EMARO-ARIA RETSY labs

Jean-Luc Béchennec, Sébastien Faucou

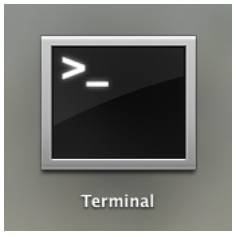
September 29, 2018

Chapter 1

Installation

Before using Trampoline, we have to install some tools. Instead of providing you a ready to use environment, we believe it is better to show you how to install one by yourself so that you will be able to install one on your own computer if you want. Users are not allowed to do system-wide installations of software on Mac OS X computers of ECN so all the installation will be done in your home directory.

During this lab, we are going to use the terminal and the command line interface of the Unix operating system. The program in charge of parsing and executing the command line is known as the shell. Maybe is it new for you. We will use Mac computers with the Mac OS X operating system. Mac OS X is a Unix system. At the bottom of the screen, you have the dock. Locate the launcher icon and click on it. The screen should show a grid of application icons similar to what you have on an iPhone or on an Android system. Click on the `Autre` item and then on the `Terminal` item.



Once the Terminal is launched you should get a prompt with the name of the computer, the current directory (which is your home directory) and the username. For instance on my computer I get:

```
Last login: Mon Nov  9 15:48:29 on ttys059
mo-bechennec:~ jlb$
```

`mo-bechennec` is the name of the computer, ~ is the current directory[1] and `jlb` is my username.

> Do not copy the commands from the PDF file, the characters you get may not have the correct code and the shell will not understand them.

When typing shell commands, spaces are important because they separate the command and its arguments. In this document, spaces in commands are represented by a white rectangle like in the following command (it is an example, do not type it):

`cd trampoline` sets the `trampoline` directory as current directory. This assumes the `trampoline` directory is a subdirectory of the current one.

1.1 Development environment

Create a directory in your home to put the development tools. Let's name it `Tools`. To do that type: `mkdir ~/Tools`

We are going to install the GNU development tools for ARM. Go to https://launchpad.net/gcc-arm-embedded. On the right there is a list of distributions (green buttons). The distribution for Mac OS X is the 5th one, click on it. Here is a direct link.

The archive should expand to the `gcc-arm-none-eabi-4_9-2015q3` directory in the `Download` directory. Rename this directory to `gnu-arm` and put it in your home directory. If your current directory is not the home directory, typing `cd` without any argument will bring you back to the home directory. Then, move and rename the directory thanks to the `mv` (move) command:

```
mv ~/Downloads/gcc-arm-none-eabi-4_9-2015q3 ~/Tools/gcc-arm
```

The ~ is obtained by the `alt-n` keys then the `space` key. When typing commands, you can use the →| key. It completes the string you are typing according to the files available in the directory. For instance, instead of typing all the letters of `Downloads` and `gcc-arm-none-eabi-4_9-2015q3`, you can type: `Dow` →| that completes to `Downloads/` then `gcc` →| that will finish the completion to the end if there is no other file starting with `gcc` in the `Downloads` directory.

1.2 Trampoline

1. Go in your home directory if you are not already in it: `cd`

[1] ~ means home directory.

2. Get Trampoline from the GitHub repository:

```
git clone https://github.com/TrampolineRTOS/trampoline.git
```

This command will create a `trampoline` directory.

1.3 Goil

Goil is the OIL compiler.

1. Go in the `makefile-macosx` directory: `cd trampoline/goil/makefile-macosx`
2. Type `./build.py` to build goil.

1.4 Teensy loader

Teensy loader is a command line tool to upload software to the Teensy board. Get a precompiled version from https://github.com/TrampolineRTOS/trampoline, scroll down to the end of the page and click on teensy-loader-cli. This zip should uncompress automatically leaving an executable named `teensy-loader-cli` in the `Downloads` directory.

In the Terminal, go back to your home: `cd`. Create a `bin` directory inside `Tools`: `mkdir Tools/bin`. Move `teensy-loader-cli` to the `bin` directory: `mv Downloads/teensy-loader-cli Tools/bin`.

1.5 Environment settings

Now that you have downloaded the tools, you have to define the configuration of your development environment. First, you have to inform the shell of the location of the tools. This is done by setting the `PATH` environment variable.

Edit your .profile in your home or create it if it does not exist and add the following lines at the end:

```
export PATH=$PATH\
:~/trampoline/goil/makefile-macosx\
:~/Tools/gcc-arm/gcc-arm-none-eabi/bin\
:~/Tools/bin
```

Warning: do not copy and paste from the pdf file to the .profile text file: pdf may use characters that are not compatible with the text file format.

Second, you have to inform Goil where to find a specific directory. This directory contains files called templates, that are used by Goil for generating the configuration of Trampoline for a given application.

