

Lift control using an ARM-based microcontroller

Alejandro López Rodríguez
Ana Maria Casanova López

Universidad Politécnica de Valencia
ETSID



June 7, 2021

Table of Contents

- 1 Goals and context
- 2 Implementation
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information



Table of Contents

- 1 Goals and context
- 2 Implementation
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information



Project description

- Development of an embedded application designed to control an industrial lift using the ARM-based, STM32F407 microcontroller.

Goal of the project

- Optimised application
- Proper clock configuration
- Pertinent input / output configuration
- Use and adequate configuration of external hardware



Table of Contents

- 1 Goals and context
- 2 Implementation**
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information



Implementation I

After the initialization, the system is ready to start. In order to move the lift to another floor, the button (blue button of STM32F4Discovery board) must be pressed. Once pressed, the servos close the door (Initially open), the stepper motor starts spinning, and the orange and red LEDs flash according to the direction of the movement.

Once 5 seconds have passed, the lift is assumed to have arrived to the corresponding floor. Then the stepper motor, and the LEDs flashing stops, and the servos open the doors of the lift. The blue and the green LEDs indicate the current floor. The different steps of the operation are sent via UART.

The variables used in the flowchart do not correspond with the ones used in the program so as to reduce the size of the diagram.



Implementation II

In the first part of the *while(1)* loop, the STM32 checks whether the **ButtonPressFlag** is set (which is updated inside the EXTI interrupt, as we will see later). If it is, first the flag is set to false, and depending on the current floor, either *liftUp()* or *liftDown()* is called. Inside these functions, the doors are closed, the direction of the movement is set, the LEDs blinking is initialized, and the timer 3 is started.

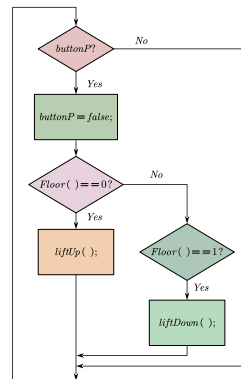


Figure 1: First part infinite loop flowchart



Implementation III

Next, the loop checks if the ***timer5endFlag*** is set. As the name states, this flag is set when the 5 seconds timer generates an interrupt (TIM3, as we will see later). If it is set, it calls the function *liftStop()*, which re-enables the EXTI0 and the TIM3 interrupts, turns off the orange and red LEDs, updates the current floor value depending on the direction of the lift's movement, and, finally, opens the doors. As we mentioned before, the system communicates the different steps via the UART.

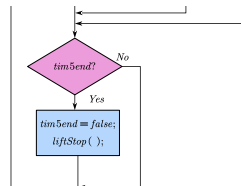


Figure 2: Second part infinite loop flowchart



Implementation IV

Regarding the interrupts, in order to reduce the time that the CPU stays inside of them, when the ISR is called, only the aforementioned flags are set. Since the structure of this project in particular is sequential, the functions that we have implemented could have been called in the callback of the ISRs themselves, but we decided not to do this because it is not good practice.

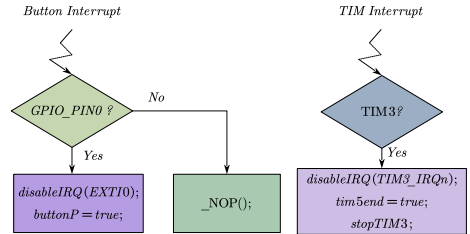


Figure 3: Interrupts flowchart



Implementation V

This is how we have organized the set of functions shown in the subfiles. We have developed more functions but these are the most relevant ones used in *lift.c*.

