AN2509/D
11/2003

I²C Slave on the HC908QT/QY Family MCU

By Stanislav Arendarik
Freescale Czech System Application Laboratory
Roznov pod Radhostem, Czech Republic

Introduction

The I²C (inter-integrated circuit) protocol is a 2-wire serial communication interface implemented in numerous microcontrollers and peripheral devices. Many microcontroller units (MCUs) do not have an I²C module, yet they must communicate with 2-wire or I²C devices. These MCUs are usually "master on I²C."

This application note describes a method of communicating on an I²C bus by digital input/output (I/O) pins when the MCUs are "slave on I²C." This method can be implemented on any Freescale MCU through the standard digital I/O pins. This document uses the HC908QT/QY Family as an example.

I²C Overview

I²C is a 2-wire communications link requiring a clock line (SCL) and a data line (SDA) to communicate. The frequency of the I²C clock can go up to 100 kHz for standard mode, and up to 400 kHz for fast mode.

An I²C bus has both a master device and a slave device attached to it. A master is defined as a device which initiates a transfer, generates a clock signal (SCL), and terminates the transfer. A slave device is addressed by the master device. I²C provides a solution for multiple masters on the same bus. This bus also provides some error checking by using acknowledgment bits during byte transfer.

I²C Slave Application

The application presented in this document illustrates a basic example of the I²C specification. It is not intended to implement all the features of an I²C bus, but to provide the basic functionality required to receive data as a slave device from a master device through a 2-wire interface.

This application provides the following functionality:

- 7-bit addressing mode of I^{2}C slave device
- Single master transmitter
- Serial clock frequency reaching up to 75 kHz (SCL) with MCU bus frequency 3.2 MHz
- 16-bit address mode for destination register
- 8-bit data (transmit and receive) for destination register
- “Byte write” function (write one data byte to the one MCU register determined by the two-byte address)
- “Random read” function (read one data byte from the MCU register determined by the two-byte address)

NOTE: *The “byte write” and “random read” terms are explained in the following text.*

By controlling two digital pins, it is possible to simulate an I^{2}C transfer. These I/O pins should be open drain. When the I/O pins are high-density complementary metal oxide semiconductor (CMOS) and not open drain, some safeguards must be implemented. A series resistor should be connected between the CMOS output pin and receiver’s input pin. This will provide some current limiting, should the two devices attempt to output conflicting logic levels.

Another consideration is supporting a logic high level for any open-drain receiver pins. A pullup resistor can be used at the receiver’s open drain pin to passively pull up to the supply voltage when the pin is not being actively driven low. This pullup resistor should be carefully chosen so that when the master pin drives low, a valid V_{IL} level is presented to the I^{2}C receiver pin.

Figure 1 illustrates how to connect digital I/O pins between I^{2}C master and slave devices.

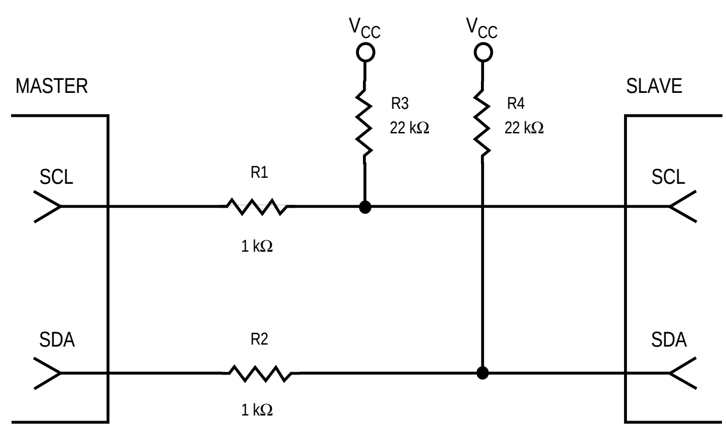

Figure 1. The I^{2}C Bus Connect

When the SCL and SDA I/O ports of the master and slave devices are open drain, the resistors R1 and R2 can be omitted.

An I^{2}C transfer is composed of specific stages defined by the states of the SCL and SDA wires. The inactive state of the I^{2}C bus is when both SCL and SDA lines are in the high logic level. **Figure 2** shows the timing between the clock (SCL) and data (SDA) lines under the START and STOP conditions; **Figure 3** shows the timing between SCL and SDA lines during the data transfer, and **Figure 4** shows the timing of the acknowledge impulse, which is sent by the slave device after it receives all eight bits of the transferred byte.

