I. Introduction:

The objective of this laboratory session is to develop a programmable timer that can set and display countdown times on the LCD screen of the EasyPIC 7 demo board. The initial setup utilizes push buttons as input devices, enabling straightforward setting and initiation of countdown times. The timer is designed to store and retrieve time settings from EEPROM, activate on user command, and display the countdown on the LCD. Upon time expiration, an audible indicator is provided by a Piezo buzzer. In subsequent stages, the push buttons used for setting will be substituted with a potentiometer, which will leverage its analog-to-digital conversion functionality to enhance the timer's operation.

II. Procedure:

- ***Part A (Programmable timer with 3 push buttons):***

  The objective of this part is to create a program for a microcontroller to develop a programmable timer; a down counter running at a rate of 1 Hz, utilizing 3 push buttons; two for the user to specify the timer duration from 0 to 255 (Up/Down) and one to initiate the timer and turn on the device. The initial time set by the up and down button is to be memorized in EEPROM, this way when the system is turned on it reads the previously set value from EEPROM variable Seconds. When the time elapses, the device is turned off, the buzzer is triggered intermittently for 3 seconds, providing an audible indication that the timer has expired.

❖ LCD16x2 interfacing with PIC18F45K22 - General

LCDs (Liquid Crystal Displays) are utilized for displaying status or parameters in embedded systems. The LCD 16x2 is a 16-pin device, comprising 8 data pins and 3 control pins (RS, RW, E), with the remaining 5 pins dedicated to power supply and backlight for the LCD.

Pin No.	Name	Type	Description
0	Ground (GND)	Power	Connects to the ground terminal of the microcontroller or power source.
1	VCC	Power	Provides voltage supply to the display.
2	V0 / VEE	Control	Regulates display contrast; connected to a variable POT (0-5V).
3	Register Select (RS)	Control	Selects between command (1) and data mode (0).
4	Read/Write (RW)	Control	Toggles between Read (1) and Write (0) operations.
5	Enable (E)	Control	Must be held high to execute Read/Write operations.
6-13	Data Pins (D0-D7)	Data	Used to send data to the display; supports 4-bit or 8-bit modes.
14	LED Anode (+)	Power	Connected to +5V to power the LCD backlight.
15	LED Cathode (-)	Power	Connected to GND for LCD backlight.

Figure 1: A table based on the datasheet

Figure 2: An LCD 2x16 Display

❖ LCD16x2 interfacing with PIC18F45K22 – Libraries

The "LCD4lib.h" header file provides a library of functions and definitions for interfacing and controlling LCD modules with a microcontroller. It includes function prototypes for:

- Initializing the LCD with the InitLCD() function, which sets up the LCD screen and control signals.
- Displaying characters stored in Read-Only Memory (ROM) using the DispRomStr() function, which takes the line, character position, and a ROM string as arguments.
- Displaying characters stored in Random Access Memory (RAM) using the DispRamStr() function, which takes the line, character position, and a character array as arguments.
- Displaying a variable string using the DispVarStr() function, which takes the number of digits, line, character position, and array size as arguments.
- Displaying blanks using the DispBlanks() function, which takes the line, character position, and number of blanks as arguments.
- Converting binary values to ASCII characters using the Bin2Asc() function, which takes a binary value and an ASCII array as arguments.

❖ Internal EEPROM Data Memory:

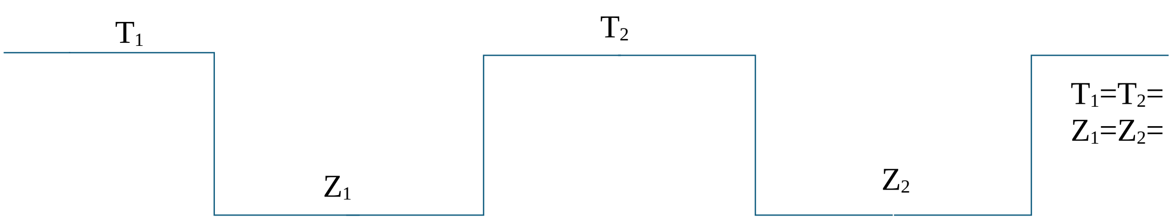
EEPROM (Electrically Erasable Programmable Read-Only Memory) is a type of non-volatile memory that offers slower response times. The "EEPROM.h" header file provides functions and macros for accessing and manipulating EEPROM data in microcontroller programming, enabling non-volatile storage of information that remains even when the microcontroller is powered off.

This header file typically includes functions such as:

- ReadEE(), which reads a byte of data from the EEPROM at a specified address.
- Wrt2EE(), which writes a byte of data from a source register to a destination register in EEPROM, allowing for data storage and manipulation. This function enables the transfer of information between registers within the EEPROM, facilitating data storage operations in microcontroller programming.

❖ Piezo Buzzer and its Duty Cycle:

A piezo buzzer is an electronic component that produces sound waves by vibrating a piezoelectric element when an electrical signal is applied. In this project, the buzzer serves as a notification to the user that the time has expired. To activate the buzzer, the duty cycle of the timer should be set to 50%, which will cause the buzzer to produce a sound, alerting the user that the time is up.



