COP8TM Development Tools QUICKSTART FOR THE DEBUG MODULE

March 1998

REVISION RECORD

REVISION A **RELEASE DATE** 03/98

SUMMARY OF CHANGES First Release

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

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

© Copyright National Semiconductor Corporation, 1998

COP8 is a trademark of National Semiconductor Corporation PC is a trademark of International Business Machine Corporation iceMaster is a trademark of MetaLink Corporation Windows and Windows NT are trademarks of Microsoft Corporation InstallShield is a registered trademark of InstallShield Corporation

S P

What you need

- Computer and monitor : 486 or higher PCTM with at least 8 MB RAM (16 MB recommended), a hard disk with at least 20MB of free disk space and a mouse
- Windows[™] 95, Windows NT[™], or Windows 3.11 running in enhanced mode (The descriptions in the document will assume a Windows 95 environment)
- (Optional) Printer
- A LED (Light Emitting Diode) and a resistor (at least 330 Ohms)

Installing ASMCOP/Linker/Lib, MetaLink iceMaster™ Debugger, and WCOP8 IDE

Begin by clearing the memory by exiting all tasks:

Identify any resident programs by lowering the mouse cursor to the taskbar, clear any program by clicking on them and exiting them. This is usually accomplished by clicking on **<u>File</u>**[**Exit**.

Install ASMCOP/Linker/Lib

- 1. Insert the disk labeled ASM/Linker/Lib into the floppy drive
- 2. Click the start button and select Run
- 3. At the windows prompt Open:
 - A) Type in a:install (where a: is your floppy drive)B) At the dos prompt: Source drive of installation disk [A]:,
 - Select a: (where a: is the drive in which the floppy is in) C) At the dos prompt for Directory for COP8 [C:\cop]:,
 - Select c:\cop (where c: is the hard drive) D) Depress the return key
- 4. The install program will now decompress the necessary files.
- 5. When done, type in "exit " and hit the <RETURN> key.

Install iceMaster-Debug Module-COP8

- 1. Insert the disk labeled *iceMaster-Debug Module-COP8* Disk1 into your floppy drive
- 2. Click the start button and select Run

A S

COP8 Debug Module Development Tools

- 3. At the windows prompt Open:
 - A) Type in a:\setup (where a: is your floppy drive)
 - B) Click $\underline{\mathbf{Y}}$ es to the prompt "Install COP8 Emulator"
 - C) Change disk(s) as requested by InstallShield[®]
 - D) Select a destination path and Click on \underline{N} ext
 - E) Click <u>N</u>ext to add files to the Program Folder
 - F) Click $\underline{\mathbf{O}}$ k to the next three menus

Install WCOP8 IDE

- 1. Insert the disk labeled *KKD WCOP8 IDE Disk1* into your floppy drive
- 2. Click the start button and select Run
- 3. At the windows prompt Open:A) Type in a:setup (where a: is your floppy drive)B) Click <u>Next</u> when you are done reading the information window
- 4. A screen will appear asking for your name, company, and the serial number on the label of the WCOP8 IDE disk. Type in all the necessary information and when done click **Next**.
- 5. A window will pop up asking for which type of installation to do.
 - A) Click on the space next to "Make new installation and overwrite all old settings" if this is a new installation of WCOP8 IDE.
 - B) Click <u>N</u>ext when done.
- 6. A window will ask for "Destination Location" which WCOP8 IDE will be installed at.
- A) Click on <u>Next</u> for the default path or click on <u>Browse</u> for a to enter another destination path.
- B) Click on $\underline{N}ext$ when done.
- 7. A window will ask for the type of Operation System in which WCOP8 IDE will be installed into.
 - A) Select the type of Operation System.
 - B) Click on <u>Next</u> when done.
- 8. A window pane will pop up asking which Program groups should the WCOP8 icons be installed.
 - A) Select a program group
 - B) Click <u>N</u>ext when done.
 - C) Click on \underline{Next} to decompress the files and finalize the install process.
 - D) Click on $\underline{\mathbf{F}}$ inish when done to exit the install program.

At the end of the installation(s) you can verify that the correct files have been installed by using Windows Explorer and comparing your installation to that shown in Fig 1.

