

LED Sequence V2.0

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 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)

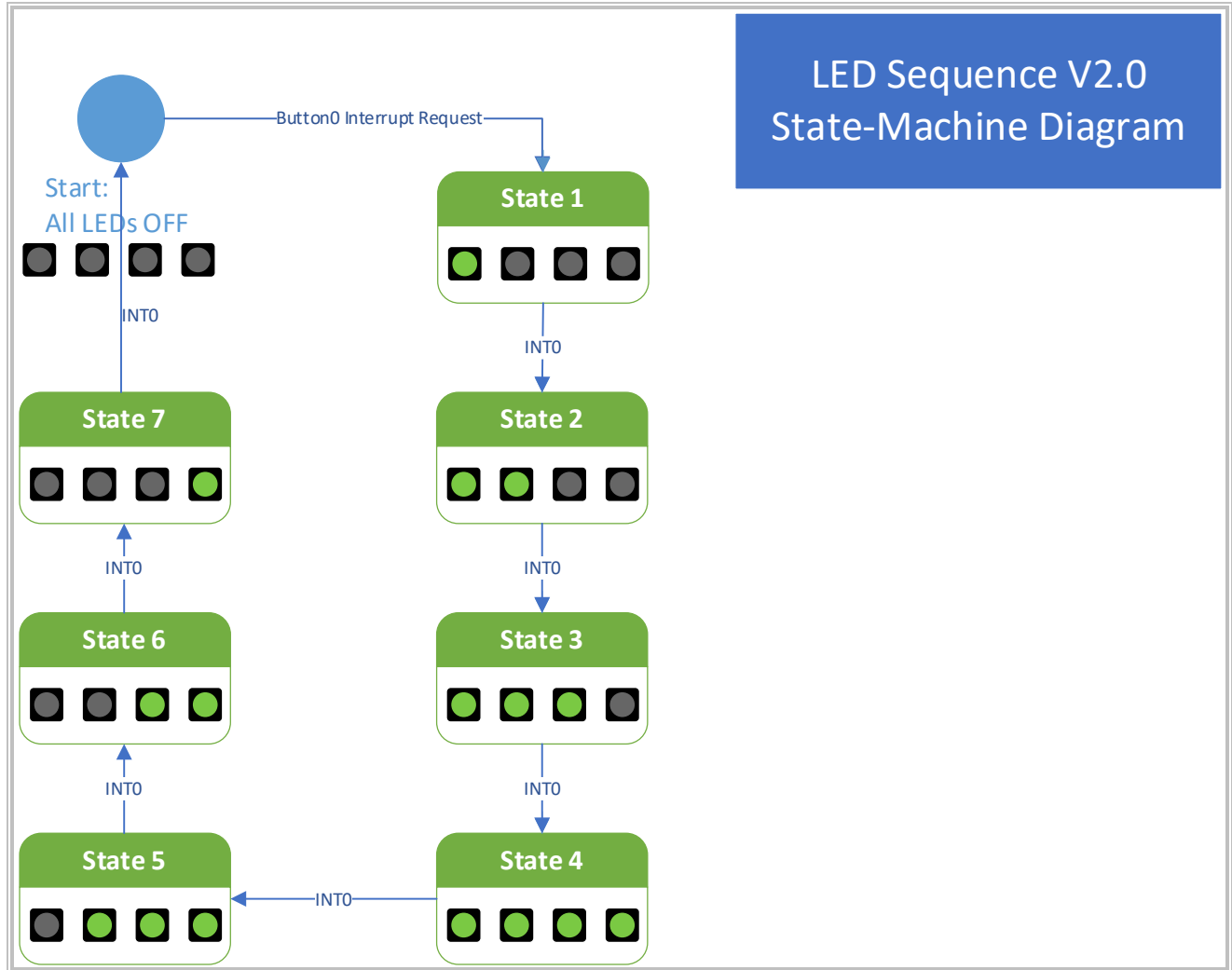
USE EXTERNAL INTERRUPTS

System Design

State Machine Diagram

Software used: Microsoft Visio

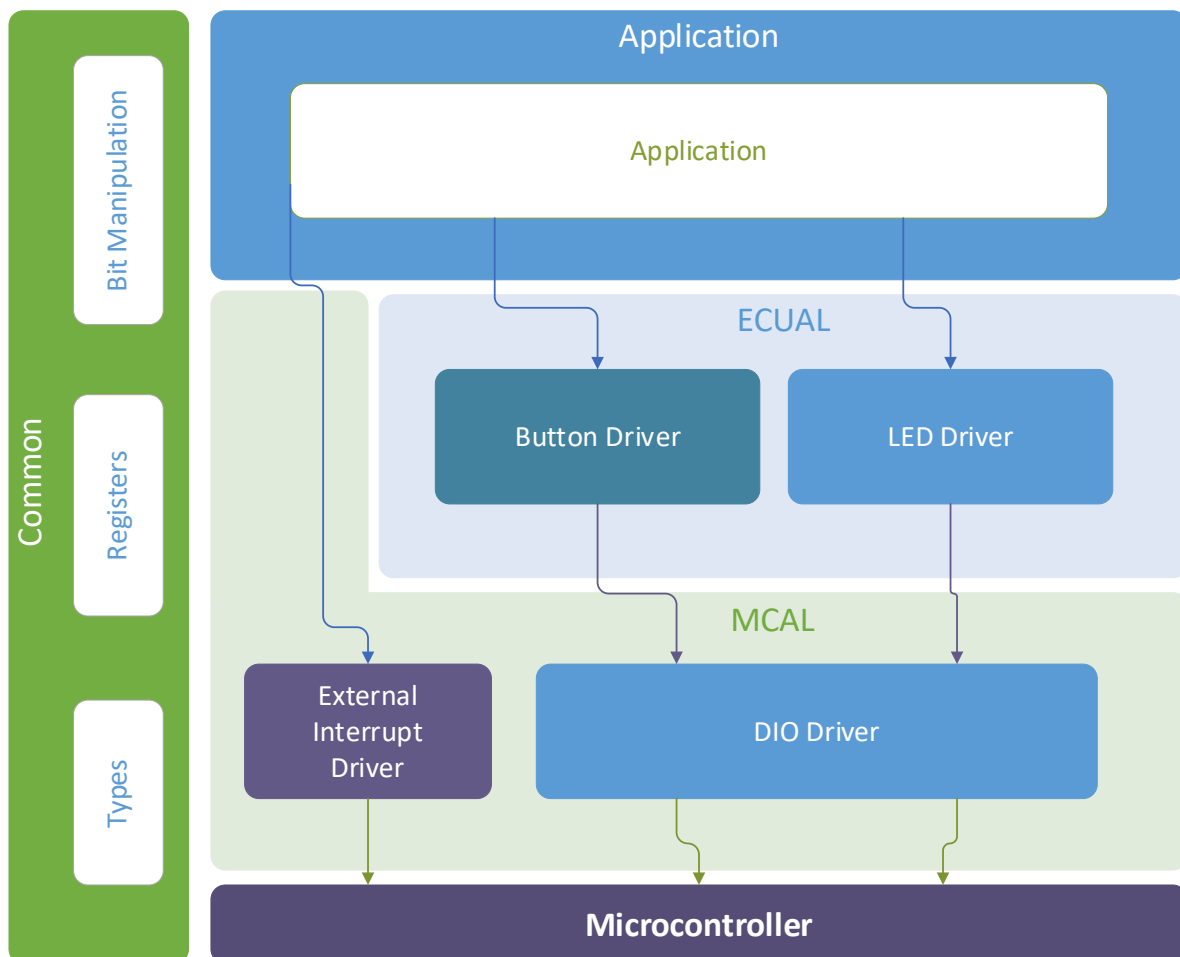
LED Sequence V2.0 State-Machine Diagram



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	<ul style="list-style-type: none">A, B, C, D	Defines available DIO ports
typedef enum	EN_DIO_DIRECTION_T	<ul style="list-style-type: none">In = 0Out = 1	Defines DIO pin direction
typedef enum	EN_DIO_Error_T	<ul style="list-style-type: none">DIO_OKDIO_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));\n\nvoid INT_VECT(void)	ISR definition
typedef enum	typedef enum EN_EXI_INT_t {\n INT0, INT1\n} EN_EXI_INT_t;	Defines Interrupt port names
typedef enum	typedef enum EN_EXI_SENSE_t {\n // Interrupts on Low Level\n LOW_LEVEL = 0xFC,\n // Interrupts on any Logical change\n ANY_LEVEL = 0x01,\n // Interrupts on Falling edge\n FALLING_EDGE = 0x02,\n // Interrupts on Rising edge\n RISING_EDGE = 0x03\n} EN_EXI_SENSE_t;	Enum for ATmega32 interrupt sense modes
typedef enum	typedef enum EN_EXI_ERROR_t {\n EXI_OK,\n EXI_ERROR\n} EN_EXI_ERROR_t;	Error return type for EXI API

