LED_SEQUANCE V2.0

Created By : Sherif Ashraf Ali

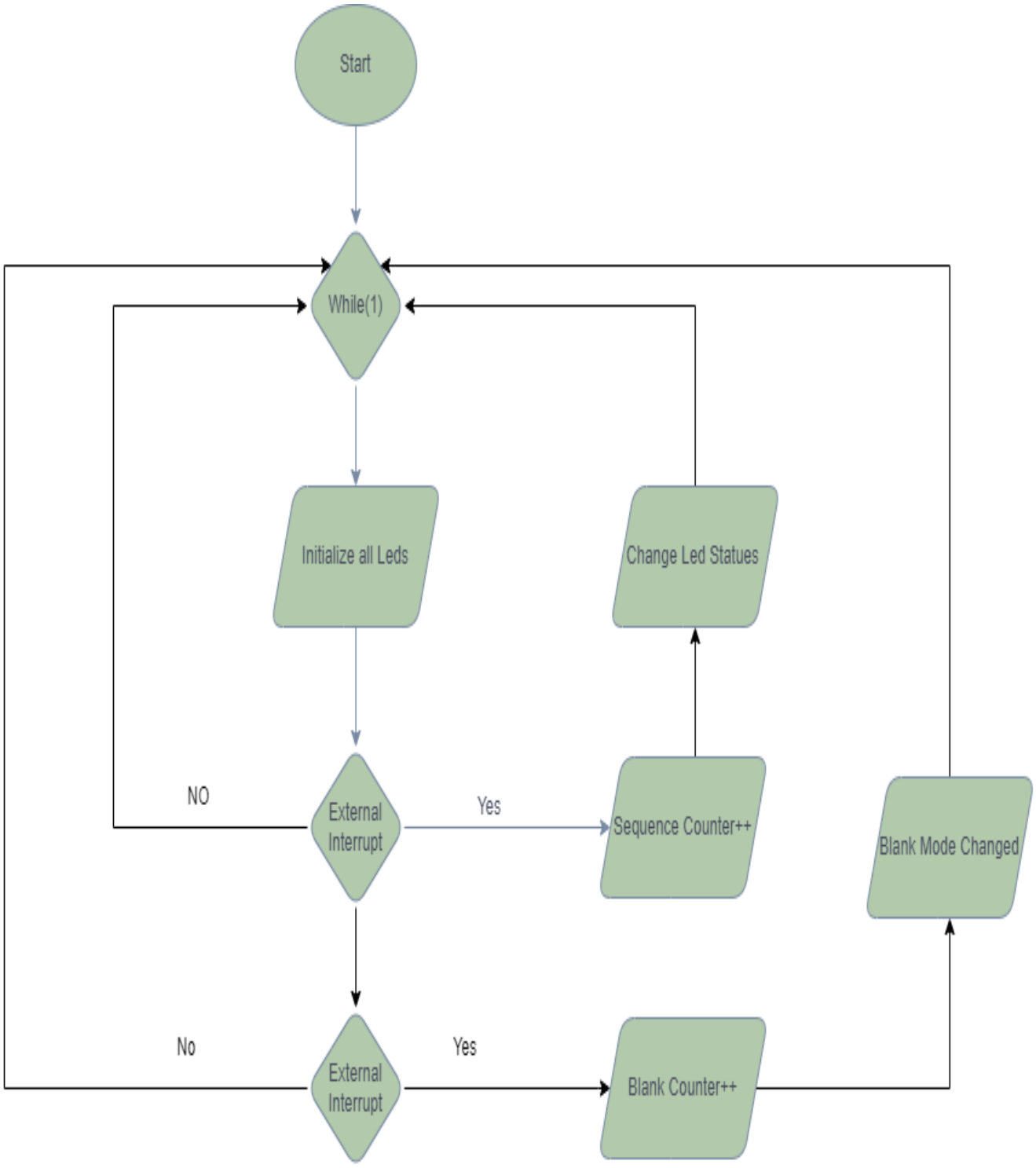
Date : 10/4/2023

Project Description

