Application Note

AN2305
Rev. 1, 3/2003

User Mode Monitor Access for MC68HC908QY/QT Series MCUs

By: Jim Sibigtroth
8/16-Bit Products Division
Austin, Texas

Overview

This application note explains a small user-mode program that provides access to the monitor mode in an M68HC908QY/QT series MCU without requiring the high voltage and external 9.8304 MHz oscillator normally needed to access monitor mode. This monitor access program causes the MCU to appear to be in normal monitor mode so existing M68HC908 programmers and debuggers can be used to erase or program FLASH memory and perform typical debug functions. Unlike typical ROM monitor programs, this program is very small because it uses large portions of the firmware in the on-chip MON08 ROM. This also ensures compatibility with the existing PC-based development tools.

This program would normally be programmed into the last 80 bytes of FLASH memory ($FFB0–$FFFF) of the MC68HC908QY/QT MCU in a very low cost evaluation board before it is shipped to an end user. Such an evaluation board can be connected to the serial port of a host PC for FLASH erase and programming operations or for application system debugging. This program is block protected so that it cannot be erased accidentally. Traditional monitor-mode tools that provide high voltage to the PTA2/$\overline{\text{IRQ}}$ pin, and also provide an external 9.8304 MHz oscillator source, can erase the entire FLASH memory, including the last 80 locations where this monitor access program is located.

Features

- Makes the MCU operate as if in normal monitor mode
- Adjusts the on-chip oscillator to allow 9600 baud serial communications
- Supports half-duplex serial communication through PTA0 pin
- Block protected to prevent accidental erasure
- Redirects user interrupts through a pseudo-vector mechanism
- Monitor bypassed if PTA2/$\overline{\text{IRQ}}$ pin is high (or unconnected) at reset
- Defaults to monitor if no user reset vector programmed
- Initializes CONFIG1 based on user-supplied nonvolatile value (if valid)
- Forces LVI on and COP off if no CONFIG1 value supplied

Hardware Requirements

This program is designed to minimize interference with user applications. The monitor uses the PTA0 I/O pin for bidirectional serial communications with the serial port of a host PC. The PTA2/$\overline{\text{IRQ}}$ pin is used briefly during power-up or reset to choose monitor control or user application control.

The PTA0 pin is connected to an RS232 level shifter like that provided on M68HC908 ICS boards to allow half-duplex communications with a host PC through a normal serial port. See **Figure 1**.

The PTA2/$\overline{\text{IRQ}}$ pin is usually connected through a normally-open pushbutton switch to ground. If this pin is high at reset (or not connected) and the user has programmed the reset pseudo-vector to a non-$FFxx value, the monitor program jumps to the user's reset pseudo-vector at $FDFD and the control passes to the user application program. If PTA2/$\overline{\text{IRQ}}$ is low at power-on reset (typically achieved by pressing the pushbutton switch to ground the pin), or the reset pseudo-vector isn't programmed, the monitor program gains control. The PTA2/$\overline{\text{IRQ}}$ pin can still be used in an application program because the monitor never refers to this pin after checking the level just after reset.

