Matt Fleetwood

ECE 372 - 2017
Project 2

Abstract: The New Haven LCD screen can be used to interface with microprocessors. For this project, the Beaglebone Black (BBB) microprocessor was used to send a message to the LCD screen. The BBB uses C and ARM in order to program and use the screen. The processor performs an initialization for the LCD and then continuously checks whether or not a message can be sent to the screen until the message has been sent.

Methods

There are at least two or three methods for initializing and sending messages to the New Haven LCD. For instance, according to the New Haven data sheet the LCD can be initialized by pulsing the SCL and SDA lines appropriately. Since the BBB uses a controller for its I2C modules, the I2C_CON register can be sent the right control words for the correct initialization. Similarly, the two main approaches to sending a message to the screen worth mentioning for this project are polling (or status-checking) and interrupts. Direct Memory Access or DMA would be a possible third option, which is not used at all for this project. Polling is used instead to achieve the basic requirements. In this case, it means writing a message to the screen.

The solution taken here sends a message by polling, or checking status bits in the I2C1_IRQ_RAW register. This is handled by the no_int_send(unsigned int dcount) function. The number of characters in the message is used as an input for this function. Inside the function, this number is put into the I2C1_CNT register so that after the message has been sent, the DCOUNT and ARDY bits get set to 1 as “done” flags. The BBB runs at a higher clock speed than the LCD is able to. This means that small delay loops (or a timer interrupt) must be used in between writing to the LCD.

Once dcount is set, a delay occurs, and then the function determines whether the bus is free. When the bus becomes free, the I2C1_CON register is written the control word that enables start bits and Master Transmitter mode, which is necessary to send a message. Finally, a while loop checks when the message has been sent, determining if a character can be written out and if true, sending a byte to the I2C1_DATA register. The mainline algorithm for this program looks like the following:

MAIN:

- Set stacks for USR and IRQ modes
- Initialize the clocks, begin software reset, change mux modes, program PSC, SCLL, SCLH, CON, and SA registers
- Use artificial delay loop to cause a “system pause”
- Initialize the New Haven LCD using the datasheet procedures and values
- Use artificial delay loop to cause a “system pause”
- Write to the screen using the polling method described above (from the Sitara manual)

Testing

The major goals for this project are described in three parts. These parts can be easily partitioned into smaller modules, which is the approach taken for this solution. For instance, in order to use polling correctly the program must initialize registers and perform the transmission. It seemed clear to write an initialization function for the I2C1 module to start the solution. Using the memory browser, it was determined that the desired control words were being written to the corresponding addresses of the required registers. For the I2C1_init function, this meant waking up the I2C1 clock, performing a software reset, enabling the system configuration wake-up enable, setting the SPI registers, programming the I2C1 (12 MHz module, 100 Kbps bus clock), and configuring the slave address after coming out of reset mode.

