

Lab Report

|  |  |
| --- | --- |
| Name: Alexis Steven Garcia | Date: 10/10/2018 |
| Course: EGCP-450 | Lab #: 4 |

Grading Criteria:

|  |  |  |
| --- | --- | --- |
| **Section** | **Earned Points** | **Possible Points** |
| Problem/Objective: |  | 10 |
| Background: |  | 15 |
| Questions/Deliverables: |  | 15 |
| Program Code: |  | 30 |
| Demo: |  | 30 |
| Total: | 0 | 100 |

**PLEASE UPLOAD YOUR REPORT IN TITANIUM. NO PAPER REPORTS.**

Professor Comments:

# Problem/Objective

State the problem statement and/or objective of the lab. This must be a complete paragraph (i.e., at least 5 sentences).

The objective of this lab was to understand how to implement linked data structures and apply our knowledge by creating a data structure in C. Also, one of this lab’s intention was to introduce us to the subject of creating segmented software systems. In addition, we familiarized ourselves with designing an FSM controller with real-time synchronization. As for the method of how we would connect the circuit, we continued using Multisim as a platform to aid us. With this program, we were able to draw and analyze our circuit before implementing our design on the protoboard.

# Background

Briefly describe what you did in the lab including technical detail. It must be at least two **complete** paragraphs to receive full credit.

This lab demonstrated how to create a finite-state machine (FSM) to mimic a portion of a 4-corner intersection. In other words, by creating a linked data structure, we were able to build a state that resemble scenarios that would occur at a stoplight. For example, a state in the FSM can be for only cars on the north side to cross. This would mean that only cars on the north side would see a green light. Then we built on this with more states. For instance, if a car on the east wants to cross, the north would see a yellow. Then when the north side see a red light, the cars on the east would get to cross. With this simple FSM, the first part of the lab concluded.

The second part of the lab required us to implement a “walk” light to represent a pedestrian wanting to cross. Some requirements to keep in mind when creating the system were traffic should not be allowed to crash and cars should not be allowed to hit the pedestrians. For this reason, this portion of the lab required a longer linked data structure to represent various situations. For example, some states to consider would be the following: should only the walk light be on or should the cars go while a pedestrian from the same side crosses the street the same direction the cars are heading. Also, when the pedestrian light is about to change, the light should blink the red “walk” light on and off.

The hardware utilized in this lab required us to use GPIO ports; therefore, we interfaced with external hardware and internal hardware. The only GPIO port used on the board was port 2 to act as the “walk” light. More specifically, P2.0 and P2.1 to represent the “walk” red and green light. The rest of the hardware was external. Before creating the circuit, one of the first things we were instructed to do was create our design on Multisim. This aided us to replicate our design on Multisim onto the protoboard. With the design in mind, it was ideal to connect all our external output to one port and all our inputs to another. In other words, our LEDs that acted as output were connected to port 4 and our pushbuttons were connected to port 5.

