LED Sequence V2.0

Khaled Mustafa Anwar

April 10, 2023

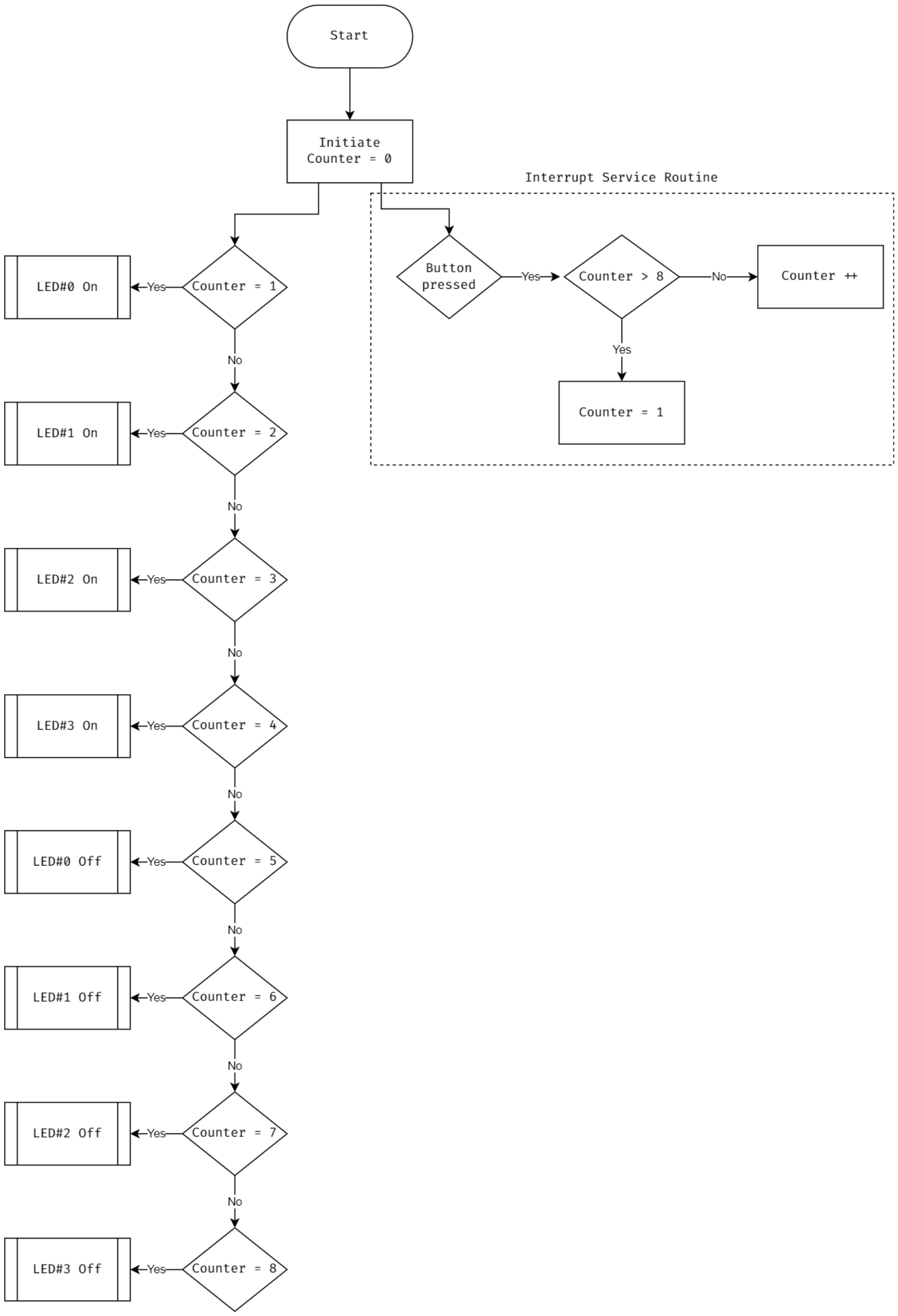
1 Introduction

This task controls the LED lighting sequence according to button pressing, and this is achieved via external interrupts. Such that the sequence is as follows:

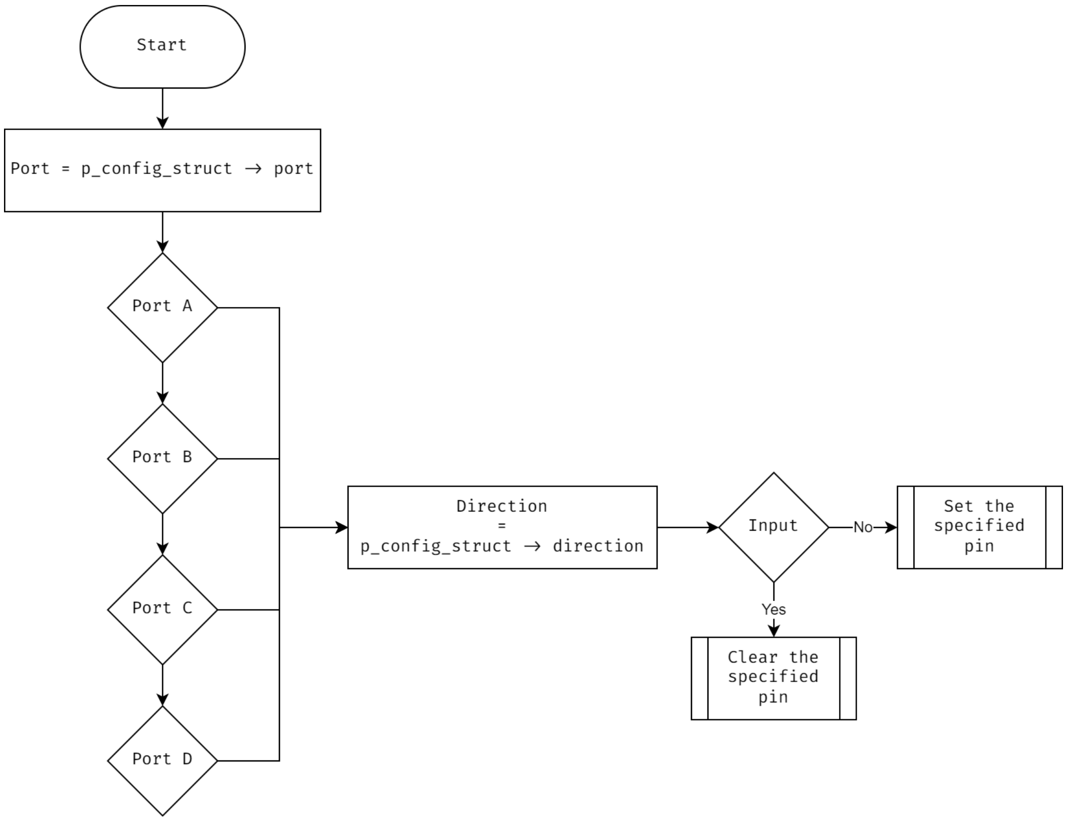
1. Initially (OFF, OFF, OFF, OFF)
2. Press 1 (ON, OFF, OFF, OFF)
3. Press 2 (ON, ON, OFF, OFF)
4. Press 3 (ON, ON, ON, OFF)
5. Press 4 (ON, ON, ON, ON)
6. Press 5 (OFF, ON, ON, ON)
7. Press 6 (OFF, OFF, ON, ON)
8. Press 7 (OFF, OFF, OFF, ON)
9. Press 8 (OFF, OFF, OFF, OFF)
10. Press 9 (ON, OFF, OFF, OFF)