Basic States

Characteristics of the basic states:

- START condition is indicated by the falling edge on SDA line while the SCL is held in the high logic level
- STOP condition is indicated by the rising edge on SDA line while the SCL is held in the high logic level
- Data on the SDA line can change only if the SCL line is in the low logic level
- Data on the SDA line is valid and is transferred through the I^{2}C bus between devices when the SCL line is in the high logic level
- Acknowledge bit (ACK) is indicated by the low logic level on the SDA line, while on the SCL line is the ninth pulse from the byte transfer. The ACK bit is usually generated by the slave device. The master produces the ACK bit only if the "multiple read function" occurred.

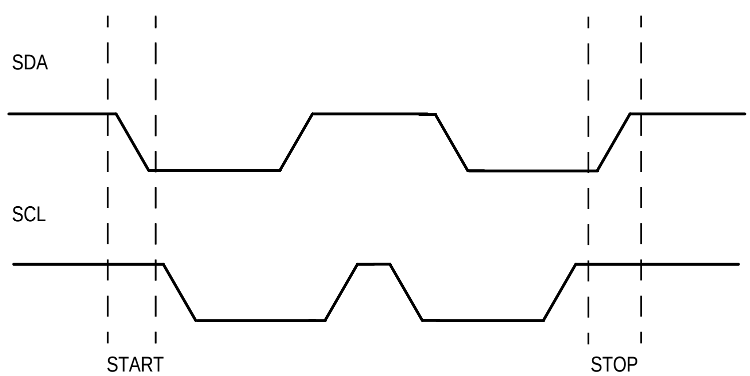

Figure 2. START and STOP Conditions on I^{2}C Bus

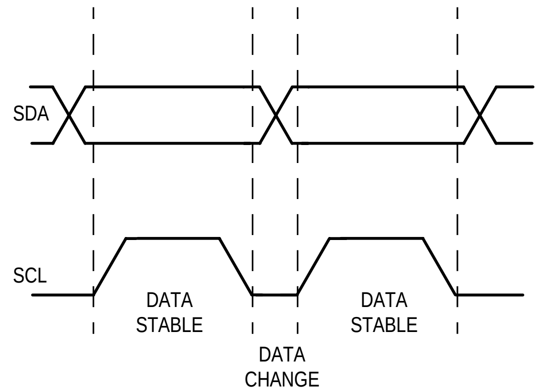

Figure 3. SCL Versus SDA Timing on I²C Bus

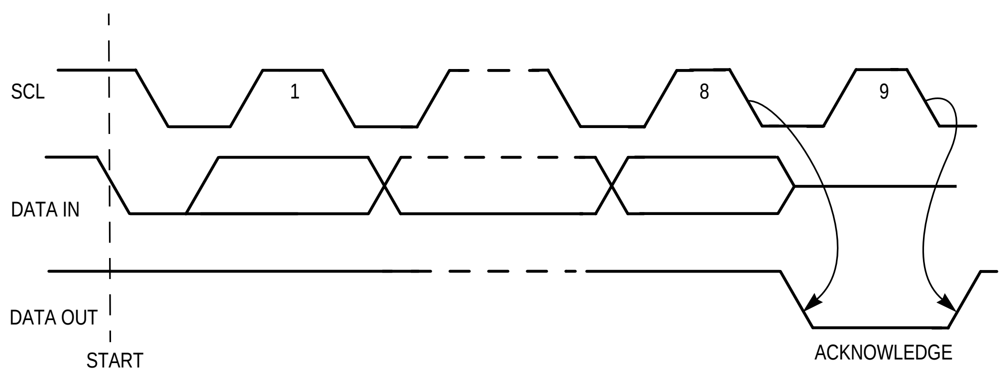

Figure 4. Acknowledge Bit Timing on I²C Bus

I²C Bus Transfer

The data transfer through the I²C bus is initiated by the START condition (START bit) produced by the master device. The start bit is followed by the device address byte with its most significant bit (MSB) first. The least significant bit (LSB) in the device address byte can be high or low, depending on whether it is a “read” or “write” operation.

With all bytes transferred on the I²C bus, a ninth clock cycle is used as an acknowledgment. The SDA line is read during this ninth clock cycle by the master and signifies whether or not the byte is acknowledged.

