

Lab 4h Stepper Motor Finite State Machine

This laboratory assignment accompanies the book, *Embedded Microcomputer Systems: Real Time Interfacing*, Second edition, by Jonathan W. Valvano, published by Thomson, copyright © 2006.

- Goals**
- To interface a stepper motor,
 - To implement background processing with periodic interrupts,
 - To develop a dynamic linked command structure.

- Review**
- Valvano Section 1.4 about open collector logic,
 - Valvano Chapter 2 about abstraction, linked lists and FSM's,
 - Valvano Chapter 8, about stepper motor interfacing,
 - Valvano Section 13.2.3, about an open loop stepper motor control system.

- Starter files**
- `Moore_9S12.zip` `OC_9S12` projects and
 - the ExpressSCH files `EE345LextraComponents.sch` `Lab4h_9S12C32.sch`

Background

Stepper motors are popular with digital control systems because they can be used in an open loop manner with predictable responses. Such applications include positioning heads in disk drives, adjusting fuel mixtures in automobiles, and controlling articulating joints in robotics. In this lab, you will control a stepper motor using a finite state machine. The finite state machine must be implemented as a linked data structure. Four switches will allow the operator to control the motor. A background periodic interrupt (either **OC** or **RTI**) thread will perform inputs from the switches and outputs to the stepper motor coils. The worm gear on the motor, as shown in Figure 4.1, produces a linear motion as the stepper motor turns. It takes 24 full-steps to complete one rotation of the shaft and 200 full-steps to move the worm gear from one end to the other. If you step the motor too far in either direction, the worm gear will disengage and you will have to manually re-engage the worm gear. Four switches determine the motor operation.

If all buttons are released, then the motor should stop.

If switch 1 is pressed, spin slowly in one direction as long as switch 1 continues to be pressed.

If switch 2 is pressed, spin slowly in the other direction as long as switch 2 continues to be pressed.

If both switch 1 and 2 are pressed, the motor should continuously vibrate as fast as possible back and forth

CW for 8 steps and CCW for 8 steps as long as both switch 1 and 2 continue to be pressed.

If switch 3 is pressed then released, then step once in one direction.

If switch 4 is pressed then released, then step once in the other direction.

Any other combination (e.g., 1+3, 2+3, 1+4, 3+4, 1+2+3, etc.) should be ignored.

You may assume the operator (you or the TA) will release the switches when it reaches the end of the worm gear.



Figure 4.1. Stepper motor with worm gear, CAT# EX-82, <http://www.allelectronics.com>.

Preparation (do this before your lab period)

With an ohmmeter, measure the resistance of one coil. Apply +5V across the coil and simultaneously measure using both a voltmeter and a current-meter the voltage and current required to activate one coil of your stepper motor. Do this before your lab period, without any connections to the 9S12 board. Design the hardware

interface between the 9S12, and prepare a circuit diagram labeling all resistors and diodes. The pin out for the stepper is shown in Figure 4.2. Include pin numbers and resistor/capacitor types and tolerances. Be sure the interface circuit (e.g., 2N2222 or L293) you select can sink enough current to activate the coil. In particular, verify the driver can supply (I_{CE} or I_{OL}) the required current for the stepper motor. Make sure you have all the parts you need before lab starts. This interface will require four 1N914 snubber diodes. If do not properly connect the diodes, the back EMF will destroy your board.

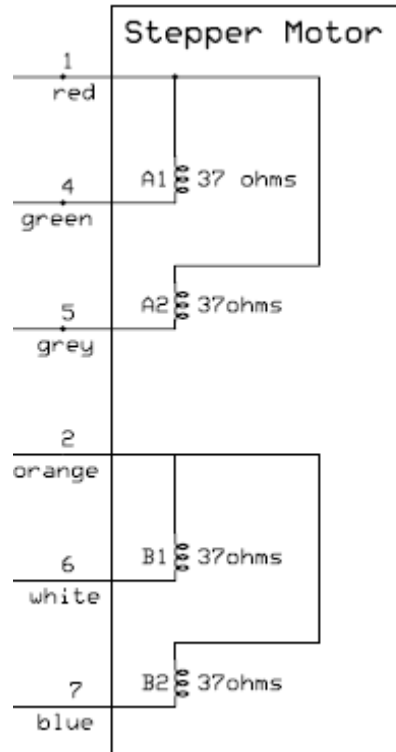


Figure 4.2. The EX-82 stepper motor has unipolar configuration with four +5V 37-Ω coils (see the starter file **Lab4h_9S12C32.sch**).

