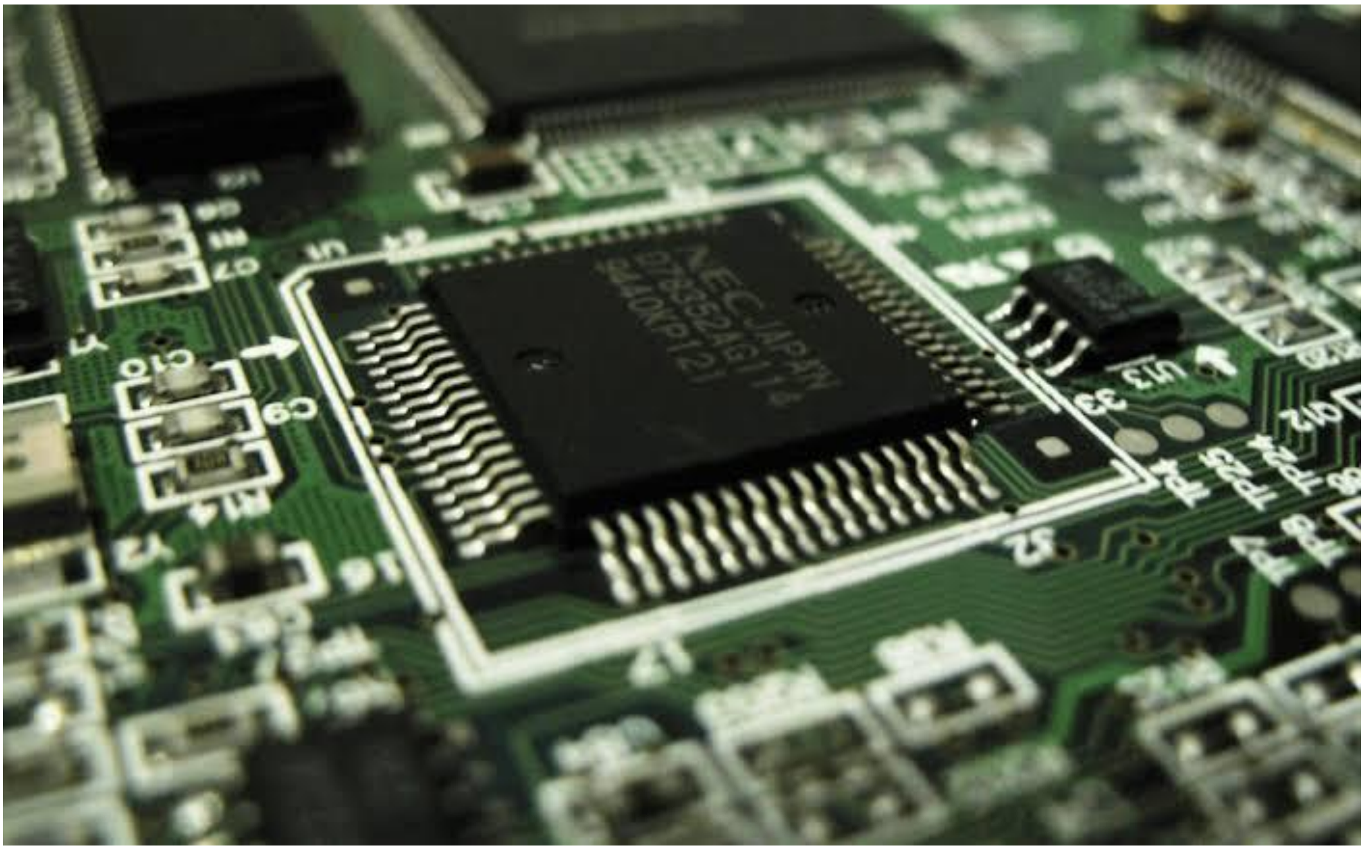
Microprocessors
Final Project



Slide Show Using GLCD

Seyit Kaan Karaalioglu
140223014

INTRODUCTION

In this project, we are asked to build a project which shows some logos on the screen of a Graphic LCD using external EEPROM with some prerequisites. The project is programmed and simulated with the help of CCS compiler and Proteus electronic design software, respectively. The requirements needed for the project can be simplified as below:

- In Project, a slide show system will be implemented using a microcontroller and a graphic LCD.
- After selecting the images and the time interval for every image using a potentiometer, and showing the duration in seven segment display, an apply button will be pressed. After selecting the images and the time for every image, we will press the edit button again and leave the settings.
- For every input, Edit button will be pressed and until the nth input, the order and the time input of the chosen images will appear in a two digit seven segment display. After selecting the properties for a given image apply button will be pressed and the timing settings for the next image will begin.
- After selecting the last image, Apply button will be used to start the project.
- When the slide show started, the order of the images will be shown in the display and the duration will decrease in the display.
- During the sliding, forward, backwards, and random modes will be implemented.
- Timer interrupt will be used for display.
- Timing calculation will be made with timer interrupt.
- The last option will be saved in EEPROM.
- The system will start working at the same place in case of forced shutdown or shutdown.
- Watchdog timer and sleep mode will be added.
- The project that satisfies the all conditions will be developed.
- The project can be further developed.
- The project will be designed with the consideration of industrial and real methods.

Components used in the project however, can be listed as below:

- PIC16F877 MCU
- AT24C512 External EEPROM
- AMPIRE 128x64 Graphic LCD Display
- 2 Digit seven segment display
- Potentiometer to choose the time interval for the successive images

SOLUTIONS AND METHODS

Some implementation problems regarding to project needed some specific solutions related to it. First drawback of my project was about the probability of images appearing on the screen in a slow fashion due to the PIC16F877 microprocessor. Another problem was the problems related to the libraries of GLCD and EEPROM, respectively. Also, I had to revise on my past weaknesses when it comes to designing a working project from scratch. I read some books, websites and forum suggestions regarding the topic as mentioned in the References. Also, during the programming phase, I had to deal with some problems about the #use fast_io directives. When I used some of them in my source-code I encountered some problems simulating GLCD particularly.

After fast_io directives, the pins and their corresponding pins mentioned with the #define directives. The libraries added with #include directives as in CCS manual. Instead of simulating the time in two seven segments, I used A, B, C, D, E and F as seconds in their corresponding values.

Then, the global variables defined as words with global letters namely INTS_PER_SECOND(interrupts per second) and PICTURE_COUNT to show up the images in order.

Then we define the variables resim, blinkcounter as integers to define the images and to count the blinks inside the program. To manage the flow of the system we define three variables namely mode, cmode and setm, and then we define the variable sayi and digits of seven segment as 2 element array called dig[2].(one showing the order of images, the other for ticking the clock backwards.

Text variable to display the welcoming message before the Onay button is pressed is defined as text1[22]. Blink and direct variables are defined using int1 data type simply as 1s and 0s. For simulating on display, I defined segment variable.

I defined timeout and time[PICTURE_COUNT], int_count, on_down (for ayar button), on_down_onay (for onay_button) variables, respectively.

COMPONENTS USED IN THE PROJECT

- PIC16F877

PIC microcontroller is one of the most known microcontrollers in the industry. It is very easy to use, code and simulate. The main advantage of PIC MCUs are Flash memory technology which enables the programmer to write and erase as many times as possible.

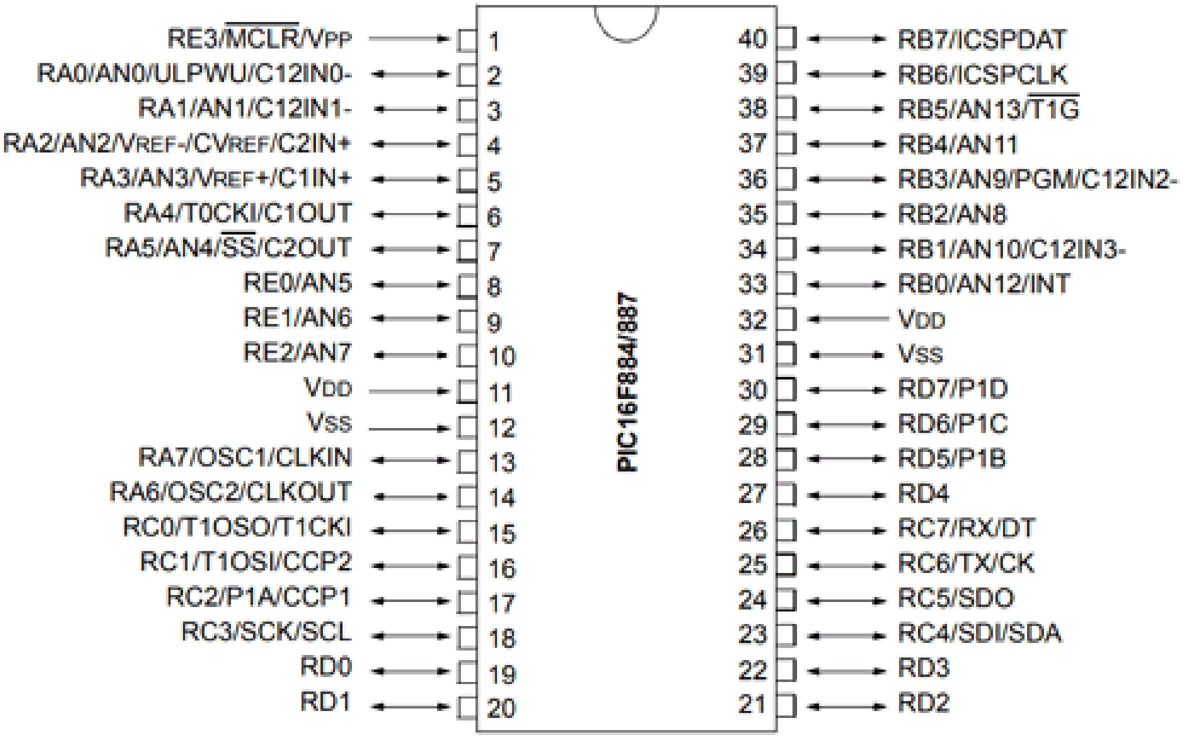


FIG.1

- AT24C512

The AT24C512 provides 524,288 bits of serial electrically erasable and programmable read only memory(**EEPROM**) organized as 65,536 words of 8 bits each.

