Small OS design

Sarah Mohamed Salah

1. Project Introduction

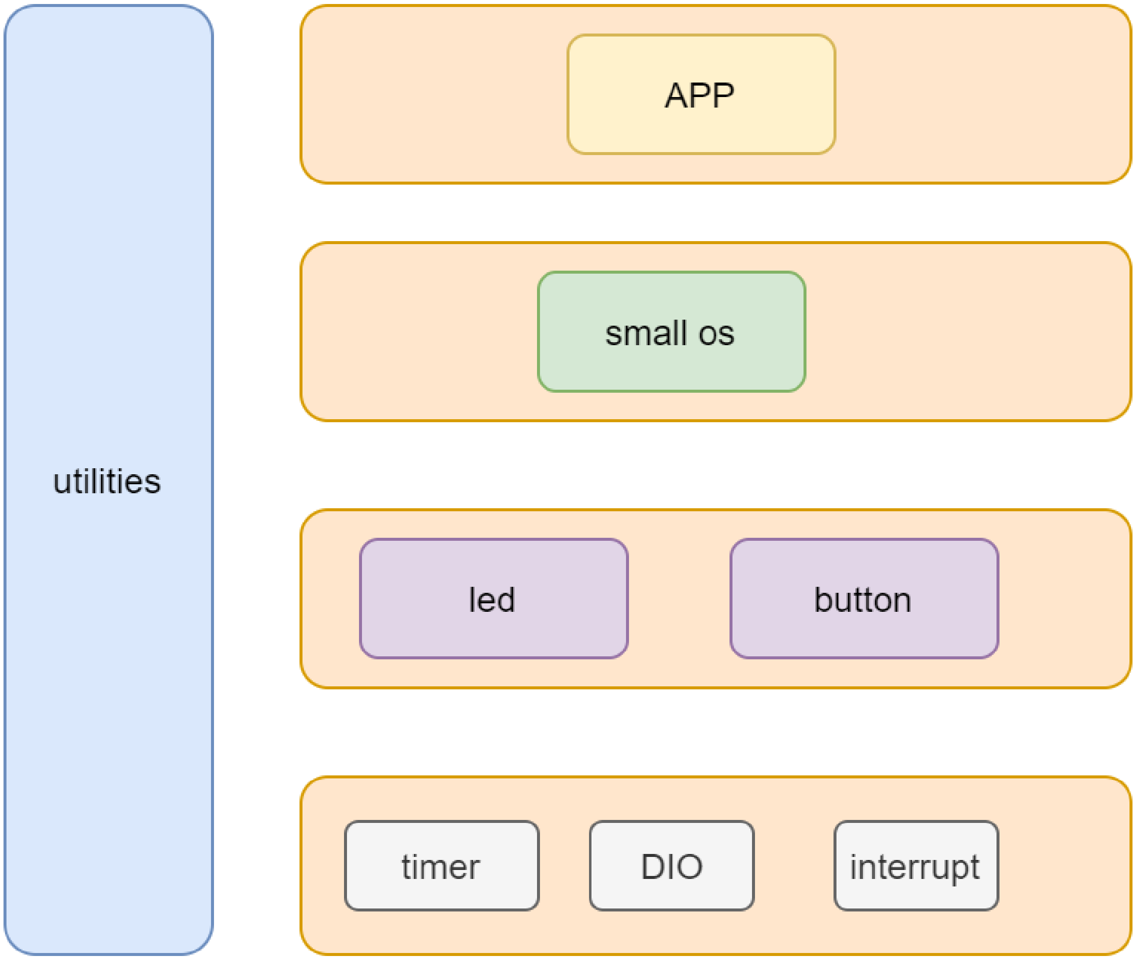
Project aims to make priority based small OS time trrigerred

1.1. Project Components

- ATmega32 microcontroller
- TWO leds
- TWO button

2. High Level Design

2.1. System Architecture



2.2. Modules Description

2.2.1. DIO (Digital Input/Output) Module

The *DIO* module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

2.2.2. LED Module

The *LED* module is responsible for turning on or off based on signal of microcontroller

2.2.3. BTN Module

The *BTN* (Button) module is responsible for reading the state of the system's buttons. It provides a set of APIs to enable/disable button interrupts, set the button trigger edge (rising/falling/both), and define an ISR that will be executed when a button press is detected.

2.2.4. EXI Module

The *EXI* (External Interrupt) module is responsible for detecting external events that require immediate attention from the microcontroller, such as a button press. It provides a set of APIs to enable/disable external interrupts for specific pins, set the interrupt trigger edge (rising/falling/both), and define an interrupt service routine (*ISR*) that will be executed when the interrupt is triggered.

2.2.5. TIMER Module

The *TIMER* module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs.

2.3. Drivers' Documentation (APIs)

2.3.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.3.2. MCAL APIs

2.3.2.1. DIO Driver

```
|
|  Function to set the direction of a given port
|
| This function takes an 8-bit value and sets the direction of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the direction of
|          [in] u8_a_portDir The desired port direction
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortDir(en_DIO_port_t en_a_port,  u8 u8_a_portDir);
|
|  Function to set the value of a given port
|
| This function takes an 8-bit value and sets the value of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the value of
|          [in] u8_a_portVal The desired port value
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortVal(en_DIO_port_t en_a_port,  u8 u8_a_portVal);
```

```
|
|  Function to set the direction of a given pin
|
| This function takes an en_DIO_pinDir_t value and sets the direction
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set direction of
|          [in] en_a_pinDir The desired pin direction (INPUT/OUTPUT)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinDir (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_pinDir_t en_a_pinDir);
```

```
|
|  Function to set the value of a given pin
|
| This function takes an en_DIO_level_t value and sets the value
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set value of
|          [in] en_a_pinDir The desired pin value (HIGH/LOW)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_level_t en_a_pinVal);
```

```
|
|  Function to toggle the value of a given pin
|
| If the pin value is high, this function sets it to low
| and if it is low it sets it to high
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to toggle value of
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_togPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t
en_a_pin);
```

```
|
|  Function to get the value of a given pin
|
| This function reads the value of the given pin and
| returns the value in the given address
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to read value of
|          [out] pu8_a_Val   address to return the pin value into
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_getPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
u8* pu8_a_Val);
```

2.3.2.2. OS APIs

```
|  description      : func to initialize sos
| Parameters
|                 void
| Return
|      en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_init(void);
|  description      : func to deinitialize sos
| Parameters
|                 void
| Return
|      en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_deinit(void);
|  description      : func to create task in  sos
| Parameters
|                 void
| Return
|      en_buttonError_t if the sos was read successfully,
enu_system_status_t SOS_create_task(void);
|  description      : func to modify task in  sos
| Parameters
|                 void
| Return
|     en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_modify_task(void);
|  description      : func to delete task in  sos
| Parameters
|                 void
| Return
|     en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_delete_task(void);
|  description      : func to run in  sos
| Parameters
|                 void
| Return
|     en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_run(void);
|  description      : func to disable task in  sos
| Parameters
|                 void
| Return
|     en_buttonError_t if the sos was read successfully,

enu_system_status_t SOS_disable(void);
```

2.3.2.3. EXI Driver

```
|
| Initializes given EXI as configured
|
| This function initializes the passed interrupt with the configured
| parameters in the configuration source file
|
| Parameters
|          [in] en_a_IntNumber the interrupt to be initialized
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_init(en_EXI_num_t en_a_intNumber);

|
|  Function to choose the trigger event for given EXI
|
| This function sets the given EXI to be triggered whenever
| an event that matches the given sense mode occurs
|
| Parameters
|          [in] en_a_IntNumber The interrupt to be configured
|          [in] en_a_SenseMode The event to trigger the EXI
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setSense(en_EXI_num_t en_a_intNumber, en_EXI_senseMode_t
en_a_senseMode);
```

```
|
| Function to enable/disable given EXI
|
| This function sets or clears the specific interrupt enable bit
| for the given interrupt to enable or disable it
|
| Parameters
|          [in] en_a_IntNumber The interrupt to be configured
|          [in] en_a_intState  EXI state (EXI_ENABLE/EXI_DISABLE)
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setState(en_EXI_num_t en_a_IntNumber, en_EXI_state_t
en_a_intState);

|
| Function to set a function to call when EXI is triggered
|
| This function sets a callback function to be called whenever
| the given interrupt is triggered
|
| Parameters
|          [in] en_a_IntNumber The desired EXI number
|          [in] pv_a_Function  The function to call
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setCallback(en_EXI_num_t en_a_IntNumber, void
(*pv_a_Function)(void));
```

2.3.2.4. TIMER Driver

```
|
| Syntax          : en_TIMER_error_t TIMER_init( void )
| Description     : Initialize Timer according to preprocessed
|                    configured definitions
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value    : en_TIMER_error_t               TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|

en_TIMER_error_t TIMER_init( void );

|
|  Syntax          : en_TIMER_error_t TIMER_setTime
|                   (en_TIMER_number_t en_a_timerUsed, f32 f32_a_desiredTime)
|  Description     : set the time at which the timer interrupts
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                   f32                            f32_a_desiredTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|

en_TIMER_error_t TIMER_setTime(en_TIMER_number_t en_a_timerUsed, f32 f32_a_desiredTime);

|
|  Syntax          : en_TIMER_error_t TIMER_pwmGenerator
|             (en_TIMER_number_t en_a_timerUsed , u16 u16_a_onTime, u16 u16_a_offTime)
|  Description     : initialize the timer to generates pwm signal using
|                    normal mode
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                      u16                         u16_a_onTime
|                      u16                         u16_a_offTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_pwmGenerator(en_TIMER_number_t en_a_timerUsed , u16 u16_a_onTime, u16
u16_a_offTime);



|
| Syntax           : en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed)
| Description      : makes the timer to start/resume counting
| Sync\Async       : Synchronous
| Reentrancy       : Reentrant
| Parameters (in)  : en_TIMER_number_t              en_a_timerUsed
| Parameters (out) : None
| Return value:    : en_TIMER_error_t               TIMER_OK = 0
|                                                   TIMER_WRONG_TIMER_USED = 1
|                                                   TIMER_WRONG_DESIRED_TIME = 2
|                                                   TIMER_NOK = 3
|
en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed);

|
| Syntax           : en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed)
| Description      : makes the timer to reset counting from the beginning
| Sync\Async       : Synchronous
| Reentrancy       : Reentrant
| Parameters (in)  : en_TIMER_number_t              en_a_timerUsed
| Parameters (out) : None
| Return value:    : en_TIMER_error_t               TIMER_OK = 0
|                                                   TIMER_WRONG_TIMER_USED = 1
|                                                   TIMER_WRONG_DESIRED_TIME = 2
|                                                   TIMER_NOK = 3
|
en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed);

|
| Syntax           : en_TIMER_error_t TIMER_getElapsedTime
|                      (en_TIMER_number_t en_a_timerUsed, u32* u32_a_elapsedTime)
| Description      : returns the elapsed time since the timer started
|                     from the beginning in microseconds
| Sync\Async       : Synchronous
| Reentrancy       : Reentrant
| Parameters (in)  : en_TIMER_number_t              en_a_timerUsed
| Parameters (out) : u32                            u32_a_elapsedTime
| Return value:    : en_TIMER_error_t               TIMER_OK = 0
|                                                   TIMER_WRONG_TIMER_USED = 1
|                                                   TIMER_WRONG_DESIRED_TIME = 2
|                                                   TIMER_NOK = 3
|
en_TIMER_error_t TIMER_getElapsedTime(en_TIMER_number_t en_a_timerUsed, u32*
u32_a_elapsedTime);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to pause counting
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t
en_a_timerUsed)
| Description     : Disables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed)
| Description     : Enables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_setCallBack
|                     (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : sets the call back function for a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
|                    void                          (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setCallBack(en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void));

|
| Syntax          : en_TIMER_error_t TIMER_setDelayTime
|                                    (en_TIMER_number_t en_a_timerUsed, f32 f32_a_timeInMS)
| Description     : Set delay time for blocking delay using a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
|                      f32                         f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                                  TIMER_WRONG_TIMER_USED = 1
|                                                  TIMER_WRONG_DESIRED_TIME = 2
|                                                  TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setDelayTime(en_TIMER_number_t en_a_timerUsed, f32 f32_a_timeInMS);

|
| Syntax          : en_TIMER_error_t TIMER_setPwmOnCallBack
|                                    (en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void))
| Description     : Set callback function for the task done while signal is high
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
|                      void                            (*funPtr)(void)
| Parameters (out): None
| Return value:    : en_TIMER_error_t        TIMER_OK = 0
|                                                   TIMER_WRONG_TIMER_USED = 1
|                                                   TIMER_WRONG_DESIRED_TIME = 2
|                                                   TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOnCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));







|
| Syntax           : en_TIMER_error_t TIMER_setPwmOffCallBack
|                                    (en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void))
| Description      : Set callback function for the task done while signal is low
| Sync\Async       : Synchronous
| Reentrancy       : Reentrant
| Parameters (in) : en_TIMER_number_t               en_a_timerUsed
|                      void                            (*funPtr)(void)
| Parameters (out): None
| Return value:    : en_TIMER_error_t        TIMER_OK = 0
|                                                   TIMER_WRONG_TIMER_USED = 1
|                                                   TIMER_WRONG_DESIRED_TIME = 2
|                                                   TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOffCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));
```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
|  description      : func to initialize led
| Parameters
|              u8_a_buttonPort:  read port number.
|              u8_a_buttonPin :  read pin number .
| Return
|     en_buttonError_t if the led state was read successfully,