EXI Functions:

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

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

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

LED Driver

LED Macros/Enums:

Type	Name/Value	Desc
typedef enum	<pre>typedef enum EN_LED_ERROR_t { LED_OK, LED_ERROR } EN_LED_ERROR_t;</pre>	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	<pre>typedef enum EN_ButtonError_t { BUTTON_OK, BUTTON_ERROR }EN_ButtonError_t;</pre>	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 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
```

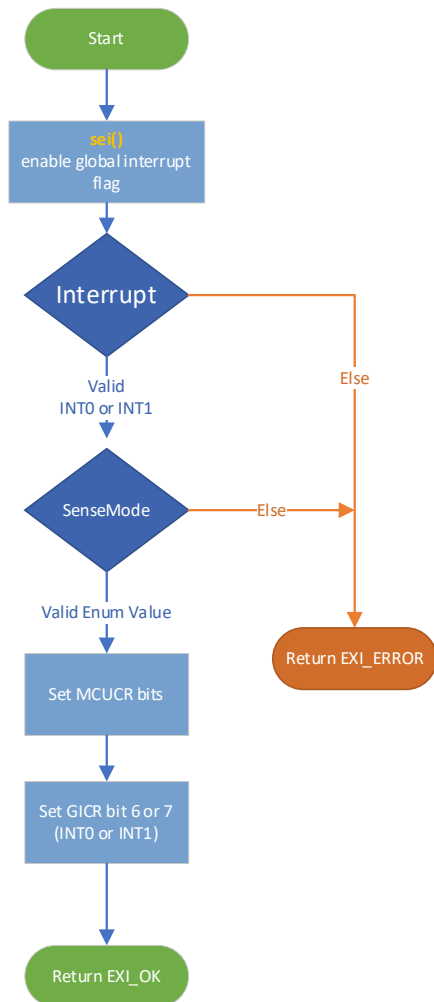
Project Modules APIs Charts

EXI Flowcharts

EXI Driver

`EXI_enableInterrupt(EN_EXI_INT_t interrupt,
EN_EXI_SENSE_t interruptSenseMode);`

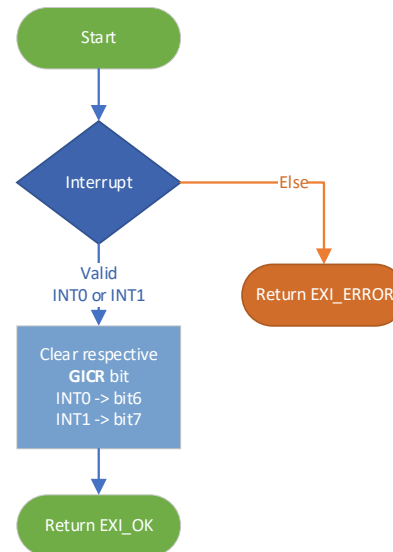
Enable Interrupt function



EXI Driver

`EXI_disableInterrupt(EN_EXI_INT_t interrupt)`

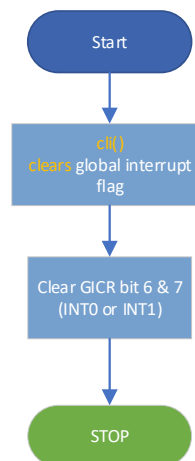
Disable Interrupt function



EXI Driver

`void EXI_disableAll(void)`

Disable All Global Interrupt

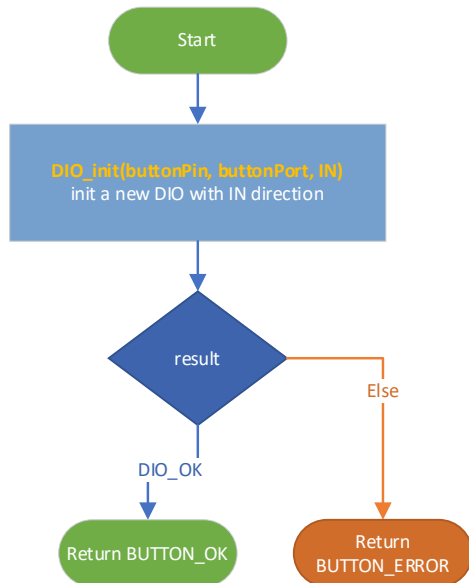


Button API Flowcharts

Button Driver

