Faculty of Engineering & Technology Electrical & Computer Engineering Department

Computer Design Laboratory ENCS 4110

Experiment #11

LCD and ADC

Prepared by:
Hasan Hamed   1190496

Instructor: Dr. Abualseoud Hanani
Assistant: Eng. Moutasem Diab

Section: 1
Date: 22/8/2021

1. Abstract

The aim of the experiment is to understand how the LCD works and how to use it to display characters or any string, and to understand how to program it to use it in ARM microcontrollers, and to understand how the ADC works and how to use it to convert analog signals to digital values so embedded system can deal with it and finally, to display the output of ADC in LCD.

Table of Contents

2. Theory

2.1. Alphanumeric LCD display

This component is specifically manufactured to be used with microcontrollers, which means that it cannot be activated by standard IC circuits. It is used for displaying different messages on a miniature liquid crystal display. The model described here is for its low price and great capabilities most frequently used in practice (LM016L LCD). It is based on the HD44780 microcontroller (Hitachi) and can display messages in two lines with 16 characters each. It displays all the letters of alphabet, Greek letters, punctuation marks, mathematical symbols etc. In addition, it is possible to display symbols made up by the user. Other useful features include automatic message shift (left and right), cursor appearance, LED backlight etc.

Figure 2.1: 2x16 LCD.

2.1.1. Function Description:

Registers

The HD44780U has two 8-bit registers, an instruction register (IR) and a data register (DR). The IR stores instruction codes, such as display clear and cursor shift, and address information for display data RAM (DDRAM) and character generator RAM (CGRAM). The IR can only be written from the MPU. The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. Data written into the DR from the MPU is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation.

Memory

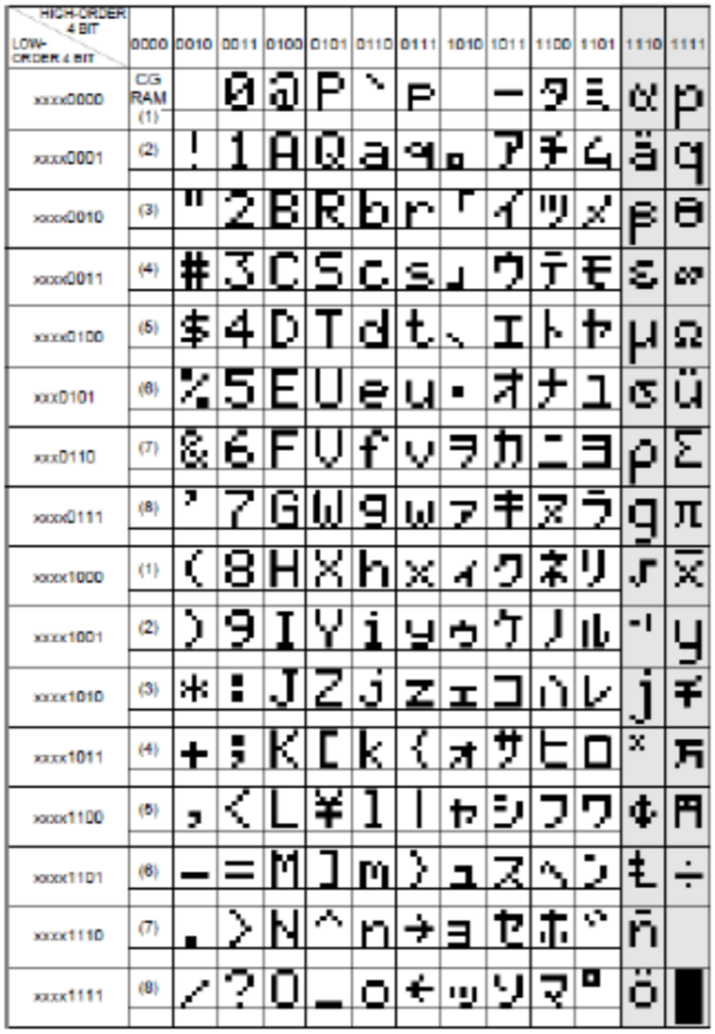
In 16×2 LCD controller HD44780, there are three memory are available to store characters, numbers and special symbols. Which are DDRAM (data display RAM) which stores ASCII codes, CGROM (character generating ROM) which is responsible for stored standard character pattern, and CGRAM (character generating RAM) which holds custom character pattern space total 8 in 2×16 module.

Display Data RAM (DDRAM)

Display data RAM(DDRAM)stores display data represented in 8-bit character codes. Its extended capacity is 80×8 bits, or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM.

Character Generator ROM (CGROM)

The character generator ROM that is responsible for stored standard character pattern generates 5×8 dot or 5×10 dot character patterns from 8-bit character codes. It can generate 208 5×8 dot character patterns and 32 5×10 dot character patterns. ¬ Character Generator RAM (CGRAM) The character generating RAM which holds custom character pattern has only 8 memory location available to store user defined characters with address 0x00 - 0x07, which is shown in the Figure 2.2.

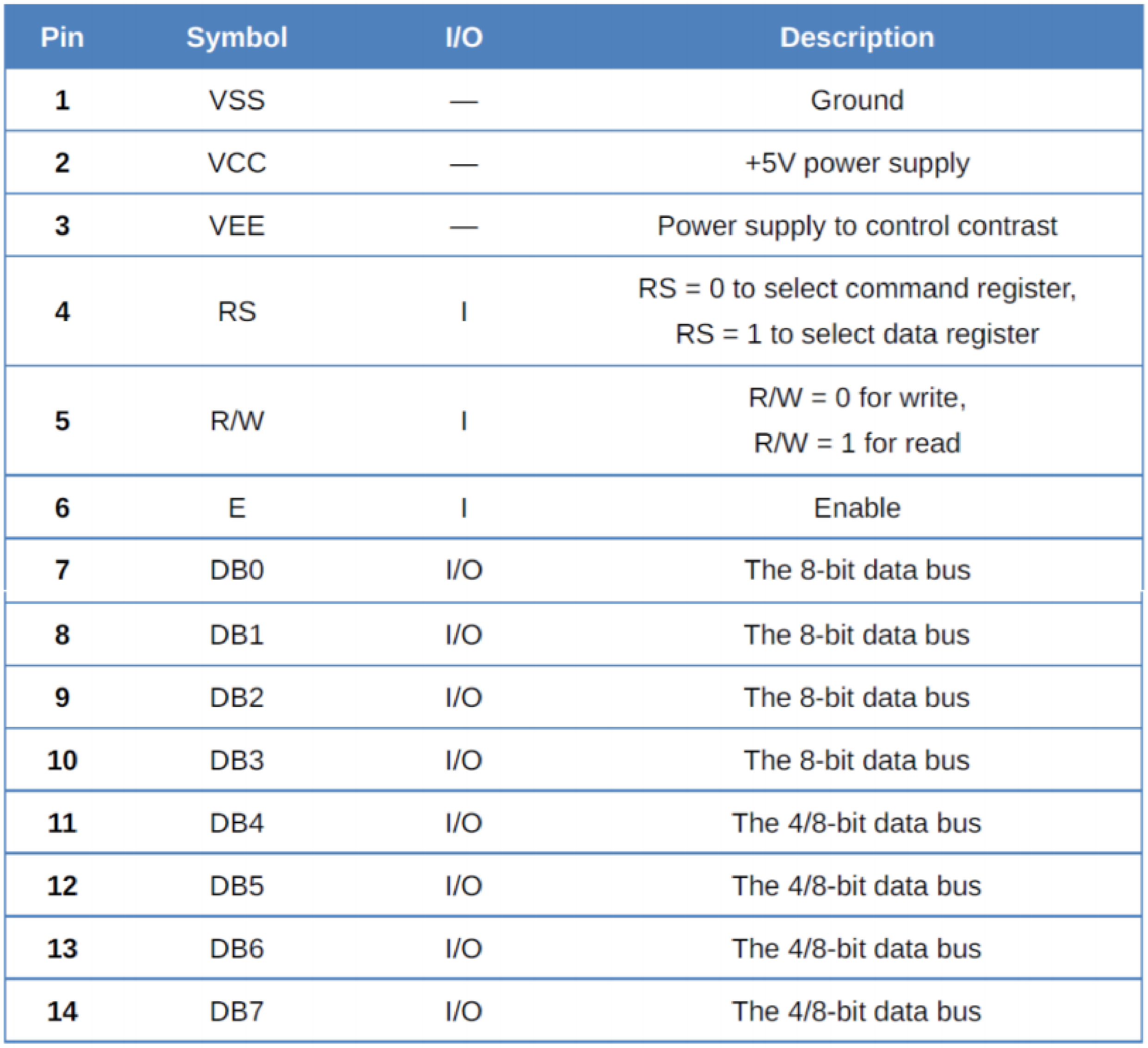


HIGH-ORDER 4 BIT / LOW-ORDER 4 BIT	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)		0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxx0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxx0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101	(6)	-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Figure 2.2: Characters Table.

