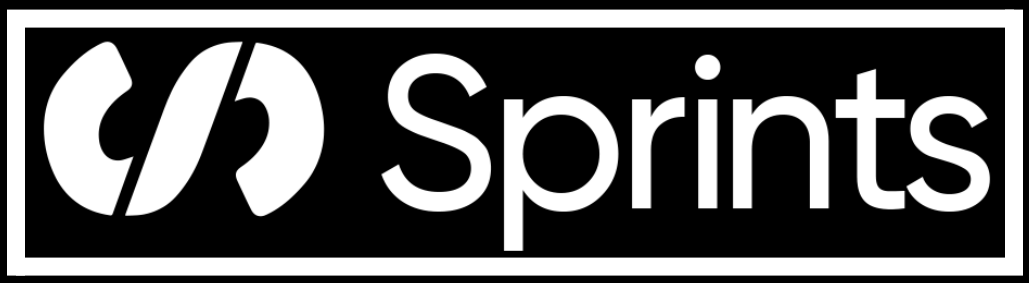
Sprints



Four Wheel Driving Car

Design

Team Member

Arafa Arafa

Bassel Yasser

Mahmoud Adel

Youssef Abbas

2023

Contents

Project Introduction 2
High Level Design 3
Layered Architecture 3
Module Description 4
Drivers' documentation 5
APP 5
SERVICE 7
HAL 10
MCAL 13
UML 22
Low Level Design 23
Flowchart 23
APP 23
HAL 25
MCAL 27
Pre-compiling configuration 39
MCAL 39
Linking Configuration 41
MCAL 41

Project Introduction

Car Components

1. Use Sprints Kit with TivaC launch pad plugged in
2. You will develop your application on the ARM microcontroller
3. Four motors (M1, M2, M3, M4)
4. One button to start (PB1)
5. One button for stop (PB2)
6. Four LEDs (LED1, LED2, LED3, LED4)

System Requirements

1. The car starts initially from 0 speed
2. When PB1 is pressed, the car will move forward after 1 second
3. The car will move forward to create the longest side of the rectangle for 3 seconds with 50% of its maximum speed
4. After finishing the first longest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second
5. The car will move to create the short side of the rectangle at 30% of its speed for 2 seconds
6. After finishing the shortest side, the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second
7. Steps 3 to 6 will be repeated infinitely until you press the stop button (PB2)
8. PB2 acts as a sudden break, and it has the highest priority
9. LEDs Operations
    1. LED1: On means moving forward on the long side
    2. LED2: On means moving forward on the short side
    3. LED3: On means stop
    4. LED4: On means Rotating

High Level Design

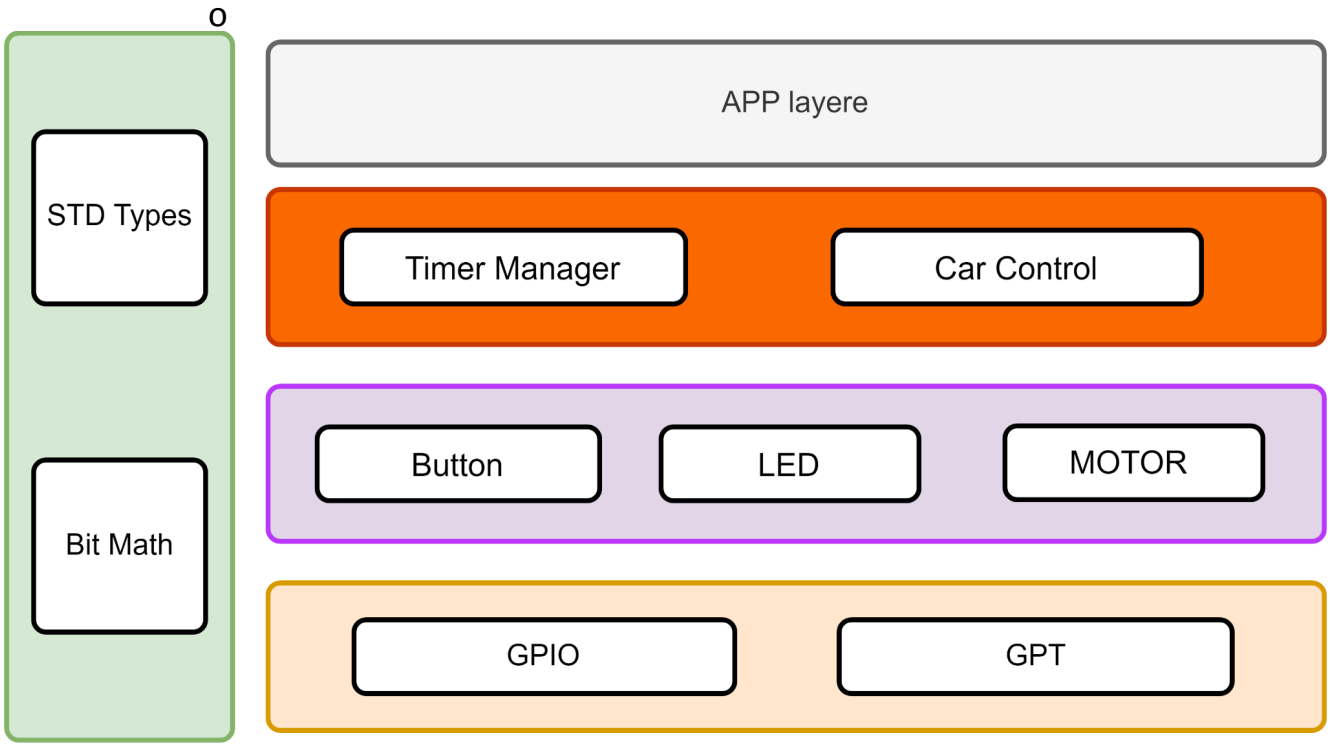
Layered Architecture

APP Layer: written in high level languages like java, C++, C# with rich GUI support. Application layer calls the middleware api in response to action by the user or an event.

HAL Layer: are a way to provide an interface between hardware and software so applications can be device independent.

MCAL Layer: is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU. Details of the MCAL software module are shown below.

Common Layer: is the layer which consists of BIT_MATH and STD types



Module Description

- **APP Layer**
  - **App:** written in high level languages like java, C++, C# with rich GUI support. Application layer calls the middleware api in response to action by the user or an event.
- **HAL Layer**
  - **button:** Initialize selected button pin as input
  - **Led:** this led module configure selected pin as output and generate volt
- **MCAL Layer**
  - **GPIO:** The GPIO module is composed of six physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F). The GPIO module supports up to 43 programmable input/output pins, depending on the peripherals being used.
  - **GPT:** Programmable timers can be used to count or time external events that drive the Timer input pins. The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters (referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters, or concatenated to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Each 32/64-bit Wide GPTM block provides 32-bit timers for Timer A and Timer B that can be concatenated to operate as a 64-bit timer. Timers can also be used to trigger µDMA transfers
- **COMMON Layer**
  - **std_types:** having basic standard types like (Uint32_t, Uint8_t, …..)
  - **bit_math :** Consist of bit manipulation like (SetBit, ClrBit, GetBit, ..)

Drivers' documentation

APP

APP_vidInit

Service name	APP_vidInit
Description	This Function Make Modules Initialization
Syntax	void APP_vidInit (void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	void
Parameters (out)	None
Return	void
Available via	app.h

APP_vidStart

Service name	APP_vidStart
Description	This Function Start the Application.
Syntax	void APP_vidStart (void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	void
Parameters (out)	None
Return	void
Available via	app.h

SERVICE

Timer Manager module

TIMM_u8Init

Service name	TIMM_u8Init
Description	GPT Timer Initialization
Syntax	`uint8_ TIMM_u8Init (st_gpt_timer_cfg_t* st_gpt_timer_cfg)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`st_gpt_timer_cfg: Address of struct Instance`
Parameters (out)	None
Return	`SUCCESS: in case of successful operation`
	`FAILED: in case of failer operation`
Available via	timerM_Interface.h

TIMM_u8Start

Service name	TIMM_u8Start
Description	Start Timer count
Syntax	`uint8_ TIMM_u8Start (void)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	**Void**
Parameters (out)	None
Return	`SUCCESS: in case of successful operation`
	`FAILED: in case of failer operation`

Available via	timerM_Interface.h

TIMM_vidStop

Service name	**TIMM_vidStop**
Description	Stop GPT Timer Counter
Syntax	`void` **TIMM_vidStop** `(void)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**void**
Parameters (out)	**NONE**
Return	`void`
Available via	timerM_Interface.h

TIMM_vidSynchDelay_ms

Service name	**TIMM_vidSynchDelay_ms**
Description	Set Delay in milli second
Syntax	`uint8_ TIMM_vidSynchDelay_ms (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_ms)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_ms: time in milli second
Parameters (out)	**NONE**
Return	`void`

Available via	timerM_Interface.h

TIMM_vidSynchDelay_us

Service name	**TIMM_vidSynchDelay_us**
Description	Set Delay in micro second
Syntax	`uint8_ TIMM_vidSynchDelay_us (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_us)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_ms: time in micro second
Parameters (out)	**NONE**
Return	void
Available via	timerM_Interface.h

TIMM_vidSynchDelay_s

Service name	**TIMM_vidSynchDelay_s**
Description	Set Delay in second
Syntax	`uint8_ TIMM_vidSynchDelay_s (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_s)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_s: time in second
Parameters (out)	**NONE**

Return	void
Available via	timerM_Interface.h

HAL

HLED module

HLed_Init

Service name	HLed_Init
Description	This Function Init LED dio pin as output
Syntax	`enu_ledError_t HLed_Init (enu_pin en_pinNum)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinNum: dio pin selection**
Parameters (out)	None
Return	en_ledError_t
Available via	hled.h

HLed_on

Service name	HLed_on
Description	This Function give LED pin logic 1
Syntax	`enu_ledError_t HLed_on (enu_pin en_pinx);`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinNum: dio pin selection**

Parameters (out)	None
Return	`en_ledError_t`
Available via	hled.h

HLed_off

Service name	HLed_off
Description	This Function give LED pin logic 0
Syntax	`enu_ledError_t HLed_off (enu_pin en_pinx)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinNum: dio pin selection**
Parameters (out)	None
Return	`en_ledError_t`
Available via	hled.h

HLed_toggle

Service name	HLed_toggle
Description	This Function Change previous state of LED pin
Syntax	`enu_ledError_t HLed_toggle (enu_pin en_pinx)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinNum: dio pin selection**
Parameters (out)	None
Return	`en_ledError_t`
Available via	hled.h

Button module

HButton_Init

Service name	HButton_Init
Description	This Function Initialize button DIO pin as input and pull up
Syntax	`enu_buttonError_t HButton_Init(button_str_btn_config_t * en_pinx)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinx: btn_configuration**
Parameters (out)	None
Return	*BTN_OK: in case of successful operation*
	BTN_NULL_PTR: null pointer based
Available via	button.h

HButton_getPinVal

Service name	HButton_getPinVal
Description	This Function Get button state
Syntax	`enu_buttonError_t HButton_getPinVal(button_str_btn_config_t *en_pinx, btn_enu_btn_state_t* enu_refVal )`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinx: btn_configuration**
Parameters (out)	**pu8_refVal:** address of variable which button state to be stored
Return	*BTN_OK: in case of successful operation*
	BTN_NULL_PTR: null pointer based
Available via	button.h

HButton_initializa_with_int

Service name	HButton_initializa_with_int
Description	This Function Get button state
Syntax	`enu_buttonError_t HButton_initializa_with_int(const button_str_btn_config_t* ptr_str_btn_config , ptr_func_t ptr_callback)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinx: btn_configuration**
	ptr_callback: pointer to callback function
Return	*BTN_OK: in case of successful operation*
	BTN_NULL_PTR: null pointer based
Available via	button.h

HButton_enable_INT

Service name	HButton_enable_INT
Description	This Function Get button state
Syntax	`enu_buttonError_t HButton_enable_INT( button_str_btn_config_t* ptr_str_btn_config)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinx: btn_configuration**
Parameters (out)	None
Return	*BTN_OK: in case of successful operation*
	BTN_NULL_PTR: null pointer based
Available via	button.h

HButton_disable_INT

Service name	HButton_disable_INT
Description	This Function Get button state
Syntax	`enu_buttonError_t HButton_disable_INT( button_str_btn_config_t* ptr_str_btn_config)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**en_pinx: btn_configuration**
Parameters (out)	None
Return	*BTN_OK: in case of successful operation*
	BTN_NULL_PTR: null pointer based
Available via	button.h

MCAL

GPIO module

dio_init_pin

Service name	dio_init_pin
Description	This Function Initialize GPIO configuration
Syntax	`dio_enu_return_state_t dio_init_pin(dio_str_pin_Config_t *ptr_str_pinconfig)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`ptr_str_pinconfig`: Address of struct Instance
Parameters (out)	None
Return	`DIO_OK`: *in case of successful operation*
	`DIO_NULL_PTR`: Null pointer provided as input.
Available via	dio_Interface.h

dio_set_pin

Service name	dio_set_pin
Description	This Function Initialize Pin Value High or Low
Syntax	`dio_enu_return_state_t dio_set_pin(dio_str_pin_Config_t *ptr_str_pinconfig, dio_enu_pin_state_t copy_enu_pin_state)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**ptr_str_pinconfig: Pointer to the pin configuration structure.**
	copy_enu_pin_state: The state to set for the pin.
Parameters (out)	None
Return	`DIO_OK`: *in case of successful operation*
	`DIO_NULL_PTR`: Null pointer provided as input.
Available via	dio _Interface.h

dio_read_pin

Service name	dio_read_pin
Description	This Function Get value from selected pin
Syntax	dio_enu_return_state_t dio_read_pin(dio_str_pin_Config_t *ptr_str_pinconfig, dio_enu_pin_state_t *ptr_enu_pin_state)
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**ptr_str_pinconfig: Pointer to the pin configuration structure.**
Parameters (out)	**ptr_enu_pin_state: Pointer to the variable that will store the pin state.**
Return	DIO_OK: *in case of successful operation*
	DIO_NULL_PTR: Null pointer provided as input.
Available via	dio _Interface.h

