LED Sequence V3.0

Khaled Mustafa Anwar

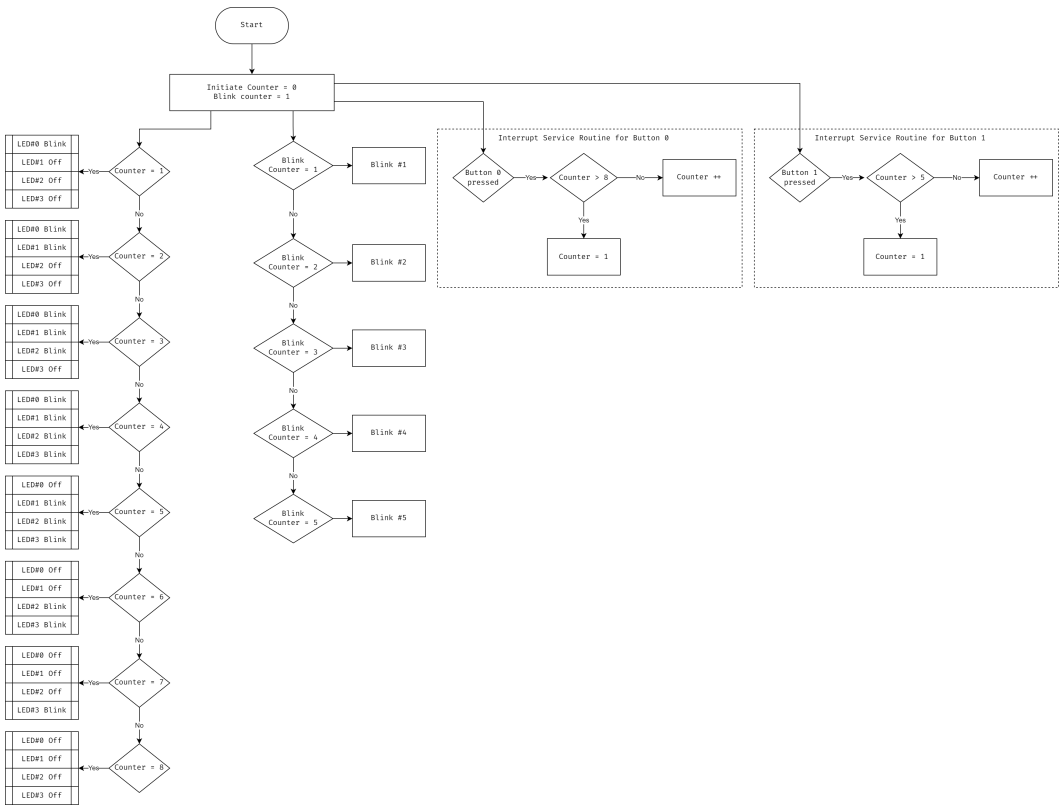
April 13, 2023

1 Introduction

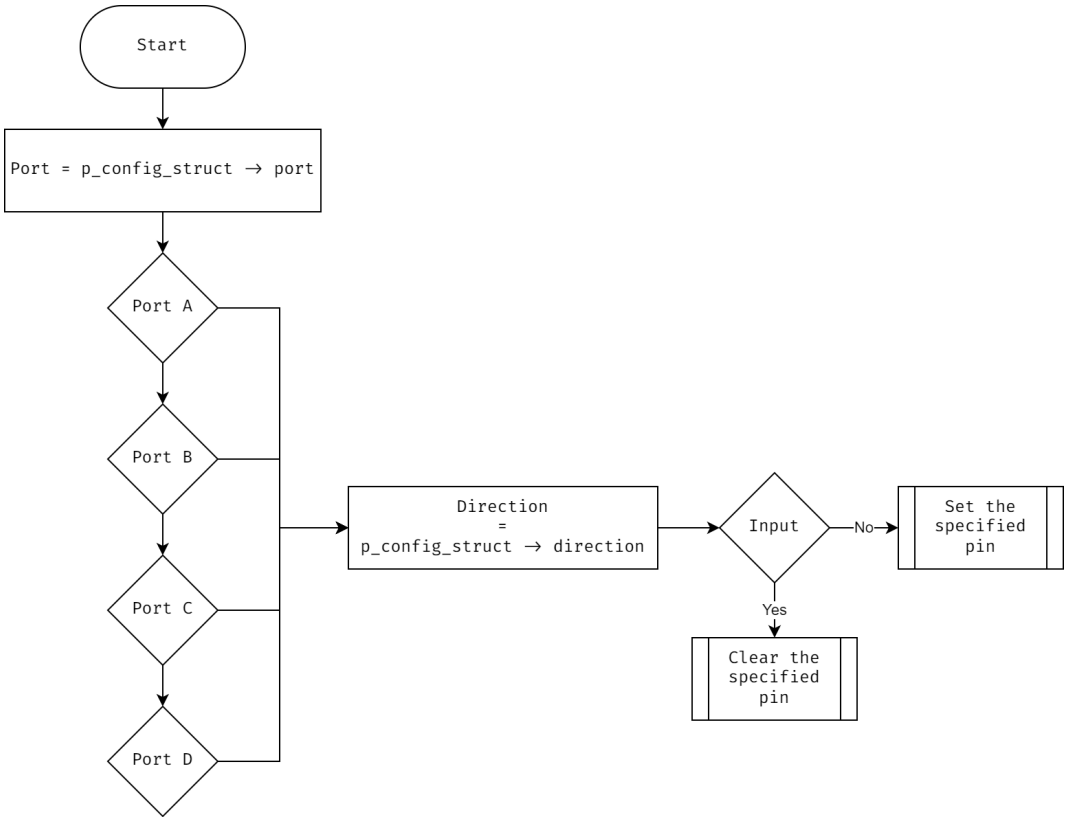
This task controls the LED lighting sequence according to button pressing, and alternates between 5 different blinking states.

