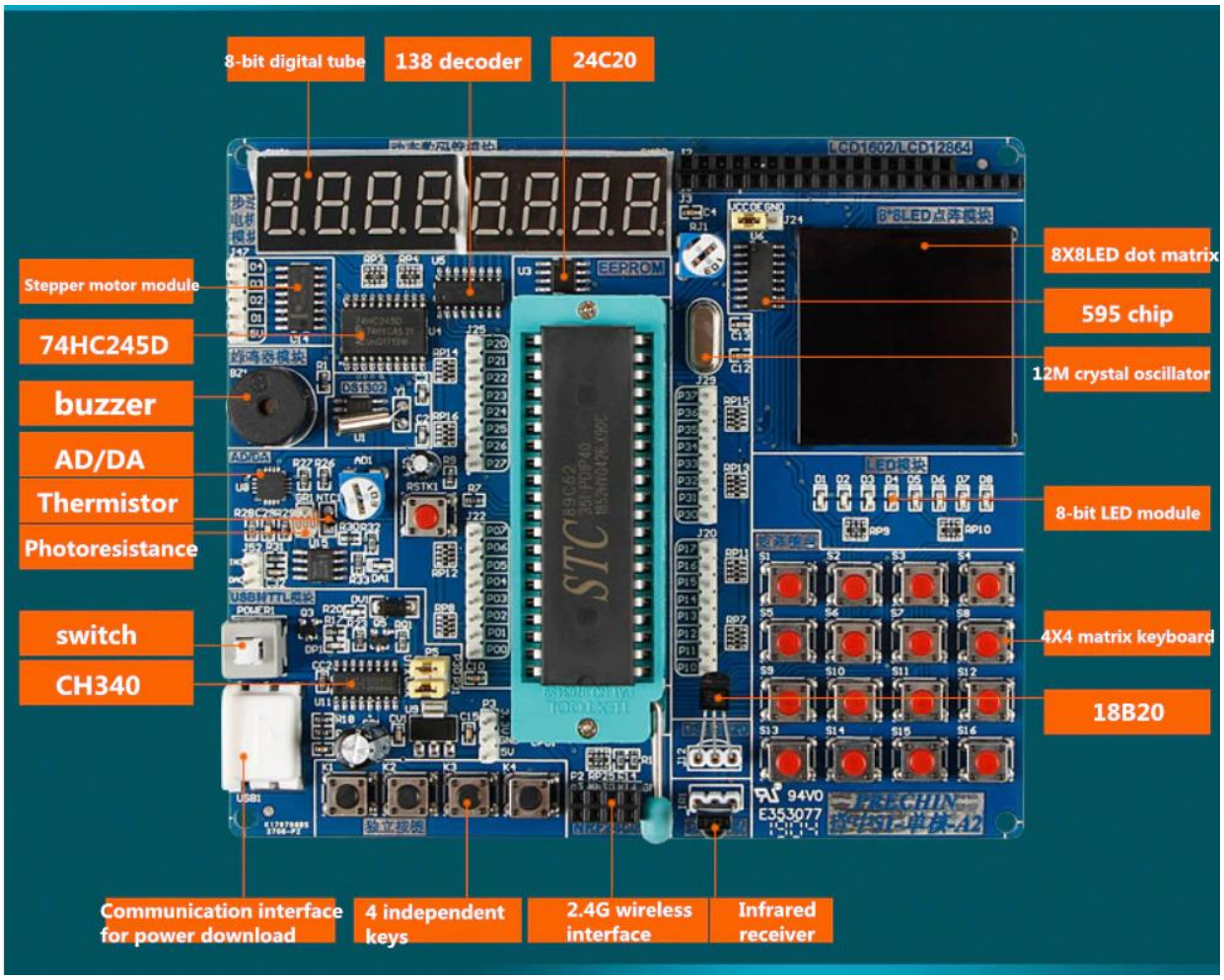
STC89C52 training manual

Instructions

1. Please carry out the tasks by yourself and don't copy paste from the internet or other developers. The purpose of this training is to develop problem solving and development skills in you which won't happen if you don't follow the instructions.
2. When you complete a task, please make a demonstration video and keep it on your PC. Send videos to your trainer on Whatsapp and via USB flash drive. The demo videos will be checked by your trainer.
3. You can use example code but not ready made programs from the internet.
4. Make a folder for each task and place all the related files like code, demo video, simulations, images in that folder e.g. Folder "Task 1" and files : "Task 1 demo video", "Task 1 code" etc.

The development board we are using is purchased from
https://www.aliexpress.com/item/4000544196373.html



Task 1 : Setting up development environment, study datasheet and schematics of the development board

1. Install development tools/IDE and drivers. Watch this video for help
https://www.youtube.com/watch?v=_GlKeRqBq4s

2. Skim through the datasheet of STC89C52RC microcontroller to get an idea of its capabilities.
   http://www.stcmicro.com/datasheet/STC89C51RC-en.pdf
   Please answer the following questions about STC89C52 microcontroller
   a. What is the size of the RAM and flash.
   b. How many IO pins are available
   c. What peripherals are available e.g. serial, ADC etc.
   d. What is meant by IAP
   e. How many timers are available

f. How many sources of interrupt are there
g. Maximum clock frequency supported

3. Study the schematic of the development board. The file name is A2 development board schematic diagram(With translation).pdf and is found in the “STC89C52 training” folder.

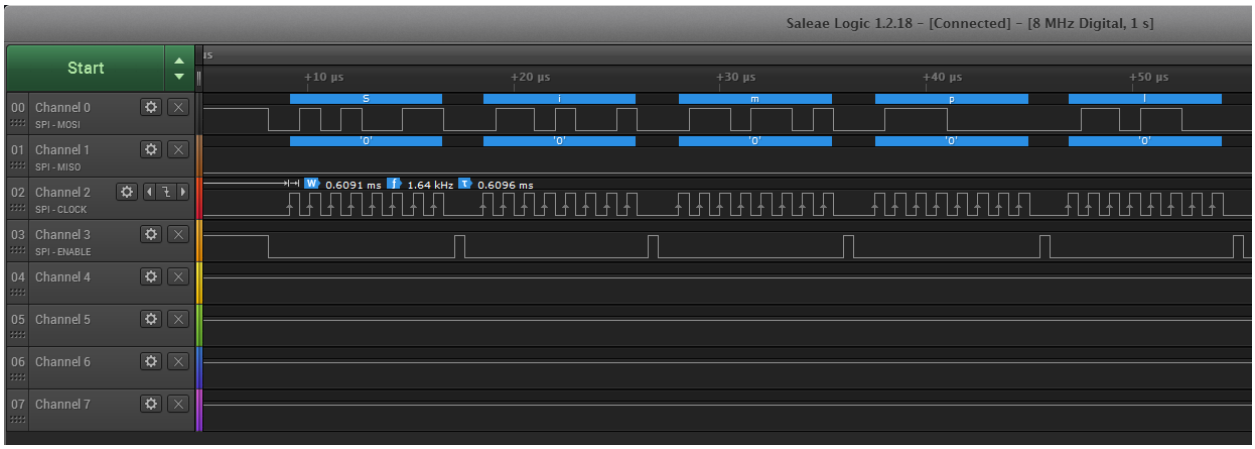
Task 2 : Logic analyzer setup

Install Logic 1.2.18 software and drivers for logic analyzer. The logic analyzer we are using is this
https://www.aliexpress.com/item/1005001451709172.html
Download link
https://downloads.saleae.com/logic/1.2.18/Logic+Setup+1.2.18.exe

Logic Analyzer

8-Channel Logic Analyzer
Maximum sampling rate is 24M, measurement bandwidth is 80ns.
Using USB interface, easy to operate, can meet a variety of usage needs.

Logic 1.1.18 software



See this video on logic analyzer
https://www.youtube.com/watch?v=rR5cEFRO9_s

Task 3 : Blink an LED

Blink the LED D1 present on the development board. Make the delay by wasting CPU cycles (you can use while or for loop). The blink delay should be 500 ms. Use a logic analyzer to measure the delay. Attach screenshots of logic analyzer showing the delay.

a. Use OR operator to set a single bit of a register without affecting other bits e.g. to set bit 0 of a register REG

   REG = REG | 0x01;

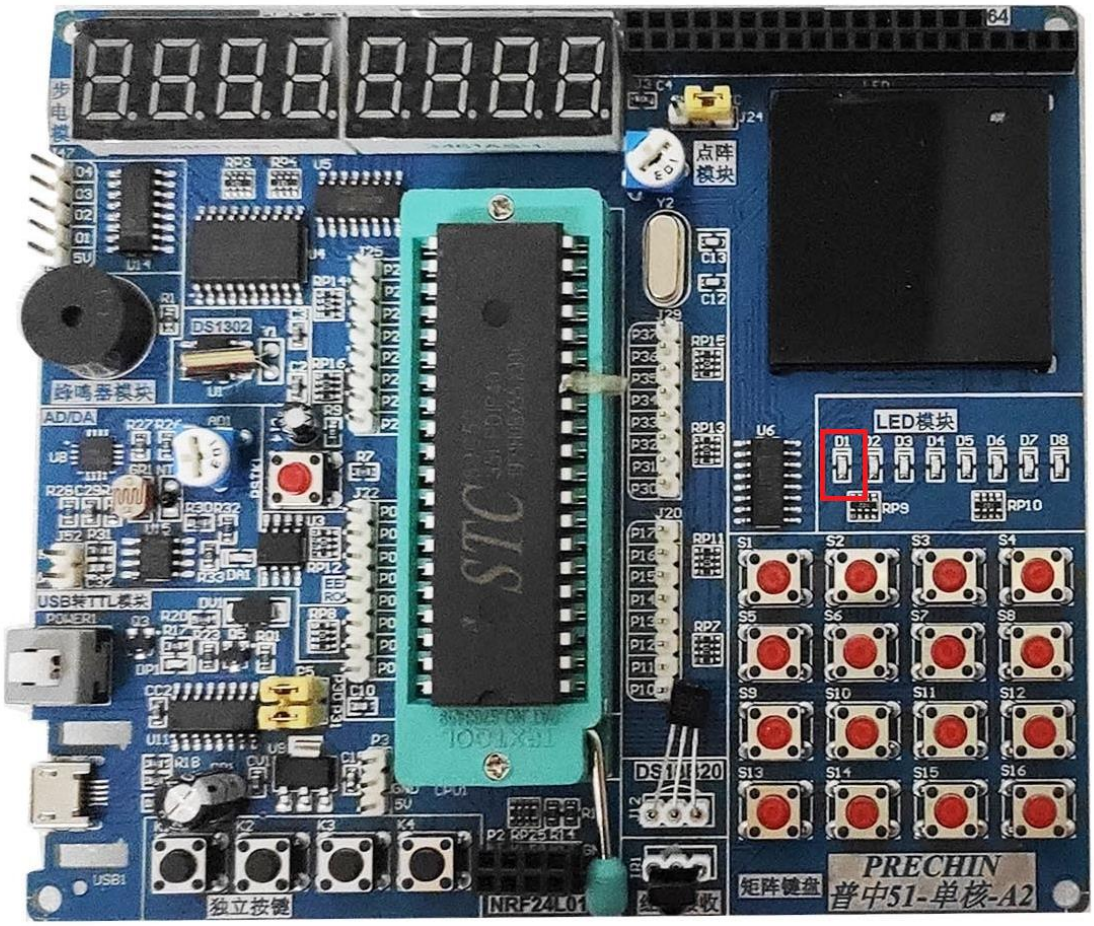
b. Use AND operator to clear a single bit of the register without affecting other bits e.g. to set bit 0 of a register REG
   REG = REG & ~0x01;

   Define all the bits and use the bit names in following tasks e.g.
   #define BIT0 0x01
   #define BIT1 0x02
   #define BIT2 0x04
   #define BIT3 0x08
   #define BIT4 0x10
   #define BIT5 0x20
   #define BIT6 0x40
   #define BIT7 0x80