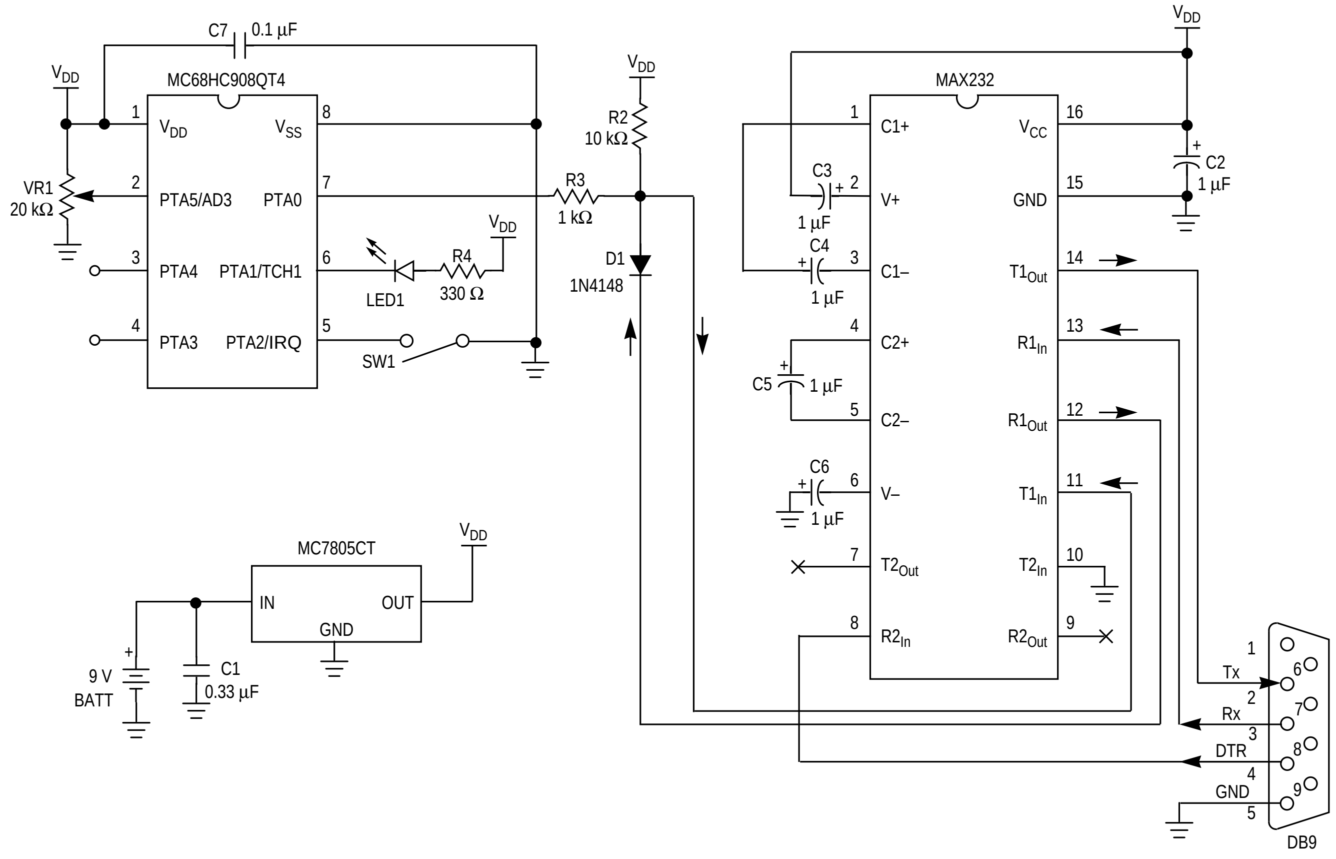

Figure 1. Low-Cost Evaluation Board Schematic

Software Description

In addition to the 4K bytes of user FLASH in the block from $EE00 through $FDFF, there are 80 bytes of FLASH at $FFB0 through $FFFF. Some of these high-address FLASH locations are used for reset and interrupt vectors and a few other locations are reserved for special purposes. Location $FFBE is the FLBPR register which controls a block protection feature in the MC68HC908QY/QT series MCUs. In this monitor access program, we program this location to $FE in order to prevent accidental erasure of this block of memory while allowing you to erase or program the 4K byte block of user FLASH as needed. Location $FFC0 is reserved for a nonvolatile trim value for the on-chip oscillator. Before the monitor access program is programmed into the FLASH, the internal oscillator frequency is measured to determine a trim value that will adjust the internal oscillator to the correct frequency so standard 9600 baud communications can be supported. This trim value is programmed into location $FFC0 where the monitor program reads it and stores it into the appropriate control register during reset initialization.

The monitor access program fits into three small blocks of FLASH at \$FFB0–\$FFBD, \$FFC2–\$FFDD, and \$FFE2–\$FFF1. By placing this monitor access program in these locations, we leave the entire 4K byte block of user FLASH available for user application programs. The monitor access program uses the reset and SWI vectors. The remaining six interrupt vectors are redirected to pseudo-vector jump instructions at the end of the 4K byte block of user FLASH. Although the reset vector points to the start of the monitor access program, a user reset pseudo-vector jump instruction is included at the end of the 4K byte user FLASH, in case the PTA2/$\overline{\text{IRQ}}$ pin was high at reset, indicating the user's reset initialization code should be executed instead of the monitor program. The monitor program uses the SWI vector to manage entry into monitor mode and for breakpoints, so user application programs should not use SWI instructions.

MON08 ROM Information

Providing access to the MON08 ROM isn't quite as simple as just using a JMP instruction after reset in normal user mode. The MON08 ROM in the MC68HC908QY/QT series devices supports two primary modes of operation with several options controlled by the state of certain I/O pins during power-on reset. The whole point of this user-mode monitor access program is to avoid extra requirements and options to minimize interference between the monitor and a user application program. This also eliminates the need for a more expensive debug interface such as the P&E[1] MON08-Cyclone stand-alone programmer or the MON08-MultiLink debug interface pod.

The two primary modes of operation for MON08 are traditional monitor mode and forced monitor mode. Traditional monitor mode requires a higher-than-V_{DD} voltage (called V_{tst}) on the PTA2/$\overline{\text{IRQ}}$ pin at power-on reset and uses an external 9.8304 MHz external clock applied to PTA5/OSC1. In addition, traditional monitor mode in MC68HC908QY/QT series devices requires that PTA1 be pulled up and PTA4 be pulled down to control factory options in the MON08 ROM. Forced monitor mode is used to allow easier monitor mode access to program a blank MCU (for instance, at the end of a product assembly line). Forced monitor mode is entered if the reset vector locations at \$FFFE and \$FFFF are erased (\$FF) at power-on reset. Depending on the level on the PTA2/$\overline{\text{IRQ}}$ pin, forced monitor mode uses an external 9.8304 MHz oscillator ($\overline{\text{IRQ}}$ high) or the internal oscillator ($\overline{\text{IRQ}}$ low).

