LED Sequence V3.0

HOSSAM ELWAHSH

EMBEDDED SYSTEMS - LEVEL 1

Table of Contents

LED Sequence V3.0

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)
- Two buttons (BUTTON0, BUTTON1)
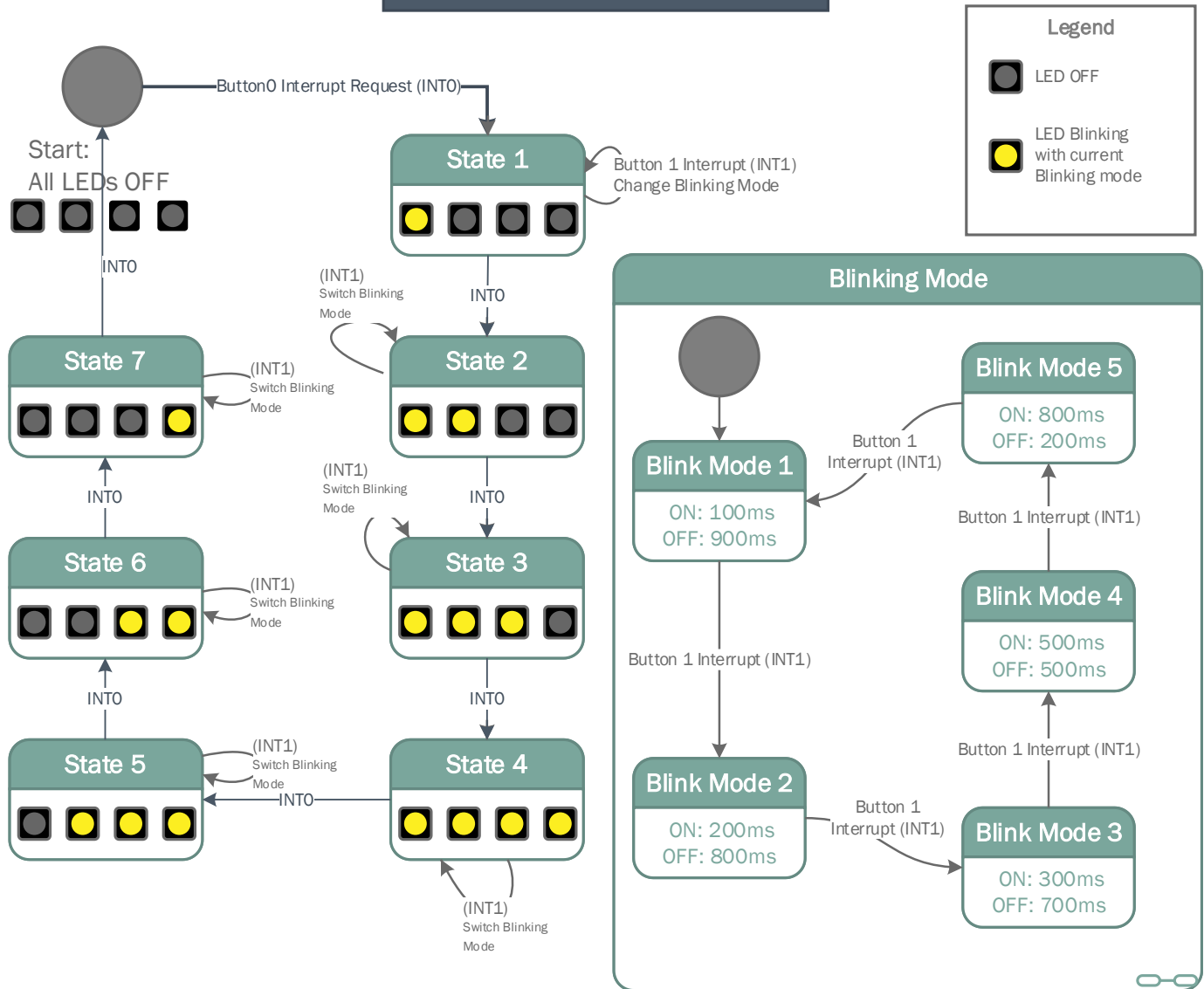
Software Requirements

1. Initially, all LEDs are OFF
2. Once **BUTTON0** is pressed, **LED0** will blink with **BLINK_1** mode
3. Each press further will make another LED blinks **BLINK_1** mode
4. At the **fifth press**, **LED0** will changed to be **OFF**
5. Each **press further** will make only one LED is **OFF**
6. This will be repeated forever
7. The sequence is described below
    1. Initially (OFF, OFF, OFF, OFF)
    2. Press 1 (BLINK_1, OFF, OFF, OFF)
    3. Press 2 (BLINK_1, BLINK_1, OFF, OFF)
    4. Press 3 (BLINK_1, BLINK_1, BLINK_1, OFF)
    5. Press 4 (BLINK_1, BLINK_1, BLINK_1, BLINK_1)
    6. Press 5 (OFF, BLINK_1, BLINK_1, BLINK_1)
    7. Press 6 (OFF, OFF, BLINK_1, BLINK_1)
    8. Press 7 (OFF, OFF, OFF, BLINK_1)
    9. Press 8 (OFF, OFF, OFF, OFF)
    10. Press 9 (BLINK_1, OFF, OFF, OFF)

8. When BUTTON1 has pressed the blinking on and off durations will be changed
    1. No press → **BLINK_1** mode (**ON**: 100ms, **OFF**: 900ms)
    2. First press → **BLINK_2** mode (**ON**: 200ms, **OFF**: 800ms)
    3. Second press → **BLINK_3** mode (**ON**: 300ms, **OFF**: 700ms)
    4. Third press → **BLINK_4** mode (**ON**: 500ms, **OFF**: 500ms)
    5. Fourth press → **BLINK_5** mode (**ON**: 800ms, **OFF**: 200ms)
    6. Fifth press → **BLINK_1** mode

9. **USE EXTERNAL INTERRUPTS**

System Design

State Machine Diagram
Software used: Microsoft Visio

LED Sequence V3.0 State-Machine Diagram

Legend
- ⬛ LED OFF
- 🟡 LED Blinking with current Blinking mode

Start: All LEDs OFF

Button0 Interrupt Request (INT0)

