INTERNATIONAL UNIVERSITY

BIOMEDICAL ENGINEERING

**MICRO\_ELECTRONIC DEVICE**

**BM062IU**

FINAL REPORT

MPS PROJECT HARD

**Submitted by**

Võ Quang Trấn - BEBEIU17008

Nguyễn Hoàng Tân - BEBEIU17032

Đinh Hoàng Sáng - BEBEIU17022

Date Submitted:

Date Performed:

Lab Section:

Course Instructor: M.ENG MINH TRIET NGUYEN

**GRADING GUIDELINE FOR LAB REPORT**

| **Number** | **Content** |  | **Score** | **Comment** |
| --- | --- | --- | --- | --- |
| 1 | **Format (max 9%)** | |  |  |
| * Font type | Yes No |  |
| * Font size | Yes No |  |
| * Lab title | Yes No |  |
| * Page number | Yes No |  |
| * Table of contents | Yes No |  |
| * Header/Footer | Yes No |  |
| * List of figures (if exists) | Yes No |  |
| * List of tables (if exists) | Yes No |  |
| * Lab report structure | Yes No |  |
| 2 | **English Grammar and Spelling (max 6%)** | |  |  |
| * Grammar | Yes No |  |
| * Spelling | Yes No |  |
| 3 | **Data and Result Analysis (max 85%)** | |  |  |
| **Total Score** | |  | |  |

Signature:

Date:

**Table of Contents**

List of Figures ........................................................................................….….............................. 3

List of Tables .................................................................................................……....................... 3

Introduction .................................................................................................……....................... 4

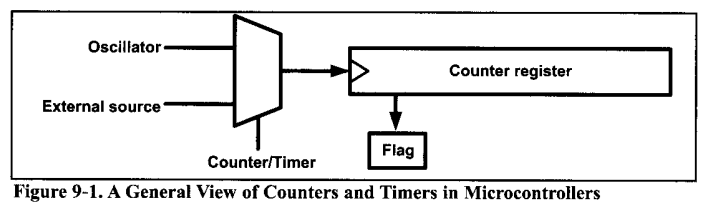
Theoretical Background .............................................................................................…….......... 4

Experimental Procedure.............................................................................................……........... 5

**Theoretical Background**

**1/Timer**

Many applications need to count an event or generate time delays. So, there are counter registers in microcontrollers for this purpose. See Figure 9-1. When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred. When we want to generate time delays, we con­nect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter reg­ister represents how many ticks have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can cal­culate the tick period, and from the content of the counter register we will know how much time has elapsed.



So, one way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number For example, consider a microcontroller with an oscillator with frequency of 1 MHz; in the microcon­troller, the content of the counter register increments once per microsecond. So, if we want a time delay of 100 microseconds, we should clear the counter and wait until it becomes equal to 100.

In the microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software. The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set. For example, in a microcontroller with a frequency of 1 MHz, with an 8-bit counter register, if we want a time delay of 3 microsec­onds, we can load the counter register with $FD and wait until the flag is set after 3 ticks. After the first tick, the content of the register increments to $FE; after the second tick, it becomes $FF; and after the third tick, it overflows (the content of the register becomes $00) and the flag is set.

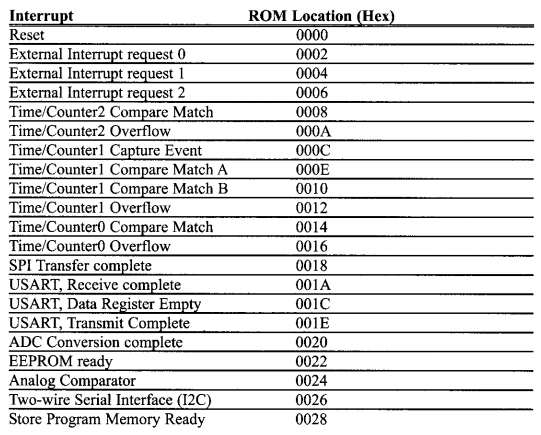
The AVR has one to six timers depending on the family member. They are referred to as Timers 0, 1,2, 3, 4, and 5. They can be used as timers to generate a time delay or as counters to count events happening outside the microcontroller.

