



**Sant'Anna**

Scuola Universitaria Superiore Pisa

# **Motorbike Dashboard implementation for STM Discovery Board**

Project report

Davide Rasla, Andrea Bachechi

September, 2019

Prof. Marco Di Natale

Design of Embedded System

Scuola Superiore Sant'Anna

---

# Contents

---

|   |           |
|---|-----------|
| <b>Contents</b>                                 | <b>i</b>  |
| <b>1 Introduction</b>                           | <b>1</b>  |
| 1.1 What the system is supposed to do . . . . . | 1         |
| <b>2 User requirements</b>                      | <b>2</b>  |
| <b>3 Functional requirements</b>                | <b>3</b>  |
| <b>4 Safety requirements</b>                    | <b>5</b>  |
| <b>5 System Hardware Architecture</b>           | <b>6</b>  |
| 5.1 Global overview . . . . .                   | 6         |
| 5.2 Hardware components . . . . .               | 9         |
| 5.2.1 Potentiometer . . . . .                   | 9         |
| 5.2.2 Buttons and debouncing . . . . .          | 10        |
| 5.3 Connections and wiring . . . . .            | 12        |
| 5.3.1 Breadboard connections . . . . .          | 12        |
| 5.3.2 Dashboard Overview . . . . .              | 13        |
| 5.3.3 Fault management . . . . .                | 14        |
| <b>6 Software implementation</b>                | <b>15</b> |
| 6.1 RTOS and API . . . . .                      | 15        |
| 6.2 Software components . . . . .               | 16        |
| 6.2.1 Main module and tasks . . . . .           | 16        |
| <b>7 Testing</b>                                | <b>18</b> |
| 7.1 Conformance Tests . . . . .                 | 18        |
| 7.2 Tests coverage . . . . .                    | 18        |

## Chapter 1

---

# Introduction

---

In this section a system description has been provided from the user perspective, while in next sections this description will be analyzed and rewritten in a more precise and rigorous definition.

### **1.1 What the system is supposed to do**

The purpose of the system is to simulate the dashboard of a generic motorbike. The system must be able to detect the input from the user, like throttle, clutch, signal lights and gears and display on the screen the state of the simulated vehicle.

## Chapter 2

# User requirements

The system is a simulation of a dashboard of a motorbike.

Here the user requirements:

1. The system must be developed in order to simulate the dashboard of a motorbike (equipped with manual gearbox);
2. The system shall be able to display: motor's RPM, actual gear, speed, fuel level, signal lights, clock, kilometers traveled, partial kilometers, oil warning light, fuel warning light;
3. The system shall be able to simulate erroneous situations like low fuel and low oil;
4. The system shall be able to simulate mismanagement of the engine such as engine melts due to the lack of oil;
5. The system shall be able to change gear and evolve in time in a realistic way. i.e: when in neutral gear the motorbike must not move, but obviously the engine must respond to the accelerator;
6. All the lights shall be coherent with the situation of the dashboard;
7. The RPM of the engine shall be coherent with the actual gear and throttle situation.
8. The user shall be able manage gears;
9. The user shall be able to turn on signal lights independently;
10. The user shall be able to reset partial kilometers;
11. The user shall be able to control Throttle and consequently the RPM value.
12. The user shall be able to control the I/O following the requirements described in 5.2.2

| Parameter Name       | Text Description  | Data Type    | Unit                   | Default  | Min  | Max     |
|----------------------|---|--------------|------------------------|----------|------|---------|
| ACTUAL SPEED         | Instant Speed of the motorbike  | unsigned int | km/h                   | 0        | 0    | 250     |
| CONSUMPTION LITRE    | Amount of km with one litre of fuel   | unsigned int | km/litre               | Costant  | 1    | Costant |
| ENGINE ON            | Status of the engine. 0 if everything is working fine, otherwise 1.                                       | boolean      | unsigned int           | 0        | 0    | 1       |
| Oil Must Be Refilled | km before being without oil, leading to a break   | unsigned int | km                     | variable | 0    | 1       |
| ADC.Clutch           | Actual value of the clutch. ON if greater than 2000, OFF otherwise  | float        | Integer                | 0        | 0    | 4000    |
| ADC.Throttle         | Actual value of the throttle. Positive acceleration if greater than 2000, negative acceleration otherwise | float        | Integer                | 0        | 0    | 4000    |
| RPM                  | Actual RPM of the engine  | unsigned int | revolutions per minute | variable | 1000 | 9000    |

Table 2.1: Data dictionary

## Chapter 3

---

# Functional requirements

---

In order to achieve the user requirements, these functional requirements are needed:

1. 8 sensors, 2 potentiometer and 6 buttons shall be connected to the board through a circuit that ends on the pins showed in 3.1
  - 1.1. 2 potentiometer shall be used to simulate the throttle and the clutch;
  - 1.2. 6 buttons shall be used to detect all the conventional events in a dashboard. In detail:
    - 1.2.1. one button to shift up the gear;
    - 1.2.2. one button to shift down the gear;
    - 1.2.3. one button to turn on and off the left arrow;
    - 1.2.4. one button to turn on and off the right arrow;
    - 1.2.5. one button to resetting the partial kilometers;
    - 1.2.6. one button to turn on and off the headlights.
2. For each sensor, the system shall show a relative icon and change the current state. More precisely:
  - 2.1. Using the throttle potentiometer shall be possible to simulate the engine increase of RPM and to update, consistently with the clutch and the gear status, the speed of the motorbike. For example: if the actual gear is 'Neutral' the speed must not change. Nonetheless the engine is still responsive to the throttle and the RPM must change;
  - 2.2. Using the clutch potentiometer shall be possible to simulate the gearbox of the engine and change gear according to the button pressed. For example: When the clutch is ON, the speed must be reduced slowly, simulating a deceleration.
  - 2.3. Using the buttons shall be possible to update the status of the arrows and the gear icon.
  - 2.4. The reset button shall allow the user to reset the partial amount of kilometers.

