**ELEC6233 — SystemC Assignment**

Yuting

Yg2m19

Microelectronics System Design

Terrence Mak

1. **Introduction**

*The objective of this assignments is familiar with another RTL-level language SystemC.*

*State the objectives of the assignment. The process of this work is like this: first, after completing the code of the given example, implement a certain function; second, complete the second design according to the given comments; finally, implement an error counter in combination with the first two designs.* *The entire compilation environment is executed in modelsim in the linux operating system. The simulation results show a great agreement with the code that required.*

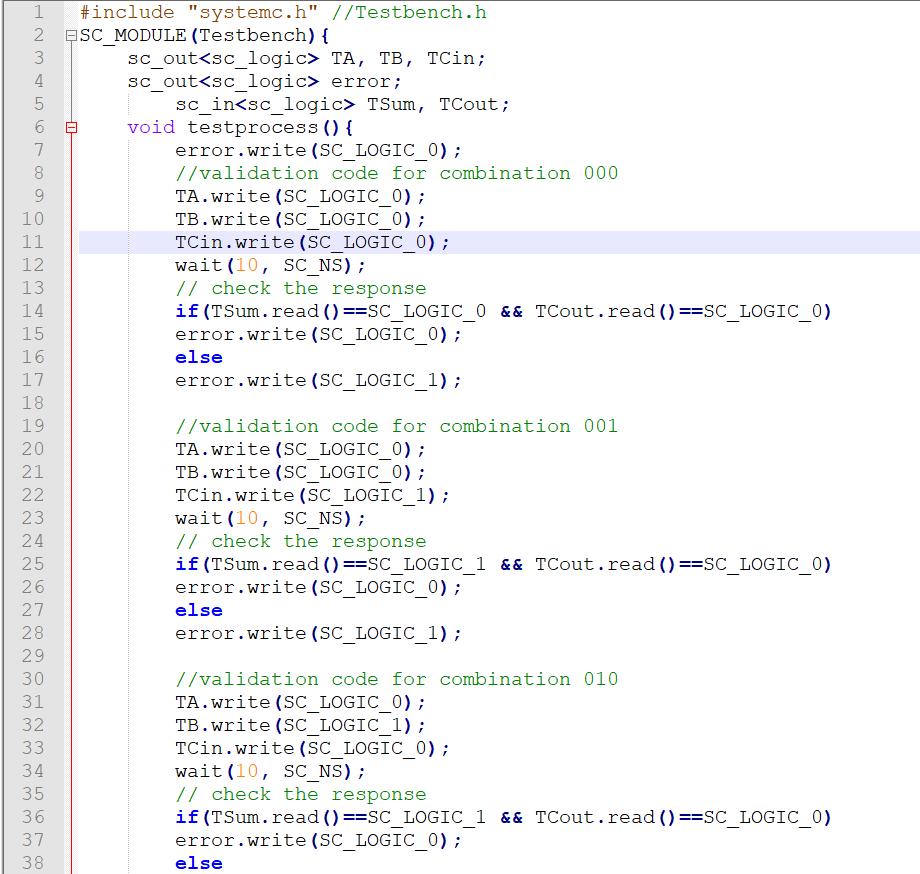
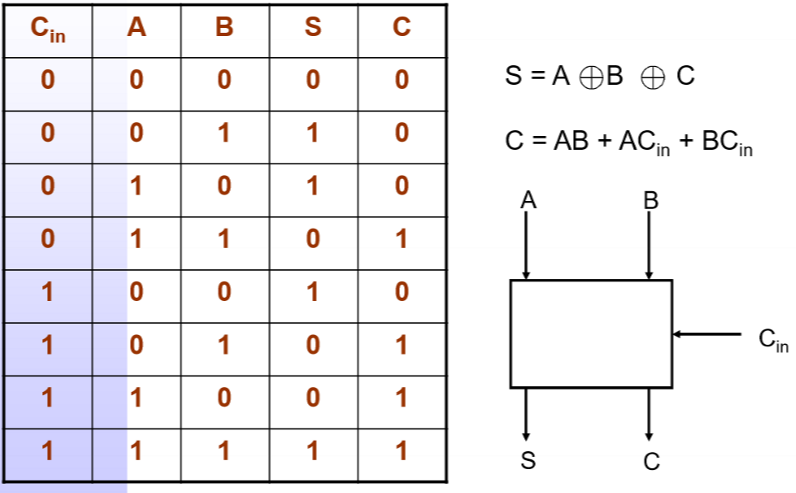
**2. Overall SystemC**

*SystemC is a codesign language supporting hardware and software. It basically is a library of classes and macros which are suitable for hardware modelling.* *Compared to SystemVerilog, SystemC supports system-level hardware description and can also meet the behavior description of RTL-level circuits. In the convention design methodology, the manual conversion from high-level C,C++ system model to a HDL language is required. ‘This process is very tedious and error prone.’ Thanks to the System C design approach that refines different part of code, a single language can finish the conversion from the system level to RTL. Therefore, SystemC* *is more suitable for system or architecture level behavior description.*