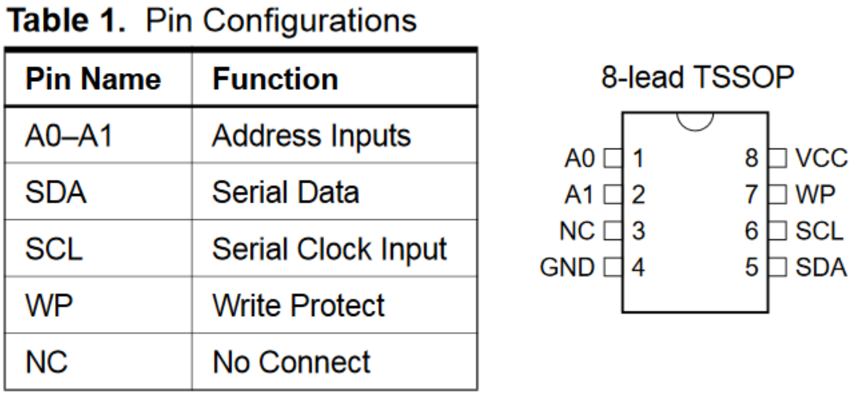
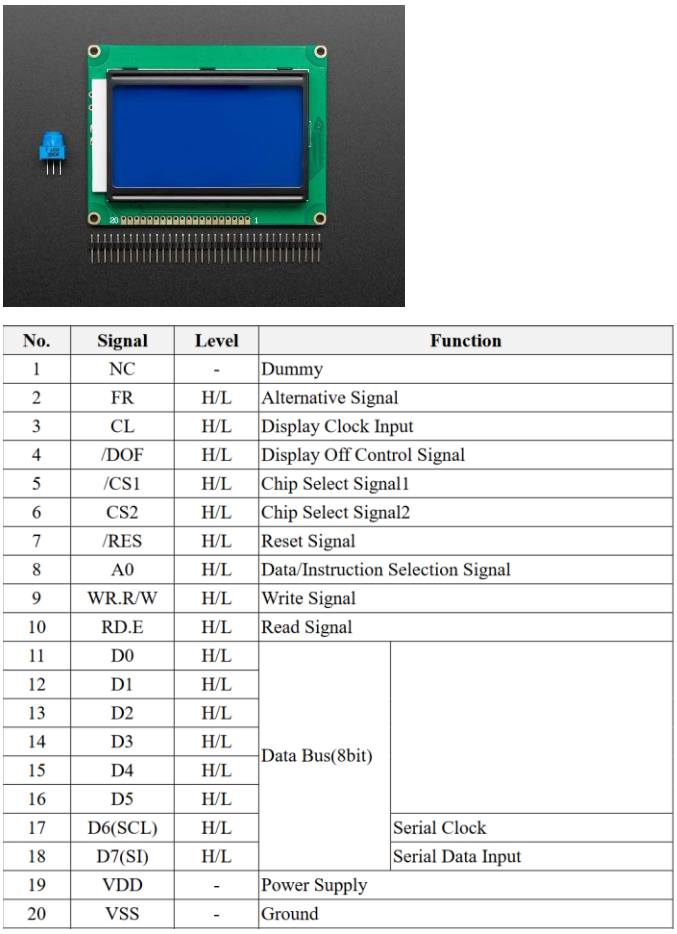


Table 1. Pin Configurations

Pin Name	Function
A0–A1	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect
NC	No Connect

FIG.2

- AMPIRE 128x64 GLCD



No.	Signal	Level	Function	
1	NC	-	Dummy	
2	FR	H/L	Alternative Signal	
3	CL	H/L	Display Clock Input	
4	/DOF	H/L	Display Off Control Signal	
5	/CS1	H/L	Chip Select Signal1	
6	CS2	H/L	Chip Select Signal2	
7	/RES	H/L	Reset Signal	
8	A0	H/L	Data/Instruction Selection Signal	
9	WR.R/W	H/L	Write Signal	
10	RD.E	H/L	Read Signal	
11	D0	H/L	Data Bus(8bit)	
12	D1	H/L		
13	D2	H/L		
14	D3	H/L		
15	D4	H/L		
16	D5	H/L		
17	D6(SCL)	H/L		Serial Clock
18	D7(SI)	H/L		Serial Data Input
19	VDD	-	Power Supply	
20	VSS	-	Ground	

FIG.3

Other components used in the project are 2 digit seven segment display, a potentiometer and two buttons for the edit and execute buttons.

GENERAL OVERVIEW OF THE PROJECT

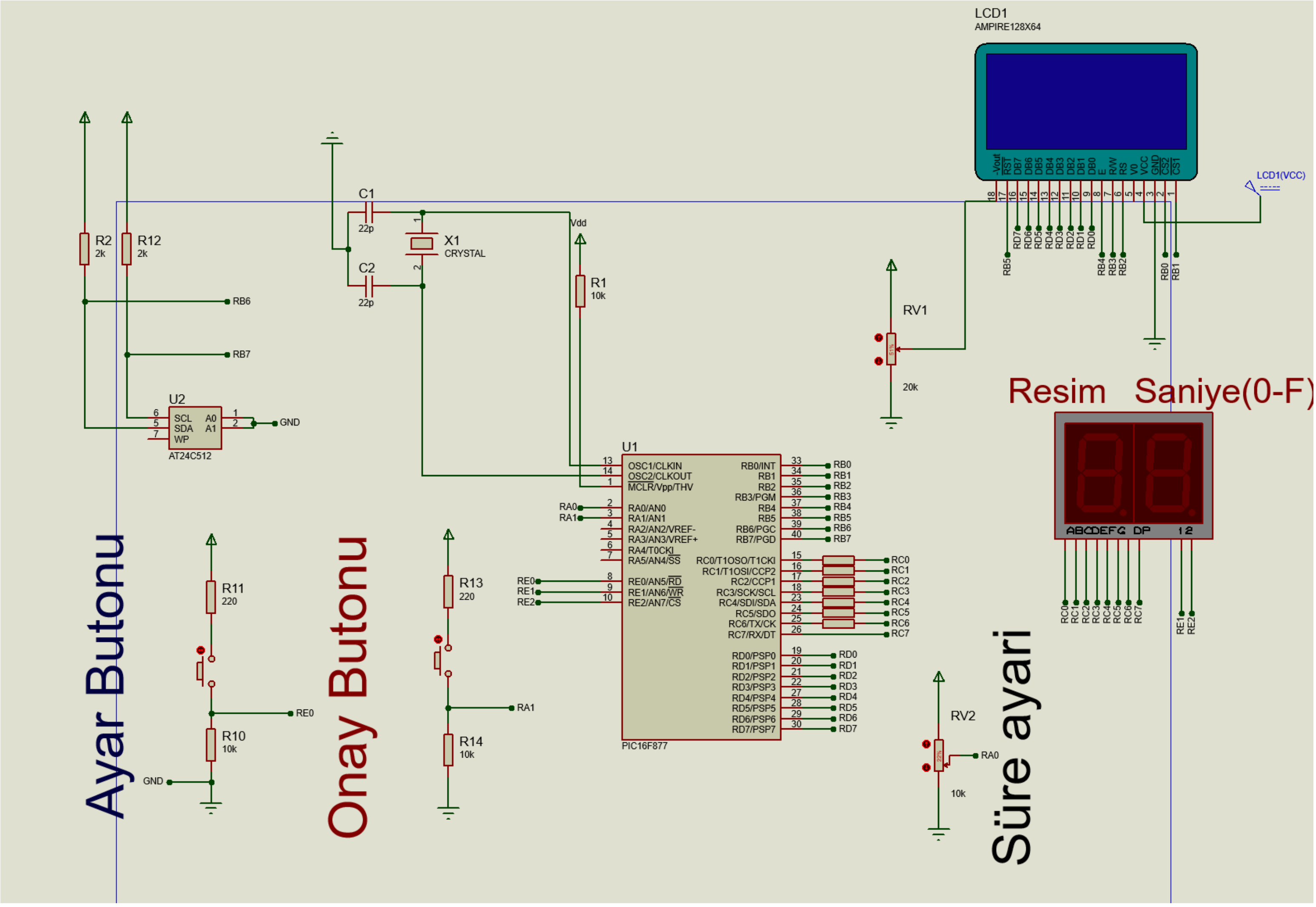
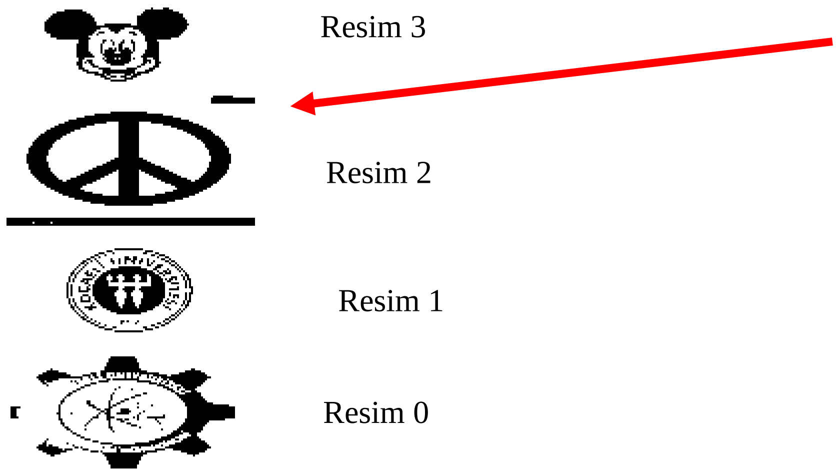


FIG.4

As you can see from above picture, the project, as expected, works with two buttons and a potentiometer to set the times for each image item. Connections between the pins in ports are made with pin labels to prevent the misunderstandings in the schematic. Pins A0 and A1 are connected to potentiometer and apply button respectively. Port B pins starting from B0 to B5 are connected to GLCD while pins B6 and B7 are connected to EEPROM SCL and SDA. Port C is used to drive the 2 digit seven segment. Port D is connected to GLCD pins B0 to B7. Pins E1 and E2 are connected to the voltage source of seven segment display. Due to the lack of clock source in PIC16F877, we provide it with crystal oscillator. The project starts with displaying a welcome message and expecting the user to press Edit(Ayar) button.

The images I selected are basically Monochrome logos with bitmap extensions. The image file I loaded to the EEPROM can be seen below. The sonuc.bmp file is loaded into EEPROM as initial contents of memory.



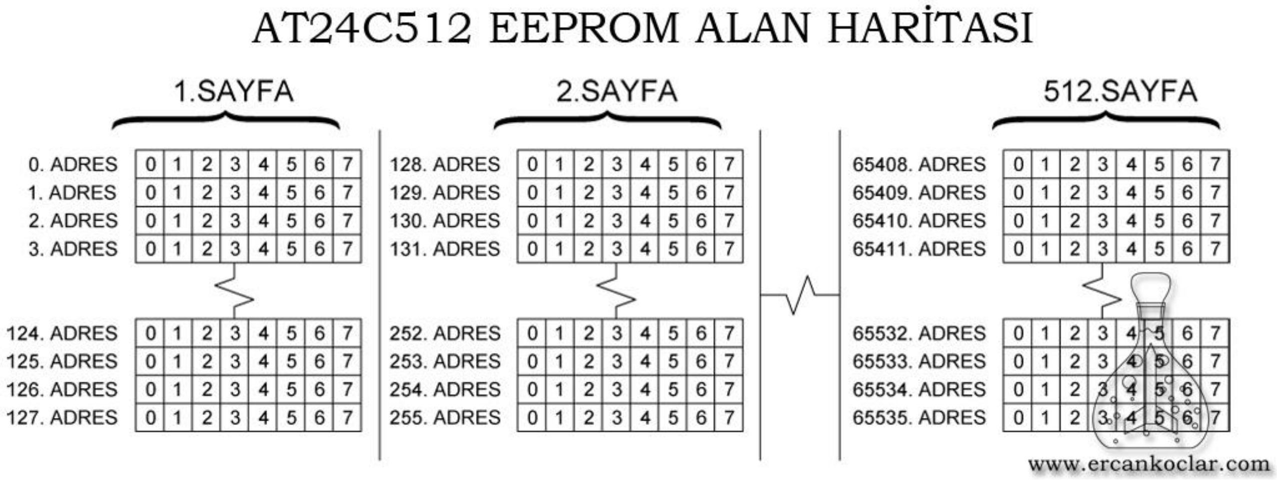
As you can see from the file, the images I found are basically bitmap logos, and the line under the Mickey mouse is equal to 64 in y coordinates in GLCD. Therefore in sonuc.bmp file we have 128x64x4 = 128x256pixels cascaded one after the other.

One of the logos lost some of its data while converting to bmp file like the Mechatronics Department logo.

FIG.5

One of the cons I encountered while programming was that not being able to see the initial contents of memory in EEPROM properties while simulating in PROTEUS. So I had to check on EEPROM memory locations schematic to make up my mind.

