Lab n° 1
Real-Time systems [ELEC-H-410]
Realization of an application under µC/OS-II

2017–2018

Purpose

The course ELEC-H-410 "Real-Time Systems" has 9 practical work sessions divided into:

- 3 labs about a real-time OS: µC/OS-II
- including 1 lab focussing on the CAN network
- 6 labs to realize a project: "Building a distributed alarm", using the concepts previously acquired.

Useful documents are stored on the network share:

```
\\labo\ELEC-H-410\Useful Documents\

  - dsPIC30F-33F Programmer's Reference.pdf
  - dsPIC33 Data Sheet.pdf
  - Introduction to MPLAB.pdf
  - Explorer 16 User Guide 51589a.pdf
  - MPLAB C30 C Compiler User's guide.pdf
  - uCOSII_RefMan.pdf
  - Enhanced Controller Area Network.pdf
  - Introduction to language C for microcontrollers.pdf
  - Troubleshooting (common errors fixing)
```

1 First lab

During this first lab, you will learn how to write a task under µC/OS-II, to make it periodic and to assign it a priority, in an intelligent way. The hardware will be composed of a microcontroller board and a logic analyser.

If you are not confident with C programming, read `Introduction to C for microcontrollers`.

Principles of the logic analyser are explained in the chapter 9; an "how to" guide for the Asix Sigma2 logic analyser is in Appendix A.

1.1 Creation of a task under µC/OS-II

A task is a succession of instructions doing a specific operation. Contrary to a function, a task cannot return a value[1]. Moreover you do not have any direct influence on the order of execution of the various tasks you create. Indeed, it is the operating system which is given the responsibility to schedule the tasks and thus to choose which task must be carried out at which time on the processor. µC/OS-II is a preemptive RTOS based on fixed priorities that you assign to the tasks. The choice of those priorities is thus critical so that the system behaves as you wish. This is why the second part of this lab will be related to the wise choice of priorities.

First, you will learn how to create a single simple task in µC/OS-II and to initiate the execution of the operating system.

Copy the project `\ELEC-H-410\uCOS-II\Exercices\Example1` in the network share to your computer and open the project with MPLAB.

---

[1] Actually, if a task returns, the OS will crash and reboot.

In the file main.c you will find the `main` function (see Listing 1) in which are executed :

- the initialization of µC/OS-II and all its internal variables : `OSInit()`
- the creation of the task `AppTaskStart`: `OSTaskCreateExt()`
- the starting of µC/OS-II: `OSStart()`

This structure cannot vary. The operating system must indeed be initialized before any creation of task and at least one task must have been created before giving control to OS. If no task were present in the system when the `OSStart()` function is called, µC/OS-II would launch a useless task "Idle" and do nothing else would be executed by the CPU.

For more details on the parameters sent during the creation of the task, refer to the µC/OS-II user's manual (page 113).

Listing 1: Function main.c

```c
#include <includes.h>
// [...] declarations...
CPU_INT16S main (void)
{
CPU_INT08U err;
OSInit();
 // Initialize "uC/OS-II"
OSTaskCreateExt(
AppTaskStart, // creates AppStartTask
        (void *)0,
        (OS_STK *)&AppTaskStartStk[0],
        APP_TASK_START_PRIO,
        APP_TASK_START_PRIO,
        (OS_STK *)&AppTaskStartStk[APP_TASK_START_STK_SIZE-1],
        APP_TASK_START_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
OSStart();
 // Start multitasking (i.e. give control to uC/OS-II)
}
return (-1);// Return an error - This line of code is unreachable
```

1.2 How to write a task

Adding tasks to the OS is very easy:

- The task must be written like a function which returns nothing (`void`).
- The task must contain an infinite loop of any kind: use one of the 2 structures `for(;;){code...}`[2] or `while(1){code...}`.
- A task must always call at least one of the services of µC/OS-II that will make the task "waiting" like:
  `OSTimeDly()`, `OSTaskSuspend()`, `OSSemPend()`, `OSMailboxPend()` or `OSMutexPend()`.
- The task must be added to the OS before the call to `OSStart()`, see Listing 1 and µC/OS-II documentation for precise syntax.

Since µC/OS-II is preemptive, the currently running task has got the highest priority among all "ready" tasks, hence if no event occurs (like an ISR making a higher priority task ready or the current task giving the control back to the scheduler) no other tasks will ever run.

