Chapter 5 Real time clock

5.1 Philips PCF8563 Real time clock (RTC)

Philips PCF8563 (U5) is an I²C compatible real time clock (RTC). Alternatively, this chip can be replaced by a software module like the Time-of-Day Clock module described in Labrosse's book[1]. However, since we are using only the 2K evaluation copy of Keil C, a software-based solution would be out of consideration due to its code size.

RTC is useful for our application in providing time stamps for data acquisition. Philips PCF8563 has been chosen because it is readily available, low cost, and I²C compatible. The last factor leads us to ATMEL serial EEPROM for data storage because it is I²C compatible as well. We will come back on serial EEPROM in chapter 6.

A high value capacitor (C16), namely the super-cap, is included as a backup power source. Its value is 0.047F (47,000μF), enough for hours of main power removal without losing data of the current time. Figure 5.1.1 shows the PCB layout with all RTC components highlighted. Figure 5.1.2 shows an extract of the schematic. There are internal pull-up resistors for P1, 2, &3 so external pull-ups have been omitted for P1.2 & P1.3.

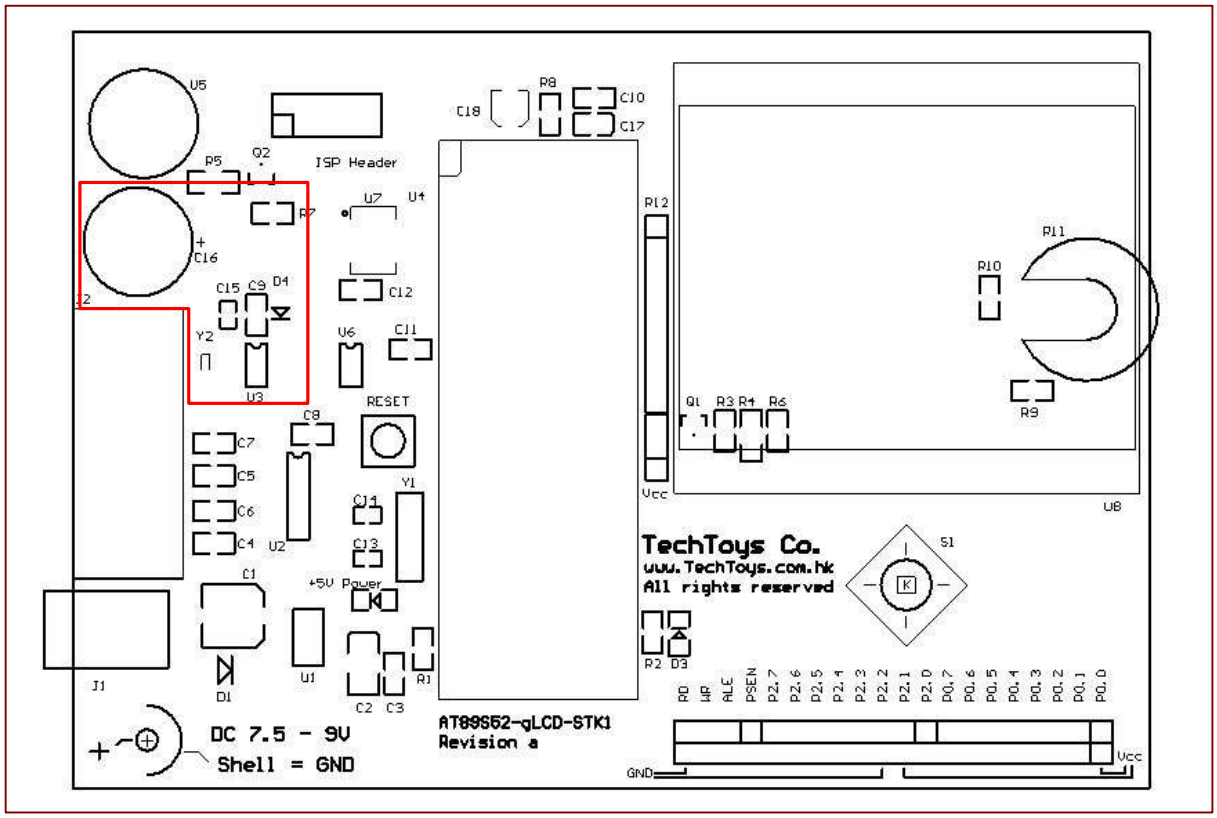

Figure 5.1.1 Real time clock PCB layout

[1] Chapter 6: Time-of-Day Clock, Embedded System Building Blocks, 2nd edition, by Jean J. Labrosse

