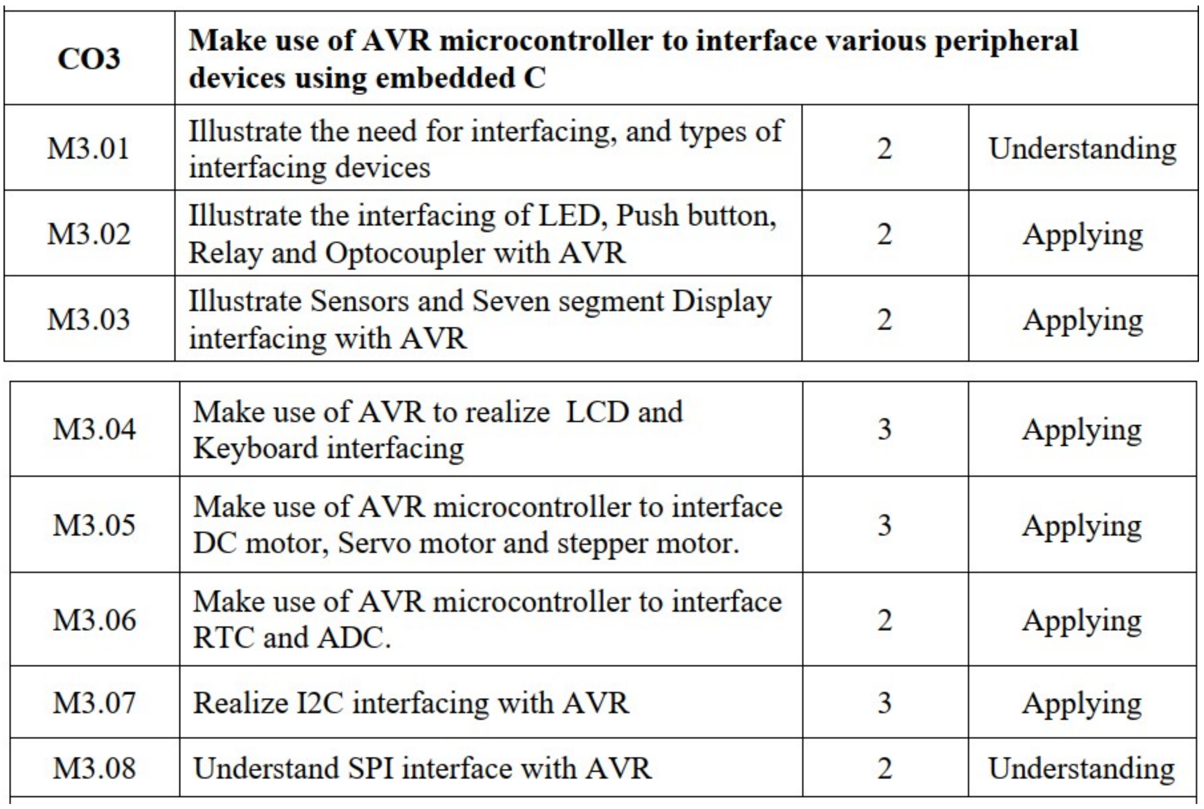
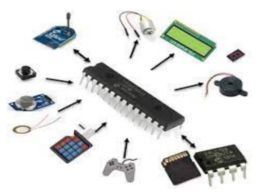
MODULE III
AVR microcontroller to interface



CO3	**Make use of AVR microcontroller to interface various peripheral devices using embedded C**		
M3.01	Illustrate the need for interfacing, and types of interfacing devices	2	Understanding
M3.02	Illustrate the interfacing of LED, Push button, Relay and Optocoupler with AVR	2	Applying
M3.03	Illustrate Sensors and Seven segment Display interfacing with AVR	2	Applying
M3.04	Make use of AVR to realize LCD and Keyboard interfacing	3	Applying
M3.05	Make use of AVR microcontroller to interface DC motor, Servo motor and stepper motor.	3	Applying
M3.06	Make use of AVR microcontroller to interface RTC and ADC.	2	Applying
M3.07	Realize I2C interfacing with AVR	3	Applying
M3.08	Understand SPI interface with AVR	2	Understanding

Syllabus Content: Interfacing of LED, Push button, Relay, Opto Coupler, Sensors (Temperature sensor, IR sensor) Seven segment Display, LCD, Keyboard Interfacing, Motor (DC, Servo, Stepper), RTC Interfacing, ADC interfacing
On-board communication interfaces with AVR – I2C interfacing(like real time clock interfacing)and basics of SPI interfacing with AVR.

I/O Sub-Systems: It facilitates the interaction of embedded system with external world. Eg. LED, Push-button switch, matrix keyboard, sensors, relays, motor, LCD.



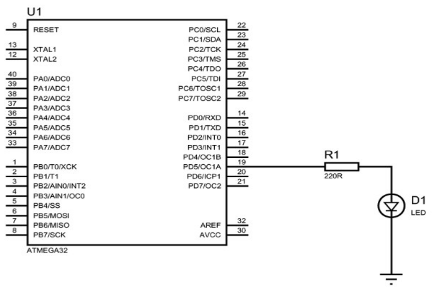
Communication Interface: They are essential for communicating with various subsystems of the embedded system and with the external world. The communication interface can be of two types:

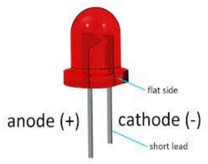
- **Onboard Communication Interface:** The communication interface channel which interconnects the various components within an embedded system. Eg: Serial interfaces like I2C, SPI, UART, 1-Wire etc and Parallel bus interface.
- **External Communication Interface:** It is responsible for data transfer between the embedded system and other devices or modules. The external communication interface can

be either wired media or wireless media and it can be a serial or parallel interface. Eg: Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS etc. (wireless) and RS-232, USB, Parallel port etc.(wired).

Interfacing of LED(Light emitted Diode):

- It is an output device & is used as an indicator for the status of various signals or situations. LED is a PN junction diode which emits light when it is forward biased.
- It has 2 terminals Anode & Cathode. For proper functioning of the LED, the anode of it should be connected to +ve terminal of the supply voltage and cathode to the –ve terminal of supply voltage.
- The LED can be connected to any of the port pin using a current limiting resistor to limit the current through the LED.
- The port pin to which the LED is connected has to be configured as an output port.





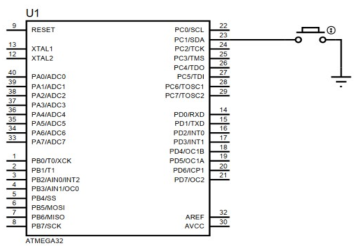
Q) Write a C-program to blink an LED connected to Pin 5 of PORT D with a delay of 1s
SOL:

```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
        {
                DDRD = 0xFF;                    //Port D configured as output port
                while(1)
                {
                        PORTD = 0x20;           //PORTD Pin5 made high
                        _delay_ms(1000);        //call delay
                        PORTD = 0x00;           //PORTD Pin5 made low
                        _delay_ms(1000);        //call delay
                }
        return 0;
 }
```

Interfacing of Push Button Switch:

- A push-button works just like a switch works. It is used to give user input to the microcontroller.
- For the interfacing of push-button with Atmega32 microcontroller, we need to declare a port as an input port. Then continuously monitoring the status of the pin it is connected.
- When a push button is pressed or unpressed, it provides either logic high or logic low output depending on the configuration mode in which the push button circuit is designed.
- There are two configuration modes to interface a push button with ATmega32:

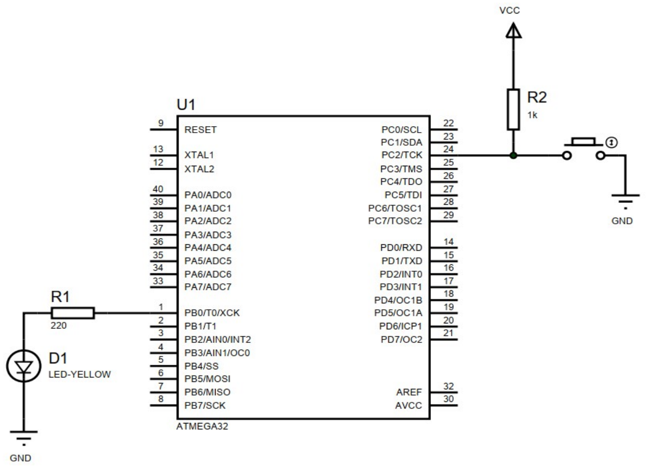
- **Pull-Up Mode**: In Pull-Up mode, when a push button is not pressed, a logic high('1') input appears at Microcontroller input pin.
- **Pull-Down Mode**: In Pull-Down mode, when a push button is not pressed, a logic low('0') input appears at Microcontroller input pin.



