Lab Session 3

Detailed structure of RTOS based application

~~3.3.1. Need of the~~

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

3.3.1. Need of the config.h file & facilities in that file

1) Why do we need to configure the RTOS ?
⇒ RTOS is customized using configuration files. It is used by the RTOS kernel for specific application being built. So we need to configure RTOS to run each specific application properly.

2) Which are the important characteristics in config.h ?
⇒ config.h contains the number of tasks, crystal frequency, OS Time Ticks, etc. These are some of the important characteristics

3) Why do we need to set crystal frequency in the configuration file ?
⇒ Crystal frequency determines the important parameter like delay, it is mandatory to keep both frequencies same to design specific clock delay.

4) Why to set maximum no. of tasks ?
⇒ While designing any system or application, application programmer must know the number of tasks that should be executed so that memory usage will be optimum. To save data memory (RAM)

5) Why to set ticks per second value ?
⇒ We can specify timing / delay using set ticks per second parameter.

3.3.2 Need of stdlib.h & stdio.h Files.

stdio.h is for standard input output while stdlib.h is header file for standard library

Links :
- www.geeksforgeeks.org
- tutorialspoint.com/header-files-stdio-h-and-stdlib-h-in-c

3.3.3. Defining the stack

1) Why every task needs to have seperate stack ?
⇒ As all the tasks are independent on each other we need to have seperate stack. If not given seperate stacks then one task may block the other task.

2) What for the stack is used ?
⇒ for context saving & retrieving.

3) Whether stack is in RAM or ROM ? why ?
⇒ Stack is in RAM as we want to read & write the data.

4) How stack is defined in ucos-II ? what is the data type of each member of the stack ?
⇒ # define TaskLength 64
OS_STK_TaskStk0 [TaskLength];

Data type: void pointer task, pd, pdata
16-bit integer opt.
stack pointer ptos.

5) Use same stack for two tasks & observe the output. Write observations & reasons.
⇒ We do not get proper output that we are expecting. When low priority task executes & its current state placed in stack & then preempted by high priority task. It uses same stack & below memory is used by low priority task

3.3.4 Task function declarations.

1) How task functions are declared ?
⇒ OSTaskCreate (Task0, (void *)0, & TaskStk0 [TaskStkLength -1], 6);

2) Why declaration of these function is required?
⇒ To use RTOS we need to first initialize it, create it with priority order & then start OS. To do this, we need to declare functions.

3.9.5 TargetInit() & OSInit() functions.

1) What does the TargetInit() fn do? How to verify that practically?
⇒ This fn is used to enable VIC, timers, UART, memory, etc.
To verify it practically, debug the code put break points before & after this function.

2) What does the OSInit() function do?
⇒ It contains multiple functions like prevent rescheduling, multitasking, move tasks, wait/ready to run for event.

3) Comment these functions one by one & observe the o/p. Write the observation & the reason.
⇒ On commenting these functions won't yield any output as all these functions are necessary for the task's execution.

3.3.6 Need of OSTaskCreate function

1) How a task is created & Write the C statement for it.

⇒ OSTaskCreate (Task_Name, (void *) 0, Add. of task, priority);

OSTaskCreate (Task0, (void *) 0, &TaskStk0 [TaskStkLength-1], 6)

2) Even if the body of the fn is already written, why we need to create the task?

⇒ Creating task means informing the OS about the task i.e. when to run, where to run etc which is not there in body of fn

3) Illustrate the meaning of each of the arguments of the fn for creating the task.

⇒ (i) OSTaskCreate () ⇒ function having definition of allocating storage for CPU status register, start of multitasking, etc.

(ii) Task0 ⇒ Name of the task created

(iii) (&TaskStk0 [TaskStkLength-1], 6) ⇒ Address of stack of task & priority.

3.3.7 Need of OSStart()

1) What does the OSStart() function do?

⇒ It is used to start the multitasking process which lets μCOS-II manage the task. But before calling of OSStart() user must call OSInit() & there should be at least one task.

2) comment this function. observe the o/p of the program & write your comments on it.
⇒ There will not be any output as OSStart() function is essential for μCOS-II

3) Put a breakpoint on the statement immediately after OSStart() function. Run the program. Write the observation. Think of the reason & write it.
⇒ Statement after OSStart() is return 0. It is the unreachable statement in the program. As OSStart() runs it will run the tasks, as tasks contain while(1) loop so return 0 is unreachable statement.

