Lab 1 for uC/OS-II: Periodic Task Emulation

Prof. Li-Pin Chang

ESSLab@NCTU

Objectives

- To implement periodic tasks
- To observe the scheduling behaviors
  - Fixed priority scheduling

Task Sets

- Two sets of periodic tasks
  - Task set 1 = { t1(1,3), t2(3,6) }
  - Task set 2 = { t1(1,3), t2(3,6), t3(4,9) }
  - Tasks all arrive at the same time
  - Show context switch behaviors
  - Show deadline violations if there is any

Issues

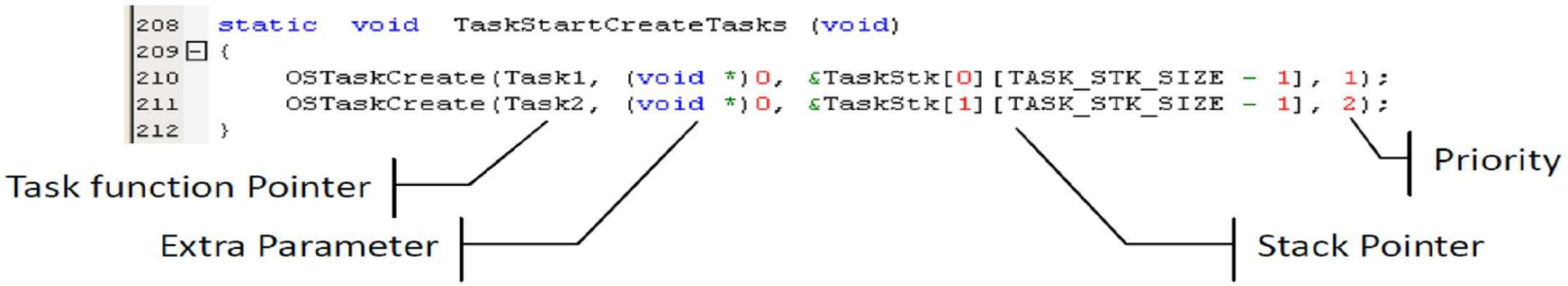
- How to create a task that executes exactly c units of time in every p units of time?
  - (c,p)
- Where in the kernel can we add code to observe context switches?
  - Voluntarily; complete
  - Involuntarily; preempted

Notices

- In actual real-time systems, task jobs are released by periodic interrupts
- Task computation time is determined by worst-case computation time analysis (WCET)
- In this project we emulate this behavior, and, more importantly, to get insights into how CPU time is allocated to tasks

Periodic tasks

- Call OSTaskCreate to create a task



- A basic task template

```
while(1)
{
        … do task computation …
        … up to C units of time …

        OSPendSem();        // posted by an ISR at a regular frequency
}
```

