CS/ECE 561: Hardware/Software Design of Embedded Systems
Fall 2016

Homework 1: Modeling with SystemC

Assigned: 8 September, 2016
Due: 20 September, 2016

Instructions:

- Please submit your solutions via canvas. Submissions should include source code files in a single zip file, with separate folders designated for every problem. The folders can contain a word or pdf file if necessary.
- Some questions might not have a clearly correct or wrong answer. In such cases, grading is based on your arguments and reasoning for arriving at a solution.

Q1. (15 points) The goal of this problem is to make you familiar with the SystemC environment, in particular compilation and simulation of SystemC code, using the simple FIFO example included with the SystemC installation. Follow the instructions in the file instructionsSystemC.pdf to set up your environment and run the simple fifo example. Inspect the sources of the example and the included Makefile to understand SystemC compilation and simulation process, experiment with the Makefile usage and start modifying the example to experiment with different features of the SystemC language:

(a) Briefly explain (in 2-3 sentences) the functionality and structure of the example. Report the output from simulation of the example. **(5 points)**

(b) Modify the example to replace the custom `fifo` channel with a corresponding `sc_fifo<char>` channel from the standard SystemC channel library. Simulate the code to verify correctness and submit the modified source code. **(10 points)**

Q2. (50 points) In this question, we will construct a simple 14-tap finite impulse response (FIR) filter with SystemC. This FIR filter has a memory module to provide data as input for FIR nodes. On each positive clock edge, a FIR x. Within the same cycle, each FIR node writes received data into its data output port connected with next FIR node so as to right shift data among FIR nodes. The task of the *accumulator* module is to sum up each cycle's multiplication results of FIR nodes and write the result into memory (see Figure 1). Ultimately, the result will be:

$$y[n] = x[n]*b_0 + x[n-1]*b_1 + \ldots x[n-i]*b_i + \ldots x[n-N]*b_N$$

where N = 14 is the number of taps, b_i is the filter coefficient known as tap weights, *n* represents the cycle when a input or output data is generated, *x[n]* is the input data, *y[n]* is the output data. At last, content in memory will be overwritten by results of FIR filter.

In the **fir** folder, all the header and implementation files are provided, together with a main file to initialize all processes and connect them together with SystemC signals. However some lines in the main and implementation (.cc) files are missing (indicated as [missing] followed by description in the code). Fill in those missing lines of code and complete this example to produce the correct output (see content of **result.log** for the reference output).

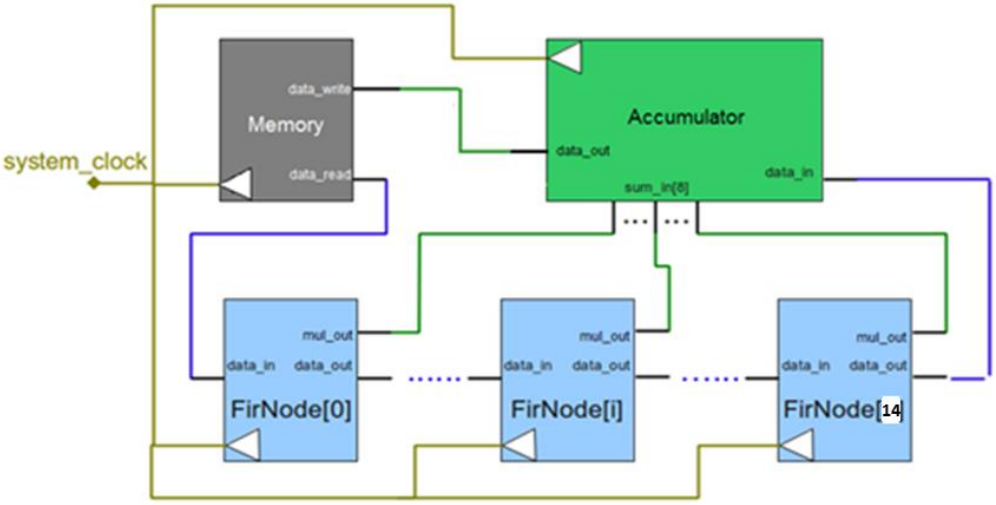

Figure 1. Diagram of 14-tap FIR filter

Q3. (50 points) In this question, you will extend an example available in the SystemC distribution (in the examples folder). The example is for a multi cast packet switch module.