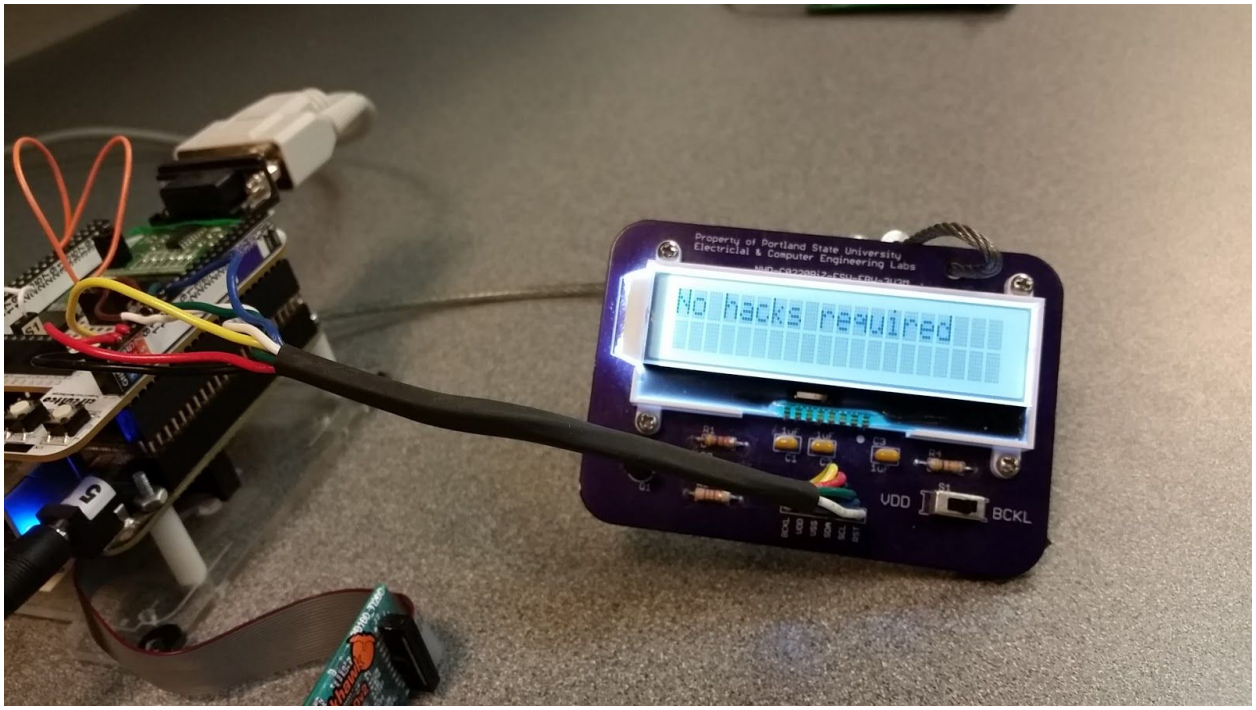
Once the init function was written and verified, an initialize function or init_LCD was written and tested. This function is a repeat of the New Haven data sheet initialization code at the end of the document. Setting the I2C1_CNT register to the number of letters in the message (11), checking the bus, using a delay, and then intermittently writing to the screen while checking XRDY occurs in init_LCD. This code was stepped-through, debugged, and memory-browsed in the same way the first initialization function was. It was obvious when this function began to work correctly because it causes black squares to display on the screen.

The polling transmit function, no_int_send, was tested in the same way as the other functions. The difference with testing this function is that the screen was tested to be displaying the message correctly or not. At first it seemed like there was an error because the first byte or character in the message did not become displayed. As a fix to this, a random character was put at the beginning of the message so it could be discarded without affecting the message.

The interrupt transmit function and int_director function were tested and debugged, but an error still exists. It is possible that the INTC_CONFIG register was not being reset properly. Regardless, the interrupt-driven part of this project was attempted and is included with the solution to the polling method for part 1.

Results

The output of this project is displaying a message on the New Haven LCD screen. The message chosen is a voiceline, “No hacks required”, from the video game Overwatch. This can be seen in the picture below, which was taken on 8 / 19 / 2017 in the Tektronix lab at PSU prior to the project demo.



Contract

Please type your name below to acknowledge the following contract. This contract must be included exactly as it is written below for all projects submitted in 37x.

This project is exclusively the product of my own work. I designed, developed and tested it myself with no help from anyone except the instructor and/or the T.A, and I did not give help to anyone else. I understand that this project will be cross-analyzed against the projects of all current and previous 37x students for plagiarism, and that evidence of copying, plagiarizing, or otherwise cheating will result in a score of 0 on the project. Since passing both projects is a requirement to pass the course, I agree it would be a bad plan to plagiarise this assignment.

