OUTLINE

- GPIO
- Seven Segment display Interface
- Stepper Motor Interface
- Logic Controller Interface
- LCD Interface
- DAC Interface
- LED
- Push Button Switch
- Keyboard Interface

GPIO

LPC2148 Microcontroller:

- It is a 32-bit microcontroller manufactured by NXP semiconductor and is one of the widely used ARM7 based microcontroller.
- It has on-chip SRAM of 32 KB and on-chip Flash memory of 512 KB.
- It has two I/O ports namely Port0 and Port1.
    - These ports are of 32 bit wide and are provided by 64 pins of the microcontroller.
    - Out of 32 pins of Port0, 28 pins can be used as GPIO (General Purpose bidirectional I/O) pins.
        - Pin 31 is an Output only pin.
        - Pins 24, 26, 27 are not available.
    - Out of 32 pins of Port1, 16 pins (pin16-pin31) are available as GPIO pins.
        - Pins 0-15 are not available.
- The default function of pins of Port0 and Port1 is GPIO. But, it is a good programming practice to mention 'PINSEL0=0' and 'PINSEL1=0' for Port0 and Port1 respectively in order to select GPIO function of the pins.
- GPIO function is the most frequently used functionality of the microcontroller. The GPIO function in both the ports is controlled by a set of 4 registers: **IODIR, IOSET, IOCLR, IOPIN**.

IODIR: It is a GPIO port direction control register and is used to set the direction i.e., either input or output of individual pins.

- When a bit in this register is set to 0, the corresponding port pin in the microcontroller is configured as input.
- When a bit in this register is set to 1, the corresponding port pin in the microcontroller is configured as output.

IOSET: It is a GPIO port output set register and can be used to set the value of a GPIO pin that is configured as output to 1 (High).

- When a bit in this register is set to 1, the corresponding port pin is set to logic 1.
- Setting a bit 0 in this register has no effect on the pin.

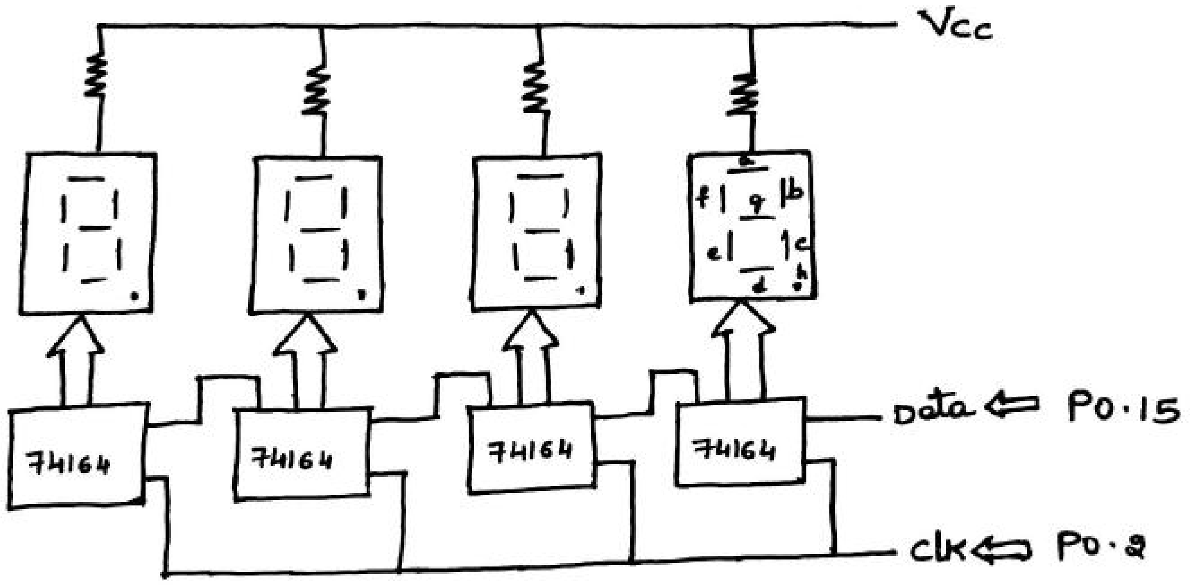
IOCLR: It is a GPIO port output clear register and can be used to set the value of a GPIO pin that is configured as output to 0 (Low).

- When a bit in this register is set to 1, the corresponding port pin is set to logic 0 and at the same time clears the corresponding bit in IOSET register.
- Setting a bit 0 in this register has no effect on the pin.

IOPIN: It is a GPIO pin value register and can be used to read or write values directly to the pin. The status of the pins that are configured as GPIO can always be read from this register irrespective of direction set for the pin (Input or Output).

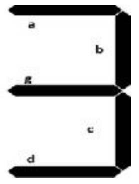
Seven Segment Display Interface

The Seven segment display is an output device for displaying alpha numeric characters. It contains 8 LED segments arranged in a special form. Out of the 8 LED segments, 7 are used for displaying alpha numeric characters and 1 is used for representing 'decimal point' in decimal number display. The Seven segments LED displays are available in two different configurations, namely: Common Anode and common Cathode.



The figure shows a four digit seven segment display interface. It is of common anode type i.e., to make the corresponding segment glow, bit value 0 is to be sent and to turn off any particular segment, bit value 1 is to be sent.

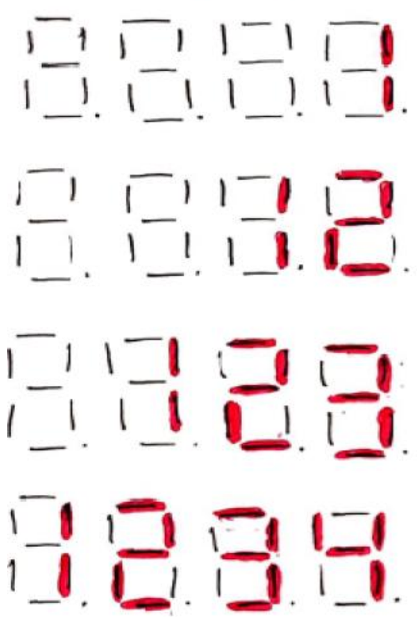
Below table shows the construction of seven segment code to display 3 on the seven segment display interface.



h	g	F	e	d	c	b	a	
1	0	1	1	0	0	0	0	**0xB0**

A serial in parallel out shift register is used to send 8 bit of data to seven segment display. Always '*h*' bit is sent first, then '*g*' bit and so on. The data bits are sent through Port P0.15 pin. Clock pulses are required to clock in the data. 8 clock pulses are required to display one byte of data. As the shift registers are cascaded, 8*4=32 clocks are required to clock in 4 bytes of data. To send “1234”, first we have to send seven segment code of ‘1’, then ‘2’,‘3’ and lastly ‘4’. Common clock is applied through Port P0.2 pin.

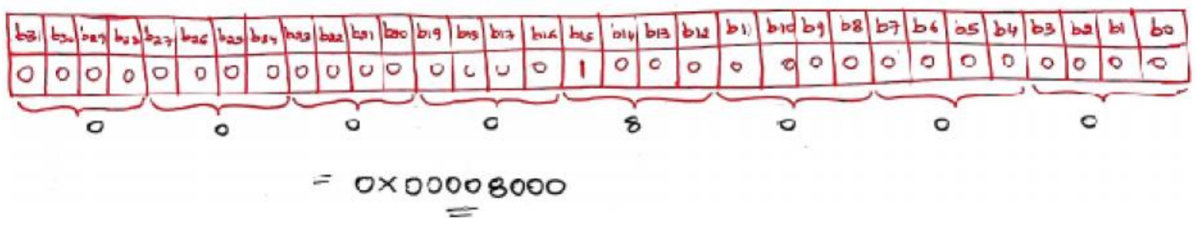
To display "1234"



```
// 7-segment code for 1234
unsigned char data[]={0xF9,
        0xA4, 0xB0, 0x99}
```

PROGRAM1: ***Develop an embedded C program to display digits 0 to F using Seven segment display interface.***

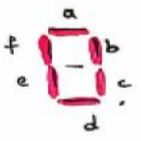
```
#include<lpc214x.h>
void writeSeg(unsigned char x)
{
         unsigned char i;
        for(i=0;i<8;i++)
        {
                if(x & (0x80 >> i))
                        IOSET0 = 0X00008000;  // Send 1 via P0.15
                else
                        IOCLR0 = 0X00008000;  // Send 0 via P0.15

                //Send high to low pulse via P0.2
                IOSET0 = 0X00000004;     //Send 1 via P0.2
                IOCLR0 = 0X00000004;     //Send 0 via P0.2
        }
}
```



b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				0				0				0				8				0				0				0			

= 0X00008000

```
void delay(unsigned int x)
{
        unsigned char i;
        while(x--)
            for(i=0;i<250;i++);
}

int main()
{
        unsigned char data[]={0xff,0xff,0xff,0xff,
                                    0xc0,0xf9,0xa4,0xb0,
                                    0x99,0x92,0x82,0xf8,
                                    0x80,0x98,0x88,0x80,
                                    0xc6,0xc0,0x86,0x8e,
                                    0xff,0xff,0xff};
        unsigned char i,j;

        PINSEL0 = 0X00000000;     //Configure P0 as GPIO port
        IODIR0 = 0X00008004;      // Set P0.15 and P0.2 as Output pins
        while(1)
       {
              for(i=0;i<20;i++)
            {
                     for(j=i;j<i+4;j++)
                           writeSeg(data[j]);
                    delay(10000);
            }
       }
       return 0;
}
```



```
h g f e  d c b a
1 1 0 0  0 0 0 0  = C0H
```

x = C0H

① i = 0   0 < 8

```
        h g f e  d c b a
        1 1 0 0  0 0 0 0
(AND) & 1 0 0 0  0 0 0 0
        ---------------
        1 0 0 0  0 0 0 0
```

Since h bit is 1 send 1 via P0.15

② i = 1   1 < 8

```
  h g f e  d c b a
  1 1 0 0  0 0 0 0
& 0 1 0 0  0 0 0 0
  ---------------
  0 1 0 0  0 0 0 0
```

