AN-1150 COP8™ FLASH ISP HANDBOOK—Intro to ISP

ABSTRACT

This application note describes the COP8™ In System Programming (ISP) Software. ISP method of programming the flash memory are thoroughly discussed.

Contents

List of Figures

List of Tables

1 Introduction

In-System Programming (ISP) allows the user to re-program a microcontroller without physical removal. The COP8™ ISP Software allows the user to program the flash memory in three ways. A user may choose to program the flash memory by using the Boot ROM's user support portion, the MetaLink™ support portion (via the Flash emulator module) or the MICROWIRE/PLUS™ support portion. The use of a user and/or MICROWIRE/PLUS™ support portions are fully documented and their requirements are specified. Other application notes that relate to the COP8™ FLASH ISP software include AN-1151 (Parallel Port Programming Adapter), AN-1152 (FLASHDOS Programmer Source), AN-1153 (Virtual E^2 Guide), AN-1154 (FLASHWIN Programmer’s Guide) and AN-1161 (FLASHDOS Programmer’s Guide).

2 Introduction to ISP—Software Topics

The Flash Family provides the capability to program the Program Memory while installed in an application board. This feature is called In System Programming (ISP). It provides a means of ISP by using the MICROWIRE/PLUS™, or the user can provide his own, customized ISP routine. This customized routine may use any of the capabilities of the device, such as USART, parallel port, etc. The factory installed ISP uses only the MICROWIRE/PLUS™ port.

2.1 Functional Description

The organization of the ISP feature consists of the user flash program memory, the factory Boot ROM, and some registers dedicated to performing the ISP function. See Figure 1 for a simplified block diagram. The factory installed ISP that uses MICROWIRE/PLUS is located in the Boot ROM. The size of the Boot ROM is 1K bytes and also includes the ICE™ monitor code. If a user chooses to write his own ISP routine, it must be located in the flash program memory.

In the next section, ADVANCED ISP SOFTWARE TOPICS, a discussion regarding the FLEX bit is presented. The FLEX bit controls whether the device exits RESET executing from the flash memory or the Boot ROM. The user must program this Configuration Register bit as appropriate for the application. In the erased state, the FLEX bit = 0 and the device will power-up executing from Boot ROM. When FLEX = 0, this assumes that either the MICROWIRE/PLUS™ ISP routine or external programming is being used to program the device. If using the MICROWIRE/PLUS™ ISP routine, the software in the Boot ROM will monitor the MICROWIRE/PLUS™ for commands to program the flash memory. When programming the flash program memory is complete, the FLEX bit will have to be programmed to a 1 and the device will have to be reset, either by pulling external Reset to ground or by software, before execution from flash program memory will occur.

If FLEX = 1, upon exiting Reset, the device will begin executing from location 0000 in the flash program memory. The assumption here, is that either the application is not using ISP, but is using MICROWIRE/PLUS™ ISP by jumping to it within the application code or is using a customized ISP routine. If a customized ISP routine is being used, then it must be programmed into the flash memory by means of MICROWIRE/PLUS™ ISP or external programming as described in the preceding paragraph.

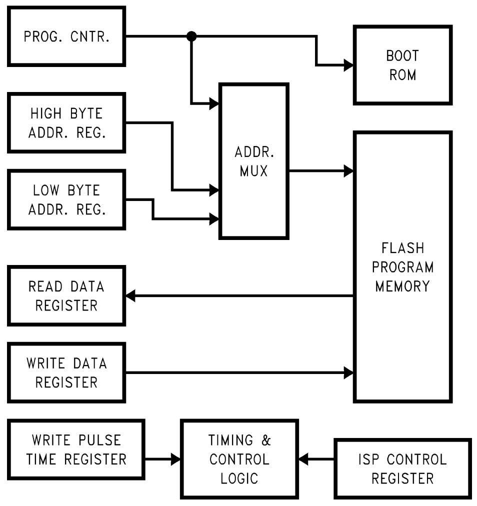

Figure 1. Block Diagram of ISP

2.1.1 REGISTERS

There are six registers required to support ISP: Address Register Hi byte (ISPADHI), Address Register Low byte (ISPADLO), Read Data Register (ISPRD), Write Data Register (ISPWR), Write Timing Register (PGMTIM), and the Control Register (ISPCNTRL).

2.1.1.1 ISP Address Registers

The address registers (ISPADHI and ISPADLO) are used to specify the address of the byte of data being written or read. For page erase operations, the address of the beginning of the page should be loaded. When reading the Option register, FFFF should be placed into the address registers. Registers ISPADHI and ISPADLO are cleared to 00 on Reset. These registers can be loaded from either flash program memory or Boot ROM and must be maintained for the entire duration of the operation.

Table 1. High Byte of ISP Address

ISPADHI							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Addr 15	Addr 14	Addr 13	Addr 12	Addr 11	Addr 10	Addr 9	Addr 8

Table 2. Low Byte of ISP Address

ISPADLO							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Addr 7	Addr 6	Addr 5	Addr 4	Addr 3	Addr 2	Addr 1	Addr 0

2.1.1.2 ISP Read Data Register

The Read Data Register (ISPRD) contains the value read back from a read operation. This register can be accessed from either flash program memory or Boot ROM. This register is undefined on Reset. CAUTION: Read/Modify/Write instructions are not allowed to be used on this register.

Table 3. ISP Read Data Register

ISPRD							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

2.1.1.3 *ISP Write Data Register*

The Write Timing Register (PGMTIM) is used to control the width of the timing pulses for write and erase operations. The value to be written into this register is dependent on the frequency of CKI and is shown in Table 17. This register must be written before any write or erase operation can take place. It only needs to be loaded once, for each value of CKI frequency. If a dedicated E^2 block exists on the device and it's in the process of writing, this register should not be changed until the E^2 write cycle is completed.

Table 4. ISP Write Data Register

ISPWR							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

2.1.1.4 *ISP Write Timing Register*

The Write Timing Register (PGMTIM) is used to control the width of the timing pulses for write and erase operations. The value to be written into this register is dependent on the frequency of CKI and is shown in Table 17. This register must be written before any write or erase operation can take place. It only needs to be loaded once, for each value of CKI frequency. This register can be loaded from either flash program memory or Boot ROM and must be maintained for the entire duration of the operation. If a dedicated E^2 block exists on the device and it's in the process of writing, this register should not be changed until the E^2 write cycle is completed.

2.1.2 MANEUVERING BACK AND FORTH BETWEEN FLASH MEMORY AND BOOT ROM

When using ISP, at some point, it will be necessary to maneuver between the flash program memory and the Boot ROM, even when using customized ISP routines. This is because it's not possible to execute from the flash program memory while it's being programmed.

The JSRB instruction is used to Jump to the Boot ROM. Refer to the COP8™ Flash Family User Manual for specific details on the operation of this instruction. The JSRB instruction must be used in conjunction with the Key register. This is to prevent jumping to the Boot ROM in the event of run-away software. For the JSRB instruction to actually jump to the Boot ROM, the Key bit must be set. This is done by writing the value shown in Table 5 to the Key register. The Key is a 6-bit key and, if the key matches, the KEY bit will be set for 8 instruction cycles. The JSRB instruction must be executed while the KEY bit is set. If the KEY does not match, then the KEY bit will not be set and the JSRB will jump to the specified location in the flash memory. In emulation mode, if a breakpoint is encountered while the KEY is set, the counter that counts the instruction cycles will be frozen until the breakpoint condition is cleared. The Key register is a memory mapped register. Its format when writing is shown in Table 5. Its format when reading is shown in Table 6. In normal operation, it is not necessary to test the KEY bit before using the JSRB instruction. The reading of the Key register is primarily used for testing. Also, located in the Key register is a bit called EFLEX. This bit is also used for testing.

Table 5. Key Register Write Format

KEY when Writing							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
1	0	0	1	1	0	X	X

Bits 7–2:— Key value that must be written to set the KEY bit.

Bits 1–0:—Don't care.

Table 6. Key Register Read Format

KEY when Reading							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0	0	0	0	0	0	EFLEX	KEY
R	R	R	R	R	R	R	R

Bits 7–2:—Read back as 0.

EFLEX (FLASH EXECUTION):—This is the bit that actually controls whether program execution occurs from flash memory or Boot ROM. It uses data from the Option Register bit in combination with other logic controlled by the JSRB instruction, and the G6 hardware override. When EFLEX = 1, execution is from the flash program memory. When EFLEX = 0, program execution occurs from the Boot ROM. This is a Read Only bit.

KEY:— This is the state of the Key. If it is set, it indicates that a valid key was written to the Key register and the JSRB instruction will jump correctly to the Boot ROM. If it's cleared, the key is not valid and the JSRB instruction will jump to the specified address in flash program memory. Once set, the hardware will clear it to 0 after 8 instruction cycles. This is used primarily for testing. This is a Read Only bit.

2.1.3 FORCED EXECUTION FROM BOOT ROM

When the user is developing his own ISP routine, he may encounter code lockups due to mistakes in his software. There is a hardware method to get out of these lockups and force execution from the Boot ROM's MICROWIRE/PLUS routine, so that the customer can erase his flash code and start over. The method to force this condition is to drive the G6 pin to high voltage (2X V_{CC}) and activate Reset. As a note for user of the parallel printer port connects, it is advisable that the user remove the G6 line from the PC when applying high voltage. The voltage may be high enough to permanently damage the PC parallel port logic circuits. The high voltage condition on G6 must be held for at least 3 instruction cycles longer than Reset is active. This special condition will start execution from location 0000 in the Boot ROM where the user can input the appropriate commands, using MICROWIRE/PLUS™, to erase the flash program memory and reprogram it.

2.1.4 RETURN TO FLASH WITHOUT HARDWARE RESET

After programming the entire program memory, including options (and setting the FLEX bit in the Option Register), it is necessary to exit the Boot ROM and return to the flash program memory for program execution. This can be accomplished through the use of the MICROWIRE/PLUS™ ISP Exit command as described later.

2.1.5 MICROWIRE/PLUS ISP COMMANDS

The MICROWIRE/PLUS™ ISP will support the following features and commands:

- Read a byte from a specified address.
- Write a byte from a specified address.
- Erase a page at a specified address.
- Erase the entire flash program memory (mass erase).
- Read multiple bytes starting at a specified address.
- Write multiple bytes starting at a specified address.
- Read Option register.
- Exit ISP by resetting the device and return execution to flash program memory if the FLEX bit is set in the Option Register.

2.1.6 VIRTUAL E^2 COMMANDS

The following commands will support transferring blocks of data from RAM to flash program memory, and vice-versa.

- Erase a page of flash memory at a specified address.
- Copy a block of data from RAM into flash program memory.
- Copy a block of data from program flash memory to RAM.

2.1.7 SAMPLE PROGRAM: *A Light Sequencer.*

Since we have completed our introduction to Flash Family device, lets begin to work on our sample application program.

The goal of this section is to familiarize the user with the following:

1. Writing and saving a program for the COP8™ Flash Family devices
2. Using the MICROWIRE/PLUS™ flash command: Set PGMTIM (write timing register)
3. Using the internal flash command: cwritebf (write a byte to the flash)
4. Using the internal flash command: creadbf (read a byte from the flash)
5. Using the internal/ MICROWIRE/PLUS™ flash command: EXIT (reset the microcontroller).

We will achieve the above commands by using the FLASHWIN Programmer's Guide. See Application Note–1154 for additional information.

2.1.7.1 Description of the Sample Application Program

The goal of the sample program is to teach the user the basics of using the COP8™ flash family devices. The schematic in Figure 2 shows the circuit we are going to use. We will be attaching 8 LEDs, Each LED will be connected in such a way as to sink current from the microcontroller. The cathode (long leg of the LED) will be connected toward the COP8™ Flash Family devices. The short leg of the LED (anode) will be connected through a resistor toward the V$_{CC}$ power supply.

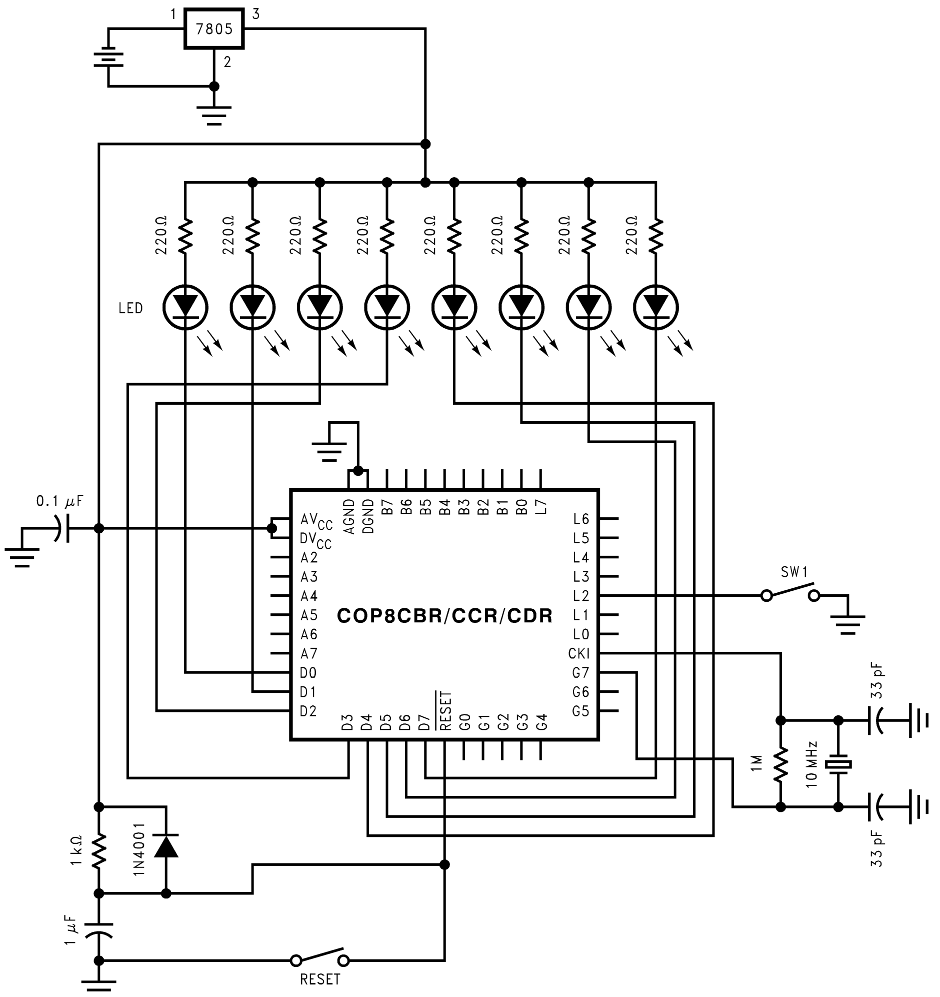

Figure 2. Sample Application Circuit

2.1.7.2 *Writing the Program*

We will begin first by writing the program. Launch the Windows 95 *Editor* by going to the taskbar and clicking on the **Start** button. Then click on **Run**. At the Run dialog ***Open*** window type in the Open field***edit***. Type in exactly as is listed in Example 1. When done, Click on ***File*** and then ***Save***. At the new dialog window type in c:\asm\sequencer.asm (where asm is the directory in which the assembler is installed in). When done, click on ***File*** and then click on ***Exit***.

Example 1. Sample Application Code (sequence.asm)

```
; Program: Sequence.asm
; Purpose: Demonstrate the use of flash routines
; Date: 02/5/00
.INCLD COP8CBR.INC                  ;INCLUDE FILE FOR THE COP8CBR
creadbf = 011                       ; Entry point for the read byte of flash command
cwritebf = 014                      ; Entry point for the write byte of flash command
exit = 062                          ; Entry point for Resetting the microcontroller
.sect data,reg,abs=0xF4             ;FOR RAM STORAGE AREA
LED_BITPOS:.DSB 1                   ;STORAGE FOR THE LED POSITION
DELAY_NUM: .DSB 1                   ;STORAGE FOR THE NUMBER OF DELAYS
.sect code,rom,abs=0                ;BEGINNING CODE SPACE
.org 0                              ;START AT LOCATION 0
MAIN:
```

Example 1. Sample Application Code (sequence.asm) (continued)

```
LD S,#000
RBIT 2,PORTLC                     ;USE PORTL.2 AS AN INPUT
SBIT 2,PORTLD                     ;CAUSE PORTL.2 TO BE AN INPUT WITH PULL-UP
LD PGMTIM,#07B                    ;FOR A 10MHZ CLOCK
LD ISPADHI,#000                   ;LOAD THE HIGH BYTE ADDRESS
LD ISPADLO,#0A0                   ;LOAD THE LOW BYTE ADDRESS
LD ISPWR,#080                     ;FOR LED POSITION 10000000
LD ISPKEY,#098                    ;LOAD THE KEY
JSRB cwritebf                     ;CALL THE ROUTINE
LD ISPADLO,#0A1                   ;LOAD THE LOW BYTE ADDRESS
LD ISPWR,#040                     ;FOR LED POSITION 01000000
LD ISPKEY,#098                    ;LOAD THE KEY
JSRB cwritebf                     ;CALL THE ROUTINE
LD ISPWR,#020                     ;FOR LED POSITION 00100000
LD ISPADLO,#0A2                   ;LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ;LOAD THE KEY
JSRB cwritebf                     ;CALL THE ROUTINE
LD ISPWR,#010                     ;FOR LED POSITION 00010000
LD ISPADLO,#0A3                   ;LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ;LOAD THE KEY
JSRB cwritebf                     ;CALL THE ROUTINE
LD ISPWR,#008                     ; FOR LED POSITION 00001000
LD ISPADLO,#0A4                   ; LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB cwritebf                     ; CALL THE ROUTINE
LD ISPWR,#004                     ; FOR LED POSITION 00000100
LD ISPADLO,#0A5                   ; LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB cwritebf                     ; CALL THE ROUTINE
LD ISPWR,#002                     ; FOR LED POSITION 00000010
LD ISPADLO,#0A6                   ; LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB cwritebf                     ; CALL THE ROUTINE
LD ISPWR,#001                     ; FOR LED POSITION 00000001
LD ISPADLO,#0A7                   ; LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB cwritebf                     ; CALL THE ROUTINE
LOOP:                             ; BEGINNING OF THE LOOP
LD LED_BITPOS,#0A0                ; POSITION IS INITIALIZED TO #0A0
SEQUENCE:                         ; BEGINING OF THE SEQUENCE
JSR DELAY                         ; JUMP TO THE DELAY ROUTINE
LD A,LED_BITPOS                   ; GET THE BIT POSITION
X A,ISPADLO                       ; SWAP IT WITH THE ISP LOW ADDRESS BYTE
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB creadbf                      ; CALL THE ROUTINE
LD A,ISPRD                        ; LOAD THE RESULTS INTO THE ACCUMULATOR
X A,PORTD                         ; SWAP IT WITH PORTD
LD A,LED_BITPOS                   ; LOAD THE LED_BITPOS VARIABLE
                                  ; ROTATE THROUGH THE BIT POSITIONS
INC A                             ; INCREMENT THE ACCUMULATOR
X A,LED_BITPOS                    ; SWAP IT WITH THE LED_BITPOS
IFEQ A,#0A7                       ; STOP AT THE EIGHTH BIT POSITION SHIFT
JP LOOP                           ; RETURNING TO THE MAIN LOOP
JP SEQUENCE                       ; RETURN TO THE MAIL LOOP SEQUENCE
DELAY:
LD DELAY_NUM,#0FF                 ; CREATE 256 NOPS
DELAY_LOOP:                       ; THE DELAY ROUTINE
NOP                               ; CREATE A 1 CYCLE DELAY
NOP                               ; CREATE A 1 CYCLE DELAY
DRSZ DELAY_NUM                    ; COUNT DOWN UNTIL ZERO
JP DELAY_LOOP                     ; OTHERWISE JUST JUMP INTO THE DELAY
RET
IFBIT 2,PORTLP                    ; DETECT IF THE SWITCH IS OFF
JP NEXT2                          ; IF OFF THEN GOTO NEXT COMMAND
```

Example 1. Sample Application Code (sequence.asm) (continued)

```
JP QUIT                           ; OTHERWISE QUIT
NEXT2:                            ; NEXT COMMAND
RET                               ; RETURN FROM THE SUBROUTINE
QUIT:                             ; THIS ROUTINE WILL EXIT WITH AN INTERNAL RESET
LD ISPKEY,#098                    ; LOAD THE KEY
JSRB exit                         ; CALL THE EXIT
.END MAIN                         ; END OF PROGRAM
```

2.1.7.3 Assembling and Linking the Sample Program

Example 2 shows the assembled code. Linking must occur after assembling the code. This can be accomplished via the command: “lncop sequencer”. A hex file named sequencer.hex will be produced as a result of this command. To launch the FLASHWIN application, see the FLASHWIN Programmer’s Guide AN-1154. Click on the Set CLOCK button to set the write timer register. A dialog box will appear. Enter 10 for a 10 MHz CKI frequency. Next, click on the “Upload from a hex file” command. Click on the “Select File” button. Either click on the file or enter it by click on it. Click Ok and the upload process will begin. After the upload is complete perform a write byte to the Configuration Operation. Write a 1 to location 0xFFFF (See Table 8 for additional information) in order to program the microcontroller for execution out of the flash memory. Next click Reboot to exit the MICROWIRE/PLUS ISP routine. Execution will be from the flash array once the reset cycle has completed.