Your Name Here
Matthew Etcyl Fleetwood

Design Log

8 / 9

- Since we need to use I2C1, note that this module is in the L4 Peripheral Interconnect domain. From Microprocessors by Hall, on page 231 (L4_PER Peripheral Memory Map), **the I2C1 registers are located at base address 0x4802_A000**
- **To wake up the I2C1 clock, look at 8.1.12.1.16** in the BBB data sheet. This shows the CM_PER_I2C1_CLKCTRL register and its field descriptions. **To turn on the clock, write 0x2 (as usual with the other clocks in the BBB) to the clock, which is at offset 48h.**

8 / 11

- **21.2.2 I2C clock and reset management** (references PRCM) ✔
- **21.3.3 I2C Reset** (use this flow to configure the device -- i.e. software reset) ✔
- Read **21.3.5 Start & stop conditions** (start is a high-to-low; BB = 1)
- We want to be operating in the **Master Transmitter** mode (**21.3.6.2**); our I2C controller will be a master and transmitting ✔
- **21.3.9 Prescaler**; not everyone uses the same prescaler value… recommended 45 - 65 (in both SCLL and SCLH, respectively). Need to be in reset mode to use prescaler. ✔
- **Interrupts:**
    - probably care about Transmit interrupt/status (XRDY)
    - Turn off FIFO if necessary since we're not using FIFO
    - "When the interrupt signal is activated, the Local Host must read the I2C_IRQSTATUS_RAW register to define the type of the interrupt, process the

request, and then write into these register the correct value to clear the interrupt flag”

- ***How to Program I2C*** on **pg. 4596**, part **21.3.15**
    1. Go into reset mode
    2. Need to get 12 MHz into the module
    3. Need 100 Kbps bus clock
    4. Configure its own address (only in I2C operating mode ...)
    5. Take it out of reset mode
- Initialization procedure - not using DMA
- Configure slave address and DATA counter registers
- **Use 3C for slave address**
- **31.3.15.54 Initiate a transfer**
- **21.3.15.6 Transmit Data**; read **XRDY** and write data into **I2C_DATA** reg
- **Fig 2-19 Table 21-12 I2C_IRQSTATUS_RAW**
    - **Bit 12 -- BB**; 0h = Bus is free, 1h = Bus is occupied
    - NEED TO PAUSE between sending data bytes (use a wait loop)
    - **Bit 2 -- ARDY**; done flag - when you are in master transmit (we are), the ARDY set condition is when DCOUNT = 0. Stop bit sent, transaction finished.
    - **Bit 4 -- XRDY**; asking for the next byte, ignore FIFO stuff, CPU can poll this bit instead of using an interrupt; CPU can only clear this bit by writing a 1… so you have to clear this bit

- Table 21-25 I2C_CNT reg field descriptions
    - Bits 15 - 0 DCOUNT. Number of bytes to be sent in transmission
- Table 21-26 I2C_DATA reg
- Table 21-27 I2C_CON reg
    - Bit 15 -- I2C_EN; when you first turn on the board it's not in reset mode. Need to reset: bring to low, change settings, then bring it high again - once in the beginning of your code
    - Bit 10 -- MST; want 1h. Every time start and stop bits are set, master mode needs to be set
    - Bit 9 -- TRX; every time transaction initiated, want 1h
    - Bits 1 and 0 -- STP and STT; start and stop conditions

Summary of sending data to the peripheral:

1. Cycle to send data: wait for XRDY, send byte, clear it, then wait again.
2. To initiate a transfer: start count, write data in data reg, config start and stop bits, make sure it knows it is the master and transmitter.
3. Then sending the bits, polling XRDY, tells us when a new byte gets written into DATA reg. Do that until DCOUNT is 0, or watch value of ARDY.
4. Put a delay between byte writes.

