CONTENTS

1. INTRODUCTION 2

2. EXECUTION STEPS 3

3. EXPERIMENTS 13

1: Display Hello word in UART 13

2:-Speed Control of the DC Motor 17

3:-Stepper Motor Interface and rotate clockwise and anticlockwise 22

4:-Display digital output for given analog input using internal ADC 27

5:-Interface DAC and generate Triangular and Square Waveform 37

6:-Interface 4X4 Matrix Keypad and display in LCD 43

7:-Generate PWM waveform and vary its duty cycle 46

8:-Using external interrupt switches toggle the led's 50

9:-Display 0-F in 7 segment display 54

10:-Interface a switch and display status in LED, Relay and Buzzer 55

11:-Interface SPI ADC and display Ambient temperature

4. Viva Question 73

INTRODUCTION

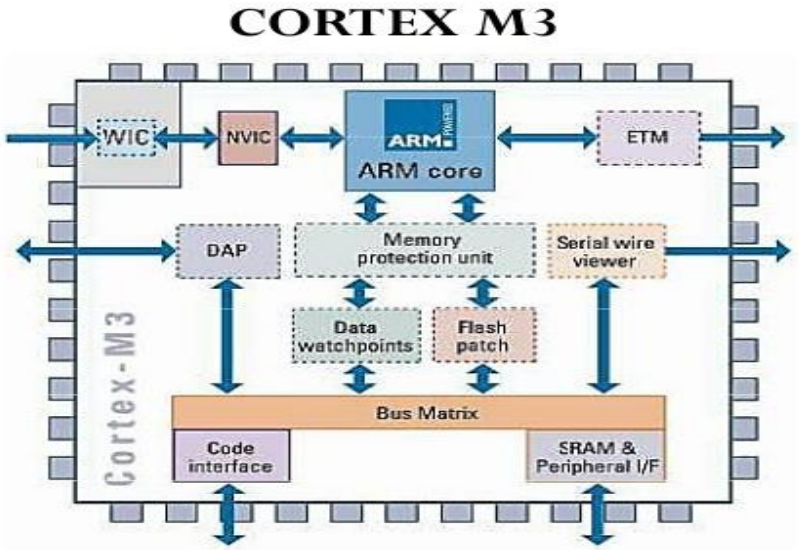
Microcontroller or Microprocessor is an electronic device which accepts data from memory or input devices, process it according to instruction and sends or store result either in output devices or memory.

Microprocessor contains no RAM, no ROM, and I/O ports on the chip itself only CPU or processor is present. To make it functional all should be added.

ARM Cortex M3 Series:

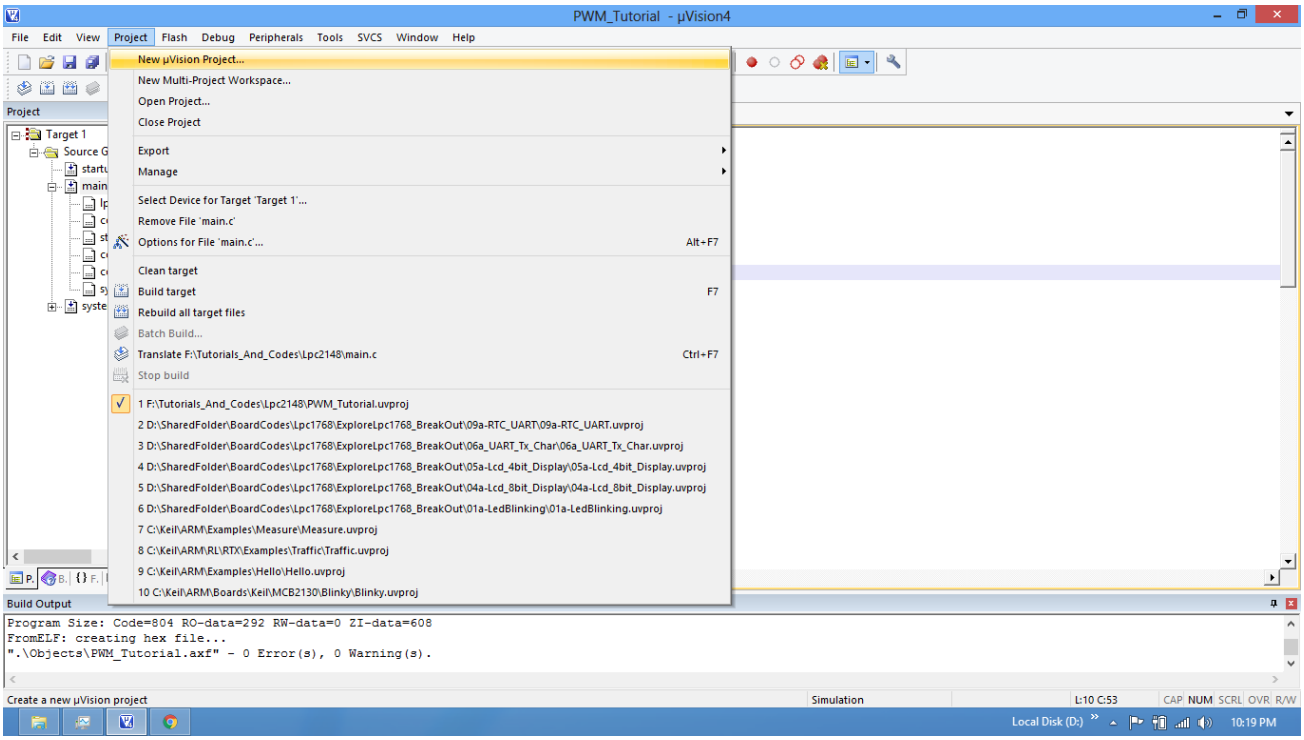
ARM was founded in 1990 as Advanced RISC Machines Ltd., a joint venture of Apple Computer, Acorn Computer Group, and VLSI Technology. In 1991, ARM introduced the ARM6 processor family, and VLSI became the initial licensee. Subsequently, additional companies, including Texas Instruments, NEC, Sharp and ST Microelectronics, licensed the ARM processor designs

Nowadays ARM partners ship in excess of 2 billion ARM processors each year. Unlike many semiconductor companies, ARM does not manufacture processors or sell the chips directly. Instead it licenses the processor designs to business partners. This business model is commonly called Intellectual

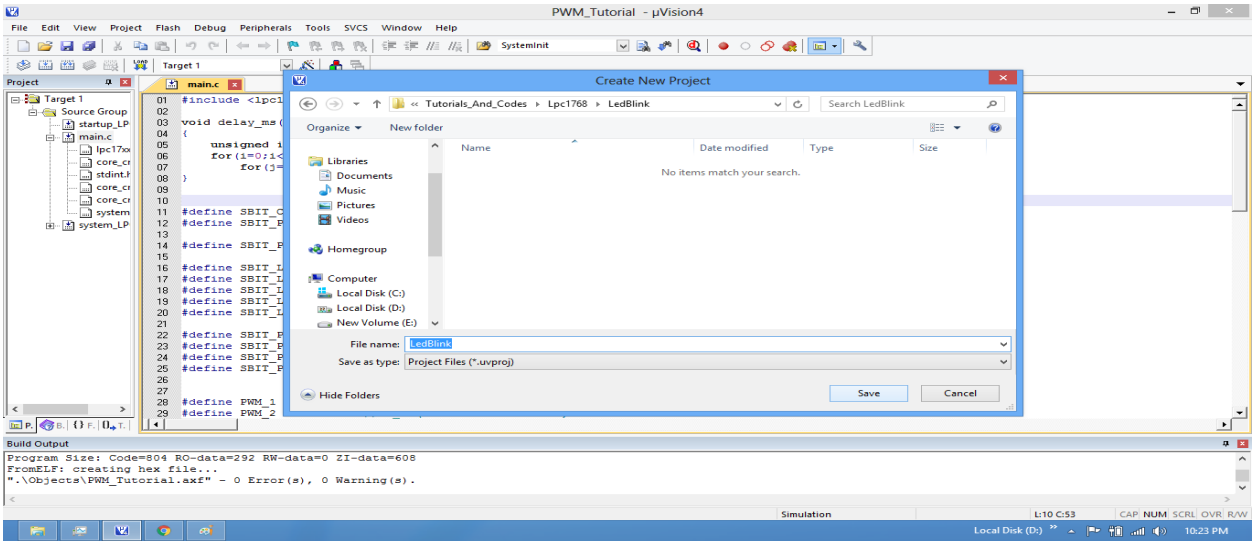


2. EXECUTION STEPS

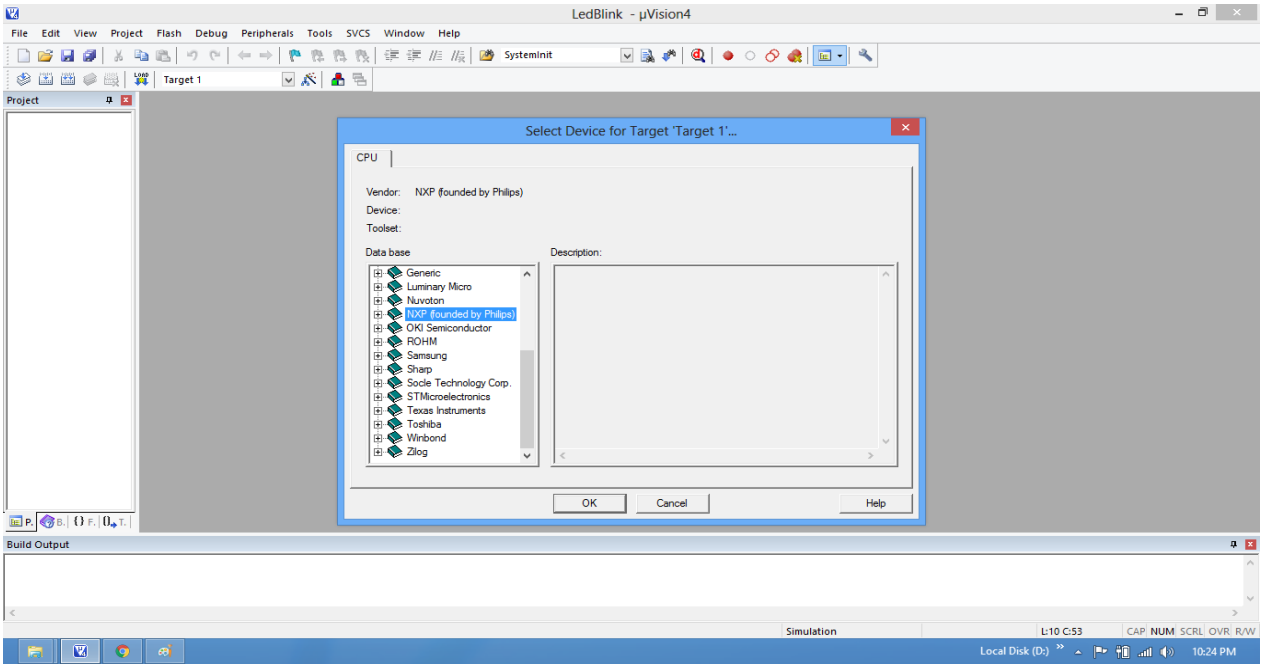
Step1: Open the Keil software and select the New Microvision project from Project Menu as shown below.



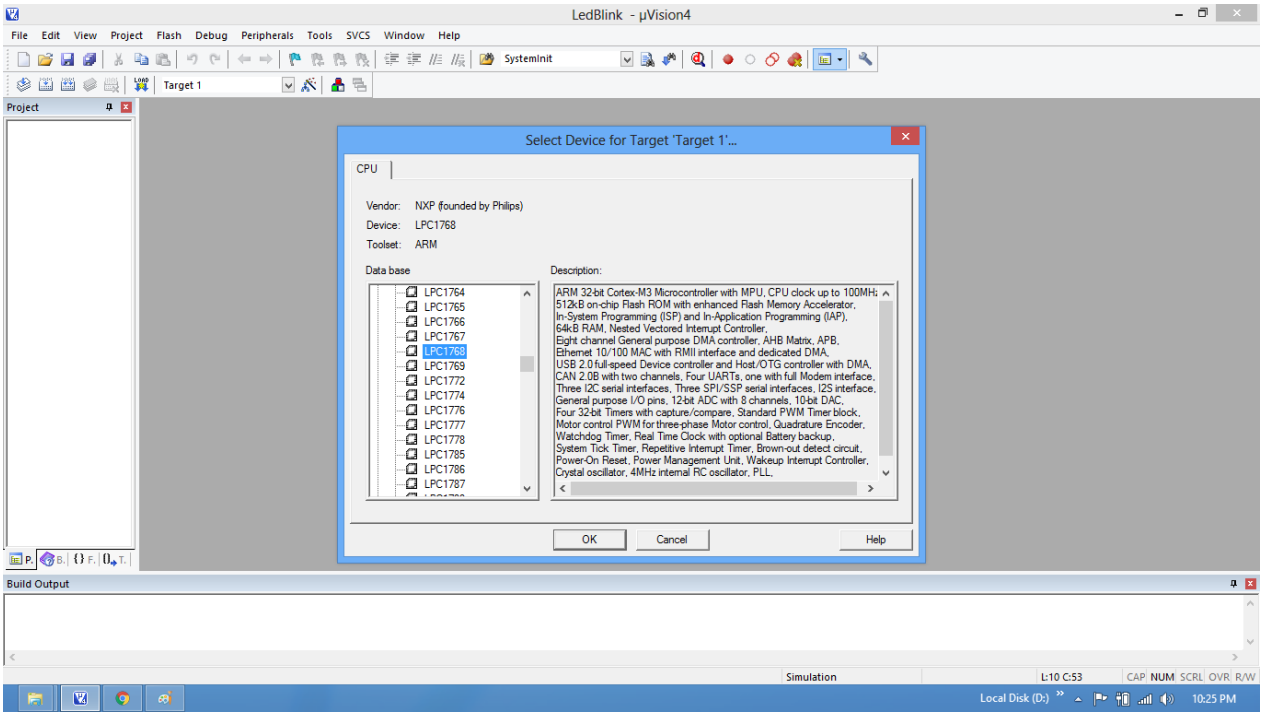
Step2: Browse to your project folder and provide the project name and click on save.



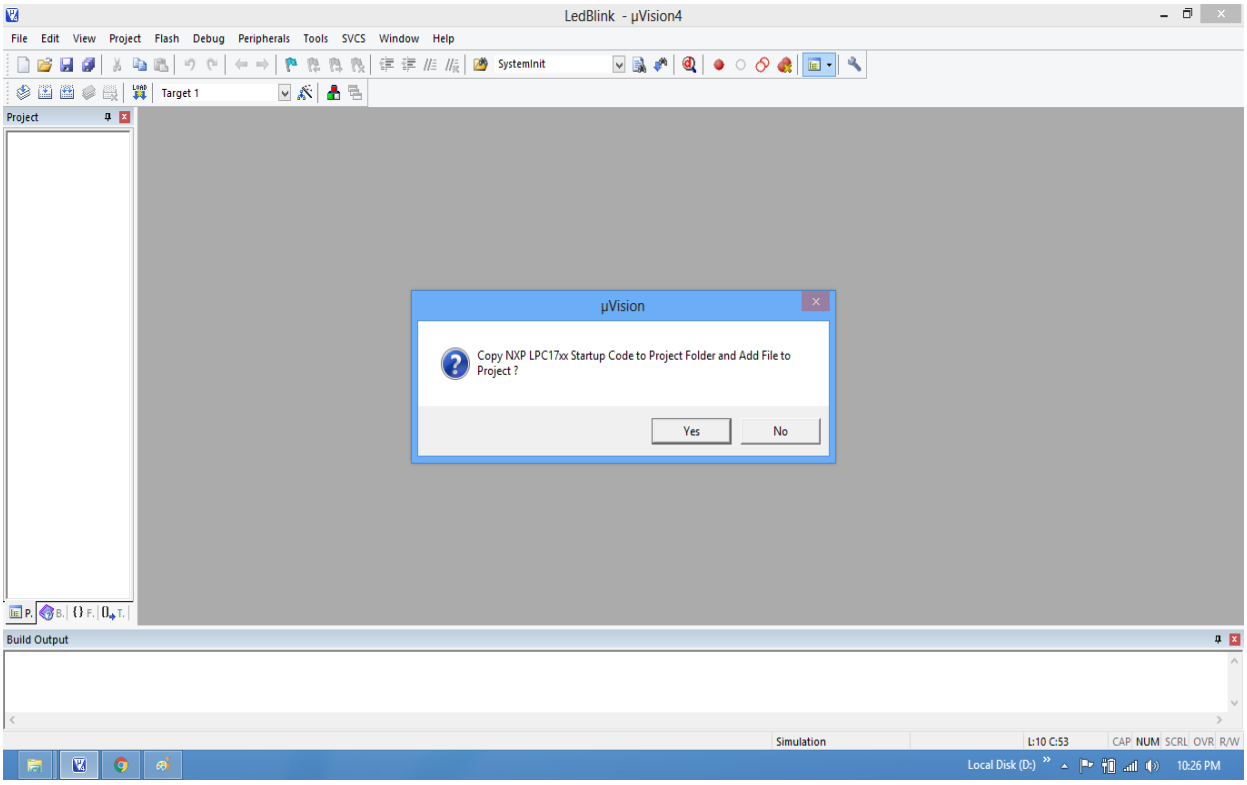
Step3: Once the project is saved a new pop up "Select Device for Target" opens, Select the controller(NXP:LPC1768) and click on OK.