Edit your .profile in your home and add the following lines at the end:

```
export GOIL_TEMPLATES=~/trampoline/goil/templates
```

The .profile file is parsed by the shell when it starts. To force your current shell instance to read the file, use the following command:

```
source ~/.profile
```

Now, test that Goil is working. The command `goil --version` should print:

```
goil : version 2.1.21
```

Test that gcc is also working. The commande `arm-none-eabi-gcc --version` should print:

```
arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.9.3 20150529 (release) [ARM/embedded-4_9-branch revision 227977]
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Last, test Teensy loader is working. The command `teensy-loader-cli` should print:

```
Filename must be specified

Usage: teensy_loader_cli -mmcu=<MCU> [-w] [-h] [-n] [-v] <file.hex>
-w : Wait for device to appear
-r : Use hard reboot if device not online
-n : No reboot after programming
-v : Verbose output

<MCU> = atmega32u4 | at90usb162 | at90usb646 | at90usb1286 | mk20dx128 | mk20dx256

For more information, please visit:
http://www.pjrc.com/teensy/loader_cli.html
```

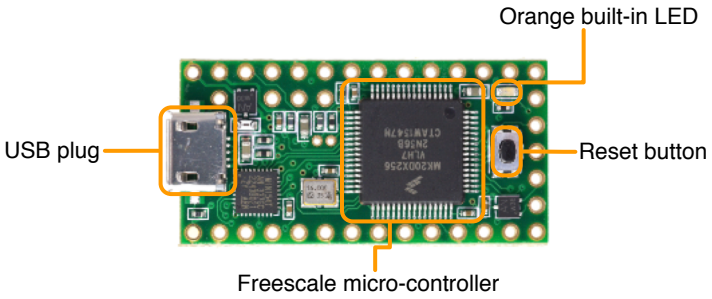
You are now ready to compile and load your first application on the board. But first, let's take a look at the board.

Chapter 2

The board

2.1 The Teensy 3.1

The board is built around a Teensy 3.1 breakout board (BB). A breakout board is a minimal board designed to be used with tiny SMD[1] on a breadboard or in a hobbyist design. The Teensy 3.1 BB is built around a Freescale Micro-controller, the MK20DX256VLH7, which has an ARM Cortex-M4 computing core. It is a 32 bits micro-controller running at 96MHz. It embeds 256kB of flash memory to store the program and the constant data and 64kB of SRAM to store the variables. The Teensy is built by PJRC, a small company from the USA. The web site is here. Here is the Teensy 3.1:



2.2 The labs board

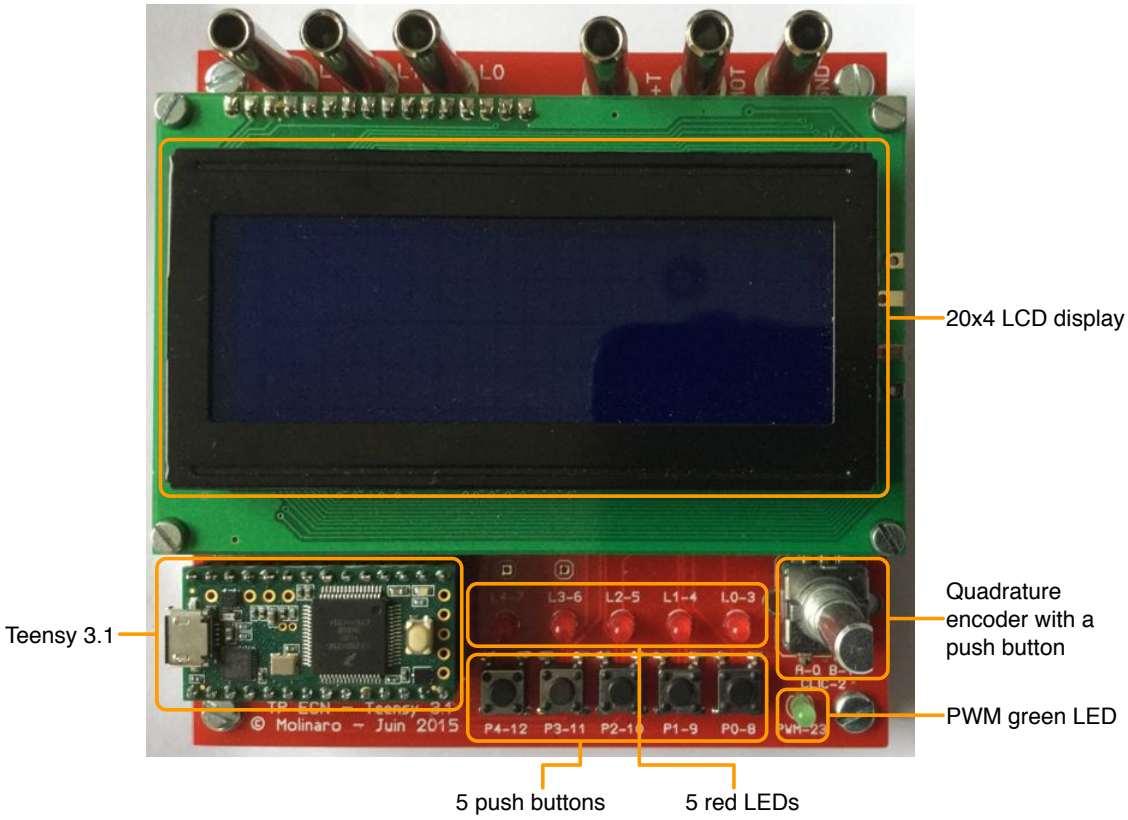
The labs board put together the following systems:

- A 20 colums, 4 lines LCD screen;
- 5 push buttons;

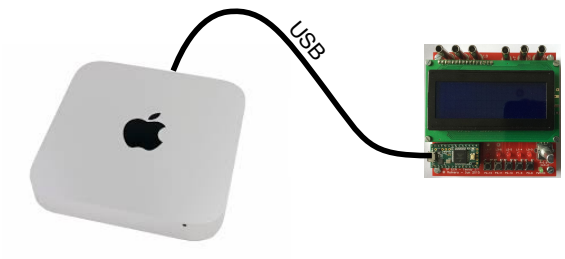
[1] Surface Mounted Device

- 5 red LEDs;
- A Quadrature encoder with a push button;
- A PWM[2] output / analog input;
- A green LED to display the PWM.

Here is a picture of the board:



The board power is supplied by the USB. Connect a Mac Mini USB port to the Teensy USB plug (see 2.1) and the board switches on.



[2]Pulse Width Modulation

2.3 LEDs, buttons and LCD

The Trampoline port for this board includes functions to turn LED on and off, to read the buttons and to write to the LCD. These functions work as the functions available on an Arduino:

void pinMode(*pin***,** *mode***)** sets the mode of a pin. *pin* is the number of the pin and *mode* can be `INPUT`, the pin is an input, `INPUT_PULLUP`, the pin is an input and a resistor pulls the pin up to 3.3V (logical 1), `OUTPUT`, the pin is an output.

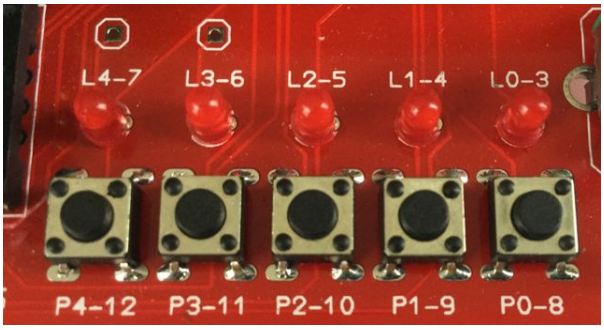
uint8 digitalRead(*pin***)** returns the logical level of a pin: HIGH (logical 1) or LOW (logical 0). *pin* is the number of the pin. If the pin is programmed as an input, the value is what is set on the external pin. If the pin is programmed as an output, the value is what was written to the pin.

void digitalWrite(*pin***,** *state***)** set the logical level of a pin *or* enable or disable the pullup. *pin* is the number of the pin, `state` is the logical level: `HIGH` (logical 1) or `LOW` (logical 0). If the pin is programmed as an output, the state is set on the external pin. If the pin is programmed as an input, a `HIGH` state enables the pullup, a `LOW` state disables the pullup.

The LCD is accessed by using the LiquidCrystalFast library which is compatible with the LiquidCrystal library. You can get the documentation here.

2.3.1 LEDs

If you look just above the LEDs, you see labels `Lx-y`. `L` means **L**ED, `x` is the identifier of the LED and `y` is the number of the Teensy pin used to control the state of the LED.



To use a LED, you need to:

1. Program the corresponding pin, `y`, as an output;
2. Set the logical level to `HIGH` to turn the LED on;
3. Set the logical level to `LOW` to turn the LED off;

Typically, pin direction programming is done in the `main` function before starting Trampoline. For instance the following source code turns on LED 0 (pinMode and digitalWrite on pin 3) before to start Trampoline by calling the StartOS function.

