Engineering and Physical Science

Embedded Software – B31DG
Assignment 3

Sylvain Jannin

H00387879

Summary

Table of figures

1. Context and electronic design

The goal of this assignment is to practise the use of FreeRTOS. There are 9 tasks, each task is easy to implement, and they are all executing with a different frequency.

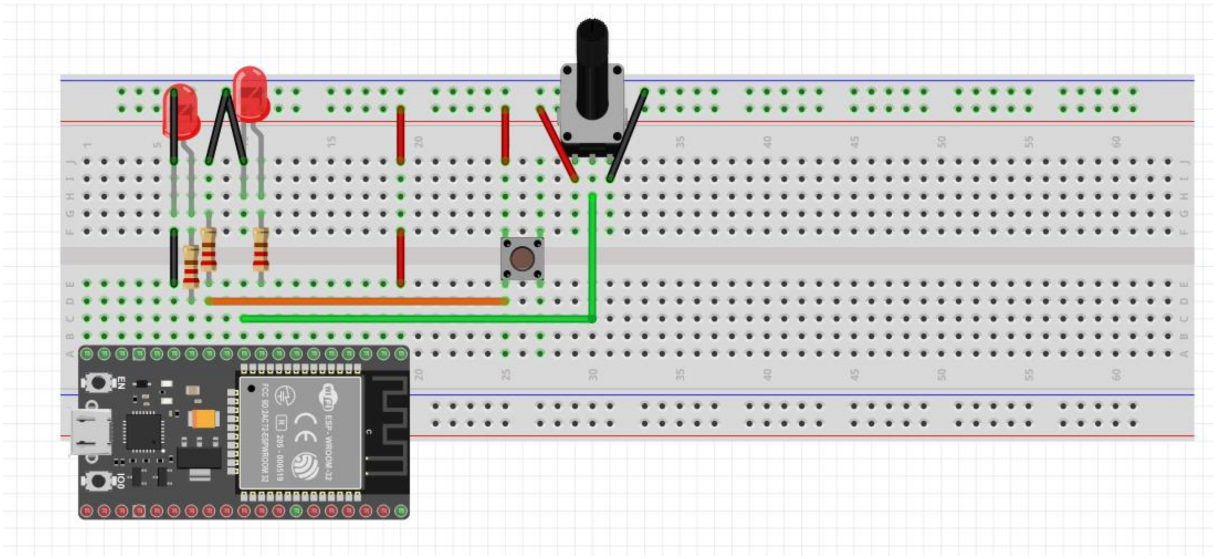


Figure 1 : electric schematic.

On the figure above there is the electric schematic. For all inputs, there are pull-down resistors in order to have better signals. There are also an LED and a resistor for task 1 to measure the signal.

On the figure below, the electronic assembly.

