B), ADDC(.B)	Set when: Positive + Positive = Negative Negative + Negative = Positive, otherwise reset
	SUB(.B), SUBC(.B), CMP(.B)	Set when: Positive – Negative = Negative Negative – Positive = Positive, otherwise reset
SCG1, SCG0	These bits control four activity states of the system-clock generator and therefore influence the operation of the processor system.	
OscOFF	If set, the crystal oscillator enters off mode: all activities cease; however, the RAM contents, the port, and the registers are maintained. Wake up is possible only through enabled external interrupts when the GIE bit is set and from the NMI.	
CPU Off	If set, the CPU enters off mode: program execution stops. However, the RAM, the port registers, and especially the enabled peripherals (for example, basic timer, UART, etc.) stay active. Wake up is possible through all enabled interrupts.	
GIE	If set, all enabled maskable interrupts are handled. If reset, all maskable interrupts are disabled. The GIE bit is cleared by interrupts and restored by the RETI instruction as well as by other appropriate instructions.	
N	Set if the result of an operation is negative.	
	Word operation:	Negative bit is set to the value of bit 15 of the result
	Byte operation:	Negative bit is set to the value of bit 7 of the result
Z	Set if the result of byte or word operation is 0; cleared if the result is not 0.	
C	Set if the result of an operation produced a carry; cleared if no carry occurred. Some instructions modify the carry bit using the inverted zero bits.	

Note: Status Register Bits V, N, Z and C

The status register bits V, N, Z, and C are modified only with the appropriate instruction. For additional information, see the detailed description of the instruction set in Appendix B.

5.1.4 The Constant Generator Registers CG1 and CG2

Commonly-used constants are generated with the constant generator registers **R2 and R3**, without requiring an additional 16-bit word of program code. The constant used for immediate values is defined by the addressing mode bits (As) as described in Table 5–3. See Section 5.3 for a description of the addressing mode bits (As).

Table 5–3. Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	– – – – –	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	–1, word processing

The major advantages of this type of constant generation are:

- No special instructions required
- Reduced code memory requirements: no additional word for the six most used constants
- Reduced instruction cycle time: no code memory access to retrieve the constant

The assembler uses the constant generator automatically if one of the six constants is used as a source operand in the immediate addressing mode. The status register SR/R2, used as a source or destination register, can be used in the register mode only. The remaining combinations of addressing-mode bits are used to support absolute-address modes and bit processing without any additional code. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act like source-only registers.

The RISC instruction set of the MSP430 only has 27 instructions. However, the constant generator allows the MSP430 assembler to support 24 additional, emulated instructions. For example, the single-operand instruction:

```
CLR     dst
```

is emulated by the double-operand instruction with the same length:

```
MOV     R3,dst
or the equivalent
MOV     #0,dst
```

where #0 is replaced by the assembler, and R3 is used with As = 00, which results in:

- One word instruction
- No additional control operation or hardware within the CPU
- Register-addressing mode for source: no extra-fetch cycle for constants (#0)

5.2 Addressing Modes

All seven addressing modes for the source operand and all four addressing modes for the destination operand can address the complete address space. The bit numbers in Table 5–4 describe the contents of the As and Ad mode bits. See Section 5.3 for a description of the source address As and the destination address Ad bits.

Table 5–4. Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand X is stored in the next word
01/1	Symbolic mode	ADDR	(PC + X) points to the operand X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

The seven addressing modes are explained in detail in the following sections. Most of the examples show the same addressing mode for the source and destination, but any valid combination of source and destination addressing modes is possible in an instruction.

5.2.1 Register Mode

The register mode is described in Table 5–5.

Table 5–5. Register Mode Description

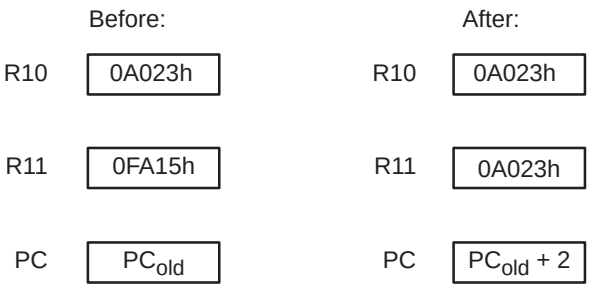
Assembler Code	Content of ROM
MOV R10,R11	MOV R10,R11

Length: One or two words

Operation: Move the content of R10 to R11. R10 is not affected.

Comment: Valid for source and destination

Example: MOV R10,R11



Note: Data in Registers

The data in the register can be accessed using word or byte instructions. If byte instructions are used, the high byte is always 0 in the result. The status bits are handled according to the result of the byte instruction.

5.2.2 Indexed Mode

The indexed mode is described in Table 5–6.

Table 5–6. Indexed Mode Description

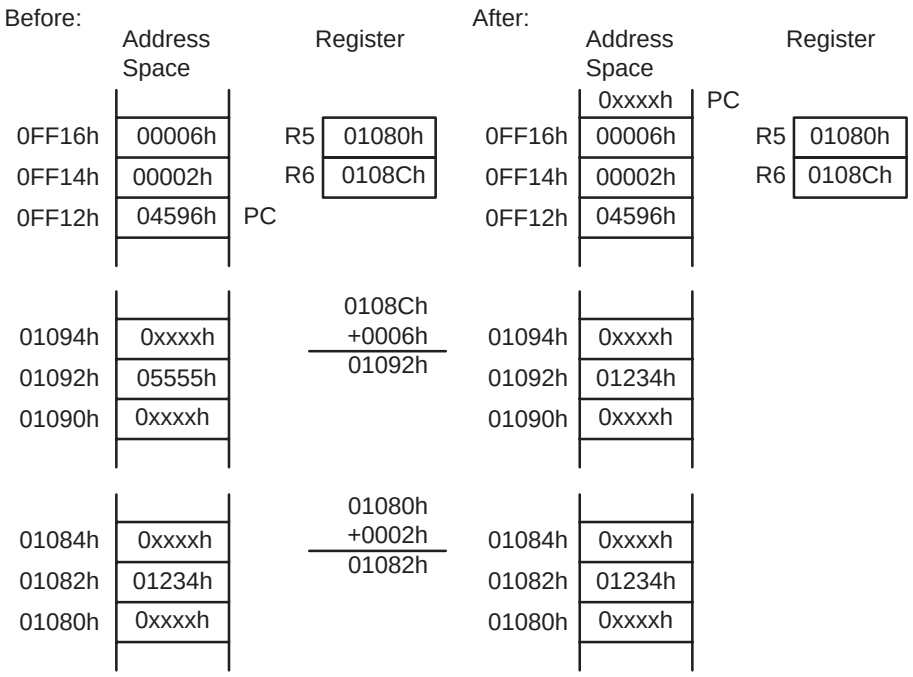
Assembler Code	Content of ROM
MOV 2(R5),6(R6)	MOV X(R5),Y(R6) X = 2 Y = 6

Length: Two or three words

Operation: Move the contents of the source address (contents of R5 + 2) to the destination address (contents of R6 + 6). The source and destination registers (R5 and R6) are not affected. In indexed mode, the program counter is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV 2(R5),6(R6):



5.2.3 Symbolic Mode

The symbolic mode is described in Table 5–7.

Table 5–7. Symbolic Mode Description

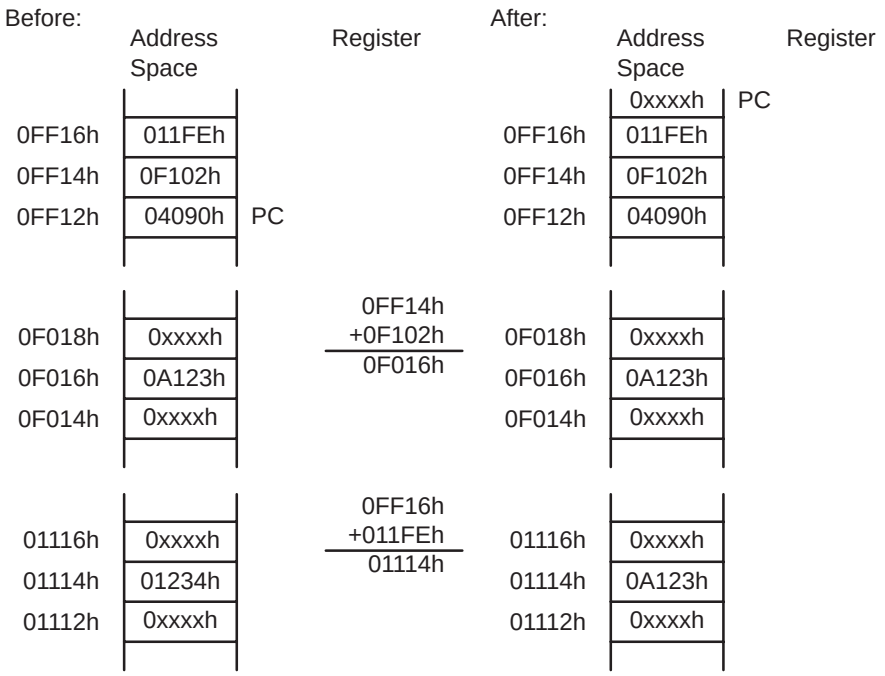
Assembler Code	Content of ROM
MOV EDE,TONI	MOV X(PC),Y(PC) X = EDE – PC Y = TONI – PC

Length: Two or three words

Operation: Move the contents of the source address EDE (contents of PC + X) to the destination address TONI (contents of PC + Y). The words after the instruction contain the differences between the PC and the source or destination addresses. The assembler computes and inserts offsets X and Y automatically. With symbolic mode, the program counter (PC) is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV EDE,TONI ;Source address EDE = 0F016h,
;dest. address TONI=01114h



5.2.4 Absolute Mode

The absolute mode is described in Table 5–8.

Table 5–8. Absolute Mode Description

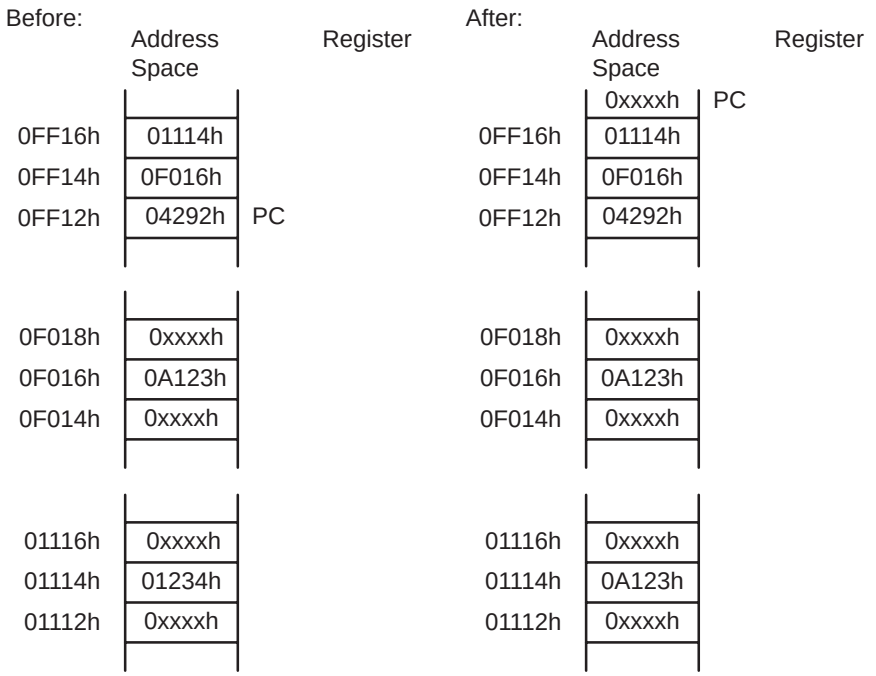
Assembler Code	Content of ROM
MOV &EDE,&TONI	MOV X(0),Y(0) X = EDE Y = TONI

Length: Two or three words

Operation: Move the contents of the source address EDE to the destination address TONI. The words after the instruction contain the absolute address of the source and destination addresses. With absolute mode, the PC is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example:
```
MOV  &EDE,&TONI     ;Source address EDE = 0F016h,
                    ;dest. address TONI=01114h
```



This address mode is mainly for hardware peripheral modules that are located at an absolute, fixed address. These are addressed with absolute mode to ensure software transportability (for example, position-independent code).

5.2.5 Indirect Mode

The indirect mode is described in table 5–9.

Table 5–9. Indirect Mode Description

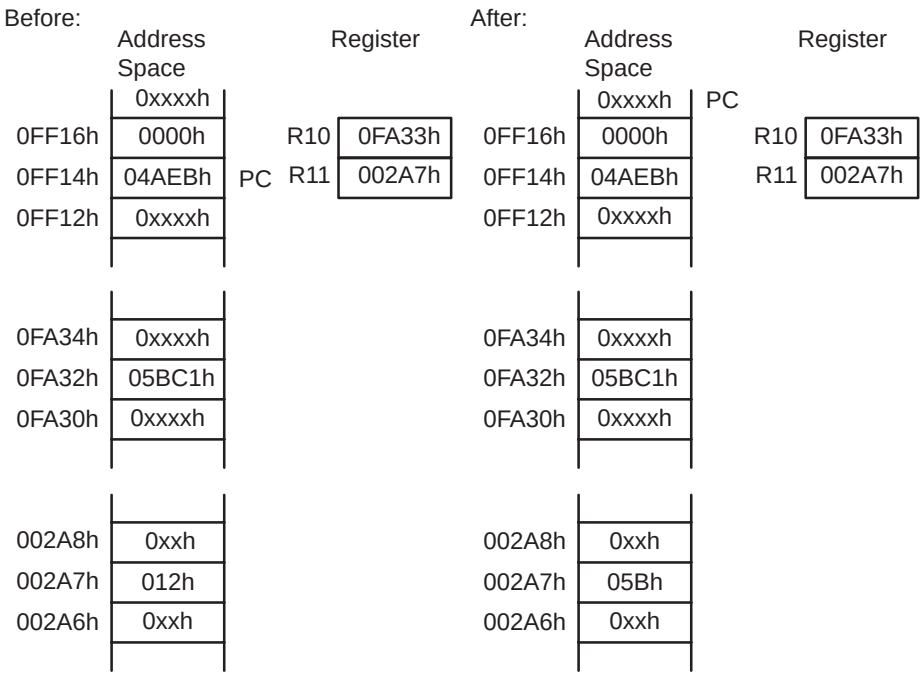
Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.

Comment: Valid only for source operand. The substitute for destination operand is 0(Rd).

Example: MOV.B @R10,0(R11)



5.2.6 Indirect Autoincrement Mode

The indirect autoincrement mode is described in Table 5–10.

Table 5–10. Indirect Autoincrement Mode Description

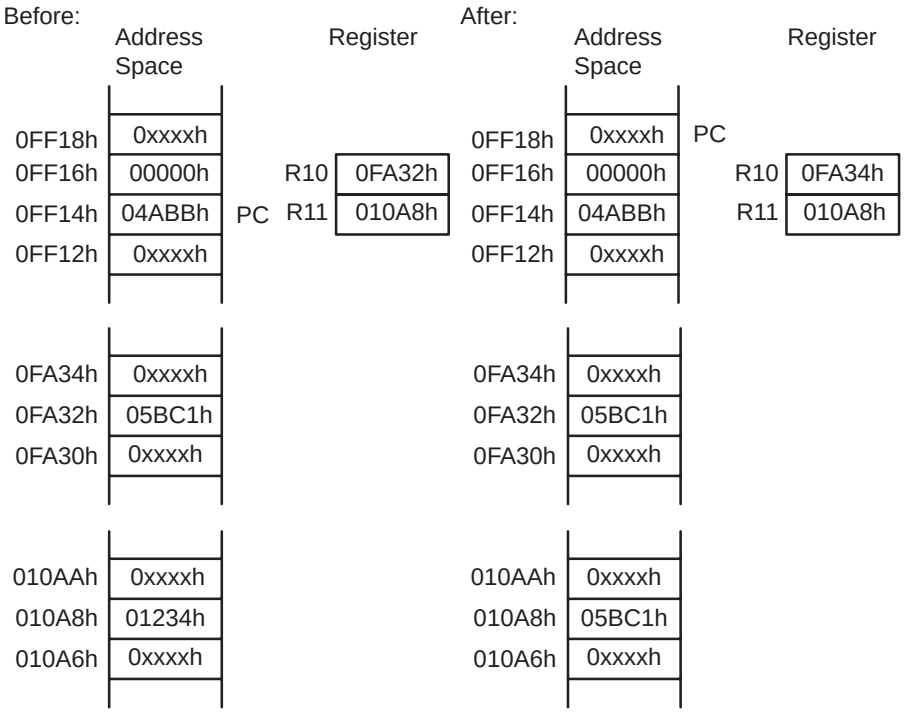
Assembler Code	Content of ROM
MOV @R10+,0(R11)	MOV @R10+,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). Register R10 is incremented by 1 for a byte operation, or 2 for a word operation after the fetch; it points to the next address without any overhead. This is useful for table processing.

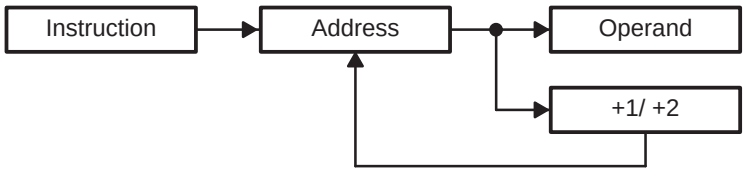
Comment: Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

Example: MOV @R10+,0(R11)



The autoincrementing of the register contents occurs after the operand is fetched. This is shown in Figure 5–6.

Figure 5–6. Operand Fetch Operation



5.2.7 Immediate Mode

The immediate mode is described in Table 5–11.

Table 5–11. Immediate Mode Description

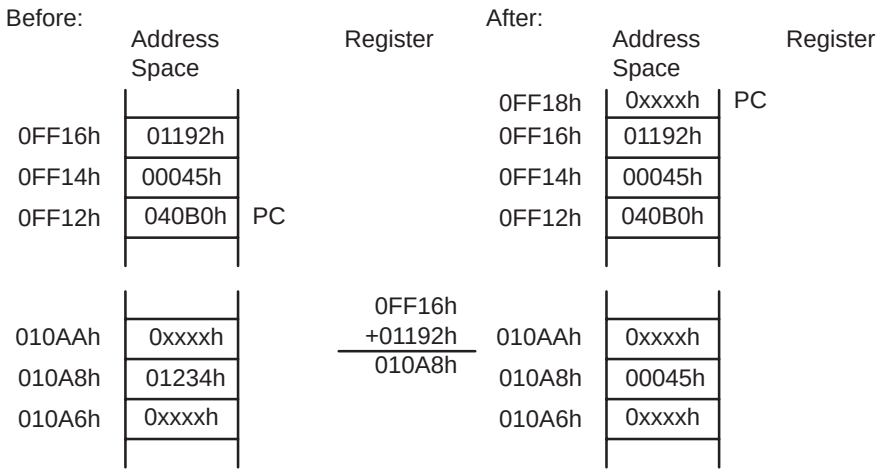
Assembler Code	Content of ROM
MOV #45,TONI	MOV @PC+,X(PC) 45 X = TONI – PC

Length: Two or three words
It is one word less if a constant of CG1 or CG2 can be used.

Operation: Move the immediate constant 45, which is contained in the word following the instruction, to destination address TONI. When fetching the source, the program counter points to the word following the instruction and moves the contents to the destination.

Comment: Valid only for a source operand.

Example: MOV #45,TONI



5.2.8 Clock Cycles, Length of Instruction

The operating speed of the CPU depends on the instruction format and addressing modes. The number of clock cycles refers to the MCLK.

5.2.8.1 Format-I Instructions

Table 5–12 describes the CPU format-I instructions and addressing modes.

Table 5–12. Instruction Format I and Addressing Modes

Address Mode		No. of Cycles	Length of Instruction	Example	
As	Ad				
00, Rn	0, Rm	1	1	MOV	R5,R8
	0, PC	2	1	BR	R9
00, Rn	1, x(Rm)	4	2	ADD	R5,3(R6)
	1, EDE		2	XOR	R8,EDE
	1, &EDE		2	MOR	R5,&EDE
01, x(Rn)	0, Rm	3	2	MOV	2(R5),R7
01, EDE			2	AND	EDE,R6
01, &EDE			2	MOV	&EDE,R8
01, x(Rn)	1, x(Rm)	6	3	ADD	3(R4),6(R9)
01, EDE	1, TONI		3	CMP	EDE,TONI
01, &EDE	1, &TONI		3	MOV	2(R5),&TONI
			3	ADD	EDE,&TONI
10, @Rn	0, Rm	2	1	AND	@R4,R5
10, @Rn	1, x(Rm)	5	2	XOR	@R5,8(R6)
	1, EDE		2	MOV	@R5,EDE
	1, &EDE		2	XOR	@R5,&EDE
11, @Rn+	0, Rm	2	1	ADD	@R5+,R6
	0, PC	3	1	BR	@R9+
11, #N	0, Rm	2	2	MOV	#20,R9
	0, PC	3	2	BR	#2AEh
11, @Rn+	1, x(Rm)	5	2	MOV	@R9+,2(R4)
11, #N	1, EDE		3	ADD	#33,EDE
11, @Rn+	1, &EDE		2	MOV	@R9+,&EDE
11, #N			3	ADD	#33,&EDE

5.2.8.2 Format-II Instructions

Table 5–13 describes the CPU format II instructions and addressing modes.

Table 5–13. Instruction Format-II and Addressing Modes

Address Mode $A_{(s/d)}$	No. of Cycles: RRA RRC SWPB SXT	No. of Cycles: PUSH/ CALL	Length of Instruction (words)	Example
00, Rn	1	3/4	1	SWPB R5
01, X(Rn) 01, EDE 01, &EDE	4 4	5 5	2 2	CALL 2(R7) PUSH EDE SXT &EDE
10, @Rn	3	4	1	RRC @R9
11, @Rn+ (see Note) 11, #N	3	4/5	1 2	SWPB @R10+ CALL #81H

Note: Instruction Format II Immediate Mode

Do not use instructions RRA, RRC, SWPB, and SXT with the immediate mode in the destination field. Use of these in the immediate mode will result in an unpredictable program operation.

5.2.8.3 Format-III Instructions

Format-III instructions are described as follows:

Jxx—all instructions need the same number of cycles, independent of whether a jump is taken or not.

Clock cycle: Two cycles
Length of instruction: One word

5.2.8.4 Miscellaneous-Format Instructions

Table 5–14 describes miscellaneous-format instructions.

Table 5–14. Miscellaneous Instructions or Operations

Activity	Clock Cycle
RETI	5 cycles 1 word†
Interrupt	6 cycles
WDT reset	4 cycles
Reset ($\overline{RST}$/NMI)	4 cycles

† Length of instruction

5.3 Instruction Set Overview

This section gives a short overview of the instruction set. The addressing modes are described in Section 5.2.

Instructions are either single or dual operand or jump.

The source and destination parts of an instruction are defined by the following fields:

src	The source operand defined by As and S-reg
dst	The destination operand defined by Ad and D-reg
As	The addressing bits responsible for the addressing mode used for the source (src)
S-reg	The working register used for the source (src)
Ad	The addressing bits responsible for the addressing mode used for the destination (dst)
D-reg	The working register used for the destination (dst)
B/W	Byte or word operation: 0: word operation 1: byte operation

Note: Destination Address

Destination addresses are valid anywhere in the memory map. However, when using an instruction that modifies the contents of the destination, the user must ensure the destination address is writeable. For example, a masked-ROM location would be a valid destination address, but the contents are not modifiable, so the results of the instruction would be lost.

5.3.1 Double-Operand Instructions

Figure 5–7 illustrates the double-operand instruction format.

Figure 5–7. Double Operand Instruction Format

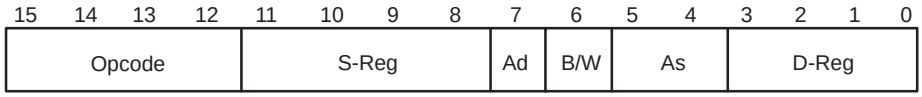


Table 5–15 describes the effects of an instruction on double operand instruction status bits.

Table 5–15. Double Operand Instruction Format Results

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV	src,dst	src –> dst	–	–	–	–
ADD	src,dst	src + dst –> dst	*	*	*	*
ADDC	src,dst	src + dst + C –> dst	*	*	*	*
SUB	src,dst	dst + .not.src + 1 –> dst	*	*	*	*
SUBC	src,dst	dst + .not.src + C –> dst	*	*	*	*
CMP	src,dst	dst – src	*	*	*	*
DADD	src,dst	src + dst + C –> dst (dec)	*	*	*	*
AND	src,dst	src .and. dst –> dst	0	*	*	*
BIT	src,dst	src .and. dst	0	*	*	*
BIC	src,dst	.not.src .and. dst –> dst	–	–	–	–
BIS	src,dst	src .or. dst –> dst	–	–	–	–
XOR	src,dst	src .xor. dst –> dst	*	*	*	*

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

Note: Instructions CMP and SUB

The instructions CMP and SUB are identical except for the storage of the result. The same is true for the BIT and AND instructions.

5.3.2 Single-Operand Instructions

Figure 5–8 illustrates the single-operand instruction format.

Figure 5–8. Single Operand Instruction Format

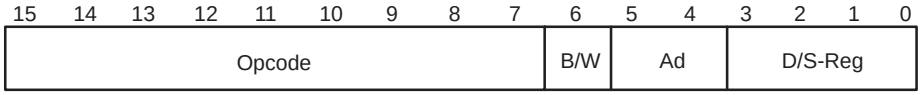


Table 5–16 describes the effects of an instruction on the single operand instruction status bits.

Table 5–16. Single Operand Instruction Format Results

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC	dst	C –> MSB –>.....LSB –> C	*	*	*	*
RRA	dst	MSB –> MSB –>....LSB –> C	0	*	*	*
PUSH	src	SP – 2 –> SP, src –> @ SP	–	–	–	–
SWPB	dst	swap bytes	–	–	–	–
CALL	dst	SP – 2 –> SP	–	–	–	–
		PC+2 –> stack, dst –> PC				
RETI		TOS –> SR, SP <– SP + 2	X	X	X	X
		TOS –> PC, SP <– SP + 2				
SXT	dst	Bit 7 –> Bit 8.....Bit 15	0	*	*	*

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

All addressing modes are possible for the CALL instruction. If the symbolic mode (ADDRESS), the immediate mode (#N), the absolute mode (&EDE) or the indexed mode X (RN) is used, the word that follows contains the address information.

5.3.3 Conditional Jumps

Conditional jumps support program branching relative to the program counter. The possible jump range is from –511 to +512 words relative to the program counter state of the jump instruction. The 10-bit program-counter offset value is treated as a signed 10-bit value that is doubled and added to the program counter. None of the jump instructions affect the status bits.

The instruction code fetch and the program counter increment technique end with the formula:

$$PC_{new} = PC_{old} + 2 + PC_{offset} \times 2$$

Figure 5–9 shows the conditional-jump instruction format.

Figure 5–9. Conditional-Jump Instruction Format

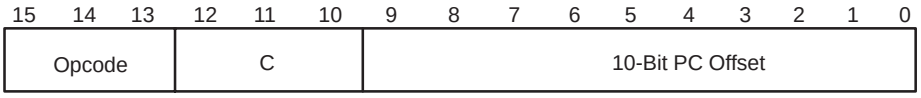


Table 5–17 describes these conditional-jump instructions.

Table 5–17. ConditIonal-Jump Instructions

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

5.3.4 Short Form of Emulated Instructions

The basic instruction set, together with the register implementations of the program counter, stack pointer, status register, and constant generator, form the emulated instruction set; these make up the popular instruction set. The status bits are set according to the result of the execution of the basic instruction that replaces the emulated instruction.

Table 5–18 describes these instructions.

Table 5–18. Emulated Instructions

Mnemonic		Description	Status Bits				Emulation	
			V	N	Z	C		
ArIthmetic Instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical Instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#–1,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data Instructions (common use)								
CLR[.W]		Clear destination	–	–	–	–	MOV	#0,dst
CLR.B		Clear destination	–	–	–	–	MOV.B	#0,dst
CLRC		Clear carry bit	–	–	–	0	BIC	#1,SR
CLRN		Clear negative bit	–	0	–	–	BIC	#4,SR
CLRZ		Clear zero bit	–	–	0	–	BIC	#2,SR
POP	dst	Item from stack	–	–	–	–	MOV	@SP+,dst
SETC		Set carry bit	–	–	–	1	BIS	#1,SR
SETN		Set negative bit	–	1	–	–	BIS	#4,SR
SETZ		Set zero bit	–	–	1	–	BIS	#2,SR

Table 5–18. Emulated Instructions (Continued)

Mnemonic		Description	Status Bits				Emulation	
			V	N	Z	C		
Data Instructions (common use) (continued)								
TST[.W]	dst	Test destination	0	*	*	*	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	*	CMP.B	#0,dst
Program Flow Instructions								
BR	dst	Branch to . . .	–	–	–	–	MOV	dst,PC
DINT		Disable interrupt	–	–	–	–	BIC	#8,SR
EINT		Enable interrupt	–	–	–	–	BIS	#8,SR
NOP		No operation	–	–	–	–	MOV	#0h,#0h
RET		Return from subroutine	–	–	–	–	MOV	@SP+,PC

5.3.5 Miscellaneous

Instructions without operands, such as CPUOff, are not provided. Their functions are switched on or off by setting or clearing the function bits in the status register or the appropriate I/O register. Other functions are emulated using dual operand instructions.

Some examples are as follows:

```
BIS     #28h,SR         ; Enter OscOff mode
                        ; + Enable general interrupt (GIE)
BIS     #18h,SR         ; Enter CPUOff mode
                        ; + Enable general interrupt (GIE)
BIC     #SVCC,&ACTL     ; Switch SVCC off
```

5.4 Instruction Map

The instruction map in Figure 5–10 is an example of how to encode instructions. There is room for more instructions, if needed.

Figure 5–10. Core Instruction Map

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0x																
04x																
08x																
0Cx																
10x	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14x																
18x																
1Cx																
20x	JNE/JNZ															
24x	JEQ/JZ															
28x	JNC															
2Cx	JC															
30x	JN															
34x	JGE															
38x	JL															
3Cx	JMP															
40x–4Cx	MOV, MOV.B															
50x–5Cx	ADD, ADD.B															
60x–6Cx	ADDC, ADDC.B															
70x–7Cx	SUBC, SUBC.B															
80x–8Cx	SUB, SUB.B															
90x–9Cx	CMP, CMP.B															
A0x–ACx	DADD, DADD.B															
B0x–BCx	BIT, BIT.B															
C0x–CCx	BIC, BIC.B															
D0x–DCx	BIS, BIS.B															
E0x–ECx	XOR, XOR.B															
F0x–FCx	AND, AND.B															

Chapter 6

Hardware Multiplier

The hardware multiplier is a 16-bit peripheral module. It is not integrated into the CPU. Therefore, it requires no special instructions and operates independent of the CPU. To use the hardware multiplier, the operands are loaded into registers and the results are available the next instruction—no extra cycles are required for a multiplication.

Topic **Page**

6.1	**Hardware Multiplier Module Support**	**6-2**
6.2	**Hardware Multiplier Operation**	**6-3**
6.3	**Hardware Multiplier Registers**	**6-9**
6.4	**Hardware Multiplier Special Function Bits**	**6-10**
6.5	**Hardware Multiplier Software Restrictions**	**6-10**

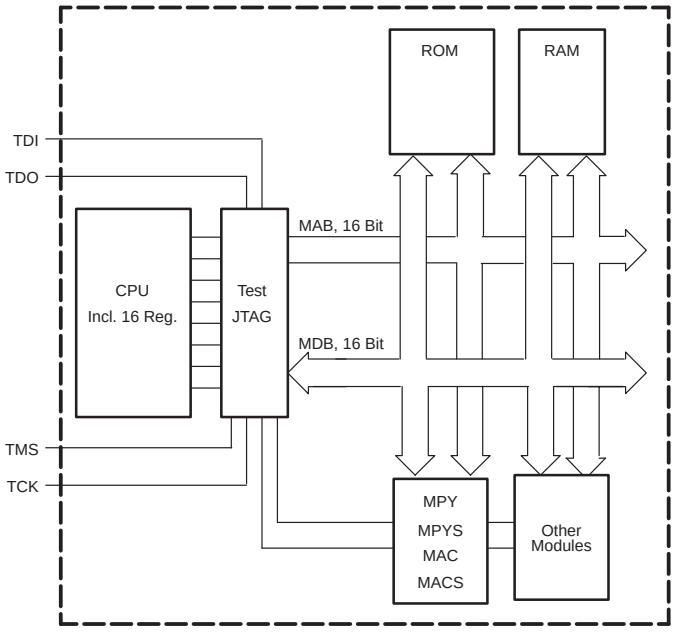
6.1 Hardware Multiplier Module Support

The hardware multiplier module expands the capabilities of the MSP430 family without changing the basic architecture. Multiplication is possible for:

- 16×16 bits
- 16×8 bits
- 8×16 bits
- 8×8 bits

The hardware multiplier module supports four types of multiplication: unsigned multiplication (MPY), signed multiplication (MPYS), unsigned multiplication with accumulation (MAC), and signed multiplication with accumulation (MACS). Figure 6–1 shows how the hardware multiplier module interfaces with the bus system to support multiplication operations.

Figure 6–1. Connection of the Hardware Multiplier Module to the Bus System



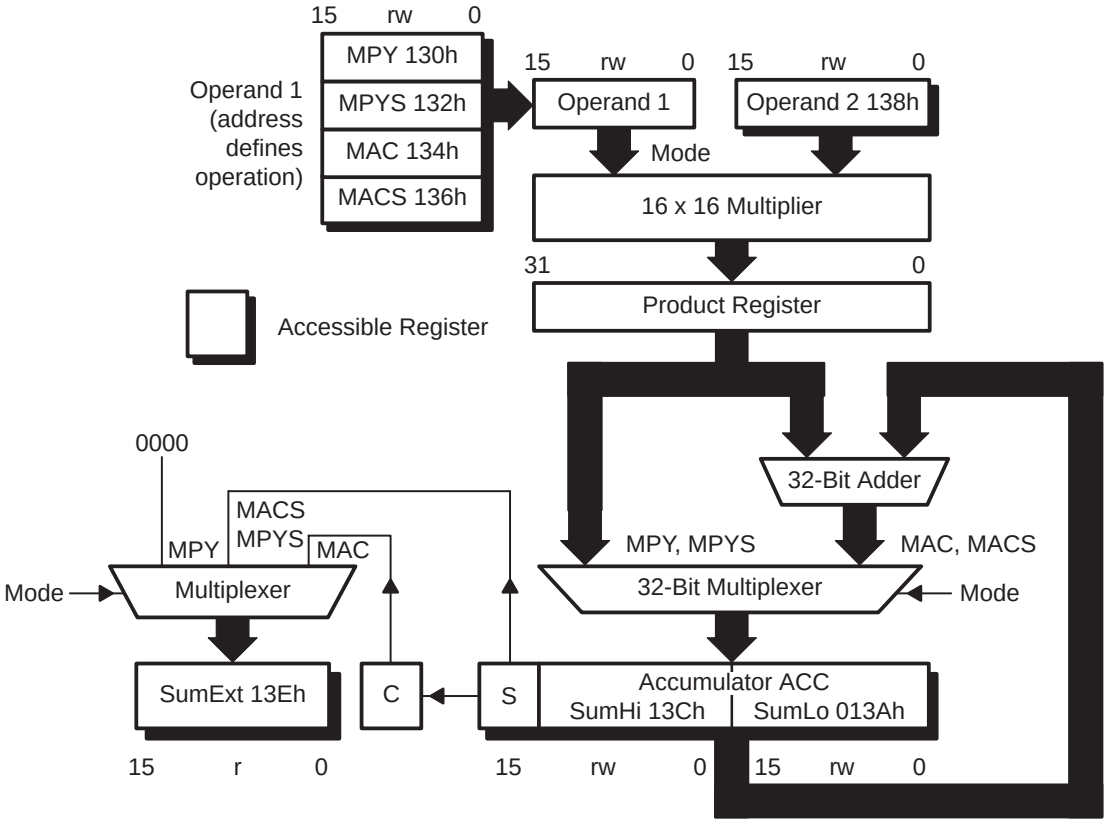
6.2 Hardware Multiplier Operation

The hardware multiplier has two 16-bit registers for both operands and three registers to store the results of the multiplication. The multiplication is executed correctly when the first operand is written to the operand register OP1 prior to writing the second operand to OP2. Writing the first operand to the applicable register selects the type of multiplication. Writing the second operand to OP2 starts the multiplication. Multiplication is completed before the result registers are accessed using the indexed address mode for the source operand. When indirect or indirect autoincrement address modes are used, another instruction is needed between the writing of the second operand and accessing the result registers. Both operands, OP1 and OP2, utilize all seven address mode capabilities.

No instruction is necessary for the multiplication; as a result, the real-time operation does not require additional clock cycles and the interrupt latency is unchanged.

The multiplier architecture is illustrated in Figure 6–2.

Figure 6–2. Block Diagram of the MSP430 16×16-Bit Hardware Multiplier



The sum extension register contents differ, depending on the operation and on the results of the operation.

Table 6–1. Sum Extension Register Contents

Register	MPY	MPYS		MAC		MACS, see Notes	
Operand1	x	+ –	+ +	(OP1×OP2 + ACC) ≤ 0FFFFFFFFh	(OP1×OP2 + ACC) > 0FFFFFFFFh	(OP1×OP2 + ACC) > 07FFFFFFFh	(OP1×OP2 + ACC) ≤ 07FFFFFFFh
Operand2	x	+ –	– –				
SumExt	0000h	0000h	0FFFFh	0000h	0001h	0FFFFh	0000h

Note: The following two overflow conditions may occur when using the MACS function and should be handled by software or avoided.

1) The result of a MACS operation is positive and larger than 07FFF FFFFh. In this case, the SumExt register contains 0FFFFh and the ACC register contains a negative number (8000 0000h .... 0FFFF FFFFh).
2) The result of a MACS operation is negative and less than or equal to 07FFF FFFFh. In this case, the SumExt register contains 0000h and the ACC register contains a positive number (0000 0000h ... 07FFF FFFFh).

6.2.1 Multiply Unsigned, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit

The following multiplication operation shows 32 bytes of program code and 32 execution cycles (16×16 bit multiplication).

```
*********************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*      HARDWARE MULTIPLIER MODULE                        *
*      USE CONSTANT OPERAND1 AND OPERAND2 TO IDENTIFY    *
*      BYTE DATA                                         *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY     .EQU   0130H
MPYS    .EQU   0132H
MAC     .EQU   0134H
MACS    .EQU   0136H
OP2     .EQU   0138H
RESLO   .EQU   013AH
RESHI   .EQU   013CH
SUMEXT  .EQU   013EH
        .BSS   OPER1,2,200H
        .BSS   OPER2,2
        .BSS   RAM,8

        .IF OPERAND1=8
        MOV.B  &OPER1,&MPY ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ELSE
        MOV    &OPER1,&MPY ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ENDIF

        .IF OPERAND1=8
        MOV.B  &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ELSE
        MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ENDIF

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE         *
*      MULTIPLICATION TO THE RAM DATA, 64BITS            *
*********************************************************

        ADD    &RESLO,&RAM   ; ADD LOW RESULT TO RAM
        ADDC   &RESHI,&RAM+2 ; ADD HIGH RESULT TO RAM+2
        ADC    &RAM+4        ; ADD CARRY TO EXTENSION WORD
        ADC    &RAM+6        ; IF 64 BIT LENGTH IS USED
```

6.2.2 Multiply Signed, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit

The following multiplication operation shows 36 bytes of program code and 36 execution cycles (16×16 bit multiplication).

```
*********************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*      HARDWARE MULTIPLIER MODULE                        *
*      IF ONE OF THE OPERANDS IS 8 BIT, SIGN EXTENSION   *
*      is NEEDED. USE CONSTANT OPERAND1 AND OPERAND2 TO  *
*      IDENTIFY BYTE DATA                                *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY    .EQU  0130H
MPYS   .EQU  0132H
MAC    .EQU  0134H
MACS   .EQU  0136H
OP2    .EQU  0138H
RESLO  .EQU  013AH
RESHI  .EQU  013CH
SUMEXT .EQU  013EH
       .BSS  OPER1,2,200H
       .BSS  OPER2,2
       .BSS  RAM,8

       .IF OPERAND1=0
       MOV   &OPER1,&MPYS ; LOAD 1ST (WORD) OPERAND,
                          ; DEFINES ADD. SIGNED MULTIPLY

       .ELSE
       MOV.B &OPER1,&MPYS ; LOAD 1ST (BYTE) OPERAND,
                          ; DEFINES ADD. SIGNED MULTIPLY
       SXT   &MPYS        ; EXPAND BYTE TO SIGNED WORD DATA
       .ENDIF
       .IF OPERAND2=0
       MOV   &OPER2,&OP2  ; LOAD 2ND (WORD) OPERAND AND
                          ; START SIGNED MULTIPLICATION

       .ELSE
       MOV.B &OPER2,&OP2  ; LOAD 2ND (BYTE) OPERAND,
       SXT   &OP2         ; RE-LOAD 2ND OPERAND AND START
                          ; SIGNED 'FINAL' MULTIPLICATION

       .ENDIF

*********************************************************
*       EXAMPLE TO ADD THE RESULT OF THE HARDWARE       *
*       MULTIPLICATION TO THE RAM DATA, 64 BITS         *
*********************************************************

      ADD  &RESLO,&RAM    ; ADD LOW RESULT TO RAM
      ADDC &RESHI,&RAM+2  ; ADD HIGH RESULT TO RAM+2
      ADDC &SUMEXT,&RAM+4 ; ADD SIGN WORD TO EXTENSION WORD
      ADDC &SUMEXT,&RAM+6 ; IF 64 BIT LENGTH IS USED
```

6.2.3 Multiply Unsigned and Accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

The following multiplication operation shows 32 bytes of program code and 32 execution cycles (16X16-bit multiplication).

```
*********************************************************
*     TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*     HARDWARE MULTIPLIER MODULE                        *
*     THE RESULT OF THE MULTIPLICATION IS ADDED TO THE  *
*     CONTENT OF BOTH RESULT REGISTERS, RESLO AND RESHI *
*     USE CONSTANT OPERAND1 AND OPERAND2 TO IDENTIFY    *
*     BYTE DATA                                         *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY     .EQU   0130H
MPYS    .EQU   0132H
MAC     .EQU   0134H
MACS    .EQU   0136H
OP2     .EQU   0138H
RESLO   .EQU   013AH
RESHI   .EQU   013CH
SUMEXT  .EQU   013EH
        .BSS   OPER1,2,200H
        .BSS   OPER2,2
        .BSS   RAM,8

        .IF OPERAND1=8
        MOV.B  &OPER1,&MAC ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ELSE
        MOV    &OPER1,&MAC ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ENDIF

        .IF OPERAND1=8
        MOV.B  &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ELSE
        MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ENDIF

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE        *
*      MULTIPLICATION TO THE RAM DATA, 64BITS           *
*      THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO*
*      AND RESHI REGISTERS. THE UPPER TWO WORDS IN THE  *
*      EXAMPLE ARE FURTHER LOCATED IN THEIR RAM LOCATION*
*********************************************************

       ADDC  &SUMEXT,&RAM+4  ; ADD SUMEXTENSTION TO RAM+4
       ADC   &RAM+6          ; IF 64 BIT LENGTH IS USED
```

6.2.4 Multiply Signed and Accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
*********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE      *
*      MULTIPLIER MODULE                                            *
*      USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA    *
*********************************************************************
OPERAND1    .EQU    0       ; 0:  OPERAND1 IS WORD (16BIT)
                            ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2    .EQU    0       ; 0:  OPERAND2 IS WORD (16BIT)
                            ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY         .EQU    0130H
MPYS        .EQU    0132H
MAC         .EQU    0134H
MACS        .EQU    0136H
OP2         .EQU    0138H
RESLO       .EQU    013AH
RESHI       .EQU    013CH
SUMEXT      .EQU    013EH
MAXMACS     .EQU    32H     ;NUMBER OF MACS FUNCTIONS WHICH COULD
                            ;BE EXECUTED TILL AN OVERFLOW OR UNDERFLOW
                            ;COULD OCCUR THE FIRST TIME
            .BSS    OPER1,2,200H
            .BSS    OPER2,2
            .BSS    RAM,8
            .BSS    MCOUNT,2
            .IF OPERAND1=8
            MOV.B   &OPER1,&MACS    ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
            SXT     &MACS           ; EXPAND BYTE TO SIGNED WORD DATA
            .ELSE
            MOV     &OPER1,&MACS    ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
            .ENDIF
            .IF OPERAND1=8
            SXT     &OPER2          ; OPER2 MEMORY LOCATION NEEDS
                                    ; 2 BYTES
            MOV.B   &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
            .ELSE
            MOV     &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
            .ENDIF
*********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION     *
*      TO THE RAM DATA IF NECESSARY                                 *
*      THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO AND        *
*      RESHI REGISTERS. THE UPPER TWO WORDS IN THE EXAMPLE ARE      *
*      FURTHER LOCATED IN THEIR RAM LOCATION                        *
*********************************************************************
            INC     MCOUNT              ; INC MACS COUNTER
            CMP     #MAXMACS,MCOUNT     ; ONLY ADD TO RAM IF NECESSARY
            JNE     NEXTMACS            ;
            ADDC    &RESLO,&RAM+0       ; ADD SUMEXTENSION TO RAM+0
            ADDC    &RESHI,&RAM+2       ; ADD SUMEXTENSION TO RAM+2
            ADDC    &SUMEXT,&RAM+4      ; ADD SUMEXTENSION TO RAM+4
            ADDC    &SUMEXT,&RAM+6      ; IF 64 BIT LENGTH IS USED
            CLR     MCOUNT
NEXTMACS
    . . .
```

6.3 Hardware Multiplier Registers

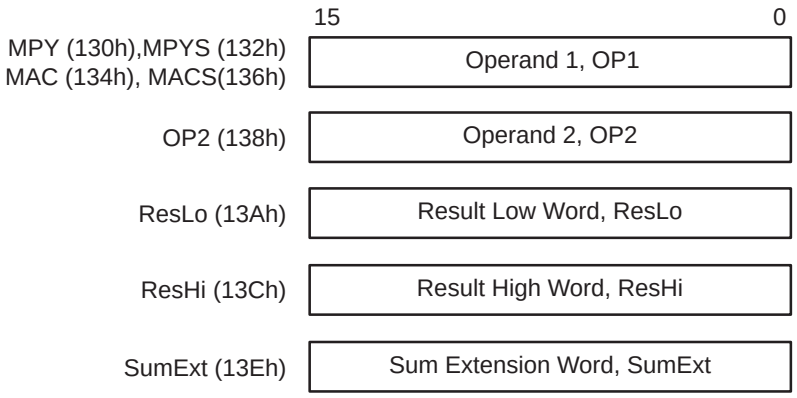
Hardware multiplier registers are word structured, but can be accessed using word or byte processing instructions. Table 6–2 describes the hardware multiplier registers.

Table 6–2. Hardware Multiplier Registers

Register	Short Form	Register Type	Address	Initial State
Multiply Unsigned (Operand1)	MPY	Read/write	0130h	Unchanged
Multiply Signed (Operand1)	MPYS	Read/write	0132h	Unchanged
Multiply+Accumulate (Operand1)	MAC	Read/write	0134h	Unchanged
Multiply Signed+Accumulate (Operand1)	MACS	Read/write	0136h	Unchanged
Second Operand	OP2	Read/write	0138h	Unchanged
Result Low Word	ResLo	Read/write	013Ah	Undefined
Result High Word	ResHi	Read/write	013Ch	Undefined
Sum Extend	SumExt	Read	013Eh	Undefined

Two registers are implemented for both operands, OP1 and OP2, as shown in Figure 6–3. Operand 1 uses four different addresses to address the same register. The different address information is decoded and defines the type of multiplication operation used.

Figure 6–3. Registers of the Hardware Multiplier



The multiplication result is located in two word registers: result high (RESHI) and result low (RESLO). The sum extend register (SumExt) holds the result sign of a signed operation or the overflow of the multiply and accumulate (MAC) operation. See Section 6.5.3 for a description of overflow and underflow when using the MACS operations.

All registers have the least significant bit (LSB) at bit0 and the most significant bit (MSB) at bit7 (byte data) or bit15 (word data).

6.4 Hardware Multiplier Special Function Bits

Because the hardware multiplier module completes all multiplication operations quickly, without interrupt intervention, no special function bits are used.

6.5 Hardware Multiplier Software Restrictions

Two restrictions require attention when the hardware multiplier is used:

- The indirect or indirect autoincrement address mode used to process the result
- The hardware multiplier used in an interrupt routine

6.5.1 Hardware Multiplier Software Restrictions—Address Mode

The result of the multiplication operation can be accessed in indexed, indirect, or indirect autoincrement mode. The result registers may be accessed without any restrictions if you use the indexed address mode including the symbolic and absolute address modes. However, when you use the indirect and indirect autoincrement address modes to access the result registers, you need at least one instruction between loading the second operand and accessing one of the result registers.

```
*********************************************************
*      EXAMPLE: MULTIPLY OPERAND1 AND OPERAND2
*********************************************************

RESLO    .SET   013AH       ; RESLO = ADDRESS OF RESLO
         PUSH   R5          ; R5 WILL HOLD THE ADDRESS OF
         MOV    #RESLO,R5   ; THE RESLO REGISTER

         MOV    &OPER1,&MPY ; LOAD 1ST OPERAND,
                            ; DEFINES ADD. UNSIGNED MULTIPLY
         MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                            ; MULTIPLICATION

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE        *
*      MULTIPLICATION TO THE RAM DATA, 64BITS           *
*********************************************************

         NOP                ; MIN. ONE CYCLES BETWEEN MOVING
                            ; THE OPERAND2 TO HW-MULTIPLIER
                            ; AND PROCESSING THE RESULT WITH
                            ; INDIRECT ADDRESS MODE
         ADD    @R5+,&RAM   ; ADD LOW RESULT TO RAM
         ADDC   @R5,&RAM+2  ; ADD HIGH RESULT TO RAM+2
         ADC    &RAM+4      ; ADD CARRY TO EXTENSION WORD
         ADC    &RAM+6      ; IF 64 BIT LENGTH IS USED

         POP    R5
```

The previous example shows that the indirect or indirect autoincrement address modes, when used to transfer the result of a multiplication operation to the destination, need more cycles and code than the absolute address mode. There is no need to access the hardware multiplier using the indirect addressing mode.

6.5.2 Hardware Multiplier Software Restrictions—Interrupt Routines

The entire multiplication routine requires only three steps:

1) Move operand OP1 to the hardware multiplier; this defines the type of multiplication.

2) Move operand OP2 to the hardware multiplier; the multiplication starts.

3) Process the result of the multiplication in the RESLO, RESHI, and SUMEXT registers.

The following considerations describe the main routines that use hardware multiplication. If no hardware multiplication is used in the main routine, multiplication in an interrupt routine is protected from further interrupts, because the GIE bit is reset after entering the interrupt service routine. Typically, a multiplication operation that uses the entire data process occurs outside an interrupt routine and the interrupt routines are as short as possible.

A multiplication operation in an interrupt routine has some feedback to the multiplication operation in the main routine.

6.5.2.1 Interrupt Following an OP1 Transfer

The two LSBs of the first operand address define the type of multiplication operation. This information cannot be recovered by any later operation. Therefore an interrupt must not be accepted between the first two steps: move operand OP1 and OP2 to the multiplier.

6.5.2.2 Interrupt Following an OP2 Transfer

After the first two steps, the multiplication result is in the corresponding registers RESLO, RESHI, and SUMEXT. It can be saved on the stack (using the PUSH instruction) and can be restored after completing another multiplication operation (using the POP instruction). However, this operation takes additional code and cycles in the interrupt routine. You can avoid this, by making an entire multiplication routine uninterruptible, by disabling any interrupt (DINT) before entering the multiplication routine, and by enabling interrupts (EINT) after the multiplication routine is completed. The negative aspect of this method is that the critical interrupt latency is increased drastically for events that occur during this period.

6.5.2.3 General Recommendation

In general, one should avoid a hardware multiplication operation within an interrupt routine when a hardware multiplication is already used in the main program. (This will depend upon the application-specific software, applied libraries, and other included software.) The methods previously discussed have some negative implications; therefore, the best practice is to keep interrupt routines as short as possible.

6.5.3 Hardware Multiplier Software Restrictions—MACS

The multiplier does not automatically detect underflow or overflow in the MACS mode. An overflow occurs when the sum of the accumulator register and the result of the signed multiplication exceed the maximum binary range.

The binary range of the accumulator for positive numbers is 0 to $2^{31}-1$ (7FFF FFFFh) and for negative numbers is –1 (0FFFF FFFFh) to -2^{31} (8000 0000h). An overflow occurs when the sum of two negative numbers yields a result that is in the range given above for a positive number. An underflow occurs when the sum of two positive numbers yields a result that is in the range for a negative number.

The maximum number of successive MACS instructions without underflow or overflow is limited by the individual application and should be determined using a worst-case calculation. Care should then be exercised to not exceed the maximum number or to handle the conditions accordingly.

Chapter 7

FLL Clock Module

This chapter discusses the FLL clock module used in the MSP430x3xx families. The FLL clock module in the MSP430x3xx includes a watch-crystal oscillator, an RC-type digitally-controlled oscillator (DCO), and a frequency-locked-loop (FLL) to ensure the accuracy of the DCO.

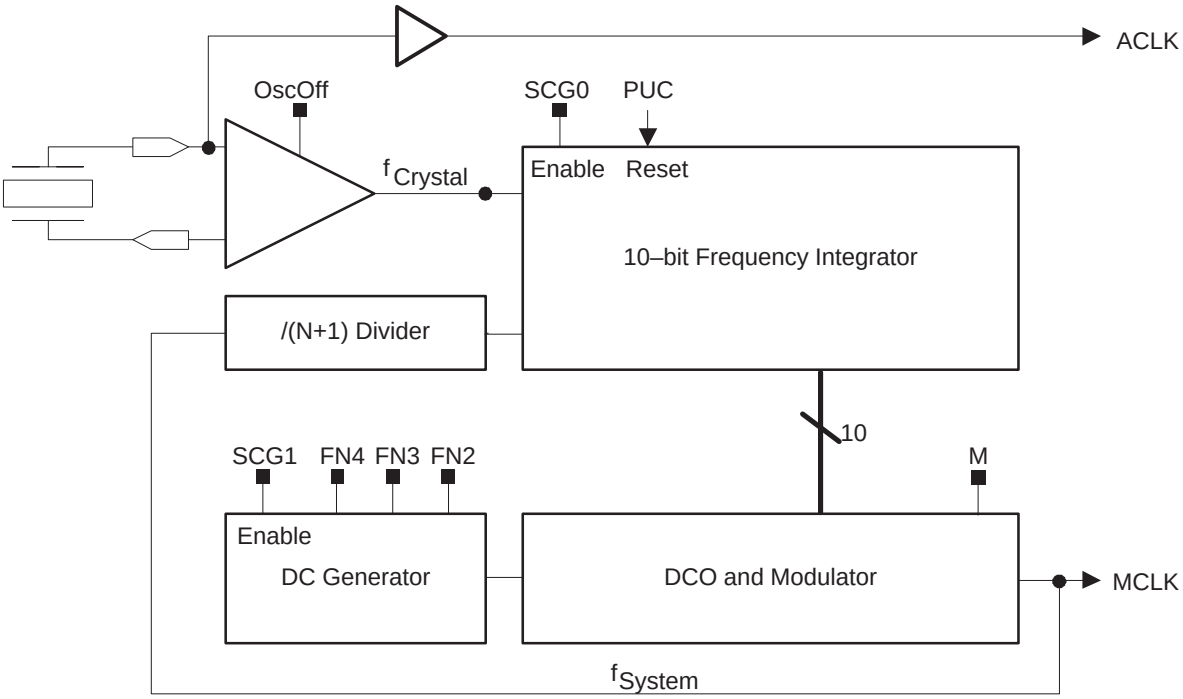
Topic **Page**

7.1	The FLL Clock Module	7-2
7.2	Crystal Oscillator	7-3
7.3	Digitally-Controlled Oscillator (DCO) and Frequency-Locked Loop	7-4
7.4	FLL Operating Modes	7-7
7.5	Buffered Clock Output	7-8
7.6	FLL Module Control Registers	7-9

7.1 The FLL Clock Module

The frequency-locked loop (FLL) clock module (shown in Figure 7–1) follows the major design targets of low system cost and low-power consumption. The FLL operates completely using a 32768-Hz watch crystal. A second asynchronous high-speed clock signal is generated on-chip using a digitally-controlled oscillator (DCO). The DCO frequency is stabilized to a multiple of the watch crystal frequency by dividing the DCO frequency and digitally locking the two frequencies. This technique is known as frequency-locked loop.

Figure 7–1. Frequency-Locked Loop



The FLL module supplies the MSP430x3xx family of devices with two clock signals and an associated software-selectable buffered clock output.

- ACLK, a crystal oscillator signal used by peripheral modules. This signal is identical to the frequency of the crystal oscillator input, XIN. ACLK is also known as $f_{crystal}$.
- MCLK, the controller's main system clock used by the CPU; this clock is software selectable for individual peripheral modules. The MCLK is identical to the frequency generated by the DCO. MCLK is also known as f_{system}.
- XBUF, buffered output of either MCLK, ACLK, ACLK/2, ACLK/4, or off.

7.2 Crystal Oscillator

The crystal oscillator supports low-current consumption by using a 32,768 Hz watch crystal. The crystal connects to XIN and XOUT without any other external components. This oscillator generates the ACLK signal which is available to on-chip peripherals and XBUF.

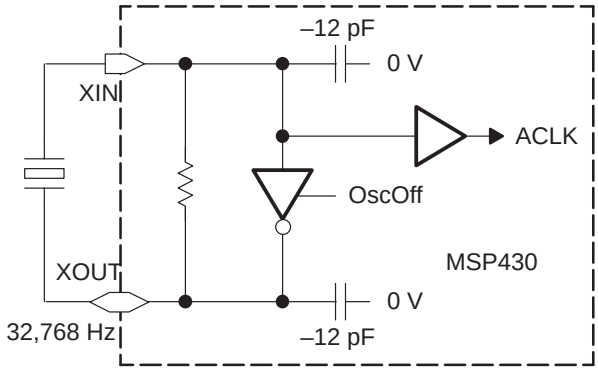
Two factors determine the choice of the watch crystal:

- Low-current consumption
- Stable time base

The oscillator operates after applying $V_{CC.}$ Since the OscOff control bit in the Status register (SR) is reset. It can be stopped by software by setting the OscOff bit in the SR (OscOff = 1). When OscOff mode is selected (see Chapter 3) the ACLK signal is held in a high state.

All components required for crystal operation are integrated into the MSP430 as shown in Figure 7–2. No additional external components are necessary for operation. Because the oscillator is designed for ultralow-power dissipation, short connections between the crystal and MSP430 devices should be used for the PWB layout.

Figure 7–2. Crystal Oscillator Schematic



7.3 Digitally-Controlled Oscillator (DCO) and Frequency-Locked Loop

The DCO is an integrated RC-type oscillator in the MSP430x3xx FLL clock module. The DCO generates a clock signal called MCLK. The MCLK generated by the DCO is used by the MSP430x3xx CPU and is available to on-chip peripherals and XBUF. MCLK is set to an N+1 multiple of ACLK. The N multiplier is contained in the lowest 7 bits of control register SCFQCTL (SCFQCTL.6 ... SCFQCTL.0). N is set to 31 on PUC by default, resulting in an effective ACLK multiplier of 32 and an MCLK of 1.048576 MHz, assuming that ACLK is 32, 768 Hz.

The multiplier (N+1) sets the frequency of MCLK:

$$MCLK = (N + 1) \times ACLK$$

MCLK is stabilized using a frequency-locked loop technique. When combined with the DCO, two important benefits result:

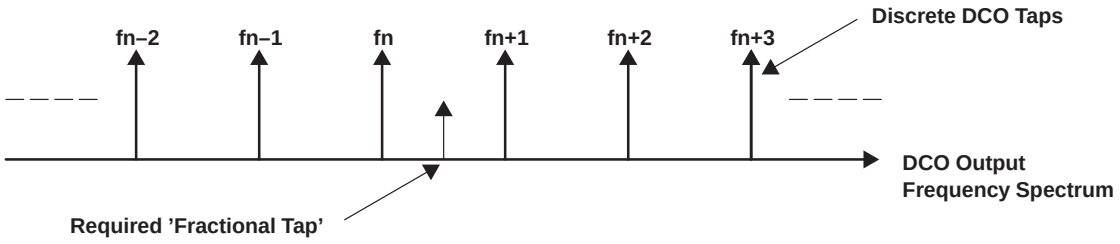
- Fast start-up. The MSP430x3xx DCO is active in less than 6 μs, which supports extended sleep periods and burst performance.
- Digital control signals. The DCO starts at exactly the same setting as when shutoff. Thus a long locking period is not required for normal operation.

User software can modify MCLK by changing the multiplier N at any time. The exact minimum and maximum MCLK allowed is specified in the device data sheet.

7.3.1 FLL Operation

As with any RC-type oscillator, frequency varies with temperature and voltage. The FLL hardware automatically stabilizes MCLK. The FLL compares the ACLK to MCLK/(N+1) and counts up or down a 10-bit frequency integrator. The MCLK is constantly adjusted to one of 1024 possible settings. The output of the frequency integrator that drives the DCO can be read in SCFI1 and SCFI0. The count is adjusted +1 or –1 with each crystal period (30.5 μs using 32,768 Hz). Of the 10-bits from the frequency integrator, 5-bits are used for DCO frequency taps and 5-bits are used for a modulator. The 5-bits for the DCO tap are contained in the SCFI1 (SCFI1.7 . . . SCFI11.3). There are 29 *taps* implemented in the DCO (TAPS 28, 29, 30, and 31 are equivalent), each being approximately 10% higher than the previous. In most applications, a fraction tap may be required to achieve the programmed MCLK over the full range of system operation (see Figure 7–3).

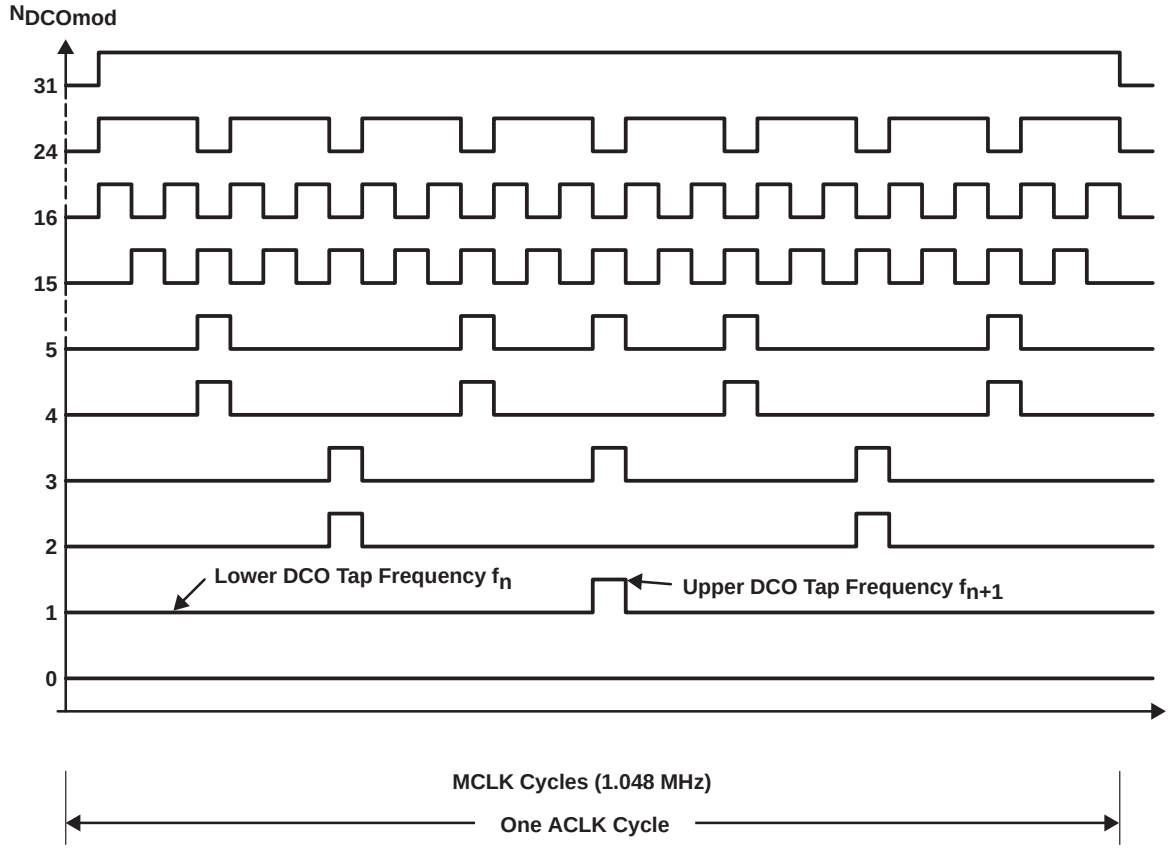
Figure 7–3. Fractional Tap Frequency Required



7.3.2 Modulator Operation

The modulator overcomes relatively-large tap steps by mixing a DCO tap with the next higher-frequency tap DCO+1. The DCO mixing or hop pattern is controlled with 5-bits; thus there are 32 possible mix patterns (see Figure 7–4). The 5-bits for the modulator are contained in SCFI1 and SCFI0 (SCFI1.2...SCFI1.0, SCFI0.1, and SCFI0.0).

Figure 7–4. Modulator Hop Patterns



7.3.3 DCO Frequency Range

The fundamental-frequency range of the DCO is centered based on $f_{nominal}$ approximately equal to 1 MHz using bits FN_2, FN_3, and FN_4 in SCFIO (see Table 7–1). The range control allows the DCO to operate near the center of the available taps for a given MCLK.

Table 7–1. The DCO Range Control Bits

FN_4	FN_3	FN_2	MCLK FREQUENCY
0	0	0	1 x $f_{nominal}$
0	0	1	2 x $f_{nominal}$
0	1	X	3 x $f_{nominal}$
1	X	X	4 x $f_{nominal}$

7.3.4 Disabling the FLL

FLL loop control and modulation can be disabled independently. FLL loop control can be disabled by setting the SCG0 bit in the status register (SR). In this case, the DCO runs at the previous tap—open loop. Then the MCLK is not automatically stabilized to (N+1) × ACLK. The influence of the modulator can be disabled by setting the modulation bit M (SCFQCTL.7). In this case the MCLK is stabilized to (N+1) × ACLK every 1024 cycles to the nearest 32 DOC taps.

7.3.5 MCLK Stability

The DCO is absolutely monotonic and the 10-bits of the frequency integrator continuously count up/down by one. The accuracy of MCLK is the same as that of ACLK if the FLL is running continuously.

The accumulated error in MCLK tends to zero over a long period. The 10-bit FLL integrator is automatically adjusted every period of the ACLK. Thus, a positive frequency deviation over one ACLK period is compensated with a negative deviation over the next ACLK period. Variation between MCLK clock periods can be approximately 10% due to the modulator mixing of DCO taps, while the accumulated system clock error over longer time periods is zero.

7.3.6 Oscillator Fault Detection

MSP430x3xx devices have a fail-safe mode when the external crystal fails. If the crystal fails and no ACLK signal is generated, the FLL will continue to count down to zero in an attempt to lock ACLK and MCLK/(N+1). An internal oscillator fault is detected if the DCO tap moves out of the range 0<Ndco<28; that is, an oscillator fault is signaled if the five bits SCFI1.7...SCFI11.3 contain one the values 0, 28, 29, 30, or 31. An oscillator fault sets the oscillator-fault interrupt flag (OFIFG) in the interrupt flag register 1 (IFG1) permanently as long as the fault condition is valid. If the oscillator-fault interrupt-enable bit (OFIE) is set by user software in the interrupt enable register 1 (IE1) and an oscillator fault occurs, a nonmaskable interrupt (NMI) is generated. When the interrupt is granted, the OFIE is reset automatically by hardware; user software must reset OFIFG. The NMI interrupt has two sources. User software must interrogate the OFIFG bit to determine if the NMI was generated by an oscillator fault.

Note:

MCLK is active even at the lowest DCO tap. The MCLK signal is available for the CPU to execute code and service an NMI.

7.4 FLL Operating Modes

Control bits SCG0, SCG1, OscOff, and CPUOff in the status register configure the MSP430x3xx operating modes as discussed in Chapter 3, *System Resets, Interrupts, and Operating Modes*.

7.4.1 Starting From Power Up Clear (PUC)

On a valid PUC, SCFQCTL = 31, SCFIO and SCFI1 are cleared, and SCG0, SCG1, OscOff, and CPUOff in the status register are reset. The FLL is fully operational and will adjust the DCO until MCLK = (31+1) × ACLK. Using a 32,768-Hz watch crystal for ACLK, MCLK will stabilize to 1.048576 MHz.

Because the DCO starts at the lowest tap on PUC, enough time must be allowed for the DCO to settle on the proper tap for normal operation. This is necessary only after PUC, or when SCFIO and SCFI1 are cleared. 32 ACLK cycles are required to get from one tap to another. Twenty-nine taps are implemented, requiring 27 × 32 ACLK cycles as the worst case for the DCO to settle on the proper tap (taps 0 and 27 are not counted since OFIFG is set at these taps). During initialization, this time should be left prior to precise MCLK timing. During normal operation, the FLL will constantly adjust the DCO, requiring no special considerations.

7.4.2 Adjusting the FLL Frequency

User software can adjust the FLL frequency at any time by changing the N multiplier in the SCFQCTL register. Also, bits FN_2, FN_3, and FN_4 are adjusted to the appropriate MCLK frequency range.

Example, MCLK = 64 × ACLK = 2097152

```
bic      #GIE,SR              ; Disable interrupts
mov.b    #(64–1),&SCFQTL      ; MCLK = 64 * ACLK
mov.b    #FN_2,&SCFIO         ; DCO centered at 2 MHz
bis      #GIE,SR              ; Enable interrupts
```

Example, MCLK = 100 × ACLK = 3276800