The device address byte is followed by two address bytes of the destination register. The high order byte is transferred first, the low order byte is transferred second. The 16-bit address is written to the slave device. At this point, there are two possible ways to continue:

- WRITE data to the destination register, determined by the 16-bit address previously written
- READ data from the destination register determined by the 16-bit address previously written

WRITE

If we write data to the register, the 16-bit address transferred on the I^{2}C bus is followed by the data byte intended to be written. After an acknowledge bit is signalled, a STOP condition is imposed on the I^{2}C bus to end the transfer. The graphical representation of this “byte write” transfer is shown in **Figure 5**.

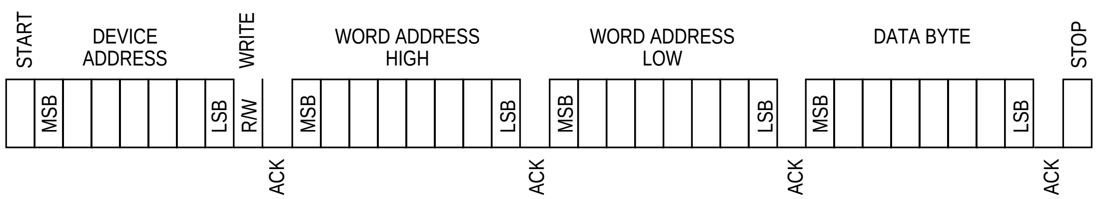

Figure 5. The “Byte Write” Transfer

READ

If we read data from the desired register, the 16-bit address transferred on the I^{2}C bus is followed by the START condition and the device address byte with LSB bit in the high logic level (READ function). After this byte is transferred, the slave produces the acknowledge bit, followed by the data from the desired register. The graphical representation of this “random read” transfer is shown in **Figure 6**.

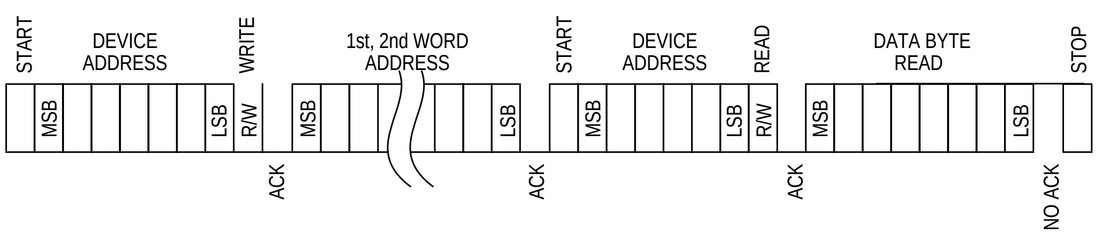

Figure 6. The “Random Read” Transfer

Program Description

The implemented software makes it possible to use the MCU as an I^2C slave device in various applications. This software routine is applicable on every MCU which can provide two I/O pins for I^2C bus connection. The MCU used can manage any of its own other tasks when the master terminates transfer. After its own tasks are completed, the slave MCU can then go back to receiving other data through the I^2C bus.

Every I^2C slave device must have a unique own device address. The value is set to A0 at address $FDE0 and can be individually changed.

In this software example, a simple write to the destination register function is used. The subroutine doing this is called write_func. A read function is achieved by the immediate read from the destination register.

If we must read data from the analog-to-digital converter (ADC) data register, the fresh data is available by starting the analog-to-digital conversion simultaneously with the I^2C bus transfer. This is possible because the analog-to-digital conversion needs only 17 MCU bus clock cycles, and it is completed before one byte is received through the I^2C bus.

Assembler Code

The assembler code that follows is available as a zip file, AN2509SW, from the Freescale website, http:www.freescale.com.

