Lab 3 part II – Program through ETS and Toggle IO Test

In this lab, we will complete the toggle IO test by extending the test program from part I. We will perform a toggle I/O test on the pic16f883 using the program you wrote in Lab 2. We will need to write this program to the program memory of the device through the test application. Review the PIC16F88X Memory Programming Specification document to understand the commands that must be passed to the device serially through the ICSPCLK (RB6) and ICSPDAT (RB7) pins.

For part II, after writing to the CONFIG registers, we will reset the device and re-enter Program/Verify mode so that we can write the test application to the flash memory beginning at address 0x00. You will add a new test function to your project to perform the toggle IO test. You may add additional helper functions to write to the program memory of the pic16f883.

In part I, we wrote to two 14-bit registers using digital vectors. For part II, we want to write all of the opcodes of the application from Lab 2 to the flash memory. It would be impractical to create a new sheet in the vector editor for every different opcode that needed to be written, so instead we will have one generic *Load Data for Program Memory* command vector, and we will overwrite the payload of that vector for each opcode that needs to be written. To do this, we will need the *dpinvecblockwrite* function, as described below:

dpinvecblockwrite (name, label, type, offset, count, array, site);

name:	pin or group name
label:	this is the label in the vector editor
type:	this is the data type. It may be DPIN_DRVDATA or DPIN_DRVALL. The difference between these two fields is that DATA forces the DPU to drive only a 1 or 0. ALL drives tri-state or invalid conditions as well. However, to get this much data, it is represented by a 2-bit code, which is more difficult to parse. We will use DPIN_DRVDATA.
offset:	this is the offset from the vector editor label needed to overwrite *JUST* the data bits. The control bits are still used in the same way, so they do not need to be overwritten (i.e. the offset in steps to get to the first bit of the 14-bit data value).
count:	this is the number of steps that you will overwrite in the digital vector editor.
array:	this is a pointer to an array of type INT64 (to be declared previously: INT64 name[size]) which will have the data to overwrite for each step – size of array should be *count*
site:	this lists which site you want to pull data from (0 for site 1, since we only have 1 site)

Create Toggle IO function

1. **Add a Load Data command using the Vector Editor**
   Create new sheet or modify one of the configuration load sheets in the vector editor to serve as a generic *Load Data for Program Memory* command. The 14-bit payload of this sheet can be set to all 1's, since this is the value used for resetting data latches in Program/Verify mode. We will overwrite this payload for each opcode to be written.
2. **Create the Toggle IO Function**
   a) Create a function called ToggleIO using the Wizard.
   b) Set up the Datasheet editor for three values. It should first log the checksum value of the program memory. This will be used to validate that the application was written correctly to program memory. It should also log verification measurements of the output pin RA7 (there should be at least two – one for testing logic high on input RB0 and one for testing logic low on RB0).
   c) Re-enter programming mode, after resetting the device following setting the configuration registers. There are three methods of putting the device into Programming/Verify mode using the MCLR, Vdd, and PGM pins. For high-voltage programming modes, Vpp (MCLR) must be raised to 10-12V, followed by V_{DD} raised to 5V. Keep PGM (RB3) at 0V. (See programming spec document for details and timing requirements)
   d) Set up an array of INT64 values which is the opcodes of your program. This will eventually be written to the digital vectors for the program load commands.
   e) Now, you may use the mcurun command with the analog sequence name as MCU_NOANALOG and the proper digital vector name. You'll need to load and write each opcode, then increment the PC once and repeat for the next opcode. Use *FOUR-WORD PROGRAMMING* method to expedite programming.
   f) A loop may be used to copy each opcode in the INT64 array into the load program vector and write each word into program memory.
   g) You may find it best to create a helper function (without the use of the Wizard) to handle programming the pic. Call the programming function within the ToggleIO function to write words to program memory.
   h) After writing the application to flash memory, reset the device, re-enter Program/Verify mode and read back the flash memory for all valid opcode locations. Store the sum of all opcodes read as a checksum.
   i) After the entire application is written to memory, restart the pic to begin program execution.
   j) Set the value on input RB0 and log the voltage measured on the output pin of your application. You may use digital vector sequences again to capture the toggle results, or simply use voltage measurements.

Write-up your results

Datalog the checksum of the opcodes which you read from flash memory, and the output for both logic high and low tests of the ToggleIO test. Calculate what checksum value should be based on the opcodes in your application, and use this value in the datasheet editor for specifying the valid range. Describe how you made use of the digital vector sequences. Record your test time. Turn in your final code and datalogs. Discuss your test results.

Due: Wednesday, April 13, 2016