***Report lab 2***

Embedded system

Ahmed Badday & Johannes Ferm

<https://github.com/AhmedBadday/Grupp1>

Task A1:

Found in the file taskA1.ino. Using a regular delay to not suspend the task and to keep it busy waiting.

Task A2:

Found in the file taskA2.ino. Using a vTaskDelay to suspend the current task allowing the other task to run. The type of scheduling used depends on the config. If time slicing is enabled then time slicing is used, meaning each task gets a slice of time and they switch back and forth. This is round-robin scheduling. We’ve turned off time slicing in the config. Therefore we have highest priority first scheduling (which is why we need the fake vTaskDelay since the tasks have the same priority).

Task A3:

When changing the priority we get pre-emption. So even though we have a vTaskDelay only one task runs (visibly). This is because the vTaskDelay is short so whenever the lower priority task gets control it only keeps control for 1 ms before it gets pre-empted by the higher priority task.

Task A4:

A vTaskDelay allows the other task to run while the current task is delayed. So after setting the pin to HIGH (in taskA2) the other task is able to run for 3 seconds also setting its pin to HIGH and then go into its own delay allowing the other/first task to run again. But this task is still in its delay, so we have a period of both LEDs being on before the first one turns off. Then when the higher priority task is past its delay it turns off the LED and the hands over the control to the lower priority task. But since the higher priority task no longer is delayed it’ll immediately pre-empt the lower priority task. Then the higher priority task starts over from the beginning, turning on its LED and going into the delay etc.

Task B1:

Found in lab2\_del\_B.ino. Task3 is used as a “driver” (setting new\_command to true) setting things up so that task 1 and 2 change the LEDs. Also using a mutex in the critical sections to ensure mutual exclusion on the shared resources (variables in our case).

Task B2:

Found in lab2\_del\_B2.ino. Uses a queue instead of a bool (as in pervious task). Task3 is still the “driver” but enqueues commands (the brightness part of the command) to the right queue instead of setting new\_command = true. In the LED-controlling tasks we check whether the queue is empty or not (instead of checking new\_command) and dequeue from the queue if it isn’t empty. Still using mutex to ensure mutual exclusivity.

Task B3:

Found in lab2\_delB3.ino. Uses an interrupt. Whenever we press the button (pin 7) an interrupt service routine is invoked/run. This isr is connected to the function reset\_queues\_isr which resets the queues.