Programming a 24LC00 EEPROM using the EZ-USB I²C Port

Introduction

The Cypress EZ-USB family contains a general-purpose I²C interface through which the 8051 accesses standard I²C devices connected to the SCL and SDA pins. In normal operation, a small EEPROM connected to the I²C port supplies vendor-specific ID information to the operating system to enable it to load the proper software driver associated with the device. These ID tags, called Vendor ID (VID), Product ID (PID), and Device ID (DID), occupy the first seven bytes of the EEPROM, as shown in *Table 1* below.

Table 1. Contents of 16-byte 24LC00 EEPROM

EEPROM Address	Value
0	0xB0
1	VID(L)
2	VID(H)
3	PID(L)
4	PID(H)
5	DID(L)
6	DID(H)
7	available

The EZ-USB chip uses the constant 0xB0 in the first byte to identify the subsequent bytes as VID/PID/DID information.

An excellent choice for the EEPROM is the Microchip 24LC00-OT, a 16-byte serial EEPROM in a 5-lead SOT-23 package. This device is similar to the widely available 24 series of EEPROMS such as the 24LC01 (128 bytes) and the 24LC02. The “LC” designation denotes low-voltage operation (2.5–6.0 volts) making the part ideal for a 3.3V USB peripheral. At this writing the Microchip 24LC00 is the only version available in the very small SOT-23 package.

The Microchip 24LC00 is used on the Cypress EZ-USB Development Board. It is provided in an 8-pin socketed DIP package so that other EEPROM types may be plugged in for study and debugging.

Two subtle differences between the 24LC00 and the other EEPROMS in the family (lack of address pins and lack of ‘page write’) are mentioned in this note. This note concludes with a set of general-purpose 8051 subroutines that perform byte reads and writes to the 24LC00 on the EZ-USB Development Board. The routines assume the lowest common functionality in the 24LC family, making them usable for any of the family devices (’00, ’01, and ’02).

EEPROM Details

The 24LC00 responds to the I²C slave address 1010xxxd, where “d” is the direction bit (d=1 for read, d=0 for write). The ‘xxx’ indicates that the device responds to all eight slave addresses between 1010 000 and 1010 111. This is due to the fact that the 24LC00 lacks the A[2..0] pins found on the 24LC01 and 24LC02 devices.

The 24LC00 contains an internal address counter that must be written before reading or writing an EEPROM byte. This address counter automatically increments for sequential EEPROM byte reads, but it does not increment for sequential byte writes. This is the major difference between the 24LC00 and the 24LC01/02 devices. The 24LC01/02 devices also have a “page write” mechanism that allows loading eight bytes at a time, where each load automatically increments the internal address counter. By writing the example routines to explicitly write an address before each EEPROM byte write, the code is compatible with all three part types.

Although the 24LC series is called ‘electrically erasable”, there is no explicit erase operation. Writing a byte first erases the byte, then reprograms it.

An EEPROM write operation consists of sending the bytes shown in *Table 2* to the 24LC00.

Table 2. 24LC00 Write Operation

Byte	Value	Meaning
1	10100000	Command byte—Write
2	0000aaaa	EEPROM address to write
3	Dddddddd	Data to write

An I²C START condition precedes the first byte, and a STOP condition follows the final byte. The STOP condition initiates the erase/write cycle, which takes a maximum of 4 milliseconds to complete in the 24LC00.

The 8051 checks for completion of an erase/write cycle by repeatedly sending “byte write” commands (byte 1 in *Table 2*) to the EEPROM, and checking the ACK bit. During the EEPROM programming cycle the EEPROM responds with ACK=0 (not acknowledge) on the I²C bus. When the programming cycle is complete it responds with ACK=1 (acknowledge) on the I²C bus. (These two conditions are shown in *Figure 3*.) Note that the I²C *bus* polarities are the opposite sense of the ACK bit in the I2CS register (I2CS.1), so the 8051 polling routine waits while I2CS.1=0 and exits when I2CS.1=1.