In the AVR some of the timers/counters are 8-bit and some are 16-bit. In ATmega32, there are three timers: TimerO, Timerl, and Timer2. TimerO and Timer2 are 8-bit, while Timerl is 16-bit. In this chapter we cover TimerO and Timer2 as 8-bit timers, and Timerl as a 16-bit timer.

If you learn to use the timers of ATmega32, you can easily use the timers of other AVRs. You can use the 8-bit timers like the TimerO of ATmega32 and the 16-bit timers like the Timerl of ATmega32.

**2/Interrupt**

A single microcontroller can serve several devices. There are two methods by which devices receive service from the microcontroller: interrupts or polling. In the interrupt method, whenever any device needs the microcontroller’s service, the device notifies it by sending an interrupt signal. Upon receiving an interrupt signal, the microcontroller stops whatever it is doing and serves the device. The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler. In polling, the microcontroller continuously monitors the sta­tus of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the microcontroller. The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time, of course); each device can get the attention of the microcon­troller based on the priority assigned to it. The polling method cannot assign pri­ority because it checks all devices in a round-robin fashion. More importantly, in the interrupt method the microcontroller can also ignore (mask) a device request for service. This also is not possible with the polling method. The most important reason that the interrupt method is preferable is that the polling method wastes much of the microcontroller’s time by polling devices that do not need service. So interrupts are used to avoid tying down the microcontroller. For example, in dis­cussing timers in Chapter 9 we used the bit test instruction “SBRS R2 0, TOV0” and waited until the timer rolled over, and while we were waiting we could not do anything else. That is a waste of microcontroller time that could have been used to perform some useful tasks. In the case of the timer, if we use the interrupt method, the microcontroller can go about doing other tasks, and when the TOVO flag is raised, the timer will interrupt the microcontroller in whatever it is doing.



**3/LCD**

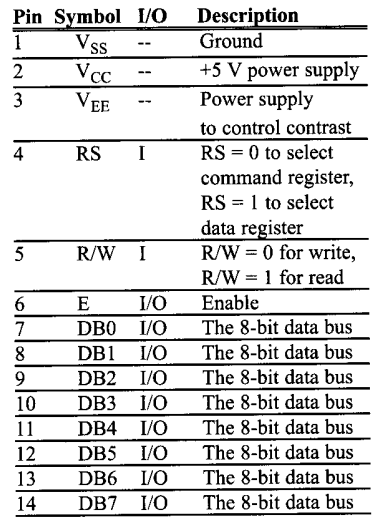
In recent years the LCD is finding widespread use replacing LEDs (seven- segment LEDs or other multisegment LEDs). This is due to the following reasons:

1. The declining prices of LCDs.

2. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.

3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.

4. Ease of programming for characters and graphics.



**Experimental Procedure**

**Question :** Design an embedded system using ATmega32 to design the digital clock,the system use port D to interface with LCD.Port C to interface with 7 buttons.Assume that,the XTAL clock=8 MHz.Develop the program Using Timer0 interrupt ISR(TIMERO\_OVF\_vect)generated in 100ms interval,normal mode,with prescaler=1024.

a) The LCD to display the digital clock in format hh:mm:ss.xx in the first line.Use 3 buttons to adjust the current time hh:mm:ss.

b) The second line of LCD display the alarm time which can be adjusted by other 3 buttons.When the clock and alarm time are equal,turn on the Buzzer which is connected to PORTB.7.The buzzer will be turn off by pressing a button.

