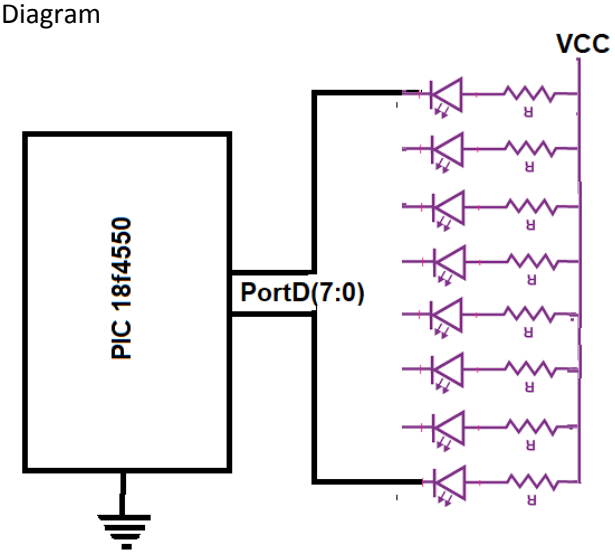
Embedded C coding Examples

PIC18F4550

```
//Example LED Glowing
#include <pic18f4550.h>

void delay(unsigned int t)
{unsigned int i,j;
for(i=0;i<t;i++);
   for(j=0;j<255;j++);
}

void main (void)
{

TRISD = 0x00; // Configure PORTC as an input port
//TRISD = 0; // Configure PORTD as an output port
while(1)
{

   for(unsigned int i=255;i>=0;i-- )
   {PORTD=i;
   delay(5);
   }

}
}
```

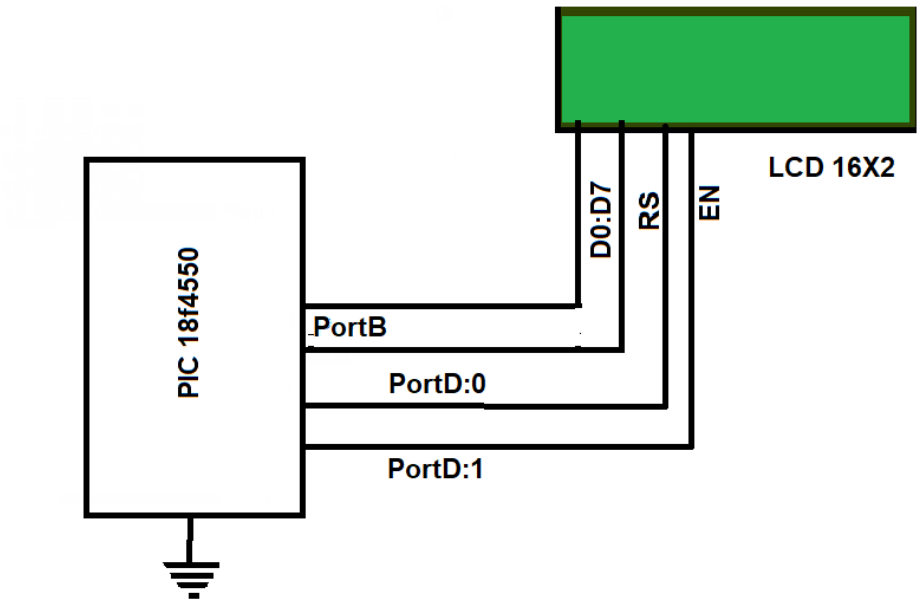
Diagram



```
//Example LCD Display


#include <pic18f4550.h>
#include "Configuration_Header_File.h"

#define RS LATD0                /*PORTD 0 pin is used for Register Select*/
#define EN LATD1                /*PORTD 1 pin is used for Enable*/
#define ldata LATB              /*PORTB is used for transmitting data to LCD*/
#define LCD_Port TRISB          /*define macros for PORTB Direction Register*/
#define LCD_Control TRISD       /*define macros for PORTD Direction Register*/


void LCD_Init();
void LCD_Command(char );
void LCD_Char(char x);
void LCD_String(const char *);
void LCD_String_xy(char ,char ,const char*);
void MSdelay(unsigned int);
/*****************************Main Program*****************************/

void main(void)
{
  OSCCON=0x72;                  /*Use Internal Oscillator with Frequency 8MHZ*/
  LCD_Init();                   /*Initialize 16x2 LCD*/
  LCD_String_xy(1,5,"Hello");           /*Display string at location(row,location).
                          * This function passes string to display*/
  LCD_String_xy(2,0,"ElectronicEngg");   /*Display string at location(row,location).
                          * This function passes string to display*/

  while(1);
}
/*****************************Functions*********************************/
void LCD_Init()
{
  MSdelay(15);          /*15ms,16x2 LCD Power on delay*/