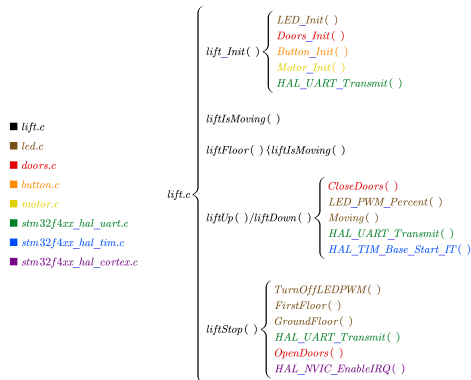


Figure 4: File dependency tree



Table of Contents

- 1 Goals and context
- 2 Implementation
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information



Clocks

The internal clocks were configured as stated in the lab sessions:

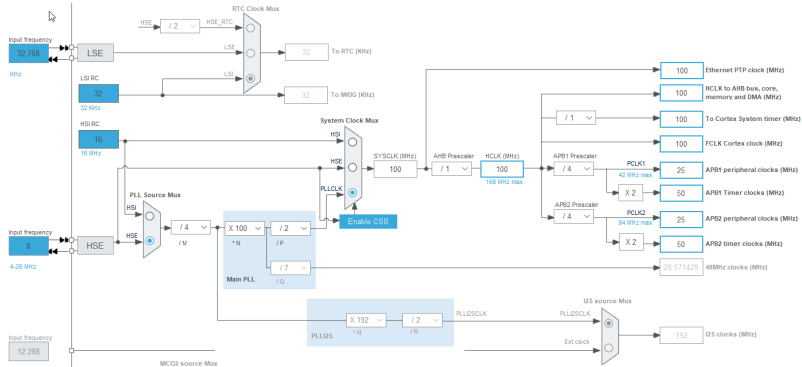


Figure 5: Internal clocks



- Blue Pushbutton
 - **PA0**: GPIO_EXTI0, no pull-up no pull-down.
- UART Communication
 - **PA2, PA3**: USART2_TX, USART2_RX.
- Stepper Motor
 - **PB12, PB13, PB14, PB15**: GPIO_Output, Push-Pull, no internal resistor, output speed low.
- Servomotor
 - **PE9**: TIM1_CH1.
 - **PE11**: TIM1_CH2.
- Floor Number LEDs
 - **PD12 (Green Led), PD15 (Blue Led)**: GPIO_Output, Push-Pull, no internal resistor, output speed low.
- State LEDs
 - **PD13 (Orange Led)**: TIM4_CH2.
 - **PD14 (Red Led)**: TIM4_CH3.



In order to fulfill the project requirements, this is the final configuration of CubeMX:

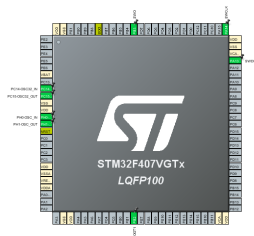
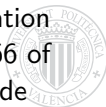


Figure 6: STM32F407 final pin configuration in CubeMX

The only auto-generated file that remains in the final project is the GPIO configuration file, *GPIO.c*, as it cannot be removed. It is important to remark that lines 65 and 66 of the file *stm32f4xx_hal_conf.h* must be uncommented after the generation of the code in CubeMX in order for the timers and UART to work.



Timers

$$\text{TIM1 (T = 20 ms)} \rightarrow \left\{ \begin{array}{l} \text{TIM1 CLK} = \frac{50 \text{ MHz}}{\text{Presc}} = \frac{50 \text{ MHz}}{500} = 100 \text{ kHz} \\ \text{Counter period} = ((\text{TIM1 CLK}) \cdot T - 1) = 1999 \end{array} \right.$$

$$\text{TIM3 (T = 5 s)} \rightarrow \left\{ \begin{array}{l} \text{TIM3 CLK} = \frac{50 \text{ MHz}}{\text{Presc}} = \frac{50 \text{ MHz}}{50000} = 1 \text{ kHz} \\ \text{Counter period} = ((\text{TIM3 CLK}) \cdot T - 1) = 4999 \end{array} \right.$$

$$\text{TIM4 (T = 1 s)} \rightarrow \left\{ \begin{array}{l} \text{TIM4 CLK} = \frac{50 \text{ MHz}}{\text{Presc}} = \frac{50 \text{ MHz}}{5000} = 10 \text{ kHz} \\ \text{Counter period} = ((\text{TIM4 CLK}) \cdot T - 1) = 999 \end{array} \right.$$



The USART pins are the ones presented in previous slides. The configuration is performed in order to have an asynchronous communication (no external clock needed), with a baud rate (bits/sec) of 9600, which is a common value for these type of communications.



