EGR 222 Mechatronics
Lab #11 Microcontroller Stepper Motor Control
March 31,April 1 & 9, 2015

11.0 Objectives:

Understanding of stepper motors and stepper sequencing principles. Also, practice and use of stored programs and the microcontroller for control, and use of the Analog to Digital converter.

11.1 Pre-Lab assignment

Read and be familiar with the text material concerning stepper motors.

Review the material in the book concerning microcontrollers and C programming, and the programming that was used in Lab 7 to properly sequence a blinking LED. The Lab 7 overlapped blinking LED exercise is a starting point for this lab exercise.

Write a program to cause a stepper motor to move in the forward direction at various speeds, as indicated by the settings of a digital value on a DIP switch connected to an input port. (See the directions below for details.) Bring the program to lab. Be sure to also bring handouts having supplementary information.

Check out your stepper motor. Determine which pins belong to each winding, and of those, which is the center tap. (If necessary solder wires to the connections on the motor before lab, if you can, since there is only one iron in the lab). Measure the winding resistance, so you can determine maximum current. Come to lab with a motor drive circuit drawn (and perhaps built) that you will use for the lab.

11.2 Stepper Motor Control Circuit Construction and Testing:

1) Reserve space on your solderless breadboard for your microcontroller. It is a 16 pin DIP. Pins 1, 2, 3, and 4 of your HCS08QG8 microcontroller on the breadboard are connected back to the DEMO board (pins 13, 6, 1, and 3 of 20 pin connector J1 respectively) to provide /Reset, "BKGD (debugging support), VDD (3.3 Volts), VSS (Ground) respectively. See Figure 1 for the HCS08QG8 pinout diagram, Figure 2 for the identity of the different signals on the 32 pin DEMO board connector, and Figure 3 for the "Recommended Connections." We will not be using the optional external clock crystal, and for the Reset / Debugging support we will use the circuitry on the DEMO board. Do not yet remove the microcontroller from its socket on the DEMO board; we will do that later after it is programmed. Put a Ta capacitor from V_{DD} to V_{SS}.

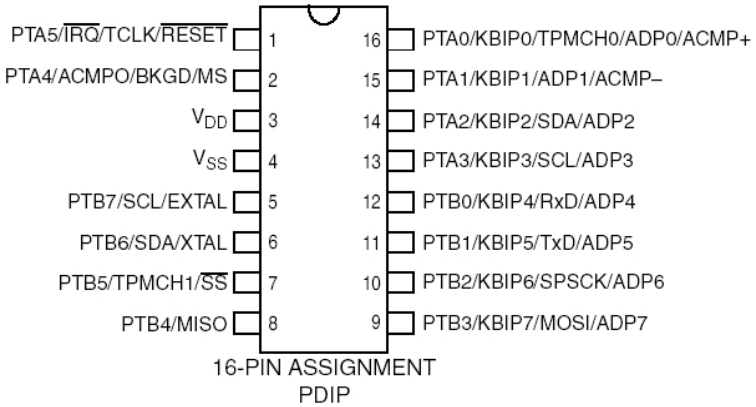

Figure 1 HCS08QG8 pinout diagram

VDD	1	2	PTA5/RESET*/IRQ*/TCLK
VSS	3	4	PTA5/ RESET*/IRQ*/TCLK
PTB1/KBI1P5/ADC1P5/TXD1	5	6	PTA4/BKGD/MS/ACMP10
PTB0/KBI1P4/ADC1P4/RXD1	7	8	PTB7/SCL1/EXTAL
PTA2/KBI1P2/ADC1P2/SDA1	9	10	PTB6/SDA1/XTAL
PTA3/KBI1P3/ADC1P3/SCL1	11	12	
PTA5/ RESET*/IRQ*/TCLK	13	14	
PTA0/KBI1P0/ADC1P0/TPM1CH0/AMCP+	15	16	
PTB3/KBI1P7/ADC1P7/MOSI1	17	18	PTA1/KBI1P1/ADC1P1/ACMP1-
PTB4/MISO1	19	20	PTA0/KBI1P0/ADC1P0/TPM1CH0/AMCP+
PTB2/KBI1P6/ADC1P6/SPSCK1	21	22	
PTB5/TPM1CH1/SS1	23	24	
PTA1/KBI1P1/ADC1P1/ACMP1-	25	26	
PTB6/SDA1/XTAL	27	28	
PTB7/SCL1/EXTAL	29	30	
PTA4/BKGD/MS/ACMP10	31	32	