        LCD_Port = 0x00;      /*Set PORTB as output PORT for LCD data(D0-D7) pins*/
  LCD_Control = 0x00;    /*Set PORTD as output PORT LCD Control(RS,EN) Pins*/
        LCD_Command(0x38);    /*uses 2 line and initialize 5*7 matrix of LCD*/
  LCD_Command(0x01);    /*clear display screen*/
  LCD_Command(0x0c);    /*display on cursor off*/
  LCD_Command(0x06);    /*increment cursor (shift cursor to right)*/
}
void LCD_Clear()
{
  LCD_Command(0x01);    /*clear display screen*/
}
void LCD_Command(char cmd )
{
        ldata= cmd;           /*Send data to PORT as a command for LCD*/
        RS = 0;             /*Command Register is selected*/
```

```
        EN = 1;             /*High-to-Low pulse on Enable pin to latch data*/
        NOP();
        EN = 0;
        MSdelay(3);
}

void LCD_Char(char dat)
{
        ldata= dat;          /*Send data to LCD*/
        RS = 1;             /*Data Register is selected*/
        EN=1;               /*High-to-Low pulse on Enable pin to latch data*/
        NOP();
        EN=0;
   MSdelay(1);
}


void LCD_String(const char *msg)
{
        while((*msg)!=0)
        {
          LCD_Char(*msg);
          msg++;
   }
}

void LCD_String_xy(char row,char pos,const char *msg)
{
   char location=0;
   if(row<=1)
   {
      location=(0x80) | ((pos) & 0x0f); /*Print message on 1st row and desired location*/
      LCD_Command(location);
   }
   else
   {
      location=(0xC0) | ((pos) & 0x0f); /*Print message on 2nd row and desired location*/
      LCD_Command(location);
   }
   LCD_String(msg);
}

/*********************************Delay Function*********************************/
void MSdelay(unsigned int val)
{
    unsigned int i,j;
      for(i=0;i<=val;i++)
         for(j=0;j<81;j++);      /*This count Provide delay of 1 ms for 8MHz Frequency */
 }
```

