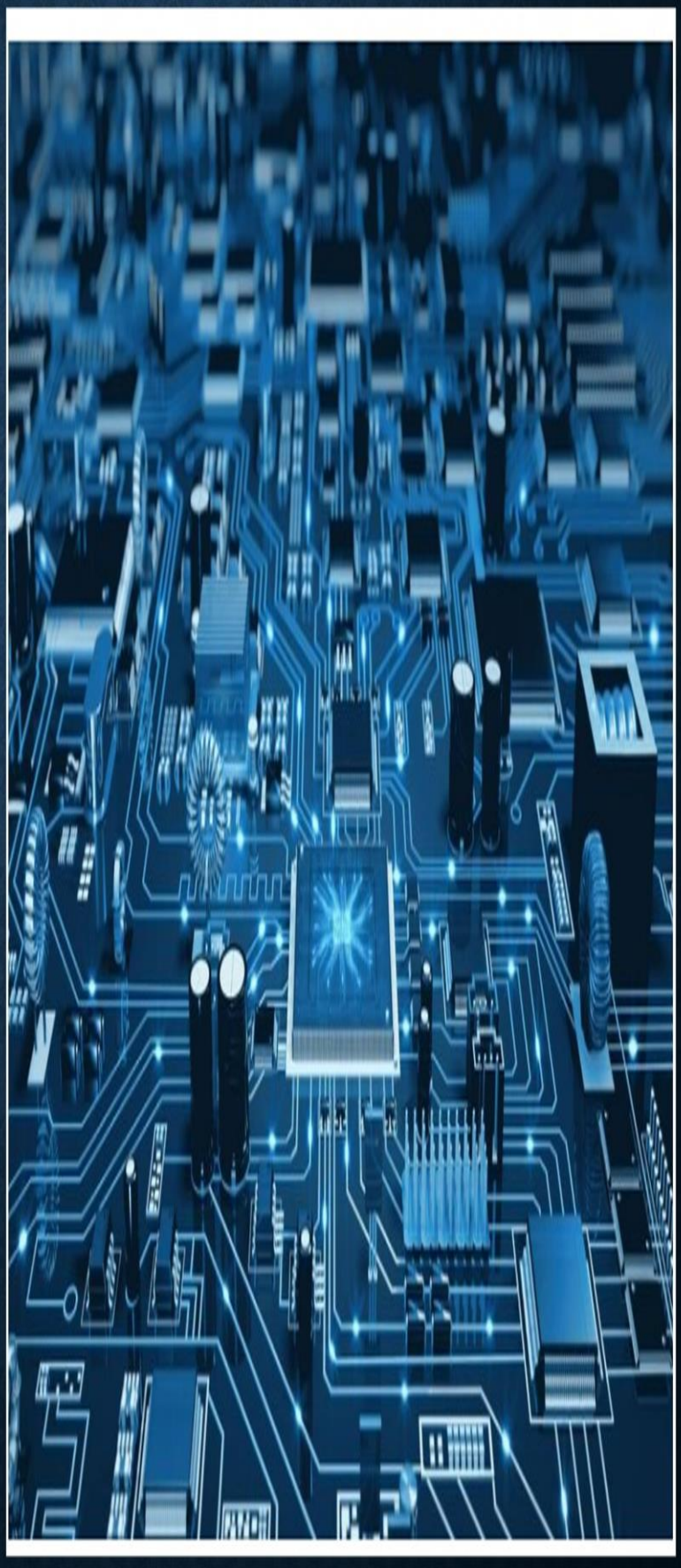
ISBN 978-93-6128-610-0

EMBEDDED SYSTEM LABORATORY MANUAL



K. PAVITHRA, M.E

REGULATION 2017

EMBEDDED SYSTEM LABORATORY MANUAL

(ELECTRONICS AND COMMUNICATION ENGINEERING)

REGULATION-2017

AUTHOR:

K. PAVITHRA., M.E

GENERAL GUIDELINES AND SAFETY NSTRUCTIONS

1. Sign in the log register as soon as you enter the lab and strictly observe your lab timings.
2. Strictly follow the written and verbal instructions given by the teacher / Lab Instructor. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teacher.
3. **Never work alone!** You should be accompanied by your laboratory partner and / or the instructors / teaching assistants all the time.
4. It is mandatory to come to lab in a formal dress and wear your ID cards.
5. Do not wear loose-fitting clothing or jewels in the lab. Rings and necklaces are usual excellent conductors of electricity.
6. Mobile phones should be switched off in the lab. Keep bags in the bag rack.
7. Keep the labs clean at all times, no food and drinks allowed inside the lab.
8. Intentional misconduct will lead to expulsion from the lab.
9. Do not handle any equipment without reading the safety instructions. Read the handout and procedures in the Lab Manual before starting the experiments.
10. Do your wiring, setup, and a careful circuit checkout before applying power. Do not make circuit changes or perform any wiring when power is on.
11. Avoid contact with energized electrical circuits.
12. Do not insert connectors forcefully into the sockets.
13. **NEVER** try to experiment with the power from the wall plug.
14. Immediately report dangerous or exceptional conditions to the Lab instructor / teacher: Equipment that is not working as expected, wires or connectors are broken, the equipment that smells or "smokes". If you are not sure what the problem is or what's going on, switch off the Emergency shutdown.
15. Never use damaged instruments, wires or connectors. Hand over these parts to the Lab instructor/Teacher.
16. Be sure of location of fire extinguishers and first aid kits in the laboratory.
17. After completion of Experiment, return the bread board, trainer kits, wires, CRO probes and other components to lab staff. Do not take any item from the lab without permission.
18. Observation book and lab record should be carried to each lab. Readings of current lab experiment are to be entered in Observation book and previous lab experiment should be written in Lab record book. Both the books should be corrected by the faculty in each lab.
19. Special Precautions during soldering practice

    a. Hold the soldering iron away from your body. Don't point the iron towards you. b. Don't use a spread solder on the board as it may cause short circuit.

    c. Do not overheat the components as excess heat may damage the components/board.

    d. In case of burn or injury seek first aid available in the lab or at the college dispensary.

PREFACE

This book on "**EMBEDDED SYSTEMS LABORATORY MANUAL (Electronics and communication Engineering)**" covers the complete syllabus prescribed by the Anna University, Chennai for the seventh semester **B.E/ B.Tech**. Degree course under **Outcome Based Education Credit System with the regulation 2017**.

This book covers Study of ARM evaluation system, Interfacing ADC and DAC, Interfacing LED and PWM, Interfacing real time clock and serial port, Interfacing keyboard and LCD, Interfacing EPROM and interrupt, Mailbox, Interrupt performance characteristics of ARM and FPGA, Flashing of LEDS, Interfacing stepper motor and temperature sensor, Implementing zig-bee protocol with ARM.

We hope that this book will be useful to both teachers and students. Finally we would request the readers to kindly send their valuable comments and suggestions towards the improvement of the manual and the same will be gratefully acknowledge.

Any suggestion from the reader for the betterment of this book can be dropped into kpavi.05ap@gmail.com

Mrs.K.PAVITHRA M.E.

ACKNOWLEDGEMENT

We are thankful to and fortunate enough to get constant encouragement, support and guideline from Chairman **Thiru.S.Ramadoss**, Secretary & Treasurer **Mr.G.Thamotharan** for his blessings to complete the book successfully.

We would not forget to remember our Principal **Dr.T.K.Gopinathan** and Vice-Principal **Dr.D.Saravanan** for his constant assistance in preparing this book.

ANNAI MIRA

College of engineering & technology
Nh- 46 chennai-bengaluru highway, vellore dist-632503

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

EC 8711 – EMBEDDED LABORATORY

ANNAI MIRA

COLLEGE OF ENGINEERING AND TECHNOLOGY

NH – 46 CHENNAI – BENGALURU HIGHWAY, VELLORE, VELLORE DIST -632503

BONAFIDE CERTIFICATE

NAME OF THE STUDENT : …….

REGISTER NUMBER : …….....

YEAR / SEMESTER : …….....

SUBJECT CODE / NAME : …….....

BATCH : …….

This is to certify that Mr/Ms.....With Register number …… has successfully completed the .....Laboratory during the academic year …….

The practical examination should be held on ……..

Staff – Incharge

Head of the department

Internal – examiner

External – examiner

EC8711 – EMBEDDED LABORATORY

INDEX

S.No	Date	Name of the Experiment	Pg.no	Signature

LED INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to blink the LED with Specific delay using LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

LED (Light Emitting Diodes)

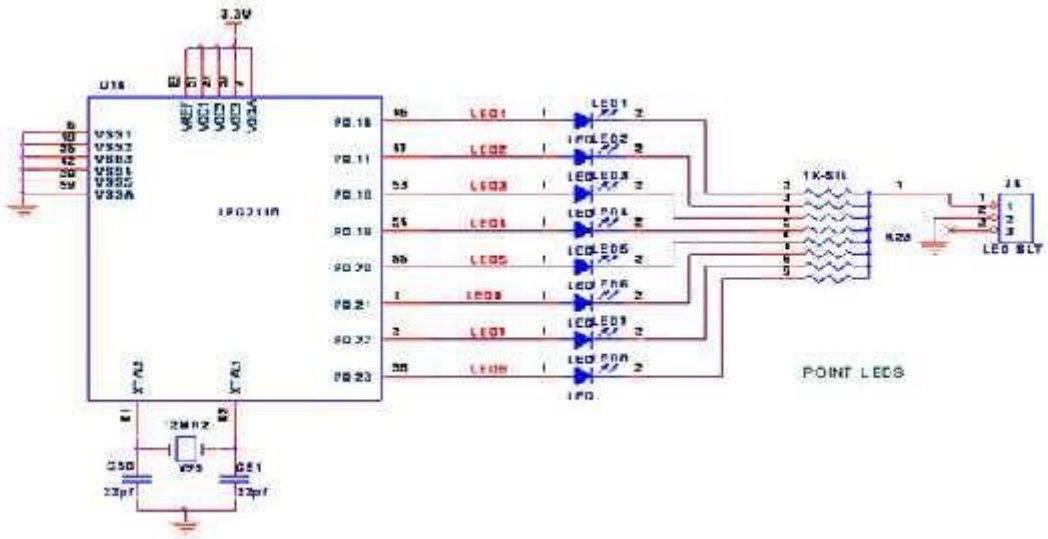
Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

Interfacing LED with LPC2148

Flash a LED using LPC2148 Primer Board. It works by turning ON a LED & then turning it OFF & then looping back to START. However the operating speed of microcontroller is very high so the flashing frequency will also be very fast to be detected by human eye.