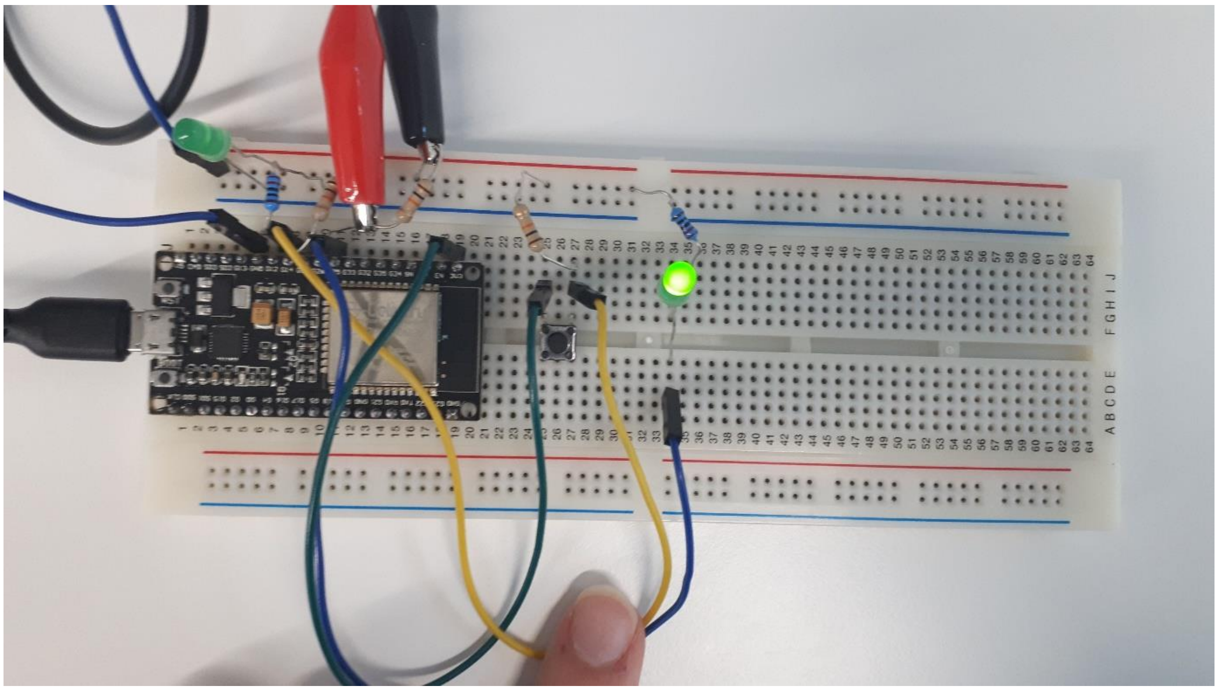


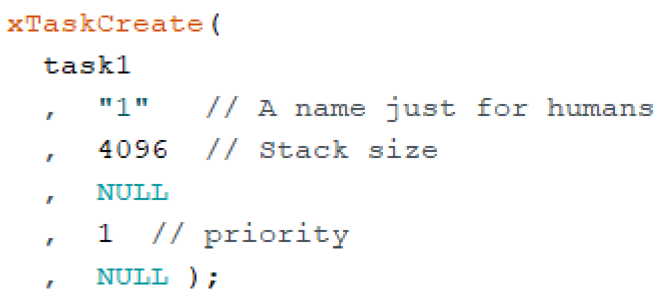
Figure 2 : electric assembly pictures.

2. Programming

2.1. FreeRTOS

FreeRTOS is a real time operating system. It makes the implementations of tasks that works together easy.

To initialises a task, we use the *xTaskCreate* function for each task in the setup function. The first parameter is the name of the function and the third one is the stack size. If the stack size is too low, it can make the EPS32 reboot.

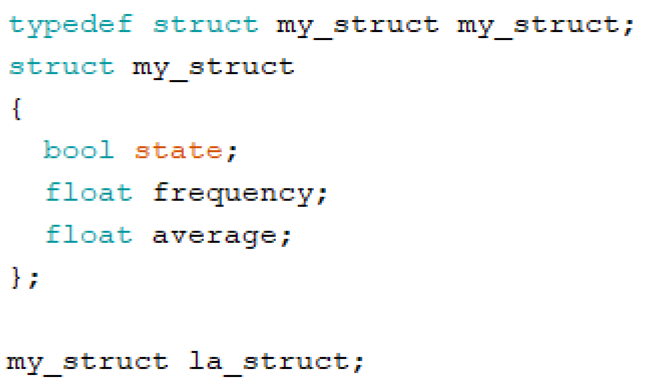


```
xTaskCreate(
  task1
  ,  "1"   // A name just for humans
  ,  4096  // Stack size
  ,  NULL
  ,  1  // priority
  ,  NULL );
```

Figure 3: task creation example.

The functions are called only one time and then run in an infinite for loop. We use *vTaskDelay*. This function works more like a timer than a delay: it lets other function working and do not block the code like a delay, but it waits for this time indicated before executing the code.

The state of the button, the frequency from task 3 and the average from the analogue inputs are stored in the same structure *my_struct*.