```
int main()
{
    pinMode(3, OUTPUT);     /* LED L0-3 pin is set as an output */
    digitalWrite(3, HIGH); /* Turn L0-3 on                      */

    /* Start Trampoline in the default appmode */
    StartOS(OSDEFAULTAPPMODE);
    return 0;
}
```

2.3.2 Pushbuttons

As for LEDs, pushbuttons are labelled with `Px_y` where `P` stands for **P**ushbutton, `x` is its number and `y` is the number of the Teensy pin used to read the state of the pushbutton.

When pushed, a pushbutton connects the corresponding pin to the ground producing a logic 0 state (LOW). When not pushed, the pin is floating. i.e. it is not connected to anything.

Tu use a pushbutton, you need to:

1. Program the corresponding pin, `y`, as an input with pullup enabled;
2. Read the pin, `y`, to be informed of the state of the pushbutton:
   - if the returned value is `HIGH` the button is not pushed;
   - if the returned value is `LOW` the button is pushed;

The following example reads button `P0_8` to start Trampoline in a specific appmode.

```
int main()
{
    pinMode(8, INPUT_PULLUP);  /* Button P0_8 pin is set as
                                  an input with pullup enabled */
    if (digitalRead(8) == LOW) /* The button is pressed        */
    {
      /* Start Trampoline in the testAppmode appmode */
      StartOS(testAppmode);
    }
    else                       /* otherwise */
    {
      /* Start Trampoline in the default appmode */
      StartOS(OSDEFAULTAPPMODE);
    }
    return 0;
}
```

2.3.3 LCD

As already stated, the LCD is driven by the LiquidCrystalFast library. This library is written in C++. An object of type LiquidCrystalFast is instantiated with the connection pins as arguments. The LCD is connected to pins 18,17,16,15,14 and 19. So a LiquidCrystalFast object must be instantiated as a global variable as follow:

```
/*
 * A LiquidCrystalFast object is instantiated.
 * Pin numbers are per Teensyduino specification
 */
LiquidCrystalFast lcd(18,17,16,15,14,19);
```

In the main function, the LCD is initialized by using the following instruction:

```
  lcd.begin(20, 4);
```

20 is the number of columns and 4 the number of lines. Once the LCD initialized, the following methods can be used:

lcd.setCursor(x**,** y**)** to put the cursor at column x and line y;

lcd.print(*something***)** to write *something* on the LCD;

lcd.println(*something***)** to write *something* on the LCD and go at the beginning of the next line.

Chapter 3

Lab 1
Understanding fixed priority scheduling

3.1 Goal

The goal of this lab is to become familiar with OSEK/VDX applications development process and with Trampoline and to understand how fixed priority scheduling works. We will see some Hook Routines too and Events. Trampoline is a Free Software implementation of the OSEK/VDX specification. Trampoline includes an OIL compiler which allows, starting from an OIL description, to generate OS level data structures of the application. In addition to the OIL description, the developer must provide the C sources of tasks and ISRs of the application.

Get the lab1 starting source files. Go at http://www.irccyn.ec-nantes.fr/~bechenne/trampoline-labs/ and get the first archive. Once downloaded it expands to a lab1 directory.

3.2 Starting point

Go into the lab1 directory. There are 2 files:

lab1.oil the OIL description of the lab1 application

lab1.cpp the C++ source for the lab1 task and hook routines (C++ is needed because of LiquidCrystalFast library)

Edit the lab1.oil and look at the `TRAMPOLINE_BASE_PATH` attribute (in OS > BUILD attribute). `TRAMPOLINE_BASE_PATH` is set to `""`. Set it to the path to Trampoline. Tip: open a new Terminal, type `cd trampoline` and then `pwd`[1]. Copy-paste the path that is displayed into the `TRAMPOLINE_BASE_PATH` attribute.

lab1 is a very simple application with only 1 task called `a_task`. `a_task` starts automatically (`AUTOSTART = TRUE { ... }` in the OIL file). Look at the OIL file and the C source file.

To compile this application, go into the lab1 directory and type:

```
goil --target=cortex/armv7/mk20dx256/teensy31 lab1.oil
```

The `-t` option gives the target system (here we generate the OS level data structures of Trampoline for a `cortex` core with the `armv7` instruction set, a `mk20dx256` micro-controller and the board is a `teensy`). The OIL file gives the names of the C source files (with `APP_SRC` for a C file or `APP_CPPSRC` for a C++ file) and the name of the executable file (with `APP_NAME`).

This generates a build script for the application (files `make.py` and `build.py`). It has to be done only once. If you change something in the OIL file or in your C++ file, you do not need to re-run the goil compiler because the build script will run it when needed. Then type:

```
./make.py
```

The application and Trampoline OS are compiled and linked together. To upload the application to the Teensy, type

```
./make.py burn
```

The following message is displayed:

```
Teensy Loader, Command Line, Version 2.0
Read "lab1_exe.hex": 12660 bytes, 4.8% usage
Waiting for Teensy device...
 (hint: press the reset button)
```

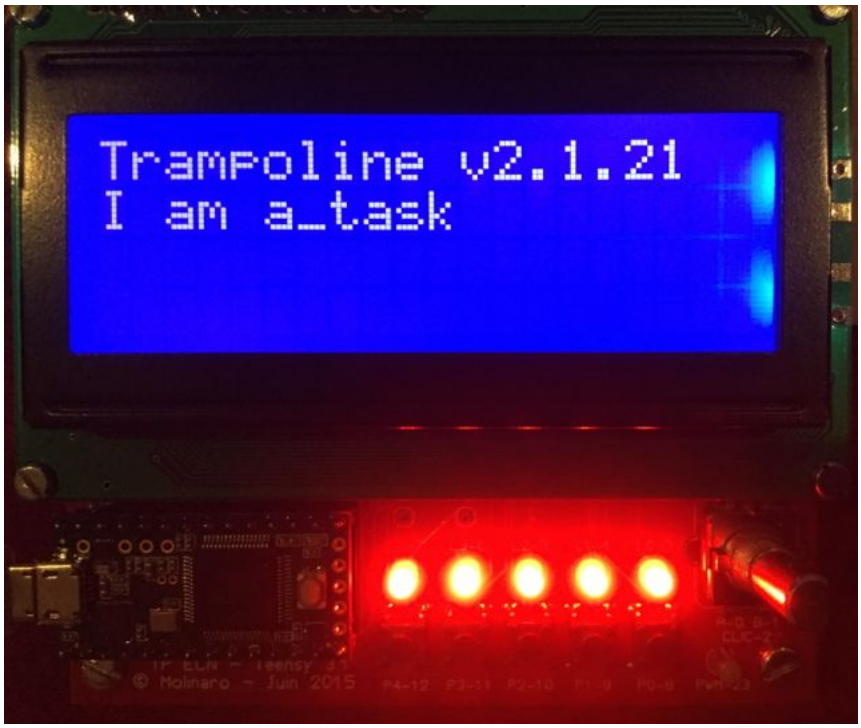
Press the reset button of the Teensy. Teensy loader uploads the binary to the flash of the micro-controller and displays:

```
Found HalfKay Bootloader
Read "lab1_exe.hex": 12660 bytes, 4.8% usage
Programming.............
Booting
```

The program starts and the following message should be displayed on the LCD (The first line corresponds to the execution of the main and the second line corresponds to

[1] Print Working Directory

the execution of task a_task). In addition, task a_task turns on all the LEDs.



3.2.1 A word about memory sections

AUTOSAR defines a way to put objects: constants, variables and functions in memory sections in a portable way[2]. For that, a set of macro are used along with a generated file : MemMap.h. Functions should be declared with the `FUNC` macro, variables with the `VAR` macro, constants with the `CONST` macro and pointers to variables, pointers to constant, constant pointers to variable and constant pointers to constant with `P2VAR`, `P2CONST`, `CONSTP2VAR` and `CONSTP2CONST` respectively. Sections are opened and close with a macro definition and the inclusion of the `tpl_memmap.h` file. For instance:

```
#define APP_Task_my_periodic_task_START_SEC_VAR_32BIT
#include "tpl_memmap.h"
VAR(int, AUTOMATIC) period;
VAR(int, AUTOMATIC) occurence;
#define APP_Task_my_periodic_task_STOP_SEC_VAR_32BIT
#include "tpl_memmap.h"
```

defines variables `period` and `occurence` in the variables section of task `my_periodic_task`.

