CHAPTER 4

Real World Interfacing with AVR and Case Studies

Objectives ...

- To learn interfacing of I/O Devices such as LED, Push Button, Buzzer to AVR.
- To understand the interfacing of Seven Segment Display, Thumbwheel Switch, DC and Stepper Motor, DAC and LCD to AVR.
- To study various case studies such as Traffic Light Controller using AVR, Single Digit event counter using Opto-interrupter and SSD.
- To understand Real Time Clock Using IC DS1307 Chip, Temperature Monitoring System using LM35 Sensor, Smart Phone Controlled Devices using Bluetooth Module HC05.

INTRODUCTION

In previous chapters, the architecture of AVR including its features, the details of timers, memory structure, serial programming, I2C, SPI and on-chip ADC were studied.

In this chapter, interfacing of I/O devices such as LED, push buttons, buzzers, DAC, seven segment display, LCD are discussed. The AVR C program for all I/O devices are also be studied.

In the last section of this chapter, few case studies such as Traffic Light controller using AVR, Single digit event counter using opto-interrupter and SSD, Real time clock using IC DS1307 chip, temperature monitoring system using LM35 sensor are discussed.

4.1 LED INTERFACING

The Light Emitting Device (LED) is an output device which gives indication when it is On or Off.

The LED interfacing with AVR is shown in below Fig. 4.1.

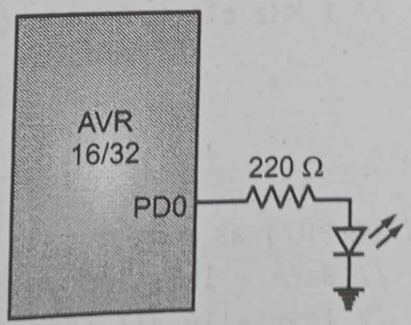

Fig. 4.1: Interfacing of LED with AVR

AVR C Program:

```
#define F_CPU 1000000UL  // Define CPU frequency as 1 MHz (adjust as needed)
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRD |= (1 << PD0);          // Set PD0 as output
    while (1)
    {
        PORTD |= (1 << PD0);     // Turn ON LED (PD0 = HIGH)
        _delay_ms(500);          // Delay 500ms
        PORTD &= ~(1 << PD0);
                                 // Turn OFF LED (PD0 = LOW)
        _delay_ms(500);          // Delay 500ms
    }
}
```

LED Blinking:

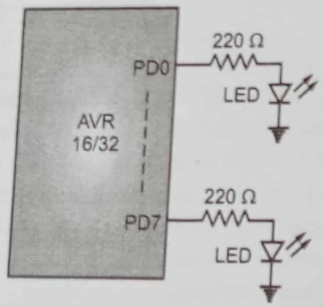

Fig. 4.2: Interfacing of LEDs with AVR

Eight LEDs are connected to PORT D (PD0-PD7).

AVR C Program:

```
#define F_CPU 1000000UL     // 1 MHz clock frequency
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    // Set all PORTD pins (PD0-PD7) as output
    DDRD = 0xFF;            // 0xFF = 11111111 in binary
    PORTD = 0x00;           // Initially all LEDs OFF
```



```
    while (1)
    {
        PORTD = 0xFF;
        _delay_ms(500);      // Turn ON all LEDs
        PORTD = 0x00;        // Delay 500 ms
        _delay_ms(500);      // Turn OFF all LEDs
    }                        // Delay 500 ms
}
```

4.2 INTERFACING OF PUSH BUTTON

A push button is small controlling device similar to switch which can operate any electrical device by pressing it. When push button is interfaced to AVR, its status, that is, it is pressed or not, can be read. Push button can be connected in two ways; pull-on mode or pull-down mode as shown in Fig. 4.2.

Pull-down Mode:

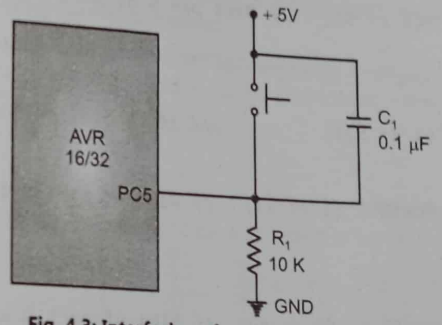

Fig. 4.3: Interfacing of push button with AVR

In Pull-down mode, when a push button is not pressed, a logic low input ('0) appears on the ATmega32 pin.

Pull-up Mode:

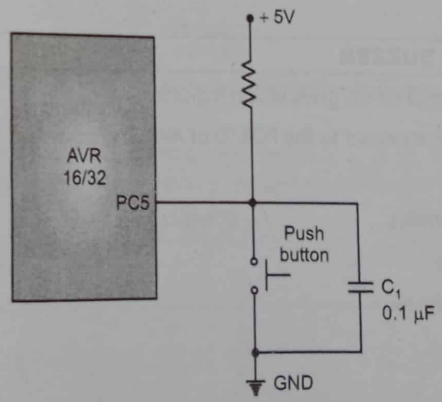

Fig. 4.4: Interfacing of push button with AVR

In Pull-up mode, when a push button is not pressed, a logic high ('1) input appears at ATmega32 pin.

AVR C Program:

A push button is connected to the fifth pin of PORT C (PC5), while an LED is connected to the fourth pin of PORT D (PD4). When button is pressed, the LED will glow; otherwise, it will remain off.

```
#define F_CPU 1000000UL          // Define CPU frequency as 1 MHz
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
   DDRD = DDRD | (1 << 4);     // Make pin 4 of port D as an output
   DDRC = DDRC & ~(1 << 5);    // Make pin 5 of port C as an input
   while (1)
   {
      if (PINC & (1 << 5))           // if PIN5 of port C is high
      {
         PORTD = PORTD | (1 << 4);   // PIN4 of port D is high
      } else
      {
         PORTD = PORTD & ~(1 << 4);  // PIN4 of port D will remain low
      }
   }
}
```

4.3 INTERFACING OF BUZZER

Buzzer is an output device which gives sound indication.

Assuming buzzers are connected to the PORTD of AVR, C program is:

AVR C Program:

```
#define F_CPU 1000000UL          // 1 MHz clock
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
   // Set PD0 as output
   DDRD |= (1 << PD0);
   while (1)
   {
      // Turn ON the buzzer
      PORTD |= (1 << PD0);
      _delay_ms(1000);              // Buzzer ON for 1 sec
      // Turn OFF the buzzer
      PORTD &= ~(1 << PD0);
      _delay_ms(1000);              // Buzzer OFF for 1 sec
   }
}
```


4.4 INTERFACING OF SEVEN SEGMENT DISPLAY

A Seven Segment Display (SSD) is an electronic display device used to display decimal numerals and some alphabets. It is commonly used in digital clocks, electronic meters, calculators and other devices that display numerical information.

A seven-segment display consists of seven LED segments arranged in a rectangular fashion. Each segment is named 'a' to 'g', and an additional dot (dp) for decimal point display.

There are two types of seven segment display-

1. **Common Cathode:**

All the cathode terminals are made common and tied to GND. Thus, the segments a to g needs a logic High signal (5 V) in order to glow as shown in below Fig. 4.5.

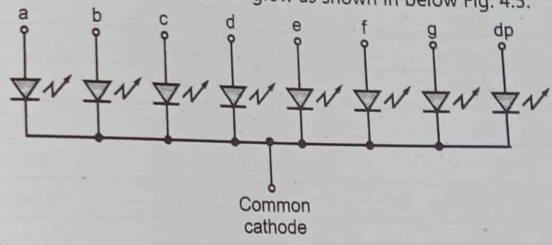

Fig. 4.5: Common cathode seven segment display

2. **Common Anode:**

All the anode terminals are made common and tied to VCC (5 V). Thus, the segments a to g needs a logic LOW signal (GND) in order to glow as shown in below Fig. 4.6.

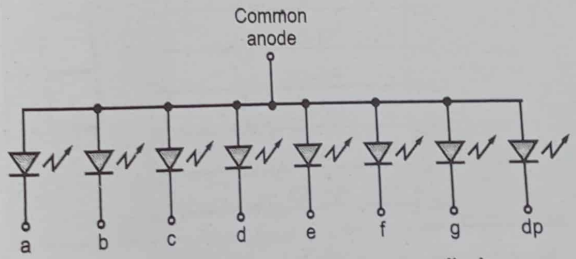

Fig. 4.6: Common anode seven segment display

Seven Segment Display:

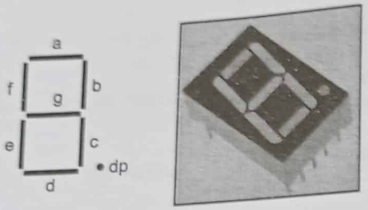

Fig. 4.7: Seven segment display

To display numbers (0-9), different combinations of segments are turned ON. Hex codes for Common Cathode Seven Segment Display are given in below table.