---

| Name                 | Text Description  | Direction | DataType | Port          | Pin |
|----------------------|---|-----------|----------|---------------|-----|
| BTN_GEAR_UP_PIN      | Button used to shift up the gear                        | IN        | boolean  | GPIOE         | 0   |
| BTN_GEAR_DOWN_PIN    | Button used to shift down the gear                      | IN        | boolean  | GPIOE         | 5   |
| BTN_TURNLIGHT_DX_PIN | Button used to turn ON and OFF the right arrow          | IN        | boolean  | GPIOE         | 4   |
| BTN_TURNLIGHT_SX_PIN | Button used to turn ON and OFF the left arrow           | IN        | boolean  | GPIOE         | 6   |
| BTN_RESET_PARTIALKM  | Button used to reset the partial amount of km           | IN        | boolean  | GPIOC         | 3   |
| BTN_Brights_DOWN_PIN | Button used to turn ON and OFF the brights              | IN        | boolean  | GPIOE         | 1   |
| ADC_CLUTCH_DEV       | Potentiometer used to simulate the clutch on ADC2       | IN        | float    | ADC_Channel_5 | 5   |
| ADC_THROTTLE_DEV     | Potentiometer used to simulate the throttle on the ADC1 | IN        | float    | ADC_Channel_4 | 4   |

Table 3.1: Functional Requirements for the I/O

## Chapter 4

---

# Safety requirements

---

The system shall be able to manage failure. The system goes into an error state, using the global variable 'StatusEngine'. Simulating an engine failure due, for example, to a lack of oil as described in 4.3.3.

If the user handles incorrectly the system, for example if the fuel goes to zero, the same variable 'StatusEngine' blocks the entire system, simulating a lack of fuel.

## Chapter 5

---

# System Hardware Architecture

---

In this chapter system's architecture is described from a global point of view, then system's hardware and software organization is shown more in detail.

### 5.1 Global overview

System global overview is shown in Figure 5.1, using SysML diagrams. *Block Definition Diagram* highlights separation between software and hardware components.



