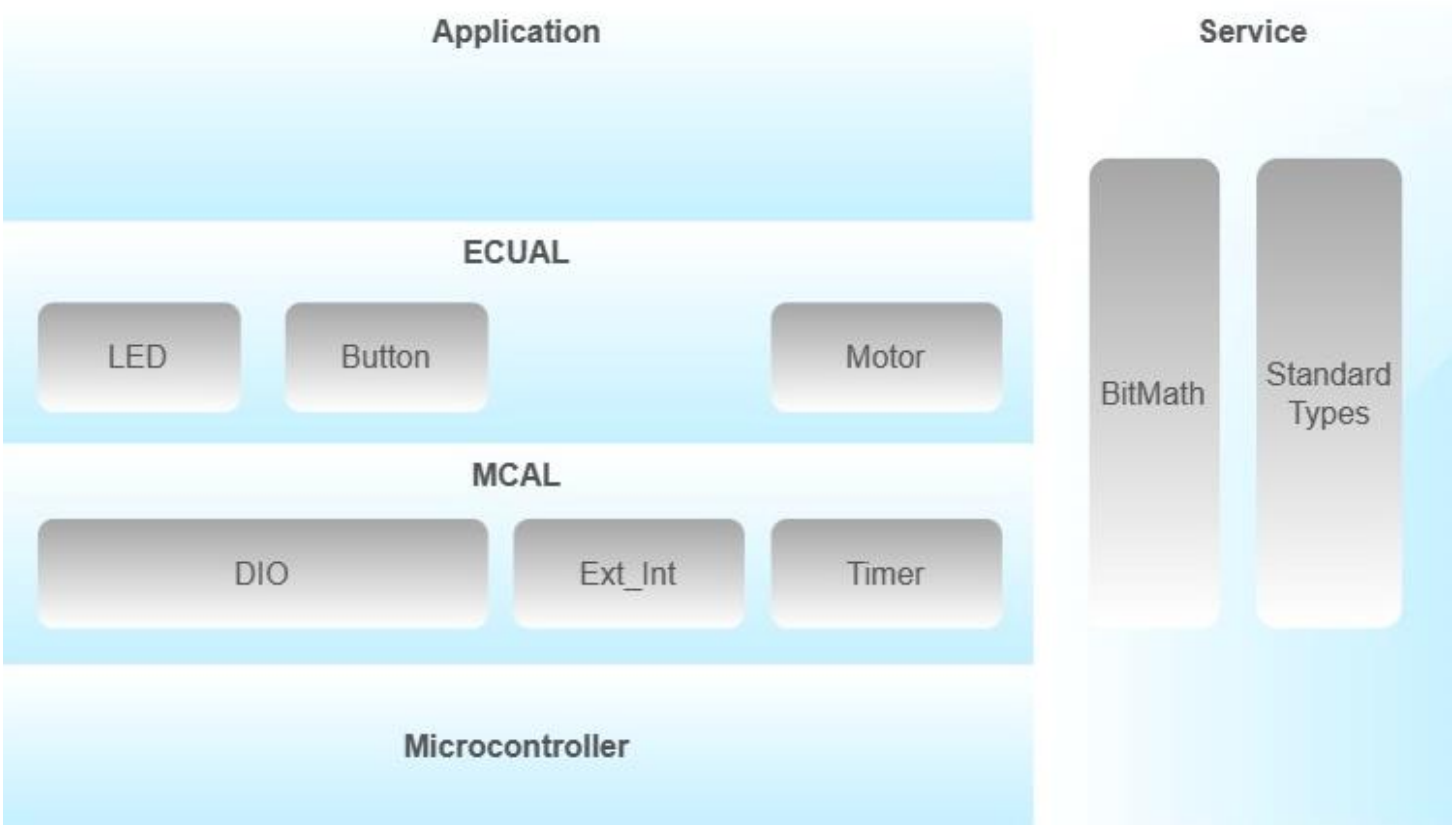
MOVING CAR PROJECT

BY: Momen Hassan, Ahmed Atef, and Ahmed Mohamed Hesham

Contents

Moving Car Design

Layered Architecture



INTRODUCTION

This project is controlling a 4 DC motors toy car using Atmega32 and controlling its speed by a generated PWM using the normal mode of the timer.

It uses external interrupts too, to suddenly stop the project.

It consists of four layers:

1- APP

This layer is responsible for integrating application modules and peripherals to perform project functionality via using their APIs

2- ECUAL

In this layer modules' drivers are developed which are the buttons' driver, LEDs' driver, and motors' driver.

This layer is like a middle junction between the application layer and the microcontroller abstraction layer

3- MCAL

In this layer, peripherals' drivers are developed: DIO's driver, timers' driver, and external interrupts' driver.

4- SERVICE

This layer consists of files.h which will serve the main three layers while developing, like it has important data types' type defs, and bit manipulation macros like functions

Module, Peripherals, & Supporting Drivers Description

DIO (Digital Input/Output): This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

Motor: This module is responsible for controlling the motors in the car, such as M1, M2, M3, and M4 as mentioned in the system requirements. It may include functions for setting motor speed, direction, and handling motor control signals.

Button: This module deals with the buttons in the system, such as PB1 and PB2 as mentioned in the system requirements. It may include functions for detecting button presses and handling button-related events.

EXT_INT (External Interrupt): This module handles external interrupts, which are signals from external sources that can trigger interrupts in the microcontroller or microprocessor. It may include functions for configuring and handling external interrupts from external devices, such as buttons or sensors.

LED: The LED module is responsible for controlling the LEDs (LED1, LED2, LED3, LED4) mentioned in the system requirements. It may include functions for setting the LED states (e.g., ON or OFF), controlling LED brightness or color (if applicable), and handling any other operations related to LED control

Timer: This module deals with timer operations, such as configuring and handling timers in the microcontroller or microprocessor. It may include functions for setting timer intervals, handling timer interrupts, and measuring time. And This module deals with generating PWM signals using normal mode, which are used for controlling the intensity of an output signal, such as controlling the speed of motors or the brightness of LEDs. It may include functions for configuring and controlling PWM signals.

BIT_MATH: This module provides functions for performing bitwise operations, such as AND, OR, XOR, and shifting, which are commonly used for manipulating individual bits in registers or memory locations.

Standard Types: This module includes standard data types, such as integer types, floating-point types, and Boolean types, which are used for representing data in a standardized way across the system.

DRIVERS' DOCUMENTATION

1. DIO

DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction);

Function Name	**DIO_init**
Description	Initializes DIO pins' direction, output current, and internal attach
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_DIRECTION, E_OK

DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value);

Function Name	**DIO_write**
Description	Write on DIO pins' a specific output High or Low
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t value
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_VALUE, E_OK

DIO_toggle(uint8_t portNumber, uint8_t pinNumber);

Function Name	**DIO_toggle**
Description	Toggle the output of a specific pin
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t *value);

Function Name	**DIO_read**
Description	Read input from a pin and send it back in a pointer to uint8_t
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	uint8_t *value
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

2. EXTERNAL INTERRUPTS

EXTINT_Init (uint8_t intNumber);

Function Name	**EXT_INT_init**
Description	Initializes External Interrupts pins' mode.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t int Number
Parameters (out)	None
Return Value	E_OK, WRONG_INTERRUPT_NUMBER

EXTINT_setCallBackInt (uint8_t intNumber, void (*funPtr) (void));

Function Name	**EXT_INT_setCallBackIntx**
Description	Sends pointer to function to be called when the interrupt fires
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Void (*funPtr) (void)
Parameters (out)	None
Return Value	None

3. TIMERS

en_timerError_t TIMER_init(u8 u8_a_timerUsed);

Function Name	**TIMER_init**
Description	Initializes a specific timer to work as a CTC or overflow timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);

Function Name	**TIMER_setTime**
Description	Used to set time at which the timer interrupt will fires and execute a desired function
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed, uint32_t desiredTime
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_start(u8 u8_a_timerUsed);

Function Name	**TIMER_start**
Description	Start specific timer to count
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_stop(u8 u8_a_timerUsed);

Function Name	**TIMER_stop**
Description	Stop specific timer from counting
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);

Function Name	**TIMER_pwmGenerator**
Description	Generates PWM signal using normal mode for a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u8_a_desiredDutyCycle
Parameters (out)	None
Return Value	en_timerError_t

void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));

Function Name	**TIMER_setCallBack**
Description	Initializes Sends pointer to function to be called when the timer’s interrupt fires
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	None

en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_stopInterrupt**
Description	Disable a specific timer's peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

Function Name	**TIMER_enableInterrupt**
Description	Generates a delay using a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u32_a_timeInMS
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_enableInterrupt**
Description	Enables a specific timer's peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

4. LED

LED_init(uint8_t ledPort,uint8_t ledPin);

Function Name	**LED_init**
Description	Initializes a specific LED as output
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8 u8_a_ledPort, u8 u8_a_ledPin
Parameters (out)	None
Return Value	en_ledError_t

LED_on(uint8_t ledPort,uint8_t ledPin);

Function Name	**LED_on**
Description	Writes on a specific LED’s pin HIGH
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8 u8_a_ledPort, u8 u8_a_ledPin
Parameters (out)	None
Return Value	en_ledError_t

en_ledError_t LED_off(uint8_t ledPort,uint8_t ledPin);

Function Name	**LED_off**
Description	Writes on a specific LED’s pin LOW
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8 u8_a_ledPort, u8 u8_a_ledPin
Parameters (out)	None
Return Value	en_ledError_t

en_ledError_t LED_toggle(uint8_t ledPort,uint8_t ledPin);

Function Name	**LED_toggle**
Description	Toggles a specific LED
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8 u8_a_ledPort, u8 u8_a_ledPin
Parameters (out)	None
Return Value	en_ledError_t

5. BUTTON

EN_buttonError_t BUTTON_init(uint8_t buttonPort, uint8_t buttonPin);

Function Name	**Button_init**
Description	Initializes a specific button as input
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t buttonPort, uint8_t buttonPin
Parameters (out)	None
Return Value	EN_buttonError_t

EN_buttonError_t BUTTON_read(uint8_t buttonPort, uint8_t buttonPin, uint8_t *buttonState);

Function Name	**BUTTON_read**
Description	Gets a specific button value
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t buttonPort, uint8_t buttonPin
Parameters (out)	Uint8_t *buttonState
Return Value	EN_buttonError_t

6. MOTOR

MOTOR_init (void);

Function Name	**MOTOR_init**
Description	Initializes motor by using motor database structure
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

MOTOR_setDirection(u8 u8_a_motorUsed ,u8 u8_a_direction)

Function Name	**MOTOR_setDirection**
Description	Sets motor’s direction
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t direction
Parameters (out)	None
Return Value	EN_motorError_t

MOTOR_speed(u8 u8_a_motorUsed, u8 u8_a_speed)

Function Name	**MOTOR_speed**
Description	Determines motor speed
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t setSpeed
Parameters (out)	None
Return Value	EN_motorError_t

MOTOR_start()

Function Name	**MOTOR_start**
Description	Starts the motors
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

MOTOR_stop()

Function Name	**MOTOR_stop**
Description	Stops the motors
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

7. APP

void APP_initModules(void);

Function Name	APP_initModules
Description	Initialize drivers used for the application and and global variables used too
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_superLoop (void);

Function Name	APP_superLoop
Description	Has the super loop of the application and it's polling on button state if pressed or released
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_carMoveForward(void);

Function Name	APP_carMoveForward
Description	Start motors to move in the same direction with 50% of max speed and turn on led 0
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_carTurnRight(void);

Function Name	**APP_carTurnRight**
Description	Start motors to move in opposite directions with a predefined duty cycle of max speed and turn on led 2, it's responsible for rotating
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_sysTickTask(void);

Function Name	**APP_sysTickTask**
Description	Checks on the number of ticks -overflows- happened since starting the timer and compare with states that have specific timing and apply a specific task for each state
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_carMoveRight(void);

Function Name	**APP_carMoveRight**
Description	Start motors to move in the same direction with 30% of max speed and turn on led 3
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_carStop(void);