Y

UICKSTA

 \bigcirc

🔄 Cop	-	主 🚈 🏧	X 🖻 🛍 🗠	× 🗗 🥤
All Folders		Contents of 'E:\COP'		
🚵 Desktop	-	Asmcop.exe	Cop888cl.inc	Libcop.exe
🔄 🗐 My Computer		Asmcop.hlp	Cop888cs.inc	🧑 Libcop .hlp
🕂 🖃 3½ Floppy (A:)		Asmread.me	Cop888eb.inc	Lmhex.exe
⊕ 🖃 5% Floppy (B:)		Asmrel.let	Cop888eg.inc	Lncop.cfg
🛨 👝 Wallace (C:)		Cop8.inc	Cop888ek.inc	Lncop.exe
🕀 🌆 Audio CD (D:)		Cop820.asm	Cop888ew.inc	A Lncop.hlp
🖃 🛲 Zip 100 (E:)		Cop820.inc	Cop888fh.inc	Promcop.ex
🕀 💼 backup		Cop8201.h	Cop888gd.inc	Runsamp.ba
Cop		Cop820a.asm	Cop888gg.inc	Sample.bat
En Cop8		Cop820b.asm	 Cop888gw.inc 	Xcrc.exe
⊡ <u> </u>		Cop820cj.inc	Cop888hg.inc	Xere.out
Example		Cop840.inc	Cop888kg.inc	a voic.ou
Plugins		Cop840cj.inc	Cop8acc.inc	
Implate		Cop8620.asm	Cop8saa.inc	
Metalink				
🖃 - 🧰 Whp2380		Cop8620.h	Cop8sab.inc	
Examples		2 Cop8620.inc	Cop8sac.inc	
		Cop8620.lib	Cop912c.inc	
		Cop8620.m	🗃 Copxxx.fil	
		(1) Cop8640.inc	Copxxx.h	
		Cop8780.inc	Copxxx1.h	
		Cop880.inc	Copxxx2.h	
		Cop888bc.inc	Dumpcoff.exe	
		Cop888cf.inc	Hexdiff.exe	
		Cop888cg.asm	Hexim.exe	
		Cop888cg.inc	🚞 install.exe	

Figure 1.

Installing the Debug Module

1. Begin by identifying all the parts of the system. Locate the serial connection cable, base unit, a power supply (there should be two for the Debug Module), and a 40 pin ribbon cable with a header. The setup should look like Fig. 2.

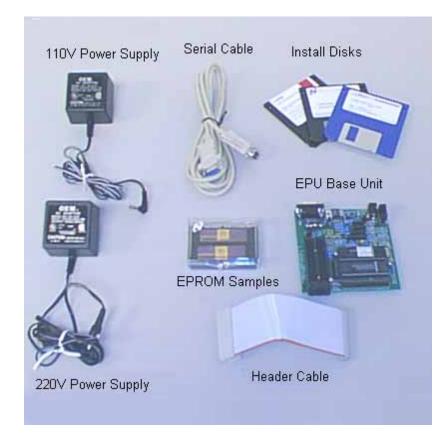


Figure 2.

A S

> 2. Connect the power supply module (110V or 220V) to the Debug Module. Plug in the power supply but do not turn on the power yet!

3. Connect the serial cable to the PC and then connect the other end to the Debug Module. This is shown in the Fig 3.



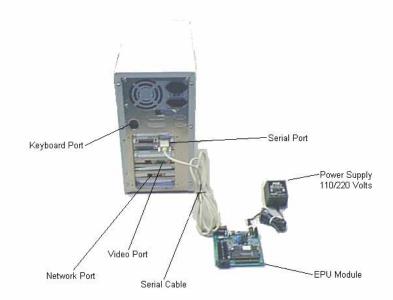


Figure 3.

- 4. Install the ribbon cable with the 40 pin header onto the connector J20 on the Debug Module.
- 5. Click on the Start menu, select Programs, select WCOP8 IDE and the program will start.

The Quick Start Exercise

The sections which follow demonstrate the typical steps and procedures for entering and modifying a program written for the COP8 Assembler, running the Assembler and Linker, downloading the program into the Debug Module for purposes of debugging and eliminating errors, and finally programming a COP microcontroller EPROM or OTP part.

We make the assumption that you are somewhat familiar with embedded microcontrollers, software text editors and assemblers, and some form of debugging tool. By following this document closely you will be able to create one example of a working set of firmware even if you've never developed software for a microcontroller. No previous experience with COP8 microcontrollers is required to understand and use the example program.

