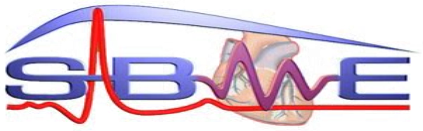


Embedded Final Project
SBE403

Delivered to:

Eng/ Hossam Ahmed
Eng/ Alaa Melek
Eng/ Laila Abbas

Submitted by:

Name	Sec	BN
Ibrahim Elsayed	1	1
Bassam Mostafa	1	22
Donia Abd Elslam	1	30
Renad Taher	1	34
Mariem Ahmed	2	29
Mustafa Yehia	2	33

Contents

LM016 Character LCD 3
- **LCD** 3
- **LCD (4-bit Mode)** 3
- **Interface of LCD with ATmega32** 3

TC72 SPI to Temperature Convertor 4
- **SPI Communication** 4
- **ATmega32 SPI Communication** 4
  - **Pin Configurations** 4
  - **SPCR: SPI Control Register** 4
  - **SPSR: SPI Status Register** 5
  - **SPDR: SPI Data Register** 5
- **Programming For TC72** 5

(4 * 3) Keypad 6
- **How to work** 6
- **Operation** 6

Analog to digital converter (ADC) 7
- **ADC Register** 7
- **ADMUX Register** 7
- **ADCSRA Register** 7

PWM to Voltage Convertor Module 9

In-Code Files (Layered Architecture) 10

Full Schematic (Proteus) 10

UML 10

Git-hub Repo 11

Link of Video 11

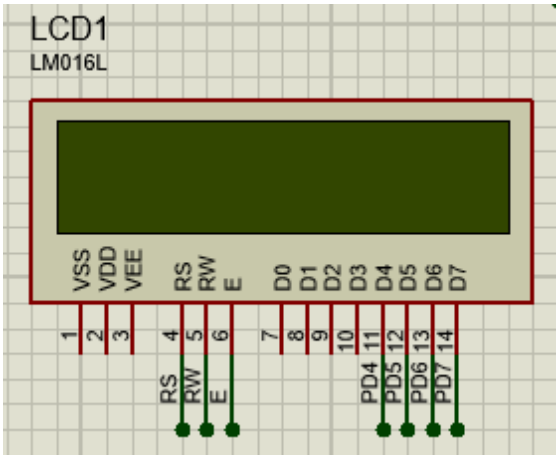
LM016 Character LCD

LCD

LCDs (Liquid Crystal Displays) are used for displaying status or parameters in embedded systems. LCD (16 * 2) is a 16-pin device which has 8 data pins (D0 - D7) and 3 control pins (RS, RW, EN). The remaining 5 pins are for supply and backlight for the LCD.

The control pins help us configure the LCD in command mode or data mode. They also help configure read mode or write mode and when to read or write.

LCD 16x2 can be used in 4-bit mode or 8-bit mode depending on the requirement of the application. To use it, we need to send certain commands to the LCD in command mode and once the LCD is configured according to our need, we can send the required data in data mode.



LCD (4-bit Mode)

- In 4-bit mode, data/command is sent in a 4-bit (nibble) format.
- To do this 1st send a Higher 4-bit and then send a lower 4-bit of data/command.
- Only 4 data (D4 - D7) pins of (16 * 2) of LCD are connected to the microcontroller and other control pins RS (Register Select), RW (Read/Write), E (Enable) is connected to other GPIO Pins of the controller.

Interface of LCD with ATmega32

Initialization

1. Wait for 15 ms, Power-on initialization time for LCD $(16 * 2)$
2. Send 0x02 command which initializes LCD $(16 * 2)$ in 4-bit mode.
3. Send 0x28 command which configures LCD in 2-line, 4-bit mode, and $(5 * 8)$ dots.
4. Send any Display ON command (0x0e, 0x0c)
5. Send 0x06 command (Increment Cursor)

Command Write Function

1. First, send a higher nibble of command.
2. Make RS pin low, RS=0 (command reg.)
3. Make RW pin low, RW=0 (write operation) or connect it to ground.
4. Give High to Low pulse at Enable (E).
5. Send lower nibble of command.
6. Give High to Low pulse at Enable (E).

