Module 7

Lab 7: Finite State Machine

Lab 7: Finite State Machine

7.0 Objectives

The purpose of this lab is to develop and test a Finite State Machine (FSM) that could be used in a robot to follow a line.

1. You will learn how to use structures and pointers in C.
2. You will understand how to use FSMs to solve problems.
3. You will implement a simple line-following algorithm with an FSM.

Good to Know: Even though you will implement this lab using switches for inputs and LEDs for output, the FSM design process can be used for robot controllers. The solution to this lab will allow a robot to follow a line (black mask tape).

7.1 Getting Started

7.1.1 Software Starter Projects

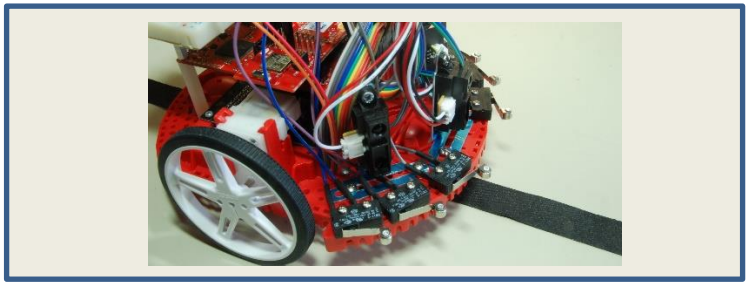
Look at these three projects:
PointerTrafficFSM (example use of a finite state machine)
LineFollowFSM (simple FSM that implements line following) and
Lab07_FSM (starter project for this lab)

7.1.2 Student Resources (in datasheets directory-Links)

Meet the MSP432 LaunchPad (SLAU596*)*
MSP432 LaunchPad User's Guide (SLAU597*)*

7.1.3 Reading Materials

Volume 1 Sections 6.1, 6.2, 6.4 and 6.5
Embedded Systems: Introduction to the MSP432 Microcontroller",
or
Volume 2 Section 3.5
Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller",



7.1.4 Components needed for this lab

Quantity	Description	Manufacturer	Mfg P/N
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R

7.1.5 Lab equipment needed

Oscilloscope (one channel at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling

7.2 System Design Requirements

The **Lab07_FSM** starter project implements the three-state FSM shown in Figure 1, which we could use to implement a line-following robot. The 500 is the time to wait in each state in ms. On the real robot, we set these delay times to be much shorter, depending on how fast the mechanical robot responds to actuator commands. However, in this lab, the 500 ms is chosen to make it easy to see the output with our eyes.

Lab 7: Finite State Machine

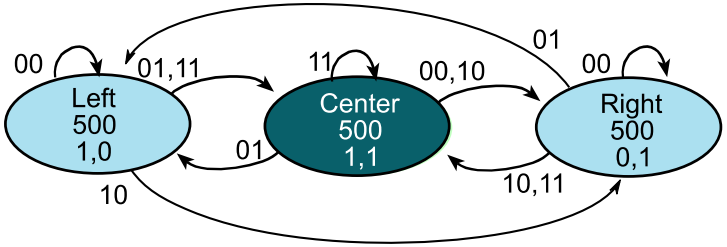


Figure 1. Moore FSM state graph to implement line following. The time in each state is shown in 1ms units.

The robot has two sensors that detect the line, see Figure 2. If the robot is properly positioned on the line, both sensors will read 1. If the robot is a little off to the left or right, one sensor reads 1, and the other sensor reads 0. If the robot is completely off the line, both sensors will read 0.

The robot has two motors, also shown in Figure 2. The two motors and a passive caster allow the robot to operate in a differential drive fashion. If the software outputs high to both motors, the robot moves forward in a straight line. If the software outputs high to just one motor, it will turn. If the software outputs low to both motors, it will stop.

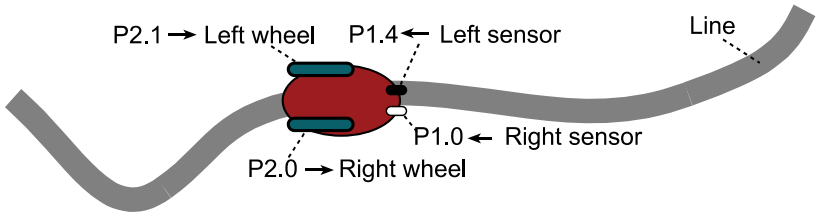


Figure 2. Robot with two line sensors and two wheel motors.