Example 2. The Listing of the Assembled Code (sequence.lis)

```
NSC ASMCOP, Version 5.3 (April 12 17:13
2000) SEQUENCE 13-Sep-00
14:45
PAGE 1
1 ; Program: Sequence.asm
2 ; Purpose: Demonstrate the use of the flash routines
3 ; Date: 02/5/00
4
5 .INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
6
7 0011 creadbf = 011 ; Entry point for the read byte of flash command
8 0014 cwritebf = 014 ; Entry point for the write byte of flash command
9 0062 exit = 062 ; Entry point for Resetting the microcontroller
10
11 00F4 .sect data,reg,abs=0xF4 ; For RAM storage area
12 00F4 LED_BITPOS:.DSB 1 ; Storage for the LED position
13 00F5 DELAY_NUM: .DSB 1 ; Storage for the number of delays
14 0000 .sect code,rom,abs=0 ; Beginning code space
15 0000 .org 0
16
17 0000 MAIN:
18 0000 DF00 3 LD S,#000 ; Use PORTL.2 as an input
19 0002 BDD16A 4 RBIT 2,PORTLC ; Cause PORTL.2 to be an input with pull-up
20 0005 BDD07A 4 SBIT 2,PORTLD ; For a 10MHz clock
21
22 0008 BCA900 3 LD ISPADHI,#000 ; Load the high byte address
23 000B BCA8A0 3 LD ISPADLO,#0A0 ; Load the low byte address
24 000E BCAB80 3 LD ISPWR,#080 ; For LED position 10000000
25 0011 BCE298 3 LD ISPKEY,#098 ; Load the key
26 0014 6114 5 JSRB cwritebf ; Call the routine
27
28 0016 BCA8A1 3 LD ISPADLO,#0A1 ; Load the low byte address
29 0019 BCAB40 3 LD ISPWR,#040 ; for the LED poistion 01000000
30 001C BCE298 3 LD ISPKEY,#098 ; Load the key
31 001F 6114 5 JSRB cwritebf ; Call the routine
32
33 0021 BCAB20 3 LD ISPWR,#020 ; For the LED poistion 00100000
```

Example 2. The Listing of the Assembled Code (sequence.lis) (continued)

```
0024 BCA8A2 3 LD ISPADLO,#0A2 ; Load the low byte address
0027 BCE298 3 LD ISPKEY,#098 ; Load the key
002A 6114 5 JSRB cwritebf ; Call the routine

002C BCAB10 3 LD ISPWR,#010 ; For the LED poistion 00010000
002F BCA8A3 3 LD ISPADLO,#0A3 ; Load the low byte address
0032 BCE298 3 LD ISPKEY,#098 ; Load the key
0035 6114 5 JSRB cwritebf ; Call the routine

0037 BCAB08 3 LD ISPWR,#008 ; For the LED poistion 00001000
003A BCA8A4 3 LD ISPADLO,#0A4 ; Load the low byte address
003D BCE298 3 LD ISPKEY,#098 ; Load the key
0040 6114 5 JSRB cwritebf ; Call the routine

0042 BCAB04 3 LD ISPWR,#004 ; For the LED poistion 00000100
0045 BCA8A5 3 LD ISPADLO,#0A5 ; Load the low byte address
0048 BCE298 3 LD ISPKEY,#098 ; Load the key
004B 6114 5 JSRB cwritebf ; Call the routine

004D BCAB02 3 LD ISPWR,#002 ; For the LED poistion 00000100
0050 BCA8A6 3 LD ISPADLO,#0A6 ; Load the low byte address
0053 BCE298 3 LD ISPKEY,#098 ; Load the key
0056 6114 5 JSRB cwritebf ; Call the routine
NSC ASMCOP, Version 5.3 (April 12 17:13
2000) SEQUENCE 13-Sep-00
14:45
PAGE 2

0058 BCAB01 3 LD ISPWR,#001 ; For the LED poistion 00000100
005B BCA8A7 3 LD ISPADLO,#0A7 ; Load the low byte address
005E BCE298 3 LD ISPKEY,#098 ; Load the key
0061 6114 5 JSRB cwritebf ; Call the routine

0063 LOOP: ; Beginning of the Loop
0063 D4A0 3 LD LED_BITPOS,#0A0 ; Position is initialized to #0A0

0065 SEQUENCE: ; Beginning of the sequence
0065 307D 5 JSR DELAY ; Jump to the delay routine
0067 9DF4 3 LD A,LED_BITPOS ; Get the bit positon
0069 9CA8 3 X A,ISPADLO ; Swap it with the ISP low address byte
006B BCE298 3 LD ISPKEY,#098 ; Load the key
006E 6111 5 JSRB creadbf ; Call the routine
0070 9DAA 3 LD A,ISPRD ; Load the results into the accumulator
0072 9CDC 3 X A,PORTD ; Swap it with PORTD

0074 9DF4 3 LD A,LED_BITPOS ; Load the LED_BITPOS variable

; Rotate through the bit positions
0076 8A 1 INC A ; Increment the accumulator

0077 9CF4 3 X A,LED_BITPOS ; Swap it with the LED_BITPOS

0079 92A7 2 IFEQ A,#0A7 ; Stop at the eighth bit position shift
007B E7 3 JP LOOP ; Returning to the main loop
007C E8 3 JP SEQUENCE ; Return to the main loop sequence

007D DELAY:
007D D5FF 3 LD DELAY_NUM,#0FF ; Create 256 NOPs
007F DELAY_LOOP: ; The delay routine
007F B8 1 NOP ; Create a 1 cycle delay
0080 B8 1 NOP ; Create a 1 cycle delay
0081 C5 3 DRSZ DELAY_NUM ; Count down until zero
0082 FC 3 JP DELAY_LOOP ; Otherwise just jump into the delay
0083 8E 5 RET
```

Example 2. The Listing of the Assembled Code (sequence.lis) (continued)

```
94 0084 BDD272 4 IFBIT 2,PORTLP ; Detect if the switch is off
95 0087 01 3 JP NEXT2 ; If off then goto next command
96 0088 01 3 JP QUIT ; Otherwise quit
97 0089 NEXT2: ; Next command
98 0089 8E 5 RET ; Return from the subroutine
99
100 008A QUIT: ; This routine will exit with an internal reset
101 008A BCE298 3 LD ISPKEY,#098 ; Load the key
102 008D 6162 5 JSRB exit ; Call the exit
103 008F .END MAIN ; End of program
**** Errors: 0, Warnings: 0
Checksum: 0x543B
Byte Count: 0x008F (143)
Input File: c:\nsc\sequence.asm
Output File: c:\nsc\sequence.obj
Memory Model: Large
Chip: 8CBR
```

2.1.7.4 1.8.4 Analysis of the Program

The best way to understand a program is to cut it apart line by line. Lines 14-53 shows how to use the function cwritebf (customer write byte). They also show the correct calling format and usage of the JSRB instruction. Loading the KEY register bit is also shown in the code sample. The code will write the bit patterns listed in Table 7 to the flash memory.

Lines 66-86 makes up the main calling routine. The code will use the creadbf function to read flash memory. Lines 92-104 makes up the delay and detect_exit routine. Lines 106-109 makes up the exit routine. The exit routine calls the reset function and will cause the microcontroller to reset.

The flow to this program is represented in Figure 3.

Table 7. Bit Patterns for the Sample Program

Sequence #	Sequence Pattern							
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	0	0
5	0	0	0	0	1	0	0	0
6	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	1

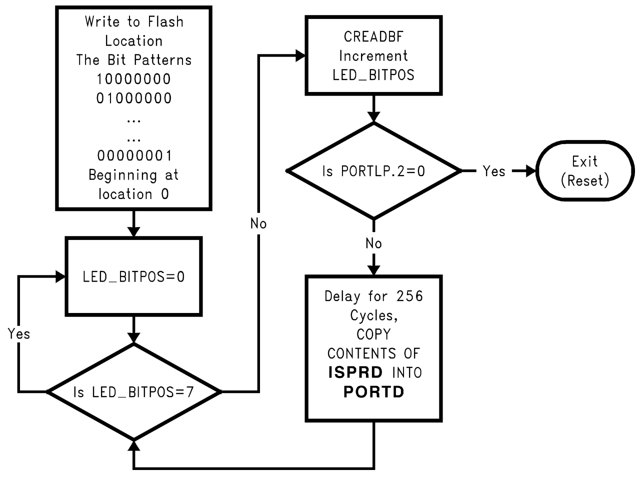
Write to Flash Location The Bit Patterns 10000000 01000000 ... ... 00000001 Beginning at location 0
LED_BITPOS=0
Yes
Is LED_BITPOS=7
No
CREADBF Increment LED_BITPOS
Is PORTLP.2=0
Yes
Exit (Reset)
No
Delay for 256 Cycles, COPY CONTENTS OF ISPRD INTO PORTD

Figure 3. Flow for the Sequencer Program

3 ADVANCED ISP—SOFTWARE TOPICS

3.1 IN SYSTEM PROGRAMMING (ISP) SUPPORT BLOCKS

The *Flash Family's Boot ROM* consists of three main blocks: The *user* support portion, the *MetaLink* support portion and the MICROWIRE/PLUS support portion. Figure 4 shows the relative organization of these support blocks. Each command portion is both independent and self contained. The entire Boot ROM is 1 Kbytes. This document assumes that the reader is fluent in the use of MICROWIRE/PLUS and its transmission protocol. For reference please refer the MICROWIRE/PLUS section of the Flash Family datasheet.

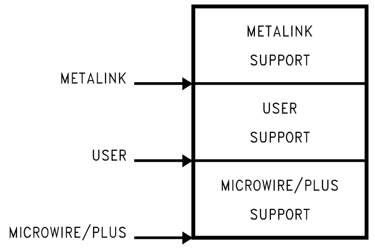

Figure 4. ISP Boot ROM Interface

3.2 Boot ROM Memory Layout

Figure 5 shows how the Boot ROM is organized. FLEX is a hardware bit that controls whether program execution occurs from flash memory of Boot_ROM. It uses data from the Option register. When the FLEX bit option register=1, execution is from the flash program memory. When bit FLEX=0, program execution occurs from the Boot ROM.

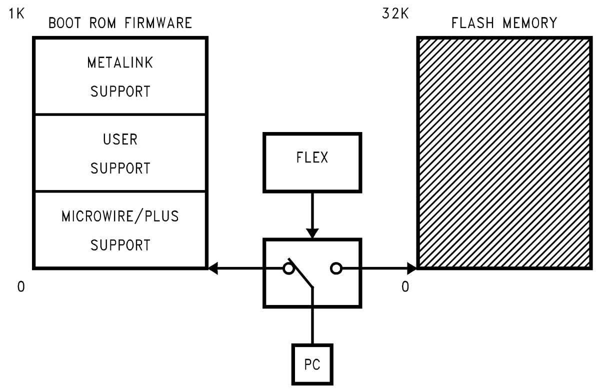

Figure 5. COP8 FLASH Memory Layout

3.3 PROGRAMMABLE OPTIONS DESCRIPTION

The programmable configuration options for this device are listed below.

- Program Memory Security
- Watchdog feature
- Halt Enable feature
- Power-up execution feature

The options will be stored in the highest location in program memory. This location will be called the Option Register. For devices with 32K of Program Memory, the options are stored at location 7FFF. For 16K devices, they will be stored at 3FFF, for 8K devices 1FFF, and for 4K devices 0FFF, however the Option Register can always be accessed by referencing Flash address FFFF. The options are programmed with either external programming or ISP. The location must be erased before programming. The user must not store instructions in the Option register location. If the software tries to execute from the Option register, 00 data will be returned to the instruction register and the device will execute the Software Trap.

3.4 OPTION REGISTER BIT ASSIGNMENTS

The format of the Option Register bit locations are shown in Table 8.

Table 8. Option and TCON Register Bit Assignments

Option Register							
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Reserved	Reserved	SEC	Reserved	Reserved	WD	HALT	FLEX

Bit 7 -— Reserved

Bit 6 -— Reserved

Bit 5 -— SEC - Security bit
=1 Security enabled
=0 Security disabled

Bits 4,3 -— Reserved

Bit 2 -— WD - Watchdog
=1 Watchdog feature is disabled with G1 being a standard I/O.
=0 Watchdog feature is enabled to G1 output with weak pull-up enabled when output is not valid.

Bit 1 -— HALT - Halt Enable
=1 Halt mode is disabled
=0 Halt mode is enabled

Bit 0 -— FLEX - Flash execution
=1 Execute from Flash program memory upon exiting Reset
=0 Execute from Boot ROM upon exiting Reset

3.5 SECURITY

The device has a security feature, when enabled, it prevents reading, writing, and page erases of the flash program memory. Bit-5 in the Option register determines whether security is enabled or disabled. If the security option is disabled, the contents of the internal flash program memory is not protected. If the security feature is enabled;

When executing from user ISP:

1. Reads, writes, page erases, mass erases are all allowed. The user is expected to enforce security within the application code.

When executing from NSC (Boot ROM) ISP or ICE emulation. All writes, reads, and page erases are prohibited.

1. Reads will return FF.
2. Mass erase is permitted. This also erases the Option register.
3. The Option register is readable by reading location FFFF.
4. Reads, writes, page erases are prohibited.

3.6 MICROWIRE/PLUS SUPPORT BLOCKS

3.6.1 2.5.1 Introduction

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

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

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

This device can operate either as a master or a slave, depending on how it is configured by the software. Figure 6 shows an example of how several devices can be connected together using the MICROWIRE/PLUS system, with the device (on the left) operating as the master, and other devices operating as slaves. The protocol for selecting and enabling slave devices is determined by the system designer.

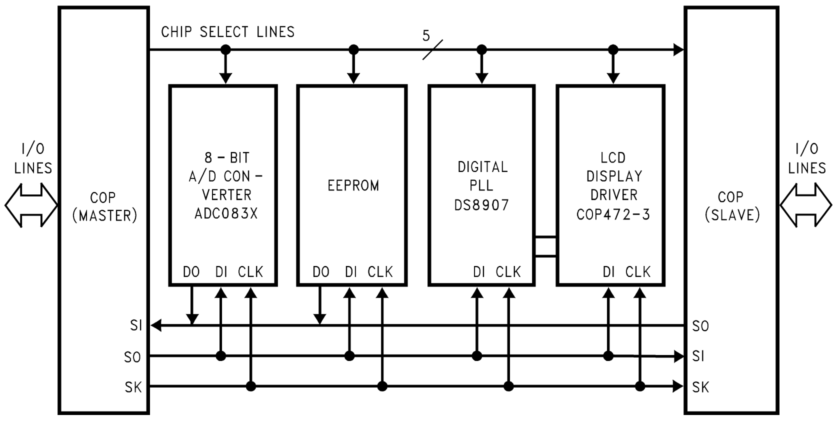

Figure 6. MICROWIRE/PLUS Example

3.6.2 Firmware—MICROWIRE/PLUS Initialization

The MICROWIRE/PLUS support block will initialize the internal communication block with the following parameters: CTRL.MSEL=1, PORTGD.SO=1, PORTGD.SK=1, PORTGC.SI=1, and PORTGC.SK=0. Table 9 and Table 10 contain information about the MICROWIRE/PLUS mode. Figure 7 shows the waveforms that are from the MICROWIRE/PLUS block.

Table 9. Initialization of the MICROWIRE/PLUS by the Firmware

Port G Config. Reg. Bits G5-G4	MICROWIRE/PLUS Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-1	Slave, Data Out and Data In	SO Output	SK Input	SI Input

Table 10. MICROWIRE/PLUS Mode Selected by the Firmware

Port G		SO Clocked Out On:	SI Sampled On:	SK Idle Phase
G6 (SKSEL) Config. Bit	G5 Data Bit			
1	1	SK Falling Edge	SK Rising Edge	High

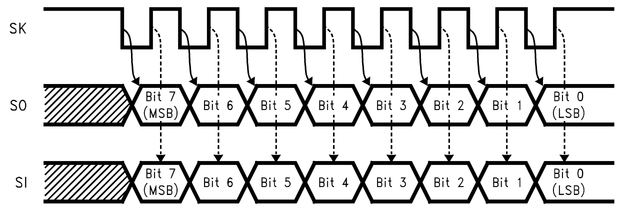

Figure 7. MICROWIRE/PLUS Interface Timing, Normal SK Mode, SK Idle Phase being High

3.7 PC to Boot from MICROWIRE/PLUS Connection Diagram

Figure 8 shows the necessary connections to attach the MICROWIRE/PLUS to the PC's parallel port. The flash microcontroller connection to the PC will be accomplished via a four wire interface.

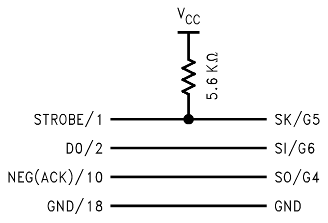

Figure 8. Parallel Port Connection Diagram

Table 11 shows the necessary connections used in the building of the parallel adapter for the COP8 Flash Family microcontroller.

Table 11. Parallel Port <-> MICROWIRE/PLUS Conversion

Parallel Port Printer Port Pin Names	Parallel Printer Port Pin Numbers	MICROWIRE/PLUS Pin Names
STROBE	1	SK/G5
DO	2	SI/G6
NEG(ACK)	10	SO/G4
GND	18	GND

3.8 FIRMWARE—MICROWIRE/PLUS INITIALIZATION

The MICROWIRE/PLUS support block will initialize the internal communication block with the following parameters: CTRL.MSEL=1, PORTGD.SO=1, PORTGD.SK=1, PORTGC.SI=1, and PORTGC.SK=0. Table 9 and Table 10 contains information about the MICROWIRE/PLUS mode. Figure 7 shows the waveforms that are from the MICROWIRE/PLUS block.

Table 12. Initialization of the MICROWIRE/PLUS by the Firmware

Port G Config. Reg. Bits G5-G4	MICROWIRE/PLUS Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
0-1	Slave, Data Out and Data In	SO Output	SK Input	SI Input

Table 13. MICROWIRE/PLUS Mode Selected by the Firmware

Port G		SO Clocked Out On:	SI Sampled On:	SK Idle Phase
G6 (SKSEL) Config. Bit	G5 Data Bit			
1	1	SK Falling Edge	SK Rising Edge	High

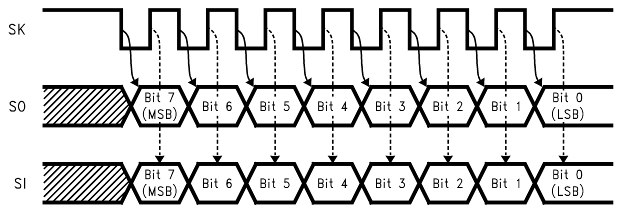

Figure 9. MICROWIRE/PLUS Interface Timing, Normal SK Mode, SK Idle Phase being High

3.8.1 The MICROWIRE/PLUS Packet Composition

A typical MICROWIRE/PLUS packet is composed of a three byte frame (although this varies with the chosen command). Figure 10 is a symbolic representation of the ISP-MICROWIRE/PLUS packet. A trigger byte is a value which will cause a ISP (In System Programming) command to be executed (e.g. erase, read or write a byte of flash). The COMMAND Byte holds this trigger byte value. Refer to Table 16 for valid MICROWIRE/PLUS commands and their trigger byte values. Bytes ADDRESS_HI and ADDRESS_LO refer to the high and low byte address of the flash memory that is to be operated upon. The symbol t_{delay} represents the delay that is required when sending the command, ADDRESS_HI and ADDRESS_LO bytes.

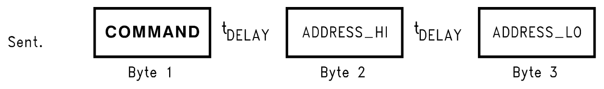

Figure 10. ISP Command Frame

3.8.2 Required Delays In Cascading Microwire Command Frames

A certain amount of delay must be observed when sending multiple command frames in a data stream. The symbol $t_{cascade\text{-}delay}$ represents the delay that is required when sending several commands in a data stream. The host must wait $t_{cascade\text{-}delay}$ cycles before sending the next command frame to the COP8 Flash Family device. Figure 11 shows the delay relationship. Refer to Table 14 for the values of $t_{cascade\text{-}delay}$. Refer to Table 15 for the values of t_{delay}. The symbol $t_1 ... t_N$ denotes individual delay requirements (which varies among different commands).

