Application Note

AN2690
03/2004

Low Frequency EEPROM Emulation on the MC68HLC908QY4

By Alan Devine
8/16-Bit Division
East Kilbride, Scotland

Freescale Semiconductor, Inc.

Introduction

To avoid the cost of using external EEPROM devices, the FLASH on Freescale microcontrollers can be used in most applications to emulate EEPROM.

Techniques for emulating EEPROM on the MC68HLC908QY4 family are discussed in application note AN2346. These techniques require the MCU to be running with a minimum bus frequency of 1 MHz. This application note discusses how to emulate EEPROM on the MC68HLC908QY4 when an external 32768 Hz crystal oscillator is used to drive the application.

As the FLASH on the device requires a minimum program bus clock frequency of 1 MHz, the crystal clock is too slow to successfully program the FLASH array. A potential solution is to run the device from the internal oscillator (typically 4 MHz) when programming the FLASH, and then to switch back to the external crystal for the main application. However, due to the security implemented on the MC68HLC908QY family, it is possible to write to the clock selection register (change clock source) only once after reset. Thus, in order to switch to the internal oscillator when running on the crystal clock, a reset of the MCU must be forced. This can be done using one of the following methods: Illegal Opcode, Illegal Address, COP Timeout and External Reset; however, resuming execution of the application is more difficult, as the reset vector is fetched and all modules and registers are reset to their default state, which could be a limitation in some applications.

Two general methods are presented; Method 1, where a reset is forced at the beginning of the loop; and Method 2, where a reset can be forced anywhere in the loop. The advantages and disadvantages of the two methods are listed. Although the particular methods may not fit the specific application exactly, it should be possible to apply one of them. Before these methods are examined, the techniques of forcing a reset are described.

© Freescale Semiconductor, Inc., 2004. All rights reserved.

For More Information On This Product,
Go to: www.freescale.com

NOTE: *The appendix at the end of this document describes a sample application that implements Method 1. It comprises a description of the application, a block diagram of the hardware, and full details of the software.*

Forced Reset Operation

This can be done using one of the following methods: Illegal Opcode, Illegal Address, COP Timeout and External Reset. In the examples presented, the Illegal Opcode Reset is used and is forced by executing the STOP instruction when the STOP bit in the CONFIG1 is cleared. This method can only be used when the application program does not use STOP mode. A more general purpose method would be to use an actual illegal opcode, which can be easily generated.

The source that caused the reset can be determined by checking the flags in the SIM Reset Status Register (SRSR). i.e. An Illegal Opcode Reset sets the ILOP bit (bit 4) in the SRSR register. To distinguish between a forced reset and an actual illegal opcode, a specific bit pattern should be written to RAM prior to forcing the reset and this pattern should be verified after the ILOP flag is detected.

A reset causes the following actions to occur:

- Reset vector is fetched.
- Data registers are set to default conditions. (For example, in general, outputs default to inputs, which could affect the application.)
- Internal registers are reset:
  - Accumulator (A) - XXXXXXXX
  - Index Register (H:X) - 00000000XXXXXXXX
  - Stack Pointer (SP) - 0000000011111111
  - Program Counter (PC) - Loaded from $FFFE - $FFFF
  - Condition Code Register (CCR) - X11X1XXX
- Peripherals are set to default conditions (usually switched off).

Forcing a reset could be an issue in some applications, as register values are changed. It is very important to put the application into a known state before forcing the reset, and to restore the registers to their known state, as quickly as possible. This can be achieved by copying critical register values to RAM, and then restoring the values after the forced reset.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Method 1 — Reset Forced at Start of Main Loop

This is the simpler of the two methods, because the reset is always forced at the start of the main loop. The application example in the appendix uses this method; it forces the reset using an illegal opcode (STOP Instruction used to generate an Illegal Opcode Reset); see the appendix for specific details.

Figure 1 shows a basic flow diagram of the operation. Out of reset, the MCU runs from the internal RC (IRC) oscillator. The code performs some common initialization tasks, which could include setting up ports and peripheral configuration.

The code then checks for a Power On Reset (POR) condition, by reading the SRSR. If the POR flag is set, the code executes the POR initialization, before enabling and switching to the crystal clock source. If a 32768 Hz crystal is used, switching to the external clock takes a relatively long time, as the crystal requires up to 4096 cycles to stabilize.

Alternatively, if an illegal opcode (ILOP) was detected, the code performs some specific initialization to restore the registers to the values prior to the reset. The ProgEE flag is then checked and; if it is set, the EEPROM is then programmed (see application note AN2346: "EEPROM Emulation using FLASH in MC68HC908QY/QT"), the ProgEE Flag is cleared to indicate programming was a success, the external crystal is enabled, and, finally, the main loop is entered.

If another reset condition was detected or the ProgEE flag is clear, the code jumps to the specific service routine, before switching to the external crystal and entering the main loop.

Each iteration of the loop checks the status of the ProgEE flag, to see if programming is to be performed, and then forces a reset, as required. This flag could be set by an external condition (for example, a switch or IRQ).

In this application, the decision to program EEPROM is always taken at the start of the loop. Thus, when the code starts up from the forced reset, it starts executing at the same part of the code, once initialization is complete. There is a latency from the event signalling to program EEPROM to the array being programmed. The maximum latency equals the maximum loop iteration + time to force reset + reset time and recovery + initialization + program time. The example application gives a typical time for this latency.

NOTE: *It could be necessary to save critical variables and/or internal registers before forcing the reset, and to restore this setup information when the MCU comes out of reset. This can be accomplished by storing the variables on the stack. Method 2 demonstrates this.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Advantages of Method 1

- Low RAM requirements (will be higher if variables must be stored on the stack)
- Simple implementation (especially if identical initialization code can be used)

Disadvantages of Method 1

- Code does not resume at the point where the reset occurred
- Relatively long time to switch from internal oscillator to crystal (approximately 125 ms)
- All modules in reset condition

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

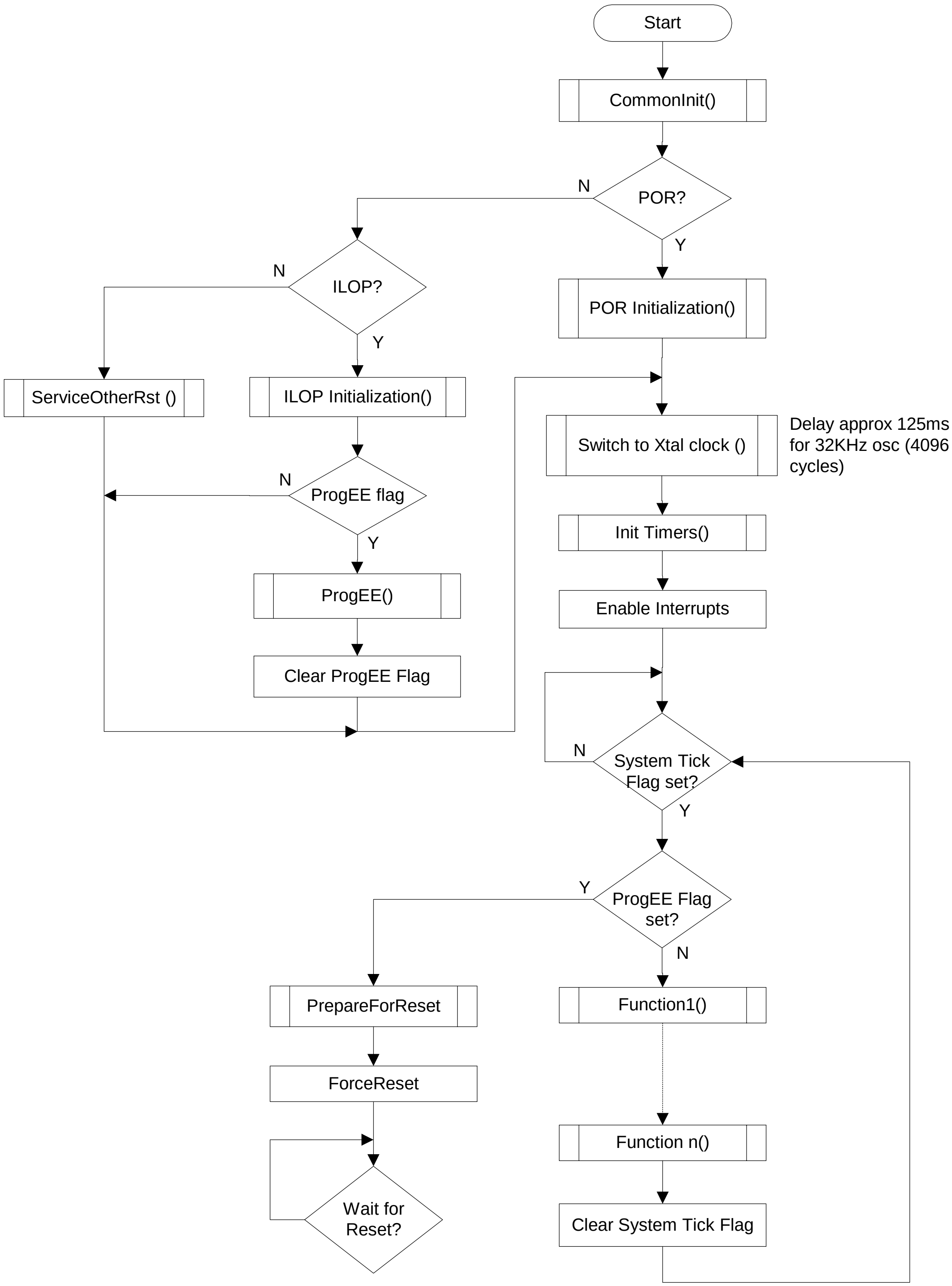

Figure 1. Method 1 — Flow Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Method 2 — Reset Forced Anywhere in Main Loop

