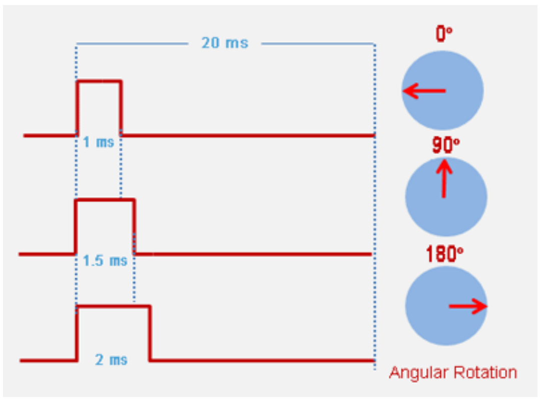
INTERFACING SERVO MOTOR USING LPC2148

Servo Motor

A Servo Motor is a combination of DC motor, position control system and gears. Servo motor's rotation is controlled by applying a PWM signal to it, width of the PWM signal decides the rotation angle and direction of the motor. Here we will be using **SG90 Servo Motor** in this tutorial, it is one of the popular and cheapest one. SG90 is a 180 degree servo. So with this servo we can position the axis from 0-180 degrees:

- Operating Voltage: +5V
- Gear Type: Plastic
- Rotation Angle: 0 to 180 deg
- Weight: 9gm
- Torque: 2.5kg/cm

We should program the MCU to send PWM signals to the signal wire of the Servo motor. There is a control circuitry inside the servo motor which reads the duty cycle of the PWM signal and positions the servo motors shaft in the respective place as shown in the picture below



- 1 ms (1 millisecond) pulse width for rotation of servo to **0 degree**

For every 20 milliseconds Servo motor checks the pulse. So, adjust the pulse width of the signal to rotate the motor’s shaft.

- 1.5ms pulse width for rotation to **90 degree** (neutral position)
- 2 ms pulse width for rotation of servo to **180 degree**.

	ADC Channel No.
P0.28	AD0.1
P0.29	AD0.2
P0.30	AD0.3
P0.25	AD0.4
P0.4	AD0.5
P0.5	AD0.6
P0.6	AD1.0
P0.8	AD1.1
P0.10	AD1.2
P0.12	AD1.3
P0.13	AD1.4
P0.15	AD1.5
P0.21	AD1.6
P0.22	AD1.7

	EDGE	START	RESERVED	PDN	RESERVED	CLKS	BURST
28-31	27	24-26	22,23	21	20	17,18,19	16

DONE	OVERRUN	RESERVED	CHN	RESERVED	RESULT	RESERVED
31	30	27-29	24-26	16-23	6-15	0-5

PWM Pins in ARM7-LPC2148

There are total six pins for PWM.

PWM Channel	Port Pins
PWM1	P0.0
PWM2	P0.7
PWM3	P0.1
PWM4	P0.8
PWM5	P0.21
PWM6	P0.9

```
PINSEL0 |= 0x00000008;  // Setting pin P0.
1 of LPC2148 as PWM3
```

PWM Registers in ARM7-LPC2148

Here is the list of registers used in LPC2148 for PWM

1. PWMPR: PWM Prescale Register

Use: It’s a 32-Bit register. It contains the number of times (minus 1) PCLK must cycle before incrementing the PWM Timer Counter (It actually holds maximum value of prescale counter).

2. PWMPC: PWM Prescaler Counter

Use: It a 32-bit register**.** It contains the incrementing counter value. When this value equals the PR value plus 1, the PWM Timer Counter (TC) is incremented.

3. PWMTCR: PWM Timer Control Register

Use: It contains the Counter Enable, Counter Reset and the PWM Enable control bits. **It is an 8-Bit register.**

7:4	**3**	**2**	**1**	**0**

RESERVED	PWM ENABLE	RESERVED	COUNTER RESET	COUNTER ENABLE

- **PWM Enable: (Bit-3)**0- PWM Disabled1- PWM Enabled
- **Counter Enable: (Bit-0)**0- Disable Counters1- Enable Counter
- **Counter reset: (Bit-1)**0- Do Nothing.1- Resets PWMTC & PWMPC on positive edge of PCLK.