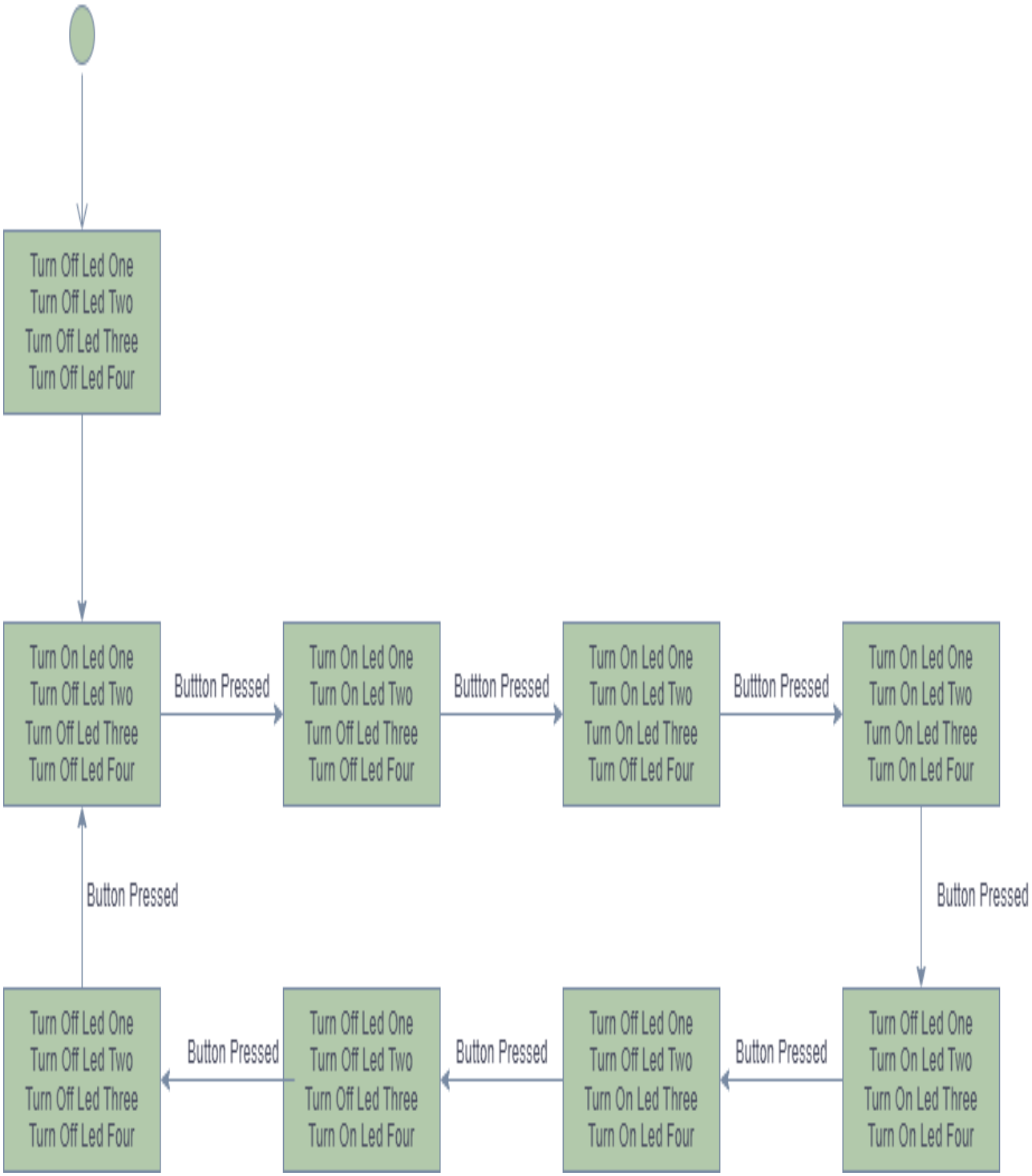
You are supposed to have a system that controls some LEDs lighting sequence according to button pressing.