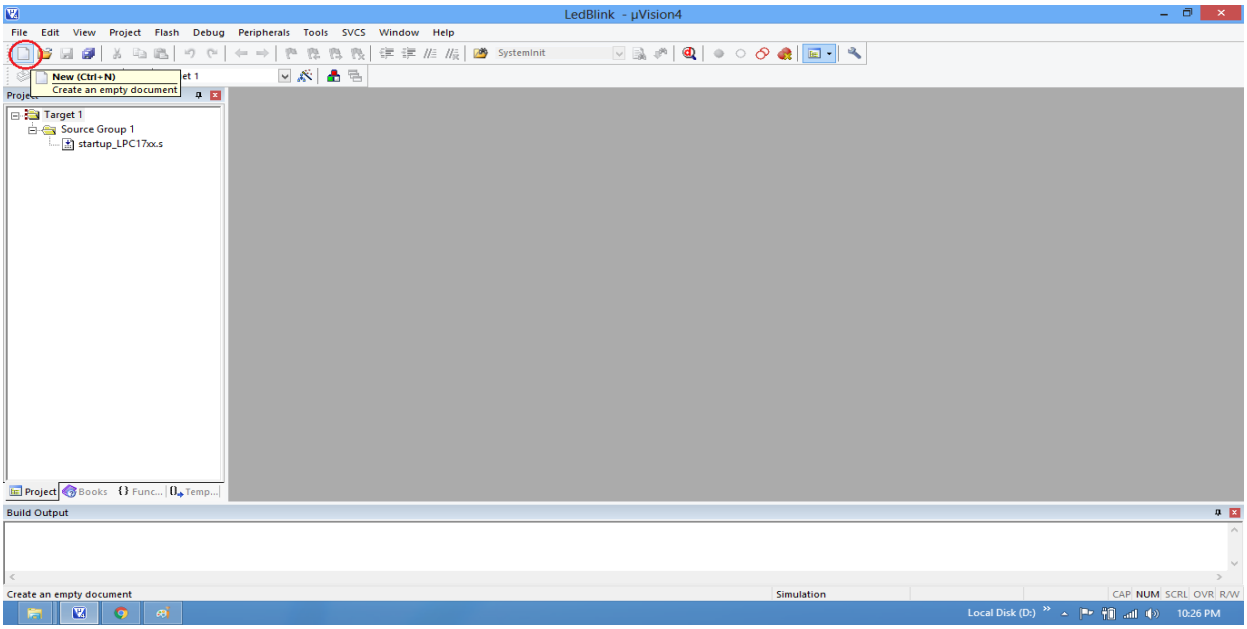
Step4: Select the controller(NXP:LPC1768) and click on OK.



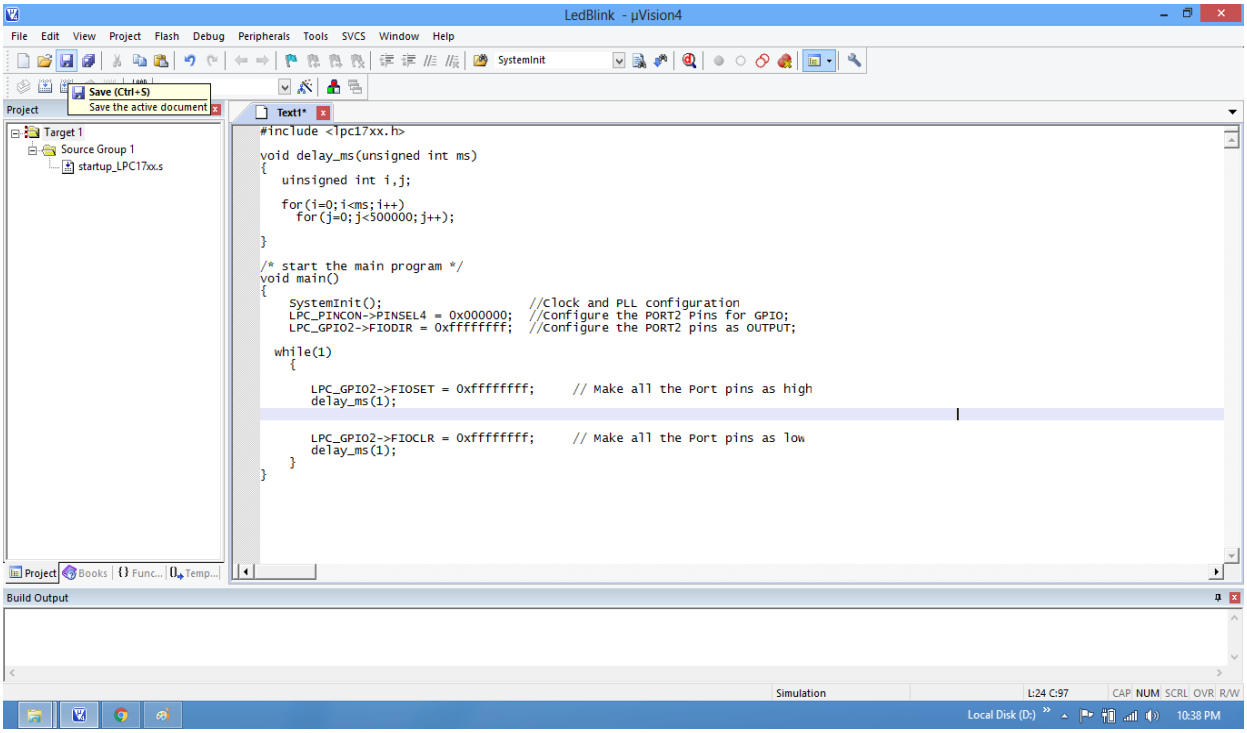
Step5: As LPC1768 needs the startup code, click on **Yes** option to include the **LPC17xx Startup** file.



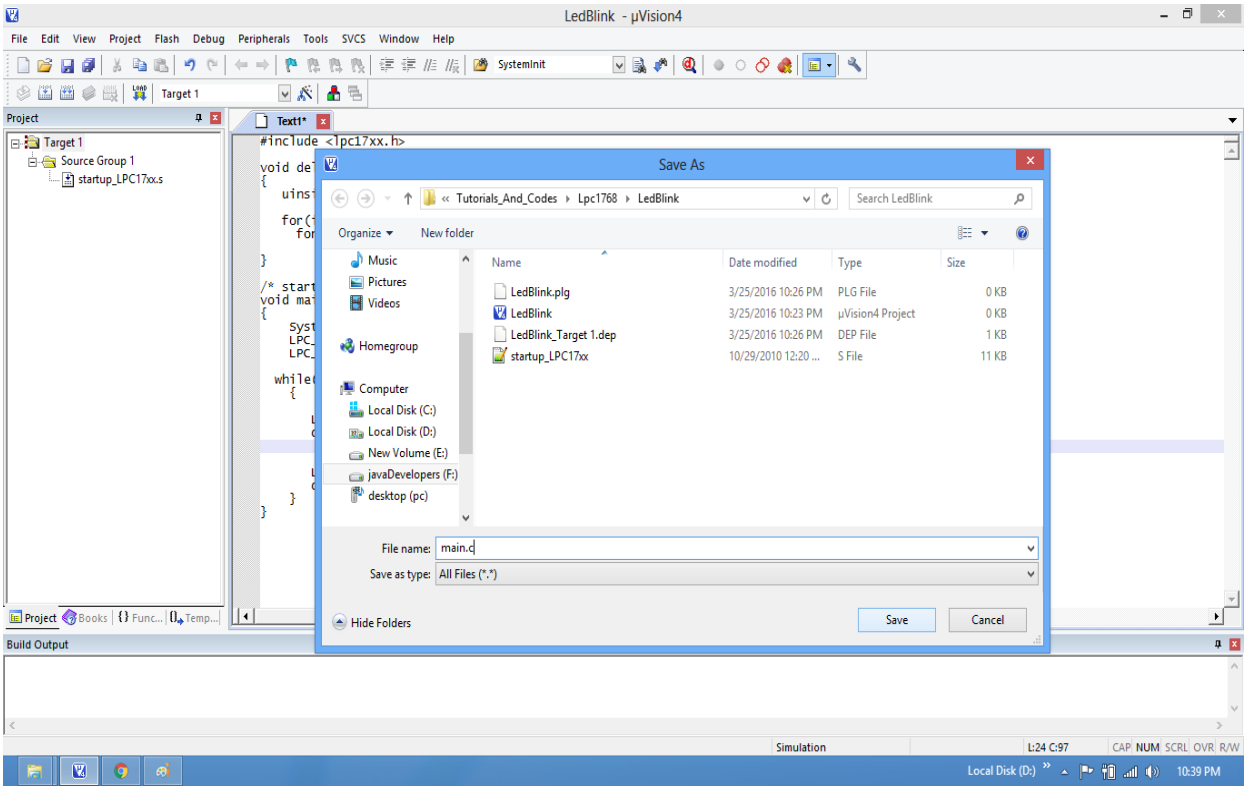
Step6: Create a new file to write the program.



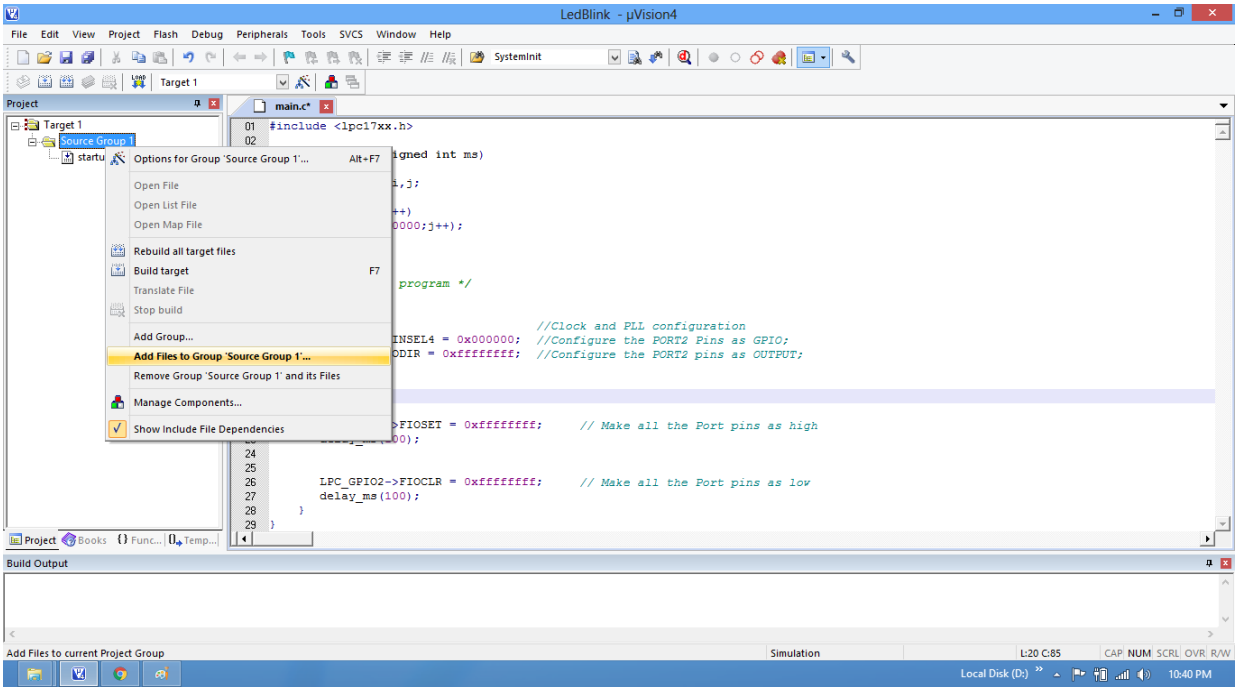
Step7: Type the code or Copy paste the below code snippet.



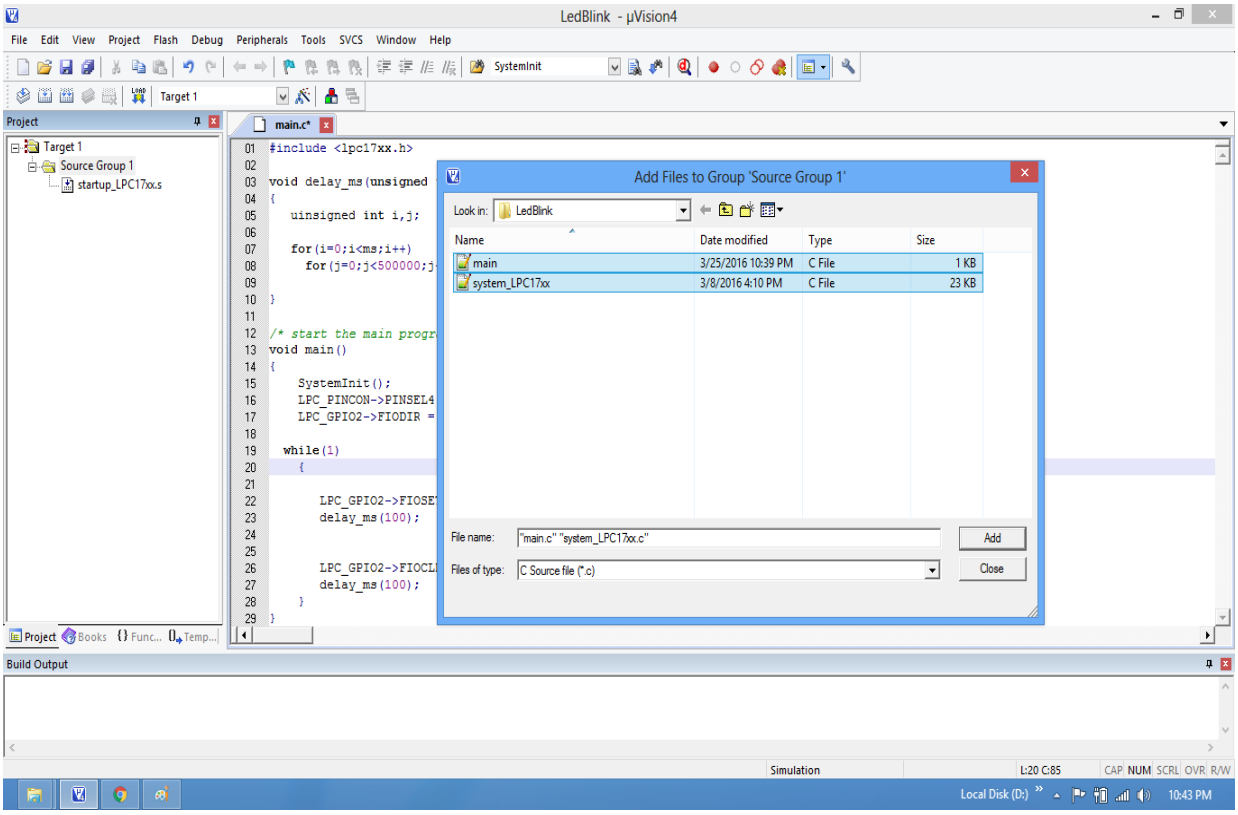
Step8: After typing the code save the file as **main.c**.



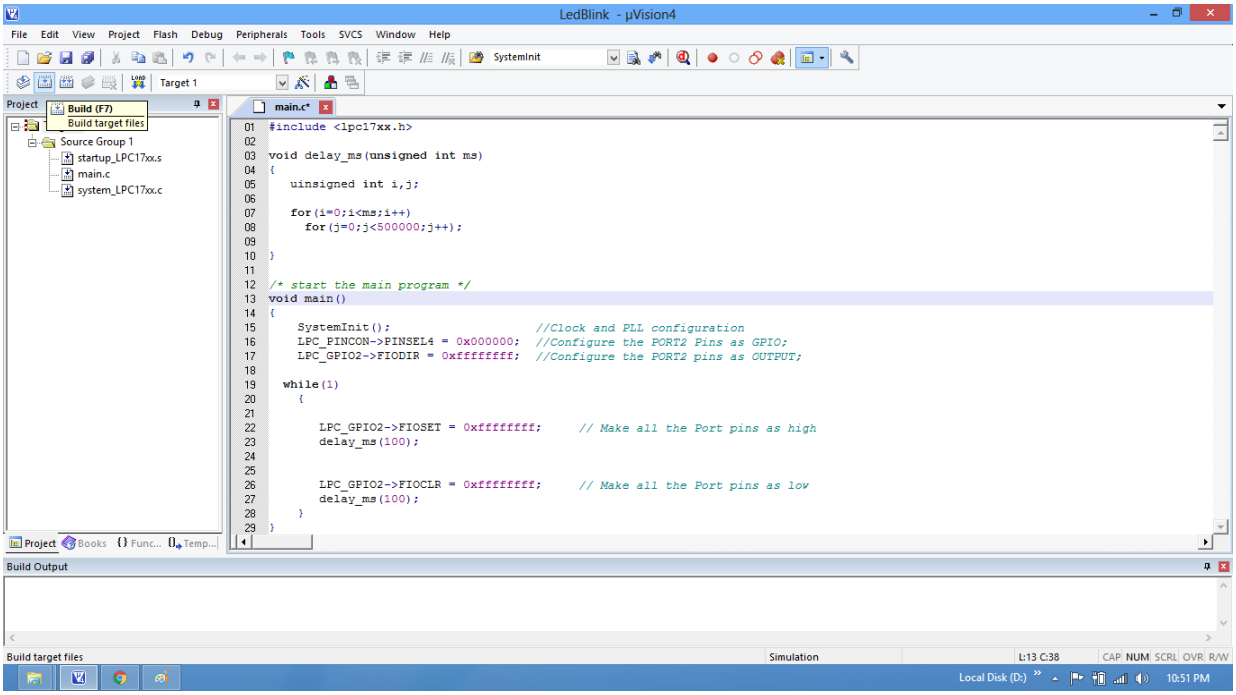
Step9: Add the recently saved file to the project.