en_ledError_t LED_init(u8 u8_a_buttonPort, u8 u8_a_buttonPin);

|  description      : func to turn on led
| Parameters
|              u8_a_buttonPort:  read port number.
|              u8_a_buttonPin :  read pin number .
|              u8_a_buttonState: pointer to read led state
| Return
|     en_buttonError_t if the led was on  successfully,

en_ledError_t LED_on(u8 u8_a_buttonPort, u8 u8_a_buttonPin);

|  description      : func to turn off led
| Parameters
|              u8_a_buttonPort:  read port number.
|              u8_a_buttonPin :  read pin number .
|              u8_a_buttonState: pointer to read led state
| Return
|     en_buttonError_t if the led was off  successfully,

en_ledError_t LED_off(u8 u8_a_buttonPort, u8 u8_a_buttonPin);
```

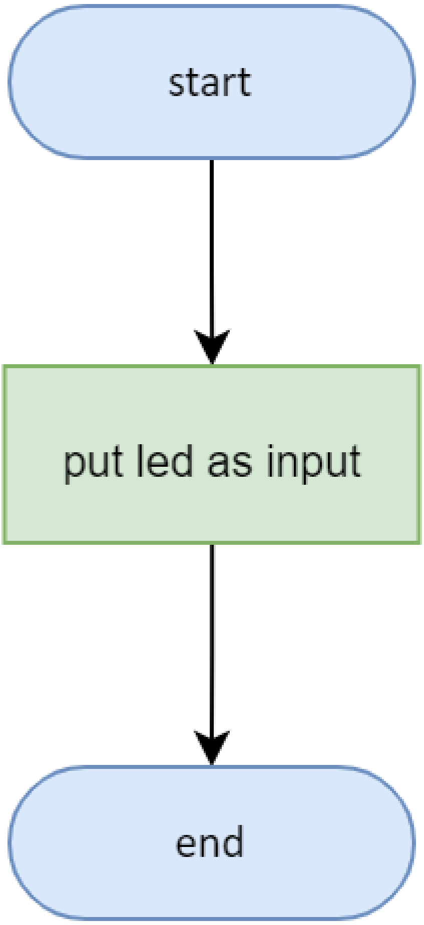
2.3.3.2. BTN APIs

```
|  description      : func to initialize button
| Parameters
|              u8_a_buttonPort:  read port number.
|              u8_a_buttonPin :  read pin number .
| Return
|     en_buttonError_t if the button state was read successfully,

en_buttonError_t BUTTON_init(u8 u8_a_buttonPort, u8 u8_a_buttonPin);

|  description      : func to read button
| Parameters
|              u8_a_buttonPort:  read port number.
|              u8_a_buttonPin :  read pin number .
|              u8_a_buttonState: pointer to read button state
| Return
|     en_buttonError_t if the button state was read successfully,

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

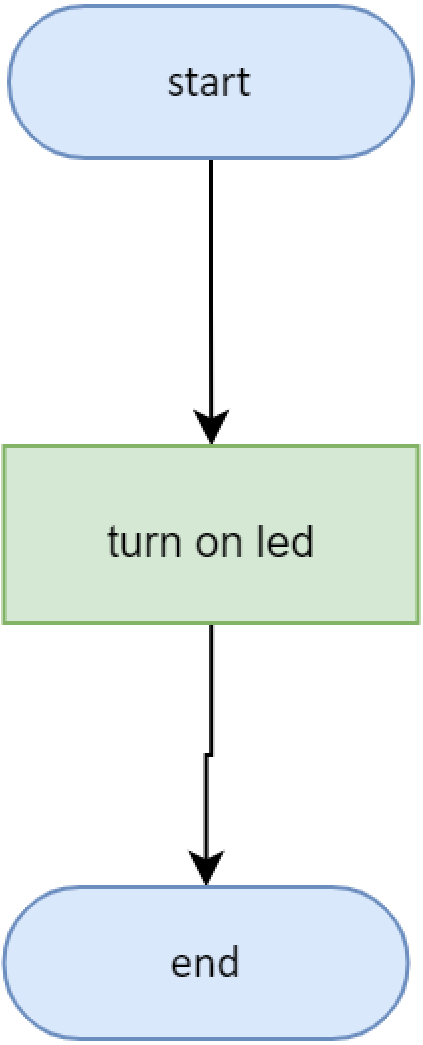
3.2. HAL Layer

3.2.1. LED Module

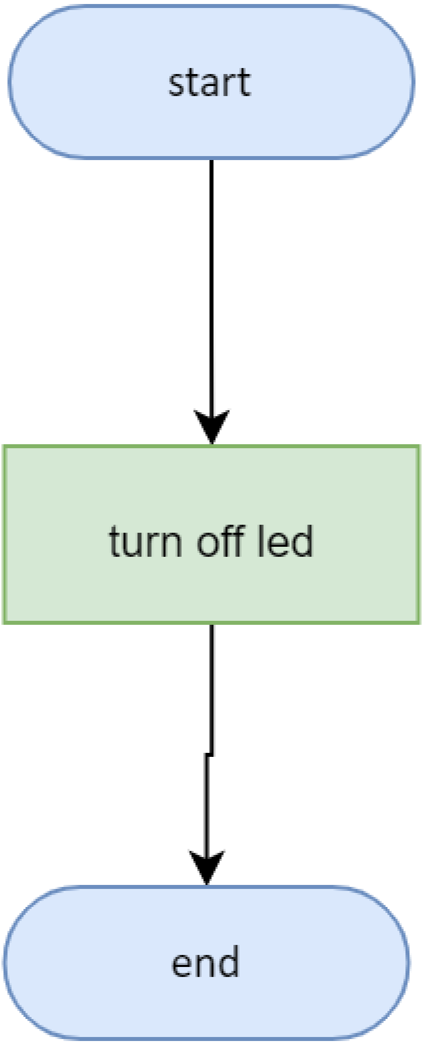
3.2.1.1. LED_init



3.2.1.1. LED_on

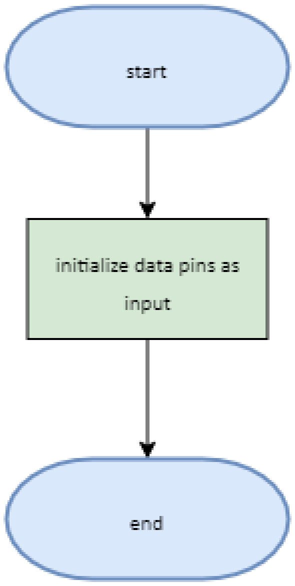


3.2.1.2. LED_off

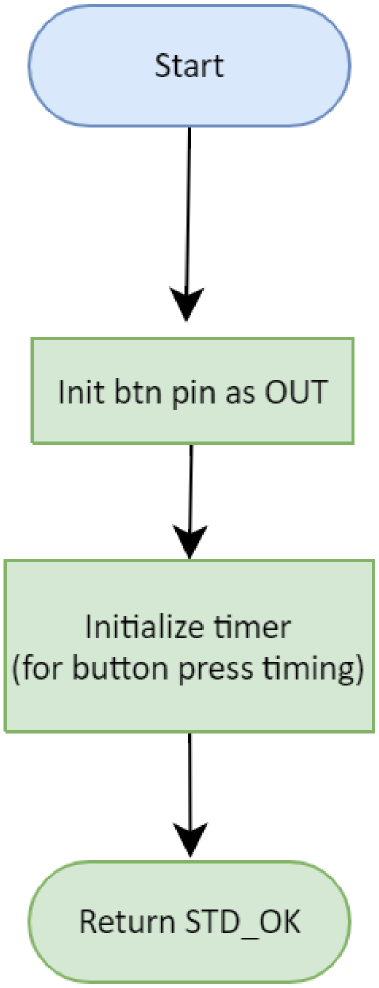


3.2.2. BTN Module

3.2.2.1. BUTTON_init

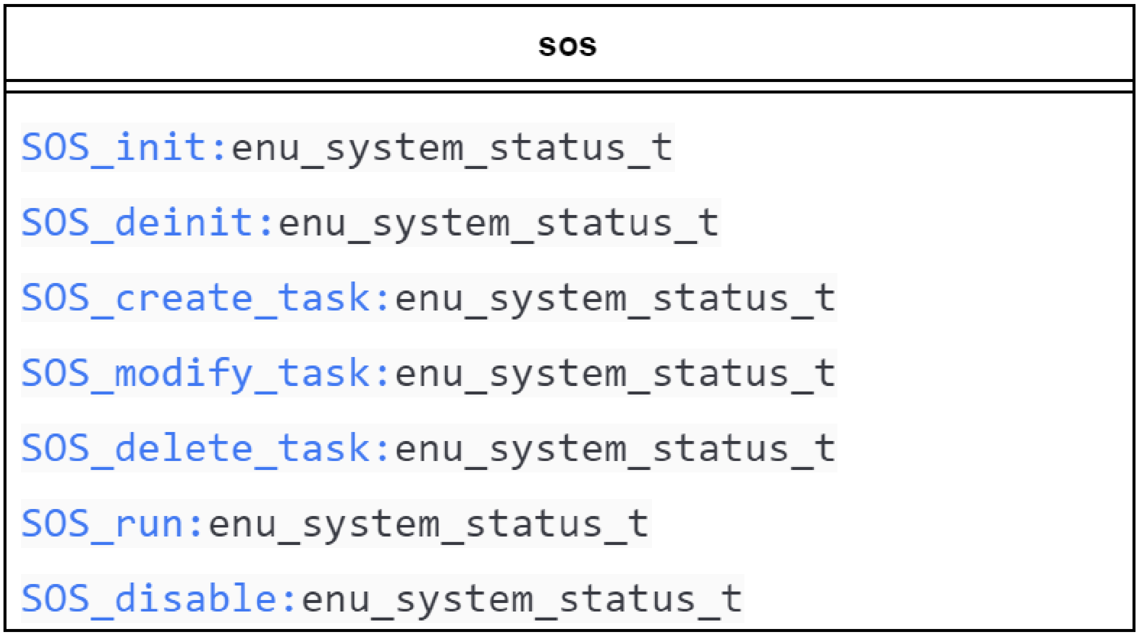

start
initialize data pins as input
end

3.2.2.2. BUTTON_read



2.4. UML

2.4.1 UML DIAGRAM


sos
SOS_init:enu_system_status_t
SOS_deinit:enu_system_status_t
SOS_create_task:enu_system_status_t
SOS_modify_task:enu_system_status_t
SOS_delete_task:enu_system_status_t
SOS_run:enu_system_status_t
SOS_disable:enu_system_status_t