Design the machine by first drawing the finite state graph. A “syntax-error-free” hardcopy listing for the software is required as preparation. This will be checked off by the TA at the beginning of the lab period. You are required to do your editing before lab. The debugging will be done during lab. Document clearly the operation of the routines. The periodic interrupt-driven background thread will execute the finite state machine, while the foreground thread initializes the system then does nothing. Figure 4.3 shows the data flow graph of the stepper motor controller.

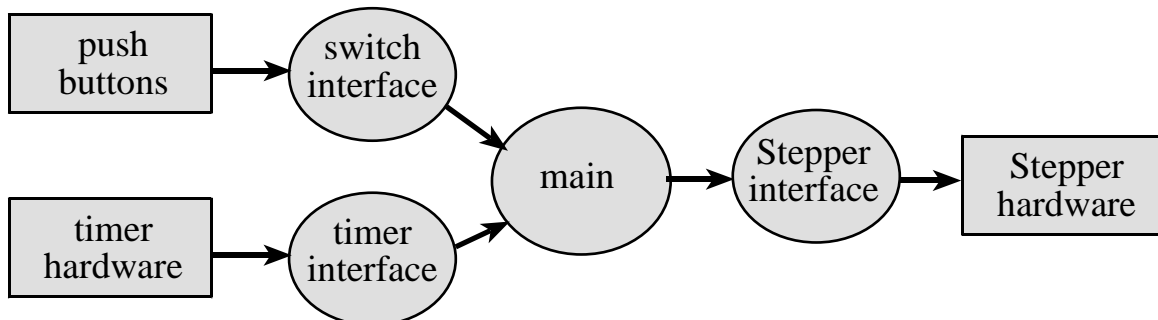


Figure 4.3. Data flows from the timer and the switches to the stepper motor.

Figure 4.4 shows a possible call graph of the system. Dividing the system into modules allows for concurrent development and eases the reuse of code.

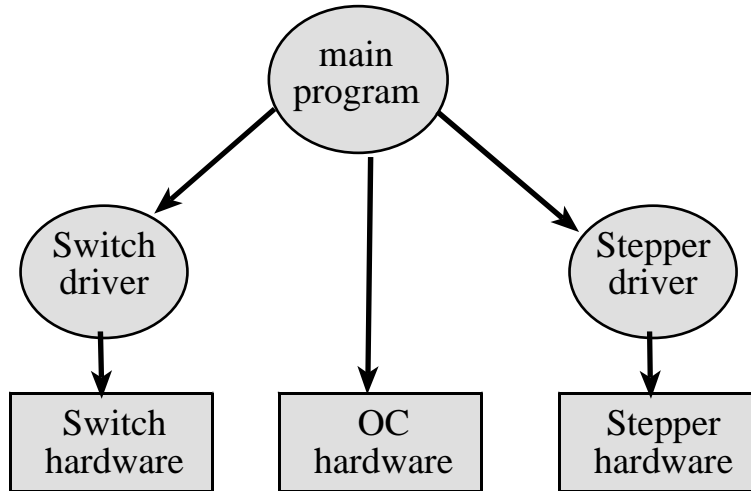


Figure 4.4. A call graph showing the three modules used by the stepper controller.

A switch device driver means you create **switch.h** and **switch.c** files, separating mechanisms (how it works) from policies (what it does). Similarly, make **stepper.h** and **stepper.c** files. Design the software in a manner that makes it easier to understand, debug, modify and reuse in other projects. Design the system in a manner that would allow you to reassign the pin connections of the interface (e.g., moving the stepper from **PTM** to **PTT**), by making changes to **switch.c** and **stepper.c** without requiring modifications to **main.c** **switch.h** or **stepper.h**.

Procedure (do this during your lab period)

Make sure your TA checks your hardware diagram before connecting it to the 9S12. We do not have extra boards, so if you fry your board, you may not be able to finish. First build the digital interface on a separate protoboard from the 9S12, as shown in Figure 4.5. Use switches as inputs for the stepper interface. You should be able to generate the 5,6,10,9 sequence with your fingers. Using a scope, look at the voltages across the coils to verify the diodes are properly eliminating the back EMF. The very fast turn-off times of the digital transistors can easily produce 100 to 200 volts of back EMF, so please test the hardware before connecting it to the 9S12.