Data Write Function

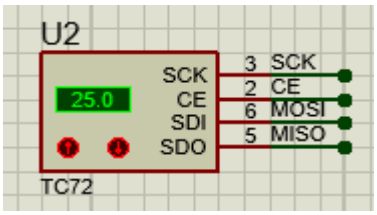
1. First, send a higher nibble of data.
2. Make RS pin high, RS=1 (data reg.)
3. Make RW pin low, RW=0 (write operation) or connect it to ground.
4. Give High to Low pulse at Enable (E).
5. Send lower nibble of data.
6. Give High to Low pulse at Enable (E).

TC72 SPI to Temperature Convertor

SPI Communication

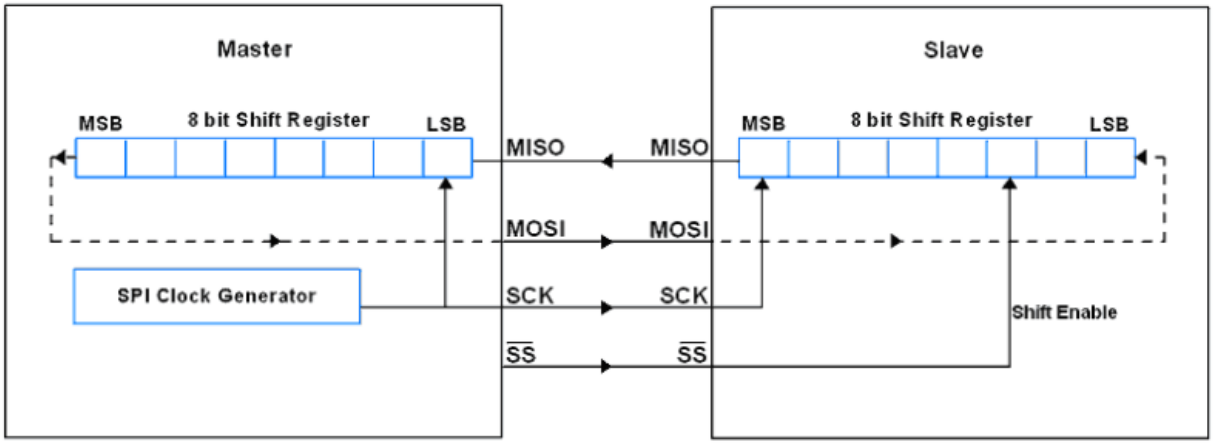
The Serial Peripheral Interface (SPI) is a bus interface connection protocol originally started by Motorola Corp. It uses four pins for communication.

- SDI (Serial Data Input)
- SDO (Serial Data Output)
- SCLK (Serial Clock)
- CS (Chip Select)



ATmega32 SPI Communication

MISO (Master-In-Slave-Out)	MOSI (Master-Out-Slave-In)
The Master receives data, and the slave transmits data.	The master transmits data, and the slave receives data.
SCK (Shift Clock)	**SS (Slave Select)**
The Master generates this clock for the communication, which is used by the slave.	Master can select slaves through this pin.



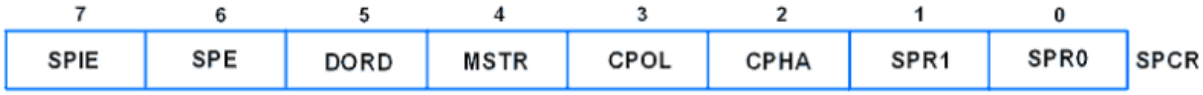
SPI Master Slave Interconnection

Pin Configurations

SPI Pins	Pin on ATmega32	Pin Direction (Master)	Pin Direction (Slave)
MISO	B6	Input	Output
MOSI	B5	Output	Input
SCK	B7	Output	Input
SS	B4	Output	Input

AVR ATMega32 uses three registers to configure SPI communication that are SPI Control Register, SPI status Register and SPI Data Register.

