2023

# LED Sequence V1.0

HOSSAM ELWAHSH
EMBEDDED SYSTEMS - LEVEL 1

# Table of Contents

System Requirements Specifications ..... 2
Brief ..... 2
Hardware Requirements ..... 2
Software Requirements ..... 2
System Design ..... 3
State Machine Diagram ..... 3
Layered Architecture ..... 4
Project Modules APIs ..... 5
DIO Driver ..... 5
EXI (External Interrupt) Driver ..... 6
LED Driver ..... 7
Button Driver ..... 8
Application ..... 9
Project Tree ..... 9
Project Modules APIs Charts ..... 10
EXI Flowcharts ..... 10
Button API Flowcharts ..... 11
LED API Flowcharts ..... 12
DIO API Flowcharts ..... 13
Application API Flowcharts ..... 14

# LED Sequence V1.0

## System Requirements Specifications

### Brief

Develop a system that controls 4 LEDs lighting sequence according to button pressing.

### Hardware Requirements

- Four LEDs (LED0, LED1, LED2, LED3)
- One button (BUTTON0)

### Software Requirements

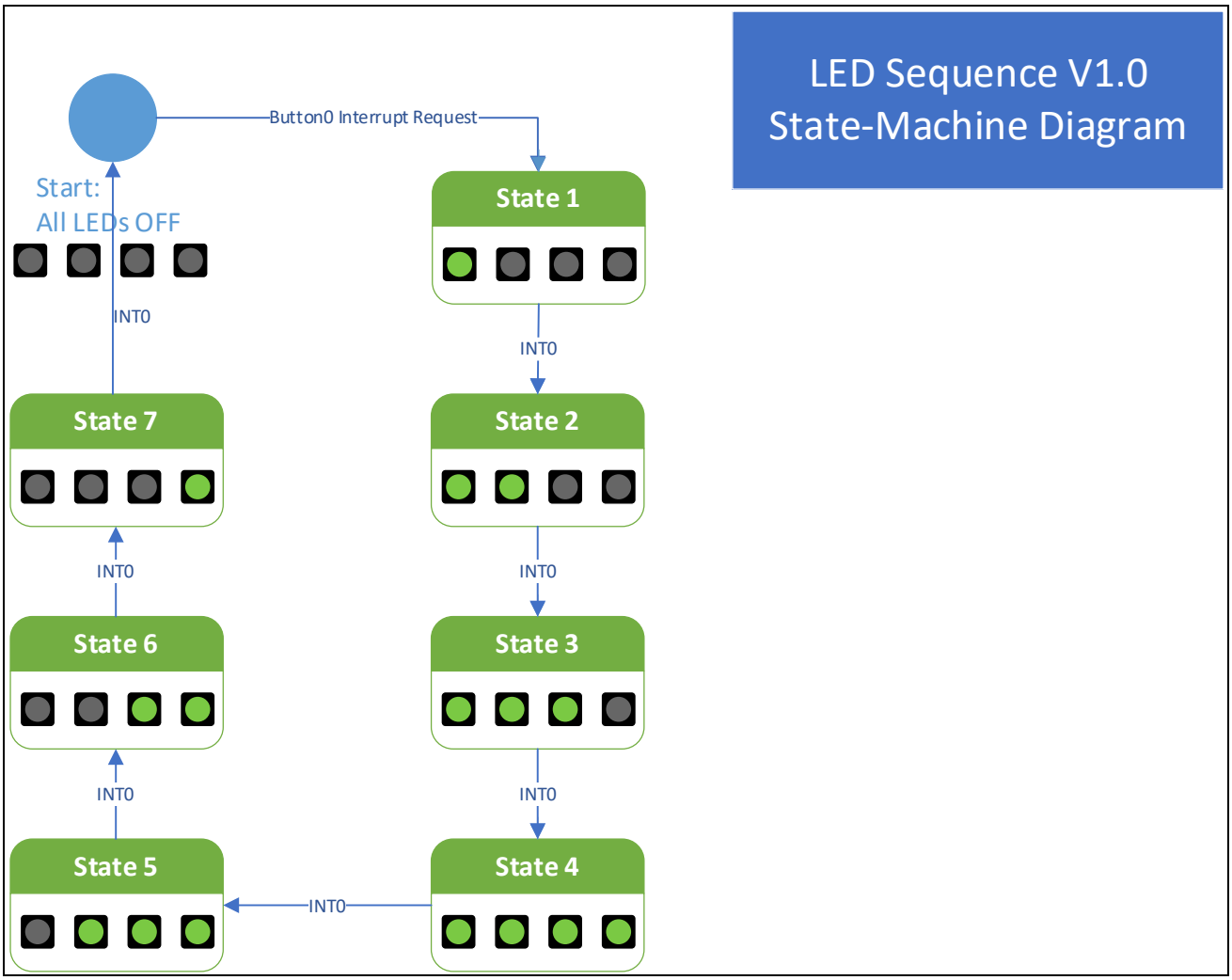
Initially, all LEDs are OFF

- Once **BUTTON0** is pressed, LED0 will be ON
- Each press further will make another LED is ON
- At the fifth press, **LED0** will changed to be OFF
- Each press further will make only one LED is OFF
- ***The following will be repeated forever***
- The sequence is described below
- Initially (OFF, OFF, OFF, OFF)
- Press 1 (ON, OFF, OFF, OFF)
- Press 2 (ON, ON, OFF, OFF)
- Press 3 (ON, ON, ON, OFF)
- Press 4 (ON, ON, ON, ON)
- Press 5 (OFF, ON, ON, ON)
- Press 6 (OFF, OFF, ON, ON)
- Press 7 (OFF, OFF, OFF, ON)
- Press 8 (OFF, OFF, OFF, OFF)
- Press 9 (ON, OFF, OFF, OFF)

