



I²C™ Memory Autodetect

Author: *Lucio Di Jasio*
Microchip Technology, Italy

INTRODUCTION

This application note describes a method to automatically detect the memory size of a serial EEPROM connected to an I²C bus. The topics include:

- Automatic detection of memory size on the I²C bus
- Standard I²C
- Smart Serial or the I²C Dilemma
- Another set of routines for I²C
- How to tell the addressing scheme
- How to tell the size
- Putting it all together
- Debugging
- Compatibility
- References

AUTOMATIC DETECTION OF MEMORY SIZE ON THE I²C BUS

The purpose of this application note is to show how to solve a common problem in microcontroller applications with Serial EEPROMs. User needs often dictate different memory sizes for different versions of an application, but cost constraints require the smallest possible memory to be used each time. A typical application example could be the base station (receiver) of a remotely controlled garage door opener. Versions capable of storing 4, 20, 200 or 1000 users could be implemented from a single source code complementing the controller with the appropriate memories.

Microchip currently offers a very broad range of memory capacities with I²C bus interface (from 16 bytes in the 24C00 up to 32k bytes in the 24C256).

The microcontroller has to be able to tell which memory it is dealing with on the I²C bus in order to address it properly.

There are two possible approaches to the problem, one is to provide some kind of configuration information to the controller by means of dip switches or jumpers, the other one is to make the controller capable of automatic

detection. In this application note, we will show how to implement the automatic detection in an easy, safe and compatible way.

The software techniques explained in the following will be demonstrated on a generic mid-range PICmicro[®] microcontroller (MCU), PIC16C62A and can be tested immediately using a PICDEM2 demo board.

All the code can be adapted to any other PICmicro MCU (12, 14 and 16 bit core) and/or pin configuration with minor modifications to the source code.

Standard I²C

The I²C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller that controls the bus, generates the serial clock (SCL) and originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = 1010) is shown in Figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the 1010 identifying address plus the three block or chip select bits A2,A1,A0.

Smart Serial or the I²C Dilemma [ref 3]

The I²C serial bus has many advantages over other common serial interfaces for serial embedded devices. The I²C bus with level-triggered inputs offers better noise immunity over edge-triggered technology. Opcodes are not needed to communicate with storage devices because all interfaces are intuitive and comparable to parallel devices.

But the standard protocol limits addressing up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2 (8x2kbytes).

Herein lies the dilemma. With the advent of the more sophisticated personal communication devices such as cellular and full-featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

So the Smart Serial concept grew from the industry's need for increased memory requirements in I²C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands.

Microchip Technology has designed an addressing scheme for I²C Serial EEPROM based on the standard I²C protocol and device addresses, but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective design.

For the first byte, or control byte, the Smart Serials adhere to the I²C protocol (reference Figure 2). The next 2 bytes (instead of one) define the address of the requested memory location.

Another Set of Routines for I²C bus

Many application notes have already been published by Microchip Technology on the I²C bus interface such as: AN515, AN537, AN558, AN567, AN608, AN554, AN578 and AN535. In the following, we will use techniques and code taken from those application notes as a base to build a new compact, powerful set of routines. The first step will be to modify a basic set of routines [ref1,2,4,6,8] to make them capable of producing Standard I²C and Smart Serial addressing, selecting the addressing scheme at run time by means of a flag (that we will call: SMART).

Listing 1 (*i2c.inc*) shows the new set of routines. As usual, there are two layers of functions:

- The lower layer (composed of routines: *BSTOP*, *BSTART*, *RXI2C*, *TXI2C*, *BITIN*, *BITOUT*, *ERR*; listing starts from line 153) deals with sending and detecting the single bits and bytes on the bus and contains no new code.
- The higher layer (composed of routines: *RDbyte*, *WRbyte* and *SETI2C*, from line 1 to 152) assembles commands and takes care of addressing schemes. This will be the focus of our discussion.

What is new here, is that we moved to function *SETI2C* (lines 112..152) all the code that deals with the details of the addressing scheme. This function gets a SMART flag as an input and provides Standard or Smart addressing according to its value. Both *RDbyte* and *WRbyte* rely on *SETI2C* for the command and address generation, and therefore are now compatible with Standard and Smart Serial.

Determining the Addressing Scheme

As a next small step toward automatic memory size detection we need to find a method to distinguish automatically between a Smart Serial and a Standard Serial EEPROM.