Table 14. Required time delays (in instruction cycles) for cascading command frames after an initial command was executed

Command	$t_{CASCADE\text{-}DELAY}$
READ_BYTE	6
WRITE_BYTE	6
BLOCKR	13
BLOCKW	6
PGERASE	6
MASS_ERASE	6
EXIT	N/A
PGMTIM_SET	6

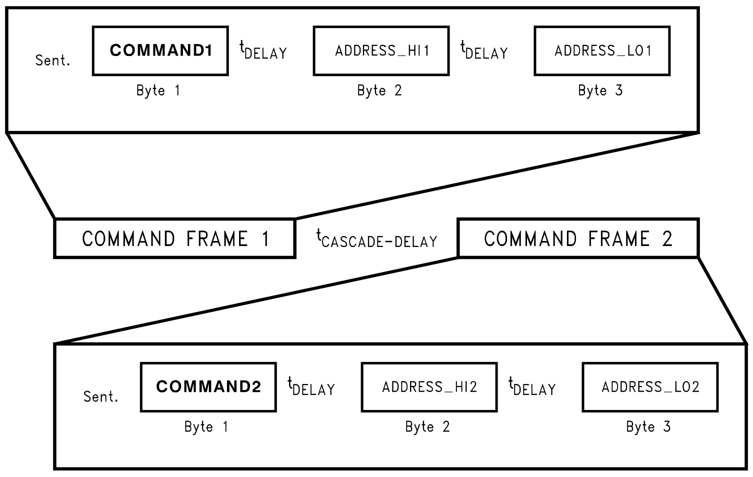

Figure 11. Cascade Delay Requirement

Table 15. Required Time Delays (In Instruction Cycles)

COMMAND	t_1	t_2	t_3	t_4	t_5	t_6	t_N
READ_BYTE	35	100	100	N/A	N/A	N/A	N/A
WRITE_BYTE	35	100	20	10	N/A	N/A	N/A
BLOCKR	35	100	100	100	140	140	140
BLOCKW	35	100	100	100	100	100	52
PGERASE	35	100	100	N/A	N/A	N/A	N/A
MASS_ERASE	25	100	N/A	N/A	N/A	N/A	N/A
EXIT	N/A	N/A	N/A	N/A	N/A	N/A	N/A
PGMTIM_SET	35	35	N/A	N/A	N/A	N/A	N/A

3.8.3 Variable Host Delay

A special type of communication has been implemented in the device firmware in order to allow the microcontroller enough time to complete a write or erase operation. This type of communication was developed since the microcontroller may be used in situations where the clock is extremely slow and writes to the flash memory will take a large amount of time. This implementation relieves the user of having to manually change the write delays in their host software. Figure 12 shows how the VARIABLE HOST DELAY configuration is implemented on a byte write. Figure 13 shows how the VARIABLE HOST DELAY configuration is implemented on a block write. Figure 14 shows how the VARIABLE HOST DELAY configuration is implemented on a page erase. Figure 15 shows how the VARIABLE HOST DELAY configuration is implemented on a mass erase. Since the SK (Serial CLOCK) is normally high, the microcontroller brings SK low to indicate to the host that a WAIT condition (i.e. the SK pin is low) exists. The host then goes into a loop until the WAIT condition changes to a READY condition (i.e., the SK pin is high again). The controller then returns to command decode and waits for the next command.

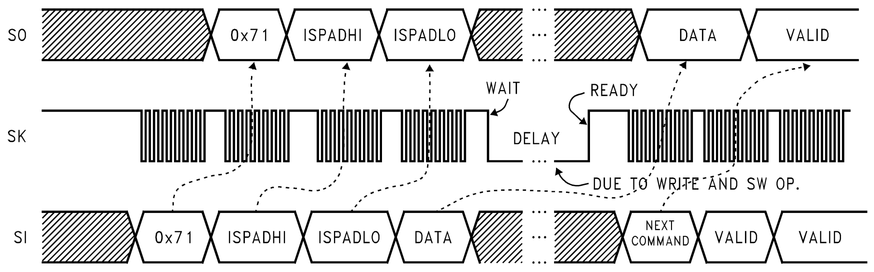

Figure 12. Byte Write Waveform (Relative Bytes are Shown)

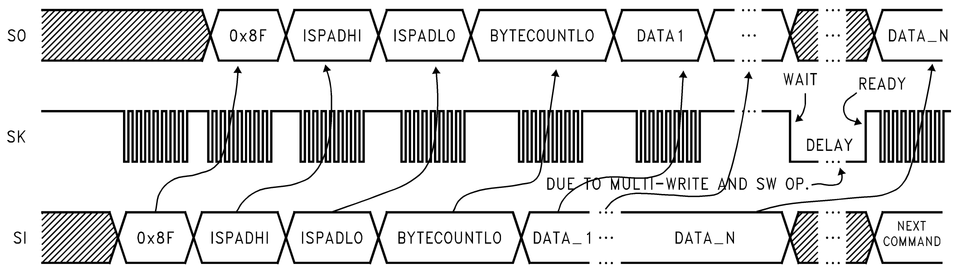

Figure 13. Block Write Waveform (Relative Bytes are Shown)

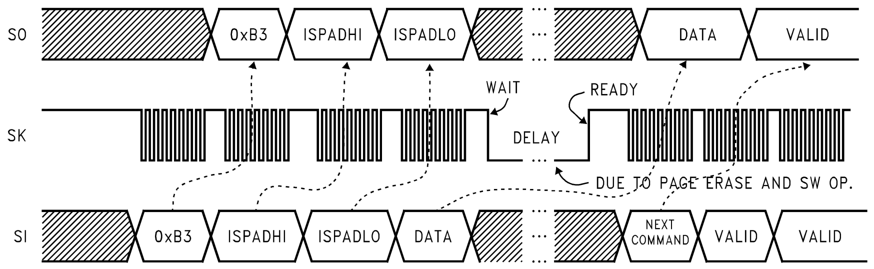

Figure 14. Page Erase Waveform (Relative Bytes are Shown)

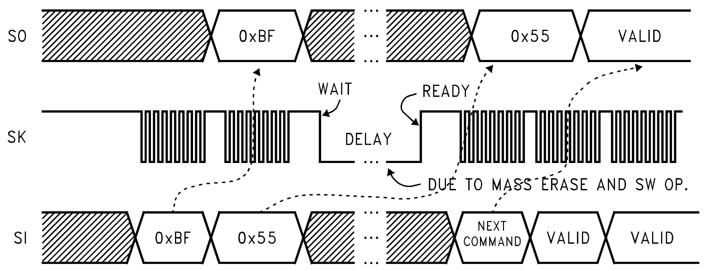

Figure 15. Mass Erase Waveform (Relative Bytes are Shown)

3.8.4 MICROWIRE/PLUS—Boot ROM Startup Behavior

Upon start-up the ISP Boot ROM will detect if the G6 pin is high. This is used to detect if a high voltage condition on the G6 pin is present (i.e., a forced Boot ROM re-entry due to code lockup, for additional information refer to Section 2.1.3.) By using this technique the Boot ROM avoids any bit that may be inadvertently entered on to the SI pin. If the G6 pin is not high at start-up, the ISP Boot ROM will try to detect if a valid command is received on a transmission. If a valid command is received, the Boot ROM firmware will check to see if the **SECURITY** bit is set. Table 16 shows the valid MICROWIRE/PLUS commands. If security is set, the Boot ROM will disable all ISP functions except the reading of the **OPTION** register at 0xFFFF, the execution of a mass erase on the flash memory and the setting of the PGMTIM Register. Read attempts of flash memory, other than location 0xFFFF, Option Register, while security is set, will result with a 0xFF sent back through the MICROWIRE/PLUS. In general, the Boot ROM firmware will decode the command, check security, execute the command (if security is off) and execute the MICROWIRE/PLUS Main Support Block (e.g., triggering the PSW.BUSY bit in order to send the data back to the host.) See Figure 16 for the ISP—MICROWIRE/PLUS Control flow.

Table 16. MICROWIRE/PLUS Commands

Command	Function	Byte Value	Parameters	Variable Host Delay Implemented?	Return Data
PGMTIM_SET	Write Pulse Timing Register	0x3B	Value	No	N/A
PAGE_ERASE	Page Erase	0xB3	Starting Address of Page	Yes	N/A
MASS_ERASE	Mass Erase	0xBF	Confirmation Code	Yes	N/A (The entire Flash Memory will be erased)
READ_BYTE	Read Byte	0x1D	Address High, Address Low	No	Data Byte if Security not set. 0xFF if Security set.
BLOCKR	Block Read	0xA3	Address High, Address Low, Byte Count (n) High, Byte Count (n) Low (0 ≤ n ≤ 32767)	No	n Data Bytes if Security not set. n Bytes of 0xFF if Security set
WRITE_BYTE	Write Byte	0x71	Address High, Address Low, Data Byte	Yes	N/A

Table 16. MICROWIRE/PLUS Commands (continued)

Command	Function	Byte Value	Parameters	Variable Host Delay Implemented?	Return Data
BLOCKW	Block Write	0x8F	Address High, Address Low, Byte Count (0 ≤ n ≤ 16), n Data Bytes Data location must be within a 64 byte segment for a 32k device, 32 byte for 1k and 4k devices (1/2 page) due to multi-byte write limitation	Yes	N/A
EXIT	EXIT	0xD3	N/A	No	N/A (Device will Reset)
INVALID	N/A		Any other invalid command will be ignored	N/A	N/A

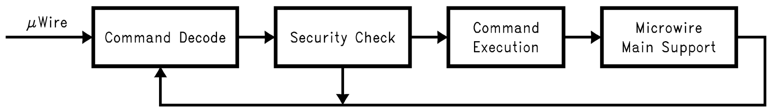

Figure 16. The ISP—MICROWIRE Control

3.9 MICROWIRE COMMANDS AVAILABLE

3.9.1 PGMTIM_SET

Sets the flash write timing register to match that of the CKI frequency. See Table 17 for values.

Description: Figure 17 shows the format of the PGMTIM_SET command. The PGMTIM_SET command will transfer the next byte sent into the flash programming time register. No acknowledgment will be sent. The symbol t_1 denotes the time delay between the command byte and the setting of the PGMTIM register. This command is always available. This command must be used before any “writes” can occur (i.e., page erase, mass erase, write byte or block write). See Table 21 for the value(s) of t_1 and t_2. Table 17 shows valid values for the PGMTIM register. This command is security independent.

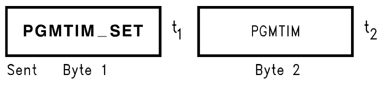

Figure 17. The Set PGMTIM Command

Table 17. Valid PGMTIM Values

Bit Values for the PGMTIM Register								Hex Value	CKI Frequency Range
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	1	0x01	25 kHz–33.3 kHz
0	0	0	0	0	0	1	0	0x02	37.5 kHz–50 kHz
0	0	0	0	0	0	1	1	0x03	50 kHz–66.67 kHz
0	0	0	0	0	1	0	0	0x04	62.5 kHz–83.3 kHz
0	0	0	0	0	1	0	1	0x05	75 kHz–100 kHz
0	0	0	0	0	1	1	1	0x07	100 kHz–133 kHz
0	0	0	0	1	0	0	0	0x08	112.5 kHz–150 kHz
0	0	0	0	1	0	1	1	0x0B	150 kHz–200 kHz
0	0	0	0	1	1	1	1	0x0F	200 kHz–266.67 kHz
0	0	0	1	0	0	0	1	0x11	225 kHz–300 kHz
0	0	0	1	0	1	1	1	0x17	300 kHz–400 kHz
0	0	0	1	1	1	0	1	0x1D	375 kHz–500 kHz

Table 17. Valid PGMTIM Values (continued)

Bit Values for the PGMTIM Register								Hex Value	CKI Frequency Range
7	6	5	4	3	2	1	0		
0	0	1	0	0	1	1	1	0x39	500 kHz–666.67 kHz
0	0	1	0	1	1	1	1	0x2F	600 kHz–800 kHz
0	0	1	1	1	1	1	1	0x3F	800 kHz–1.067 MHz
0	1	0	0	0	1	1	1	0x47	1 MHz–1.33 MHz
0	1	0	0	1	0	0	0	0x48	1.125 MHz–1.5 MHz
0	1	0	0	1	0	1	1	0x4B	1.5 MHz–2 MHz
0	1	0	0	1	1	1	1	0x4F	2 MHz–2.67 MHz
0	1	0	1	0	1	0	0	0x54	2.625 MHz–3.5 MHz
0	1	0	1	1	0	1	1	0x5B	3.5 MHz–4.67 MHz
0	1	1	0	0	0	1	1	0x63	4.5 MHz–6 MHz
0	1	1	0	1	1	1	1	0x6F	6 MHz–8 MHz
0	1	1	1	1	0	1	1	0x7B	7.5 MHz–10 MHz
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W		

3.9.2 PAGE_ERASE—Erase a Page of Flash Memory

Description: Figure 18 shows the format of the PAGE_ERASE command. The PAGE_ERASE command will erase a 128 bytes page (depends on the array size, 64 bytes for devices containing 1k and 4k) from the flash memory. The next two bytes after the PAGE_ERASE byte refer to the beginning high and low bytes of the beginning address of the target flash page. A WAIT/READY technique is used to delay the host when the controller is executing and writing to the flash memory. For a full description of the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (Section 3.8.3). The symbol t_1, t_2 denote the time delay between the command byte, the delay required after loading the high address byte, and the delay after loading of the low address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. The PAGE_ERASE command is **NOT** always available (i.e., it is security dependent). If security is set, then the command will be aborted and no acknowledgment will be sent back. See section 10.4 for details on the number of endurance cycles and the number of page erase commands that should be issued prior to writing data into the erased page. See Table 15 for the value(s) of t_1, t_2, and t_3.

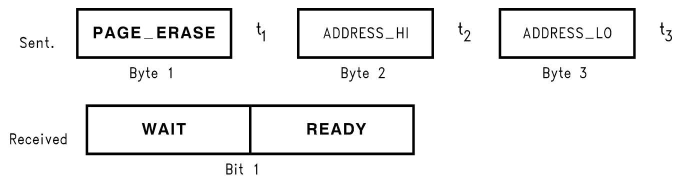

Figure 18. The PAGE ERASE Command

3.9.3 MASS_ERASE—Erase the Entire Flash Memory Array

Description: Figure 19 shows the format of the MASS_ERASE command. The MASS_ERASE command will erase the entire flash memory, including the Option Register. The next byte after the MASS_ERASE command refers to the confirmation key used to double check that a mass erase request was actually sent. The confirmation key must equal 0x55 in order for the MASS_ERASE command to continue. The symbol t_1 denotes the time delay between the command byte and the transmission of the CONFIRM_KEY. The symbol t_2 denotes the time delay after the CONFIRM_KEY has been checked. A WAIT/READY technique is used to delay the host when the controller is executing and writing to the flash memory. For a full description regarding the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (Section 3.8.3). The MASS_ERASE command is always available. It is security independent. See Table 15 for the value(s) of t_1, and t_2.

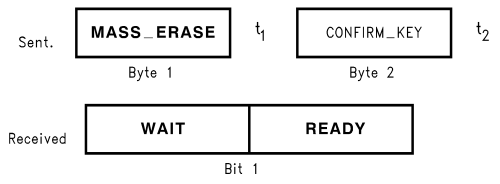

Figure 19. The MASS_ERASE Command

3.9.4 READ_BYTE—Read a Byte from the Flash Memory Array

Description: Figure 20 shows the format of the READ_BYTE command. The READ_BYTE command will read a byte from the flash memory. The next two bytes after the READ_BYTE refer to the address of the target flash location. The symbol t_1, t_2 denotes the time delay between the command byte, the delay after loading of the high address byte. Data is sent back after t_3 delay(s) has elapsed. If security is set, the user is only allowed to read location 0xFFFF (Option Register). In other words, if security is set and ADDRESS_HI and ADDRESS_LO=0xFFFF then the firmware will allow that operation, otherwise it will send back a 0xFF in the DATA_RTN byte. See Table 15 for the value(s) of t_1, t_2, and t_3.

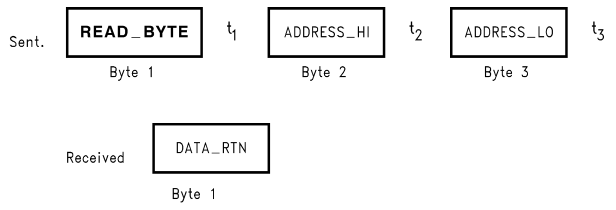

Figure 20. The READ_BYTE Command

3.9.5 WRITE_BYTE—Write a Byte to the Flash Memory Array

Description: Figure 21 shows the format of the WRITE_BYTE routine. The WRITE_BYTE command will write a byte to the flash memory. The next two bytes after the WRITE_BYTE byte refer to the high and low byte address of the target flash location. The next byte (DATA_REC) after the ADDRESS_LO byte will contain the value that will be stored into the flash location. The symbols t_1, t_2 denote the time delay between the command byte and the delay after loading of the high address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. Data is saved into the flash location after a t_4 delay. A WAIT/READY signal is used to delay the host. For a full description of the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (Section 3.8.3). The WRITE_BYTE command is **NOT** always available (i.e. it is security dependent.) If security is set, then the command will be aborted and no acknowledgment will be sent back. See Table 15 for the value(s) of t_1, t_2, t_3, and t_4.

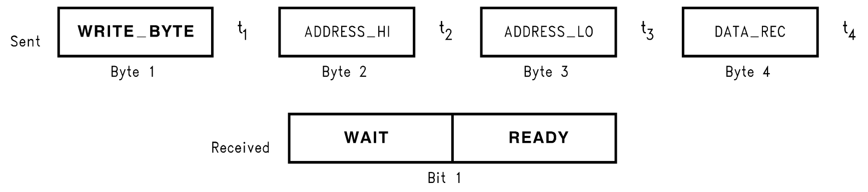

Figure 21. The WRITE_BYTE Command

3.9.6 BLOCK WRITE—Write a Block of Data to the Flash Memory Array

Description: Figure 22 is a symbolic representation of the BLOCK_WRITE routine. Data is written in sequential order. This routine is intended to write bytes of data which will reside in a page of flash memory. The next two bytes after the BLOCK_WRITE byte refer to the beginning high and low byte address of the target flash location. The next byte after the ADDRESS_LO byte refers to the BYTECOUNTLO variable. The BYTECOUNTLO variable is used by the microcontroller to transfer N bytes (i.e, N=BYTECOUNTLO). The maximum number of bytes that can be written is 16. If the number of bytes exceeds 16, it may not be guaranteed that all of the bytes were written. The data cannot cross page boundaries. Data must be placed with-in the same 1/2 page segment, 64 bytes for 32k devices and 32 bytes for 1k and 4k devices. This is due to the multi-byte write limitation. If N=0 then the firmware will abort. The symbols t_1 and t_2 denotes the time delay between the command byte and the delay after loading of the high address byte. The symbol t_3 denotes the time delay after loading the ADDRESS_LO value. The symbol t_4 denotes the necessary time delay after loading the BYTECOUNTLO variable. Data arrives at t_5 cycles after the ADDRESS_LO value is loaded (i.e. DATA1 - DATA2 have the same delay as DATA2 - DATA3). After the last byte (DATA_N) is received, a WAIT/ READY signal will be sent to delay the host. For a full description of the WAIT/READY command refer to the section regarding **VARIABLE HOST DELAY** (Section 3.8.3). The command (BLOCK_WRITE) is **NOT** always available (i.e. it is security dependent). If security is set, then the command will be aborted after the last data (DATA_N) is received and no acknowledgment will be sent back. See Table 15 for the value(s) of t_1, t_2, t_3, t_4, t_5, and t_6.

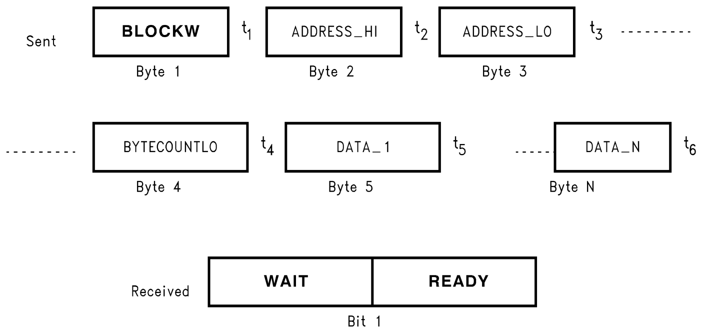

Figure 22. The Block Write Routine

3.9.7 BLOCK_READ—Read a Block from the Flash Memory Array

Description: Figure 23 shows the format of the BLOCK_READ command. The BLOCK_READ command will read multiple bytes from the flash memory. The next two bytes after the BLOCK_READ byte refer to the beginning high and low byte address of the target flash location. The next two bytes after the ADDRESS_LO byte refer to the upper and lower byte of BYTECOUNT. The BYTECOUNT variable is used by the microcontroller to send back N number of bytes (i.e, N=BYTECOUNT). The maximum value of N is 32 kBytes. If N=0 then the firmware will abort. The symbols t_1, $t_2$and , t_3 denotes the time delay between the command byte, the delay in loading of the ADDRESS_HI, and the delay after loading the ADDRESS_LO. The symbol t_4 denotes the required time delay between loading BYTECOUNTHI and BYTECOUNTLO. Subsequent data is sent to the host at t_5 cycles after BYTECOUNTLO (i.e. DATA1–DATA2 have the same delay as DATA2–DATA3). This command is capable of sending up to 32 kB of flash memory through the MICROWIRE/PLUS. This command is always available however, if security is set, the user is only allowed to read 0xFFFF (Option Register). In other words, if at anytime ADDRESS_HI and ADDRESS_LO=0xFFFF, the firmware will allow that operation. If at any time ADDRESSHI and ADDRESS_LO do not equal 0xFFFF and security is set, then the firmware will return 0xFF. This routine will acknowledge by returning data to the host.