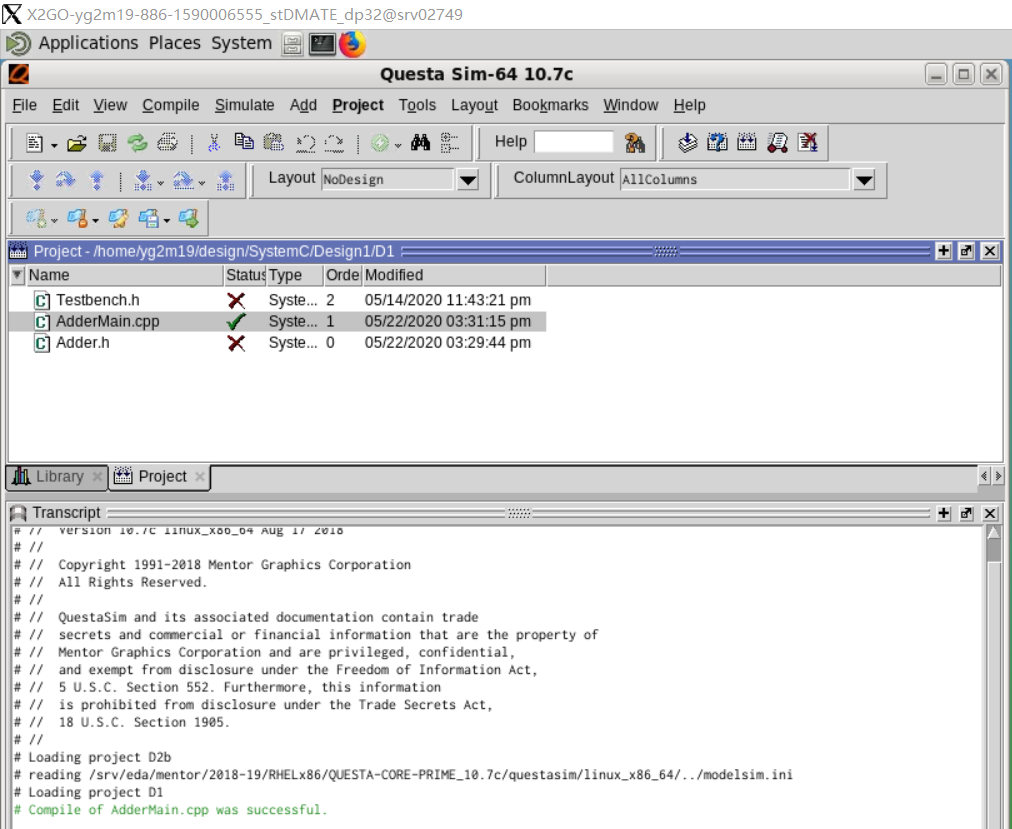
**3. Details of the assignment**

**3.1 Design 1: Modify the test bench of the full-adder and validation.**

*According to the true table of full adder, the test bench can be modified shown in the figure blow:*