Periodic tasks

- A basic task template

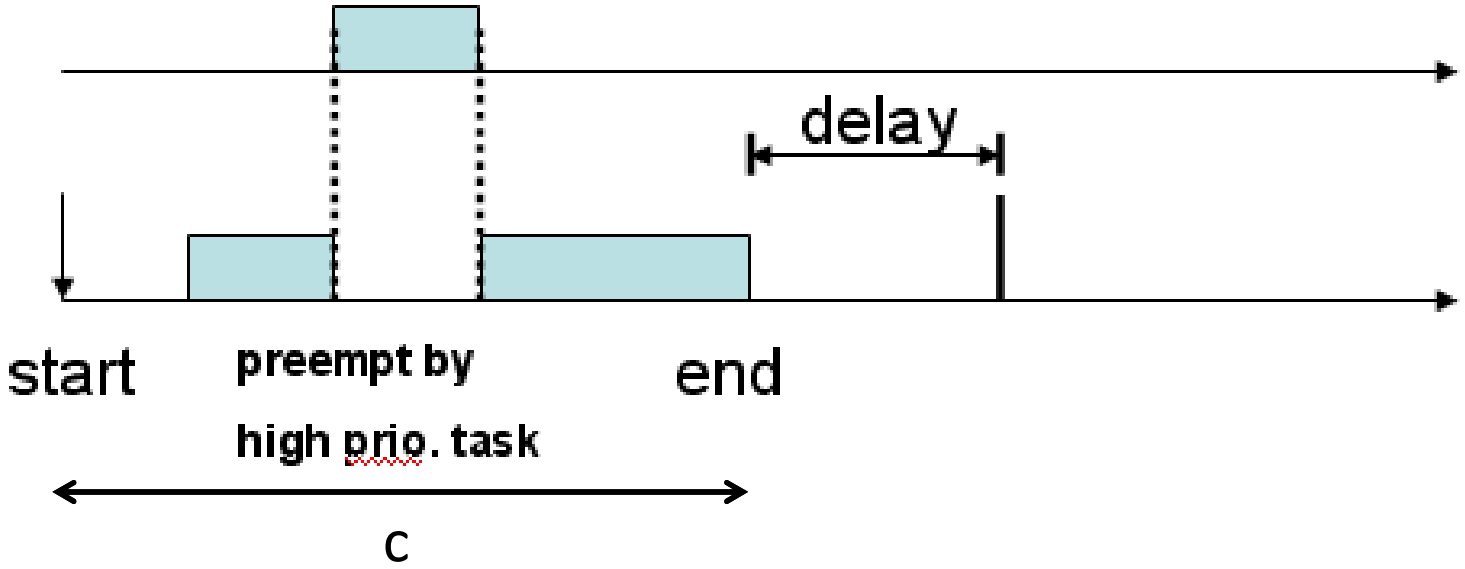
```
while(1)
{
        … do task computation …
        … up to C units of time …

        OSPendSem();          // posted by an ISR at a fixed frequency
}
```

- A straightforward emulation of (c,p)

```
while(1)
{
        Start=OSTimeGet() ;
        While(OStimeGet()-start < c) ;
        OSTimeDly (p-c) ;
}
```

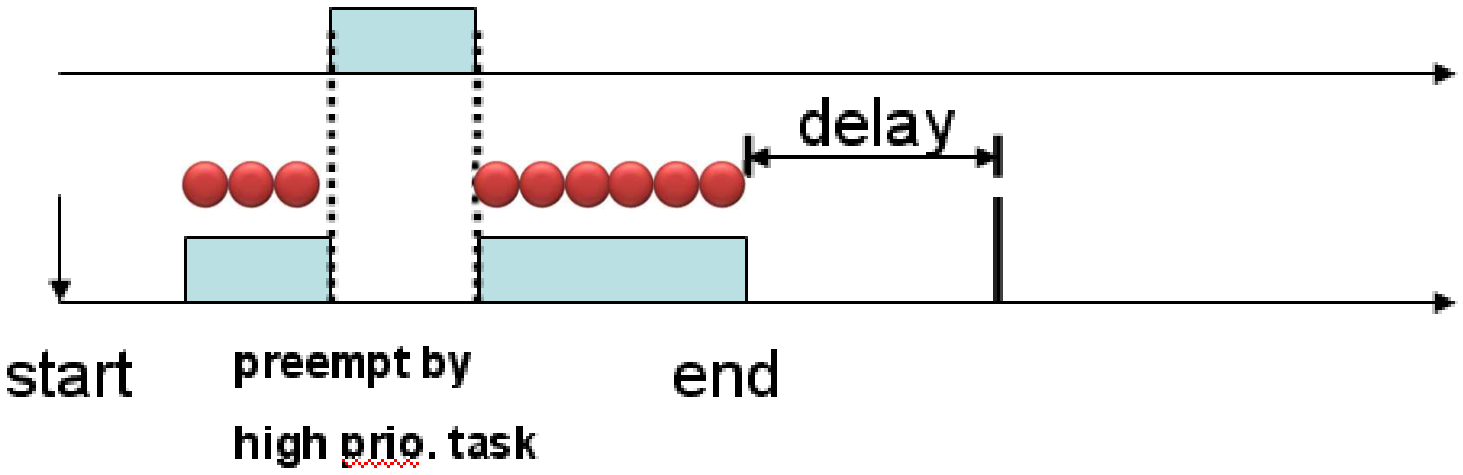
Periodic task

- Problem : the task does not receive “c” units of CPU time if it is preempted in [stard,end]



