SPRINTS

RGB LED Blinking Design

June 2023

Presented by
Abdelrhman Walaa

Presented to
Sprints

Table Of Content

RGB LED Blinking Design

1. Project Introduction

The RGB LED Blinking project aims to showcase the capabilities of the Tiva C TM4C123GXL LaunchPad board by implementing a user interface that utilizes the SW1 button. The project enables users to blink the RGB LED by pressing the SW1 button. This is achieved by generating a PWM (Pulse Width Modulation) signal on the green LED pin, allowing for adjustment of its brightness levels. By combining the power of the Tiva C microcontroller, the versatility of the RGB LED, and the PWM functionality, this project offers an interactive and visually engaging experience, empowering users to tailor the green LED's brightness to their preference.

1.1. System Requirements

1.1.1. Hardware Requirements

1. **TivaC** board
2. **One** input button **SW1**
3. **One** RGB **LED**

1.1.2. Software Requirements

1. The RGB **LED** is **off** initially.
2. The **PWM** signal has a **500** ms duration
3. The system has **four** states
    a. SW1 - **First** press
        i. The **Green** LED will be **on** with a **30**% duty cycle.
    b. SW1 - **Second** press
        i. The **Green** LED will be **on** with a **60**% duty cycle.
    c. SW1 -**Third** press
        i. The **Green** LED will be **on** with a **90**% duty cycle
    d. SW1 - **Fourth** press will be **off**
        i. The **Green** LED will be **off**
    e. On the **Fifth** press, system state will return to state **1**.

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) 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.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

2.1.2. Layered Architecture

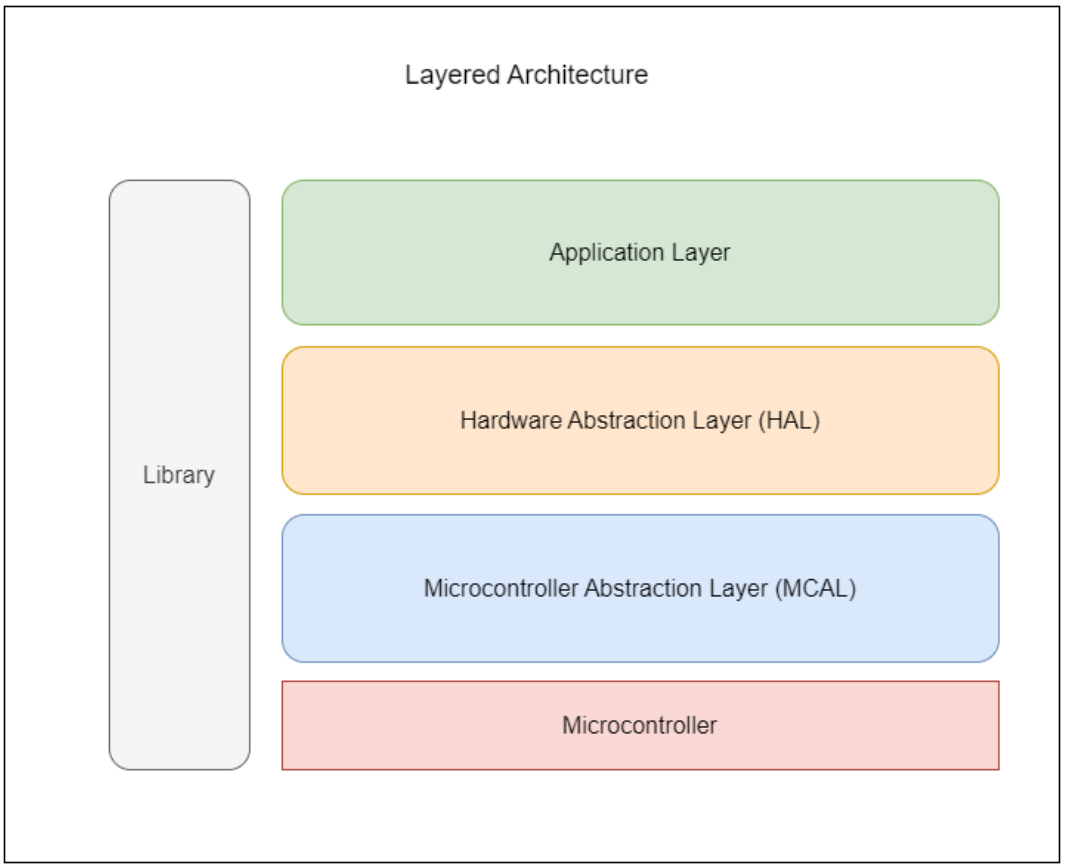


Figure 1. Layered Architecture Design

2.2. System Modules

2.2.1. Definition

A *Module* is a distinct assembly of components that can be easily added, removed or replaced in a larger system. Generally, a *Module* is not functional on its own.

In computer hardware, a *Module* is a component that is designed for easy replacement.