EXIT

exit_init_pin

Service name	**exit_init_pin**
Description	This Function init the pin as EXIT source
Syntax	`exit_enu_return_state_t exit_init_pin(exit_str_pin_Config_t *ptr_str_pin_config)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**ptr_str_pin_config: Pointer to the exit pin configuration structure.**
Parameters (out)	**NONE**
Return	`EXIT_OK: Everything done successfully.`
	`EXIT_NULL_PTR: Null pointer provided as input.`
Available via	EXIT _Interface.h
Service name	**exit_init_pin**
Description	This Function Get value from selected pin

exit_enable_int

Service name	exit_enable_int
Description	This Function enable the EXIT
Syntax	`exit_enu_return_state_t exit_enable_int(exit_str_pin_Config_t *ptr_str_pin_config)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**ptr_str_pin_config: Pointer to the exit pin configuration structure.**
Parameters (out)	**NONE**
Return	`EXIT_OK: Everything done successfully.`
	`EXIT_NULL_PTR: Null pointer provided as input.`
Available via	EXIT _Interface.h

exit_disable_int

Service name	exit_disable_int
Description	This Function enable the EXIT
Syntax	`exit_enu_return_state_t exit_disable_int(exit_str_pin_Config_t *ptr_str_pin_config)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**ptr_str_pin_config: Pointer to the exit pin configuration structure.**
Parameters (out)	**NONE**
Return	`EXIT_OK: Everything done successfully.`
	`EXIT_NULL_PTR: Null pointer provided as input.`
Available via	EXIT _Interface.h

exit_set_callback

Service name	exit_set_callback
Description	This Function set callback function
Syntax	`exit_enu_return_state_t exit_set_callback(exit_str_pin_Config_t *ptr_str_pin_config, ptr_func_t ptr_call_back)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	**ptr_str_pin_config: Pointer to the exit pin configuration structure.**
Parameters (out)	**ptr_call_back: Pointer to the callback function.**
Return	`EXIT_OK: Everything done successfully.`
	`EXIT_NULL_PTR: Null pointer provided as input.`
Available via	EXIT _Interface.h

GPT module

GPT_u8Init

Service name	GPT_u8Init
Description	GPT Timer Initialization
Syntax	`uint8_` **GPT_u8Init** `(st_gpt_timer_cfg_t*` **st_gpt_timer_cfg**`)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**st_gpt_timer_cfg**: Address of struct Instance
Parameters (out)	None
Return	`SUCCESS: in case of successful operation`
	`FAILED: in case of failure operation`
Available via	gpt_Interface.h

GPT_u8Start

Service name	GPT_u8Start
Description	Start Timer count
Syntax	`uint8_ GPT_u8Start (void)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	**Void**
Parameters (out)	None
Return	`SUCCESS: in case of successful operation`
	`FAILED: in case of failure operation`
Available via	gpt_Interface.h

GPT_vidStop

Service name	GPT_vidStop
Description	Stop GPT Timer Counter
Syntax	`void GPT_vidStop (void)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**void**
Parameters (out)	**NONE**
Return	`void`
Available via	gpt_Interface.h

GPT_vidIRQEnable

Service name	GPT_vidIRQEnable
Description	GPT enable Interrupt
Syntax	`void GPT_vidIRQEnable (void)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**void**
Parameters (out)	**NONE**
Return	`void`
Available via	gpt_Interface.h

GPT_vidIRQDisable

Service name	GPT_vidIRQDisable
Description	GPT Disable Interrupt
Syntax	`uint8_ GPT_vidIRQDisable (void)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	`Void`
Parameters (out)	**NONE**
Return	`SUCCESS: in case of successful operation`
	`FAILED: in case of failure operation`
Available via	gpt_Interface.h

GPT_u8GetCurrentVal

Service name	GPT_u8GetCurrentVal
Description	Get GPT current value
Syntax	`Void GPT_u8GetCurrentVal (uint64_* p_u64_int_cur_val)`
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in)	**void**
Parameters (out)	**p_u64_int_cur_val:** Reference to variable where the value status store on it
Return	SUCCESS: *in case of successful operation*
	FAILED: *in case of failure operation*
Available via	gpt_Interface.h

GPT_u8Delay_ms

Service name	GPT_u8Delay_ms
Description	Set Delay in millisecond
Syntax	`uint8_ TIMM_GPT_u8Delay_ms (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_ms)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_ms: time in millisecond
Parameters (out)	**NONE**
Return	void
Available via	timerM_Interface.h

GPT_u8Delay_us

Service name	GPT_u8Delay_us
Description	Set Delay in micro second
Syntax	`uint8_ TIMM_GPT_u8Delay_us (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_us)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_ms: time in micro second
Parameters (out)	**NONE**
Return	void
Available via	timerM_Interface.h

GPT_u8Delay_s

Service name	GPT_u8Delay_s
Description	Set Delay in second
Syntax	`uint8_ GPT_u8Delay_s (en_gpt_ch_id_t a_en_gpt_ch_id, uint32_ a_u32_time_s)`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	**a_en_gpt_ch_id:** {GPT_CHANNEL_0 ~ GPT_CHANNEL_5} - {GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5}
	a_u32_time_s: time in second
Parameters (out)	**NONE**

