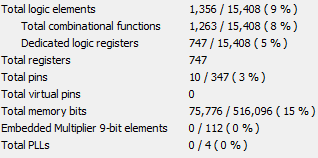
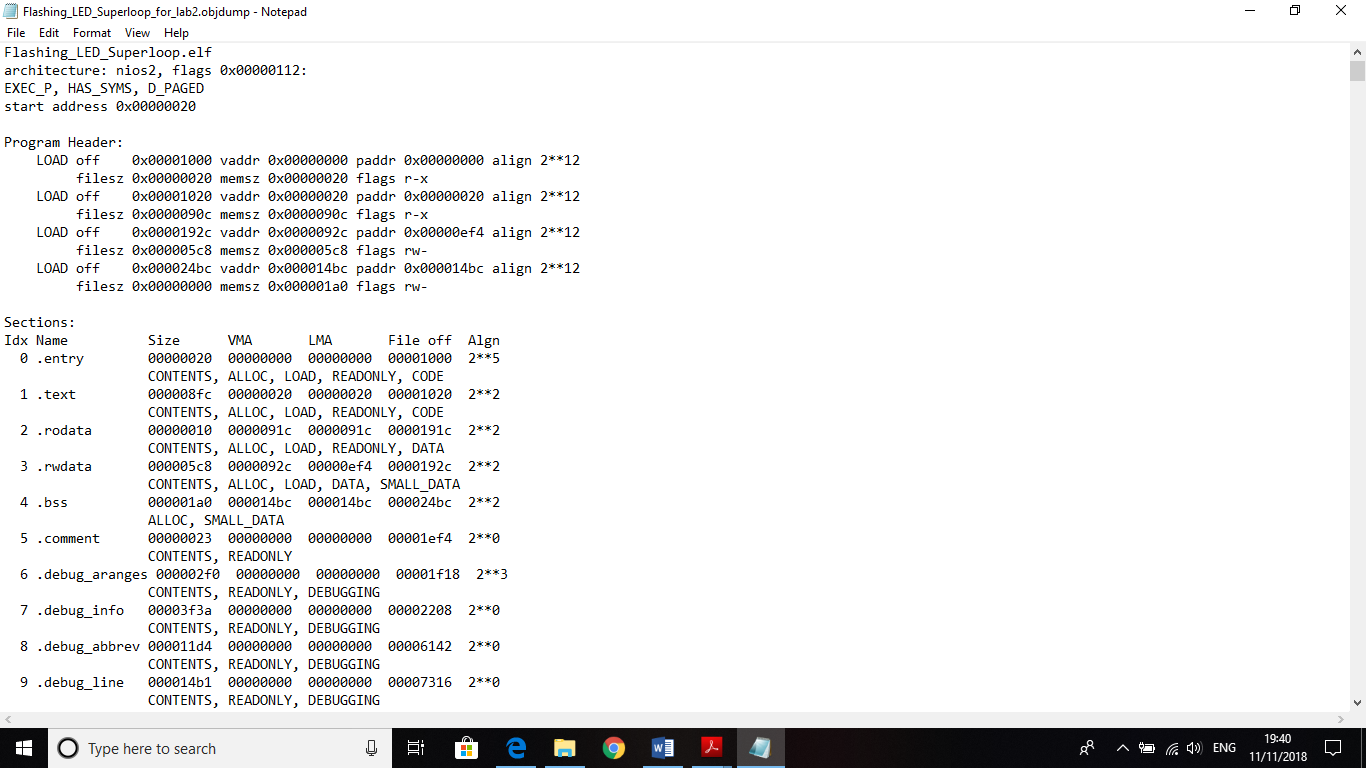
EG3205 Lab 2

Compilation Report which shows the hardware consumption from Lab 1



Dump file which shows the software consumption from Lab 1

The size of code and data are shown under the heading “Sections”, it is given in hexadecimal.



The main function initialised the scheduler, and the LED flashing task. This flashing LED task doesn’t do anything (is purely for convention) this is because the LED is actually set up in the PORT.H header file using bit shifting. Next, the LED\_Flash\_Update() task was added to the scheduler, which has no initial delay i.e. it runs as soon as the program is executed, and this task updates every 1 second. The LED\_Flash\_Update() function used a bitwise XOR to turn the LED (on a specified port) on/off. So, this program makes an LED blink, the LED remains on for one second and off for one second continuously. After all the tasks are added, the SCH\_Start() function was called to begin the scheduling process, this was achieved by globally enabling interrupts.

SCH\_Update() controlled the timing of the LED\_Flash\_Update() function. The SCH\_Update() function (this function is an interrupt service routine) which was triggered by the overflow of timer 0 (the required overflow was 0.050 seconds (50 ms) in the example done in the lab). When the scheduler determined that a task is due to run, the update function incremented the RunMe field for this task.

The SCH\_Update() function calculated when a task was due to run and set a flag, it did not run the task. The dispatcher function SCH\_Dispatch\_Tasks() executed LED\_Flash\_Update(). Thus, the dispatcher function executes the same infinite loop as was run in the first superloop program in lab 1.

A timer was set up that was used to generate the regular ‘ticks’ to drive the scheduler, the timer generated an interrupt when it overflowed. Timer 0 overflows to invoke the ISR. As Timer 0 does not have a 16-bit ‘auto reload’ mode like Timer 2; as a consequence, the timer must be manually reloaded every time it overflows. This automatic reload facility that Timer 2 has ensures that the timer keeps generating the required ticks, at precisely 1 ms intervals, without any software load, and without any intervention from the user’s program.

Various extracts have been used from “Patterns for time-triggered embedded systems” by Michael J Pont to write this report.