$T_1=T_2= T_3=…..=1ms$
$Z_1=Z_2= Z_3=…..=1ms$

A. MPLAB Code:

```
#include <p18cxxx.h>
#include <delays.h>
#include <LCD4lib.h>
#include <EEPROM.h>

#define eeseconds 0x00
#define DEVICE PORTCbits.RC7
#define START PORTCbits.RC0
#define UP PORTCbits.RC2
#define DOWN PORTCbits.RC1
#define BUZZER PORTEbits.RE1

char seconds;
char digits[3];
unsigned int S, j;
unsigned int k;
void setup(void);
void IncDec(void);
void TurnOn(void);
void Beep(unsigned int S);

void main(void) {
   setup();
   IncDec();
   TurnOn();
}

void setup(void) {
   InitLCD();
   ANSELC &= 0x78;
   TRISCbits.RC7 = 0;
   TRISC |= 0x07;
   TRISEbits.RE1 = 0;
   ANSELEbits.ANSE1 = 0;
   DispRomStr(Ln1Ch0, (ROM)"Set T then start");
   DispRomStr(Ln2Ch0, (ROM)"Dev.Time:     s");
   ReadEE(eeseconds, &seconds);
   Bin2Asc(seconds, digits);
   DispVarStr(digits, Ln2Ch10, 3);
   DEVICE = 0;
   BUZZER = 0;
}
```

```
void Beep(unsigned int S) {
    unsigned int count1 = 0;
    while (count1 != S) {
        BUZZER = 1;
        Delay1KTCYx(1);
        count1 += 1;
        BUZZER = 0;
        Delay1KTCYx(1);
    }
    BUZZER = 0;
}

void TurnOn(void) {
    DEVICE = 1;
    Wrt2EE(seconds, eeseconds);
    DispRomStr(Ln1Ch0, (ROM)"Left time:   s  ");

    while(1) {
        Bin2Asc(seconds, digits);
        DispVarStr(digits, Ln1Ch10, 3);

        if(seconds == 0) {
            for(k = 0; k <= 2; k++) {
                Beep(500);
                Delay10KTCYx(100);
            }
            break;
        }

        seconds--;
        Delay10KTCYx(100);
        Wrt2EE(seconds, eeseconds);
    }

    DEVICE = 0;
    ReadEE(eeseconds, &seconds);
}

void IncDec(void) {
    while(1) {
        if(UP) {
            seconds++;
            while(UP);
        }
        else if(DOWN) {
```

```
      seconds--;
      while(DOWN);
    }
    else if(START) {
      if(seconds != 0) {
        while(START);
        break;
      }
    }

    Delay1KTCYx(5);
    Bin2Asc(seconds, digits);
    DispVarStr(digits, Ln2Ch10, 3);
  }
}
```

Explanation of the Code

The code utilizes the following libraries:

- "p18cxxx.h" for compatibility
- "delays.h" for delays
- "LCD4lib.h" for controlling the LCD
- "EEPROM.h" for reading from and writing to the EEPROM

Global Variables

The code uses the following global variables:

- seconds: stores the countdown time
- digits[3]: an array to hold the converted time values for display
- S: indicates the total period of the buzzer
- k and j: loop control variables

o Functions

I. Setup Function

The setup function initializes the LCD using the InitLCD() function from the "LCD4lib.h" library. It then configures the PORTC bits RC0, RC1, RC2 as digital inputs and RC7 as an output. The function also sets bit 1 of PORT E as a digital output connected to the sounder.

The setup function displays the following on the LCD:

- A fixed string "Set T then start" on the first line
- A fixed string "Dev.Time: s" on the second line
- Reads the last value stored in EEPROM and stores it in the seconds variable
- Converts the binary value to ASCII using the Bin2Asc() function and stores it in the digits array
- Displays the value on the LCD using the DispVarStr() function

The function also initializes the device and buzzer to off (DEVICE = 0, BUZZER = 0).

II. Beep Function

The Beep() function constructs a PWM with a duty cycle of 50% and a period of S. It uses a for loop to repeat a series of actions S number of times, activating the buzzer for 1 ms and deactivating it for the same amount of time to achieve a 50% duty cycle.

III. TurnOn Function

The TurnOn() function turns on the device and writes the value of seconds to an EEPROM location. It displays the fixed countdown text "Left time: s" and converts the decremented value of seconds to ASCII characters, displaying them on the LCD. If seconds reaches zero, the buzzer beeps 3 times with a delay of 1s between each beep, and the system exits the loop. Otherwise, the function continues to decrement the seconds value and write the updated value to the EEPROM.

IV. IncDec Function

The IncDec() function alternates between the push buttons UP, DOWN, and START. If the user presses the Up button, the seconds value increases by 1, and if the user presses the Down button, the seconds value decreases by 1. If the user presses the START button, the function checks if the set time is zero and does nothing until a time is set. The function then waits for 5 ms to avoid debouncing and converts the binary values to an ASCII string, storing them in the digits array and displaying them on the LCD.

B. Proteus simulation results:

To implement the code on proteus we used the "LM06L" LCD and the "sounder" as buzzer.