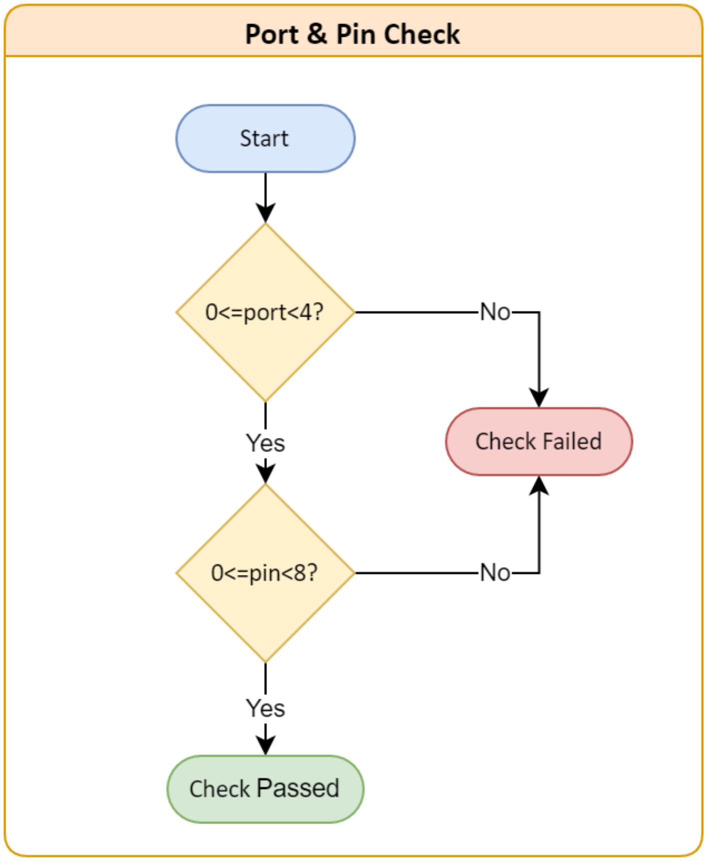
2.5. Sequence diagram

3. Low Level Design

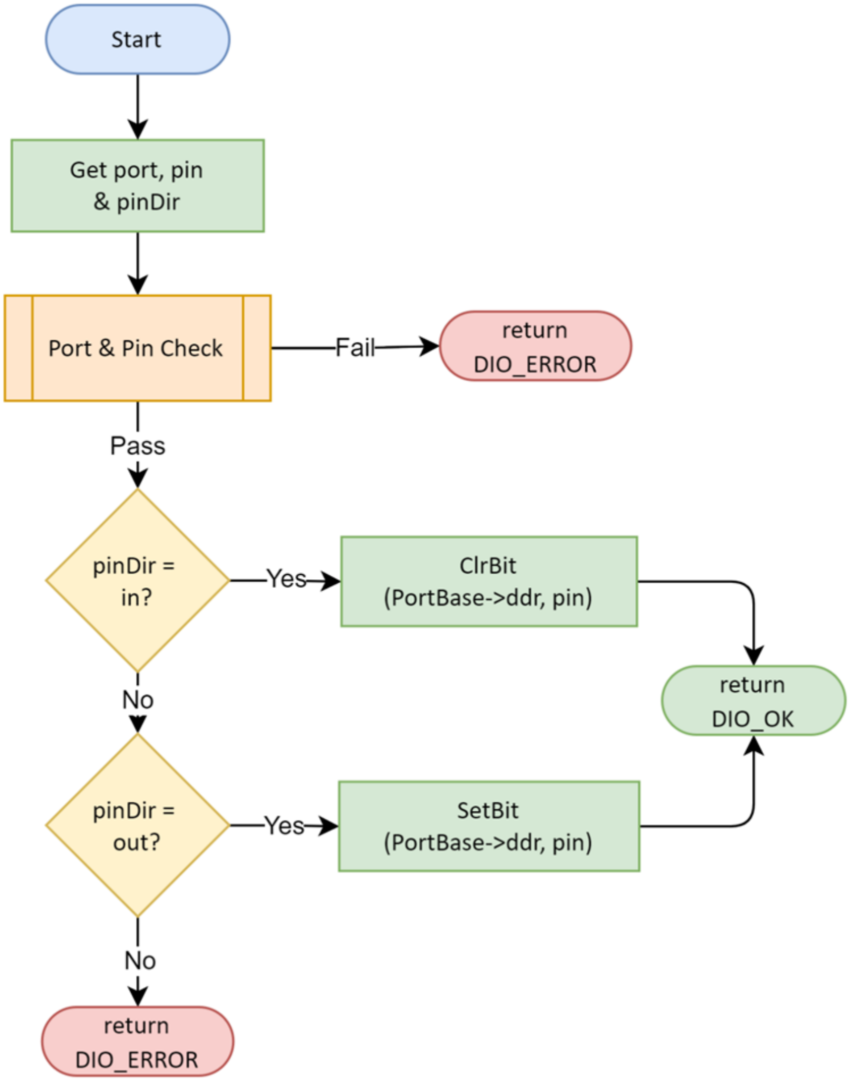
3.1. MCAL Layer

3.1.1. DIO Module

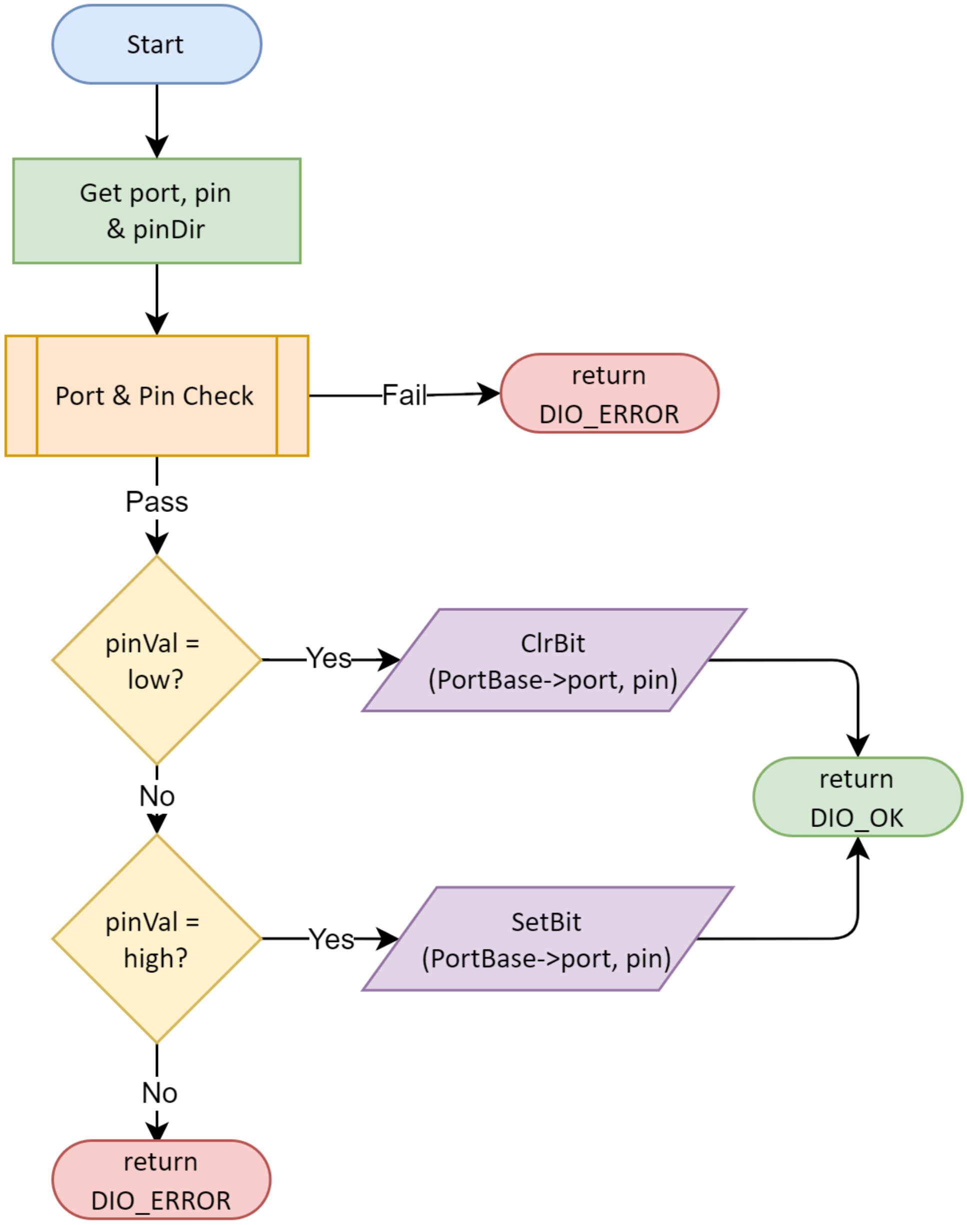
3.1.1.a. sub process



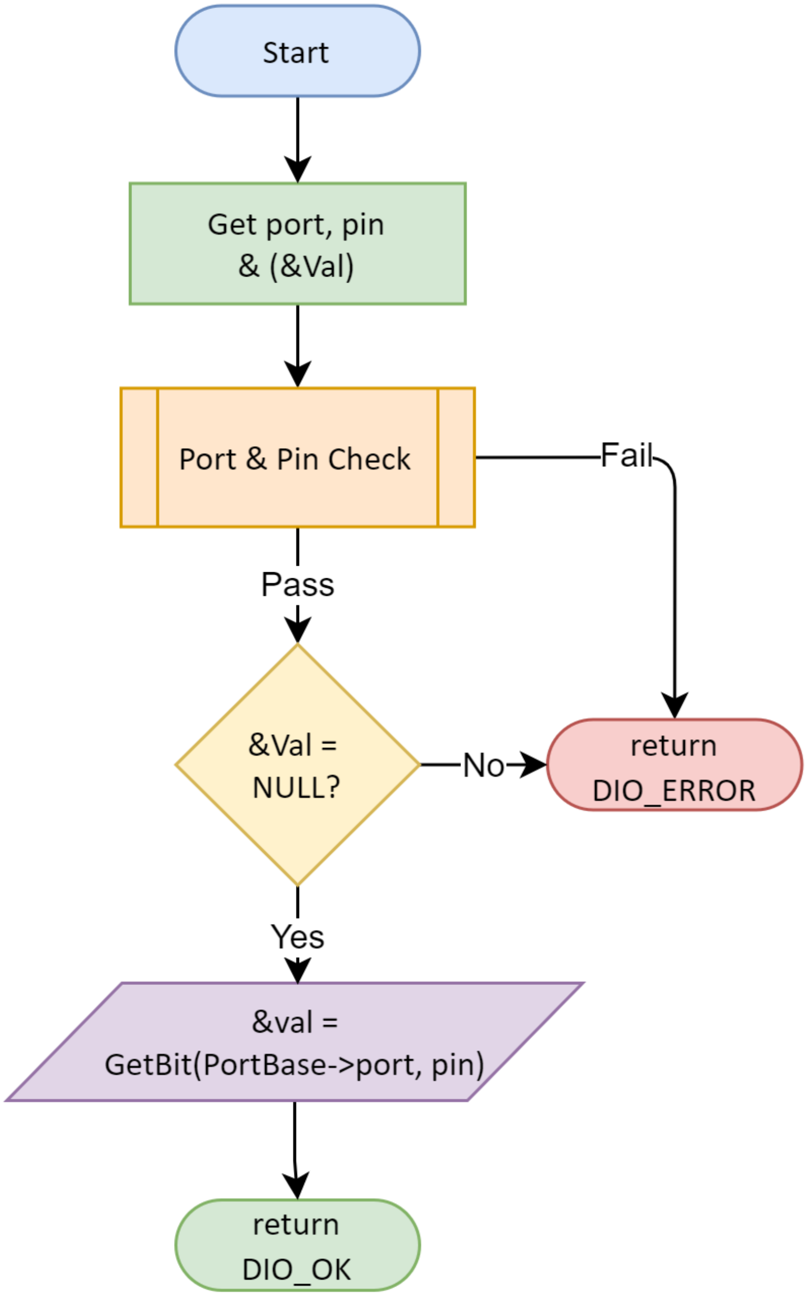
3.1.1.1. DIO_setPinDir



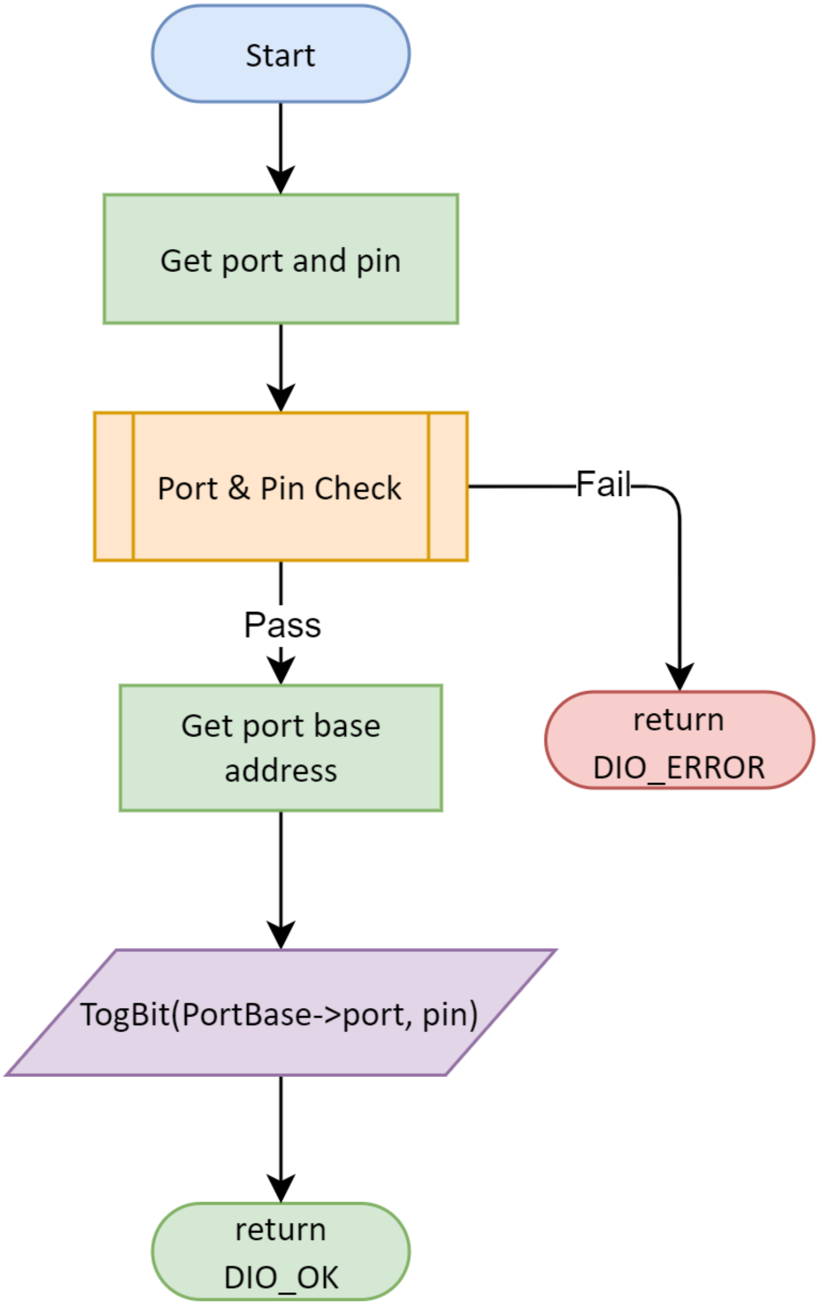
3.1.1.2. DIO_setPinVal



