**Laboratory Manual**

**Semester: VI**

**Course Code: ECL68**

**Course Name: Embedded System Design Lab**

**Ramaiah Institute of Technology**

Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

Department of Electronics and Communication Engineering

VI Semester Microcontroller Lab (ECL68)

**Embedded C programming**

1. Bit manipulation
2. Device driver for reading from stdin (keyboard) and writing to stdout (monitor) using system calls

# RTOS Programs (System level programming by Linux API)

1. Creation of processes using fork()
2. Usage of “Signal” function calls–when DEL key or CTRL C is pressed, this sends a signal for abrupt termination
3. Multithreading – One thread reads the input from the keyboard and another thread converts to uppercase. This is done until ‘Stop‟ is pressed. Number of threads can be running sharing same CPU.
4. Intertask communication using semaphore and pipes – Two threads, one for reading the input and one for converting the text to upper case letters, converting thread will wait for a semaphore to be released before it starts the operation and also pipes can be used to share the data from one thread toanother

# Interfacing programs

1. Familiarize I/O ports of a controller – on/off control of LEDs using switches.
2. Display a given string using the LCD display interface.
3. Interface keypad and display the key pressed on 7 segment LED.
4. Waveform generation using the internal DAC of LPC2148
5. Design and display a two-digit counter

**COURSE DESIGN, DELIVERY AND ASSESSMENT**

|  |  |
| --- | --- |
| Course code and Title : ECL68  Embedded System Design Lab | Course Credits :0:0:1 |
| CIE : **50 Marks** | SEE : **50 Marks** |
| Total No of Theory / Tutorial / Lab Hours : 14 | |

**Prerequisites**

|  |
| --- |
| **Prerequisite Courses with codes : Microprocessor Lab ECL48** |

**LIST OF EXPERIMENTS**

**Part A: Embedded C programming**

1. Bit manipulation
2. Calculation of Cyclic Redundancy Code
3. Device driver for reading from stdin (keyboard) and writing to stdout (monitor) using system calls

# Part B: RTOS Programs (System level programming by Linux API)

1. Creation of processes using fork()
2. Usage of “Signal‟ function calls – when DEL key or CTRL C is pressed, this sends a signal for abrupt termination
3. Multithreading – One thread reads the input from the keyboard and another thread converts to upper case. This is done until „Stop‟ is pressed. Number of threads can be running sharing same CPU.
4. Intertask communication using semaphore and pipes – Two threads, one for reading the input and one for converting the text to upper case letters, converting thread will wait for a semaphore to be released before it starts the operation and also pipes can be used to share the data from one thread to another

# Part C: Interfacing programs

1. Familiarize I/O ports of a controller – on/off control of LEDs using switches.
2. Display a given string using the LCD display interface.
3. Interface keypad and display the key pressed on 7 segment LED.
4. Waveform generation using the internal DAC of LPC2148
5. Design and display a two-digit counter

# Textbooks:

1. Dr. K. V. K. K. Prasad, “Embedded Real-Time Systems: Concepts, Design & Programming”, Reprint Edition, Dreamtech Press, 2013.
2. Shibu K. V, “Introduction to Embedded Systems”, 2nd Edition, Tata McGraw Hill Education, 2017.
3. James K. Peckol, “Embedded Systems – A Contemporary Design Tool”, Student Edition, John Wiley and Sons, 2014.

# References:

1. Steve Heath, “Embedded System Design”, 2nd Edition, Newnes Publishers, 2003.
2. LPC 2148 user manual.

# Course Outcome (COs):

1. Develop embedded C programs (POs – 1, 2, 5, 9, 10, 12, PSO – 2)
2. Demonstrate embedded C programs to create process/tasks and threads for RTOS (POs – 1, 2, 5, 9, 10, 12, PSO – 2)
3. Illustrate inter-task communication using embedded C programs (POs – 1, 2, 5, 9, 10, 12, PSO – 2)
4. Design embedded C programs to interface data converters with a microcontroller (POs – 1, 2, 3, 5, 9, 10, 12, PSO – 2)
5. Interface different types of I/O peripherals using a microcontroller for a typical application (POs

– 1, 2, 3, 5, 9, 10, 12, PSO – 2)

