Lab Session 2

Introduction to µCOS-II based application

Name : Durvesh Naresh Patil
PRN : 2019BTEEN00035
Sub. : RTOS

2.4.1. 1) List the folders in the uVision project of µCOS based application

⇒ ARM, µCOS-II, MD2148, main

2) Explain the use of each folder.

⇒ (i) ARM : Includes device dependent files as we use ARM processor. These files include processor & implementation specific constant

(ii) µCOS-II : It consist of various .c files with respect to real time operating system

(iii) MD2148 : It is a board specific folder which contains functions related to interfacing board

(iv) main : This folder is for application programmer. This folder contains all application programs i.e. main C code of program

3) Advantages of such folder structure.

⇒ Main advantage is that programmers work with only folder which is associated with their work. Few other advantages are:

(i) Easy to understand
(ii) Easy to debug & run
(iii) Easy to modify

2.4.2. 1) How many tasks are there in the application ?

⇒ 4 tasks. Task0 , Task1 , Task2 , Task3

2) Do you find every task body has infinite loop ? write code of task0.

⇒ Yes, every task body has an infinite loop

task0 code:

```
Void Task0 ( void *pdata)
{
    pdata = pdata;
    while (1)
    {
        LED-on(0);
        OSTimeDly(4);

        LED-off(0);
        OSTimeDly(4);
    }
}
```

3) What does task0 do ?

⇒ Task0 infinitely turns on & off the LED i.e. blinking of LED.

2.4.3. 1) How to built the project ?

⇒ create project
Open main c file
Save it
Built (F7) by clicking on built button at top

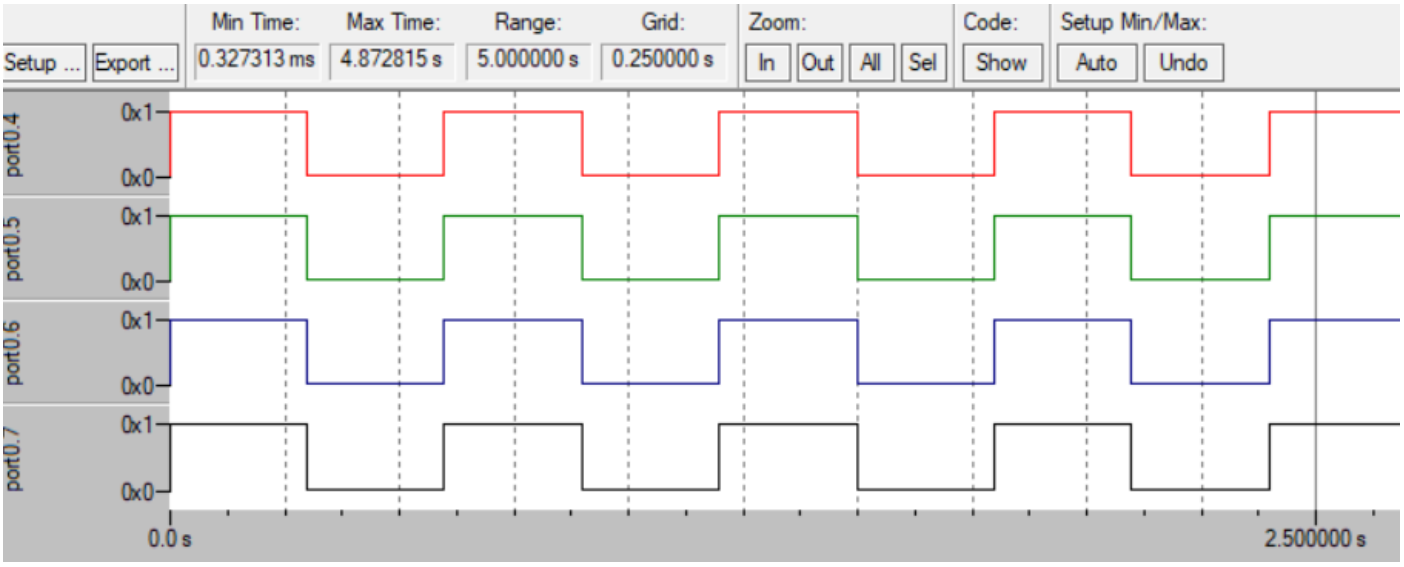
2) How to open the logic analyzer window & How to add signals ?

⇒ Built the project → Debug the project
New window will open
click on analysis window
logic analyser window will be opened.
click on setup icon at top left corner → then add name of signal → press enter → close

3) Did you do any mistakes while adding signals to logic analyser window ?

⇒ No mistakes were done.

2.4.4. Paste screenshot of observed waveform.



7) Write your observations about the waveform.