NOTE: *The software is not using the interrupt handling procedure because this would make the software relatively slow. The software is constructed as for as possible from consecutive queues of code, and the I^2C transfer speed reaches 75 kHz with the bus frequency of MCU equal to 3.2 MHz.*

```
********************************************************************************
HEADER_START
             Name:          I2C-ADC.ASM
             Project:       I2C slave implementation on HC908QT4/QY4
             Description:   QT1/4 parameter file
             Platform:      HC08
             Date:          Apr 2003
             Author:        Stanislav Arendarik
             Company:       Freescale
             Security:      General Business
================================================================================
 Copyright (c):      2003, All rights reserved.
================================================================================
THIS SOFTWARE IS PROVIDED BY FREESCALE "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF ERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL FREESCALE OR ITS CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
```

```
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
===============================================================================
 HEADER_END
********************************************************************************
          I2C Slave ADC + PORT R/W for QT, QY.
            SDA = PORTA.3,   SCL = PORTA.2
********************************************************************************
  DESCRIPTION :
This routine allows the use of the QT, QY MCU (Nitron) as a Slave I2C device. The HC908QT4
function is similar to a four channel AD converter and I/O port. Required function is ensured
by correct setting of the relevant registers.
For correct communication, the following byte format on I2C is required:
Write function:    /Standard "Byte write" function/
  1. - START bit condition
  2. - Valid device address - with R/W bit = 0
     this value is set in the program at address $FED0 with value "A0"
  3. - High byte of address of the destination register
  4. - Low byte of address of the destination register
  5. - Data byte desired to write to the destination register
  6. - STOP bit condition


Read function:     /Standard "Random read" function/
- START bit condition
- Valid device address - with R/W bit = 0 - Write function
       this value is set in the program at address $FED0 with value "A0"
  3. - High byte of address of the destination register
  4. - START bit condition
  5. - Valid device address - with R/W bit = 1 - Read function
  6. - At this point, I2C Slave sents the actual data value of desired register
  7. - STOP bit condition
********************************************************************************
             XDEF Entry,main,

             Include 'qtqy_registers.inc'
;            For the QT2, QT4, QY2, QY4
********************************************************************************
 Macros define I/O pins, which are designated for SCL and SDA of I2C
********************************************************************************
sda:         MACRO
               brclr 2,PORTA,\1
               brclr 3,PORTA,\2        ; SDA = PTA.3
             ENDM
wscl1:       MACRO
               brclr 2,PORTA,\1        ; SCL = PTA.2
             ENDM                      ; wait for SCL=1
wscl0:       MACRO
               brset 2,PORTA,\1        ; wait for SCL=0
             ENDM
tx0:         MACRO
               bset 3,DDRA             ; PTA3 as output
               bclr 3,PORTA            ; PTA3 = 0
             ENDM
```

```
slack:        MACRO
                bclr 3,PORTA            ; PTA3 = 0
                bset 3,DDRA             ; PTA3 as output
              ENDM
********************************************************************************
              org $FDE0
own_addr:     DC.B $A0      ; here the internal "Device Address" is set to value
                            ; $A0. The value can be changed.

DEFAULT_RAM            SECTION SHORT
r_w           equ $80  ; R/W flag on I2C (bit 0)
dev_addr      equ $81  ; internal device address
wr_byte1      equ  $82  ; high address byte for data to be written
wr_byte2      equ  $83  ; low address byte for data to be written
wr_byte3      equ  $84  ; data to be written
rd_byteCA     equ  $8B  ; data from current address
reg1          equ  $8C  ; auxiliary register
num_bit       equ  $8D  ; number of received bits
count1        equ  $8E  ; counter register 1
count2        equ  $8F  ; counter register 2

DEFAULT_ROM            SECTION