```
bic      #GIE,SR              ; Disable interrupts
mov.b    #(100–1),&SCFQTL     ; MCLK = 100 * ACLK
mov.b    #FN_3,&SCFIO         ; DCO centered at 3 MHz
bis      #GIE,SR              ; Enable interrupts
```

7.4.3 FLL Features for Low-Power Applications

Three conflicting requirements typically exist in battery-powered MSP430x3xx real-time applications:

- Low-frequency clock for energy conservation and time keeping
- High-frequency clock for fast reaction to events and fast burst-processing capability

- Clock stability

The MSP430x3xx FLL clock system addresses the above conflicting requirements by providing both a low-frequency ACLK with crystal stability and a stable high-frequency MCLK with near instant on-capability. The DCO, which generates the MSP430x3xx MCLK, is operation in less than 6 µS.

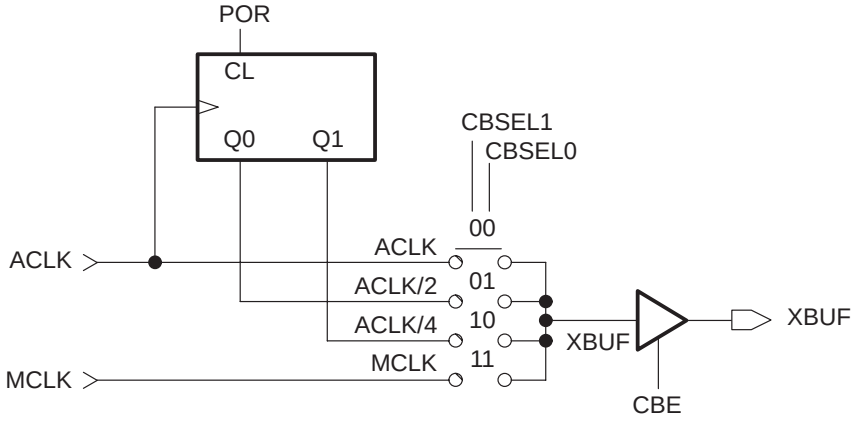
The choice of a digital frequency-locked loop versus an analog-phase locked loop enables the benefit of fast-start and stability. A phase-locked loop takes hundreds or thousands of clock cycles to start and stabilize. The MSP430x3xx frequency-locked-loop starts immediately at the exact setting prior to shut down.

For minimum power consumption, the MSP430x3xx system operates for extended periods in low-power mode 3 (LPM3) with only the ACLK active for timers and low-power peripherals. Interrupts, both from external and internal events, drive the activation of MCLK for the CPU and high-speed peripherals. In the MSP430x3xx, any interrupt stores the SR operating modes on the stack and then clears the SCG1 bit in the SR, automatically starting the DCO and MCLK. After the interrupt handler has completed, the saved SR is popped from the stack with the RETI instruction, restoring the previous operating mode.

7.5 Buffered Clock Output

The clock buffer shown in Figure 7–5 allows ACLK, ACLK/2, ACLK/4, or MCLK to be output on MSP430x3xx pin XBUF. The clock buffer is controlled using the three bits CBE, CBSEL1, and CBSEL0 in control register CBCTL.

Figure 7–5. Schematic of Clock Buffer



CBE enables XBUF when set. CBSEL1 and CBSEL0 select the clock source of an enabled XBUF. On a POR condition, CBSEL1, CBSEL0, and CBE are reset and XBUF is disabled. If either ACLK or MCLK is shut down (generating no frequency) and this clock source (or fraction of) is selected for XBUF, no frequency will be output on XBUF regardless of CBE.

Note:

Control register CBCTL is a write-only register. Only `mov.B #xxh, &CBCTL` instructions should be used to access this register. Other Format 1 instructions, which are a read-then-write type, will result in incorrect setting.

7.6 FLL Module Control Registers

The FLL module is configured using control registers SCFQCTL, SCFIO, SCFI1, CBCTL, and four bits from the CPU status register: SCG1, SCG0, OscOff, and CPUOff. User software can modify these control registers from their default condition at any time. The FLL control registers are located in the byte-wide peripheral map and should be accessed with byte (.B) instructions.

Register	Short Form	Register Type	Address	Initial State
System clock control	SCFQCTL	Read/write	052h	031h
System clock frequency integrator 0	SCFI0	Read/write	050h	Reset
System clock frequency integrator 1	SCFI1	Read/write	051h	Reset
Clock buffer	CBCTL	Write only	053h	Reset

7.6.1 MCLK Frequency Control

The contents of register SCFQCTL controls the multiplication of the crystal frequency. The contents of register SCFQCTL is shown in Figure 7–6.

Figure 7–6. SCFQCTL Register

SCFQCTL 052h

7							0
M	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

The seven bits indicate a range of 1+1 to 127+1. Any value below 1 results in unpredictable operation. The user should ensure that the value selected does not exceed the maximum MCLK value specified in the device data sheet.

$$f_{System} = (x \cdot 2^6 + x \cdot 2^5 + x \cdot 2^4 + x \cdot 2^3 + x \cdot 2^2 + x \cdot 2^1 + x \cdot 2^0 + 1) \cdot f_{crystal}$$

The default value in SCFQCTL is 31 after a PUC signal is active, resulting in a factor of 32.

The output of the frequency integrator controls the DCO. This value can be read using the SCFI1 and SCFI0 addresses as shown in Figure 7–7.

Figure 7–7. SCFI0 and SCFI1 Registers

If the modulation bit M is set, only the DCO taps determine the system frequency. Adjacent DCO taps are not mixed. Note, however, that if the FLL remains active (SCG0=0), it will continue to adjust the DCO taps. If an application requires the system frequency to remain constant for a short period of time, both the modulation and the FLL should be disabled (M=1, SCG0=1).

SCFI0 050h

7							0
0	0	0	FN_4	FN_3	FN_2	2^1	2^0
r	r	r	rw-0	rw-0	rw-0	rw-0	rw-0

SCFI1 051h

7							0
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 7–8. Crystal Buffer Control Register

CBCTL 053h

†	†	†	†	†	CBSEL 1	CBSEL 0	CBE
					w–(0)	w–(0)	w–(0)

† Not implemented

Bit 0: Bit CBE controls the output buffer configuration.
CBE = 1: Output buffer enabled
CBE = 0: Output buffer disabled

Bits 1, 2: Bits CBSEL1 and CBSEL0 select the frequency that can be applied to output pin XBUF.

CBE	CBSEL1	CBSEL0	XBUF
0	X	X	Disabled
1	0	0	ACLK
1	0	1	ACLK/2
1	1	0	ACLK/4
1	1	1	MCLK

7.6.2 Special-Function Register Bits

The FLL clock module affects two bits in the special-function registers, OFIFG and OFIE. The oscillator fault-interrupt enable bit (OFIE) is located in bit 1 of the interrupt-enable register IE1. The oscillator fault-interrupt flag bit (OFIFG) is located in bit 1 of the interrupt-flag register IFG1.

IE1 00h

7	6	5	4	3	2	1	0
						OFIE	
						rw–0	

IFG1 02h

7	6	5	4	3	2	1	0
						OFIFG	
						rw–1	

The oscillator fault signal sets the OFIFG as long as the oscillator fault condition is active. The detection and effect of the oscillator fault condition is described in section 7.3.6. The oscillator fault interrupt requests a nonmaskable interrupt if the OFIE bit is set. The oscillator interrupt-enable bit is reset automatically if a non-maskable interrupt is accepted. The initial state of the OFIE bit is reset, and no oscillator fault requests an interrupt even if a fault condition occurs.

Chapter 8

Digital I/O Configuration

This chapter describes the digital I/O configuration.

Topic **Page**

8.1	Introduction	8-2
8.2	Port P0	8-3
8.3	Ports P1, P2	8-11
8.4	Ports P3, P4	8-17

8.1 Introduction

The general-purpose I/O ports of the MSP430 are designed to give maximum flexibility. Each I/O line is individually configurable, and most have interrupt capability.

There are several different I/O port modules that function in slightly different ways. For this reason, names have been given to each port module. For example, port P0, P1, P2, etc. These names refer to specific port modules, and apply to all MSP430 devices. For example, port P0 and P1 may be available on a particular MSP430 device, while ports P1 and P3 may be available on another device. It is important for the user to understand the operating differences and which port(s) are available on the device in use.

Additionally, the I/O port pins are often multiplexed with other pin functions on the devices to provide maximum flexibility while optimizing pin count on the devices.

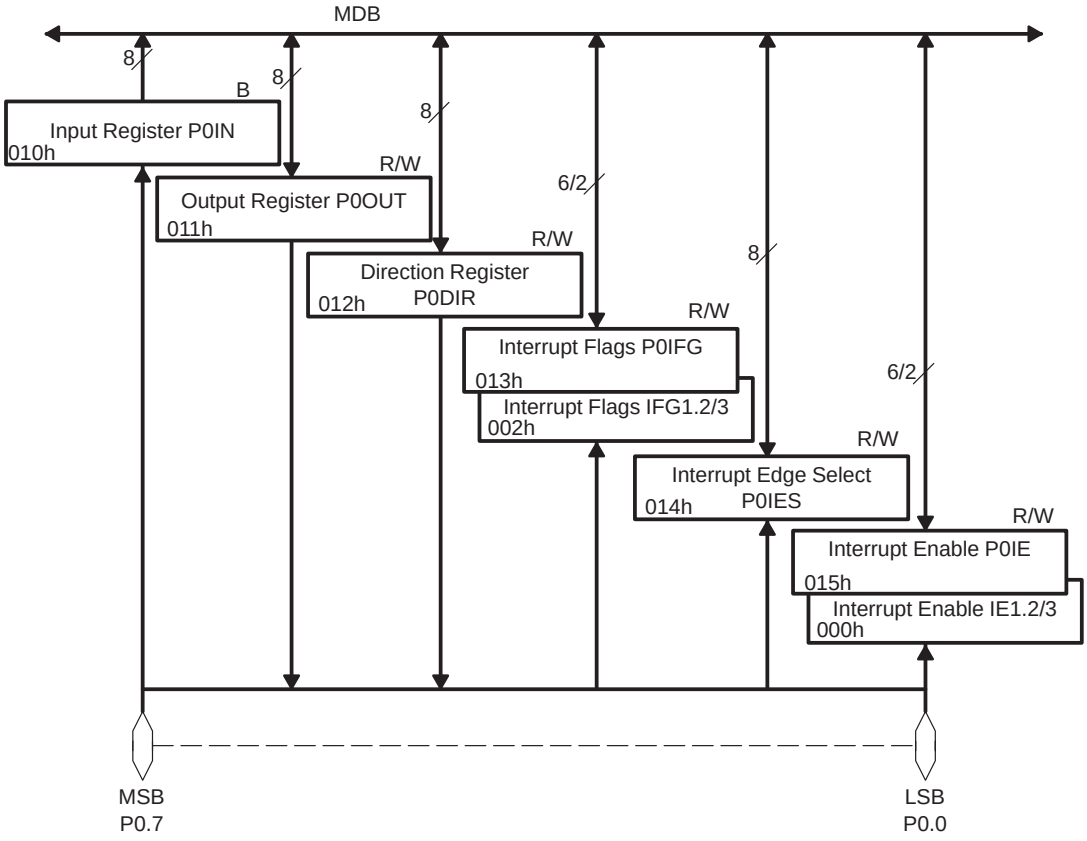
8.2 Port P0

The general-purpose port P0 contains 8 general-purpose I/O lines and the required registers to control and configure them. Each I/O line is capable of being controlled independently. In addition, each I/O line has interrupt capability.

Six registers are used to control the port I/O pins (see Section 8.2.1).

Port P0 is connected to the processor core through the 8-bit memory data bus (MDB) and the memory address bus (MAB). Port P0 should be accessed using byte instructions in the absolute address mode, such as:
MOV.B #12h,&P0OUT.

Figure 8–1. Port P0 Configuration



8.2.1 Port P0 Control Registers

Port P0 has six registers to control the I/O pins. The six control registers give maximum input/output configuration flexibility:

- ❑ All individual I/O bits are independently programmable.
- ❑ Any combination of input, output, and interrupt condition is possible.
- ❑ Interrupt processing of external events is fully implemented for all eight bits of port P0.

The six registers are shown in Table 8–1.

Table 8–1. Port P0 Control Registers

Register	Short Form	Register Type	Address	Initial State
Input	P0IN	Read only	010h	– – – – –
Output	P0OUT	Read/write	011h	Unchanged
Direction	P0DIR	Read/write	012h	Reset
Interrupt flags	P0IFG	Read/write	013h	Reset
Interrupt edge select	P0IES	Read/write	014h	Unchanged
Interrupt enable	P0IE	Read/write	015h	Reset

These registers contain eight bits except for the two LSBs in the interrupt flag register and interrupt enable register. These two bits are included in the special function register. The registers should be accessed using byte instructions and absolute address mode.

8.2.1.1 Input Register P0IN

The input register is a read-only register that shows the values of the signals at the I/O pins. The direction of the pin must be selected as input.

Note: Writing to the Read-Only Register P0IN

Any attempt to write to this read-only register results in increased current consumption while the write attempt is active.

8.2.1.2 Output Register P0OUT

The output register shows the information of the output buffer. The output buffer can be modified using all instructions that write to a destination. If read, the contents of the buffer are independent of the pin direction. A direction change does not modify the output buffer contents.

8.2.1.3 Direction Register P0DIR

The direction register contains eight bits that define the direction of each I/O pin. All bits are reset by the PUC signal.

When:

Bit = 0: The I/O pin is switched to input direction

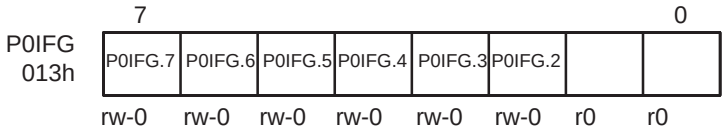
Bit = 1: The I/O pin is switched to output direction

8.2.1.4 Interrupt Flags P0IFG

The interrupt flags register contains six flags that reflect whether or not an interrupt is pending from the corresponding I/O pin, if the I/O pins are interrupt-enabled.

Three interrupt vectors are implemented for port P0; one for port P0.0, one for port P0.1, and one for interrupt events on ports P0.2 to P0.7. The six flags shown in Figure 8–2 are located in bits 7 to 2 and correspond to pins P0.7 to P0.2. The interrupt flags for pins P0.1 and P0.0 are located in the SFRs.

Figure 8–2. Interrupt Flags Register



When:

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending due to a transition at the I/O pin or software setting the bit. Manipulation of P0OUT and P0DIR can also set the P0IFG bits.

Writing a zero to an interrupt flag resets it; writing a one to an interrupt flag sets it and generates an interrupt.

Interrupt flags P0IFG.2 to P0IFG.7 use only one interrupt vector. These flags are not reset automatically when any interrupt from these events is served. The software should determine which event is served and reset the appropriate flag(s).

Flags P0IFG.0 and P0IFG.1 generate individual interrupts, and are reset automatically when serviced.

Note:

Any external interrupt event should be as long as 1.5 times MCLK or longer to ensure that is accepted and the corresponding interrupt flag is set.

8.2.1.5 Interrupt Edge Select P0IES

The interrupt edge select register contains a bit for each I/O pin, which controls which transition triggers the interrupt flag. All eight bits corresponding to pins P0.7 to P0.0 are located in this register. When:

Bit = 0: The interrupt flag is set with a low-to-high transition

Bit = 1: The interrupt flag is set with a high-to-low transition

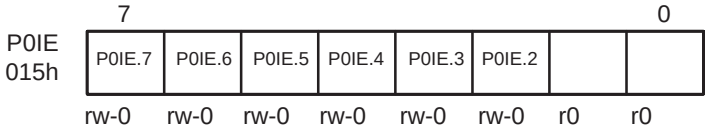
Note:

Any change in the P0IES bit(s) may result in setting the associated interrupt flags.

8.2.1.6 Interrupt Enable P0IE

The interrupt enable register contains bits for I/O pins P0.7 to P0.2, as shown in Figure 8–3, which enable an interrupt request for an interrupt event on these pins. Two interrupt enable bits for P0.0 and P0.1 are located in special function registers IE1.2 and IE1.3.

Figure 8–3. Interrupt Enable Register



When:

Bit = 0: The interrupt request is disabled

Bit = 1: The interrupt request is enabled

Note: Port P0 Interrupt Sensitivity

Only transitions, not static levels, cause interrupts.

The interrupt routine must reset the interrupt flags P0IFG.2 to P0IFG.7. Flags P0IFG.0 and P0IFG.1 are reset automatically when these interrupts are serviced.

If an interrupt flag is still set when the RETI instruction is executed (for example, a transition occurs during the interrupt service routine), an interrupt occurs again after RETI is completed. This ensures that each transition is acknowledged by the software.

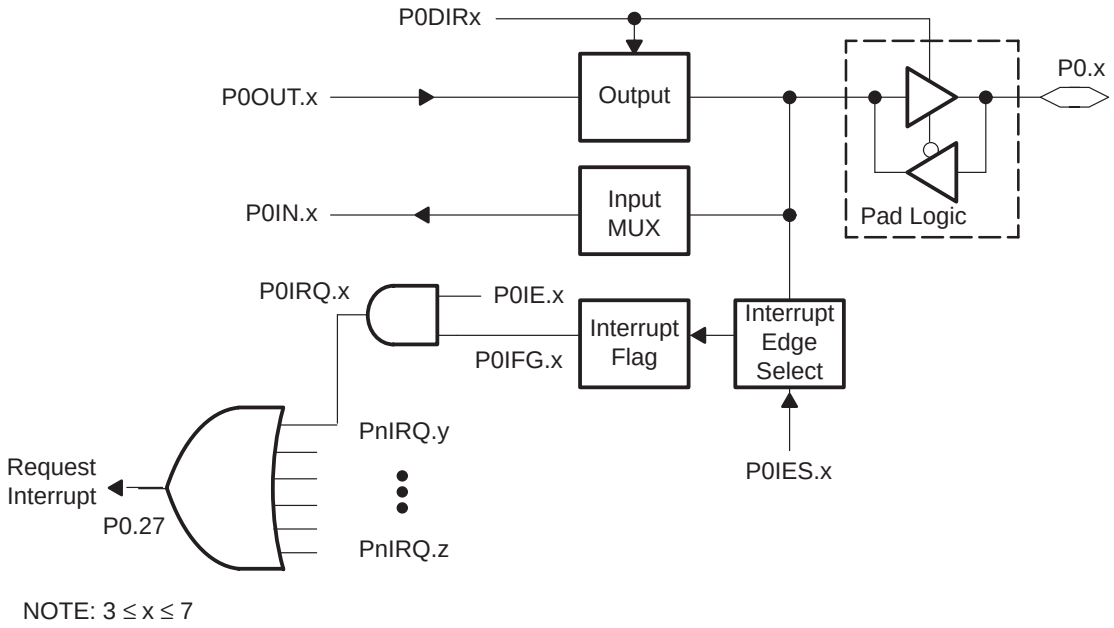
8.2.2 Port P0 Schematic

The pin logic of each individual port P0 signal can be read from and written to as described in the following sections.

8.2.2.1 Port P0, Bits P0.3 to P0.7

Each port P0 signal's pin logic is built from five identical register bits—P0DIR, P0OUT, P0IFG, P0IE, P0IES—and one read-only input buffer, P0IN. Bits 3 through 7 function identically as shown in Figure 8–4.

Figure 8–4. Schematic of Bits P0.7 to P0.3

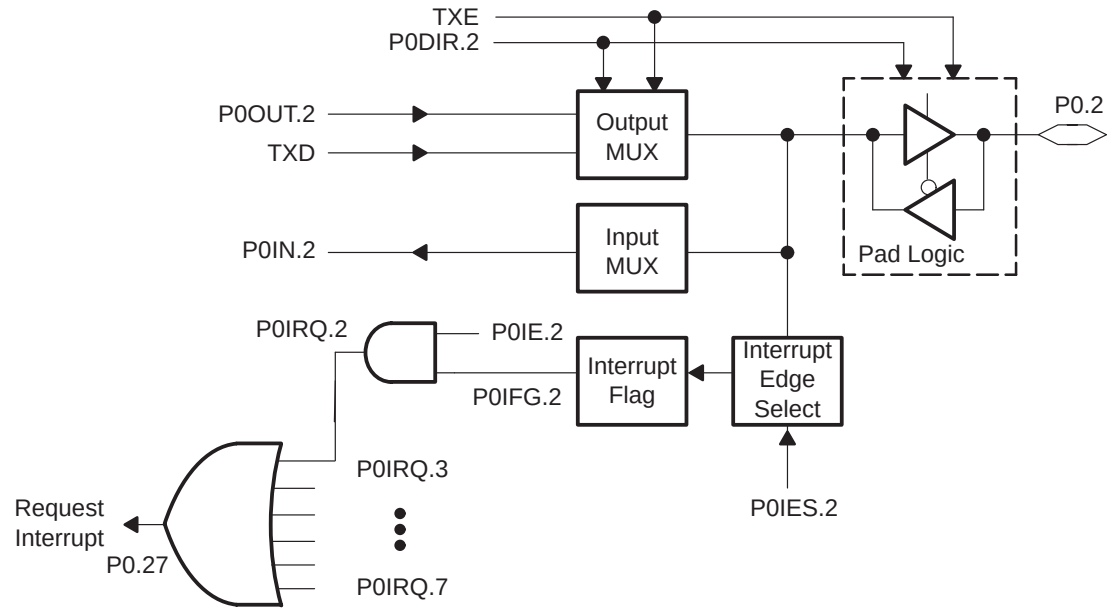


NOTE: $3 \leq x \leq 7$

8.2.2.2 Port P0, Bit P0.2

Bit 2 is slightly different from bits 3 to 7 as shown in Figure 8–5. The output signal can be determined either by the port P0OUT.2 bit or by the 8-Bit Timer/Counter signal (TXD). When the output control register bit (TXE) is set to a logic 1, the TXD signal is selected as the relevant output signal and the pad logic is switched to the output, independent of the direction control bit P0DIR.2.

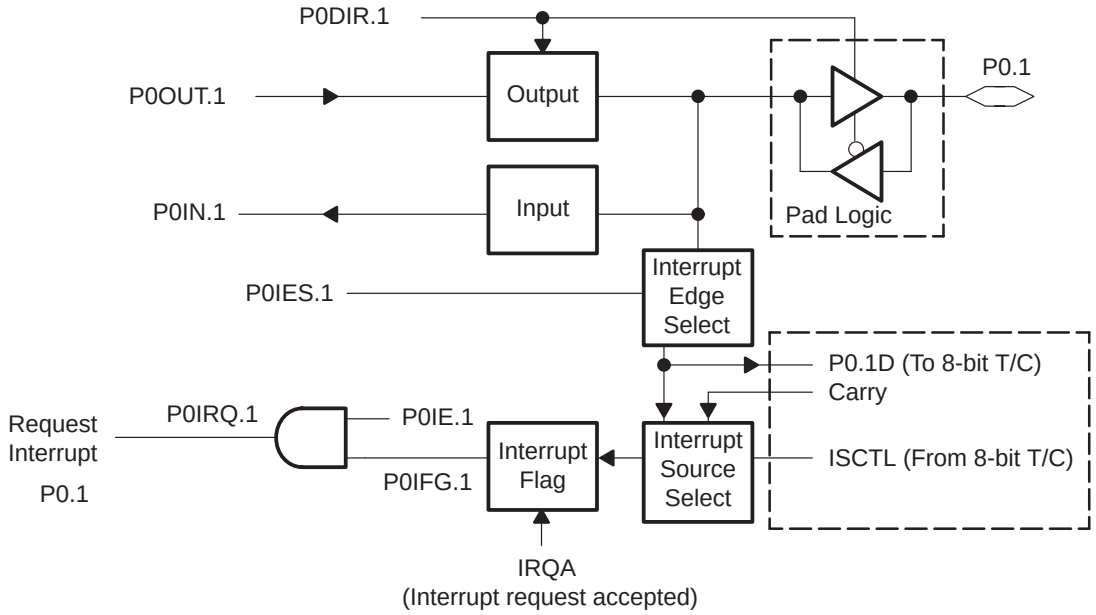
Figure 8–5. Schematic of Bit P0.2



8.2.2.3 Port P0, Bit P0.1

Bit 1 is slightly different from bits 3 to 7 as shown in Figure 8–6. The interrupt signal can be sourced by the input signal at pin P0.1, or by the 8-Bit Timer/Counter carry signal. Whenever the interrupt source control bit (ISCTL) in the 8-Bit Timer/Counter control register (TCCTL) is set, the interrupt source is switched from pin P0.1 to the carry signal from the counter in the 8-bit Timer/Counter. Flag P0IFG.1 is reset automatically when the interrupt is serviced (IRQA signal).

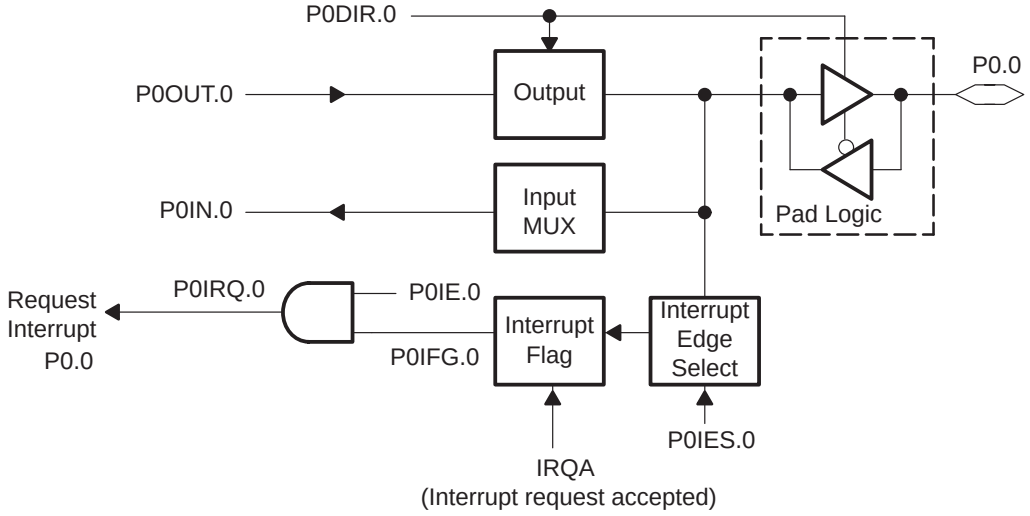
Figure 8–6. Schematic of Bit P0.1



8.2.2.4 Port P0, Bit P0.0

Bit 0 is identical to bits 3 to 7 as shown in Figure 8–7, but has its own interrupt vector. Flag P0IFG.0 is reset automatically when the interrupt is serviced (IRQA signal).

Figure 8–7. Schematic of Bit P0.0



8.2.3 Port P0 Interrupt Control Functions

Port P0 uses eight bits for interrupt flags, eight bits to enable interrupts, eight bits to select the effective edge of an interrupt event, and three different interrupt vector addresses.

The three interrupt vector addresses are assigned to:

- P0.0
- P0.1/RXD
- P0.2 to P0.7

The two port P0 signals, P0.0 and P0.1/RXD, are used for dedicated signal processing. Four bits in the SFR address range and two bits in the port0 address frame handle the interrupt events on P0.0 and P0.1/RXD :

- P0.0 interrupt flag P0IFG.0 (located in IFG1.2, initial state is reset)
- P0.1/RXD interrupt flag P0IFG.1 (located in IFG1.3, initial state is reset)
- P0.0 interrupt enable P0IE.0 (located in IE1.2, initial state is reset)
- P0.1/RXD interrupt enable P0IE.1 (located in IE1.3, initial state is reset)
- P0.0 interrupt edge select (located in P0IES.0, initial state is reset)
- P0.1/RXD interrupt edge select (located in P0IES.1, initial state is reset)

Both interrupt flags (P0IFG.0 and P0IFG.1/RXD) are single source flags and are automatically reset when the processor serves them. The enable bits and edge select bits remain unchanged.

The interrupt control bits of the remaining six I/O signals, P0.2 to P0.7, are located in the I/O address frame. Each signal uses three bits for configuration and interrupt.

- Interrupt flag, P0IFG.2 to P0IFG.7
- Interrupt enable bit, P0IE.2 to P0IE.7
- Interrupt edge select bit, P0IES.2 to P0IES.7

The interrupt flags P0IFG.2 to P0IFG.7 share the same interrupt vector. An interrupt event on one or more pins of P0.2 to P0.7 requests an interrupt when two conditions are met: the appropriate individual enable bit P0IE.x ($2 \leq x \leq 7$) is set and the general interrupt enable (GIE) bit is set. Since the interrupts share the same interrupt vector, interrupt flags P0.2 to P0.7 are not automatically reset and, therefore, continue to generate interrupts until reset. The interrupt service routine software should handle the detection of the source and reset the appropriate flag when it is serviced.

Note:

Modifying the direction control bit or interrupt edge select bit for an I/O may result in setting the interrupt flag for that I/O line.

8.2.3.1 I/O-Pin Interrupt Handler for P0.2 to P0.7: Programming Example

The following code describes how to set the I/O pin interrupt handler.

```
; The I/O-PIN interrupt handler for P0.2 to P0.7 starts here
;
IOINTR  PUSH   R5                ;Save R5
        MOV.B  &P0IFG,R5         ;Read interrupt flags
        BIC.B  R5,&P0IFG         ;Clear status flags with the
                                 ;read data
                                 ;Additional set bits are not
                                 ;cleared!
        EINT                     ;Allow interrupt nesting
;

;R5 contains information about which I/O-pin(s) cause
;interrupts:

;the processing starts here.
;
        .....
        .....
        POP    R5                ;JOB done: restore R5
        RETI                     ;Return from interrupt
        .....
        .....

;Definition of interrupt vector table

        .sect  ”IO27_vec”,0FFE0h  ;The interrupt vector for
                                  ;flags P0IFG.2 and P0IFG.7
                                  ;are at memory address 0FFE0h
                                  ;in 3xx devices.
        .WORD  IOINTR             ;I/O-Pin (2 to 7) Vector in
                                  ;ROM
        ;
        .sect  ”RST_vec”,0FFFEh   ;Interrupt Vectors
        .WORD  RESET
```

8.3 Ports P1, P2

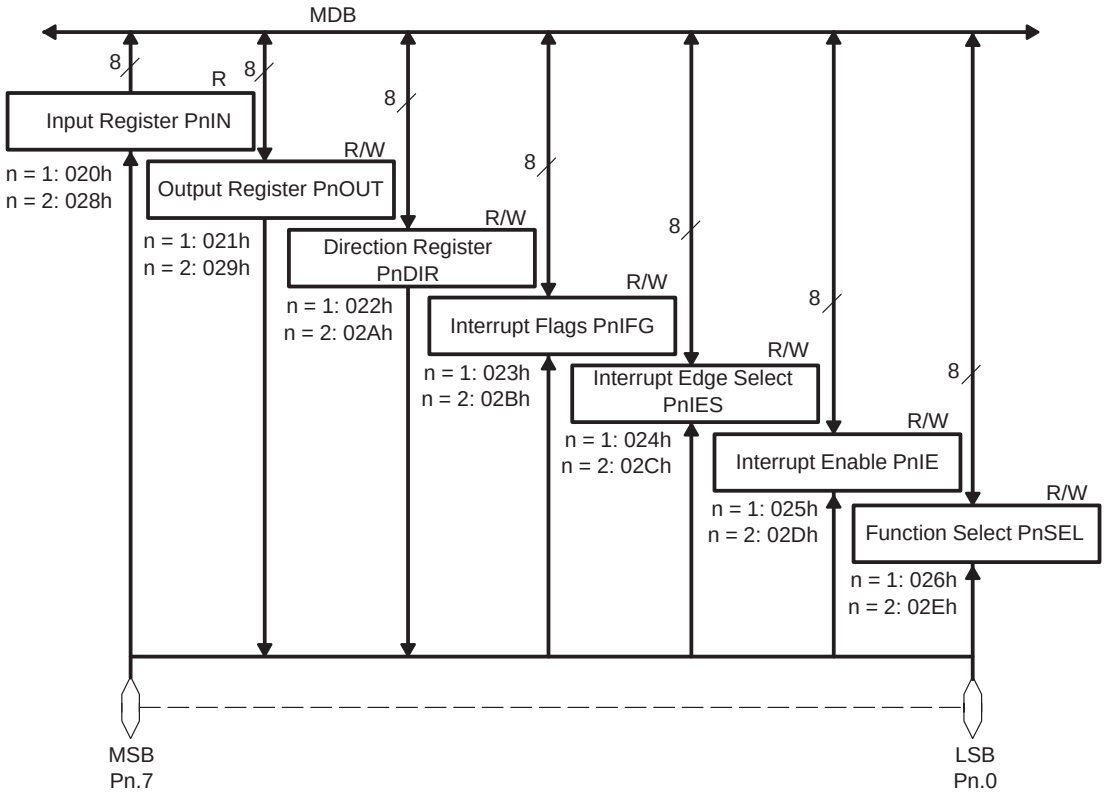
Each of the general-purpose ports P1 and P2 contain 8 general-purpose I/O lines and all of the registers required to control and configure them. Each I/O line is capable of being controlled independently. In addition, each I/O line is capable of producing an interrupt.

Separate vectors are allocated to ports P1 and P2 modules. The pins for port P1 (P1.0–7) source one interrupt, and the pins for port P2 (P2.0–7) source another interrupt.

Seven registers are used to control the port I/O pins (see Section 8.3.1).

Ports P1 and P2 are connected to the processor core through the 8-bit MDB and the MAB. They should be accessed using byte instructions in the absolute address mode.

Figure 8–8. Port P1, Port P2 Configuration



8.3.1 Port P1, Port P2 Control Registers

The seven control registers give maximum digital input/output configuration flexibility:

- ❑ All individual I/O bits are independently programmable.
- ❑ Any combination of input, output, and interrupt condition is possible.
- ❑ Interrupt processing of external events is fully implemented for all eight bits of ports P1 and P2.

The seven registers for port P1 and the seven registers for port P2 are shown in Table 8–2 and Table 8–3, respectively.

Table 8–2. Port P1 Registers

Register	Short Form	Register Type	Address	Initial State
Input	P1IN	Read only	020h	– – – – –
Output	P1OUT	Read/write	021h	Unchanged
Direction	P1DIR	Read/write	022h	Reset
Interrupt Flags	P1IFG	Read/write	023h	Reset
Interrupt Edge Select	P1IES	Read/write	024h	Unchanged
Interrupt Enable	P1IE	Read/write	025h	Reset
Function Select	P1SEL	Read/write	026h	Reset

Table 8–3. Port P2 Registers

Register	Short Form	Register Type	Address	Initial State
Input	P2IN	Read only	028h	– – – – –
Output	P2OUT	Read/write	029h	Unchanged
Direction	P2DIR	Read/write	02Ah	Reset
Interrupt Flags	P2IFG	Read/write	02Bh	Reset
Interrupt Edge Select	P2IES	Read/write	02Ch	Unchanged
Interrupt Enable	P2IE	Read/write	02Dh	Reset
Function Select	P2SEL	Read/write	02Eh	Reset

These registers contain eight bits, and should be accessed using byte instructions in absolute-address mode.

8.3.1.1 Input Registers P1IN, P2IN

Both Input registers are read-only registers that reflect the signals at the I/O pins.

Note: Writing to Read-Only Registers P1IN, P2IN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

8.3.1.2 Output Registers P1OUT, P2OUT

Each output register shows the information of the output buffer. The output buffer can be modified by all instructions that write to a destination. If read, the

contents of the output buffer are independent of pin direction. A direction change does not modify the output buffer contents.

8.3.1.3 Direction Registers P1DIR, P2DIR

The direction registers contain eight independent bits that define the direction of the I/O pin. All bits are reset by the PUC signal.

When:

Bit = 0: The port pin is switched to input direction (3-state)

Bit = 1: The port pin is switched to output direction

8.3.1.4 Interrupt Flags P1IFG, P2IFG

Each interrupt flag register contains eight flags that reflect whether or not an interrupt is pending for the corresponding I/O pin, if the I/O is interrupt-enabled.

When:

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending due to a transition at the I/O pin or from software setting the bit.

Note:

Manipulating P1OUT and P1DIR, as well as P2OUT and P2DIR, can result in setting the P1IFG or P2IFG bits.

Writing a zero to an interrupt flag resets it; writing a one to an interrupt flag sets it and generates an interrupt.

Each group of interrupt flags P1FLG.0 to P1FLG.7 and P2FLG.0 to P2FLG.7 sources its own interrupt vector. Interrupt flags P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7 are not reset automatically when an interrupt from these events is serviced. The software should determine the origin of the interrupt and reset the appropriate flag(s).

Note:

Any external interrupt event should be at least 1.5 times MCLK or longer, to ensure that it is accepted and the corresponding interrupt flag is set.

8.3.1.5 Interrupt Edge Select P1IES, P2IES

Each interrupt edge select register contains a bit for each corresponding I/O pin to select what type of transition triggers the interrupt flag.

When:

Bit = 0: The interrupt flag is set with a low-to-high transition

Bit = 1: The interrupt flag is set with a high-to-low transition

Note:

Changing the P1IES and P2IES bits can result in setting the associated interrupt flags.

PnIES.x	PnIN.x	PnIFG.x
0 → 1	0	Unchanged
0 → 1	1	May be set
1 → 0	0	May be set
1 → 0	1	Unchanged

8.3.1.6 Interrupt Enable P1IE, P2IE

Each interrupt enable register contains bits to enable the interrupt flag for each I/O pin in the port. Each of the sixteen bits corresponding to pins P1.0 to P1.7 and P2.0 to P2.7 is located in the P1IE and P2IE registers.

When:

Bit = 0: The interrupt request is disabled

Bit = 1: The interrupt request is enabled

Note: Port P1, Port P2 Interrupt Sensitivity

Only transitions, not static levels, cause interrupts.

If an interrupt flag is still set when the RETI instruction is executed (for example, a transition occurs during the interrupt service routine), an interrupt occurs again after RETI is completed. This ensures that each transition is acknowledged by the software.

8.3.1.7 Function Select Registers P1SEL, P2SEL

P1 and P2 port pins are often multiplexed with other peripheral modules to reduce overall pin count on MSP430 devices (see the specific device data sheet to determine which other peripherals also use the device pins). Control registers P1SEL and P2SEL are used to select the desired pin function—I/O port or other peripheral module. Each register contains eight bits corresponding to each pin, and each pin's function is individually selectable. All bits in these registers are reset by the PUC signal. The bit definitions are:

Bit = 0: Port P1 or P2 function is selected for the pin

Bit = 1: Other peripheral module function is selected for the pin

Note: Function Select With P1SEL, P2SEL

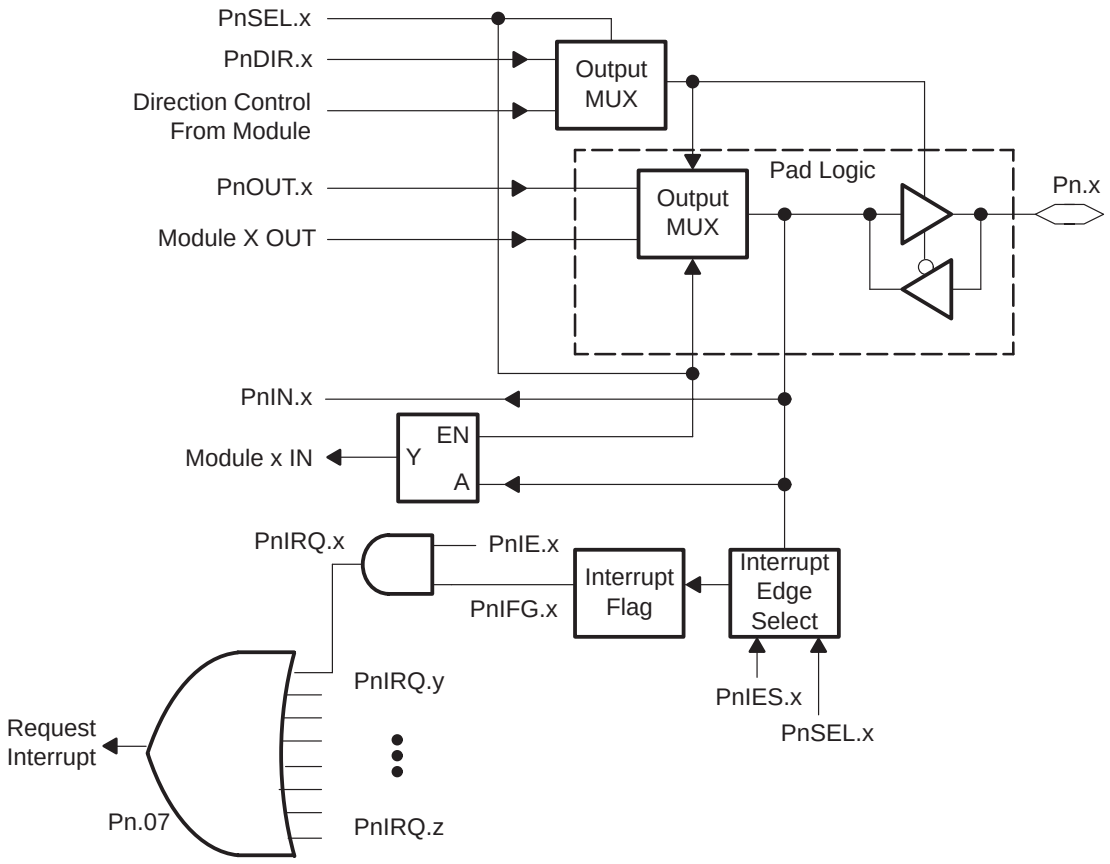
The interrupt-edge-select circuitry is disabled if control bit PnSEL.x is set. Therefore, the input signal can no longer generate an interrupt.

When a port pin is selected to be used as an input to a peripheral module other than the I/O port (PnSEL.x = 1), the actual input signal to the peripheral module is a latched representation of the signal at the device pin (see Figure 8–9 schematic). The latch uses the PnSEL.x bit as its enable, so while PnSEL.x=1, the internal input signal simply follows the signal at the pin. However, if the PnSEL.x bit is reset, then the output of the latch (and therefore the input to the other peripheral module) represents the value of the signal at the device pin just prior to the bit being reset.

8.3.2 Port P1, Port P2 Schematic

The pin logic of each individual port P1 and port P2 signal is identical. Each bit can be read and written to as shown in Figure 8–9.

Figure 8–9. Schematic of One Bit in Port P1, P2



x = 0 to 7, according to bits 0 to 7
n = 1 for Port P1 and 2 for Port P2

8.3.3 Port P1, P2 Interrupt Control Functions

Ports P1 and P2 use eight bits for interrupt flags, eight bits to enable interrupts, eight bits to select the effective edge of an interrupt event, one interrupt vector address for port P1, and one interrupt vector address for port P2.

Each signal uses three bits for configuration and interrupt:

- Interrupt flag, P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7
- Interrupt enable bit, P1IE.0 to P1IE.7 and P2IE.0 to P2IE.7
- Interrupt edge select bit, P1IES.0 to P1IES.7 and P2IES.0 to P2IES.7

The interrupt flags P1IFG.0 to P1IFG.7 source one interrupt and P2IFG.0 to P2IFG.7 source one interrupt. Any interrupt event on one or more pins of P1.0 to P1.7 or P2.0 to P2.7 requests an interrupt when two conditions are met: the appropriate individual bit PnIE.x is set, and the GIE bit is set. Interrupt flags P1IFG.0 to P1IFG.7 or P2IFG.0 to P2IFG.7 are not automatically reset. The software of the interrupt service routine should handle the detection of the source, and reset the appropriate flag when it is serviced.

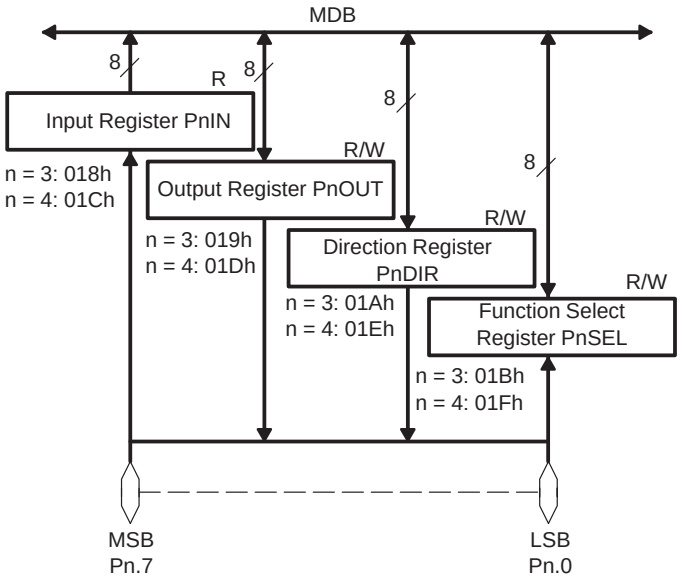
8.4 Ports P3, P4

General-purpose ports P3 and P4 function as shown in Figure 8–10. Each pin can be selected to operate with the I/O port function, or to be used with a different peripheral module. This multiplexing of pins allows for a reduced pin count on MSP430 devices.

Four registers control each of the ports (see Section 8.4.1).

Ports P3 and P4 are connected to the processor core through the 8-bit MDB and the MAB. They should be accessed with byte instructions using the absolute address mode.

Figure 8–10. Ports P3, P4 Configuration



8.4.1 Port P3, P4 Control Registers

The four control registers of each port give maximum configuration flexibility of digital I/O.

- All individual I/O bits are programmed independently
- Any combination of input is possible
- Any combination of port or module function is possible

The four registers for each port are shown in Table 8–4. They each contain eight bits and should be accessed with byte instructions.

Table 8–4. Port P3. P4 Registers

Register	Short Form	Address	Register Type	Initial State
Input	P3IN	018h	Read only	– – – – –
	P4IN	01Ch	Read only	– – – – –
Output	P3OUT	019h	Read/write	Unchanged
	P4OUT	01Dh	Read/write	Unchanged
Direction	P3DIR	01Ah	Read/write	Reset
	P4DIR	01Eh	Read/write	Reset
Port Select	P3SEL	01Bh	Read/write	Reset
	P4SEL	01Fh	Read/write	Reset

8.4.1.1 Input Registers

The input registers are read-only registers that reflect the signal at the I/O pins.

Note: Writing to Read-Only Register

Any attempt to write to these read-only registers results in an increased current consumption while the write attempt is active.

8.4.1.2 Output Registers

The output registers show the information of the output buffers. The output buffers can be modified by all instructions that write to a destination. If read, the contents of the output buffer are independent of the pin direction. A direction change does not modify the output buffer contents.

8.4.1.3 Direction Registers

The direction registers contain eight independent bits that define the direction of each I/O pin. All bits are reset by the PUC signal.

When:

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

8.4.1.4 Function Select Registers PnSEL

Ports P3, P4 pins are often multiplexed with other peripheral modules to reduce overall pin count on MSP430 devices (see the specific device data sheet to determine which other peripherals also use the device pins). Control registers PnSEL are used to select the desired pin function—I/O port or other peripheral module. Each register contains eight bits corresponding to each pin, and each pin's function is individually selectable. All bits in these registers are reset by the PUC signal. The bit definitions are:

Bit = 0: Port function is selected for the pin

Bit = 1: Other peripheral module function is selected for the pin

Note: Function Select With PnSEL Registers

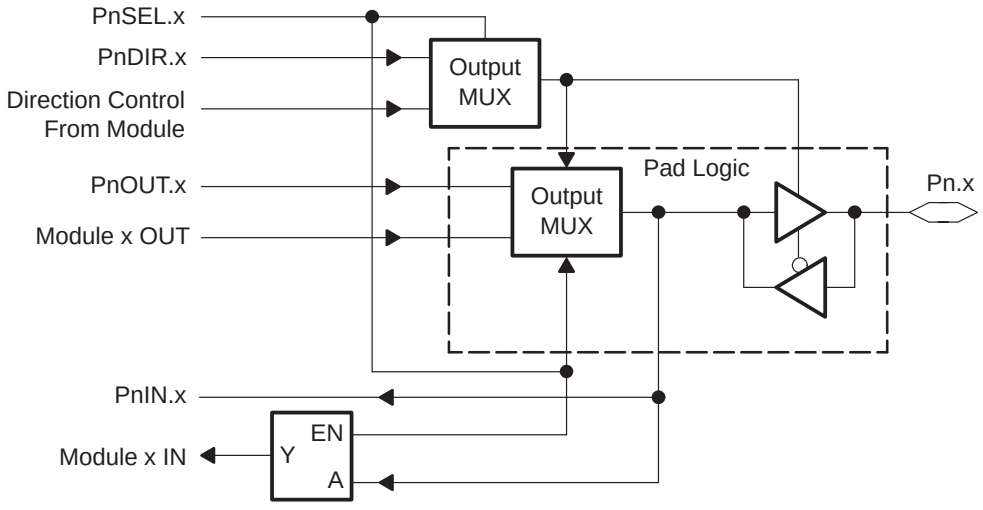
The interrupt-edge-select circuitry is disabled if control bit PnSEL.x is set. Therefore, the input signal can no longer generate an interrupt.

When a port pin is selected to be used as an input to a peripheral module other than the I/O port (PnSEL.x=1), the actual input signal to the peripheral module is a latched representation of the signal at the device pin (see Figure 8–11 schematic). The latch uses the PnSEL.x bit as its enable, so while PnSEL.x=1, the internal input signal simply follows the signal at the pin. However, if the PnSEL.x bit is reset, then the output of the latch (and therefore the input to the other peripheral module) represents the value of signal at the device pin, just prior to the bit being reset.

8.4.2 Port P3, P4 Schematic

The pin logic of each individual port signal is shown in Figure 8–11.

Figure 8–11. Schematic of Bits Pn.x



n = 3 for Port3, 4 for Port P4
x = 0 to 7, according to bits 0 to 7

Chapter 9

Universal Timer/Port Module

The Universal Timer/Port module supports the following system features:

- Up to six independent outputs
- Two 8-bit counters or one 16-bit counter
- A precision comparator for slope A/D conversion

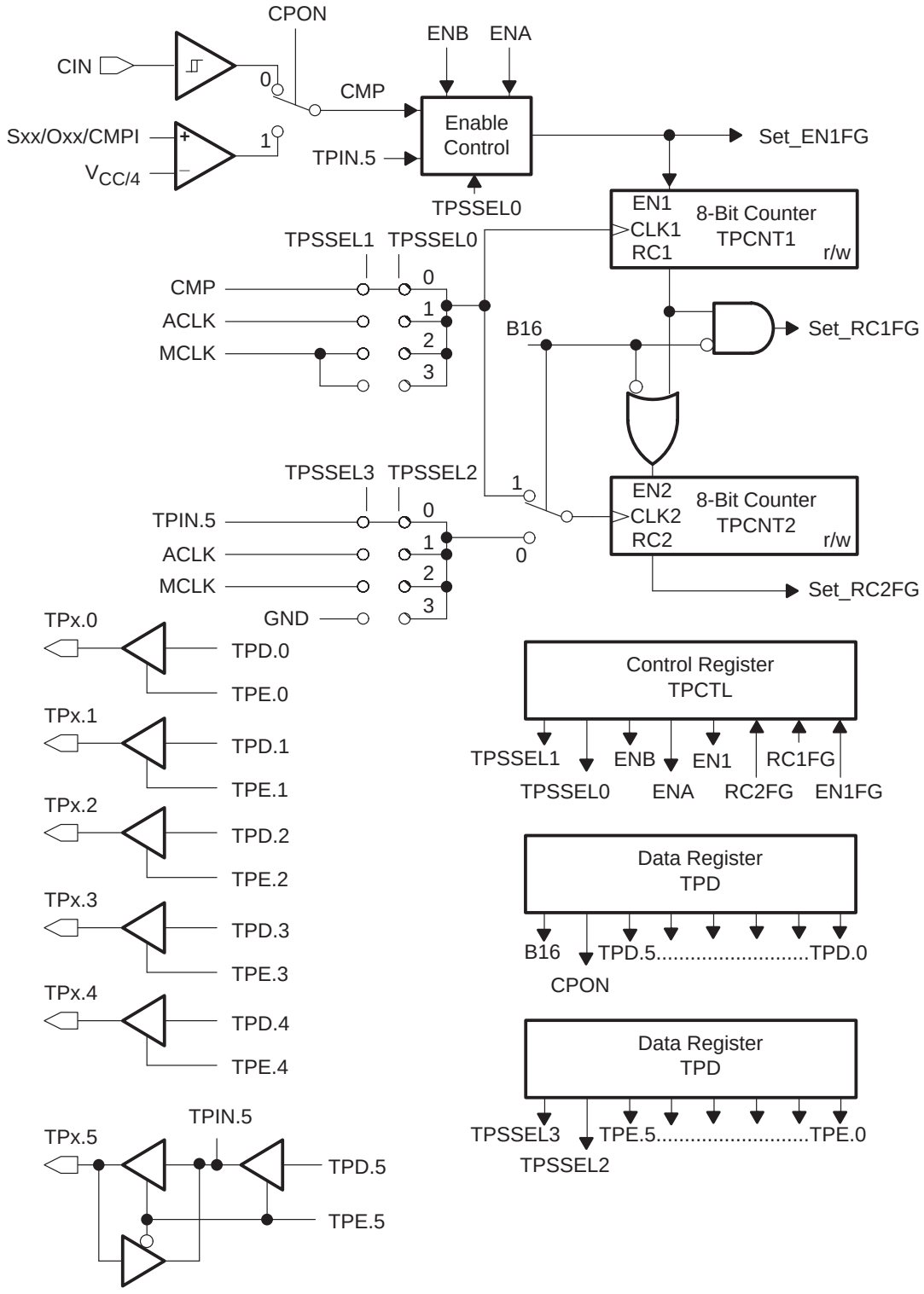
Topic **Page**

9.1	**Timer/Port Configuration**	**9-2**
9.2	**Timer/Port Module Operation**	**9-3**
9.3	**Timer/Port Registers**	**9-7**
9.4	**Timer/Port Interrupts**	**9-11**
9.5	**Timer/Port in an ADC Application**	**9-12**

9.1 Timer/Port Configuration

The Timer/Port is configured as shown in Figure 9–1.

Figure 9–1. Timer/Port Configuration



9.2 Timer/Port Module Operation

This section describes the Timer/Port counters.

9.2.1 Timer/Port Counter TPCNT1, 8-Bit Operation

Refer to Figure 9–1 for the following discussion.

The Universal Timer/Port offers much more application flexibility than other simple timer/counters by providing for flexible clocking and enable conditions.

The clock input to counter TPCNT1 can be selected from three different sources. MCLK, ACLK, or CMP (an external signal, or the comparator output) can be used to increment the timer/counter. The counter increments with each positive edge of the CLK1 clock input when enable signal EN1 is set. When the counter reaches full scale (0FFh), a ripple-carry signal RC1 goes high and remains high as long as the counter data equals 0FFh. When the counter increments from 0FFh to 000h, RC1 goes back low, but the negative edge of signal RC1 sets a ripple-carry flag bit in the TPCTL register (RC1FG) to indicate that the counter has rolled over. Setting the ripple carry flag RC1FG will generate a CPU interrupt if the Timer/Port interrupt enable flag (TPIE) is set. The RC1FG is not automatically reset, so it must be reset by the interrupt service routine (ISR).

The user has several choices to configure the enable signal EN1 (see Table 9–1). The counter is enabled when one or both ENA and ENB bits are set. Both of these bits are reset with a system reset (POR or PUC).

Further, an external event can be used to enable or disable the timer. When an external event on signals CMP or TPIN.5 disables the counter, flag EN1FG of the TPCTL register is set and a CPU interrupt is generated if the Timer/Port interrupt is enabled. The EN1FG flag is not automatically reset, so it must be reset by the ISR. Note that the EN1FG flag is not set if the counter is disabled through software manipulation of the ENA or ENB bits.

Any time the counter is disabled, the counter data is frozen, but the software can write a different value to the counter to change its data. Note that this write operation does not re-enable the counter.

The counter can be read or written to at anytime. A timer read can occur asynchronously to a timer increment if the clock source for the timer is either the ACLK or the CMP signal. In this situation the user software should perform several reads of the timer and take a majority vote to determine the correct timer value. When MCLK is selected as the clock source, the read is performed synchronously to the increment, so a majority vote software routine is not necessary.

Reading the timer/counter does not effect the count. The timer/counter will accurately increment with each clock regardless of when a read occurs. Also, performing a read of the counter directly after writing to it could result in reading different data than was written to it, depending on when the clock signal is applied.

9.2.2 Timer/Port Counter TPCNT2, 8-Bit Operation

Counter TPCNT2 operates similarly to TPCNT1, with a few differences in the enable signal and clock source.

The enable signal for TPCNT2 is primarily controlled with bit B16 of the TPD register. Bit B16 selects 8 or 16-bit mode for the Timer/Port. When B16 is reset, the Timer/Port is in 8-bit mode and counter TPCNT2 is always enabled.

Additionally, in 8-bit mode, counter TPCNT2 is completely independent from TPCNT1 and has a separate clock source. The clock source for TPCNT2 in 8–bit mode can be selected to be ACLK, MCLK, or the TPIN.5 pin.

Like TPCNT1, TPCNT2 has a ripple-carry output (RC2) that is high while the counter data is equal to 0FFh and the enable signal EN2 is high. When the counter increments from 0FFh to 000h, RC2 goes back low. The negative edge of RC2 sets a ripple-carry flag in the TPCTL register (RC2FG) to indicate that the counter has rolled over. Setting RC2FG generates a CPU interrupt if the Timer/Port interrupt is enabled. RC2FG is not automatically reset and should be reset by the ISR.

Any time the counter is disabled, the counter data is frozen, but the software can write a different value to the counter to change its data. Note that this write operation does not reenable the counter.

The counter can be read or written to at any time. A timer read can occur asynchronously to a timer increment if the clock source for the timer is either ACLK or the TPIN.5 signal. In this situation, the user software should perform several reads of the timer and take a majority vote to determine the correct timer value. When MCLK is selected as the clock source, the read is performed synchronously to the increment, so a majority vote software routine is not necessary.

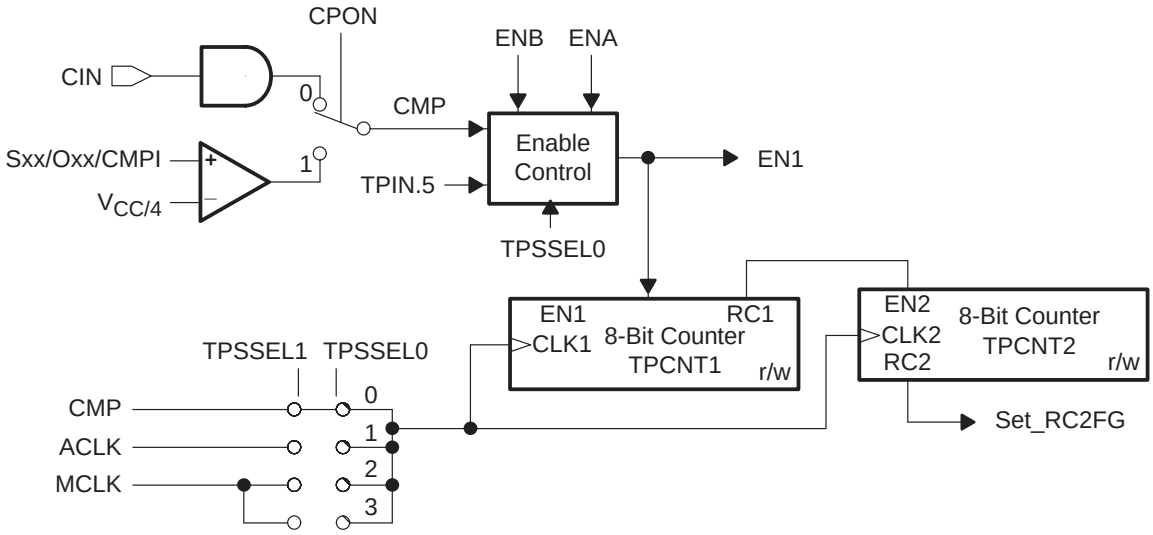
Reading the timer does not effect the count. The timer will accurately increment with each clock regardless of when a read occurs. Also, performing a read of the counter immediately after writing to it could result in reading different data than was written to it, depending on whether a clock signal was applied between the write and the read.

9.2.3 Timer/Port Counter, 16-Bit Operation

In 16-bit mode (B16 = 1), counters TPCNT1 and TPCNT2 are cascaded to form one 16-bit timer (see Figure 9–2). In this configuration, both counters operate from the same clock and the ripple-carry output of TPCNT1 serves as the enable for TPCNT2.

In 16-bit mode, clock source selection for the counter is made with the TPSSEL0 and TPSSEL1 bits, and TPSSEL2 and TPSSEL3 become *don't cares*. Clock source choices are the same as those for TPCNT1 in 8-bit mode: ACLK, MCLK, or CMP.

Figure 9–2. Timer/Port Counter, 16-Bit Operation



In 16-bit mode, the ripple carry signal is RC2 and is set when the counter value is equal to 0FFFFh. When the counter increments to 00000h, the negative edge of RC2 sets the RC2FG flag generating a CPU interrupt, and indicating that the counter has rolled over. The RC2FG flag must be reset by the ISR. RC1FG is not set in 16-bit mode – it remains unchanged.

Like in the 8-bit operation of TPCNT1, an external event can be used to enable or disable the timer when in 16-bit mode. When an external event on signal CMP or TPIN.5 disables the counter, flag EN1FG of the TPCTL register is set and a CPU interrupt is generated if the Timer/Port interrupt is enabled. The EN1FG flag is not automatically reset, so it must be reset by the ISR. Note that the EN1FG flag is not set if the counter is disabled through software manipulation of the ENA or ENB bits.

Read and write access to the Timer/Port is always done using byte instructions—even when the counter is configured in 16-bit mode. This requires special software considerations to access the counter while it is running to assure that the value read is correct. If a clock edge increments the counter between readings of the TPCNT1 and TPCNT2 values, the counter data will not be correct.

9.2.4 Enable Control

The signals ENA, ENB, TPSSEL0, and TPSSEL1 control the operation of the counter as described in Table 9–1. Therefore, the counter can be configured to run unconditionally, to run based on signals TPIN.5 or CMP, or to stop. Additionally, several clock choices are available within each operating mode.

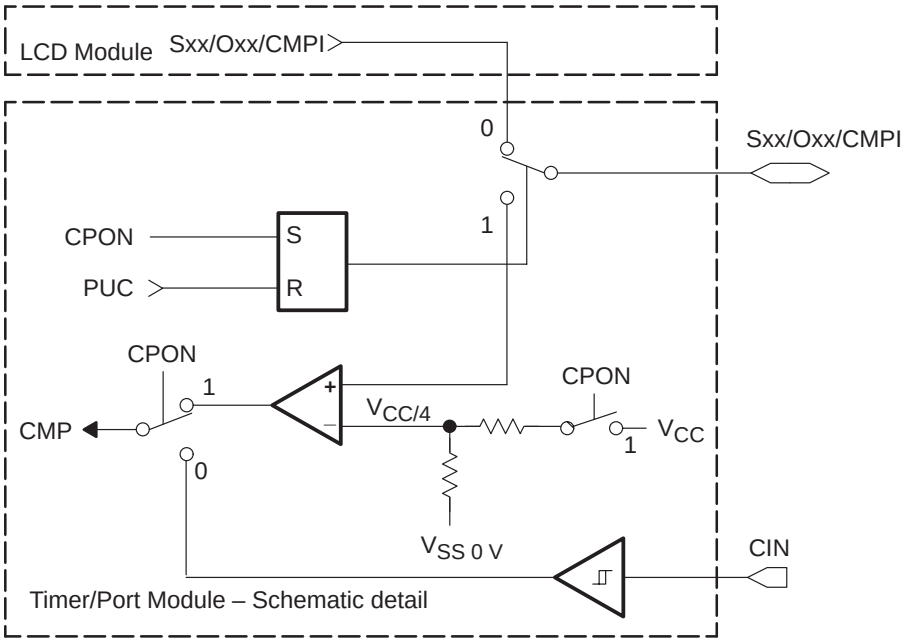
Table 9–1. Timer/Port Counter Signals, 16–Bit Operation

ENB	ENA	TPSSel1	TPSSel0	EN1	CLK1
0	0	0	0	0	CMP
0	0	0	1	0	ACLK
0	0	1	X	0	MCLK
0	1	0	0	1	CMP
0	1	0	1	1	ACLK
0	1	1	X	1	MCLK
1	0	0	0	$\overline{\text{TPIN.5}}$	CMP
1	0	0	1	TPIN.5	ACLK
1	0	1	0	$\overline{\text{TPIN.5}}$	MCLK
1	0	1	1	TPIN.5	MCLK
1	1	0	0	$\overline{\text{CMP}}$	CMP
1	1	0	1	CMP	ACLK
1	1	1	0	$\overline{\text{CMP}}$	MCLK
1	1	1	1	CMP	MCLK

9.2.5 Comparator Input

The comparator input is typically shared with one segment line as shown in Figure 9–3. The LCD segment function is selected for this pin after the PUC signal is active. The comparator input is selected when the CPON bit is set. Note that once selected, the comparator input can not be deselected without a PUC signal. See Chapter 3 for details on the PUC signal.

Figure 9–3. Timer/Port Comparator Input



9.3 Timer/Port Registers

The Timer/Port module registers listed in Table 9–2 are byte structured and must be accessed using byte instructions (suffix B).

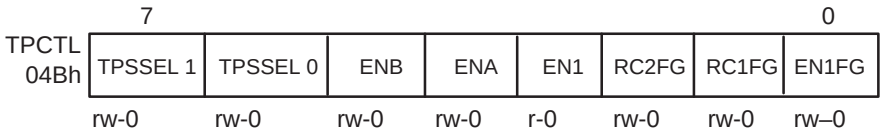
Table 9–2. Timer/Port Registers

Register	Short Form	Register Type	Address	Initial State
TP Control	TPCTL	Read/write	04Bh	Reset
TP Counter 1	TPCNT1	Read/write	04Ch	Unchanged
TP Counter 2	TPCNT2	Read/write	04Dh	Unchanged
TP Data	TPD	Read/write	04Eh	Reset
TP Data Enable	TPE	Read/write	04Fh	Reset

9.3.1 Timer/Port Control Register

The information stored in the control register (see Figure 9–4) determines the operation of the Timer/Port module.

Figure 9–4. Timer/Port Control Register



Bit 0: Enable flag EN1FG is set with the negative edge of enable signal EN1, if an event on CMP or TPIN.5 causes EN1 to go low. Note that EN1FG is not set if EN1 goes low as a result of software manipulation of ENA or ENB. EN1FG must be reset by software.

The EN1FG bit can be used during the Timer/Port interrupt service routine to determine if the interrupt event came from enable EN1 or from a ripple/carry.

Bit 1: In 8-bit mode, bit RC1FG indicates that counter TPCNT1 rolled from 0FFh to 0h (overflow condition). In 16-bit mode, RC1FG is not active. However, if software sets RC1FG, an interrupt request will be generated (if enabled), even if the counter is in 16-bit mode. RC1FG must be reset by software.

Bit 2: In 8-bit mode, bit RC2FG indicates that counter TPCNT2 rolled from 0FFh to 0h (overflow condition). In 16-bit mode, RC2FG indicates the 16-bit counter has rolled from 0FFFFh to 0000h. RC2FG must be reset by software.

Note: RC1FG and RC2FG When Software Disables the Counter

When the counter is disabled with software via bits ENA and ENB, flag RC1FG (8–bit mode), or flag RC2FG (16-bit mode) may or may not be set if the counter rolls over to zero at the same time.

Bit 3: Enable signal EN1. This bit represents the state of enable signal EN1 and can be read by software.

The signal at TPx.5 can be used in the module internally and can be read with bit EN1 when TPE.5 is reset.

Bits 4, 5: The value of enable signal EN1 is defined by bits ENA, ENB and TPSSEL0, as described in Table 9–3.

Table 9–3. Bit EN1 Level/Signal

ENB	ENA	TPSSel0	EN1
0	0	X	0
0	1	X	1
1	0	0	$\overline{\text{TPIN.5}}$
1	0	1	TPIN.5
1	1	0	$\overline{\text{CMP}}$
1	1	1	CMP

Bits 6, 7: The Timer/Port clock source-select bits TPSSEL0 and TPSSEL1 select the clock source for TPCNT1, as described in Table 9–4.

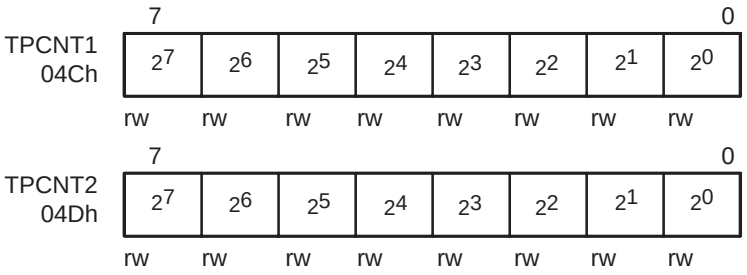
Table 9–4. Timer/Port Clock Source Selection

TPSSel1	TPSSel0	CLK1
0	0	CMP
0	1	ACLK
1	X	MCLK

9.3.1.1 Timer/Port Counter Registers TPCNT1 and TPCNT2

Both counter registers are read and written independently. The counter registers are shown in Figure 9–5.

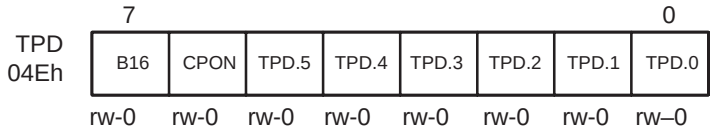
Figure 9–5. Timer/Port Counter Registers



9.3.1.2 Timer/Port Data Register

The data register holds the value of the six outputs, the 16-bit mode control bit, and the comparator control bit, as shown in Figure 9–5.

Figure 9–6. Timer/Port Data Register



Bits 0 to 5: Bits TPD.0 to TPD.5 hold the data for the output pins TPx.0 to TPx.5. The values are applied to these pins when the three-state output is enabled by TPE.0 to TPE.5. They are reset with a PUC.

Bit 6: The comparator CPON bit enables the comparator. It is reset with a PUC. Current consumption is reduced by disabling the comparator when not in use.

Bit 7: Control bit B16 selects 8- or 16-bit operation.

B16 = 0: 8-bit counter mode is selected. TPCNT1 and TPCNT2 are independent 8-bit counters.

B16 = 1: 16-bit counter mode is selected. TPCNT1 and TPCNT2 form one 16-bit counter.

9.3.1.3 Timer/Port Enable Register

The Timer/Port enable register contains the enable bits for the six outputs and two bits for clock source selection for TPCNT2.

Figure 9–7. Timer/Port Enable Register

Bits 0 to 5: Bits TPE.0 to TPE.5 are the enable bits for outputs TPx.0 to TPx.5. The bits are reset with a PUC, with the resulting outputs being in the high impedance state.

Note:

TPE.5 must be reset to use pin TPx.5 as an input.

Bits 6, 7: Timer/Port clock source-select bits TPSSEL2 and TPSSEL3 select the clock source for TPCNT2 when bit B16 is reset, as shown in Table 9–5. In 16-bit mode (B16 = 1) the clock source for counters TPCNT1 and TPCNT2 are identical and are selected by TPSSEL0 and TPSSEL1.

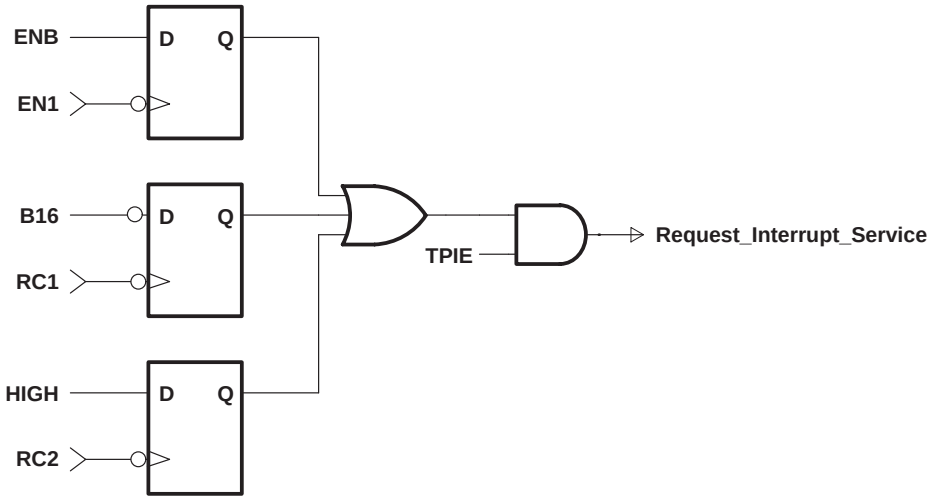
Table 9–5. Counter TPCNT2 Clock Sources

B16	TPSSel3	TPSSel2	CLK2
0	0	0	TPIN.5
0	0	1	ACLK
0	1	0	MCLK
0	1	1	MCLK
1	X	X	= CLK1

9.4 Timer/Port Interrupts

The Timer/Port has one interrupt vector sourced by up to three interrupt flags (RC1FG, RC2FG, and EN1FG), as shown in Figure 9–8. When in 8-bit mode, all three flags source the Timer/Port interrupt. When in 16-bit mode, only flags RC2FG and EN1FG source the interrupt. The Timer/Port interrupt service routine should check the flags to determine the source of a Timer/Port interrupt and handle it appropriately. All three flags must be reset by software. Note that even though RC1FG is inactive in 16-bit mode, an interrupt request will be generated (if enabled) when set by software.

Figure 9–8. Timer/Port Interrupt Scheme



The Timer/Port interrupt is enabled by the TPIE bit located in the SFR register IE2. The bit must be set to enable the Timer/Port interrupt. The initial state is reset. See chapter 3 for a discussion of the IEx registers.

Note:

When software is used to stop the counter via the ENA and ENB bits, flags RC1FG and RC2FG may be set (as appropriate, according to 8- or 16-bit mode) if the counter(s) roll over at the same time.

9.5 Timer/Port in an ADC Application

In addition to supporting a variety of counting and timing applications, the Universal Timer/Port also supports slope A/D conversion. Slope A/D conversion is extremely useful in sensor applications where the sensor is either resistive or capacitive.

In general, slope A/D conversion involves comparing the discharge times of two RC networks—one with a known time constant, and one with a sensor controlling the time constant. The value of the sensor can then be determined by a simple ratio of the discharge times.

For example, to use the Universal Timer/Port to measure a resistive sensor, one would first charge and discharge an RC network, made up of a known resistor value and a known capacitor value, while measuring the discharge time. Next, the known resistor would be replaced in the circuit by the unknown sensor and the charge/discharge cycle would be repeated, again measuring the discharge time. The value of the sensor could then be calculated by dividing the discharge times and multiplying by the known resistor value.

All of the required charging, discharging, timing, and switching of the resistors or capacitors can be done completely with the Universal Timer/Port, its high-impedance outputs, and its integrated comparator.

See the *MSP430 Application Report Book* and other application notes for details and circuit diagrams on using the Universal Timer/Port in slope A/D applications.

Application notes may be downloaded from www.ti.com/sc/msp430.

Chapter 10

Timers

The MSP430 microcontrollers offer a variety of very flexible timers that can be used to support a wide array of applications while also optimizing ultralow-power operation.

Topic **Page**

10.1 Basic Timer1 10-2

10.2 8-Bit Interval Timer/Counter 10-7

10.3 The Watchdog Timer 10-13

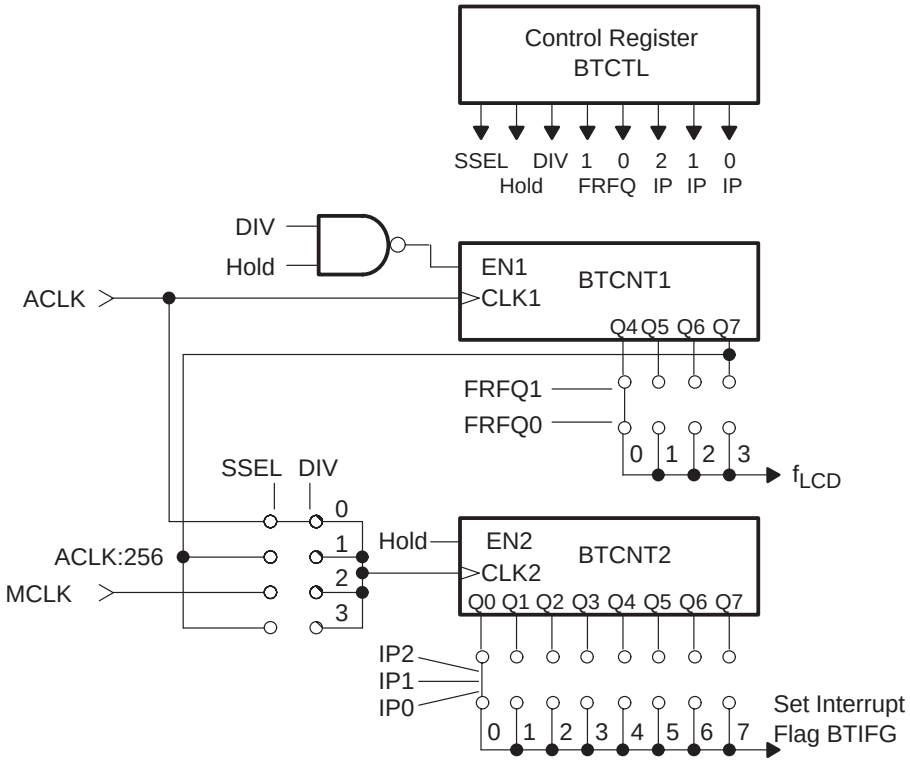
10.1 Basic Timer1

The Basic Timer1 (shown in Figure 10–1) supplies other peripheral modules or the software with low frequency control signals. The Basic Timer1 operation supports two independent 8-bit timing/counting functions, or one 16-bit timing/counting function.

Some uses for the Basic Timer1 include:

- ❏ Real-time clock (RTC)
- ❏ Debouncing keys (keyboard)
- ❏ Software time increments

Figure 10–1. Basic Timer1 Configuration



10.1.1 Basic Timer1 Registers

The Basic Timer1 register is byte structured, and should be accessed using byte processing instructions (suffix .B). Table 10–1 describes the Basic Timer1 registers.

Table 10–1. Basic Timer1 Registers

Register	Short Form	Register Type	Address	Initial State
BT Control	BTCTL	Read/write	040h	Unchanged
BT Counter 1	BTCNT1	Read/write	046h	Unchanged
BT Counter 2	BTCNT2	Read/write	047h	Unchanged

Note: The user's software should configure these registers at power-up, as there is no defined initial state.

10.1.1.1 Basic Timer1 Control Register

The information stored in the control register determines the operation of Basic Timer1. The state of the different bits selects the frequency source, the interrupt frequency, and the framing frequency of the LCD control circuitry as shown in Figure 10–2.

Figure 10–2. Basic Timer1 Control Register

BTCTL 040h

7							0
SSEL	Hold	DIV	FRFQ1	FRFQ0	IP2	IP1	IP0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 0 to 2: The three least-significant bits IP2 to IP0 determine the interrupt interval time. It is the interval of consecutive settings of the interrupt-request flag BTIFG, as illustrated in Figure 10–3.

Bits 3 to 4: The two bits FRFQ1 and FRFQ0 select the frequency f_{LCD} as described in Figure 10–3. Devices with the LCD peripheral on the chip use this frequency to generate the timing of the common and select lines.

Bit 5: See bit 7.

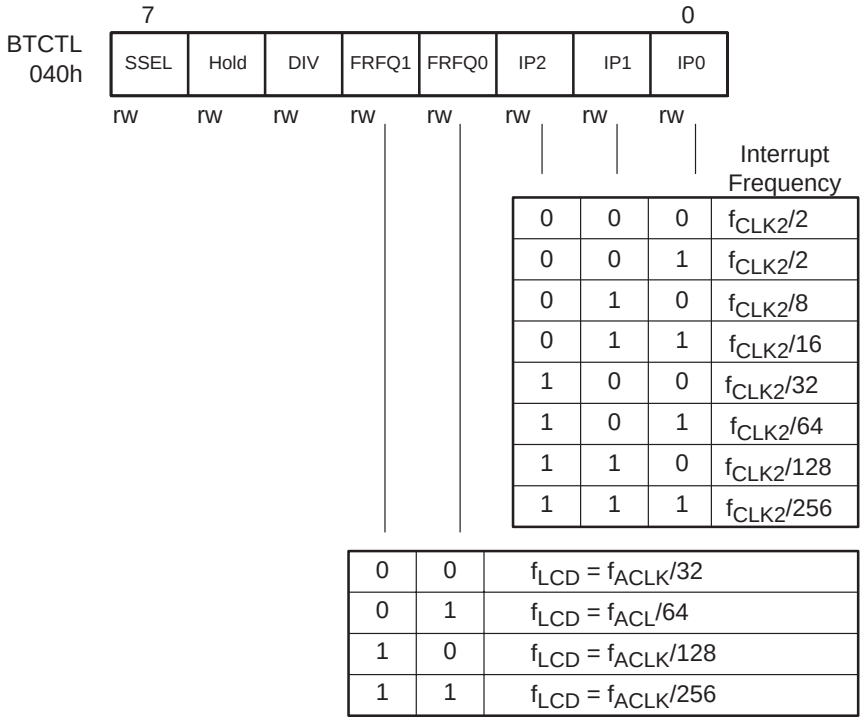
Bit 6: The hold bit stops the counter operation.
BTCNT2 is held if the hold bit is set.
BTCNT1 is held if the hold and DIV bits are set.

Bit 7: The SSEL and DIV bits select the frequency source for BTCNT2, as described in Table 10–2.

Table 10–2. BTCNT2 Input Frequency Sources

SSEL	DIV	CLK2
0	0	ACLK
0	1	ACLK/256
1	0	MCLK
1	1	ACLK/256

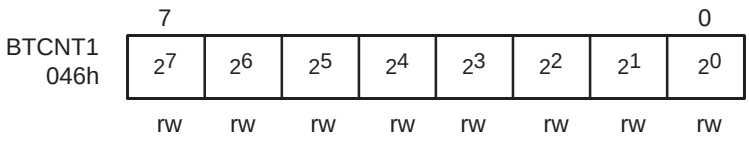
Figure 10–3. Basic Timer1 Control Register Function



10.1.1.2 Basic Timer1 Counter BTCNT1

The Basic Timer1 counter BTCNT1, shown in Figure 10–4 divides the auxiliary clock ACLK. The frame frequency for the LCD-drive is selected from four outputs of the counter's bits. The output of the most significant bit can be used for the clock input to the second counter BTCNT2. The value of bits Q0 to Q7 can be read, and the software can write to bits Q0 to Q7.

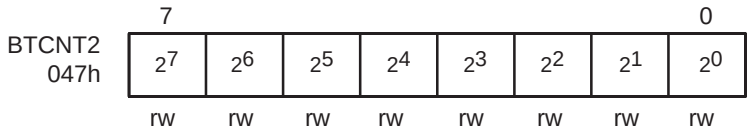
Figure 10–4. Basic Timer1 Counter BTCNT1



10.1.1.3 Basic Timer1 Counter BTCNT2

The Basic Timer1 counter BTCNT2, shown in Figure 10–5, divides the input-clock frequency. The input-clock source can be MCLK, ACLK, or ACLK/256. The interrupt period can be selected using IP0 to IP2, located in the Basic Timer1 control register BTCTL. It selects one of the eight bits of BTCNT2 as the source signal to set the Basic Timer1 interrupt flag BTIFG. The value of the counter bits can be read, as well as written.

Figure 10–5. Basic Timer1 Counter BTCNT2



10.1.2 Special Function Register Bits

Two SFR bits pertain to the Basic Timer1 Interrupt:

- Basic Timer1 interrupt flag (BTIFG) (located in IFG2.7)
- Basic Timer1 interrupt enable (BTIE) (located in IE2.7)

The BTIFG flag indicates that a Basic Timer1 interrupt is pending and is reset automatically when the interrupt is accepted.

The BTIE bit enables or disables the interrupt from the Basic Timer1 and is reset with a PUC. The Basic Timer1 interrupt is also enabled or disabled with the general interrupt enable bit, GIE.

10.1.3 Basic Timer1 Operation

The Basic Timer1 is constantly incremented by the selected clock source.

The hold bit inhibits all functions of the module and reduces power consumption. The Basic Timer1 registers may be accessed at any time, regardless of the state of the hold bit.

An interrupt can be used to control system operation. The interrupt is a single source interrupt.

The basic timer can operate in two different modes:

- Two independent 8-Bit Timer/Counters
- One 16-bit timer/counter

10.1.3.1 8-Bit Counter Mode

In the 8-Bit Timer/Counter mode, counter BTCNT1 is incremented constantly with ACLK. When reading the counters, the user should be aware that the counter clock and CPU clock may be asynchronous. Therefore, special software consideration may be required to assure a correct reading.

The BTCNT2 clock signal can be selected to be MCLK, ACLK, or ACLK/256 using the control signals SSEL and DIV. Counter BTCNT2 is incremented with the signal selected.

One of the eight counter outputs can be selected to set the Basic Timer1 interrupt flag. Read and write access can be asynchronous when ACLK or ACLK/256 is selected.

The hold bit stops all operations.

10.1.3.2 16-bit Counter Mode

The 16-bit timer/counter mode is selected when control bit DIV is set. In this mode, the clock source of counters BTCNT1 and BTCNT2 is the ACLK signal.

The hold bit stops all operations.

10.1.4 Basic Timer1 Operation: Signal f_{LCD}

The LCD controller uses the f_{LCD} signal from the Basic Timer1 to generate the timing for common and segment lines. The frequency of signal f_{LCD} is generated from ACLK. Using a 32,768-Hz crystal, the f_{LCD} frequency can be 1024 Hz, 512 Hz, 256 Hz, or 128 Hz. Bits FRFQ1 and FRFQ0 allow the correct selection of frame frequency. The proper frequency f_{LCD} depends on the LCD's requirement for framing frequency and LCD multiplex rate and is calculated by:

$$f_{LCD} = 2 \times \text{MUX rate} \times f_{Framing}$$

A 3 MUX example follows:

LCD data sheet: $f_{Framing}$ = 100 Hz .... 30 Hz

FRFQ: $f_{LCD} = 6 \times f_{Framing}$

$f_{LCD} = 6 \times 100$ Hz = 600 Hz ... 6×30 Hz = 180 Hz

Select f_{LCD}: 1024 Hz, 512 Hz, 256 Hz, or 128 Hz

f_{LCD} = 32,768/128 = 256 Hz FRFQ1 = 1; FRFQ0 = 0

See the LCD Driver chapter for more details on the LCD driver.

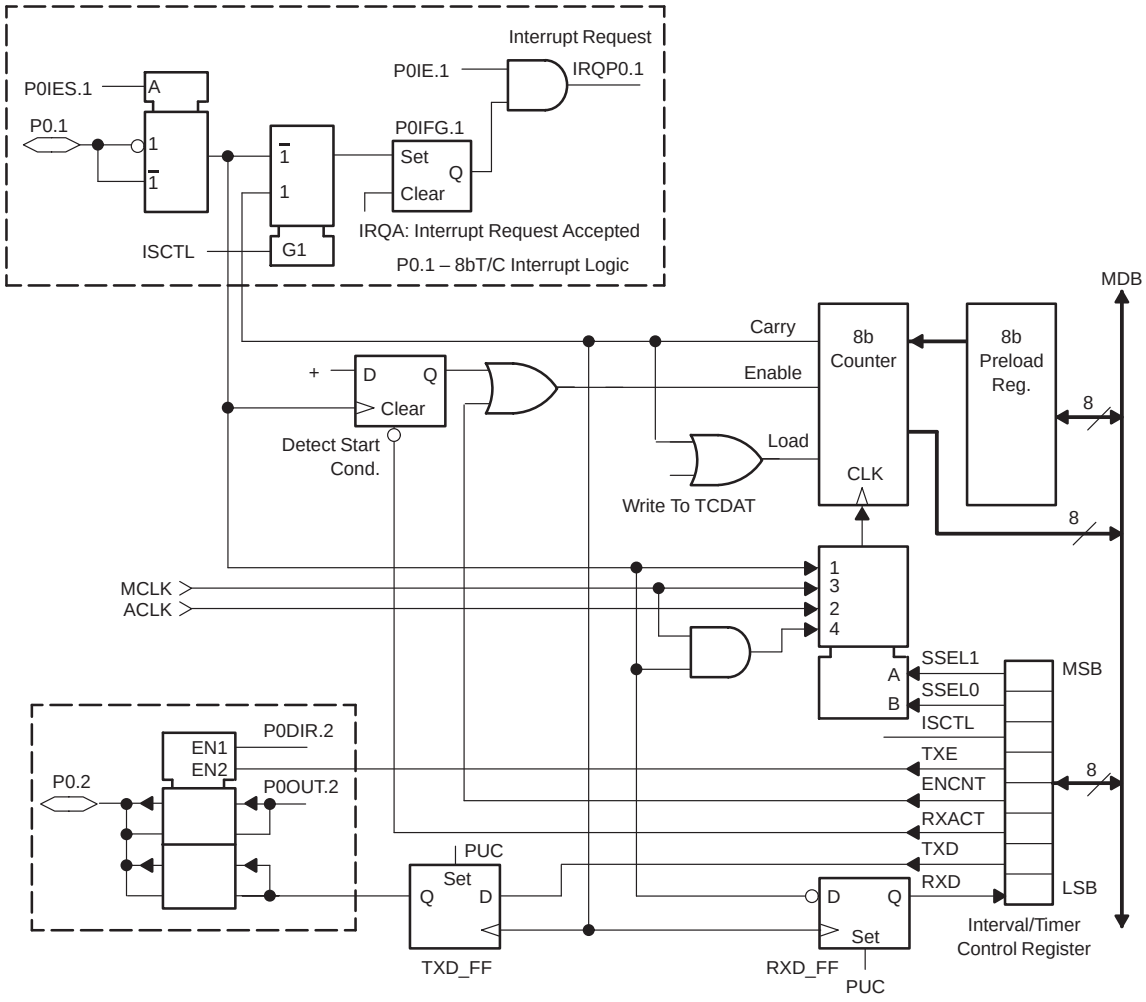
10.2 8-Bit Interval Timer/Counter

The 8-Bit Timer/Counter supports three major application functions:

- Serial communication or data exchange
- Pulse counting or pulse accumulation
- Timing

Figure 10–6 shows the 8-Bit Timer/Counter functions.

Figure 10–6. 8-Bit Timer/Counter



10.2.1 Operation of 8-Bit Timer/Counter

The 8-Bit Timer/Counter includes the following major blocks:

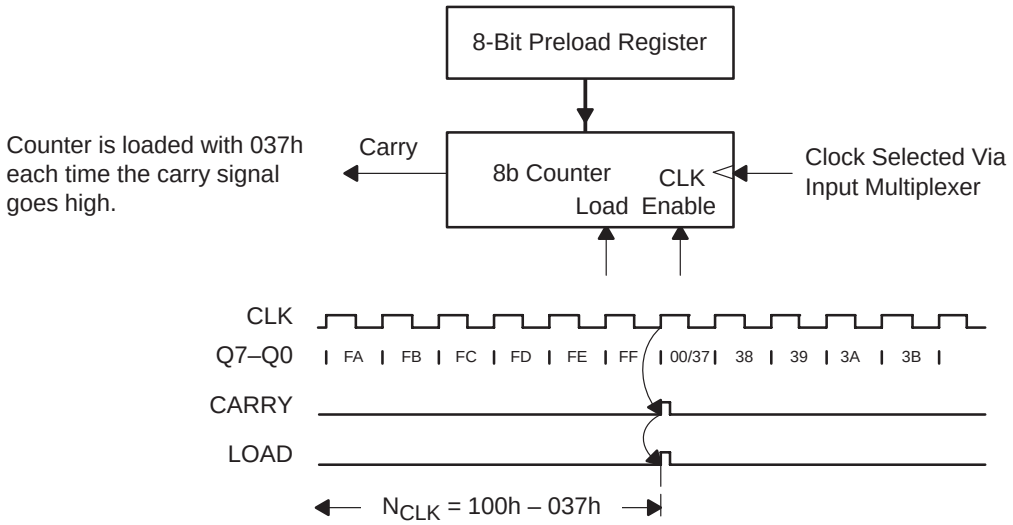
- ❑ 8-bit up-counter with a preload register
- ❑ 8-bit control register
- ❑ Input clock selector
- ❑ Edge detection, (for example, a *start bit* of asynchronous protocols)
- ❑ Input and output data latch, triggered by the carry-out signal from the 8-bit counter

10.2.1.1 8-Bit Timer/Counter With Preload Register

The 8-bit counter counts up with the selected input clock. Two counter inputs, load and enable, control the operation.

Figure 10–7 shows the 8-bit counter functions.

Figure 10–7. 8-Bit Counter Example



Either of two events controls the load function: a carry from the counter or a write access loads the counter with the data of the preload register. Note that writing to the counter (TCDAT register) loads the counter with the preload value, not the contents of the write instruction.

The software may write or read the preload register. The preload register acts as a buffer and can be written to immediately after the load of the counter is complete.

When the enable signal is set high, the counter counts up each time a positive-clock edge is applied to the counter's clock input.

10.2.1.2 8-Bit Control Register

The information stored in the 8-bit control register selects the operating mode of the timer/counter and controls the function.

10.2.1.3 Input Clock Selector

Two bits in the 8-bit control register select the source for the clock input of the 8-bit counter. The four sources are the system clock MCLK, the auxiliary clock ACLK, the external signal from pin P0.1, and the signal from the logical .AND. of MCLK and pin P0.1.

10.2.1.4 Edge Detection

Serial protocols such as UART need start-bit edge detection at the receiver to determine the start of data transmission. This edge detection is supported by the 8-Bit Timer/Counter and used to implement a UART with the timer.

10.2.1.5 Input and Output Data Latch, RXD_FF and TXD_FF

The clock used to latch data into the input and output data latches is the carry signal from the 8-bit counter. Both latches are used as single-bit buffers and change their outputs with the predefined timing.

10.2.2 8-Bit Timer/Counter Registers

The timer/counter registers, described in Table 10–3, are accessed using byte instructions.

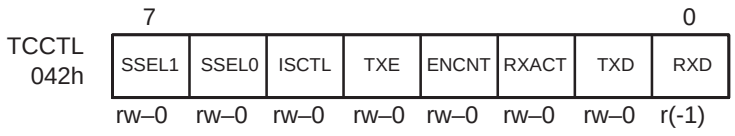
Table 10–3.8-Bit Timer/Counter Registers

Register	Short Form	Register Type	Address	Initial State
TC Control	TCCTL	Read/write	042h	Reset
Preload	TCPLD	Read/write	043h	Unchanged
Counter	TCDAT	Read/(write)	044h	Unchanged

10.2.2.1 8-Bit Timer/Counter Control Register

The information stored in the control register, as shown in Figure 10–8, determines the operation of the 8-Bit Timer/Counter.

Figure 10–8. 8-Bit Timer/Counter Control Register



Bit 0: Bit RXD is read only. The signal from external pin P0.1 is latched with the carry signal of the 8-bit counter.

Bit 1: Register bit TXD is buffered and clocked out with the carry signal from the 8-bit counter at pin P0.2 .

Bit 2: Bit RXACT controls the edge detect logic. The edge detect logic needs a reset ENCNT bit (bit 3) for correct counter-enable operation.

RXACT = 0: The edge-detect FF is cleared and it cannot be the source for enabling the counter operation.

RXACT = 1: The edge-detect FF is enabled. A positive or negative edge at pin P0.1, selected by P0IES.1, sets the FF, and the counter is prepared for count operation. Once the FF is set, it remains set until it is reset with RXACT = 0.

Bit 3: Bit ENCNT sets the counter-enable signal. The 8-bit counter increments its value with each rising edge of the clock input.

Together with bit RXACT (bit2, 0), this bit provides start/stop operation.

Bit 4: Signal TXE controls the three-state output buffer for the TXD bit:
TXE = 0: The direction control bit P0DIR.2 (see I/O chapter) determines if the buffer is active or in high-impedance state.
TXE = 1: Output buffer active (independent of the value of P0DIR.2)

Bit 5: Signal ISCTL controls the interrupt source between the I/O pin P0.1 and the carry signal of the 8-bit counter.
ISCTL = 0: The I/O pin P0.1 is the source of interrupt P0IFG.1.
ISCTL = 1: The carry signal from the 8-bit counter is the source of interrupt P0IFG.1.

Bits 6, 7: Bits SSEL0 and SSEL1 select the source of the clock input.

Table 10–4 describes the clock input source.

Table 10–4. Clock Input Source

SSEL1	SSEL0	Clock Source
0	0	Signal at pin P0.1 (according to P0IES.1)
1	0	MCLK
0	1	ACLK
1	1	Signal pin P0.1(according to P0IES.1) .AND. MCLK

10.2.2.2 8-Bit Timer/Counter Preload Register

The information stored in the preload register, not the data included with the instruction, is loaded into the 8-bit counter when a write access to the counter (TCDAT) is performed, as shown in the following code:

```
;=========   Definitions ================================
Dummy     .EQU  0     ; Value for dummy is not loaded into
                      ; counter
TCDAT     .EQU  044h  ; Address of 8-Bit Timer/Counter
;==Write pre-load register contents to 8-bit Timer/Counter=
;
          MOV.B #Dummy,&TCDAT
;
 The pre-load register (TCPLD) can be accessed using the
 address 043h.
```

10.2.2.3 8-Bit Counter Data

The data of the 8-bit counter can be read using address 044h. Writing to the counter loads the contents of the preload register—not the data included with the instruction.

10.2.3 Special Function Register Bits, 8-Bit Timer/Counter Related

The 8-Bit Timer/Counter has no individual interrupt bits; it shares the interrupt bits with port P0. Bit ISCTL, in control register TCCTL, selects the interrupt source for the interrupt flag.

The port0 signal P0.1/RXD, or the carry signal of the 8-bit counter is used for the interrupt source. One SFR bit and one port P0 bit configure the interrupt events on P0.1/RXD.1 as follows:

- ❏ P0.1/RXD interrupt enable P0IE.1 (located in IE1.3, initial state is reset)
- ❏ P0.1/RXD interrupt edge select P0IES.1 (located in P0IES, initial state is reset)

The interrupt flag is a single-source flag that automatically resets when the processor system services the interrupt. The enable bit and edge select bit remain unchanged.

10.2.4 Implementing a UART With the 8-Bit Timer/Counter

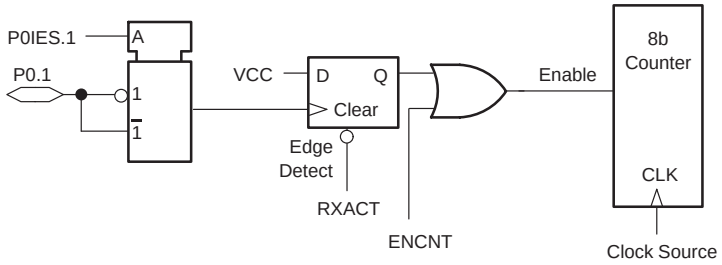
The 8-Bit Timer/Counter is uniquely capable of implementing a UART function, with the following features:

- ❏ Automatic start-bit detection – even from all ultralow-power modes
- ❏ Hardware baud-rate generation
- ❏ Hardware latching of RXD and TXD data
- ❏ Baud rates of 75 to 115,200 baud

This UART implementation is different from other microcontroller implementations where a UART may be implemented with general-purpose I/O and manual bit manipulation via software polling. Those implementations require great CPU overhead and therefore increase power consumption and decrease the usability of the CPU.

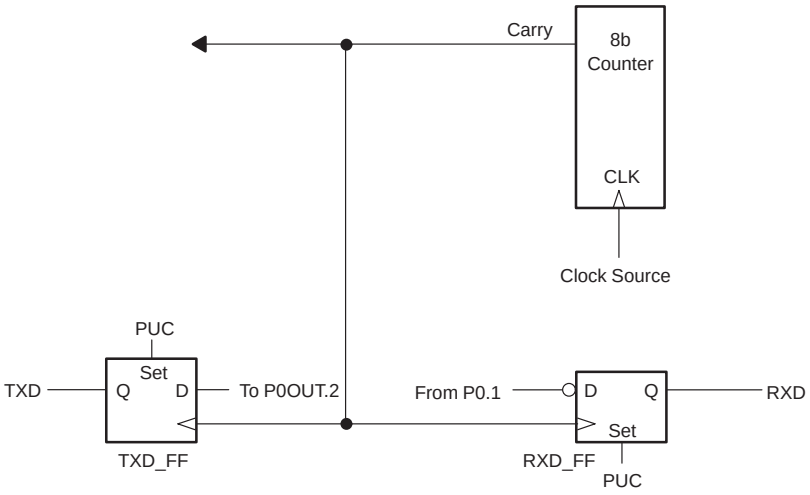
In this particular implementation, the 8-Bit Timer/Counter is configured as the baud clock and waits for the start bit. With the falling edge of the start bit, the counter begins counting (see Figure 10–9).

Figure 10–9. Start Bit Detection



Note that no CPU overhead is required for the start-bit detection. Start-bit detection is automatic and occurs if the processor is in active mode, or low power modes 0–4. When the counter reaches full-scale, the TXD and RXD data is automatically latched, the baud rate is automatically preloaded into the counter, the counter automatically begins counting, and an interrupt is generated for the CPU to retrieve the RXD data or write the next TXD data. Software overhead is only required to read and write the RXD and TXD data. (see Figure 10–10).

Figure 10–10. Data Latching

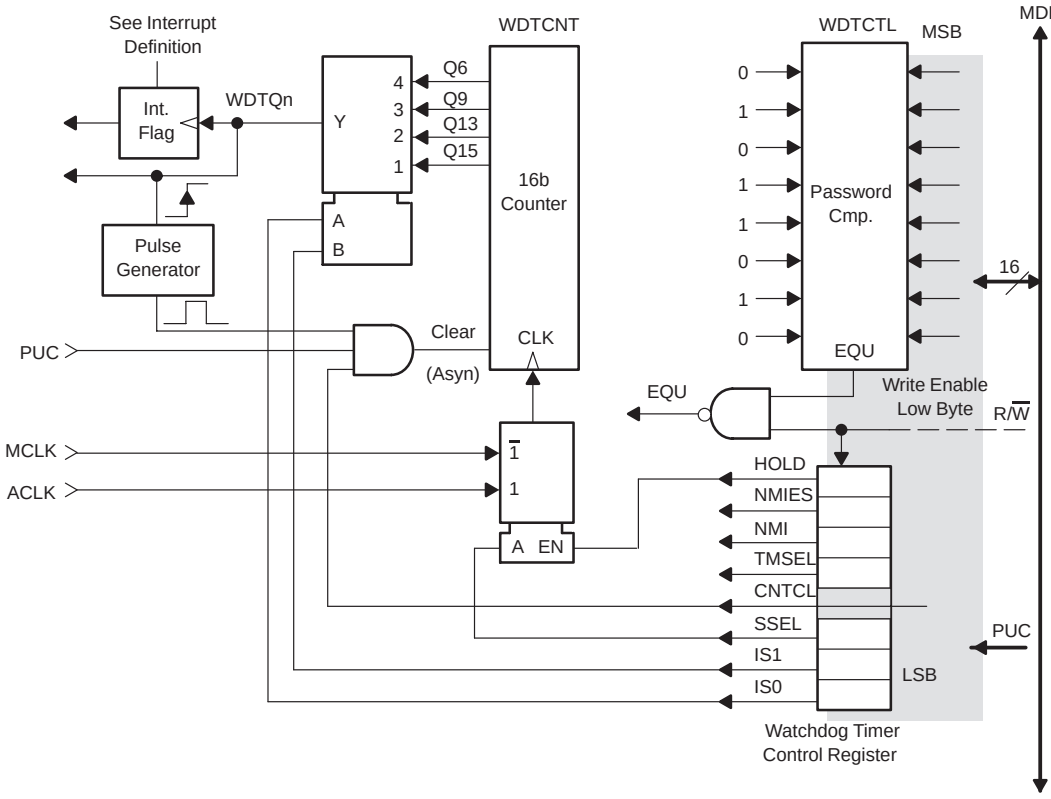


A complete application note including connection diagrams and complete software listing, may be found at www.ti.com/sc/msp430.

10.3 The Watchdog Timer

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval. The WDT diagram is shown in Figure 10–11.

Figure 10–11. Schematic of Watchdog Timer



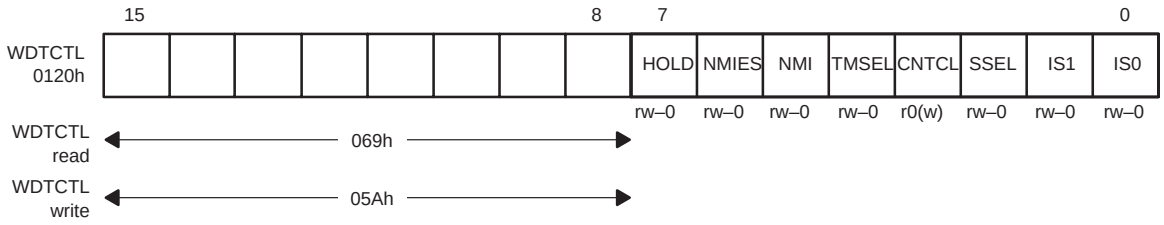
Some features of the Watchdog Timer include:

- Eight software-selectable time intervals
- Two operating modes: as watchdog or interval timer
- Expiration of the time interval in watchdog mode, which generates a system reset; or in timer mode, which generates an interrupt request
- Safeguards which ensure that writing to the WDT control register is only possible using a password
- Support of ultralow-power using the hold mode

10.3.1 Watchdog Timer Register

The watchdog-timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled through the watchdog-timer control register (WDTCTL), shown in Figure 10–12, which is a 16-bit read/write register located at the low byte of word address 0120h. Any read or write access must be done using word instructions with no suffix or *.w* suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCTL using the correct password.

Figure 10–12. Watchdog Timer Control Register



Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in Table 10–5. Assuming $f_{crystal}$ = 32,768 Hz and f_{System} = 1 MHz, the following intervals are possible:

Table 10–5. WDTCNT Taps

SSEL	IS1	IS0	Interval [ms]	
0	1	1	0.064	$t_{MCLK} \times 2^6$
0	1	0	0.5	$t_{MCLK} \times 2^9$
1	1	1	1.9	$t_{ACLK} \times 2^6$
0	0	1	8	$t_{MCLK} \times 2^{13}$
1	1	0	16.0	$t_{ACLK} \times 2^9$
0	0	0	32	$t_{MCLK} \times 2^{15}$ <– Value after PUC (reset)
1	0	1	250	$t_{ACLK} \times 2^{13}$
1	0	0	1000	$t_{ACLK} \times 2^{15}$

Bit 2: The SSEL bit selects the clock source for WDTCNT.
SSEL = 0: WDTCNT is clocked by MCLK.
SSEL = 1: WDTCNT is clocked by ACLK.

Bit 3: Counter clear bit. In both operating modes, writing a 1 to this bit restarts the WDTCNT at 00000h. The value read is not defined.

Bit 4: The TMSEL bit selects the operating mode: watchdog or timer.
TMSEL = 0: Watchdog mode
TMSEL = 1: Interval-timer mode

Bit 5: The NMI bit selects the function of the RST/NMI input pin. It is cleared by the PUC signal.
- NMI = 0: The $\overline{\text{RST}}$/NMI input works as reset input. As long as the $\overline{\text{RST}}$/NMI pin is held low, the internal signal is active (level sensitive).
- NMI = 1: The $\overline{\text{RST}}$/NMI input works as an edge-sensitive non-maskable interrupt input.

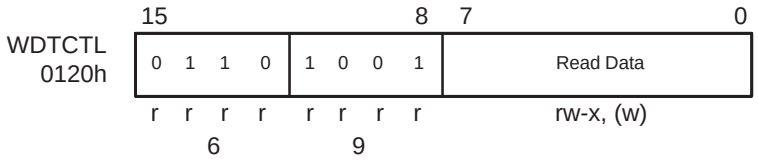
Bit 6: If the NMI function is selected, this bit selects the activating edge of the $\overline{\text{RST}}$/NMI input. It is cleared by the PUC signal.
- NMIES = 0: A rising edge triggers an NMI interrupt.
- NMIES = 1: A falling edge triggers an NMI interrupt.
- CAUTION: Changing the NMIES bit with software can generate an NMI interrupt.

Bit 7: This bit stops the operation of the watchdog counter. The clock multiplexer is disabled and the counter stops incrementing. It holds the last value until the hold bit is reset and the operation continues. It is cleared by the PUC signal.
- HOLD = 0: The WDT is fully active.
- HOLD = 1: The clock multiplexer and counter are stopped.

10.3.1.1 Accessing the WDTCTL (Watchdog Timer Control Register)

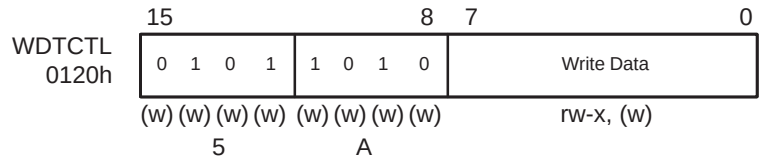
The WDTCTL register can be read or written to. As illustrated in Figure 10–13, WDTCTL can be read without the use of a password. A read access is performed by accessing word address 0120h. The low byte contains the value of WDTCTL. The value of the high byte is always read as 069h.

Figure 10–13. Reading WDTCTL



Write access to WDTCTL, illustrated in Figure 10–14, is only possible using the correct high-byte password. To change register WDTCTL, write to word address 0120h. The low byte contains the data to write to WDTCTL. The high byte is the password, which is 05Ah. A system reset (PUC) is generated if any value other than 05Ah is written to the high byte of address 0120h.

Figure 10–14. Writing to WDTCTL



10.3.2 Watchdog Timer Interrupt Control Functions

The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control.

- The WDT interrupt flag (WDTIFG) (located in IFG1.0, initial state is reset)
- The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

When using the watchdog mode, the WDTIFG flag is used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the Watchdog Timer initiated the reset condition (either by timing out or by a security key violation). If the flag is cleared, then the PUC was caused by a different source. See chapter 3 for more details on the PUC and POR signals.

When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is set after the selected time interval and a watchdog interval-timer interrupt is requested. The interrupt vector address in interval-timer mode is different from that in watchdog mode. In interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.

The WDTIE bit is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. Also, the GIE bit enables or disables the interrupt from the Watchdog Timer when it is being used in interval-timer mode.

10.3.3 Watchdog Timer Operation

The WDT module can be configured in two modes: watchdog and the interval-timer modes.

10.3.3.1 Watchdog Mode

When the WDT is configured to operate in watchdog mode, both a watchdog overflow and a security violation trigger the PUC signal, which automatically clears the appropriate system register bits. This results in a system configuration for the WDTCTL bits where the WDT is set into the watchdog mode and the $\overline{\text{RST}}$/NMI pin is switched to the reset configuration.

After a power-on reset or a system reset, the WDT module automatically enters the watchdog mode and all bits in the WDTCTL register and the watchdog counter (WDTCNT) are cleared. The initial conditions at register WDTCTL cause the WDT to start running at a relatively-low frequency, due to the range of the digitally-controlled oscillator (DCO) automatically being set in these situations. Since the WDTCNT is reset, the user software has ample time to set up or halt the WDT and to adjust the system frequency.

When the module is used in watchdog mode, the software should periodically reset the WDTCNT by writing a 1 to bit CNTCL of WDTCTL to prevent expiration of the selected time interval. If a software problem occurs and the time interval expires because the counter is no longer being reset, a system reset is generated and a system PUC signal is activated. The system restarts at the same program address that follows a power up. The cause of reset can be determined by testing bit 0 of interrupt flag register 1 in the SFRs. The appropriate time interval is selected by setting bits SSEL, IS0, and IS1 accordingly.

10.3.3.2 Timer Mode

Setting WDTCTL register bit TMSEL to 1 selects the timer mode. This mode provides periodic interrupts at the selected time interval. A time interval can also be initiated by writing a 1 to bit CNTCL in the WDTCTL register.

When the WDT is configured to operate in timer mode, the WDTIFG flag is set after the selected time interval, and it requests a standard interrupt service. The WDT interrupt flag is a single-source interrupt flag and is automatically reset when it is serviced. The enable bit remains unchanged. In interval-timer mode, the WDT interrupt-enable bit and the GIE bit must be set to allow the WDT to request an interrupt. The interrupt vector address in timer mode is different from that in watchdog mode.

Note: Watchdog Timer, Changing the Time Interval

Changing the time interval without clearing the WDTCNT may result in an unexpected and immediate system reset or interrupt. The time interval must be changed together with a counter-clear command using a single instruction (for example, MOV #05A0Ah,&WDTCTL).

Changing the clock source during normal operation may result in an incorrect interval. The timer should be halted before changing the clock source.

10.3.3.3 Operation in Low-Power Modes

The MSP430 devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the user's application and the type of clocking circuit on the MSP430 device determine how the Watchdog Timer and clocking signals should be configured. Review the clock-system chapter to determine the clocking circuit, clock signals, and low-power modes available. For example, the WDT should not be configured in watchdog mode with MCLK as its clock source if the user wants to use low-power mode 3 because MCLK is not active in LPM3, therefore the WDT would not function properly.

The WDT hold condition can also be used to support low power operation. The hold condition can be used in conjunction with low-power modes when needed.

10.3.3.4 Software Example

The following example illustrates the watchdog-reset operation.

```
; After RESET or power–up, the WDTCTL register and WDTCNT
; are cleared and the initial operating conditions are
; watchdog mode with a time interval of 32 ms.
;
;Constant definitions:
;
  WDTCTL  .EQU  0120h  ; Address of Watchdog Timer
  WDTPW   .EQU  05A00h; Password
  T250MS  .EQU  5      ; SSEL, IS0, IS1 set to 250 ms
  T05MS   .EQU  2      ; SSEL, IS0, IS1 set to 0.5 ms
  CNTCL   .EQU  8      ; Bit position to reset WDTCNT
  TMSEL   .EQU  010h   ; Bit position to select timer mode
;
; As long as watchdog mode is selected, watchdog reset has
; to be done periodically through an instruction e.g.:
;
   ........
   ........
   MOV   #WDTPW+CNTCL,&WDTCTL
;
; To change to timer mode and a time interval of 250 ms,
; the following instruction sequence can be used:
;
   MOV   #WDTPW+CNTCL+TMSEL+T250MS,&WDTCTL
            ; Clear WDTCNT and
            ; select 250 ms and timer
            ; mode
   ........
   ........
; Note: The time interval and clear of WDTCNT should be
;         modified within one instruction to avoid
;         unexpected reset or interrupt
;
; Switching back to watchdog mode and a time interval of
; 0.5 ms is performed by:
;
   ........
   ........
   MOV   #WDTPW+CNTCL,&WDTCTL  ; Reset WDT counter
;
   MOV   #WDTPW+T05MS,&WDTCTL  ; Select watchdog mode
                               ; and 0.5 ms
   ........
```

Chapter 11

Timer_A

This section describes the basic functions of the MSP430 general-purpose 16-bit Timer_A.

Note:

Throughout this chapter, the word *count* is used in the text. As used in these instances, it refers to the literal act of counting. It means that the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then the associated action will not take place. For example, the CCR0 interrupt flag is set when the timer *counts up to* the value in CCR0. The counter *must count* from CCR0–1 to CCR0. If the CCR0 value were simply written directly to the timer with software, the interrupt flag would *not* be set, even though the values in the timer and the CCR0 registers are the same.

Topic **Page**

11.1 Introduction **11-2**
11.2 Timer_A Operation **11-4**
11.3 Timer Modes **11-6**
11.4 Capture/Compare Blocks **11-13**
11.5 The Output Unit **11-19**
11.6 Timer_A Registers **11-25**
11.7 Timer_A UART **11-34**

11.1 Introduction

Timer_A is an extremely versatile timer made up of :

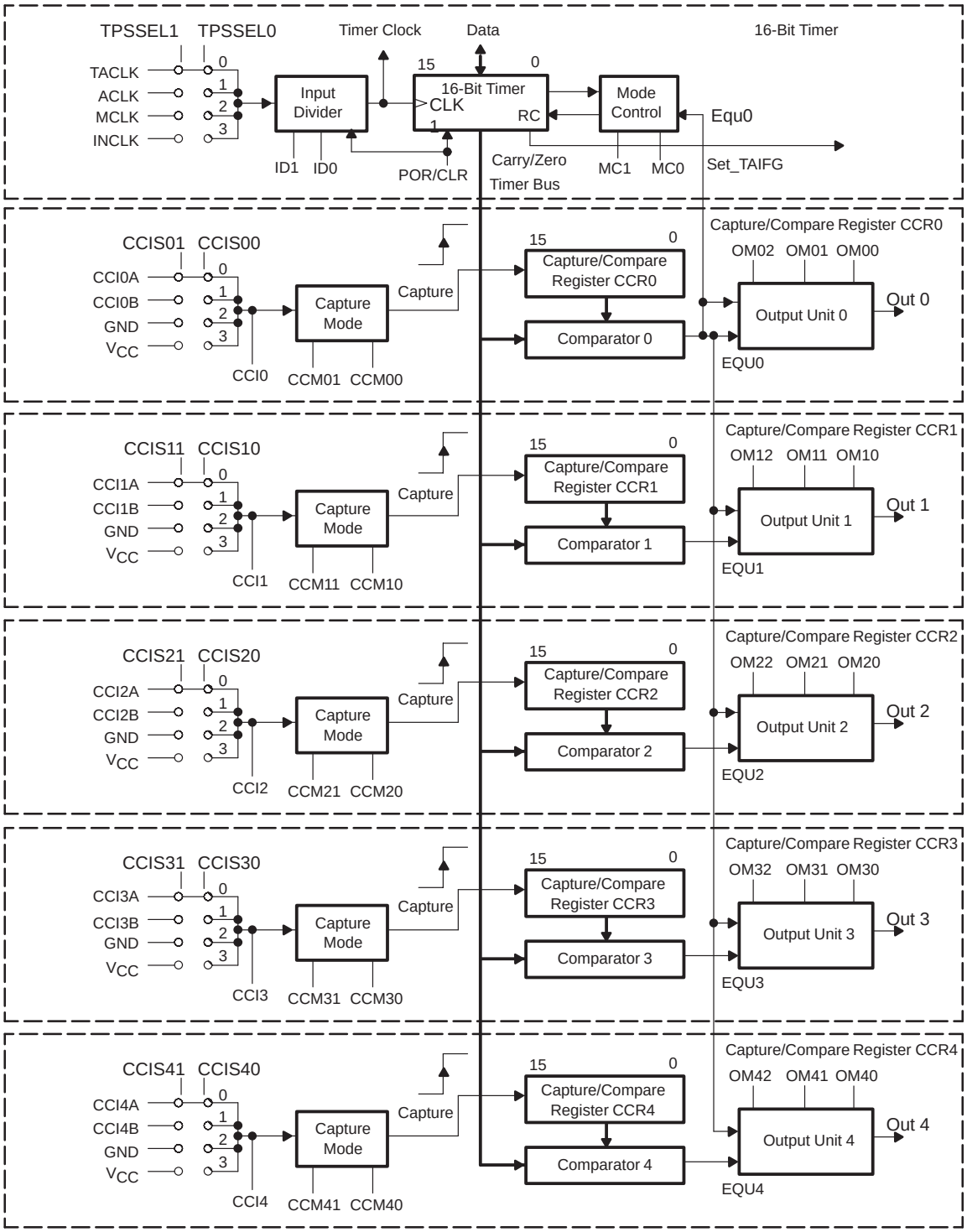
- 16-bit counter with 4 operating modes
- Selectable and configurable clock source
- Five independently configurable capture/compare registers with configurable inputs
- Five individually configurable output modules with 8 output modes

Timer_A can support multiple, simultaneous, timings; multiple capture/compares; multiple output waveforms such as PWM signals; and any combination of these.

Additionally, Timer_A has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers on captures or compares. Each capture/compare block is individually configurable and can produce interrupts on compares or on rising, falling, or both edges of an external capture signal.

The block diagram of Timer_A is shown in Figure 11–1.

Figure 11–1. Timer_A Block Diagram



11.2 Timer_A Operation

The 16-bit timer has 4 modes of operation selectable with the MC0 and MC1 bits in the TACTL register. The timer increments or decrements (depending on mode of operation) with each rising edge of the clock signal. The timer can be read or written to with software. Additionally, the timer can generate an interrupt with its ripple-carry output when it overflows.

11.2.1 Timer Mode Control

The timer has four modes of operation as shown in Figure 11–2 and described in Table 11–1: stop, up, continuous, and up/down. The operating mode is software selectable with the MC0 and MC1 bits in the TACTL register.

Figure 11–2. Mode Control

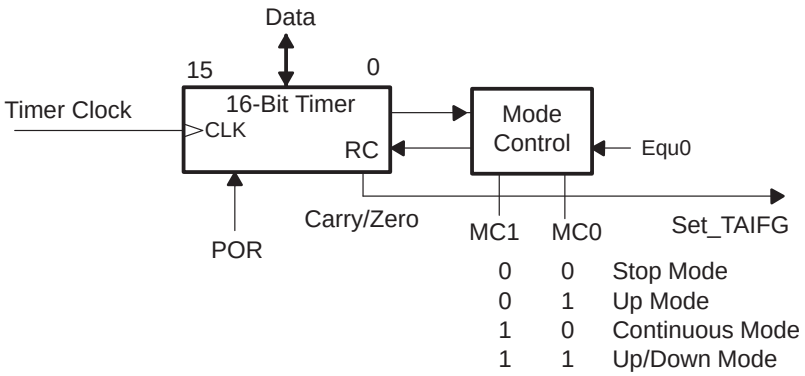


Table 11–1. Timer Modes

Mode Control			
MC1	**MC0**	**Mode**	**Description**
0	0	Stop	The timer is halted.
0	1	Up	The timer counts upward until value is equal to value of compare register CCR0.
1	0	Continuous	The timer counts upward continuously.
1	1	Up/Down	The timer counts up until the timer value is equal to compare register 0 and then it counts down to zero.

11.2.2 Clock Source Select and Divider

The timer clock can be sourced from internal clocks (i.e. ACLK, MCLK) or from an external source (TACLK) as shown in Figure 11–3. The clock source is selectable with the SSEL0 and SSEL1 bits in the TACTL register. It is important to note that when changing the clock source for the timer, errant timings can occur. For this reason it is recommended to stop the timer before changing the clock source.

The selected clock source may be passed directly to the timer or divided by 2,4, or 8, as shown in Figure 11–4. The ID0 and ID1 bits in the TACTL register select the clock division. Note that the input divider is reset by a POR signal (see chapter 3, *System Resets, Interrupts, and Operating Modes* for more information on the POR signal) or by setting the CLR bit in the TACTL register. Otherwise, the input divider remains unchanged when the timer is modified. The state of the input divider is invisible to software.

Figure 11–3. Schematic of 16-Bit Timer

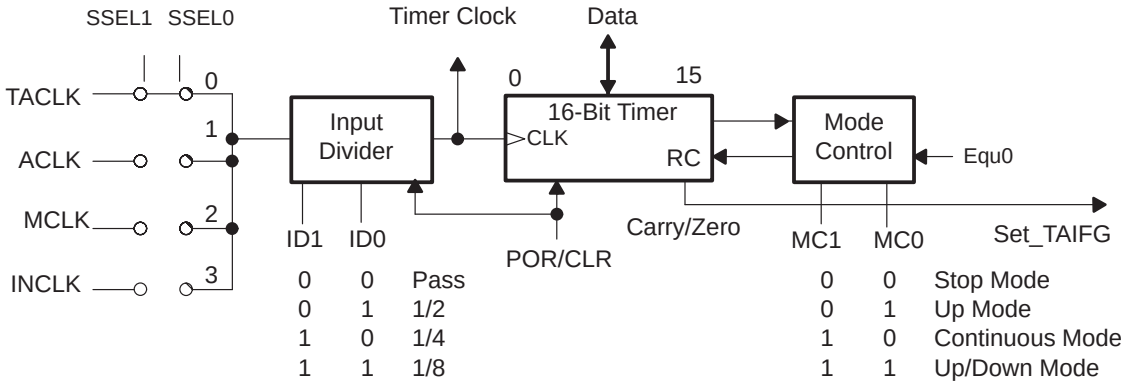
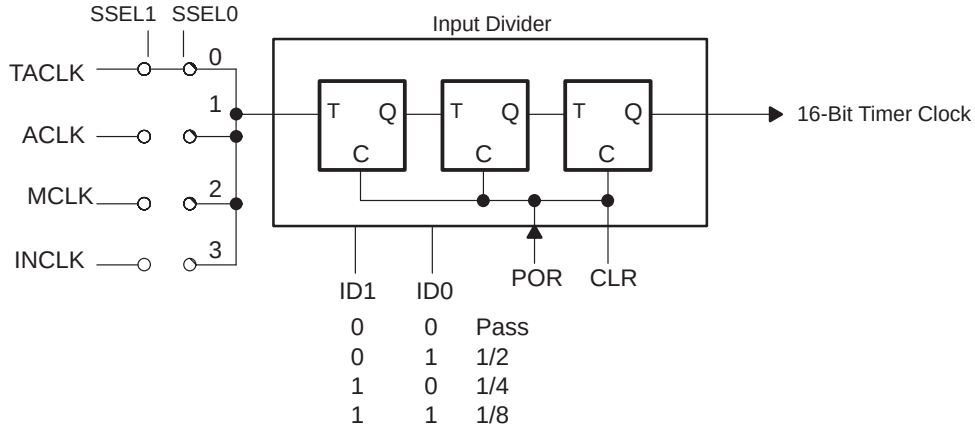


Figure 11–4. Schematic of Clock Source Select and Input Divider



11.2.3 Starting the Timer

The timer may be started or restarted in a variety of ways:

- Release Halt Mode: The timer *counts* in the selected direction when a timer mode other than stop mode is selected with the MCx bits.
- Halted by CCR0 = 0, restarted by CCR0 > 0 when the mode is either up or up/down: When the timer mode is selected to be either up or up/down, the timer may be stopped by writing 0 to capture/compare register 0 (CCR0). The timer may then be restarted by writing a non-zero value to CCR0. In this scenario, the timer starts incrementing in the up direction from zero.
- Setting the CLR bit in TACTL register: Setting the CLR bit in the TACTL register clears the timer value and input clock divider value. The timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.
- TAR is loaded with 0: When the counter (TAR register) is loaded with zero with a software instruction the timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.

11.3 Timer Modes

11.3.1 Timer – Stop Mode

Stopping and starting the timer is done simply by changing the mode control bits (MCx). The value of the timer is not affected.

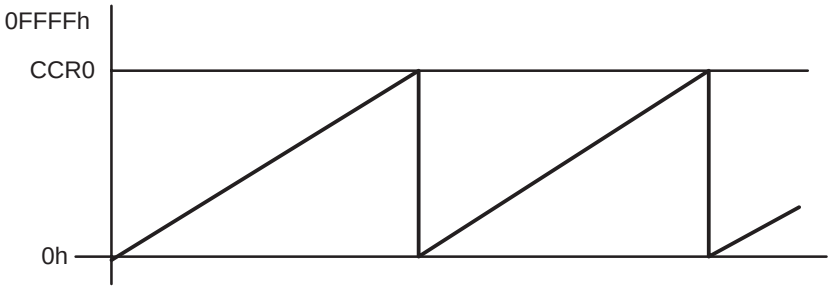
When the timer is stopped from up/down mode and then restarted in up/down mode, the timer counts in the same direction as it was counting before it was stopped. For example, if the timer is in up/down mode and counting in the down direction when the MCx bits are reset, when they are set back to the up/down direction, the timer starts counting in the down direction from its previous value. If this is not desired in an application, the CLR bit in the TACTL register can be used to clear this direction memory feature.

11.3.2 Timer – Up Mode

The up mode is used if the timer period must be different from the 65,536 (16-bit) clock cycles of the continuous mode period. The capture/compare register CCR0 data define the timer period.

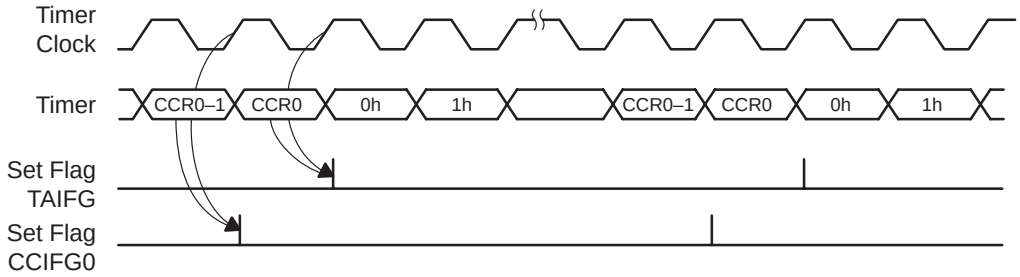
The counter *counts* up to the content of compare register CCR0, as shown in Figure 11–5. When the timer value and the value of compare register CCR0 are equal (or if the timer value is greater than the CCR0 value), the timer restarts counting from zero.

Figure 11–5. Timer Up Mode



Flag CCIFG0 is set when the timer equals the CCR0 value. The TAIFG flag is set when the timer *counts* from CCR0 to zero. All interrupt flags are set independently of the corresponding interrupt enable bit, but an interrupt is requested only if the corresponding interrupt enable bit and the GIE bit are set. Figure 11–6 shows the flag set cycle.

Figure 11–6. Up Mode Flag Setting



11.3.2.1 Timer in Up Mode – Changing the Period Register CCR0 Value

Changing the timer period register CCR0 while the timer is running can be a little tricky. When the new period is greater than or equal to the old period, the timer simply *counts* up to the new period and no special attention is required (see Figure 11–7). However, when the new period is less than the old period, the phase of the timer clock during the CCR0 update affects how the timer reacts to the new period.

If the new, smaller period is written to CCR0 during a high phase of the timer clock, then the timer rolls to zero (or begins counting down when in the up/down mode) on the next rising edge of the timer clock. However, if the new, smaller period is written during a low phase of the timer clock, then the timer continues to increment with the old period for one more clock cycle before adopting the new period and rolling to zero (or beginning counting down). This is shown in Figure 11–8.

Figure 11–7. New Period > Old Period

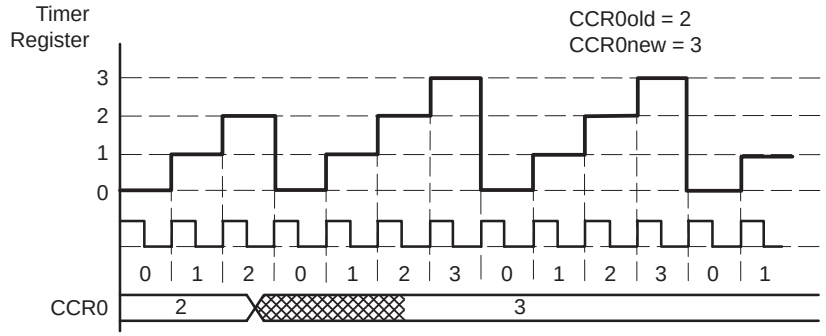
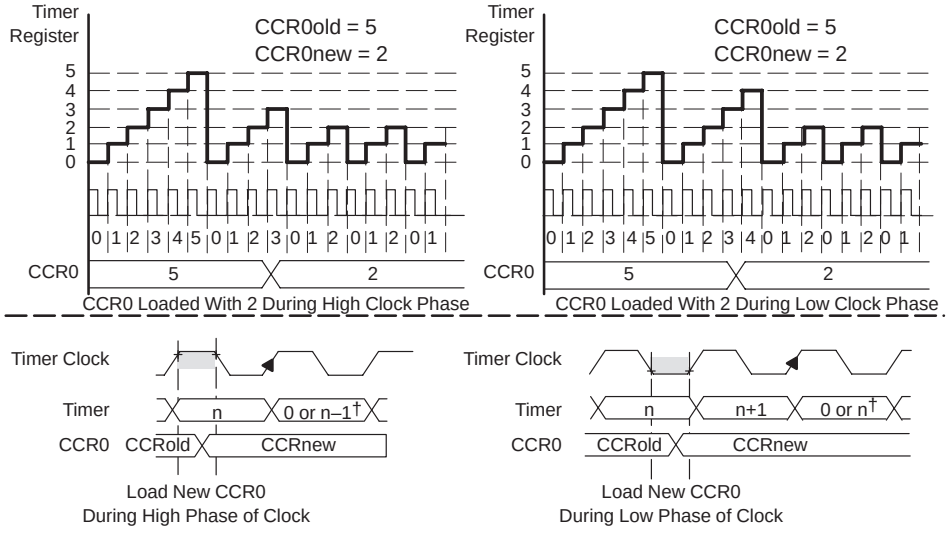


Figure 11–8. New Period < Old Period



† Up mode: 0; up/down mode: n–1

† Up mode: 0; up/down mode: n

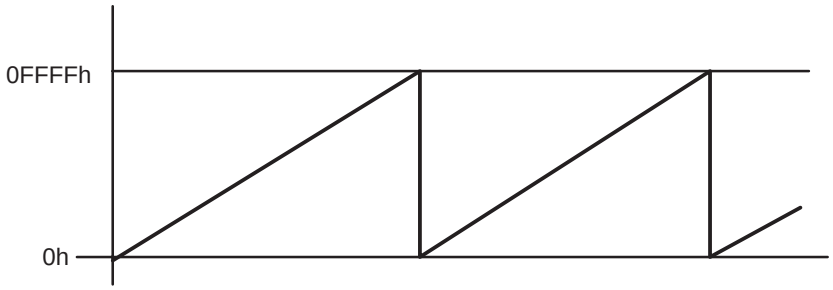
11.3.3 Timer – Continuous Mode

The continuous mode is used if the timer period of 65,536 clock cycles is used for the application. A typical application of the continuous mode is to generate multiple, independent timings. In continuous mode, the capture/compare register CCR0 works in the same way as the other compare registers.

The capture/compare registers and different output modes of each output unit are useful to capture timer data based on external events or to generate various different types of output signals. Examples of the different output modes used with timer-continuous mode are shown in Figure 11–25.

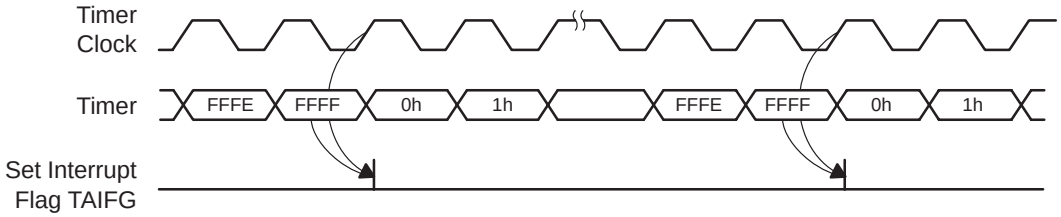
In continuous mode, the timer starts counting from its present value. The counter counts up to 0FFFFh and restarts by counting from zero as shown in Figure 11–9.

Figure 11–9. Timer Continuous Mode



The TAIFG flag is set when the timer *counts* from 0FFFFh to zero. The interrupt flag is set independently of the corresponding interrupt enable bit, as shown in Figure 11–10. An interrupt is requested if the corresponding interrupt enable bit and the GIE bit are set.

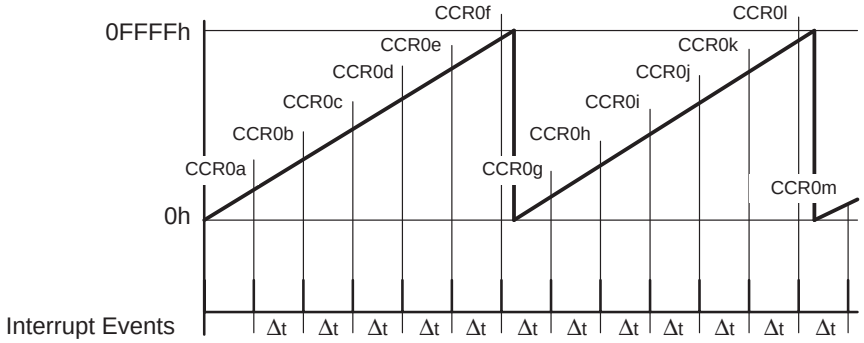
Figure 11–10. Continuous Mode Flag Setting



11.3.3.1 Timer – Use of the Continuous Mode

The continuous mode can be used to generate time intervals for the application software. Each time an interval is completed, an interrupt can be generated. In the interrupt service routine of this event, the time until the next event is added to capture/compare register CCRx as shown in Figure 11–11. Up to five independent time events can be generated using all five capture/compare blocks.

Figure 11–11. Output Unit in Continuous Mode for Time Intervals

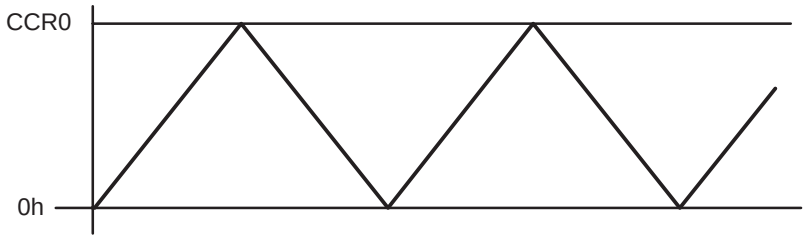


Time intervals can be produced with other modes as well, where CCR0 is used as the period register. Their handling is more complex since the sum of the old CCRx data and the new period can be higher than the CCR0 value. When the sum CCRxold plus Δt is greater than the CCR0 data, the CCR0 value must be subtracted to obtain the correct time interval. The period is twice the value in the CCR0 register.

11.3.4 Timer – Up/Down Mode

The up/down mode is used if the timer period must be different from the 65,536 clock cycles, and if symmetrical pulse waveform generation is needed. In up/down mode, the timer counts up to the content of compare register CCR0, then back down to zero, as shown in Figure 11–12. The period is twice the value in the CCR0 register.

Figure 11–12. Timer Up/Down Mode



The up/down mode also supports applications that require dead times between output signals. For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the following example (see Figure 11–13), the t_{dead} is:

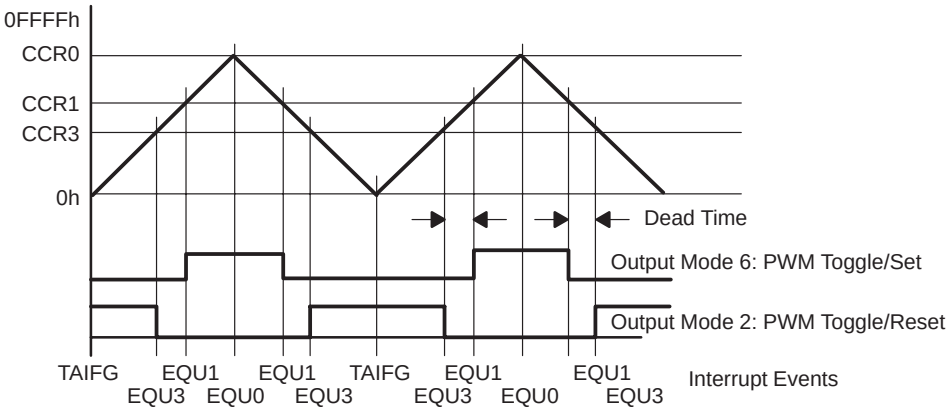
$$t_{dead} = t_{timer} \times (CCR1 - CCR3) =$$

With: t_{dead} Time during which both outputs need to be inactive

t_{timer} Cycle time of the timer clock

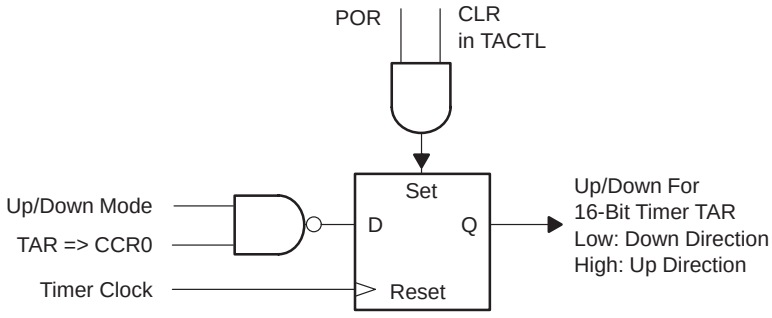
CCRx Content of capture/compare register x

Figure 11–13. Output Unit in Up/Down Mode (II)



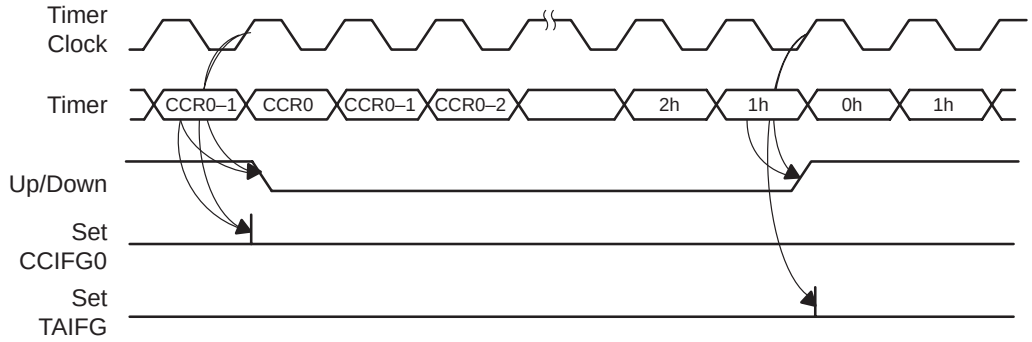
The count direction is always latched with a flip-flop (Figure 11–14). This is useful because it allows the user to stop the timer and then restart it in the same direction it was counting before it was stopped. For example, if the timer was counting down when the MCx bits were reset, then it will continue counting in the down direction if it is restarted in up/down mode. If this is not desired, the CLR bit in the TACTL register must be used to clear the direction. Note that the CLR bit affects other setup conditions of the timer. Refer to Section 11.6 for a discussion of the Timer_A registers.

Figure 11–14. Timer Up/Down Direction Control



In up/down mode, the interrupt flags (CCIFG0 and TAIFG) are set at equal time intervals (Figure 11–15). Each flag is set only once during the period, but they are separated by 1/2 the timer period. CCIFG0 is set when the timer *counts* from CCR0–1 to CCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Each flag is capable of producing a CPU interrupt when enabled.

Figure 11–15. Up/Down Mode Flag Setting

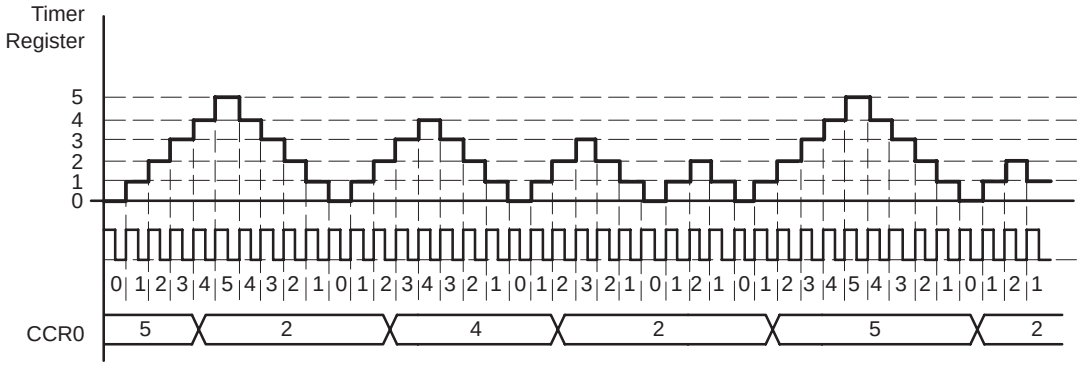


11.3.4.1 Timer In Up/Down Mode – Changing the Value of Period Register CCR0

Changing the period value while the timer is running in up/down mode is even trickier than in up mode. Like in up mode, the phase of the timer clock when CCR0 is changed affects the timer's behavior. Additionally, in up/down mode, the direction of the timer also affects the behavior.

If the timer is counting in the up direction when the new period is written to CCR0, the conditions in the up/down mode are identical to those in the up mode. See Section 11.3.2.1 for details. However, if the timer is counting in the down direction when CCR0 is updated, it continues its descent until it reaches zero. The new period takes effect only after the counter finishes counting down to zero. See Figure 11–16.

Figure 11–16. Altering CCR0 – Timer in Up/Down Mode



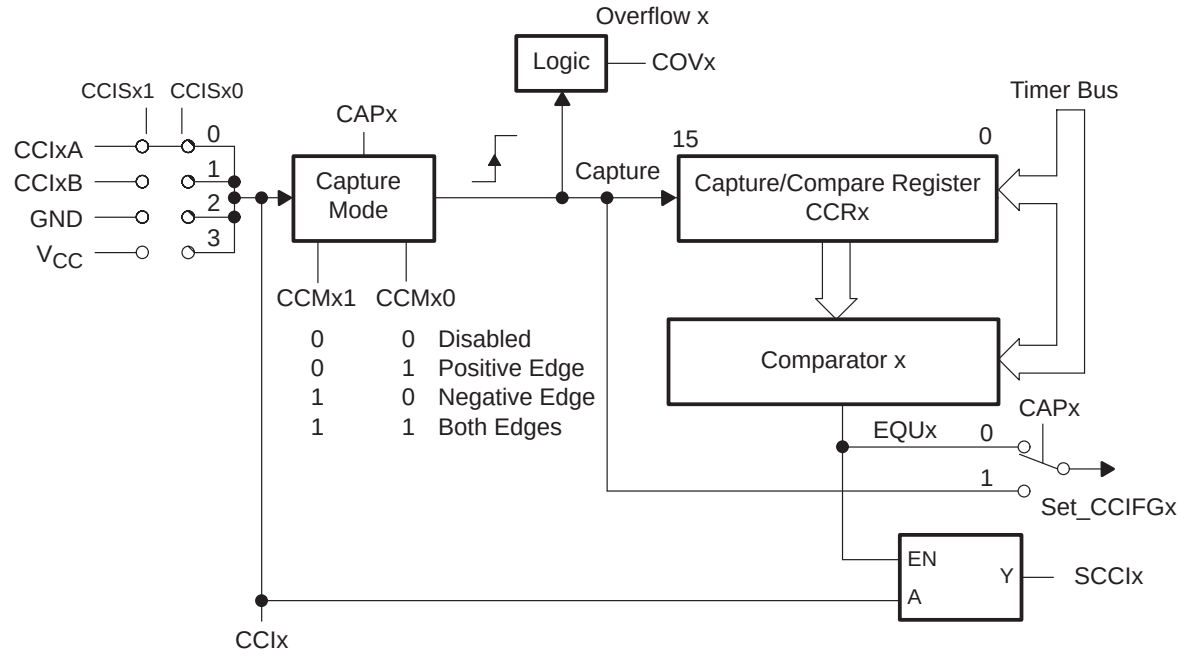
11.4 Capture/Compare Blocks

Five identical capture/compare blocks (shown in Figure 11–17) provide flexible control for real-time processing. Any one of the blocks may be used to capture the timer data at an applied event, or to generate time intervals. Each time a capture occurs or a time interval is completed, interrupts can be generated from the applicable capture/compare register. The mode bit CAPx, in control word CCTLx, selects the compare or capture operation and the capture mode bits CCMx1 and CCMx0 in control word CCTLx define the conditions under which the capture function is performed.

Both the interrupt enable bit CCIEx and the interrupt flag CCIFGx are used for capture and compare modes. CCIEx enables the corresponding interrupt. CCIFGx is set on a capture or compare event.

The capture inputs CCIxA and CCIxB are connected to external pins or internal signals. Different MSP430 devices may have different signals connected to CCIxA and CCIxB. The data sheet should always be consulted to determine the Timer_A connections for a particular device.

Figure 11–17. Capture/Compare Blocks



11.4.1 Capture/Compare Block – Capture Mode

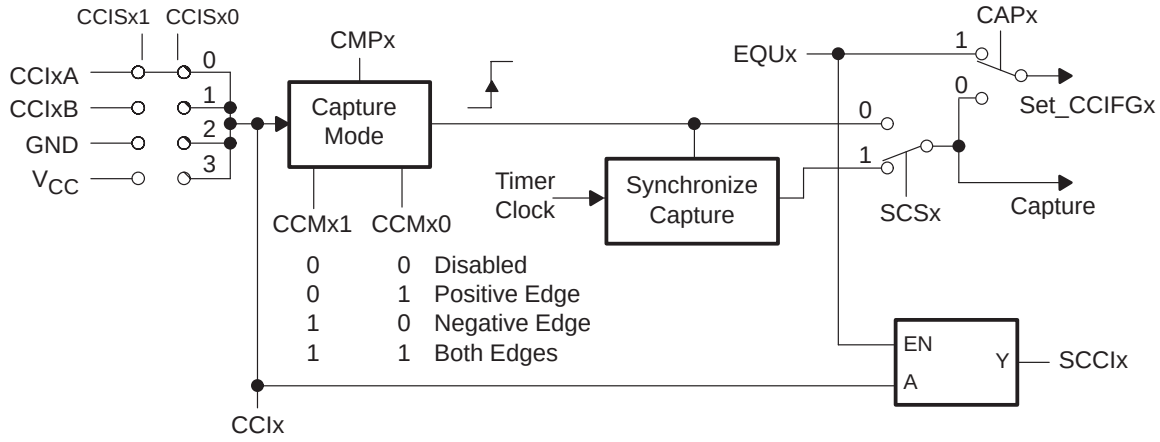
The capture mode is selected if the mode bit CAPx, located in control word CCTLx, is set. The capture mode is used to fix time events. It can be used for speed computations or time measurements. The timer value is copied into the capture register (CCRx) with the selected edge (positive, negative, or both) of the input signal. Captures may also be initiated by software as described in section 11.4.1.1.

If a capture is performed:

- The interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if both interrupt enable bits CCIEx and GIE are set.

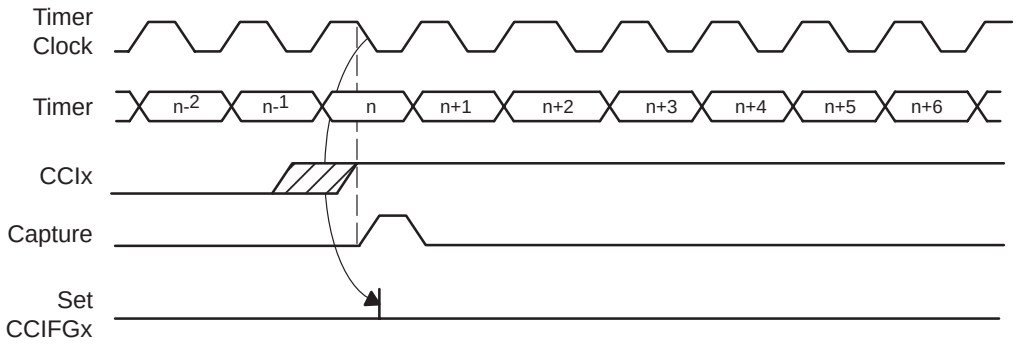
The input signal to the capture/compare block is selected using control bits CCISx1 and CCISx0, as shown in Figure 11–18. The input signal can be read at any time by the software by reading bit CCIx. The input signal may also be latched with compare signal EQUx (see SCCIx bit below) when in compare mode. This feature was designed specifically to support implementing serial communications with Timer_A. See section 11.7 for more details on using Timer_A as a UART.

Figure 11–18. Capture Logic Input Signal



The capture signal can also be synchronized with the timer clock to avoid race conditions between the timer data and the capture signal. This is illustrated in Figure 11–19. The bit SCSx in capture/compare control register CCTLx selects the capture signal synchronization.

Figure 11–19. Capture Signal

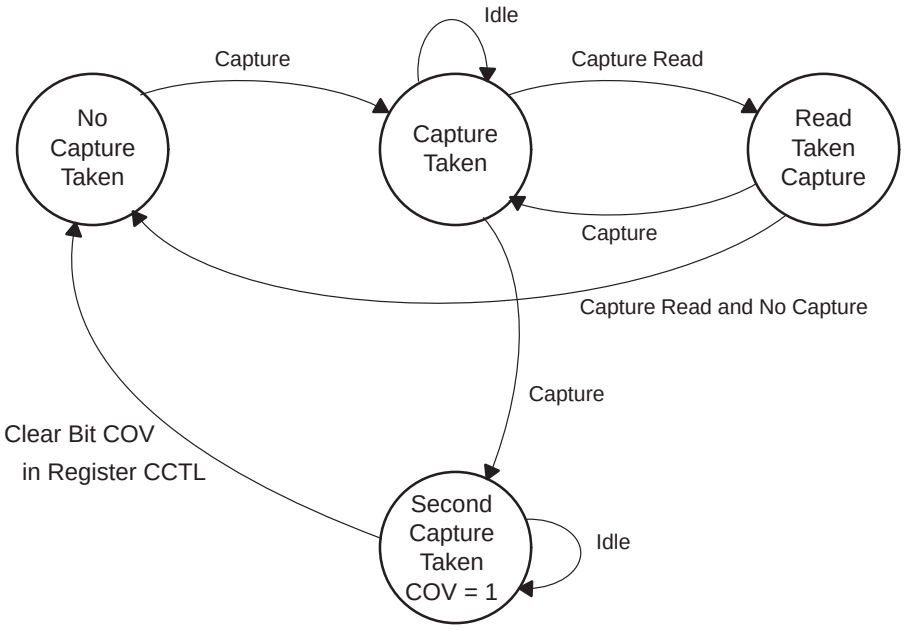


Applications with slow timer clocks can use the nonsynchronized capture signal. In this scenario the software can validate the data and correct it if necessary as shown in the following example:

```
; Software example for the handling of asynchronous
; capture signals
;
; The data of the capture/compare register CCRx are taken
; by the software in the according interrupt routine
; – they are taken only after a CCIFG was set.
; The timer clock is much slower than the system clock
; MCLK.
;
CCRx_Int_hand...                ; Start of interrupt
                                ; handler
            ...
            ...
            CMP   &CCRx,&TAR    ; Test if the data
                                ; CCRX = TAR
            JEQ   Data_Valid
            MOV   &TAR,&CCRx    ; The data in CCRx is
                                ; wrong, use the timer data
Data_Valid  ...   ...           ; The data in CCRx are valid
            ...
            ...
            RETI
;
```

Overflow logic is provided with each capture/compare register to flag the user if a second capture is performed before data from the first capture was read successfully. Bit COVx in register CCTLx is set when this occurs as shown in Figure 11–20.

Figure 11–20. Capture Cycle



Overflow bit COVx is reset by the software as described in the following example:

```
; Software example for the handling of captured data
; looking for overflow condition
;
; The data of the capture/compare register CCRx are taken
; by the software and immediately with the next
; instruction the overflow bit is tested and a decision is
; made to proceed regularly or with an error handler
;
CCRx_Int_hand   ...       ; Start of handler Interrupt
          ...
          ...
                MOV   &CCRx,RAM_Buffer
                BIT   #COV,&CCTLx
                JNZ   Overflow_Hand
          ...
          ...
          ...
                RETI
Overflow_Hand   BIC   #COV,&CCTLx  ; reset capture
                                   ; overflow flag
                                   ; get back to lost
                                   ; synchronization

          ...
          ...
;               RETI
```

Note: Capture With Timer Halted

The capture should be disabled when the timer is halted. The sequence to follow is: stop the capture, then stop the timer. When the capture function is restarted, the sequence should be: start the capture, then start the timer.

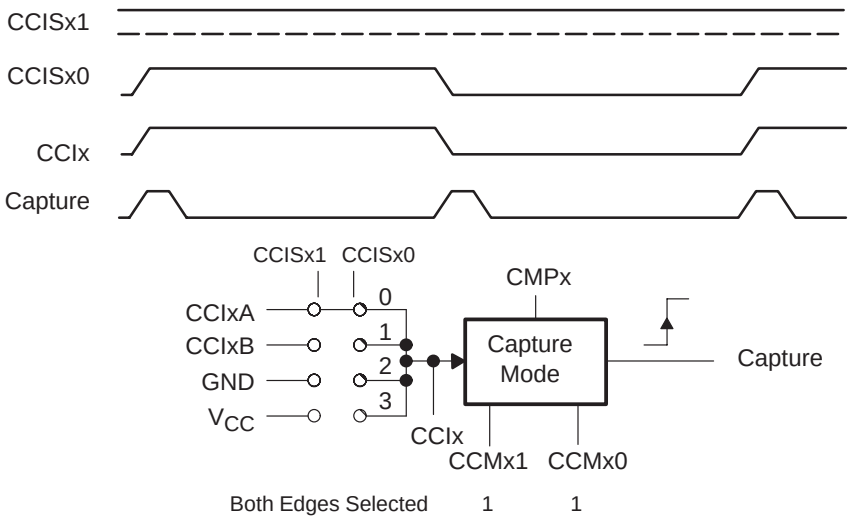
11.4.1.1 Capture/Compare Block, Capture Mode – Capture Initiated by Software

In addition to internal and external signals, captures can be initiated by software. This is useful for various purposes, such as:

- To measure time used by software routines
- To measure time between hardware events
- To measure the system frequency

Two bits, CCISx1 and CCISx0, and the capture mode selected by bits CCMx1 and CCMx0 are used by the software to initiate the capture. The simplest realization is when the capture mode is selected to capture on both edges of CCIx and bit CCISx1 is set. Software then toggles bit CCISx0 to switch the capture signal between V_{CC} and GND, initiating a capture each time the input is toggled, as shown in Figure 11–21.

Figure 11–21. Software Capture Example



The following is a software example of a capture performed by software:

```
; The data of capture/compare register CCRx are taken
; by the software. It is assumed that CCMx1, CCMx0, and
; CCISx1 bits are set. Bit CCIS0 selects the CCIx
; signal to be high or low.
;
;
          ...
          ...
          XOR   #CCISx0, &CCTLx
          ...
          ...
          ...
```

11.4.2 Capture/Compare Block – Compare Mode

The compare mode is selected if the CAPx bit, located in control word CCTLx, is reset. In compare mode all the capture hardware circuitry is inactive and the capture-mode overflow logic is inactive.

The compare mode is most often used to generate interrupts at specific time intervals or used in conjunction with the output unit to generate output signals such as PWM signals. If the timer becomes equal to the value in compare register x, then:

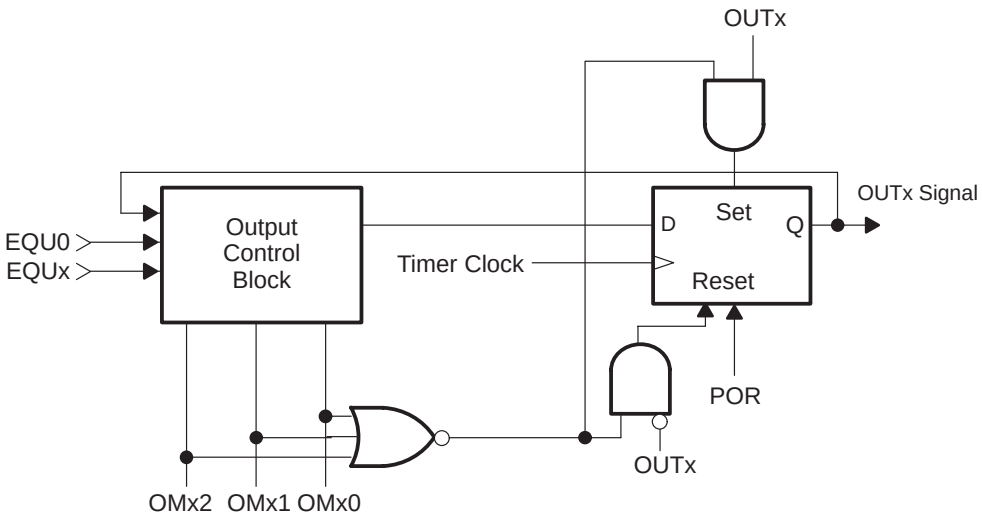
- Interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if interrupt enable bits CCIEx and GIE are set.
- Signal EQUx is output to the output unit. This signal affects the output OUTx, depending on the selected output mode.

The EQU0 signal is true when the timer value is greater or equal to the CCR0 value. The EQU1 to EQU4 signals are true when the timer value is equal to the corresponding CCR1 to CCR4 values.

11.5 The Output Unit

Each capture/compare block contains an output unit shown in Figure 11–22. The output unit is used to generate output signals such as PWM signals. Each output unit has 8 operating modes that can generate a variety of signals based on the EQU0 and EQUx signals. The output mode is selected with the OMx bits located in the CCTLx register.

Figure 11–22. Output Unit



OMx2	OMx1	OMx0	
0	0	0	Output mode: OUTx signal reflects the value of the OUTx bit
0	0	1	Set mode: OUT x signal reflects the value of signal EQUx
0	1	0	PWM toggle/reset: EQUx toggles OUTx. EQU0 resets OUTx.
0	1	1	PWM set/reset: EQUx sets OUTx. EQU0 resets OUTx
1	0	0	Toggle: EQUx toggles OUTx signal.
1	0	1	Reset: EQUx resets OUTx.
1	1	0	PWM toggle/set: EQUx toggles OUTx. EQU0 sets OUTx.
1	1	1	PWM reset/set: EQUx resets OUTx. EQUx sets OUTx.

Note: OUTx signal updates with rising edge of timer clock for all modes except mode 0.
Modes 2, 3, 6, 7 not useful for output unit 0.

11.5.1 Output Unit – Output Modes

The output modes are defined by the OMx bits and are discussed below. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0.

Output mode 0: Output mode:
The output signal OUTx is defined by the OUTx bit in control register CCTLx. The OUTx signal updates immediately upon completion of writing the bit information.

Output mode 1: Set mode:
The output is set when the timer value becomes equal to capture/compare data CCRx. It remains set until a reset of the timer, or until another output mode is selected.

Output mode 2: PWM toggle/reset mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. It is reset when the timer value becomes equal to CCR0.

Output mode 3: PWM set/reset mode:
The output is set when the timer value becomes equal to capture/compare data CCRx. It is reset when the timer value becomes equal to CCR0.

Output mode 4: Toggle mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. The output period is double the timer period.

Output mode 5: Reset mode:
The output is reset when the timer value becomes equal to capture/compare data CCRx. It remains reset until another output mode is selected.

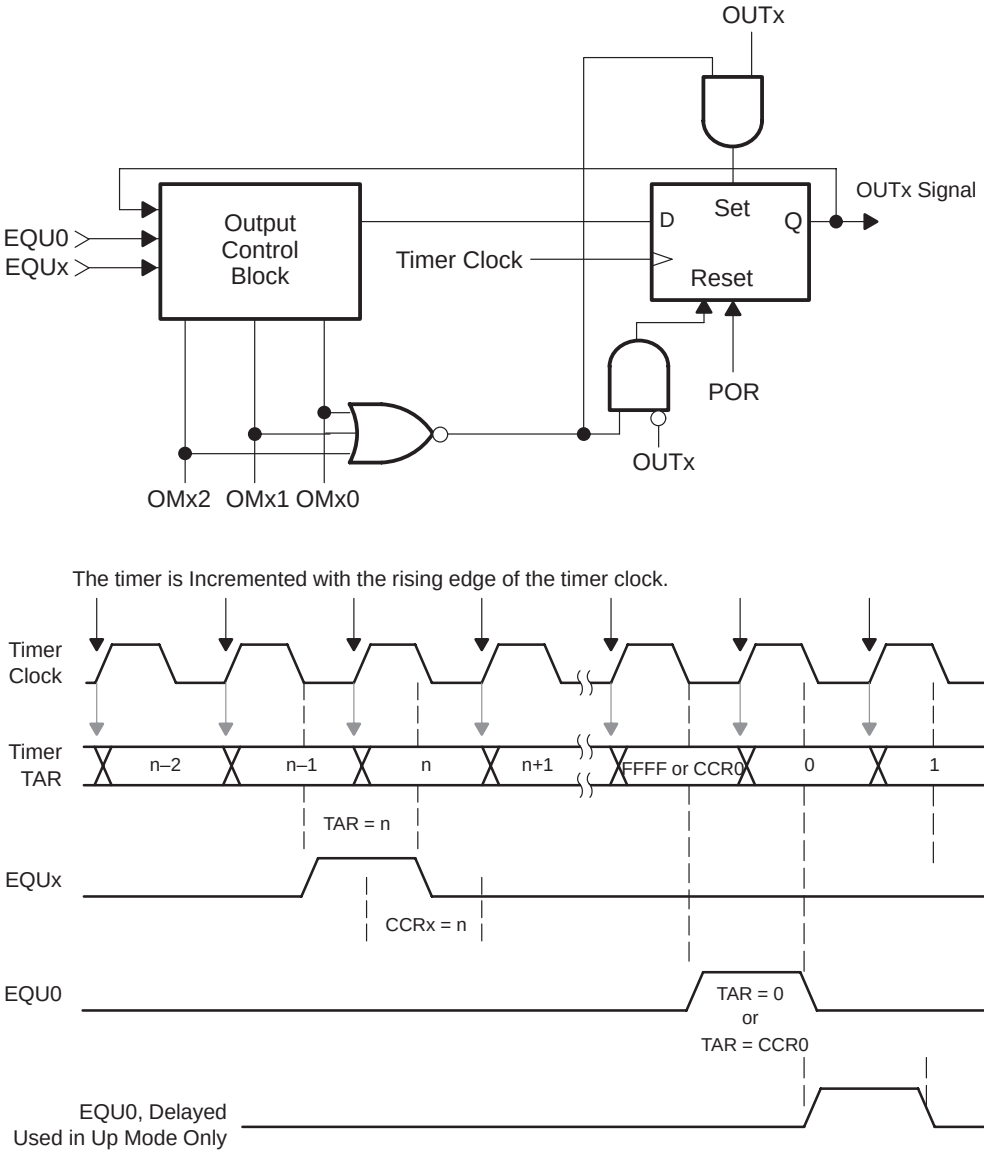
Output mode 6: PWM toggle/set mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. It is set when the timer value becomes equal to CCR0.

Output mode 7: PWM toggle/set mode:
The output is reset when the timer value becomes equal to capture/compare data CCRx. It is set when the timer value becomes equal to CCR0.

11.5.2 Output Control Block

The output control block prepares the value of the OUTx signal, which is latched into the OUTx flip-flop with the next positive timer clock edge, as shown in Figure 11–23 and Table 11–2. The equal signals EQUx and EQU0 are sampled during the negative level of the timer clock, as shown in Figure 11–23.

Figure 11–23. Output Control Block



EQU0 delayed is used in up mode, not EQU0. EQU0 is active high when TAR = CCR0. EQU0 delayed is active high when TAR = 0.

Table 11–2. State of OUTx at Next Rising Edge of Timer Clock

Mode	EQU0	EQUx	D
0	x	x	x(OUTx bit)
1	x	0	OUTx (no change)
	x	1	1 (set)
2	0	0	OUTx (no change)
	0	1	OUTx (toggle)
	1	0	0 (reset)
	1	1	1 (set)
3	0	0	OUTx (no change)
	0	1	1 (set)
	1	0	0 (reset)
	1	1	1 (set)
4	x	0	OUTx (no change)
	x	1	OUTx (toggle)
5	x	0	OUTx (no change)
	x	1	0 (reset)
6	0	0	OUTx (no change)
	0	1	OUTx (toggle)
	1	0	1 (set)
	1	1	0 (reset)
7	0	0	OUTx (no change)
	0	1	0 (reset)
	1	0	1 (set)
	1	1	0 (reset)

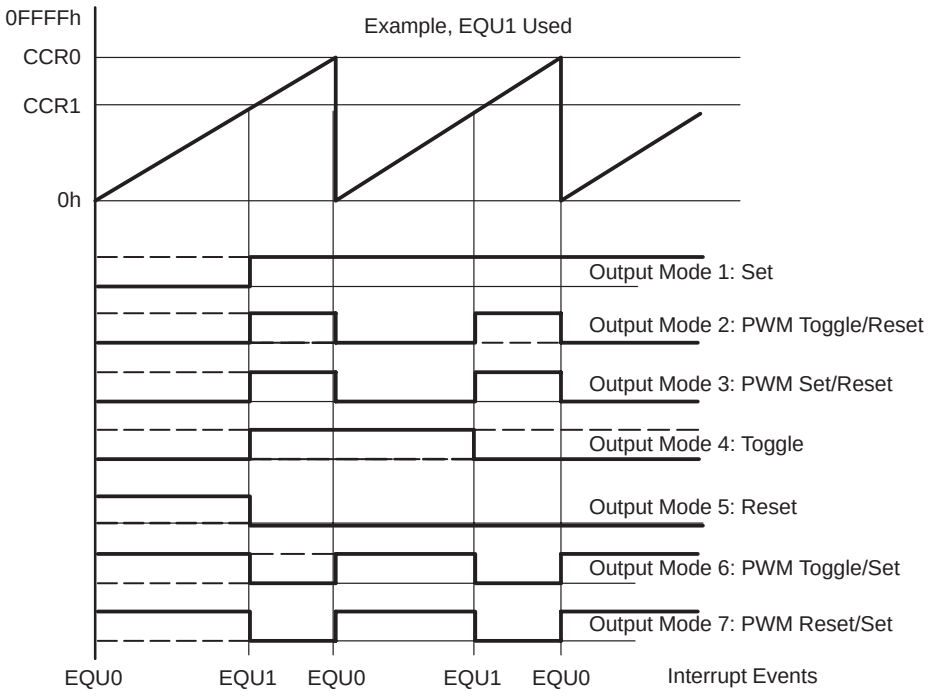
11.5.3 Output Examples

The following are some examples of possible output signals using the various timer and output modes.

11.5.3.1 Output Examples – Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the CCRx value, and rolls from CCR0 to zero, depending on the output mode, as shown in Figure 11–24.

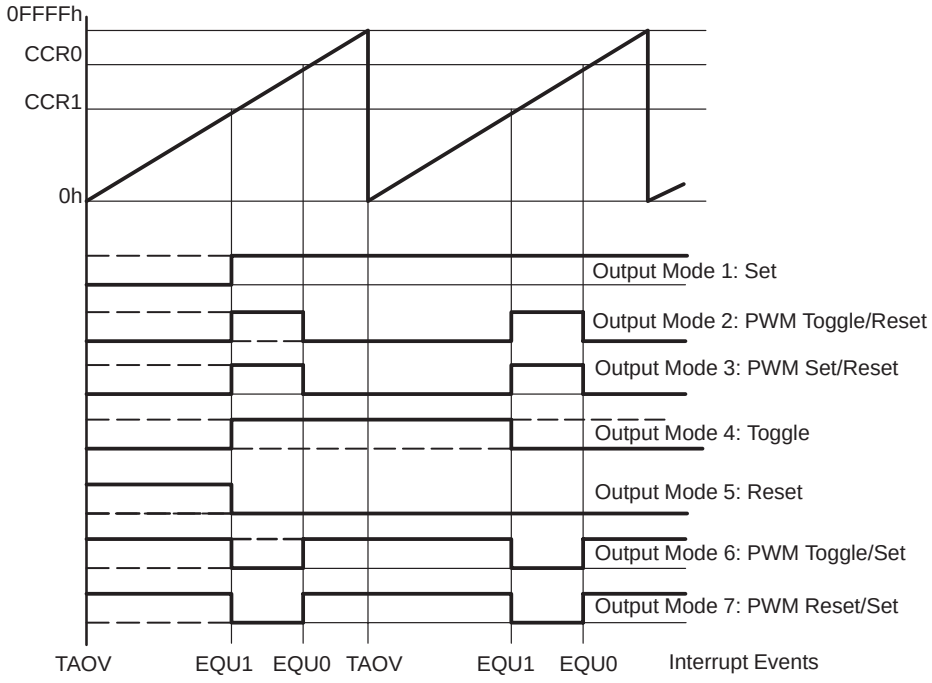
Figure 11–24. Output Examples – Timer in Up Mode



11.5.3.2 Output Examples – Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the CCRx and CCR0 values, depending on the output mode, as shown in Figure 11–25.

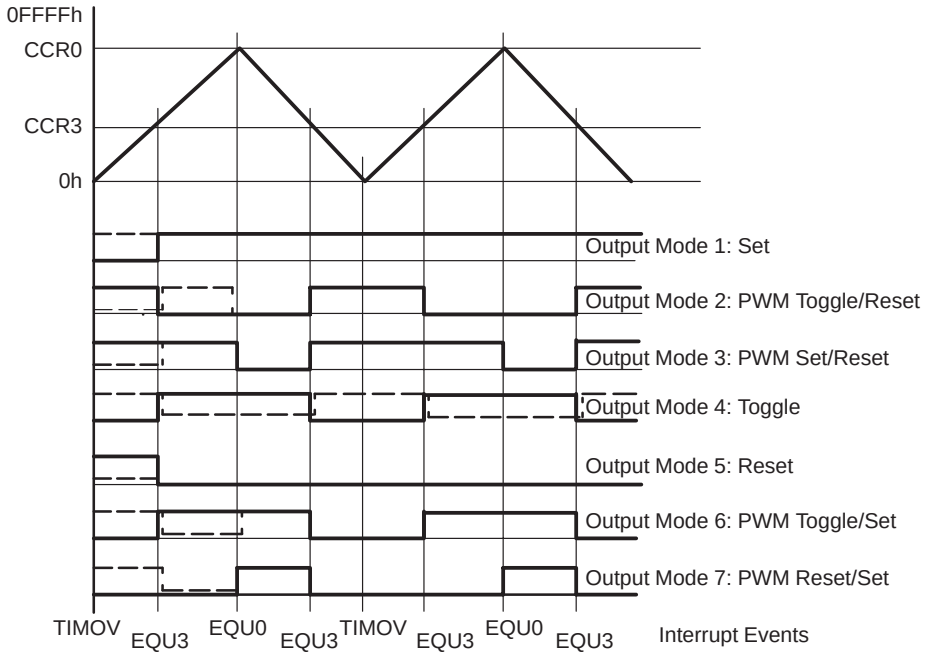
Figure 11–25. Output Examples – Timer in Continuous Mode



11.5.3.3 Output Examples – Timer in Up/Down Mode

The OUTx signal changes when the timer equals CCRx in either count direction and when the timer equals CCR0, depending on the output mode, as shown in Figure 11–26.

Figure 11–26. Output Examples – Timer in Up/Down Mode (I)



11.6 Timer_A Registers

The Timer_A registers, described in Table 11–3, are word-structured and must be accessed using word instructions.

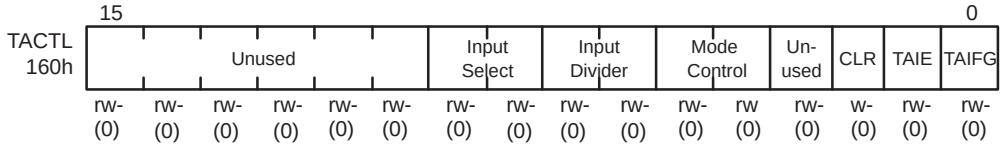
Table 11–3. Timer_A Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	160h	POR reset
Timer_A register	TAR	Read/write	170h	POR reset
Cap/com control 0	CCTL0	Read/write	162h	POR reset
Capture/compare 0	CCR0	Read/write	172h	POR reset
Cap/com control 1	CCTL1	Read/write	164h	POR reset
Capture/compare 1	CCR1	Read/write	174h	POR reset
Cap/com control 2	CCTL 2	Read/write	166h	POR reset
Capture/compare 2	CCR2	Read/write	176h	POR reset
Cap/com control 3	CCTL3	Read/write	168h	POR reset
Capture/compare 3	CCR3	Read/write	178h	POR reset
Cap/com control 4	CCTL4	Read/write	16Ah	POR reset
Capture/compare 4	CCR4	Read/write	17Ah	POR reset
Interrupt vector	TAIV	Read	12Eh	(POR reset)

11.6.1 Timer_A Control Register TACTL

The timer and timer operation control bits are located in the timer control register (TACTL) shown in Figure 11–27. All control bits are reset automati–cally by the POR signal, but are not affected by the PUC signal. The control register must be accessed using word instructions.

Figure 11–27. Timer_A Control Register TACTL



Bit 0: TAIFG: This flag indicates a timer overflow event.

- Up mode: TAIFG is set if the timer *counts* from CCR0 value to 0000h.
- Continuous mode: TAIFG is set if the timer *counts* from 0FFFFh to 0000h.
- Up/down mode: TAIFG is set if the timer *counts* down from 0001h to 0000h.

Bit 1: Timer overflow interrupt enable (TAIE) bit. An interrupt request from the timer overflow bit is enabled if this bit is set, and is disabled if reset.

Bit 2: Timer clear (CLR) bit. The timer and input divider are reset with the POR signal, or if bit CLR is set. The CLR bit is automatically reset and is always read as zero. The timer starts in the upward direction with the next valid clock edge, unless halted by cleared mode control bits.

Bit 3: Not used

Bits 4, 5: Mode control: Table 11–4 describes the mode control bits.

Table 11–4. Mode Control

MC1	MC0	Count Mode	Description
0	0	Stop	Timer is halted.
0	1	Up to CCR0	Timer counts up to CCR0 and restarts at 0.
1	0	Continuous up	Timer counts up to 0FFFFh and restarts at 0.
1	1	Up/down	Timer continuously *counts* up to CCR0 and back down to 0.

Bits 6, 7: Input divider control bits. Table 11–5 describes the input divider control bits.

Table 11–5. Input Clock Divider Control Bits

ID1	ID0	Operation	Description
0	0	/1	Input clock source is passed to the timer.
0	1	/2	Input clock source is divided by two.
1	0	/4	Input clock source is divided by four.
1	1	/8	Input clock source is divided by eight.

Bits 8, 9: Clock source selection bits. Table 11–6 describes the clock source selections.

Table 11–6. Clock Source Selection

SSEL1	SSEL0	O/P Signal	Comment
0	0	TACLK	See data sheet device description
0	1	ACLK	Auxiliary clock ACLK is used
1	0	MCLK	System clock MCLK
1	1	INCLK	See device description in data sheet

Bits 10 to 15: Unused

Note: Changing Timer_A Control Bits

If the timer operation is modified by the control bits in the TACTL register, the timer should be halted during this modification. Critical modifications are the input select bits, input divider bits, and the timer clear bit. Asynchronous clocks, input clock, and system clock can result in race conditions where the timer reacts unpredictably.

The recommended instruction flow is:

1) Modify the control register and stop the timer.

2) Start the timer operation.

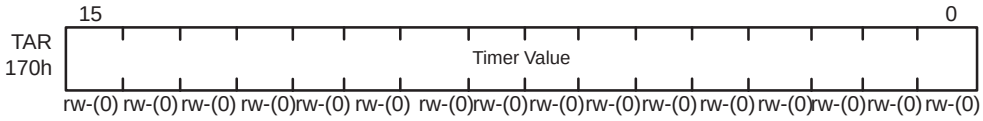
For example:

```
MOV #01C6,&TACTL    ; ACLK/8, timer stopped, timer cleared
BIS  #10h,&TACTL    ; Start timer with up mode
```

11.6.2 Timer_A Register TAR

The TAR register is the value of the timer.

Figure 11–28. TAR Register



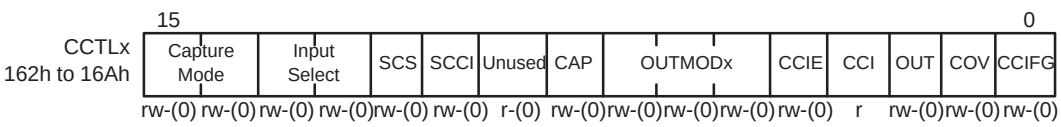
Note: Modifying Timer A Register TAR

When ACLK or the external clock TACLK or INCLK is selected for the timer clock, any write to timer register TAR should occur while the timer is not operating; otherwise, the results may be unpredictable. In this case, the timer clock is asynchronous to the CPU clock MCLK and critical race conditions exist.

11.6.3 Capture/Compare Control Register CCTLx

Each capture/compare block has its own control word CCTLx, shown in Figure 11–29. The POR signal resets all bits of CCTLx; the PUC signal does not affect these bits.

Figure 11–29. Capture/Compare Control Register CCTLx



Bit 0: Capture/compare interrupt flag CCIFGx

Capture mode:

If set, it indicates that a timer value was captured in the CCRx register.

Compare mode:

If set, it indicates that a timer value was equal to the data in the CCRx register.

CCIFG0 flag:

CCIFG0 is automatically reset when the interrupt request is accepted.

CCIFG1 to CCIFG4 flags:

The flag that caused the interrupt is automatically reset after the TAIV word is accessed. If the TAIV register is not accessed, the flags must be reset with software.

No interrupt is generated if the corresponding interrupt enable bit is reset, but the flag will be set. In this scenario, the flag must be reset by the software.

Setting the CCIFGx flag with software will request an interrupt if the interrupt-enable bit is set.

Bit 1: Capture overflow flag COV

Compare mode selected, CAP = 0:

Capture signal generation is reset. No compare event will set COV bit.

Capture mode selected, CAP = 1:

The overflow flag COV is set if a second capture is performed before the first capture value is read. The overflow flag must be reset with software. It is not reset by reading the capture value.

Bit 2: The OUTx bit determines the value of the OUTx signal if the output mode is 0.

Bit 3: Capture/compare input signal CCIx:
The selected input signal (CCIxA, CCIxB, V_{CC}. or GND) can be read by this bit. See Figure 11–18.

Bit 4: Interrupt enable CCIEx: Enables or disables the interrupt request signal of capture/compare block x. Note that the GIE bit must also be set to enable the interrupt.
0: Interrupt disabled
1: Interrupt enabled

Bits 5 to 7: Output mode select bits:
Table 11–7 describes the output mode selections.

Table 11–7. Capture/Compare Control Register Output Mode

Bit Value	Output Mode	Description
0	Output only	The OUTx signal reflects the value of the OUTx bit
1	Set	EQUx sets OUTx
2	PWM toggle/reset	EQUx toggles OUTx. EQU0 resets OUTx.
3	PWM set/reset	EQUx sets OUTx. EQU0 resets OUTx
4	Toggle	EQUx toggles OUTx signal.
5	Reset	EQUx resets OUTx.
6	PWM toggle/set	EQUx toggles OUTx. EQU0 sets OUTx.
7	PWM reset/set	EQUx resets OUTx. EQUx sets OUTx.

Note: OUTx updates with rising edge of timer clock for all modes except mode 0. Modes 2, 3, 6, 7 not useful for output unit 0.

Bit 8: CAP sets capture or compare mode.
0: Compare mode
1: Capture mode

Bit 9: Read only, always read as 0.

Bit 10: SCCIx bit:
The selected input signal (CCIxA, CCIxB, V_{CC}, or GND) is latched with the EQUx signal into a transparent latch and can be read via this bit.

Bit 11: SCSx bit:
This bit is used to synchronize the capture input signal with the timer clock.
0: asynchronous capture
1: synchronous capture

Bits 12, 13: Input select, CCIS0 and CCIS1:
These two bits define the capture signal source. These bits are not used in compare mode.
0 Input CCIxA is selected
1 Input CCIxB is selected
2 GND
3 V_{CC}

Bits 14, 15: Capture mode bits:
Table 11–8 describes the capture mode selections.

Table 11–8. Capture/Compare Control Register Capture Mode

Bit Value	Capture Mode	Description
0	Disabled	The capture mode is disabled.
1	Pos. Edge	Capture is done with rising edge.
2	Neg. Edge	Capture is done with falling edge.
3	Both Edges	Capture is done with both rising and falling edges.

Note: Simultaneous Capture and Capture Mode Selection

Captures must not be performed simultaneously with switching from compare to capture mode. Otherwise, the result in the capture/compare register will be unpredictable.

The recommended instruction flow is:

1) Modify the control register to switch from compare to capture.

2) Capture

For example:

```
BIS  #CAP,&CCTL2        ; Select capture with register CCR2
XOR #CCIS1,&CCTL2       ; Software capture:     CCIS0 = 0
                        ;                       Capture mode = 3
```

11.6.4 Timer_A Interrupt Vector Register

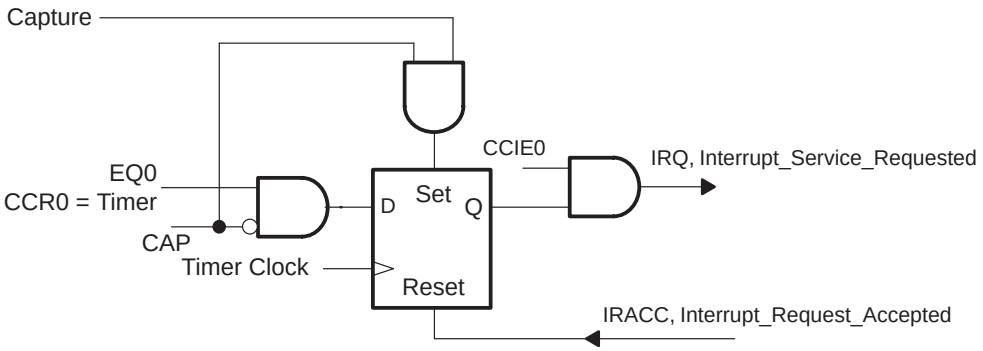
Two interrupt vectors are associated with the 16-bit Timer_A module:

- CCR0 interrupt vector (highest priority)
- TAIV interrupt vector for flags CCIFG1–CCIFGx and TAIFG.

11.6.4.1 CCR0 Interrupt Vector

The interrupt flag associated with capture/compare register CCR0, as shown in Figure 11–30, is set if the timer value is equal to the compare register value.

Figure 11–30. Capture/Compare Interrupt Flag



Capture/compare register 0 has the highest Timer_A interrupt priority, and uses its own interrupt vector.

11.6.4.2 Vector Word, TAIFG, CCIFG1 to CCIFG4 Flags

The CCIFGx (other than CCIFG0) and TAIFG interrupt flags are prioritized and combined to source a single interrupt as shown in Figure 11–31. The interrupt vector register TAIV (shown in Figure 11–32) is used to determine which flag requested an interrupt.

Figure 11–31. Schematic of Capture/Compare Interrupt Vector Word

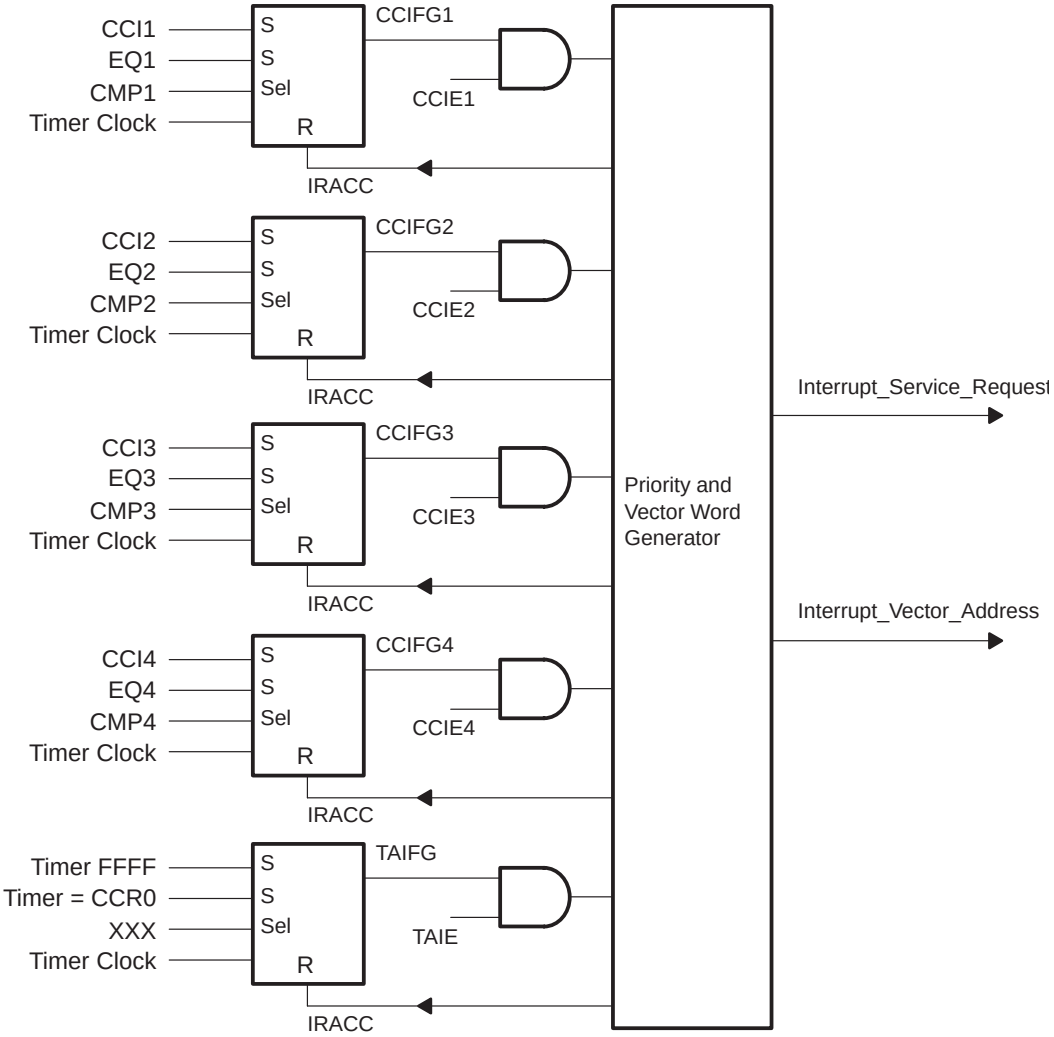
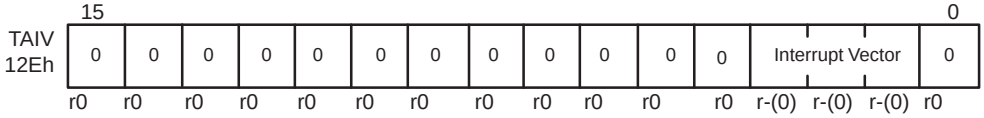


Figure 11–32. Vector Word Register



The flag with the highest priority generates a number from 2 to 12 in the TAIV register as shown in Table 11–9. (If the value of the TAIV register is 0, no interrupt is pending.) This number can be added to the program counter to automatically enter the appropriate software routine without the need for reading and evaluating the interrupt vector. The software example in section 11.6.4.3 shows this technique.

Table 11–9. Vector Register TAIV Description

Interrupt Priority	Interrupt Source	Short Form	Vector Register TAIV Contents
Highest†	Capture/compare 1	CCIFG1	2
	Capture/compare 2	CCIFG2	4
	Capture/compare 3	CCIFG3	6
	Capture/compare 4	CCIFG4	8
	Timer overflow	TAIFG	10
	Reserved		12
Lowest	Reserved		14
	No interrupt pending		0

† Highest pending interrupt other than CCIFG0. CCIFG0 is always the highest priority Timer_A interrupt.

Accessing the TAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, then another interrupt will be immediately generated after servicing the initial interrupt. For example, if both CCIFG2 and CCIFG3 are set, when the interrupt service routine accesses the TAIV register (either by reading it or by adding it directly to the PC), CCIFG2 will be reset automatically. After the RETI instruction of the interrupt service routine is executed, the CCIFG3 flag will generate another interrupt.

Note: Writing to Read-Only Register TAIV

Register TAIV should not be written to. If a write operation to TAIV is performed, the interrupt flag of the highest-pending interrupt is reset. Therefore, the requesting interrupt event is missed. Additionally, writing to this read-only register results in increased current consumption as long as the write operation is active.

11.6.4.3 Timer Interrupt Vector Register, Software Example

The following software example describes the use of vector word TAIV and the handling overhead. The numbers at the right margin show the necessary cycles for every instruction. The example is written for continuous mode: the time difference to the next interrupt is added to the corresponding compare register.

```
; Software example for the interrupt part          Cycles
;
; Interrupt handler for Capture/Compare Module 0.
; The interrupt flag CCIFG0 is reset automatically
;
TIMMOD0   ...      ; Start of handler Interrupt latency  6
          RETI                                           5
;
; Interrupt handler for Capture/Compare Modules 1 to 4.
; The interrupt flags CCIFGx and TAIFG are reset by
; hardware. Only the flag with the highest priority
; responsible for the interrupt vector word is reset.
TIM_HND   $                ; Interrupt latency          6
          ADD   &TAIV,PC   ; Add offset to Jump table   3
          RETI             ; Vector 0: No interrupt     5
          JMP   TIMMOD1    ; Vector 2: Module 1         2
```

```
          JMP    TIMMOD2      ; Vector 4: Module 2        2
          JMP    TIMMOD3      ; Vector 6: Module 3        2
          JMP    TIMMOD4      ; Vector 8: Module 4        2
;
; Module 5. Timer Overflow Handler: the Timer Register is
; expanded into the RAM location TIMEXT (MSBs)
;
TIMOVH                      ; Vector 10: TIMOV Flag
          INC    TIMEXT       ; Handle Timer Overflow     4
          RETI                                            5
;
TIMMOD2                     ; Vector 4: Module 2
          ADD    #NN,&CCR2    ; Add time difference       5
          ...                 ; Task starts here
          RETI                ; Back to main program      5
;
;
TIMMOD1                     ; Vector 2: Module 1
          ADD    #MM,&CCR1    ; Add time difference       5
          ...                 ; Task starts here
          RETI                ; Back to main program      5
; If all five CCR registers are not implemented on a
; device, the interrupt vectors for the register that are
; present must still be handled.

TIMMOD4
          RETI            ; Simply return                 5

; The Module 3 handler shows a way to look if any other
; interrupt is pending: 5 cycles have to be spent, but
; 9 cycles may be saved if another interrupt is pending
;
TIMMOD3                   ; Vector 6: Module 3
          ADD    #PP,&CCR3; Add time difference           5
          ...              ; Task starts here
          JMP    TIM_HND   ; Look for pending interrupts  2
;
          .SECT "VECTORS",0FFF0h   ; Interrupt Vectors
; The vector address may be different for different
; devices.
;
          .WORD  TIM_HND   ; Vector for Capture/Compare
                           ; Module 1..4 and timer overflow
                           ; TAIFG
          .WORD  TIMMOD0   ; Vector for Capture/Compare
                           ; Module 0
```

If the FLL is turned off, then two additional cycles need to be added for a synchronous start of the CPU and system clock MCLK.

The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles (but not the task handling itself), as described:

- ❏ Capture/compare block CCR0 — 11 cycles
- ❏ Capture/compare blocks CCR1 to CCR4 — 16 cycles
- ❏ Timer overflow TAIFG — 14 cycles

11.6.4.4 Timing Limits

With the TAIV register and the previous software, the shortest repetitive time distance t_{CRmin} between two events using a compare register is:

$$t_{CRmin} = t_{taskmax} + 16 \times t_{cycle}$$

With: $t_{taskmax}$ Maximum (worst case) time to perform the task during the interrupt routine (for example, incrementing a counter)

t_{cycle} Cycle time of the system frequency MCLK

The shortest repetitive time distance t_{CLmin} between two events using a capture register is:

$$t_{CLmin} = t_{taskmax} + 16 \times t_{cycle}$$

11.7 Timer_A UART

The Timer_A is uniquely capable of implementing a UART function, with the following features:

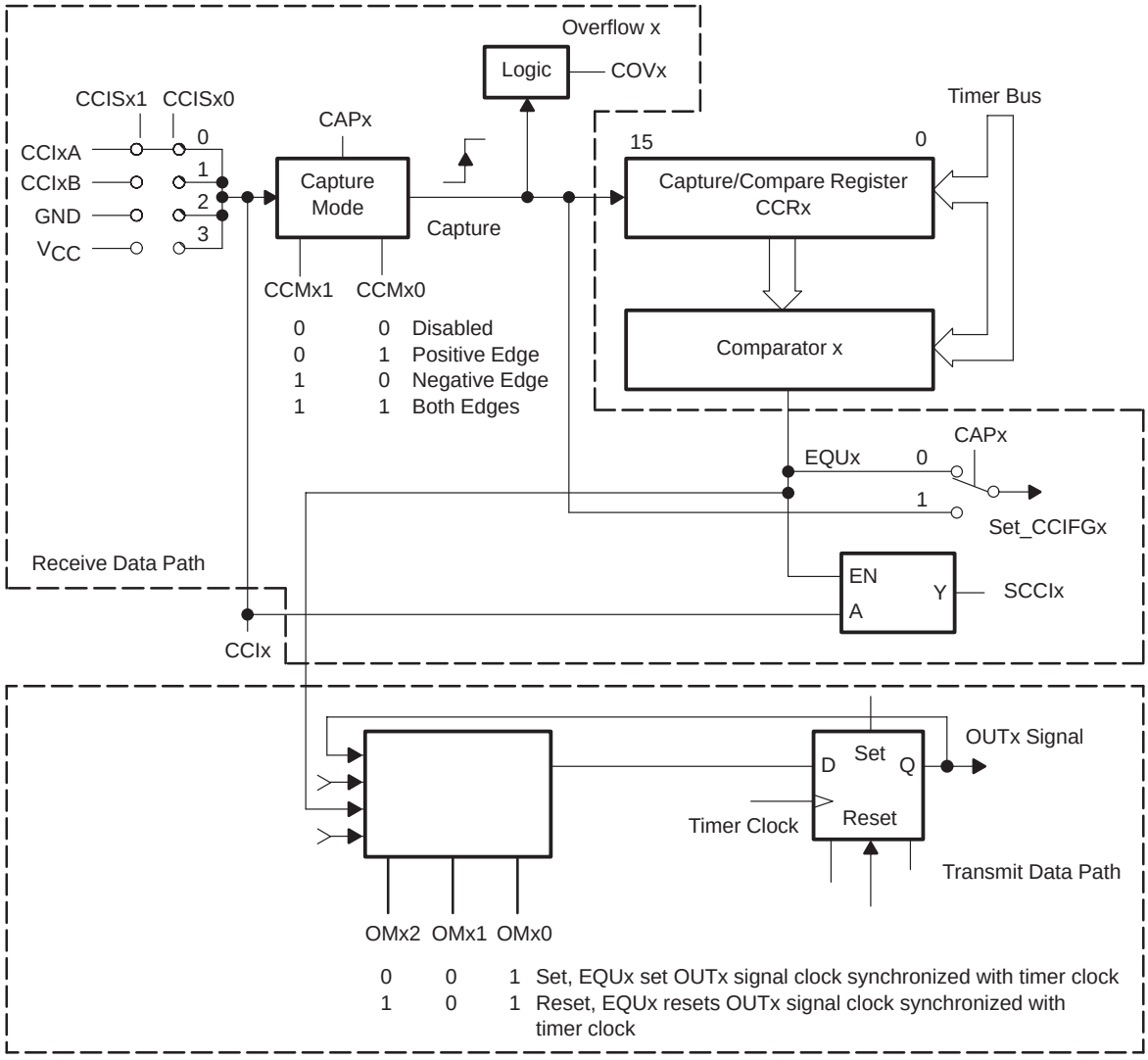
- ❑ Automatic start-bit detection – even from ultralow-power modes
- ❑ Hardware baud-rate generation
- ❑ Hardware latching of RXD and TXD data
- ❑ Baud rates of 75 to 115,200 baud
- ❑ Full-duplex operation

This UART implementation is different from other microcontroller implementations where a UART may be implemented with general-purpose I/O and manual bit manipulation via software polling. Those implementations require great CPU overhead and therefore increase power consumption and decrease the usability of the CPU.

The transmit feature uses one compare function to shift data through the output unit to the selected pin. The baud rate is ensured by reconfiguring the compare data with each interrupt.

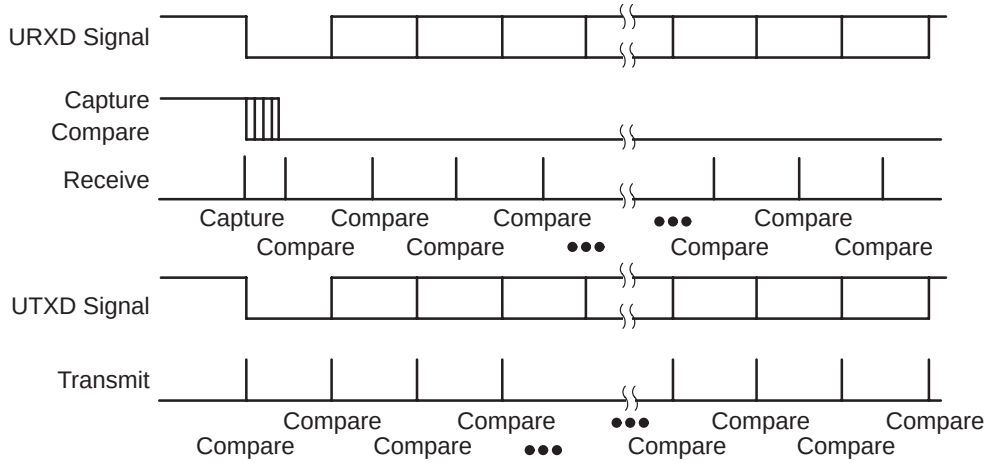
The receive feature uses one capture/compare function to shift pin data into memory through bit SCCIx. The receive start time is recognized by capturing the timer data with the negative edge of the input signal. The same capture/compare block is then switched to compare mode and the receive bits are latched automatically with the EQUx signal. The interrupt routine collects the bits for later software processing. Figure 11–33 illustrates the UART implementation.

Figure 11–33. UART Implementation



One capture/compare block is used when half-duplex communication mode is desired. Two capture/compare blocks are used for full-duplex mode. Figure 11–34 illustrates the capture/compare timing for the UART.

Figure 11–34. Timer_A UART Timing



A complete application note including connection diagrams and complete software listing may be found at www.ti.com/sc/msp430.

Chapter 12

USART Peripheral Interface, UART Mode

The universal synchronous/asynchronous receive/transmit (USART) serial-communication peripheral supports two serial modes with one hardware configuration. These modes shift a serial bit stream in and out of the MSP430 at a programmed rate or at a rate defined by an external clock. The first mode is the universal asynchronous receive/transmit (UART) communication protocol; the second is the serial peripheral interface (SPI) protocol (discussed in Chapter 13).

Bit SYNC in control register UCTL selects the required mode:

SYNC = 0:	UART – asynchronous mode selected
SYNC = 1:	SPI – synchronous mode selected

This chapter addresses the UART mode.

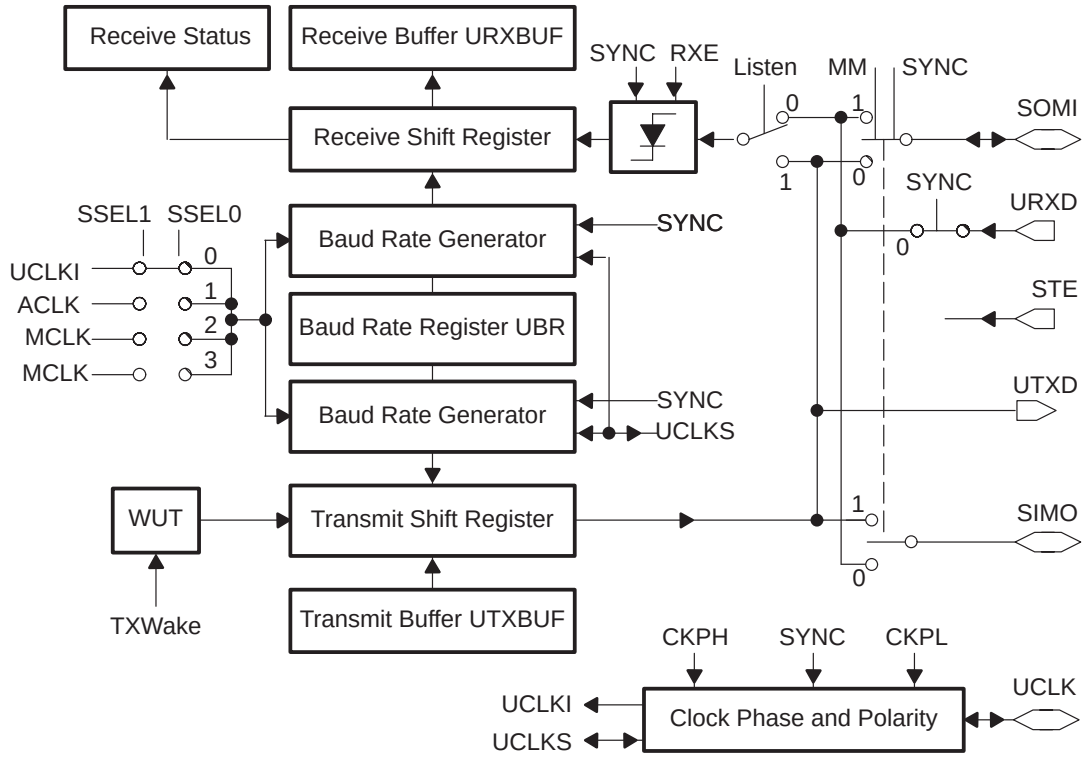
Topic **Page**

12.1 USART Peripheral Interface 12-2
12.2 USART Peripheral Interface, UART Mode 12-3
12.3 Asynchronous Operation 12-4
12.4 Interrupt and Enable Functions 12-11
12.5 Control and Status Registers 12-15
12.6 Utilizing Features of Low-Power Modes 12-23
12.7 Baud Rate Considerations 12-26

12.1 USART Peripheral Interface

The USART peripheral interface connects to the CPU as a byte peripheral module. It connects the MSP430 to the external system environment with three or four external pins. Figure 12–1 shows the USART peripheral interface module.

Figure 12–1. Block Diagram of USART



12.2 USART Peripheral Interface, UART Mode

The USART peripheral interface is a serial channel that shifts a serial bit stream of 7 or 8 bits in and out of the MSP430. The UART mode is chosen when control bit SYNC in the USART control register (UCTL) is reset.

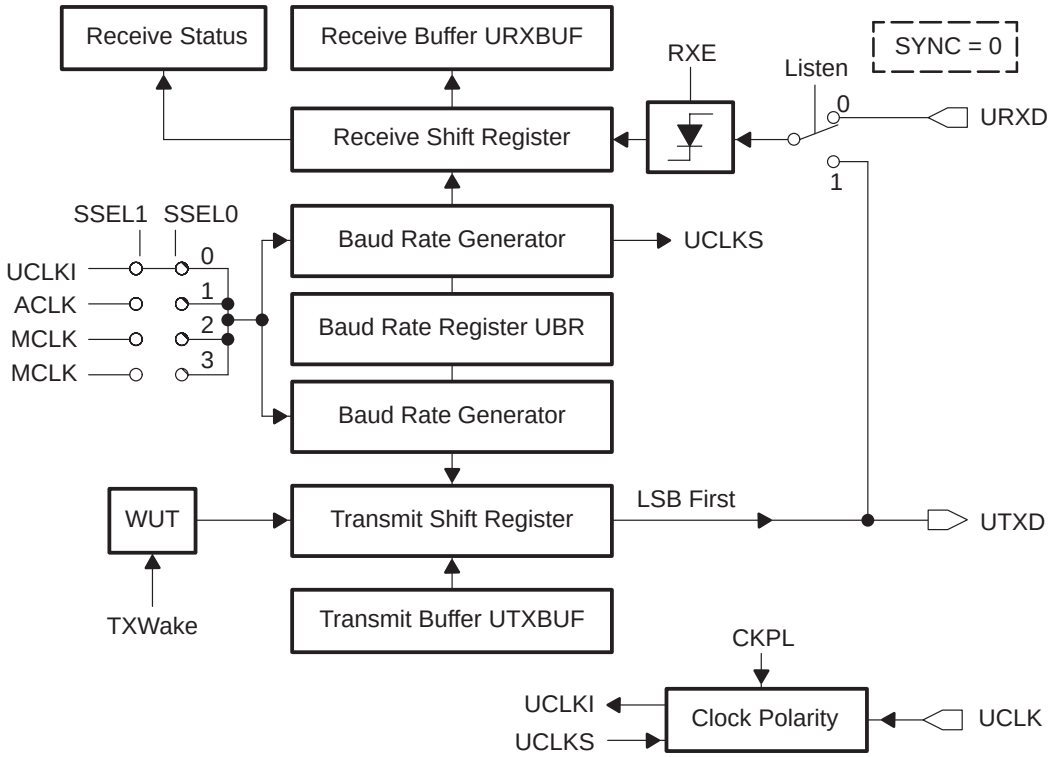
12.2.1 UART Serial Asynchronous Communication Features

Some of the UART features include:

- Asynchronous formats that include idle line/address bit-communication protocols
- Two shift registers that shift a serial data stream into URXD and out of UTXD
- Data that is transmitted/received with the LSB first
- Programmable transmit and receive bit rates
- Status flags

Figure 12–2 shows the USART in UART mode.

Figure 12–2. Block Diagram of USART – UART Mode



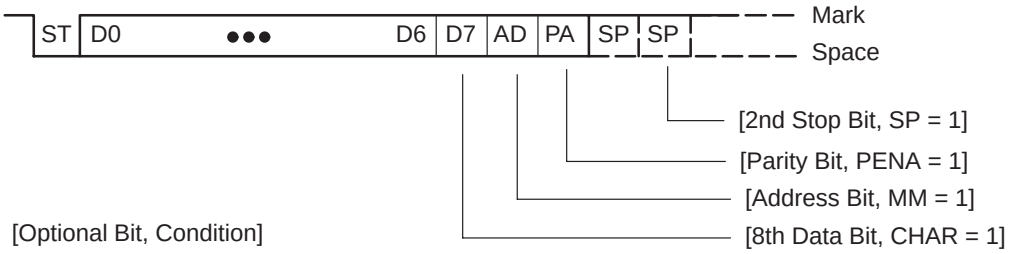
12.3 Asynchronous Operation

In the asynchronous mode, the receiver synchronizes itself to frames but the external transmitting and receiving devices do not use the same clock source; the baud rate is generated locally.

12.3.1 Asynchronous Frame Format

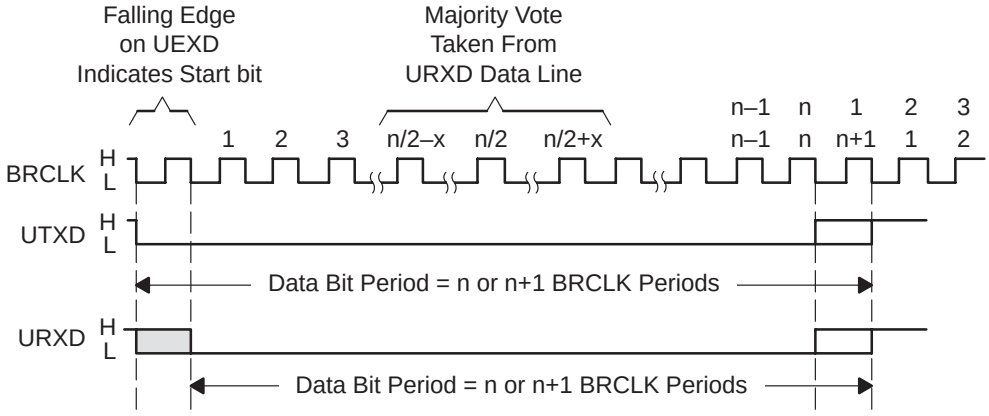
The asynchronous frame format, shown in Figure 12–3, consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit in address bit mode, and one or two stop bits. The bit period is defined by the selected clock source and the data in the baud rate registers.

Figure 12–3. Asynchronous Frame Format



The receive (RX) operation is initiated by the receipt of a valid start bit. It begins with a negative edge at URXD, followed by the taking of a majority vote from three samples where two of the samples must be zero. These samples occur at n/2–X, n/2, and n/2+X of the BRCLK periods following the negative edge. This sequence provides false start-bit rejection, and also locates the center of the bits in the frame, where the bits can be read on a majority basis. The timing of X is $^1/_{32}$ to $^1/_{63}$ times that of the BRCLK, depending on the division rate of the baud rate generator and provides complete coverage of at least two BRCLK periods. Figure 12–4 shows an asynchronous bit format.

Figure 12–4. Asynchronous Bit Format. Example for n or n + 1 Clock Periods



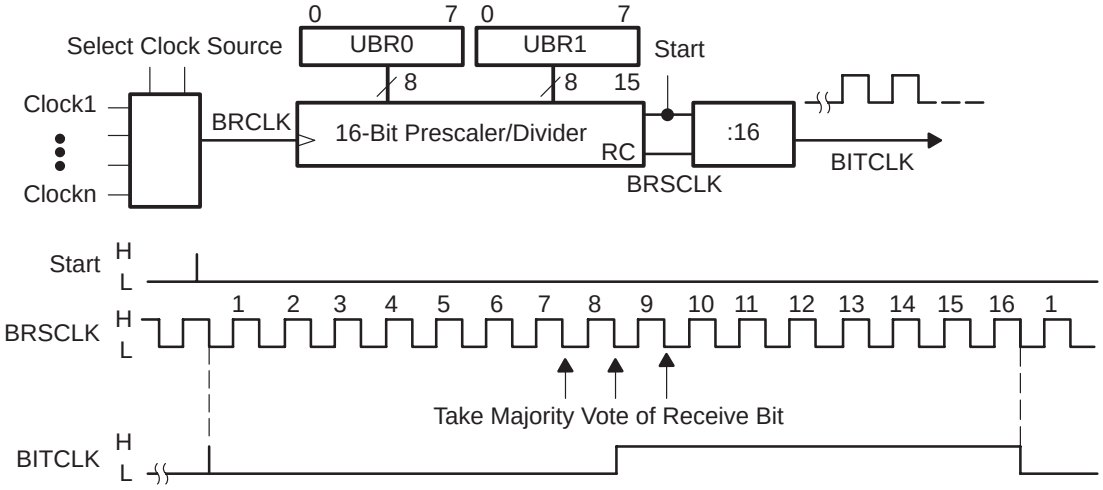
12.3.2 Baud Rate Generation in Asynchronous Communication Format

Baud rate generation in the MSP430 differs from other standard serial-communication interface implementations.

12.3.2.1 Typical Baud Rate Generation

Typical baud-rate generation uses a prescaler from any clock source and a fixed, second-clock divider that is usually divide-by-16. Figure 12–5 shows a typical baud-rate generation.

Figure 12–5. Typical Baud-Rate Generation Other Than MSP430



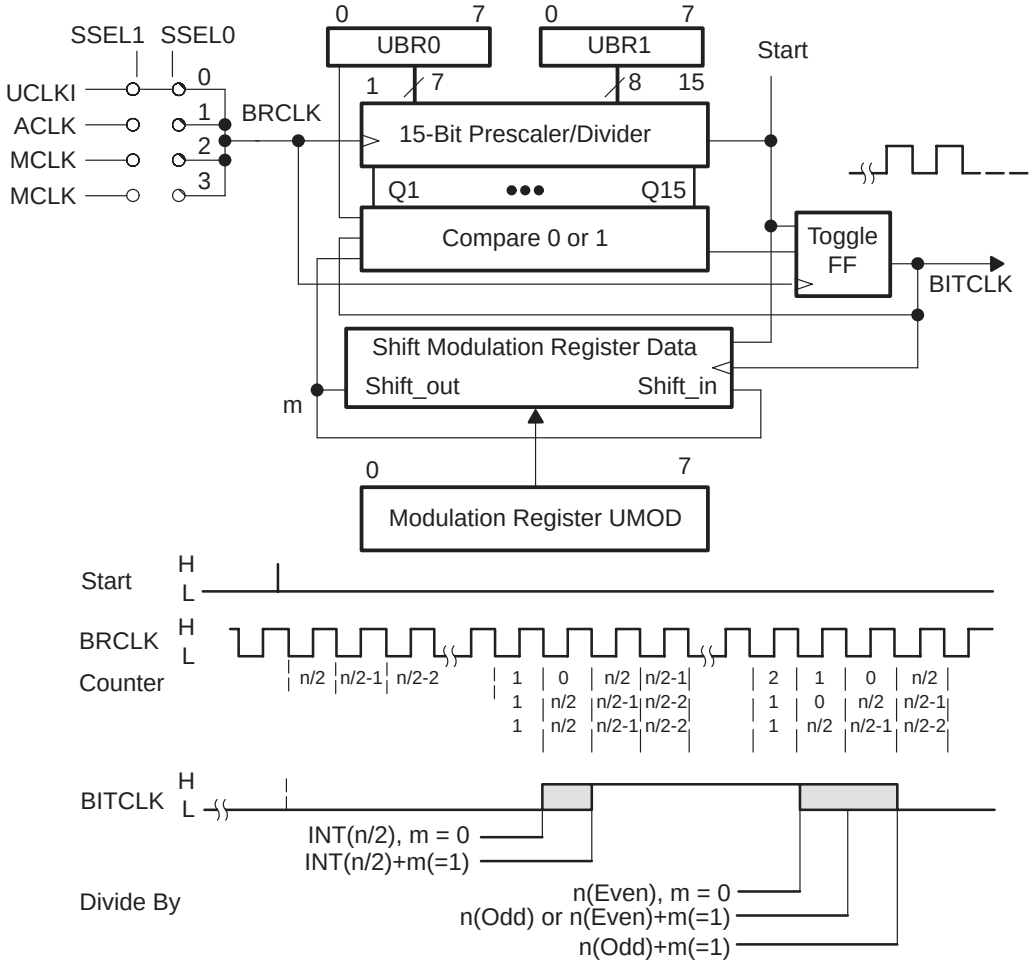
$$\text{Baud rate} = \frac{\text{BRCLK}}{n \times 16}$$

Typical baud-rate schemes often require specific crystal frequencies or cannot generate some baud rates required by some applications. For example, division factors of 18 are not possible, nor are noninteger factors such as 13.67.

12.3.2.2 MSP430 Baud Rate Generation

The MSP430 baud rate generator uses one prescaler/divider and a modulator as shown in Figure 12–6. This combination works with crystals whose frequencies are not multiples of the standard baud rates, allowing the protocol to run at maximum baud rate with a watch crystal (32,768 Hz). This technique results in power advantages because sophisticated, MSP430 low-power operations are possible.

Figure 12–6. MSP430 Baud Rate Generation. Example for n or n + 1 Clock Periods



The modulation register LSB is first used for modulation, which begins with the start bit. A set modulation bit increases the division factor by one.

Example 12–1. 4800 Baud

Assuming a clock frequency of 32,768 Hz for the BRCLK signal and a required baud rate of 4800, the division factor is 6.83. The baud rate generation in the MSP430 USART uses a factor of six plus a modulation register load of 6Fh (0110 1111). The divider runs the following sequence: 7 – 7 – 7 – 7 – 6 – 7 – 7 – 6 and so on. The sequence repeats after all eight bits of the modulator are used.

Example 12–2. 19,200 Baud

Assuming a clock frequency of 1.04 MHz (32 × 32,768 Hz) for the BRCLK signal and a required baud rate of 19,200, the division factor is 54.61. The baud rate generation in the MSP430 USART uses a factor of 54 (36h) plus a modulation register load of 0D5h. The divider runs the following sequence: 55 – 54 – 55 – 54 – 55 – 54 – 55 – 55, and so on. The sequence repeats after all eight bits of the modulator are used.

12.3.3 Asynchronous Communication Formats

The USART module supports two multiprocessor communication formats when asynchronous mode is used. These formats can transfer information between many microcomputers on the same serial link. Information is transferred as a block of frames from a particular source to one or more destinations. The USART has features that identify the start of blocks and suppress interrupts and status information from the receiver until a block start is identified. In both multiprocessor formats, the sequence of data exchanged with the USART module is based on data polling, or on the use of the receive interrupt features.

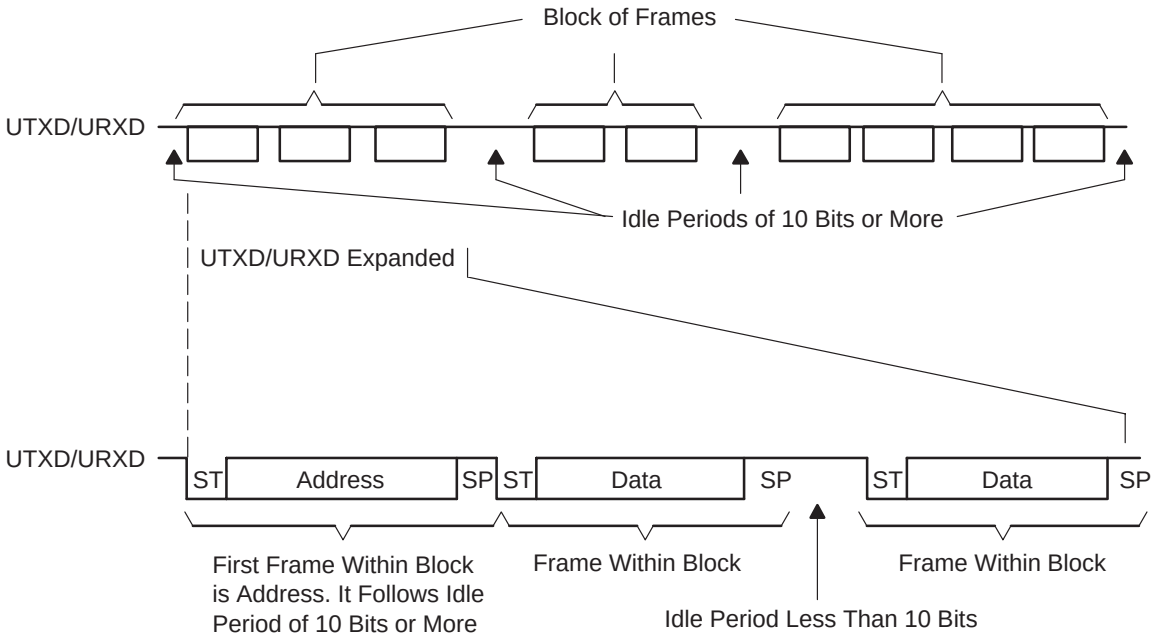
Both of the asynchronous multiprocessor formats—idle-line and address-bit—allow efficient data transfer between multiple communication systems. They can also minimize the activity of the system to save current consumption or processing resources.

The control register bit MM defines the address bit or idle-line multiprocessor format. Both use the wake-up-on-transfer mode by activating the TXWake bit (address feature function) and RXWake bit. The URXWIE and URXIE bits control the transmit and receive features of these asynchronous communication formats.

12.3.4 Idle-Line Multiprocessor Format

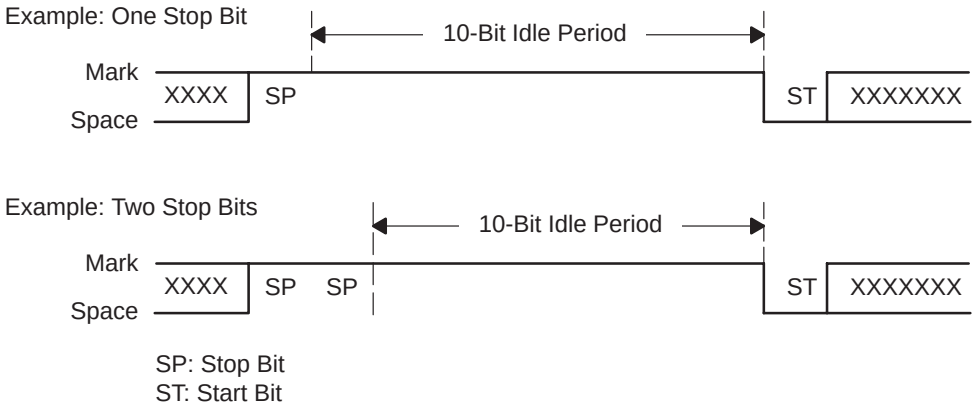
In the idle-line multiprocessor format, shown in Figure 12–7, blocks of data are separated by an idle time. An idle-receive line is detected when ten or more 1s in a row are received after the first stop bit of a character.

Figure 12–7. Idle-Line Multiprocessor Format



When two stop bits are used for the idle line, as shown in Figure 12–8, the second one is counted as the first mark bit of the idle period. The first character received after an idle period is an address character. The RXWake bit can be used as an address tag for the character. In the idle-line multiprocessor format, the RXWake bit is set when a received character is an address character and is transferred into the receive buffer.

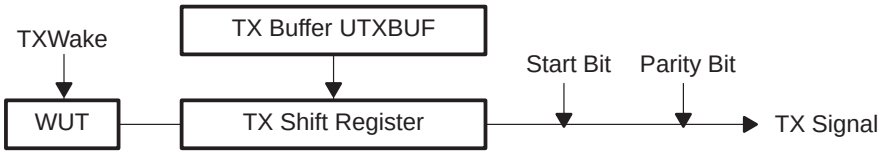
Figure 12–8. USART Receiver Idle Detect



Normally, if the USART URXWIE bit is set in the receive control register, characters are assembled as usual by the receiver. They are not, however, transferred to the receiver buffer, URXBUF, nor are interrupts generated. When an address character is received, the receiver is temporarily activated to transfer the character to URXBUF and to set the URXIFG interrupt flag. Applicable error status flags are set. The application software can validate the received address. If there is a match, the application software further processes the data and executes the operation. If there is no match, the processor waits for the next address character to arrive. The URXWIE bit is not modified by the USART: it must be modified manually to receive nonaddress or address characters.

In idle-line multiprocessor format, a precise idle period can be generated to create efficient address character identifiers. The wake-up temporary (WUT) flag is an internal flag and is double-buffered with TXWake. When the transmitter is loaded from UTXBUF, WUT is loaded from TXWake, and the TXWake bit is reset as shown in Figure 12–9.

Figure 12–9. Double-Buffered WUT and TX Shift Register



The following procedure sends out an idle frame to identify an address character:

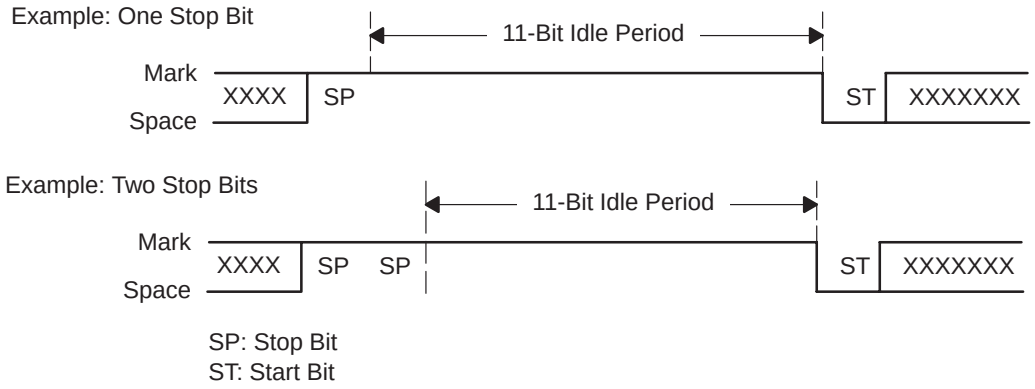
1) Set the TXWake bit and then write any word (don't care) to the UTXBUF (UTXIFG must be set).

   When the transmitter shift register is empty, the contents of UTXBUF are shifted to the transmit shift register and the TXWake value is shifted to WUT.

2) Set bit WUT, which suppresses the start, data, and parity bits and transmits an idle period of exactly 11 bits, as shown in Figure 12–10.

   The next data word, shifted out of the serial port after the address-character identifying idle period, is the second word written to the UTXBUF after the TXWake bit has been set. The first data word written is suppressed while the address identifier is sent out and ignored thereafter. Writing the first don't care word to UTXBUF is necessary to shift the TXWAKE bit to WUT and generate an idle-line condition.

Figure 12–10. USART Transmitter Idle Generation

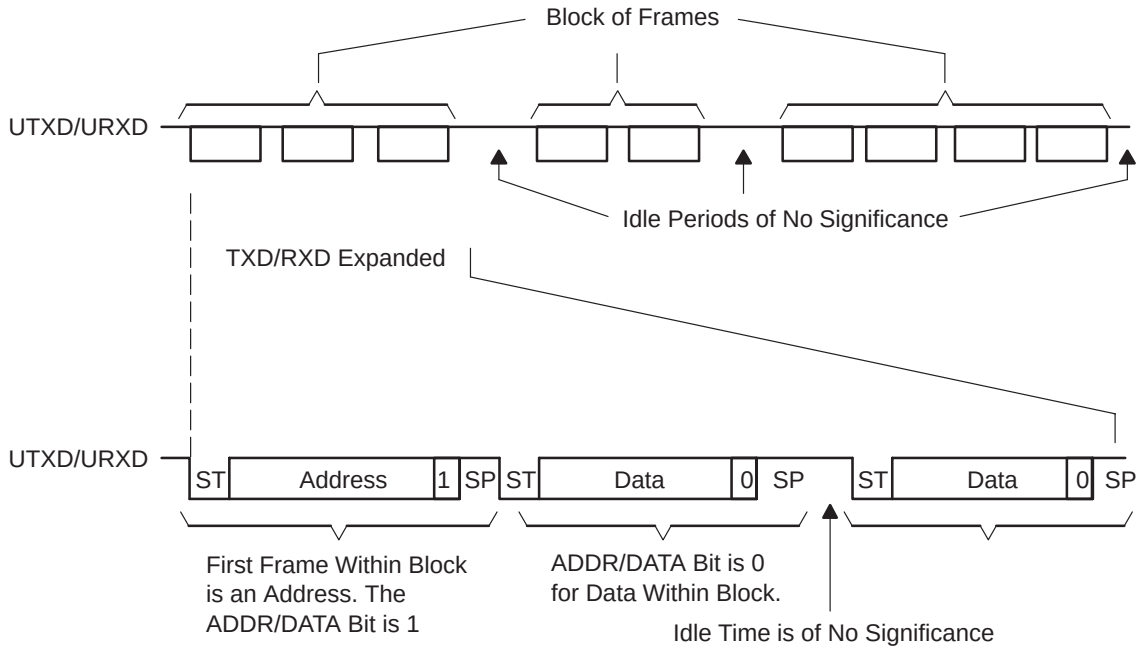


12.3.5 Address-Bit Multiprocessor Format

In the address-bit multiprocessor format shown in Figure 12–11, characters contain an extra bit used as an address indicator. The first character in a block of data carries an address bit which indicates that the character is an address. The RXWake bit is set when a received character is an address character. It is transferred into the receive buffer (receive conditions are true).

Usually, if the USART URXWIE bit is set, data characters are assembled by the receiver but are not transferred to the receiver buffer URXBUF, nor are interrupts generated. When a character that has an address bit set is received, the receiver is temporarily activated to transfer the character to URXBUF and to set the URXIFG. Error status flags are set as applicable. The application software processes the succeeding operation to optimize resource handling or reduce current consumption. The application software can validate the received address. If there is a match, the processor can read the remainder of the data block. If there is not a match, the processor waits for the next address character to arrive.

Figure 12–11. Address-Bit Multiprocessor Format



In the address-bit multiprocessor mode, the address bit of a character can be controlled by writing to the TXWake bit. The value of the TXWake bit is loaded into the address bit of that character each time a character is transferred from transmit buffer UTXBUF to the transmitter. The TXWake bit is then cleared by the USART.

12.4 Interrupt and Enable Functions

The USART peripheral interface serves two main interrupt sources for transmission and reception. Two interrupt vectors serve receive and transmit events.

The interrupt control bits and flags and enable bits of the USART peripheral interface are located in the SFR registers. They are discussed in Table 12–1. See the peripheral file map in Appendix A for the exact bit locations.

Table 12–1. USART Interrupt Control and Enable Bits – UART Mode

Receive interrupt flag	URXIFG	Initial state reset (by PUC/SWRST)
Receive interrupt enable	URXIE	Initial state reset (by PUC/SWRST)
Receive enable (see note)	URXE	Initial state reset (by PUC)
Transmit interrupt flag	UTXIFG	Initial state set (by PUC/SWRST)
Transmit interrupt enable	UTXIE	Initial state reset (by PUC/SWRST)
Transmit enable.(see note)	UTXE	Initial state reset (by PUC)

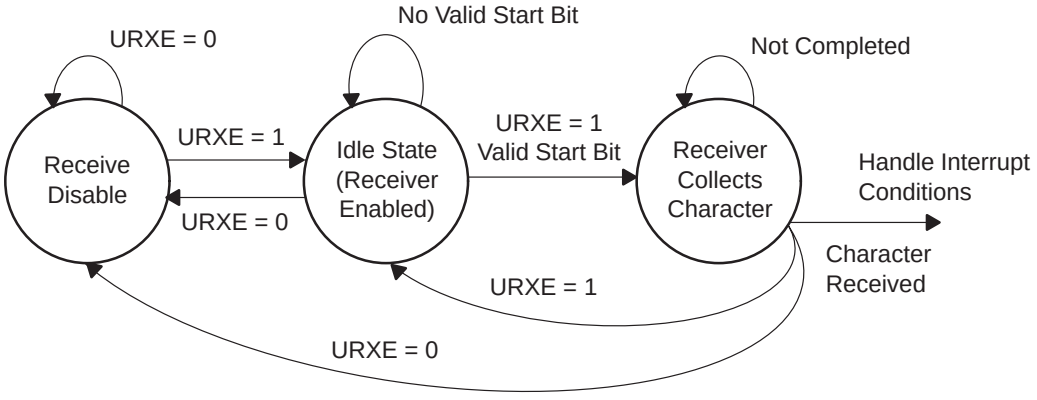
Note: Different for SPI mode, see Chapter 13.

The USART receiver and transmitter operate independently, but use the same baud rate.

12.4.1 USART Receive Enable Bit

The receive enable bit URXE, shown in Figure 12–12, enables or disables receipt of the bit stream on the URXD data line. Disabling the USART receiver stops the receive operation after completion of receiving the character, or stops immediately if no receive operation is active. Start-bit detection is also disabled.

Figure 12–12. State Diagram of Receiver Enable



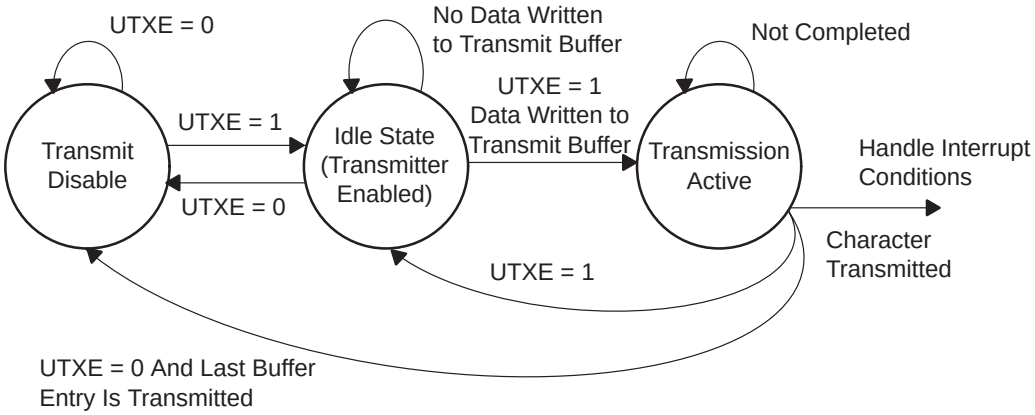
Note: URXE Reenabled, UART Mode

Because the receiver is completely disabled, reenabling the receiver is asynchronous to any data stream on the communication line. Synchronization can be performed by looking for an idle line condition before receiving a character.

12.4.2 USART Transmit Enable Bit

The transmit enable bit UTXE, shown in Figure 12–13, enables or disables a character transmission on the serial-data line. If this bit is reset, the transmitter is disabled but any active transmission does not halt until the data in the transmit shift register and the transmit buffer are transmitted. Data written to the transmit buffer before UTXE has been reset may be modified or overwritten—even after UTXE is reset—until it is shifted to the transmit shift register. For example, if software writes a byte to the transmit buffer and then resets UTXE, the byte written to the transmit buffer will be transmitted and may be modified or overwritten until it is transferred into the transmit shift register. However, after the byte is transferred to the transmit shift register, any subsequent writes to UTXBUF while UTXE is reset will not result in transmission, but UTXBUF will be updated with the new value.

Figure 12–13. State Diagram of Transmitter Enable



When UTXE is reset and the current transmission is completed, new data written to the transmit buffer will not be transmitted. Once the UTXE bit is set, the data in the transmit buffer are immediately loaded into the transmit shift register and character transmission is started.

Note: Writing to UTXBUF, UART Mode

Data should never be written to transmit buffer UTXBUF when the buffer is not ready and when the transmitter is enabled (UTXE is set). Otherwise, the transmission will have errors.

Note: Write to UTXBUF/Reset of Transmitter, UART Mode

Disabling the transmitter should be done only if all data to be transmitted has been moved to the transmit shift register.

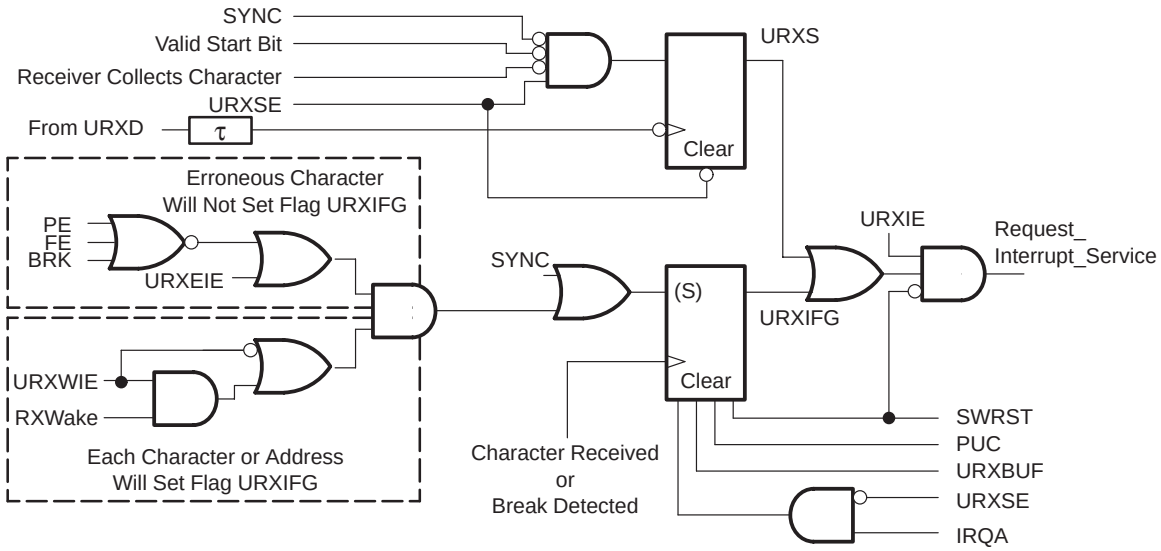
```
MOV.B  #....,&UTXBUF
BIC.B  #UTXE,&ME2        ; If BITCLK < MCLK then the
                         ; transmitter might be stopped
                         ; before the buffer is loaded
                         ; into the transmitter shift
                         ; register
```

12.4.3 USART Receive Interrupt Operation

In the receive interrupt operation, shown in Figure 12–14, the receive interrupt flag URXIFG is set or is unchanged each time a character is received and loaded into the receive buffer:

- Erroneous characters (parity, frame, or break error) do not set interrupt flag URXIFG when URXEIE is reset: URXIFG is unchanged.
- All types of characters (URXWIE = 0), or only address characters (URXWIE = 1), set the interrupt flag URXIFG. When URXEIE is set, erroneous characters can also set the interrupt flag URXIFG.

Figure 12–14. Receive Interrupt Operation



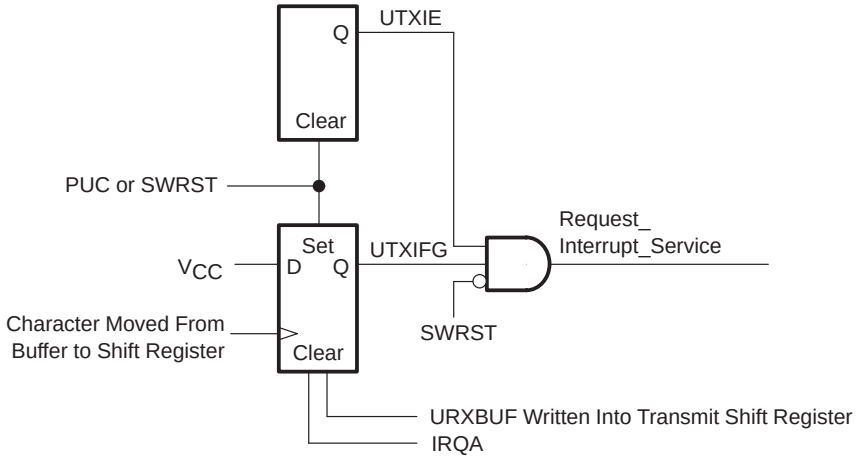
URXIFG is reset by a system reset PUC signal, or with a software reset (SWRST). URXIFG is reset automatically if the interrupt is served (URXSE = 0) or the receive buffer URXBUF is read. A set receive interrupt flag URXIFG indicates that an interrupt event is waiting to be served. A set receive interrupt enable bit URXIE enables serving a waiting interrupt request. Both the receive interrupt flag URXIFG and the receive interrupt enable bit URXIE are reset with the PUC signal and a SWRST.

Signal URXIFG can be accessed by the software, whereas signal URXS cannot. When both interrupt events—character receive action and receive start detection—are enabled by the software, the flag URXIFG indicates that a character was received but the start-detect interrupt was not. Because the interrupt software handler for the receive start detection resets the URXSE bit, this clears the URXS bit and prevents further interrupt requests from URXS. The URXIFG should already be reset since no set condition was active during URXIFG latch time.

12.4.4 USART Transmit Interrupt Operation

In the transmit interrupt operation, shown in Figure 12–15, the transmit interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt request service is started or a character is written into the UTXBUF. This flag asserts a transmitter interrupt if the local (UTXIE) and general interrupt enable (GIE) bits are set. The UTXIFG is set after a system reset PUC signal, or removal of a SWRST.

Figure 12–15. Transmit Interrupt Operation



The transmit interrupt enable UTXIE bit controls the ability of the UTXIFG to request an interrupt, but does not prevent the flag UTXIFG from being set. The UTXIE is reset with a PUC signal or a software reset (SWRST) bit. The UTXIFG bit is set after a system reset PUC signal or software reset (SWRST), but the UTXIE bit is reset to ensure full interrupt-control capability.

12.5 Control and Status Registers

The USART control and status registers are byte structured and should be accessed using byte processing instructions (suffix B). Table 12–3 lists the registers and their access modes.

Table 12–2. Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL	Read/write	070h	See section 12.5.1.
Transmit control	UTCTL	Read/write	071h	See section 12.5.2.
Receive control	URCTL	Read/write	072h	See section 12.5.3.
Modulation control	UMCTL	Read/write	073h	Unchanged
Baud rate 0	UBR0	Read/write	074h	Unchanged
Baud rate 1	UBR1	Read/write	075h	Unchanged
Receive buffer	URXBUF	Read/write	076h	Unchanged
Transmit buffer	UTXBUF	Read	077h	Unchanged

All bits are random after a PUC signal, unless otherwise noted by the detailed functional description.

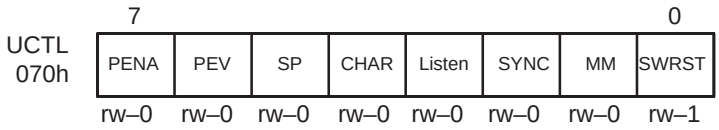
The reset of the USART peripheral interface is performed by a PUC signal or a SWRST. After a PUC signal, the SWRST bit remains set and the USART interface remains in the reset condition until it is disabled by resetting the SWRST bit.

The USART module operates in asynchronous or synchronous mode as defined by the SYNC bit. The bits in the control registers can have different functions in the two modes. All bits in this section are described with their functions in the asynchronous mode (SYNC = 0). Their functions in the synchronous mode are described in Chapter 13, *USART Peripheral Interface, SPI Mode*.

12.5.1 USART Control Register UCTL

The information stored in the USART control register (UCTL), shown in Figure 12–16, determines the basic operation of the USART module. The register bits select the communications protocol, communication format, and parity bit. All bits must be programmed according to the selected mode before resetting the SWRST bit to disable the reset.

Figure 12–16. USART Control Register UCTL



Bit 0: The USART state machines and operating flags are initialized to the reset condition (URXIFG = URXIE = UTXIE = 0, UTXIFG = 1) if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be reenabled by resetting this bit. The receive and transmit enable flags URXE and UTXE are not altered by SWRST.

The SWRST bit resets the following bits and flags: URXIE, UTXIE, URXIFG, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, and FE

The SWRST bit sets the following bits: UTXIFG, TXEPT

Bit 1: Multiprocessor mode (address/idle-line wake up)
Two multiprocessor protocols, idle-line and address-bit, are supported by the USART module. The choice of multiprocessor mode affects the operation of the automatic address decoding functions.
MM = 0: Idle-line multiprocessor protocol
MM = 1: Address-bit multiprocessor protocol
The conventional asynchronous protocol uses MM-bit reset.

Bit 2: Mode or function of USART module selected
The SYNC bit selects the function of the USART peripheral interface module. Some of the USART control bits have different functions in UART and SPI mode.
SYNC = 0: UART function is selected
SYNC = 1: SPI function is selected

Bit 3: The listen bit selects if the transmitted data is fed back internally to the receiver.
Listen = 0: No feedback
Listen = 1: Transmit signal is internally fed back to the receiver. This is commonly known as loopback mode.

Bit 4: Character length
This register bit selects the length of the character to be transmitted as either 7 or 8 bits. 7-bit characters do not use the eighth bit in URXBUF and UTXBUF. This bit is padded with *0*.
CHAR = 0: 7-bit data
CHAR = 1: 8-bit data

Bit 5: Number of stop bits
This bit determines the number of stop bits transmitted. The receiver checks for one stop bit only.
SP = 0: one stop bit
SP = 1: two stop bits

Bit 6: Parity odd/even
If the PENA bit is set (parity bit is enabled), the PEV bit defines odd or even parity according to the number of odd or even 1 bits (in both the transmitted and received characters), the address bit (address-bit multiprocessor mode), and the parity bit.
PEV = 0: odd parity
PEV = 1: even parity

Bit 7: Parity enable
If parity is disabled, no parity bit is generated during transmission or expected during reception. A received parity bit is not transferred to the URXBUF with the received data as it is not considered one of the data bits. In address-bit multiprocessor mode, the address bit is included in the parity calculation.
PEN = 0: Parity disable
PEN = 1: Parity enable

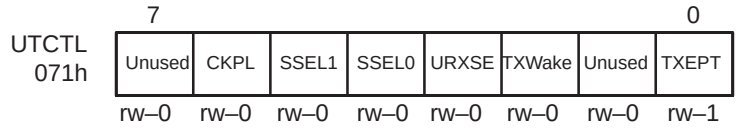
Note: Mark and Space Definitions

The mark condition is identical to the signal level in the idle state. Space is the opposite signal level: the start bit is always space.

12.5.2 Transmit Control Register UTCTL

The transmit control register (UTCTL), shown in Figure 12–17, controls the USART hardware associated with the transmit operation.

Figure 12–17. Transmitter Control Register UTCTL



Bit 0: The transmitter empty (TXEPT) flag is set when the transmitter shift register and UTXBUF are empty, and is reset when data is written to UTXBUF. It is set by a SWRST.

Bit 1: Unused

Bit 2: The TXWake bit controls the transmit features of the multiprocessor communication modes. Each transmission —started by loading the UTXBUF—uses the state of the TXWake bit to initialize the address-identification feature. It must not be cleared—the USART hardware clears this bit once it is transferred to the WUT; a SWRST also clears the TXWake bit.

Bit 3: The receive-start edge-control bit, if set, requests a receive interrupt service. For a successful interrupt service, the corresponding enable bits URXIE and GIE must be set. The advantage of this bit is that it starts the controller clock system, including MCLK, along with the interrupt service, and keeps it running by modifying the mode control bits. The USART module works with the selected MCLK even if the system is switched to a low-power mode with a disabled MCLK.

Bits 4, 5: Source select 0 and 1
The source select bit defines which clock source is used for baud-rate generation:

SSEL1, SSEL0	0	External clock, UCLKI
	1	ACLK
	2, 3	MCLK

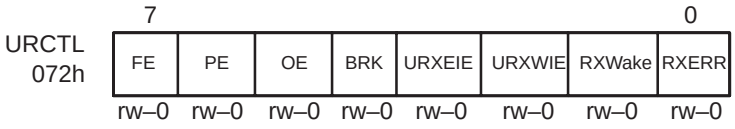
Bit 6: Clock polarity CKPL
The CKPL bit controls the polarity of the UCLKI signal.
CKPL = 0: The UCLKI signal has the same polarity as the UCLK signal.
CKPL = 1: The UCLKI signal has an inverted polarity to the UCLK signal.

Bit 7: Unused

12.5.3 Receiver Control Register URCTL

The receiver-control register (URCTL), shown in Figure 12–18, controls the USART hardware associated with the receiver operation and holds error and wake-up conditions modified by the latest character written to the receive buffer (URXBUF). Once any one of the bits FE, PE, OE, BRK, RXERR, or RXWake is set, none are reset by receiving another character. The bits are reset by accessing the receive buffer, by a USART software reset (SWRST), by a system reset PUC signal, or by an instruction.

Figure 12–18. Receiver-Control Register URCTL



Bit 0: The receive error bit (RXERR) indicates that one or more error flags (FE, PE, OE, or BRK) is set. It is not reset when the error flags are cleared by instruction.

Bit 1: Receiver wake-up detect
The RXWake bit is set when a received character is an address character and is transferred into the receive buffer.
Address-bit multiprocessor mode: RXWake is set when the address bit is set in the character received.
Idle-line multiprocessor mode: RXWake is set if an idle URXD line is detected (11 bits of mark level) in front of the received character.
RXWake is reset by accessing the receive buffer (URXBUF), by a USART software reset, or by a system-reset PUC signal.

Bit 2: The receive wake-up interrupt-enable bit (URXWIE) selects the type of character to set the interrupt flag (URXIFG):
URXWIE = 0: Each character received sets the URXIFG
URXWIE = 1: Only characters that are marked as address characters set the interrupt flag URXIFG. It operates identically in both multiprocessor modes.
The wake-up interrupt enable feature depends on the receive erroneous-character feature. See also Bit 3, URXEIE.

Bit 3: The receive erroneous-character interrupt-enable bit URXEIE selects whether an erroneous character is to set the interrupt flag URXIFG.
URXEIE = 0: Each erroneous character received does not alter the interrupt flag URXIFG.
URXEIE = 1: All characters can set the interrupt flag URXIFG as described in Table 12–4, depending on the conditions set by the URXWIE bit.

Table 12–3. Interrupt Flag Set Conditions

URXEIE	URXWIE	Char. w/Error	Char. Address	Description Flag URXIFG After a Character Is Received
0	X	1	X	Unchanged
0	0	0	X	Set
0	1	0	0	Unchanged
0	1	0	1	Set
1	0	X	X	Set (Receives all characters)
1	1	X	0	Unchanged
1	1	X	1	Set

Bit 4: The break detect bit (BRK) is set when a break condition occurs and the URXEIE bit is set. The break condition is recognized if the RXD line remains continuously low for at least 10 bits, beginning after a missing first stop bit. It is not cleared by receipt of a character after the break is detected, but is reset by a SWRST, a system reset, or by reading the URXBUF. The receive interrupt flag URXIFG is set if a break is detected.

Bit 5: The overrun error flag bit OE is set when a character is transferred into the URXBUF before the previous character is read out. The previous character is overwritten and lost. OE is reset by a SWRST, a system reset, or by reading the URXBUF.

Bit 6: The parity error flag bit PE is set when a character is received with a mismatch between the number of 1s and its parity bit, and is loaded into the receive buffer. The parity checker includes the address bit, used in the address-bit multiprocessor mode, in the calculation. The flag is disabled if parity generation and detection are not enabled. In this case the flag is read as 0. It is reset by a SWRST, a system reset, or by reading the URXBUF.

Bit 7: The framing error flag bit FE is set when a character is received with a 0 stop bit and is loaded into the receive buffer. Only the first stop bit is checked when more than one is used. The missing stop bit indicates that the start-bit synchronization is lost and the character is incorrectly framed. FE is reset by a SWRST, a system reset, or by reading the URXBUF.

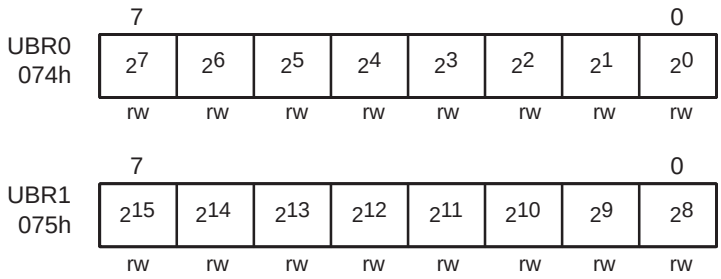
Note: Receive Status Control Bits

The receive status control bits FE, PE, OE, BRK, and RXWake are set by the hardware according to the conditions of the characters received. Once the bits are set, they remain set until the software resets them directly, or there is a reading of the receive buffer. False character interpretation or missing-interrupt capability can result in uncleared error bits.

12.5.4 Baud Rate Select and Modulation Control Registers

The baud-rate generator uses the content of the baud-rate select registers UBR0 and UBR1 shown in Figure 12–19, with the modulation control register to generate the serial data-stream bit timing.

Figure 12–19. USART Baud Rate Select Register



$$\text{Baud rate} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i=0}^{n-1} mi} \qquad \text{with UBR= [UBR1,UBR0]}$$

The baud-rate control register range is: $3 \le UBR < 0FFFFh$

Note:

Unpredictable receive and transmission occur if UBR <3.

The modulation control register, shown in Figure 12–20, ensures proper timing generation with the UBR0 and UBR01, even with crystal frequencies that are not integer multiples of the required baud rate.

Figure 12–20. USART Modulation Control Register

UMCTL 073h

7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

The timing of the running bit is expanded by one clock cycle of the baud-rate-divider input clock if bit m_i is set.

Each time a bit is received or transmitted, the next bit in the modulation control register determines the present bit timing. The first bit time in the protocol—the start bit time—is determined by UBR plus m0; the next bit is determined by UBR plus m1, and so on.

The modulation sequence is:

m0 – m1 – m2 – m3 – m4 – m5 – m6 – m7 – m0 – m1 – m2 – .....

12.5.5 Receive-Data Buffer URXBUF

The receive-data buffer (URXBUF), shown in Figure 12–21, contains previous data from the receiver shift register. Reading URXBUF resets the receive-error bits, the RXWake bit, and the interrupt flag (URXIFG).

Figure 12–21. USART Receive Data Buffer URXBUF

URXBUF 076h

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
r	r	r	r	r	r	r	r

In seven-bit length mode, the MSB of the URXBUF is always reset.

The receive data buffer is loaded with the recently-received character as described in Table 12–4, when receive and control conditions are true.

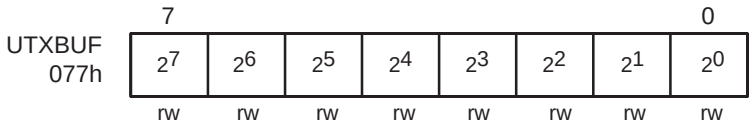
Table 12–4. Receive Data Buffer Characters

URXEIE	URXWIE	**Load URXBUF With:**	PE	FE	BRK
0	1	Error-free address characters	0	0	0
1	1	All address characters	X	X	X
0	0	Error-free characters	0	0	0
1	0	All characters	X	X	X

12.5.6 Transmit Data Buffer UTXBUF

The transmit data buffer (UTXBUF), shown in Figure 12–22, contains current data to be transmitted.

Figure 12–22. Transmit Data Buffer UTXBUF



The UTXIFG flag indicates that the UTXBUF buffer is ready to accept another character for transmission.

The transmission is initialized by writing data to UTXBUF. The transmission of this data is started immediately if the transmitter shift register is empty or is going to be empty.

Note: Writing to UTXBUF

Writing data to the transmit-data buffer must only be done if buffer UTXBUF is empty; otherwise, an unpredictable character can be transmitted.

12.6 Utilizing Features of Low-Power Modes

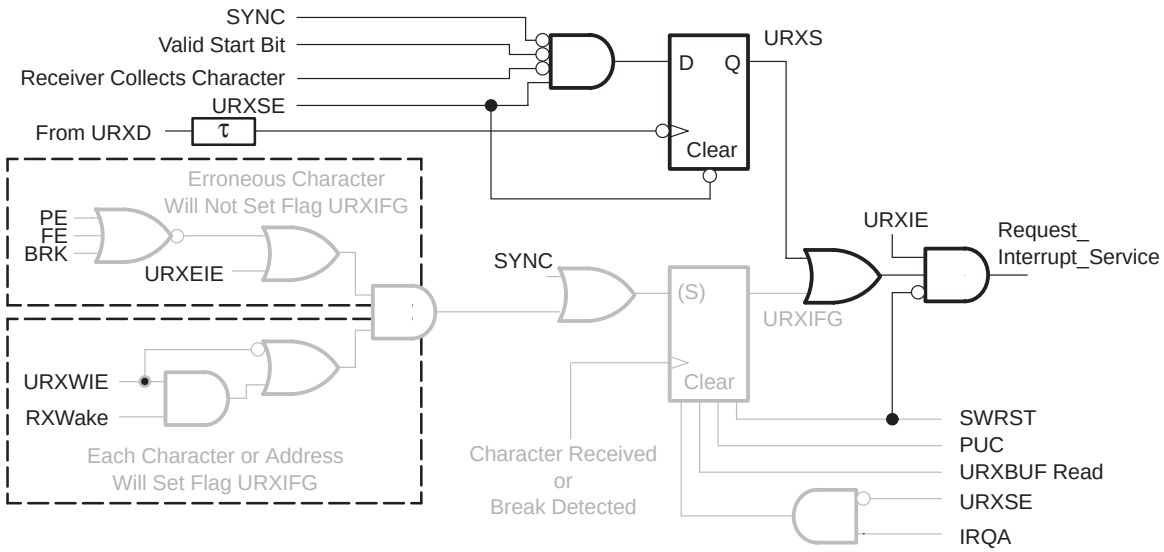
There are several functions or features of the USART that support the ultra-low power architecture of the MSP430. These include:

- Support system start up from any processor mode by sensing of UART frame-start condition
- Use the lowest input clock frequency for the required baud rate
- Support multiprocessor modes to reduce use of MSP430 resources

12.6.1 Receive-Start Operation From UART Frame

The most effective use of start detection in the receive path is achieved when the baud-rate clock runs from MCLK. In this configuration, the MSP430 can be put into a low-power mode with MCLK disabled. The receive-start condition is the negative edge from the signal on pin URXD. Each time the negative edge triggers the interrupt flag URXS, it requests a service when enable bits URXIE and GIE are set. This wakes the MSP430 and the system returns to active mode, supporting the USART transfer.

Figure 12–23. Receive-Start Conditions



Three character streams do not set the interrupt flag (URXIFG):

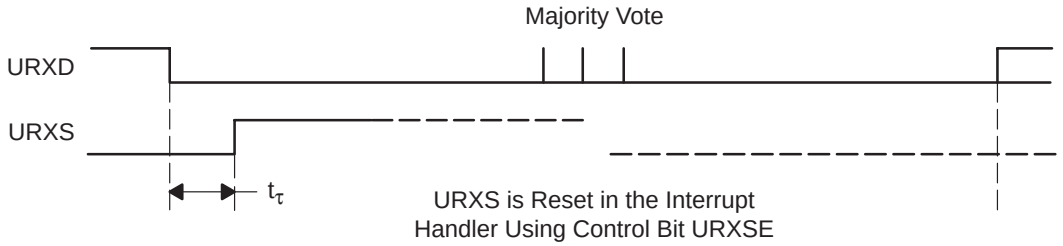
- Erroneous characters (URXEIE = 0)
- Address characters (URXWIE = 1)
- Invalid start-bit detection

The interrupt software should handle these conditions.

12.6.1.1 Start Conditions

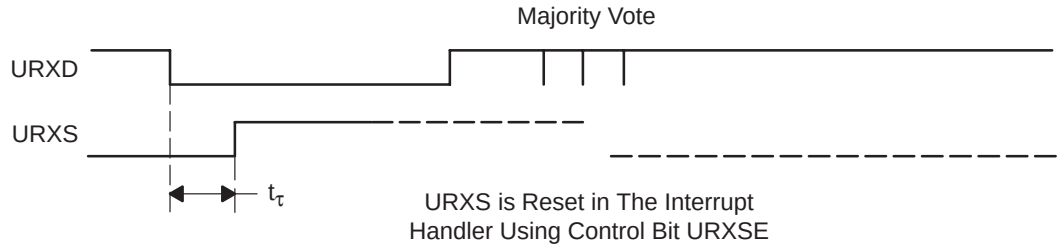
The URXD signal feeds into the USART module by first going into a deglitch circuit. Glitches cannot trigger the receive-start condition flag URXS, which prevents the module from being started from small glitches on the URXD line. Because glitches do not start the system or the USART module, current consumption is reduced in noisy environments. Figure 12–24 shows the accepted receive-start timing condition.

Figure 12–24. Receive-Start Timing Using URXS Flag, Start Bit Accepted



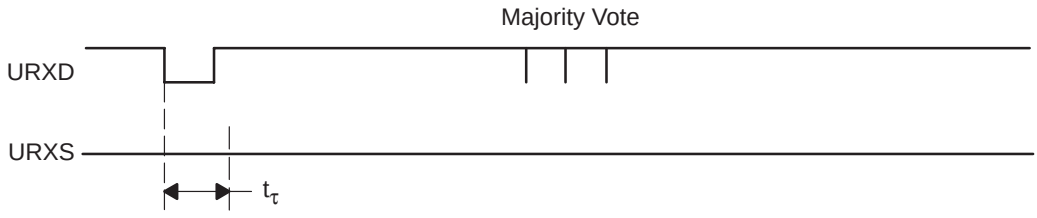
The UART stops receiving a character when the URXD signal exceeds the deglitch time t_τ but the majority vote on the signal fails to detect a start bit, as shown in Figure 12–25. The software should handle this condition and return the system to the appropriate low-power mode. The interrupt flag URXIFG is not set.

Figure 12–25. Receive Start Timing Using URXS Flag, Start Bit Not Accepted



Glitches at the URXD line are suppressed automatically and no further activity occurs in the MSP430 as shown in Figure 12–26. The data for the deglitch time t_τ is noted in the corresponding device specification.

Figure 12–26. Receive Start Timing Using URXS Flag, Glitch Suppression



The interrupt handler must reset the URXSE bit in control register UCTL to prevent further interrupt service requests from the URXS signal and to enable the basic function of the receive interrupt flag URXIFG.