Q) Write a C-program to read the status of a push button switch connected to PORT C & display the status by turning ON/OFF an LED connected to PORT B.

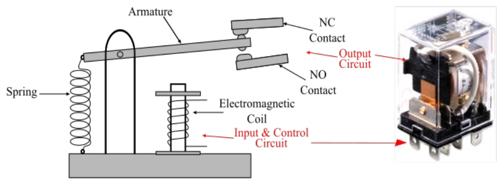
SOL:

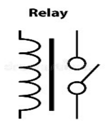
```
#include <avr/io.h>
int main(void)
{
      DDRB = 0xFF;                // PORT B is output
      DDRC = 0x00;                // PORT C is input
      while(1)
            {
                  if (PINC & (1<<2))          // Checking the status of pin PC2
                  {
                        PORTB = 0x00;
                  }
                  else
                  {
                        PORTB = 0xFF;
                  }
            }
        return 0;
}
```



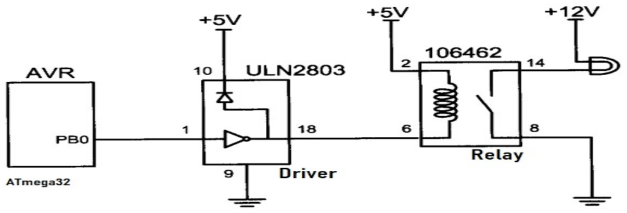
Interfacing of Electromechanical Relays:

- They are electrically operated switches used to isolate circuit with different voltage sources.
- It control a high voltage circuit using a low voltage signal.





- An electromechanical relay transfers signals between its contacts through a mechanical movement. It has three sections- input section, control section & output section.
    - The **input section** consists of input terminals where a small control signal is to be applied.
    - The **control section** has an electromagnetic coil which gets energised when control input signal is applied to the input terminals.
    - The **output section** consists of a movable armature and mechanical contacts – movable and stationary, the movement of the armature makes or breaks the electrical circuit.
    - The contacts can be **Normally Open(NO)** or **Normally Closed(NC)**. In NC type. The contacts are closed when the coil is not energized. In NO type the contact are Open when the coil is energized.
    - When an input control voltage is applied to the electromagnetic coil, it gets magnetised and the armature is attracted by the magnetic field produced by the coil. The movable mechanical contacts are attached to the armature, thus when the armature moves towards the electromagnet, the contacts closes, making the output circuit switched on. When the control signal is removed, the armature comes back to its original position by the force of spring, making output circuit off.
- The port pin to which the relay input control voltage is connected has to be configured as an output port.
- A **driver circuit(ULN2803)** or a **power transistor** is placed between the microcontroller & relay to provider the required driving current.
- The microcontroller will send a logic high signal(5V) to the relay driver circuit to drive a load(like bulb).

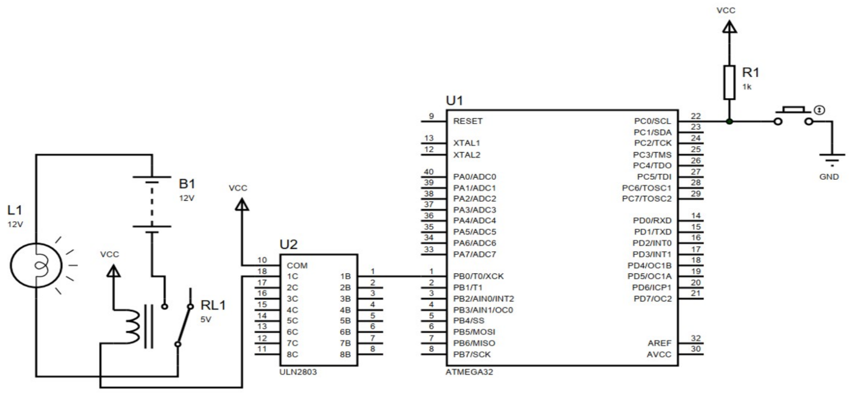


Q) Write a C-program to read the status of a push button switch connected to PORT C & display the status by turning ON/OFF a bulb connected to PORT B using relay.

SOL:

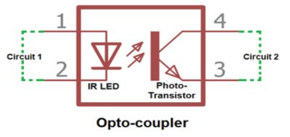
```
#include <avr/io.h>
int main(void)
{
      DDRB = 0xFF;                // PORT B is output
      DDRC = 0x00;                // PORT C is input
      PORTB = 0x00;
      while(1)
```

```
            {
                    if (PINC == 0x01)          // Checking the status of pin PC0
                    {
                          PORTB = 0x00;
                    }
                    else
                    {
                          PORTB = 0xFF;
                    }
             }
      return 0;
}
```

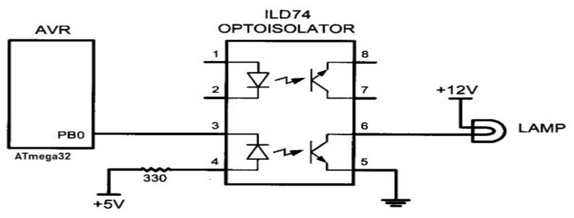


Interfacing of OptoIsolator or Optocoupler:

- It is a semiconductor device that allows an electrical signal to be transmitted between two isolated circuits.
- Two parts are used in an optocoupler: **a LED that emits infrared ligh**t & **a photosensitive device(Photodiode, Phototransistor) that detects light from the LED**. Both parts are contained within a black box with pins for connectivity. The input circuit takes the incoming signal, whether the signal is AC or DC, and uses the signal to turn on the LED**.**



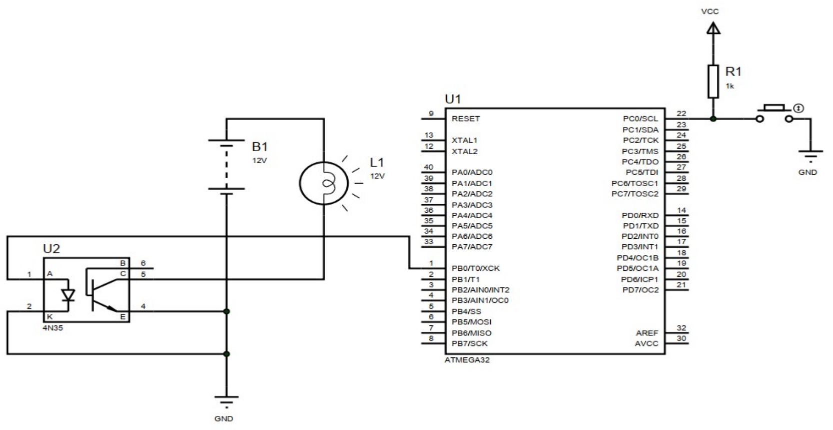
- The port pin to which the optoisolator input control voltage is connected has to be configured as an output port.
- The microcontroller will send a logic high signal(5V) to the optoisolator to drive a load(like bulb, motor).



Q) Write a C-program to read the status of a push button switch connected to PORT C & display the status by turning ON/OFF a bulb connected to PORT B using optoisolator.

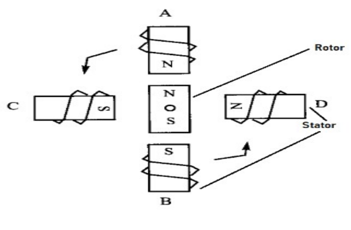
SOL:

```
#include <avr/io.h>
int main(void)
{
       DDRB = 0xFF;                 // PORT B is output
       DDRC = 0x00;                 // PORT C is input
       PORTB = 0x00;
       while(1)
              {
                     if (PINC & (1<<0))          // Checking the status of pin PC0
                     {
                            PORTB = 0x00;
                     }
                     else
                     {
                            PORTB = 0xFF;
                     }
              }
         return 0;
}
```

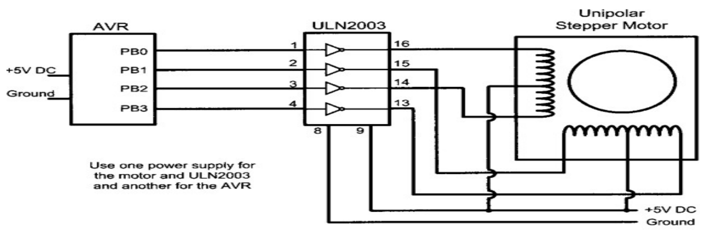


Interfacing of Stepper Motor:

- It is a widely used device to transmit electrical pulses into mechanical movements.
- Applications: Disk drives, Dot matrix printers, Robotics
- Stepper motor is a brushless DC motor that divides the full rotation angle of 360° into a number of equal steps.
- The motor is rotated by applying a certain sequence of control signals. The speed of rotation can be changed by changing the rate at which the control signals are applied.
- Stepper motor commonly have rotating permanent magnetic part called **rotor(or shaft)** surrounded by a stationary electromagnetic winding part called **stator.**



- As the current flow through the stator winding, it get energized & causes the rotor magnet to deflect or rotates.
- The direction of rotation is determined by the current flowing through the stator windings.



