FlashFlex Microcontroller
Software I²C Emulation

Application Note
July 2008

1.0 INTRODUCTION

The I²C bus (pronounced eye-squared-see), or IIC, stands for the Inter-IC bus. Philips invented the I²C protocol as an economical, effective, and efficient serial transfer protocol. The original I²C bus only requires two lines for data transfer and can achieve speeds of 100 Kbps. The first line is named SDA and is used to transfer data. The second line is called SCL and is the clock by which all data transfers in the bus must adhere to. The clock is generated by the master and received by the slave. There are possibilities within the I²C protocol to support multiple master, multiple slave buses. However, this software implementation will only show how one can use a single software controlled I²C master to communicate with one or more I²C slaves.

2.0 I²C BASICS

The I²C protocol is very simple and efficient for its uses. It is a serial transfer bus and therefore data is transferred one bit at a time. The protocol is comprised of several integral components: The Start signal, slave address, data byte(s), Stop signal. Each data transfer begins with a Start signal. As shown in Figure 2-1, the Start signal is defined by the SDA line transitioning from high to low while the SCL line is held high. After sending the Start signal, the slave address is sent. The slave address is composed of 4 fixed bits, 3 programmable bits, and one Read/Write bit. The 3 programmable bits are used to specify the slave device to be selected. The final bit is used to tell the slave whether this transfer is transmit-to-slave or a receive-from-slave. This is followed by 1 or more, 8 bit, data bytes. Each data byte is followed by an ACK (Acknowledge) or a NACK (Not Acknowledge) generated by the receiver. Each data bit is transferred by lowering the SCL line, then changing the SDA line state to reflect the bit to be transferred, then raising the SCL line. The actual value of the SDA line is read on the high level of the SCL. Because of the nature of the Start and Stop signals, the I²C protocol forbids the SDA line to change states while the SCL line is high to avoid the accidental generation of Start or Stop, which is shown in Figure 2-2. After the eighth data bit is transferred, the SDA line is left in a high state. An ACK is then generating by the receiver pulling the SDA line low for one SCL cycle. A NACK is generated by leaving the SDA line high for one SCL cycle. Refer to Figure 2-3 for timing specifications. The receiver will keep receiving bytes in this fashion until the Stop signal is received. A Stop signal is defined by the SDA line transitioning from low to high while the SCL line is held high. Figure 2-4 shows the general flow of an I²C data transfer.