   To clear BIT3 we can use
   REG = REG & ~BIT3;
   Or

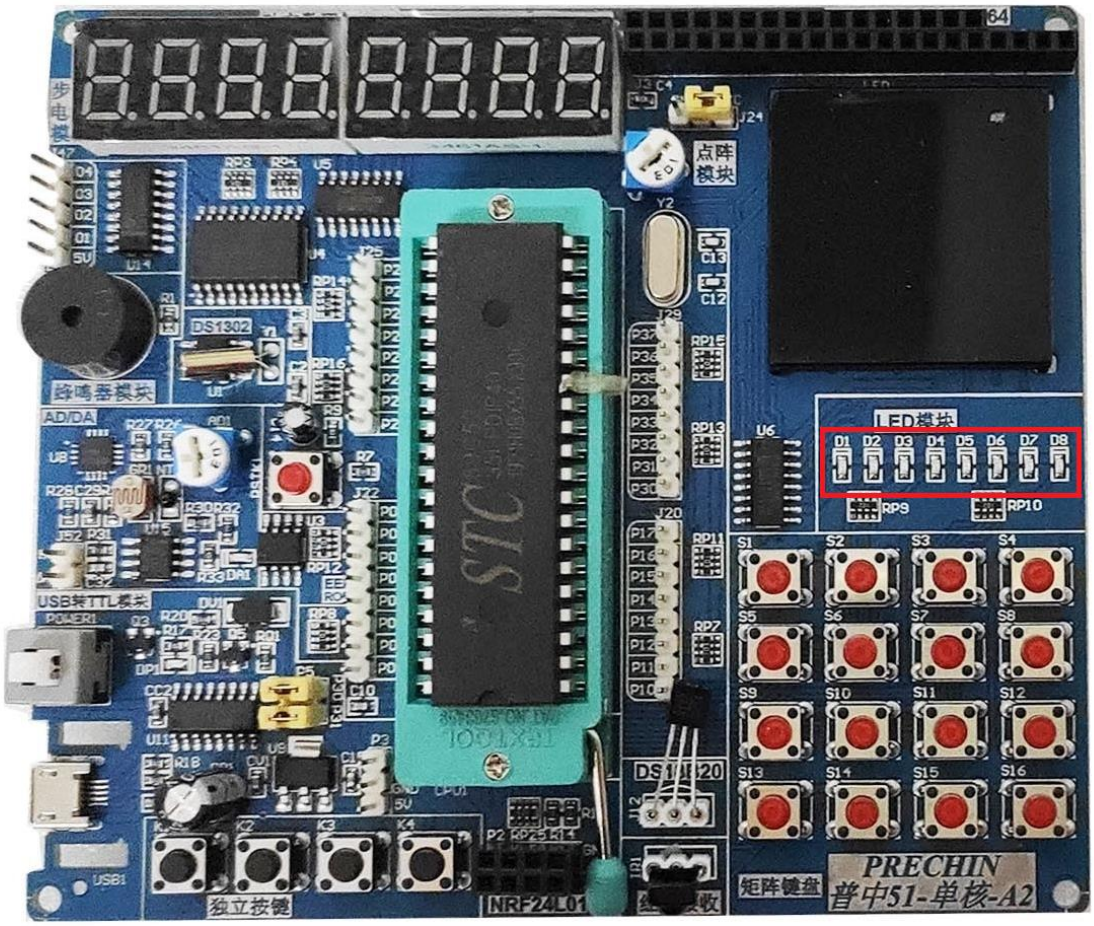
REG &= ~BIT3 ;

You can also define bits for LEDs like
#define LED_D1 0x01



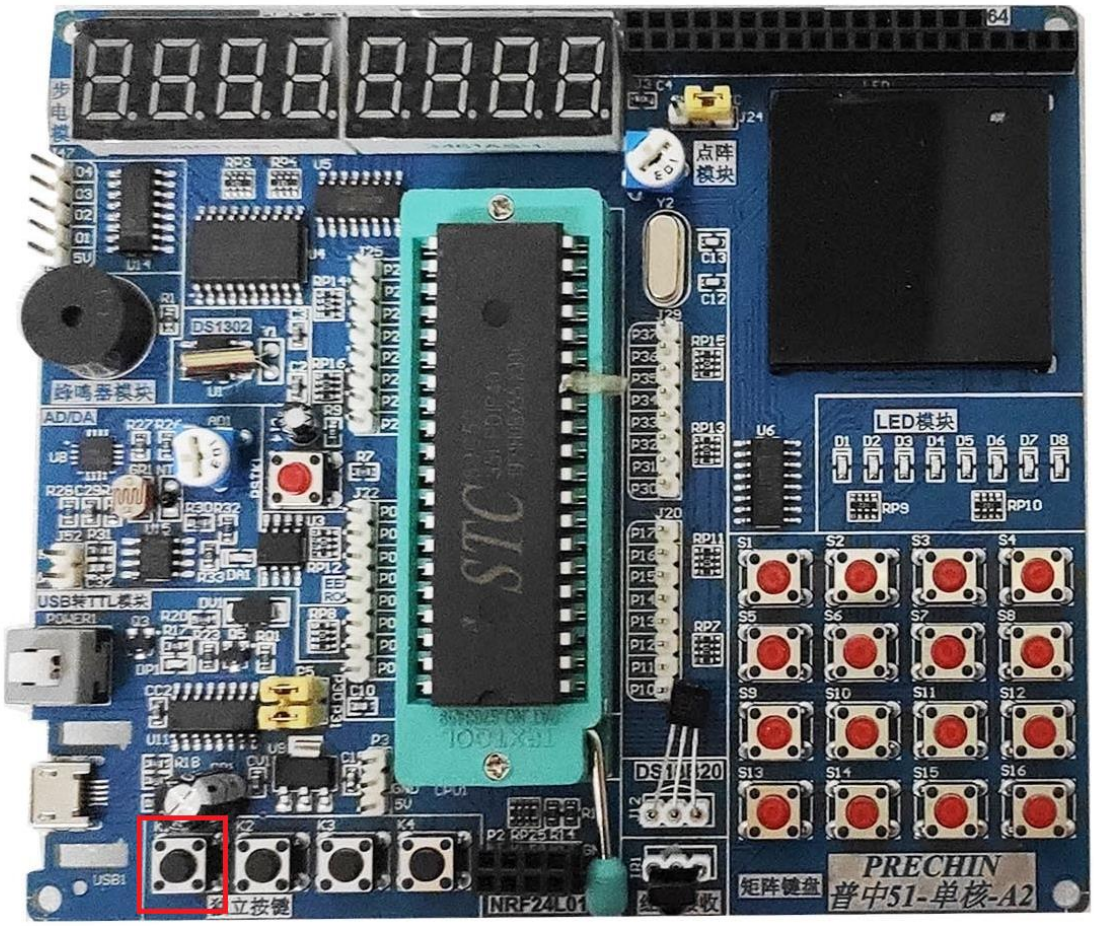
Task 4 : LED Pattern

Blink all the LEDs from D1 to D8 present on the development board in alternating pattern => D1, D3, D5, D7 ON, D2, D4, D6, D8 OFF => D1, D3, D5, D7 OFF, D2, D4, D6, D8 ON



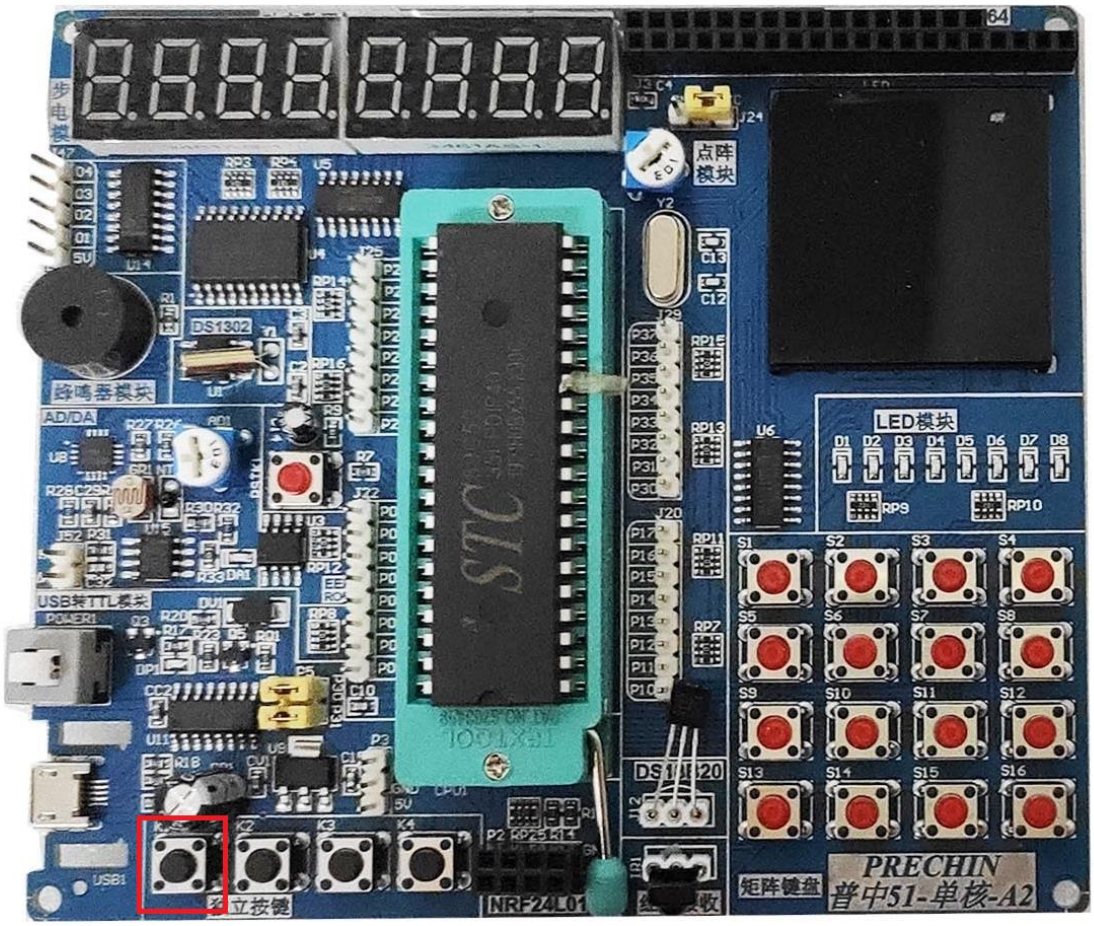
Task 5 : Button interfacing using polling

Write a C program to read the button state using polling. If button K1 is pressed it should turn on LED D1. If pressed again it should turn off the LED D1. Add debouncing in the code so one button press is detected only once. This should continue in a loop.



Task 6 : Button interfacing using interrupt

Write a C program to detect button press using interrupts. If button K1 is pressed it should turn on LED D1. If pressed again it should turn off the LED D1. Add debouncing in the code so one button press is detected only once.



Task 7 : LED control using buttons (polling)

Interface buttons K1 to K4 such that buttons K1-K4 turn ON/OFF LEDs D1-D4 using polling. K1 should control D1, K2 should control D2 and so on. Use debouncing as well.

Task 8 : LED control using buttons (interrupts)

Interface buttons K1 to K4 such that buttons K1-K4 turn ON/OFF LEDs D1-D4 using interrupts when available and polling if interrupt is not available. K1 should control D1, K2 should control D2 and so on. Use debouncing as well.

Task 9 : Timer interrupt

Configure a timer to generate an interrupt every 10 ms. In the Interrupt Service Routine (ISR) toggle LED D1. Use auto reload mode of timer so you don't have to reload the timer registers in your code manually.

Task 10 : Generating delay with timer

1. Using the 10 ms interrupt from the previous task, generate a delay of 100 ms and 1000 ms and toggle LEDs at the following rates:
LED D1 => 10 ms
LED D2 => 100 ms
LED D3 => 1000 ms

Hint : Make a variable "cntr_100ms" and initialize it to zero, every time ISR is executed increment the cntr_100ms by one. In the next line check if cntr_100ms has reached a value of 10, if so it means that ISR has executed 10 times. Since ISR executes every 10 ms it means that 100 ms have passed. Toggle the LED D2 in the body of the if condition and set the cntr_100ms variable to 0 so it counts another 100 ms and so on. The same approach can be used for 1000 ms delay.