2.1.2. LCD Display

Along one side of a small printed board there are pins used for connecting to the microcontroller. There are in total of 14 pins marked with numbers (16 if the backlight is built in). Their function is described in the Figure below.



Pin	Symbol	I/O	Description
1	VSS	—	Ground
2	VCC	—	+5V power supply
3	VEE	—	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 4/8-bit data bus
12	DB5	I/O	The 4/8-bit data bus
13	DB6	I/O	The 4/8-bit data bus
14	DB7	I/O	The 4/8-bit data bus

Figure 2.3: Pin Descriptions for LCD.

2.1.3. LCD Screen Modes

D0-D7 is the data bus and is used to pass commands and characters to the LCD. Data can be transferred to and from the display either as a single 8-bit byte or two 4-bit nibbles. In the second case only the upper four data lines (D4-D7) are used. This 4-bit mode is beneficial when using a microcontroller with few input/output pins available.

2.1.4. Displaying Standard Character on LCD

Out of these three memory locations, DDRAM and CGROM are used to generate regular standard characters (ASCII characters). By using these three memory locations, a user can generate different character fonts and symbols on LCD display. A character font describes the shape and style of the character. Each shape of a character is designed by taking the number of pixels in mind. For example, in 16x2 LCD there are 16 segments available per single line. Each segment contains pixels in 5x7 or 5x10 matrix forms.

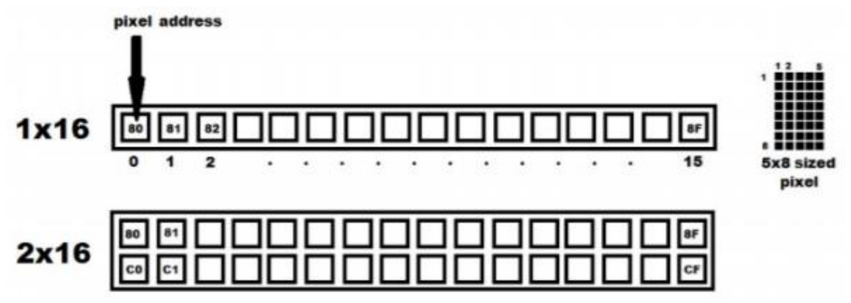
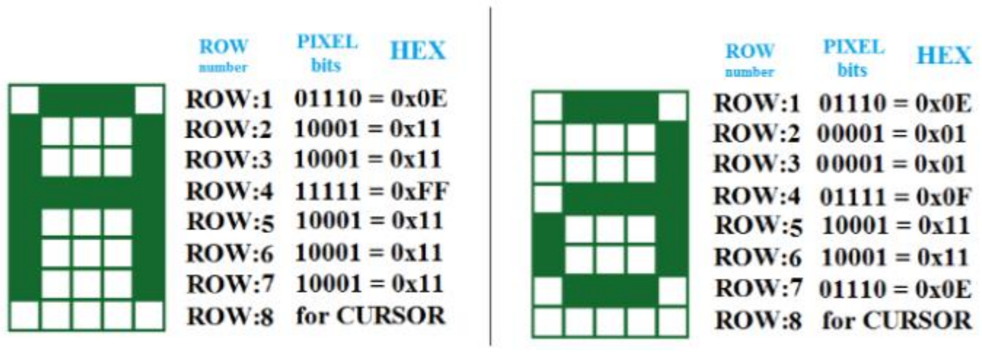


Figure 2.4: LCD Segments per Line.

For example, in 16×2 LCD there are 16 segments available per single line. Each segment contains pixels in 5×8 or 5×10 matrix forms. For example, a character in both uppercase 'A' and lowercase 'a' is designed by energizing corresponding pixels as shown below.



ROW number	PIXEL bits	HEX
ROW:1	01110 =	0x0E
ROW:2	10001 =	0x11
ROW:3	10001 =	0x11
ROW:4	11111 =	0xFF
ROW:5	10001 =	0x11
ROW:6	10001 =	0x11
ROW:7	10001 =	0x11
ROW:8	for CURSOR	

ROW number	PIXEL bits	HEX
ROW:1	01110 =	0x0E
ROW:2	00001 =	0x01
ROW:3	00001 =	0x01
ROW:4	01111 =	0x0F
ROW:5	10001 =	0x11
ROW:6	10001 =	0x11
ROW:7	01110 =	0x0E
ROW:8	for CURSOR	

Figure 2.5: Capital and Small 'A' Representation in Pixels.

All these eight hexadecimal codes (referred as character pattern) of each character are stored in character generator ROM (CGROM) area.

The Display Data RAM (DDRAM) stores the ASCII code of a character which is sent by the microcontroller. Now the LCD controller (HD44780) maps the corresponding ASCII Code in DDRAM with CGROM address to bring the hexadecimal codes (character pattern) of that particular character. By using those hexadecimal codes, the 5x7 matrix segment will light according to that character pattern to display corresponding character on it as shown in Figure 2.6.

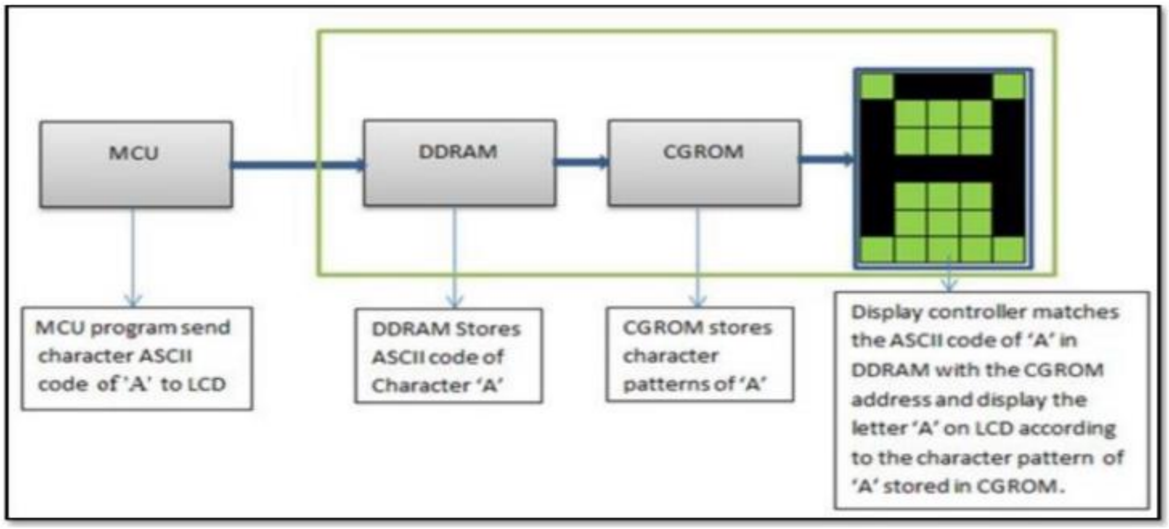

MCU
DDRAM
CGROM
MCU program send character ASCII code of 'A' to LCD
DDRAM Stores ASCII code of Character 'A'
CGROM stores character patterns of 'A'
Display controller matches the ASCII code of 'A' in DDRAM with the CGROM address and display the letter 'A' on LCD according to the character pattern of 'A' stored in CGROM.