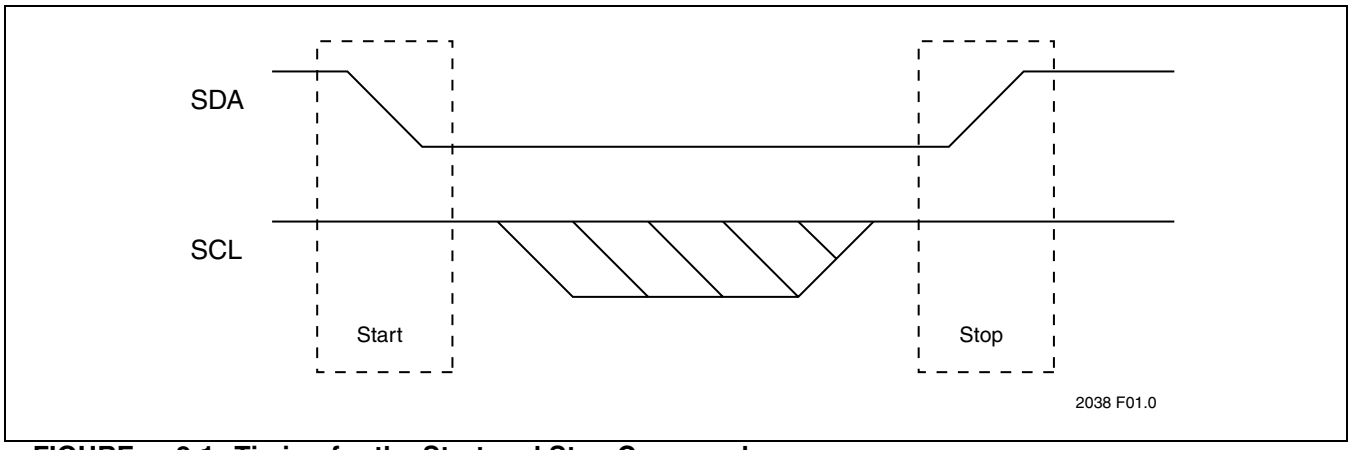

FIGURE 2-1: Timing for the Start and Stop Commands

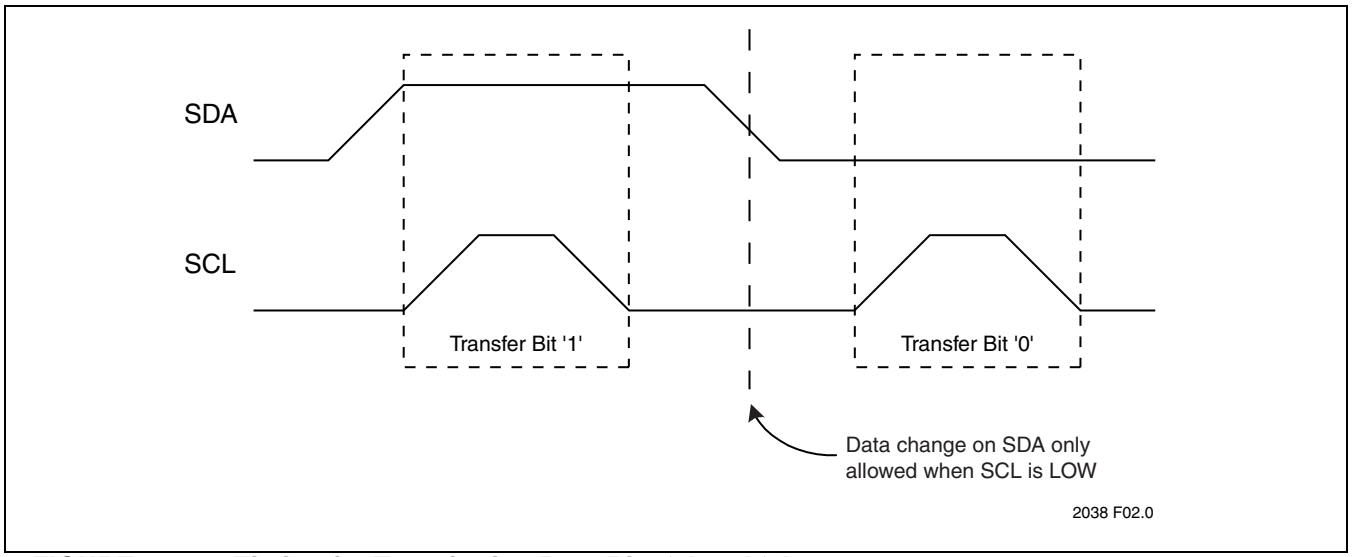

FIGURE 2-2: Timing for Transferring Data Bits ‘1’ and ‘0’