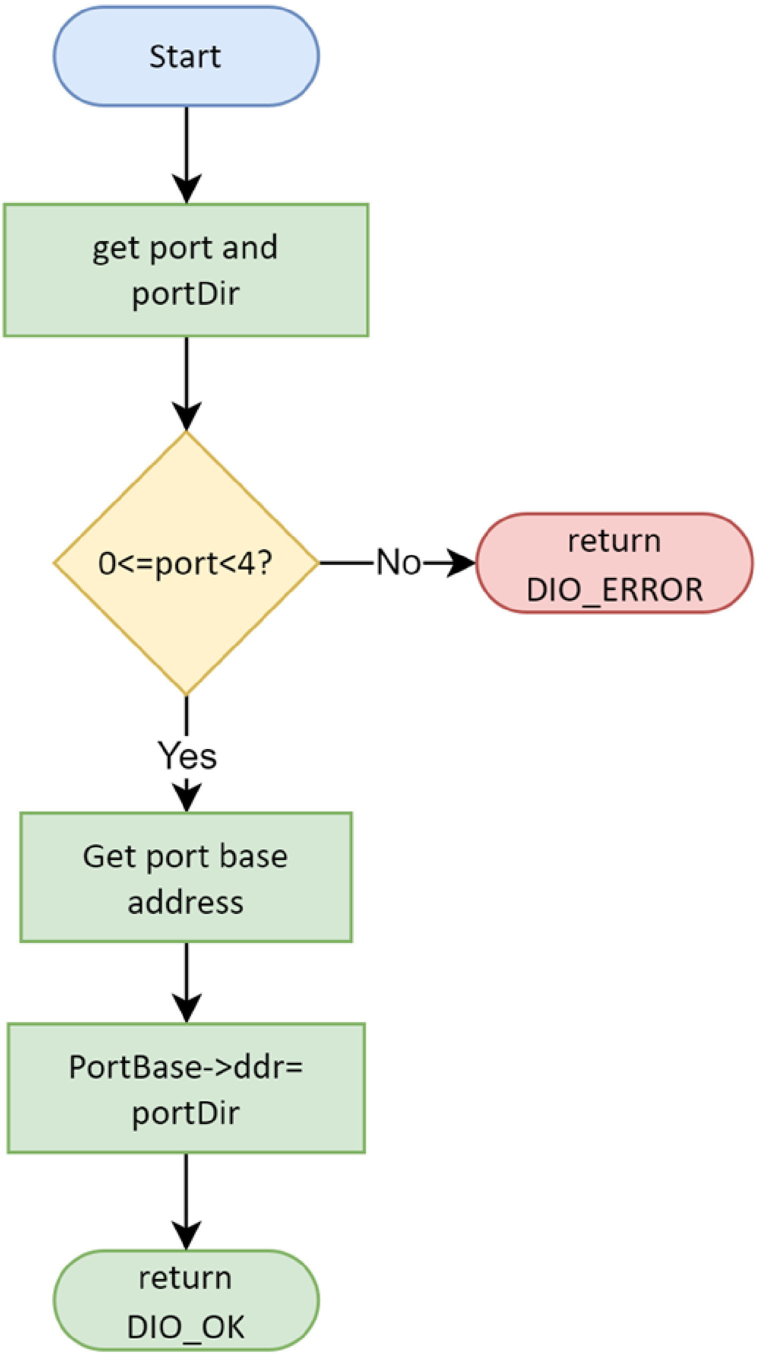
3.1.1.3. DIO_getPinVal



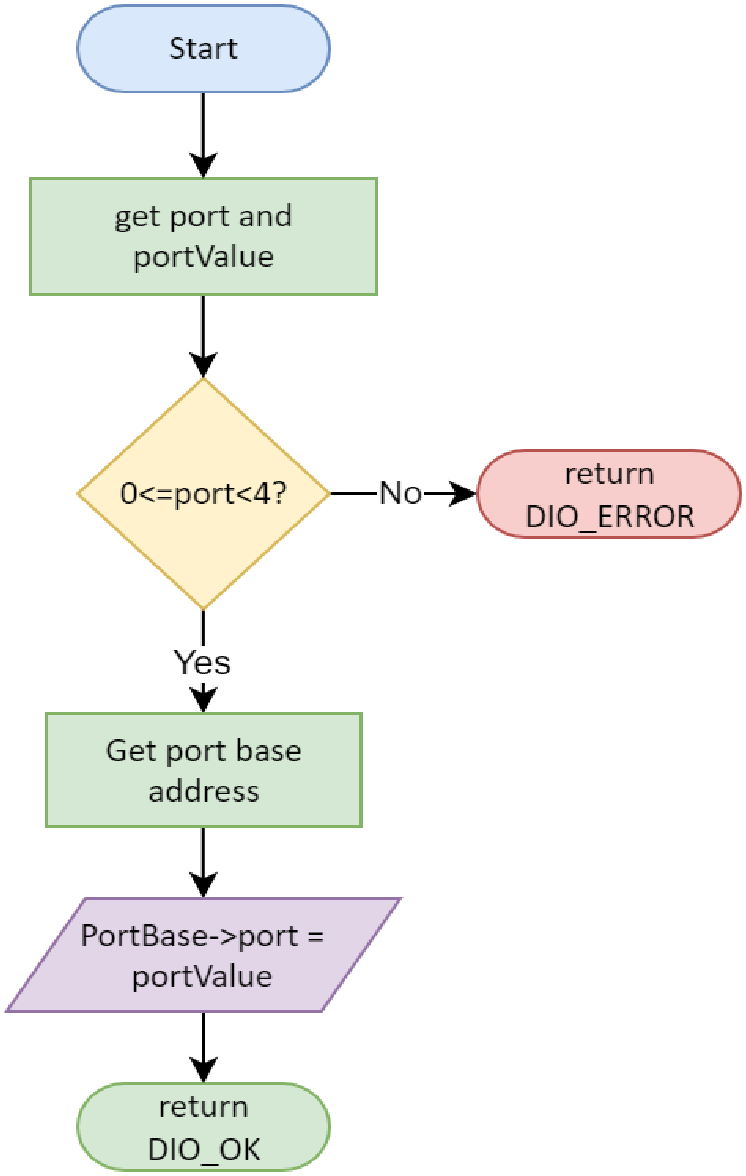
3.1.1.4. DIO_togPinVal



3.1.1.5. DIO_setPortDir

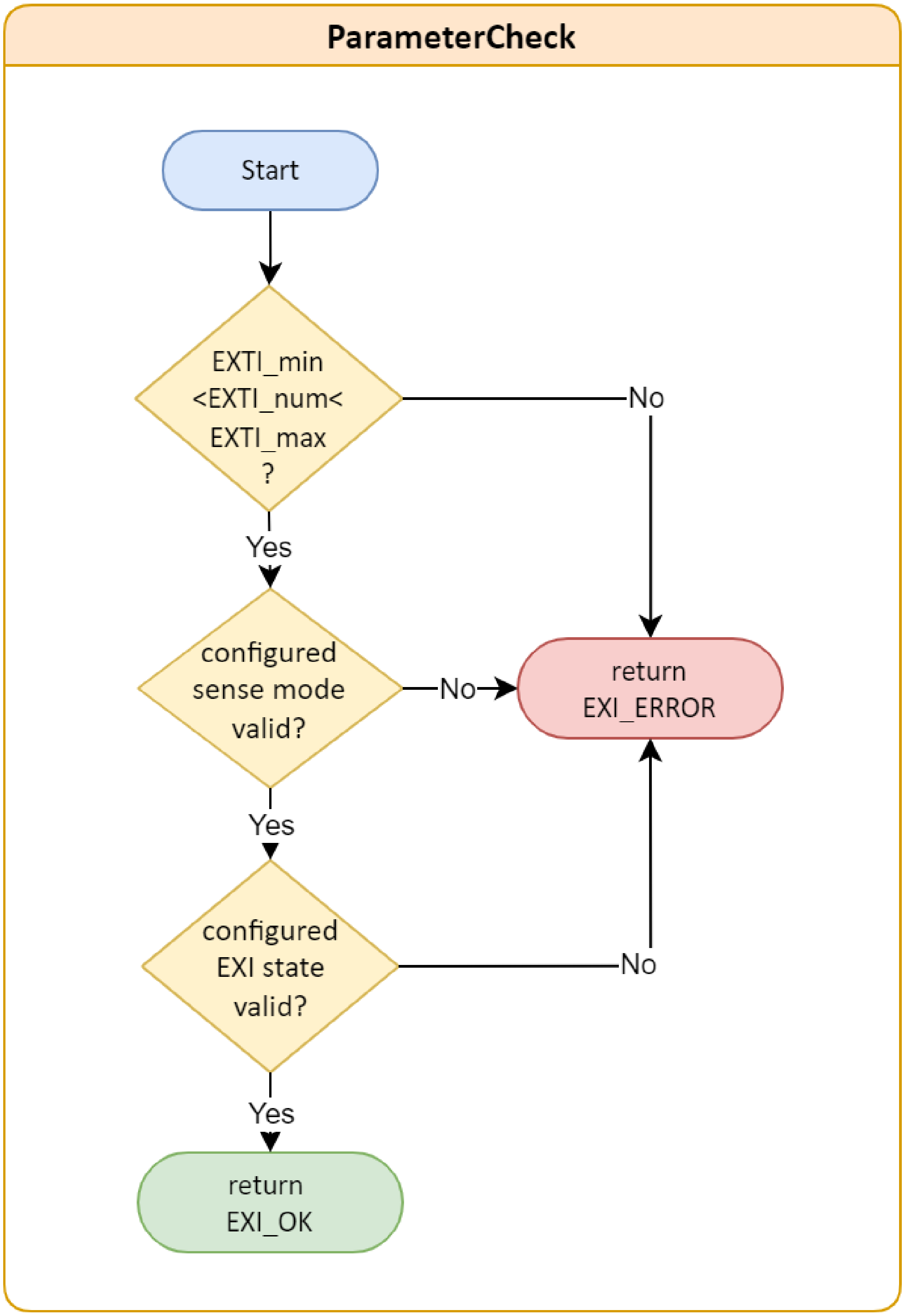


3.1.1.6. DIO_setPortVal

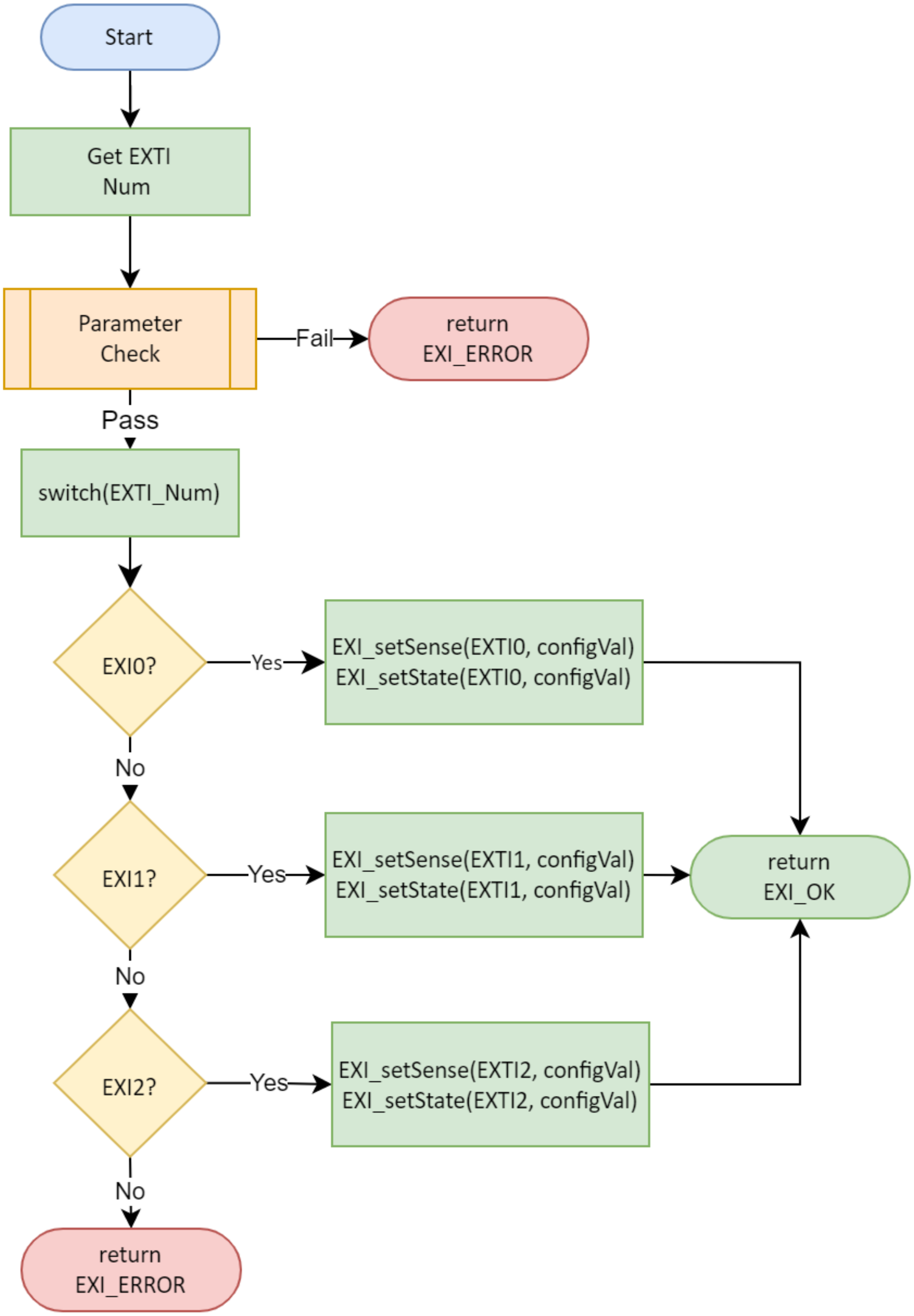

Start
get port and portValue
0<=port<4?
No
return DIO_ERROR
Yes
Get port base address
PortBase->port = portValue
return DIO_OK