Figure 2.6: Block Diagram for Characters Generation in LCD.

2.1.5. Displaying Custom Characters on LCD display

To create custom characters on LCD, the display controller (HD44780) make use of CGRAM area to store hexadecimal codes (character pattern) which are designed by user. In addition to CGRAM area, DDRAM area is also used to store the CGRAM address of a particular character, which is sent by microcontroller in hexadecimal format.

2.2. Analog-to-Digital Conversion (ADC)

Analog-to-digital conversion (ADC) is necessary because, while embedded systems deal with digital values (as we have deal with keypads and switches). Analog signals such as, temperature, speed and pressure are generated by peripheral devices such as microphones, analog cameras, sensors, and etc. They all need to be converted into digital data before being processed by the microcontroller. Figure 7.1 shows microcontroller connection to sensor via ADC.

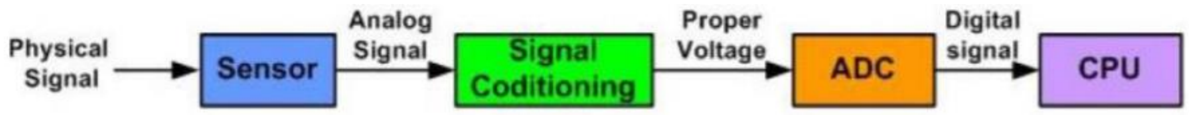


Figure 2.7: Microcontroller Connection to Sensor Via ADC.

Signals in the real world are analog: light, sound, etc. So, real-world signals must be converted into digital, using a circuit called ADC (Analog-to-Digital Converter), before they can be manipulated by digital equipment such as microcontroller. Let's say you have a sound wave, and you wish to sample it with an ADC. Here is a typical wave:

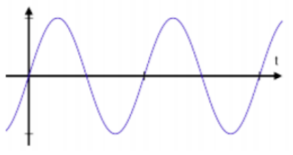


Figure 2.8: Sine Analog Wave.

When you sample the wave with an analog-to-digital converter, you have control over three variables:

The sampling rate: Controls how many samples are taken per second.

The sampling precision (resolution): Controls how many different gradations (quantization levels) are possible when taking the sample.

The reference voltages: The VREF + represents the maximum analog value that can be converted by the Analog to Digital converter. The VREF - represents the minimum analog value that can be converted by the Analog to Digital converter.

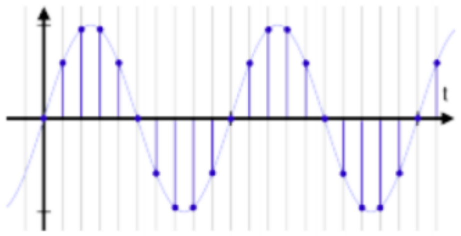


Figure 2.9: Sampling Process.

You can see that as the sampling rate and precision (resolution) increase, the similarity between the original wave and the ADC's output improves.

The Figure below shows an analog signal and quantized versions for several different number of quantization levels. With L levels, we need N=log2 L bits to represent the different levels or conversely, with N bits we can represent L = 2N levels.

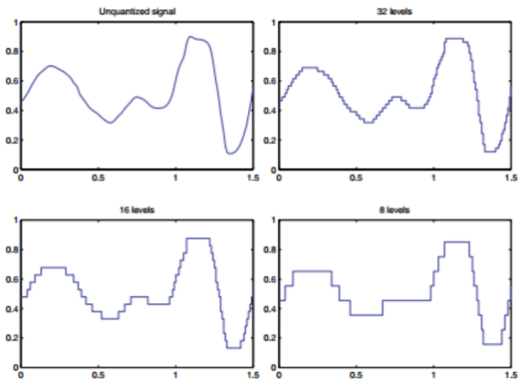


Figure 2.10: Quantized Signals with Different Levels.

2.2.1. LPC2138 ADC Programming

Registers used for ADC Programming in LPC2138

1. AD0CR– A/D Control Register: This is the main control register for AD0.

1. Bits [7 to 0] – SEL: This group of bits are used to select the pins (Channels) which will be used for sampling and conversion. Bit ‘x'(in this group) is used to select pin A0.x in case of AD0.

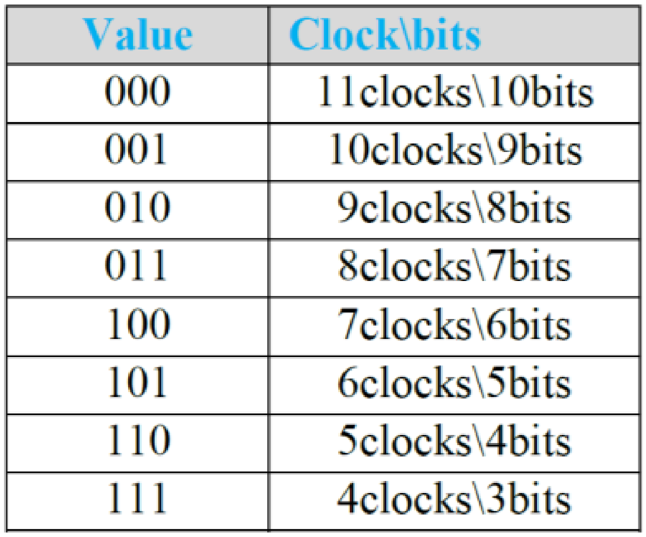
2. Bits [15 to 8] – CLKDIV: These bits store the value for CLKDIV which is used to generate the ADC clock. Peripheral clock i.e., PCLK is divided by CLKDIV+1 to get the ADC clock. Note that ADC clock speed must be <= 4.5Mhz! As per datasheet user must program the smallest value in this field which yields a clock speed of 4.5 MHz or a bit less.

3. Bit 16 – BURST: Set this to 1 for doing interrupted repeated conversions. Set this bit to 0 for software-controlled conversions, which take 11 clocks to finish.

4. Bits [19 to 17] – CLKS: These bits are used to select the number of clocks used for conversion in burst mode along with number of bits of accuracy of the result in RESULT bits of ADDR.

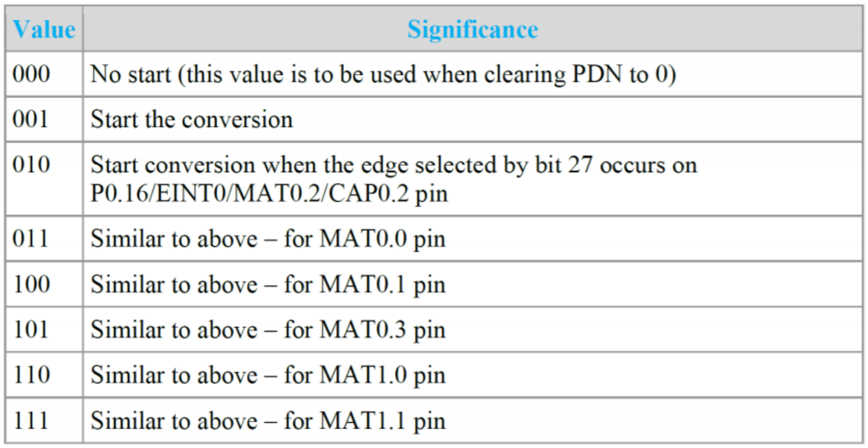
5. Bit 21 – PDN: Set it to 1 for powering up the ADC and making it operational. Set it to 0 for bringing it in power down mode.

6. Bits [26 to 24] – START: These bits are used to control the start of ADC conversion when BURST (bit 16) is set to 0. Below is the table as given in datasheet.