Periodic task

- c = clock ticks actually spent on the task; but c may be smaller than (end-start)
- *delay* is still p-(end-start) if c units of time are correctly consumed

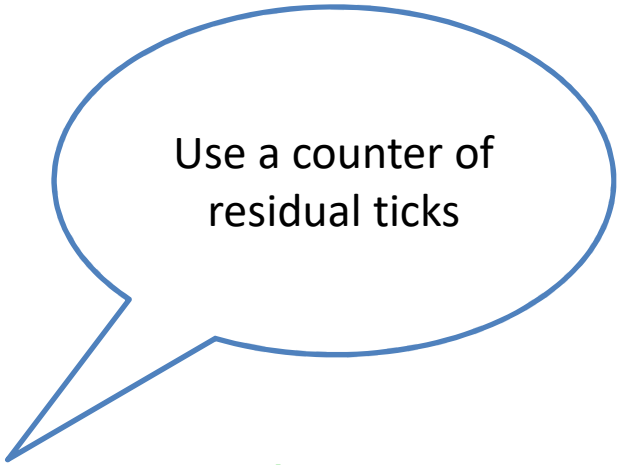


Idea

- Use an “execution counter”, just like the “delay counter”
  - Decrement when the corresponding task uses 1 tick of CPU time
- Struct OS_TCB
  - A per-task data structure, defined in uCOS-II.h
  - Add a variable compTime to store the residual clock ticks of a task
    - replenished to “c” at the beginning of every period
  - Add a variable of task period

Periodic task

```
void Task()
{
    int start ;  //the start time
    int end ;    //the end time
    int toDelay;
    start=OSTimeGet();
    while(1)
    {
        while(OSTCBCur->compTime>0)          //C ticks
        {
                  // do nothing
        }
        end=OSTimeGet() ;                    // end time
        toDelay=(OSTCBCur->period)-(end-start) ;
        start=start+(OSTCBCur->period) ;     // next start time
        OSTCBCur->compTime=C ;// reset the counter (c ticks for computation)
        OSTimeDly (toDelay);                 // delay and wait (P-C) times
    }
}
```



OS_ENTER_CRITIAL and OS_EXIT_CRITICAL should be used to warp the access to OSTCBCur->compTime

OSTimeTick

- OSTimeTick()
  - Defined in OS_CORE.C, called every time when a clock interrupt arrives
  - Add a piece of code in OSTimeTick to decrement the compTime counter in the running task's os_tcb
    - The current task has consumed 1 tick

OSInitExit

- OSIntExit()
  - Defined in OS_CORE.C
  - This function will manage the scheduling after the system has come back from the calling of ISR
  - We need to print out the “preempt” event here

OS_Sched

- OS_Sched()
  - Defined in OS_CORE.C
  - OS_Sched() is called when a task is voluntarily giving up its possession of the CPU
  - We need to print out the “complete” event here