8/15

- Initialize the New Haven LCD using the init code at the end of the data sheet

  http://www.newhavendisplay.com/specs/NHD-C0220BiZ-FSW-FBW-3V3M.pdf

  - Converted the hex numbers to decimal to use for input

8/16

- Wrote and tested init + poll & transmit functions

8/18 - /19

- Wrote and tested interrupt-driven functions. Asked Charles but he didn't know

Project Code

Also attached as a text file.

```
/**
ECE 372
Summer term
8 / 18 / 2017
Matt Fleetwood, Portland, OR
Project 2

Uses the New Haven LCD to write text onto the screen.
Polling is used instead of interrupts.
**/

//Defines Section
#define HWREG(x) (*((volatile unsigned int *)(x)))

//INTC defines
#define INTC 0x48200000
#define INTC_PENDING_IRQ2 0x482000D8
#define INTCPS_INTC_CONTROL 0x48200048
#define MIR_SET2 0xC4
#define INTC_I2C1 0x00000080
```

```
//Control Module Defines
#define CON_MOD_BA 0x44E10000
#define SPI0_D1 0x958
#define SPI0_CS0 0x95C
#define I2C_EN 0x00000072

//I2C1 Defines
#define I2C1_BA 0x4802A000
#define I2C1_CLOCK 0x48
#define I2C_SYSC 0x10
#define I2C_PSC 0xB0
#define I2C_SCLL 0xB4
#define I2C_SCLH 0xB8
#define I2C_CON 0xA4
#define I2C_CNT 0x98
#define I2C_DATA 0x9C
#define I2C_IRQ_SET 0x2C
#define I2C_IRQ_RAW 0x24
#define I2C_SA 0xAC
#define I2C_IRQ 0x24

//other defines
#define CM_PER 0x44E00000

//Value defines
#define TURN_ON 0x2
#define MSTR_TRSMTR 0x00008600
#define START 0x00008603

//Function Declarations
void I2C1_init(void);
void I2C_do_transmit(unsigned int dcount);
void delay(void);
void I2C1_interrupt_handler(void);
void init_LCD(void);
void no_int_send(unsigned int dcount);

//global variables
//these are the variables I used just for ideas
volatile unsigned int flag = 1;             //dcount reached 0 flag
volatile unsigned char data_to_slave[19] = "pNo hacks required";   //array holding what to send
volatile unsigned int num_of_bytes = 19;     //number of bytes to send
volatile unsigned char data_to_slave2[18] = "pJust another day";   //array holding what to send
```

```
volatile unsigned int num_of_bytes2 = 18;     //number of bytes to send
volatile unsigned int t_count = 0;          //number of sent bytes
volatile unsigned int USR_STACK[100];
volatile unsigned int INT_STACK[100];

int main(void)
{
      //this will disable interrupts for the initialization
         asm("   mrs              r0, CPSR\n\t"
                        "         orr              r0, r0, #0x80\n\t"
                        "         msr              CPSR_c, R0");
         //SET UP STACKS
         //init USR stack
         asm("LDR R13, =USR_STACK");
         asm("ADD R13, R13, #0x100");
         //init IRQ stack
         asm("CPS #0x12");
         asm("LDR R13, =INT_STACK");
         asm("ADD R13, R13, #0x100");
         asm("CPS #0x13");

         //Call initialization function for I2C1 clock enable, software reset, mux mode changed to
2,
      //PSC, SCLL, SCLH, CON, and SA registers set
      I2C1_init();

      //Give the LCD time to think
      delay();

      //Initiatlize the New Haven LCD to default format
      init_LCD();

      delay();

      //Send the message from the data_to_slave array using polling instead of interrupts
      no_int_send(num_of_bytes);

      //enable irqs
            asm("       mrs              r0, CPSR\n\t"
                        "         bic              r0, r0, #0x80\n\t"
                        "         msr              CPSR_c, R0");

      //set up data_to_slave with correct LCD init values for the New Haven LCD
```

```
	//from http://www.newhavendisplay.com/specs/NHD-C0220BiZ-FSW-FBW-3V3M.pdf
		I2C_do_transmit(num_of_bytes2);

	//Do-nothing loop
		while(1)
		 {}
		return 0;
}

//Enable the clocks, software resets, and IRQs. Set the CON, PSC, SCLL, SCLH, and SA
registers
void I2C1_init(void)
{
	   //I2C 1 INIT