//interface diagram

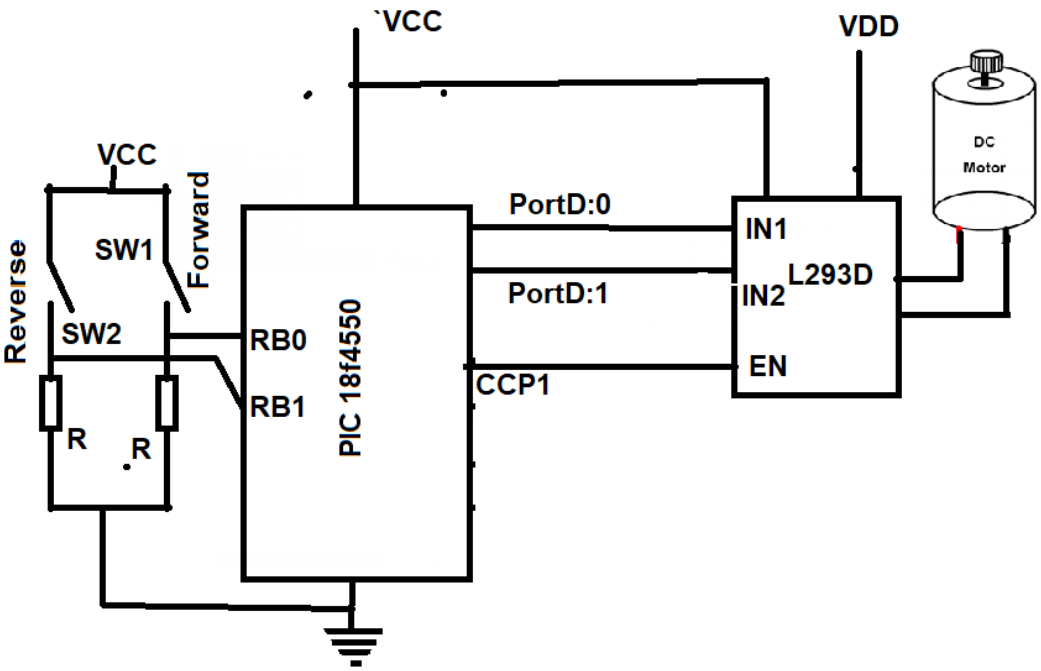


//Example of Motor control using PWM

```
#define _XTAL_FREQ 20000000
#define TMR2PRESCALE 4
#include <xc.h>
#include "pic18f4550.h"
#include "config.h"
#include<stdint.h>
// BEGIN CONFIG
/*#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit
(RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code
protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all
program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

 */
 //END CONFIG
//Switch Debounce time in us
#define DEBOUNCE_TIME  240
//Switch Status
#define SWITCH_PRESSED    1
#define SWITCH_BOUNCE     0
//Define pins for motor
#define M_a    RD0
#define M_b    RD1
//Define pins for switch
#define S_1  RB0
#define S_2  RB1

// CCP1 module is used here to generate the required PWM
// Timer2 module is used to generate the PWM
// This PWM has 10bit resolution
//max Duty
pwmMaxDuty(const unsigned int freq)
{
  return(_XTAL_FREQ/(freq*TMR2PRESCALE));
}
//Calculate the PR2 value
void initPwm(const unsigned int freq)
{
   //calculate period register value
   PR2 = (unsigned int)((_XTAL_FREQ/(freq*4*TMR2PRESCALE)) - 1);
}
```

```
//Give a value in between 0 and 1024 for duty-cycle
void applyPWMDutyCycle(unsigned int dutyCycle, const unsigned int freq)
{
  if(dutyCycle<1024)
  {
    //1023 because 10 bit resolution
    dutyCycle = (unsigned int)(((float)dutyCycle/1023)*pwmMaxDuty(freq));
    CCP1CON &= 0xCF;            // Make bit4 and 5 zero (Store fraction part of duty cycle)
    CCP1CON |= (0x30&(dutyCycle<<4)); // Assign Last 2 LSBs to CCP1CON
    CCPR1L = (uint8_t)(dutyCycle>>2); // Put MSB 8 bits in CCPR1L
  }
}
//Init the Port pin
void initPort()
{
  TRISB0 = 1;  // Make S_1 pin an input
  TRISB1 = 1;  // Make S_2 pin an input
  TRISD0 = 0;  // Make M_a pin an output
  TRISD1 = 0;  // Make M_b pin an output
  TRISC2 = 0;   //Make pin output for PWM
}
//Run motor clockwise
void motorRunClockWise()
{
  M_a=1;
  M_b=0;
  M_a=1;
  M_b=0;
}
//configure and start PWM1
void startPwm()
{
  CCP1CON = 0x0C; // Configure CCP1 module in PWM mode
  T2CON = 0x01;  // Set Prescaler to be 4
  T2CON |= 0x04; // Enable the Timer2, hence enable the PWM.
}
//Function to check the status of Switch S1
int isS1Pressed()
{
  int switchStatus =  SWITCH_BOUNCE;
  if(S_1 == SWITCH_PRESSED)
  {
    //Wait time more then bouncing period
    __delay_us(DEBOUNCE_TIME);
    switchStatus =  S_1? SWITCH_PRESSED : SWITCH_BOUNCE;
  }
  return switchStatus ;
}
//Function to check the status of Switch S2
int isS2Pressed()
{
```