```
EN_ButtonError_t BUTTON_init(EN_DIO_PORT_T  
buttonPort, uint8_t buttonPin);
```

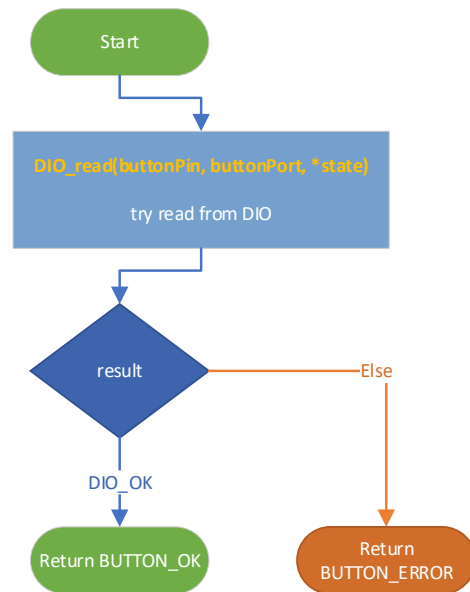
Button init function



Button Driver

```
BUTTON_read(EN_DIO_PORT_T buttonPort, uint8_t  
buttonPin, uint8_t * buttonState);
```

Button read function

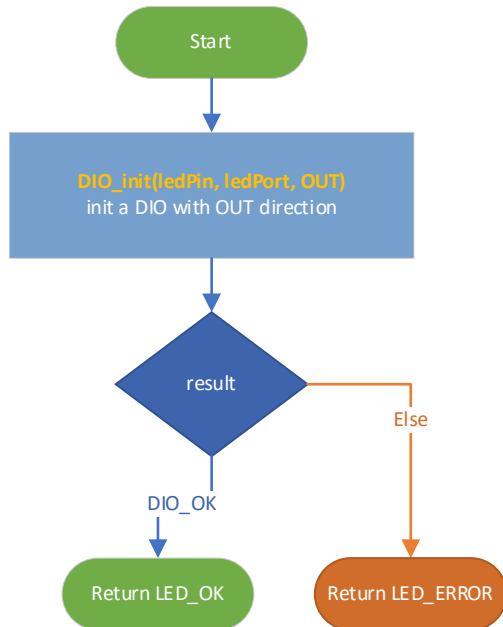


LED API Flowcharts

LED Driver

EN_LED_ERROR_t **LED_init**(EN_DIO_PORT_T
ledPort, uint8_t ledPin);

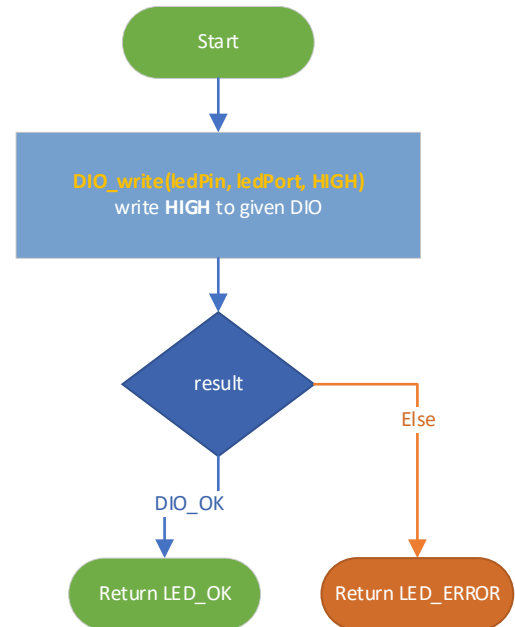
LED INIT FUNCTION



LED Driver

EN_LED_ERROR_t **LED_on**(EN_DIO_PORT_T
ledPort, uint8_t ledPin);

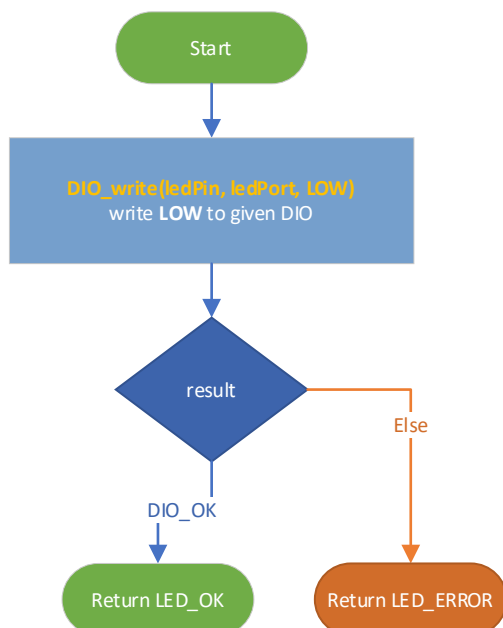
LED ON FUNCTION



LED Driver

EN_LED_ERROR_t **LED_off**(EN_DIO_PORT_T