When the MCU enters monitor mode from power-on reset, the reset and SWI vectors are modified by hardware logic so the reset vector is relocated to \$FEFE and the SWI vector is relocated to \$FEFC. \$FEFC through \$FEFF are located in the on-chip MON08 ROM so the monitor program has control in response to reset or SWI interrupts (breakpoints also use the SWI vector).

We don't want to enter the MON08 ROM at its reset vector because then we would still need to control the level on the PTA2/$\overline{\text{IRQ}}$ pin throughout a

1. P&E Microcomputer Systems' Web site is http://www.pemicro.com

debugging session (so the monitor would know to use the internal oscillator version of the SWI ISR instead of the external 9.8304 MHz version). This would effectively prevent you from using the PTA2/$\overline{\text{IRQ}}$ pin in a user application. The user-mode monitor access program avoids this problem by programming the normal user SWI vector to point directly to the internal oscillator version of the MON08 SWI service routine. Another reason not to use the MON08 firmware directly is that this would not permit a user to reset directly to the start of their application rather than entering the monitor mode.

After power-on reset into monitor mode, the PC-based development tool sends a sequence of eight security bytes which must match an 8-byte security code in FLASH. Otherwise, the FLASH memory is disabled to prevent unauthorized access. If the security sequence fails to match, the development system can choose to bulk-erase the whole FLASH memory. After the FLASH memory is erased, the security code is known to be all $FF characters. This allows the tool to cycle the power and supply this known-good code to pass security and access the MCU. After the security sequence, regardless of whether it passed or failed, the MON08 ROM firmware sends a break character and enters a loop to wait for subsequent commands.

The user-mode monitor access program described in this application note receives and echoes the eight security codes just as the MON08 firmware would, however the monitor access program does not check the values against the values in FLASH memory. Instead, the monitor access program just emulates the security checking sequence and FLASH memory is enabled regardless of the codes received. This simplifies access to the monitor mode for evaluation purposes by bypassing the security feature. If you want to engage security for a user application, you must use traditional monitor mode to override block protection and erase the whole FLASH memory and program your own application code into the FLASH memory.

Monitor Access Program Details

This section describes the small monitor access program in detail. This program is executed every time the MCU is reset in normal user mode.

On-Chip Oscillator Trim

The MC68HC908QY/QT series MCUs have an on-chip oscillator which requires no external components. The frequency of this oscillator is based on an internal band gap reference and is controlled by a binary weighted capacitor array. An 8-bit register (OSCTRIM @ $0038) controls the total effective capacitance and can adjust the frequency (actually the period) of this oscillator by about ±25% from a center frequency of about 12.8 MHz. This center frequency varies a little from lot to lot and device to device due to small process variations. At reset, OSCTRIM is initialized to $80. User software (or in this case, the monitor access program) reads a trim adjustment value from nonvolatile memory ($FFC0) and stores it to the OSCTRIM register to set the on-chip oscillator to just the right frequency to support serial communications at a standard 9600 baud rate.

The ideal way to initially set this oscillator trim value is to measure the frequency in the final application system, determine the best trim value (usually through an iterative process), and program this value into FLASH memory at location $FFC0. The trim value could also be determined during Freescale final test, however this value would not be as accurate because the factory test environment has much more noise than a typical application system. The factory trim option also has the drawback that it is lost if the FLASH memory is ever erased so you would still need the ability to determine a trim setting after the MCU is installed in the application system. Programming tools such as the P&E MON08-Cyclone and MON08-MultiLink pods can adjust the trim and program the value into the FLASH as easily as they can erase or program the FLASH memory.

In the monitor access program described in this application note, the trim value must be determined before the block protect value is programmed because the block protect setting prevents erase or programming of locations within the protected block (which includes the nonvolatile trim setting at $FFC0).

In a typical evaluation system, the on-chip oscillator can maintain an accurate enough frequency to support serial communications at a standard 9600 baud rate. The oscillator frequency is relatively insensitive to changes in supply voltage. In an evaluation board that derives 5 V V_{DD} by regulating the voltage from a 9 V transistor battery, the on-chip oscillator frequency will remain relatively steady even as the battery voltage drops and V_{DD} falls out of regulation. Typical evaluation systems also operate within a significantly narrower temperature range than that allowed by the MCU specification. System noise has a more significant effect on the trim value so it is important to provide good bypass capacitance directly from V_{DD} to V_{SS} as close as possible to the MCU. Trimming the on-chip oscillator in one system and then moving it to a different system with significantly different bypassing can result in oscillator frequency errors that are big enough to cause serial communication at standard baud rates to break down.

