ECEN 468 Advanced Logic Design
Department of Electrical and Computer Engineering
Texas A&M University

(Lab exercise created by Jaeyeon Won and Jiang Hu)

Lab 1

Introduction to SystemC and Simulator

Purpose:

SystemC is a system-level modeling language and Vista is a platform for architecture design. We will design a simple application like a memory device with SystemC, and will use Vista as a tool for simulation and verification.

Preparation:

1. General Design Flow

The design will begin in this lab with the description of a memory circuit for entire labs later using SystemC. SystemC is a set of C++ classes and macros which provide an event-driven simulation kernel in C++. These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax. SystemC processes can communicate in a simulated real-time environment, using signals of all the datatypes offered by C++, some additional ones offered by the SystemC library, as well as user defined. In certain respects, SystemC deliberately mimics the hardware description languages VHDL and Verilog, but is more aptly described as a system-level modeling language.

In the following labs we will design other parts for further goal. For every lab, we will design a separate module of our whole system, and simulate it with testbench. Finally, we simulate entire module which is designed using SystemC to verify your designs.

2. Brief introduction to SRAM as a memory element.

We will implement a Static Random Access Memory (SRAM) 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 a computer. As a starting point of our

implementation, we will look into a simple SRAM cell, and proceed to larger memory blocks with unidirectional data ports, and finally implement a memory block directly.

The basic SRAM cell represented by the block diagram symbol in Figure 1 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 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.

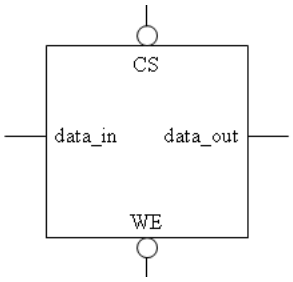


Figure 1 SRAM cell: block diagram symbol

A certain SRAM is constructed with many SRAM cells. Also, it is called SRAM cell array. Figure 2 is an 8x8 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.

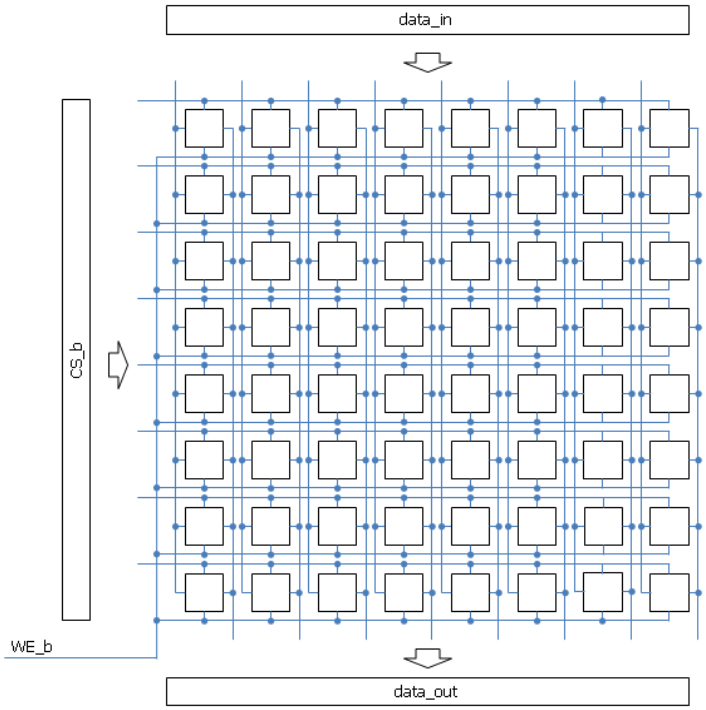


Figure 2 Cell array of 8 x 8 SRAM

To generate only one row of cell select signal (CS_b), there exists address decoder in the SRAM structure. Figure 3 shows entire structure of 8 x 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 8 x 8 SRAM. The number of pins for addressing is m, and m is 3 in case of 8 x 8 SRAM. In addition, CE_b is added in the entire structure to control entire circuit of SRAM. When CE_b is active, all addressing pins become disable which are CS_b for all memory cells.

