# On-demand
# Traffic Light control

## Project Documentation

BY:

Fady Raouf Sobh Hana

# Five sections

## 1 – system layers

System consists of three main layers:

1- Micro Controller Abstraction Layer (MCAL) the lowest layer that having access to MCU Peripheral
2- Electronic Unit Abstraction Layer (ECUAL) the layer that come above MCAL layer that treat with I/O
3- Application Layer that required program will be there

## 2 – system drivers

In our system we use five drivers and some libraries

1- Dio drive
2- Timer0 drive
3- External interrupt drive
4- Led drive
5- Button drive
6- Two Libraries one for std_types and for math-bit

## 3-placing each drive into its layer

1-For MCAL it will contain (DIO-TIMER0-EX_INT_0)

2-for ECUAL it will contain (led -button)

# 4 – defining each driver

# For DIO

```
detect_dio DIO_init(uint8_t port, uint8_t pin, uint8_t direction) // it takes port and pin numbers and direction
{
        detect_dio state = OK;
        switch (port)
        {
                case DIOA:
                /* code */
                if(direction==IN) CLR_BIT(DDRA,pin); // clear bit
                else SET_BIT(DDRA,pin);  // set bit
                break;
                case DIOB:
                if(direction==IN) CLR_BIT(DDRB,pin); // clear bit
                else SET_BIT(DDRB,pin);  // set bit
                /* code */
                break;
                case DIOC:
                /* code */
                if(direction==IN) CLR_BIT(DDRC,pin); // clear bit
                else SET_BIT(DDRC,pin);  // set bit
                break;
                case DIOD:
                /* code */
                if(direction==IN) CLR_BIT(DDRD,pin); // clear bit
                else SET_BIT(DDRD,pin);  // set bit
                break;

                default:
                   state =ERROR;//reDIO error
                break;
        }
        return state;
}
detect_dio DIO_write(uint8_t port, uint8_t pin, uint8_t value) // it takes port and pin numbers and value
{
        detect_dio state = OK;
        switch (port)
        {
                case DIOA:
                /* code */
                if(value==LOW) CLR_BIT(PORTA,pin); // clear bit
                else SET_BIT(PORTA,pin);  // set bit
                break;
                case DIOB:
                if(value==LOW) CLR_BIT(PORTB,pin); // clear bit
                else SET_BIT(PORTB,pin);  // set bit
                /* code */
                break;
                case DIOC:
                /* code */
                if(value==LOW) CLR_BIT(PORTC,pin); // clear bit
                else SET_BIT(PORTC,pin);  // set bit
                break;
                case DIOD:
                /* code */
                if(value==LOW) CLR_BIT(PORTD,pin); // clear bit
                else SET_BIT(PORTD,pin);  // set bit
                break;

                default:
                 state =ERROR;//reDIO error
                break;
        }
        return state;
}

uint8_t DIO_read(uint8_t port, uint8_t pin)// it takes port and pin numbers
{
        uint8_t value=0;
        switch (port)
        {
                case DIOA:
                value =GET_BIT(PINA,pin);// get bit value in reg pina
                break;
                case DIOB:
                value =GET_BIT(PINB,pin);// get bit value in reg pinb
                break;
                case DIOC:
                value =GET_BIT(PINC,pin);// get bit value in reg pinc
                break;
                case DIOD:
                value =GET_BIT(PIND,pin);// get bit value in reg pinf
                break;

                default:
                //reDIO error
                break;
        }
        return value; // return value
}

detect_dio DIO_toggle(uint8_t port, uint8_t pin)
{
        detect_dio state=OK;
        switch(port){
                case DIOA:
                TOGGLE_BIT(PORTA,pin);// toggle bit  in reg porta
                break;

                case DIOB:
                TOGGLE_BIT(PORTB,pin);// toggle bit  in reg portb
                break;

                case DIOC:
                TOGGLE_BIT(PORTC,pin);// toggle bit  in reg portc
                break;

                case DIOD:
```

```
                TOGGLE_BIT(PORTD,pin);// toggle bit  in reg portd
                break;

                default:
                state=ERROR;//reDIO error
                break;
        }
        return state;
}
```