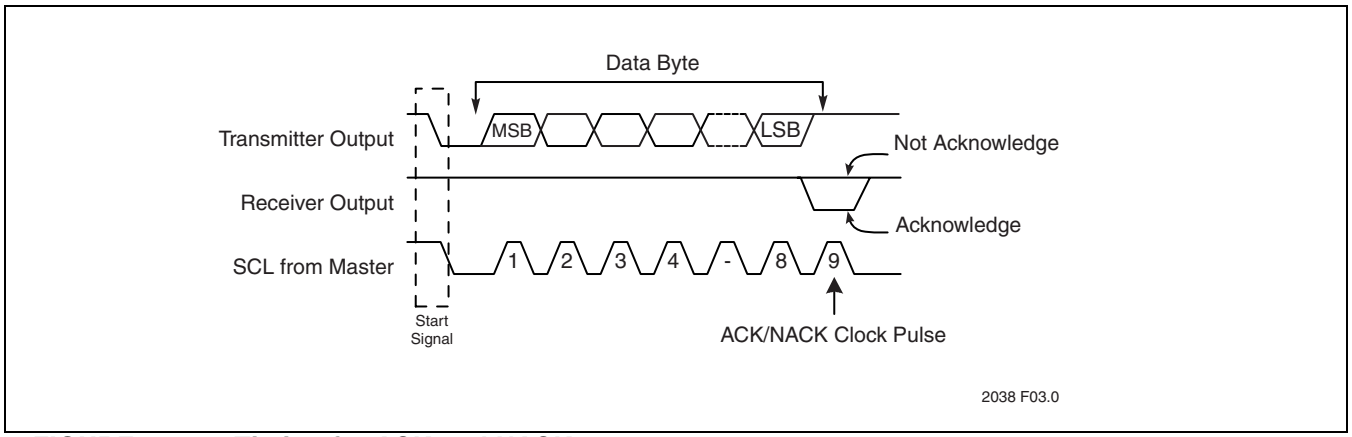

FIGURE 2-3: Timing for ACK and NACK

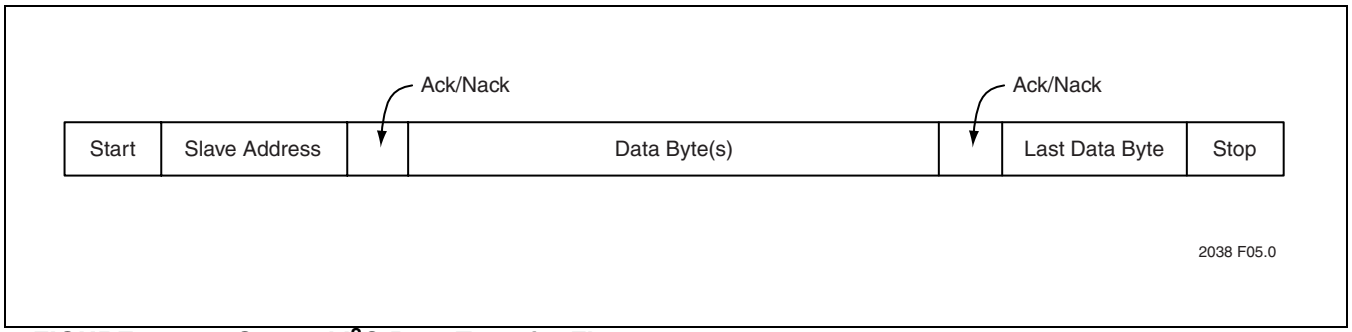

FIGURE 2-4: General I2C Data Transfer Flow

3.0 HARDWARE REQUIREMENTS

There are some hardware considerations for I2C. The two lines chosen for SDA and SCL must be connected to a positive voltage source via a pull-up resistor. A typical pull-up resistor value is 4.7 KΩ. The pull-up resistor leaves the bus signal line in a high state whenever it is not being used. There are also signal setup and hold time restrictions. For basic I2C, signal hold time is 4.7 µs minimum. The entire set of I2C specifications can be found on the Philips website.

3.1 Hardware setup

This code was designed for, and verified with, an SST 89V516RDx microcontroller master and a National Semiconductor NM24C08M serial flash slave. Figure 3-1 shows the schematic of the reference test platform. Please note that the page address lines (A_0:A_1) are permanently grounded in this test platform. This is to make the test a little simpler and does not detract, to any degree, from the functionality of the code. The source code contained will work for any page in the memory space even though the test device was hard-wired to use only page 0.

