LAB Assignment #3 for Hardware/Software Codesign with FPGAs

Assigned: Mon., Oct 7, 2013
Due: Mon., Oct 14, 2013

Description: Add a Sample Analysis Engine to Lab 2 as described below.

The LFSR from Lab 2 will generate a pseudo-random sequence of numbers. These numbers can be used as an address, as indicated in Lab 2 with the labeling of Row1, Col1, etc. YOU WILL USE ONLY THE Row1 and Col1 values from Lab 2 in this lab.

The Row1/Col1 addresses will be used as an address into an array of values. The array of values (which I will supply) need to be transferred across the serial port (using an terminal emulation program such as minicom or teraterm) and stored into a multidimensional array.

The C program that you developed in Lab 2 needs to be expanded to 'lookup' values from this array for each LFSR-generated Row/Col address. The array will include a sequence of 16 (x,y,z) values, where each x and y are repeated 16 times and represent the Row1 and Col1 address, and the z values are the 16 samples. The samples will be 8-bit values in the range from 0 to 128 and will vary across the set of 16 samples for each of the row/col address.

Your C code needs to write these samples into **slv_reg6**, one at a time. After writing each sample, you will assert the control signals (shown below) in **slv_reg7** to start the sample analysis engine, and then wait for it to finish (wait for the 'ready' bit to be set in **slv_reg8**). The average 'scaled' sample should be written into **slv_reg9** after all 16 samples have been processed. You should assert 'first_sam' on the first sample so the VHDL code knows to clear the 'sum' register (see below). You should assert 'last_sam' on the 16th sample so the VHDL code can compute the average and store the result in slv_reg9.

The entity definitions for the sample analysis engine is given as follows:

```
entity SamAnalysis is
  generic(
    VDC_NUMBITS_NB: integer := 8;
    MAX_NUM_SAMS_NB: integer := 7;
    SAM_RSHIFT_AMT_PRECISION_NB: integer := 3;
    SAM_RESULT_NB: integer := 11
    );
  port(
    Clk: in  std_logic;
    RESET: in std_logic;
    start: in std_logic;
    ready: out std_logic;
    first_sam: in std_logic;
    last_sam: in std_logic;
    log_n_num_sams: in std_logic_vector(MAX_NUM_SAMS_NB-1 downto 0);
    incoming_val: in std_logic_vector(VDC_NUMBITS_NB-1 downto 0);
    scaled_ave_volt: out std_logic_vector(SAM_RESULT_NB-1 downto 0)
    );
```

end SamAnalysis;

Inputs: ‘start’ is the start signal that you assert to start the sample analysis engine (connected to **slv_reg7**). ‘first_sam’ and ‘last_sam’ should be asserted in **slv_reg7** when the 1st and 16th samples, respectively, are written by your C code. ‘log_n_num_sams’ should be set to set 4 for the case where you want to average all 16 samples. ‘incoming_val’ is connected to **slv_reg6**, i.e., the register where you write each of the samples.

Outputs: ‘ready’ will asserted when the sample analysis engine finishes processing a sample (connected to **slv_reg8**). ‘scaled_ave_volt’ is the scaled final average value of the samples, which should be valid after you process the 16th sample with ‘last_sam’ asserted.

In order for you to obtain more precision than an integer data type will provide, you need to implement ‘fixed point’ arithmetic. Assuming we don’t want to deal with floating point data types in VHDL, we can realize fixed point arithmetic by creating a sum of the ‘incoming_val’ integer values and shifting the result. Bear in mind that the sum register needs to be wide enough to handle the sum. For example, if you intend to add 16 8-bit values, then the sum register needs to be 16-bits wide.

The constant SAM_RSHIFT_AMT_PRECISION_NB (set to 3 above) indicates the number of ‘bits’ of precision. This constant in combination with ‘log_n_num_sams’ determines how many bits you need to left shift or right shift the sum register by to compute the average. For example, if you are computing the average of 16 samples, you will need to right shift the sum register by (log_n_num_sams - SAM_RSHIFT_AMT_PERCISION_NB), which is equal to 1 using the constants above.

You need to write the shift direction and amount generically, allowing for other values to be assigned to log_n_num_sams and SAM_RSHIFT_AMT_PRECISION_NB. For example, when log_n_num_sams is less than SAM_RSHIFT_AMT_PRECISION_NB, you need to left shift the sum register to compute the average.

Store the ‘scaled_ave_volt’ in **slv_reg9**. Your demo should print out the sequence of ‘scaled_ave_volt’ values for the first 10 LFSR values that are generated. Use the ‘_exhaustive’ setting and default seed hardcoded into the LFSR from Lab 2. Demo this using all 16 samples, and then again using only the first 4 samples (of the 16).