This method is more flexible than Method 1, but is also more complicated. It allows the program to call the ProgEEprom command at any point within the application, and then to return to the next instruction after the call, when EEPROM programming is complete.

Before calling the ProgEEprom routine, any critical variables should be pushed onto the stack. The ProgEEprom routine also stores the internal registers on the stack, before jumping to the ForceRst function. The JSR instruction automatically pushes the return address (instruction after JSR ForceRst), so that the program can return to this point in the application after the programming is complete. The ForceRst routine copies the current stack pointer and a ForceRst code to predefined RAM locations, before forcing the reset with an illegal opcode. See **Figure 2** for details.

When the application restarts after a forced reset, the previously stored data registers should immediately be restored to the specific MCU registers. The code retrieves the saved stack pointer from RAM and adjusts it to point to the start of the copied data (see **Figure 3**).

Once the variables are restored and the other initialization performed, the EEPROM emulation routine should be called.

The code then switches back to the external oscillator before returning to the instruction following the forced reset. This is achieved by loading the original Stack Pointer (use TXS instruction) and executing an RTS instruction, which loads the PC with the address stored on the stack. This address is the address of the instruction immediately following the JSR instruction that was executed in the ProgEEpom routine. The process restores the contents of the internal registers before executing a RTS, which returns to the main routine. See **Figure 3** for a general startup procedure.

Advantages of Method 2

- Resumes code execution at the instruction after the forced reset
- Current program status saved and restored

Disadvantages of this Method 2

- Additional RAM required to store program setup
- Relatively long time to switch from internal oscillator to crystal (approximately 125 ms)
- All modules in reset condition.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