Related Function

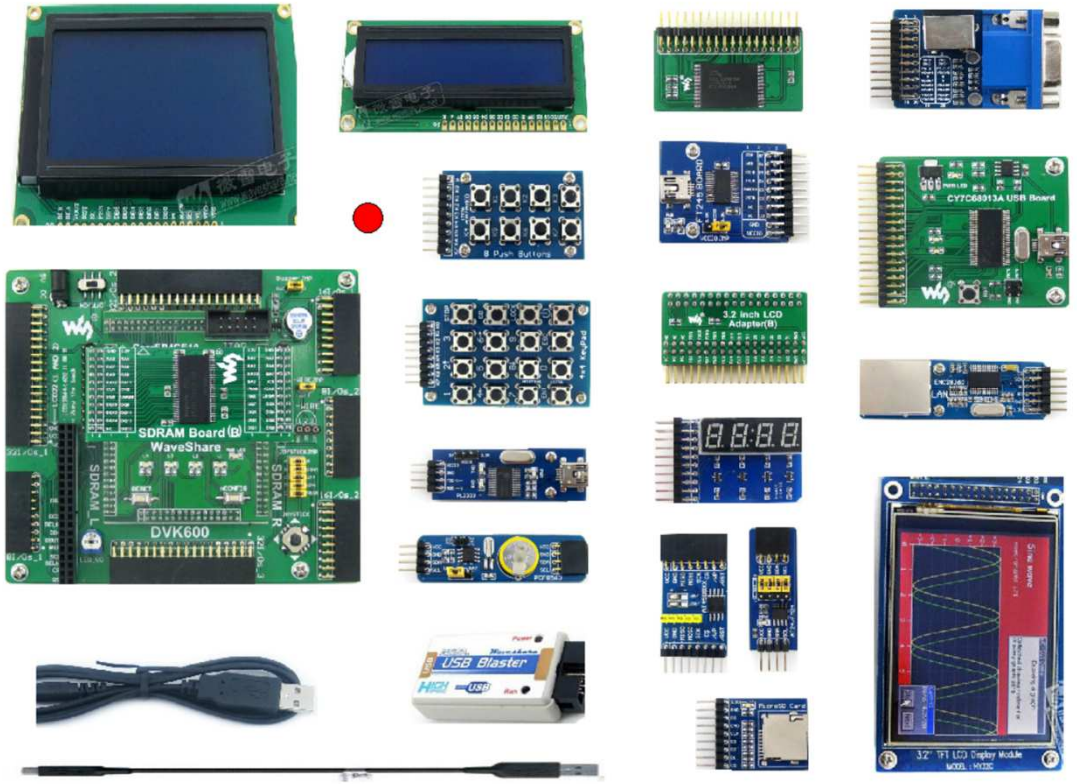
- OSStart():
  - This is function will try to find the task with the highest priority and schedule it to run.
  - Called only once when the system executing tasks for the very first time
  - This function is defined in OS_CORE.C

Printing messages

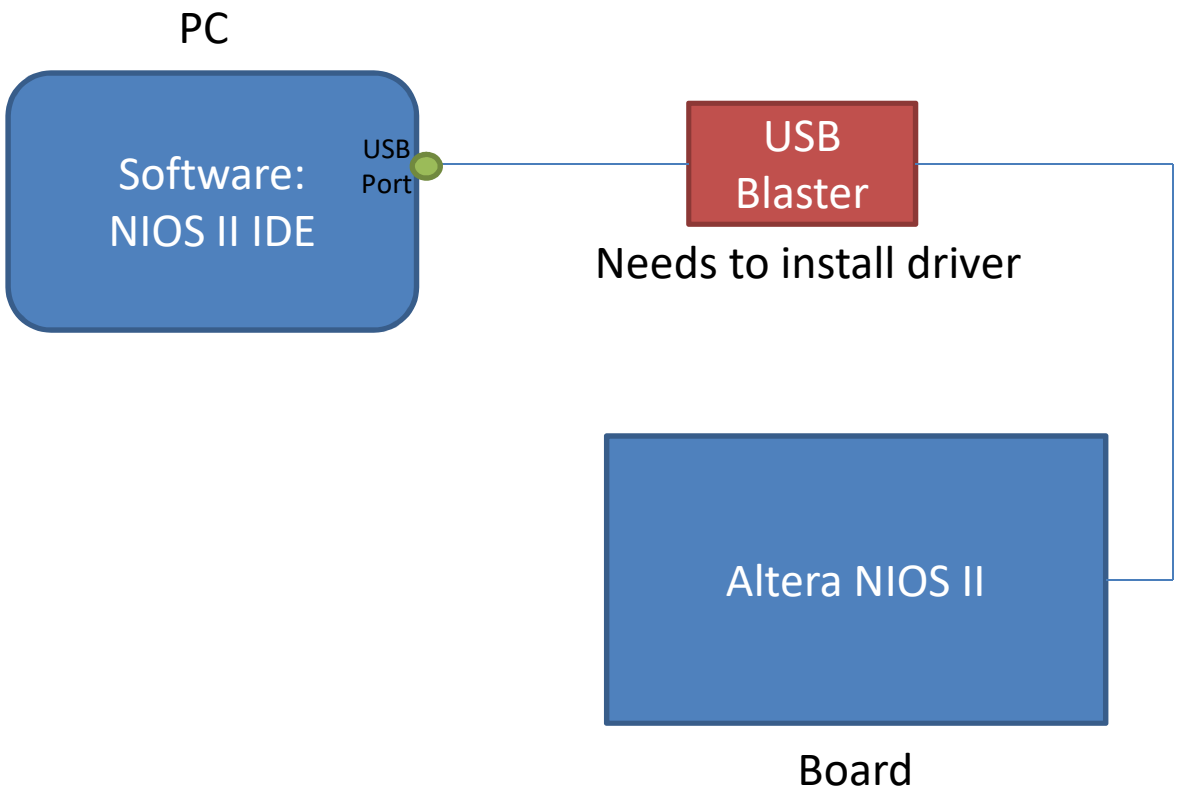
- Print messages
  - There's a printf that you can use (board)
    - E.g., printf("\n%10d Preempt ",timestamp);
  - Use PC_DispStr() in Dosbox
    - Display buffer for legacy PC
  - Do not call printf inside of an ISR, it may sleep
  - Save outputs in a buffer and have a task print the results
  - Properly use critical sections to protect the buffer

