MILESTONE 4: INTERFACING

Please see the safety warning in Instructions

LAB OBJECTIVES

The objectives of this introductory lab are the following:

1. Learn how to interface with simple motor drivers (*i.e.* amplifiers for actuators).
2. Learn how to implement pulse width modulation (PWM) on an MCU and apply it to a motor driver to change the velocity of a motor.
3. Learn how to implement the analog-to-digital converter (ADC) on an MCU.

DELIVERABLES

1. Provide a demonstration and your driver code for the simple brushed DC motor rotating in both clockwise (CW) and counter clockwise (CCW) directions at lowest possible (just beyond the 'dead-zone'), mid-range (wrt the dead-zone) and maximum velocity.
2. Submit Worksheet 1
3. Provide a demonstration and your driver code for the DC stepper motor for the operation of the motor in both CW and CCW in relative increments of 30°,60°, and 180°.
4. Submit Worksheet 2
5. Provide a demonstration and your driver code for the onboard ADC and provide a continuous eight bit demonstration using your LED bank and a potentiometer circuit as the input device.

TECHTALK

1. Safety around high voltage/current sources, and precautionary measures.
2. Sourcing parts for devices from major distributors (i.e. digikey.com, mcmaster.com), and understanding lead-time

INSTRUCTIONS

SAFTEY WARNING: Interfacing with sensors and actuators can be a tedious task. Designing the correct interface circuit, searching for the MCU setup parameters and writing the driver software is a time consuming and often fatiguing process. This is by no means a time when you can become complacent as you must ensure that all voltage requirements and safety precautions are met so you will ensure that you do not damage the devices,

harm yourselves or harm others! You will be in close proximity to equipment that will eventually carry currents of up to 4 A. Ensure that you always consult all documentation prior to interfacing the devices. If you are unsure, please ask the teaching assistant or laboratory manager to double check your circuit before powering up the motor drivers.

In class and your previous completed milestone exercises, you learned how to provide output signals to LEDs and you created a millisecond timer to perform simple timing routines. You also recently learned the fundamentals of interrupts, and you observed a demonstration in the debugger of how an interrupt works and how the registers change with unexpected inputs. You will build upon this knowledge, and develop interface 'drivers' for three purposes including: *(i)* DC motor operation with directional and velocity control *(ii)* DC stepper motor operation using PWM and *(iii)* using the onboard ADC to obtain an arbitrary analog signal. These 3 exercises are well partitioned into two areas, so if you are using the 'divide-and-conquer' approach ensure you debrief your partner on the details of your work.

NOTE: This milestone exercise is designed to **improve your skills in finding the required details in the manufacturer documentation**, and gradually increases with difficulty throughout the milestone write-up. The initial stage within this milestone write-up (Part I) has minimal requirements to source information from the documentation. Part II will very strongly point you in the right direction, while Part III of this milestone write-up will give you a general area of the manual to source information and only final output details on what is required. It is strongly recommended, that you work as a team in order to order to satisfy the specifications required and reduce the amount of time that is needed to complete the requirements.

In the first section of the milestone write-up you will be required to interface with two DC motor drivers which will provide power to a simple DC motor and a stepper motor. Ensure that you follow the instructions, read the manual where necessary, and most importantly: ask questions when in doubt. This is your opportunity to learn with the luxury of having instructors, TAs, and lab technicians for advice before you venture off into industry where you will be required to provide bug-free and safe mechatronics system designs that will potentially be duplicated thousands of times.

INTERFACING PART I: THE DC STEPPER MOTOR DRIVER

Read through this entire section before hooking anything up.