```
#define APP_Task_my_periodic_task_START_SEC_CODE
#include "tpl_memmap.h"
```

[2]memory section declaration is not part of the C standard

```
TASK(my_periodic_task)
{
  ...
  TerminateTask();
}
#define APP_Task_my_periodic_task_STOP_SEC_CODE
#include "tpl_memmap.h"
```

defines the task `my_periodic_task` in the code section of task `my_periodic_task`. `goil` generates the sections for tasks according to the description.

3.3 OS system calls and task launching

3.3.1 Task activation and scheduling

The `ActivateTask()` system call allows to activate another task of the application.

Question 1 *Add two tasks in the system:* `task_0` *and* `task_1`*.*

- *add the declaration of both tasks in the OIL file;* `task_0` *should have priority 1 and its AUTOSTART attribute should be set to FALSE;* `task_1` *should have priority 8 and its AUTOSTART attribute should be set to FALSE;*
- *in the cpp file, you should:*
  - *declare both tasks with the DeclareTask keyword,*
  - *provide the body of both tasks: each task prints its name on a line of the LCD and then terminates,*
  - *and, lastly, modify the body of task* `a_task` *to activate* `task_0` *and* `task_1` *(in this order).*

Compile and execute. Why does `task_1` *execute before* `task_0` *whereas it has been activated after?*

3.3.2 Task chaining

The `ChainTask()` system call allows to chain the execution of a task to another one. This is roughly the same thing as calling ActivateTask and TerminateTask.

Question 2 *Replace the call to* `TerminateTask` *by a* `ChainTask(task_1)` *at the end of task a task. What is happening?*

Question 3 *Chain to* `task_0` *instead of* `task_1`*. Draw a schedule of the execution of the system. What is happening?*

Question 4 *Test the error code returned by ChainTask and correct your program to handle the error.*

3.4 Extended tasks and synchronization using events

Unlike a basic task, an extended task may wait for an event. Before to proceed with the following questions, consult the slides of the course to become familiar with events in Trampoline.

Question 5 *Modify the application of Question 1:*

- *set priority of* `task_0` *to 8;*
- *add two events,* `evt_0` *and* `evt_1`*:*
  - `evt_0` *is set by task* `a_task` *to task*`task_0`
  - `evt_1` *is set by task* `a_task` *to task*`task_1`
- *modify the body of the tasks:*
  - *task* `a_task` *activates* `task_0` *and* `task_1` *then sets* `evt_0` *and* `evt_1` *before to terminates*
  - *task* `task_0` *and* `task_1` *wait for their event, clear it, and terminate.*

Before to run the application, draw a schedule of the execution. Add outputs in the bodies of the task (for instance writes to the LCD or the LEDs) to verify your schedule.

Question 6 *Program an application conforming to the following requirements:*

- *it is composed of two tasks:* `server` *priority 2,* `t1` *priority 1.*
- `server` *is an infinite loop that activates* `t1` *and waits for event* `evt_1`*.*
- `t1` *prints "I am t1" and sets* `evt_1` *of* `server`*.*

Before to run the application, draw a schedule of the execution. Add outputs in the bodies of the task (for instance writes to the LCD or the LEDs) to verify your schedule.

Question 7 *Extend the previous application by adding 2 tasks:* `t2` *and* `t3` *(priority 1 for both) and 2 events* `evt_2` *and* `evt_3`*.* `server` *activates* `t1`*,* `t2` *and* `t3` *and waits for one of the events. When one of the events is set,* `server` *activates the corresponding task again.*

Chapter 4

Lab 2
Periodic tasks and Alarms

4.1 Goal

Real-Time systems are reactive systems which have to do processing as a result of events. You have seen in Lab #1 how to start processing as a result of an internal event of the system: by activating a task (`ActivateTask` and `ChainTask` services) or by setting an event (`SetEvent` service). In this lab, you will see how to trigger processing as a result of time passing (expiration of an Alarm). This lab uses the following concepts: alarm and counter. On the TP ECN board, the Systick timer is used as interrupt source for alarms. The interrupt is sent every 1ms.

Go to http://www.irccyn.ec-nantes.fr/~bechenne/trampoline-labs and download the lab2 package.

4.2 First application

lab2 application starting point This application implements a periodic task that reads push button P0_8 of the board every 100ms. Take a look and understand the `readButton` function that implements a small state machine to convert the state of the button to states and events. We are going to use events: `BUTTON_PUSH` and `BUTTON_RELEASE`. Compile and load this application to the board. Verify it works: pressing button P0_8 should turn LED L0_3 on. Releasing button P0_8 should turn LED L0_3 off. In all the applications we will use the lab2 application starting point to read buttons.

Using the application of lab2 as a starting point, program an application which does a computation when a button is pushed. You will use a task named `t_process`, priority

3 that on odd execution displays "processing triggered" and on even executions clears the LCD.

4.3 Second application

The second application will use 2 periodic tasks: `t1` (priority 2, period 1s) and `t2` (priority 1, period 1.5s). t1 toggles LED L0_3 each time it executes and t2 toggles LED L1_4.

Question 8 *Do not program directly. Give by hand the 20 first states of the LED given by the execution of the application with the display date of each state (0 being the application startup date). Is the whole system periodic ? If yes, what is the period and the behavior.*