	//Wake-up I2C1
	HWREG(CM_PER + I2C1_CLOCK) = 0x02;

	   //Sofware reset by setting SRST bit in the I2C_SYSC reg
	HWREG(I2C1_BA + I2C_SYSC) = TURN_ON;

	   //Write wake-up value to I2C_SYSC
	HWREG(I2C1_BA + I2C_SYSC) = 0x4;

	//Reset INTC by writing to INTC_CONFIG register
	HWREG(INTC + 0x10) = TURN_ON;

	   //INTC INIT
	//Unmask IRQ for I2C1
	HWREG(INTC + MIR_SET2) = INTC_I2C1;

	//Set SPI registers in the control module with pull-up resistors, slew rate, and MUX mode
	//to enable I2C1
	HWREG(CON_MOD_BA + SPI0_CS0) = I2C_EN; //Write 0x12 to the spi0_cs0 reg at
offset 0x95C for the I2C1_SCL
	HWREG(CON_MOD_BA + SPI0_D1) = I2C_EN; //Write 0x12 to the spi0_d1 reg at offset
0x958 for the I2C1_SDA

	//Program I2C1 prescaler for approx. 12 MHz into the module
	HWREG(I2C1_BA + I2C_PSC) = 0x03;  //Divide the system clock, SCLK (192 MHz), by 16
to get ~12-MHz I2C module clk

	   //100 Kbps bus clock
	   HWREG(I2C1_BA + I2C_SCLL) = 0x37;  //55 in decimal
```

```
        HWREG(I2C1_BA + I2C_SCLH) = 0x39;  //56 in decimal

        //Get out of reset mode
        HWREG(I2C1_BA + I2C_CON) = 0x00008600;

            //Configure slave address
            HWREG(I2C1_BA + I2C_SA) = 0x3C; //Slave address is 0x3C for the New Haven
device

            return;
}

//Initialize the New Haven LCD screen according to its documentation
void init_LCD(void)
{
            //Set the COUNT reg with the number of init values
            HWREG(I2C1_BA + I2C_CNT) = 11;

    //Check if the bus is free by polling the busy bit in the IRQ_RAW reg
    while(1 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x1000))
                    { //Do nothing... waiting for bus to be free
                    }

    //Give the LCD time to think...
    delay();

            //Begin the I2C1 transmit by toggling the bits in the CON reg
             HWREG(I2C1_BA + I2C_CON) = START;

            //Give the LCD time to think...
            delay();

            //Send the hex values from the New Haven data sheet to complete the setup for the LCD
            HWREG(I2C1_BA + I2C_DATA) = 60; //60 "Slave address" (in decimal)
            while(0 == ((HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10)))
                    { //Do nothing... waiting for XRDY to be free
                    }
            delay();
            HWREG(I2C1_BA + I2C_DATA) = 0; //0 Comsend = 0x00
              while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
                    { //Do nothing... waiting for XRDY
                }
            delay();
```

```
HWREG(I2C1_BA + I2C_DATA) = 56; //56
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 57; //57
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 20; //20
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 120; //120
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 94; //94
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 109; //109
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
HWREG(I2C1_BA + I2C_DATA) = 12; //12
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
HWREG(I2C1_BA + I2C_DATA) = 1; //1
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
delay();
  while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
        { //Do nothing... waiting for XRDY
    }
HWREG(I2C1_BA + I2C_DATA) = 6; //6
```

```
        delay();

        return;
}

//Set char counter, Master + Transmitter mode, dcount, interrupts, cleanup + set ints, begin start bit
//to begin interrupt transmission
void I2C_do_transmit(unsigned int dcount)
{
        //Operating mode set to Master Transmitter
        HWREG(I2C1_BA + I2C_CON) = MSTR_TRSMTR;

        //set dcount
        HWREG(I2C1_BA + I2C_CNT) = dcount; //Total number of bytes in the msg

        //cleanup interrupts
   HWREG(I2C1_BA + I2C_IRQ_RAW) = 0x114;

        //set correct interrupts
