Alexis Nhan
EGEC 450

Question 1:

```
#include <stdint.h>
#include "SysTick.h"
#include "PLL.h"
#include "tm4c123gh6pm.h"

struct State {
	uint32_t Out;		// output
	uint32_t Walk_Light;	// walk
	uint32_t Time;		// how long to wait
	const struct State *Next[8];
}; // points to next state

typedef const struct State STyp;

#define goS		&FSM[0]
#define waitS		&FSM[1]
#define goW		&FSM[2]
#define waitW		&FSM[3]
#define waitWalkS	&FSM[4]
#define walkS		&FSM[5]
#define bWalkS		&FSM[6]
#define noBWalkS	&FSM[7]
#define bWalkS1		&FSM[8]
#define noBWalkS1	&FSM[9]
#define bWalkS2		&FSM[10]
#define noBWalkS2	&FSM[11]
#define bWalkS3		&FSM[12]
#define waitWalkW	&FSM[13]
#define walkW		&FSM[14]
#define bWalkW		&FSM[15]
#define noBWalkW	&FSM[16]
#define bWalkW1		&FSM[17]
#define noBWalkW1	&FSM[18]
#define bWalkW2		&FSM[19]
#define noBWalkW2	&FSM[20]
#define bWalkW3		&FSM[21]

#define GreenLED	0x08
#define RedLED		0x02
```

Alexis Nhan
EGEC 450

```
STyp FSM[22]={
	// no walk
	{0b100001, RedLED, 300,
{goS,waitS,goS,waitS,waitWalkS,waitWalkS,waitWalkS,waitWalkS}},			// goS
	{0b100010, RedLED,  50, {goW,goW,goW,goW,goW,goW,goW,goW}},
// waitS
	{0b001100, RedLED, 300, {goW,goW,waitW,waitW,waitW,waitW,waitW,waitW}},
// goW
	{0b010100, RedLED,  50, {goS,goS,goS,goS,goS,goS,goS,goS}},
// waitW

	// walking S
	{0b100010, RedLED,  50, {goS,waitS,walkS,waitS,walkS,walkS,walkS,walkS}},
// waitWalkS
	{0b100010, GreenLED,  50,
{bWalkS,bWalkS,bWalkS,bWalkS,bWalkS,bWalkS,bWalkS,bWalkS}},			// walkS
	{0b100100, RedLED,  10,
{noBWalkS,noBWalkS,noBWalkS,noBWalkS,noBWalkS,noBWalkS,noBWalkS,noBWalkS}},
// BWalkS
	{0b100100, 0x00,  10,
{bWalkS1,bWalkS1,bWalkS1,bWalkS1,bWalkS1,bWalkS1,bWalkS1,bWalkS1}},		//
noBWalkS
	{0b100100, RedLED,  10,
{noBWalkS1,noBWalkS1,noBWalkS1,noBWalkS1,noBWalkS1,noBWalkS1,noBWalkS1,noBWa
lkS1}},  // bWalk1
	{0b100100, 0x00,  10,
{bWalkS2,bWalkS2,bWalkS2,bWalkS2,bWalkS2,bWalkS2,bWalkS2,bWalkS2}},		//
noWalkS1
	{0b100100, RedLED,  10,
{noBWalkS2,noBWalkS2,noBWalkS2,noBWalkS2,noBWalkS2,noBWalkS2,noBWalkS2,noBWa
lkS2}},  // bWalkS2
	{0b100100, 0x00,  10,
{bWalkS3,bWalkS3,bWalkS3,bWalkS3,bWalkS3,bWalkS3,bWalkS3,bWalkS3}},		//
noWalkS2
	{0b100100, RedLED,  10, {goS,goW,goS,goW,goS,goW,goS,goW}},
// bWalkS3

	// walking W
```

