

# Routines for 2-Wire FRAM Using Microchip MCUs



Author: Bob Axtell, Micro-Firmware

Phone: 520-219-2363 eFax: 702-995-5281

Web: <http://www.micro-firmware.com>

## Overview

This application note describes bulletproof random write- and read- block routines written specifically for the Ramtron FM24C04 and FM24C16 for Microchip microcontrollers (MCUs) where the processor is the only Master. A third routine is provided to sense the presence of the chip (in case your device is socketed). Although the example here is used on the PIC16C7X, PIC16C8X, and the flash versions of the same devices, these routines are adaptable for almost all PICs as no internal 2-wire (I<sup>2</sup>C) hardware is used.

Micro-Firmware was prompted to write this application note when it became apparent that most application literature ignored Ramtron's devices, which are much easier to implement and devoid of the restrictions imposed by standard EEPROM devices. Due to the fact that Ramtron devices require no write delays, and no restrictions exist on the number of bytes that can be written. You can write the entire FRAM in a single pass! When combined with FRAM's data storage lifetime (I've never met anybody who's seen a bad device), the combination of the Microchip MCU plus FRAM memory is unbeatable.

For clarity, consult Ramtron's "FM24C04 Data Sheet" when using this application note.

## Hardware

The data pin of the FRAM must be pulled up with a suitable resistor, since the FRAM device can only pull down. Highest speeds will result from a low value pull-up, as the rise times are reduced. Ramtron recommends 1.8K as a minimum; in this application we used 3K, using a PIC16C73B and a 12M oscillator.

For highest noise immunity, this application actively drives the data pin to both states unless reading from the FRAM, even when the 2-wire bus is idle.

Although the Ramtron data sheet indicates that the clock pin should also have a resistor, in a single master system there is no reason to have it, as the clocks are driven to both levels. Only when another master can seize the line is a pull-up resistor needed.

Driving the clock improves the noise immunity of the 2-wire system somewhat.

This application shows the 2-wire pins on portc:3 and portc:4, but in fact they can be on any pin, on any port, as long as they can drive both states.

## Software

The code that follows contains three routines actually used by EDTec in several designs. The code will work properly for all Microchip clock speeds (up to 20MHz) without timing problems. All three routines consume less than 170 program code locations. The NOPs are present to show where a wait would have to go in case it is needed. Older Ramtron devices may need the waits.

The routines below can be interrupted without problem, but they should not operate within an interrupt routine due to the PIC's limited stack (8 levels max). Previous MCUs have a 2-level stack, and some calls within TX and RX will have to be rewritten into macros, to eliminate the call.

The temporary registers may be used elsewhere in non-interrupt code.

The code begins by ensuring that the 2-wire bus and the FRAM device are both in an idle state by issuing a STOP, then a START, followed by another STOP condition. A non-idle state can occur because an inadvertant START could have been issued when the system powered up.

HINT: It is usually a good idea to determine the actual SIZE of the FRAM when first powering up, since there is no other way to know.

## Write Block Routine

The write block routine is error-checked at every acknowledge point, and if any error is found, the pass will abort and start over. This allows the 2-wire to exist in a very noisy environment and still provide error-free data. The routine will attempt up to 4 passes before giving up: TMP3 is the variable set to 4. The error flag is cleared and a much-used temp counter is also cleared. Then a start condition is created to begin the command.

The slave address is transferred, 8 bits, A7 first, data true. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, that device might not be present on the 2-wire bus; in fact, this is the basis for the third routine which stops at this point.

If there was an Acknowledge, the internal 8-bit FRAM address is transferred, A7 first. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, the reason is usually that the clock signal was not properly received, and the routine will retry.

If there was an Acknowledge, the first byte to be transferred is sent, A7 first. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, the reason is usually that the clock signal was not properly received, and the routine will retry. This is repeated until all bytes have been transferred and the final acknowledge received, after which a STOP is issued.

If there were less than 4 repeats, TMP3 will be non-zero, indicating a successful write command. If TMP3 is 0, the command failed.

### Read Block Routine

Similar to the write block routine above, the read block routine is error-checked at every acknowledge point, and if an error is found, will abort the pass and start over. The routine will attempt up to 4 passes before giving up: TMP3 is the variable set to 4. Other variables are cleared and a Start condition is created to begin the command.