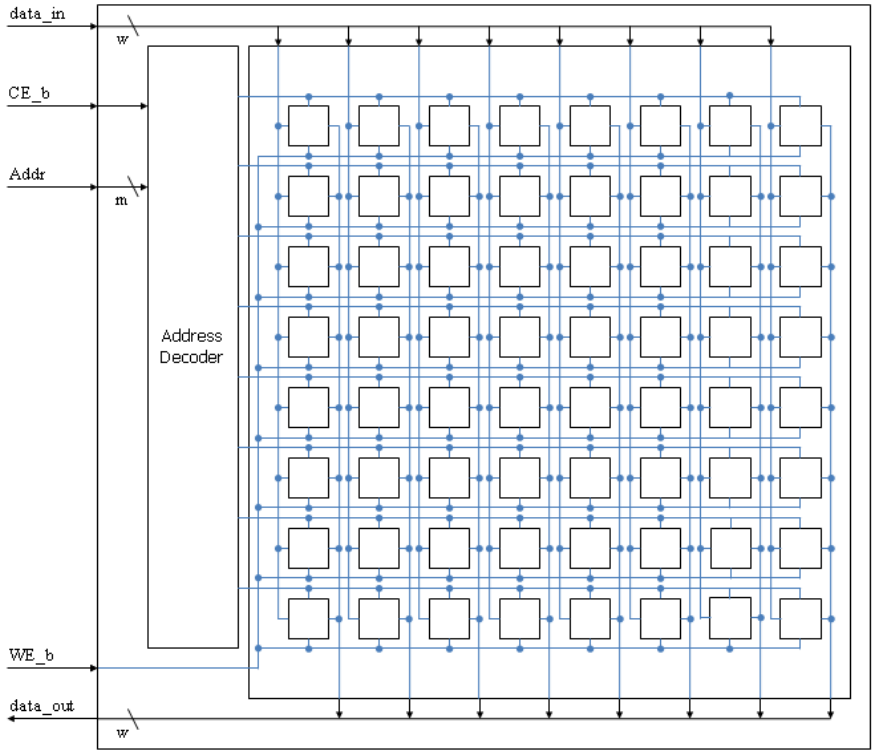
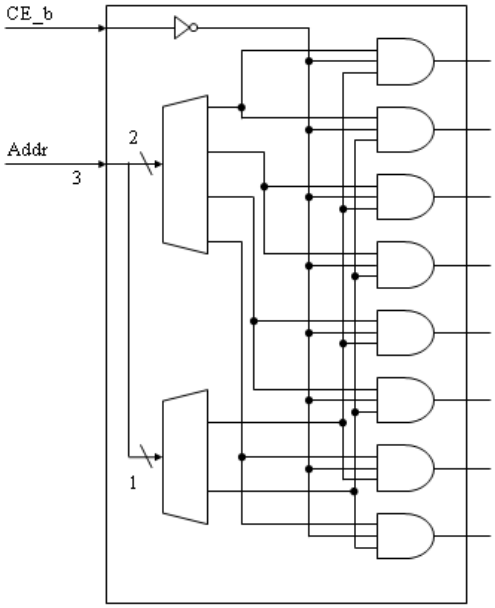


Figure 3 The entire structure of 8 x 8 SRAM



Now, we will implement 256K(262.144Bytes) x 8 SRAM which has 18 bits address pins and 8 bits data pins. To generate CS_b for all memory cells, 18 bits address signals should be decoded to control all of 256K 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 4 shows an example of two-level addressing in case of one 8 x 1 SRAM.

Figure 4 The circuit of address decoder for 8 x 1 SRAM bank

3. Setup Environment

Vista is a native ESL (Electronic System Level) platform for architecture design, verification, analysis and virtual prototyping that incorporates an advanced design, verification and analysis tool-set targeted for high-level TLM (transaction-level modeling) hardware platforms.

Open *".cshrc"* file(it is hidden by default) in your home directory. You can use any TXT editor such as vi, emacs, pico and etc. If you have two lines below in your .cshrc file, please delete them.