3.8 Task functions. Return type & Pointer argument.

1) What does a task function do? Why does a task have while(1) loop?
⇒ Task function consists a block of function codes to perform a particular activity. The while(1) loop runs each task continuously.

2) What is the need of the argument of task function?
⇒ If the task required to work on external data in its function then the data can be passed to the function through argument

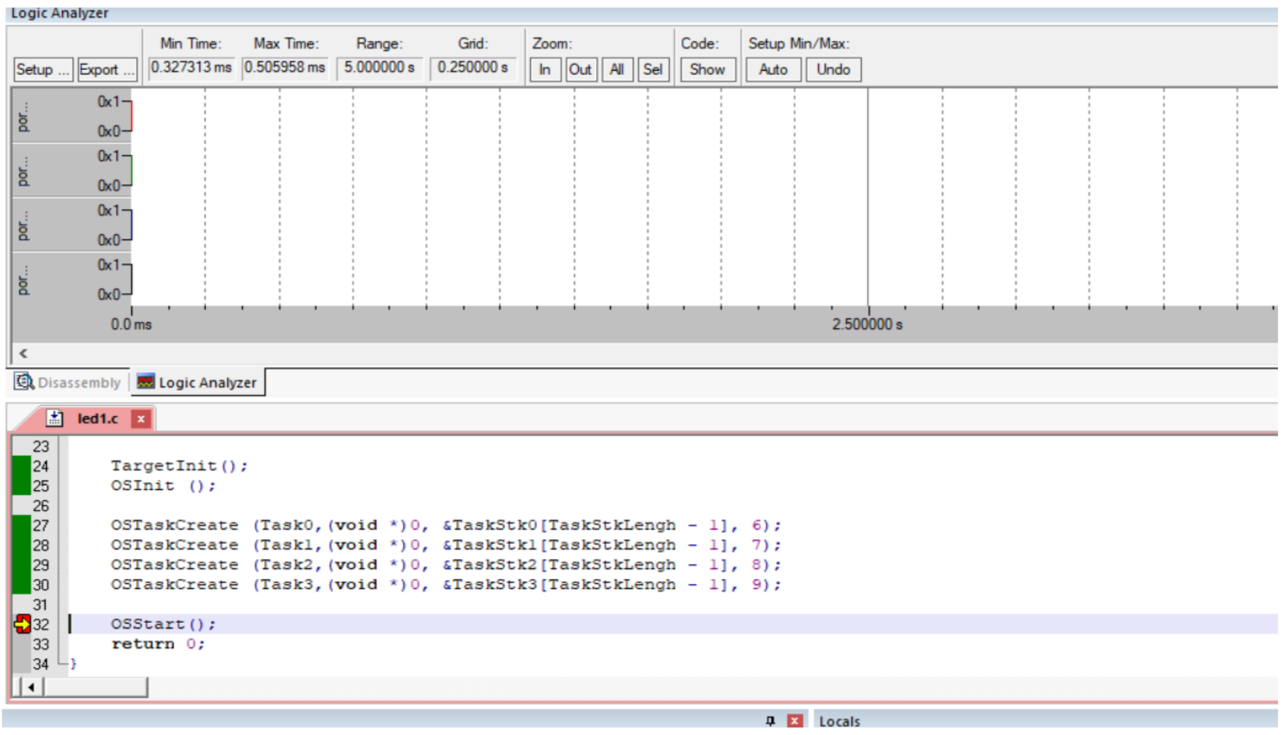
3) Why it is a void pointer type?
⇒ Allows user to send data of any datatype. It acts like an universal pointer.

4) Why the task function has no return type?
⇒ (i) RTOS scheduler that calls the function initially is not designed to handle a return from task.
(ii) The task is an infinite loop & so there is no return value.

3.3.9 <u>Stepwise execution of RTOS based application.</u>

1) Put a breakpoint at the first executable statement of all task. Run the program & write observations about the execution.
⇒ After running the program, all tasks were executed as per the priority. Lowest number means higher priority & vice versa.

