Chapter 6 Serial EEPROM

6.1 ATMEL I²C Serial EEPROM

The AT24C256 (U6) is useful for our application in storing critical data like temperature and humidity data, its time stamp, graphics, icons, or even fonts for the graphical LCD. AT24C256 provides 262,144bits of serial electrically erasable and programmable read only memory. The memory is organized as 32,768 words of bytes arranged in page size of 64-bytes each; therefore, there are 512 pages available.

A simple mathematic: 262,144 bits = 512 pages x 64bytes x 8 bits per bytes.

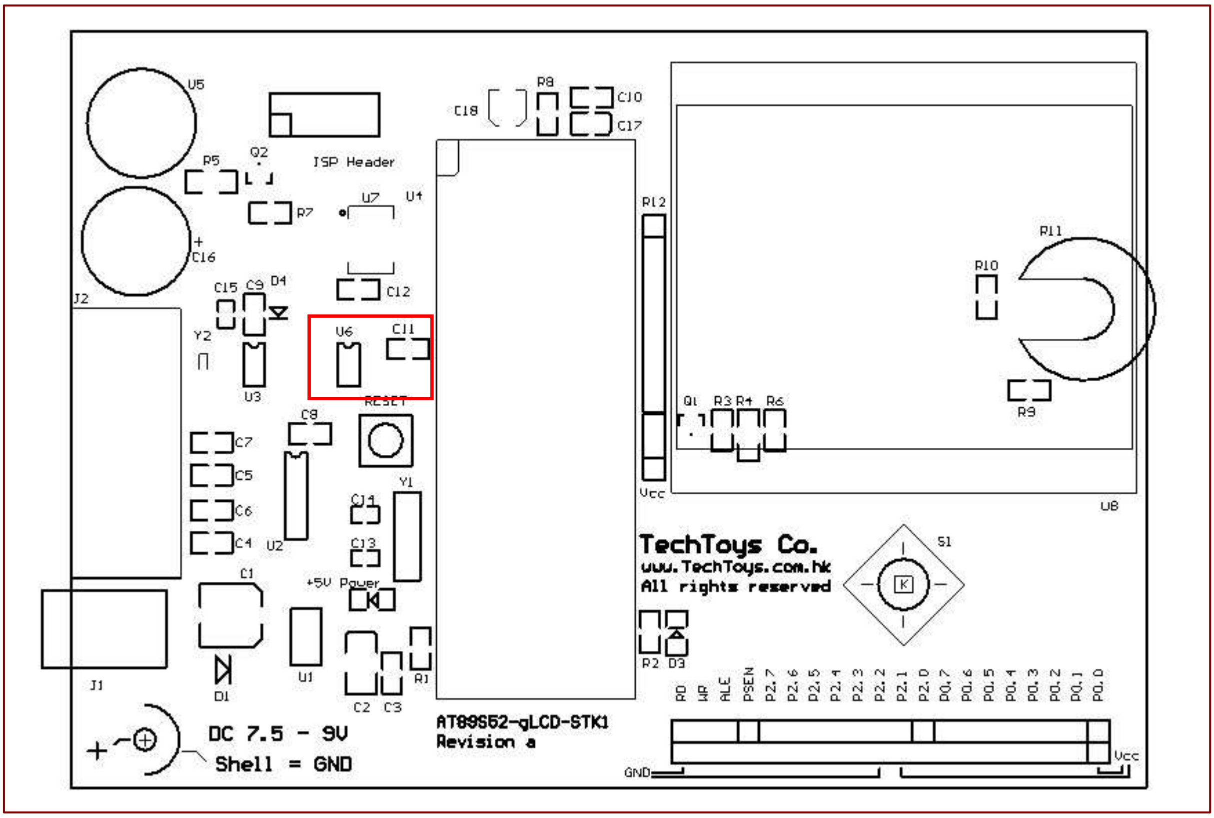

Figure 6.1.1 AT24C256 EEPROM PCB layout

AT24C256 is sharing the same I^{2}C-bus with PCF8563. The same low-level I^{2}C driver is used for both devices. The idea is illustrated in Figure 6.1.2.