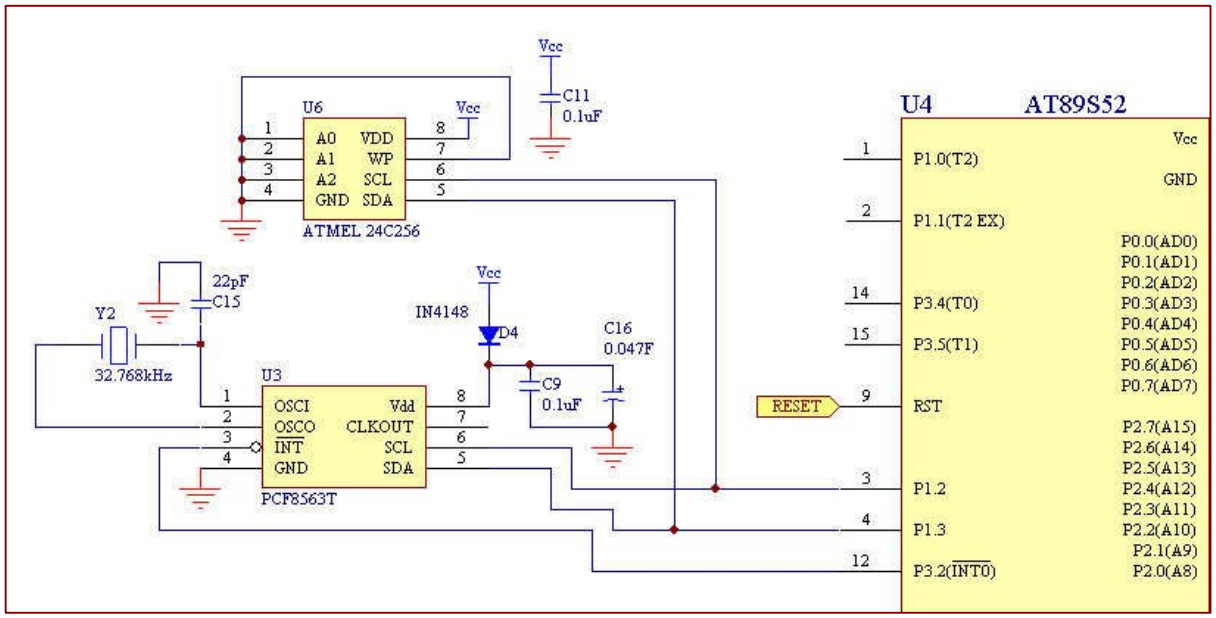

Figure 5.1.2 I²C-bus, interface for PCF8563 & 24C256

PCF8563 is sharing the same I²C-bus with ATMEL 24C256. The same low-level I²C driver is used for both devices. The idea is illustrated in Figure 5.1.3.