3.1.2. EXI Module

3.1.2.a. Sub process

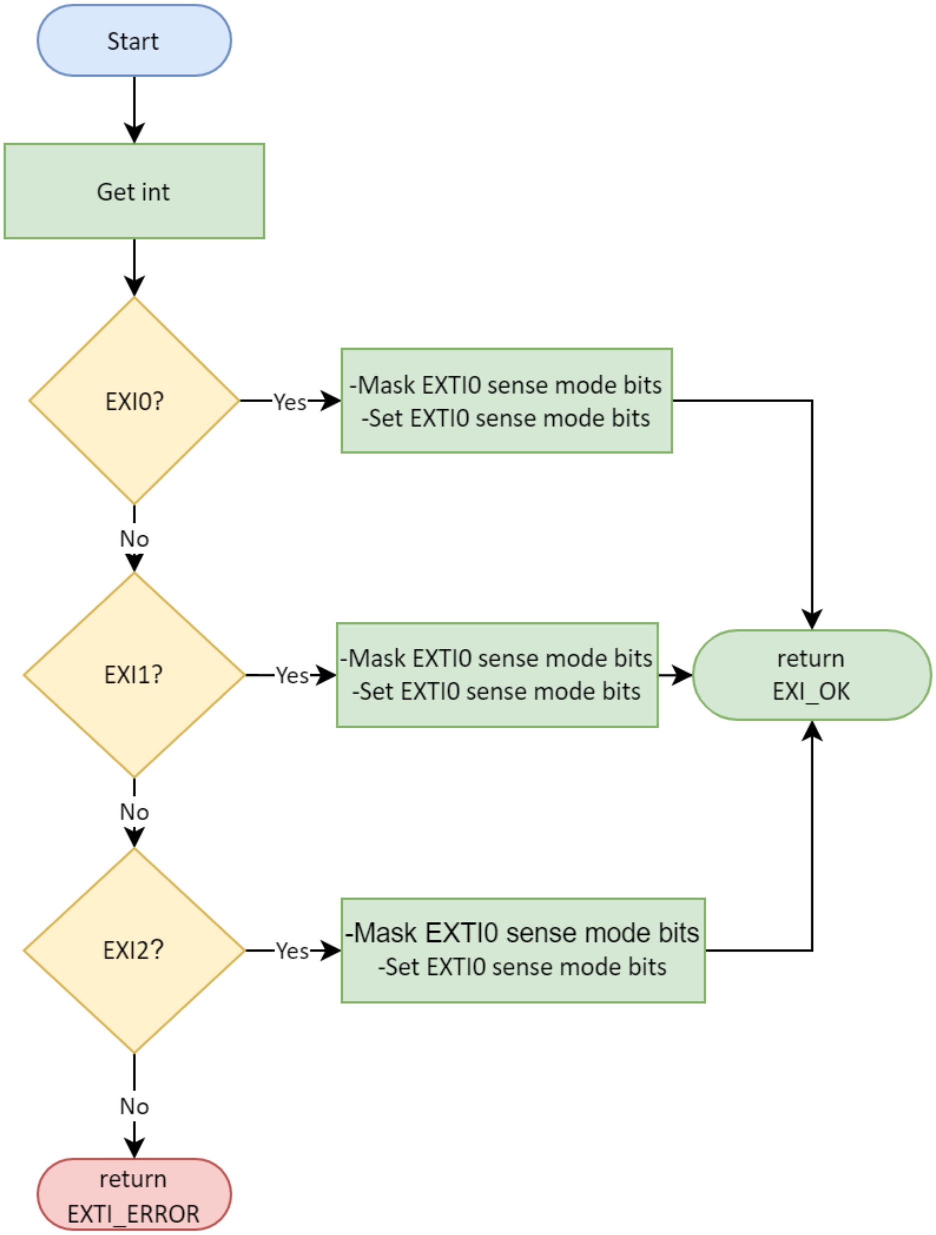

ParameterCheck
Start
EXTI_min <EXTI_num< EXTI_max ?
No
Yes
configured sense mode valid?
No
return EXI_ERROR
Yes
configured EXI state valid?
No
Yes
return EXI_OK

3.1.2.1. EXI_init

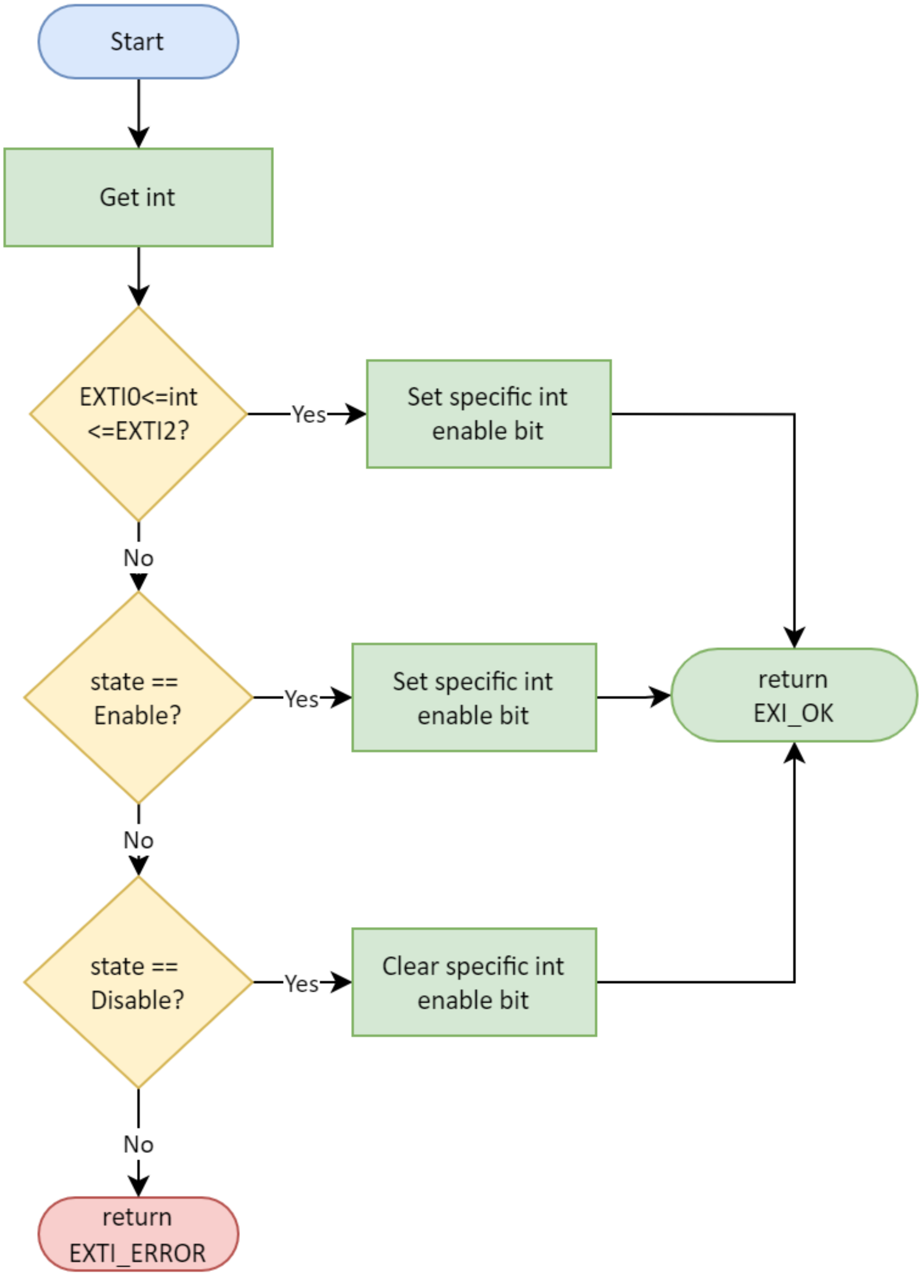


Start
Get EXTI Num
Parameter Check
Fail
return EXI_ERROR
Pass
switch(EXTI_Num)
EXI0?
Yes
EXI_setSense(EXTI0, configVal)
EXI_setState(EXTI0, configVal)
No
EXI1?
Yes
EXI_setSense(EXTI1, configVal)
EXI_setState(EXTI1, configVal)
return EXI_OK
No
EXI2?
Yes
EXI_setSense(EXTI2, configVal)
EXI_setState(EXTI2, configVal)
No
return EXI_ERROR

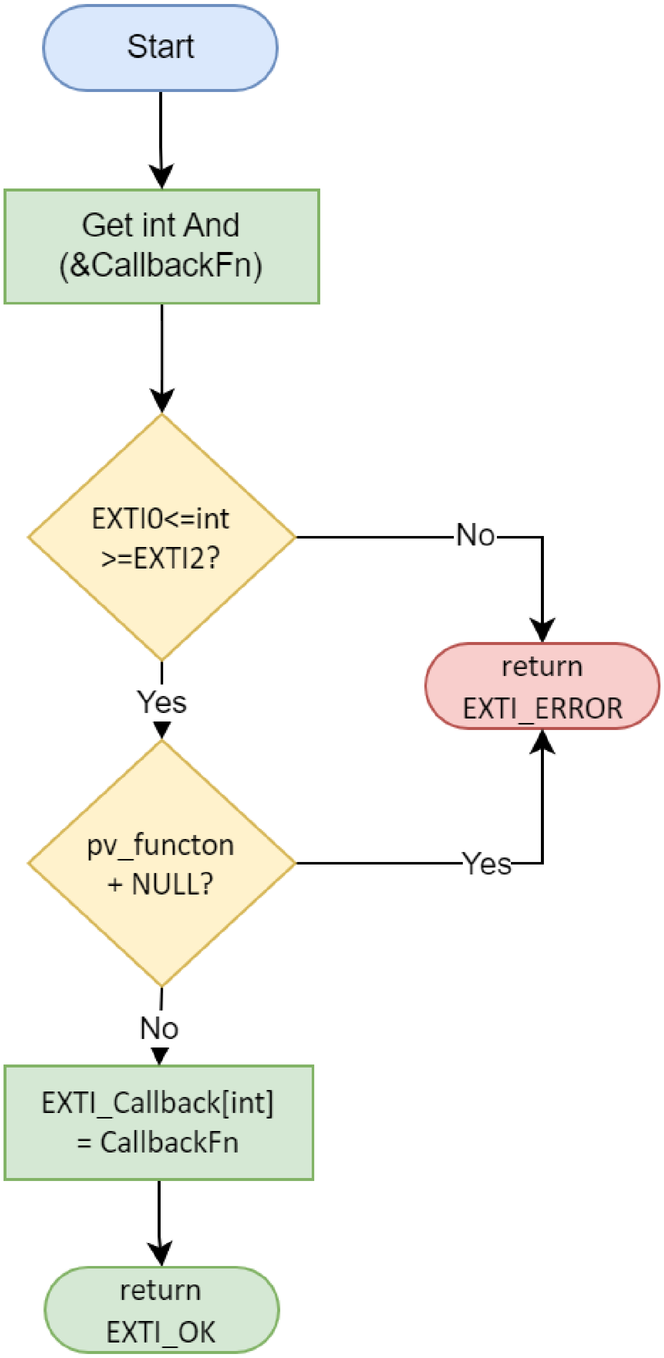
3.1.2.2. EXI_setSense


Start
Get int
EXI0?
Yes
-Mask EXTI0 sense mode bits
-Set EXTI0 sense mode bits
No
EXI1?
Yes
-Mask EXTI0 sense mode bits
-Set EXTI0 sense mode bits
return
EXI_OK
No
EXI2?
Yes
-Mask EXTI0 sense mode bits
-Set EXTI0 sense mode bits
No
return
EXTI_ERROR