**Concept Map:**

**Embedded System Design**

**Deals With**

**Deals With**

**Involves**

**Deals With**

**Deals With**

**Application Programs**

**inter-task communication using embedded C programs**

**7**

**Branch/Loop** instruction

**Embedded C Programs**

**Embedded C programs to create process/tasks and threads for RTOS**

**Interfacing Experiments**

1.1 Programs involving Bit manipulation

1.2 Calculation of cyclic redundancy code

1.3 Device driver for reading from k/b & writing to monitor using system calls

4.1 LED control

4.2 LCD Display

4.3 Keypad interface

4.4 Generation of a waveform using DAC

4.5 Interfacing the stepper motor

4.6 Design & display 2 digit counter

3.1 Programs involving intertask communication using semaphores & pipes

2.1 Creation of Processes using fork()

2.2 Usage of signal function calls

2.3 Program on multithreading

5.1 Develop & check memory profiling of serial port driver for LPC2148

5.2 Implement FIR filter on LPC2148

**Introduction to Software IAR embedded workbench**.

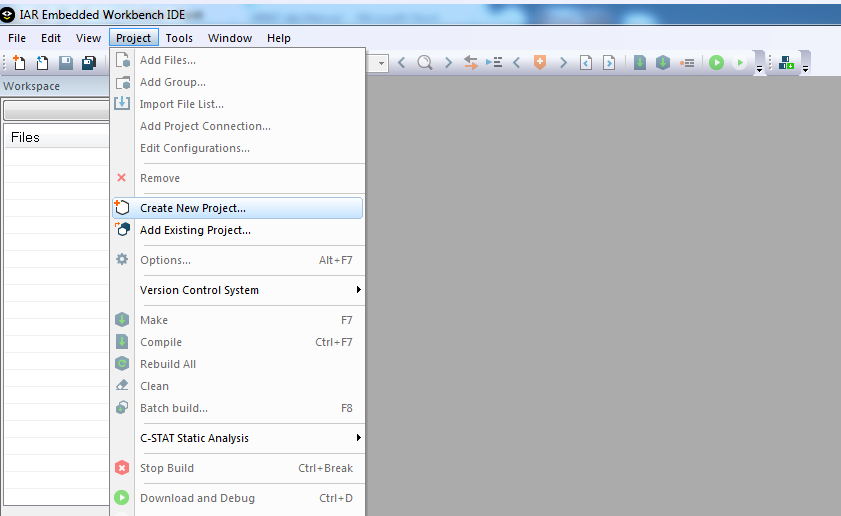
**CREATION OF NEW PROJECT and SIMULATING:**

The **IAR** compiler is used to create,compile& simulate the projects for ARM family micro-controllers. The procedure to create projects is as follows

Step 1:Create a folder with the name related to the project in any drive.

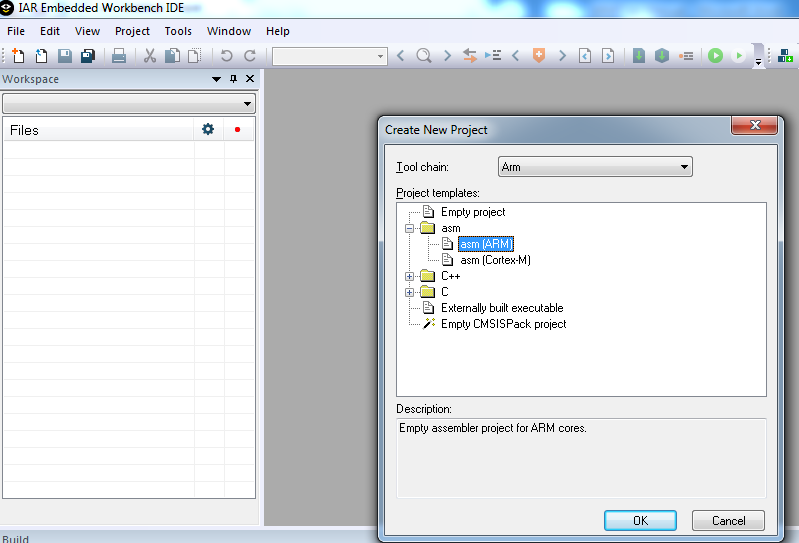
Step 2: Open **IAR Embedded Workbench** software. A new workspace launcher will open.