You are asked to extend this FSM, adding additional states, to implement the following behaviors.

1) The FSM in Figure 1 gets confused (has a bug) if the robot is off little bit to the left (input is 01, and the machine is oscillating between the Left and Center states) and then goes completely off the line to the left (input is 00). In this machine, if it happens to be in the Center state when it goes off the line, it will incorrectly move to the Right state even though the robot went off to the left. You will solve this problem by implementing two left states (so it oscillates between the two left states when a little left). For symmetry, you will implement two right states as well. Figure 3 shows a partial solution. If the input is 11, then the output should remain 11. If the input goes to 01 (it is a little left), then the output should toggle 1,0↔1,1 causing a slight right turn. Similarly, if the input goes to 10 (it is a little right), then the output should toggle 0,1↔1,1 causing a slight left turn.

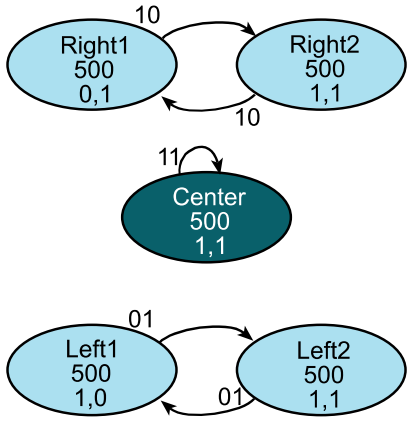


Figure 3. Expanded FSM state graph. The time in each state is shown in 1ms units.

2) The second behavior you need to implement is what happens when the robot goes completely off the line. If it goes off the line to the right (input=0,0 while in Right1 or Right2), it should make a hard left turn (output=0,1) for 5 seconds, then go straight (output=1,1) for 5 seconds. If it is still off the line at this point it should stop (output=0,0). If it finds the line, resume line following. It should take three more states to implement this behavior.

Similarly, if the robot goes off the line to the left (input=0,0 while in Left1 or Left2), it should make a hard right turn (output=1,0) for 5 seconds, then go straight (output=1,1) for 5 seconds. If it is still off the line at this point it should stop (output=0,0). If it finds the line, it should resume line following. It should take three more states to implement this behavior.

The solution should have about 11 states (5 states from Figure 3, plus 3 for lost to the right, plus 3 states for lost to the left). As long as you have 9 or more states, feel free to make assumptions or change the exact behavior of the machine. The objective of the lab is to describe the complete behavior of a system with the state transition graph, and then to implement that behavior with a very simple FSM controller. The FSM controller should have NO conditional branch statements.

Lab 7: Finite State Machine

7.3 Experiment set-up

You will implement this lab using just the MSP432 Launch Pad, without need for additional circuits, see Figure 4. The Launch Pad driver software converts the switch input to positive logic so “switch pressed” is seen as a 1, see Table 1. The LED outputs are in positive logic, see Table 2.

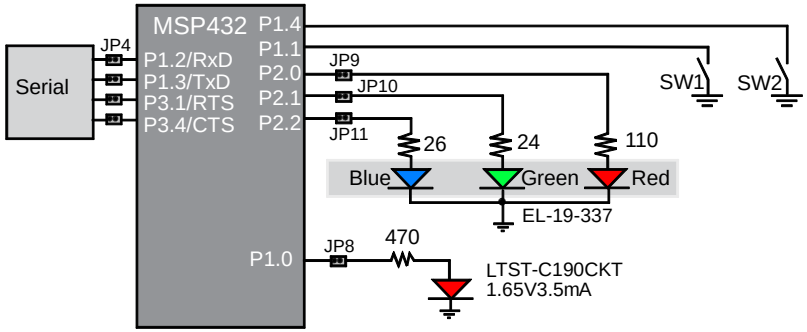


Figure 4. P1.4 is the left sensor, P1.1 is the right sensor, P2.1 is the left motor P2.0 is the right motor.

The **LaunchPad_Input** function (defined in LaunchPad.c) returns the switch position in positive logic, so pushing both switches creates an input condition of 1,1. The **LaunchPad_Output** function (defined in LaunchPad.c) sends data to the 3-bit color LED.

SW2	SW1	**LaunchPad_Input**	Meaning
pressed	pressed	1,1 = 0x03	On line
pressed	not	1,0 = 0x02	Right of line
not	pressed	0,1 = 0x01	Left of line
not	not	0,0 = 0x00	Off the line

Table 1. Switches simulate line sensors.