SPCR: SPI Control Register



Illustrartion:

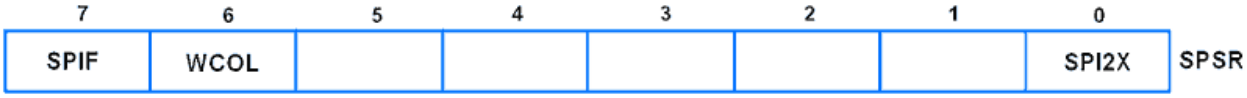
Bit-No. 7	Bit-No. 6	Bit-No. 5
(SPIE: SPI interrupt Enable bit)	(SPE: SPI Enable bit)	(DORD: Data Order bit)
Bit-No. 4	**Bit-No. 3**	**Bit-No. 2**
(MSTR: Master/Slave Select bit)	(CPOL: Clock Polarity Select bit)	(CPHA: Clock Phase Select bit)
Bit-No. 1 & Bit-No. 0		
(SPR1 & SPR0: SPI Clock Rate Select bits)		

Configuration:

Bit-No. 7	Bit-No. 6	Bit-No. 5

1: Enable SPI Interrupt 0: Disable SPI Interrupt	1: Enable SPI 0: Disable SPI	1: LSB Transmit First 0: MSB Transmit First
Bit-No. 4	**Bit-No. 3**	**Bit-No. 2**
1: Master Mode 0: Slave Mode	1: Logic One Clock 0: Logic Zero Clock	1: Data Sample on Training Clock Edge 0: Data Sample on Leading Clock Edge

SPSR: SPI Status Register



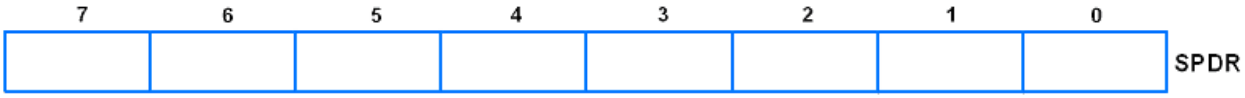
Bit 7 – SPIF: SPI interrupt flag bit
- This flag gets set when the serial transfer is complete.
- Also gets set when the SS pin is driven low in master mode.
- It can generate an interrupt when SPIE bit in SPCR and a global interrupt is enabled.

Bit 6 – WCOL: Write Collision Flag bit
- This bit gets set when SPI data register writes occur during previous data transfer.

Bit 5:1 – Reserved Bits

Bit 0 – SPI2X: Double SPI Speed bit
- When set, SPI speed (SCK Frequency) gets doubled.

SPDR: SPI Data Register



- SPI Data register used to transfer data between the Register file and SPI Shift Register.
- Writing to the SPDR initiates data transmission.

Programming For TC72

The overall programming interface lists below:

1. Set up the SPI to master mode.
2. Select SPI clock and data sampling mode.
3. Set up digital output for display.
4. Send the command to TC72.
5. Read temperature from TC72.
6. Display the Result.

(4 * 3) Keypad

The keypad is used as an input device to read the key pressed by the user and to process it.
(4 ∗ 3) keypad consists of 4 rows and 3 columns. Switches are placed between the rows and columns. A keypress establishes a connection between the corresponding row and column between which the switch is placed.
To read the keypress, we need to configure the rows as outputs and columns as inputs.
Columns are read after applying signals to the rows to determine whether a key is pressed and if pressed, which key is pressed.

