**CG2271 Real Time Operating Systems**

**Assignment 1**

|  |  |
| --- | --- |
| Member 1 Matric: A0077863N | Member 1 Name: Song Yangyu |
| Member 2 Matric: A0078040M | Member 2 Name: Li Ruize |

*Please keep to under 6 pages. Anything past page 6 will not be marked. So make judicious use of this space. For example you won’t want to be dumping large chunks of code. ☺ You may delete all italicized instructions.*

1. Introduction

*Talk about the objectives of this assignment, as you understand it. Don’t cut and paste from the assignment* *sheet:*

Implement a multi-task kernel for small RTOS called AVROs

1. The Co-operative Multitasker

*Talk about how you create tasks, and how you swap them around. What happens if a task if being run for the first time and you have no stack pointer information? How do you start up such a task?*

In the struct “TTaskBlock”, I added in one more field:

char stack[TASK\_STACK\_SIZE];

where I define “TASK\_STACK\_SIZE” to be a big enough size to hold a task function.

*Besides, I also* defined two global variable in the system:

int task\_counter = 0;

int currentTask = -1;

When I add task, I’ll assign the place of task\_counter to the task I’m creating, set the corresponding parameters, and I’ll then increase the task\_counter by one.

When I first start running a task (which has been never run before), I’ll call a function (defined by myself) called

void runTaskFirstTime(int i)

This function would set the StackPointer to the place I defined for this task, and run the task there.

When I want swap a task, I’ll first check if the “currentTask” has been defined, and try to get the next task to run. If either case is false, the swap task would not carry on.

After checking, I’ll save context by calling

portSAVE\_CONTEXT();

After this, the value of stack pointer has been saved into pxCurrentTCB. So I’ll save this value into the old task (being pre-empted)’s stack\_ptr, then update the value of current task, and try running the new task.

If the new task has never been run, it’s the same case as first running a task (which has been covered previously); if the task has been run, I’ll load the stack\_ptr into pxCurrentTCB, and call:

portRESTORE\_CONTEXT();

to restore the pointer from the position of stack\_ptr of that task. Besides, this function would also jump to the position of the next task in the assembly code, so that the pre-empted task can be run directly.

*Also talk about how you choose the next task to run.*

using “Round Robin”: nextTaskNum = (current\_task + 1)%task\_counter

*Describe all the tasks you created:* I created as required by the assignment 1 description.

*Task 1: generates the buzzing tone*

*Task 2: reads ADC channel 1, and maps the reading to an appropriate range for LED.*

*Task 3: reads ADC channel 0, and map the reading to an appropriate range for the buzzer*

*Task 4: blinks the LED at specified rate in Hz*

1. The Pre-Emptive Multitasker

*Talk about the changes you made and why. Are there atomic sections in your operating system and task code? Explain why.*

1. I made counter 0 as the interrupt for task switching. In order for the task to switch every 50ms, I set the pre-scalar to be 8, and count 100 numbers to trigger the interrupt.
2. In the “portSAVE\_CONTEXT” and “portRESTORE\_CONTEXT”, I deleted cli and sei – because the interrupt would be disabled before calling these function, and it needs to remain disable after calling them.
3. I added cli in the ISR, and calls OSSwapTask; in the OSSwapTask I’ll keep the interrupt to be disabled until the point when I call portRESTORE\_CONTEXT, or call runTaskFirstTime.
4. At the end of runTaskFirstTime, I’ll call sei() to enable interrupt
5. In task 1 (generating sound by buzzer), the process of generating the sound based on frequency would be made atomic by calling OSMakeAtomic();
6. Task 2 and task 3 should be both made atomic – they are for reading the ADC and calculate the corresponding range