Since g bit is 1 send 1 via P0.15

③ i = 2   2 < 8

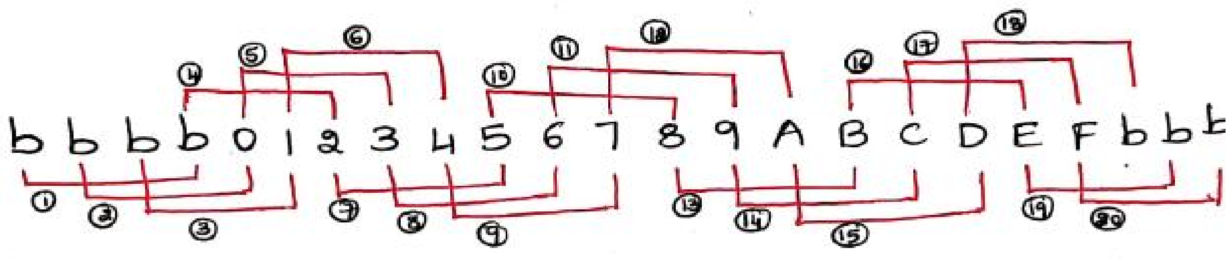
```
  h g f e  d c b a
  1 1 0 0  0 0 0 0
& 0 0 1 0  0 0 0 0
  ---------------
  0 0 0 0  0 0 0 0
```

Since f bit is 0 send 0 via P0.15

④ i = 3   3 < 8

```
  h g f e  d c b a
  1 1 0 0  0 0 0 0
& 0 0 0 1  0 0 0 0
  ---------------
  0 0 0 0  0 0 0 0
```

Since e bit is 0 send 0 via P0.15



bbbb	6789
bbb0	789A
bb01	89AB
b012	9ABC
0123	ABCD
1234	BCDE
2345	CDEF
3456	DEFb
4567	EFbb
5678	Fbbb

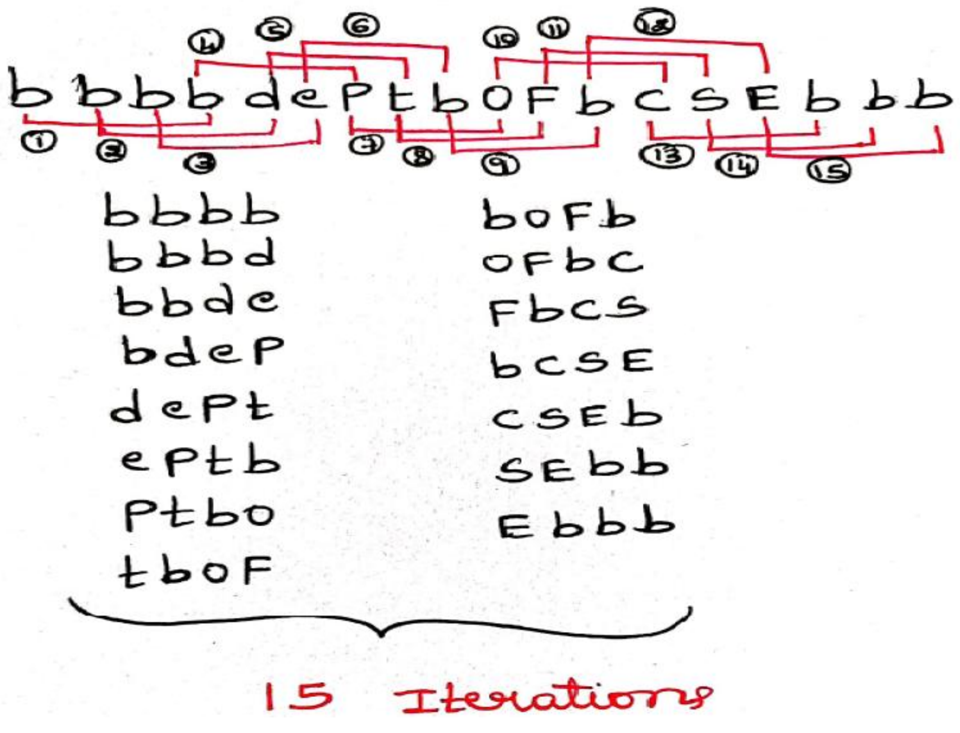
$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}$

20 Iterations

PROGRAM2: ***Develop an embedded C program to display the*** **message “dEPt OF CSE” in a rolling fashion on** ***a four digit Seven segment display interface.***

```
#include<lpc214x.h>
void writeSeg(unsigned char x)
{
         unsigned char i;
        for(i=0;i<8;i++)
        {
                if(x & (0x80 >> i))
                      IOSET0 = 0X00008000;  // Send 1 via P0.15
                else
                      IOCLR0 = 0X00008000;  // Send 0 via P0.15


                //Send high to low pulse via P0.2
                IOSET0 = 0X00000004;     //Send 1 via P0.2
               IOCLR0 = 0X00000004;     //Send 0 via P0.2
        }
}
```

```
void delay(unsigned int x)
{
        unsigned char i;
        while(x--)
            for(i=0;i<250;i++);
}

int main()
{
       unsigned char data[]={0xff,0xff,0xff,0xff,0xA1,
                                  0x86,0x8c,0x87,0xff,0xc0,
                                  0x8e,0xff,0xc6,0x92,0x86,
                                  0xff,0xff,0xff};
        unsigned char i,j;

        PINSEL0 = 0X00000000;     //Configure P0 as GPIO port
        IODIR0 = 0X00008004;      // Set P0.15 and P0.2 as Output pins
        while(1)
       {
             for(i=0;i<15;i++)
           {
                   for(j=i;j<i+4;j++)
                         writeSeg(data[j]);
                  delay(10000);
           }
       }
       return 0;
}
```



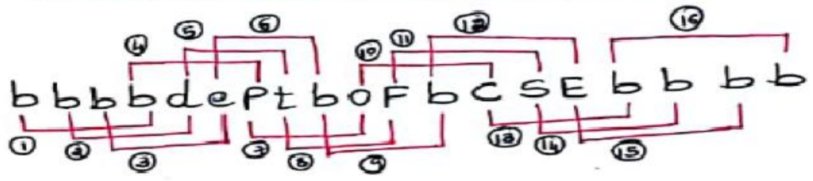
PROGRAM3: ***Develop an embedded C program to display the messages LIFE and HELP alternately on a 4-digit Seven-segment display Interface.***

```
#include<lpc214x.h>
void writeSeg(unsigned char x)
{
         unsigned char i;
        for(i=0;i<8;i++)
        {
                if(x & (0x80 >> i))
                      IOSET0 = 0X00008000;  // Send 1 via P0.15
                else
                      IOCLR0 = 0X00008000;  // Send 0 via P0.15
                //Send high to low pulse via P0.2
                IOSET0 = 0X00000004;     //Send 1 via P0.2
                IOCLR0 = 0X00000004;     //Send 0 via P0.2
        }
}

void delay(unsigned int x)
{
       unsigned char i;
       while(x--)
           for(i=0;i<250;i++);
}

int main()
{
      unsigned char LIFE[4]={0XC7,0xCF,0x8E,0x86};
      unsigned char HELP[4]={0x89,0x86,0xC7,0x8C};
       unsigned char i;

       PINSEL0 = 0X00000000;     //Configure P0 as GPIO port
       IODIR0 = 0X00008004;      // Set P0.15 and P0.2 as Output pins
       while(1)
      {
            for(i=0;i<4;i++)
                 writeSeg(LIFE[i]);
           delay(10000);

            for(i=0;i<4;i++)
                 writeSeg(HELP[i]);
           delay(10000);
      }
      return 0;
}
```

PROGRAM4: ***Develop an embedded C program to display the* message "dEPt OF CSE" from right to left and left to right on *a four digit Seven segment display interface.***

```
#include<lpc214x.h>

void writeSeg(unsigned char x)
{
        unsigned char i;
       for(i=0;i<8;i++)
       {
              if(x & (0x80 >> i))
                    IOSET0 = 0X00008000;  // Send 1 via P0.15
              else
                    IOCLR0 = 0X00008000;  // Send 0 via P0.15
              //Send high to low pulse via P0.2
              IOSET0 = 0X00000004;     //Send 1 via P0.2
              IOCLR0 = 0X00000004;     //Send 0 via P0.2
       }
}

void delay(unsigned int x)
{
      unsigned char i;

      while(x--)
         for(i=0;i<250;i++);
}

int main()
{
     unsigned char data[]={0xff,0xff,0xff,0xff,0xA1,
                           0x86,0x8c,0x87,0xff,0xc0,
                           0x8e,0xff,0xc6,0x92,0x86,
                           0xff,0xff,0xff,0xff};
      char i,j;

      PINSEL0 = 0X00000000;     //Configure P0 as GPIO port
      IODIR0 = 0X00008004;      // Set P0.15 and P0.2 as Output pins

      while(1)
     {
          for(i=0;i<16;i++)            //Right to Left
        {
                for(j=i;j<i+4;j++)
                     writeSeg(data[j]);
               delay(10000);
        }
        for(i=14;i>=0;i--)                //Left to Right
        {
                for(j=i;j<i+4;j++)
                    writeSeg(data[j]);
                delay(10000);
        }

    }
    return 0;
}
```