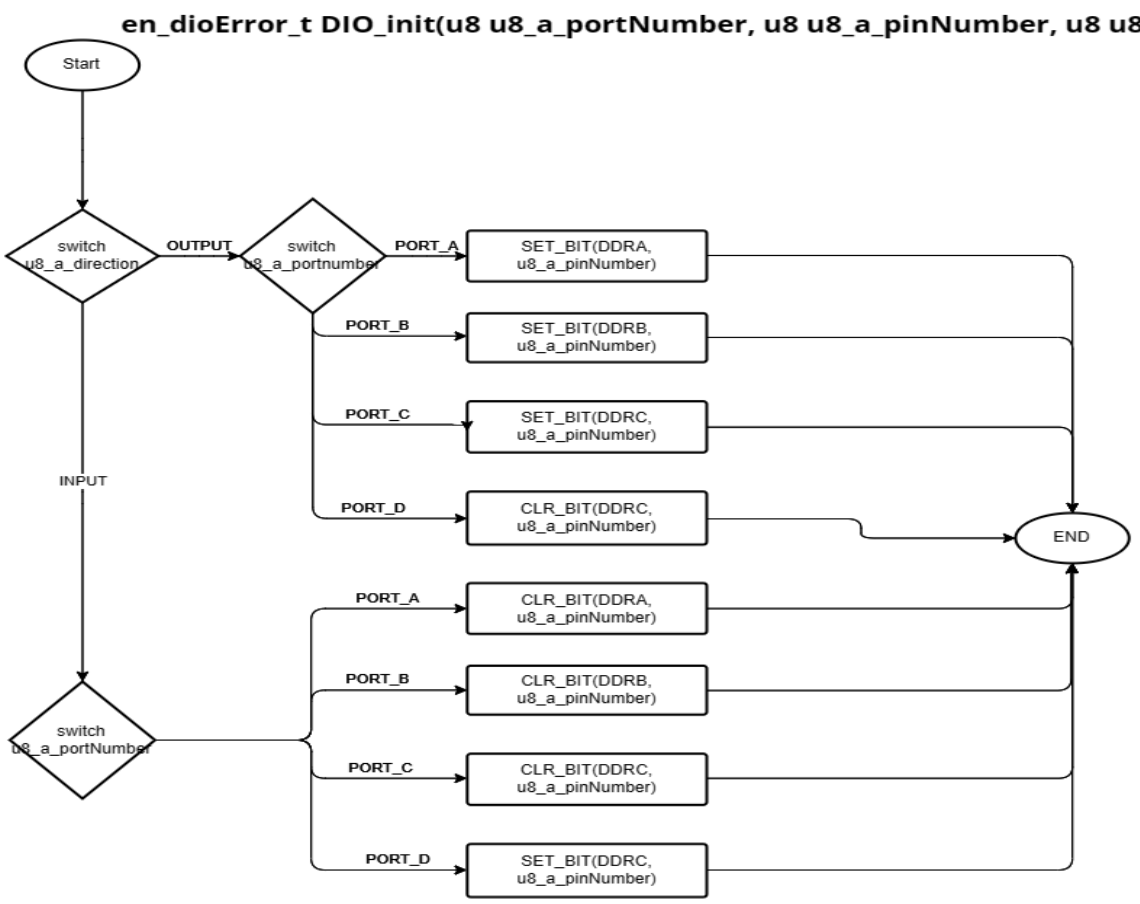
Function Name	**APP_carStop**
Description	Stops motors and turn on led 1
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

void APP_button1Task(void);

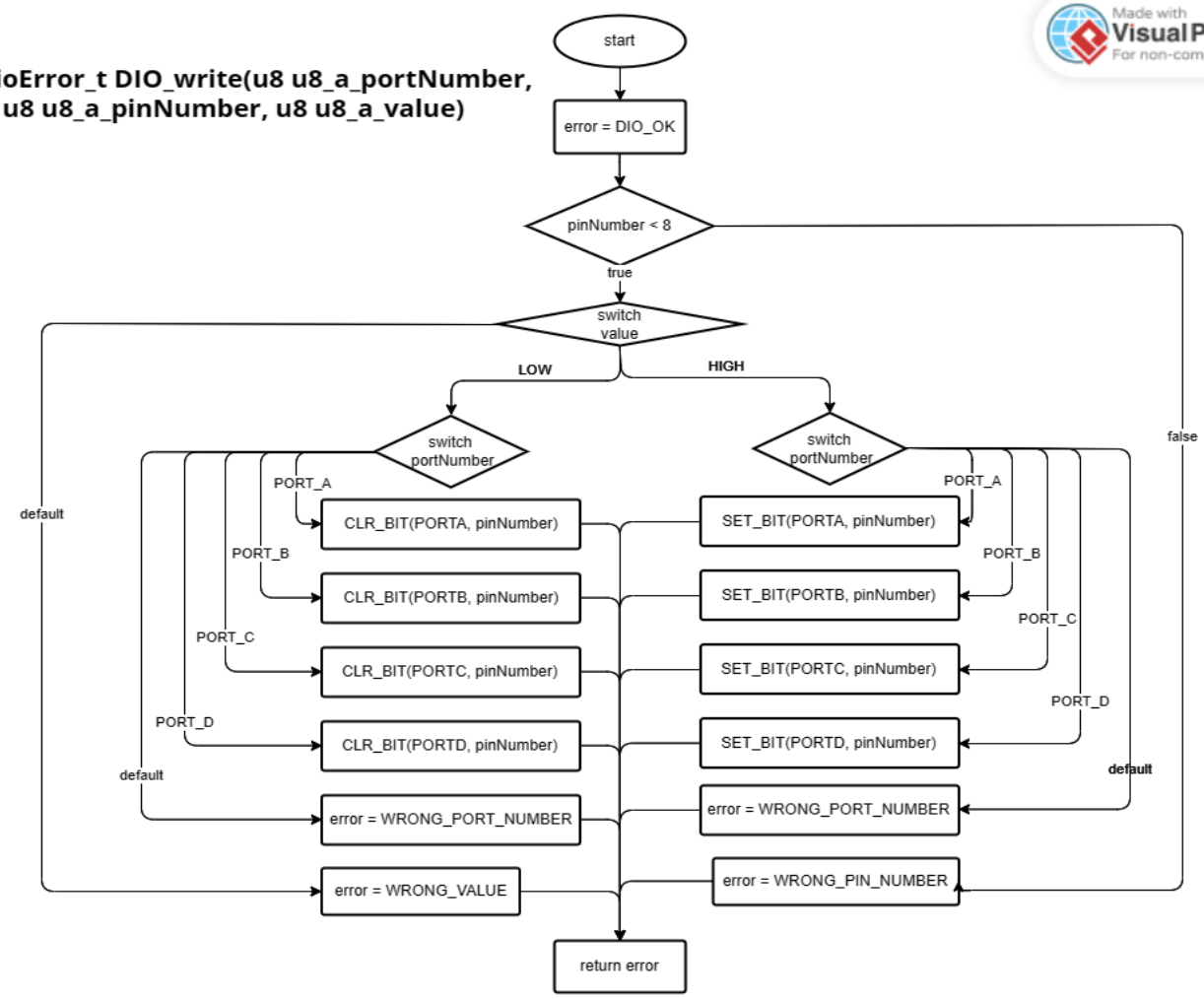
Function Name	**APP_button1Task**
Description	Called by external interrupt 2 when it is triggered and turn of motors and leds
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	NONE

FUNCTIONS' FLOWCHARTS

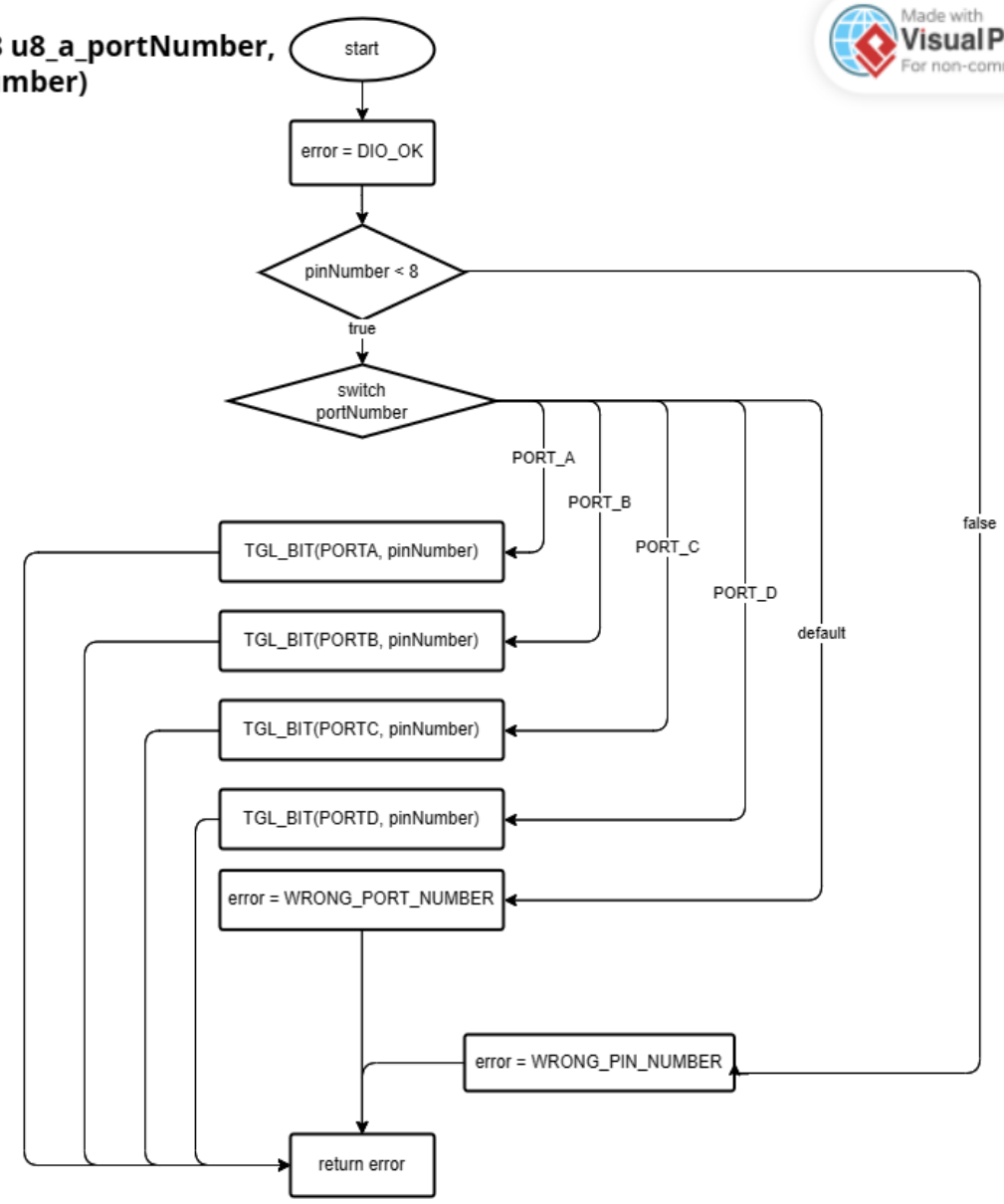
1. DIO



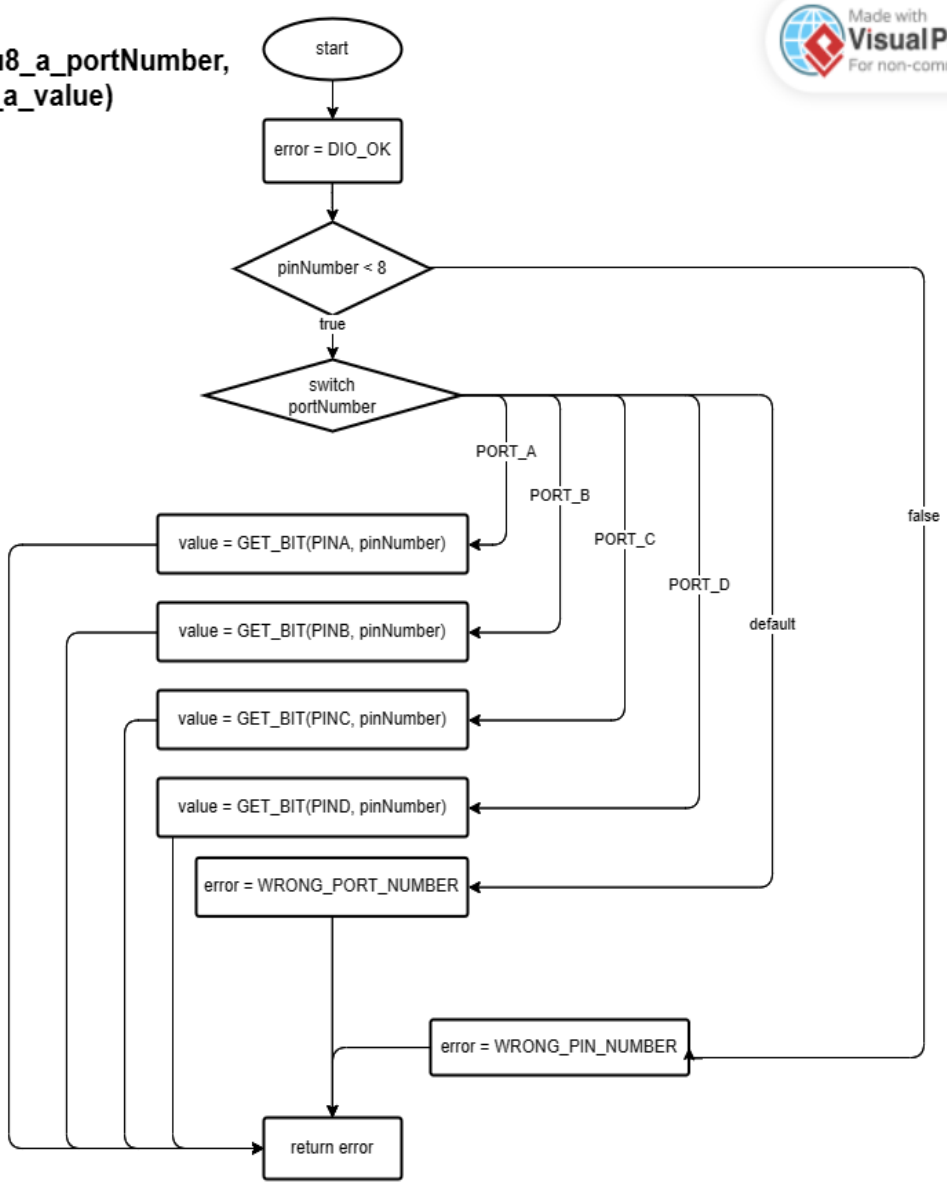
en_dioError_t DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value)