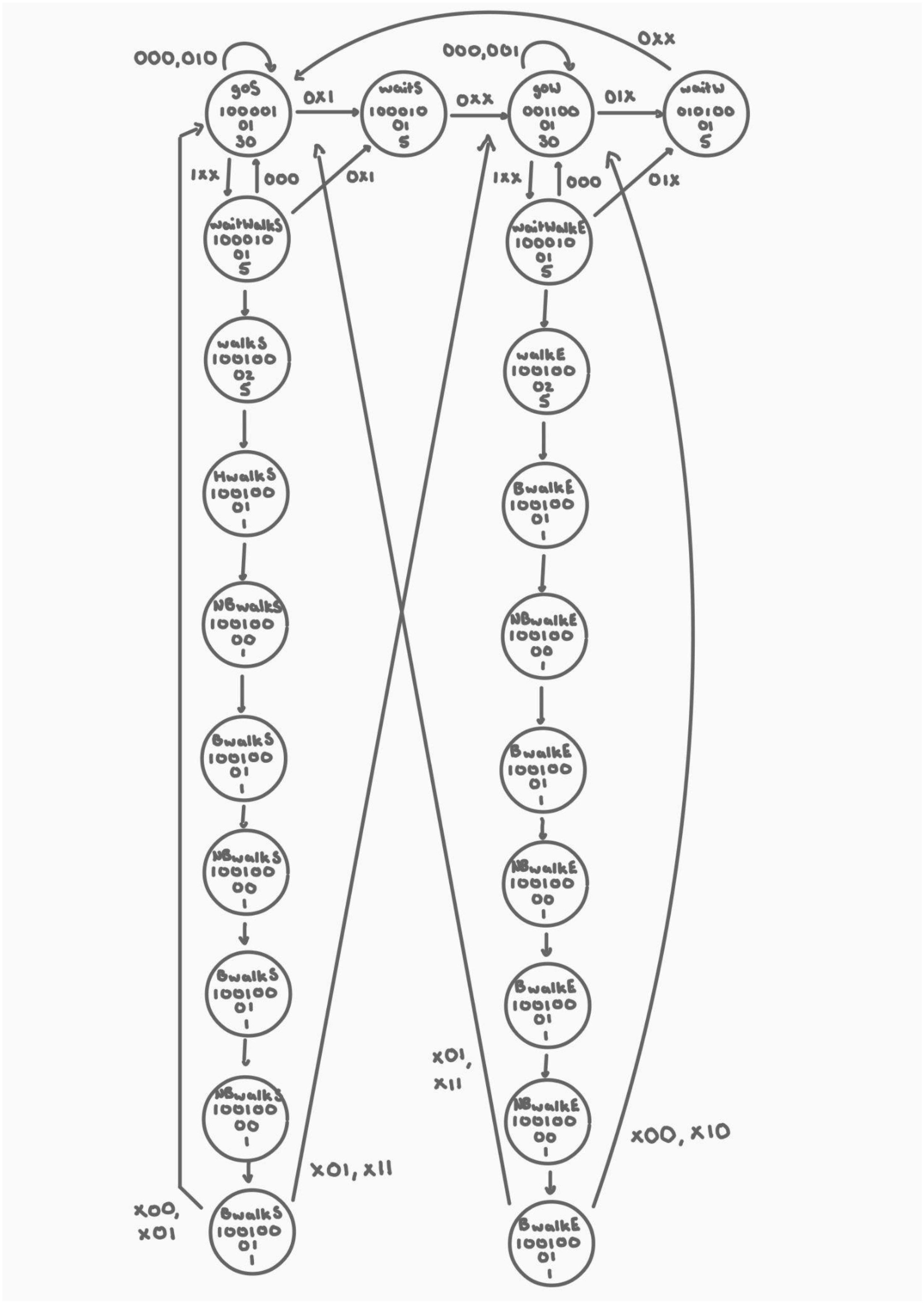
```
        {0b100100, RedLED,  50, {goW,walkW,waitW,waitW,walkW,walkW,walkW,walkW}},
//waitWalkW
        {0b100100, GreenLED,  50,
{bWalkW,bWalkW,bWalkW,bWalkW,bWalkW,bWalkW,bWalkW,bWalkW}},                    //
walkW
        {0b100100, RedLED,  10,
{noBWalkW,noBWalkW,noBWalkW,noBWalkW,noBWalkW,noBWalkW,noBWalkW,noBWalk
W}},        // bWalkW
        {0b100100, 0x00,  10,
{bWalkW1,bWalkW1,bWalkW1,bWalkW1,bWalkW1,bWalkW1,bWalkW1,bWalkW1}},
// noBWalkW
        {0b100100, RedLED,  10,
{noBWalkW1,noBWalkW1,noBWalkW1,noBWalkW1,noBWalkW1,noBWalkW1,noBWalkW1,
noBWalkW1}},  // bWalkW1
        {0b100100, 0x00,  10,
{bWalkW2,bWalkW2,bWalkW2,bWalkW2,bWalkW2,bWalkW2,bWalkW2,bWalkW2}},
// noBWalkW1
        {0b100100, RedLED,  10,
{noBWalkW2,noBWalkW2,noBWalkW2,noBWalkW2,noBWalkW2,noBWalkW2,noBWalkW2,
noBWalkW2}},  // bWalkW2
        {0b100100, 0x00,  10,
{bWalkW3,bWalkW3,bWalkW3,bWalkW3,bWalkW3,bWalkW3,bWalkW3,bWalkW3}},
// noBWalkW2
        {0b100100, RedLED,  10, {goS,goW,goS,goW,goS,goW,goS,goW}},
// bWalkW3
}; // {output, how long, order of states}

void GPIOPortB_Init() {
  SYSCTL_RCGC2_R |= 0b10;        // 1) turns on clock B
  GPIO_PORTB_AMSEL_R &= ~(0xFF); // 3) disable analog function on PB5-0
  GPIO_PORTB_PCTL_R &= ~(0xFF);  // 4) enable regular GPIO
  GPIO_PORTB_DIR_R |= 0xFF;      // 5) outputs on PB5-0
  GPIO_PORTB_AFSEL_R &= ~(0xFF); // 6) regular function on PB5-0
  GPIO_PORTB_DEN_R |= 0xFF;      // 7) enable digital on PB5-0
}

void GPIOPortE_Init() {
  SYSCTL_RCGC2_R |= 0b10000;     // 1) turns on clock for E
  GPIO_PORTE_AMSEL_R &= ~(0xFF); // 3) disable analog function on PE1-0
  GPIO_PORTE_PCTL_R &= ~(0xFF);  // 4) enable regular GPIO
```

```
    GPIO_PORTE_DIR_R &= ~(0xFF);   // 5) inputs on PE1-0
    GPIO_PORTE_AFSEL_R &= ~(0xFF); // 6) regular function on PE1-0
    GPIO_PORTE_DEN_R |= 0xFF;      // 7) enable digital on PE1-0
}

void GPIOPortF_Init(void) {
    SYSCTL_RCGCGPIO_R |= 0x20;          // 1) Activate clock for Port F; set bit 5 to turn on
clock for Port F
    SysTick_Wait10ms(1);                // Allow time for clock to stabilize
    GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY;  // (2) Unlock GPIO Port F Commit Register
    GPIO_PORTF_CR_R = 0xFF;             // Enable commit for Port F
    GPIO_PORTF_AMSEL_R = 0x00;          // (3) Disable analog functionality
