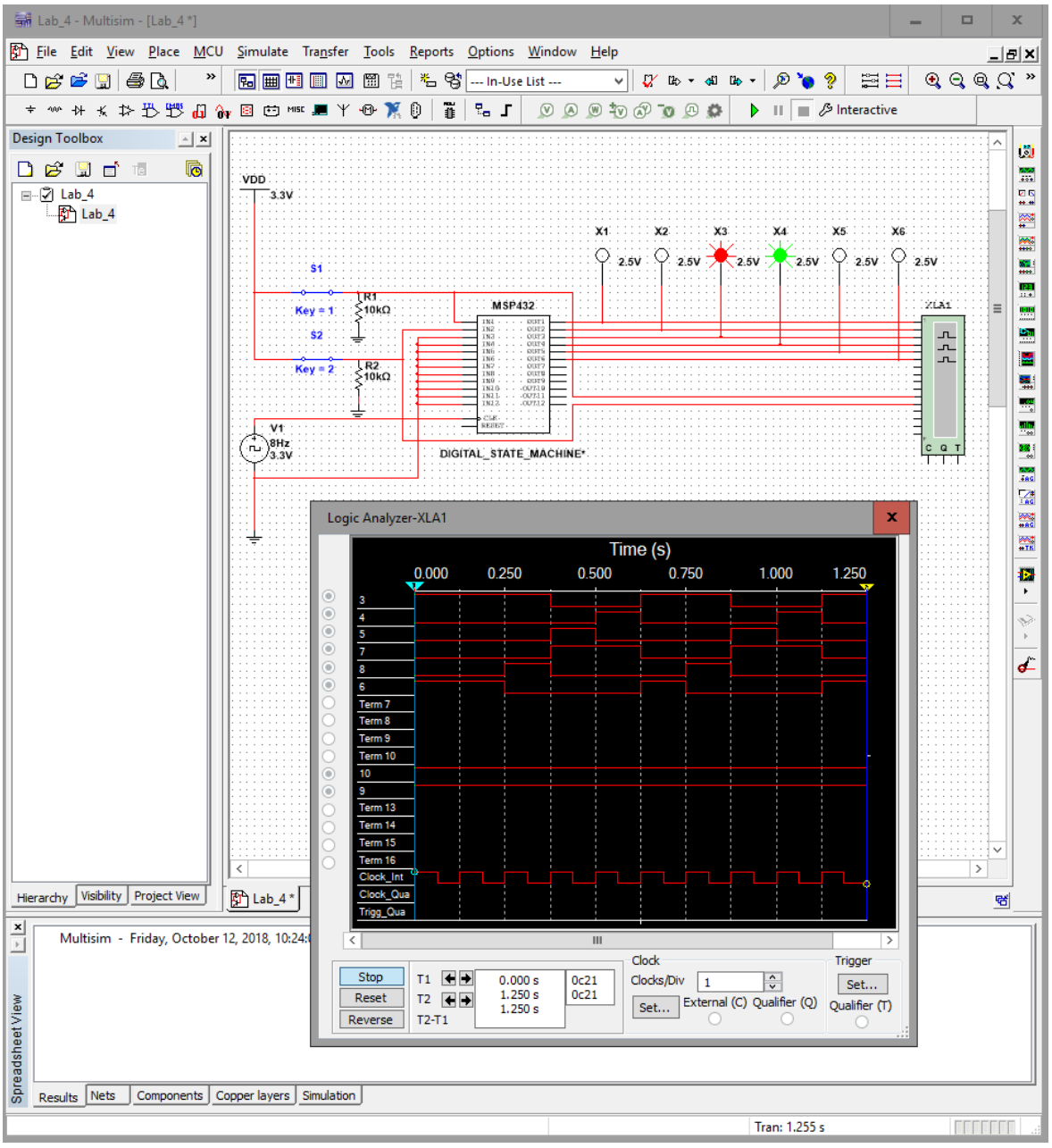
Introduction:

During this lab, we will be working with Code Composer Studio and will be looking at interfacing with the MSP432 board to simulate a traffic intersection. We will be controlling red, yellow, and green LEDs with switches connected with a breadboard. First, we will be testing the design using Multisim before implementing it on the breadboard. Next, we will be using a FSM running a loop to check for a button press and if pressed toggle the red LED using a delayed loop otherwise leave the LED on.