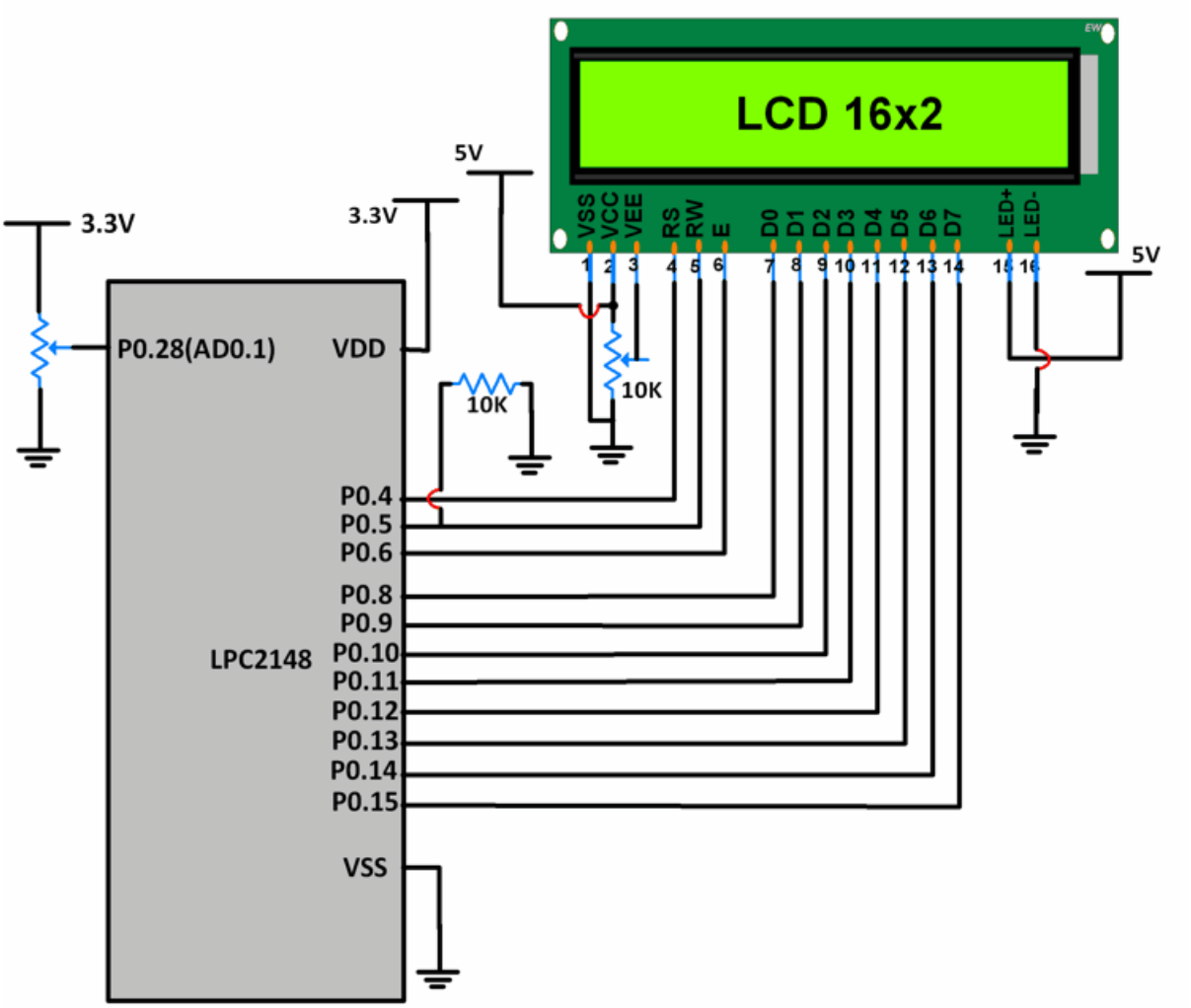
```
    int switchStatus =  SWITCH_BOUNCE;
    if(S_2 == SWITCH_PRESSED)
    {
       //Wait time more then bouncing period
       __delay_us(DEBOUNCE_TIME);
       switchStatus =  S_2? SWITCH_PRESSED : SWITCH_BOUNCE;
    }
    return switchStatus ;
}
//main function
void main()
{
    uint16_t dutycycle = 0;
    uint16_t dutyCycleApply = 0;
    const uint32_t pwmFreq = 5000;


    initPort(); //Init Gpio port
    motorRunClockWise(); //Run motor clockwise
    initPwm(pwmFreq); // Initialize PWM
    applyPWMDutyCycle(dutycycle,pwmFreq);
    startPwm();
    do
    {
       //Check the switch status for duty cycle
       dutycycle = (isS1Pressed() && isS2Pressed())? 1023: dutycycle; //100% duty cycle
       dutycycle = (isS1Pressed() && !isS2Pressed())? 768: dutycycle; //75% duty cycle
       dutycycle = (!isS1Pressed() && isS2Pressed())? 512: dutycycle; //50% duty cycle
       dutycycle = (!isS1Pressed() && !isS2Pressed())? 256: dutycycle;//25% duty cycle
       if (dutycycle != dutyCycleApply)
       {
          applyPWMDutyCycle(dutycycle,pwmFreq);
          dutyCycleApply = dutycycle;
       }
    }
    while(1);  //super loop
}
//Interface diagram
```


`VCC
VDD
DC
Motor
VCC
Forward
SW1
Reverse
SW2
R
R
RB0
RB1
PIC 18f4550
PortD:0
PortD:1
CCP1
IN1
IN2
L293D
EN