    GPIO_PORTF_PCTL_R = 0x00000000;     // (4) Configure Port F as GPIO
    GPIO_PORTF_DIR_R = 0x0E;            // (5) PF0 and PF4 input, PF3-1 output
    GPIO_PORTF_AFSEL_R = 0x00;          // 6) Regular port function; Disable alternate function
    GPIO_PORTF_PUR_R = 0x11;            // Enable weak pull-up on PF0 and PF4
    GPIO_PORTF_DEN_R = 0xFF;            // (7) Enable digital I/O on Port F
}

//  if using pb2 -> |= the data reg w 0b0100, then wait, then use &= to clear it
void srclk() {
    GPIO_PORTB_DATA_R |= 0b0010;
    SysTick_Wait10ms(1);
    GPIO_PORTB_DATA_R &= ~(0b0010); // PB1
}

void rclk() {
    GPIO_PORTB_DATA_R |= 0b1000;
    SysTick_Wait10ms(1);
    GPIO_PORTB_DATA_R &= ~(0b1000); // PB3
}

// make a clk to pulse pins, put on ser
void LedOn (uint32_t Light) {
    int i;
    uint32_t Light2;

    for (i = 0; i <= 7; i++) {      // shift register
        if (i == 6 || i == 7) {     // loads 0 into QA and QB
            Light = 0b0;
            SysTick_Wait10ms(1);
            srclk();
        } else {                    // loads new bit into QA and pushes prev bit to next Q#
            GPIO_PORTB_DATA_R &= ~(0b1);
            Light2 = Light >> i;
            Light2 &= 0x01;
            GPIO_PORTB_DATA_R |= Light2;
            SysTick_Wait10ms(1);
            srclk();
        }
    };
    rclk();
}

// walk light code should go here []
// write to port f data reg either pin 1 2 or 3
void WalkingLight (uint32_t Light) {
    GPIO_PORTF_DATA_R &= ~(0b1110);
    GPIO_PORTF_DATA_R |= Light;
}

void main(void){

    STyp *Pt;                           // state pointer
    uint32_t Input;

    PLL_Init(Bus80MHz);                         // 80 MHz, Prog 4.6
    SysTick_Init();                         // Program 4.7
    GPIOPortE_Init();
    GPIOPortB_Init();
    GPIOPortF_Init();

    Pt = goS;                           // set to first initial state

    while(1){
        LedOn(Pt->Out);                         // set lights
        WalkingLight(Pt->Walk_Light);
        SysTick_Wait10ms(Pt->Time);
        uint32_t SENSOR = GPIO_PORTE_DATA_R &= 0b111; // loads buttons into data register
and sets it to SENSOR
```

```
        Input = SENSOR;                         // read sensors (buttons); put data reg here
        Pt = Pt->Next[Input];
    }

}
```