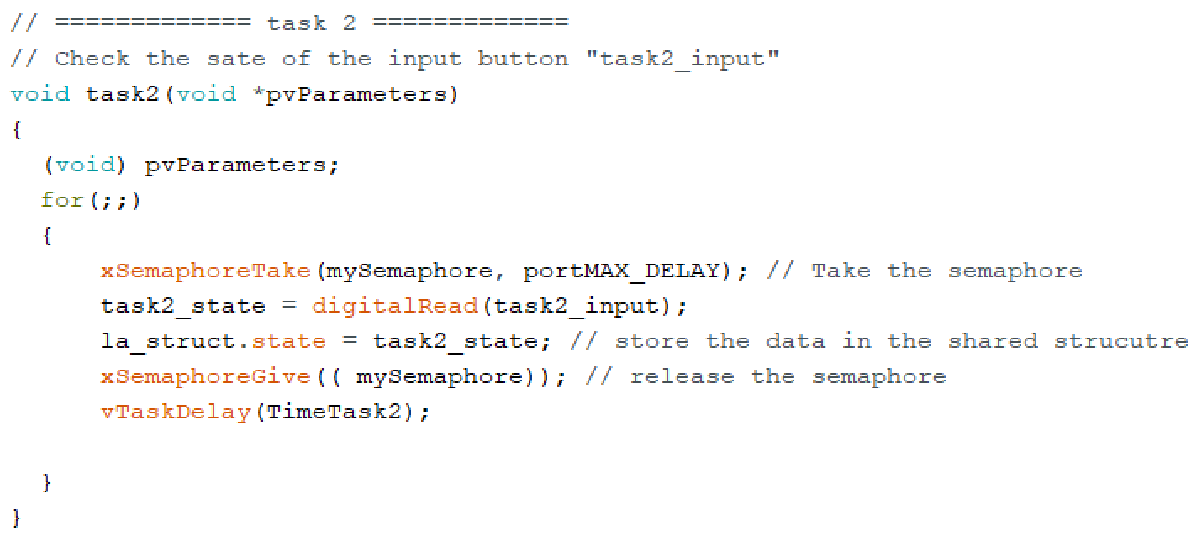


```
typedef struct my_struct my_struct;
struct my_struct
{
  bool state;
  float frequency;
  float average;
};

my_struct la_struct;
```

Figure 4: Structure impelmentation.

We use a semaphore to protect the structure. To initialise a semaphore, we used the function *xSemaphoreCreateMutex*. To take the semaphore we use the function *xSemaphoreTake* and to release it we use the function *xSemaphoreGive*. By doing this me make unable two tasks to access the shared structure at the same time. The first task who call the *xSemaphoreTake* function make the second task waits until the function *xSemaphoreGive* is classed in task1.



```
// ============= task 2 =============
// Check the sate of the input button "task2_input"
void task2(void *pvParameters)
{
  (void) pvParameters;
  for(;;)
  {
      xSemaphoreTake(mySemaphore, portMAX_DELAY); // Take the semaphore
      task2_state = digitalRead(task2_input);
      la_struct.state = task2_state; // store the data in the shared strucutre
      xSemaphoreGive(( mySemaphore)); // release the semaphore
      vTaskDelay(TimeTask2);

  }
}
```

Figure 5: Exemple task implementation.

To transmit data from task 4 to 5, we use a queue. The queue is used as a bridge between the two tasks. In task 4 we read the analogue input, then we send the data in the queue *myQueue* with the *xQueueSend* function. In task 5, we receive the data the *xQueueReceive* function. To make sure a data is received we check what *xQueueReceive* returns, if it is different *pdPASS* there is an issue with the queue. To initialise the queue, we call the function *xQueueCreate* in the setup function.

2.2. Tasks

All tasks are easy to implement, and they are call in the function at the good rate (T is in ms).

- **Task 1**:
  - T = 9,250 ms rounded at 9ms (based on signal b of assignment 1)
  - Send a signal high for 50 µs then low
  - Use delay function
- **Task 2:**
  - T = 200 ms
  - Check the state of an input
- **Task 3:**
  - T = 1000 ms

  - Compute the frequency on an input signal
  - Use an interrupt on rising edge to read the time at every rising edge
  - Use global variable to note the last to time the external signal rose
  - $f = \frac{10^6}{current_{time} - previous_{time}}$. The measure of time is in µs, so we need to multiply the frequency by 10^6 to have it in Hz.
- **Task 4:**
  - T = 42 ms (rounded)
  - Read an analogic input
- **Task 5:**
  - T = 42 ms (rounded)
  - Take the last 4 analogic inputs and do compute the average
- **Task 6:**
  - T = 100 ms
  - Send 1000 times the command "*__asm__ __volatile__ ("nop")*"
  - Use a for loop
- **Task 7:**
  - T = 333 ms (rounded)
  - Check if the average of the last 4 inputs is bigger than 1,65V
  - Change the value of *error_code* accordingly
- **Task 8:**
  - T = 333 ms (rounded)
  - Turn on or off a light regarding the value of *error_code.*
- **Task 9:**
  - T = 5000 ms
  - Print task 2 input state, task 3 input frequency and task 5 average analogic input.
  - Print a message only when the button is pressed.