**Code:**

/\* Võ Quang Trấn, Nguyễn Hoàng Tân, Đinh Hoàng Sáng \*/

/\* Atmega32-finalproject-Clock and Alarm in AVR programer-C language \*/

#include <avr/io.h> //header to enable data flow control over pins

#define F\_CPU 8000000UL // XCrytal=8Mhz //telling controller crystal frequency attached

#include <util/delay.h> //header to enable delay function in program

#include <stdlib.h> //some functions to calculate

#include <avr/interrupt.h> //initialize timer and interrupt

#include "lcd.h" //LCD 4-bit control

#include <stdbool.h> //set boolean

static volatile uint8\_t MSEC=0; // allocating unsigned integer-8bit memory for storing 0.1 seconds

static volatile uint8\_t SEC =0; // allocating unsigned integer-8bit memory for storing seconds

static volatile uint8\_t MIN =0; // allocating unsigned integer-8bit memory for storing minutes

static volatile uint8\_t HOU =0; // allocating unsigned integer-8bit memory for storing hours

static volatile uint8\_t count=0; // to count 10ms

int main(void)

{

LCDInit(LS\_NONE); // the cursor don't display

DDRC = 0x00; // PortC is input

PORTC = 0xFF; // PortC is all "high"

DDRB = 0x80; // PortB.7 is output ( buzzer)

PORTB = 0x7F; // PortB.7 is "low"

PORTD = 0x00; // PortD is all "low"

//F.timer =CPU\_F/Pre-scalar=8Mhz/1024=7812.5 Hz

//T.Tick = 1/F.timer = 1/(7812.5)= 128 us

//T.Tick\_Count = 10ms/128us= 78.125

//TCCNT for 16 bit = 256-78.125 = 177.875

TCNT0=178; //10ms

TCCR0= (1<<CS00) | (1<<CS02); //prescarlar 1024

TIMSK = (1<<TOIE0); //enabling Timer1

sei(); //enabling global interrupts

char SHOWSEC [2]; //Show characters of Seconds

char SHOWMIN [2]; //Show characters of Minutes

char SHOWHOU [2]; //Show characters of Hours

char SHOWMSEC[1]; //Show characters of 100ms

uint8\_t ALSEC = 0; // Alarm\_second unit

uint8\_t ALMIN = 0; // Alarm\_Minute unit

uint8\_t ALHOU = 0; // Alarm\_Hour unit

char SHOWALSEC [2]; //Show characters of Alarm\_second

char SHOWALMIN [2]; //Show characters of Alarm\_minute

char SHOWALHOU [2]; //Show characters of Alarm\_hour

bool status = false; // set ON/OFF ALARM

LCDClear(); // clear LCD display

while(1)

{

//first line

itoa(HOU/10,SHOWHOU,10); // Show dozen of Hour

LCDWriteString(SHOWHOU);

itoa(HOU%10,SHOWHOU,10); // Show unit of Hour

LCDWriteString(SHOWHOU);

LCDWriteString(":");

LCDGotoXY(3,0);

itoa(MIN/10,SHOWMIN,10); // Show dozen of minute

LCDWriteString(SHOWMIN);

itoa(MIN%10,SHOWMIN,10); // Show unit of minute

LCDWriteString(SHOWMIN);

LCDGotoXY(5,0);

LCDWriteString(":");

LCDGotoXY(6,0);

itoa(SEC/10,SHOWSEC,10); // Show dozen of second

LCDWriteString(SHOWSEC);

itoa(SEC%10,SHOWSEC,10); // Show unit of second

LCDWriteString(SHOWSEC);

LCDWriteString(".");

LCDGotoXY(9,0);

itoa(MSEC,SHOWMSEC,10); // Show unit of 100ms

LCDWriteString(SHOWMSEC);

LCDGotoXY(11,0);

//show ON/OFF status of ALARM

if ((PINC==0b01111111) && (status==false)) //If PINC.7 pressed then toggle the status

{

status = true;

\_delay\_ms(400);

