

# UNIT-4 LPC2148

---

By

Prof M S Srividya

Assistant Professor, Dept. of CSE

RVCE

# Features of the LPC 2148 Microcontroller

1. Uses the core ARM 7TDMI-S in a tiny LQFP64 package ( Low Profile Quad Flat Package (LQFP)), operating at 60 MHz clock speed
2. 8kb-40kb of on-chip static RAM (40kb for LPC 2148)
3. 32kb to 512kb of on-chip flash memory(512kb for LPC 2148)
4. 128 bit memory accelerator enables high-speed 60MHz operation
5. USB 2.0 Full speed device controller with DMA (used as USB device, not as a host)
6. 10 bit ADCs provide a total of 6/14(2148 has 14) analog inputs
7. Single 10-bit DAC provides variable analog output
8. Two 32bit timers/event counters
9. PWM Unit
10. RTC (Real Time Clock) with battery backup facility
11. 2 UARTs
12. I2C interface (2 sets)
13. SPI & SSP interfaces ( supports SD card )
14. Up to 45 GPIO pins
15. Single power supply with POR(Power On Reset) and BOD(Brown out Detect) circuits, support idle & power-down modes

100





# Internal Buses :

**AMBA** - Advanced Microcontroller Bus Architecture, is a standard defined by ARM for on-chip buses in its SoC (System On Chip ) designs.

It defines three buses with different protocols and speed,

---

1. Local bus – the fastest bus, which connects the processor core with the memory, as the memory accesses have to be very fast. Connected to Flash memory & SRAM using Memory controllers. Fast GPIO block is also connected to this bus
2. AHB (Advanced High Performance) bus – AHB Bridge is used to interface local bus with AHB bus, VIC (Vectored Interrupt Controller) is connected to this bus.
3. VLSI Peripheral Bus (VPB bus) – there is a bridge which communicates between the low speed VPB bus and the higher speed AHB bus, this bus is used to connect to all the on chip peripherals.

VPB Bus and Divider – VPB divider has a register, programming of this register reduces the processor clock frequency (CCLK) to generate lower speed Peripheral clock (PCLK) used by the peripherals. On reset, PCLK is  $\frac{1}{4}$  of CCLK.

### **Generation of CPU Clock (CCLK) and PCLK (Peripheral Clock):**

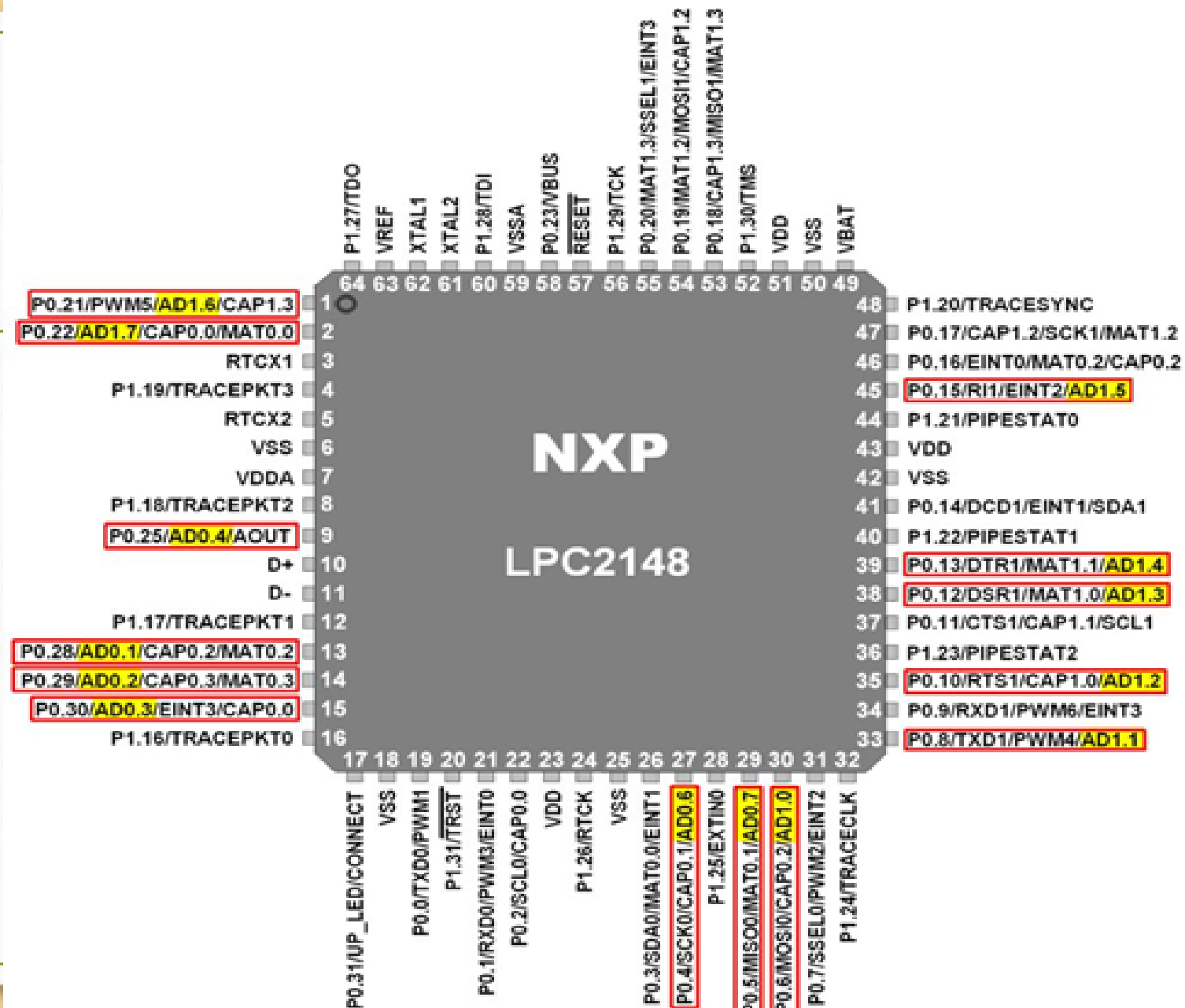
- CCLK is connected to ARM CPU core
- PCLK is derived from CCLK and used to drive the different on chip peripherals.
- CCLK and PCLK are generated from on chip PLL (Phase locked loop) modules based on the crystal connected to XTAL1 & XTAL2.
- There are two PLL modules in the LPC 2148 Microcontroller
- The PLL0, PLL1 accept the clock (FOSC) obtained from the crystal connected at XTAL1, XTAL2 and multiplies this to enhance the frequency in the range of 10MHz to 60MHz.
- By default, the peripheral clock (PCLK) runs at a quarter speed of the CCLK
- If both the PLLs are turned off on Reset :  $CCLK = FOSC$ , if crystal frequency is 12MHz, the  $CCLK = 12\text{ MHz}$ ,  $PCLK = 3\text{ MHz}$
- If PLLs are enabled, the default values are:  $FOSC = 12\text{ MHz}$  (if 12MHz Crystal is used),  $CCLK = 60\text{ MHz}$ ;  $PCLK = 15\text{ MHz}$

# Interfacing & Programming GPIO

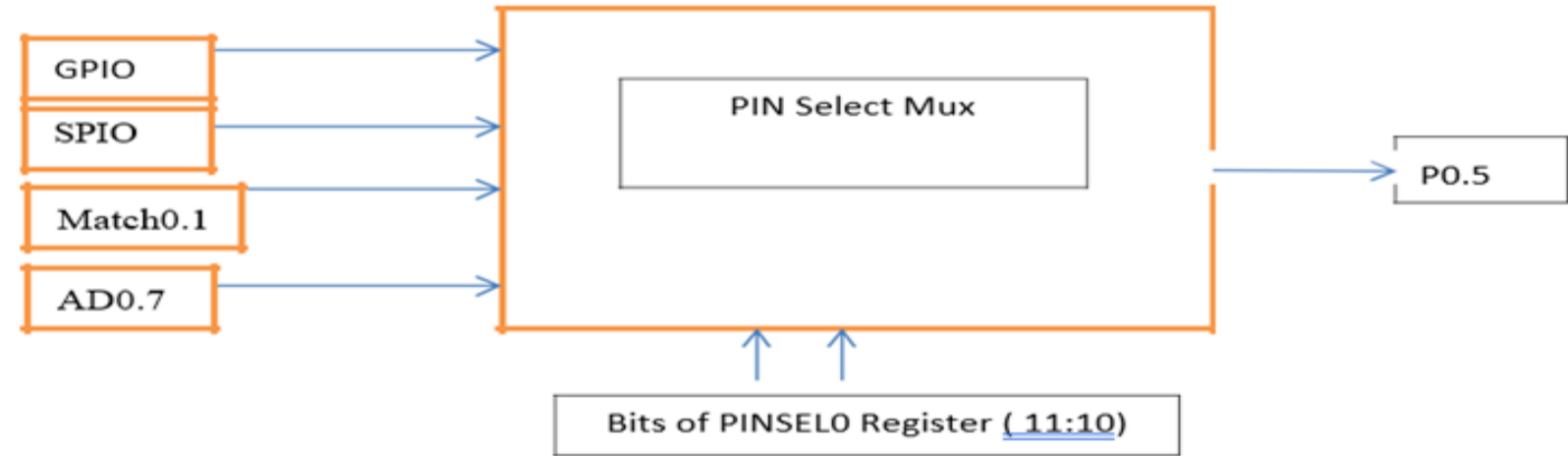
LPC 2148 provides 2 ports, of size 32 bits, used to interface LCD, relays, LEDs etc.

---

- ❖ **Port 0** is a 32 bit I/O port, of which 28 pins can be used for digital i/o, P0.31 used only for o/p, pins P0.24, P0.26 & P0.27 are reserved and not available for use. [29]
- ❖ **Port 1** is a 32 bit I/O port, but only P1.16 – P1.31 are available for I/O. Hence totally 45 I/O are possible, but all the pins have alternate functions also, as detailed below. [29+16=45]







#### Example for **Pin P0.5** :

| Bits of PINSEL0 Register | Port Pin | value of PINSEL bits | Function Selected   | Reset Value |
|--------------------------|----------|----------------------|---------------------|-------------|
| 11:10                    | P0.5     | 00                   | GPIO                | 0           |
|                          |          | 01                   | MISO0(SPI-0)        |             |
|                          |          | 10                   | Match 0.1 (timer 0) |             |
|                          |          | 11                   | AD0.7               |             |

To use P0.5 as GPIO, we have to make it 00,

`PINSEL0 = 0x0000 0000;` By default, on reset, all port pins act as GPIO pins



# Three PINSEL Registers

---

- PINSEL0,PINSEL1,PINSEL2
- Two bits are required to configure one pin
- First two are for the Port0, third is for Port1
- PINSEL0 is used to configure the pins P0.0 to P0.15
- PINSEL1 is used to configure P0.16 to P0.31
- PINSEL2 is used to configure P1.16 to P1.31.

| <i><b>PINSEL0</b></i> | <i><b>Pin Name</b></i> | <i><b>Function when 00</b></i> | <i><b>Function when 01</b></i> | <i><b>Function when 10</b></i> | <i><b>Function when 11</b></i> | <i><b>Reset Value</b></i> |
|-----------------------|------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|---------------------------|
| 1:0                   | P0.0                   | GPIO Port 0.0                  | TXD (UART0)                    | PWM1                           | Reserved                       | 00                        |
| 3:2                   | P0.1                   | GPIO Port 0.1                  | RXD (UART0)                    | PWM3                           | EINT0                          | 00                        |
| 5:4                   | P0.2                   | GPIO Port 0.2                  | SCL0 (I <sup>2</sup> C0 )      | Capture 0.0 (Timer 0)          | Reserved                       | 00                        |
| 7:6                   | P0.3                   | GPIO Port 0.3                  | SDA0 (I <sup>2</sup> C0 )      | Match 0.0 (Timer 0)            | EINT1                          | 00                        |
| 9:8                   | P0.4                   | GPIO Port 0.4                  | SCK0 (SPI0)                    | Capture 0.1 (Timer 0)          | AD0.6                          | 00                        |
| 11:10                 | P0.5                   | GPIO Port 0.5                  | MISO0 (SPI0)                   | Match 0.1 (Timer 0)            | AD0.7                          | 00                        |
| 13:12                 | P0.6                   | GPIO Port 0.6                  | MOSI0 (SPI0)                   | Capture 0.2 (Timer 0)          | AD1.0                          | 00                        |
| 15:14                 | P0.7                   | GPIO Port 0.7                  | SSEL0 (SPI0)                   | PWM2                           | EINT2                          | 00                        |
| 17:16                 | P0.8                   | GPIO Port 0.8                  | TXD (UART1)                    | PWM4                           | AD1.1                          | 00                        |
| 19:18                 | P0.9                   | GPIO Port 0.9                  | RXD (UART1)                    | PWM6                           | EINT3                          | 00                        |
| 21:20                 | P0.10                  | GPIO Port 0.10                 | RTS (UART1)                    | Capture 1.0 (Timer 1)          | AD1.2                          | 00                        |
| 23:22                 | P0.11                  | GPIO Port 0.11                 | CTS (UART1)                    | Capture 1.1 (Timer 1)          | SCL1 (I <sup>2</sup> C1)       | 00                        |
| 25:24                 | P0.12                  | GPIO Port 0.12                 | DSR (UART1)                    | Match 1.0 (Timer 1)            | AD1.3                          | 00                        |
| 27:26                 | P0.13                  | GPIO Port 0.13                 | DTR (UART1)                    | Match 1.1 (Timer 1)            | AD1.4                          | 00                        |
| 29:28                 | P0.14                  | GPIO Port 0.14                 | DCD (UART1)                    | EINT1                          | SDA1 (I <sup>2</sup> C1)       | 00                        |
| 31:30                 | P0.15                  | GPIO Port 0.15                 | RI (UART1)                     | EINT2                          | AD1.5                          | 00                        |

| <b>PINSEL1</b> | <b>Pin Name</b> | <b>Calculate</b>                       |
|----------------|-----------------|----------------------------------------|
| <b>00:01</b>   | <b>P0.16</b>    | <b><math>16-16=0 \times 2=0</math></b> |
| <b>02:03</b>   | <b>P0.17</b>    | <b><math>17-16=1 \times 2=2</math></b> |
| <b>04:05</b>   | <b>P0.18</b>    | <b><math>18-16=2 \times 2=4</math></b> |
| <b>06:07</b>   | <b>P0.19</b>    | <b><math>19-16=3 \times 2=6</math></b> |
| <b>08:09</b>   | <b>P0.20</b>    | <b><math>20-16=4 \times 2=8</math></b> |

| <b>PINSEL2</b> | <b>Pin Name</b> | <b>Calculate</b>                       |
|----------------|-----------------|----------------------------------------|
| <b>00:01</b>   | <b>P1.16</b>    | <b><math>16-16=0 \times 2=0</math></b> |
| <b>02:03</b>   | <b>P1.17</b>    | <b><math>17-16=1 \times 2=2</math></b> |
| <b>04:05</b>   | <b>P1.18</b>    | <b><math>18-16=2 \times 2=4</math></b> |
| <b>06:07</b>   | <b>P1.19</b>    | <b><math>19-16=3 \times 2=6</math></b> |
| <b>08:09</b>   | <b>P1.20</b>    | <b><math>20-16=4 \times 2=8</math></b> |

# GPIO Registers

---

- ❖ **IODIR** (IO direction register, IODIR0 for P0 & IODIR1 for P1) – The bit setting of this register configures the pin as input or output, 1 for output, 0 for input
- ❖ **IOSET** (IO set register, IOSET0 & IOSET1) – This register is used to set the output pins of the chip. To make a pin to be '1', the corresponding bit in the register is to be '1'. Writing zeros have no effect
- ❖ **IOCLR** (IO Clear register, IOCLR0 & IOCLR1) – To make an output pin to have a '0' value, i.e to clear it. The corresponding bit in this register has to be a '1'. Writing zeros have no effect.
- ❖ **IOPIN** (IO Pin register, IOPIN0 & IOPIN1): From this register the value of the corresponding pin can be read, irrespective of whether the pin is an input or output pin.