PTA2/$\overline{\text{IRQ}}$ Selects Monitor or User Reset

Shortly after reset, the monitor access program makes a choice between going to the monitor or going to the user's application program. This allows a user to develop an application program with the debug monitor and then use their program in an application system without removing the monitor access program or modifying their application program in any way. The decision is based on the state of the PTA2/$\overline{\text{IRQ}}$ pin. If PTA2 is high (as it would be in a typical application system), the monitor access program fetches the user's reset pseudo-vector and jumps to the application program instead of entering the monitor program. If PTA2 is driven low at reset, the monitor access program jumps into the monitor program. If PTA2 is high at reset, but the high-order byte of the user's reset pseudo-vector (at $FDFE) is $FF (not programmed), the monitor access program passes control to the monitor rather than running away to a non-existent application program.

The PTA2/$\overline{\text{IRQ}}$ pin and the low level at reset, were chosen to force entry into the monitor because this is similar to the mechanism used to select forced monitor mode in a blank part. Forced monitor mode is provided as a convenience for users who wish to buy trimmed blank parts from Freescale, assemble them onto a printed circuit board, and then program application code into the FLASH memory at the end of their production process.

Write-Once CONFIG1 Register

For debug operations with the monitor, the COP watchdog timer must be disabled by writing 1 to the COPD bit in the CONFIG1 register. It is also recommended that the low-voltage inhibit (LVI) function be powered up and enabled (LVIRSTD=LVIPWRD=0) to reset the MCU whenever V_{DD} drops below legal voltage levels. CONFIG1 is a write-once register that contains several control bits that configure other aspects of the COP and LVI functions as well as STOP mode settings. In order to allow some user control of these settings, the monitor access program supports a location in the user FLASH memory, just before the user's pseudo-vectors, where you can program the value you would like for the CONFIG1 register. When the monitor access program is executed after reset, this user value is read from $FDEA and checked to see if it meets the minimum requirements to support proper monitor operation (LVIRSTD=LVIPWRD=0 and COPD=1). If the value is acceptable, it is written to CONFIG1. If $FDEA has not been programmed, or has an unacceptable value, a default value of $49 is written to CONFIG1.

Security Check Emulation

Existing development software such as Metrowerks[1] CodeWarrior and P&E's Prog08 and ICD08 tools are set up to interface with the traditional M68HC08 monitor mode (MON08). As these tools attempt to establish communications with a target MCU, they send a sequence of eight security bytes serially to the target MCU. In the case of MON08, these bytes are compared to specific security key locations in the FLASH memory. As each code is sent, it is echoed once as a side-effect of the half-duplex serial connection (transmit and receive signals are both connected to the same MCU pin). MON08 firmware echoes each character again to act as an acknowledge to the host personal computer (PC). After the security code sequence is complete, MON08 either enables the FLASH (if the codes matched) or disables it to prevent unauthorized access to FLASH memory contents. The enable/disable mechanism is associated with the special monitor mode (when the PTA2/$\overline{\text{IRQ}}$ pin has a high V_{tst} voltage applied), and does not apply in normal user mode.

In the case of the monitor access program described in this application note, the target MCU is operating in normal user mode so there is no need to enable the FLASH memory. However, we don't want to change any of the development software so we emulate the security sequence by receiving and echoing eight security code bytes just as MON08 firmware would. Unlike MON08, the monitor access program doesn't check the values received during

1. Metrowerks' Web site is at http://www.metrowerks.com

this sequence. After the eighth character is echoed, the monitor access program executes an SWI instruction to start the monitor firmware in the MON08 ROM. The SWI instruction causes the CPU registers to be stored on the system stack, the interrupt mask (I-bit) in the condition code register (CCR) is set to inhibit interrupts, and execution continues at the address indicated by the SWI vector at $FFFC:FFFD.

Vector Redirection

This program resides in the same area as the interrupt vectors. This area is block protected to prevent accidental erasure of the monitor access program, so the user cannot program the interrupt vectors to point to their own interrupt service routines. A traditional vector redirection mechanism is provided by pointing the actual vectors (at $FFDE through $FFFF) at pseudo-vector locations in unprotected FLASH memory at $FDEB through $FDFF. When an interrupt occurs, the MCU's normal interrupt mechanism fetches the contents of the real vector, which points to a corresponding pseudo-vector in the unprotected FLASH. The pseudo-vector locations are programmed (by the user) with a jump (JMP) instruction to the user's interrupt service routine.

This mechanism adds about three extra bus cycles to the interrupt response time (to execute the extra jump to the user's interrupt service routine). It also uses 21 bytes at the end of the user's unprotected FLASH ($FDEB–$FDFF) for the seven pseudo-vector JMP instructions. The real reset vector points at the start of the monitor access program, but a reset pseudo-vector is included to provide a pointer to the start of the user's application program in the case that PTA2/$\overline{\text{IRQ}}$ is high after reset. The real SWI vector points at the SWI interrupt service routine (internal oscillator version) in the MON08 firmware ROM. No SWI pseudo-vector is provided and you should not use SWI instructions in your application program (SWI is reserved for monitor program use).