In the **pkt_switch** folder, a 4×4 multi-cast switch is modeled. Compile and run the given program to see the output. Read the source code and README file (in examples/sysc./pkt_switch folder) to understand its structure. Then extend this example into an 8×8 multi-cast switch. Try to get the correct output. Make sure that all packets reach their destinations successfully.

Q4. (50 points) In this example you will work with a three-stage pipeline as shown below

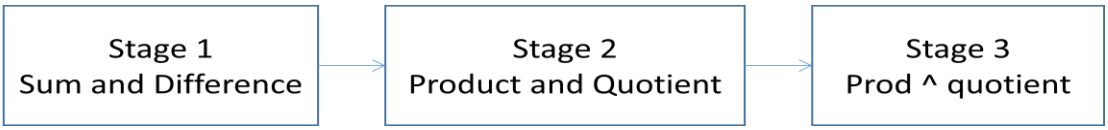

Figure 2. Schematic that depicts the pipeline

Stage 1 accepts two numbers as input and computes the sum and difference of those numbers. Stage 2 computes the product and quotient of the sum and difference. Stage 3 computes the product raised to the power of the quotient.

Read the source code and README file (in examples/sysc./pipe folder) to understand the pipeline structure. Once you compile and run this program you can see the system printing out a set of results. It can be noted that the first few results are printed as 0.000 which is the wrong output.

Your task is to make changes in the program to ensure that all the results from the very first to the last are correct and are in accordance with the given input.

Hint :
1. Try to write a simple C/C++ version of the system to know the expected output.
2. Start thinking on the lines that all the processes in a system need not be triggered in the way you expect it to be triggered (the sequence of execution of the various processes in a system may not be as you would expect/want)

Q5. (50 bonus points) In this problem, you will implement a Static Random Access Memory (SRAM) module which can be implemented with a storage cell structure that does not require refresh. Therefore, it operates faster than Dynamic Random Access Memory, and used as fast-cache memory in embedded computing systems. As a starting point of our implementation, we will look at a simple SRAM cell, and proceed to larger memory blocks with unidirectional data ports, and finally implement a memory block directly.

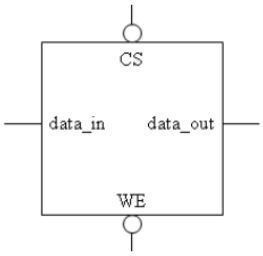

Figure 2. SRAM cell: block diagram

The basic SRAM cell represented by the block diagram symbol in Figure 2 has active-low inputs for cell select (CS_b) and write enable (WE_b). The cell select signal is generated by a decoder that selects among multiple cells in the same system. Note the absence of a clock signal. Storage registers and register files are implemented by flip-flops, but the storage devices of RAMs including SRAM and DRAM are sometimes implemented as transparent latches, which support asynchronous storage and retrieval of data and minimize the time that a RAM requires service from a shared bus.

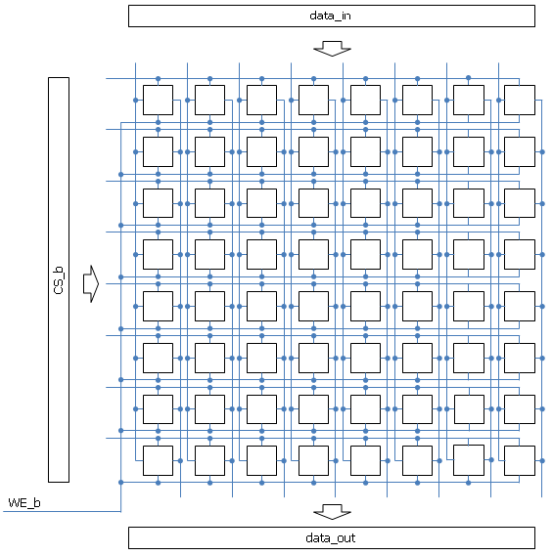

Figure 3. Cell array of 8 x 8 SRAM

An SRAM memory (e.g., L1 cache) is constructed with many SRAM cells. Also, it is called an SRAM cell array. Figure 3 is an 8×8 SRAM cell array which has 8 words and 1 word is 8 bits. To write 1 word (8 bits) into this cell array, only one CS_b signal must be enabled. The signal data_in and data_out have 8 bits width.