Step 3: Go to **Project** click “**CreateNewProject”.**

****

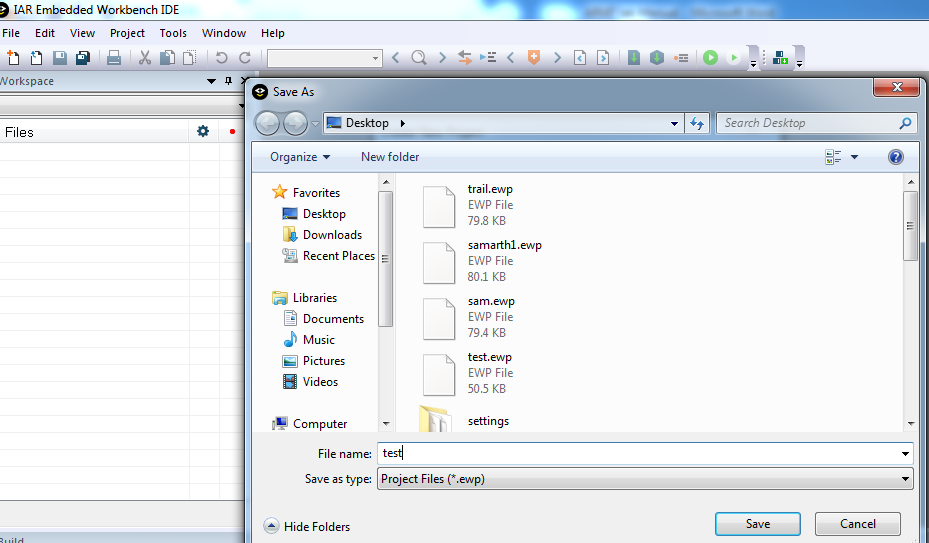
Step 4: Tool chain:Select**ARM**

Project templates : select **asm (ARM)**  if assembly code you would like write, else you would like to write **C** then select **main**

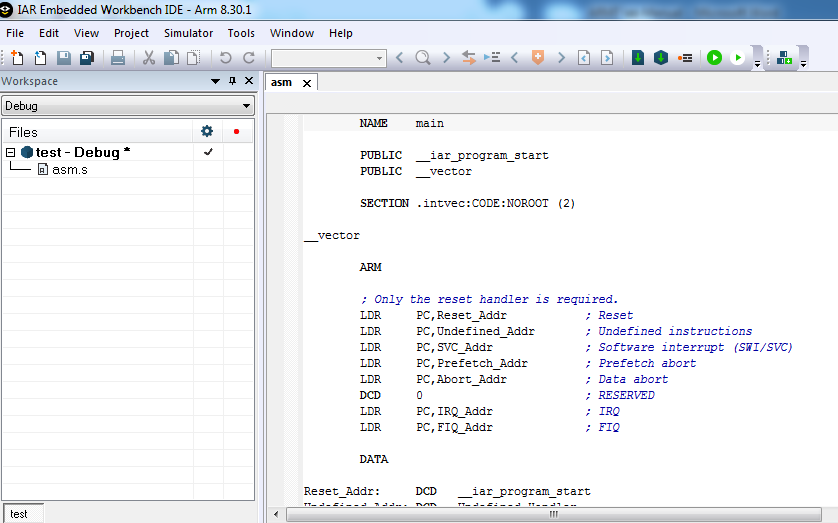


Step 5: select **asm** and click **OK.**

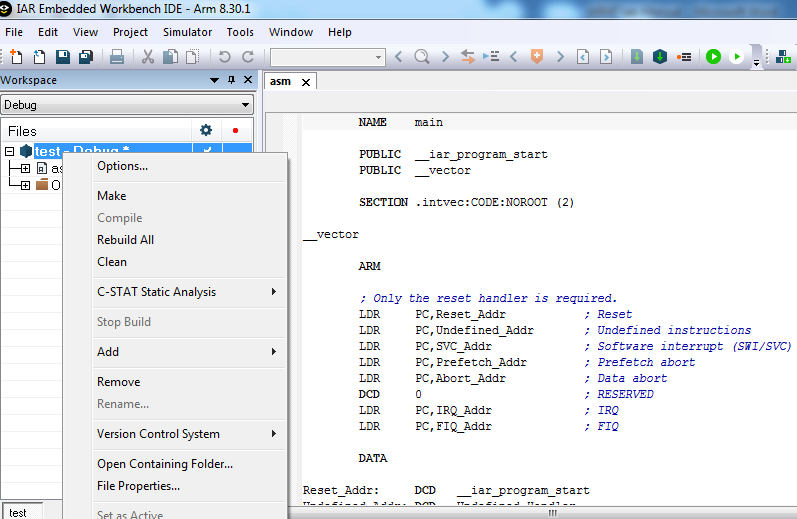
Step 6: Give project name and save the file name in corresponding folder.



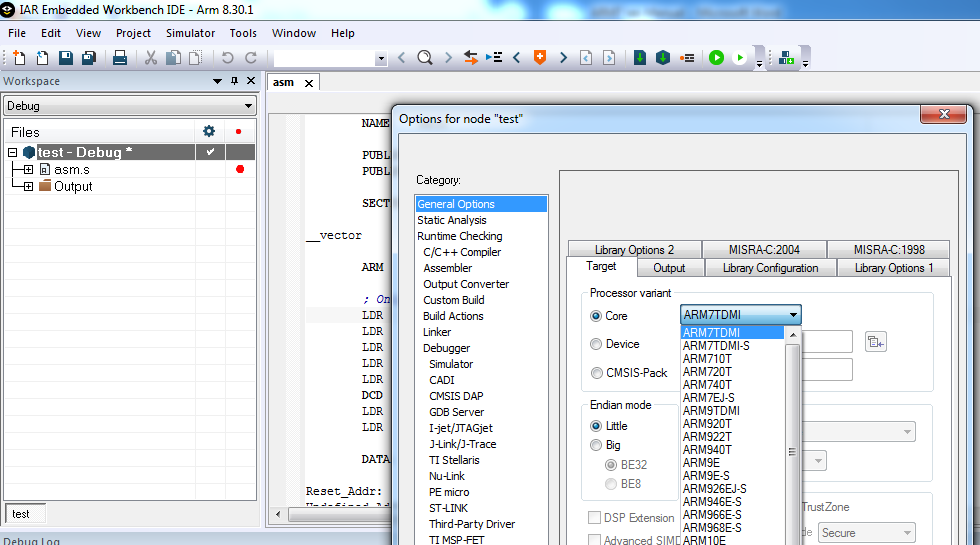
Step7: The window you can see after step6**.**