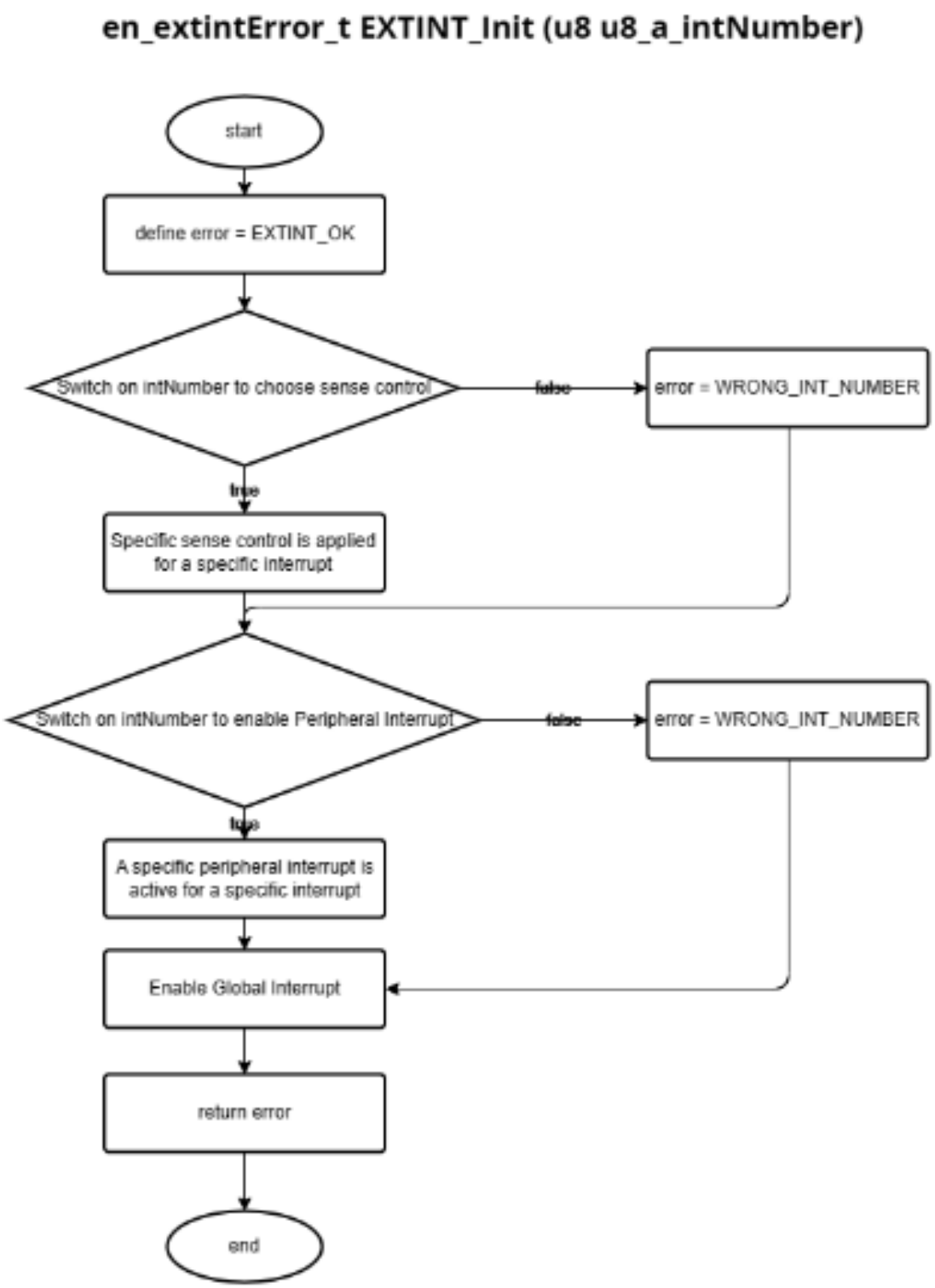
en_dioError_t DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber)



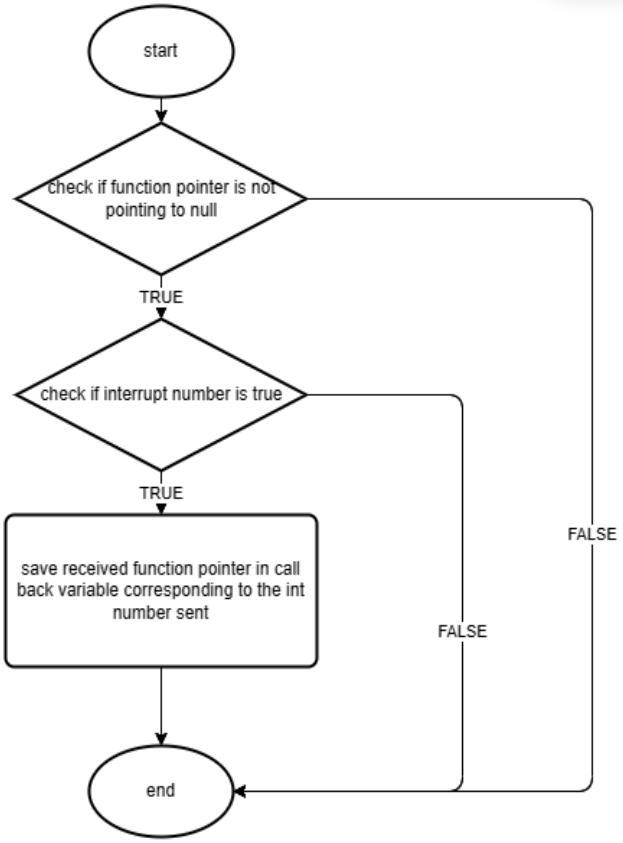
en_dioError_t DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 *u8_a_value)