The slave address is transferred, 8 bits, A7 first, data true. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, that device might not be present on the 2-wire bus; in fact, this is the basis for the third routine which stops at this point.

If there was an Acknowledge, the internal 8-bit FRAM address is transferred, A7 first. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, the reason is usually that the clock signal was not properly received, and the routine will retry.

If there was an Acknowledge, there is an immediate stop followed by an immediate Start. This halts the write block command and now will read the byte at the FRAM address pointed to.

The slave address is again transferred, 8 bits, A7 first, data true, except that A0 is a 1, indicating that a random read is required. At the end of this transfer, an Acknowledge from the FRAM is expected. If no Acknowledge is received, the routine will retry.

If there was an Acknowledge, the FRAM is now in the Random Read Mode, and will output the data pointed to by the internal FRAM address sent earlier. The data is transferred A7 first. At the end of this transfer, an Acknowledge from the MCU is provided to the FRAM, to let it know that it must continue to provide data bytes. This repeats until the MCU has received all the bytes it needs, whereupon to stop, it fails to provide an Acknowledge, and then issues a STOP condition. If TMP3 is 0, there were 4 failed attempts, otherwise the last one was successful and the data is valid.

### Test for Device on Bus

The test for device present simply uses the first few steps of a write block command, except that it STOPS after getting the very first Acknowledge.

### Demo Program

A demo program was devised to allow simple verification of the three routines. The demo program (begins at main) will first verify that the FRAM is actually present, will clear the first 256 bytes of the FM24C04, then write h'A5 and h'5A at FRAM location h'80' & h'81'.

Portb is h'00 when the unit powers up. If no errors occur, then portb:7 is set high and processing will halt. If there is an error, various pins of portb will be set according to the problem found: if unable to find the FRAM, portb:0 is high; if unable to clear the 256 bytes in FRAM, portb:1 is high; if unable to write to FRAM location \$80-\$81, portb:2 is high; if unable to read from the FRAM at location \$80/\$81, portb:3 is high; or if the data read was simply incorrect, portb:4 is set high. This simple error check allows a programmer to verify the routine with a simple voltmeter; no oscilloscope is needed.

### Problems

If problems are encountered with the application note, let us know and we'll correct it immediately. Send your comments to [engineer@cotse.net](mailto:engineer@cotse.net).

```

;*****
title    "Routines for 2-Wire FRAM Using Microchip MCUs"
subtitle "Apr 10, 1999"
;
        list  p=pic16c73a
        list  c=80,mm=off,st=off,t=on,w=1
        list

;=====
; assembler used is MicroChip MPLAB 4
;----- destination definitions for assembler-----
w          equ   h'00'
f          equ   h'01'
;----- register files -----
indf       equ   h'00'
;
status     equ   h'03'
fsr        equ   h'04'
; although this app note shows PortC bits 3&4 being used as
; the i2c interface, almost any pin on any port will work fine
porta      equ   h'05'
portb      equ   h'06'
portc      equ   h'07'
portd      equ   h'08'
; a3 = SMB Clock+ {o} I2C Clock
; a4 = SMB Data+  {i} I2C Data
; equates for portc
SCL        equ   3
SDA        equ   4
; hi bank fsrs
option_reg equ   h'81'
trisa      equ   h'85'
trisb      equ   h'86'
trisc      equ   h'87'
trisd      equ   h'88'
;----- status bits -----
z          equ   h'02'
c          equ   h'00'
; Macros
; set lower register bank
bank0 macro
    bcf    status,5
endm
; set upper register bank
bank1 macro
    bsf    status,5
endm
;
; RAM area starts
; Bank 0 Internal Ram
flags     equ   h'20'
; A0: Temp Flag, Hi if "No Ack" from FRAM
tmp       equ   h'21'      ; used in i2c; can be used elsewhere
tmp1      equ   h'22'      ; as long as there is no conflict
tmp2      equ   h'23'
tmp3      equ   h'24'
tmp4      equ   h'25'
tmp5      equ   h'26'
sadr      equ   h'27'      ; slave adr of FRAM
i2cnt     equ   h'28'      ; #bytes to write into FRAM