Evaluation boards

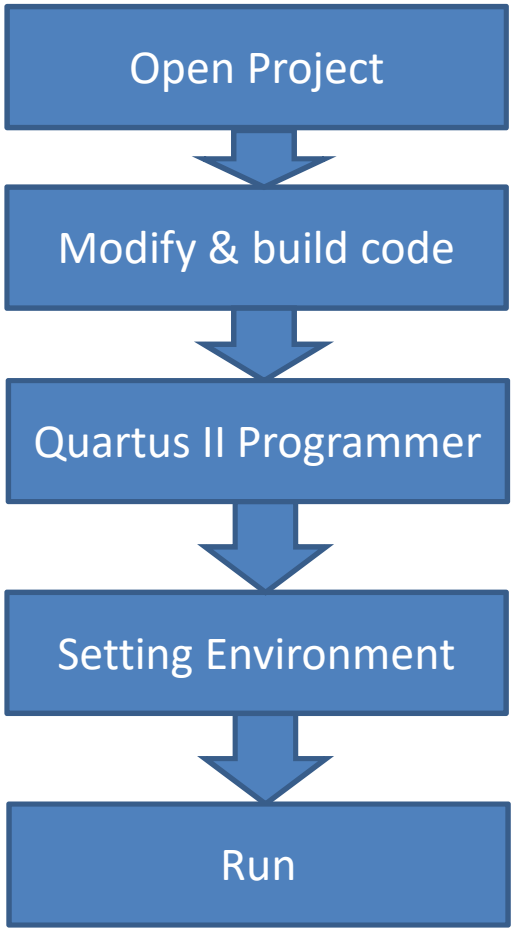
- Altera NiosII



Architecture
Altera NIOS II



Board: Altera NIOS II



Open Project

- Open NIOS II -> set workspace: C:\cps\workspace
- Create Project
  - File -> New -> Nios II Application and BSP from Template
- New project setting
  - SOPC Information File name -> C:\cps\workshop\nios2ucosii\CORE_SOPC.sopcinfo
  - Select Project Template: Hello MicroC/OS-II
  - Finish

Porting: Quartus II Programmer

- NIOS II programmer
  - NiosII -> QuartusII Programmer
- Quartus II setting
  - Add File
  - Select “C:\cps\workshop\nios2ucosii\standard.sof”
- Hardware Setup -> USB-Blaster
- Start
- Close Quartus II

Run

- click Run 
  - Select NiosII Hardware
  - Target Connect -> Refresh connection -> Apply -> Run