2. Doing timed operations in main function (raising flags from timer interrupt)

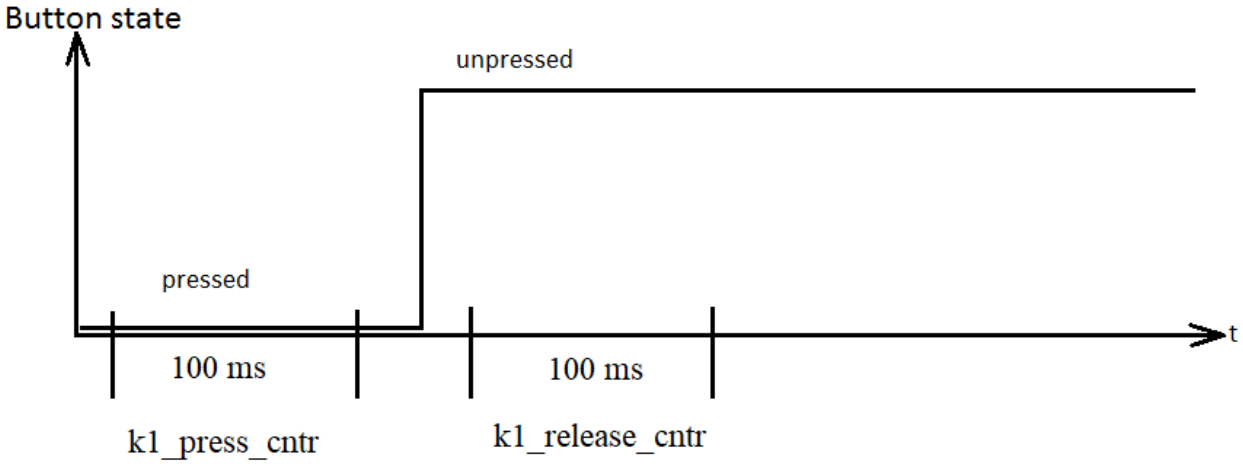
```
int cntr_100ms = 0;        //variable declaration;
int cntr_1000ms = 0;
char flag_100_ms = 0;
char flag_1000_ms = 0;

sbit LED_D1 = P2^0;     //LED D1 Connected to port P2^0
sbit LED_D2 = P2^1;     //LED D2 Connected to port P2^1
sbit LED_D3 = P2^2;     //LED D3 Connected to port P2^2
sbit LED_D4 = P2^3;     //LED D3 Connected to port P2^3

void ISR_timer0(void) interrupt 1     //interrupt service routine of 10ms overflow
{
        cntr_100ms ++ ;
        cntr_1000ms++;

        if (cntr_100ms >= 10)              //if counter reaches 10 it means 100 ms have passed
        {
                LED_D1 =~ LED_D1 ;          //toggle the LED D1
                flag_100_ms = 1;               //raise the 100 ms flag
```

```
            cntr_100ms = 0;                 //restart this counter
        }

        if (cntr_1000ms >= 10)              //if this counter reaches 100 it means 1000 ms have
passed
        {
            LED_D2 =~ LED_D2;               //toggle the LED D2
            cntr_1000ms = 0;
            flag_1000_ms = 1;                           //raise the 1000 ms flag
        }

}

void main (void )
{
        while(1)
        {
        if(flag_100_ms)
        {
            flag_100_ms = 0;
            LED_D3 =~ LED_D3;               //toggle the LED D3
            //This block of code will execute after every 100 ms
        }

        if(flag_1000_ms)
        {
            flag_1000_ms = 0;
            LED_D4 =~ LED_D4;               //toggle the LED D4
            //This block of code will execute after every 1000 ms
        }
        }

}
```

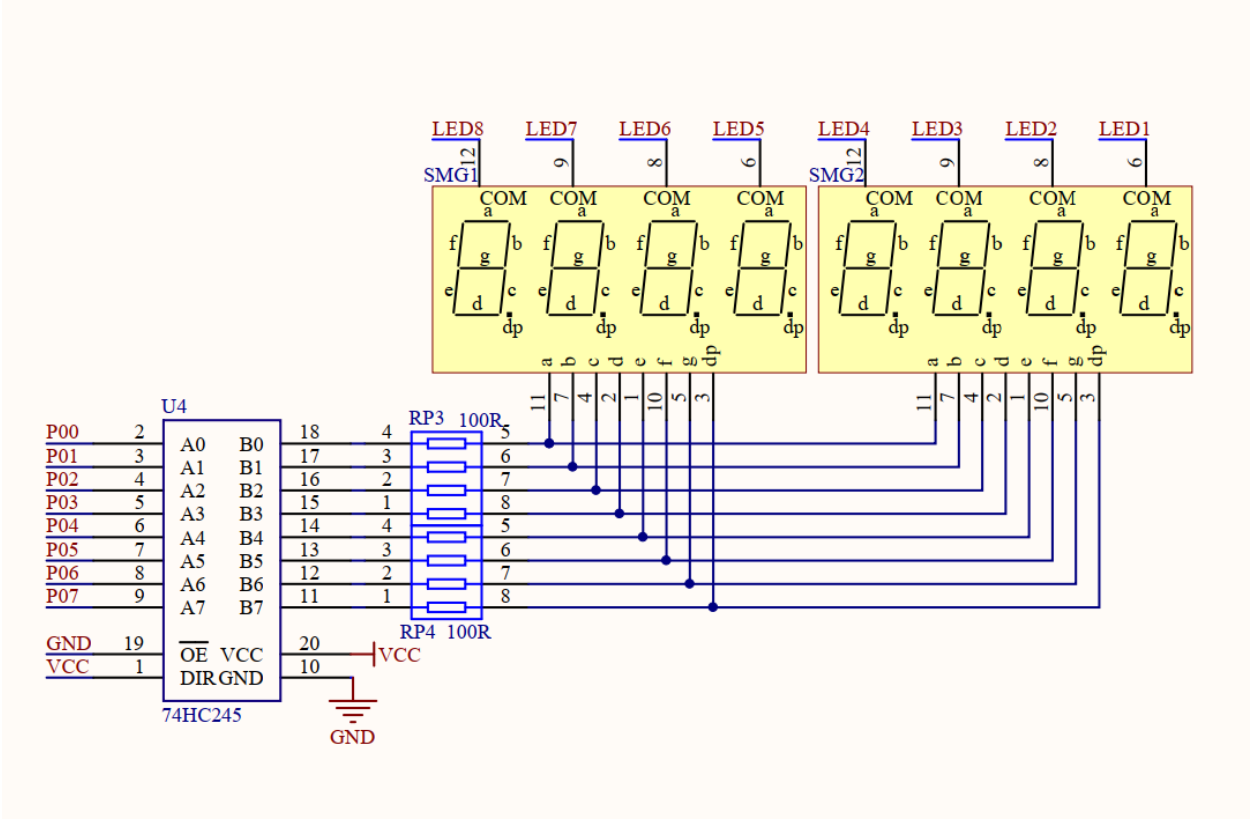
Study the above code and run it on your board. This code uses flags to signal the passage of a certain amount time from timer interrupt. e.g. in this case the timer interrupt signals to the main function every 100ms and 1000ms. The advantage is that we don't perform long operations in interrupts so we use flags in main function to check if 100ms have passed, if yes then we perform the operation, clear the flag and then again wait for the next flag.
Note : If you perform long operations in interrupts then it affects the accuracy of timing for example if an operation takes 25 ms to execute, if we perform it in a 10 ms timer ISR then the next timer interrupt will come after 25 ms and not 10 ms.

Task 11 : Debouncing using timer interrupt

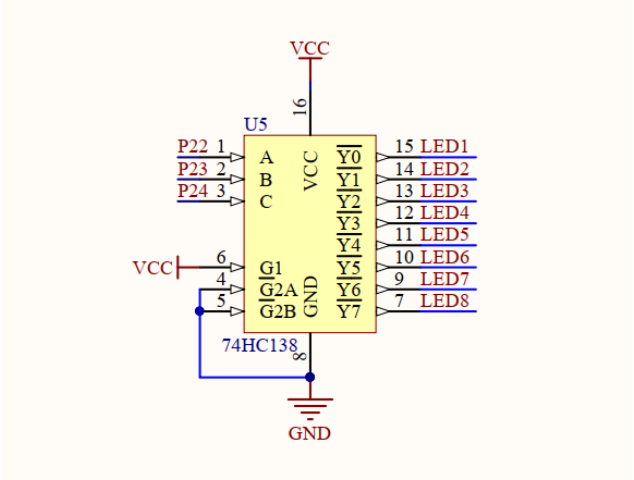
Create a variable called k1_btn_state and initialize it to 0. Create two more variables, k1_press_cntr and k1_release_cntr and initialize both of them to zero. In 10 ms timer interrupt if k1_btn_state is 0 and if button k1 is pressed, increment k1_press_cntr else set the k1_press_cntr to zero. If k1_press_cntr reaches 10 it means that the button k1 was pressed continuously for 100 ms hence toggle an LED to show a button press was detected, also change the state of k1_btn_state to 1. If the state of k1_btn_state is 1 it means that we now have to detect a button release event. In the same timer interrupt, increment k1_release_cntr by one if button k1 is released, otherwise set it to 0 . If k1_release_cntr reaches 10 it means the button was kept released for a continuous 100 ms hence change the k1_btn_state to 0 so another button press can be detected.



Task 12 : Seven segment display interfacing



There are eight 7 segment displays present on the development board. The data lines for all of them are common (P00 to P07), therefore we have to use multiplexing to use them all simultaneously. A single display can be turned ON/OFF using LED1-LED8 signals. To save pins LED1-LED8 signals are provided through a 3- to 8-Line decoder which means that three pins from the microcontroller can control 8 signals (LED1 to LED8) as shown in the following figure.



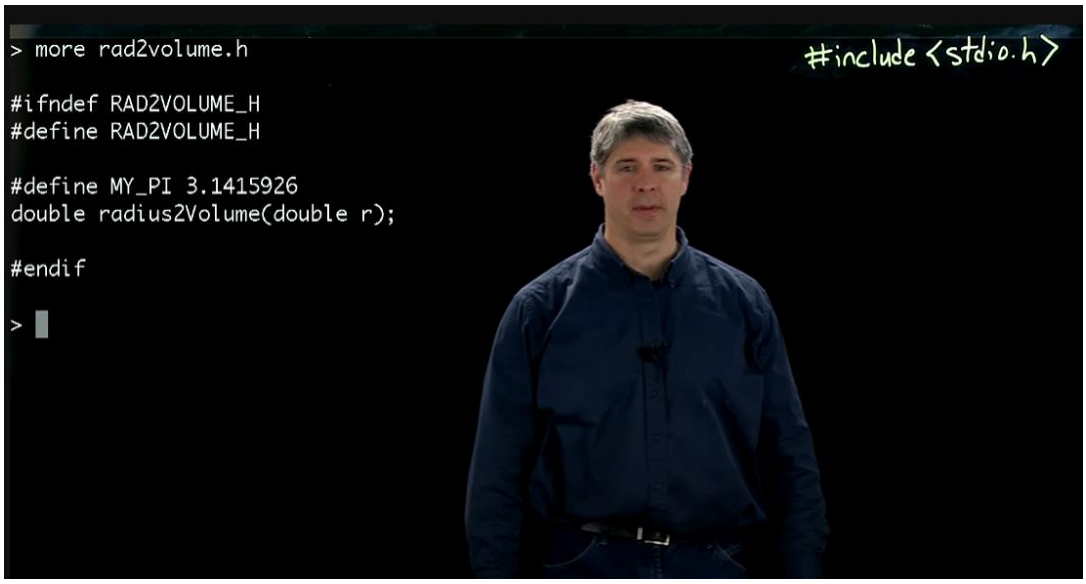
1. Study the datasheet of 74HC245 and tell what is the purpose of using it with 7 segment displays.

2. Study the datasheet of 74HC138 and tell what is the purpose of using it with 7 segment displays.
3. Write a program to select LED1 to LED8 with a delay of 500 ms (use timer delay), the final result of the program will be that 7 segment displays will turn ON one by one from first to eighth display.
4. Configure a timer interrupt for 10 ms. Now decrease the delay in the above program to 10 ms using the timer interrupt (Update seven segment displays in timer ISR and not in main function, don't use CPU cycle wasting delays). The 7 segment displays will appear as all working simultaneously. This happens because too fast moving things appear as continuous to a human eye. Calculate the refresh rate of the displays. The displays won't be smooth due to the low refresh rate.
5. Decrease the timer interrupt time so that the refresh rate of the displays becomes 50 Hz. Please note that each display gets its turn to refresh after 8 timer interrupts so the refresh rate you calculate has to be divided by 8. Show your calculations for a refresh rate of 50 Hz.
6. Write a lookup table to show numbers from 0-9 on the displays.
7. Write a program to show an eight digit number of the displays e.g 12345678. Assign a variable for each display and use the lookup table to display number on the displays.
8. Make a function called display_num_7seg(long num) which takes any arbitrary number as input, converts it and displays it on the seven segment displays. e.g. If you call
   long num = 42354245;
   display_num_7seg(num);

   The above code should display 42354245 on the seven segment displays. If num = 97561654 then the seven segment displays should display 97561654. Do conversion in real time and don't hard code the number to be displayed.

