

City University of Hong Kong
Department of Electronic Engineering
Laboratory Manual for EE3220
Academic Year 2018-19
Semester A

Venue: DS Laboratory, P1800

Laboratory Schedule:

	L01	L02
Week	(Tuesday 9:00 am)	(Wednesday 3:00 pm)
8	23 Oct. 2018	24 Oct. 2018
9	30 Oct. 2018	31 Oct. 2018
10	6 Nov. 2018	7 Nov. 2018
11	13 Nov. 2018	14 Nov. 2018
12	20 Nov. 2018	21 Nov. 2018

Grouping: 2 students in a group

Pre-Laboratory:

There are 3 tasks for the Laboratory, 1 week for Task 1, 3 weeks for Task 2 and 1 week for Task 3. However, you may continue to work on the following tasks if you complete the earlier one.

You are required to study the material and determine all the registers setting, and write the program, before coming to the laboratory since you are coming to the laboratory to test your program, NOT starting to write the program during the lab session.

You are required to submit your pre-laboratory work to your laboratory supervisor.

Log Book Requirement:

The Log Book is necessary for recoding the work and results throughout the course of carrying out the experiment. Thus it is important to present it in a clear, readable and traceable manner, so that one can easily retrieve the observation and results and to produce a Laboratory Report from it.

- Each individual student must keep his/her own Log Book
- Do not copy the procedures to the Log Book
- Make a scan copy of your Pre-Lab and stick onto your Log Book
- Follow the instructions to record the results in the Log Book
- Draw the flowchart of the programs that you have written for the experiment in your Log Book

Laboratory Report:

Each group is required to submit ONE report together with individual Log Book.

City University of Hong Kong
Department of Electronic Engineering

Implementation of Real-time Clock

and

Memory Timing Design and Memory Controller Programming

OBJECTIVES

The objectives of the project are

- To get familiar with the use of Integrated Development Environment (IDE) for programming development, compiling, linking, simulation, debugging and program download to hardware system.
- To develop programming techniques for an ARM based platform.
- To learn the control of the PLL (Phase Lock Clock) for setting operating clock frequency.
- To learn the principles of serial communication and speed control of the UART port.
- Study the technique for programming the character LCD display.
- Study the programming technique for implementing a real-time clock.
- Study the memory controller setting for SRAM timing design

EQUIPMENT

1. Embest S3CEV40 EVB ARM Board
2. Microprocessor Applications Module (AM)
3. IBM PC with EmbestIDE
4. One 9V 1A main power adaptors (part no. 961005AS or equivalent)
5. Embest UNetICE for ARM
6. J-TAG cable
7. Serial Cable
8. SRAM Module

REFERENCES

- Hardware platform user guide: S3CEV40_UserGuide.pdf
- Hardware platform circuit: S3CEV40 Schematic.pdf
- S3C44B0X User's Manual: um_s3c44box.doc (stored in c:\EmbestIDE\Doc)
- Character LCD datasheet: character_LCD.pdf
- Maxim Dallas DS2016 Static Ram datasheet: DS2016.pdf

BACKGROUND

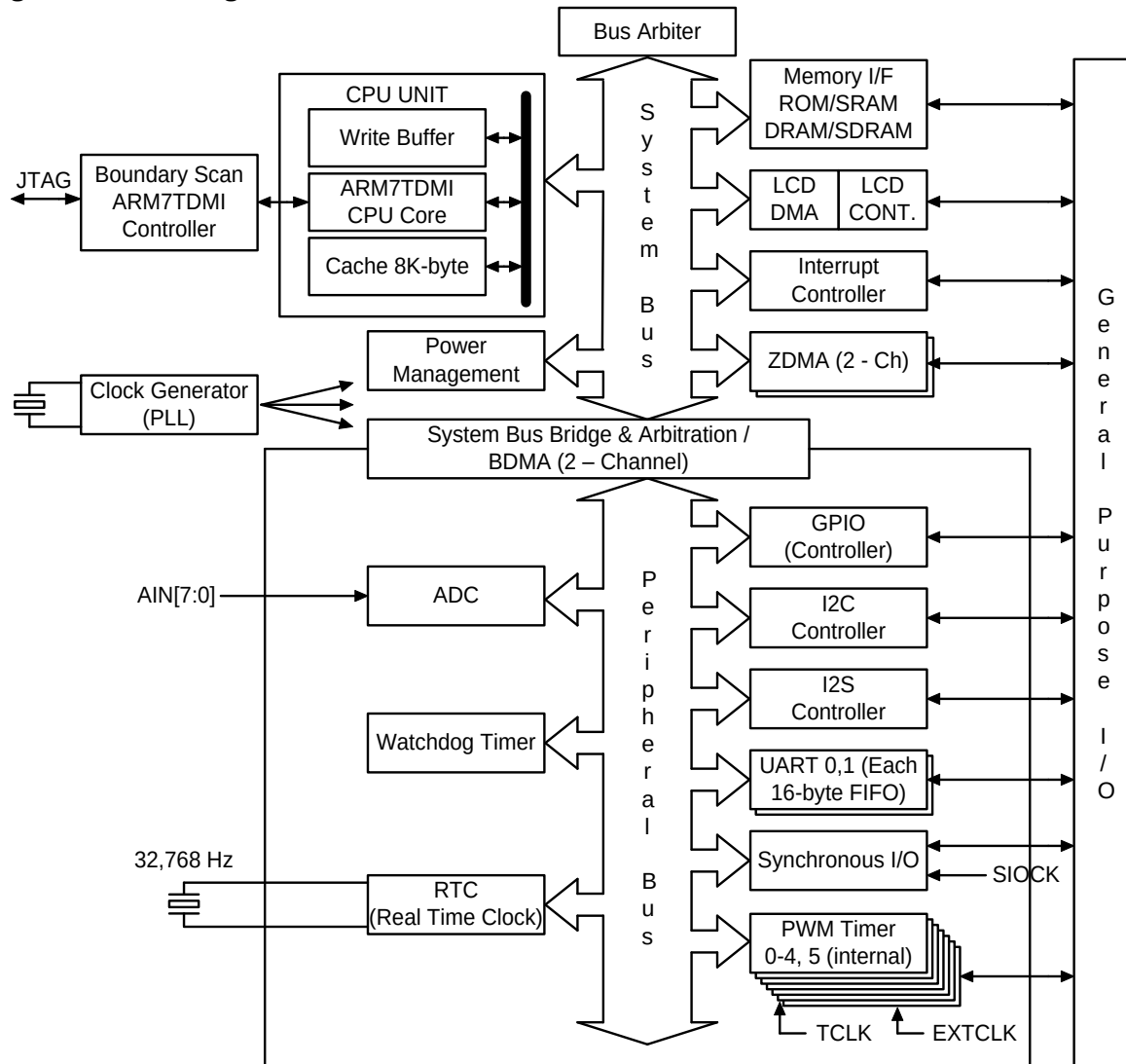
The ARM processor of the hardware platform is the Samsung ARM7 based S3C44B0X 16/32-bit RISC microprocessor. The microprocessor is designed to provide a cost-effective and high performance micro-controller solution for hand-held devices and general applications. To reduce total system cost, S3C44B0X also provides the following: 8KB cache, optional internal SRAM, LCD controller, 2-channel UART with handshake, 4-channel DMA, System manager (chip select logic, FP/EDO/SDRAM controller), 5-channel timers with PWM, I/O ports, RTC, 8-channel 10-bit ADC, IIC-BUS interface, IIS-BUS interface, Synchronous SIO interface and PLL for clock.