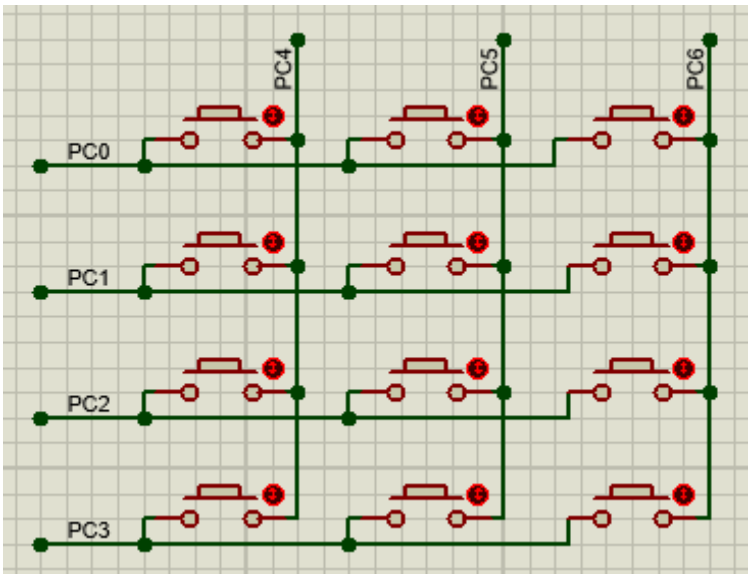
How to work

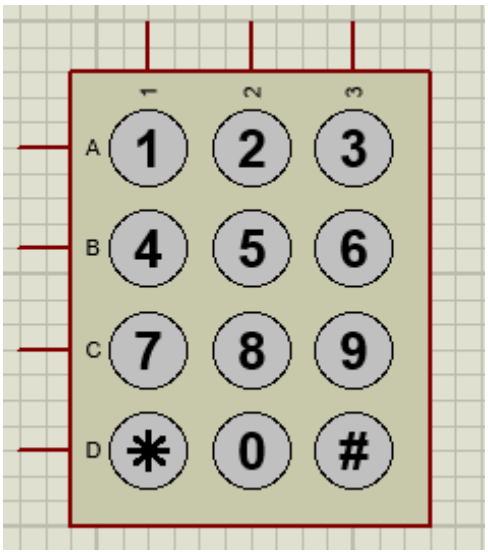
For identification of button pressed, we are going to use cross reference method. Here first we are going to either connect all columns or all rows to Vcc (High), so if rows are connected to common Vcc (High), we are going to take the columns as inputs to controller.

Operation

- In our circuit, PORTC of ATMEGA32 is connected to Keypad to get the input data (Sat-Temperature) and sat the state of 'Operation' that indicates to the heater and sensor are working.
- We put the 'Sat-Temperature' in the Stand-by stage, then we pressed '#' to start the 'Operation' stage.
- In Operation stage, Check-up what is the next stage if it is Normal or Error stage.

Note: We represent Keypad Phonebook on Proteus by a (4 ∗ 3) push buttons for more performance





Analog to digital converter (ADC)

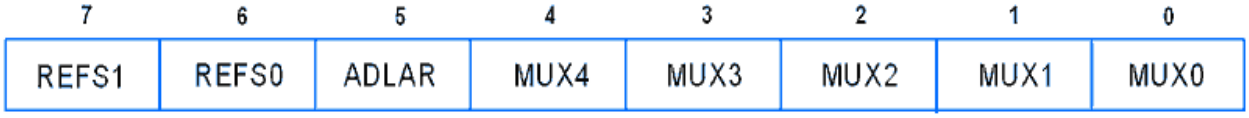
ADC Register

In AVR ADC, we need to understand four main registers:

1. ADCH: Holds digital converted data higher byte.
2. ADCL: Holds digital Converted data lower byte.
3. ADMUX: ADC Multiplexer selection register.
4. ADCSRA: ADC Control and status register.

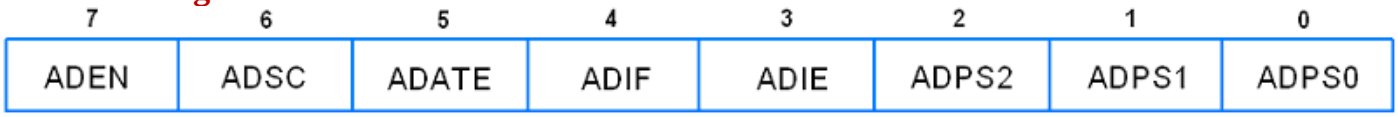
ADCH: two-register holds the digital converted data, which is 10-bit.