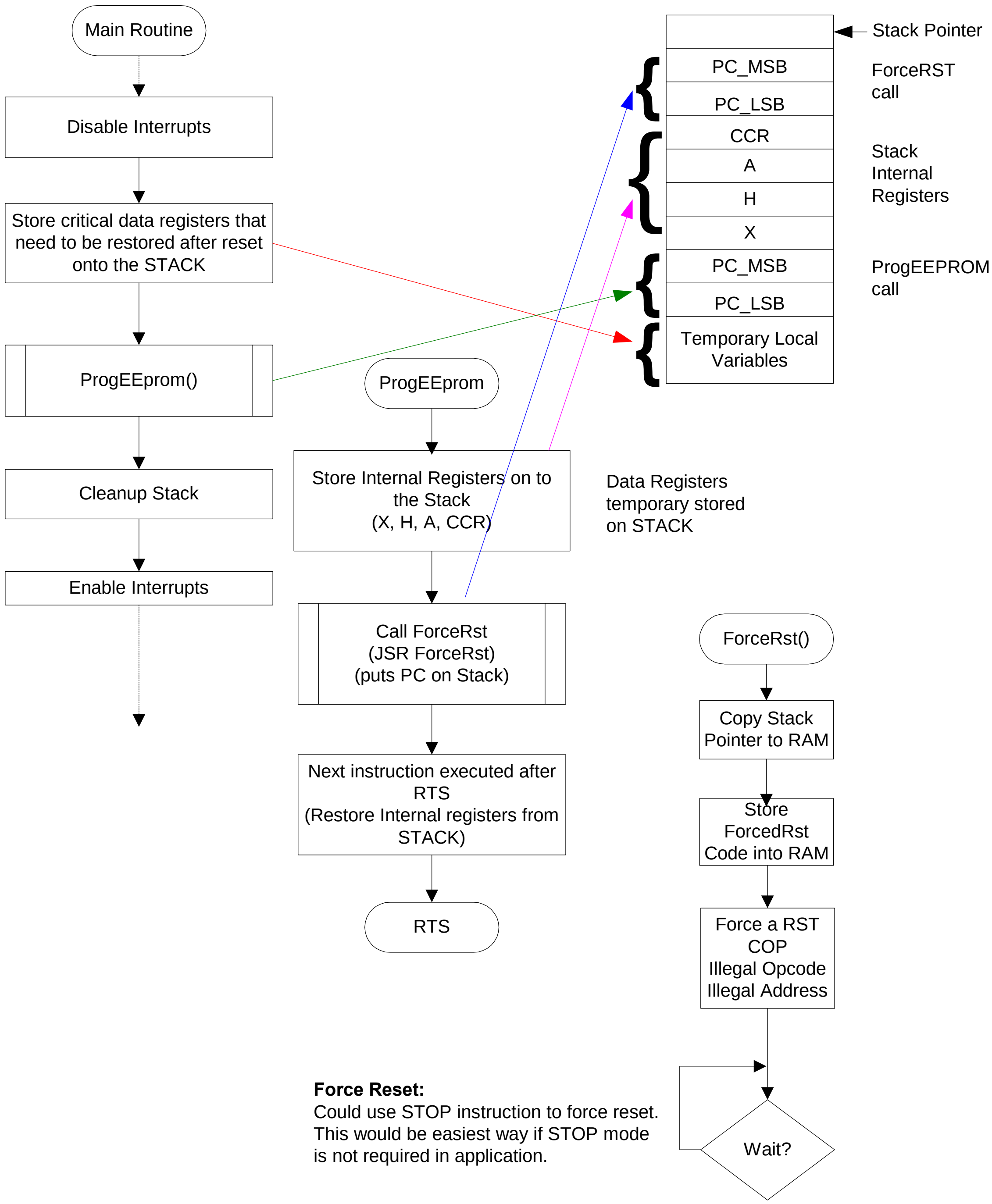

Figure 2. Method 2 — Forcing a Reset

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

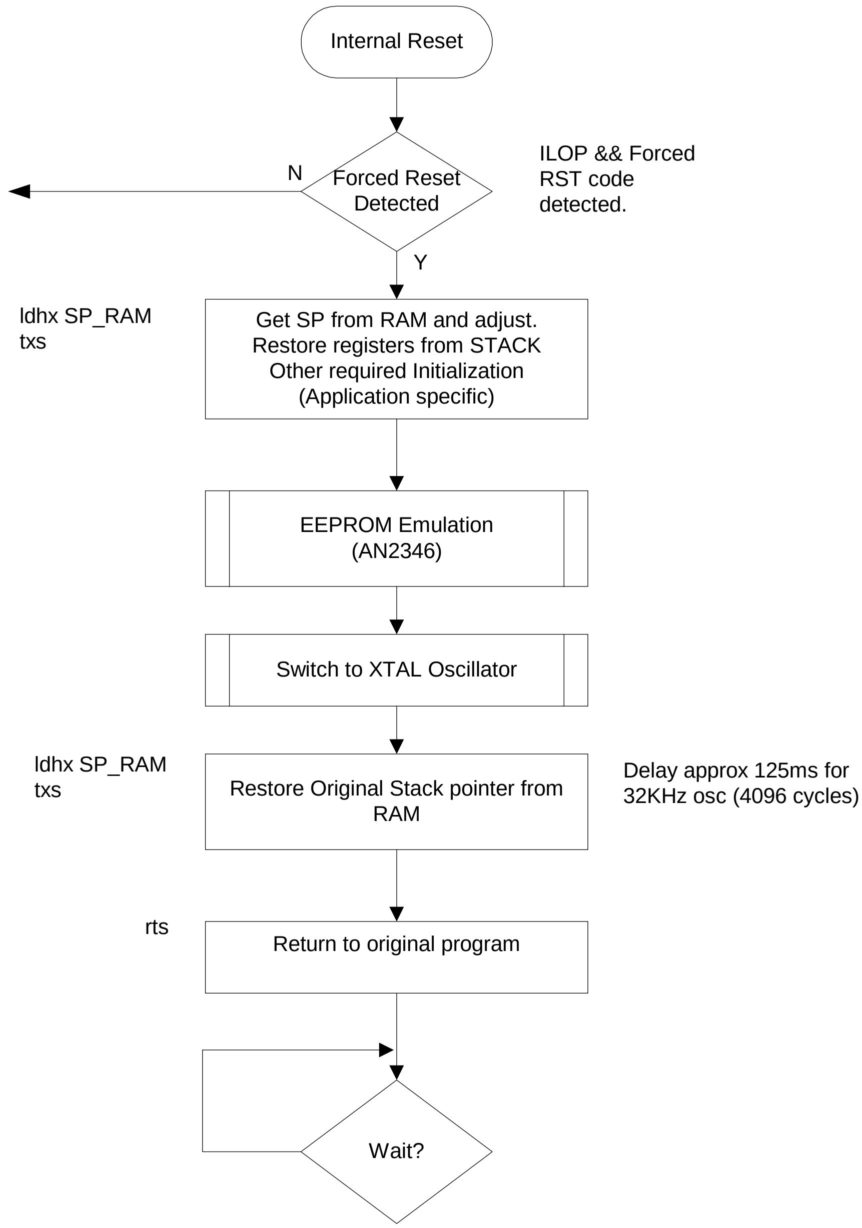

Figure 3. Method 2 — Recovering from Reset

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Conclusions

It is feasible to emulate EEPROM on the MC68HLC908QY family, when the main application is running from a 32768 Hz external crystal, by forcing a reset at a specific part of the code, thereby causing a switch to the internal oscillator and allowing the FLASH to be programmed.

However, in doing so, the MCU is reset and the data registers, internal registers and modules are put into their default conditions. This could cause problems in some applications, for example, where outputs change to inputs until initialized.

Another issue is the time required to switch back to the slow 32768 Hz external crystal clock source, as the source should not be switched before the crystal clock is stable, which can take up to 4096 cycles (125 ms for a 32 kHz crystal). This time delay could be a problem in some real-time systems.

Method 2 is the more flexible solution, as it allows the code to return to the instruction following the forced reset, whereas Method 1 is simpler and easier to implement.

The following appendix shows a simple application that demonstrates Method 1.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Appendix A: Sample Application

The sample application uses a Freescale MC68HLC908QY4 “LIN kits” slave board with an MC68HLC908LQY4 MCU installed.

An additional external 32768 Hz crystal is located in the demo area of the board. The board also has some additional resistors and jumpers, to provide flexibility. The hardware block diagram is shown in **Figure 4**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

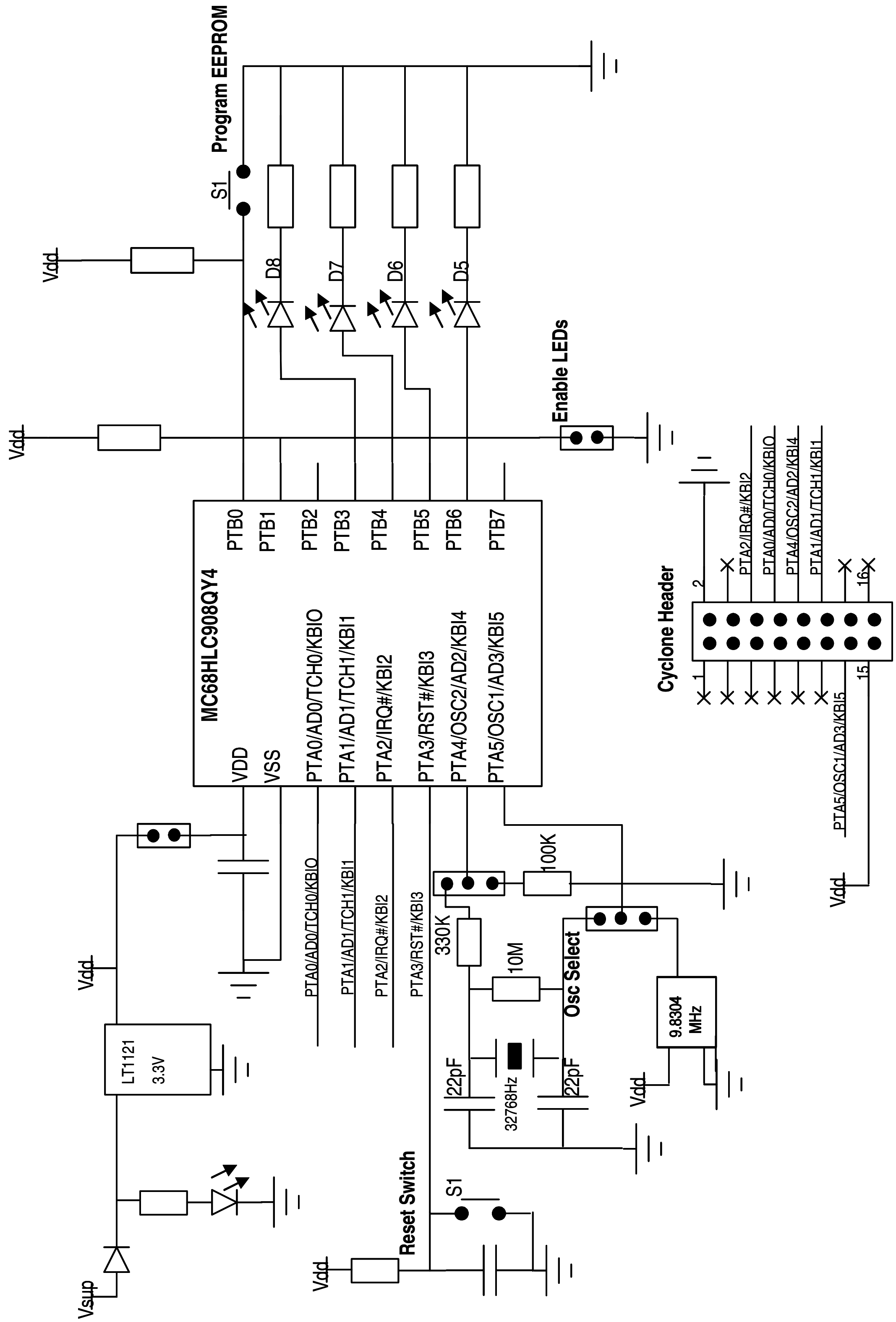

Figure 4. Hardware Block Diagram

For More Information On This Product,
Go to: www.freescale.com

The application starts up from the internal 4 MHz IRC, switches to the external 32768 Hz crystal, and then runs in an infinite loop, controlling four LEDs on the board.

Out of POR, with the emulated EEPROM page blank (FF), the first LED (D8) is lit and the other 3 LED's, D7 to D5, are off. After 1 second, the pattern is shifted to the left, such that D7 is now lit and D8, D6 and D5 are off. This pattern continues until D5 is lit and the others are off. The next iteration reverses the pattern, and this continues indefinitely.

The LEDs can be disabled by placing a jumper between pins PTB1 and GND. This allows the MCU current to be measured without including the additional current required to drive the LEDs. When the jumper is removed, the sequence starts with the same pattern as when the jumper was installed. If a “program EEPROM” request occurs when the jumper is installed, the LED off pattern is stored in EEPROM. The direction is the same as when the jumper was installed.

The code also checks the status of switch S1, which is used to signal a “program EEPROM” request. If the switch is pressed, the current LED pattern and the direction that the pattern is being shifted are stored to emulated EEPROM, with a count byte that is incremented each time new data is stored. Figure 5 shows details of the data that is stored and the FLASH area that is reserved for EEPROM emulation.

EEStart: $EE00	PORT B	First EEPROM Data Store
	ApplicationFlags	
	CountByte	
	PORT B	Second EEPROM Data Store
	ApplicationFlags	
	CountByte+1	
$EE3F		

Figure 5. EEPROM Configuration

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The code checks the PROG_EEPROM_FLAG at the start of the each iteration of the loop. If this bit is set, it forces a reset, starts up from the internal oscillator, programs EEPROM, switches back to the external crystal, and resumes execution from the start of the loop. If the application is powered down and then switched back on, the last saved pattern is restored. This demonstrates that data was actually stored to emulated EEPROM. If the power was removed before the switch was pressed, the code starts by switching on the first LED (D8). The application has been implemented such that any other reset returns the LED sequence to the start (LED D8 on).

The following sections list the code, which has been written in assembly language for the HC08, and provide flow diagrams for each function. The flow diagrams are shown first to help the reader understand the software.

