Electrical, Electronic and Computer Engineering
School of Engineering and Physical Sciences

Evrard EMONOT
H00385163

B31DG : Embedded Software

Assignment 3.

Summary

1 Introduction

In this assignment, we are going to execute tasks using an RTOS scheduler. All of those tasks are going to be executed at a certain frequency on the ESP32.

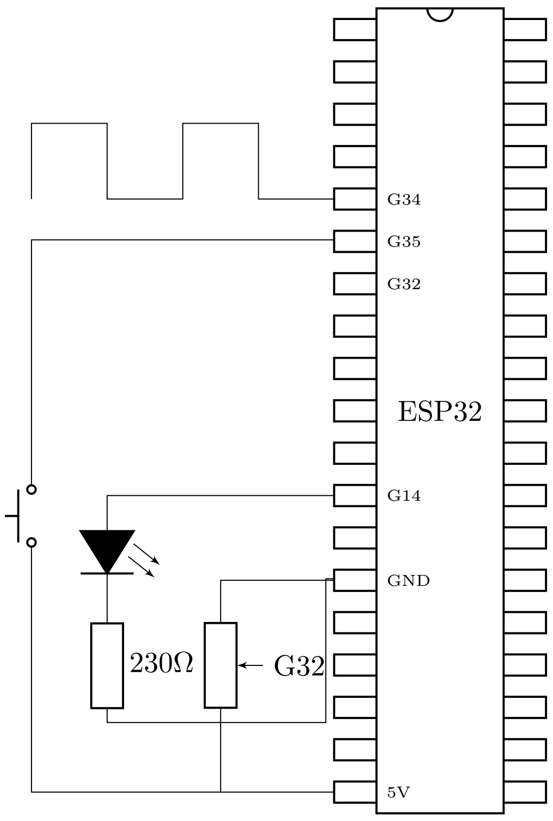
The commit and update of the project are available on that link : `https://github.com/Emonot/Embedded_Software_Assignement3`

2 Hardware

2.1 Schematics

Material used

- ESP 32
- 1 Resistor
- 1 Green LED
- 1 Switch
- 1 Potentiometer



3 Method and Software

3.1 Introduction

This program contains 10 tasks to achieve.

Each task has simple instructions, Some of them are independent but most of them communicates between each others. Several tasks gather external input or send output.

Compared to the structure of the 2nd assignment. We have a a new task n°10 that will call task n°9 if the button is pressed every 5 seconds. The data send through the serial port will be contain in a structure.

In addition, to secure the structure in the program. We use a semaphore, this semaphore will authorise the access one task at a time. By using it, there is no multiple access at the same time to the structure.

We also add a 2nd semaphore queue to send the data we read from task 4 to task 5.

3.2 Priority parameters

The tasks priority are implemented in a form of RM. It means that the shortest cycle task will be executed first. The frequency for each task are given in assignment 2.

```
xTaskCreate(task1,"task1",4096,NULL,3,NULL);
xTaskCreate(task2,"task2",4096,NULL,4,NULL);
xTaskCreate(task3,"task3",4096,NULL,7,NULL);
xTaskCreate(task4,"task4",4096,NULL,1,NULL);
xTaskCreate(task5,"task5",4096,NULL,2,NULL);
xTaskCreate(task6,"task6",4096,NULL,5,NULL);
xTaskCreate(task7,"task7",4096,NULL,6,NULL);
xTaskCreate(task8,"task8",4096,NULL,8,NULL);
xTaskCreate(task10,"task10",4096,NULL,9,NULL);
```

3.3 Structure implementation

```
struct Task10 {
   bool input_Btn_t2; //state of the button we read
   float frequency_measured_t3; //frequency of the square signal
   unsigned short average_potentiometer_t5; //value that will contain the average
    of the last 4 reads of the potentiometer
};
```

3.4 First example of Semaphore queue implementation

1st Example of semaphore implementation to access the structure.

```
//===== TASK10 =====
//launch task 9 if the button is pressed
//the task is called every 5 seconds
void task10(void *pvParameters){
  (void) pvParameters;
  for(;;){
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )//we use
    our semaphore to access the structure
    {
      if (struct_task10.input_Btn_t2 == 1) task9();

    xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for
    others.
    }
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
  }
}
```

3.5 Second example of Semaphore queue implementation

Example using a semaphore queue by receiving the potentiometer value inside the queue that has been recovered in task 4.

```
//===== TASK5 =====
//we calculate the average of the last 4 reads of the potentiometer
void task5(void *pvParameters){
  (void) pvParameters;
  for(;;){
    //FIFO list to gather all last potentiometer values
    all_potentiometer_t5[3] = all_potentiometer_t5[2];
    all_potentiometer_t5[2] = all_potentiometer_t5[1];
    all_potentiometer_t5[1] = all_potentiometer_t5[0];
    xQueueReceive(potentiometer_queue, all_potentiometer_t5, portMAX_DELAY);//we
    received our value from the queue of task4 outside the semaphore
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )//we use
    our semaphore to access the structure
    {

      struct_task10.average_potentiometer_t5 = 0;
      unsigned short i=0;
      for (i=0;i<=3;i++) struct_task10.average_potentiometer_t5 +=
    all_potentiometer_t5[i];
      struct_task10.average_potentiometer_t5 /= 4;

      xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port
    for others.
    }
    vTaskDelay( 41 / portTICK_PERIOD_MS ); //the task occurs every 41ms
  }
}
```

This semaphore queue is outside the second semaphore use for the structure as we don't need to protect the data on 2 layers.