3. Testing

We have the same result as assignment 2.

To test task 1, we did as in assignment one a check the signal on an oscilloscope. Our signal has a period of 9ms and a width of 50µs. It is verified with the captures in the next figures.

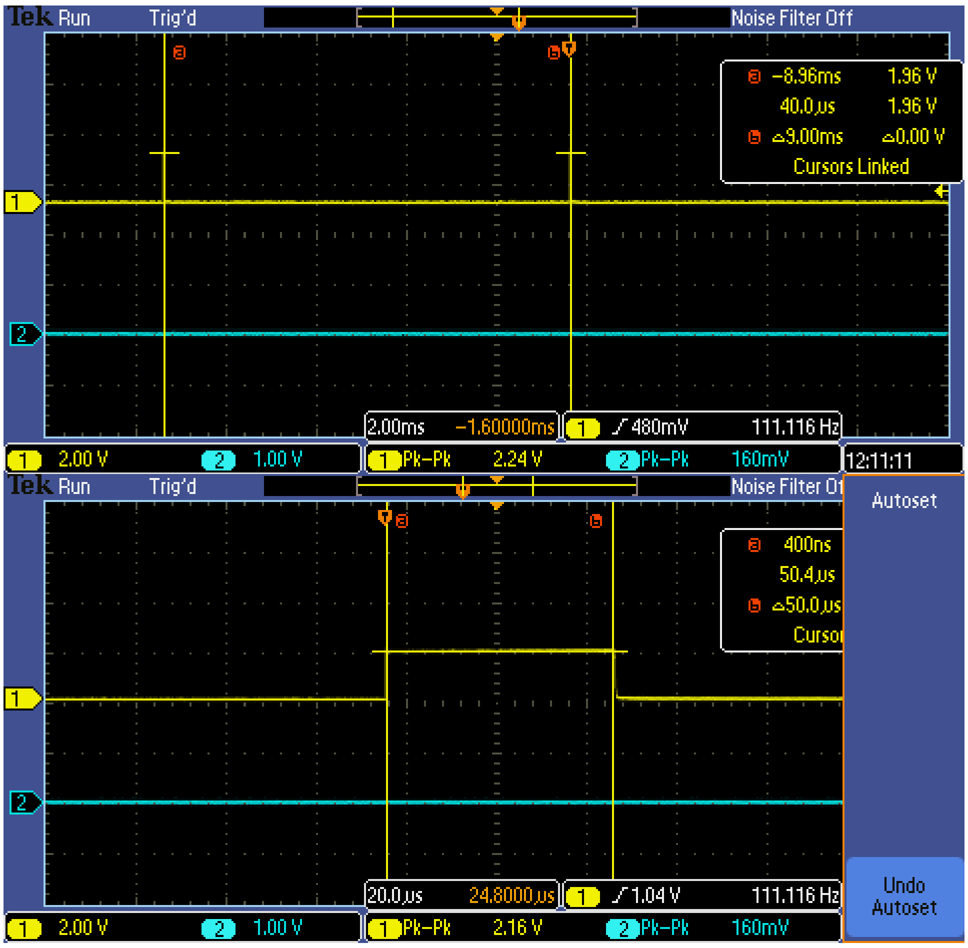


Figure 6: task 1 signal frequency and width.

For the task 2 there are 2 captures in the next figures that shows two different states for the input button.

Task 3 is the trickiest one to do. To check this, we used a DC generator to send a square signal with a frequency from 500 to 1000Hz. On the next figure there is a picture of the generator and of the signal used for the test.