The algorithm proposed is very simple and compact, made up of only the following 4 steps:

1. Put in Smart Serial mode the I²C routines (set SMART flag).
2. Issue a write command to location 0000, writing a 1.

Note: If the memory is a standard I²C, this command is interpreted as a sequential write command of two bytes that produces writing a 00 byte to location 0000 and a 01 byte to location 0001.

(0000) <- 00

(0001) <- 01

If the memory is a Smart Serial, then we get the correct interpretation.

(0000) <- 01

3. Put in Standard I²C Mode the I²C routines (clear the SMART flag).
4. Issue a read command of location 0000.

If the memory really is a Standard I²C, then this read command will give us the contents of location 0000, and that was set to 0!

If the memory is a Smart Serial, we get a read command with a partial (incomplete) addressing.

What happens in this case is not really part of the I²C bus definition, so let's analyze two possible cases.

- a) Partial addressing set only the most significant bits of the internal address register and leaves unattached the lower 8 bits. This means that we will read location 0000.
- b) Partial addressing doesn't modify at all the address register. This means that the address remains equal to the last value set (by the last Smart Write) and reading gives the contents of location 0000.

If in both cases we end up reading a 1, that tells us that it was a Smart Serial memory. If a 0 was read, then it was a Standard I²C serial memory.

Listing 2 (*i2cauto.asm*) lines 108..120 implement in just 10 lines of assembly this simple algorithm.

Note: Locations 0000 and 0001 are obviously corrupted through this procedure and there is no way to save and restore them (until the addressing scheme is known!).

Determining Memory Size

The last step toward automatic memory size detection is the development of an algorithm to tell the size of a memory given its addressing scheme. That is, suppose we know whether it is a Standard or Smart, we want to be able to measure its size.

We will base the detection algorithm on a simple assumption which is:

If a memory is of size N, then trying to address locations out of the 0..N-1 range will produce a fall back in the same range (modulus N). Since the most significant (extra) address bits will be simply ignored, they are DON'T CARE bits to the device as can be easily verified from each device data sheet.

We can develop a simple test function to tell us whether a memory is of a given size N (or smaller).

In a high level pseudo language, such a test function could look like this:

EXAMPLE 1:

```
function TestIfSizeIs(Size N): boolean
( // is memory range 0..N-1 ?
  var TEMP;
  TEMP = Read( 0000);

  if ( Read( N) == TEMP)
    Write( 0000, TEMP+1)

    if ( Read( N) == TEMP+1)
      Write( 0,TEMP-1)
      return( TRUE)
  // else
  return( FALSE)
) //end function
```

Having this function, we can then set up a loop to test memory sizes.

In the case of the Standard I²C, we can loop and test from N=128 to N=2048 corresponding to models from 24C01 up to 24C16 doubling N at each iteration as in the following:

EXAMPLE 2:

```
function StandardI2CMemDetect() : integer
( // returns a model number 1..16

  N = 128
  MODEL = 1
  loop
    if (TestIfSizeIs( N))
      break
    else
      N=N*2
      MODEL=MODEL*2
  while(N<=2048)

return ( MODEL);
) //end function
```

Similarly, a function to measure Smart Serial memories will loop with N=4096 up to N=32768.

Please note that in this second algorithm, no memory location had to be reserved. Even location 0 that is modified could always be saved and restored by the test algorithms.

PUTTING IT ALL TOGETHER

Now all the pieces of the puzzle are ready and we can complete our automatic memory size detection routine. First we determine the addressing scheme, and once that is known, we enter a loop to measure the actual memory size. Depending on the addressing scheme, we will enter the loop with different initial values corresponding to the different ranges of memory according to the memory models available on the market.

Listing 2 (*i2cauto.asm*) lines 136..174 implement in assembly in a very compact way both algorithms.

Debugging

Assembling the code and testing it on a PIC16C62A on a PICDEM2 board or any other target board (after modifying the pin definitions in listing 2 (*i2cauto.asm*) lines 48..60) will prove the functionality of the proposed code. Just insert an I²C memory in the DIL socket on the PICDEM2 board, power up or press the reset button, and voila', on the LEDs will appear the binary representation of the memory TYPE value according to [Table 1](#).