Value	Clock\bits
000	11clocks\10bits
001	10clocks\9bits
010	9clocks\8bits
011	8clocks\7bits
100	7clocks\6bits
101	6clocks\5bits
110	5clocks\4bits
111	4clocks\3bits

Figure 2.11: Bits [19 to 17] – CLKS



Value	Significance
000	No start (this value is to be used when clearing PDN to 0)
001	Start the conversion
010	Start conversion when the edge selected by bit 27 occurs on P0.16/EINT0/MAT0.2/CAP0.2 pin
011	Similar to above – for MAT0.0 pin
100	Similar to above – for MAT0.1 pin
101	Similar to above – for MAT0.3 pin
110	Similar to above – for MAT1.0 pin
111	Similar to above – for MAT1.1 pin

Figure 2.12: Bits [26 to 24] – START

7. Bit 27 – EDGE: Set this bit to 1 to start the conversion on falling edge of the selected CAP/MAT signal and set this bit to 0 to start the conversion on rising edge of the selected signal.

8. Other bits are reserved.

2. AD0GDR - A/D Global Data Register: This is the global data register for the corresponding ADC module. It contains the ADC's DONE bit and the result of the most recent A/D conversion.

1. Bits [15 to 6] – RESULT: Given DONE (below) is set to 1 these bits give a binary fraction which represents the voltage on the pin selected by the SEL field, divided by the voltage on Vref pin i.e., =V/Vref. A value of zero indicates that voltage on the given pin was less than, equal to or greater than Vssa. And a value of 0x3FF means that the voltage on the given pin was close to, equal to or greater than the reference voltage.

2. Bits [26 to 24] – CHN: It gives the channel from which RESULT bits were converted. 000 for channel 0, 001 for channel 1 and so on.

3. Bit 30 – OVERRUN: In burst mode this bit is 1 in case of an Overrun i.e., the result of previous conversion being lost(overwritten). This bit will be cleared after reading AD0GDR.

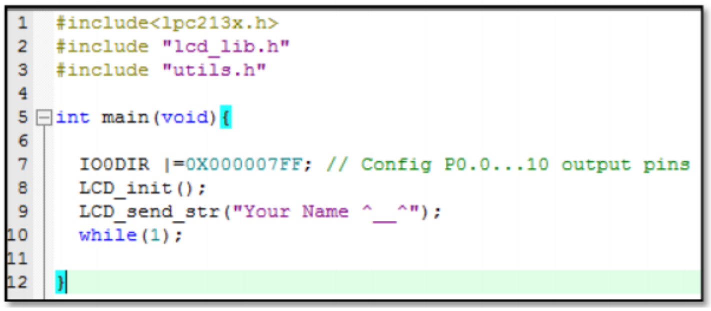
4. Bit 31 – DONE: When ADC conversion completes this bit is 1. When this register (AD0GDR) is read and AD0CR is written, this bit gets cleared i.e., set to 0. If AD0CR is written while a conversion is in progress then this bit is set and a new conversion is started.

5. Other bits are reserved. [1]

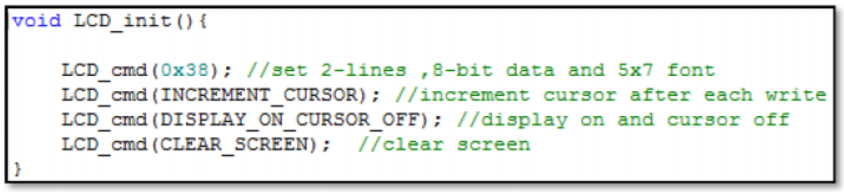
3. Procedure

3.1. Displaying Name Using LCD

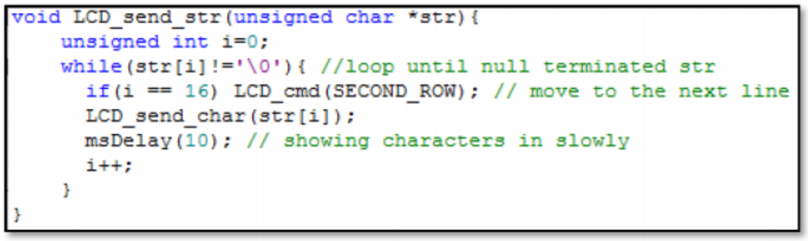
1) The following code was edited by the instructor and was built using Keil Uvision program.



```c
#include<lpc213x.h>
#include "lcd_lib.h"
#include "utils.h"

int main(void){

  IO0DIR |=0X000007FF; // Config P0.0...10 output pins
  LCD_init();
  LCD_send_str("Your Name ^__^");
  while(1);

}
```



```c
void LCD_init(){

    LCD_cmd(0x38); //set 2-lines ,8-bit data and 5x7 font
    LCD_cmd(INCREMENT_CURSOR); //increment cursor after each write
    LCD_cmd(DISPLAY_ON_CURSOR_OFF); //display on and cursor off
    LCD_cmd(CLEAR_SCREEN);  //clear screen
}
```



```c
void LCD_send_str(unsigned char *str){
    unsigned int i=0;
    while(str[i]!='\0'){ //loop until null terminated str
      if(i == 16) LCD_cmd(SECOND_ROW); // move to the next line
      LCD_send_char(str[i]);
      msDelay(10); // showing characters in slowly
      i++;
    }
}
```

Figure 3.1: Name Displaying Code.

2) The name was put in the LCD_send_str function.

3) Hex file was generated after building the code.

4) The following circuit was implemented using Proteus program.

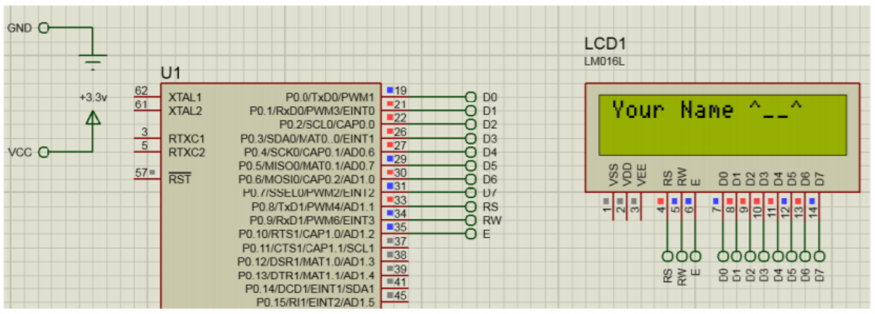

GND
VCC
+3.3v
U1
62 XTAL1
61 XTAL2
3 RTXC1
5 RTXC2
57 RST
P0.0/TxD0/PWM1 19
P0.1/RxD0/PWM3/EINT0 21
P0.2/SCL0/CAP0.0 22
P0.3/SDA0/MAT0..0/EINT1 26
P0.4/SCK0/CAP0.1/AD0.6 27
P0.5/MISO0/MAT0.1/AD0.7 29
P0.6/MOSI0/CAP0.2/AD1.0 30
P0.7/SSEL0/PWM2/EINT2 31
P0.8/TxD1/PWM4/AD1.1 33
P0.9/RxD1/PWM6/EINT3 34
P0.10/RTS1/CAP1.0/AD1.2 35
P0.11/CTS1/CAP1.1/SCL1 37
P0.12/DSR1/MAT1.0/AD1.3 38
P0.13/DTR1/MAT1.1/AD1.4 39
P0.14/DCD1/EINT1/SDA1 41
P0.15/RI1/EINT2/AD1.5 45
D0 D1 D2 D3 D4 D5 D6 D7 RS RW E
LCD1
LM016L
Your Name ^__^
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10 11 12 13 14
RS RW E D0 D1 D2 D3 D4 D5 D6 D7

Figure 3.2: Displaying Name Circuit.

5) The Hex file was uploaded to the microcontroller, then the simulation was started as shown in Figure 3.3.