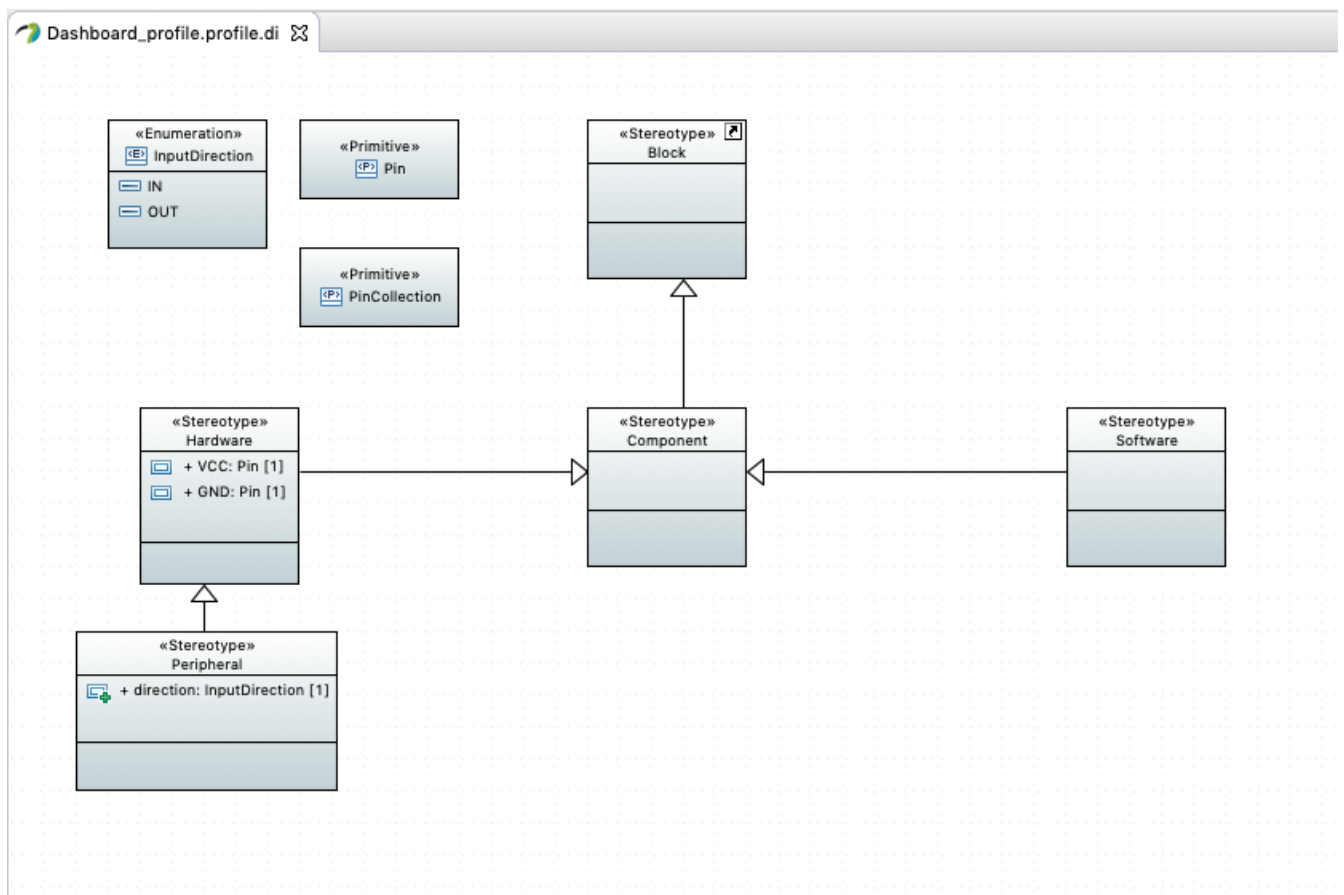


Figure 5.1: System Block Definition Diagram

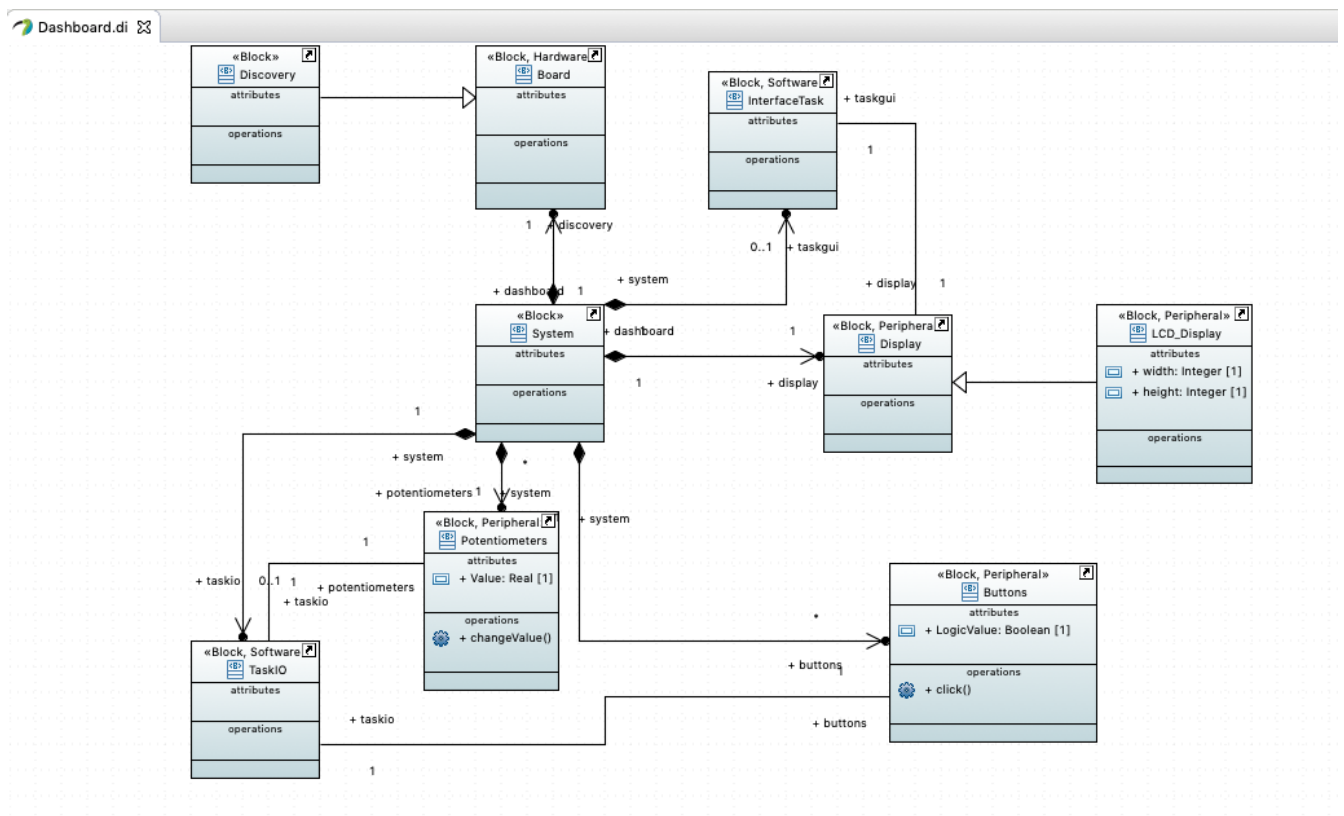


Figure 5.2: System architecture shown using SysML diagrams.