Table 4.1

Digit	Segments On	Binary Code (g-f-e-d-c-b-a)	Hex
0	a b c d e f	0b00111111	0x3F
1	b c	0b00000110	0x06
2	a b g e d	0b01011011	0x5B
3	a b g c d	0b01001111	0x4F
4	f g b c	0b01100110	0x66
5	a f g c d	0b01101101	0x6D
6	a f g e c d	0b01111101	0x7D
7	a b c	0b00000111	0x07
8	All	0b01111111	0x7F
9	a b c d f g	0b01101111	0x6F

AVR C Program for Seven Segment Display:

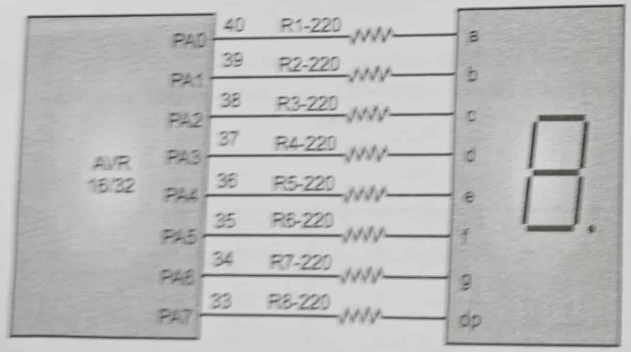

Fig. 4.8: Interfacing of seven segment display with AVR

AVR C Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
// Lookup table for digits 0-9          (Common Cathode, g-f-e-d-c-b-a)
uint8_t seg_code[10] =
{
    0x3F,     // 0 -> 0b00111111
    0x06,     // 1
    0x5B,     // 2
    0x4F,     // 3
    0x66,     // 4
    0x6D,     // 5
    0x7D,     // 6
    0x07,     // 7
    0x7F,     // 8
    0x6F      // 9
};
int main(void)
{
    DDRA = 0x7F; // Set PA0-PA6 as output (0111 1111)
    while (1)
    {
        for (int i = 0; i < 10; i++)
        {
            PORTA = seg_code[i]; // Send digit pattern to PORTD
            _delay_ms(1000);     // Wait for 1 second
        }
    }
}
```

4.5 INTERFACING OF THUMBWHEEL SWITCH

A thumbwheel switch is a mechanical device used for inputting BCD (Binary Coded Decimal) values manually. Each switch outputs 4-bit BCD code corresponding to the selected digit (0–9). Each switch is provided with a wheel. This wheel can be rotated by a thumb to change the input number hence the name.

Now-a-days instead of wheel, the thumb-wheel switch is provided with two push buttons. One push button to increment the numeric value and other to decrement the numeric value as shown in below Fig. 4.9.

Fig. 4.9: Thumbwheel switch

Source: https://www.radwell.eu/uk/buy/omron-a7ps-254-1/99108.html

Each thumbwheel has four data pins and one control input (c). Four data pins represent input number in binary form. Assuming the control input is connected to ground and each data line is tied to V_{cc}, below table shows the information on the data pins for all numeric inputs.

Table 4.2

Input	c	A3	A2	A1	A0
0	0	1	1	1	1
1	0	1	1	1	0
2	0	1	1	0	1
3	0	1	1	0	0
4	0	1	0	1	1
5	0	1	0	1	0
6	0	1	0	0	1
7	0	1	0	0	0
8	0	0	1	1	1
9	0	0	1	1	0

Similar to switches, thumbwheel switches are also interfaced through input ports. Each thumbwheel requires four input pins as shown below (Fig. 4.10):

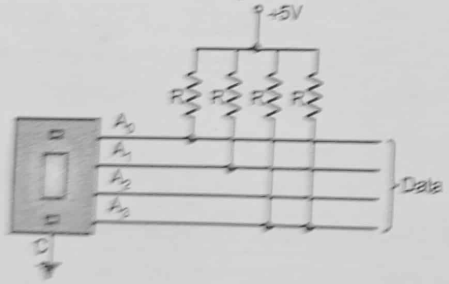

Fig. 4.10: Thumbwheel switch

These pins are typically connected to a microcontroller's input port.

Interfacing of Thumbwheel to AVR:

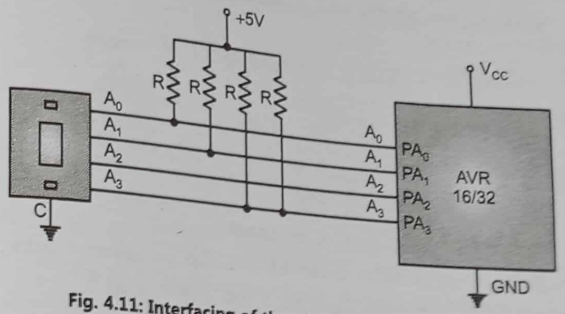

Fig. 4.11: Interfacing of thumbwheel switch with AVR

AVR C Program to read Thumbwheel Switch Input:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
void uart_init(unsigned int ubrr)
{
   // Set baud rate
   UBRRH = (unsigned char)(ubrr>>8);
   UBRRL = (unsigned char)ubrr;
   // Enable transmitter
   UCSRB = (1<<TXEN);
   // Set frame format: 8data, 1stop bit
   UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
}
void uart_transmit(unsigned char data)
{
   while (!( UCSRA & (1<<UDRE)));   // Wait for empty transmit buffer
   UDR = data;                      // Put data into buffer
}
void uart_print(const char* str)
{
   while(*str)
   {
      uart_transmit(*str++);
   }
}
```

```
void int_to_ascii_and_send(uint8_t value)
{
    uart_transmit('0' + value); // Send ASCII of number
    uart_transmit('\n');
}
int main(void)
{
    DDRA = 0x00;    // Set Port A as input
    PORTA = 0x0F;   // Enable internal pull-up on lower 4 bits
    uart_init(6); // For 9600 baud at 1 MHz (UBRR = 6)
    while (1)
    {
        uint8_t input = PINA & 0x0F; // Mask upper bits, read only PA0-PA3
        int_to_ascii_and_send(input);
        _delay_ms(500); // Delay to avoid multiple readings
    }
}
```

PINA & 0x0F only reads the lower 4 bits (PA0–PA3), which receive BCD input.

UART is used to print the value on a serial monitor for testing purposes.

For an actual embedded application, the number can be displayed on a 7-segment or LCD.

4.6 INTERFACING OF DC AND STEPPER MOTOR

4.6.1 DC Motor

A Direct Current (DC) motor translates electrical pulses into mechanical movement. A DC motor is an electromechanical device that converts direct electrical energy into mechanical rotational energy. It is used in robotics, automation, fans, toys and embedded systems.

There are two types of DC motors:

1. **Brushed DC Motor:** It has simple construction, it requires a commutator, brushes and it is easy to interface with H-bridge drivers.
2. **Brushless DC Motor (BLDC):** It requires electronic commutation. It is more efficient and durable but complex to control.

Working Principle:

When DC voltage is applied to the motor terminals, a current flows through the armature winding, producing a magnetic field. This magnetic field interacts with the fixed field of the stator (usually permanent magnets or field coils), resulting in a torque that causes the rotor to spin.

Direction of rotation can be controlled by swapping the polarity of supply voltage as shown in below Fig. 4.12.

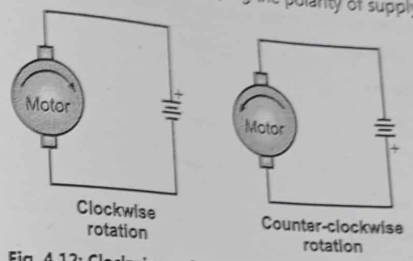

Fig. 4.12: Clockwise and counter-clockwise DC motors

AVR microcontrollers cannot drive motors directly due to limited current sourcing capability. So, motor drivers are used. So, the direction of rotation is controlled using H-bridge drivers. The popular driver IC is L293D/ L298N.

L293D is Dual H-bridge driver, which controls two DC motors. It can handle ~600 mA per channel. It has input pins (IN1, IN2) control motor direction and enable pin (EN1) controls motor speed via PWM.