           Org  $FDFD  ; setting user RESET vector
DC.B  $CC
DC.W  $EE00
           Org  $EE00
********************************************************************************
  main - This is the point where code starts executing after RESET.
********************************************************************************
Entry:
main:     clr r_w
          clr dev_addr
          clr wr_byte1
          clr wr_byte2
          clr wr_byte3
          clr num_bit
          clr reg1
          mov #$80,CONFIG2   ; IRQ pullup disconnected,
                              ; RESET pin and IRQ pin are inactive
          mov #$09,CONFIG1   ; LVI in 5V operating mode, COP module disable
          lda $FFC0          ; load the TRIM value stored in FLASH
          cmp #$FF
          beq no_trim
          sta OSCTRIM        ; if the TRIM value has been calibrated and stored
                              ; previously, use this stored value.
no_trim:  rsp
          clra
          clrx
          mov #$40,ADICLK    ; set ADC clock
main_loop:
          bsr rec_all_b      ; receive all four bytes !!
          bra main_loop
```

```
******************************************************************************
rec_all_b -     this subr receives all four bytes together
  Wait for START condition, then receive :
  1.byte: dev addr + R/W command, send slave acknowledge impulse
  2.byte: high address byte, send slave acknowledge impulse
  3.byte: low address byte, send slave acknowledge impulse
  4.byte: data byte for writing
  Wait for STOP condition
******************************************************************************
************ Wait for START condition ***************
rec_all_b:  ldhx #$FDE0
            clra
            bclr 2,DDRA   ; set PORTA.2 as input = SCL
            bclr 3,DDRA   ; set PORTA.3 as input = SDA
w_start:    sta COPCTL    ; service cop control register to prevent reset
            lda PORTA
            and #$0C
            cmp #$0C
            bne w_start   ; wait for SDA and SCL = high
wait_sta:   sta COPCTL    ; service cop control register to prevent reset
                          ; this instruction is used to avoid being
            lda PORTA        ; held in the wrong state on I2C bus.
            and #$0C
            cmp #$04      ; SCL = high, SDA = low - START condition occurred
            bne wait_sta
            sei          ; disable interrupts while MCU is receiving through I2C

******************************************************************************
;    now begin receiving 4 bytes through I2C
************* Receive1.byte *************
read_funcRR:
            clra
            mov ADSCR,reg1   ; read ADSCR reg
            bclr 7,reg1      ; clear COCO bit in ADSCR
bit17:      wscl0 bit17      ; wait for SCL falling edge
            mov reg1,ADSCR   ; write ADSCR to start AD conversion
bit17a:     wscl1 bit17a     ; wait for SCL rising edge
            brclr 3,PORTA,jmp17     ; read byte1/bit7
            add #$01         ; read SDA = 1
jmp17:      lsla             ; read SDA = 0
            ; byte1/bit7 = MSB was received
w17:        wscl0 w17        ; wait for SCL = 0
bit16:      sda bit16,jmp16  ; wait for SCL = 1
            ; read byte1/bit6
            add #$01         ; read SDA = 1
jmp16:      lsla             ; read SDA = 0
w16:        wscl0  w16       ; wait for SCL = 0
bit15:      sda bit15,jmp15  ; wait for SCL = 1
            ; read byte1/bit5
            add #$01         ; read SDA = 1
jmp15:      lsla             ; read SDA = 0
w15:        wscl0  w15       ; wait for SCL = 0
bit14:      sda bit14,jmp14  ; wait for SCL = 1
            ; read byte1/bit4
            add #$01         ; read SDA = 1
```

```
jmp14:       lsla              ; read SDA = 0
w14:         wscl0 w14         ; wait for SCL = 0
bit13:       sda bit13,jmp13   ; wait for SCL = 1
             ; read byte1/bit3
             add #$01          ; read SDA = 1
jmp13:       lsla              ; read SDA = 0
w13:         wscl0 w13         ; wait for SCL = 0
bit12:       sda bit12,jmp12   ; wait for SCL = 1
             ; read byte1/bit2
             add #$01          ; read SDA = 1
jmp12:       lsla              ; read SDA = 0
w12:         wscl0 w12         ; wait for SCL = 0
bit11:       sda bit11,jmp11   ; wait for SCL = 1
             ; read byte1/bit1
             add #$01          ; read SDA = 1
jmp11:       lsla              ; read SDA = 0
w11:         wscl0 w11         ; wait for SCL = 0
             ; device address was received, now save data from ACC
             sta dev_addr      ; save "dev address" to RAM
             mov ADR,rd_byteCA    ; read ADC data to RAM
bit10:       sda bit10,jmp10   ; wait for SCL = 1