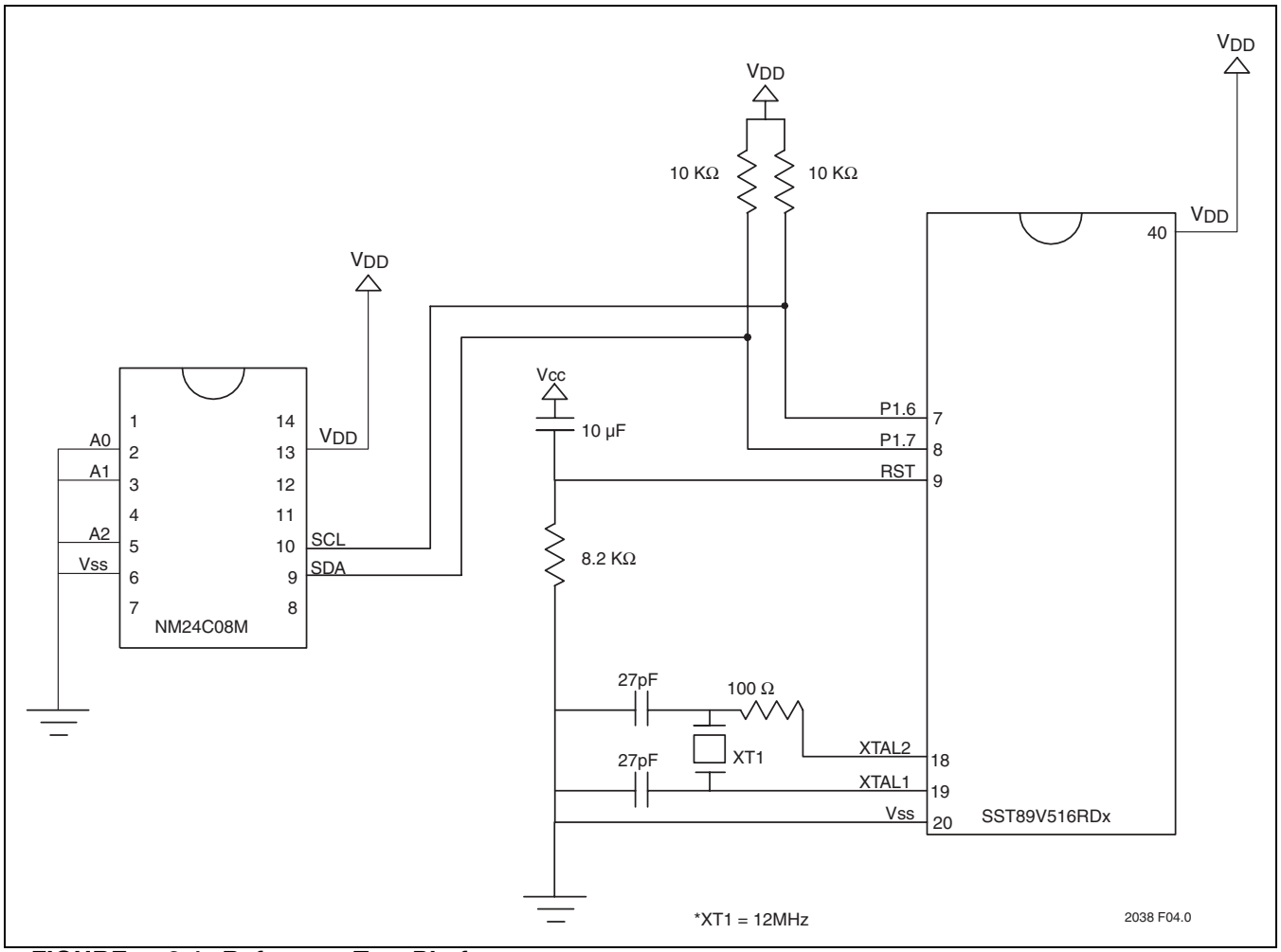

FIGURE 3-1: Reference Test Platform

4.0 SOURCE CODE

```
;=====================================================================
; Variable Declarations
;=====================================================================
SDA             EQU     p1.7                    ; Data Line.
SCL             EQU     p1.6                    ; Clock Line.

TxCnt           DATA    30h                     ; Number of bytes to transmit.
RxCnt           DATA    31h                     ; Number of Bytes to receive.
Slave_Adr       DATA    32h                     ; Slave Device address.
Byte_Adr        DATA    33h                     ; Byte address on slave device.
RxBuf           DATA    40h                     ; Buffer where received data is stored.
                                                ; Maximum of 16 bytes of storage per
                                                ;  operation.
Data_start      DATA    50h                     ; Start of test data.

;=====================================================================
;               MAIN PROGRAM
;=====================================================================
                org     0000h
                ljmp    main

                org     0100h
main:
                mov     sp, #60h                ; Move stack pointer.
                acall   setup                   ; Setup test data.
                mov     TxCnt, #08              ; Transfer 8 bytes.
                mov     Slave_Adr, #0A0h        ; Set slave address.
                mov     Byte_Adr, #66h          ; Set address to be programmed on slave.

                acall   START                   ; Send START signal.
                acall   Tx_Data                 ; Send test data. This
                                                ;  includes sending slave address,
                                                ;  byte address, and checking for ACKs.
                acall   STOP                    ; Send STOP signal.

                acall   Write_delay             ; Wait for the test device to write
                                                ;  test data to memory.

                mov     RxCnt, #8               ; Read 8 bytes.
                acall   START                   ; Send START signal.
                acall   Rx_Data                 ; Read back the test data.
                acall   STOP                    ; Send STOP signal.
```

```
;=====================================================================
; Check to see if serial flash was properly programmed
;=====================================================================
                mov     r0, #Data_start
                mov     r1, #RxBuf
                mov     r7, #08
Pass_Check:
                mov     A, @r0
                mov     B, @r1
                cjne    A, B, FAIL1             ; Compare the test original test data
                                                ;  stored at Data_Start (50h) with the
                                                ;  data read back from the slave device
                                                ;  stored at RxBuf (40h).  If all bytes
                                                ;  match, the test passes, if not, the
                                                ;  test fails
                inc     r0
                inc     r1
                djnz    r7, Pass_Check
                ajmp    PASS                    ; Test pass.


FAIL1:                                          ; Test failed.
                anl     p1, #11000000b
                orl     p1, #00000001b
                sjmp    $

PASS:                                           ; Test Passed.
                anl     p1, #11000000b
                orl     p1, #00001111b
                sjmp    $

;=====================================================================
;               SUBROUTINES
;=====================================================================
; Setup test data.
;=====================================================================
setup:
                mov     50h, #55h
                mov     51h, #45h