# For ex_inttrupt

```
void exterint_set()
{
                sei(); //to enable global interrupt
                SET_BIT(MCUCR,ISC00);//to make it work with rising edge
                SET_BIT(MCUCR,ISC01);
                SET_BIT(GIFR,INTF0);//clear flag
                SET_BIT(GICR,INT0);//enable interrupt
}
```

# For timer

```
void TIMER_init()
{
    CLR_BIT(TCCR0,WGM00); //normal mode
        CLR_BIT(TCCR0,WGM01);
}
detect_timer0 TIMER_delay(uint16_t ms)//takes the required delay time
{ detect_timer0 state =T_OK;
        if(ms<0) state=T_ERROR;
        uint16_t NOV,T_inital;
        double Tmaxdelay,Ttick;
        uint32_t OV_Count=0;

        Ttick = 256.0/1000.0;   // tick time is .256 ms as prescaler is 256
        Tmaxdelay= 65.536;      // time for max delay 65.536 as timer zero is 8 bit
counter

    if(ms == (int)Tmaxdelay)  //if delay user want equal to max delay they will be only
one overflow and tart with zero
    {
                T_inital=0;
                NOV=1;
        }
        else if(ms<Tmaxdelay)        //if delay user want less than max delay they will be
only one overflow and not start with zero
    {
                T_inital = (Tmaxdelay-ms)/Ttick;
                NOV = 1;
```

```
       }
    else                                 //if delay user want more than max delay they will 
be more than one overflow and not start with zero
    {
              NOV = ceil((double)ms/Tmaxdelay);
              T_inital = (1<<8) - ((double)ms/Ttick)/NOV;
       }

       TCNT0 = T_inital; // after detecting which one of the three condation > or = or < 
max delay timer will start with the value
           SET_BIT(TCCR0,CS02); //set 256 prescaler
       while(OV_Count<NOV) // in case delay was greater than max delay so it will be more 
thaan one overflow
    {
              while(GET_BIT(TIFR,0)==0);//busy wait
              SET_BIT(TIFR,0);//clear overflow flag
              OV_Count++;//increment counter
       }

       return state;//return state

}
```

# For led

```
detect_dio LED_init(uint8_t Port,uint8_t Pin) // takes port and pin number
{
       detect_dio state=OK;
       if(DIO_init(Port,Pin,OUT)==ERROR) state =ERROR;// init direction to be output
       return state;//  return state
}
detect_dio LED_on(uint8_t Port,uint8_t Pin)// takes port and pin number
{
              detect_dio state=OK;
              if(DIO_write(Port,Pin,HIGH)==ERROR) state =ERROR;// to write high on led 
bit
              return state;
}
detect_dio LED_off(uint8_t Port,uint8_t Pin)// takes port and pin number
{
              detect_dio state=OK;
              if(DIO_write(Port,Pin,LOW)==ERROR) state =ERROR;// to write low on led bit
              return state;
}
detect_dio LED_toggle(uint8_t Port,uint8_t Pin)// takes port and pin number
{
              detect_dio state=OK;
              if(DIO_toggle(Port,Pin)==ERROR) state =ERROR;// to blink led
              return state;
}
detect_timer0 LED_timer_setup(uint16_t ms) // takes port and pin number
{
       detect_dio state=T_OK;
       if(TIMER_delay(ms)==T_ERROR) state =T_ERROR;// start timer with ms
       return state;//  return state
}
```

```
void led_timer_on()
{
	TIMER_init();
}
void led_timer_end()
{
	NOV =0;
}}
```

## For button

```
detect_dio BUTTON_init(uint8_t Port,uint8_t Pin)// takes port and pin number
{
		detect_dio state=OK;
		exterint_set(); //enable external interrupt to read buuton value
		if(DIO_init(Port,Pin,IN)==ERROR) state =ERROR;// init direction to be
output
		return state;// return state
}

// button read
uint8_t BUTTON_read(uint8_t Port,uint8_t Pin)// takes port and pin number
{
	return DIO_read(Port,Pin); // get bit
}
```

