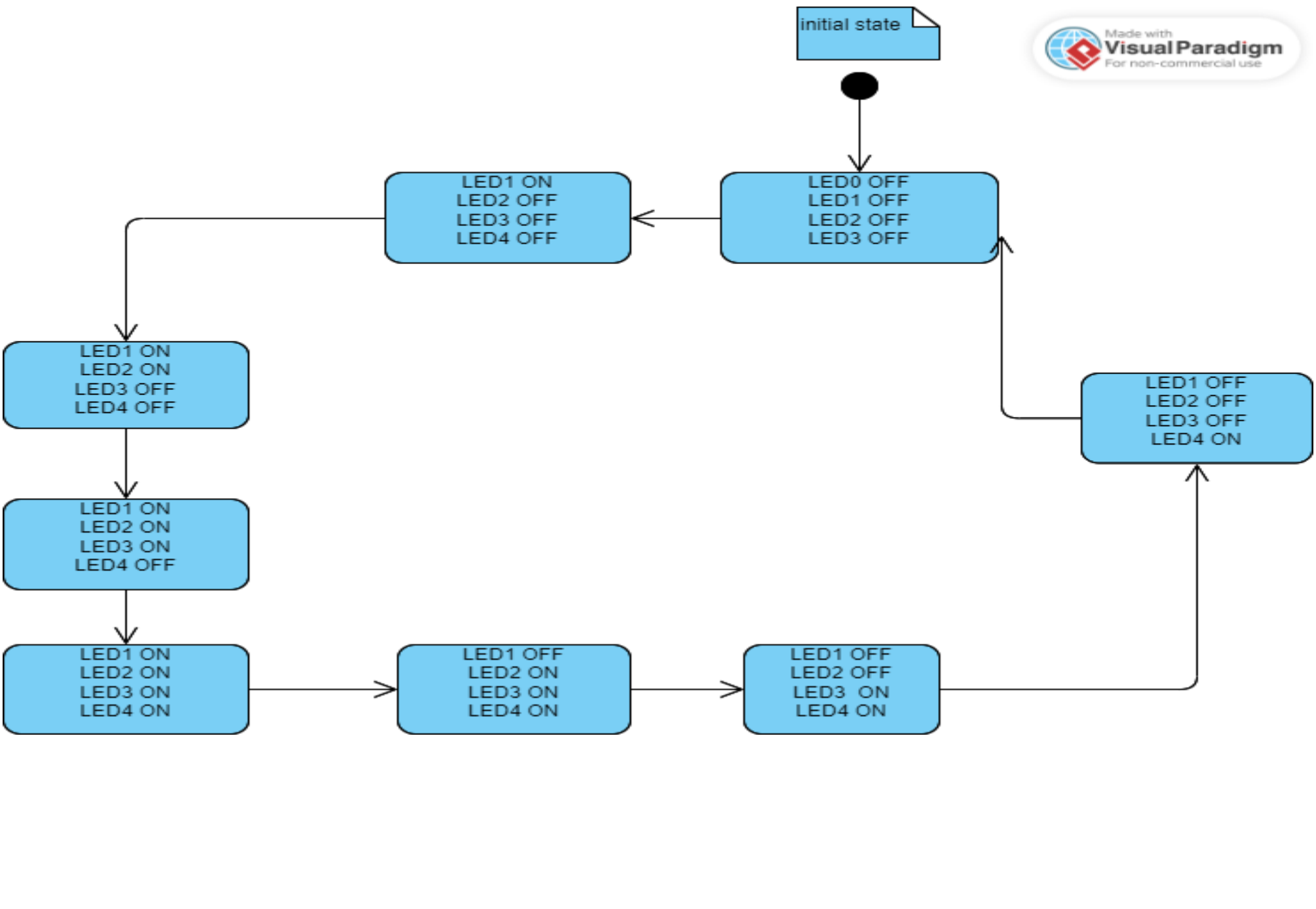
Project Title: Led Sequence V 3.0

Name: Basel Nagy

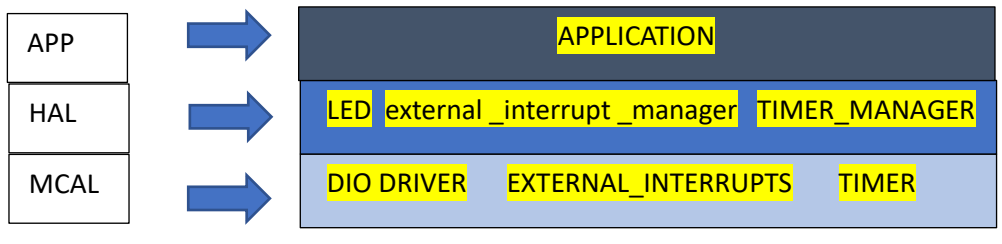
Description:

1. *Description*
    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**

State machine:



Layered architecture:



Project Modules APIs:

DIO DRIVER:

```
/*typedef*/
typedef enum DIO_PORTS
{
	porta, portb, portc, portd
} DIO_PORTS;
typedef enum DIO_PINS
{
	pin0, pin1, pin2, pin3, pin4, pin5, pin6, pin7
} DIO_PINS;
typedef enum PIN_DIRECTION
{
	INPUT,
	OUTPUT
} PIN_DIRECTION;

typedef enum PIN_STATE
{
	LOW,
	HIGH
 } PIN_STATE;

/*****************************************    APIs PROTOTYPES      ********************************************************************/
STD_return DIO_INIT (DIO_PORTS port, DIO_PINS pin, PIN_DIRECTION direction);
STD_return DIO_WRITE_PIN (DIO_PORTS port, DIO_PINS pin, PIN_STATE state);
STD_return DIO_READ_PIN (DIO_PORTS port, DIO_PINS pin, uint8_t* vale);
```

EXTERNAL_INTERRUPTS APIs:

/*typedefs*/

typedef enum INT_NUM {int0, int1, int2} INT_NUM;

typedef enum EDGE {rising,falling} EDGE;

/*** APIs PROTOTYPES **/

STD_return EDGE_SELECET (EDGE edge,INT_NUM ext_int);

STD_return EXT_INTERRUPT_ENABLE (INT_NUM ext_int);

STD_return SETCALLBACK_FUN_INT0(void (*ptr_int0) (void));

STD_return SETCALLBACK_FUN_INT1(void (*ptr_int1) (void));

STD_return SETCALLBACK_FUN_INT2(void (*ptr_int2) (void));

TIMER APIs:

/*typedefs*/
typedef enum EN_timermode_t {normal,ctc,pwm,phase_correct} EN_timermode_t;

/*** APIs PROTOTYPES **/
STD_return TIMER_INIT(EN_timermode_t timer_cfg);
STD_return TIMER_START (uint16_t prescaller);
void TIMER_STOP (void);
void TIMER_OVF_INT_ENABLE (void);
void TIMER_OVF_INT_DISABLE (void);
void TIMER_COMP_INT_ENABLE (void);
void TIMER_COMP_INT_DISABLE (void);
void OVF_VALUE (uint8_t value);
void COMP_VALUE (uint8_t value);
STD_return TIMER_OVF_CALLBACK (void (*ptr) (void));
STD_return TIMER_COMP_CALLBACK (void (*ptr) (void));

LED APIs:

typedef struct LED

{

  DIO_PORTS port;

DIO_PINS pin;

} LED;

/*** APIs PROTOTYPES **/

STD_return LED_INIT (LED* led);

STD_return LED_ON (DIO_PORTS, DIO_PINS);

STD_return LED_OFF (DIO_PORTS,DIO_PINS);

External interrupt manager APIs:

/*typedefs*/
typedef void (*func_ptr)(void);
typedef struct ST_EXT_INT_HANDLER_t
{
EN_INT_NUM_t ext_int;
EN_EDGE_t edge_select;
func_ptr function_ptr;

```
}ST_EXT_INT_HANDLER_t;
 /*****************************************    APIs PROTOTYPES      ************************************************************/
STD_return EXT_INT_HANDLER(ST_EXT_INT_HANDLER_t* handler);
```

TIMER MANAGER APIs:

```
/*typedefs*/
typedef  void (*FUNC_ptr) (void);
typedef enum EN_timer_interrupt {ENABLE, DISABLE} EN_timer_interrupt;
typedef struct ST_timerCFG_t
{
  EN_timermode_t mode;
  EN_timer_interrupt interrupt;
  FUNC_ptr func_ptr;
  uint8_t init_value;
} ST_timerCFG_t;
/*****************************************    APIs PROTOTYPES      ************************************************************/
STD_return TIMER_MANAGER_CFG (ST_timerCFG_t * timer_cfg);
void TIMER_MANAGER_STA (uint8_t prescaller);
void TIMER_MANAGER_STO (void);
```