MOHAMED SAYED HUSSEIN
Embedded Software Engineer
E-mail: *msh.comm@gmail.com*
Cellular: Serbia **+381 628332617** _ Egypt **+20 1113313898**
LinkedIn: *https://www.linkedin.com/in/mohamed-sayed-eng/*

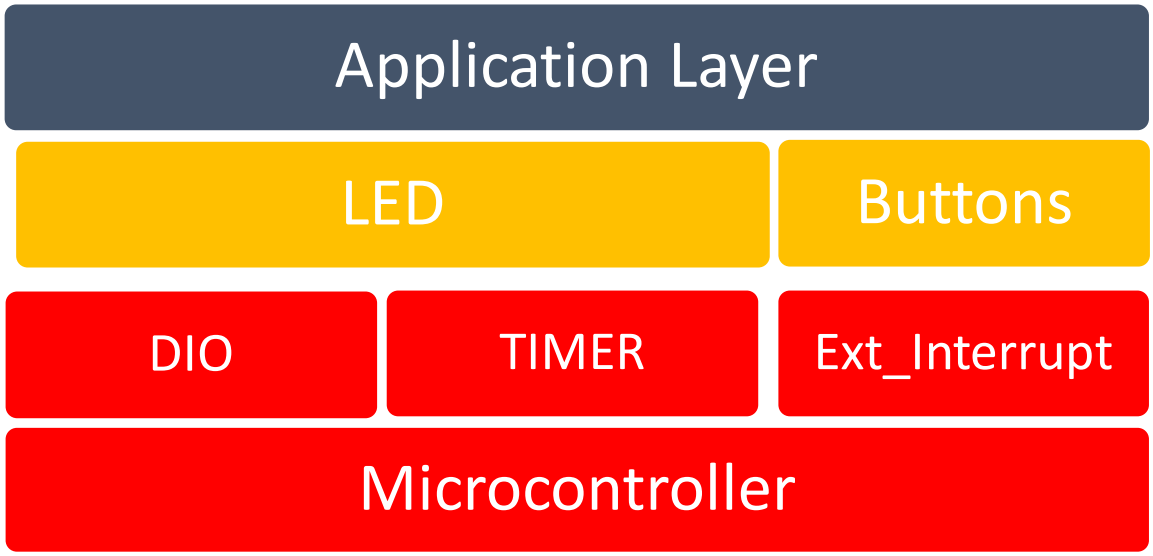
On-demand Traffic light control for Udacity

Static Architecture Design

1. **Define system layers:**
   - Microcontroller Abstraction Layer (MCAL)
   - Electronic Unit Abstraction Layer (ECUAL)
   - Application Layer

2. **Define system drivers:**
   - DIO driver
   - Timer driver
   - Interrupt driver
   - LED driver
   - Button driver

3. **Place each driver into the appropriate layer in the appropriate order:**



- MCAL : DIO driver, General purpose TIMER driver, External Interrupt driver
- ECUAL : LED module, Button module
- App Layer : Application module