State 1 — Button 1 Interrupt (INT1) Change Blinking Mode

State 2 — (INT1) Switch Blinking Mode

State 3 — (INT1) Switch Blinking Mode

State 4 — (INT1) Switch Blinking Mode

State 5 — (INT1) Switch Blinking Mode

State 6 — (INT1) Switch Blinking Mode

State 7 — (INT1) Switch Blinking Mode

Transitions: INT0

Blinking Mode

Blink Mode 1 — ON: 100ms OFF: 900ms

Button 1 Interrupt (INT1)

Blink Mode 2 — ON: 200ms OFF: 800ms

Button 1 Interrupt (INT1)

Blink Mode 3 — ON: 300ms OFF: 700ms

Button 1 Interrupt (INT1)

Blink Mode 4 — ON: 500ms OFF: 500ms

Button 1 Interrupt (INT1)

Blink Mode 5 — ON: 800ms OFF: 200ms

Button 1 Interrupt (INT1)

LED Sequence V3.0
Layered Architecture

Application

Application

ECUAL

Button Driver

LED Driver

MCAL

External Interrupt Driver

DIO Driver

Timer Driver

Microcontroller

Common

Bit Manipulation

Registers

Types
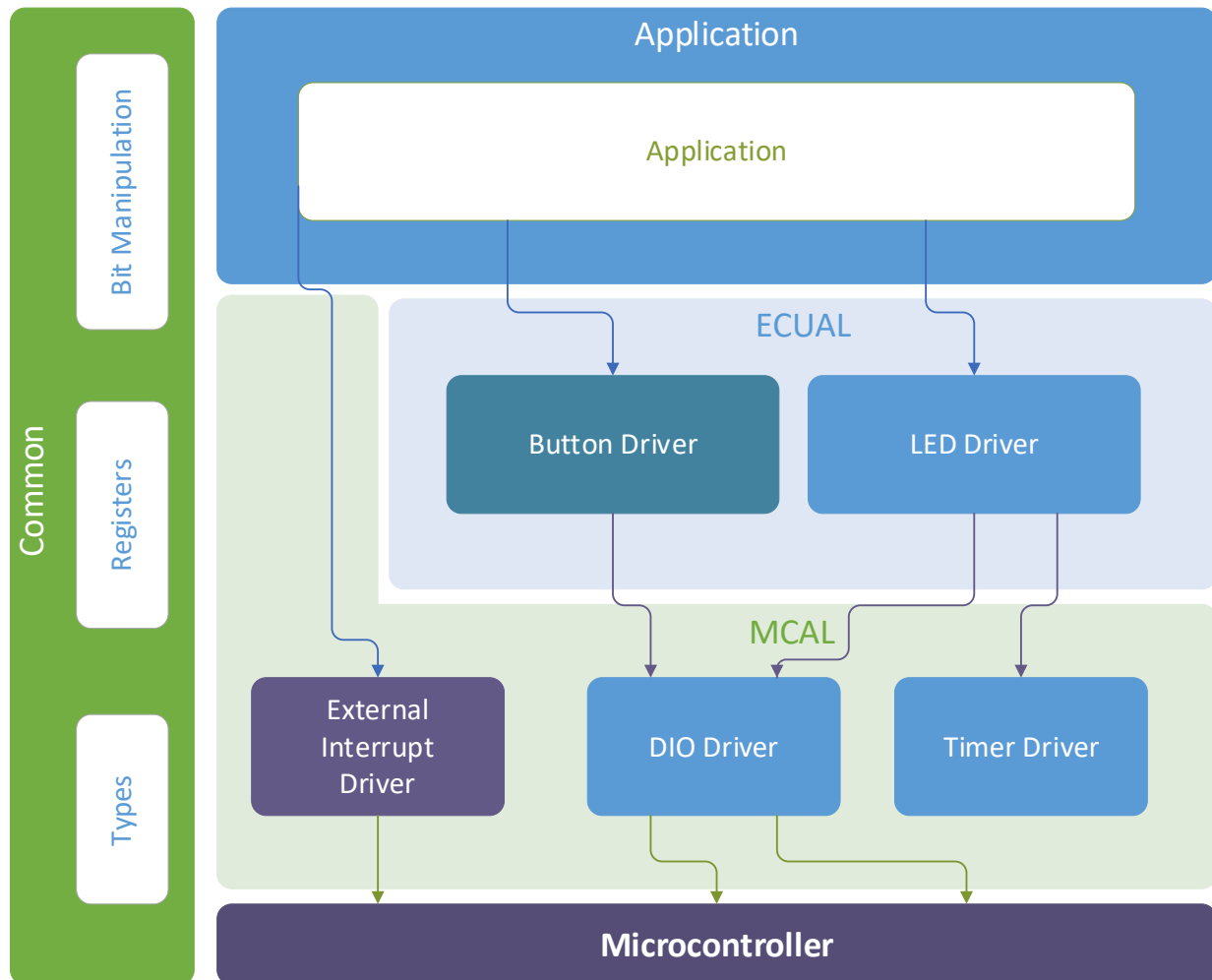
Project Modules APIs
DIO Driver
DIO Macros/Enums:

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);
```

```c
/**
 * Writes a byte to a given PORT
 * @param portNumber [in] Port to use
 * @param byte [in] value to write
 */
EN_DIO_Error_T DIO_port_write(EN_DIO_PORT_T portNumber, uint8_t byte, uint8_t mask);

/**
 * Toggles a given PORT
 * @param portNumber [in] Port to use
 * @param mask [in] (optional, 0 to disable)
 */
EN_DIO_Error_T DIO_port_toggle(EN_DIO_PORT_T portNumber, uint8_t mask);

/**
 * 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);
```

EXI (External Interrupt) Driver

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:

```c
/**
 * 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);
```

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

```c
/**
 * 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:

```c
/**
 * 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);
```

```c
/**
 * 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);
```

```c
/**
 * 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);
```

```c
/**
 * 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);
```