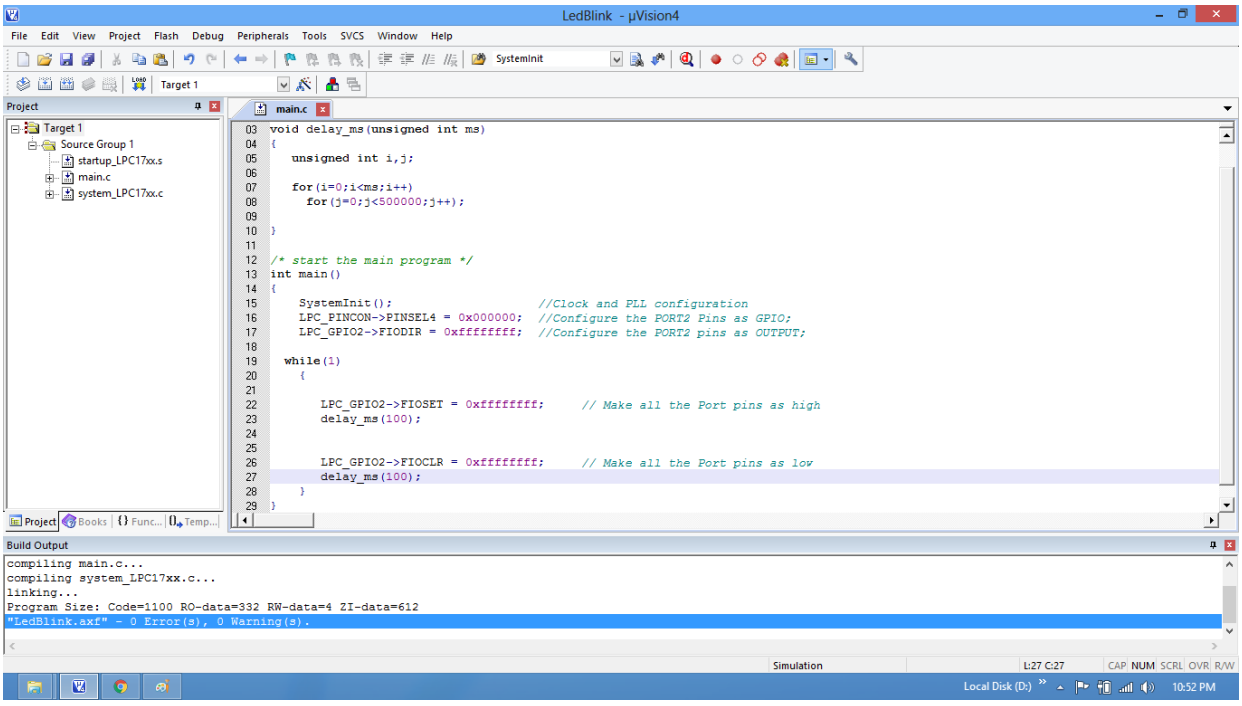
Step10: Add the main.c along with system_LPC17xx.c.



Step11: Build the project and fix the compiler errors/warnings if any.

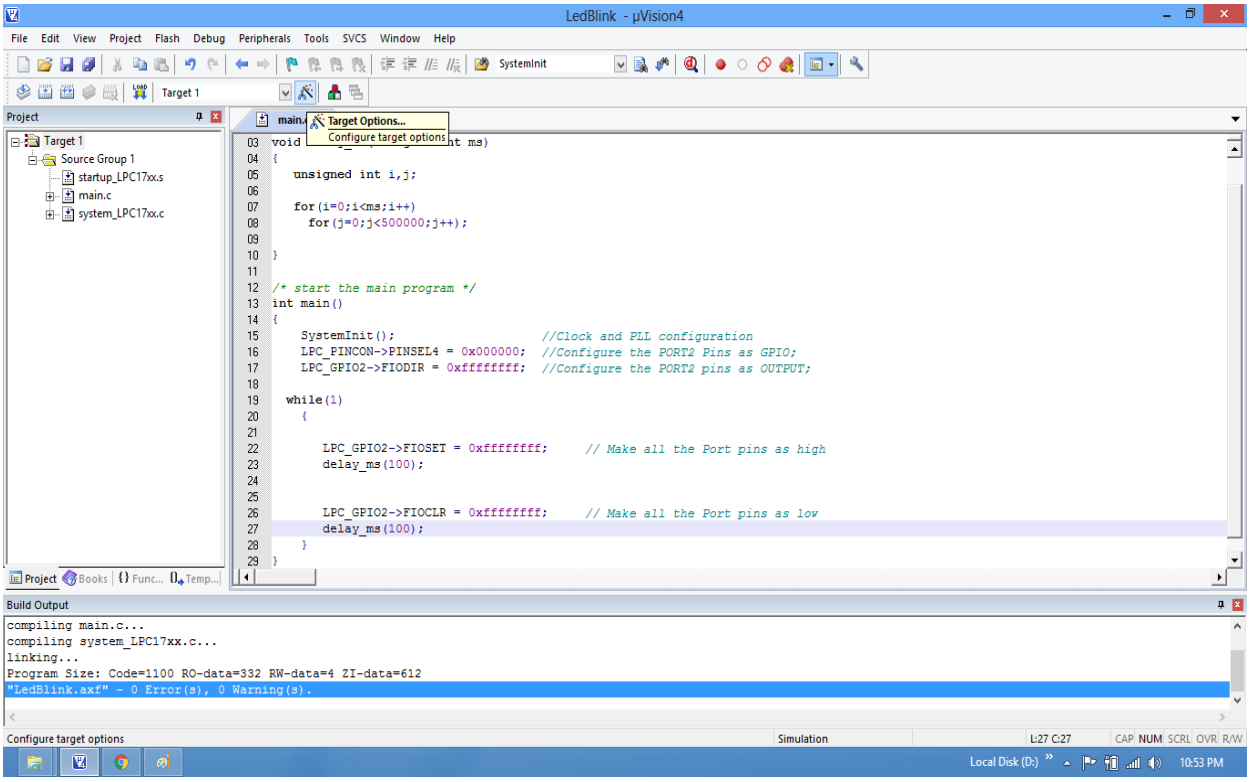


Step12: Code is compiled with no errors. The **.hex** file is still not generated.

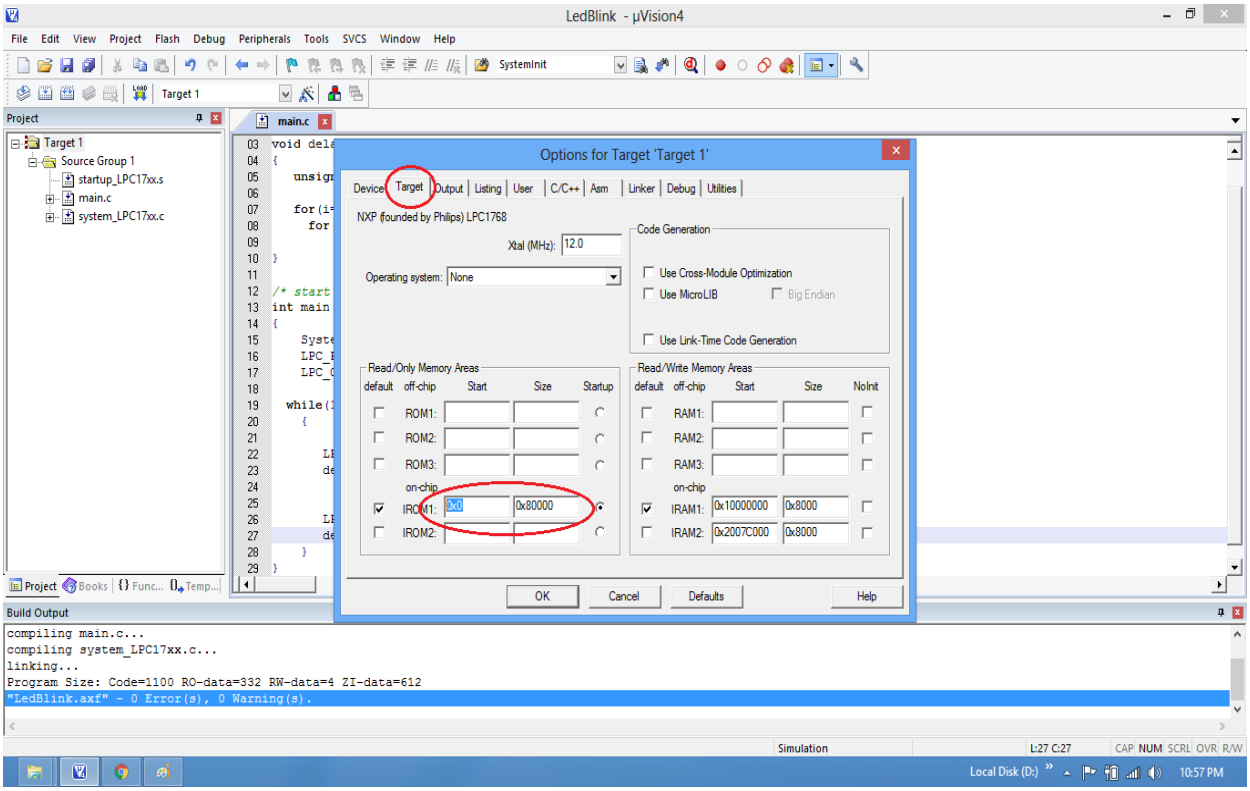


Enable Hex File Generation

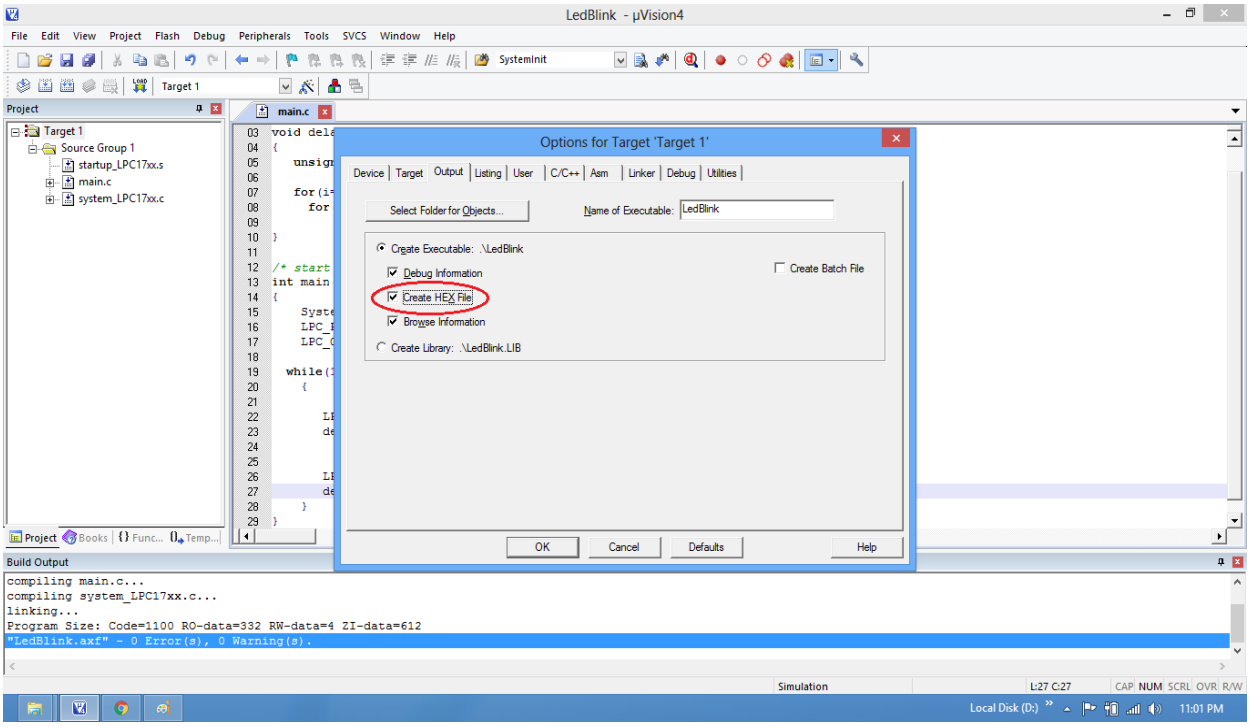
Step13: Click on **Target Options** to select the option for generating .hex file.



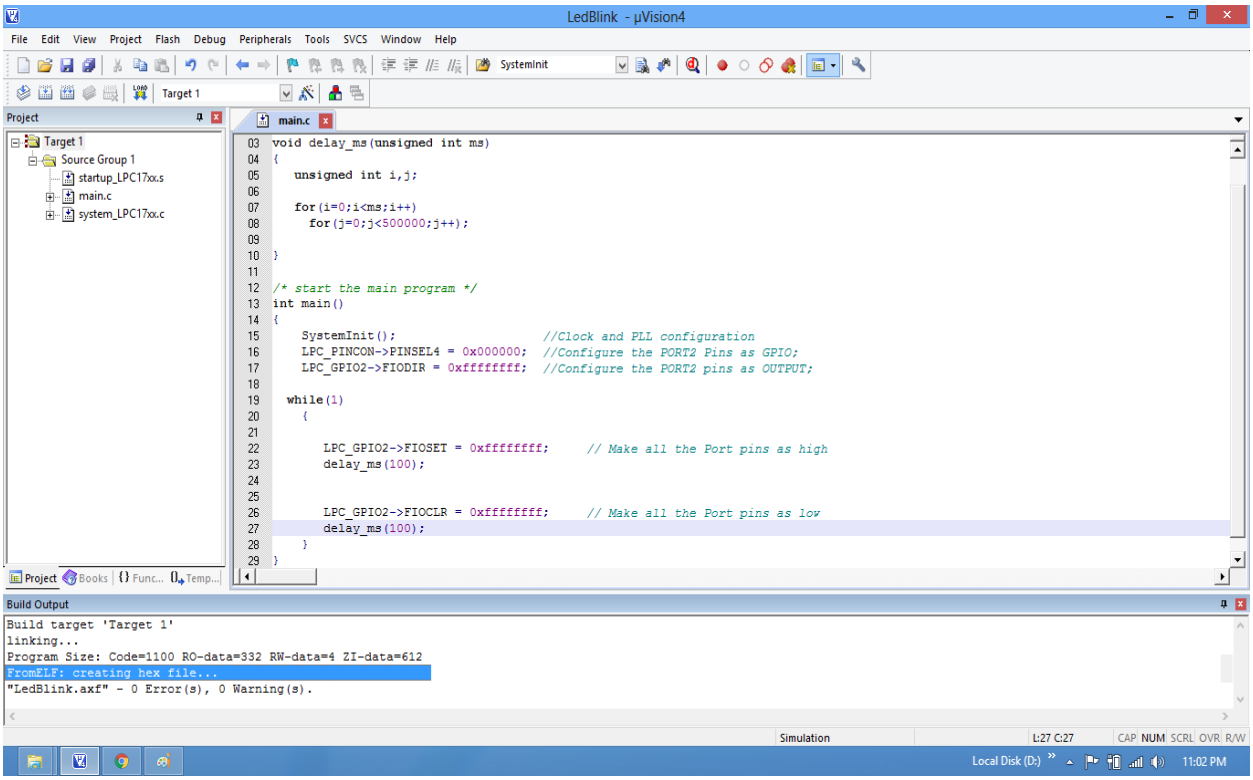
Step14: Set IROM1 start address as 0x0000.



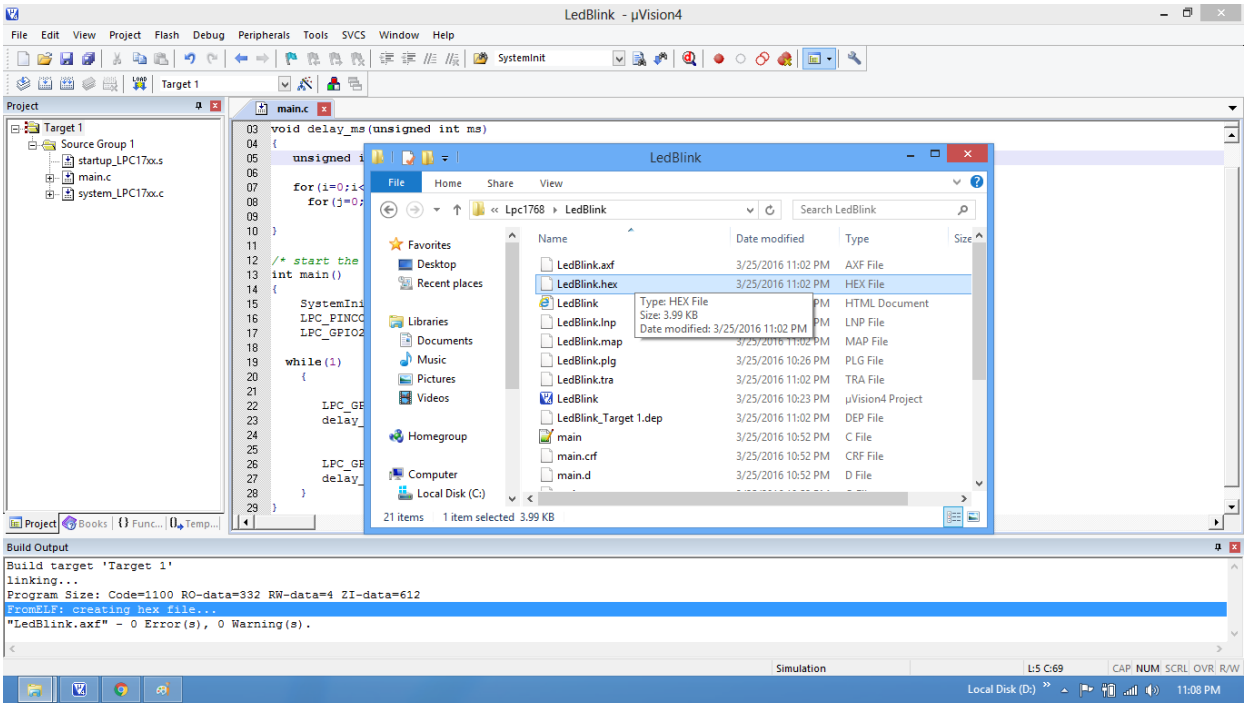
Step15: Enable the option to generate the .hex file



Step16: .Hex file is generated after a rebuild.



Step17: Check the project folder for the generated .hex file.