2 Flowchart

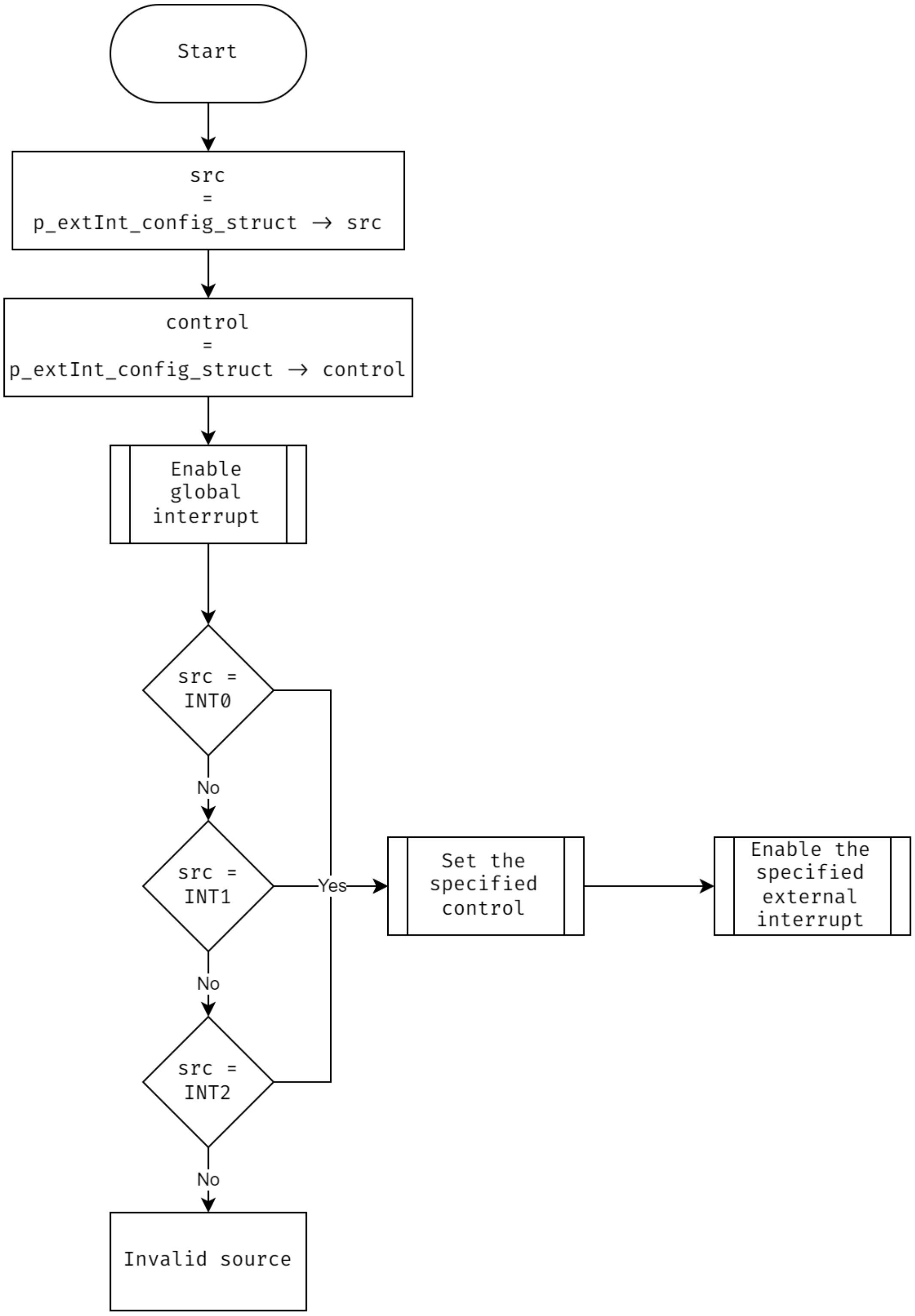
2.0.1 LED Sequence Application



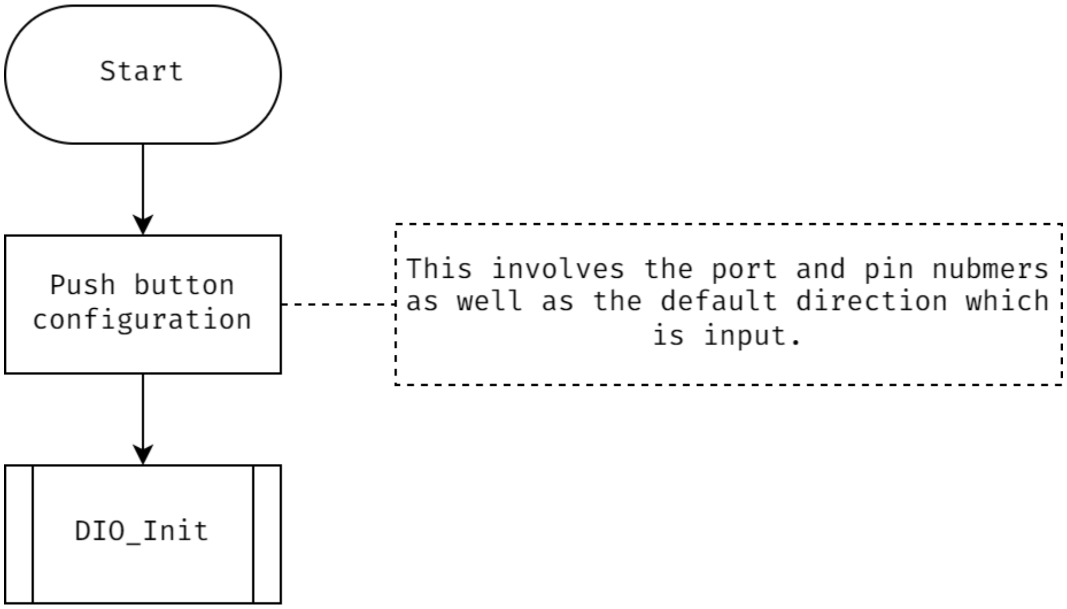
2.0.2 DIO flowchart



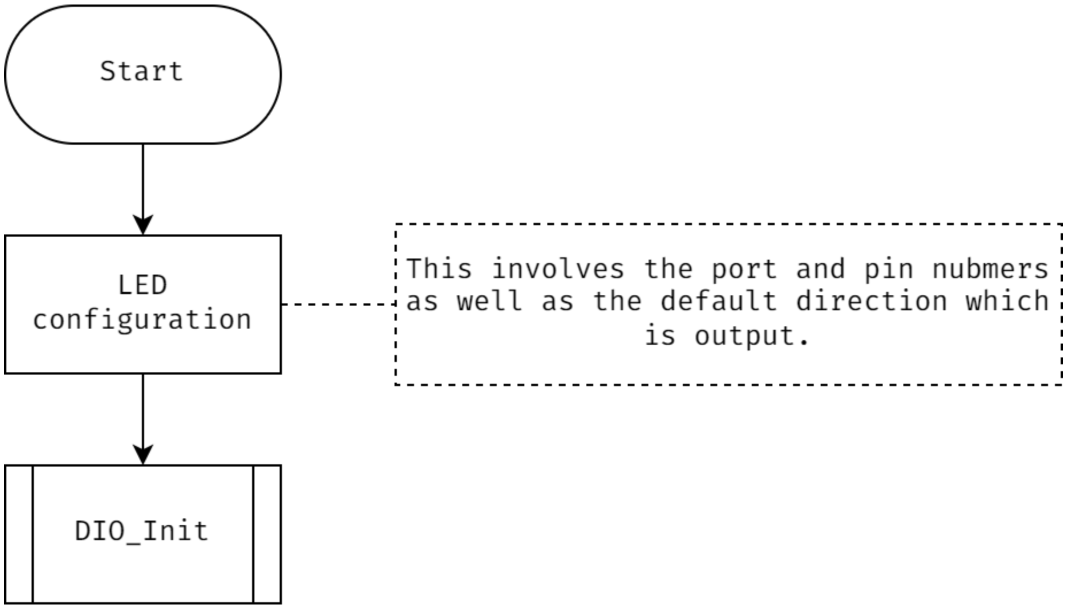
2.0.3 External interrupt flowchart


Start
src = p_extInt_config_struct → src
control = p_extInt_config_struct → control
Enable global interrupt