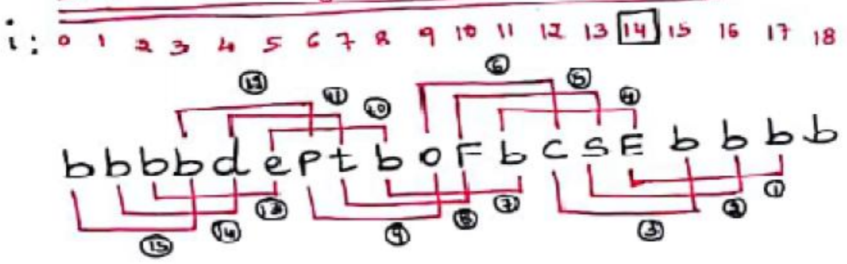
Right to Left Simulation



bbbb
bbbd
bbde
bdeP
dePt
ePtb
Ptbo
tboF
boFb
oFbc
FbcS
bcSE
cSEb
SEbb
Ebbb
bbbb

16 Iterations

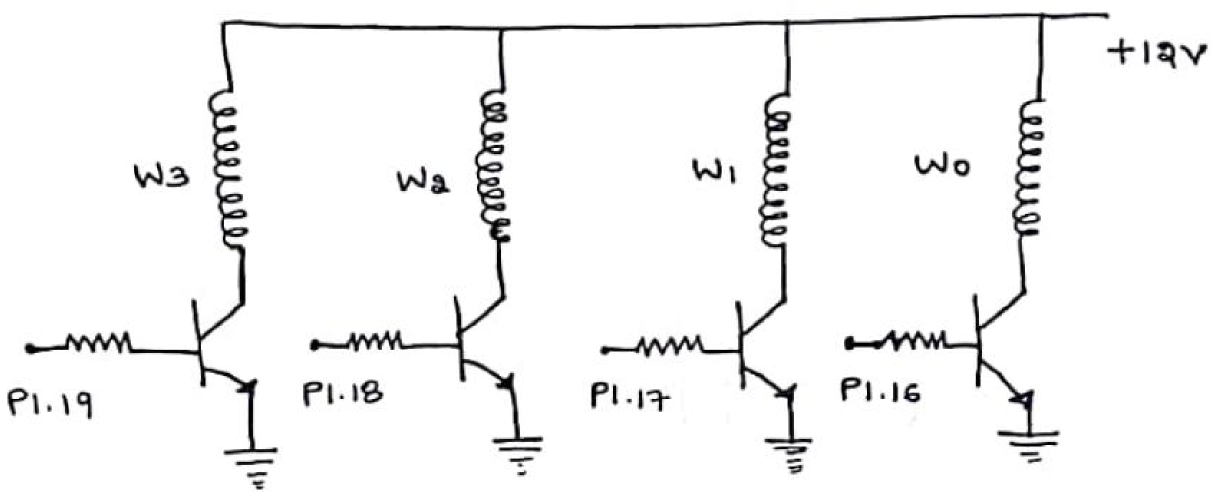
Left to Right Simulation


i: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
b b b b d e P t b o F b c S E b b b b
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩
⑪
⑫
⑬
⑭
⑮

Ebbb
SEbb
cSEb
bcSE
FbcS
oFbC
boFb
tboF
Ptbo
ePtb
dePt
bdeP
bbde
bbbd
bbbb

15 Iterations

Stepper Motor Interface



A Stepper motor is an electro-mechanical device which generates discrete motion in response to DC electrical signals. Stepper motors are widely used in industrial embedded applications, consumer electronic products and robotics control systems. The paper feed mechanism of a printer/fax makes use of stepper motor for its functioning.

The operating voltage for stepper motor is +12V DC. The stepper motor has four stator windings. The stator windings or coils are connected to the Darlington pair transistors to energize each coil. The Darlington transistor base is connected to the port P1 (P1.16-P1.19). The step angle is 1.8° i.e., 200 steps per revolution. The port P1 bits P1.19, P1.18, P1.17 and P1.16 are used to energize the four windings.

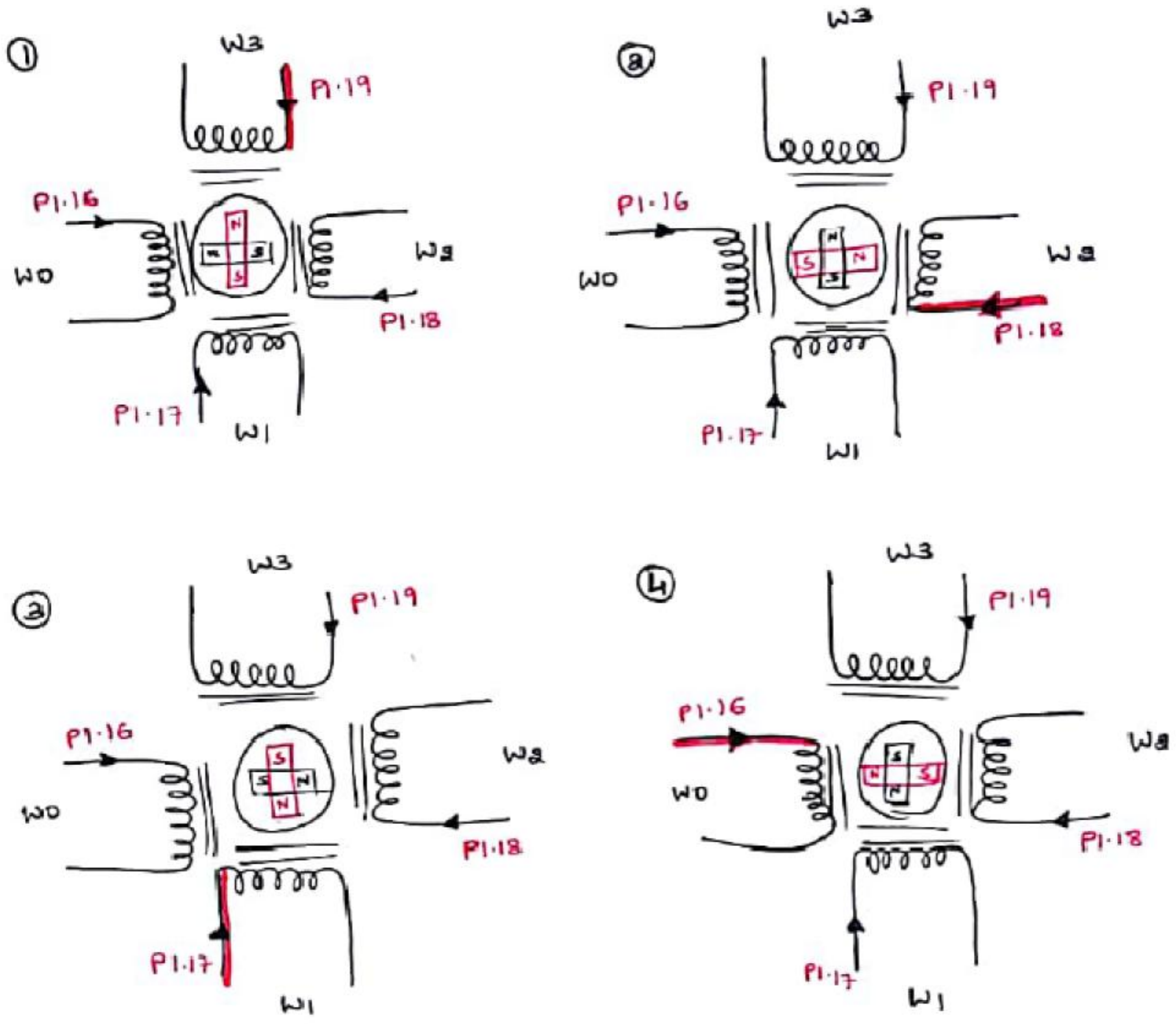
Data pattern to be sent through Port P1 to rotate the motor is as follows:

For clockwise					**For anticlockwise**			
(P1.19	**P1.18**	**P1.17**	**P1.16)**	**←windings→**	**(P1.19**	**P1.18**	**P1.17**	**P1.16)**
W3	**W2**	**W1**	**W0**		**W3**	**W2**	**W1**	**W0**
1	0	0	0		0	0	0	1
0	1	0	0		0	0	1	0
0	0	1	0		0	1	0	0
0	0	0	1		1	0	0	0

(The motor will move by 1 step for every pattern change)

For clockwise rotation:

Voltage is first supplied to Winding3 (W3) through P1.19, then to W2 through P1.18, then to W1 through P1.17 and then to W0 through P1.16.