Step 8:Go to project name, right click and select **options.**

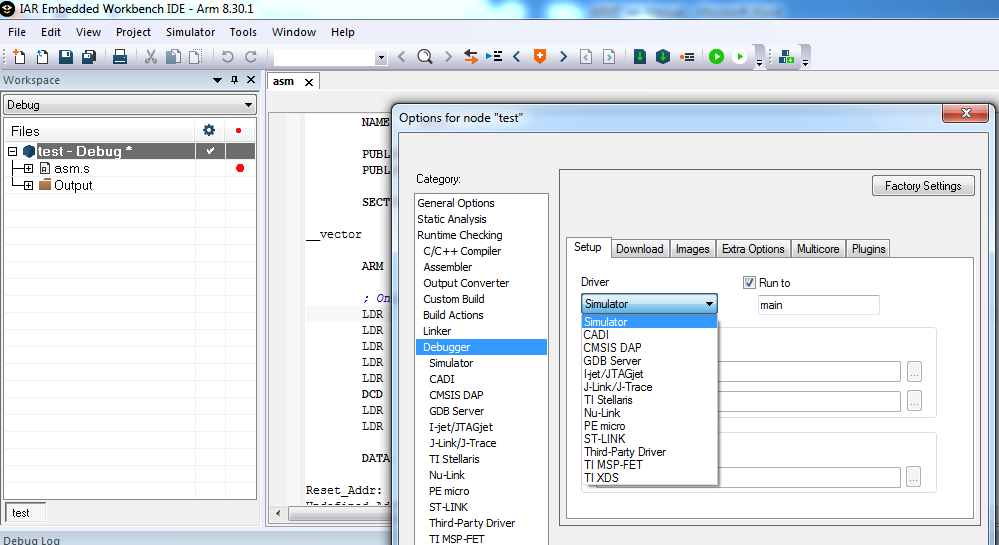


Step 9: Go to **generaloptions**>**targettab**>**core**> ARM7TDMI



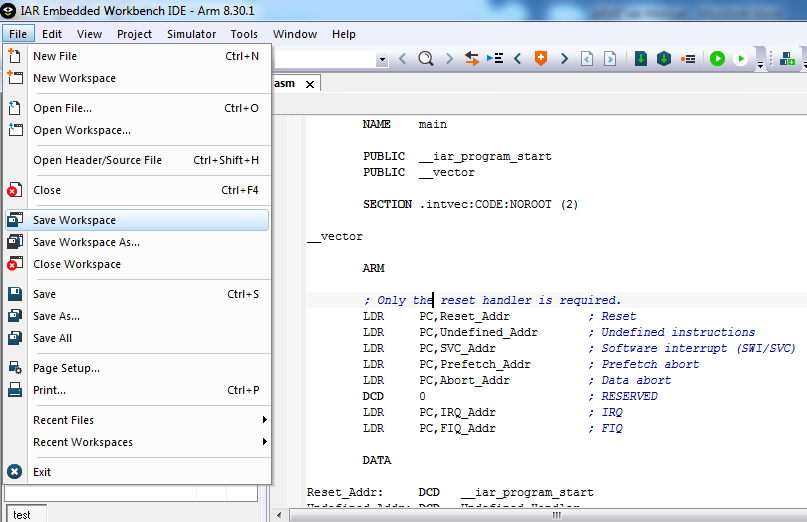
Step 10: Select device **ARM7TDMI** and click **OK.**