Procedure:

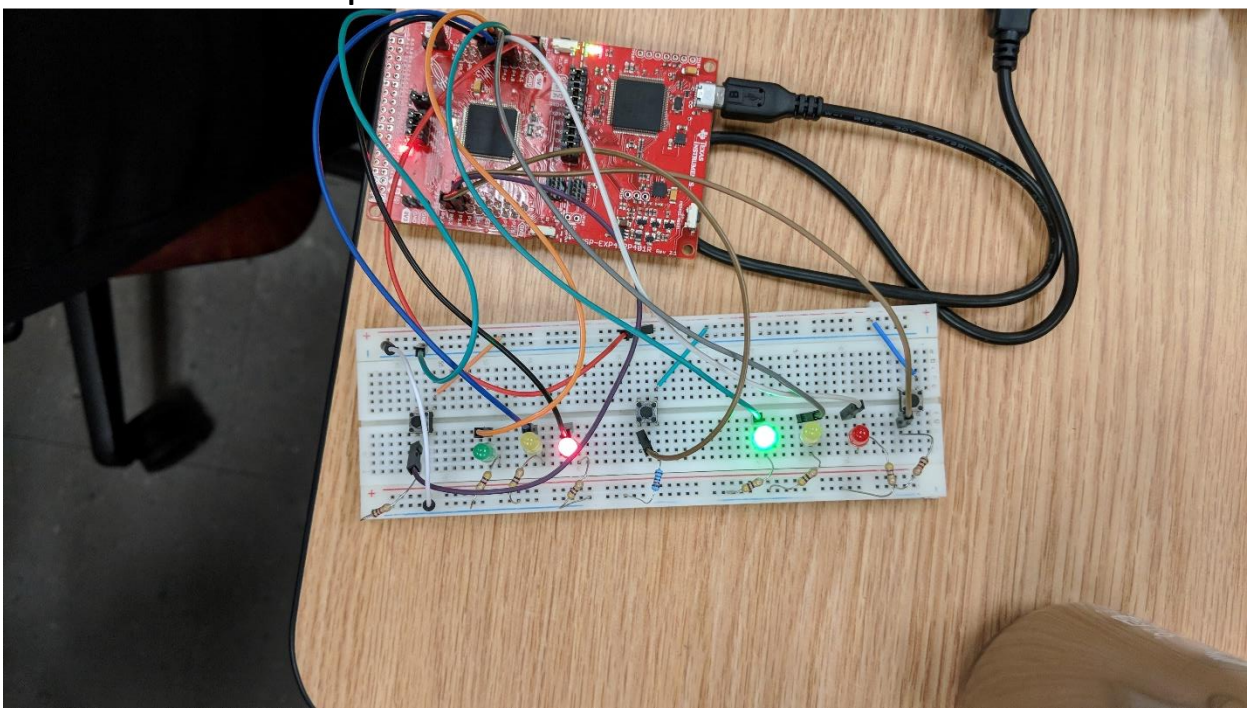
1. **Model and Simulate the External Hardware**

Hardware Circuit & Simulation



2. **Part 2 – Add Crosswalk Button**

Circuit Setup



Debug Code for MSP432

See Appendix for code

Conclusion:

The result of this lab was determining how to interface with external circuits and to implement a Finite State Machine. The FSM is a good structure to use in order to accommodate multiple inputs. Each state has a possible 8 input combination since each light has a button and the crosswalk button. The use of the state graph in the textbook was helpful to see where each input combination needed to transition to. This is how I implemented the round robin technique to handle the transition of states depending upon which buttons were pressed.

For my FSM, I only needed 14 states. I used a state after the light was green or yellow to check if the walk was pressed and then handle accordingly. Once the walk button is pressed, it took about 8 states to toggle the light from green, to blink red, and then solid red. Depending on the complexity of signal detection would be the greatest factor on how many total states in different FSMs.

References:

For this project I used the lecture slides and the textbook for references to the template code.

Appendix:

Code – Part 1:

```
#include <stdint.h>
#include "SysTick.h"
#include "msp432p401r.h"

#define SENSOR  (*((volatile uint8_t *)0x40004c40))
#define LIGHT   (*((volatile uint8_t *)0x40004c23))

struct State {
 uint32_t Out;
 uint32_t Time; // 10 ms units
 const struct State *Next[4];
};

typedef const struct State STyp;
#define goN   &FSM[0]
#define waitN &FSM[1]
#define goE   &FSM[2]
#define waitE &FSM[3]

STyp FSM[4] = {
{0x21,30,{goN,waitN,  goN,waitN}},
{0x22, 30,{goE,  goE,  goE,  goE}},
{0x0C,30,{goE,  goE,waitE,waitE}},
{0x14, 30,{goN,  goN,  goN,  goN}}
};

              int main(void) { // at 3 MHz
 STyp *Pt; // state pointer
 uint32_t Input;
 // initialize ports and timer
    SysTick_Init();    // Program 4.7
    // activate Port 4 and Port 5
    // make P4.5-P4.0 GPIO outputs
    // assumes P4SEL0 and P4SEL1 are zero
    P4DIR |= 0x3F;     // out

    // make P5.1-P5.0 GPIO inputs
    // assumes P5SEL0 and P5SEL1 are zero
    P5DIR &= ~0x03;    // in
    Pt = goN; // start state
```

```
    while(1) {
  // set lights to current state's Out
    LIGHT = Pt->Out; // set lights
  // specified wait for this state
    SysTick_Wait10ms(Pt->Time);
  // input from car detectors
    Input = (SENSOR&0x03); // read sensors
  // next depends on state and input
    Pt = Pt->Next[Input];
  }
}
```