```

```

i2adr  equ    h'29'      ; Internal FRAM adr
i2ptr  equ    h'2a'      ; uP Orig Adr of Bytes to send
i2buf  equ    h'2b'      ; useable as buffer area to rd/wr
;*****
      org    0
;*****
;   program starts here
;*****
      nop                ; out of habit, I'm an ol' Z80 guy
      clrf   porta
      clrf   portb
      movlw  h'18'
      movwf  portc      ; Set SDA, SCL to hi state (normal)
;   clrf   portd      ; use 'em if ya got 'em
      bankl
      movlw  h'85'      ; 10000110b **
      movwf  option_reg
      clrw
      movwf  trisa      ; set to all outputs
      movwf  trisb
      movwf  trisc
;   movwf  trisd      ;
; only registers needed buy i2c routines are
; initialized.
      goto   main
;*****
; I2C Routines: Do not access within interrupts!
;*****
;*****
; write i2cnt bytes @^i2ptr to FRAM @i2adr
;*****
;*****
;   Write Block Of Data to FRAM Device   ;
; variables used but unchanged:         ;
;   sadr = slave adr of FRAM (usually $a0);
;   i2cnt = # bytes to write into FRAM   ;
;   i2adr = Internal FRAM adr to put Data ;
;   i2ptr = uP Orig Adr of Bytes to send ;
; variables used, changed:              ;
;   w, tmp, tmp1-tmp4, flags:0          ;
; if error detected, tries four times   ;
; if successful, tmp3 > 0                ;
;*****
i2wr:  call   Bstrtx      ; Generate START bit
i2wry:  movf   sadr,w      ; Put raw slave address onto bus
      movwf  tmp
      call   TX          ; Output RAW SLAVE data address
      btfsc  flags,0     ; Check for no ack error
      goto  BSTOPw      ; To error Rtn
      movf  i2adr,w      ; Put slave internal data address onto bus
      movwf  tmp        ; Xmit buffer
      call  TX          ; Output WORD address. Check ACK.
      btfsc  flags,0     ; Check for no ack error
      goto  BSTOPw
      movf  i2cnt,w      ; load byte count to tmp4
      movwf  tmp4
      movf  i2ptr,w      ; load origination address to fsr
      movwf  fsr
i2wr1:  movf  indf,w      ; Move DATA
      movwf  tmp        ; into transmit buffer

```

```

        call    TX            ; Output DATA and detect acknowledgement
        btfsc  flags,0       ; Check for no ack error
        goto   BSTOPw       ;
        incf   fsr,f         ; next byte
        decfsz tmp4,f        ;
        goto   i2wrl        ;
        goto   BSTOP        ; Generate STOP bit
;*****
; read i2cnt bytes @^i2ptr fr FRAM @i2adr
;*****
;*****
; Read Block Of Data fr FRAM Device ;
; variables used but unchanged: ;
; sadr = slave adr of FRAM (usually $a0);
; i2cnt = # bytes to read from FRAM ;
; i2adr = Internal FRAM adr to get Data ;
; i2ptr = uP Dest Adr of Bytes Received ;
; variables used, changed: ;
; w, tmp, tmp1-tmp4,flags:0 ;
; if error detected, tries four times ;
; if successful, tmp3 > 0 ;
;*****
i2rd:   call    Bstrtx       ; Generate START bit
i2rdy:  movf    sadr,w       ; put raw slave address onto bus
        movwf  tmp
        call   TX            ; Output RAW SLAVE data address
        btfsc  flags,0       ; Check for error
        goto   BSTOPr       ; To Error Rtn
        movf   i2adr,w       ; Put slave internal data address onto bus
        movwf  tmp           ; Xmit buffer
        call   TX            ; Output WORD address. Check ACK.
        btfsc  flags,0       ; Check for error
        goto   BSTOPr       ;
        call   BSTOP
; at this point, the FRAM address is now set
; data may collides for a partial bit here; ignore..
        call   BSTART       ; START READ
        movf   sadr,w       ; slave adr
        movwf  tmp
        bsf    tmp,0        ; Specify READ mode (R/W = 1)
        call   TX            ; Output SLAVE address
        btfsc  flags,0       ; Check for error
        goto   BSTOPr       ;
        movf   i2cnt,w
        movwf  tmp4
        movf   i2ptr,w
        movwf  fsr
i2rd1:  call    RX            ; READ in data and provide ack
        movf   tmp1,w
        movwf  indf         ; Save data into buffer
        incf   fsr,f         ; next byte
        decfsz tmp4,f        ;
        goto   i2rd1
        goto   BSTOP        ; Generate STOP bit
; RECEIVE eight data bits subroutine
; exit = w has data
RX:     bsf    tmp2,3        ; 8 bits of data
RXLP:   call   BITIN        ; get data
        rlf    tmp1,f        ; install bit, LS last
        decfsz tmp2,f        ; 8 bits?

