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.

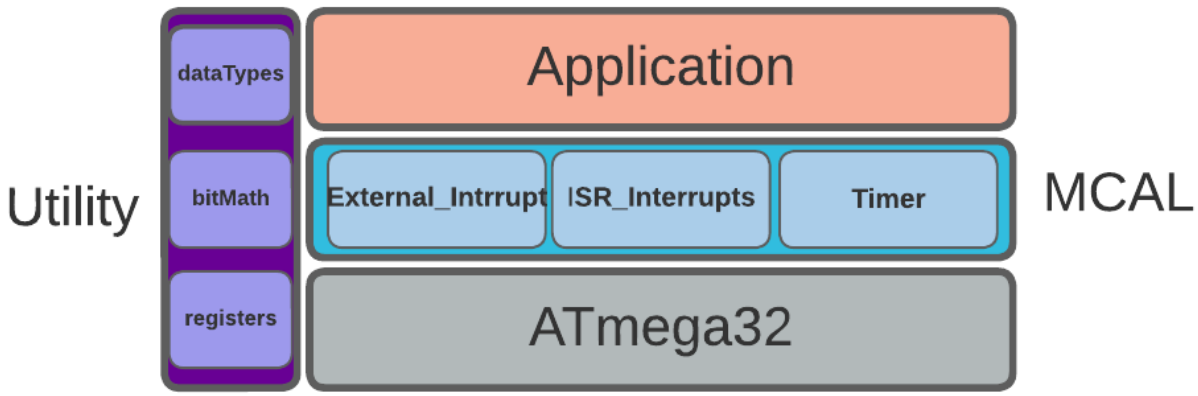
Used Hardware

- ATmega32 MCU
- 6 LEDs
- 1 Push Button

Software