2.2.2. Design

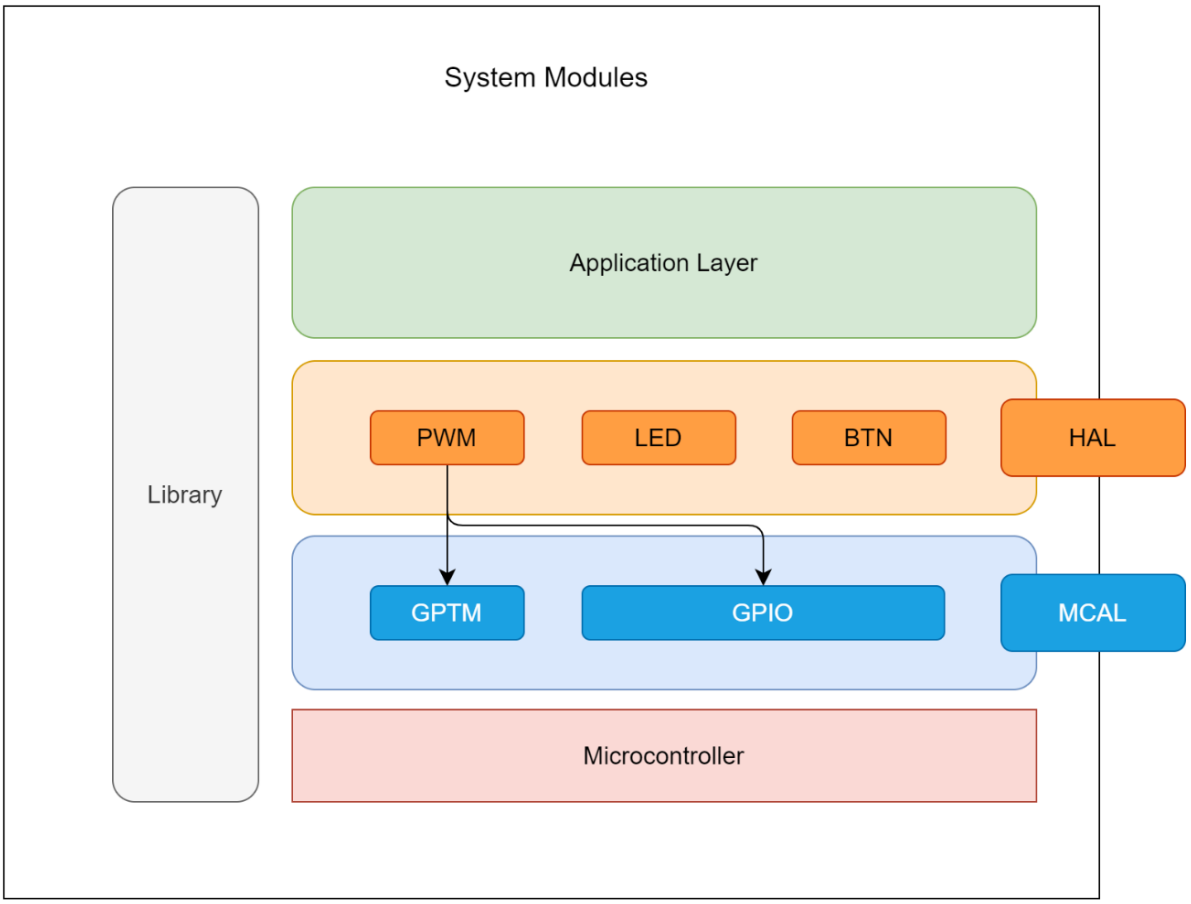


Figure 2. System Modules Design

2.3. Modules Description

2.3.1. GPIO Module

A *GPIO* (General Purpose Input/Output) driver is a fundamental component in microcontroller projects, providing the ability to interface with external devices and control digital signals. It serves as an interface between the microcontroller and the outside world, enabling the manipulation of input and output signals for various applications. The *GPIO* driver in a microcontroller project facilitates the control and configuration of the *GPIO* pins available on the microcontroller. These pins can be individually programmed as inputs or outputs to interface with external devices such as sensors, actuators, or communication modules.

2.3.2. GPTM Module

The *GPTM* (General-Purpose Timer) peripheral is a versatile timer module available in microcontrollers that provides precise timing and event counting capabilities. It is commonly used in embedded systems for a wide range of applications, including timing measurements, event capturing, and generating periodic or one-shot interrupts. The *GPTM* peripheral typically consists of multiple timer modules, each with its own set of registers and functionality. These modules can operate independently or be synchronized to perform coordinated timing operations.

2.3.3. LED Module

An *LED* (Light Emitting Diode) driver is a crucial component in microcontroller projects that involve controlling *LED*s. It provides a means to control the brightness, color, and behavior of *LEDs*, allowing for dynamic visual effects and signaling. A well-designed *LED* driver simplifies the interfacing process and enables efficient and precise control over *LED* operations.

2.3.4. BTN Module

In most of the embedded electronic projects you may want to use a *BTN* (Push Button) switch to give user inputs to the microcontroller. Push Button is basically a small controlling device that is pressed to operate any electrical device.

2.3.4. PWM Module

The *PWM* (Pulse Width Modulation) peripheral is a widely used feature in microcontrollers that enables precise control of the output signal's pulse width. *PWM* is commonly employed in various applications such as motor control, LED dimming, audio synthesis, and power regulation.

2.4. Drivers' Documentation (APIs)

2.4.1 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.