Return	`void`
Available via	timerM_Interface.h

UML

State Machine

Low Level Design

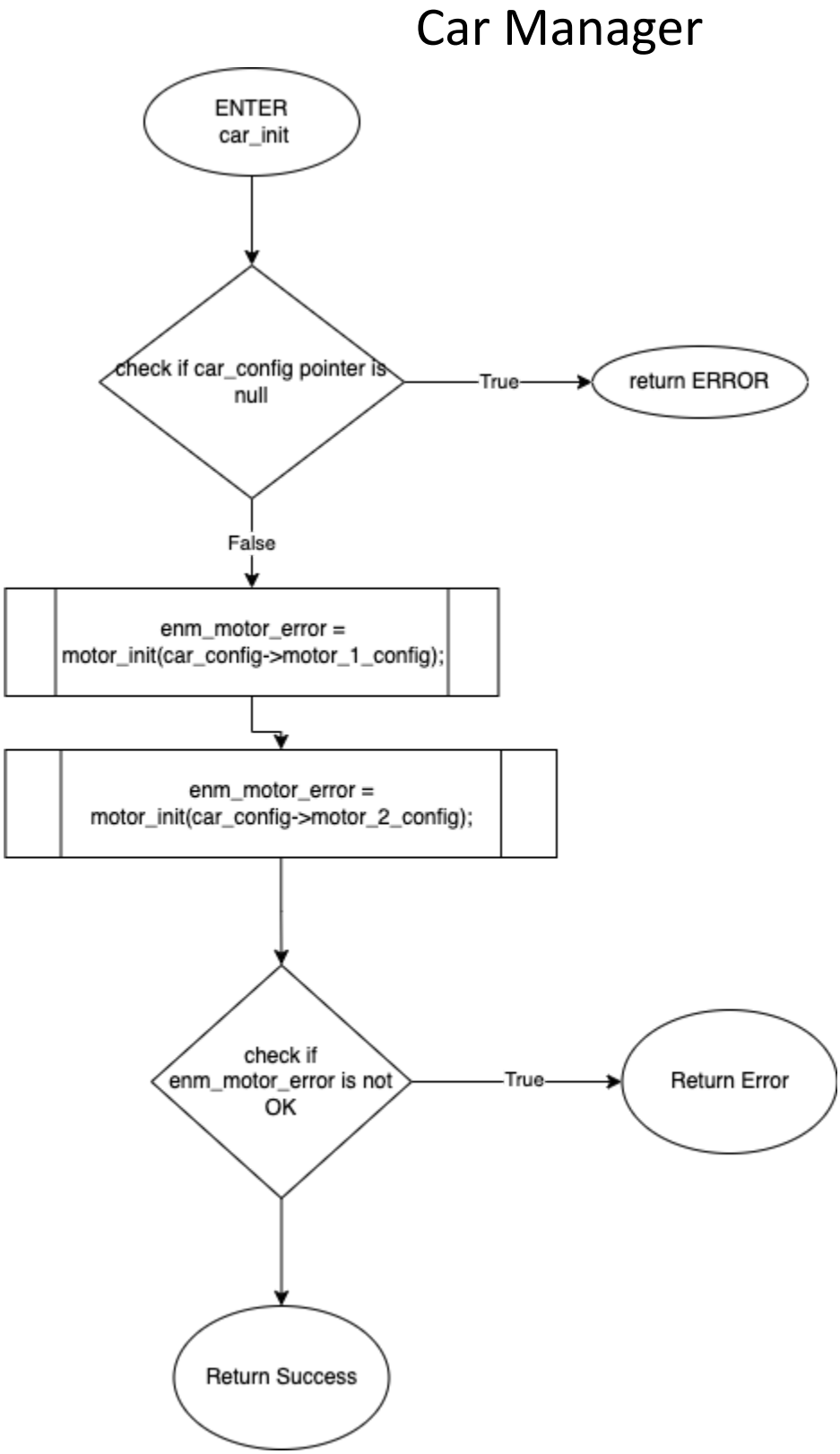
Flowchart

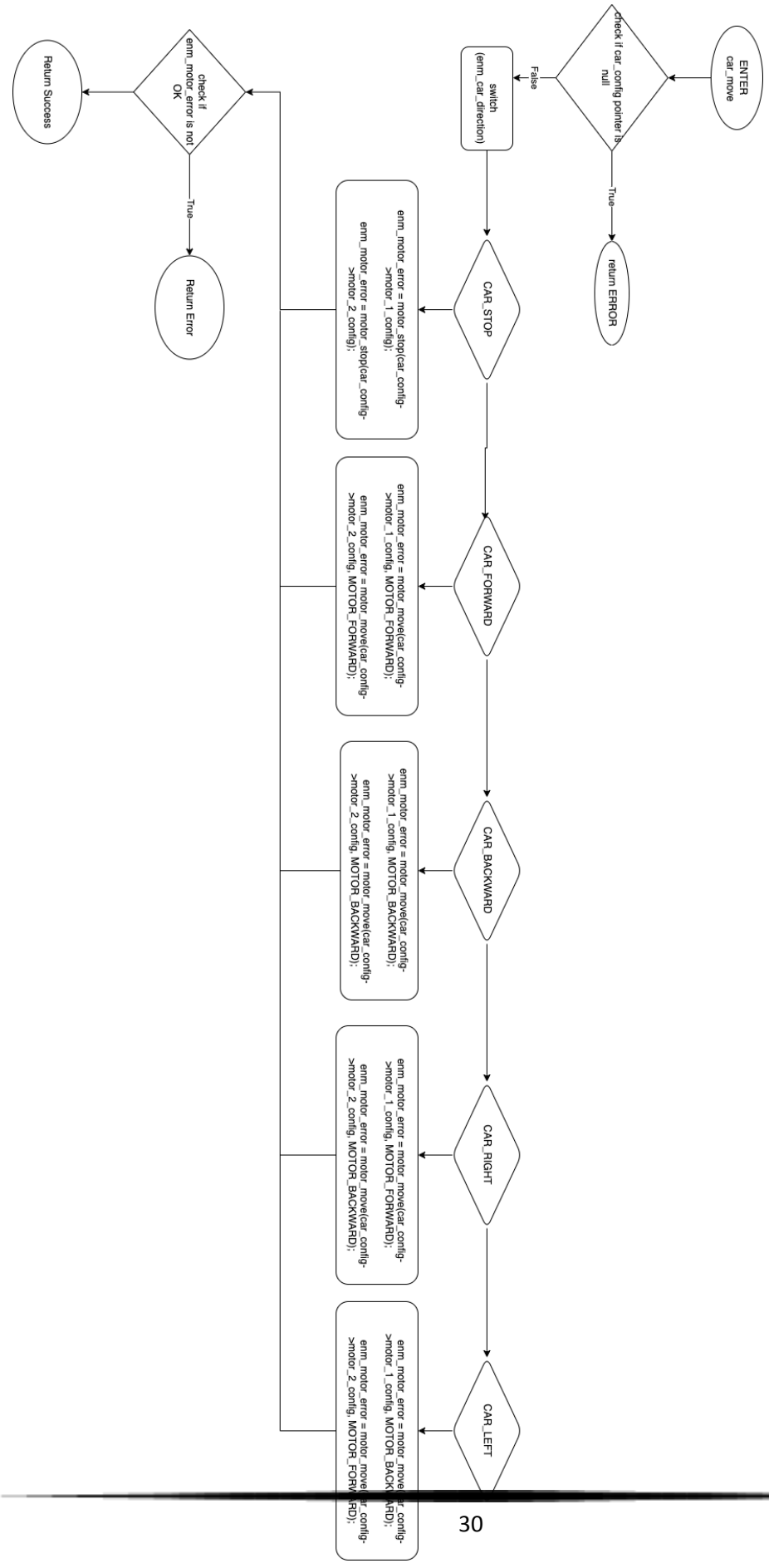
APP

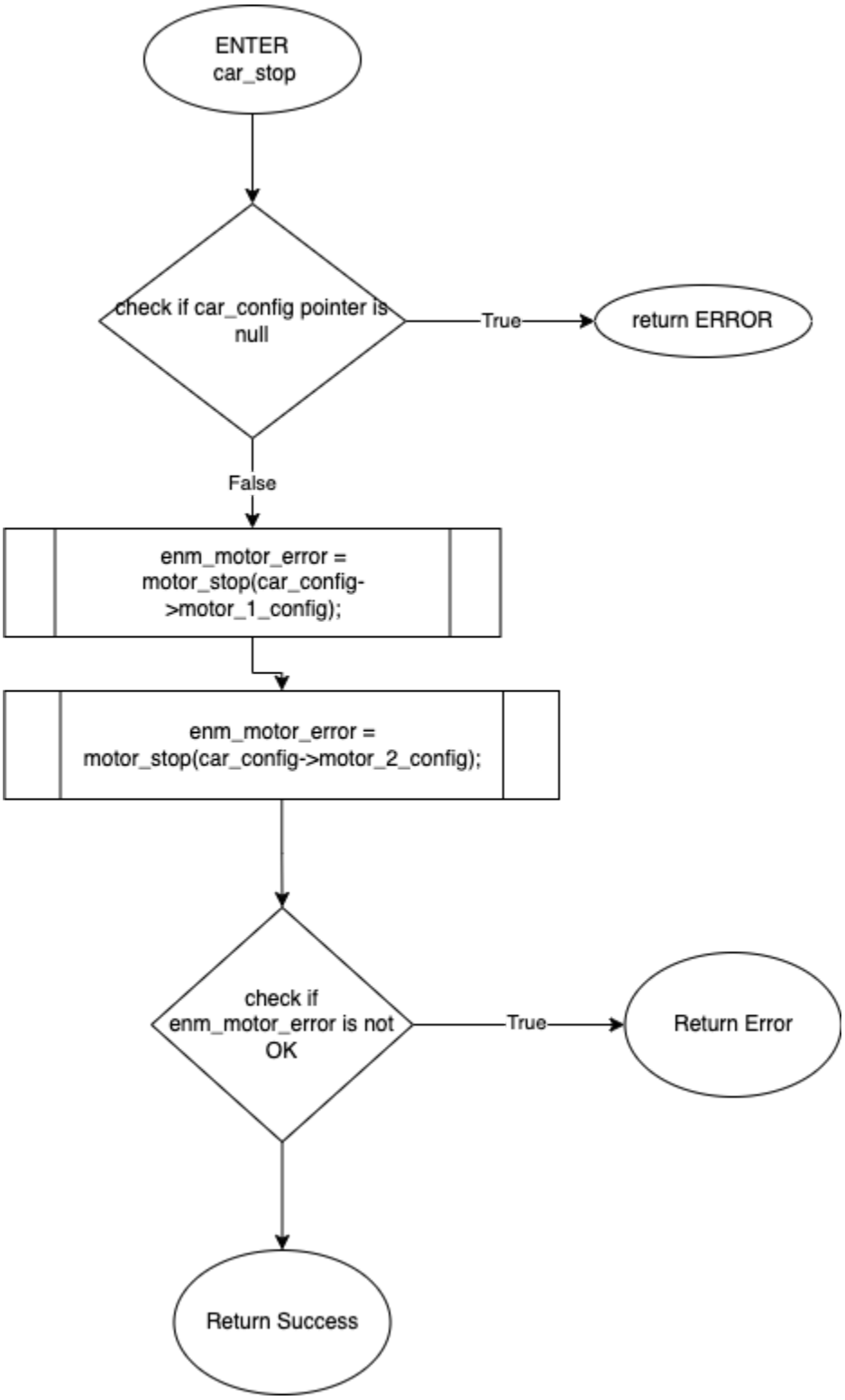
APP_vidInit

APP_vidStart

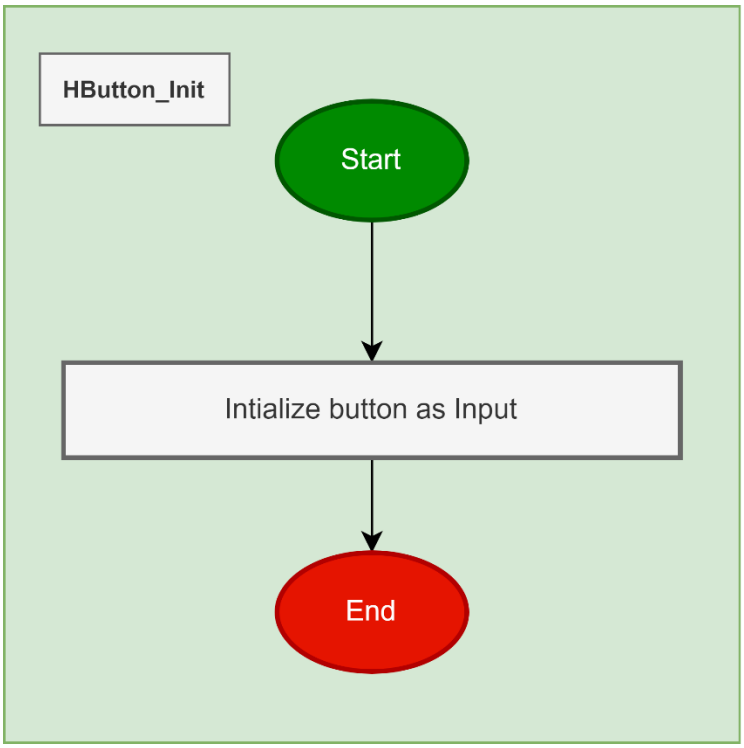
Manager

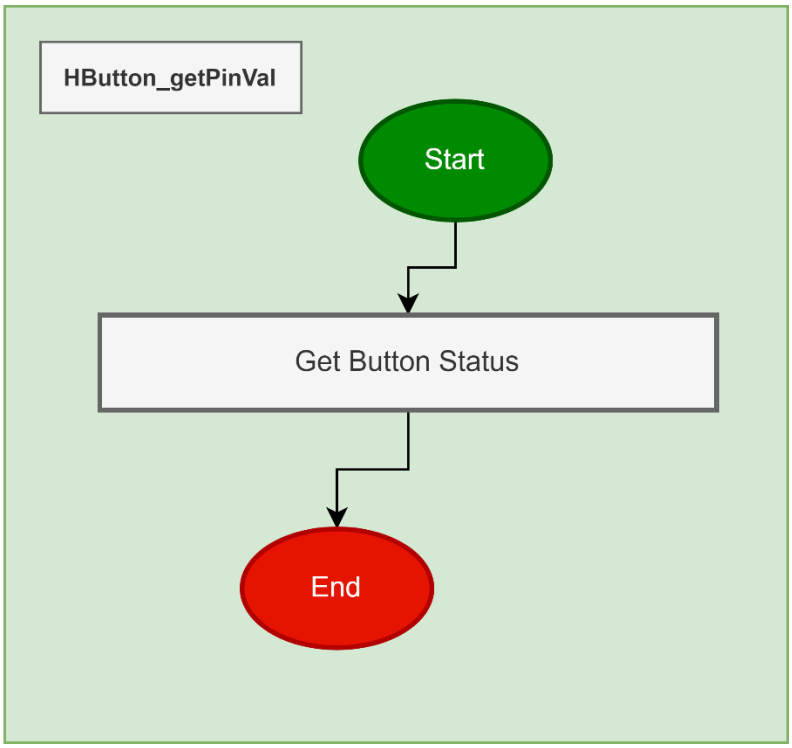

Car Manager
ENTER
car_init
check if car_config pointer is null
True
return ERROR
False
enm_motor_error = motor_init(car_config->motor_1_config);
enm_motor_error = motor_init(car_config->motor_2_config);
check if enm_motor_error is not OK
True
Return Error
Return Success

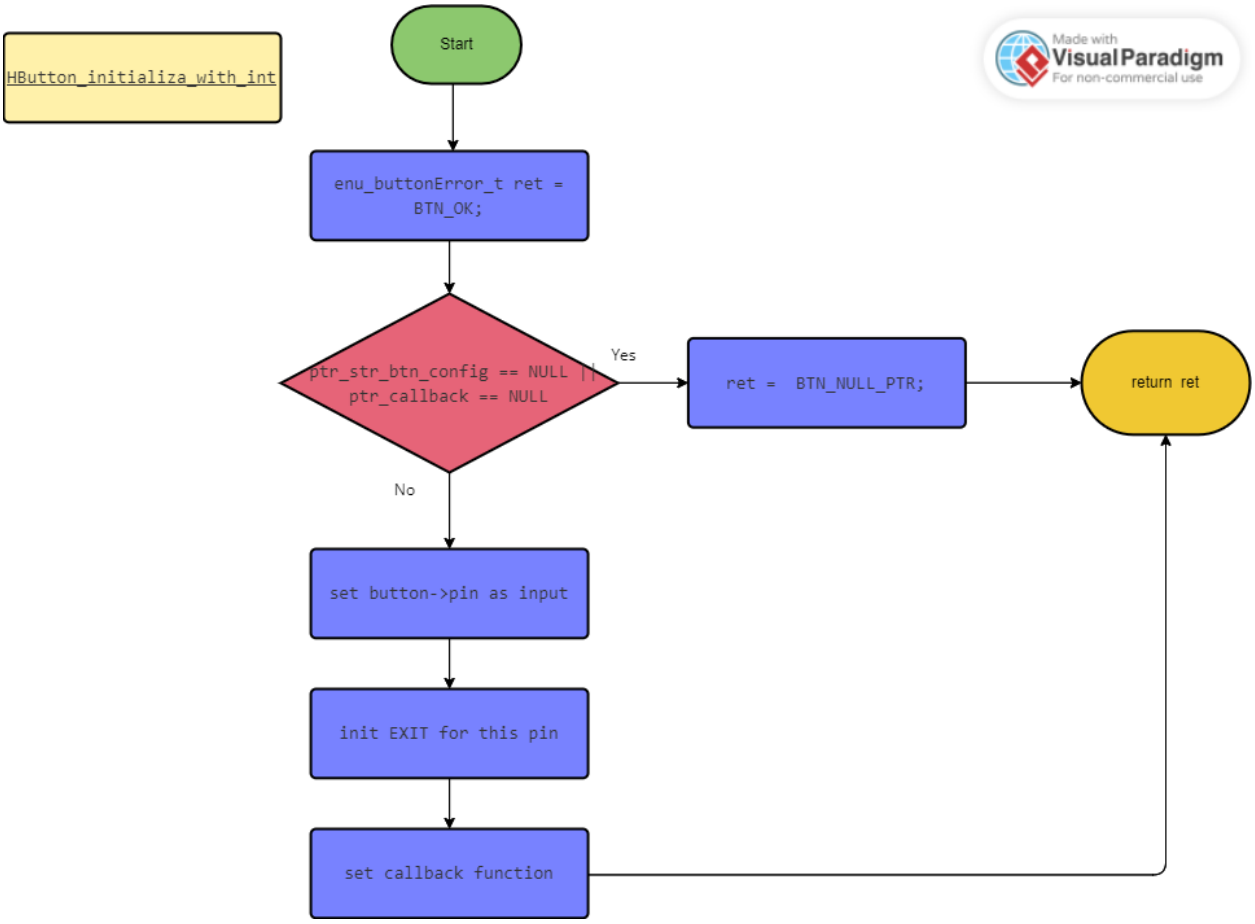

ENTER car_move
check if car_config pointer is null
True
return ERROR
False
switch (enm_car_direction)
CAR_STOP
enm_motor_error = motor_stop(car_config->motor_1_config);
enm_motor_error = motor_stop(car_config->motor_2_config);
CAR_FORWARD
enm_motor_error = motor_move(car_config->motor_1_config, MOTOR_FORWARD);
enm_motor_error = motor_move(car_config->motor_2_config, MOTOR_FORWARD);
CAR_BACKWARD
enm_motor_error = motor_move(car_config->motor_1_config, MOTOR_BACKWARD);
enm_motor_error = motor_move(car_config->motor_2_config, MOTOR_BACKWARD);
CAR_RIGHT
enm_motor_error = motor_move(car_config->motor_1_config, MOTOR_FORWARD);
enm_motor_error = motor_move(car_config->motor_2_config, MOTOR_BACKWARD);
CAR_LEFT
enm_motor_error = motor_move(car_config->motor_1_config, MOTOR_BACKWARD);
enm_motor_error = motor_move(car_config->motor_2_config, MOTOR_FORWARD);
check if enm_motor_error is not OK
True
Return Error
Return Success

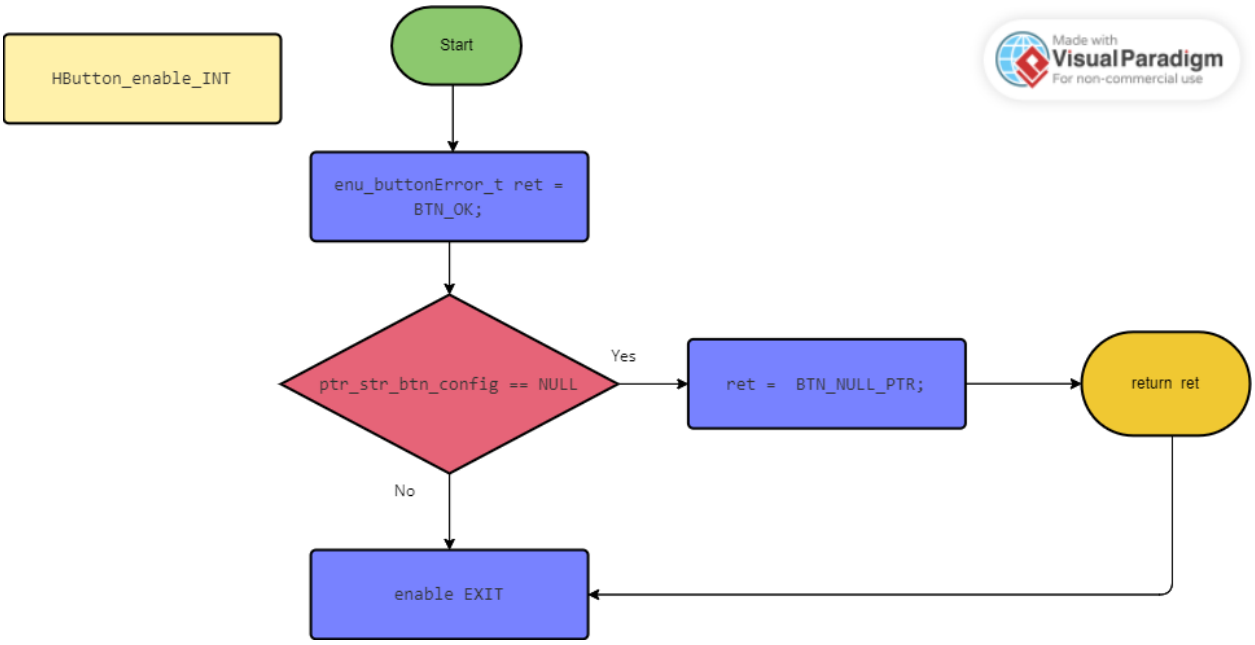

ENTER
car_stop
check if car_config pointer is null
True
return ERROR
False
enm_motor_error = motor_stop(car_config->motor_1_config);
enm_motor_error = motor_stop(car_config->motor_2_config);
check if enm_motor_error is not OK
True
Return Error
Return Success