src = INT0
No
src = INT1
Yes
Set the specified control
Enable the specified external interrupt
No
src = INT2
No
Invalid source

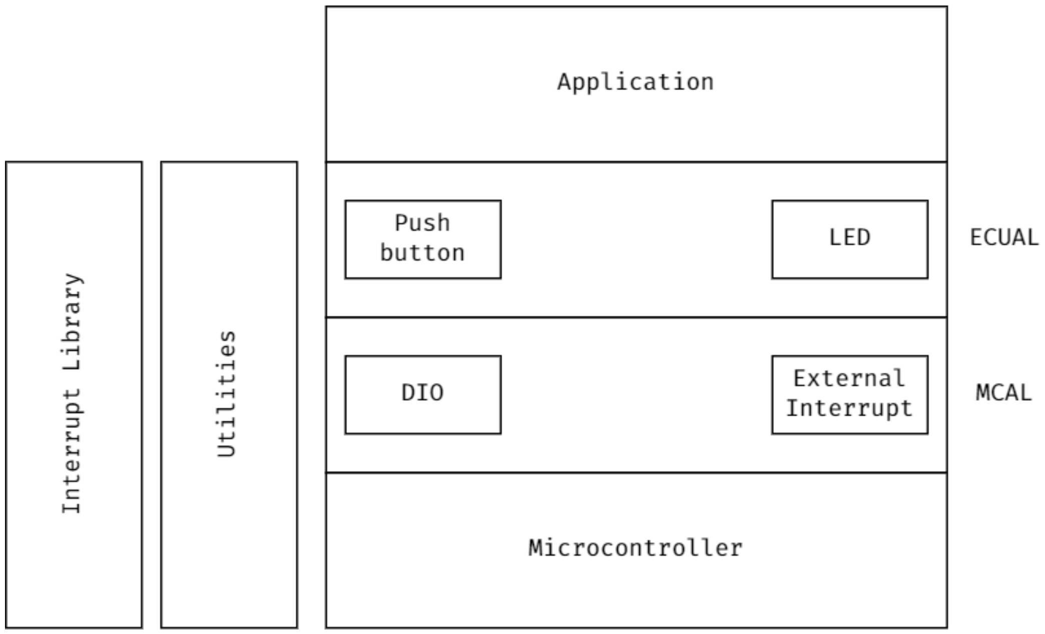
2.0.4 Push button flowchart


Start
Push button configuration
This involves the port and pin nubmers as well as the default direction which is input.
DIO_Init