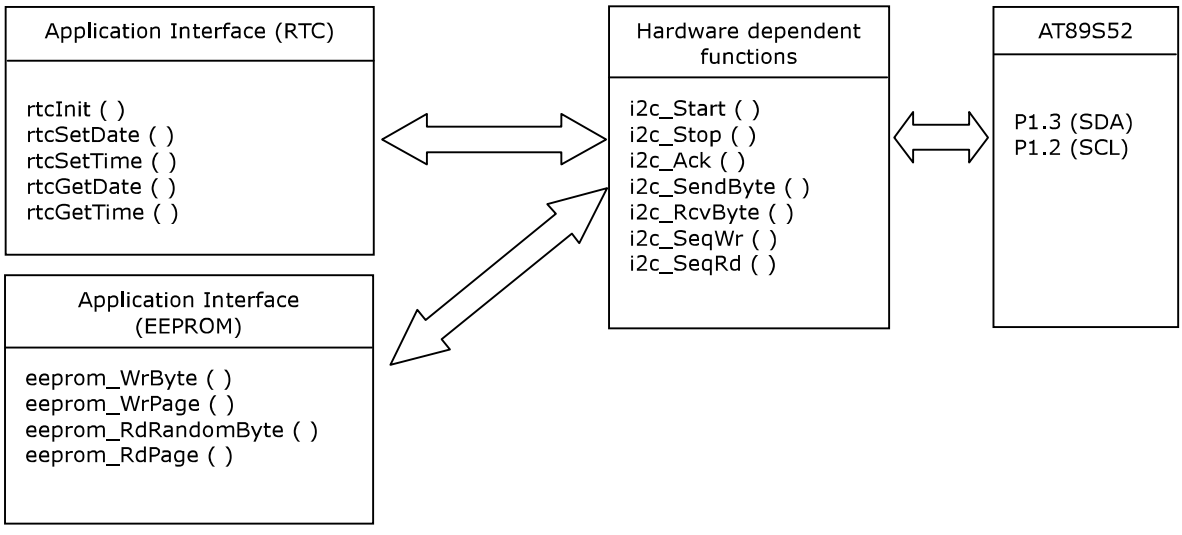

Figure 5.1.3 I²C driver module block diagram

Application interface for both devices would be hardware independent functions in C programming language. Hardware dependent functions have been coded as a separate software module (iic.c & iic.h). They have been custom made for AT89S52 at a crystal frequency of 12MHz (alright for 11.0592MHz as well) and Keil C specific. Because I²C is a common protocol for microcontrollers, this method works for other devices as long as

they are I²C compatible. If we are using the same RTC or EEPROM for other microcontrollers, or switching to another C compiler, it would be alright to modify just iic.h and iic.c and leave API for RTC and EERPOM unchanged or minimal touch-up.

References for I²C protocol can be found from the following links:

http://www.robot-electronics.co.uk/htm/using_the_i2c_bus.htm
http://users.telenet.be/educypedia/electronics/I2C.htm

Just type in iic or i2c tutorial under Yahoo or Google, there will be over hundreds of nice articles at your finger tip. So, only a correlation between the software driver and data read/write scenario is presented here rather than repeating I²C protocol itself.

For Philips RTC, the following sequence is involved to set time to 23:59:30 (say). Slave address for Philips 8563 RTC is 0xA2. This address is assigned by the chip manufacturer. This is established by some sort of international committee, I think.

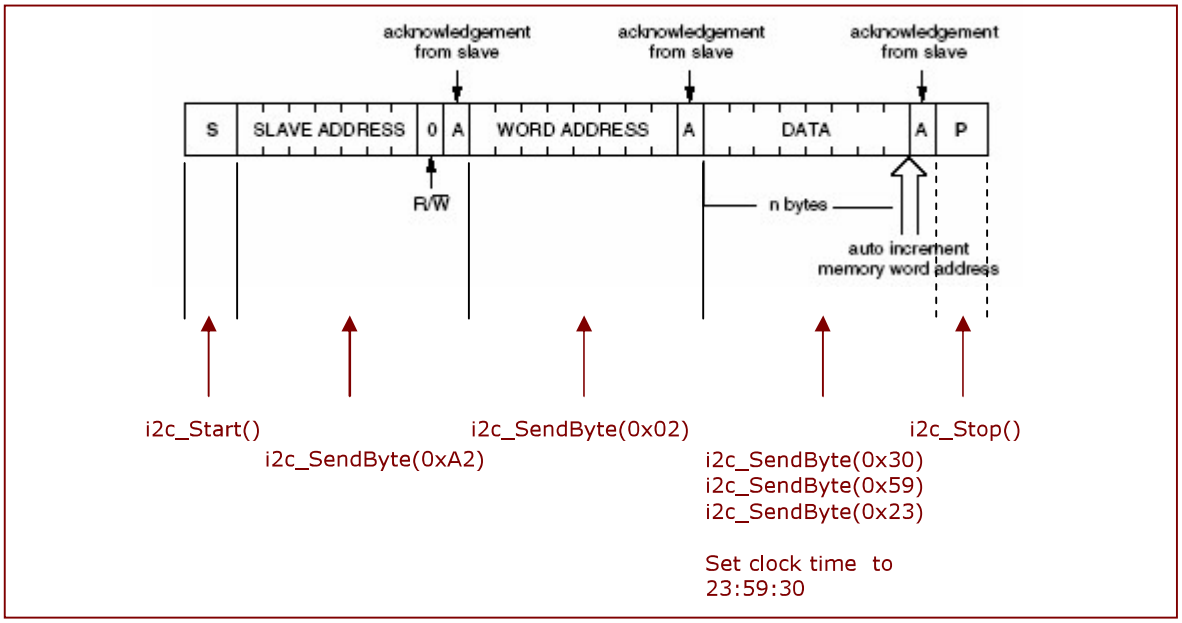

Figure 5.1.4 Set current time for Philips PCF8563 RTC

We need to fill-in the contents of the seconds, minutes, and hours registers to set the current time in sequence of sec:min:hour in BCD format, and we are doing this by repeat calling the function i2c_SendByte(). After we have identified the correct device (slave address=0xA2), the internal register address of that particular device is located by the second call to i2c_SendByte() with argument being 0x02. The WORD ADDRESS at 0x02 is exactly the register address of seconds as shown in the register map of PCF8563 (Figure 5.1.5). Then we are writing 0x30, 0x59, & 0x23 to set the current time to 23:59:30. Since there is an auto increment feature, we don't need to take care of the address pointer increment after having located the first WORD ADDRESS at 0x02. Finally, the mcu closes communication by issuing an i2c_Stop(). That's it!