2.4.2. MCAL APIs

2.4.2.1. GPIO Driver APIs

```
Precompile and LinKing Configurations:

/* GPIO Port Ids */
typedef enum
{
  GPIO_EN_PORTA = 0,
  GPIO_EN_PORTC,
  GPIO_EN_PORTD,
  GPIO_EN_PORTE,
  GPIO_EN_PORTF,
  GPIO_EN_INVALID_PORT_ID
} GPIO_enPortId_t;

/* GPIO Port Bus Ids */
typedef enum
{
  GPIO_EN_APB = 0,
  GPIO_EN_AHB,
  GPIO_EN_INVALID_BUS_ID
} GPIO_enPortBusId_t;

/* GPIO Port Modes */
typedef enum
{
  GPIO_EN_RUN_MODE = 0,
  GPIO_EN_SLEEP_MODE,
  GPIO_EN_DEEP_SLEEP_MODE,
  GPIO_EN_INVALID_PORT_MODE
} GPIO_enPortMode_t;
```

```c
/* GPIO Pin Ids */
typedef enum
{
  GPIO_EN_PIN0 = 0,
  GPIO_EN_PIN1,
  GPIO_EN_PIN2,
  GPIO_EN_PIN3,
  GPIO_EN_PIN4,
  GPIO_EN_PIN5,
  GPIO_EN_PIN6,
  GPIO_EN_PIN7,
  GPIO_EN_INVALID_PIN_ID
} GPIO_enPinId_t;

/* GPIO Pin Modes */
typedef enum
{
  GPIO_EN_DISABLE_PIN = 0,
  GPIO_EN_ENABLE_PIN,
  GPIO_EN_INVALID_PIN_MODE
} GPIO_enPinMode_t;

/* GPIO Pin Directions */
typedef enum
{
  GPIO_EN_INPUT_DIR = 0,
  GPIO_EN_OUTPUT_DIR,
  GPIO_EN_INVALID_PIN_DIR
} GPIO_enPinDirection_t;

/* GPIO Pin Values */
typedef enum
{
  GPIO_EN_PIN_LOW,
  GPIO_EN_PIN_HIGH,
  GPIO_EN_INVALID_PIN_VALUE
} GPIO_enPinValue_t;

/* GPIO Pin Pad */
typedef enum
{
  GPIO_EN_DISABLE_PIN_PAD =0,
  GPIO_EN_ENABLE_PULL_UP,
  GPIO_EN_ENABLE_PULL_DOWN,
  GPIO_EN_ENABLE_OPEN_DRAIN,
  GPIO_EN_INVALID_PIN_PAD
} GPIO_enPinPad_t;
```

```c
/* GPIO Pin Alternate Function Modes */
typedef enum
{
  GPIO_EN_PIN_GPIO_MODE = 0,
  GPIO_EN_PIN_ALT_MODE,
  GPIO_EN_INVALID_PIN_ALT_MODE
} GPIO_enPinAlternateMode_t;

/* GPIO Pin Drives */
typedef enum
{
  GPIO_EN_DISABLE_PIN_DRIVE = 0,
  GPIO_EN_2_MA_DRIVE,
  GPIO_EN_4_MA_DRIVE,
  GPIO_EN_8_MA_DRIVE,
  GPIO_EN_INVALID_PIN_DRIVE
} GPIO_enPinDrive_t;

/* GPIO Pin Interrupt Modes */
typedef enum
{
  GPIO_EN_DISABLE_INT = 0,
  GPIO_EN_ENABLE_INT,
  GPIO_EN_INVALID_INT_MODE
} GPIO_enPinInterruptMode_t;

/* GPIO Pin Interrupt Action */
typedef void (*GPIO_vpfPinInterruptAction_t) (void);

/* GPIO Port Linking Configurations Structure */
typedef struct
{
  GPIO_enPortId_t     en_a_portId;
  GPIO_enPortBusId_t  en_a_portBusId;
  GPIO_enPortMode_t   en_a_portMode;
} GPIO_stPortLinkConfig_t;

/* GPIO Pin Linking Configurations Structure */
typedef struct
{
  GPIO_enPortId_t               en_a_portId;
  GPIO_enPortBusId_t            en_a_portBusId;
  GPIO_enPinId_t                en_a_pinId;
  GPIO_enPinMode_t              en_a_pinMode;
  GPIO_enPinDirection_t         en_a_pinDirection;
  GPIO_enPinValue_t             en_a_pinValue;
  GPIO_enPinAlternateMode_t     en_a_pinAlternateMode;
  GPIO_enPinDrive_t             en_a_pinDrive;
  GPIO_enPinPad_t               en_a_pinPad;
  GPIO_enPinInterruptMode_t     en_a_pinInterruptMode;
```

```
  GPIO_vpfPinInterruptAction_t   vpf_a_pinInterruptAction;
} GPIO_stPinLinkConfig_t;

/* GPIO Error States */
typedef enum
{
  GPIO_EN_NOK = 0,
  GPIO_EN_OK
} GPIO_enErrorState_t;

| Name: GPIO_initalization
| Input: Pointer to Array of st PortLinkConfig and u8 NumberOfPorts
| Output: en Error or No Error
| Description: Function to initialize GPIO Port peripheral using Linking
|              Configurations.
|
GPIO_enErrorState_t GPIO_initalization (const GPIO_stPortLinkConfig_t
*past_a_portsLinkConfig, u8 u8_a_numberOfPorts)

| Name: GPIO_configurePin
| Input: Pointer to Array of st PinLinkConfig and u8 NumberOfPins
| Output: en Error or No Error
| Description: Function to configure GPIO Pin peripheral using Linking
|              Configurations.
|
GPIO_enErrorState_t GPIO_configurePin (const GPIO_stPinLinkConfig_t
*past_a_pinsLinkConfig, u8 u8_a_numberOfPins)

| Name: GPIO_setPinValue
| Input: en PortId, en BusId, and en PinId
| Output: en Error or No Error
| Description: Function to set Pin value.
|
GPIO_enErrorState_t GPIO_setPinValue (GPIO_enPortId_t en_a_portId,
GPIO_enPortBusId_t en_a_busId, GPIO_enPinId_t en_a_pinId)

| Name: DIO_u8GetPinValue
| Input: u8 PortId, u8 PinId, and Pointer to u8 ReturnedPinValue
| Output: u8 Error or No Error
| Description: Function to get Pin value.
|
GPIO_enErrorState_t GPIO_getPinValue (GPIO_enPortId_t en_a_portId,
GPIO_enPortBusId_t en_a_busId, GPIO_enPinId_t en_a_pinId, GPIO_enPinValue_t
*pen_a_returnedPinValue)
```

```
| Name: GPIO_clearPinValue
| Input: en PortId, en BusId, and en PinId
| Output: en Error or No Error
| Description: Function to clear Pin value.
|
GPIO_enErrorState_t GPIO_clearPinValue (GPIO_enPortId_t en_a_portId,
GPIO_enPortBusId_t en_a_busId, GPIO_enPinId_t en_a_pinId)

| Name: GPIO_togglePinValue
| Input: en PortId, en BusId, and en PinId
| Output: en Error or No Error
| Description: Function to toggle Pin value.
|
GPIO_enErrorState_t GPIO_togglePinValue (GPIO_enPortId_t en_a_portId,
GPIO_enPortBusId_t en_a_busId, GPIO_enPinId_t en_a_pinId)
```