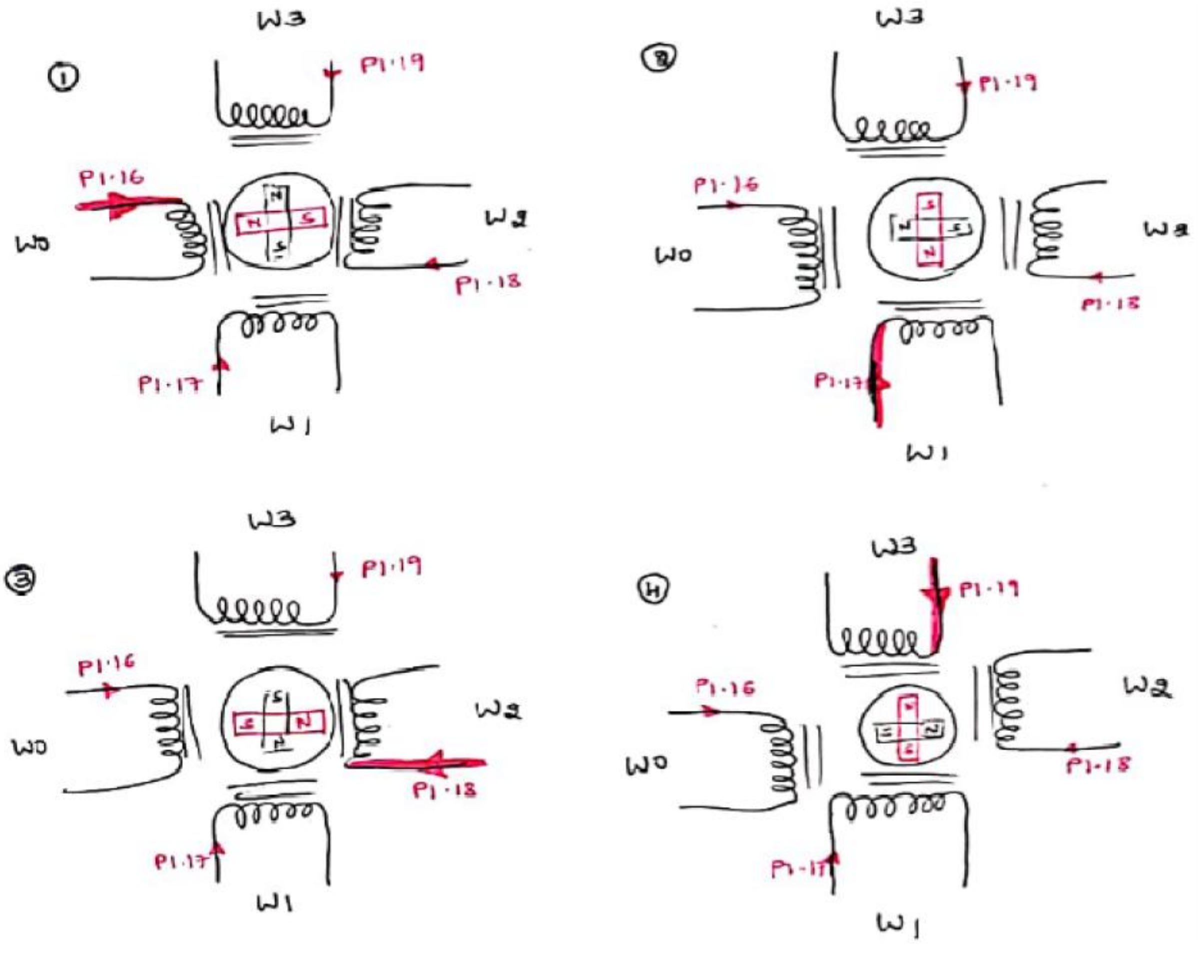
PROGRAM1: ***Develop an embedded C program to drive the Stepper motor interface to rotate the motor in clockwise direction.***

```
#include<stdio.h>
#include<lpc214x.h>
void delay(unsigned int x)
{
        unsigned char i;
        while(x--)
             for(i=0;i<200;i++);
}
```

```
int main()
{
        unsigned long i;
        PINSEL1 = 0X00000000;      //Configure P1 as GPIO port
        IODIR1 = 0X000F0000;       // Set P1.16 to P1.19 as Output pins
        while(1)
        {
                for(i=0x00080000;i>=0x00010000;i>>=1)
                {
                        IOSET1 = i;
                        delay(10000);
                        IOCLR1 = 0X000F0000;
                }
        }
        return 0;
}
```

For anticlockwise rotation:

Voltage is first supplied to Winding0 (W0) through P1.16, then to W1 through P1.17, then to W2 through P1.18 and then to W3 through P1.19.



PROGRAM2: ***Develop an embedded C program to drive the Stepper motor interface to rotate the motor in anti-clockwise direction.***

```
#include<stdio.h>
#include<lpc214x.h>

void delay(unsigned int x)
{
        unsigned char i;
        while(x--)
            for(i=0;i<200;i++);
}

int main()
{
         unsigned long i;
         PINSEL1 = 0X00000000;      //Configure P1 as GPIO port
         IODIR1 = 0X000F0000;       // Set P1.16 to P1.19 as Output pins
         while(1)
         {
                  for(i=0x00010000;i<=0x00080000;i<<=1)
                  {
                           IOSET1 = i;
                           delay(10000);
                           IOCLR1 = 0X000F0000;
                  }
         }
         return 0;
}
```

PROGRAM3: ***Develop an embedded C program to rotate the stepper motor by 360° in the clockwise direction and by 180° in the anti-clockwise direction using Stepper motor interface. Introduce suitable delay between successive steps.***

Calculation:

Number of steps required for rotating the motor by specified angle is given by

$$Count = \frac{Angle\_of\_rotation}{Step\_angle}$$

- For rotating the motor in clockwise direction for one revolution (360°), the number of steps required is computed as follows:

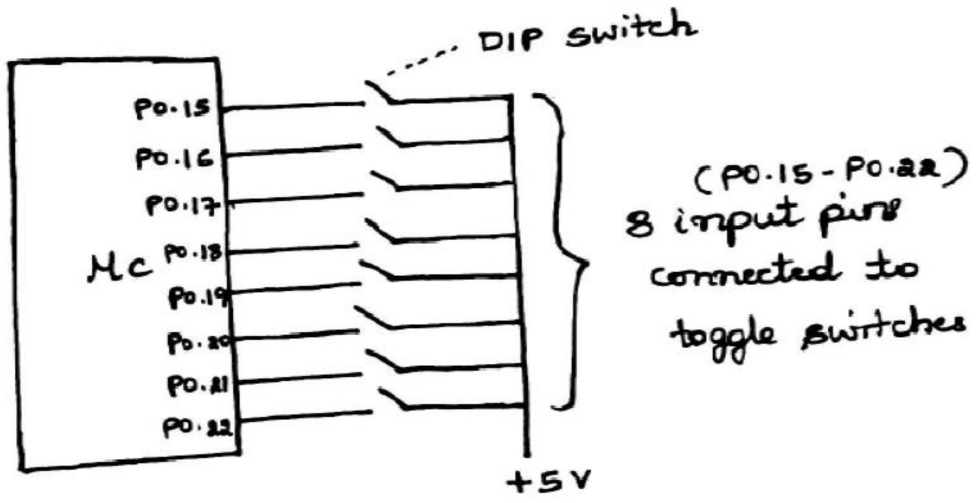
  Count = 360/1.8 = **200 steps**

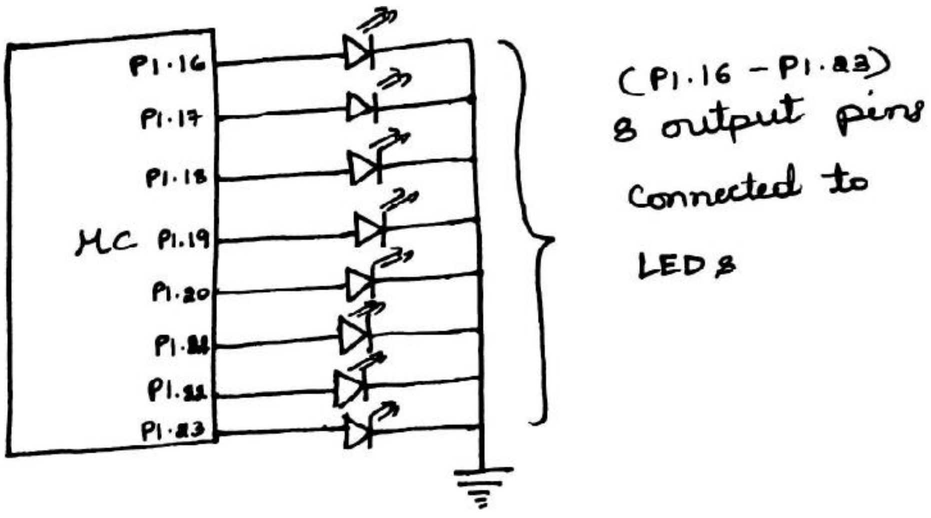
- For rotating the motor in anti-clockwise direction by 180°, the number of steps required is computed as follows:

  Count = 180/1.8 = **100 steps**

```
#include<stdio.h>
#include<lpc214x.h>

void delay(unsigned int x)
{
        unsigned char i;
        while(x--)
            for(i=0;i<200;i++);
}

int main()
{
        unsigned long i;
        unsigned char clk,aclk;
        PINSEL1 = 0X00000000;      //Configure P1 as GPIO port
        IODIR1 = 0X000F0000;       // Set P1.16 to P1.19 as Output pins
        while(1)
        {
                clk=200;   //No. of steps required for clockwise rotation
                aclk=100; //No. of steps required for anti-clockwise rotation
              while(1)
              {
                    for(i=0x00080000;i>=0x00010000;i>>=1)
                    {
                         IOSET1 = i;
                         delay(10000);
                         IOCLR1 = 0X000F0000;
                         if(--clk==0) break;
                    }
                    if(clk==0) break;
              }
            while(1)
           {
                  for(i=0x00010000;i<=0x00080000;i<<=1)
                 {
                      IOSET1 = i;
                      delay(10000);
                      IOCLR1 = 0X000F0000;
                       if(--aclk==0) break;
                  }
                  if(aclk==0) break;
              }
         }
         return 0;
    }
```

Logic Controller Interface





For Logic Controller Interface, port P0 of the microcontroller is used as an input port and port P1 is used as an output port. Port P1 pins (P1.16 to P1.23) of the microcontroller are connected to the output LEDs of the logic controller. If FFH is sent on port P1 pins, then all the output LEDs will glow to indicate ON state. If 00H is sent on P1 pins then all the eight LEDs will be in the OFF state.

The toggle switches are used to set the input in the logic controller. The Port P0 pins (P0.15 to P0.22) are connected to the switches. The switches are associated with LEDs to indicate the state of the switches. When the switch is opened, the LED is turned OFF and when the switch is closed, LED is turned ON.