             ; read byte1/bit0
             bset 0,r_w                  ; read SDA = 1 = RD function
jmp10:                         ; read SDA = 0 = WR function
             cmp ,x            ; compare "ACC" with own address
             beq go            ; go to next
             jmp rec_all_b     ; go back to receive all four bytes
                               ; only if received "dev.addr" NOT match
                               ; with internal predefined "dev.address"
go:
w10:         wscl0 w10         ; wait for SCL = 0,then make sl_ack
             brclr 0,r_w,go1   ; if R/W = 0, then continue
             jmp read_funcCA   ; if R/W = 1, then make READ FUNCTION CA !
go1:
************* Make Slave acknowledge *************
             slack             ; begin the slave ACK
sl_ack_1a:   wscl1 sl_ack_1a   ; wait for SCL = 1
sl_ack_1b:   wscl0 sl_ack_1b   ; wait for SCL = 0
             bclr 3,DDRA       ; PTA3 as input
************* Receive 2. byte *************
             clra
bit27a:      brclr 2,PORTA,bit27a        ; wait for SCL rising edge
             brclr 3,PORTA,jmp27         ; read byte2/bit7
             add #$01          ; read SDA = 1
jmp27:       lsla              ; read SDA = 0
             ; byte2/bit7 = MSB was received
w27:         wscl0 w27         ; wait for SCL = 0
bit26:       sda bit26,jmp26   ; wait for SCL = 1
             ; read byte2/bit6
             add #$01          ; read SDA = 1
jmp26:       lsla              ; read SDA = 0
w26:         wscl0 w26         ; wait for SCL = 0
bit25:       sda bit25,jmp25   ; wait for SCL = 1
             ; read byte2/bit5
             add #$01          ; read SDA = 1
```

```
jmp25:      lsla              ; read SDA = 0
w25:        wscl0 w25         ; wait for SCL = 0
bit24:      sda bit24,jmp24   ; wait for SCL = 1
            ; read byte2/bit4
            add #$01          ; read SDA = 1
jmp24:      lsla              ; read SDA = 0
w24:        wscl0 w24         ; wait for SCL = 0
bit23:      sda bit23,jmp23   ; wait for SCL = 1
            ; read byte2/bit3
            add #$01          ; read SDA = 1
jmp23:      lsla              ; read SDA = 0
w23:        wscl0 w23         ; wait for SCL = 0
bit22:      sda bit22,jmp22   ; wait for SCL = 1
            ; read byte2/bit2
            add #$01          ; read SDA = 1
jmp22:      lsla              ; read SDA = 0
w22:        wscl0 w22         ; wait for SCL = 0
bit21:      sda bit21,jmp21   ; wait for SCL = 1
            ; read byte2/bit1
            add #$01          ; read SDA = 1
jmp21:      lsla              ; read SDA = 0
w21:        wscl0 w21         ; wait for SCL = 0
bit20:      sda bit20,jmp20   ; wait for SCL = 1
            ; read byte2/bit0
            add #$01          ; read SDA = 1
jmp20:                        ; read SDA = 0
            ; byte2/bit0 = LSB was received, now save data from ACC
            sta wr_byte1      ; save "wr_byte1" to RAM
w20:        wscl0 w20         ; wait for SCL = 0, then make sl_ack
************* Make Slave acknowledge *************
            slack             ; begin the slave ACK
sl_ack_2a:  wscl1 sl_ack_2a   ; wait for SCL = 1
sl_ack_2b:  wscl0 sl_ack_2b   ; wait for SCL = 0
            bclr 3,DDRA       ; PTA3 as input
************* Receive 3.byte *************
            clra
bit37a:     brclr 2,PORTA,bit37a      ; wait for SCL rising edge
            brclr 3,PORTA,jmp37       ; read byte3/bit7
            add #$01          ; read SDA = 1
jmp37:      lsla              ; read SDA = 0
            ; byte3/bit7 = MSB was received
w37:        wscl0 w37         ; wait for SCL = 0
bit36:      sda bit36,jmp36   ; wait for SCL = 1
            ; read byte3/bit6
            add #$01          ; read SDA = 1
jmp36:      lsla              ; read SDA = 0
w36:        wscl0 w36         ; wait for SCL = 0
bit35:      sda bit35,jmp35   ; wait for SCL = 1
            ; read byte3/bit5
            add #$01          ; read SDA = 1
jmp35:      lsla              ; read SDA = 0
w35:        wscl0 w35         ; wait for SCL = 0
bit34:      sda bit34,jmp34   ; wait for SCL = 1
            ; read byte3/bit4
            add #$01          ; read SDA = 1
```

```
jmp34:      lsla              ; read SDA = 0
w34:        wscl0 w34         ; wait for SCL = 0
bit33:      sda bit33,jmp33   ; wait for SCL = 1
            ; read byte3/bit3
            add #$01          ; read SDA = 1
jmp33:      lsla              ; read SDA = 0
w33:        wscl0 w33         ; wait for SCL = 0
bit32:      sda bit32,jmp32   ; wait for SCL = 1
            ; read byte3/bit2
            add #$01          ; read SDA = 1
jmp32:      lsla              ; read SDA = 0
w32:        wscl0 w32         ; wait for SCL = 0