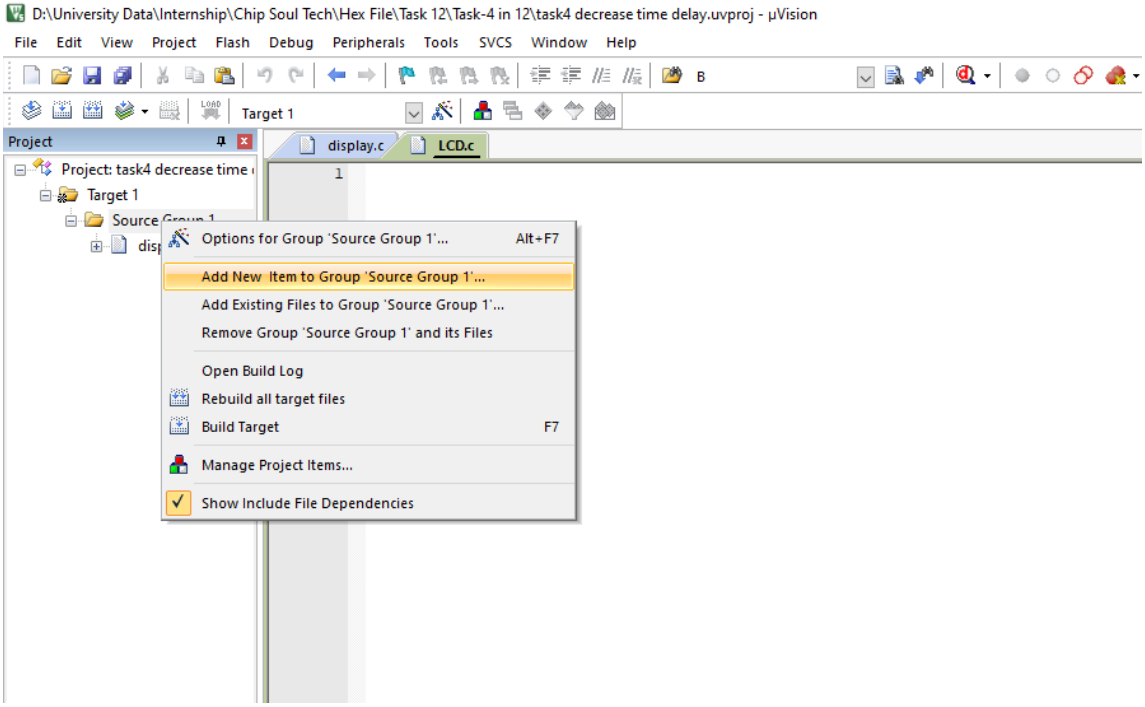
Task 13 : File organization using .h and .c files, external variables

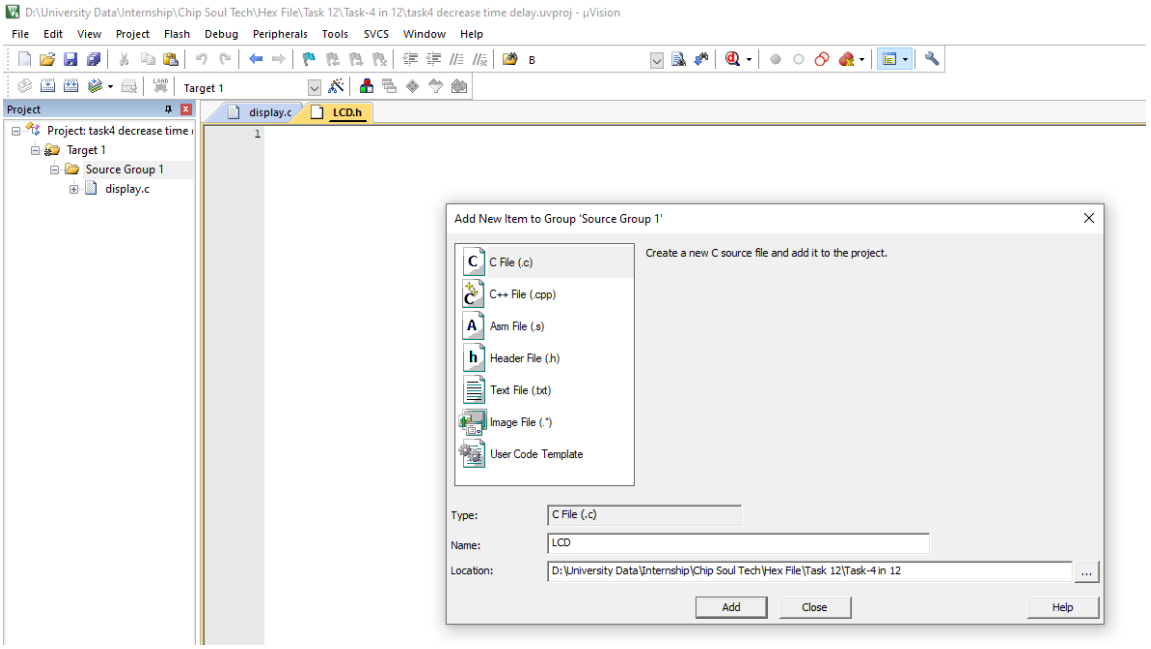
Watch this video to learn about organizing files in .h and .c files
https://www.youtube.com/watch?v=5UMHbzZGQuE

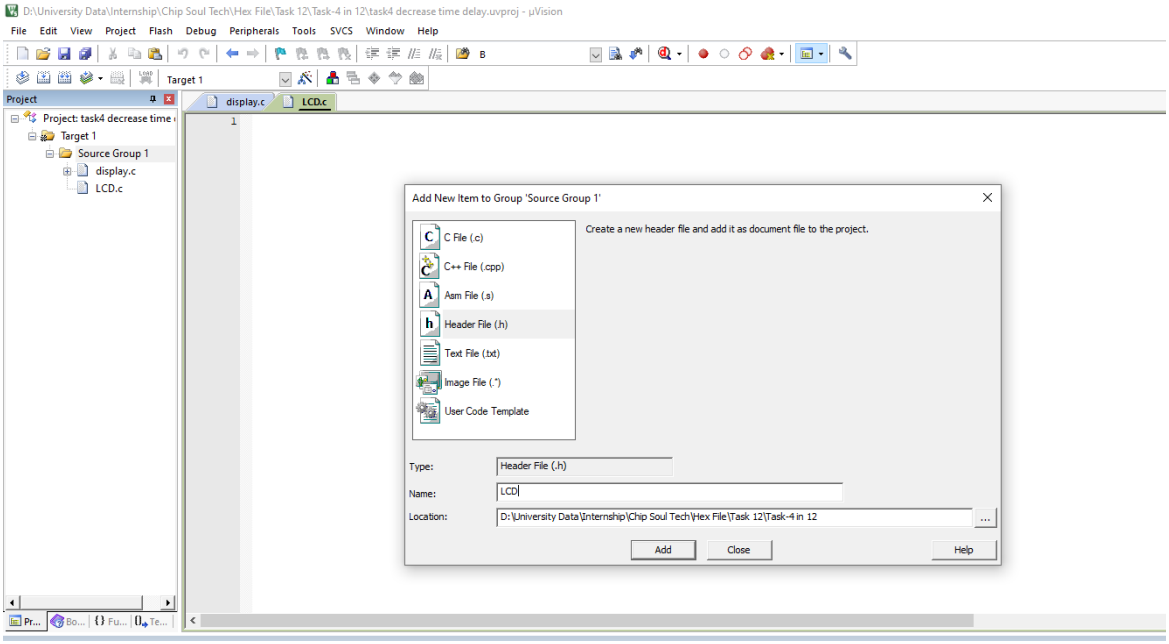


For example if we have LCD related code then we can make two files LCD.h and LCD.c to organize the LCD code. LCD.h file should contain function prototypes, constants (#defines etc.) and LCD.c file should contain the function implementation. When we need to use LCD in a project we can simply add #include “LCD.h” to the main file to use the LCD code.

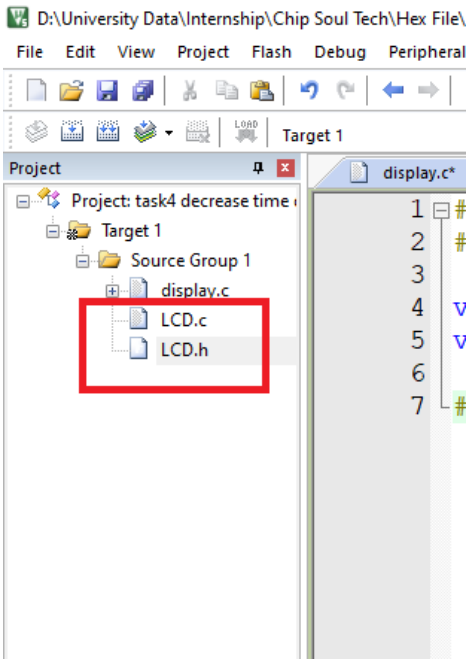
1. Add two files LCD.h and LCD.c to your project







When both files are included, the project should look something like this:

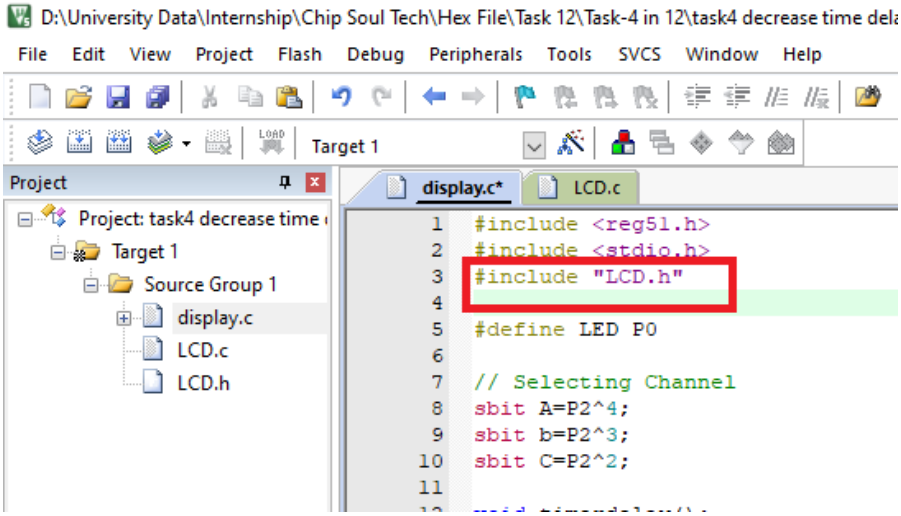


2. Now open the LCD.h file and add the following lines.

   #ifndef LCD_H
   #define LCD_H

   #endif
   This block of code makes sure that the LCD code is included only once, if you don't add this code, the compiler will generate "function redeclaration error" or similar other errors.

3. Include the LCD.h file in the main file (the file where main function exists)



4. Add the following line to the LCD.h file

#define LCD 1

All the files which include LCD.h file will have the above define available. Write a code in main file to turn ON an LED if LCD is 1 otherwise turn OFF the LED i.e.

```
if(LCD == 1)
   turn_on_led();
else
 turn_off_led();
```

Check if LED is ON. Change to #define LCD 0, compile and run the code, the LED should be OFF. This task proves that the material available in LCD.h is available to the main file.

5. Remove the following lines from LCD.h file

```
#ifndef LCD_H
#define LCD_H

#endif
```

Write the following two lines in LCD.h file

```
#define LCD 1
int x = 0;
```

Include the LCD.h file in the main file twice and compile the code. The compiler should generate “multiple initialization error”. Now introduce these lines back in LCD.h file

```
#ifndef LCD_H
#define LCD_H

#endif
```

Compile the code, it should compile fine.

Sharing variables between files

6. https://www.youtube.com/watch?v=1Dkfmf4PmvQ Watch this video to learn about external variables. Declare a variable int a = 5; inside LCD.c file, to access this variable from other files, use extern int a; in other files. If we add extern int a; to LCD.h file then any file which uses #include “LCD.h” will have access to the variable a. Task : Initialize an integer “a” in LCD.c to 5, declare extern int a; in main.c, increment and print the value of the integer “a” in main.c.

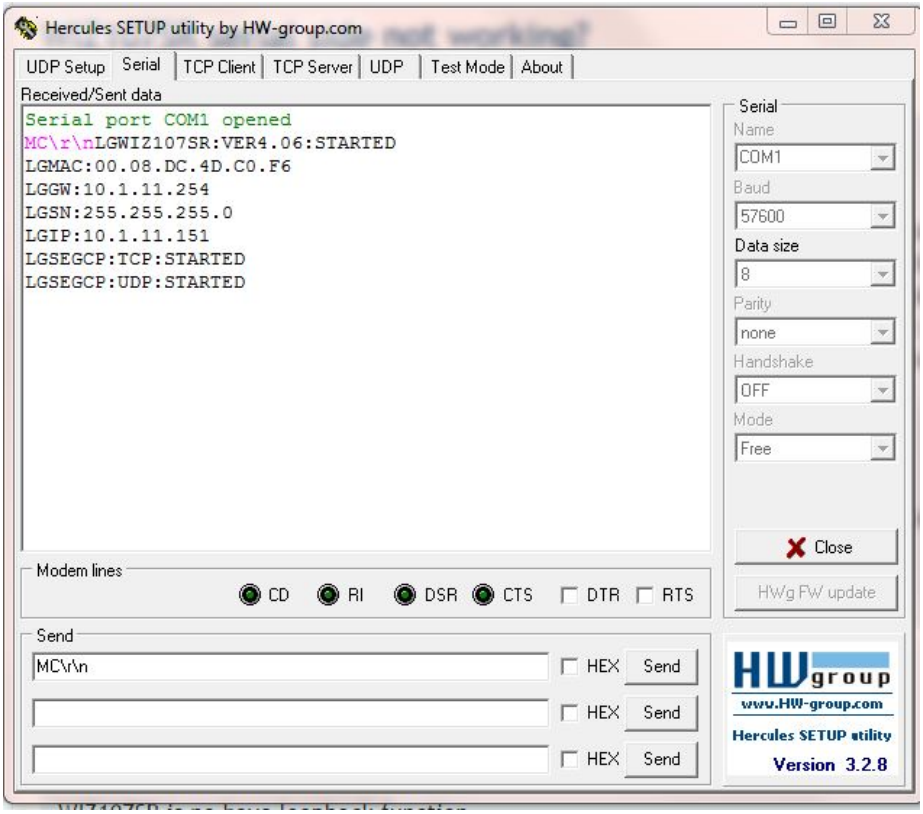
Task 14 : 1602 LCD interfacing

1. Study the datasheet of 16x2 LCD https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf
2. Study the datasheet of LCD controller HD44780 https://www.sparkfun.com/datasheets/LCD/HD44780.pdf
3. Connect LCD to the development board as shown in the image below

4. Write a driver to control a 16x2 LCD using an 8 bit interface (using eight data lines). Implement functions like lcd_init, lcd_set_cursor(row, col), lcd_write(row, col), lcd_clear
5. Repeat the above task using 4 bit interface (using four data lines).
6. Display custom characters on LCD like battery symbol, tick, cross etc.
7. Organize the LCD code in LCD.h and LCD.c files so it can be included and used in future tasks.