⇒ Though all tasks are having infinite loop they are executing simultaneously i.e. parallely. All the tasks are executed parallely i.e. it supports multiprogramming.

8) conclusion you draw from observations.

⇒ Operating system is preemptive one. It executes multiples processes simultaneously i.e. it supports parallelism.

2.4.5. Time of tasks before editing task0

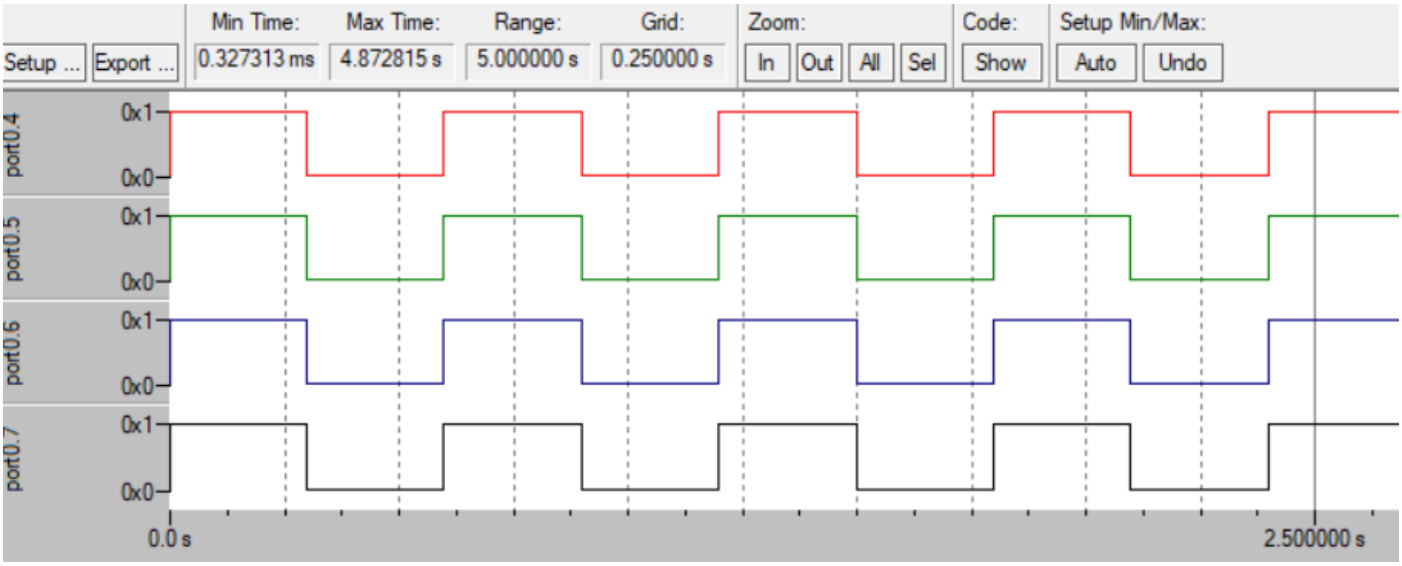
on period		off period	
task0	: 0.80 s	task0	: 1.20 s
task1	: 0.80 s	task1	: 1.20 s
task2	: 0.80 s	task2	: 1.20 s
task3	: 0.80 s	task3	: 1.20 s

Time of tasks after editing task0. (delay = 6 for task0)

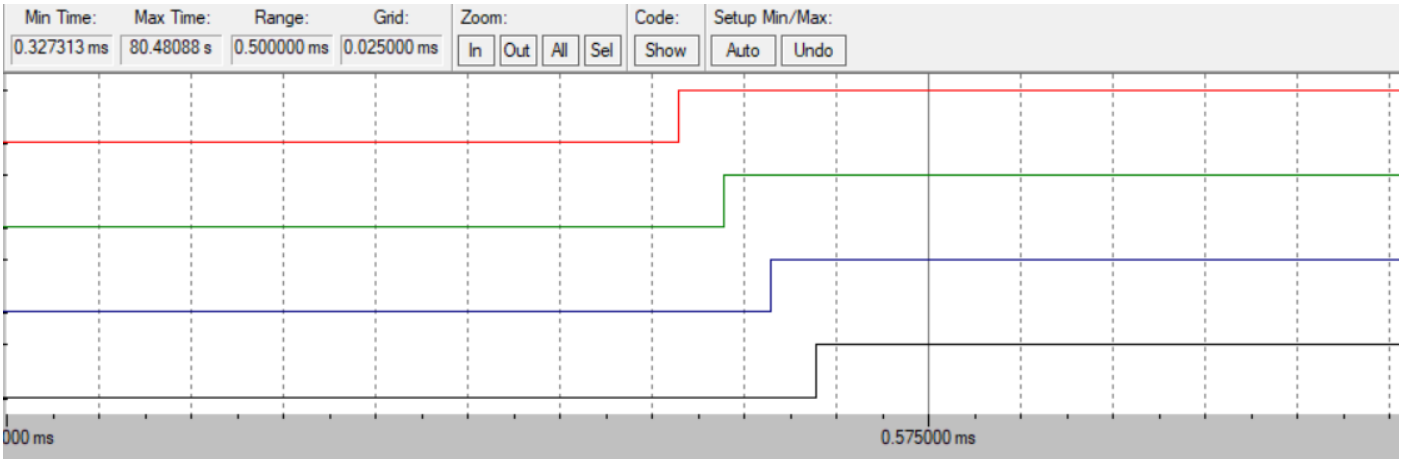
on period		off period	
task0	: 1.20 s	task0	: 1.80 s
task1	: 0.80 s	task1	: 1.20 s
task2	: 0.80 s	task2	: 1.20 s
task3	: 0.80 s	task3	: 1.20 s

