ELEN 503
Hardware-Software Co-Design

Homework #2
SystemC and TLM

Prof. Hoeseok Yang
Santa Clara University

Spring Quarter 2023

Homework #2 - Overview

- In this homework, you will practice SystemC-based modeling and simulation

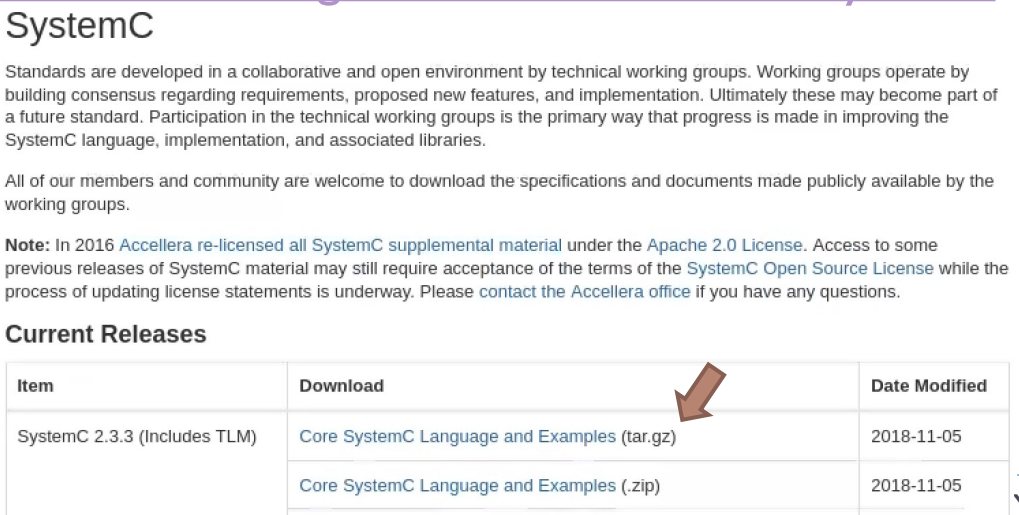
- Tasks – 10(+2) points
  - Part 1 (set up and warming up): 2 point
    - Compile and install SystemC on your ECC linux computer
    - Warming up with "sc_signal vs. sc_buffer"
  - Part 2 (modules): 2 points
    - Different triggering conditions
  - Part 3 (Functional verification - Fibonacci): 2 points
    - Untimed SystemC modeling of Fibonacci sequence generator
  - Part 4 (Transaction Level Modeling): 2 points
    - Loosely-timed TLM modeling of matmult
    - (Extra 2 points) Loosely-timed TLM modeling of matmult with MAC accelerator
  - Report: 2 points

- Submission
  - Upload your report (in pdf) and source codes (in zip) to Camino
  - Due: June 1st, 11:59pm

Part 1: Install SystemC

- Use the ECC Linux Platform
  - https://www.scu.edu/engineering/labs--research/labs/engineering-computing-center/remote-access-to-the-ecc/
  - Don't forget to secure the VPN connection when connecting from outside

- Download SystemC 2.3.3 (with TLM)
  - https://www.accellera.org/downloads/standards/systemc



SystemC

Standards are developed in a collaborative and open environment by technical working groups. Working groups operate by building consensus regarding requirements, proposed new features, and implementation. Ultimately these may become part of a future standard. Participation in the technical working groups is the primary way that progress is made in improving the SystemC language, implementation, and associated libraries.

All of our members and community are welcome to download the specifications and documents made publicly available by the working groups.

Note: In 2016 Accellera re-licensed all SystemC supplemental material under the Apache 2.0 License. Access to some previous releases of SystemC material may still require acceptance of the terms of the SystemC Open Source License while the process of updating license statements is underway. Please contact the Accellera office if you have any questions.