Step 11: Go to **debugger**>**setup** tab > select **simulator**> click **OK**

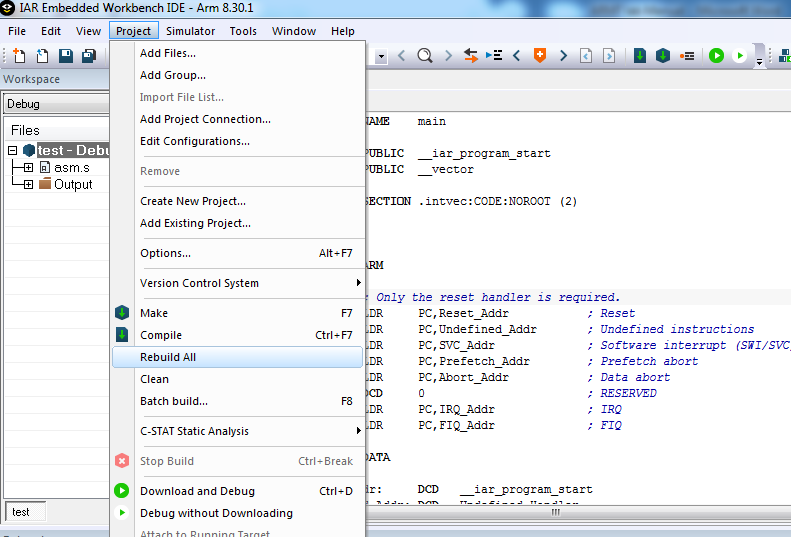


Step 12: Type program in editor window.

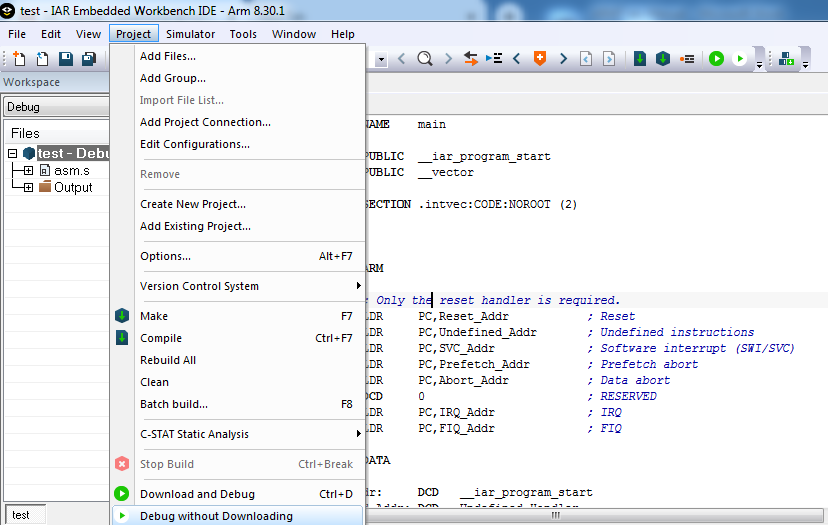
Step 13: Go to **file** ->**SaveWorkspace**, save the workspace in corresponding folder, Click **ok.**



Step 14: Go to project->**RebuildAll,**if any **errors** or **warnings** you will get the result in **build** console

****

Step 15: To **simulate** the program: Go to **Project**🡪 **Debug without Downloading**



Step 16: PC points to the initial point of the program.

Step 17:

1. If you want to see the memory then go to **view**->**memory.**
2. If you want to see the resistor then go to **view**->**register.**
3. Any resistor you can add to the watch window. Just right click on the register and click **add to watch.**
4. If you need to perform **Run**, **step over**, **step into**,**step out** and **next statement**

Then go to the **Debug** and click your option.

**Commands for Linux:**

**gedit filename.c // to edit file**

**gcc –o filename filename.c // to compile the file**

**./filename // to run the executable file**

**C Programs involving Bit manipulation**

(i) Perform logical operations on two given values.

#include <stdio.h>

int

main ()

{

int a = 7;

int b = 3;

printf ("\n The value of a AND b: %d \n", a & b);

printf ("\n The value of a OR b: %d \n", a | b);

printf ("\n The value of a XOR b: %d \n", a ^ b);

printf ("\n The value of ~a: %d \n", ~a);

printf ("\n The value of a>>1: %d \n", a >> 1);

printf ("\n The value of a>>2: %d \n", a >> 2);

printf ("\n The value of b<<1: %d \n", b << 1);

printf ("\n The value of b<<2: %d \n", b << 2);

}

(ii) Set the 5th bit in a given value

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

int value;

if (argc> 1)

{

value = atoi(argv[1]);

value |= (1<<5);

printf ("value = %d \n", value);

//}

return 0;

}

**C Program involving Device driver for reading from stdin (keyboard) and writing to stdout (monitor) using system calls**

(i) Read from keyboard and write into monitor

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main()

{

char letters[50];

int input;

input = read(0,letters,50);

printf("%d \n", input);

write(1,letters,input);

return(0);

}

(ii) Copy text from out.txt to out1.txt

//#include <stdio.h>

#include<fcntl.h>

//#include <stdlib.h>

#include <unistd.h>

//#include <string.h>

int main ()