Figure 2 DEMO9S08QG8 Connector J1 signals

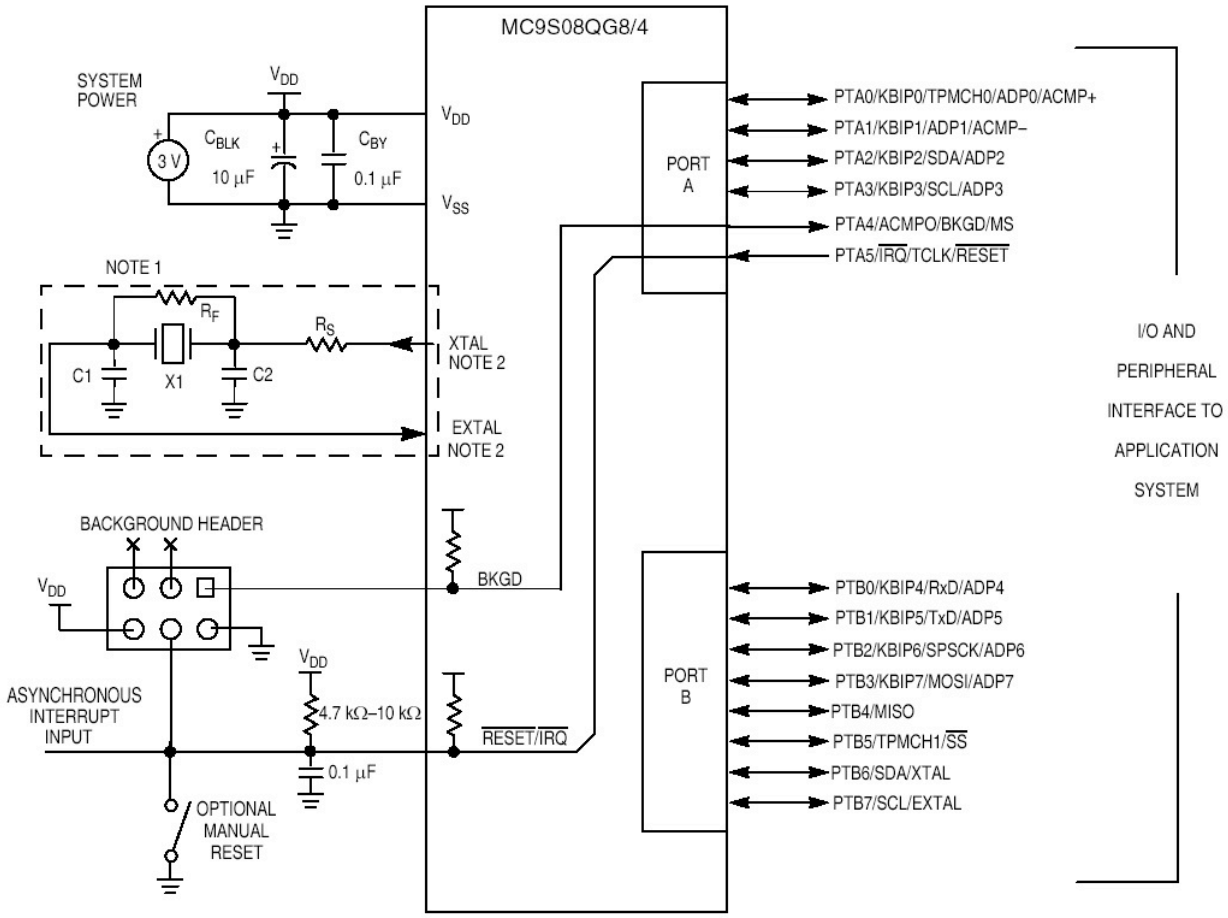

Figure 3 HCS08QG8 Recommended signal connections

Notice that pins PTA4 and PTA5 can be used in two different ways: as general purpose signals (as with PTA0 to PTA3) or as reset and debugging support. We want to be able to use them for debugging and reset, which means we only have four signals for PORT A that we can use. You want more? Buy the bigger 20 pin cousin of the device we are using.