Question 9 *The application needs a counter and 2 alarms. In Trampoline/ARM a counter is connected to a timer with a 1ms cycle time. What maximum* `TICKSPERBASE` *do you use to fulfill the application requirements ?*

Question 10 *How are the alarms configured to fulfill the application requirements ? Declare the counter and both alarms and write the application. Verify it works.*

4.4 Third application

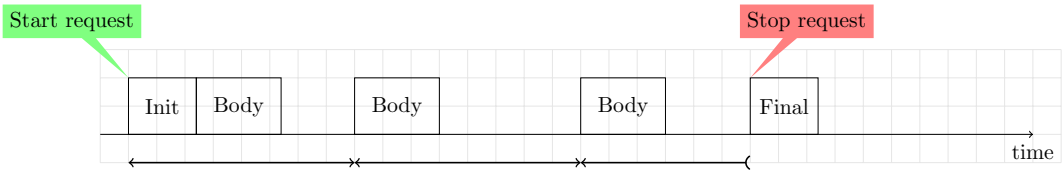
In the third application, alarms, counters and polling of the push buttons are mixed. This application is a system with 2 push buttons. After starting the system waits. When the button is pressed, the system start a *function*[1] F that is implemented using a periodic task (period = 1s). To "see" F, uses a blinking LED as in the second application. When the button is pressed again, function F is stopped. When the switch is pressed, the system is shutdown as quickly as possible (ie ShutdownOS[2] is called).

Question 11 *Design and program this application using Trampoline.*

Requirements change. Now function F implementation needs an Init code (runs once when the F is started) and a Final code (runs once when F is stopped). This corresponds to the following diagram:

[1] Here we mean a function of the application, not a function of the C language.

[2] The ShutdownOS service shutdowns the operationg system. All tasks and alarms are stopped. ShutdownOS takes a StatusType argument to specify the kind or error which occurred. In our case use E_OK as argument.



Question 12 *Modify the application to take the new requirements into account. Use 3 basic tasks to implement function F. Init and Final print their names on the LCD.*

Question 13 *Same question but with only one extended task to implement function F.*

4.5 Fourth application

In this part, you will implement a watchdog. It is a mechanism that allows to stop a processing or the waiting for an event when a deadline occurs.

Question 14 *In your application, each time P0_8 is pressed, P1_9 must be pressed within 2 seconds. In such case, you print the time between the two occurrences. Otherwise, an error message is displayed. If 2 or more P0_8 are got within 3 seconds from the first one they are ignored.*

Question 15 *What is happening if the timeout occurs just after P1_9 has been pressed but before the waiting task got the event ? (draw a Gantt diagram of this scenario) If your application does not handle correctly this scenario, modify it.*

4.6 Fifth application

Program a chase[3] with a 0.5s period. To do it, use 4 periodic tasks. Each periodic task manages a LED. The chase effect is done by using alarms with a time shift between them.

When P0_8 is pressed, the chase stops. When P1_9 is pressed, the chase continues. When P2_10 is pressed, the chase direction changes (even if it is stopped).

[3] chenillard in French

Chapter 5

Lab 3
Shared object access protection

To show resources usage, we will use a bad program that allows to corrupt a shared global variable which is not protected against concurrent writes. This has been presented in the course. This lab will show different ways to prevent this wrong behavior by using resources (standard and internal) or other solutions (preemption and priority).

Get the lab3 package directory from http://www.irccyn.ec-nantes.fr/~bechenne/trampoline-labs. The function needed in Question 21 is in `lab3.c`.

5.1 Application requirements

The application has 3 tasks and 2 global variables **declared with the volatile keyword**: `val` and `activationCount` as shown in figure 5.1:

- a background task called `bgTask`, active at start (`AUTOSTART = TRUE`) that never ends. In an infinite loop this task increments then decrements the global variable `val`. This task has a priority equal to 1.
- a periodic task called `periodicTask` that runs every 100ms[1]. This periodic task increments the global variable `activationCount`. If `activationCount` is odd, `val` is incremented, otherwise it is decremented.
- a periodic task `displayTask` that runs every second and prints on the standard output `val` and `activationCount`.

Question 16 *Before programming the application, gives the values that should be displayed for val on the standard output.*

[1] a counter gets a tick every 10ms

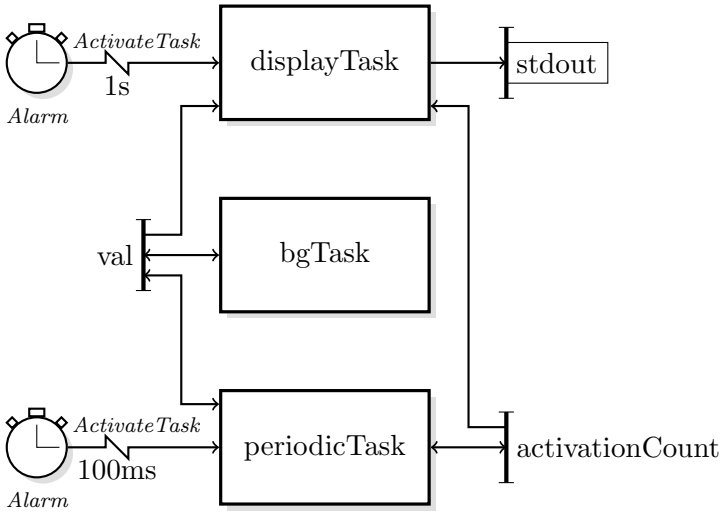


Figure 5.1: Application diagram

Describe the application in OIL and program it in C.

Question 17 *Does the behavior correspond to what you expect ? Why ?*

5.2 Global variable protection

Question 18 *Update the OIL file and the C program to protect the access to the global variable* `val`*. Use a resource to do it.*

The resource priority is automatically computed by goil according to the priorities of the tasks which use it.

Question 19 *What priority will be given to the resource ?*

The OIL compiler (goil) generates many files in the directory bearing the same name as the oil file (less the .oil suffix). Among them 3 are interesting:

- `tpl_app_define.h`
- `tpl_app_config.h`
- `tpl_app_config.c`

The file `tpl_app_config.c` contains the tasks' descriptors as long as all other data structures. These structures are commented.

Question 20 *For each task, find the priority computed by goil and the identifier. Is is the same as defined in the OIL file? if not is it aproblem?*

Question 21 *What is the priority of the resource ? Is it consistent ?*

PCP rule requires the task priority is raised to the resource priority when the resource is taken. We will show this behavior by displaying the priority of the currently running task. Since OSEK does not have a function to do that, we use the following function:

```
void displayIdAndCurrentPriority()
{
  TaskType id;
  GetTaskID(&id);
  if (id >= 0)
  {
    tpl_priority prio = (tpl_dyn_proc_table[id]->priority) >> PRIORITY_SHIFT;
    lcd.print("Id=");
    lcd.print(id);
    lcd.print(",␣Prio=");
    lcd.print(prio);
  }
}
```

And you have to add the following line at start of your C file:

```
#include "tpl_os_task_kernel.h"
```

Call the function from your tasks when the resource is not taken and when it is taken.

Question 22 *Is the behavior ok ? Explain.*

5.3 Protection with an internal resource

An internal resource is automatically taken when the task gets the CPU. Replace the standard resource by an internal resource in the OIL file. Remove the `GetResource` and `ReleaseResource` in the C file.

Question 23 *What happens ? Why ?*

Modify the task `bgTask`: instead of infinite loop, use a `ChainTask` to the `bgTask` (ie: the task chains to itself).

Question 24 *Question 10 What happens ? Explain.*

5.4 Protection using a single priority level

Keep the version with the `ChainTask` instead of the infinite loop. Modify the OIL file: remove the resource and set the priorities so that no task can be preempted.

Question 25 *What happens ? Why ?*

Chapter 6

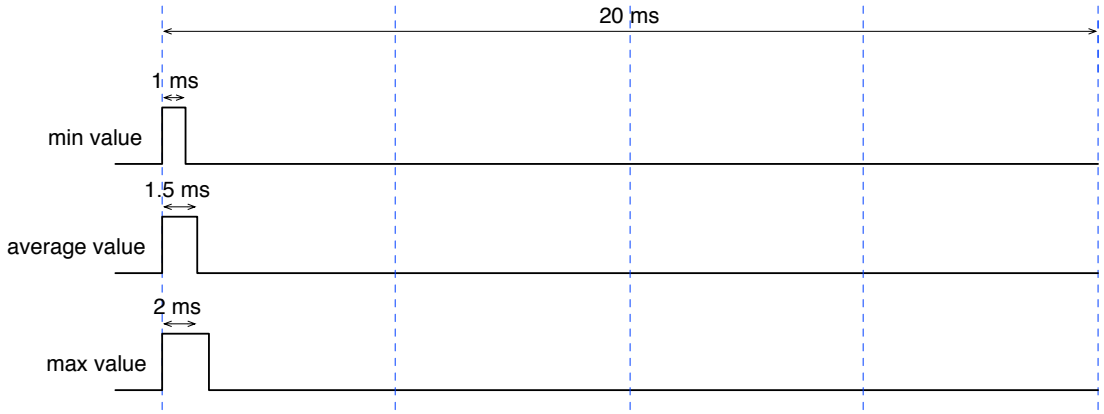
Lab4
Driving servos

The goal of this lab is to build a small application to drive servomotors. Get the lab4 package from http://www.irccyn.ec-nantes.fr/~bechenne/trampoline-labs/. Inside there are 2 files: `servos.h` and `servo.cpp`.

Update Trampoline by going to the trampoline directory and typing `git pull`. Compile goil.

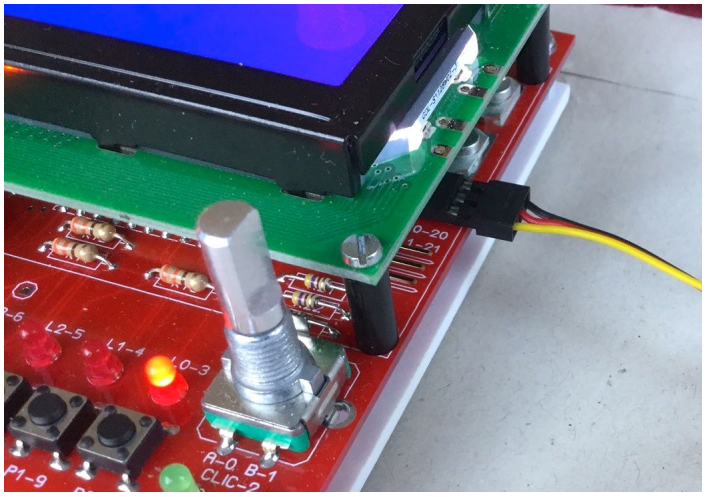
6.1 How servos work

Servos are driven by using a PWM signal. The width of the signal at the high logic state specifies the position of the servo. The width may range from 0.5ms to 2.5ms for a 180 rotation. In practice servos may not be able to turn by 180 and we are going to use a pulse width ranging from 1ms to 2ms. The signal should repeat every 20ms.



6.2 Servo PWM generation

The board has 2 connectors for servos on the right side just below the LCD. Servos must be connected with the **yellow wire nearby you** as shown on the picture below.

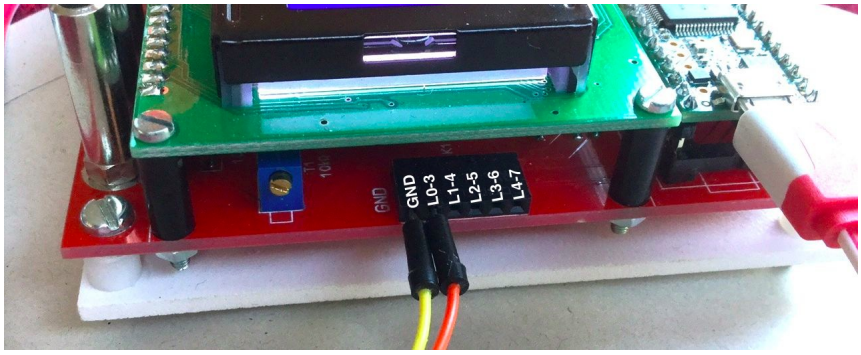


The 3.3V regulator of the Teensy is not able to power the servos. We need an external power supply connected to the bigs female banana plugs on the top of the board. Get 2 banana cords and connect them to the power supply and to the board as shown on the picture below.

We are going to drive 2 servo. Each servo is driven by a task : `t_servo0` and `t_servo1`.

Question 26 *Program tasks* `t_servo0` *and* `t_servo1`*. Both tasks are periodic with a 20ms period but are offset by half of the period.* `t_servo0` *toggles LED L0-3 and* `t_-`

`servo1` toggles LED L1-4. Verify the period using the scope. On the left there is a connector where LED signals are available as well as the GND. Use this connector to connect the scope probes. This connector is shown below



Now instead of toggling the LED, each task will use the function `pulse`. This function takes 2 arguments. The first one is the `output`, the second one is the `duration` of the pulse in μs. `pulse` does not allow values lower than 1000 and greater than 2000 in order to avoid to damage the servo. `pulse` works as follow: the `ouput` is set to 1 and a timer of the microcontroller is programmed to generate an interrupt after `duration` μs. Since the same timer is used for both tasks, the tasks have to be shifted by at least the maximum duration of the pulse. In your OIL file, you have to declare the ISR which is located in the `servo.c` file as follow:

```
ISR ftm_timer {
  CATEGORY = 1;
  PRIORITY = 1;
  SOURCE = FTM0_IRQ;
};
```

You have to add the `servo.c` file in the sources of the application and the fact you use the ftm library in the BUILD attribute of the OS object:

```
    APP_SRC = "servo.c";
    LIBRARY = ftm;
```

Question 27 *Modify `t_servo0` and `t_servo1` to use `pulse` and generate a pulse with a width equal to 1.5ms. Verify the behavior using the scope.*

Then change the pins from 3 and 4 to 20 and 21. 20 and 21 correspond to the servo connectors on the right. Connect the servos to the connectors Each servo should go to the average position. Do not forget to program pins 20 an 21 as `OUTPUT`.

6.3 Adding high level behavior

Now we want to set the position of each servo. To do that we need 2 tasks, 1 for each servo, that set the position in a global variable, 1 for each servo. The corresponding variable is read by the `t_servoX` task to drive the servo.

Question 28 *First we want the position of each servo incremented until it reaches its maximum position (2ms pulse), then decremented until it reaches its minimum position (1ms pulse). So the servo does round trips. The time we want to go from the minimum position to the maximum position is 5s. Assuming the position is incremented and decremented by one, what is the period of theses tasks in counter tick unit ?*

Now we want to be able to set the minimum and maximum positions of the servo connected to pin 20. To do that we use buttons.

Question 29 *Add to the previous application the following features and draw an automaton to model the application:*

When button P0-8 is pressed, servo connected to pin 20 stops its round-trip. If it is pressed again, round-trip continue. When the round-trip is stopped the following functions are available.

When button P4-12 is pressed, the minimum position is selected, servo connected to pin 20 goes to this position. Then push buttons P2-10 decrements the minimum position and P1-9 increments the minimum position.

When button P3-11 is pressed, the maximum position is selected, servo connected to pin 20 goes to this position. Then push buttons P2-10 decrements the minimum position and P1-9 increments the minimum position.

Minimum position should be kept greater or equal than the position corresponding to the 1ms pulse. Maximum position should be kept lower or equal than the position corresponding to the 2ms pulse. Minimum position should be kept lower or equal than the maximum position.

When button P0-8 is pressed, servo connected to pin 20 continues its round-trip and respects the minimum and maximum positions that have been set.