ADMUX Register



Bit 7:6 (REFS1)	Bit 5 (ADLAR)	Bit 4:0 (MUX4)
Reference voltage selection for ADC	Use 10-bits output as upper bits or lower bits in ADCH & ADCL.	We can select input channel ADC0 to ADC7 by using these bits. These bits are also used to select comparator (inbuilt in AVR) inputs with various gain. We will cover these comparator operations in another part.

ADCSRA Register



Bit 7 (ADEN) ADC Enable	Bit 6 (ADSC) ADC Start Conversion	Bit 5 (ADATE) ADC Auto Trigger Enable
Writing one to this bit enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.	Writing one to this bit starts the conversion.	Writing one to this bit, results in Auto Triggering of the ADC is enabled.
Bit 4 (ADIF) ADC Interrupt Flag	**Bit 3 (ADIE) ADC Interrupt Enable**	**Bits 2:0 (ADPS2:0) ADC Prescaler Select Bits**
This bit is set when an ADC conversion completes, and the Data Registers are updated.	Writing one to this bit, the ADC Conversion Complete Interrupt is activated.	These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

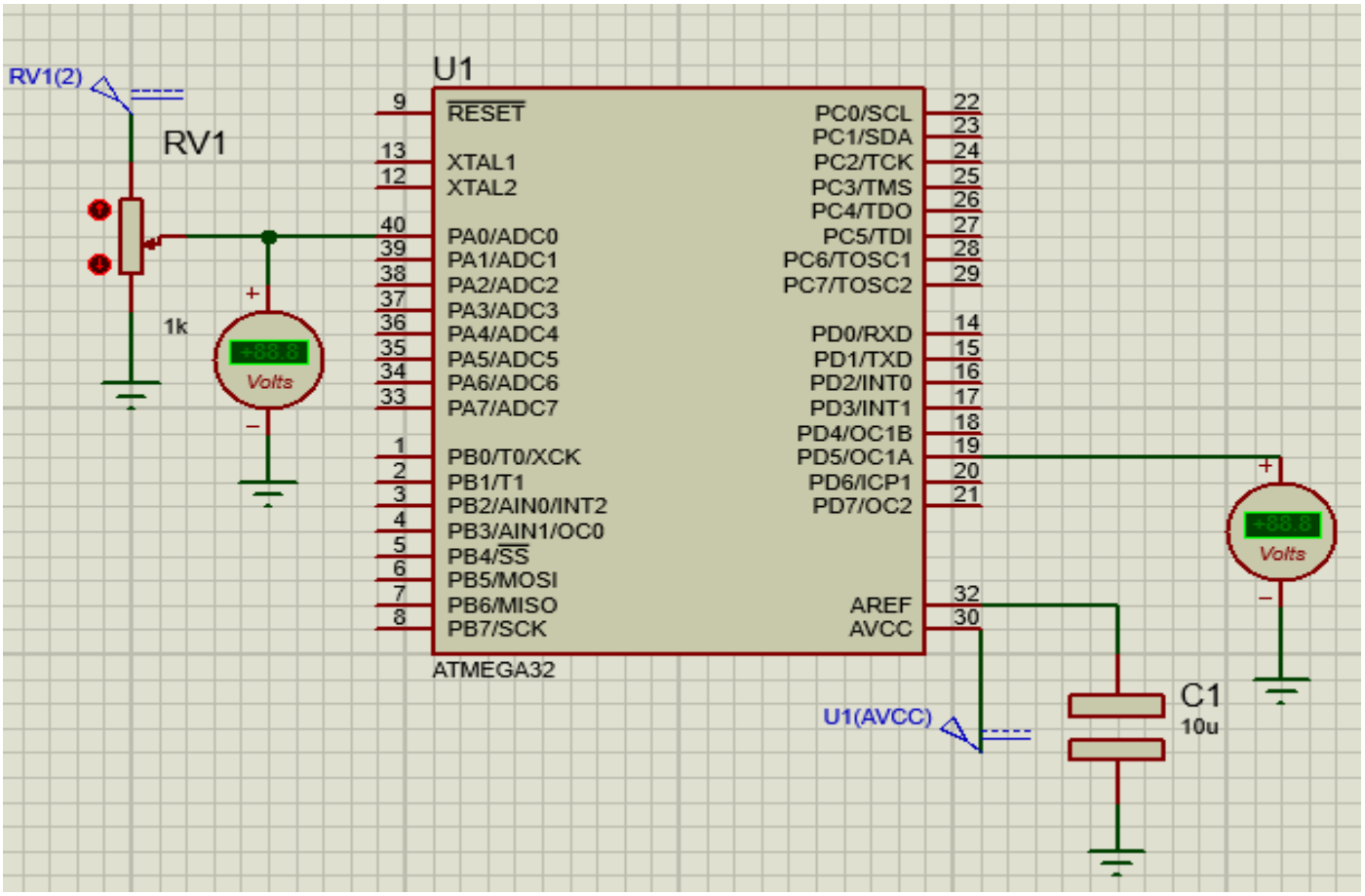
Notes about Bits 2:0:

- We can select any divisor and set frequency $Fosc/2$, $Fosc/4$, etc. for ADC, But in AVR, ADC requires an input clock frequency less than 200KHz for max. accuracy. So, we have to always take care of not exceeding ADC frequency more than 200KHz.

Steps to Program ADC:

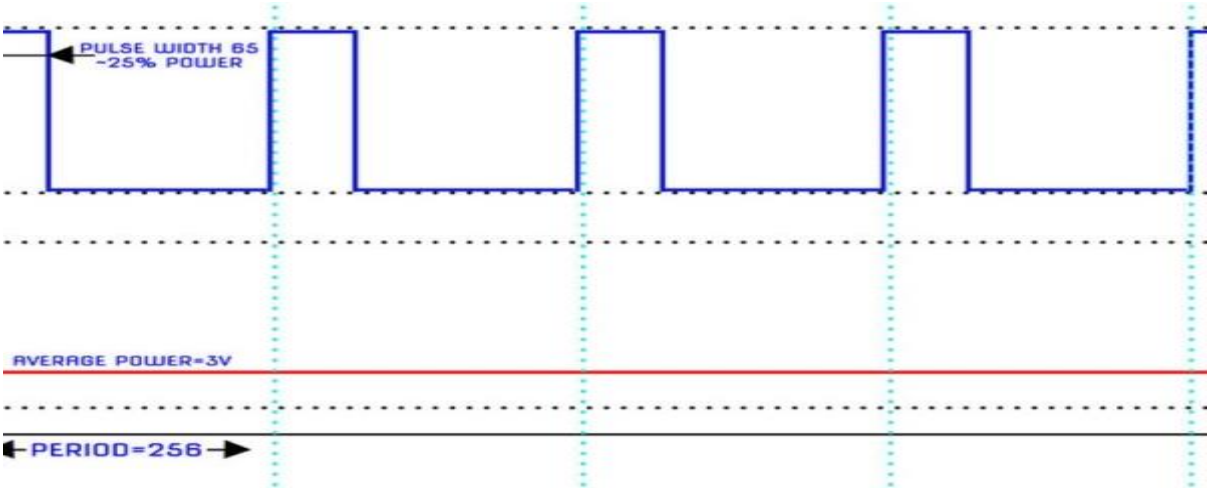
1. Make the ADC channel pin as an input.
2. Set ADC enable bit in ADCSRA, select the conversion speed using ADPS2:0. For example, we will select devisor 128.
3. Select ADC reference voltage using REFS1: REFS0 in ADMUX register, for example, we will use AVcc as a reference voltage.
4. Select the ADC input channel using MUX4:0 in ADMUX, for example, we will use channel 0.
5. So, our value in register $ADCSRA = 0x87$ and $ADMUX = 0x40$.

