MOVING CAR DESIGN

Mina George Fawzy Girgis

Contents

Project Description

In this project we will control a 4WD car to make certain types of requirements. To implement this 4WD car we should use 4 LEDs (**LED1**, **LED2**, **LED3**, **LED4**), 4 Motors (M**1**, **M2**, **M3**, **M4**), and 2 Buttons. One button (PB1) is used to start the 4WD car and follow the required states, while the other button (PB2) is used to immediately stop the car.

Before any button is pressed the car starts initially from 0 speed, When PB1 is pressed, after 1 second the car will start moving forward. It will move to create the longest side of the rectangle for 3 seconds with 50% of the car's maximum speed. After finishing the first longest side the car will stop for 0.5 seconds, and then will rotate 90 degrees to the right. Then it will again stop for 0.5 second and starts moving forward to create the short side of the rectangle at 30% of its speed for 2 seconds. After finishing the shortest side, the car will stop for 0.5 seconds, and again rotate 90 degrees to the right, and stop for 0.5 second to create the shape of rectangle. These steps will be repeated infinitely unless PB2 is pressed which is used as a sudden break and will stop the car from moving. This button has the highest priority meaning that if it's clicked the car should without doubt stop. To make sure this concept is correctly implemented both the concepts of timers and interrupts were implemented in this project. LEDs are used to determine which operation the car is currently doing. The list below briefly describes it.

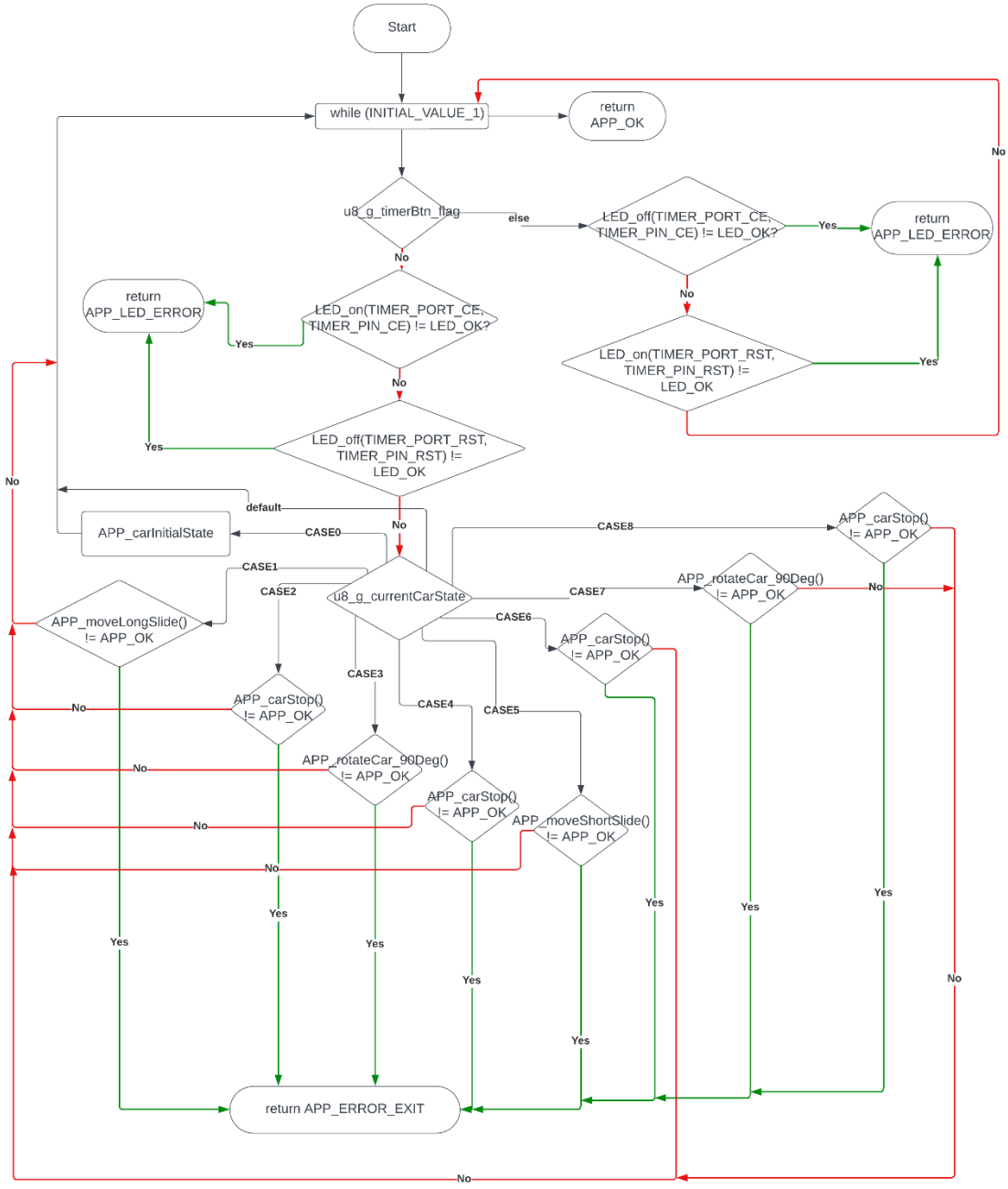
LED1 (BLUE RED): On means moving forward on the long side.

LED2 (YELLOW LED): On means moving forward on the short side.

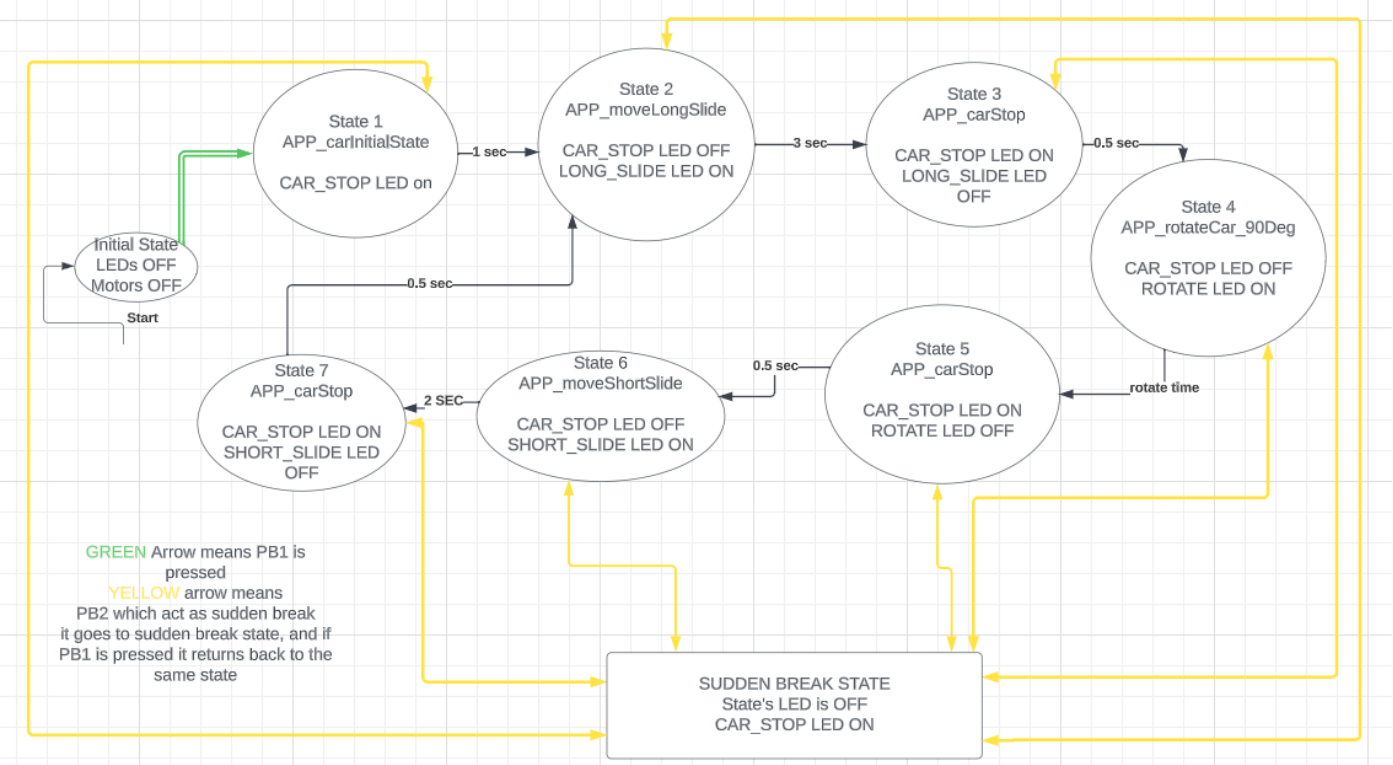
LED3 (GREEN LED): On means stop.

LED4 (RED LED): On means Rotating.

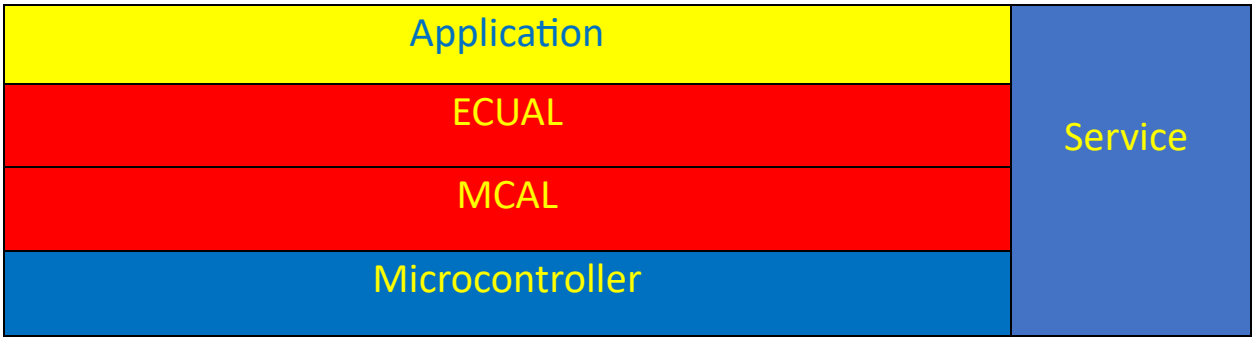
Project Flowchart



Project state machine



Layered Architecture



Application: - it's the topmost layer in a layered architecture and connects users with the lower layers of the architecture.

Service: - It acts as an intermediary between the presentation layer (user interface or application) and the lower layers of the architecture.

MCAL (Microcontroller abstraction layer): - it's created so that if there is any change in the microcontroller layer, I will only change in MCAL as well and not in ECUAL and application.

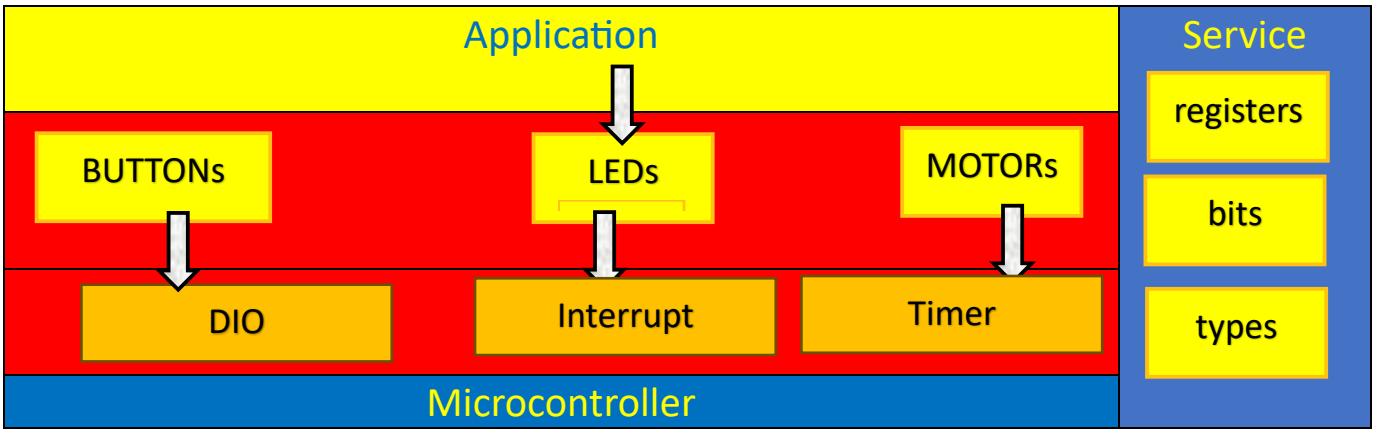
ECUAL (Electronic unit abstraction layer): - it's a layer for the outer components of the microcontroller.

Microcontroller: - the hardware layer is always available in all layered architectures.

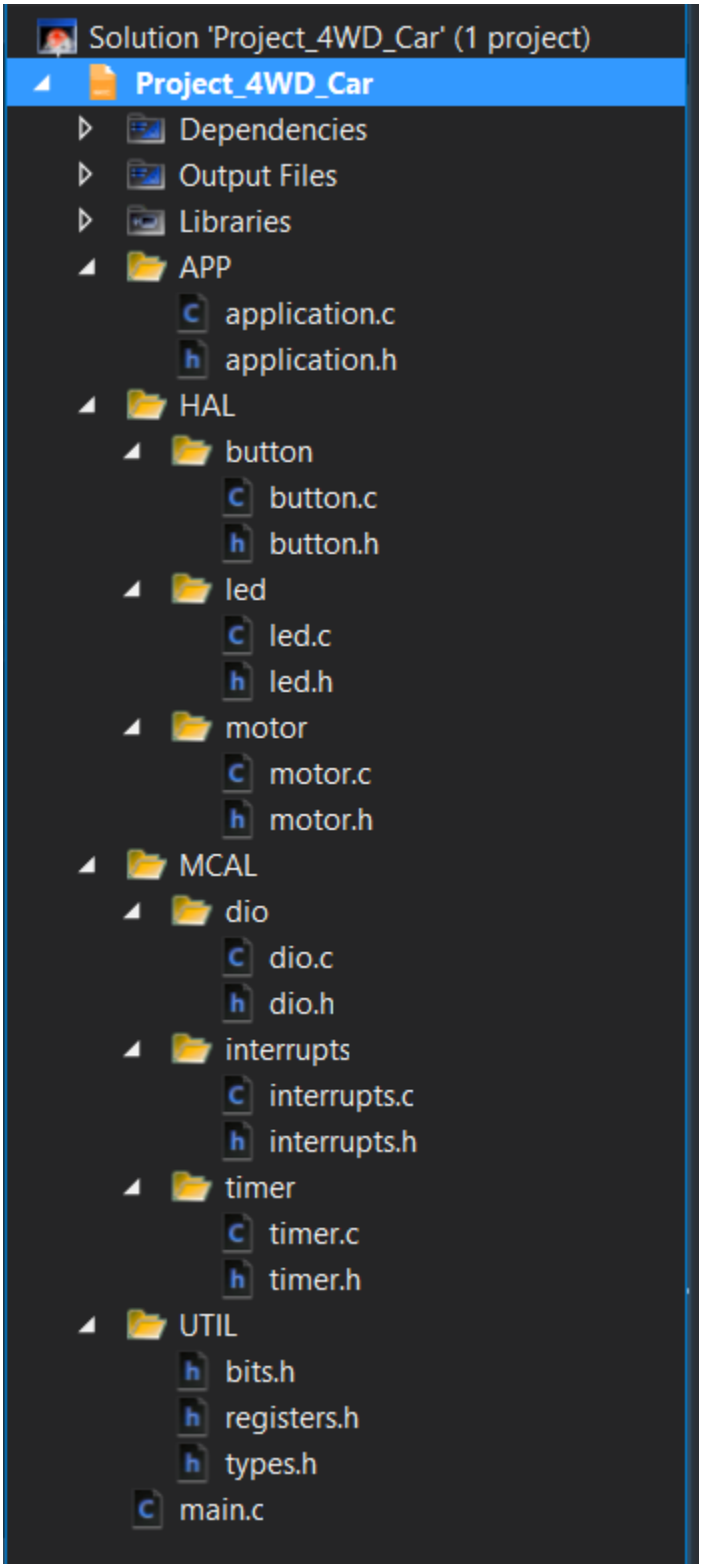
System modules

Modules used:

- 2 Buttons, PB1 to start the motor and simulation, and PB1 to act as sudden break -> ECUAL.
- 4 LEDs (LED0, LED1, LED2, LED3) -> ECUAL
- 4 Motors (**M1**, **M2**, **M3**, **M4)** -> ECUAL
- DIO -> MCAL
- Interrupt -> MCAL
- Timer -> MCAL
- Registers -> Service
- Types -> Service
- Bits -> Service
- Application to simulate concept.



APIs


Solution 'Project_4WD_Car' (1 project)
Project_4WD_Car
Dependencies
Output Files
Libraries
APP
application.c
application.h
HAL
button
button.c
button.h
led
led.c
led.h
motor
motor.c
motor.h
MCAL
dio
dio.c
dio.h
interrupts
interrupts.c
interrupts.h
timer
timer.c
timer.h
UTIL
bits.h
registers.h
types.h
main.c

DIO APIs

```c
typedef enum EN_dioError_t{
    DIO_OK, DIO_INVALID_PORT, DIO_INVALID_PIN, DIO_INVALID_INIT,DIO_INVALID_VALUE
}EN_dioError_t;



EN_dioError_t  DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction); // initilaize dio direction
EN_dioError_t  DIO_write(uint8_t portNumber,uint8_t pinNumber,uint8_t value); // write data to dio
EN_dioError_t  DIO_toggle(uint8_t portNumber,uint8_t pinNumber); // toggle dio
EN_dioError_t  DIO_read(uint8_t portNumber,uint8_t pinNumber,uint8_t* value);// read dio
```

DIO_INIT: this function takes 3 parameters portNumber (A or B or C or D), pinNumber (0-7) and direction.

It returns either `DIO_INVALID_PIN` when invalid pin enters (<0 or >7), returns `DIO_INVALID_INIT` when initializing invalid (direction not OUT or IN), returns `DIO_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `DIO_OK` when valid initializing.

DIO_write: this function takes 3 parameters portNumber (A or B or C or D), pinNumber (0-7) and value.

It returns either `DIO_INVALID_PIN` when invalid pin enters (<0 or >7), returns `DIO_INVALID_VALUE` when invalid value entered (value not HIGH or LOW), returns `DIO_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `DIO_OK` when valid write.

DIO_read: this function takes 3 parameters, portNumber (A or B or C or D), pinNumber (0-7) and pointer value (so that it can read and store the value of DIO state).

It returns either `DIO_INVALID_PIN` when invalid pin enters (<0 or >7), returns `DIO_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `DIO_OK` when valid read.

DIO_toggle: this function takes 2 parameters portNumber (A or B or C or D) and pinNumber (0-7)

It returns either `DIO_INVALID_PIN` when invalid pin enters (<0 or >7), returns `DIO_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `DIO_OK` when valid toggle.

Interrupts APIs

```c
// Setting global interrupts - setting bit 7 in the status register 1
#define sei() __asm__ __volatile__("sei"::: "memory")

// clear global interrupts, set the I-bit in status register 0
#define cli() __asm__ __volatile__("cli"::: "memory")

//ISR FUNCTION
#define ISR(INT_VECT) void INT_VECT(void) __attribute__((signal, used));\
void INT_VECT(void)

//enabling external interrupts
EN_InterruptError_t INTERRUPTS_enableExternalInterrupts(uint8_t Interrupt);

// Enable global interrupts
void INTERRUPTS_enableGlobalInterrupt(void);

// Disable global interrupts
void INTERRUPTS_disableGlobalInterrupt(void);

/* Choose the external interrupt sense - sense on rising edge or negative edge */
EN_InterruptError_t INTERRUPTS_extInterruptsEdge(uint8_t Interrupt, uint8_t edge);

//Set callbacks functions
EN_InterruptError_t INTERRUPTS_EXT_SetCallBack(uint8_t Interrupt, void( * LocalPtr)(void));

//initialize LEDS
void INTERRUPTS_leds_init(void);
```

SEI: this function is used to set the global interrupts (which means setting the 7th bit in the status register with 1)

CLI: this function is used to disable (clear) the global interrupts (which means setting the I-bit in the status register with 0)

ISR (Interrupt Service Routine): this function takes void and return void. It must contain small logic and is not called by the software. It also mustn't be optimized by the compiler. It examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value.

INTERRUPTS_enableGlobalInterrupt: this function is used to enable global interrupts by calling SEI function.

INTERRUPTS_disableGlobalInterrupt: this function is used to disable the global interrupts by calling CLI function.