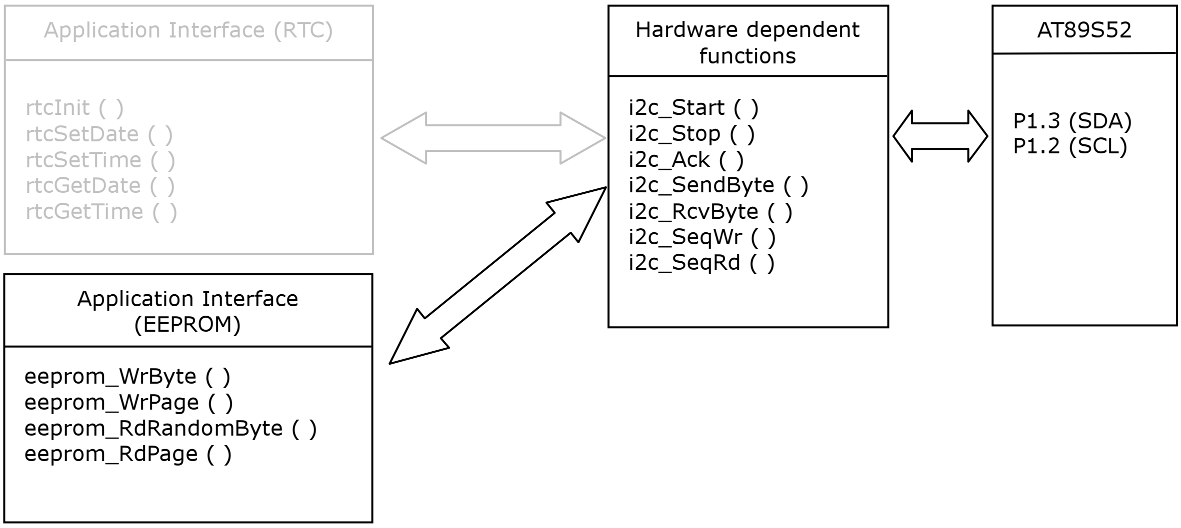

Figure 6.1.2 I^{2}C driver module block diagram

Figure 6.1.3 correlates the logic levels to function calls for writing a single byte 0xA8 to AT24C256 EEPROM at its internal address 0x00C7. The device address is 0xA0, again assigned by the manufacturer.

1. Mcu generates a start condition. The mcu initiates a data transfer by "telling" all I^{2}C devices that, now I want to "talk" to somebody.
2. Locate the address of AT24C256 since it is possible to have more than one device on the same I^{2}C-bus (it is PCF8563 in our case). READ/WRITE command is embedded as the last bit of the device address. Then, the mcu waits for an acknowledge signal from AT24C256.
3. Mcu sends the internal register address (first and second word addresses) to AT24C256. The mcu waits for ACK after each address byte sent.
4. Mcu continues sending data. It can be a single byte or multiple bytes for single byte write or page write.
5. Mcu stops communication by issuing a stop condition over the I^{2}C-bus.