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. These motors are typically used in low-power position applications. When the electrical pulses are applied in proper sequence, the shaft of the motor rotates discretely. Unlike continuous-drive actuators, they move in accurate angular increments, or steps, in response to the application of digital pulses to an electric drive circuit. The number, rate and sequence of the pulses control the position, speed and the rotational direction of the stepper motor shaft. Thus, counting the number of pulses into the stepper motor determines the angular location of the rotor. A stepper motor can provide some advantages to a regular simple DC brushed motor type that you will use in Part II. A stepper motor has the ability to operate in an open loop circuit (no position feedback) and can position itself relative to its starting position (at the time of power up). It can also be accurate to several degrees or a fraction thereof (dependant on the model of the stepper motor). That said, the motor must still 'know' where it's starting point is, so limit switches may be required to reset the stepper motor during an initialization routine or after a power-loss condition. A limit switch is a switch that is active when a motor is in a unique position. You will learn more about stepper motors during Module II of the mechatronics course.

Setting up DC stepper motor drivers can be done in many ways. The method done in this lab is by no means the only method, nor is it the best method available; however, it will give you a basic understanding on how to interface a stepper motor driver to a stepper motor. There are two classes of DC stepper motors: unipolor and bipolar stepper motors. In this exercise you will learn how to hook up a unipolar stepper motor and send the correct signals to the stepper motor driver to actuate the motor a finite amount of steps.

Figure 1 shows a simplified schematic of the coils in the stepper motor and how they are connected to the driver circuity. This schematic is based on the stepper motor configuration that you will use in your lab exercise. To understand all the connections required, please refer to the manufacturers documentation also posted on the course website (K-MD Kit). A simple diagram of the inputs to the K-CMD will be drawn on the whiteboard for you by the TAs, and a worksheet as been provided for you in Appendix A – Worksheet 1