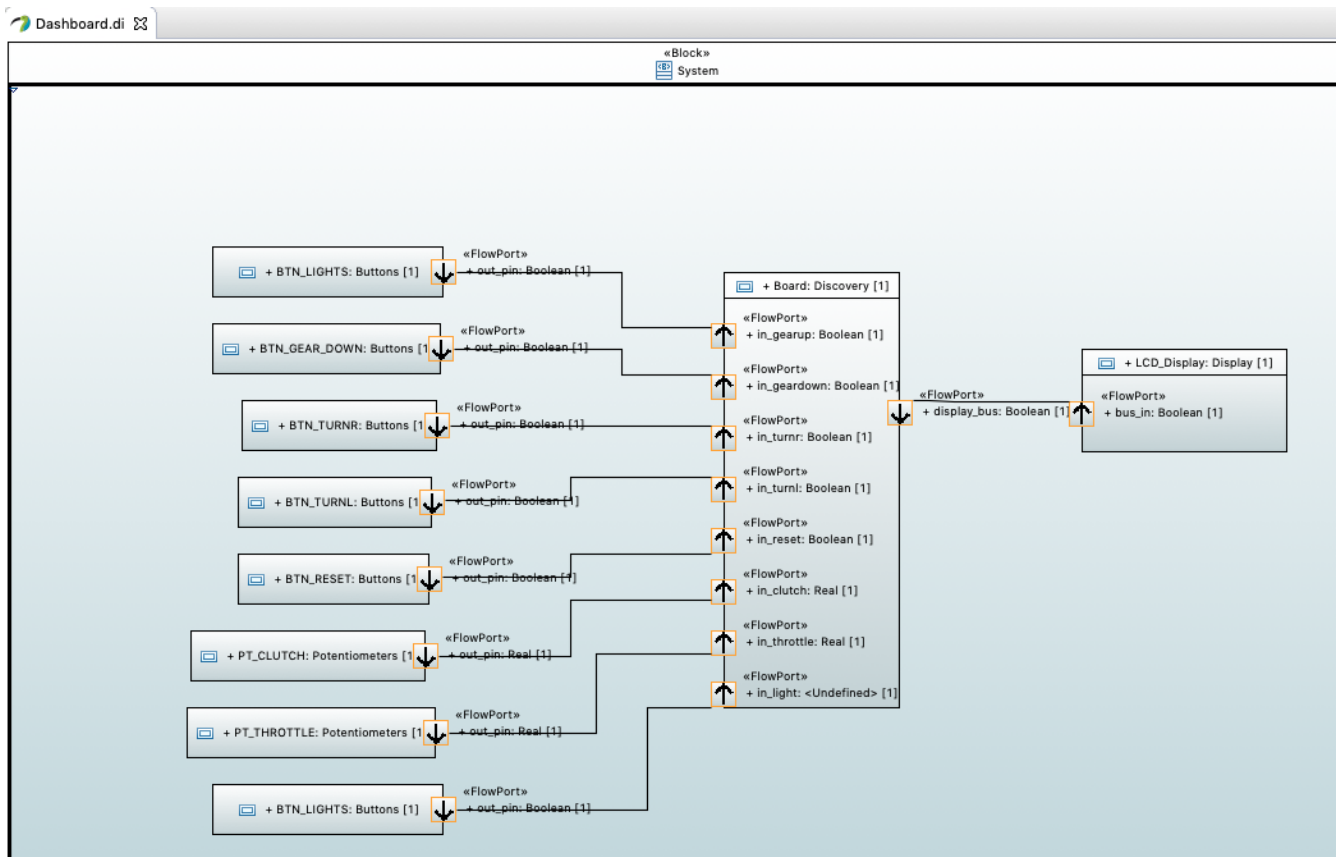


Figure 5.3: Internal architecture shown using SysML diagrams.

## 5.2 Hardware components

Hardware components choice was driven by their ease of connection with the system. Everything is connected through a circuit on a breadboard to the stm32f4.

### 5.2.1 Potentiometer

In order to simulate the throttle and the clutch two different potentiometers has been used. The range of values returned by each potentiometer is [0-4000] and has been divided into half. The first one [0-2000] has been interpreted as negative value (the clutch is OFF), instead the second one [2001-4000] has been interpreted as a positive value (the clutch is ON). In the case of the throttle, a value over 2000 is considered as a positive acceleration, otherwise is considered as deceleration. The intensity of the acceleration or deceleration depends by the readed value; the higher the value, the higher the intensity.



Figure 5.4: Potentiometer used to simulate the throttle and the clutch

### 5.2.2 Buttons and debouncing

To simulate all the events discussed in chapter 4, buttons has been used. Despite their simplicity, in order to avoid glitching and not desired input command two important constraints have been implemented, in order to implement a debouncing filter, as shown in figure 4.4:

1. User must always press and release the button in order to give a single command. As consequence to give two commands a button must be pressed twice. This constraint is important because a button has not positions, like a switch, so if the user keeps the button pressed, an unbounded number of input are sent. Imposing that the user must release each time the button avoids this undesired behaviour;
2. A button must be pressed at least for 30ms. This is important in order to avoid the glitching effect, due to a bad circuit signal.