A Note on Developing Software

The first step in developing application software is to carefully specify the operational requirements. Flow-charts or some other technique can be used to document the program sequences in the software (such as the one shown in Fig. 4). Fig. 4 is a high level "idea chart" that we will use for our exercise program. In many cases new application software is written by modifying existing software. A sample program, (main.asm) supplied with WCOP8 IDE, is used for our example. Using Windows utilities (click and drag are easiest) copy the example to your quick project directory. WCOP8 IDE allows you to organize software development into projects. The following briefly delineates the steps to set up a project.

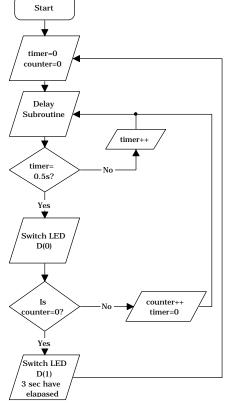


Figure 4.

4



Figure 5.

Launch the WCOP8 IDE by clicking on the Start Menu|Programs|WCOP8 IDE. WCOP8 IDE will scan the hard drive for ASMCOP/ LNCOP/PROMCOP[®] and ByteCraft's COP8C[®] compiler when it is ran for the first time. It will then create the appropriate settings for your machine. Select **Project**|New Project, and, on the New Project window (Fig.5), locate the directory c:\cop8\project\ quick. At the File Name prompt type in main.prj. This will be the project name of our lesson. Click OK, and the Project Files window appears (Fig. 6). **CKSTAR1**

SWCOP8 IDE				
<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>S</u> earch <u>O</u> ptions	Project Debug	<u>Execute</u>	Window	<u>H</u> elp
	New Projec Open Projec Close Projec	x	與戰	22
	Save Projec			
	<u>A</u> dd File(s).			
	Remove File	;		
	Project <u>S</u> etti	ngs	P	
CI.	Rescan For	<u>T</u> ools		
s Directo	No.	fu		
Project - e:\cop8\wcop8\e:	xample\main	.prj		
Project				
 e:\cop8\wcop8\example\main.prj Project files 				
Files Information				
			Pro	oject: main.prj

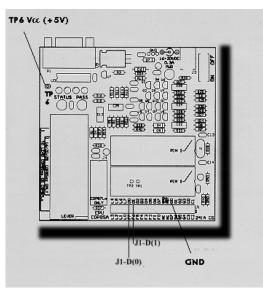
Figure 6.

Select **Project** Add Files Select main.asm as the only file for this project. Click OK, and the . The project main.prj has been created, and consists only of the one file main.asm. Click on the main.asm icon, and edit the file so that it corresponds to the listing in Appendix A, and save it.

Step1. Setup of the Circuit

The modified software, when executing on your Debug Module, will blink an LED at two different rates. The LED and a series resistor are connected between V_{cc} and one of 2 PORT "D" I/O pins. Attach a clip lead to the post TP6 which is V_{cc} . This will be used to obtain LED voltage. (See Fig. 7a and 7b).





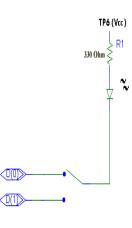


Figure 7b.

COP8 OU

When the assembled circuit (Fig 7b.) is connected, and the program is running, the LED attached at Debug Module J20-D0 will blink at approximately 1/2 second on, 1/2 second off. When the LED is attached to Debug Module J20-D₁ it will blink at approximately 3 seconds on, 3 seconds off. Rate will vary a bit depending on your processor speed. Since the Debug Module simulates fetching instructions over the serial (RS-232) host port, the dominate timing parameter is the baud rate.

Figure 7a.

Executing 🗵
Status: Finished
Information:
Project: e:\cop8\wcopmple\main.prj File: *.obj Time: 9.60 sec
Result: File(s) linked with success
✓ <u>0</u> k

Figure 8.

Click on the main.asm icon. Select **Execute**|**Build** or click on the build icon. A window with the title Executing will pop up. WCOP8 IDE will assemble, and if there are no assembly errors link the program.

If there are assembly errors, an error map is displayed. If no error(s) occurred then a display such as the one in Fig. 8 will be given.

Large embedded microprocessor projects frequently contain more than one file (module) each of which is assembled separately. The Assembler outputs are then linked together and tested as a whole. WCOP8 IDE has a **make** function that assembles only the files that have changed, and then links the files to produce the symbolic output ready for loading into the Debug Module or other MetaLink emulation tool. For this feature select **Execute**|**Make** or click on the make icon.