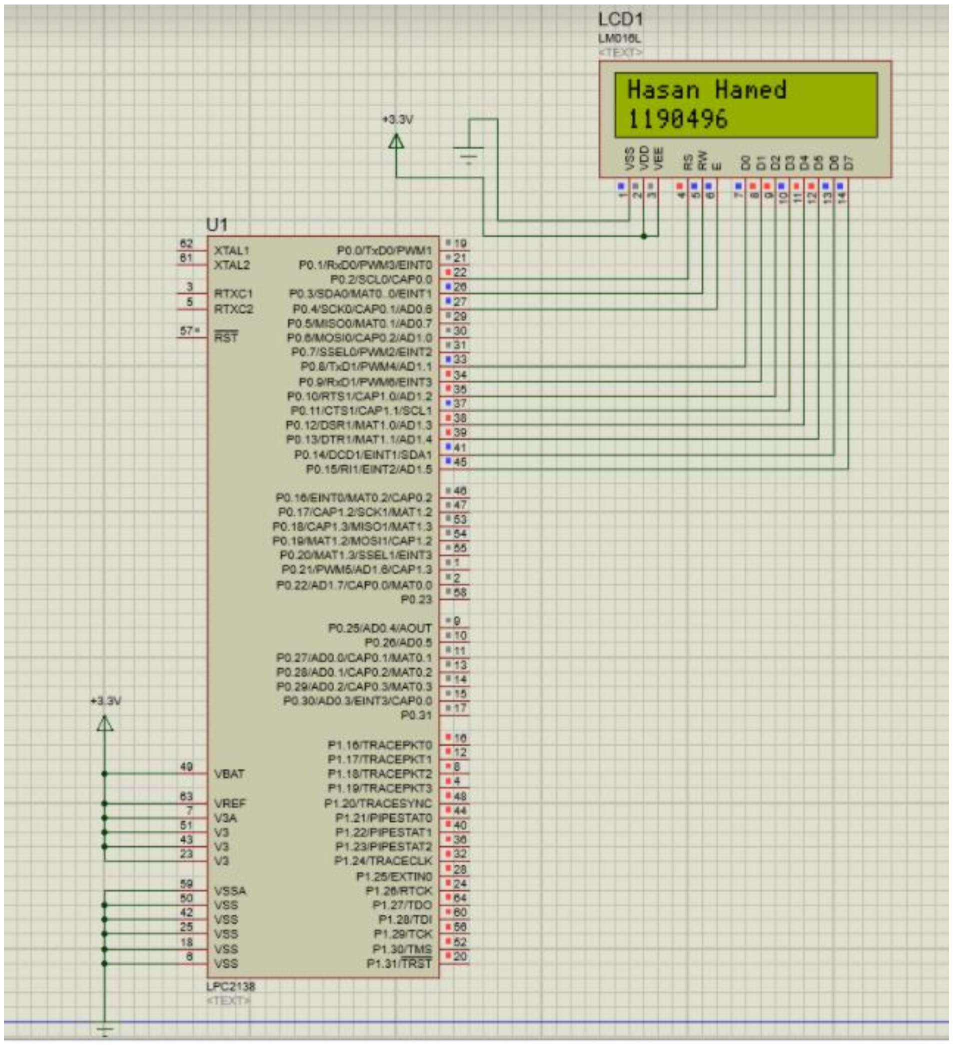
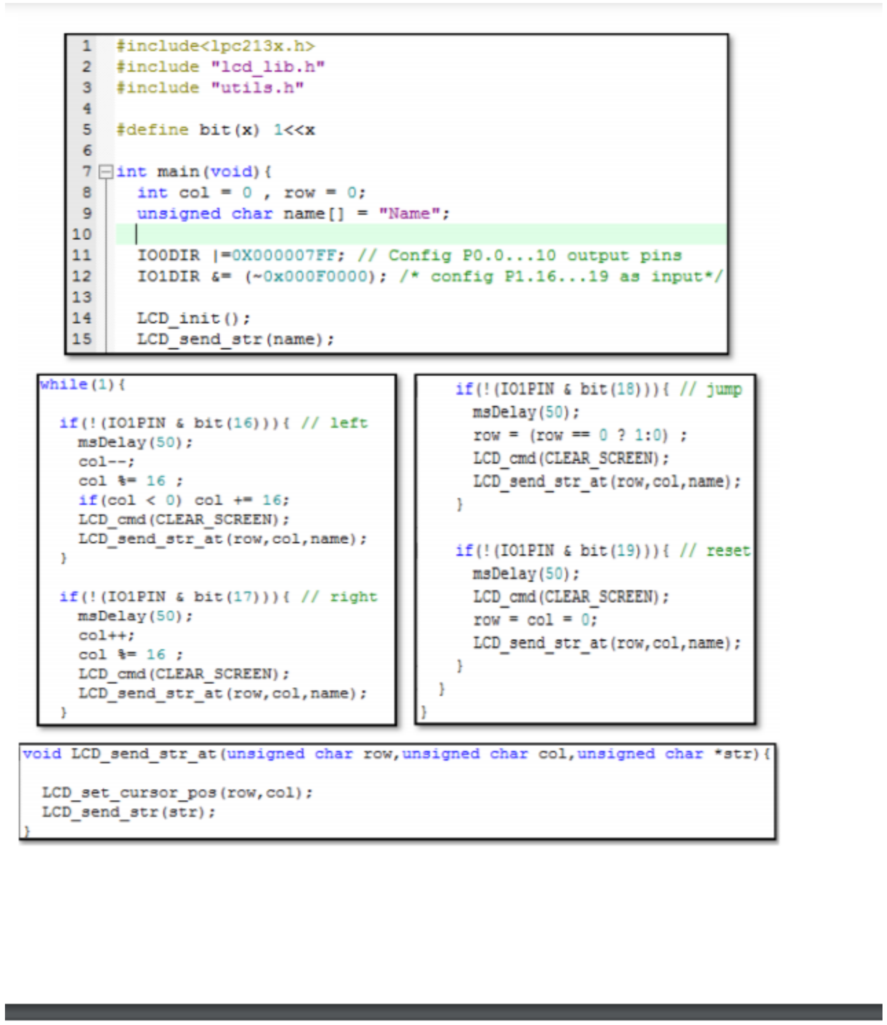


Figure 3.3: Simulation Result.

The circuit will just print the name that was put on the code earlier on the LCD.

3.2. Moving Displayed Word on LCD

1) The following code was edited by the instructor and was built using Keil Uvision.

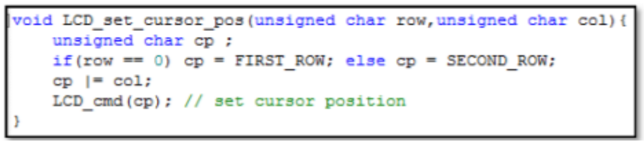


```
#include<lpc213x.h>
#include "lcd_lib.h"
#include "utils.h"

#define bit(x) 1<<x

int main(void){
  int col = 0 , row = 0;
  unsigned char name[] = "Name";

  IO0DIR |=0X000007FF; // Config P0.0...10 output pins
  IO1DIR &= (~0x000F0000); /* config P1.16...19 as input*/

  LCD_init();
  LCD_send_str(name);
```

```
while(1){

  if(!(IO1PIN & bit(16))){ // left
    msDelay(50);
    col--;
    col %= 16 ;
    if(col < 0) col += 16;
    LCD_cmd(CLEAR_SCREEN);
    LCD_send_str_at(row,col,name);
  }

  if(!(IO1PIN & bit(17))){ // right
    msDelay(50);
    col++;
    col %= 16 ;
    LCD_cmd(CLEAR_SCREEN);
    LCD_send_str_at(row,col,name);
  }
```

```
  if(!(IO1PIN & bit(18))){ // jump
    msDelay(50);
    row = (row == 0 ? 1:0) ;
    LCD_cmd(CLEAR_SCREEN);
    LCD_send_str_at(row,col,name);
  }

  if(!(IO1PIN & bit(19))){ // reset
    msDelay(50);
    LCD_cmd(CLEAR_SCREEN);
    row = col = 0;
    LCD_send_str_at(row,col,name);
  }
 }
}
```

```
void LCD_send_str_at(unsigned char row,unsigned char col,unsigned char *str){

  LCD_set_cursor_pos(row,col);
  LCD_send_str(str);
}
```



```
void LCD_set_cursor_pos(unsigned char row,unsigned char col){
    unsigned char cp ;
    if(row == 0) cp = FIRST_ROW; else cp = SECOND_ROW;
    cp |= col;
    LCD_cmd(cp); // set cursor position
}
```

Figure 3.4: Moving Code.