2) Paste the screenshot of observations. Immediately before the execution of OSStart(). Write comments.
⇒ <u>Comments</u>: No output will be generated as OSStart() function is yet to call.



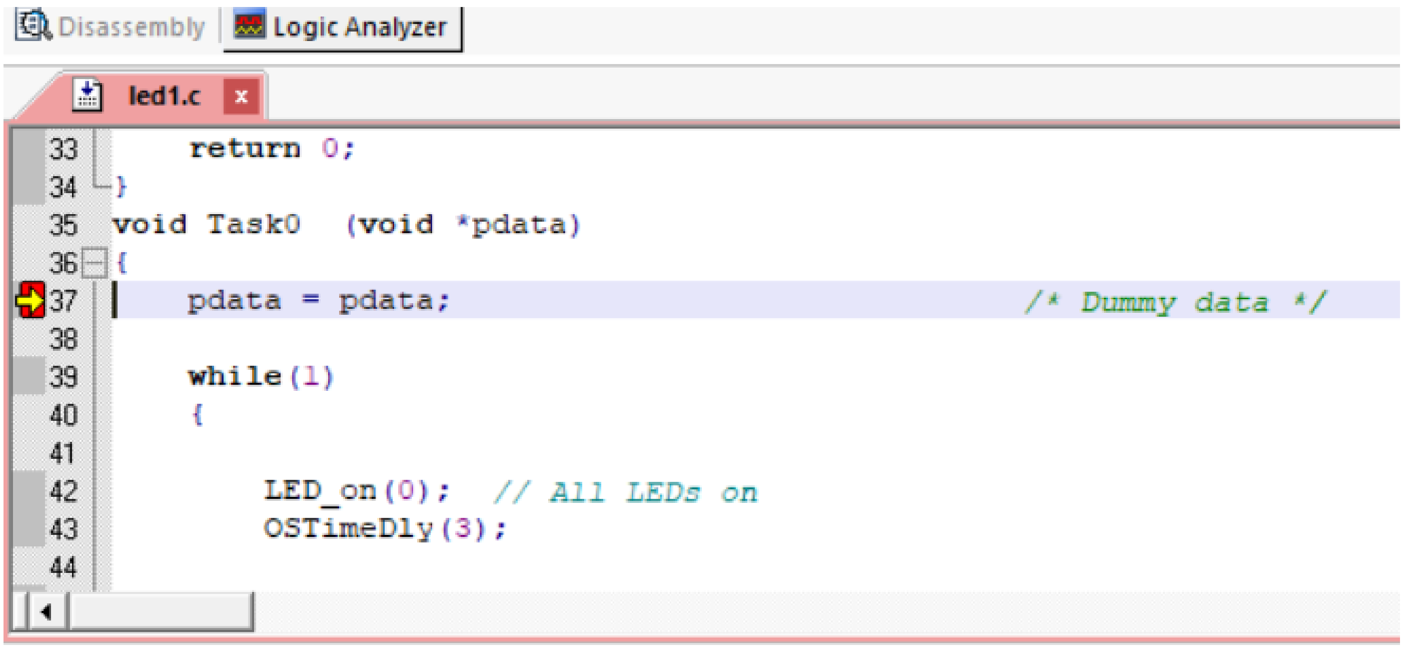
```
    TargetInit();
    OSInit ();

    OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
    OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);
    OSTaskCreate (Task2,(void *)0, &TaskStk2[TaskStkLengh - 1], 8);
    OSTaskCreate (Task3,(void *)0, &TaskStk3[TaskStkLengh - 1], 9);

    OSStart();
    return 0;
}
```

3> After pressing run button once, write comments.
⇒ Comments : Execution point goes to the task0 function which has the highest priority

4> After pressing run button once again. write comments
⇒ Comments : Execution point goes to the task1 which is the next higher priority task.