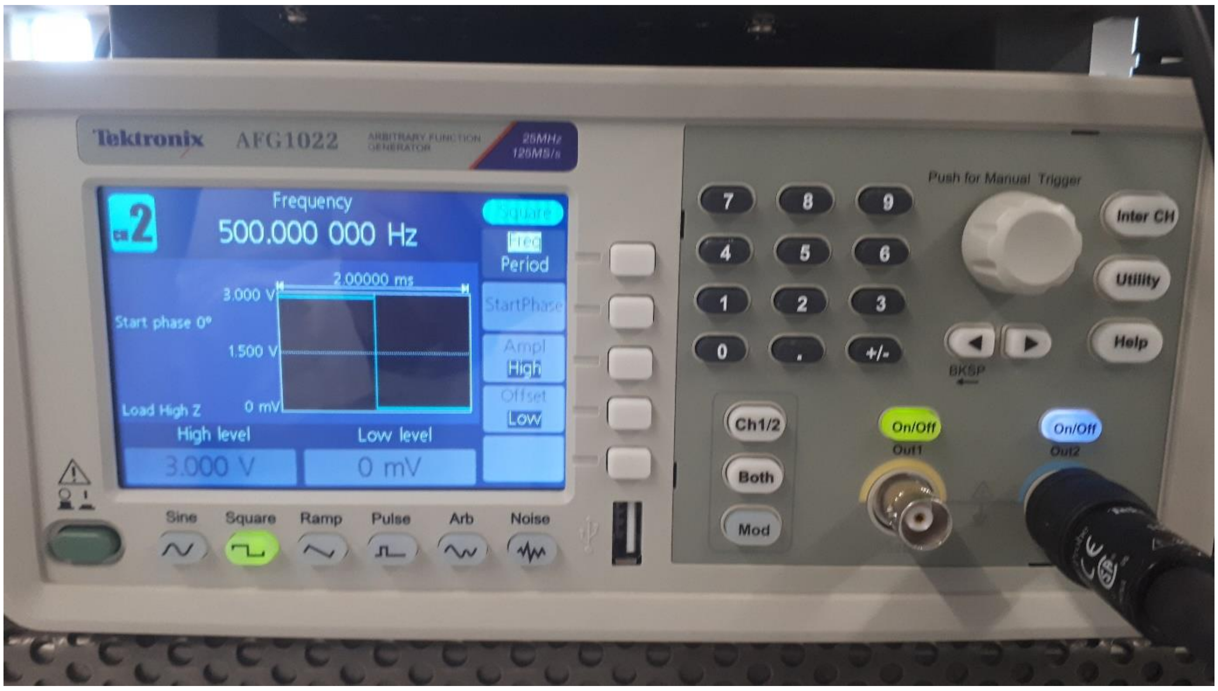
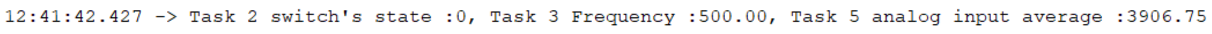


Figure 7: Picture the machine used to generate a square signal.

We wait for task 9 to print and we can check if the signal is the good one. On the following figure the signal sent was a square signal with a frequency of 500Hz.



```
12:41:42.427 -> Task 2 switch's state :0, Task 3 Frequency :500.00, Task 5 analog input average :3906.75
```

Figure 8: Capture of the task 9 print from ESP Monitor serie

For task 4 and 5, we check the input the same way as for task 3: we check the signal on the task 9 print. The average digital input is 3906.75. The tested signal was 3.05V high, to pass from a digital value to an analogue one we do: $2^{12} \times \frac{3.05}{3,3} \approx 3786$ which is a closed value from the one we measured.

