EGR 222 Mechatronics
Lab #12 Digital Motor Control
April 7-8,14-16, & 23, 2015

12.0 Objectives:

Understanding of basic digital control methods, including feedback for control, analog to digital conversion and pulse width modulation.

12.1 Pre-Lab assignment

Review the material in the book microcontrollers and C programming, and the programming that was used in earlier lab exercises 7 and 11. Also review text material on A/D conversion and pulse width modulation The HCS08QG8 A/D converter is a "successive approximation" type unit. You should look at the material about the A/D converter in the HCS08QG8 manual, to become familiar with the registers used and how they need to be set.

Given the limitations of time, we will merely do speed control rather than anything fancier. Review how you did speed control in the earlier motor control lab.

After reading this lab exercise, develop your program in C using the Code Warrior Integrated Development Environment.

12.2 Digital Speed Control Concept and Approach

The basic strategy for doing speed control digitally will be exactly the same as for the analog (electro-mechanical) system before. Speed will be measured using some method (we will use the generator) and compared to a user setting for "desired speed", and the motor given more or less energy as necessary to get these two quantities to balance. The diagram Figure 1 shows how we will do this digitally:

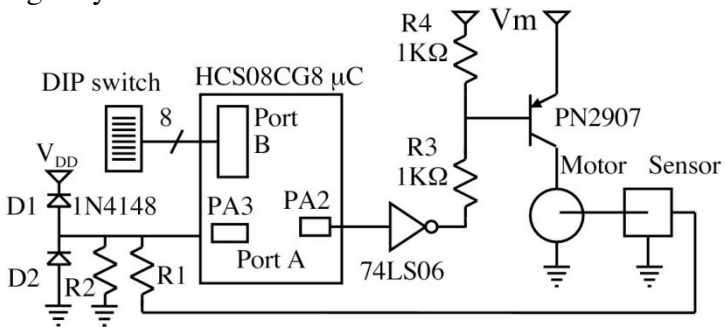

Figure 1 Conceptual Circuit

Notice that now we use a digital input (8 bits of binary from a DIP switch, just as for lab 11) for the desired speed. (We could use a small keyboard to enter this, if we wanted to and had the time to get fancy. Some telephone keypads are available if anyone wants to try that for extra credit, but do it this basic way first.) We use the A/D converter on the microprocessor to convert the analog speed information to a digital value. (If we wanted to get fancy, we could count pulses per unit time on a tachometer, but we want an excuse to use the A/D.) The comparison between actual and desired speed is done with digital arithmetic. Based on whether the difference is positive or negative, we can then increase or decrease the average voltage to the motor. But instead of doing it analog, we use off-on control, by varying the "duty cycle" of the (off/on) high end transistor circuit driving the motor. Notice the high end driver that we used earlier. We can get by with the small PN2907 transistor because it will always be either on

(saturated, with V_{CE}= .2 volts or so) or off (I_C=0), limiting dissipation to what the transistor can handle. You DO need a protection diode (not shown in the diagram, and maybe a capacitor too) to ground, since a voltage spike could do some harm in this case. (If you blow a transistor, which is very likely if you do not get saturation, then you could substitute a Darlington pair with a TIP32 + heat sink also, which is much more robust.) The 1K (R4) pull-up is to bypass the transistor when the 7406 is "off" (outputting a "1", or "high". Note that the 7406 can tolerate up to 30 Volts on the open collector (output), NOT on the chip power supply (pin 14).

The most basic way of programming the HCS08 to do its job is a relatively simple timing program with logic as follows, expressed in "C":

Start: Configure the various pins of Port A and port B, as well as the A/D converter.
Set the "Speed Count" to an initial value. Use a "short" integer (16 bits). Try a number between $10 to $100.)
Loop: Output a "0" (to turn power to the motor off)
Get the current speed from the A/D converter
Get the desired speed from the user input (port B)
Take the difference
Modify the current "Speed count" up or down, depending on + or – from subtracting.
Output a "1" (to turn power to the motor on)
Wait a time proportional to the "Speed Count"
Go back to Loop

To do this, we need a "variable", that is, a memory location in RAM that holds a value to be remembered and, as needed, modified. Since you are programming in C, the compiler decides where in memory this variable will be stored. The variable will need to be set to an initial value, perhaps $0100 will work. Whenever we see that the speed is too low (A/D output is less than the DIP switch setting), we want to increase this variable. By how much? If we bump it too much, our system may become unstable. You could make the change proportional (or even equal to) the difference, but at least to start try just going up or down one. (For extra credit if you have time, you can try other approaches to see if speed adjusts more quickly without going unstable.)

The average voltage to the motor is simply the supply voltage Vm (less the small V_{CE} drop) times the duty cycle of the waveform we put out of the pin of Port A. Figure 2 below illustrates:

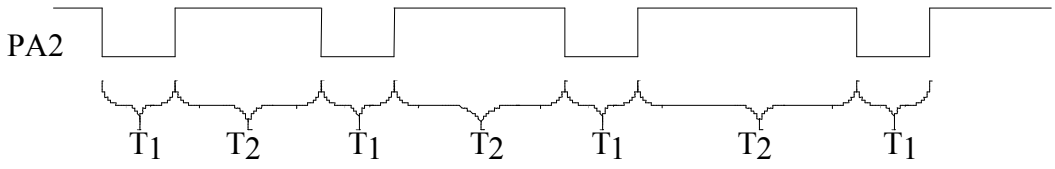

Figure 2 Example Timing Diagram

T1 is the time during which the Port A pin outputs a "0". This is when the inputs are sampled. The time for a "0" should always be about the same. On the other hand, the time T2, while a "1" goes out, will vary with the "Speed Count". Thus, increasing "Speed Count" increases the average voltage on the motor. Decreasing it reduced the average voltage on the motor. (There will be a minimum average voltage that occurs when "Speed Count" = 0. By making T1 longer,

we can decrease that minimum though it will slightly decrease the maximum average.) Note in the figure that T2 is increasing, perhaps because some load slowed down the motor, reducing its speed, and resulting in an increase in needed voltage (and power) to the motor.