TABLE 1 MEMORY TYPE VALUE

Standard I ² C			Smart Serial		
Type	Size	Model	Type	Size	Model
01	128	24C01/21/41	32	4096	24C32
02	256	24C02/62	64	8192	24C65/64
04	512	24C04	128	16384	24C128
08	1024	24C08	0	32768	24C256
16	2048	24C16/164			

The reader is invited to experiment and modify further this software to adapt it to their specific needs. When doing so, we strongly recommend having at hand the SEEVAL kit, a cheap and effective tool from Microchip Technology that allows the designer to read/write any Serial EEPROM and connects to any PC through the serial port. Further consider the "Endurance" software tool from Microchip Technology, while designing memory applications where reliability and endurance are critical.[ref 9,10]

Compatibility

While most of the code presented strictly follows the existing I²C and Smart Serial standards, it should be compatible with any Serial EEPROM device from any manufacturer, that adheres to such standards. Only Microchip Serial EEPROMs were tested. It is left up to the user to validate this code for Serial E² from other manufacturers.

Further, there is some space for discussion, as a possible future compatibility issue, on the addressing scheme detection method. As a matter of fact, the behavior of the serial memory in case of partial addressing (as it occurs during step 4 in the case of Smart Serial) is not part of the specification. While it works with current implementations of the Smart Serial protocol (from Microchip and up to the 24C256), it is not guaranteed to do so in the future.

References

- [1] AN515 Communicating with I²C™ Bus Using the PIC16C5X, Bruce Negley
- [2] AN535 Logic Powered Serial EEPROMs, R. J. Fisher and Bruce Negley
- [3] AN558 Using the 24xx65 and the 24xx32 with Stand-alone PIC16C54 Code, Dick Fisher and Bruce Negley
- [4] AN567 Interfacing the 24LCxxB Serial EEPROMs to the PIC16C54, Bruce Negley
- [5] AN608 Converting to 24LCXXB and 93LCxx Serial EEPROMs, Nathan John
- [6] AN536 Basic Serial EEPROM Operation, Steve Drehobl
- [7] AN554 Software Implementation of I²C™ Bus Master, Amar Palacherla
- [8] AN559 Optimizing Serial Bus Operations with Proper Write Cycle Times, Lenny French
- [9] AN537 Serial EEPROM Endurance, Steve Drehobl
- [10] AN602 How to get 10 Million Cycles Out of Your Microchip Serial EEPROM, David Wilkie

FIGURE 1: CONTROL BYTE ALLOCATION

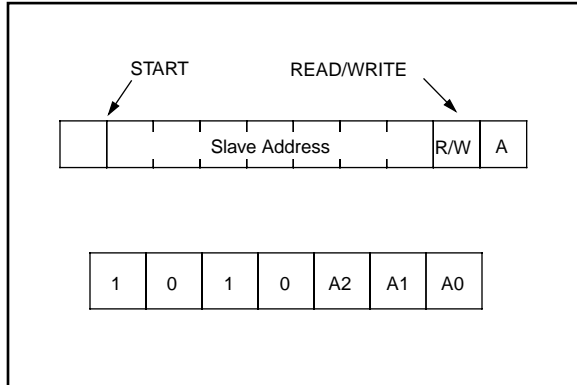
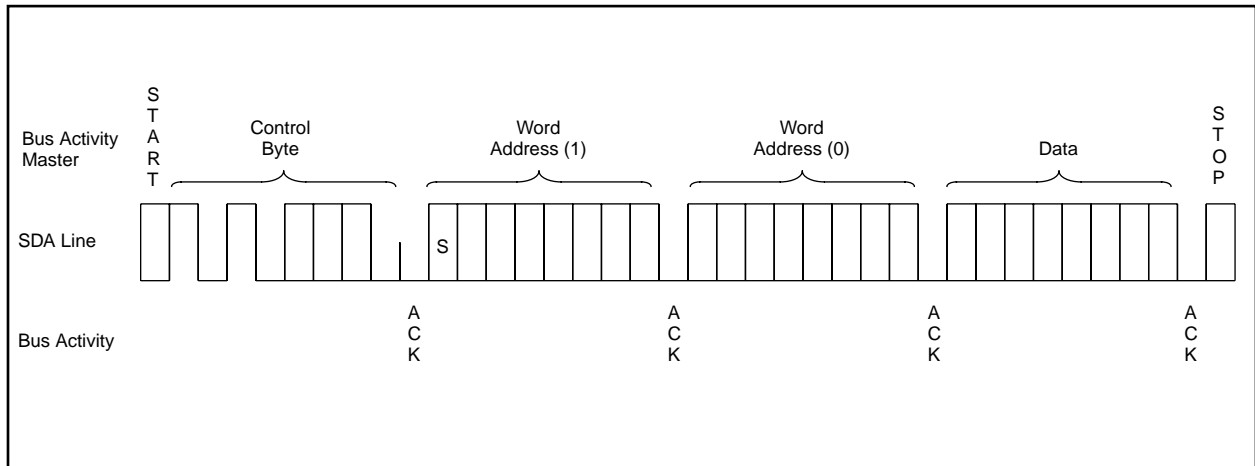