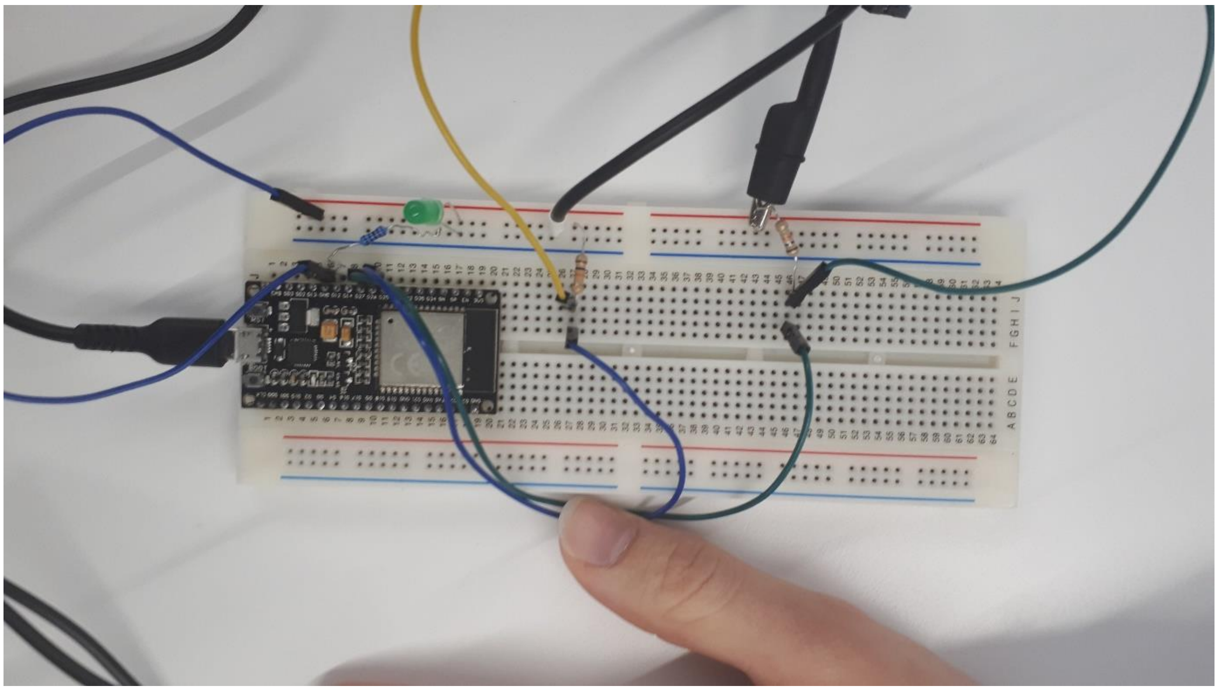
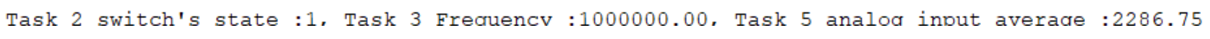


Figure 9: Electric assembly used to test task 3,4 and 5.

For task 7 and 8 we reused the generator used for task 4 and 5. And we add an LED to see if the LED in turning on or off regarding the analogue input. On the figure 2 we can see the LED on.

For task 9 there it is simply a print every 5 seconds, this print is used to check if other tasks are working correctly like in figure 6 or in the next figure.