# System Design

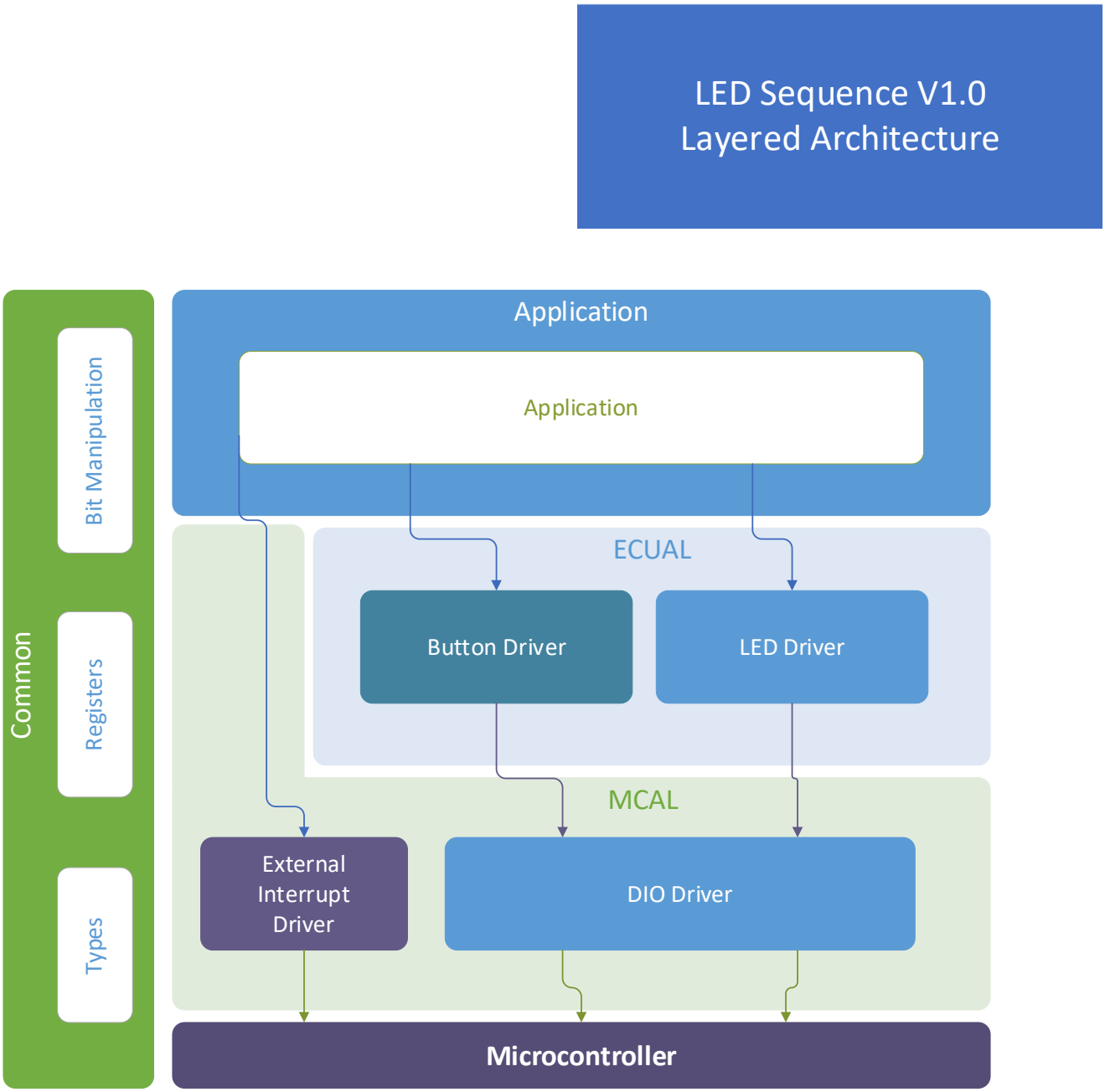
## State Machine Diagram

**Software used:** Microsoft Visio



# Layered Architecture

**Software used:** Microsoft Visio



# Project Modules APIs

## DIO Driver

### *DIO Macros/Enums:*

| Type | Name | Values | Desc |
|---|---|---|---|
| `#define` | LOW<br>HIGH | LOW = 0<br>HIGH = 1 | **Macro for digital levels** |
| `typedef enum` | EN_DIO_PORT_T | • A, B, C, D | **Defines available DIO ports** |
| `typedef enum` | EN_DIO_DIRECTION_T | • In = 0<br>• Out = 1 | **Defines DIO pin direction** |
| `typedef enum` | EN_DIO_Error_T | • DIO_OK<br>• DIO_Error | **Defines DIO return error** |

### *DIO Functions:*

```
/**
 * Configures pin at given portNumber as input/output
 * @param pinNumber [in] pin number
 * @param portNumber [in] Port to configure
 * @param direction [in] direction for pin enum (IN, OUT)
 */
EN_DIO_Error_T DIO_init(uint8_t pinNumber, EN_DIO_PORT_T portNumber, EN_DIO_DIRECTION_T
direction);
```

```
/**
 * Writes pin value for the given port/pin
 * @param pinNumber [in] pin number
 * @param portNumber [in] Port to use
 * @param value [in] value to write
 */
EN_DIO_Error_T DIO_write(uint8_t pinNumber, EN_DIO_PORT_T portNumber, uint8_t value);
```

```
/**
 * Toggles pin value for the given port/pin
 * @param pinNumber [in] pin number
 * @param portNumber [in] Port to use
 */
EN_DIO_Error_T DIO_toggle(uint8_t pinNumber, EN_DIO_PORT_T portNumber);
```

```
/**
 * Reads pin value for the given port/pin and stores it in *value
 * @param pinNumber [in] pin number
 * @param portNumber [in] Port to use
 * @param *value [out] pointer to output pin value into
 */
EN_DIO_Error_T DIO_read(uint8_t pinNumber, EN_DIO_PORT_T portNumber, uint8_t *value);
```