                mov     52h, #23h
                mov     53h, #0ffh
                mov     54h, #0f0h
                mov     55h, #0fh
                mov     56h, #0aah
                mov     57h, #77h
                ret


;=====================================================================
; Transmit a string of data.
; Before Call:          Slave address saved in Slave_Adr.
;                       Byte Address to be programmed saved in Byte_Adr.
;                       Number of bytes to be transferred saved in TxCnt.
;                       For NM24C08M, TxCnt MAX is 16.
; After Call:           None.
;=====================================================================
```

```
;====================================================================
Tx_Data:
                mov     r0, #Data_start         ; Point to data.
                mov     A, Slave_Adr            ; Transmit Slave address.
                acall   TxByte
                acall   Get_Ack                 ; Get acknowledge from slave.
                mov     A, Byte_Adr             ; Send Byte address on slave.
                acall   TxByte
                acall   Get_Ack                 ; Get acknowledge from slave.
TxByte_loop
                mov     A, @r0                  ; Send 1 byte of data
                acall   TxByte
                acall   Get_Ack                 ; Get acknowledge from slave.
                inc     r0
                djnz    TxCnt, TxByte_loop      ; If more bytes to be
                                                ; transferred, repeat.
                ret


;====================================================================
; Read anywhere from 1 byte to an entire page (256 bytes) of data.
; Before Call:          Number of Bytes to be read saved in RxCnt.
;                       Slave address saved in Slave_Adr.
;                       Starting Byte Address saved in Byte_Adr.
; After Call:           Read Bytes saved in RxBuf.
;====================================================================
; NOTE: Though this function will support the NM24C08M's function of sequential
;  read to its full extent of 256 bytes, please remember that the RxBuf
;  size is only 16 bytes.  If single transfers of more than 16 bytes are
;  required, then either the stack needs to be reallocated to allow for more
;  room in RAM, or the received data needs to be written to flash or external
;  memory as it is being read in.
;====================================================================
Rx_Data:
                mov     r0, #RxBuf
                mov     A, Slave_Adr            ; Send Slave address with
                                                ;  R/W bit set to "Write".
                acall   TxByte
                acall   Get_Ack
                mov     A, Byte_Adr             ; Send Byte Address.
                acall   TxByte
                acall   Get_Ack

                inc     Slave_Adr               ; Set Slave address to "READ".

                acall   START                   ; Re-send START signal.
                mov     A, Slave_Adr            ; Re-send READ Slave address.
                acall   TxByte
                acall   Get_Ack
RxByte_loop:
                acall   RxByte                  ; Receive data byte
                mov     @r0, A                  ; Save data byte.
                djnz    RxCnt, Not_Done
                sjmp    Done                    ; Skip the last Send_Ack since no
                                                ;  more bytes are to be received.
Not_Done:
                acall   Send_Ack
                inc     r0
                sjmp    RxByte_loop
Done:
                ret
```

```
;====================================================================
; Get Acknowledgment from slave.
; Pass Condition:     Program Continues.
; Fail Condition:     Program halts and jumps to FAIL1 loop.
;====================================================================
Get_Ack:
                    setb    SDA             ; Reset both address and clock Lines.
                    setb    SCL
                    acall   delay           ; Satisfy timing requirement by waiting.
