EEE8087 – Coursework Specification

Coursework supervisor: Alex Bystrov

Write a program for the supplied ARM-XILINX development board and implement a *Time Triggered Cooperative Scheduler* (TTCS) running several *short periodic tasks* and having protection from *overruns*.

Aim: Understanding of the concept of real-time scheduling as an approach to implementation of *concurrent* computational processes in a simple embedded system.

Platform: ARM-XILINX development board with a single ATMEL AT91SAM9261 microcontroller, custom monitor-debugger KMD and the tool chain based on GCC compiler. Programming language – C.

Scheduler specification: timer clock period 1ms, eight short periodic tasks, four of them are executed in every timer interval, two tasks are executed in every odd timer interval and two tasks are executed in every even timer interval. In the case of an overrun, the overrunning task is interrupted by the same timer, which is followed by the sequence of task execution in the next timer period. The tasks positioned in the queue after the faulty overrunning task have no chance to run, thus increasing the impact of the fault. Reorder the queue upon detecting the fault and place the faulty task at the end of the queue, thus allowing the other tasks to run. Test the overrun protection mechanism and demonstrate that it works. As ARM processors have multiple modes of operation, use the supervisor (SVC) mode in the schedule, switch back to SVC mode after the interrupt switches it to IRQ mode, use the user mode (USR) to execute the tasks.

Recommendation: as the first step, implement the TTCS scheduler without overrun protection. Use the provided FSM model as the top-level specification of the algorithm of the scheduler. Add overrun protection at a later stage. Find a way of detecting the overrun. A hint – look at the queue index at the time of interrupt. Reorder the queue in Update block. Implement fault injection by using a conditional loop placed in the task code, which will cause undue delay under test conditions. Implement the task context (data) as a global array of structures. Implement task queue as a global array of task numbers and a dedicated global variable pointing at the next task to run (index of the next array element).

Task specification: implement eight very similar tasks, each of them controlling a single LED. The LEDs must flash with the periods of 1.0s, 1.1s, 1.2s, 1.3s, 1.4s, 1.5s, 1.6s and 1.7s correspondingly. In each period of flashing an LED must gain and reduce its brightness gradually. Use *Pulse Width Modulation* (PWM) with the duty cycle gradually rising and gradually falling with the above flash periods. Use the *PWM period* of 20ms in order to exploit the inertia of human vision. Implement the maximum possible number of brightness levels for each task.

General comments: The first step is to find how to represent a continuous process as a sequence of short periodic tasks. It is a good idea to start debugging the system with only two tasks. Thus, you will avoid possible problems with overruns in case of suboptimal code. Switch to a very slow clock when debugging the PWM. Try to move the time consuming computations away from the critical cycle.