Bidirectional DC Motor Control using L298N Chip:

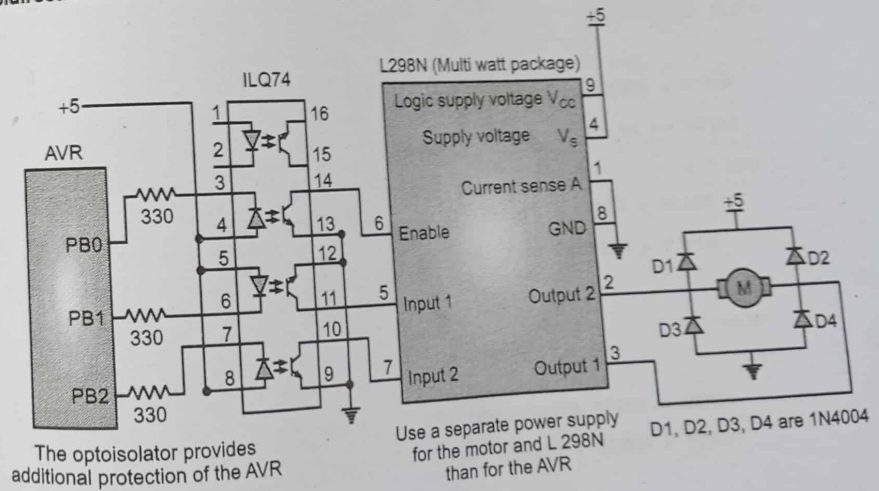

Fig. 4.13: Connection of L298N to AVR

L298N generates heat during operation. So usually, heat sinks are used with L298 chip.

ILQ74 DC is an Input Transistor Output Quad Optocoupler which provides additional protection to the AVR.

AVR C Program:

A switch is connected to PA7. If SW = 0, the DC motor rotates clockwise. If SW = 1, the DC motor rotates counterclockwise.

```
#include <avr/io.h>
#define ENABLE 0
#define MRT_1 1
#define MRT_2 2
#define SW (PINA&0x80)
int main()
{
   DDRA = 0x7F;              //make PA7 input pin
   DDRB = 0xFF;              //make PORTB output pin
   PORTB = PORTB & (~(1<<ENABLE));   1111 1110
   PORTB = PORTB & (~(1<<MTR_1));    1111 1100
   PORTB = PORTB & (~(1<<MTR_2));    1111 1000
   while (1)
   {
      PORTB = PORTB | (1<<ENABLE);   1111 1001
      if(SW == 1)
      {
         PORTB = PORTB | (1<<MRT_1));      //MTR_1 = 1    1111 1011
         PORTB = PORTB | (~(1<<MRT_2);     //MTR_2 = 0    1111 1011
      }
      else
      {
         PORTB = PORTB & (~(1<<MRT_1));    //MTR_1 = 0    1111 1101
         PORTB = PORTB | (1<<MRT_2);       //MTR_2 = 1    1111 1101
      }
   }
   return 0;
}
```


4.6.2 Stepper Motor

- A stepper motor is widely used to translate electrical pulses into mechanical movement.
- Applications of stepper motor are disk drives, dot matrix printers, robotics and position control.
- Steeper motor have a permanent magnet rotor (called as shaft) surrounded by a starter as shown in Fig. 4.14.
- It has 4 stator windings that are paired with a center-tapped common as shown Fig. 4.14.

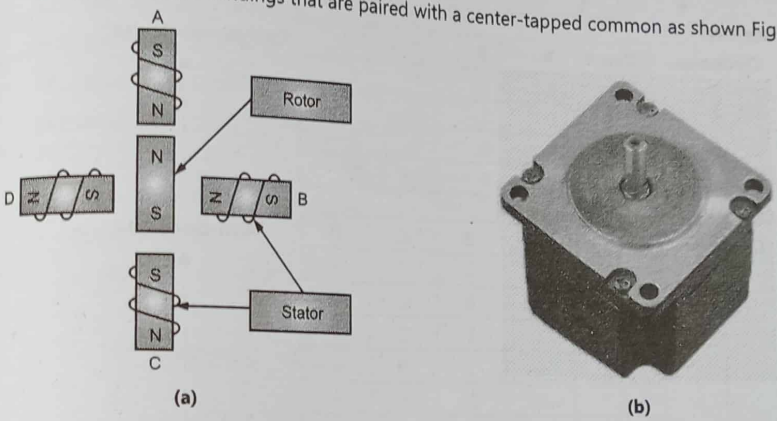

Fig. 4.14: Stepper motor

- The center tap allows a change of current direction.
- When a winding of each of two coils is grounded, it results in a change in polarity of the stator. The stepper motor shaft moves in a precise position.

Step Angle:

- Step angle is defined as 'the minimum degree of rotation with a single step'.

  Number of steps per revolution = 360° / Step angle

  Steps per second = (rpm × Steps per revolution) / 60

  Example: Step angle = 2°

  Number of steps per revolution = 180

Switching Sequence of Motor:

- As discussed earlier, the coils need to be energized for the rotation. This can be done by sending a bit sequence to one end of the coil while the other end is commonly connected.
- The bit sequence sent can make either one phase ON or two phase ON for a full step sequence or it can be a combination of one and two phase ON for half step sequence. Both are tabulated below.

Full Step:

Two Phase ON:

Clockwise ↓

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

↑ Counter-clockwise

One Phase ON

Clockwise ↓

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

↑ Counter-clockwise

The sequence is given in table below.

Clockwise ↓

Step #	A	B	C	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

↑ Anti-clockwise

Interfacing Stepper Motor with AVR:

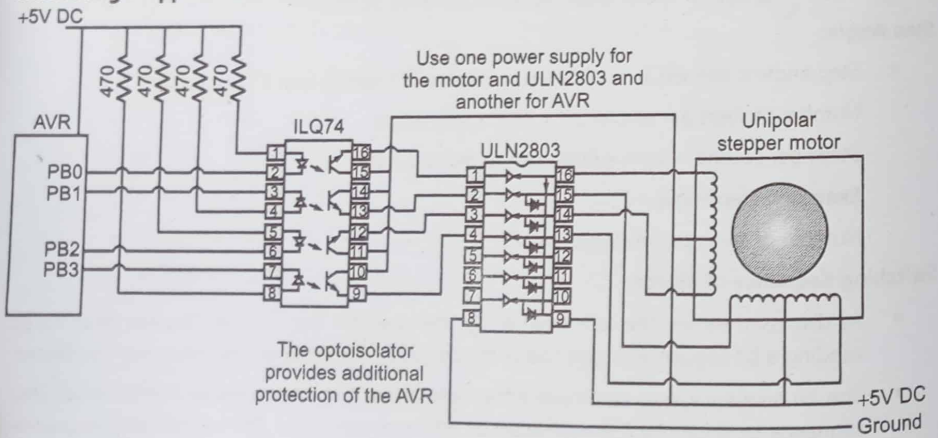

Fig. 4.15: Interfacing of stepper motor with AVR

ILQ74 DC is an Input Transistor Output Quad Optocoupler which provides additional protection to the AVR.

ULN2803 is a high-voltage and high-current Darlington transistor array and consists of eight NPN Darlington pairs which provides the proper current amplification required by the loads.

The steeper motor is connected to the PORTB of AVR.

AVR C Program:

```
#define F_CPU   8000000UL  //XTAL = 8 MHz
#include <avr/io.h>
#include <util/delay.h>
int main()
{
   DDRA = 0x00;      ip
   DDRB = 0xFF;      op
   while(1)
   {
      if(PINA&0x80) == 0)
      {
         PORTB = 0x66;        3
            delay_ms (100);
         PORTB = 0xCC;        2        full step, two phase on
            delay_ms (100);
         PORTB = 0x99;        1
            delay_ms (100);
         PORTB = 0x33;        4
            delay_ms (100);
      }
      else
      {
         PORTB = 0x66;
            delay_ms (100);
         PORTB = 0x33;
            delay_ms (100);
```

```
        PORTB = 0x99;
           delay_ms (100);
        PORTB = 0xCC;
           delay_ms (100);
    }
}
```

4.7 INTERFACING OF RELAY

A relay is an electromagnetic switch used to control high-voltage or high-current devices with a low-voltage signal from a microcontroller. It consists of electromagnet (coil), armature and contacts (Common - COM, Normally Open - NO, Normally Closed - NC). There can be one or more contacts such as SPST (single pole single throw), SPDT (single pole double throw) or DPDT (double pole double throw) as shown in below Fig. 4.16.

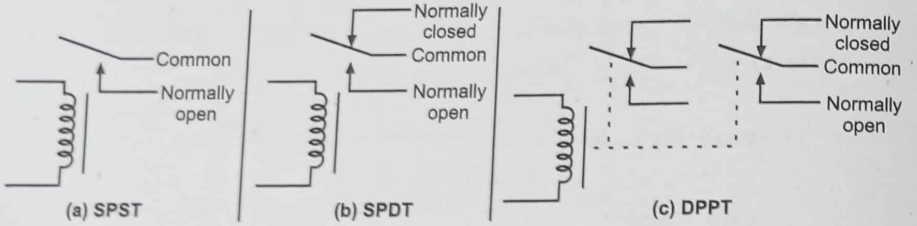

Fig. 4.16: SPST, SPDT and DPDT relays

types of relay:

1. **SPST – Single Pole Single Throw:** It acts like a simple on/off switch, 1 input (pole), 1 output (throw). When coil is energized, circuit closes. It is used in switching a single device ON or OFF.
2. **SPDT – Single Pole Double Throw:** It has 1 input (pole), 2 outputs (throws). It has common terminal (COM) switches between Normally Closed (NC) and Normally Open (NO). It is used to select between two circuits, as one path is always connected.
3. **DPDT – Double Pole Double Throw:** It is like two SPDT relays in one package. It has 2 inputs (poles), each with 2 outputs (throws). It controls two independent circuits simultaneously. It is used to forward/reverse motor control, switching polarity or dual devices.