2) The name was put in the LCD_send_str function.

3) Hex file was generated after building the code.

4) The following circuit was implemented using Proteus program.

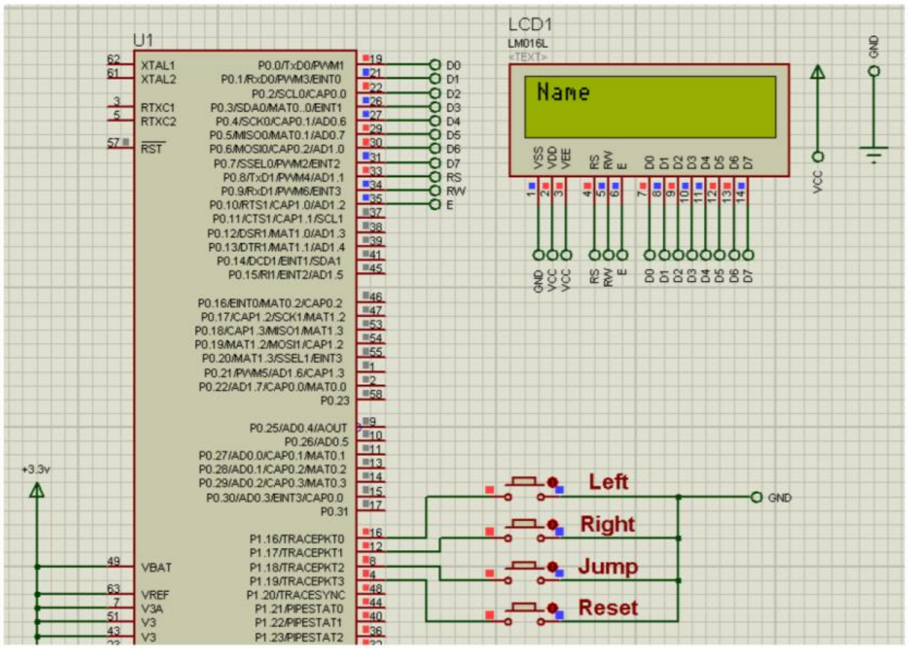


Figure 3.5: Implemented Circuit Using Proteus

5) The Hex file was uploaded to the microcontroller, then the simulation was started as shown in the following Figures.

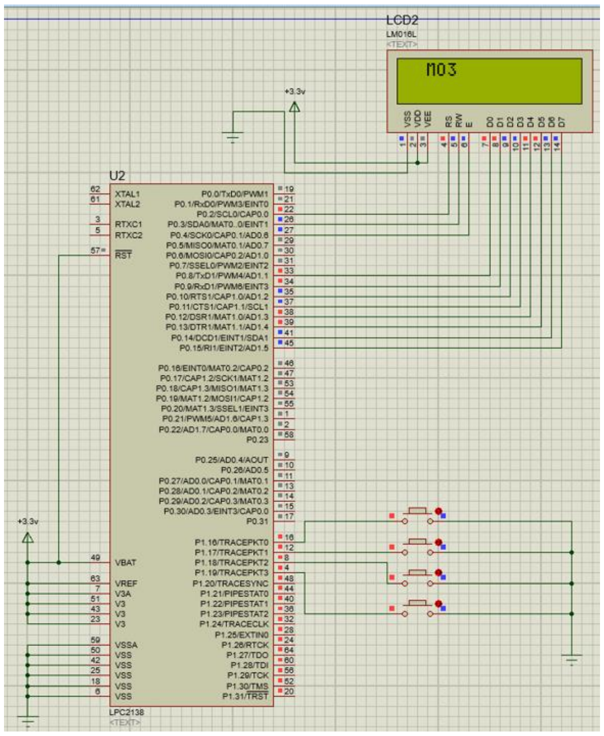


Figure 3.6: Simulation Result for Right Button.

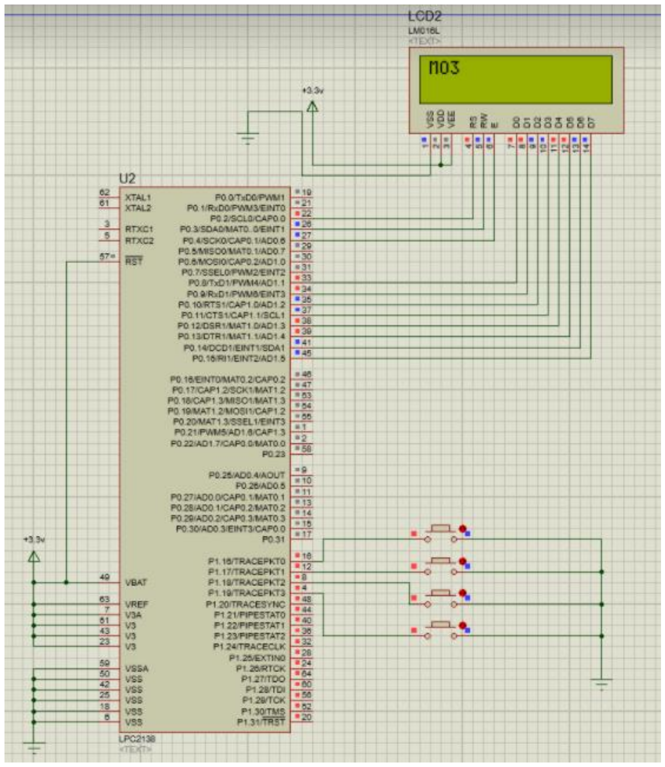


Figure 3.7: Simulation Result for Left Button.

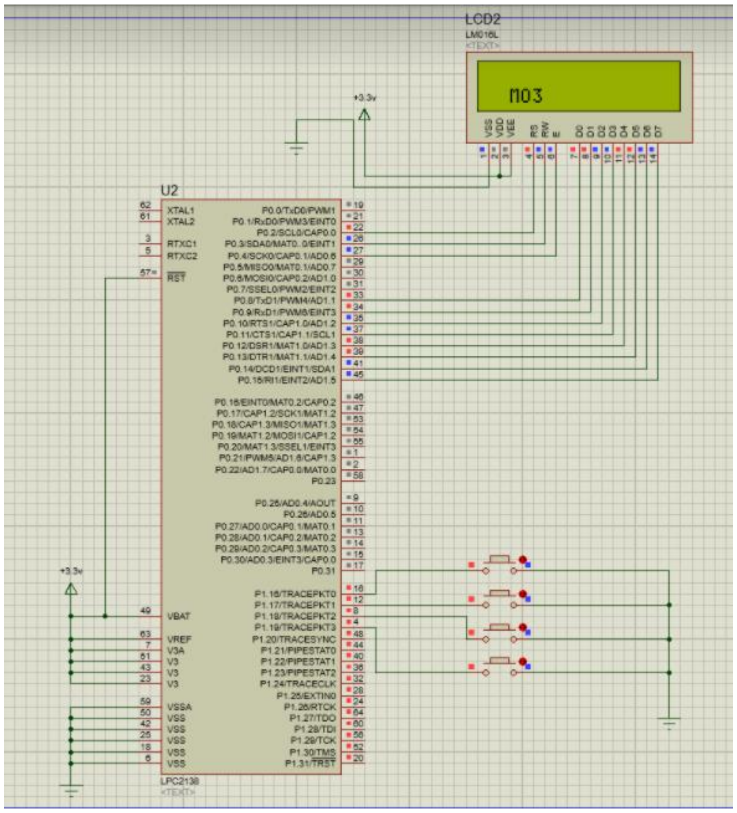


Figure 3.8: Simulation Result for Jump Button.

