Lab 4 Stepper Motor Interface

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

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

Review
- Chapter 13 of the M68HC812A4 Technical Summary, output compare interrupts
- Valvano Section 1.6 about open collector logic,
- Valvano Section 1.7 about initializing and accessing I/O ports,
- Valvano Section 2.4 about abstraction, linked lists and FSM's,
- Valvano Section 3.4.2 about accurate time delays,
- Valvano Chapter 8, about stepper motor interfacing,
- Valvano Section 13.2.3, about an open loop stepper motor control system.

Starter files
- MOORE12.*, INTERP12.C, LLFIFO.*, HEAP.C, LLTEST.*
- OC3TEST.C, OC3.C, OC3.H

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 robot arms in robotics. In this lab you will develop a command interpreter as the foreground thread (main program). You will use the serial port routines developed in a previous lab as I/O for the interpreter. The operator will enter commands to manipulate the stepper motor. For example

```
s          clear all commands stop motor
r 5 100    rotate 5 times clockwise at a rate of 100 rpm
r 1 -50    rotate once counterclockwise at a rate of 50 rpm
```

You are free to adjust the commands, the style and the units of the interpreter, as long as you maintain the educational objectives (stepper motor, interrupt processing, and dynamic linked list). The motor should continue to operate while additional commands are being entered. Each linked list node will contain counters (rotation count, step count), constants (speed and direction), and links (pointer to next command to execute.) You may implement either a single or double linked list. You may use a simple fixed block size memory manager found in `HEAP.H` and `HEAP.C`. Nodes are placed at the end of the dynamic linked list by the interpreter, and deleted off the front of the list when the background thread completes execution. You may use a simple dynamically-allocated linked list FIFO queue found in `LLFIFO.H` and `LLFIFO.C`. A background periodic interrupt thread will perform outputs to the stepper motor coils. For an example of output compare interrupts, download the files OC3TEST.C, OC3.C, OC3.H. The motor should stop when the list is empty.

Preparation (do this before your lab period)

Use an external +12 V power supply, and a voltage/current meter as measure the current required to activate one coil of your stepper motor. Do this before lab without the 6812 board. Design the hardware interface between the 6812 and prepare a circuit diagram labeling all resistors and diodes. Include pin numbers and resistor/capacitor types and tolerances. Be sure the interface circuit you select can sink enough current to activate the coil. Make sure you have all the parts you need before lab starts.

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 interrupt-driven background thread will output the specified patterns, while the foreground thread implements the command interpreter.

Procedure (do this during your lab period)

Make sure your TA checks your hardware diagram before connecting it to the 6812. We do not have extra boards, so if you fry your board, you may not be able to finish. Build the digital interface on a separate protoboard from the 6812. I suggest you get a squarewave generator and connect it one by one to the 4 digital controls (the place you will eventually connect the 6812). Use a scope, look at the voltages across the coils to verify the diodes are properly removing 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 6812.

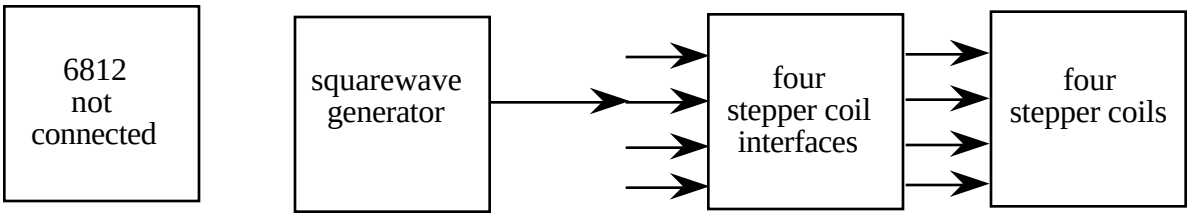

Figure 4.1. Hardware test procedure.

Once you are sure the hardware is operating properly, run a simple constant velocity software function to verify the hardware/software interface.