- Stepper motors can be driven in two different patterns or sequences. These are:
  - **Full Step Sequence:** In the full step sequence, two coils are energized at the same time and motor shaft rotates.

Normal Four-Step Sequence

Clockwise ↓ Counter-clockwise ↑

Step #	Winding A	Winding B	Winding C	Winding D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

  - **Half Step Sequence:** In Half mode step sequence, motor step angle reduces to half the angle in full mode. So, the angular resolution is also increased i.e. it becomes double the angular resolution in full mode. Also in half mode sequence the number of steps gets doubled as that of full mode. Half mode is usually preferred over full mode.

Half-Step 8-Step Sequence

Clockwise ↓ Counter-clockwise ↑

Step #	Winding A	Winding B	Winding C	Winding D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

NOTE:
Step Angle: It is defined as the angle traversed by the motor in one step. To calculate step angle, simply divide 360 by number of steps a motor takes to complete on evolution.

- A Unipolar Stepper Motor is interfaced to ATmega32 through **ULN2003A driver IC**, because the AVR lacks sufficient current to drive the stepper motor windings.
- A microcontroller can be used to apply different control signals to the motor to make it rotate according to the need of the application.
- The port to which the stepper motor driver is connected has to be configured as an output port.

Q) Write a C-program to rotate stepper motor in clock wise direction with full step sequence.
SOL:

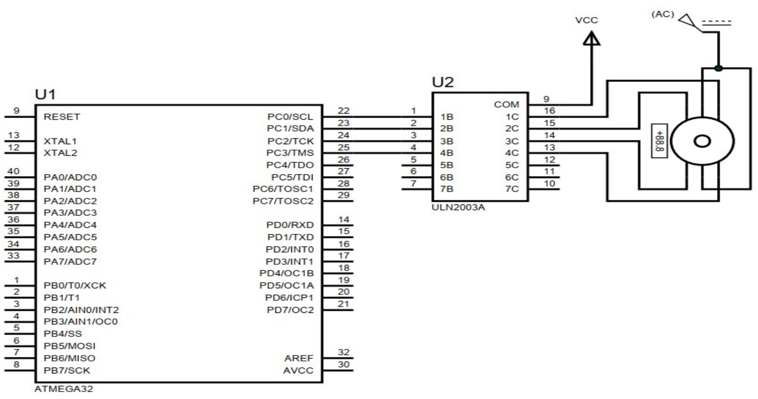
```
 #include <avr/io.h>
#include <util/delay.h>
int main(void)
```

```
{
        DDRC = 0xFF;            // PORT C is output
        while(1)
        {
                 PORTC = 0x09;
                 _delay_ms(100);
                 PORTC = 0x0C;
                 _delay_ms(100);
                 PORTC = 0x06;
                 _delay_ms(100);
                PORTC = 0x03;
                _delay_ms(100);
        }
 return 0;
 }
```

Q) Write a C-program to rotate stepper motor in clock wise direction with half step sequences.

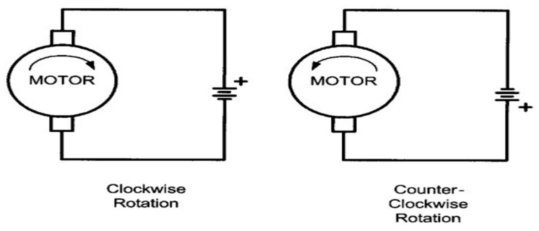
SOL:

```
 #include <avr/io.h>
#include <util/delay.h>
int main(void)
{
        DDRC = 0xFF;            // PORT C is output
        while(1)
        {
                 PORTC = 0x09;
                 _delay_ms(100);
                 PORTC = 0x08;
                 _delay_ms(100);
                 PORTC = 0x0C;
                 _delay_ms(100);
                PORTC = 0x04;
                _delay_ms(100);
                PORTC = 0x06;
                 _delay_ms(100);
                 PORTC = 0x02;
                 _delay_ms(100);
                 PORTC = 0x03;
                 _delay_ms(100);
                PORTC = 0x01;
                _delay_ms(100);
        }
 return 0;
 }
```



Interfacing of DC Motor:

- In DC motor only 2 leads(+ & -). Connecting them to DC voltage moves the motor in one direction. By reversing the polarity, motor moves in opposite direction.

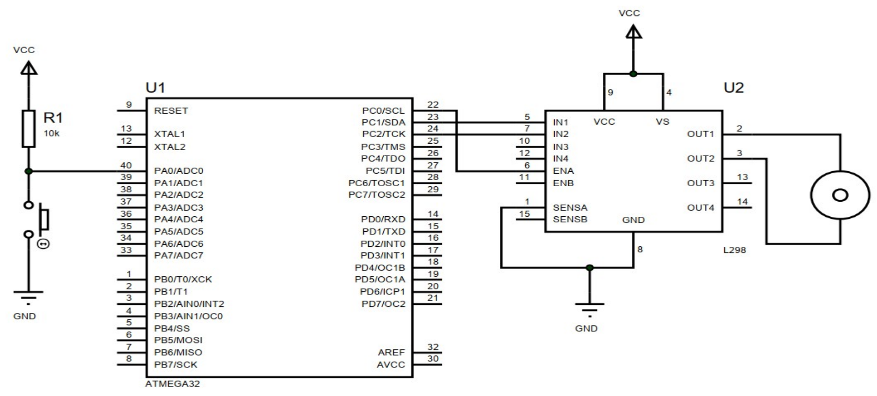


- Application: Small fans in motherboard to cool the CPU.
- DC Motor is interfaced to AVR through **L298 Motor driver IC.**
- The port pins to which the DC motor driver is connected has to be configured as output port pins.

Q) Write a C-program to monitor the status of a switch and perform the following:
(a) If switch is open, the DC motor rotates clockwise.
(b) If switch is closed, the DC motor rotates counter-clockwise.
SOL:

```
 #include <avr/io.h>
 int main(void)
 {
         DDRA = 0x00;          // PORT A is input
         DDRC = 0xFF;         // PORT C is output
         while(1)
                {
                        PORTC   =  0x01;                         //Enable L298
                         if (PINA & (1<<0))
                               {
                                  PORTC = 0x05;                  //L298 ENA = 1, IN1 = 0, IN2 = 1
                               }
                          else
                               {
                                   PORTC = 0x03;                 //L298 ENA = 1,  IN1 = 1, IN2 = 0
                                }
```

```
            }
        return 0;
}
```



Interfacing of Servo Motor:

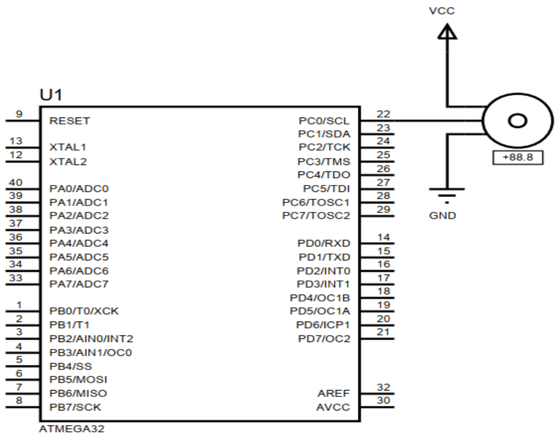
- Servo Motor is a DC Motor equipped with error sensing negative feedback mechanism to control the exact angular position of the shaft.
- Unlike DC Motors it will not rotate continuously. It is used to make angular rotations such as 0-90°, 0-180° etc.
- Stepper Motors can also be used for making precise angular rotations. But Servo Motors are preferred in angular motion applications like robotic arm, since controlling of servo motors are simple, **needs no extra drivers** like stepper motors and only angular motion is possible.
- The angular position is determined by the width of the pulse at the control input. Angular position can be controlled by varying the pulse width between 1μs to 2μs. Applying a pulse of approximate 1μs pulse width will rotate the servo motor to 0 degree, a pulse of 1.5μs pulse width will rotate the servo motor to 90 degrees and a pulse of 2μs pulse width will rotate the servo motor to 180 degrees.

- The port pins to which the servo motor driver is connected has to be configured as output port pins.