4. PWMTC: PWM Timer Counter

Use: It's a 32-Bit register. It contains the current value of the   incrementing PWM Timer. When the Prescaler Counter (PC) reaches the Prescaler Register (PR) value plus 1, this counter is incremented.

5. PWMIR: PWM Interrupt Register

Use: It's a 16-Bit Register. It contains the interrupt flags for PWM Match Channels 0-6. An interrupt flag is set when an interrupt occurs for that channel (MRx Interrupt) where X is the channel number (0 to 6).

6. PWMMR0-PWMMR6: PWM Match Register

Use: It's a 32-Bit register**.** Actually the Match Channel group allows setting 6 single-edge controlled or 3 double-edge controlled PWM outputs. You may modify the seven Match Channels to configure these PWM outputs to suit your requirements in PWMPCR.

31:21	20	19	18	..	5	4	3
RESERVED	PWMMR6S	PWMMR6R	PWMMR6I	..	PWMMR1S	PWMMR1R	PWMMR

Here x is from 0 to 6

- **PWMMRxI (Bit-0)**

ENABLE OR DISABLE PWM interrupts

0- Disable PWM Match interrupts.

1- Enable PWM Match interrupt.

- **PWMMRxR :(Bit-1)**

RESET PWMTC -Timer counter value whenever it matches PWMRx

0- Do Nothing.

1- Resets the PWMTC.

- **PWMMRxS :(Bit 2)**

STOP PWMTC & PWMPC when PWMTC reaches the Match register value

0- Disable the PWM stop feature.

1- Enable the PWM Stop feature.

8. PWMPCR: PWM Control Register

Use: It's a 16-Bit register. It contains the bits that enable PWM outputs 0-6 and select single-edge or double-edge control for each output.

31:15	14:9	8:7	6:2	1:0
UNUSED	PWMENA6-PWMENA1	UNUSED	PWMSEL6-PWMSEL2	UNUSED

- **PWMSELx (x: 2 to 6)**

1. Single Edge mode for PWMx
2. 1- Double Edge Mode for PWMx.

- **PWMENAx (x:1 to 6)**

1. PWMx Disable.
2. 1- PWMx Enabled.

9. PWMLER: PWM Latch Enable Register

Use: It's an 8-Bit Register. It contains the Match x Latch bits for each Match Channel.

31:7	6	5	4	3	2	1	0

UNUSED	LEN6	LEN5	LEN4	LEN3	LEN2	LEN1	LEN0

LENx (x:0 to 6):

0- Disable the loading of new Match Values

1- Load the new Match values from (PWMMRx) PWMMatch Register when the timer is reset.

Now lets start building the hardware setup to demonstrate the Pulse Width Modulation in ARM microcontroller.

Step 1:- Include the necessary header files for coding LPC2148

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

Step 2:- Next thing is to **configure the PLL** for clock generation as it sets the system clock and peripheral clock of LPC2148 as per programmers need. The maximum clock frequency for LPC2148 is 60Mhz. Following lines are used to configure PLL clock generation.

```
voidinitilizePLL (void) //Function to use PLL for clock generation
{
    PLL0CON = 0x01;
    PLL0CFG = 0x24;
    PLL0FEED = 0xAA;
    PLL0FEED = 0x55;
    while(!(PLL0STAT & 0x00000400));
    PLL0CON = 0x03;
    PLL0FEED = 0xAA;
    PLL0FEED = 0x55;
    VPBDIV = 0x01;
}
```

Step 3:- Next thing to do is select the PWM pins and PWM function of LPC2148 by using PINSEL register. We use PINSEL0 as we use P0.1 for PWM output of LPC2148.