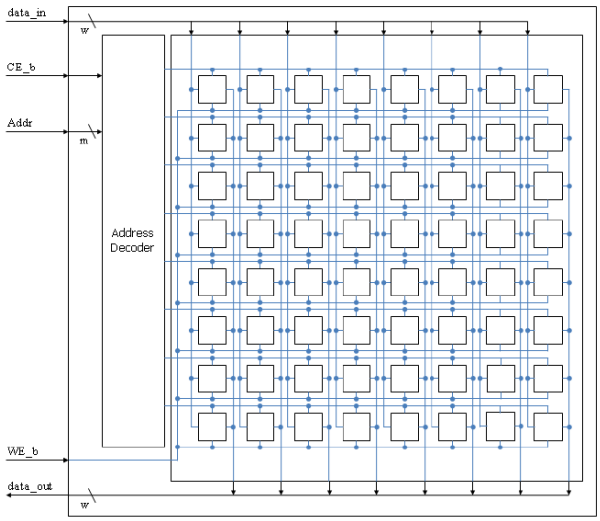

Figure 4. The entire structure of the 8×8 SRAM

To generate only one row of cell select signal (CS_b), there exists an address decoder in the SRAM structure. Figure 4 shows the entire structure of the 8×8 SRAM. The width of data_in and data_out is w bits, and w is 8 since 1 word is 8 bits in the structure of the 8×8 SRAM. The number of pins for addressing is m, and m is 3 in case of an 8×8 SRAM. In addition, CE_b is added in the entire structure to control the entire circuit of the SRAM. When CE_b is active, all addressing pins become disabled (i.e., CS_b is disabled for all memory cells).

Now, we will implement 32K (16,384 Bytes) × 8 SRAM which has 15 bits of address pins and 8 bits of data pins. To generate CS_b for all memory cells, the 15 bit address signals should be decoded to control all of the 32K cells. The size of logic to decode will increase if the size of address pin increases. To minimize the size of control logic in address decoder, two-level addressing algorithm can be used. Figure 5 shows an example of two-level addressing in case of one 8 × 1 SRAM.

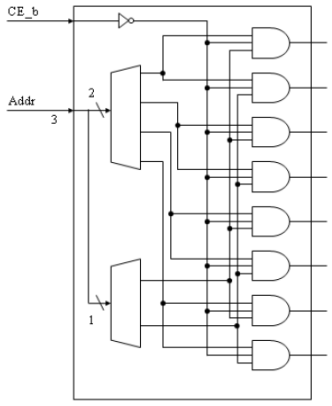

Figure 5. The circuit of address decoder for 8 × 1 SRAM bank

Your job is to design this SRAM, by adding details to the included RAM.cpp file template. Once the design is completed it must be tested. The functionality of the design module can be tested by applying a testbench and checking the results. Use the included test_RAM.cpp testbench file to test your design. The testbench module instantiates the design module and directly drive the signals in the design module. The testbench should be complied

along with the design module. At the end of compilation the simulation results will be displayed. Figure 6 shows the block diagram of the entire system with the SRAM and testbench.

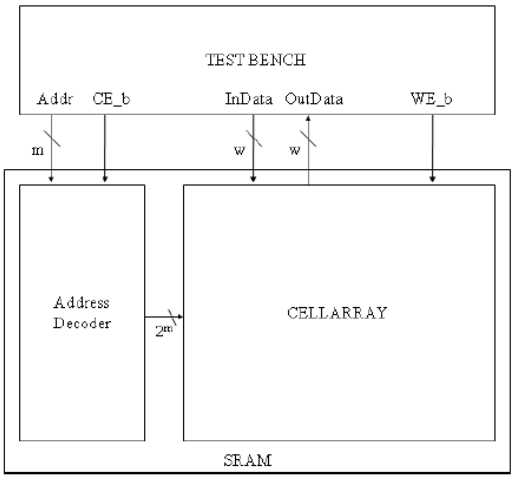

Figure 6. Block diagram for $2^m \times w$ SRAM and testbench

You may want to see waveform of the result. After your simulation, you will be able to see the file *.vcd as an output file. Then, use 'gtkwave' or any other VCD file viewer in Linux to see the waveform. This can be used as a debugging tool for your design.

In summary, your SRAM design should have a size of 32K words (1 Word : 8 bits), with Ports : bCE, bWE, Addr, InData, OutData. Submit your RAM.cpp file with detailed comments. Do not modify the test-bench; consequently you do not need to turn in the test bench file. Submit a brief report showing the exact output of executing your design with the testbench.