Egypt FWD

Embedded Systems Professional track – July Cohort

On-demand Traffic Light Control System Project

Prepared by:

Maram Nabil Ibrahim Ali

System Description

The system of this project represents a simple traffic light control system. In the normal mode of the system, the cars' traffic lights switch between each other starting from the green light, to the yellow light which blinks, then the red light, and then the yellow light and at last the green one. Each light is lit for 5 seconds before it switches to the next one. The yellow light always blinks when it's its turn to be lit.

When the pedestrian button is pressed, the system checks for which light color is lit now. If the button is pressed when the cars' Red LED is on, the pedestrian's Green LED and the cars' Red LEDs will be on for five seconds, this means that pedestrians can cross the street while the pedestrian's Green LED is on.

If the button is pressed when the cars' Green LED is on or the cars' Yellow LED is blinking, the pedestrian Red LED will be on, then both Yellow LEDs start to blink for five seconds, then the cars' Red LED and pedestrian Green LEDs are on for five seconds, this means that pedestrian must wait until the Green LED is on.

At the end of the two states, the cars' Red LED will be off and both Yellow LEDs start blinking for 5 seconds and the pedestrian's Green LED is still on.

After the five seconds the pedestrian Green LED will be off and both the pedestrian Red LED and the cars' Green LED will be on.

After that, Traffic lights signals are going to the normal mode again.

System Design

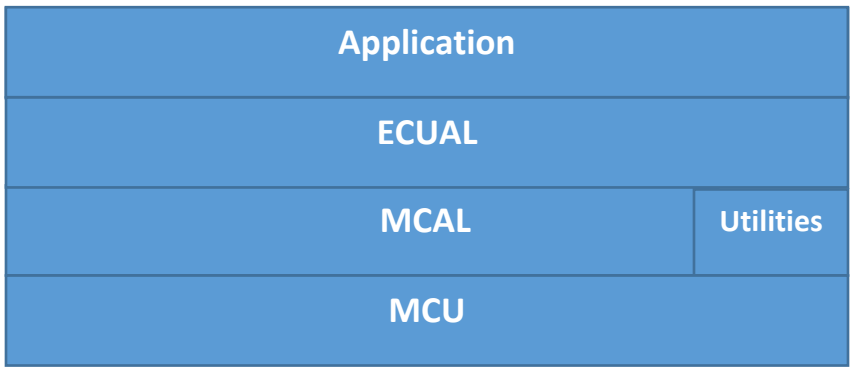
The design I chose for the system is represented in the below figure.

It consists of 3 main software layers above the Microcontroller hardware layer, which are:

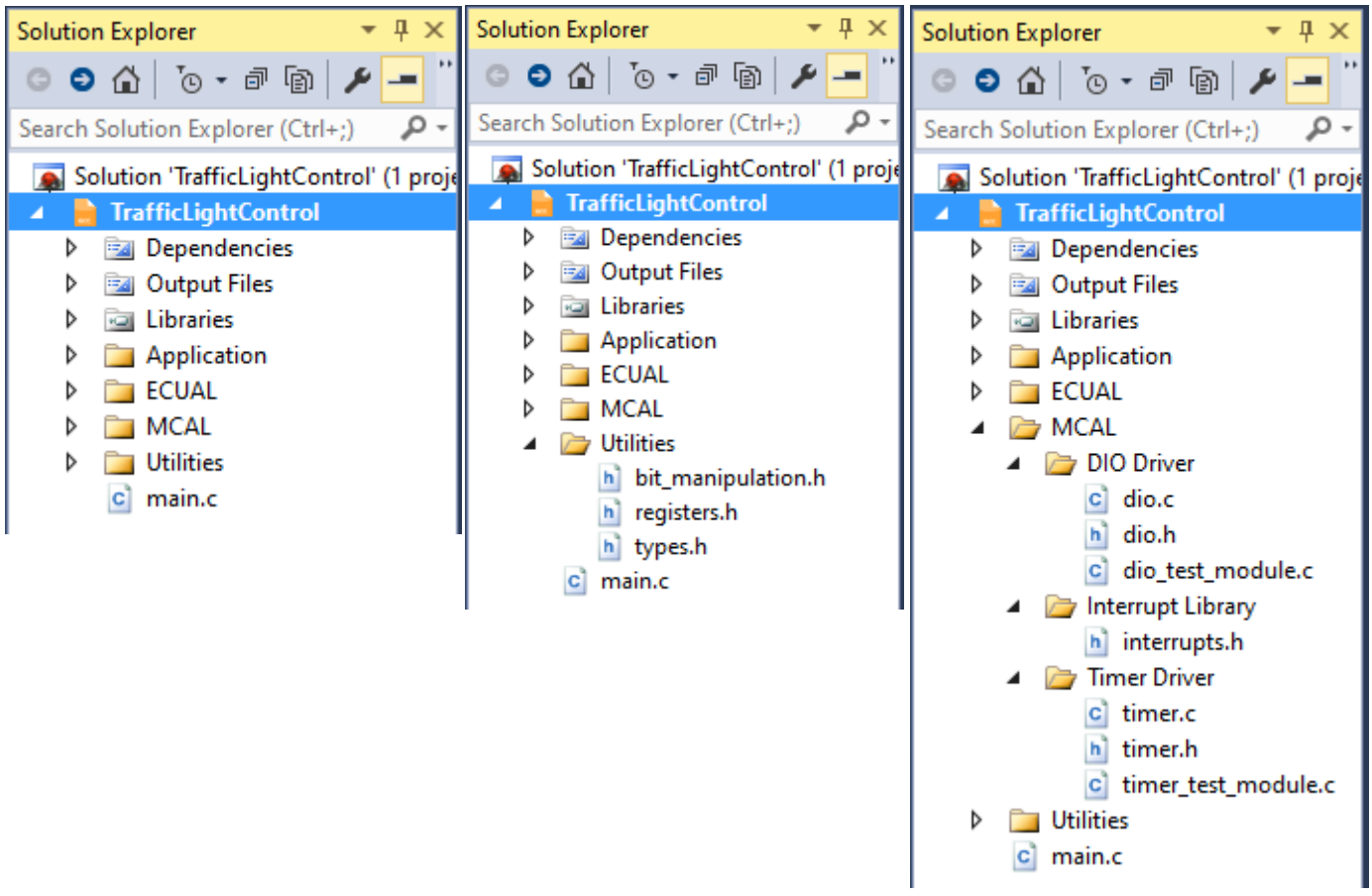
Application, ECUAL, and MCAL.

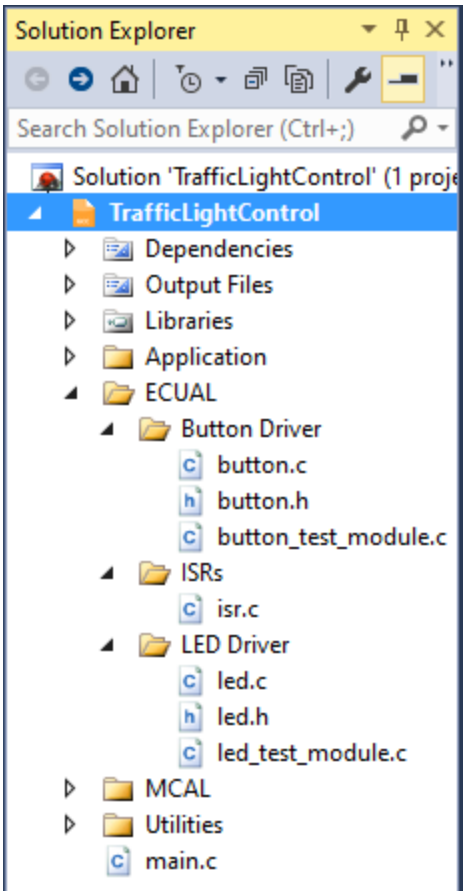
I have used an additional layer called "Utilities" which is represented as a common layer between ECUAL and MCAL layers.