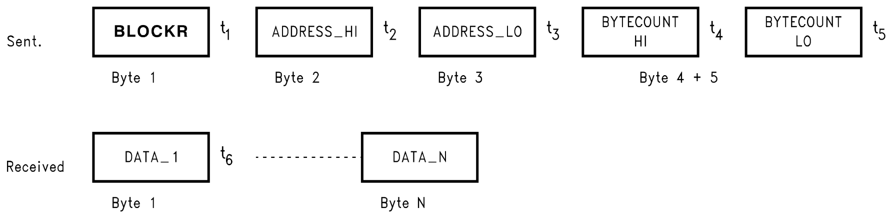

Figure 23. The Block Read Command

3.9.8 EXIT—Reset the Microcontroller

Description: Figure 24 shows the format of the EXIT command. The EXIT command will reset the microcontroller. There is no additional information required after the EXIT byte is received. No acknowledgment will be sent back regarding the operation. This command is always available. It is security independent.

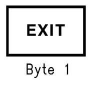

Figure 24. The EXIT Command

4 ISP DOWNLOADER

4.1 *IMPLEMENTATION EXAMPLE—BUILDING A FIRMWARE MODIFIER/DOWNLOADER*

The following section deals with construction of a portable downloader. Several microcontrollers exist which have the MICROWIRE/PLUS compatible peripheral built in. National's COP8SGx line of microcontrollers are used to demonstrate the construction of a portable downloader.

National's COP8SGx microcontrollers are easily interfaced to the COP8 Flash Family microcontrollers. Communication is established via the built-in MICROWIRE/PLUS peripheral block. A 3 + GND wire setup is used. Code samples are provided and documented procedures are given. Figure 25 shows how to interface the COP8SGR to the COP8 Flash Family devices. The 100Ω resistor is used to protect both devices from bus contention. The 5.6 kΩ pull-up resistor is used by the firmware to detect an idle condition on the bus.

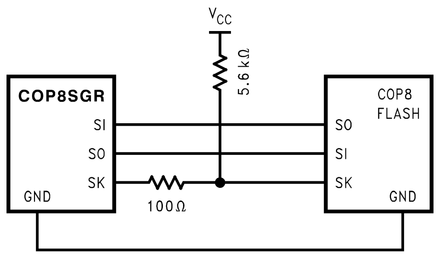

Figure 25. Interfacing the COP8SGR and COP8CBR Microcontrollers

4.1.1 COP8SGR Initialization Routine

The COP8SGR microcontroller must initialize the internal communication block with the following parameters: CTRL.MSEL=1,PORTGD.SO=1,PORTGD.SK=1,PORTGC.SI=1, and PORTGC.SK=1.Table 18 andTable 19 contain information about the MICROWIRE/PLUS mode. Figure 7 shows the waveforms from the MICROWIRE/PLUS block. Figure 26 shows the flow for the initialization routine.

Table 18. Required Initialization of the MICROWIRE/PLUS

Port G Config. Reg. Bits G5-G4	MICROWIRE/PLUS Operation	G4 Pin Function	G5 Pin Function	G6 Pin Function
1-1	Master, Data Out and Data In	SO Output	SK Input	SI Input

Example 3 Example 4 shows the sample assembly and C source for the routine.

Table 19. MICROWIRE/PLUS Mode Required for Communication

Port G		SO Clocked Out On:	SI Sampled On:	SK Idle Phase
G6 (SKSEL) Config. Bit	G5 Data Bit			
1	1	SK Falling Edge	SK Rising Edge	High

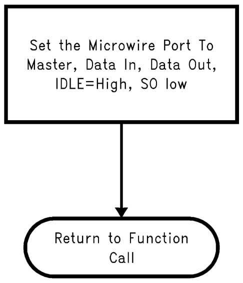
Set the Microwire Port To Master, Data In, Data Out, IDLE=High, SO low
Return to Function Call

Figure 26. Flow Chart for the Initialization Routine

Example 3. Required Initialization Routine—Assembly Version

```
; MICROWIRE/PLUS COP8SGR Initialization Routine
; Assume That The Wire Are Connected As In Figure 29
.INCLD cop8sgr.INC                ; INCLUDE FILE FOR THE COP8SGR
.sect     code,rom,abs=0          ; BEGINING CODE SPACE
                                  ; Main Routine
MAIN:
          jsr MICROINIT           ; CALL THE ROUTINE
          jp MAIN                 ; RETURN TO MAIN
                                  ;INITIALIZATION CODE
MICROINIT:
          sbit MSEL,CNTRL         ; CNTRL.MSEL= 1, SET MICROWIRE INTO
          sbit SK,PORTGC          ; PORTGC.SK = 1, MODE, DATA OUT,
          sbit SO,PORTGC          ; PORTGC.SO = 1 ,DATA IN
          rbit SO,PORTGD          ; PORTGD.SO = 0 to let the firmware know that
                                  ; you want to go into ISP Mode
                                  ; Set MICROWIRE/PLUS into standard sk mode,
                                  ; IDLE = High, SET ACCORDING TO Table 20
          sbit SK,PORTGD          ; PORTGD.SK=1
          sbit SI,PORTGC          ; PORTGC.SI = 1
          ret                     ; RETURN FROM THE CALL
.END MAIN                         ; END OF PROGRAM
```

Example 4. Required Initialization Routine—C Version

```
#include ²8sgr.h²;       // Include file for the COP8SGR Microcontroller
void microinit(void);           // The MICROWIRE/PLUS initialization routine
void main(){                    // The main
microinit();                    // Set up MICROWIRE/PLUS for CBR Xmission
while(1);                       // Endless loop
}                               // End of the main

void microinit(){               // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                   // The MICROWIRE/PLUS Control Select Bit is set.
PORTGC.SK=1;                    // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                    // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                    // To tell the firmware that you want to go into
                                // ISP Mode
                                // Set MICROWIRE/PLUS into standard sk mode, IDLE=High, Set According
                                // to Table 2
PORTGD.SK=1;                    // Set IDLE MODE=High
PORTGC.SI=1;                    // Set the configuration bit
}                               // End of microinit routine
```

4.1.2 The PGMTIM_SET Routine

Sets the flash write timing register to match that of the CKI frequency. See Table 11.5 for values. Example 5 shows the flow of the PGMTIM_SET routine. Example 5 Example 6 shows the assembly and C version of the routine.

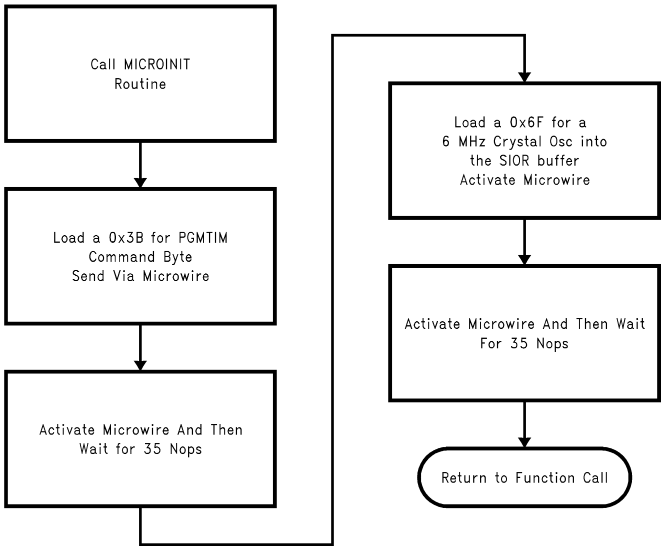
Call MICROINIT Routine
Load a 0x3B for PGMTIM Command Byte Send Via Microwire
Activate Microwire And Then Wait for 35 Nops
Load a 0x6F for a 6 MHz Crystal Osc into the SIOR buffer Activate Microwire
Activate Microwire And Then Wait For 35 Nops
Return to Function Call

Figure 27. Flow for the PGMTIM_SET Program

Example 5. Code Sample For PGMTIM_SET. Assembly Version: COP8SGR

```
; MICROWIRE/PLUS COP8SGR Write PGMTIME routine
; Assume That The Wire Are Connected As In Figure 29
               .INCLD cop8sgr.INC              ;INCLUDE FILE FOR THE COP8SGR
               .sect data,reg,abs=0xF0         ;FOR RAM STORAGE AREA
DELAY_NUM: .DSB 1                              ;STORAGE FOR THE NUMBER OF DELAYS
               .sect code,rom,abs=0            ;BEGINING CODE SPACE
MAIN:
                jsr MICROINIT                  ;CALL THE MICROWIRE INITIALIZATION ROUTINE
                jsr PGMTIM_SET                 ;CALL THE SET PROGRAMMING TIMING
                jp MAIN                        ;RETURN TO THE MAIN LOOP
PGMTIM_SET: ;THE SET WRITE TIMING ROUTINE
                ld SIOR,#03B                   ;PGMTIM_SET COMMAND Byte
                jsr MICROWIRE_SEND             ;Send the command byte out
                ld A,#023                      ;The amount of delay cycles required, For more info
                                               ;see Table 18 regarding required time delay cycles
                jsr DELAY                      ;The delay routine
                ld SIOR,# 06F                  ;Send the Write Time - Assume a 10MHz CKI CBR CKI
                                               ;frequency, See Table 2-13 for additional information
                jsr MICROWIRE_SEND             ;Send the value out to the COP8CBR
                ret                            ;end of the PGMTIM_SET routine
MICROWIRE_SEND:                                ;The MICROWIRE/PLUS send routine
                sbit BUSY,PSW                  ;SET THE PSW.BUSY BIT TO TURN ON
                ld B,#PSW                      ;while (PSW.BUSY)
wait_uwire:                                    ;THE MICROWIRE WAIT ROUTINE
                ifbit 02,[B]                   ;IF THE BIT IS ON THEN WAIT
                jp wait_uwire                  ;Otherwise stay in the loop
                ret                            ;RETURN FROM THE FUNCTION CALL
MICROINIT:
                sbit MSEL,CNTRL                ;CNTRL.MSEL= 1, SET MICROWIRE INTO
                sbit SK,PORTGC                 ;PORTGC.SK = 1, MODE, DATA OUT,
                sbit SO,PORTGC                 ;PORTGC.SO = 1, DATA IN
                rbit SO,PORTGD                 ;PORTGD.SO = 0 to let the firmware know that
                                               ;you want to go into ISP Mode
```

Example 5. Code Sample For PGMTIM_SET. Assembly Version: COP8SGR (continued)

```
; Set MICROWIRE/PLUS into standard sk mode, sk high during idle
; IDLE = High, SET ACCORDING TO Table 20
                sbit SK,PORTGD                ;PORTGD.SK=1
                sbit SI,PORTGC                ;PORTGC.SI=1
                ret                           ;RETURN FROM THE CALL
DELAY:                                        ;THE DELAY ROUTINE
; ASSUME THE AMOUNT OF NOPS IS STORED IN
; THE ACCUMULATOR A
                LD DELAY_NUM,#023             ;Corresponds to 35 cycles
LOOP_POINT:                                   ;POINT WHERE THE LOOP ACTUALLY OCCURS
                NOP                           ;THE ACTUAL NOPS
                drsz DELAY_NUM                ;DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
                jp LOOP_POINT                 ;ZERO
                ret                           ;RETURN FROM TO THE FUNCTION CALL
                .END MAIN                     ;END OF PROGRAM
```

Example 6. The PGMTIM_Set Routine—C Version

```
#include "8sgr.h";                  // Include file for the COP8SGR Microcontroller
void pgmtim_set(void);                     // The pgmtim_set routine
void delay(unsigned int delay_num);        // The actual num of delays
void microwire_send(void);
void microinit(void);
void main(){                               // The main
microinit();                               // Initialize the MICROWIRE/PLUS port
pgmtim_set(0x7B);                          // For a 10 MHZ CKI Frequency
delay(6);                                  // Just in case of cascading
while(1);                                  // Endless loop
}
void pgmtim_set(unsigned int frequency){   // The PGMTIM_SET
SIOR=0x3B;                                 // Routine, Send out the PGMTIM_SET command byte
microwire_send();                          // Start up the MICROWIRE/PLUS and send the byte
delay(35);                                 // Wait for 35 NOPs as required in the time delay
req.
SIOR=frequency;                            // Now send out the frequency
microwire_send();                          // Start out the MICROWIRE/PLUS
}                                          // End of the routine
void microwire_send(void){                 // The routine that starts the microwire
CNTRL.MSEL=1;                              // Set the MICROWIRE/PLUS Control Bit
PSW.BUSY=1;                                // Set the PSW busy bit
while(PSW.BUSY);                           // Wait until the bit has cleared
}                                          // end of the routine
void delay(unsigned int delay_num){        // The delay routine
unsigned int i;                            // temp variable
for (i=0;i<delay_num; i++)                 // The loop control
NOP;                                       // Wait on NOP
}                                          // End of the delay routine
void microinit(){                          // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                              // The MICROWIRE/PLUS Control Select Bit is set.
PORTGC.SK=1;                               // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                               // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                               // To tell the firmware that you want to go
                                           // into ISP Mode
                                           // Set MICROWIRE/PLUS into standard sk mode,
IDLE=High,
                                           // Set According to Table 2
PORTGD.SK=1;                               // Set IDLE MODE=High
PORTGC.SI=1;                               // Set the configuration bit
}                                          // End of microinit routine
```

4.1.3 PAGE_ERASE—Erase a Page of Flash Memory

Figure 28 shows the flow for the PAGE_ERASE routine. Example 7 and Example 8 shows the assembly and C code versions for the routines.

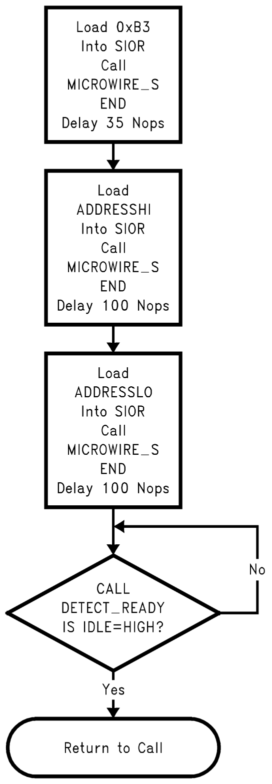

Figure 28. Flow for the Page Erase Function

Example 7. The Page Erase Routine—Assembly Version: COP8SGR

```
                                           ; ERASE A PAGE FROM FLASH
.INCLD cop8sgr.INC                                 ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0;
DELAY_NUM .DSB 1                                   ;Allocate some memory
ADDRESSHI .DSB 1                                   ;To Hold the Upper Byte
ADDRESSLO .DSB 1                                   ;To Hold the Lower Byte
.SECT CODE,ROM,ABS=0                               ;Beginning of the code
MAIN:                                              ;Beginning of the main
jsr MICROINIT                                      ;Call the initialization routine
jsr PGMTIM_SET                                     ;Set the PGMTIM_SET routine
ld ADDRESSHI,#000                                  ;Addresshi=0
ld ADDRESSLO,#000                                  ;Addresslo=0
jsr PAGE_ERASE                                     ;Call the erase page routine
ld A,#006                                          ;Create a delay of at least 6 NOPS
jsr DELAY                                          ;Jump to the delay routine
jp MAIN                                            ;Return to the main function
PAGE_ERASE:                                        ;The erase function
ld SIOR,#0B3                                       ;The command byte for page erase
jsr MICROWIRE_SEND                                 ;Start the MICROWIRE/PLUS up
ld A,#023                                          ;Wait at least 35 cycles
jsr DELAY                                          ;Call the delay routine
ld A,ADDRESSHI                                     ;Load the low address into the SIOR register
x A,SIOR                                           ;Do the swap here
jsr MICROWIRE_SEND                                 ;Start the MICROWIRE/PLUS up
ld A,#064                                          ;Create a delay of at least 100 NOPS
```

Example 7. The Page Erase Routine—Assembly Version: COP8SGR (continued)

```
jsr DELAY                                         ;Jump to the delay routine
ld A, ADDRESSLO                                   ;Load the high address into the SIOR register
x A,SIOR                                          ;Do the swap here
jsr MICROWIRE_SEND                                ; Start the MICROWIRE/PLUS up
ld A,#064                                         ;Create a delay of at least 100 NOPS
jsr DELAY                                         ;Jump to the delay routine
jsr DETECT_READY                                  ;Detect if its ready to continue
                                                  ; Variable Host Delay implementation
ret                                               ;Return from the call
DETECT_READY:                                     ; Variable Host Delay routine
rbit SK,PORTGC                                    ;Set portg.sk into read only mode
rbit SK,PORTGD ;
VARIABLE_DELAY:                                   ;The holding is here
ifbit SK,PORTGP                                   ;If high then return and proceed to next
jp NEXT                                           ;instruction
jp VARIABLE_DELAY                                 ;Otherwise stay here
NEXT:                                             ;Continue on to the next instruction
sbit SK,PORTGC                                    ;Reset to normal mode when done
sbit SK,PORTGD ;
ret                                               ;Return to function call
MICROWIRE_SEND:                                 ;The MICROWIRE/PLUS send routine
sbit BUSY,PSW                                   ;SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                                     ;THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                                  ;IF THE BIT IS ON THEN WAIT
jp wait_uwire                                   ;Otherwise stay in the loop
ret                                             ;End of MICROWIRE/PLUS_SEND
DELAY:                                          ;THE DELAY ROUTINE
                                                            ; ASSUME THE AMOUNT OF NOPS IS STORED IN
                                                             ; THE ACCUMULATOR A
x A,DELAY_NUM                                    ;SET THE DELAY_NUM VARIABLE
LOOP_POINT:                                      ;POINT WHERE THE LOOP ACTUALLY OCCURS
NOP                                             ;THE ACTUAL NOPS
drsz DELAY_NUM                                   ;DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                                   ;ZERO
ret                                                ;End of DELAY
PGMTIM_SET:                                      ;THE SET WRITE TIMING ROUTINE
ld SIOR,#03B                                     ; PGMTIM_SET COMMAND Byte
jsr MICROWIRE_SEND                               ;Send the command byte out
ld A,#023                                        ; The amount of delay cycles required, For more
information
                                                 ; see 2-11 regarding required time delay cycles
jsr DELAY                                       ; The delay routine
ld SIOR,# 07B                                ; Send the Write Time - Assume a 10MHz CKI CBR CKI
                                                 ; frequency, See Table 2-13 for other values
jsr MICROWIRE_SEND                                ; Send the value out to the COP8CBR
ret                                            ; End of PGMTIM_SET Program
MICROINIT:
sbit MSEL,CNTRL                                ; CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                                ; PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                                ; PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                                ; PORTGD.SO = 0 to let the firmware know that
                                                ; you want to go into ISP Mode
                                                ; Set MICROWIRE/PLUS into standard sk mode,
                                              ; IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                                ;PORTGD.SK=1
sbit SI,PORTGC                                ;PORTGC.SI = 1
ret                                            ;RETURN FROM THE CALL
.END MAIN                                      ;END OF PROGRAM
```

Example 8. The Page Erase Routine—C Version: COP8SGR

```
#include "8sgr.h";                                            //Include file for the COP8SGR
Microcontroller
void page_erase(unsigned int addresshi, unsigned int addresslo);
void pgmtim_set(unsigned int frequency);                             //Set the write time
void microinit(void);                                                //Call the MICROWIRE/PLUS
initialization routine
void delay(unsigned int delay_num);                                  //The delay routine
void detect_ready();                                                 //Detect if it is ready to
continue routine
void microwire_send();                                               //The send microwire routine
void main(){                                                         //The main
microinit();                                                         //Initialize the program
pgmtim_set(0x7B);                                                    //Call the pgmtim_set routine
page_erase(0,0);                                                     //Erase location 0000 of the
flash
delay(6);                                                            //Delay for at least 6 NOPs as
specified in Table 18
detect_ready();                                                      //Detect if it is ready to
continue
while(1);                                                            //Endless loop
}
                                                                    //The page erase routine
void page_erase( unsigned int addresshi, unsigned int addresslo){
SIOR=0xB3;                                                           //Send out the command byte
value
microwire_send();                                                    //tell MICROWIRE/PLUS to
transmit
delay(35);                                                           //Delay for at least 35 NOPS as
specified in Table 18
SIOR=addresshi;                                                     // Set the high address
microwire_send();                                                   // tell MICROWIRE/PLUS to
transmit
delay(100);                                                         // Delay for at least 100 NOPs
SIOR=addresslo;                                                     // Set the low address
microwire_send();                                                   // tell MICROWIRE/PLUS to
transmit
delay(100);                                                         // Delay for at least 100 NOPs
detect_ready();
}
void pgmtim_set(unsigned int frequency){                            // The PGMTIM_SET
SIOR=0x3B;                                                          // Routine, Send out the
PGMTIM_SET command byte
microwire_send();                                                   // Start up the MICROWIRE/PLUS
and send the byte
delay(35);                                                          // Wait for 35 NOPs as required
in the time delay req.
SIOR=frequency;                                                     // Now send out the frequency
microwire_send();                                                   // Start out the MICROWIRE/PLUS
}                                                                   // End of the routine
void microinit(){                                                   // MICROWIRE/PLUS Initialization
Routine
CNTRL.MSEL=1;                                                       // The MICROWIRE/PLUS Control
Select Bit is set.
PORTGC.SK=1;                                                        // Set the MICROWIRE/PLUS into
PORTGC.SK=1
PORTGC.SO=1;                                                        // MODE: DATA OUT, PORTGC.SO=1
                                                                    // Set MICROWIRE/PLUS into
standard sk mode,
                                                                    // IDLE=High, Set According to
Table 2
PORTGD.SK=1;                                                        // Set IDLE MODE=High
PORTGC.SI=1;                                                        // Set the configuration bit
}                                                                   // End of the routine
void delay(unsigned int delay_num){                                 // The delay routine
```

Example 8. The Page Erase Routine—C Version: COP8SGR (continued)

```
unsigned int i;                                    // temp variable
for (i=0;i<delay_num; i++)                         // The loop control
NOP;                                               // Wait on NOP
}                                                  // End of the delay routine
void detect_ready(){                               // Detect if the host is ready
to send routine
PORTGC.SK=0;                                       // Set the PORTG.SK into input
mode
PORTGD.SK=0;
while(PORTGP.SK==0)                                // While the CLOCK line is still
low
NOP;                                               // Stay Here
PORTGC.SK=1;                                        //Other wise reset
PORTGD.SK=1;                                        //And Exit Routine
}                                                   //End of detect_ready()
void microwire_send(void){                         // The routine that starts the
microwire
CNTRL.MSEL=1;                                      // Set the MICROWIRE/PLUS
Control Bit
PSW.BUSY=1;                                        // Set the PSW busy bit
while(PSW.BUSY);                                   // Wait until the bit has cleared
}                                                  // end of the routine
```

4.1.4 MASS_ERASE—Bulk Erase the Flash Memory

Figure 29 shows the flow for the MASS_ERASE routine. Example 9 and Example 10 shows the assembly and C code versions of the MASS_ERASE routine.