LED ARRAYS Functions

```
/**
 * Turns on a LED Array at given PORT
 * @param ledPort [in] LED Port
 * @param mask [in] (optional, 0 to disable)
 * \n mask to turn on specific LEDs only | e.g. to only turn on the first LED use 0x01 (0b0000 0001)
 */
EN_LED_ERROR_t LED_array_on(EN_DIO_PORT_T ledPort, uint8_t mask);
```

```
/**
 * Turns off a LED Array at given PORT
 * @param ledPort [in] LED Port
 * @param mask [in] (optional, 0 to disable)
 * \n mask to turn off specific LEDs only | e.g. to only turn off the first LED use 0x01 (0b0000 0001)
 */
EN_LED_ERROR_t LED_array_off(EN_DIO_PORT_T ledPort, uint8_t mask);
```

```
/**
 * Toggles a LED Array at given PORT
 * @param ledPort [in] LED Port
 * @param mask [in] (optional, 0 to disable)
 * \n mask to turn off specific LEDs only | e.g. to only turn off the first LED use 0x01 (0b0000 0001)
 */
EN_LED_ERROR_t LED_array_toggle(EN_DIO_PORT_T ledPort, uint8_t mask);
```

```
/**
 * Blinks LED array once at given port
 * @param ledPort [in] LED Port
 * @param ledPin [in] LED Pin number in ledPort
 * @param onTime [in] Time in which LED will be on (milliseconds)
 * @param offTime [in] Time in which LED will be off (milliseconds)
 * @param mask [in] optional, 0 to disable i.e. blinks all LEDs
 * \n mask to blink specific LEDs only | e.g. to only blink the first LED use 0x01 (0b0000 0001)
 */
void LED_array_blink(EN_DIO_PORT_T ledPort, uint16_t onTime, uint16_t offTime, uint8_t mask);
```

Button Driver

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);
```

typedef enum EN_ButtonError_t
{
    BUTTON_OK,
    BUTTON_ERROR
}EN_ButtonError_t;

Timer Driver

Timer Macros/Enums:

Type	Name/Value	Desc

#define(s)	```/* Microcontroller Related Macros */```
	```#define timerNBits 8```
	```#define SystemClockInMhz 1```
	
	```/* Clears mode bits in timer */```
	```#define TimerClearModes() TCCR0 &= 0xB7```
	
	```/* Clears Clock selection bits */```
	```#define TimerClearClockSelection() TCCR0 &= 0xF8```	

typdef enum	```typedef enum EN_TimerMode_t```
	```{```
	``` NORMAL = 0xB7,```
	``` CTC = 0x08,```
	``` FAST_PWM = 0x48,```
	``` PWM_PHASE_CORRECT = 0x40```
	```}EN_TimerMode_t;```
		ATmega32 Timer Modes

typedef enum	```typedef enum EN_ClockSelection_t```
	```{```
	``` NoClock = 0xF8, // stops clock - no clock is set```
	``` NoPrescaling = 0x01, // clock matches internal clock```
	```with no prescaling```
	``` Prescale8 = 0x02, // prescale clock /8```
	``` Prescale64 = 0x03, // prescale clock /64```
	``` Prescale256 = 0x04, // prescale clock /256```
	``` Prescale1024 = 0x05, // prescale clock /1024```
	
	``` /* External Clock on T0 pin with falling edge */```
	``` ExternalFallingEdge = 0x06,```
	
	``` /* External Clock on T0 pin with rising edge */```
	``` ExternalRisingEdge = 0x07,```
	```}EN_ClockSelection_t; // 1 byte```
		ATmega32 Timer/Counter clock selections

typdef enum	```typedef enum EN_timerError_t```
	```{```
	``` OK,```
	``` Error```
	```}EN_timerError_t;```
		Error return type for timer driver

Timer Functions:

```c
/**
 * Initialize timer-0 with given operating mode
 *  and automatically calculates init start value and overflow count to achieve the
 *  desiredDelayMs (ms)
 * @param operatingMode one of (Normal, CTC, FAST_PWM, PWM_PHASE_CORRECT)
 * @param desiredDelayMs desired delay in milliseconds
 */
EN_timerError_t timer_init(EN_TimerMode_t operatingMode);
```

```c
/**
 * blocks for a given delay time before returning
 * @param desiredDelay [in] (ms) delay to wait for - Range 2ms upto 60 seconds, sensitivity ~1.2 ms
 * @return Error if any
 */
void timer_delay(uint16_t desiredDelay);
```

```c
/**
 * sets given Prescaler for timer0
 * @param enClockPrescaleSelection [in] prescaler enum
 * @return
 */
static EN_timerError_t timer_start(EN_ClockSelection_t enClockPrescaleSelection);
```

```c
/*
*//**
 * Manually set initial timer start value
 * @param timerInitValue [in] value (0 -> 255)
 */
static void timer_setTimerValue(uint8_t timerInitValue);
```

```c
/**
 * Resets the timer
 */
void timer_reset();
```