⊙ Conclusion : Change in one task will not affect the other tasks, so we can conclude that all tasks are executing independently.

Before Zooming



After Zooming



2.4.6. Screenshots of before zooming & after zooming are shown above.

- what conclusion can you draw from observations?

⇒ Even though the tasks appear to be running simultaneously they are not executing at same time. There is some delay between them. Delay is due to context switching.

- write the reason of this delay. If you made any measurements, write them.

⇒ There is only one processor & there are 4 tasks that run at same time. main reason for delay is the context switching.

Measurements: task0 : 1.600355 s
task1 : 1.600369 s
task2 : 1.600383 s
task3 : 1.600397 s

- What do these measurements indicate?

⇒ The measurements indicate that there is a very little time difference bet^n tasks, so all tasks are not executing at the same time. Delay is so little that it seems like all processes are running parallely i.e. at same time.

led1.c | Os_cpu_c.c

```
092         void SWI_Exception(int SWI_Num, int *Regs)
093 {
094     OS_TCB    *ptcb;
095     unsigned int temp;      //ÒÔÇ°µÄkeil for 2.5 »ò ADS1.2ÏÂ£¬Õâ
096                             //Ö±½ÓÓÃR0¼´¿É£¬£¬
097
098     switch(SWI_Num)
099     {
100         //case 0x00:                        /* ÈÎÎñÇÐ»»°¯ÊýOS_TASK_
101         //    break;
102         //case 0x01:                        /* Æô¶¯ÈÎÎñ°¯ÊýOSStartH
103         //    break;
104         case 0x02:                          /* ¹ØÖÐ¶Ï°¯ÊýOS_ENTER_CR
105             asm
```

2.4.7. 1> Run the project & stop it. Did it open some other file & showed the code execution point in that file? Paste the screenshot.

⇒ Yes, when project was stopped, so other file was opened & showed execution point in that file.

2> What conclusion can you draw from the observations

⇒ Every time we stop the execution of project, some new file is opened up & execution point is highlighted. The reason for this is source code of operating system is part of project we are running & these source code files are also compiled with application program

2.4.8 1> open any os file (from ucos folder) & create a small syntax error. compile the project. what did you observe.

⇒ The compiler throws an error & the target is not created

2> What conclusion can you draw from observations?

⇒ Operating system source code files are also compiled along with application code & hex file is created.

2.4.9 Write the mistakes which you did & how you corrected them.

⇒ As experiment is easy one. No mistakes were done by me.

2.4.10 Write your opinion about whether final hex file contains the os code as well as application code. Justify your opinion.

⇒ It is true that hex file contains the code of os as well as application. Since we have seen that by creating error in os folder file it will create a error & hex target is not created.

2.4.11 Write your opinion about whether the RTOS runs the processor where the hex file will be downloaded (along with application code). Justify your opinion.

⇒ Yes, RTOS runs on processor where the hex file will be downloaded. When we stop the execution, we see that execution point is shown in the processor file & not in the application code or in os file. So when we upload the hex file on board, it will run on processor.

2.4.12 Type any task without copy paste. Built & run the project. write your experience below.

⇒ First I commented out the tasks & then tried to type it by myself without seeing the code. Initially I struggled at syntax as most of the functions are predefined so it is difficult to memorize, so I took help from task2 code then compiled the code & saw the waveform later I changed delay 2-3 times & noticed the change in the waveform duty cycle

2.4.13 Modify the self written task for creating different waveforms pattern. Build & run the project.

① write your experience

⇒ I generated different - different waveforms having different duty cycles by changing delays in the code.

② Paste screenshot.