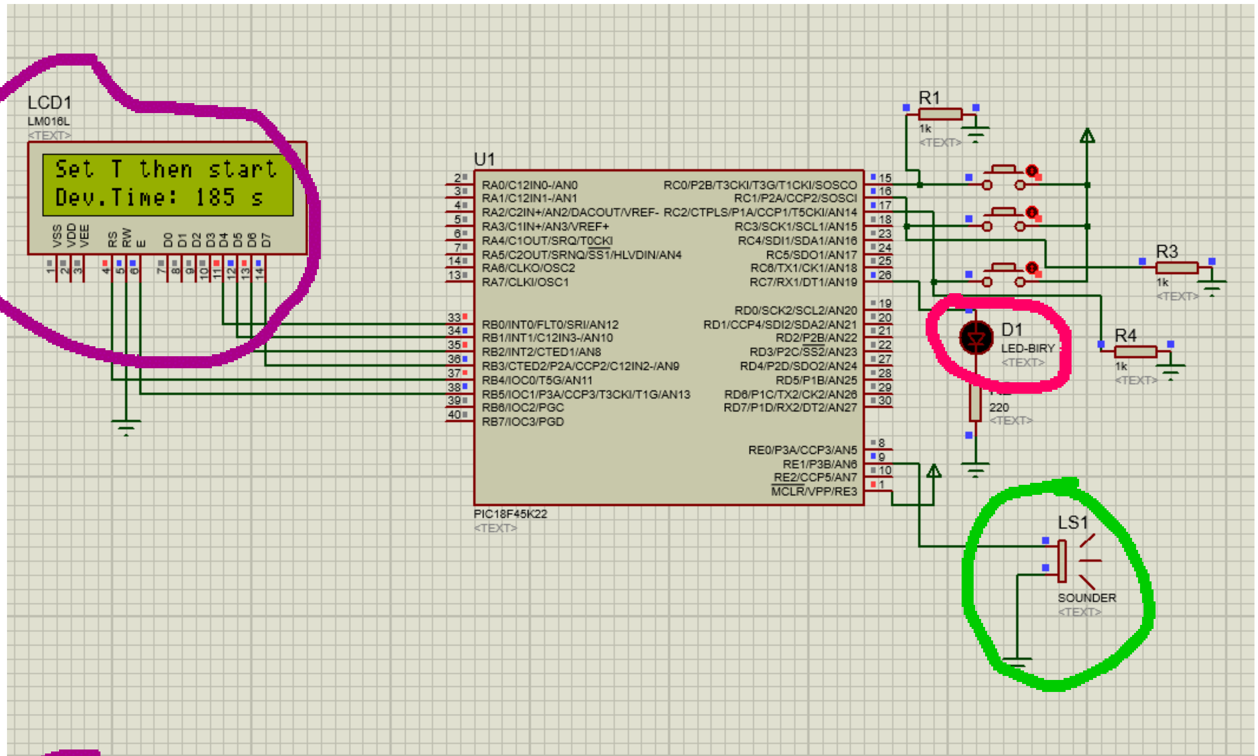


Figure 3: Proteus Connections

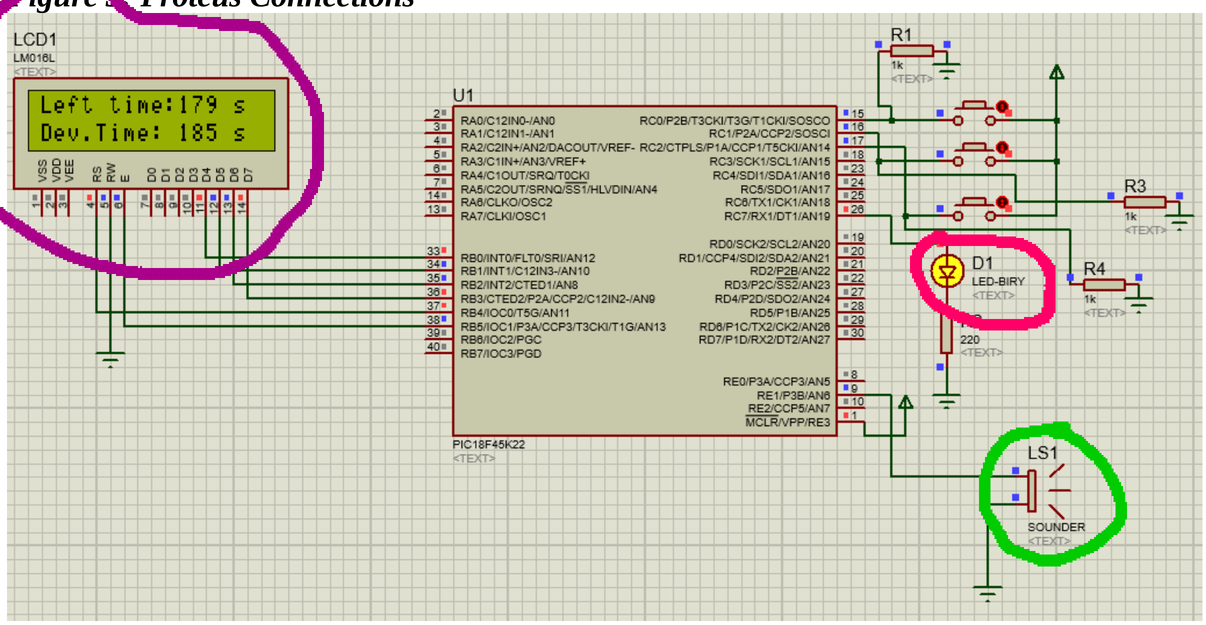


Figure 4: The countdown has started, and 6 seconds have elapsed.