**Example1:** Set the lower sixteen GPIO pins of P0 to 1.

---

First set the direction, Set the output to 1,

`IODIR0 = 0x0000FFFF`

`IOSET0 = 0x0000FFFF`

## **Example2: Interface an LED to P0.10 and turn ON and OFF the LED.**

```
#include <LPC214X.H>
int main(void)
{
    unsigned int x;
    IODIR0 = 0xFFFFFFFF; //Make all the pins as outputs
    for(;;)
    {
        IOSET0 = 1 << 10; //Set the port pin P0.10
        for(x=0;x<30000;x++); //delay for ON time
        IOCLR0 = 1 << 10; //Clear the port pin P0.10
        for(x=0;x<40000;x++); // delay for OFF time
    }
}
```

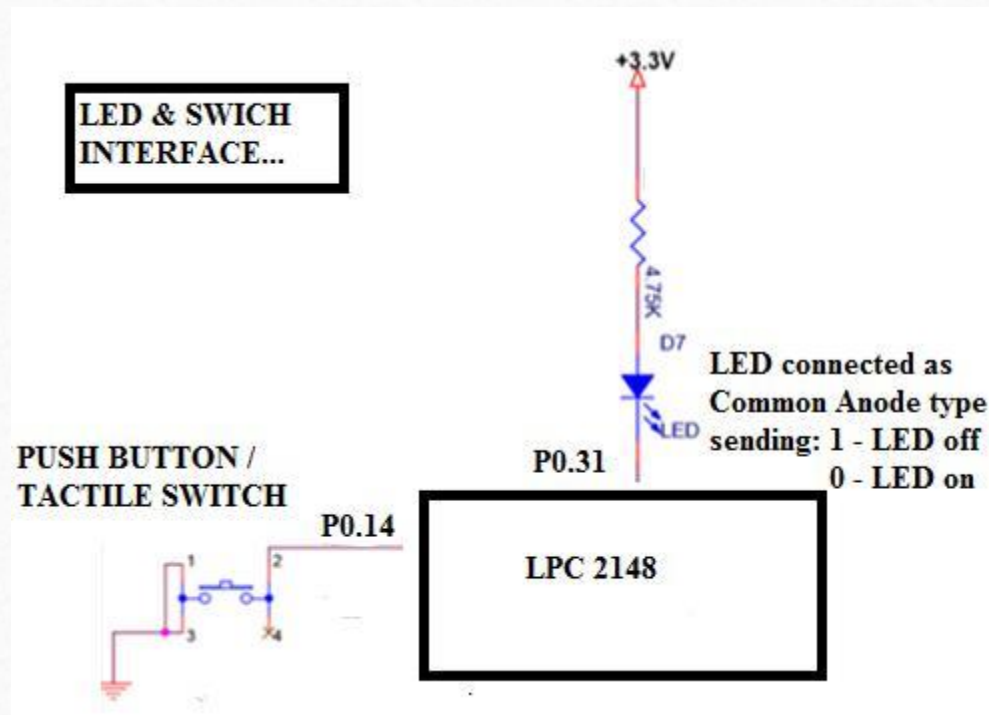
**Example2:** Generate the **asymmetric square wave** at the lowest four pins of Port0

```
#include <LPC214X.H>
int main(void)
{
    unsigned int x;
    IODIR0 = 0xFFFFFFFF;    //Make all the pins as outputs

    for(;;)
    {
        IOSET0 = 0x0000000F; //Set the port pins P0.0 to P0.3
        for(x=0;x<10000;x++);

        IOCLR0 = 0x0000000F; //Clear the port pins
        for(x=0;x<20000;x++);
    }
}
```

**Question 1: Interface an LED and Push button (press to on) Switch , and write a program to blink the led, when the switch is keep pressed.**



Assume P0.31 connected to LED, Assume P0.14 connected to Switch

Configure P0.31 as GPIO output, as LED is output device and P0.14 as GPIO input, as Switch is input device.

Working of LED - (common anode).

1 – Led off, 0 – Led on

Working of switch:

when switch is not pressed, P0.14 receives = 1,

when switch is pressed P0.14 receives = 0



```

#include <lpc214x.h>
#define LED_OFF (IO0SET = 1U << 31)
#define LED_ON (IO0CLR = 1U << 31)
#define SW2 (IO0PIN & (1 << 14))

void delay_ms(unsigned int j);

int main( )
{
    IO0DIR = 1U << 31;
    IO0SET = 1U << 31;
    while(1)
    {
        if (!(IO0PIN & (1 << 14)))    // (if(!SW2 )
        {
            IO0CLR = 1U << 31; //LED_ON
            delay_ms(250);
            IO0SET = 1U << 31; //LED_OFF
            delay_ms(250);
        }
    }
}

```

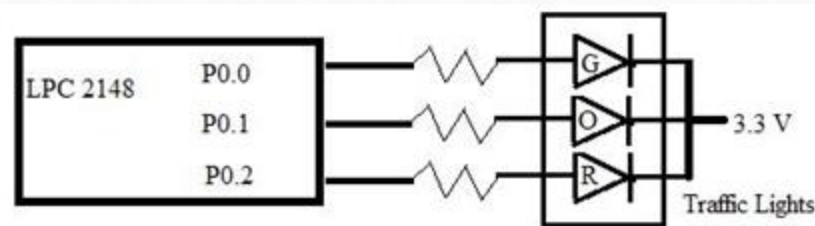
```

void delay_ms(unsigned int j) ; assume delay_ms(1) produces 1ms
{
    unsigned int x, i;
    for(i=0; i<j; i++)
    {
        for(x=0; x<1000; x++);
    }
}

```

## Interface 3LEDs (Red,Yellow,Green) to LPC 2148 and Write Embedded C program to simulate traffic light system.

- Choose any three port lines (say P0.0,P0.1,P0.2) to connect to 3 Leds to represent Traffic Lights, one end of the junction.
- **Sending '1' – LED is on, sending '0' – LED is off**



- Configure P0.0,P0.1,P0.2 as outputs, as LEDs are output devices.
-

```

#include <LPC214x.h>
#define val1 30000
#define val2 5000
#define val3 50000
void delay(unsigned long int);
int main(void)
{
    IODIR0 = 0X00000007; //make P0.0,P0.1,P0.2 as outputs
    while(1)
    {
        // set GREEN for 30seconds
        IOSET0 = 1 << 0; //set P0.0 to 1
        delay(val1);
        IOCLR0 = 1 << 0; //clear P0.0 to 0

        // set ORANGE/Yellow for 5 seconds
        IOSET0 = 1 << 1; //set P0.1 to 1
        delay(val2);
        IOCLR0 = 1 << 1; //clear P0.1 to 0

        // set RED for 105 seconds (3 x 35 = 105), assuming it
        //is part of junction with four roads
        IOSET0 = 1 << 2; //set P0.2 to 1
        delay(val3); delay(val3); delay(val2);
        IOCLR0 = 1 << 2; //clear P0.2 to 0
    }
}

```

```

void delay_ms(unsigned int j) ; assume delay_ms(1) produces 1ms
{
    unsigned int x, i;
    for(i=0; i<j; i++)
    {
        for(x=0; x<1000; x++);
    }
}

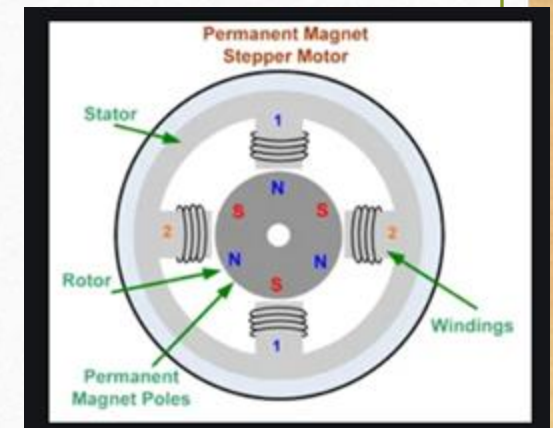
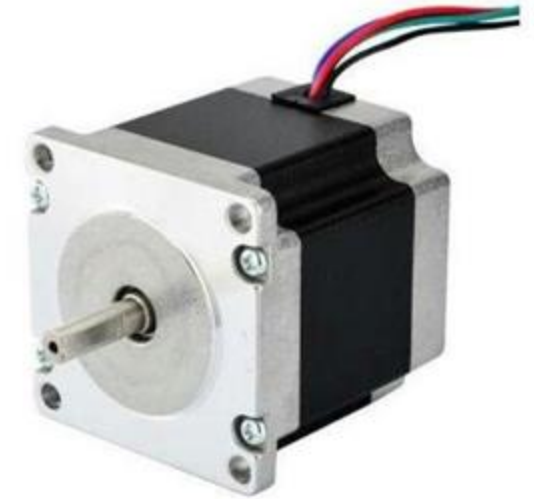
```

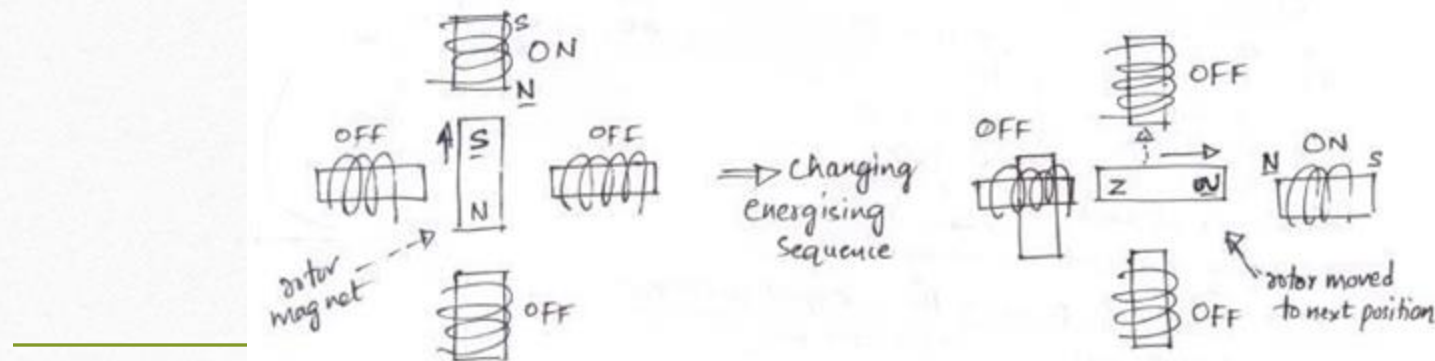
- **Interfacing and Programming (using embedded C)** with LEDs, Switches, Seven segment displays, LCD, Matrix Keypad, I2C based DAC, Stepper motor, DC Motor, Relay, Opto-isolators. Analog Interfacing using ADC Channels, interfacing with LDR and Temperature sensor.
- ❑ **Elevator Interface:** Write an Embedded C program to read the elevator switches and simulate elevator up and down movements.
- ❑ **Seven Segment Display Interface:** Write a C program to display messages “FIRE” & “HELP” on 4 digit seven segment display alternately with a suitable delay.
- ❑ **Stepper Motor Interface:** Write an Embedded C program to rotate stepper motor in clockwise direction for “M” steps, anti-clock wise direction for “N” steps.
- ❑ **DC Motor Interface:** Write an Embedded C program to generate PWM wave to control speed of DC motor. Control the duty cycle by analog input.
- ❑ **DAC Interface :** Write an Embedded C program to generate sine , full rectified ,triangular, sawtooth and square waveforms using DAC module
- ❑ **Matrix Keyboard Interface :**Write an embedded C program to interface 4 X 4 matrix keyboard using lookup table and display the key pressed on the Terminal.
- ❑ **Character LCD Interface :** Write an Embedded C program to display text messages on the multiple lines of the display.



## Stepper Motor Interfacing

- Stepper motors move in steps unlike dc motors, under the digital control
- Can digitally control:
  - The number of steps,
  - RPM (no. of revolutions per minute),
  - Direction of movement (clockwise /anticlockwise)
- Stepper motors finds application – in disk drives, printers, plotters, CNC machines, robotics etc
- **Resolution:** The term stepper motor resolution refers the angular movement, in degree per step.
- **Working principle:** Stepper motors – has two parts a) rotor made up permanent magnet or soft iron b) stator – surrounding the rotor. Stator is constructed with the number of coils / windings, which on energising (supplying current) acts like a electromagnet.

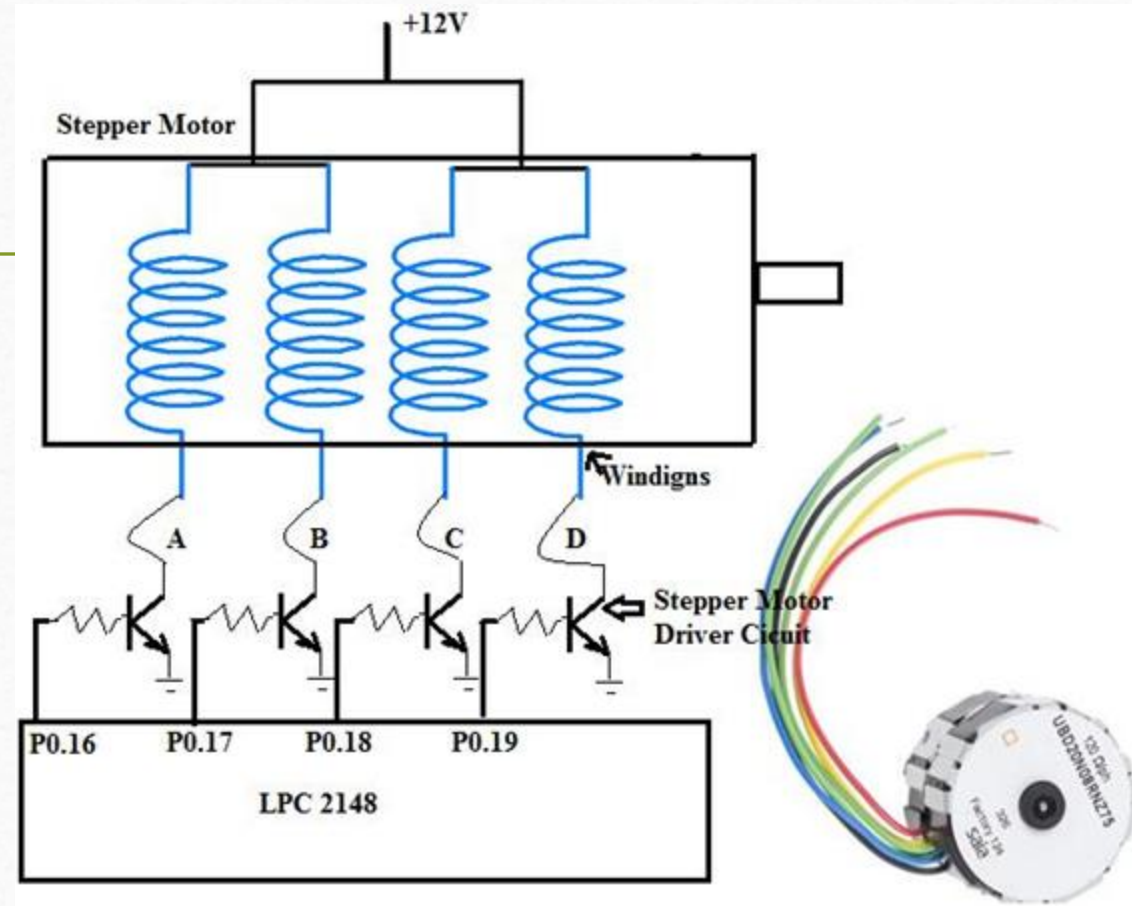




Rotor (magnet) aligns to a particular position based on the energised coil (acting as electromagnet) as S & N poles gets attracted when we change energisation sequence, rotor aligns to new position, that is how the stepper motors movement is realised.

- **Stepping Modes :** Stepper motors movement is achieved by employing one of the following coil energising sequences.
  - a) Full drive (2 coils are energised at a time, more torque) – 4 step sequence
  - b) Wave drive (one coil only energised at a time, less torque) – 4 step
  - c) Half drive (here step angle reduces by half, if 1.8° is step angle, here it becomes 0.9°. it is a combination of the above two types, hence 8 step sequence and torque is in between above types)

Commonly available (2Phase, 4winding stepper motors) comes with 5/6 wire connector, 4 for windings, namely A,B,C,D and one/two connections for Power as shown in the figure below





Energising or Driving a winding, means making the current flow in the winding. Applying logic '1' to the input of ULN2803 driver, switches on the transistor present inside the driver and the current flows through the corresponding winding, from the +12V supply

- Following table indicates the sequence in which, windings are to be energised to achieve clock wise and anticlockwise direction. Each sequence drives the motor one step (i.e 1.8degree), after 4 steps repeat the sequence to continue rotation in the same direction.

| <i>A B C D</i> | <i>A B C D</i>   | Note : Full drive sequence | <i>A B C D</i> |
|----------------|------------------|----------------------------|----------------|
| 1 0 0 0        | 0 0 0 1          |                            | 1 0 0 1        |
| 0 1 0 0        | 0 0 1 0          |                            | 1 1 0 0        |
| 0 0 0 1        | 0 1 0 0          |                            | 0 1 1 0        |
|                | 1 0 0 0          |                            | 0 0 1 1        |
| → clockwise    | ← anticlock wise |                            |                |

(energising the windings one by one; 1 – energise the winding, 0 – winding not energised)



## **Embedded C Program / Driver Program for Stepper Motor: Program to rotate stepper motor in clockwise direction for “M” steps, anti-clock wise direction for “N” steps.**

---

- Total number of steps for one revolution = 200 steps (200 teeth shaft)

$$\text{Step angle} = 360^\circ / 200 = 1.8^\circ$$

- Use appropriate delay in between consequent steps, to achieve required RPM
- 2Phase, 4winding stepper motor is used, along with driver circuit built using the ULN 2803, 12v power is used to drive the stepper motor. Digital input generated by the microcontroller, is used to drive and control the direction and rotation of stepper motors.

