9/13/2022

On-demand Traffic Light Control

Ibrahim Mohamed Hamdy Hassan

1. System Description

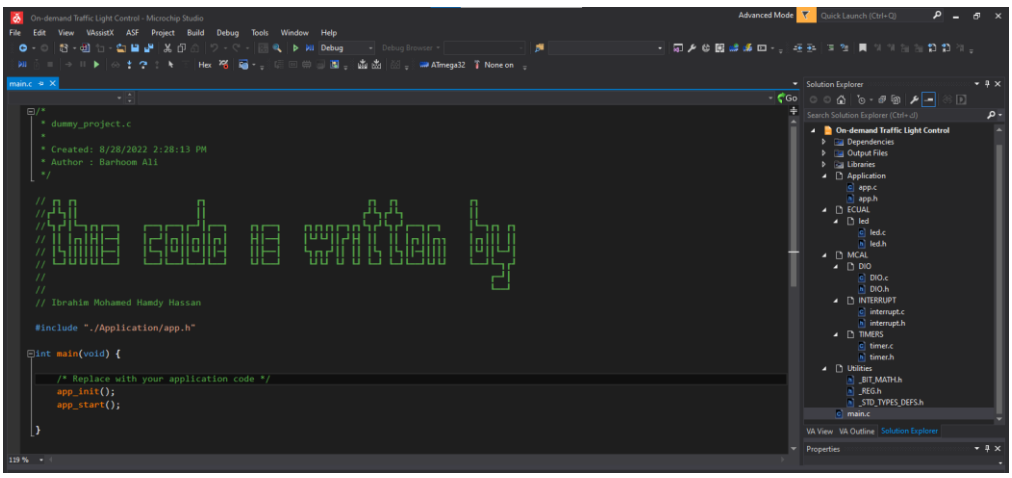
It is required to make a traffic light with a button 3 LEDs for cars and 3 LEDs for pedestrians, as when the red LED for the cars is on the green one for pedestrians is off and vice versa, when the yellow LED for cars is blinking, the yellow one for pedestrians would blink as well.

Pedestrians have the high priority in this project so if they pushed the button the controller will check for the traffic light state as if the state was green or (yellow and the next is green), it will turn off all the LEDs then start blinking both yellow LEDs then turning the red LED for cars and green LED for pedestrians then complete, but if the state is in red for cars or (yellow and the next state is red), it will ignore this action.

2. System Design

2.1. System Layers and Drivers

Our system consists of 4 basic layers with 3 shared APIs as shown below:

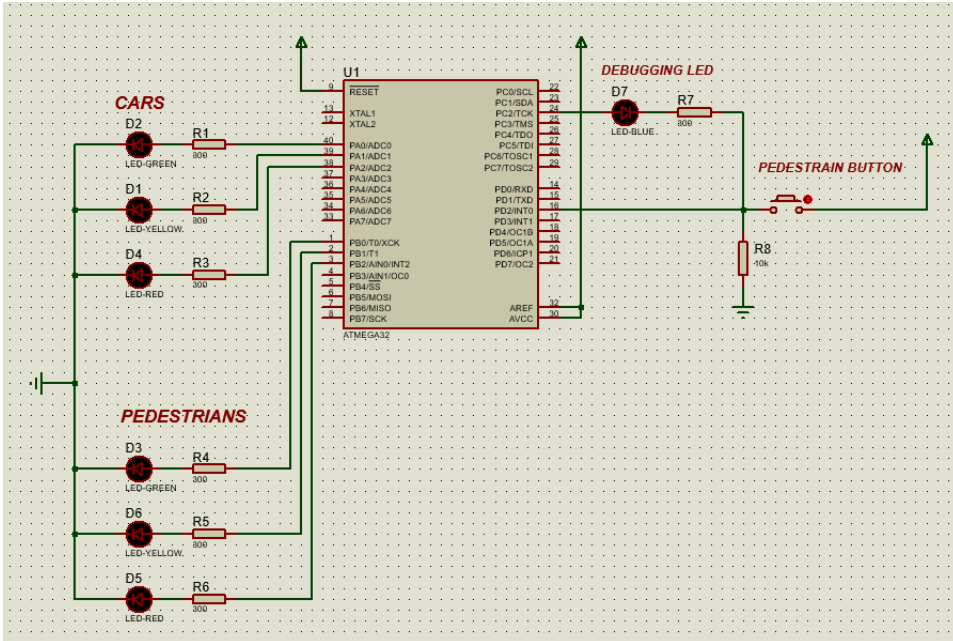


2.1.1. Microcontroller Layer:

This layer contains the hardware components that are connected to make up the circuit we need, and those components are:

- *ATMEGA32 microcontroller.*
- *6 LEDs (2 green LEDs, 2 yellow LEDs, and 2 red LEDs).*
- *Push button.*
- *7 resistors (6 of 300-ohm resistors and a 10-Kohm resistor).*

The schematic used:



2.1.2. Microcontroller Abstraction Layer (MCAL):

This layer is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Drivers used in this project are:

- *DIO Driver:*

  This driver performs port signal input/output, etc.

```
DIO.h   DIO.c
MCAL > DIO > DIO.h > ...
/* ...

// ...

#ifndef _DIO_H_
#define _DIO_H_

#include "../../Utilities/_REG.h"
#include "../../Utilities/_BIT_MATH.h"
#include "../../Utilities/_STD_TYPES_DEFS.h"

void DIO_init(uint8_t port, uint8_t pin, uint8_t direction);
void DIO_write(uint8_t port, uint8_t pin, uint8_t value);
void DIO_toggle(uint8_t port, uint8_t pin);
uint8_t DIO_read(uint8_t port, uint8_t pin);

#endif /* _DIO_H_ */
```

- *TIMERS Driver:*

  This driver performs delay functions, PWM signals, counting the number of actions happening, etc.

  In this project, we use timers only in normal mode so this driver is used for performing delays only.

```
timer.h   timer.c
MCAL > TIMERS > timer.h > ...
/* ...

// ...

#ifndef TIMER_H_
#define TIMER_H_

#include "../../Utilities/_BIT_MATH.h"
#include "../../Utilities/_REG.h"
#include "../../Utilities/_STD_TYPES_DEFS.h"

void timer_init(uint8_t timer);

void timer_start(uint8_t timer, uint16_t delay_ms, uint16_t prescaler);

void timer_stop(uint8_t timer);

uint16_t get_timer_value(uint8_t timer);

#endif /* TIMER_H_ */
```

- *INTERRUPT Driver:*
  This driver performs immediate attention to an event once the MCU receives a specific signal generated by hardware or software.

```
interrupt.h ×   interrupt.c
MCAL > INTERRUPT > interrupt.h > ...
1  /* ...
7
8  // ...
17
18 #ifndef INTERRUPT_H_
19 #define INTERRUPT_H_
20
21 #include "../../Utilities/_BIT_MATH.h"
22 #include "../../Utilities/_REG.h"
23 #include "../../Utilities/_STD_TYPES_DEFS.h"
24
25 typedef struct ST_EXT_INT_t {
26     uint8_t interruptPinSelect;
27     uint8_t senseControl;
28     void (* function) (void);
29 } ST_EXT_INT_t;
30
31 void EXT_INT_init(ST_EXT_INT_t interrupt);
32
33 #endif /* INTERRUPT_H_ */
34
```

2.1.3. Electronics Unit Abstraction Layer (ECUAL):

An ECUAL driver is a set of functions that initialized the MCU hardware via the MCAL and does the calling of MCAL functions, necessary calculations, algorithms, and utilities, to abstract the hardware handling from the application layer. So, the application code doesn't talk directly to DIO or PWM or whatever.

- *LED driver:*
  This driver performs a set of functions that control a specific LED.

```
led.h ×   led.c
ECUAL > led > led.h > ...
1  /* ...
7
8  // ...
17
18 #ifndef LED_H_
19 #define LED_H_
20
21 #include "../../MCAL/DIO/DIO.h"
22
23 void LED_init(uint8_t ledPort, uint8_t ledPin);
24
25 void LED_ON(uint8_t ledPort, uint8_t ledPin);
26
27 void LED_OFF(uint8_t ledPort, uint8_t ledPin);
28
29 void LED_toggle(uint8_t ledPort, uint8_t ledPin);
30
31 #endif /* LED_H_ */
32
```

2.1.4. Application Layer:

This layer performs the program's code and shows the output of it.

```
app.h ×   app.c
Application > app.h > ...
1  /* ...
7
8  // ...
17
18 #ifndef APP_H_
19 #define APP_H_
20
21 #include "../ECUAL/led/led.h"
22 #include "../MCAL/TIMERS/timer.h"
23 #include "../MCAL/INTERRUPT/interrupt.h"
24
25 void app_init();
26
27 void app_start();
28
29 #endif /* APP_H_ */
30
```

3. System Flowchart