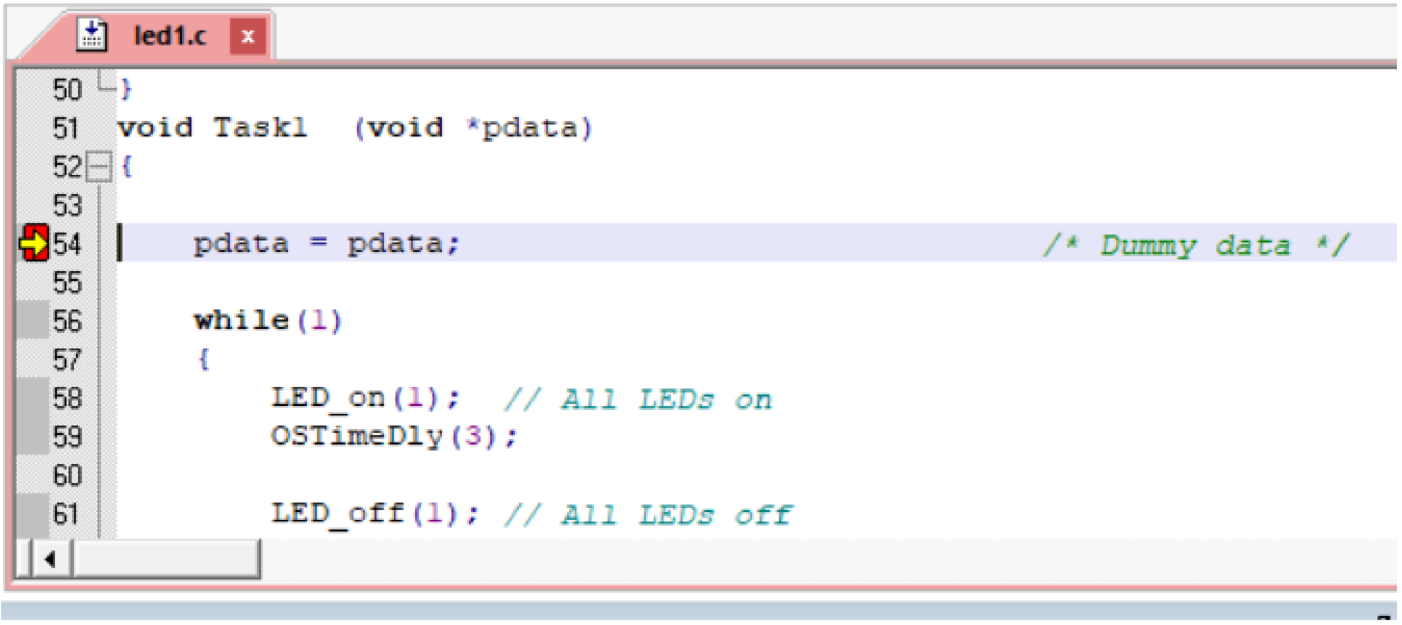


```
    return 0;
}
void Task0   (void *pdata)
{
    pdata = pdata;                                    /* Dummy data */

    while(1)
    {

        LED_on(0);  // All LEDs on
        OSTimeDly(3);

```



```
}
void Task1   (void *pdata)
{

    pdata = pdata;                                    /* Dummy data */

    while(1)
    {
        LED_on(1);  // All LEDs on
        OSTimeDly(3);

        LED_off(1); // All LEDs off
```

3.3.10 <u>Creating task & deleting task within task</u>

1) modify the program to create only task0 in main(). Create task1 in task0. Write the code of task0 only.