3.1.2.3. EXI_setState

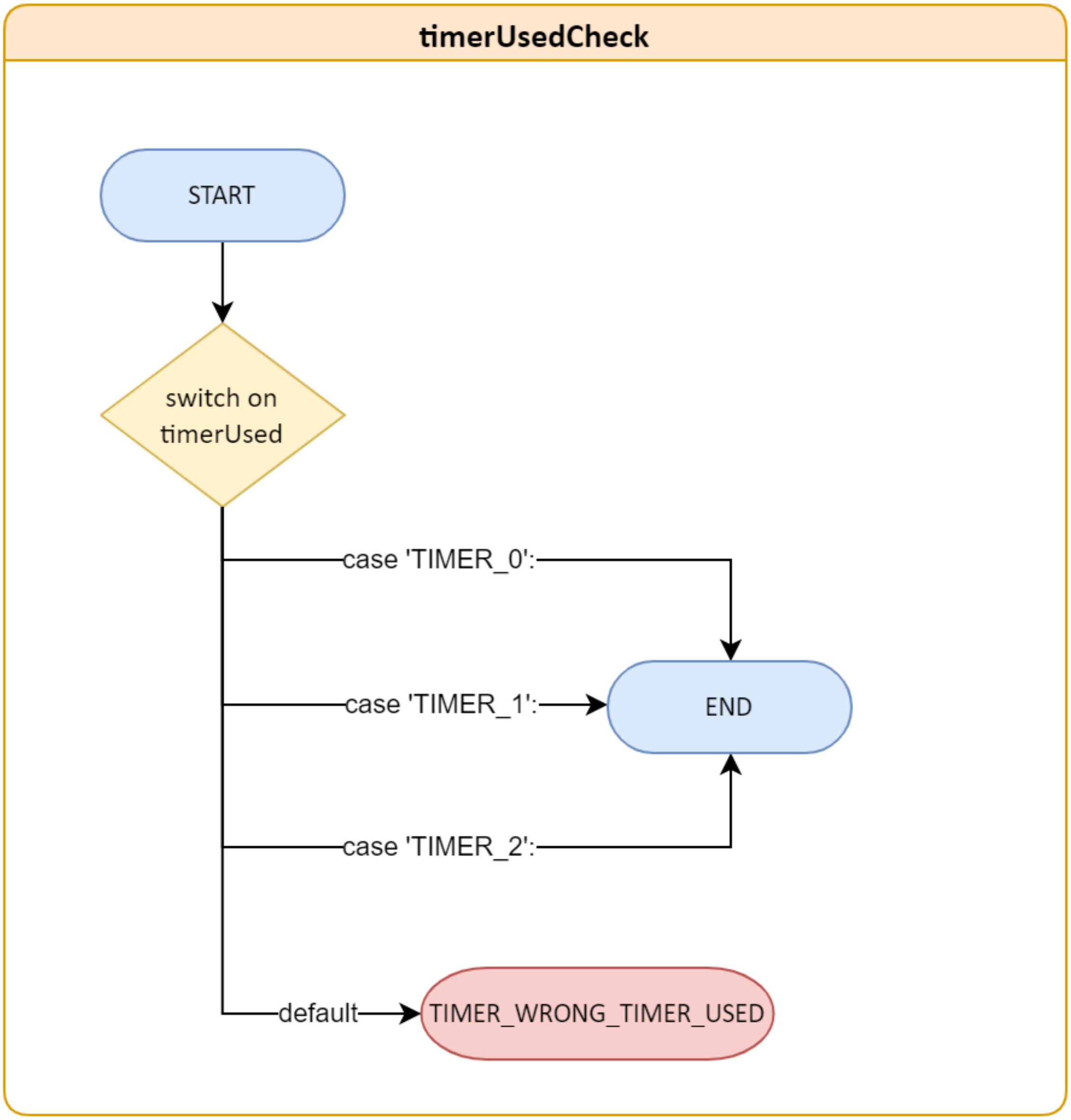


3.1.2.4. EXI_setCallback

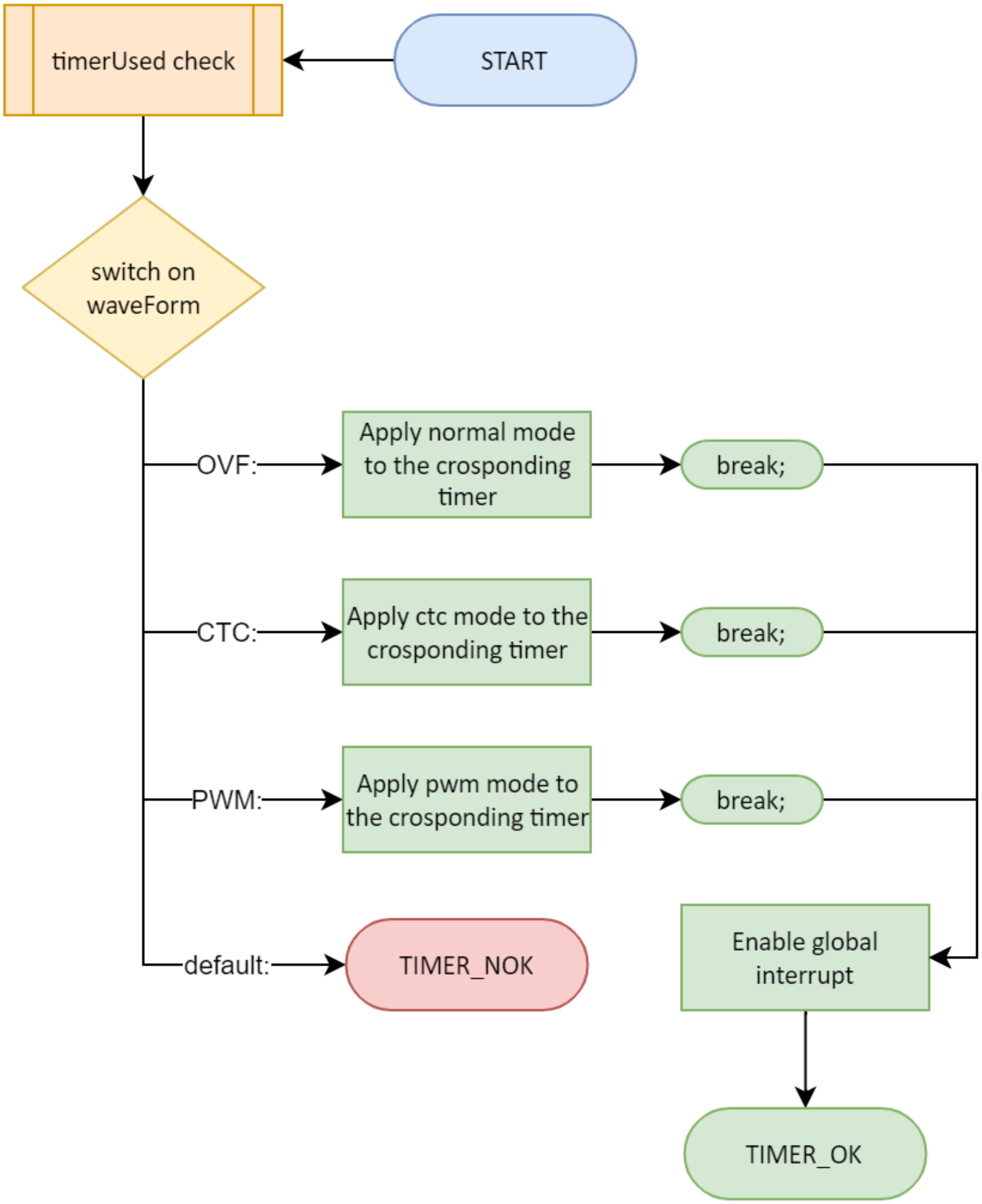


3.1.3. Timer Module

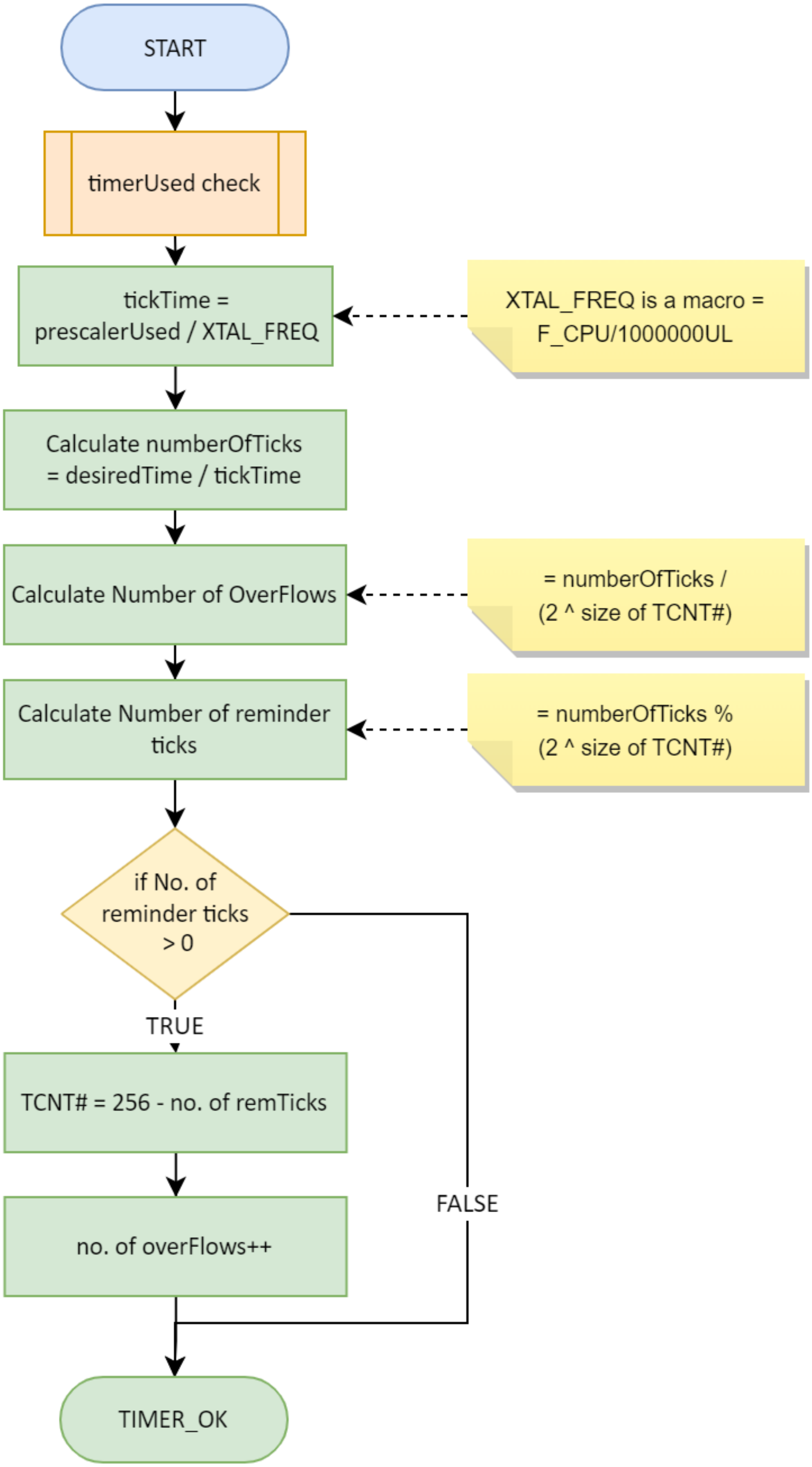
3.1.3.a. sub process



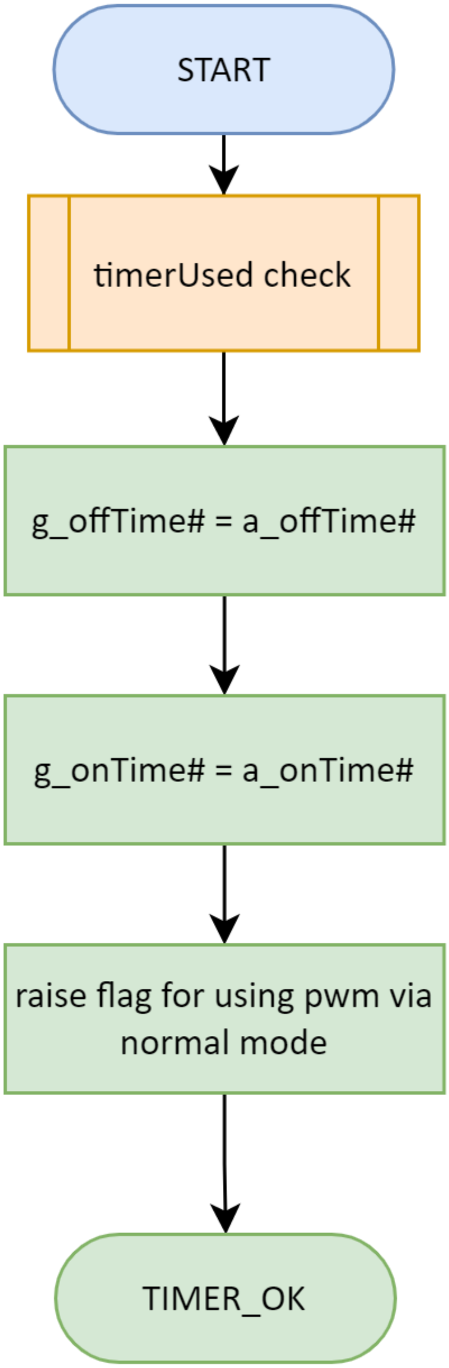
3.1.3.1. TIMER_init


START
timerUsed check
switch on waveForm
OVF:
Apply normal mode to the crosponding timer
break;
CTC:
Apply ctc mode to the crosponding timer
break;
PWM:
Apply pwm mode to the crosponding timer
break;
default:
TIMER_NOK
Enable global interrupt
TIMER_OK