PROGRAM1: ***Develop an embedded C program to read the status of 8 input bits from the Logic Controller Interface and display 'FF' if it is even parity bits otherwise display 00. Also display number of 1's in the input data.***

```
#include<lpc214x.h>
unsigned char countOnes(unsigned long x)
{
          unsigned char i,count=0;
          for(i=0;i<8;i++)
          {
                  if(x &(0x00400000>>i))
                        count++;
          }
          return count;
}
```

Bit no.:	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
	0000	0000	0 100	0000	0 000	0000	0000	0000

```
void delay(unsigned long x)
{
         unsigned char i;
        while(x--)
              for(i=0;i<250;i++);
}

int main ()
{
         unsigned long temp ;
          unsigned char count;

          PINSEL0 = 0;          //Configure P0 as GPIO port
          PINSEL1 = 0;         //Configure P1 as GPIO port

         IODIR0 = 0X00000000;  // Set port P0.15 to P0.22   as input
         IODIR1=  0X00FF0000;  // Set port P1.16 to P1.23   as output

         while(1)
         {
                temp = IOPIN0;              // read port P0
                count = countOnes(temp);
              if(count%2==0)
                     IOSET1 = 0X00FF0000;  //display FF if even parity
              else
                     IOCLR1 = 0X00FF0000;  //display 00 if odd parity
                delay(20000);
                IOPIN1 = count << 16;          //display no. of 1's in input
                delay(20000);
        }
}
```

Output Pins of Port P1 (16-23)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0	0	0	0	0	0	0	0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PROGRAM2: ***Develop an embedded C program to read the status of 8 input bits from the Logic Controller Interface and display the complement of it.***

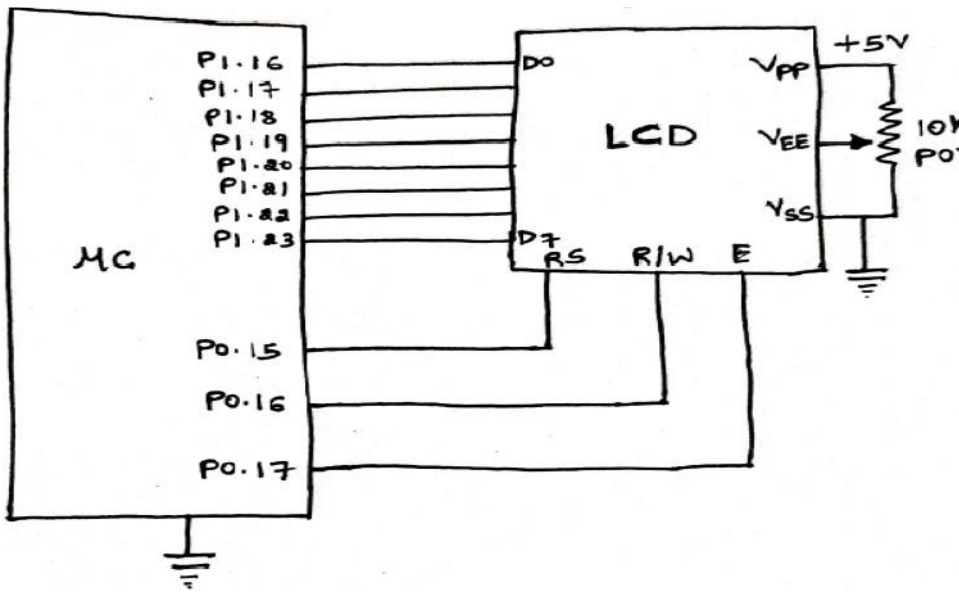
```
void delay(unsigned long x)
{
        unsigned char i;
        while(x--)
                for(i=0;i<250;i++);
}

int main ()
{
        unsigned long temp ;

        PINSEL0 = 0;          //Configure P0 as GPIO port
        PINSEL1 = 0;          //Configure P1 as GPIO port

        IODIR0 = 0X00000000;   // Set port P0.15 to P0.22   as input
        IODIR1 =  0X00FF0000; // Set port P1.16 to P1.23   as output

        while(1)
        {
                temp = IOPIN0;              // read port P0
                temp=temp<<1;

                IOPIN1 = ~temp;          //display complement of input
                delay(20000);
        }
}
```

PROGRAM3: ***Develop an embedded C program to simulate decimal up-down counter to count from 0-9 and 9-0 using Logic controller interface.***

```
void delay(unsigned long x)
{
         unsigned char i;
        while(x--)
              for(i=0;i<250;i++);
}

int main ()
{
         unsigned char i ;

        PINSEL1 = 0;          //Configure P1 as GPIO port
        IODIR1 =  0X00FF0000; // Set port P1.16 to P1.23   as output

         while(1)
         {
               for(i=0;i<9;i++)          //Decimal up counter 0-9
              {
                   IOPIN1 = i << 16;
                   delay(20000);
               }

               for(i=9;i>0;i--)            //Decimal down counter 9-0
              {
                   IOPIN1 = i << 16;
                   delay(20000);
              }
        }
        return 0;
}
```

Liquid Crystal Display (LCD) Interface

LCDs are electronic display devices. They are found everywhere -- in laptop computers, digital clocks and watches, microwave ovens, CD players and many other electronic devices.



LCD consists of Display Data RAM (DDRAM), CGROM (Character Generator ROM), shift registers, bit/pixel drivers, refreshing logics and LCD controllers. The data to be displayed on LCD is to be written on to the DDRAM using ASCII format. CGROM contains bit/pixel patterns for every character to be displayed (Pre-programmed). Shift registers are used to convert CGROM parallel data to serial data. Drivers are required to drive (ON/OFF) the bits. Refreshing logics are required to hold the display data, as the dots are displayed row by row basis continuously, like in CRT.

Whatever the data you write to LCD is of two types, either it is a command written to the instruction command code register of LCD (Configuration) or ASCII code of character to be displayed on LCD (DDRAM) which is written to the data register of LCD.

- The RS (Register Select) pin is used for selection of either the command register or the data register.
  - If RS = 0, then the instruction command code register is selected, allowing the user to send a command such as clear display etc.
  - If RS= 1, then the data register is selected, allowing the user to send data to be displayed on the LCD.

- R/W (Read/Write) pin allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W=0 when writing.

- E (Enable) pin is used by the LCD to latch the information present at the data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be minimum of 450ns wide.

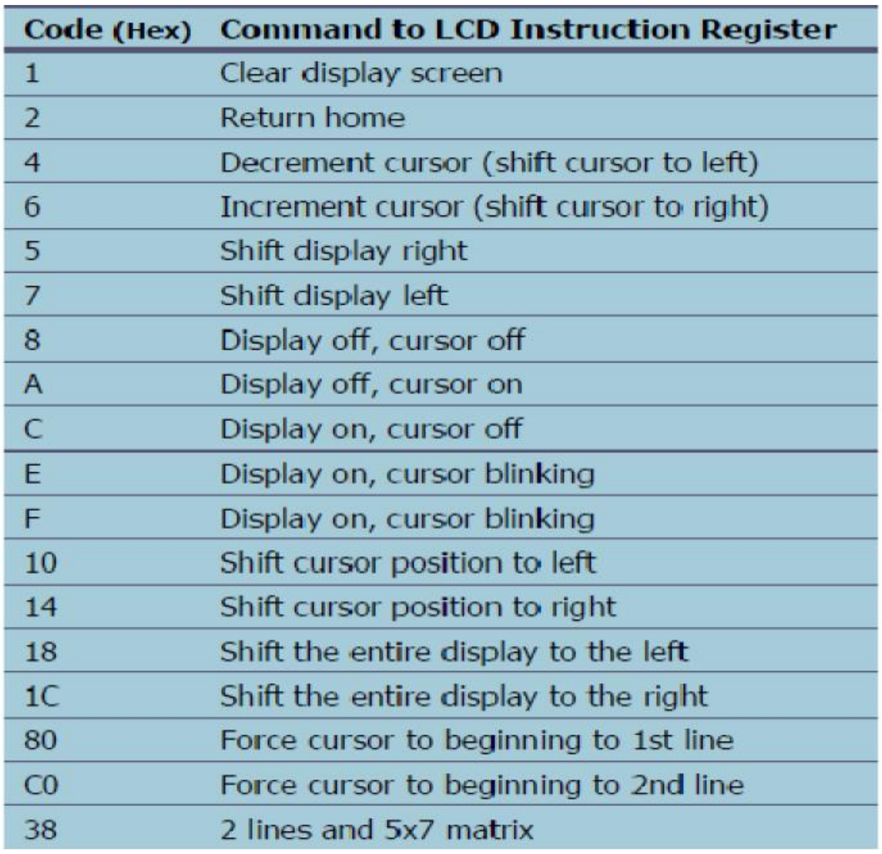
- D0-D7: These are 8-bit data pins used to send information to the LCD or read the contents of the LCD's internal registers.

- V_{PP} and V_{SS} provide +5V and ground, respectively. V_{EE} is used for controlling LCD contrast.

Following two steps are to be followed to program the LCD:

1. Configure LCD by writing commands.
2. Write the actual string data, character by character by issuing DDRAM address command 0x80 for first line, 0xC0 for second line.

LCD Command codes:



Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

List of few LCD Instructions:

Instruction	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Description
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter
Display On/Off control	0	0	0	0	0	0	1	D	C	B	Sets On/Off entire display(D), cursor On/Off (C), and blink of cursor position character (B)
Function Set	0	0	0	0	1	DL	N	F	-	-	Sets interface data length (DL), number of display lines (N), and character font (F)

PROGRAM: ***Develop an embedded C program to display "Hello World" message on a 2x16 character LCD interface.***