Delete *"source /softwares/setup/mentor/setup.vista312.linux"* and save it.
Delete *"source /softwares/setup/synopsys/setup.synopsys.tcsh"* and save it.

Close this terminal if you are using terminal.

Open one terminal, type the command below. Please remember you should open vista only in this terminal. HOME]$ source /softwares/setup/mentor/setup.vista312.linux

Open new terminal, type the command below. Please remember you should open wv only in this terminal. HOME]$ source /softwares/setup/synopsys/setup.synopsys.tcsh

4. Functional Simulation

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. The testbench module can instantiate the design module and directly drive the signals in the design module. The testbench can be complied along with the design module. At the end of compilation the simulation results will be displayed.

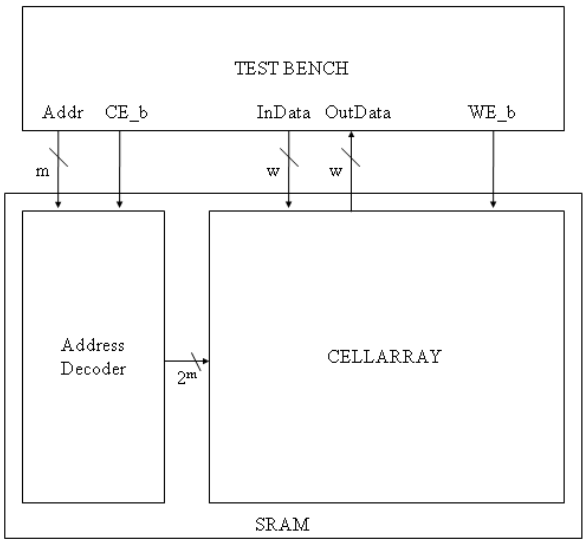


Figure 5 The block diagram for 2^m x w SRAM and testbench

a. Make working folder for this lab.

HOME]$ mkdir ECEN468/Lab1/SRC (make work folder)
HOME]$ cd ECEN468/Lab1/SRC (move to work folder)
Move the files RAM.cpp and test_RAM.cpp into the working folder

Please check the files decompressed. It should contain the files below.
RAM.cpp, test_RAM.cpp
ECEN468/Lab1/SRC]$ vista & (Start Vista)

Now, we will implement 256K x 8 RAM which has 18 bits address pins and 8 bits data pins using behavioral description as shown in Figure 5. This allows us to design complex modules much easily comparing structural modeling.
Use the files (RAM.cpp and test_RAM.cpp) given in the lecture website.

b. Make a new project

Project → New Project → Type RAM.v2p
Move to File tap → Add files (all files above)

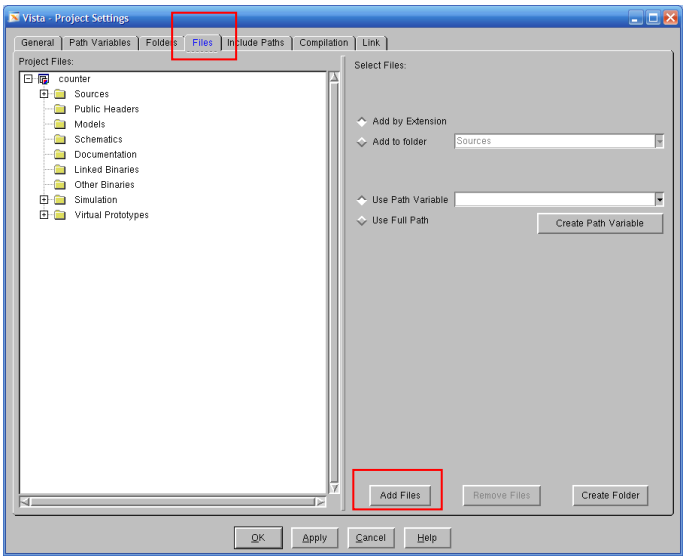


Figure 6 Project Window

c. Design a RAM in files above with referring to the example codes given in the end of this manual. Once you finish your design, you can try compile to check syntax and behaviors by using following procedure. Right-click on RAM in upper left-hand window → Build