WCOP8 IDE provides for consolidation of all of these modules as a **project**, and includes several features for the orderly processing of these multiple modules. This example, set up as a project even though the source is a single file module, nevertheless illustrate the principles of project management. Chapter 8 in the WCOP8 IDE User's Manual, *Using the WCOP8 IDE in Project Mode*, covers these additional features in detail.

Step 2. Debugging and Testing Software on the Debug Module

We will now setup WCOP8 IDE so that it will recognize the MetaLink iceMaster Debugger. Click on **Project**|**Project Settings.** Double Click MetaLink tools. Click on **Window COP8 Emulator**. Click on **Browse** button and locate where MetaLink's debug program is located. A path/program name similar to "e:\metalink\whp2380\whp2380.exe" should be found. A window similar to Fig. 9 will now be displayed.

🦠 Project Settings - quick.prj	×
NSC assembler Sylve Craft C compiler MetaLink tools DOS EPU DOS bebug Module DOS bebug Module DOS iceDEMO/iceSIM Windows COP8 Emulator	Windows COP8 Emulator MetaLink Windows COP8 Emulator: E*metaInkWnp2380whp2380 exe Image: Browse Command line parameters Image: Command line parameters Image: Non-Source Command line parameters
V Qk X Cancel	Inherit Project Settings

Figure 9.

Click on the box next to the sentence "**Use command line parameters**." Click on the selection National COFF - file (.cof) parameters Click <u>**Ok**</u> when done.

Connect the ribbon cable with the header pins to the target hardware, in this case the LED and resistor, to J20. (In a typical application, the supplied cable will be used to connect between Debug Module J20 and the microprocessor socket on the target hardware.) Power up the Debug Module and click on **Debug|Windows COP8 Emulator** to activate the Debug Module and the PC. A window will pop up asking you to select a project directory . Select a directory and click **OK**. A **Select Chip** window will pop up. Select the 8SGR (40 - pin configuration) as the **Emulation Device** and Click **OK**. Another window will pop up asking for a communications port (COM1-COM4) in which the Debug Module is connected to. Click the appropriate COM port and click **OK**. The specific device and com port information will be preserved and the user will not have to re-enter information the next time he/she load up the same project directory.

Step 3. Configuring the Debug Module

The Debug Module software generally locates the serial port through the configuration file used by the Debug Module, and establishes the connection between the PC and the Debug Module. If there is a problem, use

Configure|**Emulator** to select the serial port and baud rate. While the serial port is



usually set to the highest baud rate, it is sometimes necessary to set the baud rate to a lower value to ensure reliable operation. Refer to the "troubleshooting" section (pg. 5/6) of the Debug Module manual if you encounter any configuration problems.

Step 4. File

Select **File**|**Load** so that the executable (in this case main.cof) can be entered into the File Name box. At the prompt, "Merge into current application environment?", select no (merge allows multiple files to be loaded into memory without pre-initialization to all 0x00 content).

Note: By displaying main.asm in the edit window when selecting Debug|Windows Emulator, the default directory used by the Debug Module will be c:\cop8\ The full path for the executable, main.cof, need not be entered into the File Name entry box.

Once loaded the Debug Module is ready to execute, and following your directions, to test the example If you are optimistic you can simply click RUN and see the result. By following the techniques described below you will learn to use some of the testing features available using the Debug Module and iceMaster debugger software.



Step 6. Adding/Using a simple Break/Trace

This section covers the basics for setting, editing and removing breakpoints. Breakpoints are generally inserted at critical points in the program to verify program operation. Once started, the microcontroller runs until the next

breakpoint is reached. The process is then frozen and the microcontroller's internal state is displayed for examination and possible change. When ready, the process can be resumed (to run until the next breakpoint) or restarted.

Sourc						
	ons <u>V</u> iew	Assemble Toggle-Breakpoin				
	BCEEA0		LD	CNTRL,#X'A0	ram(EE)=00	
	BCEAE8		LD	TMR1LO,#X'E8		
	BCEB03		LD	TMR1HI,#X'03		
	BCECF4		LD	T1RALO,#X'F4		
	BCED01		LD	T1RAHI,#X'01		
	BCE6E8	Break Flags	LD	T1RBLO,#X'E8		
	BCE703	//	LD	T1RBHI,#X'03		
	BDD46B	/	RBIT	3,PORTGD ;@GD3_T1A		
	BDD578 BCEF1		SBIT LD	3,PORTGC ;@GC3_T1A		
	BOGETC		SBIT	PSW,#X11		
	AØ0028		JSRL	4,CNTRL;@T1C0 X'0028		
0021 0	AD0200		JSRL	X0220		
	F9		JP	X'0021		
	DOFF		LD	F0,#X'FF		
	CO		DRSZ	FO		
	FE		JP	X'002A		
	8E		RET			
	00		INTR			
002E	00		INTR			
002F	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00		INTR			
	00 00		INTR			
	00		INTR			
	00		INTR			
	00					
003E						1

Figure 10.