INTERRUPTS_enableExternalInterrupts: this function takes 1 parameter called u8_a_interrupt and is used to enable external the interrupt according to the value of the parameter, if it's EXTERNAL_INTERRUPT_0 it will enable interrupt 0 – INTO by by setting the 7th bit in the GICR register to 1. (GICR |= (1<<6) ;) if it's EXTERNAL_INTERRUPT_1 it will enable interrupt 1 – INTO by by setting the 6th bit in the GICR register to 1. (GICR |= (1<<7). if it's EXTERNAL_INTERRUPT_2

it will enable interrupt 2 – INTO by by setting the 5th bit in the GICR register to 1. (GICR |= (1<<5). Else it will return `INTERRUPT_INVALID_ENABLE.`

INTERRUPTS_extInterruptsEdge: this function takes 2 perimeters (u8_a_interrupt, and u8_a_edge), u8_a_interrupt is used to determine which External interrupt will be adjusted, while u8_a_edge is used to set the external interrupt sense. (Low level, logical change, falling edge, rising edge).

INTERRUPTS_EXT_SetCallBack: this function takes 2 perimeter (u8_a_interrupt, void(*localPtrl)(void) they're used to decide which external interrupt the local ptr function will set the call back to. (the set call back function is used to make the last code can get back)

INTERRUPTS_leds_init: this function is used to initialize LEDs that will be used in ISR function.

Timer APIs

```c
//Set timer0 mode (0 normal, 1 CTC, 2 PWM, 3 FastPWM)
EN_TimerError_t TIMER_timer0InitMode(uint8_t timer1_mode);

/*Timer0 pres calling (TIMER0_STOP 0, TIMER0_NO_PRESCALLING 1, TIMER0_SCALER_8 2, TIMER0_SCALER_64 3,
TIMER0_SCALER_256 4, TIMER0_SCALER_1024 5, EXTERNAL_FALLING_EDGE 6, EXTERNAL_RISING_EDGE 7 */
EN_TimerError_t TIMER_timer0Prescalar(uint8_t u8_a_prescalar);

//this functions is used to set the delay for the timer0
EN_TimerError_t TIMER_timer0Delay(float64 timeDelay);

//timer0 stop
void TIMER_timer0Stop(void);

//overflow flag clear
void TIMER_clrOF_Flag(void);

//Timer0 PWM mode
EN_TimerError_t TIMER_timer0PWM_mode(float32 pwmValue);

//Timer0 number of overflows we want to reach.
EN_TimerError_t TIMER_timer0_OVF_Number(float64 OVFNumber);

// Timer0 set initial value
EN_TimerError_t TIMER_timer0SetInitial(float64 intial_value);

/****************** TIMER2 FUNCTIONS ******************************/

//Set timer2 mode (0 normal, 1 CTC, 2 PWM, 3 FastPWM)
EN_TimerError_t TIMER_timer2InitMode(uint8_t timer2_mode);

/*Timer2 pres calling (TIMER2_STOP 0, TIMER2_NO_PRESCALLING 1, TIMER2_SCALER_8 2, TIMER2_SCALER_32 3,
TIMER2_SCALER_64 4, TIMER2_SCALER_128 5, TIMER2_SCALER_256 6, TIMER2_SCALER_1024 7 */
EN_TimerError_t TIMER_timer2Prescalar(uint8_t time2_prescalar);

// Timer2 stop
void TIMER_timer2Stop(void);

// Timer2 set initial value
EN_TimerError_t TIMER_timer2SetInitial(uint16_t intial_value);

//This function is used to Initialize timer2
EN_TimerError_t TIMER_timer2Init(void);
```

TIMER_timer0InitMode: this function takes 1 parameter mode (0 for normal mode, 1 for CTC, 2 for PWM, 3 for fast PWM). It's used to decide what is the mode for the timer.

It returns either TIMER_INVALID_MODE when invalid mode entered and returns TIMER_OK when valid.

TIMER_timer0Prescalar: this function takes 1 parameter prescalar (TIMER0_STOP 0, TIMER0_NO_PRESCALLING 1, TIMER0_SCALER_8 2, TIMER0_SCALER_64 3, TIMER0_SCALER_256 4, TIMER0_SCALER_1024 5, EXTERNAL_FALLING_EDGE 6, EXTERNAL_RISING_EDGE 7). It's used to decide what is the prescalar of the timer.
It returns either TIMER_INVALID_PRESCALAR when invalid prescalar value is entered and returns TIMER_OK when valid.

TIMER_timer0Delay: this function takes 1 parameter timeDelay. It's used to set the time delay for the timer0.
It returns either TIMER_INVALID_DELAY when invalid delay entered and returns TIMER_OK when valid.

TIMER_timer0Stop: this function is used to stop the timer. By setting the bits of timer0 (TCCR0= 0x00;)
TIMER_clrOF_Flag: this function is used to clear the overflow flag of timer. (TIFR |= (1<<0) ;)

TIMER_timer0PWM_mode: this function takes 1 parameter, and is used to set the Timer0 in PWM mode by using certain types of calculations.

TIMER_timer0_OVF_Number: this function takes 1 parameter and is used to set the number of overflows we want to reach to timer0 using certain type of calculations.

TIMER_timer0SetInitial: this function is used to set the initial value of timer0.

TIMER_timer2InitMode: this function takes 1 parameter mode (0 for normal mode, 1 for CTC, 2 for PWM, 3 for fast PWM). It's used to decide what is the mode for the timer2.

It returns either TIMER_INVALID_MODE when invalid mode entered and returns TIMER_OK when valid.

TIMER_timer2Prescalar: this function takes 1 parameter prescalar (TIMER2_STOP 0, TIMER2_NO_PRESCALLING 1, TIMER2_SCALER_8 2, TIMER2_SCALER_32 3, TIMER2_SCALER_64 4, TIMER2_SCALER_128 5, TIMER2_SCALER_256 6, TIMER2_SCALER_1024 7). It's used to decide what is the prescalar of the timer0.
It returns either TIMER_INVALID_PRESCALAR when invalid prescalar value is entered and returns TIMER_OK when valid.

TIMER_timer2Stop: this function is used to stop the timer. By setting the bits of timer2 (TCCR2= 0x00;)

TIMER_timer2SetInitial: this function is used to set the initial value of timer2.

TIMER_timer2Init: this function is used to initlaize timer2 by calling the TIMER_timer2InitMode, TIMER_timer2SetInitial, and TIMER_timer2Prescalar functions after making sure they return the correct variables. This function is also used in controlling the button flag.

BUTTON APIs

```
typedef enum EN_btnError_t{
    BTN_OK,BTN_INVALID_PORT, BTN_INVALID_PIN, BTN_INVALID_INIT,BTN_INVALID_VALUE
}EN_btnError_t;

EN_btnError_t BUTTON_init(uint8_t buttonPort,uint8_t buttonPin);//initialising a button
EN_btnError_t BUTTON_read(uint8_t buttonPort,uint8_t buttonPin,uint8_t* value);//reading the value of a button
```

BUTTON_INIT: this function takes 2 parameters buttonPort (A or B or C or D) and buttonPin (0-7) It returns either `BTN_INVALID_PIN` when invalid pin enters (<0 or >7), returns `BTN_INVALID_INIT` when initializing invalid (direction not OUT or IN), returns `BTN_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `BTN_OK` when valid initializing.

BUTTON_READ: this function takes 3 parameters, buttonPort (A or B or C or D), buttonPin (0-7) and pointer value (so that it can read and store the value of the button).

It returns either `BTN_INVALID_PIN` when invalid pin enters (<0 or >7), returns `BTN_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `BTN_OK` when valid read.

LEDS APIs

```
typedef enum EN_ledError_t{
    LED_OK,LED_INVALID_PORT, LED_INVALID_PIN, LED_INVALID_INIT,LED_INVALID_VALUE
}EN_ledError_t;


EN_ledError_t LED_init(uint8_t ledPort,uint8_t ledPin);//initialising a LED
EN_ledError_t LED_on(uint8_t ledPort,uint8_t ledPin);//turning a led on
EN_ledError_t LED_off(uint8_t ledPort,uint8_t ledPin);//turning off a led
EN_ledError_t LED_toggle(uint8_t ledPort,uint8_t ledPin);//switching an on led to off or vice versa
EN_ledError_t LED_read(uint8_t ledPort,uint8_t ledPin,uint8_t* value); //read LED Status
```

LED_INIT: this function takes 2 parameters ledPort (A or B or C or D) and ledPin (0-7).

It returns either `LED_INVALID_PIN` when invalid pin enters (<0 or >7), returns `LED_INVALID_INIT` when initializing invalid (direction not OUT or IN), returns `LED_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `LED_OK` when valid initializing.

LED_on: this function takes 2 parameters ledPort (A or B or C or D) and ledPin (0-7).

It returns either `LED_INVALID_PIN` when invalid pin enters (<0 or >7), returns `LED_INVALID_VALUE` when invalid value entered (value not HIGH or LOW), returns `LED_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `LED_OK` when valid on.

LED_off: this function takes 2 parameters ledPort (A or B or C or D) and ledPin (0-7).

It returns either `LED_INVALID_PIN` when invalid pin enters (<0 or >7), returns `LED_INVALID_VALUE` when invalid value entered (value not HIGH or LOW), returns `LED_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `LED_OK` when valid off.

LED_toggle: this function takes 2 parameters ledPort (A or B or C or D) and ledPin (0-7) and It returns either `LED_INVALID_PIN` when invalid pin enters (<0 or >7), returns `LED_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `LED_OK` when valid toggle.

LED_READ: this function takes 3 parameters, ledPort (A or B or C or D), ledPin (0-7) and pointer value (so that it can read and store the value of the led).

It returns either `LED_INVALID_PIN` when invalid pin enters (<0 or >7), returns `LED_INVALID_PORT` when port is not valid (not A or B or C or D), and it returns `LED_OK` when valid read.

MOTOR APIs

```
typedef enum EN_motorError_t {
    MOTOR_OK,
    MOTOR_INVALID_INIT,
    MOTOR_INVALID_MOVEFWD,
    MOTOR_INVALID_ROTATE,
    MOTOR_INVALID_STOP
}
EN_motorError_t;

EN_motorError_t MOTOR_init(void);
EN_motorError_t MOTOR_moveForward(void);
EN_motorError_t MOTOR_rotate(void);
void MOTOR_stop(void);
```

MOTOR_INIT: this function is used to initialize motors. It returns `MOTOR_INVALID_INIT` when either invalid pin enters (<0 or >7), or invalid direction entered, or when invalid (not A or B or C or D) port is entered, and it returns `MOTOR_OK` when valid initializing.

MOTOR_moveForward: this function is used to make motors start and move forward. It returns `MOTOR_INVALID_MOVEFWD` when either invalid pin enters (<0 or >7), or invalid value entered, or when invalid (not A or B or C or D) port is entered, and it returns `MOTOR_OK` when valid.

MOTOR_rotate: this function is used to make motors rotate. It returns `MOTOR_INVALID_ROTATE` when either invalid pin enters (<0 or >7), or invalid value entered, or when invalid (not A or B or C or D) port is entered, and it returns `MOTOR_OK` when valid.

MOTOR_stop: this function is used to make the motor stop by writing their value to low.

Application APIs

```
typedef enum EN_appError_t {
    APP_OK,
    APP_LED_ERROR,
    APP_BTN_ERROR,
    APP_INT_ERROR,
    APP_ERROR_EXIT,
    APP_MOTOR_ERROR,
    APP_TIMER_ERROR,
    APP_INTERRUPT_ERROR
}
EN_appError_t;

EN_appError_t APP_init(void);
EN_appError_t APP_start(void);

EN_appError_t APP_moveShortSlide(void);
EN_appError_t APP_moveLongSlide(void);
EN_appError_t APP_rotateCar_90Deg(void);
EN_appError_t APP_carStop(void);
EN_appError_t APP_pwmMode(float32 pwmValue);
void APP_carInitialState(void);
```

APP_INIT: this function doesn't take any parameters (Void). It's used to initialize all buttons, LEDs, motors used. It's also used to enable interrupts and alter interrupts and set callback functions. It returns either `APP_BTN_ERROR` when invalid button initializing, returns `APP_LED_ERROR` when invalid LED initializing, return `APP_MOTOR_EXIT` when invalid motor initialize, returns `APP_INTERRUPT_EXIT` when invalid enabling interrupt and it returns `APP_OK` when valid initializing.

APP_START: this function doesn't take any parameters (Void). It returns either `APP_LED_ERROR` when invalid LED toggle (On or Off), returns `APP_ERROR_EXIT` when invalid any of the car states are invalid, and it returns `APP_OK` when valid process. This function is used to provide the main concept/system requirement of the project which was described in project description.

APP_moveShortSlide: this function is used to make the car move forward for the short slide. It will create the short side of the rectangle at 30% of its speed using the APP_pwmMode function

and the MOTOR_moveFWD function, for 2 seconds. The LED of short slide is handled and turned on as well in this function. This function makes sure it follows the interrupt and timer concept. It will return APP_OK if everything is correctly implemented.

APP_moveLongSlide: this function is used to make the car move forward for the long slide. It will create the long side of the rectangle at 50% of its speed using the APP_pwmMode function and the MOTOR_moveFWD function, for 3 seconds. The LED of long slide is handled and turned on as well in this function. This function makes sure it follows the interrupt and timer concept. It will return APP_OK if everything is correctly implemented.

APP_rotateCar_90Deg: this function is used to make the car rotate. It will rotate the car using some calculation involving the car's maximum speed, the diameter, and the distance between wheels. It will also use both the APP_pwmMode function and the MOTOR_rotate function. The rotate' LED is handled and turned on as well in this function. This function makes sure it follows the interrupt and timer concept. It will return APP_OK if everything is correctly implemented.

APP_carStop: this function is used to make the car stop. It will stop the car using the MOTOR_stop function, for 0.5 sec or if PB2 is pressed which is sudden break. The car stop' LED is handled and turned on as well in this function. This function makes sure it follows the interrupt and timer concept. It will return APP_OK if everything is correctly implemented.

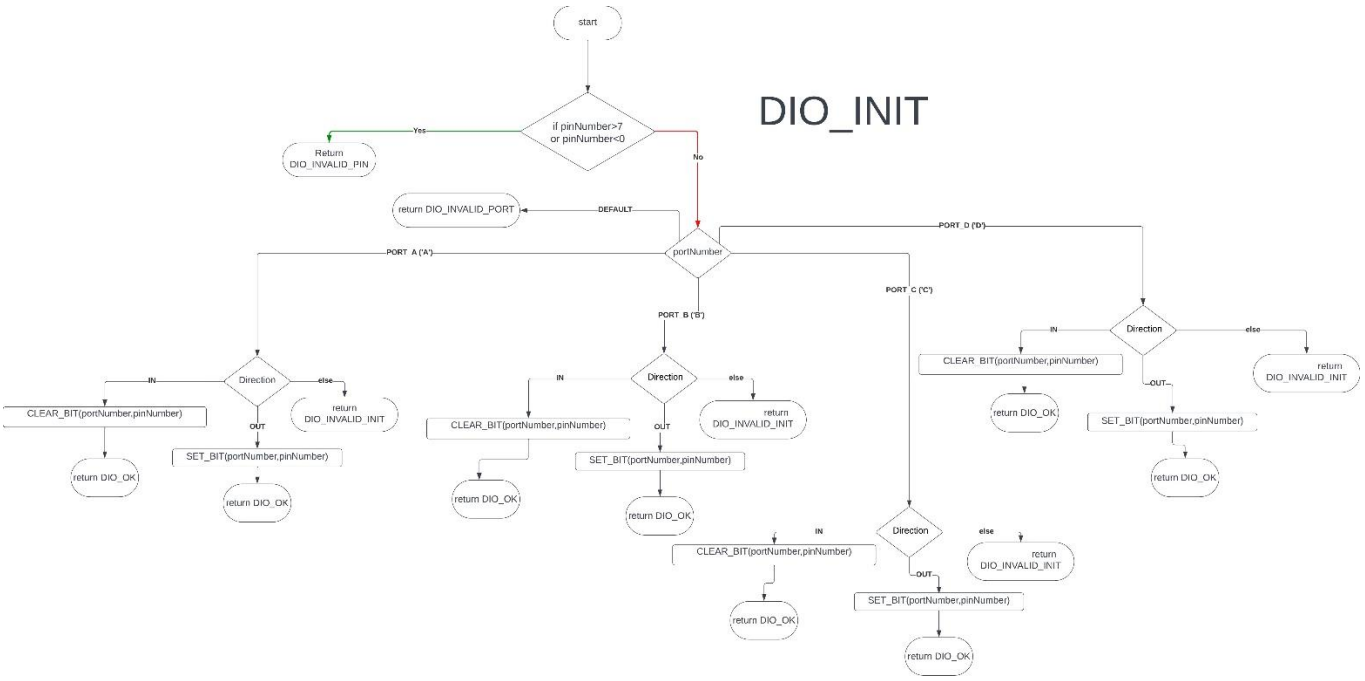
APP_pwmMode: this function is used for handling the configuration and control of a PWM signal of the timer. It calculates timer values based on a given f32_a_pwmValue parameter and checks the success of various timer and LED-related operations, returning different error codes (APP_TIMER_ERROR and APP_LED_ERROR) if any of these operations fail. If everything is successful, it returns APP_OK.

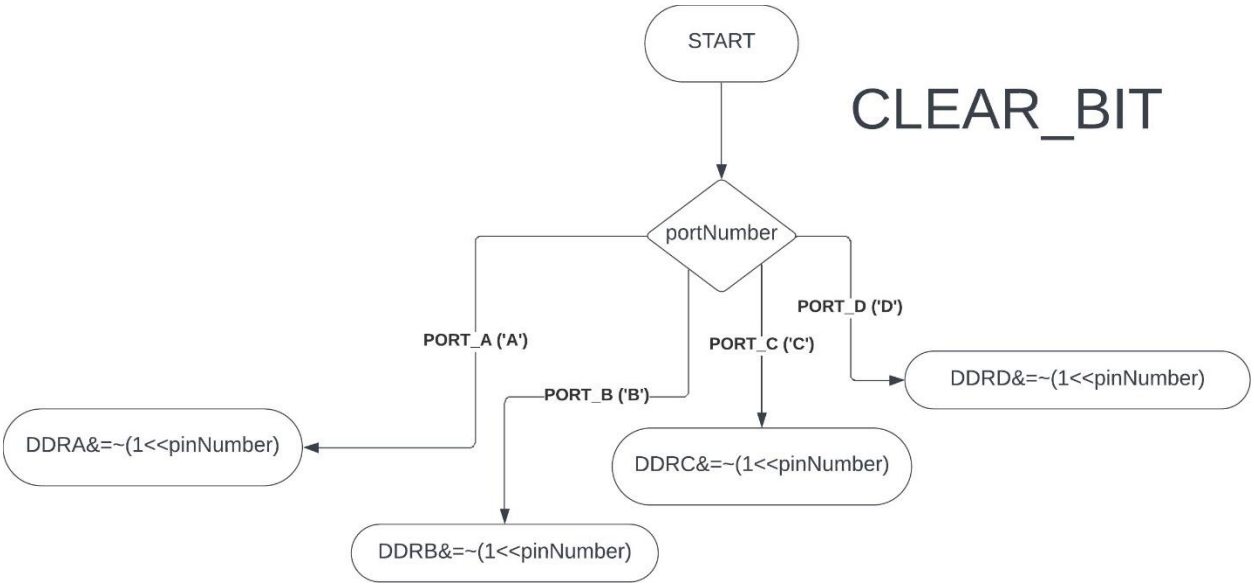
APP_carInitialState: this function is the initial state of the car. When PB1 is pressed It will stop the car using the MOTOR_stop function, for 1 sec. This function makes sure it follows the interrupt and timer concept.