EEPROM memory map is shown below;



AT24C512 EEPROM ALAN HARİTASI

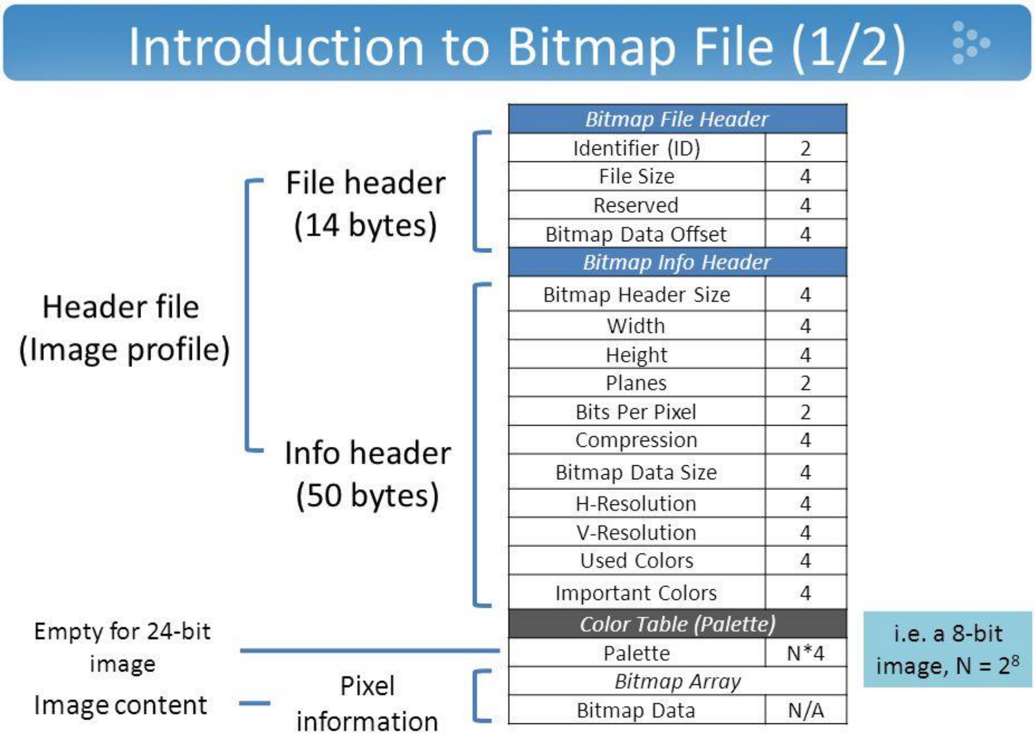
1.SAYFA		2.SAYFA		512.SAYFA	
0. ADRES	0 1 2 3 4 5 6 7	128. ADRES	0 1 2 3 4 5 6 7	65408. ADRES	0 1 2 3 4 5 6 7
1. ADRES	0 1 2 3 4 5 6 7	129. ADRES	0 1 2 3 4 5 6 7	65409. ADRES	0 1 2 3 4 5 6 7
2. ADRES	0 1 2 3 4 5 6 7	130. ADRES	0 1 2 3 4 5 6 7	65410. ADRES	0 1 2 3 4 5 6 7
3. ADRES	0 1 2 3 4 5 6 7	131. ADRES	0 1 2 3 4 5 6 7	65411. ADRES	0 1 2 3 4 5 6 7
124. ADRES	0 1 2 3 4 5 6 7	252. ADRES	0 1 2 3 4 5 6 7	65532. ADRES	0 1 2 3 4 5 6 7
125. ADRES	0 1 2 3 4 5 6 7	253. ADRES	0 1 2 3 4 5 6 7	65533. ADRES	0 1 2 3 4 5 6 7
126. ADRES	0 1 2 3 4 5 6 7	254. ADRES	0 1 2 3 4 5 6 7	65534. ADRES	0 1 2 3 4 5 6 7
127. ADRES	0 1 2 3 4 5 6 7	255. ADRES	0 1 2 3 4 5 6 7	65535. ADRES	0 1 2 3 4 5 6 7

www.ercankoclar.com

FIG.6

As you can see AT24C512 EEPROM has 512 pages and each page has 128 addresses. That makes the total number of bytes as 65536(524, 288 bits).

Bitmap file schematic however is shown below;



Introduction to Bitmap File (1/2)

Header file (Image profile)

File header (14 bytes)

Info header (50 bytes)

Empty for 24-bit image

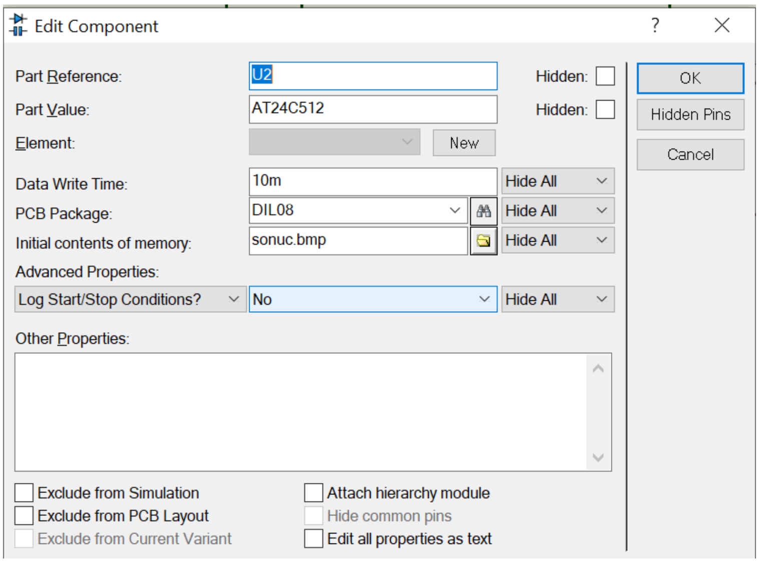
Image content

Pixel information

Bitmap File Header	
Identifier (ID)	2
File Size	4
Reserved	4
Bitmap Data Offset	4
Bitmap Info Header	
Bitmap Header Size	4
Width	4
Height	4
Planes	2
Bits Per Pixel	2
Compression	4
Bitmap Data Size	4
H-Resolution	4
V-Resolution	4
Used Colors	4
Important Colors	4
Color Table (Palette)	
Palette	N*4
Bitmap Array	
Bitmap Data	N/A

i.e. a 8-bit image, N = 2^8

FIG.7



Edit Component

Part Reference:	U2	Hidden: □	
Part Value:	AT24C512	Hidden: □	
Element:		New	
Data Write Time:	10m	Hide All	
PCB Package:	DIL08	Hide All	
Initial contents of memory:	sonuc.bmp	Hide All	
Advanced Properties:			
Log Start/Stop Conditions?	No	Hide All	

Other Properties:

□ Exclude from Simulation
□ Exclude from PCB Layout
□ Exclude from Current Variant
□ Attach hierarchy module
□ Hide common pins
□ Edit all properties as text

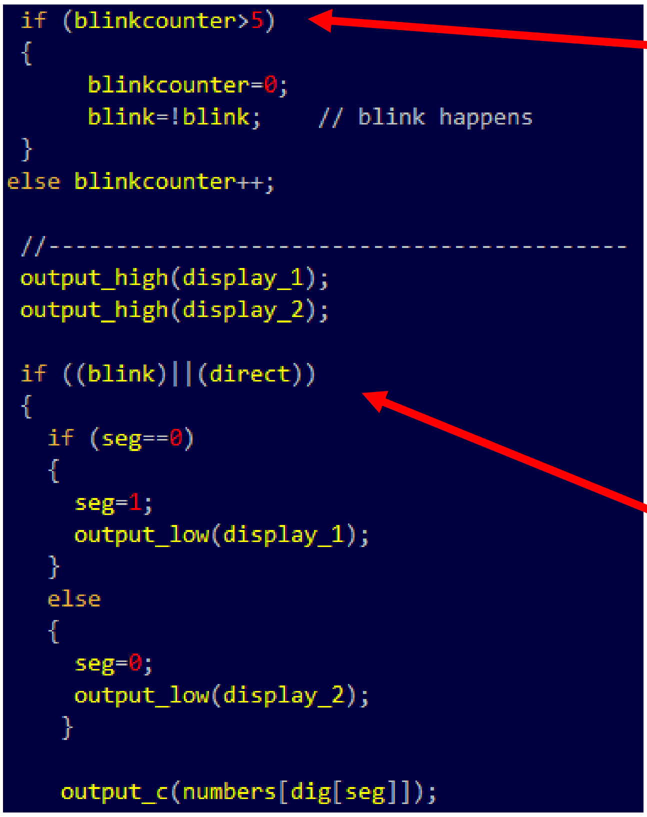
FIG.8

The image is loaded inside the EEPROM using initial contents of memory section in Proteus.

HOW THE CODE WORKS

CLOCK_ISR FUNCTION

After describing the variables to an extent, let us look at the functions used in the project. Project starts with interrupt service routine function basically to determine the frequency of the interrupts. After initializing interrupt procedure with #int_rtcc, function controls blinkcounter variable to increase or decrease the frequency in the seven segment display.



```
 if (blinkcounter>5)
 {
      blinkcounter=0;
      blink=!blink;    // blink happens
 }
else blinkcounter++;

 //-------------------------------------------
 output_high(display_1);
 output_high(display_2);

 if ((blink)||(direct))
 {
   if (seg==0)
   {
     seg=1;
     output_low(display_1);
   }
   else
   {
     seg=0;
     output_low(display_2);
    }

    output_c(numbers[dig[seg]]);
```

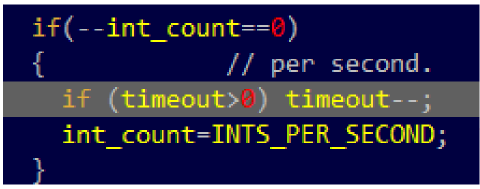
FIG.9

blinkcounter variable starts with 0, and it is incremented every time the else block is executed, when it reaches to 6, blinkcounter resets to 0 and with the statement blink=!blink, blink becomes 1, blink happens. Let me remind you, blink starts with the value of 0 unless it is specified otherwise. Then, I simply gave display_1 and display_2 output_high to open the seven segment displays.

The if block actually reverses the signal in seven segments, when blink happens and or block is executed, if we are in the 0 segment, segment is changed to segment 1 and display_1 is lowered, else, that is , if we are in segment 1, segment becomes 0 and the display_2 is lowered. After the if-else blockchain we show the numbers in port C.

This simply works in the following order, we define a sayi variable and from the ADC, we read this value, then the sayi is equated to dig[seg] in the case function, then this value is matched with the corresponding element in the numbers array.