2 Flowchart

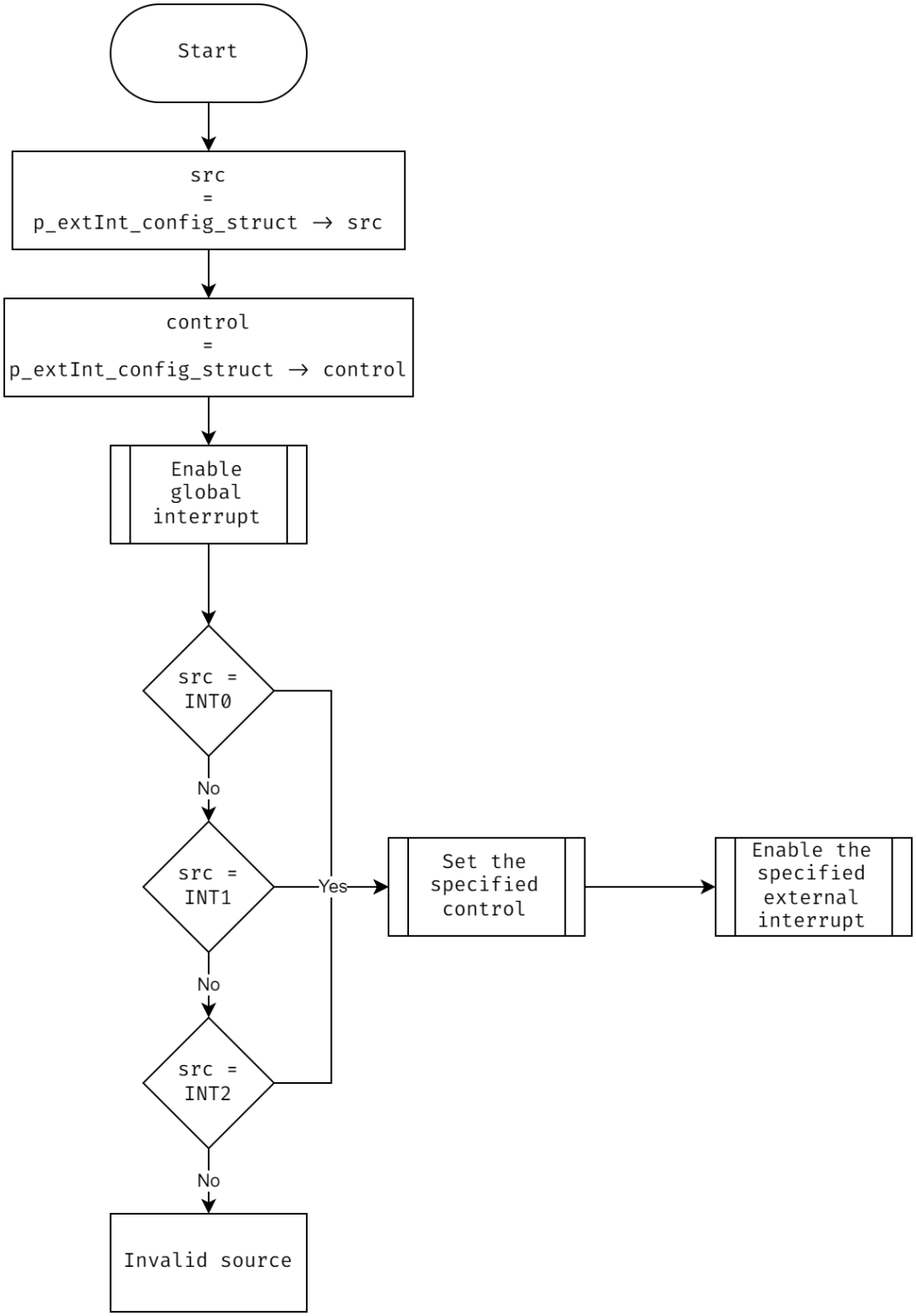
2.0.1 LED Sequence Application



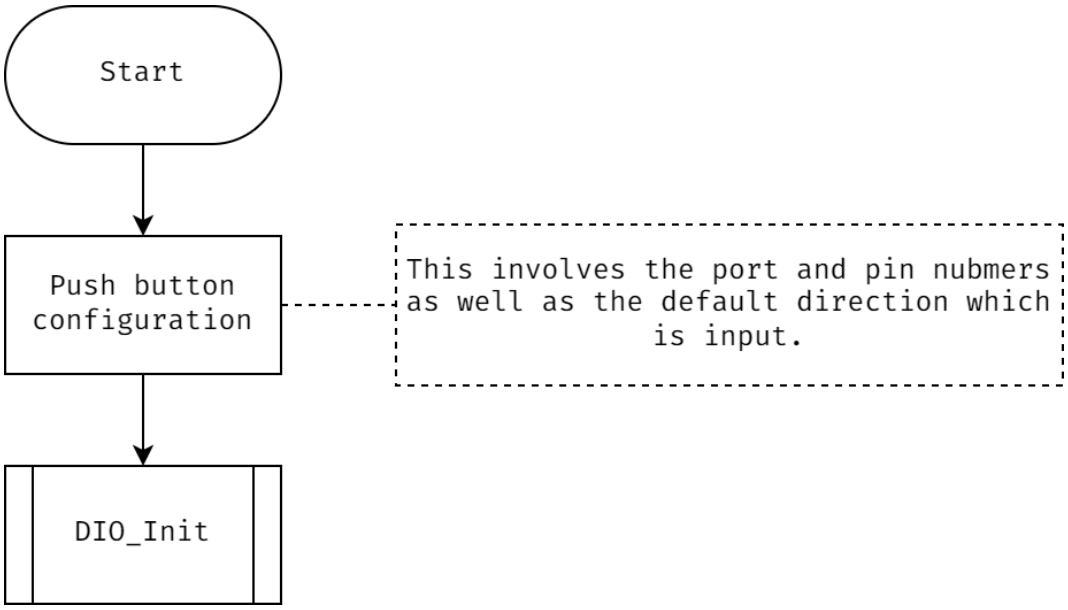
2.0.2 DIO flowchart



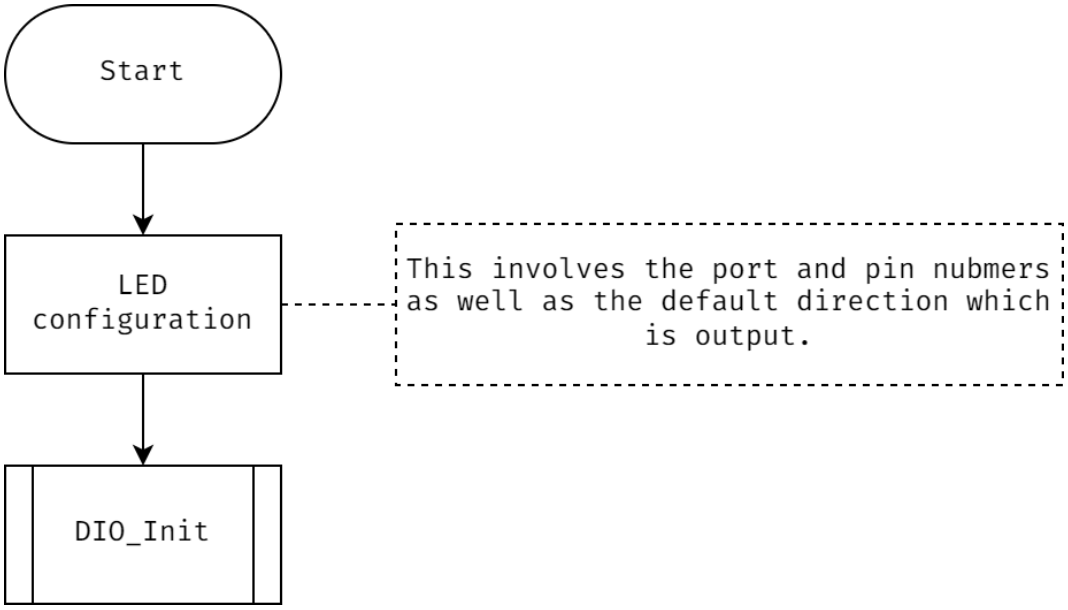
2.0.3 External interrupt flowchart



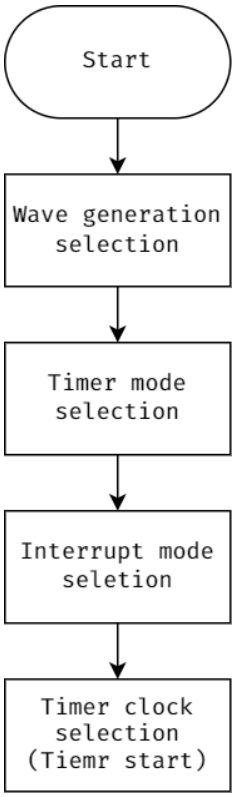
2.0.4 Push button flowchart



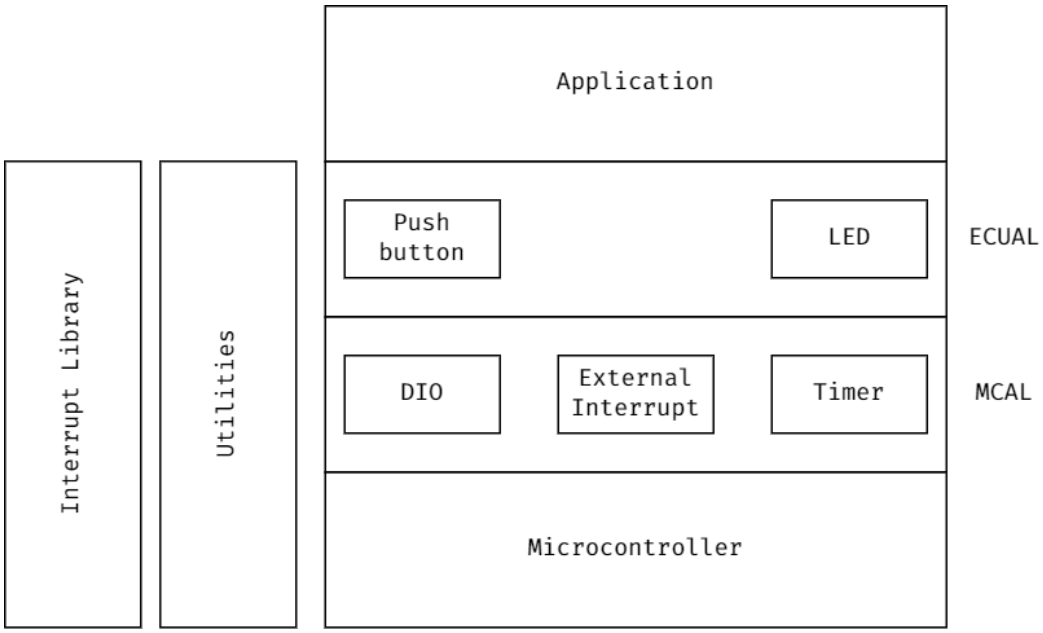
2.0.5 LED flowchart



2.0.6 Timer initialization flowchart



3 Layered architecture



4 Application Binary Interface (API)

4.1 Microcontroller Architecture Layer (MCAL)

4.1.1 DIO

```c
/**
* @enum EN_DIO_ERROR_STATE
* @brief Defines the state of DIO functions.
*/
typedef enum EN_DIO_ERROR_STATE {
  DIO_SUCCESS = 0, DIO_PORT_INVALID, DIO_DIRECTION_INVALID, DIO_PIN_INVALID
}EN_DIO_ERROR_STATE;

/**
* @enum EN_DIO_DIRECTION
* @brief Specifies the state of the pin.
*/
typedef enum EN_DIO_DIRECTION {
  DIO_INPUT = 0, DIO_OUTPUT
}EN_DIO_DIRECTION;

/**
* @enum EN_DIO_PIN