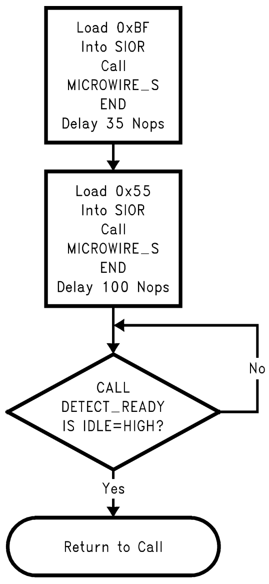

Figure 29. Flow for the Mass Erase Function

Example 9. The Mass Erase Routine—Assembly Version: COP8SGR

```
                                               ; Mass Erase The FLASH
.INCLD cop8sgr.INC                                    ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0;
DELAY_NUM .DSB 1                                      ;Allocate some memory
ADDRESSHI .DSB 1                                      ;To Hold the Upper Byte
ADDRESSLO .DSB 1                                      ;To Hold the Lower Byte
```

Example 9. The Mass Erase Routine—Assembly Version: COP8SGR (continued)

```
.SECT CODE,ROM,ABS=0                                    ;Beginning of the code
MAIN:                                                   ;Beginning of the main
jsr MICROINIT                                           ;Call the initialization routine
jsr PGMTIM_SET                                          ;Set the PGMTIM_SET routine
ld ADDRESSHI,#000                                       ;Addresshi=0
ld ADDRESSLO,#000                                       ;Addresslo=0
jsr Mass_ERASE                                          ;Call the Mass Erase routine
ld A,#006                                               ;Create a delay of at least 6 NOPS
jsr DELAY                                               ;Jump to the delay routine
jp MAIN                                                 ;Return to the main function
MASS_ERASE:                                             ;The erase function
ld SIOR,#0BF                                            ;The command byte for page erase
jsr MICROWIRE_SEND                                      ;Start the MICROWIRE/PLUS up
ld A,#023                                               ;Wait at least 35 cycles
jsr DELAY                                               ;Call the delay routine
ld A,#055                                               ;Load the confirmation code into the SIOR
register
x A,SIOR                                                ;Do the swap here
jsr MICROWIRE_SEND                                      ;Start the MICROWIRE/PLUS up
ld A,#064                                               ;Create a delay of at least 100 NOPS
jsr DELAY                                               ;Jump to the delay routine
jsr DETECT_READY                                        ;Detect if its ready to continue
                                                        ;Variable Host Delay implementation
ret                                                     ;Return from the call
DETECT_READY:                                           ;Variable Host Delay routine
rbit SK,PORTGC                                          ;Set portg.sk into read only mode
rbit SK,PORTGD                                          ;
VARIABLE_DELAY:                                         ;The holding is here
ifbit SK,PORTGP                                         ;If high then return and proceed to next
jp NEXT                                                 ;instruction
jp VARIABLE_DELAY                                       ;Otherwise stay here
NEXT:                                                   ; Continue on to the next instruction
sbit SK,PORTGC                                          ; Reset to normal mode when done
sbit SK,PORTGD                                          ;
ret                                                     ; Return to function call
MICROWIRE_SEND:                                         ; The MICROWIRE/PLUS send routine
sbit BUSY,PSW                                           ; SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                                             ; THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                                          ; IF THE BIT IS ON THEN WAIT
jp wait_uwire                                           ; Otherwise stay in the loop
ret                                                     ; End of MICROWIRE/PLUS_SEND
DELAY:                                                  ;THE DELAY ROUTINE
                                                        ; ASSUME THE AMOUNT OF NOPS IS STORED IN
                                                        ; THE ACCUMULATOR A
x A,DELAY_NUM                                           ; SET THE DELAY_NUM VARIABLE
LOOP_POINT:                                             ; POINT WHERE THE LOOP ACTUALLY OCCURS
NOP                                                     ; THE ACTUAL NOPS
drsz DELAY_NUM                                          ; DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                                           ; ZERO
ret                                                     ; End of DELAY
PGMTIM_SET:                                             ; THE SET WRITE TIMING ROUTINE
ld SIOR,#03B                                            ; PGMTIM_SET COMMAND Byte
jsr MICROWIRE_SEND                                      ; Send the command byte out
ld A,#023                                               ; The amount of delay cycles required, For more
information
                                                        ; see 2-11 regarding required time delay cycles
jsr DELAY                                               ; The delay routine
ld SIOR,# 07B                                           ; Send the Write Time -
 Assume a 10MHz CKI CBR CKI
                                                        ; frequency, See Table 2-13 for other values
jsr MICROWIRE_SEND                                      ; Send the value out to the COP8CBR
ret                                                     ; End of PGMTIM_SET Program
MICROINIT:
sbit MSEL,CNTRL                                         ; CNTRL.MSEL= 1, SET MICROWIRE INTO
```

Example 9. The Mass Erase Routine—Assembly Version: COP8SGR (continued)

```
sbit SK,PORTGC                                        ; PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                                        ; PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                                        ; PORTGD.SO = 0 to let the firmware know that
                                                      ; you want to go into ISP Mode
                                                      ; Set MICROWIRE/PLUS into standard sk mode,
                                                      ; IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                                        ; PORTGD.SK=1
sbit SI,PORTGC                                        ; PORTGC.SI = 1
ret                                                   ; RETURN FROM THE CALL
.END MAIN                                             ; END OF PROGRAM
```

Example 10. The Mass Erase Routine—C Version: COP8SGR

```
#include "8sgr.h";                             // Include file for the COP8SGR Microcontroller
void mass_erase();
void pgmtim_set(unsigned int frequency);              // Set the write time
void microinit(void);                                 // Call the MICROWIRE/PLUS initialization routine
void delay(unsigned int delay_num);                   // The delay routine
void detect_ready();                                  // Detect if it is ready to continue routine
void microwire_send();                                // Send the data out that is in the SIOR buffer
void main(){                                          // The main
microinit();                                         // Initialize the program
pgmtim_set(0x7B);                                    // Call the pgmtim_set routine
mass_erase();                                        // Mass erase the flash
delay(6);                                             //Delay for at least 6 NOPs as specified in Table
18
detect_ready();                                      // Detect if it is ready to continue
while(1);                                            // Endless loop
}
                                                      // The mass erase routine
void mass_erase( ){
SIOR=0xBF;                                          // Send out the command byte value
microwire_send();                                   // tell MICROWIRE/PLUS to transmit
delay(35);                                          // Delay for at least 35 NOPS as specified in the
Table 18
SIOR=0x55;                                         // Set the confirmation code
microwire_send();                                  // tell MICROWIRE/PLUS to transmit
delay(100);                                        // Delay for at least 100 NOPs
detect_ready();
}
void pgmtim_set(unsigned int frequency){           // The PGMTIM_SET
SIOR=0x3B;                                         // Routine, Send out the PGMTIM_SET command byte
microwire_send();                                  // Start up the MICROWIRE/PLUS and send the byte
delay(35);                                         // Wait for 35 NOPs as required in the time delay
req.
SIOR=frequency;                                   // Now send out the frequency
microwire_send();                                 // Start out the MICROWIRE/PLUS
}                                                // End of the routine
void microinit(){                               // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                                   // The MICROWIRE/PLUS Control Select Bit is set.
PORTGC.SK=1;                                    // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                                    // MODE: DATA OUT, PORTGC.SO=1
                                                // Set MICROWIRE/PLUS into standard sk mode,
IDLE=High,
                                                // Set According to Table 2
PORTGD.SK=1;                                    // Set IDLE MODE=High
PORTGC.SI=1;                                   // Set the configuration bit
}                                              // End of the routine
void delay(unsigned int delay_num){             // The delay routine
unsigned int i;                                // temp variable
for (i=0; i<delay_num; i++)                    // The loop control
```

Example 10. The Mass Erase Routine—C Version: COP8SGR (continued)

```
NOP;                                        // Wait on NOP
}                                           // End of the delay routine
void detect_ready(){                        // Detect if the host is ready to send routine
PORTGC.SK=0;                                 // Set the PORTG.SK into input mode
PORTGD.SK=0;
while(PORTGP.SK==0)                         // While the CLOCK line is still low
NOP;                                        // Stay Here
PORTGC.SK=1;                               // Otherwise reset
PORTGD.SK=1;                               // And Exit Routine
}                                          // End of detect_ready()
void microwire_send(void){                 // The routine that starts the microwire
CNTRL.MSEL=1;                              // Set the MICROWIRE/PLUS Control Bit
PSW.BUSY=1;                                 // Set the PSW busy bit
while(PSW.BUSY);                            // Wait until the bit has cleared
}                                           // end of the routine
```

4.1.5 READ_BYTE—Read a Byte from the Flash Memory Array

Figure 30 shows the flow for the READ_BYTE routine. Example 11 and Example 12 shows the assembly and C code versions for the routine.

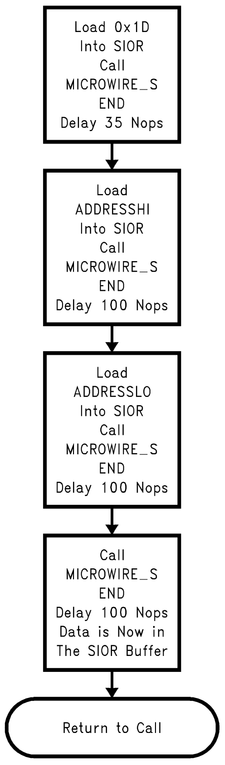
Load 0x1D Into SIOR Call MICROWIRE_S END Delay 35 Nops
Load ADDRESSHI Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load ADDRESSLO Into SIOR Call MICROWIRE_S END Delay 100 Nops
Call MICROWIRE_S END Delay 100 Nops Data is Now in The SIOR Buffer
Return to Call

Figure 30. Flow for the Read Byte Routine

Example 11. Read A Byte Of Flash—Assembly Version: COP8SGR

```
               ; The following routine will read a byte from the flash memory array
.INCLD cop8sgr.INC                          ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0                          ;Set the beginning of the RAM location
DELAY_NUM .DSB 1                          ;To be starting at 0F0
ADDRESSHI .DSB 1                          ;The high address byte
ADDRESSLO .DSB 1                          ;The low address byte
```

Example 11. Read A Byte Of Flash—Assembly Version: COP8SGR (continued)

```
DATA_READ .DSB 1                          ;The variable to store the returned data
.SECT CODE,ROM,ABS=0                      ;Beginning of the code
MAIN:                                    ;The Main
jsr MICROINIT                           ;Call the MICROWIRE/PLUS initialization routine
ld ADDRESSHI,#000                          ;Addresshi = 0 of flash
ld ADDRESSLO,#000                          ;Addresslo = 0 of flash
jsr READ_BYTE                          ;Call the read byte routine
ld A,#006                          ;Delay for 6 cycles
jsr DELAY                          ;Call the delay routine
ld A,SIOR                          ;Copy the buffer into the accumulator
x A,DATA_READ                      ;Save into the RAM location now
jp MAIN                            ;Return to the main
READ_BYTE:                          ;The read byte routine
ld SIOR,#01D                          ;The command byte for read byte
jsr MICROWIRE_SEND                    ;Call the MICROWIRE/PLUS send routine
ld A,#023                          ;Delay for 35 cycles
jsr DELAY                          ;Call the delay function
ld A,ADDRESSHI                     ;Set the actual high byte if the flash address
x A,SIOR                          ;Swap it with the MICROWIRE/PLUS buffer and send it out
jsr MICROWIRE_SEND                 ;Call the MICROWIRE/PLUS send routine
ld A,#064                          ;Delay for 100 NOPs
jsr DELAY                          ;Call the delay routine
ld A, ADDRESSLO                     ;Set the actual low byte of the flash address
x A,SIOR                          ;Swap it with SIOR
jsr MICROWIRE_SEND                 ;Call the MICROWIRE/PLUS send routine
ld A,#064                          ;Wait for 100 Nops
jsr MICROWIRE_SEND                 ;One last time to get the data
                                   ;Data should be in the SIOR register now
jsr DELAY                          ;Call the delay routine
jsr MICROWIRE_SEND                 ;ONE LAST CALL FOR THE DATA
                                   ;DATA SHOULD BE IN THE SIOR NOW
ret ;
MICROWIRE_SEND:                    ; MICROWIRE/PLUS Routine
sbit BUSY,PSW                     ; SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                        ; THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                          ; IF THE BIT IS ON THEN WAIT
jp wait_uwire                          ; Otherwise stay in the loop
ret                              ; End of MICROWIRE/PLUS_SEND
DELAY:                          ; THE DELAY ROUTINE
                                ; ASSUME THE AMOUNT OF NOPS IS STORED IN
                                 ; THE ACCUMULATOR A
x A,DELAY_NUM                          ; SET THE DELAY_NUM VARIABLE
LOOP_POINT:                          ; POINT WHERE THE LOOP ACTUALLY OCCURS
NOP                               ; THE ACTUAL NOPS
drsz DELAY_NUM                          ; DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                          ; ZERO
ret                                ; End of DELAY
MICROINIT:
sbit MSEL,CNTRL                          ; CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                          ; PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                          ; PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                          ; PORTGD.SO = 0 to let the firmware know that
                                        ; you want to go into ISP Mode
                                       ; Set MICROWIRE/PLUS into standard sk mode,
                                        ; IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                          ; PORTGD.SK=1
sbit SI,PORTGC                          ; PORTGC.SI = 1
ret                                        ; RETURN FROM THE CALL
.END MAIN                          ; END OF PROGRAM
```

Example 12. Read A Byte of Flash—C Version: COP8SGR

```
#include "8sgr.h";                                              //Include file for the COP8SGR
Microcontroller
unsigned int read_byte(unsigned int addresshi, unsigned int addresslo); //The read
byte function
void microinit(void);                                                    //The MICROWIRE/PLUS
initialization routine
void delay(unsigned int delay_num);                                    //The delay routine
void microwire_send();                                                    //The microwire send
routine
void main(){                                                             //The main routine
unsigned int data_read;                                                //The buffer that would hold
the result data
microinit();                                                           //Initialize the data vector
data_read=read_byte(0,1);                                              //Read at location 1 of the
flash
while(1);                                                              //Endless loop
}
                                                                       //The read_byte routine, it
will return the data byte read
unsigned int read_byte( unsigned int addresshi, unsigned in addresslo){
SIOR=0x1D;                                                              //The read_byte command byte
microwire_send();                                                       //Send it out to the
MICROWIRE/PLUS
delay(35);                                                              //Wait for 35 NOPs as stated
in Table 18
SIOR=addresshi;                                                         //Set up the high address
microwire_send();                                                       //Send it out on the
MICROWIRE/PLUS line
delay(100);                                                             //Wait for 100 Nops
SIOR=addresslo;                                                          //Set up the low address
microwire_send();                                                        //Send it out on the
MICROWIRE/PLUS line
delay(100);                                                             //Wait for 100 Nops
microwire_send();                                                       //One last time for the data
return SIOR;                                                            //Data should now be in the
SIOR buffer
}                                                                        //End of read_byte routine
void microinit(){                                                     // MICROWIRE/PLUS
Initialization Routine
CNTRL.MSEL=1;                                                    // The MICROWIRE/PLUS Control
Select Bit is set.
PORTGC.SK=1;                                                    // Set the MICROWIRE/PLUS into
PORTGC.SK=1
PORTGC.SO=1;                                                    // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                                                    // To tell the firmware that you
want to go into
                                                                // ISP Mode
                                                                // Set MICROWIRE/PLUS into standard
sk mode, IDLE=High,
                                                                // Set According to Table 2
PORTGD.SK=1;                                                    // Set IDLE MODE=High
PORTGC.SI=1;                                                    // Set the configuration bit
}                                                               // End of microinit routine
void delay(unsigned int delay_num){                             // The delay routine
unsigned int i;                                                 // temp variable
for (i-0; i<delay_num;i++)                                      // The loop control
NOP;                                                            // Wait on NOP
}                                                               // End of the delay routine
void microwire_send(void){                                      // The routine that starts the
microwire
CNTRL.MSEL=1;                                                    // Set the MICROWIRE/PLUS Control
Bit
PSW.BUSY=1;                                                    // Set the PSW busy bit
while(PSW.BUSY);                                                // Wait until the bit has cleared
```

Example 12. Read A Byte of Flash—C Version: COP8SGR (continued)

```
}                                                                // end of the routine
```

4.1.6 WRITE_BYTE—Write a Byte to the Flash Memory Array

Figure 31 shows the flow for the WRITE_BYTE routine. Example 13 and Example 14 and shows the assembly and C code versions for the routine.