6. Start conversion by setting bit ADSC in ADCSRA. ($e.g. ADCSRA \mid = (1 << ADSC);$)
7. Wait for conversion to complete by polling ADIF bit in ADCSRA register.
8. After the ADIF bit gone high, read ADCL and ADCH register to get digital output.
9. Notice that read ADCL before ADCH; otherwise result will not be valid.



PWM to Voltage Convertor Module

Pulse Width Modulation (PWM) is a technique in power control, which used to control the power fed to control the temperature of the heater. It is a modulation technique, which have the width of the carrier pulse is varied in accordance with the analog message signal.



Pulse Width Modulation (PWM) is a power switching technique, which designed for providing intermediate amount of electrical power between fully on and fully off levels. Usually, digital pulses have same on and off time period, but in some situations, we need the digital pulse to have more/less on time/ofttimes. In PWM technique, we create digital pulses with unequal amount of on and off state to get required intermediate voltage values.

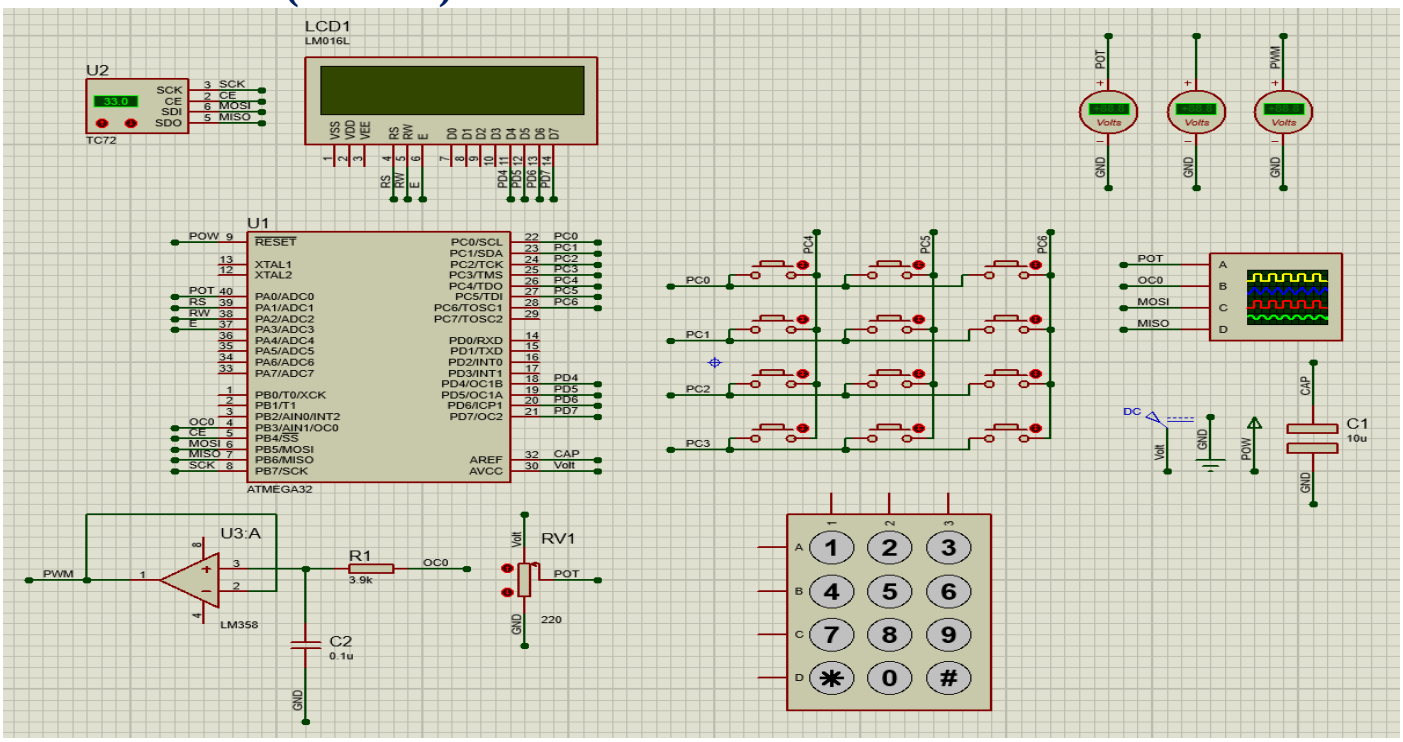
Duty cycle is defined by the percentage of high voltage duration in a complete digital pulse.