2) Connect to Port B (Signals PTB0 to PTB7) on your breadboard to a set of DIP switches, so that an 8 bit binary number can be entered. We do not need pull-up resistors, since the microcontroller has those internally that we can enable. Closing a switch connects the pin to ground, making it "0". Opening the switch lets it float up to 3.3V, making it "1". We will use PORTA to drive the stepper motor.

3) A motor driver test circuit **(optional)** is shown in Figure 4 below. You can test your motor driver (described later) using a digital counter circuit that will give two square waves, with one lagging the other by 90 degrees. This could be used to check your motor drive electronics before you hook them up to the microcontroller, or to check them if the microcontroller does not seem to be doing the job. You could also use the microcontroller with the Lab 7 program.

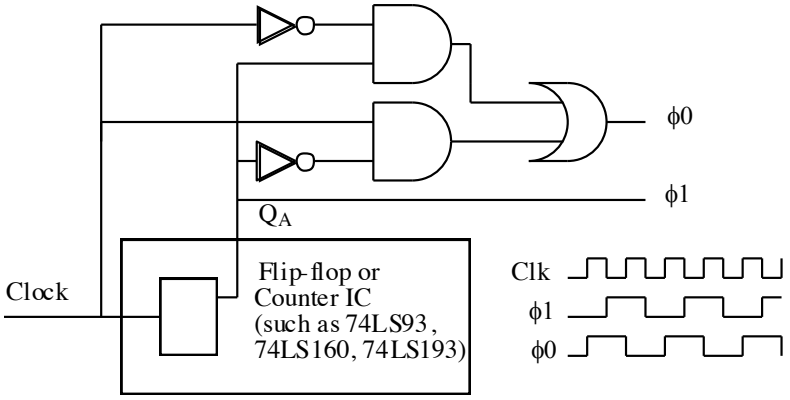

Figure 4 Stepper Motor Test Driver Circuit

(There are several ways to build this or a similar test circuit. You could use 3 NAND gates of a 74LS00 instead of AND and OR, or could use a single XOR gate (74LS86) instead of the five gates shown. You could also just invert the clock and feed it to a different flip/flop.) Be sure to set up your counter chip correctly, with an active-high "clear" grounded if necessary. The signal generator can supply the square wave, but make sure it has the DC offset on so that the voltage varies between 0 and about 4V (rather than +/- 5).

4) Design and build a stepper motor drive circuit on your solderless breadboard. Design it such that you use “Low end” drivers. Since we have a "unipolar" motor, we need only pull down switching. The "common" (center tap) for each motor winding should connect to 5 V. This is a fairly low power motor, so we can drive it with the PN2222 transistors, which have a pretty high Beta, and will have a low voltage drop when on. The power connection for the motor windings (the center tap) will be connected to 5 Volts power supply. You can determine the maximum current needed by the motor from the winding resistance. Assume a transistor Beta of 100. Select current limiting resistors and a series LED so that the transistor base current is no more than 10 mA. The LED’s will indicate when a given signal is ON (high). Expect LED’s to be about 1.6 Volts, and a “high” signal from the microcontroller to be about 3 Volts (not 5 Volts!). **Optional:** Test your stepper motor circuit with either the test circuit given above or with signals from the microcontroller stepper program of lab 7, using PTB6 and PTB7 (the signals driving the DEMO board LEDs) as inputs.

5) Connect the low order 4 pins of Port A (PTA0 to PTA3) on your solderless breadboard’s position to the motor drive circuit for your stepper motor. Now, everything on your breadboard should be ready to go, except that the microcontroller has not yet been programmed and placed into the circuit.

6) Figure out the sequence of 1's and 0's that need to go to each transistor to turn on the various phases of the motor in turn. For example, If Q0 to Q3 are the labels on the transistors connected to PTA0 to PTA3 respectively, and you want to turn on Q0 and Q3 and turn off the others, you will want to output the binary value 1001 (hexadecimal $09) from Port A. Figure out the entire sequence. You will need it for your program.