Address	Register name	BCD format tens nibble				BCD format units nibble			
		Bit 7 2^3	Bit 6 2^2	Bit 5 2^1	Bit 4 2^0	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0
02H	seconds	VL	<seconds 00 to 59 coded in BCD>						
03H	minutes	x	<minutes 00 to 59 coded in BCD>						
04H	hours	x	x	<hours 00 to 23 coded in BCD>					
05H	days	x	x	<days 01 to 31 coded in BCD>					
06H	weekdays	x	x	x	x	x	<weekdays 0 to 6>		
07H	months/century	C	x	x	<months 01 to 12 coded in BCD>				
08H	years	<years 00 to 99 coded in BCD>							
09H	minute alarm	AE	<minute alarm 00 to 59 coded in BCD>						
0AH	hour alarm	AE	x	<hour alarm 00 to 23 coded in BCD>					
0BH	day alarm	AE	x	<day alarm 01 to 31 coded in BCD>					
0CH	weekday alarm	AE	x	x	x	x	<weekday alarm 0 to 6>		

Figure 5.1.5 Register overview (Philips PCF8563)

It is like the first adjustment on date/time for a new piece of home appliance. We just need to do it once prior to using it. Thereafter the current date/time will be shown on the user panel. Showing the current time requires subsequent RTC-read. Reading the current time from PCF8563 involves the following function call (Figure 5.1.6).