The ARM7 LPC2148 Primer board has eight numbers of point LEDs, connected with I/O Port lines (P1.16 – P1.23) to make port pins high.

Circuit Diagram to Interface LED with LPC :



Program :

```
#include "lpc214x.h"
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
#define LEDS 0xFF<<8 //P0.8 to P0.15

void delayms(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<1007;j++) //5035 for 60Mhz ** 1007 for 12Mhz
    {;}
 }
}
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz */
void clock_select(void)
{
//Fosc = 12Mhz //Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
PLL0CON = 3; //Enable PLL0
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
/*-------------------------------------------------------------*/
int main(void)
{
  clock_select();

IODIR0 = LEDS; //Configure Port0 as output Port
PINSEL0 = 0;                    //Configure Port0 as General Purose IO

while(1)
  {
    IOSET0 = LEDS; //Set P0.15-P0.8 to '1'
    delayms(1000); //1 sec Delay
    IOCLR0 = LEDS; //Set P0.15-P0.8 to '0'
    delayms(1000); //1 Sec Delay
   }
}
/*-------------------------------------------------------------*/
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to blink the LED with Specific delay using LPC 2148 was written and verified Successfully using Keil C.

LCD INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to display the message on LCD interfacing with LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil μ-Vision – 4 Software

Theory :

Various graphical LCDs are available in the market with different sizes. Here **JHD12864E** Graphical LCD has been explained. This LCD has a display format of **128x64 dots** and has yellow-green color backlight. Each LCD needs a controller to execute its internal operations. This LCD uses two **KS0108 controllers**.

The **128x64 LCD** is divided into two equal halves with each half being controlled by a separate KS0108 controller. Such LCDs (using KS0108 controller) involve paging scheme, i.e., whole LCD is divided equally into pages.

1. 128x64 LCD implies 128 columns and 64 rows. In total there are 1024 pixels.
2. 128x64 LCD is divided equally into two halves. Each half is controlled by a separate controller and consists of 8 pages. In above diagram, CS stands for Controller Select.
3. Each page consists of 8 rows and 64 columns. So two horizontal pages make 128 (64x2) columns and 8 vertical pages make 64 rows (8x8).

Algorithm :

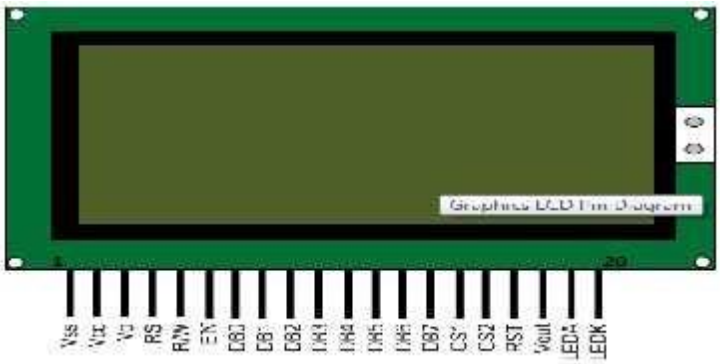
- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Program :

```
#include <LPC214X.H>
#define LCD_RS 1<<24 //Port1.24
#define LCD_RW 1<<16 //Port0.16
#define LCD_EN 1<<17 //Port0.17
#define LCD_DATA 0xFF<<16 //Port1.16 to 1.23
#define LCD_STS 1<<23 //Port1.23

//Function declaration
int main(void);
void lcdini(void);
void lsts(void);
void lcdctl(unsigned char val1);
void lputc(unsigned char lcr);
void put_s(char *str);
void delay (unsigned int k);

int main(void)
{
  IODIR0 = LCD_RW | LCD_EN; //configure LCD r/w & EN as o/p
  IODIR1 = LCD_RS; //Configure RS & DATA as o/p

  PINSEL0 = 0; // Configure Port0 as General Purpose IO
  PINSEL1 = 0; // Configure Port1 as General Purpose IO
 IOCLR0=LCD_EN; IOCLR1=LCD_RS; IOSET0=LCD_RW; /* DISABLE LCD TEMPORALY*/
  delay(1000); //LCD Power-up Delay
  lcdini(); //Initialise LCD
   put_s("TEST MESSAGE"); //Print msg
  while(1); //Terminate Program
}
/*-------------------------------------------------------------*/
void lcdini()
{
  lcdctl(0x38); /* Function Set 2 LINE 5 X 8 CHAR*/
  delay(5);                       /* Waits for 5 Msec. */
  lcdctl(0x38); /* Sends Function Set - AGAIN */
  delay(5);                       /* Waits for 5 Msec. */
  lcdctl(0x38);   /* Sends Function Set - AGAIN */
  lsts();                         /* Wait Till BUSY=0 */
  lcdctl(0x38);   /* Sends Function Set - AGAIN*/
  lsts();                         /* Wait Till BUSY=0 */
  lcdctl(0x04); /* Display off */
  lsts();                         /* Wait Till BUSY=0 */
  lcdctl(0x01); /* Clear Display */
  lsts();                         /* Wait Till BUSY=0 */
  lcdctl(0x06); /* Set Entry mode */
  lsts();                         /* Wait Till BUSY=0 */
  lcdctl(0x0c); /* Set Display ON */
 }
/******* Checks the LCD Status for busy**************/
void lsts()
{unsigned long int tp1;
  IOCLR1=LCD_RS; IOSET0=LCD_RW;
```

```
do{
  IOSET0=LCD_EN;
  tp1 = IOPIN1 & LCD_STS;
  IOCLR0=LCD_EN;
  }while(tp1);
 IOCLR0=LCD_EN;
 IOCLR0=LCD_RW;
}
/*-------------------------------------------------------------*/
void lcdctl(unsigned char val1)
{ unsigned long int dat;
 dat = ((unsigned long int)val1) << 16;
IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 /* WRITE COMMAND TO CONTROL REGISTER*/
 IOCLR1 = LCD_RS;
 IOCLR0 = LCD_RW;
 IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/******* Displays a Character in the LCD ******/
void lputc(unsigned char lcr)
{ unsigned long int dat;
 lsts();
 dat = ((unsigned long int)lcr)<<16;
 IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 IOSET1=LCD_RS; IOCLR0=LCD_RW;
  IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/*-------------------------------------------------------------*/
void put_s(char *str)
{
  while(*str)
 { lputc(*str);
        str++;
   }
}
/*-----------------------------------------*/
//Delay Program
//Input - delay value in milli seconds
void delay(unsigned int k)
{
        unsigned int i,j;
        for (j=0; j<k; j++)
                for(i = 0; i<=800; i++);
}
     /*-------------------------------------------------------*/
```

Pin Diagram of LCD :



Pin Configuration of LCD :

Pin no.	Function	Name
1	Ground (0 V)	Vss
2	Supply voltage; 5V	Vcc
3	Contrast adjustment	Vo
4	High to display data; Low for instruction code	Register select (RS)
5	Low to write to the register; High to read from the register	Read/Write (R/W)
6	Reads data when high; Writes data at high to low transition (falling edge)	Enable (EN)
7	8-bit data bus	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Chip selection for IC1; Active high	CS1
16	Chip selection for IC2; Active high	CS2
17	Reset signal; Active low	RST
18	Output voltage for LCD driving	Vout
19	Backlight V_{CC} (5V)	LED A
20	Backlight Ground (0V)	LED K

Result :

Thus the Program to display the message on LCD interfacing with LPC 2148 was written and verified Successfully using Keil C.

TEMPERATURE SENSOR INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to measure the temperature using LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

Temperature Sensor

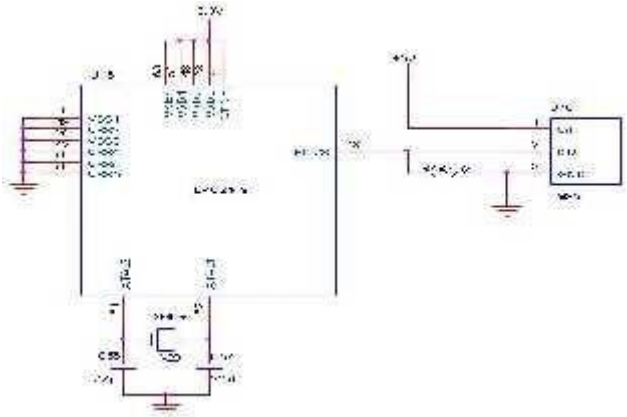
The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The output of sensor converted to digital that easy connecting with microcontroller.

Interfacing LM35 with LPC2148

Read the temperature in LPC2148 Tyro Board from temperature sensor LM35. The ARM7 LPC2148 Tyro board uses the ADC pin for reading temperature from temperature sensor LM35. The reading output is displayed into PC through UART1.

The 10 bit ADC used for reading the temperature from LM35. Basic clocking for the A/D converters is provided by the VPB clock. A programmable divider is included in each converter, to scale this clock to the 4.5 MHz (max) clock needed by the successive approximation process. A fully accurate conversion requires 11 of these clocks.

Circuit Diagram of LM35 with LPC 2148 :



Program :

```
#include <LPC214X.H>
/*---LCD Signal Declaration---*/
.24
#define LCD_RW 1<<16 //Port0.16
#define LCD_EN 1<<17 //Port0.17
#define LCD_DATA 0xFF<<16 //Port1.16 to 1.23
#define LCD_STS 1<<23 //Port1.23
/*--- ADC Signal Declaration */
#define AD0_3 1<< 28
#define CLK_DIV 1<<8
#define PDN 1<<21
#define SOC 1<<24
#define BURST 1<<16
#define DONE 1<<31