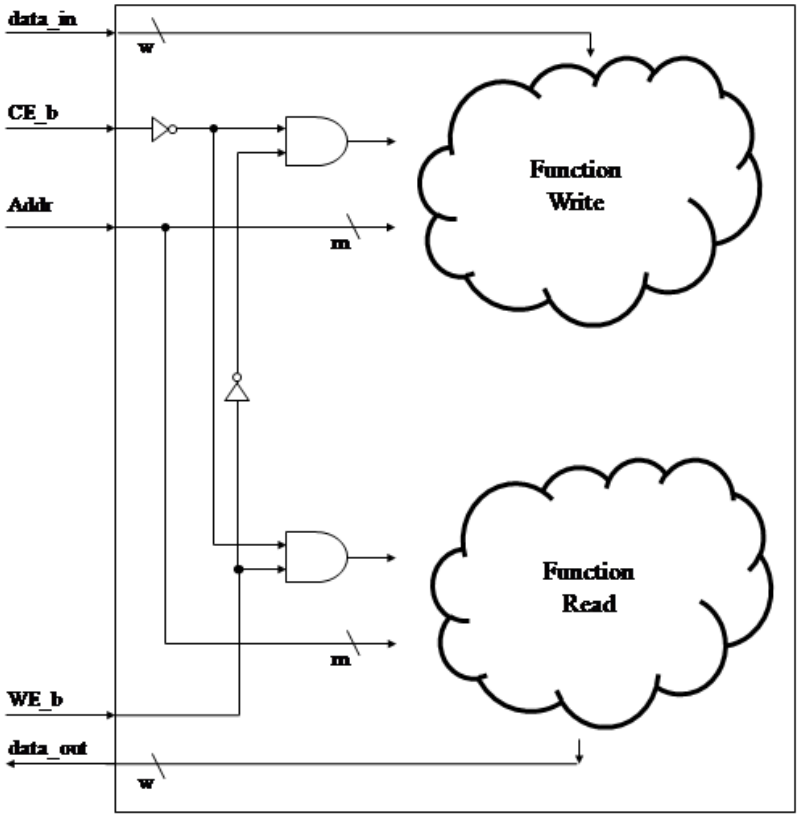


Figure 7 Behavioral diagram of read and write functions

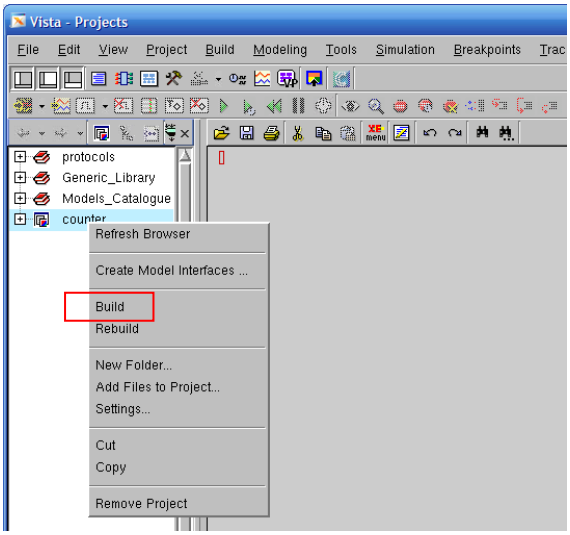


Figure 8 Screen to build project

d. In upper left-hand window, expand hierarchy and select. And, click Simulation → test_RAM.cpp. Then, choose Menu → Simulation → Simulate. Ensure that target is filled in with the name "RAM(test_RAM.cpp)" and deselect "Stop After Elaboration". This will run simulation to the end. And, click 'OK'.