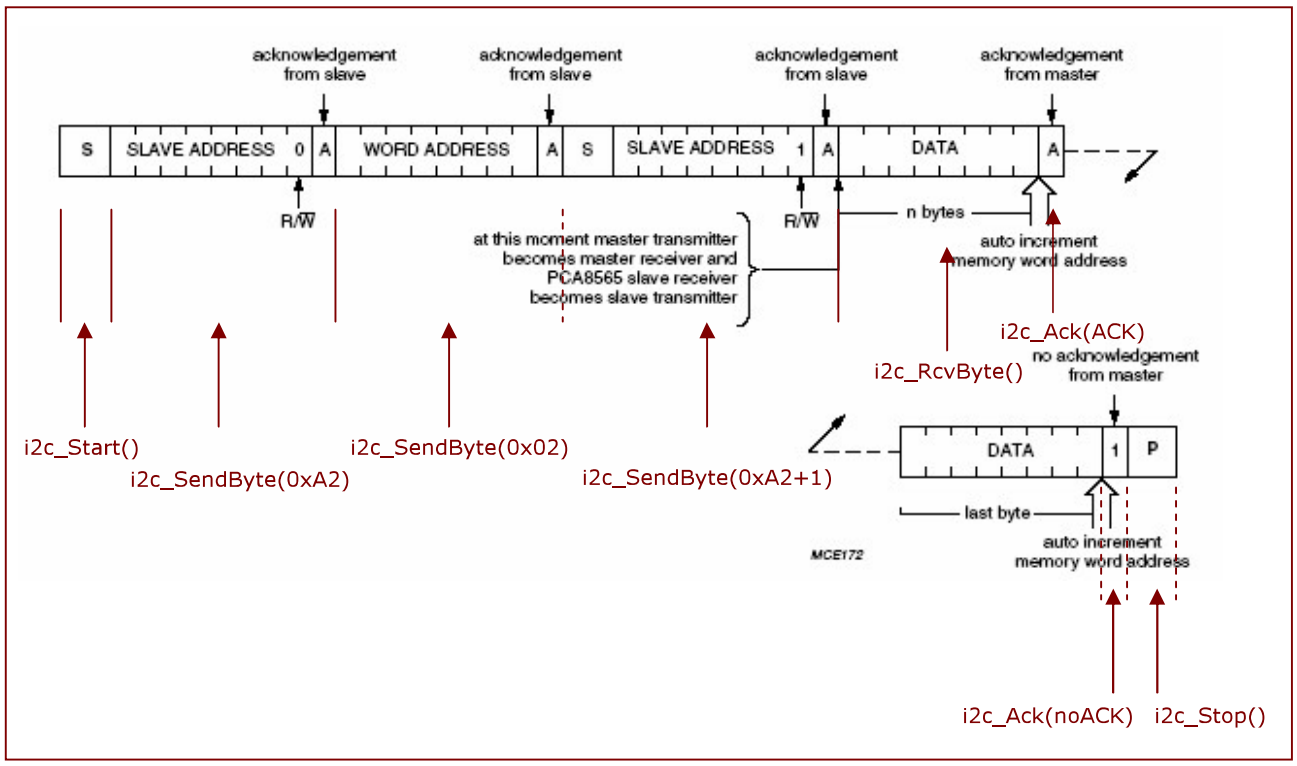

Figure 5.1.6 Read current time from PCF8563 RTC

The first three function calls are the same as that in time-set (Figure 5.1.4). They are

1. i2c_Start() // i2c start condition
2. i2c_SendByte(0xA2) // device address
3. i2c_SendByte(0x02). // internal register address to access

These functions are used to identify the correct device over the I^2C-bus, and locate the internal register of that particular device to read from.

Then we have to send a READ command by i2c_SendByte(0xA2+1). I^2C READ command is embedded in the device address with the last bit being 1. That's why the argument becomes 0xA2+1 (0xA3).

i2c_RcvByte() is used to return the byte read from the SDA data pin.
i2c_Ack(ACK) used to generate an ACK condition by the master (mcu), except for the last byte received no ACK signal required; therefore i2c_Ack(noACK) with the last byte. Finally mcu closes communication by calling i2c_Stop().

As shown above, there is not a lot of function required to write/read to-and-fro an I^2C device. Listing 5.1.1 shows the major parameters and functions.

Listing 5.1.1 shows the header file for low level I^2C functions (iic.h) for both Philips PCF8563 RTC and ATMEL 24C256 Serial EEPROM.

```
#ifndef _IIC_H_
#define _IIC_H_

sbit SDA      = P1^3;                                          (1)
sbit SCL      = P1^2;                                          (2)

#define noACK      0                                           (3)
#define ACK        1                                           (4)
#define NONE       0xFF                                        (5)


void i2c_Start(void);                                         (6)
void i2c_Stop(void);                                          (7)
void i2c_Ack(bit reply);                                      (8)
bit i2c_SendByte(unsigned char c);                            (9)
unsigned char i2c_RcvByte(void);                              (10)

bit i2c_SeqWr(unsigned char device, unsigned char addrh, unsigned char addrl,
unsigned char *s, unsigned char length);                      (11)

bit i2c_SeqRd(unsigned char device, unsigned char addrh, unsigned char addrl,
unsigned char *s, unsigned char length);                      (12)

#endif
```

Listing 5.1.1 iic.h header file

Line (1) defines the SDA for I^2C data

Line (2) defines the SCL for I^2C clock

Line (3) – (5) are public constants for portability

Line (6)

i2c_Start() void i2c_Start(void);	
* Function:	Generate an I2C start condition
* Argument:	none
* Return:	none
* Note:	Hardware-dependent for direct I/O operation on SDA and SCL required.

Line (7)

i2c_Stop() void i2c_Stop(void);	
* Function:	Generate an I2C stop condition
* Argument:	none
* Return:	none
* Note:	Hardware-dependent for direct I/O operation on SDA and SCL required.

Line (8)

i2c_Ack() void i2c_Ack(bit reply);	
* Function:	This function is called to generate an ACK or noACK by the master
* Argument:	'reply' ACK if an acknowledge is required, i.e. SDA pulled low
*	noACK if NO acknowledge required, i.e. SDA pulled high
* Return:	none
* Warning:	The argument ‘reply’ of type 'bit' is KEIL C specific
* Note:	ACK and noACK public constants defined under iic.h
*	Hardware-dependent for direct I/O operation on SDA and SCL required.

Line (9)

i2c_SendByte() bit i2c_SendByte(unsigned char c);	
* Function:	This function is called to write a byte to the i2c bus in MBS first. The function caller sends a byte, and in the 9th bit, waits for an ACK or noACK from the slave. Waiting for an ACK signal is NOT infinite to avoid program hang-up. Statement 'while((SDA==HIGH)&&(i++<0x80))' employed for ACK wait with timeout
* Argument:	'c' the byte to send
* Return:	'ACK' if the slave acknowledges
*	'noACK' if the slave didn't acknowledge
*	ACK and noACK public constants defined under iic.h
* WARNING :	the type 'bit' is KEIL C specific
* Note:	Hardware-dependent for direct I/O operation on SDA and SCL required.