The S3C44B0X was developed using a ARM7TDMI core, 0.25 um CMOS standard cells, and a memory compiler. Its low-power, simple, elegant and fully static design is particularly suitable for cost-sensitive and power sensitive applications. Also S3C44B0X adopts a new bus architecture, SAMBA II (SAMSUNG ARM CPU embedded Microcontroller Bus Architecture).

An outstanding feature of the S3C44B0X is its CPU core, a 16/32-bit ARM7TDMI RISC processor (66MHz) designed by Advanced RISC Machines, Ltd. The architectural enhancements of ARM7TDMI include the Thumb decompressor, an on-chip ICE breaker debug support, and a 32-bit hardware multiplier.

By providing a complete set of common system peripherals, the S3C44B0X minimizes overall system costs and eliminates the need to configure additional components. The integrated on-chip functions that are described in this document are as follows:

- 2.5V Static ARM7TDMI CPU core with 8KB cache. (SAMBA II bus architecture up to 66MHz)
- External memory controller. (FP/EDO/SDRAM Control, Chip Select logic)
- LCD controller (up to 256 color CSTN) with 1-ch LCD-dedicated DMA.
- 2-ch general DMAs / 2-ch peripheral DMAs with external request pins
- 2-ch UART with handshake(IrDA1.0, 16-byte FIFO) / 1-ch SIO
- 1-ch multi-master IIC-BUS controller
- 1-ch IIS-BUS controller
- 5-ch PWM timers & 1-ch internal timer
- Watch Dog Timer
- 71 general purpose I/O ports / 8-ch external interrupt source
- Power control: Normal, Slow, Idle, and Stop mode
- 8-ch 10-bit ADC.
- RTC with calendar function.
- On-chip clock generator with PLL.

Figure 1: Block Diagram of S3C44B0X

Some useful port register addresses are shown below:

Port	Address	Initial value
PCONC	0x01D20010	0xAAAAAAAA
PDATC	0x01D20014	NA
PUPC	0x01D20018	0x0
PCOND	0x01D2001C	0x0000
PDATD	0x01D20020	NA
PUPD	0x01D20024	0x0
PCONE	0x01D20028	0x00
PDATE	0x01D2002C	NA
PUPE	0x01D20030	0x00
PCONF	0x01D20034	0x0000
PDATF	0x01D20038	NA
PUPF	0x01D2003C	0x000
PCONG	0x01D20040	0x0
PDATG	0x01D20044	NA
PUPG	0x01D20048	0x0

Hardware Boards Layout

The layout of the S3CEV40 EVB Board is shown in Figure 2a and the layout of the interfacing board between the S3CEV40 EVB and application boards is shown in Figure 2b below:

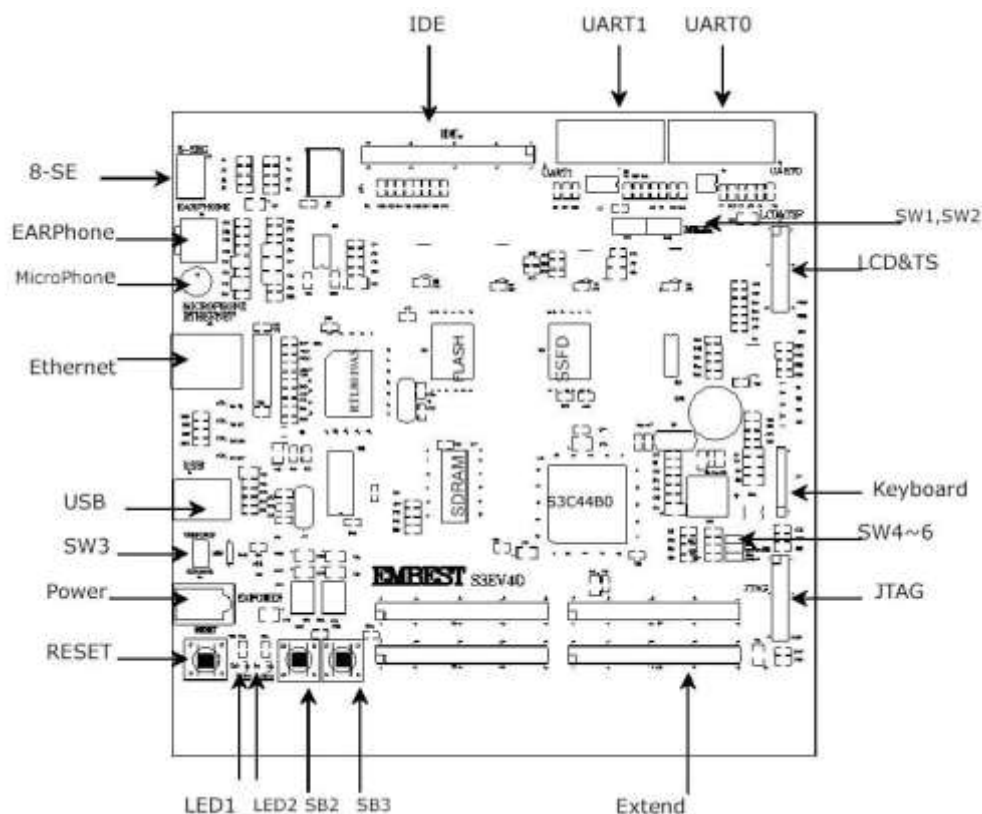


Figure 2a : Layout of S3CEV40 EVB Board

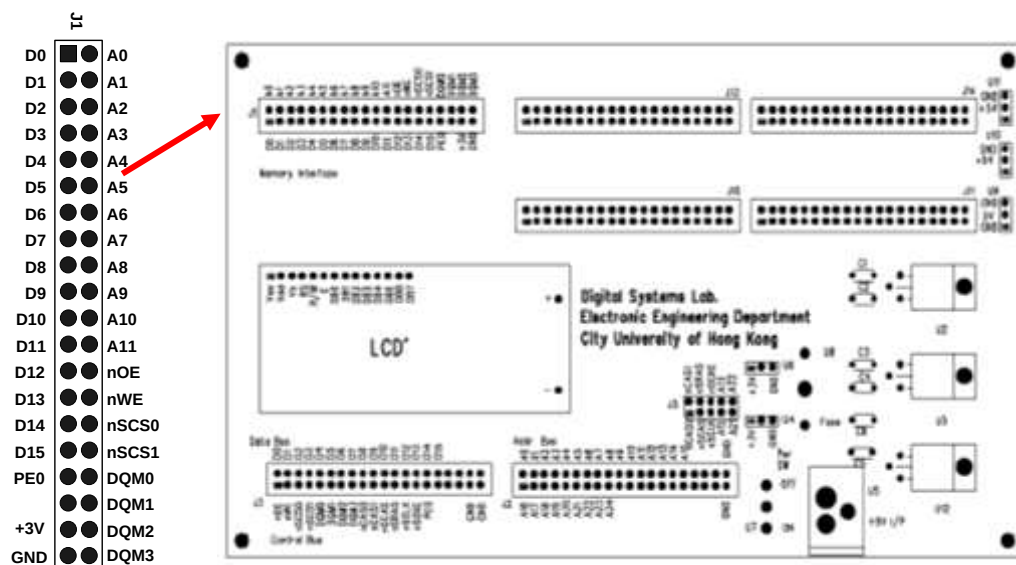


Figure 2a : ARM Interface Board Layout

S3C44B0X PLL CLOCK FREQUENCY CONTROL

The operational speed of the CPU is not necessary equivalent to the clock source connected to it. Rather, it has an internal PLL (Phase-Locked-Loop) circuit to generate high frequency from a low frequency source. This section is intended to experimenting how does the PLL affect the operational speed of the CPU.

PLL

The PLL within the clock generator is the circuit which synchronizes an output signal with a reference input signal in frequency and phase. In this application, it includes the following basic blocks: the VCO (Voltage Controlled Oscillator) to generate the output frequency proportional to input DC voltage, the divider P to divide the input frequency (F_{in}) by p , the divider M to divide the VCO output frequency by m which is input to PFD (Phase Frequency Detector), the divider S to divide the VCO output frequency by s which is F_{pllo} (the output frequency from PLL block), the phase difference detector, charge pump, and loop filter. The output clock frequency F_{pllo} is related to the reference input clock frequency F_{in} by the following equation:

$$F_{pllo} = (m \times F_{in}) / (p \times 2^s)$$

$$m = MDIV + 8, \quad p = PDIV + 2, \quad s = SDIV$$

NOTE: F_{pllo} must be greater than 20Mhz and less than 66Mhz.

Example of setting PLLCON:

If $F_{in}=14.318\text{MHz}$ and $F_{out}=60\text{MHz}$, the calculated value is as follows;
MDIV=59, PDIV=6 and SDIV=1

$$F_{pllo} = \frac{(59 + 8) \times 14.318}{(6 + 2) \times 2^1} = 59.96 \text{ MHz}$$

PLL VALUE SELECTION GUIDE

1. $F_{pllo} \times 2^s$ has to be less than 170 MHz.
2. S should be as great as possible.
3. (F_{in} / p) is recommended to be 1MHz or above. But, $(F_{in} / p) < 2\text{Mhz}$.

Initial values for PLLCON, CLKCON and LOCKTIME:

Register	Address	R/W	Description	Reset Value
PLLCON	0x01D80000	R/W	PLL configuration Register	0x38080

PLLCON	Bit	Description	Initial State
MDIV	[19:12]	Main divider control	0x38
PDIV	[9:4]	Pre-divider control	0x08
SDIV	[1:0]	Post divider control	0x0

CLOCK CONTROL REGISTER (CLKCON)

Register	Address	R/W	Description	Reset Value
CLKCON	0x01D80004	R/W	Clock generator control Register	0x7ff8

LOCK TIME COUNT REGISTER (LOCKTIME)

Register	Address	R/W	Description	Reset Value
LOCKTIME	0x01D8000C	R/W	PLL lock time count register	0xffff

LOCKTIME	Bit	Description	Reset Value
LTIME CNT	[11:0]	PLL lock time count value	0xffff

IMPORTANT INFORMATION:

When the system is powered on, the operating frequency is according to either the crystal oscillator or Ext. clock source (Fin is 8 MHz for our development board). Although there is a reset value assigned to PLLCON, it will NOT be effective since the PLL clock circuit require a short period of time (LOCKTIME) to stabilize its output frequency.

Even if the user wants to maintain the default value of PLLCON register, the user should write the same value into PLLCON register.