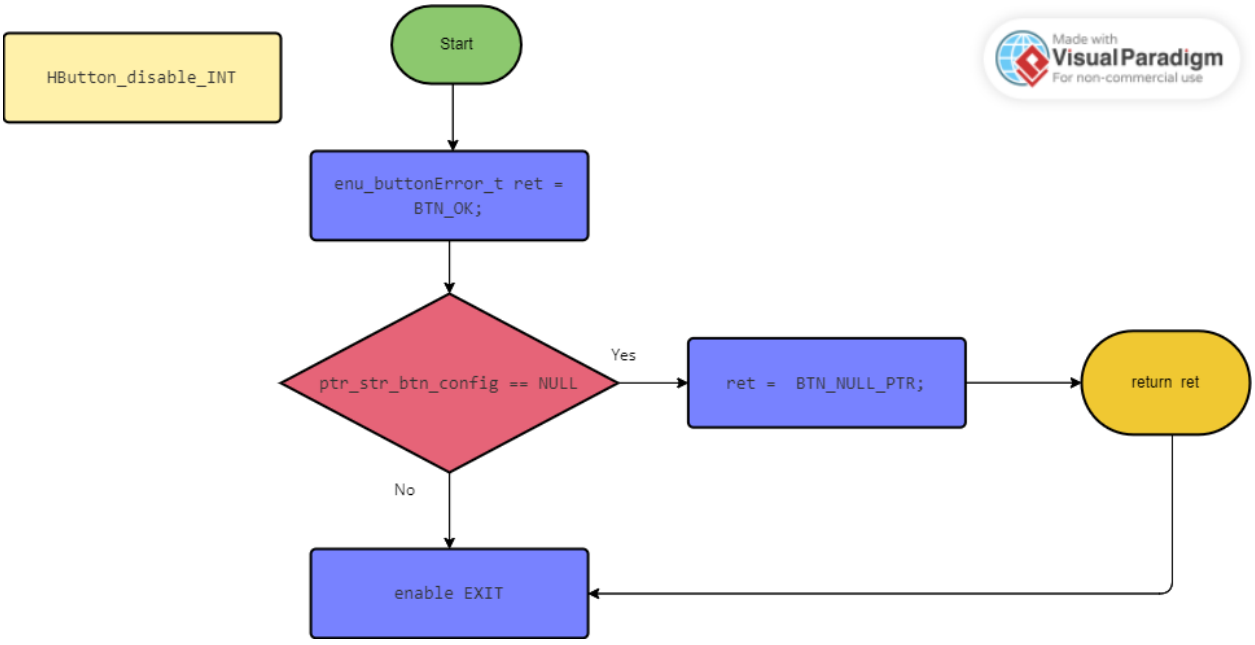
Button module



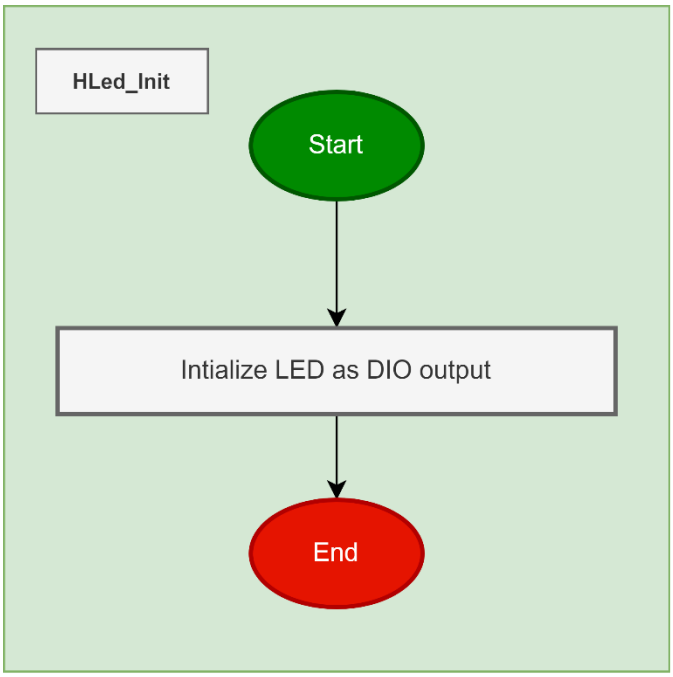


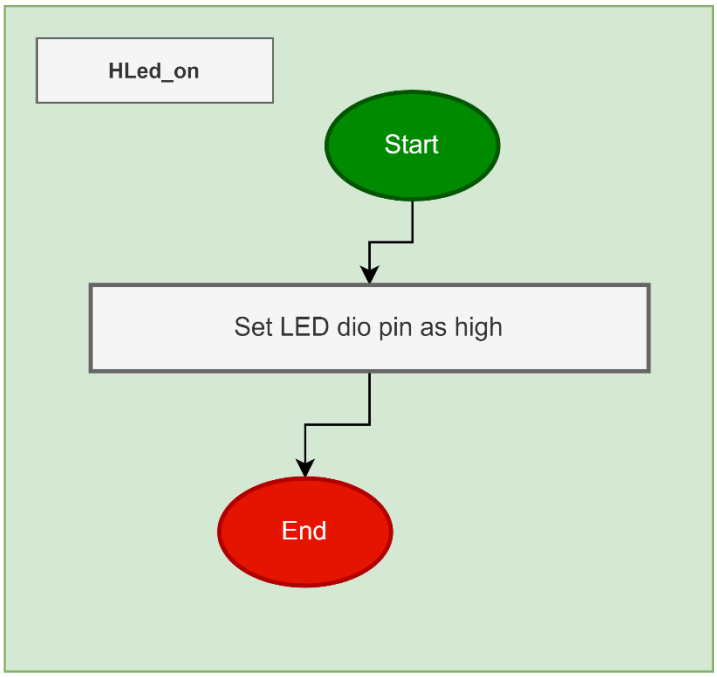

HButton_initializa_with_int
Start
enu_buttonError_t ret = BTN_OK;
ptr_str_btn_config == NULL || ptr_callback == NULL
Yes
ret = BTN_NULL_PTR;
return ret
No
set button->pin as input
init EXIT for this pin
set callback function
Made with Visual Paradigm For non-commercial use

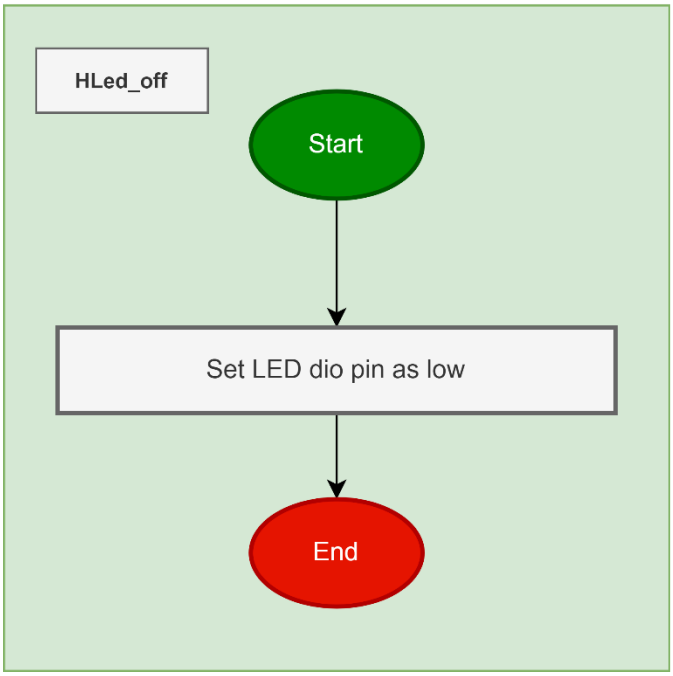

HButton_enable_INT
Start
enu_buttonError_t ret = BTN_OK;
ptr_str_btn_config == NULL
Yes
ret = BTN_NULL_PTR;
return ret
No
enable EXIT
Made with Visual Paradigm
For non-commercial use

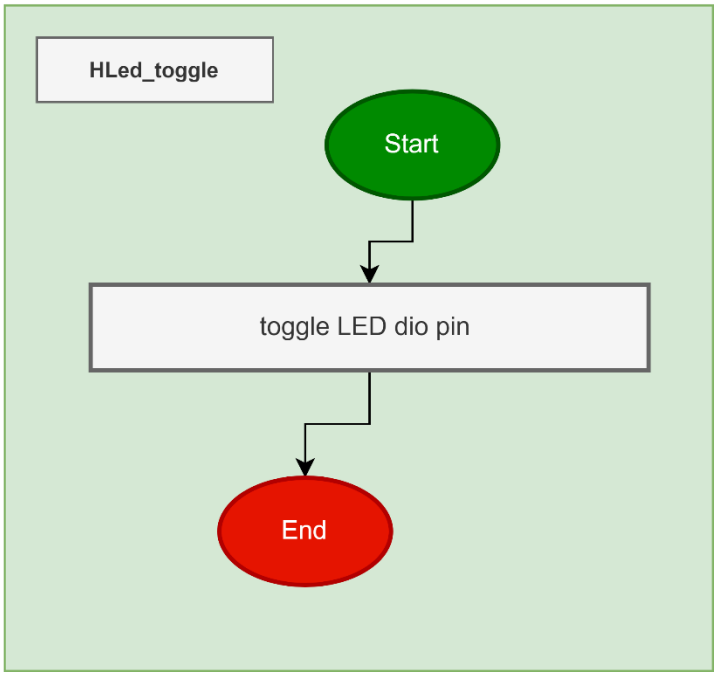

HButton_disable_INT
Start
enu_buttonError_t ret = BTN_OK;
ptr_str_btn_config == NULL
Yes
ret = BTN_NULL_PTR;
return ret
No
enable EXIT
Made with Visual Paradigm
For non-commercial use

HLED module

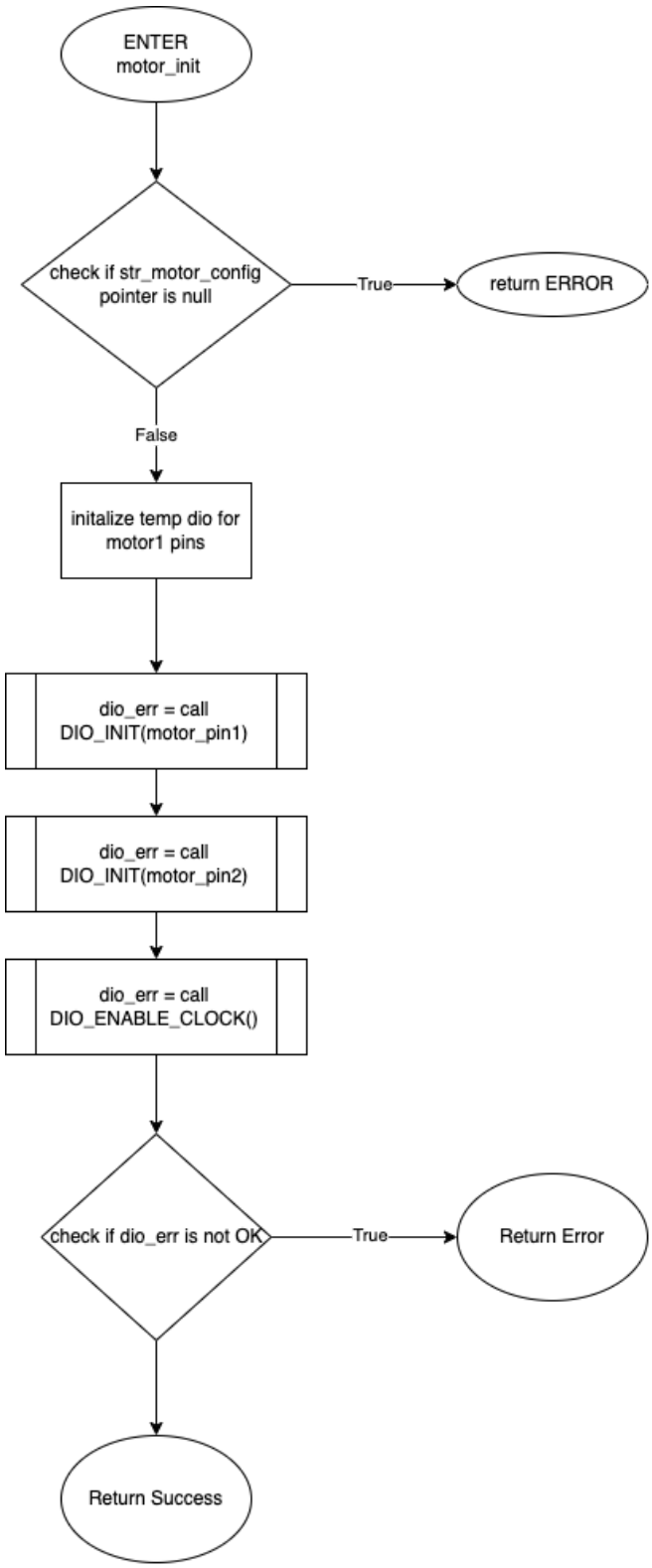

HLed_Init
Start
Intialize LED as DIO output
End


HLed_on
Start
Set LED dio pin as high
End

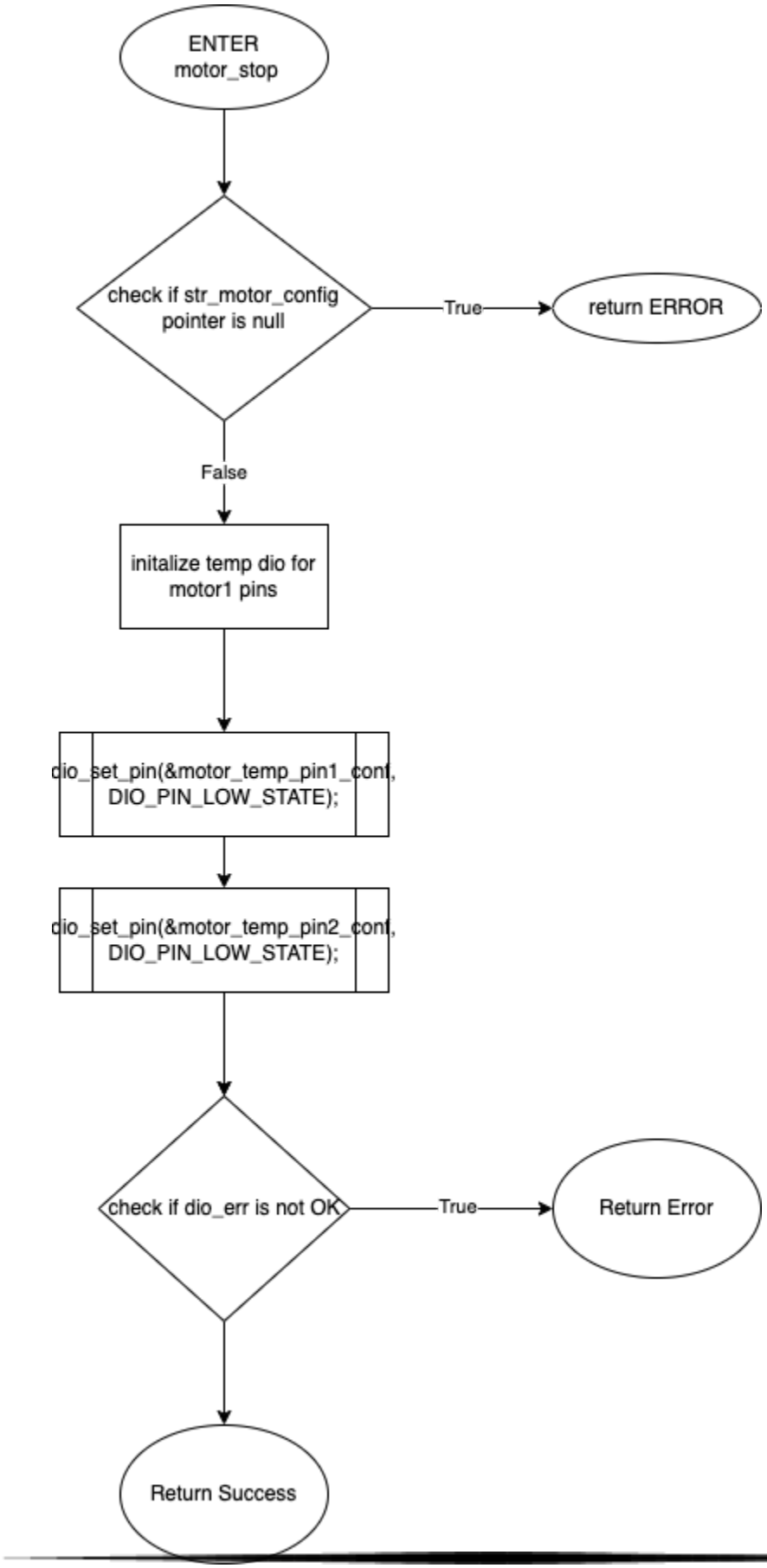

HLed_off
Start
Set LED dio pin as low
End


HLed_toggle
Start
toggle LED dio pin
End

MOTOR


ENTER
motor_init
check if str_motor_config
pointer is null
True
return ERROR
False
initalize temp dio for
motor1 pins
dio_err = call
DIO_INIT(motor_pin1)
dio_err = call
DIO_INIT(motor_pin2)
dio_err = call
DIO_ENABLE_CLOCK()
check if dio_err is not OK
True
Return Error
Return Success