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
│   ├───Visio Diagrams
│   │           *.visio
│   ├───LED Sequence V3.0 - Design.pdf
│
├───ECUAL
│   ├───Button Driver
│   │       button.c
│   │       button.h
│   │
│   └───LED Driver
│           led.c
│           led.h
│
├───MCAL
│   │   registers.h
│   │
│   ├───DIO Driver
│   │       dio.c
│   │       dio.h
│   │
│   ├───Timer Driver
│   │       timer.c
│   │       timer.h
│   │
│   └───EXI Driver
│           interrupts.c
│           interrupts.h
│
└───Proteus
        Proteus_LED_Sequence_V3.0.pdsprj
```
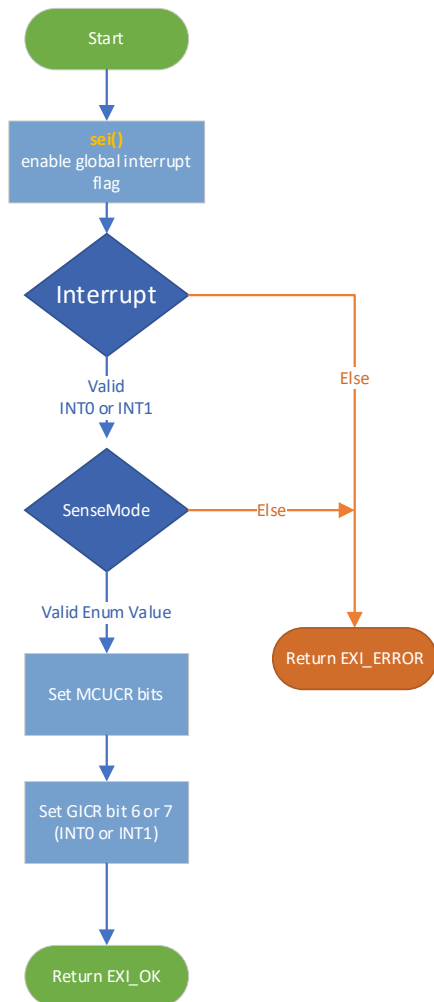
Project Modules APIs Charts

EXI Flowcharts

EXI Driver

EXI_enableInterrupt(EN_EXI_INT_t interrupt, EN_EXI_SENSE_t interruptSenseMode);

Enable Interrupt function

Start

sei()
enable global interrupt flag

Interrupt

— Else → Return EXI_ERROR

Valid
INT0 or INT1

SenseMode — Else →

Valid Enum Value

Set MCUCR bits

Set GICR bit 6 or 7
(INT0 or INT1)

Return EXI_OK

EXI Driver

EXI_disableInterrupt(EN_EXI_INT_t interrupt)

Disable Interrupt function

Start

Interrupt — Else → Return EXI_ERROR

Valid
INT0 or INT1

Clear respective
GICR bit
INT0 -> bit6
INT1 -> bit7

Return EXI_OK
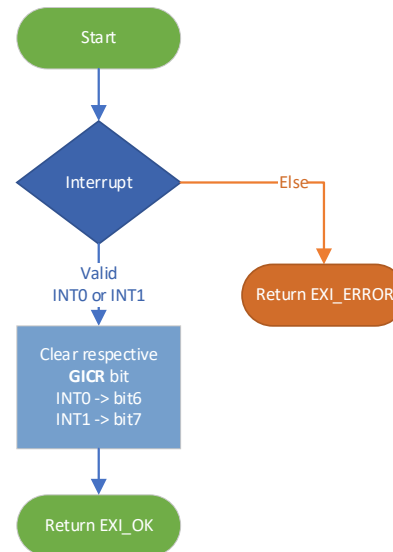
EXI Driver

void EXI_disableAll(void)

Disable All Global Interrupt

Start

cli()
clears global interrupt flag

Clear GICR bit 6 & 7
(INT0 or INT1)

STOP
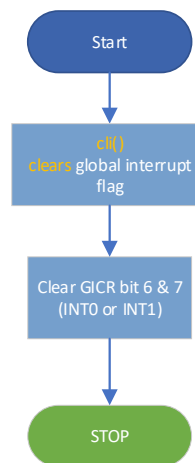
Button API Flowcharts

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
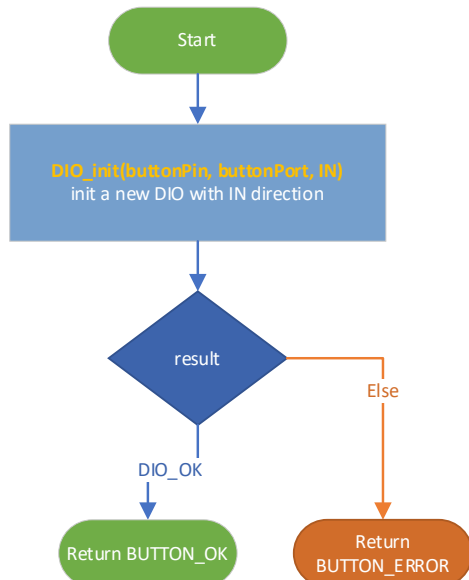
Button Driver

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 ──▶ Return LED_ERROR
  │
 DIO_OK
  │
  ▼
Return LED_OK
```
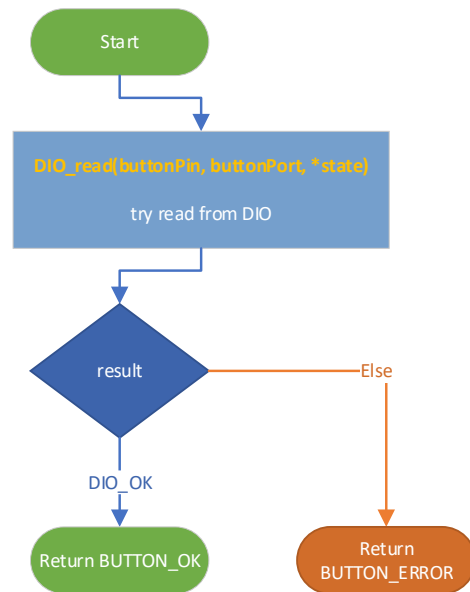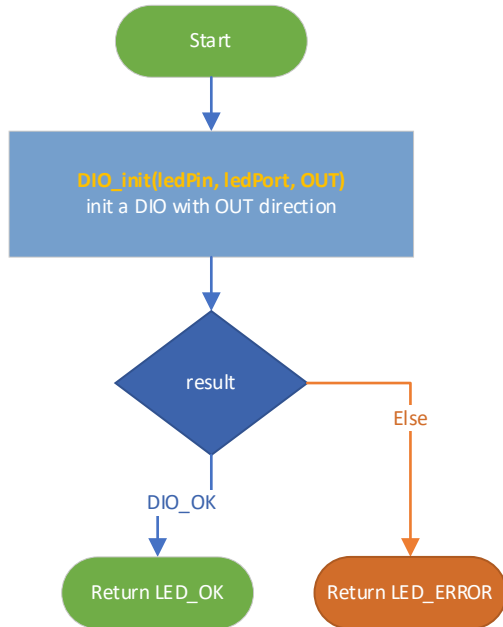
LED Driver

EN_LED_ERROR_t LED_on(EN_DIO_PORT_T ledPort, uint8_t ledPin);