S3C44B0X UART

S3C44B0X UART (Universal Asynchronous Receiver and Transmitter) unit equips with two independent asynchronous serial I/O ports, of which they can operate in interrupt-based or DMA-based mode. Each UART contains a baud-rate generator, transmitter, receiver and control unit. The baud-rate of the UART is programmable with bit rates of up to 115.2K bps. The character frame format is also programmable: one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data width and parity checking.

Each UART channel contains two 16-byte FIFOs and data shifters for receive and transmit data. Data, which is to be transmitted, is written to FIFO and then copied to the transmit shifter. It is then shifted out by the transmit data pin (TxDn). The received data is shifted from the received data pin (RxDn), and then copied to FIFO from the shifter.

Data Transmission

The data frame for transmission is programmable by the line control register (UCONn). Data to be transmitted is written into the Tx FIFO (Tx holding register in the case of Non-FIFO mode).

Data Reception

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits in the line control register (UCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

Receive time-out condition occurs when it does not receive data during the 3 word time and the Rx FIFO is not empty in the FIFO mode.

Baud-Rate Generation

The baud rate divisor register (UBRDIVn) controls the baud rate. The serial Tx/Rx clock rate (baud rate) is calculated as follows:

$$UBRDIVn = (\text{round_off})(MCLK / (bps \times 16)) - 1$$

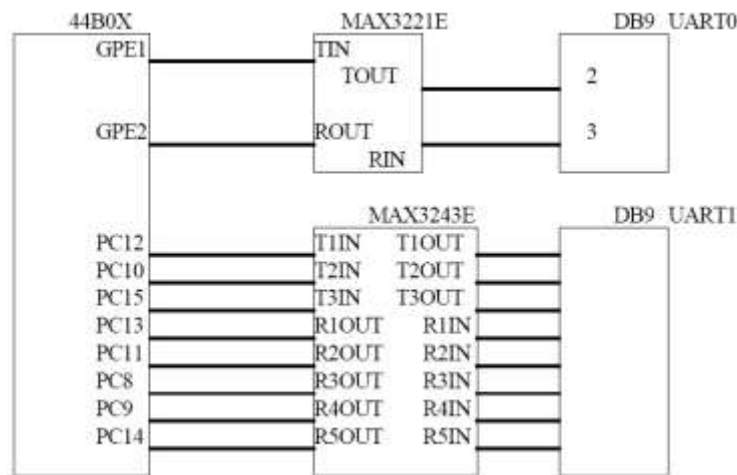
The divisor should be from 1 to (216-1). For example, if the baud-rate is 115200 bps and MCLK is 40 MHz, UBRDIVn is:

$$\begin{aligned} UBRDIVn &= (\text{int})(40000000 / (115200 \times 16) + 0.5) - 1 \\ &= (\text{int})(21.7 + 0.5) - 1 \\ &= 22 - 1 = 21 \end{aligned}$$

For programming the UART correctly, please refer to Chapter 10 UART of the S3C440B0X Data manual for details of register setting.

UART Interface Circuit

The serial port circuit of the S3CEV40 board is shown in figure below. The board provides two serial ports DB9. The UART1 is the main serial port that can be connected to PC or MODEM. Because 44B0X didn't provide standard I/O signals such as DCD, DTE, DSR, RIC, etc. the general I/O port signals are used. UART0 has only 2 lines RXD and TXD that can be used only for simple data transmitting and receiving without using handshaking signal. The full UART1 connects to MAX3243E voltage converter. The simple UART0 connects to MAX3221 voltage converter.



Serial port circuit signals

Port E Configuration Register setting:

PCONE	Bit	Description
PE8	[17:16]	00 = Reserved (ENDIAN) 01 = Output 10 = CODECLK 11 = Reserved PE8 can be used as ENDIAN only during the reset cycle.
PE7	[14:15]	00 = Input 01 = Output 10 = TOUT4 11 = VD7
PE6	[12:13]	00 = Input 01 = Output 10 = TOUT3 11 = VD6
PE5	[10:11]	00 = Input 01 = Output 10 = TOUT2 11 = TCLK in
PE4	[8:9]	00 = Input 01 = Output 10 = TOUT1 11 = TCLK in
PE3	[6:7]	00 = Input 01 = Output 10 = TOUT0 11 = Reserved
PE2	[4:5]	00 = Input 01 = Output 10 = Rx D0 11 = Reserved
PE1	[2:3]	00 = Input 01 = Output 10 = Tx D0 11 = Reserved
PE0	[1:0]	00 = Input 01 = Output 10 = FpIlo out 11 = Fout out

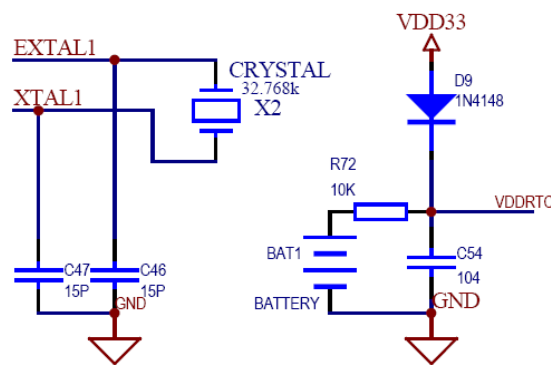
S3C44B0X RTC

The RTC unit is a specific module (or separate IC) that can provide date/time, data storage, and other functions. The S3C44B0X is equipped with RTC and its function diagram is shown in the following figure. The RTC unit is operated by the onboard backup battery while the system power is off. The RTC can transmit 8-bit data to CPU as BCD (Binary Coded Decimal) values using the STRB/LDRB ARM operation. The data include second, minute, hour, date, day, month, and year. The RTC unit works with an external 32.768 KHz crystal and also can perform the alarm function.