Line (10)

i2c_RcvByte() unsigned char i2c_RcvByte(void);	
* Function:	This function is called to read a byte from the i2c bus.
* Argument:	none
* Return:	The byte read from the i2c bus
* Note:	Hardware-dependent for direct I/O operation on SDA and SCL required.

Line (11)

i2c_SeqWr() bit i2c_SeqWr(unsigned char device, unsigned char addrh, unsigned char addrl, unsigned char *s, unsigned char length);		
* Function:	This function is called to sequentially write an array/string of a certain length to an iic slave. STOP bit is not generated at the end of each byte except the last byte.	
* Argument:	'device'	iic device address, e.g. 0xA0 for 24C256 EEPROM, 0xA2 for PCF8563 RTC, etc
*	'addrh'	internal address's high byte, if any. If address is 8-bit only, this should be passed NONE defined under iic.h
*	'addrl'	internal address's low address byte, always present
*	'*s'	pointer to the array/string to write
*	'length'	length of the array or string to write
* Return:	'noACK'	if any of the iic write action returns a bus collision error
*	'ACK'	if all of the iic write action successful
* WARNING :	the type 'bit' is KEIL C specific	
Example: * s[0] = 0x00; s[1] = 0x10; * i2c_SeqWr(0xA2, NONE, 0x00, &s[0], 2); //This is to initialize PCF8563 rtc.		

Line (12)

i2c_SeqRd()
bit i2c_SeqRd(unsigned char device, unsigned char addrh, unsigned char addrl, unsigned char *s, unsigned char length);
* Function: This function is called to sequentially read an array/string of a certain length from an iic slave. * Argument: 'device' iic device address, e.g. 0xA0 for 24C256 EEPROM, 0xA2 for PCF8563 RTC etc * 'addrh' internal address's high byte, if any. If address is 8-bit only, * this should be passed NONE defined under iic.h * 'addrl' internal address's low address byte. * '*s' pointer to the array/string to read * 'length' length of the array or string to write * Return: 'noACK' if any of the iic write action returns a bus collision error * 'ACK' if all of the iic write action successful * WARNING: the type 'bit' is KEIL C specific. Array/string passed to *s should be large enough to hold the total data length.
Example: uchar s[3]; i2c_SeqRd(0xA2, NONE, 0x02, &s[0], 3); to read the sec:min:hour from PCF8563

Let's turn to the API functions for the real time clock (RTC). There aren't a lot of functions required to read/write to a RTC. Only five API functions have been created. Listing 5.1.2 shows the pcf8563.h file.

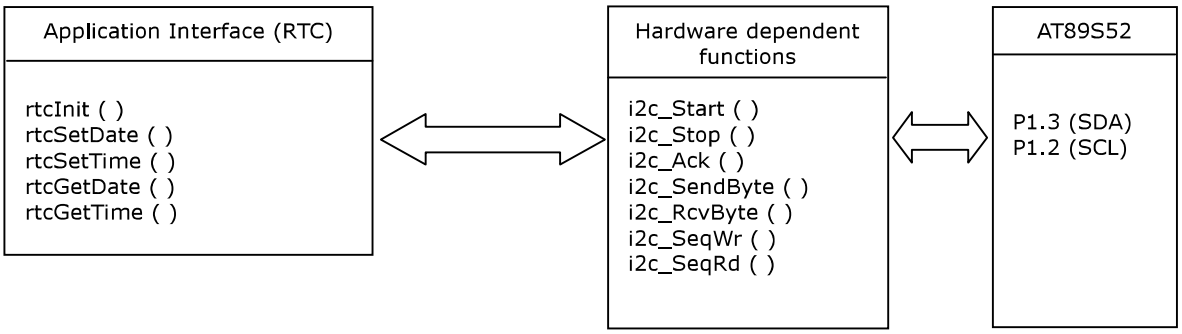

Figure 5.1.7 Application Interface for RTC

```
#ifndef _PCF8563_H_
#define _PCF8563_H_

#define VL_DETECTED       0xFF                                              (1)
#define VL_NDETECTED      1                                                 (2)

#ifndef _uchar_
#define _uchar_
typedef unsigned char uchar;                                                (3)
#endif

void   rtcInit(void);                                                       (4)
void   rtcSetDate(uchar day, uchar dow, uchar month, uchar year);           (5)
void   rtcSetTime(uchar sec, uchar min, uchar hour);                        (6)
void   rtcGetDate(uchar *date, uchar *dow, uchar *month, uchar *year);      (7)
uchar rtcGetTime(uchar *sec, uchar *min, uchar *hour);                      (8)

#endif
```