I²C Data Transfers

The 8051 communicates with the I²C bus using two registers, shown below:

I2CS			I^2C Control and Status				7FA5
b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	x	x	0	0	0

I2DAT			I^2C Data				7FA6
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

The 8051 initiates a transfer by setting the START bit (I2CS.7), and then writing a command byte to the I2DAT register. When the I^2C controller is ready, the 8051 then writes or reads data to/from I2DAT. Finally, the 8051 sets the STOP bit to terminate the transaction.

The I2CS register bits operate as follows:

DONE

When the 8051 initiates an I^2C transfer, the DONE bit (I2CS.0) goes LOW, and returns HIGH when the transfer completes and the EZ-USB I^2C controller is ready. The DONE bit is automatically cleared when the 8051 reads or writes I2DAT.

STOP

The 8051 terminates an I^2C transfer by setting STOP=1. This bit stays HIGH until the I^2C controller finishes sending the STOP condition on the I^2C bus, at which time it clears the STOP bit. Completion of the STOP condition on the I^2C bus has no effect on the DONE bit.

For an I^2C read operation, the 8051 must read the last data byte from I2DAT before the STOP condition completes (within about 11 microseconds). Therefore the 8051 code should read the I2DAT register immediately after setting the STOP bit. It is therefore good practice to begin every I^2C transfer routine with a check for STOP=0, indicating that any previous STOP condition has completed and the I^2C controller is "listening."

LastRD

The 8051 sets the LASTRD bit (I2CS.5) before clocking in the last byte in a read operation. This instructs the EZ-USB I^2C controller not to generate an ACK for the last transfer. The lack of an ACK from the I^2C master (the EZ-USB I^2C controller) signals the I^2C peripheral to stop sending.

ACK, BERR

After every transfer (when DONE goes HIGH), two status bits indicate if the transfer was acknowledged (ACK, I2CS.1) and if another I^2C device interfered by driving the bus at the same time as the EZ-USB controller (BERR, I2CS.2). The example code checks the ACK bit for completion of an erase/program cycle. For simplicity, and because no other I^2C device is present on the EZ-USB Development board, the example code does not check the BERR bit.

ID1–ID0

These bits, which indicate the EEPROM type detected by the EZ-USB I^2C boot loader, may be safely ignored for these examples. For reference, Chapter 4 of the *EZ-USB Technical Reference Manual* describes the meaning of these read-only bits.

Code Description

The code listing at the end of this note contains several subroutines:

1. Reset EEPROM address to 0000 ('reset_address', line 218).
2. Read first eight bytes from the EEPROM ('read8_eeprom', line 63). The 8051 code stores these bytes in internal registers at 0x80-0x87.
3. Write the sixteen EEPROM bytes with the data in 8051 internal registers at 0x80-0x8F ('write16_eeprom', line 125).
4. Test the above routines ('test', line 39), as follows. Call (1) to reset the EEPROM address pointer to zero, call routine (2) to read the first eight EEPROM bytes into 0x80-0x87, fill the eight bytes at 0x88-0x8F with the values 8,7,6,5,4,3,2,1, and finally call (3) to write the EEPROM using the sixteen bytes at 0x80-0x8F.

The result is an EEPROM that contains the first eight bytes previously stored in the EEPROM, and the count 8->1 in the last eight bytes. It should be easy to modify the example test routine to write any desired data.

Programming the LastRD Bit

The LastRD bit in I2CS.5 instructs the EZ-USB I^2C controller to float the SDA line at 'ACK' time (the ninth SCL of the byte transfer) to alert the slave (EEPROM) to stop sending. Looking at the code, it may appear that the LastRD bit is actually set two bytes before the last transfer, since the code sequence to read eight bytes is as follows:

1. Perform a dummy read of I2DAT to send out the first nine SCL pulses which clock the first byte into the I2DAT register.
2. Read the first six bytes from I2DAT.
3. Set the LastRD bit.
4. Read the seventh byte from I2DAT.