LED Glowing with ARM LPC 2148

```
#include <LPC214X.H>                    // Header file for LPC Devices
void delay(unsigned int t)
{
unsigned int i,j;
for(i=0;i<t;i++);
for(j=0;j<10000;j++);
}
int main()
{
   unsigned char k;
   IODIR0 = 0x000000FF;         //Port-0.0 to Port-0.7 pins as Output Pi

while(1)
  {
      IOSET0 = 0x000000FF;             //P0.0 to P0.7 are high
delay(100);
      IOCLR0 = 0x000000FF;            //P0.0 to P0.7 are low
delay(100);
   }
```

ADC Interface with LPC2148




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

void delay_ms(uint16_t j) /* Function for delay in milliseconds  */
{
    uint16_t x,i;
      for(i=0;i<j;i++)
      {
    for(x=0; x<6000; x++);     /* loop to generate 1 millisecond
delay with Cclk = 60MHz */
      }
}

void LCD_Command(char command)
{
      IO0PIN = ( (IO0PIN & 0xFFFF00FF) | (command<<8) );
      IO0SET = 0x00000040; /* EN = 1 */
      IO0CLR = 0x00000030; /* RS = 0, RW = 0 */
      delay_ms(2);
      IO0CLR = 0x00000040; /* EN = 0, RS and RW unchaned(i.e. RS =
RW = 0) */
      delay_ms(5);
```

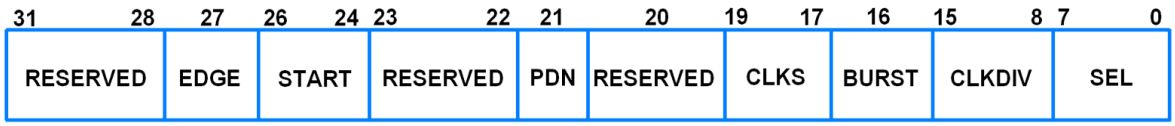
```
}

void LCD_Init(void)
{
      IO0DIR = 0x0000FFF0; /* P0.8 to P0.15 LCD Data. P0.4,5,6 as RS
RW and EN */
      delay_ms(20);
      LCD_Command(0x38);  /* Initialize lcd */
      LCD_Command(0x0C);   /* Display on cursor off */
      LCD_Command(0x06);  /* Auto increment cursor */
      LCD_Command(0x01);   /* Display clear */
      LCD_Command(0x80);  /* First line first position */
}

void LCD_Char (char msg)
{
            IO0PIN = ( (IO0PIN & 0xFFFF00FF) | (msg<<8) );
            IO0SET = 0x00000050; /* RS = 1, , EN = 1 */
            IO0CLR = 0x00000020; /* RW = 0 */
            delay_ms(2);
            IO0CLR = 0x00000040; /* EN = 0, RS and RW unchanged(i.e.
RS = 1, RW = 0) */
            delay_ms(5);
}

int main(void)
{
      uint32_t result;
      float voltage;
      char volt[18];
      LCD_Init();
      PINSEL1 = 0x01000000; /* P0.28 as AD0.1 */
     AD0CR = 0x00200402; /* ADC operational, 10-bits, 11 clocks for
conversion */
      while(1)
      {
            AD0CR = AD0CR | (1<<24); /* Start Conversion */
            while ( !(AD0DR1 & 0x80000000) ); /* Wait till DONE */
            result = AD0DR1;
            result = (result>>6);
            result = (result & 0x000003FF);
            voltage = ( (result/1023.0) * 3.3 ); /* Convert ADC value
to equivalent voltage */
            LCD_Command(0x80);
            sprintf(volt, "Voltage=%.2f V  ", voltage);
            LCD_String(volt);
            memset(volt, 0, 18);
      }
}




