2023

LED Sequence V2.0

HOSSAM ELWAHSH
EMBEDDED SYSTEMS - LEVEL 1

Table of Contents

LED Sequence V2.0

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

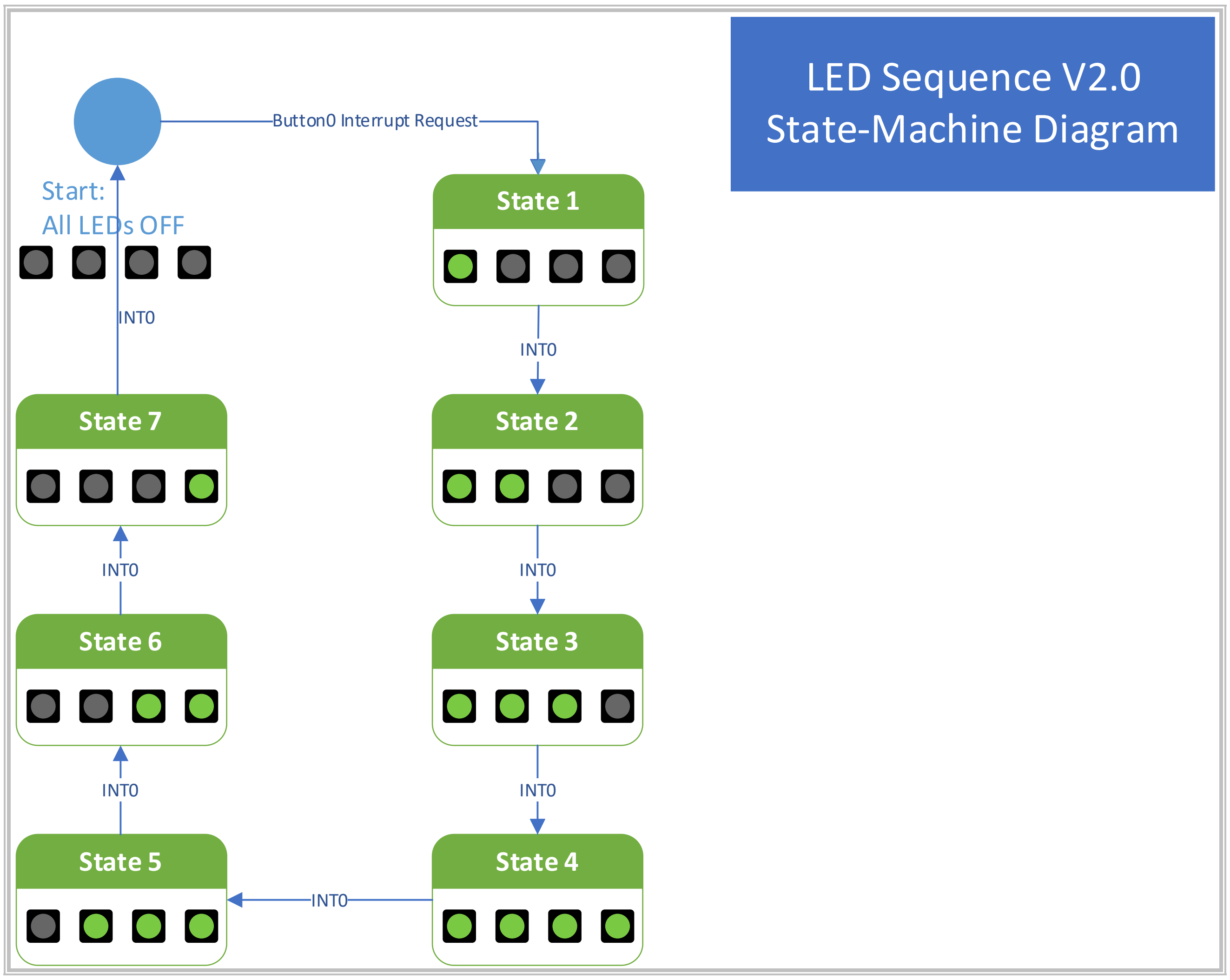
Initially, all LEDs are OFF

- 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

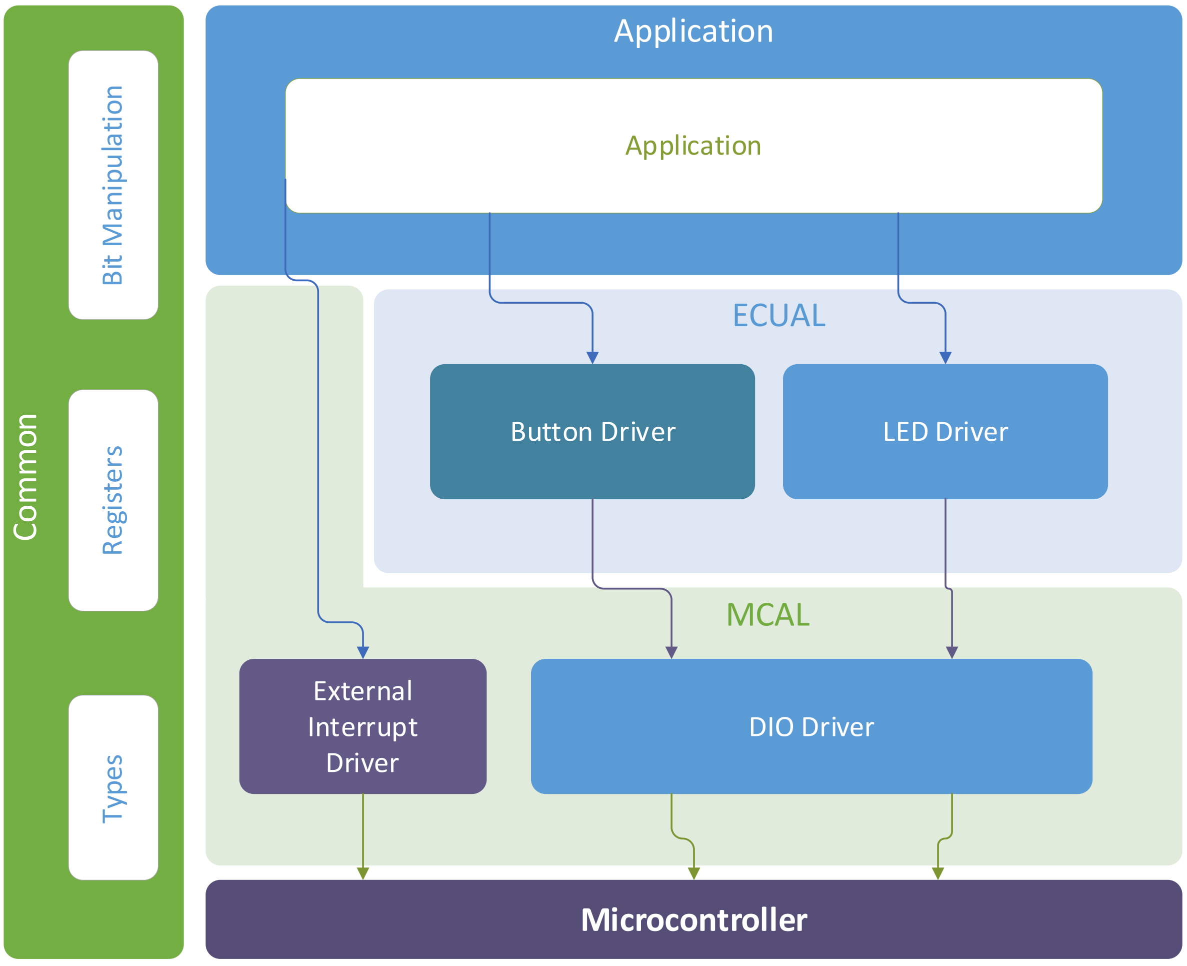
Software used: Microsoft Visio



Layered Architecture

Software used: Microsoft Visio

LED Sequence V2.0
Layered Architecture



Project Modules APIs

DIO Driver

DIO Macros/Enums:

Type	Name	Values	Desc

#define	LOW
	HIGH
		LOW = 0
		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 • Out = 1	**Defines DIO pin direction**
typedef enum	EN_DIO_Error_T	• DIO_OK • 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));\`
	
	`void INT_VECT(void)`
		ISR definition

typedef enum	`typedef enum EN_EXI_INT_t {`
	`    INT0, INT1`
	`} EN_EXI_INT_t;`
