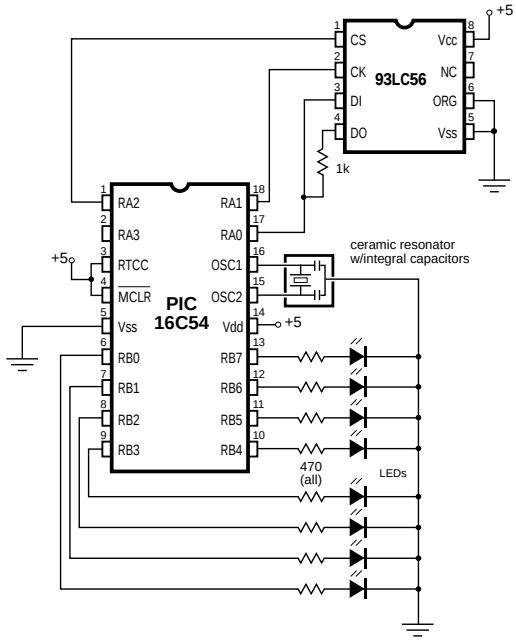
Using Serial EEPROMs

Introduction. This application note shows how to use the Microchip 93LC56 EEPROM to provide 256 bytes of nonvolatile storage. It provides a tool kit of subroutines for reading, writing, and erasing the EEPROM. (Note that EEPROMs made by other manufacturers will not work with the PIC16Cxx.)

Background. Many designs take advantage of the PIC's ability to store tables of data in its EPROM program memory. The trouble is that the larger the tables are, the smaller the space left for code. And many applications could benefit from the ability to occasionally update data tables for calibration or other purposes. What the PIC needs is the equivalent of a tiny disk drive.



The Microchip 93C56 and 93LC56 electrically erasable PROMs (EEPROMs) are perfect for these applications. They communicate serially via a three- or four-wire bus using a simple synchronous (clocked) communication protocol at rates of up to 2 million bits per

second (Mbps). It's possible to read a byte in as little as 10 microseconds (including the time required to send the instruction opcode and address). Once a program has begun reading data from the EEPROM, it can continue reading subsequent bytes without stopping. These are clocked in at the full 2 Mbps rate; 1 byte every 4 microseconds.

Erasing and writing these serial EEPROMs happens at a more leisurely pace. While the opcode, address, and data can be clocked into the chip at high speed, the EEPROM requires about 2 milliseconds to erase or write a byte. During this time, the chip cannot process additional instructions. The PIC can poll a flag to determine when the automatic erase/write programming cycle is over. As soon as this flag goes high, the EEPROM is ready for more instructions.

Data stored in the EEPROM will be retained for 10 years or more, according to the manufacturer. The factor that determines the EEPROM's longevity in a particular application is the number of erase/write cycles. Depending on factors such as temperature and supply voltage, the EEPROM is good for 10,000 to 1 million erase/write cycles. This rules out its use as a substitute for ordinary RAM, since many PIC applications write to RAM thousands of times a second. At that rate, the EEPROM could be unusable in as little as 20 seconds! For a thorough discussion of EEPROM endurance, see the Microchip Embedded Control Handbook, publication number DS00092A, October 1992.

How it works. The circuit in the figure specifies a 93LC56 EEPROM, but a 93C56 will work as well. The difference is that the LC device has a wider Vcc range (2.5–5.5 V, versus 4–5.5 V), lower current consumption (3 mA versus 4 mA), and can be somewhat slower in completing internal erase/write operations, presumably at lower supply voltages. In general, the LC type is less expensive, and a better match for the operating characteristics of the PIC.

The schematic shows the data in and data out (DI, DO) lines of the EEPROM connected together to a single PIC I/O pin. The 1k resistor prevents the PIC and DO from fighting over the bus during a read operation. During a read, the PIC sends an opcode and an address to the EEPROM. As soon as it has received the address, the EEPROM activates DO and puts a 0 on it. If the last bit of the address is a 1, the PIC

could end up sourcing current to ground through the EEPROM. The resistor limits the current to a reasonable level.

The program listing is a collection of subroutines for reading, writing, and erasing the EEPROM. All of these rely on *Shout*, a routine that shifts bits out to the EEPROM. To perform an EEPROM operation, the software loads the number of clock cycles into *clocks* and the data to be output into *temp*. It then calls *Shout*, which does the rest.

If you don't have the EEPROM data handy (Microchip Data Book, DS00018D, 1991), you should know about a couple of subtleties. First, when the EEPROM powers up, it is write protected. You must call *EEnable* before trying to write or erase it. It's a good idea to call *EEdisbl* (disable writes) as soon as possible after you're done. Otherwise, a power glitch could alter the contents of your EEPROM. Also, you cannot write all locations (*EEWrall*) without first erasing all locations (*EEwipe*).

Modifications. The table of constants at the beginning of the listing specifies the opcodes for each of the EEPROM operations. Although the opcodes are only three bits long, they are combined with a trailing don't-care bit. This bit is required for compatibility with the 512-byte 93LC66. With the '66, this would be address bit A8. If you want to modify this code for the '66, add a line to the read, write, and byte erase routines to copy A8 into bit 4 of *temp* just before calling *Shout*. If you want to increase capacity by adding more EEPROMs, you can bus the data and clock lines together and provide separate chip selects to each device.