Table of Contents

- 1 Goals and context
- 2 Implementation
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware**
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information



LEDs

The green and the blue LEDs are controlled by a set of HAL functions. In order to control the orange and red LEDs, the function **LED_PWM_Percent()** was implemented.

```
1 void LED_PWM_Percent(uint16_t LED, uint8_t value){
2     uint32_t Pulse = 0;
3     /* Calculate the Pulse value given a duty cycle percentage */
4     if(value > 100) Pulse = 9999;
5     else if(value == 0) Pulse = 0;
6     else Pulse = (uint32_t)(value * 100 - 1);
7     /* Update the PWM duty cycle by modifying the CCRx value */
8     if(LED == LD3_Pin) /* Orange LED */
9         TIM4->CCR2 = Pulse;
10    else if(LED == LD5_Pin) /* Red LED */
11        TIM4->CCR3 = Pulse;
12 }
```



Servo motors

Two servo motors are connected to PWM pins PE9 and PE11. Their duty cycle is calculated as a function of the required angle in the function ***Servo_PWM_Angle()***. The pulse duration was obtained from the datasheet [1] provided by the manufacturer.

```
1 void Servo_PWM_Angle(uint16_t Servo, uint8_t angle){
2     uint32_t Pulse = 0;
3     /* Calculate the Pulse value given a duty cycle percentage */
4     if(angle > 180) Pulse = 224;
5     else if(angle == 0) Pulse = 74;
6     else Pulse = (uint32_t)(angle * (5.0/6.0) + 74);
7     /* Update the PWM duty cycle by modifying the CCRx value */
8     if(Servo == Servo1_Pin) /* Orange LED */
9         TIM1->CCR1 = Pulse;
10    else if(Servo == Servo2_Pin) /* Red LED */
11        TIM1->CCR2 = Pulse;
12 }
```



Stepper motors

The stepper motor is controlled using a LUT in half-step mode (8 possible steps). We have taken advantage of the SysTick interrupt (every ms) to increment/decrement the LUT index depending on the destination floor, and to write its contents to the 4 pins that control the stepper motor driver.

```
1 void SysTick_Handler(void){
2     HAL_IncTick();
3
4     if(liftIsMoving() && DirUP)
5         (step_index == 7)? step_index = 0 : step_index++;
6     else if(liftIsMoving() && DirDOWN)
7         (step_index == 0)? step_index = 7 : step_index--;
8
9     GPIOB->ODR &= ~(0xF << 12);
10    GPIOB->ODR |= step_positions[step_index] << 12;
11 }
```



UART communication

Since the UART configuration was previously defined, in order to send UART messages from the microcontroller, we have used the existing HAL function ***HAL_UART_Transmit()***.

```
1 unsigned char mssg[]="Initialization complete\n";  
2 HAL_UART_Transmit(&huart2,mssg,sizeof(mssg)/sizeof(mssg[0]),0xffff);
```



Table of Contents

- 1 Goals and context
- 2 Implementation
- 3 Clock and peripherals configuration
 - Clocks
 - GPIO
 - Timers
 - USART
- 4 Hardware
 - LEDs
 - Servo motors
 - Stepper motor
 - UART communication
- 5 Extra information




This project is available on [Github](#), as well as the necessary documents to reproduce the operation of the system. In addition, other documents such as this presentation are also accessible. We encourage you to read the project documentation we created with the help of *Doxygen*, on this link you can find the explanation of this presentation and a video showcasing the developed application in real time.



Thank you!



-  *Parallax Standard Servo*, 900-00005, Rev. 2.2, Parallax INC., Oct. 2011. [Online]. Available:
<https://www.parallax.com/package/parallax-standard-servo-downloads/>.