- Microchip Studio for developing and debugging
- Proteus for Simulation

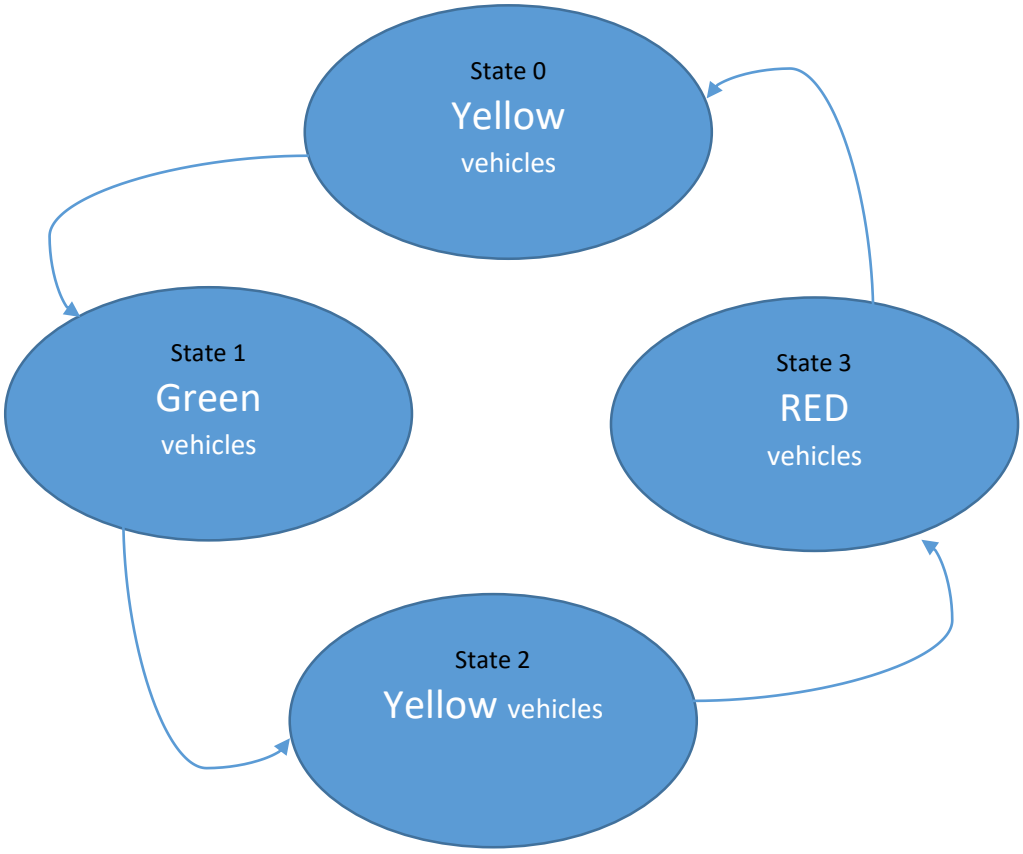
Software Architecture



Layers Architecture

System Design