Now , let us have a look at the other if statement following this,



```
 if(--int_count==0)
 {             // per second.
   if (timeout>0) timeout--;
   int_count=INTS_PER_SECOND;
 }
```

FIG.10

I defined INTS_PER_SECOND as a global variable, and calculated it according to my my project. So let's have a look how I calculated it.

$$fout = \frac{fclk}{\text{Prescaler} * (256 - \text{TMR0}) * \text{Count}}$$

FIG.11

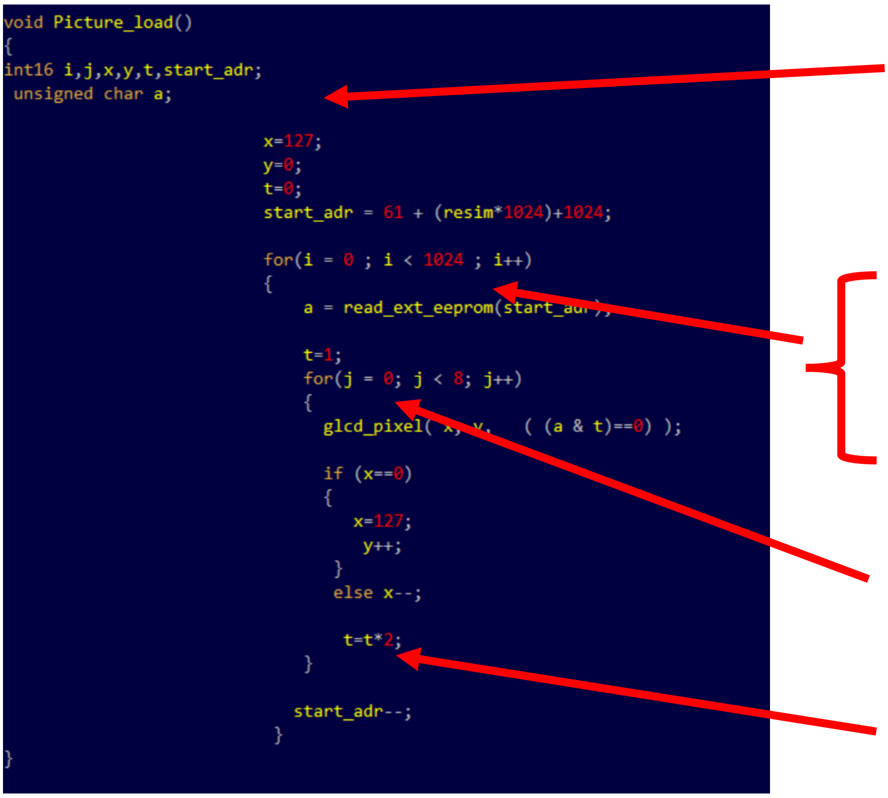
I used 20Mhz clock frequency, started tmr0 from 0 and equated count to obtain;

$$\frac{20mHZ}{1 * 4 * \ 256 * 128} = 152{,}58$$

And I defined INTS_PER_SECOND as this value, to obtain more or less, a second.

Now the code picture in Figure 10 shows the second if statement in clock_isr() function. Int_count function in main is equated to INTS_PER_SECOND. So every time this if block is executed, int_count will be decremented and when int_count equals to zero, if the timeout is greater than 0, timeout variable will be decremented. And then int_count will made equal to INTS_PER_SECOND again. After describing this function, we will look at how the logos are shown on the screen with the help of another function, namely Picture_Load().

PICTURE_LOAD FUNCTION



```
void Picture_load()
{
int16 i,j,x,y,t,start_adr;
 unsigned char a;

                          x=127;
                          y=0;
                          t=0;
                          start_adr = 61 + (resim*1024)+1024;

                          for(i = 0 ; i < 1024 ; i++)
                          {
                              a = read_ext_eeprom(start_adr);

                              t=1;
                              for(j = 0; j < 8; j++)
                              {
                                glcd_pixel( x, y,   ( (a & t)==0) );

                                if (x==0)
                                {
                                   x=127;
                                    y++;
                                 }
                                 else x--;

                                  t=t*2;
                              }

                             start_adr--;
                           }
}
```

Picture load function shows icons using a for loop and reads from the start address of EEPROM with read_ext_eeprom(start_adr) statement. For the first picture for example, resim = 0 and the start_adr is at the end of the EEPROM flash memory.

In 128x64 GLCD, we have 128 over 8 bytes, which is 16, and 64 lines,

16 * 64 = 1024 bytes, that is , this is gonna be all the points in the screen as bytes.

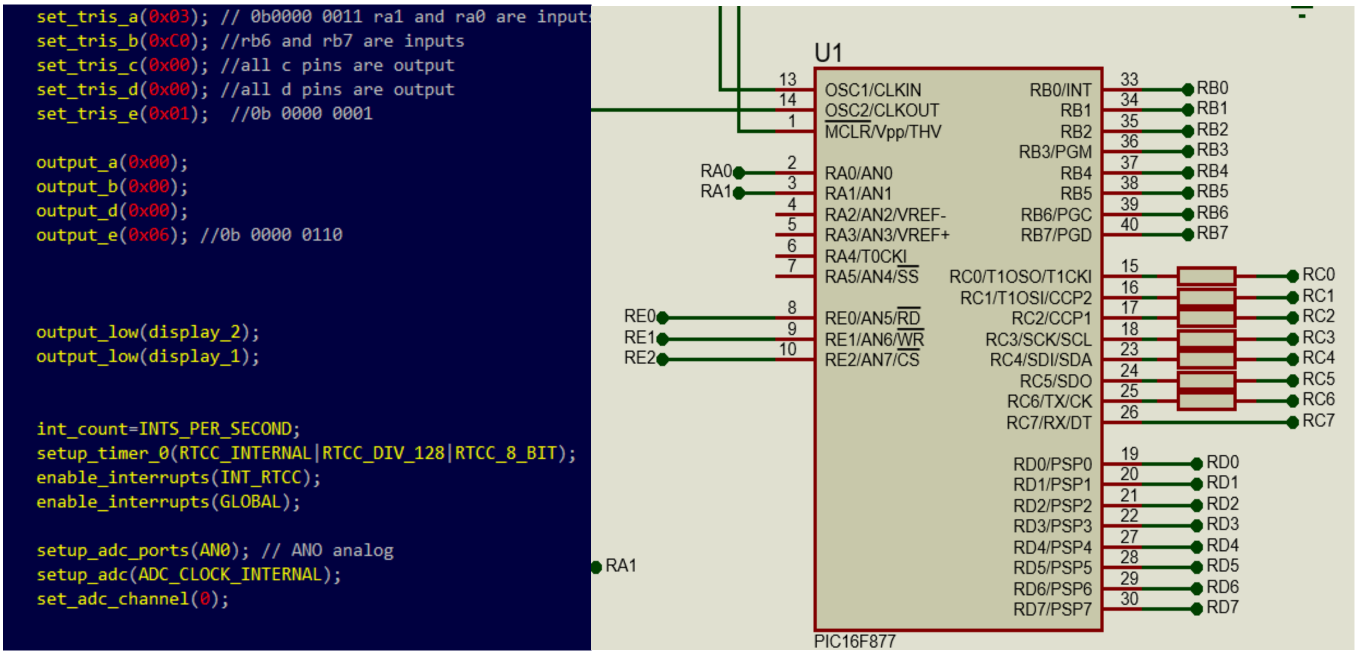
With for j loop, we look for every bits to shot the logo on the screen.

With t = t * 2, we move from the first bits to the next bit with multiplying the bit to switch to the left next valued bit.

FIG.12

MAIN FUNCTION

Main function starts with setups and initializations. PSP unit is disabled, timer1 and timer2 is disabled. Capture and Compare units (CCP) are disabled. And then Tris settings are made. As you can see below, the codes and the corresponding pin connections are shown.



```
set_tris_a(0x03); // 0b0000 0011 ra1 and ra0 are input
set_tris_b(0xC0); //rb6 and rb7 are inputs
set_tris_c(0x00); //all c pins are output
set_tris_d(0x00); //all d pins are output
set_tris_e(0x01);  //0b 0000 0001

output_a(0x00);
output_b(0x00);
output_d(0x00);
output_e(0x06); //0b 0000 0110



output_low(display_2);
output_low(display_1);


int_count=INTS_PER_SECOND;
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_128|RTCC_8_BIT);
enable_interrupts(INT_RTCC);
enable_interrupts(GLOBAL);

setup_adc_ports(AN0); // ANO analog
setup_adc(ADC_CLOCK_INTERNAL);
set_adc_channel(0);
```

FIG.13

set_tris_a(0x03) in binary is 0b 0000 0011, and therefore we used RA0 and RA1 pins as inputs from the potentiometer and Apply (Onay) button.

set_tris_b(0xC0) in binary is 0b 1100 0000, RB6 and RB7 are inputs from external EEPROM.

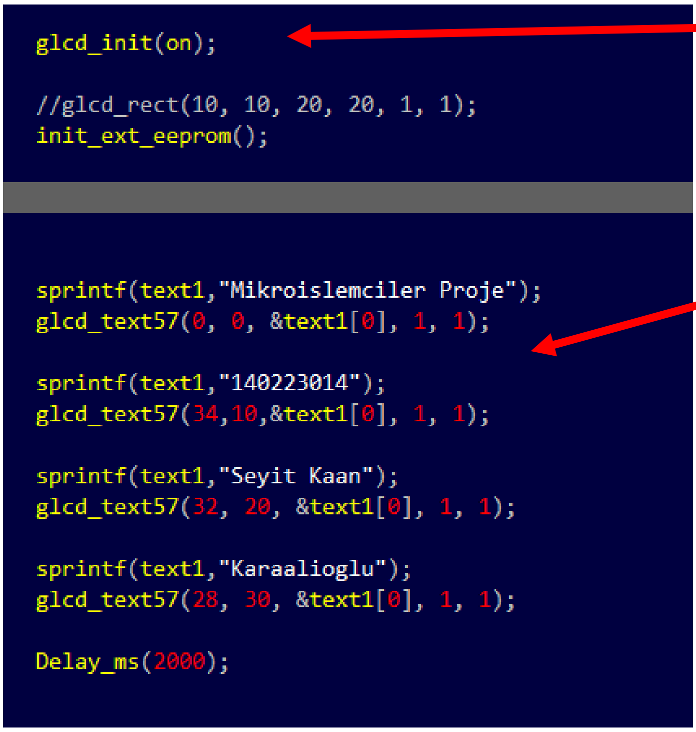
set_tris_c(0x00) and set_tris_d(0x00) defines as full outputs. set_tris_c(0x00) for the seven segment display and set_tris_d(0x00) for the GLCD.

output_a(0x00), output_b(0x00) and output_d(0x00), because they are basically feed from other external sources. But output_e(0x06), in binary, 0b 0000 0110, is used to power the seven segment displays(PINS E0 AND E1).

We start the main by lowering the display values to prevent previous values appearing. Then as mentioned before, INTS_PER_SECOND global variables is equated to int_count counter variable.

We set the timer with setup_timer_0 internal with RTCC_DIV_128 and 8 bits. We enable the internal interrupts and we allow the GLOBAL interrupts.

Then we initialize analog to digital converter, used in AN0 in my project. We allow the internal ADC clock. And we set the ADC in channel 0 as in ccs example projects.

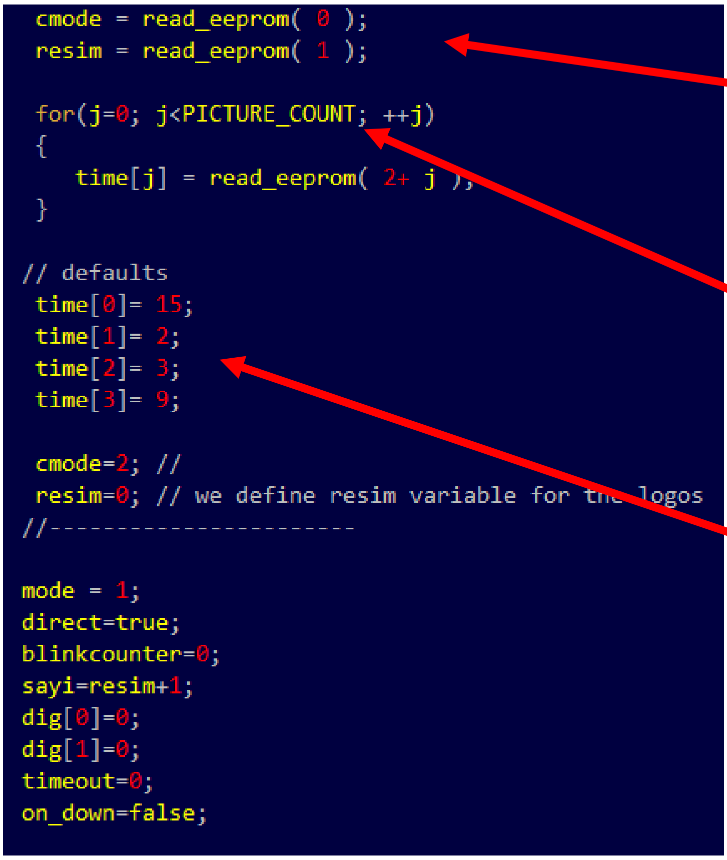


```
  glcd_init(on);

  //glcd_rect(10, 10, 20, 20, 1, 1);
  init_ext_eeprom();



  sprintf(text1,"Mikroislemciler Proje");
  glcd_text57(0, 0, &text1[0], 1, 1);

  sprintf(text1,"140223014");
  glcd_text57(34,10,&text1[0], 1, 1);

  sprintf(text1,"Seyit Kaan");
  glcd_text57(32, 20, &text1[0], 1, 1);

  sprintf(text1,"Karaalioglu");
  glcd_text57(28, 30, &text1[0], 1, 1);

  Delay_ms(2000);
```

FIG.14

Main function continues with the glcd_init(on) to power the GLCD and initializing the external EEPROM.