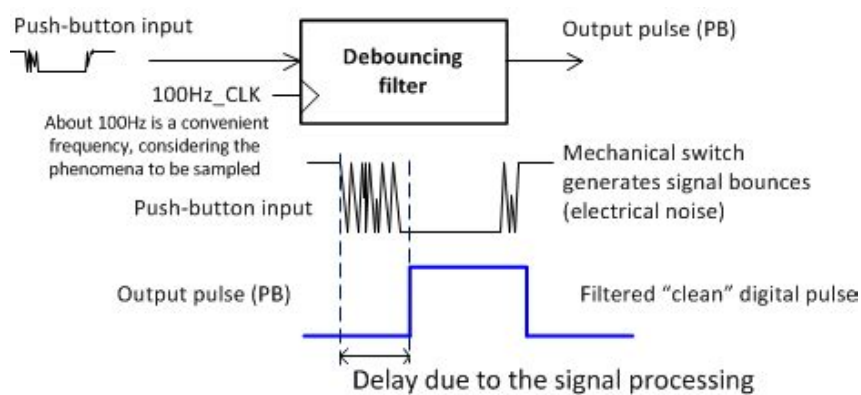
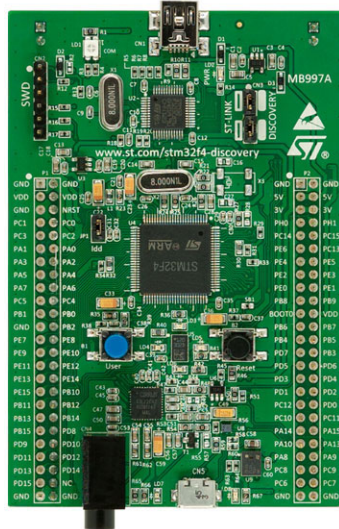


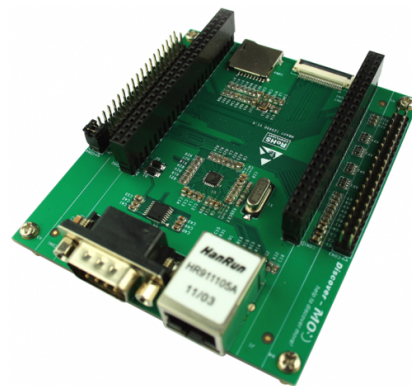
Figure 5.5: Physical explanation of debouncing

The debouncing filter is provided by the task `Task_ReadSensors` described in 6.2.1.

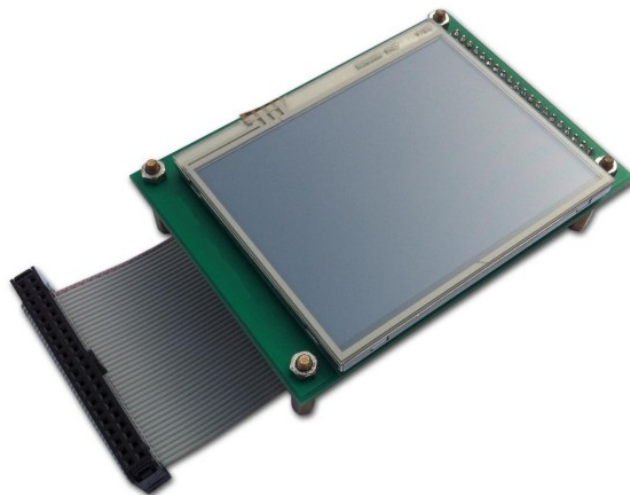
To realize the system the STM32F4 Discovery board, which runs both the applicative code using ERIKA OS as real-time operating system has been used. The Discovery Board is shown in Figure 5.6a. Power can be provided to the board using a standard USB Mini Type-B connector, connected to a 5 V power supply<sup>1</sup>; board includes also two buttons, respectively a reset button and a user button.



(a) STM32F4 Discovery board



(b) Expansion board for STM32F4 Discovery board



(c) LCD Screen included with Expansion board kit

Figure 5.6: STM32F4 Discovery kit

<sup>1</sup>Power supply must not supply more than 100 mA.

Since the system requires a screen on which to show the dashboard, an expansion kit is needed to connect the Discovery board to his LCD screen (STM32F4DIS-LCD). Expansion board and its LCD screen are both shown in Figure 5.6.

### 5.3 Connections and wiring

External hardware components have been connected to the board using a circuit on a bread-board, as shown in figure 5.3. The figure is also in attached in a .pdf file.

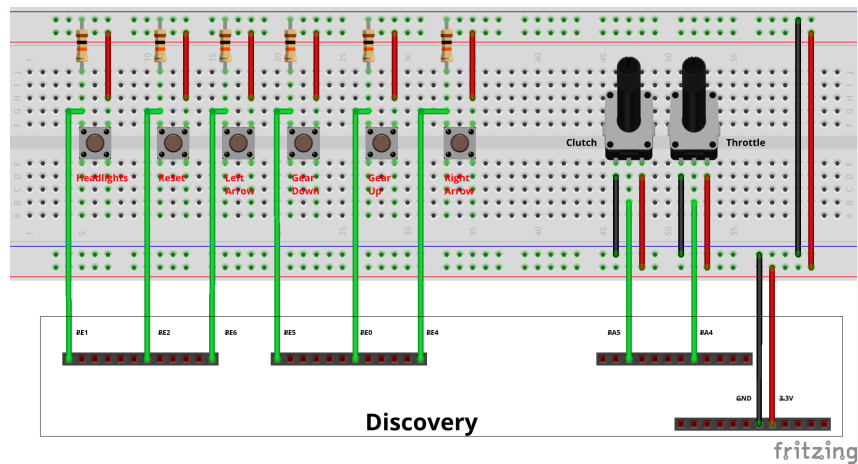


Figure 5.7: The circuit on the BreadBoard

#### 5.3.1 Breadboard connections

All the input described in the table 3.1 are realized through the circuit shown in 5.8