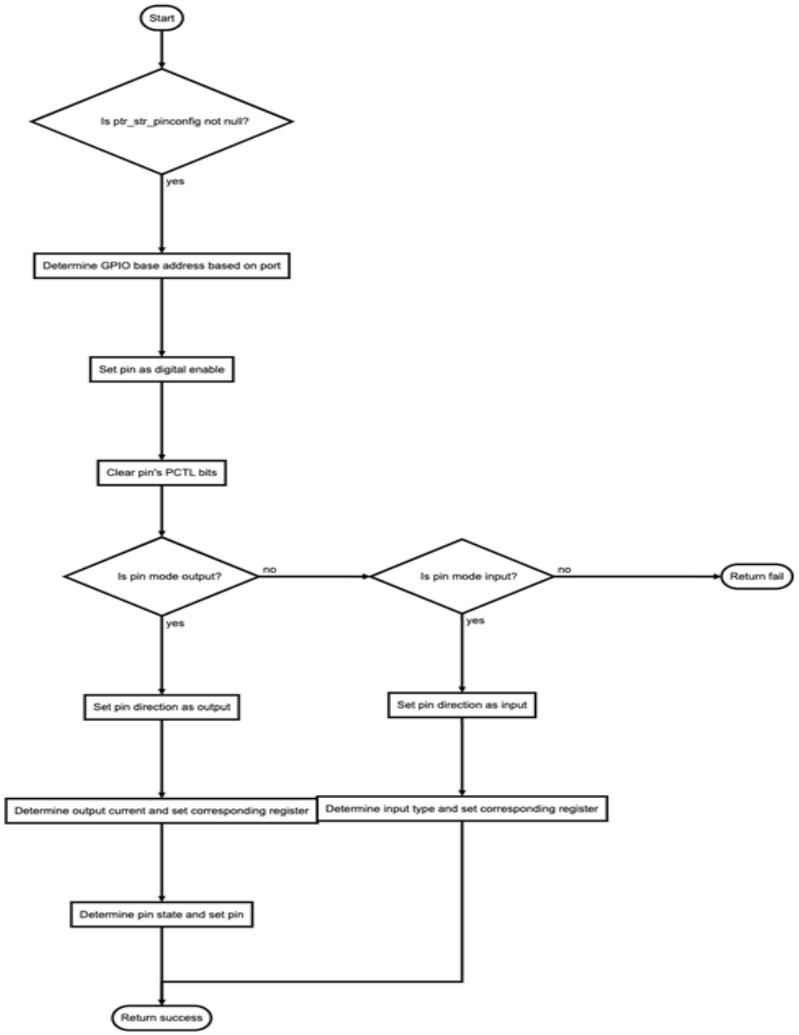
ENTER
motor_move
check if str_motor_config
pointer is null
True
return ERROR
False
initalize temp dio for
motor1 pins
switch (motor_dir)
MOTOR_FORWARD
MOTOR_BACKWARD
dio_set_pin(&motor_temp_pin1_conf,
DIO_PIN_HIGH_STATE);
dio_set_pin(&motor_temp_pin2_conf,
DIO_PIN_LOW_STATE);
dio_set_pin(&motor_temp_pin1_conf,
DIO_PIN_LOW_STATE);
dio_set_pin(&motor_temp_pin2_conf,
DIO_PIN_HIGH_STATE);
check if dio_err is not OK
True
Return Error
Return Success


ENTER
motor_stop
check if str_motor_config
pointer is null
True
return ERROR
False
initalize temp dio for
motor1 pins
dio_set_pin(&motor_temp_pin1_cont,
DIO_PIN_LOW_STATE);
dio_set_pin(&motor_temp_pin2_cont,
DIO_PIN_LOW_STATE);
check if dio_err is not OK
True
Return Error
Return Success

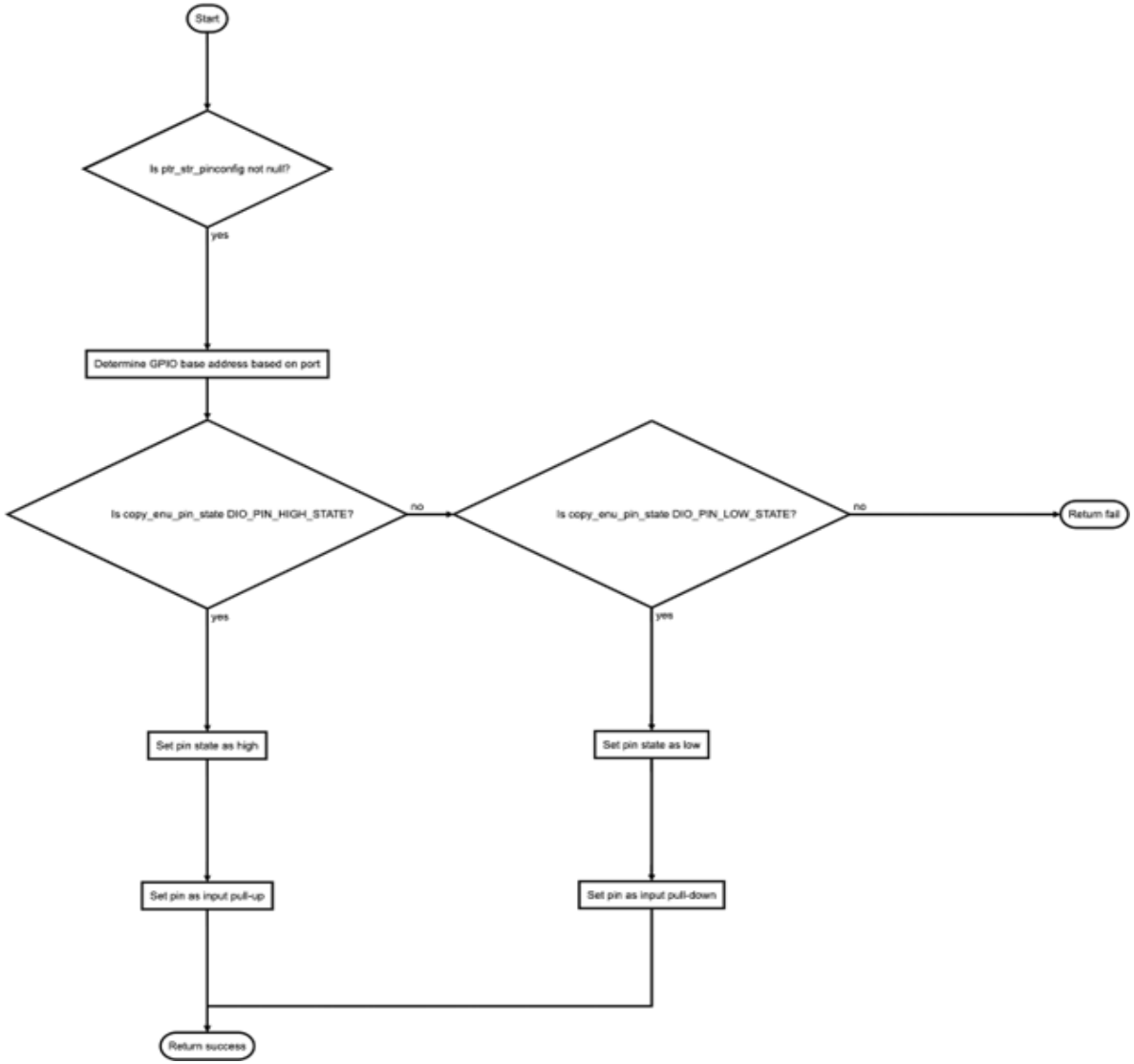
MCAL

GPIO module

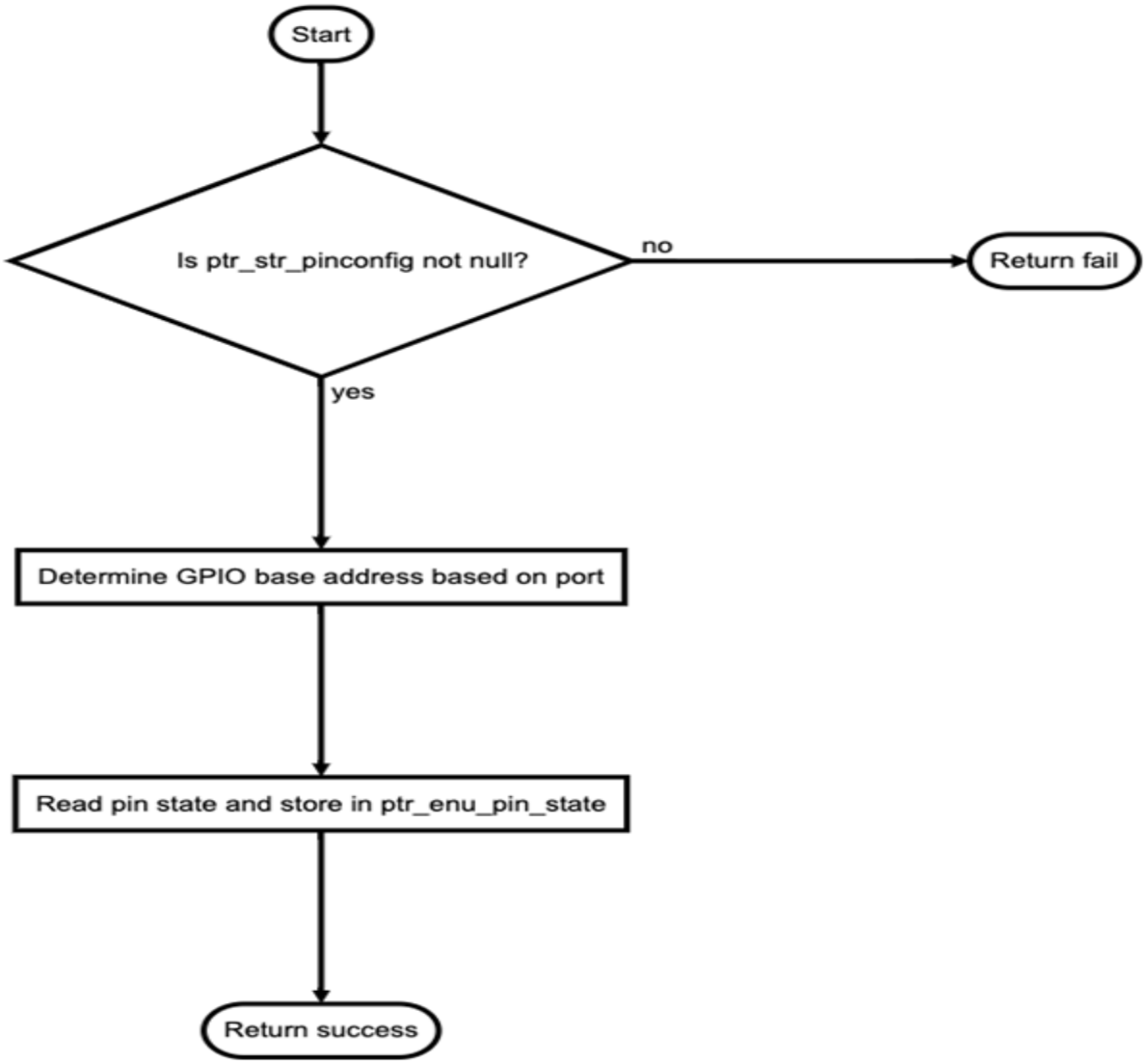
dio_init_pin



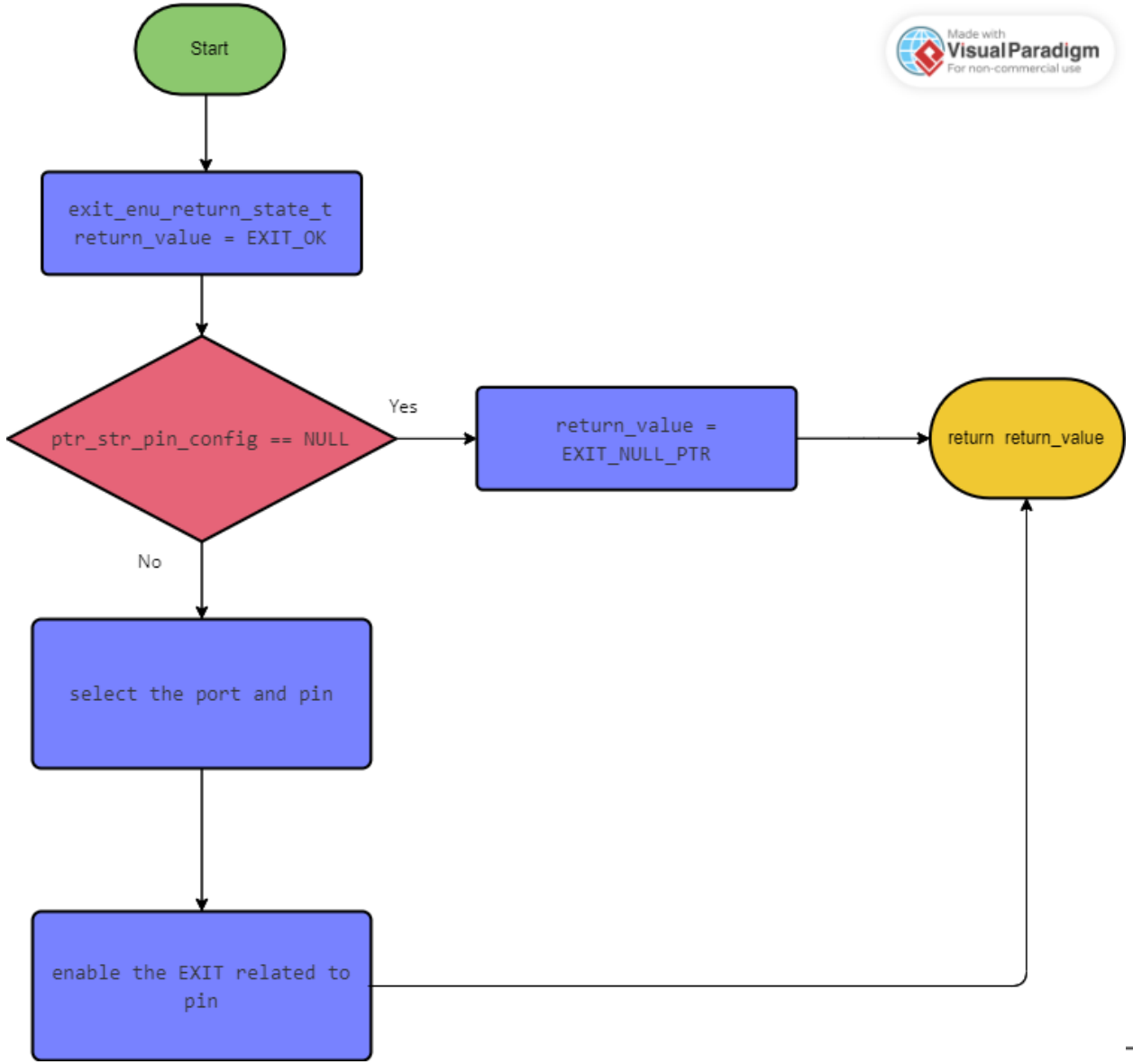
dio_set_pin


Start
Is ptr_str_pinconfig not null?
yes
Determine GPIO base address based on port
Is copy_enu_pin_state DIO_PIN_HIGH_STATE?
no
Is copy_enu_pin_state DIO_PIN_LOW_STATE?
no
Return fail
yes
yes
Set pin state as high
Set pin state as low
Set pin as input pull-up
Set pin as input pull-down
Return success