		Defines Interrupt port names

typedef enum	`typedef enum EN_EXI_SENSE_t {`
	`    // Interrupts on low level`
	`    LOW_LEVEL = 0xFC,`
	`    // Interrupts on any logical change`
	`    ANY_LEVEL = 0x01,`
	`    // Interrupts on Falling edge`
	`    FALLING_EDGE = 0x02,`
	`    // Interrupts on Rising edge`
	`    RISING_EDGE = 0x03`
	`} EN_EXI_SENSE_t;`
		Enum for ATmega32 interrupt sense modes

typedef enum	`typedef enum EN_EXI_ERROR_t {`
	`    EXI_OK,`
	`    EXI_ERROR`
	`} 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 {     LED_OK,     LED_ERROR }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`
	`{`
	`    BUTTON_OK,`
	`    BUTTON_ERROR`
	`}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:

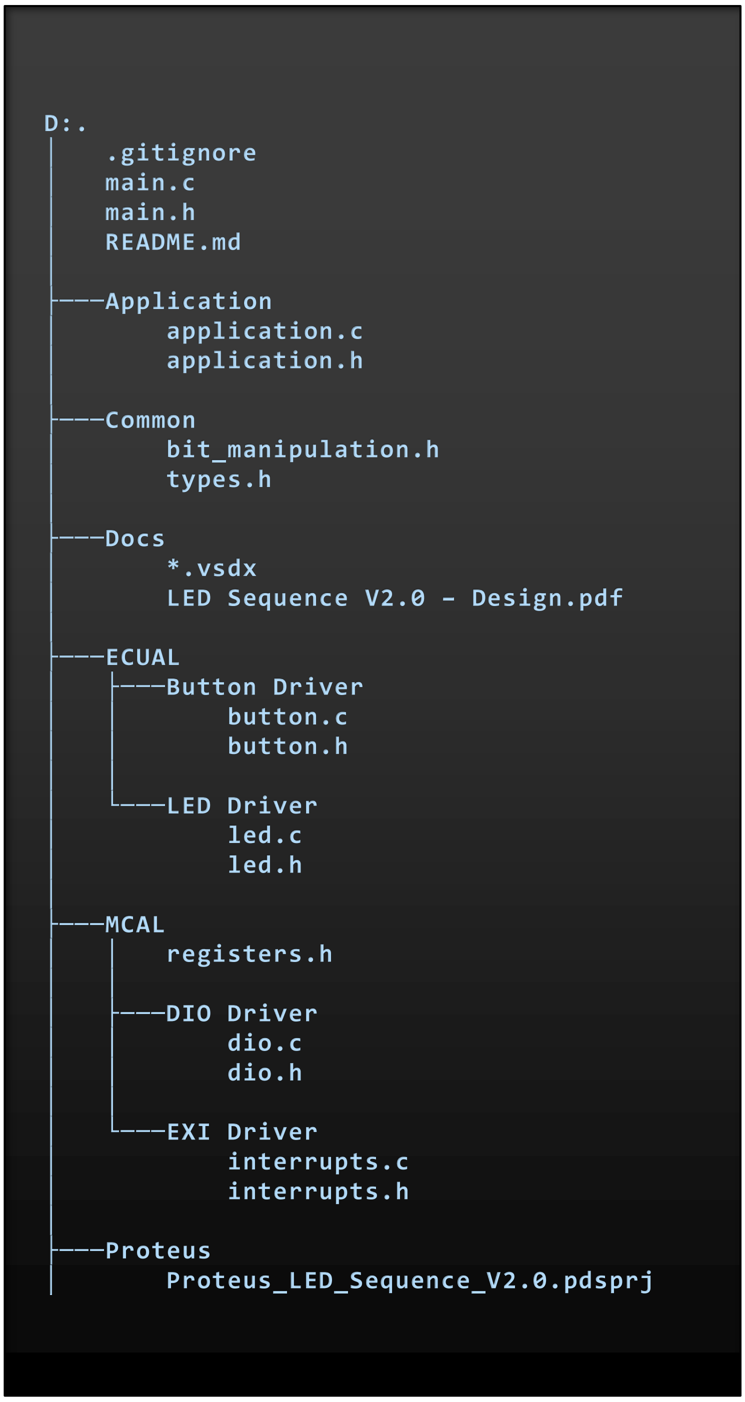
```c
#include "../ECUAL/LED Driver/led.h"
#include "../ECUAL/Button Driver/button.h"
#include "../MCAL/EXI Driver/interrupts.h"
```