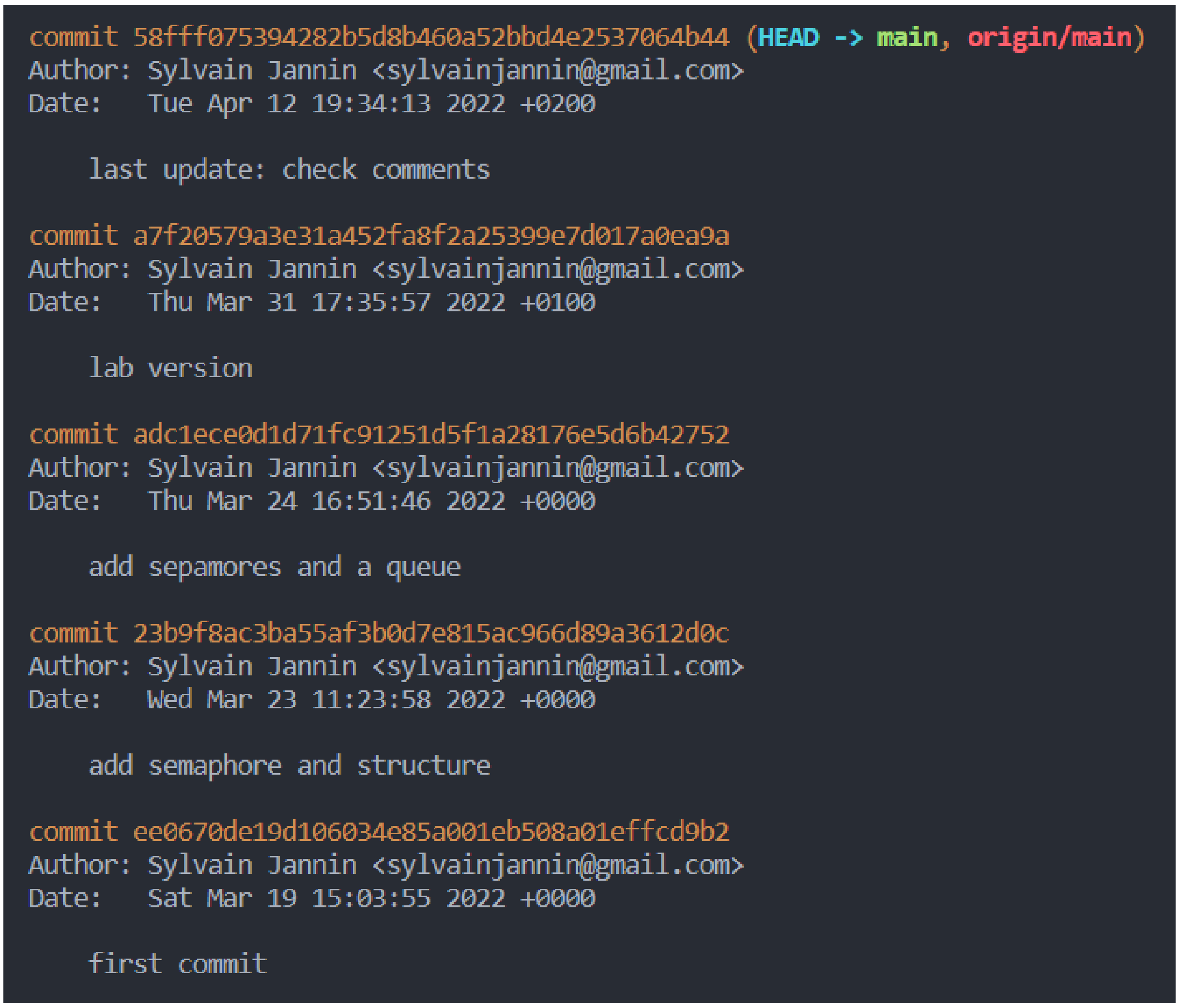
```
Task 2 switch's state :1, Task 3 Frequency :1000000.00, Task 5 analog input average :2286.75
```

Figure 10: Capture of task 9 with a different input analogue input and with task 2 switch on.

In conclusion, all the tasks are working properly. Furthermore, we can see that the tasks have the good timing because task 9 is printing every 5 seconds (there is not a moment where this task is skipped) and on figure 6 the period of task one (the most called) is respected. In addition, task 9 print the message only when the button from task 2 is pressed.

4. Git

My git repository for this class is https://github.com/SylvainJnn/EmbeddedSoftware . There is going to be only folder by assignments. On the figure below there is a part of the commit history.



```
commit 58fff075394282b5d8b460a52bbd4e2537064b44 (HEAD -> main, origin/main)
Author: Sylvain Jannin <sylvainjannin@gmail.com>
Date:   Tue Apr 12 19:34:13 2022 +0200

    last update: check comments

commit a7f20579a3e31a452fa8f2a25399e7d017a0ea9a
Author: Sylvain Jannin <sylvainjannin@gmail.com>
Date:   Thu Mar 31 17:35:57 2022 +0100

    lab version

commit adc1ece0d1d71fc91251d5f1a28176e5d6b42752
Author: Sylvain Jannin <sylvainjannin@gmail.com>
Date:   Thu Mar 24 16:51:46 2022 +0000

    add sepamores and a queue

commit 23b9f8ac3ba55af3b0d7e815ac966d89a3612d0c
Author: Sylvain Jannin <sylvainjannin@gmail.com>
Date:   Wed Mar 23 11:23:58 2022 +0000

    add semaphore and structure

commit ee0670de19d106034e85a001eb508a01effcd9b2
Author: Sylvain Jannin <sylvainjannin@gmail.com>
Date:   Sat Mar 19 15:03:55 2022 +0000

    first commit
```

Figure 11 : Capture of git log repository.