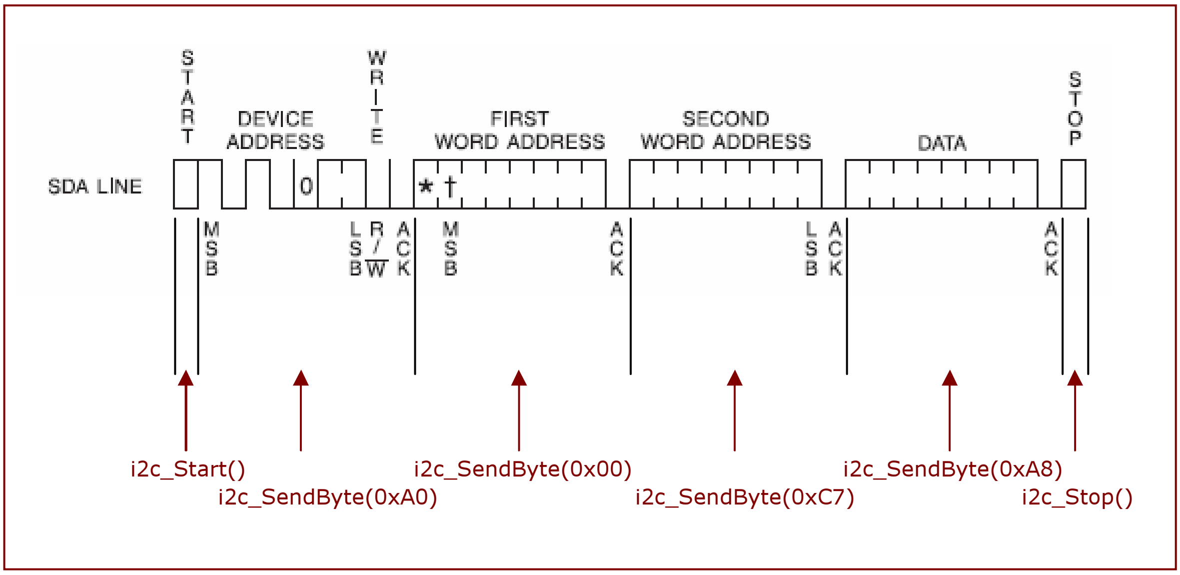

Figure 6.1.3 AT24C256 EEPROM single-byte write

Figure 6.1.3 translates to API function i2c_SeqWr() as shown in listing 6.1.1. There is no need to use i2c_Start() and i2c_SendByte() individually as we have encapsulated the whole sequence inside i2c_SeqWr() with argument 'length' equate 1.

```
uchar eeprom_WrByte(uchar eeprom_addr, uint dataptr, uchar c)
{
        uchar dataptrh, dataptrl;

        dataptrh = (uchar) (dataptr>>8);
        dataptrl = (uchar) (dataptr);

        if(i2c_SeqWr(eeprom_addr, dataptrh, dataptrl, &c, 1)==noACK)
        return (EEPROM_BUS_ERR);

        return (EEPROM_BUS_OK);
}
```

Listing 6.1.1 eeprom_WrByte() function

Warning: a write cycle does not happen instantaneously. There is a Write Cycle Time as stated on data sheet. Its value ranges from 20ms to 5ms. Figure 6.1.4 shows an extract of the data sheet about Byte Write operation.

BYTE WRITE: A write operation requires two 8-bit data word addresses following the device address word and acknowledgment. Upon receipt of this address, the EEPROM will again respond with a zero and then clock in the first 8-bit data word. Following receipt of the 8-bit data word, the EEPROM will output a zero. The addressing device, such as a microcontroller, then must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally-timed write cycle, t_{WR}, to the nonvolatile memory. All inputs are disabled during this write cycle and the EEPROM will not respond until the write is complete (see Figure 8 on page 11).

Figure 6.1.4 BYTE WRITE OPERATION

That is why we need to delay for 5ms after each eeprom_WrByte() in the coming demonstration program (Listing 6.1.3).

If a large block of data is written to EEPROM, we may use Page Write. A page write is initiated the same way as a byte write, but the microcontroller does not send a stop condition after the first data word is clocked in. Instead, after the EEPROM acknowledges receipt of the first data word, the microcontroller can transmit up to 63 more data words.

Refer to the source code for details on eeprom_WrPage ( ).

What about reading a byte from EEPROM? Take an example of Random Read (Figure 6.1.5).

> **RANDOM READ:** A random read requires a "dummy" byte write sequence to load in the data word address. Once the device address word and data word address are clocked in and acknowledged by the EEPROM, the microcontroller must generate another start condition. The microcontroller now initiates a current address read by sending a device address with the read/write select bit high. The EEPROM acknowledges the device address and serially clocks out the data word. The microcontroller does not respond with a zero but does generate a following stop condition (see Figure 11 on page 12).