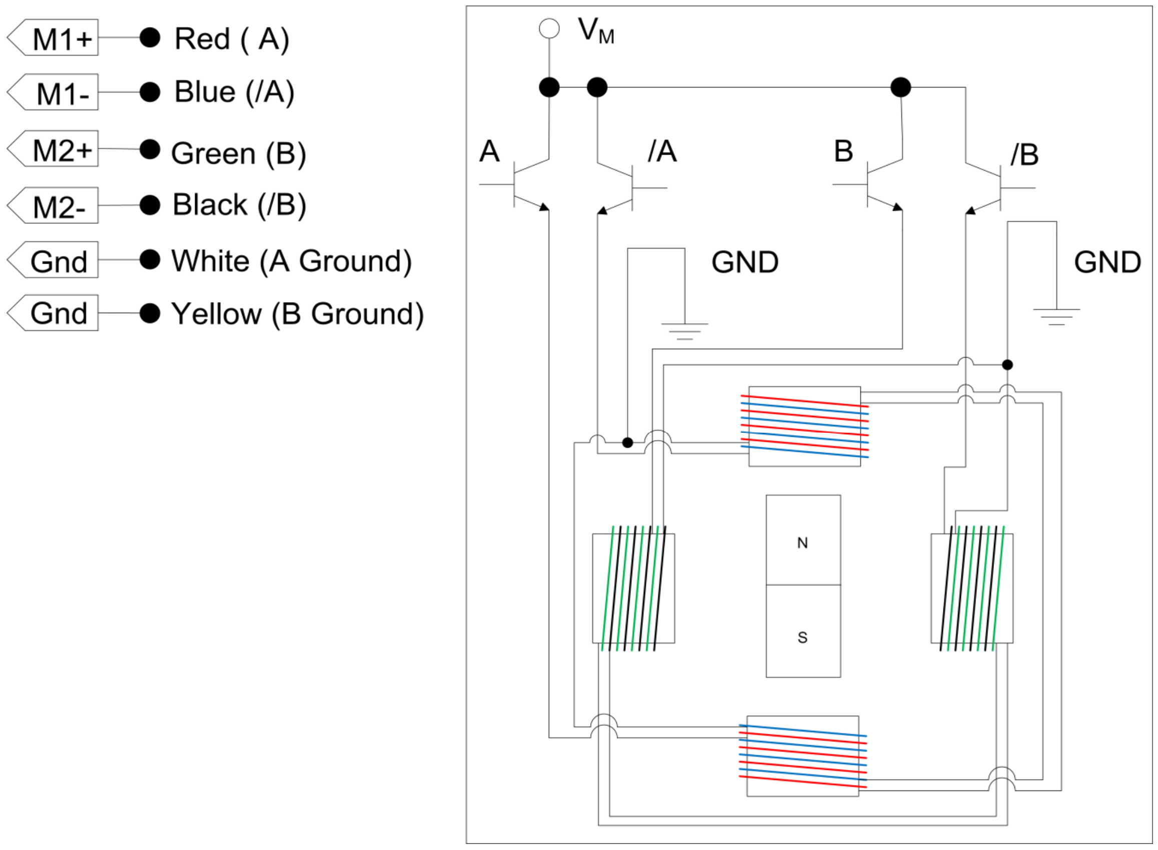

Figure 1. Stepper motor schematic

In order to actuate the stepper motor, as you may have guessed the coils need to be energized in the right order otherwise you will may get forces that conflict with each other resulting in no motion, or purely erratic motion. To generate the simplest sequence in order to achieve motion of the stepper motor, use the following order in Table I.

Table I. Stepper motor control signal order

	Coil 1 (M1)			Coil 2 (M2)		
Step	**Enable 0**	**I1**	**I2**	**Enable 1**	**I3**	**I4**
1	1	1	0	0	X	X
2	1	0	1	0	X	X
3	0	X	X	1	1	0
4	0	X	X	1	0	1

This is a simple board that supports a popular motor driver chip known as the L298 dual full bridge driver. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. For more information, please find manufacturer documentation on the L298 using any search engine on the web. A worksheet has been provided for you and can be found in Appendix A.

INTERFACING PART II: A DC MOTOR DRIVER WITH SPEED CONTROL

THE DC MOTOR DRIVER

DC motors are used widely in speed control and high ratio of torque to rotor inertial makes DC motors attractive for many applications. These motors are used for automatic processes and machinery applications that require easy control with fast adjustable speeds and high starting torque.

First, you're going to have to provide the physical connects between the MCU and the motor driver. Here's an overall block diagram of the setup in Figure 2. Notice, that the logic power and the motor power supply are separated. This is a preferred setup to ensure that the logic circuitry is not inadvertently affected or damaged by the high voltage or electrical disturbances from the motors (noise).

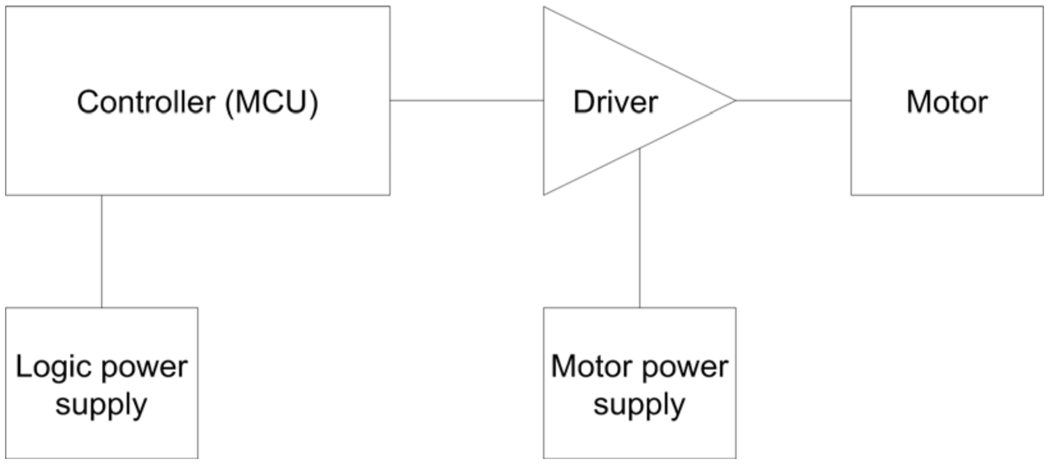

Figure 2. Block diagram of the DC motor subsystem.

The next step requires you to read through some of the manufacturer documentation. Luckily for you, the I/O table that you need has been provided on a worksheet (see Appendix A – Worksheet 2). Please see the course website for the full manufacturer documentation (Pololu Motor Driver). Observe the two responses that you require from the motor driver (CW and CCW), determine the inputs required for each state and provide the necessary connections from the MCU port pins to the motor driver ports. I STRONGLY advise you to design and implement your software first. Test your software using LED output to determine if you are getting the proper output BEFORE you hook up the system. This will ensure you do not see blue smoke during the development stage.

Below in Figure 3, a DC motor configuration with 2 poles is shown. With the application of current on the coil, magnetic flux will be generated and the interaction between the stator poles and the rotor poles will result in the torque which causes rotation. The polarization of the rotor will be the same as long as the brushes and the corresponding commutators are in contact. However, for this configuration, after 180 degrees of rotation, rotor reaches the commutation point, where the brushes move on to the neighboring commutators which will reverse the polarization on the rotor. This results in continuous rotation.

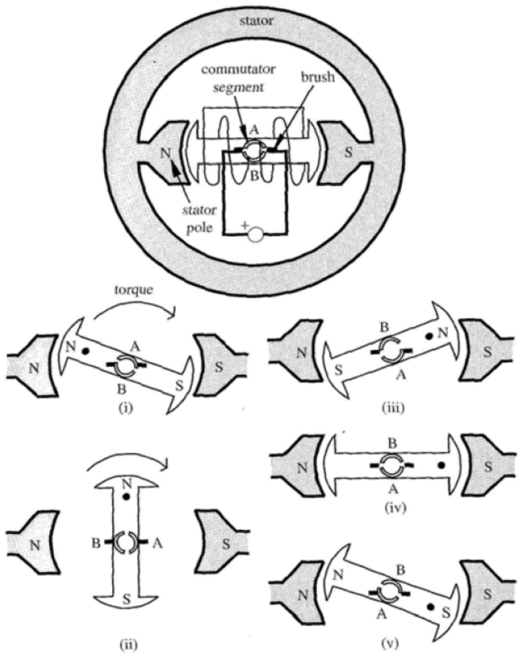

Figure *3*. Armature field- stator field interaction

More information will be provided for you about the operation of such motors during your Module 2: Sensors and Actuators series.

The driver you are required to implement will interface the controlling functionality of the MCU with the functionality of the motor driver. There are two requirements necessary to integrate this software driver with the hardware driver. Firstly, you will need to implement a PWM signal from your MCU using some of the MCU special functions. This PWM signal controls the voltage level that is applied to the coils of the DC motor thereby controlling the velocity of the motor. Ensure that you understand that although this may seem like an good solution in controlling a motor, there is absolutely no ‘feed-back’ from the motor itself. Therefore, the motors are ‘flying blind’, or are open-loop (as opposed to closed-loop). If you want your motor to spin at a constant velocity with many different loads in many different environmental conditions you require a form of feedback (e.g. motor encoders to provide position indication), and a control method (e.g. proportional-integral-derivative). This would ensure that the controller would respond to an external feedback and act appropriately (increase the voltage, or decrease the voltage) to provide the torque or velocity that you require for the task. This will not be covered during these lab exercises, but could be applied with your MCU such as the one you have with the addition of a position encoder on the motor connected to your I/O ports.

SETTING UP THE MCU FOR PULSE WIDTH MODULATION OUTPUT

The second step to in setting up this DC motor driver involves initializing the PWM signal to produce the signal that will control the voltage applied to energize the coils of the motor. This is a non-trivial task and you will be given some strong hints on how to set up the MCU modules (i.e. set up the timer, and the PWM port). You will have to determine which bit in the register to set after you have read the description from the comments in the steps that are provided.

The difference between using the timer for this PWM signal and the timer used in the last exercise is the use of interrupts. The settings below will allow the interrupts to be 'self-managed'. This means that the MCU will take care of resetting the flags. In the end, you'll have an output that looks similar to the following figure. Please verify this with an oscilloscope to make you believe that this is actually happening, once you get things configured. Once again, you can easily see the transitions of the output in your debugger (just ensure you set the scaling factor to something that will not make your finger get tired from clicking the mouse or hitting F11 – Step).

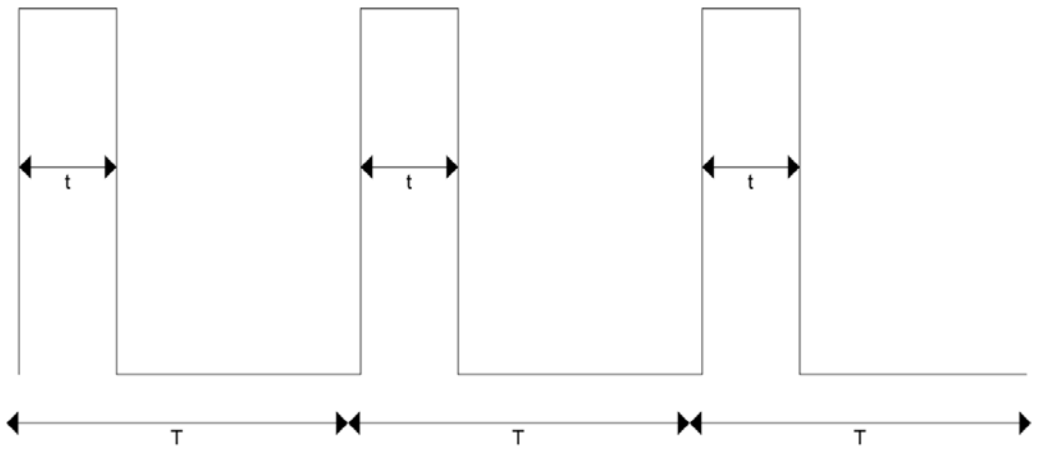

Figure 4. A pulse wave modulation signal

The signal in Figure 4 is an example of a pulse wave signal. This signal provides a method of control. In this lab exercise you will use it to control the amount of voltage applied to a DC motor which subsequently turns the shaft. The Pololu chip will accept up to 20 kHz and I tested it with periodic frequencies as low as 700 Hz. The period, *T*, of this signal depends on the period that you will set on the MCU (discussed below). The duty cycle is the ratio of the width of the pulse *t* and the period T , $\left(DC = \frac{t}{T}\right)$. You can control the width of the pulse by entering a value into an output compare register. The MCU will activate the eight bit timer and count from $0x00_H$. Each value of the timer is compared in hardware to a value that you will programmatically (written in your program) enter into an output compare register. Once the value in the timer is equal to the value in the output compare register one of the output compare pins will toggle (go from 1 → 0) and remain at logic 0 until the timer reaches ($0xFF_H$). The opposite output is also possible (0 → 1) depending on how you set up the registers. This is only one way to use the timer and PWM, and many more methods exist. First, learn how to use the timer to generate a PWM signal in the method that will be described, and then 'tweak' it to you needs if desired.

The following description will assist you in setting up the PWM; however, you need to look through the documentation (i.e. this is the second part where you need to do some searching). Extremely strong hints will guide you to the right place in the manual on how to set up PWM. If you go through each of the points successfully, your PWM should work. You can calculate the signal of your PWM before you begin; however, understand that the output voltage of the motor driver depends on the RATIO of the duty cycle and not the actual period itself in this case. Don't spend too much time finding the optimal frequency on paper. Get this working and then modify the settings and see how the DC changes.

Follow these directions in order to set up fast-mode PWM on your MCU using output compare A (OCA) of Timer 0. Ensure you test the output in the debugger to guarantee that you are setting the correct bits in the registers. Note, the documentation refers to MAX as the maximum value of the 8 or 16 bit register and TOP as the value that the output compare is set to.

STEP 1. Set the timer to Fast PWM mode with the TOP equal to the maximum value of the 8 bit register that is possible. The output compare register A (OCRA) should be updated at TOP, and the timer overflow (TOV) flag should be set on MAX (see page 113 and onward). Fig. 5 illustrates the output of the PWM in comparison to the timer value stored in TCNT. Note, the 3 bits required to set the values of WGM are broken up and spread over two registers Timer Control Register 0 A and Timer Control Register 0 B. This is not nice, but this is the way the manufacturer decided to do this.

