Embedded Systems 2019/2020
Computer Engineering
University of Salerno

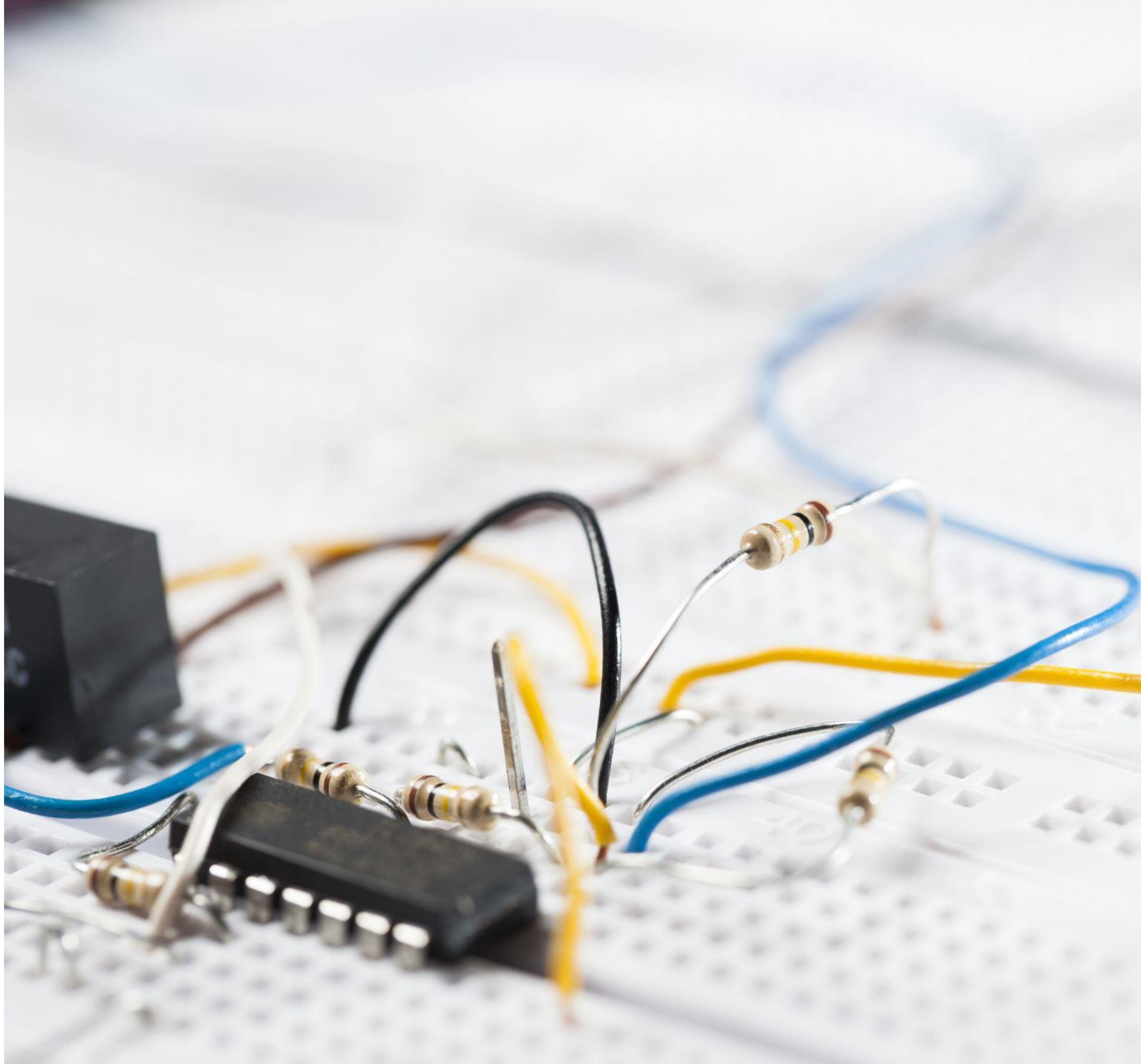
Home Security System

Stay At Home Group

Carmine Napolano, 0627701202

Emilio Sorrentino, 0622701199

Francesco Rosa, 0622701095

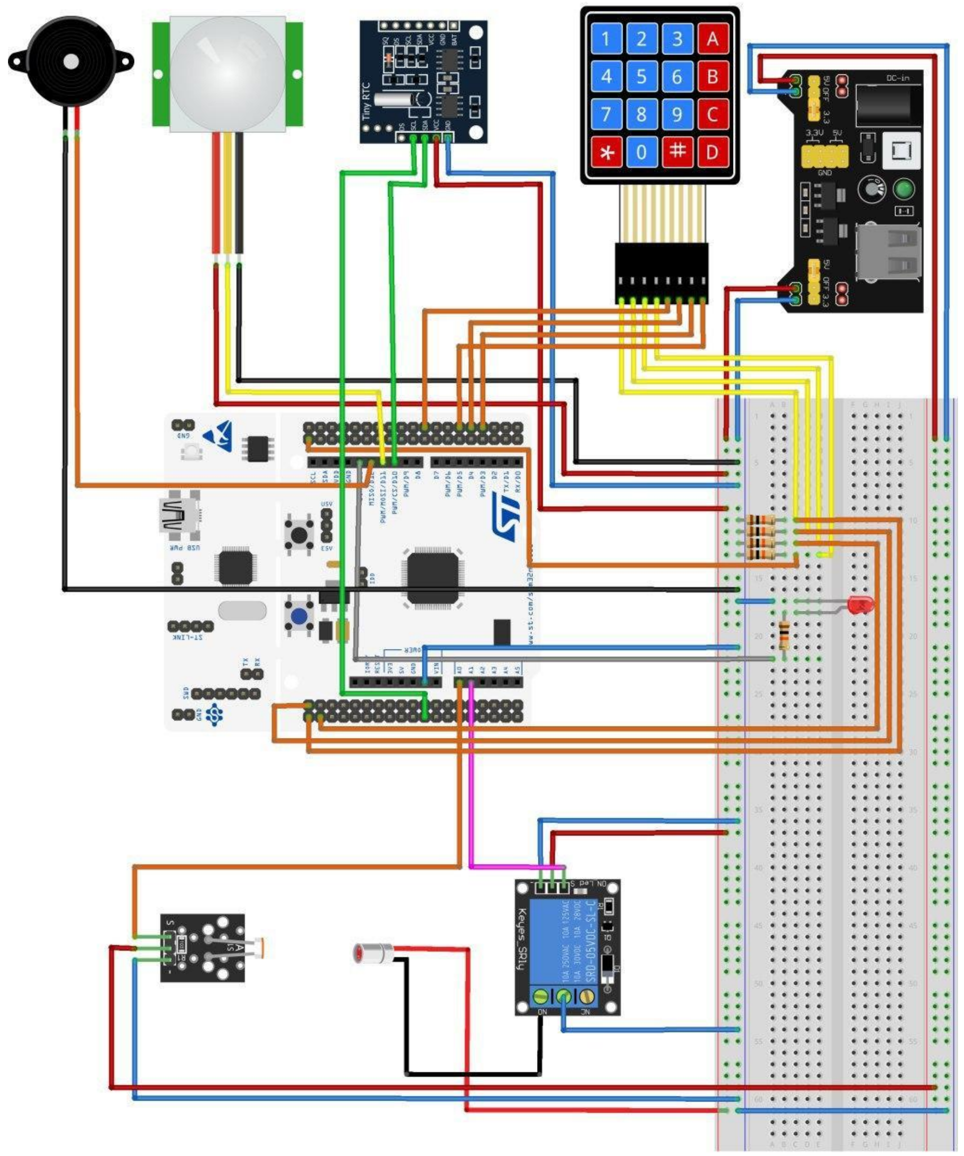


Hardware connection

This is the Hardware Connection scheme. The used component are (from the top left to bottom right):

- Active buzzer to emit the alarm sound
- Pir to detect movements
- RTC to save the current date and time
- Keypad 4x4 to insert the commands
- Power module to supply all the components
- Photoresistor and laser to realize the barrier sensor
- Relay to control the laser activation

The components are supplied by 5V except for the photoresistor (3.3 V).



Protocols

- **UART**

It is used to perform communication between the PC and the board. The UART protocol is used to allow:

  - user to insert its own configuration;
  - board to send the system status to the PC;
  - board to send the command result to the PC.