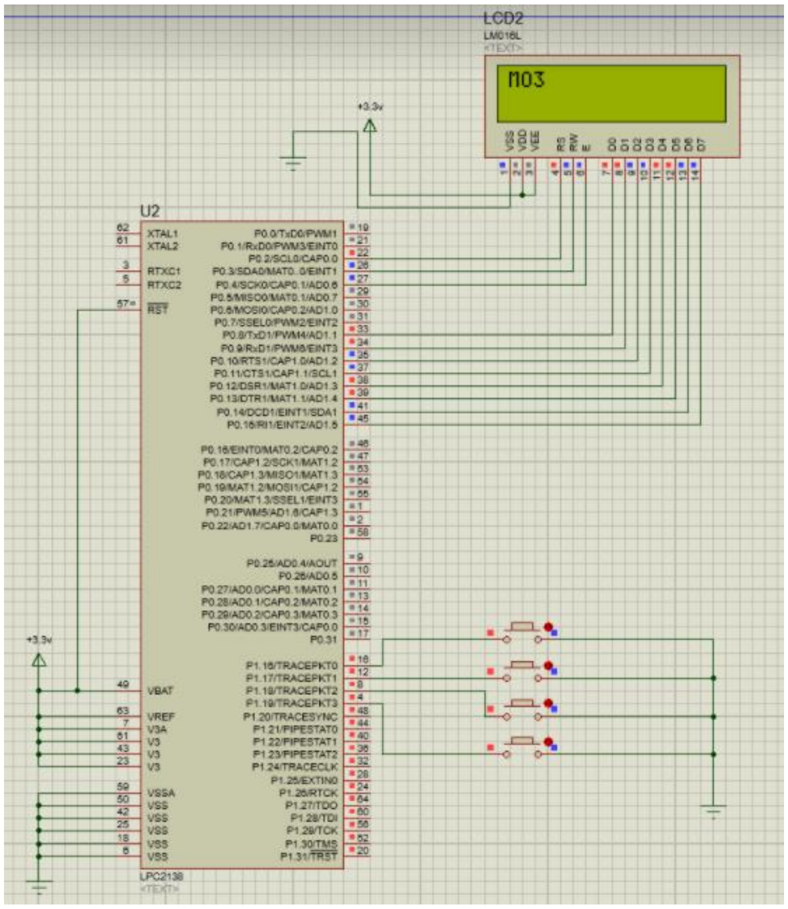
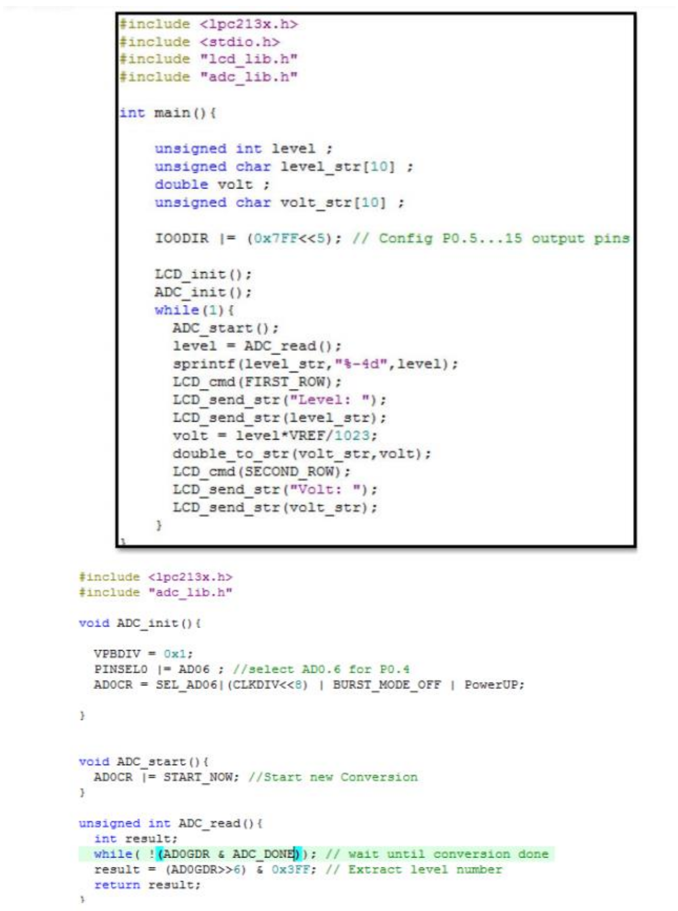


Figure 3.9: Simulation Result for Reset Button.

The LCD will move the word to the right after pushing the right push button as shown in Figure 3.6, and then after pushing the left button the word will move left as shown in Figure 3.7, and the word will jump to the next line after pushing the jump button as shown in Figure 3.8, and finally the word will reset to the first state after pushing the reset button as shown in Figure 3.9.

3.3. Digital Voltmeter

1) The following code was edited by the instructor and was built using Keil Uvision program.



```c
#include <lpc213x.h>
#include <stdio.h>
#include "lcd_lib.h"
#include "adc_lib.h"

int main(){

    unsigned int level ;
    unsigned char level_str[10] ;
    double volt ;
    unsigned char volt_str[10] ;

    IO0DIR |= (0x7FF<<5); // Config P0.5...15 output pins

    LCD_init();
    ADC_init();
    while(1){
      ADC_start();
      level = ADC_read();
      sprintf(level_str,"%-4d",level);
      LCD_cmd(FIRST_ROW);
      LCD_send_str("Level: ");
      LCD_send_str(level_str);
      volt = level*VREF/1023;
      double_to_str(volt_str,volt);
      LCD_cmd(SECOND_ROW);
      LCD_send_str("Volt: ");
      LCD_send_str(volt_str);
    }
}
```

```c
#include <lpc213x.h>
#include "adc_lib.h"

void ADC_init(){

  VPBDIV = 0x1;
  PINSEL0 |= AD06 ; //select AD0.6 for P0.4
  AD0CR = SEL_AD06|(CLKDIV<<8) | BURST_MODE_OFF | PowerUP;

}


void ADC_start(){
  AD0CR |= START_NOW; //Start new Conversion
}

unsigned int ADC_read(){
  int result;
  while( !(AD0GDR & ADC_DONE)); // wait until conversion done
  result = (AD0GDR>>6) & 0x3FF; // Extract level number
  return result;
}
```

Figure 3.10: Voltmeter Code.

2) Hex file was generated after building the code.

3) The following circuit was implemented using Proteus program.

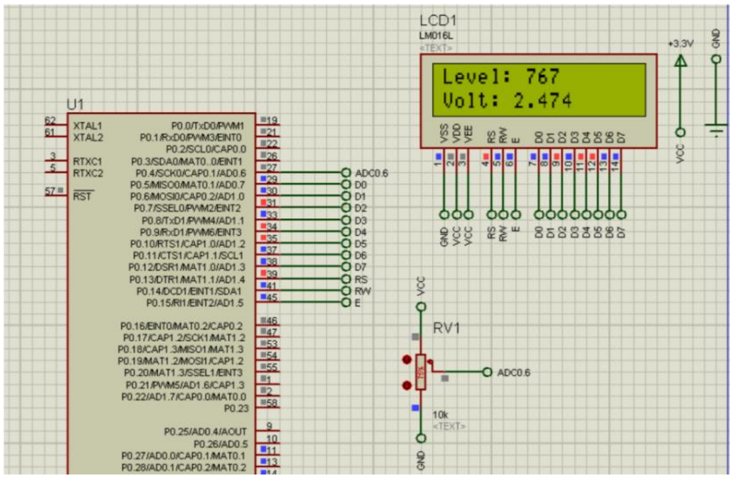


Figure 3.11: Voltmeter Circuit Using Proteus.

4) The Hex file was uploaded to the microcontroller, then the simulation was started as shown in the following Figure.