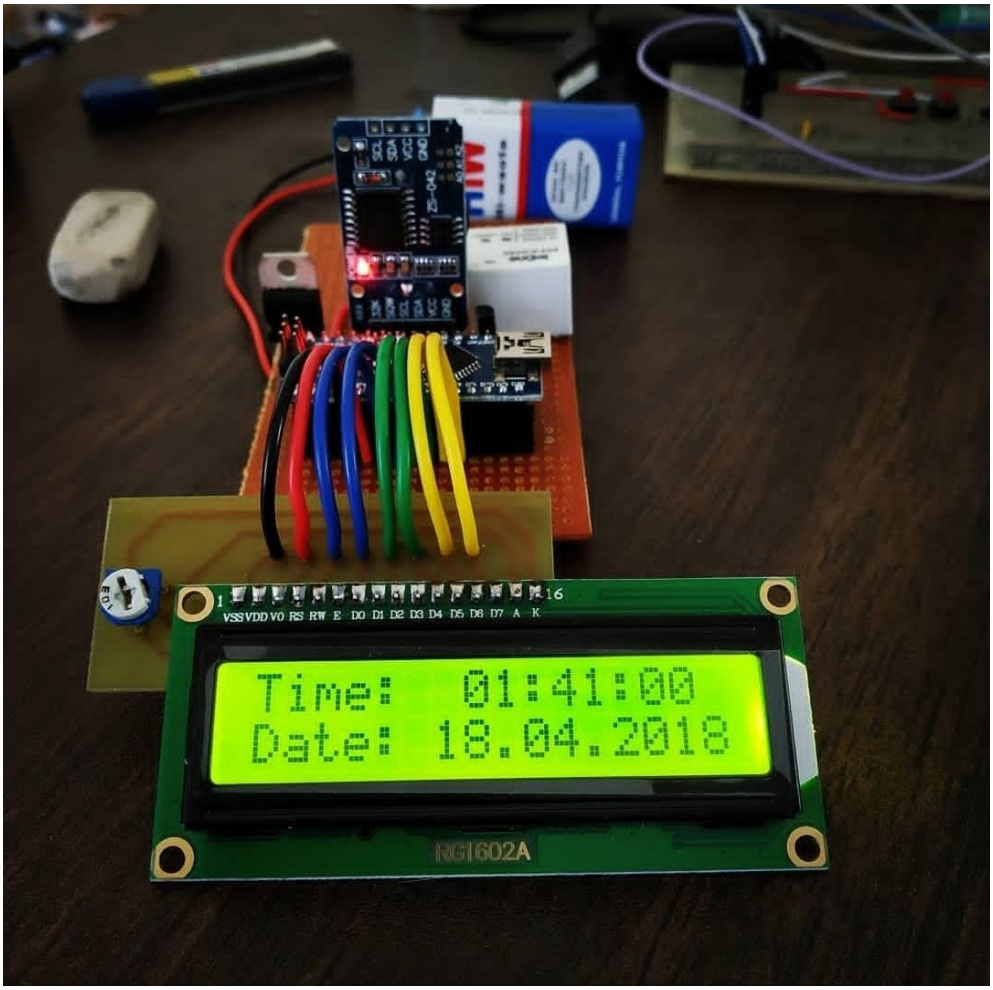
Task 15 : Serial port programming

1. Download Hercules serial terminal software from https://www.hw-group.com/software/hercules-setup-utility

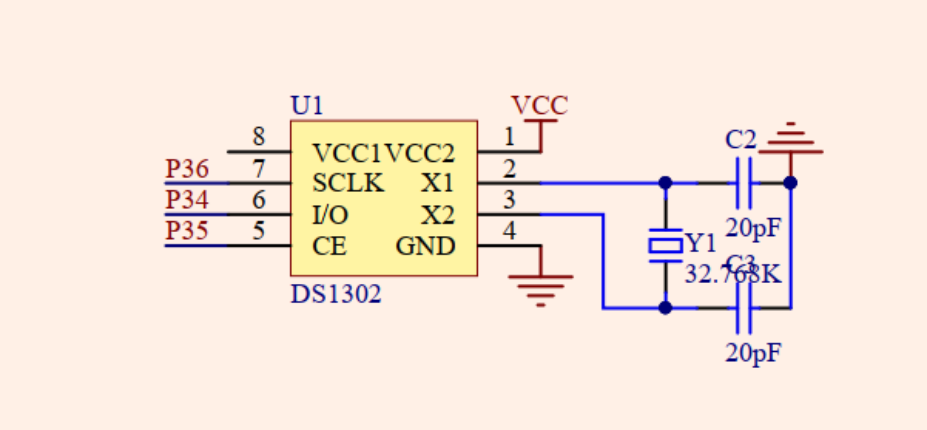


2. Write a program to send data to PC from the microcontroller using serial port at 9600 baud rate. Send any message like “Hello world”. Use \n\r to see how it takes the cursor to a new line on the Hercules software e.g “Hello world\n\r”

3. Send data from PC to the microcontroller. Use \n\r to terminate the message and use if condition in the program to detect when a message has arrived completely.
4. Send time to microcontroller over serial port in the following format
   hh:mm:ss\n\r
   e.g.
   11:58:05\n\r
   In the program, keep receiving bytes from the serial port till a new line sequence (\n\r) is detected. When a newline sequence is detected, stop receiving data and process the received string to parse/extract the values of hour, minute and second from the received string and put these values in variables hour, min and sec.
5. Display the values of hour, minute and second processed in the last step on 16x2 LCD. Additionally you can also send date information via serial port and display on the LCD as shown below. Use LCD.h and LCD.c files to include and use the LCD code.

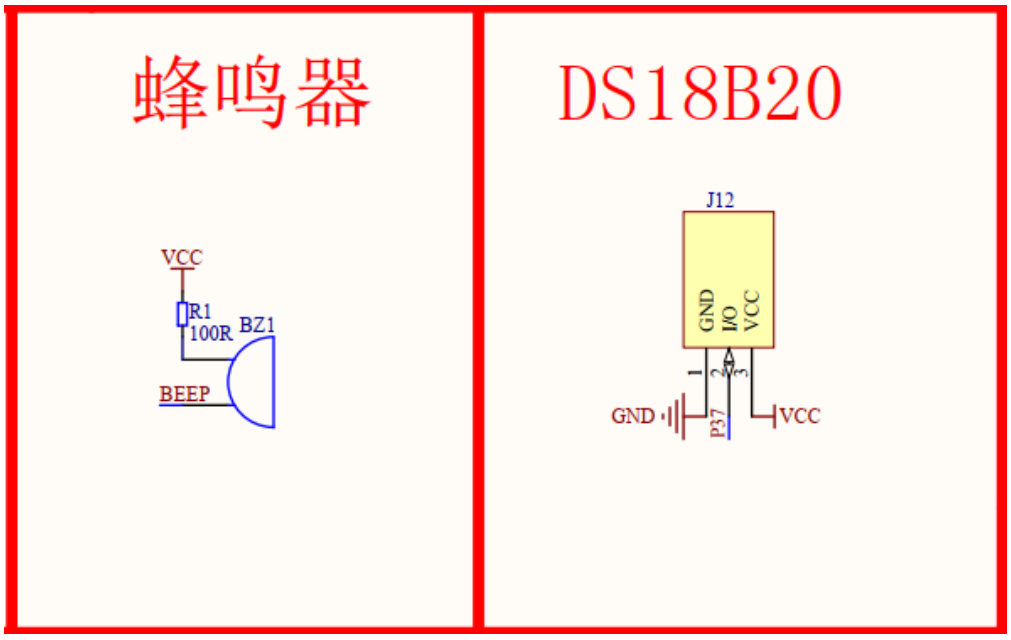


Task 16 : DS1302 RTC interfacing



1. Study the datasheet of RTC DS1302 https://datasheets.maximintegrated.com/en/ds/DS1302.pdf
2. Write a driver for DS1302 to read and write data/time.
3. Read time from DS1302 after every one second and display on 16x2 LCD.
4. Set time on the RTC from PC using Hercules software + serial port.
5. Create DS1302.h and DS1302.c files and organize all the RTC functions and code inside them.

Task 17 : DS18B20 temperature sensor & buzzer interfacing



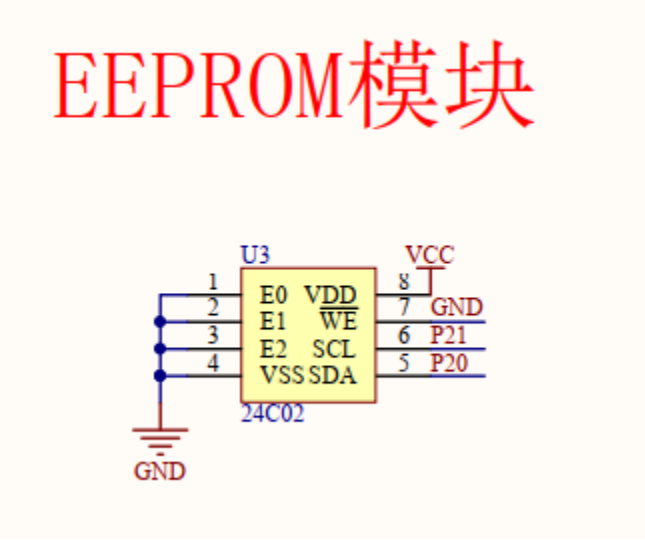
1. Study the datasheet of DS18B20 https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
2. Write a driver for the sensor, read temperature and display on LCD, 7 segment displays and serial port at the same time, use decimal point to display the decimal part of the temperature reading.
3. If the temperature is higher than 37 degrees Celsius, sound the on board buzzer (ON for 500 ms, OFF for 500 ms and repeat). The buzzer present on the development kit is a passive buzzer so you need to generate and supply an AC voltage to it. Use a timer to toggle the buzzer pin at 4000 Hz. The resulting sound will be like this https://www.youtube.com/watch?v=dy5C_58kXEk Watch this video to understand the difference between active and passive buzzers https://www.youtube.com/watch?v=L7H5PiJLeBc
4. Send the temperature threshold value from Serial to STC microcontroller. For example if you send the value 50, the buzzer should sound if the temperature reading goes beyond 50.

Tips:

1. Read temperature every second using timer interrupt.

2. Use your body heat to raise the temperature of the sensor but there is a limit of about 37 degrees Celsius. For higher temperatures you can use a match stick but be careful not to burn anything (use lighter to avoid the smell of match sticks).