Q) Write a C-program to rotate servo motor 0^0, 90^0, 180^0.
SOL:

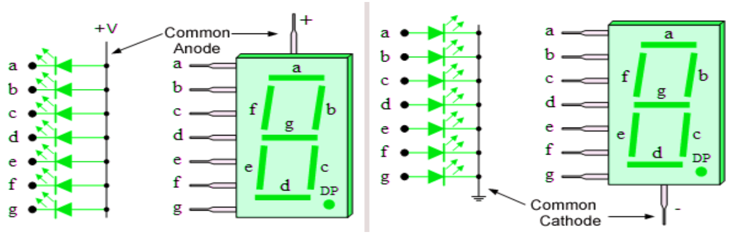
```
 #include <avr/io.h>
 #include <util/delay.h>
 int main(void)
 {
            DDRC = 0x01;            //Makes PC 0 output pin
            PORTC = 0x00;
            while(1)
                {
                         //Rotate Motor to 0 degree by applying 1 μs pulse
                          PORTC = 0x01;
```

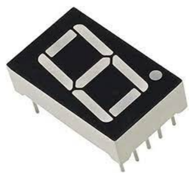
```
                    _delay_us(1000);
                    PORTC = 0x00;
                    _delay_ms(2000);

                  //Rotate Motor to 90 degree by applying 1.5µs pulse
                    PORTC = 0x01;
                  _delay_us(1500);
                    PORTC = 0x00;
                   _delay_ms(2000);

                 //Rotate Motor to 180 degree by applying 2µs pulse
                  PORTC = 0x01;
                  _delay_us(2000);
                   PORTC = 0x00;
                  _delay_ms(2000);
           }
  return 0;
}
```



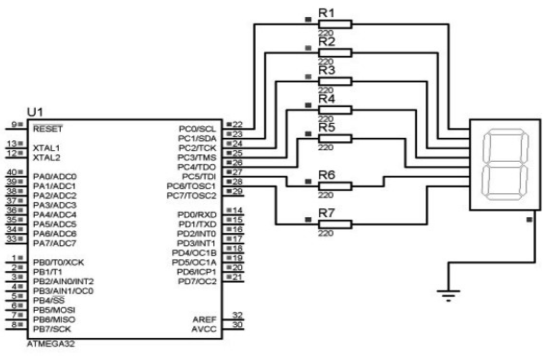
Interfacing of Seven Segment LED Display:

- 7-segment displays are made up of 8 LED segments. 7 of these LED segments are in the shape of a line, whereas 1 segment is circular.
- The 7 line-shaped LED segments are used for displaying numbers 0 to 9 and letters. The circular segment is used for displaying a decimal point.
- Each of the 8 elements has a pin associated with it which can be driven HIGH or LOW according to the type of display and the number or alphabet to be displayed.
- There are 2 types of seven-segment:
  - **Common Cathode:** In this type of segments all the cathode terminals are made common and tied to GND. Thus the LEDs needs a logic High signal(5v) in order to glow.
  - **Common Anode:** In this type of segments all the anodes terminals are made common and tied to VCC(5v). Thus the LEDs needs a logic LOW signal(GND) in order to glow.





- For interfacing with ATmega32, connect each pin from the seven-segment(a to g) display to port pins(P_0-P_6) through current limiting resistors and use the software to control the LEDs lighting and display the numbers we want.
- The port pins to which the seven-segment display is connected has to be configured as output port pins.
- For the seven-segment display the hex code represent the digit from 0 to 9 is shown below:



Numbers	Common Cathode		Common Anode	
	(DP)GFEDCBA	HEX Code	(DP)GFEDCBA	HEX Code
0	00111111	0x3F	11000000	0xC0
1	00000110	0x06	11111001	0xF9
2	01011011	0x5B	10100100	0xA4
3	01001111	0x4F	10110000	0xB0
4	01100110	0x66	10011001	0x99
5	01101101	0x6D	10010010	0x92
6	011111101	0x7D	10000010	0x82
7	00000111	0x07	11111000	0xF8
8	01111111	0x7F	10000000	0x80
9	01101111	0x6F	10010000	0x90

Q) Write a C-program to display digits 0 to 9 with 1s delay by turning ON the appropriate segments of 7 segment display.

SOL:

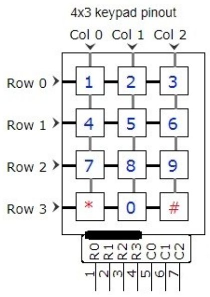
```
#include <avr/io.h>
#include <util/delay.h>
unsigned char seg_7[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
                                    // initializing an array to store the hex value to be displayed
```

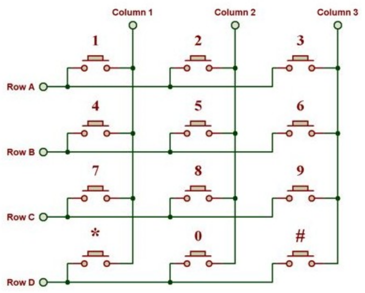
```
int main(void)
 {
            DDRC = 0xFF;            //port C is output
            PORTC = 0x00;
            int i= 0;
            while(1)
                   {
                          PORTC = seg_7[i];
                          _delay_ms(1000);
                           i = i+1;
                           if(i>=10)
                                   {
                                         i=0;
                                    }
                   }
         return 0;
 }
```

Q) Develop AVR embedded C program to display 00 and 99 with a 1sec delay in a seven segment display connected to its port. (6 marks)

Interfacing of Matrix Keyboard:

- It is one of the most widely used **input device** interfaced with a micro controller.
- A **matrix keypad** is an arrangement of push button switches arranged in **rows** and **columns** i.e. in matrix fashion.

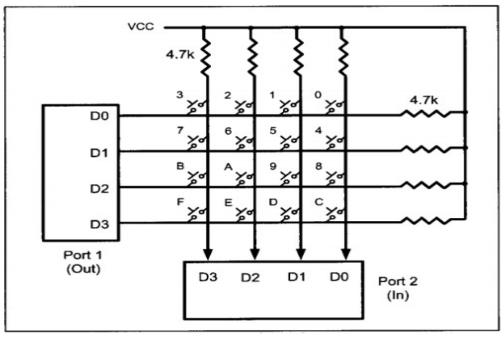




- The CPU accesses both rows and columns through ports, therefore it can be connected with a microcontroller.
- When a key is pressed, a column and a row make a contact, otherwise, there is no connection between rows and columns.
- The **rows** are connected to an **output port pins** and the **columns** are connected to an i**nput port pins**.
- To **detect which key is pressed** from the matrix keyboard, **the row lines are to be made low one by one and read the columns**. **Pull-up resistors** are connected to the column lines to limit the current that flows to the row line on a key press
- If no key has been pressed, reading the input port will give **1s** for all columns since they are connected to high (V_{CC}). **If all the rows are grounded and key is pressed, one of the columns**

have 0 since the key is pressed provides the path to ground. It is the function of microcontroller to scan the keyboard continuously to detect and identify the key pressed.

- If the data read from the columns is **D3-D0=1111**, no key has been pressed and the process continues until a key pressed is detected by system.
- However, if one of the column bits has a zero, this means that a key press has occurred. For example, if **D3 - D0 = 1101**, it means that a key in the **D1 column** has been pressed. After a key press is detected, the microcontroller will pass through the process of identifying the key.
- The output can be displayed using LEDs/7-segment display/ LCD display & port to whic these output devices are connected are configured as an output port.



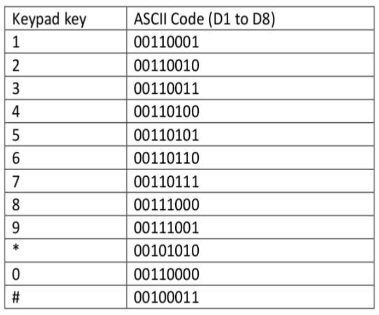
Q) Write a C-program to interface a 4X3 Matrix keypad with AVR microcontroller & display the results using LEDs connected to PORTD.