FIGURE 2: BYTE WRITE



APPENDIX A:

LISTING 1: I2C.INC

```
*****
;*  Filename:   I2C.INC
;*  Author:    Lucio Di Jasio
;*  Company:   Microchip Technology
;*  Revision:  RevA0
;*  Date:      5-7-98
;*  Assembled using MPASM v02.15
*****
;*  Two wire/I2C Bus READ/WRITE Sample Routines
;*  both Smart Serial and Standard I2C addressing schemes supported
;*  PIC16CXXX mid-range (14 bit core) version
;*
;*  Note:  1) All timing is based on a reference crystal frequency of 4MHz
;*         which is equivalent to an instruction cycle time of 1 usec.
;*        2) Address and literal values are read in hexadecimal unless
;*         otherwise specified.
*****
;*
;*      Register File Assignment
*****
CBLOCK
    FLAGS
    INDHI      ; address
    INDLO
    DATO       ; data buffer for read write functions
    ERCODE     ; error code (see table below)
    EEBUF      ; read write buffer
    SLAVEbuf   ; SLAVE address (+ addrHi on 24LC16)
    COUNT
    AUX
ENDC
*****
; flag definitions
;
#define FLAG_EE    FLAGS,0      ; I2C bus error
#define SMART     FLAGS,1      ; Smart(1) Standard(0)
;
*****
;*
;*      Bit Assignments
*****
#define SLAVE     B'10100000' ; Device address (1010xxx0)

; error codes
#define ERR_NACK   1           ; no ACK reading
#define ERR_STOP  2           ; SDA locked in STOP
#define ERR_TOWR  3           ; time out in read (>20ms)
#define ERR_LOCK  4           ; SDA locked in BITOUT

*****
;*
;*      RDbyte
;*      read one byte from serial EEPROM device
;*
;*      Input   :      INDHI/LO
;*              SLAVE = device address (1010xxx0)
;*      Output  :      DATO   = data read from serial EEPROM
*****
;
RDbyte bcf     FLAG_EE      ; reset error flag
       call   SETI2C       ; set address pointer

; enter here for sequential reading
```

```

RDnext call    BSTART          ; START
      movf    SLAVEbuf,W      ; use SLAVE addr(+IndHi se 24LC16)
      movwf   EEBUF
      bsf     EEBUF,0         ; it's a read command
      call    TXI2C           ; Output SLAVE + address + read command
      call    RXI2C           ; read in DATO and ACKnowledge
      movf    EEBUF,W
      movwf   DATO

      bsf     STATUS,C        ; set ACK = 1 (NOT ACK)
      call    BITOUT          ; to STOP further input
      goto   BSTOP           ; generate STOP bit

;*****
;*      WRbyte
;*      write one byte to EEPROM device
;*
;*      Input   :      DATO    = data to be written
;*              INDHI/LO= EEPROM data address
;*              SLAVE    = device address (1010xxx0)
;*              PROT     = 1-> SmartSerial | 0> Standard
;*      Output  :      FLAG_EE = set if operation failed
;*****

WRbyte bcf     FLAG_EE        ; reset error condition
      call    SETI2C         ; set address pointer
      movf    DATO,W         ; move DATO
      movwf   EEBUF         ; into buffer
      call    TXI2C         ; output DATO and detect ACKnowledge
      call    BSTOP         ; generate STOP bit

; loop waiting for writing complete
      movlw   .80            ; 80 test=20ms timeout
      movwf   AUX
WRpoll CLRWDT                ; keep the WDT from resetting
      bcf     FLAG_EE
      call    BSTART         ; invia start
      movlw   SLAVE
      movwf   EEBUF
      call    TXI2C         ; ed un comando di scrittura
      btfs   FLAG_EE        ; se non da ACK -> ercode 3 -> BUSY
      goto   WRpolle
WRbusy decfsz  AUX,F
      goto   WRpoll
      movlw   ERR_TOWR      ; time out in scrittura
      call    ERR
WRpolle goto   BSTOP         ; exit sending the stop condition

;*****
;*      SETI2C
;*      set the address pointer at INDHI/LO, use Smart or Standard
;*      addressing scheme according to SMART flag
;*
;*      Input   :      INDHI   = EEPROM data address
;*              INDLO
;*              SLAVE    = device address (1010xxx0)
;*              SMART   = 1-> Smart Serial | 0> Standard I2C
;*      Output  :      SLAVEbuf for sequential read
;*****
SETI2C
      btfs   SMART          ; if clear -> Standard I2C
      goto   Smart         ; if set -> Smart Serial

Standard
      bcf     STATUS,C      ;