Step 4:- Next, we need to RESET the timers using PWMTCR (Timer Control Register).

```
PWMTCR = 0x02;  // Reset and disable counter for PWM
```

And then next set the prescale value which decides the resolution of PWM is set.

```
PWMPR = 0x1D;  // Prescale Register value
```

Step 5:- Next, set the PWMMCR (PWM match control register) as it sets operation like reset, interrupts for PWMMR0 and PWMMR3.

```
PWMMCR = 0x00000203;// Reset and interrupt on MR0 match, interrupt on MR3 match
```

Step 6:- The maximum period of the PWM channel is set using PWMMR0 and the Ton of the PWM duty cycle is initially set to 0.65msec

```
PWMMR0 = 20000;// Time period of PWM wave, 20msec
PWMMR3 = 650; // Ton of PWM wave 0.65 msec
```

Step 7:- Next, we need to set the Latch Enable to the corresponding match registers using PWMLER

```
PWMLER = 0x09;// Latch enable for PWM3 and PWM0
```

(We use PWMMR0 & PWMMR3) So enable the corresponding bit by setting 1 in PWMLER

Step 8:- To enable the PWM output to the pin we need to use the PWMTCR for enabling the PWM Timer counters and PWM modes.

```
PWMPCR = 0x0800;     // Enable PWM3 and PWM 0, single edge controlled
PWMPWMTCR = 0x09;    // Enable PWM and counter
```

Step 9:- Now we need to get the potentiometer values for setting duty cycle of PWM from ADC pin P0.28. So, we use ADC module in LPC2148 for converting potentiometers analog input (0 to 3.3V) to the ADC values (0 to 1023).

Step 10:- For **selecting ADC pin P0.28 in LPC2148,** we use

```
PINSEL1 = 0x01000000;//Setting P0.28 as ADC INPUT
AD0CR = (((14)<<8) | (1<<21)); //Setting clock and PDN for A/D Conversion
```

The following lines **capture the Analog input (0 to 3.3V)** and convert it into digital value (0 to 1023). And then this digital values are divided by 4 to convert them into **(0 to 255)** and finally fed as **PWM output in P0.1 pin** of LPC2148. Here we are **converting the values from 0-1023 to 0-255 by dividing it with 4 as PWM of LPC2148 has 8-Bit resolution (28)**.

```
AD0CR |= (1<<1); //Select AD0.1 channel in ADC register
delaytime(10);
AD0CR |= (1<<24); //Start the A/D conversion
while( (AD0DR1 & (1<<31)) == 0 );//Check the DONE bit in ADC Data register
    adcvalue = (AD0DR1>>6) & 0x3ff; //Get the RESULT from ADC data register
    dutycycle = adcvalue/4;//formula to get dutycycle values from (0 to 255)
    PWMMR1 = dutycycle; //set dutycycle value to PWM match register
    PWMLER |= (1<<1); //Enable PWM output with dutycycle value
```

Step 11:- Next, we display those values in the LCD (16X2) Display module. So we add the following lines to initializes LCD display module