## EXI (External Interrupt) Driver

### *EXI Macros/Enums:*

| Type | Name/Value | Desc |
|---|---|---|
| `#define` | `EXT_INT_0 __vector_1` | **Interrupt vector naming** |
| `#define` | `EXT_INT_1 __vector_2` | **Interrupt vector naming** |
| `#define` | `EXT_INT_2 __vector_3` | **Interrupt vector naming** |
| `#define` | `sei() __asm__ __volatile__ ("sei" ::: "memory")` | **Enables global interrupt** |
| `#define` | `cli() __asm__ __volatile__ ("cli" ::: "memory")` | **Disables global interrupt** |
| `#define` | `ISR(INT_VECT) void INT_VECT(void) __attribute__ ((signal,used));\`<br><br>`void INT_VECT(void)` | **ISR definition** |
| `typedef enum` | `typedef enum EN_EXI_INT_t {`<br>`  INT0, INT1`<br>`} EN_EXI_INT_t;` | **Defines Interrupt port names** |
| `typedef enum` | `typedef enum EN_EXI_SENSE_t {`<br>`  // Interrupts on low level`<br>`  LOW_LEVEL = 0xFC,`<br>`  // Interrupts on any logical change`<br>`  ANY_LEVEL = 0x01,`<br>`  // Interrupts on Falling edge`<br>`  FALLING_EDGE = 0x02,`<br>`  // Interrupts on Rising edge`<br>`  RISING_EDGE = 0x03`<br>`} EN_EXI_SENSE_t;` | **Enum for ATmega32 interrupt sense modes** |
| `typedef enum` | `typedef enum EN_EXI_ERROR_t {`<br>`  EXI_OK,`<br>`  EXI_ERROR`<br>`} EN_EXI_ERROR_t;` | **Error return type for EXI API** |

### *EXI Functions:*

```
/**
 * Sets and enables an external interrupt pin with given mode
 * @param interrupt [in] Interrupt number (INT0, INT1)
 * @param interruptSenseMode [in] sense mode enum
 */
EN_EXI_ERROR_t EXI_enableInterrupt(EN_EXI_INT_t interrupt, EN_EXI_SENSE_t
interruptSenseMode);
```

```
/**
 * Disables a given interrupt pin
 * @param interrupt [in] enum (INT0, INT1)
 */
EN_EXI_ERROR_t EXI_disableInterrupt(EN_EXI_INT_t interrupt);
```

```
/**
* Disables global interrupts
*  sets I-(7th) bit in SREG to 0
*/
void EXI_disableAll(void); // no return needed
```

## LED Driver

### *LED Macros/Enums:*

| Type | Name/Value | Desc |
|---|---|---|
| **typedef enum** | typedef enum EN_LED_ERROR_t<br>{<br>LED_OK,<br>LED_ERROR<br>}EN_LED_ERROR_t; | **Enum for LED error return** |

### *LED Functions:*

```
/**
 * Initializes LED on given port & pin
 * @param ledPort [in] LED Port
 * @param ledPin [in] LED Pin number in ledPort
 */
EN_LED_ERROR_t LED_init(EN_DIO_PORT_T ledPort, uint8_t ledPin);
```

```
/**
 * Turns on LED at given port/pin
 * @param ledPort [in] LED Port
 * @param ledPin [in] LED Pin number in ledPort
 */
EN_LED_ERROR_t LED_on(EN_DIO_PORT_T ledPort, uint8_t ledPin);
```

```
/**
 * Turns off LED at given port/pin
 * @param ledPort [in] LED Port
 * @param ledPin [in] LED Pin number in ledPort
 */
EN_LED_ERROR_t LED_off(EN_DIO_PORT_T ledPort, uint8_t ledPin);
```

```
/**
 * Toggles LED at given port/pin
 * @param ledPort [in] LED Port
 * @param ledPin [in] LED Pin number in ledPort
 */
EN_LED_ERROR_t LED_toggle(EN_DIO_PORT_T ledPort, uint8_t ledPin);
```

## Button Driver

### *Button Macros/Enums:*

| Type | Name/Value | Desc |
|---|---|---|
| **typedef enum** | typedef enum EN_ButtonError_t<br>{<br>BUTTON_OK,<br>BUTTON_ERROR<br>}EN_ButtonError_t; | **Button Error Types** |

### *Button Functions:*

```
/**
 * Initializes port and pin as button
 * @param buttonPort [in] Port to use
 * @param buttonPin [in] Pin number in port
 */
EN_ButtonError_t BUTTON_init(EN_DIO_PORT_T buttonPort, uint8_t buttonPin);
```

```
// Read Button State
/**
 * Reads button state and stores value in buttonState
 * @param buttonPort [in] Port to use
 * @param buttonPin [in] Pin number in port
 * @param buttonState [out] Store Button State (1:High / 0:Low)
 */
EN_ButtonError_t BUTTON_read(EN_DIO_PORT_T buttonPort, uint8_t buttonPin, uint8_t
* buttonState);
```

## Application

*Application Includes:*

```
#include "../ECUAL/LED Driver/led.h"
#include "../ECUAL/Button Driver/button.h"
#include "../MCAL/EXI Driver/interrupts.h"
```

*Application Functions:*

```
/// Application initialization
void App_init();
```