2.4.2.2. GPTM Driver APIs

```
Precompile and Linking Configurations:

/* GPTM Timer Ids */
typedef enum
{
  GPTM_EN_TIMER_0 = 0,
  GPTM_EN_TIMER_1,
  GPTM_EN_TIMER_2,
  GPTM_EN_TIMER_3,
  GPTM_EN_TIMER_4,
  GPTM_EN_TIMER_5,
  GPTM_EN_WIDE_TIMER_0,
  GPTM_EN_WIDE_TIMER_1,
  GPTM_EN_WIDE_TIMER_2,
  GPTM_EN_WIDE_TIMER_3,
  GPTM_EN_WIDE_TIMER_4,
  GPTM_EN_WIDE_TIMER_5,
  GPTM_EN_INVALID_TIMER_ID
} GPTM_enTimerId_t;

/* GPTM Timer Modes */
typedef enum
{
  GPTM_EN_ONE_SHOT_MODE = 0,
  GPTM_EN_PERIODIC_MODE,
  GPTM_EN_RTC_MODE,
  GPTM_EN_EDGE_COUNT_MODE,
  GPTM_EN_EDGE_TIME_MODE,
  GPTM_EN_PWM_MODE,
```

```
  GPTM_EN_INVALID_TIMER_MODE
} GPTM_enTimerMode_t;

/* GPTM Timer Uses */
typedef enum
{
  GPTM_EN_INDIVIDUAL_A = 0,
  GPTM_EN_INDIVIDUAL_B,
  GPTM_EN_CONCATUNATED,
  GPTM_EN_INVALID_TIMER_USE
} GPTM_enTimerUse_t;

/* GPTM Timer Interrupt Modes */
typedef enum
{
  GPTM_EN_DISABLED_INT = 0,
  GPTM_EN_ENABLED_INT,
  GPTM_EN_INVALID_TIMER_INT_MODE
} GPTM_enTimerInterruptMode_t;

/* GPTM Timer Interrupt Numbers */
typedef enum
{
  GPTM_EN_TIMER_INT_0_A = TIMER0A_IRQn,
  GPTM_EN_TIMER_INT_0_B = TIMER0B_IRQn,
  GPTM_EN_TIMER_INT_1_A = TIMER1A_IRQn,
  GPTM_EN_TIMER_INT_1_B = TIMER1B_IRQn,
  GPTM_EN_TIMER_INT_2_A = TIMER2A_IRQn,
  GPTM_EN_TIMER_INT_2_B = TIMER2B_IRQn,
  GPTM_EN_TIMER_INT_3_A = TIMER3A_IRQn,
  GPTM_EN_TIMER_INT_3_B = TIMER3B_IRQn,
  GPTM_EN_TIMER_INT_4_A = TIMER4A_IRQn,
  GPTM_EN_TIMER_INT_4_B = TIMER4B_IRQn,
  GPTM_EN_TIMER_INT_5_A = TIMER5A_IRQn,
  GPTM_EN_TIMER_INT_5_B = TIMER5B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_0_A = WTIMER0A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_0_B = WTIMER0B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_1_A = WTIMER1A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_1_B = WTIMER1B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_2_A = WTIMER2A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_2_B = WTIMER2B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_3_A = WTIMER3A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_3_B = WTIMER3B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_4_A = WTIMER4A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_4_B = WTIMER4B_IRQn,
  GPTM_EN_WIDE_TIMER_INT_5_A = WTIMER5A_IRQn,
  GPTM_EN_WIDE_TIMER_INT_5_B = WTIMER5B_IRQn,
  GPTM_EN_INVALID_TIMER_INT_NUMBER
} GPTM_enTimerInterruptNumber_t;
```

```c
/* GPTM Interrupt Action */
typedef void (*GPTM_vpfTimerInterruptAction_t) (void);

/* GPTM Linking Configurations Structure */
typedef struct
{
  GPTM_enTimerId_t               en_a_timerId;
  GPTM_enTimerMode_t             en_a_timerMode;
  GPTM_enTimerUse_t              en_a_timerUse;
  u64                            u64_a_timerStartValue;
  GPTM_enTimerInterruptMode_t    en_a_timerInterruptMode;
  GPTM_enTimerInterruptNumber_t  en_a_timerInterruptNumber;
} GPTM_stTimerLinkConfig_t;

/* GPTM Error States */
typedef enum
{
  GPTM_EN_NOK = 0,
  GPTM_EN_OK
} GPTM_enErrorState_t;

/* GPTM Time Units */
typedef enum
{
  GPTM_EN_TIME_US,
  GPTM_EN_TIME_MS,
  GPTM_EN_TIME_SEC,
  GPTM_EN_INVALID_TIME_UNIT
} GPTM_enTimeUnit_t;

| Name: GPTM_initialization
| Input: Pointer to Array of st TimerLinkConfig and u8 NumberOfTimers
| Output: en Error or No Error
| Description: Function to initialize GPTM peripheral using Linking Configurations.
|
GPTM_enErrorState_t GPTM_initialization (const GPTM_stTimerLinkConfig_t
*past_a_timersLinkConfig, u8 u8_a_numberOfTimers)

| Name: GPTM_setTimer
| Input: en TimerId, en TimerUse, u64 Time, and en TimeUnit
| Output: en Error or No Error
| Description: Function to set GPTM peripheral to count in microseconds(us),
|              milliseconds (ms), and seconds (sec).
|
GPTM_enErrorState_t GPTM_setTimer (GPTM_enTimerId_t en_a_timerId,
GPTM_enTimerUse_t en_a_timerUse, u64 u64_a_time, GPTM_enTimeUnit_t
en_a_timeUnit)
```

```
| Name: GPTM_enableTimer
| Input: en TimerId and en TimerUse
| Output: en Error or No Error
| Description: Function to enable GPTM.
|
GPTM_enErrorState_t GPTM_enableTimer (GPTM_enTimerId_t en_a_timerId,
GPTM_enTimerUse_t en_a_timerUse)

| Name: GPTM_disableTimer
| Input: en TimerId and en TimerUse
| Output: en Error or No Error
| Description: Function to disable GPTM.
|
GPTM_enErrorState_t GPTM_disableTimer (GPTM_enTimerId_t en_a_timerId,
GPTM_enTimerUse_t en_a_timerUse)

| Name: GPTM_enableInterrupt
| Input: en TimerInterruptNumber
| Output: en Error or No Error
| Description: Function to enable GPTM Interrupt.
|
GPTM_enErrorState_t GPTM_enableInterrupt (GPTM_enTimerInterruptNumber_t
en_a_timerInterruptNumber)

| Name: GPTM_disableInterrupt
| Input: en TimerInterruptNumber
| Output: en Error or No Error
| Description: Function to disable GPTM Interrupt.
|
GPTM_enErrorState_t GPTM_disableInterrupt (GPTM_enTimerInterruptNumber_t
en_a_timerInterruptNumber)

| Name: GPTM_getTimeoutStatusFlag
| Input: en TimerId, en TimerUse, and Pointer to u8 TimeoutStatusFlag
| Output: en Error or No Error
| Description: Function to get GPTM clear Timeout Status Flag.
|
GPTM_enErrorState_t GPTM_getTimeoutStatusFlag (GPTM_enTimerId_t en_a_timerId,
GPTM_enTimerUse_t en_a_timerUse, u8 *pu8_a_timeoutStatusFlag)

| Name: GPTM_clearTimeoutStatusFlag
| Input: en TimerId and en TimerUse,
| Output: en Error or No Error
| Description: Function to clear GPTM clear Timeout Status Flag.
|
GPTM_enErrorState_t GPTM_clearTimeoutStatusFlag (GPTM_enTimerId_t
en_a_timerId, GPTM_enTimerUse_t en_a_timerUse)
```

```
| Name: GPTM_setCallback
| Input: en TimerId, en TimerUse, and Pointer to Function that takes void and
|        returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function (in an Upper Layer) to
|              be called back in IRQ function of the passed Timer (TimerId),
|              the address is passed through a pointer to function
|              (TimerInterruptAction), and then pass this address to the IRQ function.
|
GPTM_enErrorState_t GPTM_setCallback (GPTM_enTimerId_t en_a_timerId,
GPTM_enTimerUse_t en_a_timerUse, void(*vpf_a_timerInterruptAction)(void))
```

2.4.3. HAL APIs

2.4.3.1. LED Driver APIs

```
Precompile and Linking Configurations:

/* LED IDs Counted from 0 to 7 */
#define LED_U8_0          0
#define LED_U8_1          1
#define LED_U8_2          2
#define LED_U8_3          3
#define LED_U8_4          4
#define LED_U8_5          5
#define LED_U8_6          6
#define LED_U8_7          7

/* LED Operations Counted from 0 to 2 */
#define LED_U8_ON         0
#define LED_U8_OFF        1
#define LED_U8_TOGGLE     2

/* LED Error States */
typedef enum
{
  LED_EN_NOK = 0,
  LED_EN_OK
} LED_enErrorState_t;

| Name: LED_initialization
| Input: u8 LedId
| Output: en Error or No Error
| Description: Function to initialize LED peripheral, by initializing GPIO peripheral.
|
LED_enErrorState_t LED_initialization (u8 u8_a_ledId)

| Name: LED_setLEDPin
| Input: u8 LedId and u8 Operation
| Output: en Error or No Error
| Description: Function to switch LED on, off, or toggle.
|
LED_enErrorState_t LED_setLEDPin( u8 u8_a_ledId, u8 u8_a_operation )
```