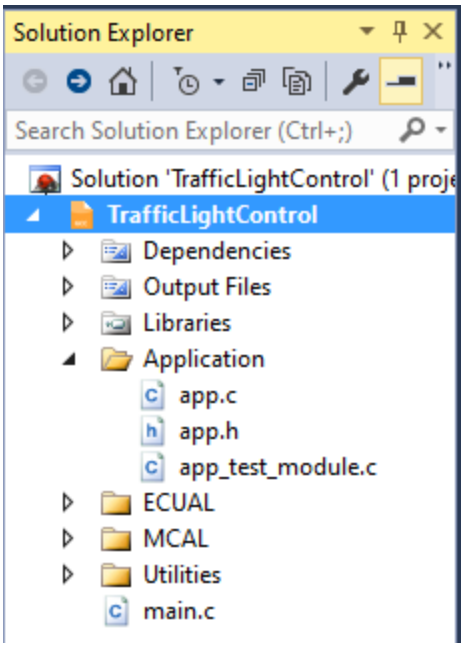
It's represented in this form in the figure because it can be used by both MCAL and ECUAL layers, but the Service Layer and the Application layer are abstracted from it.



Project Folder Structure







Static Architecture

System Layers Description:

- **MCU (Microcontroller Unit):** this layer is the hardware represented in the microcontroller and the connected peripherals to it.
- **MCAL (Microcontroller Abstraction Layer):** this layer contains drivers of the internal peripherals of the MCU, like: DIO driver, Interrupt Library, and the Timer driver.
- **ECUAL (Electronic Unit Abstraction Layer):** this layer contains drivers of external devices like: Button driver, LED driver and ISRs Library (to implement the interrupt service routines and to abstract the implementation of the ISR from the interrupt pins configurations on the target microcontroller).

I made the library of ISRs in ECUAL because it needs to include the libraries of the ECUAL layer to implement the ISR, and at the same time the application layer should be abstracted from it.

- **Utilities:** this layer is a helper layer which includes some header files which are general and can be used with any microcontroller ("bit_manipulation.h", which contains macros for bit directions, bit values and pins numbers, and "types.h", which includes generally used typedefs), and another header file which is specific to the ATmega32 microcontroller (registers.h) which contains macros for the addresses of the used registers in this project.
- **Application:** this layer contains functions for the system software abstracted from the hardware details and connections.

System Drivers:

MCAL:

- **DIO driver:** contains dio.h file, which contains the dio function prototypes and return states, dio.c file, which contains the functions implementations, and dio_test_module.c file, which contains test cases for this driver.
- **Interrupt Library:** contains interrupts.h file, which contains macros for interrupt vectors and pin number, macros for setting and clearing the global interrupt, and a macro for ISR definition.
- **Timer driver:** contains timer.h file, which contains function-like macros and functions prototypes for using the timer, timer.c file, which contains the implementation of these functions, timer_test_module.c file, which contains test cases for this driver.

ECUAL:

- **Button driver:** contains button.h file, which contains macros for button port and pin, type definition for functions return states, and functions prototypes, and button.c file, which contains button functions implementation and interrupt initialization for initializing the button work, and button_test_module.c file, which contains test cases for this driver.

- **LED driver:** contains led.h file, which contains needed typedefs and functions return states, macros for LEDs ports and pins, a global variable declaration and functions prototypes, and led.c file which contains global variable initialization and functions implementation, and led_test_module.c file which contains test cases for this driver.
- **ISRs:** contains isr.c file, which I used to implement the external interrupt 0 vector. It's placed in this layer because it needs to use some macros defined in the led.h file, so to be able to include it, it must be in the ECUAL layer or in a layer above it.

Utilities: contains helper header files, which are:

- **bit_manipulation.h:** contains macros for bit direction, bit values and pins numbers.
- **registers.h:** contains macros for registers addresses in ATmega32, and macros for its ports.
- **types.h:** contains general data types.