2. EXTERNAL INTERRUPTS

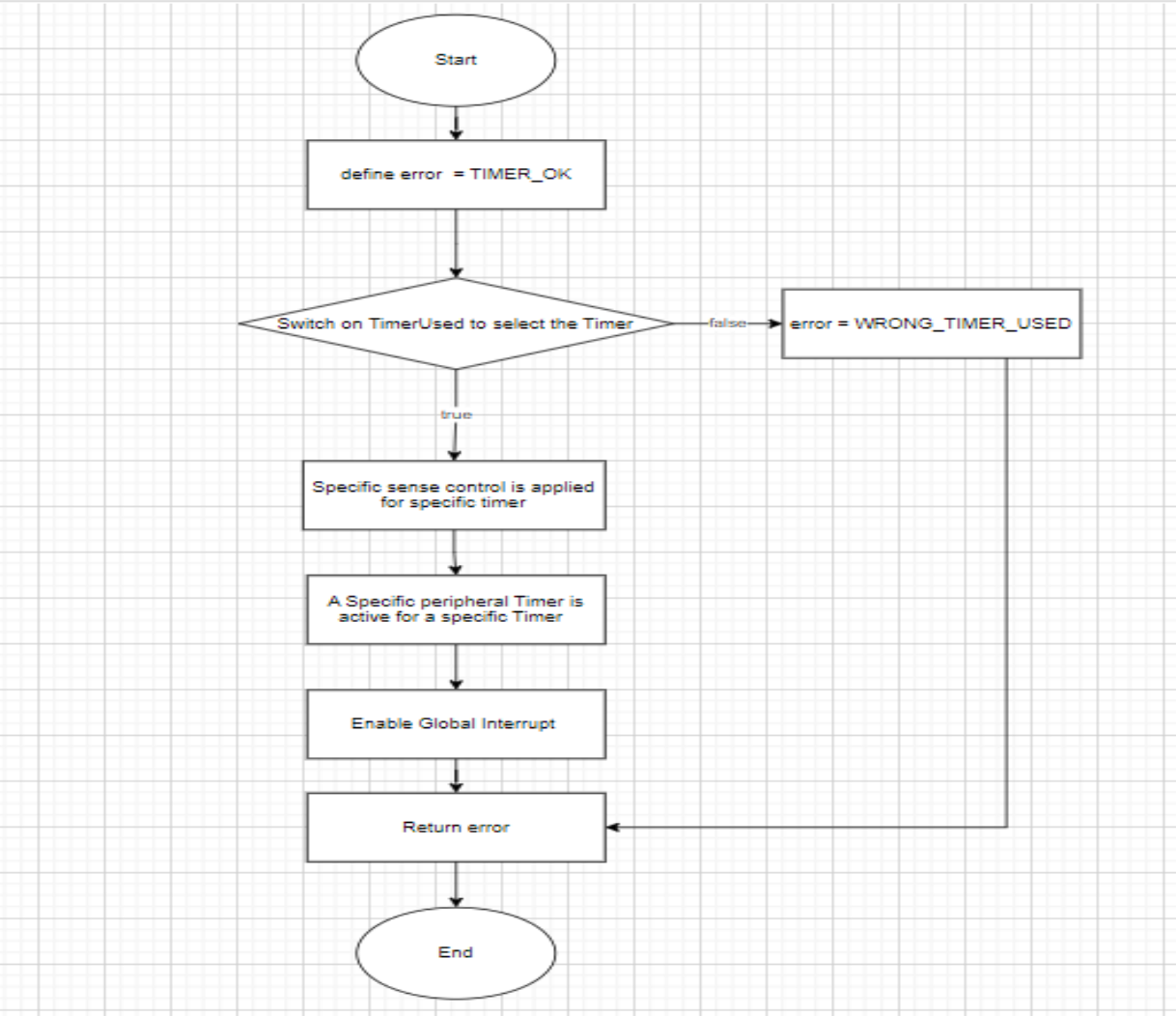


void EXTINT_setCallBackInt (u8 u8_a_intNumber, void (*funP

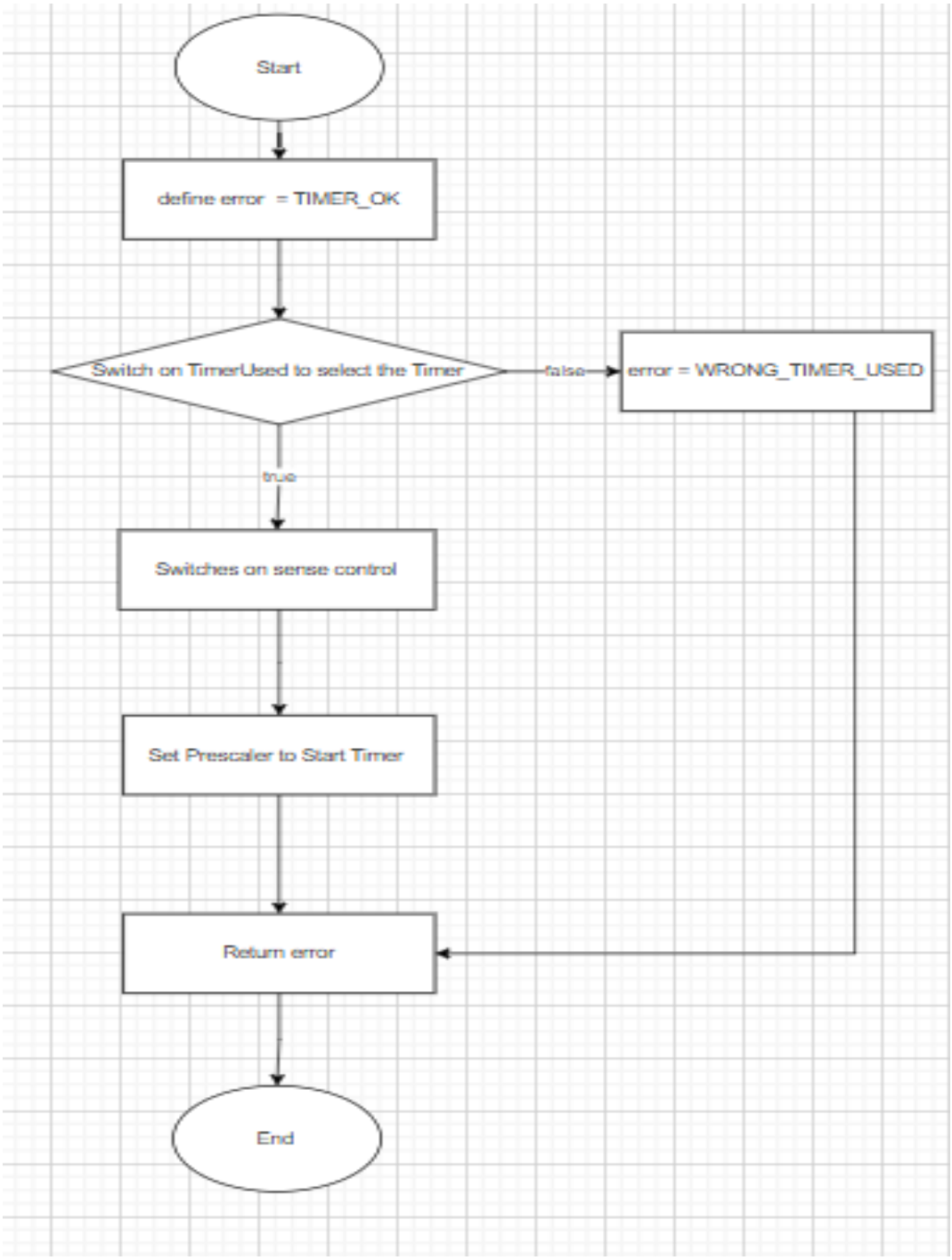


3. TIMERS

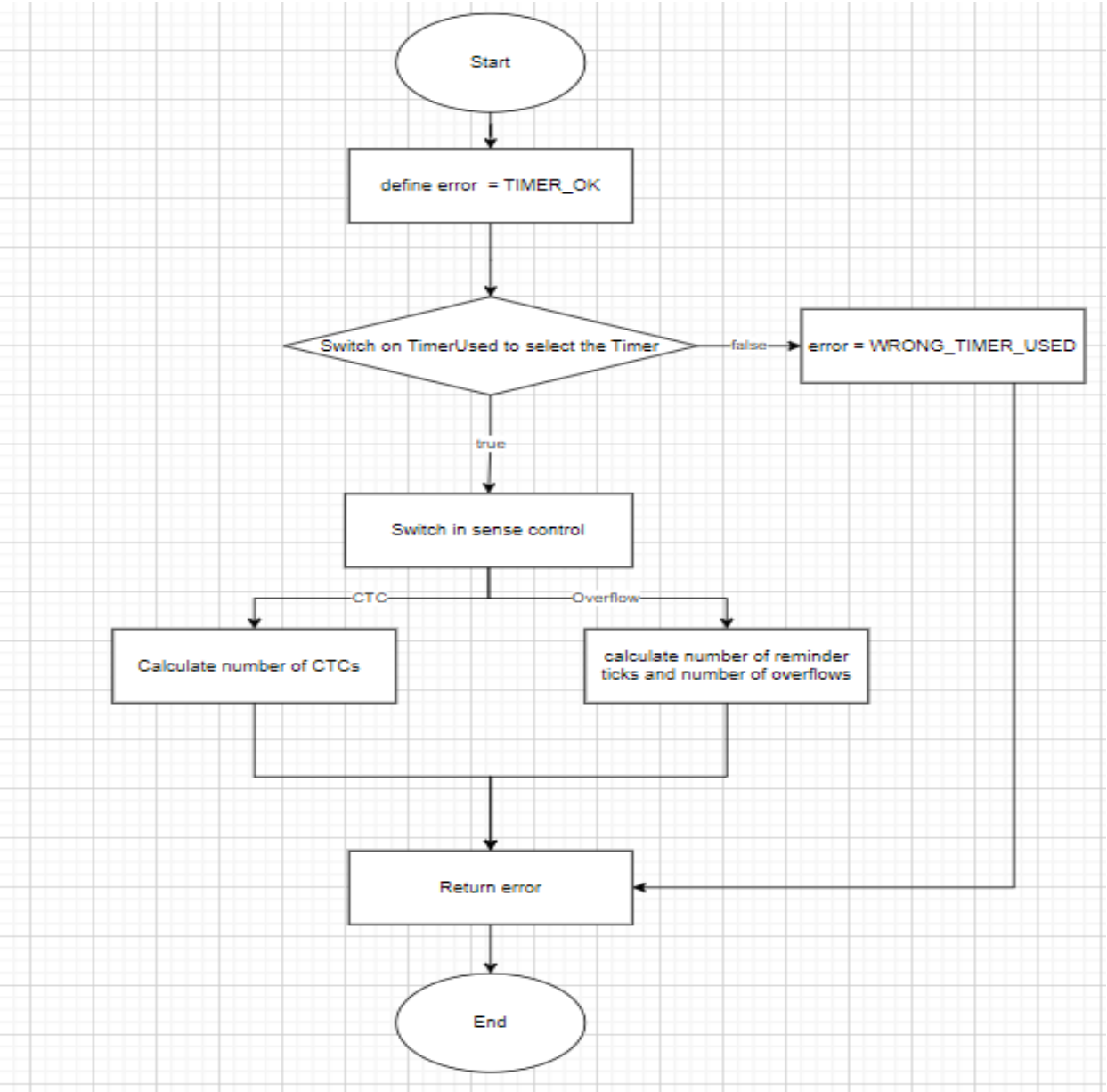
en_timerError_t TIMER_init(u8 u8_a_timerUsed);



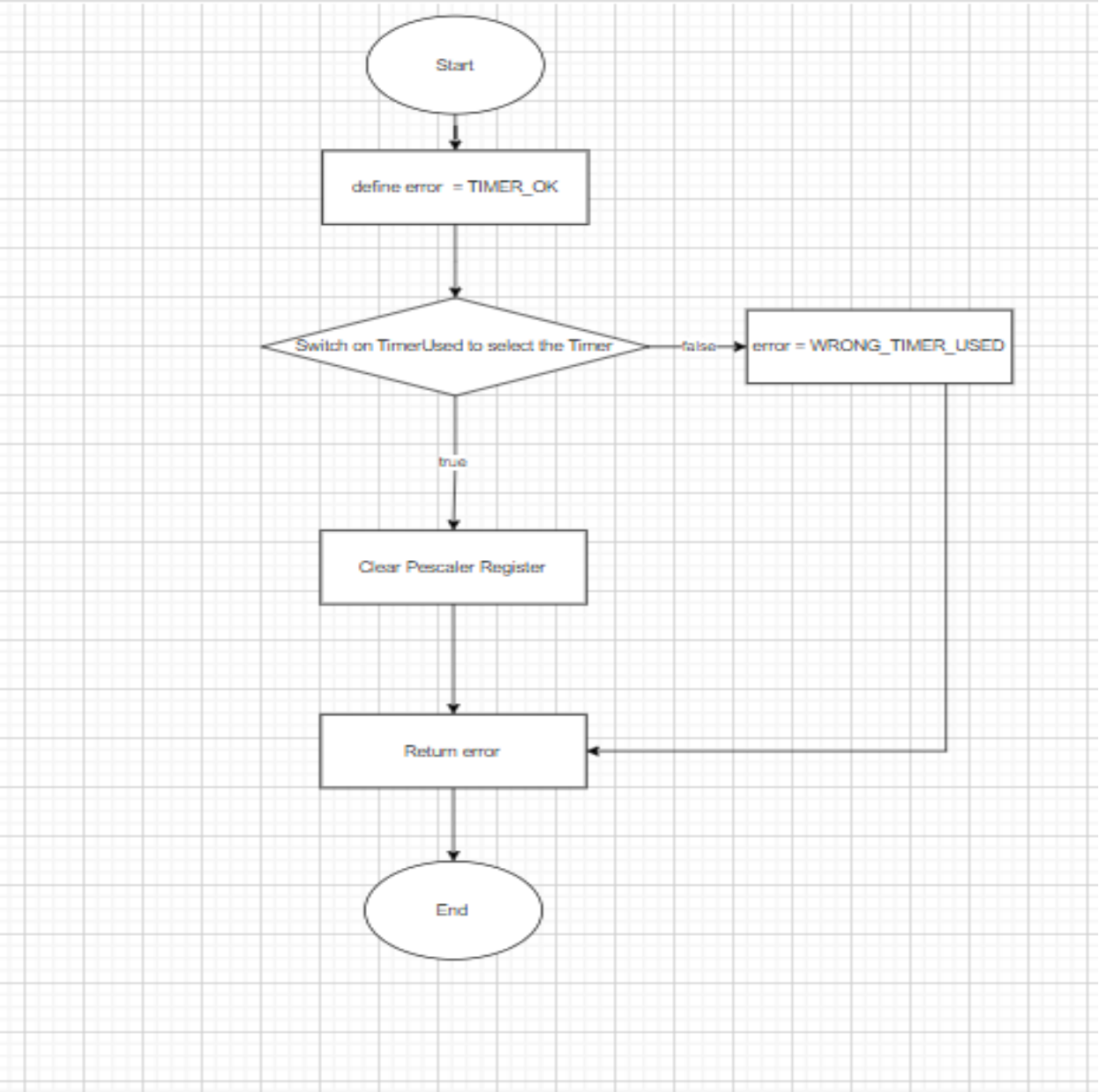
en_timerError_t TIMER_start(u8 u8_a_timerUsed);



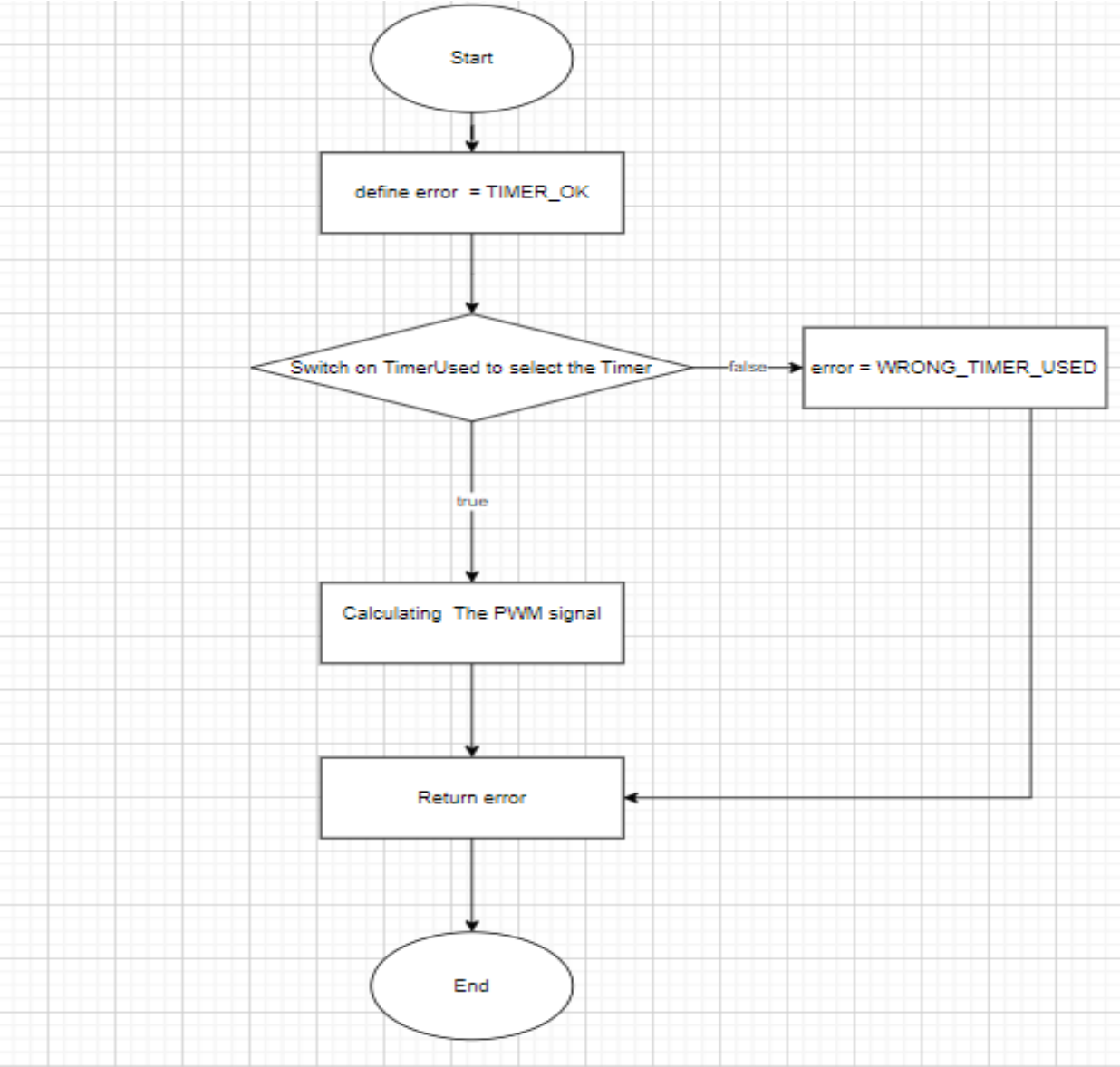
en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);