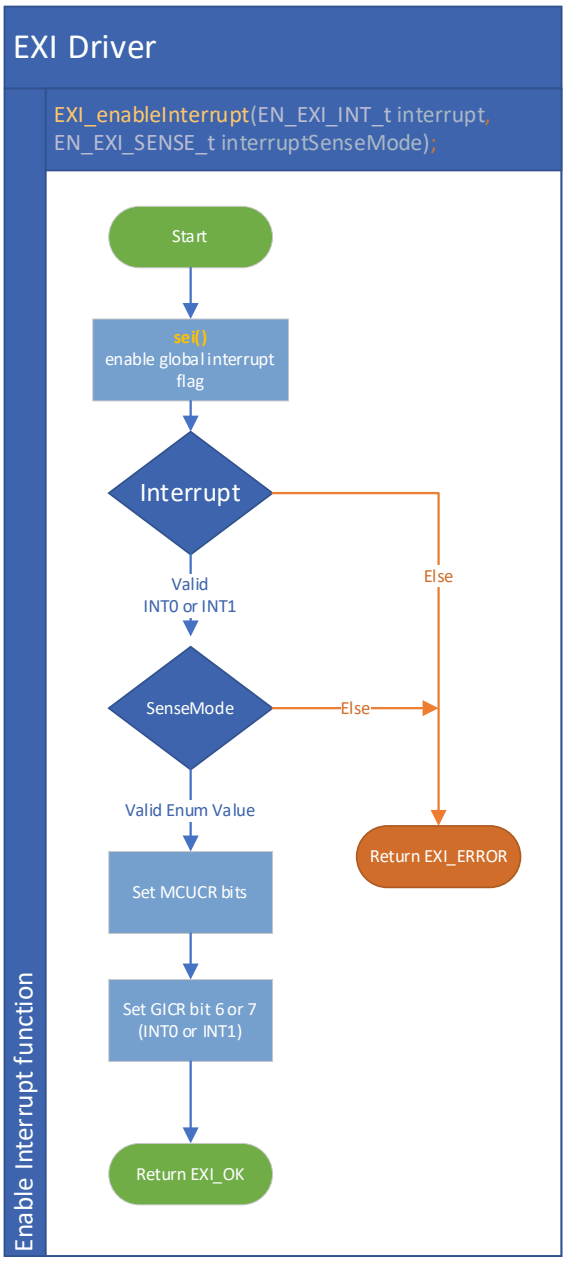
```
/// Start Application routine
void App_Start();
```

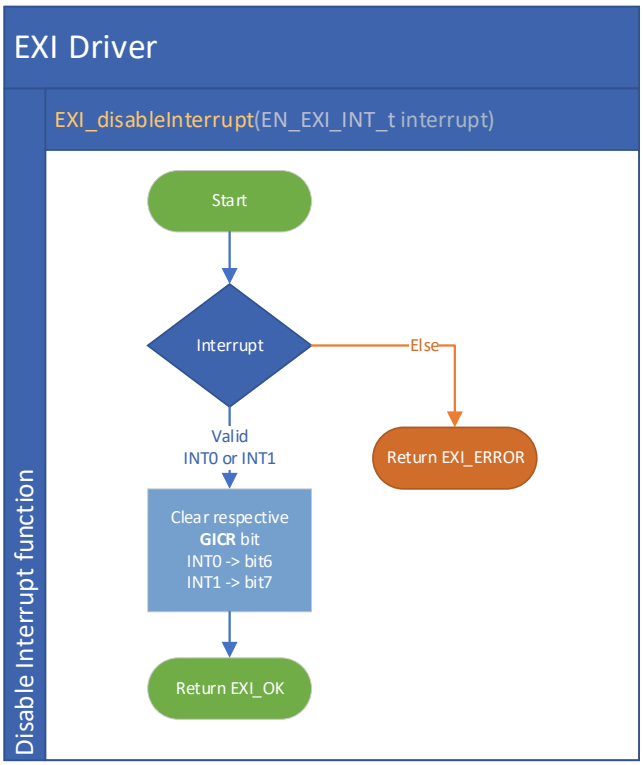
## Project Tree

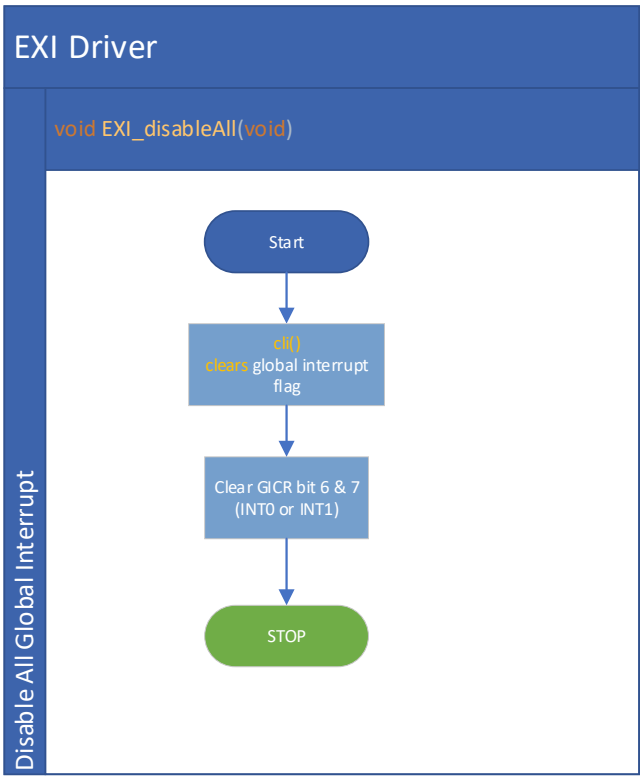
```
D:.
    .gitignore
    main.c
    main.h
    README.md

----Application
        application.c
        application.h

----Common
        bit_manipulation.h
        types.h

----Docs
        *.vsdx
        LED Sequence V1.0.pdf

----ECUAL
    ----Button Driver
            button.c
            button.h

    ----LED Driver
            led.c
            led.h

----MCAL
    |   registers.h
    |
    ----DIO Driver
    |       dio.c
    |       dio.h
    |
    ----EXI Driver
            interrupts.c
            interrupts.h

----Proteus
        Proteus_LED_Sequence_V1.0.pdsprj
```