Application: contains the following files:

- **app.h:** contains typedefs for return states, and functions prototypes (APP_init and APP_start).
- **app.c:** contains the implementation of APP_init and APP_start functions.
- **app_test_module.**c: contains test cases for this driver.

Drivers' API Description

Timer Driver:

- `#define TMAX (256.0/1000000):`
  - Description: Function-like macro calculates the maximum delay from timer0 which is 8-bit timer, so the maximum delay = 2^number of bits / CPU frequency.
  - Input arguments: None.
  - Output arguments: None.
  - Return: doesn't return a value, it just replaces the TMAX with the division float value.
- `void timer0_init(uint8_t initialValue):`
  - Description: it initializes the timer0 with a given input initial value.

- Input arguments: uint8_t initialValue (the timer initial value).
- Output arguments: None.
- Return: void.

- `void _delay_ms(double milliSeconds):`
  - Description: it delays for an input period of time in milliseconds.
  - Input arguments: double milliSeconds (time of delay in milliseconds).
  - Output arguments: None.
  - Return: void.
- `uint32_t NumberOfOverflows(double delayInMs):`
  - Description: it calculates the number of timer overflows needed to delay the required time period.
  - Input arguments: double delayInMs (the delay time in milliseconds).
  - Output arguments: None.
  - Return: uint32_t (the number of overflows).

DIO Driver:

- `DIO_ERROR_STATE:`
  - Description: a typedef which indicates whether an error related to DIO functions occurred or not.
  - Values: DIO_ERROR (an error occurred), DIO_OK (no error occurred).
- `DIO_ERROR_STATE DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction):`
  - Description: it initializes the required pin in the required port by a specific direction (input / output).
  - Input arguments: uint8_t portNumber (the port needed), uint8_t pinNumber (the pin number needed), uint8_t direction (IN / OUT).
  - Output arguments: None.
  - Return: DIO_ERROR_STATE (whether the initialization occurs successfully or not).
- `DIO_ERROR_STATE DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value):`
  - Description: it writes a value (either HIGH or LOW), on the required pin in the required port.

- Input arguments: uint8_t portNumber (the port needed), uint8_t pinNumber (the pin number needed), uint8_t value (HIGH/LOW).
- Output arguments: None.
- Return: DIO_ERROR_STATE (whether the write operation occurs successfully or not).

- `DIO_ERROR_STATE DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t* value):`
  - Description: it reads the value of the required pin in the required port.
  - Input arguments: uint8_t portNumber (the port needed), uint8_t pinNumber (the pin number needed).
  - Output arguments: uint8_t* value (a pointer to a variable that saves the pin value).
  - Return: DIO_ERROR_STATE (whether the read operation occurs successfully or not).

- `DIO_ERROR_STATE DIO_toggle(uint8_t portNumber, uint8_t pinNumber):`
  - Description: it toggles the value of the required pin in the required port (if it's zero it becomes one or if it's one it becomes zero).
  - Input arguments: uint8_t portNumber (the port needed), uint8_t pinNumber (the pin number needed).
  - Output arguments: None.
  - Return: DIO_ERROR_STATE (whether the toggle operation occurs successfully or not).

Button Driver:

- `BUTTON_ERROR_STATE:`
  - Description: a typedef which indicates whether an error related to the button functions occurred or not.
  - Values: BUTTON_ERROR (an error occurred), BUTTON_OK (no error occurred).

- `BUTTON_ERROR_STATE Button_init(uint8_t ButtonPort, uint8_t ButtonPin):`
  - Description: it initializes the button pin as input, and initializes the interrupt for the button to function.
  - Input arguments: uint8_t ButtonPort (the port of the button), uint8_t ButtonPin (the pin number of the button).

- Output arguments: None.
- Return: BUTTON_ERROR_STATE (whether the button initialization occurs successfully or not).