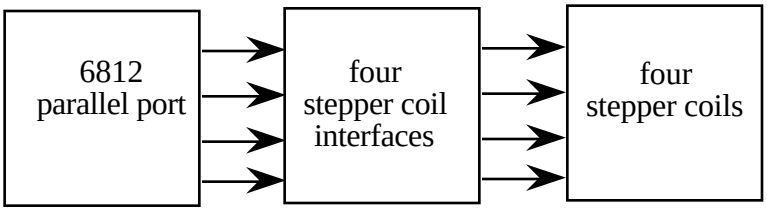

Figure 4.2. Hardware block diagram.

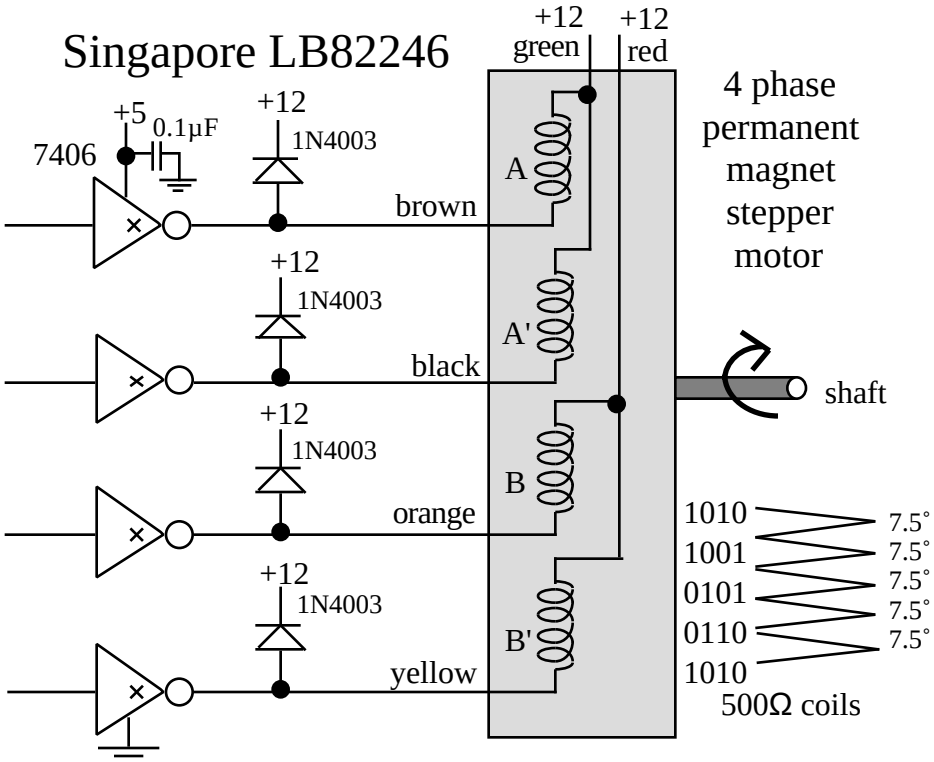

Figure 4.3. Singapore LB82246 Stepper Motor Specifications

Checkout (show this to the TA)

You should be able to demonstrate correct operation of the motor:

- spins the correct number of times;
- stops at the same angle it started;
- spins at the correct speed;
- spins the correct direction.

You should be able to demonstrate correct operation of the interpreter:

- verify the proper handling of illegal formats;
- motor operates during command input;
- demonstrate your software does not crash.

Hints

1. Stepper motors other than the *Singapore LB82246* will have different coil resistances and different coil voltages. Be sure the interface driver (e.g., the 7406 in Figure 4.3) has an I_{OL} large enough to deliver the needed coil current. Please do not use the Adapt812 +5V regulated supply to run external components that require more than 100 mA of current. Although the *Singapore LB82246* will operate on the Adapt812's +5V, the torque is much better running with an external +12V supply. For larger motors always use a separate power supply. 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, make sure the supply can deliver the required current.
4. Debug the lab in small steps.
5. Even though the TExaS simulator does not include a stepper motor (yet), you can connect the 4 digital outputs to a logic analyzer and test the software functions. In particular, the LLTEST.UC, LLTEST.IO files can be run on the TExaS simulator. The files MOORE12.UC, MOORE12.IO, and MOORE12.SCP files can also be simulated.