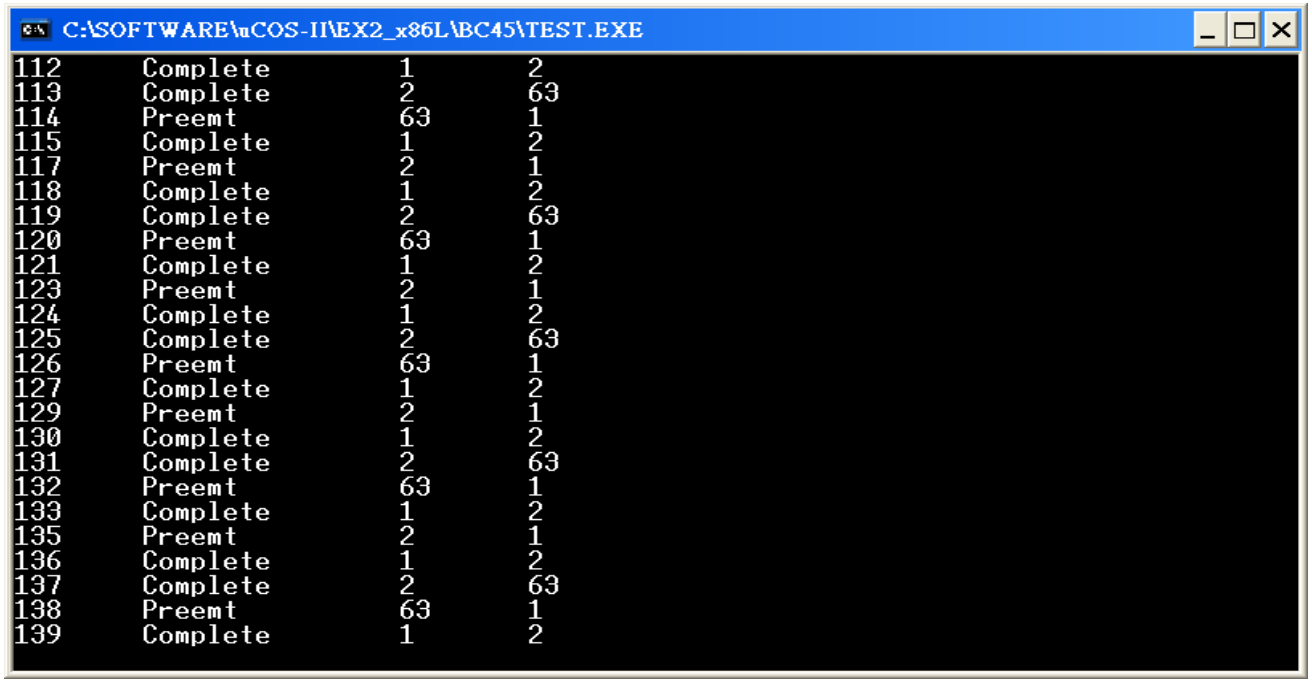
Output Results

- expected output:

Current time Event [From Task ID] [To Task ID]

Time tick Preempt TaskID(priority) TaskID(priority)

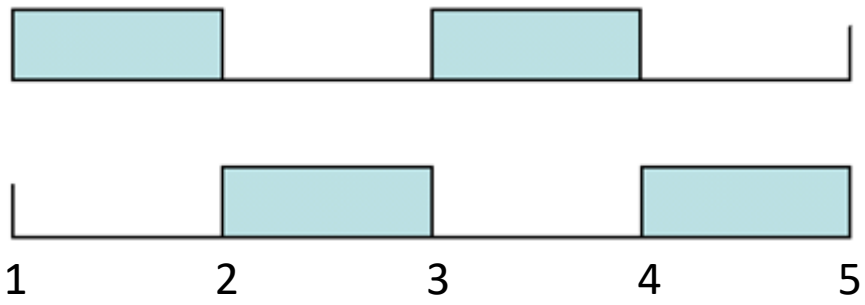
Time tick Complete TaskID(priority) TaskID(priority)