The A/D converter is multiplexed, with inputs at one of the pins of Port A (or bottom 4 pins of Port B) as designated by the "Channel Number", which we store into register ADCSC1 to initiate data collection. Pins V_{ss} and V_{DD} (Ground and 3.3V power) indicate the voltages for the lowest (00000000) and highest (11111111) possible values of the A/D. (We could operate the A/D in 10 bit mode, so values would range from 0x0000 to 0x03ff hexadecimal instead.) We are using signed arithmetic. So we never want to see a value above 01111111 ($7f = 127 decimal) which is the largest positive number for a byte. Choose R1 and R2 to give a voltage in the range 0 to 1.5 V from the generator for the desired range of speeds. Do not let the voltage to this pin go above 3 Volts or it can damage the processor. (Diode D1 gives some protection from over Voltage, and D2 provides some protection from an accidental negative generator voltage.)

To initiate A/D conversion, control data (the channel corresponding to the A/D input pin) is written into the special register "ADCSC1" (A/D Status and Control Register 1). Bit 7 (the most significant) is a read-only flag that (when 1) indicates that the A/D conversion is complete. When a "write" is made to this register, the A/D process is initiated. A "3" in the bottom 4 bits initiates a conversion for pin PTA3.

The analog value collected is available in the 16 bit register "ADCR" upon completion of the conversion. We can tell when the conversions are complete by checking bit 7 "CCF" of ADCSC1.

12.3 Construction, Testing, and Operation

Use the DIP switches from the stepper motor lab for the desired speed inputs. Figure out, for your program, what values on the DIP switched correspond to what speeds. (You know the generator's constant for converting rpm's to Voltage.) Determine the needed resistor R1 and R2 values, and build your motor control circuit. (Use fairly large values, to avoid loading down the generator. You can always add other resistors in parallel across the generator to impose more load on the motor.) You should test motor control subsystem by itself (to make sure it turns the motor on and off) before hooking it up to your processor. Be sure to use a diode to protect your transistor! (One reason we are using a high end driver is that the 7406 will isolate the potentially high switching voltage spikes from your HCS08QG8.) Make sure the Voltage you get from the generator is positive and in the proper range. You should test other parts separately as well. Ultimately, you will download your program and then run it to control the speed of the motor.

Recommended Incremental Testing:

12.3a Testing the A/D converter

You could try first using a short program just to sample the analog values just to see that the A/D conversion works as expected. In debug mode, and bring it up in communication with your PC and single step, while watching the variables for registers and for your count variable, to see if it is working correctly (to increase or decrease the count variable) for when the analog value is either above or below the DIP switch value.

An extra experiment for extra credit: As the system is, the dynamics outside of the microcontroller are a second order system because you are exerting a force on a rotating mass, but if we only care about speed and not position this is effectively a first order system as for the analog speed control. You'd think it would be stable. It isn't, as you can see with an oscilloscope. The duty cycle jumps around, and you may even see noticeable jitters in the motor's speed. The effects of the microcontroller are both nonlinear and incorporate a state variable, the SPEEDCOUNT. This can make the system at least second order and, since nonlinear, potentially chaotic. You could try alternative systems that are both simpler and more complex, and see how they do. Simpler would be to simply cut off the motor when it is too fast, and cut it all the way on when it is too slow. How well does that work? More complex would be to add a fairly large capacitor across R2 (perhaps with a time constant on the order of .1 second). This smooths the generator voltage, eliminating noise spikes from commutation. It also adds another state variable to the system. Does this help, or hurt, the performance of the system?

Perhaps a better program would adjust the duty cycle while keeping the period of the PWM waveform constant. That makes the Voltage vary proportionally with the Speedcount, giving more linear operation (and hence, we would expect, less chaotic). The complexity of the program does increase a bit. Try to do this and see if it performs better. You could also try changing the period of the waveform and see what effect that has.

12.4 Report:

Submit an informal report that includes the circuit built, your program with comments, a short description of what the program does (your control strategy), and a table of results. Remark on whether it worked correctly, and if not, why as best you can determine. Also remark and give data as appropriate on anything else of interest, including anything for which you expect to receive extra credit, such as an LED display of speed.

Lab 12 report Names: _____

12.4.1 Your motor control circuit schematic (Port B input and from Port A output to the A/D input)

12.4.2 Your C language code (Attach separate sheet. Include comments.)

12.4.3 Observations:For line 6, repeat one of the previous DIP switch speed settings while steady moderate friction is applied to the coupler.

DIP switch setting	T1	T2	Duty cycle	Vm	Vg	Speed
1. _____	_____	_____	_____	_____	_____	_____
2. _____	_____	_____	_____	_____	_____	_____
3. _____	_____	_____	_____	_____	_____	_____
4. _____	_____	_____	_____	_____	_____	_____
5. _____	_____	_____	_____	_____	_____	_____
6. _____	_____	_____	_____	_____	_____	_____

12.4.4 Additional observations and comments