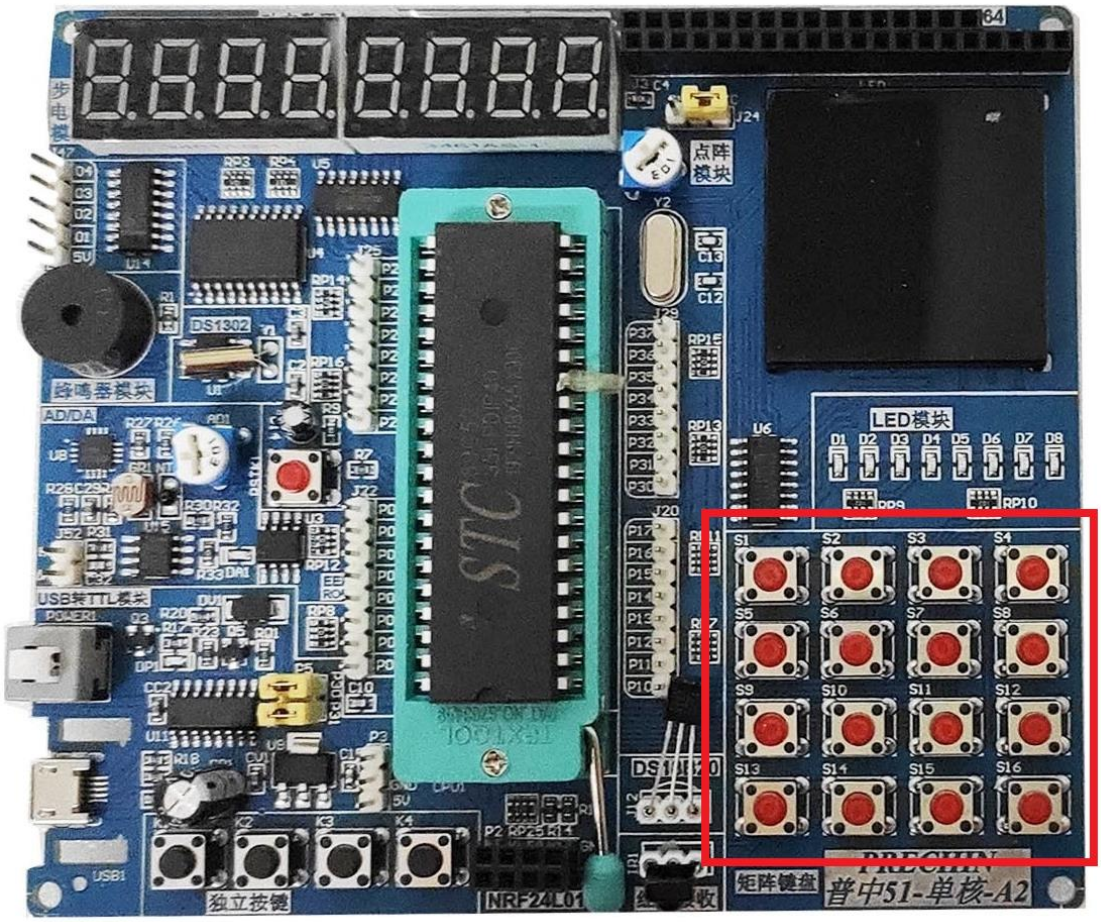
Task 18 : EEPROM interfacing



1. Study the datasheet of the EEPROM IC 24C02 http://ww1.microchip.com/downloads/en/DeviceDoc/21202j.pdf
2. Write a software based I2C driver for the EEPROM which can read/write data from/to the EEPROM.

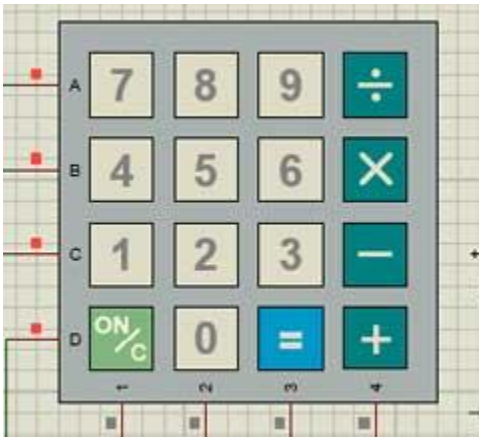
3. Implement a counter of one second (8 bit) and save the value of the counter in EEPROM after each second. Show the count on seven segment displays. On startup, load the value of the counter from EEPROM. Reset the board and see if the counter value remains preserved between resets. An unwritten memory location in EEPROM has a value of 0xFF or 255 which you can use to check if it is a new board or if the value is legitimate.
4. Increase the time between saving counter values to the EEPROM because an EEPROM has limited write cycles and writing too much to it will wear it out quickly. Find the number of write cycles of the EEPROM from the datasheet and calculate the delay between counter value writes to EEPROM if you want the EEPROM to last at least 10 years.
5. Implement a counter of 4 bytes (32 bits) and save it to EEPROM in the same way as done above. Use 4 adjacent memory locations in the EEPROM to save the value of the 4 byte counter. You can use shifts and | operator to assemble a 4 byte value from individual bytes. Similarly to separate the 4 bytes, you can use shift and & operators.

Task 19 : keypad interfacing



Interface the 4x 4 keypad present on the board with STC89C52 microcontroller. Use a non blocking approach so if for example the user keeps a button pressed then it should not block the

code execution. Also add non blocking debouncing. Label the keypad like this (use a piece of paper marked with the following symbols)

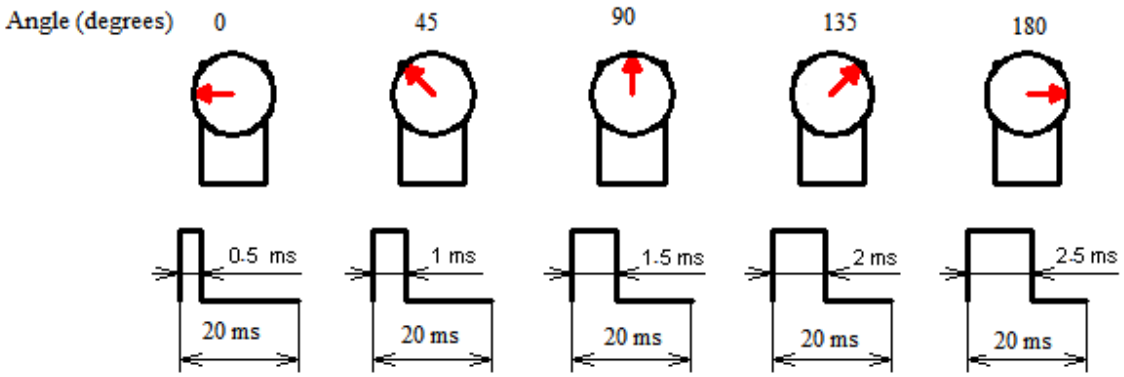


1. Display the pressed key on the LCD and seven segment display.
2. Make a basic calculator using keypad and LCD/seven segment display. The calculator should support addition, subtraction, multiplication and division.
3. Watch these videos to understand the concept of state machines
   https://www.youtube.com/watch?v=TzTl4pdEYWE&ab_channel=5MinutesEngineering
   https://www.youtube.com/watch?v=TM4xTNRH1bw&ab_channel=5MinutesEngineering
   The goal is to run 3 different applications in one program (one application will be active at a time). The applications are **a.** Calculator **b.** Real time clock **c.** Temperature monitor. All of these applications have been made in previous tasks, we just need to combine them in a single program. Use the ON/C button on the keypad to change from application to another e.g. Initially, the calculator app will be active, when the user presses ON/C button on the keypad the application will be changed to real time clock (display time and date read from RTC), again pressing the ON/C button will activate the temperature monitor application. Pressing the ON/C button again will activate the Calculator app and the sequence will continue. First, please draw a state machine diagram of the program just described.
4. Implement the state machine diagram drawn in the above task, use switch case to easily manage the state machine. You can change the value of the variable the switch uses to change the state of the program. Organize the code of different applications in separate files like calculator.c, calculator.h, RTC.c, RTC.h and temp_monitor.c, temp_monitor.h. This will make the code easier to manage.
5. Modify the temperature monitor application so that the user can also enter the temperature threshold value from the keypad.

Task 20 : Servo motor interfacing

1. Watch this video to learn about the working of a servo motor
   https://www.youtube.com/watch?v=1WnGv-DPexc&ab_channel=TheEngineeringMindset

2. SG90 servo motor requires a 50 Hz PWM signal to function. This means that we have to keep sending PWM signals every 20 ms. First, we will do it with a while loop and in the next step with a timer interrupt. The following diagram shows the required pulse timing for achieving a sweep from 0 to 180 degrees.

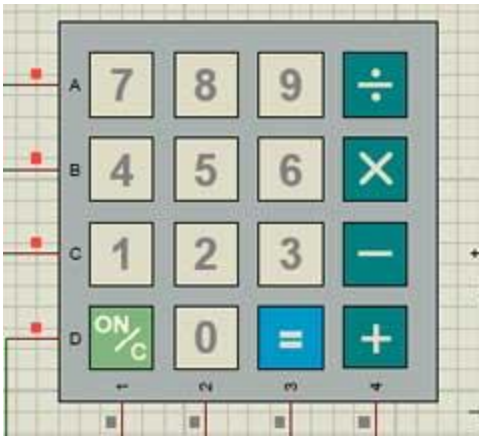


3. Write a program to output logic one on a pin, give a delay of 0.5 ms (500 us), output logic low on the same pin and give a delay of 19.5 ms. Run this code repeatedly in a while loop (make the delay by wasting CPU cycles). The total time of the high + low signal for this program is 0.5+19.5 = 20 ms (50 Hz). You will notice that the SG90 servo motor will be at an angle of 0 degrees. Check the PWM signal using a logic analyzer or an oscilloscope. Now increase the high delay to 1 ms so that the pin is high for 1 ms, low for 19 ms and this sequence repeats. This will result in rotating the arm of the servo motor to 45 degrees. Congratulations, we have achieved a basic PWM signal generator but the problem is that the timings will get disturbed when we add more code to the same program.

4. To solve the timing accuracy problem we can generate the PWM signal by toggling a GPIO pin in a timer interrupt. Configure a timer interrupt significantly faster than 20 ms. Let's say we need a resolution of 1 degree. With a high pulse of 500 us the servo moves to 0 degrees and for a high pulse of 2500 us the servo moves to 180 degrees, so the pulse range is 2500-500 = 2000 us. To move the servo by one degree we need a pulse of 2000 us/180 degrees = 11.11 us/degree. We need to configure the time interrupt for 11.11 us. Create a variable named servo_position (range : 0-180) and initialize it to zero. Control the value of the servo_position variable using keys K1 and K2. Use key K1 to decrease the angle and K2 to increase it. Generate a PWM signal on one of the GPIO pins of STC89C52 microcontroller based on the value set in the servo_position variable by the user. Calculate in the code how many timer ticks you need to keep the GPIO pin high and how many timer ticks you need to keep the GPIO pin low to generate the required PWM signal. Hint: The ON+OFF time of the PWM signal should always be 20 ms (50 Hz). Check the PWM signal using a logic analyzer or an oscilloscope. Let’s take an example; the timer interrupt is configured for 11.11 us, user needs the servo motor arm to move to 45 degrees, the number of timer ticks in this case to toggle the pin is 45 because one tick is one degree (2000 us/180 degrees = 11.11 us/degree). Also display the servo position (in degrees) on the seven segment display/LCD.

5. Supply the PWM signal generated in the previous step to a servo motor. Demonstrate the rotation of the servo motor from 0 to 180 degrees using keys K1 and K2. Make sure you connect the ground of the servo motor with the ground of the STC board.

Task 21 : Digital door lock



Design a digital door lock using keypad for password input and LCD for status.

1. When the program runs for the first time, set the default password as a random 6 digit number (use rand() function to generate). Send the randomly generated password to the serial port so that the workers in the factory know the first time password, we can print this password on the product package. Also save this password to the EEPROM so that the next time the device is turned on the password remains the same as printed on the package.

2. The LCD should display the message “Enter password” on the first line. When the user enters password on the keypad, the second line of the LCD should display what the user

entered e.g. 324618. If the password is correct a servo motor should change its angle from 0 to 180 degrees to unlock the door and the first line of the LCD should display “Door Unlocked” space “time counting after which the door will be locked again” (5 seconds time). The buzzer should short beep (300 ms) if the password is correct. The door should remain unlocked for 5 seconds and should lock again (servo should rotate back to 0 degree) and the message on the LCD should change back to “Enter password”.

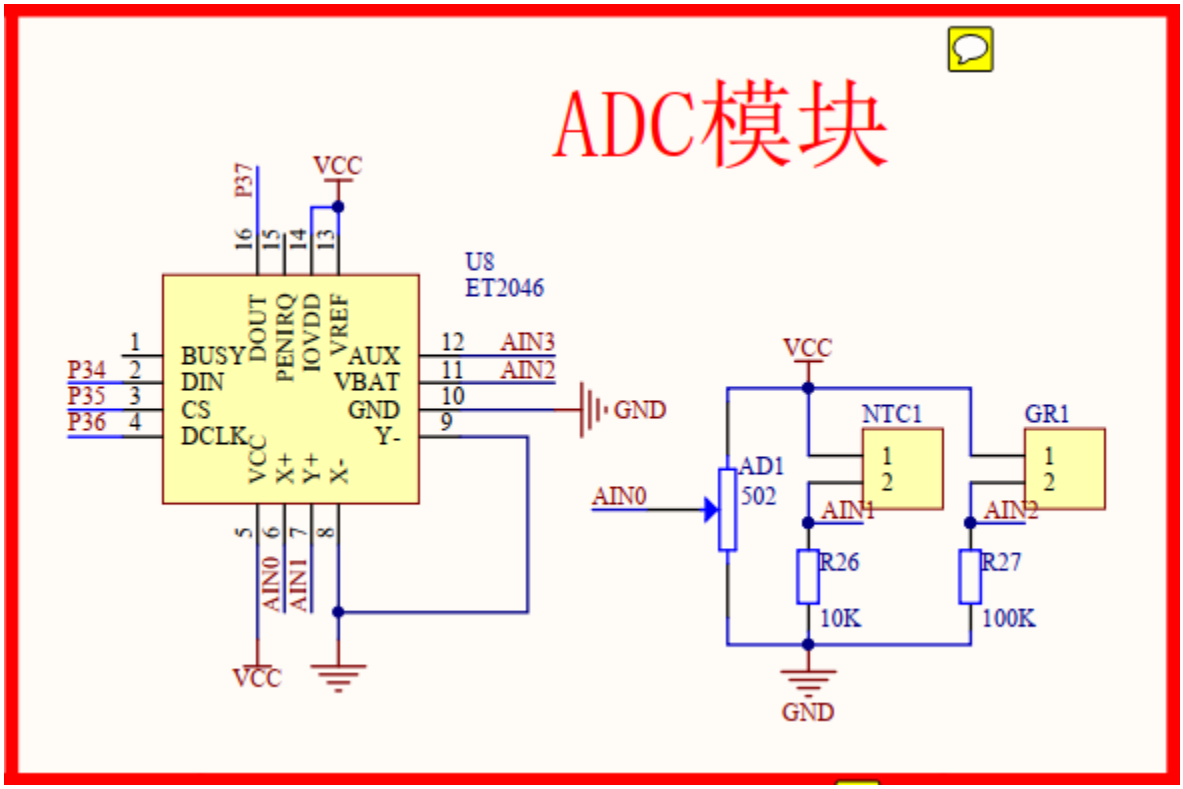
3. If the password entered is wrong, the LCD should display W1, W2 or W3…. to show the number of times wrong password was entered and the buzzer should long beep (1 second). After 5 wrong attempts the LCD should display “Door disabled” and the system should not take keypad input, also the buzzer should beep continuously. The only way to recover from this condition should be to send the device to the factory to reprogram the STC microcontroller (on a newly programmed device erase the EEPROM locations used in your program, more on this in the hint section). The number of wrong attempts should be saved to EEPROM so turning off power to the device has no effect.