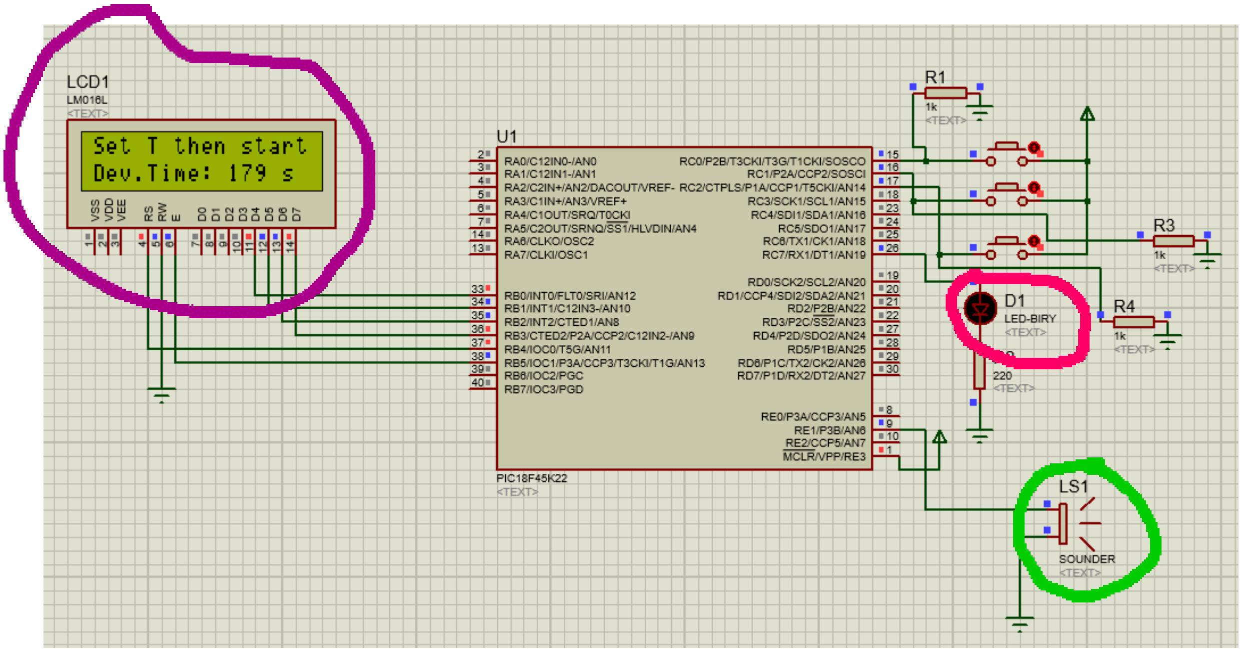


Figure 5: The remaining time has been reset to the time that was set after the device was powered off.

C. <u>Demo Board Implementation:</u>

Our project utilizes the EasyPIC v7 Demo Board, requiring the mikroProg Suite™ for PIC® to program it. In our case, we connected an LCD to the board. To turn on the LCD we should first turn on the V_{DD} (+5v) in SW4 pin 6 then we must adjust the backlight of the LCD using the potentiometer LCD contrast. As for the buzzer to be connected as mentioned in the code, we must connect the yellow jumper to RE1 for the piezo Buzzer.

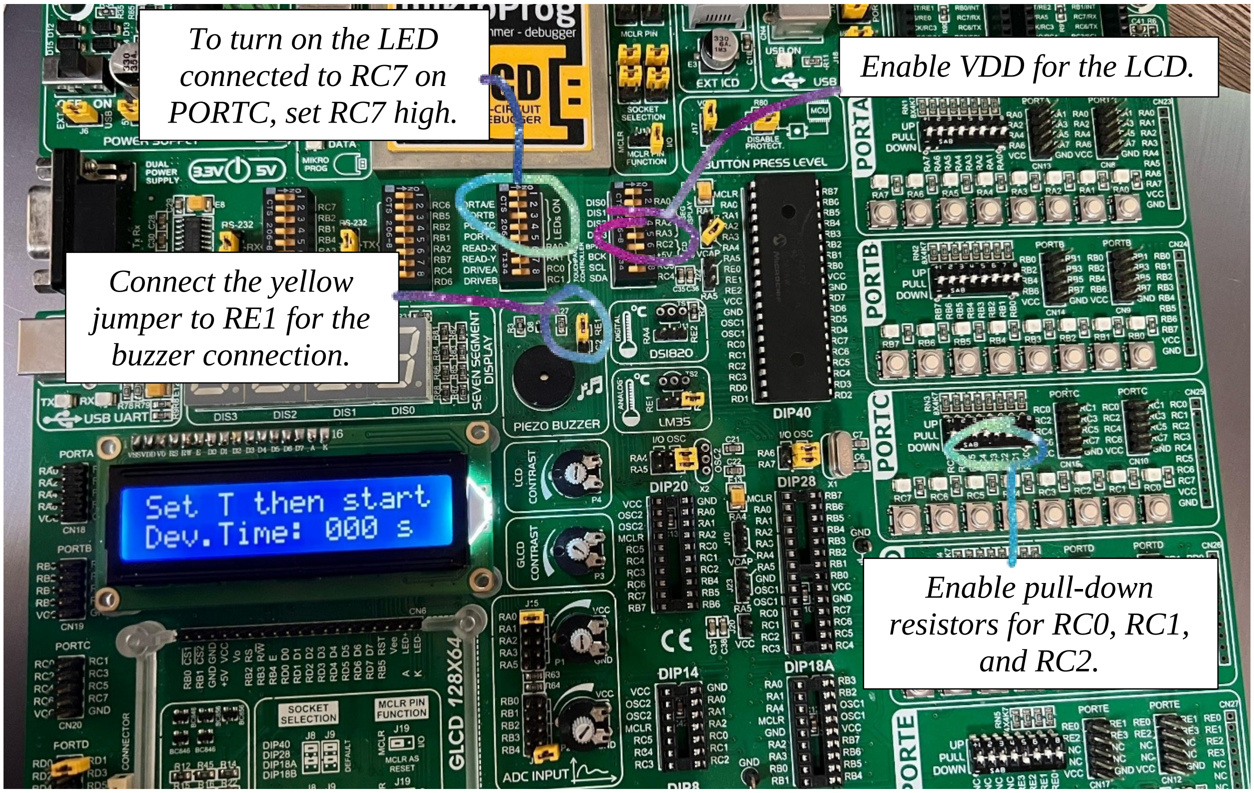


Figure 6: Setting a Start Time.