P2.1	P2.0	**LaunchPad_Output**	LED	Meaning
off	off	0,0 = 0x00	black	Stop
off	on	0,1 = 0x01	red	Turn left
on	off	1,0 = 0x02	green	Turn right
on	on	1,1 = 0x03	yellow	Straight

Table 2. LEDs simulate robot motor

7.4 System Development Plan

7.4.1 Line Follow FSM

The first step is to compile, download and run the **LineFollowFSM** example shown below. Using the debugger, single step through the controller (step over the functions) and observe **Input**, **Output**, and the pointer **Spt**. Notice how the structure is defined and how the pointer is used to access data in the structure. Using the debugger, determine where in memory is the FSM located (is it in RAM or ROM)?

Lab 7: Finite State Machine

```
struct State {
  uint32_t out;                // 2-bit output
  uint32_t delay;              // time to delay in 1ms
  const struct State *next[4]; // Next if input is 0-3
};
typedef const struct State State_t;

#define Center &fsm[0]
#define Left   &fsm[1]
#define Right  &fsm[2]
StateType fsm[3]={
  {0x03, 500, { Right, Left,   Right,  Center }},
  {0x02, 500, { Left,  Center, Right,  Center }},
  {0x01, 500, { Right, Left,   Center, Center }}
};
State_t *Spt;  // pointer to the current state

uint32_t Input;
uint32_t Output;

int main(void){ uint32_t heart=0;
  Clock_Init48MHz();
  LaunchPad_Init();
  TExaS_Init(LOGICANALYZER);  // optional
  Spt = Center;
  while(1){
    Output = Spt->out;            // set output from FSM
    LaunchPad_Output(Output);     // output to motors
    TExaS_Set(Input<<2|Output);   // optional
    Clock_Delay1ms(Spt->delay);   // wait
    Input = LaunchPad_Input();    // read sensors
    Spt = Spt->next[Input];       // next
    heart = heart^1;
    LaunchPad_LED(heart);         // optional
  }
}
```

In this program, this FSM performs the 4-step sequence over and over:

1) *Output* depends on *State* (LaunchPad LED)
2) *Wait* depends on State
3) *Input* (LaunchPad buttons)
4) *Next* depends on (*Input, State*)

Run the program and observe the static behavior.

i) Fill in Table 3 describing what this machine does if the input remains constant.

SW2	SW1	Input	Meaning	Output behavior
pressed	pressed	1,1	On line	
pressed	not	1,0	Right of line	
not	pressed	0,1	Left of line	

Table 3. Static response table of the simple FSM.

When just one switch is pressed, it represents the condition where the robot is a little off the line. In this situation, one wheel is active and other wheel oscillates on and off. This oscillation causes this wheel to spin, but at a slower rate. If P2.1 is high, the left wheel spins at 100%. The **duty cycle** of a digital wave is defined as the percentage of the time the signal is high. If the duty cycle on P2.0 is *n*=(high/ (high+low)), then the right motor spins at *n**100%, and the robot will gently turn. Use an oscilloscope or logic analyzer to measure the oscillation rate and duty cycle on Port P2.0. See Figure 5.

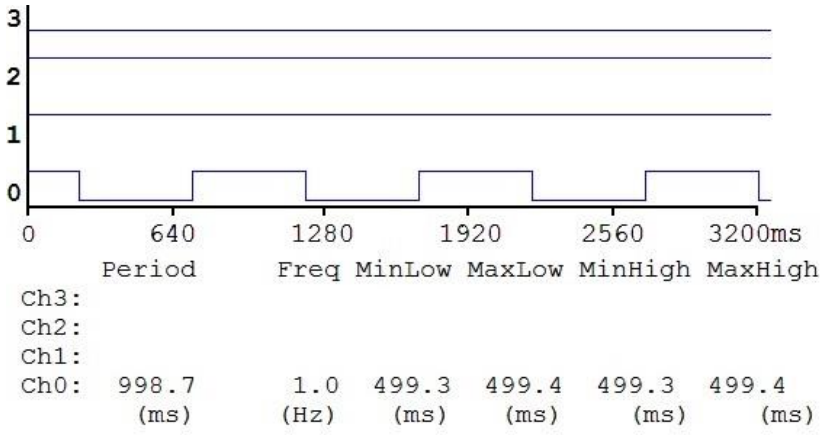


Figure 5. Logic analyzer trace showing the oscillation on the right wheel Channel 0 is 1 Hz and has a 50% duty cycle.