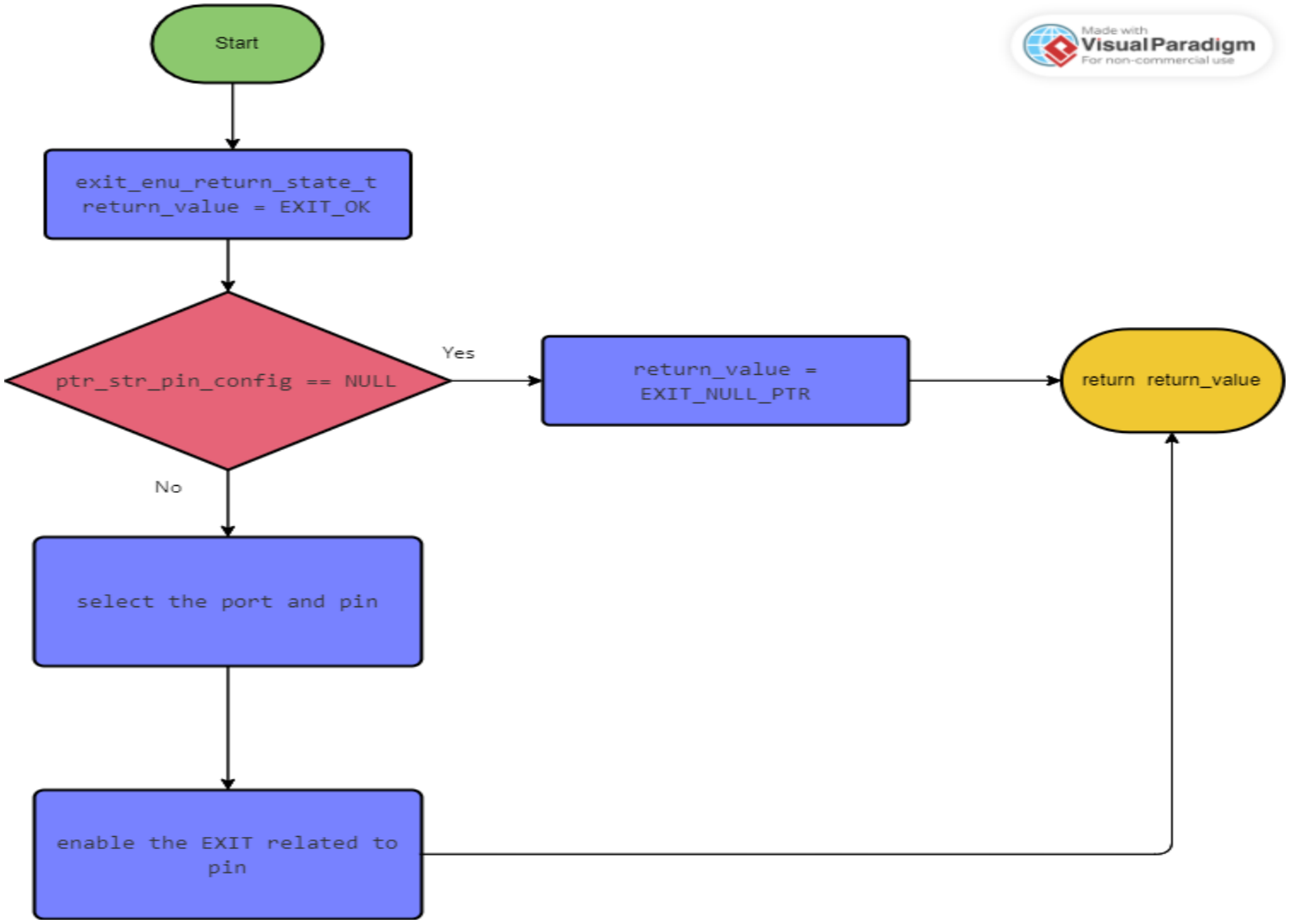
dio_read_pin


Start
Is ptr_str_pinconfig not null?
no
Return fail
yes
Determine GPIO base address based on port
Read pin state and store in ptr_enu_pin_state
Return success