SOL:

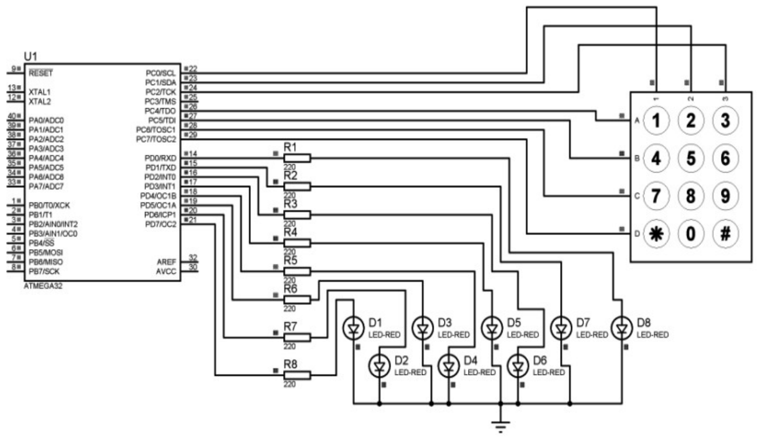
```
#include <avr/io.h>
#include <util/delay.h>
unsigned char keypad[4][3] = {'1','2','3','4','5','6','7','8','9','*','0','#'};
int main(void)
 {
        unsigned char colloc, rowloc;
         DDRD = 0xFF;                        //Port D is output for LED connection
         DDRC = 0xF8;                        //Port C PC7-PC4 is output for connecting Row lines &
                                             // PC2-PC0 is input for connecting Column lines

        PORTC = 0xFF;                        // Initializing PORT C values 1s for scanning
         while(1)                            //repeat forever
             {
                 while(1)
                      {
                         PORTC = 0xEF;                //ground row A
                         colloc = (PINC & 0x07);      //read the columns
                         if (colloc != 0x07)          //column detected
                                 {
                                      rowloc = 0;     // saves row location
                                       break;         // exit inner while loop
                                  }
                          PORTC = 0xDF;               //ground row B
                          colloc = (PINC & 0x07);     //read the columns
```

```
                        if (colloc != 0x07)                    //column detected
                                {
                                        rowloc = 1;              // saves row location
                                          break;                 // exit inner while loop
                                   }
                        PORTC = 0xBF;                         //ground row C
                        colloc = (PINC & 0x07);               //read the columns
                        if (colloc != 0x07)                    //column detected
                                {
                                        rowloc = 2;              // saves row location
                                        break;                   // exit inner while loop
                                }
                         PORTC = 0x7F;                        //ground row D
                         colloc = (PINC & 0x07);              //read the columns
                        if (colloc != 0x07)                    //column detected
                                 {
                                          rowloc = 3;            // saves row location
                                          break;                     // exit inner while loop
                                 }
                          }
// check column & send result to PORTD
                    if(colloc == 0x03)
                          {
                               PORTD = (keypad[rowloc][0]);
                          }
                     else if(colloc == 0x05)
                          {
                                 PORTD = (keypad[rowloc][1]);
                           }
                      else
                          {
                                  PORTD = (keypad[rowloc][2]);
                          }
                 }
            return 0;
  }
```

ASCII codes of keypad switches are:



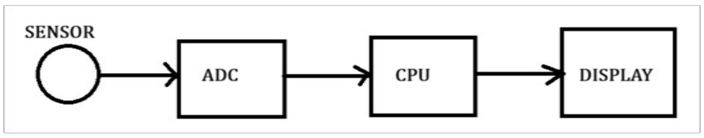
Keypad key	ASCII Code (D1 to D8)
1	00110001
2	00110010
3	00110011
4	00110100
5	00110101
6	00110110
7	00110111
8	00111000
9	00111001
*	00101010
0	00110000
#	00100011



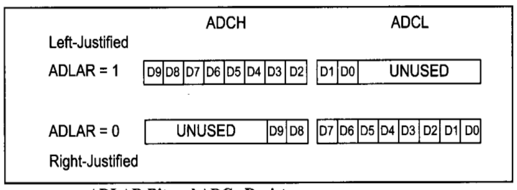
Q) Illustrate Matrix keyboard interfacing. (3 marks)
Q) Explain Matrix Keyboard interfacing with AVR. (6 marks)

ADC(Analog-to-Digital Conversion) interfacing:

- Digital computers use binary values, but in physical world everything is analog.
- The physical quantity(temperature, humidity, pressure..) is converted to electrical signal(voltage ,current..)using sensors(transducer) .
- Sensors produce an output voltage or current which is analog.
- So we need an ADC to translate analog to digital numbers.

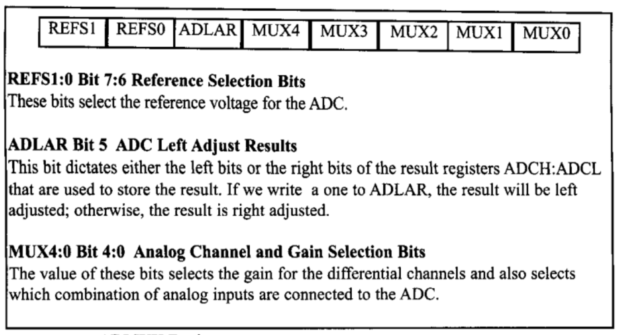


- ATmega32 has an on-chip ADC:
  - 10 bit ADC
  - 8 analog input channel(ADC0,ADC1,…..ADC7), 7 differential input channels, 2 differential input channels with optional gain of 10x and 200x.
  - The converted output binary data is held by two special function registers: ADCL and ADCH.
  - ADCH:ADCL registers can hold 16 bits, but ADC output is 10 bits wide, 6 bits of the 16 are unused. We have the option of making either upper 6 bits or lower 6 bits unused
  - We have 3 options for Vref . Vref can can be connected to Avcc(Analog Vcc), internal 2.56V reference or external AREF pin.
  - Conversion time is dictated by the crystal frequency connected to XTAL pins(Fosc) and ADPSC 0:2 bits.
  - 5 registers associated with ADC
    - ADCH(high data)
    - ADCL(low data)
    - ADCSRA(ADC Control & Status Register)
    - ADMUX (ADC Multiplexer)
    - SPIOR(Special function I/O register)



	ADCH	ADCL
Left-Justified ADLAR = 1	D9 D8 D7 D6 D5 D4 D3 D2	D1 D0 UNUSED
ADLAR = 0 Right-Justified	UNUSED D9 D8	D7 D6 D5 D4 D3 D2 D1 D0

ADLAR Bit and ADCx Registers



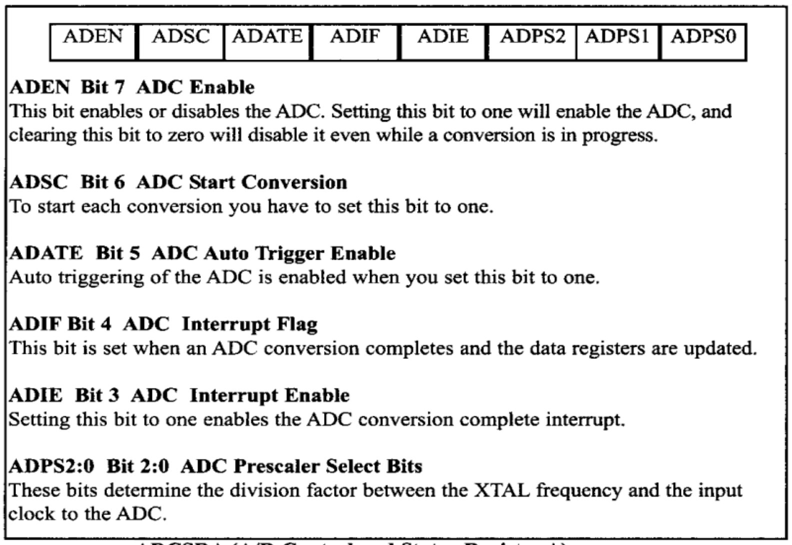
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1:0 Bit 7:6 Reference Selection Bits
These bits select the reference voltage for the ADC.

ADLAR Bit 5 ADC Left Adjust Results
This bit dictates either the left bits or the right bits of the result registers ADCH:ADCL that are used to store the result. If we write a one to ADLAR, the result will be left adjusted; otherwise, the result is right adjusted.

MUX4:0 Bit 4:0 Analog Channel and Gain Selection Bits
The value of these bits selects the gain for the differential channels and also selects which combination of analog inputs are connected to the ADC.

ADMUX Register



ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN Bit 7 ADC Enable
This bit enables or disables the ADC. Setting this bit to one will enable the ADC, and clearing this bit to zero will disable it even while a conversion is in progress.

ADSC Bit 6 ADC Start Conversion
To start each conversion you have to set this bit to one.

ADATE Bit 5 ADC Auto Trigger Enable
Auto triggering of the ADC is enabled when you set this bit to one.

ADIF Bit 4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the data registers are updated.

ADIE Bit 3 ADC Interrupt Enable
Setting this bit to one enables the ADC conversion complete interrupt.

ADPS2:0 Bit 2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADCSRA (A/D Control and Status Register A)

ADC start conversion: The **ADSC** bit of ADCSRA register is the start conversion input.

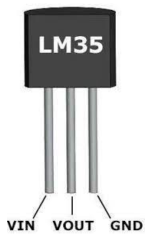
Analog to Digital conversion time: By using ADSP2-ADSP0 bits of ADCSRA register we can select the A/D conversion time.

ADPS2	ADPS1	ADPS0	ADC Clock
0	0	0	Reserved
0	0	1	CK/2
0	1	0	CK/4
0	1	1	CK/8
1	0	0	CK/16
1	0	1	CK/32
1	1	0	CK/64
1	1	1	CK/128

Q) Outline ADC interfacing with AVR. (3 marks)

Interfacing of Temperature Sensor(LM34/35):

- **LM34/35** is an analog sensor that converts the surrounding temperature to a proportional analog voltage. **LM34** produce analog voltage proportional to temperature in proportional to temperature in **Fahrenheit**(ºF) & **LM35** produce analog voltage proportional to temperature in **Celsius**(ºC).