bit31:      sda bit31,jmp31   ; wait for SCL = 1
            ; read byte3/bit1
            add #$01          ; read SDA = 1
jmp31:      lsla              ; read SDA = 0
w31:        wscl0 w31         ; wait for SCL = 0
bit30:      sda bit30,jmp30   ; wait for SCL = 1
            ; read byte3/bit0
            add #$01          ; read SDA = 1
jmp30:                        ; read SDA = 0
            ; byte3/bit0 = LSB was received, now save data from A
            sta wr_byte2      ; save "wr_byte2" to RAM
w30:        wscl0 w30         ; wait for SCL = 0, then make sl_ack
************* Make Slave acknowledge *************
            slack             ; begin the slave ACK
sl_ack_3a:  wscl1 sl_ack_3a   ; wait for SCL = 1
sl_ack_3b:  wscl0 sl_ack_3b   ; wait for SCL = 0
            bclr 3,DDRA       ; PTA3 as input
            clra
sl_ack_3c:  wscl1 sl_ack_3c   ; wait for SCL = 1
************* Receive 4.byte  or START *************
            brclr 3,PORTA,jmp47        ; read byte4/bit7
            add #$01          ; read SDA = 1
start3:     brset 3,PORTA,start3a      ; test START condition
            jmp read_funcRR   ; jump if START occurred
                              ; make READ FUNCTION RandomRead
start3a:    wscl0 start3      ; wait for SCL = 0
jmp47:      lsla              ; read SDA = 0
            ; byte4/bit7 = MSB was received
w47:        wscl0 w47         ; wait for SCL = 0
bit46:      sda bit46,jmp46   ; wait for SCL = 1
            ; read byte4/bit6
            add #$01          ; read SDA = 1
jmp46:      lsla              ; read SDA = 0
w46:        wscl0 w46         ; wait for SCL = 0
bit45:      sda bit45,jmp45   ; wait for SCL = 1
            ; read byte4/bit5
            add #$01          ; read SDA = 1
jmp45:      lsla              ; read SDA = 0
w45:        wscl0 w45         ; wait for SCL = 0
bit44:      sda bit44,jmp44   ; wait for SCL = 1
            ; read byte4/bit4
            add #$01          ; read SDA = 1
jmp44:      lsla              ; read SDA = 0
```

```
w44:        wscl0 w44        ; wait for SCL = 0
bit43:      sda bit43,jmp43  ; wait for SCL = 1
            ; read byte4/bit3
            add #$01         ; read SDA = 1
jmp43:      lsla             ; read SDA = 0
w43:        wscl0 w43        ; wait for SCL = 0
bit42:      sda bit42,jmp42  ; wait for SCL = 1
            ; read byte4/bit2
            add #$01         ; read SDA = 1
jmp42:      lsla             ; read SDA = 0
w42:        wscl0 w42        ; wait for SCL = 0
bit41:      sda bit41,jmp41  ; wait for SCL = 1
            ; read byte4/bit1
            add #$01         ; read SDA = 1
jmp41:      lsla             ; read SDA = 0
w41:        wscl0 w41        ; wait for SCL = 0
bit40:      sda bit40,jmp40  ; wait for SCL = 1
            ; read byte4/bit0
            add #$01         ; read SDA = 1
jmp40:                       ; read SDA = 0
            ; byte4/bit0 = LSB was received, now save data from A
            sta wr_byte3     ; save "wr_byte3" to RAM
w40:        wscl0 w40        ; wait for SCL = 0, then make sl_ack
************* Make Slave acknowledge *************
            slack            ; begin the slave ACK
sl_ack_4a:  wscl1 sl_ack_4a  ; wait for SCL = 1
sl_ack_4b:  wscl0 sl_ack_4b  ; wait for SCL = 0
            bclr 3,DDRA      ; PTA3 as input
************* Wait for STOP condition *************
w_stop:     sta COPCTL       ; service cop control register to prevent reset
            bclr 3,DDRA      ; set PORTA.3 as input
            bclr 2,DDRA      ; set PORTA.2 as input
            lda PORTA        ; test for SDA = 0, SCL = 1
            and #$0C
            cmp #$04
            bne w_stop       ; wait for SDA and SCL = high
wait_sto:   sta COPCTL       ; service cop control register to prevent reset
            lda PORTA        ; test for SDA=1, SCL=1 = STOP condition
            and #$0C
            cmp #$0C
            bne wait_sto     ; STOP condition occurred
            brset 0,r_w,endw    ; WRITE function only if R/W-bit = 0.
            jsr write_func
endw:       rts

********************************************************************************
;   write_func - write byte to address:    data from wr_byte3 will be written to
;              address pointed to by wr_byte2 (low byte) and wr_byte1 high byte)
********************************************************************************
write_func: ldhx wr_byte1    ; load wr_byte1 and wr_byte2 to H:X reg
            lda wr_byte2     ; load low address byte
            cmp #$04         ; test for address of DDRA register
            beq wr_func1     ; mask write to the DDRA register
            cmp #$0          ; test for address of PORTA