Main Flow Diagram

The main function starts operating from the internal oscillator and performs general initialization of registers, checks the source of reset and, depending on the source, provides additional specific configuration. It then switches the clock source to the external crystal, initializes the timers, enables the global interrupt before entering an indefinite loop. The loop is timed by a 100 Hz timer overflow.

During each iteration of the loop, the status of the program EEPROM switch is checked and, if a valid signal is detected, the PROG_EEPROM_FLAG is set and a reset is forced using the STOP command. In addition, the LEDs are updated every second to demonstrate that the program is running properly.

NOTE: *This flow diagram shows a specific implementation of the general flow diagram shown in **Figure 1**.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

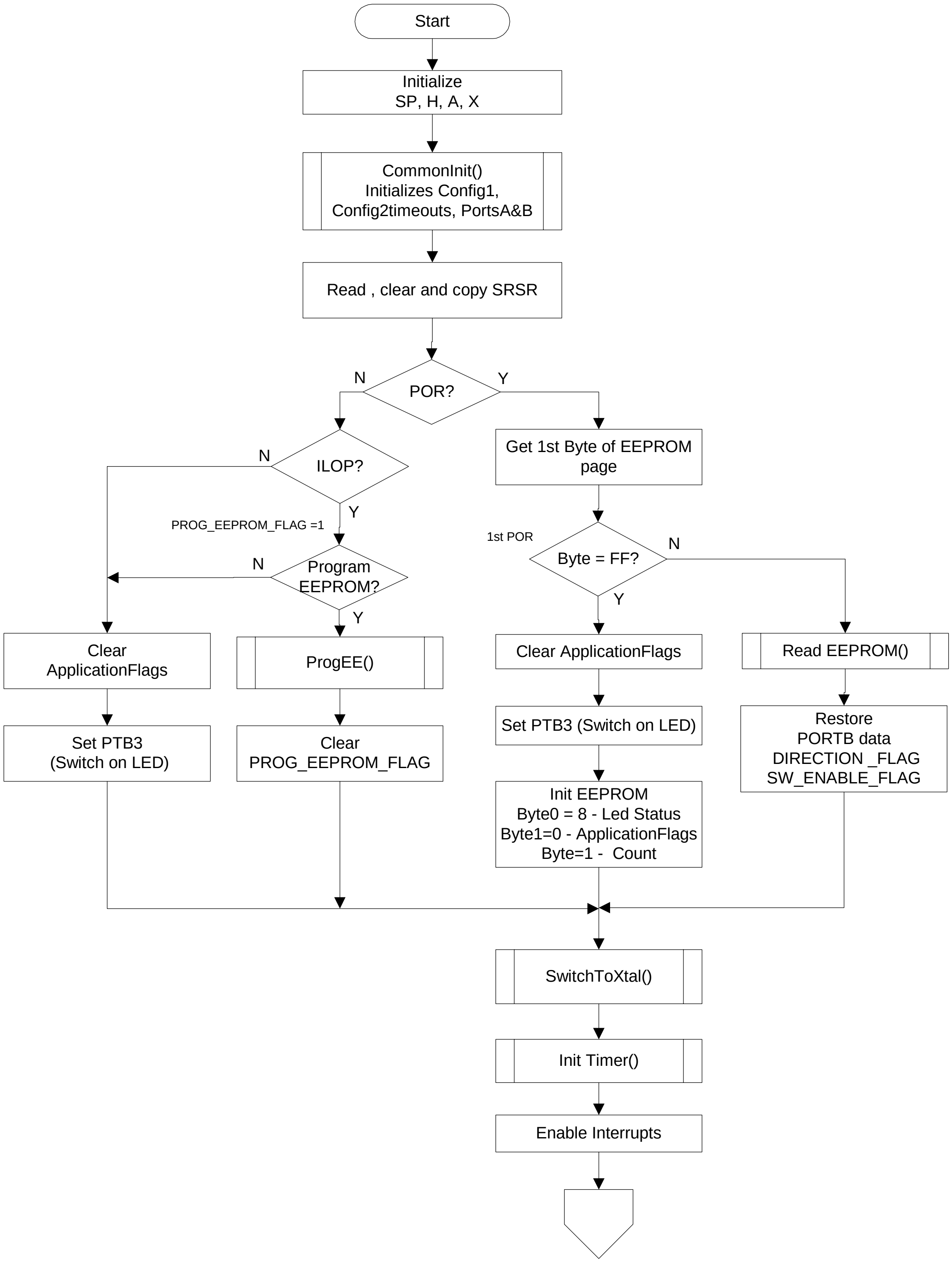

Figure 6. Main Flow (Part 1)

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

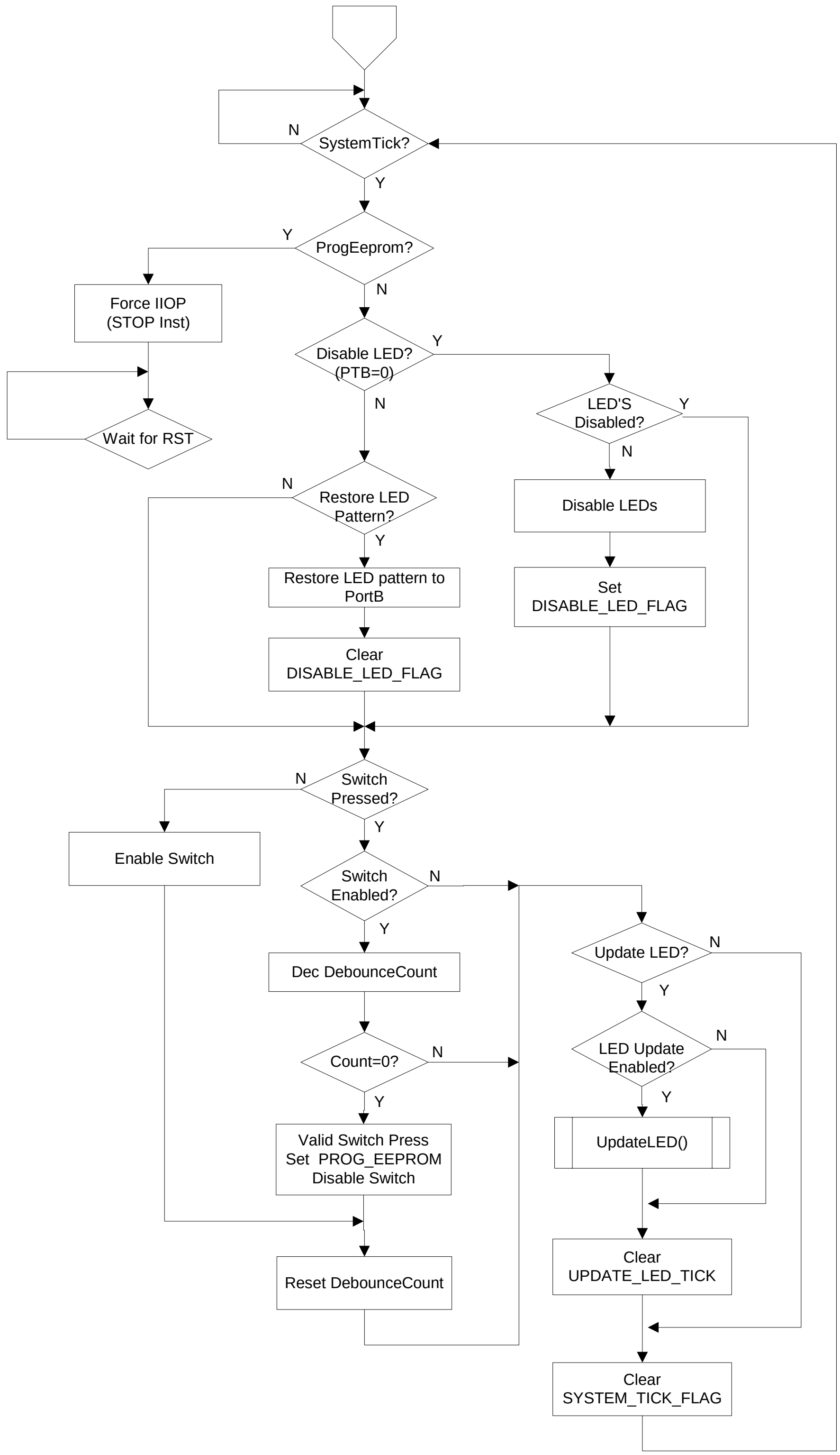

Figure 7. Main Flow (Part 2)

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

CommonInit Flow Diagram

This routine initializes common registers that must be configured irrespective of the reset source. It also initializes the EEPROM driver for operation from a 1 MHz bus.