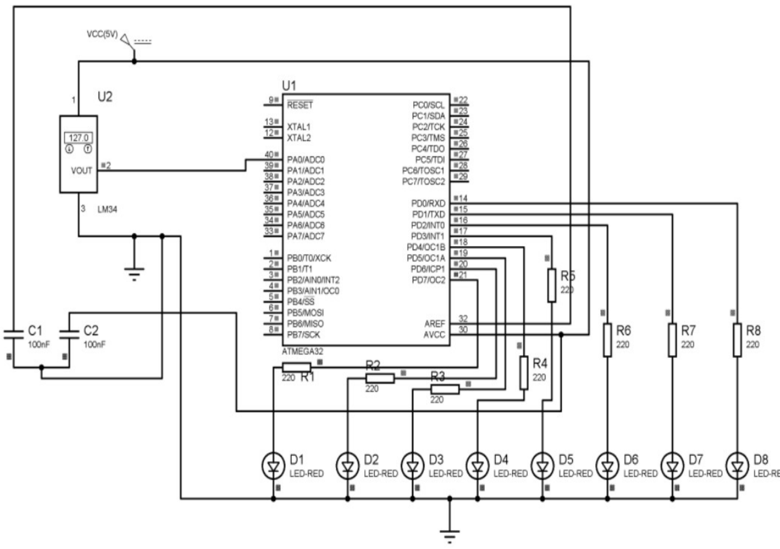


- To use ATmega32 ADC, the analog input need to be connected to **PORTA** pins which are multiplexed as ADC input pins (labeled as **ADC0 to ADC7**).
- **LM34/35 Temperature sensor (V_{OUT})** is connected to any one pin of **PORTA** of ATmega32. PORTA is configured as an input port.
- The output can be displayed using **LEDs/7-segment display/ LCD display** & port to which these output devices are connected are configured as an output port.
- The **ADC** has **10-bit** resolution. So number of steps will be 1024. We use the internal **2.56 V** reference voltage as **Vref**, so the step size would be 2.56 V/1024 = 2.5 mV.
- LM34/35 sensor produces **10 mV** for each degree of temperature change and the ADC step size is **2.5 mV**. So each degree of temperature change corresponds to **4 ADC steps(10 mV/2.5 mV = 4)**. This makes the binary output number for the ADC four times the real temperature. So the 10-bit output of the ADC is divided by 4 to get the real temperature.
- To divide the **10-bit output** of the ADC by **4** we choose the left-justified option and only read the **ADCH** register. It is same as shifting the result 2 bits right (division by 4 in binary).
- A **100nF** capacitor is connected between the AREF pin and GND to make Vref voltage more stable and increase the precision of ADC. ATmega32 AVCC pins are connected to **5V DC**.
- The port pin to which LM34/35 sensor is connected are configured as an input pin.

Q) Write a C-program to temperature sensor(LM34/35) interfacing with AVR using ADC peripheral of ATmega32 & display the results using LEDs connected to PORTD.

SOL:

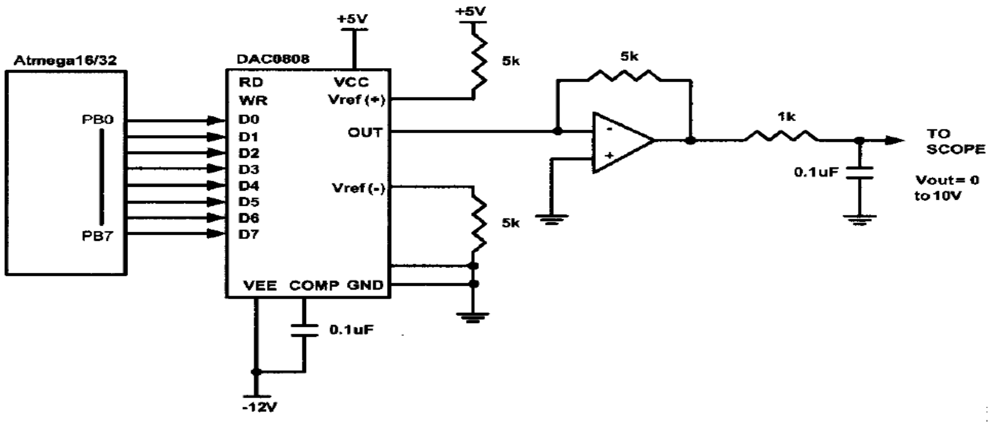
```
#include <avr/io.h>
int main(void)
    {
            DDRD = 0xFF;        //port D output
            DDRA = 0x00;        //port A input. LM 34 connected to ADC0
            ADCSRA = 0x87;      //ADC enable, ck/128
            ADMUX = 0xE0;       //2.56 V ref, ADC0 single ended left justified data

            while(1)
                {
                  ADCSRA |= (1<<ADSC);              //start conversion
                  while((ADCSRA & (1<<ADIF))==0); //wait for EOC flag ADIF
                  PORTD = ADCH;                     //high byte to port B
                }

            return 0;
    }
```



DAC(Digital to Analog converter) interfacing:

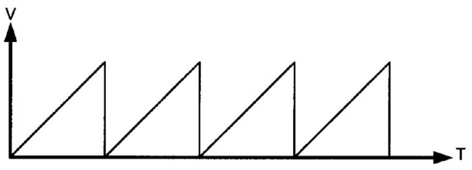
- Digital to Analog converter is a device widely used to convert digital pulses to analog signals.
- **DAC0808** is commonly used for interfacing with AVR**.** In DAC0808, the digital inputs are converted to current (I_{out}). It is a R-2R ladder type DAC.
- 741 Op Amp is configured as a I to V converter.
- The port to which DAC0808 is connected are configured as an **output port**.
- Usually **PORTA** is used for **ADC** operations**,** so other ports (**PORTB, PORTC, PORTD**) are used foe **DAC** operations.

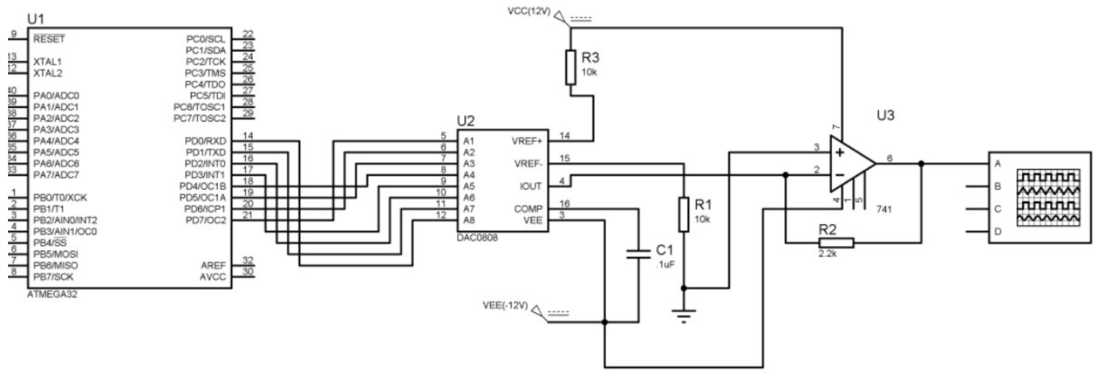
- **8** digital input pins **D_0 to D_7** of DAC0808 are connected to respective port.
- The port pins to which DAC is connected are configured as an output pins.



Q) Write a C-program to interfacing DAC0808 with ATmega32 for the generation of a stair case ramp waveform & observe the result using an oscilloscope.
SOL:

```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
    {
            unsigned char i = 0;
            DDRD = 0xFF;
            while(1)
                    {
                            PORTD = i;
                            _delay_us(100);
                            i++;
                    }
            return 0;
    }
```

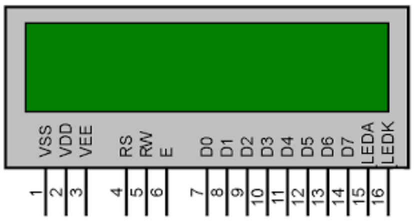


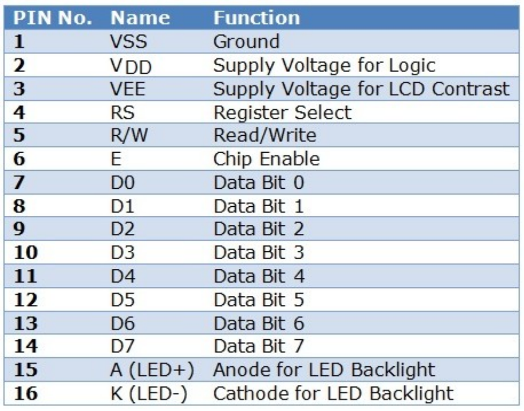


LCD(Liquid Crystal Displays) interfacing:

- LCDs are used for displaying status or parameters in embedded systems.
- LCD 16x2 is a 16-pin device that has 8 data pins (D0-D7) and 3 control pins (RS, RW, EN). The remaining 5 pins(VSS, VDD, VEE, LEDA, LEDK) are for the 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 also when to read or write.
- The port pins to which LCD is connected are configured as an output pins.

