MTE 241 RTOS Project

Configure uVision5 project:

- Project > New uVision Project
- Select Device NXP LPC1768
- Manage Run-Time Environment dialog box
    - expand Device; check Startup
    - expand CMSIS; click CORE
    - do not choose an RTOS
    - click OK
- double-click Target 1 > Device > startup_LPC17xx.s
    - choose Configuration Wizard tab
    - change Stack size to 0x2000 (= 8 kiB)
    - leave Heap size at 0x0
- right-click Target 1 > Options for Target 'Target 1'
    - in C/C++ tab Define __RTGT_UART
    - do not select Use MicroLIB in Target tab
- right-click Target 1 > Source Group 1
    - add existing files Retarget.c, uart.c, main_default.c, context.c

Background

The Cortex-M3 has two processor modes: Handler mode and Thread mode. Handler mode is used automatically for exception handlers (except the Reset handler) and has privilege level Privileged: that is Handler mode code can execute privileged and unprivileged instructions. Application code executes in Thread mode and can have privilege level Unprivileged or Privileged. There are two stack pointers: the Main Stack Pointer (MSP) and the Processor Stack Pointer (PSP). Handler mode uses the Main stack. Thread mode can use the Main stack or the Process stack depending on the SPSEL bit in the Control register: 0 = MSP, 1 = PSP.

After reset the thread mode defaults to privilege level Privileged and the Main stack. We will leave the privilege level alone but will change the stack to the Process stack. The Main stack will be used for exception handlers and we'll leave it 2 kiB. The tasks will each have their own stack that is 1 kiB in size. Since we configured the stack size to 8 kiB that means that we can have up to 6 tasks. Here is the memory map:

Table 1 RTOS Memory Map

starting address		size
00000000	Text Section	
10000000	Data Section	
	TCB[6]	
	Stack 0	1 kiB
	Stack 1	1 kiB
	Stack 2	1 kiB
	Stack 3	1 kiB
	Stack 4	1 kiB
	Stack 5	1 kiB
	Main stack	2 kiB

The Task Control Blocks (TCBs) are actually part of the Data section (you should declare them global variables). The base address for the Main stack is found in Vector 0 of the Vector Table at address 0x0.

Initialization

You will need to provide an initialization function for your RTOS that you will call from main(). It will initialize each TCB with the base address for its stack. It is up to you to define the contents of the TCB. The initialization function will also change the Thread mode stack from the MSP to the PSP and when it returns `main()` will be one of the tasks[1]. This will require copying the stack contents from the Main stack to the stack of the `main()` task. Steps:

1. Initialize each TCB with the base address of its stack. You can work backwards from the base address of the Main stack from Vector 0.
2. Copy the Main stack contents to the Process stack of the new main() task and set the MSP to the Main stack base address.
3. Switch from using the MSP to the PSP by changing the Stack Pointer Select (SPSEL) bit in the Control register. The Stack Pointer register (R13) is a banked register. Before switching the SPSEL bit it will have the address of the MSP. After switching it, it will have the address of

1 Alternatively, the initialization call could not return and main() does not continue. I'm not sure which is easier.

the PSP which will not contain a valid address. You should change it to the top of the stack that you selected for your `main()` task.

CMSIS provides “intrinsics” (functions that expose processor-specific assembly instructions). Use the Core Register Access functions (https://www.keil.com/pack/doc/CMSIS/Core/html/group__Core__Register__gr.html) to access the MSP, PSP and Control registers. There is a CONTROL_Type union that you can use when setting the SPSEL bit in the Control register.