4. Define APIs for each driver:

Define APIS with its documentation, description, input arguments, output arguments, and return:

```
/*******************************************************************************
 *                                DIO Function Definition                      *
 *******************************************************************************/

/*
 * Service Name       : DIO_Init
 * Parameters (in)    : portNumber, pinNumber, direction, alternative
 * Parameters (out)   : None
 * Return value       : EN_returnError_t
 * Description        : Function to Setup the pin configuration:
 *                      Setup the pin as Digital GPIO pin
 *                      Setup the internal resistor for i/p pin
 *                      Setup the mode of alternative function
 *                      Setup the initial value of the pin High or Low
 *                      If the input port number or pin number are not correct, The function will return   WRONG_RETURN as EN_returnError_t type.
 */
EN_returnError_t DIO_Init(ST_DIO_ConfigPin_t *pinPTR);



/*
 * Service Name       : DIO_PortInit
 * Parameters (in)    : portNumber, direction
 * Parameters (out)   : None
 * Return value       : EN_returnError_t
 * Description        : Function to Setup the PORT configuration:
 *                      Setup the PORT as Digital GPIO pin
 *                      Setup the initial value of the port High or Low
 *                      If the input port number is not correct, The function will return WRONG_RETURN as EN_returnError_t type.
 */
EN_returnError_t DIO_PortInit(uint8 portNumber, EN_portDirection direction);


/*
 * Service Name       : Dio_ReadPin
 * Parameters (in)    : *pinPTRr
 * Parameters (out)   : *readValue
 * Return value       : EN_returnError_t
 * Description        : write the value for the required pin, it should be LOGIC_HIGH or LOGIC_LOW.
 *                      If the input port number or pin number are not correct, The function will return WRONG_RETURN as EN_returnError_t type.
 */
EN_returnError_t Dio_ReadPin(ST_DIO_ConfigPin_t *pinPTR, boolean *readValue);


/*
 * Service Name       : Dio_WritePin
 * Parameters (in)    : *pinPTRr, writeValue
 * Parameters (out)   : None
 * Return value       : EN_returnError_t
 * Description        : Write input value for the required pin, it should be LOGIC_HIGH or LOGIC_LOW.
 *                      If the input port number or pin number are not correct, The function will return WRONG_RETURN as EN_returnError_t type.
 */
EN_returnError_t Dio_WritePin(ST_DIO_ConfigPin_t *pinPTR, boolean writeValue);


/*
 * Service Name       : Dio_ToggelPin
 * Parameters (in)    : *pinPTRr
 * Parameters (out)   : None
 * Return value       : EN_returnError_t
 * Description        : Toggle the value for the required pin, it should flapping LOGIC_HIGH to LOGIC_LOW and vice versa .
 *                      If the input port number or pin number are not correct, The function will return WRONG_RETURN as EN_returnError_t type.
 */

EN_returnError_t Dio_TogglePin(ST_DIO_ConfigPin_t *pinPTR);
```

```
/*******************************************************************************
 *                                gpTimer Function Definitions                 *
 *******************************************************************************/
```