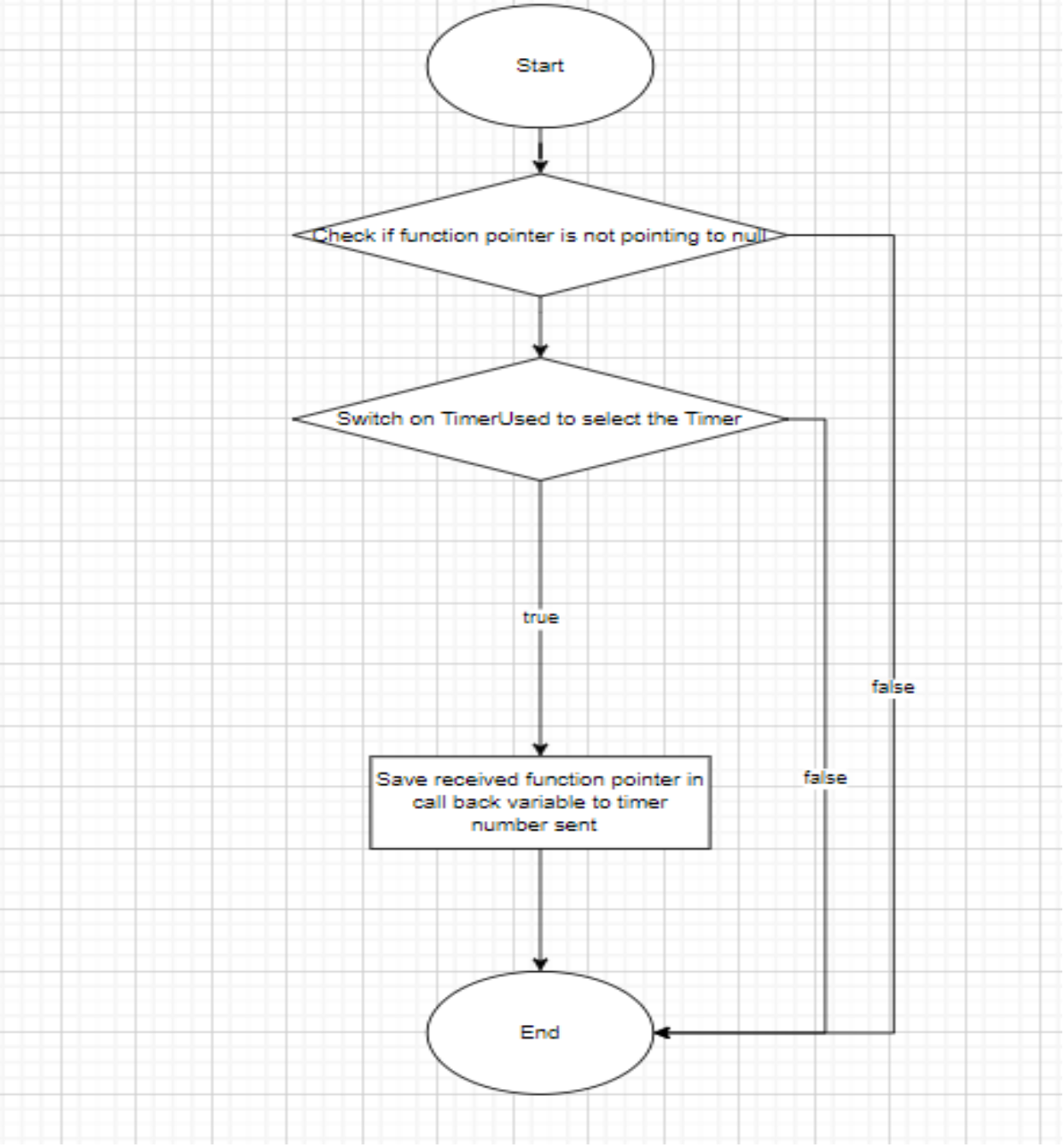
en_timerError_t TIMER_stop(u8 u8_a_timerUsed);



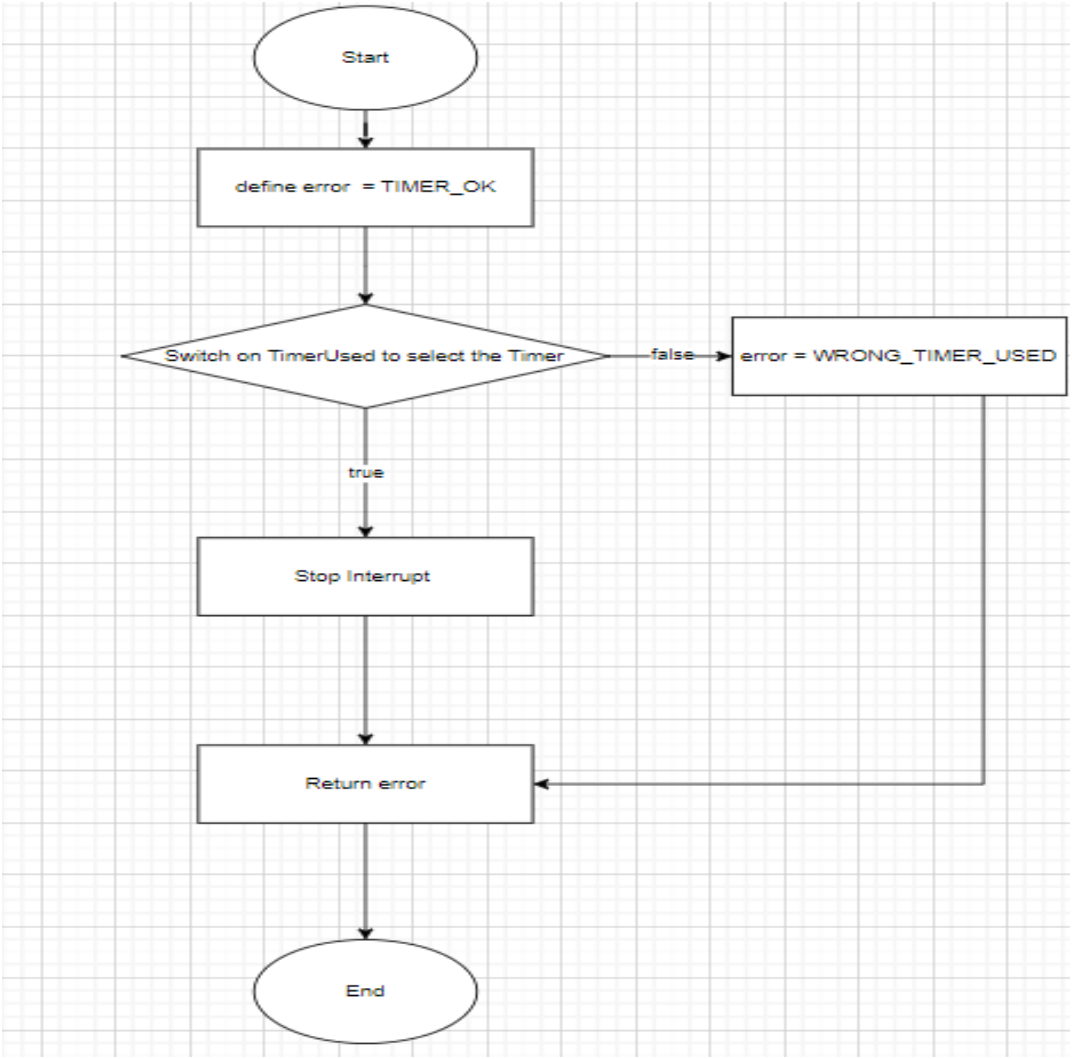
en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);



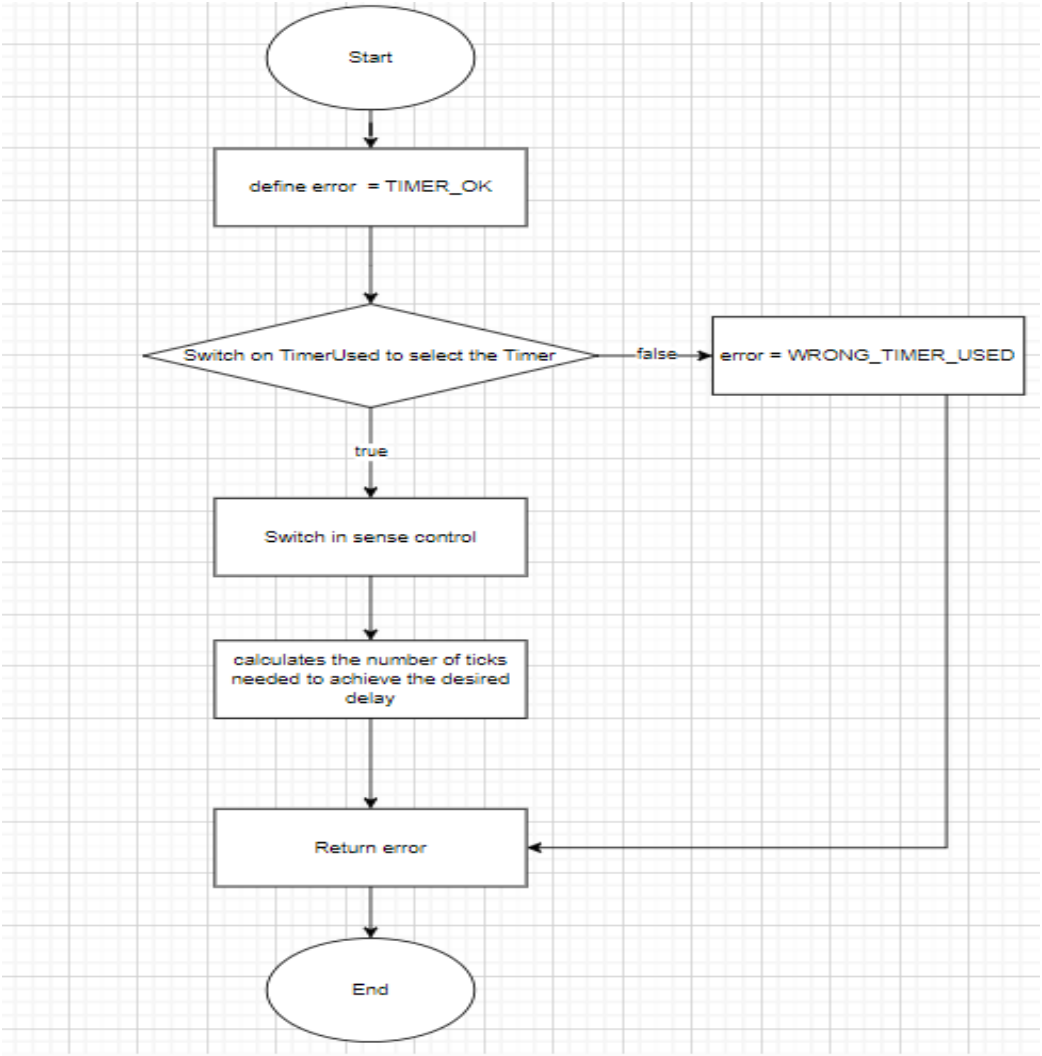
Void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));