//Function declaration
int main(void);
void lcdini(void);
void lsts(void);
void lcdctl(unsigned char val1);
void lputc(unsigned char lcr);
void put_s(char *str);
void lputval(unsigned int lbuf);
unsigned int adc_read( unsigned char channel);
void adc_init(void);
void delay (unsigned int k);
 /*----Main Program------*/
int main(void)
{
unsigned int tp1;

  IODIR0 = LCD_RW | LCD_EN; //configure LCD r/w & EN as o/p
  IODIR1 = LCD_RS; //Configure RS o/p
    PINSEL0 = 0; //Configure Port0 as General Purpose IO
  PINSEL1 = 0 | AD0_3; // Enable AD0.3
   IOCLR0=LCD_EN; IOCLR1=LCD_RS; IOSET0=LCD_RW; /* DISABLE LCD TEMPORALY*/
  delay(1000); //LCD Power-up Delay
  lcdini();   //Initialise  LCD
  adc_init(); //Initialise on-chip ADC
  do
  { /* LCD - Return Home Command */
            lsts(); //Check LCD ready?
            lcdctl(2); //Send CMD
            lsts();    //Check LCD ready?
            put_s("Temprature="); //Print Msg in LCD
            tp1 = adc_read(3); // Channel AD0 0.3
     lputval(tp1);
     lputc('C');
  }while(1);
```

```
}
/*-------------------------------------------------------------*/
void lcdini()
{
  lcdctl(0x38);    /* Function Set 2 LINE 5 X 8 CHAR*/
  delay(5);                    /* Waits for 5 Msec. */
  lcdctl(0x38);    /* Sends Function Set - AGAIN */
  delay(5);                    /* Waits for 5 Msec. */
  lcdctl(0x38); /* Sends Function Set - AGAIN */
  lsts();                      /* Wait Till BUSY=0 */
  lcdctl(0x38); /* Sends Function Set - AGAIN*/
  lsts();                      /* Wait Till BUSY=0 */
  lcdctl(0x04); /* Display off */
  lsts();                      /* Wait Till BUSY=0 */
  lcdctl(0x01); /* Clear Display */
  lsts();                      /* Wait Till BUSY=0 */
  lcdctl(0x06); /* Set Entry mode */
  lsts();                      /* Wait Till BUSY=0 */
  lcdctl(0x0c); /* Set Display ON */
 }
/******* Checks the LCD Status for busy***************/
void lsts()
{
unsigned long int tp1;
 IOCLR1=LCD_RS; IOSET0=LCD_RW;
do{
  IOSET0=LCD_EN;
  tp1 = IOPIN1 & LCD_STS;
  IOCLR0=LCD_EN;
  }while(tp1);

 IOCLR0=LCD_EN;
 IOCLR0=LCD_RW;
}
void lcdctl(unsigned char val1)
{ unsigned long int dat;
  dat = ((unsigned long int)val1) << 16;
IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 /* WRITE COMMAND TO CONTROL REGISTER*/
 IOCLR1 = LCD_RS;
 IOCLR0 = LCD_RW;
 IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/******* Displays a Character in the LCD ******/
void lputc(unsigned char lcr)
{ unsigned long int dat;
```

```
 lsts();
 dat = ((unsigned long int)lcr)<<16;
 IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 IOSET1=LCD_RS; IOCLR0=LCD_RW;
  IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/*-------------------------------------------------------------*/
void put_s(char *str)
{
  while(*str)
 { lputc(*str);
          str++;
   }
}
/*-------------------------------------------------------------*/
void adc_init()
{
unsigned long int ADC_CH;

          ADC_CH = 0 | 1 << 3; //Channel AD0.3
    AD0CR = SOC | PDN | CLK_DIV | ADC_CH | BURST;
}
/*-------------------------------------------------------------*/
unsigned int adc_read( unsigned char channel)
{
            unsigned   int   aval;
            unsigned long int val;
    if (channel == 1) val = AD0DR1;
    else if (channel == 2) val = AD0DR2;
   else if (channel == 3) val = AD0DR3;
   val = val >> 6;
    val = val & 0x3FF;
    aval = val;
    aval = ((aval*100)/465);
    return (aval);
}
/*-------------------------------------------------------------*/
void lputval(unsigned int lbuf)
{          unsigned int val;
           unsigned char tp1;
   val=lbuf;
//Tens
  tp1 = (val/10) + '0';
  lputc(tp1);
//One
  tp1 = (val%10) + '0';
```

```
  lputc(tp1);

 }
//Delay Program //Input - delay value in milli seconds
void delay(unsigned int k)
{
        unsigned int i,j;
        for (j=0; j<k; j++)
                for(i = 0; i<=800; i++);
}
/*-------------------------------------------------------------*/
```

Algorithm :

- Open Keil μ-Vision 4
  - Create a Project and Choose the Hardware tool
  - Create a file , type the Program and Save it as .c format
  - Add the files to Source Group
  - Run the program and get the error free output.
  - Connect hardware to the PC and Download the Program using Flash Magic
  - Verify the output.

Result :

Thus the Program to measure the temperature using LPC 2148 was written and verified Successfully using Keil C.

STEPPER MOTOR INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to rotate the Stepper motor in both Clock wise and Anti – Clockwise direction using LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

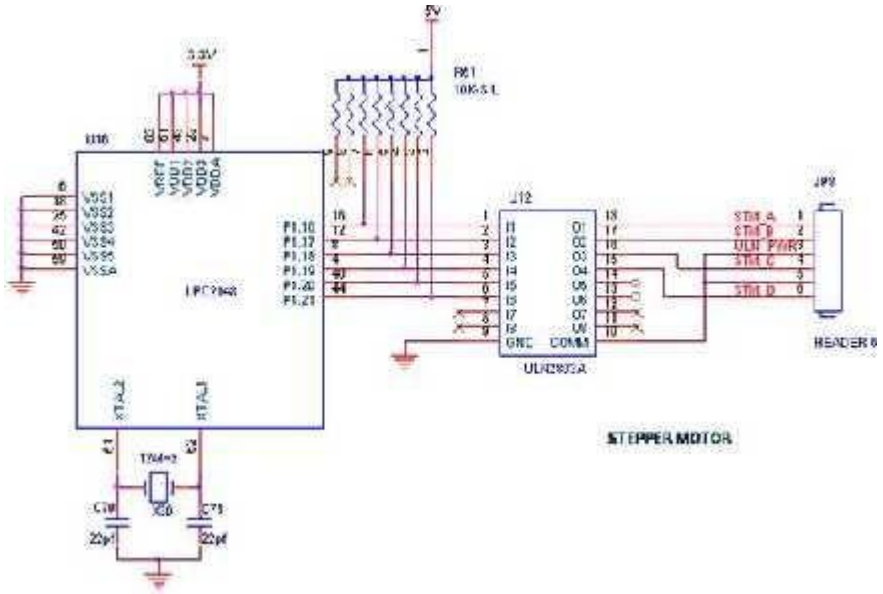
Stepper Motor

A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation. Every revolution of the stepper motor is divided into a discrete number of steps, and the motor must be sent a separate pulse for each step.

Interfacing Stepper Motor with LPC2148

Controlling a stepper motor using LPC2148 Primer Board. It works by turning ON & OFF a four I/O port lines generating at a particular frequency. The ARM7 LPC2148 Primer board has four numbers of I/O port lines, connected with I/O Port lines (P1.16 – P1.19) to rotate the stepper motor. ULN2803 is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.

Circuit Diagram to Interface Stepper Motor with LPC2148 :



Program :

```
#define STEP_DELAY 50 //Step dealy in msec.
#include "lpc214x.h"
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
//IO Ports
#define DIR 1<<16
#define STEP 0xF<<8
unsigned long int step_table[]={ 0x5, 0x9, 0xA,0x6};
void delayms(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<5035;j++) //5035 for 60Mhz ** 1007 for 12Mhz
    {;}
 }
}
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz      */
void clock_select(void)
{
//Fosc = 12Mhz //Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
PLL0CON = 3; //Enable PLL0
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
/*-------------------------------------------------------------*/
int main(void)
{
  unsigned char tptr;
 clock_select(); //Config. PLL0 & Set CPU clock 60Mhz
 PINSEL0 = 0; // Configure Port0 as General Purpose IO -- P0.0 to P0.15
 PINSEL1 = 0; // Configure Port0 as General Purpose IO -- P0.16 to P0.31
 IODIR0 = STEP; //Set Stepper motor control Signal as output
 tptr = 0; //Initialize pointer
do{
   IOCLR0 = STEP; //Set P0.11-P0.8 to '0'
   if(IOPIN0 & DIR) IOSET0 = step_table[tptr++] << 8 ;
      else IOSET0 = step_table[tptr--] << 8 ;
   tptr = tptr & 3; //Set pointer limit 0 to 3
   delayms(STEP_DELAY);
 }while(1);
}
/*-------------------------------------------------------------*/
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to rotate Stepper motor in both Clockwise and anti- Clockwise rotation using LPC 2148 was written and verified Successfully using Keil C.

ADC INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to interface ADC with LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil μ-Vision – 4 Software

Theory:

The reality is that a great many of the devices that you will want to interact with *aren't* going to be digital, or consist of only two possible 'states' (On/Off, True/False, etc.). Almost every dial or knob on any modern electronics device, for example, is probably analog. Since our microcontrollers are digital, what that means is that we need to find a way to convert those analog signals into something 'digital' that our microcontroller can actually understand. That's where Analog to Digital Converters (ADCs) comes in, and thankfully for us the LPC2148 has two of them built in.

ADCs essentially act as a bridge (or a 'translator') between the messy outside analog world and the cozy, black and white (green and white?) digital world our microcontrollers live in. They work by converting voltage to a numeric value that the microcontroller can understand. For example, with an internal voltage of 3.3V (which is the Vref on the LPC2148) and your ADC set to return the maximum 10-bit data (meaning you have possible values between 0 and 1023), 0.0V would return 0, 3.3V (or higher) would return 1023, and 1.65V would return ~512. They only work in one direction (reading data from outside and sending it 'into' the chip), but life without them would be a lot more challenging ... or at the very least a lot more expensive (analog devices are often much cheaper than their digital counterparts).