In the first two cases the protocol is used in interrupt mode. In particular, for the first case this mode allows to abort any kind of transfer after a particular event happened (the protocol duration time elapses).
About the second case, this operation mode is suitable for allowing the CPU to perform other operations since the non-blocking nature of this kind of mode.
In the third case the blocking mode has been chosen because there are not particular reasons for choose a non-blocking mode, since the message size is very small and no-particular preprocessing is required.

- **I2C**

It is used to perfom communication between the board and the DS1307 rtc module.
This protocol has been used in two different modes:

  - Blocking mode:
  - DMA mode;

The first mode has been used in two cases:

  - During the rtc initialization;
  - During the configuration protocol, when an update of the rtc memory is requested (user date-time inserted) or when data from the rtc must be taken (default configuration);

The second mode is used when the system log message must be prepared.
In the first case, there is no need to have a non-blocking mode, because the set of operation where these requests are performed is merely sequential.
Instead, in the second case, the system could receive asynchronous events from the external that have to be managed.

Timer

We have used different timers for different applications.
All of them are set to counter mode up with 15999 as prescaler, since the CPU clock frequency is set to 16 MHz (this is enough for proper components working).

TIM1/2

The first is used to check pir' signal stability, the second to blink system led

Period: 999

With this configuration the timer fires the update event after 1 s from its starting.

We manage it with PeriodElapsed_Callback

TIM3

Used in PWM mode to manage the buzzer sound

Period: 999

Duty cycle: 0

The duty cycle is changed based on which sensor is alarmed. The PWM signal repeats for "duration" time with a period of 1 s.

TIM10

Used for the system configuration time and, after it, for the system log

Period: initially set to 29999 (30 s) and, after the configuration, to 9999 (10 s)

We manege the PeriodElapsedCallback, in this way, every 10 s a system log message is sent.

TIM11

Used for delay/duration time of an alarm.

Period: changed based on which sensor is alarmed and the timer purpose. It can be:

- count duration →
  period = duration
- count delay →
  period = sensor delay

Keypad

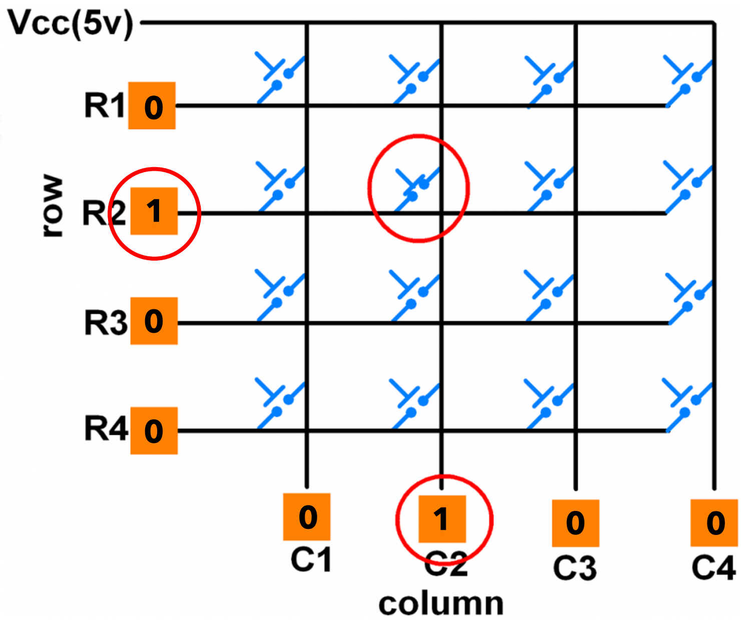
GPIO configuration

```
/*Configure GPIO pins : PB2 PB3 PB4 PB5 */
GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PC9 PC10 PC11 PC12 */
GPIO_InitStruct.Pin = GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

Keypad Matrix