* @brief Specifies the number of pin.
*/
typedef enum EN_DIO_PIN {
  PIN0 = 0, PIN1, PIN2, PIN3, PIN4, PIN5, PIN6, PIN7, PIN8
}EN_DIO_PIN;

/**
* @enum EN_DIO_PORT
* @brief Specifies the port number.
* the port number and returns the address of the corresponding port.
*/
typedef enum EN_DIO_PORT {
  PORT_A = 0, PORT_B, PORT_C, PORT_D
}EN_DIO_PORT;

/**
* @enum EN_DIO_LEVEL
* @brief Specifies the level of the pin.
*/
typedef enum EN_DIO_LEVEL {
  DIO_LOW = 0, DIO_HIGH
}EN_DIO_LEVEL;

/**
* @struct DIO_Init_t
* @brief Holds the configuration of a specific pin of a port.
* @var DIO_Init_t::port
* Member 'port' sets the port to be configured.
* @var DIO_Init_t::pin
* Member 'pin' sets the pin to be configured.
* @var DIO_Init_t::direction
* Member 'direction' sets the direction of the pin.
```

```
 * @var DIO_Init_t::pin_value
 * Member 'pin_value; contains the value of the pin when it's configured as input
   mode.
 * @var DIO_Init_t::port_value
 * Member 'port_value' contains the value to be written to the port register if the
   pin is configured as output.
 */
typedef struct DIO_Init_t {
  EN_DIO_PORT port;
  EN_DIO_PIN pin;
  EN_DIO_DIRECTION direction;
  union {
    uint8 pin_value;
    uint8 port_value;
  };
}DIO_Init_t;

/**
* @brief Initializes the direction of the specified pin.
* @param[in] p_config_struct Address of the configuration structure.
* @return DIO_PORT_INVALID Port in invalid.
* @return DIO_SUCCESS The pin initialization is a success.
*/
EN_DIO_ERROR_STATE DIO_Init(DIO_Init_t *p_config_struct);


```

```
75
76  /**
77  * @brief Reads the state of a specific pin.
78  * @param[in] p_config_struct Address of the configuration structure.
79  * @return DIO_PORT_INVALID Port is invalid.
80  * @return DIO_DIRECTION_INVALID Reading from a pin that is configured as output.
81  * @return DIO_SUCCESS The read operation is a success.
82  */
83  EN_DIO_ERROR_STATE DIO_ReadPin(DIO_Init_t *p_config_struct);
84
85  /**
86  * @brief Write a specific level to a specified pin.
87  * @param[in] p_config_struct Address of the configuration structure.
88  * @return DIO_PORT_INVALID Port is invalid.
89  * @return DIO_DIRECTION_INVALID Writing to a pin that is configured as input.
90  * @return DIO_SUCCESS The write operation is a success.
91  */
92  EN_DIO_ERROR_STATE DIO_WritePin(DIO_Init_t *p_config_struct);
93
94  /**
95  * @brief Toggles the current level of a pin.
96  * @param[in] p_config_struct Address of the configuration structure.
97  *  @return DIO_PORT_INVALID Port is invalid.
98  * @return DIO_DIRECTION_INVALID Toggle a pin that is configured as input.
99  * @return DIO_SUCCESS The toggle operation is a success.
100 */
101 EN_DIO_ERROR_STATE DIO_TogglePin(DIO_Init_t *p_config_struct);
```