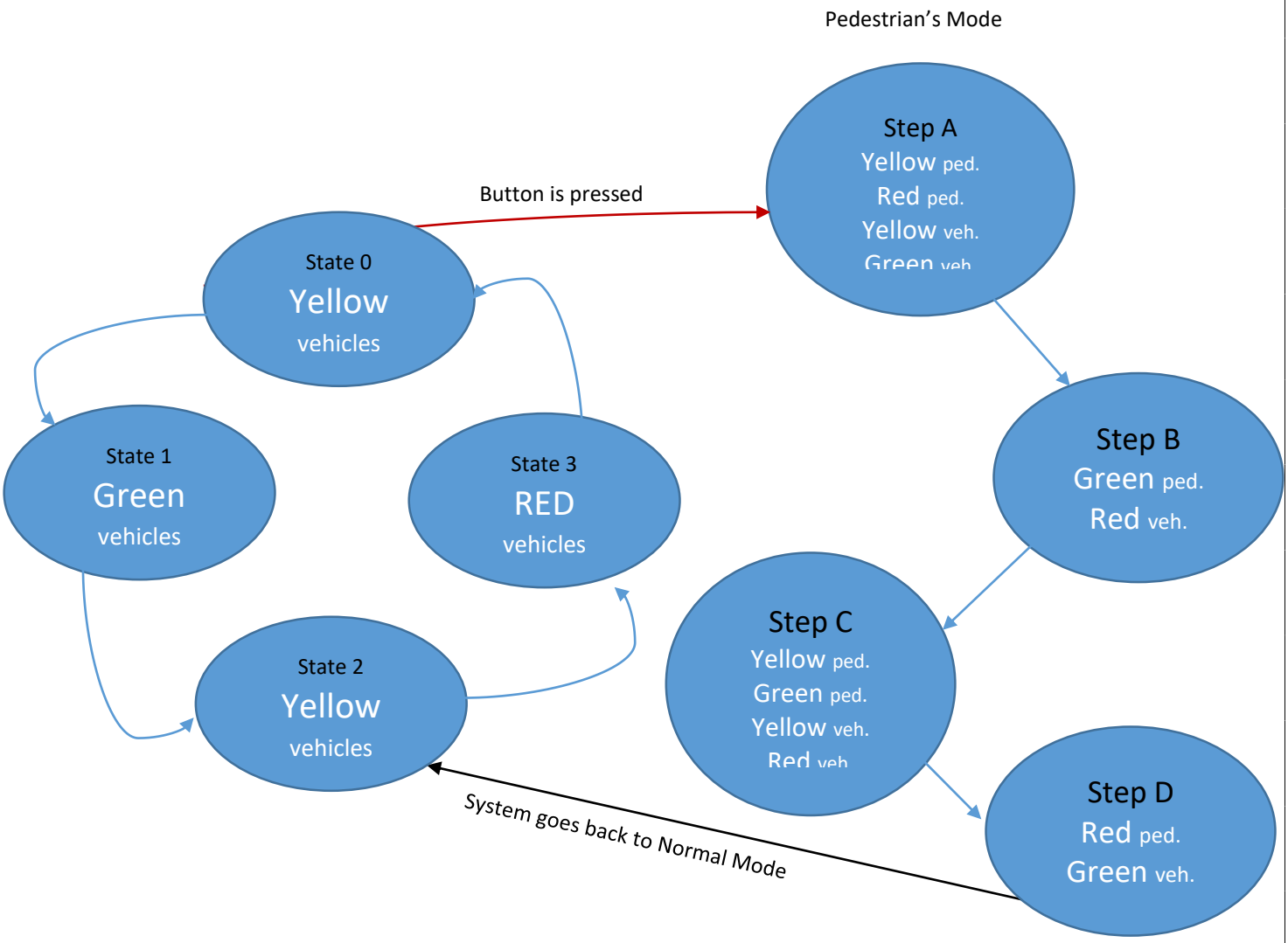
Normal Mode



Pedestrian's Mode

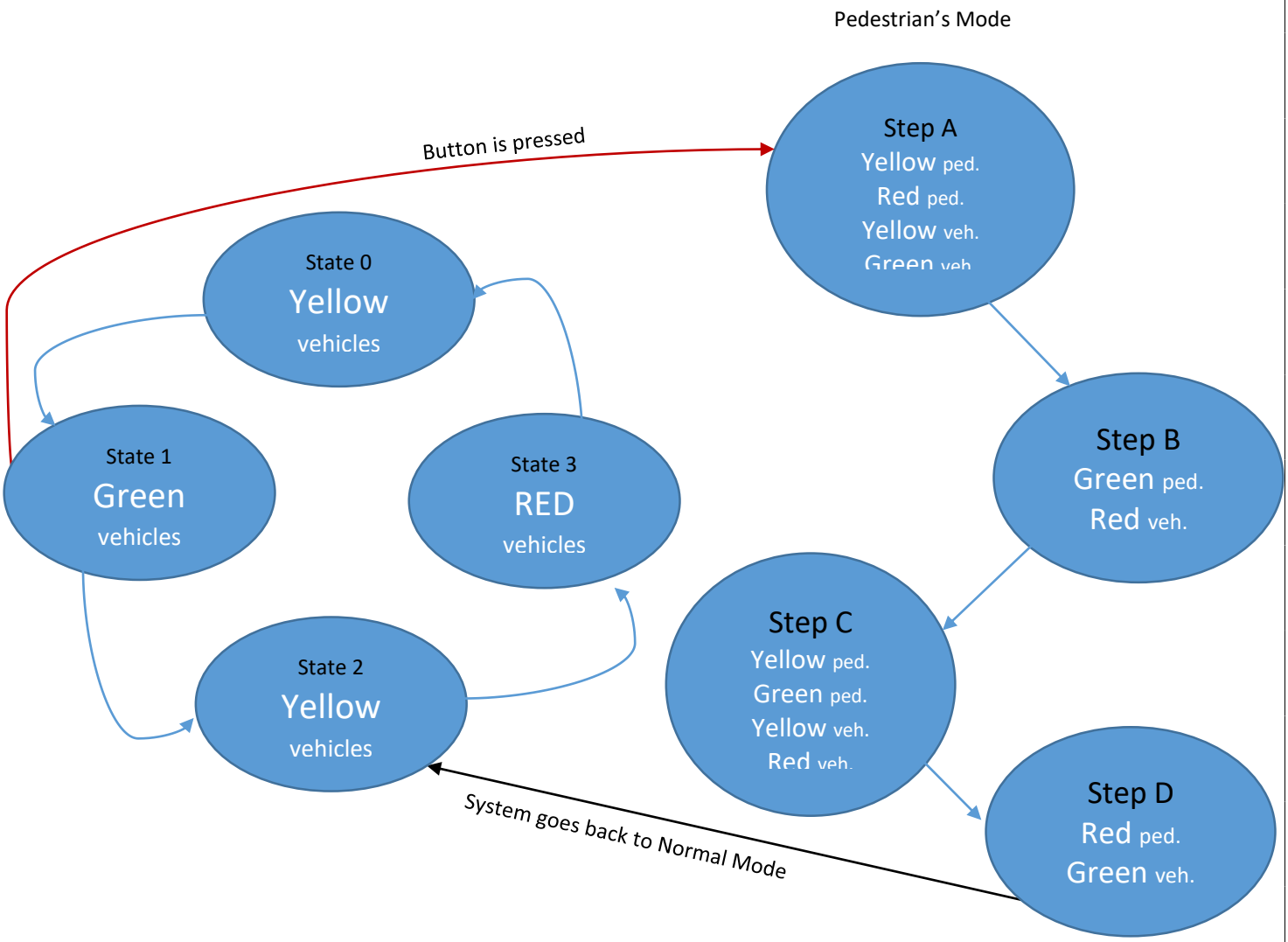
Scenario 1:

Button is Pressed while Yellow before Green is blinking.



