# System Description

The system consists of two modes normal mode and pedestrian’s mode.

It controls two traffic lights one for cars and one for pedestrians.

It has button if the button pressed the system will change from normal mode to pedestrian’s mode.

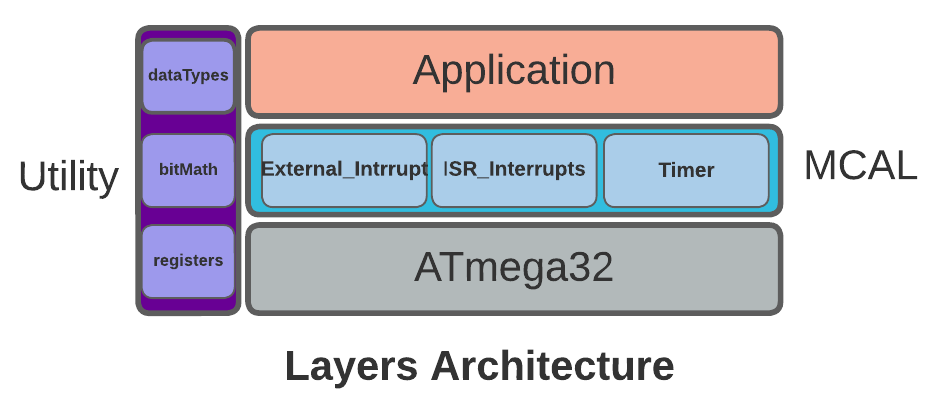
## Used Hardware

* ATmega32 MCU
* 6 LEDs
* 1 Push Button

## Software

* Microchip Studio for developing and debugging
* Proteus for Simulation

# Software Architecture



# System Design

## Normal Mode

## Pedestrian’s Mode

### Scenario 1:

Button is Pressed while Yellow before Green is blinking.

Button is pressed

System goes back to Normal Mode

Pedestrian’s Mode

### Scenario 2:

Button is Pressed while Green on.

Button is pressed

System goes back to Normal Mode

Pedestrian’s Mode

### Scenario 3:

Button is Pressed while Yellow After Green is blinking.

Button is pressed

System goes back to Normal Mode

Pedestrian’s Mode

### Scenario 4:

Button is Pressed while Red is on.

Button is pressed

System goes back to Normal Mode

Pedestrian’s Mode

# Modules

## Utility Layer

### bitMath driver

provides the functions that set, get, clear and toggle one, more or all bits in a register.

Enums and Typedefs:

No Enums or Typedefs.

Macros:

macros function for setting, getting, clearing and toggling bits in the input register.

For a single bit in the register

SETBit(REG,BIT\_NO)

CLRBit(REG,BIT\_NO)

TGLBit(REG,BIT\_NO)

GETBit(REG,BIT\_NO)

For custom number of bits in the register

SETBits(REG,bMsk)

CLRBits(REG,bMsk)

TGLBits(REG,bMsk)

For all bits in the register

SETALLBits(REG)

CLRALLBits(REG)

TGLALLBits(REG)

REG : is the input register

BIT\_NO : the bit number of the input register.

bMsk : the mask of desired bits of the input register.

Functions:

No Functions.

### dataTypes driver

provides the alias names for data types for code portability.

Enums and Typedefs:

typedef unsigned char u8;

typedef signed char s8;

typedef unsigned int u16;

typedef signed int s16;

typedef unsigned long u32;

typedef signed long s32;

Macros:

No Macros.

Functions:

No Functions.

### registers driver

provides the Alias Names for addresses and single bits of the used registers in the system.

Enums and Typedefs:

Alias names for Pins in PORTA, PORTB, PORTC, PORTD.

typedef enum{PA0,PA1,PA2,PA3,PA4,PA5,PA6,PA7}PortA;

typedef enum{PB0,PB1,PB2,PB3,PB4,PB5,PB6,PB7}PortB;

typedef enum{PC0,PC1,PC2,PC3,PC4,PC5,PC6,PC7}PortC;

typedef enum{PD0,PD1,PD2,PD3,PD4,PD5,PD6,PD7}PortD;

Macros:

Alias Names for the addresses of the registers.