*//P0.16 to P0.19 are connected to Windings of SMotor*

```
#include <lpc214x.h>

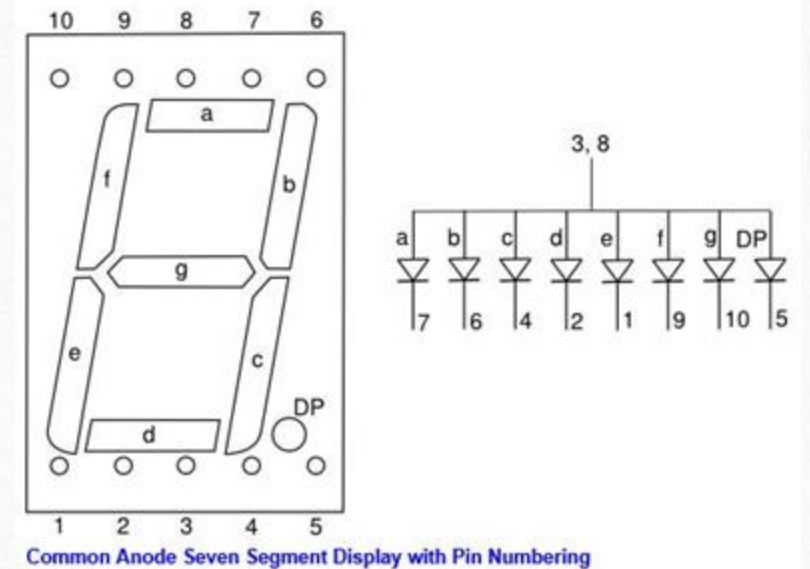
void delay_ms(unsigned int j);
int main()
{
    unsigned int no_of_steps_clk = 100, no_of_steps_aclk = 100;
    IO0DIR |= 0x00FF0000; // to set P0.16 to P0.23 as o/ps do{

    IO0CLR = 0X000F0000;IO0SET = 0X00010000;delay_ms(10);if(--no_of_steps_clk == 0) break;
    IO0CLR = 0X000F0000;IO0SET = 0X00020000;delay_ms(10);if(--no_of_steps_clk == 0) break;
    IO0CLR = 0X000F0000;IO0SET = 0X00040000;delay_ms(10);if(--no_of_steps_clk == 0) break;
    IO0CLR = 0X000F0000;IO0SET = 0X00080000;delay_ms(10);if(--no_of_steps_clk == 0) break;
    }while(1);
    do{
        IO0CLR = 0X000F0000;IO0SET = 0X00080000;delay_ms(10);if(--no_of_steps_aclk == 0) break;
        IO0CLR = 0X000F0000;IO0SET = 0X00040000;delay_ms(10);if(--no_of_steps_aclk == 0) break;
        IO0CLR = 0X000F0000;IO0SET = 0X00020000;delay_ms(10);if(--no_of_steps_aclk == 0) break;
        IO0CLR = 0X000F0000;IO0SET = 0X00010000;delay_ms(10);if(--no_of_steps_aclk == 0) break;
    }while(1);
    IO0CLR = 0X00FF0000;
    while(1);
}

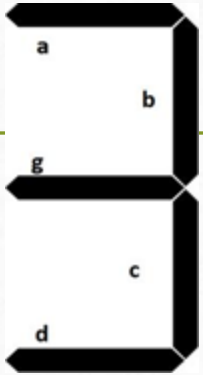
void delay_ms(unsigned int j)
{
    unsigned int x,i;
    for(i=0;i<j;i++)
    {
        for(x=0; x<10000; x++);
    }
}
```

## Seven Segment Display Interface

- Seven Segment displays, are more preferred compared to normal LEDs
- These displays are visible during all the times of a day, can be seen from long distance and very cost effective
- One 8 bit shift register is required to drive one seven segment display, as each seven segment display requires 8 bits of information, to drive 8 segments a to dp
- The seven segment displays are available in two types,
  - common anode display and common cathode display
- In common anode display, applying '0' voltage to the cathodes of the segments, will enable the segment to glow.



To display '3', we have to send following bit pattern



This is 'B0' in hexadecimal

| DP | G | f | e | d | c | b | a |
|----|---|---|---|---|---|---|---|
| 1  | 0 | 1 | 1 | 0 | 0 | 0 | 0 |



```
int main()

{
    IO0DIR |= 1U << 31 | 1U << 19 | 1U << 20 | 1U << 30 ;
    // to set as o/ps


---


    LED_ON; // make D7 Led on .. just indicate the
    program is running
    SystemInit();
    while(1)
    {
        alphadisp7SEG("fire ");
        delay_ms(500);
        alphadisp7SEG("help ");
        delay_ms(500);
    }
}
```

```
unsigned char getAlphaCode(unsigned char alphachar)
```

```
{
```

```
    switch (alphachar)
```

```
    {
```

```
// dp g f e d c b a - common anode: 0 segment on, 1 segment off case 'f': return 0x8e;
```

```
    case 'i': return 0xf9;
```

```
    case 'r': return 0xce;
```

```
    case 'e':return 0x86; // 1000 0110 case 'h':return 0x89;
```

```
    case 'l': return 0xc7; case 'p':return 0x8c; case ' ': return 0xff;
```

```
    //similarly add for other digit/characters default : break;
```

```
    }
```

```
    return 0xff;
```

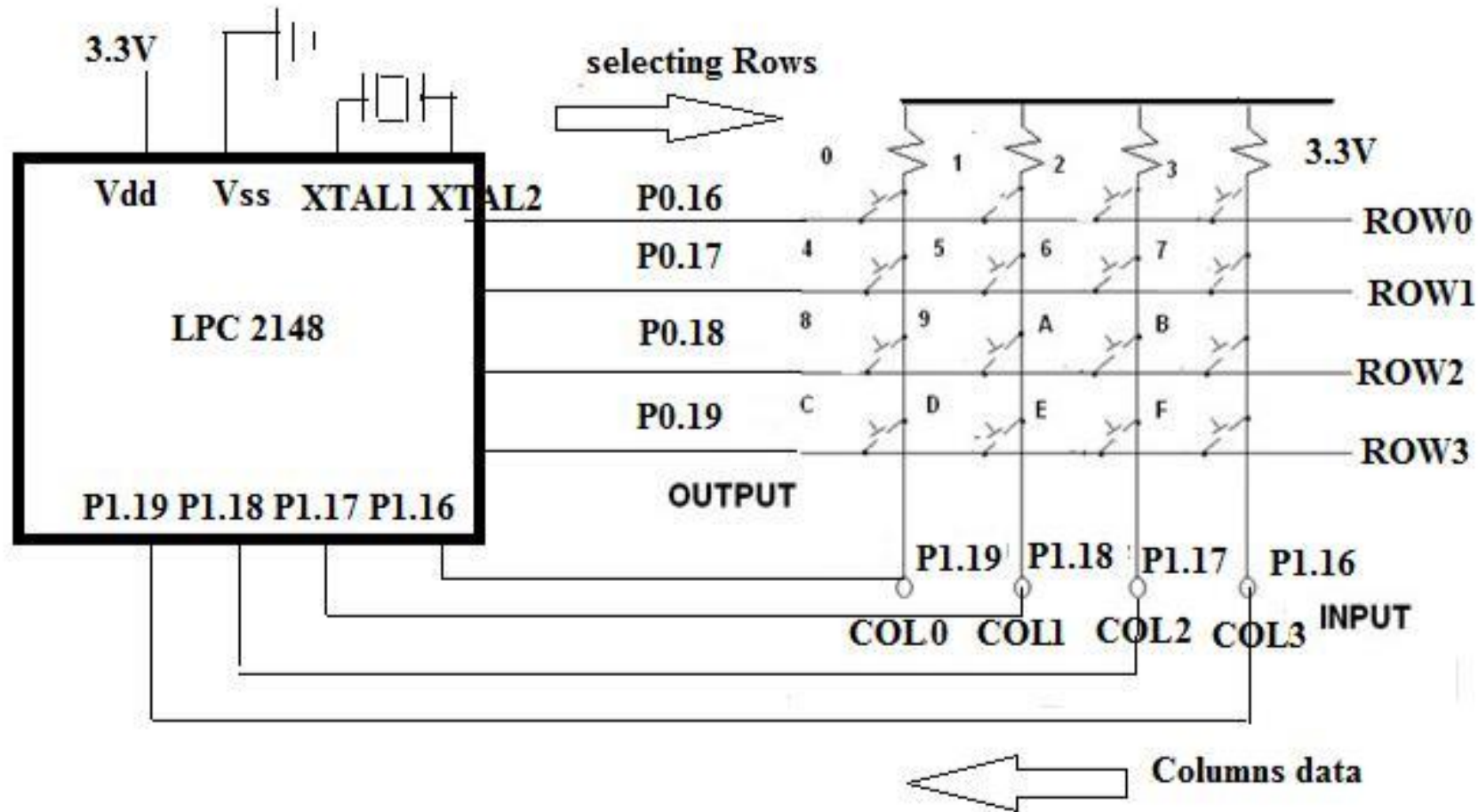
```
}
```

```

void alphadisp7SEG(char *buf)
{
    unsigned char i,j;
    unsigned char seg7_data,temp=0;
    for(i=0;i<5;i++) // because only 5 seven segment digits are present
    {
        seg7_data = getAlphaCode(*(buf+i));
        // instead of this look up table can be used
        // to shift the segment data(8bits)to the hardware (shift registers) using Data,Clock,Strobe
        for (j=0 ; j<8; j++)
        {
            //get one bit of data for serial sending
            temp = seg7_data & 0x80; // shift data from Most significant bit (D7) if(temp == 0x80)
            IOSET0 |= 1 << 19; //IOSET0 / 0x00080000;
            else
                IOCLR0 |= 1 << 19; //IOCLR0 / 0x00080000; //send one clock pulse
            IOSET0 |= 1 << 20; //IOSET0 / 0x00100000; delay_ms(1);
            IOCLR0 |= 1 << 20; //IOCLR0 / 0x00100000;
            seg7_data = seg7_data << 1; // get next bit into D7 position
        }
        // send the strobe signal
        IOSET0 |= 1 << 30; //IOSET0 | 0x40000000;
        delay_ms(1); //nop();
        IOCLR0 |= 1 << 30; //IOCLR0 / 0x40000000;
        return;
    }
}

```

**Matrix Keyboard Interface: Write an embedded C program to interface 4 X 4 matrix keyboard using lookup table and display the key pressed on the Terminal**





## Embedded Software / Driver for reading matrix keyboard

Algorithm for reading matrix keyboard / Working method:

- If no key is pressed, we will have on columns 0-3, '1111' on P1.16 to P1.19, as all the inputs are pulled up by pull up resistors.
- If we press any key, let '0' key be pressed, it will short row0 and col0 lines (P0.16 & P1.19), so whatever data (0 or 1) available at row0 (P0.16) is available at col0 (P1.19). Since already columns are pulled high, it is required to apply logic '0' to see change in col0 when the key is pressed.
- To identify which key is pressed, perform the following steps in the loop,
  - Check for a key press in first row by out putting – '0111'on row's, check which column data is changed, if no key press go for next row. If key press found, exit loop with row no and column no.
  - Check for a key press in second row by out putting – '1011'on row's, check which column data is changed, if no key press go for next row. If key press found, exit loop with row no and column no.
  - Check for a key press in third row by out putting – '1101'on row's, check which column data is changed, if no key press go for next row. If key press found, exit loop with row no and column no.
  - Check for a key press in last row by out putting – '1110'on row's, if no key is pressed go for the first row again. If key press found, exit loop with row no and column no.
- Once the key press is found, use the row number and column number and use the "look up table" to convert the key position(row no : column no) to corresponding ascii code/key code related to that key. Use appropriate delay for debouncing.
- Output the key code on the serial port or it can be used for further processing
- Wait for the pressed key release; be in the loop until all the columns data become "1111"

*//Matrix 4 x 4 Keyboard*

*//Columns & Rows are pulled to +5v,if dont press key, we receive '1' on columns*

*//Method: Sending '0' to a selected row, checking for '0' on each column*

*//ROWS - ROW0-ROW3 -> P0.16,P0.17,P0.18,P0.19*

---

*//COLS - COL0-COL3 -> P1.19,P1.18,P1.17,P1.16*

#include <lpc214x.h>

#define PLOCK 0x00000400

#define LED\_OFF (IO0SET = 1U << 31)

#define LED\_ON (IO0CLR = 1U << 31)

#define COL0 (IO1PIN & 1 <<19)

#define COL1 (IO1PIN & 1 <<18)

#define COL2 (IO1PIN & 1 <<17)

#define COL3 (IO1PIN & 1 <<16)