Then we show the name of the lesson, student ID and student name and we wait for a couple of seconds.



```
  cmode = read_eeprom( 0 );
  resim = read_eeprom( 1 );

  for(j=0; j<PICTURE_COUNT; ++j)
  {
     time[j] = read_eeprom( 2+ j );
  }

 // defaults
  time[0]= 15;
  time[1]= 2;
  time[2]= 3;
  time[3]= 9;

  cmode=2; //
  resim=0; // we define resim variable for the logos
 //-----------------------

 mode = 1;
 direct=true;
 blinkcounter=0;
 sayi=resim+1;
 dig[0]=0;
 dig[1]=0;
 timeout=0;
 on_down=false;
```

FIG.15

In case of shutdown, we read the values of cmode and resim, from the eeprom memory. This values written to the eeprom during the slide.(We will come upon that later.)

PICTURE_COUNT global variable is 4, showing the number of images. We read the time intervals from internal eeprom memory.

When I was trying the timing of the pictures first, I defined some default time variables to calculate the time using my telephone and Proteus.

mode 1 variable is just a checking condition in the upcoming switch-case statement. I could start it with 0 and develop the switch-case according to that.

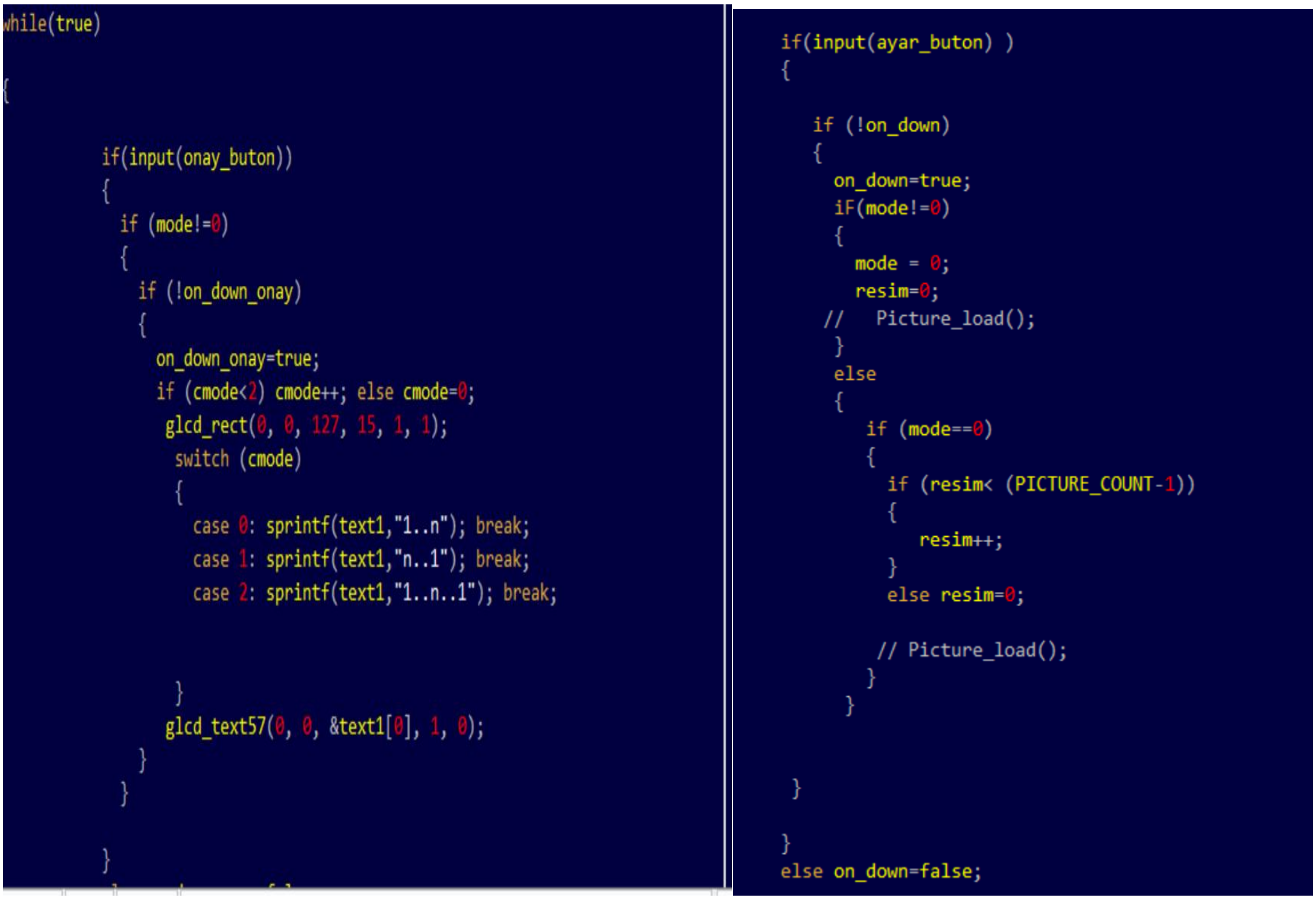
direct variable is just a label allowing us to locate inside the switch statements. Direct = true means the pictures appearing on the screen, direct false means we are in switch 0.

Blinkcounter starts with 0 and the sayi is incremented to 1 in the main.

Seven segment displays start with 0 initial values like 0 timeout. After main function, while(true) infinite loop starts checking whether the onay_buton or ayar_buton is pressed and then we check for whether mode equals to zero or not, if it is not zero which is ‘ayar modu’, ‘onay buton’ is executed. Then we check for onay_down_onay and on_down conditions to enter to the loop. When the on_down_onay turns true, cmode is shown on the screen. Forwards, backwards and random conditions are shown on the upper left of the screen. If the onay_buton is not pressed, else statement is executed.

Variable ayar_buton works the same, except this time, we look for on_down condition. When the ayar_buton is pressed, on_down becomes equal to true. If it’s not in mode 0, I mean, if the onay_buton is pressed, we reset the mode and resim variables to their 0 values.

If we are in ayar mode, every time we press the button, images will increment on the screen. Now, let us look at the codes.

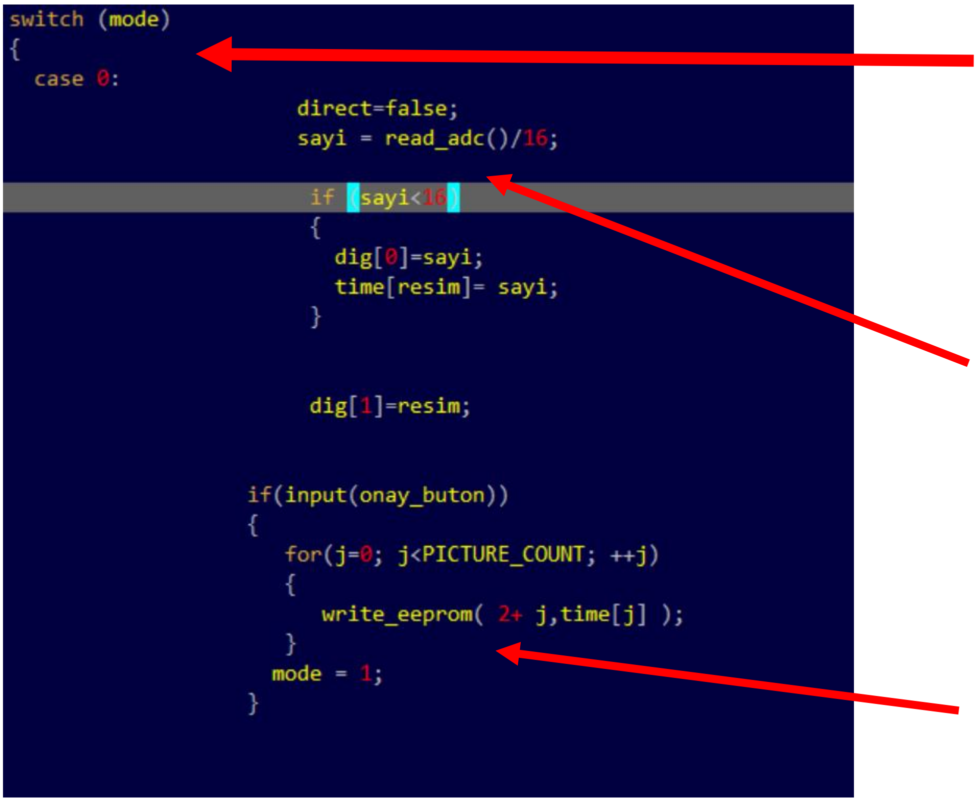


```
while(true)

{


        if(input(onay_buton))
        {
          if (mode!=0)
          {
            if (!on_down_onay)
            {
              on_down_onay=true;
              if (cmode<2) cmode++; else cmode=0;
               glcd_rect(0, 0, 127, 15, 1, 1);
                switch (cmode)
                {
                  case 0: sprintf(text1,"1..n"); break;
                  case 1: sprintf(text1,"n..1"); break;
                  case 2: sprintf(text1,"1..n..1"); break;


                }
               glcd_text57(0, 0, &text1[0], 1, 0);
            }
          }

        }
```

```
if(input(ayar_buton) )
{

   if (!on_down)
   {
     on_down=true;
     iF(mode!=0)
     {
       mode = 0;
       resim=0;
    //   Picture_load();
     }
     else
     {
        if (mode==0)
        {
          if (resim< (PICTURE_COUNT-1))
          {
             resim++;
          }
          else resim=0;

         // Picture_load();
        }
      }


 }

}
else on_down=false;
```

Statements to control the button functions FIG.16

Now, let us look at how the following switch-case statements work.