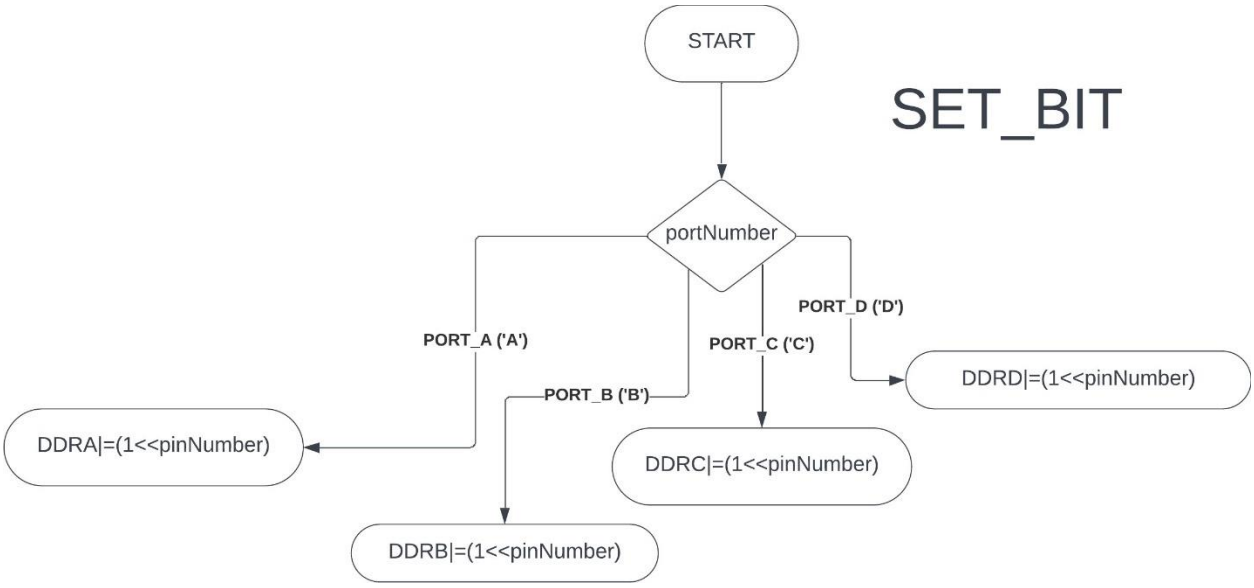
‘

DIO APIs flowchart

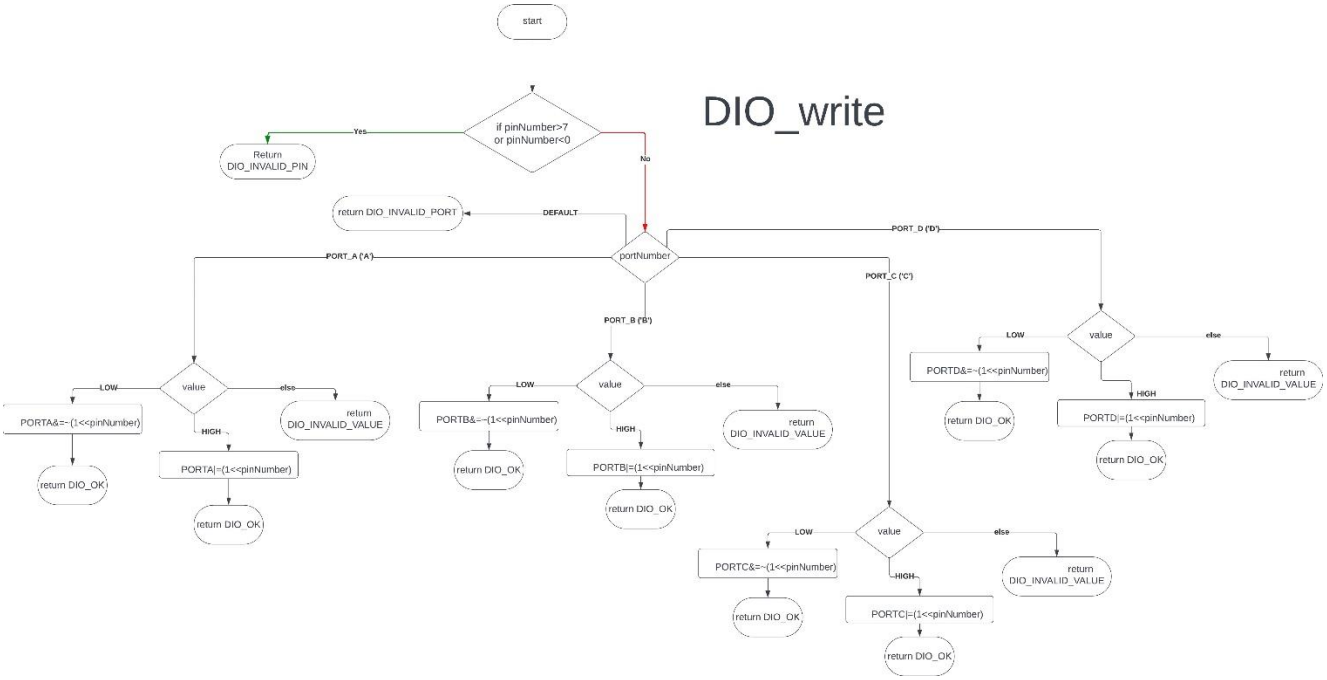
DIO_Init: - for better view click [here]

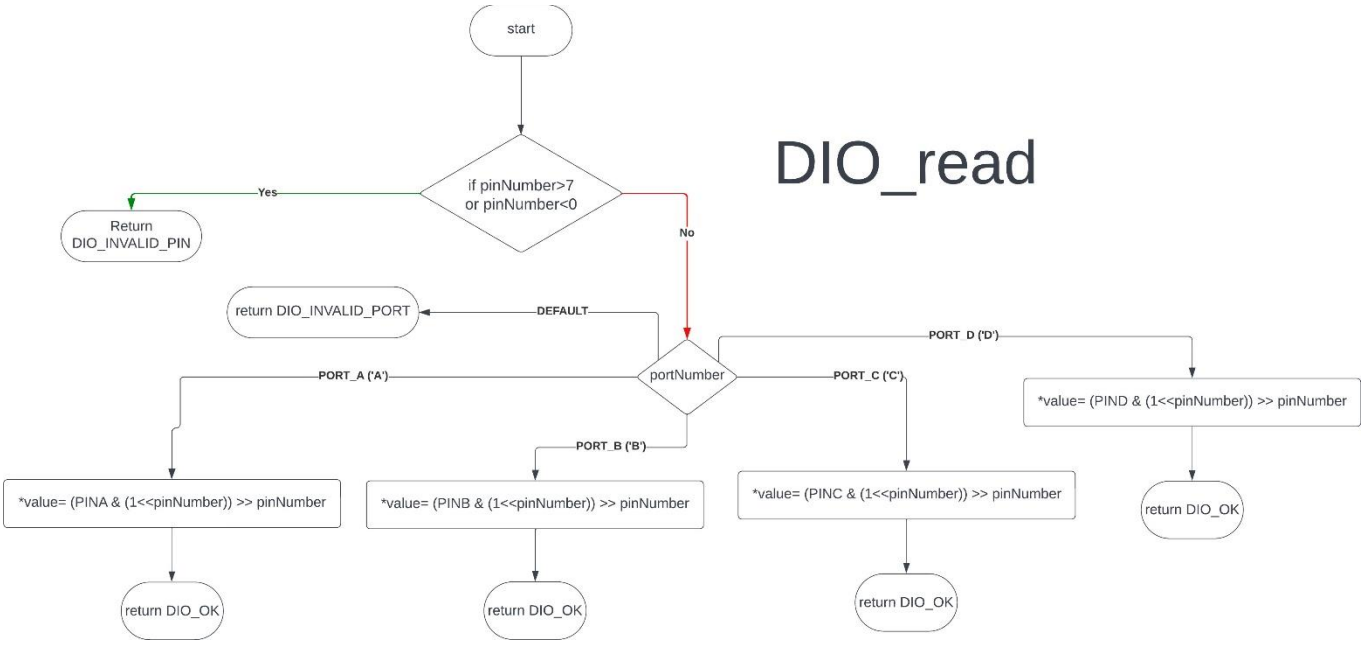


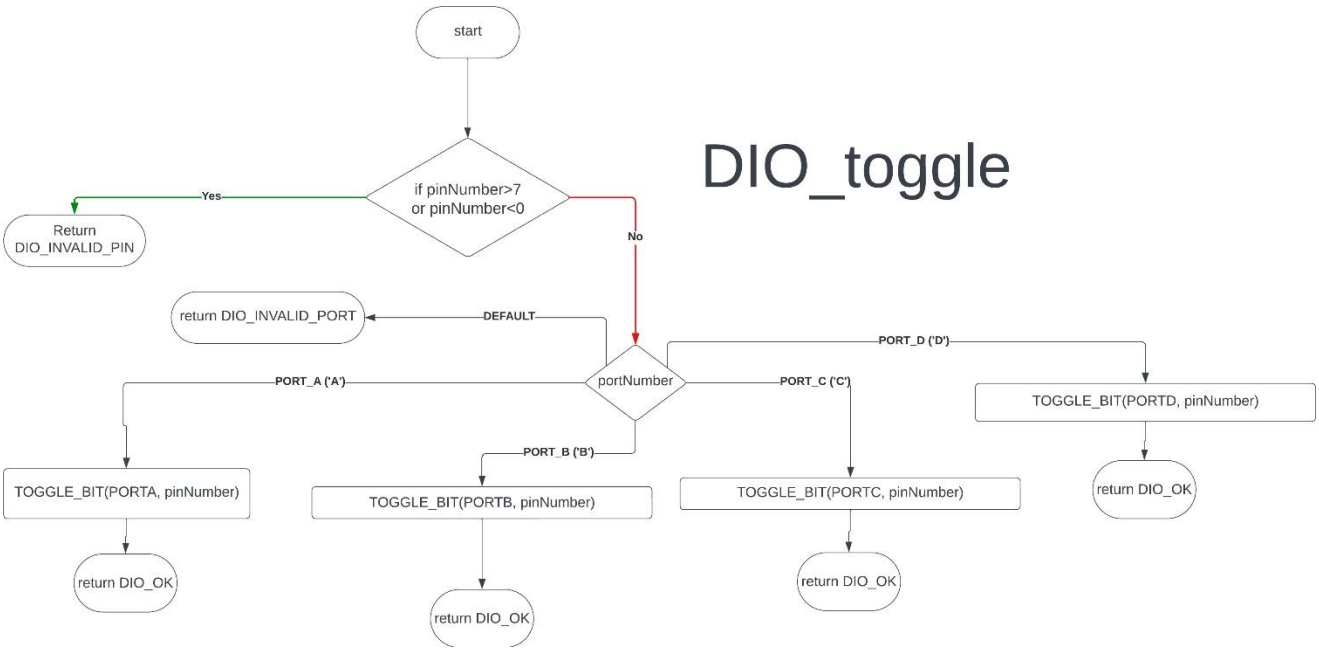




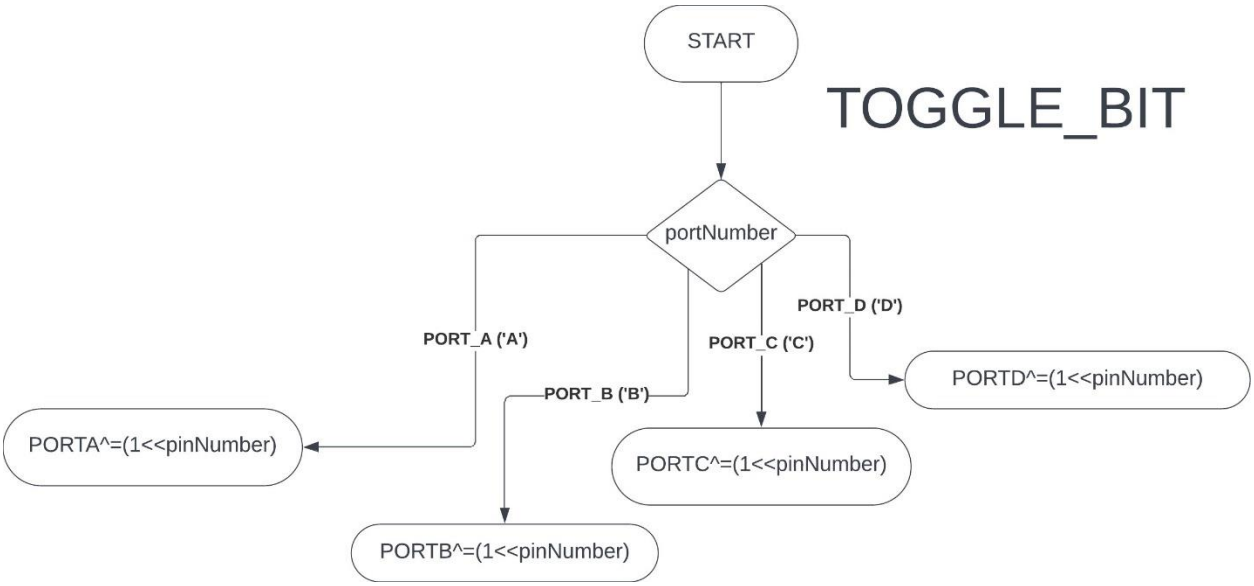
DIO_WRITE for better view click [here]



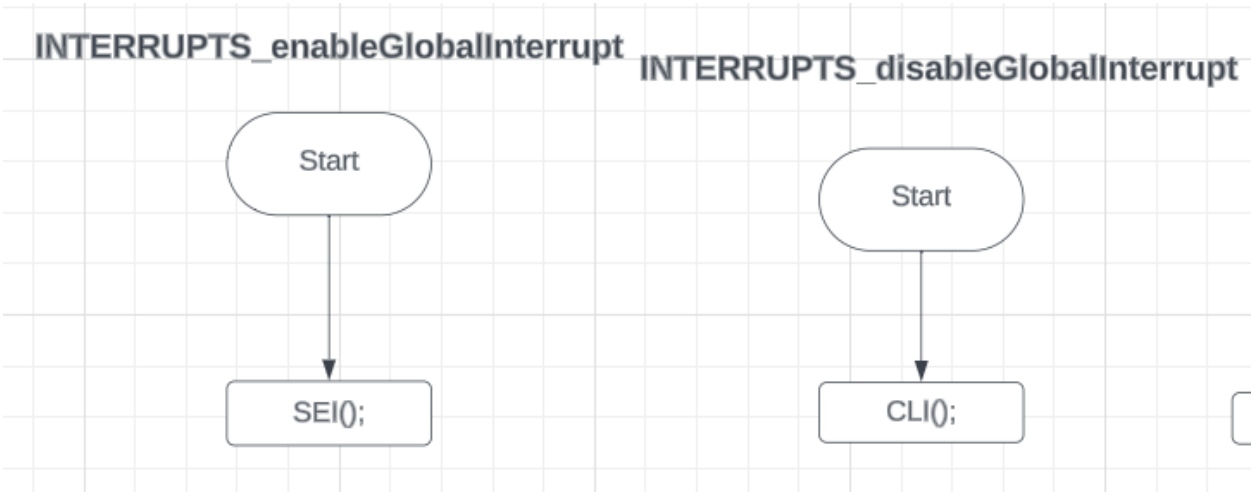

DIO_read
start
if pinNumber>7
or pinNumber<0
Yes
Return
DIO_INVALID_PIN
No
return DIO_INVALID_PORT
DEFAULT
portNumber
PORT_A ('A')
PORT_B ('B')
PORT_C ('C')
PORT_D ('D')
*value= (PINA & (1<<pinNumber)) >> pinNumber
*value= (PINB & (1<<pinNumber)) >> pinNumber
*value= (PINC & (1<<pinNumber)) >> pinNumber
*value= (PIND & (1<<pinNumber)) >> pinNumber
return DIO_OK
return DIO_OK
return DIO_OK
return DIO_OK

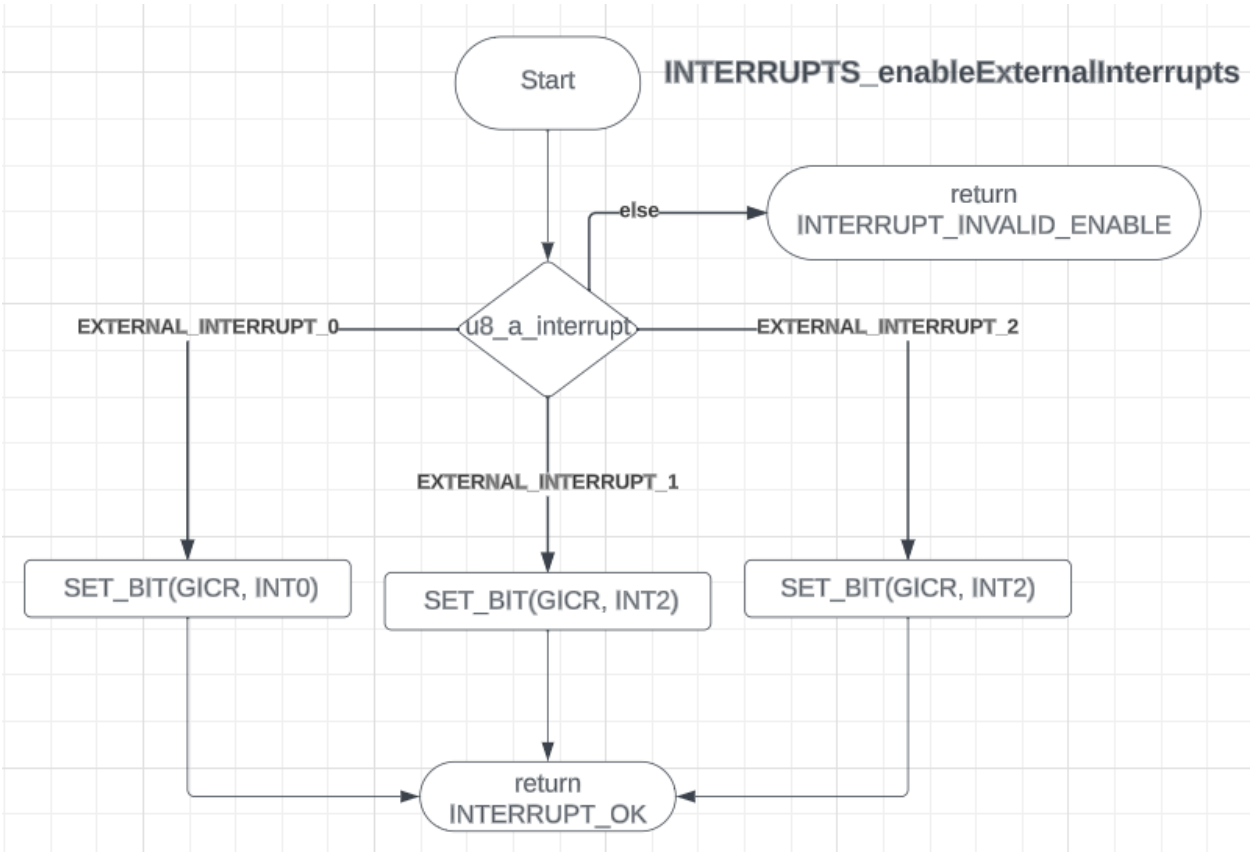

DIO_toggle
start
if pinNumber>7
or pinNumber<0
Yes
Return
DIO_INVALID_PIN
No
return DIO_INVALID_PORT
DEFAULT
portNumber
PORT_A ('A')
PORT_B ('B')
PORT_C ('C')
PORT_D ('D')
TOGGLE_BIT(PORTA, pinNumber)
TOGGLE_BIT(PORTB, pinNumber)
TOGGLE_BIT(PORTC, pinNumber)
TOGGLE_BIT(PORTD, pinNumber)
return DIO_OK
return DIO_OK
return DIO_OK
return DIO_OK