}

if ((PINC==0b01111111) && (status==true))

{

status = false;

\_delay\_ms(400);

}

if (status==true) //if "ON mode" and Clock equal Alarm then trigger buzzer else turn off buzzer

{

LCDWriteString("ON ");

if ((ALHOU==HOU)&(ALMIN==MIN)&(ALSEC==SEC))

{

PORTB|=(1<<PINB7); // buzzer on

}

}

else

{

LCDWriteString("OFF");

PORTB &= ~(1<<PINB7); // buzzer off

}

//second line, same first line

LCDGotoXY(0,1);

LCDWriteString("ALARM:"); // print ALARM

LCDGotoXY(7,1);

itoa(ALHOU/10,SHOWALHOU,10);

LCDWriteString(SHOWALHOU);

itoa(ALHOU%10,SHOWALHOU,10);

LCDWriteString(SHOWALHOU);

LCDGotoXY(9,1);

LCDWriteString(":");

LCDGotoXY(10,1);

itoa(ALMIN/10,SHOWALMIN,10);

LCDWriteString(SHOWALMIN);

itoa(ALMIN%10,SHOWALMIN,10);

LCDWriteString(SHOWALMIN);

LCDGotoXY(12,1);

LCDWriteString (":");

LCDGotoXY(13,1);

itoa(ALSEC/10,SHOWALSEC,10);

LCDWriteString(SHOWALSEC);

itoa(ALSEC%10,SHOWALSEC,10);

LCDWriteString(SHOWALSEC);

LCDGotoXY(0,0);

// Control "Clock"

if (PINC==0b11111110) // PinC.0 pressed then second value is increased 1 unit

{

if (SEC<60)

{

SEC++;

\_delay\_ms(400);

}

if (SEC==60) // if second value = 60 then minute value is increased 1 unit

{

if (MIN<60)

{

MIN++;

}

SEC=0;

\_delay\_ms(400);

}

}

if (PINC==0b11111101) // PinC.1 pressed then minute value is increased 1 unit

{

if (MIN<60)

{

MIN++;

\_delay\_ms(400);

}

if (MIN==60) // if minute value = 60 then hour value is increased 1 unit

{

if (HOU<24)

{

HOU++;

}

MIN=0;

\_delay\_ms(400);

}

}

if (PINC==0b11111011) // PinC.2 pressed then hour value is increased 1 unit

{

if (HOU<24)

{

HOU++;

}

\_delay\_ms(400);

if (HOU==24) // if hour value = 24 then hour value equal 0

{

HOU=0;

}

}

//Control "ALARM", same "CLOCK"

if (PINC==0b11110111) // button alarm second

{

if (ALSEC<60)

{

ALSEC++;

\_delay\_ms(400);

}

if (ALSEC==60)

{

if (ALMIN<60)

{

ALMIN++;

}

ALSEC=0;

\_delay\_ms(400);

}

}

if (PINC==0b11101111) // button alarm minute

{

if (ALMIN<60)

{

ALMIN++;

\_delay\_ms(400);

}

if (ALMIN==60)

{

if (ALHOU<24)

{

ALHOU++;

}

ALMIN=0;

\_delay\_ms(400);

}

}

if (PINC==0b11011111) // button hour alarm

{

if (ALHOU<24)

{

ALHOU++;

}

\_delay\_ms(400);

if (ALHOU==24)

{

ALHOU=0;

}

}

}

}

ISR(TIMER0\_OVF\_vect) // Interrupt Overflow each 100ms

{

count++;

TCNT0=178;

if (count == 10)

{

// low byte

if (MSEC<10) // increase 100ms variable

{

MSEC++;

}

if (MSEC==10) // 10 times of 100ms, second variable is increased 1

{

if (SEC<60)

{

SEC++;

}

MSEC=0; // return 100ms variable is 0

}

if (SEC==60) // if second variable is 60, minute variable is increased 1

{

if (MIN<60)