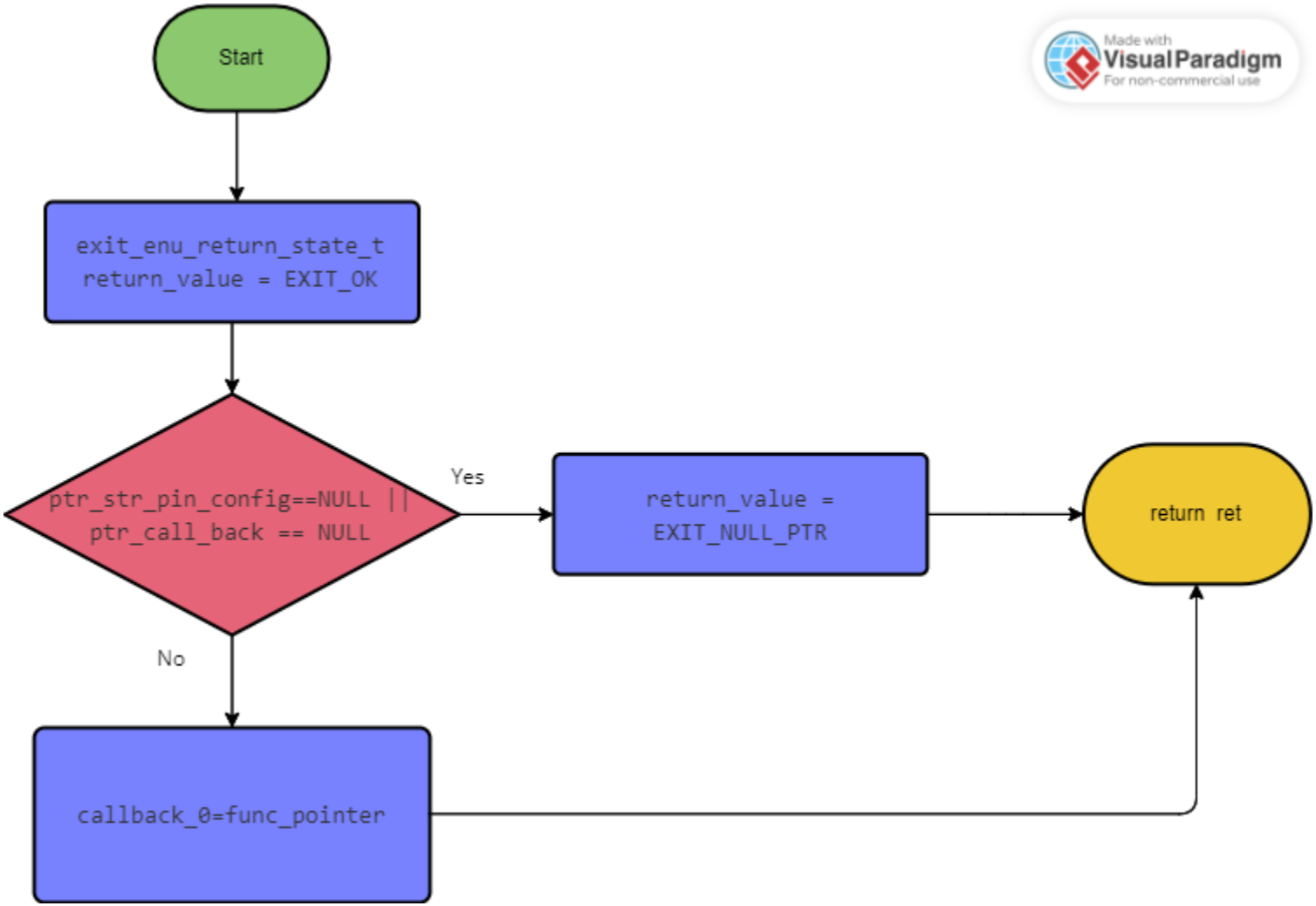
exit_enable_int



exit_disable_int

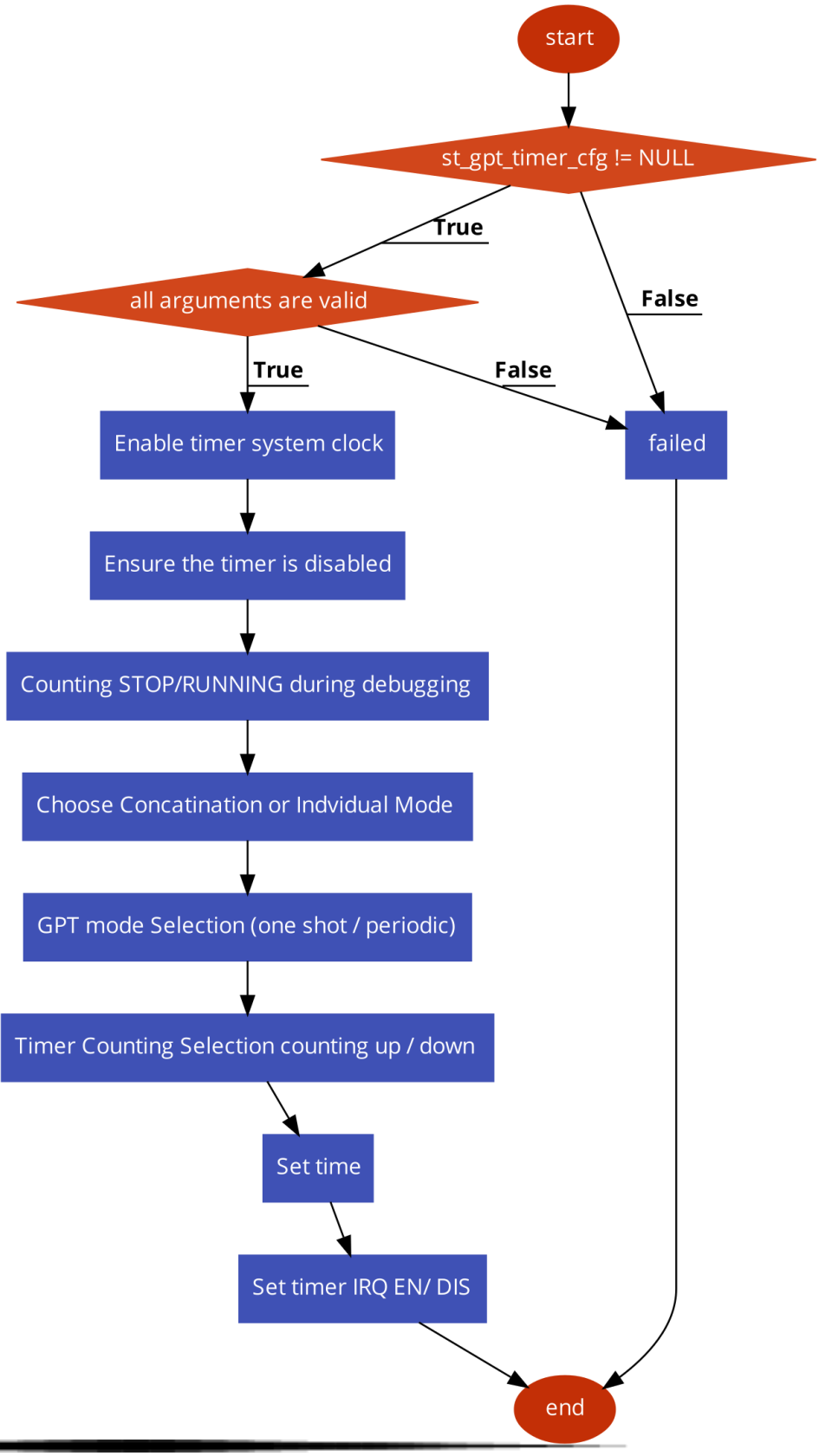


exit_set_callback

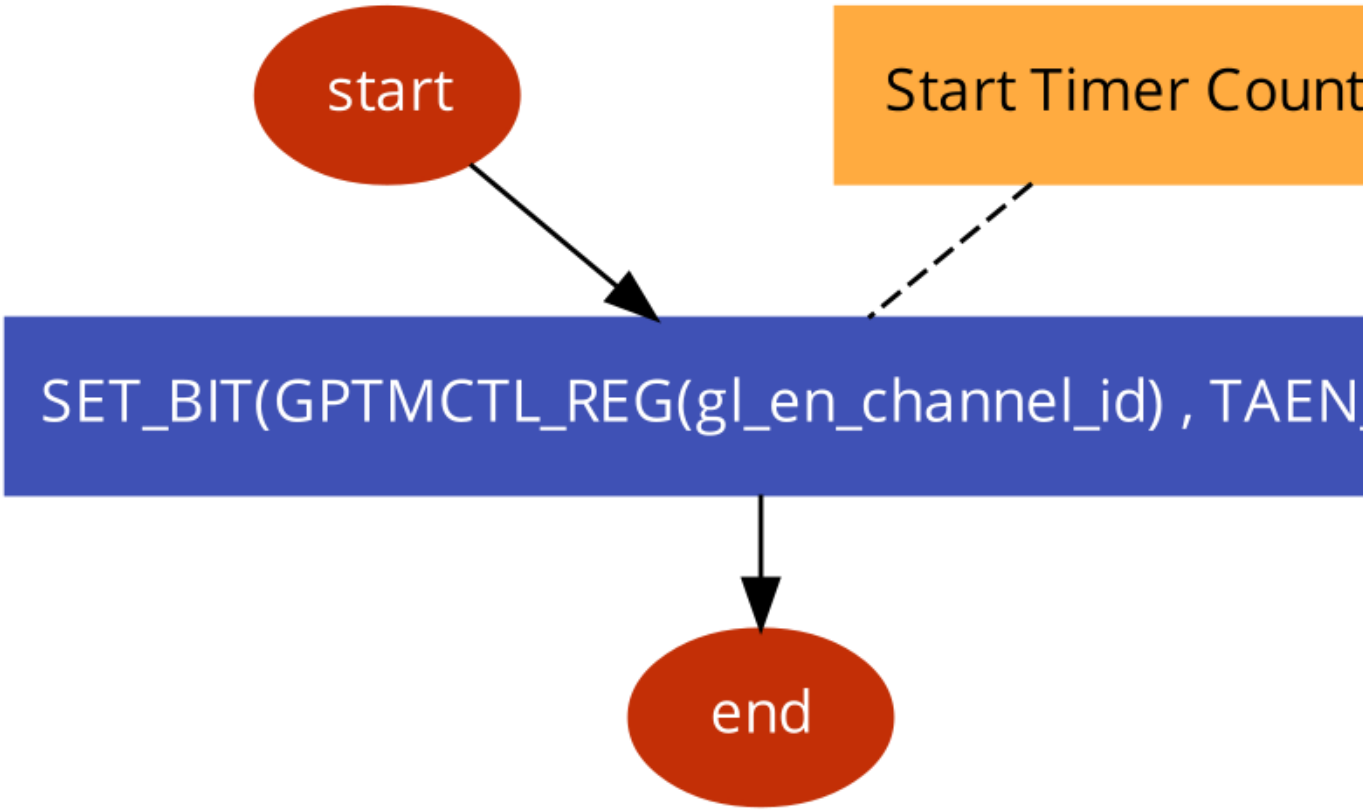


GPT module

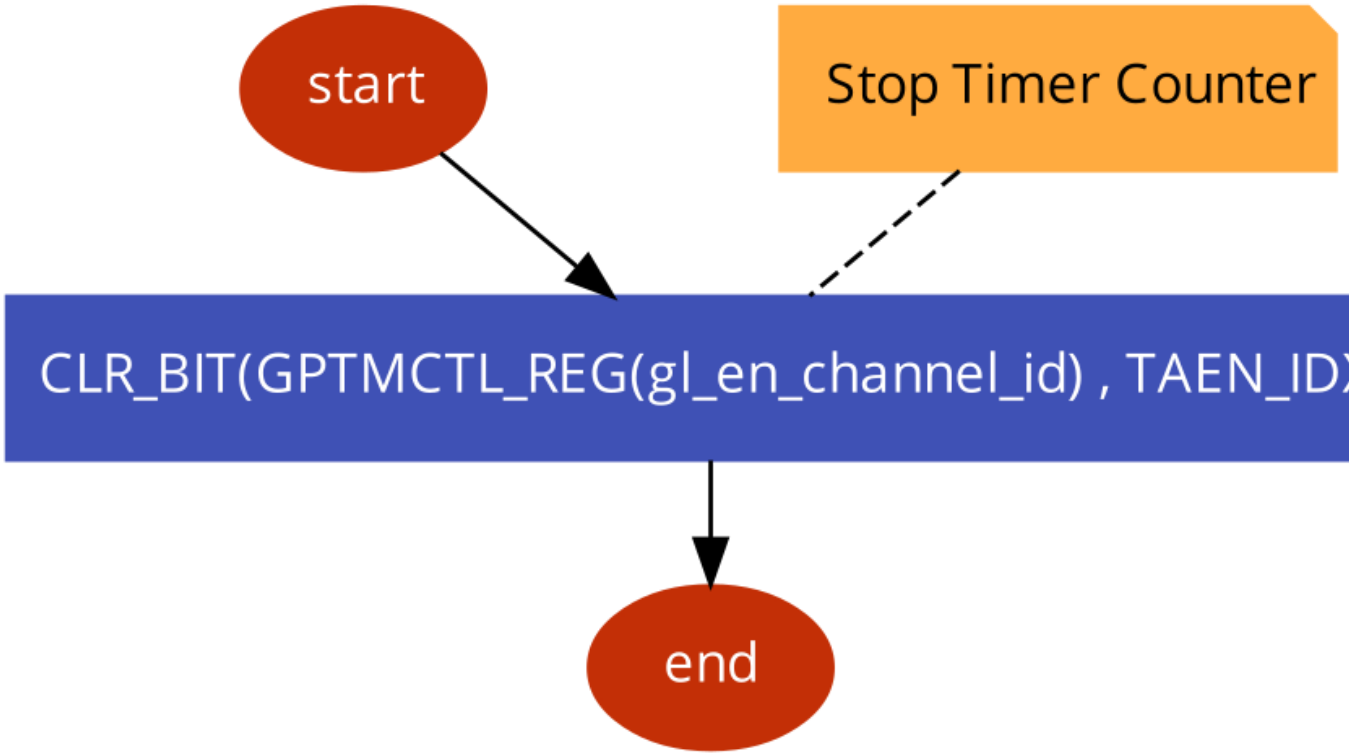
GPT_u8Init



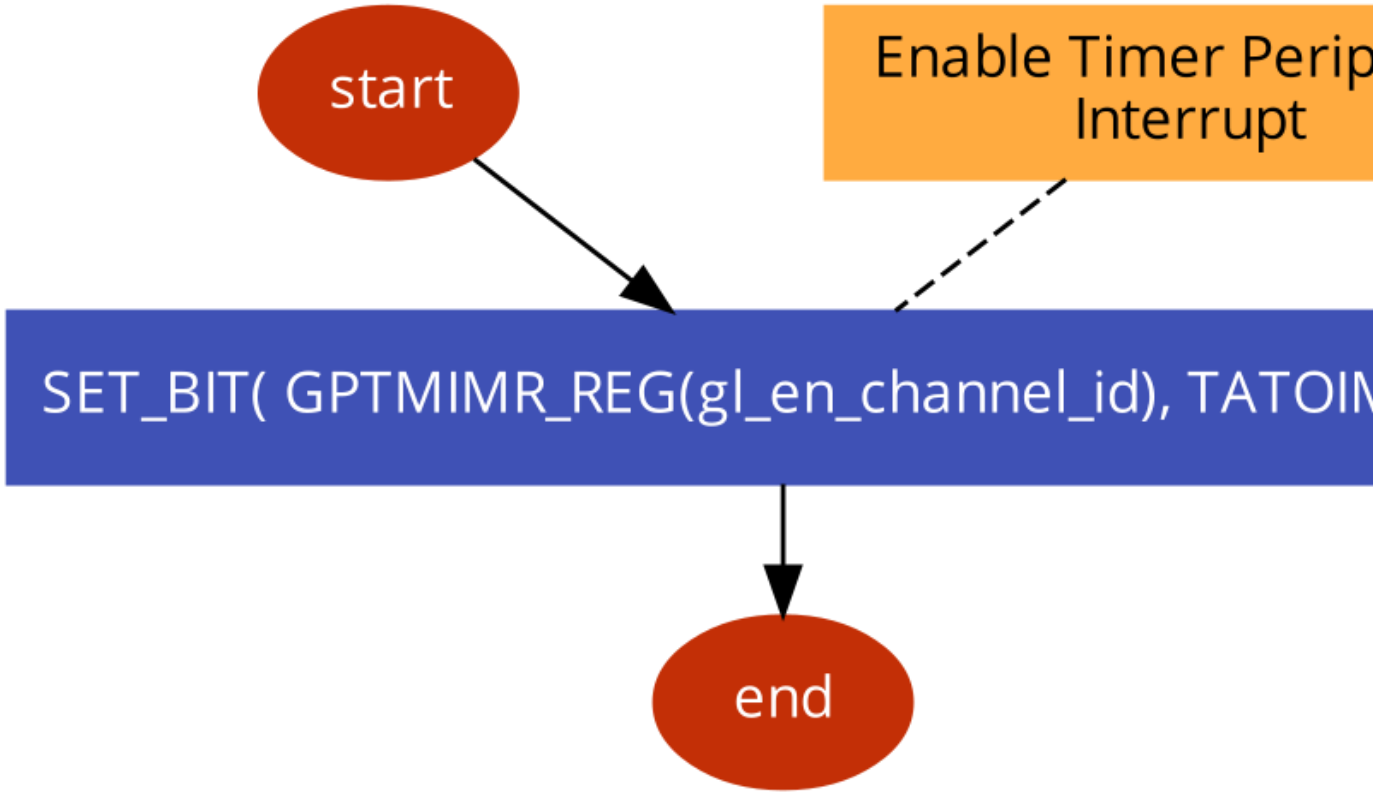
GPT_u8Start


start
Start Timer Count
SET_BIT(GPTMCTL_REG(gl_en_channel_id) , TAEN
end

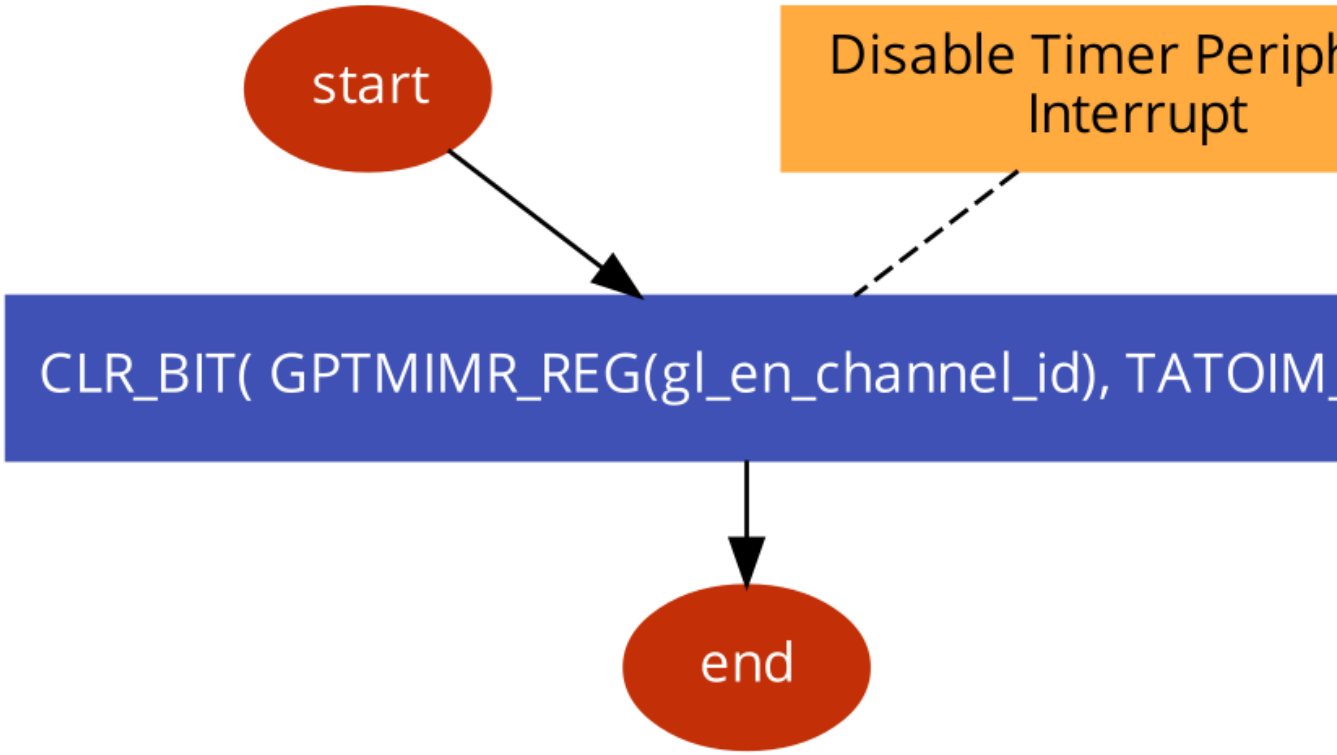
GPT_vidStop


start
Stop Timer Counter
CLR_BIT(GPTMCTL_REG(gl_en_channel_id) , TAEN_ID
end

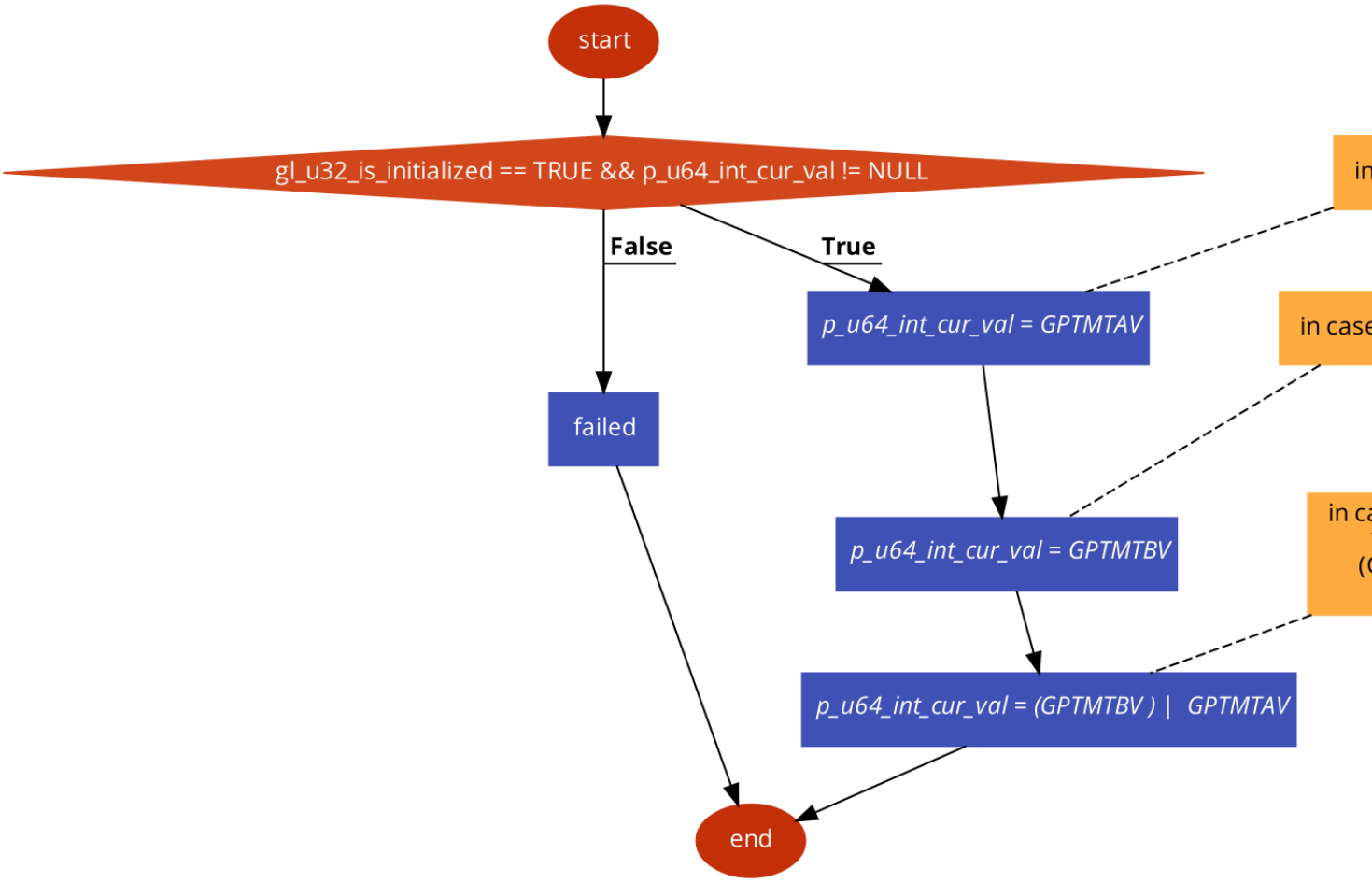
GPT_vidIRQEnable



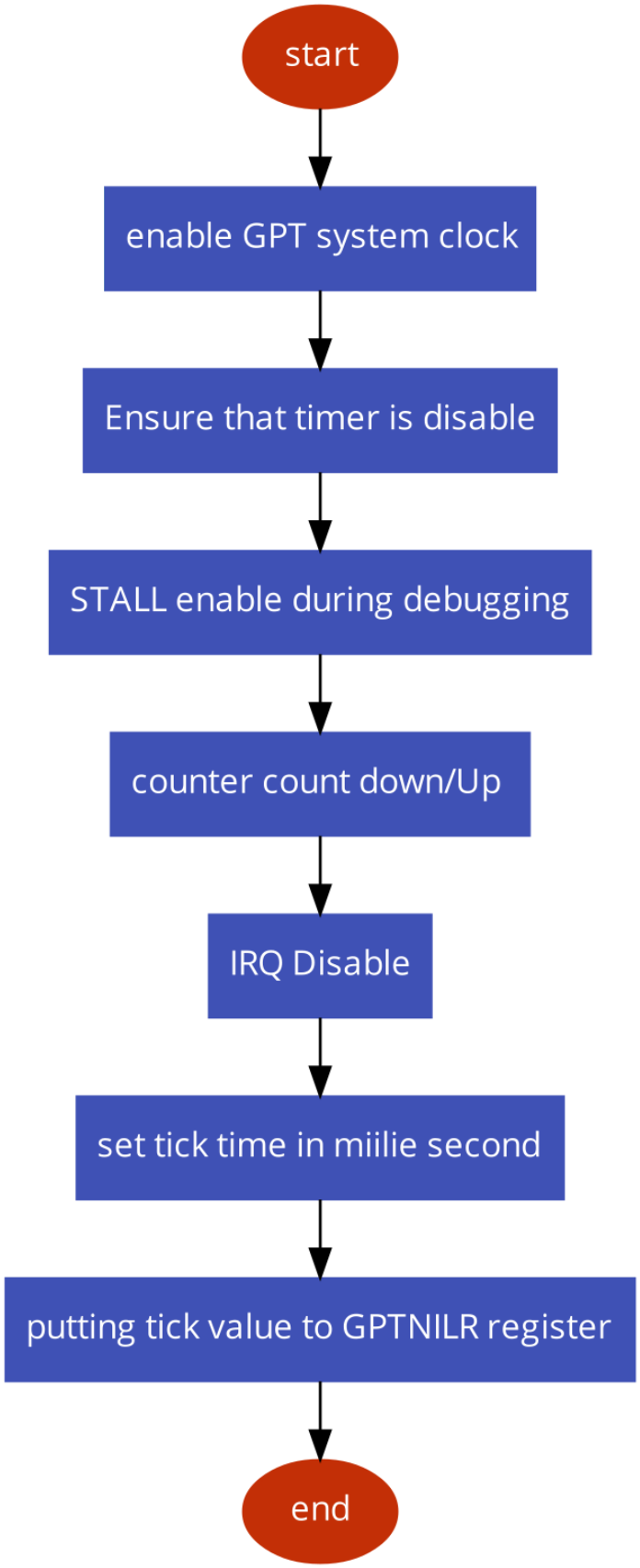
GPT_vidIRQDisable



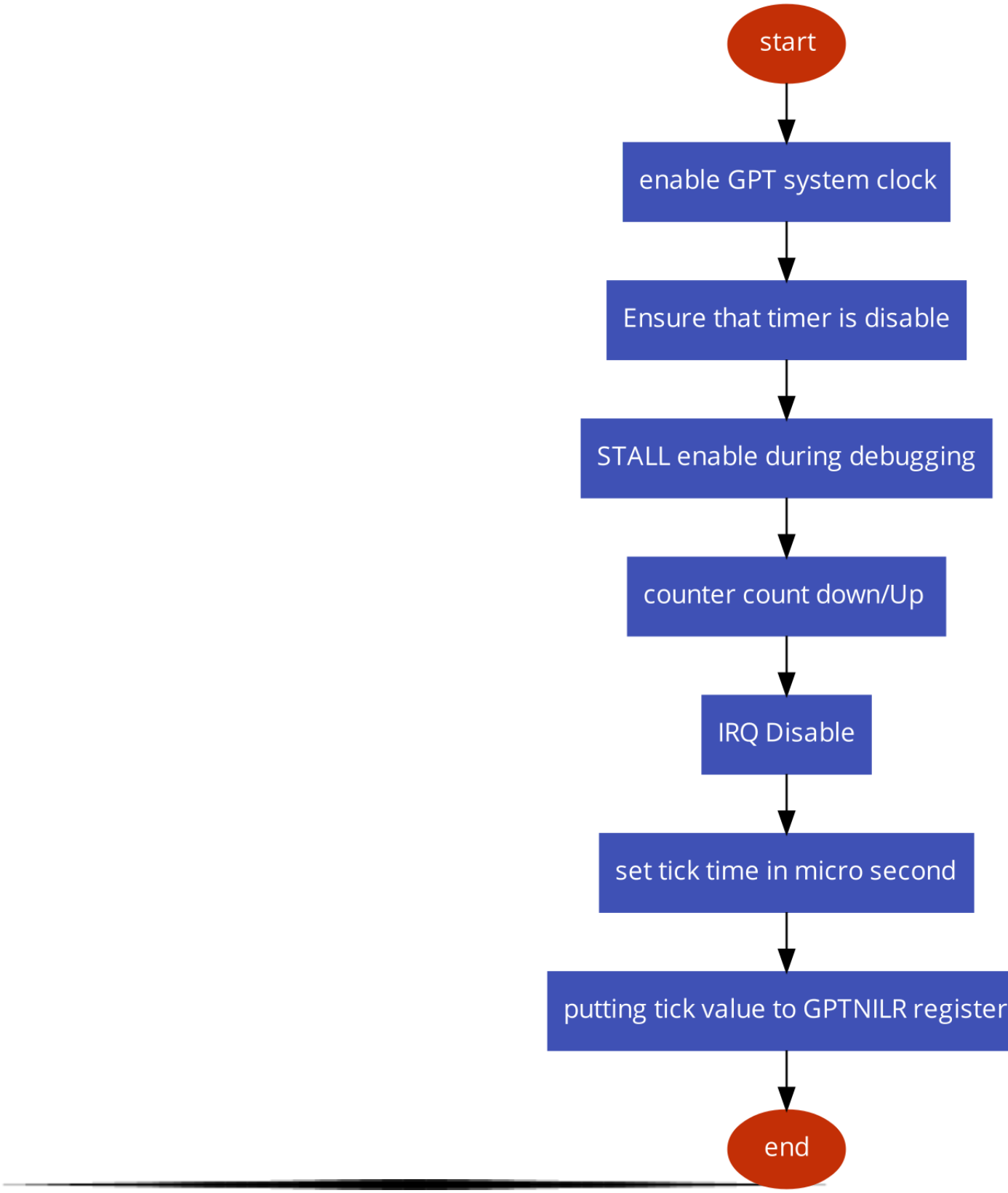
GPT_u8GetCurrentVal



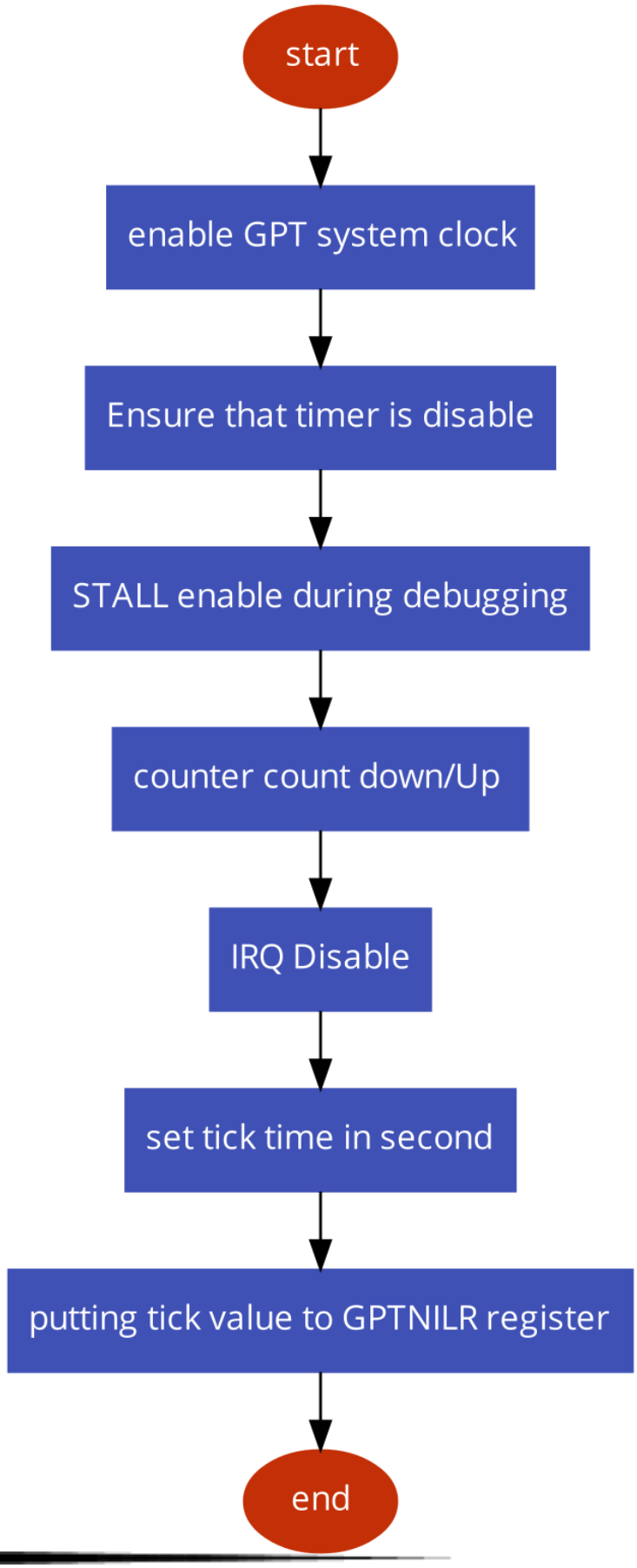
GPT_u8Delay_ms



GPT_u8Delay_us



GPT_u8Delay_s



Pre-compiling configuration

MCAL

MGPIO module

GPIO_BUS_TYPE

Name	GPIO_BUS_TYPE
Type	MACRO
Description	Define GPIO_bus
Configuration	***GPIO_APB***
	GPIO_AHB
Found in	mgpio_private.h

GPT module

GPT_TIMER_INDV_CONC_SELECTION

Name	GPT_TIMER_INDV_CONC_SELECTION
Type	MACRO
Description	Individual Timer Mode or Concatenation Timer mode selection
Configuration	***GPT_TIMER_INDVIDUAL_TIMER_A***
	GPT_TIMER_INDVIDUAL_TIMER_B
	GPT_TIMER_CONCATINATION
Found in	gpt_Interface.h

GPT_TIMER_COUNT_SELECTION

Name	GPT_TIMER_COUNT_SELECTION
Type	MACRO
Description	Timer Counting UP / Counting DOWN
Configuration	***GPT_COUNT_DOWN***
	GPT_COUNT_UP
Found in	gpt_Interface.h

Linking Configuration

MCAL

MGPIO module

dio_str_pin_Config_t

<table>
<tr><td>Name</td><td colspan="2">dio_str_pin_Config_t</td></tr>
<tr><td>Type</td><td colspan="2">struct</td></tr>
<tr><td>Description</td><td colspan="2">GPIO pin configuration</td></tr>
<tr><td rowspan="5">Members</td><td colspan="2">enu_port</td></tr>
<tr><td colspan="2">enu_pin</td></tr>
<tr><td colspan="2">enu_pin_dir_mode</td></tr>
<tr><td rowspan="2">un_input_output_type</td><td>dio_str_output_type_and_speed_and_state_t</td></tr>
<tr><td>dio_enu_input_type_t</td></tr>
<tr><td>Found in</td><td colspan="2">dio_Interface.h</td></tr>
</table>

dio_enu_pinx_t

Name	dio_enu_pinx_t
Type	enum
Description	GPIO pin Selection