4.1.2 External Interrupts

```
1   /**
2   * @enum EN_INT_ERROR_STATE
3   * @brief Specifies the state of DIO functions.
4   */
5   typedef enum EN_INT_ERROR_STATE {
6     INT_SUCCESS = 0, INT_GLOBAL_INT_NOT_SET, INT_INVALID_CONTROL,
      INT_INVALID_EXTERNAL_SRC
7   }EN_INT_ERROR_STATE;
8
9   /**
10  * @enum EN_EXT_INT_SENSE_CONTROL
11  * @brief Specifies the triggering mechanism for the external interrupts.
12  */
13  typedef enum EN_EXT_INT_SENSE_CONTROL {
14    LOW_LEVEL = 0, ANY_LOGIC_CHANGE, FALLING_EDGE, RISING_EDGE
15  }EN_EXT_INT_SENSE_CONTROL;
16
17  /**
18  * @enum EN_EXT_INTERRUPT_SRC
19  * @brief Specifies the external interrupts source.
20  */
21  typedef enum EN_EXT_INTERRUPT_SRC {
22    INT0 = 0, INT1, INT2
23  }EN_EXT_INTERRUPT_SRC;
24
```

```
/**
* @enum EN_EXT_INTERRUPT_BITS
* @brief Control sense configuration bits.
*/
typedef enum EN_EXT_INTERRUPT_BITS {
  ISC00 = 0, ISC01, ISC10, ISC11, ISC2 = 6
}EN_EXT_INTERRUPT_BITS;

/**
* @struct INT_ExtInit_t
* @brief Holds the configuration of external interrupts.
* @var INT_ExtInit_t::src
* Member 'src' specifies the external source of the interrupt.
* @var INT_ExtInit_t::control
* Member 'control' specifies how the external pin gets triggered.
*/
typedef struct INT_ExtInit_t {
  EN_EXT_INTERRUPT_SRC src;
  EN_EXT_INT_SENSE_CONTROL control;
}INT_ExtInit_t;

/**
* @brief Enables and sets the sense control of the external interrupt.
* @param ext_int_config_struct
* @return INT_INVALID_EXTERNAL_SRC
* @return INT_SUCCESS
*/
EN_INT_ERROR_STATE INT_ExtInterruptInit(INT_ExtInit_t *ext_int_config_struct);

/**
* @brief Enables external interrupt 0.
* @return INT_GLOBAL_INT_NOT_SET
* @return INT_SUCCESS
*/
EN_INT_ERROR_STATE INT_EnableINT0();

/**
* @brief Enables external interrupt 1.
* @return INT_GLOBAL_INT_NOT_SET
* @return INT_SUCCESS
*/
EN_INT_ERROR_STATE INT_EnableINT1();

/**
* @brief Enables external interrupt 2.
* @return INT_GLOBAL_INT_NOT_SET
* @return INT_SUCCESS
*/
EN_INT_ERROR_STATE INT_EnableINT2();