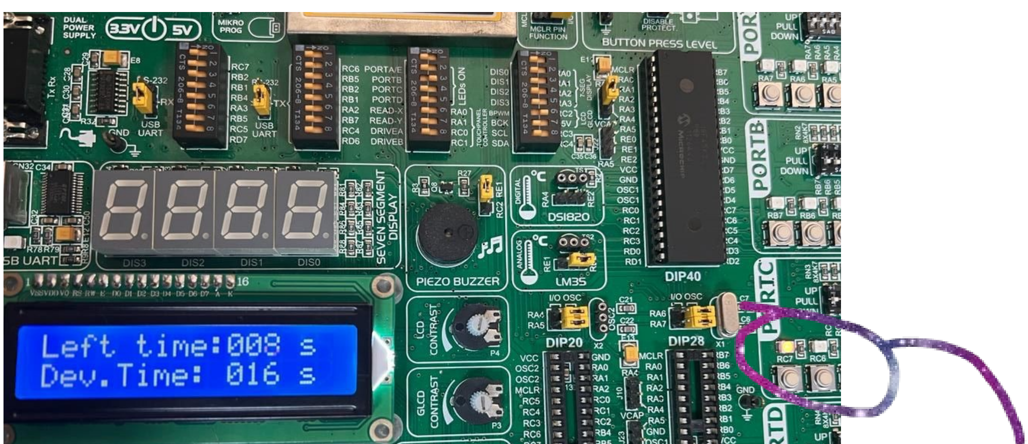


Figure 7:the device is on and the timer is counting down.

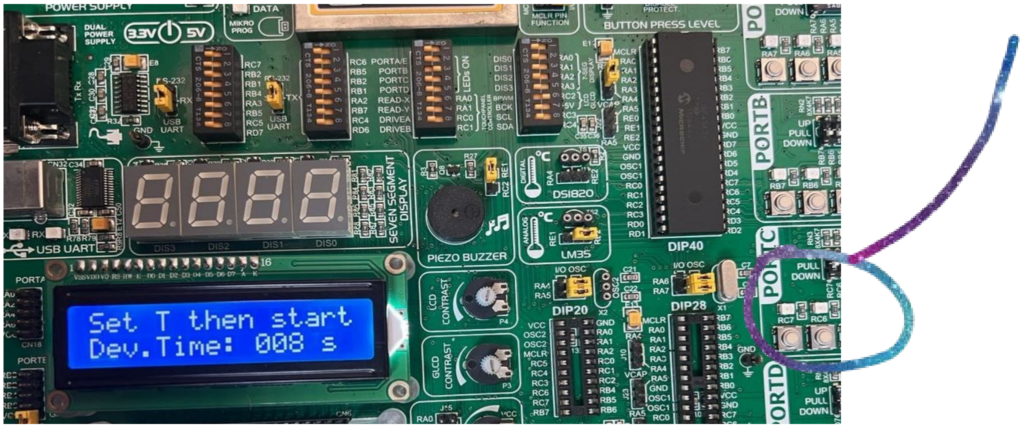


Figure 8: Set time is now the previous left time after the device was turned off.

- ***Part B (using Voltage Divider / Potentiometer):***

In this modified version, the time selection push buttons (Up/Down) are replaced by a potentiometer tied to analog channel AN0, while the Start push button is kept to turn on the device and initiate countdowns.

A. **MPLAB Code:**

```
#include <p18cxxx.h>
#include <delays.h>
#include <LCD4lib.h>
#include <EEPROM.h>

#define eeseconds 0x00
#define DEVICE PORTCbits.RC7
#define START  PORTCbits.RC0
#define BUZZER PORTEbits.RE1

char seconds;
char digits[3];
unsigned int S, j;
unsigned char k;

void setup(void);
void IncDec(void);
void TurnOn (void);
void Beep (unsigned int S);
```

```
void main(void){
   setup();
   IncDec();
   TurnOn();
}
void setup(void){
   InitLCD();

   ANSELAbits.ANSA0= 1;
   ANSELEbits.ANSE1=0;
   ANSELC &= 0x7E;

   TRISAbits.RA0=1;
   TRISCbits.RC0 = 1;
   TRISCbits.RC7 = 0;
   TRISEbits.RE1=0;

   ADCON0 = 0x01;
   ADCON1 = 0x00;
   ADCON2 = 0b00001001;

   DispRomStr(Ln1Ch0,(ROM)"Set T then start");
   DispRomStr(Ln2Ch0,(ROM)"Dev.Time:     s");

   ReadEE(eeseconds,&seconds);
   Bin2Asc(seconds,digits);
   DispVarStr(digits,Ln2Ch10,3);

   DEVICE = 0;
   BUZZER = 0;
}

void Beep (unsigned int S){
   for(j=0;j<S;j++){
   BUZZER=1;
   Delay1KTCYx(1);
   BUZZER=0;
   Delay1KTCYx(1);
   }

}

void TurnOn (void){
   DEVICE = 1;
   Wrt2EE(seconds, eeseconds);
```

```
    DispRomStr(Ln1Ch0,(ROM)"Left time:    s ");

    while(1){
        Bin2Asc(seconds,digits);
        DispVarStr(digits, Ln1Ch10, 3);
        if(seconds == 0){
        for(k=0;k<=2;k++){
        Beep(500);
        Delay10KTCYx(100);
        }
        break;
        }
        seconds--;
        Delay10KTCYx(100);
        Wrt2EE(seconds, eeseconds);
    }
    DEVICE = 0;
    ReadEE(eeseconds,&seconds);
    Reset();
}

void IncDec (void){
    while(1){
    ADCON0bits.GO_DONE = 1;

    while(ADCON0bits.NOT_DONE);
        seconds = ADRESH;
    if(START){
        if (seconds!=0){
            while (START);
            break;
        }
    }
    Delay1KTCYx(5);
    Bin2Asc(seconds,digits);
    DispVarStr(digits,Ln2Ch10,3);
    }
}
```

B. <u>The modifications:</u>

I. The Setup function:

```
1.  ANSELAbits.ANSA0= 1;
TRISAbits.RA0=1;
```

Configure the pin connected to the potentiometer AN**0** as analog input.

```
2.   TRISCbits.RC0 = 1;
 TRISCbits.RC7 = 0;
 ANSELC &= 0x7E;
```

Configure only the pins **RC0** and **RC7** as digital input and output respectively.

```
3.  ADCON0 = 0x01;
  ADCON1 = 0x00;
  ADCON2 = 0b00001001;
```

a) Configure the A/D control register 0, ADCON0, with the hex value 0x01 to select the analog channel (AN0), ensuring bits 6-2 are set to 00000. Additionally, set the A/D conversion status bit to zero, indicating the conversion is completed and not in progress, by setting bit 1 'GO/DONE' to 0. Lastly, enable the ADC by setting bit 0 'ADON' to 1, as per the configuration requirements outlined in the provided figure.

ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	CHS<4:0>					$GO/\overline{DONE}$	ADON
bit 7							**bit 0**

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented**: Read as '0'

Bit 6-2 **CHS<4:0>: Analog Channel Select bits**
00000 = AN0
00001 = AN1GO/$\overline{DONE}$
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
10101 = AN21(1)
10110 = AN22(1)
10111 = AN23(1)
11000 = AN24(1)
11001 = AN25(1)
11010 = AN26(1)
11011 = AN27(1)
11100 = Reserved
11101 = CTMU
11110 = DAC
11111 = FVR BUF2 (1.024V/2.048V/2.096V Volt Fixed Voltage Reference)(2)

bit 1 **GO/$\overline{\textbf{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed / not in progress)

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

Note 1: Available on PIC18(L)F4XK22 devices only.
Note 2: Allow greater than 15 μs acquisition time when measuring the Fixed Voltage Reference.

Figure 8: ADCON0

b) Configure the A/D control register 1, ADCON1, with the hex value 0x00 to connect the A/D VREF+/- to internal signals, as required for the desired configuration.

ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
TRIGSEL	—	—	—	PVCFG<1:0>		NVCFG<1:0>	
bit 7							**bit 0**

R = Readable bit W = Writable bit U = Unimplemented bit, read as ‘0’
-n = Value at POR ‘1’ = Bit is set ‘0’ = Bit is cleared x = Bit is unknown

bit 7 **TRIGSEL**: Special Trigger Select bit
1 = Selects the special trigger from CTMU
0 = Selects the special trigger from CCP5

bit 6-4 **Unimplemented:** Read as ‘0’

bit 3-2 **PVCFG<1:0>:** Positive Voltage Reference Configuration bits
00 = A/D VREF+ connected to internal signal, AVDD
01 = A/D VREF+ connected to external pin, VREF+
10 = A/D VREF+ connected to internal signal, FVR BUF2
11 = Reserved (by default, A/D VREF+ connected to internal signal, AVDD)

bit 1-0 **NVCFG<1:0>:** Negative Voltage Reference Configuration bits
00 = A/D VREF- connected to internal signal, AVSS
01 = A/D VREF- connected to external pin, VREF-
10 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)
11 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)

Figure 9: ADCON1

c) Configure the A/D control register 2, ADCON2, with the binary value 0b00001001 to set the A/D result format to left justified (with 8 bits being sufficient, as the timer values range from 0 to 255), which requires bit 7 'ADFM' to be 0. Additionally, select an acquisition time of 2 TAD by setting bits 5-3 'ACQT <2:0>' to 001, and choose a conversion clock of FOSC/8 by setting bits 2-0 'ADCS <2:0>' to 001.

ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	–	ACQT<2:0>			ADCS<2:0>		
bit 7							bit 0

bit 7 **ADFM**: A/D Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 Unimplemented: Read as '0'

bit 5-3 **ACQT<2:0>**: A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/$\overline{\text{DONE}}$ bit is set until conversions begins.
000 = 0[(1)]
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS<2:0>**: A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC /8
010 = FOSC /32
011 = FRC[(1)] (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC /4
101 = FOSC /16
110 = FOSC /64
111 = FRC[(1)] (clock derived from a dedicated internal oscillator = 600 kHz nominal)

Note 1: When the A/D clock source is selected as FRC then the start of conversion is delayed by one instruction cycle after the GO/$\overline{\text{DONE}}$ bit is set to allow the SLEEP instruction to be executed.

Figure 10: ADCON2

II. The IncDec Function:

```
void IncDec (void){
  while(1){
  ADCON0bits.GO_DONE = 1;

  while(ADCON0bits.NOT_DONE);
    seconds = ADRESH;
  if(START){
    if (seconds!=0){
      while (START);
      break;
    }
  }
  Delay1KTCYx(5);
  Bin2Asc(seconds,digits);
  DispVarStr(digits,Ln2Ch10,3);
  }
}
```

To initiate and complete the ADC conversion:

- Initiate the ADC conversion by setting the GO_DONE bit to start the conversion process.
- Wait for the conversion to complete by entering a loop that checks the NOT_DONE bit of the ADCON0 register. The loop will continue to run until the conversion is finished, indicated by the NOT_DONE bit being cleared.
- Once the conversion is complete, read the result from the ADRESH register, which will contain a value between 0 and 255, representing the desired time in seconds.

Note: No action will be taken until the user sets a valid time.

C. **Proteus simulation results:**

In this part we used the potentiometer **POT-GH** connected to V_{DD} and ground and connected it to the appropriate pin RA0.