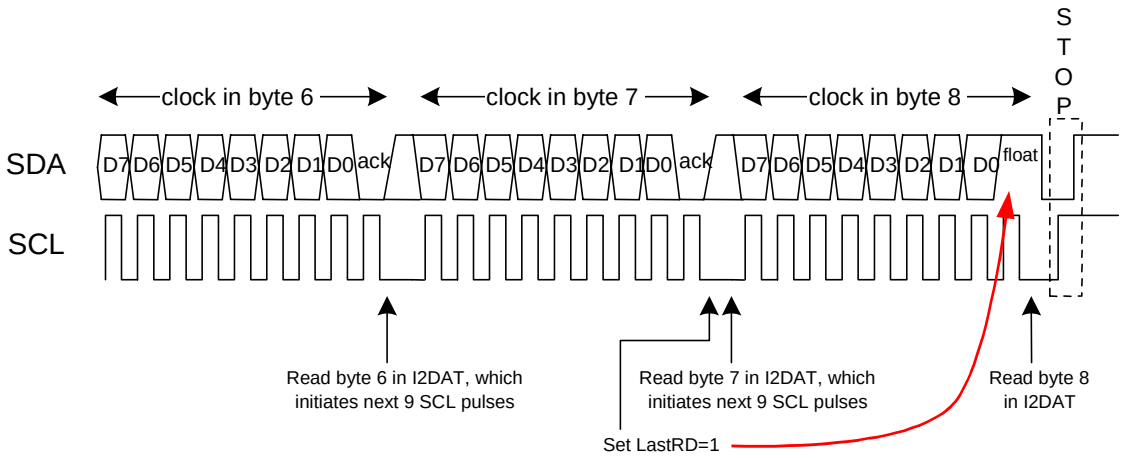

Figure 1. The 8051 Sets LastRD=1 Before Reading the Second-to-last byte from I2DAT

5. Read the eighth byte from I2DAT.
6. Set STOP=1.

The reason the LastRD is set to one before reading the seventh, and not the eighth byte, is that the 8051 must set LastRD=1 before the last nine clock pulses are sent out by the EZ-USB controller. Therefore LastRD must be set to 1 before the 8051 reads I2DAT to retrieve the second-to-last byte (byte 7). This is illustrated in *Figure 1*.

Waveforms

Figure 2 illustrates the sixteen EEPROM byte write operations. Trace 2 is the port pin PORTA.0, which is programmed to be HI while a write cycle is in progress (set HI in lines 174–176, and LO in lines 198–200). The sixteen byte write times can clearly be seen in trace 2. Note that the 16-byte write takes less than 40 milliseconds. *Figure 3* (next page) shows the same data magnified to show the last write command byte that returns a non-ACK (EEPROM still busy writing) followed by the command byte that returns ACK to indicate that the write cycle has finished.

The 8051 sends the command byte “10100000” in both transfers in *Figure 3*. In the first transfer the 24LC00 sends ACK=1 (not-acknowledge) to indicate that a write cycle is still in progress. In the second transfer it sends ACK=0 (acknowledge) to indicate that the write cycle has completed and another byte can be erased/written.

