20/01/2019 # Report Extraproject02 Magnani Simone 265171

This report is divided in four sections, each one concerning the main functionalities and some general considerations about the delivery. Hopefully, I will exploit all the important aspects of my project, which serves one of the many possible solution to this assignment.

General View

From my point of view, starting from the previous code (extrapoint01) it was possible to extend it in 2 different ways: by using one more timer for the sound interrupt or by implementing this feature in one of the already used timers, moving its old task in the RIT handler and using polling as the method to understand when to do those actions.

Since I prefer not to use polling, I activated one more timer to implement the sound specification. Moreover, I even used the RIT to manage not only the button bouncing, but also the joystick and the potentiometer.

Interestingly, I created a header file named “config.h” where all the values used in the project are defined. This was extremely useful and makes the project easier to be managed, especially when you debug it using the Hardware instead of the software emulation (time increases differently).

Differently from the previous project, the main logic of the program is, in addition to the timers, in the RIT handler and no more in the button’s ones. This was forced by the implementation of the de-bouncing, thanks to we can manage multiple accidental pressures of the buttons. In addition, even the timer1 needed to be slightly modified to manage the sound by enabling or not the timer2.

Timers logic

Timer0 implements the semaphore state logic. It is responsible of changing the state according to the given times (15” and 5”) and correctly set the actions to be activated or deactivated in the following state. In fact, its handler is the more complex compared to the ones of the other two timers, which are mainly managed not only by timer0, but also by the buttons (and as a result, by the RIT). The only state which this timer ignores is the “Maintenance”, because according to the specification that state must be only managed by the joystick. Apart from some improvements, timer0 is compliant to its implementation in the previous project.

As previously announced, timer1 slightly changes, gaining some more tasks. In fact, it not only manages the leds blinking in both the two state which require it, but it also enables and disables the timer2 to turn on/off the sound when required. Fortunately, I was able to implement both these functionalities in the same timer because the specifications allow that, considering the equal time for blinking and sound emission. As expected, this timer is turned on not only by the timer0 interrupts, but also by the pressure of the buttons if required. We need to consider the “blind button” pressure event even in the state where the pedestrian semaphore is already green or flashing. As a result, this project can manage all the possible situation, considering even the most improbable ones which may not happen in real life situations.

Finally, the timer2 has only to reproduce the sound according to the current value of the potentiometer. In fact, I decided not only to divide the sinusoid by two, but I also multiplied each value by a proportion of the current value of the potentiometer over the maximum value. In this way it is possible to turn up/down the volume in the “Maintenance” state.

RIT logic

The RIT interrupt is the most vital in this project, since it not only manage the buttons, but also the potentiometer and the joystick. In fact, its body it is the most consistent one, checking all these possible peripherals events.

Firstly, each button is treated equally, except for the one dedicated to the blind persons, which in addition turns on the loudspeaker until it is released. Then, if the current state allows it, it is activated timer1 to start making sound (if green or flashing green for pedestrian). All the other actions are the same of the previous project, apart from some variable initialization like the one used to save the request of a blind person if the semaphore is green or yellow for the car.

Secondly, the joystick pressures are handled, checking if you are in the correct state to enter/exit the “Maintenance” state. If not allowed, these events are discarded. Otherwise, it is set not only the timer1, but also the lights to make the semaphore blink correctly. This part considers also the case when a blind person presses the button, but immediately after someone use the joystick to enter the maintenance: in this case, it records the event “blind person pressed” in order to re-active the sound once the system exits from this state and turns back into the green pedestrian semaphore one.

Finally, it is controlled the current state of the system to understand if the potentiometer needs to be enabled or not. Since this peripheral is used only in the maintenance state, I decided not to call the “ADC_start_conversion()” function every time, to prevent that the current value is overridden when it would not be possible.

Final Consideration

Of course, implementing all these types of control and interrupts was harder than using the polling strategy, but I am really satisfied of my project because as explained many lectures ago this result is more power and time efficient than the other solution. Moreover, I tried to implement something scalable and reusable the most, to cover other possible user cases (for example if you use the project on a different frequency, you just must change the defines since there are no “magic numbers” in the code, except for some variable which needs to emulate the Boolean variable behaviour).