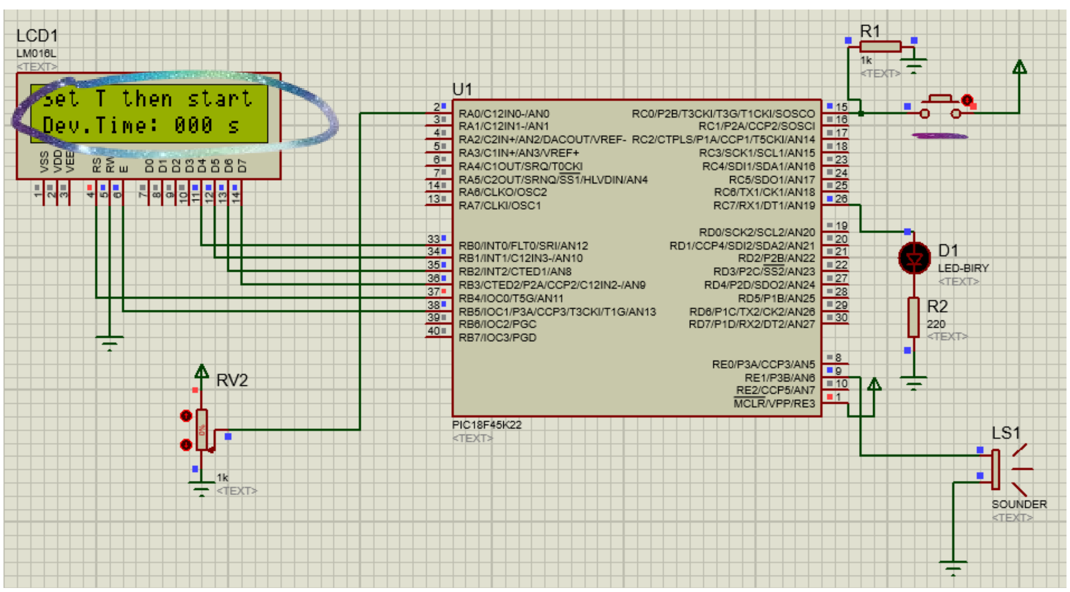


Figure 9: Time t = 0s, START is ON, nothing is activated.

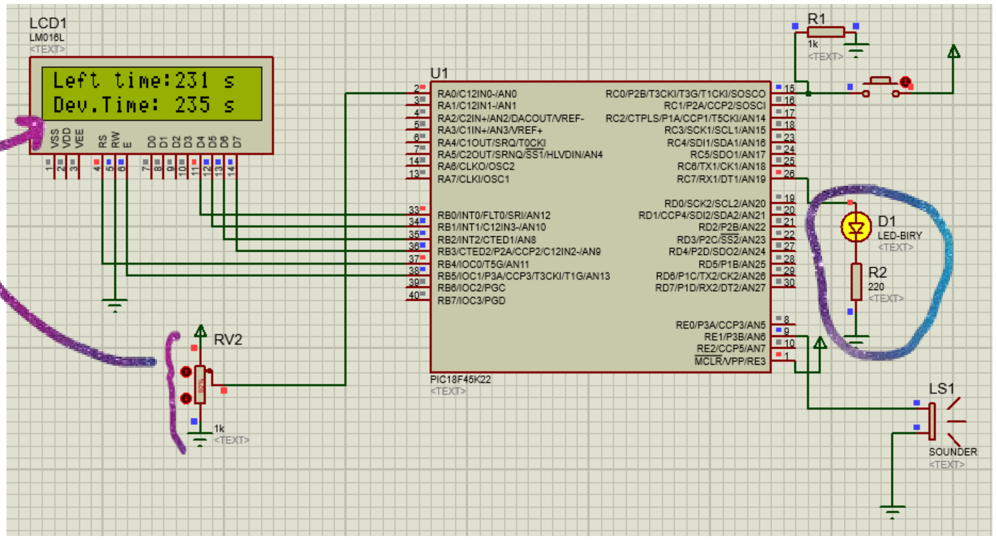


Figure 10: Time selection where 4 seconds elapsed.

D. <u>**Demo Board Implementation:**</u>

In this part we must connect the potentiometer to the board, so we must put the yellow jumper on the **RA0** of the ADC input of the demo board.