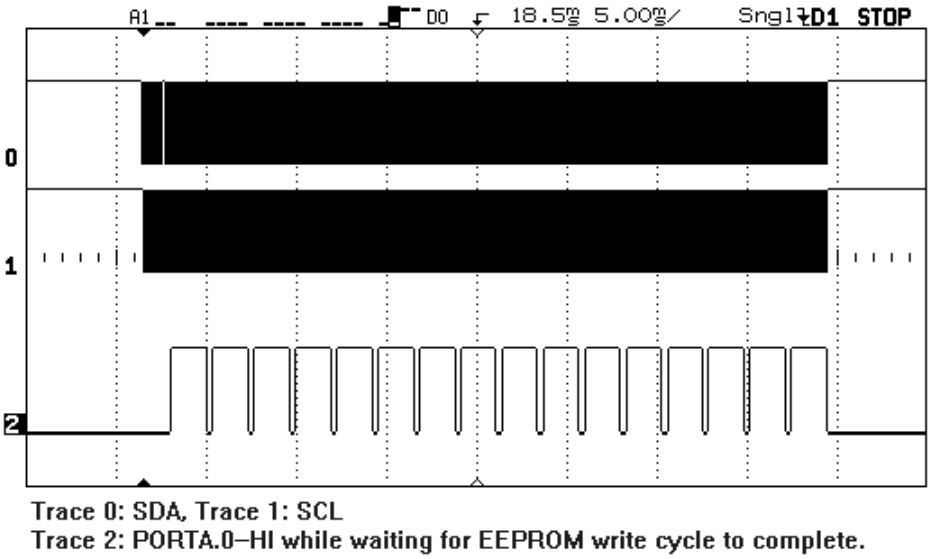

Figure 2. Scope Traces for Listing A

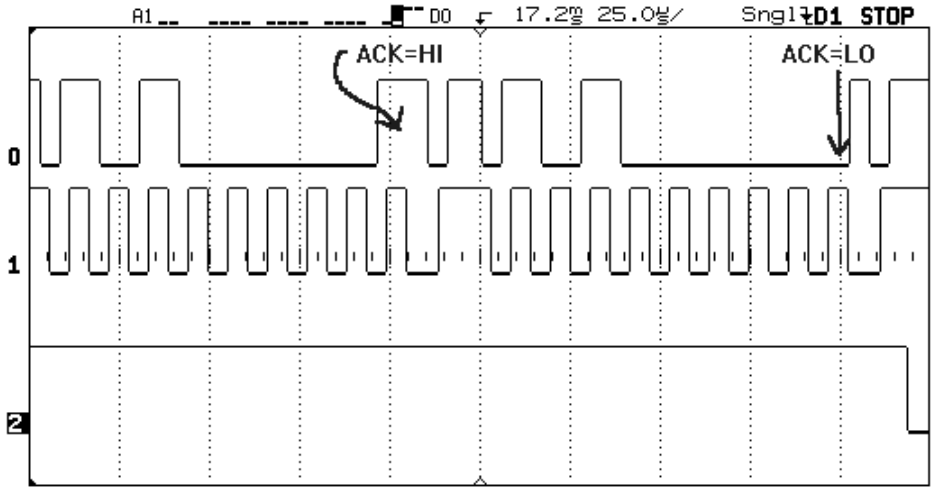

Figure 3. Two Command Bytes Indicate 'busy' and 'done'.