Basic clocking for the A/D converters is provided by the APB clock. A programmable divider is included in each converter, to scale this clock to the 4.5 MHz (max) clock needed by the successive approximation process. A fully accurate conversion requires 11 of these clocks.

Program :

```
#include <LPC214X.H>
/*---LCD Signal Declaration---*/
#define LCD_RS 1<<24 //Port1.24
#define LCD_RW 1<<16 //Port0.16
#define LCD_EN 1<<17 //Port0.17
#define LCD_DATA 0xFF<<16 //Port1.16 to 1.23
#define LCD_STS 1<<23 //Port1.23
/*--- ADC Signal Declaration */
#define AD0_1 1<< 24
#define CLK_DIV 1<<8
#define PDN 1<<21
#define SOC 1<<24
#define BURST 1<<16
#define DONE 1<<31
//Function declaration
int main(void);
void lcdini(void);
void lsts(void);
void lcdctl(unsigned char val1);
void lputc(unsigned char lcr);
void lputval(unsigned int lbuf);
unsigned int adc_read( unsigned char channel);
void adc_init(void);
void delay (unsigned int k);
 /*----Main Program------*/
int main(void)
{
unsigned int tp1;
  IODIR0 = LCD_RW | LCD_EN; //configure LCD r/w & EN as o/p
  IODIR1 = LCD_RS; //Configure RS o/p
    PINSEL0 = 0; //Configure Port0 as General Purpose IO
  PINSEL1 = 0 | AD0_1; // Enable AD0.1
   IOCLR0=LCD_EN; IOCLR1=LCD_RS; IOSET0=LCD_RW; /* DISABLE LCD TEMPORALY*/
  delay(1000); //LCD Power-up Delay
  lcdini();   //Initialise  LCD
  adc_init(); //Initialise on-chip ADC
 do
  { tp1 = adc_read(1); // Channel AD0 0.1
     lputval(tp1);
     delay(500);
     lcdctl(0x01); // Clear Display
     lsts();
  }while(1);
  return 0;
}
/*-------------------------------------------------------------*/
void lcdini()
{
  lcdctl(0x38);     /* Function Set 2 LINE 5 X 8 CHAR*/
```

```
  delay(5);                         /* Waits for 5 Msec. */
  lcdctl(0x38);    /* Sends Function Set - AGAIN */
  delay(5);                         /* Waits for 5 Msec. */
  lcdctl(0x38); /* Sends Function Set - AGAIN */
  lsts();                           /* Wait Till BUSY=0 */
  lcdctl(0x38); /* Sends Function Set - AGAIN*/
  lsts();                           /* Wait Till BUSY=0 */
  lcdctl(0x04); /* Display off */
  lsts();                           /* Wait Till BUSY=0 */
  lcdctl(0x01); /* Clear Display */
  lsts();                           /* Wait Till BUSY=0 */
  lcdctl(0x06); /* Set Entry mode */
  lsts();                           /* Wait Till BUSY=0 */
  lcdctl(0x0c); /* Set Display ON */
 }
/******* Checks the LCD Status for busy***************/
void lsts()
{
unsigned long int tp1;
  IOCLR1=LCD_RS; IOSET0=LCD_RW;
do{
  IOSET0=LCD_EN;
  tp1 = IOPIN1 & LCD_STS;
  IOCLR0=LCD_EN;
  }while(tp1);

 IOCLR0=LCD_EN;
 IOCLR0=LCD_RW;
}
void lcdctl(unsigned char val1)
{ unsigned long int dat;
  dat = ((unsigned long int)val1) << 16;
IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 /* WRITE COMMAND TO CONTROL REGISTER*/
 IOCLR1 = LCD_RS;
 IOCLR0 = LCD_RW;
 IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/******* Displays a Character in the LCD ******/
void lputc(unsigned char lcr)
{ unsigned long int dat;
 lsts();
 dat = ((unsigned long int)lcr)<<16;
 IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 IOSET1=LCD_RS; IOCLR0=LCD_RW;
  IOCLR1 = LCD_DATA;
```

```
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/*-------------------------------------------------------------*/
void adc_init()
{
unsigned long int ADC_CH;

            ADC_CH = 0 | 1 << 1; //Channel AD0.1
    AD0CR = SOC | PDN | CLK_DIV | ADC_CH | BURST ;
}
unsigned int adc_read( unsigned char channel)
{
             unsigned int aval;
             unsigned long int val;

    if (channel == 1) val = AD0DR1;
    else if (channel == 2) val = AD0DR2;
    else if (channel == 3) val = AD0DR3;

    val = val >> 6;
    val = val & 0x3FF;
    aval = val;
    return (aval);
}
/*-------------------------------------------------------------*/
void lputval(unsigned int lbuf)
{          unsigned int val;
           unsigned char tp1;
   val=lbuf;
//Seprate thousands
  tp1 = (val/1000) + '0';
  lputc(tp1);

  val = val%1000;
//Hundreds
  tp1 = (val/100) + '0';
  lputc(tp1);

  val=val%100;
//Tens
  tp1 = (val/10) + '0';
  lputc(tp1);
//One
  tp1 = (val%10) + '0';
  lputc(tp1);
 }
//Delay Program
```

```
//Input - delay value in milli seconds
void delay(unsigned int k)
{
        unsigned int i,j;
        for (j=0; j<k; j++)
                for(i = 0; i<=800; i++);
}
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to interface ADC with LPC 2148 was written and verified Successfully using Keil C.

DAC INTERFACING USING LPC 2148 – SINE WAVEFORM GENERATION

Ex.No :

Date :

Aim :

To write a program in Keil – C to generate Sine Waveform using DAC interfacing with LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil μ-Vision – 4 Software

Theory :

Digital to Analog Converter : The function of DAC is to convert digital values or binary patterns to analog signals usually. The DAC performs exactly reverse of ADC. LPC2148 has one Digital to Analog Converter which receives digital value from register and give output to A_{OUT} analog output pin.

Features of LPC2148 DAC

- 10 bit digital to analog converter
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable speed vs. power

Program : Generate Sine Waveform using DAC

```
#include <lpc214x.h>
#include <math.h>
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
#define AOUT 1<<19 // PINSEL1 value for AOUT config.
#define PI2    6.28    //2Pi = 3.14*2
#define STEPS  500    // No. of Steps per Cycle.
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz */
void clock_select(void)
{
//Fosc = 12Mhz
//Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
PLL0CON = 3; //Enable PLL0
//PLL FEED
```

```
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
/*---------------------------------------------------------*/
int main(void)
{
 unsigned short val, dval; //16bit variable
 float sa; //sine angle varies from 0 - 2pi
 clock_select(); //CPU Clock Configuration
PINSEL0 = 0;      //Configure Port0.0 to P0.15 as General Purose IO
PINSEL1 = AOUT; //Configure Port0.25 as Analog Output PIN
while(1)
  {
    for(val=0; val<STEPS; val++)
     {
       //Convert Steps as angle from 0-2pi
          sa = (val*PI2)/STEPS;
       //Find Sine Value of current angle
          sa = sin (sa);
       //Convert current value to DAC value 0-1022
      //Negative plots shifted above 0 hence mid point fixed at counts 511
          sa = (sa*511.0) + 511.0; //Mid point of 511
          dval = sa;
          DACR = dval<<6;            //Send value to DAC
     }
   }
 }
/*---------------------------------------------------------*/
```

Algorithm :

- Open Keil µ-Vision 4 and Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to generate Sine Waveform using DAC interfacing with LPC 2148 was written and verified Successfully using Keil C.

DAC INTERFACING USING LPC 2148 – SQUARE WAVEFORM GENERATION

Ex.No :

Date :

Aim :

To write a program in Keil – C to generate Square Waveform using DAC interfacing with LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil μ-Vision – 4 Software

Theory :

Digital to Analog Converter : The function of DAC is to convert digital values or binary patterns to analog signals usually. The DAC performs exactly reverse of ADC. LPC2148 has one Digital to Analog Converter which receives digital value from register and give output to A_{OUT} analog output pin.

Features of LPC2148 DAC

- 10 bit digital to analog converter
- Resistor string architecture
- Buffered output
- Power-down mode
- Selectable speed vs. power

Program : Generate Square Waveform using DAC

```
#include "lpc214x.h"
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
#define AOUT 1<<19 // PINSEL1 value for AOUT config.
void delayms(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<5035;j++) //5035 for 60Mhz ** 1007 for 12Mhz
   {;}
 }
}
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz      */
void clock_select(void)
```

```
{
//Fosc = 12Mhz
//Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;

PLL0CON = 3; //Enable PLL0
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
/*---------------------------------------------------------------*/
int main(void)
{

 clock_select(); //CPU Clock Configuration

PINSEL0 = 0;      //Configure Port0.0 to P0.15 as General Purose IO
PINSEL1 = AOUT; //Configure Port0.25 as Analog Output PIN

while(1)
  {
    DACR = 0;              //DAC ouput = 0V
    delayms(1);           //1 msec Delay
    DACR = 0x3FF << 6;      //DAC ouput = 2.2V
    delayms(1);           //1 mSec Delay
   }
}
/*---------------------------------------------------------------*/
```

Algorithm :

- Open Keil µ-Vision 4 and Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to generate Square Waveform using DAC interfacing with LPC 2148 was written and verified Successfully using Keil C.

PULSE WIDTH MODULAION USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to generate Pulse Width Modulation Waveform using LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil μ-Vision – 4 Software

Theory:

The PWM is based on the standard Timer block and inherits all of its features, although only the PWM function is pinned out on the LPC17xx. The Timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers. The PWM function is in addition to these features, and is based on match register events.

The ability to separately control rising and falling edge locations allows the PWM to be used for more applications. For instance, multi-phase motor control typically requires three non-overlapping PWM outputs with individual control of all three pulse widths and positions.

Two match registers can be used to provide a single edge controlled PWM output. One match register (PWMMR0) controls the PWM cycle rate, by resetting the count upon match. The other match register controls the PWM edge position. Additional single edge controlled PWM outputs require only one match register each, since the repetition rate is the same for all PWM outputs. Multiple single edge controlled PWM outputs will all have a rising edge at the beginning of each PWM cycle, when an PWMMR0 match occurs.

Three match registers can be used to provide a PWM output with both edges controlled. Again, the PWMMR0 match register controls the PWM cycle rate. The other match registers control the two PWM edge positions. Additional double edge controlled PWM outputs require only two match registers each, since the repetition rate is the same for all PWM outputs. With double edge controlled PWM outputs, specific match registers control the rising and falling edge of the output. This allows both positive going PWM pulses (when the rising edge occurs prior to the falling edge), and negative going PWM pulses (when the falling edge occurs prior to the rising edge)

Program :