The features of the RTC are listed as follows:

- BCD number: second, minute, hour, date, day, month, year
- Leap year generator
- Alarm function: alarm interrupt or wake-up from power down mode.
- Year 2000 problem is removed.
- Independent power pin (VDDRTC)
- Supports millisecond tick time interrupt for RTOS kernel time tick.
- Round reset function

The real-time peripheral circuit is shown in the following figure:



Real-Time Clock Peripheral Circuit

RTC Read/Write Registers

Bit 0 of the RTCCON register **must be set** in order to read and write the register in RTC block. To display the sec., min., hour, date, month, and year, the CPU should read the data in BCDSEC, BCDMIN, BCDHOUR, BCDDAY, BCDDATE, BCDMON, and BCDYEAR registers, respectively, in the RTC block. However, a one second deviation may exist because multiple registers are read. For example, suppose that the user reads the registers from BCDYEAR to BCDMIN, and the results are 1959(Year), 12(Month), 31(Date), 23(Hour) and 59(Minute). If the user reads the BCDSEC register and the result is a value from 1 to 59(Second), there is no problem, but, if the result is 0 sec., the year, month, date, hour, and minute may be changed to 1960(Year), 1(Month), 1(Date), 0(Hour) and 0(Minute) because of the one second deviation that was mentioned. In this case (when BCDSEC is zero), the user should re-read from BCDYEAR to BCDSEC.

TASK 1: CPU MASTER CLOCK AND UART PROGRAMMING (ONE WEEK)

The S3CEV40 system provides a folder “source” with path C:\EmbestIDE\Examples\source\.. This folder contains a number of examples showing how to operate the board.

Basic Project Setting: Follow the Simplified Embest IDE User guide

Procedures:

1. **Connect** the **UART_0 port** of the system board to the COM port of the PC and start a hyper-terminal on the Microsoft Window with baud rate 115200 bits per second, 8 data bits, none parity, 1 stop bits, none flow control.
2. **Duplicate** the folder Uart_Test and **rename** it with a name of your own desire (**note the directory path of this folder must not consist of space**).
3. **Replace** the original main.c file by the following main.c file (this program reads the key pressed onto the PC's keyboard from the UART0 port and returns it to the hyper-terminal via the UART port).
4. **Change the Project Setting** and then build the application program. Refer to the Simplified Embest_IDE User Guide for the basis project settings.

Remote device connection: choose UNetICE with USB communication type.

For programming in C code, additional settings are listed as follows (may vary dependent on the program and directory setup):

Compiler – Include Directoried setting:- C:\EmbestIDE\Examples\source\common

Assembler – Include Directoried setting:- C:\EmbestIDE\Examples\source\common

Linker – Image Entry Options:- 44binit.o

Linker – Include Object and Library Modules setting:- -lc

Linker – Include Object and Library Modules setting:- -lgcc

Linker – Add Library Searching Path:-

C:\EmbestIDE\Build\xgcc-arm-elf\arm-elf\lib

Linker – Add Library Searching Path:-

C:\EmbestIDE\Build\xgcc-arm-elf\lib\gcc-lib\arm-elf\3.0.2

5. Choose Remote connect from the Debug pull-down Menu.
6. Choose Download from the Debug pull-down Menu.
7. Choose Go from the Debug pull-down Menu to execute the program.
8. Press the keys on the keyboard and observe the screen output of the hyper-terminal. You should see the character on the screen.

```

/*****
* File      Main.c - hyper-terminal echo
* Desc      Main entry
*****/

/*--- include files ---*/
#include "44bplib.h"
#include "44b.h"

/*--- function code ---*/
char str_send[17] = "UART_ECHO_TEST >\0";
char str_error[50] = "TERMINAL OVERFLOW : 256 character max !";
char str[256];
char CR[1] = {0x0A};          /* 0x0A = Line feed */

void Main(void)
{
    char input_char;           /* user input char */
    int i;
    char *pt_str = str;

    Port_Init();               /* Initial 44BOX's I/O port */
    rI_ISPC = 0xffffffff;      /* clear all interrupt pend */
    Uart_Init(0,115200);        /* set UART0 for baud rate, etc */

    /* get user input */
    Delay(500);
    /* Terminal handler
    Uart_SendString(str_send);
    while(1)
    {
        *pt_str = Uart_Getch(); /* read character from UART0 */
        Uart_SendByte(*pt_str); /* send the character back to UART0 */
    }
}

```

The following functions are embedded in library function file 44bplib.c (in Common directory). Many other useful functions are contained in Uart.c, especially for UART related ISR.

```

Port_Init()
Uart_Init(0,115200)
Delay(500)
Uart_SendString(str_send)
Uart_Getch()
Uart_SendByte(*pt_str)

```

9. For the Uart_Init() function, add c-code to change the CPU clock frequency to 48MHz by writing to the PLLCON register with $F_{in} = 8 \text{ MHz}$.
10. **Modify** the PConE register so that the port pin PE0 provide master clock output and use an oscilloscope to confirm the CPU clock frequency. (Leave PE1 and PE2 unchanged since they are chosen as UART0 TxD and RxD; See Appendix for the original PORT setting - Port_Init(void) function.)

The modifications of 9 and 10 should replace the two red lines in Uart_Init().

11. Modify the baud rate to 19200 and change the character format to 8 bit data, odd parity and 2 stop bits. Rebuild your program and execute it to verify the communication with PC. **Don't forget to change the hyper terminal setting.**

SHOW YOUR RESULTS TO LABORATORY SUPERVISOR!!!

```
void Uart_Init(int mclk,int baud)
{
    int i;
    if(mclk==0)          // replace these two lines for new
        mclk=MCLK;      // for mclk and set PE0 for mclk output;
                        // PE1 and PE2 for TxD and RxD output

    rUFCON0=0x0;         //FIFO disable
    rUFCON1=0x0;
    rUMCON0=0x0;
    rUMCON1=0x0;
//UART0
    rULCON0=0x3;         //Normal,No parity,1 stop,8 bit
    rUCON0=0x245;        //rx=edge,tx=level,disable timeout int.,enable rx
error int.,normal,interrupt or polling
    rUBRDIV0=( (int)(mclk/16./baud + 0.5) -1 );
//UART1
    rULCON1=0x3;
    rUCON1=0x245;
    rUBRDIV1=( (int)(mclk/16./baud + 0.5) -1 );

    for(i=0;i<100;i++);
}
```

12. Replace the main.c file by the following file. This file not only echoes the key pressed back to the hyper-terminal, but also stores the characters as a string. If the CR (Enter) key is pressed, the whole string will be sent to the hyper-terminal.