Working Principle:

When a small voltage is applied to the coil, it creates a magnetic field. The magnetic field pulls the armature, changing the contact position from NC to NO. This allows the relay to switch on/off external devices like bulbs, fans etc.

Interfacing Relay with AVR:

AVR pins cannot provide the current required by the relay coil, so NPN transistor (e.g., BC547 or 2N2222) is used to drive the relay.

As shown in Fig. 4.17, NPN transistor BC547 is used to control the relay. A diode (1N4007/1N4148) is connected across the relay coil to protect the transistor from damage due to the BACK EMF generated in the relay's inductive coil when the transistor is turned OFF. The LED is used to indicate that the RELAY has been turned ON.

Relay is ON when PA0 is set to High. Relay is OFF when PA0 is set to Low.

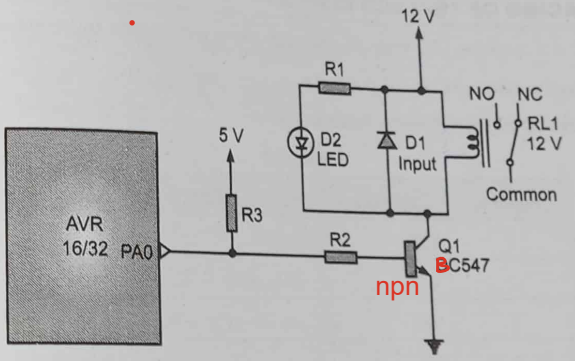

Fig. 4.17: Interfacing of Relay with AVR

AVR C Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#define RELAY_PIN PA0
void relay_on()
{
   PORTA |= (1 << RELAY_PIN);
}
void relay_off()
{
   PORTA &= ~(1 << RELAY_PIN);
}
int main(void)
{
   DDRA |= (1 << RELAY_PIN); // Set PA0 as output
```

1111 1111

1111 1110

0000 0001

```
    while (1)
    {
        relay_on();              // Turn ON relay (device ON)
        _delay_ms(2000);         // Wait for 2 seconds
        relay_off();             // Turn OFF relay (device OFF)
      _delay_ms(2000);           // Wait for 2 seconds
    }
}
```

4.8 INTERFACING OF 16×2 LCD DISPLAY

A 16×2 LCD is a character LCD that can display16 characters per line and there are 2 lines.

It is based on the Hitachi HD44780 controller.

The pinout of LCD is given in below Table 4.3.

Table 4.3

Pin No.	Symbol	Function
1	VSS	Ground
2	VCC	+5V Supply
3	VEE	Contrast control (via pot)
4	RS	Register Select
5	RW	Read/Write (grounded for write)
6	EN	Enable
7-14	D0-D7	Data lines
15	LED +	Backlight +
16	LED –	Backlight –

Key Features:

Table 4.4

Feature	Description
Character Display	5×8 pixel matrix per character
Interface	8-bit or 4-bit data bus
Control Pins	RS, RW, EN
Supply Voltage	Typically 5V
Contrast Control	Via potentiometer (VEE pin)

Interfacing of 16×2 LCD with AVR:

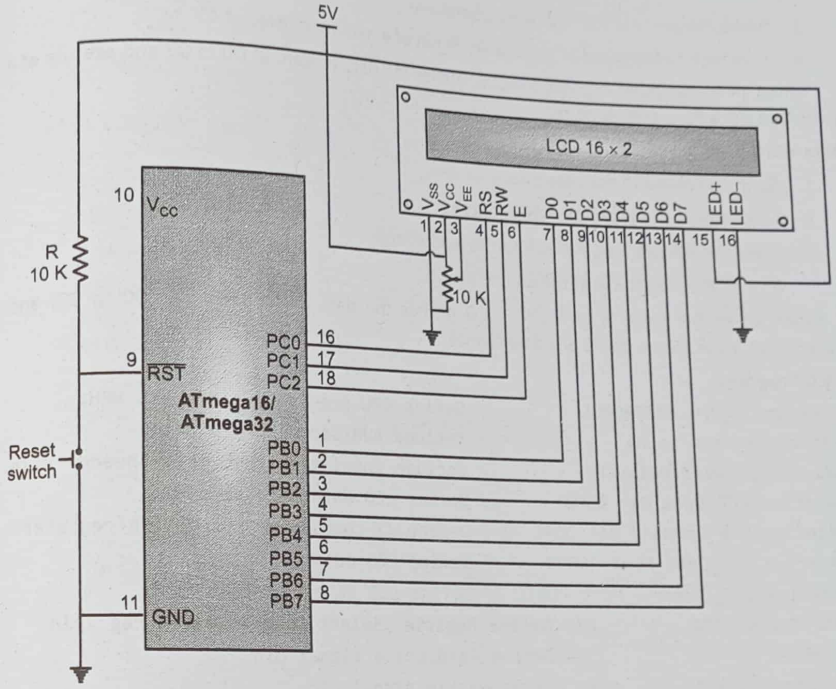

Fig. 4.18: Interfacing of 16×2 LCD with AVR

Data pins D0-D7 are connected to PORTB, RS is connected to PC0, RW to PC1 and E is connected to PC2.

Initializing:

1. Power ON the LCD.
2. Wait for 15 ms ('Power ON' initialization time for LCD16×2).
3. Send 0×38 command to initialize 2 lines, 5×8 matrix, 8-bit mode LCD16×2.
4. Send any 'Display ON' command (0×0E, 0×0C) to LCD16×2.
5. Send 0×06 command (increment cursor) to LCD16×2.

After initializing, LCD is ready to accept data for displaying.

Command Write Function:

1. Send the command value to the LCD16×2 data port.
2. Make RS pin low, RS = 0 (command reg.)

3. Make RW pin low, RW = 0 (write operation)
4. Give high to low pulse at the Enable (E) pin of a minimum delay of 450 ns.

When an enable pulse is given, the LCD latches the data present at D0 to D7 and execute as a command since RS is command reg.

Data Write Function:

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

When an enable pulse is given, the LCD latches the data present (on pins D0 to D7) and displays it on a 5x8 matrix, as RS is a data register.

AVR C Program:

```
#define F_CPU 8000000UL          // Define CPU Frequency e.g. here 8MHz
#include <avr/io.h>              // Include AVR std. library file
#include <util/delay.h>          // Include inbuilt defined Delay header file
#define LCD_Data_Dir DDRB        // Define LCD data port direction
#define LCD_Command_Dir DDRC     // Define LCD command port direction register
#define LCD_Data_Port PORTB      // Define LCD data port
#define LCD_Command_Port PORTC // Define LCD data port
#define RS PC0           // Define Register Select (data/command reg.)pin
#define RW PC1           // Define Read/Write signal pin
#define EN PC2          // Define Enable signal pin
void LCD_Command(unsigned char cmnd)
{
   LCD_Data_Port= cmnd;
   LCD_Command_Port &= ~(1<<RS);       // RS=0 command reg.
   LCD_Command_Port &= ~(1<<RW);       // RW=0 Write operation
   LCD_Command_Port |= (1<<EN);        // Enable pulse
   _delay_us(1);
   LCD_Command_Port &= ~(1<<EN);
   _delay_ms(3);
}
void LCD_Char (unsigned char char_data)    // LCD data write function
{
   LCD_Data_Port= char_data;
   LCD_Command_Port |= (1<<RS);            // RS=1 Data reg.
   LCD_Command_Port &= ~(1<<RW);           // RW=0 write operation
```



```
   LCD_Command_Port |= (1<<EN);
   _delay_us(1);                           // Enable Pulse
   LCD_Command_Port &= ~(1<<EN);
   _delay_ms(1);
}
void LCD_Init (void)                       // LCD Initialize function
{
   LCD_Command_Dir = 0xFF;                 // Make LCD command port direction as o/p
   LCD_Data_Dir = 0xFF;                    // Make LCD data port direction as o/p
   _delay_ms(20);                          // LCD Power ON delay always >15ms
   LCD_Command (0x38);                     // Initialization of 16X2 LCD in 8bit mode
   LCD_Command (0x0C);                     // Display ON Cursor OFF
   LCD_Command (0x06);                     // Auto Increment cursor
   LCD_Command (0x01);                     // Clear display
   LCD_Command (0x80);                     // Cursor at home position
}
void LCD_String (char *str)  // Send string to LCD function
{
   int i;
   for(i=0;str[i]!=0;i++)    // Send each char of string till the NULL
   {
      LCD_Char (str[i]);
   }
}
void LCD_String_xy (char row, char pos, char *str)
                                          // Send string to LCD with xy position
{
   if (row == 0 && pos<16)
   LCD_Command((pos & 0x0F)|0x80);   // Command of first row
   else if (row == 1 && pos<16)
   LCD_Command((pos & 0x0F)|0xC0);   // Command of first row
   LCD_String(str);                  // Call LCD string function
}
void LCD_Clear()
{
   LCD_Command (0x01);               // clear display
   LCD_Command (0x80);               // cursor at home position
}
```