LED ON FUNCTION
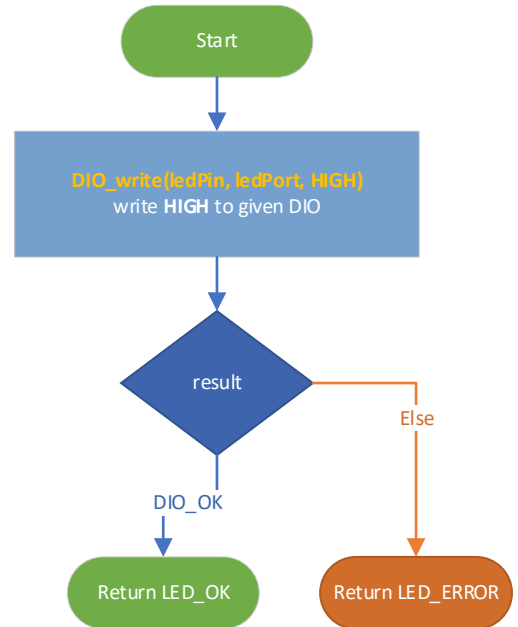
```
Start
  │
  ▼
DIO_write(ledPin, ledPort, HIGH)
write HIGH to given DIO
  │
  ▼
result ──── Else ──▶ Return LED_ERROR
  │
 DIO_OK
  │
  ▼
Return LED_OK
```

LED Driver

EN_LED_ERROR_t LED_off(EN_DIO_PORT_T ledPort, uint8_t ledPin);

LED OFF FUNTION

```
Start
  │
  ▼
DIO_write(ledPin, ledPort, LOW)
write LOW to given DIO
  │
  ▼
result ──── Else ──▶ Return LED_ERROR
  │
 DIO_OK
  │
  ▼
Return LED_OK
```
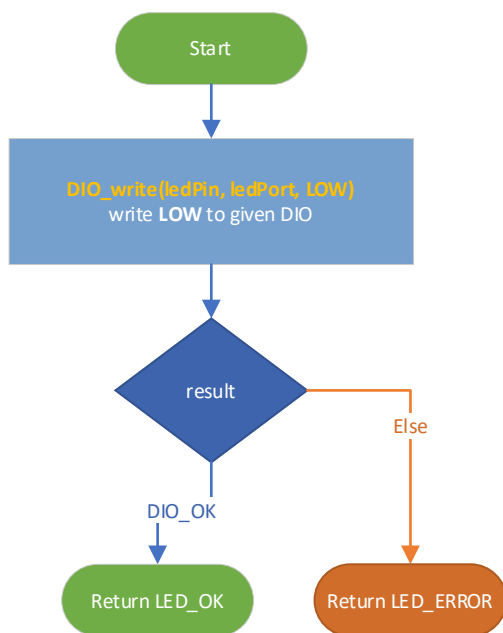
LED Driver