```

```
76
77  /**
78  * @brief Controls the triggering mechanism of INT1 and INT0.
79  * Interrupt 0 sense control.
80  * | ISC01 Bit 3 | ISC00 Bit 2 | Description                       |
81  * |-------------|-------------|-----------------------------------|
82  * | 0           | 0           | INT0 triggered on low level.      |
83  * | 0           | 1           | Any logic change triggers.        |
84  * | 1           | 0           | Falling edge generates interrupt. |
85  * | 1           | 1           | Rising edge generates interrupt.  |
86  * Interrupt 1 sense control
87  * | ISC11 Bit 3 | ISC10 Bit 2 | Description                       |
88  * |-------------|-------------|-----------------------------------|
89  * | 0           | 0           | INT1 triggered on low level.      |
90  * | 0           | 1           | Any logic change triggers.        |
91  * | 1           | 0           | Falling edge generates interrupt. |
92  * | 1           | 1           | Rising edge generates interrupt.  |
93  * Interrupt 2 sense control
94  * 0 - Falling edge activates the interrupt.
95  * 1 - Rising edge activates the interrupt.
96  */
97  EN_INT_ERROR_STATE INT_ExtIntSenseControl(EN_EXT_INTERRUPT_SRC src,
      EN_EXT_INT_SENSE_CONTROL control);
```

4.1.3 Timer

```
1   /**
2   * @enum TIMER0_WaveFormGeneration
3   * @brief Specifies the PWM mode of operation for the timer.
4   */
5   typedef enum {
6     NORMAL_WG, PWM_WG, CTC_WG, FAST_PWM_WG
7   }TIMER0_WaveFormGeneration;
8
9   /**
10  * @enum TIMER0_ClockSelect
11  * @brief Specifies the clock source to feed the timer.
12  */
13  typedef enum {
14    NO_CLK, F_CPU_CLK, F_CPU_8, F_CPU_64, F_CPU_256, F_CPU_1024
15  }TIMER0_ClockSelect;
16
17  /**
18  * @enum TIMER0_InterruptMode
19  * @brief Specifies the interrupt mode for the timer.
20  */
21  typedef enum {
22    INTERRUPT_DISABLED, INTERRUPT_OVERFLOW_ENABLED, INTERRUPT_COMPARE_MATCH_ENABLED,
      INTERRUPT_ENABLED
23  }TIMER0_InterruptMode;
24
25  /**
26  * @enum TIMER0_Mode
27  * @brief Specifies the mode of operation for Timer_0.
28  */
```

```c
typedef enum {
  TIMER0_OVERFLOW, TIMER0_COMPARE_TOGGLE, TIMER0_COMPARE_CLEAR, TIMER0_COMPARE_SET
}TIMER0_Mode;

/**
* @enum TIMER0_Status
* @brief Specifies the state of the timer.
*/
typedef enum {
  TIMER0_STATUS_OVERFLOW, TIMER0_STATUS_COMPARE_MATCH, TIMER0_ERROR
}TIMER0_Status;

/**
* @struct TIMER0_Config
* @brief Holds the user's configuration for Timer_0.
* @var TIMER0_Config::timerMode
* Member 'timerMode' specifies the mode of operation for the timer.
* @var TIMER0_Config::timerClock
* Member 'timerClcok' specifies the clock source of the timer.
* @var TIMER0_Config::timerWaveGeneration
* Member 'timerWaveGeneration' specifies the
* @var TIMER0_Config::TIMER0_InitValue
* Member 'TIMER0_InitValue' specifies the initial loaded value of the timer.
* @var TIMER0_Config::TIMER0_CompareValue
* Member 'TIMER0_CompareValue' specifies the compare value which the timer will
  compare to.
* @var TIMER0_Config::InterruptMode
* Member 'InterruptMode' specifies the interrupt mode of the timer.
*/
typedef struct {
  TIMER0_Mode timerMode;
  TIMER0_ClockSelect timerClock;
  TIMER0_WaveFormGeneration timerWaveGeneration;
  uint8 TIMER0_InitValue;
  uint8 TIMER0_CompareValue;
  TIMER0_InterruptMode InterruptMode;
}TIMER0_Config;

/**
* @brief Initialize Timer0 with the user configuration and starts the timer.
* @param TIMER0_UserConfig
*/
void Timer0_Init(TIMER0_Config *TIMER0_UserConfig);

/**
* @brief Disables Timer0 by disengage of the clock source.
*/
void Timer0_DeInit(void);

/**
* @brief Sets an initial value for Timer0 to start counting from.
* @param Timer0_InitValue The value to be loaded to Timer0.
*/
void Timer0_SetValue(uint8 Timer0_InitValue);

/**
```

```
 * @brief Gets the overflow and compare status of Timer0.
 * @return TIMER0_STATUS_OVERFLOW
 * @return TIMER0_STATUS_COMPARE_MATCH
 */
TIMER0_Status Timer0_GetStatus(void);

/**
 * @brief Sets a delay assuming that the prescaler value is 1024.
 * @param delay_ms The amount of time is mSec.
 */
void Timer0_SetDelay(uint32 delay_ms);

/**
 * @brief Sets the call-back function when Timer0 triggers an interrupt.
 * @param p_func Address of the function to be executed when an interrupt is
   triggered.
 */
void Timer0_Int_Callback(void(*p_func)(void));
```

