AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
Credit Hours Programs

Summer Semester, 2021

CSE 345 / CSE 347 : Embedded Systems

Lab 2: Creating task stacks for switching

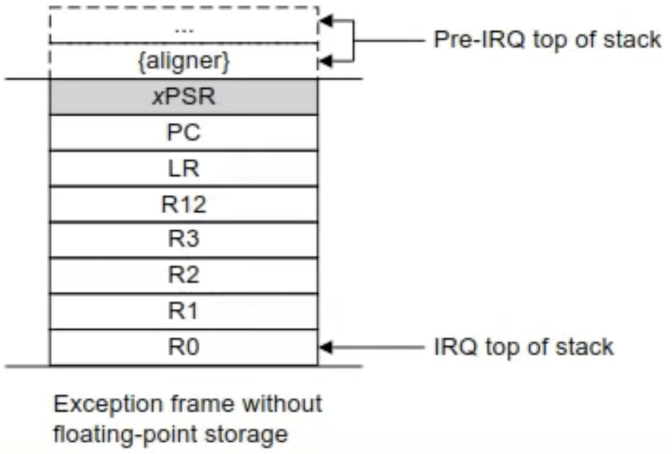
Goals of this Lab:

Real time operating systems are composed of various tasks. Tasks are infinite loop functions that serve certain functionalities. The operating system Kernel switches the processor resources (processing time, hardware peripherals, etc.) between tasks. The goal of this lab is to implement a code that does the context switch. This code has to switch between two tasks.

Introduction:

What we did last lab which is manually switching by changing the value of the PC is not quite legal. It would also not work in larger applications. Even more, it would cause some really serious problems. We also still need to have the code run automatically without interfering every time we need to switch tasks.

To do so we need to look at what happens in the context switching exactly, and what happens exactly in the stacks. As per the TM4C datasheet please find the frames that are added to the stack while doing the context switch of an exception. The screenshot attached is valid if we turned off the Floating point Unit (FPU) from the project settings.



ARM exception frame layout (without the FPU)

NOTE: The stack pointer grows upward in the ARM which means the top of stack would be the address with the lower value. A hint : You may need to flip the image above to match the stack if the values of the stack are put in ascending order.

Task 1: Use the code created in Lab 1

Open the lab 1 progress. Make sure you have the systick timer with the interrupt. Make sure also you have two tasks one of the RED led and the other for the BLUE LED.

Task 2: Create a stack for each task:

Create a stack that could be used to store the values used for each task. You can create the stack as an array of 40 Uint32 elements because the stack is mainly a piece of memory we need to reserve assembled. We also need a pointer that shall point to the top of the stack exactly like what the SP does.

```
/* Example for Task 1 */
uint32_t stack_RedBlinky[40];
uint32_t *sp_RedBlinky = &stack_RedBlinky[40];  /* The stack pointer is initialized to pint one word after the stack because the stack grows down that is from the end of the stack array to its beginning */
```

Now do the same step above also for the second task.

Task 3: In the main function, fill the stacks created with the data to make it look as if it was preempted by an interrupt

According to the ARM exception frame layout attached above. We shall start from the high memory end of the stack because it grows from high to low memory.

Refer to the datasheets for the positions of the bits.

Step 1: Pre decrement the pointer to get to the first address location, we then need to write the THUMB bit in the PSR register

Step 2: Pre decrement the pointer again this time to write the PC. In C you can get the start address of a function using it's name (ex: &fun1)
You need also to cast the pointer to make sure it fits inside the pointer (ex: (uint32_t)&fun1)

Step 3: The rest of the registers would not really matter a lot in the switch so for testing we shall set them to values we know so that we can make sure these values are already copied to the registers.

Step 4: Do steps 1 to 3 again, but this time for the second task. (Blinky LED 2)

Step 5: Create an infinite loop at the end of the main function.

Task 4: Seeing the stacks in the memory view

Step 1: Look at how your stacks are allocated in memory and make sure you have the correct values in each place. (Screenshot_1)

Step 2: Put both of your created stack pointers in the watch window. (Screenshot_1)

Step 3: Put a breakpoint in your systick Interrupt handler. Then wait till the debugger hits the breakpoint.

Step 4: Manually change the value in the SP register to the stack pointer of Blinky1,remove the breakpoint, and watch as the debugger goes to the Blinky1 function. (Screenshot_2)

Step 5: Now to switch to Blinky2, put back the breakpoint in the ISR. Now, we first need to take the current value in the SP register and put it inside the created stack pointer for Blinky 1 because that value is now the top of stack for Blinky1. After that we can put the value of Blinky2 stack pointer in the SP register. (Screenshot 3)