Name: Nivedita Londhe
PRN: 22420003
Batch: EN-4
Practical No. 8

Title: Concept of multitasking, virtual parallelism using RTOS

Objectives

- Understanding the concept of multitasking.
- Understanding the concept of virtual parallelism.
- Installation of µCos-II based application.
- Significance of all folders and files.

Multitasking

Multitasking, as the name suggests, is a technique used to handle the execution of multiple tasks. It can be defined as, "the execution of multiple software routines in pseudo-parallel. Each routine represents a separate thread of execution. The OS simulates parallelism by dividing the CPU processing time to each individual thread. The second term to define is real-time. A real-time system is a system based on stringent time constraints. For a system to be real-time, there must be a guaranteed time frame where within a task must execute. In a multitasking RTOS the task scheduling, switching and execution elements are key to the effectiveness of the OS in achieving its real-time requirements.

Concept of virtual parallelism

- Virtual parallelism is a technique used in RTOS to emulate the behaviour of running multiple tasks in parallel, even when the underlying hardware is single core. The idea is to create the illusion that several tasks are executing simultaneously, despite the limitation of the hardware.
- Virtual parallelism is a concept related to parallel computing, which involves the simultaneous execution of multiple tasks or processes to improve computational efficiency and speed. Virtual parallelism, however, does not necessarily involve physical parallel processing units like multiple processors or cores. Instead, it is a technique that makes it appear as if parallelism exists, even when the underlying hardware might not support true parallel execution.
- In above example we observed that, in first waveform it looks like each task execute at the same time, but after zooming in we can observe significant difference between execution of first task, second task, third task. It is practical example of virtual parallelism.
- Virtual parallelism is particularly useful in scenarios where hardware constraints limit the degree of parallelism that can be achieved. For example, on a single core processor, multiple tasks can be scheduled in such a way that they share processing time, giving the appearance of parallelism. This approach can improve overall system responsiveness Page No. - RTOS LAB and throughput, especially in applications where tasks may spend a significant amount of time waiting for external resources (I/O operations, network requests, etc.)
- Virtual parallelism can be achieved by task decomposition, task scheduling, concurrency, resource management, and synchronization. In task decomposition, first task or problem is divided into smaller, independent task or threads, and then executed. Task scheduling is required at the time of single processor, a scheduler or task manager is used to interleave the execution of these sub-tasks. In concurrency, tasks are not truly executing in parallel on separate processing units, they are being executed concurrently, meaning that multiple tasks are taking turns running in a way that creates the impression of parallelism. Virtual parallelism often involves managing shared resources, like memory and input/output operations, to ensure that tasks do not interfere with each other and cause conflicts. To maintain data consistency and avoid race. conditions, synchronization mechanisms are used to coordinate the execution of tasks

Installation

Installation of uCOS-II based application requires IDE i.e., Integrated Development Environment, and for that here we use Keil uVision 4.

Step 1: Download the setup of Keil from below link, after downloading extract this file
https://www.keil.com/demo/eval/armv4.htm

Step 2: Install the mdk410 setup file , then click on Next, then Next, then agree to terms and condition

Step 3: Then do further process, and fill information in customer information. After completing this , click on Next, it will start to setup and display status about it.

Step 4: After this finish the setup

Significance of folder

1) ARM

In this folder all device dependent files are present as we are using ARM processor. These files include processor and implementation specific constants, these files are in c and assembly language. In Os_cpu_c.c folder contain many files related to interfaces like lcd, led, uart, config file, relay etc. which includes program of each interface and which includes basic functions.

2) µCOS-II

In this folder all device independent files are present, which needs to include as system can be of any type or of any system, so for compatibility it must be included in project. Page No. - RTOS LAB It consists of various .c files with respect to real time operating system. It includesfile like OS_core.c, OS_FLAG.c, OS_SEM.c , OS_TASK.c , OS_TIME.c and many more, which includes functions related to OS.

3) MD2148

It includes a board specific folder which contains function related to interfacing board. Like lcd.c , led.c, uart.c etc.

4) Main:

It includes folder for application programmer. This folder contains all application programs.

Program for 3 tasks of LED Blinking with different delay

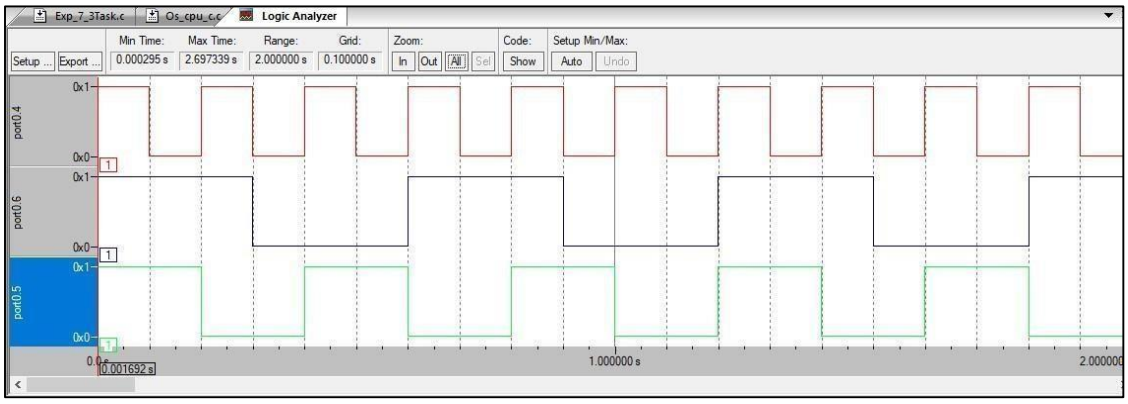
Code:

```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>
#define TaskStkLengh 64         //Define the Task0 stack length
OS_STK TaskStk0 [TaskStkLengh]; //Define the Task stack
OS_STK TaskStk1 [TaskStkLengh]; //Define the Task stack
OS_STK TaskStk2 [TaskStkLengh]; //Define the Task stack
void Task0(void *pdata);
void Task1(void *pdata);
void Task2(void *pdata);
// Required forsending time to serial port
char buffer[25];
int main (void){
```