```
#include "lpc214x.h"
//PWM ON Duration -- Change this to modify duty cycle.
#define TON 100 //Value between 0 - 900 --- 500 = 50%
#define PWM4 1<<17 //PWM4 Pin Enable bit
#define PWMMR0R 1<<1 //PWM MR0 Reset bit
#define PWMENA4 1<<12 //PWM4 Enable bit
#define EPWMM0L 1<<0 //PWM MR0 Latch Eanble bit
#define EPWMM4L 1<<4 //PWM MR4 Latch Enable bit

void clock_select(void)
{
        //Fosc = 12Mhz
        PLL0CON = 0; //Disbale PLL0 So Fosc = CCLK
   PLL0FEED=0xAA;
   PLL0FEED=0x55;
        VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 3Mhz
}
/*-------------------------------------------------------------*/
int main(void)
{
IODIR0 = 0xFFFFFFFF; //Configure Port0 as output Port
//Configure CPU pin P0.8 as PWM4
PINSEL0 = PWM4;
clock_select();     //PLL settings and PCLK settings
//Reset on PWMMR0: the PWMTC will be reset if PWMMR0 matches it.
PWMMCR = PWMMR0R;
//Selects single edge controlled mode for PWM4.
//Enable PWM4 output
PWMPCR = 0 | PWMENA4;
//Configure PWM prescaler for divide value of 3
//So output of Prescaler = PCLK / 3 = 1Mhz
PWMPR = 2; //Divide value = 3 So count=2
//Timer Clock input = 1Mhz = 1000Khz
//Select PWM Freq. as 1Khz So the Count for One Cycle = 1000
PWMMR0 = 1000; //PWM Rate 'T' Freq. 1Khz

//Ton Duration -- Change this to modify power
PWMMR4 = TON; //Ton
//PWM Latch Enable Register
//Enable MR0 & MR4 Latch bits
PWMLER = EPWMM0L | EPWMM4L ;
PWMTCR = 9; //Make high PWM Enable and Counter Enable
//Start of PWM output
while(1); //waits here --- End of Program

}
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to generate PWM waveform using LPC 2148 was written and verified Successfully.

GENERATION OF INTERRUPT IN LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to generate interrupt in LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

External Interrupts

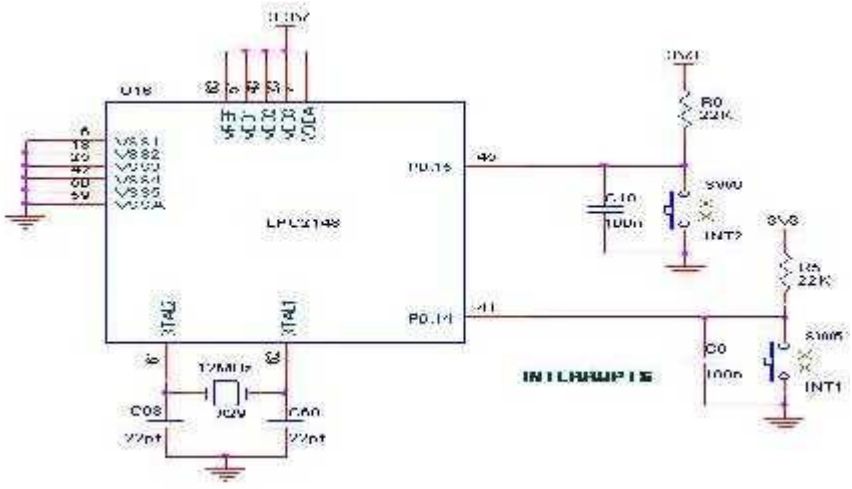
An interrupt caused by an external source such as the computer operator, external sensor or monitoring device, or another computer. Interrupts are special events that require immediate attention.

Interfacing External Interrupts with LPC2148

When an external interrupt signal is occured in LPC2148 Primer Board,the message "LOW" will be displayed on PC. The Interrupt signal is occurred by using switches. When the switch is pressed to LOW, then the external interrupt is occurred.

The Vectored Interrupt Controller (VIC) takes 32 interrupt request inputs and programmable assigns them into 3 categories, FIQ, vectored IRQ, and non-vectored IRQ. The ARM7 LPC2148 Primer board has two numbers of External Interrupts, connected with I/O Port lines (P0.14 & P0.15) as switches.

Circuit Diagram to Interface Ext-Interrupt with LPC2148



Program :

```
#include <lpc214x.h>

#define EINT2 3<<14    //Configure P0.7 as EINT2 input => PINSEL0
#define LEDS 0xFF<<8 //LED => P0.8 to P0.15

void eint2_isr(void)__attribute__((interrupt("IRQ")));

unsigned long count;

void delay(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<1007;j++) //5035 for 60Mhz ** 1007 for 12Mhz
   {;}
 }
}
/*-------------------------------------------------------*/
void eint2_isr(void)
{
EXTINT = 1<<2; //External Interrupt Flag register
count++;
if(count>255) count=0;
VICVectAddr=0xFF;
}
/*-------------------------------------------------------*/
int main(void)
{

IODIR0 = LEDS; //Configure Port0 as output Port
PINSEL0 = EINT2; //Configure Port0.7 as EINT2 Input
PINSEL1 = 0;

VICIntEnable = 1 << 16; //Enable EINT2 interrupt
VICIntSelect = 0; //Interrupt request assigned to the IRQ category
EXTMODE = 1<<2;
EXTPOLAR = 0;
VICVectAddr0 = (unsigned long) eint2_isr;
VICVectCntl0 =0x20 | 16;

count=0;

while(1){
        IOSET0 = LEDS; //Switch OFF all LEDS
        IOCLR0 = count << 8; //Set VAlue
        delay(1000);
       }
}
/*-------------------------------------------------------*/
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to generate Interrupt in LPC 2148 was written and verified Successfully using Keil C.

EEPROM INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to interface EEPROM with LPC 2148 Controller.

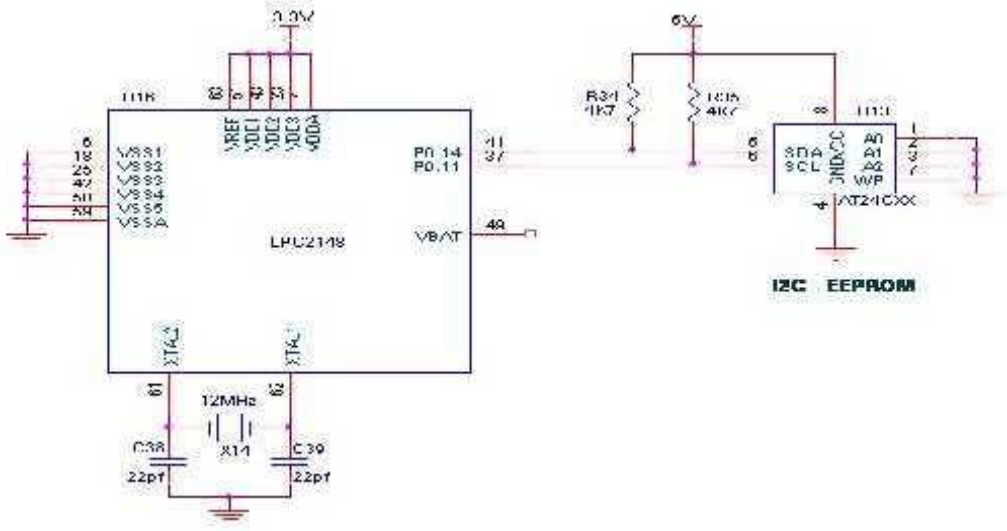
Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

The EEPROM is used to store the information from the registers. It has no direct connection to the other parts of the device besides the registers. The intention of the EEPROM is to store device settings so they get not lost when the device is powered down. The information stored in the EEPROM is loaded back into the registers on power up. This organization of the memory enables the device to configure itself in a application where no I2C bus is present. It also enables a quick loading of a standard configuration at power up. The amount of write and read cycles to the EEPROM is typically greater then 1000. An internal low frequency clock generator is used to transfer data to the EERPOM. This ensures that the EEPROM can be accessed also if no clock reference (crystal or LVCMOS input) is active.

Circuit Diagram to Interface I2C–EEPROM with LPC2148 :



Program :

```
#include <LPC214X.H>
/*---- I2C0 Signals ----------*/
#define SDA0 1<<6
#define SCL0 1<<4
/* ---I2C0 Signal Declaration */
#define I2EN 1<<6
#define STA  1<<5
#define STO  1<<4
#define SI  1<<3
#define AA  1<<2
//Function declaration
int main(void);
void i2c_init();
void i2c_word_write( unsigned char slave_addr, unsigned char word_addr, unsigned char data );
unsigned char i2c_word_read( unsigned char slave_addr, unsigned char word_addr );
void i2c_write(unsigned char addr, unsigned char dat);
void disp_value(unsigned char value);
void delay(unsigned int k);
//Seven Segment Code
unsigned char seg_dat[]={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x98};
int main()
{
unsigned char count;
//Initialize I2C
i2c_init();
//Load Count value from EEPROM
count = i2c_word_read(0xA0, 0);
//Limit the value
if(count>9) count=0;

while(1)
 { disp_value(count); //Display Count
   i2c_word_write(0xA0, 0 , count); //Store count
   delay(2000);       //2 Sec Delay
   count++;          //increment count
   //Limit the value
   if(count>9) count=0;
  }
return 0;
}
/*-------------------------------------------------------------*/
void i2c_init()
{
//Enable SDA0 & SCL0 Pins
PINSEL0 = 0 | SDA0 | SCL0;
//Set I2C0 Clock rate to 100Khz
I2C0SCLH = 200; I2C0SCLL = 200;
//Enable I2C
I2C0CONSET = I2EN;
}
```