Current Releases

Item	Download	Date Modified
SystemC 2.3.3 (Includes TLM)	Core SystemC Language and Examples (tar.gz)	2018-11-05
	Core SystemC Language and Examples (.zip)	2018-11-05

Part 1: Install SystemC (cont'd)

1. Unzip the downloaded tarball
   - $SYSTEMC: your systemc home directory

2. Go to the directory and make a temporary directory for compilation
   a. cd $SYSTEMC
   b. mkdir objdir
   c. cd objdir

3. Designate g++ as the compiler to be used for SystemC
   - export CXX=g++

4. Generate Makefile
   - ../configure

Part 1: Install SystemC (cont'd)

5. Compile
   - gmake

6. Check if everything is okay with the compilation
   - gmake check

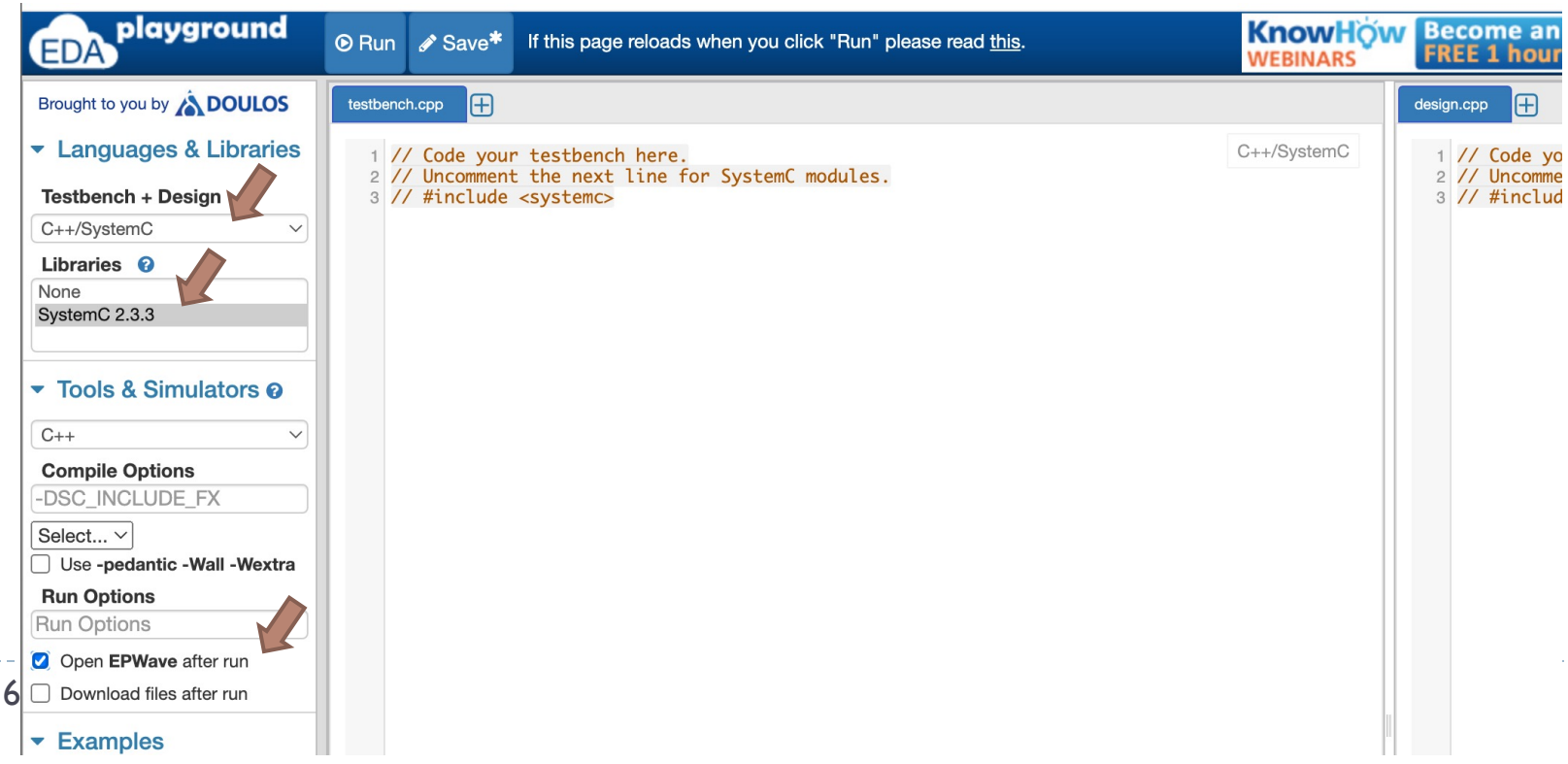
7. Install (copy compiled library)
   - gmake install

```
gmake[4]: Entering directory `/DCNFS/us
objdir/examples/sysc'
PASS: fft/fft_flpt/test.sh
PASS: fft/fft_fxpt/test.sh
PASS: fir/test.sh
PASS: fir/test_rtl.sh
PASS: pipe/test.sh
PASS: pkt_switch/test.sh
PASS: risc_cpu/test.sh
PASS: rsa/test.sh
PASS: simple_bus/test.sh
PASS: simple_fifo/test.sh
PASS: simple_perf/test.sh
PASS: 2.1/dpipe/test.sh
PASS: 2.1/forkjoin/test.sh
PASS: 2.1/reset_signal_is/test.sh
PASS: 2.1/sc_export/test.sh
PASS: 2.1/sc_report/test.sh
PASS: 2.1/scx_barrier/test.sh
PASS: 2.1/scx_mutex_w_policy/test.sh
PASS: 2.1/specialized_signals/test.sh
```

```
gmake[6]: Leaving directory `/DCNFS/users/faculty/hyang8/systemc
bjdir/examples/tlm'
==================================================================
Testsuite summary for TLM 2.0.5
==================================================================
# TOTAL: 11
# PASS:  11
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
==================================================================
gmake[5]: Leaving directory `/DCNFS/users/faculty/hyang8/systemc
```