Pin diagram of a 16x2 LCD:





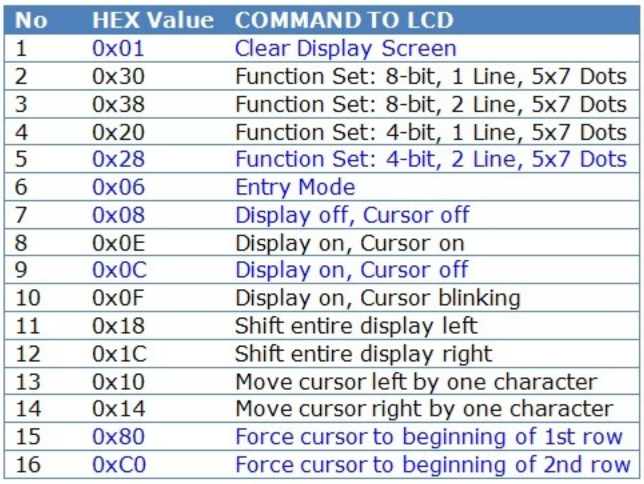
PIN No.	Name	Function
1	VSS	Ground
2	VDD	Supply Voltage for Logic
3	VEE	Supply Voltage for LCD Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Chip Enable
7	D0	Data Bit 0
8	D1	Data Bit 1
9	D2	Data Bit 2
10	D3	Data Bit 3
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7
15	A (LED+)	Anode for LED Backlight
16	K (LED-)	Cathode for LED Backlight

Registers of LCD: A 16×2 LCD has 2 registers:

- **Data register:** The main function of the data register is to store the information(ASCII value) which is to be exhibited on the LCD screen.
- **Command register:** The main function of the command register is to store the instructions of command which are given to the display.

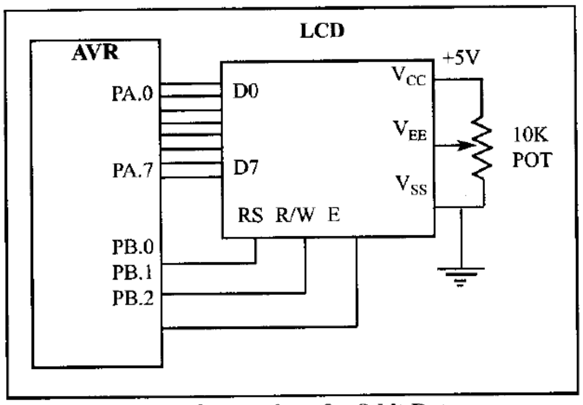
The **RS (register select)** is mainly used to change from one register to another. When the register set is '**0**', then it is known as **command register**. Similarly, when the register set is '**1**', then it is known as **data register**.

Command Codes to LCD:



No	HEX Value	COMMAND TO LCD
1	0x01	Clear Display Screen
2	0x30	Function Set: 8-bit, 1 Line, 5x7 Dots
3	0x38	Function Set: 8-bit, 2 Line, 5x7 Dots
4	0x20	Function Set: 4-bit, 1 Line, 5x7 Dots
5	0x28	Function Set: 4-bit, 2 Line, 5x7 Dots
6	0x06	Entry Mode
7	0x08	Display off, Cursor off
8	0x0E	Display on, Cursor on
9	0x0C	Display on, Cursor off
10	0x0F	Display on, Cursor blinking
11	0x18	Shift entire display left
12	0x1C	Shift entire display right
13	0x10	Move cursor left by one character
14	0x14	Move cursor right by one character
15	0x80	Force cursor to beginning of 1st row
16	0xC0	Force cursor to beginning of 2nd row

Programming for LCD16x2 with ATmega32



LCD Connections for 8-bit Data

Initialize LCD:

- Power ON the LCD
- Wait for 15 ms (‘Power ON’ initialization time for LCD)
- Send 0x38 command to initialize 2 lines, 5x8 matrix, 8-bit mode LCD
- Send any ‘Display ON’ command (0x0E, 0x0C) to LCD
- Send 0x06 command (increment cursor) to LCD

Command Write function:

- Send the command value to the LCD16x2 data port.
- Make RS pin low, RS = 0 (command reg.)
- Make RW pin low, RW = 0 (write operation)
- Give high to low pulse at the Enable (E) pin of a minimum delay of 450 ns.

Data write function:

- Send command to the data port.
- Make the RS pin High, RS = 1 (Data reg.)
- Make the RW pin Low, RW = 0 (Write operation)
- Give high to low pulse at the Enable (E) pin

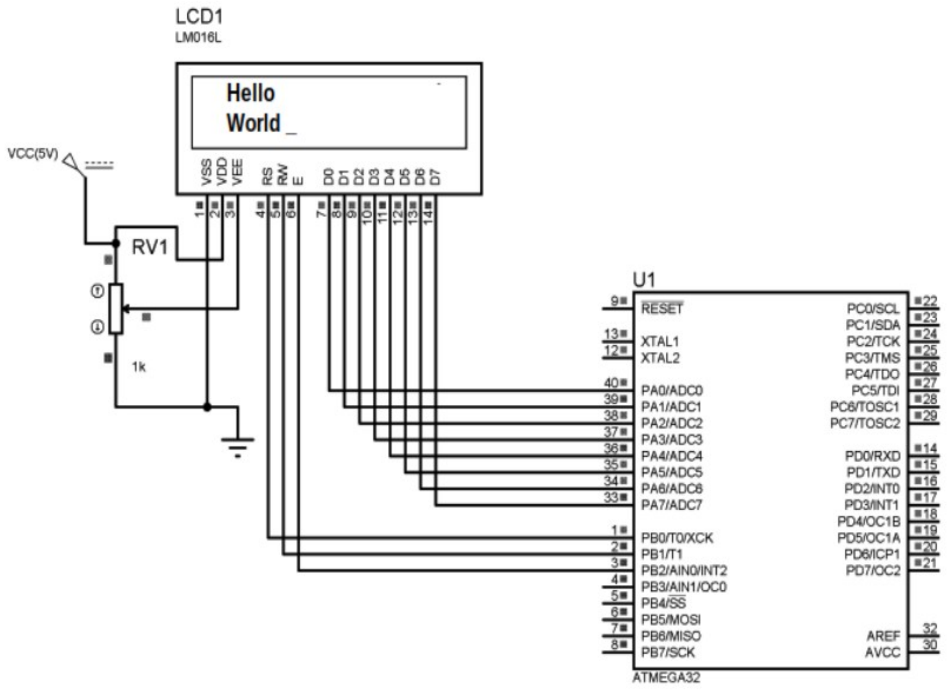
Display String function

- This function takes a string (an array of characters) and sends one character at a time to the LCD data function till the end of the string. A ‘for loop’ is used for sending a character in each iteration. A NULL character indicates end of the string.

Write a C-program to interfacing 16x2 LCD with ATmega32.
SOL:

```
#define F_CPU 1000000UL //1 MHz clock
#include <avr/io.h>
#include <util/delay.h>

int main(void)
        {
                DDRA = 0xFF;                //LCD data pins connected to Port A
                DDRB = 0xFF;                //LCD ctrl pins connected to Port B
                PORTB &= ~(1<<2);           //EN = 0
                _delay_us(2000);
                lcdCommand(0x38);           //2 line 5X7 matrix
                lcdCommand(0x0E);           //display on cursor on
                lcdCommand(0x01);           //clear lcd
                _delay_us(2000);
                lcdCommand(0x06);           //shift cursor right
                lcdCommand(0x80);
                lcd_print("Hello");
                lcdCommand(0xC0);
                lcd_print("World");

        while(1);
        return 0;
        }

void lcdCommand(unsigned char cmnd)
        {
                PORTA = cmnd;
                PORTB &= ~(1<<0);           //RS = 0 for command
                PORTB &= ~(1<<1);           //RW = 0 for write
                PORTB |= (1<<2);     /      /EN=1 for H to L pulse
                _delay_us(1);
                PORTB &= ~(1<<2);           //EN=0 for H to L pulse
                _delay_us(100);
        }

void lcdData(unsigned char data)
        {
                PORTA = data;
                PORTB |= (1<<0);             //RS = 1 for data
                PORTB &= ~(1<<1);           //RW = 0 for write
                PORTB |= (1<<2);            //EN=1 for H to L pulse
                _delay_us(1);
                PORTB &= ~(1<<2);           //EN=0 for H to L pulse
                _delay_us(100);

}
void lcd_print(char *str)
        {
                unsigned char i = 0;
                while(str[i] != 0)
                {
                        lcdData(str[i]);
                        i++;
                }
```