{

char\* c;

int n;

int fin, fout;

fin = open ("out.txt", O\_RDONLY);

fout = open ("out1.txt", O\_WRONLY|O\_CREAT,0777);

while(read (fin, &c, 1)==1)

write (fout, &c, 1);

return 0;

}

(iii) Enter data from keyboard and echo into monitor and also text file

#include<fcntl.h>

#include <unistd.h>

int main ()

{

char c[50];

int n;

int f1

f1 = open ("out.txt", O\_RDWR | O\_CREAT, 0777);

n = read (0, c, 50);

write(1,c,n);

write (f1, c, n);

close(f1);

return 0;

}

# RTOS Programs (System level programming by Linux API)

**Write a C program to demonstrate usage of fork.**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main() {

   fork();

   printf("Fork testing code\n");

   return 0;

}

**Write a C program to demonstrate usage of fork with child and parent process ID is printed and parent waits for child process to terminate.**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

int

main (void)

{

  pid\_t pid;

  char \*message;

  int no, NO1 = 1;

  int i, l;

  printf ("calling fork \n");

  pid = fork ();

   switch (pid)

    {

    case -1:

      printf ("fork failed \n");

      exit (1);

    case 0:

      message = "Child Process";

      i = 1;

      no = getpid ();

      NO1 = getppid ();

      break;

    default:

      message = "Parent Process";

      i = 1;

      no = getpid ();

      NO1 = getppid ();

      break;

    }

    if(pid !=0) {

    printf("HP: hello from parent\n");

        wait(NULL);

        printf("CT: child has terminated\n");

    }

  for (l = i; l > 0; l--)

    {

      puts (message);

      printf ("My ID is %d \n", no);

      printf ("My parent ID is %d \n", NO1);

          }

  return (0);

}

**Usage of “Signal” function calls–**

**Write a C program to demonstrate usage of signal function calls: when CTRL C is pressed a signal is sent for abrupt termination.**

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

void my\_handler(int signal)

{

printf("Problem encountered %d \n", signal);

}

int main()

{

(void) signal (SIGINT,my\_handler);

while(1)

{

printf("Hello \n");

sleep(2);

}

}

**Write a C program to demonstrate usage of signal function calls: when CTRL C is pressed a signal ignore the signal.**

#include <signal.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

(void) signal (SIGINT,SIG\_IGN);

while(1)

{

printf("%d \n", getpid());

sleep(1);

}

}

**Multithreading**

gcc –o filename filename.c -lpthread // to compile the file

**Write a C program for : One thread reads the input from the keyboard and another thread converts to upper case. This is done until Stop‟ is pressed. Use concept of multithreading**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include<string.h>

#include<ctype.h>

#include <pthread.h>

#define BUFFER\_SIZE 1024

char buffer[BUFFER\_SIZE];

void \*read\_thread (void \*arg)

{

  while (strncmp ("stop", buffer, 4) != 0)

    {

      printf ("Enter text:  ");

      fgets (buffer, BUFFER\_SIZE, stdin);

      sleep (1);

    }

  pthread\_exit ("read\_thread exit successful");

}

void \*convert\_thread ()

{

  int i;

  while (strncmp ("stop", buffer, 4) != 0)

    {

      sleep (1);

      printf ("Converted text:  ");

      for (i = 0; i < strlen (buffer); i++)

printf ("%c", toupper (buffer[i]));

    }

  pthread\_exit ("convert\_thread exit successful");

}

int main ()

{

  int result;

  pthread\_t rthread, cthread;

  void \*thread\_result;

  printf("Enter text, the program will convert it into upper case, \n To stop enter 'stop' \n");

  pthread\_create (&rthread, NULL, read\_thread, NULL);

  pthread\_create (&cthread, NULL, convert\_thread, NULL);

  pthread\_join (rthread, &thread\_result);

  printf("read\_thread joined, %s\n",(char \*)thread\_result);

  pthread\_join(cthread, &thread\_result);

  printf ("convert\_thread joined, %s\n", (char \*) thread\_result);

return(0);

}

**Write a C program for : One thread reads the input from the keyboard and another thread converts to upper case. This is done until Stop‟ is pressed. Use concept of semaphore, so that multiple printing is avoided.**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include<string.h>

#include <ctype.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 1024

sem\_t sem;

char buffer[BUFFER\_SIZE];

void \*read\_thread(void \*arg)

{

while(strncmp("stop",buffer,4) != 0)

{

printf("Enter text:  ");

fgets(buffer, BUFFER\_SIZE, stdin);

sem\_post(&sem);

printf("%d \n",sem);

sleep(2);

}

pthread\_exit("read\_thread exit successful");

}

void \*convert\_thread()

{

int i;

sem\_wait(&sem);

while(strncmp("stop", buffer, 4) != 0)

{

printf("Converted text:  ");

for(i=0; i<strlen(buffer); i++)

printf("%c", toupper(buffer[i]));

sem\_wait(&sem);