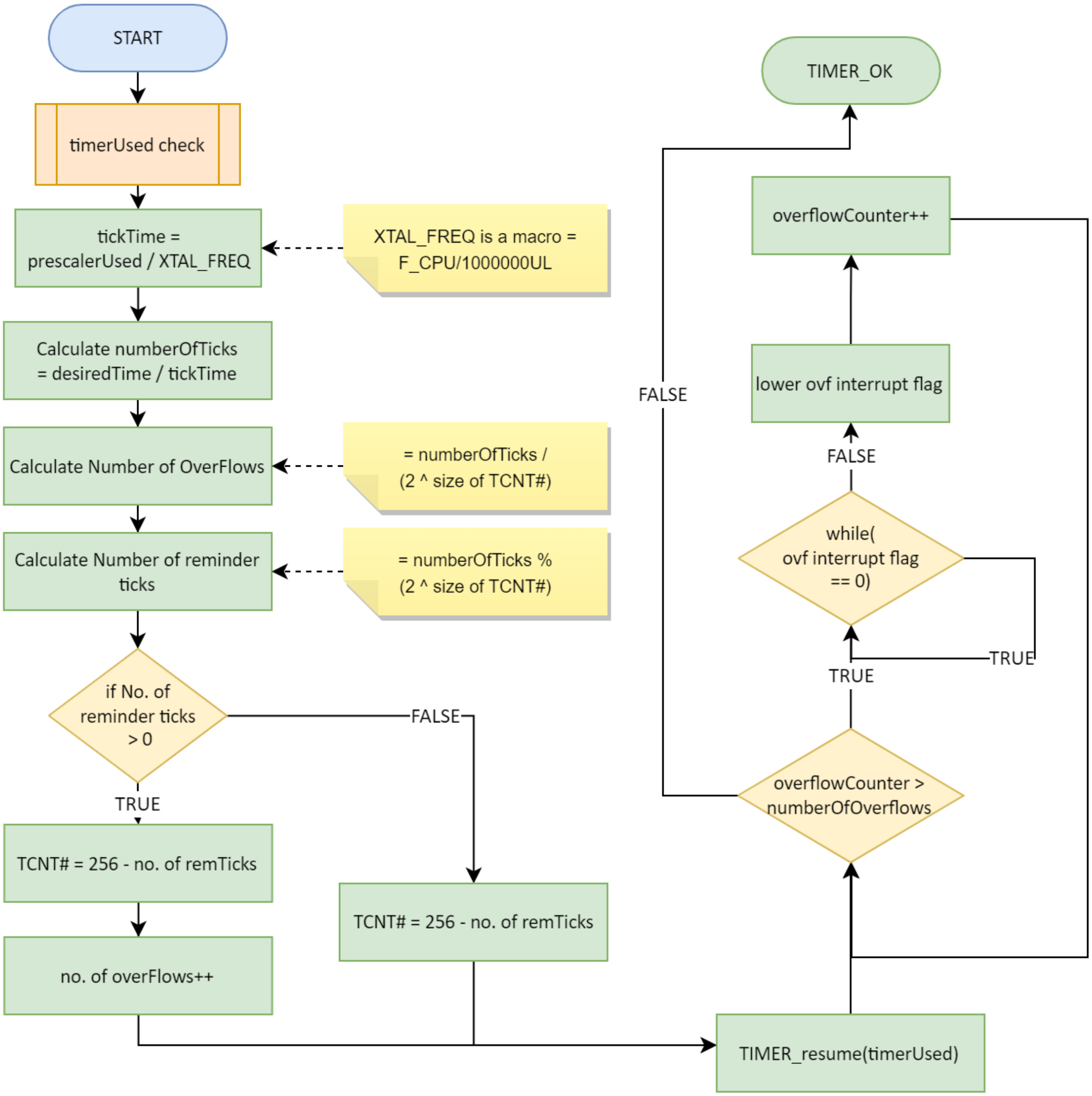
3.1.3.2. TIMER_setTime

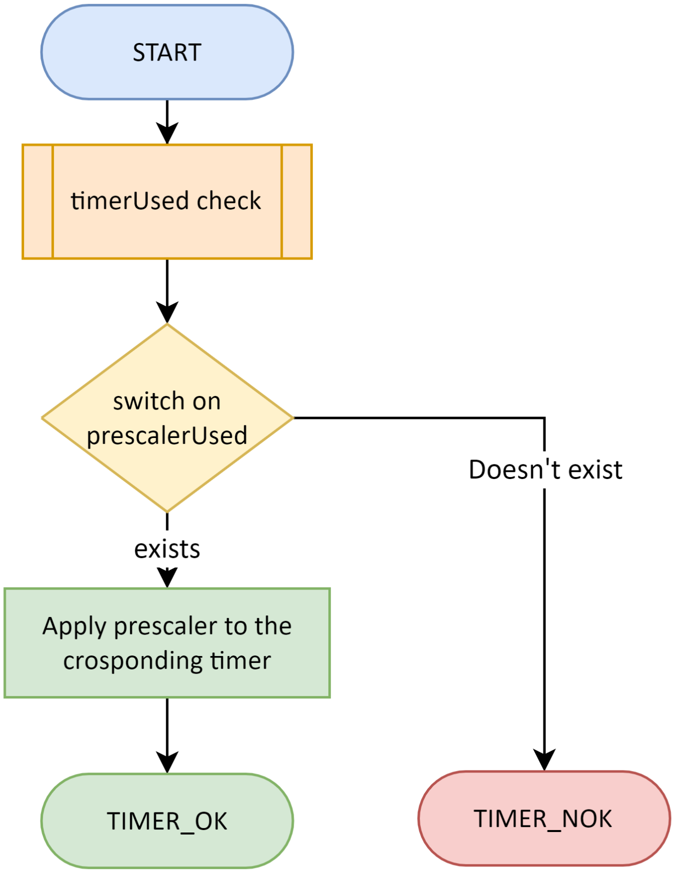

START
timerUsed check
tickTime =
prescalerUsed / XTAL_FREQ
XTAL_FREQ is a macro =
F_CPU/1000000UL
Calculate numberOfTicks
= desiredTime / tickTime
Calculate Number of OverFlows
= numberOfTicks /
(2 ^ size of TCNT#)
Calculate Number of reminder
ticks
= numberOfTicks %
(2 ^ size of TCNT#)
if No. of
reminder ticks
> 0
TRUE
TCNT# = 256 - no. of remTicks
no. of overFlows++
FALSE
TIMER_OK

3.1.3.3. TIMER_pwmGenerator

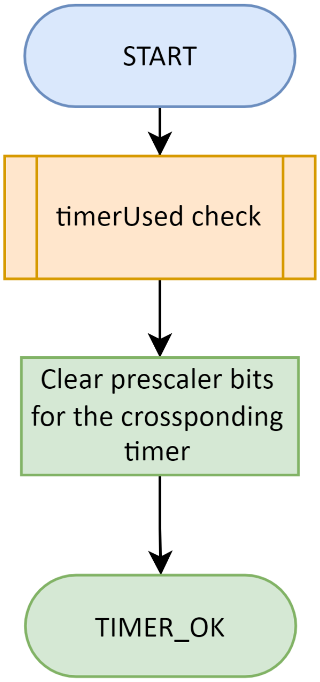


3.1.3.4. TIMER_resume

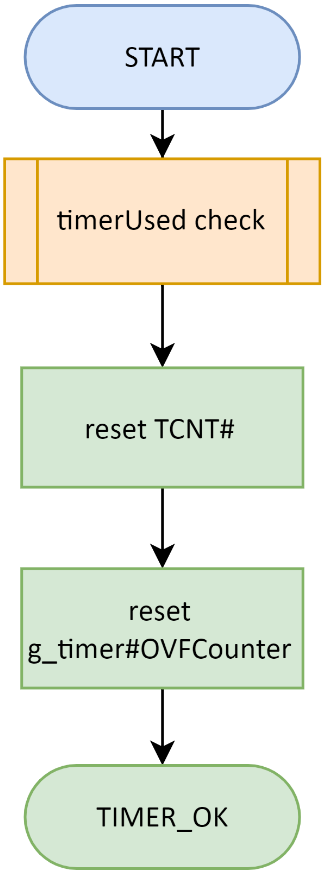

START
timerUsed check
tickTime = prescalerUsed / XTAL_FREQ
XTAL_FREQ is a macro = F_CPU/1000000UL
Calculate numberOfTicks = desiredTime / tickTime
Calculate Number of OverFlows
= numberOfTicks / (2 ^ size of TCNT#)
Calculate Number of reminder ticks
= numberOfTicks % (2 ^ size of TCNT#)
if No. of reminder ticks > 0
FALSE
TRUE
TCNT# = 256 - no. of remTicks
no. of overFlows++
TCNT# = 256 - no. of remTicks
TIMER_resume(timerUsed)
overflowCounter > numberOfOverflows
FALSE
TRUE
while(ovf interrupt flag == 0)
TRUE
FALSE
lower ovf interrupt flag
overflowCounter++
TIMER_OK



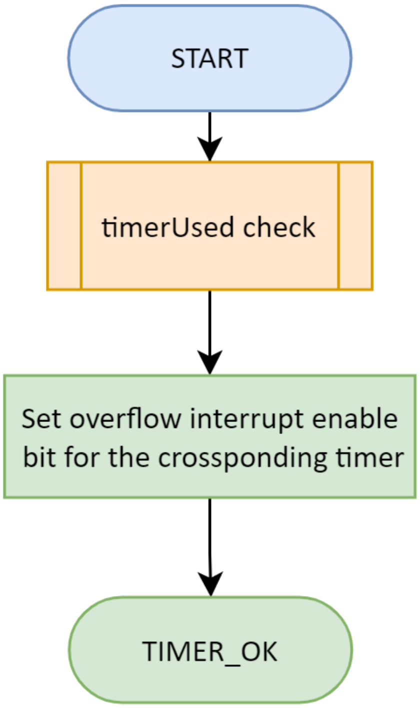
3.1.3.5. TIMER_pause


START
timerUsed check
Clear prescaler bits for the crossponding timer
TIMER_OK

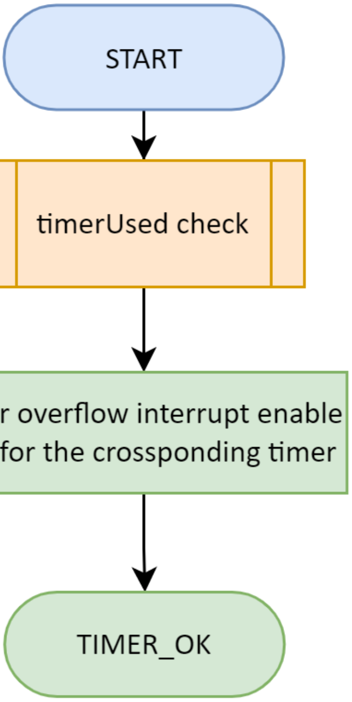
3.1.3.6. TIMER_reset



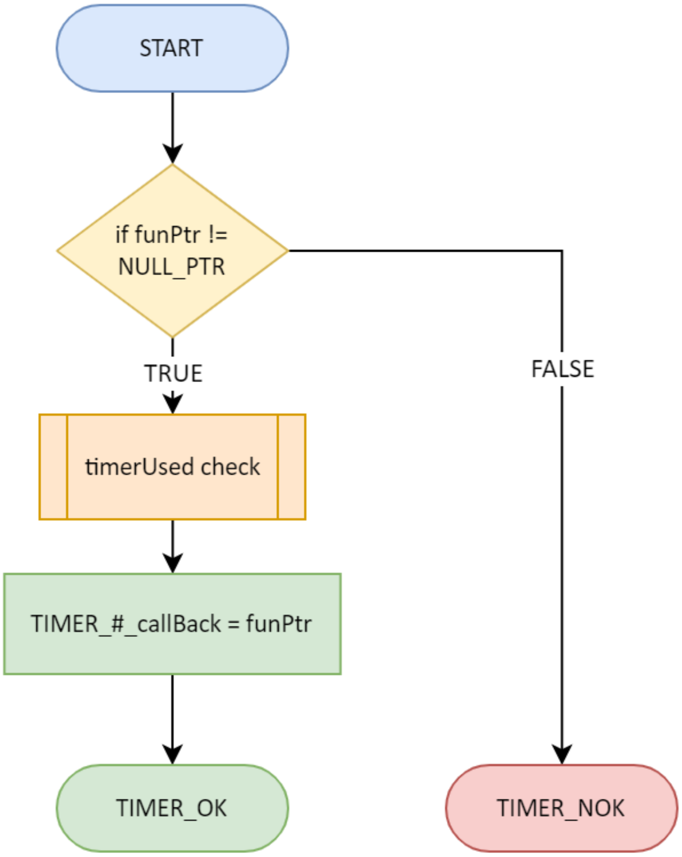
3.1.3.7. TIMER_enableInterrupt


START
timerUsed check
Set overflow interrupt enable bit for the crossponding timer
TIMER_OK