```
/*---------------------------------------------------------*/
void i2c_word_write( unsigned char slave_addr,
                     unsigned char word_addr,
                     unsigned char data )
{
unsigned char sts;

//Transmit Start Condition
 I2C0CONSET = STA;
 //Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
 I2C0CONCLR = STA; //Clear STA
//Transmit Slave Address
 I2C0DAT = slave_addr; //Slave Addr.+W
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//Transmit Slave Word Address
 I2C0DAT = word_addr; //Slave Word address
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//--------- Write Data
 I2C0DAT = data; //data
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//--------- End of write
//Stop Bit
 I2C0CONSET = STO | AA; //Stop & Ack bit
 I2C0CONCLR = SI; //Clear SI flag
 delay(15);
 I2C0CONCLR = SI; //Clear SI flag
}
/*---------------------------------------------------------*/
unsigned char i2c_word_read( unsigned char slave_addr, unsigned char word_addr )
{
  unsigned char sts;
  unsigned char data;
```

```
//Transmit Start Condition
 I2C0CONSET = STA;
 //Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
 I2C0CONCLR = STA; //Clear STA

//Transmit Slave Address
 I2C0DAT = slave_addr; //Slave Addr.+W
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//Transmit Slave Word Address
 I2C0DAT = word_addr; //Slave Word address
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);

//Stop Bit
 I2C0CONSET = STO | AA; //Stop & Ack bit
 I2C0CONCLR = SI; //Clear SI flag
 delay(2);
 I2C0CONCLR = SI; //Clear SI flag
/*-------------------------------------------------------*/
//Repeat Start Condition
 I2C0CONSET = STA;
 //Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
 I2C0CONCLR = STA; //Clear STA
//Transmit Slave Address (Read)
 I2C0DAT = slave_addr | 1; //Slave Addr.+R
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
/*------Data Read ------*/
  I2C0CONSET = AA; //Ack bit
  I2C0CONCLR = SI; //Clear SI flag
   //Wait for rx to complete
```

```
  do{
      sts = I2C0CONSET & SI;
      }while(sts!=SI);
    data = I2C0DAT; //read data
/** Send Data with NOT ACK **/
  I2C0CONCLR = SI | AA; //Clear SI flag
   //Wait for rx to complete
  do{
      sts = I2C0CONSET & SI;
      }while(sts!=SI);
//Stop Bit
 I2C0CONSET = STO | AA; //Stop & Ack bit
 I2C0CONCLR = SI; //Clear SI flag
 delay(2);
 I2C0CONCLR = SI; //Clear SI flag
 return(data);
}
/*-------------------------------------------------------------*/
void i2c_write(unsigned char addr, unsigned char dat)
{
unsigned char sts;
//Transmit Start Condition
 I2C0CONSET = STA;
 //Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
 I2C0CONCLR = STA; //Clear STA
//Transmit Slave Address
 I2C0DAT = addr; //Slave Addr.+W
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//Transmit data
 I2C0DAT = dat; //data
 I2C0CONSET = AA; //Ack bit
 I2C0CONCLR = SI; //Clear SI flag
//Wait for tansmit to complete
do{
   sts = I2C0CONSET & SI;
   }while(sts!=SI);
//Stop Bit
 I2C0CONSET = STO | AA; //Stop & Ack bit
 I2C0CONCLR = SI; //Clear SI flag
 delay(2);
 I2C0CONCLR = SI; //Clear SI flag
}
/*-------------------------------------------------------------*/
```

```
void disp_value(unsigned char value)
{
//Convert as 7Seg. code and Display in DS3 & DS4
i2c_write(0x70, seg_dat[value]); //PCF8574A => 0x70, PCF8574P => 0x40
}
/*-------------------------------------------------------*/
//Delay Program
//Input - delay value in milli seconds
void delay(unsigned int k)
{
        unsigned int i,j;
        for (j=0; j<k; j++)
                for(i = 0; i<=5035; i++);
}
/*-------------------------------------------------------*/
```

Algorithm :

- Open Keil µ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to interface EEPROM with LPC 2148 was written and verified Successfully using Keil C.

REAL TIME CLOCK INTERFACING USING LPC 2148

Ex.No :

Date :

Aim :

To write a program in Keil – C to display the time on LCD using RTC interfacing with LPC 2148 Controller.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

RTC (Real Time Clock)

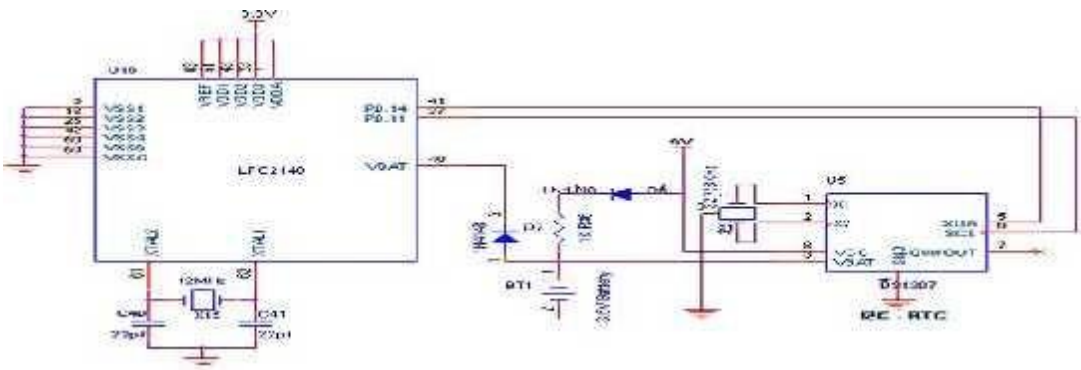
The DS1307 Serial Real-Time Clock is a low-power; full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator.

Interfacing I2C – RTC with LPC2148

Read date & time by using I2C - RTC in LPC2148 Primer Board. Wiring up an I2C based RTC to the I2C port is relatively simple. The RTC also makes the software easier as it takes care of all calendar functions; accounting for leap years etc. The DS1307 (RTC) Real Time Clock IC (an I2C real time clock) is an 8 pin device using an I2C interface.

In **LPC2148 Primer Kit,** 2 nos. of RTC lines are controlled by I2C Enabled drivers. I2C Lines serial clock **SCL (P0.2),** serial data **SDA (P0.3)** connected to the I2C based serial RTC ds1307 IC. The date & times are read in LPC2148 Primer Kit by using these SDA & SCL I2C lines.

Circuit Diagram to Interface I2C–RTC with LPC2148



Program:

```
#include <LPC214X.H>
/*---LCD Signal Declaration---*/

#define LCD_RS 1<<24 //Port1.24
#define LCD_RW 1<<16 //Port0.16
#define LCD_EN 1<<17 //Port0.17
#define LCD_DATA 0xFF<<16 //Port1.16 to 1.23
#define LCD_STS 1<<23 //Port1.23
/*--- RTC Signal Declaration */
#define CLKEN 1
#define CLKSRC 1<<4
//Function declaration
int main(void);
void lcdini(void);
void lsts(void);
void lcdctl(unsigned char val1);
void lputc(unsigned char lcr);
void delay (unsigned int k);
void rtc_init(void);
void t_disp(void);
void t_set(unsigned char shrs, unsigned char smin, unsigned char ssec);
/*----Main Program -----*/
int main(void)
{
 IODIR0 = LCD_RW | LCD_EN; //configure LCD r/w & EN as o/p
 IODIR1 = LCD_RS; //Configure RS o/p
 PINSEL0 = 0; //Configure Port0 as General Purpose IO
 PINSEL1 = 0;
 IOCLR0=LCD_EN; IOCLR1=LCD_RS; IOSET0=LCD_RW; /* DISABLE LCD TEMPORALY*/
 delay(1000); //LCD Power-up Delay
 lcdini(); //Initialise LCD
 rtc_init(); //Initialise RTC
 //t_set(11,15, 0); //time Set function
do
 { /* LCD - Return Home Command */
          lsts(); //Check LCD ready?
          lcdctl(2); //Send CMD
          lsts();    //Check LCD ready?