```

```

        goto    RXLP
        bsf     status,c    ; try no ack
        decfsz tmp4,w      ; last one?
        bcf     status,c    ; needs ack
        goto    BITOUT
; TRANSMIT 8 data bits subroutine in tmp
TX:     bsf     tmp2,3      ; 8 bits of data
        clrwdt
TXLP:   rlf     tmp,f       ; Shift data bit out.
        call    BITOUT      ; Serial data out
        decfsz tmp2,f      ; 8 bits done?
        goto    TXLP       ; No.
        call    BITIN      ; Read acknowledge bit
        btfss  status,c    ; acknowledgement?
        retlw  0           ; yep, done
        bsf     flags,0    ; not received
        retlw  0
; Single bit receive from I2C to PIC
; data in carry
BITIN:  bank1
        bsf     trisc,SDA  ; Set SDA for input
        bank0
        nop
        bsf     portc,SCL  ; Clock high
        bcf     status,c   ; def=0 (has ack)
        btfsc  portc,SDA  ; Read SDA pin, for ACK low
        bsf     status,c   ; no ack detected
        nop
        bcf     portc,SCL  ; finish bit in case
        nop
        retlw  0
; Single bit data transmit from PIC to I2C
; Input= carry bit
BITOUT: btfss  status,c
        goto    BIT0
        bsf     portc,SDA  ; drive it high
        goto    CLK1
BIT0:   bcf     portc,SDA  ; drive low
CLK1:   bank1
        bcf     trisc,SDA  ; Output bit 0
        bank0
        bsf     portc,SCL
        nop
        bcf     portc,SCL  ; on low
        retlw  0
; start but also set loop count
Bstrtx: bsf     tmp3,2      ; 4 x max
Bstrty: bcf     flags,0    ; no error yet
        clrfs  tmp2       ; pre-clear to 00
; START bit generation routine
; Generate START bit (SCL is high while SDA goes from high to low transition)
; quiescent state is both high....
BSTART: bank1
        bcf     trisc,SDA  ; both driven
        bcf     trisc,SCL
        bank0
        clrwdt            ; need one someplace
        bcf     portc,SDA  ; SDA=lo = START CDX
        nop
        bcf     portc,SCL

```

```

        nop
        retlw 0
; write repeat stop; deeper stack not needed
BSTOPw: call    BSTOP    ; issue a full stop
        call    Bstrty   ; new start
        decfsz tmp3,f
        goto    i2wry
        goto    BSTOP
; read repeat stop
BSTOPr: call    BSTOP
        call    Bstrty
        decfsz tmp3,f
        goto    i2rdy
; STOP bit generation routine
;Generate STOP bit (SDA goes from low to high during SCL high state)
BSTOP:  bank1
; loop until SDA is released
; NOTE: needed in case FRAM is driving SDA and a stop
; would be ignored. Havoc might result! Max# clks is 9
; but usually only 1 is needed
        bsf    trisc,SDA
        bank0
BSLP:   btfsc   portc,SDA
        goto   BSTP
        bsf    portc,SCL
        nop
        btfsc   portc,SDA
        goto   BSTP
        bcf    portc,SCL
        nop
        goto   BSLP
BSTP:   bcf    portc,SCL
        bcf    portc,SDA
        bank1
        bcf    trisc,SCL ; both driven
        bcf    trisc,SDA
        nop
        bank0
        bsf    portc,SCL
        nop
        bsf    portc,SDA ; SDA=hi = STOP CDX
        nop
        nop
        retlw 0
;*****;
; Test For Presence of FRAM Device ;
; variables used: ;
; sadr = slave adr of FRAM (usually $a0);
; w, tmp, tmp2 = temp regs ;
; if present, returns carry=0 ;
;*****;
i2ready:call    BSTART
        movf    sadr,w
        movwf   tmp
        clrf    tmp2
        bsf    tmp2,3 ; 8 bits of data
i2redy: rlf    tmp,f ; Shift data bit out.
        call    BITOUT ; Serial data out
        decfsz tmp2,f ; 8 bits done?
        goto    i2redy ; No.