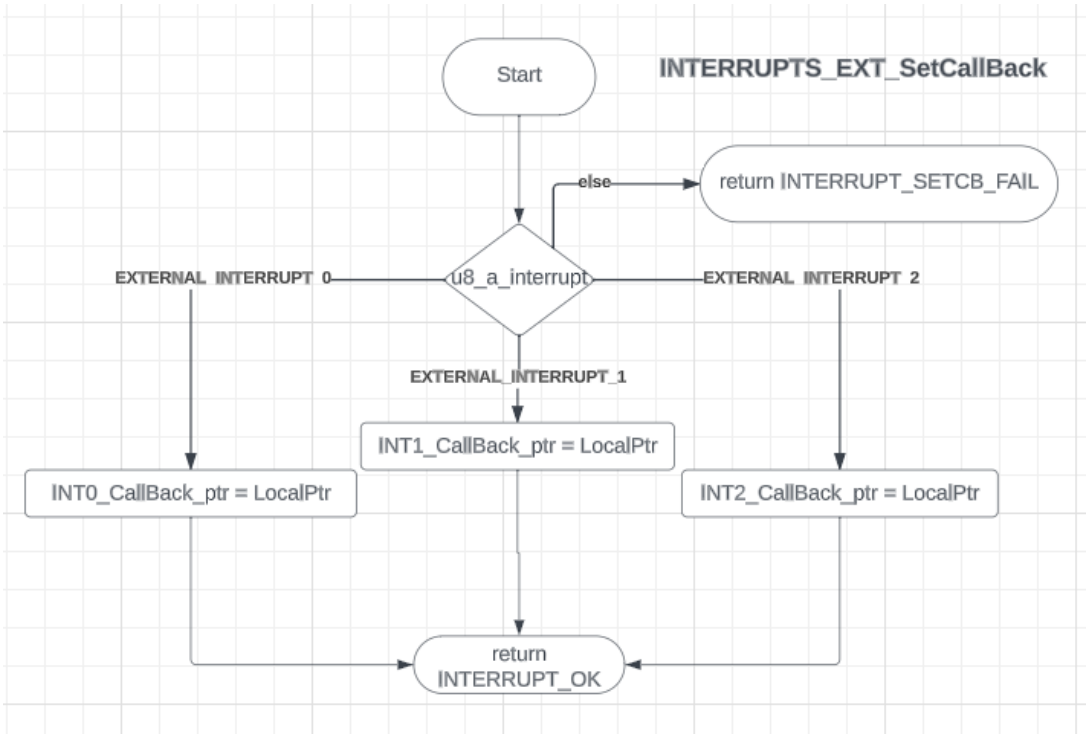
TOGGLE_BIT

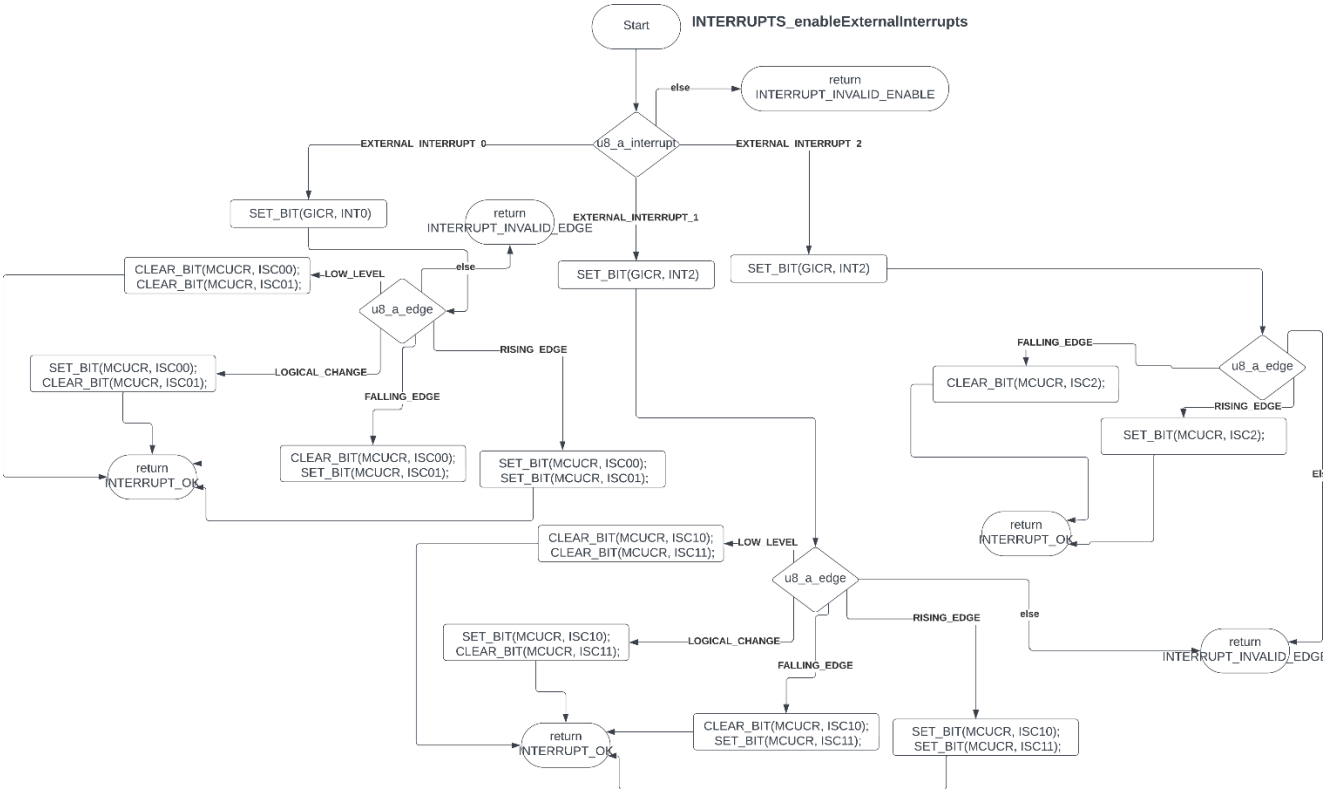


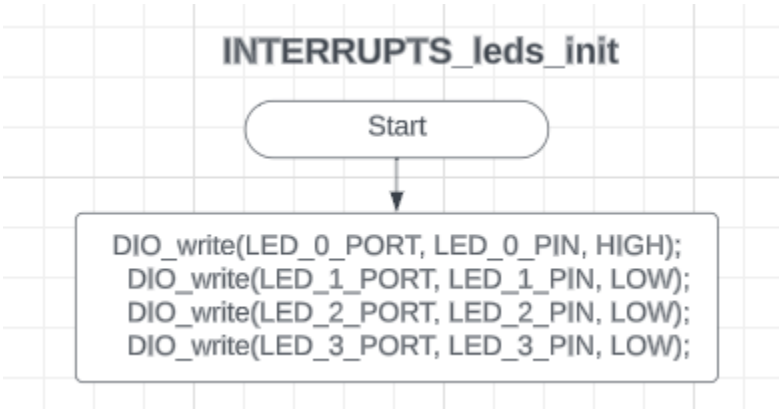
Interrupts APIs flowchart

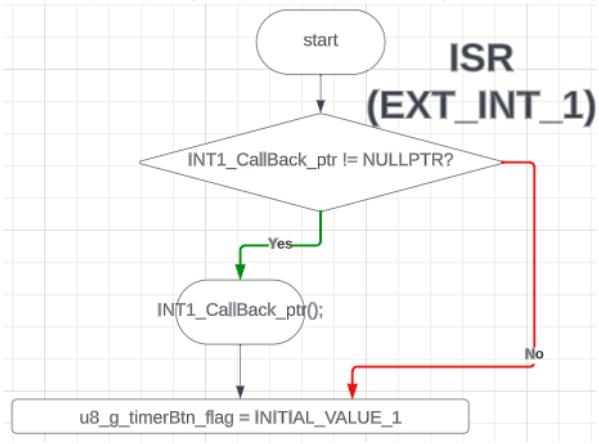


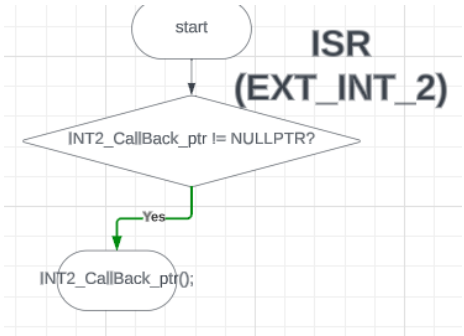

Start
INTERRUPTS_enableExternalInterrupts
else
return
INTERRUPT_INVALID_ENABLE
EXTERNAL_INTERRUPT_0
u8_a_interrupt
EXTERNAL_INTERRUPT_2
EXTERNAL_INTERRUPT_1
SET_BIT(GICR, INT0)
SET_BIT(GICR, INT2)
SET_BIT(GICR, INT2)
return
INTERRUPT_OK

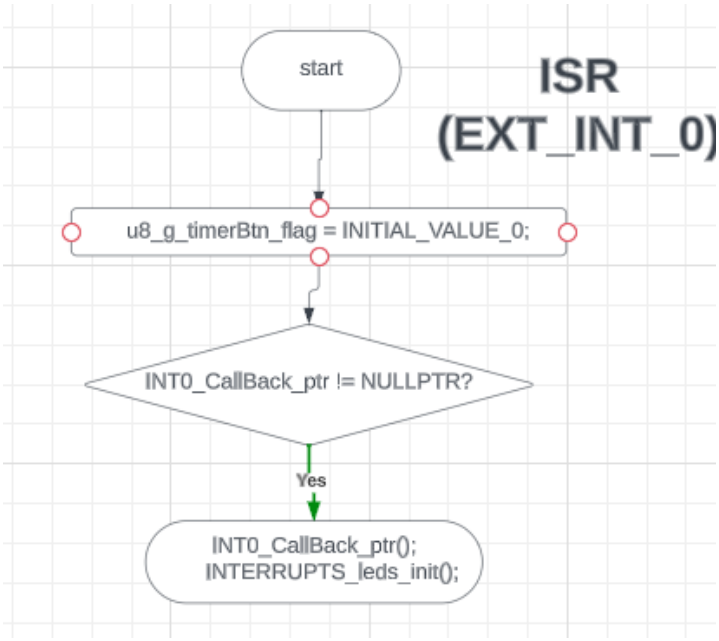

Start
INTERRUPTS_EXT_SetCallBack
else
return INTERRUPT_SETCB_FAIL
EXTERNAL_INTERRUPT_0
u8_a_interrupt
EXTERNAL_INTERRUPT_2
EXTERNAL_INTERRUPT_1
INT1_CallBack_ptr = LocalPtr
INT0_CallBack_ptr = LocalPtr
INT2_CallBack_ptr = LocalPtr
return
INTERRUPT_OK


Start
INTERRUPTS_enableExternalInterrupts
else
return INTERRUPT_INVALID_ENABLE
u8_a_interrupt
EXTERNAL_INTERRUPT_0
EXTERNAL_INTERRUPT_1
EXTERNAL_INTERRUPT_2
SET_BIT(GICR, INT0)
return INTERRUPT_INVALID_EDGE
SET_BIT(GICR, INT2)
SET_BIT(GICR, INT2)
CLEAR_BIT(MCUCR, ISC00);
CLEAR_BIT(MCUCR, ISC01);
LOW_LEVEL
else
u8_a_edge
RISING_EDGE
SET_BIT(MCUCR, ISC00);
CLEAR_BIT(MCUCR, ISC01);
LOGICAL_CHANGE
FALLING_EDGE
CLEAR_BIT(MCUCR, ISC00);
SET_BIT(MCUCR, ISC01);
SET_BIT(MCUCR, ISC00);
SET_BIT(MCUCR, ISC01);
return INTERRUPT_OK
FALLING_EDGE
u8_a_edge
CLEAR_BIT(MCUCR, ISC2);
RISING_EDGE
SET_BIT(MCUCR, ISC2);
return INTERRUPT_OK
CLEAR_BIT(MCUCR, ISC10);
CLEAR_BIT(MCUCR, ISC11);
LOW_LEVEL
u8_a_edge
SET_BIT(MCUCR, ISC10);
CLEAR_BIT(MCUCR, ISC11);
LOGICAL_CHANGE
RISING_EDGE
else
return INTERRUPT_INVALID_EDGE
FALLING_EDGE
CLEAR_BIT(MCUCR, ISC10);
SET_BIT(MCUCR, ISC11);
SET_BIT(MCUCR, ISC10);
SET_BIT(MCUCR, ISC11);
return INTERRUPT_OK

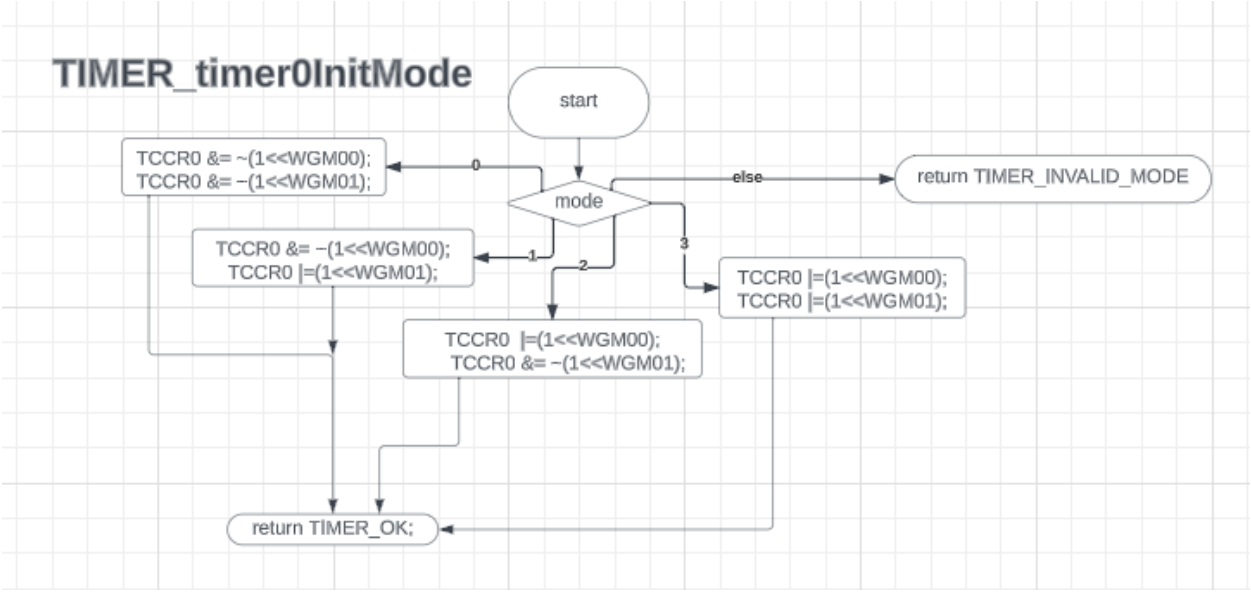

INTERRUPTS_leds_init
Start
DIO_write(LED_0_PORT, LED_0_PIN, HIGH);
DIO_write(LED_1_PORT, LED_1_PIN, LOW);
DIO_write(LED_2_PORT, LED_2_PIN, LOW);
DIO_write(LED_3_PORT, LED_3_PIN, LOW);

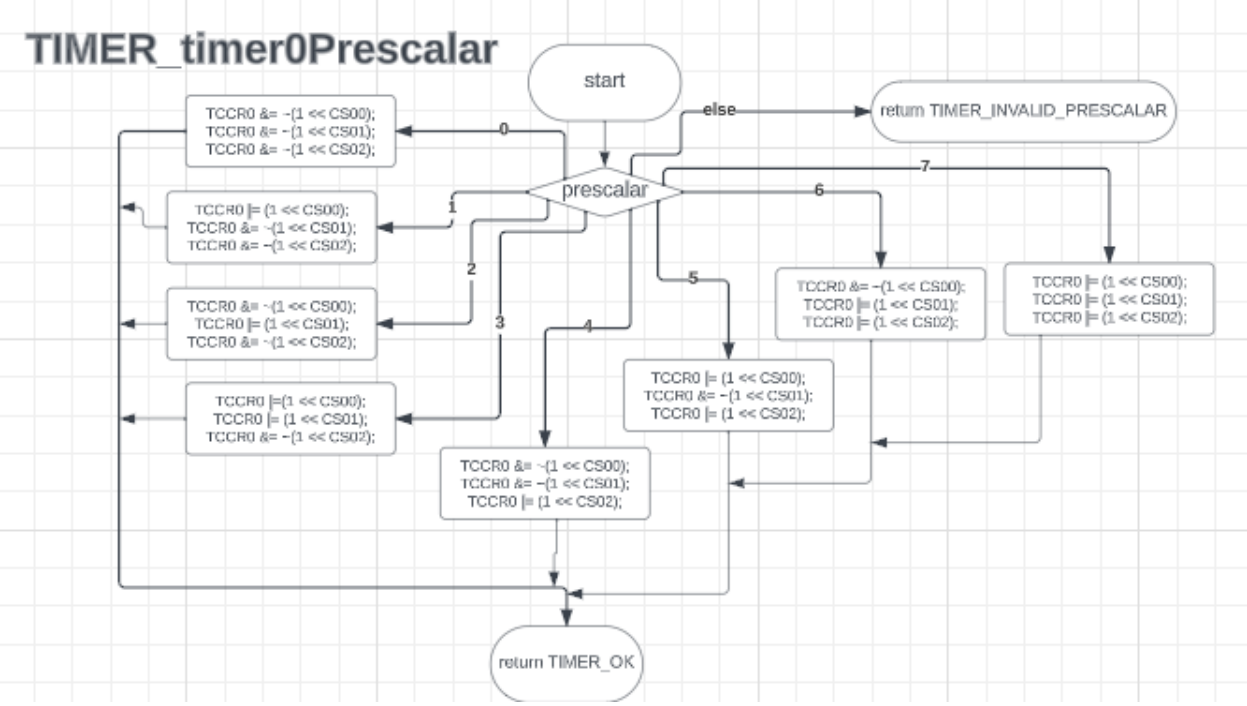

start
ISR
(EXT_INT_1)
INT1_CallBack_ptr != NULLPTR?
Yes
INT1_CallBack_ptr();
No
u8_g_timerBtn_flag = INITIAL_VALUE_1

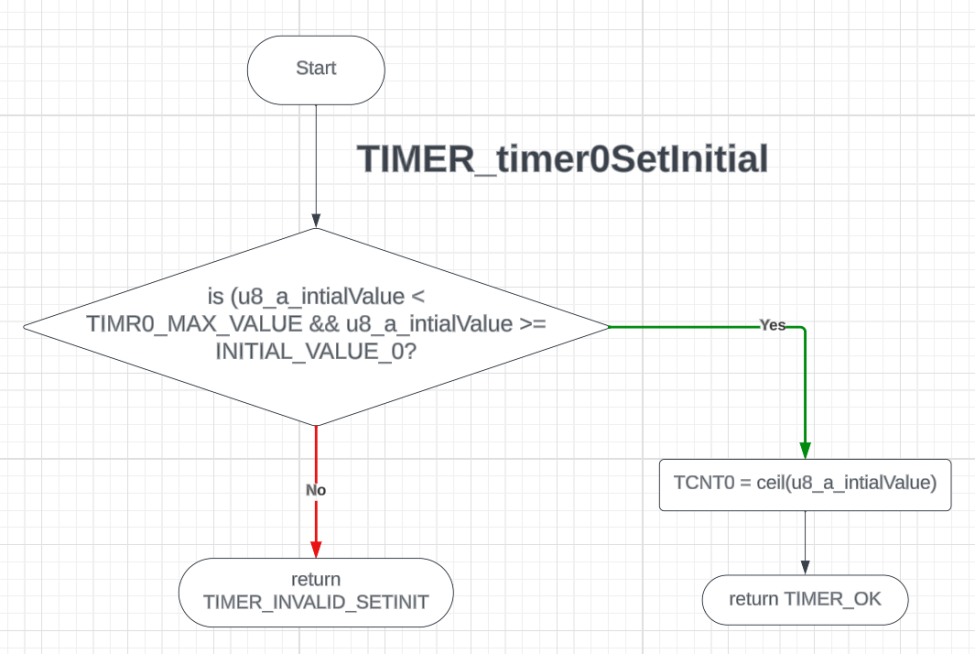

start
ISR
(EXT_INT_2)
INT2_CallBack_ptr != NULLPTR?
Yes
INT2_CallBack_ptr();