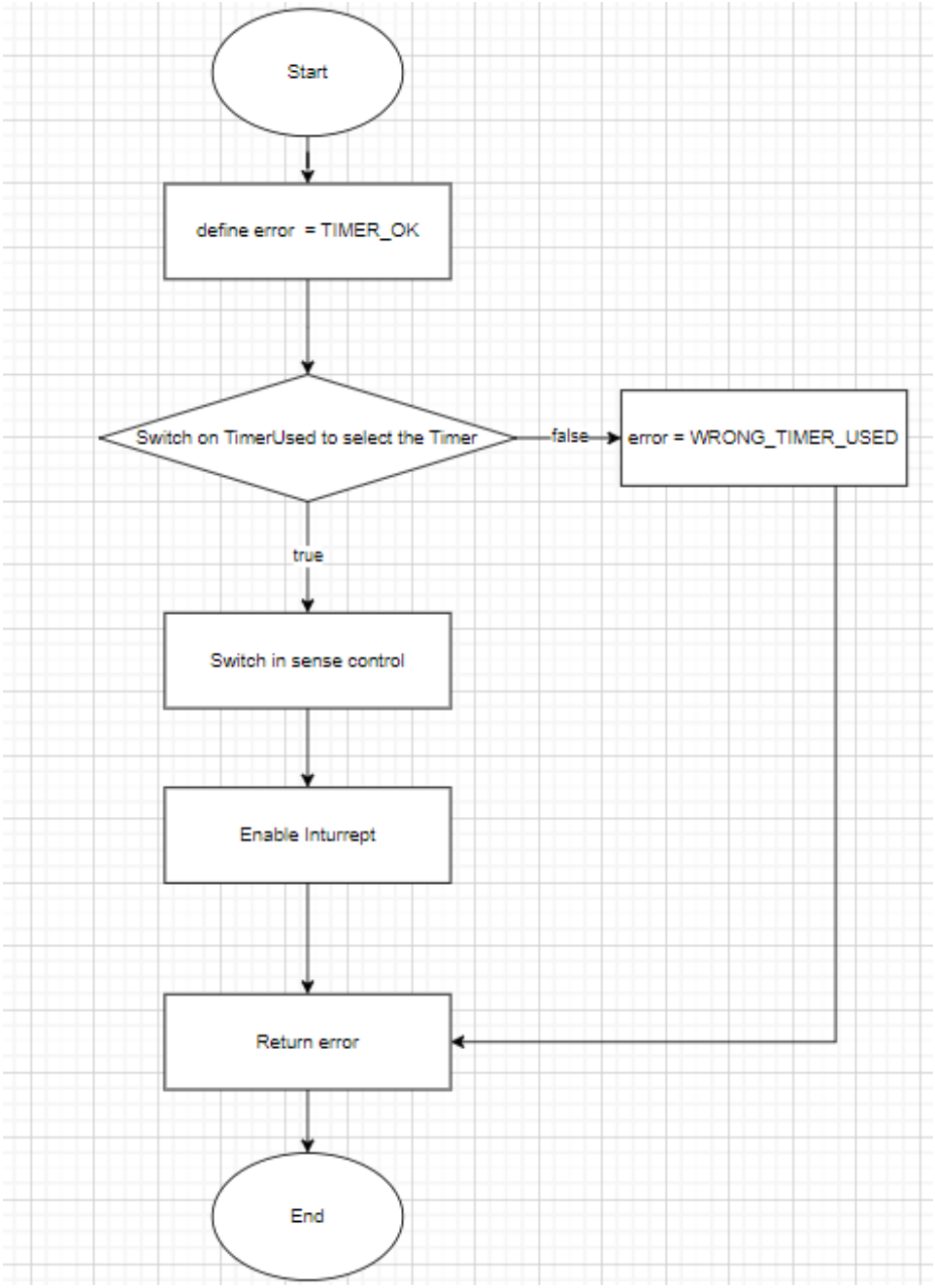
en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);



en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

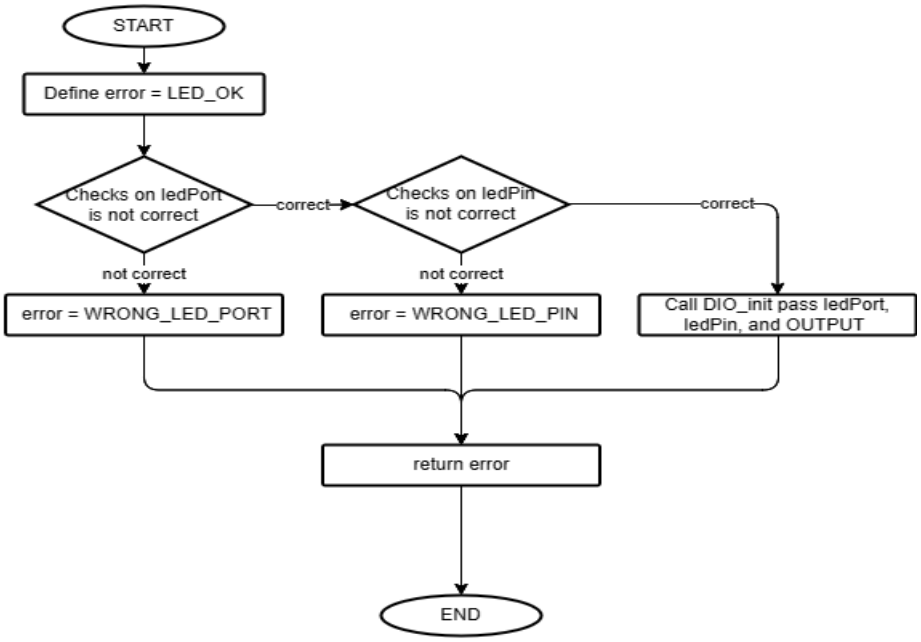


en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

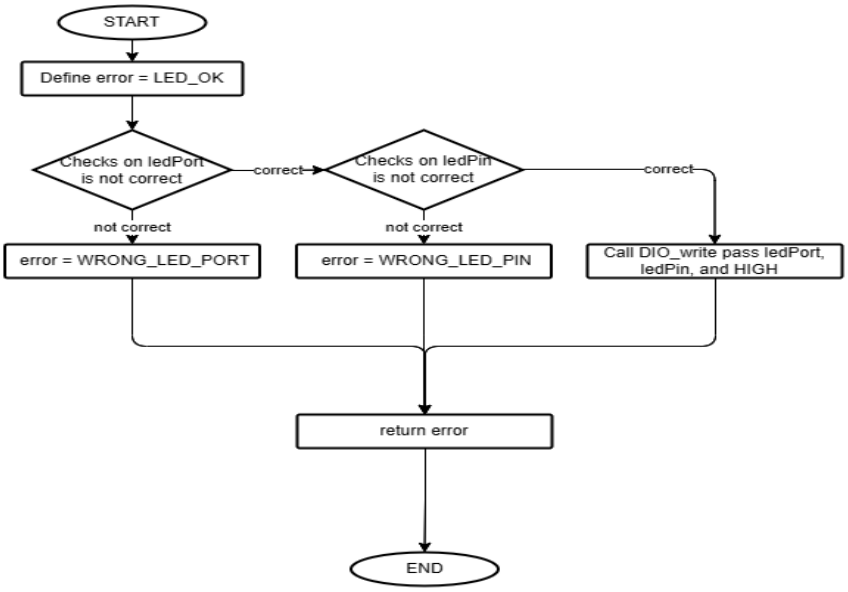


4. LED

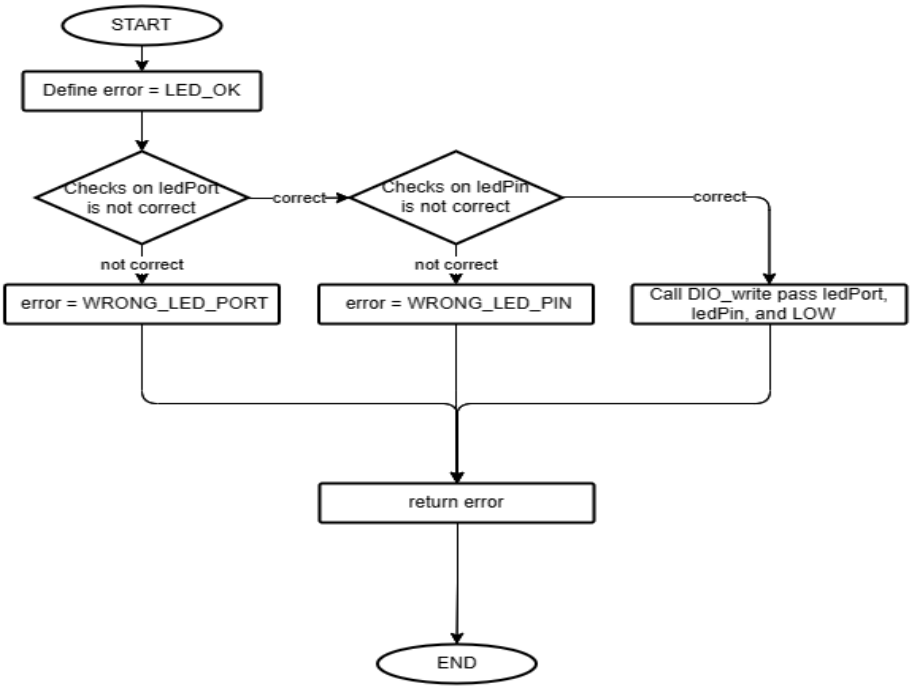
en_ledError_t LED_init(u8 u8_a_ledPort, u8 u8_a_ledPin)



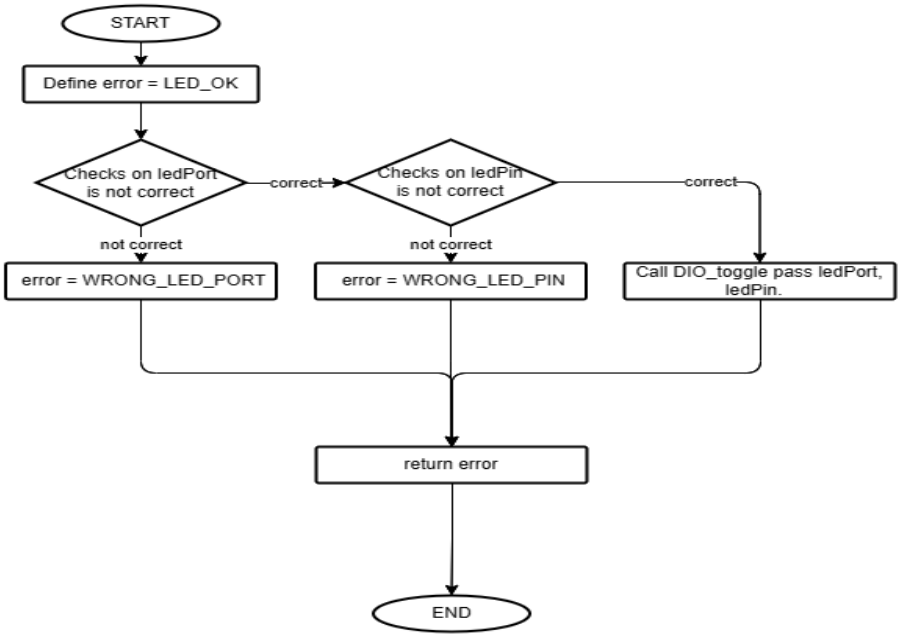
en_ledError_t LED_on(u8 u8_a_ledPort, u8 u8_a_ledPin)



en_ledError_t LED_off(u8 u8_a_ledPort, u8 u8_a_ledPin)

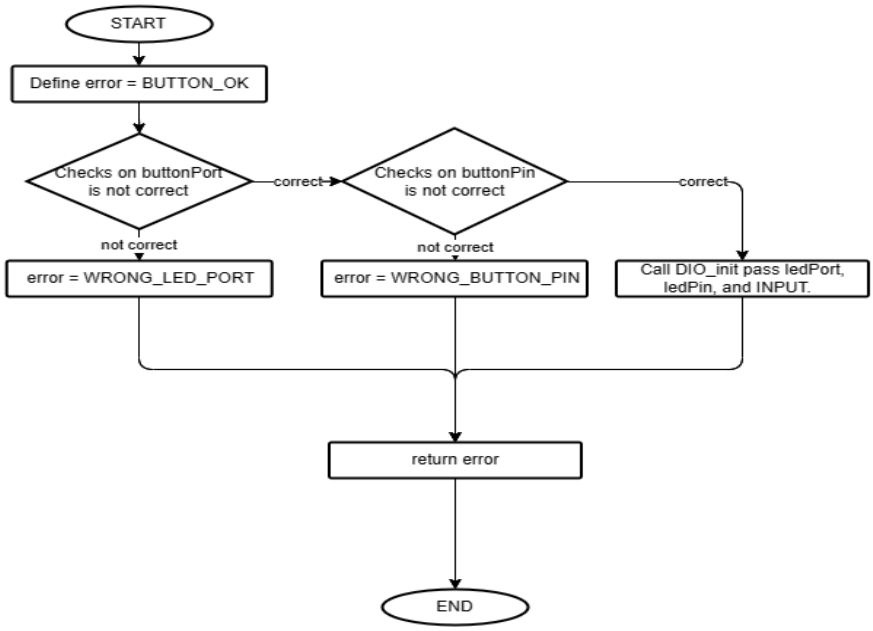


en_ledError_t LED_toggle(u8 u8_a_ledPort, u8 u8_a_ledPin)

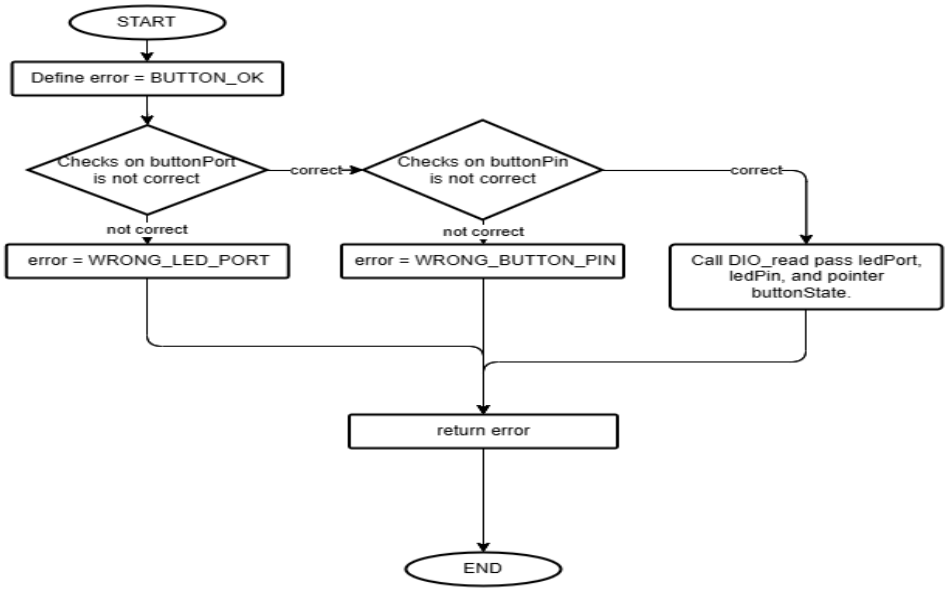


5. BUTTON

en_buttonError_t BUTTON_init(u8 u8_a_buttonPort, u8 u8_a_buttonPin)