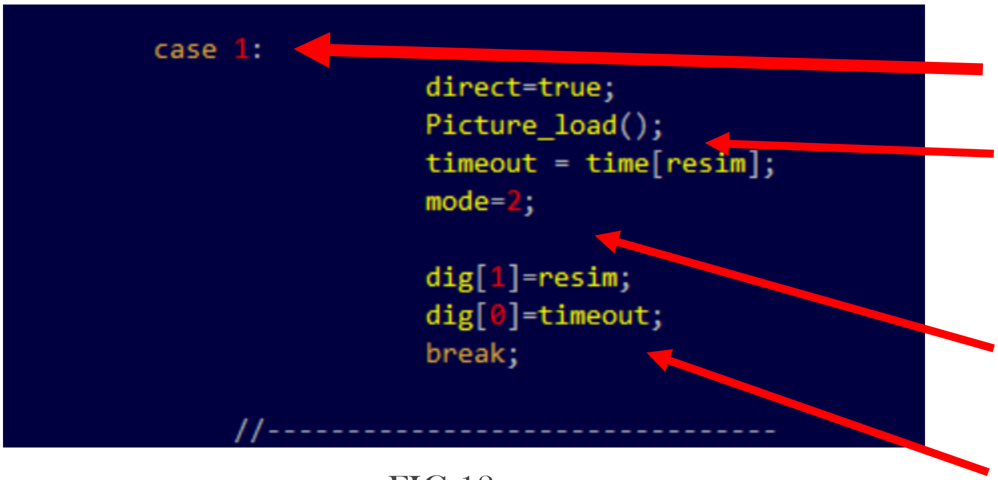
```
switch (mode)
{
  case 0:
                      direct=false;
                      sayi = read_adc()/16;

                       if (sayi<16)
                       {
                         dig[0]=sayi;
                         time[resim]= sayi;
                       }


                       dig[1]=resim;


                   if(input(onay_buton))
                   {
                      for(j=0; j<PICTURE_COUNT; ++j)
                      {
                        write_eeprom( 2+ j,time[j] );
                      }
                     mode = 1;
                   }
```

Switch(mode) statement is where we decide in which loop is executed. Mode 0 was editing button for the time settings while the Mode 1 is used for Picture_Load function to show the pictures.

We start by reading our ADC value and equating this value to sayi variable. We send this variable to the dig[0] variable to time array to keep track of the timing . dig[1] is the first seven segment on the left and shows the number of picture.

When the onay button is pressed, for every picture, we write the order of the picture and the timing for every picture to the EEPROM.

FIG.17



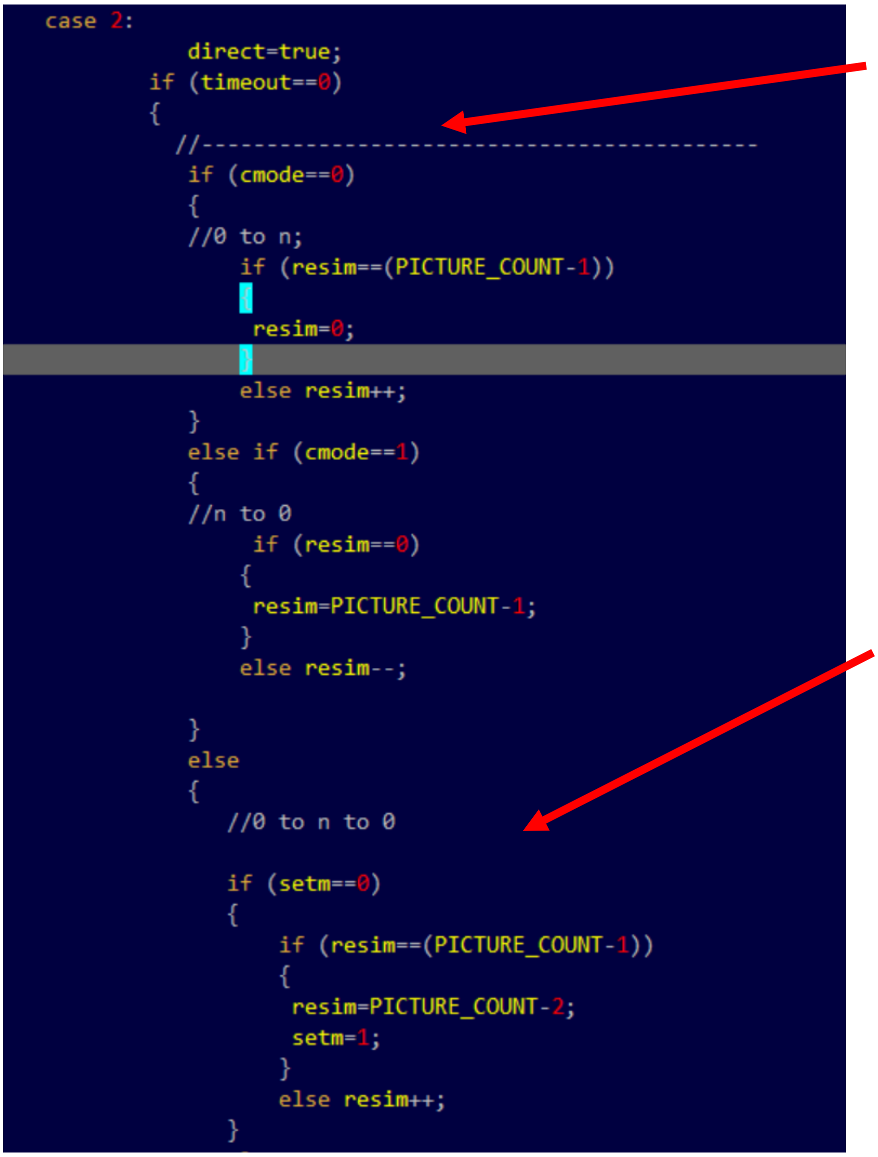
```
            case 1:
                          direct=true;
                          Picture_load();
                          timeout = time[resim];
                          mode=2;

                          dig[1]=resim;
                          dig[0]=timeout;
                          break;

                //--------------------------------
```

Case 1 is basically loads the icons to the screen in successive order. Time[resim] element in the array Equated to timeout to keep track of timing.

Then mode is equated to 2, to show the icons forwards and backwards and then in random order.

Dig[1] is the picture on the left of the seven segment while dig[0] is the time.

FIG.18



```
case 2:
           direct=true;
        if (timeout==0)
        {
          //------------------------------------------
           if (cmode==0)
           {
           //0 to n;
               if (resim==(PICTURE_COUNT-1))
               {
                resim=0;
               }
               else resim++;
           }
           else if (cmode==1)
           {
           //n to 0
                if (resim==0)
               {
                resim=PICTURE_COUNT-1;
               }
               else resim--;

           }
           else
           {
              //0 to n to 0

              if (setm==0)
              {
                  if (resim==(PICTURE_COUNT-1))
                  {
                   resim=PICTURE_COUNT-2;
                   setm=1;
                  }
                  else resim++;
              }
```

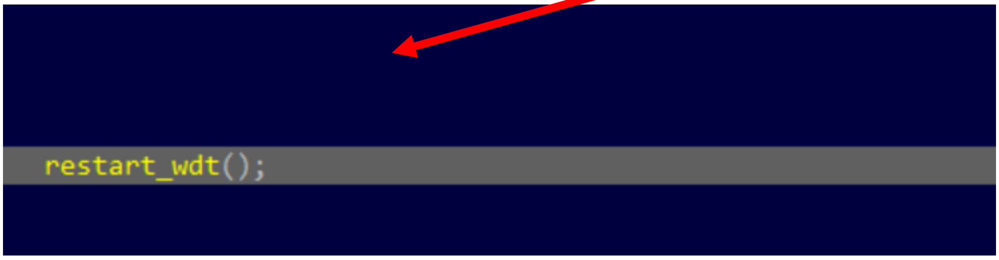
Case 2 starts with timeout control, then if cmode equals to zero, we start incrementing the pictures in 0 to n order then if cmode equals to 1, we decrement the pictures with resim = PICTURE_COUNT – 1 statement.

Then when setmode equals to zero, we show the pictures, 0 to n and n to 0 order.

Case 2 statements followed by saving the picture showed to the EEPROM in case of an emergency shutdown.

FIG.19

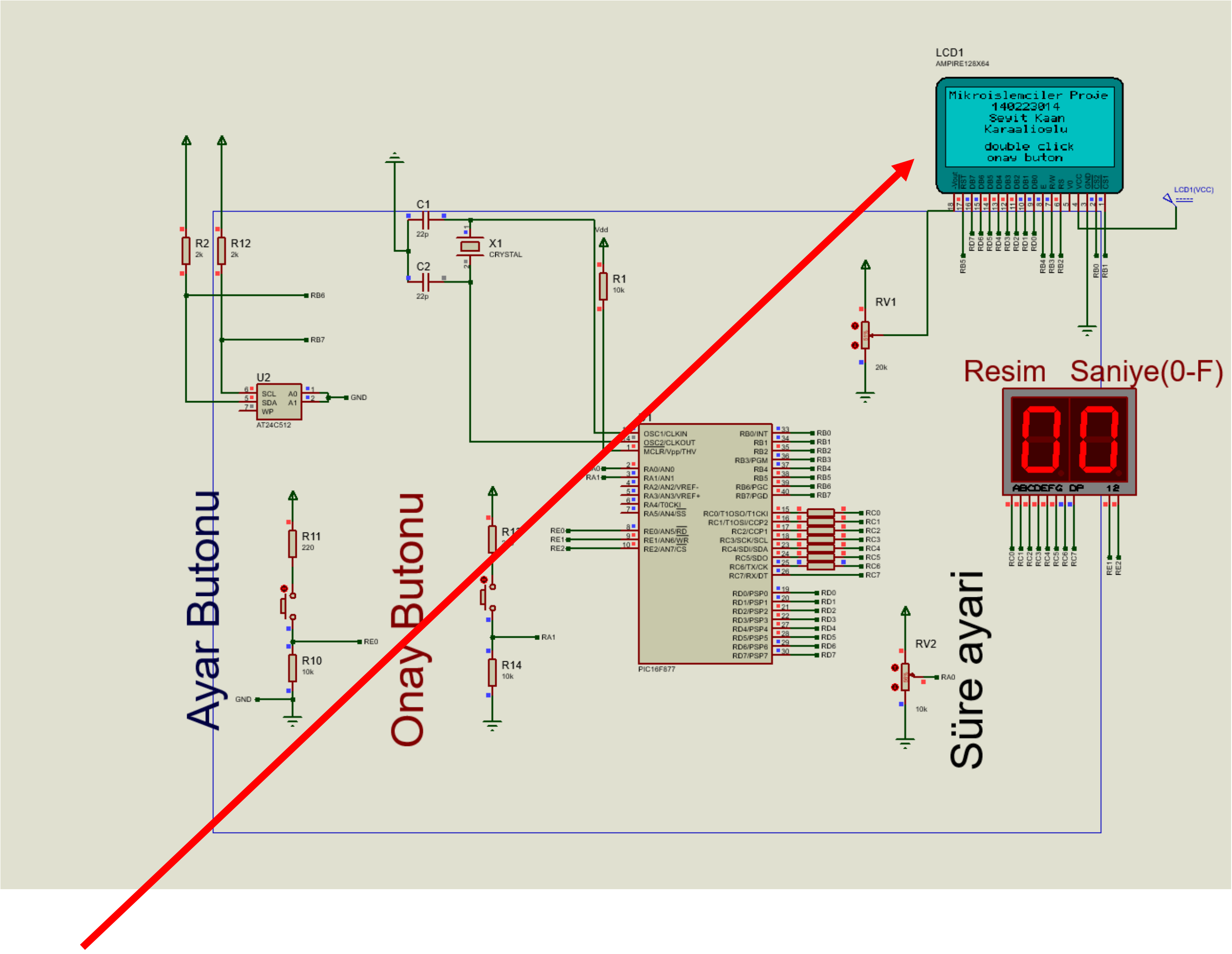
Watchdog timer is restarted every time the while loop is executed.



```
restart_wdt();
```