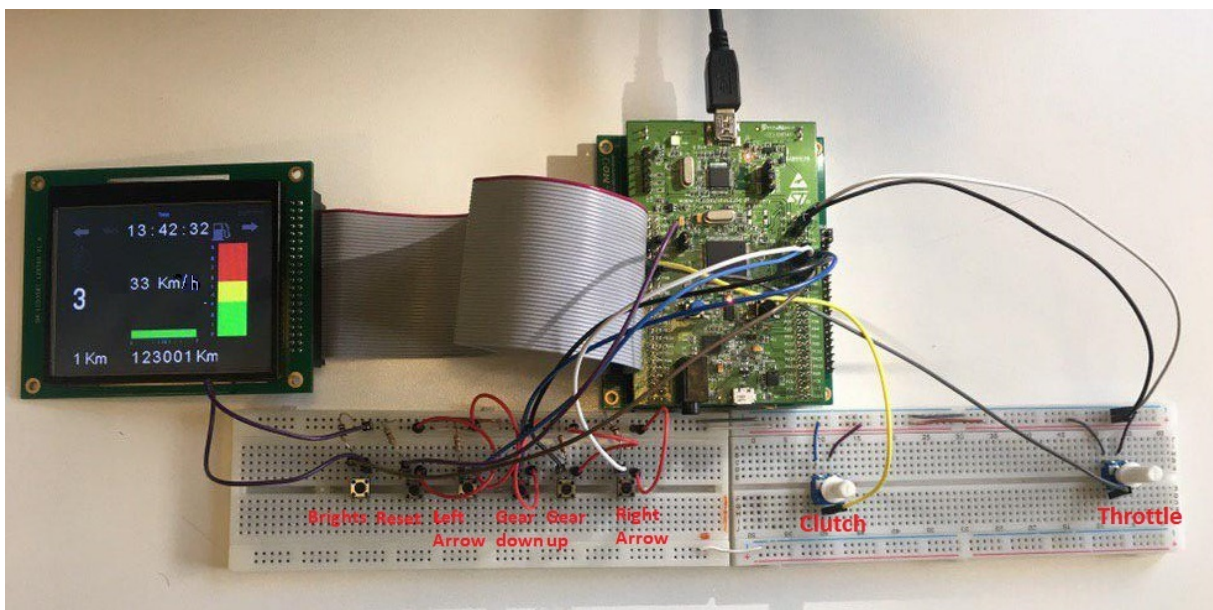


Figure 5.8: The entire system



### 5.3.2 Dashboard Overview

in figure 5.9 is shown an overview of the entire system where there is an example of it running.