# Project Modules APIs Charts

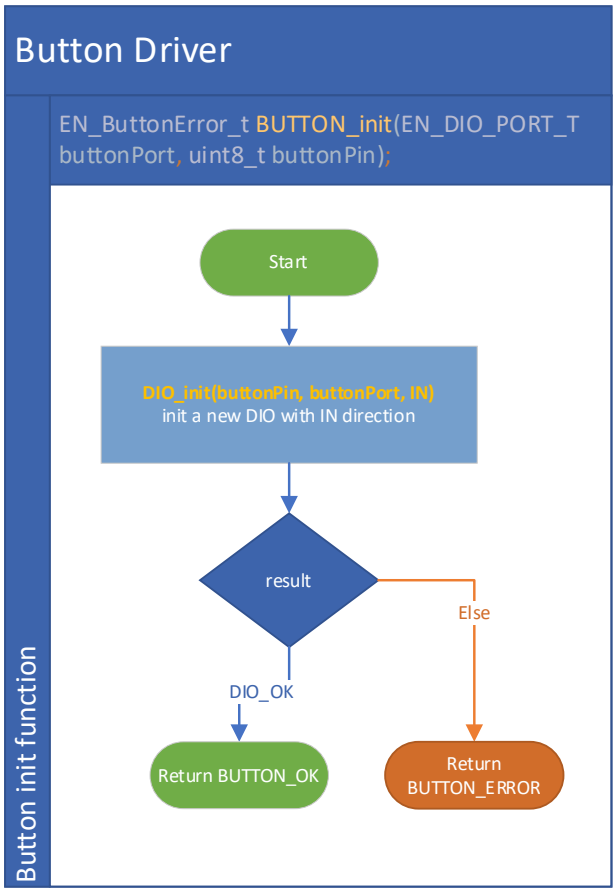
## EXI Flowcharts

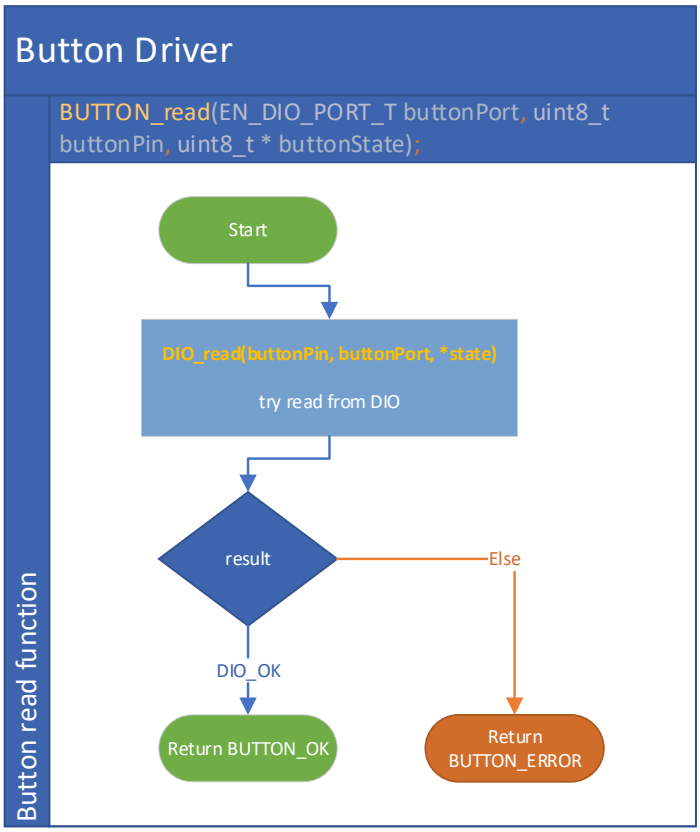


## Button API Flowcharts

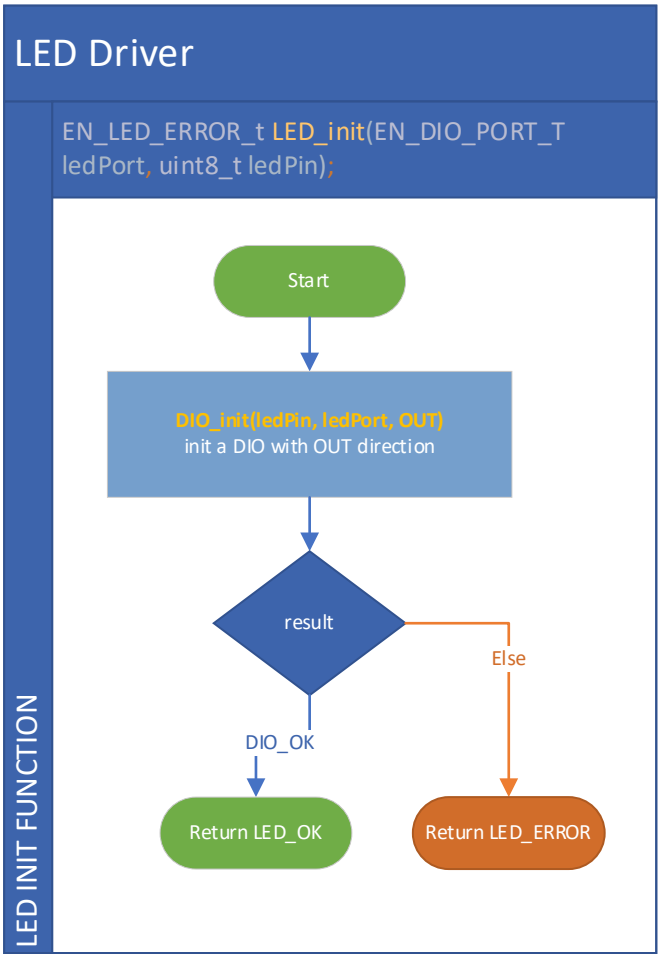


## LED API Flowcharts

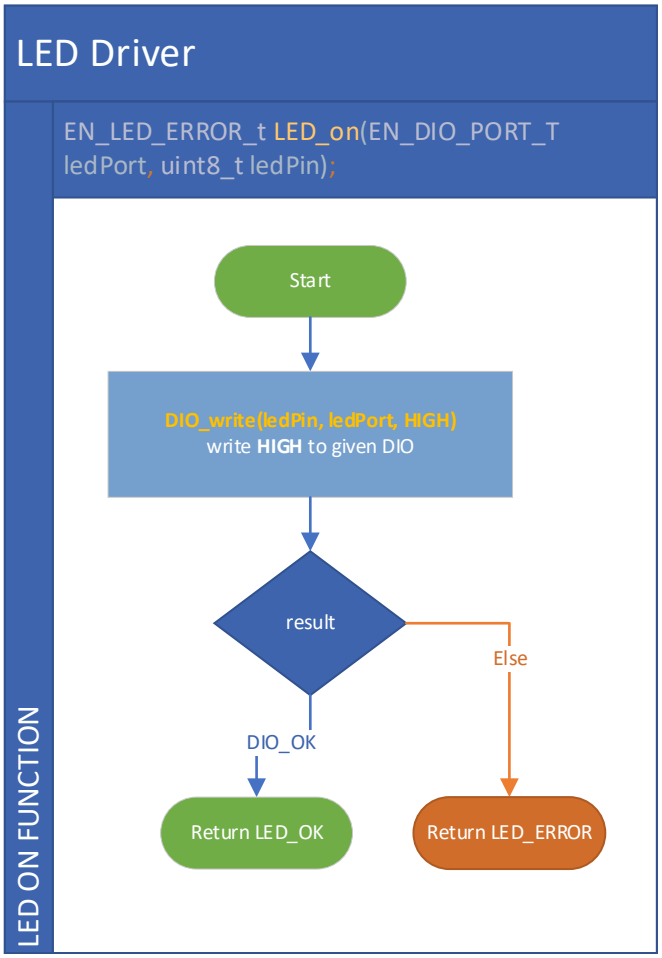