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 ──▶ Return LED_ERROR
  │
 DIO_OK
  │
  ▼
Return LED_OK
```
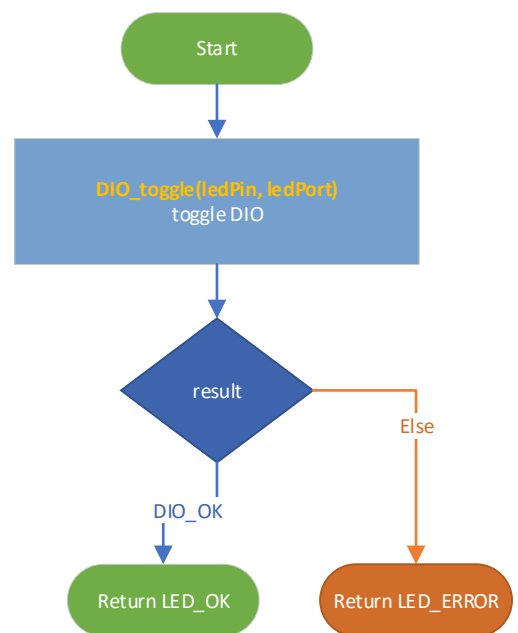
LED Driver

EN_LED_ERROR_t LED_array_on(EN_DIO_PORT_T ledPort, uint8_t mask)

Led Array ON Function

Start

DIO_port_write(ledPort, 0xFF, mask)
write **0xFF** to given DIO PORT passing over mask

result → Else

DIO_OK

Return LED_OK

Return LED_ERROR
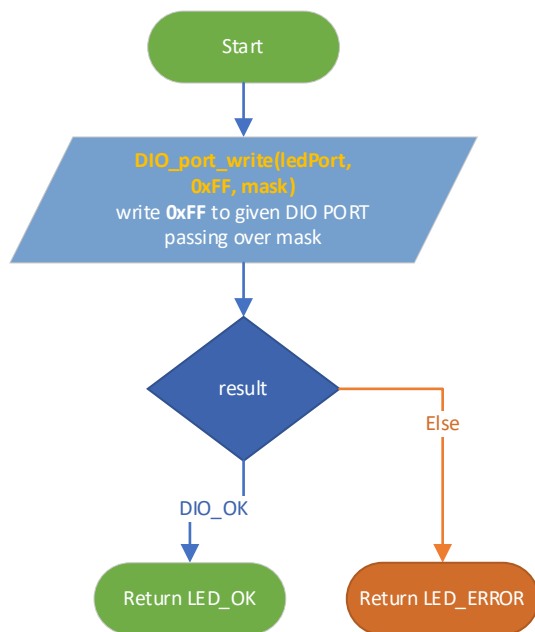
LED Driver

EN_LED_ERROR_t LED_array_off(EN_DIO_PORT_T ledPort, uint8_t mask)

Led Array OFF Function
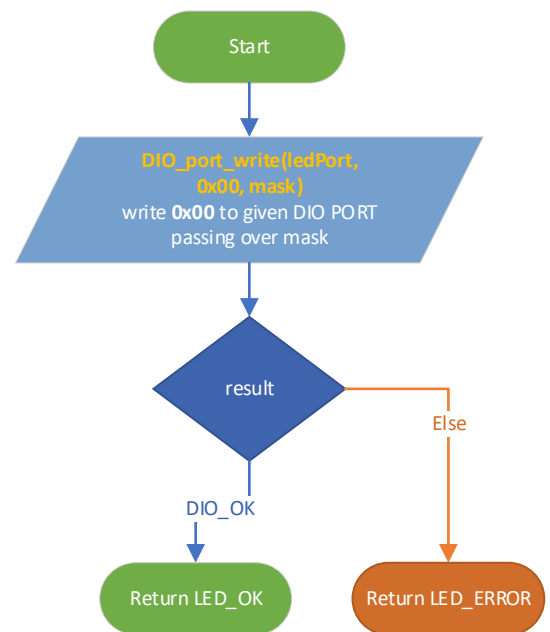
Start

DIO_port_write(ledPort, 0x00, mask)
write **0x00** to given DIO PORT passing over mask

result → Else

DIO_OK

Return LED_OK

Return LED_ERROR

LED Driver

EN_LED_ERROR_t LED_array_toggle(EN_DIO_PORT_T ledPort, uint8_t mask)

LED Array Toggle Function

Start

DIO_port_write(ledPort, 0xFF, mask)
write **0xFF** to given DIO PORT passing over mask

result → Else

DIO_OK

Return LED_OK

Return LED_ERROR
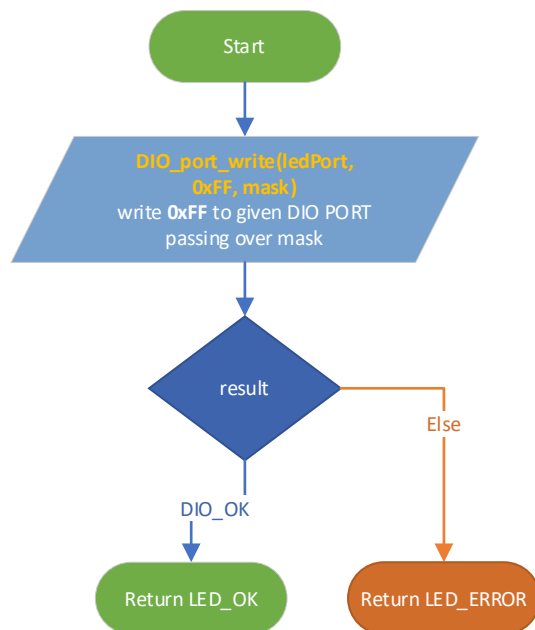
LED Driver

void LED_array_blink(EN_DIO_PORT_T ledPort, uint16_t onTime, uint16_t offTime, uint8_t mask)

LED Array Toggle Function

Start

LED_array_on(ledPort, mask);

timer_delay(onTime);

LED_array_off(ledPort, mask);

timer_delay(offTime);

Stop

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

Start

Recurring check(1)

pinNumber in 0 .. 7

portNumber in A .. D → No

Direction in IN, OUT → No

No

Return DIO_ERROR

IN: BitSet
OUT: BitClear → Return LED_OK

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 → FAIL

Write Bit value to desired pin/ port

Return LED_OK

Return LED_ERROR

DIO Driver

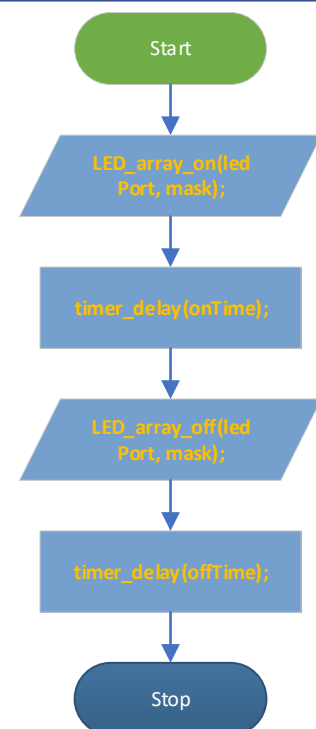EN_DIO_Error_T DIO_read(uint8_t pinNumber, EN_DIO_PORT_T portNumber, uint8_t *value);

DIO READ FUNCTION

Start

Check (1) → FAIL

READ BIT AND STORE IT'S VALUE IN `*value`

Return LED_OK

Return LED_ERROR

DIO Driver

EN_DIO_Error_T DIO_toggle(uint8_t pinNumber, EN_DIO_PORT_T portNumber);

DIO TOGGLE FUNCTION

Start

DIO_read() → DIO_ERROR

Flip read value

DIO_write(~) → DIO_ERROR → DIO_ERROR

DIO_OK

Return LED_OK

Return LED_ERROR

DIO Driver

EN_DIO_Error_T DIO_port_write(EN_DIO_PORT_T portNumber, uint8_t byte, uint8_t mask)

Start

mask == 0 — true → mask = 0xFF

False

Switch `portNumber`

Case `A` → PORT_A = ((NOT(mask)) AND PORT_A) OR (mask AND byte)

Case `B` → PORT_B = ((NOT(mask)) AND PORT_B) OR (mask AND byte)

Case `C` → PORT_C = ((NOT(mask)) AND PORT_C) OR (mask AND byte)

Case `D` → PORT_D = ((NOT(mask)) AND PORT_D) OR (mask AND byte)

Else → RETURN DIO_Error

RETURN DIO_OK
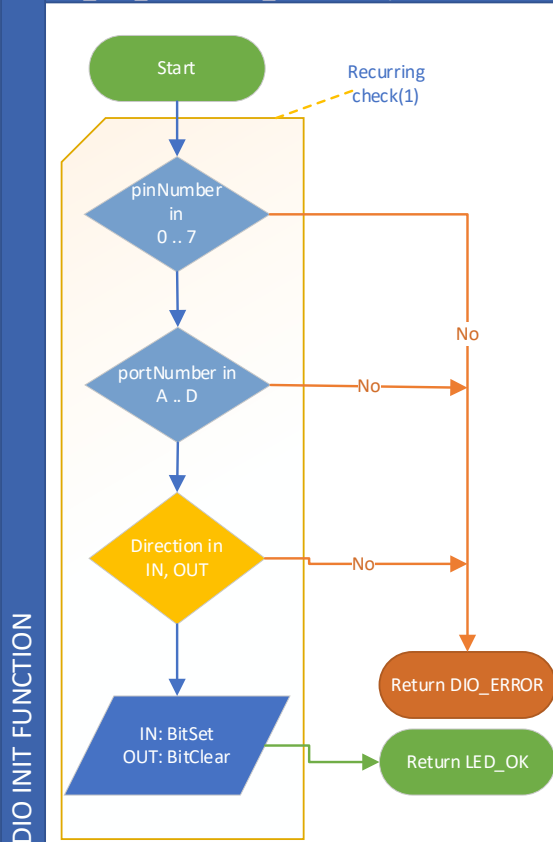
DIO Port Write Function

DIO Driver

EN_DIO_Error_T DIO_port_toggle(EN_DIO_PORT_T portNumber, uint8_t mask)

Start

mask == 0 — true → mask = 0xFF

False

Switch `portNumber`

Case `A` → PORT_A = PORT_A XOR `mask`

Case `B` → PORT_D = PORT_D XOR `mask`

Case `C` → PORT_C = PORT_C XOR `mask`

Case `D` → PORT_D = PORT_D XOR `mask`

Else → RETURN DIO_Error

RETURN DIO_OK
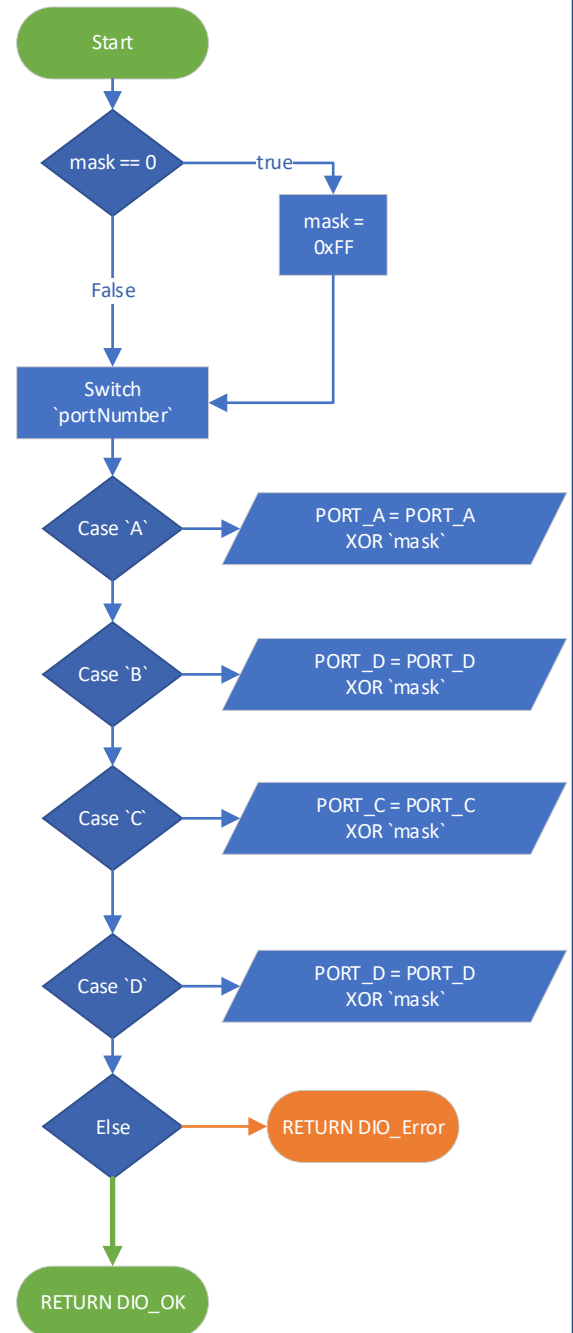
DIO Port Write Function

Timer API Flowcharts

Timer Driver

void timer_delay(uint16_t desiredDelay)

Timer Delay Function