                    jb      SDA, FAIL1      ; If slave sends ack, SDA would be
                                            ; pulled low.  If SDA is not low,
                                            ; program halts and jumps to FAIL1.
                    clr     SCL
                    ret

;====================================================================
; Send Acknowledgment to slave.
;====================================================================
Send_Ack:
                    clr     SDA
                    acall   delay
                    acall   SCLPulse
                    setb    SDA
                    ret

;====================================================================
; Send STOP signal to slave
;====================================================================
STOP:
                    clr     SDA
                    setb    SCL
                    acall   delay
                    setb    SDA
                    acall   delay
                    ret

;====================================================================
; Send START signal to slave
;====================================================================
START:
                    orl     p1, #11000000b
                    acall   delay
                    clr     SDA
                    acall   delay
                    clr     SCL
                    acall   delay
                    ret
;====================================================================
; Receive one single byte from slave.
; Before Call:        None.
; After Call:         Received Byte Stored in Acc.
;====================================================================
RxByte:
                    mov     r7, #8
RxBit_loop:
                    setb    SCL
                    acall   delay
                    mov     C, SDA
                    clr     SCL
                    RLC     A
                    nop
                    nop
                    djnz    r7, RxBit_loop
                    ret
```

```
;====================================================================
; Transmit one single byte to slave.
; Before Call:      Byte to be transferred is saved in Acc.
; After Call:       None.
;====================================================================
TxByte:
                  mov     r7, #8
TxBit_loop:
                  rlc     A
                  mov     SDA, C
                  acall   delay
                  acall   SCLPulse
                  djnz    r7, TxBit_loop
                  ret

;====================================================================
; Pulses the clock line while taking into consideration timing needs.
; Clock pulses from low to high to low.  Initial Low state of SCL line
; is assured due to the fact that all functions that manipulate
; the SCL line place the SCL line in a low state before exiting.
;====================================================================
SCLPulse:
                  setb    SCL
                  acall   delay
                  clr     SCL
                  acall   delay
                  ret

;====================================================================
; Creates a 10,000 machine cycle delay. (10ms for 12 MHz clock)
; This is used to satisfy timing after each Write operation as the
;  NM24C08M requires a maximum of 10ms to complete any write operation.
;  This function will need to be modified to delay at least 10ms if
;  using a crystal faster than 12 MHz.
;====================================================================
Write_delay:
                  mov     r5, #0
                  mov     r6, #40
Write_delay_loop:
                  djnz    r5, $
                  djnz    r6, Write_delay_loop
                  ret

;====================================================================
; Creates a 5 machine cycle delay. (5us for 12MHz clock)
; This is used to satisfy timing after each SDA or SCL level change.
; This function will need to be modified to delay at least 4.7µs if
;  using a crystal faster than 12 MHz.
;====================================================================
delay:
                  nop
                  ret

                  end
```