Exercises 2IN60.2

Today we will start with programming the MC9S12XF512 micro-controller. In particular, we will experiment with different variants of the cyclic executive.

2.1 As Fast As Possible (AFAP)

In this exercise you will implement two tasks and execute them using the AFAP approach. Each task senses a sensor connected via the ATD converter and actuates a led in case the sensed value crosses a threshold. The CodeWarrior project for this exercise is in the directory `exercise2_1`.

1. **The `main.c` file contains a specification of the two tasks: `Task1()` and `Task2()`. Implement these tasks, making use of the `ATDReadChannel()` function introduced during the lecture.**

   Hint: you may like to look at the `Led_driver.h` and `Led_driver.c` files inside the `Drivers` directory for functions to toggle and set the leds.

2. **Implement the `main()` function, making sure that `Task1()` is executing before `Task2()` according to the AFAP cyclic executive approach.**

Deliver the modified `main.c` file as answers to steps 1 and 2.

Let $T_i^{\min}$ and $T_i^{\max}$ be the minimum and maximum inter-arrival time between two consequtive jobs of task τ_i, respectively. Activation jitter for task τ_i is defined in terms of the maximum and minimum inter-arrival time between its two consecutive jobs: $J_i = T_i^{\max} - T_i^{\min}$.

3. **Measure the minimum and maximum execution times (in cycles) of `Task1` and `Task2`.**

   Note that the simulated ATD converter always reads a value 0 from all the ports.

4. **Give a formula for the inter-arrival time between two consecutive jobs for `Task1` and `Task2`.**

5. **Derive the formula for the activation jitter for `Task1` and `Task2`.**

2.2 Time-driven AFAP

In this exercise you will make the control loop periodic. The CodeWarrior project for this exercise is in the directory `exercise2_2`.

1. Copy your task definitions and control loop from Exercise 2.1 into the `main.c` file in this project.

2. The Freescale HCS12 instruction set provides instructions `STOP` and `WAI` which can be used to suspend the processor. These instructions are described in Section 5.27 of the HCS12 manual (`http://www.win.tue.nl/~mholende/automotive/S12CPUV2.pdf`). **Use one of these instructions to implement Time-driven AFAP, i.e. to activate the task sequence periodically.** Check the implementation of `ATDReadChannel()` in the`ATD_driver.c` file for the syntax for writing assembly instructions in C code.

3. The Freescale MC9S12XF512 micro-controller can generate a Real-Time Interrupt at a fixed frequency, which is derived from the main CPU clock by means of a divider. The frequency of the timer is set in the `CPUInitRTI()` function in the `cpu.c` file. It is currently set to 1KHz. **Consult Section 2.3.2.8 in the MC9S12XF512 manual (`http://www.win.tue.nl/~mholende/automotive/MC9S12XF512RMV1.pdf`) and change the frequency to 2Hz.**

2.3 Activation jitter and drift

In this exercise you will investigate drift. The CodeWarrior project for this exercise is in the directory `exercise2_3`.

1. **The `main.c` file contains a specification of the two tasks: `Task1()` and `Task2()`. Implement these tasks.**

2. The `main()` function contains a time-driven AFAP control loop, which iterates over the task sequence 1000 times. Place brake points at the `asm nop;` instructions around the control loop and **measure the activation jitter of the 1001st job of `Task1()`.**

3. Modify the `main()` function to implement a simple AFAP control loop, which iterates over the task sequence 1000 times. Place brake points at the `asm nop;` instructions around the control loop and **measure the activation jitter of the 1001st job of `Task1()`.**

4. **Does the AFAP or the time-driven AFAP control loop suffer from drift? Motivate your answer.**