LED Driver:

- `LED_ERROR_STATE:`
  - Description: a typedef which indicates whether an error in the LED functions occurred or not.
  - Values: LED_ERROR (an error occurred), LED_OK (no error occurred).
- `ON_LED:`
  - Description: a typedef which is used to determine which LED is on / blinking right now.
  - Values: CAR_GREEN_LED, CAR_YELLOW_LED, CAR_RED_LED.
- `LED_STATE:`
  - Description: a typedef which is used to determine the state of the required LED, whether it's on, off or blinking.
  - Values: ON, OFF, BLINKING.
- `LED_ERROR_STATE LED_init(uint8_t ledPort, uint8_t ledPin):`
  - Description: it initializes the required LED pin in the required port as output.
  - Input arguments: uint8_t ledPort (the required led port), uint8_t ledPin (the required led pin).
  - Output arguments: None.
  - Return: LED_ERROR_STATE (whether the initialization occurs successfully or not).
- `LED_ERROR_STATE LED_on(uint8_t ledPort, uint8_t ledPin):`
  - Description: it turns on the required LED in the required port by writing 1 to it.
  - Input arguments: uint8_t ledPort (the required led port), uint8_t ledPin (the required led pin).
  - Output arguments: None.
  - Return: LED_ERROR_STATE (whether the write operation occurs successfully or not).
- `LED_ERROR_STATE LED_off(uint8_t ledPort, uint8_t ledPin):`
  - Description: it turns off the required LED in the required port by writing 0 to it.

- Input arguments: uint8_t ledPort (the required led port), uint8_t ledPin (the required led pin).
- Output arguments: None.
- Return: LED_ERROR_STATE (whether the write operation occurs successfully or not).

- `LED_ERROR_STATE LED_toggle(uint8_t ledPort, uint8_t ledPin):`
  - Description: it toggles the value of the required LED in the required port.
  - Input arguments: uint8_t ledPort (the required led port), uint8_t ledPin (the required led pin).
  - Output arguments: None.
  - Return: LED_ERROR_STATE (whether the toggle operation occurs successfully or not).

- `LED_ERROR_STATE two_LEDs_blink(uint8_t ledPort1, uint8_t ledPin1, uint8_t ledPort2, uint8_t ledPin2, uint16_t delayMilli):`
  - Description: it makes two leds blink for an input time period (delayMilli) and with a toggle delay of 200ms.
  - Input arguments: uint8_t ledPort1 (the first required led port), uint8_t ledPin1 (the first required led pin), uint8_t ledPort2 (the second required led port), uint8_t ledPin2 (the second required led pin), uint16_t delayMilli (the required delay period in milliseconds).
  - Output arguments: None.
  - Return: LED_ERROR_STATE (whether the function occurs successfully or not).