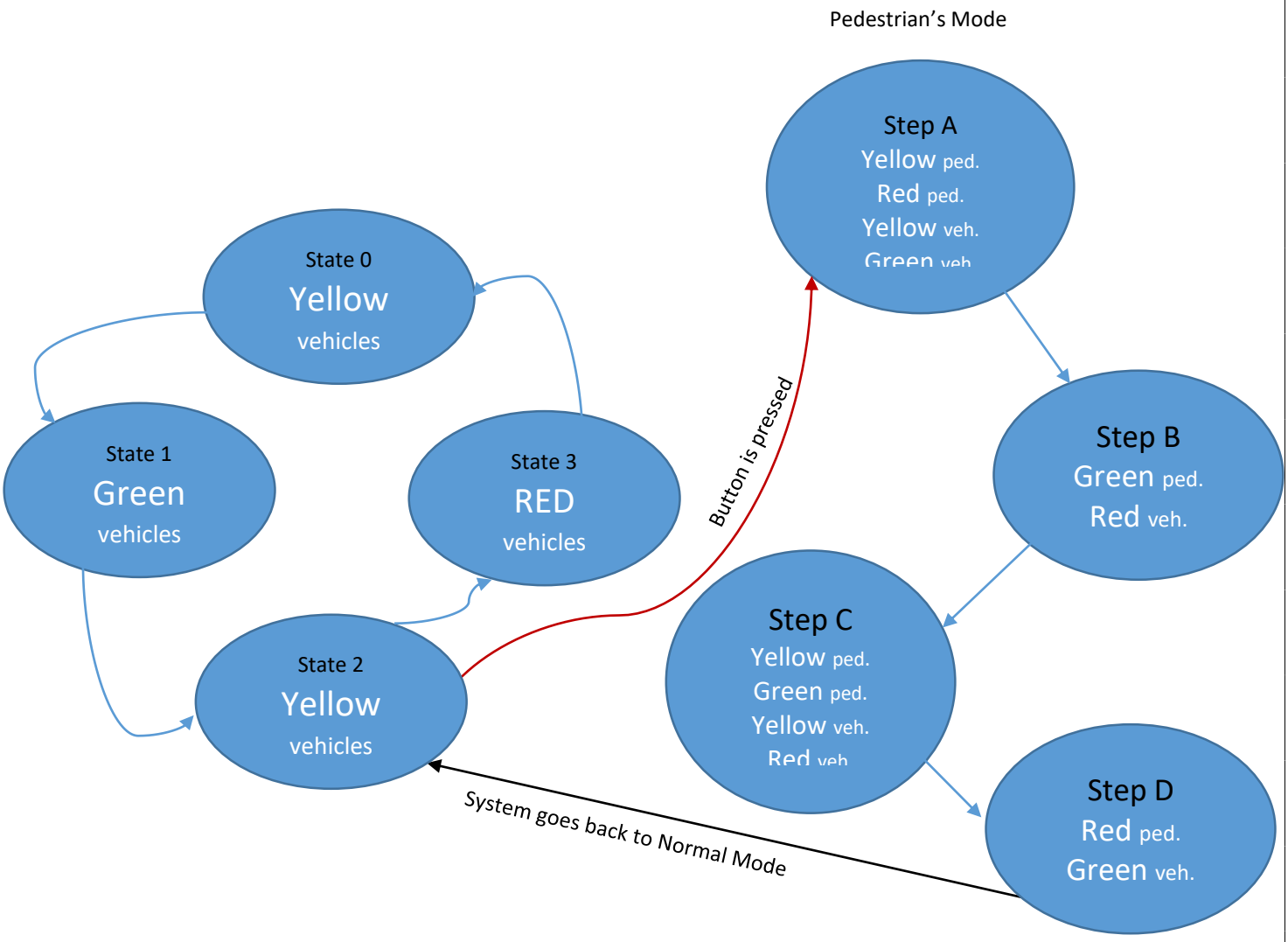
Scenario 2:

Button is Pressed while Green on.