### LED Driver

EN_LED_ERROR_t LED_init(EN_DIO_PORT_T ledPort, uint8_t ledPin);



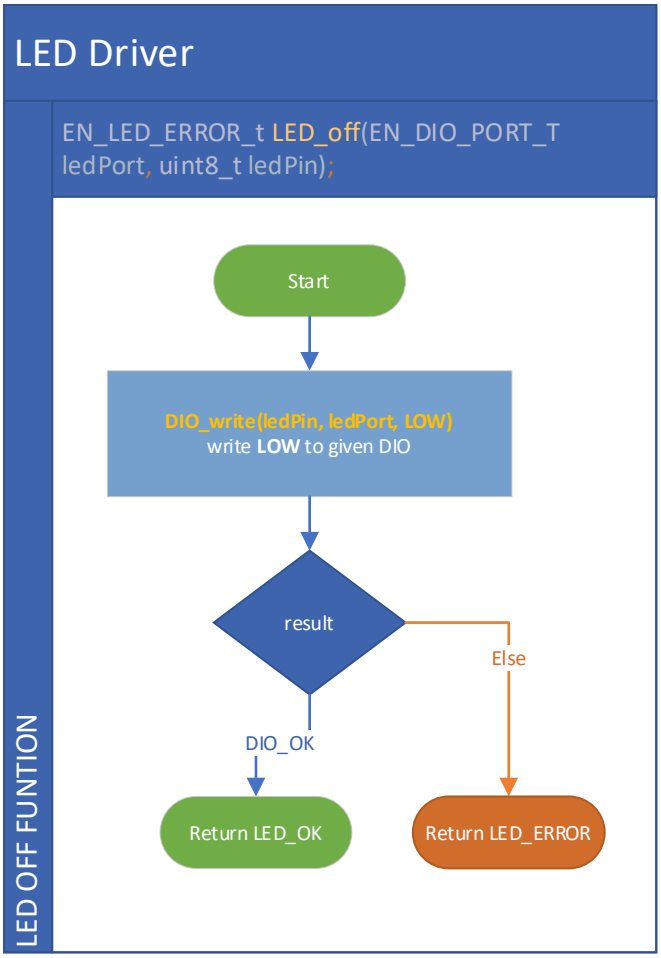

### LED Driver

EN_LED_ERROR_t LED_on(EN_DIO_PORT_T ledPort, uint8_t ledPin);