```

AN690

```
    rlf     INDHI,W           ; add address MSb
    iorlw   SLAVE            ; to slave address
    movwf   EEBUF
    movwf   SLAVEbuf        ; save for sequential read
    call    BSTART          ; generate START bit
    call    TXI2C           ; output first comand byte
    goto    SETseq

Smart
    movlw   SLAVE            ; prepare slave address
    movwf   EEBUF
    movwf   SLAVEbuf        ; save for sequential read
    call    BSTART          ; generate START bit
    call    TXI2C           ; output first command byte
    movf    INDHI,W         ;
    movwf   EEBUF          ; output address MSB
    call    TXI2C

SETseq
    movf    INDLO,W         ; send address LSB
    movwf   EEBUF
    goto    TXI2C           ; Output WORD address

;*****
;*      TXI2C
;*      transmit 8 data bits
;*
;*      Input   :      EEBUF
;*      Output  :      none
;*****
TXI2C
    movlw   .8              ; Set counter for eight bits
    movwf   COUNT

TXlp
    rlf     EEBUF,F         ; data bit in CARRY
    call    BITOUT          ; Send bit
    decfsz COUNT,F         ; 8 bits done?
    goto    TXlp           ; No.

    call    BITIN           ; Read acknowledge bit
    movlw   ERR_NACK
    btfsc  STATUS,C        ; Check for acknowledgement
    call    ERR             ; No acknowledge from device
    return

;*****
;*      BITOUT
;*      send single bit
;*
;*      Input   :      bit in CARRY
;*      Output  :      Bit transmitted over I2C
;*      Error bits set as necessary
;*****
BITOUT
    btfss  STATUS,C        ; is it 0/1?
    goto   Bit0

Bit1
    bsf    STATUS,RP0      ; select RAM bank 1
    bsf    SDA             ; input SDA (pull up->1)
    bcf    STATUS,RP0      ; back to RAM bank 0
    movlw  ERR_LOCK
    btfss  SDA             ; Check for error
    call   ERR             ; SDA locked low by device
    goto   Clk1
```



```

Bit0
    bsf    STATUS,RP0    ; select RAM bank 1
    bcf    SDA           ; Output SDA
    bcf    STATUS,RP0    ; back to RAM bank 0
    bcf    SDA           ; clear 0
    nop                    ; Delay

Clk1
    bsf    SCL           ; rise SCL
    nop
    nop
    nop                    ; Timing delay 4us minimum
    nop
    nop
    bcf    SCL           ; lower SCL
    return

;
;*****
;*    RXI2C
;*    receive eight data bits
;*
;*    Input    :    None
;*    Output   :    RXBUF = 8-bit data received
;*****
RXI2C
    movlw  .8            ; 8 bits of data
    movwf  COUNT
    clrf   EEBUF

RXlp
    call   BITIN         ; new bit in CARRY
    rlf   EEBUF,F        ; enter new bit
    decfsz COUNT,F      ; 8 bits?
    goto  RXlp
    return

;
;*****
;*    BITIN
;*    Single bit receive
;*
;*    Input    :    None
;*    Output   :    EEBUF,0 bit received
;*****
BITIN
    bsf    STATUS,RP0    ; select RAM bank 1
    bsf    SDA           ; Set SDA for input
    bcf    STATUS,RP0    ; back to RAM bank 0
    bsf    SCL           ; Clock high
    nop
    nop
    nop
    nop                    ; provide minimum Tset up
    CLRC
    btfsc SDA           ; Read SDA pin in CARRY
    bsf    STATUS,C
    bcf    SCL           ; Return SCL to low
    return

;*****
;*    START bit generation
;*
;*    input    : none
;*    output   : initialize bus communication