Figure 6.1.5 Extract from AT24C256 data sheet on RANDOM READ

Again, no more individual call to i2c_Start() & i2c_SendByte() required. We simply make use of i2c_SeqRd() by putting argument length =1.

```
uchar eeprom_RdRandomByte(uchar eeprom_addr, uint dataptr, uchar *s)
{
        uchar dataptrh, dataptrl;

        dataptrh = (uchar) (dataptr>>8);
        dataptrl = (uchar) (dataptr);

        if(i2c_SeqRd(eeprom_addr, dataptrh, dataptrl, &s[0], 1)==noACK)
        return (EEPROM_BUS_ERR);

        return (EEPROM_BUS_OK);
}
```

Listing 6.1.2 RANDOM READ BYTE OPERATION

Refer to the source code for details on eeprom_RdPage ( ).

Listing 6.1.3 shows at24c256_demo1.c.
The source code at cd:\src\chp6\src6_1\at24c256_demo1.Uv2.

```
#include <REGX52.h>
#include "at24c256.h"
#include "delay.h"
#include <stdio.h>          //Keil library

void init_uart()
{
 SCON  = 0x52;
 TMOD  = 0x20;
 TCON  = 0x69;
 TH1   = 0xfd;
}

void main(void)
{
        unsigned char eeprom_dat;
        unsigned int  i;
        unsigned char code s[64]={                                              (1)
        0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,\
        20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,\
        40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,\
        60,61,62,63};

        unsigned char k[64];                                                    (2)

        init_uart();
        printf("TESTING EEPROM\n");                                             (3)
        DelayMs(1000);

        eeprom_WrPage(EEPROM_ADDR, 0x0000, &s[0], 64);                          (4)
        DelayMs(10);                                                            (5)

        eeprom_RdPage(EEPROM_ADDR, 0x0000, &k[0], 64);                          (6)
        for(i=0;i<64;i++)
                printf("eeprom_dat:%bu  eeprom_address at: %u\n", k[i],i);      (7)

        i=64;
        while(i<200)
        {
         if(eeprom_WrByte(EEPROM_ADDR, i, 0xA8)==EEPROM_BUS_ERR)                (8)
         {
                if(eeprom_WrByte(EEPROM_ADDR, i, 0xA8)==EEPROM_BUS_ERR)         (9)
                printf("Write error at address: %u\n", i);                      (10)
         }

         DelayMs(5);                                                            (11)

         if(eeprom_RdRandomByte(EEPROM_ADDR, i, &eeprom_dat)==EEPROM_BUS_OK)    (12)
                printf("eeprom_dat:%bX  eeprom_address at: %u\n", eeprom_dat,i); (13)
         else
                printf("Read error at address: %u\n", i);                       (14)

         i++;
        }

        for(;;)
        {
                ;
        }
}
```

Listing 6.1.3 at24c256_demo1.c

Figure 6.1.6a and Figure 6.1.6b show the Docklight screenshots by running this demonstration.

eeprom_dat runs from 0 to 63, until eeprom_address at 64, eeprom_dat = A8. eeprom_dat equates 0xA8 as we have instructed in Line (8) until eeprom_addr finishes at 199.