Code – Part 2:

```
#include <stdint.h>
#include "SysTick.h"
#include "msp432p401r.h"

#define SENSOR  (*((volatile uint8_t *)0x40004C40))
#define LIGHT   (*((volatile uint8_t *)0x40004C23))
#define WALKLIGHT (*((volatile uint8_t *)0x40004C03))

struct State {
 uint32_t Out;
 uint32_t Time; // 10 ms units
 uint32_t Out2;
 const struct State *Next[8];
};

typedef const struct State STyp;
#define goN       &FSM[0]
#define waitN     &FSM[1]
#define goE       &FSM[2]
#define waitE     &FSM[3]
#define waitAll   &FSM[4]

#define checkN    &FSM[5]
#define checkE    &FSM[6]

#define walk      &FSM[7]
#define hurry1    &FSM[8]
#define hurry2    &FSM[9]
#define hurry3    &FSM[10]
#define hurry4    &FSM[11]
#define hurry5    &FSM[12]
```

```
#define hurry6     &FSM[13]
#define dontWalk   &FSM[14]

// fix for 8 states
STyp FSM[15] = {
{0x0C,300,0x01,{goN, goN, waitN, waitN, checkN, checkN, checkN, checkN}}, //FSM[0] goN
{0x14,100,0x01,{waitAll, waitAll, goE, goE, waitAll, waitAll, waitAll, waitAll}}, //FSM[1] waitN
{0x21,300,0x01,{goE, waitE, goE, waitE, checkE, checkE, checkE, checkE}}, //FSM[2] goE
{0x22,100,0x01,{waitAll, goN, waitAll, goN, waitAll, waitAll, waitAll, waitAll}}, //FSM[3] waitE
{0x24,100,0x01,{goN, goN, goE, goN, walk, walk, walk, walk}}, //FSM[4] waitAll

{0x0C,200,0x01,{goN, goN, waitN, waitE, walk, walk, walk, walk}}, //FSM[5]
{0x21,200,0x01,{goE, waitE, goE, waitN, walk, walk, walk, walk}}, //FSM[6]

{0x24,300,0x02,{  hurry1,   hurry1,   hurry1, hurry1, hurry1, hurry1, hurry1, hurry1}}, //FSM[7]
{0x24, 50,0x01,{  hurry2,   hurry2,   hurry2, hurry2, hurry2, hurry2, hurry2, hurry2}}, //FSM[8]
{0x24, 50,0x00,{  hurry3,   hurry3,   hurry3, hurry3, hurry3, hurry3, hurry3, hurry3}}, //FSM[9]
{0x24, 50,0x01,{  hurry4,   hurry4,   hurry4, hurry4, hurry4, hurry4, hurry4, hurry4}}, //FSM[10]
{0x24, 50,0x00,{  hurry5,   hurry5,   hurry5, hurry5, hurry5, hurry5, hurry5, hurry5}}, //FSM[11]
{0x24, 50,0x01,{  hurry6,   hurry6,   hurry6, hurry6, hurry6, hurry6, hurry6, hurry6}}, //FSM[12]
{0x24, 50,0x00,{dontWalk, dontWalk, dontWalk, dontWalk, dontWalk, dontWalk, dontWalk,
dontWalk}}, //FSM[13]
{0x24,100,0x01,{    goN,     goN,     goE, goE, walk, goN, goE, goN}}, //FSM[14]
};

int main(void) { // at 3 MHz
 STyp *Pt; // state pointer
 uint32_t Input;
 // initialize ports and timer
    SysTick_Init();    // Program 4.7
    // activate Port 4 and Port 5
    // make P4.5-P4.0 GPIO outputs
    // assumes P4SEL0 and P4SEL1 are zero
    P4DIR |= 0x3F;     // out
    P2DIR |= 0x03;
    // make P5.1-P5.0 GPIO inputs
    // assumes P5SEL0 and P5SEL1 are zero
    P5DIR &= ~0x07;    // in              // change to 0111
    Pt = goN; // start state

    while(1) {
   // set lights to current state's Out
    LIGHT = (LIGHT&~0X3F) | Pt->Out; // set lights
    WALKLIGHT = (WALKLIGHT&~0x03) | Pt->Out2; // set lights
```

```
  // specified wait for this state
    SysTick_Wait10ms(Pt->Time);
  // input from car detectors
    Input = (SENSOR&0x07); // read sensors
  // next depends on state and input
    Pt = Pt->Next[Input];
  }

  //set if statement for walk red led flash
}
```