The Debug Module also retains a trace of the most recent 100 frames that occurred in the execution cycle. In addition to the trace, content of the internal registers and stack, condition of the input/output ports, and memory content (RAM and ROM) are also available.

A breakpoint is added by clicking on the code line, and then clicking on **Toggle-Breakpoint**. A breakpoint is enabled when a small square appears to the right of the instruction address (Fig. 10). Using the Debug Module you can enable up to 32k breakpoints. A breakpoint can be cleared by selecting the set breakpoint and clicking on Toggle-Breakpoint a second time. Here we will add a breakpoint at line 21 and line 24. Line 21 and Line 24 are where the subroutines are called.

Step 7. Running the Code

<u>S</u>

Note that the Debug Module is an in-circuit **simulator** as opposed to the more common in-circuit **emulator**. While the in-circuit emulator runs in real time, the in-circuit simulator is controlled by software, executes instructions one at a time, and runs much slower (approximately 10 KHz). Instruction fetch and trace are performed within the PC with the microcontroller code memory loaded cycle by cycle over the serial port. Execution speed is primarily a function of baud rate.

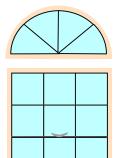
It is good practice to reset the microprocessor before starting the simulation. This is done by selecting **Run**|**Reset**|**Processor**. Selecting **Run**|**Go** (function key **F4**) causes the processor to run to the next breakpoint and stop. Select **Run**|**Go**.

ile Op	tions ⊻iew	<u>A</u> ssemble	Toggle-Breakpoint	Run-Until				
0000	BCEEA0			LD	CNTRL,#X		ram[EE]=00	
0003	BCEAE8			LD	TMR1LO,#			
0006	BCEB03			LD	TMR1HI,#			
0009	BCECF4			LD	T1RALO,#			
000C	BCED01			LD	T1RAHI,#>			
000F	BCE6E8			LD	T1RBLO,#			
0012	BCE703			LD	T1RBHI,#>			
0015	BDD46B			RBIT);@GD3_T1A		
0018	BDD57B			SBIT		;@GC3_T1A		
001B	BCEF11			LD	PSW,#X'11			
001E	BDEE7C			SBIT	4,CNTRL;	@T1C0		
	AD0028			JSRL	X'0028		v0028 SP=6F	
	AD0200			JSRL	X'0200			
0027	F9			JP	X'0021			
0028	DOFF			LD	F0,#X'FF			
002A	CO			DRSZ	FO			
002B	FE			JP	X'002A			
002C	8E			RET				
002D	00			INTR				
	ce <1>							
	tions ⊻iew	<u>S</u> earch						
- 36	0000			3	LD	CNTRL,#X'A0		
- 33	0003			3	LD	TMR1LO,#X'E8		
- 30	0006			3	LD	TMR1HI,#X'03		
- 27	0009			3	LD	T1RALO,#X'F4		
- 24	000C			3	LD	T1RAHI,#X'01		
- 21	000F			3	LD	T1RBLO,#X'E8		
- 18	0012			3	LD	T1RBHI,#X'03		
- 15	0015			4	RBIT	3,PORTGD;@GD3_T1A		
- 11	0018			4	SBIT	3,PORTGC ;@GC3_T1A		
- 7	001B			3	LD	PSW,#X'11		
- 4	001E			. 4	SBIT	4,CNTRL;@T1C0		
	Ins 0021	struction Ab	out to be Executed	1:	JSRL X'002			



Select **Window**|**Trace** to use the Debug Module trace facility. This allows the user to view the instructions that have been executed prior to the breakpoint. After arriving at the breakpoint and enabling the **Trace** function you should have a window similar to that of Fig. 11. This is important when verifying instruction execution based on branches within the program. An alternative method of simulation is to step through the program one instruction at a time. While this approach can be time consuming, it is possible to determine the step-by-step status of the microcontroller. This is accomplished by selecting **Run**|**Step** (function key **F7**).