1. *Description*
2.
    1. ***Hardware Requirements***
        1. Four LEDs (**LED0**, **LED1**, **LED2**, **LED3**)
        2. **Two** buttons (**BUTTON0** and **BUTTON1**)
    2. ***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**

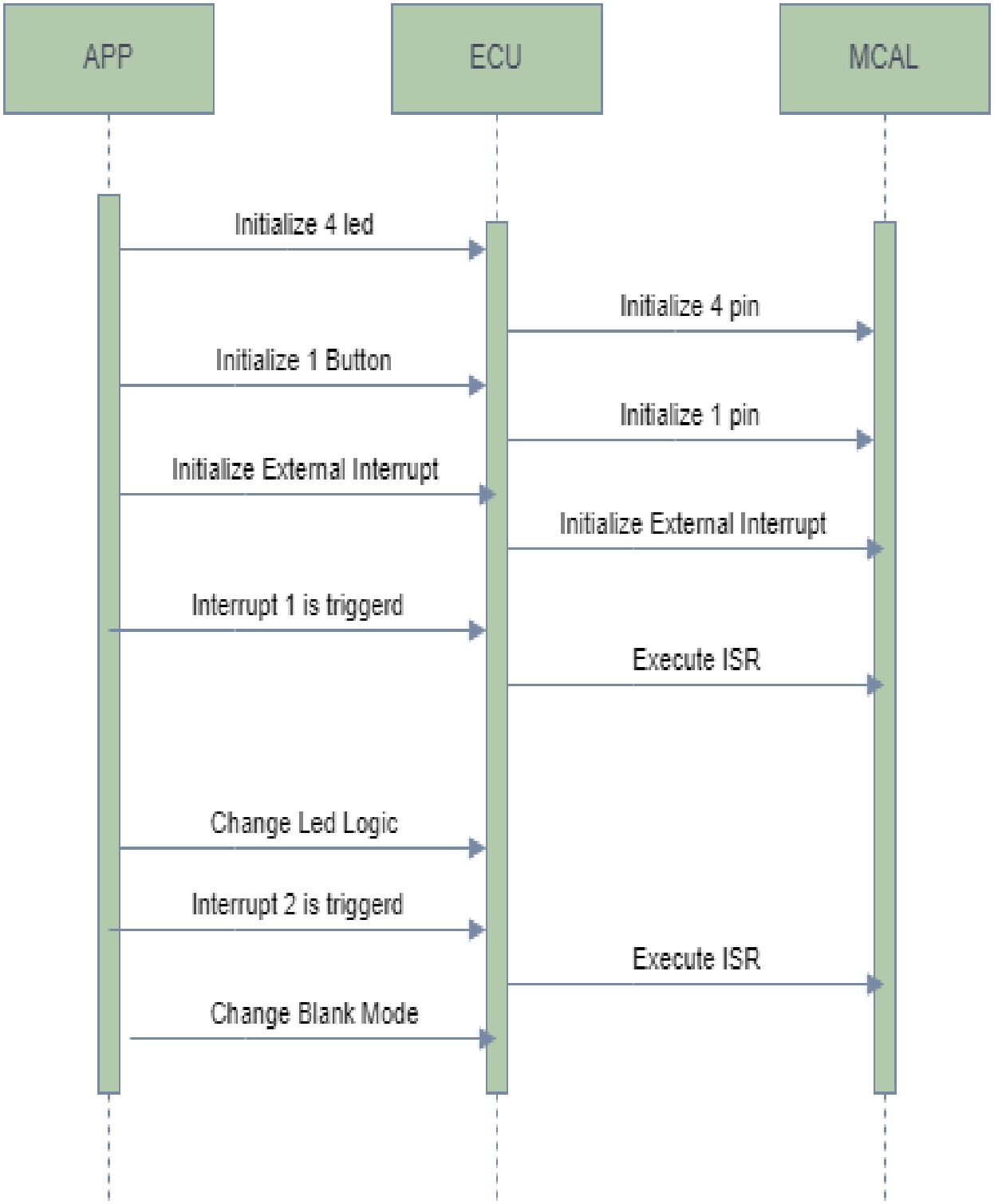
PROJECT FLOWCHART



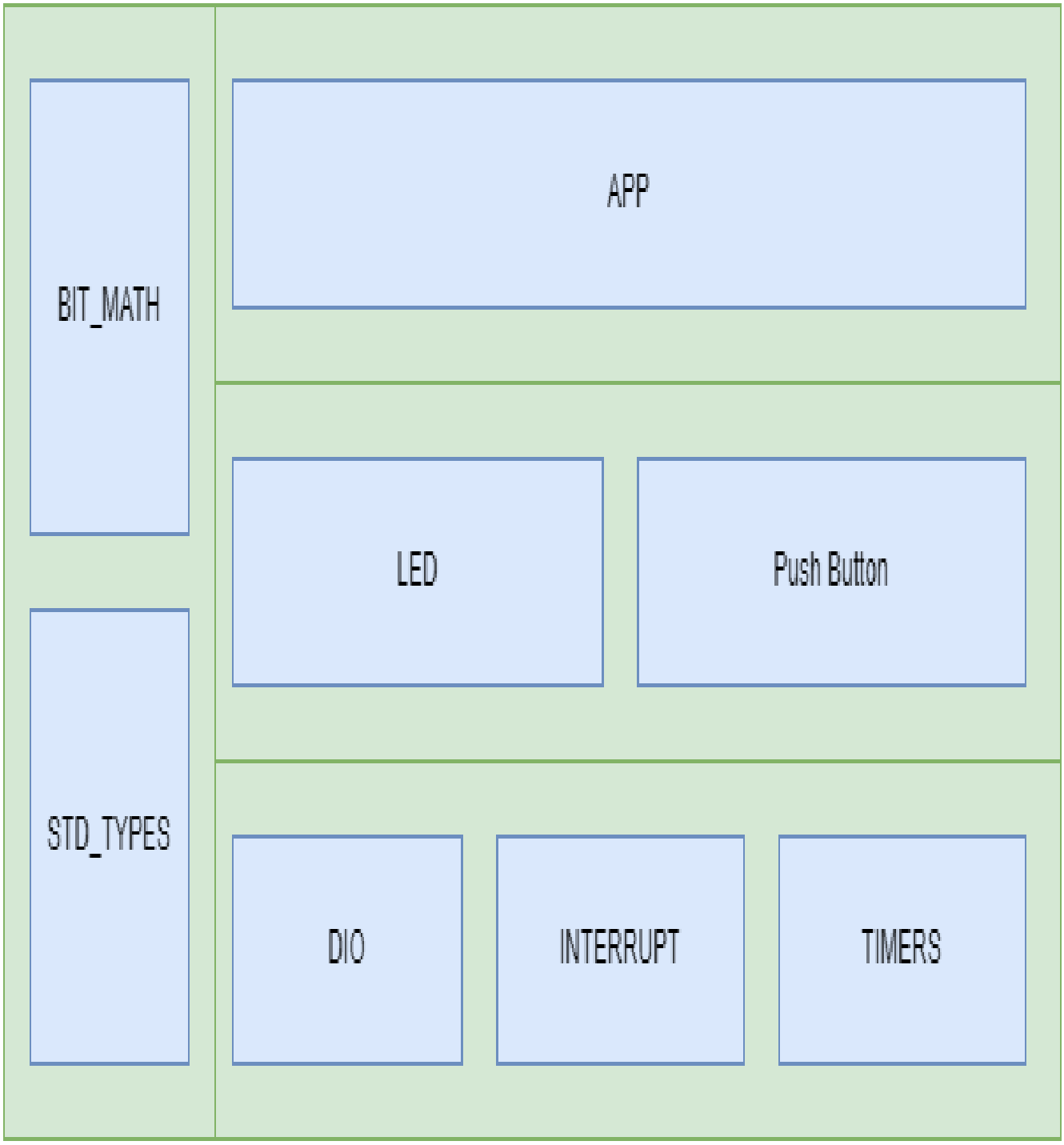
STATE MACHINE



Sequence Diagram



Layered architecture



Project Modules APIs

1-MCAL

1.1 DIO

```
typedef enum{
GPIO_LOGIC_LOW = 0,
GPIO_LOGIC_HIGH
}logic_t;
typedef enum{
GPIO_DIRECTION_OUTPUT = 0,
GPIO_DIRECTION_INPUT
}direction_t;

typedef enum{
  GPIO_PIN0 = 0,
  GPIO_PIN1,
  GPIO_PIN2,
  GPIO_PIN3,
  GPIO_PIN4,
  GPIO_PIN5,
  GPIO_PIN6,
  GPIO_PIN7
}pin_index_t;

typedef enum{
   GPIO_PORTA_INDEX = 0,
   GPIO_PORTB_INDEX,
   GPIO_PORTC_INDEX,
   GPIO_PORTD_INDEX,
```

```
   GPIO_PORTE_INDEX,
}port_index_t;


typedef struct{
  uint8 port : 3;
  uint8 pin : 3;
 uint8 direction : 1;
 uint8 logic : 1;
}pin_config_t;


Std_ReturnType GPIO_pin_direction_intialize(const pin_config_t *_pin_config);
Std_ReturnType GPIO_pin_get_direction_status(const pin_config_t *_pin_config ,
direction_t *direction_status);
Std_ReturnType GPIO_pin_write_logic(const pin_config_t *_pin_config , logic_t
logic);
Std_ReturnType GPIO_pin_read_logic(const pin_config_t *_pin_config , logic_t
*logic_status);
Std_ReturnType GPIO_pin_toggle_logic(const pin_config_t *_pin_config);
Std_ReturnType GPIO_pin_intialize(const pin_config_t *_pin_config);
Std_ReturnType GPIO_port_direction_intialize(port_index_t port , uint8 direction);
Std_ReturnType GPIO_port_get_direction_status(port_index_t port , uint8
*direction_status);
Std_ReturnType GPIO_port_write_logic(port_index_t port , uint8 logic);
Std_ReturnType GPIO_port_read_logic(port_index_t port , uint8 *logic_status);
Std_ReturnType GPIO_port_toggle_logic(port_index_t port);
```