2.0.5 LED flowchart


Start
LED configuration
This involves the port and pin nubmers as well as the default direction which is output.
DIO_Init

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;

```

```c
/**
* @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);


```

```c

/**
* @brief Reads the state of a specific pin.
* @param[in] p_config_struct Address of the configuration structure.
* @return DIO_PORT_INVALID Port is invalid.
* @return DIO_DIRECTION_INVALID Reading from a pin that is configured as output.
* @return DIO_SUCCESS The read operation is a success.
*/
EN_DIO_ERROR_STATE DIO_ReadPin(DIO_Init_t *p_config_struct);

/**
* @brief Write a specific level to a specified pin.
* @param[in] p_config_struct Address of the configuration structure.
* @return DIO_PORT_INVALID Port is invalid.
* @return DIO_DIRECTION_INVALID Writing to a pin that is configured as input.
* @return DIO_SUCCESS The write operation is a success.
*/
EN_DIO_ERROR_STATE DIO_WritePin(DIO_Init_t *p_config_struct);

/**
* @brief Toggles the current level of a pin.
* @param[in] p_config_struct Address of the configuration structure.
*  @return DIO_PORT_INVALID Port is invalid.
* @return DIO_DIRECTION_INVALID Toggle a pin that is configured as input.
* @return DIO_SUCCESS The toggle operation is a success.
*/
EN_DIO_ERROR_STATE DIO_TogglePin(DIO_Init_t *p_config_struct);
```

4.1.2 External Interrupts

```c
/**
* @enum EN_INT_ERROR_STATE
* @brief Specifies the state of DIO functions.
*/
typedef enum EN_INT_ERROR_STATE {
  INT_SUCCESS = 0, INT_GLOBAL_INT_NOT_SET, INT_INVALID_CONTROL,
  INT_INVALID_EXTERNAL_SRC
}EN_INT_ERROR_STATE;

/**
* @enum EN_EXT_INT_SENSE_CONTROL
* @brief Specifies the triggering mechanism for the external interrupts.
*/
typedef enum EN_EXT_INT_SENSE_CONTROL {
  LOW_LEVEL = 0, ANY_LOGIC_CHANGE, FALLING_EDGE, RISING_EDGE
}EN_EXT_INT_SENSE_CONTROL;

/**
* @enum EN_EXT_INTERRUPT_SRC
* @brief Specifies the external interrupts source.
*/
typedef enum EN_EXT_INTERRUPT_SRC {
  INT0 = 0, INT1, INT2
}EN_EXT_INTERRUPT_SRC;

```

```c
/**
* @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();

```

```c

/**
* @brief Controls the triggering mechanism of INT1 and INT0.
* Interrupt 0 sense control.
* | ISC01 Bit 3 | ISC00 Bit 2 | Description                        |
* |-------------|-------------|-----------------------------------|
* | 0           | 0           | INT0 triggered on low level.       |
* | 0           | 1           | Any logic change triggers.         |
* | 1           | 0           | Falling edge generates interrupt.  |
* | 1           | 1           | Rising edge generates interrupt.   |
* Interrupt 1 sense control
* | ISC11 Bit 3 | ISC10 Bit 2 | Description                        |
* |-------------|-------------|-----------------------------------|
* | 0           | 0           | INT1 triggered on low level.       |
* | 0           | 1           | Any logic change triggers.         |
* | 1           | 0           | Falling edge generates interrupt.  |
* | 1           | 1           | Rising edge generates interrupt.   |
* Interrupt 2 sense control
* 0 - Falling edge activates the interrupt.
* 1 - Rising edge activates the interrupt.
*/
EN_INT_ERROR_STATE INT_ExtIntSenseControl(EN_EXT_INTERRUPT_SRC src,
  EN_EXT_INT_SENSE_CONTROL control);
```

4.2 Electronic Unit Architecture Layer (ECUAL)

4.2.1 LED

```c
/**
* @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;


```

```c

/**
* @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

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