A Note On Window Displays



Simulation results are shown in the Debug Module window which is divided into five window panes: Source, Core Registers, Registers, Status and RAM Memory. Each window pane can be expanded so that all information can be viewed. The user can also adjust the size of the window panes to suit the data viewing requirements.

The Source window pane shows the hexadecimal machine code and the source assembly code, and indicates the active breakpoints. The RAM window pane shows data in the RAM memory. The Status

window pane presents the simulation data including breakpoint address and other related data. The Registers window pane shows the data in the registers, the timers and input/output ports. The Core Register window pane shows the accumulator, stack pointer, B and X registers as well as the flags in the Program Status Word (PSW).

Since the first few instructions in the example program set up timer registers, results of these instructions can be verified in the Registers pane.

Step 8. Misc. Section - Programming the (E)PROM

This section contains the procedures for programming the COP One Time Programmable (OTP) and erasable microprocessors supported by the Debug Module. Select **File**|**PROM Programmer**|**Device** to display the set of COP devices that can be programmed by the Debug Module. Select the appropriate device from the list.



A window similar to that of Fig. 12 should pop up to allow programming of the COP microcontroller.

MI COPE	8 MetaLink	ICE - COP8SAC (D	M)	
Elle Con	figure <u>R</u> un	Display/Alter Break/T	race Window Help	
Load.				🗶 🌃 Core Registers <1> 💷 🗵
Ugioa	d	emble Toggle-Bre	sakpoint Run-Until	Elle Options
Store.			LD CNTRL,#XA0 tan/EE =00	A 00
Resto			LD TMR1L0,#XE8	B E5 X A5
_			LD TMR1HI,#X103 LD T1RALO,#XF4	SP 6F
BROM	Programmer	- No	LD T1RAHI,#X01	HC 0
E⊻it		-0	f PROM Programmer 🛛 🔀	C 0 T1PNDA 0
0012	BDD46B		Load File Source Code Memory Configuration Checksum	TIENA 0
0018	BDD57B			EXPND 0
001B	BCEF11		Device COP8SGR7N40 (40-pin DIP)	BUSY 0
001E	BDEE7C		ECON (Configuration/Device): 0x00/Unread	EXEN 0 GIE 0
0021	AD0028 AD0200		Security bit (Configuration/Device): Disabled/Unread	OIE 0
0027	F9		User Data (Configuration, 0x8001-0x8008): 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x	
0028	DOFF		User Data (Device): Unread	
002A	CO		Programming Adapter not required.	
002B 002C	FE 8E		r regramming radiate nor required.	
002C	00			
002E	00		Install COP8Sxx/COP87Lx (JB35) shunt on DM4 before programming.	
002F	00			
0030	00		Warning: Do not put a chip in programming socket until told to do so.	
0031	00 00			Registers <1>
0033	00		Programming	File Options
0034	00		© Automatic Start Operation	PORTED 00
0035	00		O Manual EPROM Blankcheck	PORTEC 00
0036	00 00			PORTEP FF
0037	00		Exit	WDSVR C1
0039	00		INTR	WKEDG 00 WKEN 00
003A	00		INTR	WKPND FD
003B	00		INTR	PORTLD 00
003C	00		INTR	PORTLC 00
-	1 Memory <	4	Status <1>	PORTLP F5
	lMemory< tions ⊻iew		File Options	PORTGC 00
				PORTGP FE
010	31 19 DD 5	6 26 BA E8 9F 6D	78 80 5D DE 83 4A 62 ' (.X. # (.].▲ 9 4E CE AE 4F C6 6D 71 1 V & mNC■ Break Address: 0x0000	PORTCD 00 PORTCC 00
020	32 42 B7 9	IC 38 77 42 E7 14	C7 1D 30 11 16 4B 23 2 B 8 w B 0 . ock (microseconds): 0	PORTCP EF
			52 33 B4 B0 B9 0C 97 ; w_~ HR 3 e Count (resets,etc.): 0	PORTD FF
			75 42 75 40 C6 6A 7B C S. wu B u C Repetition Count: 1 43 8A 30 76 B5 B3 EA @Hs1.0}, %C.0v Emulation Status; None	T1RBLO 09
			C2 18 A6 11 10 D3 A1 I G U 3 Trace Status: Empty	TIRBHI FF
			FF FF FF FF FF FF FF FF Trace Read: 0%	ICNTRL 00
080	FF FF FF F	F FF FF FF FF FF	FF	TMR1L0 FF

Figure 12.