Observe and understand the program flow.

```
/*----- include files -----*/
/*****
* File      Main.c - Hyper-terminal echo and send string back to terminal
*****/
```

```

#include "44blib.h"
#include "44b.h"

/*--- function code ---*/
char str_send[17] = "UART_ECHO_TEST >\0";
char str_error[50] = "TERMINAL_OVERFLOW : 256 character max !";
char str[256];
char CR[1] = {0x0A};          /* 0x0A = Line feed */

/*****
* name:           Main
* func:           c code entry
* para:           none
* ret:            none
*****/
void Main(void)
{
    char input_char;          /* user input char */
    int i;
    char *pt_str = str;

    Port_Init();              /* Initial 44BOX's I/O port */
    rI_ISPC = 0xffffffff;     /* clear all interrupt pend */
    Uart_Init(0,115200);       /* Initial Serial port 1 */

    /* get user input */
    Delay(500);
    /* Terminal handler
    Uart_SendString(str_send);
    while(1)
    {
        *pt_str = Uart_Getch();
        Uart_SendByte(*pt_str);
        if (*pt_str == 0x0D)          /* 0x0D = CR */
        {
            if (pt_str != str)
            {
                /* Send str_send
                Uart_SendByte(CR[0]);
                /* Send received string
                pt_str = str;
                while (*pt_str != 0x0D)
                {
                    Uart_SendByte(*pt_str);
                    pt_str++;
                }
                pt_str = str;
            }
            Uart_SendByte(CR[0]);
            Uart_SendString(str_send);
        }
        else
            pt_str++;
    }
}

```

TASK 2: REAL-TIME CLOCK IMPLEMENTATION (3 WEEKS)

The Ports D and G of the S3CEV40 board are interfaced with a 2×16 character LCD display with connection shown in the following table.

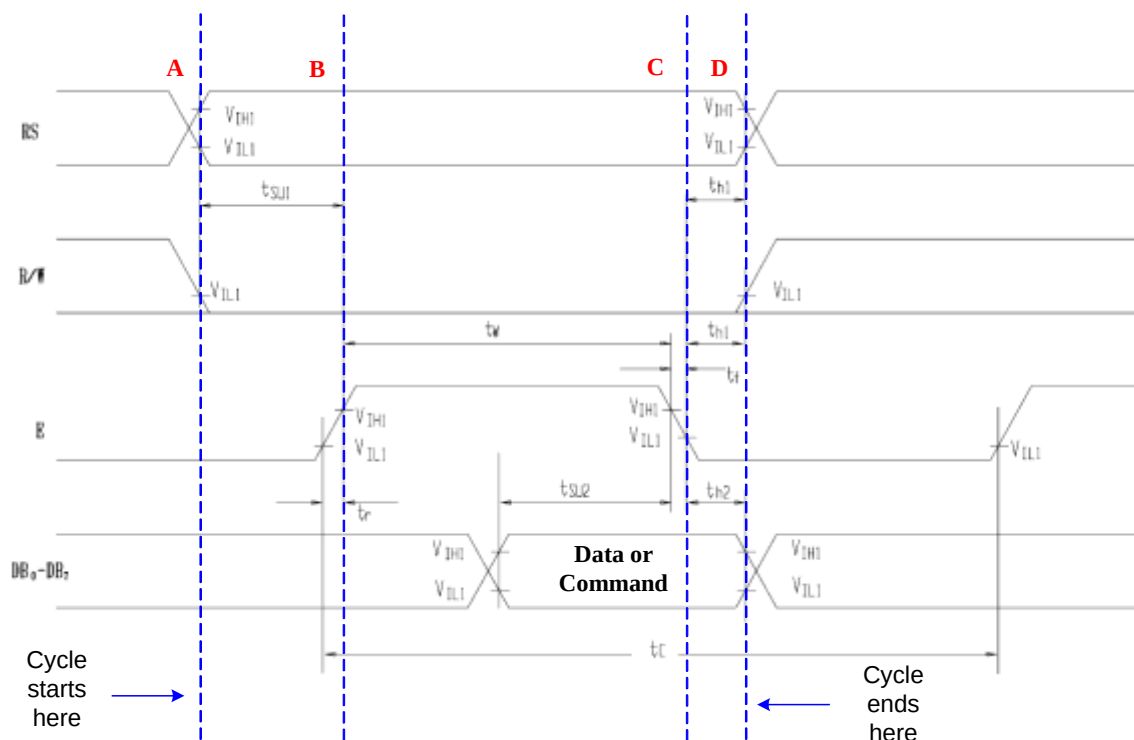
Pin	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LCD Pin Function	Vss	Vdd	Vo	RS	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7
Port Name	0V	5V	0V	PG0	PG1	PG2	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7

LCD Module Pin Assignment

In order to operate the LCD module correctly, a series of command should be sent to the LCD for proper initialization (refer to page 5 of the data sheet for list of command; control signal **RS=L** is for sending command and **RS=H** is for data READ/WRTIE).

Sending the control signal, command and data to the LCD must fulfill the timing requirement as shown below. **Each cycle can only send one command or data.** The control signals change at 4 time points (A, B, C and D) and the delay time gap between these time points can be realized by executing a simple delay function, for example the `DelayMs ( )` function. (**Note the signal level of RS signifies DB₀ ~ DB₇ to be either command or data.**)

Characteristic	Symbol	Min.	Typ.	Max.	Unit	Test pin
E cycle time	t_c	500	---	---	ns	E
E rise time	t_r	---	---	25	ns	E
E fall time	t_f	---	---	25	ns	E
E pulse width (High, Low)	t_w	220	---	---	ns	E
R/W and RS set-up time	t_{SU1}	40	---	---	ns	R/W, RS
R/W and RS hold time	t_{h1}	10	---	---	ns	R/W, RS
Data set-up time	t_{SU2}	60	---	---	ns	DB ₀ ~ DB ₇
Data hold time	t_{h2}	10	---	---	ns	DB ₀ ~ DB ₇



Procedures – Real Time Clock Display

1. **Write two functions** to simplify the software design: one for sending command to LCD and one for sending data to the LCD for displaying. The steps in the function **MUST** simulate the control signal sequence (points A, B, C and D) by writing “1” or “0” to E, RS and R/W in appropriate sequence according to the timing diagram.