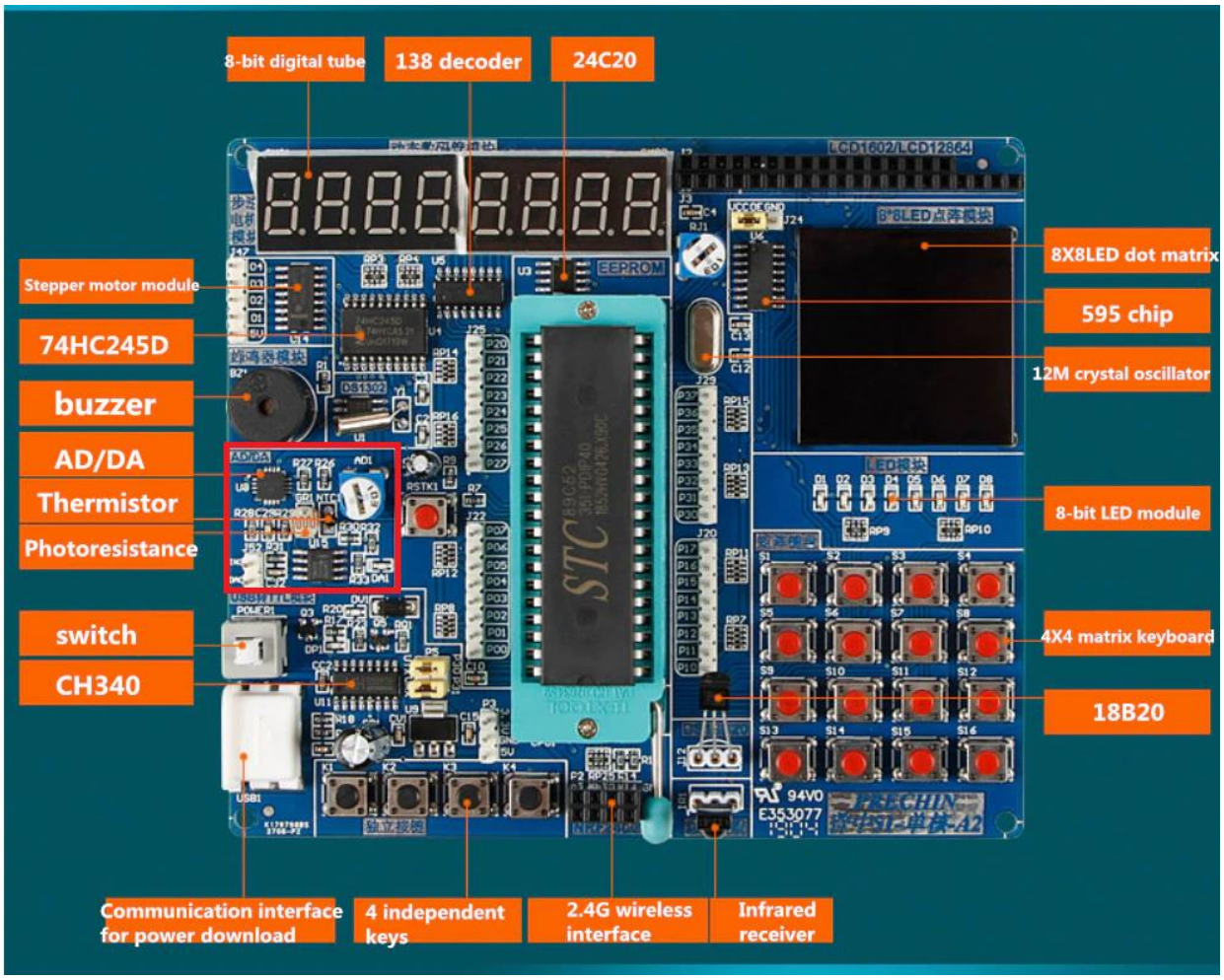
4. The user should be able to change the password by long pressing the ON/C key for 10 seconds. After pressing the ON/C key for 10 seconds the device should prompt the user to enter the old password. If the old password is entered correctly the user should enter change password mode otherwise the device should go back to the main screen after short beeping the buzzer 3 times and saving the number of times wrong old password was entered, also display the number of wrong attempts on the Enter old password screen as C1, C2, C3. After three wrong attempts the keypad should ignore any further key presses and the buzzer should beep non stop. In the change password mode the LCD should display “Enter new pass” on the first line and the second line should show the number that the user is entering. The new password should be saved to EEPROM when 6 digits are entered by the user. This new password should not be sent to the serial port otherwise it will be a big vulnerability.

Hint: Use state machine (implement using switch case) with states as NEW_DEVICE, ENTER_PASS, DISABLED, NEW_PASS etc. to make the code manageable.

A newly programmed STC microcontroller has 0xFF in the flash memory locations not used by the program. You can reserve a memory location so that it is not used by the program. When the device is newly programmed this location will change to 0xFF no matter its previous value. When the program runs, check this location and if it is 0xFF then perform what needs to be done the first time (like generating a random password, sending to the serial port and saving to EEPROM, clearing EEPROM locations to be used by the program so any garbage/previous data gets erased). After doing this, change the value of the reserved flash memory location to something like 0, 0x55 or 0xAA. The next time the device is powered off and on, the program will check the same location and if its value is not 0xFF then it will know that the program is not running the first time so it should not generate the random password etc.

Task 22 : External ADC interfacing





STC89C52 doesn't has a built-in/internal ADC so we will use an external ADC to read analog voltages. Download from http://www.ruikang.net/upload/datasheet/ET2046.pdf or search for the datasheet of ET2046 chip on Google, you will find a datasheet in Chinese language. Download and translate the pdf to English using this website https://www.onlinedoctranslator.com/en/ This is a very common problem with datasheets being in Chinese, you will face it many times in your career but the solution is to use the document translator website. Sometimes, datasheets of Chinese parts are not available on Google, in that case you can search on Baidu (Chinese search engine) or request the hardware/PCB designer to provide the datasheet.

Study the datasheet of ET2046. ET2046 was designed for use in early mobile phones/tablets to control resistive touch and read battery voltage. It essentially is a successive approximation register analog to digital converter (SAR ADC).

1. **Develop a driver** in C language to read data from the external ADC ET2046.

2. **Digital voltmeter design :** Vary the potentiometer AD1 on the development kit and display the voltage that is present on the AIN0 pin (the middle pin of the potentiometer) on the LCD. The voltage should vary from 0-5 volts. Compare the reading on the LCD with

the reading of the voltage you are getting with a voltmeter/DMM. The readings should match or the difference in readings should be very small.

3. **Thermometer design :** There is a thermistor attached to the AIN1 input of the ADC. Read the voltage on AIN1 pin, convert it into temperature in degrees Celsius and display on the LCD next to potentiometer voltage.

4. **Light meter :** An LDR is connected on pin AIN2 of the ADC. Display the relative intensity of light on the beginning of 2nd line of the LCD such that all three readings: potentiometer voltage, temperature and relative light intensity are visible. Scale the LDR reading such that for complete darkness it is 0 and for very bright light it is 100 (use a bright flashlight very near to the LDR for testing). **Advanced learning :** LDR can be calibrated to make a lux meter but it is an advanced topic so we will not implement it, you can implement it if enough time is available, read more here https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/

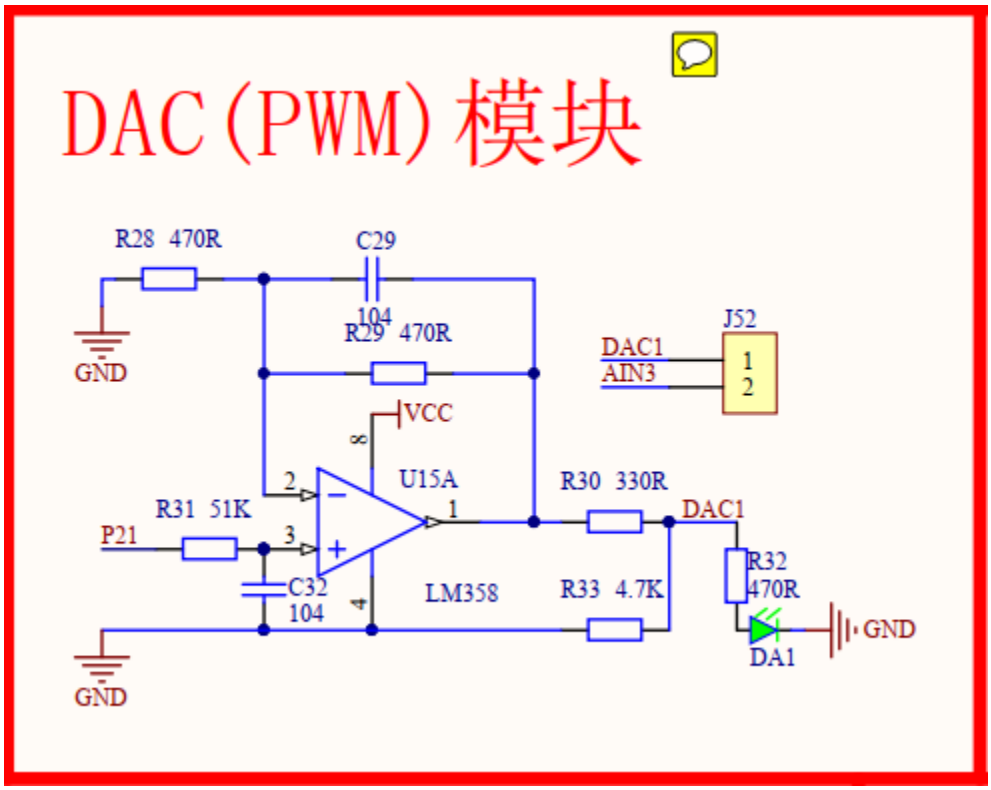
5. **Automatic Air Conditioning controller :** Design an AC controller which turns the AC on/off only if it is day time. Use 26 degrees Celsius as the lower point at which the AC should turn off and 29 degrees Celsius as the upper point where the AC should turn on. The temperature controller should work only during day time when people are present in the office/building (use LDR to detect if it is day or night). Display the message **AC ON** or **AC OFF** on the right side of the LCD on the 2nd line such that other readings potentiometer, temperature and LDR are also shown as done previously. Keep the AC turned off during night time.

6. Write another program to alert the user if the temperature is too high. Use the potentiometer AD1 to set a temperature point. Show both the set temperature and the current temperature on the LCD. The buzzer should beep with 300 ms beeps if the temperature read by the thermistor is higher than the set point. The voltage reading of potentiometer in the range of 0-5 volts can be scaled for temperature in the range of 20-40 degrees Celsius (0 should be scaled to 20 degrees and 5 volts to 40 degrees Celsius).

7. **Solar load management :** There is a well known problem with solar energy when used to power heavy electrical devices like air conditioners, heaters, water pumps etc. The issue is that when sunlight is available then solar inverter can easily run the heavy load if the rating of the inverter and the number of solar panels is appropriate. When at night the sunlight goes away then the inverter manages to run the heavy load from mains electricity, if in case mains electricity gets interrupted due to load shedding or a fault, the inverter will continue to run the heavy load by using energy stored in batteries. This will result in discharging the batteries quickly because most solar energy systems in Pakistan use tubular batteries which don't have enough energy storage capability. To prevent this problem we can design an intelligent load controller which will check if mains electricity fails and it is night time then it will disconnect the heavy load using a contactor (watch this

video to learn more about contactors
https://www.youtube.com/watch?v=tHml7C5diHo&ab_channel=DeepakkumarYadav)
The task is to check if mains electricity is present (emulated by AD1 potentiometer). If the potentiometer voltage is higher than 4 volts, we will take it as electricity present and if it is lower than 4 volts then we will consider it as mains electricity failure. In case if it is night time (detect using LDR) and the mains electricity fails then open the contactor to disconnect power delivery to heavy load. Use an LED to show if the contractor is closed or open.

