Physics 335 – Analog Output with a PIC

May 10, 2010

1 Introduction

Last week we introduced you to the PIC microprocessor. This week, you'll get a little more practice programming. In addition, you'll have to design your own circuit. This one will be simple, but your own. We'll use the AD558 8 bit Digital to Analog converters, which you are already familiar with from the A/D lab, in addition to our PIC Processor. So hopefully it won't be over taxing for you. In order to complete the first couple sections, you'll need to write some code that includes conditional branching. The following are simple variations on what you know already, but hopefully will give you a little more programming practice in preparation for the coming test. The focus is on conditional branching and allowing you to make some slightly more complicated programs than the simple ones you wrote last week.

2 First things first

Get the Datasheet for the AD558 from the course website. Using this information, and your knowlege of the PIC microprocessor, draw a circuit diagram that will allow you to hook the 8 bit D/A converter up to the microprocessor so that you can get data from the microprocessor in some form and turn it into an analog voltage output.

Carefully diagram the entire circuit. You should be certain you include all connections for each of the chips in question. It is ok in some cases to leave certain pins unconnected or tied to a voltage rail, but they should still be noted on the diagram and included, with or without relative connections. If more than one wire is to be hooked together, you should indicate it with a solid dot where they join to indicate a junction.

3 A test program

Now that you have your initial circuit, you should begin thinking about some sort of program to test it.

We'll use, again, the simplest program we can think of to do so. We'll start with a Sawtooth Wave.

Write a program that:

- Declares some register variables for your use. You may find the CBLOCK statement useful
- Has an 8 bit output value that will be the 8 bit input to the D/A D-ports
- Outputs that value to an appropriate port (be sure you set TRIS appropriate in setup)

- Increments that register
- Loops back and starts over

We'll just let that register overflow – In other words, when our 8 bit value gets to H'FF', it's as big as it can get in an 8 bit register. Incrementing it again, the value "Overflows" back to H'00', as you know from working previously with counters. So our register slowly "grows" until it suddenly finds itself overflowing back to 0.

For the moment, don't worry about the frequency of the sawtooth. We'll come back to that later, if you have time.

You may find that your sawtooth wave has lots of little "spikeiness" on it, you are seeing the digital "switching" transitions. You can get rid of most of this by doing a good job of "bypassing" the supplies with capacitors. This is the bane of mixed signal (analog and digital) circuits and it can be a real pain to get rid of it. To get a nice looking waveform, I found I needed bypass capacitors between the +5 Volt and ground between the microprocessor and D/A chip on both supply busses and it also helped greatly to add a bypass cap right over the supply for the chip. If it bothers you, try to clean things up a little, but don't spend a ton of time on it. Whole courses and books are devoted to solving problems like these and a reasonable looking sawtooth wave will be adequate for today.

Have your TA check off a functional saw tooth generator and your diagram.

4 Variation

Now a slight variation. Modify your code so that you now produce a triangle wave generator. To do so, you'll need some sort of conditional test to see if we're at the "top" or "bottom" of the sawtooth. Look at the PIC datasheet to see what sorts of conditional branching instructions the processor allows.

Review your class notes. There are a number of ways to do this. A programatically "graceful" way is by using various bits of the status register. We won't restrict you to that if your heart is set on another method, but you should at least have a concept of how it should be done. If you use this method, be sure you note which processor flags are set for each instruciton.

5 More Practice Using Conditional Branching

Now that you've ensured your circuit works and is outputing an appropriate analog value, practice using conditional branching statements to create a "Serated Sawtooth" waveform. In other words, a waveform that will increase for a short period (say, 16 steps), then descrese for a shorter period (say, 8 steps), then increase again. Some things that may be useful to your are bit test and zero test functions, etc. When you get done, you should generate a sawtooth waveform with nasty little teeth.

6 Definite frequency

Frequency Variation is also possible of course, if you know how to do the timing.

Using appropriate delays, code a variation that outputs approximately a 50 Hz triangle wave. You can use NOP instructions or similar to tune the timing if needed.

7 Bonus variations

If you succeed at all of the above, you can try some additional variations.

Use a second port, configured as an input, to allow for an 8 bit input of a binary value that is a "Divide by" value... i.e. we can have anything from a divide by 15 to a maximum value. You can use the zero case to output a DC value.

This is trickier than in sounds at first, for a couple reasons. First, remember that checking your input port will take some time too, and that will affect your frequency and perhaps the shape of your waveform if you're not careful. Also, be careful of the DC case. Finally, note that the limited instruction set of the 16F84A does not have a divide by instruction. You'll have to implement this functionality in some other way. There are of course microcontrollers that do have divide instructions. This is one of many engineering compromises you'll probably make in your life to make "what you need" work with "what you're given". In this case it isn't too hard. In some cases, one little detail will give you months of headaches.

If you're up for a further challenge, try using a table in program memory to output a rudimentary sine wave or some other waveform. This is considerably trickier yet, but quite doable. Still, we expect only the real go-getters of the class to get this far. Don't worry if you don't... You'll have time to work with memory tables in next week's lab. If you do go for it, a simple sampling of 16 or so points from a sine wave will be adequate to demonstrate you know what's going on.

8 Writeup

For a writeup, include a circuit diagram of your circuit, a brief description of how it works, and a physical printout for each group member of your working code. You should include brief comments for how it works. Full credit will be granted only to fully functional circuits. A small incentive of 10% bonus for particularly tidy and readable code or 10% punishment for particularly obscure and unreadable code can be assessed at the discression of your TA. Save an extra copy of the final code in the MyDocuments Folder ont he computer with the name 'PicWaveformXX-YourLastNames', as well as retaining a copy for yourself on your UW account or flash drive.