7) Bring up the "Code Warrior" Integrated Development Environment (IDE) on your computer, and write your program to control the stepper motor. Refer back to the material for Lab 7. Here's the way it should be organized:

a) You will need a variable that will be counted down for your delay timing.

b) In "initialization" section that sets up the ports. You want Port B to be all inputs, with pull-ups turned on. You want the bottom pins of Port A to be outputs. Furthermore, you want them to be "high strength" outputs. By setting the bottom 4 bits of the Port A drive strength register (PTADS) to 1, you make the maximum current 10 mA instead of just 2 mA. The initialization code precedes the "main loop."

c) In the "main loop" you go through 4 steps, each of which starts with changing the value in Port A to take the next step. Before the next step, you want a timing delay. But, this will be different from what was done in Lab 7, since you are using all 8 bits of Port B to get the delay constant that is counted down. Shift the constant to the left 7 bits (or multiply by 128). That converts the input value of 0-255 to have the value of 0-32640 instead, for longer delays. If you want very slow times (and slower speeds), make the variable a "long" instead of an "int" or "short." That makes decrementing slower. It also allows you to use a larger shift (or multiply) so that the initial value of the count can be larger than 32640. To avoid timing out during the delay between steps, you should either "shoot the dog" by including "SOPT1=0X52;" in the "initialization" or "feed the dog" during each countdown.

7) Make your project, and correct any programming errors as necessary until you have a compiled application to download into the microcontroller. Then, do a download to the microcontroller using the "DEBUG" button. Now the microcontroller should be ready to run. However, do NOT start it (yet).

8) Test your program. We want to do that with the microcontroller still on the DEMO board. But, the DEMO board has all those buttons, LED's potentiometer and photosensor connected to signals we are using differently. So, if you want to test it on the board, you must pull out the straps that connect those on-board peripherals. I suggest you remove all 6 "user" straps. Those are the ones grouped together labeled "USER EN", as shown in Figure 5. Don't lose the shorting straps! You will want them later. Leave the other three straps in their default configuration as shown in the figure.

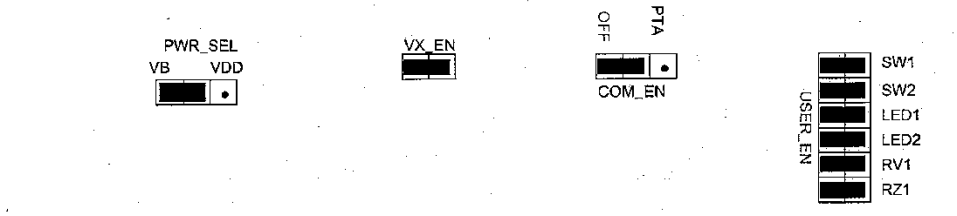

Figure 5 Shorting straps on DEMO9S08QG8 board

Now, start the program. Observe the signals on PTA0 to PTA3 and see if they are what you expect. Put one oscilloscope channel on PTA0, the other on PTA1, then PTA2, PTA3. Are you seeing the patterns you expect? You should see the 90 degree shift on transistors driving the

respective phases. You can also use the debugger to stop the program and observe what it does one step at a time, and set breakpoints, and see the count variable change when counting down.

Once you are satisfied that it is working, go on to the next step below.

9) Unplug the USB cable to the DEMO board. Remove your microcontroller from the DEMO board (carefully!) and place it on your solderless breadboard as shown in Figure 6. Plug the USB cable in again. The microcontroller should be powered. (Check it!) Now, apply 5 Volt power to your stepper circuit. The motor should run!