A general function to provide the address

#define SELECTOR(ADDRESS) (\*((volatile u8\*)ADDRESS))

#define SELECTOR\_16(ADDRESS) (\*((volatile u16\*)ADDRESS))

Port A Register

#define PORTA SELECTOR(0x3B)

#define DDRA SELECTOR(0x3A)

#define PINA SELECTOR(0x39)

Port B Register

#define PORTB SELECTOR(0x38)

#define DDRB SELECTOR(0x37)

#define PINB SELECTOR(0x36)

Port C Register

#define PORTC SELECTOR(0x35)

#define DDRC SELECTOR(0x34)

#define PINC SELECTOR(0x33)

EXTERNAL INTERRUPT REGESTERS

#define MCUCR SELECTOR(0X55)

#define MCUCSR SELECTOR(0X54)

#define GICR SELECTOR(0X5B)

#define GIFR SELECTOR(0X5A)

#define SREG SELECTOR(0x5F)

Timers Registers

#define TCCR0 SELECTOR(0x53)

#define OCR0 SELECTOR(0x5C)

#define TCNT0 SELECTOR(0x52)

#define TIMSK SELECTOR(0x59)

#define TIFR SELECTOR(0x58)

Timer0 registers

#define TCCR0 SELECTOR(0x53)

#define TCNT0 SELECTOR(0x52)

#define OCR0 SELECTOR(0x5C)

#define TIMSK SELECTOR(0x59)

#define TIFR SELECTOR(0x58)

Functions: No Functions.

## MCAL Layer

### Timer driver

Enums and Typedefs:

Provide selection for the timer modes

typedef enum{NORMAL,Phase\_PWM,CTC,FPWM}timer\_modes\_EN;

Provide selection for Prescalers of the frequency of the timer

typedef enum{STOP,NO\_PRESC,\_8\_PRESC,\_64\_PRESC,\_256\_PRESC, \_1024\_PRESC} Prescaler\_EN;

Macros:

This bits define the Prescaler

#define CS00 0

#define CS01 1

#define CS02 2

This bits define compare output mode

#define COM00 4

#define COM01 5

This bits define compare output mode

#define WGM00 6

#define WGM01 3

Timer/Counter Interrupt Mask

#define TOIE0 0

Functions:

* Timer0\_Init();

For Initializing the timers configuration and Selecting the Timer Mode and prescaler.

Inputs:

* + **Tmode** The mode for Timer0, it can be selected from timer\_modes\_EN enum.
  + **pres** is used to set he prescaler or set stop condition for Timer0, it can be selected from Prescaler\_EN enum.

Returning:

No Returning.

* Timer0\_start();

Makes the Counter Register start counting.

Inputs:

No Inputs.

Returning:

No Returning.

* Timer0\_Stop();

Makes the Counter Register stop counting.

Inputs:

No Inputs.

Returning:

No Returning.

* ResetTimer();

Resets the Counting and Counter Register to zero.

As It resets the global variable **overflow** which contains the overflow times done by counter register.

Inputs:

No Inputs.

Returning:

No Returning.

* ISR(TIMER0\_OVF\_vect);

ISR Function starts when counter register does overflow

It increments the global variable **overflow** which holds the overflow times.

Inputs:

No Inputs.

Returning:

No Returning.

* timer\_delay\_us();

Start busy Waiting (delay) for Period of time provided by input.

It uses ResetTimer() , Timer0\_start() and Timer0\_Stop() to start and stop counting.

It also uses a global variable **overflow** as holder for number of times of overflows done by counter register during the waiting period.

Inputs:

**delay** The period of time in microseconds.

Returning:

No Returning.

* Force\_Stop\_Timer0();

Stop the Busy Waiting by maximizing the global variable **overflow** which contains overflow times done by counter register.

This Function is used in ISR\_INT0 function to force the system to stop busy waiting during the interrupted timer\_delay\_us().

Inputs:

No Inputs.

Returning:

No Returning.

Global Variables:

* + **overflow**

is a static u32 variable that contains the overflow times done by counter register.

* + **prescaler**

is a static Prescaler\_EN enum variable that allow the user to select prescaler of timer0 {STOP,NO\_PRESC,\_8\_PRESC,\_64\_PRESC,\_256\_PRESC,\_1024\_PRESC}.

* + **Tick\_Time**

is a static double variable that hold the time period of one tick done by counter register.

* + **Overflow\_Time**
  + is a static double variable that hold the time period of one overflow done by counter register.

**Overflow\_Time** = 256 \* **Tick\_Time**.

* + **Maximum\_Overflow\_Times**

is a static u32 variable that hold the number of overflow times should be done until the **delay** time ends.

**delay** The period of time in microseconds which is input to timer\_delay\_us().

### ISR\_Interrupts driver

Enums and Typedefs:

No Enums or Typedefs.