Listing 5.1.2 pcf8563.h header file

Line (1) & (2) define the VL constants for voltage-low detector flag. PCF8563 has an on-chip voltage-low detector. When V_{DD} drops below V_{low}, bit VL in the seconds register is set to indicate that the integrity of the clock information is no longer guaranteed. The VL flag can only be cleared by software. Bit VL is intended to detect the situation when V_{DD} is decreasing slowly, for example under battery operation. Should V_{DD} reach V_{low} before power is re-asserted then bit VL will be set. This will indicate that the time may be corrupted.

We make use of these constants in the function rtcGetTime( ), see line (8)

Line (3) is a type definition for unsigned char. This one is optional. It is alright to use unsigned char instead of uchar. The purpose is to shorten typing only.

Line (4) declares the function prototype for RTC initialization. There are two control registers at the internal addresses 00h and 01h. Please consult PCF8563 data sheet for details (extract shown in Figure 5.1.8). The purpose of this function rtcInit(void) is to configure the chip RTC features as below:

Control/status 1: TEST1 bit=0 (normal mode),
STOP bit=0 (RTC source clock runs)
TESTC bit=0 (normal op)
Control/status 2: AIE bit =0 (alarm interrupt disabled)
TIE bit =0 (timer interrupt disabled)

Address	Register name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00H	control/status 1	TEST1	0	STOP	0	TESTC	0	0	0
01H	control/status 2	0	0	0	TI/TP	AF	TF	AIE	TIE
0DH	CLKOUT control	FE	x	x	x	x	x	FD1	FD0
0EH	timer control	TE	x	x	x	x	x	TD1	TD0
0FH	timer	<timer countdown value>							

Figure 5.1.8

Line (5) sets the day, weekday, month, and year by writing to registers at 05h to 08h as shown in Figure 5.1.9.

Bit positions labelled as x are not implemented.

Address	Register name	BCD format tens nibble				BCD format units nibble			
		Bit 7 2^3	Bit 6 2^2	Bit 5 2^1	Bit 4 2^0	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0
02H	seconds	VL	<seconds 00 to 59 coded in BCD>						
03H	minutes	x	<minutes 00 to 59 coded in BCD>						
04H	hours	x	x	<hours 00 to 23 coded in BCD>					
05H	days	x	x	<days 01 to 31 coded in BCD>					
06H	weekdays	x	x	x	x	x	<weekdays 0 to 6>		
07H	months/century	C	x	x	<months 01 to 12 coded in BCD>				
08H	years	<years 00 to 99 coded in BCD>							
09H	minute alarm	AE	<minute alarm 00 to 59 coded in BCD>						
0AH	hour alarm	AE	x	<hour alarm 00 to 23 coded in BCD>					
0BH	day alarm	AE	x	<day alarm 01 to 31 coded in BCD>					
0CH	weekday alarm	AE	x	x	x	x	<weekday alarm 0 to 6>		

Figure 5.1.9

For example, we want to set the date for 18th June 2006, Sunday, and then we need to call rtcSetDate(0x18, 0, 0x06, 0x06)

uchar day = 0x18, in BCD format 18th is formatted as the hex number 0x18, with the high nibble being 1, lower nibble being 8. Suppose we want to set day to be 21st, we need making day=0x21.

uchar dow = 0 (Sunday=0, Monday=1, Tuesday=2, Wednesday=3,..., Saturday=6).

uchar month = 0x06, in BCD format again. Therefore, June is formatted as 0x06. For December, month = 0x12, etc. Bit7 'C' of the months/century register indicates century for year 19xx (bit7=1), and year 20xx (bit7=0).

uchar year = 0x06 for year 2006. This one in BCD format. Therefore, year 2012 would be formatted as 0x12.

Line (6) sets time by writing to registers at 02h, 03h, and 04h.

For example, if we want to set the current time to 23:59:30, we need to call rtcSetTime(uchar sec, uchar min, uchar hour) by substitution of parameters as:

sec = 0x30
min = 0x59
hour = 0x23, all in BCD format.

Line (7) declares prototype for date-read function.
Line (8) declares prototype for time-read function. It also returns the VL flag (voltage-low detector flag) embedded in the seconds register at 02h for detection of when V_{DD} drops below V_{low}.

Example: A primitive example can be found at cd:\src\chp5\src5_1\pcf8563_demo1.uv2. This program is just to show PCF8563 work. There are many features like alarm and timer features not utilized yet.