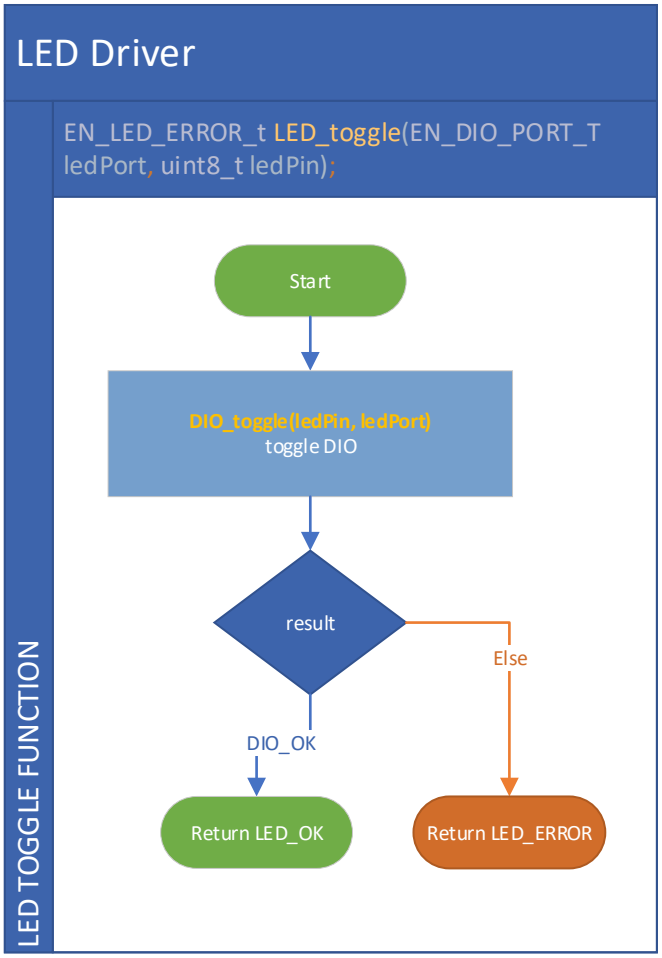

### LED Driver

EN_LED_ERROR_t LED_off(EN_DIO_PORT_T ledPort, uint8_t ledPin);




### LED Driver

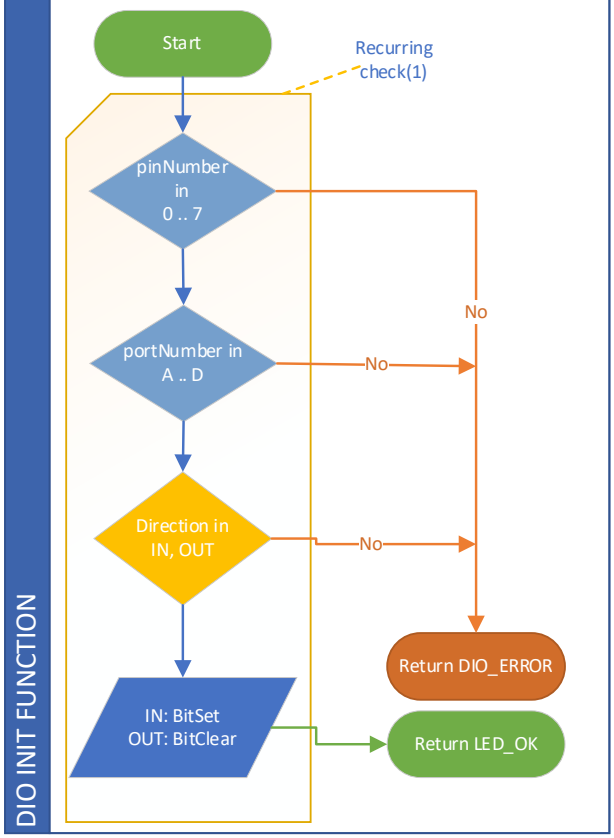
EN_LED_ERROR_t LED_toggle(EN_DIO_PORT_T ledPort, uint8_t ledPin);



# DIO API Flowcharts

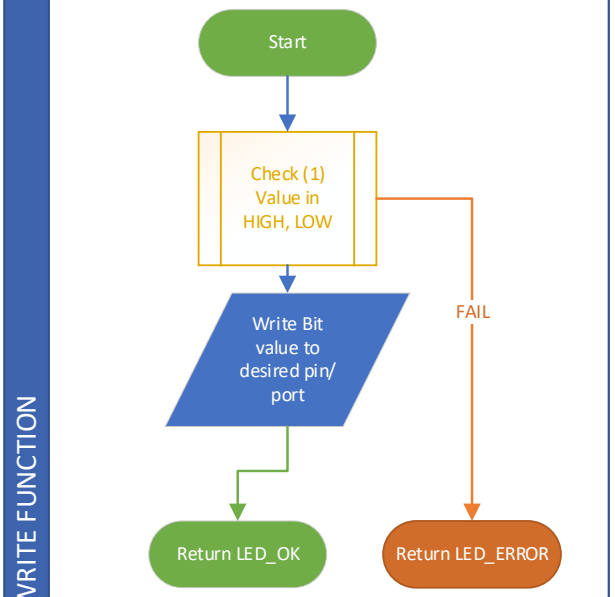
## DIO Driver

EN_DIO_Error_T DIO_init(uint8_t pinNumber,
EN_DIO_PORT_T portNumber,
EN_DIO_DIRECTION_T direction);