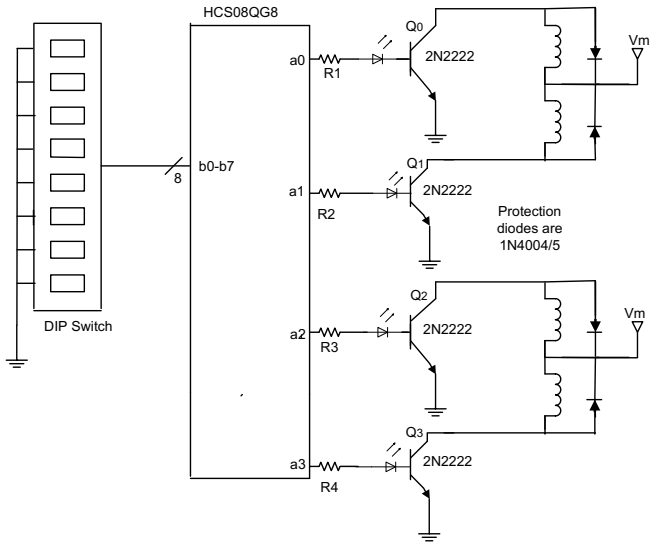

Figure 6 Stepper Motor Drive Circuit

10) You should still be able to use the debugger. Check and see. Within Code Warrior, give the "Debug" command and establish a connection. You don't need to erase and reprogram as you have made no changes. It should be possible to control the program from within the debugger just as when the microcontroller was on the DEMO board. But now, its connections are different and it is driving the motor.

11) Run your stepper and its controller in stand-alone mode: After turning off power and disconnecting the USB link, disconnect the microcontroller board from the DEMO board. Put 10K pull-up resistors (to VDD) on the now unused pins PTA4 and PTA5 (BKGDand /Reset) so that the microcontroller will operate in a "normal" stand-alone mode. Put 3 diodes (1N4148 is OK) from 5 Volt supply to VDD to supply about 3.0 Volts to the microcontroller. (You can go back to debug mode by reconnecting the Reset, BKGD, and Vss connections to the DEMO board, if you want to change the program.)

12) Observe the stepper motor speeds for three different settings of the DIP switch inputs to Port B. Note: Low values, especially near zero (all switches closed) may try to drive the motor faster than it can go. Be careful not to let the motor overheat. If it gets hot, give it a rest.

13) **(Optional)** Instead of using a binary value from 8 bits of Port B, change your program and hardware to use a potentiometer input. That means using the A/D converter. Channel 4 of the A/D converter is pin PTB0. So, to do this, disconnect the DIP switch from PTAB0 and tie that pin to the wiper of a potentiometer between V_{DD} and V_{SS}. In the initialization statements include:

```
ADCSC2=0;
ADCCFG=0;
```

These set up the A/D to operate fast, in 8 bit mode (the default). (Remember, you don't want the pull-up on this pin to be active now.) To read the potentiometer within the main loop, use the statements:

```
ADCSC1=4;  /* Put the channel number into the ADCSC1 register */
for(;ADCSC1&0x80==0;){} /* Wait for the top bit of ADCSC! To go to 1 */
i=ADCR;       /* Read the 8 bit A/D result as a 16 bit number */
```

These would replace reading the count variable from the DIP switch.

After reprogramming, the Potentiometer should control the speed of the stepper motor, with a high Voltage giving a slow speed and speed increasing exponentially as the Voltage goes down.

11.4 Report

The report for this lab is to be an informal report. Include your circuit diagram, including the HCS08QG8, the transistor motor drivers, and the motor and its connections. Include a copy of your C program (include comments) used to control the stepper motor speed using the DIP switch settings. Include the table giving at least 6 DIP switch settings and the corresponding measured and calculated speeds. (If you can, check one of these speeds with the strobe and report what you find.) How does the DIP switch setting correspond to speed? Say whether the project worked correctly.

11.5 For extra credit:

Modify your program so that the motor can run in either direction at the flip of a switch. Read some signal, maybe PTB7, to indicate the direction the motor is to turn.

Lab 11 report: **Names:**_____

11.6.1 Binary input / Stepper driver circuit

11.6.2 Table of results

Binary switch setting:	PTA0 period	Calculated speed	Measured speed
1. _____	_____	_____	_____
2. _____	_____	_____	_____
3. _____	_____	_____	_____
4. _____	_____	_____	_____
5. _____	_____	_____	_____
6. _____	_____	_____	_____
7. _____	_____	_____	_____

11.6.3 Remarks: At a minimum, provide statement of success, brief description of measurement and/or analysis method used to obtain information in 11.6.2 Table of Results, and a description of any stepper performance issues.

11.6.4 Your program: (Use additional paper or attach a separate sheet if necessary. Include comments.)