Application Functions:

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

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

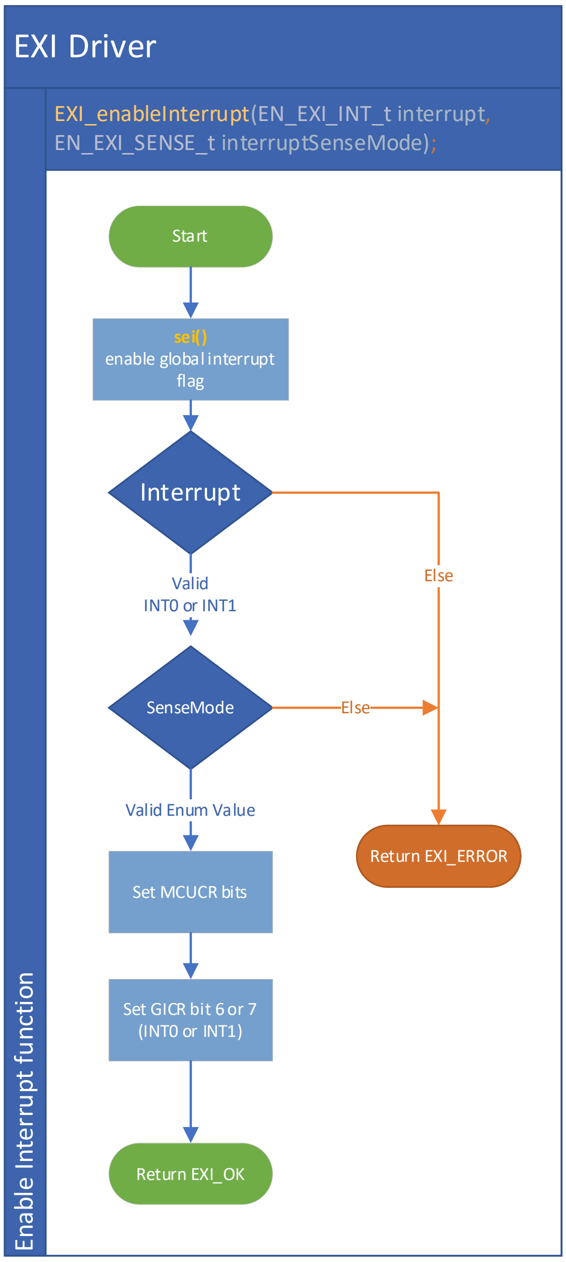
Project Tree

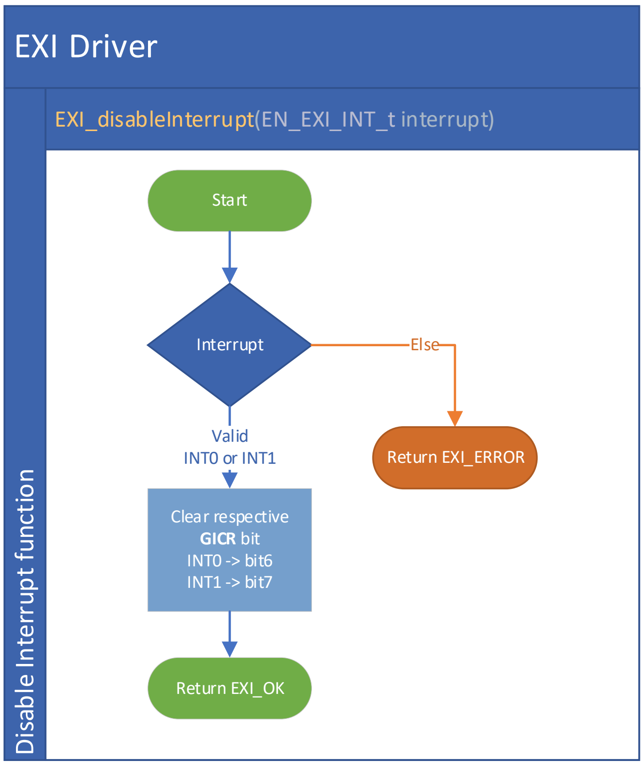


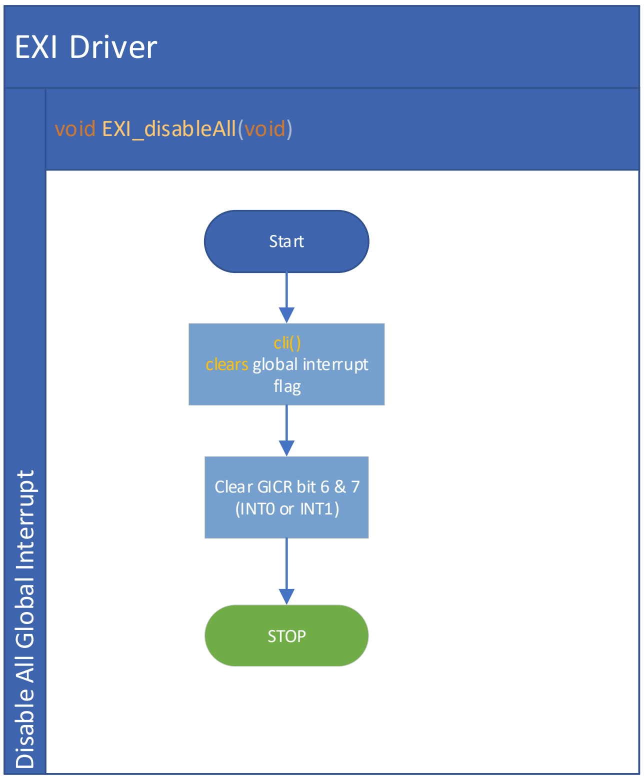
```
D:.
│   .gitignore
│   main.c
│   main.h
│   README.md
│
├───Application
│       application.c
│       application.h
│
├───Common
│       bit_manipulation.h
│       types.h
│
├───Docs
│       *.vsdx
│       LED Sequence V2.0 - Design.pdf
│
├───ECUAL
│   ├───Button Driver
│   │       button.c
│   │       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_V2.0.pdsprj
```