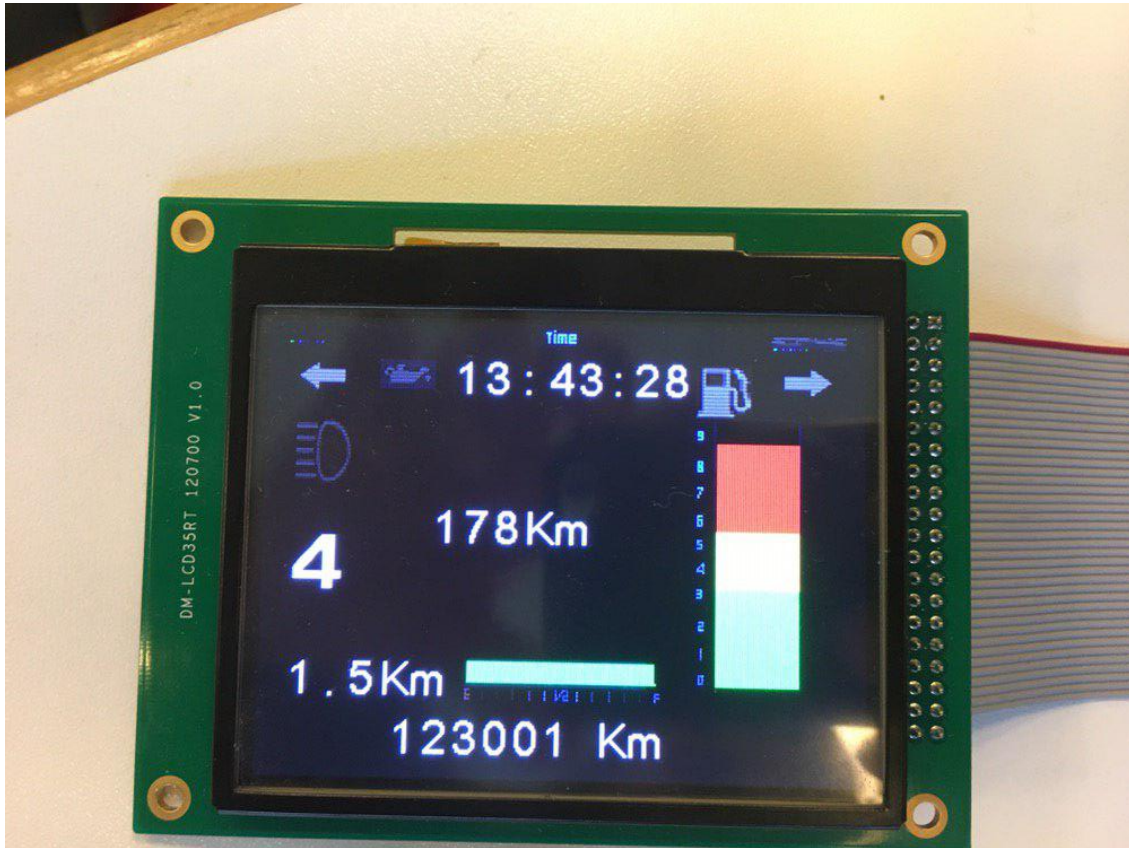


Figure 5.9: Dashboard Overview

in this case;

1. The RPM of the engine are at their maximum possibility;
2. The actual speed is 178 km/h;
3. The oil light is OFF, meaning that there is a enough level of the oil itself.
4. The fuel bar is still green, meaning that it is full;
5. The arrows are OFF;
6. The fuel light is OFF, because the petrol level is still high;
7. The brights are OFF;
8. The partial amount of km is 1.5 km;
9. The total amount of km is 123001;
10. The actual gear is the fourth.

### 5.3.3 Fault management

Situations of errors, like in the case that the fuel bar or the oil level goes to zero, are managed. Indeed, the system continuously checks and updates the current status of the dashboard, and when the levels are critical, turns ON the relative lights.

In addition to turning on the lights, if there is no reset, after a fixed number of additional km (that can be changed in the file config.h) the entire system stops, simulating a motorcycle break.

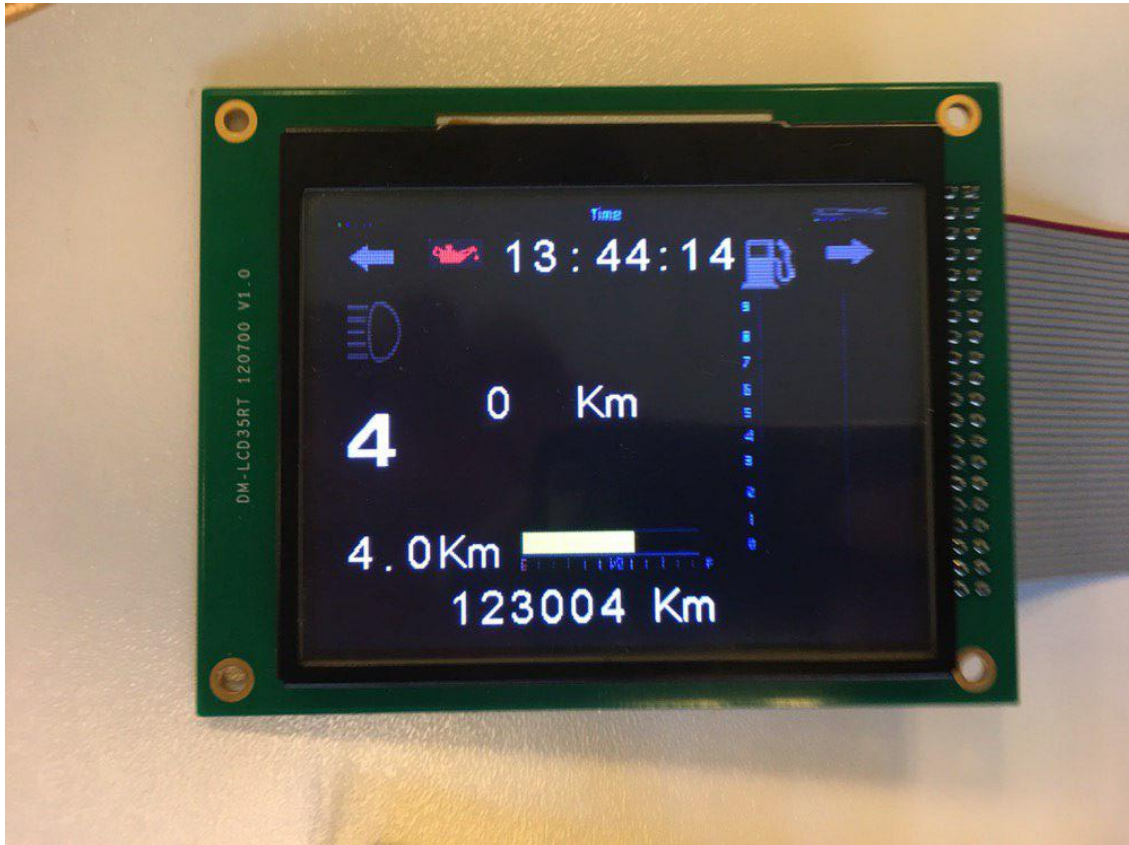


Figure 5.10: Simulation of a break

In figure 6.1 there is an example, where after 2 km the oil light goes on and after 4 km the entire system stops. In this state the actual speed and the RPM of the engine are zero and the system isn't responsive anymore to the sensors, therefore the throttle and the clutch are not useful anymore and the only way to restart the system is thought the reset.



## Chapter 6

---

# Software implementation

---

In this chapter is shown briefly software organization, highlighting major features provided by each of its components.