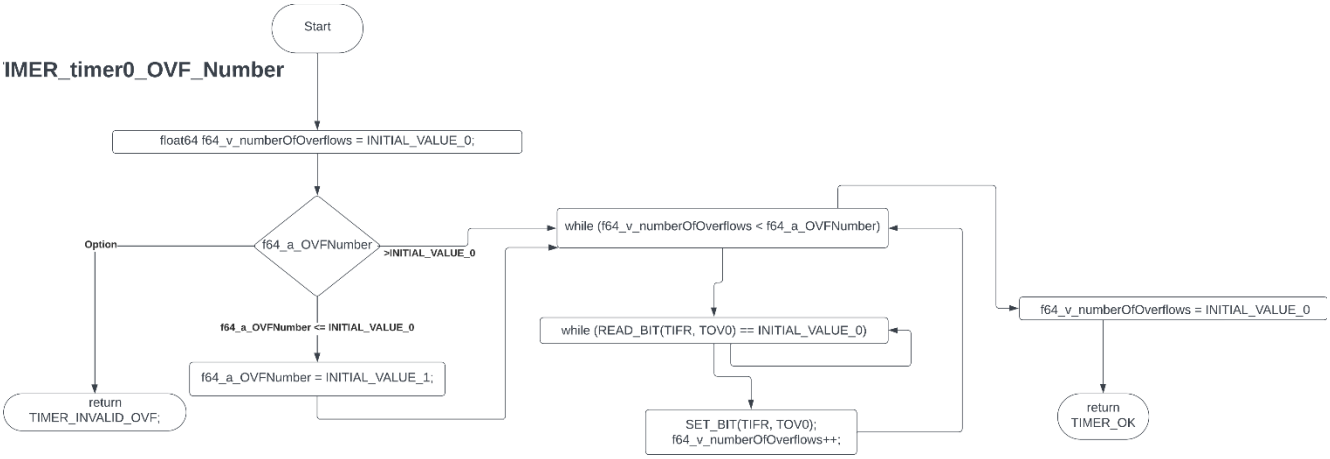


Timer APIs flowchart

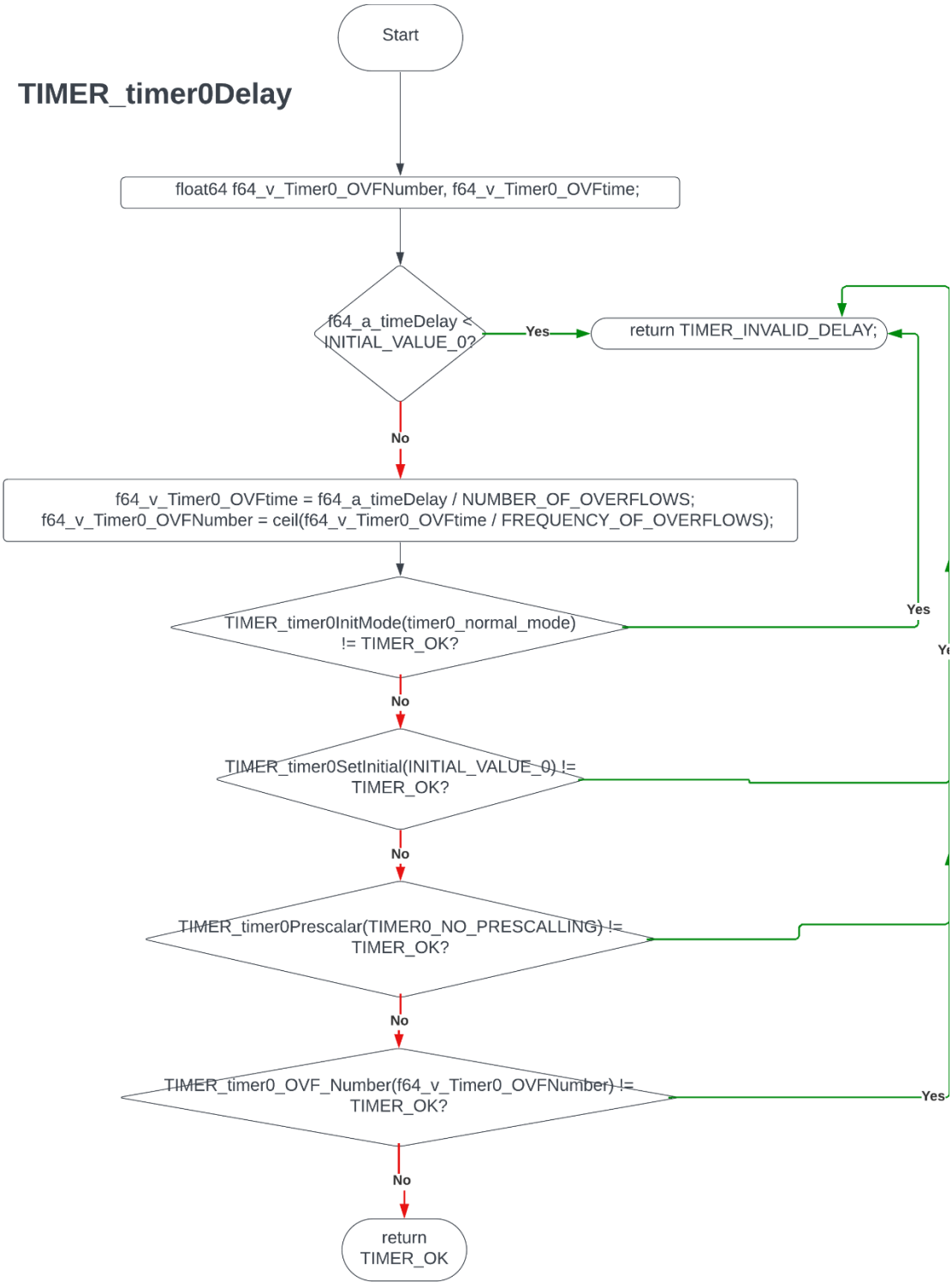


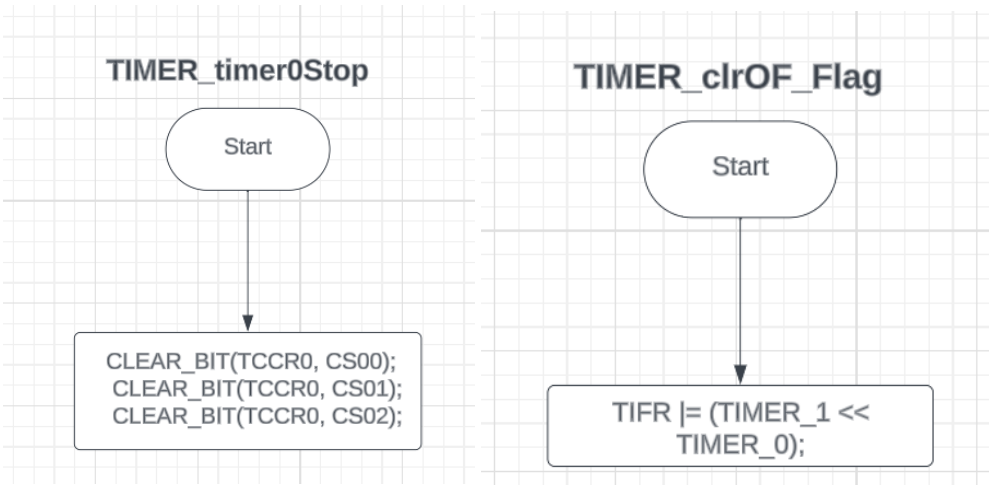

TIMER_timer0Prescalar
start
prescalar
else
return TIMER_INVALID_PRESCALAR
0
1
2
3
4
5
6
7
TCCR0 &= ~(1 << CS00);
TCCR0 &= ~(1 << CS01);
TCCR0 &= ~(1 << CS02);
TCCR0 |= (1 << CS00);
TCCR0 &= ~(1 << CS01);
TCCR0 &= ~(1 << CS02);
TCCR0 &= ~(1 << CS00);
TCCR0 |= (1 << CS01);
TCCR0 &= ~(1 << CS02);
TCCR0 |=(1 << CS00);
TCCR0 |= (1 << CS01);
TCCR0 &= ~(1 << CS02);
TCCR0 &= ~(1 << CS00);
TCCR0 &= ~(1 << CS01);
TCCR0 |= (1 << CS02);
TCCR0 |= (1 << CS00);
TCCR0 &= ~(1 << CS01);
TCCR0 |= (1 << CS02);
TCCR0 |= (1 << CS00);
TCCR0 |= (1 << CS01);
TCCR0 |= (1 << CS02);
return TIMER_OK

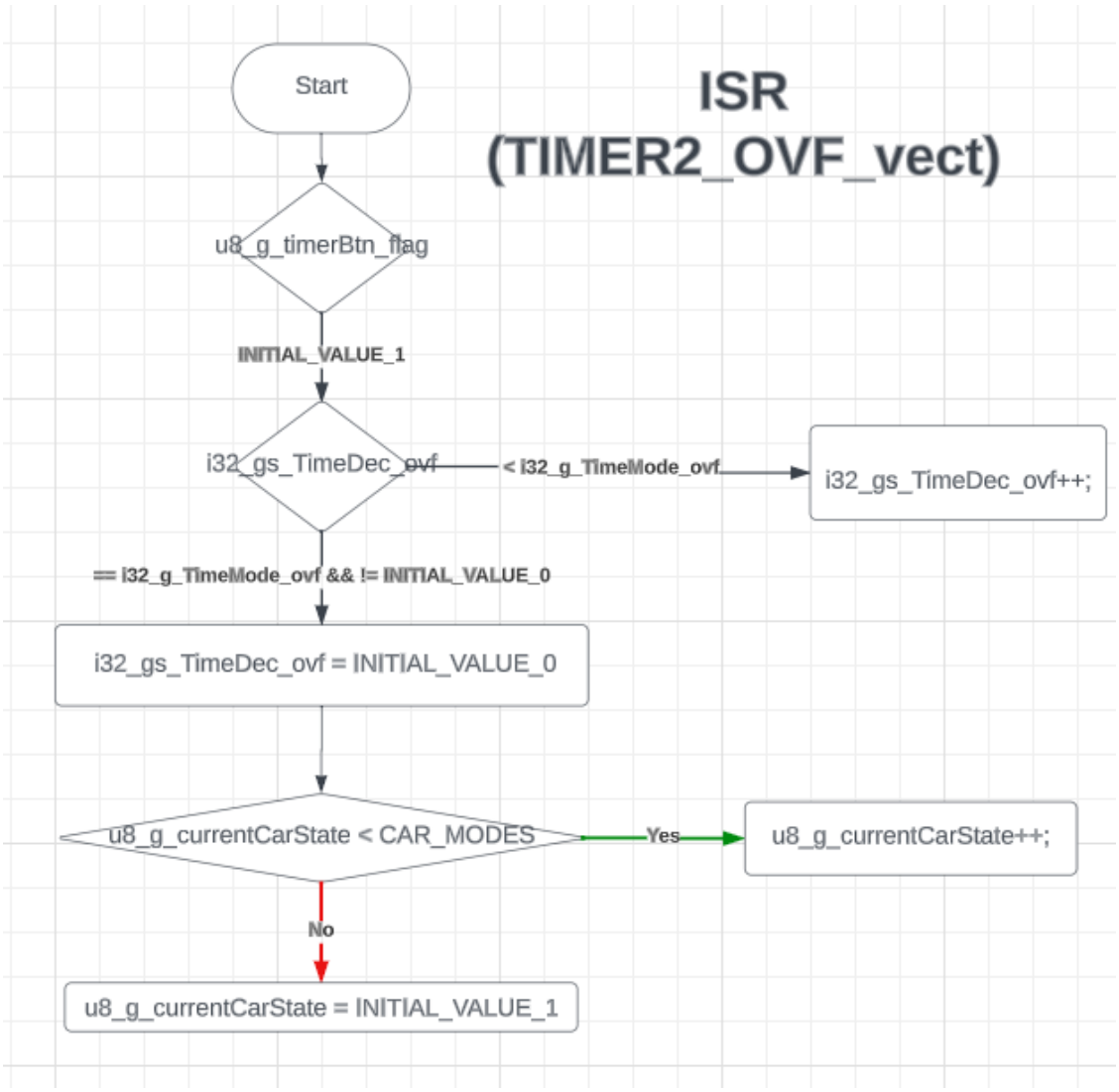

Start
TIMER_timer0SetInitial
is (u8_a_intialValue < TIMR0_MAX_VALUE && u8_a_intialValue >= INITIAL_VALUE_0?
Yes
No
TCNT0 = ceil(u8_a_intialValue)
return TIMER_OK
return TIMER_INVALID_SETINIT

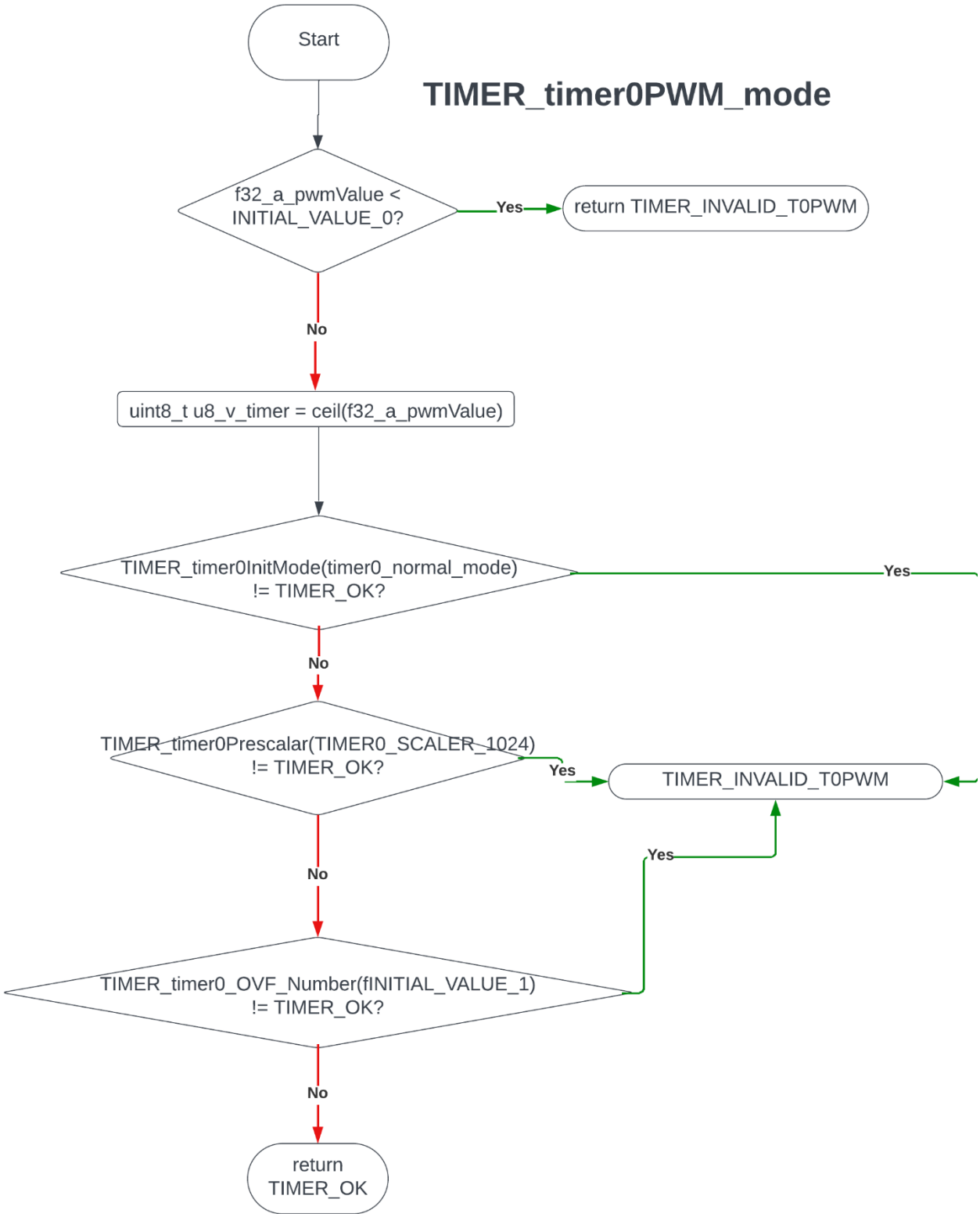

IMER_timer0_OVF_Number
Start
float64 f64_v_numberOfOverflows = INITIAL_VALUE_0;
f64_a_OVFNumber
Option
>INITIAL_VALUE_0
f64_a_OVFNumber <= INITIAL_VALUE_0
return TIMER_INVALID_OVF;
f64_a_OVFNumber = INITIAL_VALUE_1;
while (f64_v_numberOfOverflows < f64_a_OVFNumber)
while (READ_BIT(TIFR, TOV0) == INITIAL_VALUE_0)
SET_BIT(TIFR, TOV0);
f64_v_numberOfOverflows++;
f64_v_numberOfOverflows = INITIAL_VALUE_0
return TIMER_OK

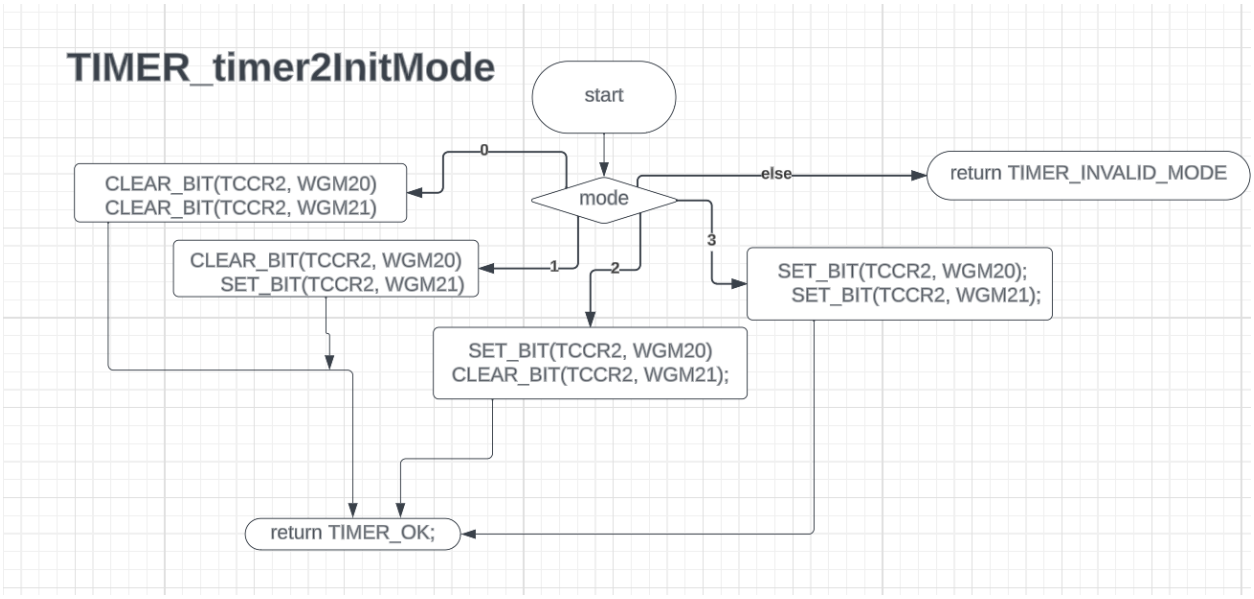
TIMER_timer0Delay

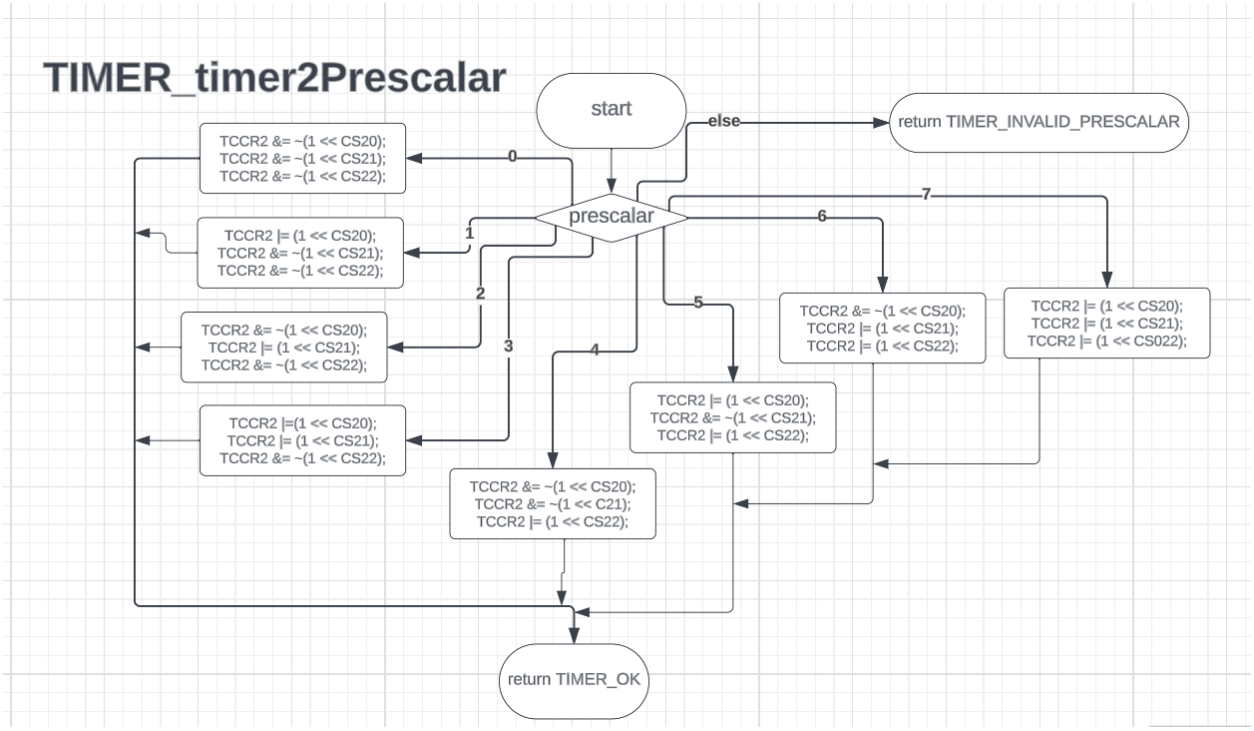


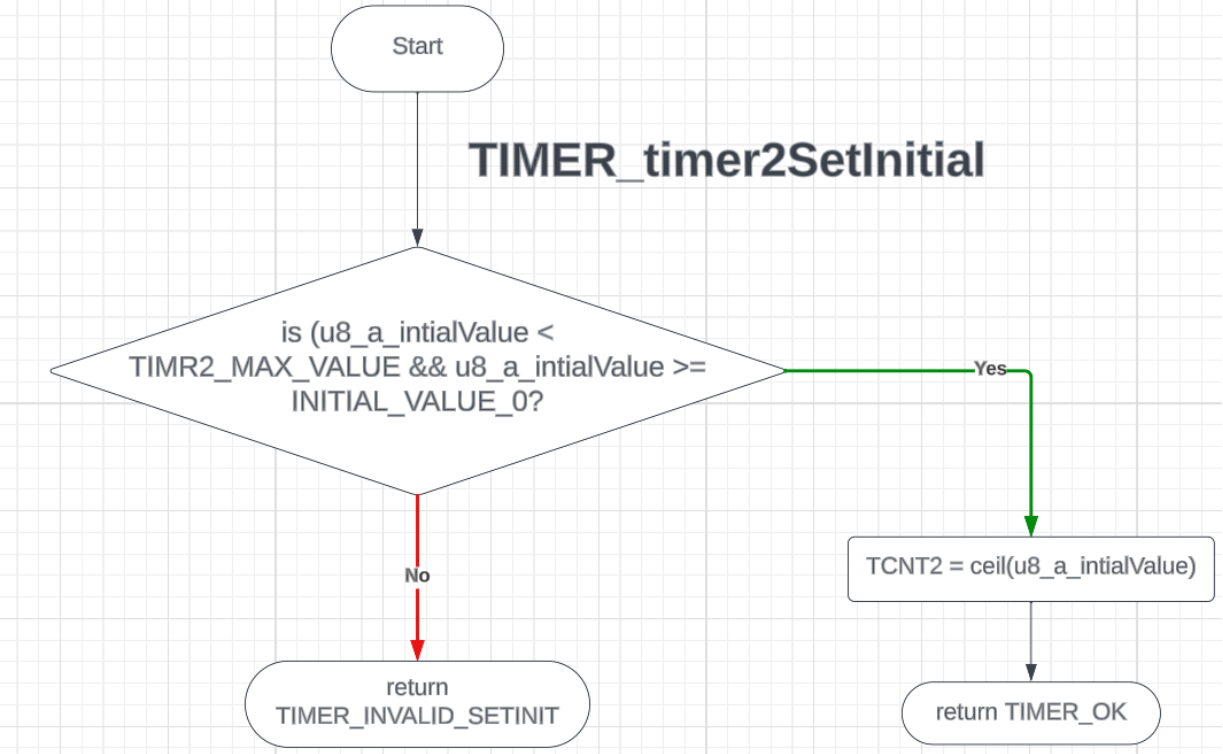

TIMER_timer0Stop
Start
CLEAR_BIT(TCCR0, CS00);
CLEAR_BIT(TCCR0, CS01);
CLEAR_BIT(TCCR0, CS02);
TIMER_clrOF_Flag
Start
TIFR |= (TIMER_1 <<
TIMER_0);