Project Modules APIs Charts

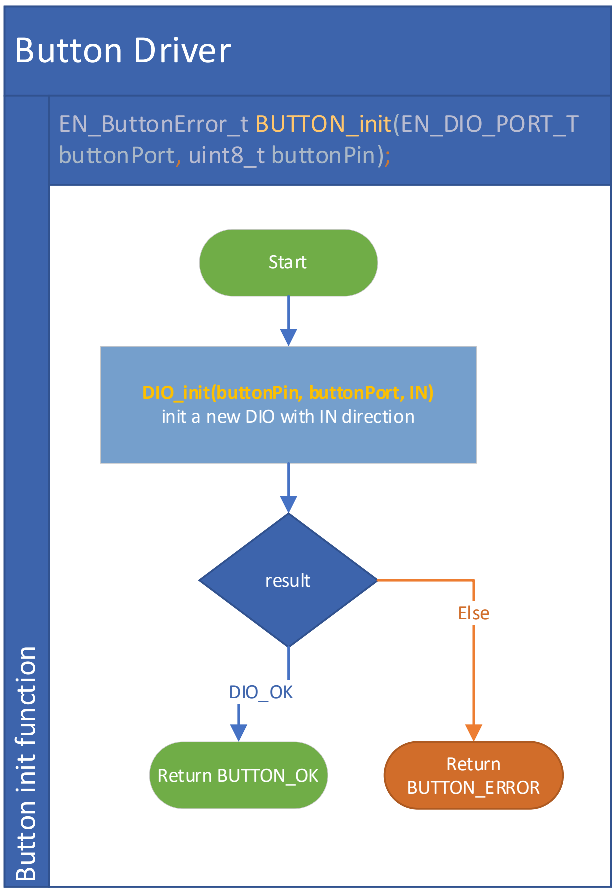
EXI Flowcharts

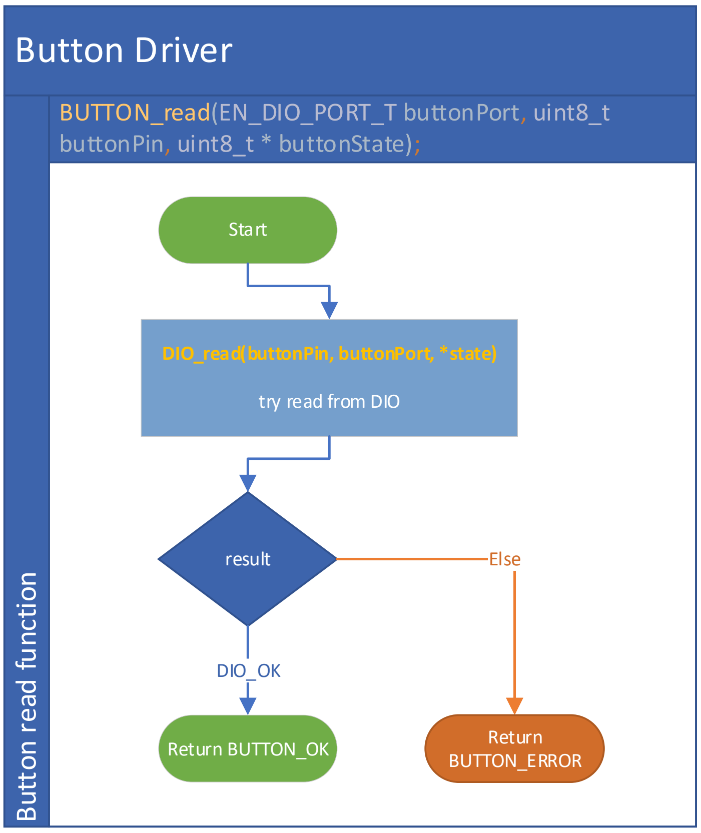




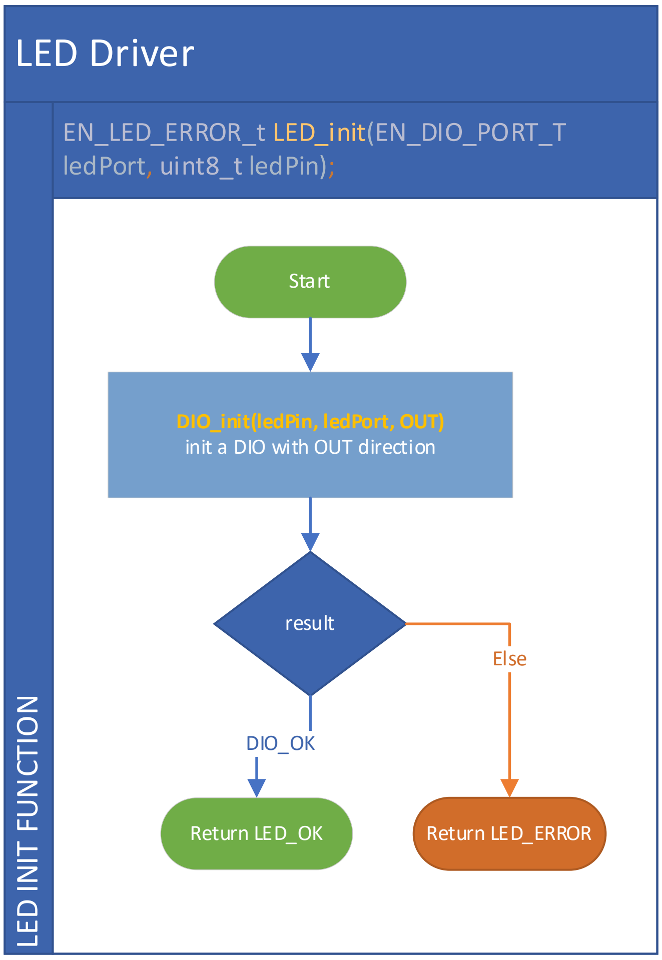

EXI Driver
void EXI_disableAll(void)
Disable All Global Interrupt
Start
cli()
clears global interrupt flag
Clear GICR bit 6 & 7
(INT0 or INT1)
STOP