2.4.3.2. BTN Driver APIs

Precompile and Linking Configurations Snippet:

```
/* BTN IDs Counted from 0 to 7 */
#define BTN_U8_0          0
#define BTN_U8_1          1
#define BTN_U8_2          2
#define BTN_U8_3          3
#define BTN_U8_4          4
#define BTN_U8_5          5
#define BTN_U8_6          6
#define BTN_U8_7          7

/* BTN Values Counted from 0 to 1 */
#define BTN_U8_LOW        0
#define BTN_U8_HIGH       1

/* BTN Error States */
typedef enum
{
  BTN_EN_NOK = 0,
  BTN_EN_OK
} BTN_enErrorState_t;

| Name: BTN_initialization
| Input: u8 BTNId
| Output: en Error or No Error
| Description: Function to initialize BTN peripheral, by initializing GPIO peripheral.
|
BTN_enErrorState_t BTN_initialization (u8 u8_a_btnId)

| Name: BTN_getBTNState
| Input: u8 BTNId and Pointer to u8 ReturnedBTNState
| Output: en Error or No Error
| Description: Function to get BTN state.
|
BTN_enErrorState_t BTN_getBTNState (u8 u8_a_btnId, u8 *pu8_a_returnedBTNState)
```

2.4.3.3. PWM Driver APIs

```
Precompile and LinKing Configurations:

/* PWM IDs Counted from 0 to 7 */
#define PWM_U8_0              0
#define PWM_U8_1              1
#define PWM_U8_2              2
#define PWM_U8_3              3
#define PWM_U8_4              4
#define PWM_U8_5              5
#define PWM_U8_6              6
#define PWM_U8_7              7

/* PWM Port Id Select */
/* Options: GPIO_EN_PORTA
            GPIO_EN_PORTB
            GPIO_EN_PORTC
            GPIO_EN_PORTD
            GPIO_EN_PORTE
            GPIO_EN_PORTF
 */
#define PWM_U8_PORT_ID_SELECT         GPIO_EN_PORTF

/* PWM Port Bus Id Select */
/* Options: GPIO_EN_APB
            GPIO_EN_AHB
 */
#define PWM_U8_PORT_BUS_ID_SELECT     GPIO_EN_APB

/* PWM Pin Id Select */
/* Options: GPIO_EN_PIN0
            GPIO_EN_PIN1
            GPIO_EN_PIN2
            GPIO_EN_PIN3
            GPIO_EN_PIN4
            GPIO_EN_PIN5
            GPIO_EN_PIN6
            GPIO_EN_PIN7
 */
#define PWM_U8_PIN_ID_SELECT          GPIO_EN_PIN3

/* PWM Timer Id Select */
/* Options: GPTM_EN_TIMER_0
            GPTM_EN_TIMER_1
            GPTM_EN_TIMER_2
            GPTM_EN_TIMER_3
            GPTM_EN_TIMER_4
            GPTM_EN_TIMER_5
```

```
            GPTM_EN_WIDE_TIMER_0
            GPTM_EN_WIDE_TIMER_1
            GPTM_EN_WIDE_TIMER_2
            GPTM_EN_WIDE_TIMER_3
            GPTM_EN_WIDE_TIMER_4
            GPTM_EN_WIDE_TIMER_5
 */
#define PWM_U8_TIMER_ID_SELECT        GPTM_EN_TIMER_0

/* PWM Timer Use Select */
/* Options: GPTM_EN_INDIVIDUAL_A
            GPTM_EN_INDIVIDUAL_B
            GPTM_EN_CONCATUNATED
 */
#define PWM_U8_TIMER_USE_SELECT       GPTM_EN_CONCATUNATED

/* PWM Periods Flags */
#define PWM_OFF_PERIOD_FLAG           0
#define PWM_ON_PERIOD_FLAG            1

/* PWM Maximum Duty Cycle */
#define PWM_U8_MAX_DUTY_CYCLE         100

/* PWM Time Units */
typedef enum
{
  PWM_EN_TIME_US,
  PWM_EN_TIME_MS,
  PWM_EN_TIME_SEC,
  PWM_EN_INVALID_TIME_UNIT
} PWM_enTimeUnit_t;

/* PWM Error States */
typedef enum
{
  PWM_EN_NOK = 0,
  PWM_EN_OK
} PWM_enErrorState_t;

| Name: PWM_initialization
| Input: u8 PWMId
| Output: en Error or No Error
| Description: Function to initialize PWM peripheral, by initializing GPIO and
|              GPTM peripherals.
|
PWM_enErrorState_t PWM_initialization (u8 u8_a_pwmId)
```

```
| Name: PWM_enable
| Input: u8 DutyCycle
| Output: en Error or No Error
| Description: Function to enable PWM.
|
PWM_enErrorState_t PWM_enable (u64 u64_a_totalPeriod, PWM_enTimeUnit_t
en_a_timeUnit, u8 u8_a_dutyCycle)

| Name: PWM_disable
| Input: void
| Output: void
| Description: Function to disable PWM.
|
void PWM_disable (void)
```

2.4.4. APP APIs

Precompile and Linking Configurations:

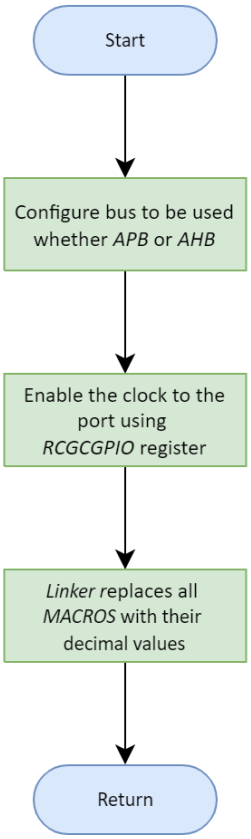
```
/* APP States */
#define APP_U8_STATE_0        0
#define APP_U8_STATE_1        1
#define APP_U8_STATE_2        2
#define APP_U8_STATE_3        3
#define APP_U8_STATE_4        4

/* APP PWM Total Period */
#define APP_U16_PWM_TOTAL_PERIOD            500

/* APP PWM Duty Cycles */
#define APP_U8_PWM_30_PERCENT_DUTY_CYCLE    30
#define APP_U8_PWM_60_PERCENT_DUTY_CYCLE    60
#define APP_U8_PWM_90_PERCENT_DUTY_CYCLE    90

/* APP PWM Total Period Divider */
#define APP_F32_PWM_TOTAL_PERIOD_DIVIDER    0.02f

| Name: APP_initialization
| Input: void
| Output: void
| Description: Function to Initialize the Application.
|
void APP_initialization (void)

| Name: APP_startProgram
| Input: void
| Output: void
| Description: Function to Run the basic flow of the Application.
|
void APP_startProgram (void)
```