```
Start
  │
  ▼
• Calculatations for Desired Delay
• Overflow Count
• Timer Initial Value
  │
  ▼
SET Timer Initial Value
`timer_setTimerValue()`
  │
  ▼
SET timer prescaler and start timer
`timer_start(prescaler)`
  │
  ▼
set
CurrentOverflow = 0
  │
  ▼
TIFROV0 == 0 ?  ──Yes──┐ (loop back)
  │ No
  ▼
CurrentOverflow ++
  │
  ▼
CurrentOverflow <
OverflowCount  ──Yes──→ (loop back to TIFROV0)
  │
  ▼
Reset Timer
`timer_reset()`
  │
  ▼
END
```

Timer Driver

void timer_reset()

Timer Reset

```
Start                              STOP
  │                                 ▲
  ▼                                 │
CLEAR      →    CLEAR      →     CLEAR
TCCR0           TIFOV0           TCNT0
```

Timer Driver

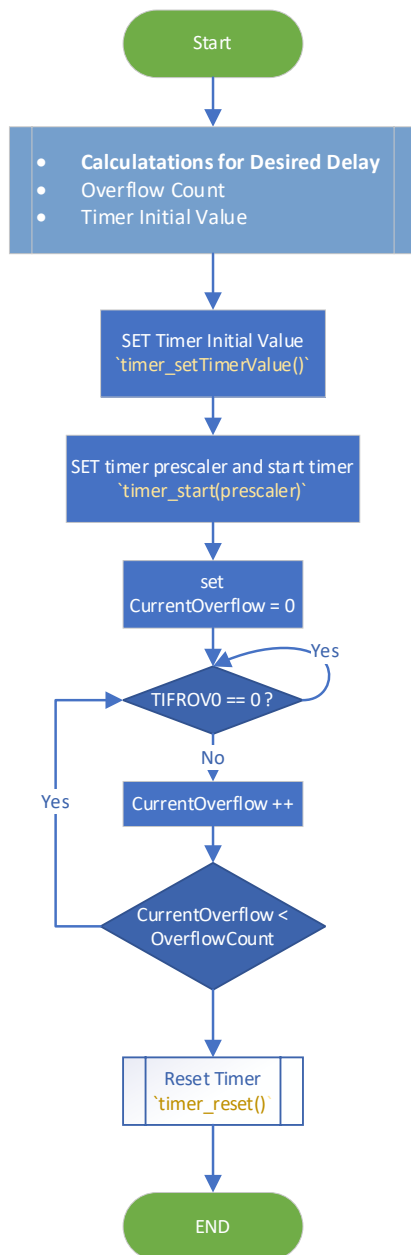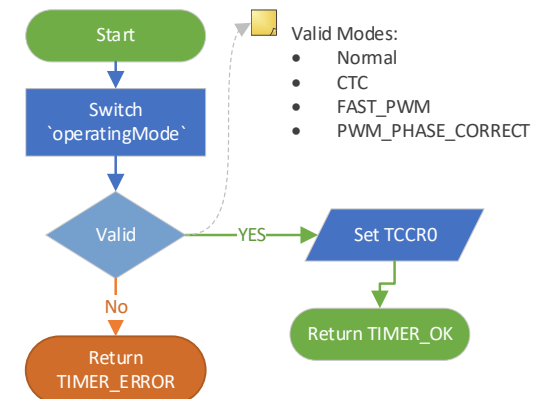### EN_DIO_Error_T DIO_init(uint8_t pinNumber, EN_DIO_PORT_T portNumber, EN_DIO_DIRECTION_T direction);

TIMER INIT FUNCTION

Valid Modes:
- Normal
- CTC
- FAST_PWM
- PWM_PHASE_CORRECT