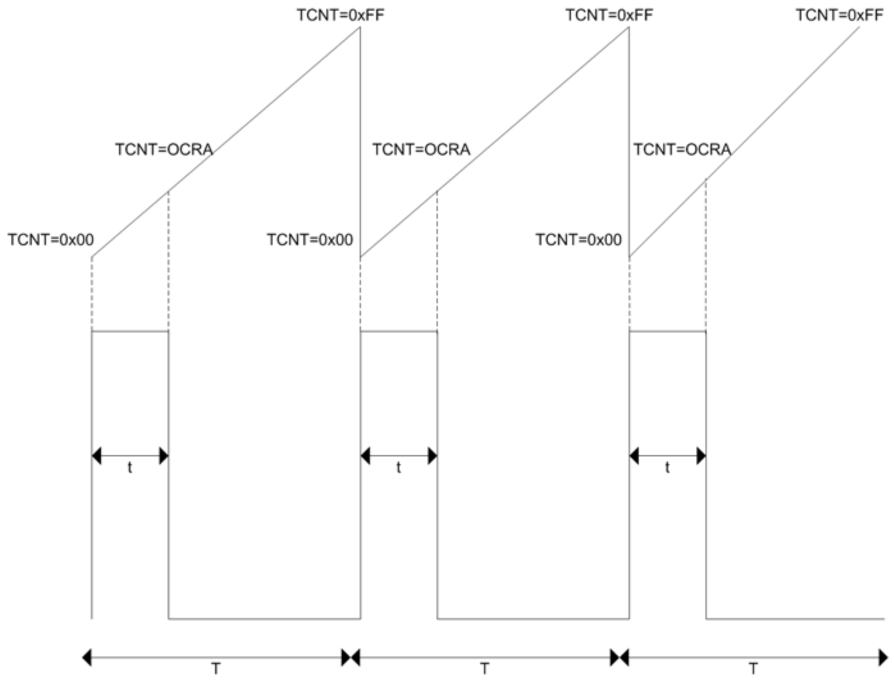

Figure 5. The value of the timer TCNT (top) in comparison to the PWM output (bottom)

STEP 2. Enable output compare interrupt enable (OCIE0A) for Timer 0 (See Page 115 and onward) in the Timer/Counter Interrupt Mask register (TIMSK0). Use timer/counter0 output compare match A interrupt enable. Read the description to find out what this does, and set the required bits.

STEP 3. In the Timer/Counter Control Register A (see Page 110 on onward), set the compare match output mode to clear (change to 0) on a compare match + set the output compare A (change to 1) when the timer reaches TOP (the value of the timer that you wanted to match the output compare A register to) (Hint, we are using Fast PWM mode).

STEP 4. Set the Clock Select bits (clock prescalar) in the Timer/Counter Control Register B (pg 113 and onward) to a reasonable value for the period of the PWM (*T*) signal for now, and experiment by changing this value later. Note, the ranges of PWM that the Pololu motor driver can accept is up to 20 kHz

STEP 5. Set the value of the Output Compare Register A (OCRA) to the value you want the PWM signal to be reset at (sets the duty cycle of the PWM signal – See Fig. 5).

STEP 6. The output compare used in these steps is OC0A. Once again, this pin will generate the PWM signal for the Pololu motor driver. Determine which PORT OC0A exists on and change the appropriate data direction register pin to output (HINT: remember the diagram that I showed you in class, this has the answer on it, look hard). This will also be the pin you measure the PWM signal on.

STEP 7. Finally once EVERYTHING has been set, enable the global interrupts using the same method discussed in class during the demonstration. You PWM will be active. You should try this in the debugger. You can set one of the PORTS equal to TCNT to see its current value and you will see OC0A oscillate between 1 and 0. If you do try this, understand that OCRA will take one full cycle to update; therefore, run through one full clock cycle and then look for the output to occur. Finally upload the code to you MCU and measure the output using an oscilloscope. If you don't know how to use an oscilloscope the TA's will help you.

CONFIGURATION THE DC MOTOR CONTROL SIGNALS

Now that you've set up the motor I/O lines, and the PWM signal, and have tested these signals for the correct outputs (right?), you are nearly ready to hook up to the motor and begin testing. Please complete Worksheet 2, prior to making any physical connections. This worksheet contains a truth table indicating the 4 necessary control signals (not including PWM) that must be provided to control the motor. The 2 operating modes that you want to perform are Clockwise (CW), and counterclockwise (CCW). Please disregard the 'break to …' options for now.

Also, recall, that I said a kill switch was a good idea. If you unplug the plug as your form of kill switch you are generating a LARGE change in current which can potentially damage your device. It is possible to either disrupt your program which is a good idea in the final milestone, OR you can also disrupt the PWM signal as well. Most importantly, you want to remove any signal that will cause any motor to move and cause further damage to life or property when in a critical situations.

INTERFACING PART III: ANALOG TO DIGITAL MEASUREMENTS

Analog to digital measurements are frequently performed in day-to-day MCU applications. The ability to digitize an electric potential measurement based on a relative value allows us to monitor environmental conditions such as temperature, pressure, etc, and to process these measurements and react accordingly with the output of an MCU based system.

In this lab exercise you will set up an ADC channel using PORTF of the MCU (see Chapter 25: Analog to Digital Measurements, page 313 in the Full manual). You will only be given the basic specifications of the circuit and you must provide read through the documentation and provide a solution. You should now have the knowledge in how to seek information on registers, how to deal with interrupts and set up basic circuits. The information discussed in class on the ADC fundamentals should provide you with the basic understanding needed to configure this circuit properly.

ADC CIRCUIT SPECIFICATIONS

You are required to build the simple circuit in Figure 6 which consists of potentiometer that uses the MCU V_{CC} as the power supply, and write ADC driver code explicitly outlining the choices you made in setting up the required registers for ADC conversion. You will use this code again during Milestone 5, so ensure that it is self-contained, and modular (easy to use in any program).

To create the circuit required for ADC demonstration, attach the 'wiper' of the potentiometer to an input on Port F (try 0 or 1), and configure the MCU to sample the analog voltage input and display the result on your eight bit display (ensure you read the documentation).

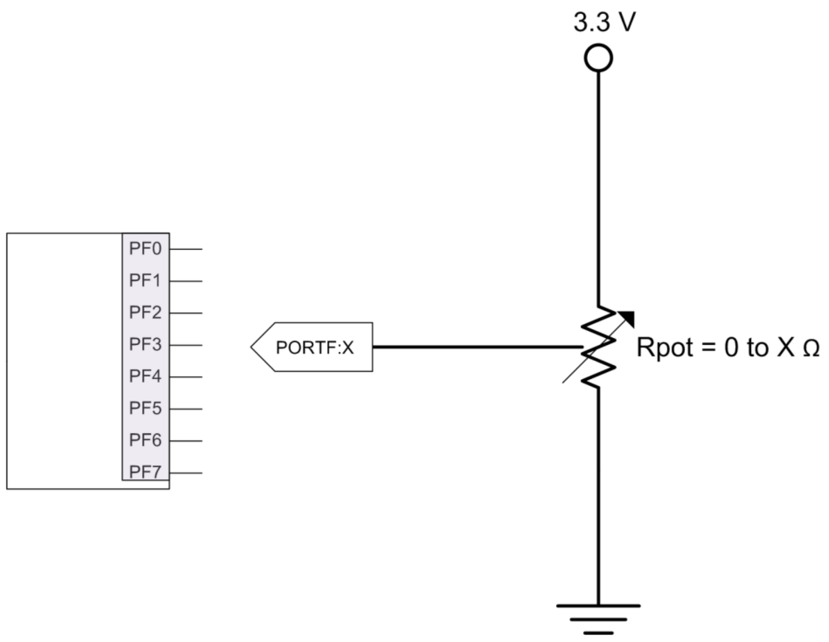

Figure 6. Simple analog-to-digital circuit with a potentiometer as the input