Task 23 : DAC interfacing (Generate a DC voltage using PWM)



There is a low pass filter circuit present in the STC development kit. The circuit is built around LM358. It can convert a 0-100% PWM signal on pin P21 to a DC voltage of 0-5 volts.

1. Watch this video to understand the concept
   https://www.youtube.com/watch?v=6rDwfFuAUD0&ab_channel=EngrEdu
2. Write a function which takes 0-100 as input and generates a PWM signal on pin P21.
3. Observe the PWM signal with a logic analyzer or an oscilloscope. The DC voltage on the point DAC1 in the above circuit should vary by varying the duty cycle of the PWM signal.

Observe the DC voltage with a voltmeter. The brightness of LED DA1 should change with varying the PWM signal.

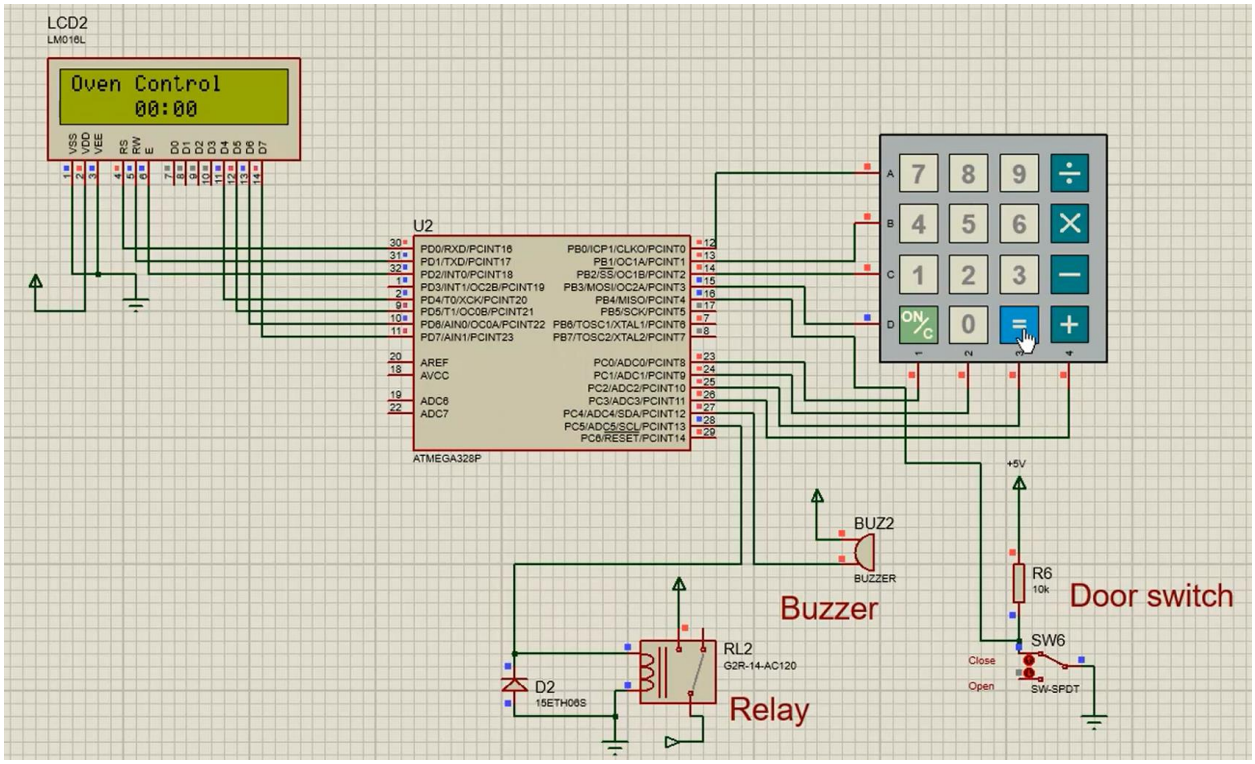
4. Close the J52 jumper, doing so will supply the DC voltage generated by the DAC circuit to the AIN3 input of the ADC. Read the voltage generated by the DAC and display it on the LCD.
5. **Basic DC Power supply design :** Use key K1 to decrease the PWM duty cycle and K2 to increase it. Show the output voltage of DAC on the LCD and digital voltmeter. We just designed a very basic DC power supply that we can use to power small electronic circuits.
6. **Advanced DC Power supply design :** Use the keypad on the STC kit to enter a value from 0-5 (including decimal points) e.g. if the user enters 2.3 on the keypad, the output of the DAC circuit should supply 2.3 volts DC power. Be careful not to connect heavy loads with this supply because its current capability is very small. Commercial power supplies use transistors on the output stage to amplify the output current.
7. **Waveform generator :** Generate sine and sawtooth waveforms using the DAC circuit. Show the waveforms on oscilloscope.

Task 24 : Application => Microwave oven timer

Implement a typical Microwave oven timer. The time for which the oven should run should be entered on the 4x4 keypad present on the STC development kit. Status should be visible on a 16x2 LCD (Hitachi HD44780 driver). The oven should be started or paused using the ON key. Time entered can be cleared by pressing the = key on the keypad. Microwave radiation is harmful to humans so we have to use a safety door switch in the circuit. Opening the door switch stops the oven, then after closing the door the oven can be resumed by pressing the ON key. A buzzer is used to indicate various events like key presses, time out and warnings. Use an electromagnetic relay to control the power of the magnetron. Use a finite state machine to implement the program.

The final product should look like the device in this video, you can additional features if you want

https://www.youtube.com/watch?v=oVj2MSlzRCo&ab_channel=ChipSoulTechnology%28SMC-Private%29Limited



Task 25 : Software RTC

There are some cases where we can't or don't want to add an RTC chip to the system. For example, to save cost or if there aren't enough pins available on the microcontroller or the space on PCB is limited and timing accuracy is not a concern.
Implement a software based real time clock in such a way that it can keep time if power is available. In case of power loss or system reset the time will be lost but it is not a concern because we use software RTC in systems which have time sync function available e.g. time sync from the internet or from mobiles devices using BLE/Bluetooth.

1. Run a timer for one second interrupt. You can use counters to count one second.

2. Make a structure called time or __time because the **time** keyword might be reserved by the compiler and can give errors on compilation. The struct should contain variables from second to year.

3. In the timer ISR, increment the second variable. If the second variable is equal to 60 after incrementing then make it 0 and increment the minute variable of the struct. Similarly, do the same for minute, hour, day, month and year. Compensate February for 29 days in leap years.

There is a formula to find if a year is leap or not. You can use an array for reference to how many days are in a month.

4. Display the time from software RTC on 16x2 LCD. The screen should update each second and there should be no flickering.

5. Port the code from previous tasks so that you can set the time on software RTC using Serial port from PC.

6. Modify the program so that on the first line of the 16 x 2 LCD the time from software RTC is visible and on the second line the time from DS1302 is visible. Sync your PC time with NTP server, then set the time on both RTCs from the PC using Serial and leave the system running for at least 24 hours. On the next day, sync your PC with NTP server again and compare the drift in time of both RTCs with the time on your PC. NTP server time is very accurate so it will give a good idea of error in both of the RTCs. This video explains how to sync PC time with server https://www.youtube.com/watch?v=xPEKM3zcOaA

7. There are techniques to compensate RTC for error, for example by calibrating the software RTC from an external accurate timing signal and temperature drift compensation. The 11.0592 MHz crystal on the STC development kit is not 100% accurate, it has an offset which keeps changing with temperature. We can measure the temperature and can count less or more depending upon temperature to increment a second. For example, we can count 990 ms or 1020 ms to increment the second variable depending upon the crystal frequency being higher or lower than the rated frequency. This is a time consuming and complex task and its testing takes days so we won't implement RTC error compensation in this training. You can give it a try if you want.

Task 26 : Frequency counter

Implement a frequency counter using timer and external interrupts. Use a 16 bit timer so we can hold more counts and thus be able to measure low frequencies as well. Use a frequency generator to supply a square wave signal on the INT0 pin of STC89C52 microcontroller. Please make sure that you are using the TTL output (5V max amplitude) of the frequency generator otherwise it will fry the board. Also, connect the GND of the frequency generator to the board GND. Seven segment display is preferred to display the measured frequency.

1. Enable external interrupt on a capable pin.

2. When the rising edge of the input signal arrives it will trigger the external interrupt. In the ISR, start a timer so it counts time.

3. When the next rising edge arrives, stop the timer. Read the count value and depending upon the clock frequency, calculate the frequency of the input signal. For this, you will need to calculate the clock that is running the timer.

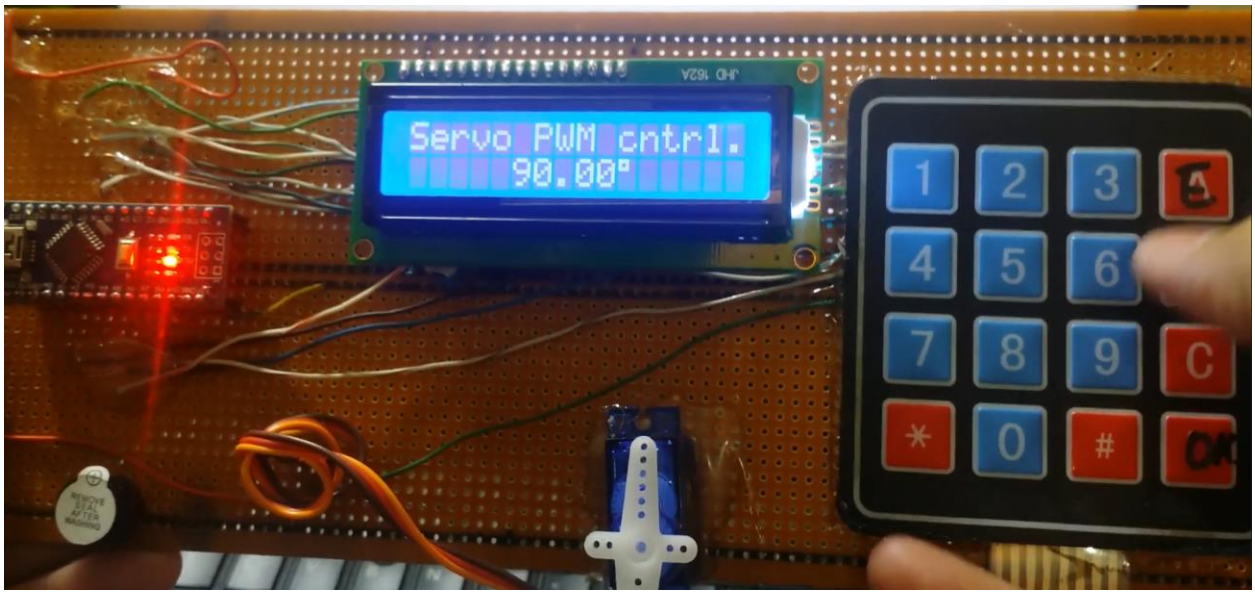
4. If the input signal is very slow, then the timer will overflow some time after the first rising edge arrives. In case of an overflow, show "Error" message on the LCD or "--------" on the seven segment LED display.

5. Check practically the frequency range that can be measured with your device and show the theoretical reasons the upper and lower limits exist.

Task 27 : Servo motor controller

Move the arm of a servo motor to a desired position. Enter the desired angle using the 4x4 keypad present on the STC development kit. The buzzer should make a short beep sound as a feedback when any key is pressed on the keypad.Use a 16x2 LCD to show the current angle and also use the same screen to enter a new angle. The arm position should be entered in the range of 0-180 degrees. If the entered angle is outside of this range then the program should ignore it and sound a long beep on the buzzer to indicate error. Please do not use delay to make beeps instead use timer interrupt and flags so there is no blocking in the code. The final product should look like this or better:

https://www.youtube.com/watch?v=CHUjnqX7LyA&ab_channel=ChipSoulTechnology%28SMC-Private%29Limited



Future tasks

CPU usage calculation
Data types uint8_t, uint16_t
Main.h file and file inclusion
Structs and unions
Memory overlap using unions
Saving memory by using bit fields
Water pump controller
Temperature controller
Temperature controlled fan using PWM
Speed trap
short beep when a key is pressed on the keypad
-LED matrix task
--NRF24L01
-LCD12864
-Five-wire four-phase stepper motor ULN2003D, DC motor (PWM)
-Infrared receiver module
Hardware
-DC5V-->3.3V study
-Crystal oscillator circuit
-Reset circuit study
-USB to TTL circuit study