            beq wr_func2     ; mask write to the PORTA
```

```
            lda wr_byte3     ; load data to ACC
            sta ,x           ; write data to H:X address
            rts
wr_func1:   lda DDRA         ; load data from DDRA to A
            and #$0C         ; mask SCL and SDA bits
            sta reg1         ; save SCL and SDA value of DDRA
            lda wr_byte3     ; load data to write
            and #$F3         ; cut SCL and SDA bits
            ora reg1         ; OR original value of SCL & SDA DDRA bits
            sta DDRA         ; set new value of DDRA register
            rts
wr_func2:   lda PORTA        ; load data from PORTA to A
            and #$0C         ; mask SCL and SDA bits
            sta reg1         ; save SCL and SDA value of PORTA
            lda wr_byte3     ; load data to write
            and #$F3         ; cut SCL and SDA bits
            ora reg1         ; OR original value of SCL and SDA bits
            sta PORTA        ; set new value of PORTA
            rts

********************************************************************************
read_funcCA:
*************  Slave  ACK *************
            slack            ; begin the slave ACK
rca1:       wscl1 rca1       ; wait for SCL=1
            ldhx wr_byte1    ; load high and low address bytes for
                             ; current address read
            lda ,x           ; load data from current address
            sta rd_byteCA    ; save data to RAM
rca0:       wscl0 rca0       ; wait for SCL=0

*************  End of Slave ACK *************
            brclr 7,rd_byteCA,tx07     ; data (MSB) to SDA
            bset 3,PORTA
tx07:       wscl1 tx07       ; wait for SCL=1
            sta COPCTL
tx07a:      wscl0 tx07a      ; wait for SCL=0
            bclr 3,PORTA     ; clear SDA
            brclr 6,rd_byteCA,tx06     ; data (bit6) to SDA
            bset 3,PORTA
tx06:       wscl1 tx06       ; wait for SCL=1
            sta COPCTL
tx06a:      wscl0 tx06a      ; wait for SCL=0
            bclr 3,PORTA     ; clear SDA
            brclr 5,rd_byteCA,tx05     ; data (bit5) to SDA
            bset 3,PORTA
tx05:       wscl1 tx05       ; wait for SCL=1
            sta COPCTL
tx05a:      wscl0 tx05a      ; wait for SCL=0
            bclr 3,PORTA     ; clear SDA
            brclr 4,rd_byteCA,tx04     ; data (bit4) to SDA
            bset 3,PORTA
tx04:       wscl1 tx04       ; wait for SCL=1
            sta COPCTL
tx04a:      wscl0 tx04a      ; wait for SCL=0
```

```
              bclr 3,PORTA     ; clear SDA
              brclr 3,rd_byteCA,tx03     ; data (bit3) to SDA
              bset 3,PORTA
tx03:         wscl1 tx03       ; wait for SCL=1
              sta COPCTL
tx03a:        wscl0 tx03a      ; wait for SCL=0
              bclr 3,PORTA     ; clear SDA
              brclr 2,rd_byteCA,tx02     ; data (bit2) to SDA
              bset 3,PORTA
tx02:         wscl1 tx02       ; wait for SCL=1
              sta COPCTL
tx02a:        wscl0 tx02a      ; wait for SCL=0
              bclr 3,PORTA     ; clear SDA
              brclr 1,rd_byteCA,tx01     ; data (bit1) to SDA
              bset 3,PORTA
tx01:         wscl1 tx01       ; wait for SCL=1
              sta COPCTL
tx01a:        wscl0 tx01a      ; wait for SCL=0
              bclr 3,PORTA     ; clear SDA
              brclr 0,rd_byteCA,tx00     ; data (LSB) to SDA
              bset 3,PORTA
tx00:         wscl1 tx00       ; wait for SCL=1
              sta COPCTL
tx00a:        wscl0 tx00a      ; wait for SCL=0
              bclr 3,PORTA     ; clear SDA
************  N O    A C K ************
              bclr 2,DDRA      ; set PORTA.2 as input
              bclr 3,DDRA      ; set PORTA.3 as input
w_stop1:      sta COPCTL       ; service cop control register to prevent reset
              lda PORTA        ; test for SDA = 0, SCL = 1
              and #$0C
              cmp #$04
              bne w_stop1      ; wait for SDA and SCL = high
wait_sto1:    sta COPCTL       ; service cop control register to prevent reset
              lda PORTA        ; test for SDA = 1, SCL=1 = STOP condition
              and #$0C
              cmp #$0C
              bne wait_sto1
              lda ADSCR        ; clear COCO bit in ADSCR to start
              and #$7F         ; next conversion
              sta ADSCR
              clr r_w          ; clear RD flag
              bclr 2,DDRA      ; set PORTA.2 as input
              bclr 3,DDRA      ; set PORTA.3 as input
              jmp main_loop    ; go to start of I2C communication
********************************************************************************
END
```

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