```
*******************************************************
*    Interrupt handler for frame start condition and     *
*    Character receive                                   *
*******************************************************

IFG2     .EQU  3              ; URXIFG and UTXIFG in
                              ; address 3
UTCTL    .EQU  71h            ;
UTXIFG   .EQU  0              ;
URXSE    .EQU  8              ;
                              ;
URX_Int  BIT.B #URXIFG,&IFG2  ; test URXIFG signal to
         JNE   ST_COND        ; check if frame start
                              ; condition
         .....
         .....
ST_COND  BIC.B #URXSE,&UTCTL  ; clear ff/signal URXS,
                              ; stop further interrupt
                              ; requests
         BIS.B #URXSE,&UTCTl  ; Prepare FF_URXS for next
                              ; frame start bits and set
         .....                ; the conditions to run the
         .....                ; clock needed for UART RX
```

Note: Break Detect (BRK) Bit With Halted UART Clock

If the UART operates with the wake-up-on-start-condition mode and switches off the UCLK whenever a character is completely received, a communication line break cannot be detected automatically by the UART hardware. The break detection requires the baud-rate generator BRSCLK, but it is stopped upon the missing UCLK.

12.6.2 Maximum Utilization of Clock Frequency vs Baud Rate UART Mode

The current consumption increases linearly with the clock frequency. It should be kept to the minimum required to meet application conditions. Fast communication speed is needed for calibration and testing in manufacturing processes, alarm responses in critical applications, and response time to human requests for information.

The MSP430 USART can generate baud rates up to one third of the clock frequency. An additional modulation of the baud-rate timing adjusts timing for individual bits within a frame. The timing is adjusted from bit to bit to meet timing requirements even when a noninteger division is needed. Baud rates up to 4800 baud can be generated from a 32,768 Hz crystal with maximum errors of 11 percent. Standard UARTs—even with the worst maximum error (–14.6 percent)—can obtain maximum baud rates of 75 baud.

12.6.3 Support of Multiprocessor Modes for Reduced Use of MSP430 Resources

Communication systems can use multiprocessor modes with multiple-character idle-line or address-bit protocols. The first character can be a target address, a message identifier, or can have another definition. This character is interpreted by the software and, if it is of any significance to the application, the succeeding characters are collected and further activities are defined. An insignificant first character would stop activity for the processing device. This application is supported by the wake-up interrupt feature in the receive operation, and sends wake-up conditions along with a transmission. Avoiding activity on insignificant characters reduces consumption of MSP430 resources and the system can remain in the most efficient power-conserving mode.

In addition to the multiprocessor modes, rejecting erroneous characters saves MSP430 resources. This practice prevents interrupt handling of the erroneous characters. The processor waits in the most efficient power-conserving mode until a character is processed.

12.7 Baud Rate Considerations

The MSP430 baud-rate generator uses a divider and a modulator. A given crystal frequency and a required baud rate determines the required division factor N:

$$N = \frac{\text{BRCLK}}{\text{baud rate}}$$

The required division factor N usually has an integer part and a fraction. The divider in the baud rate generator realizes the integer portion of the division factor N, and the modulator meets the fractional part as closely as possible. The factor N is defined as:

$$N = UBR + \frac{1}{n}\sum_{i=0}^{n-1} mi$$

Where

N: Target division factor
UBR: 16-bit representation of registers UBR1 and UBR0
i: Actual bit in the frame
n: Number of bits in the frame
m_i: Data of the actual modulation bit

$$Baud\ rate = \frac{BRCLK}{N} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i}$$

12.7.1 Bit Timing in Transmit Operation

The timing for each individual bit in one frame or character is the sum of the actual bit timings as shown in Figure 12–27. The baud-rate generation error shown in Figure 12–28 in relation to the required ideal timing, is calculated for each individual bit. The relevant error information is the error relative to the actual bit, not the overall relative error.

Figure 12–27. MSP430 Transmit Bit Timing

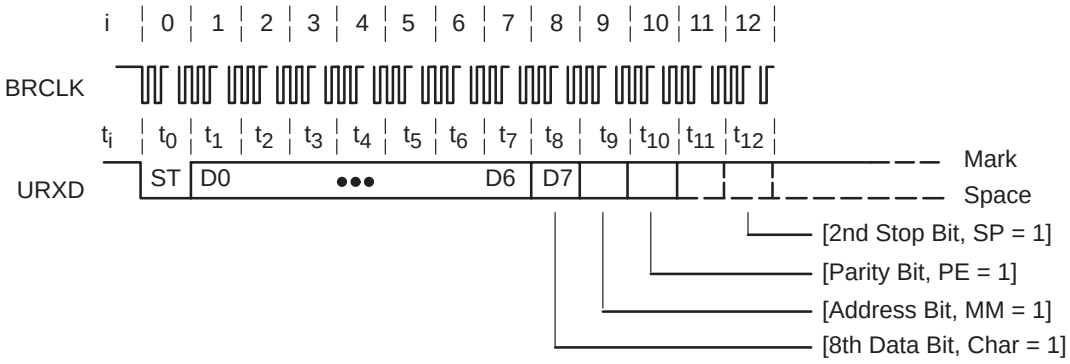
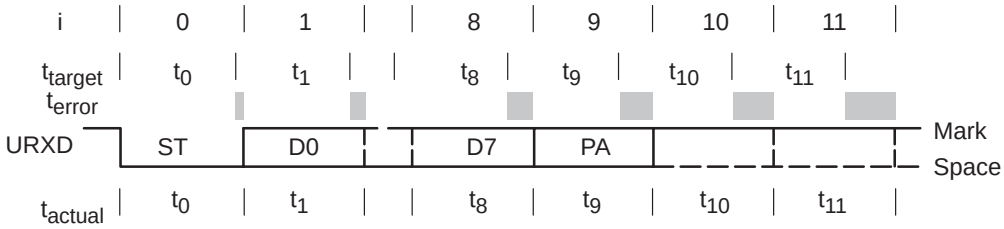


Figure 12–28. MSP430 Transmit Bit Timing Errors



Even small errors per bit (relative errors) can result in large cumulative errors. They must be considered to be cumulative, not relative. The error of an individual bit can be calculated by:

$$Error[\%] = \frac{\sum_{i=0}^{n-1} t_{actual_i} - \sum_{i=0}^{n-1} t_{target_i}}{t_{baud\ rate}} \times 100\%$$

or,

$$Error\ [\%] = \left\{ \frac{baud\ rate}{BRCLK} \times \left[(i+1) \times UBR + \sum_{i=0}^{n-1} m_i \right] - (i+1) \right\} \times 100\%$$

With:

baud rate: Required baud rate
BRCLK: Input frequency – selected for UCLK, ACLK, or MCLK
i = 0 for the start bit, 1 for the data bit D0, and so on
UBR: Division factor in registers UBR1 and UBR0

Example 12–3. Error Example for 2400 Baud

The following data are assumed:

Baud rate =	2400
BRCLK =	32,768 Hz (ACLK)
UBR =	13, since the ideal division factor is 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((0+1)\times \text{UBR} + 1) - 1\right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((1+1)\times \text{UBR} + 2) - 2\right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((2+1)\times \text{UBR} + 2) - 3\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((3+1)\times \text{UBR} + 3) - 4\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((4+1)\times \text{UBR} + 3) - 5\right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((5+1)\times \text{UBR} + 4) - 6\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((6+1)\times \text{UBR} + 5) - 7\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((7+1)\times \text{UBR} + 5) - 8\right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((8+1)\times \text{UBR} + 6) - 9\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((9+1)\times \text{UBR} + 7) - 10\right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((10+1)\times \text{UBR} + 7) - 11\right) \times 100\% = -1.37\%$$

$$\text{Stop bit 2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((11+1)\times \text{UBR} + 8) - 12\right) \times 100\% = 1.17\%$$

12.7.2 Typical Baud Rates and Errors

The standard baud rate data needed for the baud rate registers and the modulation register are listed in Table 12–6 for the 32,768-Hz watch crystal (ACLK) and MCLK, assumed to be 32 times the ACLK frequency. The error listed is calculated for the transmit and receive paths. In addition to the error for the receive operation, the synchronization error must be considered.

Table 12–5. Commonly Used Baud Rates, Baud Rate Data, and Errors

	Divide by		ACLK (32,768 Hz)						MCLK (1,048,576 Hz)				
Baud Rate	ACLK	MCLK	UBR1	UBR0	UMOD	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UBR1	UBR0	UMOD	Max. TX Error%	Max. RX Error %
75	436.91	13,981	1	B4	FF	–0.1/0.3	–0.1/0.3	±2	36	9D	FF	0/0.1	±2
110	297.89	9532.51	1	29	FF	0/0.5	0/0.5	±3	25	3C	FF	0/0.1	±3
150	218.45	6990.5	0	DA	55	0/0.4	0/0.4	±2	1B	4E	FF	0/0.1	±2
300	109.23	3495.25	0	6D	22	–0.3/0.7	–0.3/0.7	±2	0D	A7	00	–0.1/0	±2
600	54.61	1747.63	0	36	D5	–1/1	–1/1	±2	06	D3	FF	0/0.3	±2
1200	27.31	873.81	0	1B	03	–4/3	–4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	6/3	–6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	–9/11	–9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	–21/12	–21/12	±15	0	6D	03	–0.4/1	±2
19,200		54.61							0	36	6B	–0.2/2	±2
38,400		27.31							0	1B	03	–4/3	±2
76,800		13.65							0	0D	6B	–6/3	±4
115,200		9.10							0	09	08	–5/7	±7

The maximum error is calculated for the receive and transmit modes. The receive-mode error is the accumulated time versus the ideal scanning time in the middle of each bit. The transmit error is the accumulated timing error versus the ideal time of the bit period.

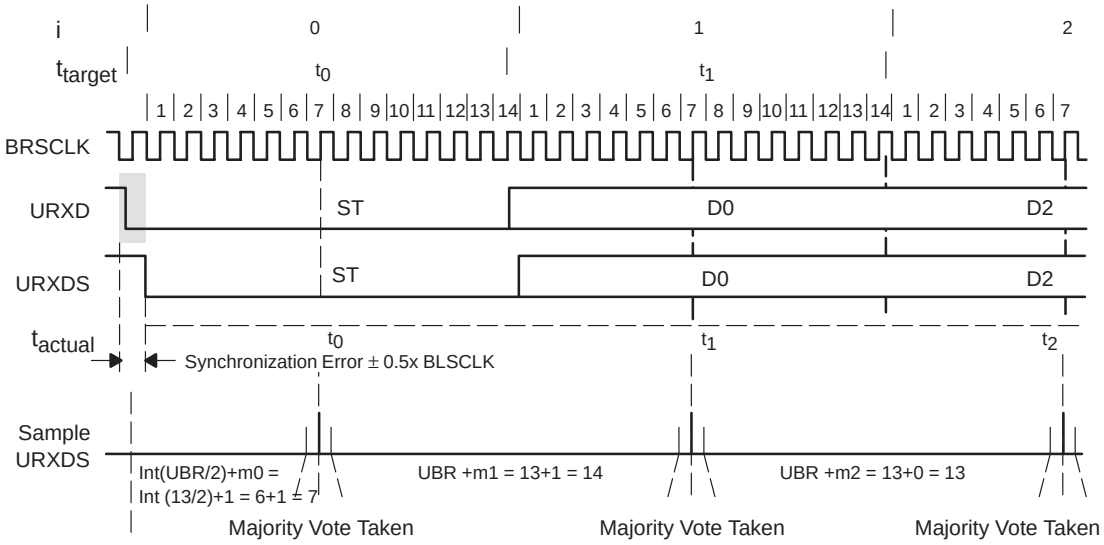
The MSP430 USART peripheral interface allows baud rates nearly as high as the clock rate. It has a low error accumulation as a result of modulating the individual bit timing. In practice, an error margin of 20% to 30% supports standard serial communication.

12.7.3 Synchronization Error

The synchronization error, shown in Figure 12–29, results from the asynchronous timing between the URXD pin data signal and the internal clock system. The receive signal is synchronized with the BRSCLK clock. The BRSCLK clock is sixteen to thirty-one times faster than the bit timing, as described.

BRSCLK = BRCLK	for		N	≤ 1F
BRSCLK = BRCLK/2	for	20h	≤ N	≤ 3Fh
BRSCLK = BRCLK/4	for	40h	≤ N	≤ 7Fh
BRSCLK = BRCLK/8	for	80h	≤ N	≤ FFh
BRSCLK = BRCLK/16	for	100	≤ N	≤ 1FF
BRSCLK = BRCLK/32	for	200	≤ N	≤ 3FFh
BRSCLK = BRCLK/64	for	400	≤ N	≤ 7FFh
BRSCLK = BRCLK/128	for	800h	≤ N	≤ FFFh
BRSCLK = BRCLK/256	for	1000h	≤ N	≤ 1FFFh
BRSCLK = BRCLK/512	for	2000h	≤ N	≤ 3FFFh
BRSCLK = BRCLK/1024	for	4000h	≤ N	≤ 7FFFh
BRSCLK = BRCLK/2048	for	8000h	≤ N	≤ FFFFh

Figure 12–29. Synchronization Error



The target start-bit detection-baud-rate timing $t_{target(0)}$ is half the baud-rate timing $t_{baud\ rate}$ because the bit is tested in the middle of its period. The target baud rate timing $t_{targeti}$ for all of the other succeeding bits is the baud rate timing $t_{baud\ rate}$.

$$Error\ [\%] = \frac{t_{actual_0} + t_{target_0}}{0.5 \times t_{target_0}} + \frac{\sum_{i=1}^{n-1} t_{actual_i} - \sum_{i=1}^{n-1} t_{target_i}}{t_{target_i}} \times 100\%$$

OR

$$Error\ [\%] = \left(\frac{baud\ rate}{BRCLK} \times \left\{ 2 \times \left[m0 + int\ (UBR/2) \right] + \left(i \times UBR + \sum_{i=1}^{n-1} m_i \right) \right\} - 1–i \right) \times 100\%$$

Where:

baud rate is the required baud rate
BRCLK is the input frequency—selected for UCLK, ACLK, or MCLK
i = 0 for the start bit, 1 for data bit D0, and so on
UBR is the division factor in registers UBR1 and BRB0

Example 12–4. Synchronization Error—2400 Baud

The following data are assumed:

Baud rate =	2400
BRCLK =	32,768 Hz (ACLK)
UBR =	13, since the ideal division factor is 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (0 \times \text{UBR} + 0 –0)]–1 \right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (1 \times \text{UBR} + 1)]–1–1 \right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (2 \times \text{UBR} + 1)]–1–2 \right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (3 \times \text{UBR} + 2)]–1–3 \right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (4 \times \text{UBR} + 2)]–1–4 \right) \times 100\% = –1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (5 \times \text{UBR} + 3)]–1–5 \right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (6 \times \text{UBR} + 4)]–1–6 \right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (7 \times \text{UBR} + 4)]–1–7 \right) \times 100\% = –1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (8 \times \text{UBR} + 5)]–1–8 \right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (9 \times \text{UBR} + 6)]–1–9 \right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (10 \times \text{UBR} + 6)]–1–10 \right) \times 100\% = –1.37\%$$

$$\text{Stop bit 2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (11 \times \text{UBR} + 7)]–1–11 \right) \times 100\% = –1.17\%$$

Chapter 13

USART Peripheral Interface, SPI Mode

The universal synchronous/asynchronous receive/transmit (USART) serial-communication peripheral supports two serial modes with one hardware configuration. These modes shift a serial-bit stream in and out of the MSP430 at a programmed rate or at a rate defined by an external clock. The first mode is the universal asynchronous-receive/transmit (UART) communication protocol (discussed in Chapter 12); the second is the serial peripheral-interface (SPI) protocol.

Bit SYNC in control register UCTL selects the required mode:

SYNC = 0:	UART—asynchronous mode selected
SYNC = 1:	SPI—synchronous mode selected

This chapter describes the SPI mode.

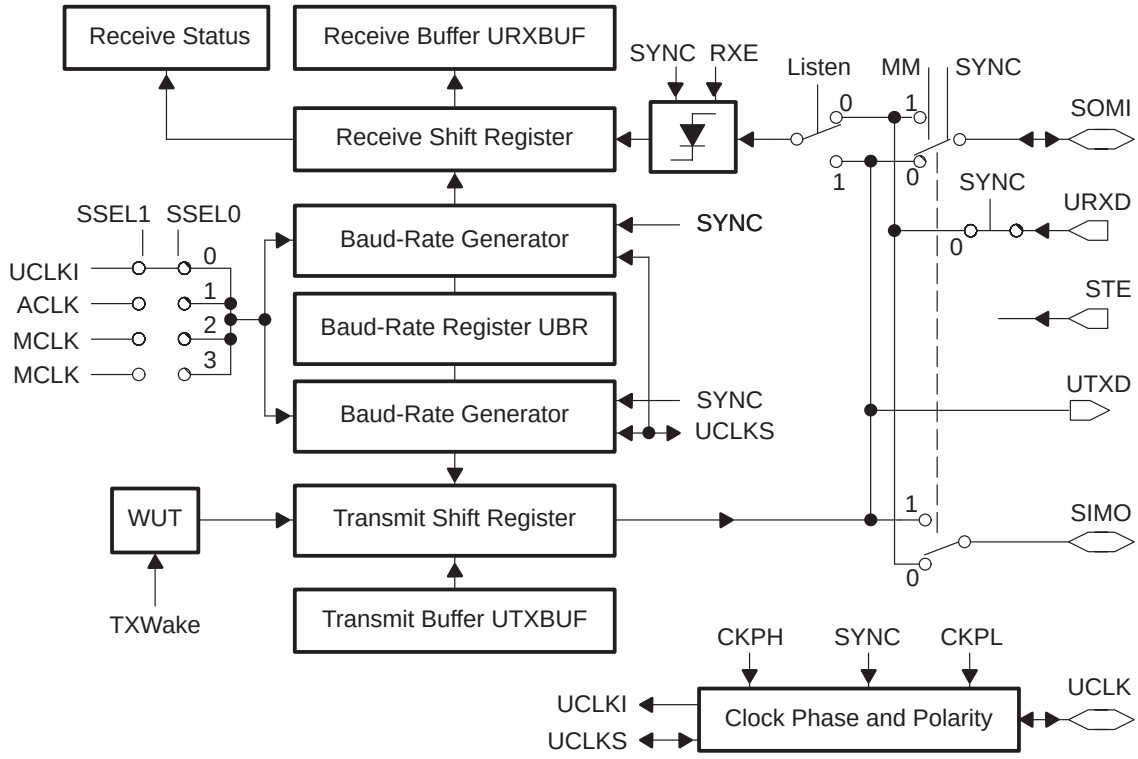
Topic **Page**

13.1 USART Peripheral Interface 13-2
13.2 USART Peripheral Interface, SPI Mode 13-3
13.3 Synchronous Operation 13-4
13.4 Interrupt and Control Functions 13-9
1.5 Control and Status Registers 13-15

13.1 USART Peripheral Interface

The USART peripheral interface connects to the CPU as a byte-peripheral module. It connects the MSP430 to the external system environment with three or four external pins. Figure 13–1 shows the USART peripheral-interface module

Figure 13–1. Block Diagram of USART



13.2 USART Peripheral Interface, SPI Mode

The USART peripheral interface is a serial channel that shifts a serial bit stream of 7 or 8 bits in and out of the MSP430. The SPI mode is chosen when control bit SYNC in the USART control register (UCTL) is set.

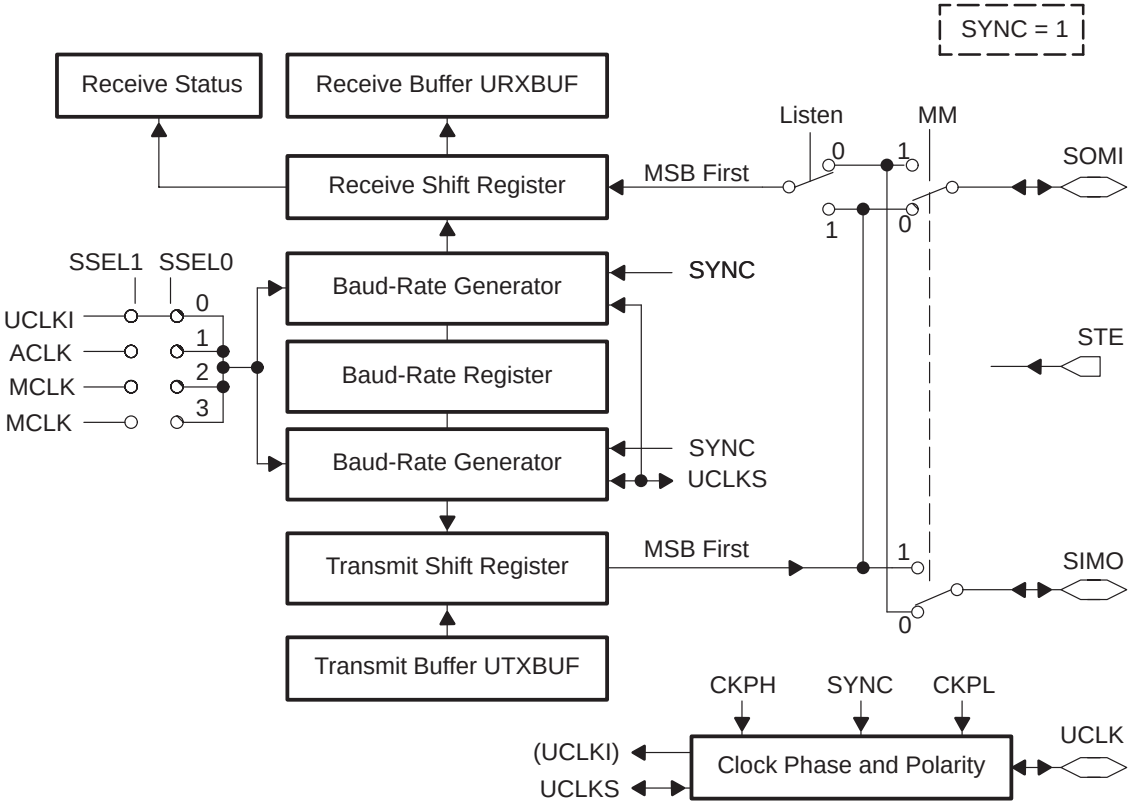
13.2.1 SPI Mode Features

The features of the SPI mode are:

- Supports three-pin and four-pin SPI operations via SOMI, SIMO, UCLK, and STE
- Master or slave mode
- Separate shift registers for receive (URXBUF) and transmit (UTXBUF)
- Double buffers for receiving and transmitting
- Has clock-polarity and clock-phase control
- Has clock-frequency control in master mode
- Supports a character length of seven or eight bits per character

Figure 13–2 shows the USART module in SPI mode.

Figure 13–2. Block Diagram of USART—SPI Mode



13.3 Synchronous Operation

In USART synchronous mode, data and clock signals transmit and receive serial data. The master supplies the clock and data. The slaves use this clock to shift serial information in and out.

The four-pin SPI mode also uses a control line to enable a slave to receive and transmit data. The line is controlled by the master.

Three or four signals are used for data exchange:

- SIMO Slave in, master out
  The direction is defined by SIMODIR (SIMODIR=0, input direction) SIMODIR = [SYNC .and. MM .and. (STC .or. STE)]
  Output direction is selected when SPI + Master Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

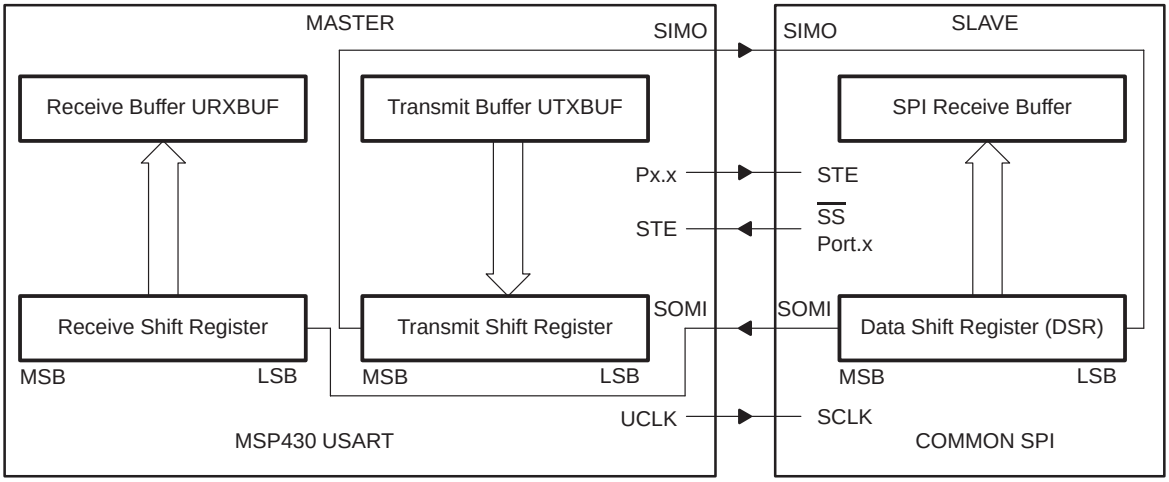
- SOMI Slave out, master in
  The direction is defined by SOMIDIR (SIMODIR=0 input direction) SOMIDIR = [SYNC .and. .not.(MM)] .or. [STC .or. .not.(STE)]
  Output direction is selected when SPI + Slave Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

- UCLK USART clock. The master drives this signal and the slave uses it to receive and transmit data.
  The direction is defined by UCLKDIR (UCLKDIR=0 input direction) UCLKDIR = [SYNC .and. MM .and. (STC .or. STE)]
  Output direction is selected when SPI + Master Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

- STE Slave transmit enable. Used in four-pin mode to control more than one slave in a multiple master and slave system.

The interconnection of the USART in synchronous mode to another device's serial port with one common transmit/receive shift register is shown in Figure 13–3, where MSP430 is master or slave. The operation of both devices is identical.

Figure 13–3. MSP430 USART as Master, External Device With SPI as Slave



The master initiates the transfer by sending the UCLK signal. For the master, data is shifted out of the transmit shift register on one clock edge, and shifted into the receive shift register on the opposite edge. For the slave, the data shifting operation is the same and uses one common register for transmitting and receiving data. Master and slave send and receive data at the same time.

Whether the data is meaningful or dummy data depends on the application software:

- Master sends data and slave sends dummy data
- Master sends data and slave sends data
- Master sends dummy data and slave sends data

Figures 13–4 and 13–5 show an example of a serial synchronous data transfer for a character length of seven bits. The initial content of the receive shift register is 00. The following events occur in order:

A) Slave writes 98h to the data shift register (DSR) and waits for the master to shift data out.

B) Master writes B0h to UTXBUF, which is immediately transferred to the transmit shift register, and starts the transmission.

C) First character is finished and sets the interrupt flags.

D) Slave reads 58h from the receive buffer (right justified).

E) Slave writes 54h to the DSR and waits for the master to shift out data.

F) Master reads 4Ch from the receive buffer URXBUF (right justified).

G) Master writes E8h to the transmit buffer UTXBUF and starts the transmission.

Note: If USART is in slave mode, no UCLK is needed after D), until G). However, in master mode, two clocks are used internally (not on UCLK signal) to end transmit/receive of first character and prepare the transmit/receive of the next character.

H) Second character is finished and sets the interrupt flag.

I) Master receives 2Ah and slave receives 74h (right justified).

Figure 13–4. Serial Synchronous Data Transfer

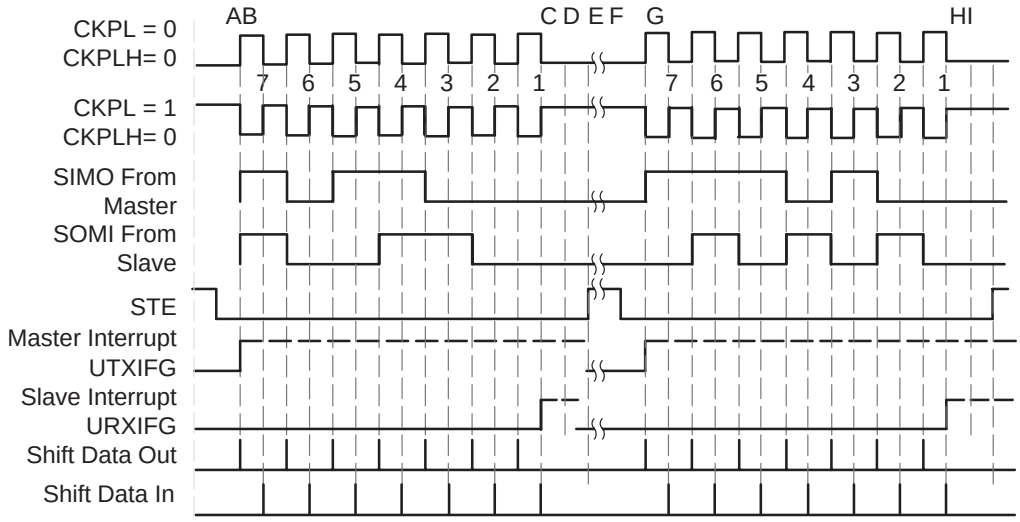
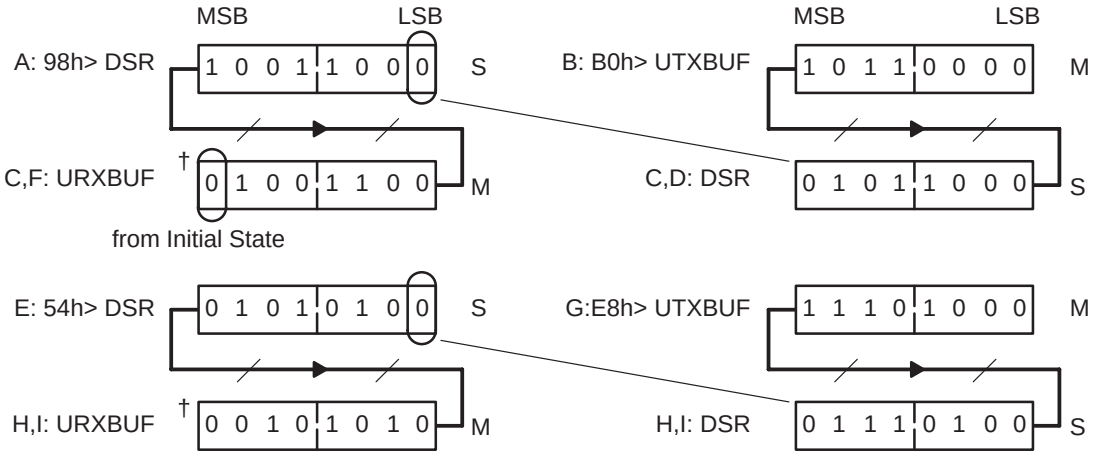


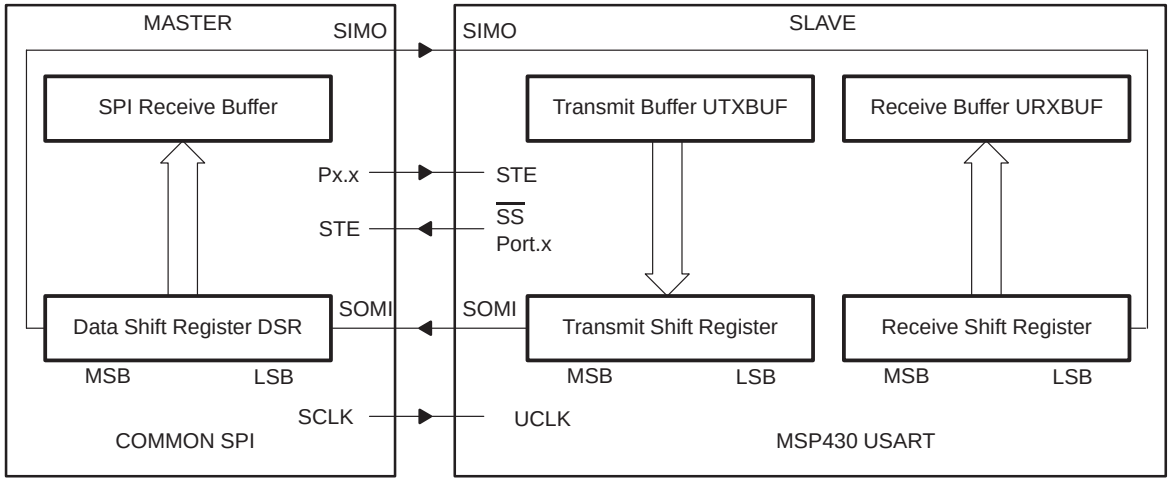
Figure 13–5. Data Transfer Cycle



† In 7 bit mode, the MSB of RXBUF is always read as 0.
S: Slave M: Master

Figure 13–6 illustrates the USART module functioning as a slave in a three or four-pin SPI configuration.

Figure 13–6. MSP430 USART as Slave in Three-Pin or Four-Pin Configuration



13.3.1 Master SPI Mode

The master mode is selected when the master-mode bit (MM) in control register UCTL is set. The USART module controls the serial-communication network by providing UCLK at the UCLK pin. Data is output on the SIMO pin during the first UCLK period and latched from the SOMI pin in the middle of the corresponding UCLK period.

The data written to the transmit buffer (UTXBUF) is moved to the transmit shift register as soon as the shift register is empty. This initiates the data transfer on the SIMO pin starting with the most-significant bit. At the same time, received data is shifted into the receive shift register and, upon receiving the selected number of bits, the data is transferred to the receive buffer (URXBUF) setting the receive interrupt flag (URXIFG). Data is shifted into the receive shift register starting with the most-significant bit. It is stored and right-justified in the receive buffer (URXBUF). When previous data is not read from the receive buffer (URXBUF), the overrun error bit (OE) is set.

Note: USART Synchronous Master Mode, Receive Initiation

The master writes data to the transmit buffer UTXBUF to receive a character. The receive starts when the transmit shift register is empty and the data is transferred to it. Receive and transmit operations always take place together, at opposite clock edges.

The protocol can be controlled using the transmit-interrupt flag UTXIFG, or the receive-interrupt flag URXIFG. By using UTXIFG immediately after sending the shift-register data to the slave, the buffer data is transferred to the shift register and the transmission starts. The slave receive timing should ensure that there is a timely pick-up of the data. The URXIFG flag indicates when the data shifts out and in completely. The master can use URXIFG to ensure that the slave is ready to correctly receive the next data.

13.3.1.1 Four-Pin SPI Master Mode

The signal on STE is used by the active master to prevent bus conflicts with another master. The STE pin is an input when the corresponding PnSEL bit (in the I/O registers) selects the module function. The master operates normally while the STE signal is high. Whenever the STE signal is low, for example, when another device makes a request to become master, the actual master reacts such that:

- The pins that drive the SPI bus lines SIMO and UCLK are set to inputs.
- The error bit FE and the interrupt flag URXIFG in register URCTL are set.

The bus conflict is then removed: SIMO and UCLK do not drive the bus lines, and the error flag indicates the system integrity violation to the software. Pins SIMO and UCLK are forced to the input state while STE is in a low state, and they return to the conditions defined by the corresponding control bits when STE returns to a high state.

In the three-pin mode, the STE input signal is not relevant.

13.3.2 Slave SPI Mode

The slave mode is selected when bit MM of the control register is reset and synchronous mode is selected.

The UCLK pin is used as the input for the serial-shift clock supplied by an external master. The data-transfer rate is determined by this clock and not by the internal bit-rate generator. The data, loaded into the transmit shift register through the transmit buffer (UTXBUF) before the start of UCLK, is transmitted on the SOMI pin using the UCLK supplied from the master. Simultaneously, the serial data applied to the SIMO pin are shifted into the receive shift register on the opposite edge of the clock.

The receive-interrupt flag URXIFG indicates when the data is received and transferred into the receive buffer. The overrun-error bit is set when the previously-received data is not read before the new data is written to the receive buffer.

13.3.2.1 Four-Pin SPI Slave Mode

In the four-pin SPI mode, the STE signal is used by the slave to enable the transmit and receive operations. It is applied from the SPI master. The receive and transmit operations are disabled when the STE signal is high, and enabled when it is low. Whenever the STE signal becomes high, any receive operation in progress is halted and then continues when the STE signal is low again. The STE signal enables one slave to access the data lines. The SOMI is input if STE is set high.

13.4 Interrupt and Control Functions

The USART peripheral interface serves two main interrupt sources for transmission and reception. Two interrupt vectors serve receive and transmit interrupt events.

The interrupt control bits and flags and enable bits of the USART peripheral interface are located in the SFR address range. The bit functions are described below in Table 13–1. See the peripheral-file map in Appendix A for the exact bit locations.

Table 13–1. USART Interrupt Control and Enable Bits—SPI Mode

Receive interrupt flag	URXIFG	Initial state reset (by PUC/SWRST)
Receive interrupt enable	URXIE	Initial state reset (by PUC/SWRST)
Receive/transmit enable (see Note)	USPIIE	Initial state reset (by PUC)
Transmit interrupt flag	UTXIFG	Initial state set (by PUC/SWRST)
Transmit interrupt enable	UTXIE	Initial state reset (by PUC/SWRST)

Note: Different for UART mode, see Chapter 12.

The USART receiver and transmitter operate in parallel and use the same baud-rate generator in synchronous master mode. In synchronous slave mode, the external clock applied to UCLK is used for the receiver and the transmitter. The receiver and transmitter are enabled and disabled together with the USPIIE bit.

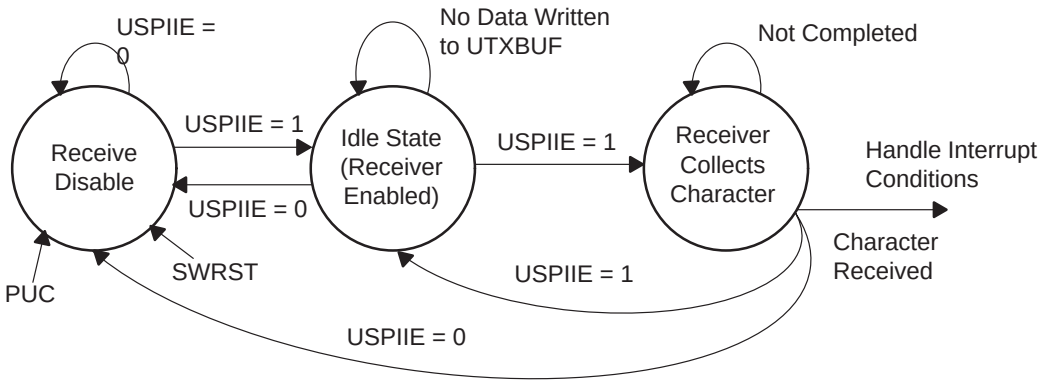
13.4.1 USART Receive/Transmit Enable Bit, Receive Operation

The receive/transmit enable bit (USPIIE) enables or disables collection of the bit stream on the URXD/SOMI data line. Disabling the USART receiver (USPIIE = 0) stops the receive operation after completion, or stops a pending operation if no receive operation is active. In synchronous mode, UCLK does not shift any data into the receiver shift register.

13.4.1.1 Receive/Transmit Enable Bit—MSP430 as Master

The receive operation functions identically for three-pin and four-pin modes, as shown in Figure 13–7, when the MSP430 USART is selected to be the SPI master.

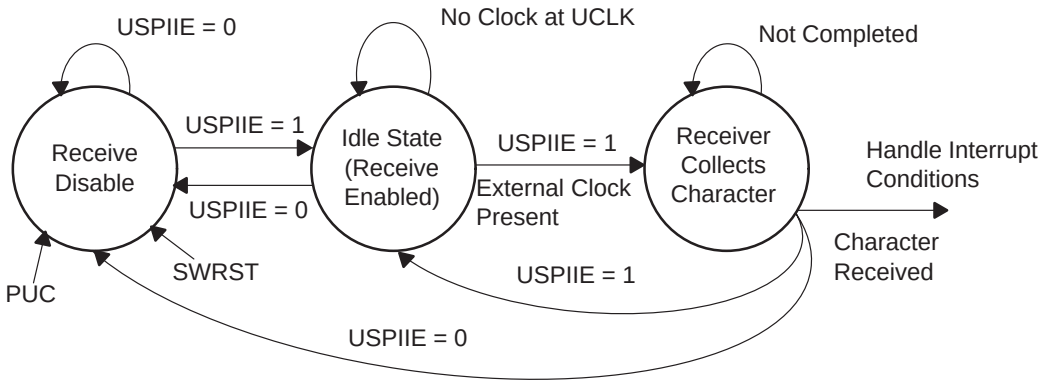
Figure 13–7. State Diagram of Receiver Enable Operation—MSP430 as Master



13.4.1.2 Receive/Transmit Enable Bit—MSP430 as Slave, Three-Pin Mode

The receive operation functions differently for three-pin and four-pin modes when the MSP430 USART module is selected to be the SPI slave. In the three-pin mode, shown in Figure 13–8, no external SPI receive-control signal stops an active receive operation. A PUC signal, a software reset (SWRST), or a receive/transmit enable (USPIIE) signal can stop a receive operation and reset the USART.

Figure 13–8. State Diagram of Receive/Transmit Enable—MSP430 as Slave, Three-Pin Mode



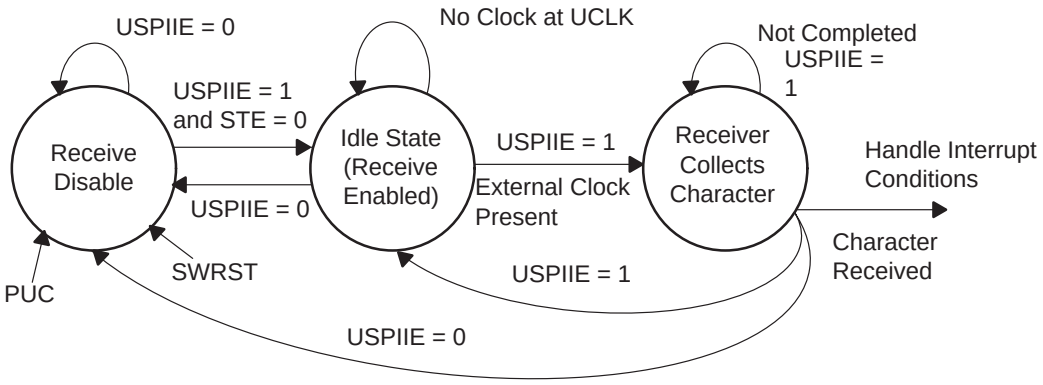
Note: USPIIE Re-Enabled, SPI Mode

After the receiver is completely disabled, a reenabling of the receiver is asynchronous to any data stream on the communication line. Synchronization to the data stream is handled by the software protocol in three-pin SPI mode.

13.4.1.3 Receive/Transmit Enable Bit—MSP430 as Slave, Four-Pin Mode

In the four-pin mode, shown in Figure 13–9, the external SPI receive-control signal applied to pin STE stops a started receive operation. A PUC signal, a software reset (SWRST), or a receive/transmit enable (USPIIE) can stop a receive operation and reset the operation-control state machine. Whenever the STE signal is set to high, the receive operation is halted.

Figure 13–9. State Diagram of Receive Enable—MSP430 as Slave, Four-Pin Mode



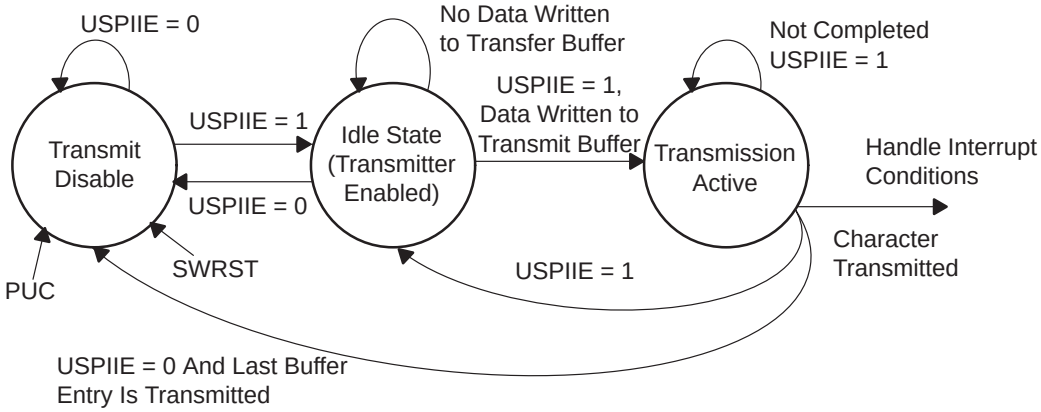
13.4.2 USART Receive/Transmit Enable Bit, Transmit Operation

The receive/transmit enable bit USPIIE, shown in Figures 13–10 and 13–11, enables or disables the shifting of a character on the serial data line. If this bit is reset, the transmitter is disabled, but any active transmission does not halt until all data previously written to the transmit buffer is transmitted. If the transmission is completed, any further write operation to the transmitter buffer does not transmit. When the UTXBUF is ready, any pending request for transmission remains, which results in an immediate start of transmission when USPIIE is set and the transmitter is empty. A low state on the STE signal removes the active master (four-pin mode) from the bus. It also indicates that another master is requesting the active-master function.

13.4.2.1 Receive/Transmit Enable—MSP430 as Master

Figure 13–10 shows the transmit-enable activity when the MSP430 is master.

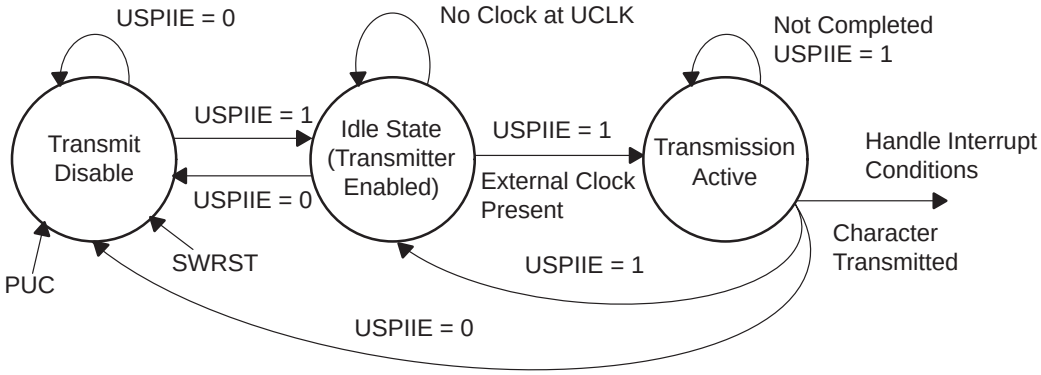
Figure 13–10. State Diagram of Transmit Enable—MSP430 as Master



13.4.2.2 Receive/Transmit Enable, MSP430 is Slave

Figure 13–11 shows the receive/transmit-enable-bit activity when the MSP430 is slave.

Figure 13–11. State Diagram of Transmit Enable—MSP430 as Slave



When USPIIE is reset, any data can be written regularly into the transmit buffer, but no transmission is started. Once the USPIIE bit is set, the data in the transmit buffer are immediately loaded into the transmit shift register and character transmission is started.

Note: Writing to UTXBUF, SPI Mode

Data should never be written to transmit buffer UTXBUF when the buffer is not ready (UTXIFG = 0) and the transmitter is enabled (USPIIE is set). If data is written, character shifting can be random.

Note: Write to UTXBUF/Reset of Transmitter, SPI Mode

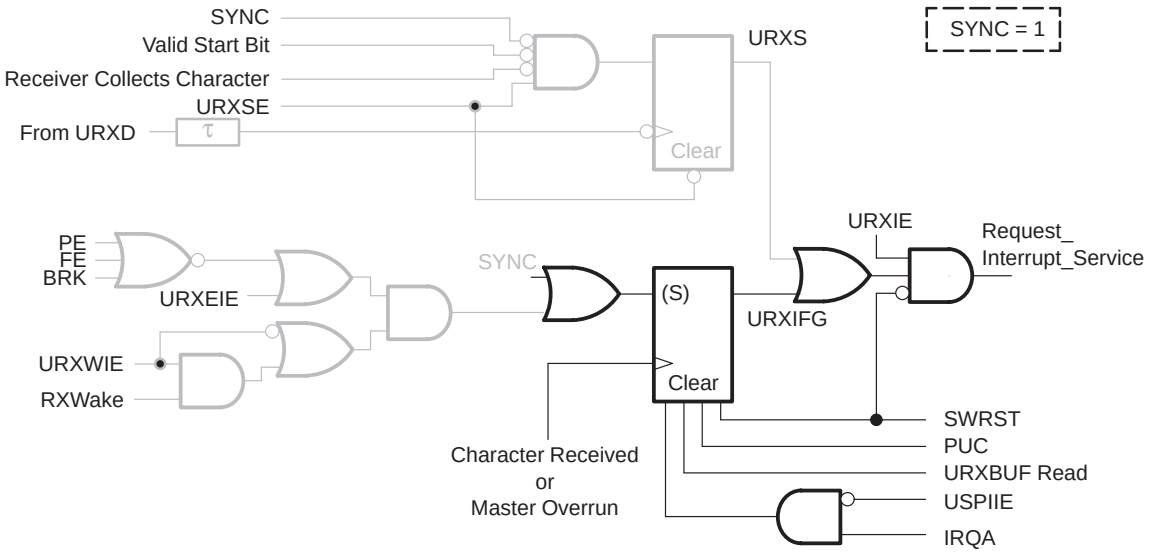
Disabling of the transmitter should be done only if all data to be transmitted have been moved to the transmit shift register.

