ECE350 Lab4 Report

Group 57

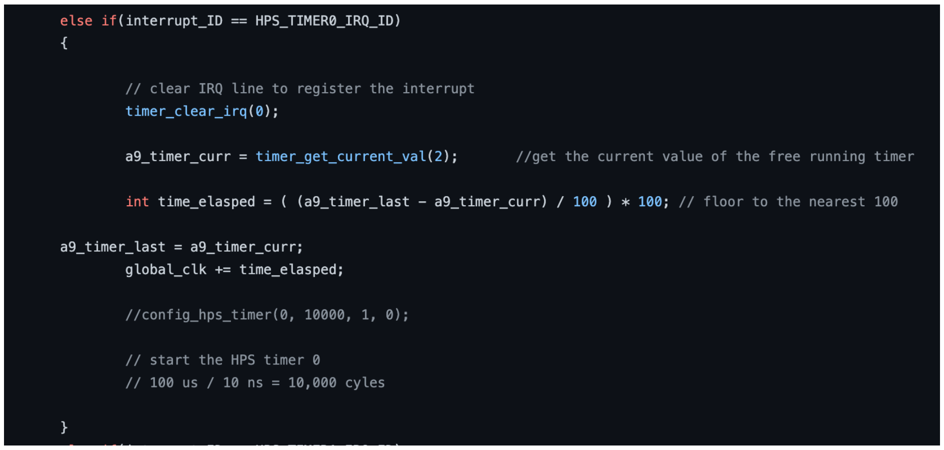
Algorithms & Data Structures

Timers

The A9 timer is used to update the global clk variable in our implementation. The C_IRQ_HANDLER function gets the current time value and subtracts it from the last recorded time to get the elapsed time and add it on to the global clock.

HPS Timer 0 is used to interrupt the program at each time quantum to preempt the scheduler based on the updated deadlines for each real-time task. Before the preemption, the global clk is updated.

The global_clk variable is then used to check for missed deadlines in the k_tsk_done_rt function.

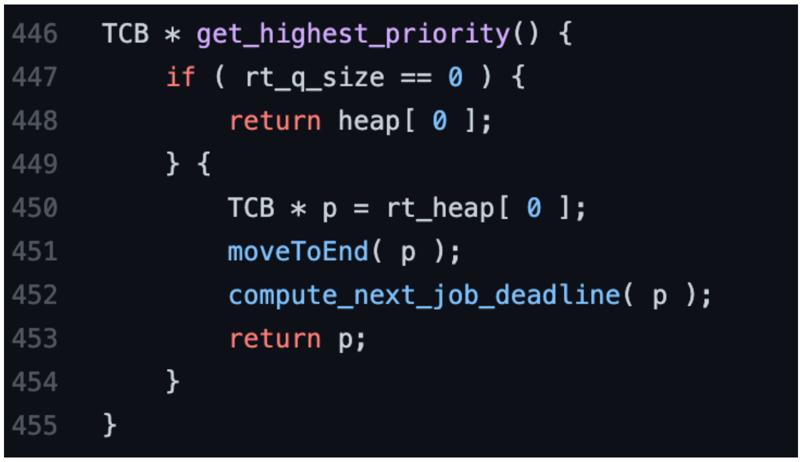


```
    else if(interrupt_ID == HPS_TIMER0_IRQ_ID)
    {

            // clear IRQ line to register the interrupt
            timer_clear_irq(0);

            a9_timer_curr = timer_get_current_val(2);       //get the current value of the free running timer

            int time_elasped = ( (a9_timer_last - a9_timer_curr) / 100 ) * 100; // floor to the nearest 100

    a9_timer_last = a9_timer_curr;
            global_clk += time_elasped;

            //config_hps_timer(0, 10000, 1, 0);

            // start the HPS timer 0
            // 100 us / 10 ns = 10,000 cyles

    }
```

This is where the global_clk is updated upon the interrupt of the HPS Timer 0. This interrupt occurs after every time quantum.

EDF Scheduling

We then implement the EDF scheduling policy in our scheduler using two heaps. One heap has the non real-time tasks, and the other has real-time tasks. The real-time task heap is a priority heap based on the periods of the tasks and the updated deadlines. The scheduler is preempted every time quantum and updates the task deadlines, before choosing the next task to run.



```
TCB * get_highest_priority() {
    if ( rt_q_size == 0 ) {
        return heap[ 0 ];
    } {
        TCB * p = rt_heap[ 0 ];
        moveToEnd( p );
        compute_next_job_deadline( p );
        return p;
    }
}
```

This shows the two different heaps and how they're handled. If it is a real-time task heap, then the next job deadline is calculated.

Test Cases

Case 1: Create two real-time user tasks. (one with mailbox size non-zero)

Here we are testing the creation of real-time tasks and making sure they are initialized with the right parameters. We are also testing for the successful allocation of memory for the mailbox when the size is greater than or equal than the minimum size needed to create a mailbox. If the mbx_size_byte variable in the RTX_TASK_INFO struct is 0, then a mailbox should not be created.

We are also looking to see that the two real-time tasks are populated in the real-time heap for the scheduler and one of them is chosen to be the first task run by the scheduler.

Case 2: Create two real-time user tasks with different periods.

Here we are testing the timer and interrupt functionality of our implementation with two real-time user tasks. One has a period of 2 seconds while the other is 5 seconds. The task with the shorter period should be the first to run, with an interrupt right after preempting the scheduler. The preempt should then go to the next task for that task's time period, before getting preempted again and switching to the other task.

We should see the global clk variable being updated as the tasks run and switch.

Case 3: Create two real-time tasks, one being a privileged task and the other being a user task.

Here we are testing the timer and interrupt functionality when there is a privileged task running and the interrupts are disabled. In this case, once the privileged task runs, it cannot be interrupted when the HPS 0 timer goes to 0. It can only be exited using yield, done_rt or tsk_exit.