```

```

        call    BITIN      ; Read acknowledge bit
        goto    BSTOP
;*****
; The following demo program will first verify that the FRAM
; is actually present, then it will clear the first 256 bytes
; of a FM24C04 then write h'a5' and h'5a' at location h'80'/h'81'.
; If no errors occur, then porta:7 is hi
; if unable to find the FRAM, porta:0 is hi
; if unable to clear the 256 bytes in FRAM, porta:1 is hi
; if unable to write to FRAM Loc $80-$81, porta:2 is hi
; if unable to read fr FRAM at Loc $80/$81, porta:3 is hi
; if the data read was incorrect, porta:4 is hi
; processing then halts (endless loop)
main:   call    BSTOP
        call    BSTART
        call    BSTOP      ; to initialize the i2c bus to idle
        movlw  h'a0'      ; slave adr for FM24C04
        movwf  sadr
        call   i2ready
        btfsc  status,c
        goto  maine0
        clrf  i2buf      ; clr 8 locs in i2c buffer
        clrf  i2buf+1
        clrf  i2buf+2
        clrf  i2buf+3
        clrf  i2buf+4
        clrf  i2buf+5
        clrf  i2buf+6
        clrf  i2buf+7
        movlw d'32'      ; 32 loops = 256 bytes
        movwf tmp5      ; loop counter
; the next instructions are redundant
;   movlw  h'a0'      ; slave adr for FM24C04
;   movwf  sadr
        clrf  i2adr      ; Internal FRAM adr start
        movlw i2buf      ; from this uP RAM adr
        movwf i2ptr      ; uP Orig Adr of Bytes to send
        movlw d'8'
        movwf i2cnt      ; #bytes per pass
main1:  call   i2wr      ; write 8 bytes
        movf  tmp3,w
        btfsc status,z
        goto  maine1    ; error if = 0
        movf  i2cnt,w   ; w=8
        addwf i2adr,f   ; add 8 to FRAM DEST adr
        addwf i2ptr,f   ;      "   uP   ORIG  "
        decfsz tmp5,f
        goto  main1
        movlw h'a5'
        movwf i2buf      ; set 2 locs in i2c buf
        movlw h'5a'
        movwf i2buf+1
        movlw h'80'
        movwf i2adr      ; Internal FRAM adr = $80
        movlw i2buf      ; from this uP RAM adr
        movwf i2ptr      ; uP Orig Adr of Bytes to send
        movlw d'2'
        movwf i2cnt      ; #bytes per pass
        call  i2wr      ; write 2 bytes
        movf  tmp3,w

```

```

        btfsc    status,z
        goto    maine2    ; error if tmp3 = 0
        clrf    i2buf     ; optional- clr 2 locs in i2buf,
        clrf    i2buf+1   ; make sure I didn't flimflam ya
; the following values never changed, so these are redundant
;
        movlw   h'80'
;
        movwf   i2adr     ; Internal FRAM adr = $80
;
        movlw   i2buf     ; from this uP RAM adr
;
        movwf   i2ptr     ; uP Orig Adr of Bytes to send
;
        movlw   d'2'
;
        movwf   i2cnt     ; #bytes per pass
        call    i2rd      ; read 2 bytes
        movf    tmp3,w
        btfsc   status,z
        goto    maine3    ; error if = 0
        movlw   h'a5'
        xorwf   i2buf,w   ; see if i2buf is $A5 (better be)
        btfss   status,z
        goto    maine4
        movlw   h'5a'
        xorwf   i2buf+1,w ; see if i2buf+1 is $5A (better be)
        btfss   status,z
        goto    maine4
; good exit: set portb,7 = 1 and halt.
        bsf     portb,7
main0:   clrwdt          ; endless loop
        goto    main0
; misc errors: portb will reflect where error occurred
maine0:  bsf     portb,0 ; couldn't find the FRAM
        goto    main0
maine1:  bsf     portb,1 ; write error, couldn't clear FRAM
        goto    main0
maine2:  bsf     portb,2 ; write error, couldn't write to loc $80/$81
        goto    main0
maine3:  bsf     portb,3 ; read error, couldn't read from FRAM Loc $80/$81
        goto    main0
maine4:  bsf     portb,4 ; The data I got back wasn't $a5/$5a
        goto    main0
;
        end
;
        list off
;

```