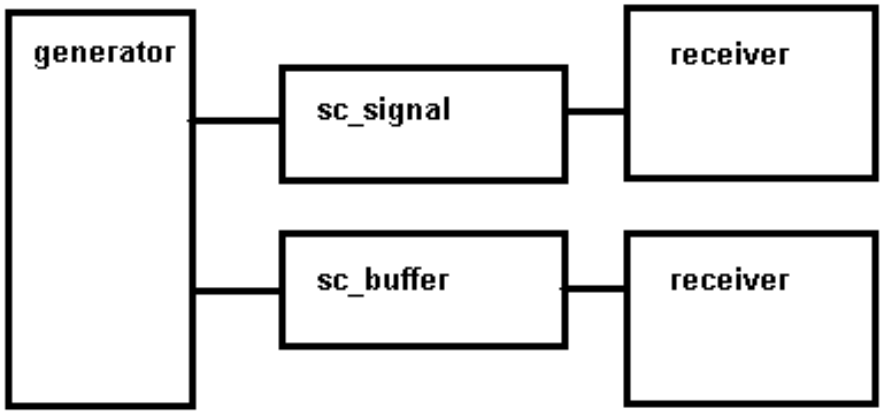
EDA Playground

- You may use EDA playground alternatively
  - https://www.edaplayground.com/
- (Important!) in case you use this
  - Save your work frequently and locally in your computer



Part 1: sc_signal vs. sc_buffer

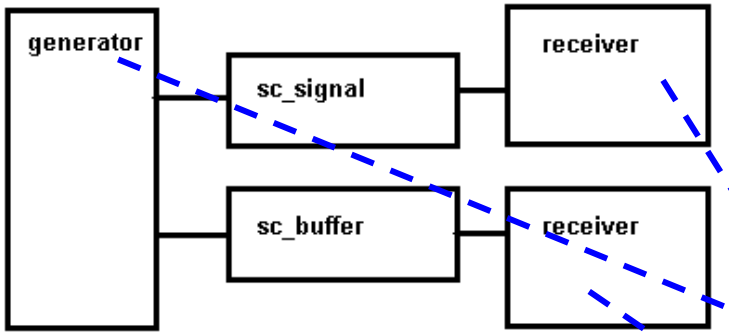
- Given the simple signal generator/receiver architecture (main.cpp)



- Customize the generator module and show (and explain) the difference between sc_signal and sc_buffer based on the console output

Part 1: sc_signal vs. sc_buffer (cont'd)

- main.cpp




```
1  // main.cpp
2  #include "systemc.h"
3
4  SC_MODULE (generator) {
5    sc_out<short>  sig;
6    sc_out<short>  buf;
7
8    void do_it(void) {
9      wait(5, SC_NS);
10     sig.write(5); buf.write(5); wait(10, SC_NS);
11         // put your codes here
12         // ...
13   }
14
15   SC_CTOR(generator) {
16     SC_THREAD(do_it);
17     sig.initialize(0);
18     buf.initialize(0);
19   }
20 };
21
22
23
24 SC_MODULE (receiver) {
25   sc_in<short>   iport;
26
27   void do_it(void) {
28     cout << sc_time_stamp() << ":" << name() << " got " << iport.read() << endl;
29   }
30
31   SC_CTOR(receiver) {
32     SC_METHOD(do_it);
33     sensitive << iport;
34     dont_initialize();
35   }
36 };
37
38 int sc_main(int argc, char *argv[]) {
39   sc_signal<short> sig;
40   sc_buffer<short> buf;
41
42   generator GEN("GEN");
43   GEN.sig(sig); GEN.buf(buf);
44
45   receiver  REV_SIG("REV_SIG");
46   REV_SIG.iport(sig);
47   receiver  REV_BUF_A("REV_BUF");
48   REV_BUF_A.iport(buf);
49
50   // trace file creation
51   sc_trace_file *tf = sc_create_vcd_trace_file("wave");
52   sc_trace(tf, sig, "sig"); sc_trace(tf, buf, "buf");
53
54   sc_start();
55   sc_close_vcd_trace_file(tf);
56   return(0);
57 }
58
```

Part 2: SystemC Modules

- Given source codes
  - Top-level file: main2.cpp
  - Input stimulus: stimulus.h
  - Adders: adder_method.h, adder_thread1.h, adder_thread2.h, adder_thread3.h, adder_cthread.h

- Different adder modules are given as follows
  - Adder_method : SC_METHOD
  - Adder_thread : SC_THREAD, sensitivity – clk
  - Adder_thread2 : SC_THREAD, sensitivity – a, b
  - Adder_thread3 : SC_THREAD, sensitivity – a, b, clk
  - Adder_cthread : SC_THREAD

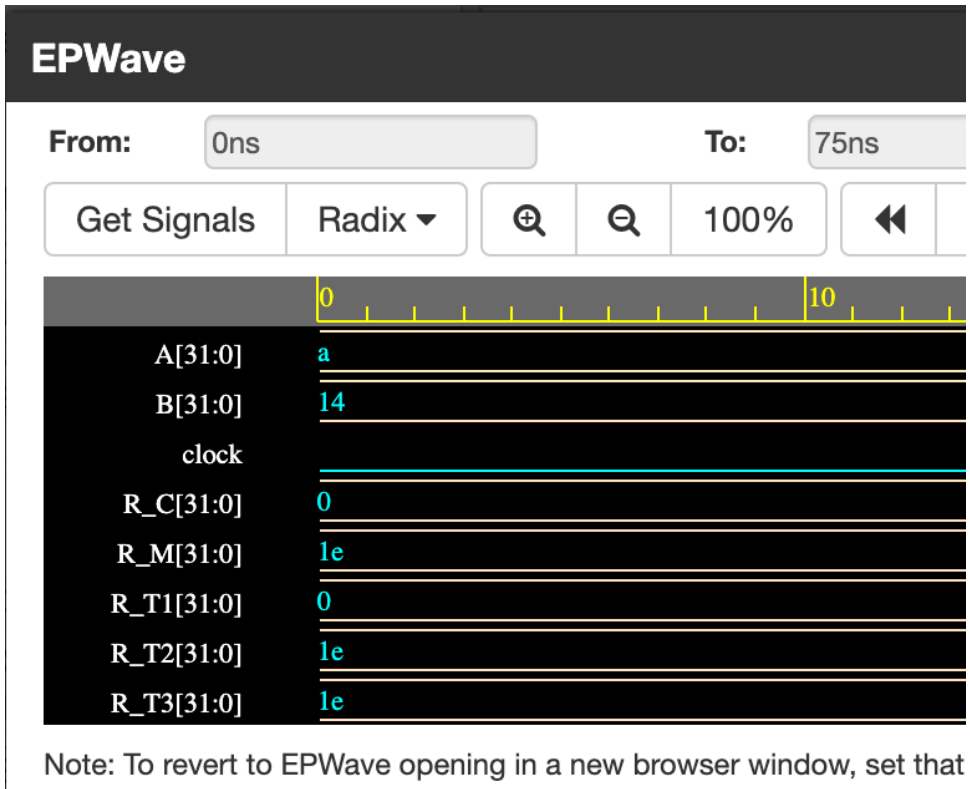
Part 2: SystemC Modules (cont'd)