Miscellaneous Program Details

This section provides additional information about particular instructions and techniques that were used in the monitor access program. This program is somewhat different than a typical application program because it is designed to fit into small unused spaces around the vectors. The program starts at $FFB0 but the FLBPR register and the nonvolatile TRIMLOC values are located at $FFBE—$FFC1 so a "BRA toFFC2" instruction is used to skip over these locations. Another "BRA toFFE2" instruction is used to skip over the interrupt vectors for the ATD and KBD systems. Application programmers wouldn't usually go to so much trouble to fit small (less than 16 bytes each) blocks of code into these discontinuous areas. The monitor access program was squeezed into these spaces so that the whole 4K byte block of FLASH at $EE00–$FDFF could be left for user application programs.

The "BSET PTAPUE2,PTAPUE" instruction is included to enable the internal pullup resistor on the PTA2/$\overline{\text{IRQ}}$ pin. After reset, this pin is initially configured to be a general-purpose input pin with the pullup disabled. In a typical evaluation board, this pin would have a pushbutton switch that provides a momentary closure to ground. In order to avoid using an external pullup resistor (which would add cost and assembly time to the circuit board), the internal pullup resistor is enabled with this instruction in the monitor access program. Later, a “BRCLR PTA2,PTA,MonStart” instruction is used to decide whether to jump to the user application program or continue into the monitor.

The “BCLR PTA0,PTA” instruction is needed to clear the port data register bit for the PTA0 pin before calling the IGetPut subroutine in the MON08 ROM the first time. IGetPut uses the PTA0 pin for bidirectional serial communications. During receive operations, PTA0 is configured as a high-impedance input and the value in the PTA0 bit of the port A data register is unimportant. During transmit operations, the level on PTA0 is controlled by writing zero or one to the data direction control bit for PTA0 (and the value in the PTA0 data register is assumed to be zero). When the data direction control is written to zero, the pin becomes a high-impedance input and the pin is pulled up through an external pullup resistor. When the data direction control is written to one, the pin becomes an output and because the PTA0 data register is zero, the pin is driven low. The reset state of the PTA0 data register is uninitialized so the BCLR instruction is needed to make sure it is zero.

In the sequence that checks for a valid user reset pseudo-vector, the program uses a “LDA AltRESET+1” instruction to check the second byte of the jump instruction at the user reset pseudo-vector locations. Since we are only trying to detect the difference between an erased pseudo-vector ($FF:FF:FF) or a jump to a valid address ($CC:hh:ll) we only need to check the high half of the address of the jump instruction. Any valid jump instruction would point to an address in the 4Kbyte user FLASH ($EE00–$FDFF) so we know the high half of the address would be a value between $EE and $FD rather than the erased value $FF.

User Programs

This section explains differences in the way you would write an application program to co-exist with the monitor access program compared to the way you would write programs for normal systems that do not include the monitor access program. The section begins with a discussion of how the monitor access program alters some registers compared to their raw reset states before going to the user’s pseudo-reset vector. Next, we will discuss how to write your programs to use the pseudo-vector mechanism which redirects interrupt vectors. Finally, we discuss ways to use $\overline{\text{IRQ}}$ requests to cause an asynchronous break from a user application program to monitor control.

Reset State Details

Unlike more sophisticated emulation systems, the monitor access program emphasizes ease-of-use and lowest possible cost rather than exact duplication of raw reset conditions. Our goal was to make it as easy as possible to get started with application programs. Because of this, a few control registers are slightly different when the application program starts compared to the way they would be in a system that did not include the monitor access program.

Normally after reset, the OSCTRIM register is initialized to $80. The monitor access program replaces this value with a trim adjustment value from $FFC0 to trim the on-chip oscillator to just the right frequency to support serial communication at a standard 9600 baud rate. A user application program can write a new value to this register, but this is likely to cause monitor communications to fail.

The monitor access program sets the PTAPUE2 bit in the PTAPUE register to enable an internal pullup resistor on the PTA2/$\overline{\text{IRQ}}$ pin. Since the monitor access program is intended for small evaluation boards that have a push-button switch connected from PTA2/$\overline{\text{IRQ}}$ to ground, it is unlikely that a user would not want the internal pullup resistor enabled for this pin.

The PTA0 data bit in the PTA data register is normally uninitialized after reset, but the monitor requires this bit to be zero so the monitor access program clears it. A user application program can write other values to the PTA register, but if PTA0 is not zero, monitor communications cannot function properly. For all practical purposes, the PTA0 pin is dedicated to the monitor communication function in systems that have the monitor access program, so this should not be a significant limitation. In some cases a user application program could use the PTA0 pin for serial communications, but in that case, PTA0 would still need to be zero.

If the monitor is entered directly after reset, the CONFIG1 register (which is a write-once register) is written with a user supplied non-volatile value from user FLASH memory at $FDEA. If no valid user value is provided, a default value is written to CONFIG1 to ensure that the COP watchdog timer is disabled. If the user application program is entered directly from reset (by detecting a high level on PTA2/$\overline{\text{IRQ}}$), CONFIG1 is not written and the user application program can write any value to CONFIG1. In this case, if the monitor is entered later due to an SWI or $\overline{\text{IRQ}}$ interrupt, and the COP watchdog is not disabled in CONFIG1, the monitor will fail. For compatibility with the monitor, your application program should ensure that the COPD bit in CONFIG1 is written to 1 to disable the COP watchdog system.