void LCD_String (char* msg)
```

```
{
	uint8_t i=0;
	while(msg[i]!=0)
	{
		LCD_Char(msg[i]);
		i++;
	}
}

void LCD_Data(int8_t data)
{
	IO0PIN = ( (IO0PIN & 0xFFFF00FF) | (data<<8) );
	IO0SET = 0x00000050; /* RS = 1, EN = 1 */
	IO0CLR = 0x00000020; /* RW = 0 */
	delay_ms(2);
	IO0CLR = 0x00000040; /* EN = 0, RS and RW unchanged(i.e. RS =
1, RW = 0) */
	delay_ms(5);
}
```

# ADC0 Registers

1. **AD0CR (ADC0 Control Register)**

- AD0CR is a 32-bit register.
- This register must be written to select the operating mode before A/D conversion can occur.
- It is used for selecting channel of ADC, clock frequency for ADC, number of clocks or number of bits in result, start of conversion and few other parameters.




**AD0CR (ADC0 Control Register)**

- **Bits 7:0 – SEL**
  These bits select ADC0 channel as analog input. In software-controlled mode, only one of these bits should be 1.e.g. bit 7 (10000000) selects AD0.7 channel as analog input.
- **Bits 15:8 – CLKDIV**
  The APB(ARM Peripheral Bus)clock is divided by this value plus one, to produce the clock for ADC.
  This clock should be less than or equal to 4.5MHz.
- **Bit 16 – BURST**
  0 = Conversions are software controlled and require 11 clocks
  1 = In Burst mode ADC does repeated conversions at the rate selected by the **CLKS** field for the analog inputs selected by **SEL** field. It can be terminated by clearing this bit, but the conversion that is in progress will be completed.
  When Burst = 1, the **START** bits must be 000, otherwise the conversions will not start.
- **Bits 19:17 – CLKS**
  Selects the number of clocks used for each conversion in burst mode and the number of bits of accuracy of Result bits of AD0DR.

e.g. 000 uses 11 clocks for each conversion and provide 10 bits of result in corresponding **ADDR** register.

**000** = 11 clocks / 10 bits

**001** = 10 clocks / 9 bits

**010** = 9 clocks / 8 bits

**011** = 8 clocks / 7 bits

**100** = 7 clocks / 6 bits

**101** = 6 clocks / 5 bits

**110** = 5 clocks / 4 bits

**111** = 4 clocks / 3 bits

- **Bit 20 – RESERVED**
- **Bit 21 – PDN**

  0 = ADC is in Power Down mode

  1 = ADC is operational
- **Bit 23:22 – RESERVED**
- **Bit 26:24 – START**

  When **BURST** bit is 0, these bits control whether and when A/D conversion is started

  **000** = No start (Should be used when clearing PDN to 0)

  **001** = Start conversion now

  **010** = Start conversion when edge selected by bit 27 of this register occurs on CAP0.2/MAT0.2 pin

  **011**= Start conversion when edge selected by bit 27 of this register occurs on CAP0.0/MAT0.0 pin

  **100** = Start conversion when edge selected by bit 27 of this register occurs on MAT0.1 pin

  **101** = Start conversion when edge selected by bit 27 of this register occurs on MAT0.3 pin

  **110** = Start conversion when edge selected by bit 27 of this register occurs on MAT1.0 pin

  **111** = Start conversion when edge selected by bit 27 of this register occurs on MAT1.1 pin
- **Bit 27 – EDGE**

  This bit is significant only when the Start field contains 010-111. In these cases,

  0 = Start conversion on a rising edge on the selected CAP/MAT signal

  1 = Start conversion on a falling edge on the selected CAP/MAT signal
- **Bit 31:28 – RESERVED**

2. **AD0GDR (ADC0 Global Data Register)**

- AD0GDR is a 32-bit register.
- This register contains the ADC's DONE bit and the result of the most recent A/D conversion.


**AD0GDR (ADC0 Global Data Register)**

- **Bit 5:0 – RESERVED**
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bit 23:16 – RESERVED**
- **Bits 26:24 – CHN**
  These bits contain the channel from which ADC value is read.
  e.g. 000 identifies that the **RESULT** field contains ADC value of channel 0.
- **Bit 29:27 – RESERVED**
- **Bit 30 – Overrun**