unsigned char lookup\_table[4][4]={ { '0', '1', '2','3'},  
                                  { '4', '5', '6','7'},  
                                  { '8', '9', 'a','b'},  
                                  { 'c', 'd', 'e','f'}};

```
int main( )
{
    SystemInit();
    uart_init();//initialize UART0 port
    IO0DIR |= 1U << 31 | 0x00FF0000; // to set P0.16 to P0.23 as o/ps
do
{
    while(1)
    {
        //check for keypress in row0,make row0 '0',row1=row2=row3='1'
        rowssel=0;IO0SET = 0X000F0000;IO0CLR = 1 << 16;
        if(COL0==0){colsel=0;break;};if(COL1==0){ colsel=1;break;};
        if(COL2==0){colsel=2;break;};if(COL3==0){ colsel=3;break;};
        //check for keypress in row1,make row1 '0'
        rowssel=1;IO0SET = 0X000F0000;IO0CLR = 1 << 17;
        if(COL0==0){colsel=0;break;};if(COL1==0){ colsel=1;break;};
        if(COL2==0){colsel=2;break;};if(COL3==0){ colsel=3;break;};
```



```

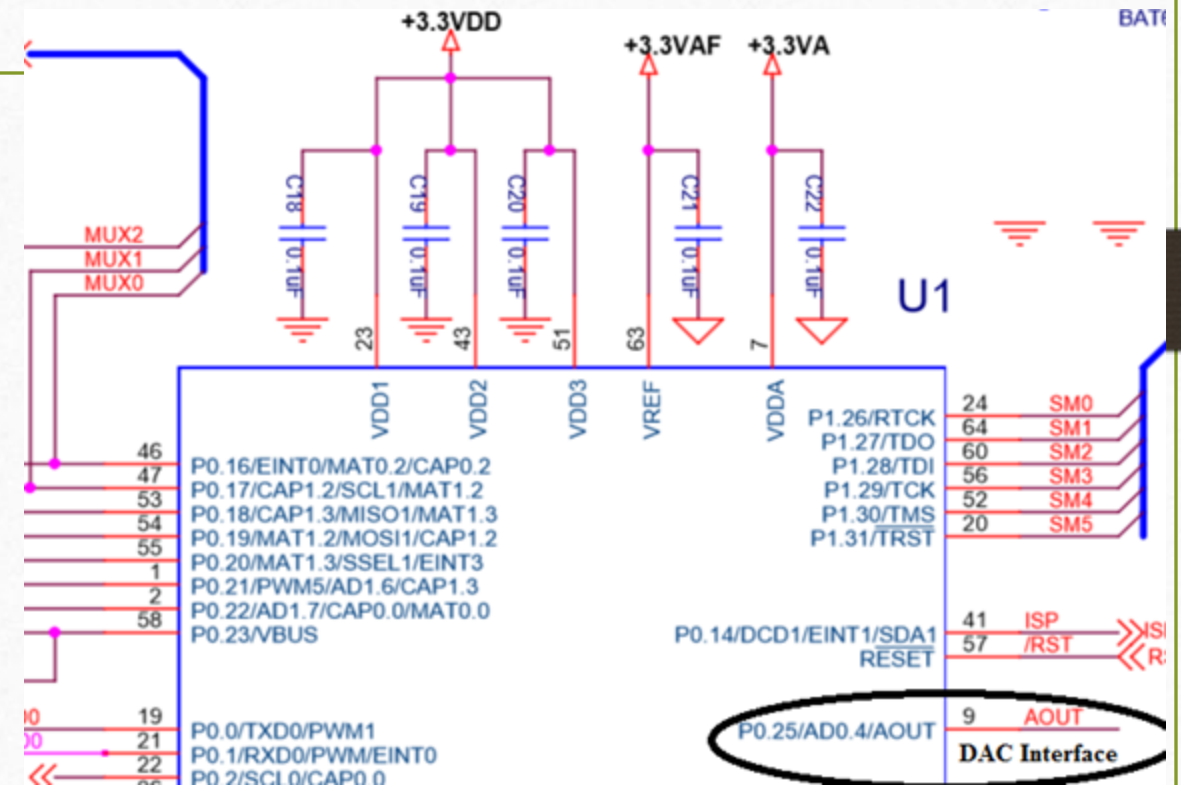
//check for keypress in row2,make row2 '0'
rowssel=2;IO0SET = 0X000F0000;IO0CLR = 1 << 18;//make row2 '0'
if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};
if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;}; //check for keypress in
row3,make row3 '0'
rowssel=3;IO0SET = 0X000F0000;IO0CLR = 1 << 19;//make row3 '0'
if(COL0==0){colsel=0;break;};if(COL1==0){colsel=1;break;};
if(COL2==0){colsel=2;break;};if(COL3==0){colsel=3;break;};
};
delay_ms(50); //allow for key debouncing
while(COL0==0 || COL1==0 || COL2==0 || COL3==0);//wait for key release delay_ms(50);
//allow for key debouncing
IO0SET = 0X000F0000; //disable all the rows
U0THR = lookup_table[rowssel][colsel]; //send to serial port(check on the terminal)
}
while(1);
}

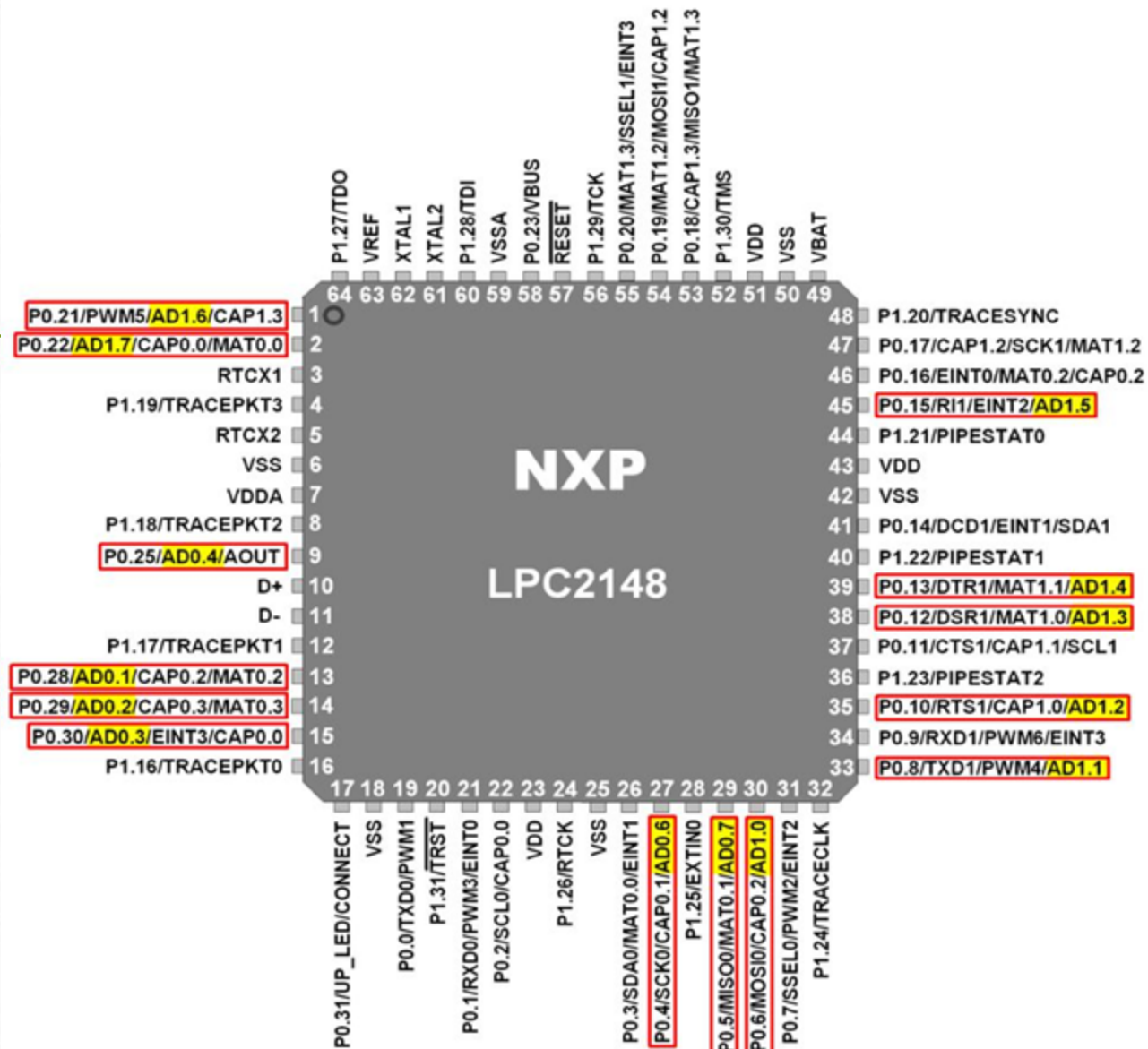
```



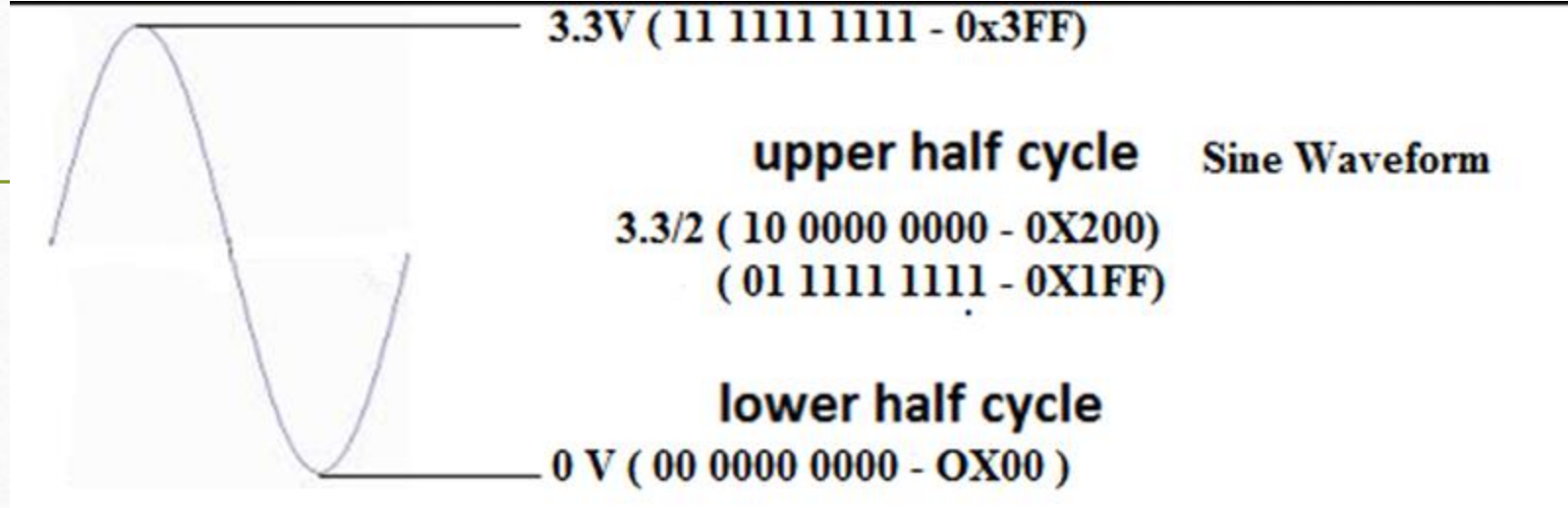
## DAC Interface : Write an Embedded C program to generate sine, full rectified sine, Triangular, Sawtooth and Square waveforms using DAC module

- DAC module of LPC 2148 is 10 bit Digital to Analog converter used to convert 10 bit Digital data to corresponding Analog voltage.
- Digital I/P : 000 to 3FF (0 to 1023), corresponding Analog O/P : 0V to 3.3V
- Resolution =  $(3.3/1024) \approx 3.2\text{mV}$





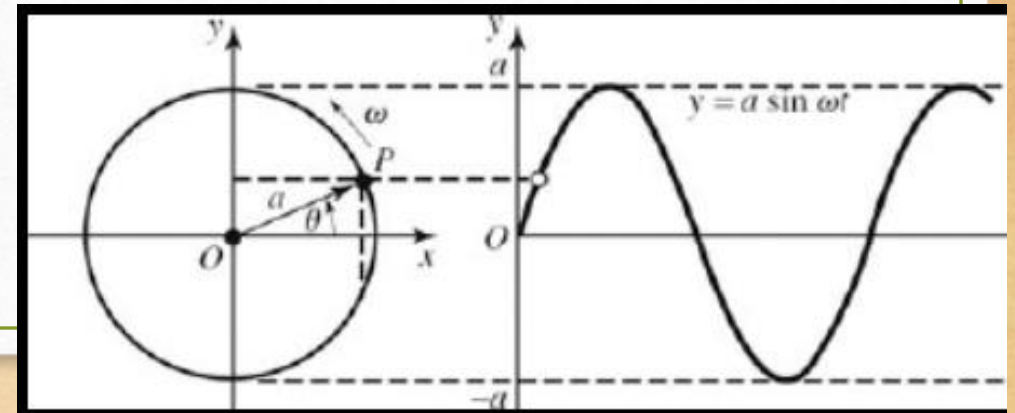
**Look up Table Creation:** Look up tables are used extensively in embedded systems, to store precomputed digital values, corresponding to analog voltages and used to generate different waveforms using DAC Module.



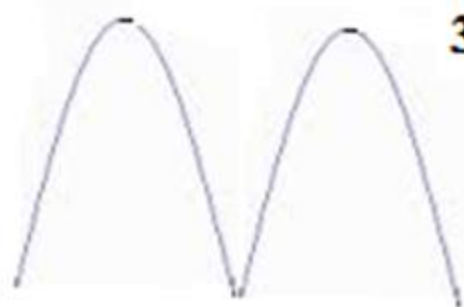
**Formula for calculation of the sine table entries:  $512 + 511 \times \sin \theta$**

**(512 Corresponds to 1FFh, i.e.  $3.3/2$  V,  $511 \times \sin 90$  gives 511, so  $512 + 511 = 1023$  (for 3.3V). Calculate the digital values to be outputted to DAC for angles in the steps of  $6^\circ$ ,**

|                                              |                                              |
|----------------------------------------------|----------------------------------------------|
| <b><math>511 \times \sin 0 = 0</math></b>    | <b><math>511 \times \sin 48 = 380</math></b> |
| <b><math>511 \times \sin 6 = 53</math></b>   | <b><math>511 \times \sin 54 = 413</math></b> |
| <b><math>511 \times \sin 12 = 106</math></b> | <b><math>511 \times \sin 60 = 442</math></b> |
| <b><math>511 \times \sin 18 = 158</math></b> | <b><math>511 \times \sin 66 = 467</math></b> |
| <b><math>511 \times \sin 24 = 208</math></b> | <b><math>511 \times \sin 72 = 486</math></b> |
| <b><math>511 \times \sin 30 = 256</math></b> | <b><math>511 \times \sin 78 = 503</math></b> |
| <b><math>511 \times \sin 36 = 300</math></b> | <b><math>511 \times \sin 84 = 510</math></b> |
| <b><math>511 \times \sin 42 = 342</math></b> | <b><math>511 \times \sin 90 = 511</math></b> |







3.3V ( 11 1111 1111 - 0x3FF)

Full Rectified sinewaveform

0 V ( 00 0000 0000 - 0X00 )

(11 1111 1111 - 0X3FF)

Triangular Waveform

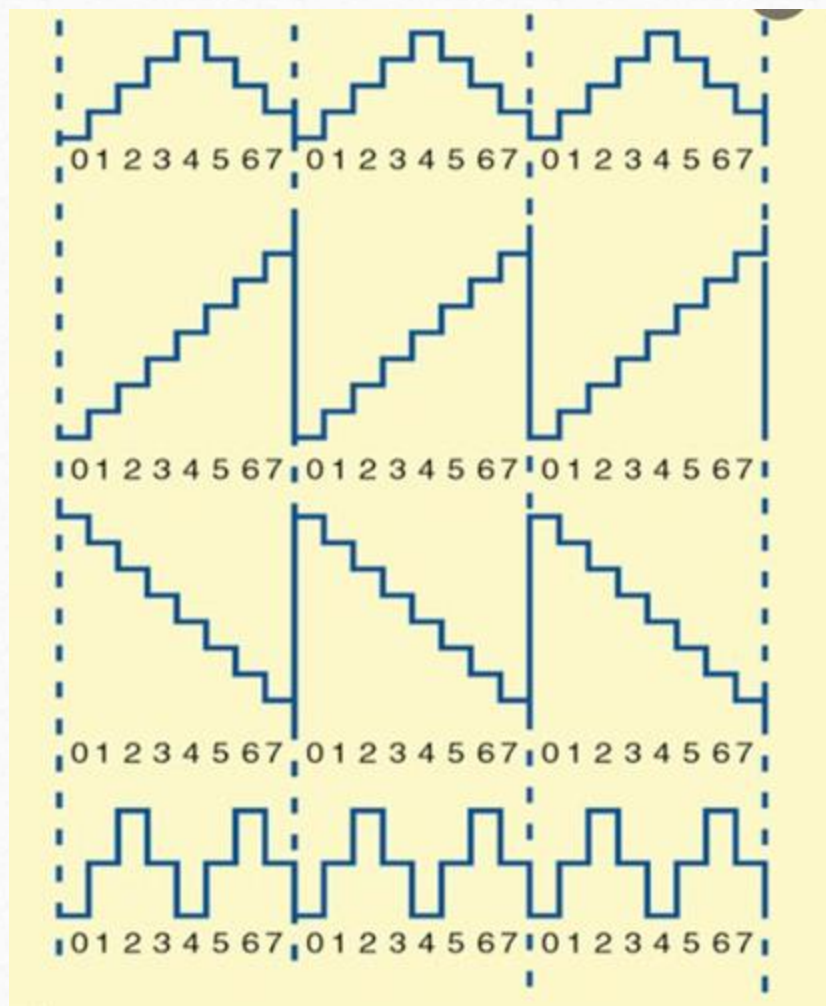
(00 0000 0001 - 0X001)  
(00 0000 0000 - 0X000)

(11 1111 1111 - 0X3FF)

Sawtooth waveform

(00 0000 0000 - 0X000)





//DAC: LAB PROGRAM

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

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

---

```
#define PLOCK 0x00000400
```

```
#define LED_OFF (IO0SET = 1U << 31)
```

```
#define LED_ON (IO0CLR = 1U << 31)
```

```
#define SW2 (IO0PIN & (1 << 14))
```

```
#define SW3 (IO0PIN & (1 << 15))
```

```
#define SW4 (IO1PIN & (1 << 18))
```

```
#define SW5 (IO1PIN & (1 << 19))
```

```
#define SW6 (IO1PIN & (1 << 20))
```

```
void SystemInit(void);
```

```
static void delay_ms(unsigned int j); //millisecond delay
```

```
short int sine_table[ ] =  
{512+0,512+53,512+106,512+158,512+208,512+256,512+300,512+342,512+380,512  
+413,512+442,512+467,512+486,512+503,512+510,512+511,  
512+510,512+503,512+486,512+467,512+442,512+413,512+380,512+342,512+300,5  
12+256,512+208,512+158,512+106,512+53,512+0,  
512-53,512-106,512-158,512-208,512-256,512-300,512-342,512-380,512-413,512-  
442,512-467,512-486,512-503,512-510,512-511,  
512-510,512-503,512-486,512-467,512-442,512-413,512-380,512-342,512-300,512-  
256,512-208,512-158,512-106,512-53};  
short int sine_rect_table[ ] =  
{512+0,512+53,512+106,512+158,512+208,512+256,512+300,512+342,512+380,512  
+413,512+442,512+467,512+486,512+503,512+510,512+511,  
512+510,512+503,512+486,512+467,512+442,512+413,512+380,512+342,512+300,5  
12+256,512+208,512+158,512+106,512+53,512+0};
```

```
int main()
{
    short int value,i=0;
    SystemInit();
    PINSEL1 |= 0x00080000;    /* P0.25 as DAC output :option 3 - 10 (bits18,19)*/
    IO0DIR |= 1U << 31 | 0x00FF0000 ; // to set P0.16 to P0.23 as o/ps
```