Note: Writing a “1” to an output port data register bit will generate a control signal of “HIGH”, likewise, a “0” will generate a control signal of “LOW”.

Inserting a delay() function in between each port pin signal changing function will generate the required delay as specified in the timing diagram.

Each command/data is encapsulated in one of these two functions. Sending multiple commands or data to the LDC must be done one by one.

2. By referring to the initialization procedure shown below (p.6 of data sheet), use the send command function written above to write a function to initialize the LCD with 8-bit display, 2-line display, 5 x 10 dots and proper entry.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function Set (Specify the number of display lines and character font.) The number of display lines and character font cannot be changed afterwards.
0	0	0	0	1	1	N	F	*	*	
0	0	0	0	0	0	1	0	0	0	Display Off
0	0	0	0	0	0	0	0	0	1	Display Clear
0	0	0	0	0	0	0	1	I/D	S	Entry Mode Set
?	?	?	?	?	?	?	?	?	?	Display On

Note the LCD display has an internal buffer of 24 characters for one line, but it can only display 16 characters.

3. Write a program to read the characters entered through the hyper-terminal and display them on the character LCD. The program should follow the flow as below:
 - i. Set the ports D & G; (See Appendix A for examples on port setting.)
 - ii. Initializing the LCD;
 - iii. Read character from hyper-terminal;
 - iv. Write character to LCD with appropriate position control.

(Your program should also perform backspace function for displaying on LCD.)

Figure below shows an example of the output on the character LCD.



4. Write a function to set the real-time clock by writing to registers **BCD SECOND REGISTER (BCDSEC)**, **BCD MINUTE REGISTER (BCDMIN)** and **BCD HOUR REGISTER (BCD HOUR)**. (See pages 14-8 & 14-9 of S3C44B0X data book for details.)

The names and addresses of these registers are already defined in header file **44b.h**. For example, the simplest way to set the time to 11:25:07 is:

```
rBCD HOUR = 0x11; // BCD HOUR = 0001 0001 in binary format
rBCD MIN = 0x25;  // BCD MIN  = 0010 0101 in binary format
rBCD SEC = 0x07;  // BCD SEC   = 0000 0111 in binary format
```

5. Write a function to read the time from registers BCDHOUR, BCDMIN and BCDSEC. Since the data is in BCD format, you may use bitwise operation “&” to extract each BCD value and then convert it to ASCII code for displaying in the LCD. (*Refer to the ASCII table below.*)
6. Write a `main.c` program to integrate the functions developed above to:
 - i. write the time to the RTC
 - ii. read the time from the RTC;
 - iii. display the time as shown below;
 - iv. read the time from the RTC;
 - v. update the display if time has changed (check BCDSEC value);
 - vi. repeat steps v.

Note: The “time” value in the RTC registers is in BCD format, and the characters shown in the LCD is in ASCII format. The relationship between the BCD and character is a simple offset of 0x30, e.g. BCD digit of “3” in ASCII representation is 0x33.



SHOW YOUR RESULTS TO LABORATORY SUPERVISOR!!!

Procedures – Real Time Clock Time Setting

A push button board for setting the RTC with button connection detail is given in the following table:

PUSH Button INPUT BOARD Configuration:
P1 connect to J2, P2: Power 3.3V and GND

Button Name	PG4	PG5	PG6	PG7
Button Address	Port G: Pin 4	Port G: Pin 5	Port G: Pin 6	Port G: Pin 7



7. Write functions to read to detect if these button is pressed.
8. Button functions may include: mode (set/done); hr/min/sec; + ; - ;
9. Integrate these functions to the program developed in Step 6 so that the time can be set by means of pressing these buttons.

Care must be taken when setting the time, e.g., hour must be ≤ 23 ; minute and second must be ≤ 59 .

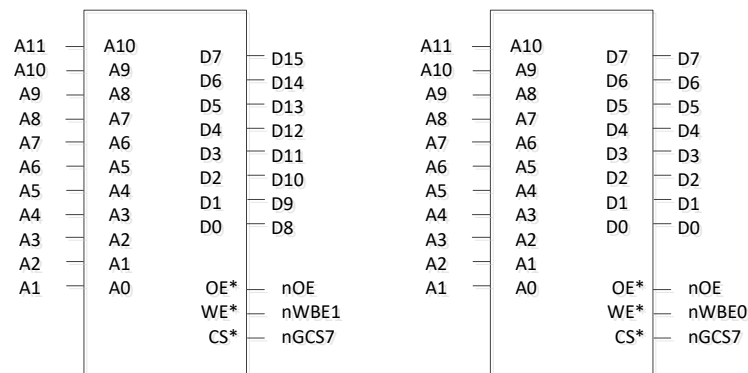
SHOW YOUR RESULTS TO LABORATORY SUPERVISOR!!!

ASCII Table

$C_3C_2C_1C_0$	000	001	010	$C_6C_5C_4$ 011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EQT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	I	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	S0	RS	.	>	N	^	n	~
1111	S1	US	/	?	O	_	o	DEL

TASK 3: MEMORY TIMING AND CONTROLLER SETTING (ONE WEEK)

A memory module with the circuit shown below is connected to the J1 of the application board of the ARM system. The module consists of two DS2016 2k x 8 SRAM. Memory Bank 6 is interfaced to on-board SDRAM and the external SRAM module is interfaced to memory bank 7.



ARM Interface Board Layout

With reference to Chapter 4 of the data book of S3C44B0X and lecture note Chapter 10, you are required to carry out the memory timing calculation and controller setting so that the CPU can successfully access the memory chips.