Macros:

Enables the global interrupt

# define sei() \_\_asm\_\_ \_\_volatile\_\_ ("sei" ::: "memory")

Disables the global interrupt

# define cli() \_\_asm\_\_ \_\_volatile\_\_ ("cli" ::: "memory")

Defines IRQ0 Handler (external Interrupt INT0)

#define INT0\_vect \_\_vector\_1

Defines Timer0 Overflow Handler

#define TIMER0\_OVF\_vect \_\_vector\_11

A General Macro Function for ISRs

#define ISR(INT\_VECT) void INT\_VECT(void)\_\_attribute\_\_((signal,used));\

void INT\_VECT(void)

Functions:

No Functions.

Global Variables:

No Global Variables

### External\_Interrupt driver

Enums and Typedefs:

Provide Selection for the external interrupt number.

typedef enum {INT\_2=5,INT\_0,INT\_1}INT\_NUM;

Provide Selection for the Sense Control Mode.

typedef enum {low\_level,any\_level,rising\_edge,falling\_edge}SENSE\_CONTROL;

Macros:

No Macros.

Functions:

* INT\_init();

Initialize The configuration for the external interrupt.

As it Initialize the Interrupt Number and the Sense Control Mode

Inputs:

* + **int\_num** is parameter that indicate the number of external interrupt pin (INT0, INT1 or INT2), it can be selected from INT\_NUM enum.
  + **sense\_control** is a parameter that indicate the Sense Control Mode (Low Level, Any Level, Rising Edge, Falling Edge), it can be selected from SENSE\_CONTROL enum.

Returning:

No Returning.

Global Variables:

No Global Variables

## Application Layer

Enums and Typedefs:

Provide Selection for Current State in Normal Mode Yellow, Green, Yellow and Red.

typedef enum {Yellow\_BEFORE\_GREEN,GREEN,Yellow\_AFTER\_GREEN,RED}STATE\_type;

Macros:

#define PD\_PORT PORTA DIO Port of Pedestrians

#define PD\_DDR DDRA DIO DDR of Pedestrians

#define CR\_PORT PORTB DIO Port of Vehicles

#define CR\_DDR DDRB DIO DDR of Vehicles

#define BUTTON\_PIN INT0 Push Button external interrupt pin

#define PD\_RED\_PIN PA0 Pedestrians RED Led pin

#define PD\_YELLOW\_PIN PA1 Pedestrians YELLOW Led pin

#define PD\_GREEN\_PIN PA2 Pedestrians GREEN Led pin

#define CR\_RED\_PIN PB0 Vehicles RED Led pin

#define CR\_YELLOW\_PIN PB1 Vehicles YELLOW Led pin

#define CR\_GREEN\_PIN PB2 Vehicles GREEN Led pin

#define DELAY\_TIME (5000000) Delay Period (5 sec)

Functions:

* APP\_Start();

Initialize The Configuration for the System.

As it Initialize the External Interrupt 0 and Timer0 by calling INT\_init(INT\_0, rising\_edge); and Timer0\_Init(NORMAL);

It Initialize DDRA and DDRB pins as output that are used for the 6 LEDs used in the system.

Inputs:

No Inputs.

Returning:

No Returning.

* APP\_Run();

Starts the system by starting The Actions of the Normal Mode (YELLOW GREEN YELLOW RED).

And waits for the button pressing event so it enters the Pedestrian's Mode

Then goes back to the Normal Mode.

Inputs:

No Inputs.

Returning:

No Returning.

* Pedestrian\_Mode();

Starts the actions of Pedestrian Mode LEDs depending on the interrupted state and that is done by a Comparing Statement (if statement) and a Global Variable **state**.

Inputs:

No Inputs.

Returning:

No Returning.

* ISR(INT0\_vect)

ISR Function that runs when ISR0 event happen (the button pressing) it check if **pedestrian\_mode\_flag** global variable is zero before starting the pedestrian mode and modifies its value to 1 just before starting the pedestrian mode to prevent the double press effect.

This ISR happens at the rising edge so the long press has no effect.

It also modifies **ON\_Period** global variable to 0 to ignore next busy waiting in normal mode.

It also calls Force\_Stop\_Timer0() to stop the interrupted busy waiting function timer\_delay\_us().

Inputs:

No Inputs.

Returning:

No Returning.

Global Variables:

* **state** a flag variable that indicate the current state in Normal Mode (Yellow\_BEFORE\_GREEN,GREEN,Yellow\_AFTER\_GREEN,RED), it takes selection from STATE\_type enum.
* **pedestrian\_mode\_flag** a flag variable in its value in normal mode = 0 and that by ISR(INT0\_vect) it becomes 1 so the system enters the pedestrian mode during APP\_Run() running.
* **ON\_Period** is the value of on period of the LEDs which is basically 5 seconds, it becomes 0 by ISR(INT0\_vct) - when the event of button pressing happens - so the current state in normal mode ends immediately without busy waiting the rest of actions.