Configuration	***DIO_PIN_0 ~ DIO_PIN_7***
Found in	dio_Interface.h

dio_enu_pin_mode_t

Name	dio_enu_pin_mode_t
Type	enum
Description	GPIO Mode Selection
Configuration	***DIO_PIN_INPUT***

	DIO_PIN_OUTPUT
	DIO_PIN_AFM
	DIO_PIN_ANALOG
Found in	dio_Interface.h

dio_enu_output_current_t

Name	dio_enu_output_current_t
Type	enum
Description	GPIO Ampere mode Selection
Configuration	***DIO_PIN_2MA***
	DIO_PIN_4MA
	DIO_PIN_8MA
Found in	dio_Interface.h

EXIT module

exit_str_pin_Config_t

Name	exit_str_pin_Config_t
Type	struct
Description	GPIO Interrupt mode Selection
Members	***enu_port***
	enu_pin
	enu_trigger_mode
	enu_idle_state
Found in	EXIT_Interface.h

exit_enu_trigger_mode_t

Name	exit_enu_trigger_mode_t
Type	enum
Description	EXIT trigger mode
Configuration	***EXIT_RISING_EDGE***
	EXIT_FALLING_EDGE
	EXIT_BOTH_EDGE
Found in	EXIT_Interface.h

exit_enu_idle_mode_t

Name	exit_enu_idle_mode_t
Type	enum
Description	GPIO Ampere mode Selection
Configuration	***EXIT_PULL_UP***
	EXIT_PULL_DOWN
	EXIT_OPEN_DRAIN
Found in	EXIT_Interface.h

SYSTICK module

st_gpt_timer_cfg_t

Name	st_gpt_timer_cfg_t
Type	struct
Description	GPT configuration
Members	*en_gpt_ch_id*
	en_gpt_mode
	en_gpt_stall
	en_gpt_time_x
	u32_set_time
	en_gpt_irq;
	ptr_func
Found in	gpt_Interface.h

en_gpt_ch_id_t

Name	en_gpt_ch_id_t
Type	enum
Description	GPT channel Id selection
Configuration	*GPT_CHANNEL_0 ~ GPT_CHANNEL_5*
	GPT_WIDE_CHANNEL_0 ~ GPT_WIDE_CHANNEL_5
Found in	gpt_Interface.h

en_gpt_irq_t

Name	en_gpt_irq_t
Type	enum
Description	GPT IRQ EN/DIS
Configuration	*GPT_IRQ_DISABLE*
	GPT_IRQ_ENABLE
Found in	gpt_Interface.h

en_gpt_mode_t

Name	en_gpt_mode_t
Type	enum
Description	GPT channel mode selection
Configuration	*GPT_CH_MODE_ONE_SHOT*
	GPT_CH_MODE_PERIODIC
Found in	gpt_Interface.h

en_gpt_stall_t

Name	en_gpt_stall_t
Type	enum
Description	Counting stop or still running during debug
Configuration	*GPT_STALL_DISABLE*
	GPT_STALL_ENABLE
Found in	gpt_Interface.h

en_gpt_time_x_t

Name	en_gpt_time_x_t
Type	enum
Description	Set time in (micro seconds, milli seconds, seconds) selection
Configuration	***GPT_TIME_US***
	GPT_TIME_MS
	GPT_TIME_S
Found in	gpt_Interface.h