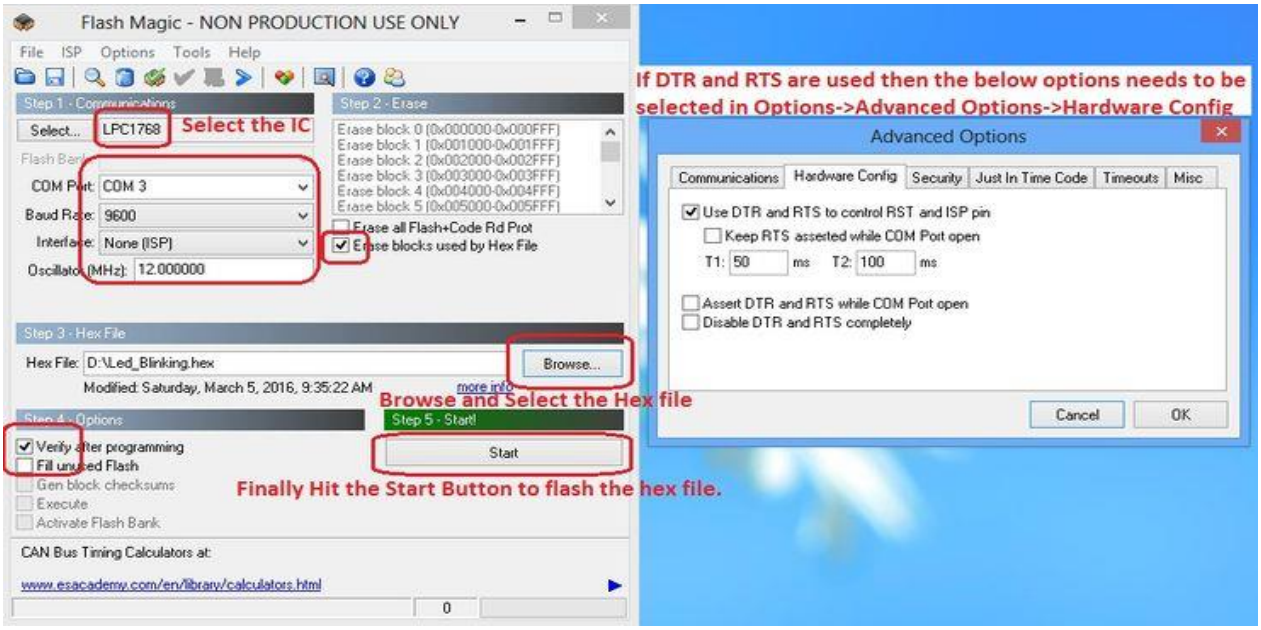


Now .hex file with project name will be generated.

Working with Flash Magic Software :-

Now open the flash magic software and follow the below steps.

1. Select the IC from Select Menu(LPC1768).
2. Select the COM Port. **Check the device manger for detected Com port**.
3. Select Baud rate from 9600
4. Select None Isp Option.
5. Oscillator Freq 12.000000(12Mhz).
6. Check the Erase blocks used by Hex file option
7. Browse and Select the hex file.
8. Check the Verify After Programming Option.
9. If DTR and RTS are used then go to Options->Advanced Options-> Hardware Config and select the Use DTR and RTS Option.
10. Hit the Start Button to flash the hex file.
11. Once the hex file is flashed, Reset the board. Now the controller should run your application code.


Flash Magic - NON PRODUCTION USE ONLY
File ISP Options Tools Help
Step 1 - Communications
Select...
LPC1768
Select the IC
COM Port: COM 3
Baud Rate: 9600
Interface: None (ISP)
Oscillator (MHz): 12.000000
Step 2 - Erase
Erase block 0 (0x000000-0x000FFF)
Erase block 1 (0x001000-0x001FFF)
Erase block 2 (0x002000-0x002FFF)
Erase block 3 (0x003000-0x003FFF)
Erase block 4 (0x004000-0x004FFF)
Erase block 5 (0x005000-0x005FFF)
Erase all Flash+Code Rd Prot
Erase blocks used by Hex File
Step 3 - Hex File
Hex File: D:\Led_Blinking.hex
Browse...
Modified: Saturday, March 5, 2016, 9:35:22 AM
more info
Browse and Select the Hex file
Step 4 - Options
Verify after programming
Fill unused Flash
Gen block checksums
Execute
Activate Flash Bank
Step 5 - Start!
Start
Finally Hit the Start Button to flash the hex file.
CAN Bus Timing Calculators at:
www.esacademy.com/en/library/calculators.html
0
If DTR and RTS are used then the below options needs to be selected in Options->Advanced Options->Hardware Config
Advanced Options
Communications
Hardware Config
Security
Just In Time Code
Timeouts
Misc
Use DTR and RTS to control RST and ISP pin
Keep RTS asserted while COM Port open
T1: 50 ms T2: 100 ms
Assert DTR and RTS while COM Port open
Disable DTR and RTS completely
Cancel
OK

EXPERIMENTS

Experiment No1: Display Hello word in UART

UART module

UART module and registers. LPC1768 has 4-UARTs numbering 0-3, similarly the pins are also named as RXD0-RXD3 and TXD0-TXD3.As the LPC1768 pins are multiplexed for multiple functionalities, first they have to be configured as UART pins.

Below table shows the multiplexed UARTs pins.

Port Pin	Pin Number	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2	PINSEL_FUNC_3
P0.02	98	GPIO	**TXD0**	ADC0[7]	
P0.03	99	GPIO	**RXD0**	ADC0[6]	
P2_0	48	GPIO	PWM1[1]	**TXD1**	
P2.1	49	GPIO	PWM1[2]	**RXD1**	
P0.10	62	GPIO	**TXD2**	SDA2	MAT3[0]
P0.11	63	GPIO	**RXD2**	SCL2	MAT3[1]
P0.0	82	GPIO	CAN1_Rx	**TXD3**	SDA1
P0.1	85	GPIO	CAN1_Tx	**RXD3**	SCL1

UART Registers

The below table shows the registers associated with LPC1768 UART.

Register Description

RBR	Contains the recently received Data
THR	Contains the data to be transmitted
FCR	FIFO Control Register
LCR	Controls the UART frame formatting(Number of Data Bits, Stop bits)
DLL	Least Significant Byte of the UART baud rate generator value.
DLM	Most Significant Byte of the UART baud rate generator value.

Steps for Configuring UART0

Below are the steps for configuring the UART0.

1. Step1: Configure the GPIO pin for UART0 function using PINSEL register.
2. Step2: Configure the FCR for enabling the FIXO and Reset both the Rx/Tx FIFO.
3. Step3: Configure LCR for 8-data bits, 1 Stop bit, Disable Parity and Enable DLAB.
4. Step4: Get the PCLK from PCLKSELx register 7-6 bits.
5. Step5: Calculate the DLM,DLL values for required baudrate from PCLK.
6. Step6: Update the DLM,DLL with the calculated values.
7. Step6: Finally clear DLAB to disable the access to DLM,DLL.

After this the UART will be ready to Transmit/Receive Data at the specified baudrate.

Main Code:-

```
#include <lpc17xx.h>
#include "stdutils.h"
#define SBIT_WordLenght    0x00u
#define SBIT_DLAB          0x07u
#define SBIT_FIFO          0x00u
#define SBIT_RxFIFO        0x01u
#define SBIT_TxFIFO        0x02u
#define SBIT_RDR           0x00u
#define SBIT_THRE          0x05u

/* Function to initialize the UART0 at specifief baud rate */
void uart_init(uint32_t baudrate)
{
   uint32_t var_UartPclk_u32,var_Pclk_u32,var_RegValue_u32;

   LPC_PINCON->PINSEL0 &= ~0x000000F0;
   LPC_PINCON->PINSEL0 |= 0x00000050;          // Enable TxD0 P0.2 and p0.3

   LPC_UART0->FCR = (1<<SBIT_FIFO) | (1<<SBIT_RxFIFO) | (1<<SBIT_TxFIFO); //
Enable FIFO and reset Rx/Tx FIFO buffers
```

```
  LPC_UART0->LCR = (0x03<<SBIT_WordLenght) | (1<<SBIT_DLAB);          // 8bit data,
1Stop bit, No parity

  /** Baud Rate Calculation :
    PCLKSELx registers contains the PCLK info for all the clock dependent peripherals.
    Bit6,Bit7 contains the Uart Clock(ie.UART_PCLK) information.
    The UART_PCLK and the actual Peripheral Clock(PCLK) is calculated as below.
    (Refer data sheet for more info)

    UART_PCLK    PCLK
      0x00       SystemFreq/4
      0x01       SystemFreq
      0x02       SystemFreq/2
      0x03       SystemFreq/8
   **/

  var_UartPclk_u32 = (LPC_SC->PCLKSEL0 >> 6) & 0x03;

  switch( var_UartPclk_u32 )
  {
  case 0x00:
     var_Pclk_u32 = SystemCoreClock/4;
     break;
  case 0x01:
     var_Pclk_u32 = SystemCoreClock;
     break;
  case 0x02:
     var_Pclk_u32 = SystemCoreClock/2;
     break;
  case 0x03:
     var_Pclk_u32 = SystemCoreClock/8;
```

```
      break;
   }


   var_RegValue_u32 = ( var_Pclk_u32 / (16 * baudrate ));
   LPC_UART0->DLL =  var_RegValue_u32 & 0xFF;
   LPC_UART0->DLM = (var_RegValue_u32 >> 0x08) & 0xFF;

   util_BitClear(LPC_UART0->LCR,(SBIT_DLAB));  // Clear DLAB after setting DLL,DLM
}



/* Function to transmit a char */
void uart_TxChar(char ch)
{
   while(util_IsBitCleared(LPC_UART0->LSR,SBIT_THRE)); // Wait for Previous
transmission
   LPC_UART0->THR=ch;                          // Load the data to be transmitted
}
int main()
{
   char a[]="\n\rHello World";
   int i;
   SystemInit();
   uart_init(9600);  // Initialize the UART0 for 9600 baud rate
    for(i=0;a[i];i++)  //transmit a predefined string
      uart_TxChar(a[i]);
}
```

Experiment No 2:-Speed Control of the DC Motor

DC Motor Control using PWM of LPC1768

In most of the applications controlling the speed of DC motor is essential where the precision and protection are the essence. Here we will use the PWM technique to control the speed of the motor

LPC1768 has one PWM channel with six ports. PWM changes the average output voltage by fast switching. By changing the on time, the output voltage can be 0 to 100%. There are two software parameters that need a little explanation: cycle and offset. Cycle is the length of a PWM duty cycle and offset is the on time of a duty cycle.

SELECTING THE PWM FUNCTION TO GPIO:

The block diagram below shows the PWM pins multiplexed with other GPIO pins. The PWM pin can be enabled by configuring the corresponding PINSEL register to select PWM function. When the PWM function is selected for that pin in the Pin Select register, other Digital signals are disconnected from the PWM input pins.

PWM Channel	Port Pin	Pin Functions	Associated PINSEL Register
PWM1.1	P2.0	0-GPIO, 1- PWM1.1, 2-TXD1, 3-	1,0 bits of PINSEL4
PWM1.2	P2.1	0-GPIO, PWM1.2, 2-RXD1, 3-	3,2 bits of PINSEL4
PWM1.3	P2.2	0-GPIO, PWM1.3, 2-CTS1, 3-	5,4 bits of PINSEL4
PWM1.4	P2.3	0-GPIO, 1- PWM1.4, 2-DCD1, 3-	7,6 bits of PINSEL4
PWM1.5	P2.4	0-GPIO, 1- PWM1.5, 2-DSR1 , 3-	9,8 bits of PINSEL4
PWM1.6	P2.5	0-GPIO, 1- PWM1.6, 2-DTR1 , 3-	11,10 bits of PINSEL4

PWM REGISTERS:

The registers associated with LPC1768 PWM are

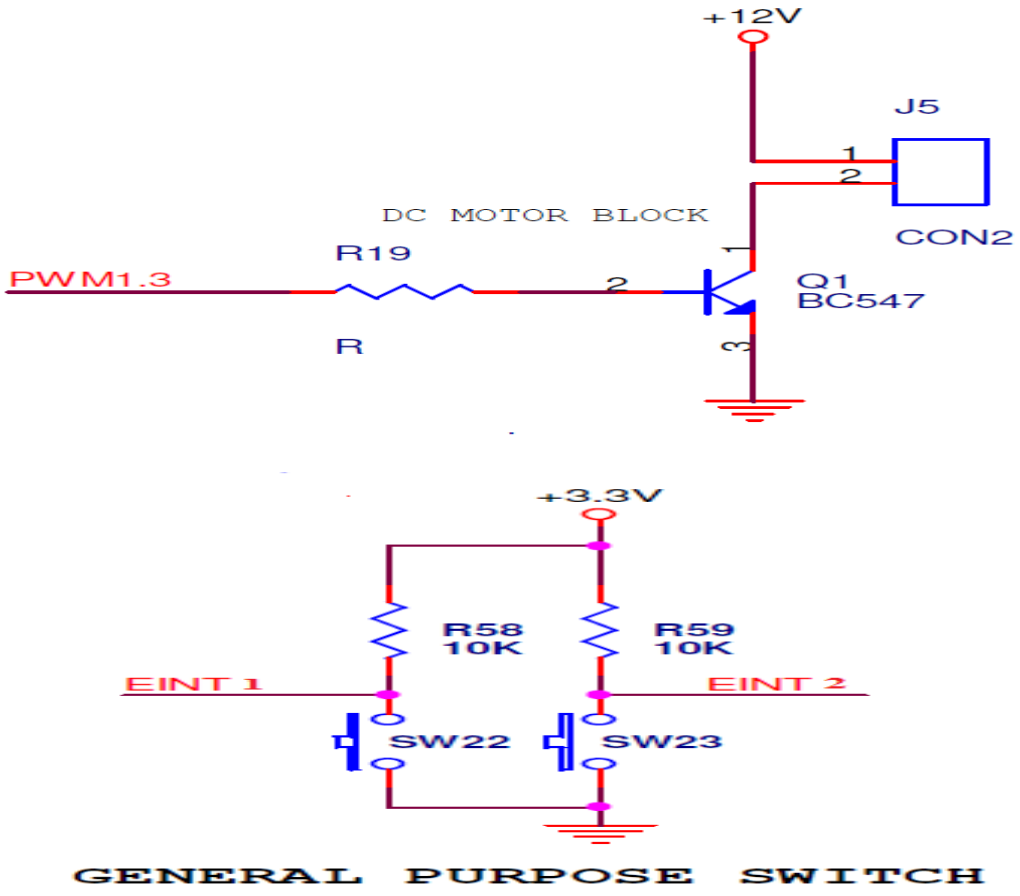
- IR-> Interrupt Register: The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.

- TCR-> Timer Control Register: The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.
- PR- > Prescale Register: The TC is incremented every PR+1 cycles of PCLK.
- MCR-> Match Control Register: The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.
- MR0 – MR6-> Match Register: Each can be enabled in the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt when it matches the TC.
- PCR-> PWM Control Register: Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.
- LCR-> Load Enable Register: Enables use of new PWM match values.

Note: for detailed description of each registers kindly refer PWM waveform section

If you need to control the speed of a DC motor you have a few options. Controlling the speed by controlling either voltage or current is inefficient. Let's understand a bit the speed control of DC motor Using Pulse Width Modulation because controlling how long the voltage is applied with a certain frequency gives you the best control over the motor's speed.

Conventional power supplies tend to generate lots of heat because are working as variable resistors pumping current through external circuits. The pulse width modulation circuits are digital circuits which produce pulsed current. Due to the fact that the pulsed width modulation power supplies works in a state in between on and off, the heat generated is very low compared to the conventional power supplies.



The duty cycle of the circuit can be changed by pressing the switches SW22 and SW23. If we increase the duty cycle(press SW22), the speed of the motor increases and if we decrease the duty cycle(press SW23), the speed of the motor decreases.

Main code:-

```
#include <lpc17xx.h>

void delay_ms(unsigned int ms)// delay routine
{
   unsigned int i,j;
   for(i=0;i<ms;i++)
      for(j=0;j<60000;j++);
}
#define SBIT_CNTEN     0 //counter enable
#define SBIT_PWMEN     2 //pwm 2 block enable
#define SBIT_PWMMR0R   1 //This bit is used to Reset PWMTC
 #define SBIT_PWMENA3   11 //This bit is used to enable/disable the PWM
 #define PWM_3          4 //P2_2 (0-1 Bits of PINSEL4)

int main(void)
{
   int dutyCycle;
   SystemInit();
   /* Cofigure pins(P2_2 ) for PWM mode. */
   LPC_PINCON->PINSEL4 = (1<<PWM_3) ;

   /* Enable Counters,PWM module */
   LPC_PWM1->TCR = (1<<SBIT_CNTEN) | (1<<SBIT_PWMEN);

   LPC_PWM1->PR  =  0x00;              /* No Prescalar */
   LPC_PWM1->MCR = (1<<SBIT_PWMMR0R);  /*Reset on PWMMR0, reset TC if it
matches MR0 */
   LPC_PWM1->MR0 = 100;                /* set PWM cycle(Ton+Toff)=100) */
   /* Enable the PWM output pins for PWM_1-PWM_4(P2_0 - P2_3) */
```

```
   LPC_PWM1->PCR = (1<<SBIT_PWMENA3);
   while(1)
   {
         {
         LPC_PWM1->MR3 = dutyCycle;  /* Increase the dutyCycle from 0-100 */
          delay_ms(5);
      }
          if(!(LPC_GPIO2->FIOPIN & 0x00000800))//if sw 23 pressed
  {
    while(!(LPC_GPIO2->FIOPIN & 0x00000800));

   dutyCycle-=10;       //decrement duty cycle 10%
         if(dutyCycle<0)
   {
     dutyCycle=0;
   }
  }
  else if(!(LPC_GPIO2->FIOPIN & 0x00001000)) //if SW 22 pressed
  {
    while(!(LPC_GPIO2->FIOPIN & 0x00001000));

   dutyCycle+=10;                  //increment duty cycle 10%
         if(dutyCycle>100)
   {
     dutyCycle=99;
   }
  }
   }
}
```

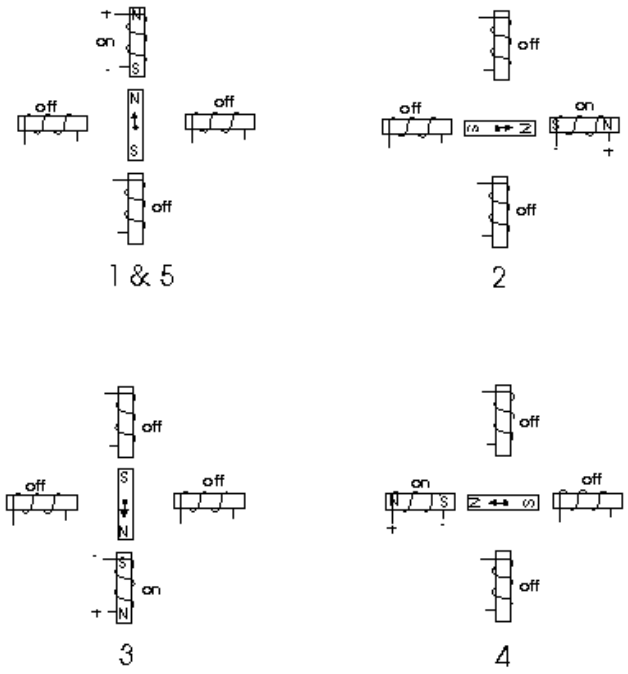
Experiment No 3:-Stepper Motor Interface and rotate clockwise and anticlockwise

Stepper motor

A stepper motor or step motor or stepping motor is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any position sensor for feedback (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed.