- - Frequency = 1MHz
- - Prescaler = 1024
- - Resolution = 1/ (Frequency/ Prescaler) = 1024µs
- - Ticks = 244
- - T = Resolution*Ticks = 0.25S
- - So you need 4 times timer over flow to get 1 second
- - Timer mode = compare mode
- - Timer initial value = 0
- - Timer compare value = 244

```
/*
 * Service Name        : Timer_init
 * Parameters (in)     : *Timer_Config
 * Parameters (out)    : None
 * Return value        : EN_TimerError_t
 * Description         : Function to Setup the TIMER configuration:
 *                       1- The Timer Id
 *                       2- The Timer Mode (Normal , Compare)
 *                       3- The Timer Prescalar
 *                       4- The Timer Initial Value That will start counting from it
 *                       5- The The Timer Compare Value (In Compare Mode Only)
 *                       If the Timer initialization does not done correctly,
 *                       The function will return EN_TimerError_t type according to the timer.
 */
EN_TimerError_t Timer_init(const ST_Timer_Config_t *Timer_ConfigPTR);



/*
 * Description: Function to set the Call Back function address of Timer0.
 */
void Timer0_setCallBack(void(*T0_ptr)(void));

/*
 * Description: Function to set the Call Back function address of Timer1.
 */
void Timer1_setCallBack(void(*T1_ptr)(void));

/*
 * Description: Function to set the Call Back function address of Timer2.
 */
void Timer2_setCallBack(void(*T2_ptr)(void));
```

```
/*******************************************************************************
 *                                External Interrupt Function Definitions                                *
 *******************************************************************************/
/*
 * Service Name        : Ext_Interrupts_Init
 * Parameters (in)     : *Ext_Int_ConfigPTR
 * Parameters (out)    : None
 * Return value        : EN_ExtIntError_t
 * Description         : Function to Setup the External Interrupts configuration:
 *                       1- The External Interrupt Id
 *                       2- The External Interrupt Mode (LowLevel, ChangeLevel, FailingEdge, RisingEdge)
 *                       3- The External Interrupt Enable must be TRUE to Enable INT bit in GICR (General Interrupt Control Register)
 *                       4- Enable General Interrupt bit (I-bit)
 *                       If the External Interrupt initialization does not done correctly,
 *                       The function will return EN_ExtIntError_t type according to the error.
 */
EN_ExtIntError_t Ext_Interrupts_Init(ST_Ext_Int_Config_t *Ext_Int_ConfigPTR);


/*
 * Service Name        : Ext_Interrupts_deinit
 * Parameters (in)     : *Ext_Int_ConfigPTR
 * Parameters (out)    : None
 * Return value        : EN_ExtIntError_t
 * Description         : Function to uninitialize the External Interrupts configuration:
 *                       1- The External Interrupt Id
 *                       2- The External Interrupt must be disabled (Ext_Int_Enable = FALSE or 0) in Ext_Int_ConfigPTR
 *                          to clear INT0, INT1, or INT2 bit in GICR (General Interrupt Control Register)
 *                       If the External Interrupt uninitialized correctly,
 *                       The function will return EN_ExtIntError_t type according to the error.
 */
EN_ExtIntError_t Ext_Interrupts_deinit(ST_Ext_Int_Config_t *Ext_Int_ConfigPTR);


/*
 * Description: Function to set the Call Back function address of External interrupt 0.
 */
void Ext_Interrupt0_setCallBack(void(*T0_ptr)(void));


/*
 * Description: Function to set the Call Back function address of External interrupt 1.
 */
void Ext_Interrupt1_setCallBack(void(*T1_ptr)(void));


/*
 * Description: Function to set the Call Back function address of External interrupt 2.
 */
void Ext_Interrupt2_setCallBack(void(*T2_ptr)(void));
```

```
/*******************************************************************************
 *                                LED module Function Definitions                                *
 *******************************************************************************/

/*
 * Service Name                 : LED_init
 * Parameters (in)      : None
 * Parameters (out)     : None
 * Return value                 : EN_LEDError_t
 * Description                  : Function to Setup the LED configuration:
 *                                                      If the input port number or pin number are not correct, The function will return
WRONG_RETURN as EN_LEDError_t type.
 */
EN_LEDError_t LED_init(void);


/*
 * Service Name                 : LED_setOn
 * Parameters (in)      : ledPTR