        //Enable BF_E, XRDY_IE, and ARDY_IE
   HWREG(I2C1_BA + I2C_IRQ_SET) = 0x00000114;      //Set bit 4, XRDY_IE, to enable transmit data ready, bit 2 for ARDY_IE
                                                   //and set bit 8, BF_E, to enable bus free ready
   //Generate a start condition
   HWREG(I2C1_BA + I2C_CON) = 0x1;

        while(flag) //waiting for flag
        {

        }

        while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x4)) //checking stop condition
        {

        }

        flag = 1;

      return;
}

//Transmit a message using polling (no interrupts)
```

```
void no_int_send(unsigned int dcount)
{
    //Set DCOUNT in the COUNT reg
    HWREG(I2C1_BA + I2C_CNT) = dcount;

    //Give the LCD time to think
    delay();

    //Check if the bus is free by polling the busy bit in the IRQ_RAW reg
    while(1 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x1000))
            { //Do nothing... waiting for bus to be free
        }

    delay();

    //Begin the I2C1 transmit by toggling Master Transmitter mode, and the start and stop bits in
the CON reg
    HWREG(I2C1_BA + I2C_CON) = START;

    delay();

    int msg_sent = 0;
    while(msg_sent != 19) //While there are still chars to send, transmit the message
        {
         //Check if XRDY is asserted by polling the XRDY bit in the IRQ_RAW reg
           while(0 == (HWREG(I2C1_BA + I2C_IRQ_RAW) & 0x10))
                 { //Do nothing... waiting for XRDY
            }

             delay();
         //Send a char or byte from the message array using the msg_sent counter as an index
         HWREG(I2C1_BA + I2C_DATA) = data_to_slave[msg_sent];
         msg_sent = msg_sent + 1; //Get to the next char in the message
        }

    return;
}

//This function will be hooked in the startup_ARMCA8.s file
void int_handler(void){
   //Do not STMFD / LDMFD; the LR is automatically saved

   //HWREG(INTC + MIR_SET2) = 0;
```

```
    //Check for I2C1 interrupts on PENDING_IRQ2
    if(HWREG(INTC_PENDING_IRQ2) == INTC_I2C1) //INTC_I2C1 = 0x00000080
      I2C1_interrupt_handler();

    HWREG(INTCPS_INTC_CONTROL) = 0x1; //This value also resets
    asm("LDMFD SP!, {R11}"); //Gets rid of the extra stack value
    asm("LDMFD SP!, {LR}"); //Gets the correct LR value back
    asm("SUBS PC, LR, #0x4"); //Goes back to wait-loop, USR mode
}

//Responds to the I2C1 interrupts by sending a message to the New Haven LCD
void I2C1_interrupt_handler(void)
{
        int status = 0;
      //get IRQ status
      status = HWREG(I2C1_BA + I2C_IRQ_RAW);

        if(status & 0x10)       //transmit ready
        {
                //send next character
              HWREG(I2C1_BA + I2C_DATA) = data_to_slave2[t_count];
              delay();
                //clear transmit ready IRQ
      HWREG(I2C1_BA + I2C_IRQ_RAW) = 0x00000010;
                //Update the char counter
      t_count = t_count + 1;
                if(t_count == num_of_bytes2)
                {
                        //disable transmit interrupts
       HWREG(I2C1_BA + I2C_IRQ_RAW) = 0x00000010;

                }
        }

        if(status & 0x100)      //stop condition (bus free)
        {
                //disable correct interrupts
      HWREG(I2C1_BA + I2C_IRQ_RAW) = 0x00000114;
      //Clear I2C1 interrupt requests in the ITR register
      HWREG(INTC + 0xC0) = 0x00000080;
                flag = 0;
        }
```

```
        if(status & 0x2)    //f in an error state
        {
                //HWREG(I2C1_BA + I2C1_IRQ_DIS) = 0x11A;
                HWREG(I2C1_BA + I2C_CON) |= 0x2;
                flag = 0;
        }

    return;
}

//Use this to give the LCD board time to think
void delay(void)
{
        int y;
        for(y = 0; y <= 10000; y ++)
        {
                //small delay
        }

        return;
}
```