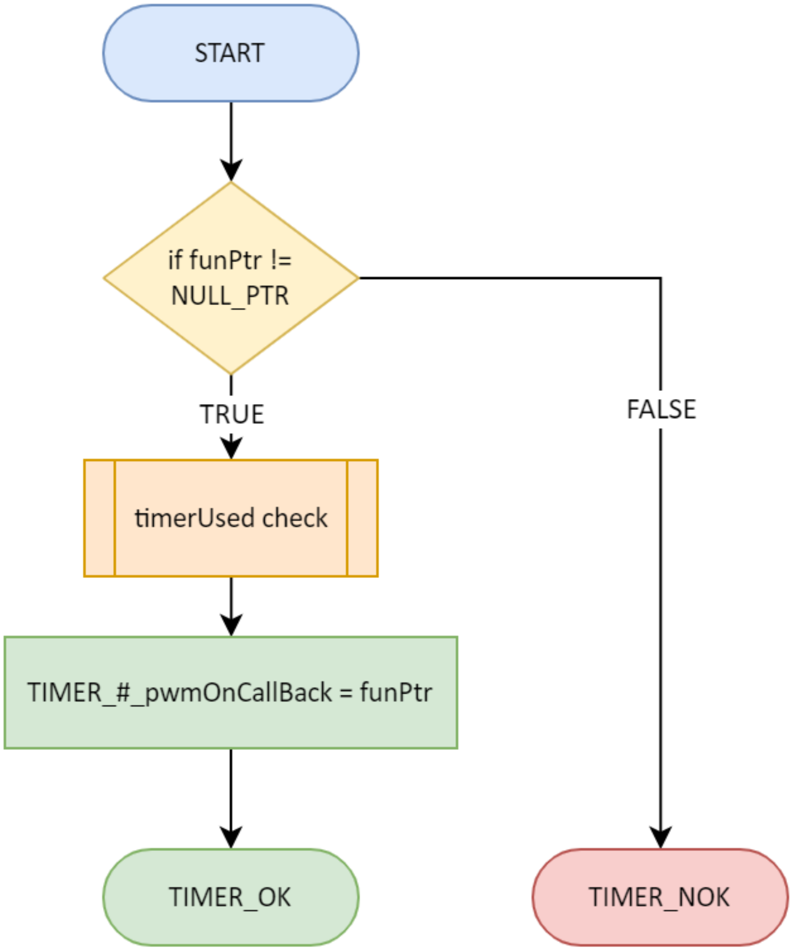
3.1.3.8. TIMER_enableInterrupt


START
timerUsed check
Clear overflow interrupt enable bit for the crossponding timer
TIMER_OK

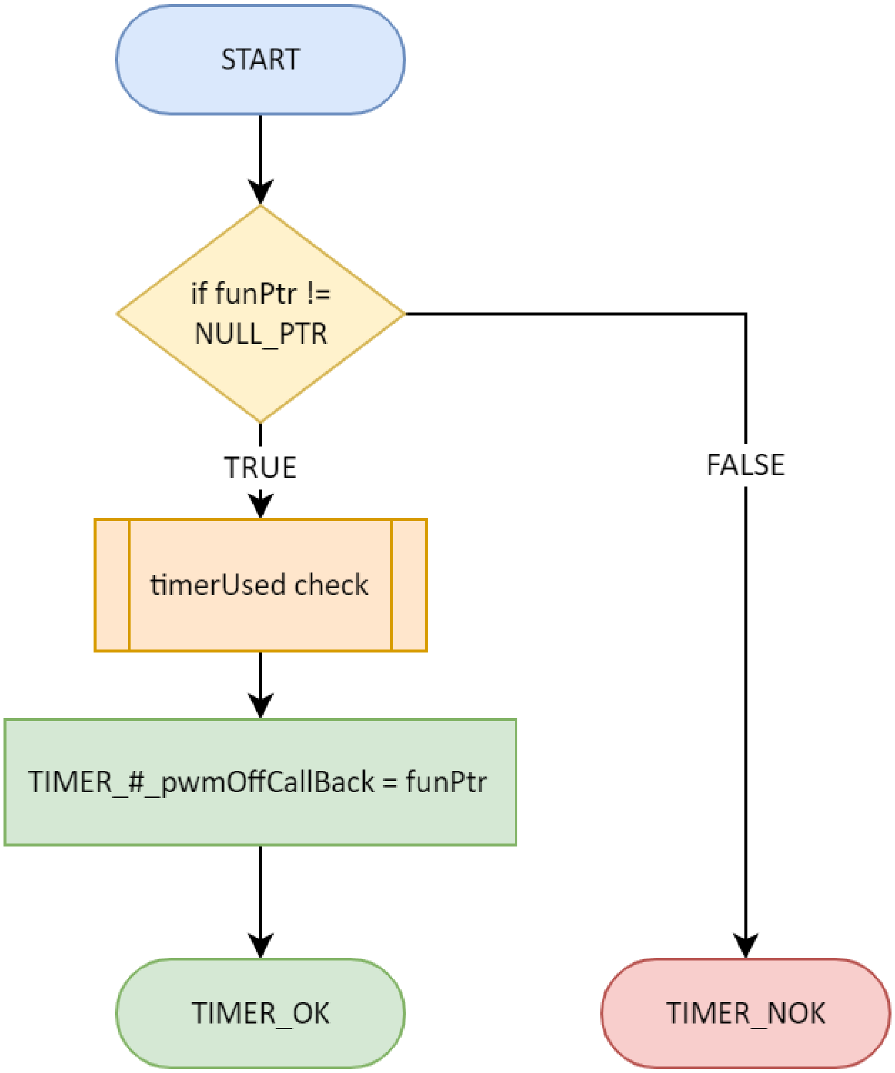
3.1.3.9. TIMER_setCallBack



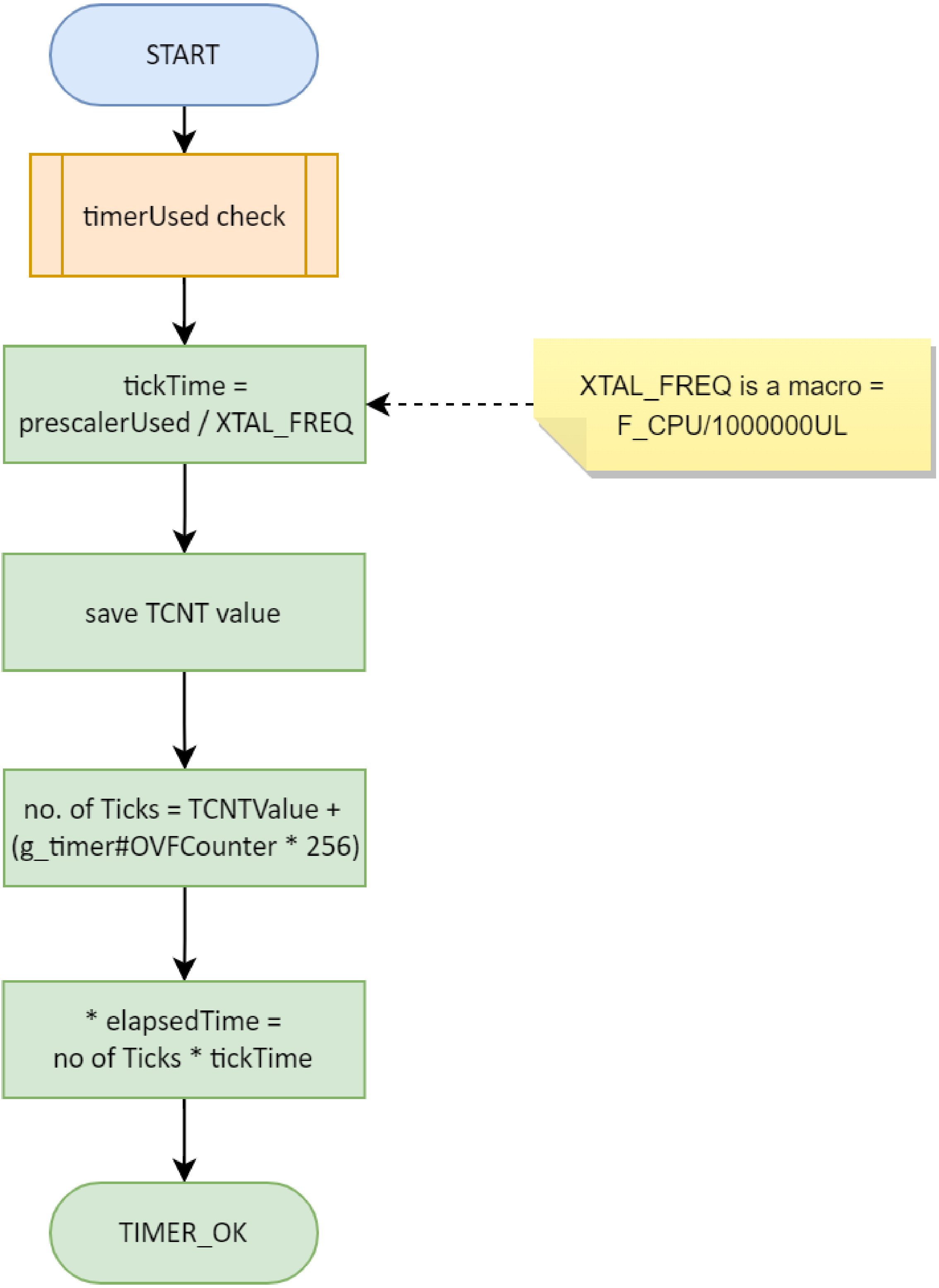
3.1.3.10. TIMER_setPwmOnCallBack



3.1.3.11. TIMER_setPwmOffCallBack



3.1.3.12. TIMER_getElapsedTime



3.1.3.13. TIMER_setDelayTime

4. Pre-compiling and linking configurations

4.1. EXI Driver

4.1.1. Linking Configuration

```
typedef enum
{
       EXTI0,
       EXTI1,
       EXTI2
}en_EXI_num_t;

typedef enum
{
       EXI_DISABLE ,
       EXI_ENABLE
}en_EXI_state_t;

typedef enum
{
       LOW_LEVEL,
       ON_CHANGE,
       FALLING_EDGE,
       RISING_EDGE
}en_EXI_senseMode_t;

typedef enum
{
       EXI_OK,
       EXI_ERROR
}en_EXI_error_t;

typedef struct
{
       en_EXI_num_t  EXI_NUM;
       en_EXI_senseMode_t SENSE_MODE;
       en_EXI_state_t      EXI_EN;
}st_EXI_config_t;
```

```
|
| Options:
|                   .EXI_NUM =   EXTI0
|                                 EXTI1
|                                 EXTI2
|
|                   .SENSE_MODE = LOW_LEVEL          [for EXTI0 & EXTI1 only]
|                                 ON_CHANGE          [for EXTI0 & EXTI1 only]
|                                 FALLING_EDGE
|                                 RISING_EDGE
|
|                   .EXI_EN      = EXI_ENABLE
|                                        EXI_DISABLE
|
const st_EXI_config_t arr_g_exiConfigs[EXI_PINS_NUM] =
{
       {
             .EXI_NUM= EXTI0,
             .SENSE_MODE = RISING_EDGE,
             .EXI_EN = EXI_DISABLE
       },

       {
             .EXI_NUM= EXTI1,
             .SENSE_MODE = FALLING_EDGE,
             .EXI_EN = EXI_DISABLE
       },

       {
             .EXI_NUM= EXTI2,
             .SENSE_MODE = RISING_EDGE,
             .EXI_EN = EXI_DISABLE
       }
};
```

4.2. Timer Driver

4.2.1. Linking configurations

```
/***************************************************************************
***
 *  GLOBAL  DATA

****************************************************************************
*/
 const st_TIMER_config_t st_TIMER_config [NUMBER_OF_TIMERS_USED] =
{
/*    TIMER_number,     waveformUsed,     prescalerUsed     */
      {TIMER_0,         TIMER_OV,         TIMER_PRESCLNG_64},
      {TIMER_1,         TIMER_OV,         TIMER_PRESCLNG_64},
      {TIMER_2,         TIMER_OV,         TIMER_PRESCLNG_64}
};
```

4.2.2 Pre-compiled Configurations

```
/***************************************************************************
***
 * GLOBAL  CONSTANT  MACROS

****************************************************************************
*/

/********************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_*******************
**/
/*
 *    Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                         8000000UL

/*************************_NUMBER_OF_TIMERS_USED_*************************/
/*
 *    number of timers used
 */
#define NUMBER_OF_TIMERS_USED                         3
```

4.3. BTN Driver

4.3.1. pre-compiling configuration

```
/* Macros */
#define MAX_PIN_NUMBER  7
#define MAX_PORT_NUMBER 3

#define BTN_DELAY_BTN_DEBOUNCE      50
```