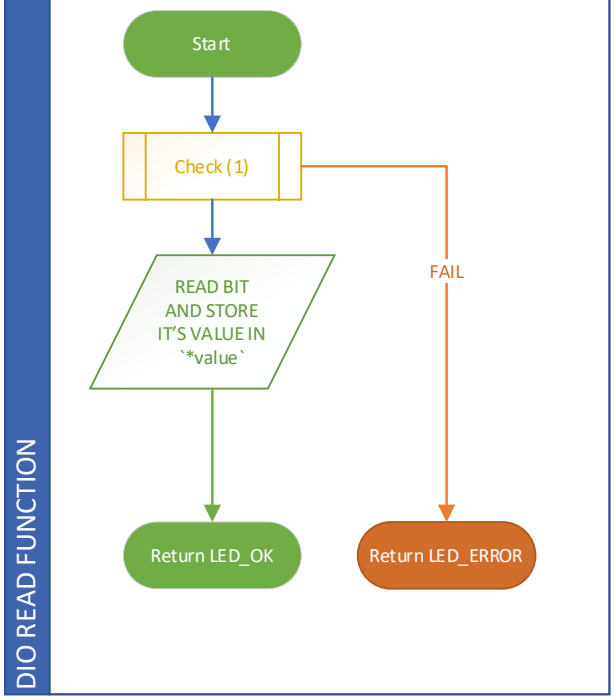

## DIO Driver

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



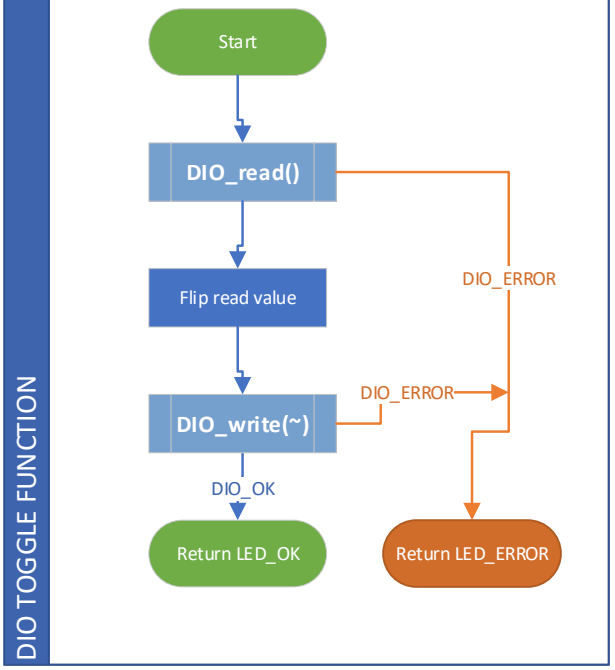

## DIO Driver

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




## DIO Driver

EN_DIO_Error_T DIO_toggle(uint8_t pinNumber,
EN_DIO_PORT_T portNumber);



# Application API Flowcharts

## Application

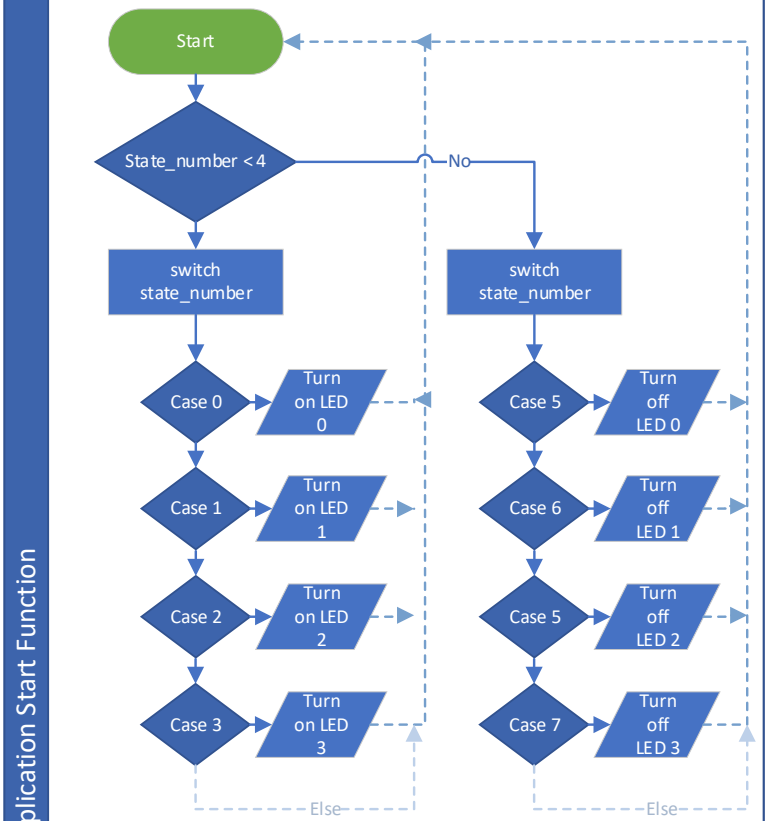
Globals

```
/* LEDs */
#define LED_0_PORT C
#define LED_0_PIN 0
#define LED_1_PORT C
#define LED_1_PIN 1
#define LED_2_PORT C
#define LED_2_PIN 2
#define LED_3_PORT C
#define LED_3_PIN 3

/* Buttons */
#define BUTTON_0_port D
#define BUTTON_0_PIN 3

/* Magic Numbers */
#define NUMBER_OF_LED_STATES 7

/// Global Variables
uint8_t state_number = 7;
```

Globals

## Application

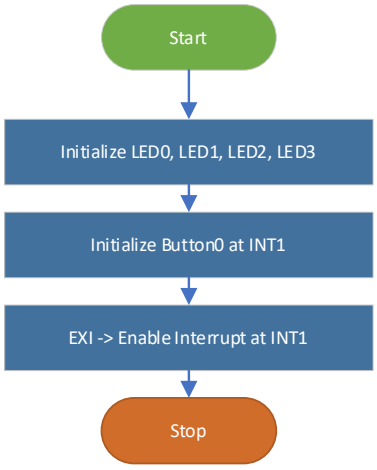
void App_Start();




Application Start Function

## Application

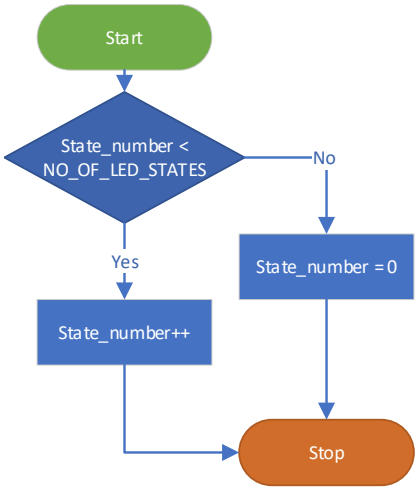
void App_init();




Application init function

## Application

ISR(EXT_INT_1)




ISR function for INT1