FIG.20

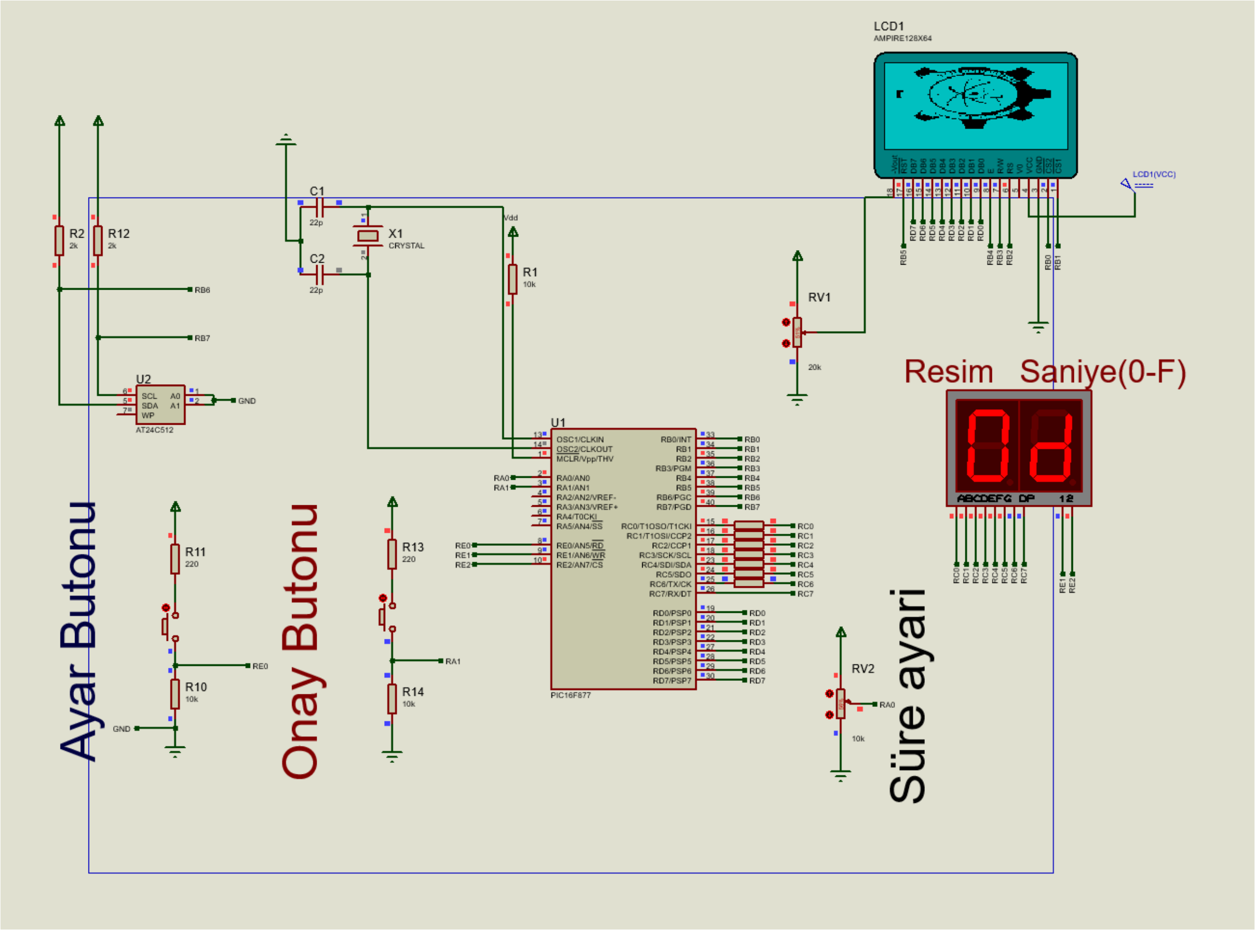
SIMULATIONS FROM THE PROJECT



Mikroislemciler Proje
140223014
Seyit Kaan
Karaalioglu

double click
onay buton

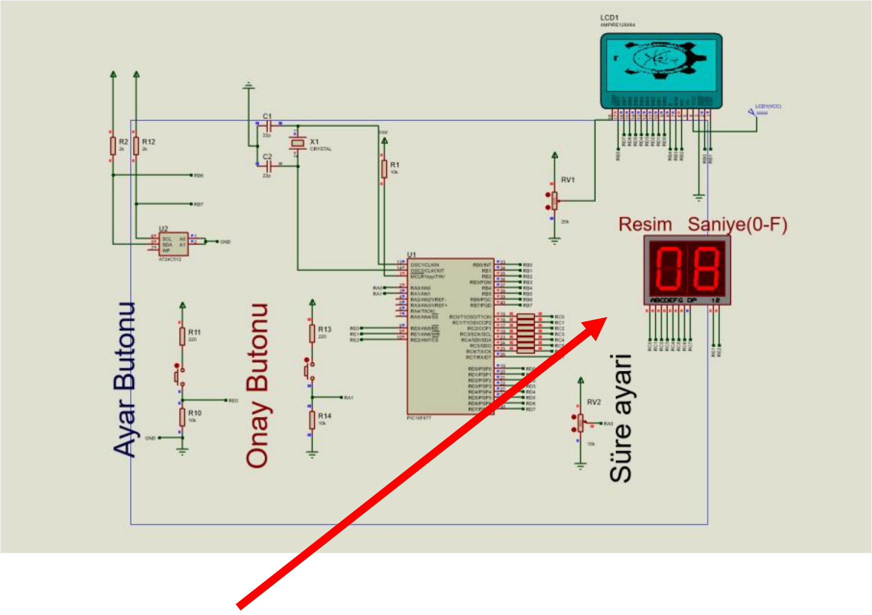
Project starts with welcoming the user and showing the seven segments displays and then starts with manual mode and decrement the seconds on the right part of the screen.



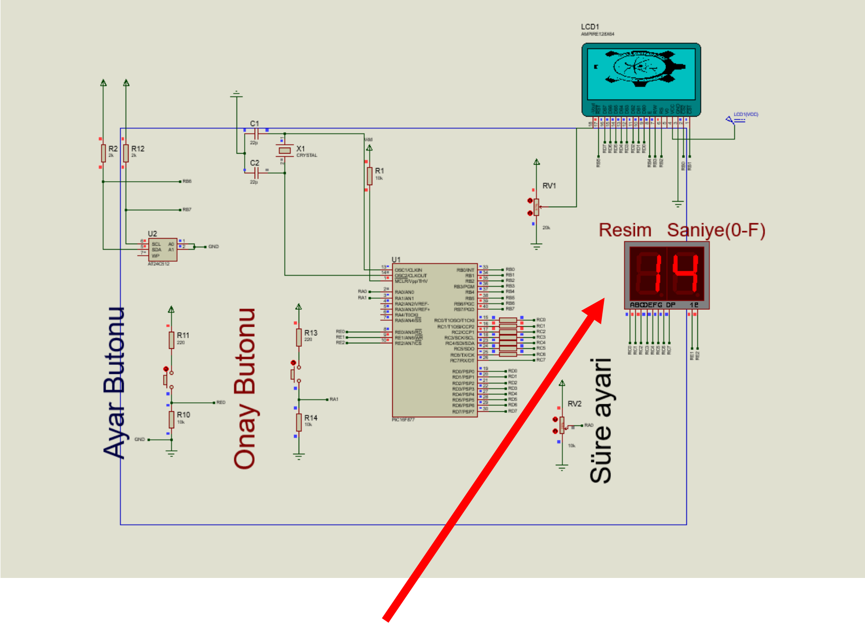
Then the first picture with the value in the EEPROM starts and the seconds are decremented on the right side of the secrren.

After the ayar button is double clicked, the pictures can be selected with every press to the ayar button. Let’s see the how the other image is selected.

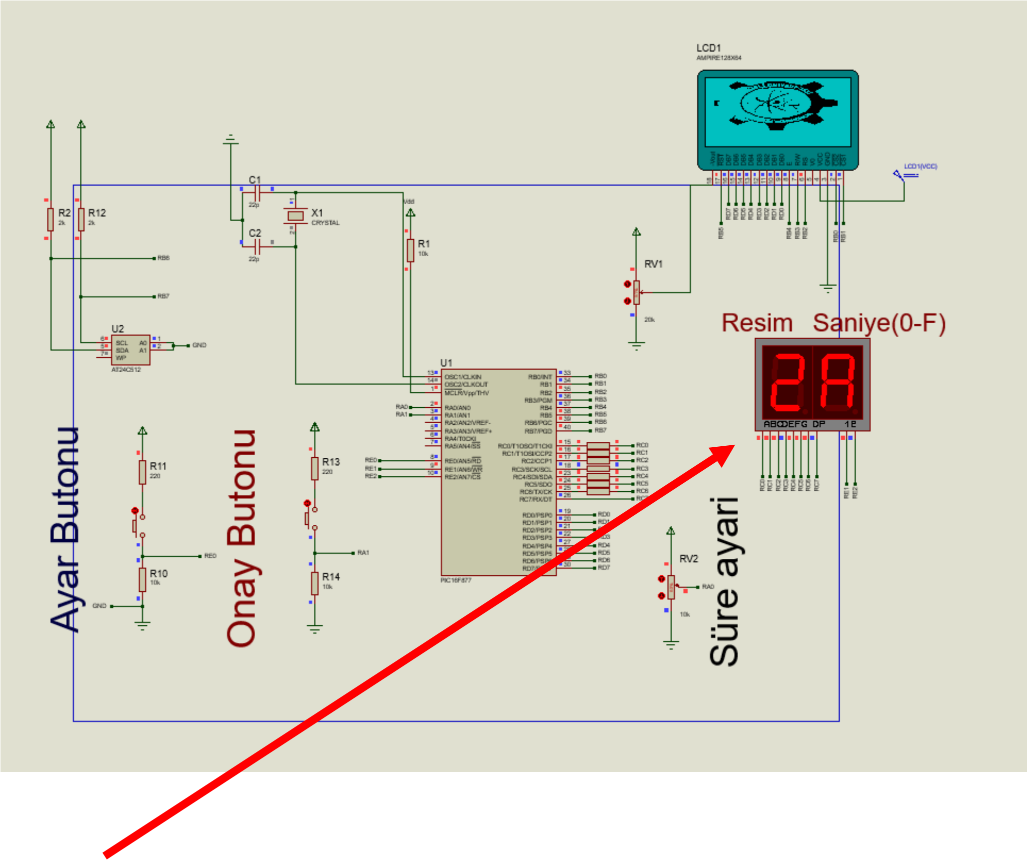
After the picture is showed, we can double click ayar button and change time with the potentiometer and see the seconds in the display, with the onay buton the value is saved.



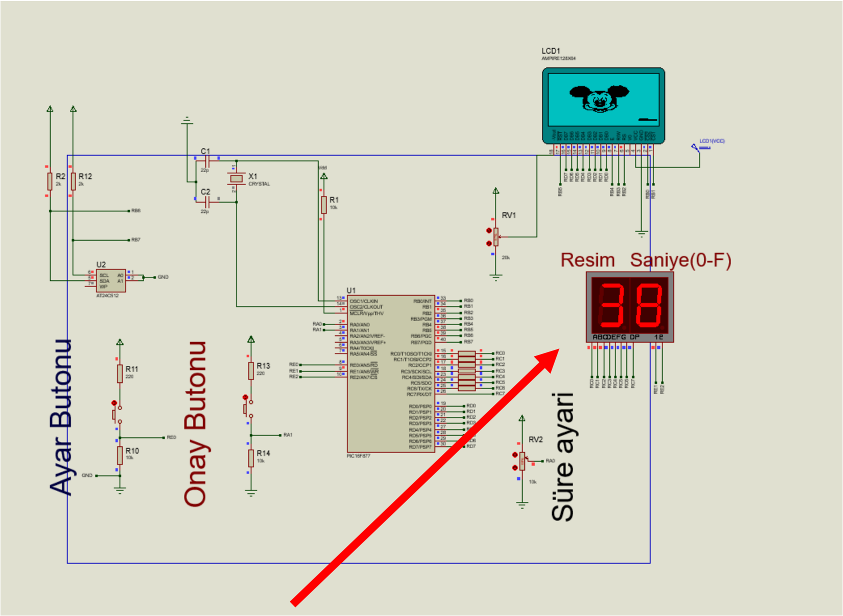
Time interval can be set with a potentiometer and the value is shown on the display.