```
MOV.B  #....,&UTXBUF
BIC.B  #USPIIE,&ME2      ; If BITCLK < MCLK then the
                         ; transmitter might be stopped
                         ; before the buffer is loaded
                         ; into the transmitter
                         ; shift register
```

13.4.3 USART Receive-Interrupt Operation

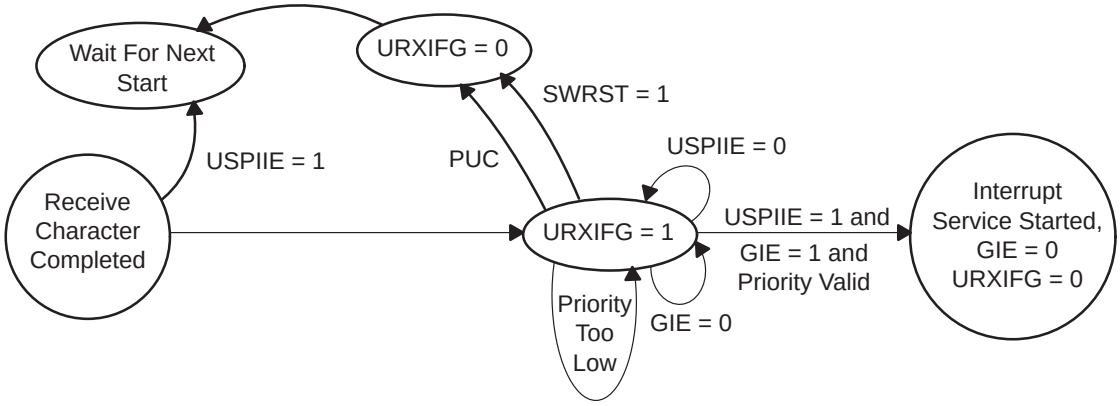
In the receive-interrupt operation shown in Figure 13–12, the receive-interrupt flag URXIFG is set each time a character is received and loaded into the receive buffer.

Figure 13–12. Receive Interrupt Operation



URXIFG is reset by a system reset PUC signal, or by a software reset (SWRST). URXIFG is reset automatically if the interrupt is served or the receive buffer URXBUF is read. The receive interrupt enable bit (USPIIE), if set, enables a CPU interrupt request as shown in Figure 13–13. The receive interrupt flag bits URXIFG and USPIIE are reset with a PUC signal or a SWRST.

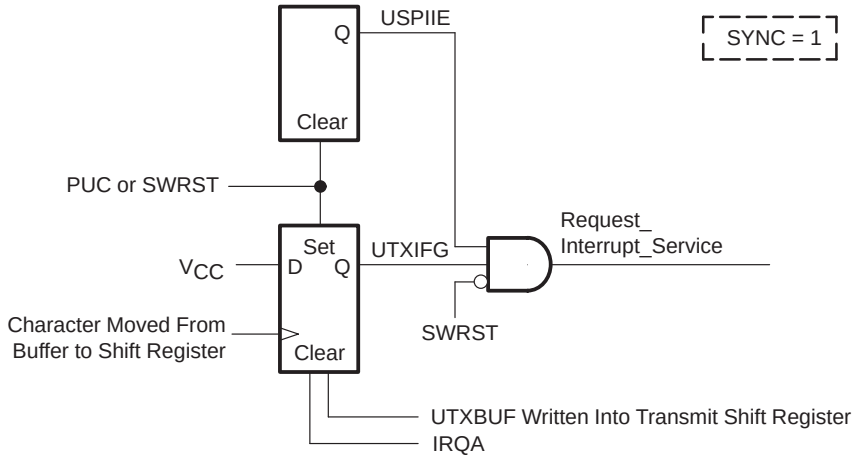
Figure 13–13. Receive Interrupt State Diagram



13.4.4 Transmit-Interrupt Operation

In the transmit-interrupt operation shown in Figure 13–14, the transmit-interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt-request service is started or a character is written to the UTXBUF. This flag activates a transmitter interrupt if bits USPIIE and GIE are set. The UTXIFG is set after a system reset PUC signal, or removal of SWRST.

Figure 13–14. Transmit-Interrupt Operation



The transmit-interrupt enable bit UTXIE controls the ability of the UTXIFG to request an interrupt, but does not prevent the UTXIFG flag from being set. The USPIIE is reset with a PUC signal or a SWRST. The UTXIFG bit is set after a system reset PUC signal or a SWRST, but the USPIIE bit is reset to ensure full interrupt-control capability.

13.5 Control and Status Registers

The USART registers, shown in Table 13–2, are byte structured and should be accessed using byte instructions.

Table 13–2. USART Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL	Read/write	070h	See Section 13.5.1
Transmit control	UTCTL	Read/write	071h	See Section 13.5.2
Receive control	URCTL	Read/write	072h	See Section 13.5.3
Modulation control	UMCTL	Read/write	073h	Unchanged
Baud Rate 0	UBR0	Read/write	074h	Unchanged
Baud Rate 1	UBR1	Read/write	075h	Unchanged
Receive buffer	URXBUF	Read/write	076h	Unchanged
Transmit buffer	UTXBUF	Read	077h	Unchanged

All bits are random following the PUC signal, unless otherwise noted by the detailed functional description.

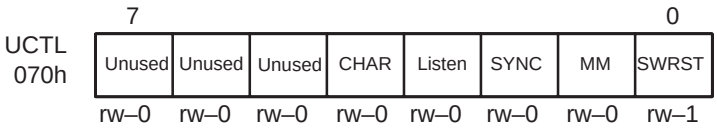
Reset of the USART module is performed by the PUC signal or a SWRST. After a PUC signal, the SWRST bit remains set and the USART module remains in the reset condition. It is disabled by resetting the SWRST bit. The SPI mode is disabled after the PUC signal.

The USART module operates in asynchronous or synchronous mode as defined by the SYNC bit. The bits in the control registers can have different functions in the two modes. All bits are described with their function in the synchronous mode—SYNC = 1. Their function in the asynchronous mode is described in Chapter 12.

13.5.1 USART Control Register

The information stored in the control register, shown in Figure 13–15, determines the basic operation of the USART module. The register bits select the communication mode and the number of bits per character. All bits should be programmed to the desired mode before resetting the SWRST bit.

Figure 13–15. USART Control Register



Bit 0: The USART state machines and operating flags are initialized to the reset condition (URXIFG=USPIIE=0, UTXIFG=1) if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be reenabled by resetting this bit.

Bit 1: Master mode is selected when the MM bit is set. The USART module slave mode is selected when the MM bit is reset.

Bit 2: Peripheral module mode select
The SYNC bit sets the function of the USART peripheral-interface module. Some of the USART control bits have different functions in UART and SPI modes.
SYNC = 0: UART function is selected
SYNC = 1: SPI function is selected

Bit 3: The listen bit determines the transmitted data to feed back internally to the receiver. This is commonly called loopback mode.

Bit 4: Character length
This register bit sets the length of the character to be transmitted as either seven or eight bits.
CHAR = 0: 7-bit data
CHAR = 1: 8-bit data

Bit 5: Unused

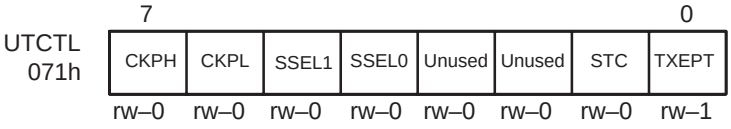
Bit 6: Unused

Bit 7: Unused

13.5.2 Transmit Control Register UTCTL

The transmit control register (UTCTL), shown in Figure 13–16, controls the USART hardware associated with transmitter operations.

Figure 13–16. Transmit Control Register UTCTL



Bit 0: Master mode:
The transmitter-empty flag TXEPT is set when the transmitter shift register and UTXBUF are empty, and reset when data are written to UTXBUF. It is set again by a SWRST.

Slave mode:
The transmitter-empty flag TXEPT is not set when the transmitter shift register and UTXBUF are empty.

Bit 1: The slave transmit-control bit STC selects if the STE pin is used for master and slave mode:

STC = 0: The four-pin mode of SPI is selected. The STE signal is used by the master to avoid bus conflicts, or is used in slave mode to control transmit and receive enable.

STC = 1: The three-pin SPI mode is selected. STE is not used in master or slave mode.

Bit 2: Unused

Bit 3: Unused

Bits 4, 5: Source select 0 and 1

The source-select bits define which clock source is used for baud-rate generation only when master mode is selected:

SSEL1,SSEL0	0	External clock UCLK selected
	1	Auxiliary clock ACLK selected
	2, 3	MCLK

In master mode (MM = 1), an external clock at UCLK cannot be selected since the master supplies the UCLK signal for any slave. In slave mode, bits SSEL1 and SSEL0 are not relevant. The external clock UCLK is always used.

Bits 6, 7: Clock polarity CKPL and clock phase CKPH

The CKPL bit controls the polarity of the SPICLK signal.

CKPL = 0: The inactive level is low; data is output with the rising edge of UCLK; input data is latched with the falling edge of UCLK.

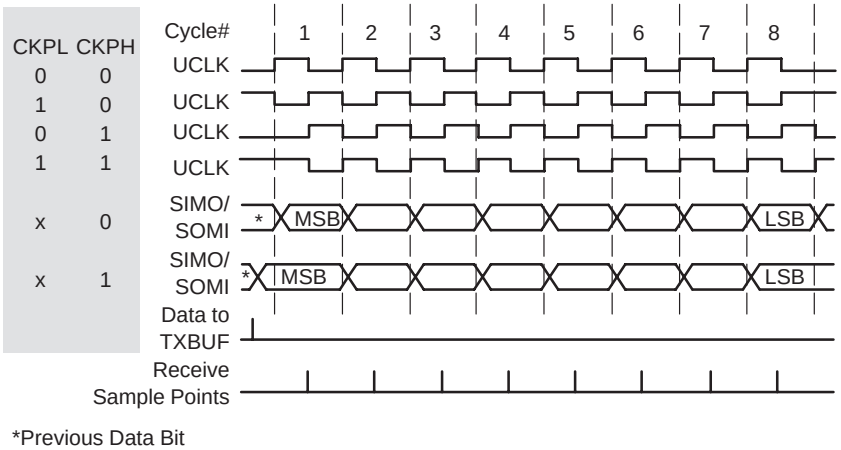
CKPL = 1: The inactive level is high; data is output with the falling edge of UCLK; input data is latched with the rising edge of SPICLK.

The CKPH bit controls the polarity of the SPICLK signal as shown in Figure 13–17.

CKPH = 0: Normal UCLK clocking scheme

CKPH = 1: UCLK is delayed by one half cycle

Figure 13–17. USART Clock Phase and Polarity



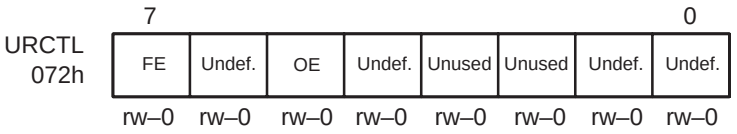
*Previous Data Bit

When operating with the CKPH bit set, the USART (synchronous mode) makes the first bit of data available after the transmit shift register is loaded and before the first edge of the UCLK. In this mode, data is latched on the first edge of UCLK and transmitted on the second edge.

13.5.3 Receive Control Register URCTL

The receive control register (URCTL), shown in Figure 13–18, controls the USART hardware associated with the receiver operation and holds error conditions.

Figure 13–18. Receive Control Register URCTL



Bit 0: Undefined, driven by USART hardware

Bit 1: Undefined, driven by USART hardware

Bit 2: Unused

Bit 3: Unused

Bit 4: Undefined, driven by USART hardware

Bit 5: The overrun-error-flag bit (OE) is set when a character is transferred to URXBUF before the previous character is read. The previous character is overwritten and lost. OE is reset by a SWRST, a system reset, by reading the URXBUF, or by an instruction.

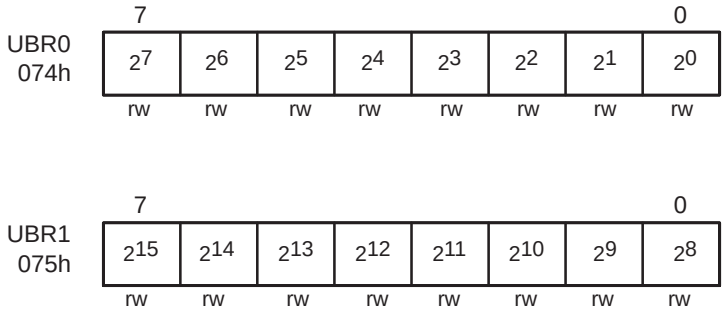
Bit 6: Undefined, driven by USART hardware

Bit 7: Frame error. The FE bit is set when four-pin mode is selected and a bus conflict stops an active master by applying a negative transition signal to pin STE. FE is reset by a SWRST, a system reset, by reading the URXBUF, or by an instruction.

13.5.4 Baud Rate Select and Modulation Control Registers

The baud-rate generator uses the content of baud-rate select registers UBR1 and UBR0, shown in Figure 13–19, to generate the serial-data-stream bit timing. The smallest division factor is two.

Figure 13–19. USART Baud-Rate Select Register

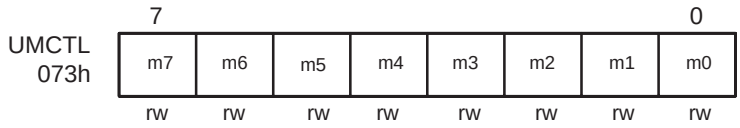


$$\text{Baud rate} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i}^{n} mi} \text{ with UBR= [UBR1,UBR0]}$$

The maximum baud rate that can be selected for transmission in master mode is half of the clock-input frequency of the baud-rate generator. In slave mode, the rate is determined by the external clock applied to UCLK.

The modulation control register, shown in Figure 13–20, is not used for serial synchronous communication. It is best kept in reset mode (bits m0 to m7 = 0).

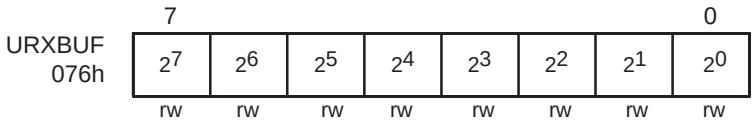
Figure 13–20. USART Modulation Control Register



13.5.5 Receive Data Buffer URXBUF

The receive data buffer (URXBUF), shown in Figure 13–21, contains previous data from the receiver shift register. URXBUF is cleared with a SWRST or a PUC signal. Reading URXBUF resets the receive-error bits and the receive-interrupt flag URXIFG.

Figure 13–21. Receive Data Buffer URXBUF

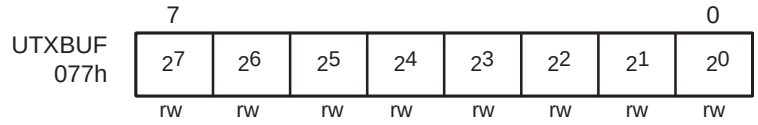


The MSB of the URXBUF is always reset in seven-bit-length mode.

13.5.6 Transmit Data Buffer UTXBUF

The transmit data buffer (UTXBUF), shown in Figure 13–22, contains current data for the transmitter to transmit.

Figure 13–22. Transmit Data Buffer UTXBUF



The UTXIFG bit indicates that UTXBUF is ready to accept another character for transmission. In master mode, the transmission is initialized by writing data to UTXBUF. The transmission of this data is started immediately if the transmit shift register is empty.

When seven-bit character-length is used, the data moved into the transmit buffer must be left-justified since the MSB is shifted out first.

Chapter 14

Liquid Crystal Display Drive

This chapter describes the MSP430x3xx liquid crystal display (LCD) driver.

Topic **Page**

14.1 LCD Drive Basics 14-2
14.2 LCD Controller/Driver 14-7
14.3 Code Examples 14-21

14.1 LCD Drive Basics

LCDs must be driven with ac voltages. DC voltage signals applied to LCD segments can harm and even destroy an LCD. The LCD controller/driver on the MSP430 devices simplifies the use of LCD displays by creating the ac voltage signals automatically.

Static LCDs have one pin for each segment and one pin for the ground plane, so one can see how the pin-counts of large LCDs with many segments could easily become cumbersome. For example, an 80-segment, static LCD requires 81 pins. To reduce pin-counts, LCDs are often *multiplexed*. This means the individual LCD segments are arranged in a matrix of the segment pins and common pins, such that each LCD segment has a unique combination of a segment pin and a common pin for activation, but each segment pin can be used for more than one segment. For example, a 2-MUX LCD contains one segment pin for every two segments and 2 common layers, each with a pin. The two segments that share any pin are connected to different common layers for individual control. The table below shows a possible pin configuration for a 2-MUX, 16-segment LCD. There are 10 total pins.

Pin 1	segment 1	segment 2
Pin 2	segment 3	segment 4
Pin 3	segment 5	segment 6
Pin 4	segment 7	segment 8
Pin 5	segment 9	segment 10
Pin 6	segment 11	segment 12
Pin 7	segment 13	segment 14
Pin 8	segment 15	segment 16
	common pin 0	common pin 1

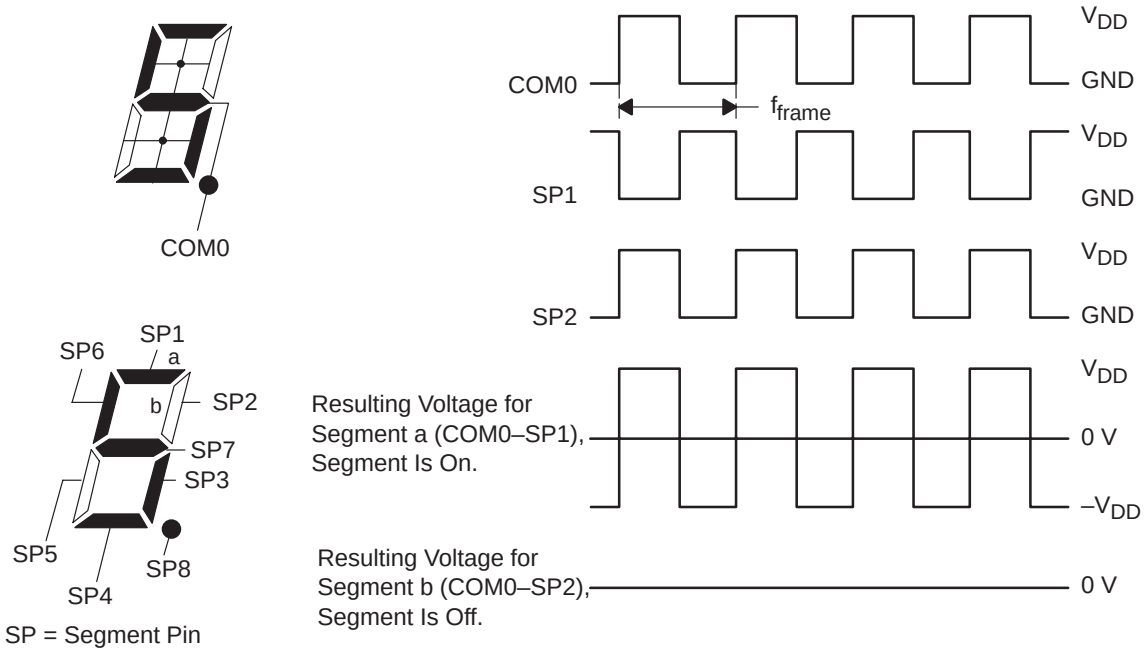
LCDs with more common planes realize greater pin-count reductions. For example, a possible pin configuration of a 4-MUX, 16-segment LCD is shown below. This LCD has 8 total pins for a reduction of 2 pins over the 2-MUX configuration above. 2 pins is generally not significant, however, in the case of a 132-segment LCD for example, the required pins for a 2-MUX version would be 68 (132/2 +2), whereas the required pins for a 4-MUX version would be 37 (132/4 +4).

Pin 1	segment 1	segment 2	segment 3	segment 4
Pin 2	segment 5	segment 6	segment 7	segment 8
Pin 3	segment 9	segment 10	segment 11	segment 12
Pin 4	segment 13	segment 14	segment 15	segment 16
	common pin 0	common pin 1	common pin 2	common pin 3

Because of the multiplexing of segments with segment pins, the required drive signals for the segment pins and common pins can be complicated. Each segment and common pin of a multiplexed LCD requires a time-division-multiplexed signal in order to only turn on the desired segments and to avoid having a dc voltage on any segment. Some examples of segment and common signals are shown below. Fortunately for the user, the MSP430 creates all these signals automatically.

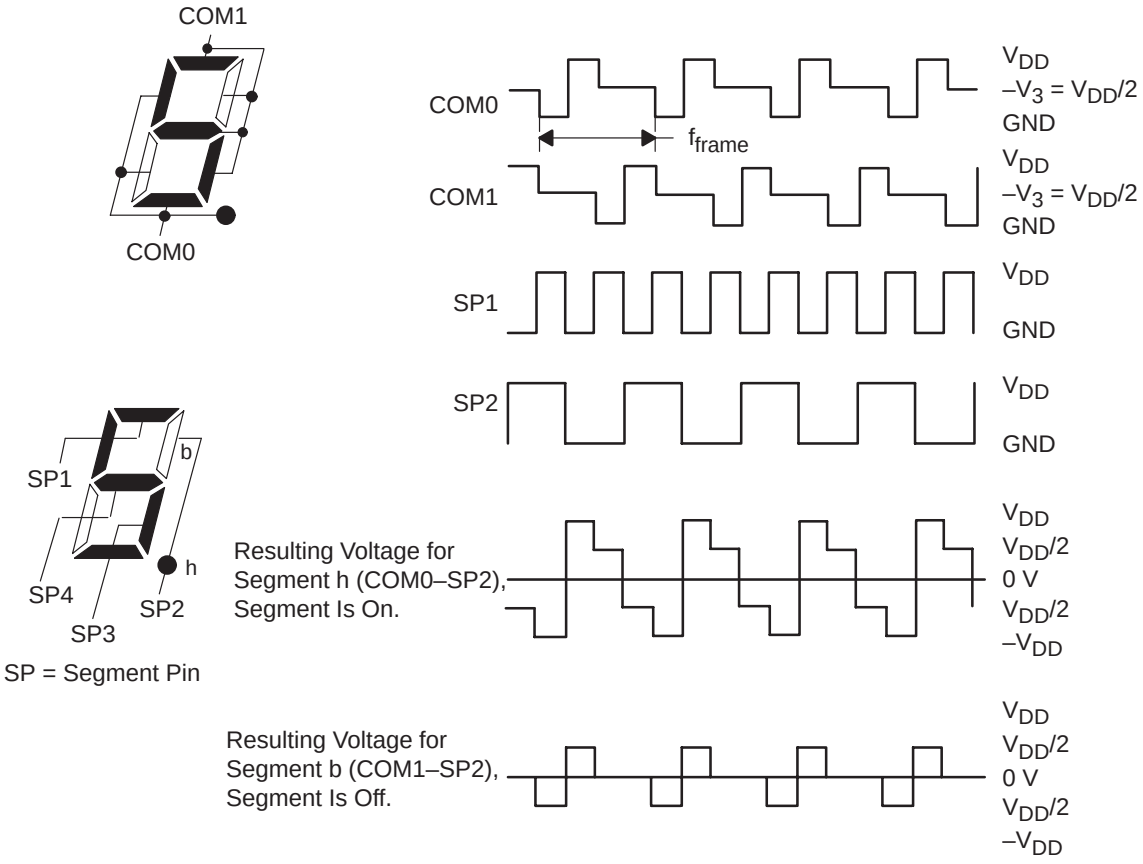
With static LCDs, each segment pin drives one segment. Figure 14–1 shows some example waveforms with a typical pin assignment.

Figure 14–1. Static Wave-Form Drive



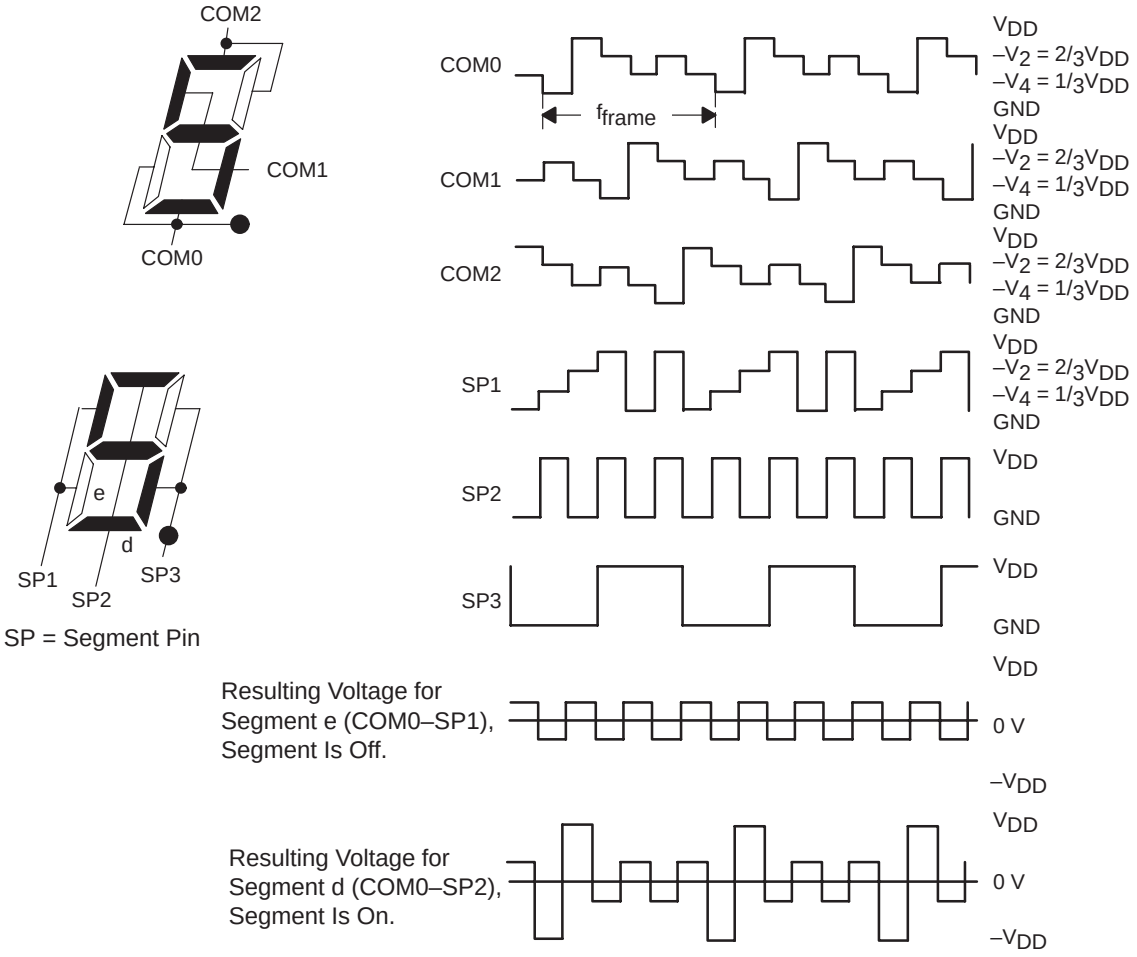
With 2-MUX LCDs, each segment pin drives two segments (see Figure 14–2).

Figure 14–2. Two-*MUX Wave-Form Drive*



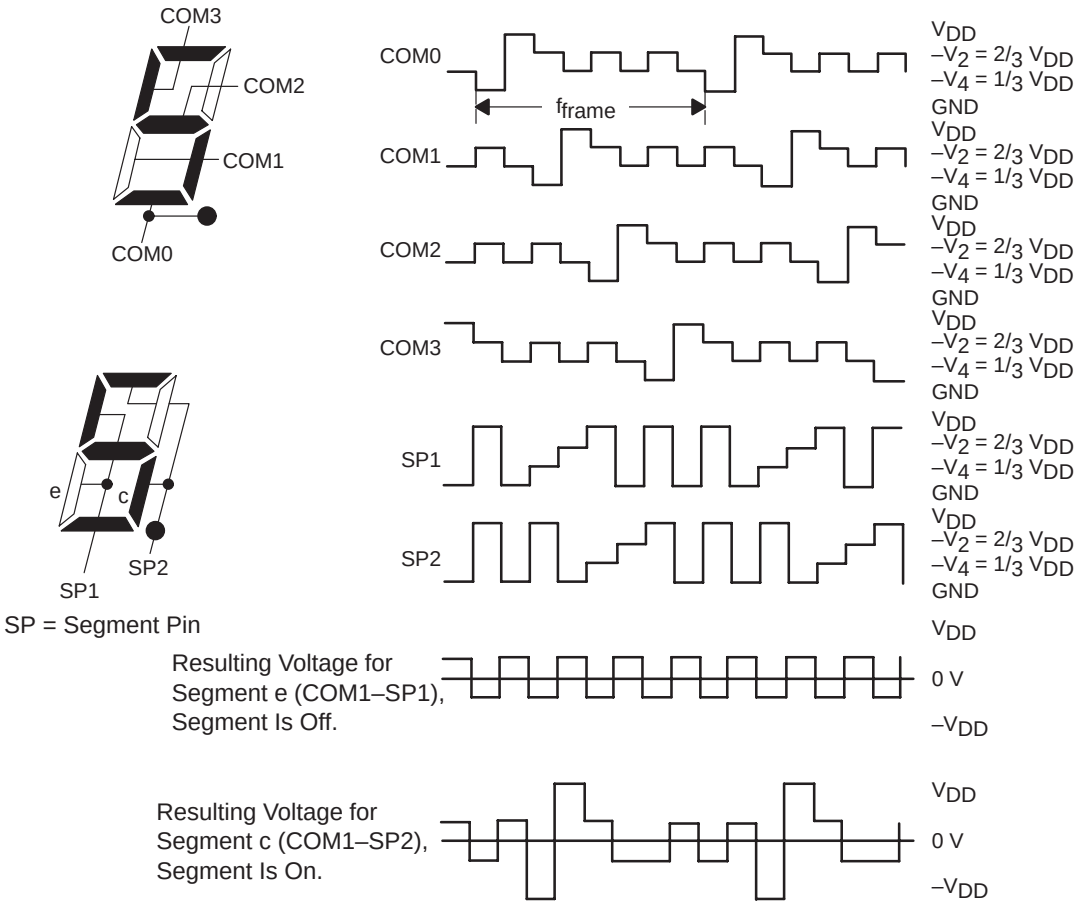
With three-MUX LCDs, each segment line drives three segments.

Figure 14–3. Three-*MUX Wave-Form Drive*



With 4-MUX LCDs, each segment pin drives four segments.

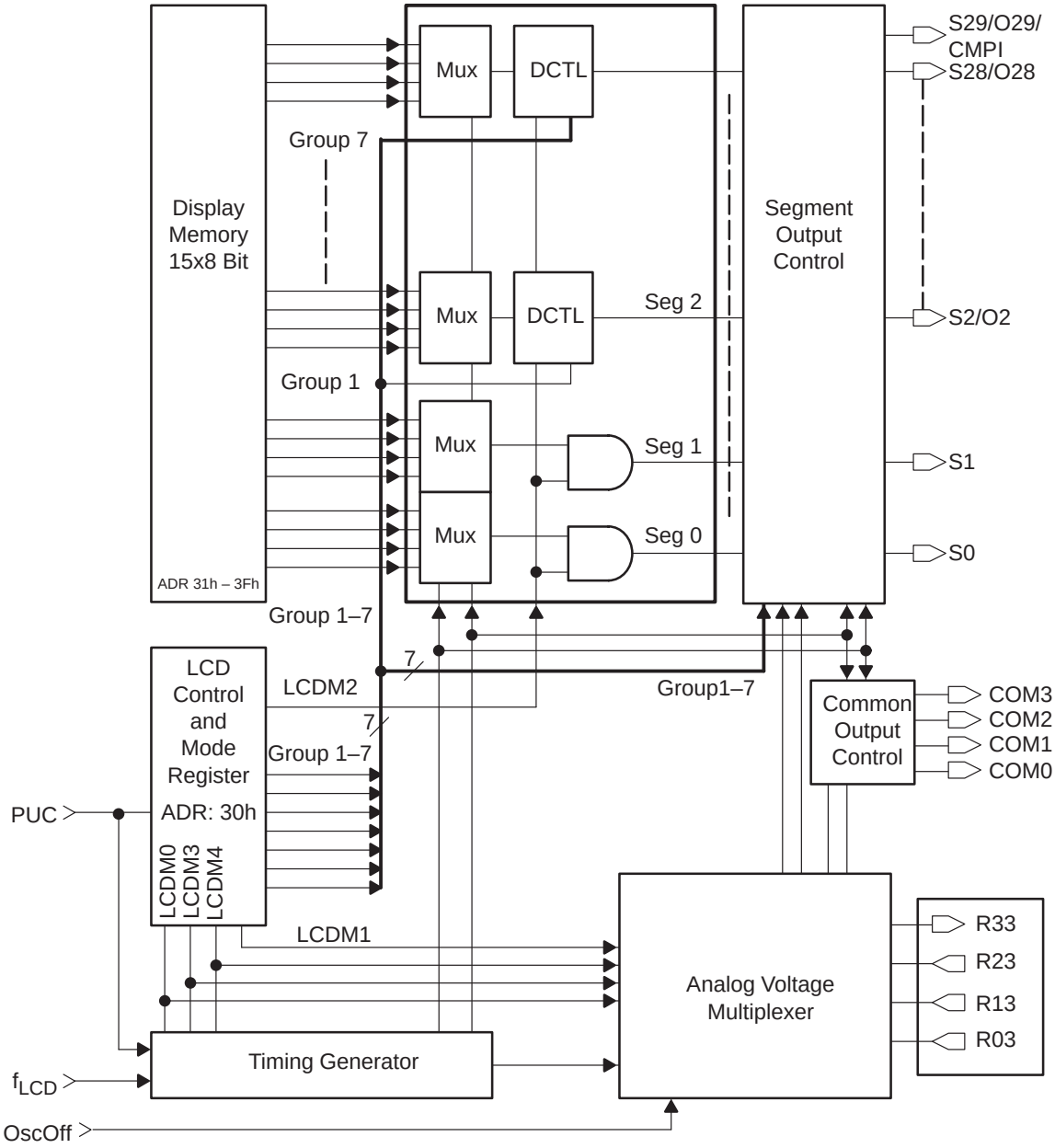
Figure 14–4. Four-*MUX Wave-Form Drive*



14.2 LCD Controller/Driver

The LCD controller/driver peripheral, shown in Figure 14–5, contains all the functional blocks and generates the segment and common signals required to drive an LCD.

Figure 14–5. LCD Controller/Driver Block Diagram



14.2.1 LCD Controller/Driver Features

The LCD controller/driver features are:

- ❏ Display memory
- ❏ Automatic signal generation
- ❏ Support for 4 types of LCDs:
  - ■ Static
  - ■ 2 MUX, 1/2 bias
  - ■ 3 MUX, 1/3 bias
  - ■ 4 MUX, 1/3 bias
- ❏ Multiple frame frequencies
- ❏ Unused segment outputs may be used as general-purpose outputs.
- ❏ Unused display memory may be used as normal memory
- ❏ Operates using the basic timer with the auxiliary clock (ACLK).

The LCD-line frame frequencies include:

- ❏ Static mode: $f_{frame} = \frac{1}{2} \times f_{LCD}$
- ❏ 2 MUX: $f_{frame} = \frac{1}{4} \times f_{LCD}$
- ❏ 3 MUX: $f_{frame} = \frac{1}{6} \times f_{LCD}$
- ❏ 4 MUX: $f_{frame} = \frac{1}{8} \times f_{LCD}$

14.2.2 LCD Timing Generation

The LCD controller uses the f_{LCD} signal from the Basic Timer1 (discussed in Chapter 10) to generate the timing for common and segment lines. The frequency f_{LCD} of signal is generated from ACLK. Using a 32,768-Hz crystal, the f_{LCD} frequency can be 1024 Hz, 512 Hz, 256 Hz, or 128 Hz. Bits FRFQ1 and FRFQ0 allow the correct selection of frame frequency. The proper frequency f_{LCD} depends on the LCD's requirement for framing frequency and LCD multiplex rate, and is calculated by:

$$f_{LCD} = 2 \times \text{MUX rate} \times f_{Framing}$$

A 3 MUX example follows:

LCD data sheet: $f_{Framing}$ = 100 Hz .... 30 Hz

FRFQ: $f_{LCD} = 6 \times f_{Framing}$

$f_{LCD} = 6 \times 100$ Hz = 600 Hz ... 6×30 Hz = 180 Hz

Select f_{LCD}: 1024 Hz, 512 Hz, 256 Hz, or 128 Hz

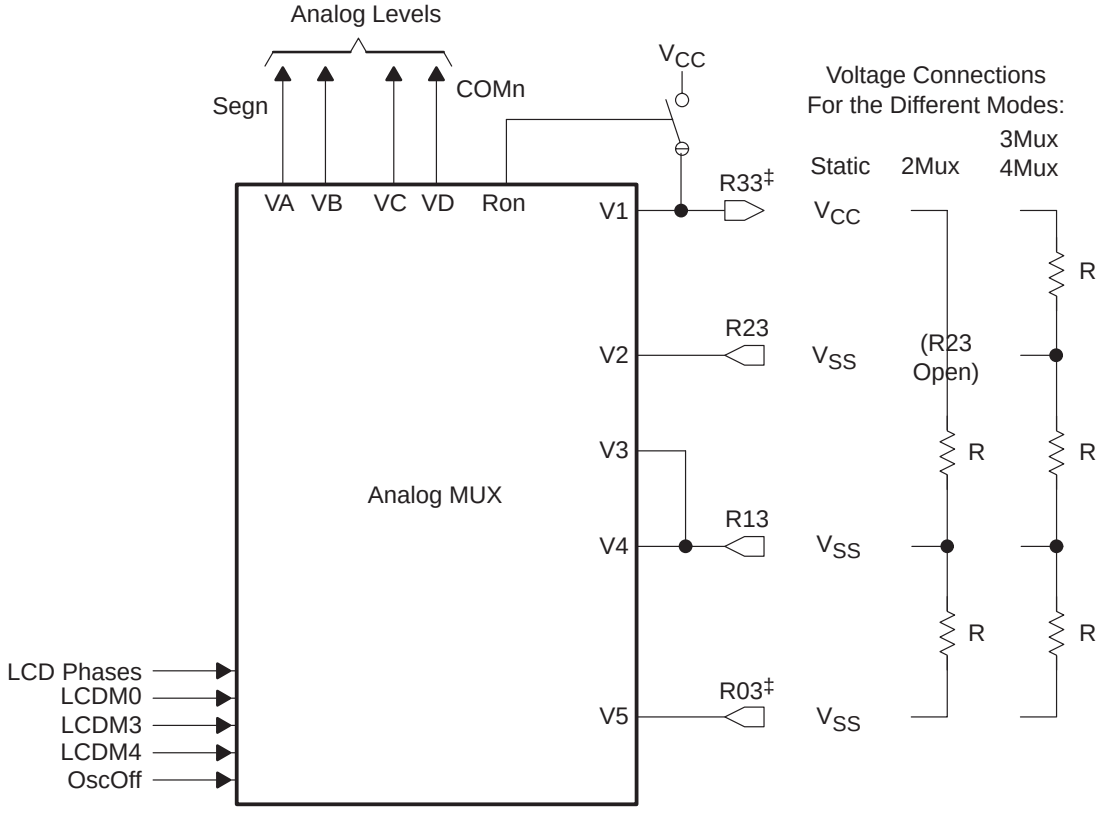
f_{LCD} = 32,768/128 = 256 Hz FRFQ1 = 1; FRFQ0 = 0

14.2.3 LCD Voltage Generation

The voltages required for the LCD signals are supplied externally and are applied to pins R33, R23, R13, and R03 (see Figure 14–6). Generally, the voltages are generated with an equal-weighted resistor ladder. Note that pins R33 and R03 are not present on all MSP430 devices. Check the datasheet for the presence of these pins.

When pins R33 and R03 are not preset, voltage V1 is tied to V_{CC} and voltage V5 is tied to V_{SS} internally. When these pins are present, they provide two advantages to the user. First, R33 is a switched-V_{CC} output. This allows the power to the resistor ladder to be turned off reducing current consumption. Also, when these pins are present, R03 is not tied internally to V_{SS}. This allows the user to control the offset of the LCD voltages thereby providing for temperature compensation or contrast adjustment. If this not desired, the user may simply connect R03 to V_{SS}.

Figure 14–6. External LCD Module Analog Voltage



OscOff	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	RON†
X	X	X	0	0	0	0	0	OFF
1	X	X	X	0	0	0	0	OFF
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	ON
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON

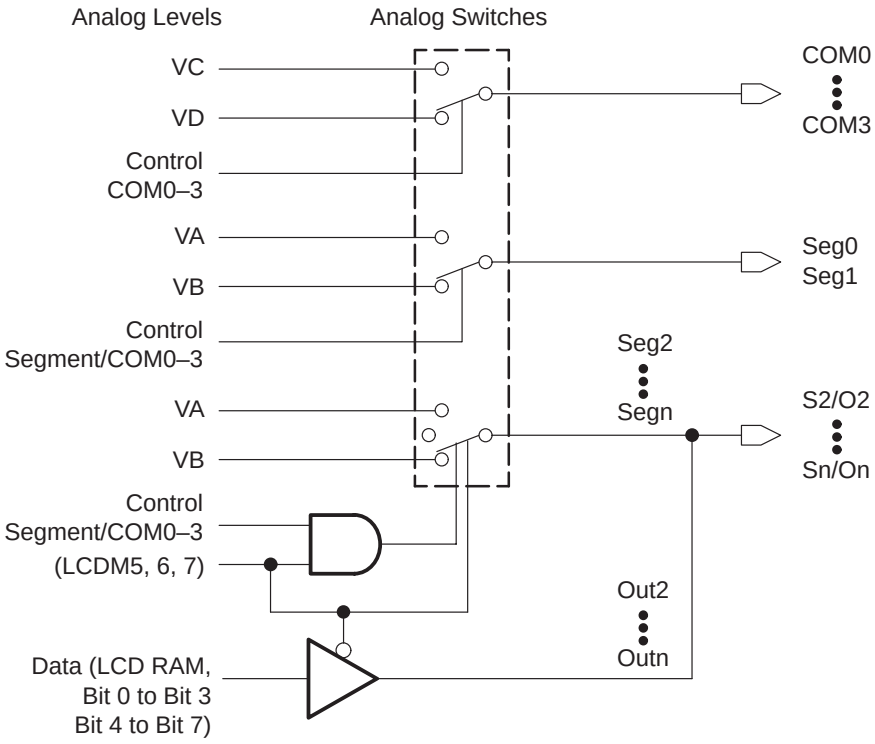
† Indicates the position of the Ron switch, controlled by the LCDM0 bit.

‡ Supply pins for V1 and V5 are optional. Devices without R33 and R03 pins have V1 tied to V_{CC} and V5 tied to V_{SS}. In this case, the resistor ladder should also be tied to V_{CC} and V_{SS}.

14.2.4 LCD Outputs

The LCD outputs use transmission gates to transfer the analog voltage to the output pin where they are used to drive liquid crystal displays. Groups of LCD outputs can be configured to operate as digital outputs as shown in Figure 14–7.

Figure 14–7. Schematic of LCD Output

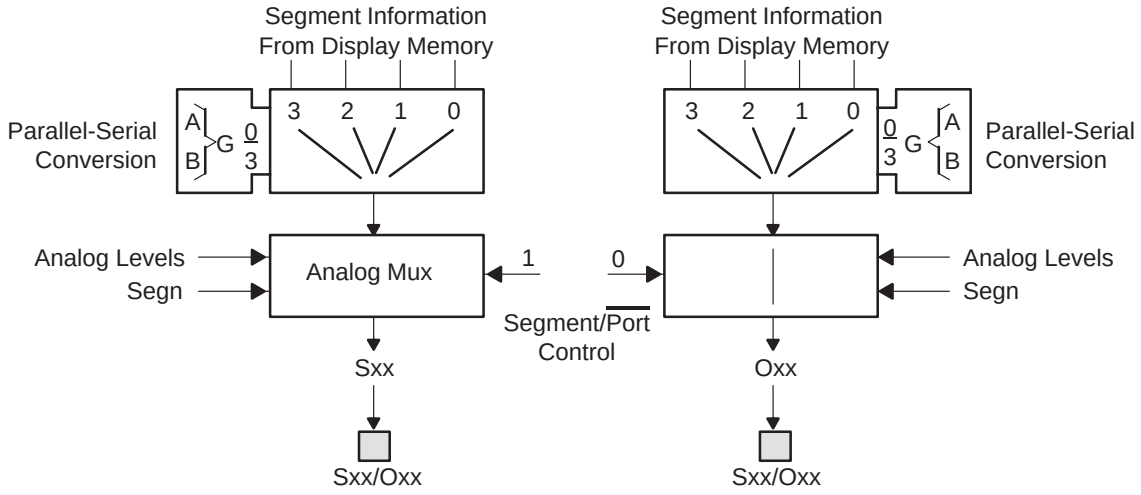


NOTE: The signals VA,VB,VC, and VD are from the LCD analog voltage generator.

14.2.4.1 LCD Port as General-Purpose Outputs

The logic level of an Oxx output is defined by the 4 display memory bits assigned to the pin (see Figure 14–8). All 4 bits must have the same value or the output will not be static. For example, if pin S10/O10 is used as a general-purpose output, Its state is defined by bits 0–3 at memory address 036h and the bits must have the same value.

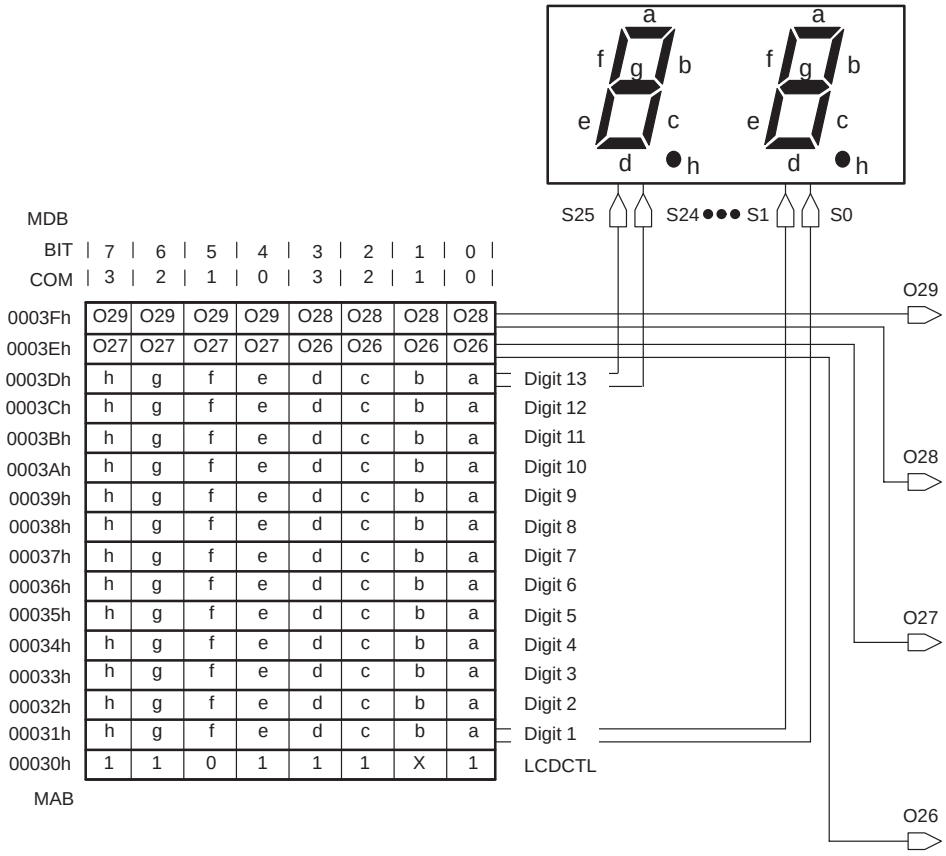
Figure 14–8. Segment Line or Output Line



14.2.4.2 Mixed LCD and Port Mode Application

Figure 14–9 illustrates the mixed mode using a four-MUX LCD drive for 13 digits and one port group as digital outputs. In the example below, digital outputs O26 – O29 are defined to be general-purpose outputs by bits LCDM5, LCDM6, and LCDM7 in the LCDCTL register. In this example, the value of the LCDMx bits is 06h.

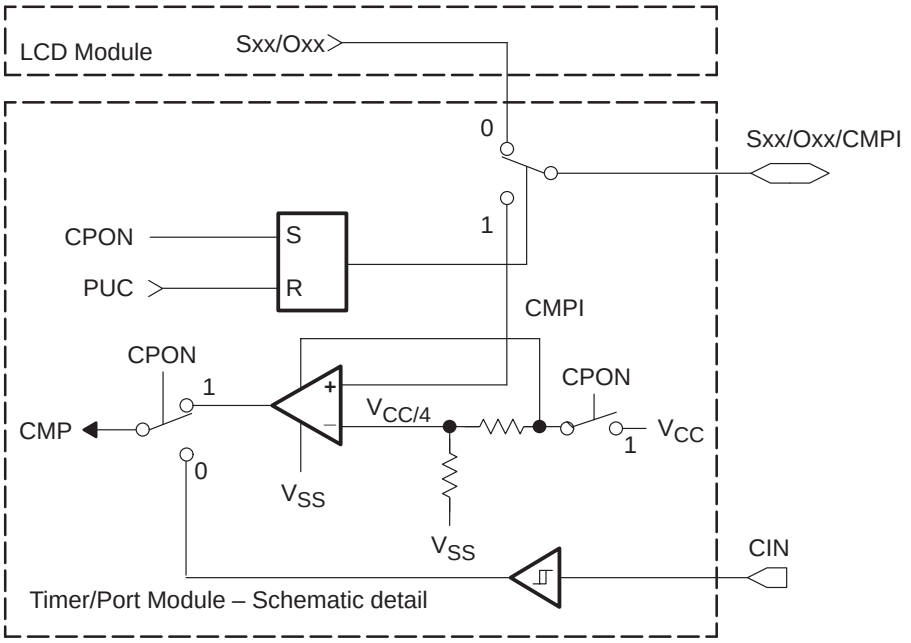
Figure 14–9. Mixed LCD and Port Mode Application



14.2.4.3 LCD Port—Timer/Port Comparator Input

The comparator input associated with the Timer/Port module is typically shared with one segment line as shown in Figure 14–10. The LCD segment function is selected for this pin after the PUC signal is active. The comparator input is selected once the CPON bit—located in the Timer/Port module—has been set. Once the CPON bit is set, the comparator input remains selected for the pin until it is deselected by a PUC signal. Therefore, this pin is not available for the LCD function if it is used for the comparator function. Additionally, once selected for the CMPI function, it can not be switched back to the LCD function without a PUC (power-up clear).

Figure 14–10. Schematic of LCD Pin – Timer/Port Comparator



NOTE: The comparator is selected with the CPON bit. It remains selected, consumes current, and the comparator reference consumes current as long as the CPON bit is set.

14.2.5 LCD Control Register

The LCD control register contents define the mode and operating conditions. The LCD module is byte structured and should be accessed using byte instructions (suffix .B). All LCD control register bits are reset with a PUC signal.

Figure 14–11. LCD Control and Mode Register

LCDCTL 030h

7							0
LCDM7	LCDM6	LCDM5	LCDM4	LCDM3	LCDM2	LCDM1	LCDM0
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

LCDM0: LCDM0 = 0: The timing generator is switched off. Common and segment lines are low. Ron is off. Outputs selected as port output lines are not affected.

LCDM0 = 1: Common and segment lines active. Ron is on. Outputs selected as port output lines are not affected.

LCDM1: Not used

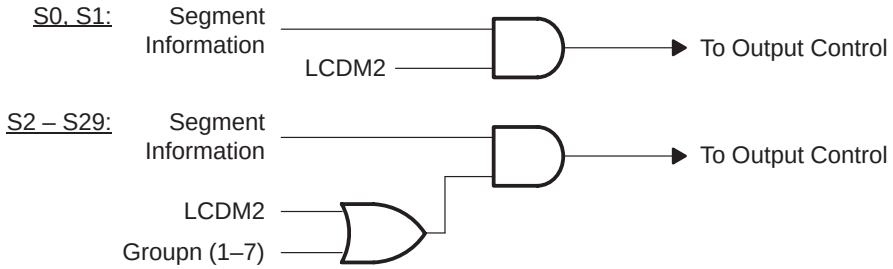
LCDM2 to 4: These three bits select the display mode as described in Table 14–1.

Table 14–1. LCDM Selections

LCDM4	LCDM3	LCDM2	Display Mode
X	X	0	All segments are deselected. The port outputs remain stable. This supports flashing LCD applications.
0	0	1	Static mode
0	1	1	2 MUX mode
1	0	1	3 MUX mode
1	1	1	4 MUX mode

The primary function of the LCDM2 bit is to support flashing or blinking the LCD. The LCDM2 bit is logically ANDed with each segment's display memory value to turn each LCD segment on or off (see Figure 14–12). When LCDM2=1, each LCD segment is on or off according to the LCD display memory. When LCDM2=0, each LCD segment is off, therefore blanking the LCD.

Figure 14–12. Information Control



LCDM5 to 7: These three bits select groups of outputs to be used for LCD segment drive or as general-purpose outputs, as described in Table 14–2. The pins selected as general-purpose outputs reflect the state of the corresponding display memory bits and no longer function as part of the LCD segment lines

Table 14–2. LCDM Signal Outputs for Port Functions

LCDM7	LCDM6	LCDM5	Group0	Group1	Group2	Group3	Group4	Group5	Group6	Group7	
0	0	0	S0-S1	O2-O5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29	←reset condition
0	0	1	S0-S1	S2-S5	O6-O9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29	
0	1	0	S0-S1	S2-S5	S6-S9	O10-O13	O14-O17	O18-O21	O22-O25	O26-O29	
0	1	1	S0-S1	S2-S5	S6-S9	S10-S13	O14-O17	O18-O21	O22-O25	O26-O29	
1	0	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	O18-O21	O22-O25	O26-O29	
1	0	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	O22-O25	O26-O29	
1	1	0	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	O26-O29	
1	1	1	S0-S1	S2-S5	S6-S9	S10-S13	S14-S17	S18-S21	S22-S25	S26-S29	

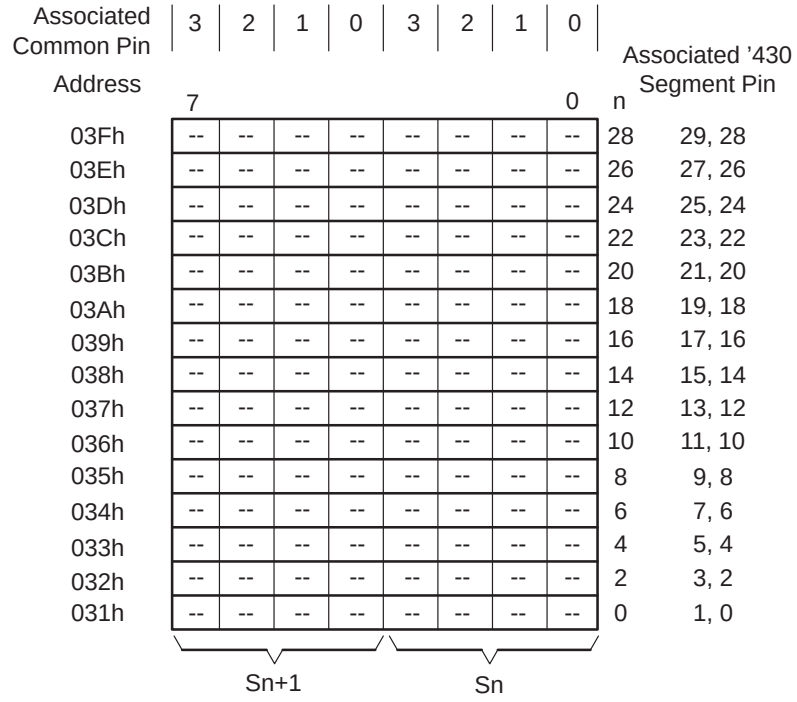
S = LCD segment function
O = GP output function

14.2.6 LCD Memory

The LCD memory map is shown in Figure 14–13. Each individual memory bit corresponds to one LCD segment. To turn on an LCD segment the memory bit is simply set. To turn off an LCD segment, the memory is reset.

The mapping of each LCD segment in an application depends on the connections between the ’430 and the LCD and on the LCD pin-out. Examples for each of the four modes follow including an LCD with pin out, the ’430-to-LCD connections, and the resulting data mapping.

Figure 14–13. Display Memory Bits Attached to Segment Lines

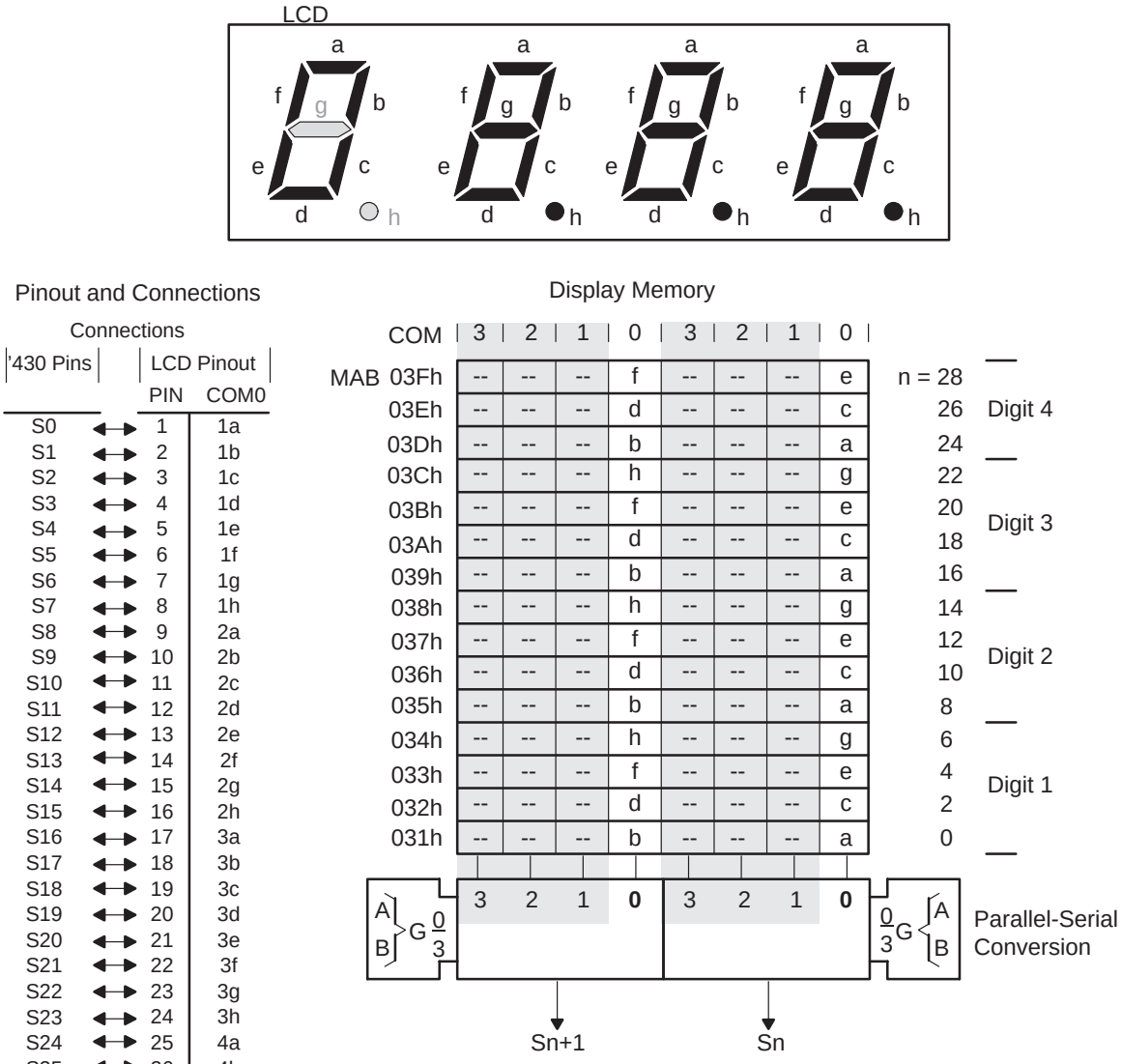


14.2.6.1 Example Using the Static Drive Mode

The static drive mode uses one common line, COM0. In this mode, only bit 0 and bit 4 are used for segment information. The other bits can be used like any other memory.

Figure 14–14 shows an example static LCD, pin-out, LCD-to-’430 connections, and the resulting data mapping. Note this is only an example. Segment mapping in a user’s application completely depends on the LCD pin-out and on the ’430-to-LCD connections.

Figure 14–14. Example With the Static Drive Mode

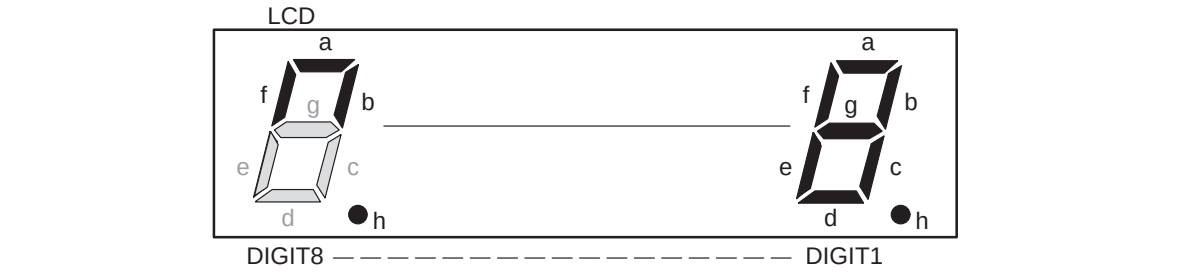


14.2.6.2 Example Using Two-MUX, 1/2-Bias Drive Mode

The two-MUX drive mode uses COM0 and COM1. In this mode, bits 0, 1, 4, and 5 are used for segment information. The other bits can be used like any other memory.

Figure 14–15 shows an example two-MUX LCD, pin-out, LCD-to-'430 connections, and the resulting data mapping. Note this is only an example. Segment mapping in a user's application completely depends on the LCD pin-out and on the '430-to-LCD connections.

Figure 14–15. Example With the Two-MUX Mode



Pinout and Connections

Connections

'430 Pins		LCD Pinout PIN	COM0	COM1
S0	↔	1	1f	1a
S1	↔	2	1h	1b
S2	↔	3	1d	1c
S3	↔	4	1e	1g
S4	↔	5	2f	2a
S5	↔	6	2h	2b
S6	↔	7	2d	2c
S7	↔	8	2e	2g
S8	↔	9	3f	3a
S9	↔	10	3h	3b
S10	↔	11	3d	3c
S11	↔	12	3e	3g
S12	↔	13	4f	4a
S13	↔	14	4h	4b
S14	↔	15	4d	4c
S15	↔	16	4e	4g
S16	↔	17	5f	5a
S17	↔	18	5h	5b
S18	↔	19	5d	5c
S19	↔	20	5e	5g
S20	↔	21	6f	6a
S21	↔	22	6h	6b
S22	↔	23	6d	6c
S23	↔	24	6e	6g
S24	↔	25	7f	7a
S25	↔	26	7h	7b
S26	↔	27	7d	7c
S27	↔	28	7e	7g
S28	↔	29	8f	8a
S29	↔	30	8h	8b
NC	↔	31	8d	8c
NC	↔	32	8e	8g
COM0	↔	33	COM0	
COM1	↔	34		COM1
COM2	NC			
COM3	NC			

Display Memory

MAB	COM	3	2	1	0	3	2	1	0		
	03Fh	--	--	b	h	--	--	a	f	n = 28	1/2 Digit 8
	03Eh	--	--	g	e	--	--	c	d	26	Digit 7
	03Dh	--	--	b	h	--	--	a	f	24	
	03Ch	--	--	g	e	--	--	c	d	22	Digit 6
	03Bh	--	--	b	h	--	--	a	f	20	
	03Ah	--	--	g	e	--	--	c	d	18	Digit 5
	039h	--	--	b	h	--	--	a	f	16	
	038h	--	--	g	e	--	--	c	d	14	Digit 4
	037h	--	--	b	h	--	--	a	f	12	
	036h	--	--	g	e	--	--	c	d	10	Digit 3
	035h	--	--	b	h	--	--	a	f	8	
	034h	--	--	g	e	--	--	c	d	6	Digit 2
	033h	--	--	b	h	--	--	a	f	4	
	032h	--	--	g	e	--	--	c	d	2	Digit 1
	031h	--	--	b	h	--	--	a	f	0	
		3	2	**1**	**0**	3	2	**1**	**0**		Parallel-Serial Conversion

A B G 0 3 — 0 3 G A B

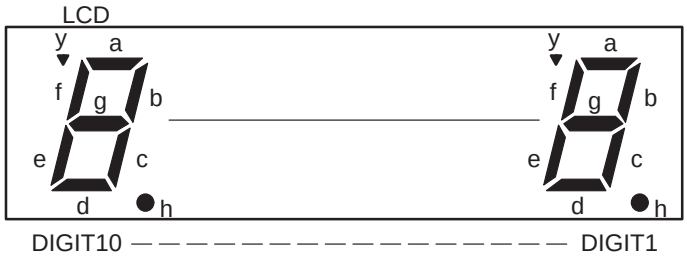
Sn+1 Sn

14.2.6.3 Example Using Three-MUX, 1/3-Bias Drive Mode

The three-MUX drive mode uses COM0, COM1, and COM2. In this mode, bits 0, 1, 2, 4, 5, and 6 are used for segment information. The other bits can be used like any other memory.

Figure 14–16 shows an example three-MUX LCD, pin-out, LCD-to-'430 connections, and the resulting data mapping. Note this is only an example. Segment mapping in a user's application completely depends on the LCD pin-out and on the '430-to-LCD connections.

Figure 14–16. Example With the 3-MUX Mode



Pinout and Connections

Connections

'430 Pins		LCD Pinout PIN	COM0	COM1	COM2
S0	↔	1	1e	1f	1y
S1	↔	2	1d	1g	1a
S2	↔	3	1h	1c	1b
S3	↔	4	2e	2f	2y
S4	↔	5	2d	2g	2a
S5	↔	6	2h	2c	2b
S6	↔	7	3e	3f	3y
S7	↔	8	3d	3g	3a
S8	↔	9	3h	3c	3b
S9	↔	10	4e	4f	4y
S10	↔	11	4d	4g	4a
S11	↔	12	4h	4c	4b
S12	↔	13	5e	5f	5y
S13	↔	14	5d	5g	5a
S14	↔	15	5h	5c	5b
S15	↔	16	6e	6f	6y
S16	↔	17	6d	6g	6a
S17	↔	18	6h	6c	6b
S18	↔	19	7e	7f	7y
S19	↔	20	7d	7g	7a
S20	↔	21	7h	7c	7b
S21	↔	22	8e	8f	8y
S22	↔	23	8d	8g	8a
S23	↔	24	8h	8c	8b
S24	↔	25	9e	9f	9y
S25	↔	26	9d	9g	9a
S26	↔	27	9h	9c	9b
S27	↔	28	10e	10f	10y
S28	↔	29	10d	10g	10a
S29	↔	30	10h	10c	10b
COM0	↔	31	COM0		
COM1	↔	32		COM1	
COM2	↔	33			COM2
COM3	NC				

Display Memory

	COM	3	2	1	0	3	2	1	0		
MAB	03Fh	--	b	c	h	--	a	g	d	n = 28	Digit 10
	03Eh	--	y	f	e	--	b	c	h	26	
	03Dh	--	a	g	d	--	y	f	e	24	Digit 9
	03Ch	--	b	c	h	--	a	g	d	22	Digit 8
	03Bh	--	y	f	e	--	b	c	h	20	Digit 7
	03Ah	--	a	g	d	--	y	f	e	18	
	039h	--	b	c	h	--	a	g	d	16	Digit 6
	038h	--	y	f	e	--	b	c	h	14	Digit 5
	037h	--	a	g	d	--	y	f	e	12	
	036h	--	b	c	h	--	a	g	d	10	Digit 4
	035h	--	y	f	e	--	b	c	h	8	Digit 3
	034h	--	a	g	d	--	y	f	e	6	
	033h	--	b	c	h	--	a	g	d	4	Digit 2
	032h	--	y	f	e	--	b	c	h	2	Digit 1
	031h	--	a	g	d	--	y	f	e	0	

A/B G 0/3 — 3 2 1 0 | 3 2 1 0 — 0/3 G A/B — Parallel-Serial Conversion

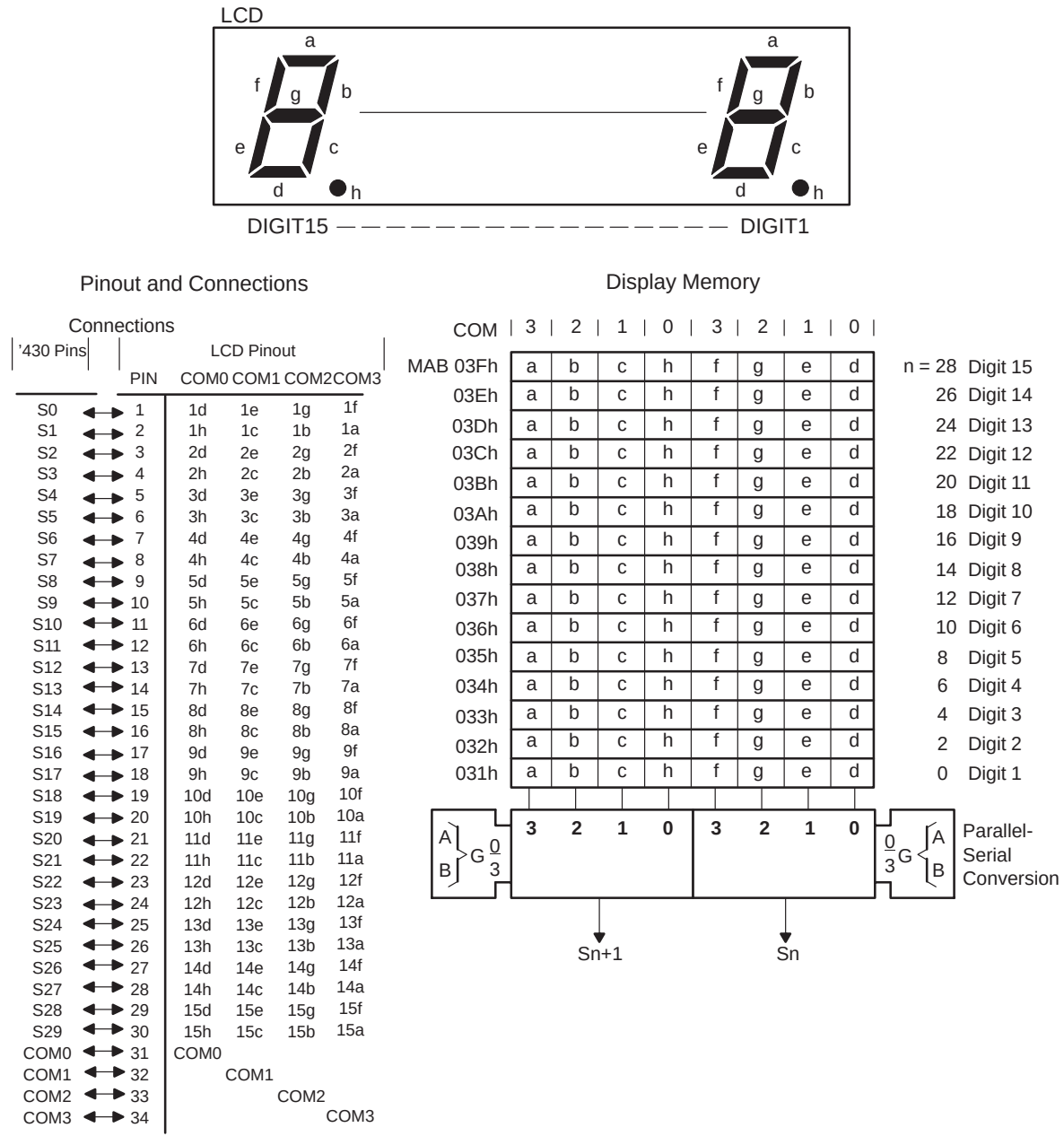
Sn+1 Sn

14.2.6.4 Example Using Four-MUX, 1/3-Bias Drive Mode

The four-MUX drive mode uses all four common lines. In this mode, bits 0 through 7 are used for segment information.

Figure 14–17 shows an example four-MUX LCD, pin-out, LCD-to-'430 connections, and the resulting data mapping. Note this is only an example. Segment mapping in a user's application completely depends on the LCD pin-out and on the '430-to-LCD connections.

Figure 14–17. Example With the Four-MUX Mode



14.3 Code Examples

Code examples for the four modes follow.

14.3.1 Example Code for Static LCD

```
      .sect ”lcd1mux”,0f000h
;  All eight segments of a digit are often located in four
;  display memory bytes with the static display method.
;
a     .EQU  001h
b     .EQU  010h
c     .EQU  002h
d     .EQU  020h
e     .EQU  004h
f     .EQU  040h
g     .EQU  008h
h     .EQU  080h
;  The register content of Rx should be displayed.
:  The Table represents the ’on’–segments according to the
;  content of Rx.

      ..........
;
LCD1  .EQU  00031h
      ..........
      ..........
LCD15 .EQU  0003Fh
;
      ..........
      MOV.B Table (Rx),RY   ; Load segment information
                            ; into temporary memory.
                            ; (Ry) = 0000 0000 hfdb geca
      MOV.B Ry,&LCDn        ; Note:
                            ; All bits of an LCD memory
                            ’ byte are written
      RRA   Ry              ; (Ry) = 0000 0000 0hfd bgec
      MOV.B Ry,&LCDn+1      ; Note:
                            ; All bits of an LCD memory
                            ; byte are written
      RRA   Ry              ; (Ry) = 0000 0000 00hf dbge
      MOV.B Ry,&LCDn+2      ; Note:
                            ; All bits of an LCD memory
                            ’ byte are written
      RRA   Ry              ; (Ry) = 0000 0000 000h fdbg
      MOV.B Ry,&LCDn+3      ; Note:
                            ; All bits of an LCD memory
                            ’ byte are written

      ..........
      ..........
;
Table .BYTE a+b+c+d+e+f     ; displays ”0”
      .BYTE b+c;            ; displays ”1”
      ..........
      ..........
      .BYTE
      ..........
```

14.3.2 Example Code for Two MUX, 1/2-Bias LCD

```
      .sect ”lcd2mux”,0f000h
;  All eight segments of a digit are often located in two
;  display memory bytes with the 2MUX display rate
;
a     .EQU  002h
b     .EQU  020h
c     .EQU  008h
d     .EQU  004h
e     .EQU  040h
f     .EQU  001h
g     .EQU  080h
h     .EQU  010h
;  The register content of Rx should be displayed.
;  The Table represents the ’on’–segments according to the
;  content of Rx.
;
      ..........
;
LCD1  .EQU  00031h
      ..........
      ..........
LCD15 .EQU  0003Fh
;
      ..........
      ..........
      MOV.B Table(Rx),Ry ; Load segment information into
                         ; temporary memory.
      MOV.B Ry,&LCDn     ; (Ry) = 0000  0000  gebh  cdaf
                         ; Note:
                         ; All bits of an LCD memory byte
                         ; are written
      RRA   Ry           ; (Ry) = 0000  0000  0geb  hcda
      RRA   Ry           ; (Ry) = 0000  0000  00ge  bhcd
      MOV.B Ry,&LCDn+1   ; Note:
                         ; All bits of an LCD memory byte
                         ; are written
      ..........
      ..........
      ..........
Table .BYTE a+b+c+d+e+f     ; displays ”0”
      ..........
      .BYTE a+b+c+d+e+f+g+h ; displays ”8”
      ..........
      ..........
      .BYTE
      ..........
;
```

14.3.3 Example Code for Three MUX, 1/3-Bias LCD

```
      .sect ”lcd3mux”,0f000h
;  The 3MUX rate can easily support nine segments for each
;  digit. The nine segments of a digit are located in
;  1 1/2 display memory bytes.
;
a     .EQU  0040h
b     .EQU  0400h
c     .EQU  0200h
d     .EQU  0010h
e     .EQU  0001h
f     .EQU  0002h
g     .EQU  0020h
h     .EQU  0100h
Y     .EQU  0004h
;  The  LSDigit of register Rx should be displayed.
;  The Table represents the ’on’–segments according to the
;  LSDigit of register of Rx.
;  The register Ry is used for temporary memory
;
LCD1  .EQU  00031h
      ..........
LCD15 .EQU  0003Fh
      ..........
      ..........
ODDDIGRLA   Rx           ; LCD in 3MUX has 9 segments per
                         ; digit; word table required for
                         ; displayed characters.
      MOV   Table(Rx),Ry ; Load segment information to
                         ; temporary mem.
                         ; (Ry) = 0000  0bch  0agd  0yfe
      MOV.B Ry,&LCDn     ; write ’a, g, d, y, f, e’ of
                         ; Digit n (LowByte)
      SWPB  Ry           ; (Ry) =  0agd  0yfe  0000  0bch
      BIC.B #07h,&LCDn+1 ; write ’b, c, h’ of Digit n
                         ; (HighByte)
      BIS.B Ry,&LCDn+1
      .....
EVNDIGRLA   Rx           ; LCD in 3MUX has 9 segments per
                         ; digit; word table required for
                         ; displayed characters.
      MOV   Table(Rx),Ry ; Load segment information to
                         ; temporary mem.
                         ; (Ry) = 0000  0bch  0agd  0yfe
      RLA   Ry           ; (Ry) = 0000  bch0  agd0  yfe0
      RLA   Ry           ; (Ry) = 000b  ch0a  gd0y  fe00
      RLA   Ry           ; (Ry) = 00bc  h0ag  d0yf  e000
      RLA   Ry           ; (Ry) = 0bch  0agd  0yfe  0000
      BIC.B #070h,&LCDn+1
      BIS.B Ry,&LCDn+1   ; write ’y, f, e’ of Digit n+1
                         ; (LowByte)
      SWPB  Ry           ; (Ry) = 0yfe  0000  0bch  0agd
      MOV.B Ry,&LCDn+2   ; write ’b, c, h, a, g, d’ of
                         ; Digit n+1 (HighByte)
      ..........
Table .WORD a+b+c+d+e+f  ; displays ”0”
      .WORD b+c          ; displays ”1”
      ..........
      ..........
      .WORD a+e+f+g      ; displays ”F”
```

14.3.4 Example Code for Four MUX, 1/3-Bias LCD

```
       .sect ”lcd4mux”,0f000h
;  The 4MUX rate is the most easy–to–handle display rate.
;  All eight segments of a digit can often be located in
;  one display memory byte
a      .EQU  080h
b      .EQU  040h
c      .EQU  020h
d      .EQU  001h
e      .EQU  002h
f      .EQU  008h
g      .EQU  004h
h      .EQU  010h
;
;  The  LSDigit of register Rx should be displayed.
;  The Table represents the ’on’–segments according to the
;  content of Rx.
;
       ...........
LCD1   .EQU  00031h          ; Address of LC Display Memory
       ...........
       ...........
LCD15  .EQU  0003Fh
       ...........
       ...........
       ...........
       MOV.B Table(Rx),&LCDn ; n = 1 ..... 15
                             ; all eight segments are
                             ; written to the display
                             ; memory
       ...........
       ...........