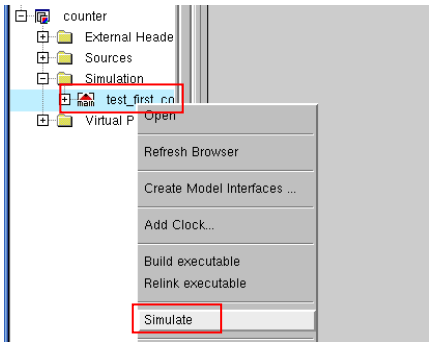


Figure 9 Screen to simulate project

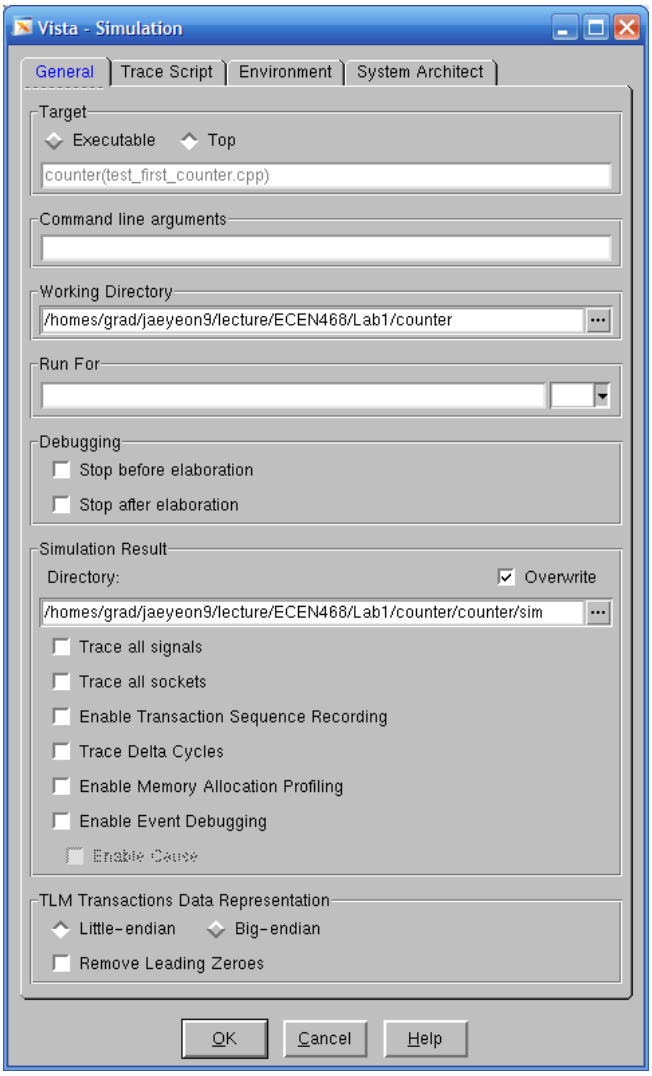


Figure 10 Simulation window

e. After simulation, click on simulation output frame at bottom of window and choose

Ctrl-X Ctrl-S and enter sim.out as filename. Hit enter to save.

f. You may want to see waveform of the result. After your simulation, you will be able to see the file *.vcd as a output file. Then, use ‘WaveView’ as a tool to see the waveform. Go to your terminal prompt and use the command below.