{

MIN++;

}

SEC=0; // return second variable is 0

}

if (MIN==60) // if minute variable is 60, hour variable is increased 1

{

if (HOU<24)

{

HOU++;

}

MIN=0; // return minute variable is 0

}

if (HOU==24)

{

HOU=0; // if hour variable is 24, it equals 0

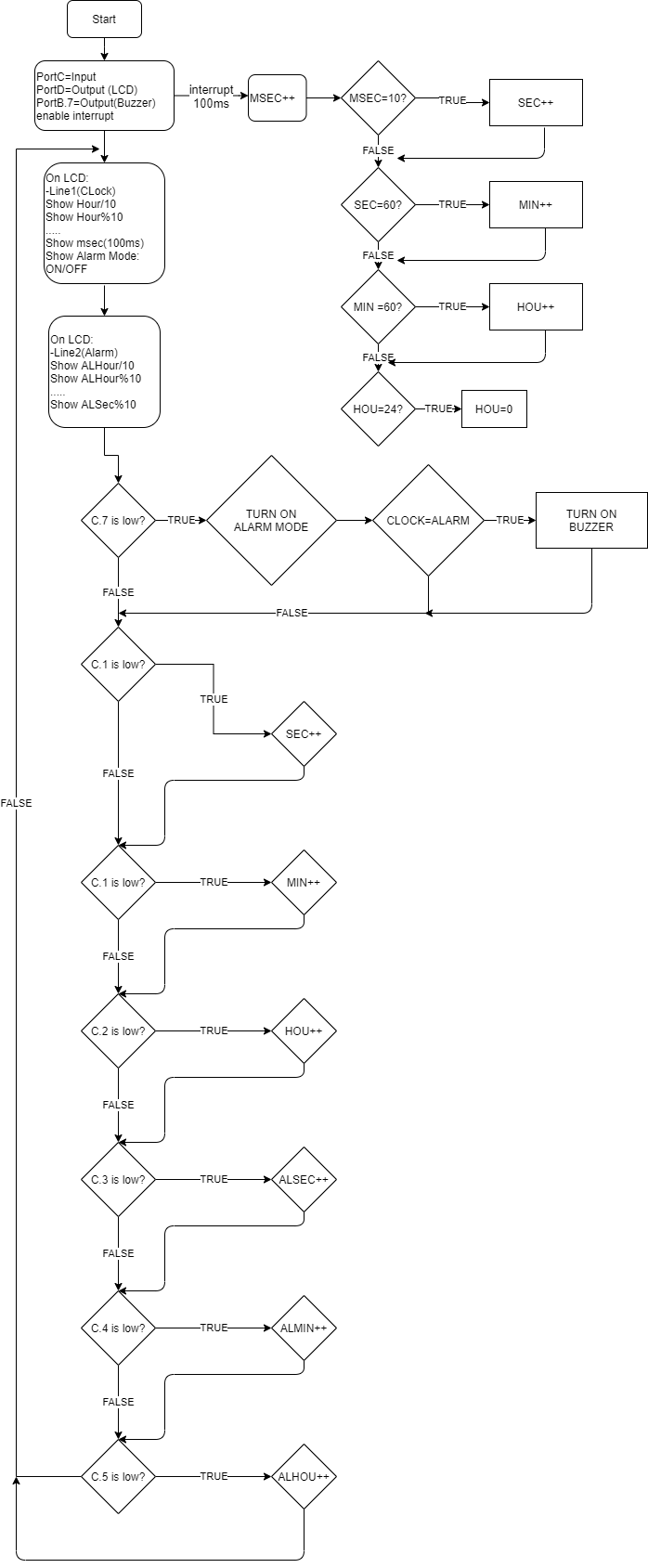
}

count=0;