3. Low Level Design

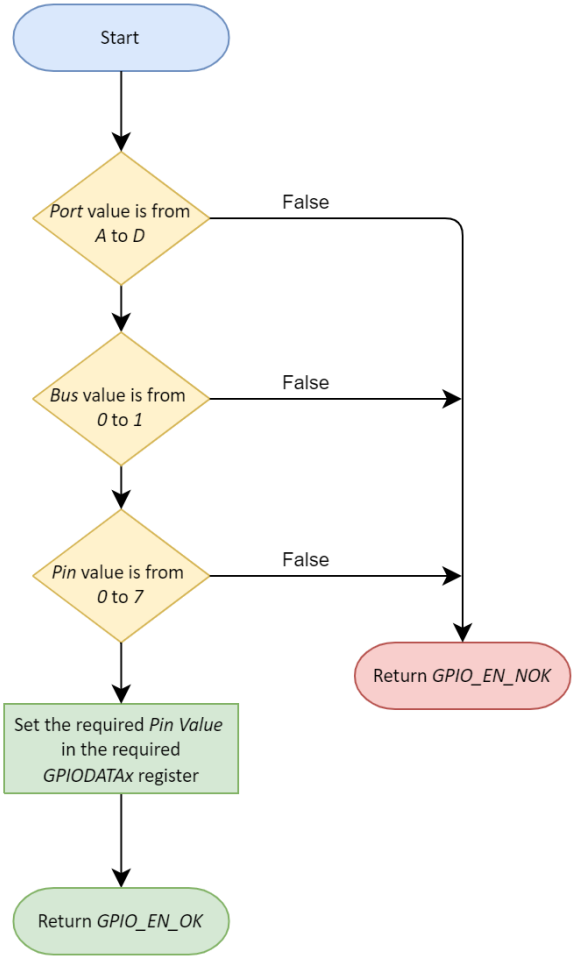
3.1. MCAL Layer

3.1.1. GPIO Module

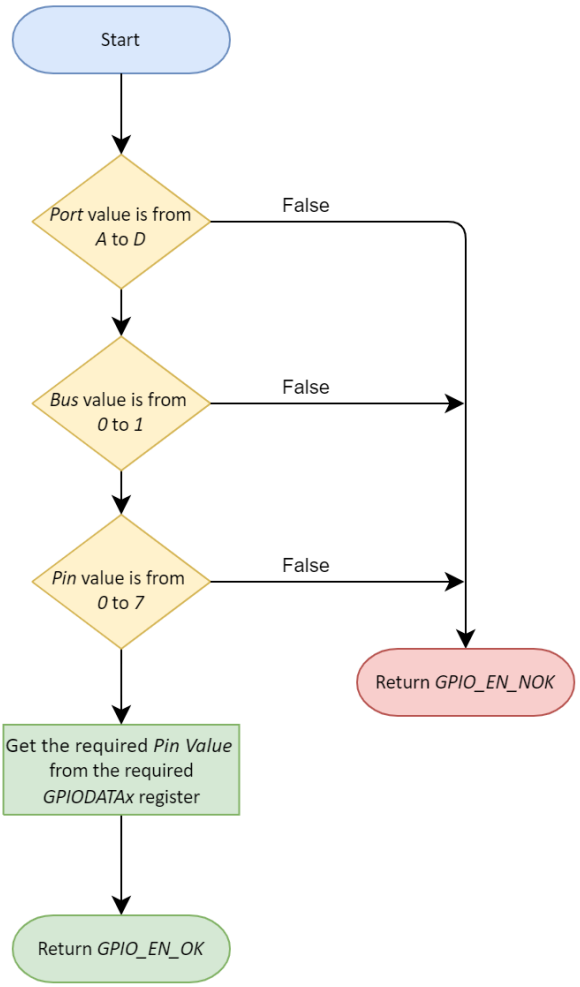
A. GPIO_initialization



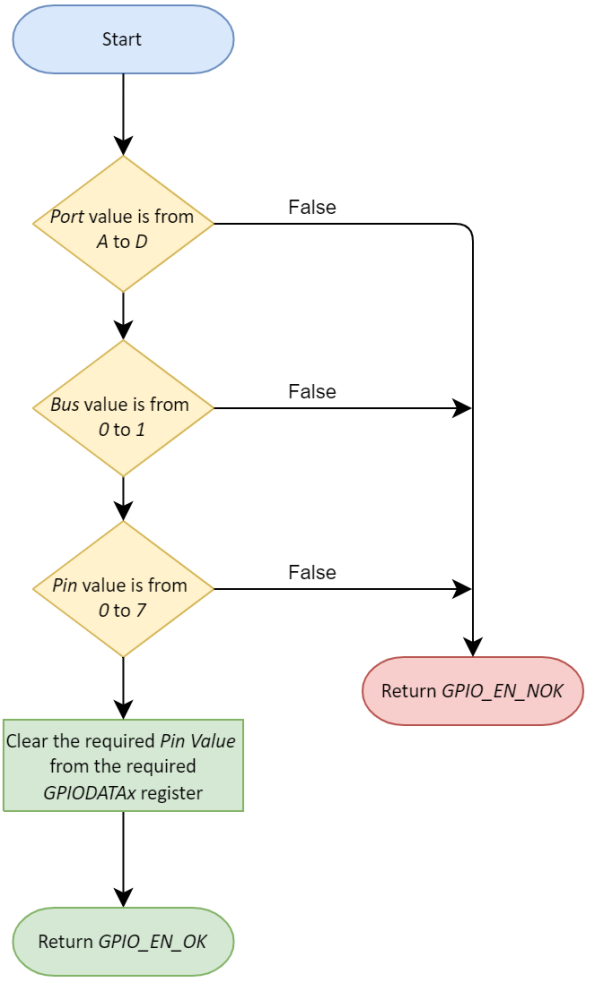
B. GPIO_setPinValue



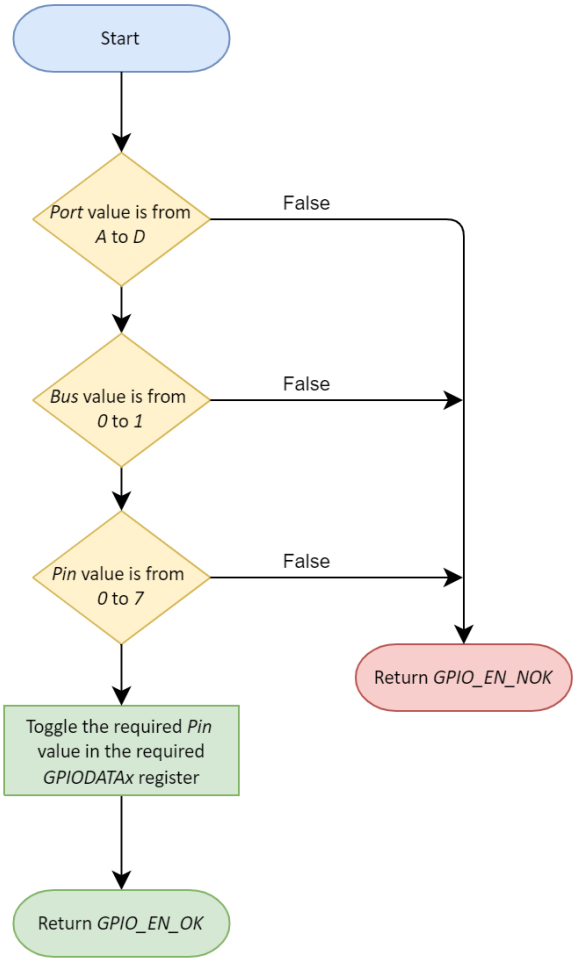
C. *GPIO_getPinValue*



D. GPIO_clearPinValue

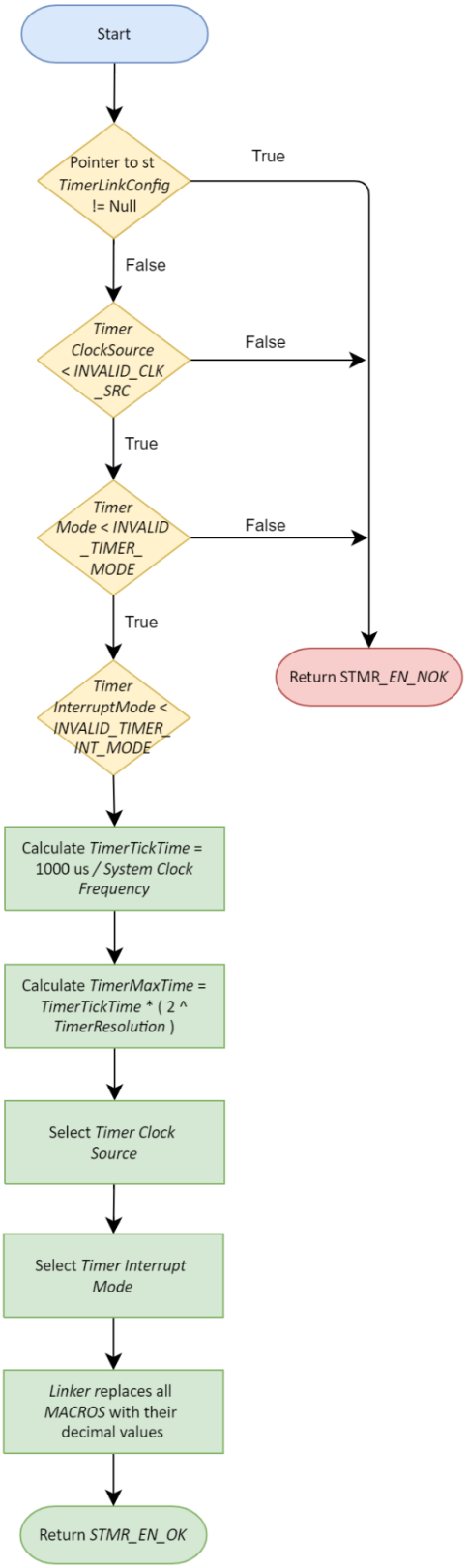


E. *GPIO_togglePinValue*

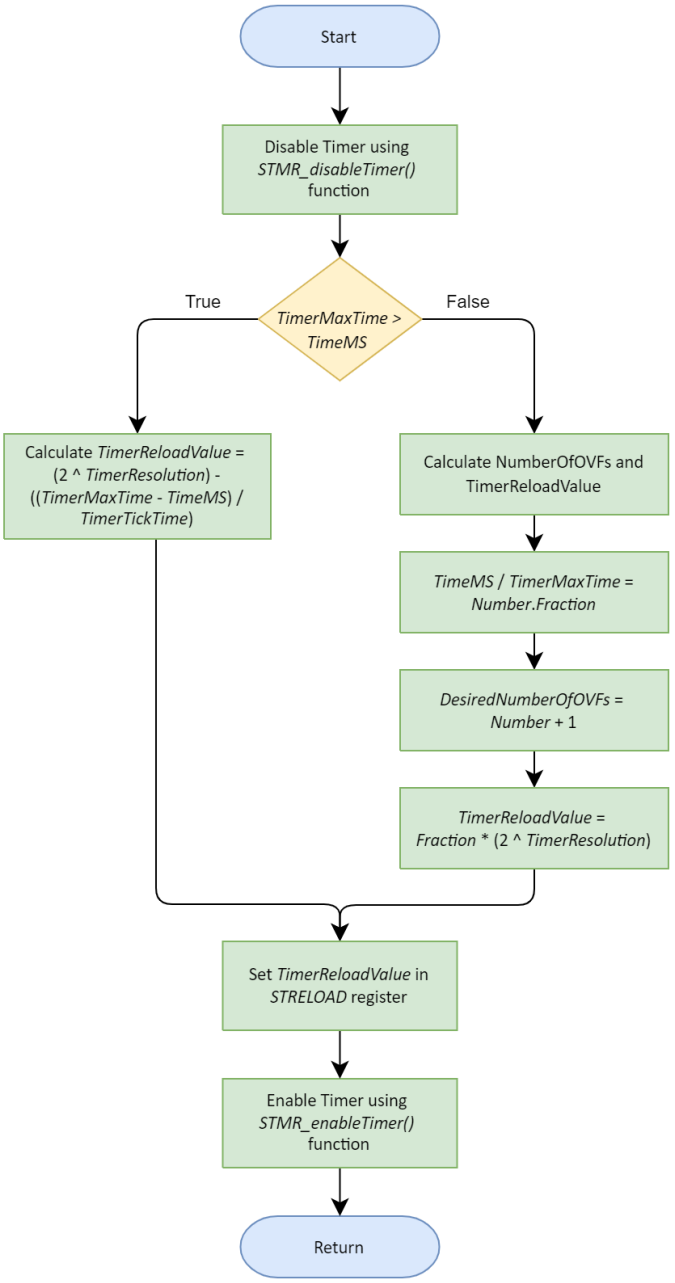


3.1.2. GPTM Module

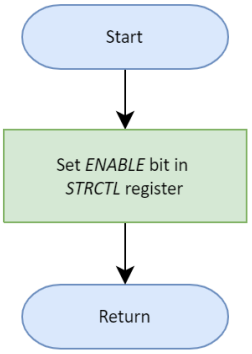