---

```
while(1)
{
    if (!SW2)        /* If switch for sine wave is pressed */
    {
        while (i!=60 )
        {
            value = sine_table[i++];
            DACR = ( (1<<16) | (value<<6) );
            delay_ms(1);
        }
        i=0;
    }
}
```



```

else if (!SW3)
{
    while ( i!=30 )
    {
        value = sine_rect_table[i++];
        DACR = ( (1<<16) | (value<<6) );
        delay_ms(1);
    }
    i=0;
}
else if ( !SW4)      /* If switch for triangular wave is pressed */
{
    value = 0;
    while ( value != 1023 )
    {
        DACR = ( (1<<16) | (value<<6) );
        value++;
    }
    while ( value != 0 )
    {
        DACR = ( (1<<16) | (value<<6) );
        value--;
    }
}
}

```

```
else if ( !SW5 )    /* If switch for sawtooth wave is pressed */
{
    value = 0;
    while ( value != 1023 )
    {
        DACR = ( (1<<16) | (value<<6) );
        value++;
    }
}
else if ( !SW6 )    /* If switch for square wave is pressed */
{
    value = 1023;
    DACR = ( (1<<16) | (value<<6) );
    delay_ms(1);
    value = 0;
    DACR = ( (1<<16) | (value<<6) );
    delay_ms(1);
}
```

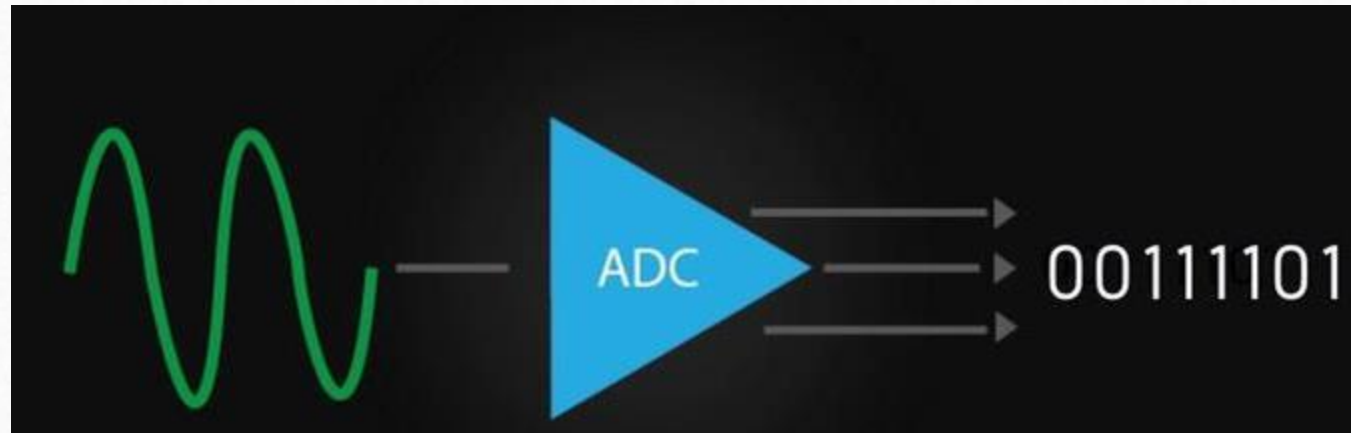
```
else    /* If no switch is pressed, 3.3V DC */
{
    value = 1023;
    DACR = ( (1<<16) | (value<<6) );
}
}
}
```

---

```
void delay_ms(unsigned int j)
{
    unsigned int x,i;
    for(i=0;i<j;i++)
    {
        for(x=0; x<10000; x++);
    }
}
```

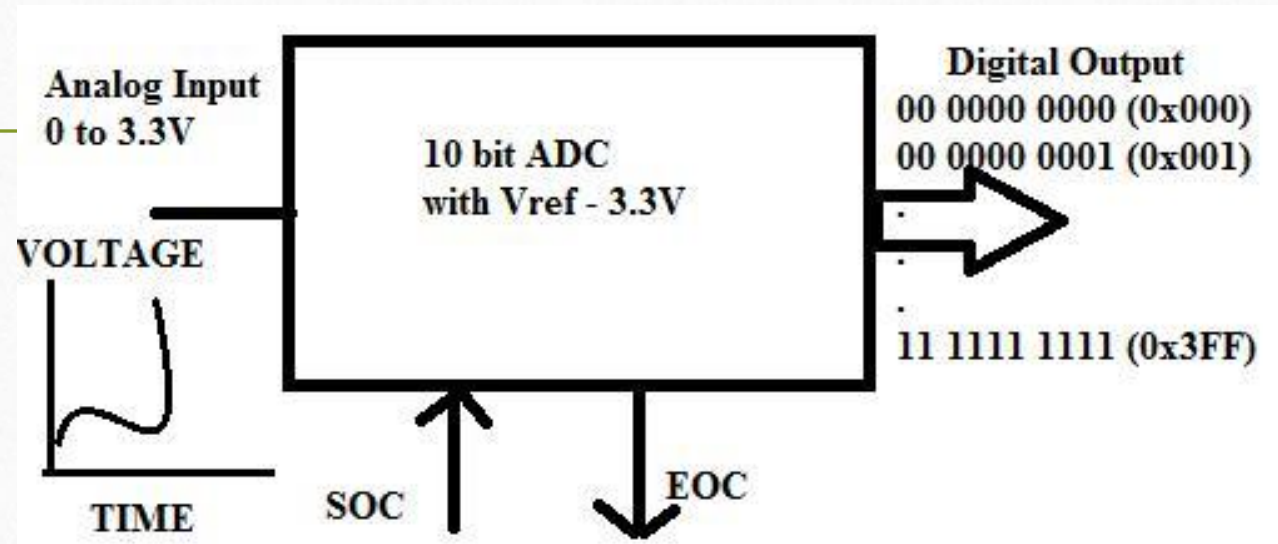
## ADC interfacing

- ADCs finds applications in weight measurement, temperature measurement, speed measurement etc.
- In general, different physical quantities like temperature, humidity are converted by sensors in to electrical domain i.e voltage or current
- Which in turn fed to ADC for conversion to digital domain, for doing different types of processing.





A typical ADC has analog input (one which is varying with time), and corresponding digital output, as shown below.



SOC – Start of conversion, indicates initiation of conversion process.

EOC – End of Conversion signal, indicates conversion of analog input to digital data is completed.

## Resolution of ADC

### 1. For 8 bit ADC ,

- analog input – 0 to 5V
  - digital input - 00 to FF,
  - Step size =  $5 \text{ V} / 2^8 = 20 \text{ mv (appr), Resolution}$
- 

- that means minimum of 20 mv is required to produce 1 bit change at the output
- For a given analog input,
  - digital\_value =  $(\text{input} / 5) \times 256$  [ = analog\_input / step size ]

### 2. For 10 bit ADC like LPC 2148,

- analog input – 0 to 3.3v
- digital output – 000 to 3FF,
- Step size =  $3.3 \text{ V} / 2^{10} = 3.2 \text{ mv (appr), Resolution}$
- that means minimum of 3.2 mv is required to produce 1 bit change at the output.
- For a given analog input,
  - digital\_value =  $(\text{input} / 3.3) \times 1024$  [ = analog\_input / step size ]

## **ADC Module of LPC 2148:**

### **Analog Signal Interfacing using LPC 2148 Features**

- LPC 2148 provides two ADC's / Analog channels: ADC0 & ADC1
- ADC0 provides 6 analog inputs, ADC1 provides 8 analog inputs

---

- The features of ADC are:
- It is of 10 bit successive approximation type analog to digital converter (2nos)
- Input multiplexing among 6 or 8 pins (ADC0 & ADC1)
- POWER-DOWN mode
- Measurement range 0 to Vref (typically 3V)
- 10 bit conversion time  $\geq 2.44\mu\text{s}$
- Burst conversion mode or single or multiple inputs



Pins provided by ADC:

**ADC0 CHANNEL** (6 analog inputs)

AD0.7 (P0.5),

AD0.6 (P0.4),

AD0.4 (P0.25),

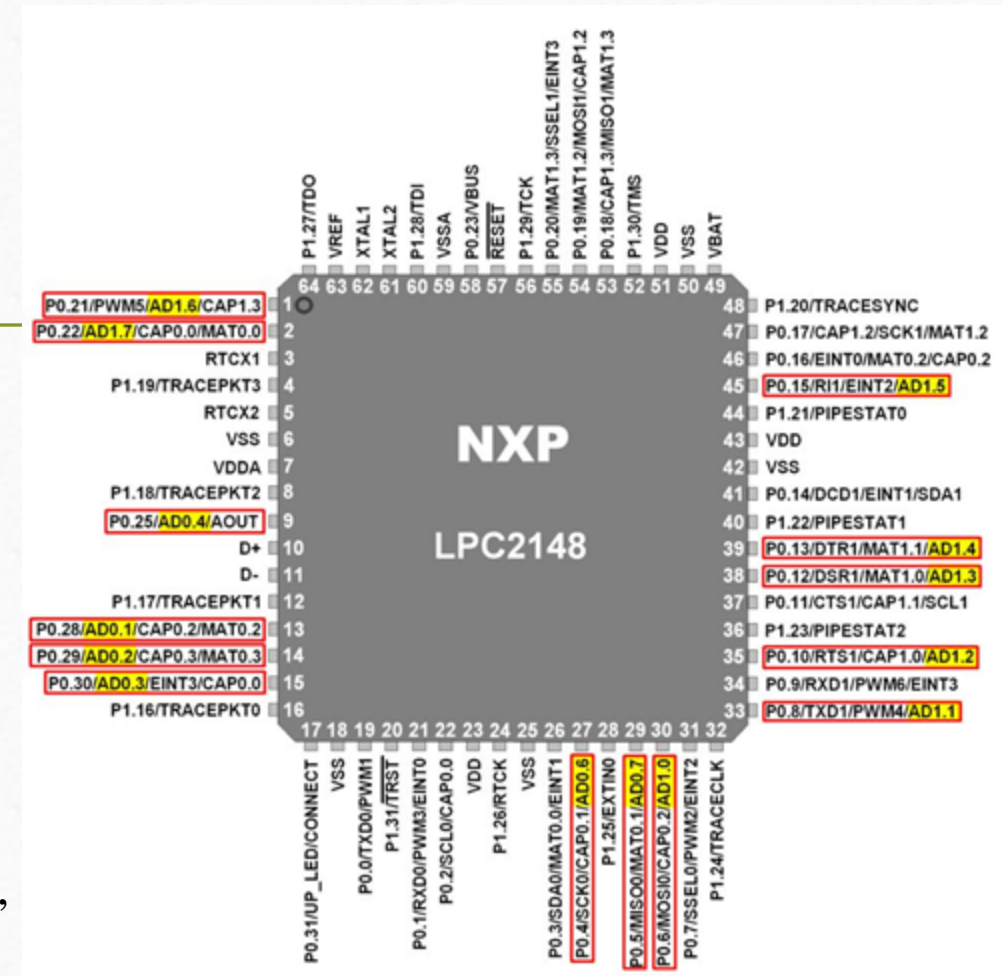
AD0.3 (P0.30),

AD0.2 (P0.29),

AD0.1 (P0.28),

**ADC1 CHANNEL** (8 analog inputs),

AD1.0 to AD1.7 are the input analog pins and the corresponding port numbers are [ P0.6, P0.8 ,P0.10, P0.12, P0.13,P0.15, P0.21, P0.22 ]





| Register Name  | Register Function                                                                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ADCR</b>    | A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.                                                                              |
| <b>ADGDR</b>   | A/D Global Data Register. This register contains the ADC's DONE bit and the result of the most recent A/D conversion.                                                                              |
| <b>ADSTAT</b>  | A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt flag.                                                                 |
| <b>ADGSR</b>   | A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt flag.                                                                 |
| <b>ADINTEN</b> | A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt. |
| <b>ADDRX</b>   | A/D Channel X Data Register. This register contains the result of the most recent conversion completed on channel X.                                                                               |

| ADC 0         |              | ADC 1         |              |
|---------------|--------------|---------------|--------------|
| ADC Channel 0 | LPC2148 Pins | ADC Channel 1 | LPC2148 Pins |
| <b>AD0.1</b>  | <b>P0.28</b> | <b>AD1.0</b>  | P0.6         |
| <b>AD0.2</b>  | P0.29        | <b>AD1.1</b>  | P0.8         |
| <b>AD0.3</b>  | P0.30        | <b>AD1.2</b>  | P0.10        |
| <b>AD0.4</b>  | P0.25        | <b>AD1.3</b>  | P0.12        |
| <b>AD0.6</b>  | P0.4         | <b>AD1.4</b>  | P0.13        |
| <b>AD0.7</b>  | P0.5         | <b>AD1.5</b>  | P0.15        |
|               |              | <b>AD1.6</b>  | P0.21        |
|               |              | <b>AD1.7</b>  | P0.22        |

## ADC Control Register (32 bit Register) :

**Bits - 7 to 0: ( SEL )** Selects which of the AD0.7 : AD0.0 / AD1.7 :AD1.0 to be sampled and converted.

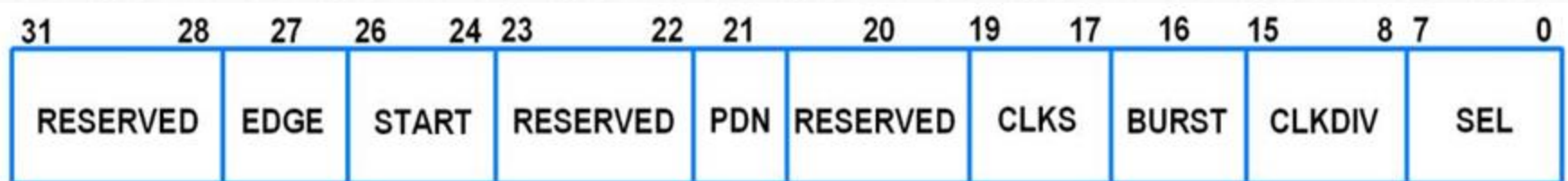
Example: for ADC0, bit 0 selects Pin AD0.0, and bit 7 selects AD0.7. In software-controlled mode, any one of these bits should be set to 1.