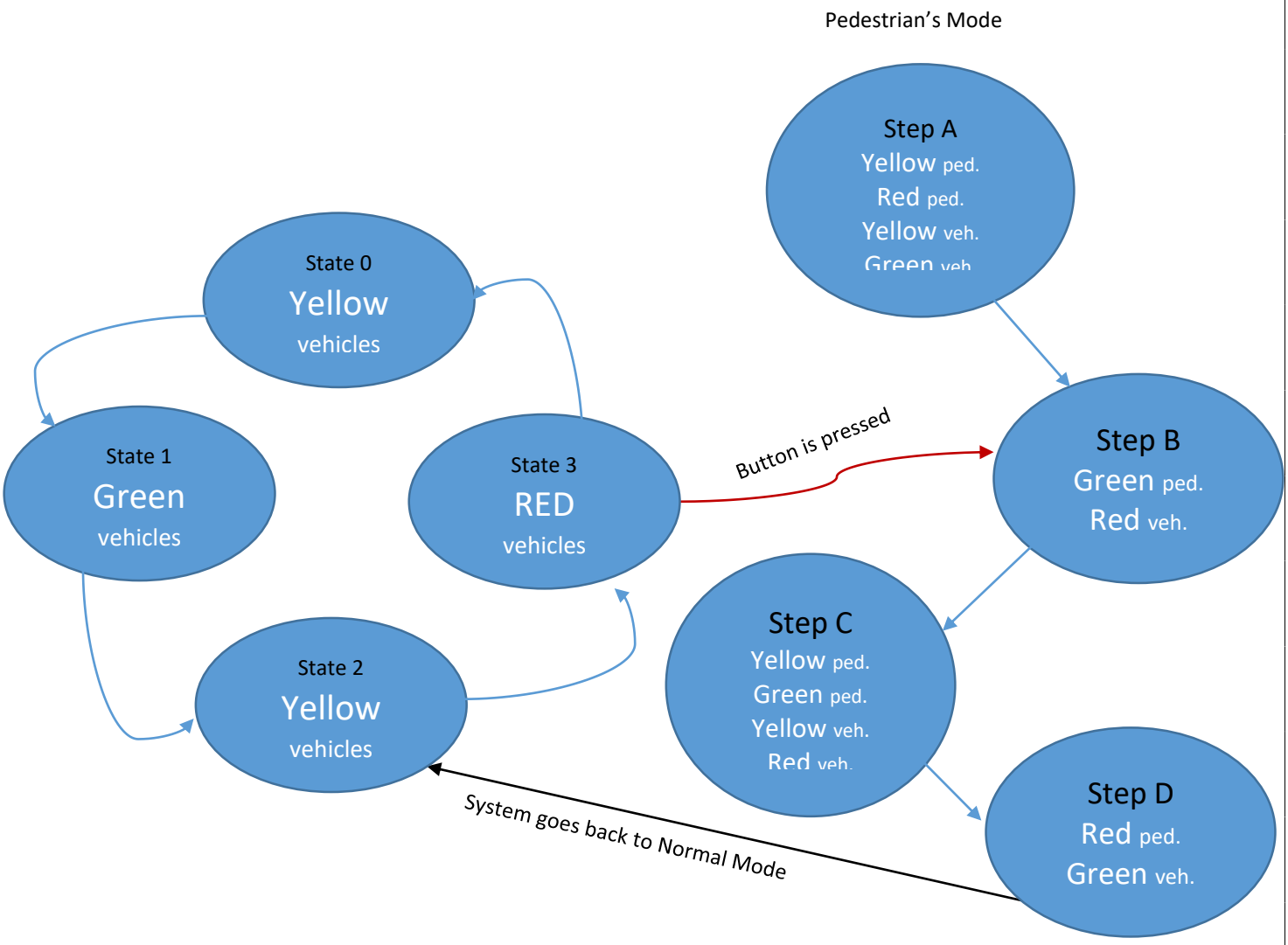
Scenario 3:

Button is Pressed while Yellow After Green is blinking.



Scenario 4:

Button is Pressed while Red is on.



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;

Provide selection for Prescalers of the frequency of the timer
typedef enum{STOP,NO_PRESC,_8_PRESC,_16_PRESC,_64_PRESC,_256_PRESC,
_1024_PRESC} Prescaler;
```

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 timer configuration and Selecting the Timer Mode.`

  Inputs:

  `Tmode` The mode for Timer0, it can be selected from `timer_modes` 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.

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.