### 6.1 RTOS and API

Since this system has to deal with real-time requirements, a real-time operating system (RTOS) is needed to ensure the fulfillment of these requirements. To this, Erika Enterprise RTOS is used, which provides support to fixed and dynamic priority scheduling, priority ceiling and so on.

System requirements could be accomplished by simply including only fixed priority scheduling support, hence we used the smallest-footprint version available to implement system's controller, including only needed ST libraries. In addition to ST provided libraries, we included some other higher-level libraries, to manage respectively servo motor and input/output pins.

In order to meet all the deadlines and make the system work correctly, an RTOS has mounted on the STM32F4 board. In particular it is used Erika Enterprise that is an open-source OS-EK/VDX Hard Real Time Operating System. Erika OS provide an hard real-time support with fixed priority scheduling and immediate priority ceiling. Furthermore provide also support for earliest deadline first (EDF) and resource reservation schedulers.

## 6.2 Software components

In this section the software organization is shown.

### 6.2.1 Main module and tasks

Main module is in charge of coordinating all other modules. It handles *SysTick* timer interrupts and manages the four concurrent real-time tasks that compose the system:

1. Task CheckEvents: checks whether an event occurs and updates the system status in a proper way;
2. Task GUI FAST: Updates the widget on the GUI that need to be changed frequently, like the actual speed and the RPM of the engine;
3. Task GUI SLOW: Updates the widget on the GUI that don't need to be changed frequently, like the fuel bar, the km amount and the gear;
4. Task GUI ReadSensors: Reads all the I/O and sets global variables in order to set an event, managed by the task CheckEvents. The whole evolution of the system depends on this task, as shown in figure 6.1.

In figure 6.1 are shown the parameters of each task.

| Task Name        | Description  | Period (ms) | Priority |
|------------------|--|-------------|----------|
| Task GUI SLOW    | Updates the lights, the clock and all the widget that don't need to be update frequently | 1000        | 0x01     |
| Task CheckEvents | When a event is set, moves the system in another state                                   | 50          | 0x02     |
| Task GUI FAST    | Updates the speed and the RPM  | 200         | 0x03     |
| Task ReadSensors | Reads from the I/O all the inputs  | 40          | 0x04     |

Table 6.1: Task Parameters

In figure 6.2 is described the behaviour of the tasks.

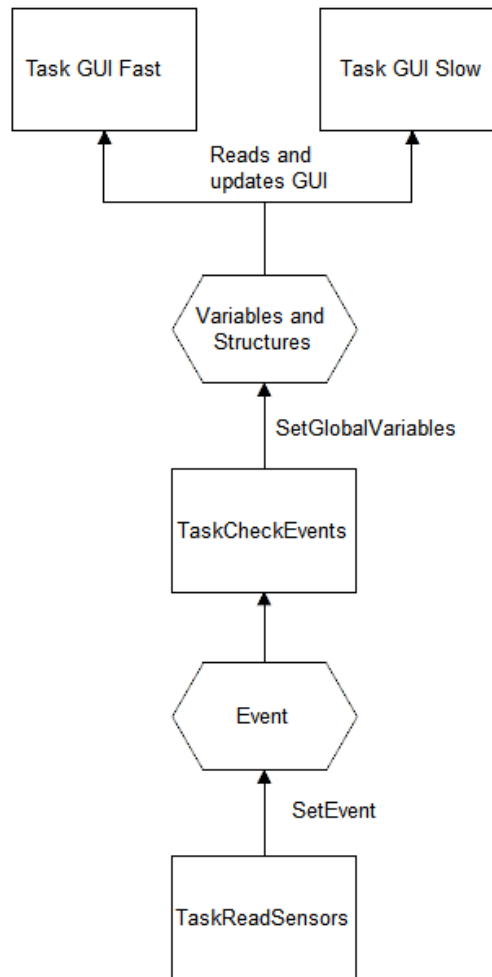


Figure 6.1: Behaviour of the tasks

From a general overview, the task with the higher priority is the task that must read all the I/O environment in order to generate the events and the signals that can provide a transition in another state of the system. It is worth considering that two tasks to manage the graphic are necessary in order to avoid useless refreshing on the screen and to not overload the microprocessor on the stm4 board.

## Chapter 7

---

# Testing

---

The tests that have been performed on the system are described in this chapter.

Through all phases of system development, several tests have been performed, either on hardware or software components.

Apart from these tests, individual software components have also been tested using *CUnit* suite, performing functional and conformance testing.

### 7.1 Conformance Tests

This section illustrates tests performed by the suite, that performs conformance testing of the module `io.c`, where the function that manage the throttle and the clutch are implemented.

### 7.2 Tests coverage

In order to prove actual goodness of these test suites coverage of testing code has been completed. To do so, *GCov* suite is used in combination with *CUnit* to obtain code lines execution statistics.

Thanks to these tools, has been proved that our tests cover 100% of tested code, hence proving their goodness. This result is shown in Table 7.1.



| Filename | Lines Coverage  |        |         | Functions |     |
|----------|---|--------|---------|-----------|-----|
| main.c   |  | 100.0% | 23 / 23 | 100.0%    | 3/3 |
| io.c     |  | 100.0% | 39/39   | 100.0%    | 6/6 |

Table 7.1: Summary of the coverage