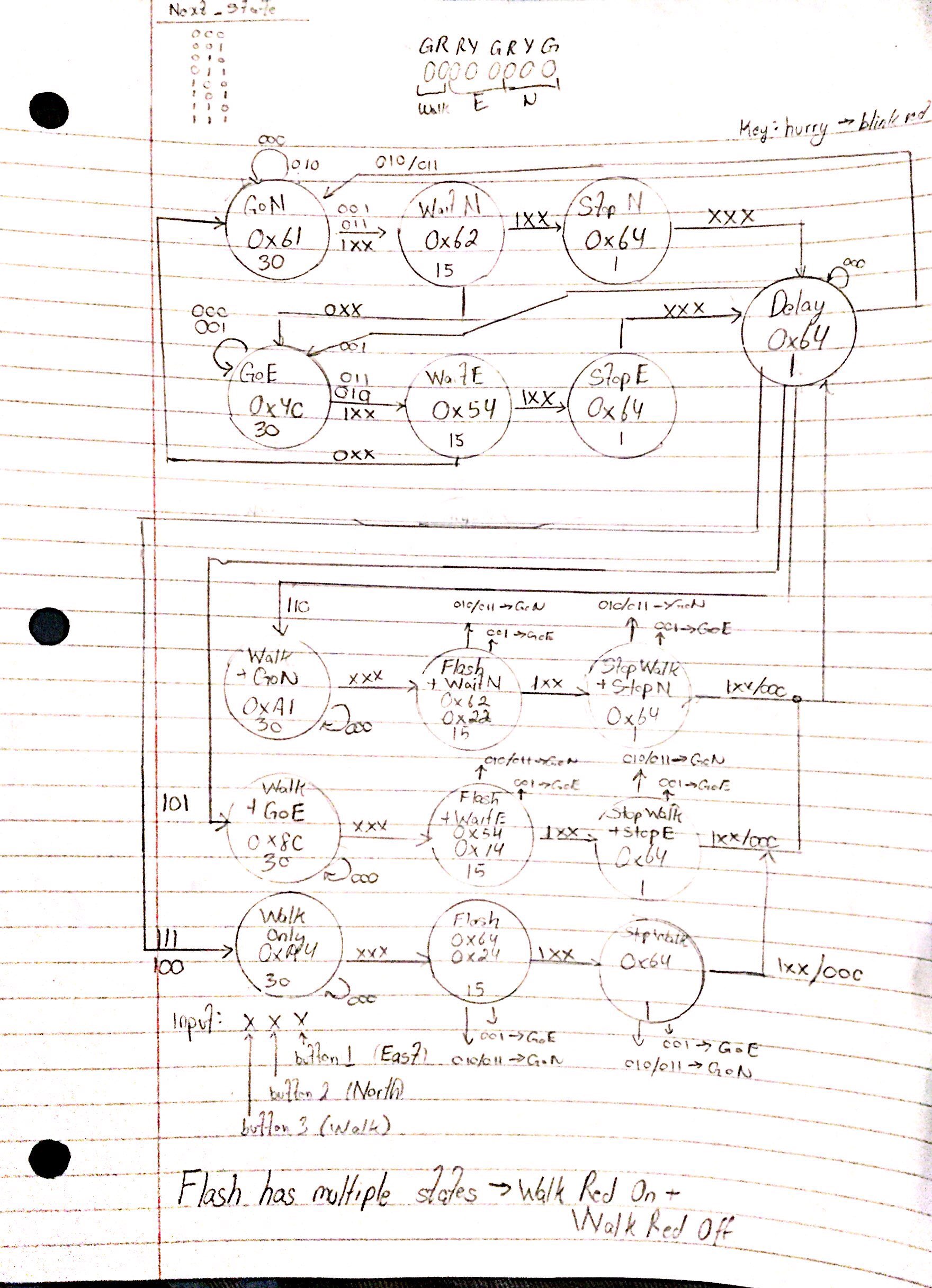
The software proved to be more challenging at first; however, after spending some time on it, the logic wasn’t to hard to follow. The first step was to create a linked data structure that would be the barebones for each state. Each representation of the data structure would require a name, an output, a time to wait, and 8 next state pointers. With the amount of variation in our system, our FSM required at least 15 states to function as intended. After this, it was a matter of translating what we would do in assembly to C. For example, we would need to define variables with their contents being addresses to locations of desired control registers, manipulating control registers to act as GPIO, and creating a loop run the main program.