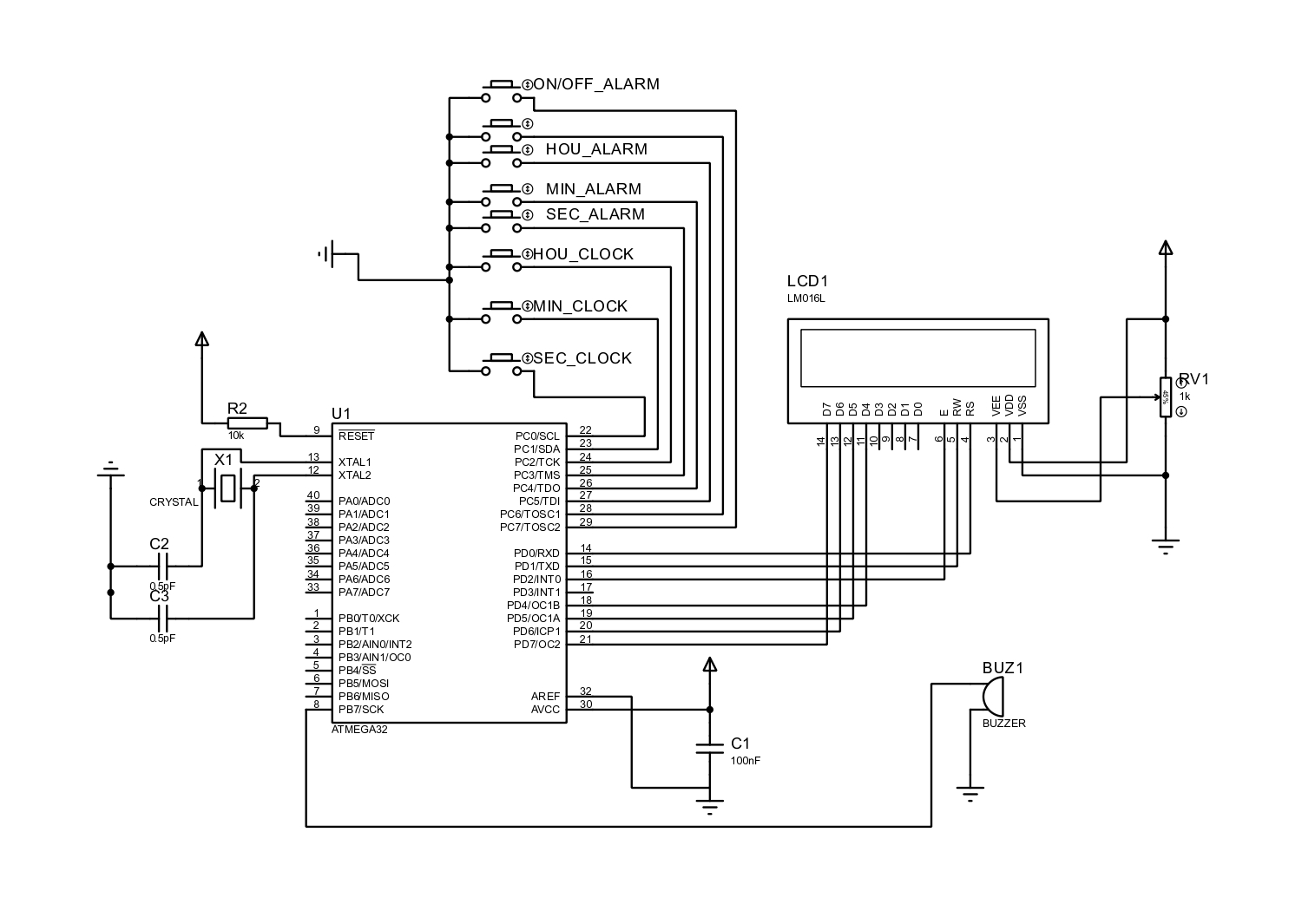
}

}

**Flowchart:**

****

**Proteus:**

****