```
int main()
{
   LCD_Init();                          // Initialize LCD
   LCD_String("ElectronicWINGS");       // write string on 1st line of LCD
   LCD_Command(0xC0);                   // Go to 2nd line
   LCD_String("Hello World");           // Write string on 2nd line
   return 0;
}
```

4.9 DAC INTERFACING (WAVEFORM GENERATION USING DAC)

Digital-to-Analog Converter (DAC) is an electronic device that converts digital signals (binary values) into corresponding analog voltages or currents.

AVR microcontrollers are digital devices. It process and output binary signals (0s and 1s). But real-world devices (speakers, motors, sensors, actuators) often require analog signals.

So, DACs are essential for Audio output (e.g., sound generation), Analog control signals (e.g., motor speed, light brightness), Signal generation (waveform generation, sine wave etc.)

There are two types of DACs such as Binary-weighted DAC and R-2R Ladder DAC.

There are 8-bit, 10 and 12-bit DACs available. The number of data bit inputs decides the resolution of DAC. The DAC0808 is an 8-bit DAC which provides 256 discrete voltage (or current) levels of output.

Block diagram of DAC is shown in below Fig. 4.19.

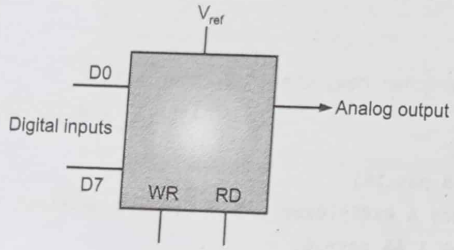

Fig. 4.19: Block diagram of DAC

The DAC0808 converts digital input to current (I_{out}). By connecting a resistor, this current is converted into analog voltage. I_{out} is a function of binary number at the D0-D7 inputs of dAC0808 and the reference current I_{ref} as follow:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

Where D0 is LSB and D7 is MSB of the input, I_{ref} is the input current that must be applied to pin14.

Interfacing of DAC with AVR:

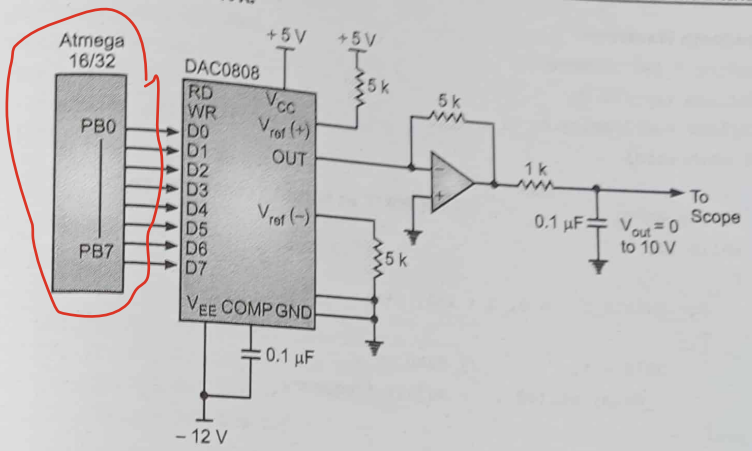

Fig. 4.20: Interfacing of DAC0808 with AVR

As shown in above Fig. 4.20, I to V converter is used at the output of DAC.

AVR C Programs for Waveform Generation using DAC:

Using DAC, various waveforms such as square, triangular, sawtooth and sine can be generated.

To generate waveforms, first send a sequence of 8-bit digital values from AVR to DAC0808. DAC converts digital values to analog current. An Operational Amplifier in I-to-V mode will convert current to analog voltage. And by sending a proper sequence (ramping, sine lookup table), various waveforms can be created.

1. Square Waveform:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRB = 0xFF;        // PORTB connected to DAC0808 (D0-D7) as output
    while (1)
    {
        PORTB = 0xFF;   // Max output voltage
        _delay_ms(500); // High time
        PORTB = 0x00;   // Min output voltage
        _delay_ms(500); // Low time
    }
}
```

2. **Sawtooth Waveform:**

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRB = 0xFF;              // Set PORTC as output (connected to DAC)
    while (1)
    {
        for (uint8_t i = 0; i < 255; i++)
        {
            PORTB = i;        // Ramp up
            _delay_us(100);   // Adjust frequency
        }
    }
}
```

3. **Triangle Waveform:**

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRB = 0xFF;
    while (1)
    {
        for (uint8_t i = 0; i < 255; i++)
        {
            PORTB = i;        // Ascending
            _delay_us(100);
        }
        for (uint8_t i = 255; i > 0; i--)
        {
            PORTB = i;        // Descending
            _delay_us(100);
        }
    }
}
```


4. **Sine Waveform (using Lookup Table):**

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
// 64-point sine wave (values from 0 to 255)
uint8_t sine_table[64] =
{
    128,140,153,165,176,186,195,202,
    208,212,215,216,215,212,208,202,
    195,186,176,165,153,140,128,115,
    102, 90, 79, 69, 60, 53, 47, 43,
    40, 39, 40, 43, 47, 53, 60, 69,
    79, 90,102,115,128,140,153,165,
    176,186,195,202,208,212,215,216,
    215,212,208,202,195,186,176,165
};
int main(void)
{
    DDRB = 0xFF;
    while (1)
    {
        for (uint8_t i = 0; i < 64; i++)
        {
            PORTB = sine_table[i];
            _delay_us(200);       // Adjust frequency
        }
    }
}
```

4.10 CASE STUDIES

Various case studies are discussed in this section:

1. **Traffic Light Controller using AVR:**

The Traffic Light Controller is an embedded system application designed to simulate and control traffic signal operations using a microcontroller. The objective is to automate traffic signal transitions (RED–YELLOW–GREEN) based on predefined timing intervals.

A 4-way automatic traffic light system that controls vehicle flow at an intersection using an AVR microcontroller (e.g., ATmega16/32) can be designed. Each road gets GREEN for a fixed time, followed by YELLOW and RED while others proceed.

Each road will have 3 LEDs, RED, YELLOW, GREEN. So total 12 LEDs (4 roads × 3 LEDs) are required.

Logic Sequence is user defined. Here, the logic sequence is considered as:

Each road gets GREEN for 5 seconds, YELLOW for 2 seconds and RED for rest of the time.

Cycle as:

North → Yellow → South → Yellow → East → Yellow → West → Yellow → Repeat

LEDs are connected as:

Road	Red	Yellow	Green
North	PA0	PA1	PA2
South	PA3	PA4	PA5
East	PB0	PB1	PB2
West	PB3	PB4	PB5

AVR C Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
// NORTH (PA0-PA2), SOUTH (PA3-PA5), EAST (PB0-PB2), WEST (PB3-PB5)
void init_ports()
{
   DDRA = 0x3F; 0011 1111  // PA0-PA5 as output
   DDRB = 0x3F;            // PB0-PB5 as output
   PORTA = 0x00;
   PORTB = 0x00;
}
void all_red()
{
   PORTA = (1 << PA0) | (1 << PA3);     // Red for North & South
   PORTB = (1 << PB0) | (1 << PB3);     // Red for East & West
}                0000 1001  0000 1001
void north_go()
{
   all_red();
   PORTA &= ~(1 << PA0);  // Red off
```



```
   PORTA |= (1 << PA2);   // Green on
   _delay_ms(5000);
   PORTA &= ~(1 << PA2);
   PORTA |= (1 << PA1);   // Yellow on
   _delay_ms(2000);
   PORTA &= ~(1 << PA1);
}
void south_go()
{
   all_red();
   PORTA &= ~(1 << PA3);
   PORTA |= (1 << PA5);
   _delay_ms(5000);
   PORTA &= ~(1 << PA5);
   PORTA |= (1 << PA4);
   _delay_ms(2000);
   PORTA &= ~(1 << PA4);
}
void east_go()
{
   all_red();
   PORTB &= ~(1 << PB0);
   PORTB |= (1 << PB2);
   _delay_ms(5000);
   PORTB &= ~(1 << PB2);
   PORTB |= (1 << PB1);
   _delay_ms(2000);
   PORTB &= ~(1 << PB1);
}
void west_go()
{
   all_red();
   PORTB &= ~(1 << PB3);
   PORTB |= (1 << PB5);
   _delay_ms(5000);
```

```
        PORTB &= ~(1 << PB5);
        PORTB |= (1 << PB4);
        _delay_ms(2000);
        PORTB &= ~(1 << PB4);
    }
    int main(void)
    {
       init_ports();
       while (1)
       {
          north_go();
          south_go();
          east_go();
          west_go();
       }
    }
```

The above traffic signal system will work in sequential manner, divided into four main phases. In Phase 1, the North road is given the Green signal for 5 seconds, allowing vehicles to move, followed by a Yellow signal for 2 seconds to indicate caution before stopping. Phase 2 activates the South road, which similarly receives a Green light for 5 seconds, followed by a Yellow light for 2 seconds. Next, Phase 3 enables the East road with a 5-second Green signal and a 2-second Yellow signal. Finally, in Phase 4, the West road is given the Green signal for 5 seconds, followed by the Yellow signal for 2 seconds. During each phase, all other roads remain on Red to avoid collisions and the cycle repeats continuously to manage traffic in an orderly manner.