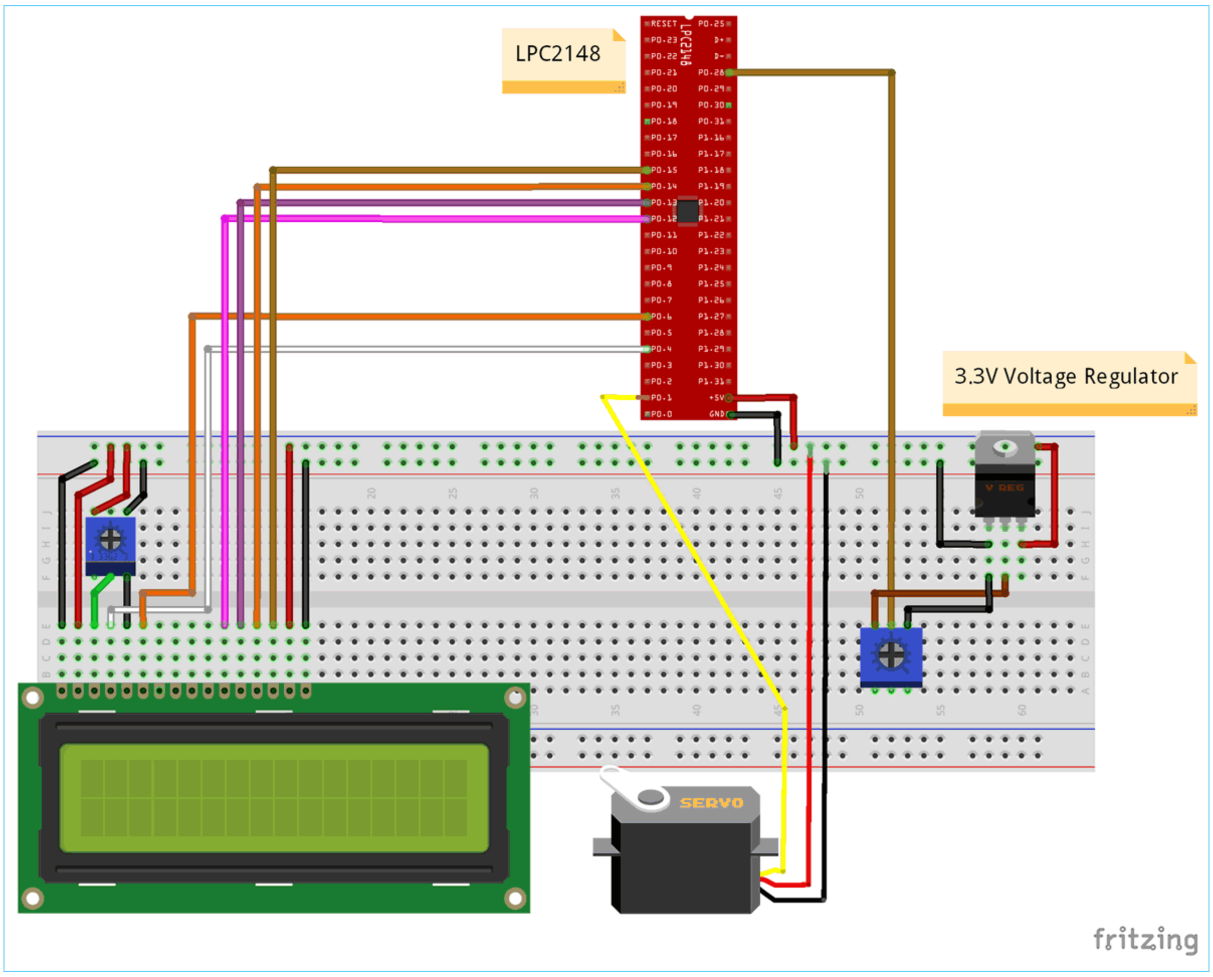
```
void LCD_INITILIZE(void)  //Function to get ready the LCD
{
    IO0DIR = 0x0000FFF0; //Sets pin P0.12,P0.13,P0.14,P0.15,P0.4,P0.6 as OUTPUT
    delaytime(20);
    LCD_SEND(0x02);  // Initialize lcd in 4-bit mode of operation
    LCD_SEND(0x28);  // 2 lines (16X2)
    LCD_SEND(0x0C);   // Display on cursor off
    LCD_SEND(0x06);  // Auto increment cursor
    LCD_SEND(0x01);   // Display clear
    LCD_SEND(0x80);  // First line first position
}
```