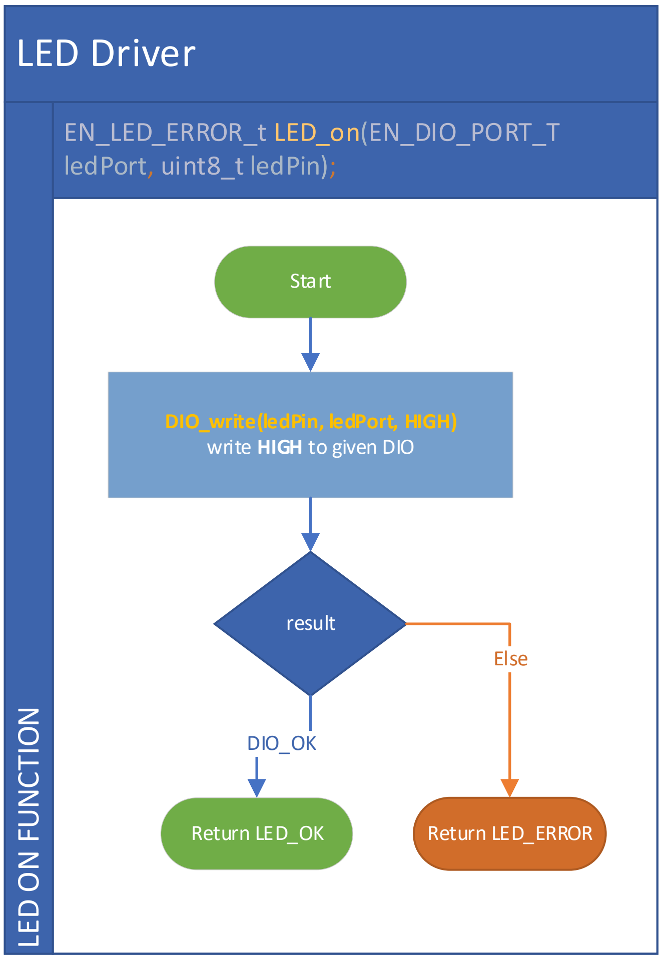
Button API Flowcharts

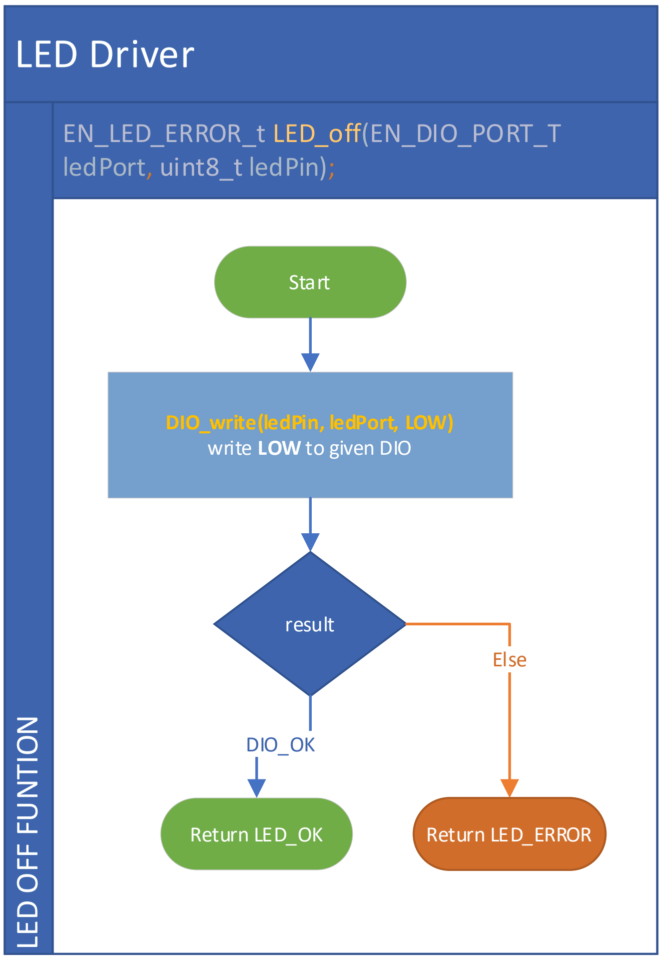

Button Driver
EN_ButtonError_t BUTTON_init(EN_DIO_PORT_T buttonPort, uint8_t buttonPin);
Button init function
Start
DIO_init(buttonPin, buttonPort, IN)
init a new DIO with IN direction
result
Else
DIO_OK
Return BUTTON_OK
Return BUTTON_ERROR