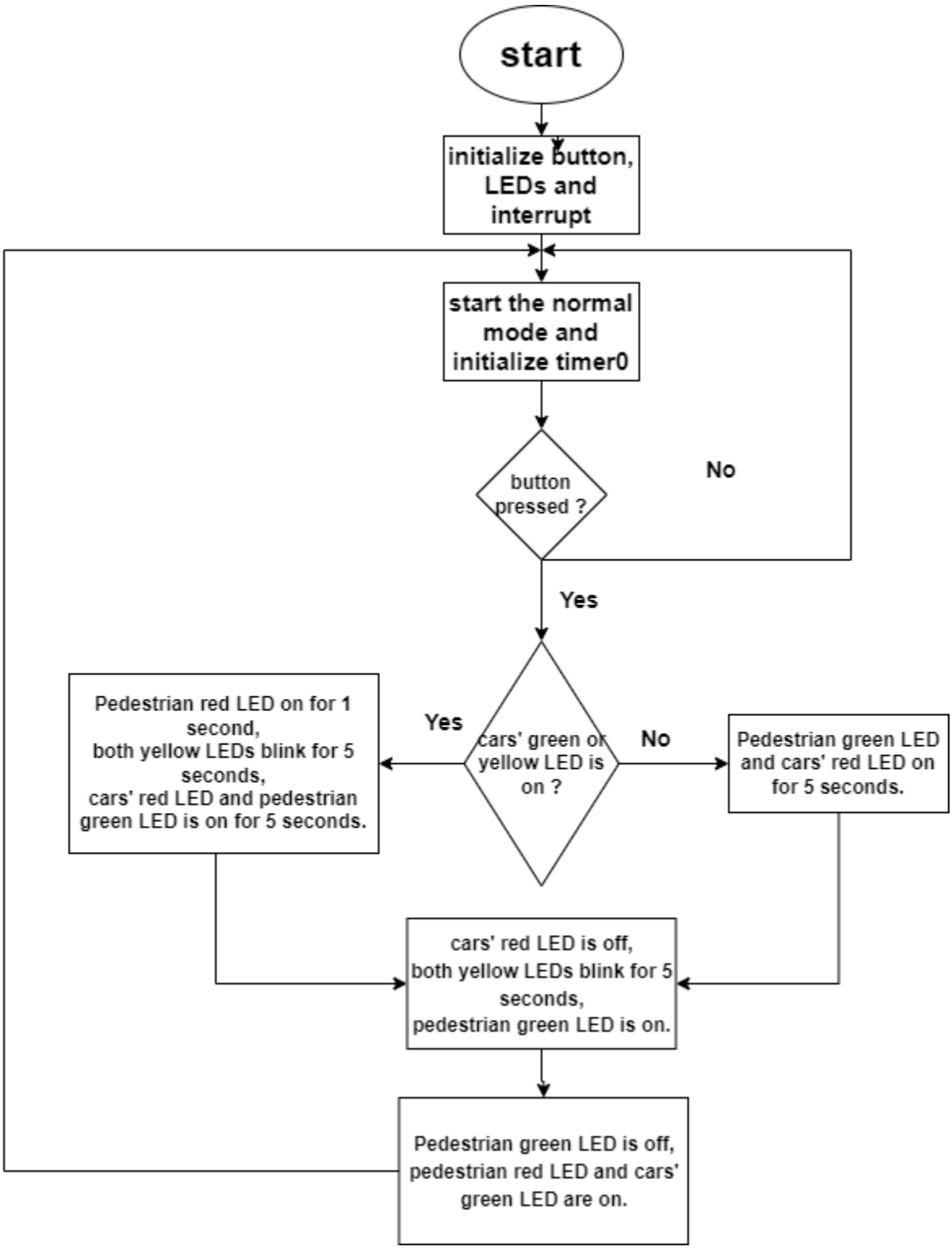
- `LED_STATE getLEDState(uint8_t ledPort, uint8_t ledPin):`
  - Description: it gets the state of the required led in the required port, whether it's on, off or blinking.
  - Input arguments: uint8_t ledPort (the required led port), uint8_t ledPin (the required led pin).
  - Output arguments: None.
  - Return: LED_STATE (whether the led state is on, off or blinking).

- `ON_LED getOnLED():`
  - Description: it gets which led of the cars' leds is on/blinking right now.
  - Input arguments: None.
  - Output arguments: None.
  - Return: ON_LED (whether the on/blinking led is the red, yellow or the green led).

Application:

- `APP_ERROR_STATE:`
  - Description: a typedef which indicates whether an error in the application functions occurred or not.
  - Values: APP_ERROR (an error occurred), APP_OK (no error occurred).
- `APP_ERROR_STATE APP_init():`
  - Description: it initializes the LEDs and Button pins.
  - Input arguments: None.
  - Output arguments: None.
  - Return: APP_ERROR_STATE (whether all the initializations occur successfully or not).
- `APP_ERROR_STATE APP_start():`
  - Description: it starts the application at the normal mode and waits for the button interrupt (pressing) to occur.
  - Input arguments: None.
  - Output arguments: None.
  - Return: APP_ERROR_STATE (whether the starting of the application occurs successfully or not).

System Flow Chart



System Constraints

- The pedestrian button can’t be double pressed to perform a specific action. If it’s double pressed, the first press will only be considered and will perform the usual operations of the pedestrian mode.
- If the pedestrian button is pressed a long press, nothing will happen.
- Only a short press on the pedestrian button will be considered to perform the button action.

Explaining Video