As we connected LCD in 4-Bit mode with LPC2148 we need to send values to be displayed as nibble by nibble (Upper Nibble & Lower Nibble). So following lines are used.

```
void LCD_DISPLAY (char* msg) //Function to print the characters sent one by one
{
    uint8_t i=0;
    while(msg[i]!=0){
        IO0PIN = ((IO0PIN & 0xFFFF00FF)|((msg[i]&0xF0)<<8)); //Sends Upper nibble
        IO0SET = 0x00000050;  //RS HIGH & ENABLE HIGH to print data
        IO0CLR = 0x00000020;  //RW LOW Write mode
        delaytime(2);IO0CLR = 0x00000040; // EN = 0, RS and RW unchanged(i.e. RS = 1, RW =
0)
        delaytime(5);
        IO0PIN = ( (IO0PIN & 0xFFFF00FF) | ((msg[i] & 0x0F)<<12) ); //Sends Lower nibble
        IO0SET = 0x00000050; //RS & EN HIGH
        IO0CLR = 0x00000020;
        delaytime(2);
        IO0CLR = 0x00000040;
        delaytime(5);i++;
    }
}
```

To display those ADC & PWM values we use following lines in the *int main()* function.

```
LCD_SEND(0x80);
sprintf(displayadc, "adcvalue=%f", dutycycle);
LCD_DISPLAY(displayadc);                                  //Display ADC value (0 to 1023)
angle =  (adcvalue/5.7);                       //Formula to convert ADC value into angle
(o to 180 deg)
LCD_SEND(0xC0);
sprintf(anglevalue, "ANGLE=%.2f deg  ", angle);
LCD_DISPLAY(anglevalue);
```



