SVPM'S
COLLEGE OF ENGINEERING, MALEAON (BK)
DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

Experiment No: 1 **Title: MULTITASKING**

Class: BE E&TC

TITLE: MULTITASKING

OJECTIVE: To understand concept of multitasking using uCos II

AIM: Write a program to create three tasks and start the multitasking using ucos II.

Task1: This task to blink LED.

Task2: This task is to interface UART and display message on Terminal

Task3: This task display character on 16x2 LCD, key pressed from keyboard interfacing.

SOFTWARE USED:

- SCARM(IDE)
- Flash Magic.

HARDWARE USED:

- Educational practice board for LPC2148(ARM7 kit)
- Adapter(9V DC,500mA)
- RS232
- PC

THEORY:

- **Kernel**

The kernel is the part of a multitasking system responsible for the management of tasks (that is, for managing the CPU's time) and communication between tasks. The fundamental service provided by the kernel is context switching. The use of a real-time kernel will generally simplify the design of systems by allowing the application to be divided into multiple tasks managed by the kernel. A kernel will add overhead to your system because it requires extra ROM (code space), additional RAM for the kernel data structures but most importantly, each task requires its own stack space which has a tendency to eat up RAM quite quickly. A kernel will also consume CPU time (typically between 2 and 5%). Single chip microcontrollers are generally not able to run a real-time kernel because they have very little RAM.A kernel can allow you to make better use of your CPU by providing you with indispensable services such as semaphore management, mailboxes, queues, time delays, etc. Once you design a system using a real-time kernel, you will not want to go back to a foreground/background system.

- **Scheduler**

The scheduler, also called the *dispatcher*, is the part of the kernel responsible for determining which task will run next. Most real-time kernels are priority based. Each task is assigned a priority based on its importance. The priority for each task is application specific. In a priority-based kernel, control of the CPU will always be given to the highest priority task ready-to-run. When the highest-priority task gets the CPU, however, is determined by the type of kernel used. There are two types of priority-based kernels: *non-preemptive* and *preemptive*.

Resolution: The horizontal and vertical screen size expressed in pixels (e.g., 1024x768). Unlike CRT monitors, LCD monitors have a native-supported resolution for best display effect.

Dot pitch: The distance between the centers of two adjacent pixels. The smaller the dot pitches size, the less granularity is present, resulting in a sharper image. Dot pitch may be the same both vertically and horizontally, or different (less common).

Viewable size: The size of an LCD panel measured on the diagonal (more specifically known as active display area).

Response time: The minimum time necessary to change a pixel's color or brightness. Response time is also divided into rise and fall time.

Input lag - a delay between the moment monitor receives the image over display link and the moment the image is displayed. Input lag is caused by internal digital processing such as image scaling, noise reduction and details enhancement, as well as advanced techniques like frame interpolation. Input lag can measure as high as 3-4 frames (in excess of 67 ms for a 60p/60i signal).

Brightness: The amount of light emitted from the display (coll., more specifically known as luminance).

Contrast ratio: The ratio of the intensity of the brightest bright to the darkest dark.

16x2 LCD:

This is a 16 pin, single line connector, designed for connection to standard, text LCD modules. The pin/signal correspondence is designed to be matching with that required by such LCD modules.

- Pin 1 = GND
- Pin 2 = +5V
- Pin 3 = Vlcd
- Pin 4 = P1.25 (Used as RS of LCD)
- Pin 5 = GND
- Pin 6 = P1.24 (Used as EN of LCD)
- Pin 7 to 10 = No Connection/GND
- Pin 11 = P0.15 (Used as D4 of LCD)
- Pin 12 = P0.17 (Used as D5 of LCD)
- Pin 13 = P0.22 (Used as D6 of LCD)
- Pin 14 = P0.30 (Used as D7 of LCD)
- Pin 15 = Back lighting
- Pin 16 = GND

PIN ASSIGNMENT:

LPC2148 Reference	LCD

P0.15	D4 LCD Data Pin
P0.17	D5 LCD Data Pin
P0.22	D6 LCD Data Pin
P0.30	D7 LCD Data Pin
P1.24	EN
P1.25	RS

INCLUDED FILES:

HEADER FILE	**SOURCE FILE**
Philips\LPC2148.h	App.c
"includes.h"	

PROCEDURE:

1. Connect 9V DC Power Supply to the trainer kit.
2. Connect RS232 to evaluation board and com port of PC.
3. Open Side_arm software.
4. Create a new project. => Select manufacture- Phillips
    - ⇨ select microcontroller- LPC 2148
5. Then in editor window type main program.
6. If header files are included then add those files in ‘c files’ tab in workspace window.
7. Build and compile (Check if any errors are present and remove them)
8. Hex file will be created.
9. Now open flash magic and switch on the board
10. Step 1 => microcontroller ARM7 family, LPC2148
    - ⇨ =>Com port- Select proper com port
    - ⇨ Baud rate- 19200 (min) , 38400(max)
    - ⇨ Interface- none (ISP)
    - ⇨ Oscillator (MHz)- 12
11. Step 2 => Check ‘Erase all blocks used by Hex file’
12. Step 3 => Add respective Hex file using ‘Browse’ option
13. Click On ‘start’.
14. Thus the Hex file will get downloaded in the memory of the device.
15. Switch on respective switches present on the board
16. You will see the output

OUTPUT:

1. LEDs Interfaced on kit starts blinking.
2. Display the Character on SPJ Terminal.
3. Interface the 4*4 keyboard matrix and displays the pressed key on LCD.

CONCLUSION:

```
/*
*********************************************************************
**********************************
*                                  uC/OS-II
*                             The Real-Time Kernel
*
*
* File : APP.C
*********************************************************************
**********************************
*/

#include "includes.h"
/*
*********************************************************************
**********************************
*                                                  main()
*
* Description : This is the standard entry point for C code.  It is
assumed that your code will call
*               main() once you have performed all necessary ARM and C
initialization.
* Arguments   : none
*********************************************************************
**********************************
*/
OS_STK Task1Stk[1024];
OS_STK Task2Stk[1024];
OS_STK Task3Stk[1024];

int  main (void)
{
    BSP_Init();                             /* Initialize BSP functions
*/

    OSInit();                                /* Initialize "uC/OS-II,
The Real-Time Kernel"              */


     OSTaskCreate(App_Task1,(void *)0,&Task1Stk[1023],6);
    OSTaskCreate(App_Task2,(void *)0,&Task2Stk[1023],7);
```

```
      OSTaskCreate(App_Task3,(void *)0,&Task3Stk[1023],8);

    OSStart();                                  /* Start multitasking (i.e.
give control to uC/OS-II)       */
    while(1);

}

unsigned int i8ch ;
char szTemp[16] ;


void  App_Task1 (void *pdata)
{
    (void)pdata;

      IO0DIR |= 0x00004000 ;

    while(1)
    {
      IO0CLR = (1<<14) ;
      OSTimeDlyHMSM(0,0,0,500);
            IO0SET = (1<<14) ;
         OSTimeDlyHMSM(0,0,0,500);
    }
}


void  App_Task2 (void *pdata)
{
    (void)pdata;

    while(1)
    {

        printf("N");
        OSTimeDlyHMSM(0,0,0,600);

    }
}
```

```
void  App_Task3 (void *pdata)
{
    (void)pdata;

    DisplayRow(1,"Keypad Test ");
     DisplayRow(2,"Press Any Key  ");


    while(1)
    {

        i8ch = KBD_rdkbd() ;

        sprintf(szTemp,"KeyCode = %X",i8ch);
           DisplayRow(2,szTemp) ;

        OSTimeDlyHMSM(0,0,0,500);

    }
}
```