  This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the **RESULT** bits.
  This bit is cleared by reading this register.
- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the AD0CR is written.
  If AD0CR is written while a conversion is still in progress, this bit is set and new conversion is started.

3. **ADGSR (A/D Global Start Register)**

- ADGSR is a 32-bit register.
- Software can write to this register to simultaneously start conversions on both ADC.


**ADGSR (A/D Global Start Register)**

- **BURST (Bit 16), START (Bit <26:24>) & EDGE (Bit 27)**
  These bits have same function as in the individual ADC control registers i.e. AD0CR & AD1CR. Only difference is that we can use these function for both ADC commonly from this register.

4. **AD0STAT (ADC0 Status Register)**

- AD0STAT is a 32-bit register.
- It allows checking of status of all the A/D channels simultaneously.

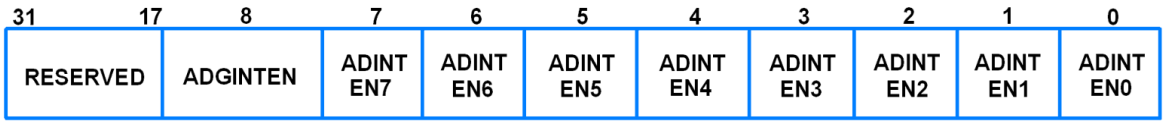

**AD0STAT (ADC0 Status Register)**

- **Bit 7:0 – DONE7:DONE0**
  These bits reflect the **DONE** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 15:8 – OVERRUN7:OVERRUN0**
  These bits reflect the **OVERRUN** status flag from the result registers for A/D channel 7 - channel 0.
- **Bit 16 – ADINT**
  This bit is 1 when any of the individual A/D channel DONE flags is asserted and enables ADC interrupt if any of interrupt is enabled in AD0INTEN register.
- **Bit 31:17 – RESERVED**

5. **AD0INTEN (ADC0 Interrupt Enable)**

- AD0INTEN is a 32-bit register.

- It allows control over which channels generate an interrupt when conversion is completed.

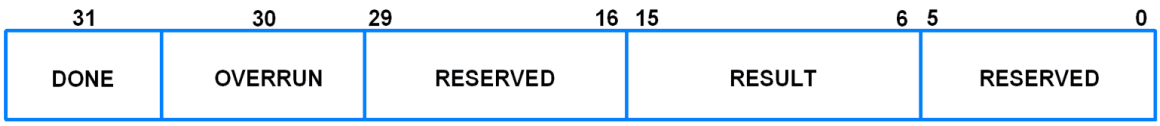



**AD0INTEN (ADC0 Interrupt Enable)**

- **Bit 0 – ADINTEN0**
  0 = Completion of a A/D conversion on ADC channel 0 will not generate an interrupt
  1 = Completion of a conversion on ADC channel 0 will generate an interrupt
- Remaining **ADINTEN** bits have similar description as given for **ADINTEN0**.
- **Bit 8 – ADGINTEN**
  0 = Only the individual ADC channels enabled by **ADINTEN7:0** will generate interrupts
  1 = Only the global **DONE** flag in A/D Data Register is enabled to generate an interrupt

6. **AD0DR0-AD0DR7 (ADC0 Data Registers)**

- These are 32-bit registers.
- They hold the result when A/D conversion is completed.
- They also include flags that indicate when a conversion has been completed and when a conversion overrun has occurred.




**AD0 Data Registers Structure**

- **Bit 5:0 – RESERVED**
- **Bits 15:6 – RESULT**
  When **DONE** bit is set to 1, this field contains 10-bit ADC result that has a value in the range of 0 (less than or equal to VSSA) to 1023 (greater than or equal to VREF).
- **Bit 29:16 – RESERVED**
- **Bit 30 – Overrun**

This bit is set to 1 in burst mode if the result of one or more conversions is lost and overwritten before the conversion that produced the result in the **RESULT** bits.
This bit is cleared by reading this register.

- **Bit 31 – DONE**
  This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.

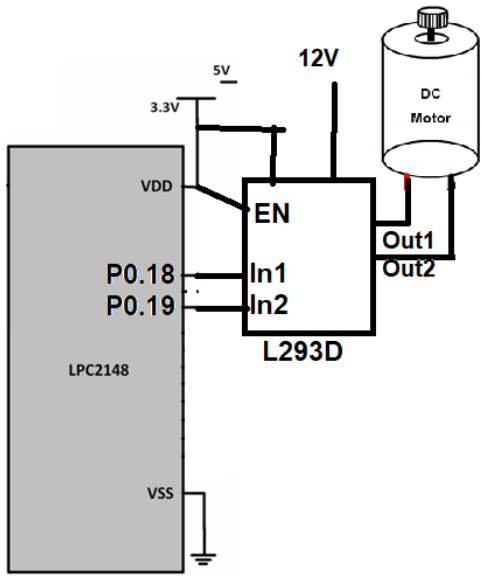
## Steps for Analog to Digital Conversion

1. Configure the ADxCR (ADC Control Register) according to the need of application.
2. Start ADC conversion by writing appropriate value to START bits in ADxCR. (Example, writing 001 to START bits of the register 26:24, conversion is started immediately).
3. Monitor the DONE bit (bit number 31) of the corresponding ADxDRy (ADC Data Register) till it changes from 0 to 1. This signals completion of conversion. We can also monitor DONE bit of ADGSR or the DONE bit corresponding to the ADC channel in the ADCxSTAT register.
4. Read the ADC result from the corresponding ADC Data Register.
   ADxDRy. E.g. AD0DR1 contains ADC result of channel 1 of ADC0.

Dc motor with pwm using LPC2148