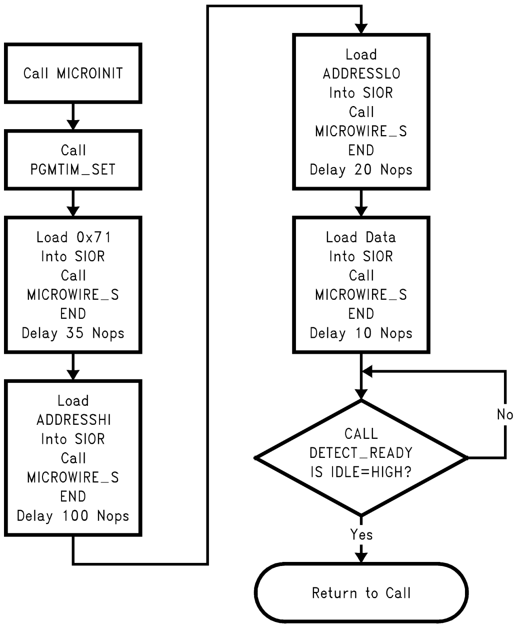
Call MICROINIT
Call PGMTIM_SET
Load 0x71 Into SIOR Call MICROWIRE_S END Delay 35 Nops
Load ADDRESSHI Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load ADDRESSLO Into SIOR Call MICROWIRE_S END Delay 20 Nops
Load Data Into SIOR Call MICROWIRE_S END Delay 10 Nops
CALL DETECT_READY IS IDLE=HIGH?
No
Yes
Return to Call

Figure 31. Flow for the Write Byte Function

Example 13. Write a Byte to the Flash—Assembly Version: COP8SGR

```
                                                                ; WRITE A BYTE TO THE FLASH ROUTINE
.INCLD cop8sgr.INC                                       ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0                                       ;DECLARE SOME VARIABLE
DELAY_NUM .DSB 1                                       ;The number of delays
ADDRESSHI .DSB 1                                       ;The high address byte
ADDRESSLO .DSB 1                                       ;The low address byte
WRITE_DATA .DSB 1                                        ;The variable where the data is to be read
                                                                ; from
.SECT CODE,ROM,ABS=0                               ;Beginning of the program
MAIN:                                                   ;The main routine
jsr MICROINIT                                           ;Initialize the MICROWIRE/PLUS
jsr PGMTIM_SET                                          ;Set the write time
ld A,#006                                               ;Delay for 6 cycles
jsr DELAY                                               ;Call the delay routine
ld ADDRESSHI,#000                                       ;Set the high address byte
ld ADDRESSLO,#001                                       ;Set the low address byte
ld WRITE_DATA,#005                                      ;Load a 5 where the write data variable is
jsr WRITE_BYTE                                          ;Call the write byte variable
jp MAIN                                                 ;Jump to the main function
WRITE_BYTE:                                             ;The write byte variable
ld SIOR,#071                                            ;The write byte command values
jsr MICROWIRE_SEND                                      ;Start-up the MICROWIRE/PLUS function
ld A,#023                                               ;Delay for 35 Nops
jsr DELAY                                               ;Call the delay function
ld A,ADDRESSHI                                          ;Set the high address byte
```

Example 13. Write a Byte to the Flash—Assembly Version: COP8SGR (continued)

```
x A,SIOR                          ;Then swap it with the SIOR register
jsr MICROWIRE_SEND                ;Call the MICROWIRE/PLUS send routine
ld A,#064                          ;Delay for 100 Nops
jsr DELAY                          ;Call the delay routine
ld A, ADDRESSLO                   ;Set the low address byte
x A,SIOR                          ;Swap it with the SIOR register
jsr MICROWIRE_SEND                ;Call the MICROWIRE/PLUS send routine
ld A,#014                          ;Delay for 20 Nops
jsr DELAY                          ;Call the delay routine
ld A, WRITE_DATA                   ;Set the low address byte
x A,SIOR                          ;Swap it with the SIOR register
jsr MICROWIRE_SEND                 ;Call the MICROWIRE/PLUS send routine
ld A,#014                          ;Delay for 20 Nops
jsr DELAY                          ;Call the delay routine
ld A,WRITE_DATA                    ;Get the data
x A,SIOR                          ;Get ready to send it out
jsr MICROWIRE_SEND                ;ONE LAST CALL FOR THE DATA
ld A,#00A                          ;Delay for 10 Nops
jsr DELAY                          ;Call the delay routine
jsr DETECT_READY                   ;Call the routine to detect if it is ready
ret                                 ;Return to the function call
PGMTIM_SET:                         ;THE SET WRITE TIMING ROUTINE
ld SIOR,#03B                         ;PGMTIM_SET COMMAND Byte
jsr MICROWIRE_SEND                  ;Send the command byte out
ld A,#023                          ; The amount of delay cycles required, For more informa-
                                   ; tion see Table 18 regarding required time delay cycles
jsr DELAY                          ; The delay routine
ld SIOR,# 07B                     ; Send the Write Time - Assume a 10MHz CKI CBR
; CKI frequency
ld A,#064                          ; The amount of delay cycles required, For more informa-
                                    ; tion see Table 18 regarding required time delay cycles
jsr DELAY                          ; The delay routine
jsr MICROWIRE_SEND                 ; Send the value out to the COP8CBR
ret                                ; End of PGMTIM_SET routine
DETECT_READY:                      ; Variable Host Delay routine
rbit SK,PORTGC                     ; Set portg.sk into read only mode
rbit SK,PORTGD                      ;
VARIABLE_DELAY:                     ; The holding is here
ifbit SK,PORTGP                     ; If high then return and proceed to next
jp NEXT                             ; instruction
jp VARIABLE_DELAY                   ; Otherwise stay here
NEXT:                               ; Continue on to the next instruction
sbit SK,PORTGC                      ; Reset to normal mode when done
sbit SK,PORTGD                       ;
ret                                  ;Return to function call
DELAY:                                 ; THE DELAY ROUTINE
                                       ; ASSUME THE AMOUNT OF NOPS IS STORED IN
                                       ; THE ACCUMULATOR A
x A,DELAY_NUM                          ; SET THE DELAY_NUM VARIABLE
LOOP_POINT:                            ; P01NT WHERE THE LOOP ACTUALLY OCCURS
NOP                                     ; THE ACTUAL NOPS
drsz DELAY_NUM                          ; DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                           ; ZERO
ret                                       ; End of DELAY
MICROINIT:
sbit MSEL,CNTRL                          ; CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                          ; PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                          ; PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                          ; PORTGD.SO = 0 to let the firmware know that
                                        ; you want to go into ISP Mode
                                         ; Set MICROWIRE/PLUS into standard sk mode,
                                       ; IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                          ; PORTGD.SK=1
sbit SI,PORTGC                          ; PORTGC.SI = 1
```

Example 13. Write a Byte to the Flash—Assembly Version: COP8SGR (continued)

```
ret                                    ; RETURN FROM THE CALL
MICROWIRE_SEND:                           ; MICROWIRE/PLUS Routine
sbit BUSY,PSW                           ; SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                           ; THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                           ; IF THE BIT IS ON THEN WAIT
jp wait_uwire                           ; Otherwise stay in the loop
ret                                   ; End of MICROWIRE/PLUS_SEND
.END MAIN                            ; End of the Program
```

Example 14. Write a Byte to the Flash—C Version: COP8SGR

```
#include "8sgr.h";                   // Include file for the COP8SGR Microcontroller
void write_byte(unsigned int addresshi, unsigned int addresslo, usigned int write_data);
                                             // The write byte routine
void microinit(void);                        // Initialize the MICROWIRE/PLUS initialization
void delay(unsigned int delay_num);          // Delay routine
void detect_ready();                         // Detect when CBR is ready
void microwire_send();                       // The microwire send routine
void main(){                                 // The main routine
microinit();                                 // The main initialization routine
pgmtim_set(0x7B);                            // Call the pgmtim_set routine
write_byte(0,1,5);                           // Write a byte at location 1 of flash with a 5
detect_ready();
while(1);                                    // Endless loop
}
void pgmtim_set(unsigned int frequency){ // The PGMTIM_SET
SIOR=0x3B;                                // Routine, Send out the PGMTIM_SET command byte
microwire_send();                         // Start up the MICROWIRE/PLUS and send the byte
delay(35);                                // Wait for 35 NOPs as required in the time delay req.
SIOR=frequency;                           // Now send out the frequency
microwire_send();                         // Start out the MICROWIRE/PLUS
delay(100);                               // Wait for at least 100 NOPs
}                                         // End of the routine
void write_byte( unsigned int addresshi, unsigned in addresslo, unsigned int write_data){
                                          // The actual function call
SIOR=0x1D;                                // The command byte value
microwire_send();                         // Send it out to the CBR
delay(35);                                // Delay for 35 Nops
SIOR=addresshi;                           // Send out the high address byte
microwire_send();                         // Tell MICROWIRE/PLUS to send it
delay(100);                               // Delay for 100 Nops
SIOR=addresslo;                           // Send out the low address byte
microwire_send();                         // Tell MICROWIRE/PLUS to send it
delay(100);                               // Delay for 100 Nops
microwire_send();                         // Now send the actual data to the microcontroller
delay(10);                                // Wait till the end of 10 cycles
detect_ready();                           // Detect if the CBR is ready for additional data
}                            // end of the write_byte routine
void microinit(){                        // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                            // The MICROWIRE/PLUS Control Select Bit is set
PORTGC.SK=1;                            // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                            // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                            // To tell the firmware that you want to go into
                                        // ISP Mode
                                        // Set MICROWIRE/PLUS into standard sk mode, IDLE=High,
                                        // Set According to Table 2
PORTGD.SK=1;                            // Set IDLE MODE=High
PORTGC.SI=1;                            // Set the configuration bit
}                            // End of microinit routine
void detect_ready(){                             //Detect if the host is ready to send routine
PORTGC.SK=0;                                     //Set the PORTG.SK into input mode
```

Example 14. Write a Byte to the Flash—C Version: COP8SGR (continued)

```
PORTGD.SK=0;
while(PORTGP.SK==0)                          //While the CLOCK line is still low
NOP;                                         //Stay Here
PORTGC.SK=1;                                 //Otherwise reset
PORTGD.SK=1; //And Exit Routine
}                           //End of detect_ready()
void delay(unsigned int delay_num){                  // The delay routine
unsigned int i;                                      // temp variable
for (i=0;i<delay_num;i++)                            // The loop control
NOP;                                                // Wait on NOP
}                           //End of delay routine
void detect_ready(){          // Detect if the host is ready to send routine
PORTGC.SK=0;                 //Set the PORTG.SK into input mode
PORTGD.SK=0;
while(PORTGP.SK==0)          //While the CLOCK line is still low
NOP;                         //Stay Here
PORTGC.SK=1;                 //Otherwise reset
PORTGD.SK=1;                 /And Exit Routine
}                           //End of the detect ready event
void microwire_send(void){                // The routine that starts the microwire
CNTRL.MSEL=1;                             // Set the MICROWIRE/PLUS Control Bit
PSW.BUSY=1;                               // Set the PSW busy bit
while(PSW.BUSY);                          // Wait until the bit has cleared
}                                         // end of the routine
```

4.1.7 BLOCK WRITE—Write a Block of Data to the Flash Memory Array

Figure 32 shows the flow for the BLOCK WRITE routine. Example 15 and Example 16 shows the assembly and C code versions for the routine.

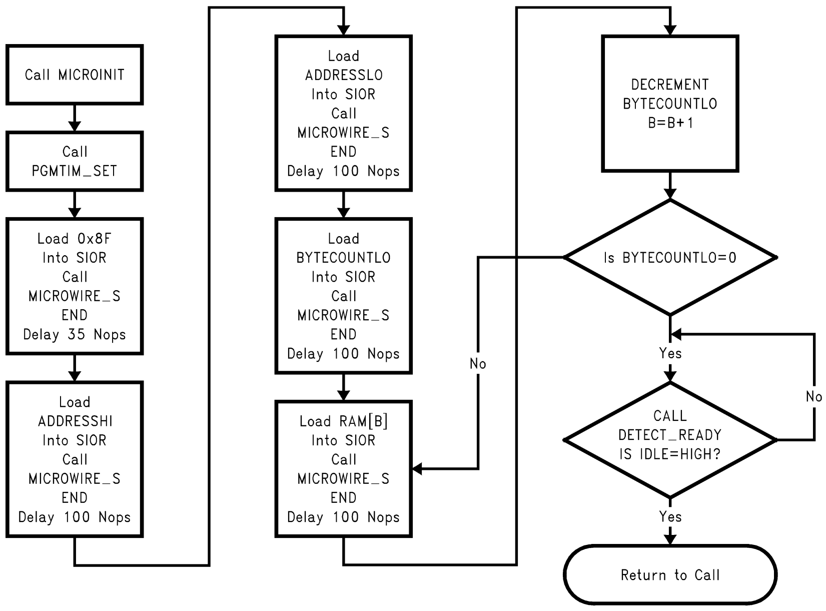
Call MICROINIT
Call PGMTIM_SET
Load 0x8F Into SIOR Call MICROWIRE_S END Delay 35 Nops
Load ADDRESSHI Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load ADDRESSLO Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load BYTECOUNTLO Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load RAM[B] Into SIOR Call MICROWIRE_S END Delay 100 Nops
DECREMENT BYTECOUNTLO B=B+1
Is BYTECOUNTLO=0
No
Yes
CALL DETECT_READY IS IDLE=HIGH?
No
Yes
Return to Call

Figure 32. Flow for the Block Write Function

Example 15. The Block Write Sample Code—Assembly Version: COP8SGR

```
                          ; WRITE A BLOCK OF DATA TO THE FLASH MEMORY
.INCLD cop8sgr.INC                            ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0                            ;Declare some variables
```

Example 15. The Block Write Sample Code—Assembly Version: COP8SGR (continued)

```
DELAY_NUM .DSB 1                          ;For the delay routine
ADDRESSHI .DSB 1                          ;To hold the high address byte
ADDRESSLO .DSB 1                          ;To hold the low address byte
BYTECOUNTLO .DSB 1                          ;To hold the number of bytes to send
WRITE_DATA1 .DSB 1                          ;To hold the sample data #1
WRITE_DATA2 .DSB 1                          ;To hold the sample data #2
WRITE_DATA3 .DSB 1                          ;To hold the sample data #3
.SECT CODE,ROM,ABS=0                          ;Beginning of the program
MAIN:                             ;The main function
jsr PGMTIM_SET                             ;Set the write time
ld A,#006                              ;Delay for 6 Nops before jumping
jsr DELAY                              ;Call the delay routine
jsr MICROINIT                              ;Initialize the MICROWIRE/PLUS routine
ld WRITE_DATA1,#001                        ;Set the first variable WRITE_DATA1=1
ld WRITE_DATA2,#002                       ;Set the second var. WRITE_DATA2=2
ld WRITE_DATA3,#003                       ;Set the third var. WRITE_DATA3=3
ld ADDRESSHI,#000                        ;The high address byte is 0
ld ADDRESSLO,#001                        ;The low address byte is 1
ld B,#WRITE_DATA1                        ;Set the pointer to the WRITE_DATA1
ld BYTECOUNTLO,#003                     ;Set the number of bytes to read as 3
jsr BLOCK_WRITE                          ;Call the block write routine
ld A,#006                              ;Delay for 6 Nops
jsr DELAY                              ;Call the delay routine
jp MAIN                              ;Return to the MAIN function
BLOCK_WRITE:                               ;The BLOCK_WRITE routine definition
ld SIOR,#08F                          ;The BLOCK_WRITE command byte
jsr MICROWIRE_SEND                    ;Startup MICROWIRE/PLUS
ld A,#023                              ;Delay for 35 Nops as specified in Table 18
jsr DELAY                              ;Call the delay routine
ld A,ADDRESSHI                         ;Load the high byte of the address to the
x A,SIOR                             ;SIOR Buffer
jsr MICROWIRE_SEND                     ;Send it out to the COP8CBR
ld A,#064                              ;Delay for 100 Nops
jsr DELAY                              ;Call the delay function
ld A, ADDRESSLO                        ;Load the low byte of the address to the
x A,SIOR                               ;SIOR Buffer
jsr MICROWIRE_SEND                    ;Send it out to the COP8CBR
ld A,#064                              ;Delay for 100 Nops
jsr DELAY                              ;Call the delay function
ld A,BYTECOUNTLO                       ;Load the value of bytecountlo into the
                                       ; accumulator
x A,SIOR                             ;Swap it with the SIOR buffer
jsr MICROWIRE_SEND                    ;Call MICROWIRE/PLUS to send it out
ld A,#064                              ;Setup a delay for 100 Nops
jsr DELAY                              ;Call the delay function
READ_PT:                             ;The point where the read loops around
ld A,[B+]                              ;Load the value where the B pointer is pointing at
x A,SIOR                             ;It assume the user has already setup the B pointer
jsr MICROWIRE_SEND                    ;Call the MICROWIRE/PLUS to send it out
ld A,#064                              ;Swap it with the SIOR buffer and delays it for 100 Nops
jsr DELAY                              ;Call the delay routine
drsz BYTECOUNTLO                        ;Decrement bytecount and jump next if zero
jp READ_PT                              ;Go back to the read point
jsr DETECT_READY                        ;Detect if it is ready to send
ret                             ;End of Block Write Routine
DETECT_READY:                              ;Variable Host Delay routine
rbit SK,PORTGC                              ;Set portg.sk into read only mode
rbit SK,PORTGD                              ;
VARIABLE_DELAY:                               ;The holding is here
ifbit SK,PORTGP                               ;If high then return and proceed to next
jp NEXT                                       ;instruction
jp VARIABLE_DELAY                               ;Otherwise stay here
NEXT:                                          ;Continue the next instruction
sbit SK,PORTGC                              ;Reset to normal mode when done
```

Example 15. The Block Write Sample Code—Assembly Version: COP8SGR (continued)

```
sbit SK,PORTGD                             ;
ret                             ;Return to function call
DELAY:                                 ;THE DELAY ROUTINE
                                           ; ASSUME THE AMOUNT OF NOPS IS STORED IN
                                           ; THE ACCUMULATOR A
x A,DELAY_NUM                             ;SET THE DELAY_NUM VARIABLE
LOOP_POINT:                               ;P01NT WHERE THE LOOP ACTUALLY OCCURS
NOP                                       ;THE ACTUAL NOPS
drsz DELAY_NUM                             ;DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                             ;ZERO
ret                             ;End of DELAY
MICROINIT:
sbit MSEL,CNTRL                      ; CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                       ; PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                       ; PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                       ; PORTGD.SO = 0 to let the firmware know that
                                     ; you want to go into ISP Mode
                                     ; Set MICROWIRE/PLUS into standard sk mode,
                                     ; IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                         ; PORTGD.SK=1
sbit SI,PORTGC                        ; PORTGC.SI = 1
ret                                   ; RETURN FROM THE CALL
PGMTIM_SET:                           ; THE SET WRITE TIMING ROUTINE
ld SIOR,#03B                          ; PGMTIM_SET COMMAND Byte
jsr MICROWIRE_SEND                    ; Send the command byte out
ld A,#023                             ; The amount of delay cycles required, For more informa-
                                      ; tion see Table 18 regarding required time delay cycles
jsr DELAY                             ; The delay routine
ld SIOR,# 07B                         ; Send the Write Time - Assume a 10MHz CKI CBR CKI frequency
jsr MICROWIRE_SEND                    ; Send the value out to the COP8CBR
ld A,#064                             ; The amount of delay cycles required, For more informa-
                                      ; tion see Table 18 regarding required time delay cycles
jsr DELAY                             ; The delay routine
ret                             ; End of PGMTIM_SET
MICROWIRE_SEND:                       ; MICROWIRE/PLUS Routine
sbit BUSY,PSW                         ; SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                           ; THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                        ; IF THE BIT IS ON THEN WAIT
jp wait_uwire                         ; Otherwise Wait until cleared
ret                                   ; End of MICROWIRE/PLUS_SEND
.END MAIN                             ; END OF PROGRAM
```

Example 16. The Block Write Sample Code–C Version: COP8SGR

```
#include "8sgr.h";                           // Include file for the COP8SGR Microcontroller
                                                    // The write byte routine
void block_write(unsigned int addresshi, unsigned int addresslo, unsigned int
bytecountlo); void microinit(void);                          // Initialize the
// MICROWIRE/PLUS initalization
void delay(unsigned int delay_num);                          // Delay it for 6 Nops
void detect_ready();                           // Detect when CBR is ready
void microwire_send();                           // The microwire send routine
unsigned int write_data[3];                           // Declare three data points for data storage
void main(){                              //The main routine
microinit();                              //The main initialization routine
pgmtim_set(0x7B);                              //Call the pgmtim_set routine
write_data[1]=1;                              //Storage location 1 for the block write routine
write_data[2]=2;                              //Storage location 2 for the block write routine
write_data[3]=3;                              //Storage location 3 for the block write routine
block_write(0,1,3);                              //Write a block of data beginning at flash(001) and
//bytecountlo=3, the assumed array is global with the name
```

Example 16. The Block Write Sample Code–C Version: COP8SGR (continued)

```
write_data[]
while(1);                           //Endless loop
}
void block_write( unsigned int addresshi, unsigned int addresslo, unsigned
int bytecountlo){                           //The actual function call
unsigned int i;                           //A counter variable
SIOR=0x8F;                          //The command byte value
microwire_send();                           //Send it out to the CBR
delay(35);                          //Delay for 35 Nops
SIOR=addresshi;                           //Send out the high address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                          //Delay for 100 Nops
SIOR=addresslo;                           //Send out the low address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                          //Delay for 100 Nops
microwire_send();                           //Now send the actual data to the microcontroller
SIOR=bytecountlo;                           //Send out the low address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                          //Delay for 100 Nops
for (i=0; i<bytecountlo;i++){                           //Send the data out until it reaches
bytecountlo
SIOR=write_data[i];                           //Send out the low address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                          //Delay for 100 Nops
}
detect_ready();                           //Detect if the CBR is ready for additional data
}                            //end of the block write routine
void microinit(){                           // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                           // The MICROWIRE/PLUS Control Select Bit is set.
PORTGC.SK=1;                           // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                           // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                           // To tell the firmware that you want to go into
                                       // ISP Mode
                                      // Set MICROWIRE/PLUS into standard sk mode, IDLE=High,
                                       // Set According to Table 2
PORTGD.SK=1;                           // Set IDLE MODE=High
PORTGC.SI=1;                           // Set the configuration bit
}                            // End of microinit routine
void detect_ready(){                           // Detect if the host is ready to continue
PORTGC.SK=0;                           // Set the PORTG.SK into input mode
PORTGD.SK=0;
while(PORTGP.SK==0)                           // While the CLOCK line is still low
NOP;                           // Stay Here
PORTGC.SK=1;                           // Otherwise reset
PORTGD.SK=1;                           // And Exit Routine
}                            // End of detect_ready()
void pgmtim_set(unsigned int frequency){ // The PGMTIM_SET
SIOR=0x3B;                           // Routine, Send out the PGMTIM_SET command byte
microwire_send();                           // Start up the MICROWIRE/PLUS and send the byte
delay(35);                           // Wait for 35 NOPs as required in the time delay req.
SIOR=frequency;                           // Now send out the frequency
microwire_send();                           // Start out the MICROWIRE/PLUS
delay(100);
}                            // End of the routine
void microwire_send(void){                           // The routine that starts the microwire
CNTRL.MSEL=1;                           // Set the MICROWIRE/PLUS Control Bit
PSW.BUSY=1;                           // Set the PSW busy bit
while(PSW.BUSY);                           // Wait until the bit has cleared
}                            // end of the routine
```

4.1.8 BLOCK_READ_Read a Block from the Flash Memory Array

Figure 33 shows the flow for the BLOCK READ routine. Example 17 and Example 18 shows the assembly and C code versions for the routine.