How Stepper Motors Work

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Figure 1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.



- Figure 1
- In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 1.8 degrees would move its rotor 1.8 degrees per step, thereby requiring 200 pulses (steps) to complete a full 360 degree rotation.

Here we are using 200 pole stepper motor hence it gives **360degree/200 pole=1.8 degree per step.**

So for example if we need 120 degree rotation then we have to apply approximately 67 pulses to complete 120 degree rotation
120/1.8=66.66==67 steps approximately.

Here one cycle means 4 steps. So if we need 90 degree rotation then 90/1.8=50 steps.
Here one cycle means 4 steps. So 50/4=12.5 =~ 13. So we need 13 cycles to rotate 90 degree.
If we want to run 180 degree then 180/1.8=100. So 100/4=25 cycles would make a stepper motor to rotate 180 degree.

Main code:- Step Angle Rotation

```
#include <lpc17xx.h>

void delay(unsigned int ms)
{
   unsigned int i,j;

   for(i=0;i<ms;i++)
      for(j=0;j<20000;j++); //delay subroutine
```

```
}

/* start the main program */
int main()
{
int cycle;
   SystemInit();                  //Clock and PLL configuration

   LPC_GPIO0->FIODIR   |=  0x00078000;  //Configure the PORT0 pins as OUTPUT;



        for(cycle=0; cycle<13; cycle++)// for loop condition for number of rotation. It gives
approx 90 degree rotation
      {
     LPC_GPIO0->FIOPIN = 0x00008000 ;  // p0.15 pin
        delay(100);
                LPC_GPIO0->FIOPIN = 0x00010000 ;// p0.16 pin
        delay(100);
                LPC_GPIO0->FIOPIN = 0x00020000 ; //      p0.17 pin
        delay(100);
                LPC_GPIO0->FIOPIN = 0x00040000 ;// p0.18 pin
        delay(100);
                }
   }
```

Stepper motor rotation clockwise and anticlockwise.

```
#include "lpc17xx.h"
void  delay();

void delay()
{
```

```
  int i,j;
  for(i=0;i<0xff;i++)
     for(j=0;j<0x400;j++);
}

int main (void)
{
char rotate=0;
          SystemInit();

  LPC_GPIO0->FIODIR   |=  0x00078000;
  while(1)
  {
   if(rotate==1)
   {
    LPC_GPIO0->FIOPIN = 0x00008000 ;
          delay();
                LPC_GPIO0->FIOPIN = 0x00010000 ;
          delay();
                LPC_GPIO0->FIOPIN = 0x00020000 ;
          delay();
                LPC_GPIO0->FIOPIN = 0x00040000 ;
          delay();
   }
   else
   {
    LPC_GPIO0->FIOPIN = 0x00040000 ;
          delay();
                LPC_GPIO0->FIOPIN = 0x00020000 ;
          delay();
                LPC_GPIO0->FIOPIN = 0x00010000 ;
```

```
            delay();
                    LPC_GPIO0->FIOPIN = 0x00008000 ;
            delay();
    }

    if(!(LPC_GPIO2->FIOPIN & 0x00000800))
    {
      while(!(LPC_GPIO2->FIOPIN & 0x00000800));

      rotate=1;
    }
    else if(!(LPC_GPIO2->FIOPIN & 0x00001000))
    {
      while(!(LPC_GPIO2->FIOPIN & 0x00001000));

      rotate=0;
    }
   }
}
```

Experiment No 4:-Display digital output for given analog input using internal ADC

LPC1768 ADC Block

LPC1768 has an inbuilt 12 bit Successive Approximation ADC which is multiplexed among 8 input pins.

The ADC reference voltage is measured across VREFN to VREFP, meaning it can do the conversion within this range. Usually the VREFP is connected to VDD and VREFN is connected to GND.

As LPC1768 works on 3.3 volts, this will be the ADC reference voltage.

Now the $$resolution of ADC = 3.3/(2^{12}) = 3.3/4096 =0.000805 = 0.8mV$$

The below block diagram shows the ADC input pins multiplexed with other GPIO pins.

The ADC pin can be enabled by configuring the corresponding PINSEL register to select ADC function.

When the ADC function is selected for that pin in the Pin Select register, other Digital signals are disconnected from the ADC input pins.

Adc Channel	Port Pin	Pin Functions	Associated PINSEL Register
AD0	P0.23	0-GPIO, 1-AD0[0], 2-I2SRX_CLK, 3-CAP3[0]	14,15 bits of PINSEL1
AD1	P0.24	0-GPIO, 1-AD0[1], 2-I2SRX_WS, 3-CAP3[1]	16,17 bits of PINSEL1
AD2	P0.25	0-GPIO, 1-AD0[2], 2-I2SRX_SDA, 3-TXD3	18,19 bits of PINSEL1
AD3	P0.26	0-GPIO, 1-AD0[3], 2-AOUT, 3-RXD3	20,21 bits of PINSEL1
AD4	P1.30	0-GPIO, 1-VBUS, 2- , 3-AD0[4]	28,29 bits of PINSEL3
AD5	P1.31	0-GPIO, 1-SCK1, 2- , 3-AD0[5]	30,31 bits of PINSEL3
AD6	P0.3	0-GPIO, 1-RXD0, 2-AD0[6], 3-	6,7 bits of PINSEL0
AD7	P0.2	0-GPIO, 1-TXD0, 2-AD0[7], 3-	4,5 bits of PINSEL0

ADC Registers

The below table shows the registers associated with LPC1768 ADC.

We are going to focus only on ADCR and ADGDR as these are sufficient for simple A/D conversion.

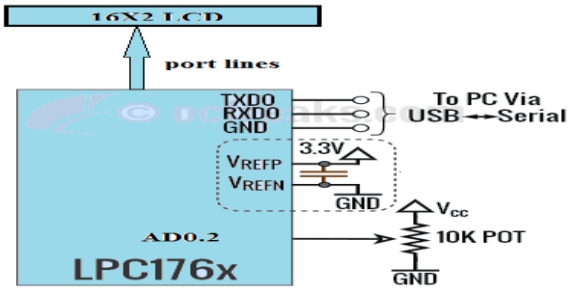
However once you are familiar with LPC1768 ADC, you can explore the other features and the associated registers.

Register	Description
ADCR	A/D COntrol Register: Used for Configuring the ADC
ADGDR	A/D Global Data Register: This register contains the ADC's DONE bit and the result of the most recent A/D conversion
ADINTEN	A/D Interrupt Enable Register
ADDR0 - ADDR7	A/D Channel Data Register: Contains the recent ADC value for respective channel
ADSTAT	A/D Status Register: Contains DONE & OVERRUN flag for all the ADC channels

Steps for Configuring ADC

Below are the steps for configuring the LPC1768 ADC.

1. Configure the GPIO pin for ADC function using PINSEL register.
2. Enable the CLock to ADC module.
3. Deselect all the channels and Power on the internal ADC module by setting ADCR.PDN bit.
4. Select the Particular channel for A/D conversion by setting the corresponding bits in ADCR.SEL
5. Set the ADCR.START bit for starting the A/D conversion for selected channel.
6. Wait for the conversion to complete, ADGR.DONE bit will be set once conversion is over.
7. Read the 12-bit A/D value from ADGR.RESULT.
8. Use it for further processing or just display on LCD.



Main ADC code

```
#include "lpc17xx.h"
#include "lcd.h"
#define VREF       3.3 //Reference Voltage at VREFP pin, given VREFN = 0V(GND)
#define ADC_CLK_EN (1<<12)
#define SEL_AD0_2  (1<<2) //Select Channel AD0.2
#define CLKDIV     1 //ADC clock-divider (ADC_CLOCK=PCLK/CLKDIV+1) = 12.5Mhz @
25Mhz PCLK
#define PWRUP      (1<<21) //setting it to 0 will power it down
#define START_CNV  (1<<24) //001 for starting the conversion immediately
#define ADC_DONE   (1U<<31) //define it as unsigned value or compiler will throw #61-D
warning
#define ADCR_SETUP_SCM ((CLKDIV<<8) | PWRUP)
//////////   Init ADC0 CH2  /////////////////
Init_ADC()
{
  // Convert Port pin 0.25 to function as AD0.2
   LPC_SC->PCONP |= ADC_CLK_EN; //Enable ADC clock
         LPC_ADC->ADCR =  ADCR_SETUP_SCM | SEL_AD0_2;
         LPC_PINCON->PINSEL1 |= (1<<18) ; //select AD0.2 for P0.25
}
```

```
//////////   READ ADC0 CH:2  /////////////////
unsigned int Read_ADC()
{
  unsigned int i=0;
LPC_ADC->ADCR |= START_CNV; //Start new Conversion
                while((LPC_ADC->ADDR2 & ADC_DONE) == 0); //Wait untill conversion is
finished
                i = (LPC_ADC->ADDR2>>4) & 0xFFF; //12 bit Mask to extract result

}
//////////   DISPLAY ADC VALUE /////////////////
Display_ADC()
{
 unsigned int adc_value = 0;
 char buf[4] = {5};
 float voltage = 0.0;
        adc_value = Read_ADC();
        sprintf((char *)buf, "%3d", adc_value);         // display 3 decima place
        lcd_putstring16(0,"ADC VAL = 000    "); //1st line display
        lcd_putstring16(1,"Voltage    00 V"); //2nd line display
        lcd_gotoxy(0,10);
        lcd_putstring(buf);
        voltage = (adc_value * 3.3) / 4095 ;
        lcd_gotoxy(1,8);
        sprintf(buf, "%3.2f", voltage);
        lcd_putstring(buf);
}
//////////   MAIN /////////////////
int main (void)
{
  init_lcd();
```

```
 Init_ADC();
 lcd_putstring16(0,"** HMSIT **");
 lcd_putstring16(1,"** TUMKUR **");
 delay(60000);
 delay(60000);
 delay(60000);
 lcd_putstring16(0,"ADC Value.. ");
 lcd_putstring16(1,"voltage.....");
 while(1)
 {
        Display_ADC();
        delay(100000);
 }
}
```

No	HEX Value	COMMAND TO LCD
1	0x01	Clear Display Screen
2	0x30	Function Set: 8-bit, 1 Line, 5x7 Dots
3	0x38	Function Set: 8-bit, 2 Line, 5x7 Dots
4	0x20	Function Set: 4-bit, 1 Line, 5x7 Dots
5	0x28	Function Set: 4-bit, 2 Line, 5x7 Dots
6	0x06	Entry Mode
7	0x08	Display off, Cursor off
8	0x0E	Display on, Cursor on
9	0x0C	Display on, Cursor off
10	0x0F	Display on, Cursor blinking
11	0x18	Shift entire display left
12	0x1C	Shift entire display right
13	0x10	Move cursor left by one character
14	0x14	Move cursor right by one character
15	0x80	Force cursor to beginning of 1st row
16	0xC0	Force cursor to beginning of 2nd row

LCD Code:-

```
#include "lpc17xx.h"
#include "lcd.h"
void Lcd_CmdWrite(unsigned char cmd);
void Lcd_DataWrite(unsigned char dat);
#define LCDRS          9
#define LCDRW         10
#define LCDEN         11
#define LCD_D4  19
#define LCD_D5  20
#define LCD_D6  21
#define LCD_D7  22
#define LcdData       LPC_GPIO0->FIOPIN
#define LcdControl    LPC_GPIO0->FIOPIN
#define LcdDataDirn   LPC_GPIO0->FIODIR
#define LcdCtrlDirn   LPC_GPIO0->FIODIR
#define LCD_ctrlMask   ((1<<LCDRS)|(1<<LCDRW)|(1<<LCDEN))
#define LCD_dataMask   ((1<<LCD_D4)|(1<<LCD_D5)|(1<<LCD_D6)|(1<<LCD_D7))
void delay(unsigned int count)
{
int j=0, i=0;
for (j=0;j<count;j++)
for (i=0;i<50;i++);
}
  void sendNibble(char nibble)
{
    LcdData&=~(LCD_dataMask);                  // Clear previous data
    LcdData|= (((nibble >>0x00) & 0x01) << LCD_D4);
    LcdData|= (((nibble >>0x01) & 0x01) << LCD_D5);
    LcdData|= (((nibble >>0x02) & 0x01) << LCD_D6);
    LcdData|= (((nibble >>0x03) & 0x01) << LCD_D7);
```

```
}
 void Lcd_CmdWrite(unsigned char cmd)
{
   sendNibble((cmd >> 0x04) & 0x0F);  //Send higher nibble
   LcdControl &= ~(1<<LCDRS); // Send LOW pulse on RS pin for selecting Command register
   LcdControl &= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
   LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
   delay(100);
   LcdControl &= ~(1<<LCDEN);

   delay(10000);
   sendNibble(cmd & 0x0F);           //Send Lower nibble
   LcdControl &= ~(1<<LCDRS); // Send LOW pulse on RS pin for selecting Command register
   LcdControl &= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
   LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
   delay(100);
   LcdControl &= ~(1<<LCDEN);

   delay(1000);
}

void Lcd_DataWrite(unsigned char dat)
{
   sendNibble((dat >> 0x04) & 0x0F);  //Send higher nibble
   LcdControl |= (1<<LCDRS);  // Send HIGH pulse on RS pin for selecting data register
   LcdControl &= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
   LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
   delay(100);
   LcdControl &= ~(1<<LCDEN);
   delay(1000);
   sendNibble(dat & 0x0F);           //Send Lower nibble
```

```
    LcdControl |= (1<<LCDRS);  // Send HIGH pulse on RS pin for selecting data register
    LcdControl &= ~(1<<LCDRW); // Send LOW pulse on RW pin for Write operation
    LcdControl |= (1<<LCDEN);  // Generate a High-to-low pulse on EN pin
    delay(100);
    LcdControl &= ~(1<<LCDEN);
    delay(1000);
}
void lcd_clear( void)
{
  Lcd_CmdWrite( 0x01 );
}
int lcd_gotoxy( unsigned char x, unsigned char y)
{
  unsigned char retval = TRUE;

  if( (x > 1) && (y > 15) )
   {
     retval = FALSE;
   }
  else
   {
     if( x == 0 ) Lcd_CmdWrite( 0x80 + y );    //Move the cursor to beginning of the first line
           else if( x==1 ) Lcd_CmdWrite( 0xC0 + y );// Move the cursor to beginning of the second
line
   }
   return retval;
}
void lcd_putchar( unsigned char c )
{
  Lcd_DataWrite( c );
}
```

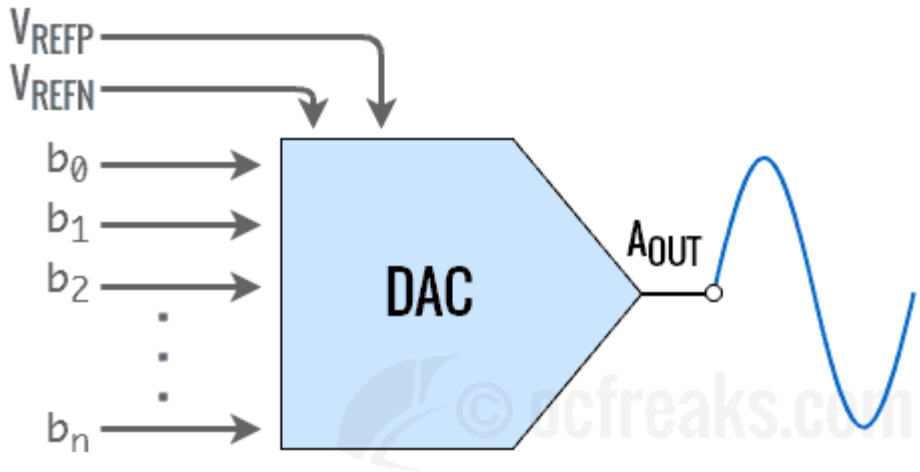
```
void lcd_putstring( char *string )
{
   while(*string != '\0')
  {
    lcd_putchar( *string );
    string++;
  }
}
void lcd_putstring16( unsigned char line, char *string )
{
  unsigned char len = 16;

  lcd_gotoxy( line, 0 );
  while(*string != '\0' && len--)
  {
    lcd_putchar( *string );
    string++;
  }
}
  void init_lcd( void )
  {
   LcdDataDirn |= LCD_dataMask;  // Configure all the LCD pins as output
   LcdCtrlDirn |= LCD_ctrlMask;
    Lcd_CmdWrite(0x03);//
   delay(2000);
  Lcd_CmdWrite(0x03);//
   delay(1000);
  Lcd_CmdWrite(0x03);//
   delay(100);
  Lcd_CmdWrite(0x2);// Initialize LCD in 4-bit mode
  Lcd_CmdWrite(0x28);// enable 5x7 mode for chars
```

```
 Lcd_CmdWrite(0x0e);// Display OFF, Cursor ON
 Lcd_CmdWrite(0x06);//Entry mode
Lcd_CmdWrite(0x01);// Clear Display
 delay(1);
        }
```

Experiment No 5:-Interface DAC and generate Triangular and Square Waveform

LPC1768 DAC Programming Tutorial

In this article, we will go through a discussion on ARM Cortex-M3 LPC1768 DAC programming tutorial. As you might be knowing, DAC stands for Digital to Analog Conversion. The DAC block in ARM Cortex-M3 LPC176x microcontroller is one of the simplest to program and also supports DMA(direct memory access). This tutorial is also applicable for LPC1769 MCU. Basically we assign a digital value to a register and the DAC block outputs it equivalent analog signal as shown below:



LPC1768/LPC1769 DAC Block

ARM Cortex-M3 LPC176x MCUs incorporate a 10 bit DAC and provide buffered analog output. As per the datasheet, it is implemented as a string DAC which is the most simplest form of DAC consisting of 2^N resistors in series where N = no. of bits which simply forms a Kelvin-Varley Divider. LPC176x DAC has only 1 output pin, referred to as **AOUT**. The Analog voltage at the output of this pin is given as:

$$V_{AOUT} = \frac{VALUE * (V_{REFP}-V_{REFN})}{1024} + V_{REFN}$$

When we have V_{REFN} = 0, the equation boils down to:

$$V_{AOUT} = \frac{VALUE * V_{REFP}}{1024}$$

Where VALUE is the 10-bit digital value which is to be converted into its Analog counterpart and V_{REF} is the input reference voltage.