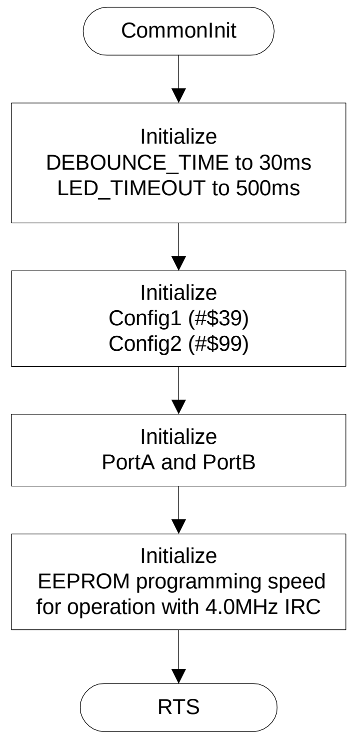

Figure 8. Common Initialization

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

SwitchToXtal Flow Diagram

This function switches the oscillator source from the internal 4 MHz oscillator to the external 32768 Hz oscillator, which is used to drive the application.

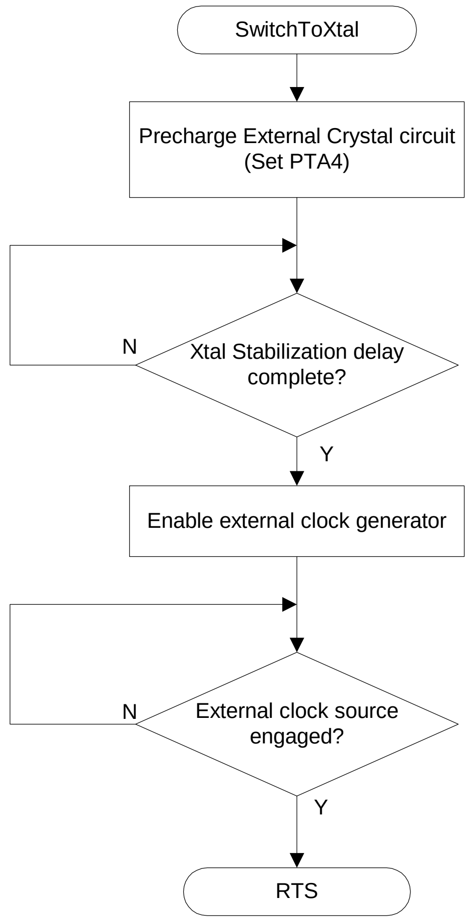

Figure 9. Switch to External Crystal

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

ProgEEPROM Flow Diagram

This routine reads the last data written into EEPROM, increments the count value and reprograms the EEPROM with the current LED status, the current ApplicationFlags and an updated count value.

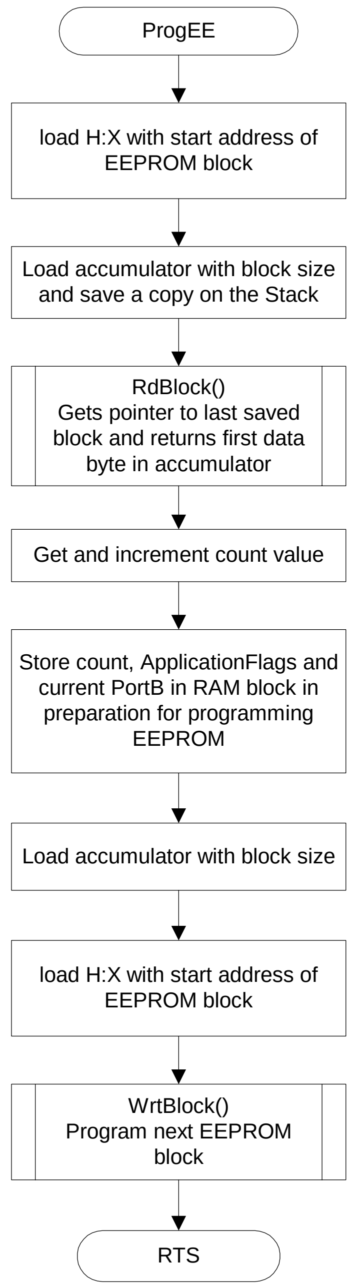

Figure 10. Program EEPROM

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

InitTimer Flow Diagram

This routine sets up a timer overflow (rate = 100 Hz (10 ms)), which is used to pace the main loop. The bus clock is approximately 8 kHz, as the MCU is driven from the external 32768 Hz clock. This gives an overflow of 10 ms with prescaler = 1 and modulo value set to 80d.

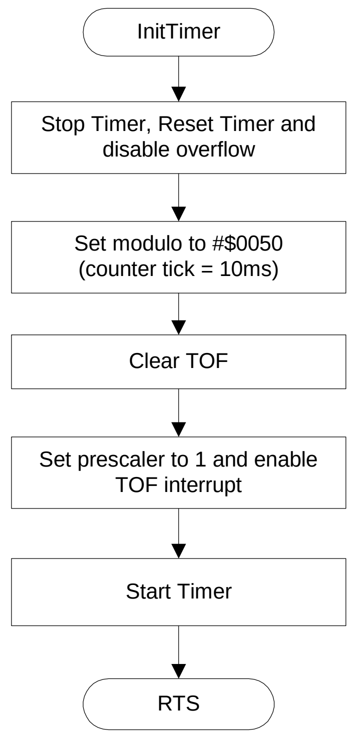

Figure 11. Initialize Timer

Freescale Semiconductor, Inc.

LedDriver Flow Diagram

This routine controls the position of the illuminated LED. At reset, D8 (PTB3) is lit. This is shifted to the left after each timeout (1000 ms). When D5 (PTB6) is lit, the sequence is reversed, and the bit is shifted back to the right. This pattern continues indefinitely.

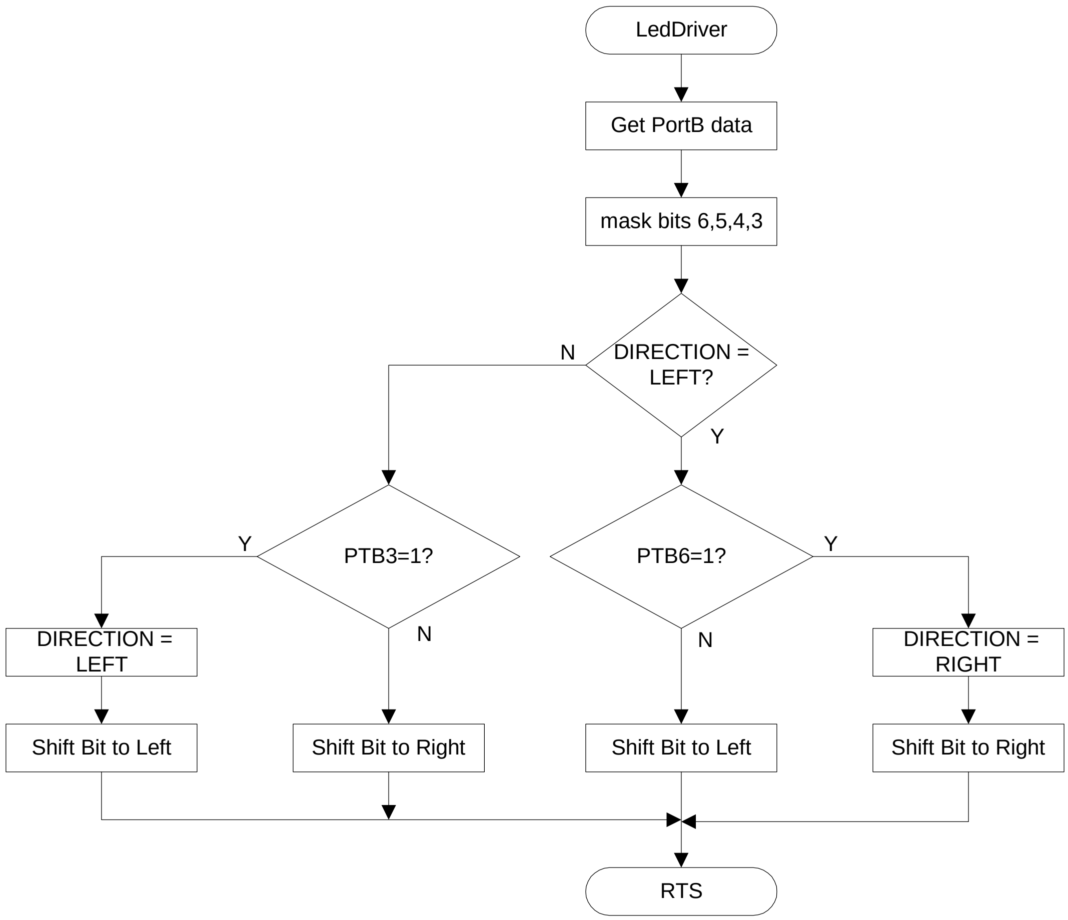

Figure 12. LED Driver

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

TimerISR Flow Diagram

This routine is the interrupt service routine (ISR) for the timer overflow. The SYSTEM_TICK_FLAG is set on every entry of the ISR to indicate a 10 ms timeout. The ISR also sets the UPDATE_LED_TICK every second to time the update rate for the LEDs. Finally, the interrupt is serviced by clearing the TOF flag, before exiting the ISR.