ECEN468/Lab1/SRC]$ wv &

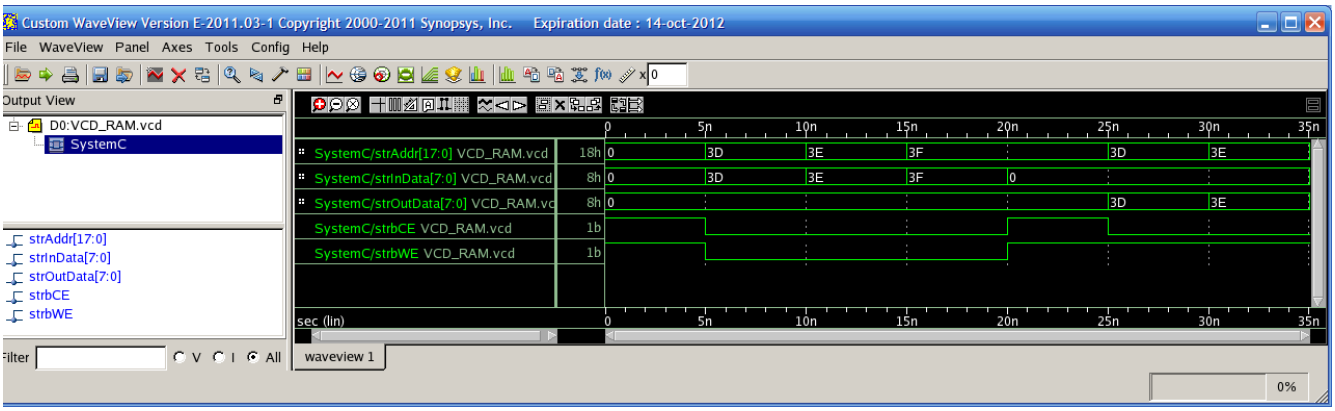


Figure 11 WaveView window

g. You can check if the simulation result is correct through waveform of your simulation. If the result is not correct comparing your estimation, you will need to modify any incorrect files and save them, and then repeat process c.

h. Please show your result to TA within your regular lab hour to get credit for lab evaluation.

Requirements:

1. General requirements.
   a. It should control SRAM correctly.
      - Size : 256K words (1 Word : 8 bits)
      - Ports : bCE, bWE, Addr, InData, OutData
   b. Detailed comments.
   c. Please do not change the names of the signals given and the other functions not mentioned in the report. (You will lose some points if you change any names of the input or output signals.)
   d. You may modify the test-bench given to get the results only if it is resaonable. Please write detailed comments if you have modified.
   e. Late penalty : Certain portion of total score will be deducted on each subsequent weekday after due date.

2. Submit hardcopy of your report to TA. The report should include contents below.
   a. RAM.cpp (5 points)
   b. sim.out (test message output) with analysis. (15 points)
      - The message should show the data and the address transferred using ‘cout’.
   c. Captured waveform with analysis. (10 points)
   d. Q1: What are differences between asynchronous SRAM and synchronous SRAM? (5 points)

Note : Please send your source codes, testbench, results(sim.out and *.vcd) to TA’s email address. (5 points)

Example codes

```
//=============================================
// Function : 4Bit Adder
//=============================================
#include "systemc.h"

#define DATA_WIDTH  4

SC_MODULE (mAdder) {
   sc_in       <sc_uint<DATA_WIDTH> >     dInA      ;
   sc_in       <sc_uint<DATA_WIDTH> >     dInB      ;
   sc_out     <sc_uint<DATA_WIDTH+1> > dOut      ;

   // ----- Code Starts Here -----
   void function_adder () {
         dOut.write(dInA.read()+dInB.read());
   }

   // ----- Constructor for the SC_MODULE -----
   // sensitivity list
   SC_CTOR(mAdder) {
      SC_METHOD (function_adder);
           sensitive << dInA << dInB;
   }
};
```

```
#include "adder.cpp"

int sc_main (int argc, char* argv[]) {
        // Declare Input/Output Signals
        sc_signal < sc_uint<DATA_WIDTH> >       tInA;
        sc_signal < sc_uint<DATA_WIDTH> >       tInB;
        sc_signal < sc_uint<DATA_WIDTH+1> >     tOut;

        int i,j;

        // Connect the DUT(Design Under Test)
        mAdder Adder_01("SIMULATION_adder");
                Adder_01.dInA(tInA);
                Adder_01.dInB(tInB);
                Adder_01.dOut(tOut);

        // Open VCD(Value Change Dump) file
        sc_trace_file *wf = sc_create_vcd_trace_file("VCD_ADDER");

        // Dump the desired signals
         sc_trace(wf, tInA, "strInA");
         sc_trace(wf, tInB, "strInB");
         sc_trace(wf, tOut, "strOut");

        // Initialize all variables
        tInA.write(0);
        tInB.write(0);
        sc_start(5);

        // Behavior
          for(i=0; i<16; i++){
                tInA.write(i);
                for(j=0; j<16; j++){
                        tInB.write(j);
                        sc_start(2);
cout << "@" << sc_time_stamp() << ":: [" << tInA << "]+[" << tInB << "]=[" << tOut
<< "]" << endl;
                }
          }

        // Close trace file
        sc_close_vcd_trace_file(wf);

        return 0;       // Terminate simulation
}
```