1.2 INTERRUPT

```
#define EXT_INT0 __vector_1
#define EXT_INT1 __vector_2
#define EXT_INT2 __vector_3
```

```
#define ISR(INT_VECT)void INT_VECT(void) __attribute__
((signal,used));\
void INT_VECT(void)

typedef enum
{
MCUCR_REG_ISC00_BITS = 0,
MCUCR_REG_ISC01_BITS,
MCUCR_REG_ISC10_BITS,
MCUCR_REG_ISC11_BITS
}EN_MCUCR_REG_BITS;


typedef enum
{
MCUCSR_REG_ISC2_BITS = 6,
}EN_MCUCSR_REG_BITS;

typedef enum
{
GICR_REG_INT2_BITS = 5,
GICR_REG_INT0_BITS,
GICR_REG_INT1_BITS
}EN_GICR_REG_BITS;

typedef enum
{
GIFR_REG_INTF2_BITS = 5,
GIFR_REG_INTF0_BITS,
GIFR_REG_INTF1_BITS
}EN_GIFR_REG_BITS;


typedef enum
{
LOW_LEVEL_SENSE_CONTROL = 0,
ANY_LOGICAL_SENSE_CONTROL,
FALLING_EDGE_SENSE_CONTROL,
RISING_EDGE_SENSE_CONTROL
}EN_EXT_INTERRUPT_Sense_Control;
```

```
typedef enum
{
EXT0_INTERRUPTS = 0,
EXT1_INTERRUPTS,
EXT2_INTERRUPTS
}EN_EXT_INTERRUPTS;

typedef struct
{
void(*INTERRUPT_EXTERNAL_HANDLER)(void);
EN_EXT_INTERRUPTS EXTERNAL_INTERRUPRT_Number;
EN_EXT_INTERRUPT_Sense_Control
EXTERNAL_INTERRUPRT_Sense_Control;
}ST_EXT_INTERRUPTS_CFG;


Std_ReturnType EXT_vINTERRUPT_Init(const
ST_EXT_INTERRUPTS_CFG *EXT_INTx);
Std_ReturnType EXT_vINTERRUPT_Denit(const
ST_EXT_INTERRUPTS_CFG *EXT_INTx);
```

1.3 TIMERS

```
void delay_ms(uint16_t delay_time)
{
   TCCR1A = 0;
   TCCR1B = (1 << CS01) | (1 << CS00);
   uint16_t timer_counts = (F_CPU / 64UL) * (delay_time / 1000.0);
   TCNT1 = 0;
   while (TCNT1 < timer_counts);
   TCCR1B = 0;
}
```

2. ECU

2.1 LED

```
typedef enum{
   LED_STATUS_OFF = 0,
   LED_STATUS_ON,
}led_status_t;

typedef struct{
   uint8 port_name :3;
   uint8 pin : 3;
   uint8 led_status : 1;
   uint8 reserved : 1;
}led_t;

Std_ReturnType LED_initialize(const led_t *led);
Std_ReturnType LED_turn_on(const led_t *led);
Std_ReturnType LED_turn_off(const led_t *led);
Std_ReturnType LED_toggle(const led_t *led);
```