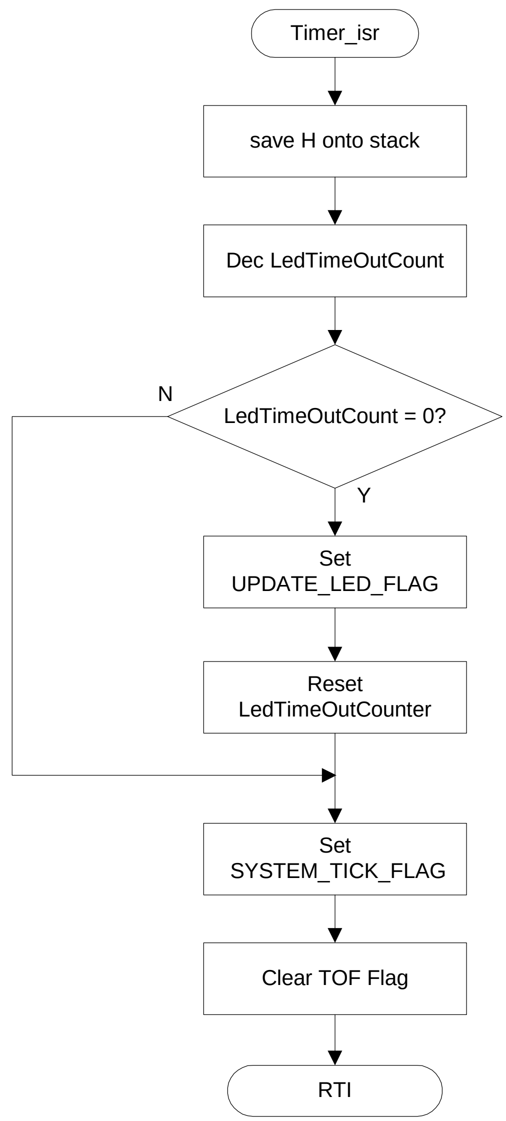

Figure 13. Timer Interrupt Service Routine

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Software Listing

```
;/*****************************************************************************************
;                         (c) Freescale  2004 all rights reserved.
;
;File Name           :   main.asm

;Engineer            :   r29414 (Alan Devine 8/16bit systems)

;Location            :   EKB

;Date Created        :   05/03/2004

;Description         :   Example of Flash programing when application on 908QY4 is
;                        running from 32768Hz watch crystal.  The application
;                        forces an illigal opcode reset using the STOP command,
;                        programs the FLASH when bus is being driven by an internal
;                        1MHz Osc and then switch back to external 32768Hz clock.

; Rev       Issue Date      Author      Change Description
; ---       ----------      ------      ------------------
; 0.0       08/01/2004      A.D.        Initial release to P.T
; 1.0       05/03/2004      A.D.        Version included in Application note.
;                                       Add function to disable Leds
;                                       Simplify Freescale disclaimer

;************************************** export symbols ************************************
            XDEF Entry, main, Timer_isr

;**************************** include derivative specific macros **************************
            Include 'qy4_registers.inc'

;**************************************** Equates  ******************************************
;Bit defs for ApplicationFlags
SYSTEM_TICK_FLAG    EQU 0               ; Loop timeout flag. Set in Timer_isr
UPDATE_LED_FLAG     EQU 1               ; Indicates that LED's sequence to be updated
PROG_EEPROM_FLAG    EQU 2               ; Program Flash when set (set when S1 pushed)
DIRECTION_FLAG      EQU 3               ; Direction that Leds move. Left=0, Right=1
SW_ENABLE_FLAG      EQU 4               ; Switch enable. Clear = Enabled; Set = Disabled
                                        ; Switch is enabled after a transition 0 -->1 or
                                        ; POR occurs
DISABLE_LED_FLAG    EQU 5               ; Indictes if LED's are to be disabled.  0 - Leds
                                        ; enabled.  1 - Leds disabled

;Bit masks for ApplicationFlags
mSYSTEM_TICK_FLAG   EQU %00000001
mUPDATE_LED_FLAG    EQU %00000010
mPROG_EEPROM_FLAG   EQU %00000100
mDIRECTION_FLAG     EQU %00001000
mSW_ENABLE_FLAG     EQU %00010000
mDISABLE_LED_FLAG   EQU %00100000

; Equates for ROM Subroutines and start of RAM
EraRnge             EQU $2806           ;FLASH erase routine in ROM
PgrRnge             EQU $2809           ;FLASH programming routine in ROM
CtrlByt             EQU $88             ;control byte for ROM subroutines
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
CPUSpd              EQU $89             ;CPU speed in units of 0.25MHz
LstAddr             EQU $8A             ;last FLASH address to be programmed

; Equates for Flash address used for EEPROM
EeStart             EQU $EE00           ;Start of EEPROM page in FLASH

;Bit defs for SRSR
LVI                 EQU 1
MODRST              EQU 2
ILAD                EQU 3
ILOP                EQU 4
COP                 EQU 5
PIN                 EQU 6
POR                 EQU 7

;Bit masks for SRSR register
mLVI                EQU %00000010
mMODRST             EQU %00000100
mILAD               EQU %00001000
mILOP               EQU %00010000
mCOP                EQU %00100000
mPIN                EQU %01000000
mPOR                EQU %10000000

;Bit defs for PortA
PTA0                EQU 0
PTA1                EQU 1
PTA2                EQU 2
PTA3                EQU 3
PTA4                EQU 4
PTA5                EQU 5
PTA6                EQU 6
PTA7                EQU 7

;Bit defs for DDRA
DDRA0               EQU 0
DDRA1               EQU 1
DDRA2               EQU 2
DDRA3               EQU 3
DDRA4               EQU 4
DDRA5               EQU 5
DDRA6               EQU 6
DDRA7               EQU 7

;Bit defs for PortB
PTB0                EQU 0
PTB1                EQU 1
PTB2                EQU 2
PTB3                EQU 3
PTB4                EQU 4
PTB5                EQU 5
PTB6                EQU 6
PTB7                EQU 7

;Bit defs for DDRB
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
DDRB0               EQU 0
DDRB1               EQU 1
DDRB2               EQU 2
DDRB3               EQU 3
DDRB4               EQU 4
DDRB5               EQU 5
DDRB6               EQU 6
DDRB7               EQU 7

;Bit defs for OSCSTAT register
ECGST               EQU 0
ECGON               EQU 1

;Constants
LED_TIMEOUT_PERIOD  EQU 100             ; Led update rate = 100 x System tick(10ms) = 1000ms
DEBOUNCE_TIME       EQU 3               ; Debounce time = 3 x System tick = 30ms

;********************************** variable/data section **********************************
MY_ZEROPAGE0: SECTION  SHORT            ; Section bytes $80-$87
ApplicationFlags    ds.b 1              ; Flags used in application
CopySRSR            ds.b 1              ; Temp copy of SRSR register
DebounceCounter     ds.b 1              ; Used to time debounce period
LedTimeOutCount     ds.b 1              ; Used to time LED update rate
CopyLedPattern      ds.b 1              ; Temp copy of Led pattern.  Could CopySRSR be used


ROM_ROUTINES_RAM:   SECTION SHORT       ; Reserved RAM for ROM Routines ($88-$8F)
Reserved0           ds.b 1              ; CtrlByt         $88
AppCPUSpd           ds.b 1              ; CPUSpd          $89
Reserved1           ds.b 2              ; LstAddr         $8A-$8B
RamBfrStrt          ds.b 3              ; data buffer size - BfrStrt     $8C-$8E

MY_ZEROPAGE1: SECTION SHORT             ; Section bytes $90-$FF

;************************************** code section **************************************
MyCode:       SECTION                   ; Code Starts at $EE40
main:
Entry:
    rsp                                 ; Reset SP to $FF. Stacksize $30
    clrh
    clra
    clrx

    jsr     CommonInit                  ;Initialise common variables:Config1,Config2
                                        ;Debounce and LED timeout, PortA and PortB
    lda     SRSR                        ;Read and clear reset status register
    sta     CopySRSR                    ;Copy to temp variable, as read clears flags
    ;Check for POR
    brset   POR, CopySRSR, PORset

    ;Check for ILOP
    brset   ILOP, CopySRSR, ILOPset
    ;****Include other reset checks here ****

OtherRst:
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    ;catches all other reset sources
    mov     #$08, PORTB                 ;Initilise LED sequence
    clr     ApplicationFlags
    bra     SwXtal


PORset:
    ;Check for first POR
    lda     EeStart                     ;Get 1st byte in EEpage
    cmp     #$FF                        ;Check that its blank
    bne     EEnotBlank                  ;Not Blank

    clr     ApplicationFlags            ;Switch enabled

    ;Init PortB
    mov     #$08,PORTB                  ;Initilise LED sequence

    ;Init EEPROM
    mov     #$08, RamBfrStrt            ;Initilise LED sequence
    clr     RamBfrStrt+1                ;ApplicationFlags Clear - DIR Left
    mov     #$01, RamBfrStrt+2          ;Count = 1

    ldhx    #EeStart
    lda     #$3                         ;3 Bytes to be programmed
    jsr     WrtBlock                    ;Go program EEPROM

    bra     SwXtal

EEnotBlank:
    ldhx    #EeStart                    ;Get current count value
    lda     #3
    psha                                ;save buffer size on stack
   jsr     RdBlock                  ;gets pointer to latest data block: Returns start address
                                     ;of most recent data in H:X and 1st Byte in accumulator

    sta     PORTB                       ;Restore PortB
    lda     1,x                         ;get stored ApplicationFlags
    and     #mDIRECTION_FLAG            ;Only interested in Direction
    sta     ApplicationFlags            ;Restore saved Direction bit; Switch Enabled

    bra     SwXtal
ILOPset:
                                        ;Check if EEprom to be programmed
    brclr   PROG_EEPROM_FLAG, ApplicationFlags, OtherRst

    bsr     ProgEE                      ;PROG_EPROM_FLAGS = 1
                                        ;Reset flag for next program request
    bclr    PROG_EEPROM_FLAG, ApplicationFlags
SwXtal:
    bsr     SwitchToXtal                ;Configures QY4 for operation from 32768Hz Xtal

    jsr     InitTimer                   ;Init Timer1 overflow

    cli                                 ;enable interrupts
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
MainLoop:                               ;Wait for interrupt
    brclr   SYSTEM_TICK_FLAG,ApplicationFlags, MainLoop

    brset   PROG_EEPROM_FLAG, ApplicationFlags, ProgEeprom
                                        ;Are Leds to be disabled. Disable if PTB1 = 0
    brclr   PTB1, PORTB, DisableLEDs
                                      ;Are Leds to be restored. Restore if DISABLE_LED_FLAG =1
    brclr   DISABLE_LED_FLAG,ApplicationFlags, ChkSw1
                                        ;Restore Led pattern
    mov     CopyLedPattern, PORTB       ;Clear disable LED flags
    bclr    DISABLE_LED_FLAG, ApplicationFlags

    bra     ChkSw1

DisableLEDs:                       ;Are Leds already disabled. Disabled if DISABLE_LED_FLAG =1
    brset   DISABLE_LED_FLAG, ApplicationFlags, ChkSw1

    lda     PORTB
    sta     CopyLedPattern              ;Store Current LED pattern
    and     #%10000111                  ;Clear bits 6,5,4,3
    sta     PORTB
                                        ;Indicate Leds are diabled
    bset    DISABLE_LED_FLAG, ApplicationFlags

ChkSw1:
    brclr   PTB0, PORTB, ChkSwEn        ;Switch pressed - PTB0 = 0
    bclr    SW_ENABLE_FLAG, ApplicationFlags;Enable switch
    bra     RstDebounce
ChkSwEn:                                ;Look to see if switch is enabled
    brset   SW_ENABLE_FLAG, ApplicationFlags,ChkLedUpdate
    dec     DebounceCounter
    bne     ChkLedUpdate                ;Check for Timeout?
    bset    PROG_EEPROM_FLAG, ApplicationFlags ;Timeout
    bset    SW_ENABLE_FLAG, ApplicationFlags;Disable switch


RstDebounce:                            ;Reset debounce counter for next itereation
    mov     #DEBOUNCE_TIME, DebounceCounter

ChkLedUpdate:
    brclr   UPDATE_LED_FLAG, ApplicationFlags, EndMainLoop
                                        ;Are leds disabled?
    brset   DISABLE_LED_FLAG, ApplicationFlags, SkipLedDriver

    bsr     LedDriver

SkipLedDriver:                          ;Reset LED update flag for next iteration
    bclr    UPDATE_LED_FLAG, ApplicationFlags

EndMainLoop:                            ;Reset for next iteration
    bclr    SYSTEM_TICK_FLAG,ApplicationFlags

    bra     MainLoop

ProgEeprom:
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    ;prepare for reset
    STOP                                ;Force Ilegal Opcode reset

WaitForReset:
    bra    WaitForReset

;*******************************************************************************************
;* Name:                  CommonInit
;* Description:           Initialises the registers that are not Reset specific.  The registers
;*                        initialised are CONFIG1,CONFIG2, PORTA,DDRA, PORTB, and DDRB. The
;*                        EEPROM driver speed is also configured in this routine
;*
;*
;* Calling Convention:  bsr CommonInit