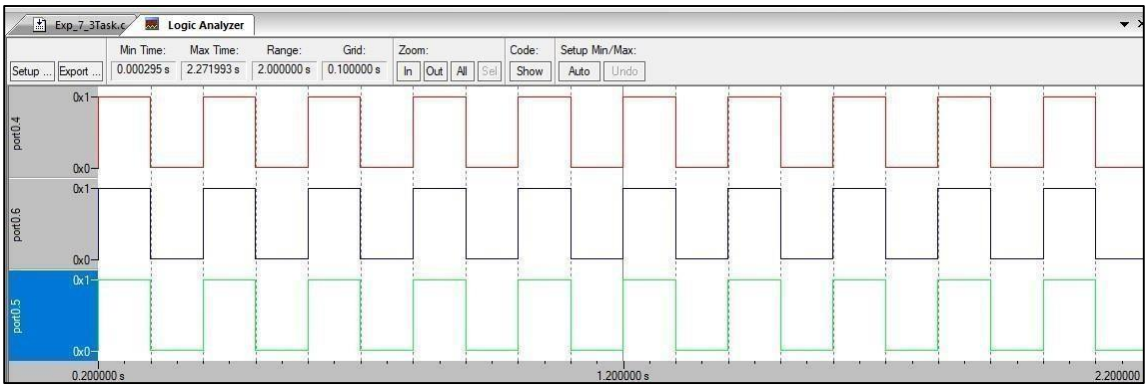
```
    LED_init();
    TargetInit();
    OSInit ();
    OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
    OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);
    OSTaskCreate (Task2,(void *)0, &TaskStk2[TaskStkLengh - 1], 8);
    OSStart();
    return 0;
}

void Task0 (void *pdata){
    pdata = pdata;      // Dummy data
    while(1){
        LED_on(0);
        OSTimeDly(10);
        LED_off(0);
        OSTimeDly(10);
    }
}

void Task1 (void *pdata){
    pdata = pdata; /* Dummy data */
    while(1){
        LED_on(1);
        OSTimeDly(20);
        LED_off(1);
        OSTimeDly(20);
    }
}

void Task2 (void *pdata){
    pdata = pdata; /* Dummy data */
    while(1){
        LED_on(2);
        OSTimeDly(30);
        LED_off(2);
        OSTimeDly(30);
    }
}
```

Observation

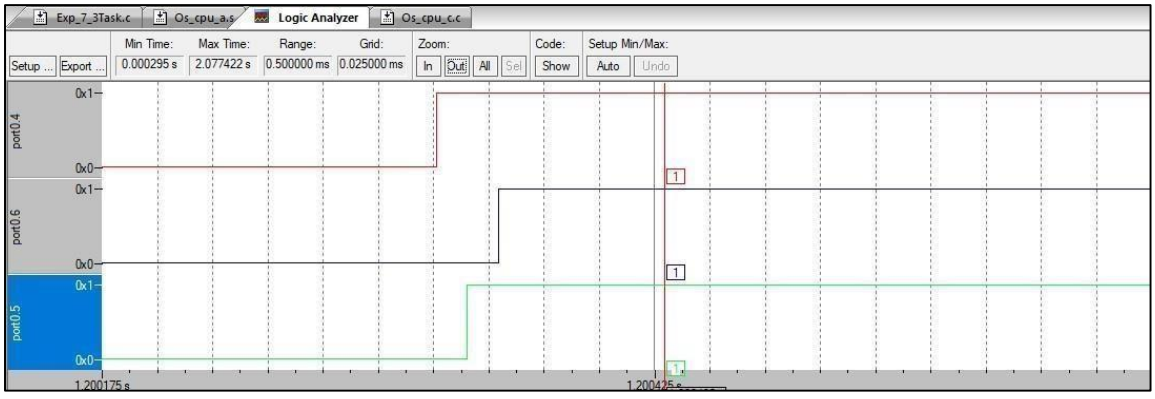
1) Different time delay:



2) Same time delay:



If we zoom in the above waveforms, it can be seen that all tasks are not carried out at same time



Here we can see difference between two waveform is in microsec. It looks like waveforms are executing at the same time, but it is not. So, it can be understood using concept of virtual parallelism.

Conclusion:

1) From the output of the code, we can see that multiple tasks are running at the same time. Hence it is a multitasking.

2) Virtual parallelism means, to create an illusion that several tasks are running simultaneously.