 * Parameters (out)     : None
 * Return value                 : EN_LEDError_t
 * Description                  : Set the LED state to ON
 *                                                      in case of fail to set LED to ON, The function will return ERROR_LED
 */
EN_LEDError_t LED_setOn(ST_DIO_ConfigPin_t *ledPTR);


/*
 * Service Name                 : LED_setOff
 * Parameters (in)      : *ledPTR
 * Parameters (out)     : None
 * Return value                 : EN_LEDError_t
 * Description                  : Set the LED state to OFF
 *                                                      in case of fail to set LED to OFF, The function will return ERROR_LED
 */
EN_LEDError_t LED_setOff(ST_DIO_ConfigPin_t *ledPTR);


/*
 * Service Name                 : LED_toggle
 * Parameters (in)      : *ledPTR
 * Parameters (out)     : None
 * Return value                 : EN_LEDError_t
 * Description                  : Toggle the LED state
 *                                                      in case of fail to toggling LED state, The function will return ERROR_LED
 */
EN_LEDError_t LED_toggle(ST_DIO_ConfigPin_t *ledPTR);


/*
 * Service Name                 : LED_toggle
 * Parameters (in)      : *ledPTR
 * Parameters (out)     : None
 * Return value                 : EN_LEDError_t
 * Description                  : Refresh the LED state, by reading led output value on pin and rewrite the same value
 *                                                      in case of fail to Refreshing LED state, The function will return ERROR_LED
 */
/* Description: Refresh the LED state */
EN_LEDError_t LED_refreshOutput(ST_DIO_ConfigPin_t *ledPTR);
```

```
/*******************************************************************************
 *                                  Button module Function Definitions                                  *
 *******************************************************************************/
/*
 * Service Name        : Button_Init
 * Parameters (in)     : None
 * Parameters (out)    : None
 * Return value        : EN_LEDError_t
 * Description         : Function to Setup the LED configuration:
 *                       If the led initialization does not done correctly, The function will return EN_buttonError_t type according to the DIO or external
 *                        interrupt error otherwise return OK_BUTTON
 */
EN_buttonError_t Button_Init(void);



/*******************************************************************************
 *                                  Application module Function Definitions                                  *
 *******************************************************************************/

/*
 * Service Name        : normalMode
 * Parameters (in)     : None
 * Parameters (out)    : None
 * Return value        : EN_appError_t
 * Description         : Function to call the yellow, red, and green tasks in the normal sequence required
 *                       If all tasks processed correctly, The function will return OK_APP
 *                       Else return as EN_appError_t type for error handling according to task fail
 */
EN_appError_t normalMode(void);



/*
 * Service Name        : pedestrianMode
 * Parameters (in)     : None
 * Parameters (out)    : None
 * Return value        : EN_appError_t
 * Description         : Function to be called if the Button pressed
 *                       If all tasks checking the cases of pressing Button correctly, the function will return OK_APP
 *                       Else return as EN_appError_t type for error handling according to task fail
 */
EN_appError_t pedestrianMode(void);



/*
 * Service Name        : APP_Init
 * Parameters (in)     : None
 * Parameters (out)    : None
 * Return value        : EN_appError_t
 * Description         : Function to Setup the application configuration
 *                        initialize LEDs as output and Button as input
 *                       set the call back functions
 *                       If the initialization processed correctly, the function will return OK_APP
 *                       Else return as EN_appError_t type for error handling according to task fail
 */
EN_appError_t APP_Init(void);



/*
 * Service Name        : APP_start
 * Parameters (in)     : None
 * Parameters (out)    : None
 * Return value        : EN_appError_t
 * Description         : Function to start the application by calling modes
 *                       If mode called correctly, the function will return OK_APP
 *                       Else return as EN_appError_t type for error handling according to mode fail
 */
EN_appError_t APP_start(void);
```