User Interrupts

Since the interrupt vectors are located in the same block-protected area of FLASH as the monitor access program, you cannot erase and reprogram them to point at your own interrupt service routines. A pseudo-vector mechanism is provided to allow you to use interrupts in an application program. The vectors for on-chip peripherals are pointed at specific locations at the end of the unprotected FLASH where you can program jump instructions to your application interrupt service routines.

The SWI is dedicated to monitor functions so no corresponding pseudo-vector is provided. User application programs should not use SWI instructions. The reset vector points to the start of the monitor access program, but a user reset pseudo-vector is provided in case PTA2/$\overline{\text{IRQ}}$ is high at reset which causes the monitor access program to jump directly to the user application program instead of entering the monitor.

Table 1 shows the pseudo-vector addresses for each user interrupt source. To use these interrupts, the user application program would place JMP instructions to the application interrupt service routines at the corresponding interrupt pseudo-vector locations.

Table 1. Pseudo-Vector Addresses

Label	Address	Interrupt Source
`AltADC`	\$FDEB	A to D Converter
`AltKBD`	\$FDEE	Keyboard Wakeup
`AltTOF`	\$FDF1	Timer Overflow
`AltTCH1`	\$FDF4	Timer Channel 1
`AltTCH0`	\$FDF7	Timer Channel 0
`AltIRQ`	\$FDFA	$\overline{\text{IRQ}}$
`AltRESET`	\$FDFD	$\overline{\text{RESET}}$

Special $\overline{\text{IRQ}}$ Considerations

It is often desirable to be able to stop an application program during debugging so the monitor program can be used to check the contents of memory or CPU registers. This section discusses how an interrupt such as $\overline{\text{IRQ}}$ can be used for this. The monitor access program does not include this feature because that would limit the use of the PTA2/$\overline{\text{IRQ}}$ pin for other user application purposes. $\overline{\text{IRQ}}$ can still be used to break to the monitor program if the user application uses the PTA2/$\overline{\text{IRQ}}$ pin as an $\overline{\text{IRQ}}$ input and the application program includes the necessary support. The simplest implementation would be the case where PTA2/$\overline{\text{IRQ}}$ was dedicated for monitor use only, but we will also discuss a more complex case where $\overline{\text{IRQ}}$ is used for the application and as a monitor break input.

The first thing we need to do is to configure the PTA2/$\overline{\text{IRQ}}$ pin to act as the $\overline{\text{IRQ}}$ interrupt input. This is controlled by the write-once CONFIG2 register which also includes bits to configure the PTA3/$\overline{\text{RESET}}$ pin and to configure oscillator options. In the case of a system that includes the monitor access program, we would want to set the bits in CONFIG2 as described in **Table 2**.

Table 2. CONFIG2 Control Settings

Bit Name(s)	Bit Number(s)	Setting	Comments
`IRQPUD`	7	`1`	Enable pullup for PTA2/$\overline{\text{IRQ}}$ pin
`IRQEN`	6	`1`	Enable $\overline{\text{IRQ}}$ function on PTA2/$\overline{\text{IRQ}}$ pin
`R`	5	`0`	Unused bit
`OSCOPT1:0`	4:3	`0:0`	Internal oscillator option
`R:R`	2:1	`0:0`	Unused bits
`RSTEN`	0	–	0 or 1 depending on application

So depending on whether the PTA3/$\overline{\text{RESET}}$ pin will be used as a general purpose I/O pin or as a $\overline{\text{RESET}}$ input, you would write \$C0 or \$C1 to CONFIG2. Don’t use BSET or BCLR instructions for this because you can only write to this register one time (additional writes are ignored).

Next, usually after you have finished other initialization, you must enable interrupts by clearing the I bit in the condition codes register (CCR). The $\overline{\text{IRQ}}$ break mechanism only works while the I bit is clear. In application programs, there are some cases where the I bit would be set temporarily, and the break-to-monitor function will not operate until the I bit is cleared again. Breakpoints, which use the non-maskable SWI mechanism and vector, do not require I to be clear in order to break to the monitor.

Examples of application cases where the I bit would be set include:

- When an interrupt is serviced, the I bit is set when the interrupt is honored (as you vector to the ISR), and the I bit is normally cleared during execution of the RTI instruction at the end of the ISR.
- When you program or erase FLASH memory, the I bit should be set to prevent any interrupts while the FLASH memory is disabled from the memory map. Typically you would set I, call a subroutine that performs the desired FLASH changes, and then clear the I bit to allow interrupts.
- In rare cases you may want to mask interrupts during some critical timing sequence.

In the simplest case where $\overline{\text{IRQ}}$ is not used for any other application functions, you can program the $\overline{\text{IRQ}}$ pseudo-vector to JMP to IMonSwi (\$2CF9). This is the address of the internal oscillator version of the SWI service routine in the MON08 monitor ROM.