;* Inputs:                none
;* Outputs:               none
;* Routines used:         none
;* Stack usage:           none
;*******************************************************************************************
CommonInit:

    mov     #DEBOUNCE_TIME, DebounceCounter      ;Init debounce counter for next itereation
    mov     #LED_TIMEOUT_PERIOD, LedTimeOutCount ;Init Led TimeOut

   ;**** Config Registers *****
    mov     #$39,CONFIG1     ;COPRS = 0      - COP Reset Period = (2^18-2^4)xBUSCLKK4 cycles
                             ;LVISTOP = 0    - LVI Disabled during STOP Mode
                             ;LVIRSTD = 1    - LVI Module resets disabled
                             ;LVIPWRD = 1    - LVI Module power disabled
                             ;LVDLVR = 1     - LVI trip voltage level set to LVR trip voltage
                             ;SSREC = 0      - Stop mode recovery after 4096 BUSCLKX4 cycles
                             ;STOP = 0       - STOP Instruction treated as illegal opcode
                             ;COPD = 1       - COP Disabled

    mov     #$99,CONFIG2     ;IRQPUD = 1     - IRQ Internal pullup disconnected
                             ;IRQEN = 0      - IRQ Pin function disabled
                             ;R = 0
                             ;OSCOPT1:0= 11 - Xtal Crystal
                             ;R = 0
                             ;R = 0
                             ;RSTEN = 1      - RST Pin function Active

    mov     #$00,PORTA       ;PortA inputs
    mov     #$10,DDRA        ;PTA4 set as output.

    mov     #$78,DDRB        ;PORTB7 = 0 - Input
                             ;PORTB6 = 1 - Output (D5)
                             ;PORTB5 = 1 - Output (D6)
                             ;PORTB4 = 1 - Output (D7)
                             ;PORTB3 = 1 - Output (D8)
                             ;PORTB2 = 0 - Input
                             ;PORTB1 = 0 - Input
                             ;PORTB0 = 0 - Input (S1)
    mov     #$4,AppCPUSpd    ;Init EEPROM programming driver for operation
                         with 1MHz bus. (4x0.25MHz)
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    rts                        ; return

;*************************************************************************************
;* Name:                  SwitchToXtal
;* Description:           Switches the osc source from the internal Oscillator to the external
;*                        32768Hz oscillator.
;*
;* Calling Convention:  bsr SwitchToXtal
;* Inputs:                none
;* Outputs:               none
;* Routines used:         none
;* Stack usage:           none
;*************************************************************************************
SwitchToXtal:
    bset    PTA4, PORTA                 ; Precharge external crystal circuit
    nop
    nop

    lda     #$A2                        ;Wait 4096 cycles of 32KHz crystal.
    clrx                                ;= 125ms = 125000cycles of 1meg bus

stxL1:
    dbnzx   stxL1                       ;Inner loop = 256 x 3 cycles = 768 cycles
   dbnza   stxL1                  ;Outer loop = 3 x 162 (A2h) + 162 x 768 cycles = 124902 cycles

    bset    ECGON, OSCSTAT              ; External clock generator enabled

stxL2:
    brclr   ECGST, OSCSTAT, stxL2       ; Wait for external clock source to be engaged

    bclr    PTA1, PORTA                 ; clear external osc engaged flag
    bset    DDRA1, DDRA                 ; PortA, bit1 is an output

    rts                                 ;return

;*************************************************************************************
;* Name:                  ProgEE
;* Description:           This routine reads the last data written into EEPROM, increments the
;*                        count value and reprograms the EEPROM with the current led status
;*                        the ApplicationFlags and the updated count value.
;*
;* Calling Convention:  bsr ProgEE
;* Inputs:                none
;* Outputs:               none
;* Routines used:         RdBlock, WrtBlock
;* Stack usage:           1 byte
;*************************************************************************************
ProgEE:
    ldhx    #EeStart                    ;Get start address of EEprom Block
    lda     #3                          ;number of bytes in EEPROM
    psha                                ;save buffer size on stack
    jsr     RdBlock                     ;gets pointer to latest data block
    lda     2,x                         ;get count value
    inca                                ;inc count
    sta     RamBfrStrt+2                ;store in buffer
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    mov     ApplicationFlags, RamBfrStrt+1 ;Store current application flags
    lda     PORTB                          ;get PortB
    and     #%01111000                     ;only interested in Ptb6 - ptb3
    sta     RamBfrStrt                     ;Copy port status variable into ram location
    pula                                   ;get buffer size back
    ldhx    #EeStart
    jsr     WrtBlock

    rts                                    ; return

;*********************************************************************************************
;* Name:                    InitTimer
;* Description:             Sets up a timer overflow rate = 100Hz (10ms).For bus clock = 8KHz
;*                          Pre-Scale = 1, Modulo = 80 (50H)
;*
;* Calling Convention:  bsr InitTimer
;* Inputs:                  none
;* Outputs:                 none
;* Routines used:           none
;* Stack usage:             none
;*********************************************************************************************
InitTimer:
    mov     #$30,TSC                       ; Stop timer, Reset Timer,
                                           ; Disable timer overflow interrupt
    mov     #$00,TMODH                     ; set modulo to 80 (50H)
    mov     #$50,TMODL
    lda     TSC                            ; Clear TOF flag - Read then write 0 to TOF
    bclr    7,TSC
    mov     #$60,TSC                       ; Enable TOF Interrupt, Timer stopped, PS = 1 (000)
    bclr    5,TSC                          ; Start timer
    rts                                    ;return

;*********************************************************************************************
;* Name:                    LedDriver
;* Description:             This routine controls the position of the illuminated LED.  At
;*                          reset the D8 (PTB3) is iluminated.  This is shifted to the left
;*                          after each timeout (1000ms). When D5 (PTB6) is lit the sequence
;*                          is reversed and the bit is shifted back to the right.
;*
;* Calling Convention:  bsr LedDriver
;* Inputs:                  none
;* Outputs:                 none
;* Routines used:           none
;* Stack usage:             none
;*********************************************************************************************
LedDriver:
    lda     PORTB
    and     #%01111000                     ;Only Interested in bits 6,5,4,3
                                           ; DIRECTION = 0 (LEFT)
    brset   DIRECTION_FLAG, ApplicationFlags, Right
Left:
    brset   PTB6, PORTB, Ptb6Set
    lsla
    sta     PORTB
    bra     ledend
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
Ptb6Set:                                ; DIRECTION = 1 (RIGHT)
    bset    DIRECTION_FLAG, ApplicationFlags
    lsra
    sta     PORTB
    bra     ledend

Right:
    brset   PTB3, PORTB, Ptb3Set
    lsra
    sta     PORTB
    bra     ledend

Ptb3Set:                                ; DIRECTION = 0 (LEFT)
    bclr    DIRECTION_FLAG, ApplicationFlags
    lsla
    sta     PORTB

ledend:
    rts

;*******************************************************************************************
;*
;*  RdBlock - Reads a block of data from FLASH and puts it in RAM
;*
;*  Calling convention:    ldhx   #Blk1page
;*                         lda    #Blk1Size
;*                         jsr    RdBlock
;*
;*  Inputs:  H:X - pointing to start of FLASH page used for data
;*           A   - block size
;*
;*  Returns: H:X - pointing to start of FLASH block containing data
;*           A   - data from first byte of block
;*
;*  Uses:    FindClear
;*
;*******************************************************************************************

RdBlock:
    psha                                ;save block size
    bsr     FindClear                   ;find first erased block

    cmp     #$FF                        ;was an erased block found ?
    bne     skipdec                     ;if not then don't go back a block
    txa                                 ;get LS byte of address
    and     #$3F                        ;only look at address within page
    beq     skipdec                     ;if 0 then no data so don't go back
    txa                                 ;if not get LS byte of address again
    sub     1,sp                        ;and subtract block size to point
    tax                                 ;to start of valid data block

skipdec:
    lda     ,x                          ;get first byte of data
    ais     #1                          ;de-allocate stack
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    rts

;**************************************************************************************
;*
;*  WrtBlock - Writes a block of data into FLASH from RAM buffer
;*
;*  Calling convention:    ldhx    #Blk1page
;*                         lda     #Blk1Size
;*                         jsr     WrtBlock
;*
;*  Inputs:  H:X - pointing to start of FLASH page used for data
;*           A   - block size
;*
;*  Returns: nothing
;*
;*  Uses:    FindClear, EraRnge (ROM), PgrRnge (ROM)
;*
;**************************************************************************************

WrtBlock:
    mov     #13,CPUSpd                  ;3.2MHz/0.25MHz = 13
    clr     CtrlByt                     ;page (not mass) erase
    psha                                ;save block size
    bsr     FindClear                   ;find first available erased block
    cmp     #$FF                        ;erased block found ?
    beq     blkfnd                      ;if so write to it
    jsr     EraRnge                     ;if not then erase page
    txa                                 ;get LS byte of FLASH address
    and     #$C0                        ;and reset it to start of page
    tax                                 ;H:X now pointing to first block

blkfnd:
    pula                                ;get block size
    pshx                                ;save start address LS byte
    add     1,sp                        ;add block size to LS byte
    deca                                ;back to last address in block
    tax                                 ;last address now in H:X
    sthx    LstAddr                     ;save in RAM for use by ROM routine
    pulx                                ;restore X (H hasn't changed)
    jmp     PgrRnge                     ;program block (includes RTS)

;**************************************************************************************
;*
;*  FindClear - Finds first erased block within page
;*
;*  Inputs:  H:X - pointing to start of page used for required data
;*           Stack - block size last thing on stack
;*
;*  Returns if erased block found:
;*           H:X - pointing to start of first erased block in page
;*           A   - $FF
;*  Returns if no erased block found (page full):
;*           H:X - pointing to start of last written block
;*           A   - $00
;*
```