Note: Channel 3-2 are Input =1 (left sensor=0, right sensor =1), showing the condition a little bit off to left. Channels 1-0 are the Output (left motor=1, right motor oscillating), showing a gentle right turn.

Lastly, you will observe the bug in this FSM.
1) Start with both switches pressed (on the line);
2) Release SW2 (the robot is a little off to the left); then
3) Release SW1.

At this point you are completely off the line to the left. Repeat this 1-2-3 step sequence multiple times, and you will find sometimes it correctly ends up in the left state, but sometimes it incorrectly ends up in the right state.

7.4.2 Design an improved FSM

The second step is to design an FSM as described in the requirements section Figure 3.0. As long as your machine has 9 or more states, feel free to adjust exactly how the machine operates. In this lab section you will:

i) Draw the state transition graph

ii) Create a state transition table, and enter the C code for the data structure.
All three should be exactly the same information (no more no less). This equivalency is called **one-to-one** and it is an important feature of good FSM design. If the graph is one-to-one with the data structure in C, then we can be confident the system operates as described by the graph.

iii) You will test your system using the same 1-2-3 step sequence shown at the end of section 7.4.1. However, as long as you wait at least 500 ms with SW2 released before you release SW1, then you should always end up in one of the left states.

Perform this test at least ten times to verify it works correctly. Similarly for the right side states,
1) Start with both switches pressed (on the line);
2) Release SW1 (the robot is a little off to the right); then
3) Release SW2

At this point you are completely off the line to the right. Repeat this 1-2-3 step ten multiple times and you should always end up in one of the right states.

Use the logic analyzer to test the static behavior of the system. Assuming the input remains constant fill in Table 4.There are two off the line conditions: off to left and off to right.

SW2	SW1	Input	Meaning	Output behavior
pressed	pressed	1,1	On line	
pressed	not	1,0	Right of line	
not	pressed	0,1	Left of line	
not	not	0,0	Off the line	
not	not	0,0	Off the line	

Table 4. Static response table of the Lab FSM.

7.5 Troubleshooting

Can't program LaunchPad:

- Check the cables, jumpers on the LaunchPad development board.
- Check the Windows driver to see if the board is recognized by the operating system.
- Try another LaunchPad on this computer.
- Try this LaunchPad on another computer

Hard fault:

- Verify **Spt** always points an entry in the FSM.

Time delays are too slow or too fast:

- Verify the computer is running at 48 MHz.
- Go back and make sure Lab in Module 6GPIO still works

Lab 7: Finite State Machine

7.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab.

- Can there be two states with the same output? Why?
- How does the FSM create the 50% duty cycle output wave? What would you change to make it 75% (even more gentle turn)? What would you change to make it 25% (sharper turn)?
- It is important that the state transition graph and the data structure in C are one-to-one. What does one-to-one mean and explain how is it true?
- This lab uses I/O abstraction in the four functions that begin with **LaunchPad**_. What information is in the header file LaunchPad.h? What is in the code file LaunchPad.c? What benefits does this abstraction provide?
- This FSM had 2 inputs. What would change if there were 3 inputs? 4 inputs?
- This FSM had 2 outputs. What would change if there were 3 outputs? 4 outputs?
- How is the FSM tested?

7.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. You could extend the system or propose something completely different. For example,

- Replace the switch input with the actual line sensor interfaced in Lab 6. If you use the line sensor, you can expand the input from 2 bits to 4 bits.
- Use the FSM method to solve similar problems like the traffic light controller or a stepper motor controller
- This FSM used a pointer to define the current state. You could implement the FSM using an index to access the parameters of the state. E.g., `Output = fsm[index].out;`

7.8 Which modules are next?

The FSM is a powerful design tool for solving complex systems. Effective solutions to many of the possible robot challenges will include FSMs.

Module 8) Interface actual switches and LEDs to the microcontroller. This will allow for more inputs and outputs increasing the complexity of the system.
Module 9) Develop a simple PWM output to adjust duty cycles
Module 10) Develop debugging techniques to prove behavior for complex systems
Module 12) Connect the line sensor and motors to the robot, and run the solution to this lab on the actual robot.

7.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Use ***struct*** to organize data
- Access data using a pointer
- Use multiple files in a project to implement abstraction
- Design a simple FSM drawing a state transition graph
- Convert a state transition graph into C data structure
- Use a logic analyzer to measure timing between inputs/outputs
- Debug the FSM and verify its proper behavior

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2018, Texas Instruments Incorporated