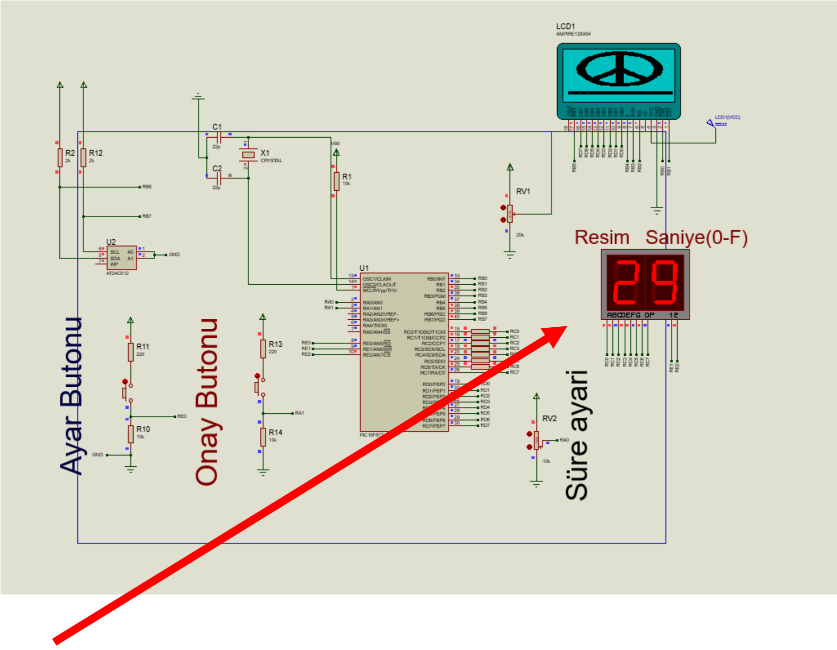
After ayar button is pressed, we switch to the first resim.



Timing for the second picture is set.



Third picture is shown above



Second picture is shown.

CODES

```
/*
Mechatronics Engineering Microprocessors Final Project
Slide Show Using GLCD
Author : Seyit Kaan Karaalioglu
ID     : 140223014
                              Date: 29/12/2020
*/



#if defined(__PCM__)
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)

#endif



// fast_io initializations
#use fast_io(a)
//#use fast_io(b)
#use fast_io(c)
//#use fast_io(d)
#use fast_io(e)



//after setting up the time, we start displaying by choosing start button
#define ayar_buton pin_e0
#define onay_buton pin_a1

//7 segment define statements
#define display_1 pin_e1
#define display_2 pin_e2
```

```
// GLCD define and include statements

#define GLCD_CS1  pin_b0  // chip select 1
#define GLCD_CS2  pin_b1 // chip select 2
#define GLCD_DI   pin_b2 // Data or instruction input
#define GLCD_RW   pin_b3 // Read and Write pin
#define GLCD_E    pin_b4 // GLCD enable
#define GLCD_RST  pin_b5 // GLCD reset
#include <GLCD.c>


//EEPROM define and include statements
#define EEPROM_SDA pin_b6
#define EEPROM_SCL pin_b7
#include <24512.c>



#define INTS_PER_SECOND 152     // (20000000/(4*128*256))
#define PICTURE_COUNT 4         // number of pictures to show up

int numbers[16] ={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07, 0x7F, 0x6F, 0x77 , 0x7C, 0x39, 0x5E,
0x79, 0x71};
// numbers as hex codes for two digit seven segments

int16 resim, blinkcounter;
unsigned char mode, cmode, setm, sayi, dig[2];
char text1[22];
int1 blink, direct;
int8 seg;
```

```
int8 timeout, time[PICTURE_COUNT];      // A running seconds counter
int8 int_count;    // Number of interrupts left before a second has elapsed
int1 on_down, on_down_onay; // on_down and on_down_onay are defined as booleans

/*
void sevenseg(void)
{
   if (blink)
   {



      output_c( numbers[ dig[0] ] );
      output_low(display_2);
      Delay_ms(2);
      output_high(display_2);

      output_c( numbers [dig[1]] );
      output_low(display_1);
      Delay_ms(2);
      output_high(display_1);

   }

}
*/




#int_rtcc
void clock_isr()
{

  //------------------------------------------

   if (blinkcounter>5)
   {
        blinkcounter=0;
        blink=!blink;    // blink happens
   }
  else blinkcounter++;

   //---------------------------------------------
   output_high(display_1);
   output_high(display_2);

   if ((blink)||(direct))
   {
     if (seg==0)
     {
       seg=1;
       output_low(display_1);
     }
     else
     {
       seg=0;
       output_low(display_2);
     }

      output_c(numbers[dig[seg]]);

   }
   //---------------------------------------------
   if(--int_count==0)
   {             // per second.
     if (timeout>0) timeout--;
     int_count=INTS_PER_SECOND;
   }
}

void Picture_load()
```

```
{
int16 i,j,x,y,t,start_adr;
 unsigned char a;

                         x=127;
                         y=0;
                         t=0;
                         start_adr = 61 + (resim*1024)+1024;

                         for(i = 0 ; i < 1024 ; i++)
                         {
                             a = read_ext_eeprom(start_adr);

                             t=1;
                             for(j = 0; j < 8; j++)
                             {
                               glcd_pixel( x, y,    ( (a & t)==0) );

                               if (x==0)
                               {
                                  x=127;
                                   y++;
                                }
                                else x--;

                                 t=t*2;
                             }

                            start_adr--;
                          }
}

void main()
{
   int j;
```

```
setup_psp(PSP_DISABLED);
setup_timer_1(T1_DISABLED);    // timer1 disabled
setup_timer_2(T2_DISABLED,0,1);  // timer 2 disabled
setup_CCP1(CCP_OFF);
setup_CCP2(CCP_OFF);

set_tris_a(0x03); // 0b0000 0011 ra1 and ra0 are inputs
set_tris_b(0xC0); //rb6 and rb7 are inputs
set_tris_c(0x00); //all c pins are output
set_tris_d(0x00); //all d pins are output
set_tris_e(0x01);  //0b 0000 0001

output_a(0x00);
output_b(0x00);
output_d(0x00);
output_e(0x06); //0b 0000 0110



output_low(display_2);
output_low(display_1);



int_count=INTS_PER_SECOND;
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_128|RTCC_8_BIT);
enable_interrupts(INT_RTCC);
enable_interrupts(GLOBAL);

setup_adc_ports(AN0); // ANO analog
setup_adc(ADC_CLOCK_INTERNAL);
set_adc_channel(0);



glcd_init(on);
```

```
//glcd_rect(10, 10, 20, 20, 1, 1);
init_ext_eeprom();

/*
        for(j = 0; j < 8; j++)
        {
        a = read_ext_eeprom(j);
        sprintf(text1,"%02X ",a);
        glcd_text57(0, j*8, &text1[0], 1, 1);
        }
        */




sprintf(text1,"Mikroislemciler Proje");
glcd_text57(0, 0, &text1[0], 1, 1);

sprintf(text1,"140223014");
glcd_text57(34,10,&text1[0], 1, 1);

sprintf(text1,"Seyit Kaan");
glcd_text57(32, 20, &text1[0], 1, 1);

sprintf(text1,"Karaalioglu");
glcd_text57(28, 30, &text1[0], 1, 1);

sprintf(text1, "double click");
glcd_text57(28, 45, &text1[0], 1, 1);

sprintf(text1, "onay buton");
glcd_text57(30, 55, &text1[0], 1, 1);
```

```
Delay_ms(1500);



 //-----------------------------
 // load values from internal eeprom

 cmode = read_eeprom( 0 );
 resim = read_eeprom( 1 );

 for(j=0; j<PICTURE_COUNT; ++j)
 {
    time[j] = read_eeprom( 2+ j );
 }

// defaults
 time[0]= 15;
 time[1]= 2;
 time[2]= 3;
 time[3]= 9;

 cmode=2; //
 resim=0; // we define resim variable for the logos
//-----------------------

mode = 1;
direct=true;
blinkcounter=0;
sayi=resim+1;
dig[0]=0;
dig[1]=0;
timeout=0;
on_down=false;

//setup_wdt(WDT_2304MS);
```

```
while(true)

{

           if(input(onay_buton))
           {
             if (mode!=0)
             {
               if (!on_down_onay)
               {
                 on_down_onay=true;
                 if (cmode<2) cmode++; else cmode=0;
                  glcd_rect(0, 0, 127, 15, 1, 1);
                   switch (cmode)
                   {
                     case 0: sprintf(text1,"1..n"); break;
                     case 1: sprintf(text1,"n..1"); break;
                     case 2: sprintf(text1,"1..n..1"); break;



                   }
                  glcd_text57(0, 0, &text1[0], 1, 0);
               }
             }

           }
           else on_down_onay=false;



        if(input(ayar_buton) )
        {

           if (!on_down)
           {
             on_down=true;
```

```
      iF(mode!=0)
      {
        mode = 0;
        resim=0;
     //   Picture_load();
      }
      else
      {
         if (mode==0)
         {
           if (resim< (PICTURE_COUNT-1))
           {
              resim++;
           }
           else resim=0;

          // Picture_load();
         }
       }


  }

 }
 else on_down=false;





 switch (mode)
 {
   case 0:
                         direct=false;
                         sayi = read_adc()/16;
```

```
                    if (sayi<16)
                    {
                      dig[0]=sayi;
                      time[resim]= sayi;
                    }



                    dig[1]=resim+1;



                 if(input(onay_buton))
                 {
                    for(j=0; j<PICTURE_COUNT; ++j)
                    {
                        write_eeprom( 2+ j,time[j] );
                    }
                   mode = 1;
                 }




             /*
                 dig[0]=sayi;
                 if (sayi==15)
                 sayi=0; else sayi++;

               */



                 break;

      case 1:
                 direct=true;
                 Picture_load();
                 timeout = time[resim];
```

```
                mode=2;


                dig[1]=resim;
                dig[0]=timeout;
                break;

      //--------------------------------
  case 2:
             direct=true;
          if (timeout==0)
          {
           //------------------------------------------
             if (cmode==0)
             {
             //0 to n;
                 if (resim==(PICTURE_COUNT-1))
                 {
                  resim=0;
                 }
                 else resim++;
             }
             else if (cmode==1)
             {
             //n to 0
                  if (resim==0)
                 {
                  resim=PICTURE_COUNT-1;
                 }
                 else resim--;

             }
             else
             {
                //0 to n to 0
                if (setm==0)
                {
                    if (resim==(PICTURE_COUNT-1))
                    {
                     resim=PICTURE_COUNT-2;
                     setm=1;
                    }
                    else resim++;
                }
                else
                {

                   if (resim==0)
                   {
                     resim=1;
                     setm=0;
                   }
                   else resim--;
                }


             }
           write_eeprom( 2, resim );
           //-------------------------------------------




           mode=1;
           sayi = resim+1;
          }

          dig[1]=resim;
          dig[0]=timeout;
          break;
```

```
      }



    restart_wdt();


}


}
```

REFERENCES

- CCS manual
- Embedded C Programming : techniques and applications of c and pic mcus – Mark Siegesmund
- Pic Microcontrollers – Know it all – Elsevier
- https://exploreembedded.com/wiki/Displaying_Images_and_Icons_on_GLCD
- https://www.ercankoclar.com/2018/11/at24c512-eeprom-mikroc-kutuphanesi/
- https://www.dataman.com/media/datasheet/Atmel/AT24C512.pdf
- https://en.wikipedia.org/wiki/BMP_file_format