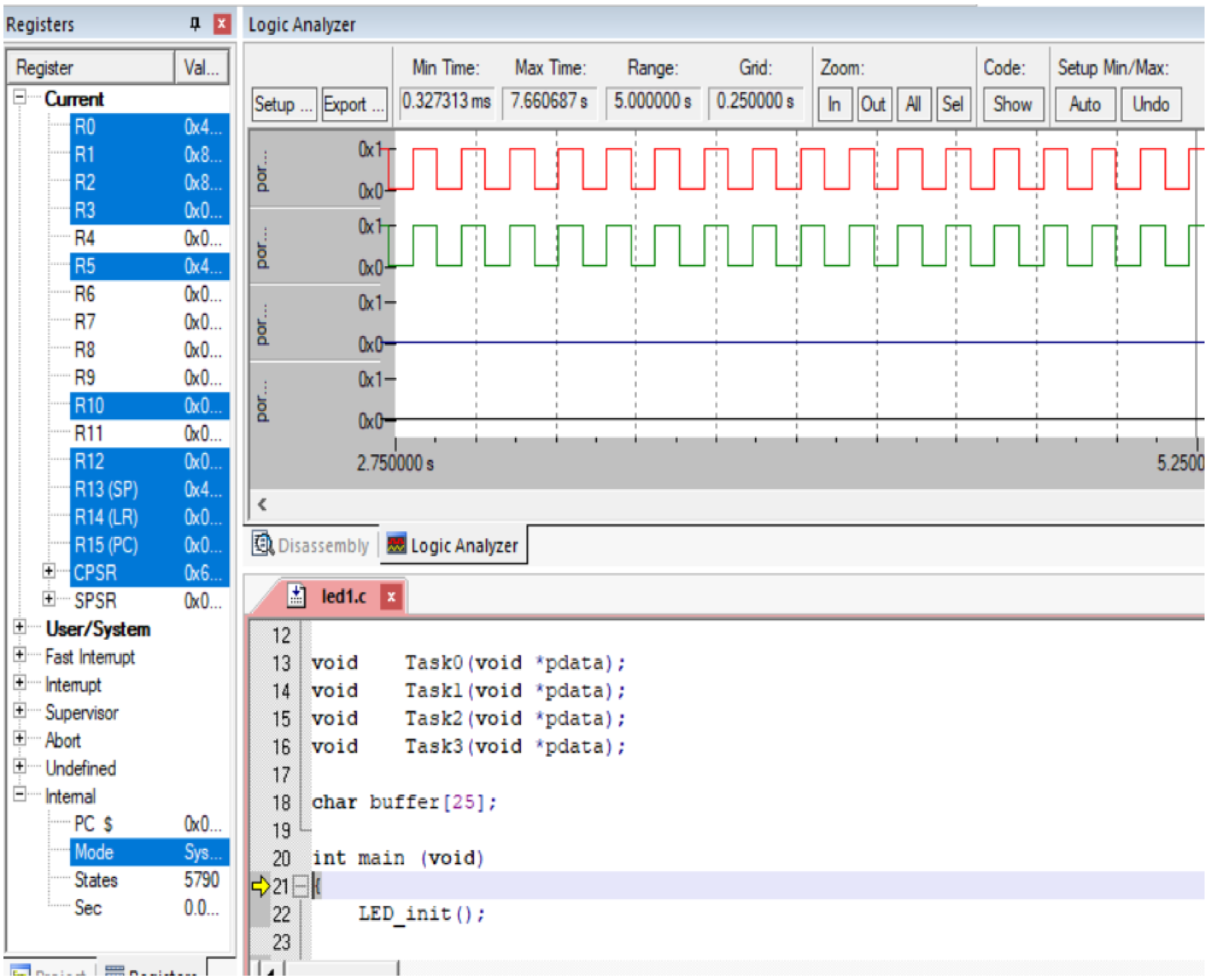
⇒

```
void Task0 (void *pdata)
{
    pdata = pdata;
    OSTaskCreate( Task1, (void *)0, &TaskStk1[
        TaskStkLength-1], 7);
    while(1)
    {
        LED_ON (0);
        OS_TimeDly (4);

        LED_OFF (0);
        OSTimeDly (4);
    }
}
```

2) Run the program. Paste screenshot & write observations.

⇒ <u>Comments</u> : Both task0 & task1 were executed properly



Register	Val...
Current	
R0	0x4...
R1	0x8...
R2	0x8...
R3	0x0...
R4	0x0...
R5	0x4...
R6	0x0...
R7	0x0...
R8	0x0...
R9	0x0...
R10	0x0...
R11	0x0...
R12	0x0...
R13 (SP)	0x4...
R14 (LR)	0x0...
R15 (PC)	0x0...
CPSR	0x6...
SPSR	0x0...
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x0...
Mode	Sys...
States	5790
Sec	0.0...