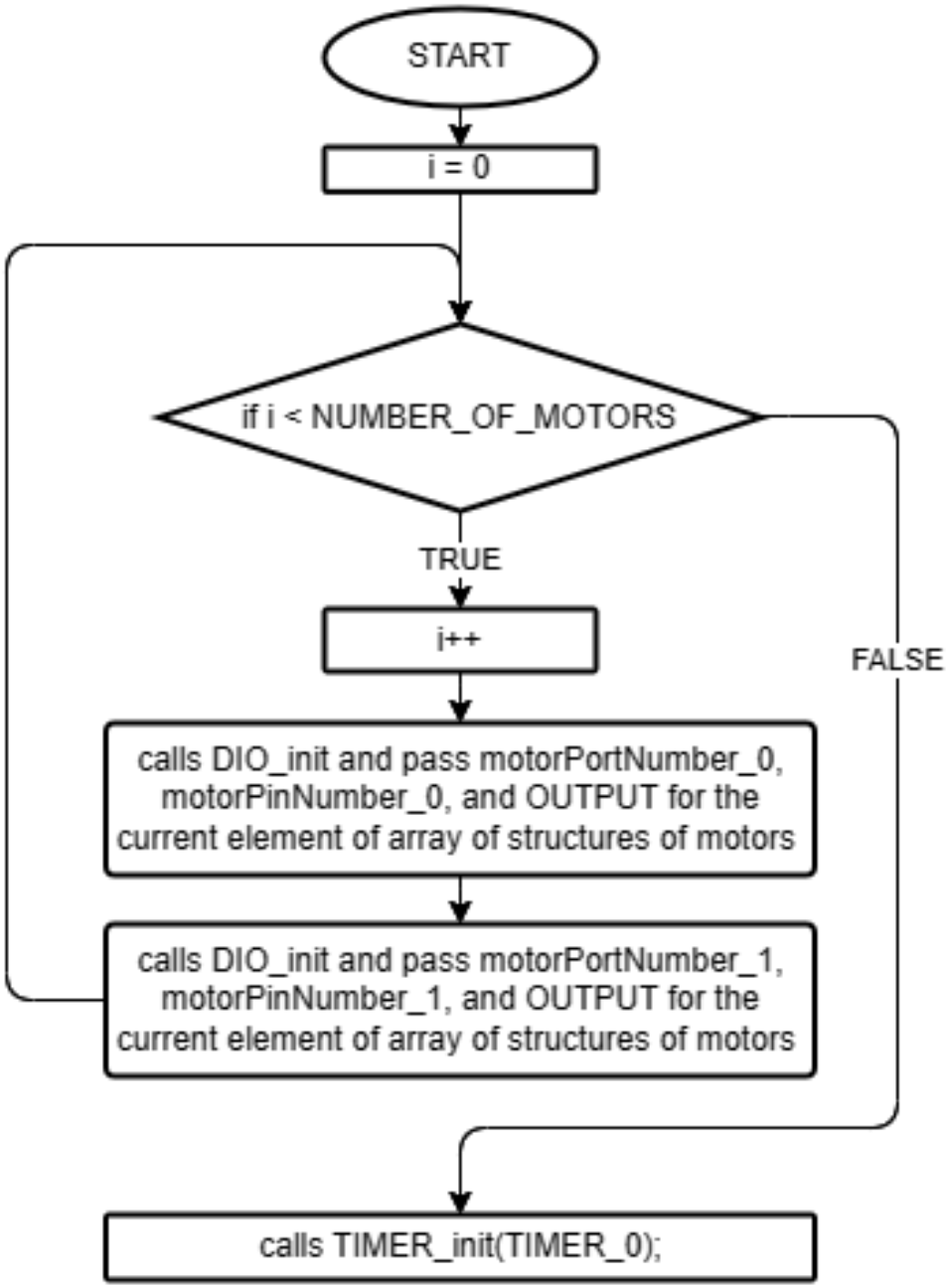


en_buttonError_t BUTTON_read(u8 u8_a_buttonPort, u8 u8_a_buttonPin, u8 *u8_a_buttonState)

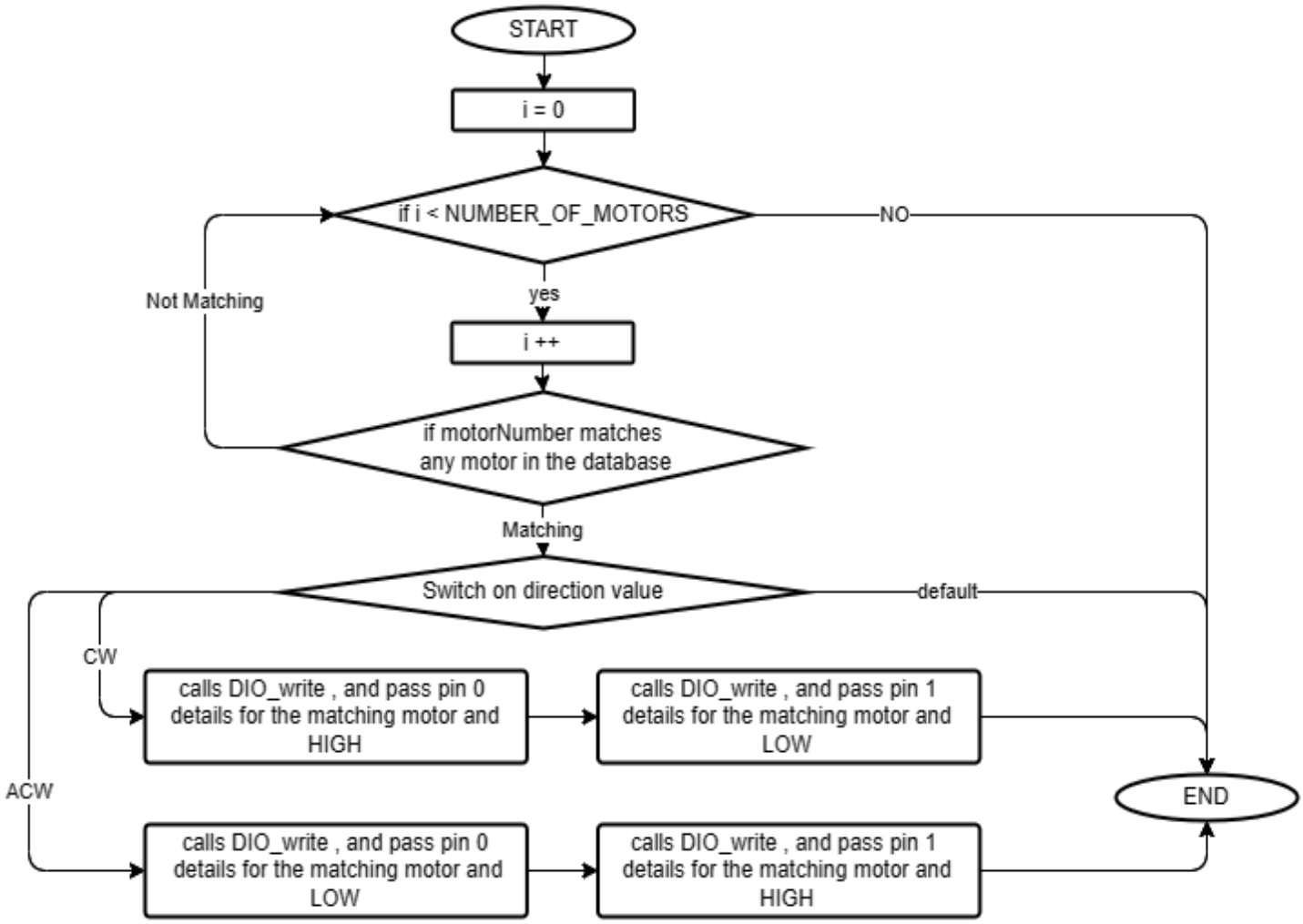


6. Motor

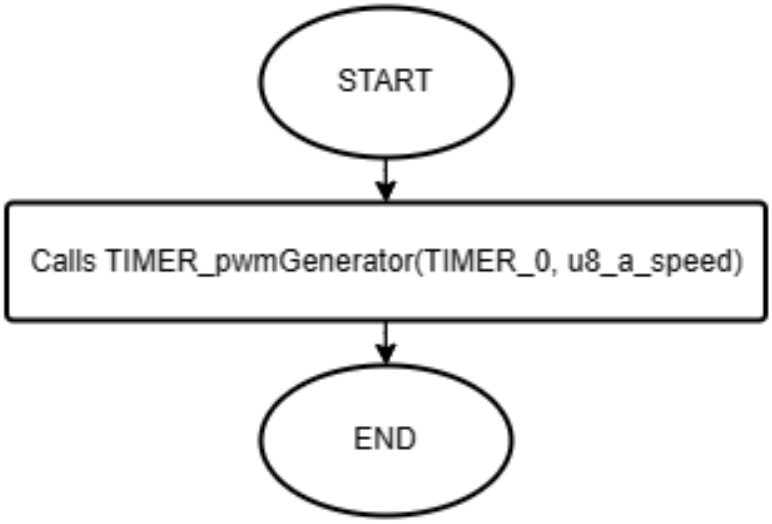
void MOTOR_init (void)



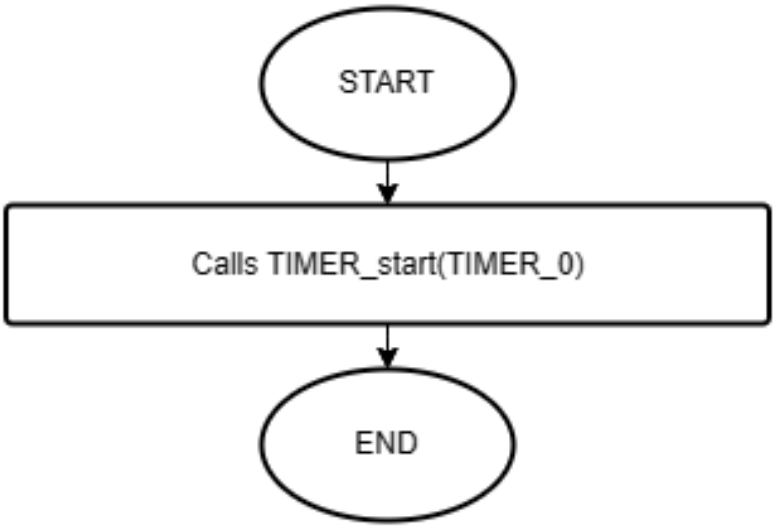
void MOTOR_setDirection (u8 u8_a_motorNumber, u8 u8_a_direction)



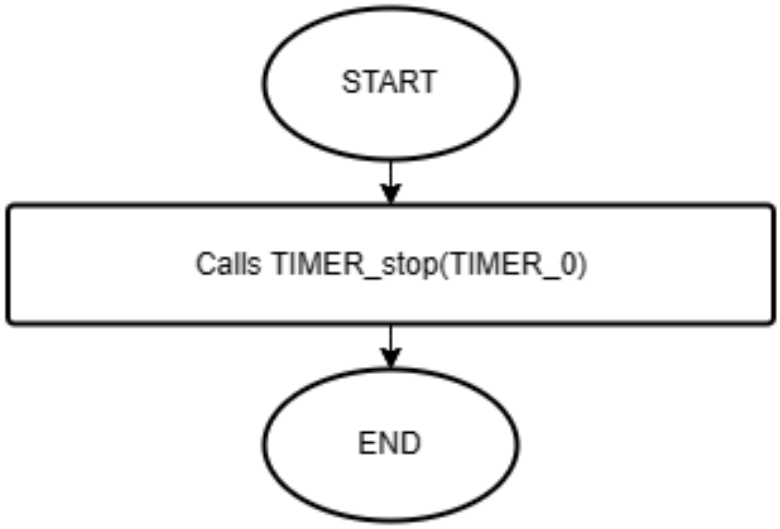
void MOTOR_speed (u8 u8_a_speed)