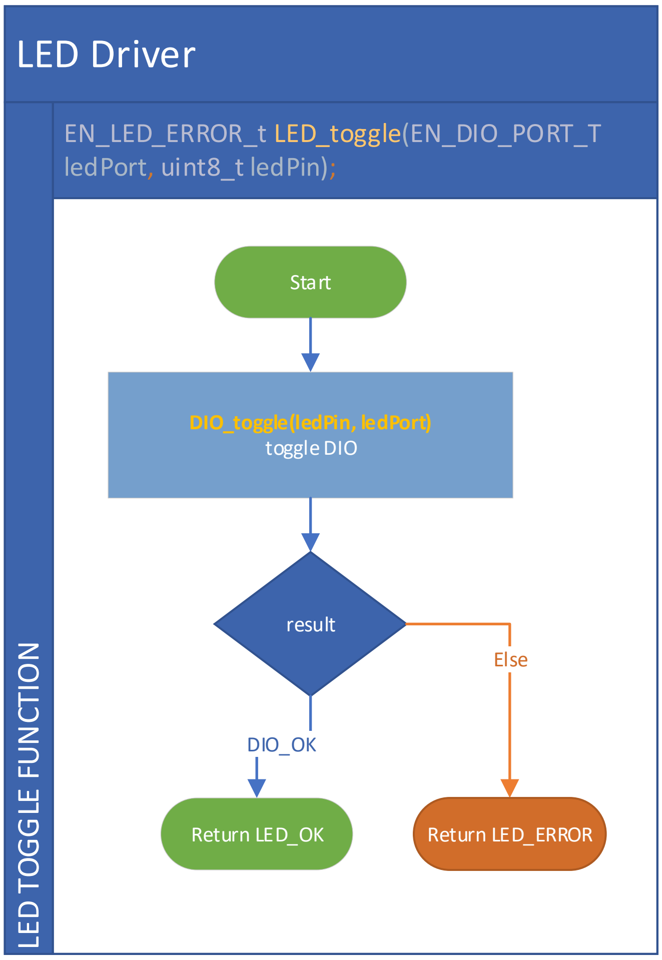

Button Driver
BUTTON_read(EN_DIO_PORT_T buttonPort, uint8_t buttonPin, uint8_t * buttonState);
Button read function
Start
DIO_read(buttonPin, buttonPort, *state)
try read from DIO
result
Else
DIO_OK
Return BUTTON_OK
Return BUTTON_ERROR

LED API Flowcharts

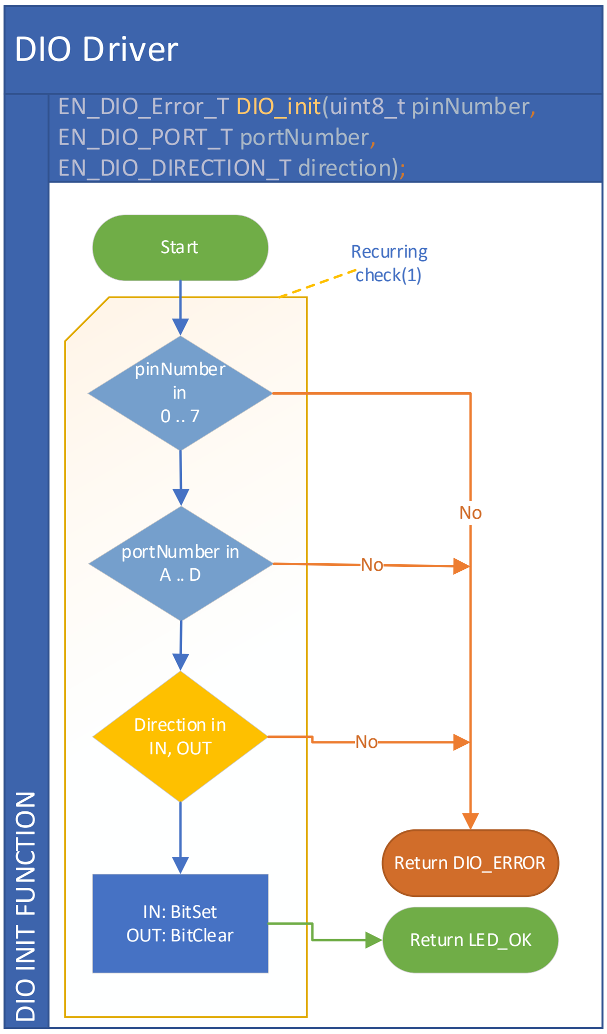

LED Driver
EN_LED_ERROR_t LED_init(EN_DIO_PORT_T ledPort, uint8_t ledPin);
LED INIT FUNCTION
Start
DIO_init(ledPin, ledPort, OUT)
init a DIO with OUT direction
result
Else
DIO_OK
Return LED_OK
Return LED_ERROR