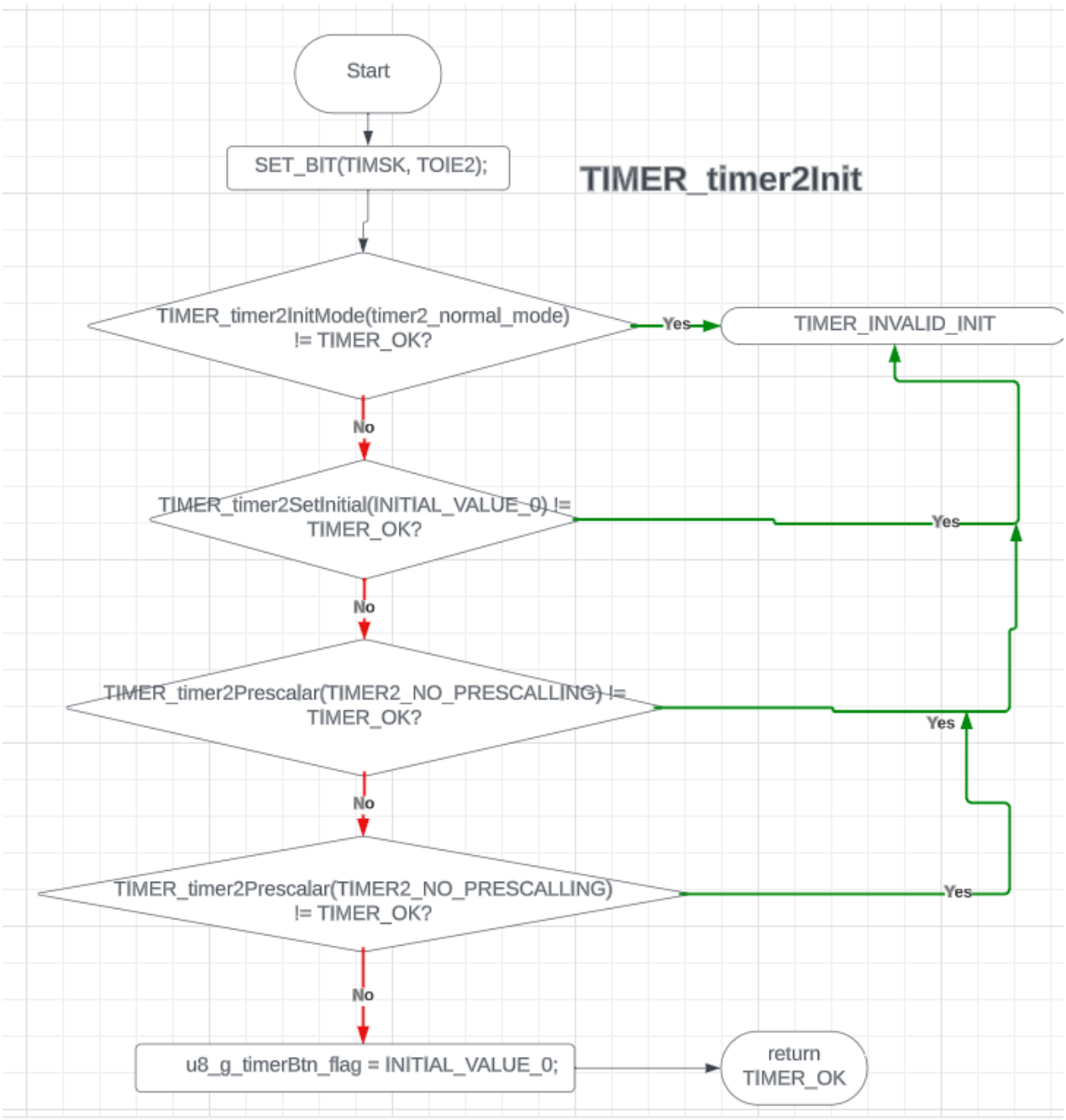

Start
ISR
(TIMER2_OVF_vect)
u8_g_timerBtn_flag
INITIAL_VALUE_1
i32_gs_TimeDec_ovf
< i32_g_TimeMode_ovf
i32_gs_TimeDec_ovf++;
== i32_g_TimeMode_ovf && != INITIAL_VALUE_0
i32_gs_TimeDec_ovf = INITIAL_VALUE_0
u8_g_currentCarState < CAR_MODES
Yes
u8_g_currentCarState++;
No
u8_g_currentCarState = INITIAL_VALUE_1

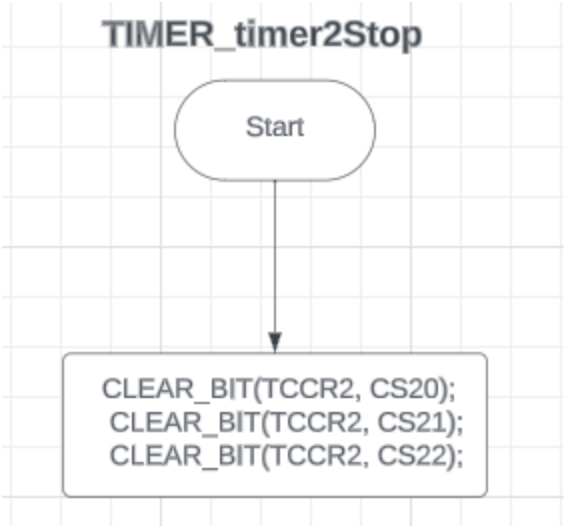

TIMER_timer0PWM_mode
Start
f32_a_pwmValue < INITIAL_VALUE_0?
Yes
return TIMER_INVALID_T0PWM
No
uint8_t u8_v_timer = ceil(f32_a_pwmValue)
TIMER_timer0InitMode(timer0_normal_mode) != TIMER_OK?
Yes
No
TIMER_timer0Prescalar(TIMER0_SCALER_1024) != TIMER_OK?
Yes
TIMER_INVALID_T0PWM
No
Yes
TIMER_timer0_OVF_Number(fINITIAL_VALUE_1) != TIMER_OK?
No
return TIMER_OK


TIMER_timer2InitMode
start
mode
0
1
2
3
else
CLEAR_BIT(TCCR2, WGM20)
CLEAR_BIT(TCCR2, WGM21)
CLEAR_BIT(TCCR2, WGM20)
SET_BIT(TCCR2, WGM21)
SET_BIT(TCCR2, WGM20)
CLEAR_BIT(TCCR2, WGM21);
SET_BIT(TCCR2, WGM20);
SET_BIT(TCCR2, WGM21);
return TIMER_INVALID_MODE
return TIMER_OK;

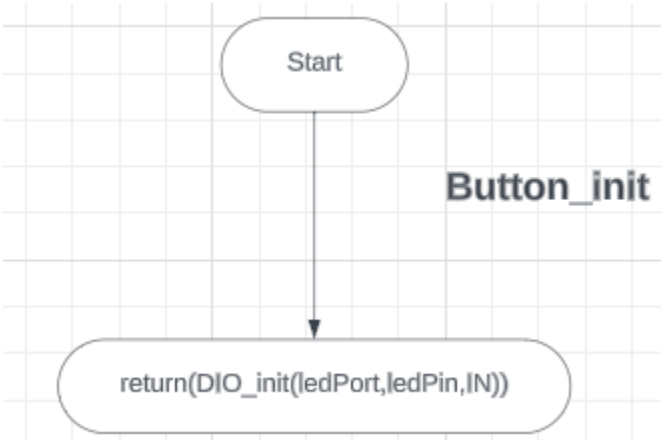

TIMER_timer2Prescalar
start
prescalar
else
return TIMER_INVALID_PRESCALAR
0
1
2
3
4
5
6
7
TCCR2 &= ~(1 << CS20);
TCCR2 &= ~(1 << CS21);
TCCR2 &= ~(1 << CS22);
TCCR2 |= (1 << CS20);
TCCR2 &= ~(1 << CS21);
TCCR2 &= ~(1 << CS22);
TCCR2 &= ~(1 << CS20);
TCCR2 |= (1 << CS21);
TCCR2 &= ~(1 << CS22);
TCCR2 |=(1 << CS20);
TCCR2 |= (1 << CS21);
TCCR2 &= ~(1 << CS22);
TCCR2 &= ~(1 << CS20);
TCCR2 &= ~(1 << C21);
TCCR2 |= (1 << CS22);
TCCR2 &= ~(1 << CS20);
TCCR2 |= (1 << CS21);
TCCR2 |= (1 << CS22);
TCCR2 |= (1 << CS20);
TCCR2 |= (1 << CS21);
TCCR2 |= (1 << CS022);
TCCR2 |= (1 << CS20);
TCCR2 &= ~(1 << CS21);
TCCR2 |= (1 << CS22);
return TIMER_OK

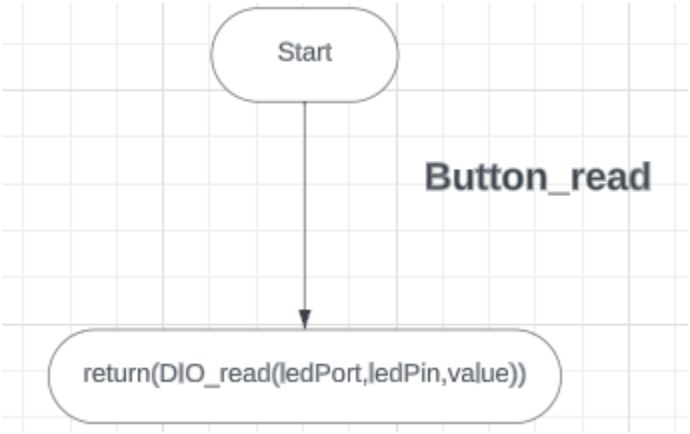

Start
TIMER_timer2SetInitial
is (u8_a_intialValue < TIMR2_MAX_VALUE && u8_a_intialValue >= INITIAL_VALUE_0?
Yes
No
TCNT2 = ceil(u8_a_intialValue)
return TIMER_INVALID_SETINIT
return TIMER_OK


TIMER_timer2Init
Start
SET_BIT(TIMSK, TOIE2);
TIMER_timer2InitMode(timer2_normal_mode) != TIMER_OK?
Yes
TIMER_INVALID_INIT
No
TIMER_timer2SetInitial(INITIAL_VALUE_0) != TIMER_OK?
Yes
No
TIMER_timer2Prescalar(TIMER2_NO_PRESCALLING) != TIMER_OK?
Yes
No
TIMER_timer2Prescalar(TIMER2_NO_PRESCALLING) != TIMER_OK?
Yes
No
u8_g_timerBtn_flag = INITIAL_VALUE_0;
return TIMER_OK