}

pthread\_exit("convert\_thread exit successful");

}

int main()

{

int result;

pthread\_t rthread, cthread;

void \*thread\_result;

sem\_init(&sem, 0, 1);

printf("Enter text, the program will convert it into upper case, \n To stop enter 'stop' \n");

pthread\_create(&cthread, NULL, convert\_thread, NULL);

pthread\_create(&rthread, NULL, read\_thread, NULL);

result = pthread\_join(rthread, &thread\_result);

printf("read\_thread joined, %s\n",(char \*)thread\_result);

pthread\_join(cthread, &thread\_result);

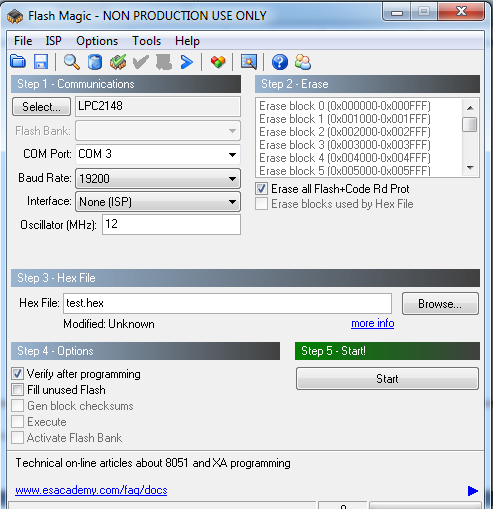
printf("convert\_thread joined, %s\n",(char \*)thread\_result);

sem\_destroy(&sem);

exit(0);

}

**Steps for hardware Interfacing using Flashmagic**

****

**Familiarize I/O ports of LPC 2148 – on/off control of LEDs using switches.**

#include "lpc214x.h"

#include "stdint.h"

unsignedintdelay\_ms,led\_val;

unsigned char index;

unsigned intmvright[]={0x80808080,0x40404040,0x20202020,0x10101010,0x08080808,0x04040404,0x02020202,0x01010101,0x00};

voidInitLPC(void)

{

PINSEL0 = 0x00L;

IODIR = 0XFFFFFFFF;

}

void Delay(unsigned intdms)

{

delay\_ms = dms;

while(delay\_ms> 0)

delay\_ms--;

}

main()

{

index=0;

InitLPC();

while(1)

{

index&= 0x7;

led\_val = mvright[index++];

IOSET =led\_val;

Delay(20000);

IOCLR=0xFFFFFFFF;

}

}

**Interface keypad and display the key pressed on 7 segment LED display.**

#include "lpc214x.h"

#include "stdint.h"

unsignedinti,delay\_ms,segval;

unsigned char index, lcdval,row,keyscan,keyret,keynum=0, keypress,scanret = 0xFF;

unsigned char seg7[] =

{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67,0x77,0x7c,0x39,0x5e,0x79,0x71,0x00,0x00,0x0};

unsigned char scan[]    = {0xEF,0xDF,0xBF,0x7F,0x00} ;

unsigned char keycode[]  = {0xEE,0xED,0xEB,0xE7,0xDE,0xDD,0xDB,0xD7,0xBE,0xBD,0xBB,0xB7,0x7E,0x7D,0x7B,0x77,0x00};

voidInitLPC(void)

{

PINSEL0 = 0x00L;

IO0DIR = 0XFFFFFFF0;

}

void Delay(unsigned intdms)

{ delay\_ms = dms;

while(delay\_ms> 0)

delay\_ms--;

}

voidGetKey()

{

       row=0;

       while(1)

       {

               IO0CLR = 0xFF;

               row&= 0x3;

               keyscan=scan[row];

               IO0SET = keyscan;

               Delay(2);

               keyret = IO0PIN;

               if (keyscan != keyret)

                   break;

               row++;

       }

for(i=0;i<0x10;i++)

       {

               if(keycode[i]==keyret)

                       keynum=i;

       }

       IO0CLR = 0xFF00;

       segval = seg7[keynum];

       segval<<= 8;

       IO0SET = segval;

}

void main(void)

{

       InitLPC();

       index=0;

while(1)

           GetKey();

}

**Waveform generation using the internal DAC of LPC 2148.**

//triangle

#include "lpc214x.h"

#include "stdint.h"

voiddelay\_ms(uint16\_t j)

{

uint16\_tx,i;

for(i=0;i<j;i++)

{

for(x=0; x<6000; x++); /\* loop to generate 1 milisecond delay with Cclk = 60MHz \*/

}

}

int main (void)

{

uint16\_t value;

uint16\_ti = 0;

PINSEL1 = 0x00080000; /\* P0.25 as DAC output \*/

IO0DIR = 0xFFFFFFFF; /\* Input pins for switch. P0.8 sine, P0.9 triangular, P0.10 sawtooth, P0.11 square \*/

while(1)