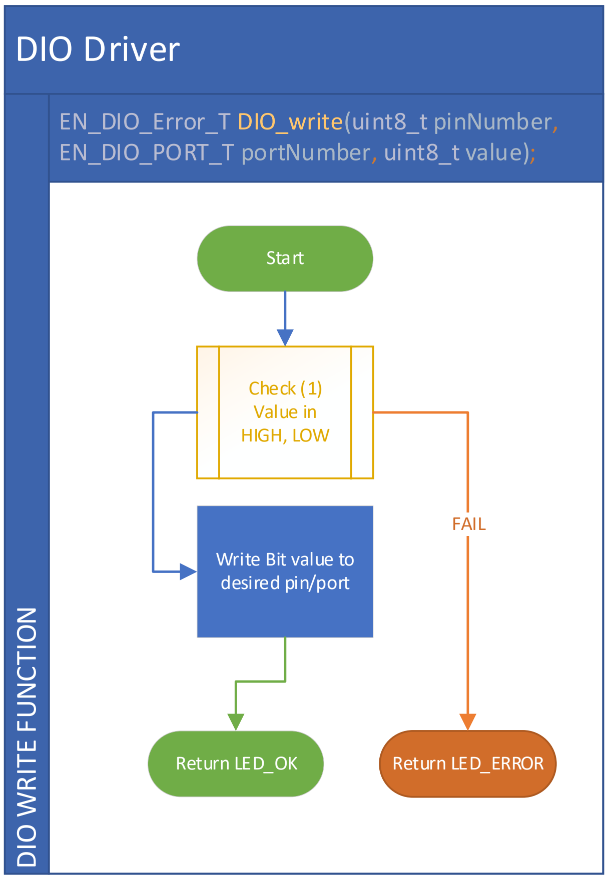


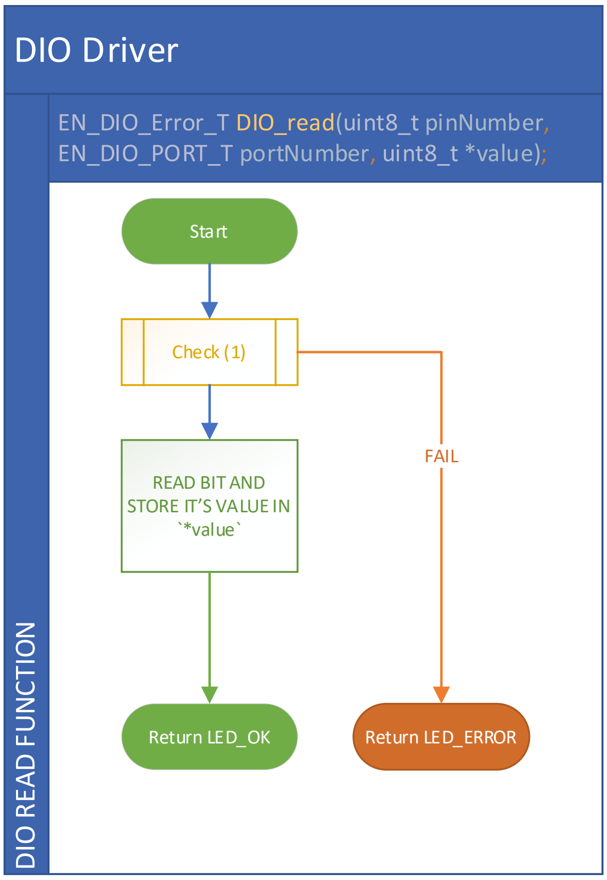


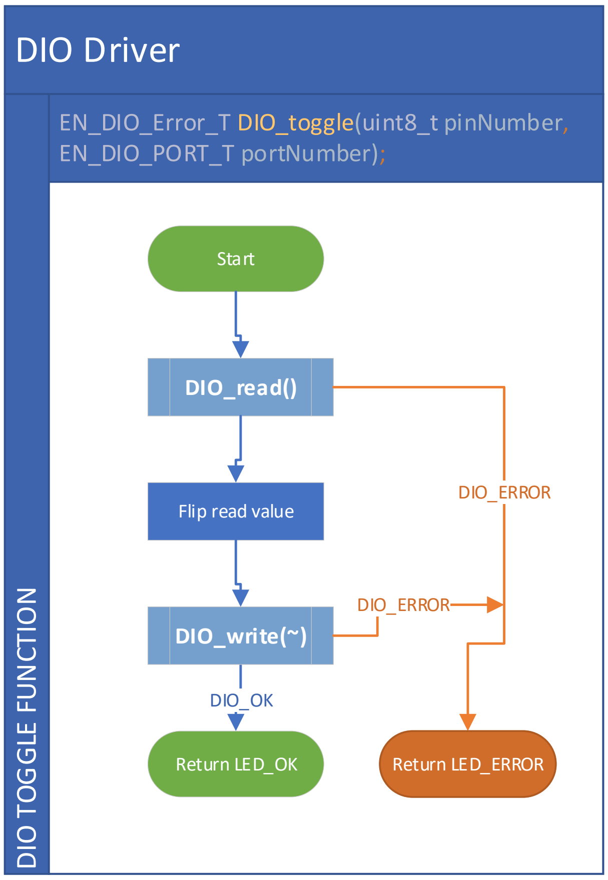

LED Driver
EN_LED_ERROR_t LED_toggle(EN_DIO_PORT_T ledPort, uint8_t ledPin);
LED TOGGLE FUNCTION
Start
DIO_toggle(ledPin, ledPort)
toggle DIO
result
Else
DIO_OK
Return LED_OK
Return LED_ERROR

DIO API Flowcharts




DIO Driver
EN_DIO_Error_T DIO_write(uint8_t pinNumber, EN_DIO_PORT_T portNumber, uint8_t value);
DIO WRITE FUNCTION
Start
Check (1) Value in HIGH, LOW
Write Bit value to desired pin/port
FAIL
Return LED_OK
Return LED_ERROR


DIO Driver
EN_DIO_Error_T DIO_read(uint8_t pinNumber, EN_DIO_PORT_T portNumber, uint8_t *value);
DIO READ FUNCTION
Start
Check (1)
READ BIT AND STORE IT'S VALUE IN `*value`
FAIL
Return LED_OK
Return LED_ERROR