Table  .BYTE a+b+c+d+e+f     ; displays ”0”
       .BYTE b+c             ; displays ”1”
       ...........
       ...........
       .BYTE b+c+d+e+g       ; displays ”d”
       .BYTE a+d+e+f+g       ; displays ”E”
       .BYTE a+e+f+g         ; displays ”F”
```

Chapter 15

ADC12+2 A-to-D Converter

The ADC12+2 features include:

- ❑ Eight analog or digital input channels
- ❑ A programmable current source on four analog pins
- ❑ Ratiometric or absolute measurement
- ❑ Built-in sample-and-hold
- ❑ End-of-conversion interrupt flag
- ❑ ADAT register that holds conversion results until the next start of conversion
- ❑ Low-power consumption
- ❑ Stand-alone conversion without CPU processing overhead
- ❑ Programmable 12-bit or 14-bit resolution
- ❑ Four programmable ranges that give 14-bit dynamic range
- ❑ Fast-conversion time
- ❑ Large supply-voltage range
- ❑ Monotonic conversion

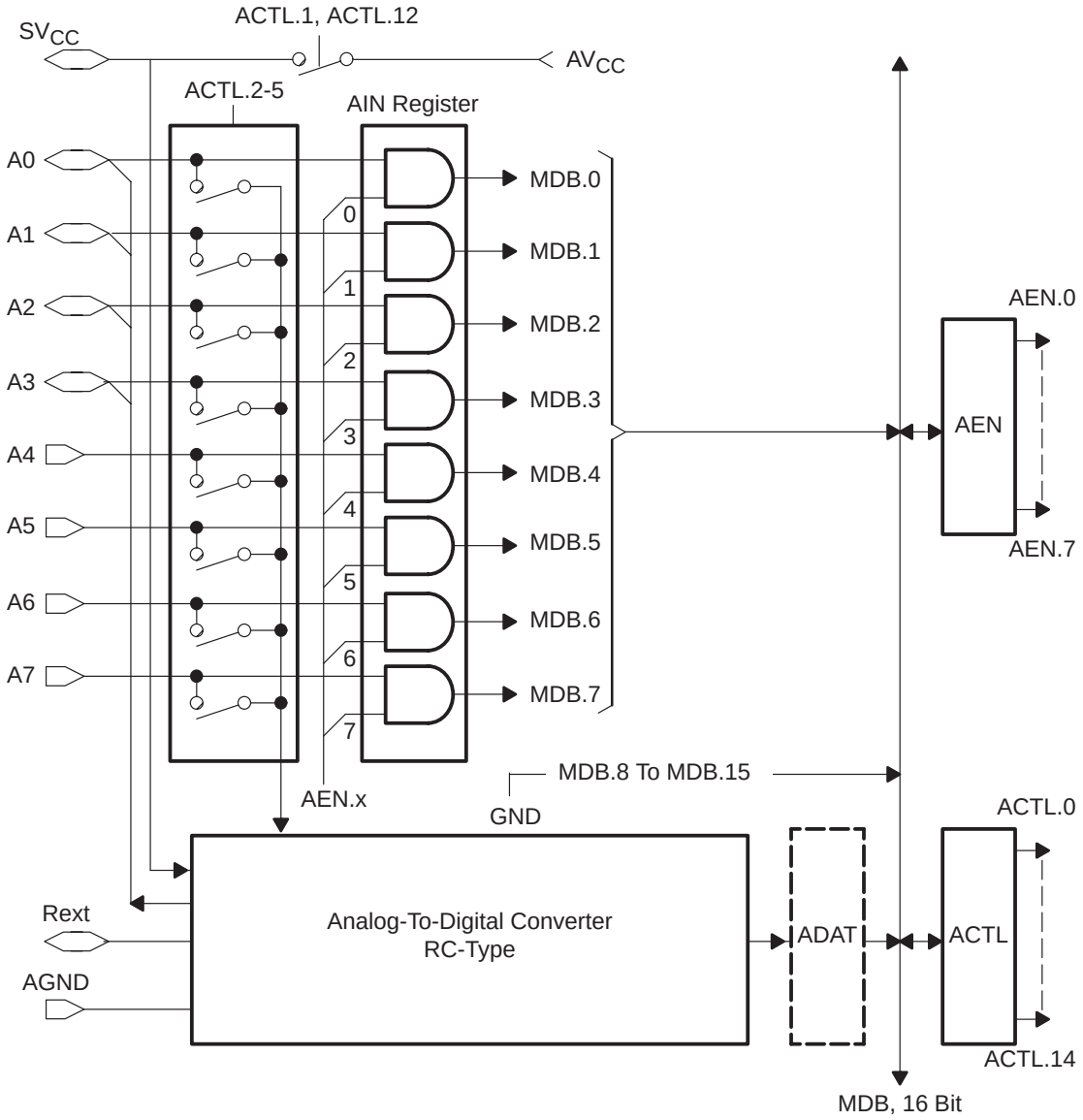
Topic **Page**

15.1 Introduction 15-2
15.2 Analog-to-Digital Operation 15-4
15.3 ADC12+2 Control Registers 15-13

15.1 Introduction

The 12+2-bit ADC is a peripheral module accessed using word instructions. Conversion results are contained in the ADAT register. The converted bits are visible during a conversion and are immediately available to be read at the end of conversion in the ADAT register. The conversion result is not cleared until the next conversion is initiated by setting the SOC bit in the ACTL register. The SOC bit clears the ADAT register for the new result and starts the ADC12+2 clock for another conversion. Figure 15–1 shows the ADC12+2 module configuration.

Figure 15–1. ADC12+2 Module Configuration



The ADC12+2 module has eight individually-configurable input channels. A conversion can be made on any one of these channels at any time. Four of the channels, A0, A1, A2, and A3, may also be configured as current source outputs whose values can be programmed by external resistor R_{ext}. Any of the current source outputs can be turned on (one at a time) to drive external sensors in order to make ratiometric measurements. Absolute measurements can also be made by applying an external reference to pins SV_{CC} or AV_{DD}.

Additionally, the eight channels can be configured as digital inputs. Each input channel is individually configurable, so each input may be either an analog or a digital input. The selection is made with the bits in the AEN register. When used as digital inputs, the values of the digital input signals are read from the AIN register.

Note:

When sensitive analog conversion takes place, any digital activity on adjacent channels may cause crosstalk and interference, giving noisy or incorrect conversion results.

The converter has two modes of operation: 12-bit, and 12+2-bit conversion, depending on the status of ACTL register bit 11. When the range of the input signal is known the input range may be preselected and the converter can be used in 12-bit mode. The converter samples the input and then converts it to 12 bits of resolution within any one of the four ranges (see subsection 15.2.3).

In 12+2-bit mode (setting ACTL register bit 11), the range is automatically selected by the converter to resolve to 14 bits. The input is sampled twice: once for the 2-bit range selection, and once again for the remaining 12-bits of the conversion, to give a 12+2-bit result.

In both modes, when a conversion is completed, the interrupt flag (EOC) is set automatically. The EOC signal disables the ADC clock to conserve power until the SOC bit is set again.

Note: ADC, Start-of-Conversion

A conversion must always be completed before the next conversion is initiated. Otherwise, unpredictable conversion data will result.

When powered-down (Pd bit in ACTL register), the ADC current consumption is stopped. This is valid while SV_{CC} is not externally driven. Upon a conversion start-up or a power-up signal the converter wakes up, but it can take up to 6 µs to reach steady-state conditions.

15.2 Analog-to-Digital Operation

The following sections describe the ADC12+2 and operation.

15.2.1 A/D Conversion

After power-up, the ACTL register must be programmed to make a ratiometric or absolute measurement and to manually or automatically select a range. In manual (12-bit) mode, once the range bits are selected they cannot be changed during the conversion, as this invalidates the results.

Setting the SOC bit in the ACTL register activates the ADC clock to begin a new conversion. The conversion is based on a successive approximation technique that uses a resistor array to resolve the M MSBs first, and uses a switched capacitor array to resolve the remaining L LSBs.

The resistor array, consisting of 2^M individually and equally weighted resistors, forms a DAC; the capacitor array, consisting of L capacitors, forms an A/D charge redistribution. The capacitors are binary-weighted. The number of capacitors corresponds to the converter range, or to the digital-output code L bits.

The sequence, shown in Figure 15–2, starts by selecting the applicable analog channel and sampling the analog-input voltage onto the top plates of the capacitor array. The analog multiplexer is then disconnected from the ADC and the analog input does not need to be present after this sample period.

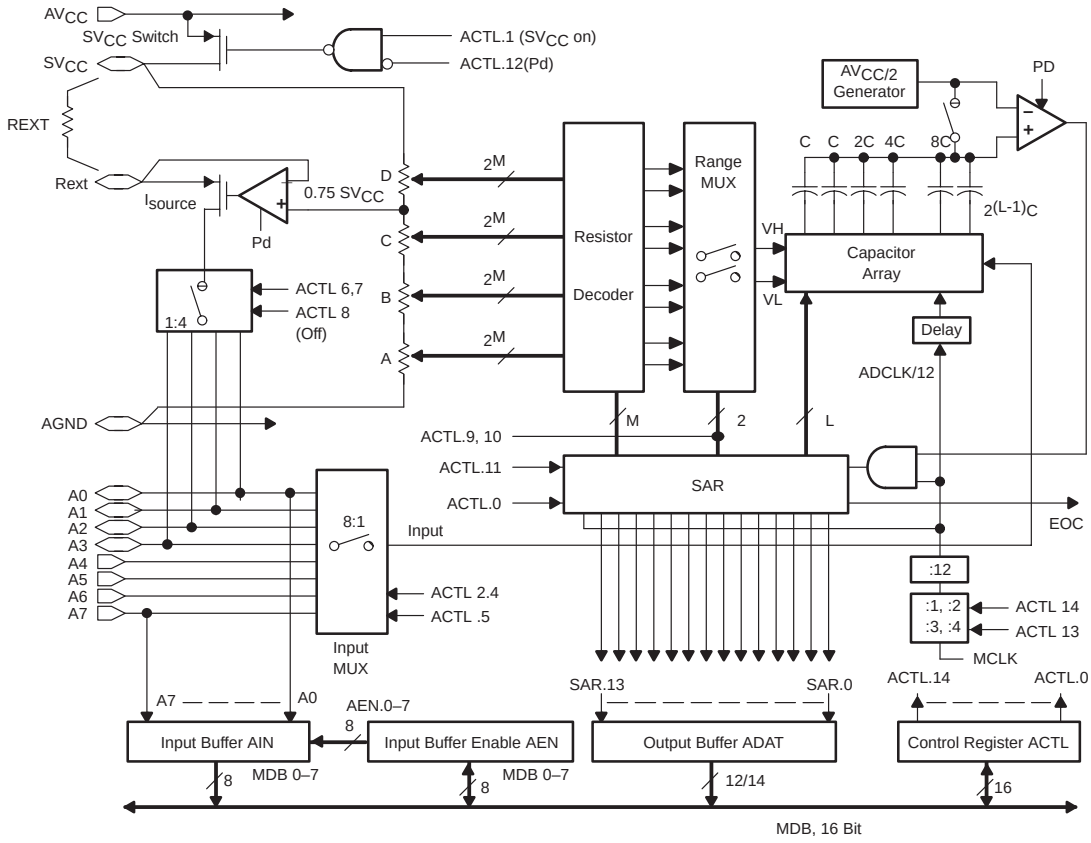
A successive approximation is performed on the resistor string to find the tap that corresponds to a voltage within 2^L LSBs of VIN. This yields the VH and VL voltages across one element of the resistor array, and resolves the M MSBs. The capacitor array then resolves the difference voltage (VH–VL) to L bits of resolution using a similar successive approximation search on the capacitor array, starting with the MSB capacitor.

This switching procedure continues with the MSB or largest capacitor to the smallest (LSB) capacitor in the capacitor array, whereby the initial charge is redistributed among the capacitors. The particular setting of the switches (both in the resistor array and in those connected to the bottom plates of the capacitors) has then induced a change on the top plate that is as close to the input voltage (VIN) as possible. The switch settings then correspond to the binary code [12-bit or 12+2-bit] that represents the fraction VIN/VREF.

The top plate voltage is monitored by a comparator with built-in input-offset cancellation circuitry that senses whether the input voltage is less or greater than the voltage on the top plate. It generates a digital output that determines the direction of the successive approximation search.

When this sequence is completed, the top plate voltage is as close to zero as the converter resolution allows, and the LSB is determined. An EOC signal is then sent to indicate that the conversion result is available from the ADAT register.

Figure 15–2. ADC12+2 Schematic



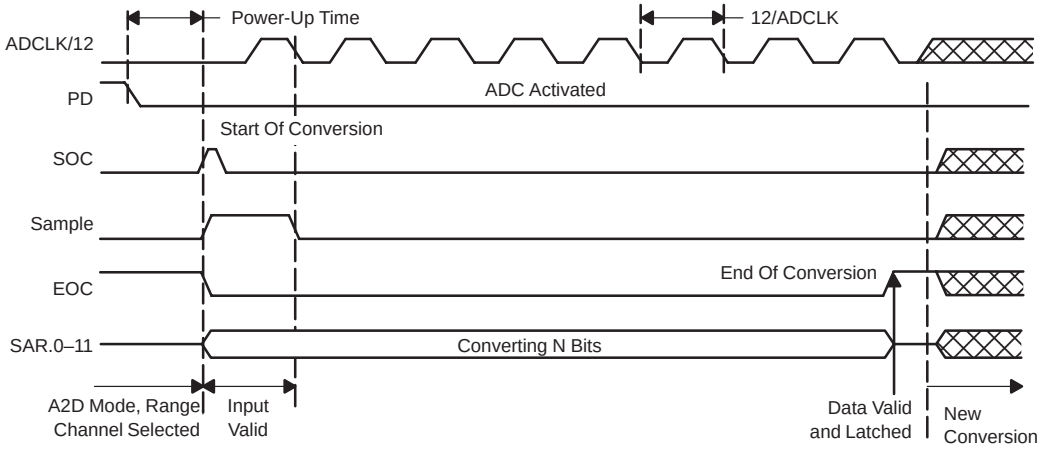
15.2.1.1 A/D Conversion Timing

After the ADC12+2 module is activated (Pd bit is reset), at least 6 µs must elapse before a new conversion is attempted in order to allow the correct internal biases to be established.

The ADC always runs at one-twelfth the clock rate of the ADCLK. ADCLK is always MCLK divided by 1, 2, 3, or 4. The ADCLK frequency must be chosen to meet the conversion time defined in the electrical characteristics (see device's data sheet). The ADCLK frequency is selected with two bits (ADCLK) in control register ACTL. If the ADCLK is too fast, an accurate conversion to 12 bits cannot be guaranteed due to the internal time constants associated with analog input sampling and the conversion network. Also, if the ADCLK is too slow, an accurate conversion to 12 bits cannot be guaranteed, due to charge loss within the ADC-capacitor array, even if the input signal is valid and steady for the required acquisition time.

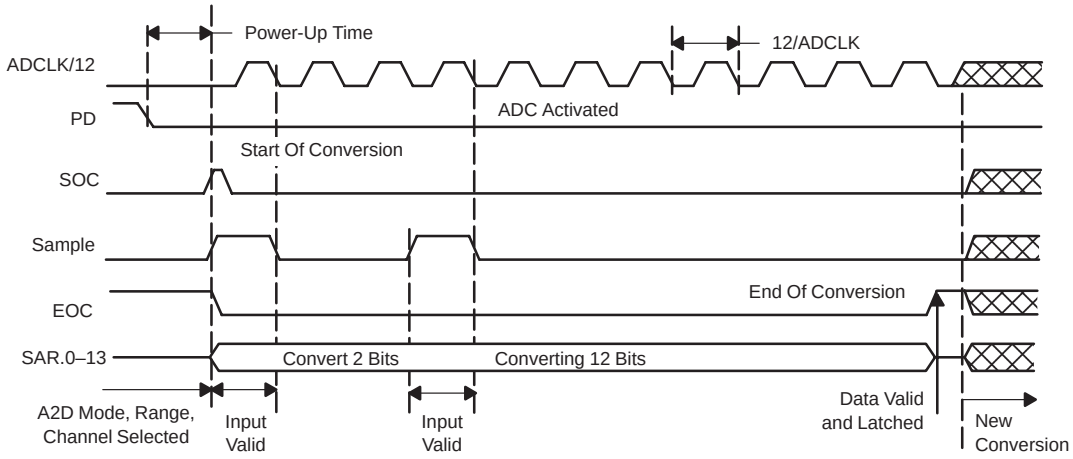
Sampling the analog input signal takes 12 ADCLK pulses, and the 12-bit conversion takes 84 (12 × 7) additional ADCLK cycles. Therefore, a 12-bit conversion with a preselected range takes 96 ADCLK cycles. This is illustrated in Figure 15–3.

Figure 15–3. ADC12+2 Timing, 12-Bit Conversion



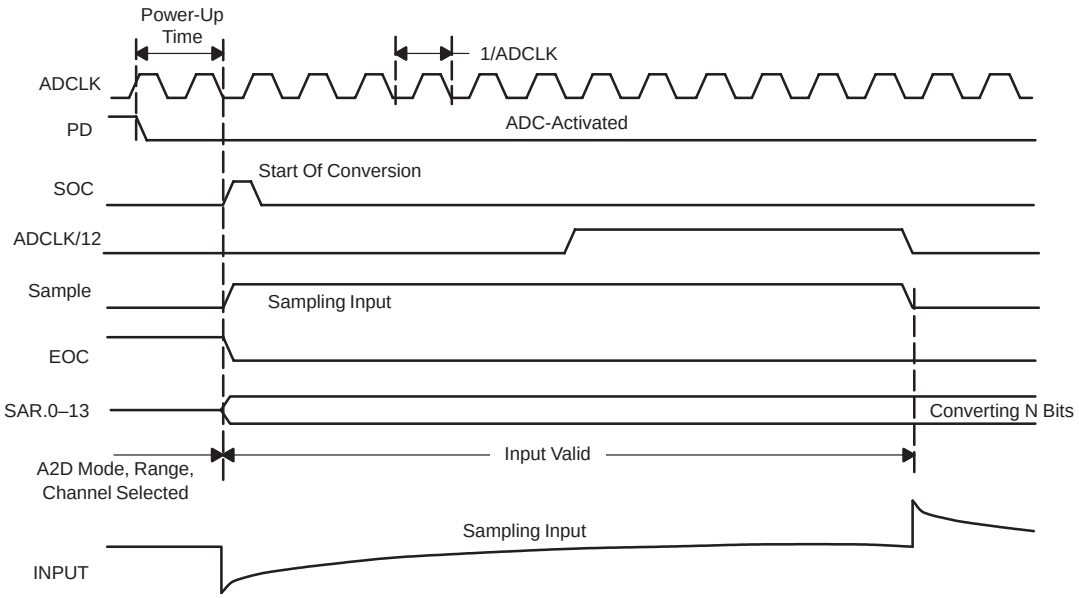
In 12+2-bit mode, the analog input signal is sampled twice, each sampling taking twelve ADCLK clock pulses. After the first sampling of the input signal, the range conversion occurs and takes 24 ADCLK clocks. After the second sampling of the input signal (the second sampling occurs automatically), the 12-bit conversion occurs and takes 84 (12×7) additional ADCLK clock cycles. Altogether, the 12+2-bit conversion takes 132 ADCLK cycles as illustrated in Figure 15–4.

Figure 15–4. ADC12+2 Timing, 12+2-Bit Conversion



The input signal must be valid and steady during the sampling period for an accurate conversion (Figure 15–5). To ensure that supply glitching and ground bounce errors or crosstalk interferences do not corrupt the results, avoid digital activity on channels adjacent to the analog input during the conversion.

Figure 15–5. ADC, Input Sampling Timing



The ADC12+2 uses the charge redistribution method and thus the internal switching of the inputs for sampling causes displacement currents to flow in and out of the analog inputs. These current spikes or transients occur at the leading and falling edges of the internal sample pulse. They quickly decay and settle before causing any problems, because the time constant is less than that of the effective internal RC. Internally, the analog inputs see a nominal RC consisting of a nominal 40-pF (C-array) capacitor in series with a nominal 2-kΩ resistor (R_{on} of switches). However, if the external dynamic-source impedance is large, these transients might not settle within the allocated sampling time to within 12 or 12+2 bits of accuracy.

15.2.2 A/D Interrupt

When an A/D conversion is complete, the EOC signal goes high, setting the interrupt flag ADIFG. The ADIFG flag is located in the SFR registers in IFG2.2. The flag is automatically reset when the interrupt is serviced.

Two additional bits control the generation of a CPU interrupt: The ADIE bit in the SFR register (IE2.2) and the GIE bit. The ADIE bit is an individual bit to enable or disable the A/D interrupt—its initial state is reset. The GIE bit is the global interrupt enable bit. When both bits are set, a CPU interrupt is generated at the end of an A/D conversion.

15.2.3 A/D Ranges

One of four ranges can be selected manually to yield 12 bits of resolution within any given range. The range is defined with bits ACTL.9 and ACTL.10 prior to conversion. The converter can also find the appropriate range automatically, resulting in an overall 12+2-bit conversion.

The ranges are:

$0.00 \times VREF \leq VIN < 0.25 \times VREF$ Range A
$0.25 \times VREF \leq VIN < 0.50 \times VREF$ Range B
$0.50 \times VREF \leq VIN < 0.75 \times VREF$ Range C
$0.75 \times VREF \leq VIN < 1.00 \times VREF$ Range D

Where:

VREF is the voltage at the SV_{CC} pin, either applied externally or derived by closing the SV_{CC} switch with bit 12 of the ACTL register.

After the proper range is selected, the input channel, selected by the applicable bits in the control register, is connected to the converter input. The ADC processes the signal at the selected input channel, and the software can then access the conversion result through the ADAT register.

The digital code (decimal) expected within any one range is:

$$N_{typ} = INT \left| \frac{VIN \times 2^{14}}{VREF} - 2^{13} \times ACTL.10 - 2^{12} \times ACTL.9 \right|$$

Where:

ACTL.10 and ACTL.9 are bits 10 and 9 (respectively) in the ACTL register.

Thus, for a 12-bit conversion, the ranges are:

$0000h \leq N \leq 0FFFh$ Range A
$0000h \leq N \leq 0FFFh$ Range B
$0000h \leq N \leq 0FFFh$ Range C
$0000h \leq N \leq 0FFFh$ Range D

and for a 12+2-bit conversion:

$0000h \leq N \leq 3FFFh$

Note: ADC12+2 Offset Voltage

Any offset voltage (V_{io}) due to voltage drops at the bottom or top of the resistor array, caused by parasitic impedances to the SV_{CC} pin or the ground AGND pin, distorts the digital code output and formula.

15.2.4 A/D Current Source

When the ADC12+2 is used in sensor applications in conjunction with resistive elements, current sources may be required so that the input signal can be referred back to the supply voltage or voltage reference. This allows a ratiometric measurement independent of the accuracy of the reference.

One of four analog channels can be used for the current-source output, as shown in Figure 15–6. The current-source (Isource) output can be programmed by an external resistor (Rext) and is then available on pins A0, A1, A2, and A3, with the value:

$I_{source} = (0.25 \times SV_{CC})/Rext$

Where:

SV_{CC} is the voltage at pin $SV_{CC,}$ and Rext is the external resistor between pins SV_{CC} and Rext.

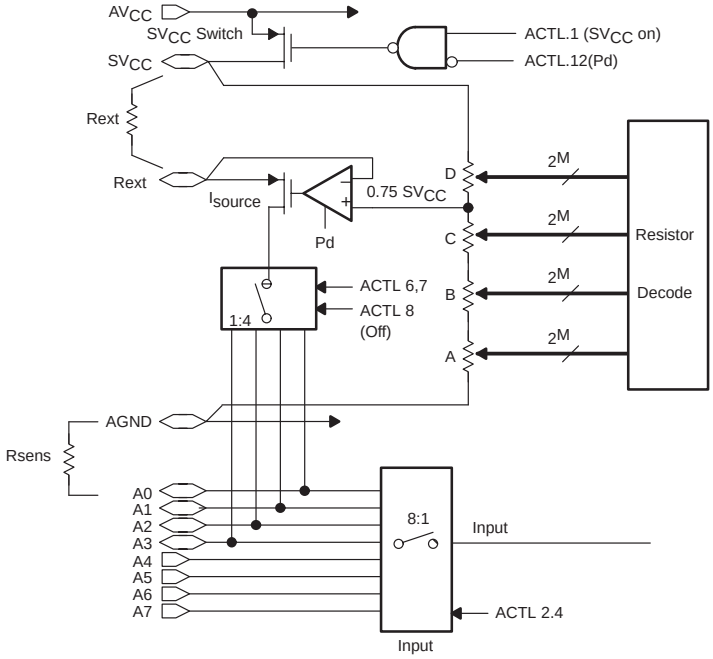
Therefore, for ratiometric measurements, the voltage (V_{in}), developed across the channel input with the resistive elements (channels A0, A1, A2, and A3 only) is:

$$Vin = (0.25 \times SV_{CC}) \times (Rsens/Rext)$$

Where:

R_{sens} is the external resistive element.

Figure 15–6. A/D Current Source



15.2.5 Analog Inputs and Multiplexer

The analog inputs and the multiplexer are described in the following sections.

15.2.5.1 Analog Inputs

The analog-input signal is sampled onto an internal capacitor and held during conversion. The charge is supplied by the input source, and the charging time is defined to be twelve ADCLK clocks. Therefore, the external source resistances and dynamic impedances must be limited so that the RC time constant is short enough to allow the analog inputs to completely settle to 12-bit accuracy within the allocated sampling time. This time constant is typically less than $0.8/f_{ADCLK}$.

High source impedances have an adverse affect on the accuracy of the converter, not only due to RC-settling behavior, but also due to input voltage drops as a result of leakage current or averaged dc-input currents (input

switching currents). Typically, for a 12-bit converter, the error in LSBs due to leakage current is:

Error (LSBs) = 4 × (μA of leakage current) × (kΩ of source resistance)/(volt of VREF).

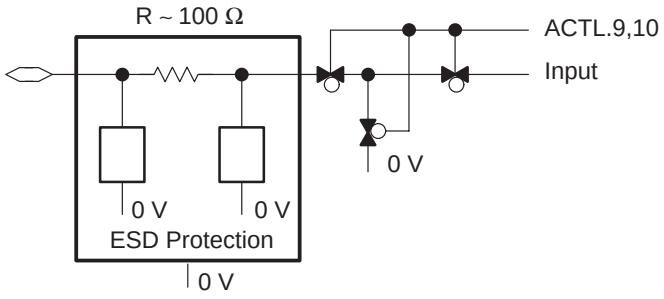
Example: 50-nA leakage, 10-kΩ source resistance, 3-V VREF results in 0.7 LSBs of error.

This also applies to the output impedance of the voltage-reference source VREF. The impedance must be low enough to enable the transients to settle within ($0.2/_{ADCLK}$) seconds and to generate leakage-current-induced errors of << 1LSB.

15.2.5.2 Analog Multiplexer

The analog multiplexer selects one of eight single-ended input channels, as determined by the ACTL register bits. It is based on a T-switch to minimize the coupling between channels, which corrupts the analog input. Channels that are not selected are isolated from the ADC and the intermediate node connected to the analog ground (AGND) so that the stray capacitance is grounded to eliminate crosstalk.

Figure 15–7. Analog Multiplexer



Crosstalk exists because there is always parasitic coupling capacitance across and between switches. This can take several forms, such as coupling from the input to the output of an off switch, or coupling from an off-analog input channel to the output of an adjacent *on* output channel, causing errors. Thus, for high-accuracy conversions, crosstalk interference must be minimized through shielding and other well-known printed-circuit board (PCB) layout techniques.

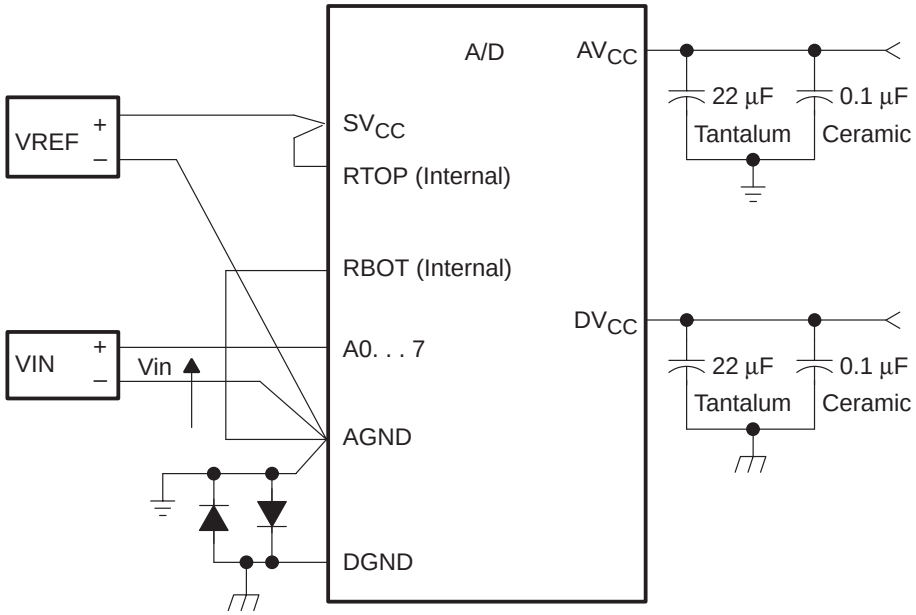
15.2.6 A/D Grounding and Noise Considerations

As with any high-resolution converter (≥ 12 bits), care and special attention must be given to the printed-circuit board layout and the grounding scheme to eliminate ground loops and any unwanted parasitic components/effects and noise. Many common techniques are documented in application notes that address these issues.

Ground loops can be formed when the ADC12+2 resistor-divider return current flows through traces that are common to other analog or digital

circuitry. This current can generate small unwanted offset voltages that can add to or subtract from the ADC reference or input voltages. One way to avoid ground loops is to use a star-connection scheme for the AGND; in this way, the ground or reference currents do not flow through any common input leads, eliminating any voltage errors (see Figure 15–8).

Figure 15–8. A/D Grounding and Noise Considerations



The digital ground (DGND) and the analog ground (AGND) can also be star-connected together. However, if separate supplies are used, two reverse-biased diodes limit the voltage difference to less than ±700 mV (see Figure 15–8).

Power-supply rippling and noise spikes from digital switching or switching power supplies can cause conversion errors. Normally, the internal ADC noise is very small and the total input-referred noise is far less than one LSB, so the output code is fairly stable. However, as noise couples into the device through the supply and ground, the noise margin is reduced, and code uncertainty and jitter can result. Several readings might be required to average out the noise effects.

Another consequence of noise is that, as one of the reference voltages SV_{CC} or VREF is reduced, the absolute value of the LSB is also reduced. Therefore, the noise becomes even more dominant. Thus, a clean, noise-free design becomes even more important to achieve the desired accuracy.

In addition to physical layout techniques, adding carefully-placed bypass capacitors returned to the respective ground planes helps to stabilize the supply current and minimize the noise.

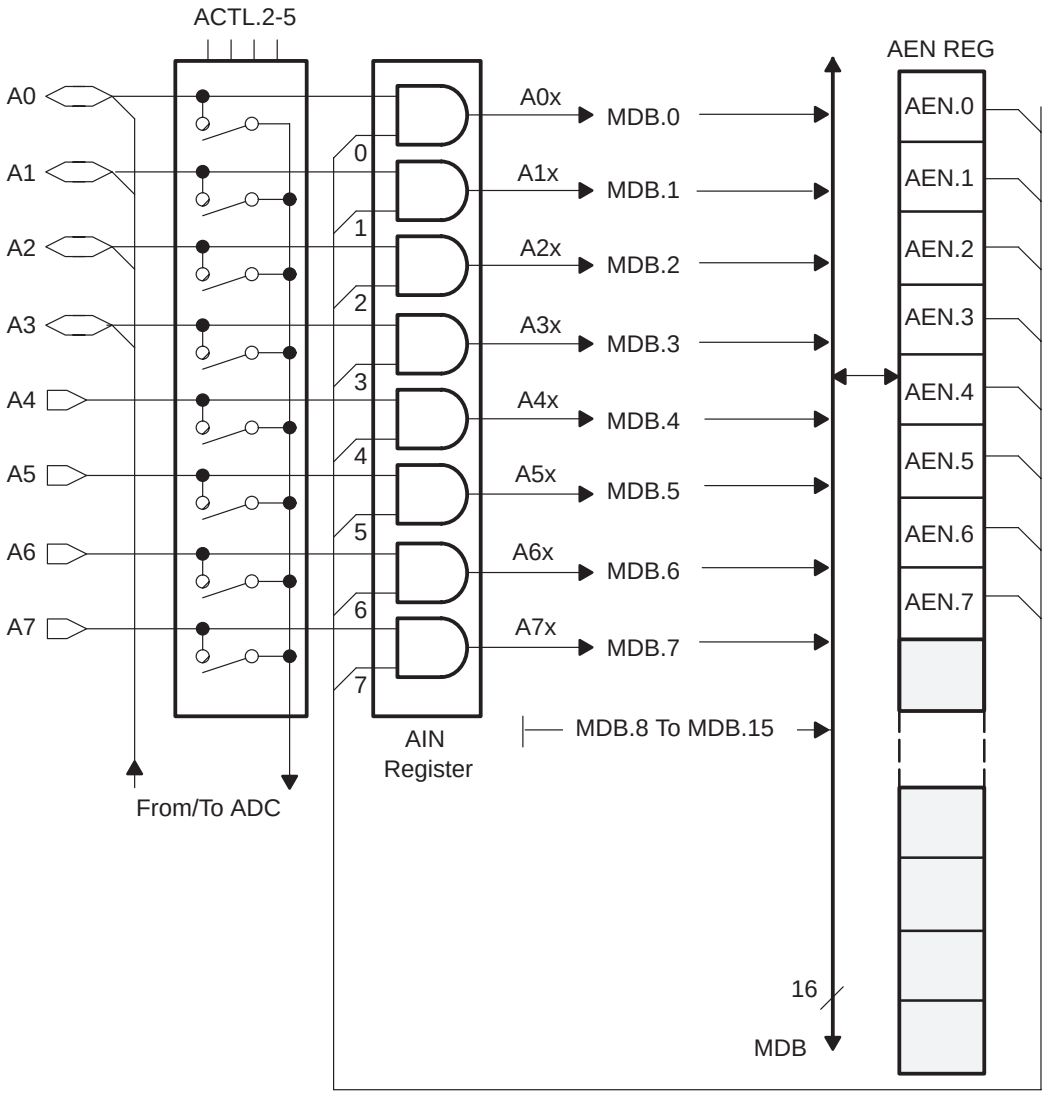
15.2.7 A/D Converter Input and Output Pins

The following sections describe the various ADC12+2 pins.

15.2.7.1 Input Pins

There are two different types of input signals: analog signals A0 through A7, and signals I_{SOURCE} and SV_{CC}. The input signals coming from channels A0 to A7 are configurable as ADC analog signals or as digital inputs (see Figure 15–9). Pin SV_{CC} is used as an output or input. It is an input when the internal SV_{CC} switch is off and the VREF is applied externally. It is an output when the internal SV_{CC} switch is on.

Figure 15–9. ADC12+2 Input Register, Input Enable Register



15.2.7.2 Output Pins

There are two different types of output signals: outputs A0, A1, A2, A3, and output SV_{CC}. Current flows out of one of the analog pins A0, A1, A2, A3 if the current source function is selected. An external resistor between Rext and SV_{CC} determines the amount of current. The SV_{CC} pin outputs a voltage just below AV_{CC} when the SV_{CC} switch is on.

15.2.7.3 Supply Pins

There are four supply pins to split the digital and analog current paths: AV_{CC}, DV_{CC}, A_{GND}, and D_{GND}. Some of the MSP430 family members may have all four supply pins bonded out, while others may have analog and digital V_{CC} and/or GND rails internally connected. Check the specific device's data sheet for configuration.

15.3 ADC12+2 Control Registers

The four ADC12+2 control registers are described in Table 15–1.

Table 15–1. ADC12+2 Control Registers

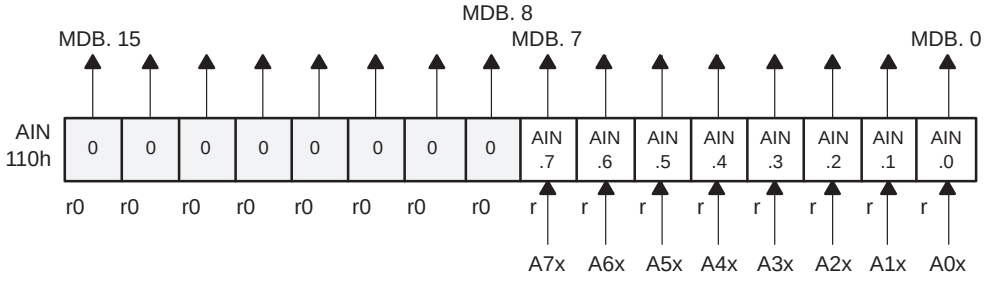
Register	Short Form	Register Type	Address	Initial State
Input	AIN	Read only	0110h	– – –
Input enable	AEN	Read/write	0112h	Reset
ADC control	ACTL	Read/write	0114h	See Figure 15–13
Reserved				0116h
ADC Data	ADAT	Read	0118h	– – –

15.3.1 Input Register AIN

When any of the inputs A0 to A7 are configured as digital inputs, the digital values are read from the AIN register.

Input register AIN is a read-only register connected to the 16-bit MDB; however, only the register low byte is implemented. MDB.0 to MDB.7 correspond to A0 to A7 as shown in Figure 15–10. The register high byte is read as 00h.

Figure 15–10. Input Register AIN

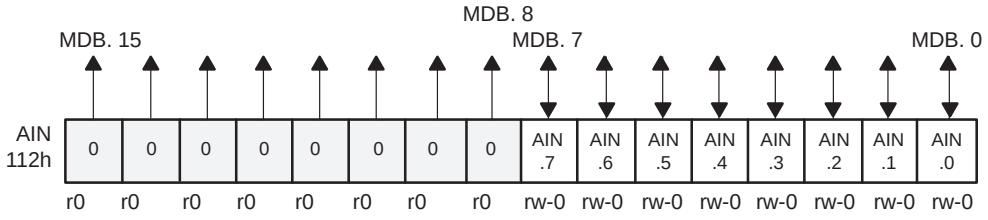


The signal at the corresponding input is logically ANDed with the applicable enable signal (see Figure 15–9). Unselected (disabled) bits are read as 0.

15.3.2 Input Enable Register AEN

Input enable register AEN, shown in Figure 15–11, is a read/write register connected to the 16-bit MDB; however, only the register low byte is implemented. MDB.0 to MDB.7 correspond to A0 to A7. The register high byte is read as 00h.

Figure 15–11. Input Enable Register AEN



The input enable register bits control the definition of the individual bit:

AEN.x = 0: Analog input. The bit read while accessing the AIN register is 0.

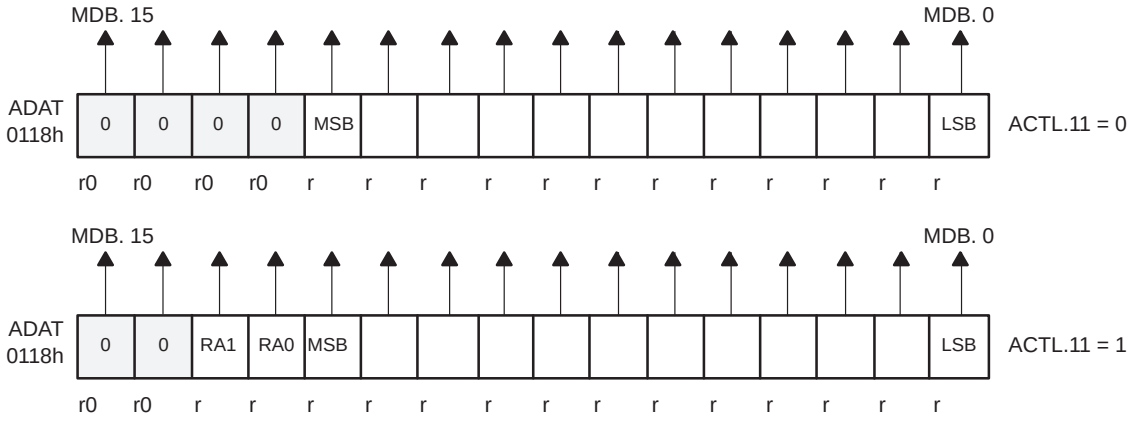
AEN.x = 1: Digital input. The bit read while accessing the AIN register represents the logic level at the applicable pin.

The initial state of all AEN bits is reset.

15.3.3 ADC12+2 Data Register ADAT

The ADC data register (ADAT), shown in Figure 15–12, holds the result of the analog-to-digital conversion. The register data at the end of a conversion are correct until another conversion begins by setting the SOC bit.

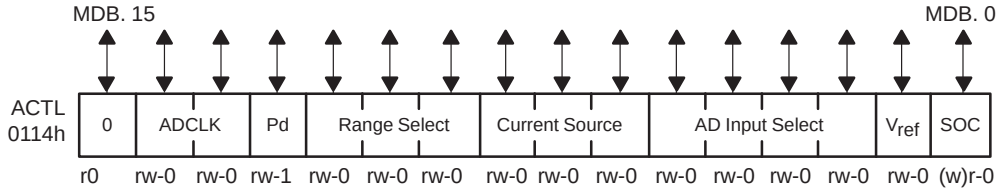
Figure 15–12. ADC12+2 Data Register ADAT



15.3.4 ADC12+2 Control Register ACTL

The ADC12+2 control register (ACTL) is illustrated in Figure 15–13.

Figure 15–13. ADC12+2 Control Register ACTL



Bit 0: Start of conversion
Setting this bit starts the ADC conversion. It is automatically reset.

Bit 1: Source of VREF
ACTL.1 = 0 : Switch SV_{CC} is off. The ADC reference voltage must be supplied from an external source.
ACTL.1 = 1 : Switch SV_{CC} is on. The SV_{CC} pin is connected to V_{CC} internally and configured as an output. The ADC reference voltage must not be supplied from an external source.

Bits 2–5: A/D input selection
These bits select the channel for conversion as described in Table 15–2. Channels should be changed only after completing a conversion. Changing the channel while a conversion is active invalidates the conversion in progress.

Table 15–2.A/D Input Selection

ACTL.5	ACTL.4	ACTL.3	ACTL.2	Channel
0	0	0	0	A0
0	0	0	1	A1
0	0	1	0	A2
0	0	1	1	A3
0	1	0	0	A4
0	1	0	1	A5
0	1	1	0	A6
0	1	1	1	A7
1	X	X	X	NONE

Bits 6–8: A/D current source output selection
These bits select the channel for current source output as described in Table 15–3. Channels should be changed only after completing a conversion. Changing the channel while a conversion is active invalidates the conversion in progress.

Table 15–3. A/D Current Source Selection

ACTL.8	ACTL.7	ACTL.6	Channel
0	0	0	A0
0	0	1	A1
0	1	0	A2
0	1	1	A3
1	X	X	NONE

Bits 9–11: Range selection
These bits select the range for 12-bit mode conversion as described in Table 15–4. They must not be changed after a conversion starts. Any manipulation of these bits during conversion results in incorrect conversion data. Their states are ignored when 12+2-bit mode is selected.

Table 15–4. Range Selection

ACTL.11	ACTL.10	ACTL.9	Range
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	D
1	X	X	Auto

Bit 11: Conversion mode
ACTL.11 = 0 : 12-bit mode selected. The range selection bits ACTL.9 and ACTL.10 must be used for manual range selection.
ACTL.11 = 1 : 12+2-bit mode selected. The automatic range selection is active. The state of the range selection bits ACTL.9 and ACTL.10 is *don't care*.

Bit 12: Power down (Pd)
ACTL.12 = 0: ADC12+2 is powered. Note, the ADC12+2 needs about 6 µs to stabilize after bit Pd is reset.
ACTL.12 = 1: SV_{CC} switch is off.
Comparator is powered down.
Current source is off.

Bit 13, 14: ADCLK
The ADC12+2 clock is selected as described in Table 15–5.

Table 15–5. ADCLK Clock Frequency

ACTL.14	ACTL.13	ADCLK
0	0	MCLK
0	1	MCLK/2
1	0	MCLK/3
1	1	MCLK/4

Bit 15: Reserved

Appendix A

Peripheral File Map

This appendix summarizes the MSP430x3xx peripheral file (PF) and control-bit information into a single location for reference.

Each PF register is presented as a row of boxes containing the control or status bits belonging to the register. The register symbol (e.g. P0IN) and the PF hex address are to the left of each register.

Topic **Page**

A.1 Overview A-2
A.2 Special Function Register of MSP430x3xx Family, Byte Access . . . A-2
A.3 Digital I/O, Byte Access A-3
A.4 LCD Registers, Byte Access A-5
A.5 8-Bit Timer/Counter, Basic Timer, Timer/Port, Byte Access A-6
A.6 FLL Registers, Byte Access A-6
A.7 EPROM Control Register and Crystal Buffer, Byte Access A-7
A.8 USART, UART Mode (Sync=0), Byte Access A-7
A.9 USART SPI Mode (Sync=1), Byte Access A-8
A.10 ADC12+2, Word Access A-9
A.11 Watchdog/Timer, Word Access A-10
A.12 Hardware Multiplier, Word Access A-10
A.13 Timer_A Registers, Word Access A-11

A.1 Overview

Bit accessibility and/or hardware definitions are indicated following each bit symbol:

- rw: Read/write
- r: Read only
- r0: Read as 0
- r1: Read as 1
- w: Write only
- w0: Write as 0
- w1: Write as 1
- (w): No register bit implemented; writing a 1 results in a pulse. The register bit is always read as 0.
- h0: Cleared by hardware
- h1: Set by hardware
- –0,–1: Condition after PUC signal active
- –(0),–(1): Condition after POR signal active

The tables in the following sections describe byte access to each peripheral file according to the previously-described definitions.

A.2 Special Function Register of MSP430x3xx Family, Byte Access

000Fh								
Module enable 2, ME2 0005h							UTXE rw-0	URXE USPIIE rw-0
Module enable 1, ME1 0004h								
Interrupt flag 2, IFG2 0003h	BTIFG rw					ADIFG rw-0	UTXIFG rw-0	URXIFG rw-0
Interrupt flag 1, IFG1 0002h				NMIIFG rw-0	P0IFG.1 rw-0	P0IFG.0 rw-0	OFIFG rw-1	WDTIFG rw-0
Interrupt enable 2, IE2 0001h	BTIE rw-0				TPIE‡ rw-0	ADIE† TPIE† rw-0	UTXIE rw-0	URXIE rw-0
Interrupt enable 1, IE1 0000h					P0IE.1 rw-0	P0IE.0 rw-0	OFIE rw-0	WDTIE rw-0

† ADIE – ADC12+2 interrupt enable (32x devices)
TPIE – Timer/Port interrupt enable (31x devices)
‡ TPIE – Timer/Port interrupt enable (32x, 33x devices)

Note: SFR bits are not implemented on devices without the corresponding peripheral.

A.3 Digital I/O, Byte Access

Bit # -	7	6	5	4	3	2	1	0
Function select, P4SEL 001Fh	P4SEL.7 rw-0	P4SEL.6 rw-0	P4SEL.5 rw-0	P4SEL.4 rw-0	P4SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
Direction register, P4DIR 001Eh	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P4DIR.4 rw-0	P4DIR.3 rw-0	P4DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
Output register, P4OUT 001Dh	P4OUT.7 rw	P4OUT.6 rw	P4OUT.5 rw	P4OUT.4 rw	P4OUT.3 rw	P4OUT.2 rw	P4OUT.1 rw	P4OUT.0 rw
Input register, P4IN 001Ch	P4IN.7 r	P4IN.6 r	P4IN.5 r	P4IN.4 r	P4IN.3 r	P4IN.2 r	P4IN.1 r	P4IN.0 r
Function select, P3SEL 001Bh	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.4 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
Direction register, P3DIR 001Ah	P3DIR.7 rw-0	P3DIR.6 rw-0	P3DIR.5 rw-0	P3DIR.4 rw-0	P3DIR.3 rw-0	P3DIR.2 rw-0	P3DIR.1 rw-0	P3DIR.0 rw-0
Output register, P3OUT 0019h	P3OUT.7 rw	P3OUT.6 rw	P3OUT.5 rw	P3OUT.4 rw	P3OUT.3 rw	P3OUT.2 rw	P3OUT.1 rw	P3OUT.0 rw
Input register, P3IN 0018h	P3IN.7 r	P3IN.6 r	P3IN.5 r	P3IN.4 r	P3IN.3 r	P3IN.2 r	P3IN.1 r	P3IN.0 r
0017h								
0016h								
Interrupt enable, P0IE 0015h	P0IE.7 rw-0	P0IE.6 rw-0	P0IE.5 rw-0	P0IE.4 rw-0	P0IE.3 rw-0	P0IE.2 rw-0	† r0	† r0
Interrupt edge select, P0IES 0014h	P0IES.7 rw	P0IES.6 rw	P0IES.5 rw	P0IES.4 rw	P0IES.3 rw	P0IES.2 rw	P0IES.1 rw	P0IES.0 rw
Interrupt flags, P0IFG 0013h	P0IFG.7 rw-0	P0IFG.6 rw-0	P0IFG.5 rw-0	P0IFG.4 rw-0	P0IFG.3 rw-0	P0IFG.2 rw-0	† r0	† r0
Direction register, P0DIR 0012h	P0DIR.7 rw-0	P0DIR.6 rw-0	P0DIR.5 rw-0	P0DIR.4 rw-0	P0DIR.3 rw-0	P0DIR.2 rw-0	P0DIR.1 rw-0	P0DIR.0 rw-0
Output register, P0OUT 0011h	P0OUT.7 rw	P0OUT.6 rw	P0OUT.5 rw	P0OUT.4 rw	P0OUT.3 rw	P0OUT.2 rw	P0OUT.1 rw	P0OUT.0 rw
Input register, P0IN 0010h	P0IN.7 r	P0IN.6 r	P0IN.5 r	P0IN.4 r	P0IN.3 r	P0IN.2 r	P0IN.1 r	P0IN.0 r

† These interrupt enable bits and flags are included in the SFR frame.

A.3 Digital I/O, Byte Access (Continued)

Bit # -	7	6	5	4	3	2	1	0
002Fh								
Function select, P2SEL 002Eh	P2SEL.7 rw-0	P2SEL.6 rw-0	P2SEL.5 rw-0	P2SEL.4 rw-0	P2SEL.3 rw-0	P2SEL.2 rw-0	P2SEL.1 rw-0	P2SEL.0 rw-0
Interrupt enable, P2IE 002Dh	P2IE.7 rw-0	P2IE.6 rw-0	P2IE.5 rw-0	P2IE.4 rw-0	P2IE.3 rw-0	P2IE.2 rw-0	P2IE.1 rw-0	P2IE.0 rw-0
Interrupt edge select, P2IES 002Ch	P2IES.7 rw	P2IES.6 rw	P2IES.5 rw	P2IES.4 rw	P2IES.3 rw	P2IES.2 rw	P2IES.1 rw	P2IES.0 rw
Interrupt flags, P2IFG 002Bh	P2IFG.7 rw-0	P2IFG.6 rw-0	P2IFG.5 rw-0	P2IFG.4 rw-0	P2IFG.3 rw-0	P2IFG.2 rw-0	P2IFG.1 rw-0	P2IFG.0 rw-0
Direction register, P2DIR 002Ah	P2DIR.7 rw-0	P2DIR.6 rw-0	P2DIR.5 rw-0	P2DIR.4 rw-0	P2DIR.3 rw-0	P2DIR.2 rw-0	P2DIR.1 rw-0	P2DIR.0 rw-0
Output register, P2OUT 0029h	P2OUT.7 rw	P2OUT.6 rw	P2OUT.5 rw	P2OUT.4 rw	P2OUT.3 rw	P2OUT.2 rw	P2OUT.1 rw	P2OUT.0 rw
Input register, P2IN 0028h	P2IN.7 r	P2IN.6 r	P2IN.5 r	P2IN.4 r	P2IN.3 r	P2IN.2 r	P2IN.1 r	P2IN.0 r
0027h								
Function select, P1SEL 0026h	P1SEL.7 rw-0	P1SEL.6 rw-0	P1SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P1SEL.2 rw-0	P1SEL.1 rw-0	P1SEL.0 rw-0
Interrupt enable, P1IE 0025h	P1IE.7 rw-0	P1IE.6 rw-0	P1IE.5 rw-0	P1IE.4 rw-0	P1IE.3 rw-0	P1IE.2 rw-0	P1IE.1 rw-0	P1IE.0 rw-0
Interrupt edge select, P1IES 0024h	P1IES.7 rw	P1IES.6 rw	P1IES.5 rw	P1IES.4 rw	P1IES.3 rw	P1IES.2 rw	P1IES.1 rw	P1IES.0 rw
Interrupt flags, P1IFG 0023h	P1IFG.7 rw-0	P1IFG.6 rw-0	P1IFG.5 rw-0	P1IFG.4 rw-0	P1IFG.3 rw-0	P1IFG.2 rw-0	P1IFG.1 rw-0	P1IFG.0 rw-0
Direction register, P1DIR 0022h	P1DIR.7 rw-0	P1DIR.6 rw-0	P1DIR.5 rw-0	P1DIR.4 rw-0	P1DIR.3 rw-0	P1DIR.2 rw-0	P1DIR.1 rw-0	P1DIR.0 rw-0
Output register, P1OUT 0021h	P1OUT.7 rw	P1OUT.6 rw	P1OUT.5 rw	P1OUT.4 rw	P1OUT.3 rw	P1OUT.2 rw	P1OUT.1 rw	P1OUT.0 rw
Input register, P1IN 0020h	P1IN.7 r	P1IN.6 r	P1IN.5 r	P1IN.4 r	P1IN.3 r	P1IN.2 r	P1IN.1 r	P1IN.0 r

A.4 LCD Registers, Byte Access

Bit # –	7	6	5	4	3	2	1	0
LCD memory 15 003Fh	S29C3 rw	S29C2 rw	S29C1 rw	S29C0 rw	S28C3 rw	S28C2 rw	S28C1 rw	S28C0 rw
LCD memory 14 003Eh	S27C3 rw	S27C2 rw	S27C1 rw	S27C0 rw	S26C3 rw	S26C2 rw	S26C1 rw	S26C0 rw
LCD memory 13 003Dh	S25C3 rw	S25C2 rw	S25C1 rw	S25C0 rw	S24C3 rw	S24C2 rw	S24C1 rw	S24C0 rw
LCD memory 12 003Ch	S23C3 rw	S23C2 rw	S23C1 rw	S23C0 rw	S22C3 rw	S22C2 rw	S22C1 rw	S22C0 rw
LCD memory 11 003Bh	S21C3 rw	S21C2 rw	S21C1 rw	S21C0 rw	S20C3 rw	S20C2 rw	S20C1 rw	S20C0 rw
LCD memory 10 003Ah	S19C3 rw	S19C2 rw	S19C1 rw	S19C0 rw	S18C3 rw	S18C2 rw	S18C1 rw	S18C0 rw
LCD memory 9 0039h	S17C3 rw	S17C2 rw	S17C1 rw	S17C0 rw	S16C3 rw	S16C2 rw	S16C1 rw	S16C0 rw
LCD memory 8 0038h	S15C3 rw	S15C2 rw	S15C1 rw	S15C0 rw	S14C3 rw	S14C2 rw	S14C1 rw	S14C0 rw
LCD memory 7 0037h	S13C3 rw	S13C2 rw	S13C1 rw	S13C0 rw	S12C3 rw	S12C2 rw	S12C1 rw	S12C0 rw
LCD memory 6 0036h	S11C3 rw	S11C2 rw	S11C1 rw	S11C0 rw	S10C3 rw	S10C2 rw	S10C1 rw	S10C0 rw
LCD memory 5 0035h	S9C3 rw	S9C2 rw	S9C1 rw	S9C0 rw	S8C3 rw	S8C2 rw	S8C1 rw	S8C0 rw
LCD memory 4 0034h	S7C3 rw	S7C2 rw	S7C1 rw	S7C0 rw	S6C3 rw	S6C2 rw	S6C1 rw	S6C0 rw
LCD memory 3 0033h	S5C3 rw	S5C2 rw	S5C1 rw	S5C0 rw	S4C3 rw	S4C2 rw	S4C1 rw	S4C0 rw
LCD memory 2 0032h	S3C3 rw	S3C2 rw	S3C1 rw	S3C0 rw	S2C3 rw	S2C2 rw	S2C1 rw	S2C0 rw
LCD memory 1 0031h	S1C3 rw	S1C2 rw	S1C1 rw	S1C0 rw	S0C3 rw	S0C2 rw	S0C1 rw	S0C0 rw
LCD control & mode, LCDC 0030h	LCDM7 rw-0	LCDM6 rw-0	LCDM5 rw-0	LCDM4 rw-0	LCDM3 rw-0	LCDM2 rw-0	LCDM1 rw-0	LCDM0 rw-0

Note: The LCD memory bits are named with the MSP430 convention. The first part of the bit name indicates the corresponding segment line and the second indicates the corresponding common line.
Example for a segment using S4 and Com3: S4C3

A.5 8-Bit Timer/Counter, Basic Timer, Timer/Port, Byte Access

Bit # –	7	6	5	4	3	2	1	0
Timer/Port enable reg., TPE 04Fh	TPSSEL3 rw-0	TPSSEL2 rw-0	TPE.5 rw-0	TPE.4 rw-0	TPE.3 rw-0	TPE.2 rw-0	TPE.1 rw-0	TPE.0 rw-0
Timer/Port data reg., TPD 04Eh	B16 rw-0	CPON rw-0	TPD.5 rw-0	TPD.4 rw-0	TPD.3 rw-0	TPD.2 rw-0	TPD.1 rw-0	TPD.0 rw-0
Timer/Port counter1, TPCNT2 04Dh	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Timer/Port counter1, TPCNT1 04Ch	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Timer/Port control reg., TPCTL 04Bh	TPSSEL1 rw-0	TPSSEL0 rw-0	ENB rw-0	ENA rw-0	EN1 r-0	RC2FG rw-0	RC1FG rw-0	EN1FG rw-0
Counter data, 8-Bit Basic Timer, BTCNT2 0047h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
Counter data, 8-Bit Basic Timer, BTCNT1 0046h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
0045h								
Counter data, 8-Bit Timer/Counter, TCDAT 0044h	TCDAT.7 rw	TCDAT.6 rw	TCDAT.5 rw	TCDAT.4 rw	TCDAT.3 rw	TCDAT.2 rw	TCDAT.1 rw	TCDAT.0 rw
Preload register, 8-Bit Timer/Counter, TCPLD 0043h	TCPLD.7 rw	TCPLD.6 rw	TCPLD.5 rw	TCPLD.4 rw	TCPLD.3 rw	TCPLD.2 rw	TCPLD.1 rw	TCPLD.0 rw
Control register, 8-Bit Timer/Counter, TCCTL 0042h	SSEL1 rw-0	SSEL0 rw-0	ISCTL rw-0	TXEN rw-0	ENCNT rw-0	RXACT rw-0	TXD rw-0	RXD r(-1)
0041h								
Basic Timer, BTCTL 0040h	SSEL rw	Hold rw	DIV rw	FRFQ1 rw	FRFQ0 rw	IP2 rw	IP1 rw	IP0 rw

A.6 FLL Registers, Byte Access

Bit # –	7	6	5	4	3	2	1	0
Frequency control, SCFQCTL 0052h	M rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw-1	2^0 rw-1
Frequency integrator, SCFI1 0051h	2^9 rw-0	2^8 rw-0	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0
Frequency integrator, SCFI0 0050h	0 r	0 r	0 r	FN_4 rw-0	FN_3 rw-0	FN_2 rw-0	2^1 rw-0	2^0 rw-0

A.7 EPROM Control Register and Crystal Buffer, Byte Access

Bit # –	7	6	5	4	3	2	1	0
EPROM control register† EPCTL 0054h	r-0	r-0	r-0	r-0	r-0	r-0	VPPS rw-0	EXE rw-0
Crystal buffer control register‡ CBCTL 0053h						CBSEL1 w-(0)	CBSEL0 w-(0)	CBE w-(0)

† NonEPROM devices may use this register for other control purposes.
‡ Devices without XBUF may use this register for other control purposes.

A.8 USART, UART Mode (Sync=0), Byte Access

Bit # –	7	6	5	4	3	2	1	0
USART Transmit buffer UTXBUF 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART Receive buffer URXBUF 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART Baud rate UBR1 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART Baud rate UBR0 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART Modulation control UMCTL 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART Receive control URCTL 072h	FE rw-0	PE rw-0	OE rw-0	BRK rw-0	URXEIE rw-0	URXWIE rw-0	RXWake rw-0	RXERR rw-0
USART Transmit control UTCTL 071h	Unused rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	URXSE rw-0	TXWAKE rw-0	Unused rw-0	TXEPT rw-1
USART USART control UCTL 070h	PENA rw-0	PEV rw-0	SP rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

A.9 USART, SPI Mode (Sync=1), Byte Access

Bit # –	7	6	5	4	3	2	1	0
USART Transmit buffer UTXBUF 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART Receive buffer URXBUF 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART Baud rate UBR1 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART Baud rate UBR0 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART Modulation control UMCTL 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART Receive control URCTL 072h	FE rw-0	Undef. rw-0	OE rw-0	Undef. rw-0	Unused rw-0	Unused rw-0	Undef. rw-0	Undef. rw-0
USART Transmit control UTCTL 071h	CKPH rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	Unused rw-0	Unused rw-0	STC rw-0	TXEPT rw-1
USART USART control UCTL 070h	Unused rw-0	Unused rw-0	Unused rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

A.10 ADC12+2, Word Access

Bit # –	15	14	13	12	11	10	9	8
11Fh								
ADC12+2, Data register ADAT 118h	r0	r0	R1† r0	R0† r0	2^{11} r	2^{10} r	2^{9} r	2^{8} r
Reserved 116h								
ADC12+2, Control register ACTL 114h	ACTL.15 r0	ACTL.14 rw-0	ACTL.13 rw-0	ACTL.12 rw-1	ACTL.11 rw-0	ACTL.10 rw-0	ACTL.9 rw-0	ACTL.8 rw-0
ADC12+2, Input enable register AEN 112h	r0	r0	r0	r0	r0	r0	r0	r0
ADC12+2, Input data register AIN 110h	r0	r0	r0	r0	r0	r0	r0	r0

† The bits ADAT.12 and ADAT.13 are read as 0 when ACTL.11 = 0; otherwise, signals R0 and R1 are read.

Bit # –	7	6	5	4	3	2	1	0
11Eh								
ADC12+2, Data register ADAT 118h	2^{7} r	2^{6} r	2^{5} r	2^{4} r	2^{3} r	2^{2} r	2^{1} r	2^{0} r
Reserved 116h								
ADC12+2, Control register ACTL 114h	ACTL.7 rw-0	ACTL.6 rw-0	ACTL.5 rw-0	ACTL.4 rw-0	ACTL.3 rw-0	ACTL.2 rw-0	ACTL.1 rw-0	ACTL.0 (w)r0
ADC12+2, Input enable register AEN 112h	AEN.7 rw-0	AEN.6 rw-0	AEN.5 rw-0	AEN.4 rw-0	AEN.3 rw-0	AEN.2 rw-0	AEN.1 rw-0	AEN.0 rw-0
ADC12+2, Input data register AIN 110h	AIN.7 r	AIN.6 r	AIN.5 r	AIN.4 r	AIN.3 r	AIN.2 r	AIN.1 r	AIN.0 r

A.11 Watchdog/Timer, Word Access

Bit # –	15							8
Watchdog Timer, Control register WDTCTL 120h	<———			Read as 069h			———	>
	<———			Written as 05Ah			———	>

Bit # –	7	6	5	4	3	2	1	0
Watchdog Timer, Control register WDTCTL 120h	HOLD rw-0	NMIES rw-0	NMI rw-0	TMSEL rw-0	CNTCL (w),r0	SSEL rw-0	IS1 rw-0	IS0 rw-0

A.12 Hardware Multiplier, Word Access

Bit # –	15	14	13	12	11	10	9	8
Sum extend, SumExt 013Eh	† r	† r	† r	† r	† r	† r	† r	† r
Result-high word ResHI 013Ch	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Result-low word ResLO 013Ah	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Second operand OP2 0138h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
MPYS+ACC MACS 0136h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
MPY+ACC MAC 0134h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply signed MPYS 0132h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply unsigned MPY 0130h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw

Bit # –	7	6	5	4	3	2	1	0
Sum extend, SumExt 013Eh	† r	† r	† r	† r	† r	† r	† r	† r
Result-high word ResHI 013Ch	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Result-low word ResLO 013Ah	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Second operand OP2 0138h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
MPYS+ACC MACS 0136h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
MPY+ACC MAC 0134h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply signed MPYS 0132h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply unsigned MPY 0130h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw

† The Sum Extend register SumExt holds a 16×16-bit multiplication (MPYS) sign result, or the overflow of the multiply and accumulate (MAC) operation, or the sign of the signed multiply and accumulate (MACS) operation. Overflow and underflow of the MACS operation must be handled by software.

A.13 Timer_A Registers, Word Access

Bit # –	15	14	13	12	11	10	9	8
017Eh								
017Ch								
Cap/com register CCR4† 017Ah	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR3† 0178h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR2 0176h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR1 0174h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR0 0172h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Timer_A register TAR 0170h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
016Eh								
016Ch								
Cap/com control CCTL4†, 016Ah	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	SCCI4 rw-(0)	Unused r0	CAP4 rw-(0)
Cap/com control CCTL3†, 0168h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	SCCI3 rw-(0)	Unused r0	CAP3 rw-(0)
Cap/com control CCTL2, 0166h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	SCCI2 rw-(0)	Unused r0	CAP2 rw-(0)
Cap/com control CCTL1, 0164h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	SCCI1 rw-(0)	Unused r0	CAP1 rw-(0)
Cap/com control CCTL0, 0162h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	SCCI0 rw-(0)	Unused r0	CAP0 rw-(0)
Timer_A control TACTL 0160h	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	SSEL2 rw-(0)	SSEL1 rw-(0)	SSEL0 rw-(0)

† Registers are reserved on devices with Timer_A3.

A.13 Timer_A Registers, Word Access (Continued)

Bit # –	7	6	5	4	3	2	1	0
017Eh								
017Ch								
Cap/com register CCR4† 017Ah	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR3† 0178h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR2 0176h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR1 0174h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR0 0172h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Timer_A register TAR 0170h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
016Eh								
016Ch								
Cap/com control CCTL4†, 016Ah	OutMod42 rw-(0)	OutMod41 rw-(0)	OutMod40 rw-(0)	CCIE4 rw-(0)	CCI4 r	OUT4 rw-(0)	COV4 rw-(0)	CCIFG4 rw-(0)
Cap/com control CCTL3†, 0168h	OutMod32 rw-(0)	OutMod31 rw-(0)	OutMod30 rw-(0)	CCIE3 rw-(0)	CCI3 r	OUT3 rw-(0)	COV3 rw-(0)	CCIFG3 rw-(0)
Cap/com control CCTL2, 0166h	OutMod22 rw-(0)	OutMod21 rw-(0)	OutMod20 rw-(0)	CCIE2 rw-(0)	CCI2 r	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
Cap/com control CCTL1, 0164h	OutMod12 rw-(0)	OutMod11 rw-(0)	OutMod10 rw-(0)	CCIE1 rw-(0)	CCI1 r	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
Cap/com control CCTL0, 0162h	OutMod02 rw-(0)	OutMod01 rw-(0)	OutMod00 rw-(0)	CCIE0 rw-(0)	CCI0 r	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
Timer_A control TACTL 0160h	ID1 rw-(0)	ID0 rw-(0)	MC1 rw-(0)	MC0 rw-(0)	Unused rw-(0)	CLR rw-(0)	TAIE rw-(0)	TAIFG rw-(0)

† Registers are reserved on devices with Timer_A3.

Bit # –	15	14	13	12	11	10	9	8
Timer_A interrupt vector TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0

Bit # –	7	6	5	4	3	2	1	0
Timer_A interrupt vector TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	TAIV r-(0)	r-(0)	r-(0)	0 r0

TAIV Vector, Timer_A5 (five capture/compare blocks integrated)
- 0: No interrupt pending
- 2: CCIFG1 flag set, interrupt flag of capture/compare block 1
- 4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
- 6: CCIFG3 flag set, interrupt flag of capture/compare block 3 (CCIFG1=CCIFG2=0)
- 8: CCIFG3 flag set, interrupt flag of capture/compare block 3 (CCIFG1=CCIFG2=CCIFG3=0)
- 10: TAIFG flag set, interrupt flag of Timer_A register/counter (CCIFG1=CCIFG2=CCIFG3=CCIFG4=0)

TAIV Vector, Timer_A3 (three capture/compare blocks integrated)
- 0: No interrupt pending
- 2: CCIFG1 flag set, interrupt flag of capture/compare block 1
- 4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
- 6: Reserved
- 8: Reserved
- 10: TAIFG flag set, interrupt flag of Timer_A register/counter (CCIFG1=CCIFG2=CCIFG3=CCIFG4=0)

Appendix B

Instruction Set Description

The MSP430 core CPU architecture evolved from a reduced instruction set with highly-transparent instruction formats. Using these formats, core instructions are implemented into the hardware. Emulated instructions are also supported by the assembler. Emulated instructions use the core instructions with the built-in constant generators CG1 and CG2 and/or the program counter (PC). The core and emulated instructions are described in detail in this section. The emulated instruction mnemonics are listed with examples.

Program memory words used by an instruction vary from one to three words, depending on the combination of addressing modes.

Topic **Page**

B.1 Instruction Set Overview B-2

B.2 Instruction Set Description B-8

B.1 Instruction Set Overview

The following list gives an overview of the instruction set.

				Status Bits			
				V	N	Z	C
*	ADC[.W];ADC.B	dst	dst + C –> dst	*	*	*	*
	ADD[.W];ADD.B	src,dst	src + dst –> dst	*	*	*	*
	ADDC[.W];ADDC.B	src,dst	src + dst + C –> dst	*	*	*	*
	AND[.W];AND.B	src,dst	src .and. dst –> dst	0	*	*	*
	BIC[.W];BIC.B	src,dst	.not.src .and. dst –> dst	–	–	–	–
	BIS[.W];BIS.B	src,dst	src .or. dst –> dst	–	–	–	–
	BIT[.W];BIT.B	src,dst	src .and. dst	0	*	*	*
*	BR	dst	Branch to	–	–	–	–
	CALL	dst	PC+2 –> stack, dst –> PC	–	–	–	–
*	CLR[.W];CLR.B	dst	Clear destination	–	–	–	–
*	CLRC	Clear carry bit		–	–	–	0
*	CLRN	Clear negative bit		–	0	–	–
*	CLRZ	Clear zero bit		–	–	0	–
	CMP[.W];CMP.B	src,dst	dst – src	*	*	*	*
*	DADC[.W];DADC.B	dst	dst + C –> dst (decimal)	*	*	*	*
	DADD[.W];DADD.B	src,dst	src + dst + C –> dst (decimal)	*	*	*	*
*	DEC[.W];DEC.B	dst	dst – 1 –> dst	*	*	*	*
*	DECD[.W];DECD.B	dst	dst – 2 –> dst	*	*	*	*
*	DINT	Disable interrupt		–	–	–	–
*	EINT	Enable interrupt		–	–	–	–
*	INC[.W];INC.B	dst	Increment destination, dst +1 –> dst	*	*	*	*
*	INCD[.W];INCD.B	dst	Double-Increment destination, dst+2–>dst	*	*	*	*
*	INV[.W];INV.B	dst	Invert destination	*	*	*	*
	JC/JHS	Label	Jump to Label if Carry-bit is set	–	–	–	–
	JEQ/JZ	Label	Jump to Label if Zero-bit is set	–	–	–	–
	JGE	Label	Jump to Label if (N .XOR. V) = 0	–	–	–	–
	JL	Label	Jump to Label if (N .XOR. V) = 1	–	–	–	–
	JMP	Label	Jump to Label unconditionally	–	–	–	–
	JN	Label	Jump to Label if Negative-bit is set	–	–	–	–
	JNC/JLO	Label	Jump to Label if Carry-bit is reset	–	–	–	–
	JNE/JNZ	Label	Jump to Label if Zero-bit is reset	–	–	–	–

				Status Bits			
				V	N	Z	C
	MOV[.W];MOV.B	src,dst	src –> dst	–	–	–	–
*	NOP		No operation	–	–	–	–
*	POP[.W];POP.B	dst	Item from stack, SP+2 → SP	–	–	–	–
	PUSH[.W];PUSH.B	src	SP – 2 → SP, src → @SP	–	–	–	–
	RETI		Return from interrupt TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SZP	*	*	*	*
*	RET		Return from subroutine TOS → PC, SP + 2 → SP	–	–	–	–
*	RLA[.W];RLA.B	dst	Rotate left arithmetically	*	*	*	*
*	RLC[.W];RLC.B	dst	Rotate left through carry	*	*	*	*
	RRA[.W];RRA.B	dst	MSB → MSB → ....LSB → C	0	*	*	*
	RRC[.W];RRC.B	dst	C → MSB → .....LSB → C	*	*	*	*
*	SBC[.W];SBC.B	dst	Subtract carry from destination	*	*	*	*
*	SETC		Set carry bit	–	–	–	1
*	SETN		Set negative bit	–	1	–	–
*	SETZ		Set zero bit	–	–	1	–
	SUB[.W];SUB.B	src,dst	dst + .not.src + 1 → dst	*	*	*	*
	SUBC[.W];SUBC.B	src,dst	dst + .not.src + C → dst	*	*	*	*
	SWPB	dst	swap bytes	–	–	–	–
	SXT	dst	Bit7 → Bit8 ..... Bit15	0	*	*	*
*	TST[.W];TST.B	dst	Test destination	0	*	*	1
	XOR[.W];XOR.B	src,dst	src .xor. dst → dst	*	*	*	*

Note: Asterisked Instructions

Asterisked (*) instructions are emulated. They are replaced with core instructions by the assembler.

B.1.1 Instruction Formats

The following sections describe the instruction formats.

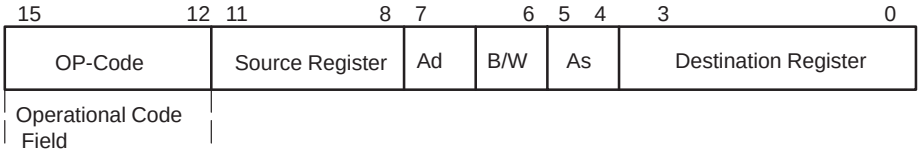
B.1.1.1 Double-Operand Instructions (Core Instructions)

The instruction format using double operands, as shown in Figure B–1, consists of four main fields to form a 16-bit code:

- operational code field, four bits [op-code]
- source field, six bits [source register + As]
- byte operation identifier, one bit [BW]
- destination field, five bits [dest. register + Ad]

The source field is composed of two addressing bits and a four-bit register number (0....15). The destination field is composed of one addressing bit and a four-bit register number (0....15). The byte identifier B/W indicates whether the instruction is executed as a byte (B/W = 1) or as a word instruction (B/W = 0).

Figure B–1. Double-Operand Instructions



				Status Bits			
				V	N	Z	C
ADD[.W];	ADD.B	src,dst	src + dst –> dst	*	*	*	*
ADDC[.W];	ADDC.B	src,dst	src + dst + C –> dst	*	*	*	*
AND[.W];	AND.B	src,dst	src .and. dst –> dst	0	*	*	*
BIC[.W];	BIC.B	src,dst	.not.src .and. dst –> dst	–	–	–	–
BIS[.W];	BIS.B	src,dst	src .or. dst –> dst	–	–	–	–
BIT[.W];	BIT.B	src,dst	src .and. dst	0	*	*	*
CMP[.W];	CMP.B	src,dst	dst – src	*	*	*	*
DADD[.W];	DADD.B	src,dst	src + dst + C –> dst (dec)	*	*	*	*
MOV[.W];	MOV.B	src,dst	src –> dst	–	–	–	–
SUB[.W];	SUB.B	src,dst	dst + .not.src + 1 –> dst	*	*	*	*
SUBC[.W];	SUBC.B	src,dst	dst + .not.src + C –> dst	*	*	*	*
XOR[.W];	XOR.B	src,dst	src .xor. dst –> dst	*	*	*	*

Note: Operations Using the Status Register (SR) for Destination

All operations using Status Register SR for destination overwrite the SR contents with the operation result; as described in that operation, the status bits are not affected.

Example: ADD #3,SR ; Operation: (SR) + 3 —> SR

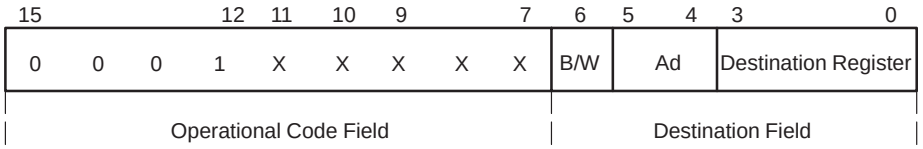
B.1.1.2 Single Operand Instructions (Core Instructions)

The instruction format using a single operand, as shown in Figure B–2, consists of two main fields to form a 16-bit code:

- operational code field, nine bits with four MSBs equal to 1h
- byte operation identifier, one bit [B/W]
- destination field, six bits [destination register + Ad]

The destination field is composed of two addressing bits and the four-bit register number (0....15). The destination field bit position is the same as that of the two operand instructions. The byte identifier (B/W) indicates whether the instruction is executed as a byte (B/W = 1) or as a word (B/W = 0).

Figure B–2. Single-Operand Instructions



				Status Bits			
				V	N	Z	C
RRA[.W];	RRA.B	dst	MSB → MSB ...LSB → C	0	*	*	*
RRC[.W];	RRC.B	dst	C → MSB .....LSB → C	*	*	*	*
PUSH[.W];	PUSH.B	dst	SP – 2 → SP, src → @SP	–	–	–	–
SWPB		dst	swap bytes	–	–	–	–
CALL		dst	PC→2 + @SP, dst → PC	–	–	–	–
RETI		dst	TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT		dst	Bit 7 → Bit 8 ..... Bit 15	0	*	*	*

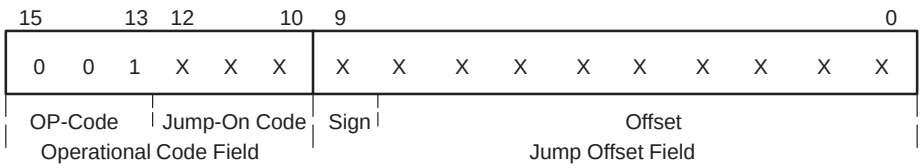
B.1.2 Conditional and Unconditional Jumps (Core Instructions)

The instruction format for conditional and unconditional jumps, as shown in Figure B–3, consists of two main fields to form a 16-bit code:

- operational code (op-code) field, six bits
- jump offset field, ten bits

The operational-code field is composed of the op-code (three bits), and three bits according to the following conditions.

Figure B–3. Conditional and Unconditional Jump Instructions



Conditional jumps jump to addresses in the range of –511 to +512 words relative to the current address. The assembler computes the signed offsets and inserts them into the op-code.

JC/JHS	Label	Jump to label if carry bit is set
JEQ/JZ	Label	Jump to label if zero bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally
JN	Label	Jump to label if negative bit is set
JNC/JLO	Label	Jump to label if carry bit is reset
JNE/JNZ	Label	Jump to label if zero bit is reset

Note: Conditional and Unconditional Jumps

Conditional and unconditional jumps do not affect the status bits.

A jump that is taken alters the PC with the offset:

PCnew = PCold + 2 + 2*offset

A jump that is not taken continues the program with the ascending instruction.

B.1.3 Emulated Instructions

The following instructions can be emulated with the reduced instruction set without additional code words. The assembler accepts the emulated instruction mnemonic, and inserts the applicable core instruction op-code.

The following list describes the emulated instruction short form.

Mnemonic		Description	Status Bits V	N	Z	C	Emulation	
Arithmetical instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#0FFFFh,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data instructions (common use)								
CLR[.W]		Clear destination	–	–	–	–	MOV	#0,dst
CLR.B		Clear destination	–	–	–	–	MOV.B	#0,dst
CLRC		Clear carry bit	–	–	–	0	BIC	#1,SR
CLRN		Clear negative bit	–	0	–	–	BIC	#4,SR
CLRZ		Clear zero bit	–	–	0	–	BIC	#2,SR
POP	dst	Item from stack	–	–	–	–	MOV	@SP+,dst
SETC		Set carry bit	–	–	–	1	BIS	#1,SR
SETN		Set negative bit	–	1	–	–	BIS	#4,SR
SETZ		Set zero bit	–	–	1	–	BIS	#2,SR
TST[.W]	dst	Test destination	0	*	*	1	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	1	CMP.B	#0,dst
Program flow instructions								
BR	dst	Branch to	–	–	–	–	MOV	dst,PC
DINT		Disable interrupt	–	–	–	–	BIC	#8,SR
EINT		Enable interrupt	–	–	–	–	BIS	#8,SR
NOP		No operation	–	–	–	–	MOV	#0h,#0h
RET		Return from subroutine	–	–	–	–	MOV	@SP+,PC

B.2 Instruction Set Description

This section catalogues and describes all core and emulated instructions in alphabetical order. Some examples serve as explanations and others as application hints.

The suffix .W or no suffix in the instruction mnemonic results in a word operation.

The suffix .B at the instruction mnemonic results in a byte operation.

ADC[.W] Add carry to destination
ADC.B Add carry to destination

Syntax

```
ADC     dst   or   ADC.W   dst
ADC.B   dst
```

Operation dst + C –> dst

Emulation

```
ADDC    #0,dst
ADDC.B  #0,dst
```

Description The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
Set if dst was incremented from 0FFh to 00, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)     ; Add LSDs
ADC     2(R12)          ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)     ; Add LSDs
ADC.B   1(R12)          ; Add carry to MSD
```

ADD[.W] Add source to destination
ADD.B Add source to destination

Syntax

```
ADD     src,dst     or      ADD.W   src,dst
ADD.B   src,dst
```

Operation src + dst –> dst

Description The source operand is added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the result, cleared if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD     #10,R5
JC      TONI        ; Carry occurred
.....               ; No carry
```

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD.B   #10,R5      ; Add 10 to Lowbyte of R5
JC      TONI        ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
.....               ; No carry
```

ADDC[.W] Add source and carry to destination
ADDC.B Add source and carry to destination

Syntax

```
ADDC     src,dst    or    ADDC.W    src,dst
ADDC.B   src,dst
```

Operation src + dst + C –> dst

Description The source operand and the carry bit (C) are added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset otherwise
V**:** Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 32-bit counter pointed to by R13 is added to a 32-bit counter, eleven words (20/2 + 2/2) above the pointer in R13.

```
ADD      @R13+,20(R13)   ; ADD LSDs with no carry in
ADDC     @R13+,20(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

Example The 24-bit counter pointed to by R13 is added to a 24-bit counter, eleven words above the pointer in R13.