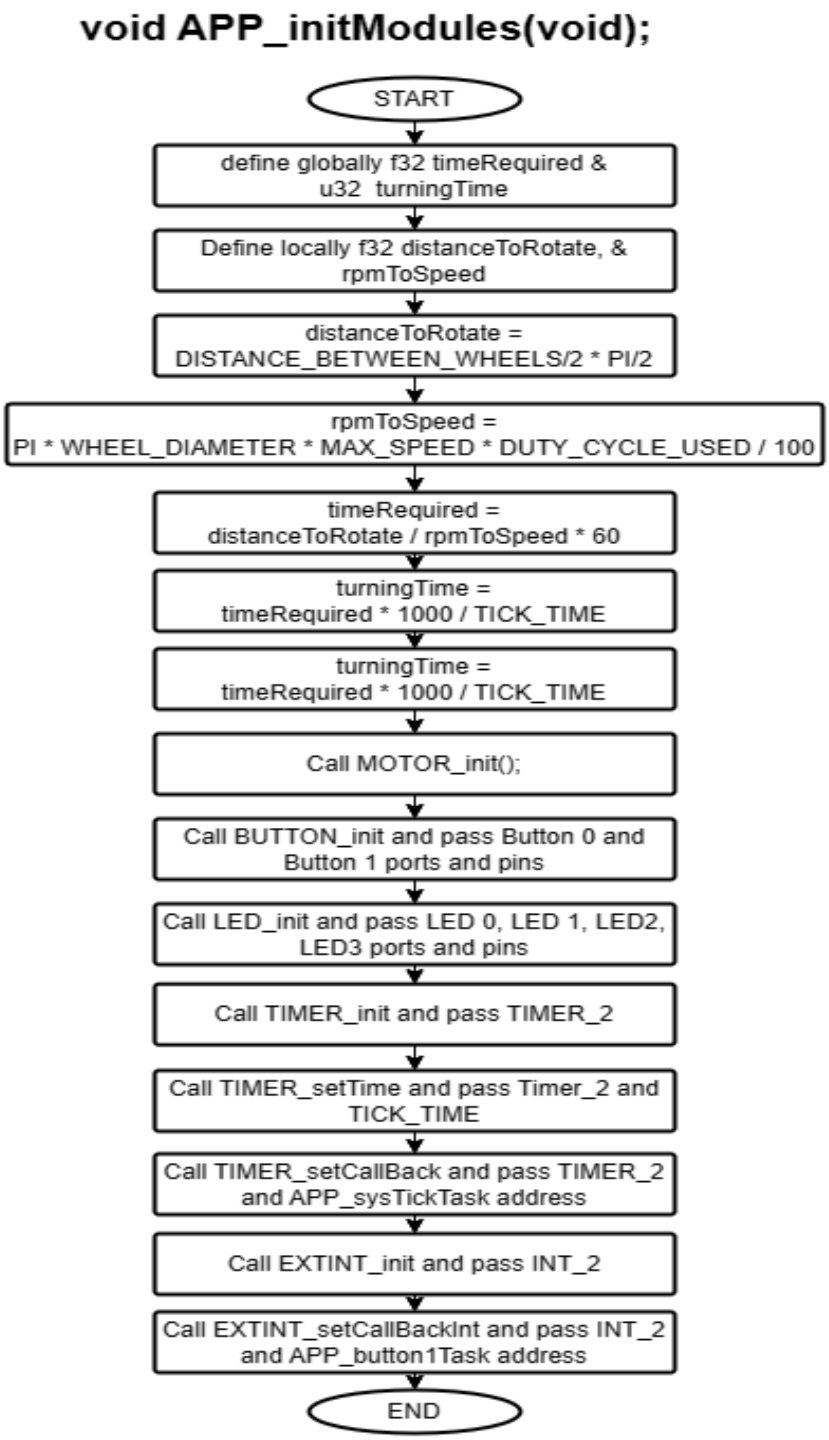
void MOTOR_start (void)



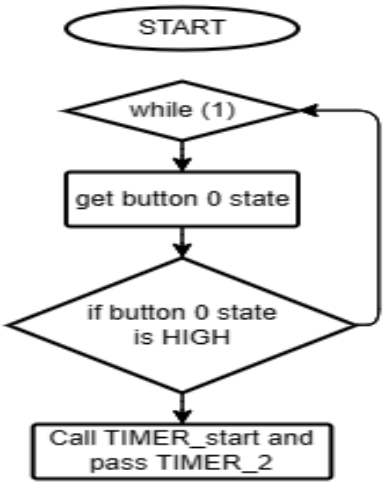
void MOTOR_stop (void)



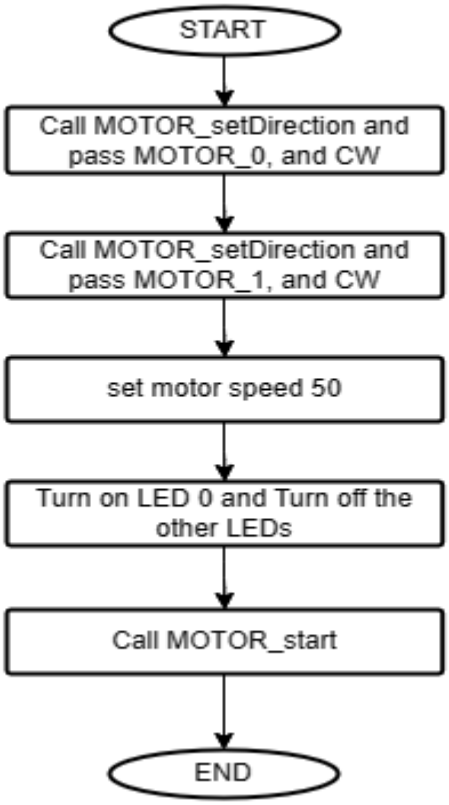
7. APP


void APP_initModules(void);
START
define globally f32 timeRequired &
u32 turningTime
Define locally f32 distanceToRotate, &
rpmToSpeed
distanceToRotate =
DISTANCE_BETWEEN_WHEELS/2 * PI/2
rpmToSpeed =
PI * WHEEL_DIAMETER * MAX_SPEED * DUTY_CYCLE_USED / 100
timeRequired =
distanceToRotate / rpmToSpeed * 60
turningTime =
timeRequired * 1000 / TICK_TIME
turningTime =
timeRequired * 1000 / TICK_TIME
Call MOTOR_init();
Call BUTTON_init and pass Button 0 and
Button 1 ports and pins
Call LED_init and pass LED 0, LED 1, LED2,
LED3 ports and pins
Call TIMER_init and pass TIMER_2
Call TIMER_setTime and pass Timer_2 and
TICK_TIME
Call TIMER_setCallBack and pass TIMER_2
and APP_sysTickTask address
Call EXTINT_init and pass INT_2
Call EXTINT_setCallBackInt and pass INT_2
and APP_button1Task address
END

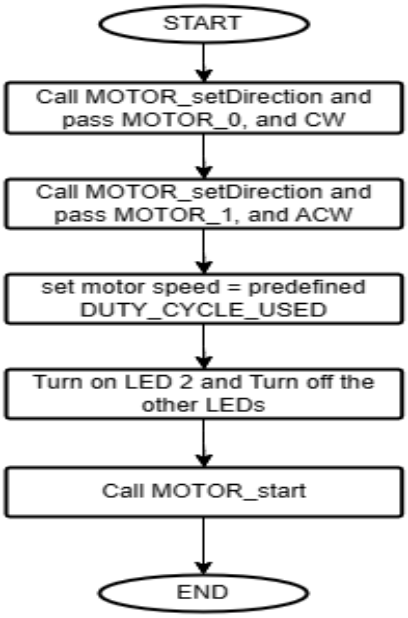
void APP_superLoop (void)



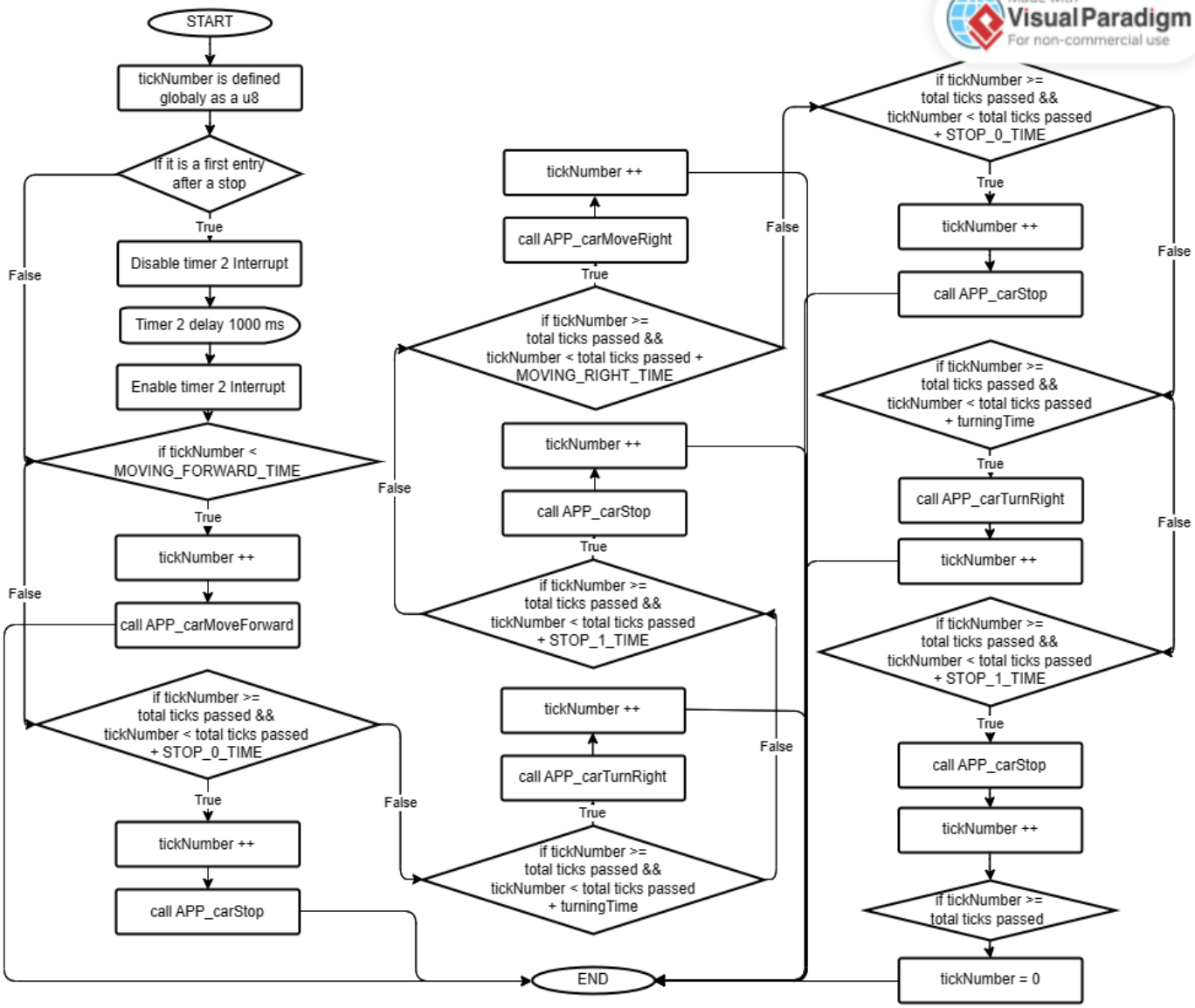
void APP_carMoveForward(void)



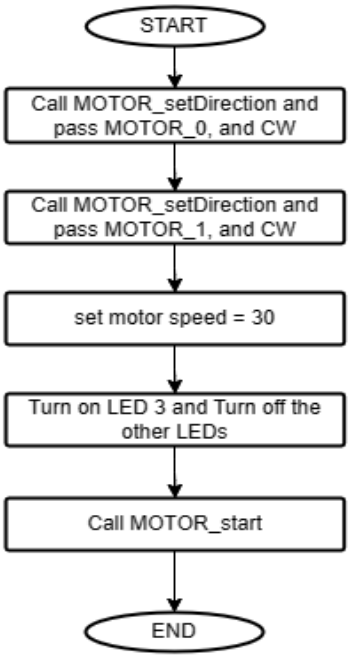
void APP_carTurnRight(void)



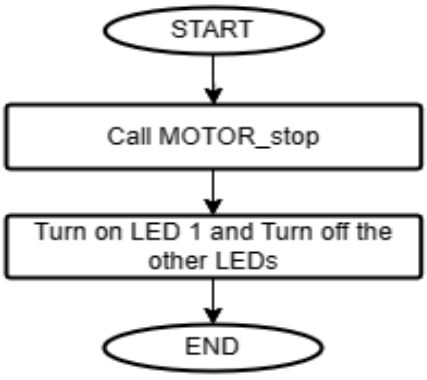
void APP_sysTickTask(void)



void APP_carMoveRight(void)



void APP_stop(void)



void APP_button1Task(void)