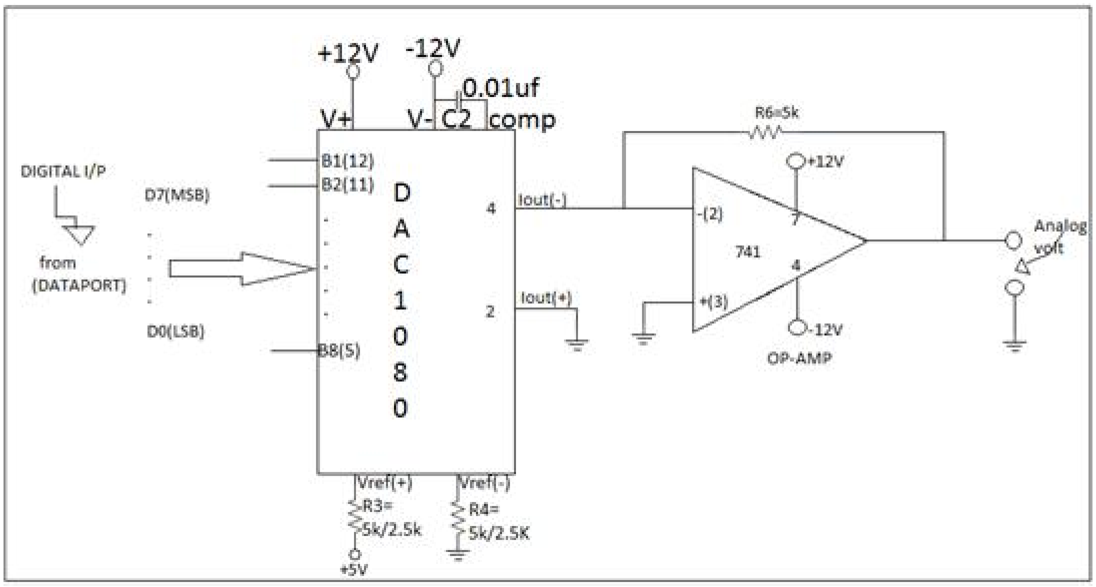
```
#include <lpc214x.h>
void delay(unsigned long x)
{
        unsigned char i;
        while(x--)
             for(i=0;i<250;i++);
}

void LCD_Command(unsigned char cmd)
{
    IOCLR0 = 0X00018000;//P0.15-RS=0,P0.16-RW=0;CMD register & write operation

      IOPIN1 = cmd << 16; //P1.23-P1.16 of P1 to send data or command to LCD

     IOSET0 = 0X00020000;   //Send High to low pulse
     delay(2);
     IOCLR0 = 0x00020000;
     delay(250);
}

void LCD_init(void)
{
        LCD_Command(0x38);   // Function set 8-bit, 2 line 5x7 font
        LCD_Command(0x0c);   // Display on off control: Display on, cursor off
        LCD_Command(0x01);   //Clear the LCD display
}
```

```
}
```

Command	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Function set (0X38)	0	0	1	DL	N	F	-	-
	0	0	1	1	1	0	0	0
	DL=1 means 8-bit data (byte mode), N=1 means 2 line display, F=0 means 5x7 font							
Display On/Off control (0x0C)	0	0	0	0	1	D	C	B
	0	0	0	0	1	1	0	0
	D=1 means Display ON, C=0 means Cursor OFF, B=0 means No Blinking of cursor position character							
Clear entire display	0	0	0	0	0	0	0	1

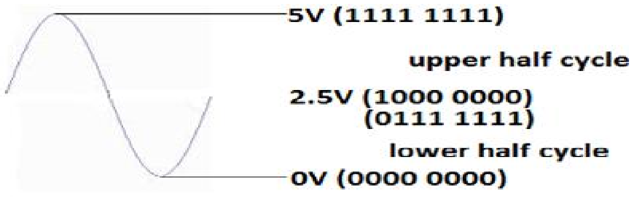
```
void LCD_Data(unsigned char data)
{
        IOSET0 = 0X00008000; //P0.15-RS=1 Data register
        IOCLR0 = 0X00010000; // P0.16 - RW =0 write operation

        IOPIN1=data << 16; //P1.23-P1.16 of P1 to send data or command to LCD
        IOSET0 = 0X00020000;  //Send high to low pulse
        delay(2);
        IOCLR0 = 0x00020000;
        delay(250);
}

int main()
{
          unsigned char str[]="Hello World";
          unsigned char i;
          PINSEL0 = 0;  // Configure P0 as GPIO port
          PINSEL1 = 0;  // Configure P1 as GPIO port
          IODIR0 = 0X00038000; // Set P0 pins 15, 16 and 17 as output
          IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output
          LCD_init();
          LCD_Command(0x80);
          for(i=0;str[i]!='\0';i++)
                LCD_Data(str[i]);
         while(1);
         return 0;
}
```

Digital to Analog Converter (DAC) Interface



DAC is a device widely used to convert digital pulses to analog signals. The DAC interface consists of DAC chip, OPAMP 741 and voltage converter. A reference voltage of +5V is generated using the voltage regulator and is fed to V_{ref} point of DAC. The number of data bit inputs decides the resolution of DAC. i.e., if there are n digital input pins then there are 2^n analog voltage levels. The DAC IC used in the interface has 8 input pins which provides 256 analog voltage levels. The Output of DAC is connected to an op-amp with a feedback resistor. The final output can be connected to CRO to display the waveform. The output voltage of +5V is obtained from DAC when the digital input is FFH and the output voltage will be 0V when the digital input is 00H. The output of DAC is fed to the operational amplifier to get the final output. The digital values 00H-FFH corresponds to the analog voltage 0-5V. The 8-bit data can be fed into DAC from Port P1 (P1.16-P1.23) of the microcontroller.

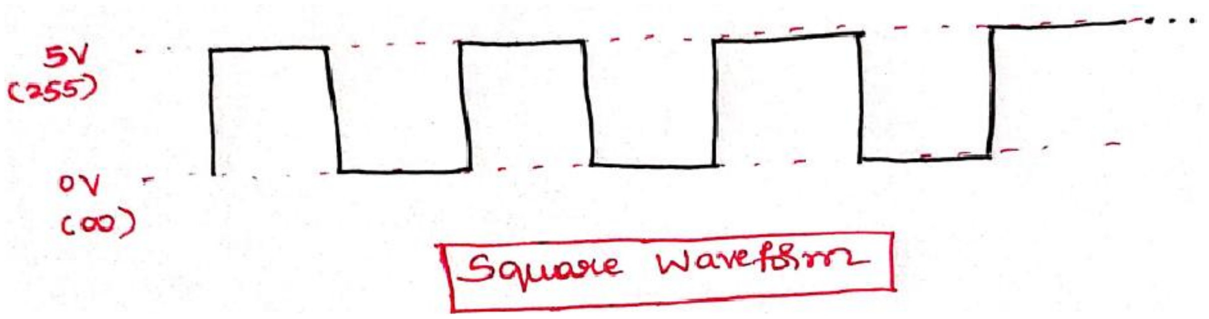
The ***digital formula*** used to get the values for positive cycle of the sine waveform is as follows:
127 sinΘ + 127

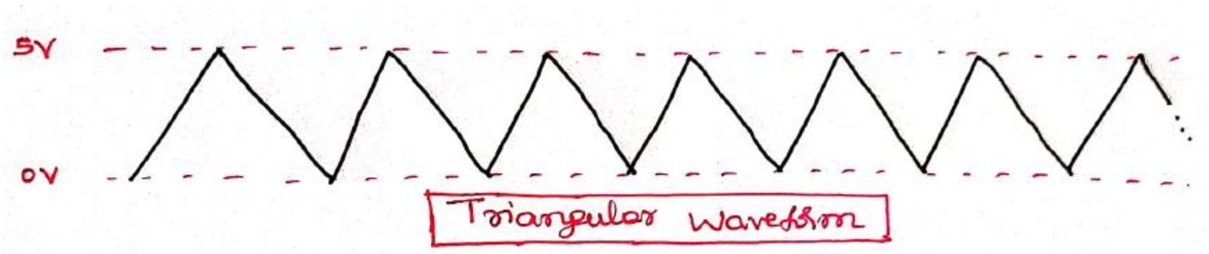


If 30 samplings are done to get the positive cycle of sine waveform then, we can get the values by using the digital formula for every 6° increment (180°/30=6°) i.e.,
127 sin 0° +127 =127

127 sin 6° +127=140 and so on

This can be continued till 90° and the values computed can be considered in reverse order for sampling values from 90° -180°.

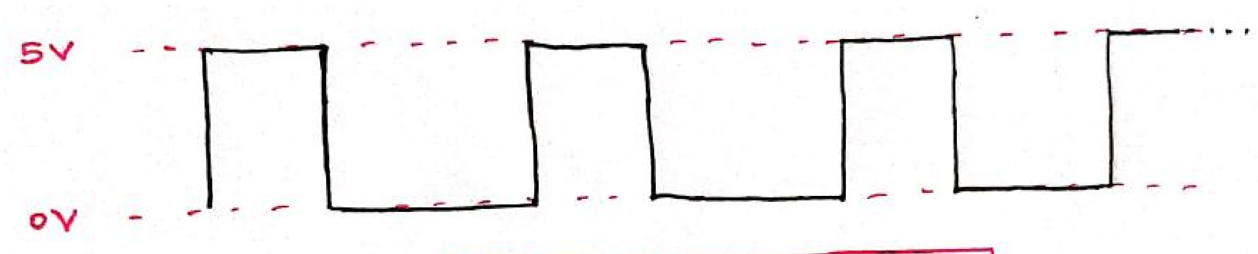




PROGRAM1: ***Develop an embedded C program to generate square and triangular waveforms using the DAC Interface.***

```
#include <lpc214x.h>
void delay(unsigned long x)
{
        unsigned char i;
        while(x--)
             for(i=0;i<250;i++);
}

int main()
{
       unsigned long i;
       unsigned char j;

        PINSEL1 = 0;                // Configure P1 as GPIO
        IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

        while(1)
        {
              //to display square waveform
```

```
            for(i=0;i<1000;i++)
            {
                  IOSET1 = 0X00FF0000;     // Send FFH on P1 for some delay
                  delay(10);

                  IOCLR1 = 0X00FF0000;   //Send 0 on P1 for some delay
                  delay(10);
             }

           // to display triangular waveform
            for(i=0;i<1000;i++)
           {
                 for(j=0;j<255;j++)
                     IOPIN1 = j<<16;
                 for(j=255;j>0;j--)
                     IOPIN1 = j<<16;
            }
        }
        return 0;
      }
```

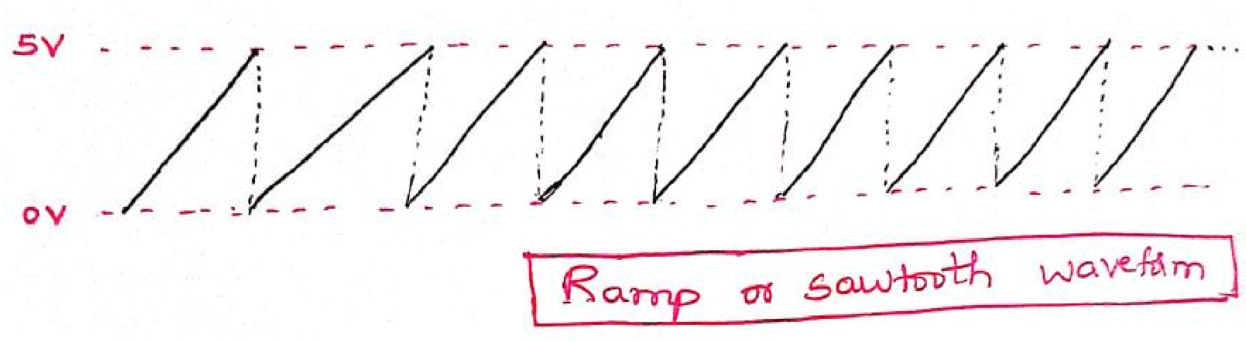


Rectangular Waveform

PROGRAM2: ***Develop an embedded C program to generate rectangular waveform using the DAC Interface.***