Min Time:	Max Time:	Range:	Grid:
0.327313 ms	7.660687 s	5.000000 s	0.250000 s

Zoom: Code: Setup Min/Max:

```

void    Task0(void *pdata);
void    Task1(void *pdata);
void    Task2(void *pdata);
void    Task3(void *pdata);

char buffer[25];

int main (void)
{
    LED_init();

```

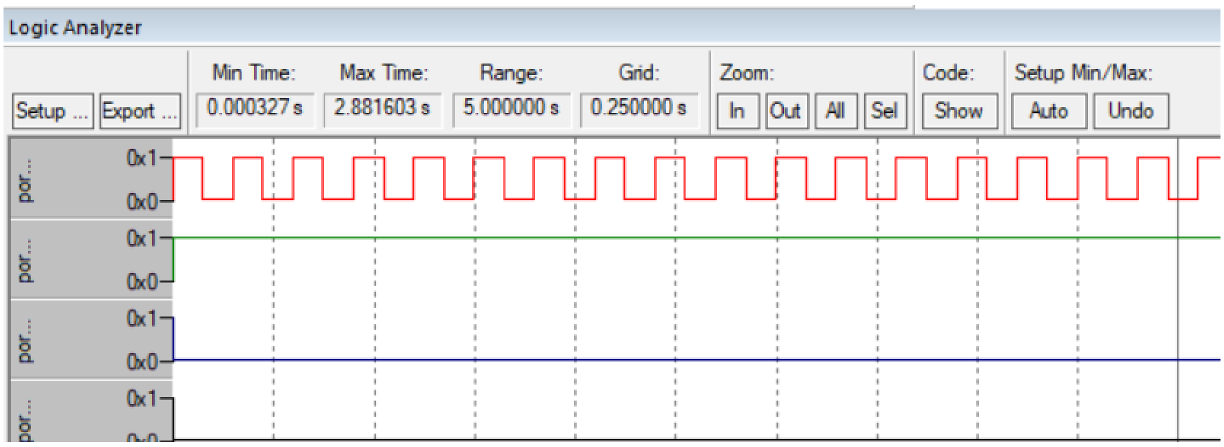
3) Modify the program to create only task0 in main(). In while(1) of task0, create task1. Take delay of 500ms. Then delete the task1. Again take delay of 500ms. Repeat this. Write the code of task0 only.

⇒ (Note: OS_TICKS_PER_SEC 10)

```
void Task0(void *pdata)
{
    pdata = pdata;
    while(1)
    {
        OSTaskCreate(Task1, (void *)0, TaskStk1
            [TaskStkLength-1], 7);
        OSTimeDly(5);
        OSTaskDel(7);
        OSTimeDly(5);
    }
}
```

4) Run the program. Paste screenshot & write observation:

⇒ Observation :
    Task1 gets executed even after creating it in Task0.



Min Time: 0.000327 s
Max Time: 2.881603 s
Range: 5.000000 s
Grid: 0.250000 s

M T W T F S S
Page No.:
Date:
YOUVA

3.3.11 <u>Observing the effect of OSTimeDly on the execution sequence of application.</u>

1) Put the breakpoints at LED on/off as well as OSTimeDly function in task0 as well as task1. Run the program & write your observation.

⇒ Put breakpoints on LED-ON(0) of task0, OSTimeDly(1) of task0 and so on.
The execution flow will be as below:

$$\begin{tikzcd}
\begin{array}{c}\text{LED-on}\\ \text{(task 0)}\end{array} \arrow[r, Rightarrow] & \begin{array}{c}\text{Delay}\\ \text{(task 0)}\end{array} \arrow[r] & \begin{array}{c}\text{LED-on}\\ \text{(task 1)}\end{array} \arrow[r, Rightarrow] & \begin{array}{c}\text{Delay}\\ \text{(task 1)}\end{array} \arrow[r, Rightarrow] & \begin{array}{c}\text{LED-off}\\ \text{(task 0)}\end{array} \arrow[d, Rightarrow] \\
 & & \begin{array}{c}\text{Delay}\\ \text{(task 1)}\end{array} & \begin{array}{c}\text{LED-off}\\ \text{(task 1)}\end{array} \arrow[l, Leftarrow] & \begin{array}{c}\text{Delay}\\ \text{(task 0)}\end{array} \arrow[l, Leftarrow]
\end{tikzcd}$$

2) Write reasons to support these operations.

⇒ Calling delay function causes a context switching and forces the uCOS-II to execute the next task of high priority so it is like, after turning on the LED on from task0, execution point goes to LED on from task 1.