# 5-New data type :

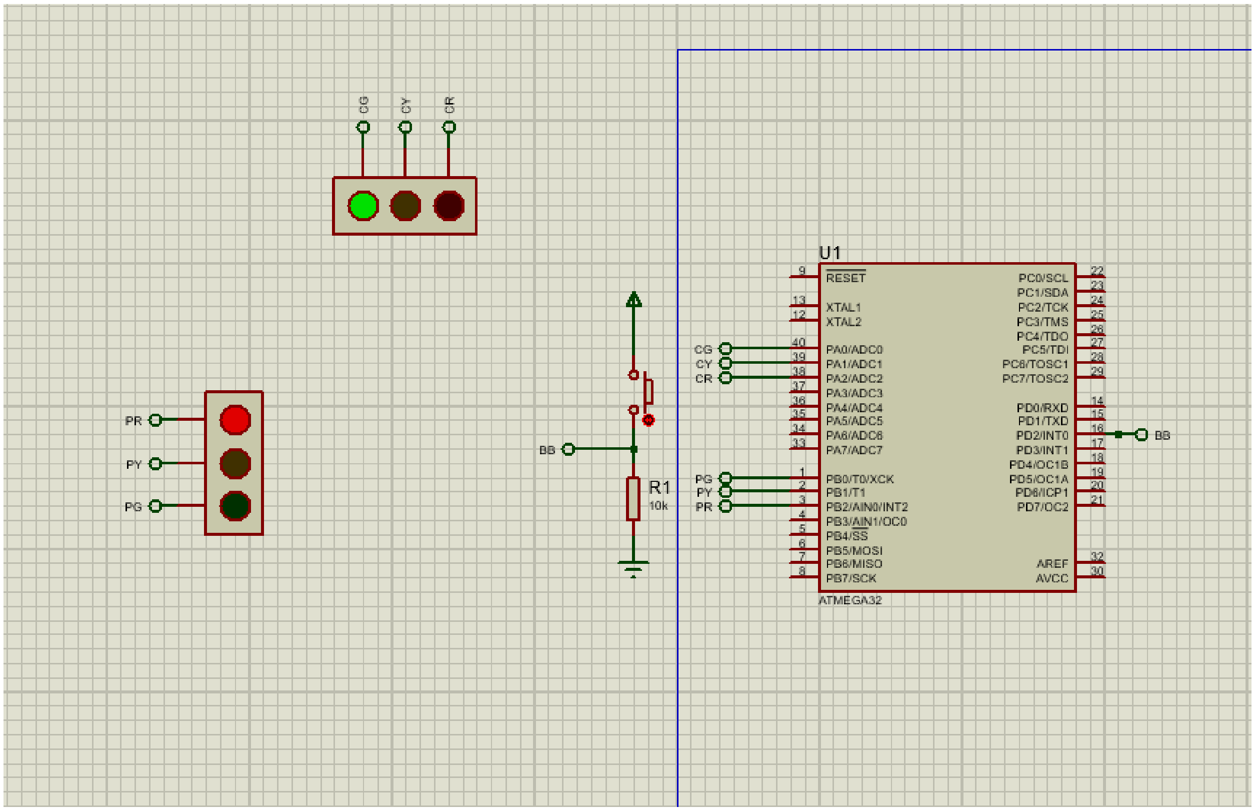
```
typedef enum
{
	OK,
	ERROR
}detect_dio;

typedef enum
{
	T_OK,
	T_ERROR
```

}detect_timer;

For checking if error occurred

# System description:



## Overview:

The purpose of the system is to control traffic lights and to allow people to pass between cars through adding pedestrians mode

## Functionality:

As traditional traffic lights control system it has 3 colors switches between them every 5 seconds

(green – yellow -red -yellow -green -… and so on)

But by adding pedestrians’ button this help people to pass street safely as when the button is pressed if it is pressed while cars’ led was on that mean that green led for pedestrians will be on for five seconds so people can pass the street then both yellow led will be on for 5 seconds after 5 second they will be off then cars’ green led and pedestrians’ red led will be on for five seconds after that system will go back to normal mode

In case of  pedestrians button was pressed while cars’ green or yellow was on that mean people can’t pass the street so pedestrians’ red led will be on and both yellow led will be on for 5 seconds after that they will be off and pedestrians green led and cars red led will be on for five seconds then the to yellow be on as cars’ red led will be off and pedestrians still on with yellow blinks after five seconds cars’ green led will be on and pedestrians red led will be off for five seconds after that system will go back to normal mode