```
char KEYPAD_INT_Buttons[4][4] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'},
};
```



The keypad is managed like a matrix where all the columns are set to 1 and all the rows are set to 0. When a key is pressed an interrupt (rising edge) is caught.
In order to get which button is pressed the following procedure is executed:

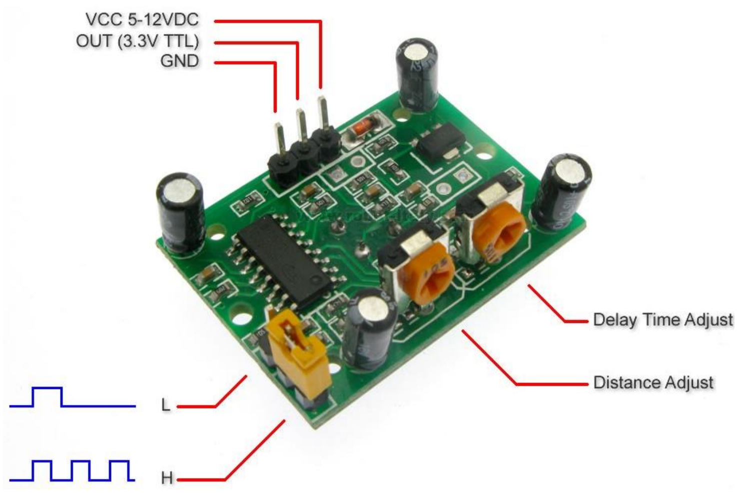
- All columns are reset;
- Each column is set, one by one, until a change on the row is detected.

With this procedure we are able to detect the column that is in short with the row that has raised the interrupt.

Area sensor

GPIO configuration

```
/*Configure GPIO pin : PA7 */
GPIO_InitStruct.Pin = GPIO_PIN_7;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```



We have set the PIR in «L mode», so the output is high (3.3 V) until a movement is detected (for this reason we have set delay time to the minimum value).
If the PIR detects a movement the output goes high and TIM1 is started:

- If the output remains high for the TIM1 period, the update event is raised, and by handling the PeriodElapsed_Callback, we set the PIR state to ALARMED.
- Otherwise, the timer is stopped before the update event and the sensor state is not changed.

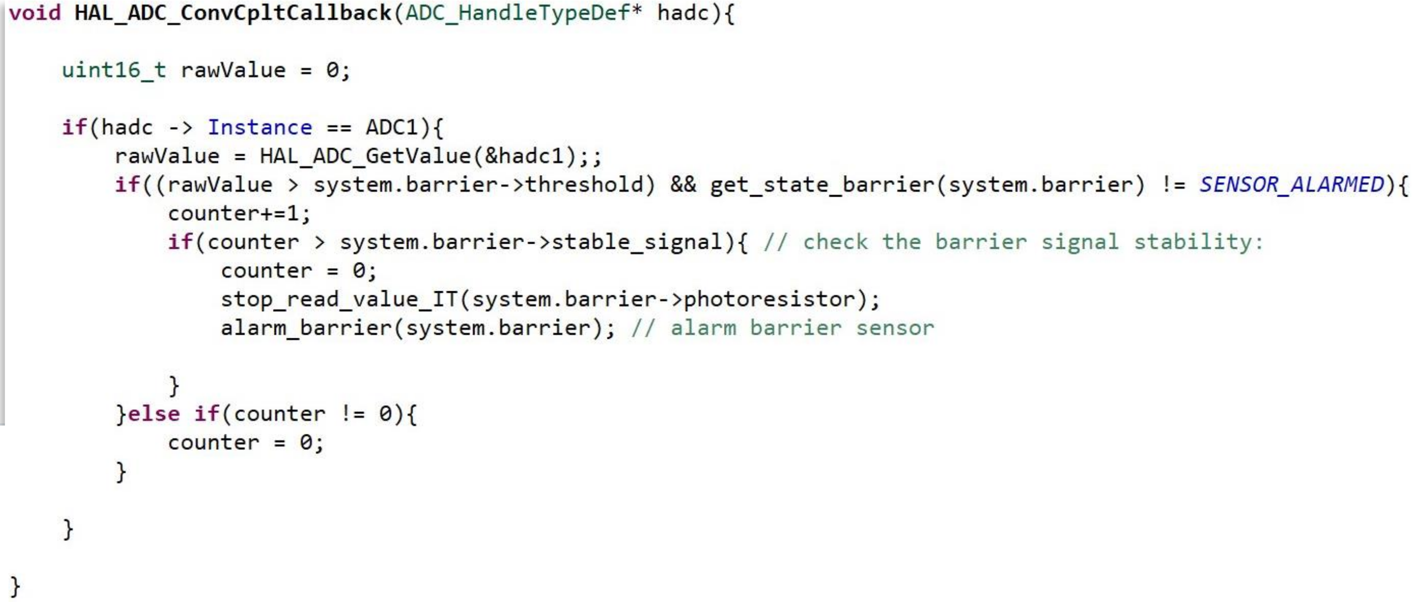
```
if(GPIO_Pin == PIR_SENSOR_PIN){  //system.pir->sensor.GPIO_Pin

    if((system.state == SYSTEM_ACTIVE || system.state == SYSTEM_ALARMED) && get_state_pir(system.pir) == SENSOR_ACTIVE){
        //check if system is alarmed or active and pir is active
            if(HAL_GPIO_ReadPin(PIR_SENSOR_PORT, PIR_SENSOR_PIN)){ // rising edge
                //pir_rising_edge();
                set_pir_pin_state(system.pir, GPIO_PIN_SET); // update pin state
                set_timer_period(&htim1, SIGNAL_STABILITY_S); // set period for checking signal stability
                reset_timer_counter(&htim1); // reset timer counter
                start_timer_IT(&htim1);
            }else if(!HAL_GPIO_ReadPin(PIR_SENSOR_PORT, PIR_SENSOR_PIN)
                    && get_state_pir(system.pir) != SENSOR_ALARMED){ // falling edge and timer update event not fired
                set_pir_pin_state(system.pir, GPIO_PIN_RESET); // update pin state
                stop_timer_IT(&htim1);
                reset_timer_counter(&htim1);
            }else
                set_pir_pin_state(system.pir, GPIO_PIN_RESET);

    }

}
```