If you plan to run your PIC faster than 8 MHz, add one or two *nop*s where marked in the listing. The clock must be high for at least 500 nanoseconds. The low time must also be greater than 500 ns, but the move-data, rotate, and looping instructions provide enough delay.

Program listing. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

```
; PROGRAM: EEPROM.SRC
; This program is a collection of subroutines for reading, writing, and erasing the
; 93LC56 (or 'C56) serial EEPROM. As a demonstration, it writes a scanning pattern
; to the 256 bytes of the EEPROM, and then reads it back to eight LEDs connected
; to port rb.

; Remember to change device info when programming part
                device      pic16c54,xt_osc,wdt_off,protect_off
                reset       start

D               =           ra.0            ; Pins DI and DO of the EEPROM
CLK             =           ra.1            ; Clock pin--data valid on rising
                                            ; edge
CS              =           ra.2            ; Chip select--high = active
ROP             =           192             ; Opcode for read
WROP            =           160             ; Opcode for write
EWEN            =           152             ; Opcode to enable erase and
                                            ; write
EWDS            =           128             ; Opcode to disable erase and
                                            ; write
ERAS            =           224             ; Opcode to erase a byte
ERAL            =           144             ; Opcode to erase all bytes
WRAL            =           136             ; Opcode to write all bytes with
                                            ; specified data

                org         8
temp            ds          1               ; Temporary variable for EE
                                            ; routines
EEdata          ds          1               ; Passes data to EEwrite/wrall,
                                            ; from EEread
EEaddr          ds          1               ; Passes address to EErase,
                                            ; EEwrite, EEread
clocks          ds          1               ; Number of clock cycles for
                                            ; SHiftOUT
tick1           ds          1               ; Timer for Delay--not required for
                                            ; EE routines
tick2           ds          1               ; Timer for Delay--not required for
                                            ; EE routines
                org         0
start           mov         ra,#0           ; Clear ports
                mov         rb,#0
                mov         !ra,#0          ; Make all ra pins outputs initially
                mov         !rb,#0
                call        EEnable         ; Turn off write/erase protection
                mov         EEdata,#1
                mov         EEaddr,#0

:loop           call        EEwrite         ; Write scanning pattern to
                                            ; EEPROM
```

```
                call    Busy
                rr      EEdata
                ijnz    EEaddr,:loop
                call    EEdisbl         ; Turn write/erase protection back
                                        ; on
                mov     EEaddr,#0
:loop2          call    EEread
                mov     rb,EEdata
                inc     EEaddr
                call    delay
                goto    :loop2

; Shift out the bits of temp to the EEPROM data line.
Shout           rl      temp            ; Rotate bit7 of temp into carry
                movb    D,c             ; Move carry bit to input of
                                        ; EEPROM
                setb    CLK             ; Clock the bit into EEPROM
                nop                     ; Clock must be high > 500 ns
                clrb    CLK
                djnz    clocks,Shout
                ret

; Read the byte in EEaddr into EEdata.
EEread          mov     temp,#ROP       ; Move the read opcode into temp

                mov     clocks,#4       ; Number of bits to shift out (op+1)
                setb    CS
                call    Shout
                mov     clocks,#8
                mov     temp,EEaddr
                call    Shout
                mov     !ra,#1
                mov     clocks,#8
:read           setb    CLK
                movb    c,D
                rl      temp
                clrb    CLK
                djnz    clocks,:read
                mov     EEdata,temp
                mov     !ra,#0
                clrb    CS
                ret

; Call to unprotect after EEdisbl or power up.
EEnable         setb    CS
                mov     clocks,#12
                mov     temp,#EWEN
                call    Shout
                clrb    CS
```

```
                ret

; Call to protect against accidental write/erasure.
EEdisbl         setb    CS
                mov     clocks,#12
                mov     temp,#EWDS
                call    Shout
                clrb    CS
                ret

; Write the byte in EEdata to EEaddr.
EEwrite mov     temp,#WROP
                mov     clocks,#4
                setb    CS
                call    Shout
                mov     clocks,#8
                mov     temp,EEaddr
                call    Shout
                mov     clocks,#8
                mov     temp,EEdata
                call    Shout
                clrb    CS
                ret

; Erase the byte in EEaddr. Erasure leaves FFh (all 1s) in the byte.
EErase          mov     temp,#ERAS
                mov     clocks,#4
                setb    CS
                call    Shout
                mov     clocks,#8
                mov     temp,EEaddr
                call    Shout
                ret

; Erase the entire EEPROM--all 256 bytes. Call before using EEwrall below.
EEwipe          setb    CS
                mov     temp,#ERAL
                mov     clocks,#12
                call    Shout
                clrb    CS
                ret

; Write the byte in EEdata to every address. Call EEwipe first.
EEwrall         setb    CS
                mov     temp,#WRAL
                mov     clocks,#12
                call    Shout
                mov     clocks,#8
                mov     temp,EEdata
```

```
                call    Shout
                clrb    CS
                ret

; Check flag to determine whether the EEPROM has finished its self-timed erase/
; write. Caution: This will lock up your program until D goes high.
Busy            nop
                mov     !ra,#1
                setb    CS
:wait           jnb     D,:wait
                clrb    CS
                mov     !ra,#0
                ret

Delay           djnz    tick1,Delay         ; Delay routine for demo.
                djnz    tick2,Delay
                ret
```