```
Start
  │
  ▼
Switch
`operatingMode`
  │
  ▼
Valid  ──YES──→ Set TCCR0
  │ No              │
  ▼                 ▼
Return         Return TIMER_OK
TIMER_ERROR
```

Timer Driver

EN_timerError_t timer_start(EN_ClockSelection_t enClockPrescaleSelection)

TIMER START FUNCTION

Valid Clocks:
- NoClock
- NoPrescaling
- Prescale8
- Prescale64
- Prescale256
- Prescale1024
- ExternalFallingEdge
- ExternalRisingEdge

```
Start
  │
  ▼
Switch
`Clock Prescale Selection`
  │
  ▼
Valid  ──Yes──→ SET TCCR0
  │ No              │
  ▼                 ▼
Return         Return TIMER_OK
TIMER_ERROR
```
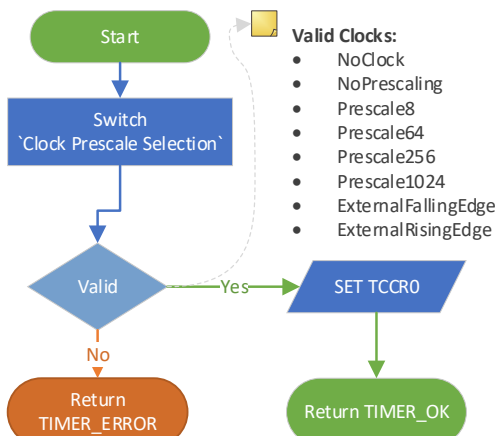
Timer Driver

void timer_setTimerValue(uint8_t timerInitValue)

Timer setValue

```
Start
  │
  ▼
Set TCNT0  ──→  STOP
```
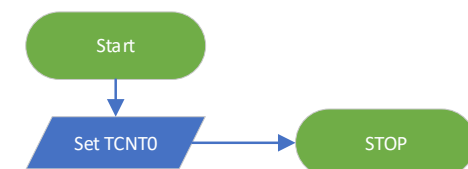
Application API Flowcharts

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   // INT0
#define BUTTON_1_port D
#define BUTTON_1_PIN 2   // INT1

/* Magic Numbers */
#define NUMBER_OF_LED_STATES 7

/* Global Variables */
uint8_t state_number = 0;
uint8_t current_blink_mode_index = 1;

/* App Blink Modes */
ST_APP_BLINK_MODES_t appBlinkModes;
ST_APP_BLINK_MODE_t currentBlinkMode;
```
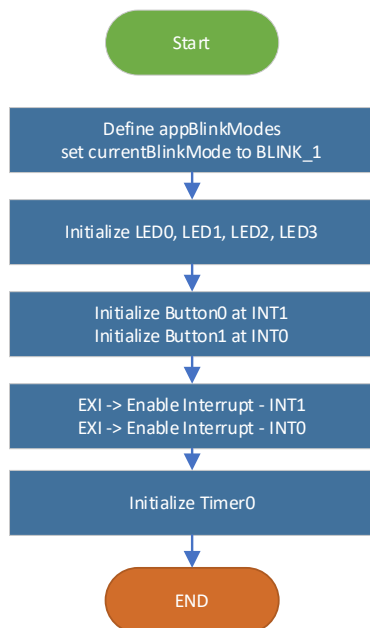
Application

void App_init();

Start

Define appBlinkModes
set currentBlinkMode to BLINK_1

↓

Initialize LED0, LED1, LED2, LED3

↓

Initialize Button0 at INT1
Initialize Button1 at INT0

↓

EXI -> Enable Interrupt - INT1
EXI -> Enable Interrupt - INT0

↓

Initialize Timer0

↓

END

Application

void App_Start();

Start

↓

switch
state_number

↓

| Case 0 | → | currentOnLEDs = 0 |

| Case 1 | → | currentOnLEDs = 0x01 |

| Case 2 | → | currentOnLEDs = 0x03 |

| Case 3 | → | currentOnLEDs = 0x07 |

| Case 4 | → | currentOnLEDs = 0x0F |

| Case 5 | → | currentOnLEDs = 0x0E |

| Case 6 | → | currentOnLEDs = 0x0C |

| Case 7 | → | currentOnLEDs = 0x08 |

| Else | → | State_number = 0
currenOnLEDs = 0 |

↓

currentOnLEDs != 0 ?

Yes →

LED ARRAY
BLINK(currentOnLEDs)

No →

LED ARRAY OFF
ALL LEDs

Application

ISR(EXT_INT_0)

```
Start
  │
  ▼
Switch
`current blink mode index`
  │
  ▼
BLINK 1  ──────►  Set current mode -> BLINK_2
  │                index ++
  ▼
BLINK 2  ──────►  Set current mode -> BLINK_3
  │                index ++
  ▼
BLINK 3  ──────►  Set current mode -> BLINK_4
  │                index ++
  ▼
BLINK 4  ──────►  Set current mode -> BLINK_5
  │                index ++
  ▼
BLINK 5  ──────►  Set current mode -> BLINK_1
  │                index = 1
  ▼
ELSE/DEF  ──────►                    END
```

ISR function for INT0

Application

ISR(EXT_INT_1)

```
              Start
                │
                ▼
        State_number <  ── No ──►  State_number = 0
      NO_OF_LED_STATES                   │
                │                        │
               Yes                       │
                │                        │
                ▼                        ▼
        State_number++  ──────────►    END
```

ISR function for INT1