Listing 2: task1.c Basic task example

```c
void task1 (void *data){
... //init code, variable declaration
while(1){
        ... //task code
        OSTimeDly(10);  //ask the RTOS to put task1 in "waiting"
                        //state for at least 9 ticks
}//while
}
```

---

[2]Please, avoid this one, really.

1.3   Put a task to sleep for some time

Sometimes, it is necessary to let a task sleep for a while (maybe the job is complete, the task needs some resource/data not available yet to complete its job. . .)

To let a task sleep in "waiting" state for some time, one might call the `OSTimeDly(INT16U tick_nbr)` function; The parameter `tick_nbr` is an unsigned 16bit integer[3] which determines the number of ticks during which the task will sleep. The timer creating the periodic interrupts has been configured for a frequency of 1kHz, hence 1 tick = 1 ms. More precisely the task will sleep at least (`tick_nmbr-1`), if you want to be sure to sleep during 1 tick you should specifiy `tick_nmbr=2`. To demonstrate that, draw a chronogram of tick interrupts and imagine where the call `OSTimeDly()` could occur.

Exercise 1. Create one second task in the Example1 project which lights a LED of the uC board at a frequency of 1Hz.

To change the state of the LED, toggle pin `LATAbits.LATA3`. Remember that you have to configure the pin in the output direction by using the instruction `TRISAbits.TRISA3 = 0`.
(see `dsPIC33 Data Sheet` p157 or `Explorer 16 User Guide 51589a.pdf` p33/43)

Use the code for `AppTask1` as a model, don't forget stack and priority declarations.

1.4   Creation of periodic tasks

In Exercise 1, you have created a periodic task, *i.e.* a task executing forever at regular intervals. In most industrial applications, those tasks are frequent and the periodicity should be realized with a good precision (see example of PI controller in chapter 3 of the course).

Open the project `Example_Periodicity`.

You will find 4 tasks in this example:

- `AppTaskStart` whose only goal is to create the three other tasks
- `AppTask1` which should have a period of 10ms;
- `AppTask2` which should have a period of 50ms;
- `AppTask3` which should have a period of 100ms.

Exercise 2. Scheduling verification. We will use the logic analyser to verify if your task is scheduled correctly in the RTOS.

- Switch the logic analyser on and launch the display interface on the PC.
- Open the test file `elec-h-410.stf` in the ELEC-H-410 folder. You can customize it if you want.
- Start your program `Example_Periodicity` on the microcontroller and launch a first data acquisition with the logic analyser.
- Observe the evolution of the value of the bus `RunningTaskId` which shows the identifier of the tasks running on the processor. Observe preemptions of certain tasks when a higher priority task is active (see signals `Task1Active`, `Task2Active` et `Task3Active` whose value is 1 when the tasks `AppTask1`, `AppTask2` and `AppTask3` are respectively active, *i.e.*, between its first instruction until its completion).
- Use the logic analyser to measure the real period of real activation of each task. Hint: press *space* to set a marker and move the cursor, the time between the cursor and the marker will be shown in a tooltip. Are they exactly in conformity with the desired periods? Identify 2 causes of these errors.

1.4.1   Use of `OSTimeGet()`

µC/OS-II provides the `OSTimeGet()` function which returns a 32 bit integer (`INT32U`) representing the number of ticks since the launching of OS.

Exercise 3. Compute after how long this counter will overflow.

Exercise 4. Use `OSTimeGet()` in each task to compensate for the error over the period.

---

[3]ranging from 0 and 65535

1.4.2   Use of a software timer

It is possible to use software timers in µC/OS-II. Those are used exactly in the same way as hardware timers, except that they are entirely managed by the operating system and that they are synchronized on the ticks of the system. The function `OSTimerCreate()` allows to create a software timer (see µC/OS-II manual for details) and `OSTmrStart()` to start it. When a timer expires, it calls a function whose pointer was given in the parameters.

Open the project `Example_Timer`. You will find the same 4 tasks as in the previous example except that their period are generated by using three software timers.

Functions `OSTaskSuspend()` and `OSTaskResume()` allow to suspend and restart the execution of a specific task.

Exercise 5.   Check with the logic analyser that the periods are strictly respected.