Barrier sensor



```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){

    uint16_t rawValue = 0;

    if(hadc -> Instance == ADC1){
        rawValue = HAL_ADC_GetValue(&hadc1);;
        if((rawValue > system.barrier->threshold) && get_state_barrier(system.barrier) != SENSOR_ALARMED){
            counter+=1;
            if(counter > system.barrier->stable_signal){ // check the barrier signal stability:
                counter = 0;
                stop_read_value_IT(system.barrier->photoresistor);
                alarm_barrier(system.barrier); // alarm barrier sensor

            }
        }else if(counter != 0){
            counter = 0;
        }

    }

}
```

Barrier sensor is composed by two elements:

- Photoresistor;
- Laser module;

Its goal is to detect obstacles that pass between the laser and the photoresistor.

While the laser is simply configured as a digital output, to use the photoresistror, and detect brightness variations, we have configured an ADC.

ADC configuration:

- ADCClk = SystemClk/2
- ADC Resolution 12 bit
- Continuous Convertion Mode
- ADC Sample time 480 cycles
- Interrupt mode

We obtain a new conversion each:

$$t_c = Sampling\ time + Processing\ time = (480 + 15)\ ADC\ Clock\ Cycles$$

In order to have a check on the signal stability, the sensor must read $\#conversion\ for\ stability = \frac{t_{stability}}{t_c} = \frac{1\ s}{61\ \mu s} = 16393$ samples over the set threshold.

During the initialization the photoresistor reads (thanks to ADC) two different raw values for setting the threshold:

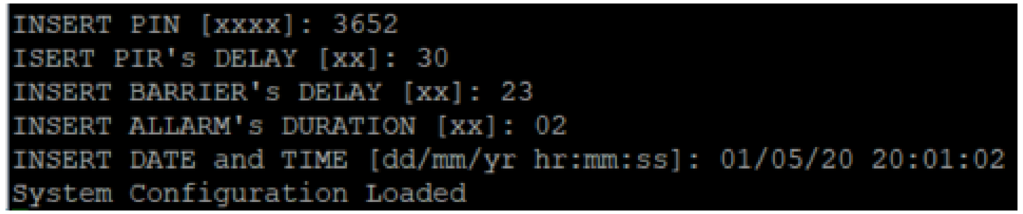
- Threshold_up , value with the laser off;
- Threshold_down, value with the laser on.

The final threshold is set as the mean of the two values above.
In the ADC Conversion Complete Callback we check, for each raw value read, if it is over or under the threshold.
If the module reads $\#conversion\ for\ stability$ samples over the threshold the module is alarmed.

Configuration Protocol

This module implements the system configuration procedure, that is run immediately after the system boot.
When the module is initialized, it takes in input the references to the following parameters:

- System_configuration, structure where store the configuration parameters;
- Timer_handler, timer that counts for the configuration protocol duration;
- RTC, used for update the current system date and time.