```

AN690

```
*****
BSTART
    bsf    STATUS,RP0      ; select RAM bank 1
    bsf    SDA             ; SDA input (pull-up ->1)
    bcf    STATUS,RP0      ; back to RAM bank 0
    bsf    SCL             ; Set clock high
    nop
    nop
    nop
    nop                    ; 5us before falling SDA
    bsf    STATUS,RP0      ; select RAM bank 1
    bcf    SDA             ; SDA output
    bcf    STATUS,RP0      ; back to RAM bank 0
    bcf    SDA             ; set SDA = 0
    nop
    nop
    nop
    nop                    ; 4us before falling SCL
    bcf    SCL             ; Start clock train
    return

*****
;*      STOP bit generation
;*
;*      Input   :      None
;*      Output  :      Bus communication, STOP condition
*****
BSTOP
    bsf    STATUS,RP0      ; select RAM bank 1
    bcf    SDA             ; SDA output
    bcf    STATUS,RP0      ; back to RAM bank 0
    bcf    SDA             ; set SDA = 0
    bsf    SCL             ; Set SCL high
    nop
    nop
    nop
    nop                    ; 4us before rising SDA
    bsf    STATUS,RP0      ; select RAM bank 1
    bsf    SDA             ; SDA input (pull-up ->1) while SCL high
    bcf    STATUS,RP0      ; back to RAM bank 0
    movlw  ERR_STOP        ; Ready error code
    btfss  SDA             ; High?
    call   ERR             ; Error, SDA locked before STOP

    bcf    SCL             ; lower SCL
    return

;
*****
;*      Two wire/I2C - CPU communication error status table
;*
;*      input   :      W-reg  = error code
;*      output  :      ERCODE = error code
;*              FLAG(ERROR) = 1
*****
ERR
    bcf    STATUS,RP0      ; back to RAM bank 0
; record last error
    movwf  ERCODE         ; Save error code
    bsf    FLAG_EE        ; Set error flag
    return
```

LISTING 2: I2CAUTO.ASM

```

LIST      n=0, c=132
RADIX    HEX
PROCESSORPIC16C62A
;*****
;*  Filename:   I2CAUTO.ASM
;*****
;*  Author:    Lucio Di Jasio
;*  Company:   Microchip Technology
;*  Revision:  RevA0
;*  Date:      5-7-98
;*  Assembled using MPASM v02.15
;*****
;*  Include files:
;*    p16c62A.inc rev1.01
;*
;*****
;*  software detection of I2C memory size
;*
;*    PIC16CXXX          /+5V
;*  +-----+          |
;*  |          Vdd+-----+          24CXXX
;*  |          |          +++          |          +-----+
;*  |          |          ||          |          +--+Vdd
;*  |          |          ||          |          |          |
;*  |          |          +++          |          |          |
;*  |          RC4+-----+          +SDA
;*  |          RC3+-----+          +SCL
;*  |          |          |          |
;*  |          Vss+-----+          +Vss
;*  +-----+          |          +-----+
;*                   |          |
;*                   GND
;*
;*  can be tested on a PICDEM2 demo board
;*****

INCLUDE    "P16C62A.INC"

__CONFIG   _XT_OSC & _CP_OFF & _WDT_ON
__IDLOCS   H'62A0'

;*****
;*  external 4MHZ crystal oscillator
;*  no code protection
;*  no watchdog
;*  ID code is "62A0"
;*****

;  pin assignments

#define     SDA      PORTC,4      ; i I2C SDA
#define     SCL      PORTC,3      ; o I2C SCL

MASKA      equ      0FF          ; unused all inputs

MASKB      equ      00          ; all outputs to LEDs

MASKC      equ      b'11110111' ; SCL and SDA on this port
; enable SCL as output
;
;-----
;  RAM assignments
;
CBLOCK     20
TEMP
SIZELO     ; memory size

