COSE321 Computer Systems Design

Mini-Project

No late turn-in accepted

In the assignment, you are going to design a mini-OS (Operating System); mostly low-level interface between hardware and high-level programs. It may sound overwhelming. But, the project is actually very simple. Three different user applications (task1, task2, and task3) written in C are given to you. Your job is to run the tasks continuously in a round-robin fashion, one at a time, on Zynq.

Download the application programs from the link below:

http://esca.korea.ac.kr/teaching/cose321_CSD/assignments/extra-credit/tasks.7z

Each application runs forever (i.e., never exits). Take a look at the application programs. The following is a brief description;

- Task1: Selection-sorting. Its outcome is displayed on LEDs.
- Task2: DhryStone Benchmark. It is used to measure the CPU performance. The benchmark is composed of several functions. Each function name is printed via UART after execution. After the execution of the whole benchmark, a message 'Task2 finished' will be printed on UART console.
- Task3: Simply send 'Hello World' to UART console.

The assignment requirements are as follows;

- System Initializations (in **Supervisor mode**)
  - Stack setup for each mode
  - GIC configuration
  - Private Timer configuration
  - UART configuration
  - TCB (Task-Control Block) setup for each user program (See page#3)
  - Stack setup for each user program

- Task Scheduling (in **IRQ mode**)
  - Set the timer interval to 1ms for task scheduling
    - Some people refer to it (periodic timer interrupt) as **tick**
  - Context switch in ISR upon private timer interrupt
  - Task priorities are the same for 3 tasks
  - Use the round-robin mechanism for the task scheduling (task1 → task2 → task3→ task1→ task2 …)

Note that the application programs run in **User mode**.

What and How to submit:

1. Upload your source code to Blackboard.
2. Upload video clip (5-min?) to YouTube and provide the link to Blackboard. Your video clip should have **at least** the following contents:
    - Your smiling face
    - Understandable explanation of the mini-OS operation and your code (system initialization, context switching, etc)
    - Demo on Zedboard

Note: This is an individual project. You are welcome to discuss, but DO NOT COPY solutions. **The plagiarism will not be tolerated!**

- Task Control Block (TCB) or Process Control Block (PCB)

When a task is created, it is assigned a task Control Block (TCB). TCB is a data structure that is used by OS to maintain the state of a task when it is preempted. When the task regains control of the CPU, the TCB allows the task to resume execution exactly where it left off. The TCB is initialized when a task is created. An example TCB structure used in µC/OS-II (a Real-Time Operating System) is shown below

```
struct os_tcb {

   OS_STK         * OSTCBStkPtr ; // a pointer to the current top-of-stack for the task
   struct os_tcb * OSTCBNext;  // for doubly linked list of TCBs
   struct os_tcb * OSTCBPrev;  // for doubly linked list of TCBs
   INT16U        OSTCBDly;     // a task needs to be delayed for this amount of clock ticks
   INT8U         OSTCBStat;    // task state: Dormant, Ready, Running, Waiting
   INT8U         OSTCBPrio:    // task priority

} OS_TCB
```

In this class project, things are much simpler because we have only 3 tasks, all tasks are always ready to run, and the priorities of the tasks are the same. Thus, it would be enough to save the following registers to TCB when context-switched.

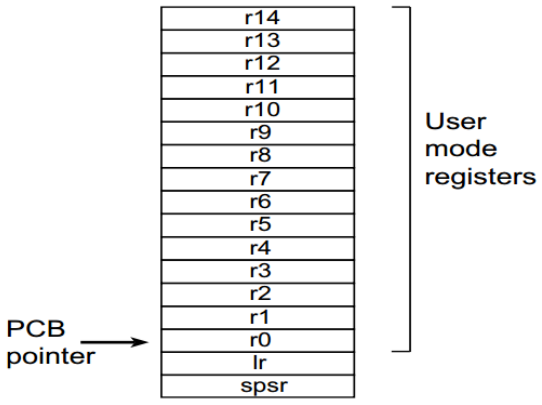

Figure 6-2 PCB layout

- One thing you have to be careful: The fact that User mode registers (because the tasks are running in User mode) have to be stored in PCB means that the ISR (or scheduler) in Interrupt (IRQ) mode should have access to User mode registers. To make your life easier, ARM provides optional suffix (^) for LDM and STM. With that suffix, the User mode registers are accessible in other (IRQ, FIQ…) mode instead of the current (IRQ, FIQ…) mode registers.

For example, assume that the following `STM` instruction is executed in ISR (Interrupt Service Routine), which is running in IRQ mode.

`STM sp, {r0-lr}^` // store r0 ~ r14 **User-mode registers** to stack.