$$\%\ of\ Duty\ cycle = \frac{T_{on}}{T_{period}} * 100$$

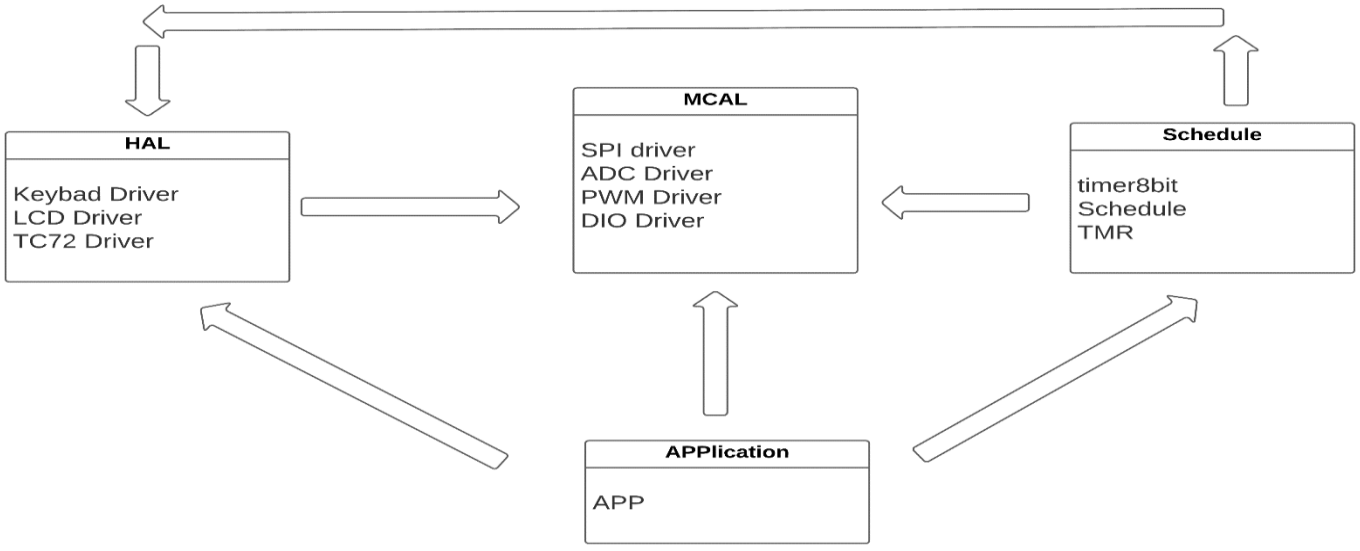
In-Code Files (Layered Architecture)

Standard	MCAL	HAL	App Layer	Schedular
std_macros.h Basic_Types.h	DIO.h DIO.c PWM.h PWM.c ADC.h ADC.c SPI.h SPI.c	LCD_config.h LCD.h LCD.c Keypad.h Keypad.c	app.h app.c	TMR.h Scheduler.h timer8bit.h timer8bit.c

Full Schematic (Proteus)



UML



Git-hub Repo

- Embedded Final Project Repo

Link of Video

- Video Link