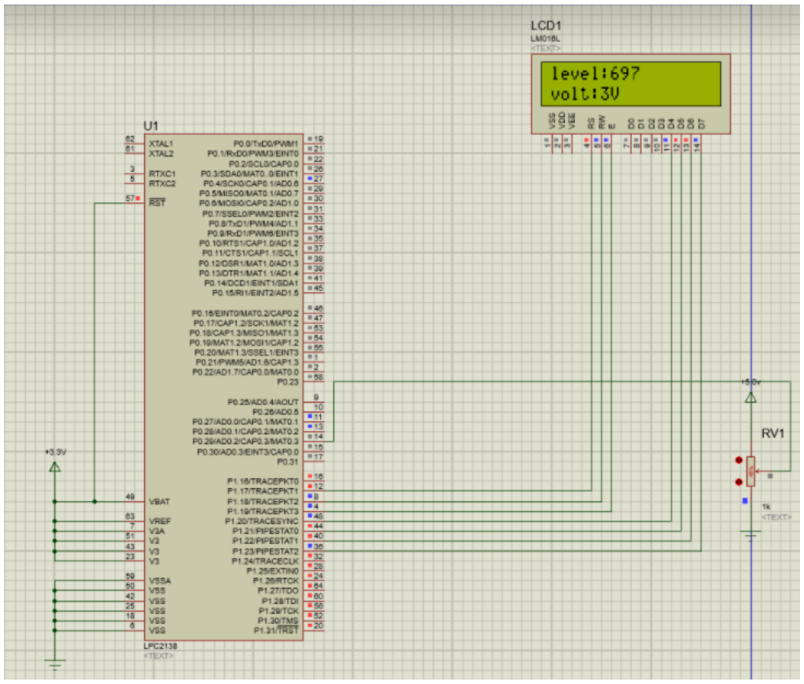
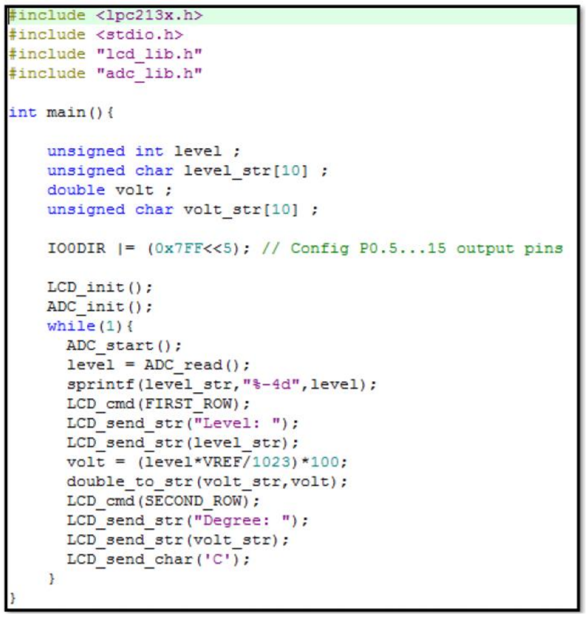


Figure 3.12: Voltmeter Simulation.

The circuit will just find the voltage of the variable resistor using ADC, and then it will display it on the LCD.

3.4. Temperature Sensor

1) The following code was edited by the instructor and was built using Keil Uvision.



```c
#include <lpc213x.h>
#include <stdio.h>
#include "lcd_lib.h"
#include "adc_lib.h"

int main(){

    unsigned int level ;
    unsigned char level_str[10] ;
    double volt ;
    unsigned char volt_str[10] ;

    IO0DIR |= (0x7FF<<5); // Config P0.5...15 output pins

    LCD_init();
    ADC_init();
    while(1){
      ADC_start();
      level = ADC_read();
      sprintf(level_str,"%-4d",level);
      LCD_cmd(FIRST_ROW);
      LCD_send_str("Level: ");
      LCD_send_str(level_str);
      volt = (level*VREF/1023)*100;
      double_to_str(volt_str,volt);
      LCD_cmd(SECOND_ROW);
      LCD_send_str("Degree: ");
      LCD_send_str(volt_str);
      LCD_send_char('C');
    }
}
```

Figure 3.13: Sensor Code.

2) Hex file was generated after building the code.

3) The following circuit was implemented using Proteus program.

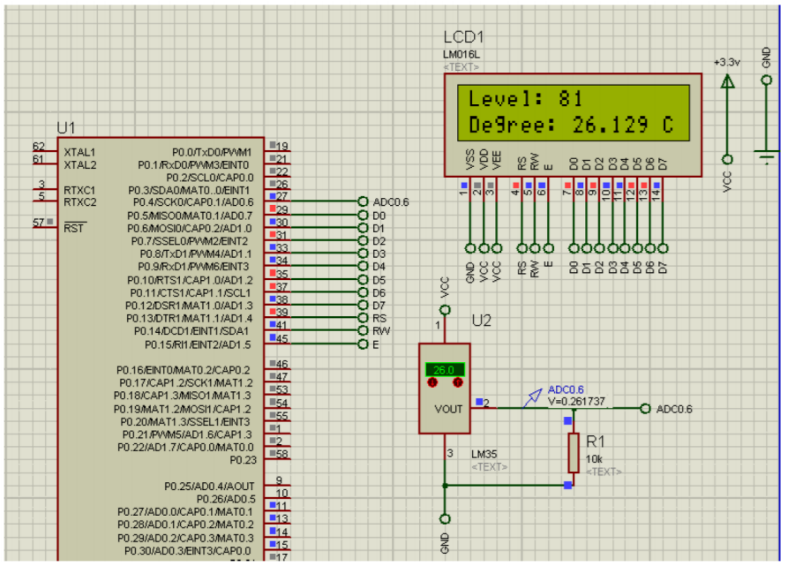


Figure 3.14: Temperature Sensor Circuit.

4) The Hex file was uploaded to the microcontroller, then the simulation was started as shown in the following Figure.

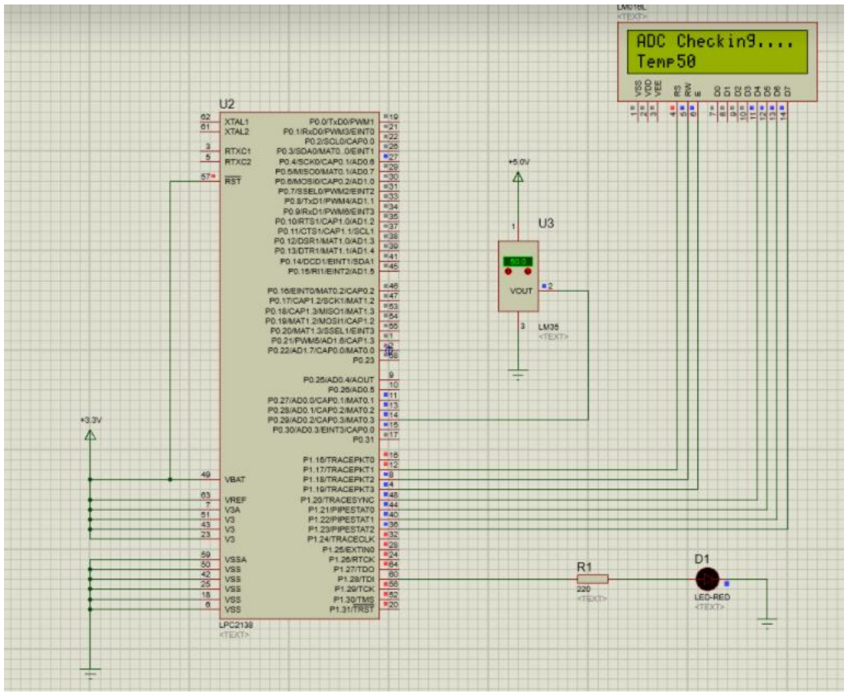


Figure 3.15: Temperature Sensor Simulation

The sensor will sense the temperature and using ADC, it will display it to the LCD.

4. Conclusion

In conclusion, we understood how LCD works and how to use it to display words, and we learned how to make operations to this word such as moving right or left or jump from line to line, and we learn how to use ADC to convert the analog signal into digital values so that the digital system can deal with, and finally we learned how to use both for real-life operations, such as voltmeters and sensors.

5. References

[1] Computer Design Laboratory Manual.