         //Display time
           t_disp();
  }while(1);
}
/*  ----------------------------------------------------------------*/
void lcdini()
{
 lcdctl(0x38); /* Function Set 2 LINE 5 X 8 CHAR*/
 delay(5);                          /* Waits for 5 Msec. */
 lcdctl(0x38); /* Sends Function Set - AGAIN */
```

```
  delay(5);                          /* Waits for 5 Msec. */
  lcdctl(0x38); /* Sends Function Set - AGAIN */
  lsts();                            /* Wait Till BUSY=0 */
  lcdctl(0x38); /* Sends Function Set - AGAIN*/
  lsts();                            /* Wait Till BUSY=0 */
  lcdctl(0x04); /* Display off */
  lsts();                            /* Wait Till BUSY=0 */
  lcdctl(0x01); /* Clear Display */
  lsts();                            /* Wait Till BUSY=0 */
  lcdctl(0x06); /* Set Entry mode */
  lsts();                            /* Wait Till BUSY=0 */
  lcdctl(0x0c); /* Set Display ON */
 }
/******* Checks the LCD Status for busy***************/
void lsts()
{
unsigned long int tp1;
 IOCLR1=LCD_RS; IOSET0=LCD_RW;
do{
  IOSET0=LCD_EN;
  tp1 = IOPIN1 & LCD_STS;
  IOCLR0=LCD_EN;
  }while(tp1);
 IOCLR0=LCD_EN;
 IOCLR0=LCD_RW;
}
/*-------------------------------------------------------------*/
void lcdctl(unsigned char val1)
{ unsigned long int dat;
 dat = ((unsigned long int)val1) << 16;
IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 /* WRITE COMMAND TO CONTROL REGISTER*/
 IOCLR1 = LCD_RS;
 IOCLR0 = LCD_RW;
 IOCLR1 = LCD_DATA;
 IOSET1 = dat;
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
/******* Displays a Character in the LCD ******/
void lputc(unsigned char lcr)
{ unsigned long int dat;
 lsts();
 dat = ((unsigned long int)lcr)<<16;
 IODIR1 = LCD_RS | LCD_DATA; //Configure RS & DATA as o/p
 IOSET1=LCD_RS; IOCLR0=LCD_RW;
 IOCLR1 = LCD_DATA;
 IOSET1 = dat;
```

```
 IOSET0=LCD_EN;
 IOCLR0=LCD_EN;
 IOSET0=LCD_RW;
 IODIR1 = LCD_RS; //Configure RS as o/p & DATA as i/p
 }
//Delay Program
void delay(unsigned int k)
{
        unsigned int i,j;
        for (j=0; j<k; j++)
                for(i = 0; i<=800; i++);
}
//RTC Initialization Program
void rtc_init()
{
//Enable Clock and Select Clock Source 32Khz Ext. Crystal
CCR = CLKSRC | CLKEN;
CIIR = 0; //Disble Count interrupt
//Alarm Mask Register
AMR = 0xff; //Disable alarm
}
/*-------------------------------------------------------------*/
void t_disp()
{unsigned char ten, one;
 unsigned char tp1;
//Display HRS
tp1 = HOUR & 0x1F;
ten = (tp1/10)+'0';
one = (tp1%10)+'0';
lputc(ten); lputc(one); lputc(':');

//Display Min.
tp1 = MIN & 0x3F;
ten = (tp1/10)+'0';
one = (tp1%10)+'0';
lputc(ten); lputc(one); lputc(':');
//Display Sec.
tp1 = SEC & 0x3F;
ten = (tp1/10)+'0';
one = (tp1%10)+'0';
lputc(ten); lputc(one);
}
/*-------------------------------------------------------------*/
void t_set(unsigned char shrs, unsigned char smin, unsigned char ssec)
{
HOUR = shrs & 0x1F;
MIN = smin & 0x3F;
SEC = ssec & 0x3F;
}
/*-------------------------------------------------------------*/
```

Algorithm :

- Open Keil μ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to display the time on LCD using RTC interfacing with LPC 2148 was written and verified Successfully using Keil C.

COMMUNICATION BETWEEN TWO LPC 2148 USING ZIGBEE

Ex.No :

Date :

Aim :

To write a program in Keil – C to Communicate with two LPC 2148 Controller using Zigbee.

Components Required :

- LPC 2148 Kit with LCD interfacing
- Zigbee Transceiver
- Personal Computer
- Keil µ-Vision – 4 Software

Theory :

ZigBee

ZigBee is a specification for a suite of high level communication protocols using small, low- power digital radios based on an IEEE 802 standard for personal area networks.

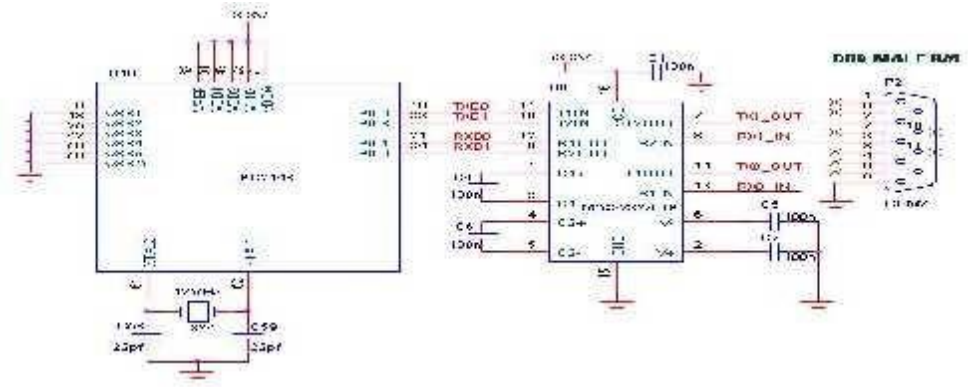
Digi ZigBee

The Digi Xbee 802.15.4 modules are the easiest to use, most reliable and cost-effective RF devices we've experienced. The 802.15.4 Xbee modules provide two friendly modes of communication - a simple serial method of transmit/receive or a framed mode providing advanced features. These modules can communicate point to point, from one point to a PC, or in a mesh network.

Interfacing Zigbee with LPC2148

Interfacing ZigBee module with LPC2148 Primer Board for used for controlling application through UART0. The data communication is done in internet by using the ZigBee module through MAX232 into the SBUF register of LPC2148 microcontroller. The serial data from the Zigbee receiver is taken by using the Serial Interrupt of the controller. +5V and ground is connected to provide power to the module. While TX and RX pin is connected for communication.

Circuit Diagram to Interface Zigbee with LPC2148



Program : Zigbee – Remote (Transmitter)

```
#include <lpc214x.h>
//Switch Port
#define KEY (IOPIN0 & 1<<7 ) //Key at P0.7
unsigned char const seg_dat[]={0x3F, 0x6, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x7, 0x7F, 0x67};
#define DS3   1<<13                // P0.13
#define DS4   1<<12                // P0.12
#define SEG_CODE 0xFF<<16     // Segment Data from P0.16 to P0.23
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
//UART1 Values
//Buad rate Divide value - 9600 BPS
#define MULVAL 10<<4
#define DIVADDVAL 6
#define UDIV 61
#define DLAB 1<<7
#define TXD1 1<<16
#define RXD1 1<<18
char tr_buf[] = { 0x55, 0x55, 0x55, //Sync Pulses
                0xAA, 0x55, 0 }; //Start Sequence
/*-------------------------------------------------------------*/
void delay_ms(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<5035;j++) //5035 for 60Mhz ** 1007 for 12Mhz
    {;}
 }
}
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz
void clock_select(void)
{
//Fosc = 12Mhz
//Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;

PLL0CON = 3; //Enable PLL0
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
/*  Function To Configure UART1 to  */ /*  1200 Baud Rate No parity Data=8bits    */
void UART1_Init(void)
{
//Pin Function Select UART1 RXD/TXD
```

```
 PINSEL0 |= RXD1 | TXD1 ;
 U1FCR = 0; // Disable FIFO's
 U1LCR = 3 | DLAB; // 8N1, enable Divisor latch bit DLAB=1
 // PCLK = 15.000.000 MHz
 U1FDR = MULVAL | DIVADDVAL;
 U1DLL = UDIV; // baud rate fixed to 9600 @ PCLK = 15 Mhz
 U1DLM = 0;
 U1LCR = 3; // Disable Divisor latch bit DLAB=0
}
/*-------------------------------------------------------------*/
/* Function to send one Character to UART #1 */
void uart_send(unsigned char tr)
{
//Wait for Previous data Transmit to complete
  while(!(U1LSR & 0x20));
      U1THR = tr; //Send char to UART1
}
/*-------------------------------------------------------------*/
/* Function Transmits String to UART #1 */
void uart_str(char *text)
{
   while(*text)
    {
      //Wait for Previous data Transmit to complete
      while(!(U1LSR & 0x20));
      U1THR = *text++; //Send to UART1
            }
}
/*-------------------------------------------------------------*/
int main (void)
{ unsigned char count;
 //Configure Port 0 & 1 as General Purose IO
 PINSEL0 = 0;
 PINSEL1 = 0;
 PINSEL2 = 0;
//Configure Segement data & Select signal as output
 IODIR0 = SEG_CODE | DS3 | DS4;
 IODIR1 = 0; //Port 1 as input
 IOSET0 = SEG_CODE | DS3 ; //Disable DS3 display
 IOCLR0 = DS4;  //Enable DS4 Display
 clock_select(); //Set CPU Clock
   UART1_Init(); //Configure UART #1
   delay_ms(1000);
   count = 0; //Initial value of count
   while (1)
     {
       //Display Count value
      IOCLR0 = SEG_CODE;
      IOSET0 = seg_dat[count]<<16;

      uart_str(tr_buf); //Transmit Sync pulse & Start Seq.
```

```
    uart_send(count);

    while(KEY);          //Wait for Key Press
    while(!KEY);         //Wait for Key Release

    count++;             //Increment Count Value
     if(count>=10) count=0; //Limit to 9

     uart_str(tr_buf); //Transmit Sync pulse & Start Seq.
     uart_send(count);
    }
}
/*-------------------------------------------------------------------------*/
```

Program : Base (Receiver)

```
#include <lpc214x.h>
unsigned char const seg_dat[]={0x3F, 0x6, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x7, 0x7F, 0x67};
#define DS3   1<<13              // P0.13
#define DS4   1<<12              // P0.12
#define SEG_CODE 0xFF<<16     // Segment Data from P0.16 to P0.23
//PLL Register Values
#define MSEL 4     //M-1 since M=5
#define PSEL 1<<5 //P-1 since P=2
//Buad rate Divide value - 9600 BPS
#define MULVAL 10<<4
#define DIVADDVAL 6
#define UDIV 61
#define DLAB 1<<7
#define TXD1 1<<16
#define RXD1 1<<18
unsigned char srdy; //UART #1 Data Index
unsigned char count; //Value Received from Tr
unsigned char frmrdy; //Data Ready Flag
/*-------------------------------------------------------------*/
void delay_ms(int n)
{
int i,j;
for(i=0;i<n;i++)
 {for(j=0;j<5035;j++) //5035 for 60Mhz ** 1007 for 12Mhz
   {;}
 }
}
/* Function Sets PLL0 So CPU Clock=60Mhz PCLK=15Mhz */
void clock_select(void)
{
//Fosc = 12Mhz //Select CCLK = 60Mhz & Fcco = 240Mhz
PLL0CFG = PSEL | MSEL;
//PLL FEED
PLL0FEED=0xAA;
```