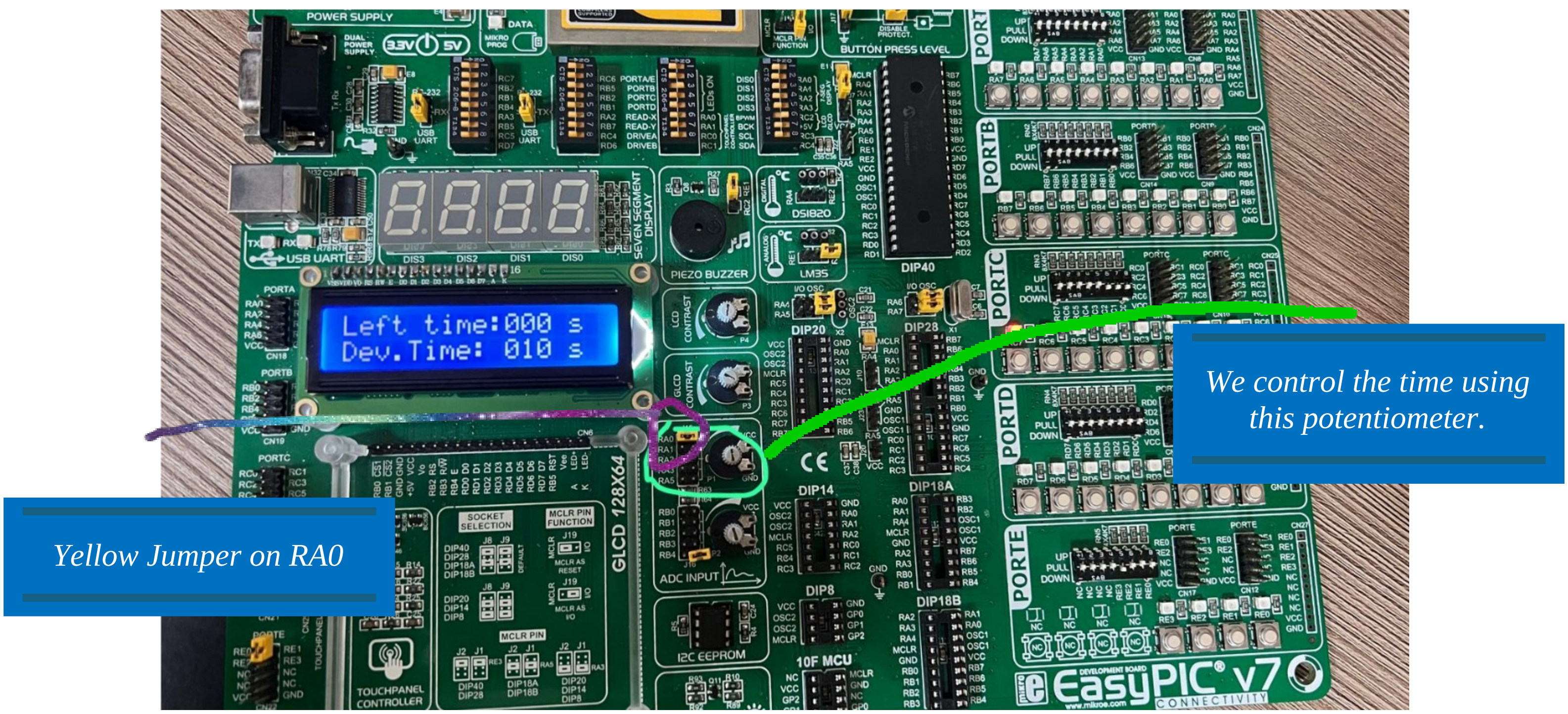


Figure 11: Display Time Selection

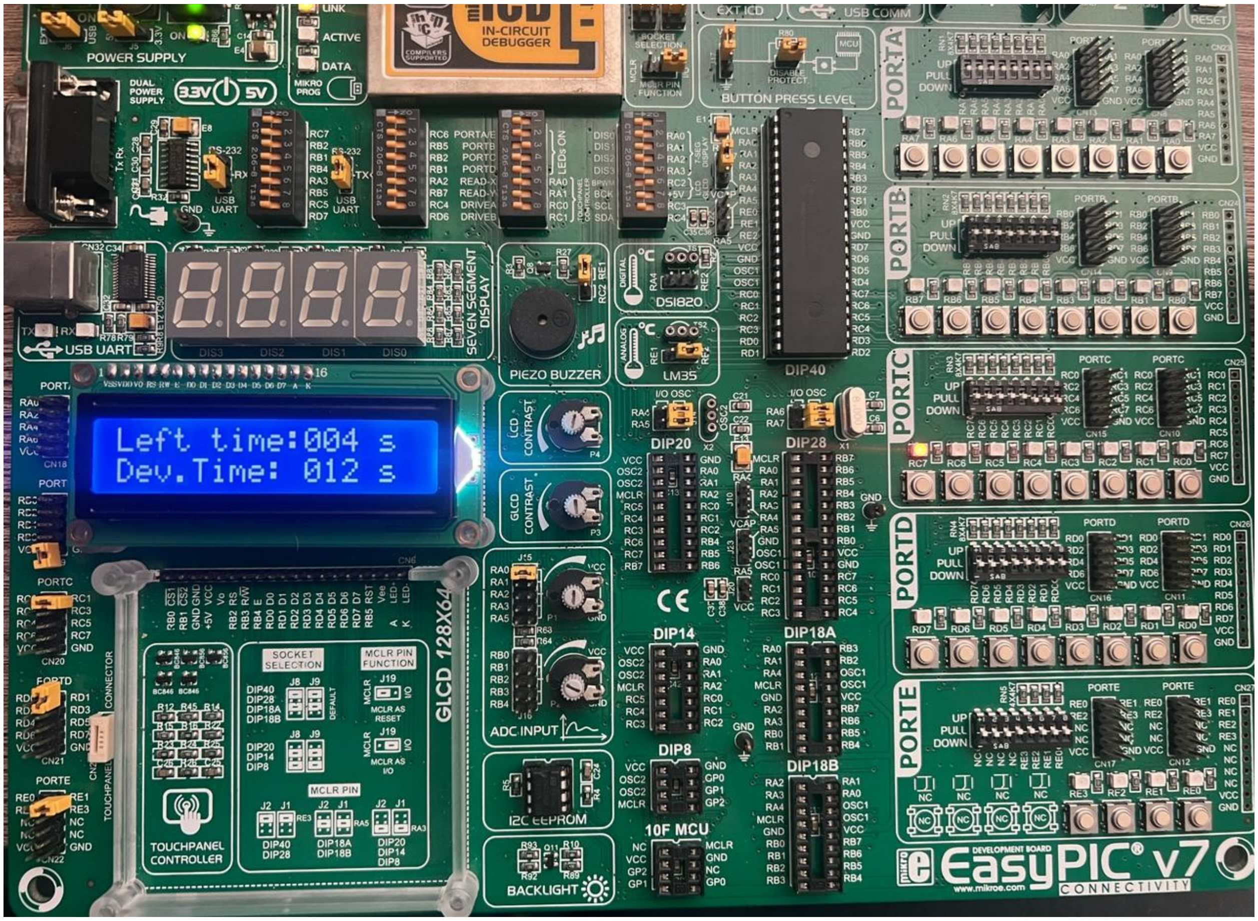


Figure 12: --Continuation

III. Conclusion:

The experiment successfully implemented a programmable timer, utilizing push buttons, EEPROM, LCD, and a Piezo buzzer to display and alert the user. The setup was later enhanced by replacing push buttons with a potentiometer, making time setting more intuitive and user-friendly, and ultimately achieving its goals while providing valuable practical experience.