Pins relating to LPC1768 DAC block:

Pin	Description
AOUT (P0.26)	Analog Output pin. Provides the converted Analog signal which is referenced to V_{SSA} i.e. the Analog GND. Set Bits[21:20] in PINSEL1 register to [10] to enable this function.
V_{REFP}, V_{REFN}	These are reference voltage input pins used for both ADC and DAC. V_{REFP} is positive reference voltage and V_{REFN} is negative reference voltage pin. In example shown below we will use V_{REFN}=0V(GND).
V_{DDA}, V_{SSA}	V_{DDA} is Analog Power pin and V_{SSA} is Ground pin used to power the ADC module. These are generally same as V_{CC} and GND but with additional filtering to reduce noise.

DAC Registers in ARM Cortex-M3 LPC176x

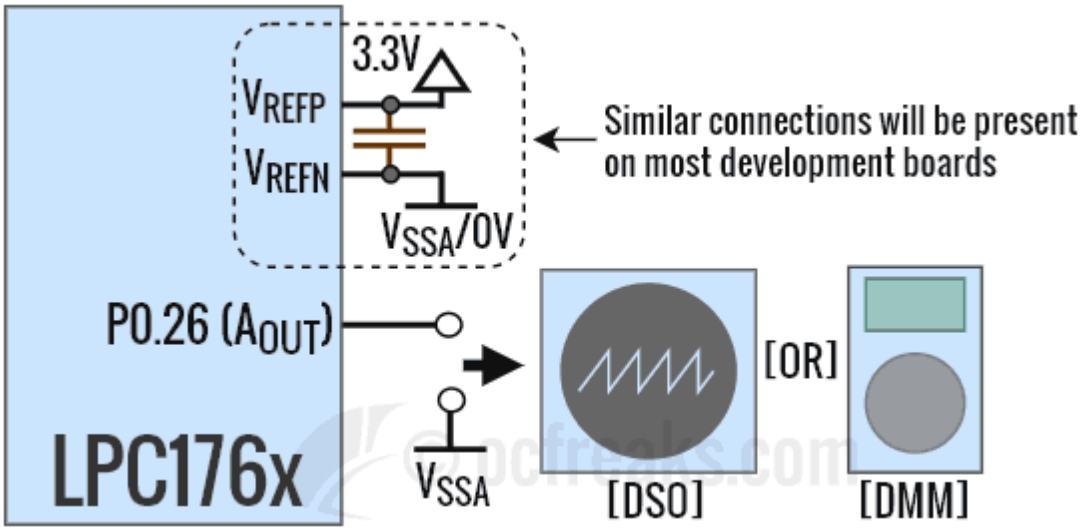
The DAC module in ARM LPC1768/LPC1769 has 3 registers viz. DACR, DACCTRL, DACCNTVAL. In this tutorial we will only go through to DACR register since the other two are related with **DMA** operation, explaining which is not in the scope of this tutorial. Just note that DMA is used to update new values to DACR from memory without the intervention of CPU. This is particularly useful when generation different types of waveforms using DAC. We will cover this in another tutorial.

Also note that the DAC doesn't have a power control it in PCONP register. Simply select the AOUT alternate function for pin P0.26 using PINSEL1 register to enable DAC ouput.

The DACR register in LPC1768

The field containing bits [15:6] is used to feed a digital value which needs to be converted and bit 16 is used to select settling time. The bit significance is as shown below:

1. **Bit[5:0]:** Reserved.
2. **Bit[15:6] – VALUE:** After a new VALUE is written to this field, given settling time selected using BIAS has elapsed, we get the converted Analog voltage at the output. The formula for analog voltage at AOUT pin is as shown above.
3. **Bit[16] – BIAS:** Setting this bit to 0 selects settling time of 1us max with max current consumption of 700uA at max 1Mhz update rate. Setting it to 1 will select settling time of 2.5us but with reduce max current consumption of 300uA at max 400Khz update rate.
4. **Bits[31:17]**: Reserved



Triangle wave generation:-

```
#include "LPC17xx.h"
uint32_t val;
int main()
{
 SystemInit();
 LPC_PINCON->PINSEL1 |= 0x02<<20; //p0.26 pinsel bits 20 and 21
 while(1)
 {
   while(1)
   {
            val++;// increment values by 6
     LPC_DAC->DACR=(val<<6);//send values to dac
     if(val>=0x3ff)// if value exceeds 1024
     {
       break;
     }
   }
   while(1)
   {
            val--; // decrement value by 6
     LPC_DAC->DACR=(val<<6);// send value to dac
     if(val<=0x000)// if value come down by 0
     {
      break;
     }
   }
 }
}
```

Square Wave generation:-

```
#include "LPC17xx.h"
void delay(unsigned int ms)// delay subroutine
{
   unsigned int i,j;

   for(i=0;i<ms;i++)
      for(j=0;j<2000;j++);
}
int main()
{
 SystemInit();
 LPC_PINCON->PINSEL1 |= 0x02<<20;//p0.26 pinsel bits 20 and 21

 while(1)
 {
     LPC_DAC->DACR=0xffff;// send maximum values to dac

    delay(20); // delay

     LPC_DAC->DACR=0x0000;// send min values to dac

     delay(20);
 }

}
```

Experiment No 6:-Interface 4X4 Matrix Keypad and display in LCD

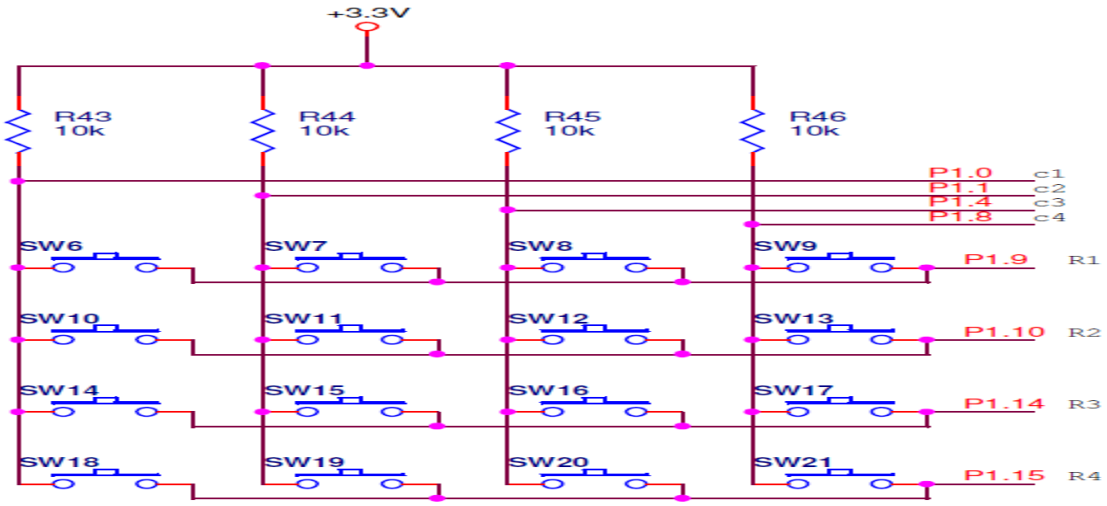
Introduction 4x4 matrix keypad interface:-

There are various methods to provide an input for the GPIO pins of a microcontroller which can be software controlled or hardware controlled. We have seen a hardware controlled method by taking an input from a switch or a press button. Push Button Interfacing with LPC1768
The keypad is another external input device which is hardware controlled, but is used for a specific purpose.
In this tutorial we take a look at how a 4×4 matrix keypad is interfaced with the LPC1768 microcontroller. The input taken from the matrix keypad will be displayed on a 16x2 lcd display

Basic Configuration

NOTE: Please read this section carefully as it is important to understand the hardware connection before writing a software for this application.

- As we are using a 4×4 matrix keypad, a total of 8 input-output pins of the micro-controller will be required for interfacing.
- One half of the 8 pins will be hardware controlled and the other half will be software controlled.



//

```
// Matrix Keypad Scanning Routine
//
// COL1 COL2 COL3 COL4
// 0    1    2    3    ROW 1
// 4    5    6    7    ROW 2
// 8    9    A    B    ROW 3
// C    D    E    F    ROW 4
////////////////////////////////////////////
```

Main code:-

```
#include "lpc17xx.h"
#include "lcd.h"
#define COL1            0
#define COL2            1
#define COL3            4
#define COL4            8
#define ROW1                    9
#define ROW2                    10
#define ROW3                    14
#define ROW4                    15
#define COLMASK                         ((1<<COL1) |(1<< COL2) |(1<< COL3) |(1<< COL4))
#define ROWMASK                         ((1<<ROW1) |(1<< ROW2) |(1<< ROW3) |(1<<
ROW4))
#define KEY_CTRL_DIR        LPC_GPIO1->FIODIR
#define KEY_CTRL_SET       LPC_GPIO1->FIOSET
#define KEY_CTRL_CLR       LPC_GPIO1->FIOCLR
#define KEY_CTRL_PIN       LPC_GPIO1->FIOPIN

/////////////// COLUMN WRITE //////////////////////
void col_write( unsigned char data )
{
  unsigned int temp=0;
```

```
   temp=(data) & COLMASK;
  KEY_CTRL_CLR |= COLMASK;
  KEY_CTRL_SET |= temp;
}
//////////////////////////////// MAIN  ////////////////////////////////////
int main (void)
{
unsigned char key, i;
unsigned char rval[] = {0x77,0x07,0x0d};
unsigned char keyPadMatrix[] =
{
   '4','8','B','F',
   '3','7','A','E',
   '2','6','0','D',
   '1','5','9','C'
};
 SystemInit();
  init_lcd();
  KEY_CTRL_DIR |= COLMASK;    //Set COLs as Outputs
  KEY_CTRL_DIR &= ~(ROWMASK); //  Set ROW lines as Inputs
  lcd_putstring16(0,"Press HEX Keys..");// 1st line display
  lcd_putstring16(1,"Key Pressed =    ");// 2nd line display
while (1)
 {
   key = 0;
   for( i = 0; i < 4; i++ )
   {
      // turn on COL output one by one
                  col_write(rval[i]);
        // read rows - break when key press detected
      if (!(KEY_CTRL_PIN & (1<<ROW1)))
```

```
            break;
        key++;
        if (!(KEY_CTRL_PIN & (1<<ROW2)))
            break;
        key++;
        if (!(KEY_CTRL_PIN & (1<<ROW3)))
            break;
        key++;
                    if (!(KEY_CTRL_PIN & (1<<ROW4)))
            break;
        key++;
    }

            if (key == 0x10)
                    lcd_putstring16(1,"Key Pressed =   ");
            else
                    {
                            lcd_gotoxy(1,14);
                            lcd_putchar(keyPadMatrix[key]);
                    }
 }
}
```

Experiment No 7:-Generate PWM waveform and vary its duty cycle

LPC1768 PWM Module

LPC1768 has 6 PWM output pins which can be used as 6-Single edged or 3-Double edged. There as seven match registers to support these 6 PWM output signals. Below block diagram shows the PWM pins and the associated Match(Duty Cycle) registers.

PWM Channel	Port Pin	Pin Functions	Associated PINSEL Register	Corresponding Match Register
PWM_1	P2.0	0-GPIO, 1-**PWM1[1]**, 2-TXD1, 3-	0,1 bits of PINSEL4	MR1
PWM_2	P2.1	0-GPIO, 1-**PWM1[2]**, 2-RXD1, 3-	2,3 bits of PINSEL4	MR2
PWM_3	P2.2	0-GPIO, 1-**PWM1[3]**, 2-CTS1, 3-TRACEDATA[3]	4,5 bits of PINSEL4	MR3
PWM_4	P2.3	0-GPIO, 1-**PWM1[4]**, 2-DCD1, 3-TRACEDATA[2]	6,7 bits of PINSEL4	MR4
PWM_5	P2.4	0-GPIO, 1-**PWM1[5]**, 2-DSR1, 3-TRACEDATA[1]	8,9 bits of PINSEL4	MR5
PWM_6	P2.5	0-GPIO, 1-**PWM1[6]**, 2-DTR1, 3-TRACEDATA[0]	10,11 bits of PINSEL4	MR6

LPC1768 PWM Registers

The below table shows the registers associated with LPC1768 PWM.

Register	Description
IR	Interrupt Register: The IR can be read to identify which of eight possible interrupt sources are pending. Writing Logic-1 will clear the corresponding interrupt.
TCR	Timer Control Register: The TCR is used to control the Timer Counter functions(enable/disable/reset).
TC	Timer Counter: The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is

	controlled through the TCR.
PR	Prescalar Register: This is used to specify the Prescalar value for incrementing the TC.
PC	Prescale Counter: The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.
MCR	Match Control Register: The MCR is used to control the reseting of TC and generating of interrupt whenever a Match occurs.
MR0	Match Register: This register hold the max cycle Time(Ton+Toff).
MR1-MR6	Match Registers: These registers holds the Match value(PWM Duty) for corresponding PWM channels(PWM1-PWM6).
PCR	PWM Control Register: PWM Control Register. Enables PWM outputs and selects PWM channel types as either single edge or double edge controlled.
LER	Load Enable Register: Enables use of new PWM values once the match occurs.

Steps to Configure PWM

1. Configure the GPIO pins for PWM operation in respective PINSEL register.
2. Configure TCR to enable the Counter for incrementing the TC, and Enable the PWM block.
3. Set the required pre-scalar value in PR. In our case it will be zero.
4. Configure MCR to reset the TC whenever it matches MR0.
5. Update the Cycle time in MR0. In our case it will be 100.
6. Load the Duty cycles for required PWMx channels in respective match registers MRx(x: 1-6).
7. Enable the bits in LER register to load and latch the new match values.
8. Enable the required pwm channels in PCR register.

Main code:-

```
#include <lpc17xx.h>

void delay_ms(unsigned int ms)
```

```
{
   unsigned int i,j;
   for(i=0;i<ms;i++)
      for(j=0;j<60000;j++);
}



#define SBIT_CNTEN     0
#define SBIT_PWMEN     2

#define SBIT_PWMMR0R   1

#define SBIT_PWMENA1   9

#define PWM_1          0 //P2_0 (0-1 Bits of PINSEL4)

int main(void)
{
   int dutyCycle;
   SystemInit();
   /* Cofigure pins(P2_0 ) for PWM mode. */
   LPC_PINCON->PINSEL4 = (1<<PWM_1) ;

   /* Enable Counters,PWM module */
   LPC_PWM1->TCR = (1<<SBIT_CNTEN) | (1<<SBIT_PWMEN);

   LPC_PWM1->PR  =  0x00;              /* No Prescalar */
   LPC_PWM1->MCR = (1<<SBIT_PWMMR0R);  /*Reset on PWMMR0, reset TC if it
matches MR0 */

   LPC_PWM1->MR0 = 100;                /* set PWM cycle(Ton+Toff)=100) */
```

```
    /* Enable the PWM output pins for PWM_1-PWM_4(P2_0 - P2_3) */
    LPC_PWM1->PCR = (1<<SBIT_PWMENA1);

    while(1)
    {
        for(dutyCycle=0; dutyCycle<100; dutyCycle++)
        {
            LPC_PWM1->MR1 = dutyCycle;  /* Increase the dutyCycle from 0-100 */

            delay_ms(5);
        }

        for(dutyCycle=100; dutyCycle>0; dutyCycle--)
        {
            LPC_PWM1->MR1 = dutyCycle;  /* Decrease the dutyCycle from 100-0 */

            delay_ms(5);
        }
    }
}
```

Experiment No 8:-Using external interrupt switches toggle the led's

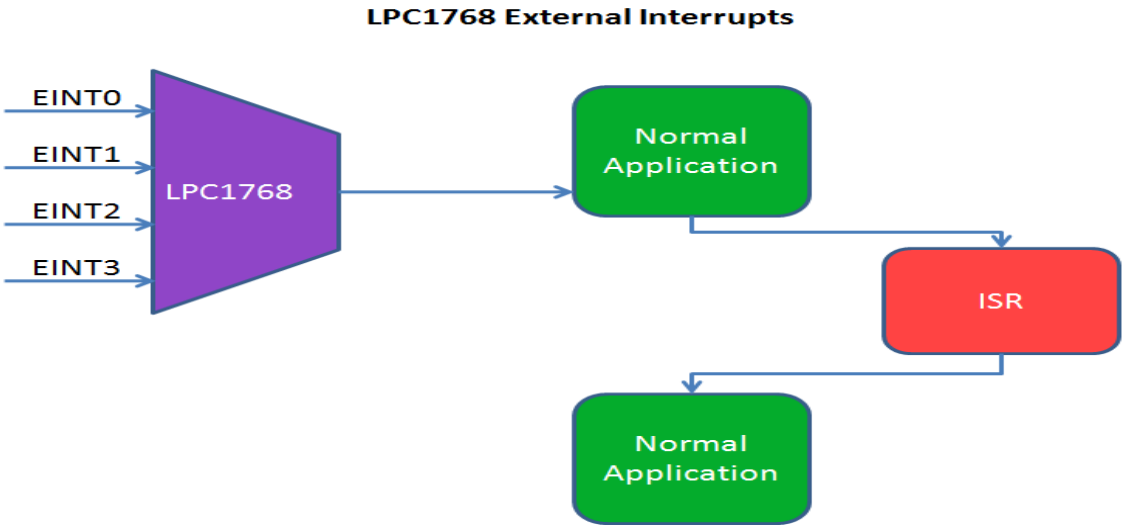
EINTx Pins

LPC1768 has four external interrupts EINT0-EINT3.