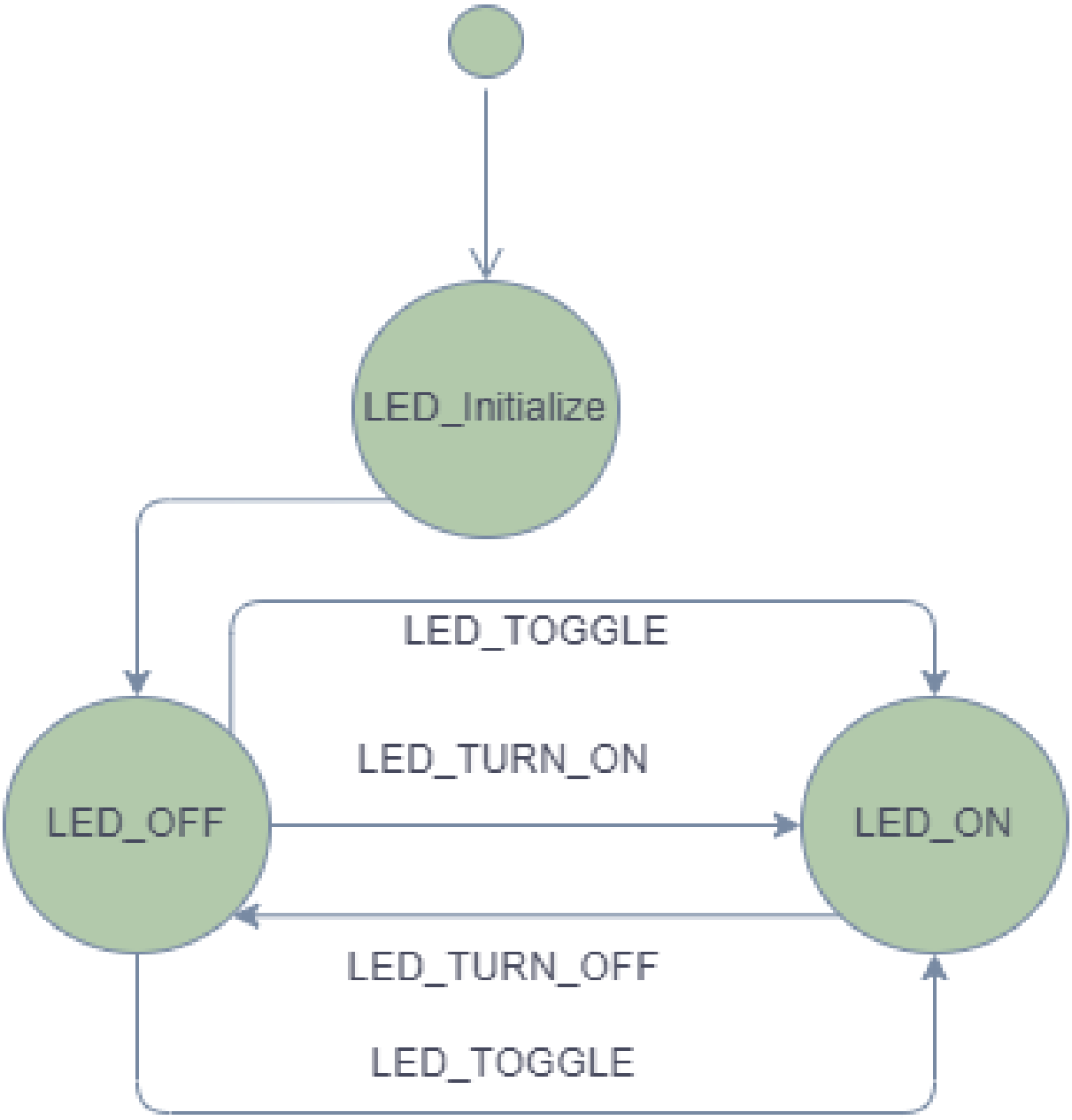
2.2 BUTTON

```
typedef enum{
   PUSH_BTN_STATE_PRESSED = 0,
   PUSH_BTN_STATE_RELEASED
}PUSH_BTN_state_t;



typedef enum{
```