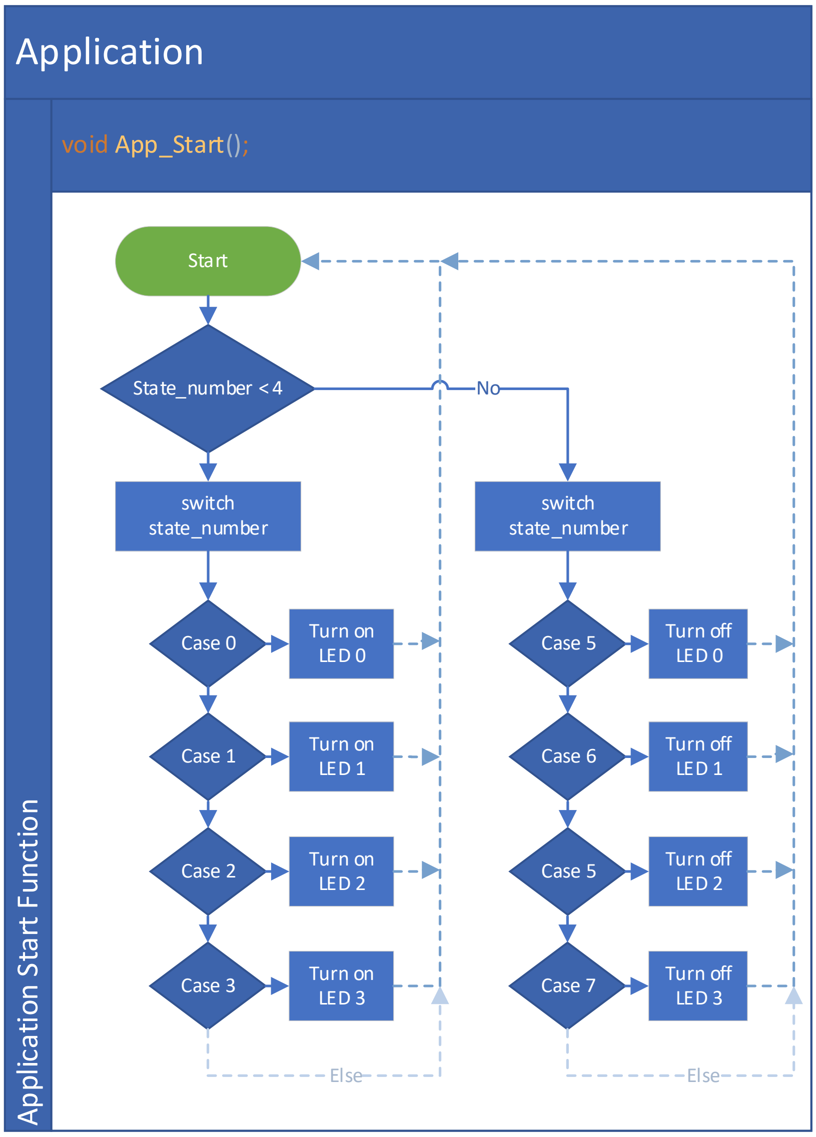
Application API Flowcharts

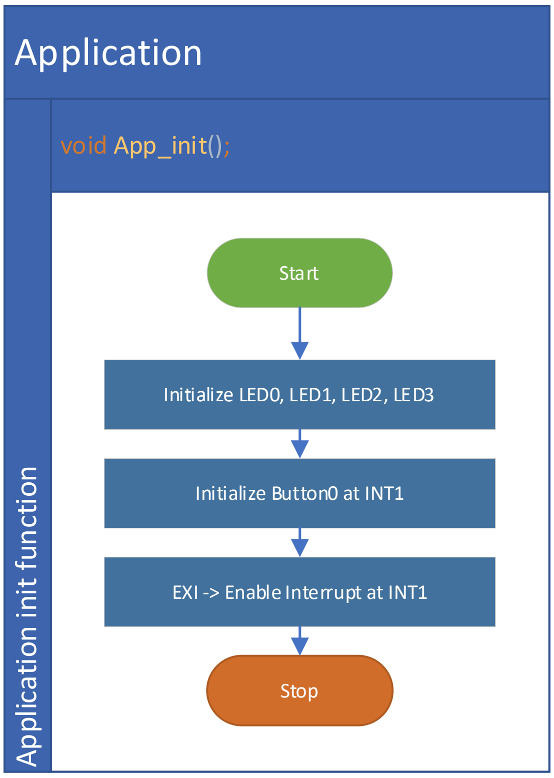
Application

Globals

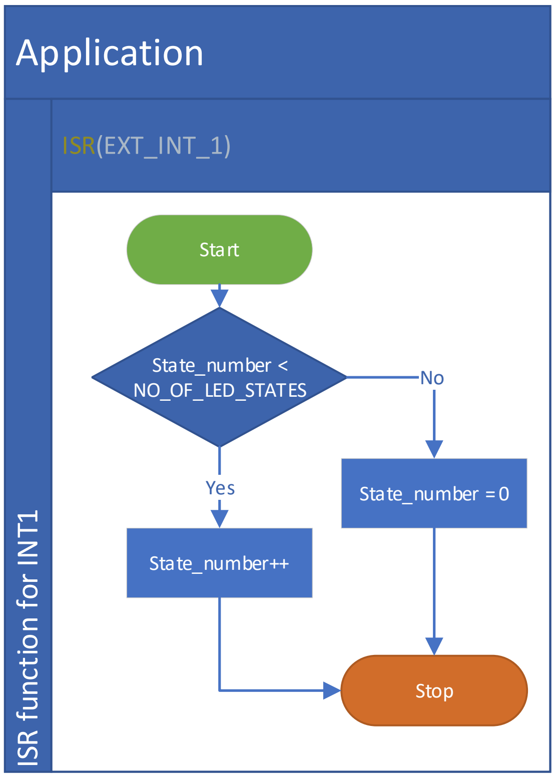
```
/* LEDs */
#define LED_0_PORT C
#define LED_0_PIN 0
#define LED_1_PORT C
#define LED_1_PIN 1
#define LED_2_PORT C
#define LED_2_PIN 2
#define LED_3_PORT C
#define LED_3_PIN 3

/* 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_init();
Application init function
Start
Initialize LED0, LED1, LED2, LED3
Initialize Button0 at INT1
EXI -> Enable Interrupt at INT1
Stop



Application
ISR(EXT_INT_1)
ISR function for INT1
Start
State_number < NO_OF_LED_STATES
No
Yes
State_number = 0
State_number++
Stop