As LPC1768 pins are multi functional, these four interrupts are available on multiple pins.

Below table shows mapping of EINTx pins.

Port Pin	PINSEL_FUNC_0	PINSEL_FUNC_1	PINSEL_FUNC_2	PINSEL_FUNC_3
P2.10	GPIO	**EINT0**	NMI	
P2.11	GPIO	**EINT1**	I2STX_CLK	
P2_12	GPIO	**EINT2**	I2STX_WS	
P2.13	GPIO	**EINT3**	I2STX_SDA	



EINT Registers

Below table shows the registers associated with LPC1768 external interrupts.

Register	Description
PINSELx	To configure the pins as External Interrupts
EXTINT	External Interrupt Flag Register contains interrupt flags for EINT0,EINT1, EINT2 & EINT3.
EXTMODE	External Interrupt Mode register(Level/Edge Triggered)

EXTPOLAR External Interrupt Polarity(Falling/Rising Edge, Active Low/High)

EXTINT

31:4 3 2 1 0

RESERVED EINT3 EINT2 EINT1 EINT0

EINTx: Bits will be set whenever the interrupt is detected on the particular interrupt pin.

If the interrupts are enabled then the control goes to ISR.

Writing one to specific bit will clear the corresponding interrupt.

EXTMODE

31:4 3 2 1 0

RESERVED EXTMODE3 EXTMODE2 EXTMODE1 EXTMODE0

EXTMODEx: This bits is used to select whether the EINTx pin is level or edge Triggered

0: EINTx is Level Triggered.

1: EINTx is Edge Triggered.

EXTPOLA

R

31:4 3 2 1 0

RESERVED EXTPOLAR3 EXTPOLAR2 EXTPOLAR1 EXTPOLAR0

EXTPOLARx: This bits is used to select polarity(LOW/HIGH, FALLING/RISING) of the EINTx interrupt depending on the EXTMODE register.

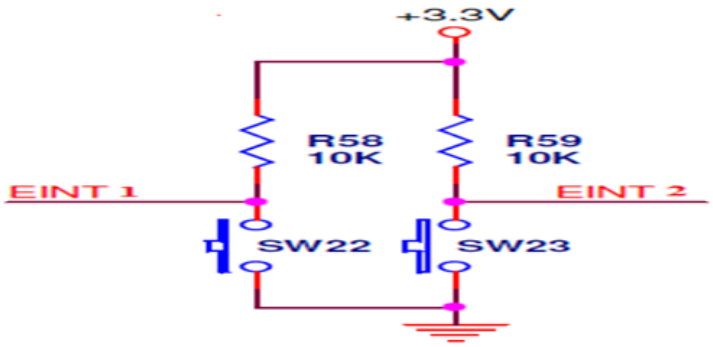
0: EINTx is Active Low or Falling Edge (depending on EXTMODEx).

1: EINTx is Active High or Rising Edge (depending on EXTMODEx).

Steps to Configure Interrupts

1. Configure the pins as external interrupts in PINSELx register.
2. Clear any pending interrupts in EXTINT.

3. Configure the EINTx as Edge/Level triggered in EXTMODE register.
4. Select the polarity(Falling/Rising Edge, Active Low/High) of the interrupt in EXTPOLAR register.
5. Finally enable the interrupts by calling NVIC_EnableIRQ() with IRQ number.



Main code:-

```
#include <lpc17xx.h>

#define PINSEL_EINT1    22 // interrupt 1
#define PINSEL_EINT2    24 // interrupt 2

#define LED1            25  // led at p1.25
#define LED2            26  // led at p1.26

#define SBIT_EINT1      1  //extint bit 1
#define SBIT_EINT2      2  //extint bit 2

#define SBIT_EXTMODE1   1        //extint mode bit 1
#define SBIT_EXTMODE2   2        //extint mode bit 2

#define SBIT_EXTPOLAR1  1  //extint polarity mode bit 1
#define SBIT_EXTPOLAR2  2  //extint polarity mode bit 2
```

```
void EINT1_IRQHandler(void)
{
   LPC_SC->EXTINT = (1<<SBIT_EINT1);  /* Clear Interrupt Flag */
   LPC_GPIO1->FIOPIN ^= (1<< LED1);   /* Toggle the LED1 everytime INTR1 is generated
*/
}


void EINT2_IRQHandler(void)
{
   LPC_SC->EXTINT = (1<<SBIT_EINT2);  /* Clear Interrupt Flag */
   LPC_GPIO1->FIOPIN ^= (1<< LED2);   /* Toggle the LED2 everytime INTR2 is generated
*/
}

int main()
{
   SystemInit();

   LPC_SC->EXTINT      = (1<<SBIT_EINT1)    | (1<<SBIT_EINT2);         /* Clear Pending
interrupts */
   LPC_PINCON->PINSEL4 = (1<<PINSEL_EINT1)  | (1<<PINSEL_EINT2);   /* Configure
P2_11,P2_12 as EINT1/2 */
   LPC_SC->EXTMODE     = (1<<SBIT_EXTMODE1) | (1<<SBIT_EXTMODE2);  /*
Configure EINTx as Edge Triggered*/
   LPC_SC->EXTPOLAR    = (1<<SBIT_EXTPOLAR1)| (1<<SBIT_EXTPOLAR2); /*
Configure EINTx as Falling Edge */
```

```
   LPC_GPIO1->FIODIR   = (1<<LED1)  | (1<<LED2);                /* Configure LED pins as
OUTPUT */
   LPC_GPIO1->FIOPIN   =  0x00;

   NVIC_EnableIRQ(EINT1_IRQn);    /* Enable the EINT1,EINT2 interrupts */
   NVIC_EnableIRQ(EINT2_IRQn);

   while(1)
   {
    // Do nothing
   }
}
```

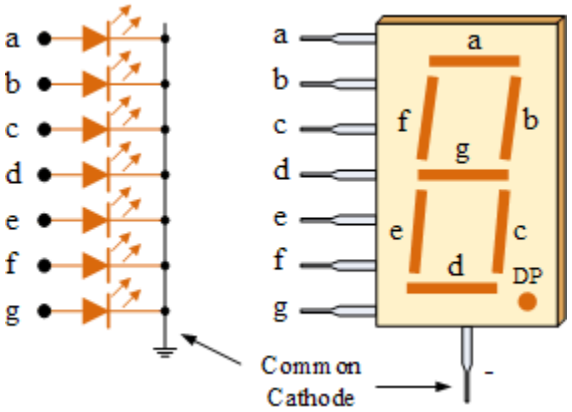
Experiment No 9:-Display 0-F in 7 segment display

The displays common pin is generally used to identify which type of 7-segment display it is. As each LED has two connecting pins, one called the "Anode" and the other called the "Cathode", there are therefore two types of LED 7-segment display called: **Common Cathode** (CC) and **Common Anode** (CA).

The difference between the two displays, as their name suggests, is that the common cathode has all the cathodes of the 7-segments connected directly together and the common anode has all the anodes of the 7-segments connected together and is illuminated as follows.

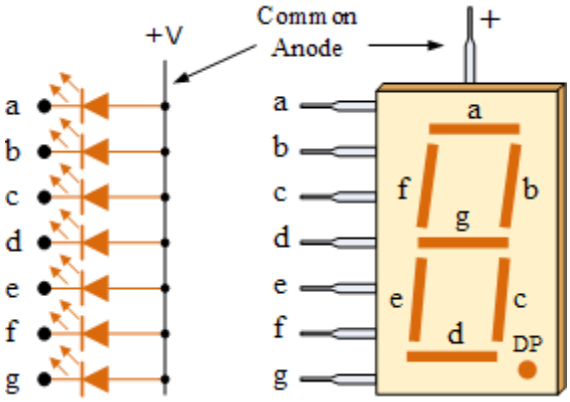
1. The Common Cathode (CC) – In the common cathode display, all the cathode connections of the LED segments are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", or logic "1" signal via a current limiting resistor to forward bias the individual Anode terminals (a-g).

Common Cathode 7-segment Display



2. The Common Anode (CA) – In the common anode display, all the anode connections of the LED segments are joined together to logic "1". The individual segments are illuminated by applying a ground, logic "0" or "LOW" signal via a suitable current limiting resistor to the Cathode of the particular segment (a-g).

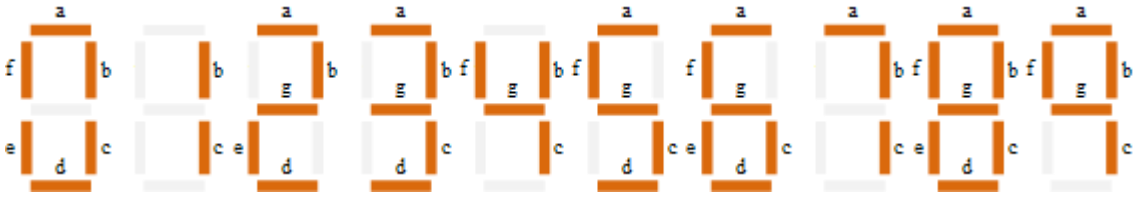
Common Anode 7-segment Display



In general, common anode displays are more popular as many logic circuits can sink more current than they can source. Also note that a common cathode display is not a direct replacement in a circuit for a common anode display and vice versa, as it is the same as connecting the LEDs in reverse, and hence light emission will not take place.

Depending upon the decimal digit to be displayed, the particular set of LEDs is forward biased. For instance, to display the numerical digit 0, we will need to light up six of the LED segments corresponding to a, b, c, d, e and f. Thus the various digits from 0 through 9 can be displayed using a 7-segment display as shown.

7-Segment Display Segments for all Numbers.



Then for a 7-segment display, we can produce a truth table giving the individual segments that need to be illuminated in order to produce the required decimal digit from 0 through 9 as shown below.

7-segment Display Truth Table

Decimal Digit	Individual Segments Illuminated						
	a	b	c	d	e	f	g
0	×	×	×	×	×	×	

1		×	×				
2	×	×		×	×		×
3	×	×	×	×			×
4		×	×			×	×
5	×		×	×		×	×
6	×		×	×	×	×	×
7	×	×	×				
8	×	×	×	×	×	×	×
9	×	×	×			×	×

Driving a 7-segment Display

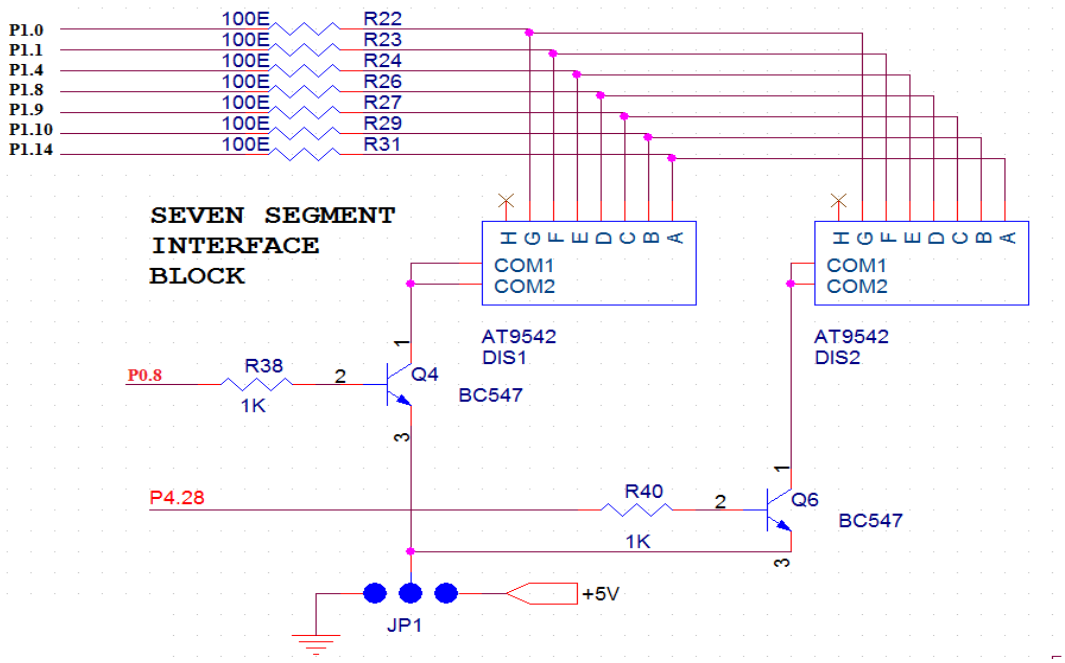
Although a 7-segment display can be thought of as a single display, it is still seven individual LEDs within a single package and as such these LEDs need protection from over current. LEDs produce light only when it is forward biased with the amount of light emitted being proportional to the forward current.

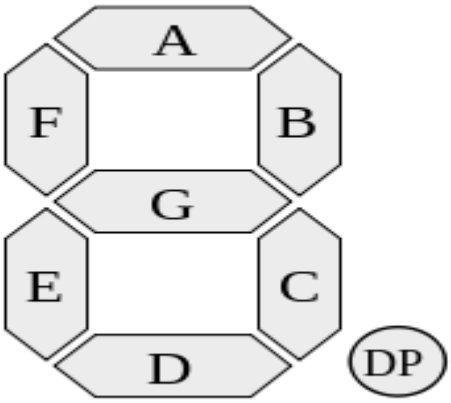
This means then that an LEDs light intensity increases in an approximately linear manner with an increasing current. So this forward current must be controlled and limited to a safe value by an external resistor to prevent damage to the LED segments.

The forward voltage drop across a red LED segment is very low at about 2-to-2.2 volts, (blue and white LEDs can be as high as 3.6 volts) so to illuminate correctly, the LED segments should be connected to a voltage source in excess of this forward voltage value with a series resistance used to limit the forward current to a desirable value.

Typically for a standard red coloured 7-segment display, each LED segment can draw about 15 mA to illuminated correctly, so on a 5 volt digital logic circuit, the value of the current limiting resistor would be about 200Ω (5v – 2v)/15mA, or 220Ω to the nearest higher preferred value.

So to understand how the segments of the display are connected to a 220Ω current limiting resistor consider the circuit below.





For illuminate a digit we have to make a appropriate **port Low (send 0).Here 1st make P0.8 or P4.28 High.** It will send control active signal to segment. Then initially make **P1.0,P1.1,P1.4,P1.8,P1.9,P1.10, P1.14 all high. Now all the digits become off. Then make respected segment low by sending CLR signal to GPIO.**

This cause the segment to illuminate.