The situation is more complicated if $\overline{\text{IRQ}}$ is used for other application purposes. In that case, you must determine some way to distinguish between a user $\overline{\text{IRQ}}$

and a request to enter the monitor. A simple demonstration program is supplied with one very low cost evaluation board that uses the monitor access program. In this demo program, $\overline{\text{IRQ}}$ is used to switch between a mode where a potentiometer controls the blinking speed of an LED and a second mode where the same potentiometer controls the brightness of the LED. That demonstration program distinguishes between the application mode select function and a request to enter the monitor based on how long the $\overline{\text{IRQ}}$ pin remains low (how long the switch is pressed). If the switch is pressed and released, it is interpreted as a request to toggle between LED blinking mode and LED brightness mode. If the $\overline{\text{IRQ}}$ push-button is pressed and held for a relatively long time, it is interpreted as a request to exit to the monitor program.

Monitor Access Program Listing

This section includes the complete listing for the monitor access program. This program manages initialization, vector redirection, and entry into the monitor program located in the MON08 firmware ROM. Command decode and processing is handled by the MON08 ROM. The only differences between this monitor access program and traditional monitor mode are the reset initialization and the processing of interrupts related to the SWI vector. This monitor access program skips the reset initialization portion of the MON08 ROM program so that we don't have to process all of the factory test options that use some of the MCU pins. Skipping the initialization also allows us to use the trimmed internal oscillator rather than an external 9.8304 MHz oscillator (this also simplifies the use of more MCU pins for user applications). The SWI vector is pointed directly at the internal oscillator version of the SWI ISR rather than going through a test of the level on PTA2/$\overline{\text{IRQ}}$ to choose between an internal oscillator version and an external 9.8304 MHz oscillator version. This leaves the PTA2/$\overline{\text{IRQ}}$ pin available for the user application.

Typically, a user would not be concerned about the details of monitor commands because a free (or very low cost) development software package would usually be used with this system. The development software running on a host personal computer takes care of all monitor command details and offers a powerful graphic user interface for debugging and FLASH memory programming.