ledPort, uint8_t ledPin);

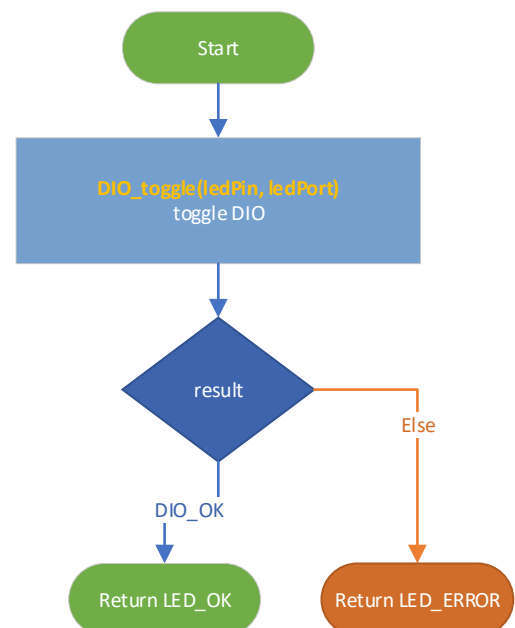
LED OFF FUNCTION



LED Driver

EN_LED_ERROR_t **LED_toggle**(EN_DIO_PORT_T
ledPort, uint8_t ledPin);

LED TOGGLE FUNCTION

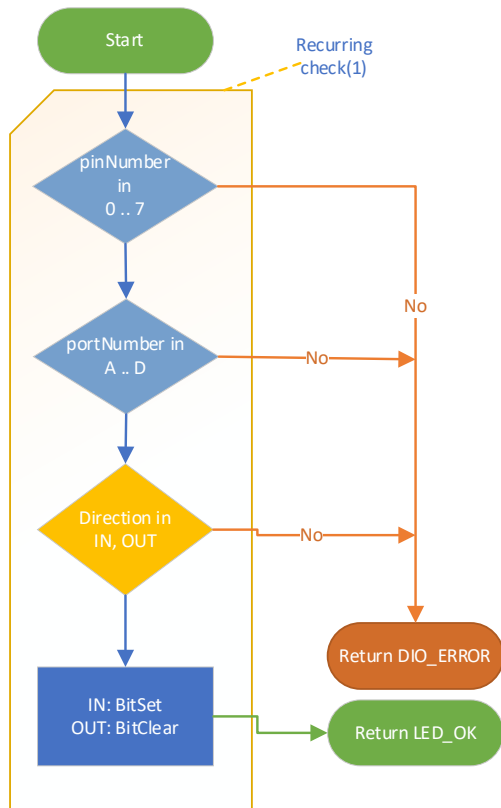


DIO API Flowcharts

DIO Driver

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

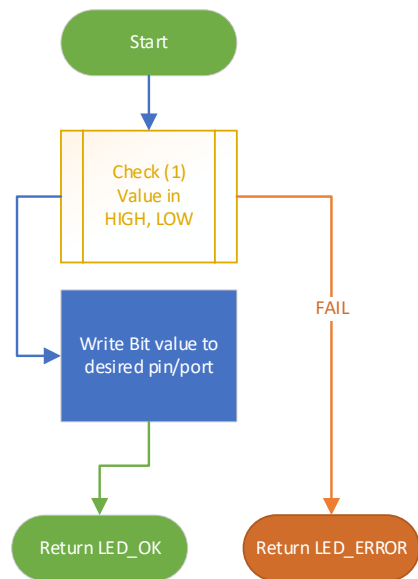
DIO INIT FUNCTION



DIO Driver

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

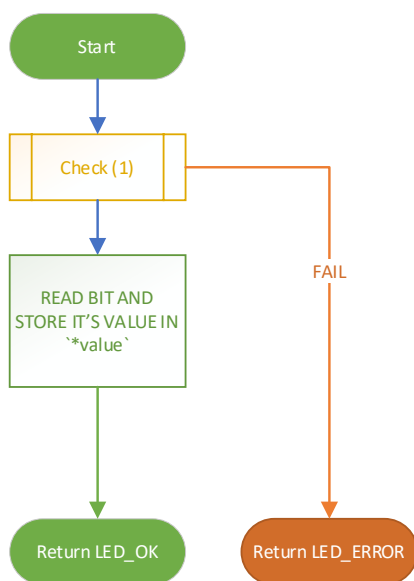
DIO WRITE FUNCTION



DIO Driver

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

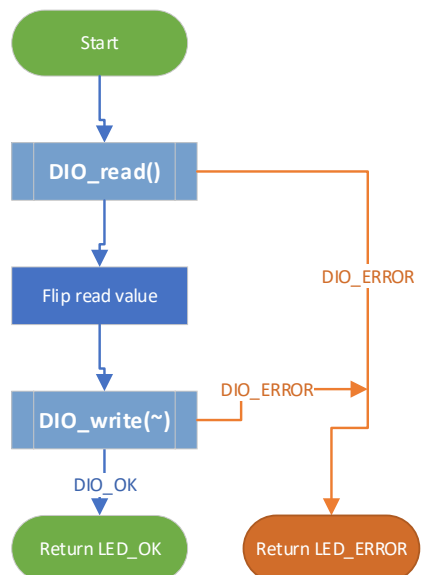
DIO READ FUNCTION



DIO Driver