Freescale Semiconductor, Inc.

```
;*******************************************************************************************

FindClear:
    lda     #$40                        ;number of bytes in a page
    sub     3,sp                        ;less number in first block
    psha                                ;save bytes left

floop:
    lda     ,x                          ;get first data byte in block
    cmp     #$FF                        ;erased byte ?
    beq     finish1                     ;if so then exit, otherwise try next

    pula                                ;bytes left
    sub     3,sp                        ;less number in next block
    psha                                ;resave bytes left
    bmi     finish2                     ;enough for another block ?

    txa                                 ;yes, get LS byte of address
    add     4,sp                        ;add block size
    tax                                 ;put it back (can't be a carry)
    bra     floop                       ;and try again

finish2:
    clra                                ;no room (A shouldn't be $FF)
finish1:
    ais     #1                          ;fix stack pointer
    rts

;*******************************************************************************************
;* Name:                  Timer_isr
;* Description:           ISR for overflow timer.  Systemtick flag set to indicate that timeout
;*                        has occured.  The TOF flag is also cleared before exiting ISR
;*
;* Calling Convention:  bsr Timer_isr
;* Inputs:                none
;* Outputs:               none
;* Routines used:         none
;* Stack usage:           none
;*******************************************************************************************
Timer_isr:
    pshh                                ; save H reg.
    *
    dec     LedTimeOutCount             ; dec count
    bne     SetSystemTick               ; Look for LedTimeOut = 0
    bset    UPDATE_LED_FLAG, ApplicationFlags    ; Set LED flag
    mov     #LED_TIMEOUT_PERIOD, LedTimeOutCount ; Reset Led TimeOut

SetSystemTick:
    bset  SYSTEM_TICK_FLAG,ApplicationFlags    ;Interrupt occured
    lda  TSC
    bclr 7,TSC                          ; clear TOF
    *
    pulh                                ; get H back
    rti
;*******************************************************************************************
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.
Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale Semiconductor, Inc.