Lab n° 2
Real-Time systems [ELEC-H-410]
Implementation of a multicyclic scheduler

2021–2022

Introduction

During this lab, you will implement a multicyclic scheduler for a time-driven system. In such a system, all the tasks are known at design-time (i.e. when you write your microcontroller code), and all scheduling can be done beforehand.
Such systems are preferred for critical real-time systems, as they can be entirely deterministic and entirely safe. The drawback is that they are not very flexible: adding new functionalities to the system is limited, as we will see in this lab.

Useful documentation:

- Getting started with PSoC 5LP: `https://www.cypress.com/file/41436/download`
- Video example on how to use the PSoC: `https://www.youtube.com/playlist?list=PLIOkqhZiy83F8KPvHejA4ujvAfwJYpAtP`
- The extension board schematics: `Extension_PSoC.pdf`
- Getting started with the Logic Analyzer: `https://learn.sparkfun.com/tutorials/using-the-usb-logic-analyzer-with-sigrok-pulseview`

Lab kit contents

Today, you will use the following items of your lab kit:

- the PSoC microcontroller mounted on an extension board,
- the logic analyzer.

1 Scheduling of periodic tasks

1.1 Implementation of a minimalistic scheduler

We will start by implementing a minimalistic scheduler, which we will use to implement a muti-cyclic scheduling system. The scheduler should tick at a fixed rate (i.e. the `TICK` period of the system). To ensure a stable period, we will use a `Timer` block that is connected to an `Interrupt ReQuest (IRQ)` pin. The tick of the timer corresponds to the period of one micro-cycle, as shown in Figure 1. At the beginning of each micro-cycle, the scheduler should determine which tasks need to be executed in the current micro-cycle, and launch them.

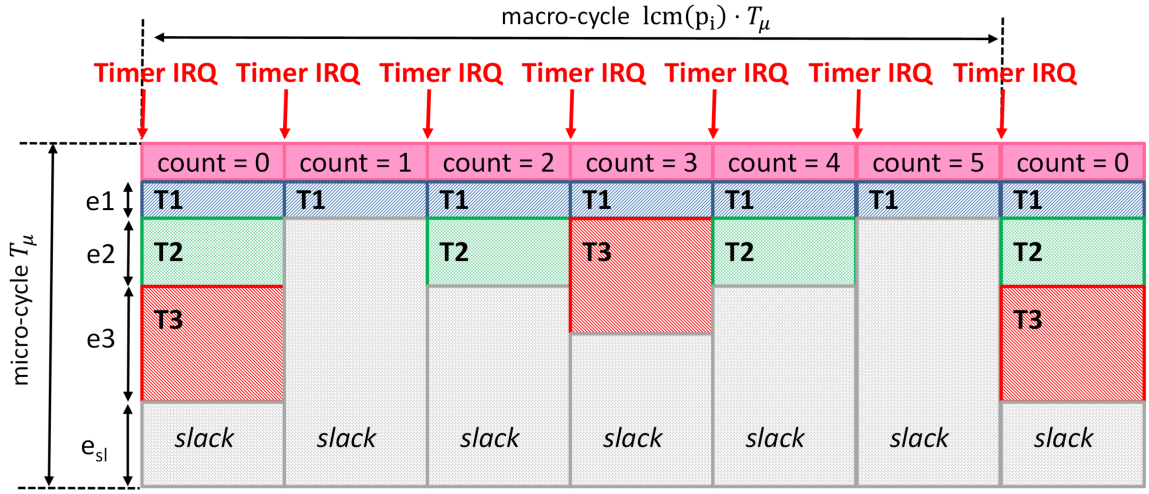

Figure 1: A muti-cyclic scheduling system.

Exercise 1. Open the `Lab2` project (since our versions of PSoC Creater might not match, go to `Project` and select `Update Components`. Follow the instructions with the default settings). The timer and the IRQ have already been instantiated in the `TopDesign.cysch` file in order to tick at a period of 10 ms.

- Write the structure of the Timer Interrupt Service Routine (ISR). You can check the following tutorial to see how to write the ISR: `https://www.youtube.com/watch?v=pHd3DcWWY3c&list=PLIOkqhZiy83F8KPvHejA4ujvAfwJYpAtP&index=3`.
- The scheduler is executed in the ISR. The scheduler should determine which tasks have to be executed in the current micro-cycle. **Watch out:** the tasks should not be executed in the ISR itself! The scheduler should only raise boolean flags associated to tasks that need to be executed in the current micro-cycle. The tasks themselves should be executed in the main `for(;;)` loop.
- The Timer interrupt flag should be lowered manually at the end of the ISR by calling the `Timer_ReadStatusRegister()` function (more information about this can be found in the Timer datasheet).
- Use the logic analyzer to measure the execution time of the scheduler. What is the CPU usage of the scheduler?