Listing 1. Monitor Access Program Listing

```
Metrowerks HC08-Assembler
(c) COPYRIGHT METROWERKS 1987-2002

Loc  Obj. code   Source line
---- ---------   -----------
                 ;.pagewidth   98t                 ;Set Page Width (For P&E)
                 ;**********************************************************************
                 ;* Title: UserMonQT4.asm                     Copyright (c)  2002
                 ;* User Mode Monitor for MC68HC908QT/QY Series        Jim Sib  7/20/02
                 ;* For additional information see Application note AN2305/D
                 ;*
                 ;* This code is programmed into the flash memory to allow entry into
                 ;* monitor mode (Mon08) without any special levels or Vtst.
                 ;* This will allow a very low cost target system to be programmed or
                 ;* debugged directly from P&E or CodeWarrior through a simple serial
                 ;* cable.
                 ;* Like other ROM monitors, this code requires a pseudo-vector
                 ;* mechanism to redirect vectors to locations in unprotected flash
                 ;* below the area of this small code segment.
                 ;**********************************************************************
                 ;             base   10t          ;change default to decimal (For P&E)

                 ;*** Equates for registers and routines in MC68HC908Qx ***************
     0000 0000   PTA:          equ    $0000         ;Port A register
     0000 0000   PTA0:         equ    0             ;bit # for PTA0
     0000 0002   PTA2:         equ    2             ;bit # for PTA2/IRQ pin
     0000 000B   PTAPUE:       equ    $000B         ;Port A pullup enable bits
     0000 0002   PTAPUE2:      equ    2             ;bit # for PTA2 pullup enable
     0000 001F   CONFIG1:      equ    $001F         ;CONFIG1 register
     0000 0000   COPD:         equ    0             ;COPD in bit #0
     0000 0001   mCOPD:        equ    %00000001     ;bit mask for COPD
     0000 0049   InitConfig1: equ    %01001001     ;default initialization for monitor
                 ;                     ||||||||      CONFIG1 is a write-once register
                 ;                     |||||||+-COPD    - 1 disable COP watchdog
                 ;                     ||||||+--STOP    - 0 disable STOP instruction
                 ;                     |||||+---SSREC   - 0 4096 cycle STOP recovery
                 ;                     ||||+----LVI5OR3 - 1 set LVI for 5v system
                 ;                     |||+-----LVIPWRD - 0 enable power to LVI system
                 ;                     ||+------LVIRSTD - 0 enable reset on LVI trip
                 ;                     |+-------LVISTOP - 1 enable LVI in STOP mode
                 ;                     +--------COPRS   - 0 long COP timeout
     0000 0038   OSCTRIM:      equ    $0038         ;working ICG trim setting
     0000 FFC0   TRIMLOC:      equ    $FFC0         ;nonvolatile trim value (flash)
     0000 FFBE   FLBPR:        equ    $FFBE         ;flash block protect reg (flash)

     0000 2D6B   IGetPut:      equ    $2D6B         ;Mon08 GetPut routine for int osc
     0000 2CF9   IMonSwi:      equ    $2CF9         ;Mon08 SWI ISR for int osc

     0000 FDEA   UConfig1:     equ    $FDEA         ;nonvolatile user value for CONFIG1
                 ;* Monitor checks this user supplied value and uses it if it has been
                 ;* programmed with a CONFIG1 value that is compatible with the monitor
                 ;* which requires COP to be disabled and LVI to be powered and enabled
                 ;* If these conditions are not met, the monitor uses InitConfig1 value

                 ;*** Equates to setup alternate vectors ******************************
                 ;* actual vectors will pass control to these locations.
                 ;* user code would include jump instructions at these locations that
                 ;* jump to the user's interrupt service routines, for example
```

```
                        ;*           org    AltADC
                        ;*           jmp    ADCisr
                        ;*           jmp    KBDisr
                        ;*
                        ;*********************************************************************
     0000 FDEB          AltADC:      equ    $FDEB          ;Alternate ADC interrupt vector
     0000 FDEE          AltKBD:      equ    $FDEE          ;    '     KBD wakeup '    '
     0000 FDF1          AltTOF:      equ    $FDF1          ;    '     TOF        '    '
     0000 FDF4          AltTCH1:     equ    $FDF4          ;    '     Timer Ch.1 '    '
     0000 FDF7          AltTCH0:     equ    $FDF7          ;    '     Timer Ch.0 '    '
     0000 FDFA          AltIRQ:      equ    $FDFA          ;    '     IRQ        '    '
     0000 FDFD          AltRESET:    equ    $FDFD          ;    '     RESET      '    '

                                     org    FLBPR          ;flash block protect location
FFBE FE                              fcb    $FE            ;protect this code, FLBPR,& vectors

                                     org    $FFB0          ;start of upper 96 bytes of flash
FFB0 C6 FFC0        UMonReset:       lda    TRIMLOC        ;get nonvolatile trim value
FFB3 B7 38                           sta    OSCTRIM        ;set working trim value in ICG
FFB5 11 00                           bclr   PTA0,PTA       ;initialize PTA0 for serial comms
FFB7 14 0B                           bset   PTAPUE2,PTAPUE ;enable int pullup on PTA2/IRQ pin
FFB9 C6 FDFE                         lda    AltRESET+1     ;MSB of user reset vector
FFBC 20 xx                           bra    toFFC2         ;around FLBPR and trim values
                    ;* FLBPR and oscillator trim value are located at $FFBE-FFC1
                                     org    $FFC2          ;immediately after nonvolatile trim
FFC2 4C             toFFC2:          inca                  ;test for $FF->$00
FFC3 27 06                           beq    MonStart       ;if reset vector blank, go MonStart
FFC5 05 00 03                        brclr  PTA2,PTA,MonStart ;If IRQ=0 @ reset, enter monitor
FFC8 CC FDFD                         jmp    AltRESET       ;IRQ high so start user program
FFCB C6 FDEA        MonStart:        lda    UConfig1       ;get user's value for CONFIG1
FFCE 97                              tax                   ;save a copy
FFCF A4 31                           and    #%00110001     ;check for LVI on, COP reset off
FFD1 A1 01                           cmp    #%00000001     ;LVIRSTD=LVIPWRD=0, COPD=1
FFD3 27 02                           beq    skipload       ;if so value in X is OK
FFD5 AE 49                           ldx    #InitConfig1   ;else force default settings
FFD7 BF 1F          skipload:        stx    CONFIG1        ;write-once register to set config
FFD9 20 xx                           bra    toFFE2         ;around ADC and KBD vectors
                    ;* ADC and KBD vectors are located at $FFDE-FFE1
                                     org    $FFE2          ;unused block in vectors
FFE2 AE 08          toFFE2:          ldx    #8             ;loop count for 8 characters
FFE4 CD 2D6B        LoopSec:         jsr    IGetPut        ;get and echo a security character
FFE7 5B FB                           dbnzx  LoopSec        ;get and ignore 8 characters
FFE9 83                              swi                   ;start Mon08

                                     org    $FFDE          ;beginning of QY4/QT4 vectors
FFDE FDEB           VADC:            fdb    AltADC         ;point to alternate vectors
FFE0 FDEE           VKBD:            fdb    AltKBD
                    ;* There is a gap from $FFE2-FFF1 that is used for user monitor code
                                     org    $FFF2          ;skip unused vector locations
FFF2 FDF1           VTOF:            fdb    AltTOF
FFF4 FDF4           VTCH1:           fdb    AltTCH1
FFF6 FDF7           VTCH0:           fdb    AltTCH0
FFF8 FFFF                            fdb    $FFFF          ;fill unused location with $FFFF
FFFA FDFA           VIRQ:            fdb    AltIRQ
FFFC 2CF9           VSWI:            fdb    IMonSwi        ;SWI service routine in Mon08 ROM
FFFE FFB0           VRESET:          fdb    UMonReset      ;to start of user mode monitor
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor