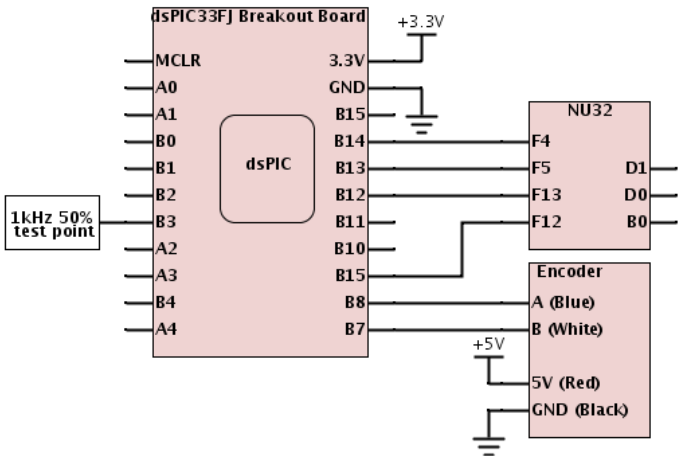
ME 333 Intro to Mechatronics
Winter 2014
Final Assignment, Part 1

Tip: Build your circuits neatly, with wires lying flat! This will make it much easier to debug, and you are less likely to have wires pull loose during transport.

Use the breakout boards for the dsPIC33FJ64MC802, A3909, and MAX9918, and the datasheets for the A3909 and MAX9918, to design and build the following circuits. Test the circuits by loading 'demo.hex' onto the NU32 with NU32utility and running 'demo_menu' in MATLAB. Be prepared to demo the diagnostics in 'demo_menu' in class on Thursday, March 6.

1. **External encoder counter circuit.** The dsPIC33FJ64MC802 is a 28-pin, 16-bit microcontroller with special motor control peripherals that are not included on the PIC32. We are interested in the Quadrature Encoder Interface (QEI) module, a peripheral that can count encoder A and B pulses with x4 decoding (meaning that 1 count is registered every time channel A or B goes high or low) and store the count as a 16-bit unsigned integer. The dsPIC33FJ64MC802 breakout board is programmed to run at 40MHz, and pin B3 outputs a 1kHz 50% duty cycle PWM that you can view to verify that the dsPIC is programmed and working. The dsPIC uses QEI1 to monitor encoder pulses on B7 and B8. The count is initialized to half the value of the largest possible 16-bit unsigned integer, or 65535/2. The Serial Peripheral Interface (SPI) module is used to communicate with the NU32. SPI1 is setup in slave mode with SS on B15, SDI on B14, SDO on B13, and SCK on B12. On the NU32, SS is F12, SDI is F4, SDO is F5, and SCK is F13. When the SPI master (the NU32) sends a '1' to the dsPIC, the dsPIC replies with the QEI count. When the SPI master sends a '0', the dsPIC resets the count to 65535/2 and returns that value.
    a. Build the dsPIC, encoder and NU32 circuit.



    b. Load 'demo.hex' using NU32utility. Open MATLAB and use the diagnostics provided in 'demo_menu' to read the encoder. Rotate the motor shaft one revolution and check that the encoder count changes by 396. (Remember that your encoder has 99 lines per revolution, and we are using x4 decoding.)

2. **H-bridge circuit.** The A3909 H-bridge breakout board breaks out all of the pins on the A3909 and places a 10 uF and a 0.1 uF capacitor between VBB and GND. The A3909 can output more current when configured for parallel operation: two H-bridges operate in parallel, so the available current is doubled. The truth table for the paralleled A3909 shows that when both inputs are low, the motor will coast. When both inputs are high, the motor will brake, and when the inputs are opposite we get forward or reverse.
    a. Wire the A3909 to the NU32 in parallel configuration, using OC1 as IN1 and OC2 as IN2, and use the 6 V battery pack as VBB. Put the motor between OUT1 and OUT2.
    b. Use the diagnostics in 'demo_menu' to set the PWM and check that you can control the speed and direction of the motor.

3. **Current sensor circuit.** The MAX9918 breakout board gives access to all of the pins on the MAX9918 and puts a 0.015 Ω current sensing resistor between R+ and R-. Reading the data sheet, you see that the MAX9918 amplifies the voltage across the sensing resistor by a factor of (1 + (R2/R1)), sums this with the offset reference voltage on the REF pin, and puts the resulting voltage on the OUT pin. When the current from the H-bridge is passed through the current sense resistor, the MAX9918 generates an output voltage proportional to the current (but offset so the output voltage is always positive).
    a. Build the MAX9918 current sensor circuit, using a voltage divider to supply REF with 1.65 V, the midpoint of the PIC32's ADC input voltage range 0-3.3 V. Calculate a gain so that the OUT voltage fills the range 0 V to 3.3 V when the current is -1.5 A to 1.5 A. Build the gain using the resistors you have (getting reasonably close to the desired gain), keeping in mind "low Z feeds high Z" in relation to your voltage divider circuit supplying REF.
    b. You will use the current sensor to read the H-bridge current. Because the PWM is at 20 kHz, there will be variations in the sensor current at 20 kHz. We are not interested in these high-speed variations of the current, but rather "average" current at a lower frequency. To get this, build an RC low-pass filter with a cutoff frequency of approximately $f = 1 \text{ kHz} = 1/(2\pi RC)$. The input to the RC LPF is the MAX9918 OUT pin, and the output of the LPF goes to the NU32's pin B0 (which will be used as an analog input).
    c. Use your nScope and the diagnostics in 'demo_menu' to view the analog signal from the low pass filter. Experiment with the unloaded motor spinning at a high speed (large PWM duty cycle) vs. the motor stalled under the same PWM. Do you see the current increase when the motor is stalled?