**Bits - 15 to 8: (CLKDIV)**- PCLK is divided by (this value + 1) to get ADC clock, which should be less than or equal to 4.5MHz

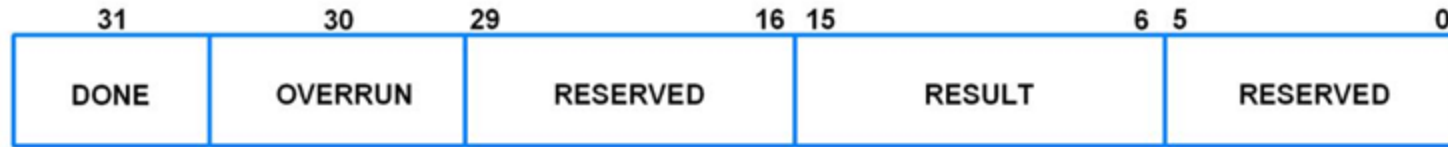
**Bits – 21 : (PDN) :** 1 – The ADC is operational , 0 – the ADC is in Power Down mode

**Bits – 26 to 24 : 001** Start Conversion Now

( **Note: Unused/reserved bits should not be written with 1** )



## ADC Global Data Register (32 bit Register)



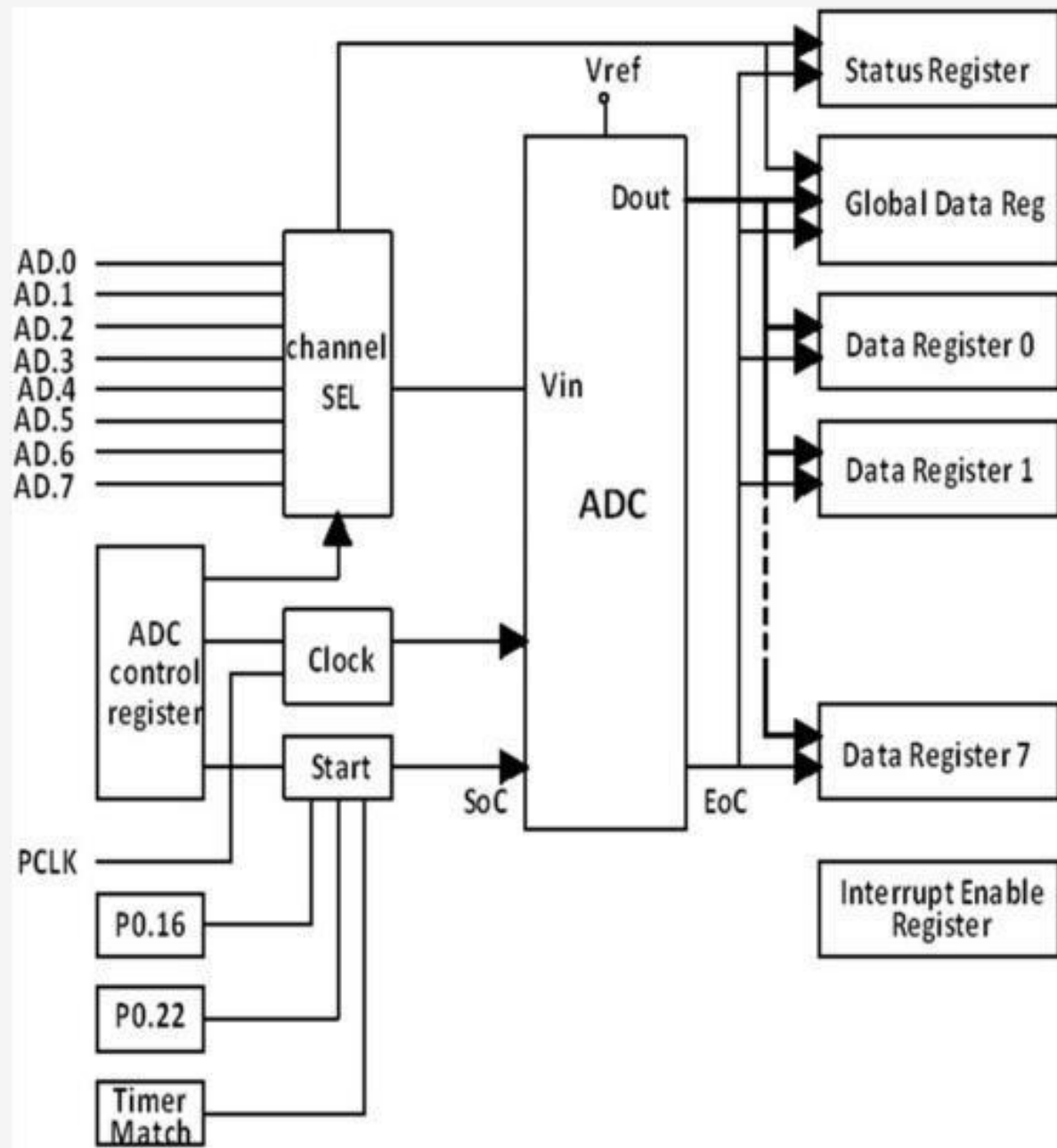
This register contains the ADC's DONE bit and the result of the most recent A/D conversion.

**Bit 5:0** - Reserved

**Bits 15:6** - Result ,when DONE bit is set to 1, this field contains 10 bit ADC result 0 to 1023, correspond to 0V to 3.3V ( Vref )

**Bit 31 - DONE**, This bit set to 1, when conversion completes, It is cleared when this register is read and when the AD0CR is written





## Program Steps to Read Analog Input :

1. Select the analog input pin (1 to 6 of ADC0 or 0 to 7 of ADC1) and configure the pin as Analog input using PINSEL register
2. Write to ADC control Register:

- Make ADC operational (set the Bit 21)
- Select the channel(set any one bit of Bit0 to Bit7 – for AD0 to AD7)
- Issue SOC signal (set 001 on bits : 26 25 24)

Example: select AD0.1

`AD0CR = (1 << 1) | (1 << 21) | (1 << 24); // assuming other bits are zero`

3. Check for the conversion to complete, by reading Bit31 of GDR

Example: for ADC0

`while( (AD0GDR & (unsigned long) 1 << 31) == 0);`

4. Read the Digital output from GDR, after aligning the result to LSB (Bit0) and masking other bits to Zero

Example:

`i. = (AD0GDR >> 6) & 0x3FF;`

### Example Program :

**LDR (Light Dependant Resistor ) / PhotoDiode is connected to P0.28 and LED is connected to P0.21, Read the Light intensity. Make the LED on whenever there is less/no light.**

```
#include <lpc214x.h>
#define LED_ON IO0SET = 1 << 21
#define LED_OFF IO0CLR = 1 << 21
int main()
{
    unsigned int i;

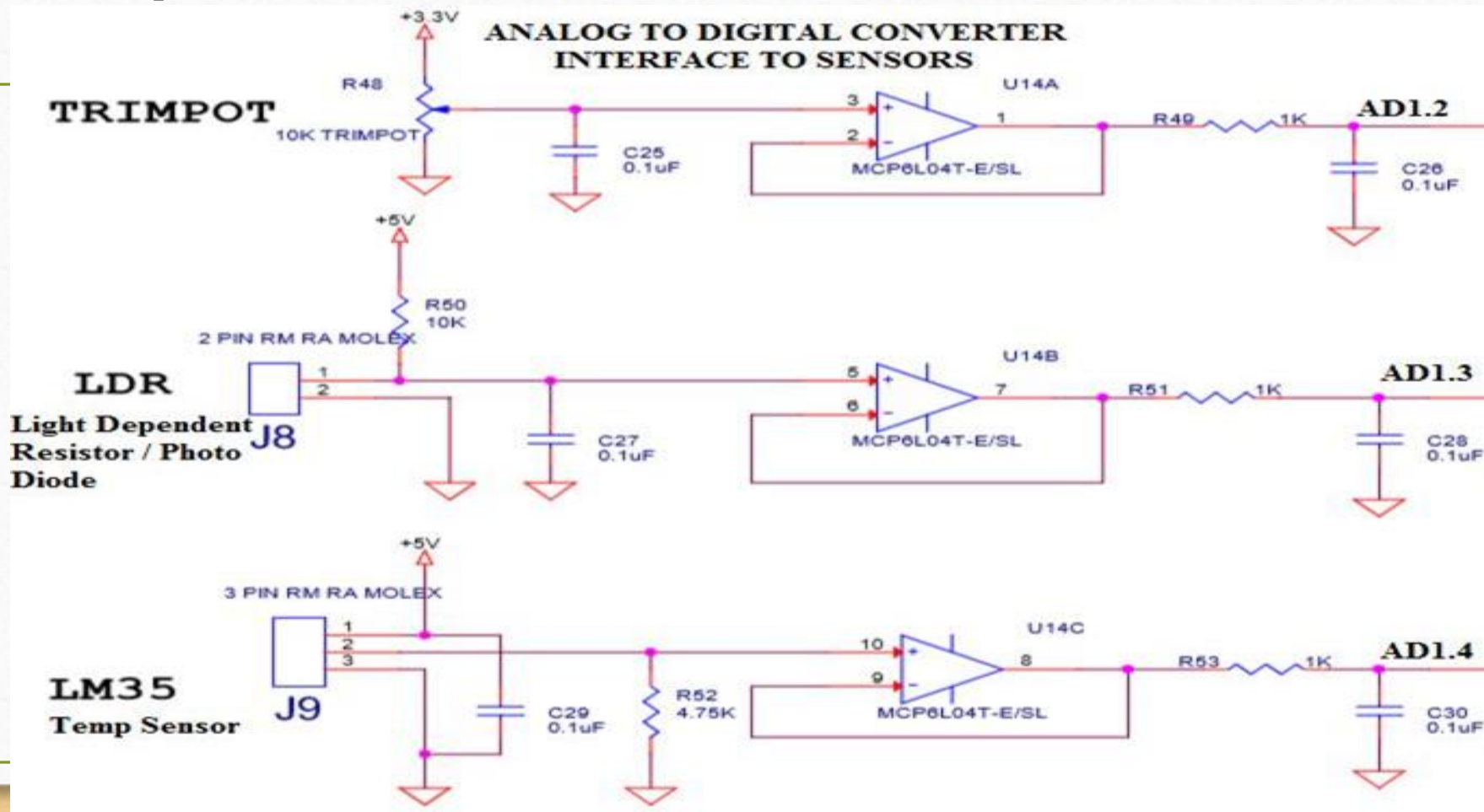
    IO0DIR = (1 << 21) ; // P0.21-o/p
    PINSEL1 = 1 << 24; // P0.28 AS AD0.1(01)
    LED_ON;
    do
    {
        AD0CR=(1<<1|1<<21|1<<24);
        While ( (AD0GDR & (unsigned long) 1 << 31) == 0);
        i = (AD0GDR >> 6 ) & 0x3FF ;
        if (i > 100)

            LED_OFF;
        else
            LED_ON;
    }
    while(1);
}
```

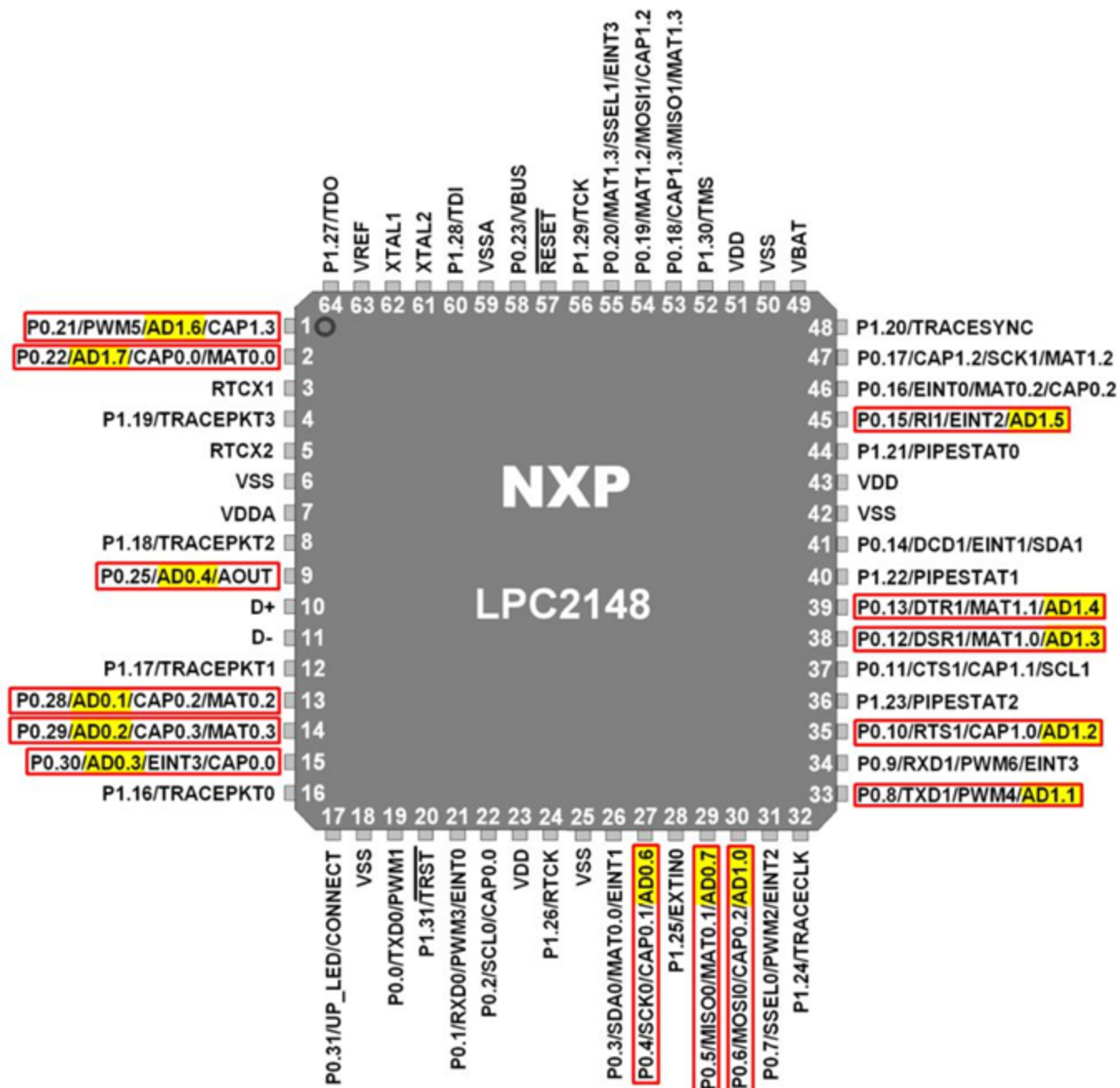


## Interfacing of Potentiometer , LDR and Temperature Sensor (LM35)

Here ADC channel 1 is selected, and three ADC inputs AD1.2, AD1.3 and AD1.4 are used to read the the analog voltage from trimpot, LDR and LM35. All the inputs are connected to LPC 2148 through voltage followers for protection and noise reduction.







```

unsigned int adc(int no,int ch)
{
//  adc(1,4) for temp sensor LM35, digital value will increase as temp increases
//  adc(1,3) for LDR - digital value will reduce as the light increases
//  adc(1,2) for trimpot - digital value changes as the pot rotation
    unsigned int val;


---


    PINSEL0 |= 0x0F300000;      /* Select the P0_13 AD1.4 for ADC function (26 and 27 bit to 1)*/
    /* Select the P0_12 AD1.3 for ADC function (24 and 25 bit to 1) */
    /* Select the P0_10 AD1.2 for ADC function (20 and 21 bit to 1)*/
    switch (no)
    {
        //select adc
        case 0: AD0CR = 0x00200600 | (1<<ch); //select channel
                AD0CR |= (1<<24) ; //start conversion

                while ( ( AD0GDR & ( 1U << 31 ) ) == 0);
    }
}

```

```
    val = AD0GDR;  
    break;
```

```
case 1: AD1CR = 0x00200600 | ( 1 << ch ); //select channel  
       AD1CR|= (1<<24); //start conversion
```

---

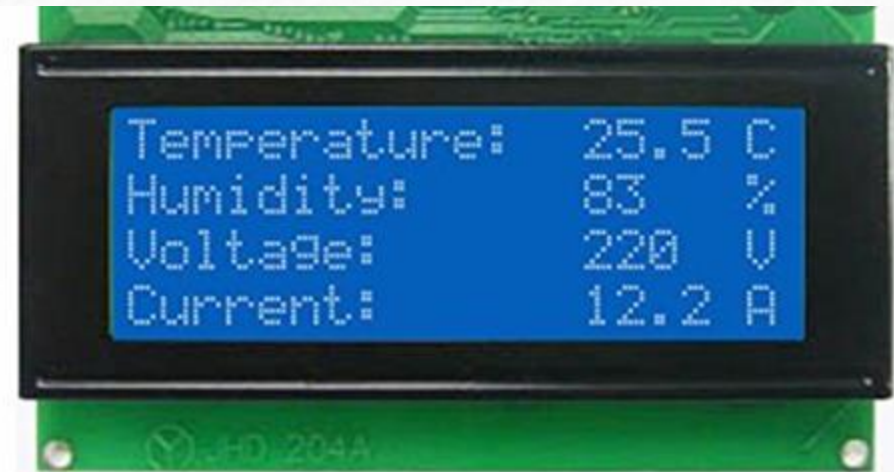
```
    while ( ( AD1GDR & (1U << 31) ) == 0);  
    val = AD1GDR;  
    break;
```

```
    }  
    val = (val >> 6) & 0x03FF;  
    // bit 6:15 is 10 bit AD value  
    return val;  
}
```



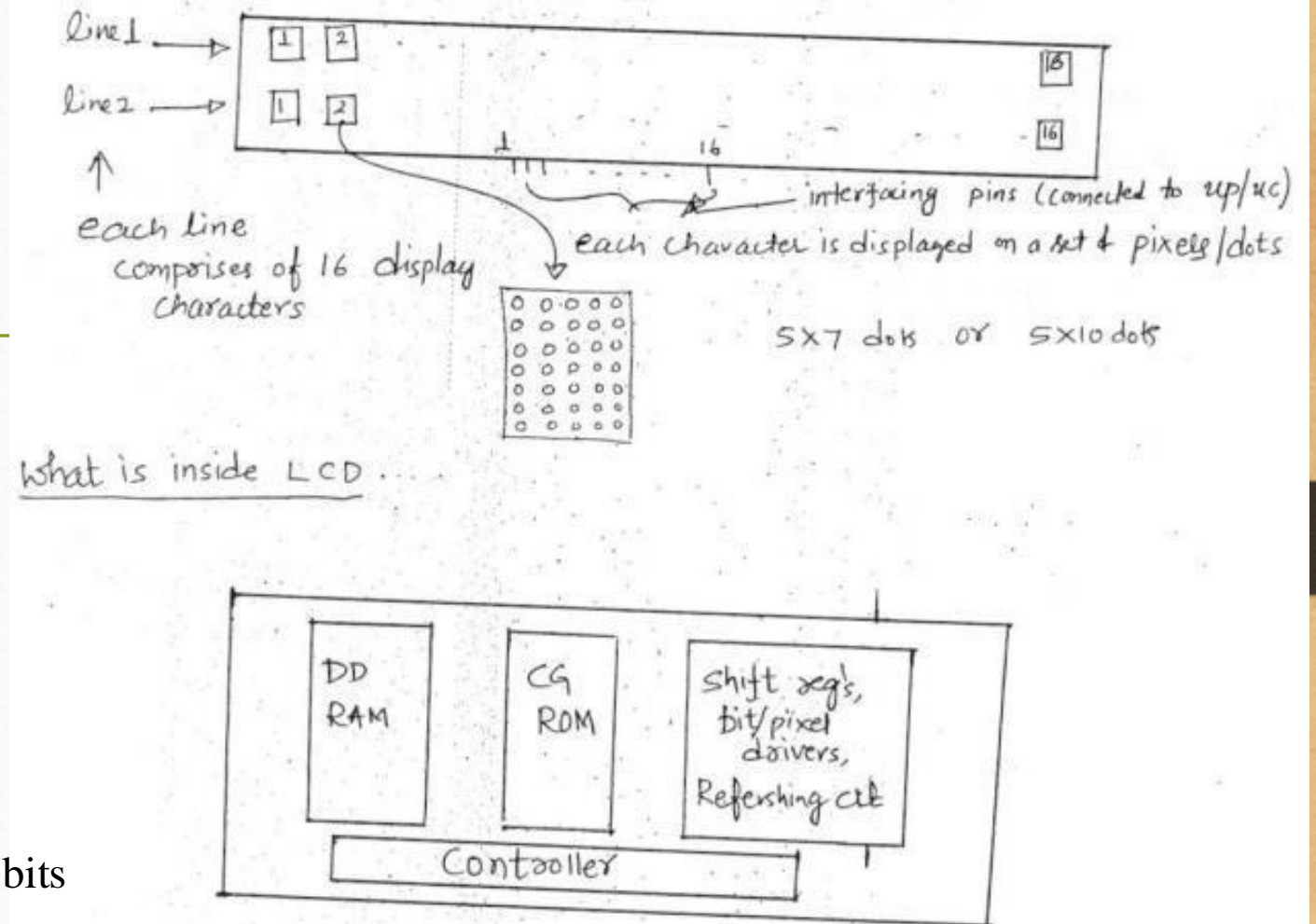
## Alpha Numeric LCD Interface : Interface LCD and Write an Embedded C program to display text messages on the multiple lines of the display.