Clicking on the Configuration button will bring up another window (Fig 13.) which will allow a detailed configuration of the microcontroller. For the 40 pin devices, the configuration of the COP device must be finalized.

Select Security, and then	🚮 PROM Programmer Configuration Op	otions 🔀
choose Disable or Enable. For testing purposes choose Disable. The clock option is chosen by selecting Clock Option.	Clock Crystal (internal bias resistor disabled) Crystal (internal bias resistor enabled) Crystal Crystal Crystal Crystal Crystal Crystal Crystal Crystal	Halt Mode © Enabled Disabled Port E © Enabled Disabled Disabled
For testing purposes	Power On Reset RAM Size	ecurity <u>W</u> atchdog
choose RC Oscillator. The clock configuration on the		Enabled © Enabled Disabled © Disabled
target hardware will determine	ne Fi	gure 13.

the selection of External Oscillator, RC Oscillator or Crystal Oscillator. Enable the POR (Power On Reset) circuit by clicking on the Power On Reset|Enabled.

The RAM size selection is not available for the SGR Debug Module. If the program has not been loaded, then select Load to load the program into the Debug Module so that the COP microprocessor can be programmed. This is not necessary if the program has been loaded as part of the debugging process; the otp will program from the same memory that was used by the debugger for simulation.

Programming the COP microcontroller is accomplished by selecting File|Programming|Automatic and clicking on the Start Operation. This first checks that the COP device is blank, programs the code, and Configuration (ECON) and Signature registers, and verifies the programming by reading the just programmed device and comparing the data to the file. Selecting File|Programming|Manual|Eprom Program will program only the code space.

Click on the button **Start Operation** to begin programming our microcontroller. Follow the directions on the pop up window. After programming Click on the Exit button to get back to the main client window. The software will ask you to remove the chip from the programming socket. Make sure the chip is not in the socket. Leaving the chip in may cause damage to both the simulator board and/or the surrounding circuit.

After "burning" the microcontroller you can test the behavior of the code at full speed. Replace the ribbon cable header with the newly burnt chip and apply a clean 5 volts (preferably from a power supply.) Detach the clip from TP6 and attach it to a supply VCC (+5V) and the microcontroller ground to that of supply GND. Make sure that the /RESET line is tied high as to enable the POR (Power On Reset) circuit. The setup should look similar to that of Fig. 14.

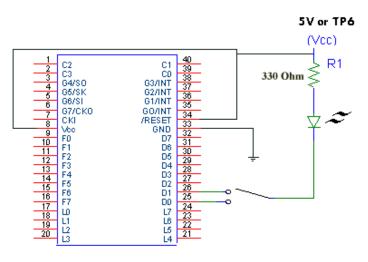
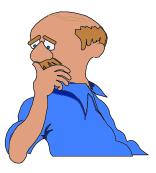


Figure 14.

Step 9. Conclusion And Final Thoughts

The *WCOP8 IDE* is a powerful software tool for organizing the development of single and multiple module programs for the COP8 family of microprocessors. The Debug Module emulator is an inexpensive tool for debugging and testing COP8 software and verifying the operation of the target hardware.



Appendix A

Assembly Code For The QuickStart Lesson

; ; COMPANY : K&K Development *; ;* PROJECT : WCOP8 IDE Test Assembler Project *; ;* FILENAME : Main.asm *; ;* VERSION : 1.0 *; ; *_____ - * : ;* This file is part of the test project that comes *; ;* with the WCOP IDE software. The only purpose of the *; ;* project is to serve as an example when exploring the *; ;* features of WCOP8 IDE. ; * * : ;* K&K Development makes no warranty, representation or *; ;* guarantee regarding the suitability of this project *; ;* for any particular purpose. *; .chip COP888EG .incld main.inc ; Set up memory location COUNT1 as a register so that the ; instruction DRSZ (Decrement and Skip if Zero) can be used ; to test the result. See it used below.) .sect REGISTER, REG COUNT1: .dsb 1 ; .endsect ; This tells this software module that there is a software subroutine in another module called "Subroutine". Not used as part ; ; of the demo. ;.extrn Subroutine ;Subr. from ext. module ; This section of code is given the name "codel". The first line of ; code, a call to a subroutine name "Subroutine" has been commented ; out since it is not used as part of the demonstration. The second ; instruction sets up Timer 1 to produce a rectangular pulse train ; on pin 3 of Port G. Instructions 3 and 4 initialize 16-bit Timer ; T1. The next 4 instructions initialize two 16-bit registers whose ; contents are alternately loaded into Timer each time it counts ; down to 0 and generates an interrupt. If both register were ; reloaded with the same value, a square wave output would be ; produced. (Indicated times are for real timer operation.) .sect code1,rom init: ; JSR Subroutine ;Call subr from ext. module $^{\rm LD}$ CNTRL, #B'10100000 ;PWM Mode, T1A Toggle