# Questions/Deliverables:

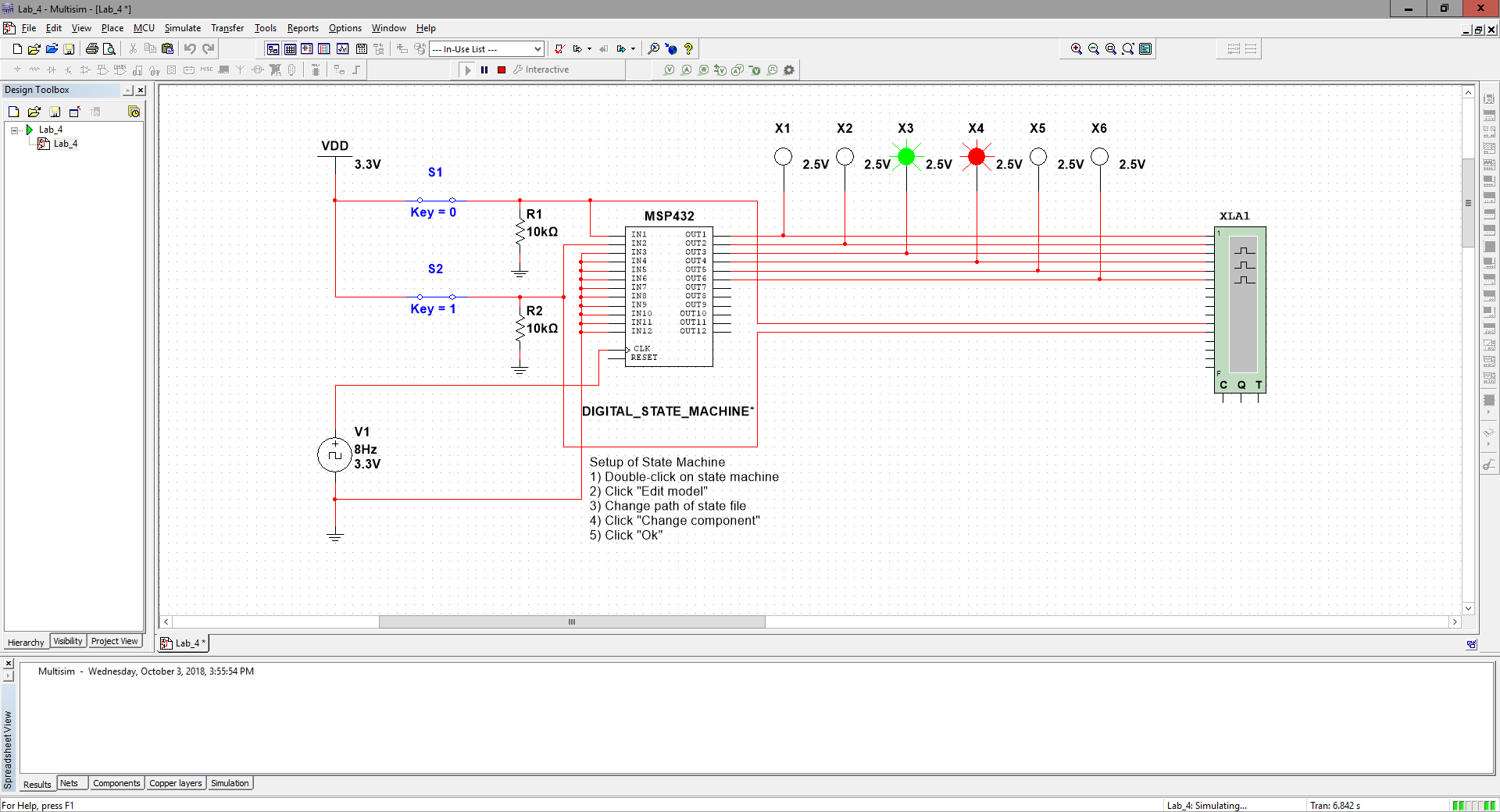
1. Using 50 words or less, explain the differences between a Mealy and Moore FSM. For which types of problems should you implement with Mealy? For which types of problems should you implement with Moore?

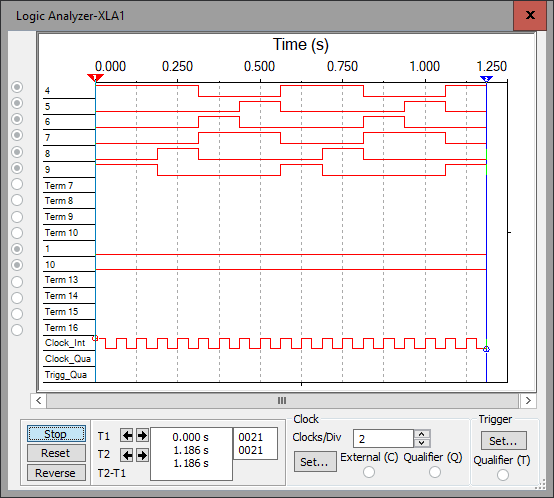
The difference between Mealy and Moore FSM is that Moore depends only on current state while Mealy relies on an additional input. Mealy is more suitable for problems that require less states or don’t need to wait for a clock edge. Moore finite-state machines are appropriate when a system is built with the idea of negating asynchronous feedback or using a clock to decide when an output changes.

1. Give the updated FSM state diagram for your implementation of Part 2 of the lab. If you want, you can hand write your work, take a picture, and paste the image here. One app that I would suggest to easily do this is “CamScanner”.



1. Also, include a screenshot of your Multisim schematic and waveform here.





# Program Code

Copy your code here. Please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your code as text, not as a screenshot/image**.

**#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

**const** **struct** State \*Next[19];

};

**typedef** **const** **struct** State STyp;

**#define** goN &FSM[0]

**#define** waitN &FSM[1]

**#define** stopN &FSM[2]

**#define** goE &FSM[3]

**#define** waitE &FSM[4]

**#define** stopE &FSM[5]

**#define** delay &FSM[6]

**#define** walkOnly &FSM[7]

**#define** flashOn &FSM[8]

**#define** flashOff &FSM[9]

**#define** stopWalking &FSM[10]

**#define** walkAndGoN &FSM[11]

**#define** flashOnAndWaitN &FSM[12]

**#define** flashOffAndWaitN &FSM[13]

**#define** stopAndStopN &FSM[14]

**#define** walkAndGoE &FSM[15]

**#define** flashOnAndWaitE &FSM[16]

**#define** flashOffAndWaitE &FSM[17]

**#define** stopAndStopE &FSM[18]

/\*

\* The data structure has exactly these components:

\* a name, an output, a time to wait, and 8 next state pointers

\* (one for each input).