2. Single Digit Event Counter using Opto-Interrupter and SSD:

A system that counts discrete physical events (e.g., object passing through a slot) and displays the count (0-9) on a single-digit Seven Segment Display, using an opto-interrupter sensor for detection and AVR microcontroller for control logic is discussed here.

The components required for this system are: IR opto-interrupter, single seven segments display and AVR controller with power supply and pull up registers.

The system will work as follows:

- The opto-interrupter has an IR LED and a phototransistor.
- When an object passes between them, it interrupts the IR beam and the sensor output changes (logic LOW or HIGH).
- The microcontroller detects this transition and increments a counter.
- The count (0-9) is displayed on the Seven Segment Display.

Opto-Interrupter MOC7811 is shown below:

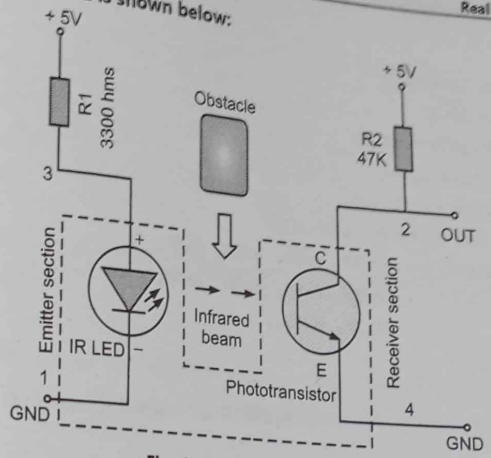

Fig. 4.21: MOC7811 sensor

Source: From data sheet

MOC7811 has VCC (anode) connected to +5V, GND (cathode) connected to GND and Output (collector).

In our system, the output collector is connected to INT0 pin (PD2) of ATmega16/32 with pull-up resistor and seven segments a-g (common cathode) are connected to PORTC PC0-PC6.

SSD Segment Code Map for 0-9 is as follows:

Digit	Segments	Hex Code
0	a–b–c–d–e–f	0×3F
1	b–c	0×06
2	a–b–d–e–g	0×5B
3	a–b–c–d–g	0×4F
4	b–c–f–g	0×66
5	a–c–d–f–g	0×6D
6	a–c–d–e–f–g	0×7D
7	a–b–c	0×07
8	all	0×7F
9	a–b–c–d–f–g	0×6F

AVR C program:

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
volatile uint8_t count = 0;
// SSD digit map (Common Cathode)
uint8_t SSD_digit[] = {0x3F, 0x06, 0x5B, 0x4F,
                       0x66, 0x6D, 0x7D, 0x07,
                       0x7F, 0x6F};
void init_SSD()
{
    DDRC = 0x7F;              // PC0–PC6 as output (segments A–G)
}
void display_digit(uint8_t digit)
{
    if (digit < 10)
        PORTC = SSD_digit[digit];
    else
        PORTC = 0x00;         // Blank
}
void init_interrupt()
{
    DDRD &= ~(1 << PD2);      // INT0 as input
    PORTD |= (1 << PD2);      // Pull-up
    MCUCR |= (1 << ISC01);    // Falling edge INT0
    GICR |= (1 << INT0);      // Enable INT0
    sei();                    // Global interrupt enable
}
ISR(INT0_vect)
{
  _delay_ms(50);              // Debounce
  count++;
  if (count > 9) count = 0; // Wrap around
  display_digit(count);
}
```



```
int main(void)
{
    init_SSD();
    init_interrupt();
    display_digit(0);             // Start with 0
    while (1)
    {
        // Main loop does nothing; interrupt handles counting
    }
}
```

When an object passes through the opto-interrupter, the IR beam is broken. The INT0 interrupt is triggered and the counter increments. The new value is shown on the SSD (0 to 9). After reaching 9, it wraps around to 0.

3. Real Time Clock using IC DS1307 Chip:

A real-time clock system that displays the current time and date using the DS1307 RTC chip, an I2C-based peripheral, interfaced with an AVR microcontroller is discussed here.

DS1307 maintains real-time clock and calendar data even when the MCU is powered off.

It communicates using the I2C protocol. The AVR microcontroller reads data from DS1307 over TWI (I2C) and displays it.

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. It counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week and Year with Leap-Year Compensation valid up to 2100.

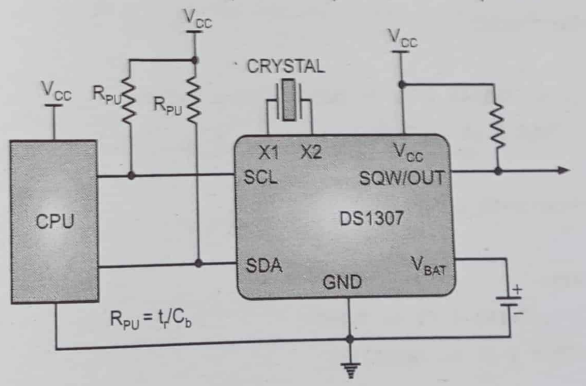

Fig. 4.22: DS1307 real time clock

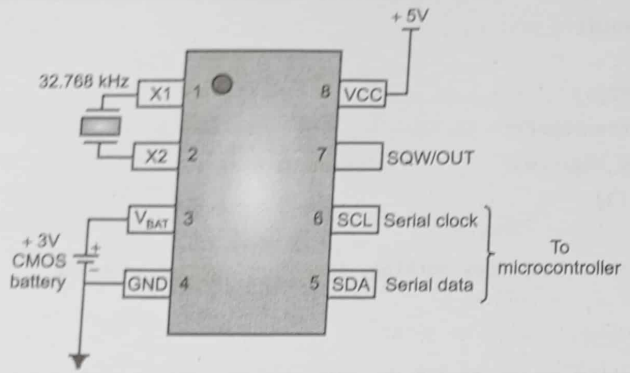

Fig. 4.23: Pin diagram of DS1307

PC0 is connected to SCL (DS1307 pin 6) and PC1 is connected to SDA (DS1307 pin 5).

AVR C Program:

```
// TWI/I2C Initialization
void TWI_init(void)
{
   TWSR = 0x00;
   TWBR = 0x47;            // SCL frequency = F_CPU/(16 + 2*TWBR*Prescaler)
   TWCR = (1 << TWEN);     // Enable TWI
}
// RTC Read/Write Functions
void TWI_start(void)
{
   TWCR = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);
   while (!(TWCR & (1 << TWINT)));
}
void TWI_write(uint8_t data)
{
   TWDR = data;
   TWCR = (1 << TWEN) | (1 << TWINT);
   while (!(TWCR & (1 << TWINT)));
}
```