This method for creating periodic task gives very precise results. However, it is rather heavy and should therefore be used when this precision is absolutely required.

Exercise 6.   Create a new timer which switches a LED on after 5s. There is no need to write a complete task for this exercise.

1.4.3   Choice of the priorities

As explained earlier, the choice of the priorities of the task is the only tool at our disposal to help the operating system to choose which task must be running at which time. To be convinced of the importance of a judicious choice of these priorities, we will look at a simple example.

Exercise 7.   Open the project `Example_Priorites`.

- The task `AppTask1` should run every 1ms
- The task `AppTask2` should run every 100ms

- Check the behaviour of the tasks with the logic analyser.
- Reverse the priorities of `AppTask1` and `AppTask2` and reverify what occurs.
- By comparing the periods of each task and the priorities assigned, which systematic rule of assignment can you deduce?
- How is called this method to assign the priorities?
- What happens when tasks have relative deadlines different from their periods?
- Which scheduling algorithm would you use if you could assign priorities directly to jobs instead of tasks?
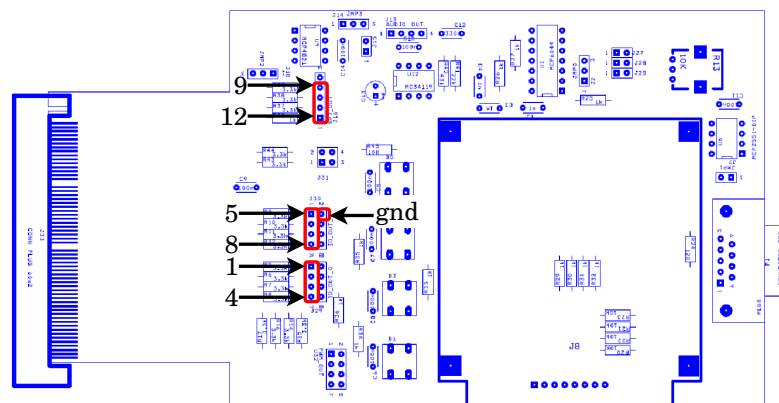
A The *Asix Sigma2* logic analyser

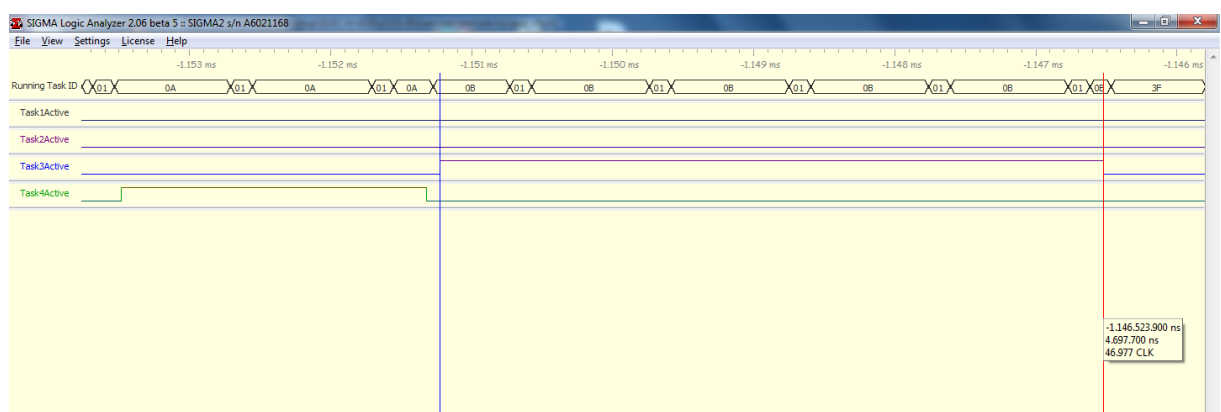The Asix Sigma2 logic analyser is like:

A.1 Electrical connections to the Explorer 16 board

Connect the analyser to the extension board with the numbered ribbon cable following this scheme:

A.2 Software on the computer

The software interface of the logic analyser looks like:

A.3 Basic measurements

The red line on the screen is a cursor showing the time and values of signals in the main window. To place a marker (blue line), press space. If you move your cursor, the difference between the marker and the cursor will show in a tooltip.

The first acquisition must be launched by software. Following acquisitions can be done using the "go" button of the analyser.