```
EN_DIO_Error_T DIO_toggle(uint8_t pinNumber,  
EN_DIO_PORT_T portNumber);
```

DIO TOGGLE FUNCTION



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

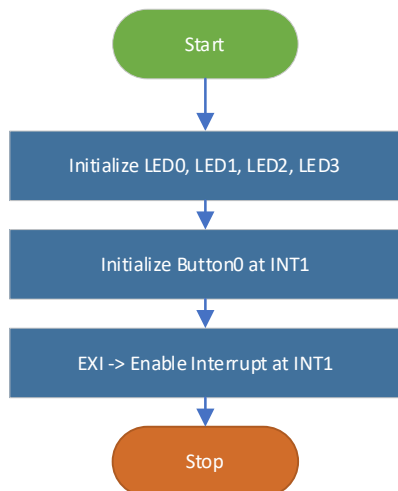
/// Global Variables
uint8_t state_number = 7;
```

Globals

Application

void App_init();

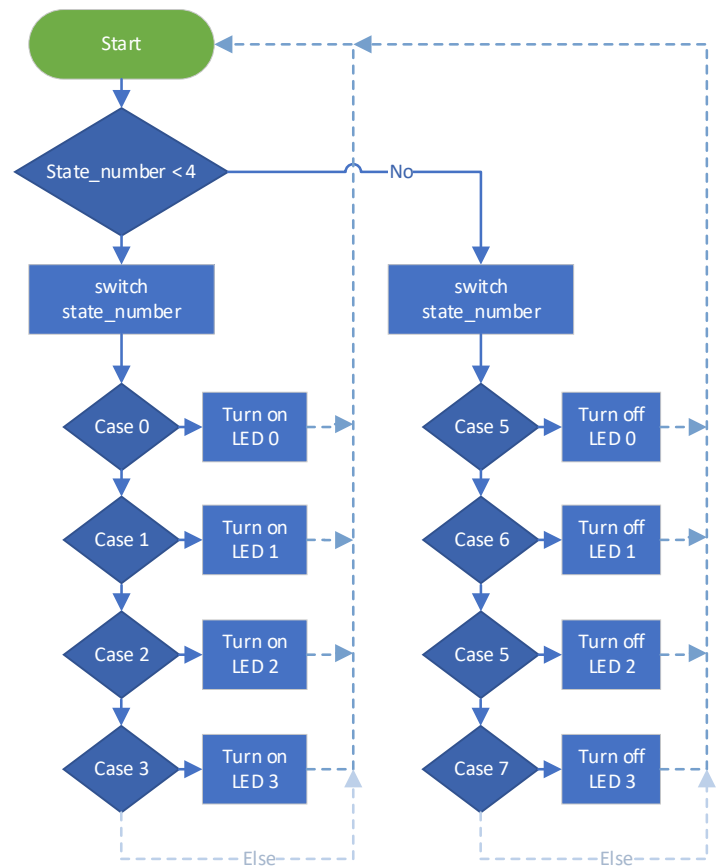
Application init function



Application

void App_Start();

Application Start Function



Application

ISR(EXT_INT_1)

ISR function for INT1