```
uint8_t TWI_read_ack(void)
{
   TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);
   while (!(TWCR & (1 << TWINT)));
   return TWDR;
}
uint8_t TWI_read_nack(void)
{
   TWCR = (1 << TWEN) | (1 << TWINT);
   while (!(TWCR & (1 << TWINT)));
   return TWDR;
}
void TWI_stop(void)
{
   TWCR = (1 << TWSTO) | (1 << TWINT) | (1 << TWEN);
}
// Convert BCD to Decimal
uint8_t BCD_to_decimal(uint8_t bcd)
{
   return ((bcd >> 4) * 10) + (bcd & 0x0F);
}
// Read Time and Display on LCD if connected to AVR
void RTC_read_time(uint8_t *hour, uint8_t *min, uint8_t *sec)
{
   TWI_start();
   TWI_write(0xD0);    // DS1307 address + Write
   TWI_write(0x00);    // Set register pointer to 00
   TWI_start();
   TWI_write(0xD1);    // DS1307 address + Read
   *sec = BCD_to_decimal(TWI_read_ack());
   *min = BCD_to_decimal(TWI_read_ack());
   *hour = BCD_to_decimal(TWI_read_nack());
   TWI_stop();
}
```

For example, LCD output can be

Time: 12:45:38

Date: 11-06-2025

DS1307 maintains time and date in real-time. AVR continuously reads values via I2C and updates the LCD. Time remains intact even if the main power is lost due to CMOS battery.

4. Temperature Monitoring System using LM35 Sensor:

A simple and accurate temperature monitoring system using an LM35 analog temperature sensor interfaced with an AVR microcontroller is discussed here. LCD (16x2) display can be used to display the temperature.

The LM35 sensor outputs a voltage directly proportional to the temperature (10 mV per °C).

The AVR microcontroller uses its internal ADC (Analog-to-Digital Converter) to read the voltage. The value is converted into temperature in °C and displayed on an LCD.

LM35 is three terminal linear temperature sensor from National semiconductors. It can measure temperature from −550°C to +1500°C.

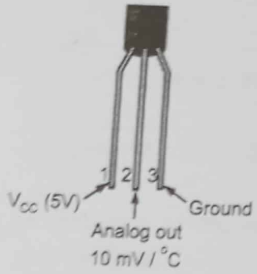

Fig. 4.24: LM35 temperature sensor

LM 35 is a 3-pin device, V_{cc} (+5V), GND and V_{out} (Analog voltage output = 10 mV/°C).

So, at 25°C, output voltage = 250 mV

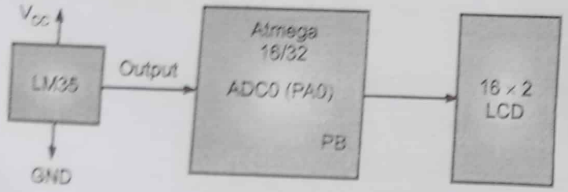

Fig. 4.25: Interfacing of LM35 with AVR

RS, E, D4-D7 are connected to PORTB of AVR.

AVR C Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include "lcd.h"          // Assume LCD library available
void ADC_init()
{
    ADMUX = (1 << REFS0); // AVCC as reference, ADC0 channel
    ADCSRA = (1 << ADEN) | (1 << ADPS2);      // Enable ADC, prescaler 16
}
uint16_t ADC_read(uint8_t channel)
{
   ADMUX = (ADMUX & 0xF0) | (channel & 0x0F); // Select ADC channel
   ADCSRA |= (1 << ADSC);                     // Start conversion
   while (ADCSRA & (1 << ADSC));              // Wait for conversion
   return ADC;
}
int main(void)
{
   char temp_str[16];
   uint16_t adc_value;
   float temperature;
   LCD_init();
   ADC_init();
   LCD_cmd(0x80);
   LCD_print("Temp: ");
   while (1)
   {
      adc_value = ADC_read(0);  // Read from ADC0 (LM35)
      temperature = adc_value * 5.0 * 100 / 1024;  // 10mV/°C
      sprintf(temp_str, "%.2f C   ", temperature);
      LCD_cmd(0x86); // Move to position
      LCD_print(temp_str);
      _delay_ms(1000);
   }
}
```

AVR has 10-bit in built ADC.

ADC resolution is 10-bit, that is 1024 steps for 0-5V.

Voltage per step = $\frac{5\ V}{1024} \approx 4.88\ mV$

LM35 output = 10 mV/°C

So, temperature (°C) = $\frac{(ADC\_value \times 5.0 \times 100)}{1024}$

5. Smart Phone Controlled Devices using Bluetooth Module HC-05:

A system that allows a user to control electrical devices (e.g., LEDs, fan, motor) via Bluetooth communication from a smartphone, using the HC-05 module and an AVR microcontroller (ATmega16/32) is discussed here.

In this system, a smartphone app (e.g., Bluetooth Terminal or custom Android app) sends commands over Bluetooth. The HC-05 Bluetooth module receives these commands and sends them to the AVR via UART. The AVR interprets the commands and switches devices ON or OFF accordingly.

The HC-05 module is a Bluetooth module and uses serial communication. It can be operated within 4-6 V of power supply and it supports baud rate of 9600, 19200, 38400, 57600 etc. It will neither send or receive data from external sources, as it can be operated in Master-Slave mode. It uses 2.45 GHz frequency band and the transfer rate of the data can vary up to 1Mbps and is in range of 10 meters.

The HC-05 module available in market is shown in below Fig. 4.26.

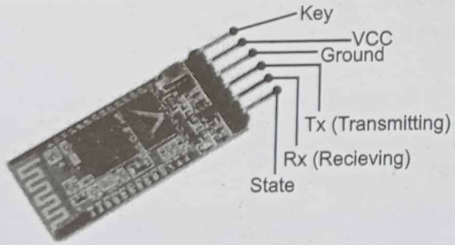

Fig. 4.26: HC-05 Bluetooth module

Source: From data sheet

VCC is connected to +5V power supply. Ground is connected to ground of power supply.

Tx (Transmitter) pin transmits the received data serially and is connected to RXD of AVR (PD0).

Rx (Receiver) is used for broadcasting data serially over Bluetooth. The State pin is used to check if the Bluetooth is working properly or not. The Key pin is optional for AT commands.

Rx pin is connected to TXD of AVR (PD1) via 1k–2kΩ resistor divider.

Smartphone sends command, e.g., 'A' to turn ON LED, 'a' to turn it OFF. AVR receives the character via UART and switches device state accordingly (turns ON/OFF LED or relay). Then system responds with feedback.

AVR C Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
void USART_init(unsigned int ubrr)
{
   UBRRH = (unsigned char)(ubrr >> 8);
   UBRRL = (unsigned char)ubrr;
   UCSRB = (1 << RXEN) | (1 << TXEN);    // Enable receiver and transmitter
   UCSRC = (1 << URSEL) | (3 << UCSZ0); // 8-bit data, 1 stop bit
}
char USART_receive()
{
   while (!(UCSRA & (1 << RXC)));
   return UDR;
}
void USART_transmit(char data)
{
   while (!(UCSRA & (1 << UDRE)));
   UDR = data;
}
int main(void)
{
   DDRC |= (1 << PC0); // PC0 as output to control device (e.g., LED or
                       relay)
   USART_init(6);      // Baud rate 9600 for 1 MHz: UBRR = (F_CPU/16/baud)
                       - 1 = 6