1. Complete the top-level file (main2.cpp)
   - Connection between stimulus and adder

```
 1 // main.cpp
 2 #include "adder_method.h"
 3 #include "adder_thread1.h"
 4 #include "adder_thread2.h"
 5 #include "adder_thread3.h"
 6 #include "adder_cthread.h"
 7 #include "stimulus.h"
 8 int sc_main(int argc, char *argv[]) {
 9   sc_signal<int> A, B, R_M, R_T1, R_T2, R_T3, R_C;
10   sc_set_time_resolution(1, SC_NS); // V2.0
11   sc_set_default_time_unit(1, SC_NS);
12   sc_clock clock("clock", 20, SC_NS, 0.5, 15, SC_NS);
13
14   // example: adder_method
15   adder_method AD_M("adder_method"); // instantiation
16   AD_M(A, B, R_M);                   // port connection
17   // instantiate other adders too
18   // put your code here ...
19
20
21   stimulus STIM("stimulus");  // instantiation
22   STIM(A, B, clock); // port connection
23
24   // trace file creation
25   sc_trace_file *tf = sc_create_vcd_trace_file("wave");
26   sc_trace(tf, clock, "clock");
27   sc_trace(tf, A,     "A");
28   sc_trace(tf, B,     "B");
29   sc_trace(tf, R_M,   "R_M");
30   sc_trace(tf, R_T1,  "R_T1");
31   sc_trace(tf, R_T2,  "R_T2");
32   sc_trace(tf, R_T3,  "R_T3");
33   sc_trace(tf, R_C,   "R_C");
34   sc_start(80, SC_NS);
35   return(0);
36 }
37
38
```

Part 2: SystemC Modules (cont'd)

2. Based on the waveform result
   - Explain how the given modules differ from each other



Part 3: Fibonacci Series

- From scratch, build a SystemC model that output Fibonacci series (based on your work in Homework #1)
  - No source code is given
  - You must use at least one sc_fifo
  - Use the SDF model you have done in Homework #1

- It would be helpful to have a look at simple_fifo example
  - $SYSTEMC/examples/sysc/simple_fifo

Part 4: TLM simulation of MatMult

- From scratch, build a loosely-timed modeling of a matrix multiplication
  - No source code is given
  - one CPU (initiator) + one memory (target) connected through 32-bit (4B) bus
    - Assume that multiplication and addition takes 2ns and 1ns respectively
    - Assume that it takes 5ns for initiating a read/write and it takes additional 5ns for 4-byte read/write
      - Single 4B read → 5ns + 5ns = 10ns
    - Burst read/write transactions available
      - Burst 12B write → 5ns (initiation) + 5ns * 3 = 20ns

- Write a SystemC code that simulates a matrix multiplication between two random 16x16 matrices (each element being integer – 4B)

- Loosely-timed modeling
  - Use *tlm_generic_payload* and *b_transport* (of TLM 2.0)
  - You may customize the TLM 2.0 Tutorial 1 example
    - https://www.doulos.com/knowhow/systemc/tlm-20/tutorial-1-sockets-generic-payload-blocking-transport/

Part 4: TLM simulation of MatMult (cont'd)

- The initiator module should perform the following steps
  1. Writing two randomly generated 16x16 matrices into memory (target)
     - Assume that rand() takes no time
  2. Perform matrix multiplication by doing the following steps iteratively
     1) Reading *n* pair of elements from the two matrices
     2) Perform *n* multiply-accumulate (MAC) operations
     3) Update the output matrix
  3. Read the output matrix and print the output and timing information

Extra 2 Points – TLM simulation of MAC Accelerator

- Assume that you consider adding a MAC accelerator that can perform a MAC operation for 4 pairs of integer values within 2ns
  - Still you have to feed the input values (4 pairs of integer values) to this accelerator through the same bus
  - You also have to read the output (single integer value) value after its completion
- Extend your work to support this accelerator
- Show the timing improvement by this accelerator through SystemC simulation

Report

- Your report must include following
  - Part 1
    - Source code
    - Waveform or console output that explains the difference btw sc_buffer and sc_signal
    - Your analysis
  - Part 2
    - Source code
    - Waveform (screenshot)
    - Your analysis and explanation on the differences btw the given modules
  - Part 3
    - Source code
    - Output (waveform or console output)
    - Block diagram of your SystemC Fibonacci model
  - Part 4 (and extra points)
    - Source code
    - Output (console output with timing information)