Output Results

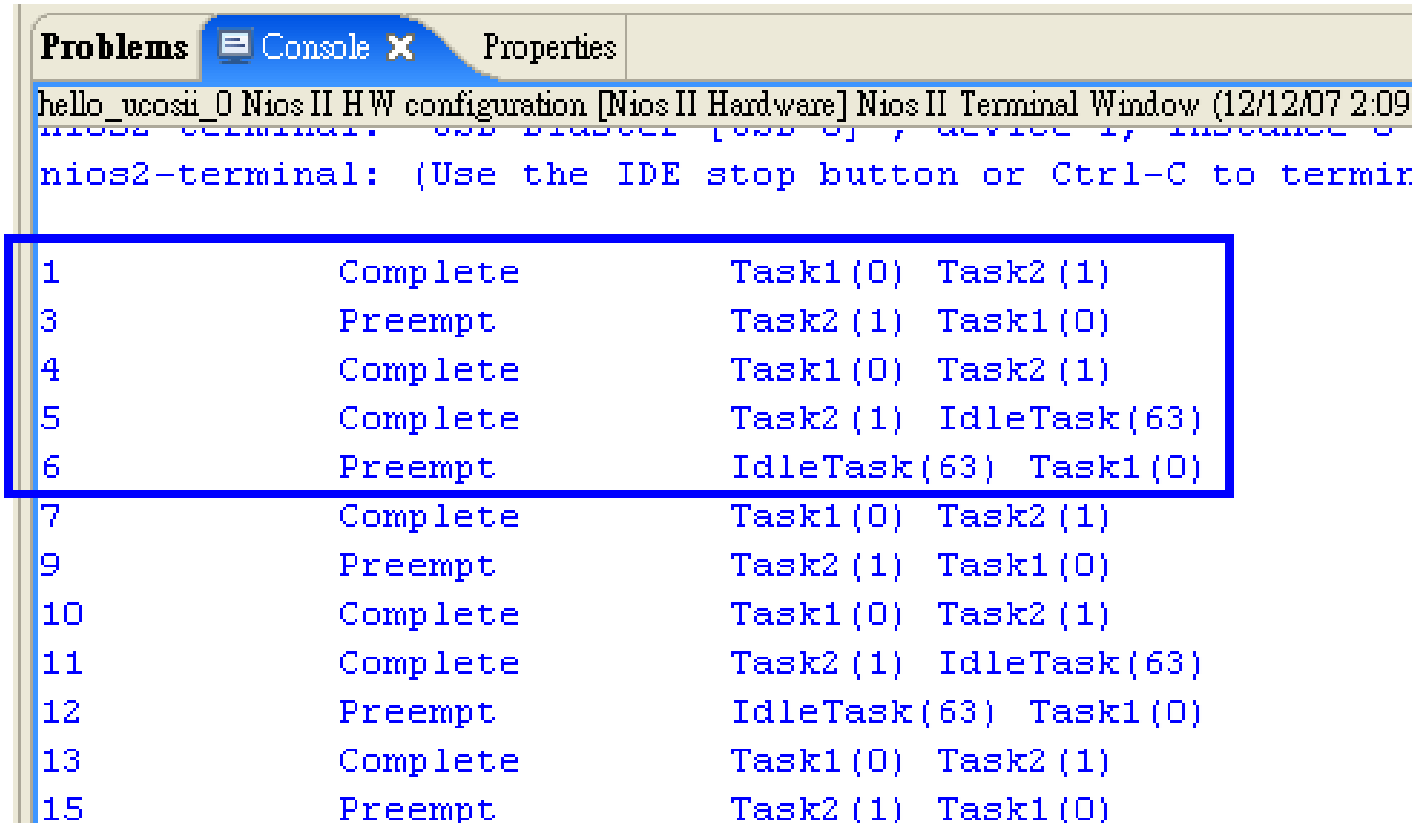
- Example Taskset ={t1(1,2),t2(2,4)}
  - Suppose program start at time tick 1
    - System time is the “OStime” global variable

Time	**event**	**from**	**to**
1	Preempt	63	1
2	Complete	1	2
3	Preempt	2	1
4	Complete	1	2



Output Results

altera NIOS II



More Information

- Remember to save your code for further use in Lab 1 and 2
- You can use OSTimeSet(0) to reset the tick counter if necessary
- Classroom: EC222 (!)

Grading

- Produce the correct schedules for the following tasks using RM
  - { (1,3), (3,6) }
  - { (1,3), (3,6), (4,9) }