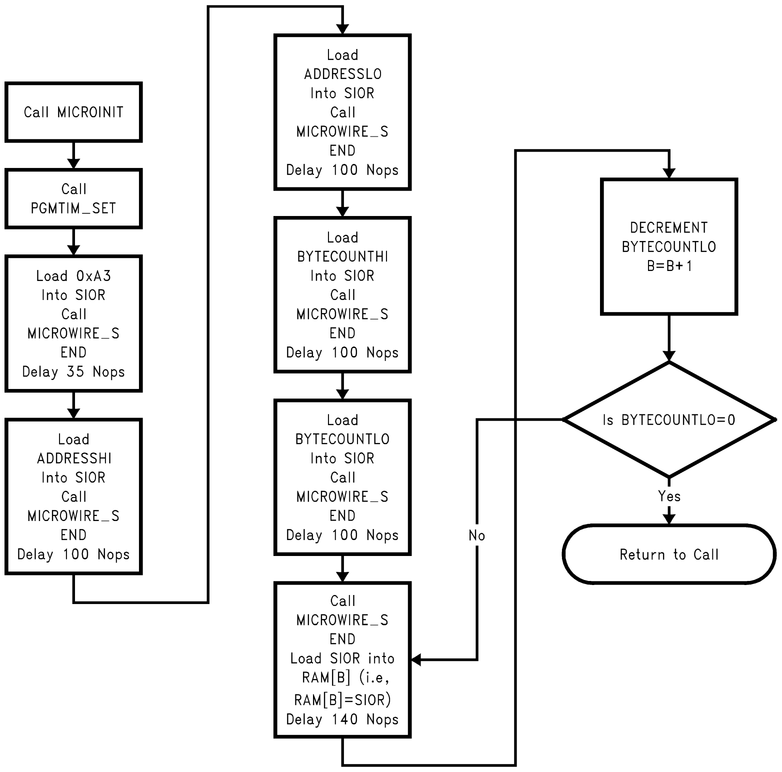
Call MICROINIT
Call PGMTIM_SET
Load 0xA3 Into SIOR Call MICROWIRE_S END Delay 35 Nops
Load ADDRESSHI Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load ADDRESSLO Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load BYTECOUNTHI Into SIOR Call MICROWIRE_S END Delay 100 Nops
Load BYTECOUNTLO Into SIOR Call MICROWIRE_S END Delay 100 Nops
Call MICROWIRE_S END Load SIOR into RAM[B] (i.e, RAM[B]=SIOR) Delay 140 Nops
DECREMENT BYTECOUNTLO B=B+1
Is BYTECOUNTLO=0
No
Yes
Return to Call

Figure 33. Flow for the BLOCK_READ Function

Example 17. The MICROWIRE/PLUS BLOCK_READ Routine—Assembly Routine: COP8SGR

```
                          ;READ A BLOCK OF DATA FROM THE FLASH MEMORY
.INCLD cop8sgr.INC                           ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0                           ;Declare some variables
DELAY_NUM .DSB 1                           ;For the delay routine
ADDRESSHI .DSB 1                           ;To hold the high address byte
ADDRESSLO .DSB 1                           ;To hold the low address byte
BYTECOUNTLO .DSB 1                           ;To hold the lower num of bytes to rec.
BYTECOUNTHI .DSB 1                           ;To hold the upper num of bytes to rec.
READ_DATA1 .DSB 1                           ;To hold the sample data #1
READ_DATA2 .DSB 1                           ;To hold the sample data #2
READ_DATA3 .DSB 1                           ;To hold the sample data #3
.SECT CODE,ROM,ABS=0                           ;Beginning of the program
MAIN:                           ;The main function
jsr MICROINIT                           ;Initialize the MICROWIRE/PLUS routine
ld ADDRESSHI,#000                           ;The high address byte is 0
ld ADDRESSLO,#001                           ;The low address byte is 1
ld B,#READ_DATA1                           ;Set the pointer to the READ_DATA1
ld BYTECOUNTLO,#003                           ;Set the lower num of bytes to read as 3
ld BYTECOUNTHI,#000                           ;Set the up num of bytes to read
jsr BLOCK_WRITE                           ;Call the block write routine
ld A,#006                           ;Delay for 6 Nops
jsr DELAY                           ;Call the delay routine
jp MAIN                           ;Return to the MAIN function
BLOCK_READ:                           ;The BLOCK_WRITE routine definition
ld SIOR,#0A3                           ;The BLOCK_WRITE command byte
jsr MICROWIRE_SEND                           ;Start-up the MICROWIRE/PLUS
```

Example 17. The MICROWIRE/PLUS BLOCK_READ Routine—Assembly Routine: COP8SGR (continued)

```
ld A,#023                            ;Delay for 35 Nops as specified in Table 18
jsr DELAY                            ;Call the delay routine
ld A,ADDRESSHI                            ;Load the high byte of the address to the
x A,SIOR                            ;SIOR Buffer
jsr MICROWIRE_SEND                            ;Send it out to the COP8CBR
ld A,#064                            ;Delay for 100 Nops
jsr DELAY                            ;Call the delay function
ld A, ADDRESSLO                            ;Load the low byte of the address to the
x A,SIOR                            ;SIOR Buffer
jsr MICROWIRE_SEND                            ;Send it out to the COP8CBR
ld A,#064                            ;Delay for 100 Nops
jsr DELAY                            ;Call the delay function
ld A,BYTECOUNTHI                            ;Load the value of bytecountlo into the
                          ;accumulator
x A,SIOR                            ;Swap it with the SIOR buffer
jsr MICROWIRE_SEND                            ;Call MICROWIRE/PLUS to send it out
ld A,#064                            ;Setup a delay for 100 Nops
jsr DELAY                            ;Call the delay function
ld A,BYTECOUNTLO                            ;Load the value of bytecountlo into the accumulator
x A,SIOR                            ;Swap it with the SIOR buffer
jsr MICROWIRE_SEND                            ;Call MICROWIRE/PLUS to send it out
ld A,#064                            ;Setup a delay for 100 Nops
jsr DELAY                            ;Call the delay function
SAVE_PT:                            ;The point where the read loops around
ld A,#08C                            ;First wait until the delay is ready, 140 NOPs
jsr DELAY                            ;by calling the delay routine
jsr MICROWIRE_SEND                            ;Tell MICROWIRE/PLUS to send the data over
ld A,[B+]                            ;Load the value where the B pointer is pointing at
x A,SIOR                            ;It assumes the user has already setup the B pointer
drsz BYTECOUNTLO                            ;Decrement bytecount and jump next if zero
jp READ_PT                            ;Go back to the read point
jsr DETECT_READY                            ;Detect if it is ready to send
ret                            ;Return to the call
DETECT_READY:                            ;Variable Host Delay routine
rbit SK,PORTGC                            ;Set portg.sk into read only mode
rbit SK,PORTGD                            ;
VARIABLE_DELAY:                            ;The holding is here
ifbit SK,PORTGP                            ;If high then return and proceed to next
jp NEXT                            ;instruction
jp VARIABLE_DELAY                            ;Otherwise stay here
NEXT:                            ;Continue the next instruction
sbit SK,PORTGC                            ;Reset to normal mode when done
sbit SK,PORTGD                            ;
ret                            ;Return to function call
DELAY:                            ;THE DELAY ROUTINE
                          ;ASSUME THE AMOUNT OF NOPS IS STORED IN
                          ;THE ACCUMULATOR A
x A,DELAY_NUM                            ;SET THE DELAY_NUM VARIABLE
LOOP_POINT:                            ;P01NT WHERE THE LOOP ACTUALLY OCCURS
NOP                                          ;THE ACTUAL NOPS
drsz DELAY_NUM ;DECREMENT DELAY_NUM, SKIP NEXT INSTR. IF
jp LOOP_POINT                            ;ZERO
ret                            ;End of DELAY
MICROINIT:
sbit MSEL,CNTRL                            ;CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                            ;PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                            ;PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                            ;PORTGD.SO = 0 to let the firmware know that
                          ;you want to go into ISP Mode
                          ;Set MICROWIRE/PLUS into standard sk mode,
                          ;IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                            ;PORTGD.SK=1
sbit SI,PORTGC                            ;PORTGC.SI = 1
ret                            ;RETURN FROM THE CALL
```

Example 17. The MICROWIRE/PLUS BLOCK_READ Routine—Assembly Routine: COP8SGR (continued)

```
MICROWIRE_SEND:                           ;MICROWIRE/PLUS Send Routine
sbit BUSY,PSW                           ;SET THE PSW.BUSY BIT TO TURN ON
wait_uwire:                           ;THE MICROWIRE WAIT ROUTINE
ifbit BUSY,PSW                           ;IF THE BIT IS ON THEN WAIT
jp wait_uwire                           ;STAY MICROLOOP IF NOT DOT
ret                           ;End of MICROWIRE/PLUS_SEND
.END MAIN                           ;END OF PROGRAM
```

Example 18. The MICROWIRE/PLUS BLOCK_READ Routine—C Version: COP8SGR

```
#include "8sgr.h";                           // Include file for the COP8SGR Microcontroller
// The write byte routine
void block_read(unsigned int addresshi, unsigned int addresslo, unsigned int bytecounthi,
unsigned int bytecounthi);
void microinit(void);                           // Initialize the
                                                    // MICROWIRE/PLUS initalization
void delay(unsigned int delay_num); // Delay it for 6 Nops
void detect_ready();                           // Detect when CBR is ready
void microwire_send();
unsigned int write_data[3];                           //Declare three data points for data storage
void main(){                           //The main routine
microinit();                           //The main initialization routine
pgmtim_set(0x7B);                           //Call the pgmtim_set routine
write_data[1]=1;                           //Storage location 1 for the block write routine
write_data[2]=2;                           //Storage location 2 for the block write routine
write_data[3]=3;                           //Storage location 3 for the block write routine
block_write(0,1,0,3);                           //Write a block of data beginning at flash(001) and
                          //bytecounthi=0 and bytecountlo=3, the assumed array
                          //is global with the name write_data[]
while(1);                           //Endless loop
}
void block_read( unsigned int addresshi, unsigned in addresslo, unsigned int bytecounthi,
unsigned int bytecountlo){                           //The actual
                                                    //function call
unsigned int i, j ;                           //Counter Variables
SIOR=0x8F;                           //The command byte value
microwire_send();                           //Send it out to the CBR
delay(35);                           //Delay for 35 Nops
SIOR=addresshi;                           //Send out the high address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
SIOR=addresslo;                           //Send out the low address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
microwire_send();                           //Now send the actual data to the microcontroller
SIOR=bytecounthi;                           //Send out the high address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
SIOR=bytecountlo;                           // Send out the low address byte
microwire_send();                           // Tell MICROWIRE/PLUS to send it
delay(100);                           // Delay for 100 Nops
for (j=0;j<bytecountlo;j++){                           // Send the data out until it reaches
bytecountlo
delay(140);                           // Delay for 140 Nops
read_data[i]=SIOR;                           // Send out the low address byte
microwire_send();                           // Tell MICROWIRE/PLUS to send it
}
detect_ready();                           // Detect if the CBR is ready for additional data
}                           // end of the block write routine
void microinit(){                           // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                           // The MICROWIRE/PLUS Control Select Bit is set.
```

Example 18. The MICROWIRE/PLUS BLOCK_READ Routine—C Version: COP8SGR (continued)

```
PORTGC.SK=1;                           // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                           // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                           // To tell the firmware that you want to go into
                          // ISP Mode
                          // Set MICROWIRE/PLUS into standard sk mode, IDLE=High,
                          // Set According to Table 2
PORTGD.SK=1;                           // Set IDLE MODE=High
PORTGC.SI=1;                           // Set the configuration bit
}                           // End of microinit routine
void microwire_send(void){ // The routine that starts the microwire
CNTRL.MSEL=1;                           // Set the MICROWIRE/PLUS Control Bit
PSW.BUSY=1;                           // Set the PSW busy bit
while(PSW.BUSY);                           // Wait until the bit has cleared
}                           // end of the routine
```

4.1.9 EXIT—Reset the Microcontroller

Figure 34 shows the flow for the EXIT routine. Example 19 Example 20 shows the assembly and C version for the routine.

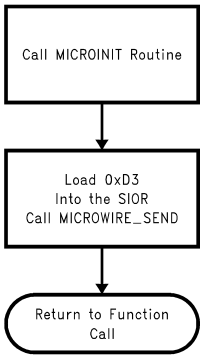

Figure 34. Flow for the Exit Routine

Example 19. The MICROWIRE/PLUS Exit Routine—Assembly Routine: COP8SGR

```
                          ;Cause the microcontroller to perform a reset
.INCLD cop8sgr.INC                           ;INCLUDE FILE FOR THE COP8SGR
.sect DATA,REG,ABS=0F0                           ;Declare some variables
DELAY_NUM .DSB 1                           ;For the delay routine
.SECT CODE,ROM,ABS=0                           ;Begining of the program
MAIN:                           ;The main function
jsr MICROINIT                           ;Initialize the MICROWIRE/PLUS routine
jsr EXIT                           ;Jump to the exit routine
jp MAIN                           ;Return to the MAIN function
EXIT:                           ;The MICROWIRE/PLUS send routine
ld SIOR,#0D3                           ;The MICROWIRE/PLUS command byte
jsr MICROWIRE_SEND                           ;Tell the MICROWIRE/PLUS to send the byte out
ret                           ;End of the Exit routine
MICROINIT:
sbit MSEL,CNTRL                           ;CNTRL.MSEL= 1, SET MICROWIRE INTO
sbit SK,PORTGC                           ;PORTGC.SK = 1, MODE, DATA OUT,
sbit SO,PORTGC                           ;PORTGC.SO = 1, DATA IN
rbit SO,PORTGD                           ;PORTGD.SO = 0 to let the firmware know that
                          ;you want to go into ISP Mode
                          ;Set MICROWIRE/PLUS into standard sk mode,
                          ;IDLE = High, SET ACCORDING TO Table 20
sbit SK,PORTGD                           ;PORTGD.SK=1
```

Example 19. The MICROWIRE/PLUS Exit Routine—Assembly Routine: COP8SGR (continued)

```
sbit SI,PORTGC                              ;PORTGC.SI = 1
ret                           ;RETURN FROM THE CALL
MICROWIRE_SEND:                              ;MICROWIRE/PLUS Send Routine
sbit BUSY,PSW                              ;Set the PSW.BUSY bit to turn on
wait_uwire:                              ;The MICROWIRE/PLUS wait routine
ifbit BUSY,PSW                              ;If the busy bit is on then wait.
jp wait_uwire                              ;Otherwise wait here until it has cleared
ret                           ;End of MICROWIRE/PLUS_SEND
.END MAIN                           ;END OF PROGRAM
```

Example 20. The MICROWIRE/PLUS Exit Routine—C Routine: COP8SGR

```
#include "8sgr.h";                           //Include file for the COP8SGR Microcontroller
                                                    //The write byte routine
void block_read(unsigned int addresshi, unsigned int addresslo, unsigned int bytecounthi,
unsigned int bytecounthi);
void microinit(void);                           //Initialize the
                                                    //MICROWIRE/PLUS initialization
void delay(unsigned int delay_num);                           //Delay it for 6 Nops
void detect_ready();                           //Detect when CBR is ready
void exit();
unsigned int write_data[3];                           //Declare three data points for data storage
void microwire_send();
void main(){                           //The main routine
microinit();                           //The main initialization routine
exit();                           //Call the exit routine
while(1);                           //Endless loop
}
void exit(){                           //The MICROWIRE/PLUS exit routine
SIOR=0xD3;                           //The exit command byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it out
}
void block_read( unsigned int addresshi, unsigned in addresslo, unsigned int bytecounhi,
unsigned int bytecountlo){                           //The actual
                                                    //function call
unsigned int i, j ;                           //Counter Variables
SIOR=0x8F;                           //The command byte value
microwire_send();                           //Send it out to the CBR
delay(35);                           /Delay for 35 Nops
SIOR=addresshi;                           //Send out the high address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
SIOR=addresslo;                           //Send out the low address byte
microwire_send();                           //Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
microwire_send();                           //Now send the actual data to the microcontroller
SIOR=bytecounthi;                           //Send out the high address byte
microwire_send();                           /Tell MICROWIRE/PLUS to send it
delay(100);                           //Delay for 100 Nops
}
void microinit(){                           // MICROWIRE/PLUS Initialization Routine
CNTRL.MSEL=1;                           // The MICROWIRE/PLUS Control Select Bit is set.
PORTGC.SK=1;                           // Set the MICROWIRE/PLUS into PORTGC.SK=1
PORTGC.SO=1;                           // MODE: DATA OUT, PORTGC.SO=1
PORTGD.SO=0;                           // To tell the firmware that you want to go into
                          // ISP Mode
                          // Set MICROWIRE/PLUS into standard sk mode, IDLE=High,
                          // Set According to Table 2
PORTGD.SK=1;                           // Set IDLE MODE=High
PORTGC.SI=1;                           // Set the configuration bit
}                            // End of microinit routine
```

Example 20. The MICROWIRE/PLUS Exit Routine—C Routine: COP8SGR (continued)

```
void microwire_send(void){                           // The routine that starts the microwire
CNTRL.MSEL=                          // Set the PSW busy bit
while(PSW.BUSY);                          // Wait until the bit has cleared
}                           // end of the routine
```

4.2 USER SUPPORT BLOCK

This section deals with the User Support Block. Entry point locations are shown in Table 20. Register locations are shown in Table 21. Registers are shown in Table 22. In addition, each description contains details about security dependencies. There are no checks made for the validity of the ISP Address and the BYTECOUNT register. Data transfers will take place from whatever RAM locations are specified by the segment register.

4.2.1 JSRB LABELS User Routines

To execute commands listed in Table 20, the JSRB instruction must be used. In order for correct behavior, a “KEY” must be set prior to executing the JSRB instruction. The PGMTIM register must be set prior to any write or erase commands. It is up to the user to enforce security when using these commands. At the end of each command, a RETF is issued to return control back to the user code in flash memory.

Table 20. User Entry Points and Their Associated Labels

Command/Labels	ROM ADDRESS
cpgerase	0x17
cmserase	0x1A
creadbf	0x11
vblockr	0x26
cwritebf	0x14
vblockw	0x23
exit	0x62

Table 21. Registers

Register Name	Purpose	RAM LOCATION
ISPADHI	High Address of Flash Memory	0xA9
ISPADLO	Low Address of Flash Memory	0xA8
ISPWR	Must store the byte to be written into this register before jumping into the write byte routine.	0xAB
ISPRD	Data will be returned to this register after the read byte routine execution.	0xAA
ISPKEY	Register which will hold the KEY value. The KEY value is utilized to verify that a JSRB execution is requested in the next 6 instruction cycles.	0xE2
BYTECOUNTLO	Holds the low byte of the counter.	0xF1
SIOR	MICROWIRE/PLUS Buffer	0xE9
PGMTIM	Write Timing Register	0xE1
Confirmation Code	The code used to verify that a mass erase was requested, Confirmation Code must be loaded prior to the jump to the cmserase routine.	Accumulator A
KEY	Must write to the ISPKEY register before a JSRB executed.	0x98

Table 22. User Entry Points

Command/ Label	Function	Parameters	Return Data
cpgerase	Page Erase	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	N/A (A page of memory beginning at ISPADHI, ISPADLO will be erased).
cmserase	Mass Erase	Accumulator A contains the confirmation key 0x55.	N/A (The entire Flash Memory will be erased). The Boot ROM will return the user to the MICROWIRE/PLUS Boot ISP since the Flash is completely erased.
creadbf	Read Byte	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address.	Data Byte in Register ISPRD.
cblockr	Block Read	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. X pointer contains the beginning RAM address where the result(s) will be returned. Register BYTECOUNTLO contains the number of n bytes to read ($0 \leq n \leq 255$). Register BYTECOUNTHI is **ignored**. It is up to the user to setup the segment register.	n Data Bytes if $0 \leq n \leq 255$ Data will be returned beginning at a location pointed to by the RAM address in X.
cwritebf	Write Byte	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register ISPWR contains the Data Byte to be written.	N/A
cblockw	Block Write	Register ISPADHI is loaded by the user with the high byte of the address. Register ISPADLO is loaded by the user with the low byte of the address. Register BYTECOUNTLO contains the number of n bytes to write ($0 \leq n \leq 16$) X pointer contains the beginning RAM address of the data to be written. It is the user's responsibility to initialize the segment register. Data must be placed with-in the 1/2 page segment (64 byte for 32k devices and 32 byte for 1k and 4k devices). This limitation is due to the multi-byte write limitation.	N/A
exit	EXIT	N/A	N/A (Device will Reset).