Again, Docklight is used here for program debug. Visualization of the current time is possible by making the function call:

printf("hour:%bu min:%bu sec:%bu", hour,min,sec); //send hour,min,sec to PC

Similarly, current date is sent to Docklight via the following function call:

printf(" date:%bu Dow:%bu month:%bu year:200%bu\n", date,dow,month,year); //send date,dow,month,year to PC, formatted by printf function.

Listing 5.1.3 shows the full program.

```
#include <REGX52.h>
#include "pcf8563.h"                                                    (1)
#include "delay.h"                                                      (2)
#include <stdio.h>                                                      (3)


void main(void)
{
        unsigned char hour, min, sec, date, dow, month, year;           (4)

        init_uart();                                                    (5)
        rtcInit();                                                      (6)
        rtcSetTime(0x30,0x59,0x23);                                     (7)
        rtcSetDate(0x18, 0, 0x06, 0x06);                                (8)

for(;;)
{
rtcGetTime(&sec, &min, &hour);                                          (9)
printf("hour:%bu min:%bu sec:%bu", hour,min,sec);                       (10)
rtcGetDate(&date, &dow, &month, &year);                                 (11)
printf(" date:%bu Dow:%bu month:%bu year:200%bu\n", date,dow,month,year);  (12)
DelayMs(500);                                                           (13)
}

}
```

Listing 5.1.3 PCF8563 demo program pcf8563_demo1.c

Line (1) includes the pcf8563.h header file. That is why it is legal to call rtcGetTime(), rtcSetTime() in our main program.

Line (2) includes software delay functions, like previous examples
Line (3) includes standard library stdio.h provided by Keil C for printf() function
Line (4) declares all variables holding current date/time.
Line (5) initializes UART for RS232 program debug
Line (6) initializes the real time clock
Line (7) & (8) reconcile previous statements concerning date/time adjustment functions. The purpose of these lines is to set the current date/time to

18th June 2006, Sunday
23:59:30

We only call these codes once in our program; else, we will repeat date/time set-up. A loophole in this program is that, every time we reset the system, the date/time is set to 18th June '06, 23:59:30! If we want to deploy a real-life application, we probably need to design a user interface for date/time adjustment upon certain menu entry. Alternatively, we may comment lines (7) & (8), and re-program the chip. Since there is a backup power from the super-cap (C16), the real time data will be preserved even on main power removal.

Line (9) simply reads the current time by calling rtcGetTime(). The result is formatted and printed to PC via Line (10). Similarly, Line (11) reads the date and sends it to PC (line 12) for debug.

Launch Docklight to view the result. Figure 5.1.10 shows a screenshot of Docklight. Now we see the clock is alive! Notice the real time jumps from

23:59:59, 18th June 2006, Sunday to 00:00:00, 19th June 2006, Monday

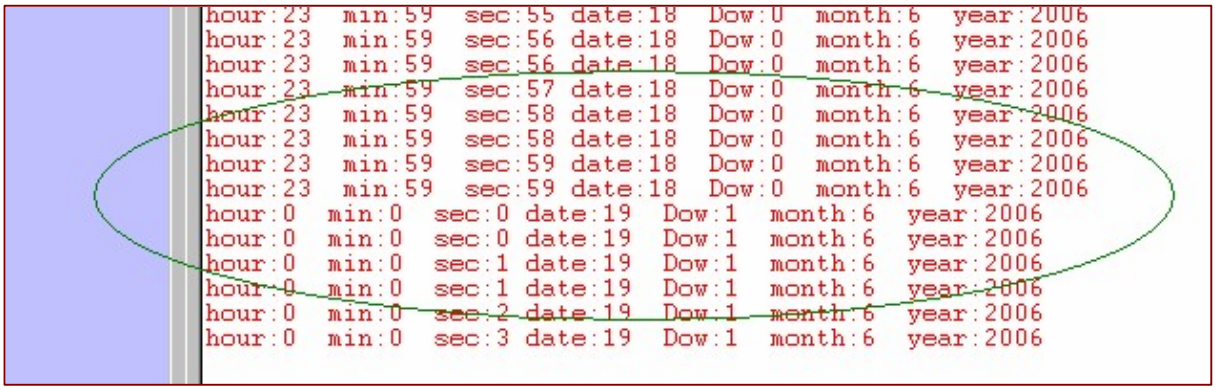

Figure 5.1.10 Docklight screenshot for pcf8563_demo1.Uv2

⇒ Chapter 6