```
#include<lpc214x.h>

                    #define MOTOR (1<<18) | (1<<19)
                    #define BUTTON (1<<16)


                    int main()
                    {
                      unsigned char i=0;
```

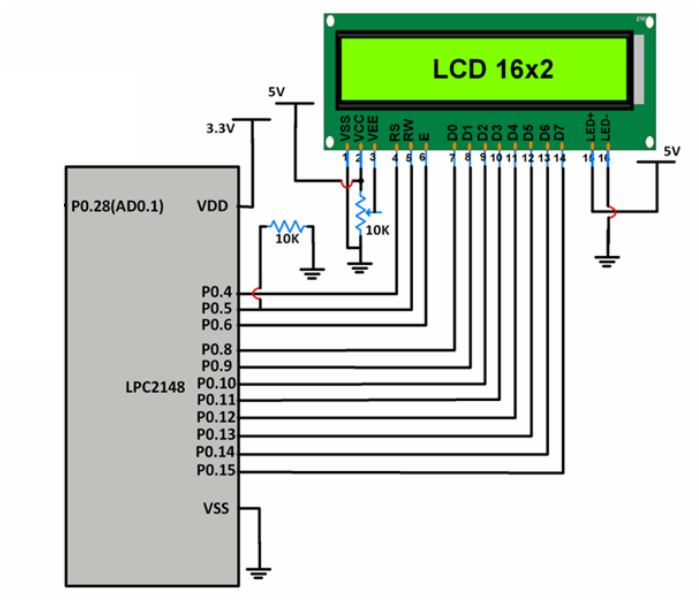
```
  PINSEL0 = 0x00000000;
  PINSEL1 = 0x00000000;
  PINSEL2 = 0x00000000;

  IODIR1 = MOTOR;
  IODIR0 = IODIR0 & (~BUTTON);

  PLL0CON = 0x01;
  PLL0CFG = 0x24;
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;
  while(!(PLL0STAT & 0x00000400));
  PLL0CON = 0x03;
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;
  VPBDIV = 0x01;

  while(1)
  {
    while((IOPIN0 & BUTTON)==0);
    i=~i;


    if(i==0xFF)
    {
      IOSET1 = (1<<18);
      IOCLR1 = (1<<19);
    }
    else
    {
      IOSET1 = (1<<19);
      IOCLR1 = (1<<18);
    }

    while((IOPIN0 & BUTTON)==0x00010000);
  }
}
```




Lcd with arm lpc 2148

```
#include<lpc214x.h>
#define bit(x) (1<<x)
#define delay for(i=0;i<1000;i++);
unsigned int i;
void lcd_int();
void dat(unsigned char);
void cmd(unsigned char);
void string(unsigned char *);
void main()
{
   IO0DIR|=0XFFF;
   lcd_int();
   cmd(0x8a);
   string("EMBETRONICX.COM ");
   while(1) {
      cmd(0x18);
      delay;
   }
}
void lcd_int()
{
   cmd(0x30);
```

```
    cmd(0x0c);
    cmd(0x06);
    cmd(0x01);
    cmd(0x80);
}
void cmd(unsigned char a)
{
    IO0PIN&=0x00;
    IO0PIN|=(a<<0);
    IO0CLR|=bit(8);              //rs=0
    IO0CLR|=bit(9);              //rw=0
    IO0SET|=bit(10);              //en=1
    delay;
    IO0CLR|=bit(10);              //en=0
}
void dat(unsigned char b)
{
    IO0PIN&=0x00;
    IO0PIN|=(b<<0);
    IO0SET|=bit(8);              //rs=1
    IO0CLR|=bit(9);              //rw=0
    IO0SET|=bit(10);              //en=1
    delay;
    IO0CLR|=bit(10);              //en=0
}
void string(unsigned char *p)
{
    while(*p!='\0') {
        dat(*p++);
    }
}
```


LCD 16x2
VSS
VCC
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
LED+
LED-
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5V
3.3V
5V
10K
10K
P0.28(AD0.1)
VDD
P0.4
P0.5
P0.6
P0.8
P0.9
P0.10
P0.11
P0.12
P0.13
P0.14
P0.15
LPC2148
VSS