Introduction to Embedded Systems Traffic Lights Project

Team 2 – G2 S1

Presented to:

Prof. Sherif Hammad

Eng. Ayman Ahmed

Eng. Ahmed Farouk

Made By:

Shehab El Din Adel – 18P3863

Ziad Yakan – 18P9538

Mostafa Shams – 18P6781

Daniel Tarek – 18P1185

Abdelrahman Shemis – 18P9565

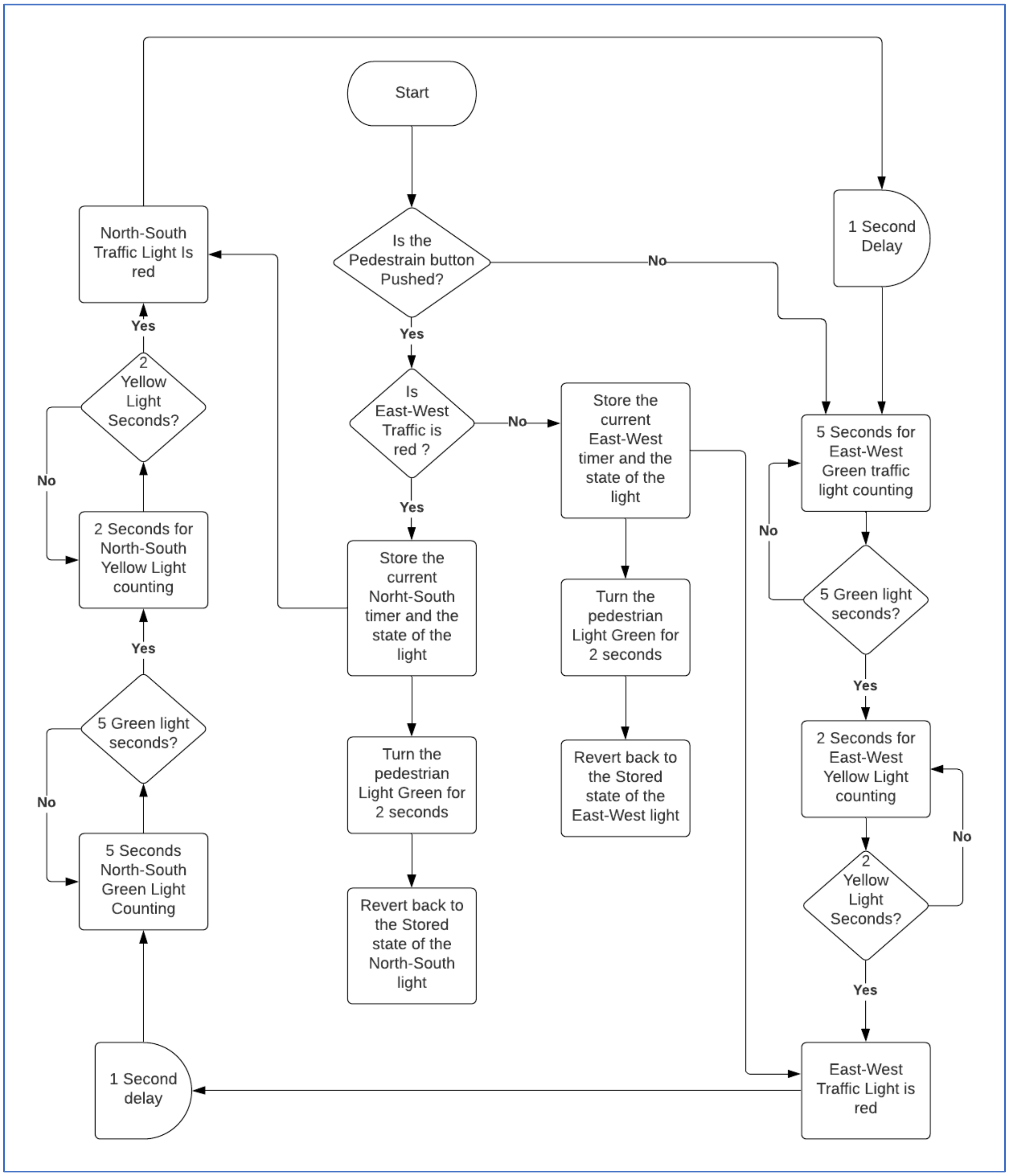
Introduction

The goal of this project is to design a traffic light control system. The system contains simply two traffic lights. One allows cars to move from north to south, and the other one allows cars to move from east to west. Beside each traffic there is also a pedestrian traffic light. The pedestrian has to press on a button to have his light green to cross the street safely.