```

AN690

```
        SIZEHI
        TYPE      ; memory type
    ENDC

;*****

        org      00          ; reset vector

        goto     Start

;*****

        org      04          ; interrupt vector

        retfie    ; esce riabilitando gli interrupt

;*****

        INCLUDE "i2c.inc"

;*****
;* MemDetect,
;* automatic detection of memory size
;*
;* INPUT:
;* none
;* OUTPUT:
;* SIZEHI/LO memory size as detected
;* TYPE      memory type (see table below)
;* FLAG_EE   bus error flag
;* ERCODE    bus error code
;*
;*          Standard I2C                Smart Serial
;* TYPE  SIZE  MODEL          TYPE  SIZE  MODEL
;* 01   128   24C01/21/41     32   4096  24C32
;* 02   256   24C02/62       64   8192  24C65/64
;* 04   512   24C04           128 - 16384  24C128
;* 08  1024   24C08           0 - 32768  24C256
;* 16  2048   24C16/164
;*
;*****
MemDetect
    clrf    INDHI        ; address 0000h
    clrf    INDLO
    bsf     SMART        ; write(smart, 0000, 1)
    movlw   1
    movwf  DATO
    call   WRbyte
    bcf    SMART
    call   RDbyte        ; read(standard, 0000)
    movf   DATO,W
    btfsc  STATUS,Z
    goto   StandardD

SmartD
    bsf     SMART        ; it is a Smart Serial
    movlw  HIGH(.4096)
    movwf  SIZEHI        ; size = 4096 byte
    clrf   SIZELO
    movlw  .32
    movwf  TYPE          ; start with TYPE = 24C32
    goto   TestD

StandardD
    bcf     SMART        ; it is a Standard Serial
    movlw  .128
```

```

        movwf  SIZELO      ; size = 128 byte
        clrf   SIZEHI
        movlw  01
        movwf  TYPE       ; start with TYPE = 24C01

TestD
        call   RDbyte     ; TEMP=read(0)
        movf   DATO,W
        movwf  TEMP

LoopDet
        movf   SIZELO,W   ; DATO=read(SMART, size)
        movwf  INDLO
        movf   SIZEHI,W
        movwf  INDHI
        call   RDbyte
        movf   DATO,W
        xorwf  TEMP,W     ; compare TEMP with DATO
        btfss STATUS,Z
        goto   LoopDN
        incf   TEMP,W     ; if same value than TEMP=TEMP+1
        movwf  TEMP
        movwf  DATO
        clrf   INDHI
        clrf   INDLO
        call   WRbyte     ; write(SMART, 0000, TEMP)
        movf   SIZELO,W   ; if (read(SMART, size) == TEMP)
        movwf  INDLO
        movf   SIZEHI,W
        movwf  INDHI
        call   RDbyte
        movf   DATO,w     ; if still same value it means
        xorwf  TEMP,W     ; we reached the actual memory size
        btfsc STATUS,Z
        goto   DetEx

LoopDN
        bcf   STATUS,C    ; double memory size
        rlf   SIZELO,F
        rlf   SIZEHI,F
        bcf   STATUS,C
        rlf   TYPE,F     ; double TYPE code
        btfss TYPE,4
        goto   LoopDet   ;

DetEx
        nop
        return

;*****
; init ports and option register
;
Start
        bsf   STATUS,RP0  ; select RAM bank 1
        movlw MASKA      ; set tris registers
        movwf PORTA     ; PORTA
        movlw MASKB      ;
        movwf PORTB     ; PORTB
        movlw MASKC      ;
        movwf PORTC     ; PORTC

        movlw b'00000111' ; enable pull_ups, prescale TMR0 1:256
        movwf OPTION_REG

        bcf   STATUS,RP0
        clrf  FLAGS     ; reset all flags

```

AN690

```
;*****  
;  
  
Main  
    call    MemDetect      ; determine memory size  
  
    movf   TYPE,W         ; if using a PICDEM2 board  
    movwf  PORTB         ; send TYPE to the LEDs  
  
MainLoop  
    goto   MainLoop      ; stuck in the loop until reset  
  
    END
```

NOTES:

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: <http://www.microchip.com>

Rocky Mountain

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta

500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York

150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia

Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing

Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu

Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou

Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai

Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen

Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong

Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India

Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan

Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark

Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France

Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02