```
PLL0FEED=0x55;

PLL0CON = 3; //Enable PLL0
//PLL FEED
PLL0FEED=0xAA;
PLL0FEED=0x55;
VPBDIV = 0; //PCLK = CCLK/4 So PCLK = 15Mhz
}
// Serial Port Interrupt Service Routine
/*-------------------------------------------------------------------*/
__irq void serial_isr(void)
{
 unsigned char sdat;

   sdat = U1RBR;

  switch(srdy)
  {
    case 0: if(sdat==0xAA) srdy=1; //Check Start Seq. Data #1
            break;

    case 1: if(sdat==0x55) srdy=2; //Check Start Seq. Data #2
            else srdy=0; //Set index = 0
            break;

    case 2: count = sdat; //Read Count
            frmrdy=1;       //Set Data Ready Flag
            srdy=0;         // Set Index = 0 to get next data
            break;

    default: srdy=0;
            break;
   }
VICVectAddr=0xFF;
}
/*  Function To Configure UART1 to      */ /*  1200 Baud Rate No parity Data=8bits                */
void UART1_Init(void)
{
//Pin Function Select UART1 RXD/TXD
 PINSEL0 |= RXD1 | TXD1 ;

 U1FCR = 0; // Disable FIFO's
 U1LCR = 3 | DLAB; // 8N1, enable Divisor latch bit DLAB=1

 // PCLK = 15.000.000 MHz
 U1FDR = MULVAL | DIVADDVAL;
 U1DLL = UDIV; // baud rate fixed to 9600 @ PCLK = 15 Mhz
 U1DLM = 0;
 U1LCR = 3; // Disable Divisor latch bit DLAB=0
/** UART1 Interrupt Configuration **/
 U1IER = 1; //Enable the Rxd interrupts.
```

```
VICIntEnable = 1 << 7; //Enable UART1 interrupt
VICIntSelect = 0; //Interrupt request assigned to the IRQ category
VICVectAddr0 = (unsigned long) serial_isr; //IRQ0 Address
VICVectCntl0 =0x20 | 7; //Assign IRQ0
}
/*-------------------------------------------------------------*/
/* Function to send one Character to UART #1 */
void uart_send (unsigned char tr)
{
//Wait for Previous data Transmit to complete
  while(!(U1LSR & 0x20));
      U1THR = tr; //Send char to UART1
}
/*-------------------------------------------------------------*/
/* Function Transmits String to UART #1 */
void uart_str(char *text)
{
   while(*text)
    {
      //Wait for Previous data Transmit to complete
      while(!(U1LSR & 0x20));
      U1THR = *text++; //Send to UART1
            }
}
/*-------------------------------------------------------------*/
int main (void)
{
  //Configure Port 0 & 1 as General Purose IO
 PINSEL0 = 0;
 PINSEL1 = 0;
 PINSEL2 = 0;

//Configure Segement data & Select signal as output
 IODIR0 = SEG_CODE | DS3 | DS4;
 IODIR1 = 0; //Port 1 as input

 IOSET0 = SEG_CODE | DS3 ; //Disable DS3 display
 IOCLR0 = DS4;             //Enable DS4 Display

  frmrdy=0; srdy=0; //Initial value of index & Flag

   clock_select(); //Set CPU Clock
   UART1_Init(); //Configure UART #1

   delay_ms(1000);
   count = 0;  //Initial value
   //Display Count value
   IOCLR0 = SEG_CODE;
   IOSET0 = seg_dat[count]<<16;
   while (1)
     {
```

```
      if(frmrdy==1) //Count value Received
      {
       frmrdy = 0;         //Clear Rx. Flag

       //Display Count
       value IOCLR0 =
       SEG_CODE;
       IOSET0 = seg_dat[count]<<16;
      }
     }
}
/*-------------------------------------------------------------------*/
```

Algorithm :

- Open Keil µ-Vision 4
- Create a Project and Choose the Hardware tool
- Create a file , type the Program and Save it as .c format
- Add the files to Source Group
- Run the program and get the error free output.
- Connect hardware to the PC and Download the Program using Flash Magic
- Verify the output.

Result :

Thus the Program to make Communication between two LPC 2148 Controller using Zigbee was written and verified Successfully using Keil C.

MAILBOX USING KEIL C SOFTWARE

Ex.No:

Date:

Aim:

To develop a ‘C’code to create a mailbox and to understand the RTOS functions

Apparatus & Software Required:

1. LPC2148 Development board.
2. KeilµVision5software.
3. Flash Magic.
4. USB cable

Theory :

Real-time and embedded systems operate in constrained environments in which computer memory and processing power are limited. They often need to provide their services within strict time deadlines to their users and to the surrounding world. It is these memory, speed and timing constraints that dictate the use of real-time operating systems in embedded software. The "kernel" of a real-time operating system ("RTOS") provides an "abstraction layer" that hides from application software the hardware details of the processor (or set of processors) up on which the application software will run.

In providing this "abstraction layer" the RTOS kernel supplies five main categories of basic services to application software The most basic category of kernel services is Task Management. This set of services allows application software developers to design their software as a number of separate "chunks" of software -- each handling a distinct topic, a distinct goal, and perhaps its own real-time deadline. Each separate "chunk" of software is called a "task." The main RTOS service in this category is the scheduling of tasks as the embedded system is in operation.

The second category of kernel services is Inter task Communication and Synchronization. These services make it possible for tasks to pass information from one to another, without danger of that information ever being damaged. They also make it possible for tasks to coordinate, so that they can productively cooperate with one another. Without the help of these RTOS services, tasks might well communicate corrupted information or otherwise interfere with each other.

Since many embedded systems have stringent timing requirements, most RTOS kernels also provide some basic Timer services, such as task delays and time-outs. Many (but not all) RTOS kernels provide Dynamic Memory Allocation services. This category of services allows tasks to "borrow" chunks of RAM memory for temporary use in application software. Often these chunks of memory are then passed from task to task, as a means of quickly communicating large amounts of data between tasks. Some very small RTOS kernels that are intended for tightly memory-limited environments, do not offer Dynamic memory allocation.

PROGRAM:

```
/*********************************************************************************/
/* RTOs Mailbox Example                                                          */
/*********************************************************************************/

#include "LPC214x.H"   /* LPC21xx definitions */
#include <ucos.h>     /* OS Header File */

#define  TASK_STK_SIZE      100   /* Size of each task's stacks (# of WORDs) */
#define  NO_TASKS           3     /* Number of identical tasks */

OS_STK   TaskStk[NO_TASKS][TASK_STK_SIZE];     /* Tasks stacks */
OS_STK   TaskStartStk[TASK_STK_SIZE];
char     TaskData[NO_TASKS];                   /* Parameters to pass to each task */
OS_EVENT *mbox;

void init_timer (void); void
UART1_Init(void);  void
SendString(char *text);

/* Local functions */
void Task2 (void *data)
{
        char   txmsg[] = "\r\n Msg from Task-2";

        data = data;                    /* Prevent compiler warning */

        while(1)
        {
                OSTimeDly(5);
                SendString("\r\n Task2");
     OSMboxPost(mbox,  (void *)&txmsg);  /* Send message to Task #1          */
        }
}
/*--------------------------------------------------------------------*/
void Task1 (void *data)
{
        U8 err;
        char  *rxmsg;

        rxmsg = (char *)OSMboxPend(mbox,  0, &err);    /* Wait for message from Task #2 */

        while(1)
        {
          SendString("\r\n Task1");
          rxmsg = (char *)OSMboxPend(mbox,  0, &err);  /* Wait for message from Task #2 */
          SendString(rxmsg);
          OSTimeDly(5);
        }
}
/*--------------------------------------------------------------------*/
void TaskStart (void *data)
{
```

```
        init_timer();
   UART1_Init();

        mbox  = (OS_EVENT *) OSMboxCreate((void  *)0);                  /* Create 2 message
mailboxes              */

        TaskData[1] = 1;        /* Each task will display its own letter */
        TaskCreate(Task1,  (void *)&TaskData[1],  "Task1", 1);

        TaskData[2] = 1;        /* Each task will display its own letter */
        TaskCreate(Task2,  (void *)&TaskData[2],  "Task2", 2);

        while(1)
        {
                OSTimeDly(1500);      /* Wait 1.5 second */
        }
}
/*-----------------------------------------------------------------*/
int main (void)
{

        OSInit();       /* Initialize uC/OS-II                              */
        TaskCreate(TaskStart,  (void *)0, "Task Start", 0);
        OSStart();    /* Start multitasking  */

        return 0;                    // Actually we should never come here
}
/* ********************************************************************** */
void UART1_Init(void)
{
 PLL0CON = 0;
 PLL0FEED=0xAA;
 PLL0FEED=0x55;
 VPBDIV = 1;
 // Fpclk = 12.000.000 MHz
 // DLM,DLH = Fpclk / (19200*16) = 39 = 0x27
 PINSEL0 |= 0x00050000; // Select UART1 RXD/TXD
 U1FCR = 0; // Disable FIFO's
 U1LCR = 0x83; // 8N1, enable Divisor latch bit
 U1DLL = 0x27; // baud rate fixed to 19200 @ PCLK = 12 Mhz
 U1DLM = 0;
 U1LCR = 3; // Disable Divisor latch bit
}
/*-----------------------------------------------------------------*/
void SendString(char *text)
{
  while(*text)
   {
     while(!(U1LSR & 0x20));//Wait for Tranmit to complete
     U1THR = *text++;   //Send to UART1
        }
}
/*-----------------------------------------------------------------*/
```

Algorithm :

1. Follow the steps to create a New project
2. Type the below code and save it with the name (anyname.c)
3. Follow the steps to create a New Project to compile and build the program
4. Follow the procedures in to download your Hex code to processor using Flash Magic Software.

Result:

The C-Language program to create a mailbox and to understand the about the RTOS functions is developed and is verified.