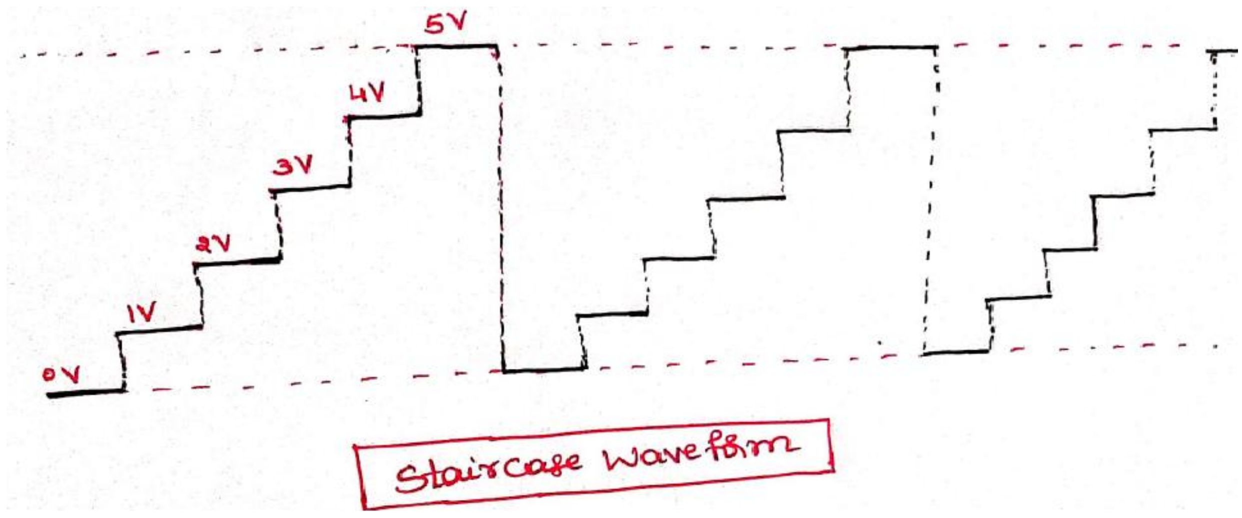
```
#include <lpc214x.h>

void delay(unsigned long x)
{
      unsigned char i;

      while(x--)
          for(i=0;i<250;i++);
}
```

```
int main()
{
       PINSEL1 = 0;                  // Configure P1 as GPIO
       IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

       while(1)
       {
                  IOSET1 = 0X00FF0000;     // Send FFH on P1 for some delay
                  delay(10);

                  IOCLR1 = 0X00FF0000;   //Send 0 on P1 for some delay
                  delay(20);
        }
        return 0;
       }
```



PROGRAM3: ***Develop an embedded C program to generate Ramp waveform using the DAC Interface.***

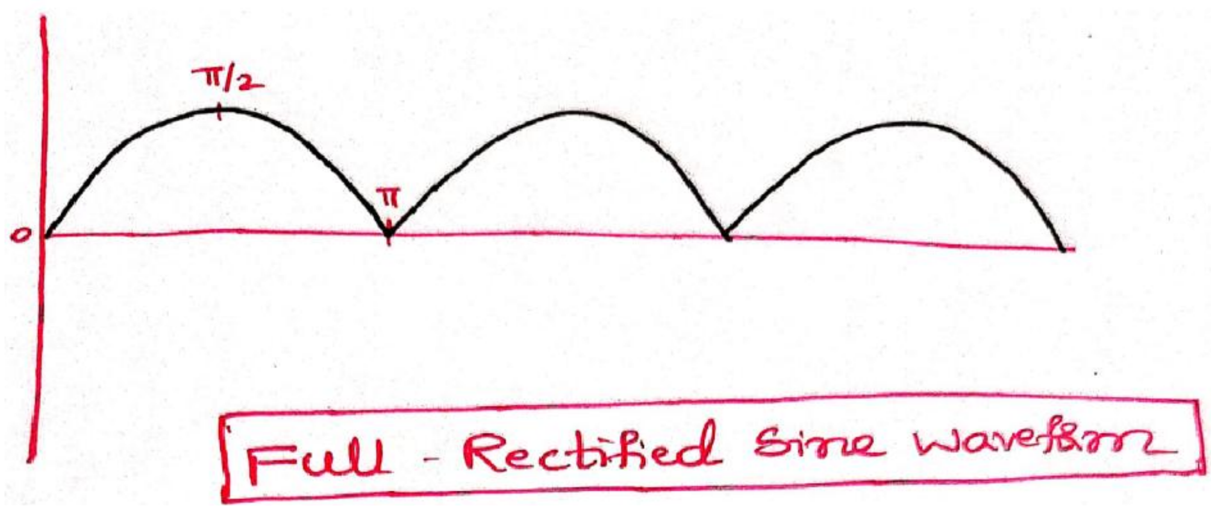
```
#include <lpc214x.h>

int main()
{
      unsigned char i;

       PINSEL1 = 0;                 // Configure P1 as GPIO
       IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

       for(i=0;i<=255;i++)
           IOPIN1 = i<<16;
      return 0;
}
```



For 5V, the digital value is 255 (FFH)
For 1V, the digital value is 255/5 = 51 (33H)

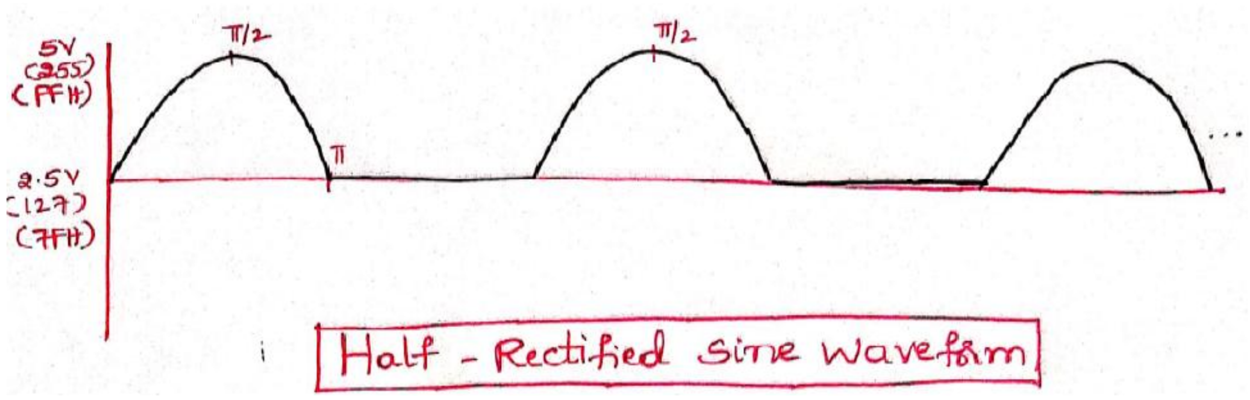
PROGRAM4: ***Develop an embedded C program to generate Staircase waveform using the DAC Interface.***

```
#include <lpc214x.h>
void delay(unsigned long x)
{
        unsigned char i;
        while(x--)
             for(i=0;i<250;i++);
}

int main()
{
       unsigned char i,k;
        PINSEL1 = 0;                // Configure P1 as GPIO
        IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output
        while(1)
        {
            i=0;
            for(k=0;k<6;k++)
            {
                  IOPIN1 = i<<16;
                  delay(10);
                  i=i+51;
             }
         }
         return 0;
}
```



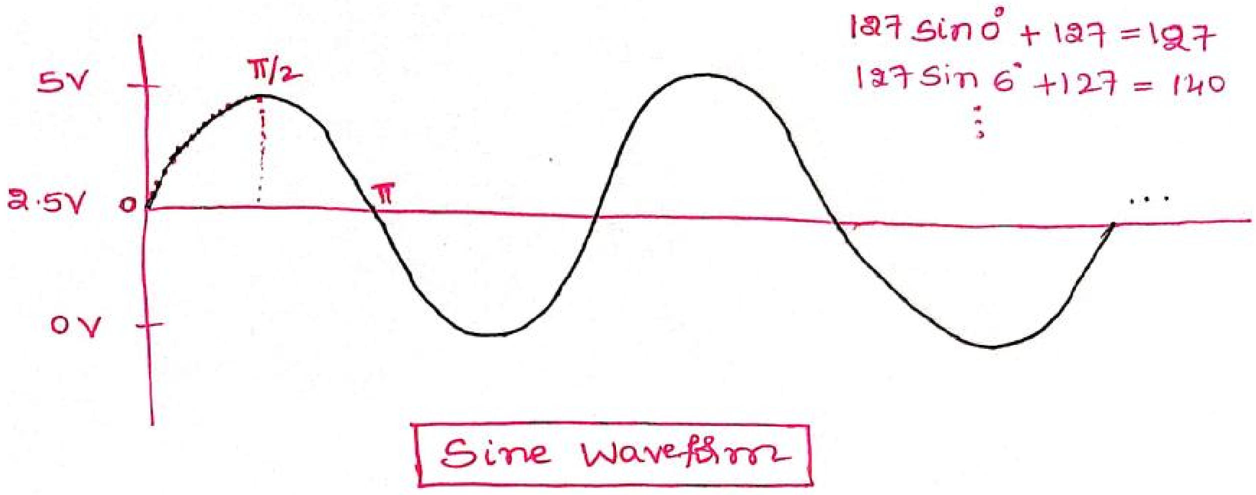
PROGRAM5: ***Develop an embedded C program to generate Full Rectified Sine waveform using the DAC Interface.***

```
#include <lpc214x.h>

int main()
{
     unsigned char i;
     unsigned char data[]={127, 140, 153, 166, 178, 190, 201,
                          211, 221, 229, 236, 243, 247, 251,
                          253, 255, 253, 251, 247, 243, 236,
                          229, 221, 211, 201, 190, 178, 166, 153, 140, 127};

       PINSEL1 = 0;              // Configure P1 as GPIO

       IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

        while(1)
        {
              for(i=0;i<31;i++)
                    IOPIN1 = data[i]<<16;
         }
        return 0;
    }
```