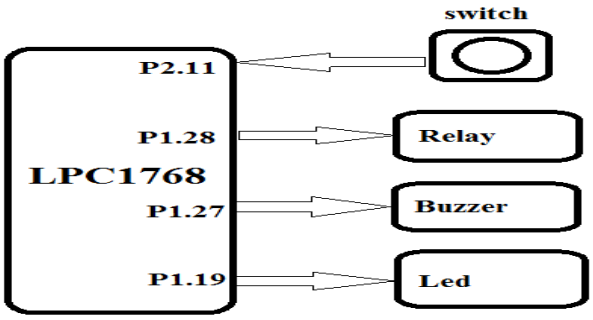
Main Code:-

```
#include <lpc17xx.h>

void delay_ms(unsigned int ms)
{
   unsigned int i,j;

   for(i=0;i<ms;i++)
      for(j=0;j<40000;j++);
}

/* start the main program */
int main()
{
   SystemInit();                  //Clock and PLL configuration

   LPC_GPIO0->FIODIR = 0x00000100;  //Configure the PORT0 pins as OUTPUT;
 LPC_GPIO1->FIODIR = 0x00004713;// all segment as output



   while(1)
   {

      LPC_GPIO0->FIOSET = 0x00000100;     // Make all the Port pins as high


                  LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
      LPC_GPIO1->FIOSET = 0x00004000;// set g to make 0

      delay_ms(200);
                  LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
```

```
LPC_GPIO1->FIOSET = 0x00004701;//set a,d,e,f,g to make 1
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000410;//set c,f to make 2
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000600;//set e and f to make 3
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000301;//set a,d,e to make 4
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000202;//set b,e to make 5
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000002;//set b to make 6
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00004700;//set d,e,f,g to make 7
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000000;//set all 0 to make 8
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000300;//set e to make 9
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000100;//set d to make A
delay_ms(200);
LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
LPC_GPIO1->FIOSET = 0x00000003;//set a,b to make b
```

```
            delay_ms(200);
            LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
            LPC_GPIO1->FIOSET = 0x00004012;//set set b,c,g to make C
            delay_ms(200);
            LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
            LPC_GPIO1->FIOSET = 0x00000401;//set a,f to make d
            delay_ms(200);
            LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
            LPC_GPIO1->FIOSET = 0x00000012;//set b,c to make E
            delay_ms(200);
            LPC_GPIO1->FIOCLR = 0x00004713;//clear all segments
            LPC_GPIO1->FIOSET = 0x00000112;//set b,c,d to make F
            delay_ms(200);


    }
}
```

Experiment No 10:-Interface a switch and display status in LED, Relay and Buzzer



Here switch is pulled up to +3.3 volt. So when pressed port should read active low signal(0),normal condition is active high(1)

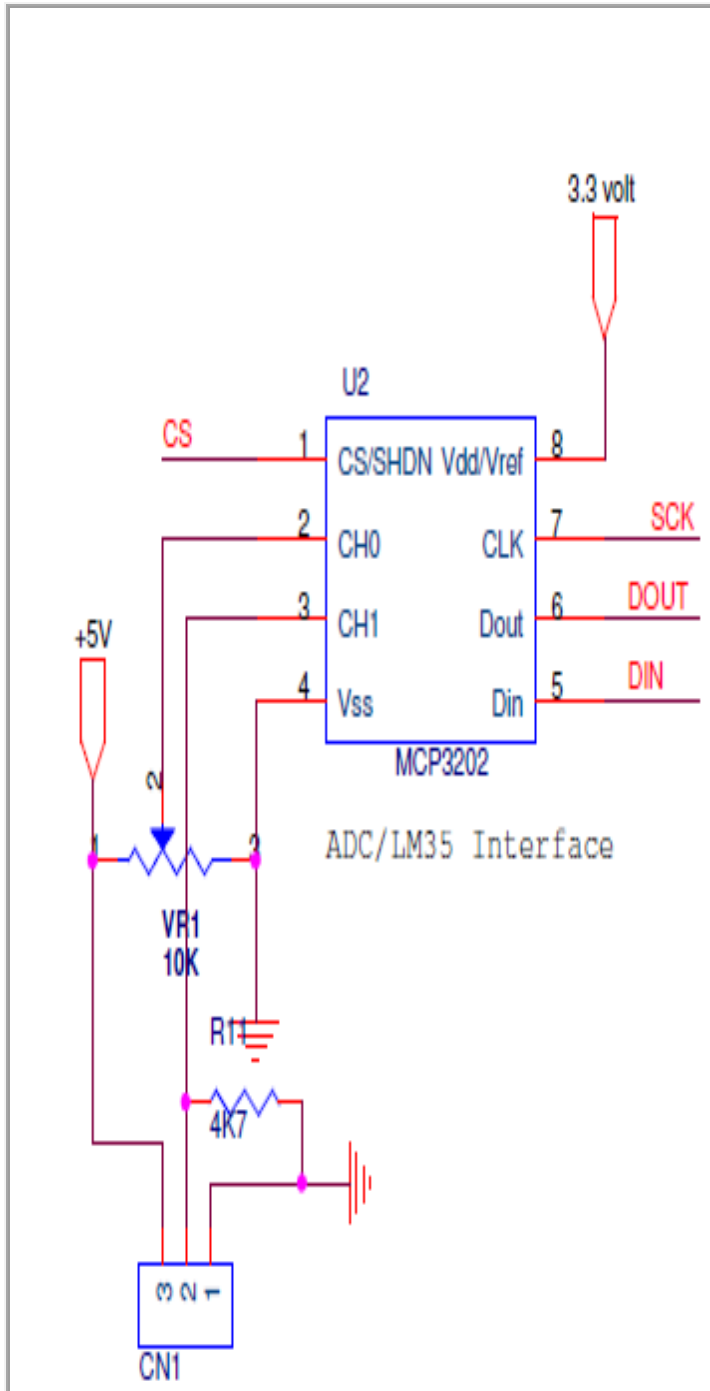
Main code:-

```
#include <lpc17xx.h>

#define SwitchPinNumber 11// port P2.11 connected to switch
#define LedPinNumber     19 //port 1.19-p1.26 connected to 8 leds
#define relayPinNumber     28 //port P1.28 connected to relay
#define buzzerPinNumber     27 //port P1.27 connected to buzzer

int main (void)
{

  uint32_t switchStatus;
  LPC_GPIO1->FIODIR = 0x1ff80000;              /* P1.xx defined as Outputs */
  LPC_GPIO1->FIOCLR = 0x1ff80000;              /* turn off all the LEDs&relay&buzzer */
  LPC_GPIO2->FIODIR = 0x00000000;              /* P2.xx defined as input */
```

```
 while(1)
   {

    /* Turn On all the leds and wait for one second */
      switchStatus = (LPC_GPIO2->FIOPIN>>SwitchPinNumber) & 0x01 ;  // Read the switch
status

      if(switchStatus == 0)                    //if switch pressed
      {
       LPC_GPIO1->FIOPIN =
(1<<buzzerPinNumber)|(1<<relayPinNumber)|(1<<LedPinNumber);//turn on led,relay and
buzzer

      }
      else          //if switch released
      {
       LPC_GPIO1->FIOPIN =
(0<<buzzerPinNumber)|(0<<relayPinNumber)|(0<<LedPinNumber);          //if switch released
led,relay,buzzer off
      }
}
}
```

Experiment No 11:-Interface SPI ADC and display Ambient temperature

Pin Assignment with LPC1768

	SPI - ADC	LPC1768 Lines	SPI - ADC
MCP 3202	CS	P0.28	
	CLK	P0.27	
	Dout	P3.26	
	Din	P3.25	

Main code:-

```
#include <LPC17xx.H>
#include <stdint.h>
#include <stdio.h>
#include "delay.h"
#include "spi_manul.h"
#include "lcd.h"
#define pulse_val 2
main()
{
unsigned int spi_rsv=0;
float vin;
char buf[20];
 SystemInit ();
   lcd_init();
   lcd_str("SPI 3202-b");
   delay(60000);
     delay(60000);
   while(1)
   {
   lcd_clr();
   lcd_cmd(0x80);
   spi_rsv = spi_data1(15);
   vin = ( ( spi_rsv & 0xfff ) * (3.3) ) / 4096  ;
    sprintf(buf,"Temp: %0.2f degC",(vin*100) );
    lcd_str(buf);
   delay(50000);
   delay(50000);
   }
}
```

SPI ADC data fetching program:-

```
#include <LPC17xx.H>
#include "delay.h"
#define pulse_val 2
#define CLK 1<<27
#defineCS       1<<28
#define DDOUT 26
#defineDOUT  1<<25
#defineDIN       1<<26
#define spi_stst 0
unsigned int spi_data(char sel)
{
  char clks = 4;
  LPC_GPIO0->FIODIR |= CS|CLK;
  LPC_GPIO3->FIODIR = DOUT;

  LPC_GPIO0->FIOSET = CS|CLK;
  LPC_GPIO3->FIOCLR       = DOUT;
   nop_delay(100);
#if spi_stst
  if ( LPC_GPIO3->FIOPIN & DIN )
  {
     return 'P';
  }
#endif
   LPC_GPIO0->FIOCLR = CS;
            nop_delay(pulse_val);
   while(clks)
   {
            LPC_GPIO0->FIOCLR = CLK;
             nop_delay(pulse_val);
```

```
            LPC_GPIO3->FIOPIN = (sel & 1) << DDOUT;
            sel = sel >> 1;

            LPC_GPIO0->FIOSET = CLK;
            nop_delay(pulse_val);

            clks--;
   }
    LPC_GPIO0->FIOCLR = CLK;
            nop_delay(pulse_val);
   if (!( LPC_GPIO3->FIOPIN & DIN ) )
  {
     return 'U';
  }
                    clks = 12;
   while(clks)
   {
     clks--;
            LPC_GPIO0->FIOCLR = CLK;
            nop_delay(pulse_val);
            LPC_GPIO0->FIOSET = CLK;
            nop_delay(pulse_val);
   }
            nop_delay(pulse_val);
  if (!( LPC_GPIO3->FIOPIN & DIN ) )
  {
     return 'U';
  }
   return 'Z';
}
unsigned int spi_data1(char sel)
```

```
{
 unsigned int spi_reg=0;
  char clks = 12;
  LPC_GPIO0->FIODIR |= CS|CLK;
  LPC_GPIO3->FIODIR = DOUT;

  LPC_GPIO0->FIOSET = CS|CLK;
  LPC_GPIO3->FIOSET       = DOUT;
  LPC_GPIO3->FIOPIN       = DIN;

   nop_delay(100);

  LPC_GPIO0->FIOCLR = CS;
  //start condi

  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET       = DOUT;
  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
  nop_delay(5);

  //single mode
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET       = DOUT;

  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
  nop_delay(5);
  //chanl 1
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET       = DOUT;
```

```
  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
  nop_delay(5);
  //msb first
  LPC_GPIO0->FIOCLR = CLK;
  LPC_GPIO3->FIOSET        = DOUT;

  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
  nop_delay(5);
  //smpling
//  LPC_GPIO0->FIOCLR = CLK;
//  nop_delay(pulse_val);
//  LPC_GPIO0->FIOSET = CLK;
//  nop_delay(2);

  //null bit
  LPC_GPIO0->FIOCLR = CLK;
  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
//  while ( ( LPC_GPIO3->FIOPIN & DIN ) == DIN );
// if( !( LPC_GPIO3->FIOPIN & DIN ) );
//  {
//      return 'U';
//  }
  nop_delay(5);
            clks = 12;
            while(clks)
            {
              LPC_GPIO0->FIOCLR = CLK;
```

```
  nop_delay(pulse_val);
  LPC_GPIO0->FIOSET = CLK;
 if( ( LPC_GPIO3->FIOPIN & DIN ) )
  {
     spi_reg |= 1<<(clks-1);
  }
  else
  {
              spi_reg = spi_reg;
  }
  clks--;
  nop_delay(5);
            }
  nop_delay(1);
   return spi_reg;
}
```

LCD display program:-

```
#include <LPC17xx.H>
#include "delay.h"
#define RRW (7<<9)
#define DATA_L (15<<19)
void lcd_pin(void)
{
   LPC_GPIO0->FIODIR |= RRW|DATA_L;
}

void lcd_cmd(unsigned char cmd)
{
```

```
        LPC_GPIO0->FIOPIN =( ( ( cmd & 0xf0 )>> 4) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN =( ( ( cmd & 0xf0 )>> 4) << 19 );

        LPC_GPIO0->FIOCLR |= RRW|DATA_L;
        delay(10);

        LPC_GPIO0->FIOPIN = ( ( ( cmd & 0xf ) ) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN = ( ( ( cmd & 0xf ) ) << 19 );

}

void lcd_data(unsigned char cmd)
{
LPC_GPIO0->FIOPIN =(1<<9)|( ( ( cmd & 0xf0 )>> 4) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN =( ( ( cmd & 0xf0 )>> 4) << 19 );

        LPC_GPIO0->FIOCLR |= RRW|DATA_L;
        delay(10);

        LPC_GPIO0->FIOPIN = (1<<9)|( ( ( cmd & 0xf ) ) << 19 )| (1<<11);
        delay(200);
        LPC_GPIO0->FIOPIN = ( ( ( cmd & 0xf ) ) << 19 );
}
void lcd_init(void)
{
lcd_pin();
  lcd_cmd(0x03);
  delay(3000);
```

```
  lcd_cmd(0x03);
  delay(1000);
  lcd_cmd(0x03);
  delay(100);
  lcd_cmd(0x2);
  lcd_cmd(0x28);
  lcd_cmd(0x0e);
  lcd_cmd(0x06);
  lcd_cmd(0x01);
  delay(1);
}

void lcd_str(char *lstr)
{
  while(*lstr)
  {
    lcd_data(*lstr);
          lstr++;
  }
}
void lcd_clr(void)
{
           lcd_cmd(0x03);
  delay(10);
  lcd_cmd(0x2);
  lcd_cmd(0x28);
  lcd_cmd(0x0e);
  lcd_cmd(0x06);
  lcd_cmd(0x01);
  delay(10);
}
```

4. Viva Question

1. These are few very simple and general ARM processor interview questions
2. What are the types of CORTEX-M series ?
3. How do you select a specific CORTEX-M processor ?
4. What is Microprocessor?
   Microprocessor is a CPU fabricated on a single chip program controlled device, which fetched the instructions from memory, decodes and execute the instructions. Three basic characteristic differentiate microprocessor.
5. Instruction Set: The set of instruction microprocessor can execute.
6. Bandwidth: The number of bit's processed in a single instruction
7. Clock Speed: Given in MHz Megahertz, the clock speed determines how many instructions per second the processor can execute.
8. In addition to this, microprocessors are classified as being RISC (Reduced Instruction Set Computer) or CISC (Complex Instruction Set Computer).
9. What are the basic units of Microprocessor?
   The basic units or block of microprocessor are ALU, an Array of Registers and control unit.
10. Give Examples for8/16/32-bit Microprocessor?
    8-bit Processors- 8085, Z80, 6800
    16-bit Processors- 8086, 68000, Z8000
    32-bit Processors- 80386, 80486
    64-bit Processors- Intel 64(x64), AMD64, IBM (Power PC), SUN (SPARC).
11. What are 1^{st}/ 2^{nd}/$3^{rd/4th}$ generation processors?
    The processors made of PMOS, NMOS, HMOS, HCMOS technology are called 1^{st}/ 2^{nd}/$3^{rd/4th}$ generation processor's and are made up of 4, 8, 16, 32-bits.
12. What does microprocessor speed depends on?
    The speed of microprocessor depends on various factors such as Data Bus Width (Number of instruction it processes) and clock speed.
13. What is Software and Hardware?
    The software is set of instruction or commands needed for performing a specific task by programmable device or a computing machine. The hardware refers to the component or device used to form computing machine in which software can be run and tested. Without software hardware is idle machine.
14. Distinguish between microprocessor and microcontroller?
    The microprocessor is a digital integrated circuit that can be programmed with a series of instructions to perform a specified function on data. The microcontroller is tiny little computer on single integrated circuit, which has memory, input-output on chip itself. So we can say microprocessor can perform few functions but microcontroller can perform many functions.
15. What are disadvantages of Microprocessor?
    Microprocessor has limitation on size of data. Most microprocessor does not support floating point operation.
16. What is the difference between microprocessor and microcontroller?
    op-codes and more bit handling instructions also microcontroller defined as a device that includes microprocessor, memory and input-output signal lines in a single chip.
17. What is an Instruction?
    An instruction is an order given to a computer processor by a computer program. At the lowest level each instruction set is a sequence of 0s and 1s that describes a physical

operation that computer is to perform (such as “Add”) and depending on the particular instruction type, the specification of special storage areas called registers that may contain data be used in carrying out the instruction or the location in computer memory of data.

18. What is clock cycle?
    The speed of computer processor is determined by clock cycle, which is amount of time between two pulses of an oscillator. In general, the higher number of pulses per second the faster the computer processor will be able to process information.
19. ARM stands for Advanced RISC Machines _____
    a) Advanced Rate Machines
    b) Advanced RISC Machines
    c) Artificial Running Machines
    d) Aviary Running Machines
    View Answer
    Answer: b
    Explanation: ARM is a type of system architecture.
20. The main importance of ARM micro-processors is providing operation with _____
    a) Low cost and low power consumption
    b) Higher degree of multi-tasking
    c) Lower error or glitches
    d) Efficient memory management
    View Answer
    Answer: a
    Explanation: The Stand alone feature of the ARM processors is that they’re economically viable.
21. ARM processors where basically designed for _____
    a) Main frame systems
    b) Distributed systems
    c) Mobile systems
    d) Super computers
    View Answer
    Answer: c
    Explanation: These ARM processors are designed for handheld devices.
22. The ARM processors doesn’t support Byte address ability ?
    a) True
    b) False
    View Answer
    Answer: b
    Explanation: The ability to store data in the form of consecutive bytes.
23. The address space in ARM is _____
    a) 2^{24}
    b) 2^{64}
    c) 2^{16}
    d) 2^{32}
    View Answer
    Answer: d
    Explanation: None.

24. The address system supported by ARM systems is/are _____
    a) Little Endian
    b) Big Endian
    c) X-Little Endian
    d) Both Little & Big Endian
    View Answer
    Answer: d
    Explanation: The way in which, the data gets stored in the system or the way of address allocation is called as address system.
25. Memory can be accessed in ARM systems by _____ instructions.
    i) Store
    ii) MOVE
    iii) Load
    iv) arithmetic
    v) logical
    a) i,ii,iii
    b) i,ii
    c) i,iv,v
    d) iii,iv,v
    View Answer
    Answer: b
    Explanation: None.
26. RISC stands for _____
    a) Restricted Instruction Sequencing Computer
    b) Restricted Instruction Sequential Compiler
    c) Reduced Instruction Set Computer
    d) Reduced Induction Set Computer
    View Answer
    Answer: c
    Explanation: This is a system architecture, in which the performance of the system is improved by reducing the size of the instruction set.
27. In ARM, PC is implemented using ____
    a) Caches
    b) Heaps
    c) General purpose register
    d) Stack
    View Answer
    Answer: c
    Explanation: PC is the place where the next instruction about to be executed is stored.
28. The additional duplicate register used in ARM machines are called as _____
    a) Copied-registers
    b) Banked registers
    c) EXtra registers
    d) Extential registers
    View Answer
    Answer: b
    Explanation: The duplicate registers are used in situations of context switching.

29. The banked registers are used for _____
    a) Switching between supervisor and interrupt mode
    b) Extended storing
    c) Same as other general purpose registers
    d) None of the mentioned
    View Answer
    Answer: a
    Explanation: When switching from one mode to another, instead of storing the register contents somewhere else it'll be kept in the duplicate registers and the new values are stored in the actual registers.
30. Each instruction in ARM machines is encoded into ____ Word.
    a) 2 byte
    b) 3 byte
    c) 4 byte
    d) 8 byte
    View Answer
    Answer: c
    Explanation: The data is encrypted to make them secure.
31. All instructions in ARM are conditionally executed.
    a) True
    b) False
    View Answer
    Answer: a
    Explanation: None.
32. The addressing mode where the EA of the operand is the contents of Rn is _____
    a) Pre-indexed mode
    b) Pre-indexed with write back mode
    c) Post-indexed mode
    d) None of the mentioned
    View Answer
33. The effective address of the instruction written in Post-indexed mode, MOVE[Rn]+Rm is _____
    a) EA = [Rn].
    b) EA = [Rn + Rm].
    c) EA = [Rn] + Rm
    d) EA = [Rm] + Rn
    View Answer
    Answer: a
    Explanation: Effective address is the address that the computer acquires from the current instruction being executed.