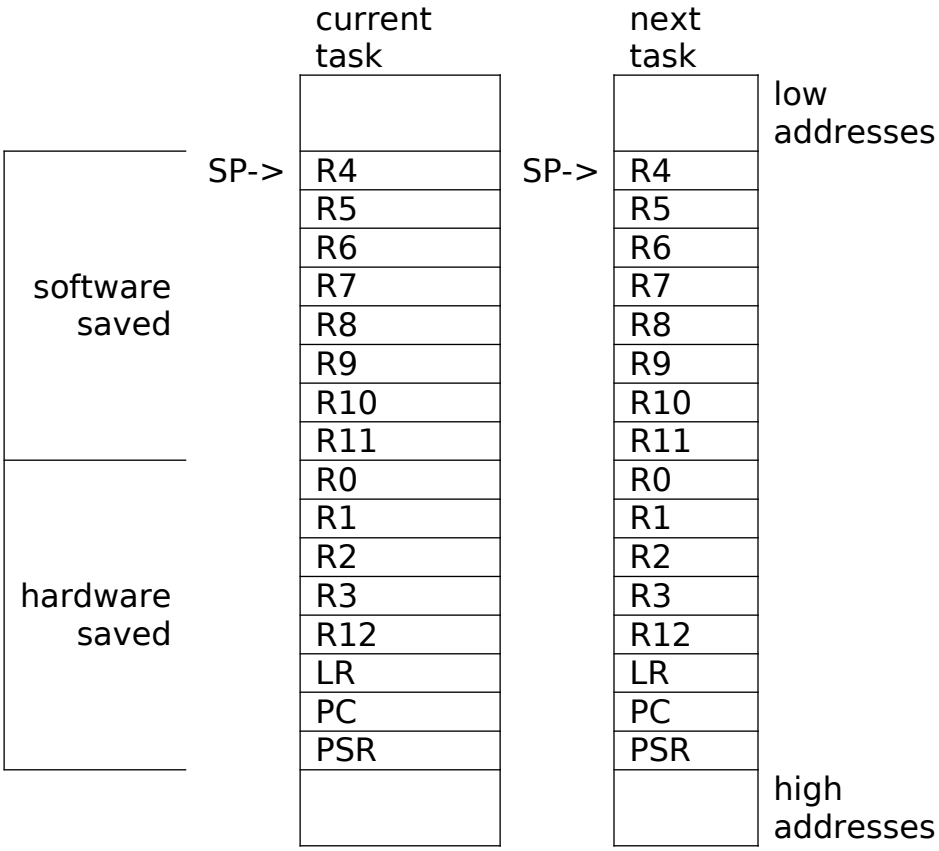
Context Switch

To perform a context switch you:

1) push register contents onto the current task’s stack
2) record the current stack pointer address in the current task’s TCB
3) load the next task’s top of stack address into the stack pointer
4) pop register contents from the next task’s stack

On ARM Cortex-M3 you implement the context switch in the PendSV_Handler(). When the Cortex-M3 enters an exception handler it pushes these eight registers on the stack R0-R3,R12,LR,PC,PSR (LR =R14 = link register for procedure call return, PC =R15 = program counter, PSR = processor status register). When it exits the exception handler it pops from the stack into those registers. To perform a context switch then, the software just needs to push/pop R4-R11 to/from the stack and manipulate the SP (SP = R13 = stack pointer). This is pictured below.

Table 2 Context on Task Stack



Two functions are provided to help with context switching:

```
uint32_t storeContext(void);
void restoreContext(uint32_t sp);
```

`storeContext`() returns the value of the stack pointer after pushing R4-R11 onto the stack. restoreContext() takes the address of the top of the task to switch to as a parameter and pops into R4-R11 after changing the SP.

Create Task

You will need to provide a function to create new tasks. The parameters should include a pointer to the task function, and a task function argument. The function pointer type can be declared as:

```
typedef void (*rtosTaskFunc_t)(void *args);
```

`rtosTaskFunc_t` is the type name for a pointer to a function that takes a void * parameter and returns void.

When you create a new task, you need to initialize its stack according to Table 2. The only parts that matter are:

- R0 – argument to task function
- PC – address of task fuction
- PSR – default value is 0x01000000

However the other parts (R4-R11, R1-R3,R12,LR) must exist so that the context pop works correctly.

SysTick Handler

The `main_default.c` that is provided shows how to configure the SysTick to 1ms and declare the `SysTick_Handler()`. You will need to modify the SysTick handler to suit your purposes including triggering the PendSV exception when a context switch is required. To do this you need to write a 1 to the PENDSVSET bit of the Interrupt Control and State Register (ICSR). This register is described in Table 4-15 of the Cortex-M3 Devices Generic User Guide found at http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/index.html. By including the LPC17xx.h header file you can reference the ICSR as `SCB->ICSR` in your code.

Deliverables

- Inform Andrew Morton and Bernie Roehl by email if your team intends to implement an RTOS instead of a game for Lab 5. Andrew will arrange the time and place for your Lab 5 demo and will do the marking.
- Your RTOS should implement the following: initialization, task creation, context switching, fixed-priority preemptive scheduling, blocking semaphores, and mutexes with owner test on release and priority inheritance.
- Create a test suite that demonstrates the following:
  - context switching between at least two tasks 40%
  - FPP scheduling 20%
  - blocking semaphores 20%
  - mutex with owner test on release (10%) and priority inheritance (10%)