Procedures:

1. With reference to the data book and lecture notes of EE3220, determine your own setting of PLLCON, BANKSIZE, BWSCON and BANKCON7 registers according to the delay time specifications of the SRAM when operating in 3.3V. *(NOTE: The reset value of BANKSIZE is 0x00000000; BWSCON is 0x00000000; PLLCON is 0x00038080; and Fin = 8MHz. If you do not write to PLLCON after reset, the CPU clock frequency will be equal to Fin. The entry point of your C-program is 44binit.o, the registers PLLCON, BWSCON, BANKSIZE and BANKCON7 are predefined in the source file 44binit.s. You may redefine BWSCON and BANKSIZE in your program by only modifying memory BANK – DO NOT modify 44binit.s.)*

To begin with, use 8 MHz to design the timing parameters for memory controller registers.

(Show how to calculate the delay and the selection of parameters of the registers to Laboratory supervisor.)

2. Determine the starting and ending address of the SRAM system according to your setting. Note that the starting address of Bank7 is related to the BANKSIZE setting of Bank6 and refer to page 4-2 of data manual for detail.

3. Write a C-program (i) with your own settings of BANKSIZE, BWSCON and BANKCON7 registers written inside your program; (ii) **fill up** the SRAM memory system with all **visible alphanumeric ASCII characters**.

Here is an example code to illustrate how to read from and write to specific memory location by means of byte, hword and word size format. (you NEED to write your own code with byte size to cater for ASCII characters):

RWramc.c source code:

```

////////// RAM read/write using C language
#define RWram (*(unsigned long *)0x0c010200)
void cRWramtest(void)
{
    unsigned long * ptr = 0x0c010200;//RWram;
    unsigned short * ptrh = 0x0c010200;//RWram;
    unsigned char * ptrb = 0x0c010200;//RWram;
    char i;
    unsigned char tmpb;
    unsigned short tmpb;
    unsigned long tmpw;
    *ptr = 0xAA55AA55;
    tmpw = *ptr; /*// Read by Word.*/
    *ptr = tmpw+1; /*// Write by Word.*/
    tmpb = *ptrh; /*// Read by half Word.*/
    *ptrh = tmpb+1; /*// Write by half Word.*/
    tmpb = *ptrb; /*// Read by half Byte.*/
    *ptrb = tmpb+1; /*// Write by half Byte.*/
}

```

4. Execute your program and use the memory window of the EmbestIDE to verify the memory content.
5. Since the parameter selection of BANKCON7 is based on the memory access time provided in the data sheet and the actual delay time is shorter than the specified value, set the PLLCON to 64 MHz and determine the minimum value of Tacc in BANKCON7 so that the SRAM still function properly.

SHOW YOUR RESULTS TO LABORATORY SUPERVISOR!!!

Appendix A

Below is a program fragment, showing the port setting of the development board, extracted from 44blib.c

```
//-----PORTS----- //
// These settings are only for initializing the on-board devices. //
// And requires modifications if external circuit is used. //

void Port_Init(void)
{
//CAUTION:Follow the configuration order for setting the ports.
// 1) setting value
// 2) setting configuration register
// 3) configure pull-up resistor.

//16bit data bus configuration

// PORT A GROUP
// BIT      9      8      7      6      5      4      3      2      1      0
//          A24 A23 A22 A21 A20 A19 A18 A17 A16 A0
//          0      1      1      1      1      1      1      1      1      1
rPCONA = 0x1ff;

// PORT B GROUP
// BIT      10      9      8      7      6      5      4      3      2      1      0
//          /CS5 /CS4 /CS3 /CS2 /CS1 GPB5 GPB4 /SRAS /SCAS SCLK SCKE
//          EXT  NIC  USB  IDE  SMC  NC   NC   Sdram Sdram Sdram Sdram
//          1      1      1      1      1      0      0      1      1      1      1

rPDATB = 0x7ff;           // P9-LED1 P10-LED2
rPCONB = 0x1cf;

// PORT C GROUP
// BUSWIDTH=16
// PC 15 14 13 12 11 10 9 8
//     I  I  RXD1 TXD1 I  I  I  I
//     NC NC  Uart1 Uart1 NC NC NC NC
//     00 00 11 11 00 00 00 00

// PC 7 6 5 4 3 2 1 0
//     I  I  I  I  I  I  I  I
//     NC NC NC NC IISCLK IISDI IISDO IISLRCK
//     00 00 00 00 11 11 11 11
rPDATC = 0xff00;
rPCONC = 0x0ff0ffff;
rPUPC = 0x30ff;           //PULL UP RESISTOR should be enabled to I/O

// PORT D GROUP
// PORT D GROUP(I/O OR LCD)
// BIT 7 6 5 4 3 2 1 0
//     VF VM VLINE VCLK VD3 VD2 VD1 VD0
//     00 00 00 00 00 00 00 00
rPDATD = 0xff;
```

```

rPCOND= 0xaaaa;
rPUPD = 0x0;
// These pins must be set only after CPU's internal LCD controller is enable

// PORT E GROUP
// Bit      8      7      6      5      4      3      2      1      0
// CODECLK LED4 LED5 LED6 LED7 BEEP RXD0 TXD0 LcdDisp
//      10      01      01      01      01      01      10      10      01
rPDATE  = 0x1ff;
rPCONE  = 0x25529;
rPUPE  = 0x6;

// PORT F GROUP
// Bit      8      7      6      5      4      3      2      1      0
//      IISCLK IISDI IISDO IISLRCK Input Input Input IICSDA IICSCL
//      100    100    100    100    00    00    00    10      10
rPDATF = 0x0;
rPCONF = 0x252a;
rPUPF  = 0x0;

// PORT G GROUP
// BIT      7      6      5      4      3      2      1      0
//      INT7 INT6 INT5 INT4 INT3 INT2 INT1 INT0
//      S3   S4   S5   S6   NIC  EXT  IDE  USB
//      11    11    11    11    11    11    11    11
rPDATG = 0xff;
rPCONG = 0xffff;
rPUPG  = 0x0;           //should be enabled
rSPUCR = 0x7;           //D15-D0 pull-up disable

/* Non Cache area */
rNCACHBE0=((Non_Cache_End>>12)<<16)|(Non_Cache_Start>>12);
/* Low level default */
rEXTINT=0x0;
}

```