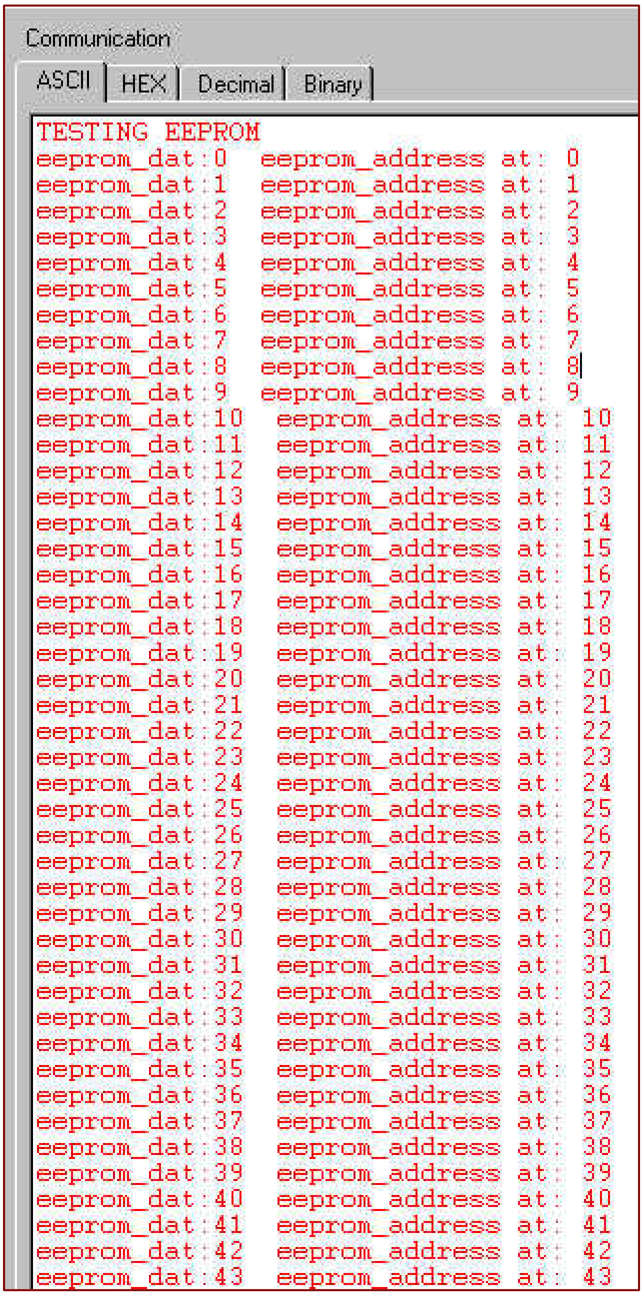

Figure 6.1.6a

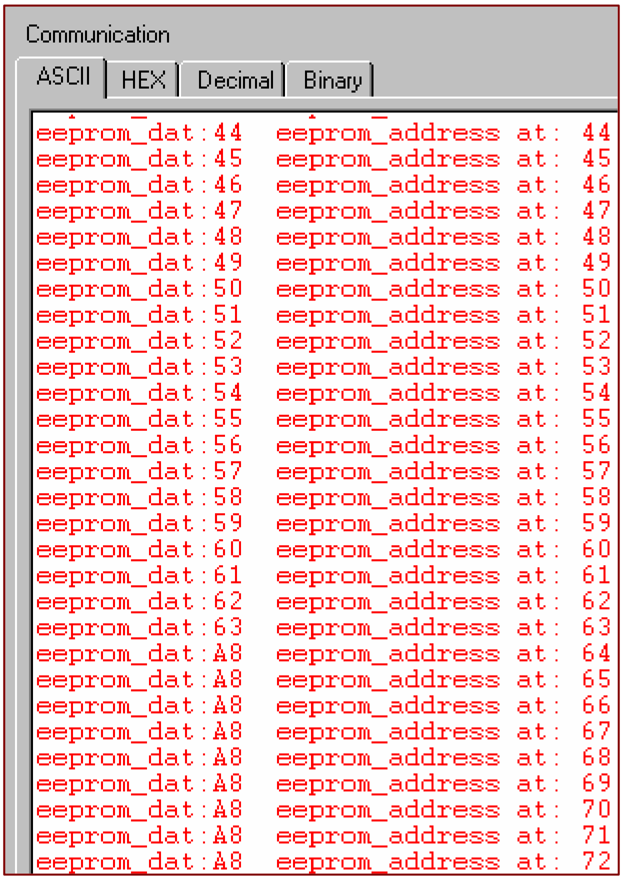

Figure 6.1.6b

⇒ Chapter 7