- LCD's are preferred to seven segment displays because of their versatility and capability to house more information.
- 2 line (2 x 16) and 4 line (4x20) is the most popular, low cost character oriented LCD, suitable for understanding the working and programming of LCD.
- LCD modules are popularly used in many of the electronics devices like coin phone, billing machine and weighing machines.
- It is a powerful display options for stand-alone systems. Because of low power dissipation, high readability, flexibility for programmers.

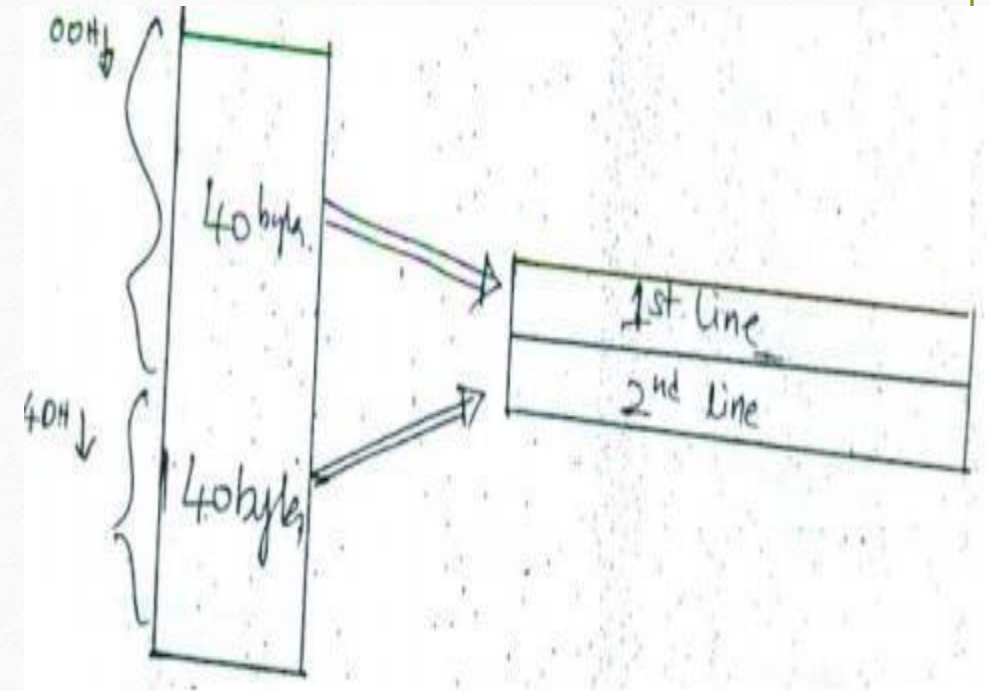




- LCD consists of DDRAM, CGROM, Shift registers, bit/pixel drivers, refreshing logics and lcd controller.
- The data to be displayed on lcd, is to be written on to the DDRAM-display data Ram using the ascii format.
- CGROM-Character generator rom, contains dot/pixel patterns for every character to be displayed (pre programmed).
- Shift registers are used to convert CGROM parallel data to serial data(serializing)
- 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.



- ❑ **DDRAM** : The data to be displayed on LCD, is to be written on to the display data RAM using ASCII format, i.e. if 'A' is to be displayed, ASCII code of A. i.e. 65 to be written.
- ❑ The first 40 bytes (00 – 39 decimal) of RAM allocated for 1<sup>st</sup> Line, another 40 bytes (40 to 79 decimal) of RAM to 2<sup>nd</sup> line
- ❑ First 16 locations are allocated to corresponding 16 display digits of 1st line. If we want to write to the 4th digit, we have to write to the 4th location in DDRAM.
- ❑ Even though display shows 16 digits at a time, memory holds 40 characters for each line, so using shift command we can rotate the display to make all the 40 characters visible

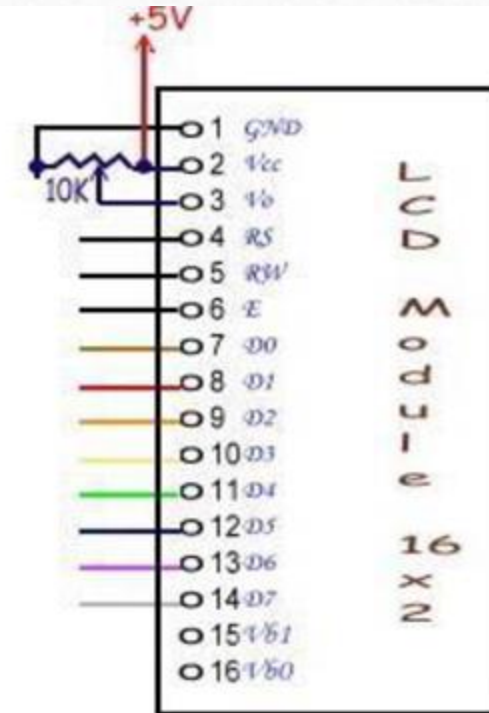
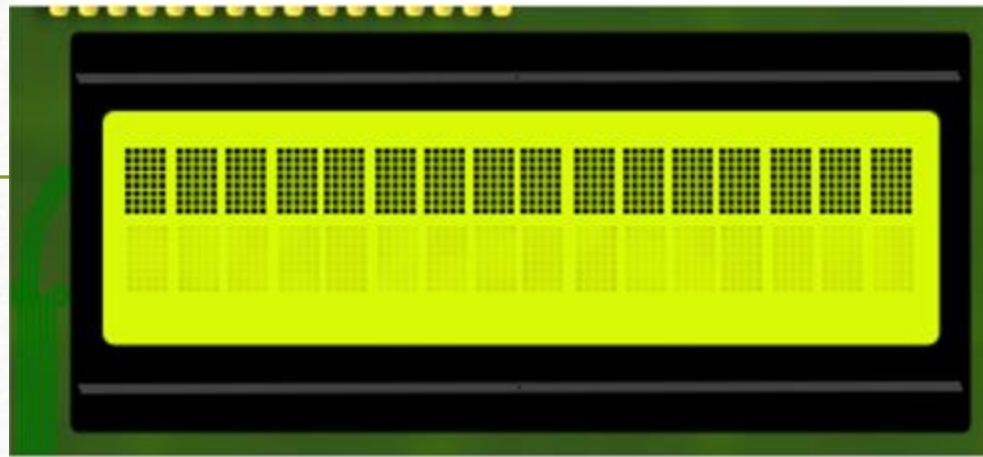


❖ CGROM : character generator ROM : it contains dot/pixel patterns for every character to be displayed (pre programmed), in 5 x 7 mode, 7 bytes of CGROM required to produce the character on the display so CGROM is used to convert ascii code to dot patterns on display.

---

❖ Shift – reg's, bit/pixel drives and refreshing logic: Shift registers used to convert CGROM parallel data to serial data (serialising), drives 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 data we send to LCD is of two types,
  - Command to the LCD(to configure) or
  - ASCII code of character to be displayed on LCD (to DDRAM).



❑ **RS (Register Select): 0 or 1**

- 0 - writing command byte into command register of LCD
- 1 - writing data (ASCII code) into Data register of LCD

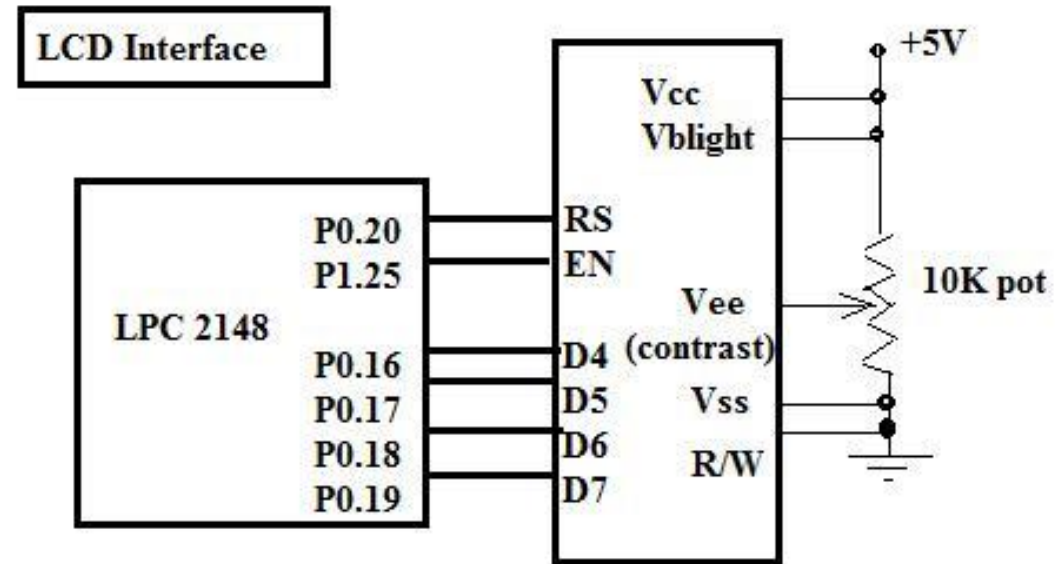
❑ **R/W : (Read/Write) (Note: Many a times, R/W is grounded)**

- 0- Write to LCD (Data/Command)
- 1- Read from the LCD

❑ **E (Enable) : 1 to 0 pulse**

- Enable is required to perform the writing/reading to LCD, E – '1' (for 450nsec) & then '0' (High to Low Pulse)

❑ **D0-D7** - It is a bidirectional data bus, used to write data/command to LCD or reading status. In 4bit nibble mode, only lines D4 – D7 are used for communication.






LCD Command Table

| Instruction             | D7 | D6 | D5 | D4 | D3  | D2  | D1  | D0 | Descripti<br>on                                                                                                                                                         |
|-------------------------|----|----|----|----|-----|-----|-----|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clear display           | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 1  | Clears Display and returns cursor to home position.                                                                                                                     |
| Cursor home             | 0  | 0  | 0  | 0  | 0   | 0   | 1   | X  | Returns cursor to home position. Also returns display being shifted to the original position.                                                                           |
| Entry mode set          | 0  | 0  | 0  | 0  | 0   | 1   | I/D | S  | I/D = 0 - cursor is in decrement position.<br>I/D = 1 - cursor is in increment position.<br>S = 0 - Shift is invisible.<br>S = 1 - Shift is visible                     |
| Display ON- OFF Control | 0  | 0  | 0  | 0  | 1   | D   | C   | B  | D- Display, C- Cursor, B-Blinking cursor<br>0-OFF<br>1-ON                                                                                                               |
| Cursor/ Display Shift   | 0  | 0  | 0  | 1  | S/C | R/L | X   | X  | S/C = 0 - Move cursor.<br>S/C = 1 - Shift display.<br>R/L = 0 - Shift left.<br>R/L = 1- Shift right.                                                                    |
| Function Set            | 0  | 0  | 1  | DL | N   | F   | X   | X  | DL = 0 - 4 bit interface.<br>DL = 1 - 8 bit interface.<br>N = 0 - 1/8 or 1/11 Duty (1 line).<br>N = 1 - 1/16 Duty (2 lines).<br>F = 0 - 5x7 dots.<br>F = 1 - 5x10 dots. |

## Embedded software / Driver Program for 4 x 20 alphanumeric LCD

Two steps are involved,

1. Configure the LCD for different parameters/settings, by writing series of commands (command bytes) like
- 

-  Function set command(0x28)
-  Display On command(0x0E)
-  Clear display (0x01)

1. Writing actual string data to LCD, character by character, (by default characters are displayed from line1 first column position, we can issue DDRAM address command - 0x80 + char pos, for first line, 0xc0 + char pos, for second line, 0x94+char pos, for third line, 0xD4+char pos, for fourth line).

*//Alpha-numeric LCD Interface (4Lines,20characters)*

*//Connected in 4bit nibble mode*

*//LCD handshaking:RS->P0.20,EN-*

*>P0.25 ,R/W -Gnd*

*//LCD data:D4,D5,D6,D7 ->*

*P0.16,P0.17,P0.18,P0.19*

`#include <lpc214x.h>`

`#define PLOCK 0x00000400`

`#define LED_OFF (IO0SET = 1U << 31)`

`#define LED_ON (IO0CLR = 1U << 31)`

`#define RS_ON (IO0SET = 1U << 20)`

`#define RS_OFF (IO0CLR = 1U << 20)`

`#define EN_ON (IO1SET = 1U << 25)`

`#define EN_OFF (IO1CLR = 1U << 25)`

`void SystemInit(void);`

`static void delay_ms(unsigned int j);`

*//millisecond delay* `static void`

`delay_us(unsigned int count);`

*//microsecond delay*

`static void LCD_SendCmdSignals(void);`

`static void LCD_SendDataSignals(void);`

`static void LCD_SendHigherNibble(unsigned char dataByte);`

`static void LCD_CmdWrite( unsigned char cmdByte);`

`static void LCD_DataWrite( unsigned char dataByte);`

`static void LCD_Init(void);`

`void LCD_DisplayString(const char *ptr_stringPointer_u8);`



```

int main()
{
    SystemInit();
    IO0DIR |= 1U << 31 | 0x00FF0000 ;
    // to set P0.16 to P0.23 as o/ps
    IO1DIR |= 1U << 25;
    // to set P1.25 as o/p used for EN
    // make D7 Led on off for testing
    LED_ON; delay_ms(500);
    LED_OFF; delay_ms(500);
    LCD_Init(); delay_ms(100);
    LCD_CmdWrite(0x80); LCD_DisplayString("RV College Of Engrng");
    LCD_CmdWrite(0xc0); LCD_DisplayString(" Computer Science");
    LCD_CmdWrite(0x94); LCD_DisplayString(" 4th Semester");
    LCD_CmdWrite(0xD4); LCD_DisplayString(" C Section");
    while(1);
}

```

```

static void LCD_Init(void)
{
    delay_ms(100);
    LCD_Reset();
    LCD_CmdWrite(0x28u);
    //Initialize the LCD for 4-bit 5x7
    matrix type
    LCD_CmdWrite(0x0Eu); // Display ON
    cursor ON
    LCD_CmdWrite(0x01u); //Clear the LCD
    LCD_CmdWrite(0x80u); //go to First line
    First Position
}