In this project, we mainly relayed on the TivaWare API functions, so that it can make it easier to implement the project and to have one format for our project.

The functions documentation is inside the header files in the source code. Each function has its own parameters, return type, and description.

Project’s Flowchart


Start
Is the Pedestrain button Pushed?
No
Yes
Is East-West Traffic is red ?
No
Yes
Store the current East-West timer and the state of the light
Turn the pedestrian Light Green for 2 seconds
Revert back to the Stored state of the East-West light
Store the current Norht-South timer and the state of the light
Turn the pedestrian Light Green for 2 seconds
Revert back to the Stored state of the North-South light
North-South Traffic Light Is red
Yes
2 Yellow Light Seconds?
No
2 Seconds for North-South Yellow Light counting
Yes
5 Green light seconds?
No
5 Seconds North-South Green Light Counting
1 Second delay
1 Second Delay
5 Seconds for East-West Green traffic light counting
No
5 Green light seconds?
Yes
2 Seconds for East-West Yellow Light counting
No
2 Yellow Light Seconds?
Yes
East-West Traffic Light is red

Project’s Structure

Definitions and Macros

Buttons

```
#define PedButtons GPIO_PORTF_BASE
#define ButtonEW GPIO_PIN_0
#define ButtonNS GPIO_PIN_4
```

Car Traffic

```
#define Car_BASE GPIO_PORTA_BASE

#define GreenNS GPIO_PIN_2
#define YellowNS GPIO_PIN_3
#define RedNS GPIO_PIN_4

#define GreenEW GPIO_PIN_5
#define YellowEW GPIO_PIN_6
#define RedEW GPIO_PIN_7

#define Led_CarGreenEW (RedNS | GreenEW)
#define Led_CarYellowEW (RedNS | YellowEW)
#define Led_CarRedEW (RedNS | RedEW)
#define Led_CarGreenNS (GreenNS | RedEW)
#define Led_CarYellowNS (YellowNS | RedEW)
#define Led_CarRedNS (RedNS | RedEW)

#define FSM_CarGreenEW (0)
#define FSM_CarYellowEW (1)
#define FSM_CarRedEW (2)
#define FSM_CarGreenNS (3)
#define FSM_CarYellowNS (4)
#define FSM_CarRedNS (5)
```

Pedestrian Traffic

```
#define Ped_BASE GPIO_PORTC_BASE
#define PedGreen_NS GPIO_PIN_4
#define PedRed_NS GPIO_PIN_5
#define PedGreen_EW GPIO_PIN_6
#define PedRed_EW GPIO_PIN_7

#define Led_PedGreenNS (PedGreen_NS | PedRed_EW)
#define Led_PedRedNS (PedRed_NS | PedRed_EW)
#define Led_PedGreenEW (PedRed_NS | PedGreen_EW)
#define Led_PedRedEW (PedRed_NS | PedRed_EW)
```

```
#define FSM_PedGreen_EW (0)
#define FSM_PedRed_EW (1)
#define FSM_PedGreen_NS (2)
#define FSM_PedRed_NS (3)
```

Timers

```
#define CarTimer TIMER1_BASE
#define PedTimer TIMER2_BASE
```

Buttons

In this module we start by initializing the buttons that are used inside our projects, and also defining their interrupt handlers since they are related to the Pedestrian Traffic Lights.

initButtons

In this function we firstly set the interrupt priority of the buttons, in order to make sure that their handle functions will be handled above any other handlers. Then we start assigning the system's clock to the GPIO PortF, then unlock the Pins used which are Pin 0 and Pin 4. Then, we defined ButtonEW and ButtonNS to be input GPIOs.

ButtonEW →Pin 0

ButtonNS → Pin 4

Then we configure both buttons as pull-up and enable interrupts on both buttons, and finally configure their Interrupt handler which is the PedButtonINT function.

PedButtonINT

In this method we check if either ButtonEW or ButtonNS is pressed, and based on it we take the following actions:

1. Clear interrupt from the pressed button
2. Disable the CarTraffic Timer in order to continue it later from its state
3. Change the Pedestrian state to Green
4. Change the color of the car Traffic to Red based on the output of the FSM_PED_State Car output
5. Change the colors of the Pedestrian Traffic to Green based on the FED_PED_State pedestrian output
6. Wait for 2 seconds
7. Enable the CarTraffic timer back again

CarTraffic

Check CarTraffic.c for brief explanation

In this module we defined a struct called CarFSM which contains metadata about a state of Car Traffic Light like the current output which represents the bits for the Car Traffic Led, the time delay that the Car Traffic state takes, and finally its next state that will move to.

```
struct CarFSM
{
  uint8_t Out;   // output bits for car traffic led
  uint16_t Time; // delay in Milli Second
  uint8_t Next;  // The next state
};
```

initCarLight

Also we have a method called initCarLight which is used in initializing the Car Traffic Lights by the following steps

1. Assigning the clock to GPIO PortA
2. Setting the Pins 2,3,4,5,6,7 as output pins
3. Starting the delay of the CarTrafficLight by providing it the Time variable of the CarFSM as an argument.

FSM_CarTL

We created instances from the CarFSM struct called FSM_CarTL array which has all our Car Traffic combinations. For example, the following state outputs Green light, has a delay of 5 seconds, and has a next state which is FSM_CarYellowEW

```
{Led_CarGreenEW, 5000, FSM_CarYellowEW}
```

While the FSM_CarYellowEW has the following properties which are

1) Turning on the yellow traffic light
2) Waiting for 2 seconds
3) Assigning FSM_CarRedEW as a next state.

```
{Led_CarYellowEW, 2000, FSM_CarRedEW}
```

PedTraffic

Check PedTraffic.c for brief explanation

In this module we defined a struct called PedFSM which contains metadata about a state of Pedestrian Traffic like the current output of the Pedestrian Traffic Leds, current **outputs** of the Car Traffic Leds, and the next state.

```
struct PedFSM
{
  uint8_t PedOut; // output of Pedestrian Traffic Leds
  uint8_t CarOut[6]; // output of Car Traffic Leds
  uint8_t Next; // Next State
};
```

initPedLight

In this method, we used it in initializing the Ped Traffic Lights by the following steps

1. Assigning the clock to GPIO PortC
2. Setting Pins 4,5,6,7 as output pins

FSM_PedTL

We created instances of the PedFSM struct called FSM_PedTL array which has the combination states of the Pedestrian Traffic. For example, the following element inside the array has Green light output in the East West Direction, and an array of Car Traffic states based on the current Pedestrian State, and the next Pedestrian State.

```
Led_PedGreenEW, {Led_CarRedEW, Led_CarRedEW, Led_CarRedEW, Led_CarGreenNS,
Led_CarYellowNS, Led_CarRedNS}, FSM_PedRed_EW
```

Timers

initCarTimer and initPedTimer

In these method, we used them in initializing the Car and Pedestrian Timers by the following steps

Timer 1 → Car Traffic Timer

Timer 2 → Pedestrian Timer

1. Setting the lowest priority interrupt for the Car Timer interrupt and setting the highest priority interrupt for the Pedestrian Timer Interrupt
2. Enabling the timer
3. Disabling the Timer before configuration
4. Configuring the timer as a down periodic timer
5. Enabling interrupt on the timer
6. Enabling the timer

CarTimer_Delay and PedTimer_Delay

Each of these delays takes time in milliseconds as an argument and set the Timer Load value of either CarTimer or PedTimer by the following equation

$$(delay * 16000) - 1$$

PedTimer_TIMEOUT

Inside the Pedestrian Timer Handler, we first disable the Car Traffic timer so we can continue it later and clear the interrupt of the Pedestrian Timer. Secondly, we change the state of the Pedestrian Traffic Light to its next state. Finally, we turn on the traffic lights of the Pedestrian and Car Traffic Lights based on their outputs inside the FSM_PedTL struct. Then, we enable back the Car Traffic Timer.

CarTimer_TIMEOUT

In this method we do the following steps

1) Clear the Car Timer interrupt flag
2) Change the FSM_Car_State to the next state based on the next state value inside the struct instance
3) Start the CarTraffic_Delay by providing it the delay value from the struct instance
4) Turn on the Car Traffic Lights
5) Enable the Car Timer back again

Main

```
__asm("CPSID I");
  SysCtlDelay(3);
  initCarTimer();
  initCarLight();
  initPedTimer();
  initButtons();
  initPedLights();

  __asm("CPSIE I");

  while (1)
  {
    SysCtlSleep();
  }
```

In the main function we do the following steps.

1) Change Processor Status Interrupt Disable, which allows the processor to ignore interrupts
2) Wait for 3 milliseconds
3) Initialize the Car Timer
4) Initialize the Car Traffic Lights
5) Initialize the Pedestrian Timer
6) Initialize the Buttons
7) Initialize the Pedestrian Lights
8) Change Processor Status Interrupt Enable, which allows the processor to accept interrupts
9) Sleep the processor and wait for interrupts

Conclusion

In the end, we were able to deliver a perfectly working traffic light control system using TivaWare and Arm Cortex M4 TivaC. We recommend looking at the source code header files for other brief explanation of some functions. We appreciate your patience and consideration for looking at our project's documentation and your extreme hard work with us during the semester.

Thank you!