In case of button was pressed twice it doesn’t make effect or if it was pressed for long time

# System design:

**The system consists of:**

1-AVR Atmega32 (1MHz)

2- 1 Push Button

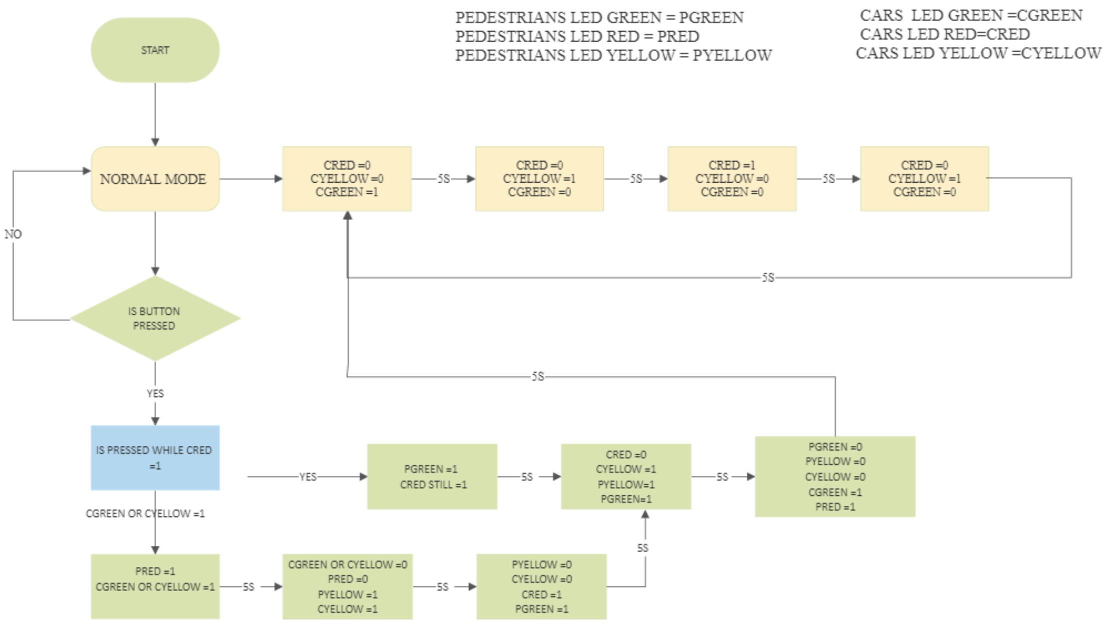
3- 2 traffic lights led

**Operating Environment:**

1-Portus simulator

2-Atmel studio

# System flowchart:


PEDESTRIANS LED GREEN = PGREEN
PEDESTRIANS LED RED = PRED
PEDESTRIANS LED YELLOW = PYELLOW
CARS LED GREEN =CGREEN
CARS LED RED=CRED
CARS LED YELLOW =CYELLOW
START
NORMAL MODE
CRED =0 CYELLOW =0 CGREEN =1
5S
CRED =0 CYELLOW =1 CGREEN =0
5S
CRED =1 CYELLOW =0 CGREEN =0
5S
CRED =0 CYELLOW =1 CGREEN =0
5S
NO
IS BUTTON PRESSED
YES
5S
IS PRESSED WHILE CRED =1
YES
PGREEN =1 CRED STILL =1
5S
CRED =0 CYELLOW =1 PYELLOW=1 PGREEN=1
5S
PGREEN =0 PYELLOW =0 CYELLOW =0 CGREEN =1 PRED =1
CGREEN OR CYELLOW =1
5S
PRED =1 CGREEN OR CYELLOW =1
5S
CGREEN OR CYELLOW =0 PRED =0 PYELLOW =1 CYELLOW =1
5S
PYELLOW =0 CYELLOW =0 CRED =1 PGREEN =1

# system constraints

**Accessibility:** any wayfarer can push the button to pass the street

**Low Overhead:** as system there's no additional function overhead the system

**Minimum Hassle:** to reduce it we make long press button doesn't make effect as double press