Listing

```
1 ; ---------------------------------------------------------------------------
2 ; eeprom.A51  10-7-98  LTH
3 ; Subroutines to read and write the 24LC00 attached to the AN2131Q on the EZ-USB
4 ; Development Board.
5 ;
6 ; reset_address      send 0000 to the EEPROM address counter
7 ; read8_eeprom       read the first 8 bytes from the (16-byte) 24LC00, stash
8 ;                   data in internal register memory 0x80-0x87.
9 ; write16_eeprom     write 16 bytes at internal register memory 0x80-0x8F to
10 ;                  the 24LC00
11 ;
12 ; test              read first 8 EEPROM bytes at 0x80-0x87, write new data to
13 ;                  0x88-0x8F, then re-program the EEPROM with sixteen bytes at
14 ;                  0x80-0x8F.  The EEPROM then contains the previous
15 ;                  first 8 bytes (the VID/PID/DID info needed for correct EZ-USB
16 ;                  Development Board operation) plus 'user' data in the last
17 ;                  eight bytes.
18 ; ---------------------------------------------------------------------------
19 $NOMOD51            ; disable predefined 8051 registers
20 $nolist
21 $INCLUDE (REG320.INC); *** for the integrated 8051 core
22 $include (ezregs.inc); EZ-USB register assignments
23 $list
24 ;
25 NAME         eeprom
26 ;
27             ISEG AT 60H
28 stack:      ds      20
29 ;
30             DSEG at 20H           ; start of bit-addressable regs
31 EEaddr:     ds      1
32 EEdata:     ds      1
33 ;
34             CSEG AT 0
35             ljmp    test
36 ; ------------------------------------------------
37             org     200h
38 ; ------------------------------------------------
```

```
39 test:       mov    SP,#STACK-1    ; set stack
40 ;
41             mov    dptr,#OEA      ; enable PORTA.0 for output
42             mov    a,#1           ; bit 0
43             movx   @dptr,a        ; (use as scope 'eeprom busy' indication)
44 ;
45             call   read8_eeprom   ; put 16 bytes at idata 80H-8FH
46 ;
47             mov    r0,#88H        ; modify the last 8 bytes (88-8FH)
48             mov    r7,#8
49 modify:     mov    a,r7
50             mov    @r0,a          ; write values 8->1
51             inc    r0
52             djnz   r7,modify
53 ;
54             call   write16_eeprom
55 spin:       sjmp    spin          ; hang here
56 ;
57
58 ;-------------------------------------------------
59 ; Read the first eight bytes from the 24LC00 EEPROM
60 ;-------------------------------------------------
61 ; 0. Make sure STOP is not in progress
62 ;
63 read8_eeprom:call  stop_check
64 ;
65 ; 1. Set EEPROM address pointer to 0000
66 ;
67             call   reset_address  ; set eeprom address pointer to 0
68             call   stop_check
69 ;
70 ; 2. Set the START bit
71 ;
72             mov    dptr,#I2CS
73             mov    a,#10000000b   ; b7=start bit
74             movx   @dptr,a
75 ;
76 ; 3. Write the EEPROM address 1010 000 and indicate read operation (b0=1)
77 ;
78             mov    dptr,#I2DAT
79             mov    a,#10100001b
80             movx   @dptr,a
81             call   wait_done
82 ;
83 ; 4. Read first eight bytes
84 ;
85             mov    dptr,#I2DAT
86             movx   a,@dptr        ; dummy read to generate first 9 SCL pulses
87             call   wait_done
88 ;
89             mov    r7,#6          ; read first 6 bytes
90             mov    r0,#80H        ; deposite the bytes here
91 rdloop:             mov    dptr,#I2DAT
92             movx   a,@dptr
93             mov    @r0,a
94             inc    r0
95             call   wait_done
96             djnz   r7,rdloop
97 ;
98 ; 5. Set the LastRd bit and read 7th and 8th bytes
99 ;
100            mov    dptr,#I2CS
101            mov    a,#00100000b   ; b5=LastRD
102            movx   @dptr,a
```

```
103             mov     dptr,#I2DAT
104             movx    a,@dptr         ; read the 7th byte
105             mov     @r0,a           ; save it
106             inc     r0               ; bump dest pointer
107             call    wait_done
108 ;
109             mov     dptr,#I2DAT
110             movx    a,@dptr         ; read the 8th byte
111             mov     @r0,a           ; save it
112 ;
113 ; 6. Set the STOP bit
114 ;
115             mov     dptr,#I2CS
116             mov     a,#01000000b
117             movx    @dptr,a
118             ret
119 ;
120
121 ; ------------------------------------------------------------
122 ; Write EEPROM with 16 bytes at internal register RAM 0x80-0x8F.