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

```
#include <string.h>

void initilizePLL(void);
void initilizePLL (void)//Function to use PLL for clock generation
{
  PLL0CON = 0x01;
  PLL0CFG = 0x24;
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;
  while(!(PLL0STAT & 0x00000400));
  PLL0CON = 0x03;
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;
  VPBDIV = 0x01;
}

 void delay_ms(uint16_t z) // function to generate 1 milisecond delay with Cclk (60MHz)
 {
    uint16_t x,i;
        for(i=0;i<z;i++){
            for(x=0; x<6000; x++);
            }
        }
        __irq void PWM_ISR (void){

                if ( PWMIR & 0x0001 )
                {
                        PWMIR = 0x0001;
                }
                if ( PWMIR & 0x0008 )
                {
                PWMIR = 0x0008;
                }

VICVectAddr = 0x00000000;
}

void LCD_SEND(char command)                                  //Function to send hex commands
{
        IO0PIN = ( (IO0PIN & 0xFFFF00FF) | ((command & 0xF0)<<8) );  //Send upper nibble of c
        IO0SET = 0x00000040;                                         //Making Enable HIGH
        IO0CLR = 0x00000030;                                         //Making RS & RW LOW
        delay_ms(5);
        IO0CLR = 0x00000040;                                         //Makeing Enable LOW
        delay_ms(5);
        IO0PIN = ( (IO0PIN & 0xFFFF00FF) | ((command & 0x0F)<<12) ); //Send Lower nibble of c
        IO0SET = 0x00000040;                                         //ENABLE HIGH
        IO0CLR = 0x00000030;                                         //RS & RW LOW
        delay_ms(5);
        IO0CLR = 0x00000040;                                         //ENABLE LOW
        delay_ms(5);
}

void LCD_INITILIZE(void)                                     //Function to get ready the LCD
{
        IO0DIR = 0x0000FFF0;                                         //Sets pin P0.12,P0.13,P
        delay_ms(20);
        LCD_SEND(0x02);                                              // Initialize lcd in 4-b
        LCD_SEND(0x28);                                              // 2 lines (16X2)
        LCD_SEND(0x0C);                                              // Display on cursor off
        LCD_SEND(0x06);                                              // Auto increment cursor
        LCD_SEND(0x01);                                              // Display clear
        LCD_SEND(0x80);                                              // First line first posi
}

void LCD_DISPLAY (char* msg)                              //Function to print the characte
{
    uint8_t i=0;
    while(msg[i]!=0)
    {
            IO0PIN = ( (IO0PIN & 0xFFFF00FF) | ((msg[i] & 0xF0)<<8) );   //Sends Upper nibble
            IO0SET = 0x00000050;                                         //RS HIGH & ENABLE H
            IO0CLR = 0x00000020;                                         //RW LOW Write mode
            delay_ms(2);
            IO0CLR = 0x00000040;                                         // EN = 0, RS and RW
            delay_ms(5);
            IO0PIN = ( (IO0PIN & 0xFFFF00FF) | ((msg[i] & 0x0F)<<12) );  //Sends Lower nibble
            IO0SET = 0x00000050;                                         //RS & EN HIGH
            IO0CLR = 0x00000020;
            delay_ms(2);
            IO0CLR = 0x00000040;
            delay_ms(5);
            i++;
    }
}

int main()
{

    LCD_INITILIZE();                                //Calls function to get ready the LCD to d
    char displayadc[18];
    float adc;
    float angle;
    char anglevalue[18];
    LCD_DISPLAY("CIRCUIT DIGEST");
    delay_ms(900);
    LCD_SEND(0xC0);
    LCD_DISPLAY("SERVO LPC2148");
    delay_ms(900);

    PINSEL0 |= 0x00000008;                          // Setting pin P0.1 of LPC2148 as PWM3
    VICVectAddr0 = (unsigned) PWM_ISR;              // PWM ISR Address
    VICVectCntl0 = (0x00000020 | 8);                // Enable PWM IRQ slot
    VICIntEnable = VICIntEnable | 0x00000100;       // Enable PWM interrupt
    VICIntSelect = VICIntSelect | 0x00000000;       // PWM configured as IRQ

    PWMTCR = 0x02;                                  // Reset and disable counter for PWM
    PWMPR = 0x1D;                                   // Prescale Register value
    PWMMR0 = 20000;                                 // Time period of PWM wave, 20msec
    PWMMR3 = 650;                                 // Ton of PWM wave 0.65 msec
    PWMMCR = 0x00000203;                         // Reset and interrupt on MR0 match, interrup
    PWMLER = 0x09;                                 // Latch enable for PWM3 and PWM0
    PWMPCR = 0x0800;                         // Enable PWM3 and PWM 0, single edge controlled
    PWMTCR = 0x09;                                 // Enable PWM and counter

  float dutycycle;
```

```
   unsigned short int adcvalue;

     PINSEL1 = 0x01000000;                          //Setting P0.28 as ADC INPUT (from potenti
     AD0CR = (((14)<<8) | (1<<21));                 //Setting clock and PDN for A/D Conversion
     PWMPCR |= (1<<9);                              //To enable PWM3 output at pin P0.1 of LPC
     while(1)
     {

         AD0CR |= (1<<1);                               //Select AD0.1 channel in ADC register
         AD0CR |= (1<<24);                              //Start the A/D conversion
         while( (AD0DR1 & (1<<31)) == 0 );              //Check the DONE bit in ADC Data regis
         adcvalue = (AD0DR1>>6) & 0x3ff;                //Get the RESULT from ADC data registe
         dutycycle = (adcvalue/4);                      //formula to get dutycycle values from
         PWMMR3 = dutycycle;                            //set dutycycle value to PWM match reg
         PWMLER = 0x08;                                 //Enable PWM output with dutycycle val
         delay_ms(50);
         LCD_SEND(0x80);
         sprintf(displayadc, "adcvalue=%f", dutycycle);
         LCD_DISPLAY(displayadc);        //Display ADC value (0 to 1023)
         angle =  (adcvalue/5.7);                       //Formula to convert ADC value into an
         LCD_SEND(0xC0);
         sprintf(anglevalue, "ANGLE=%.2f deg  ", angle);
         LCD_DISPLAY(anglevalue);                       //Display angle value

    }
 }
```