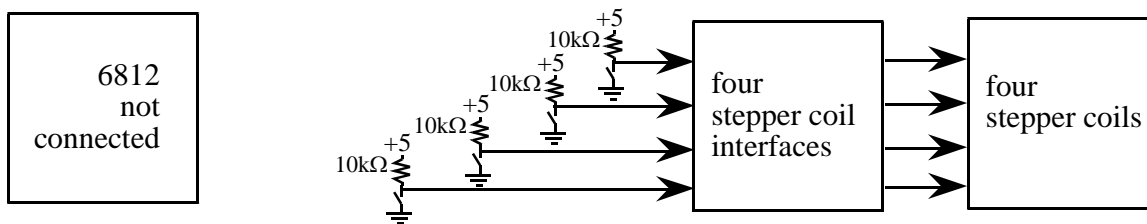


Figure 4.5. Hardware test procedure.

The switches will eventually be replaced by 9S12 outputs, as shown in Figure 4.6. Once you are sure the hardware is operating properly, run a simple constant velocity software function to verify the hardware/software interface.

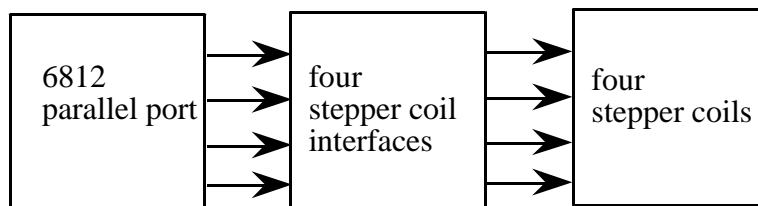


Figure 4.6. Hardware block diagram.

To determine the fastest rate the motor can vibrate back and forth (CW for 8 steps and CCW for 8 steps), experimentally test the system at smaller and smaller delays between steps. You will be able to vibrate more quickly using a non-uniform time delay between steps. Read the discussion of jerk in Chapter 8 of the textbook.

Deliverables (exact components of the lab report)

- A) Objectives (1/2 page maximum)
- B) Hardware Design (ExpressSCH printout)
 - stepper motor interface, showing all external components
 - switch interfaces, showing all external components
- C) Software Design (a hardcopy software printout is due at the time of demonstration)
 - Draw figures illustrating the major data structures used, e.g., the finite state graph
 - If you organized the system different than Figure 4.3 and 4.4, then draw its data flow and call graphs
- D) Measurement Data
 - Prep) Give the voltage, current, and resistance measurements
 - Specify the fastest rate the motor can vibrate back and forth
- E) Analysis and Discussion (1 page maximum)

Checkout (show this to the TA)

You should be able to demonstrate correct operation of the stepper motor system. Be prepared to describe how your stepper interface works. Explain how your system handles the switch bounce. Demonstrate debugging features that allow you to visualize the software behavior.

A hardcopy printout of your software will be given to your TA, and graded for style at a later time.

Hints

1. Be sure the interface driver (e.g., the 2N2222 or L293) has an I_{CE} or I_{OL} large enough to deliver the needed coil current. You may use the 9S12C32 +5V regulated supply to run this stepper motor because it requires less than 500 mA total current. Although many steppers will operate at voltages less than the rated voltage, the torque is much better when using the proper voltage. Remember to actually measure the coil current rather than dividing voltage by resistance.
2. Be sure to put an appropriate delay between each step to prevent the motor from burning up.
3. To prevent the power supply from overheating, do not leave the system plugged in too long. Even while stopped the stepper draws current. So, you will notice the 9S12 regulator will get warm.
4. The **TEaS** simulator does include a stepper motor, so you can develop and test the software in parallel with the hardware development. In particular, the **OC** project can be run on the **TEaS** simulator.
5. You are done if
 - there is a 1-1 mapping from the FSM graph and the C data structure
 - it is a FSM with 4 inputs and 8 outputs
 - there is a delay for each state, which is saved in the FSM structure
 - the motor output is always one of these values 5, 6, 10, or 9
 - the motor moves such that the output never skips from 5 to 10, 10 to 5, 6 to 9, or 9 to 6
 - there are about 20 to 30 states, and it works approximately the way it is described in the lab assignment
 - the FSM runs in the background using periodic interrupts
 - the foreground (main) initializes the FSM, then executes `for (; ;) { } do nothing loop`
 - the ISR has no backward jumps. i.e., there are no wait function calls in the ISR
6. See the data sheets and materials on stepper motors at

<http://users.ece.utexas.edu/~valvano/DataSheets/>

2N2222.pdf	L293.pdf
1N914.pdf	B3F-1059.pdf , B3F-switch.pdf
Stepper.pdf	
StepperBasic.pdf	StepperDriveBasic.pdf
StepperHalfstep.pdf	StepperMicrostep.pdf
StepperSelection.pdf	Stepper_ST.pdf