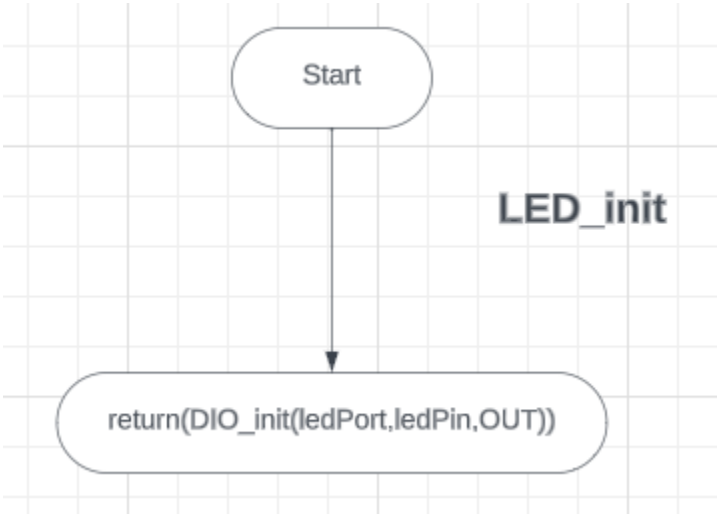


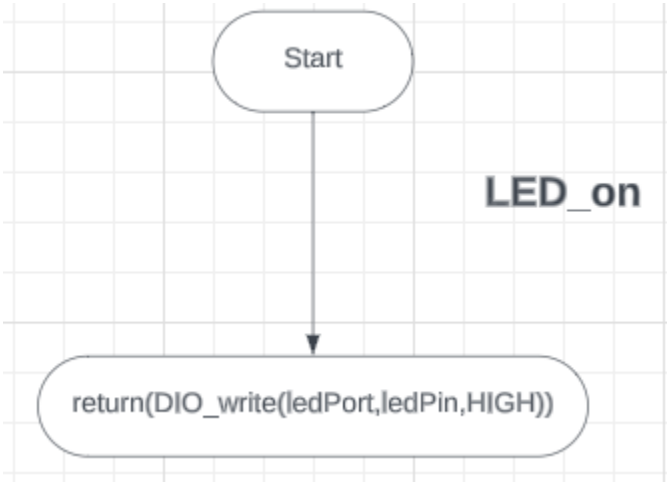
BUTTON APIs flowchart

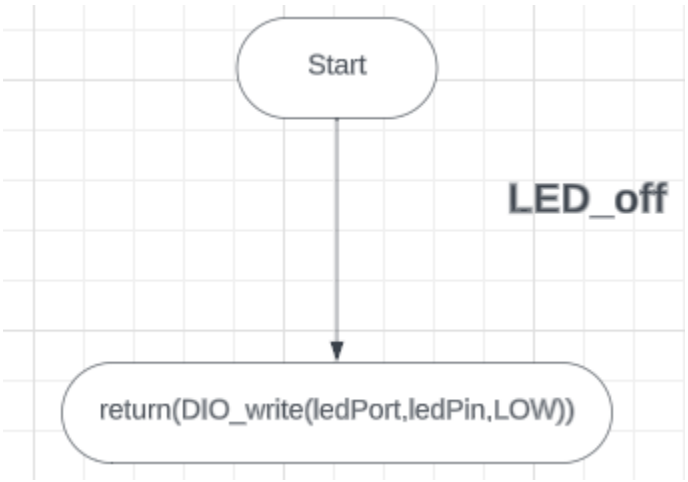


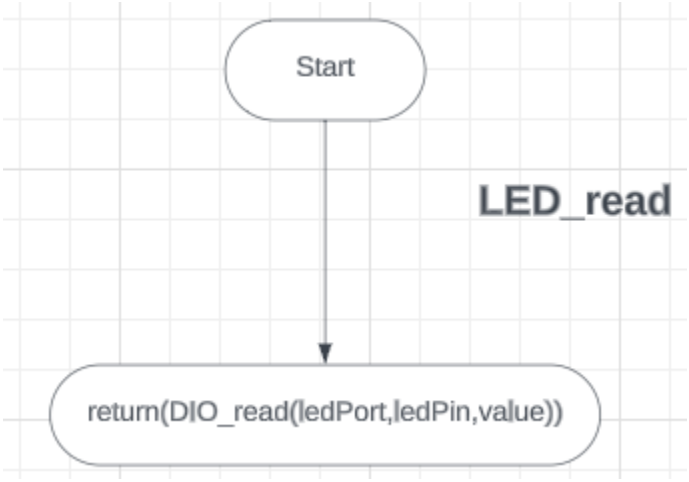


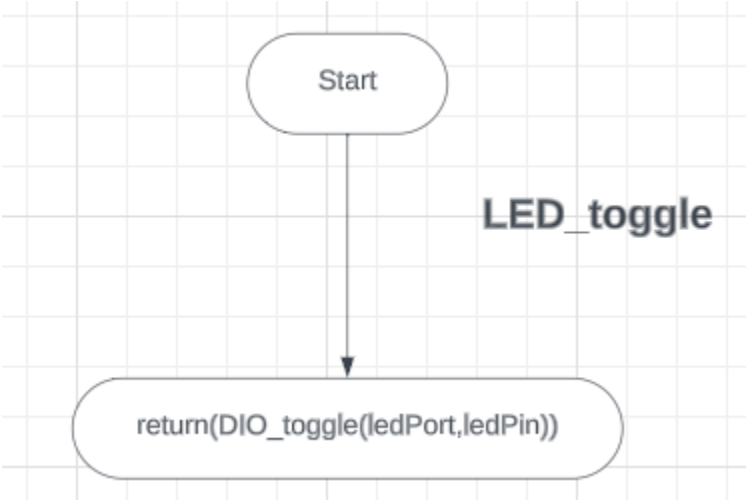
LED APIs flowchart



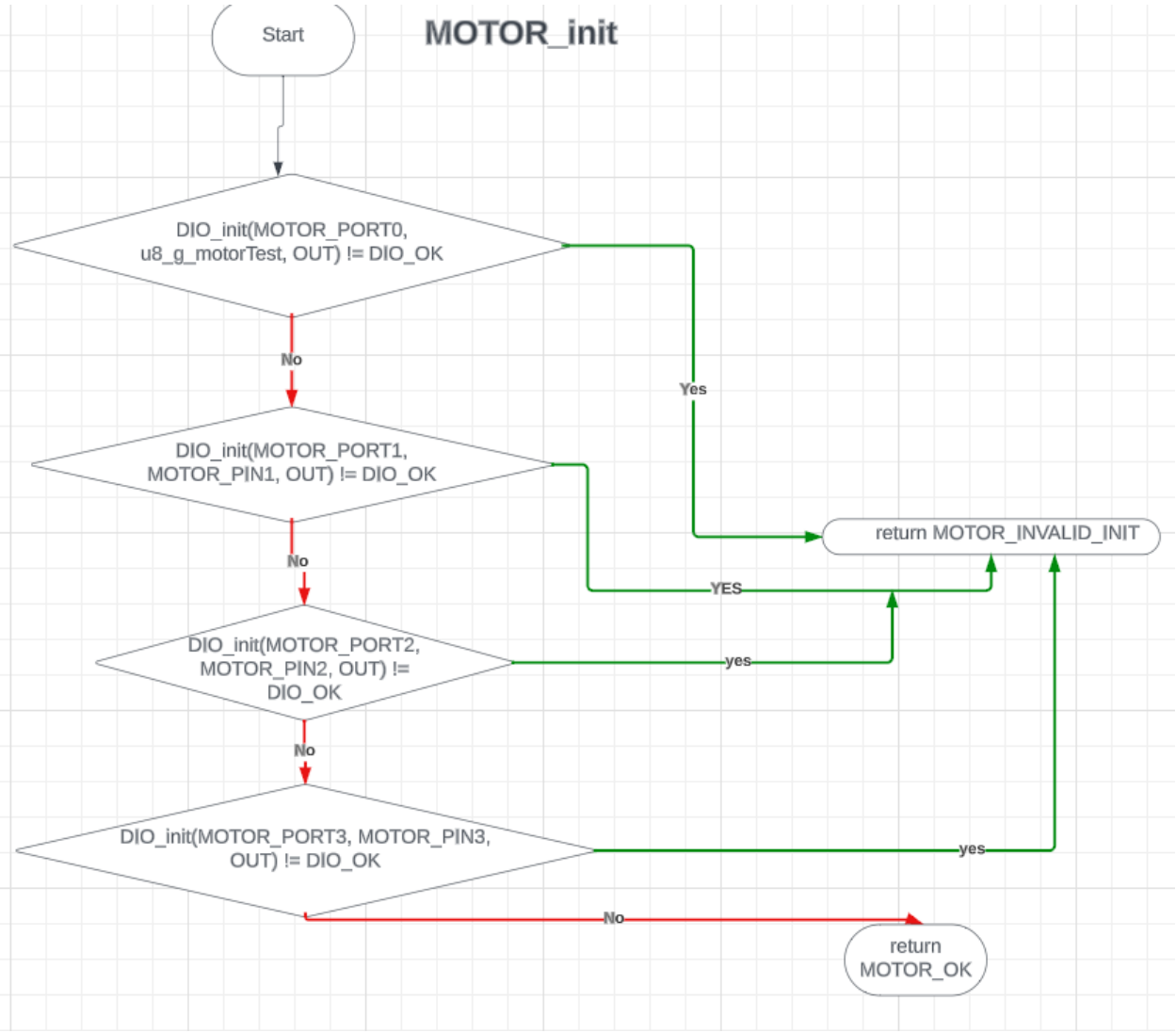


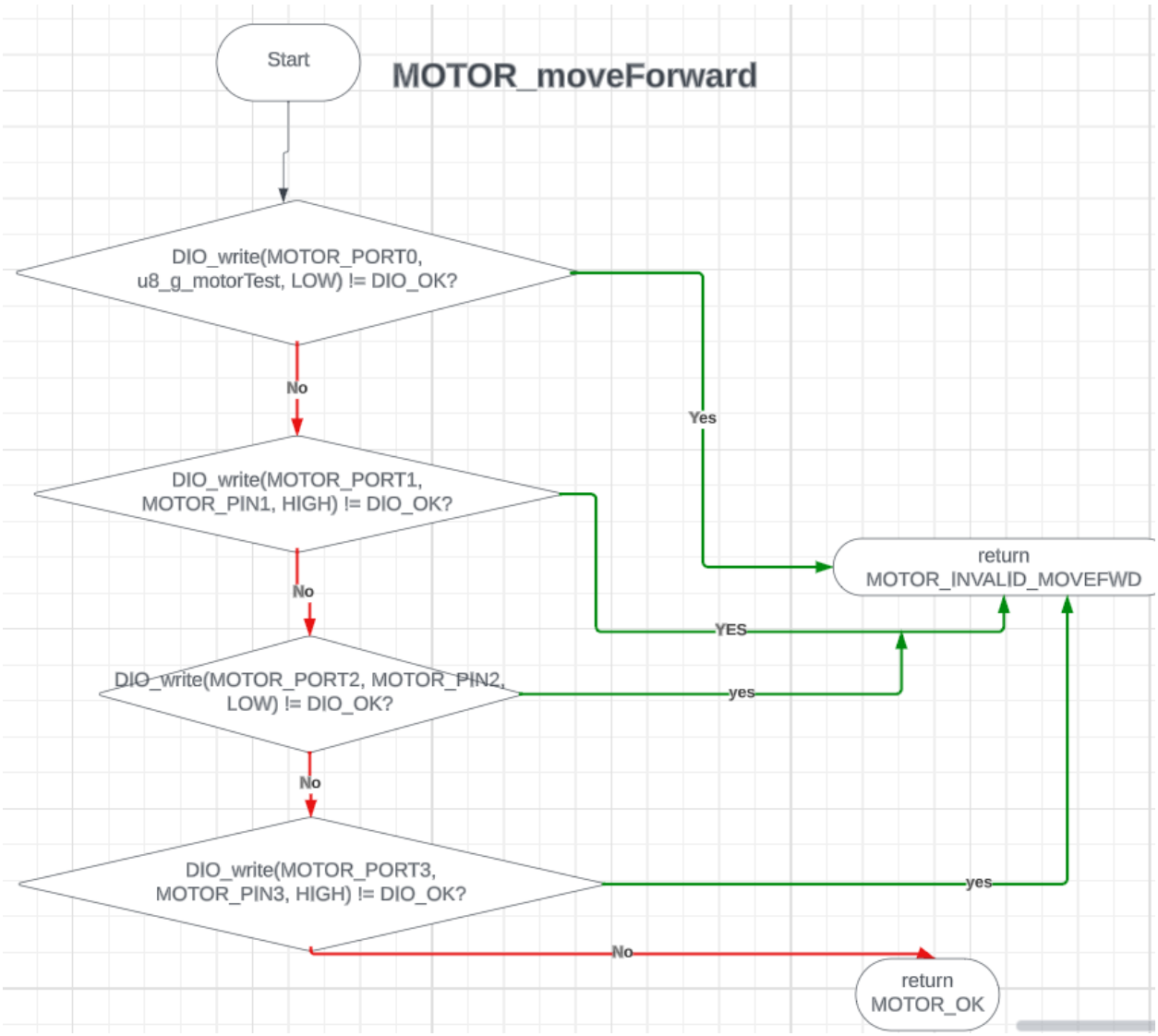


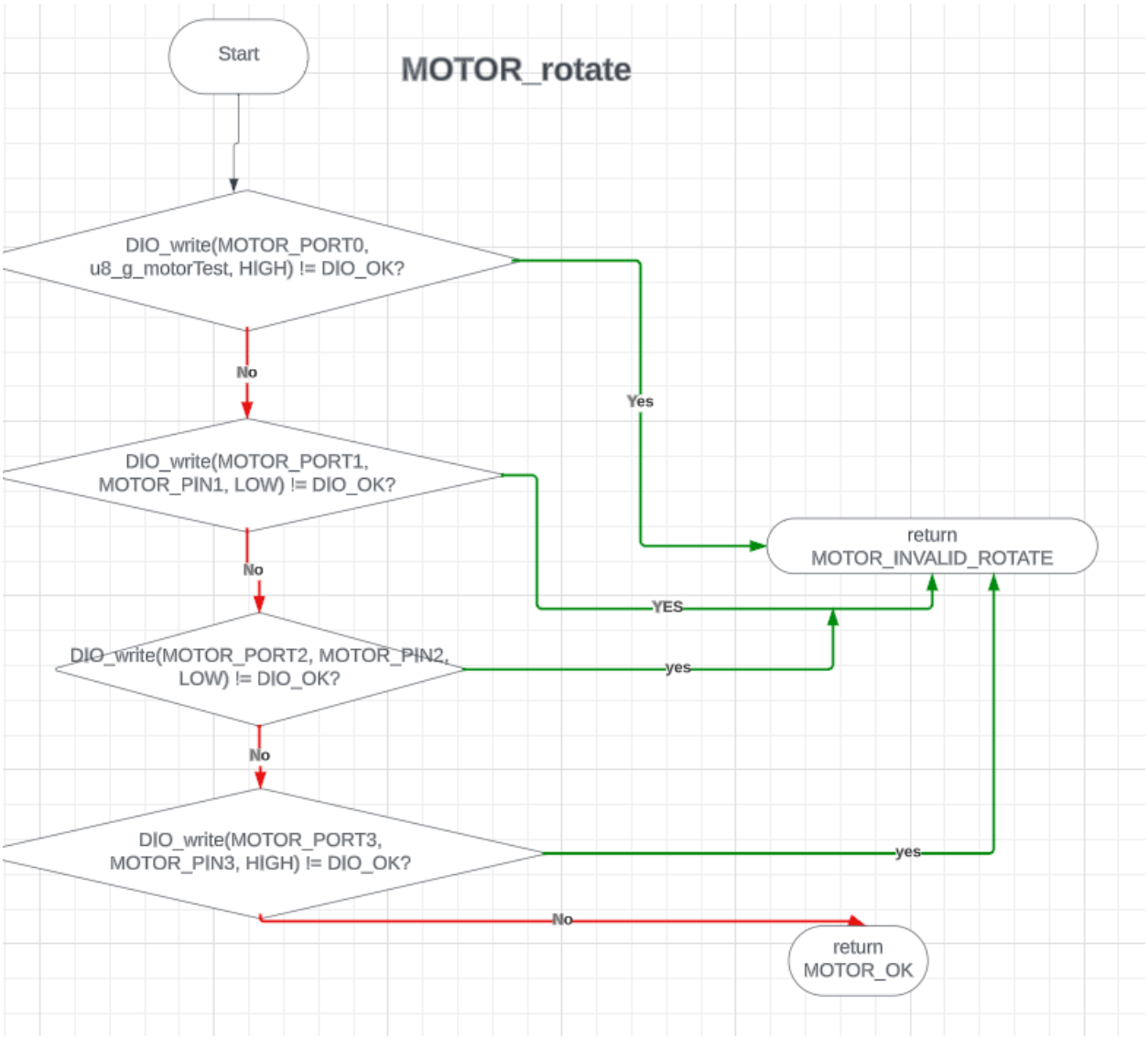

Start
LED_read
return(DIO_read(ledPort,ledPin,value))

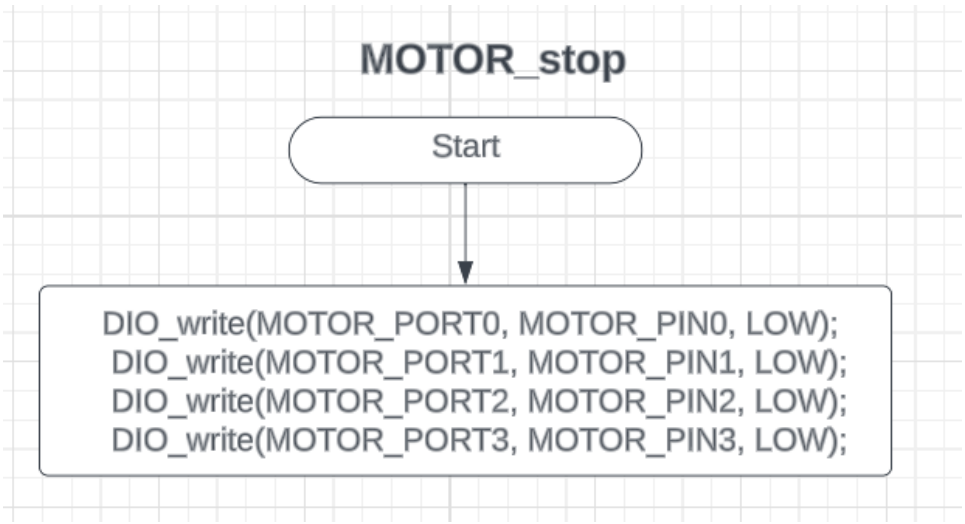

Start
LED_toggle
return(DIO_toggle(ledPort,ledPin))

Motor APIs flowchart



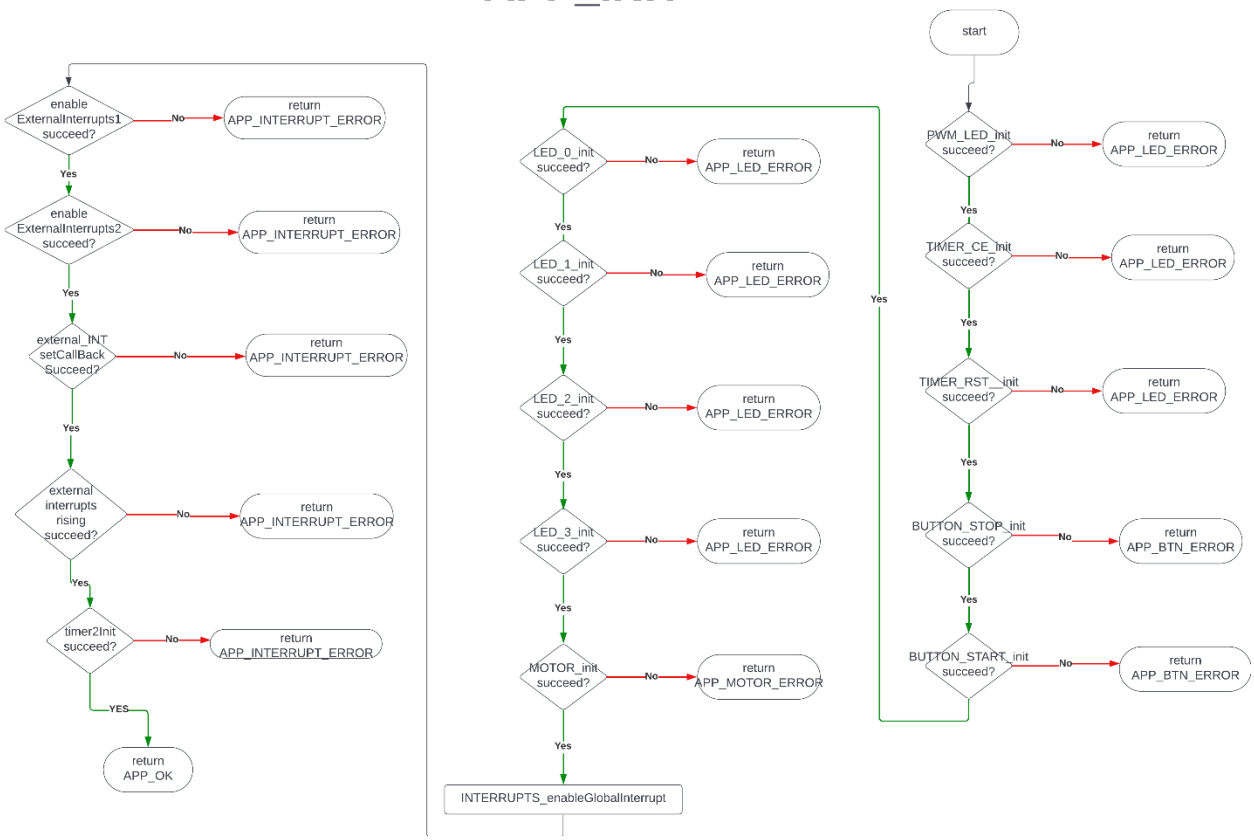

Start
MOTOR_moveForward
DIO_write(MOTOR_PORT0, u8_g_motorTest, LOW) != DIO_OK?
No
Yes
DIO_write(MOTOR_PORT1, MOTOR_PIN1, HIGH) != DIO_OK?
No
YES
return MOTOR_INVALID_MOVEFWD
DIO_write(MOTOR_PORT2, MOTOR_PIN2, LOW) != DIO_OK?
yes
No
DIO_write(MOTOR_PORT3, MOTOR_PIN3, HIGH) != DIO_OK?
yes
No
return MOTOR_OK

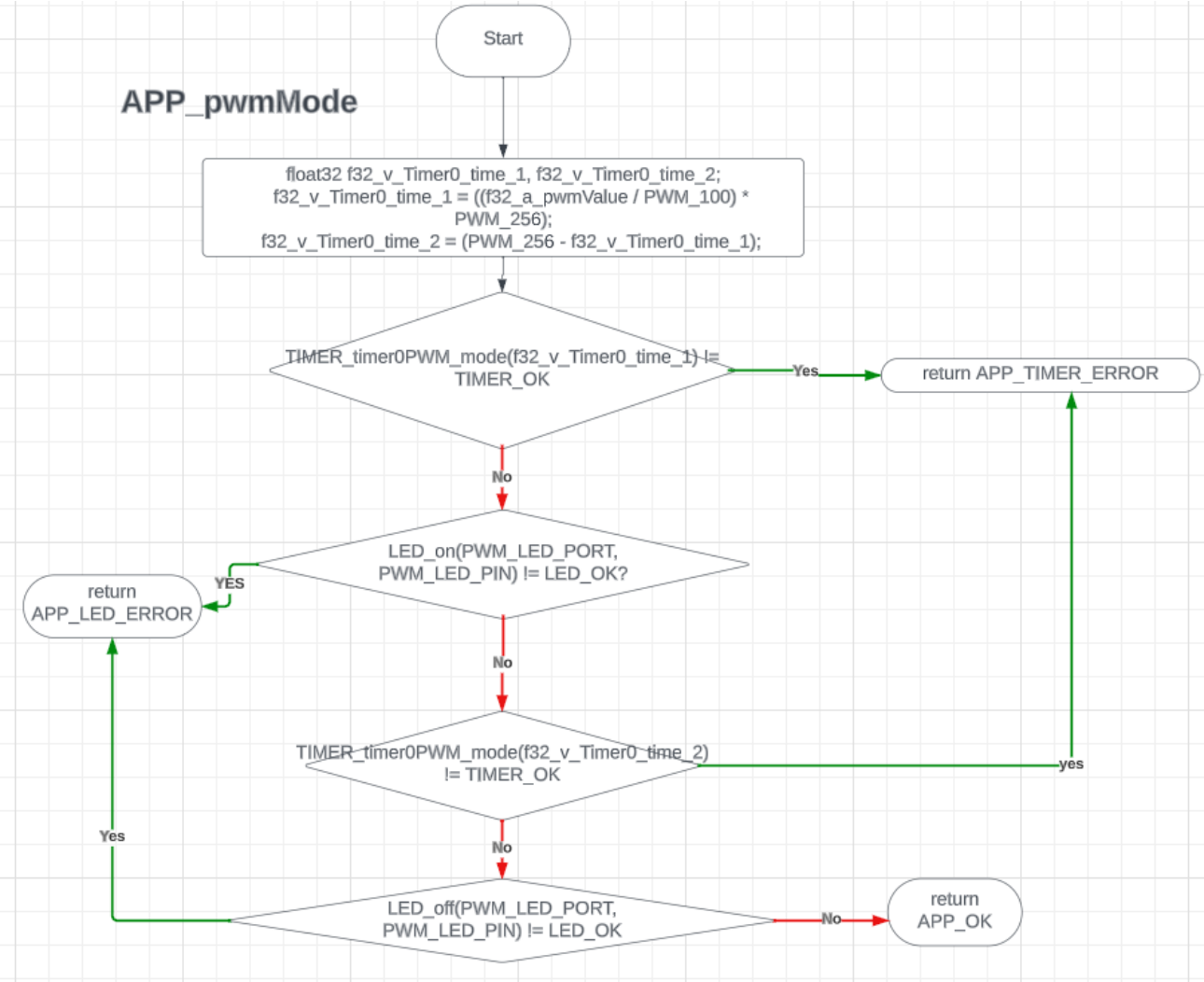

Start
MOTOR_rotate
DIO_write(MOTOR_PORT0, u8_g_motorTest, HIGH) != DIO_OK?
No
Yes
DIO_write(MOTOR_PORT1, MOTOR_PIN1, LOW) != DIO_OK?
No
YES
return MOTOR_INVALID_ROTATE
DIO_write(MOTOR_PORT2, MOTOR_PIN2, LOW) != DIO_OK?
yes
No
DIO_write(MOTOR_PORT3, MOTOR_PIN3, HIGH) != DIO_OK?
yes
No
return MOTOR_OK

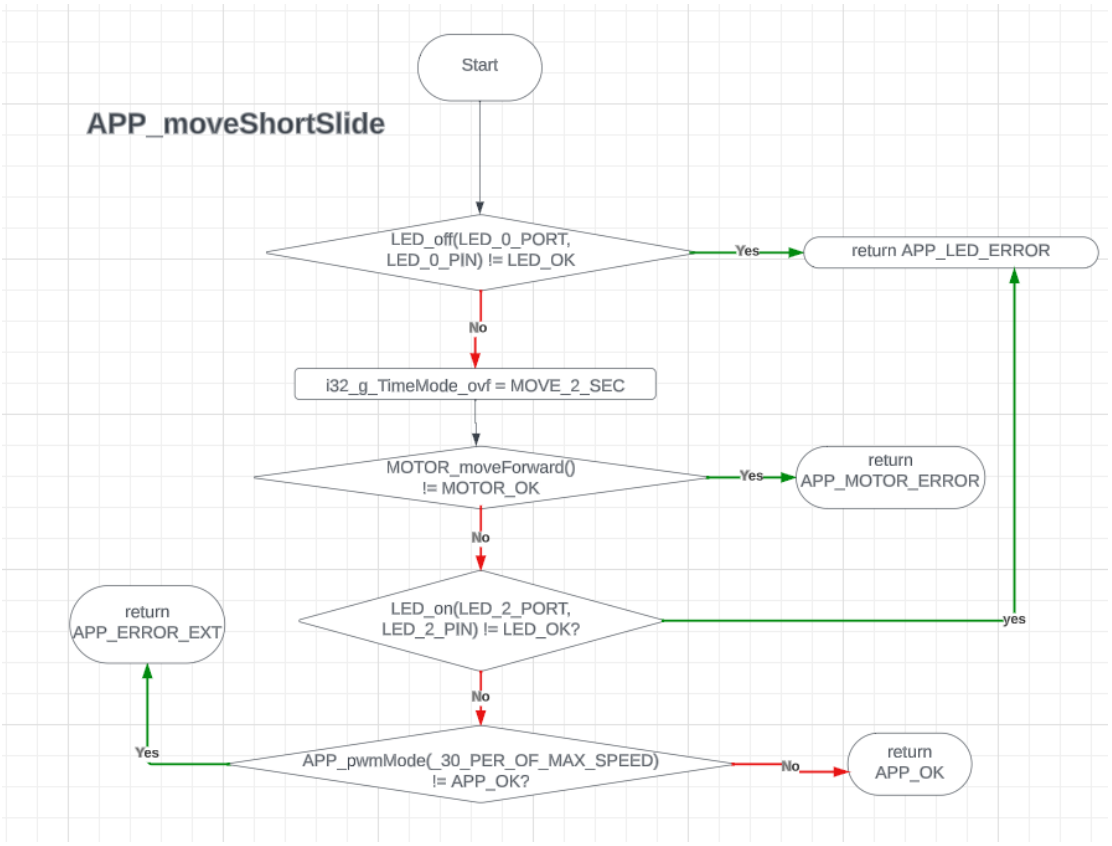

MOTOR_stop
Start
DIO_write(MOTOR_PORT0, MOTOR_PIN0, LOW);
DIO_write(MOTOR_PORT1, MOTOR_PIN1, LOW);
DIO_write(MOTOR_PORT2, MOTOR_PIN2, LOW);
DIO_write(MOTOR_PORT3, MOTOR_PIN3, LOW);