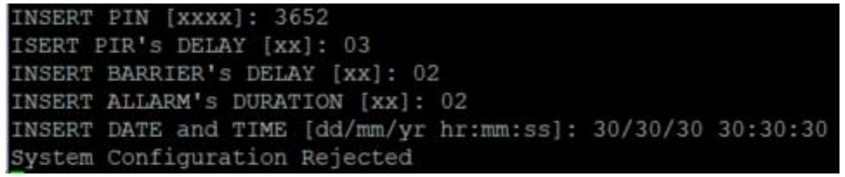


```
INSERT PIN [xxxx]: 3652
ISERT PIR's DELAY [xx]: 30
INSERT BARRIER's DELAY [xx]: 23
INSERT ALLARM's DURATION [xx]: 02
INSERT DATE and TIME [dd/mm/yr hr:mm:ss]: 01/05/20 20:01:02
System Configuration Loaded
```

Configuration Accepted

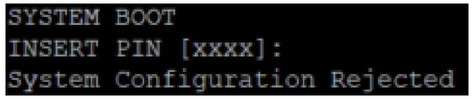
The protocol lasts 30 seconds, during which, the user can insert its own configuration.

If the configuration inserted by the user doesn't match the specifications or the protocol duration time is elapsed, the system starts up with the default configuration.



```
INSERT PIN [xxxx]: 3652
ISERT PIR's DELAY [xx]: 03
INSERT BARRIER's DELAY [xx]: 02
INSERT ALLARM's DURATION [xx]: 02
INSERT DATE and TIME [dd/mm/yr hr:mm:ss]: 30/30/30 30:30:30
System Configuration Rejected
```

Configuration Rejected



```
SYSTEM BOOT
INSERT PIN [xxxx]:
System Configuration Rejected
```

Time elapsed

The protocol sends a message if the custom configuration is accepted or rejected.
The protocol is performed over the UART interface.
Messages are sent by using the functions proposed by the system log module, in particular, the protocol uses the non-blocking functions, this means that the Tx Complete Callback and the Rx Complete Callback are used in order to handle the protocol steps.

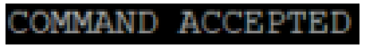
System Log

This module is in charge of manage the messages sent over UART interface.

When initialized, it takes in input the references to the following modules:

- RTC, used for compose the log message
- UART_handler, used for send messages over UART interface
- TIM_handler, used as reference to the timer that counts the log frequency

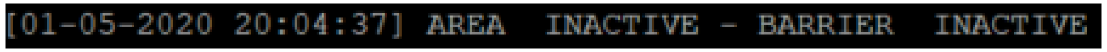
It implements functions that can be used for sending and receiving messages in both blocking and non-blocking mode, these functions are used during the configuration protocol (for receiving and trasmitting the configuration parameters), and for trasmitting the result for a given command.



```
COMMAND ACCEPTED
```

Moreover, it allows to start and stop the system log procedure.

This procedure is characterized by the fact that each 10 sencods a message, with the following format:



```
[01-05-2020 20:04:37] AREA  INACTIVE - BARRIER  INACTIVE
```

is sent over the UART.

The system log procedure starts when the timer, given to the system log module, fires its update event, that is handled by the Period Elapsed Callback

System

This is the core of all the software modules.

The system works in two different phases:

- **Phase 1**, Configuration and Initialization procedures.

During this phase, the system initializes all the support modules:

- Uart handler
- RTC with I2C protocol
- System log
- Configuration protocol

Then, it commands to the Configuration protocol module to run the protocol, in order to obtain the configuration parameters.
After the protocol, the system initializes all the sensor modules with the given configuration parameters.

- Module Pir
- Buzzer
- Laser
- Photoresistor

$\xrightarrow{}$ Module Barrier

- **Phase 2**, Running phase

During this phase, the system executes the commands received through the Keypad.

It receives the notification that a module is alarmed and based on its particular configuration it sets and starts TIM11 for the delay, if any, and for the duration, moreover it commands the buffer to emit the sound corresponding to the alarmed module.

The commands executed are:

- Activate System
- Deactivate System
- Activate module pir/barrier
- Deactivate module pir/barrier

The actions perfomed are:

- Alarm System
- Dealarm System
- Alarm module pir/barrier
- Dealarm module pir/barrier
- Activate Buzzer
- Deactivate Buzzer