Question 2:

Alexis Nhan
EGEC 450



The type of FSM I created was the Moore FSM. If a Mealy FSM was used instead, then the controller would need to reference the previous output in order for the state to be changed. For this to work in this lab, you would need to wait for an whether there is a car or person at each light. Once there is a car or person, then that input and the current state you are in will trigger the LEDs to function as expected. It would then switch states based on what input comes in and what the current state is. In my personal FSM, I have 22 states and it was defined as:

```
#define goS         &FSM[0]
#define waitS       &FSM[1]
#define goW         &FSM[2]
#define waitW       &FSM[3]
#define waitWalkS   &FSM[4]
#define walkS       &FSM[5]
#define bWalkS      &FSM[6]
#define noBWalkS    &FSM[7]
#define bWalkS1     &FSM[8]
#define noBWalkS1   &FSM[9]
#define bWalkS2     &FSM[10]
#define noBWalkS2   &FSM[11]
#define bWalkS3     &FSM[12]
#define waitWalkW   &FSM[13]
#define walkW       &FSM[14]
#define bWalkW      &FSM[15]
#define noBWalkW    &FSM[16]
#define bWalkW1     &FSM[17]
#define noBWalkW1   &FSM[18]
#define bWalkW2     &FSM[19]
#define noBWalkW2   &FSM[20]
#define bWalkW3     &FSM[21]
```

Now based on my specific FSM, it is possible to use fewer states while maintaining the correct functionality of the controller. I feel this could be done by changing how the walk cycle states are implemented.

Question 3:

Every state has 8 next-state arrows. This is related to the system since the system takes in 3 bit inputs, each from the three switches, making each state have 8 possible next state arrows. If there were four next-state transitions between states there would only be 2 bit inputs. For this lab, that would be the case if there was no walking light. Now the 8 next-state arrows were defined in my code as:

```
const struct State *Next[8];
```

Question 4:

If my intersection was put on trial, how I would prove that my FSM works properly is that I would collect the time data it would take for each street light and walking light would cycle. From there I would show the consistency of the light change. Now how I would collect this data is measuring the time it takes for each street and walking light to cycle, how long two would cycle when triggered at once, and how long all three would cycle if triggered at the same time.

Question 5:

My system switches to a green light in about 3 seconds. If a car arrives at the intersection every second, then each street would have on average 3 cars lined up in the intersection at the red light. If for every 0.5 seconds a car shows up, there would be around 6 cars per red light. If there was a pedestrian crossing the street they would be waiting 4 seconds in order to cross. To make the system more efficient by having the timing of the lights be predicted. This prediction can be done roughly through the use of the queueing theory. What the queueing theory does is create a probability in which how long each light should last by taking the average time usage for each street or walking light trigger.

Question 6:

As my shift register is shifting, my LEDs do not change. Instead my LEDs only change once all the shifting is done. What is responsible for making my LEDs change when the shift register is done is my ‘rclk’ function. The ‘rclk’ function is placed after the shift register for loop code seen here:

```
for (i = 0; i <= 7; i++) {          // shift register
        if (i == 6 || i == 7) {     // loads 0 into QA and QB
                Light = 0b0;
                SysTick_Wait10ms(1);
                srclk();
        } else {                    // loads new bit into QA and pushes prev bit to next Q#
                GPIO_PORTB_DATA_R &= ~(0b1);
                Light2 = Light >> i;
                Light2 &= 0x01;
                GPIO_PORTB_DATA_R |= Light2;
                SysTick_Wait10ms(1);
                srclk();
        }
}
rclk();
```

With the ‘rclk’ function triggering after the for loop it allows the loop to shift all the values from the data register into each Q section of the shift register. Once it moves all the values in, the ‘rclk’ will trigger causing all the data in the shift register to be pushed into the pins outputting the LEDs properly.

If I wanted to make it so that each shift would cause an output to the LEDs, I would move my ‘rclk’ function to be within the loop of the shift register at the end of the else code.

Question 7:

Students I had discussed this assignment with were Brenden Dack and Ryley Seifert.

[1] “SNx4HC595 8-Bit Shift Registers With 3-State Output Registers,” Austin, Texas: Texas Instruments , 2021

[2] *TivaTM TM4C123GH6PM Microcontroller DATA SHEET*. Austin, Texas: Texas Instruments Incorporated, 2007.

[3] J. W. Valvano, *Embedded Systems. Volume 1, Introduction to the ARM® Cortex(TM)-M3 Microcontrollers*. Austin, Texas: Jonathan W. Valvano, 2019.