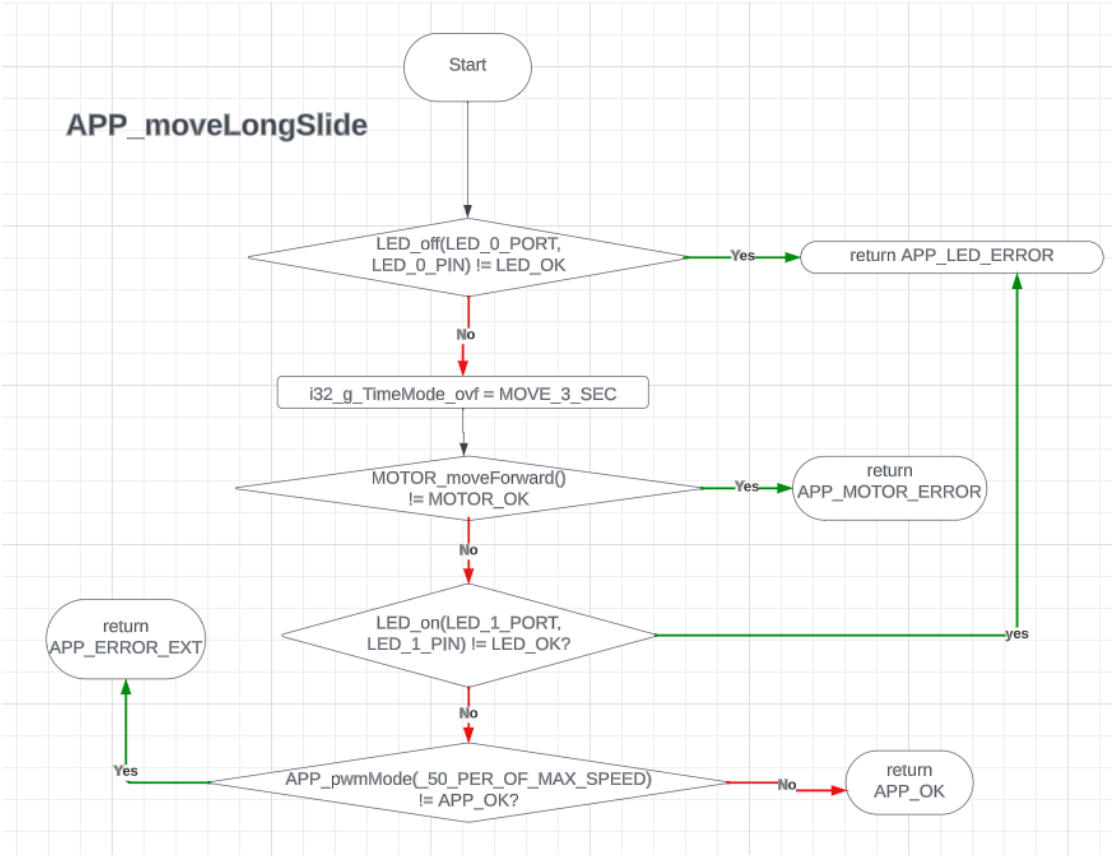
Application APIs flowchart

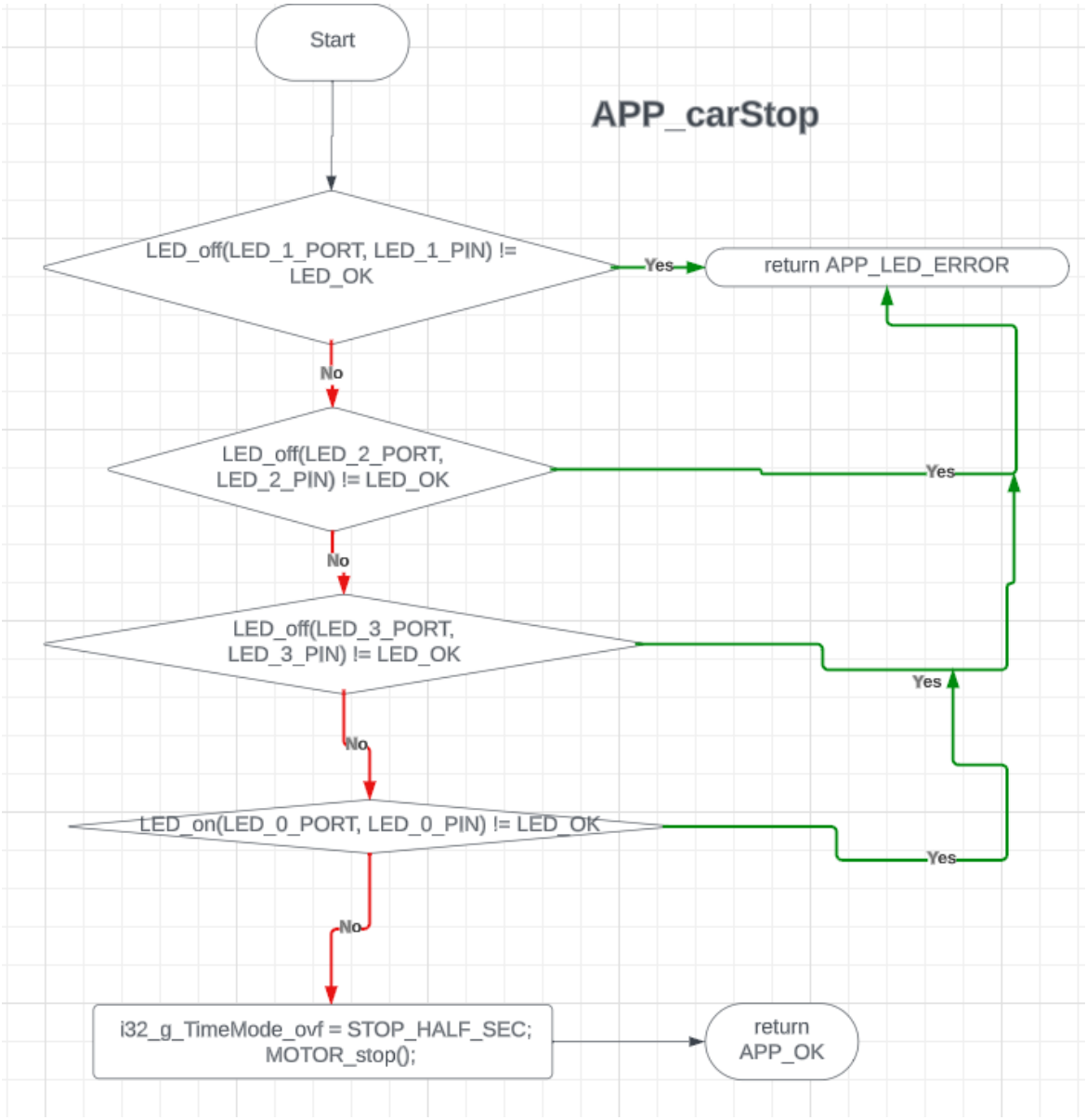
APP_INIT

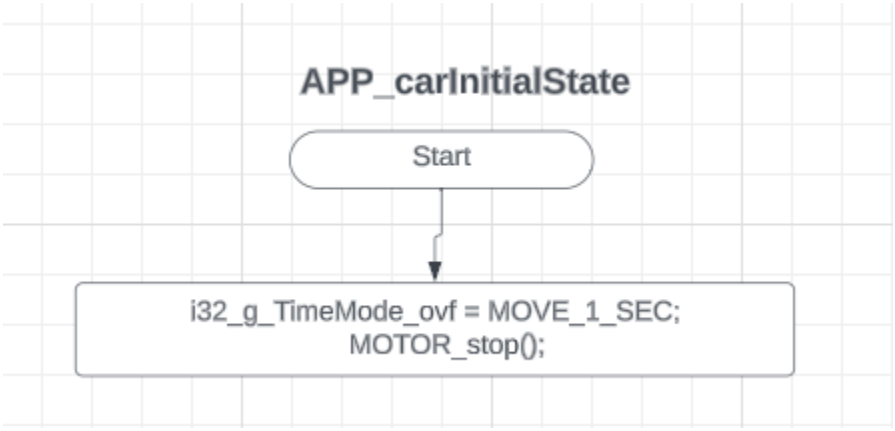


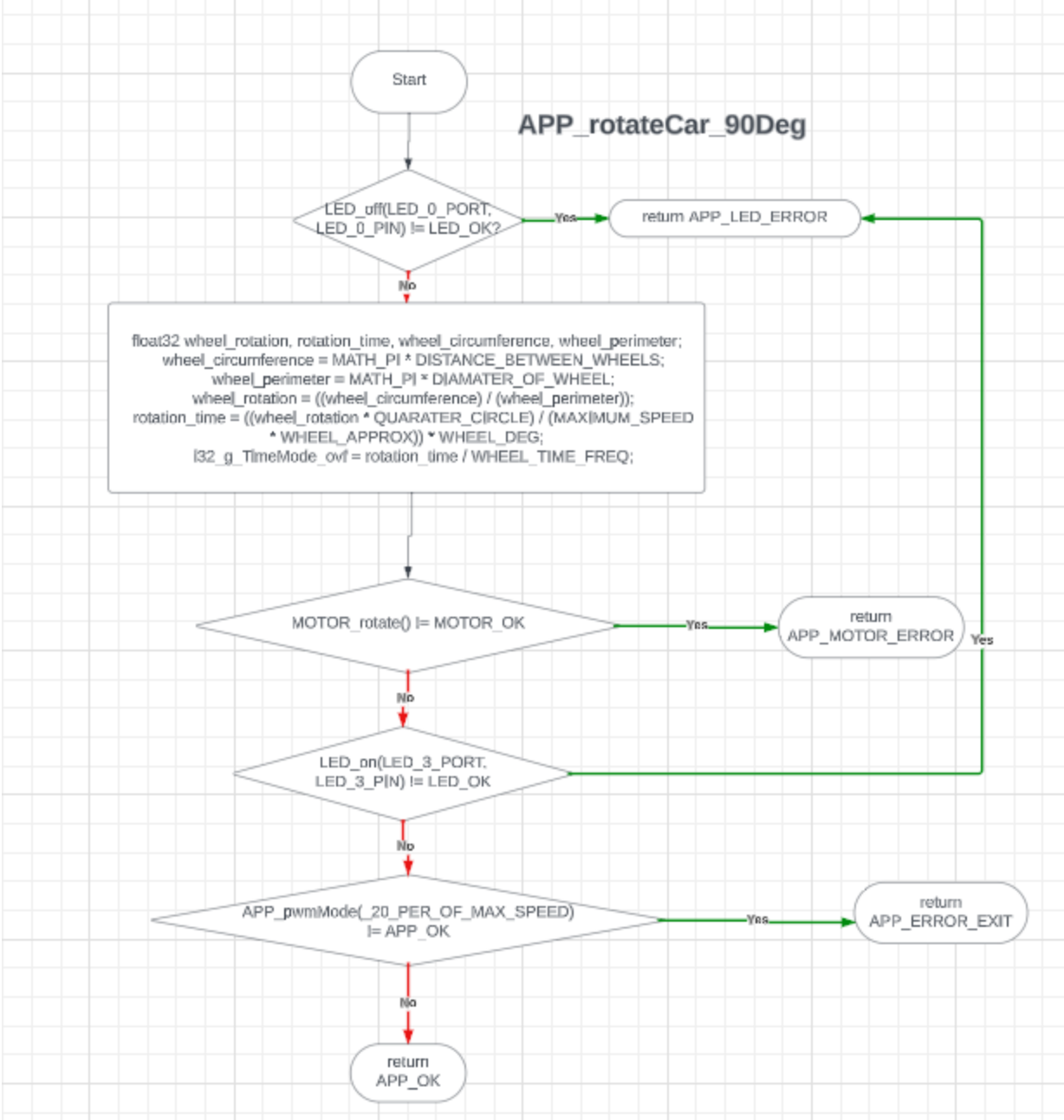

Start
APP_pwmMode
float32 f32_v_Timer0_time_1, f32_v_Timer0_time_2;
f32_v_Timer0_time_1 = ((f32_a_pwmValue / PWM_100) *
PWM_256);
f32_v_Timer0_time_2 = (PWM_256 - f32_v_Timer0_time_1);
TIMER_timer0PWM_mode(f32_v_Timer0_time_1) !=
TIMER_OK
Yes
return APP_TIMER_ERROR
No
LED_on(PWM_LED_PORT,
PWM_LED_PIN) != LED_OK?
YES
return
APP_LED_ERROR
No
TIMER_timer0PWM_mode(f32_v_Timer0_time_2)
!= TIMER_OK
yes
No
Yes
LED_off(PWM_LED_PORT,
PWM_LED_PIN) != LED_OK
No
return
APP_OK


APP_moveShortSlide
Start
LED_off(LED_0_PORT, LED_0_PIN) != LED_OK
Yes
return APP_LED_ERROR
No
i32_g_TimeMode_ovf = MOVE_2_SEC
MOTOR_moveForward() != MOTOR_OK
Yes
return APP_MOTOR_ERROR
No
LED_on(LED_2_PORT, LED_2_PIN) != LED_OK?
yes
No
return APP_ERROR_EXT
Yes
APP_pwmMode(_30_PER_OF_MAX_SPEED) != APP_OK?
No
return APP_OK

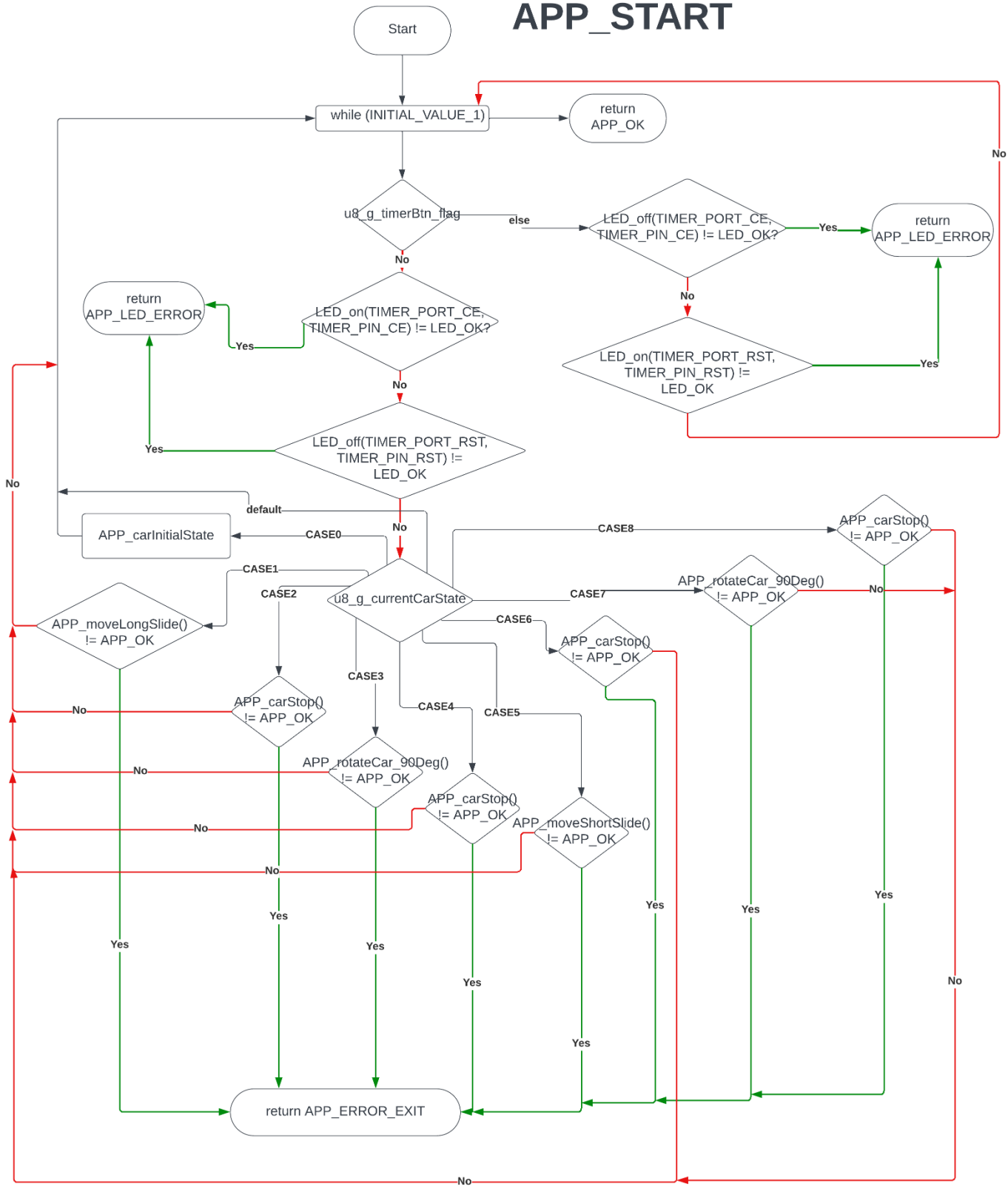

APP_moveLongSlide
Start
LED_off(LED_0_PORT, LED_0_PIN) != LED_OK
Yes
return APP_LED_ERROR
No
i32_g_TimeMode_ovf = MOVE_3_SEC
MOTOR_moveForward() != MOTOR_OK
Yes
return APP_MOTOR_ERROR
No
LED_on(LED_1_PORT, LED_1_PIN) != LED_OK?
yes
No
return APP_ERROR_EXT
Yes
APP_pwmMode(_50_PER_OF_MAX_SPEED) != APP_OK?
No
return APP_OK


APP_carStop
Start
LED_off(LED_1_PORT, LED_1_PIN) != LED_OK
Yes
return APP_LED_ERROR
No
LED_off(LED_2_PORT, LED_2_PIN) != LED_OK
Yes
No
LED_off(LED_3_PORT, LED_3_PIN) != LED_OK
Yes
No
LED_on(LED_0_PORT, LED_0_PIN) != LED_OK
Yes
No
i32_g_TimeMode_ovf = STOP_HALF_SEC;
MOTOR_stop();
return APP_OK


APP_carInitialState
Start
i32_g_TimeMode_ovf = MOVE_1_SEC;
MOTOR_stop();


Start
APP_rotateCar_90Deg
LED_off(LED_0_PORT, LED_0_PIN) != LED_OK?
Yes
return APP_LED_ERROR
No
float32 wheel_rotation, rotation_time, wheel_circumference, wheel_perimeter;
wheel_circumference = MATH_PI * DISTANCE_BETWEEN_WHEELS;
wheel_perimeter = MATH_PI * DIAMATER_OF_WHEEL;
wheel_rotation = ((wheel_circumference) / (wheel_perimeter));
rotation_time = ((wheel_rotation * QUARATER_CIRCLE) / (MAXIMUM_SPEED * WHEEL_APPROX)) * WHEEL_DEG;
I32_g_TimeMode_ovf = rotation_time / WHEEL_TIME_FREQ;
MOTOR_rotate() != MOTOR_OK
Yes
return APP_MOTOR_ERROR
Yes
No
LED_on(LED_3_PORT, LED_3_PIN) != LED_OK
No
APP_pwmMode(_20_PER_OF_MAX_SPEED) != APP_OK
Yes
return APP_ERROR_EXIT
No
return APP_OK

APP_START


Start
while (INITIAL_VALUE_1)
return APP_OK
u8_g_timerBtn_flag
else
LED_off(TIMER_PORT_CE, TIMER_PIN_CE) != LED_OK?
Yes
return APP_LED_ERROR
No
LED_on(TIMER_PORT_RST, TIMER_PIN_RST) != LED_OK
Yes
No
LED_on(TIMER_PORT_CE, TIMER_PIN_CE) != LED_OK?
Yes
return APP_LED_ERROR
No
LED_off(TIMER_PORT_RST, TIMER_PIN_RST) != LED_OK
Yes
No
default
APP_carInitialState
CASE0
No
u8_g_currentCarState
CASE1
APP_moveLongSlide() != APP_OK
CASE2
APP_carStop() != APP_OK
No
CASE3
APP_rotateCar_90Deg() != APP_OK
No
CASE4
APP_carStop() != APP_OK
No
CASE5
APP_moveShortSlide() != APP_OK
No
CASE6
APP_carStop() != APP_OK
CASE7
APP_rotateCar_90Deg() != APP_OK
No
CASE8
APP_carStop() != APP_OK
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
No
No
No
No
No
return APP_ERROR_EXIT