Assembly Code Continued

;1 ms, tc = 1 us LD TMR1LO, #L(1000) LDTMR1HI, #H(1000) LD T1RALO, #L(500) ;0.5 ms T1RAHI, #H(500) $^{\rm LD}$ T1RBLO, #L(1000) ;1.0 ms LD LDT1RBHI, #H(1000) RBIT 3, PORTGD ;set up Port G bit 3 SBIT ;as an output 3, PORTGC LD PSW, #B'00010001 ;Enable global and timer q ;interrupts SBIT 4, CNTRL ;Start Timer T1 ; Once started the software will loop through this section until ; ; halted. Operation of the DELAY and TOGGLE subroutines is described ; below. WAIT: DELAY JSR JSR TOGGLE JP WAIT .endsect ;*****Interrupt handler(s)****** ; This section, the timer interrupt handler software, is given the ; name "intr", and is located at address 0xFF. All interrupt ; software for the COP8 family must start at location oxFF. The exclusive OR instruction is used to toggle bit ; 0 of output port ; ; D. The timer pending flag, PSW bit 5, is set whenever a timer ; interrupt occurs, and must be cleared by the interrupt handler. ; Interrupts are disabled whenever an interrupt is detected. RETI is ; a special return instruction which re-enables the interrupts. ; Save the state of the registers before jumping to the Interrupt ; Service Routine ; Note: The COP uses a Vectored Interrupt Structure versus ; a polled interrupt structure .sect intr,rom,abs=0xff .=00FF ; Start at interrupt address ; This is needed to store ; the state of the CPU before ; the "jump" to the ISR Save: PUSH А ; Push Accumulator contents onto ; stack LDA,B PUSH ; Push B pointer onto stack Α LD A,X

Assembly Code Continued

PUSH ; Push X pointer onto stack Α VIS ; Vector to the appropriate ; interrupt routine Restore: ; This is needed to re-store ; the state of the CPU before ; the "jump" to the ISR POP ; Pop X pointer from stack А ; Restore X pointer Х A,X POP ; Pop B pointer from the stack А Х A,B ; Restore B pointer POP ; Restore Accumulator contents А RETI Timer1A_Service: 5,PSW RBIT ;Reset Timer T1A pending flag LD A, PORTD ;Input Port D XOR A,#001 ;Toggle bit 0, 1.5ms ;Output changed port bit A, PORTD Х JP Restore ; These interrupts are not used in ; the program NotUsed: ; They do nothing JP Restore .endsect ;******Vector Table****** ; ; This is the table which corresponds to the ISR(s) above ; There is a typical ISR table in page "3-4 Interrupts" of ; the feature family user's manual ; Make the edit to the table as required .sect Interrupt_TABLE, ROM, ABS=0x1E0 ; Vector Table ; Now Define where the interrupt are going ; be at. We start at location 0x1E0.Addrw NotUsed .Addrw NotUsed .Addrw NotUsed .Addrw NotUsed. .Addrw NotUsed .Addrw Timer1A_Service .Addrw NotUsed .Addrw NotUsed .Addrw NotUsed .Addrw NotUsed

Assembly Code Continued

.endsect

; This section of code is given the name "delay". Register COUNT1 is ; is initialized to a count of $0{\rm xFF}.$ The DRSZ instruction decrements ; COUNT1, and compares the result to zero. If zero the jump \triangleleft ; instruction back to LABEL is skipped and the delay routine is ; exited. .sect delay,rom DELAY: COUNT1,#0FF LD LABEL1: ; Decrement COUNT1, skip if zero DRSZ COUNT1 JP LABEL1 RET .endsect ; This section of code is given the name "toggle", and is placed in ; ROM. The exclusive OR instruction is used to toggle bit $\ensuremath{\mathbf{1}}$; of output port D. The DELAY subroutine inserts a time delay. toggle,rom .sect TOGGLE: LD A, PORTD ;Input Port D XOR A,#002 ;Toggle bit 1 A, PORTD ;Output changed port bit х JSR DELAY ;Time delay RET .endsect .end init