```

```
static void LCD_CmdWrite( unsigned char cmdByte)
{
    LCD_SendHigherNibble(cmdByte);
    LCD_SendCmdSignals();
    cmdByte = cmdByte << 4;
    LCD_SendHigherNibble(cmdByte);
    LCD_SendCmdSignals();
}
```

```
static void LCD_DataWrite( unsigned char dataByte)
{
    LCD_SendHigherNibble(dataByte);
    LCD_SendDataSignals();
    dataByte = dataByte << 4;
    LCD_SendHigherNibble(dataByte);
    LCD_SendDataSignals();
}
```

```
static void LCD_SendHigherNibble(unsigned char dataByte)
{
    //send the D7,6,5,D4(upper nibble) to P0.16 to P0.19
    IO0CLR = 0X000F0000;
    IO0SET = ((dataByte >>4) & 0x0f) << 16;
}
```

```
static void LCD_SendCmdSignals(void)
{
    RS_OFF; // RS - 1
    EN_ON;delay_us(100);EN_OFF; // EN - 1 then 0
}
static void LCD_SendDataSignals(void)
{
    RS_ON;// RS - 1
    EN_ON;delay_us(100);EN_OFF; // EN - 1 then 0
}
```

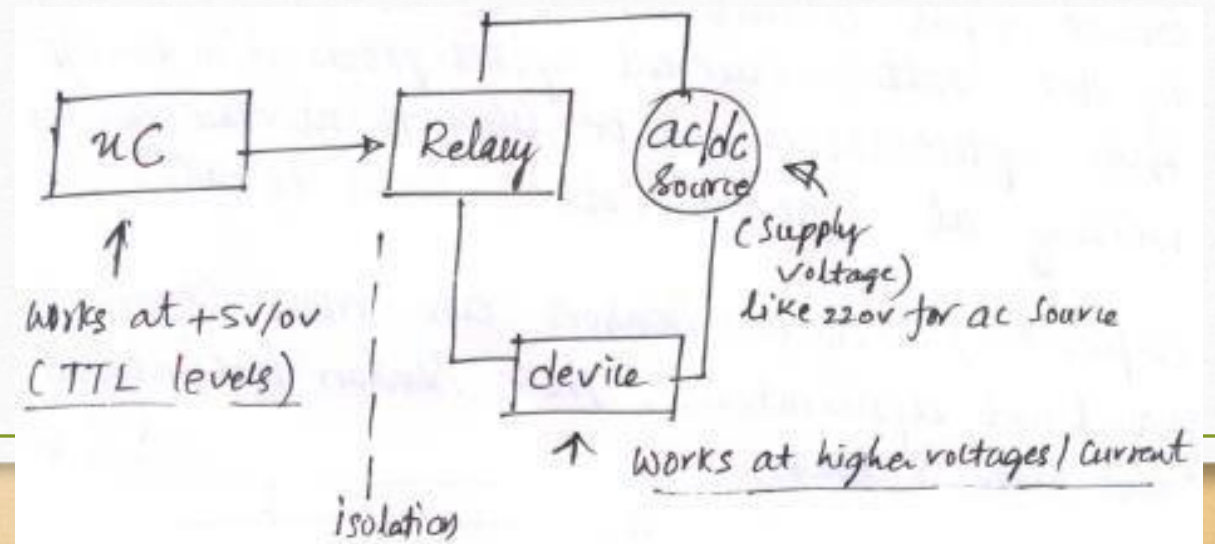
---

```
void LCD_DisplayString(const char *ptr_string)
{
    // Loop through the string and display char by char
    while((*ptr_string)!=0)
        LCD_DataWrite(*ptr_string++);
}
```



## Interfacing High power devices using relays

- Microcontroller port pin cannot be directly used to switch ON/OFF (control) high power ac/dc devices like bulb, fan, dc motor, solenoids etc (unlike LED's which can be directly controlled / connected by port pins) as they don't drive required high current or voltages by high power devices.
- Relays are used to interface high power devices to microcontrollers.
- Relays are devices that can turn on or off the power supplied to other device like bulb, like a switch.
- Relays are used to control high-power circuits by low-power devices like microcontrollers.
- Relays provide isolation between microcontroller and high power source



Relays are of two types

- Electromechanical Relay – cheap, less reliable, because of movable parts used in the construction of these relays. Relay driver is required to drive these relays from the microcontroller ports.
- Solid State Relay – costly, reliable (durable), no moving parts, no relay driver required.

**Interfacing:** Interface AC device to LPC 2148 using mechanical relay:

Where:

C is the Common terminal

NO is the Normally Open contact

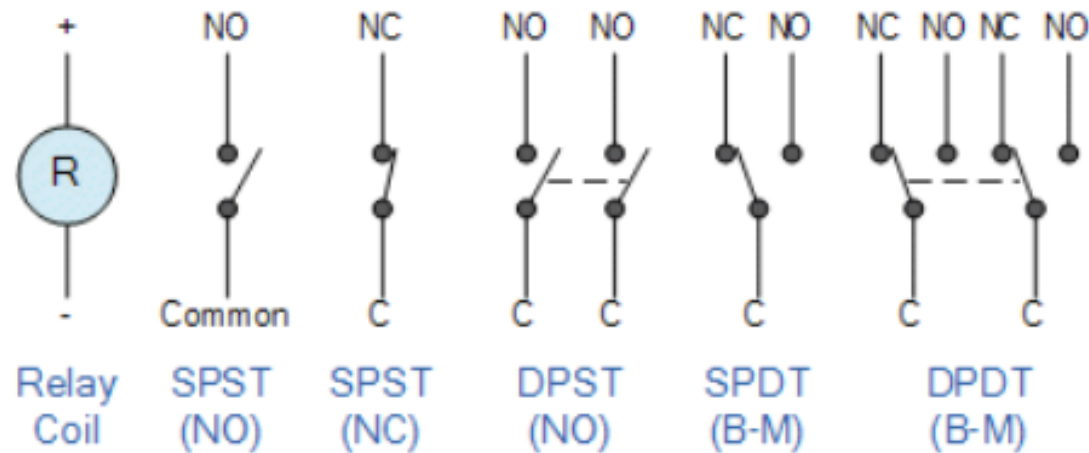
NC is the Normally Closed contact

SPST – Single Pole Single Throw

SPDT – Single Pole Double Throw

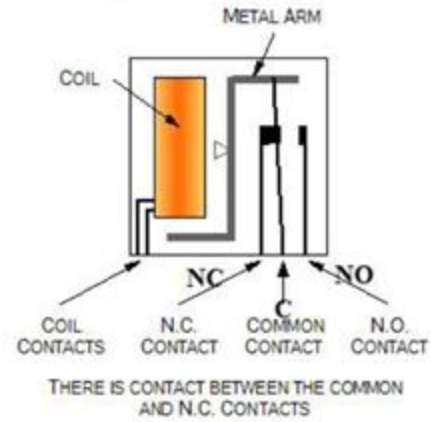
DPST – Double Pole Single Throw

DPDT – Double Pole Double Throw

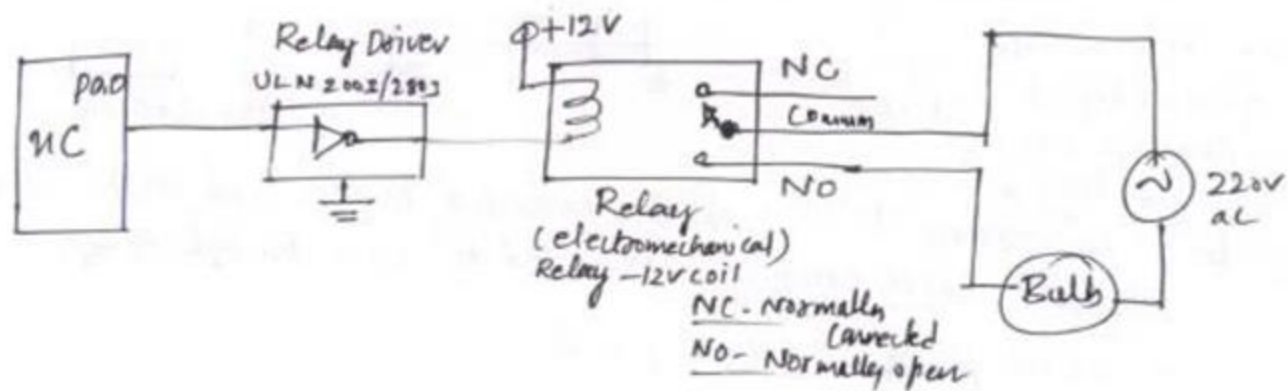
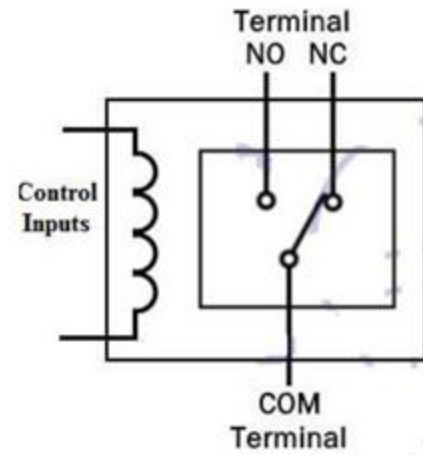
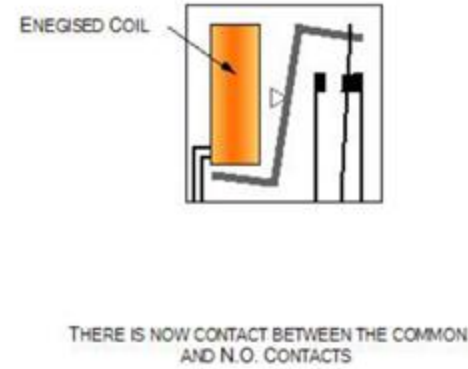


“Single Pole Double Throw – (Break before Make)”, or SPDT – (B-M)

Relay off



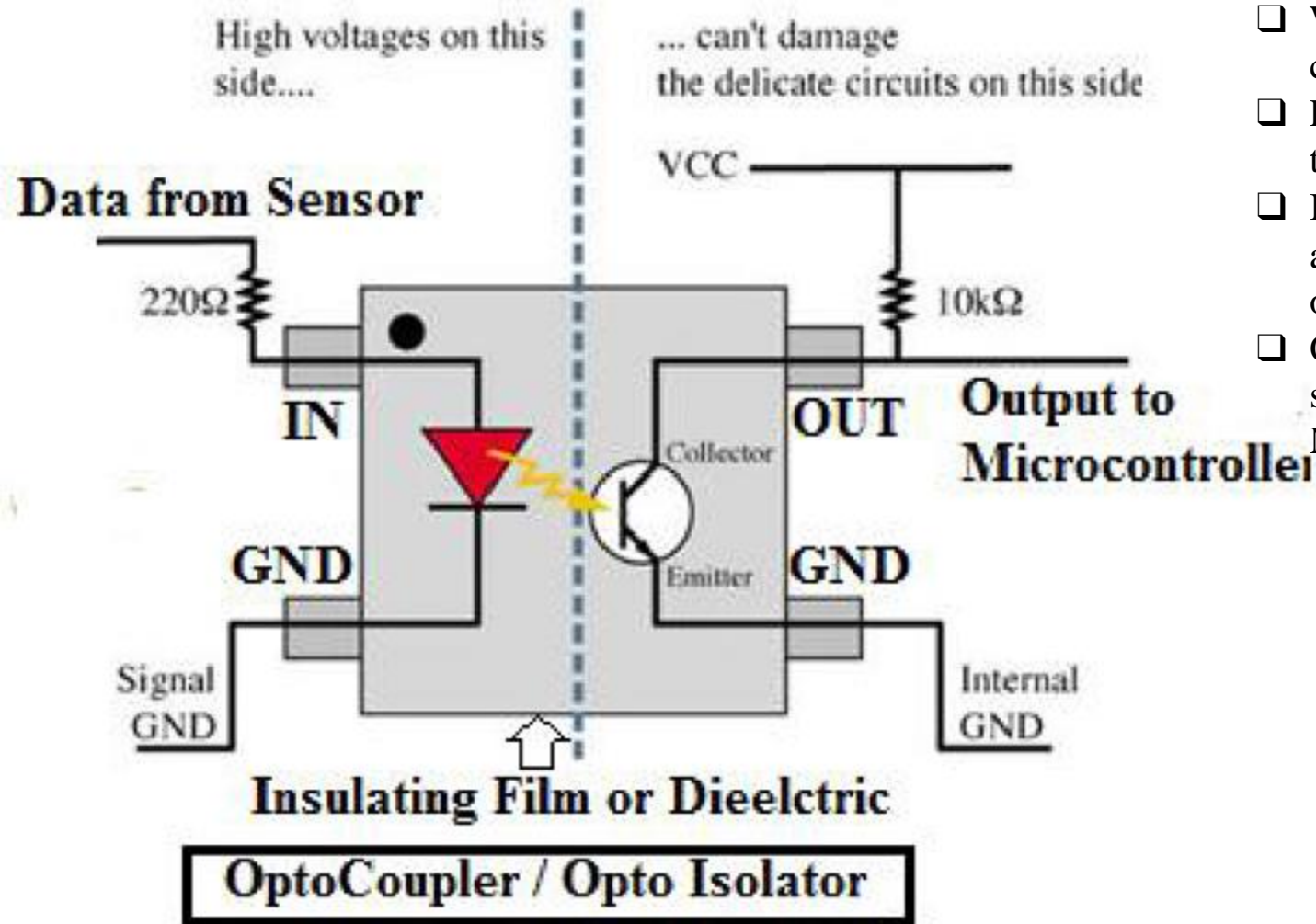
Relay on





## Interfacing Industrial Proximity sensor using Opto-Isolator

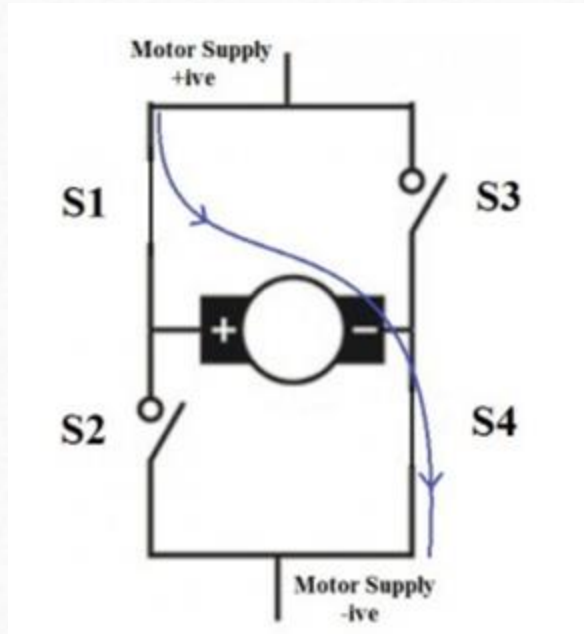
Opto-isolator has photodiode and photo transistor:



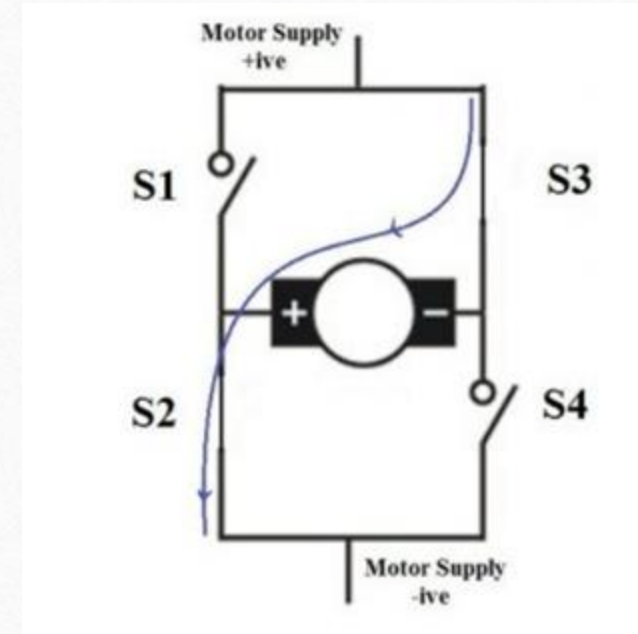
- ☐ When diode conducts, it emits light and the transistor conducts.
- ☐ Provides complete isolation between diode and transistor
- ☐ Input circuit which drives the diode do not get affected by the voltage/ current spikes produced in the o/p circuit.
- ☐ Opto-isolators are used to interface devices (like sensors) working at different voltage levels to Microcontrollers.

# DC Motor: H Bridge

- DC motors were the first form of motor widely used, as they could be powered from existing direct-current lighting power distribution systems.
- A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings.



When switches S1 and S4 are switched on, motor runs in clockwise direction.



When S2 and S3 are switched on, motor runs in anticlockwise direction.