\* 000 001 010 011 100 101 110 111

\*/

//If needed x11 is to go North first

//Original: 3000, 500, 3000, 500

// 1 = 10ms

// 10 = 100ms

// 100 = 1000ms = 1s

STyp FSM[19] = {

{0x61, 300,{goN,waitN,goN,waitN,waitN,waitN,waitN,waitN}}, //goN

{0x62, 50,{goE,goE,goE,goE,stopN,stopN,stopN,stopN}}, //waitN

{0x64, 10,{delay,delay,delay,delay,delay,delay,delay,delay}}, //stopN

{0x4C, 300,{goE,goE,waitE,waitE,waitE,waitE,waitE,waitE}}, //goE

{0x54, 50,{goN,goN,goN,goN,stopE,stopE,stopE,stopE}}, //waitE

{0x64, 10,{delay,delay,delay,delay,delay,delay,delay,delay}}, //stopE

{0x64, 100,{goN,goE,goN,goN,walkOnly,walkAndGoE,walkAndGoN,walkAndGoN}}, //delay

{0xA4, 300,{walkOnly,flashOn,flashOn,flashOn,flashOn,flashOn,flashOn,flashOn}}, //walkOnly

{0x64, 50,{flashOff,flashOff,flashOff,flashOff,flashOff,flashOff,flashOff,flashOff}}, //flashOn

{0x24, 50,{stopWalking,goE,goN,goN,stopWalking,stopWalking,stopWalking,stopWalking}}, //flashOff

{0x64, 10,{delay,goE,goN,goN,delay,delay,delay,delay}}, //stopWalking

{0xA1, 300,{walkAndGoN,flashOnAndWaitN,flashOnAndWaitN,flashOnAndWaitN,flashOnAndWaitN,flashOnAndWaitN,flashOnAndWaitN,flashOnAndWaitN}}, //walkAndGoN

{0x62, 50,{flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN,flashOffAndWaitN}}, //flashOnAndWaitN

{0x22, 50,{stopAndStopN,goE,goN,goN,stopAndStopN,stopAndStopN,stopAndStopN,stopAndStopN}}, //flashOffAndWaitN

{0x64, 10,{delay,goE,goN,goN,delay,delay,delay,delay}}, //stopAndStopN

{0x8C, 300,{walkAndGoE,flashOnAndWaitE,flashOnAndWaitE,flashOnAndWaitE,flashOnAndWaitE,flashOnAndWaitE,flashOnAndWaitE,flashOnAndWaitE}}, //walkAndGoE

{0x54, 50,{flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE,flashOffAndWaitE}}, //flashOnAndWaitE

{0x14, 50,{stopAndStopE,goE,goN,goN,stopAndStopE,stopAndStopE,stopAndStopE,stopAndStopE}}, //flashOffAndWaitE

{0x64, 10,{delay,goE,goN,goN,delay,delay,delay,delay}} //stopAndStopE

};

**int** **main**(**void**) {

STyp \*Pt; // state pointer

uint32\_t Input;

uint32\_t Temp;

// initialize ports and timer

SysTick\_Init(); //activate port4

P2->SEL0 &= ~0x03;//make P2.1-P2.0 GPIO Outputs

P2->SEL1 &= ~0x03;

P2->DIR |= 0x03;

P4->SEL0 &= ~0x3F; //make P4.5-P4.0 GPIO Outputs

P4->SEL1 &= ~0x3F;

P4->DIR |= 0x3F;

P5->SEL0 &= ~0x16; //make P5.2-P5.0 GPIO Inputs

P5->SEL1 &= ~0x16;

P5->DIR &= ~0x16;

Pt = goN; // start state

**while**(1) {

LIGHT = (LIGHT&~0x3F)|(Pt->Out); // set lights

WALKLIGHT = (WALKLIGHT&~0x03)|((Pt->Out)>>6);

SysTick\_Wait10ms(Pt->Time);

Temp = (SENSOR&0x10)>>4; //Get P5.4 and shift it to be in position 5.0

Input = (SENSOR&0x06)|Temp; // read sensors

Pt = Pt->Next[Input];

}

}