1.2 Scheduling two periodic tasks

We will now write the actual tasks that will be executed.

The first task will perform an Analog-to-Digital conversion at the period of the Timer. The ADC pin is configured to measure the voltage drop over the potentiometer on the extension board. This voltage drop should be between 0 V and 5 V. You can use the following functions to control the ADC:

- `ADC_StartConvert()`: to initialize the ADC and launch it;
- `ADC_IsEndConversion(ADC_RETURN_STATUS)`: returns `True` if the conversion is finished, `False` otherwise;
- `ADC_GetResult32()`: returns the actual digitized value.

Exercise 2. The ADC task should be executed every 10 ms.

- adapt the code of the scheduler so that the ADC conversion is done every 10 ms;
- write the code of the ADC task to perform the actual analog-to-digital conversion;
- the digitized value should be stored in a global variable (which we will use later on). To quickly check this variable, you can print it's value on the LCD.

The second task will print a message on the LCD. Since the message is longer than the 16 characters on the LCD, the message will need to translate to the left every 200 ms, as shown in Figure 2.

Figure 2: Message translating on the LCD screen.

Exercise 3. The LCD task should be executed every 200 ms.

- Adapt the code of the scheduler accordingly.
- Write the code of the LCD task such that the message shifts left on the LCD by one character every time the LCD task is called.

1.3 Scheduling a variable-period task

In this section, we will write a task that has a variable period. The task will light the LEDs 1 to 4 successively, as shown in Figure 3. The speed at which the LEDs light up should be controlled by the ADC potentiometer.

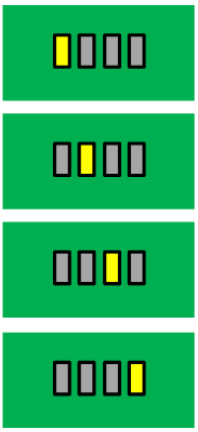

Figure 3: LEDs 1 to 4 lighting up successively.

Exercise 4. The LED task should be executed with a variable period.

- Adapt the code of the scheduler to have a period between 10 ms and 1 s, depending on the position of the potentiometer. The value read by the ADC should be converted accordingly;
- write the code of the LED task such that the LEDs 1 to 4 light up successively;
- use the logic analyzer to measure the execution time of the ADC task, the LCD task and the LED task. How much slack time is there in our system? What would be the shortest possible period for the scheduler tick? What happens if you use a tick period smaller than this minimum period?

1.4 Using the slack to put the microcontroller to sleep mode

The slack time can be used to put the microcontroller to sleep in order to save energy, or the schedule aperiodic tasks. The PSoC has several low-power mode: the `AltAct` mode, the `Sleep` mode and the `Hibernate` mode. These modes differ in the way the microcontroller can be woken up (see more information in the application note Cypress AN77900).
To put the PSoC to `AltAct` mode, use the `CyPmAltAct()` function. The parameters of the function are the following:

- `wakeupTime`: specifies how long the PSoC goes to `AltACt` mode. Use `PM_SLEEP_TIME_NONE` if you don't want the PSoC to wake up after a certain time;
- `wakeupSource`: specifies the source that can wake up the PSoC. Use `PM_ALT_ACT_SRC_INTERRUPT` to have the PSoC wake up when an interrupt occurs.

Exercise 5. Use the previous function calls to put the PSoC to `AltAct` mode during the slack time. Use the logic analyzer to verify that all the tasks are still executed at the right period.

2 Scheduling of aperiodic tasks

In this section, we will use the slack time to schedule aperiodic tasks, i.e. tasks whose activation time is not known at design time. We will design a task that will count the number of times `SW1` is pressed and print this on the right-hand side of the LCD.
The GPIO corresponding to `SW1` has been connected to an interrupt line in the `TopDesign.cysch` file. The interrupt should be masked (i.e. disabled) to avoid interfering with the timing of the system. To check whether the interrupt has been triggered, the interrupt flag should be polled by using the `SW1_ClearInterrupt()` function. This function call will return `True` if the interrupt flag was raised, `False`otherwise. The function call will also clear the interrupt flag.

Exercise 6. Modify the scheduler in order to poll the interrupt flag to check whether `SW1` was pressed (don't forget to disable the interrupt!). When this happens, the SW1 task should be scheduled. Then write the SW1 task such that the right-half of the LCD prints the number of times SW1 was pressed.

Exercise 7. Use the logic analyzer to ensure the aperiodic task does not affect the timing of the periodic tasks.