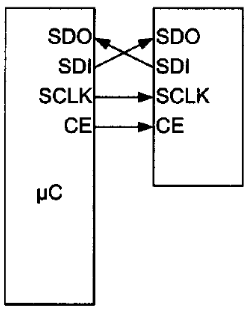
```
    }
```



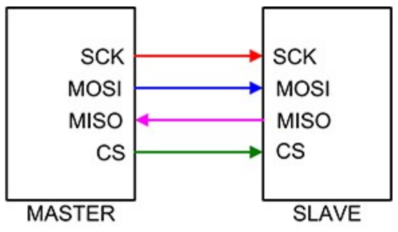
Q) Develop embedded C program for interfacing LCD with AVR. (6 marks)

Serial Peripheral Interface (SPI):

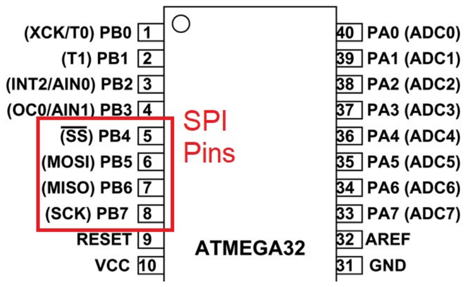
- It is a **bus interface connection protocol** originally started by **Motorola Corp(Freescale).**
- SPI is a **synchronous serial interface** in which data in an 8-bits can be shifted in and/or out one bit at a time.
- It can be used to communicate with a serial peripheral device(flash, EEPROM, sensors) or with another microcontroller with an SPI interface
- **It uses four pins for communication.**
  - **SDI** (Serial Data Input)
  - **SDO** (Serial Data Output)
  - **SCLK** (Serial Clock)
  - **CE**(Chip Enable)/**CS** (Chip Select)
- It has two pins for **data transfer** called **SDI (Serial Data Input)** and **SDO (Serial Data Output)**. **SCLK (Serial Clock)** pin is used to synchronize data transfer and Master provides this clock. **CS (Chip Select)** pin is used by the master to select the slave device.



- SPI peripherals in AVR can operate in **Master or Slave modes**. If **master mode** is selected, then it takes care of generating the clock signal and starting the transfer. The slave only waits for the CS line to pull down to get ready for transfer.



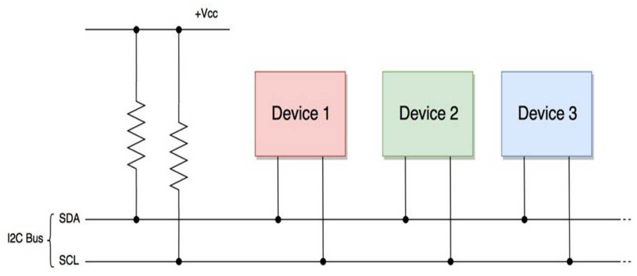
- ATmega32 has an inbuilt SPI module. It can act as a master and slave SPI device. SPI communication pins in AVR ATmega are:
    - **MISO**(Master In Slave Out): The Master receives data and the slave transmits data through this pin.
    - **MOSI**(Master Out Slave In): The Master transmits data and the slave receives data through this pin.
    - **SCK**(Synchronization Clock): The Master generates this clock for the communication, which is used by the slave. The only master can initiate a serial clock.
    - **SS**(Slave Select): Master can select slaves through this pin.



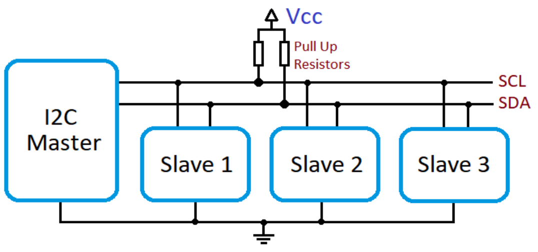
Q) Explain SPI interfacing. (3 marks)

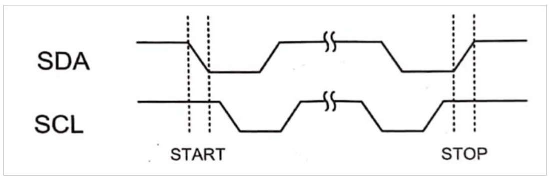
I2C or IIC(Inter-Integrated Circuit):

- It is a bus interface connection incorporated into many devices such as sensors, RTC(Real Time Clock) and EEPROM.
- I2C ideal for **attaching low speed peripherals** to a motherboard or embedded system or anywhere that a reliable communication over a short distance is required.
- I2C devices use only **2 pins** for data transfer.
    - **SCL**(Serial Clock): The line carries the clock signal & it synchronize the data transfer.
    - **SDA**(Serial Data): The line send and receive data.
- SDA and SCL make the I2C a **2 wire interface(TWI)**.
- I2C devices use only 2 bidirectional open drain pins for data communication.
- To implement I2C, only **4.7KΩ** pull up resistor for each bus line is needed.

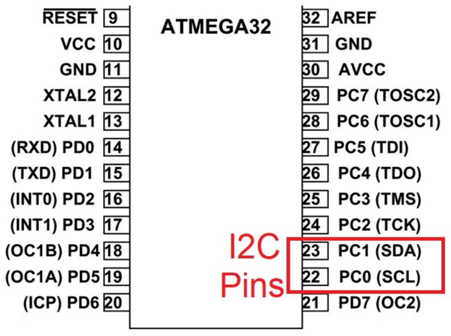


- In the AVR upto 120 different devices can share an I2C bus.
- Each of these device is called node.
- In I2C terminology, each node can operate as either **master or slave**.
- **Master** is a device that generates the clock for the system. It also initiates and terminates a transmission.
- **Slave** is the node that receives the clock and is addressed by the master.
- In I2C both master and slave can receive or transmit data.





- I2C is a connection oriented communication protocol. That means each transmission is initiated by a **START** condition and is terminated by a **STOP** condition.
- **START** and **STOP** conditions are generated by keeping the level of the **SCL** line high and then changing the level of **SDA** line.
- **START** condition is generated by a **high to low change in SDA line when SCL is high**.
- **STOP** condition is generated by a **low to high change in SDA line when SCL is high**.
- ATmega32 has an inbuilt I2C module. I2C communication pins in AVR ATmega are:



Q) Explain I2C interfacing with AVR. (6 marks)
Q) I2C is a ……………. communication protocol (1 mark)

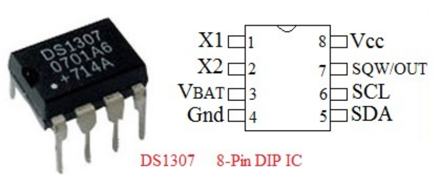
RTC(Real Time clock):

- A widely used device that provides accurate time and data information for many applications.
- Many systems such as x86 PC come with such a chip on motherboard.
- RTC chip in x86 PC provides the time components of hour, minute and second, in addition to the date /calender components of year, month, and day.
- Many RTC chips use an internal battery, which keeps the time and date even when the power is off.
- DS 1307 is a serial RTC with an I2C BUS.

DS 1307 features:

- It is a Serial RTC
- Clock/calender provides seconds, minutes, hours, day, date, month and year information.

- Clock operates in either the 24 hour or 12 hour format with AM/PM indicator.
- Has a built in power sense circuit that detects power failures and automatically switches to battery supply.
- Total of 64 bytes of RAM space with addresses 00-3FH.
  - **Pin out of DS1307:**
  - **X1-X2**
    - Input pins that allows connection to an external
    - crystal oscillator to provide clock source to the chip.
    - 32.768KHz quartz crystal is used in standard.



  - **V_{BAT}**
    - This pin is connected to an external +3V lithium
    - battery(power source in the absence of external supply voltage).
    - Connect this pin to ground when not used.
  - **GND**
    - PIN 4 is grounded.
  - **SDA(Serial Data)**
    - Must be connected to the SDA line of I2C bus.
  - **SCL(Serial Clock)**
    - Must be connected to the SCL LINE OF I2C bus.
  - **SWQ/OUT**
    - Output pin providing 1KHz, 4KHz, 8KHz, or 32 KHz frequency if enabled.
    - This pin needs an external pull up resistor to generate the frequency because it is open drain.
    - If not used, omit external pull up resistor

Q) Explain RTC interfacing with AVR. (6 marks)
Q) The expansion form of RTC is ………………………… (1 mark)