A. STMR_initialization



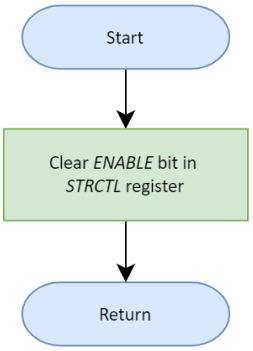
B. STMR_setTimerMS



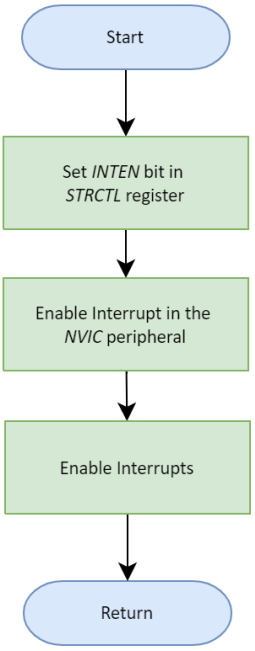
C. STMR_enableTimer



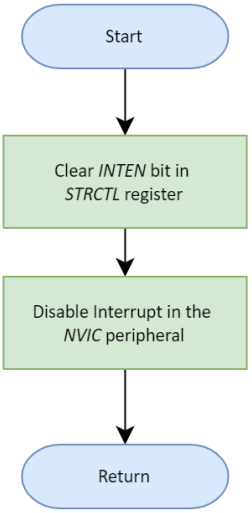
D. STMR_disableTimer



E. STMR_enableInterrupt



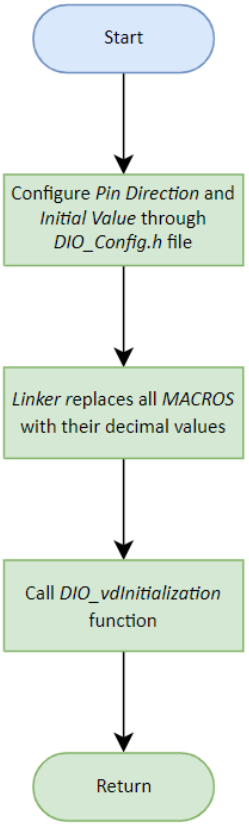
F. *STMR_disableInterrupt*



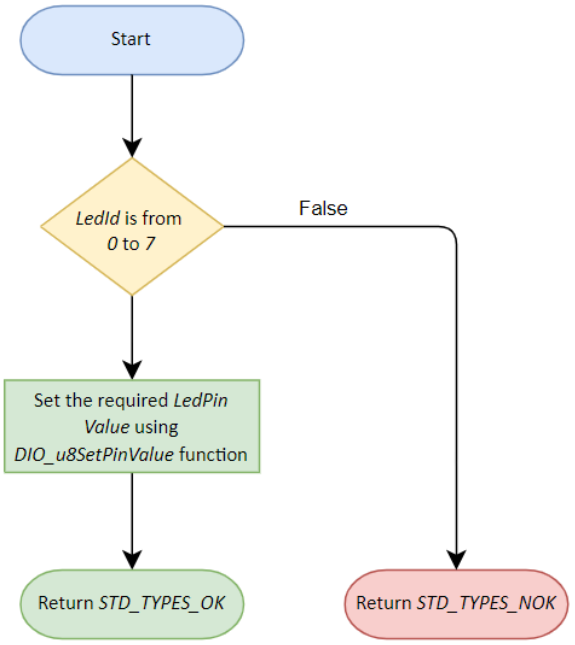
3.2. HAL Layer

3.2.1. LED Module

A. LED_initiailization

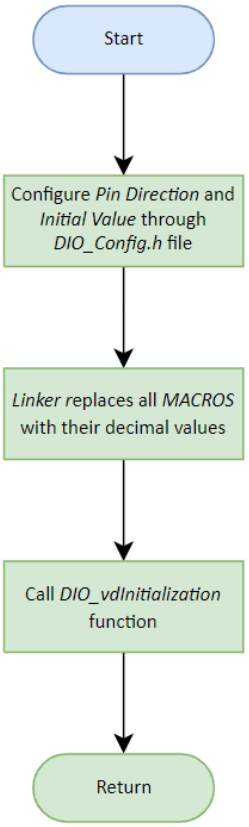


B. LED_setLEDPin