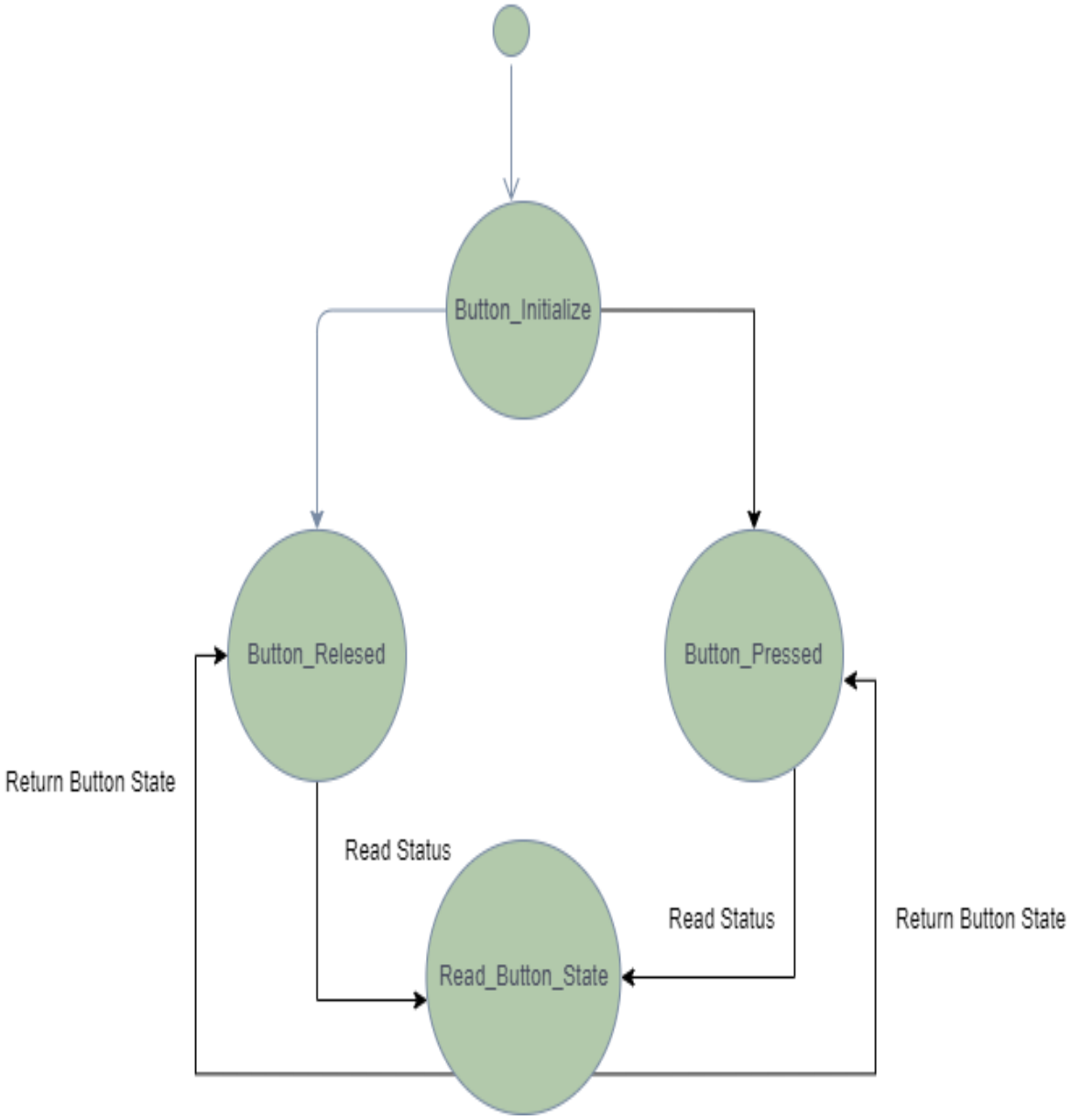
```
    PUSH_BTN_PULL_UP = 0,
    PUSH_BTN_PULL_DOWN
}PUSH_BTN_active_t;

typedef struct{
    pin_config_t PUSH_BTN_pin;
    PUSH_BTN_state_t PUSH_BTN_state;
    PUSH_BTN_active_t PUSH_BTN_connection;
}PUSH_BTN_t;



Std_ReturnType PUSH_BTN_intialize(const PUSH_BTN_t *btn);
Std_ReturnType PUSH_BTN_read_state(const PUSH_BTN_t *btn , PUSH_BTN_state_t
*btn_state);
```

APIs state machine

LED



PUSH BUTTON


Button_Initialize
Button_Relesed
Button_Pressed
Return Button State
Read Status
Read Status
Return Button State
Read_Button_State