```
ADD.B    @R13+,10(R13)   ; ADD LSDs with no carry in
ADDC.B   @R13+,10(R13)   ; ADD medium Bits with carry
ADDC.B   @R13+,10(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

AND[.W] Source AND destination
AND.B Source AND destination

Syntax

```
AND     src,dst     or   AND.W src,dst
AND.B   src,dst
```

Operation src .AND. dst –> dst

Description The source operand and the destination operand are logically ANDed. The result is placed into the destination.

Status Bits

N: Set if result MSB is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI.

```
MOV     #0AA55h,R5      ; Load mask into register R5
AND     R5,TOM          ; mask word addressed by TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
;
;
;       or
;
;
AND     #0AA55h,TOM
JZ      TONI
```

Example The bits of mask #0A5h are logically ANDed with the low byte TOM. If the result is zero, a branch is taken to label TONI.

```
AND.B   #0A5h,TOM       ; mask Lowbyte TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
```

BIC[.W] Clear bits in destination
BIC.B Clear bits in destination

Syntax

```
BIC     src,dst    or  BIC.W  src,dst
BIC.B   src,dst
```

Operation .NOT.src .AND. dst –> dst

Description The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The six MSBs of the RAM word LEO are cleared.

```
BIC     #0FC00h,LEO          ; Clear 6 MSBs in MEM(LEO)
```

Example The five MSBs of the RAM byte LEO are cleared.

```
BIC.B   #0F8h,LEO            ; Clear 5 MSBs in Ram location LEO
```

Example The port pins P0 and P1 are cleared.

```
P0OUT   .equ   011h;         Definition of port address
P0_0    .equ                 01h
P0_1    .equ                 02h

BIC.B   #P0_0+P0_1,&P0OUT    ;Set P0.0 and P0.1 to low
```

BIS[.W] Set bits in destination
BIS.B Set bits in destination

Syntax

```
BIS     src,dst    or  BIS.W    src,dst
BIS.B   src,dst
```

Operation src .OR. dst –> dst

Description The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The six LSBs of the RAM word TOM are set.

```
BIS     #003Fh,TOM; set the six LSBs in RAM location TOM
```

Example Start an A/D- conversion

```
ASOC    .equ    1       ; Start of conversion bit
ACTL    .equ    114h    ; ADC control register

BIS     #ASOC,&ACTL     ; Start A/D-conversion
```

Example The three MSBs of RAM byte TOM are set.

```
BIS.B   #0E0h,TOM       ; set the 3 MSBs in RAM location TOM
```

Example Port pins P0 and P1 are set to high.

```
P0OUT   .equ    011h
P0      .equ    01h
P1      .equ    02h

BIS.B   #P0+P1,&P0OUT
```

BIT[.W] Test bits in destination
BIT.B Test bits in destination

Syntax BIT src,dst or BIT.W src,dst

Operation src .AND. dst

Description The source and destination operands are logically ANDed. The result affects only the status bits. The source and destination operands are not affected.

Status Bits
N: Set if MSB of result is set, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example If bit 9 of R8 is set, a branch is taken to label TOM.

```
BIT     #0200h,R8       ; bit 9 of R8 set?
JNZ     TOM             ; Yes, branch to TOM
...                     ; No, proceed
```

Example Determine which A/D channel is configured by the MUX.

```
ACTL    .equ            114h            ; ADC control register
BIT     #4,&ACTL                        ; Is channel 0 selected?
jnz     END                             ; Yes, branch to END
```

Example If bit 3 of R8 is set, a branch is taken to label TOM.

```
BIT.B   #8,R8
JC      TOM
```

Example A serial communication receive bit (RCV) is tested. Because the carry bit is equal to the state of the tested bit while using the BIT instruction to test a single bit, the carry bit is used by the subsequent instruction; the read information is shifted into register RECBUF.

```
;
; Serial communication with LSB is shifted first:
                                ; xxxx  xxxx  xxxx  xxxx
BIT.B   #RCV,RCCTL      ; Bit info into carry
RRC     RECBUF          ; Carry –> MSB of RECBUF
                        ; cxxx  xxxx
.....                   ; repeat previous two instructions
.....                   ; 8 times
                        ; cccc  cccc
                        ;  ^     ^
                        ; MSB   LSB
; Serial communication with MSB is shifted first:
BIT.B   #RCV,RCCTL      ; Bit info into carry
RLC.B   RECBUF          ; Carry –> LSB of RECBUF
                        ; xxxx  xxxc
.....                   ; repeat previous two instructions
.....                   ; 8 times
                        ; cccc  cccc
                        ; |     LSB
                        ; MSB
```

* **BR, BRANCH** Branch to ..... destination

Syntax BR dst

Operation dst –> PC

Emulation MOV dst,PC

Description An unconditional branch is taken to an address anywhere in the 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
BR      #EXEC   ;Branch to label EXEC or direct branch (e.g. #0A4h)
                ; Core instruction MOV @PC+,PC

BR      EXEC    ; Branch to the address contained in EXEC
                ; Core instruction MOV X(PC),PC
                ; Indirect address

BR      &EXEC   ; Branch to the address contained in absolute
                ; address EXEC
                ; Core instruction MOV X(0),PC
                ; Indirect address

BR      R5      ; Branch to the address contained in R5
                ; Core instruction MOV R5,PC
                ; Indirect R5

BR      @R5     ; Branch to the address contained in the word
                ; pointed to by R5.
                ; Core instruction MOV @R5,PC
                ; Indirect, indirect R5

BR      @R5+    ; Branch to the address contained in the word pointed
                ; to by R5 and increment pointer in R5 afterwards.
                ; The next time—S/W flow uses R5 pointer—it can
                ; alter program execution due to access to
                ; next address in a table pointed to by R5
                ; Core instruction MOV @R5,PC
                ; Indirect, indirect R5 with autoincrement

BR      X(R5)   ; Branch to the address contained in the address
                ; pointed to by R5 + X (e.g. table with address
                ; starting at X). X can be an address or a label
                ; Core instruction MOV X(R5),PC
                ; Indirect, indirect R5 + X
```

CALL Subroutine

Syntax CALL dst

Operation

dst	–> tmp	dst is evaluated and stored
SP – 2	–> SP	
PC	–> @SP	PC updated to TOS
tmp	–> PC	dst saved to PC

Description A subroutine call is made to an address anywhere in the 64K address space. All addressing modes can be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
CALL    #EXEC   ; Call on label EXEC or immediate address (e.g. #0A4h)
                ; SP–2 → SP, PC+2 → @SP, @PC+ → PC

CALL    EXEC    ; Call on the address contained in EXEC
                ; SP–2 → SP, PC+2 → @SP, X(PC) → PC
                ; Indirect address

CALL    &EXEC   ; Call on the address contained in absolute address
                ; EXEC
                ; SP–2 → SP, PC+2 → @SP, X(PC) → PC
                ; Indirect address

CALL    R5      ; Call on the address contained in R5
                ; SP–2 → SP, PC+2 → @SP, R5 → PC
                ; Indirect R5

CALL    @R5     ; Call on the address contained in the word
                ; pointed to by R5
                ; SP–2 → SP, PC+2 → @SP, @R5 → PC
                ; Indirect, indirect R5

CALL    @R5+    ; Call on the address contained in the word
                ; pointed to by R5 and increment pointer in R5.
                ;  The next time—S/W flow uses R5 pointer—
                ; it can alter the program execution due to
                ;  access to next address in a table pointed to by R5
                ; SP–2 → SP, PC+2 → @SP, @R5 → PC
                ; Indirect, indirect R5 with autoincrement

CALL    X(R5)   ; Call on the address contained in the address pointed
                ; to by R5 + X (e.g. table with address starting at X)
                ; X can be an address or a label
                ; SP–2 → SP, PC+2 → @SP, X(R5) → PC
                ; Indirect indirect R5 + X
```

*** CLR[.W]** Clear destination
* **CLR.B** Clear destination

Syntax
```
CLR     dst     or   CLR.W dst
CLR.B   dst
```

Operation 0 –> dst

Emulation
```
MOV     #0,dst
MOV.B   #0,dst
```

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Example RAM word TONI is cleared.

```
CLR     TONI        ; 0 –> TONI
```

Example Register R5 is cleared.

```
CLR     R5
```

Example RAM byte TONI is cleared.

```
CLR.B   TONI        ; 0 –> TONI
```

*** CLRC** Clear carry bit

Syntax CLRC

Operation 0 –> C

Emulation BIC #1,SR

Description The carry bit (C) is cleared. The clear carry instruction is a word instruction.

Status Bits
N: Not affected
Z: Not affected
C: Cleared
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
CLRC                ; C=0: defines start
DADD  @R13,0(R12)   ; add 16-bit counter to low word of 32-bit counter
DADC  2(R12)        ; add carry to high word of 32-bit counter
```

*** CLRN** Clear negative bit

Syntax CLRN

Operation 0 → N
or
(.NOT.src .AND. dst –> dst)

Emulation BIC #4,SR

Description The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits
N: Reset to 0
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The Negative bit in the status register is cleared. This avoids special treatment with negative numbers of the subroutine called.

```
        CLRN
        CALL    SUBR
        ......
        ......
SUBR    JN      SUBRET  ; If input is negative: do nothing and return
        ......
        ......
        ......
SUBRET  RET
```

*** CLRZ** Clear zero bit

Syntax CLRZ

Operation 0 → Z
or
(.NOT.src .AND. dst –> dst)

Emulation BIC #2,SR

Description The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits
N: Not affected
Z: Reset to 0
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The zero bit in the status register is cleared.

```
CLRZ
```

CMP[.W] Compare source and destination
CMP.B Compare source and destination

Syntax

```
CMP     src,dst    or    CMP.W    src,dst
CMP.B   src,dst
```

Operation

dst + .NOT.src + 1
or
(dst – src)

Description The source operand is subtracted from the destination operand. This is accomplished by adding the 1s complement of the source operand plus 1. The two operands are not affected and the result is not stored; only the status bits are affected.

Status Bits

N: Set if result is negative, reset if positive (src >= dst)
Z: Set if result is zero, reset otherwise (src = dst)
C: Set if there is a carry from the MSB of the result, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 and R6 are compared. If they are equal, the program continues at the label EQUAL.

```
CMP     R5,R6      ; R5 = R6?
JEQ     EQUAL      ; YES, JUMP
```

Example Two RAM blocks are compared. If they are not equal, the program branches to the label ERROR.

```
        MOV   #NUM,R5            ; number of words to be compared
L$1     CMP   &BLOCK1,&BLOCK2    ; Are Words equal?
        JNZ   ERROR              ; No, branch to ERROR
        DEC   R5                 ; Are all words compared?
        JNZ   L$1                ; No, another compare
```

Example The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL.

```
        CMP.B  EDE,TONI          ; MEM(EDE) = MEM(TONI)?
        JEQ    EQUAL             ; YES, JUMP
```

Example Check two keys connected to port pins P0 and P1. If key1 is pressed, the program branches to label MENU1; if key2 is pressed, the program branches to MENU2.

```
P0IN    .EQU   010h
KEY1    .EQU   01h
KEY2    .EQU   02h

        CMP.B  #KEY1,&P0IN
        JEQ    MENU1
        CMP.B  #KEY2,&P0IN
        JEQ    MENU2
```

*** DADC[.W]** Add carry decimally to destination
* **DADC.B** Add carry decimally to destination

Syntax

```
DADC      dst   or   DADC.W   src,dst
DADC.B    dst
```

Operation dst + C –> dst (decimally)

Emulation

```
DADD      #0,dst
DADD.B    #0,dst
```

Description The carry bit (C) is added decimally to the destination.

Status Bits

N: Set if MSB is 1
Z: Set if dst is 0, reset otherwise
C: Set if destination increments from 9999 to 0000, reset otherwise
Set if destination increments from 99 to 00, reset otherwise
V: Undefined

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD      R5,0(R8)  ; Add LSDs + C
DADC      2(R8)     ; Add carry to MSD
```

Example The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD.B    R5,0(R8)  ; Add LSDs + C
DADC      1(R8)     ; Add carry to MSDs
```

DADD[.W] Source and carry added decimally to destination
DADD.B Source and carry added decimally to destination

Syntax

```
DADD    src,dst    or  DADD.W  src,dst
DADD.B  src,dst
```

Operation src + dst + C –> dst (decimally)

Description The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry bit (C) are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers.

Status Bits

N: Set if the MSB is 1, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if the result is greater than 9999
Set if the result is greater than 99
V: Undefined

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The eight-digit BCD number contained in R5 and R6 is added decimally to an eight-digit BCD number contained in R3 and R4 (R6 and R4 contain the MSDs).

```
CLRC            ; CLEAR CARRY
DADD    R5,R3   ; add LSDs
DADD    R6,R4   ; add MSDs with carry
JC      OVERFLOW ; If carry occurs go to error handling routine
```

Example The two-digit decimal counter in the RAM byte CNT is incremented by one.

```
CLRC            ; clear Carry
DADD.B  #1,CNT  ; increment decimal counter
```

or

```
SETC
DADD.B  #0,CNT  ; ≡ DADC.B  CNT
```

*** DEC[.W]** Decrement destination
*** DEC.B** Decrement destination

Syntax

DEC dst or DEC.W dst
DEC.B dst

Operation

dst – 1 –> dst

Emulation

SUB #1,dst

Emulation

SUB.B #1,dst

Description

The destination operand is decremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 1, reset otherwise
C: Reset if dst contained 0, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08000h, otherwise reset.
Set if initial value of destination was 080h, otherwise reset.

Mode Bits

OscOff**,** CPUOff, and GIE are not affected.

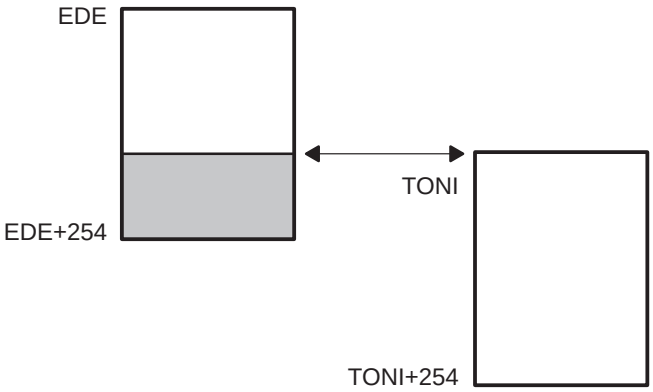
Example R10 is decremented by 1

```
                DEC     R10     ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to memory location starting with
;TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE
; to EDE+0FEh
;
                MOV     #EDE,R6
                MOV     #255,R10
L$1             MOV.B   @R6+,TONI–EDE–1(R6)
                DEC     R10
                JNZ     L$1

; Do not transfer tables using the routine above with the overlap shown in Figure B–4.
```

Figure B–4. Decrement Overlap



Example Memory byte at address LEO is decremented by one.

```
                DEC.B   LEO     ; Decrement MEM(LEO)

; Move a block of 255 bytes from memory location starting with EDE to memory location starting with
; TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE
; to EDE+0FEh
;
                MOV     #EDE,R6
                MOV.B   #255,LEO
L$1             MOV.B   @R6+,TONI–EDE–1(R6)
                DEC.B   LEO
                JNZ     L$1
```

*** DECD[.W]** Double-decrement destination
*** DECD.B** Double-decrement destination

Syntax
DECD dst or DECD.W dst
DECD.B dst

Operation
dst – 2 –> dst

Emulation
SUB #2,dst
Emulation
SUB.B #2,dst

Description
The destination operand is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 2, reset otherwise
C: Reset if dst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08001 or 08000h, otherwise reset.
Set if initial value of destination was 081 or 080h, otherwise reset.

Mode Bits
OscOff, CPUOff, and GIE are not affected.

Example
R10 is decremented by 2.

```
        DECD    R10         ; Decrement R10 by two
```

```
; Move a block of 255 words from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;
        MOV     #EDE,R6
        MOV     #510,R10
L$1     MOV     @R6+,TONI–EDE–2(R6)
        DECD    R10
        JNZ     L$1
```

Example
Memory at location LEO is decremented by two.

```
        DECD.B  LEO         ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two.

```
        DECD.B  STATUS
```

*** DINT** Disable (general) interrupts

Syntax DINT

Operation 0 → GIE
or
(0FFF7h .AND. SR → SR / .NOT.src .AND. dst –> dst)

Emulation BIC #8,SR

Description All interrupts are disabled.
The constant 08h is inverted and logically ANDed with the status register (SR). The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits GIE is reset. OscOff and CPUOff are not affected.

Example The general interrupt enable (GIE) bit in the status register is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                    ; All interrupt events using the GIE bit are disabled
NOP
MOV     COUNTHI,R5      ; Copy counter
MOV     COUNTLO,R6
EINT                    ; All interrupt events using the GIE bit are enabled
```

Note: Disable Interrupt

If any code sequence needs to be protected from interruption, the DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or should be followed by an NOP.

*** EINT** Enable (general) interrupts

Syntax EINT

Operation 1 → GIE
or
(0008h .OR. SR –> SR / .NOT.src .OR. dst –> dst)

Emulation BIS #8,SR

Description All interrupts are enabled.
The constant #08h and the status register SR are logically ORed. The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits GIE is set. OscOff and CPUOff are not affected.

Example The general interrupt enable (GIE) bit in the status register is set.

```
; Interrupt routine of port P0.2 to P0.7
; The interrupt level is the lowest in the system
; P0IN is the address of the register where all port bits are read. P0IFG is the address of
; the register where all interrupt events are latched.
;
                PUSH.B  &P0IN
                BIC.B   @SP,&P0IFG  ; Reset only accepted flags
                EINT                ; Preset port 0 interrupt flags stored on stack
                                    ; other interrupts are allowed
                BIT     #Mask,@SP
                JEQ     MaskOK      ; Flags are present identically to mask: jump
                ......
MaskOK          BIC     #Mask,@SP
                ......
                INCD    SP          ; Housekeeping: inverse to PUSH instruction
                                    ; at the start of interrupt subroutine. Corrects
                                    ; the stack pointer.
                RETI
```

Note: Enable Interrupt

The instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enable.

*** INC[.W]** Increment destination
*** INC.B** Increment destination

Syntax

```
INC     dst    or   INC.W  dst
INC.B   dst
```

Operation dst + 1 –> dst

Emulation

```
ADD     #1,dst
```

Description The destination operand is incremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The item on the top of a software stack (not the system stack) for byte data is removed.

```
SSP     .EQU    R4
;
        INC     SSP     ; Remove TOSS (top of SW stack) by increment
                        ; Do not use INC.B since SSP is a word register
```

Example The status byte of a process STATUS is incremented. When it is equal to 11, a branch to OVFL is taken.

```
        INC.B   STATUS
        CMP.B   #11,STATUS
        JEQ     OVFL
```

*** INCD[.W]** Double-increment destination
*** INCD.B** Double-increment destination

Syntax
```
INCD    dst   or  INCD.W   dst
INCD.B  dst
```

Operation dst + 2 –> dst

Emulation
```
ADD     #2,dst
```
Emulation
```
ADD.B   #2,dst
```

Example The destination operand is incremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFEh, reset otherwise
Set if dst contained 0FEh, reset otherwise
C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
Set if dst contained 0FEh or 0FFh, reset otherwise
V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The item on the top of the stack (TOS) is removed without using a register.

```
.......
PUSH    R5      ; R5 is the result of a calculation, which is stored
                ; in the system stack
INCD    SP      ; Remove TOS by double-increment from stack
                ; Do not use INCD.B, SP is a word-aligned
                ; register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B  0(SP)   ; Byte on TOS is increment by two
```

*** INV[.W]** Invert destination
*** INV.B** Invert destination

Syntax
```
INV     dst
INV.B   dst
```

Operation .NOT.dst –> dst

Emulation
```
XOR     #0FFFFh,dst
```
Emulation
```
XOR.B   #0FFh,dst
```

Description The destination operand is inverted. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if initial destination operand was negative, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example Content of R5 is negated (twos complement).
```
MOV     #00Aeh,R5   ;                       R5 = 000AEh
INV     R5          ; Invert R5,            R5 = 0FF51h
INC     R5          ; R5 is now negated,    R5 = 0FF52h
```

Example Content of memory byte LEO is negated.
```
MOV.B   #0AEh,LEO   ;                       MEM(LEO) = 0AEh
INV.B   LEO         ; Invert LEO,           MEM(LEO) = 051h
INC.B   LEO         ; MEM(LEO) is negated,  MEM(LEO) = 052h
```

JC Jump if carry set
JHS Jump if higher or same

Syntax

```
JC      label
JHS     label
```

Operation

If C = 1: PC + 2 × offset –> PC
If C = 0: execute following instruction

Description

The status register carry bit (C) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits

Status bits are not affected.

Example

The P0IN.1 signal is used to define or control the program flow.

```
BIT     #10h,&P0IN      ; State of signal –> Carry
JC      PROGA           ; If carry=1 then execute program routine A
.....                   ; Carry=0, execute program here
```

Example

R5 is compared to 15. If the content is higher or the same, branch to LABEL.

```
CMP     #15,R5
JHS     LABEL           ; Jump is taken if R5 ≥ 15
.....                   ; Continue here if R5 < 15
```

JEQ, JZ Jump if equal, jump if zero

Syntax JEQ label, JZ label

Operation If Z = 1: PC + 2 × offset –> PC
If Z = 0: execute following instruction

Description The status register zero bit (Z) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is not set, the instruction following the jump is executed.

Status Bits Status bits are not affected.

Example Jump to address TONI if R7 contains zero.

```
TST     R7          ; Test R7
JZ      TONI        ; if zero: JUMP
```

Example Jump to address LEO if R6 is equal to the table contents.

```
CMP     R6,Table(R5)  ; Compare content of R6 with content of
                      ; MEM (table address + content of R5)
JEQ     LEO           ; Jump if both data are equal
.....                 ; No, data are not equal, continue here
```

Example Branch to LABEL if R5 is 0.

```
TST     R5
JZ      LABEL
.....
```

JGE Jump if greater or equal

Syntax JGE label

Operation If (N .XOR. V) = 0 then jump to label: PC + 2 × offset –> PC
If (N .XOR. V) = 1 then execute the following instruction

Description The status register negative bit (N) and overflow bit (V) are tested. If both N and V are set or reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If only one is set, the instruction following the jump is executed.

This allows comparison of signed integers.

Status Bits Status bits are not affected.

Example When the content of R6 is greater or equal to the memory pointed to by R7, the program continues at label EDE.

```
CMP     @R7,R6      ; R6 ≥ (R7)?, compare on signed numbers
JGE     EDE         ; Yes, R6 ≥ (R7)
.....               ; No, proceed
.....
.....
```

JL Jump if less

Syntax JL label

Operation If (N .XOR. V) = 1 then jump to label: PC + 2 × offset –> PC
If (N .XOR. V) = 0 then execute following instruction

Description The status register negative bit (N) and overflow bit (V) are tested. If only one is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If both N and V are set or reset, the instruction following the jump is executed.

This allows comparison of signed integers.

Status Bits Status bits are not affected.

Example When the content of R6 is less than the memory pointed to by R7, the program continues at label EDE.

```
CMP     @R7,R6      ; R6 < (R7)?,  compare on signed numbers
JL      EDE         ; Yes, R6 < (R7)
.....               ; No, proceed
.....
.....
```

JMP	Jump unconditionally
Syntax	MP label
Operation	PC + 2 × offset –> PC
Description	The 10-bit signed offset contained in the instruction LSBs is added to the program counter.
Status Bits	Status bits are not affected.
Hint:	This one-word instruction replaces the BRANCH instruction in the range of –511 to +512 words relative to the current program counter.

JN Jump if negative

Syntax JN label

Operation if N = 1: PC + 2 × offset –> PC
if N = 0: execute following instruction

Description The negative bit (N) of the status register is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If N is reset, the next instruction following the jump is executed.

Status Bits Status bits are not affected.

Example The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.

```
        SUB     R5,COUNT    ; COUNT – R5 –> COUNT
        JN      L$1         ; If negative continue with COUNT=0 at PC=L$1
        ......              ; Continue with COUNT≥0
        ......
        ......
        ......
L$1     CLR     COUNT
        ......
        ......
        ......
```

JNC Jump if carry not set
JLO Jump if lower

Syntax

```
JNC     label
JNC     label
```

Operation

if C = 0: PC + 2 × offset –> PC
if C = 1: execute following instruction

Description

The status register carry bit (C) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits

Status bits are not affected.

Example

The result in R6 is added in BUFFER. If an overflow occurs, an error handling routine at address ERROR is used.

```
        ADD     R6,BUFFER   ; BUFFER + R6 –> BUFFER
        JNC     CONT        ; No carry, jump to CONT
ERROR   .....               ; Error handler start
        .....
        .....
        .....
CONT    .....               ; Continue with normal program flow
        .....
        .....
```

Example

Branch to STL2 if byte STATUS contains 1 or 0.

```
        CMP.B   #2,STATUS
        JLO     STL2        ; STATUS < 2
        .....               ; STATUS ≥ 2, continue here
```

JNE, JNZ Jump if not equal, jump if not zero

Syntax JNE label, JNZ label

Operation If Z = 0: PC + 2 × offset –> PC
If Z = 1: execute following instruction

Description The status register zero bit (Z) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is set, the next instruction following the jump is executed.

Status Bits Status bits are not affected.

Example Jump to address TONI if R7 and R8 have different contents.

```
CMP     R7,R8     ; COMPARE R7 WITH R8
JNE     TONI      ; if different: jump
.....             ; if equal, continue
```

MOV[.W] Move source to destination
MOV.B Move source to destination

Syntax

MOV src,dst or MOV.W src,dst
MOV.B src,dst

Operation

src –> dst

Description

The source operand is moved to the destination.
The source operand is not affected. The previous contents of the destination are lost.

Status Bits

Status bits are not affected.

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

The contents of table EDE (word data) are copied to table TOM. The length of the tables must be 020h locations.

```
        MOV   #EDE,R10              ; Prepare pointer
        MOV   #020h,R9              ; Prepare counter
Loop    MOV   @R10+,TOM–EDE–2(R10)  ; Use pointer in R10 for both tables
        DEC   R9                    ; Decrement counter
        JNZ   Loop                  ; Counter ≠ 0, continue copying
        ......                      ; Copying completed
        ......
        ......
```

Example

The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations

```
        MOV   #EDE,R10              ; Prepare pointer
        MOV   #020h,R9              ; Prepare counter
Loop    MOV.B @R10+,TOM–EDE–1(R10)  ; Use pointer in R10 for
                                    ; both tables
        DEC   R9                    ; Decrement counter
        JNZ   Loop                  ; Counter ≠ 0, continue
                                    ; copying
        ......                      ; Copying completed
        ......
        ......
```

*** NOP** No operation

Syntax NOP

Operation None

Emulation MOV #0,#0

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected.

The NOP instruction is mainly used for two purposes:

- To hold one, two or three memory words
- To adjust software timing

Note: Emulating No-Operation Instruction

Other instructions can emulate no-operation instruction using different numbers of cycles and code words.

Examples:

```
MOV     0(R4),0(R4)     ; 6 cycles, 3 words
MOV     @R4,0(R4)       ; 5 cycles, 2 words
BIC     #0,EDE(R4)      ; 4 cycles, 2 words
JMP     $+2             ; 2 cycles, 1 word
BIC     #0,R5           ; 1 cycle, 1 word
```

*** POP[.W]** Pop word from stack to destination
*** POP.B** Pop byte from stack to destination

Syntax

```
POP     dst
POP.B   dst
```

Operation

```
@SP   –> dst
SP + 2 –> SP
```

Emulation

```
MOV     @SP+,dst    or    MOV.W    @SP+,dst
```

Emulation

```
MOV.B   @SP+,dst
```

Description The stack location pointed to by the stack pointer (TOS) is moved to the destination. The stack pointer is incremented by two afterwards.

Status Bits Status bits are not affected.

Example The contents of R7 and the status register are restored from the stack.

```
POP     R7      ; Restore R7
POP     SR      ; Restore status register
```

Example The contents of RAM byte LEO is restored from the stack.

```
POP.B   LEO     ; The low byte of the stack is moved to LEO.
```

Example The contents of R7 is restored from the stack.

```
POP.B   R7      ; The low byte of the stack is moved to R7,
                ; the high byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the status register are restored from the stack.

```
POP.B   0(R7)   ; The low byte of the stack is moved to the
                ; the byte which is pointed to by R7
                : Example:  R7 = 203h
                ;           Mem(R7) = low byte of system stack
                : Example:  R7 = 20Ah
                ;           Mem(R7) = low byte of system stack
POP     SR
```

Note: The System Stack Pointer

The system stack pointer (SP) is always incremented by two, independent of the byte suffix.

PUSH[.W] Push word onto stack
PUSH.B Push byte onto stack

Syntax

```
PUSH    src    or    PUSH.W    src
PUSH.B  src
```

Operation

SP – 2 → SP
src → @SP

Description The stack pointer is decremented by two, then the source operand is moved to the RAM word addressed by the stack pointer (TOS).

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The contents of the status register and R8 are saved on the stack.

```
PUSH    SR      ; save status register
PUSH    R8      ; save R8
```

Example The contents of the peripheral TCDAT is saved on the stack.

```
PUSH.B  &TCDAT  ; save data from 8-bit peripheral module,
                ; address TCDAT, onto stack
```

Note: The System Stack Pointer

The system stack pointer (SP) is always decremented by two, independent of the byte suffix.

*** RET**	Return from subroutine
Syntax	RET
Operation	@SP→ PC SP + 2 → SP
Emulation	MOV @SP+,PC
Description	The return address pushed onto the stack by a CALL instruction is moved to the program counter. The program continues at the code address following the subroutine call.
Status Bits	Status bits are not affected.

RETI Return from interrupt

Syntax RETI

Operation

TOS → SR
SP + 2 → SP
TOS → PC
SP + 2 → SP

Description The status register is restored to the value at the beginning of the interrupt service routine by replacing the present SR contents with the TOS contents. The stack pointer (SP) is incremented by two.

The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restoration is performed by replacing the present PC contents with the TOS memory contents. The stack pointer (SP) is incremented.

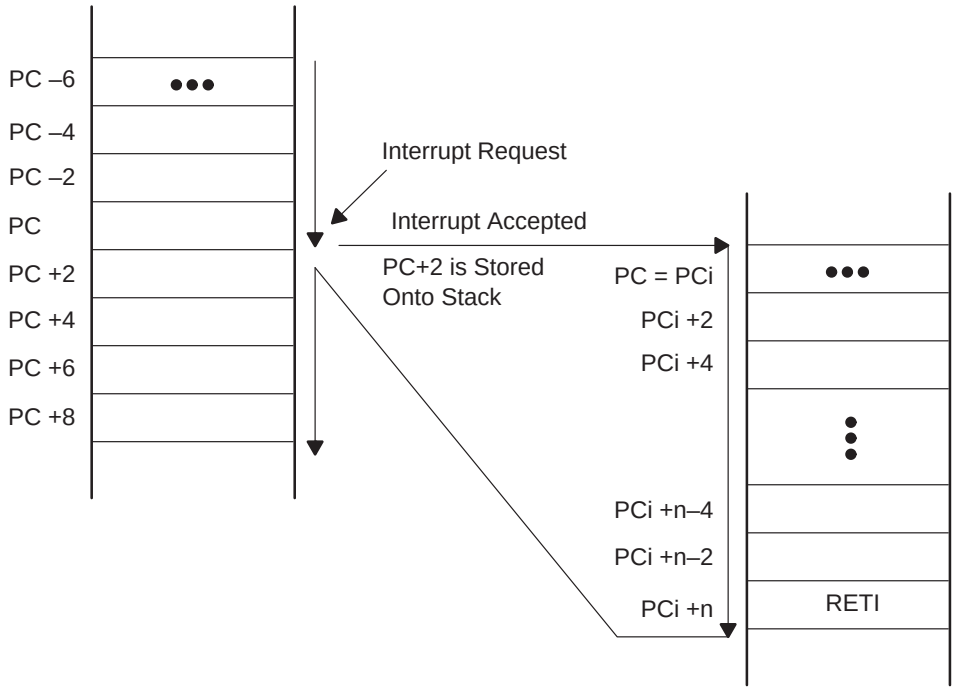
Status Bits

N: restored from system stack
Z: restored from system stack
C: restored from system stack
V: restored from system stack

Mode Bits OscOff, CPUOff, and GIE are restored from system stack.

Example Figure B–5 illustrates the main program interrupt.

Figure B–5. Main Program Interrupt



*** RLA[.W]** Rotate left arithmetically
*** RLA.B** Rotate left arithmetically

Syntax

RLA dst or RLA.W dst
RLA.B dst

Operation

C <– MSB <– MSB–1 .... LSB+1 <– LSB <– 0

Emulation

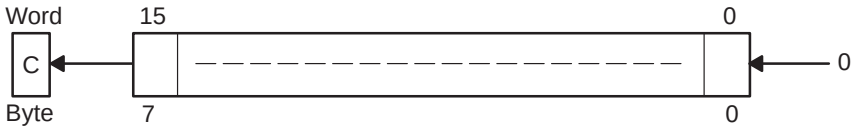
ADD dst,dst
ADD.B dst,dst

Description

The destination operand is shifted left one position as shown in Figure B–6. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.

An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed: the result has changed sign.

Figure B–6. Destination Operand—Arithmetic Shift Left



An overflow occurs if dst ≥ 040h and dst < 0C0h before the operation is performed: the result has changed sign.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs:
the initial value is 04000h ≤ dst < 0C000h; otherwise it is reset
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 0C0h; otherwise it is reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

R7 is multiplied by 4.

```
RLA     R7      ; Shift left R7 (× 2) – emulated by  ADD  R7,R7
RLA     R7      ; Shift left R7 (× 4) – emulated by  ADD  R7,R7
```

Example

The low byte of R7 is multiplied by 4.

```
RLA.B   R7      ; Shift left low byte of R7 (× 2) – emulated by
                ; ADD.B  R7,R7
RLA.B   R7      ; Shift left low byte of R7 (× 4) – emulated by
                ; ADD.B  R7,R7
```

Note: RLA Substitution

The assembler does not recognize the instruction:

RLA @R5+ nor RLA.B @R5+.

It must be substituted by:

ADD @R5+,–2(R5) or ADD.B @R5+,–1(R5).

*** RLC[.W]** Rotate left through carry
*** RLC.B** Rotate left through carry

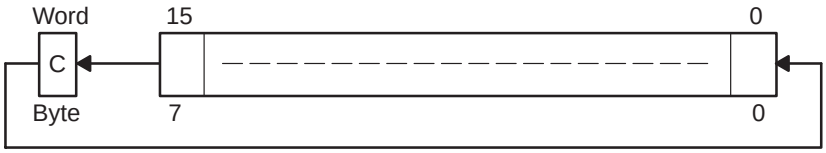
Syntax
RLC dst or RLC.W dst
RLC.B dst

Operation
C <– MSB <– MSB–1 .... LSB+1 <– LSB <– C

Emulation
ADDC dst,dst

Description
The destination operand is shifted left one position as shown in Figure B–7. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

Figure B–7. Destination Operand—Carry Left Shift



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if arithmetic overflow occurs, reset otherwise
Set if 03FFFh < $dst_{initial}$ < 0C000h, reset otherwise
Set if 03Fh < $dst_{initial}$ < 0C0h, reset otherwise

Mode Bits
OscOff, CPUOff, and GIE are not affected.

Example
R5 is shifted left one position.

```
RLC     R5          ; (R5 x 2) + C –> R5
```

Example
The input P0IN.1 information is shifted into the LSB of R5.

```
BIT.B   #2,&P0IN    ; Information –> Carry
RLC     R5          ; Carry=P0in.1 –> LSB of R5
```

Example
The MEM(LEO) content is shifted left one position.

```
RLC.B   LEO         ; Mem(LEO) x 2 + C –> Mem(LEO)
```

Example
The input P0IN.1 information is to be shifted into the LSB of R5.

```
BIT.B   #2,&P0IN    ; Information –> Carry
RLC.B   R5          ; Carry = P0in.1 –> LSB of R5
                    ; High byte of R5 is reset
```

Note: RLC and RLC.B Emulation

The assembler does not recognize the instruction:

```
RLC     @R5+.
```

It must be substituted by:

```
ADDC    @R5+,–2(R5).
```

RRA[.W] Rotate right arithmetically
RRA.B Rotate right arithmetically

Syntax

RRA dst or RRA.W dst
RRA.B dst

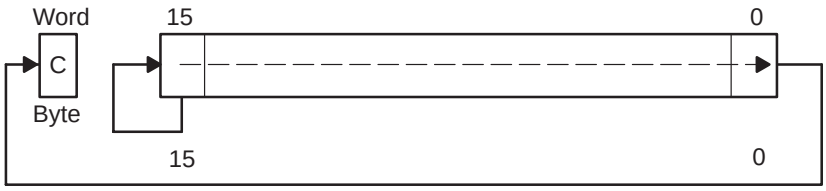
Operation

MSB –> MSB, MSB –> MSB–1, ... LSB+1 –> LSB, LSB –> C

Description

The destination operand is shifted right one position as shown in Figure B–8. The MSB is shifted into the MSB, the MSB is shifted into the MSB–1, and the LSB+1 is shifted into the LSB.

Figure B–8. Destination Operand—Arithmetic Right Shift



Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
                RRA     R5          ; R5/2 –> R5

;               The value in R5 is multiplied by 0.75 (0.5 + 0.25).
;
                PUSH    R5          ; hold R5 temporarily using stack
                RRA     R5          ; R5 × 0.5  –>  R5
                ADD     @SP+,R5     ; R5 × 0.5 + R5 = 1.5 × R5  –> R5
                RRA     R5          ; (1.5 × R5) × 0.5 = 0.75 × R5  –> R5
                ......
                ......
; OR
;
                RRA     R5          ; R5 × 0.5  –>  R5
                PUSH    R5          ; R5 × 0.5  –>  TOS
                RRA     @SP         ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5  –> TOS
                ADD     @SP+,R5     ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5  –> R5
                ......
```

Example The low byte of R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
                RRA.B   R5          ; R5/2 –> R5: operation is on low byte only
                                    ; High byte of R5 is reset

;               The value in R5 (low byte only) is multiplied by 0.75 (0.5 + 0.25).
;
                PUSH.B  R5          ; hold low byte of R5 temporarily using stack
                RRA.B   R5          ; R5 × 0.5  –>  R5
                ADD.B   @SP+,R5     ; R5 × 0.5 + R5 = 1.5 × R5  –> R5
                RRA.B   R5          ; (1.5 × R5) × 0.5 = 0.75 × R5  –> R5
                ......
; OR
;
                RRA.B   R5          ; R5 × 0.5  –>  R5
                PUSH.B  R5          ; R5 × 0.5  –>  TOS
                RRA.B   @SP         ;TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5  –> TOS
                ADD.B   @SP+,R5     ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5  –> R5
                ......
```

RRC[.W] Rotate right through carry
RRC.B Rotate right through carry

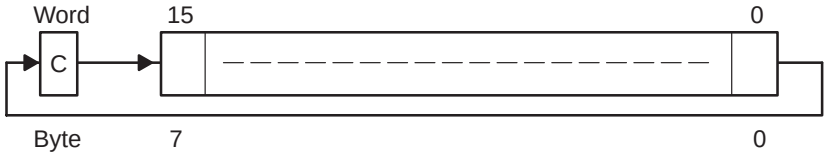
Syntax

```
RRC     dst     or      RRC.W   dst
RRC     dst
```

Operation C –> MSB –> MSB–1 .... LSB+1 –> LSB –> C

Description The destination operand is shifted right one position as shown in Figure B–6. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

Figure B–9. Destination Operand—Carry Right Shift



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Set if initial destination is positive and initial carry is set, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRC     R5          ; R5/2 + 8000h –> R5
```

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRC.B   R5          ; R5/2 + 80h –> R5; low byte of R5 is used
```

*** SBC[.W]** Subtract (borrow*) from destination
*** SBC.B** Subtract (borrow*) from destination

Syntax

```
SBC     dst     or      SBC.W   dst
SBC.B   dst
```

Operation

dst + 0FFFFh + C –> dst
dst + 0FFh + C –> dst

Emulation

```
SUBC    #0,dst
SUBC.B  #0,dst
```

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Reset if dst was decremented from 0000 to 0FFFFh, set otherwise
Reset if dst was decremented from 00 to 0FFh, set otherwise
V: Set if initially C = 0 and dst = 08000h
Set if initially C = 0 and dst = 080h

Mode Bits OscOff**,** CPUOff, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)     ; Subtract LSDs
SBC     2(R12)          ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)     ; Subtract LSDs
SBC.B   1(R12)          ; Subtract carry from MSD
```

Note: Borrow Is Treated as a .NOT.

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

*** SETC** Set carry bit

Syntax SETC

Operation 1 –> C

Emulation BIS #1,SR

Description The carry bit (C) is set.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example Emulation of the decimal subtraction:
Subtract R5 from R6 decimally
Assume that R5 = 3987 and R6 = 4137

```
DSUB    ADD     #6666h,R5       ; Move content R5 from 0–9 to 6–0Fh
                                ; R5 = 03987 + 6666 = 09FEDh
        INV     R5              ; Invert this (result back to 0–9)
                                ; R5 = .NOT. R5 = 06012h
        SETC                    ; Prepare carry = 1
        DADD    R5,R6           ; Emulate subtraction by addition of:
                                ; (10000 – R5 – 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 4137 + 06012 + 1 = 1 0150 = 0150
```

*** SETN**	Set negative bit
Syntax	SETN
Operation	1 –> N
Emulation	BIS #4,SR
Description	The negative bit (N) is set.
Status Bits	N: Set Z: Not affected C: Not affected V: Not affected
Mode Bits	OscOff, CPUOff, and GIE are not affected.

*** SETZ**	Set zero bit
Syntax	SETZ
Operation	1 –> Z
Emulation	BIS #2,SR
Description	The zero bit (Z) is set.
Status Bits	N: Not affected Z: Set C: Not affected V: Not affected
Mode Bits	OscOff, CPUOff, and GIE are not affected.

SUB[.W] Subtract source from destination
SUB.B Subtract source from destination

Syntax

SUB src,dst or SUB.W src,dst
SUB.B src,dst

Operation

dst + .NOT.src + 1 –> dst
or
[(dst – src –> dst)]

Description

The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the constant 1. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset otherwise.
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

See example at the SBC instruction.

Example

See example at the SBC.B instruction.

Note: Borrow Is Treated as a .NOT.

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

SUBC[.W]SBB[.W] Subtract source and borrow/.NOT. carry from destination
SUBC.B,SBB.B Subtract source and borrow/.NOT. carry from destination

Syntax

SUBC	src,dst	or	SUBC.W	src,dst	or
SBB	src,dst	or	SBB.W	src,dst	
SUBC.B	src,dst	or	SBB.B	src,dst	

Operation

dst + .NOT.src + C –> dst
or
(dst – src – 1 + C –> dst)

Description

The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the carry bit (C). The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive.
Z: Set if result is zero, reset otherwise.
C: Set if there is a carry from the MSB of the result, reset otherwise.
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, reset otherwise.

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

Two floating point mantissas (24 bits) are subtracted.
LSBs are in R13 and R10, MSBs are in R12 and R9.

```
SUB.W    R13,R10   ; 16-bit part, LSBs
SUBC.B   R12,R9    ;  8-bit part, MSBs
```

Example

The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).

```
SUB.B    @R13+,R10      ; Subtract LSDs without carry
SUBC.B   @R13,R11       ; Subtract MSDs with carry
...                     ; resulting from the LSDs
```

Note: Borrow Is Treated as a .NOT. Carry

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

SWPB Swap bytes

Syntax SWPB dst

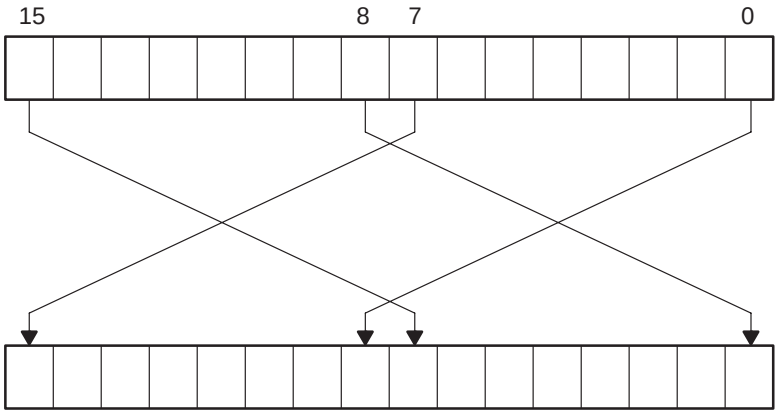
Operation Bits 15 to 8 <–> bits 7 to 0

Description The destination operand high and low bytes are exchanged as shown in Figure B–10.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Figure B–10. Destination Operand Byte Swap



Example

```
MOV     #040BFh,R7      ; 0100000010111111 –> R7
SWPB    R7              ; 1011111101000000 in R7
```

Example The value in R5 is multiplied by 256. The result is stored in R5,R4.

```
SWPB    R5              ;
MOV     R5,R4           ;Copy the swapped value to R4
BIC     #0FF00h,R5      ;Correct the result
BIC     #00FFh,R4       ;Correct the result
```

SXT Extend Sign

Syntax SXT dst

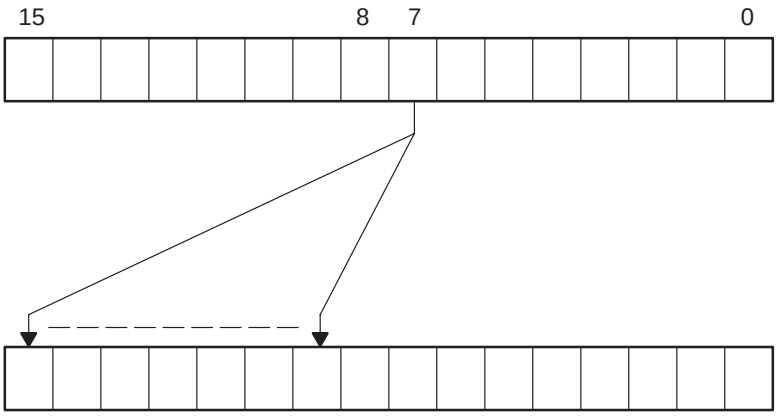
Operation Bit 7 –> Bit 8 ..... Bit 15

Description The sign of the low byte is extended into the high byte as shown in Figure B–11.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Figure B–11. Destination Operand Sign Extension



Example R7 is loaded with the Timer/Counter value. The operation of the sign-extend instruction expands bit 8 to bit 15 with the value of bit 7.
R7 is then added to R6.

```
MOV.B   &TCDAT,R7   ; TCDAT = 080h:     . . . .  . . . . 1000 0000
SXT     R7          ; R7 = 0FF80h:      1111 1111 1000 0000
ADD     R7,R6       ; add value of EDE to 16-bit ACCU
```

*** TST[.W]** Test destination
*** TST.B** Test destination

Syntax

```
TST     dst    or   TST.W  dst
TST.B   dst
```

Operation

dst + 0FFFFh + 1
dst + 0FFh + 1

Emulation

```
CMP     #0,dst
CMP.B   #0,dst
```

Description The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.

Status Bits

N: Set if destination is negative, reset if positive
Z: Set if destination contains zero, reset otherwise
C: Set
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
        TST     R7      ; Test R7
        JN      R7NEG   ; R7 is negative
        JZ      R7ZERO  ; R7 is zero
R7POS   .....           ; R7 is positive but not zero
R7NEG   .....           ; R7 is negative
R7ZERO  .....           ; R7 is zero
```

Example The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
        TST.B   R7      ; Test low byte of R7
        JN      R7NEG   ; Low byte of R7 is negative
        JZ      R7ZERO  ; Low byte of R7 is zero
R7POS   .....           ; Low byte of R7 is positive but not zero
R7NEG   .....           ; Low byte of R7 is negative
R7ZERO  .....           ; Low byte of R7 is zero
```

XOR[.W] Exclusive OR of source with destination
XOR.B Exclusive OR of source with destination

Syntax

```
XOR     src,dst    or    XOR.W    src,dst
XOR.B   src,dst
```

Operation src .XOR. dst –> dst

Description The source and destination operands are exclusive ORed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Set if result MSB is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if both operands are negative

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The bits set in R6 toggle the bits in the RAM word TONI.

```
XOR     R6,TONI     ; Toggle bits of word TONI on the bits set in R6
```

Example The bits set in R6 toggle the bits in the RAM byte TONI.

```
XOR.B R6,TONI       ; Toggle bits in word TONI on bits
                    ; set in low byte of R6,
```

Example Reset to 0 those bits in low byte of R7 that are different from bits in RAM byte EDE.

```
XOR.B   EDE,R7      ; Set different bit to “1s”
INV.B   R7          ; Invert Lowbyte, Highbyte is 0h
```

Appendix C

EPROM Programming

This appendix describes the MSP430 EPROM module. The EPROM module is erasable with ultraviolet light and electrically programmable. Devices with an EPROM module are offered in a windowed package for multiple programming and in an OTP package for one-time programmable devices.

Topic **Page**

C.1 EPROM Operation C-2

C.2 FAST Programming Algorithm C-4

C.3 Programming an EPROM Module Through a Serial Data Link Using the JTAG Feature C-5

C.4 Programming an EPROM Module With Controller's Software C-6

C.5 Code C-8

C.1 EPROM Operation

The CPU acquires data and instructions from the EPROM. When the programming voltage is applied to the TDI/VPP terminal, the CPU can also write to the EPROM module. The process of reading the EPROM is identical to the process of reading from other internal peripheral modules. Both programming and reading can occur on byte or word boundaries.

C.1.1 Erasure

The entire EPROM may be erased before programming begins. Erase the EPROM module by exposing the transparent window to ultraviolet light.

Note: EPROM Exposed to Ambient Light (1)

Since normal ambient light contains the correct wavelength for erasure, cover the transparent window with an opaque label when programming a device. Do not remove the table until it has to be erased. Any useful data in the EPROM module must be reprogrammed after exposure to ultraviolet light.

The data in the EPROM module can be programmed serially through the integrated JTAG feature, or through software included as a part of the application software. The JTAG implementation features an internal mechanism for security purposes provided by the implemented fuse. Once the security fuse is activated, the device cannot be accessed through the JTAG functions. The JTAG is permanently operating in the by-pass mode.

Refer to the appropriate data sheet for more information on the fuse implementation.

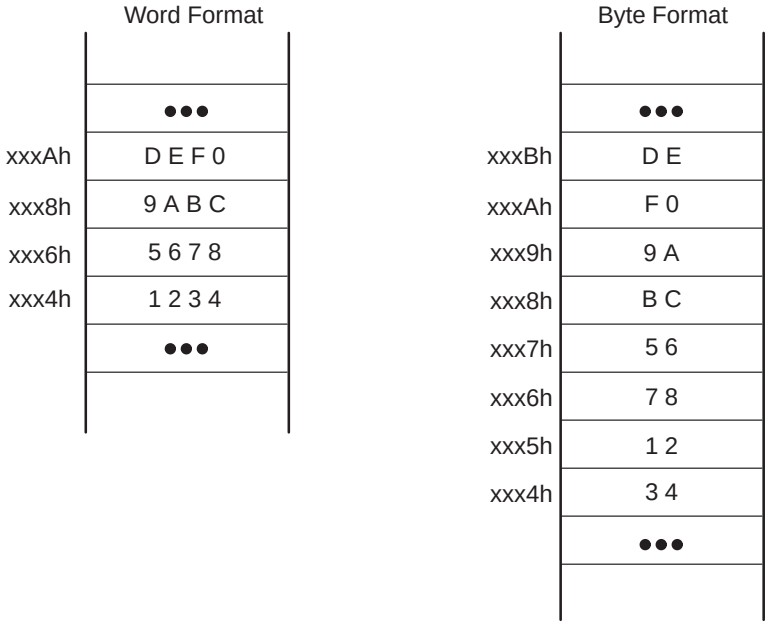
C.1.2 Programming Methods

The application must provide an external voltage supply to the TDI/VPP terminal to provide the necessary voltage and current for programming. The minimum programming time is noted in the electrical characteristics of the device data sheets.

The EPROM control register EPCTL controls the EPROM programming, once the external voltage is supplied. The erase state is a 1. When EPROM bits are programmed, they are read as 0.

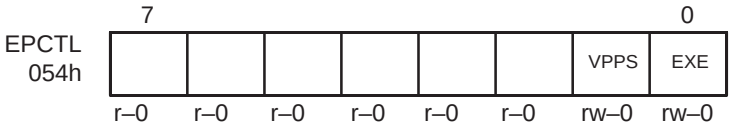
The programming of the EPROM module can be done for single bytes, words, blocks of individual length, or the entire module. All bits that have a final level of 0 must be erased before the EPROM module is programmed. The programming can be done on single devices or even in-system. The supply voltage should be in the range required by the device data sheet but at least the maximum supply voltage of the target application. The levels on the JTAG terminals are defined in the device data sheet, and are usually CMOS levels.

Example C–1. MSP430 On-Chip Program Memory Format



C.1.3 EPROM Control Register EPCTL

Figure C–1.EPROM Control Register EPCTL



For bit 0, the executable bit EXE initiates and ends the programming to the EPROM module. The external voltage must be supplied to the TDI/VPP or Test/VPP before the EXE bit is set. The timing conditions are noted in the data sheets.

For bit 1, when the VPPS bit is set, the external programming voltage is connected to the EPROM module. The VPPS bit must be set before the EXE bit is set. It can be reset together with the EXE bit. The VPPS bit must not be cleared between programming operations.

Note:

Ensure that no VPP is applied to the programming voltage pin (TDI/VPP or Test/VPP) when the software in the device is executed or when the JTAG is not fully controlled. Otherwise, an undesired write operation may occur.

C.1.4 EPROM Protect

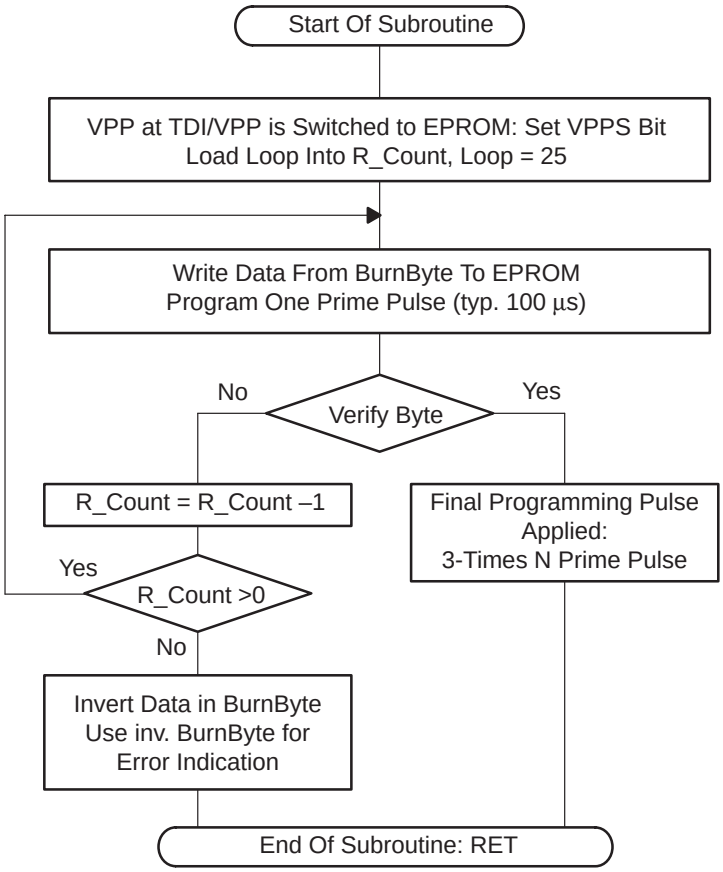
The EPROM access through the serial test and programming interface JTAG can be inhibited when the security fuse is activated. The security fuse is activated by serial instructions shifted into the JTAG. Activating the fuse is not reversible and any access to the internal system is disrupted. The by-pass function described in the standard IEEE 1149.1 is active.

C.2 FAST Programming Algorithm

The FAST programming cycle is normally used to program the data into the EPROM. A programmed logical 0 can be erased only by ultraviolet light.

Fast programming uses two types of pulses: prime and final. The length of the prime pulse is typically 100μs (see the latest datasheet). After each prime pulse, the programmed data are verified. If the verification fails 25 times, the programming operation was false. If correct data are read, the final programming pulse is applied. The final programming pulse is 3 times the number of prime pulses applied.

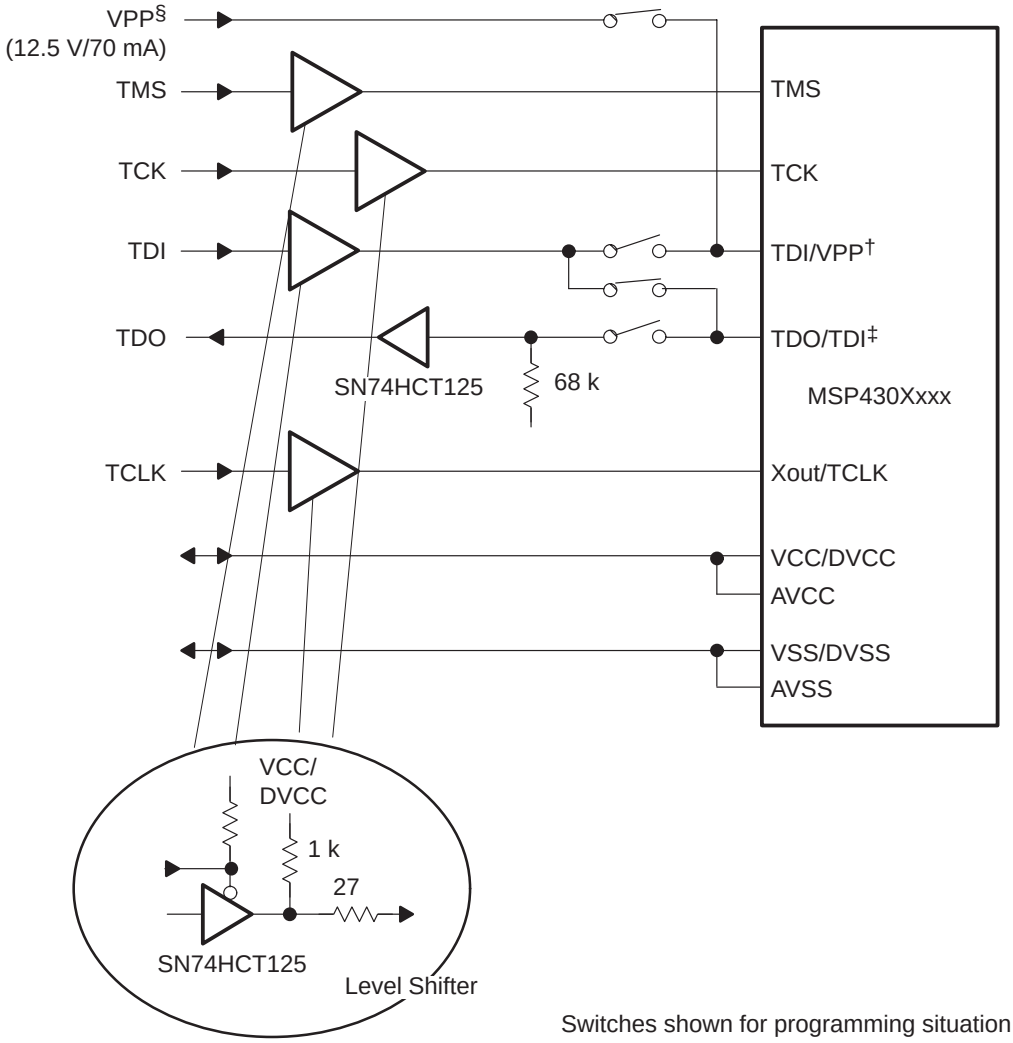
Example C–2. Fast Programming Subroutine



C.3 Programming an EPROM Module Through a Serial Data Link Using the JTAG Feature

The hardware interconnection of the JTAG terminals is established through four separate terminals, plus the ground or VSS reference level. The JTAG terminals are TMS, TCK, TDI(/VPP), and TDO(/TDI).

Figure C–2.EPROM Programming With Serial Data Link



† TDI in standard mode, VPP input during programming
‡ TDO in standard mode, data input TDI during programming
§ See electrical characteristics in the latest data sheet

C.4 Programming an EPROM Module With Controller's Software

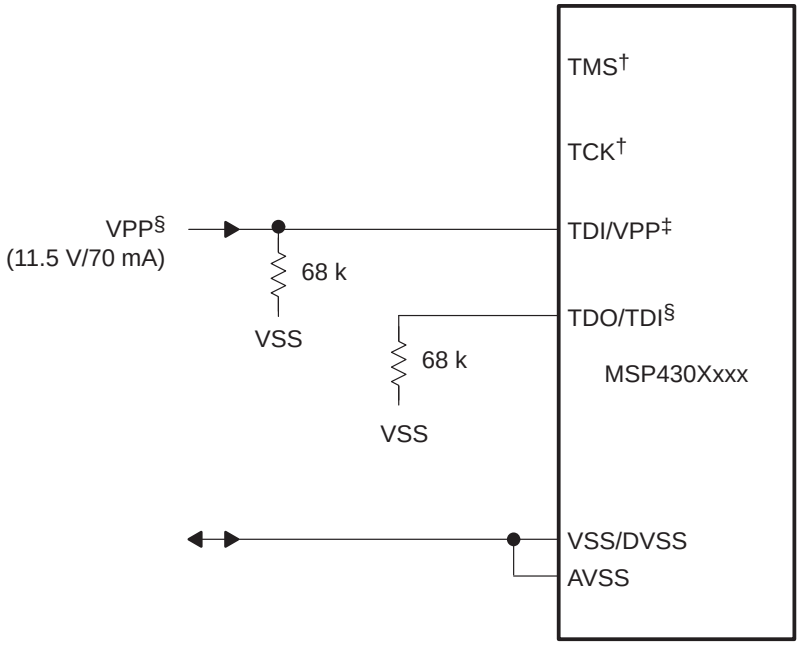
The procedure for programming an EPROM module is as follows:

1) Connect the required supply to the TDI/VPP terminal.

2) Run the proper software algorithm.

The software algorithm that controls the EPROM programming cycle cannot run in the same EPROM module to which the data are being written. It is impossible to read instructions from the EPROM and write data to it at the same time. The software needs to run from another memory such as a ROM module, a RAM module, or another EPROM module.

Figure C–3.EPROM Programming With Controller's Software

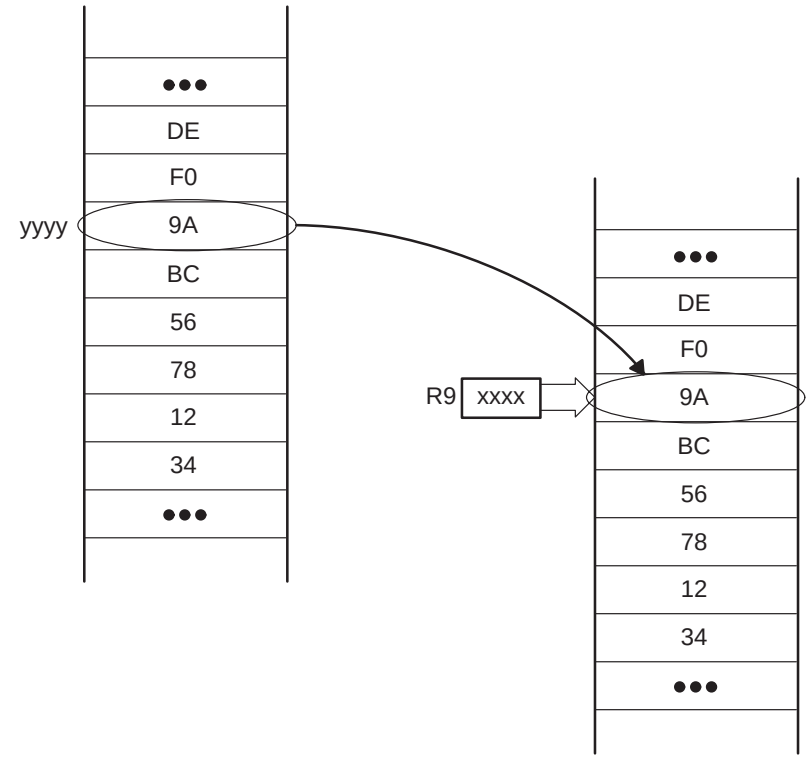


† Internally a pullup resistor is connected to TMS and TCK
‡ ROM devices of MSP430 have an internal pullup resistor at pin TDI/VPP. MSP430Pxxx or MSP430Exxx have no internal pullup resistor. They should be terminated according to the device data sheet.
§ The TDO/TDI pin should be terminated according to the device data sheet.

C.4.1 Example

The software example writes one byte into the EPROM with the fast programming algorithm. The code is written position-independent, and will have been loaded to the RAM before it is used. The programming algorithm runs during the programming sequence in the RAM, thus avoiding conflict when the EPROM is written. The data (byte) that should be written is located in the RAM address BurnByte. The target address of the EPROM module is held in the register pointer defined with the set directive. The timing is adjusted to a cycle time of 1μs. When another cycle time/processor frequency is selected, the software should be adjusted according to the operating conditions.

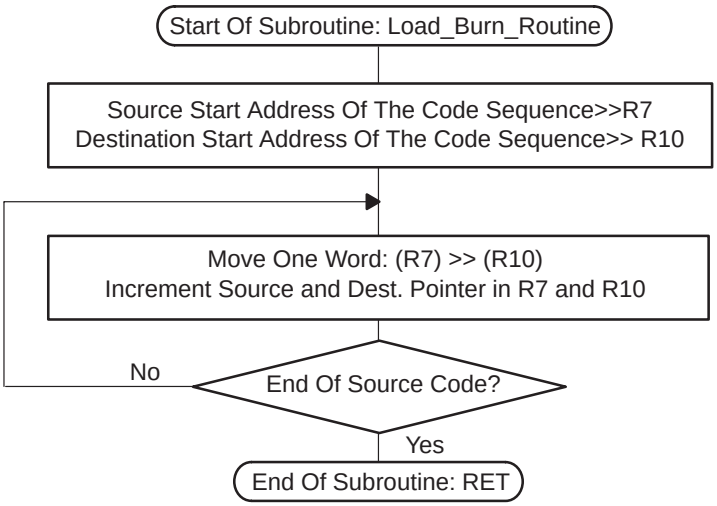
Example C–3. Programming EPROM Module With Controller's Software



Example: Write data in yyyy into location xxxx
BumByte = (yyyy) = (9Ah)
R9 = xxxx

The target EPROM module cannot execute the programming code sequence while the data are being written into it. In the example, a subroutine moves the programming code sequence into another memory, for example, into the RAM.

Example C–4. Subroutine



C.5 Code

```
;-------------------------------------------------------------
; Definitions used in Subroutine :
; Move programming code sequence into RAM (load_burn_routine)
; Burn a byte into the EPROM area         (Burn_EPROM)
;-------------------------------------------------------------

EPCTL     .set   054h  ; EPROM Control Register
VPPS      .set   2     ; Program Voltage bit
EXE       .set   1     ; Execution bit
BurnByte  .set   0220h ; address of data to be written
Burn_orig .set   0222h ; Start address of burn
                       ; program in the RAM
loops     .set   25
r_timer   .set   r8    ; 1us = 1 cycle
pointer   .set   r9    ; pointer to the EPROM address
                       ; r9 is saved in the main routine
                       ; before subroutine call is executed
r_count   .set   r10
lp        .set   3     ; dec r_timer   : 1 cycle  : loop_t100
                       ; jnz           : 2 cycles : loop_t100
ov        .set   2     ; mov #(100-ov)/lp,r_timer : 2 cycles
```

; Load EPROM programming sequence to another location e.g. RAM, Subroutine

```
;--- The address of Burn_EPROM (start of burn EPROM code) and
;--- the address of Burn_end (end of burn EPROM code) and
;--- the start address of the location of the destination
;--- code area (RAM_Burn_EPROM) are known at assembly/linking time

RAM_Burn_EPROM  .set    Burn_orig
load_burn_routine
   push     r9
   push     r10
   mov      #Burn_EPROM,R9        ; load pointer source
   mov      #RAM_Burn_EPROM,R10   ; load pointer dest.
load_burn1
   mov      @R9,0(R10)            ; move a word
   incd     R10                   ; dest.  pointer + 2
   incd     R9                    ; source pointer + 2
   cmp      #Burn_end,R9          ; compare to end_of_table
   jne      load_burn1
   pop      r9
   pop      r10
   ret
```

; Program one byte into EPROM, Subroutine

```
;--- Burn subroutine: position independent code is needed
;    since in this examples it is shifted to RAM  >> only
;--  relative addressing, relative jump instructions, is used!
;--- The timing is correct due to 1us per cycle

Burn_EPROM
      dint                          ; ensure correct burn timing
      mov.b #VPPS,&EPCTL            ; VPPS on
      push  r_timer                 ; save registers
      push  r_count                 ; programming subroutine
      mov   #loops,r_count          ; 2 cycles = 2 us
Repeat_Burn
      mov.b &BurnByte,0(pointer)    ; write to data to EPROM
```

```
                                ; 6 cycles = 6 us
      bis.b #EXE,&EPCTL         ; EXE on
                                ; 4 cycles = 4 us
                                ; total cycles VPPon to EXE
                                ; 12 cycles = 12 us (min.)
      mov   #(100-ov)/lp,r_timer  ;:programming pulse of 100us
wait_100                        ;:starts, actual time  102us
      dec   r_timer             ;:
      jnz   wait_100            ;:
      bic.b #EXE,&EPCTL         ;:EXE / prog. pulse off

      mov   #4,r_timer          ;:wait min. 10 us
wait_10                         ;:before verifying
      dec   r_timer             ;:programmed EPROM
      jnz   wait_10             ;:location, actual 13+ us

      cmp.b &BurnByte,0(pointer)  ; verify data = burned data
      jne   Burn_EPROM_bad        ; data ‡ burned data > jump

; Continue here when data correctly burned into EPROM location
      mov.b &BurnByte,0(pointer)  ; write to EPROM again
      bis.b #EXE,&EPCTL         ; EXE on
      add   #(0ffffh-loops+1),r_count
                                ; Number of loops for
                                ; successful programming
final_puls
      mov   #(300-ov)/lp,r_timer  ;:programming pulse of
wait_300                        ;:3*100us*N starts
      dec   r_timer             ;:
      jnz   wait_300            ;:
      inc   r_count             ;:
      jn    final_puls          ;:
      clr.b &EPCTL              ;:EXE off / VPPS off
      jmp   Burn_EPROM_end
Burn_EPROM_bad
      dec   r_count             ; not ok : decrement
                                ; loop counter
      jnz   Repeat_Burn         ; loop not ended : do
                                ; another trial
      inv.b &BurnByte           ; return the inverted data
                                ; to flag
                                ; failing the programming
                                ; attempt the EPROM address
                                ; is unchanged
                                ;
Burn_EPROM_end
         pop   r_count
         pop   r_timer
         eint
         ret
Burn_end
```