Lab 3 – Program through ETS and Toggle IO Test
Part I – Write CONFIG Registers

In this lab, 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 I, we will attempt to successfully enter Program/Verify mode on the PIC, and to write to the two CONFIG registers. You will add a new test function to your continuity project to write to the CONFIG registers and read back the values to verify they are correct. You may add additional helper functions to write to the program memory of the pic16f883.

Create WriteConfig function

1. **Clock Set-Up in UserInit**

   In order to use the master clock provided in the ATE, we need to set up the main clock to provide the period for the DPUs in the UserInit function. You will need to set the clock period to meet the timing requirements needed for the digital sequence vectors for programming, including the setup and hold times for the data signal, and the delays between the command and payloads (see the programming spec for timing details). Refer to the appendix of this handout for more details on some of the functions that will be needed for setting up the clock and dpin units.

   Master Clock Set-Up

   mcuset: set the main clock to an appropriate value to be divided down further by *dpinperiod*

   mcuclksource: connect the main clock to CLKA so that it can be used by the digital system

   Digital Clock Set-Up

   dpinperiod: use this to divide the clock down to the desired period for the digital vectors

   dpinlevel: set all pins in the digital vector file to output voltage 5 and 0 volts and a receive voltage thresholds of 4 and 1V.

   dpintimeset: multiple timesets are necessary, one for each vector sequence. (i) the clock (ICSPCLK) should have the clock format (KT), so that each digital clock will have a rising and falling edge. You can set the edges at 0 and 50 percent of the full cycle. (ii) the data signal (ICSPDAT) can be No Format (NZ) or have a non-return to zero format (NRZ) with the first edge at the same time as the rising edge of the clock (the *PIC* will latch the data on the falling edge of the clock).

2. **Load the Digital Inputs and Outputs using the Vector Editor**

   Create new sheets in the vector editor for each of the commands you will need to use in Programming/Verify mode of the *PIC*. Add the RB6 and RB7 pins to these sequences, and rename them to identify which is ICSPCLK and ICSPDAT. Depending on how you plan to accomplish the necessary timing requirements you may make each of the commands 6 bits of sequence, and have separate 16 bit sequences for the payloads. Or you may make commands without a payload 6 bits in length, and commands with a payload 22 bits in length. Keep in mind that both the command values and the payloads are LSB-first.

3. **Create the WriteConfig Function**

   a) Create a function called WriteConfig using the Wizard.

   b) Set up the Datasheet editor to store the value of both CONFIG words. We will read back the registers after writing to them to verify the write succeeded.

   c) Set up the analog resources for programming mode. There are three methods of putting the device into Programming/Verify mode using the MCLR, V_{DD}, and PGM pins. For high-voltage programming modes, Vpp (MCLR) must be raised to 10-12V. (See programming spec document for details and timing requirements)

   d) Define INT64 values for the CONFIG registers. 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. Execute the necessary instructions to enter configuration memory, go to address 0x2007, then write to and read back from the registers at 0x2007 and 0x2008.
f) You may find it best to create a helper function (without the use of the Wizard) to handle programming the *PIC*. You can call the programming function within the WriteConfig function to write words to program memory.
g) To load each opcode into program memory without having dozens of entries in the digital vector editor, we will use a new command, *dpinvecblockwrite*, to overwrite the data-bits in the vector editor. This will be particularly useful for Part II of the lab.

dpinvecblockwrite (Name, Label, Type, Offset, Count, Array Name, 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.
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 be filled with the data
site: this lists which site you want to pull data from (0 for site 1, since we only have 1 site)

h) To verify the writes to register, we will perform a program memory read, and extract the received data using *dpingetcapturedata*.

dpingetcapturedata(name, type, count, pArray, site);

name: pin or group name
type: the type of digital status data to retrieve. Valid values are: DPIN_CAPTDATA or DPIN_CAPTALL. We will use DPIN_ CAPTDATA.
count: this is the number of captured steps to read. -1 indicates all steps.
pArray: this is a pointer to an array of type INT64 (to be declared previously: INT64 name[size]) which will be filled with the data
site: this lists which site you want to pull data from (0 for site 1, since we only have 1 site)

Write-up your results

Increment to the CONFIG registers at addresses 0x2007 and 0x2008. Write the desired CONFIG register values for each, and read back the value at each register to verify that it was written correctly. Datalog the two register values. 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. Note the time spent in WriteConfig function, can this be improved?

Due: Wednesday, March 30, 2016

Appendix: DPU Clocking

In this lab you will need to use a digital clock to set the period for the serial digital vector sequences that will be used to write the program to the program memory of the PIC. Therefore, you will need some additional dpin commands to set-up the clocking scheme.

mcuset (clock source, freq);

This function sets the main clock frequency.

source:	select which clock will be the main source: MAIN, AUXILLARY, OTHER
freq:	any value between 25MHz to 66MHz as a starting clock. The signal can be further divided down later.

mcuclksource (clock, source);

This function sets-up the multiplexor that determines whether the master clock previously set will go to sourceA or source.

clock:	output clock: connecting source to clockA, clockB, or both?
source:	main source: is the main source MAIN, AUXILLARY, 50MHz, 25MHz, or OTHER? This must match the setting in mcuset. If you used MAIN in mcuset, use MAIN here

dpinperiod (clock cycles);

This function divides the master clock down to the frequency to drive the digital pins. The DPU's do not use the mcumode command to do this. In fact, if you are only running the DPUs, many of the other mcu functions (mcuconnect, mcusequence) are not necessary. The master clock MUST be placed on clockA using the mcuset and mcuclksource commands. Then, the clock can be divided down by this function.

clock cycles:	This is an INTEGER that sets the divide down ratio,

dpintimeset (pin, digitalLabel, timesetName, timesetType, clockFormat, Edge A, Edge B, Edge C, Edge D, Site);

This function sets up the clock type and the timing information as to when information will be sent or received relative to an edge change, as seen in the waveforms of the dpintimeset document.

pin:	pin name or group; use DPU_ALL_PINS for all digital pins
digitalLabel:	label, in quotes, of the digital vector in the vector editor (sheet label)
timesetName:	one of the default timesets may be used here (such as "DefaultTimesetInOut")
timesetType:	type of timeset being programmed (use the normal setting)
clockFormat:	there are many different clock formats (see Clock Formats handout). We will use multiple clocking schemes in this lab to show you how several of them can be used.
Edge A-D:	this allows you to set when in the period you want the digital levels to change
Site:	to get all sites, use MS_ALL

dpinlevel (pin, Digital Label, Driver High, Driver Low, Receiver High, Receiver Low, Site);

This function sets up the digital input levels (driver) and the output levels (receiver) that will be considered a high or low.

pin:	pin name or group; use DPU_ALL_PINS for all digital pins
Digital Label:	Label, in quotes, of the digital vector in the vector editor
High Driver:	voltage level that the input should be for a "1"
Low Driver:	voltage level that the input should be for a "0"
High Receiver:	output voltage level that is considered a "1"
Low Receiver:	output voltage level that is considered a "0"
Site:	to get all sites, use MS_ALL