4.3 Interrupt Lock Out Time

Interrupts are inhibited during execution from Boot ROM. Table 23 shows the amount of time that the user is **LOCKED OUT** of their interrupt service routine(s). The servicing of interrupts will be resumed once the ISP Boot ROM returns the user to the Flash. Any interrupt(s) that are pending during user ISP will be serviced after the user has returned to the Flash area. The user should take into account the amount of time they are locked out of their interrupts. Some of the **LOCK OUT** times are dependent upon the PGMTIM. PGMTIM is a value entered into the PGMTIM register (refer to Section 3.9.1 regarding PGMTIM). Although Section 3.9.1 pertains to MICROWIRE/PLUS commands, the user code **MUST** set the PGMTIM register before any write routines occur (e.g., a LD PGMTIM,#06F is needed to specify a CKI frequency of 6 MHz).

Table 23. Required Interrupt Lockout Time (in Instruction Cycles)

Flash Routines (User Accessible)	Minimum Interrupt Latency Time (In Instruction Cycles)
cppgerase	120 + 100*PGMTIM[1]
cmserase	120 + 300*PGMTIM[1]
creadbf	100
cblockr	140/BYTE
cblockw	100 + 3.5*PGMTIM/BYTE[1] + 68/BYTE
cwritebf	168 + 3.5*PGMTIM[1]
exit	100

[1] Refer to Section 3.9.1 for additional information on the PGMTIM variable.

4.4 cpgerase—User Entry Point: Erase a Page of Flash Memory

This routine requires that ISPADHI and ISPADLO are loaded before the jump. A KEY is a number which must be loaded into the KEY Register (location at 0xE2) before issuing a JSRB instruction. Table 1-5 shows the possible format of the KEY number. Loading the KEY, and a "JSRB cpgerase" are all that is needed to complete the call to the routine. No acknowledgment will be sent back regarding the operation. For details regarding the registers ISPADHI and ISPADLO please refer to Table 21. See Table 25 for details on the number of endurance cycles and the number of page erase commands that should be issued prior to writing data into the erase page. Since this is a user command, this routine will work regardless of security (security independent). Example 21 is an example of how to use the cpgerase function—assembly version. Example 22 shows the C version of the pgerase() function. Table 24 shows the necessary resources needed to run the routine.

Example 21. SAMPLE cpgerase (PAGE ERASE) EXECUTION—Assembly Version

```
; ERASE A PAGE FROM FLASH, 0x0080
; ASSUME A 6 MHz CKI FREQUENCY
.INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
.sect code,rom,abs=0 ; BEGINING CODE SPACE
MAIN: ; THE MAIN FUNCTION
LD PGMTIM,#06F ; FOR A 10 MHz CLOCK (DEFAULT)
LD ISPADHI,#000 ; LOAD THE HIGH BYTE ADDRESS
LD ISPADLO,#080 ; LOAD THE LOW BYTE ADDRESS
LD ISPKEY,#098 ; LOAD THE KEY
JSRB cpgserase ; CALL THE ROUTINE
.END MAIN ; END OF THE PROGRAM
```

Example 22. SAMPLE pgerase(unsigned long) EXECUTION—C Version

```
#include "8cbr.h"; // Include file for the COP8CBR Microcontroller
#include "flash_OP.h"; // Include file that contain the flash routines
void main(){
PGMTIM=0x6F; // For a 6 MHz CKI Frequency
page_erase(0x80); // Call the erase routine
```

Table 24. Resource Utilization for the Command: cpgerase (Page Erase)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPADHI ISPADLO	YES	YES	NO	YES	YES	NONE	120 +100* PGMTIM[1]	4

[1] Refer to Section 3.9.1 for additional information on the PGMTIM variable.

Table 25. Typical Endurance Cycles vs. Erase Time and Temperature

Erase Time in ms	Low End of Operating Temp Range			
	-40°C	-20°C	0°C	25°C
1	60k	60k	60k	100k
2	60k	60k	60k	100k
3	60k	60k	60k	100k
4	60k	60k	100k	100k
5	70k	70k	100k	100k
6	80k	80k	100k	100k
7	90k	90k	100k	100k
8	100k	100k	100k	100k

4.5 cmserase—User Entry Point: Mass Erase the Flash Memory

This routine requires the Accumulator A to contain 0x55 prior to the jump. The value 0x55 is used to verify that a mass erase was requested. Loading the KEY, “LD A,#055”, and a “JSRB cmserase” are all that is needed to complete the function. No acknowledgment will be sent back regarding the operation. Since this is a user command, this routine will work regardless of security (security independent). After a mass erase is executed the user will be brought back (after 112 instruction cycles) to the beginning of the Boot ROM. Control and execution will be returned to the MICROWIRE/PLUS ISP handling routine. Table 24 shows the necessary resources needed to run the routine. Example 23 is an example of how to use the cmserase function—assembly version. Example 24 shows the C version of the mass_erase() function. Table 26 shows the necessary resources needed to run the routine.

Example 23. SAMPLE cmserase (MASS ERASE) EXECUTION—Assembly Version

```
; MASS ERASE THE FLASH
; ASSUME A 6 MHz CKI FREQUENCY
.INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
.sect code,rom,abs=0 ; BEGINING CODE SPACE
MAIN: ; THE MAIN FUNCTION
LD PGMTIM,#06F ; FOR A 6 MHz CLOCK
LD ISPADHI,#055 ; LOAD THE CONFIRMATION CODE
LD ISPKEY,#098 ; LOAD THE KEY
JSRB cmserase ; CALL THE FUNCTION
.END MAIN ; END OF THE PROGRAM
```

Example 24. SAMPLE mass_erase() EXECUTION—C Version

```
#include "8cbr.h";             // Include file for the COP8CBR Microcontroller
#include "flash_OP.h";               // Include file that contain the flash routines
void main(){
PGMTIM=06F;                         // For a 10 MHz CKI Frequency
mass_erase();                      // Call the erase routine
}
```

Table 26. Resource Utilization for the Command: cmserase (Mass Erase)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
Confirmation Code ""0x55"" is Loaded into Accumulator A	YES	YES	NO	YES	YES	NONE	120 +300* PGMTIM[(1)]	6

(1) Refer to Section 3.9.1 for additional information on the PGMTIM variable.

4.6 creadbf—User Entry Point: Read a Byte of Flash Memory

This routine requires that ISPADHI and ISPADLO are loaded before the jump. Loading the KEY, and a “JSRB creadbf” are all that is needed to complete the call to the routine. Data will be returned to the ISPRD Register. No acknowledgment will be sent back regarding the operation. For details regarding the registers ISPADHI, ISPADLO, and ISPRD please refer to Table 21. Since this is a user command, this routine will work regardless of security (security independent). Example 25 is an example of how to use the creadbf function—assembly version. Example 26 shows the C version of the readbf() function. Table 24 shows the necessary resources needed to run the routine.

Example 25. SAMPLE unsigned int creadbf (Read a Byte of Flash Memory) EXECUTION

```
; READ A BYTE FROM THE FLASH [0001]
.INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
.sect code,rom,abs=0 ; BEGINING CODE SPACE
MAIN: ; BEGINING OF THE CODE
LD ISPADHI,#000 ; LOAD THE HIGH ADDRESS BYTE
LD ISPADLO,#001 ; LOAD THE LOW ADDRESS BYTE
LD ISPKEY,#098 ; LOAD THE KEY
JSRB creadbf ; CALL THE FUNCTION
; TRANSFER RAM[5]=FLASH[0001]
LD A,ISPRD ; LOAD THE RESULT ONTO THE ACCUMULATOR
X A,005 ; TRANSFER IT TO RAM[0x05]
.END MAIN ; END OF PROGRAM
```

Example 26. SAMPLE readbf(unsigned long) EXECUTION—C Version

```
#include "8cbr.h";            // Include file for the Flash Microcontroller
#include "flash_OP.h";               // Include file that contain the flash routines
void main(){
unsigned int storage[10];
                                      // read a byte from flash location 0x0001 and store it
storage[5]
storage[5]=readbf(0x01);
}
```

Table 27. Resource Utilization for the Command: creadbf (Read a Byte of Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPADHI ISPADLO	YES	YES	NO	YES	YES	Data/ISPRD Register	100	4

4.7 cblockr—User Entry Point: Read a Block of Flash Memory

The cblockr routine will read multiple bytes from the Flash memory. ISPADHI and ISPADLO are assumed to be loaded before the jump. Register BYTECOUNTLO is also assumed to be loaded. The X pointer contains the address where the data will be placed. The BYTECOUNTLO register is used by the microcontroller to send back N number of bytes (i.e., N=BYTECOUNTLO). If N=0 then the firmware will abort. Data is saved into the RAM address pointed to by the X pointer. It is up to the user to setup the segmentation register. This routine is capable of reading up to 256 bytes of flash memory (limited due to the memory available) to RAM. This routine is limited to reading blocks of 128 bytes due to the RAM segmentation. If an attempt to read greater than 128 bytes is issued, the firmware will begin to write to RAM locations beginning at 0x80 (possibly corrupting the I/O and CONTROL REGISTERS) and above. After the X pointer and the BYTECOUNTLO is set, the KEY must be loaded, and a “JSRB cblockr” must be issued. No acknowledgment will be sent back regarding the operation. For details regarding the registers ISPADHI, ISPADLO, and BYTECOUNTLO please refer to Table 21. Since this is a user command, this routine will work regardless of security (security independent). Example 27 is an example of how to use the cblockr function—assembly version. Example 28 shows the C version of the block_read() function. Table 28 shows the necessary resources needed to run the routine.

Example 27. SAMPLE cblockr (Read a Block of Flash Memory) EXECUTION —Assembly Version

```
; BLOCK READ, READ 3 BYTE(S) BEGINING
; AT FLASH [0001] AND PLACE DATA AT
; RAM[0x0D]
.INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
.sect code,rom,abs=0 ; BEGINING CODE SPACE
MAIN: ; THE BEGINING OF THE MAIN ROUTINE
LD ISPADHI,#000 ; THE HIGH ADDRESS BYTE
LD ISPADLO,#001 ; THE LOW ADDRESS BYTE
LD S,#000 ; SETUP OF THE SEGMENTATION REGISTER
LD X,00D ; THE RESULTS ARE FROM RAM[0x0D]
LD BYTECOUNTLO,#003 ; THE NUMBER OF BYTES TO WRITE
LD ISPKEY,#098 ; LOAD THE KEY
JSRB cblockr ; CALL THE ROUTINE
.END MAIN ; END OF PROGRAM
```

Example 28. SAMPLE block_read(unsigned long, unsigned int, unsigned long)—C Version

```
#include "8cbr.h";           // Include file for the COP8CBR Microcontroller
#include "flash_OP.h";               // Include file that contain the flash routines
void main(){
                                      // storage location with room for 10 elements
unsigned int storage[10] @ 0x0D;
                                     // read 3 bytes from flash, beginning at location 0x0001
                                      // and store it in storage memory array beginning at 0x0D.
block_readf(0x01,3,0x0D);
}
```

Table 28. Resource Utilization for the Command: cblockr (Block Read of the Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
BYTCONTLO ISPADHI ISPADLO	YES	YES	YES	YES	YES	DATA/RAM[X]	140/BYTE	6

4.8 cblockw—User Entry Point: Write to a Block Flash Memory

ISPADHI and ISPADLO must be set by the user prior to the jump command. The BYTECOUNTLO variable is used by the microcontroller to transfer N number of bytes (i.e, N=BYTECOUNTLO). This variable also must be set prior to the jump command. The maximum number of bytes that can be written is 16. If the number of bytes exceed 16, then the user may not be guaranteed that all of the bytes were written. The data cannot cross 1/2 page boundaries (i.e. all data must be within the 64 bytes segment for 32k devices and within 32 bytes for 4k, and 1k devices). If N=0 then the firmware will abort. Data is read from the RAM address pointed to by the X pointer. It is up to the user to setup the segmentation register. Data transfers will take place from whatever the RAM locations are specified by the segment register. However, if the X pointer exceeds the top of the segment, the firmware will begin to transfer from 0x80 (I/O and CONTROL REGISTERS) and above. After the X pointer and the BYTECOUNTLO is set, the KEY must be loaded, and a “JSRB cblockw” must be issued. For details regarding the ISPADHI, ISPADLO, and BYTECOUNTLO registers please refer to Table 21. Since this is a user command, this routine will work regardless of security (security independent). Example 29 is an example of how to use the cblockw function—assembly version. Example 30 shows the C version of the block_write() function. Table 29 shows the necessary resources needed to run the routine.

Example 29. SAMPLE cblockw (Write to a Block of Flash Memory) EXECUTION

```
                                 ; BLOCK WRITE, READ 16 BYTE(S) BEGINING
   : AT RAM[008] AND PLACE DATA BEGINING AT FLASH [0080]                                 ; ASSUME A 6
MHz CKI FREQUENCY
          .INCLD COP8CBR.INC     ; INCLUDE FILE FOR THE COP8CBR          .sect   code,rom,abs=0 ;
BEGINING CODE SPACE
MAIN:                            ; MAIN PROGRAM CODE SPACE
          LD PGMTIM,#06F         ; FOR A 6 MHz CLOCK
          LD ISPADHI,#000        ; THE HIGH ADDRESS BYTE
          LD ISPADLO,#080        ; THE LOW ADDRESS BYTE
          LD S,#000              ; SETUP OF THE SEGMENTATION REGISTER
          LD X,#008              ; THE DATA TO BE WRITTEN BEGINS AT RAM[0x08]          LD
BYTECOUNTLO,#010    ; THE NUMBER OF BYTES TO WRITE
          LD ISPKEY,#098         ; LOAD THE KEY
          JSRB cblockw           ; CALL THE ROUTINE
          .END MAIN              ;END OF PROGRAM
```

Example 30. SAMPLE block_writef(unsigned long, unsigned int, unsigned long)—C Version

```
#include "8cbr.h"; // Include file for the COP8CBR Microcontroller
#include "flash_OP.h";     // Include file that contain the flash routines
void main(){
PGMTIM=0x6F;                // Assume A 6 MHz CKI Frequency
                           // storage location with room for 20 elements
unsigned int storage[20] @ 0x0D;
                              // write 16 bytes from the storage array to flash, beginning at
                                 // location 0x0D in RAM to the starting location 0x0080 in the flash
block_writef(0x80,16,0x0D);
}
```

Table 29. Resource Utilization for the Command: cblockw (Write to a Block of Flash Memory)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	WD Serviced	JSRB/Key Required	Return Data	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
BYTECOUNTLO Data is Assumed to be in the ***RAM[X]*** Location(s)	YES	YES	YES	YES	YES	NONE	100 + 3.5*PGM-TIME/BYTE + 68/BYTE[(1)]	6

(1) Refer to Section 3.9.1 for additional information on the PGMTIM variable.

4.9 cwritebf—User Entry Point: Write a Byte to the Flash Memory

This routine requires that ISPADHI, ISPADLO, and ISPWR registers all be loaded prior to the jump. Loading the KEY and a "JSRB cwritebf" are all that is needed to complete this call. No acknowledgment will be sent back regarding the operation. For details regarding the ISPADHI, ISPADLO, and ISPRD registers please refer to Table 21. Since this is a user command, this routine will work regardless of security (security independent). Example 31 is an example of how to use the cwritebf function—assembly version. Example 32 shows the C version of the writebf() function. Table 30 shows the necessary resources needed to run the routine.

Example 31. SAMPLE cwritebf (Write a Byte to Flash Memory) EXECUTION—Assembly Version

```
                                 ; WRITE A BYTE TO THE FLASH [000A]=5
                                 ; ASSUME A 10 MHz CKI FREQUENCY
          .INCLD COP8CBR.INC     ; INCLUDE FILE FOR THE COP8CBR          .sect   code,rom,abs=0 ;
BEGINING CODE SPACE
MAIN: ; THE MAIN ROUTINE
          LD PGMTIM,#06F         ; FOR A 6 MHz CLOCK
          LD ISPWR,#005          ; LOAD THE WRITE REGISTER WITH 5
          LD ISPADHI,#000        ; LOAD THE HIGH ADDRESS BYTE
          LD ISPADLO,#00A        ; LOAD THE LOW ADDRESS BYTE
          LD ISPKEY,#098         ; LOAD THE KEY
          JSRB cwritebf          ; CALL THE FUNCTION
                                   ; TRANSFER FLASH[000B]=RAM[0x0C]          LD ISPADHI,#000        ;
LOAD THE HIGH ADDRESS BYTE
          LD ISPADLO,#00B        ; LOAD THE LOW ADDRESS BYTE
          LD A,00C               ; LOAD THE DATA INTO THE ACCUMULATOR
          X  A,ISPWR             ; SWAP IT WITH THE ISPWR REGISTER
          LD ISPKEY,#098         ; LOAD THE KEY
          JSRB cwritebf          ; CALL THE FUNCTION
          .END MAIN              ;END OF PROGRAM
```

Example 32. SAMPLE writebf(unsigned int, unsigned long, unsigned int) (Write a byte to flash memory) EXECUTION—C Version

```
#include "8cbr.h";            // Include file for the COP8CBR Microcontroller
#include "flash_OP.h";        // Include file that contain the flash routines
  void main(){                              // location in RAM
unsigned int variable @ 0x0C;

  PGMTIM=0x6F;                  // Assume a 6 MHz CKI Frequency                            // write
to location 0x000A with 5cwritebf(0x00,0x0A,5);
                                // write to location 0x00B of flash memory with the contents
 // of RAM at location 0x0C (variable)
cwritebf(0x00,0x0B,variable); // Call the exit routine
}
```

Table 30. Resource Utilization for the Command: cwritebf (Write a Byte to the Flash)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	JSRB/Key Required	Returned Data/ Location	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
ISPWR Contains the Data	YES	YES	NO	YES	NONE	168 + 3.5* PGMTIM	4

4.10 Exit—Reset the Microcontroller

This routine will cause the microcontroller to reset itself. Loading the KEY, and a "JSRB Exit" are the only actions needed to complete the call. No additional parameters need to be passed. Since this is a user command, this routine will work regardless of security (security independent). Example 33 is an example of how to use the exit function—assembly version. Example 34 shows the C version of the exit() function. Table 31 shows the necessary resources needed to run the routine.

Example 33. SAMPLE exit (reset the microcontroller) EXECUTION—Assembly Version

```
; RESET THE MICROCONTROLLER
.INCLD COP8CBR.INC ; INCLUDE FILE FOR THE COP8CBR
.sect code,rom,abs=0 ; BEGINING CODE SPACE
MAIN: ; THE MAIN ROUTINE
LD ISPKEY,#098 ; LOAD THE KEY
JSRB exit ; CALL THE FUNCTION
.END MAIN ; END OF PROGRAM
```

Example 34. SAMPLE exit() (reset the microcontroller) EXECUTION—C Version

```
#include "8cbr.h";         // Include file for the COP8CBR Microcontroller
#include "flash_OP.h";     // Include file that contain the flash routines
  void main(){reset();                  // reset the microcontroller}
```

Table 31. Resource Utilization for the Command: exit (reset the microcontroller)

Input Data	Accumulator A Used?	B Pointer Used?	X Pointer Used?	Register(s) Used?	JSRB/Key Required	Return Data	Interrupt Lock Out Cycles	Stack Usage (in Bytes)
N/A	YES	NO	NO	NO	YES	NONE	100	2

4.11 WATCHDOG™ SERVICES

The Watchdog register will be serviced periodically in order to ensure that a watchdog event has not occurred. All routines in the ISP Boot ROM incorporate automatic watchdog services. Periodically, the Boot ROM firmware will service the watchdog if the routine will take greater than the 8k upper window requirement.

Appendix A

A.1 MICROWIRE AND USER INTERFACE MECANISIMS

The following instructions will allow the user to interface directly to the routines in the Boot ROM.

A.2 JSRB—Jump Subroutine in Boot ROM

Syntax:— JSRB ADDR

Description:— The JSRB instruction causes execution to begin at the address specified within the first 256 bytes of the Boot ROM. The switch to Boot ROM is only successful if the JSRB instruction was immediately preceded by writing a valid key to the ISP KEY register. The instruction pushes the return address onto the software stack in data memory and then jumps to the subroutine address in Boot ROM. If the key has not been written, or if the key was invalid, the instruction jumps to the same address in program memory.
The contents of PCL (Lower 8 bits of PC) are transferred to the data memory location referenced by SP (Stack Pointer). SP is then decremented, followed by the contents of PCU (Upper 7 bits of PC) being transferred to the new data memory location referenced by SP. The return address is now saved on the software stack in data memory RAM. Then SP is again decremented to set up the software stack reference for the next subroutine.
Next, the values found in the second byte of the instruction are transferred to PCL. PCU is loaded with 0. The program then jumps to the program memory location accessed by PC in the Boot ROM, if the key write was successful, or in program memory if it was not.

Operation:— [SP] <- PCL
[SP - 1] <- PCU
[SP - 2]: SET UP FOR NEXT STACK REFERENCE
PC14-8 <- 00
PC7-0 <- LOADDR (SECOND BYTE OF INSTRUCTION)

Instruction	Addressing Mode	Instruction Cycles	Bytes	Hex Op Code
JSRB ADDR	Absolute	5	2	61/LOADDR

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		