PROGRAM6: *Develop an embedded C program to generate Half Rectified Sine waveform using the DAC Interface.*

```
#include <lpc214x.h>

int main()
{
      unsigned char i;
      unsigned char data[]={127, 140, 153, 166, 178, 190, 201,
                                    211, 221, 229, 236, 243, 247, 251,
                                    253, 255, 253, 251, 247, 243, 236,
                                    229, 221, 211, 201, 190, 178, 166, 153, 140, 127};

        PINSEL1 = 0;                  // Configure P1 as GPIO

        IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

         while(1)
         {
                for(i=0;i<31;i++)
                      IOPIN1 = data[i]<<16;

                for(i=0;i<31;i++)
                      IOPIN1 = 0x7F<<16;

          }
          return 0;
    }
```



Sine Waveform

PROGRAM7: ***Develop an embedded C program to generate Sine waveform using the DAC Interface.***

```
#include <lpc214x.h>
int main()
{
     unsigned char i;
     unsigned char data[]={127, 140, 153, 166, 178, 190, 201,
                              211, 221, 229, 236, 243, 247, 251,
                              253, 255, 253, 251, 247, 243, 236,
                              229, 221, 211, 201, 190, 178, 166, 153, 140, 127};

       PINSEL1 = 0;              // Configure P1 as GPIO
       IODIR1 = 0X00FF0000; // Set P1 pins 16 to 23 as output

         while(1)
         {
               for(i=0;i<31;i++)
                    IOPIN1 = data[i]<<16;

               for(i=0;i<31;i++)
                    IOPIN1 = (~data[i]+1)<<16;

          }
          return 0;
    }
```

Light Emitting Diode (LED)

- LED is an important output device for visual indication in any embedded system.
- LED can be used as an indicator for the status of various signals or situations.
  - Typical examples are indicating the presence of power conditions like 'Device ON', Battery low' or 'charging of battery' for a battery operated handheld embedded devices.
- LED is a p-n junction diode and it contains an anode and a 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.
- A resistor is used in series between the power supply and the LED to limit the current through the LED.
- The ideal LED interfacing circuit is shown in below figure:

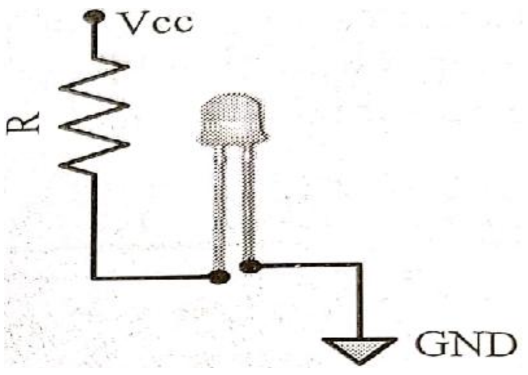


Figure: LED Interfacing

- LEDs can be interfaced to the port pin of a processor/controller in two ways:

  - In the first method, the anode is directly connected to the port pin and the port pin drives the LED. In this approach, the port pin 'sources' current to the LED when the port pin is at logic 1 (High).

  - In the second method, the cathode of the LED is connected to the port pin and the anode to the supply voltage through a current limiting resistor. The LED is turned on when the port pin is at logic 0 (Low). Here the port pin 'sinks' current. In this approach, the current is directly sourced by the power supply and the port pin acts as the sink for current. Here, we will get the required brightness for the LED.

Push Button Switch

- It is an input device.
- It comes in two configurations, namely 'Push to Make' and 'Push to Break'.
  - In the 'Push to Make' configuration, the switch is normally in the open state and it makes a circuit contact when it is pushed or pressed.
  - In the 'Push to Break' configuration, the switch is normally in the closed state and it breaks the circuit contact when it is pushed or pressed.
- In embedded applications, push button is generally used as reset and start switch and pulse generator.
- The Push button is normally connected to the port pin of the host processor/controller. Depending on the way in which the push button is interfaced to the controller, it can generate either a 'HIGH' pulse or a 'LOW' pulse.
- Below figure illustrates how the push button can be used for generating 'LOW' and 'HIGH' pulses.

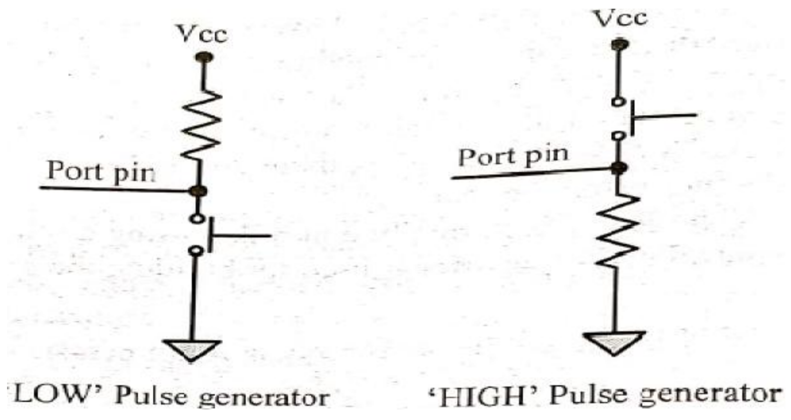


Figure: Push Button Switch Configurations

Keyboard Interfacing

- Keyboard is an input device for user interfacing.
- There may be situations demanding a large number of keys for user input. In such situations, it may not be possible to interface each key to a port pin due to the limitation in the number of general purpose port pins available for the processor/controller in use and moreover it is wastage of port pins.
- Matrix Keyboard is an optimum solution for handling large key requirements.
  - It greatly reduces the number of interface connections.
- For example, for interfacing 16 keys, in the direct interfacing technique, 16 port pins are required, whereas in the matrix keyboard, only 8 lines are required.
- Below figure illustrates the connection of 16 keys in a matrix keyboard (4 Column x 4 Row matrix).

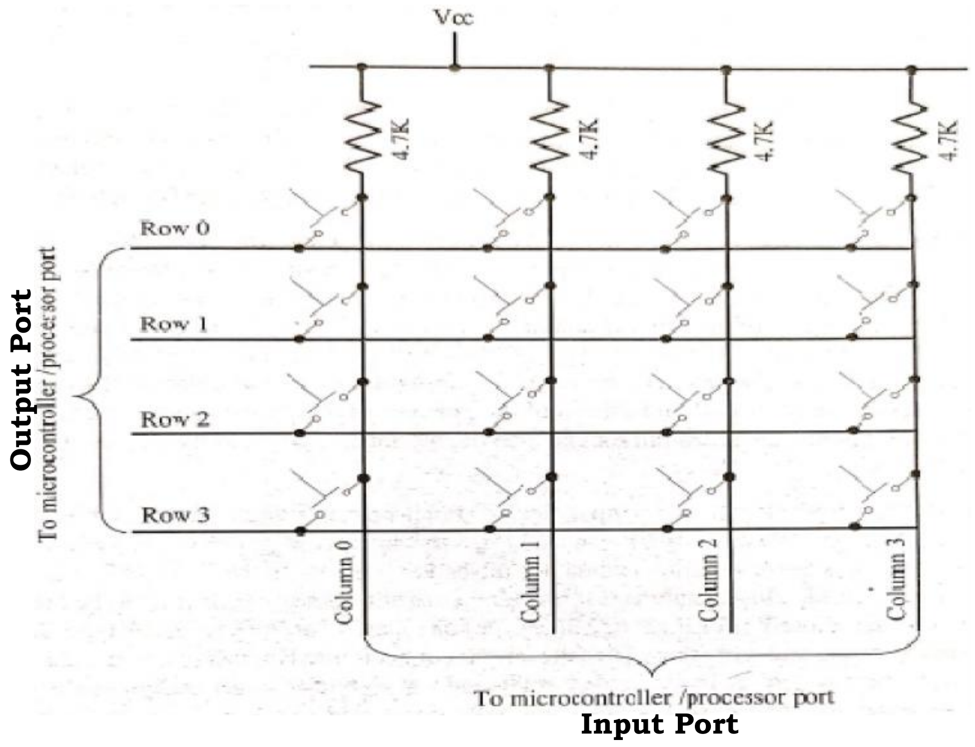


Figure: Keyboard Interfacing

- In a matrix keyboard, the keys are arranged in matrix fashion (i.e., they are connected in the form of rows and columns).

- Pull-up resistors are connected to the column lines to limit the current that flows to the Row line on a key press.

For detecting a key press, the keyboard uses the scanning technique.

- If no key is pressed , we will have the bit status '1111' on input port pins (column0-column3), as all the inputs are pulled up by pull up resistors.
- If any key is pressed, say first key, it will short row0 and column0 lines, so whatever data (0 or 1) available at row0 is available at column0. Since already columns are pulled high, it is required to apply logic '0' at row0 to see change in column0 when the key is pressed.
    - *Each row of the matrix is pulled low and the columns are read.*
    - *After reading the status of each column corresponding to a row, the row is pulled high and the next row is pulled low and the status of the columns are read.*
    - This process is repeated until the scanning for all rows are completed.
    - When a row is pulled low and if a key connected to the row is pressed, reading the column to which the key is connected will give logic 0.

********** End of Unit - 5 ***********