123 ; ------------------------------------------------------------
124 ;
125 write16_eeprom:mov r0,#80H          ; source data pointer
126             mov     EEaddr,#0       ; starting address in eeprom
127             mov     r7,#16          ; pass count
128 weloop:     mov     EEdata,@r0
129             inc     r0
130             call    write_eeprom_byte
131             inc     EEaddr
132             djnz    r7,weloop
133             ret
134 ;
135 ; -------------------------------------------------------
136 ; Write an eeprom byte.  Data in EEdata, addr in EEaddr.
137 ; -------------------------------------------------------
138 write_eeprom_byte:
139 ;
140 ; 0. Make sure STOP is not in progress
141 ;
142             call    stop_check
143 ;
144 ; 1. Set the START bit
145 ;
146             mov     dptr,#I2CS
147             mov     a,#10000000b    ; b7=start bit
148             movx    @dptr,a
149 ;
150 ; 2. Write the EEPROM address 1010 000 and indicate write operation (b0=0)
151 ;
152             mov     dptr,#I2DAT
153             mov     a,#10100000b
154             movx    @dptr,a
155             call    wait_done
156 ;
157 ; 3. Send the EEPROM 'word address'
158 ;
159 wloop:      mov     dptr,#I2DAT
160             mov     a,EEaddr        ; address pointer
161             movx    @dptr,a         ; send command byte
162             call    wait_done
163 ;
164             mov     dptr,#I2DAT
165             mov     a,EEdata        ; send eeprom data byte
166             movx    @dptr,a
```

```
167             call    wait_done
168 ;
169             mov     dptr,#I2CS      ; send STOP.  This initiates an internal write cycle
170             mov     a,#01000000b
171             movx    @dptr,a
172             call    stop_check      ; wait for completion of STOP
173 ;
174             mov     dptr,#OUTA      ; for the scope
175             mov     a,#1
176             movx    @dptr,a         ; hi means waiting to finish write cycle
177 ;
178 ; Wait for the erase/write cycle to complete.  This is done by sending a 'write' command
179 ; byte and checking the ACK bit.  The EEPROM will not respond (ACK) while a write cycle
180 ; is in progress.
181 ;
182 write_wait:mov     dptr,#I2CS      ; send START
183             mov     a,#10000000b    ; b7=start bit
184             movx    @dptr,a
185             mov     dptr,#I2DAT     ; write a command byte
186             mov     a,#10100000b    ; eeprom write
187             movx    @dptr,a
188             call    wait_done
189             mov     dptr,#I2CS      ; send STOP
190             mov     a,#01000000b
191             movx    @dptr,a
192             call    stop_check      ; wait for completion of STOP
193 ;
194             mov     dptr,#I2CS      ; check ACK bit
195             movx    a,@dptr
196             jnb     acc.1,write_wait; keep trying until ACK=1
197 ;
198             mov     dptr,#OUTA      ; done waiting--scope signal LO
199             mov     a,#0
200             movx    @dptr,a
201 ;
202             ret
203 ;
204 ;----------subroutines-------------
205 ;
206 stop_check:mov     dptr,#I2CS      ; check the STOP bit in I2CS
207 stck:      movx    a,@dptr
208             jb      acc.6,stck
209             ret
210 ;
211 wait_done: mov     dptr,#I2CS      ; check the DONE bit in I2CS
212 cd1:       movx    a,@dptr
213             jnb     acc.0,cd1
214             ret
215 ;
216 ; Set the internal EEPROM address counter to zero
217 ;
218 reset_address:
219 ;
220 ; 0. Make sure STOP is not in progress
221 ;
222             call    stop_check
223 ;
224 ; 1. Set the START bit
225 ;
226             mov     dptr,#I2CS
227             mov     a,#10000000b    ; b7=start bit
228             movx    @dptr,a
229 ;
230 ; 2. Write the eeprom address 1010 000 and indicate write operation (b0=0)
```

```
231 ;
232             mov     dptr,#I2DAT
233             mov     a,#10100000b   ; address the EEPROM
234             movx    @dptr,a
235             call    wait_done
236 ;
237 ; 3. Sent a 0 data byte to zero the EEPROM internal address pointer.
238 ;
239             mov     dptr,#I2DAT
240             mov     a,#0           ; data=0
241             movx    @dptr,a
242             call    wait_done
243 ;
244 ;
245 ; 4. Set the STOP bit
246 ;
247             mov     dptr,#I2Cs
248             mov     a,#01000000b
249             movx    @dptr,a
250             ret
251 ;
252             END
253
254
255
```