4.2 Electronic Unit Architecture Layer (ECUAL)

4.2.1 LED

```
/**
 * @enum EN_LED_API_STATE
 * @brief Defines the state of LED functions.
 */
typedef enum EN_LED_API_STATE {
  LED_SUCCESS = 0, LED_PORT_INVALID, LED_STATUS_INVALID
}EN_LED_API_STATE;

/**
 * @enum EN_LED_STATUS
 * @brief Defines the LED status.
 */
typedef enum EN_LED_STATUS {
  LED_OFF = 0, LED_ON
}EN_LED_STATUS;


```

```

/**
* @struct LED_Init_t
* @brief Holds the port number and the pin number of the LED.
* @var LED_Init_t::port
* Member 'port' specifies the port number.
* @var LED_Init_t::pin
* Member 'pin' specifies the pin number.
* @var LED_INIT_t::led_status
* Member 'led_status' specifies the status of the LED.
*/
typedef struct LED_Init_t {
  EN_DIO_PORT port;
  EN_DIO_PIN pin;
  EN_LED_STATUS led_status;
}LED_Init_t;

/**
* @brief Initializes the pin attached to the LED.
* @param[in] p_config_struct Address of the configuration structure.
* @return LED_SUCCESS Initialization is done successfully.
*/
EN_LED_API_STATE LED_Init(LED_Init_t *p_led_config_struct);

/**
* @brief Turns the LED on.
* @param[in] p_config_struct Address of the configuration structure.
* @return LED_PORT_INVALID
* @return LED_STATUS_INVALID
* @return LED_SUCCESS
*/
EN_LED_API_STATE LED_On(LED_Init_t *p_led_config_struct);

/**
* @brief Turns the LED off.
* @param[in] p_config_struct Address of the configuration structure.
* @return LED_PORT_INVALID
* @return LED_STATUS_INVALID
* @return LED_SUCCESS
*/
EN_LED_API_STATE LED_Off(LED_Init_t *p_led_config_struct);
```

4.2.2 Push Button

```

/**
* @enum EN_PB_API_STATE
* @brief Specifies the state of the push button.
*/
typedef enum EN_PB_API_STATE {
  PB_SUCCESS = 0, PB_PORT_INVALID, PB_DIRECTION_INVALID
}EN_PB_API_STATE;


```

```c

/**
* @enum EN_PB_LEVEL
* @brief Specifies the state of push button.
*/
typedef enum EN_PB_LEVEL {
  PB_LOW = 0, PB_HIGH
}EN_PB_LEVEL;

/**
* @struct PB_Init_t
* @var PB_Init_t::port
* Member 'port' specifies the port which the push button is connected to.
* @var PB_Init::pin
* Member 'pin' specifies the pin number which the push button is connected to.
*/
typedef struct PB_Init_t {
  EN_DIO_PORT port;
  EN_DIO_PIN pin;
  uint8 pb_status;
}PB_Init_t;

/**
* @brief Initializes the state of the pin connected to the push button.
* @param[in] p_config_struct Address of the configuration structure.
*/
EN_PB_API_STATE PB_Init(PB_Init_t *p_pb_config_struct);

/**
* @brief Reads the current state of the push button.
* @param[in/out] p_config_struct Address of the configuration structure.
* @return PB_PORT_INVALID The selected port doesn't corresponds to the MCU ports.
* @return PB_SUCCESS
*/
EN_PB_API_STATE PB_ReadState(PB_Init_t *p_pb_config_struct);
```