5. Define the new data types you will use in these drivers:

```
/*******************************************************************************
 *                      DIO driver Data Types                                  *
 *******************************************************************************/
/* Enum type to define the pin direction */
typedef enum EN_pinDirection{
        PIN_INPUT=0 , PIN_OUTPUT=1
}EN_pinDirection;


/* Enum type to define the port direction */
typedef enum EN_portDirection{
        PORT_INPUT=0 , PORT_OUTPUT=0xFF
}EN_portDirection;


/* Enum type to define the alternative function of pin */
typedef enum EN_pinAlternative{
        PIN_DIO=0, PIN_ANALOUGE, PIN_COUNTER, PIN_COMPARATOR, PIN_ICU, PIN_SPI_MOSI, PIN_SPI_MISO, PIN_UART_TX,
PIN_UART_RX, PIN_PWM
}EN_pinAlternative;


/* Enum type to define the internal resistor status */
typedef enum EN_pinInternalResisrot{
        OFF, PULL_UP, PULL_DOWN
}EN_pinInternalResisrot;


typedef struct ST_DIO_ConfigPin_t{
        uint8                   portNumber;
        uint8                   pinNumber;
        EN_pinDirection         pinDirection;       /* PIN_INPUT or PIN_OUTPUT */
        EN_pinAlternative       alternativeFunction;  /* select the pin mode as GPIO or alternative function... */
        EN_pinInternalResisrot  pinInternalResistor;  /* OFF, PULL_UP, or PULL_DOWN */
        uint8                   pinLevelValue;      /* LOGIC_HIGH or LOGIC_LOW */
}ST_DIO_ConfigPin_t;


/* Enum type for return type, to handle APIs error */
typedef enum EN_returnError_t
{
        OK_DOI, WRONG_PIN, WRONG_PORT, WRONG_DIRCTION
}EN_returnError_t;
```

```
/*******************************************************************************
 *                      gpTimer driver Data Types                              *
 *******************************************************************************/

/* NOTE:  Timer0 :  8-bit counter
 *        Timer1 : 16-bit counter
 *        Timer2 :  8-bit counter */

/* enum data type to choose one of the 3 AVR timers */
typedef enum EN_Timer_Number{
        Timer0, Timer1, Timer2
}EN_Timer_Number;


/* enum data type to choose AVR Timer Mode */
typedef enum EN_Timer_Modes{
        Normal_Mode, Compare_Mode
}EN_Timer_Modes;


/* enum data type to choose AVR Timer Prescalar For Timer 0,1 */
typedef enum EN_Timer_Prescalar{
        No_Clock, F_CPU_CLOCK, F_CPU_8, F_CPU_64, F_CPU_256, F_CPU_1024,
        Ext_CLK_Falling_Edge, Ext_CLK_Rising_Edge
}EN_Timer_Prescalar;


/* enum data type to choose AVR Timer Prescalar For Timer2 as it has different prescalar than TIMERS 0,1 */
typedef enum EN_Timer2_Prescalar{
        no_clock, f_cpu_clock, f_cpu_8, f_cpu_32, f_cpu_64, f_cpu_128,
        f_cpu_256, f_Cpu_1024
}EN_Timer2_Prescalar;


/* Configuration Structure for AVR Timer Driver Which configure:
 1- The timer ID we want to use (0,1,2)
 2- The Timer driver modes (NormalMode or Compare_mode)
 3- The Timer Prescalar
 4- The Timer Initial value
 5- The Timer Compare Value
*/
typedef struct ST_Timer_Config_t{
        EN_Timer_Number    Timer_ID;
        EN_Timer_Modes     Timer_mode ;
        EN_Timer_Prescalar  Prescalar;
        EN_Timer2_Prescalartimer2_prescalar;
        uint16             Timer_Initial_value;
        uint16             Timer_Compare_value;
}ST_Timer_Config_t;


/* Enum type for return type, to handle APIs error */
typedef enum EN_TimerError_t{
         ERROR_TIMER0 ,ERROR_TIMER1, ERROR_TIMER2, ERROR_TIMER_ID, ERROR_TIMER_MODE, OK_TIMER
}EN_TimerError_t;
```

```
/*******************************************************************************
 *                                External Interrupt driver Data Types                                 *
 *******************************************************************************/
/* Enum data type to choose one of the 3 AVR external interrupt */
typedef enum EN_Ext_Interrupt_Number_t{
        EXT_INT0, EXT_INT1, EXT_INT2
}EN_Ext_Interrupt_Number_t;


/* enum data type to choose AVR Timer Mode */
typedef enum EN_Ext_Int_Modes{
        LowLevel, HighLevel, ChangeLevel, FailingEdge, RisingEdge
}EN_Ext_Int_Modes;


/* Configuration Structure for AVR external interrupt Driver Which configure:
 1- The Ext_Int_ID we want to use (0,1,2)
 2- The Ext_Int_Mode driver modes (Low Level, High Level, any Level Change, Failing Edge, or Rising Edge)
 3- The Ext_Int_Enable want to Enable or Disable (TRUE or FALSE)
*/
typedef struct ST_Ext_Int_Config_t{
        EN_Ext_Interrupt_Number_t       Ext_Int_ID;
        EN_Ext_Int_Modes                Ext_Int_Mode;
        boolean                                                 Ext_Int_Enable;
}ST_Ext_Int_Config_t;


/* Enum type for return type, to handle APIs error */
typedef enum EN_ExtIntError_t{
         ERROR_EXT_INT0 ,ERROR_EXT_INT1, ERROR_EXT_INT2,
         ERROR_EXT_INT_ID, ERROR_EXT_INT_MODE, OK_EXT_INT
}EN_ExtIntError_t;



/*******************************************************************************
 *                                LED module Data Types                                                *
 *******************************************************************************/
/* Set the led ON/OFF according to its configuration Positive logic or negative logic */
#define LED_ON  LOGIC_HIGH
#define LED_OFF LOGIC_LOW


/* Enum type for return type, to handle APIs error */
typedef enum EN_LEDError_t{
        OK_LED, ERROR_LED ,ERROR_LED_RED, ERROR_LED_YALLOW, ERROR_LED_GREEN
}EN_LEDError_t;



/*******************************************************************************
 *                                Button module Data Types                                             *
 *******************************************************************************/
/* Enum type for return type, to handle APIs error */
typedef enum EN_buttonError_t{
        OK_BUTTON, ERROR_BUTTON_DIO_INIT, ERROR_BUTTON_EXT_INT_INIT
}EN_buttonError_t;



/*******************************************************************************
 *                                Application Module Data Types                                        *
 *******************************************************************************/
/* Enum type for return type, to handle APIs error */
typedef enum EN_appError_t{
        OK_APP, ERROR_APP_LED_INIT, ERROR_APP_BUTTON_INIT, ERROR_APP_TIMER_INIT,
        ERROR_APP_RED_ON, ERROR_APP_YELLOW_ON, ERROR_APP_GREEN_ON,
        ERROR_NORMAL_MODE, ERROR_PEDESTRIAN_MODE
}EN_appError_t;
```