PreLab 7: DAC INL and DNL

In this lab, we will design the function to test the INL and DNL specifications of the TLV5616 DAC.

a) In order to test all codes, we need a very large integer array. How could you program this array so that you don't have to input the codes for an 4086 element array? Given the datasheet specifications of INL and DNL, how many codes should actually be tested?
b) Given the datasheet specifications, how should DNL be calculated?
c) Given the datahseet specifications, how should INL be calculated? Which algorithm should be used to calculate the ideal line?
d) I need to serially load Din for each code. To do this, I need an array of 1's and 0's that can be shifted into the DAC. If I have an array of integers (which will be represented as a stream of 1's and 0's in the computer), how can I convert each integer into a array of 1's and 0's representing the same number? e.g.
   int data[3] = {7,8,9};
   This will look like this in the computer:
   int data[3] = {000000000111, 000000001000,000000001001};
   However, to load a 7 into the DAC, I need:
   int serialdata[12] = {0,0,0,0,0,0,0,0,0,1,1,1}
   to load an 8 into the DAC, I need:
   int serialdata[12] = {0,0,0,0,0,0,0,0,1,0,0,0}
   How could you make this happen?
e) When you get on the tester, how will you know that our code is working correctly? (Hint: it is not enough to see that INL and DNL pass. They could pass with bad data.)

Lab 7 - DAC INL and DNL

In this lab, you will develop the INL and DNL test. Since this is a resistive string DAC, you will need data from all codes in the dataconverter. Of course, since the data converter has a potential 10 bit deadzone, this data will only make sense if we start at the 10th code, as indicated in the datasheet.

A. Set up a New Sheet in the Vector Editor

1. Create a New Sheet and name it "AllCodes"
2. Add the digital pins to the new sheet
3. Copy the code to serially load 1 piece of data into the DAC onto the new sheet
4. Save it

B. Set up UserInit

1. Copy and paste all of the digital levels and timing commands from the previous test. Rename the digital vector label to "AllCodes."

C. Create the Linearity Function

1. Create a new function called Linearity.
2. Set up the Datasheet file for the INL and DNL specifications.
3. Set up the analog resources.
4. Set up an array of integers. This will eventually be a very large array (10 to 4095), so do this algorithmically using a loop. To start with, just set it up with 10 elements in the array. This will provide a quick test to see if everything is working without waiting forever for the code to run. Start with code 10 and increment to 20. Please use *#define codes 10* to start with. If you set up all of your loops and math to be based on "codes", then if you change the number of codes, everything should work.
5. To load the data serially without having a million entries in the digital vector editor, we will overwrite the data-bits in the vector editor.
6. The Array Type (INT64) requires us to convert the integer array into the necessary format so that the proper function can be used. Each integer must be represented by an array of 1's and 0's that maps to the integer. To do this conversion, you must mask off the LSB from the integer and save it into the INT64 array (done with a bitwise AND with 0x1, in C by:) *Value[bit] = integer & 0x1;* and then shift the integer value to get the next bit and repeat the masking process for all bits (done in C by:
   integer = integer >> 1; Value[bit+1] = integer;
   Obviously, this must be done in a loop.
7. Once the INT64 array has been properly loaded with the integer value, you can overwrite the data bits in the vector editor. Unfortunately, this function does not drive both sites. Therefore, you must use this command twice, once for site 1 (set site=0) and once for site 2 (set set=1).
8. Now, you may use the run the clocks and the digital vector editor. This time, instead of saving all of the data in a large array and sorting through it (which we may run out of memory), we will simply run the digital vector editor to load the data and then perform a DC measurement. We could have done this on the last as well, but then you would not have learned how to drive the analog and digital clocks simultaneously!
9. Get the DC results from the QMS.
10. Now, to perform this function on multiple codes, you must put this in a loop for all code values. But, you need to save each result in a large data array. Create a 2-d array for storing the final results:
    double VoDataArray [2] [codes];
    This will contain the data for all codes for 2 sites. When you get your results from the QMS, store them into the correct location in the VoDataArray.

D. Calculate INL and DNL from the Data

1. Now that you have the measurements, you can produce the math to calculate INL. Be sure to use the *#define codes* in all of your calculations.
   a. Calculate the Gain and Vos using the endpoint calculation for each site.
   b. Calculate each point for an ideal line based upon the calculated Gain and Vos. This will require yet another 2-d array.
   c. Find the worst case INL deviation from the ideal line. Be sure to normalize it with an ideal VLSB. Note: when comparing the current measurement with the previous worst case INL,

be sure to use absolute values on both cases so that it will catch whether the worst case is over the ideal line or under the ideal line. But datalog with the positive or negative, since this gives information to the product engineer.

2. Using the measured data array, find the worst case DNL. Be sure to use the absolute values for this comparison too, but datalog with the positive or negative.
3. Datalog the results.
4. Turn off the equipment.

E. Validate your Program

1. Make sure that your program compiles and runs without alarms on your laptop.
2. Backup your program and move it over to the ATE and verify your program works on the first 10 codes. It is not enough to verify that the INL and DNL pass. You MUST set a break point in your program and look at your arrays. Given that the VLSB=10mV, do the results make sense on both sites?
   If yes, then does the calculated gain and offset voltage of your shortened array make sense?
   If yes, then does the calculated INL and DNL make sense?
   If yes, then run the code without a break point multiple times. Does it appear to be relatively repeatable? If not, then you may need to increase you wait time or number of samples. (Note: if the absolute value of the number remains fairly constant, it's repeatable. A negative shift in this case merely means that a different code is close to the same value but in the opposite direction. A little noise makes one code win over the other.)
   Please address each of these questions (e.g. how you knew it made sense) in your lab memo.
3. Verify that the program is overwriting the data bits correctly by using the Mixed-Signal Viewer. Look at the digital pins and the output measurements made by the QMS.
   Do the digital pin values make sense?
   Are your analog values changing too much, or do they seem pretty stable? You may want to look at the points rather than the graph to make that determination.
   Capture the image for your lab memo and answer the questions above.
4. If everything seems to be working, increase the number of codes in your integer array to test all codes. If you have done everything relative to the #define codes, this should simply be a matter of changing the number for #define codes.
5. Verify that the datalog remained close to the previous measurements (though it may not match exactly). Repeat a few tests and save the datalogs for your final memo.

F. Write up Your Results

Turn in your final code with figures captured and explanations indicating why you believe the test is working accurately. Discuss any challenges may have faced.