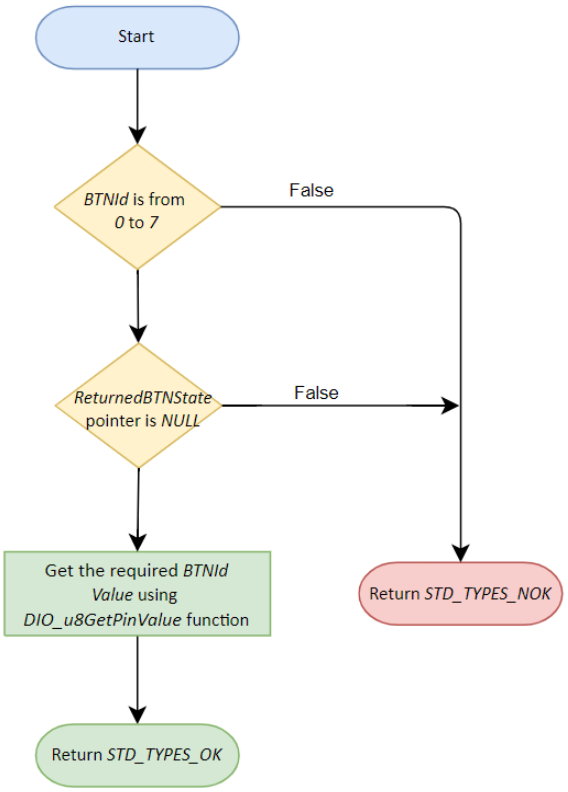


3.2.2. BTN Module

A. BTN_initiailization



B. BTN_getBTNState



3.2.3. PWM Module

3.3. APP Layer

4. References

1. *Draw IO*
2. *Layered Architecture | Baeldung on Computer Science*
3. *Microcontroller Abstraction Layer (MCAL) | Renesas*
4. *Hardware Abstraction Layer - an overview | ScienceDirect Topics*
5. *What is a module in software, hardware and programming?*
6. *Embedded Basics – API's vs HAL's*
7. *Using Push Button Switch with Atmega32 and Atmel Studio*