{

i=0;

while(i!=1023)

{

DACR=i<<6;

i++;

}

i=1023;

while(i!=0)

{

DACR=i<<6;

i--;

}

}

}

//SQUARE

#include "lpc214x.h"

#include "stdint.h"

voiddelay\_ms(uint16\_t j)

{

uint16\_tx,i;

for(i=0;i<j;i++)

{

for(x=0; x<6000; x++); /\* loop to generate 1 milisecond delay with Cclk = 60MHz \*/

}

}

int main (void)

{

uint16\_t value;

uint16\_ti = 0;

PINSEL1 = 0x00080000; /\* P0.25 as DAC output \*/

IO0DIR = 0xFFFFFFFF; /\* Input pins for switch. P0.8 sine, P0.9 triangular, P0.10 sawtooth, P0.11 square \*/

while(1)

{

DACR=1023<<6;

delay\_ms(10);

DACR=0;

delay\_ms(10);

}

}

**Design and display a 4 digit counter.**

#include "lpc214x.h"

#include "stdint.h"

#define IO1 0x10000

#define IO2 0x20000

#define IO3 0x40000

#define IO4 0x80000

#define IOX 0xF0000

#defineIOXcl 0xFFFFF

//Multiplexed 7segment Display

int count=0x0000;

unsigned int d0,d1,d2,d3;

unsigned char seg[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67,0x77,0x7c,0x39,0x5e,0x79,0x71,0x00};

voidinit\_gpio()

{

PINSEL0 = 0x00000000;

PINSEL1 = 0x00000000;

PINSEL2 = 0x00000000;

IO0DIR = 0XFFFFFFFF;

IO1DIR = 0XFFFFFFFF;

}

void delay()

{

int c = 100000;

while(c) //while count is more than zero loop

{

c--;

}

}

voidshow\_disp()

{

//Digit 3

d3 = count & 0x0F000;

d3 >>= 12;

IO0CLR = IOXcl;

IO0SET= seg[d3]; //Willdisplay data 1 on 7seg

IO1SET = IOX; //ALL display are OFF

IO1CLR = IO4; //Display1 is made on

delay();

IO1SET = IOX; //ALL display are OFF

//Digit 2

d2 = count & 0x0F00;

d2 >>= 8;

IO0CLR = IOXcl;

IO0SET= seg[d2]; //Willdisplay data 2 on 7seg

IO1SET = IOX; //ALL display are OFF

IO1CLR = IO3; //Display1 is made on

delay();

IO1SET= IOX; //ALL display are OFF

//Digit 1

d1 = count & 0x00F0;

d1 >>= 4;

IO0CLR = IOXcl;

IO0SET = seg[d1]; //Willdisplay data 3 on 7seg

IO1SET = IOX; //ALL display are OFF

IO1CLR = IO2; //Display1 is made on

delay();

IO1SET = IOX; //ALL display are OFF

//Digit 0

d0 = count & 0x000F;

IO0CLR = IOXcl;

IO0SET = seg[d0]; //Will display data 4 on 7seg

IO1SET = IOX; //ALL display are OFF

IO1CLR = IO1; //Display1 is made on

delay();

IO1SET = IOX; //ALL display are OFF

}

int main( void )

{

init\_gpio();

while(1)

{

show\_disp();

count++;

count&= 0xFFFF;

}

}

**Display message on LCD**

**//SINGLE LINE**

#include "lpc214x.h"

#include "stdint.h"

unsignedint cmd8[] = {0X38,0x38,0x0E,0x02,0x01,0x00};

unsigned intmsg[] = {'H','e','l','l','o',0x20,'R','I','T',0x20,0x00};

unsignedintlcdval,index,delay\_ms;

voidInitLPC(void)

{

PINSEL0 = 0x00L;

IO0DIR = 0XFFFFFFFF;

}

void Delay(unsigned intdms)

{

delay\_ms = dms;

while(delay\_ms> 0)

{

delay\_ms--;

}

}

Void InitLCD()

{

index=0;

lcdval=cmd8[index];

while(lcdval !=0x0)

{

IO0SET = lcdval;

lcdval |= 0x400;

IO0SET = lcdval;

Delay(500);

IO0CLR=0xFFFF;

index++;

lcdval=cmd8[index];

}

}

voidShowMsg()

{

index=0;

lcdval=msg[index];

while(lcdval !=0x0)

{

IO0SET = lcdval;

lcdval |= 0x500;

IO0SET = lcdval;

Delay(500);

IO0CLR=0xFFFF;

index++;

lcdval=msg[index];

}

}

void main(void)

{

InitLPC();

while(1)

{

InitLCD();

ShowMsg();

Delay(5000);

}

}