   while (1)
   {
      char command = USART_receive();
```

```
        if (command == 'A')
        {
            PORTC |= (1 << PC0);      // Turn ON device
            USART_transmit('1');      // Optional feedback
        } else if (command == 'a')
        {
            PORTC &= ~(1 << PC0);     // Turn OFF device
            USART_transmit('0');      // Optional feedback
        }
    }
}
```

Command 'A' will take the action of turning ON device 1. Command 'a' will turn OFF the device 1. (Use "Bluetooth Terminal" app or a custom Android app with buttons to send these characters).

Exercise

[I] Multiple Choice Questions:

1. Which pin of AVR is commonly used to read input from a push button?
   (a) VCC (b) GND
   (c) PORTC as output (d) Any PORT pin configured as input
2. What is the function of a current limiting resistor in an LED circuit?
   (a) Increase brightness
   (b) Store energy
   (c) Prevent excess current and protect the LED
   (d) Short the circuit
3. A seven-segment display with common cathode requires:
   (a) Negative logic on segment pins (b) Logic HIGH on segment pins to turn on
   (c) PWM signal on each pin (d) No power supply
4. What is the purpose of a thumbwheel switch in embedded systems?
   (a) Set digital values manually (b) Control analog devices
   (c) Measure temperature (d) Trigger interrupts
5. To rotate a stepper motor precisely in small steps, which technique is used?
   (a) PWM (b) Sequence energizing of coils
   (c) Interrupt disabling (d) Full wave rectification
6. Which type of signal is used to activate a relay using a microcontroller?
   (a) Analog sine wave (b) Digital HIGH/LOW output
   (c) RF signal (d) PWM only

7. In a 16×2 LCD, what does '16×2' represent?
   (a) 16 characters total (b) 16 characters per line, 2 lines
   (c) 16-bit data, 2-bit control (d) 16 pins and 2 rows of LEDs
8. Which pin is used to select command/data mode in 16×2 LCD?
   (a) E (b) RS
   (c) R/W (d) VSS
9. DAC0808 gives output in:
   (a) Digital format (b) Analog voltage/current
   (c) PWM only (d) ASCII value
10. What type of waveform can be generated using a DAC and microcontroller?
    (a) Only digital pulse (b) Sine, triangular, square waveforms
    (c) Thermal noise (d) RF waves
11. In a 4-way traffic controller using AVR, how many total LEDs are typically needed?
    (a) 4 (b) 6
    (c) 12 (d) 8
12. In the event counter using opto-interrupter, which AVR pin is ideal to connect the sensor output?
    (a) INT0 (PD2) (b) ADC0
    (c) PC7 (d) TXD
13. What is the function of the DS1307 chip?
    (a) Serial communication (b) Voltage regulation
    (c) Real-time clock keeping (d) Analog to digital conversion
14. The LM35 sensor outputs:
    (a) PWM signal
    (b) Digital logic
    (c) Analog voltage proportional to temperature
    (d) Serial UART data
15. What is the resolution of the ADC in AVR microcontrollers like ATmega16?
    (a) 8-bit (b) 10-bit
    (c) 12-bit (d) 16-bit
16. In a smartphone-controlled AVR device using HC-05, the AVR communicates with the module using:
    (a) SPI (b) UART
    (c) I2C (d) ADC
17. What is the typical baud rate used to communicate with an HC-05 module?
    (a) 9600 (b) 4800
    (c) 115200 (d) 1200

18. Which protocol is used by DS1307 to communicate with AVR?
    (a) UART (b) I2C
    (c) SPI (d) USB
19. In temperature monitoring systems, the ADC value must be multiplied with which factor to convert to °C (LM35)?
    (a) 5 (b) 1000
    (c) 500 / 1024 or approx. 0.488 (d) 10
20. Which of the following combinations is best suited for a real-time environmental monitor system?
    (a) LM35 + DS1307 + LCD + AVR (b) HC-05 + Relay + Thumbwheel
    (c) SSD + Buzzer + Stepper (d) DAC + Opto-interrupter + Button

Answers

1. (d)	2. (c)	3. (b)	4. (a)	5. (b)	6. (b)	7. (b)	8. (b)	9. (b)	10. (b)
11. (c)	12. (a)	13. (c)	14. (c)	15. (b)	16. (b)	17. (a)	18. (b)	19. (c)	20. (a)

[II] State True or False:

1. A push button can be used as a digital input to an AVR microcontroller.
2. A buzzer requires analog voltage from a DAC to produce sound.
3. Common cathode seven segment displays require logic HIGH to turn on segments.
4. A thumbwheel switch provides analog voltage levels as output.
5. A stepper motor rotates continuously when a single coil is energized.
6. Relays can be directly driven from microcontroller output pins without a transistor.
7. The RS pin of a 16×2 LCD selects between command and data mode.
8. DAC interfacing is used to generate analog waveforms like sine or square waves.
9. In a traffic light controller using AVR, all roads can show green light at the same time.
10. An opto-interrupter can be used to count objects passing through a path.
11. DS1307 communicates with AVR using the SPI protocol.
12. DS1307 continues to track time even when the main power is disconnected.
13. LM35 provides a digital output proportional to the ambient temperature.
14. ADC in AVR is used to convert LM35's analog signal to digital form.
15. HC-05 Bluetooth module can be interfaced with AVR using UART communication.
16. The baud rate commonly used with HC-05 is 9600 bps.
17. A DAC is used to convert digital data into an analog voltage or current.

18. LCD interfacing requires only one data pin to display characters.
19. DC motors can be controlled using PWM signals from a microcontroller.
20. A single-digit event counter system cannot be built using an SSD and opto-interrupter.

Answers

1. True	2. False	3. True	4. False	5. False
6. False	7. True	8. True	9. False	10. True
11. False	12. True	13. False	14. True	15. True
16. True	17. True	18. False	19. True	20. False

[III] Fill in the Blanks:

1. A/an _____ is used to give visual output and can be turned ON/OFF using a digital signal from a microcontroller.
2. A _____ is used to give user input to a microcontroller by making or breaking an electrical contact.
3. A _____ produces sound and is typically activated by a digital HIGH signal.
4. A seven segment display is used to display _____ digits and characters.
5. A _____ switch is a mechanical rotary switch that outputs a fixed binary value set by the user.
6. A _____ motor moves in discrete steps and is used in precise positioning applications.
7. A _____ motor rotates continuously and can be controlled using PWM signals.
8. A _____ is an electromechanical switch controlled by a digital signal, often used to control high-voltage devices.
9. The RS pin in a 16×2 LCD is used to select between _____ and data modes.
10. DAC stands for _____ to _____ converter.
11. In DAC interfacing, _____ waveforms such as sine, square, and triangular can be generated.
12. In the traffic light controller case study, each direction is given a green signal for a fixed duration, followed by a _____ signal.
13. An opto-interrupter is used in an event counter to detect _____ of an object.
14. The IC _____ is a Real-Time Clock chip that keeps track of time even during power loss.
15. The DS1307 communicates with AVR using the _____ protocol.
16. The LM35 sensor gives an analog output of _____ mV per °C of temperature.

17. The LM35 sensor's analog output is converted to digital using the _____ of the AVR.
18. The HC-05 Bluetooth module communicates with the AVR using the _____ protocol.
19. A smartphone can be used to control AVR-based devices wirelessly via _____ communication.
20. In the event counter system, the count value is displayed on a _____.

Answers

1. LED	2. Push button
3. Buzzer	4. Numeric
5. Thumbwheel	6. Stepper
7. DC	8. Relay
9. Command	10. Digital to analog
11. Analog	12. Yellow
13. Interruption or movement	14. DS1307
15. I2C	16. 10
17. ADC	18. UART
19. Bluetooth	20. Seven segment display

[IV] Short Answer Questions:

1. What is the function of a current limiting resistor in an LED circuit?
2. How does a push button work when connected to a microcontroller input pin?
3. What are the typical applications of a buzzer in embedded systems?
4. Differentiate between a common cathode and common anode seven segment display.
5. What is a thumbwheel switch and where is it used?
6. How is the direction and speed of a DC motor controlled using AVR?
7. What is the principle of stepper motor operation? How does it differ from a DC motor?
8. Why is a transistor used to drive a relay with a microcontroller?
9. List the control and data pins required to interface a 16×2 LCD with an AVR microcontroller.
10. What is the purpose of using a DAC in an embedded system?
11. Name three types of waveforms that can be generated using DAC interfacing.
12. Explain the working of a traffic light controller using AVR for a 4-way junction.
13. How does an opto-interrupter work in a single-digit event counter?
14. What is the function of the DS1307 IC in an embedded system?

15. How does the DS1307 maintain time during a power failure?
16. Why is the I2C protocol suitable for interfacing RTC chips like DS1307?
17. How does the LM35 sensor convert temperature to a measurable electrical signal?
18. What role does the ADC play in a temperature monitoring system using LM35?
19. Describe the communication process between the HC-05 Bluetooth module and the AVR.
20. Give one application example of a smartphone-controlled AVR system using the HC-05 module.

[V] Long Answer Questions:

1. Explain the interfacing of an LED and a push button with an AVR microcontroller. Write the C program to toggle the LED using the push button. 2
2. Explain the interfacing and working of a seven-segment display (SSD) with AVR. Write a C program to display the digits from 0 to 9. 3
3. What is a thumbwheel switch? Describe how to interface a BCD thumbwheel switch with an AVR microcontroller. Write its C program. 4
4. Explain the DC and stepper motor with the help of diagram. 6
5. Draw the diagram of interfacing of DC motor with AVR and write a C program to rotate the motor. 6
6. Draw the diagram of interfacing of stepper motor with AVR and write a C program to rotate the motor clockwise. 8
7. Explain how a relay is interfaced with an AVR microcontroller. Write a C program for the same. 9
8. Explain various types of relays. 9
9. Explain how a 16×2 LCD is interfaced with an AVR microcontroller. Describe the function of key pins (RS, RW, E, Data) and provide a flowchart for displaying characters. 10
10. Write C program for interfacing the LCD with AVR 11
11. What is a DAC? Describe how a DAC0808 can be interfaced with an AVR microcontroller to generate analog waveforms. Write C programs to generate sine, square and triangular waveform. 12
12. Design a traffic light controller for a 4-way road intersection using an AVR microcontroller. Describe the logic and write a C program for the same. 13
13. Explain the case study of single digit event counter using opto-interrupter and SSD including the AVR C program.
14. Discuss the design and operation of a real-time clock system using the DS1307 RTC chip and an AVR microcontroller. Explain I2C communication protocol and how time/date is read and displayed.

15. Describe how the LM35 temperature sensor is interfaced with an AVR microcontroller. Explain how the analog signal is converted using ADC and displayed on an LCD. Write a temperature calculation formula. Write a AVR C program for the same.
16. Explain how a smartphone can be used to control electrical devices via an AVR microcontroller and HC-05 Bluetooth module. Write AVR C program for to control the device.
17. Explain the working of stepper motor. 7
18. Explain common cathode and common anode seven segment displays. 3
19. Write a AVR C program to blink LED. 1 or 2
20. Explain types of DC motor. With the help of diagram, explain how DC motor can be rotated clockwise and anticlockwise. 6

■■■