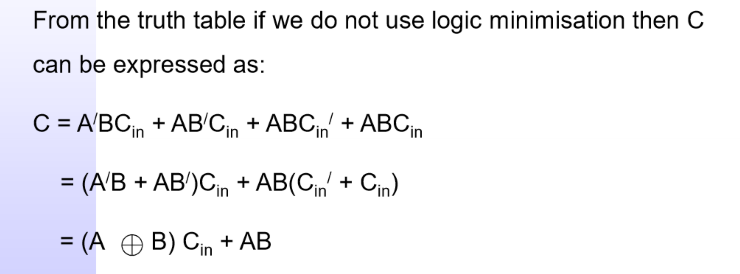


*Questa prompts code compilation success.*

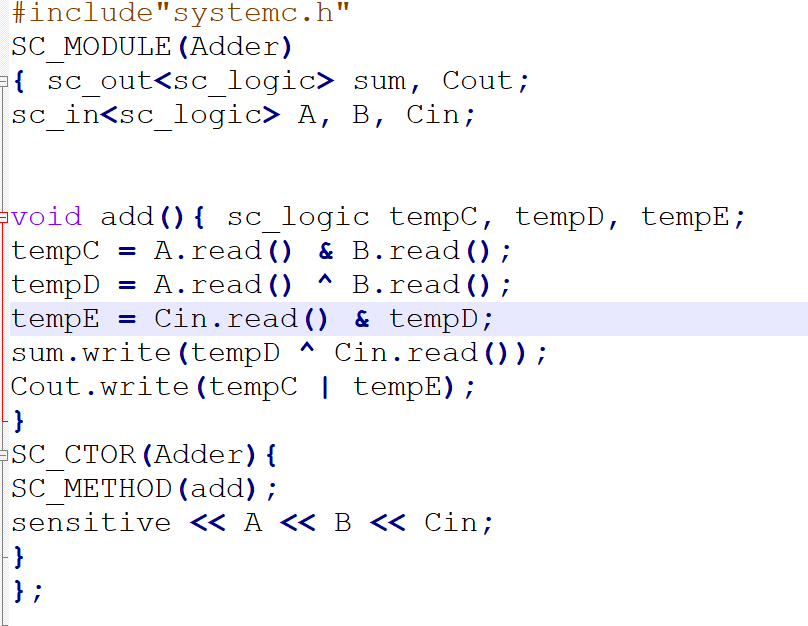


*The given code of the Adder.h is wrong. According to the Boolen function of the full adder,*

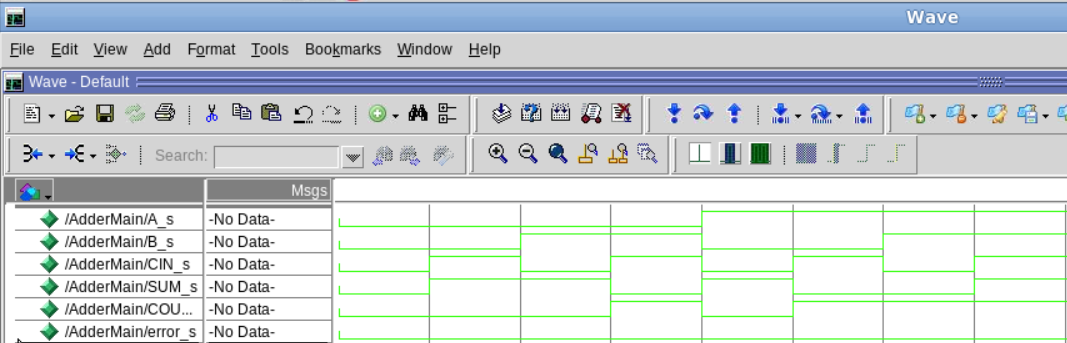
*tempE should be Cin.read() & tempD.*



*Therefore, the code can be* *modified like this:*



*The corresponding waveforms are as follows: exactly matches the truth table and no error signal been detected.*



**3.2 Design 2: Follow the instructions in the comments to complete an 8bit-counter.**

**Design2, File Counter8bit.h**

**#include"systemc.h"**//Include SystemC header file

**SC\_MODULE (Counter8bit) {**

**sc\_in\_clk clock;**//Define a clock inlet for the Counter, named 'clock'

**sc\_in<bool> reset;**//Define an input port of bool type, named 'reset'

**sc\_in<bool> enable;**//Define an input port of bool type, named 'enable'

**sc\_out<sc\_uint <8> >counter\_out;**//Define an output port of 8-bit SystemC unsigned integer type, named 'counter\_out'

**sc\_uint<8> count;** //Define a local register variable of 8-bit SystemC unsigned integer type, named 'count'

**void incr\_count()**{ //Define a void function with no parameters and body, named 'incr\_count', -start of 'incr\_count'

**while(true){**

**if(enable.read() == true){** //If enable is true, then, -start If

**if(reset.read() == true){** //If reset is true, -start If

**count = 0;**  //count is set to zero

**counter\_out.write(count);** //counter\_out is written with value count }// -end if

**else{** //else -start else

**count = count + 1;** //Increment count by 1

**counter\_out.write(count);** //counter\_out is written with value count }//end else

**}** // -end if

**wait(clock.posedge\_event());** // wait for next change of sensitivity variable/signal

**}**

**}** //-end of 'incr\_count'

**SC\_CTOR(Counter8bit){**  //Start Constructor

**SC\_THREAD(incr\_count);**//Register 'incr\_count' as SC\_THREAD process

**sensitive << clock.pos()**;//Make 'incr\_count' sensitive to clock's positive edge

**count = 0;** // Initialize count as ZERO

**}** //End Constructor End ConstructorEnd Constructor

**};**

**Design2, File CounterMain.cpp**

#include"Counter8bit.h"

#include"Testbench.h" //Include the Module header files

**SC\_MODULE(CounterMain){**

//Declare signals to be tied to the modules

**sc\_clock clock;** //Declare the clock, named 'clock' (without constructor)

**sc\_signal<bool> reset\_s,enable\_s;** //Declare signals necessary for interconnecting 'reset', 'enable' of the TestbenchDesign2 and Counter8bit module

**sc\_signal<sc\_uint <8> > counter\_out\_s;** //Declare signal for connecting the 'counter\_out' of Counter8bit module

**Counter8bit counter;** //Declare an instance of Counter8bit, named 'counter' (without constructor)

**Testbench test1;** //Declare an instance of TestbenchDesign2, named 'test1' (without constructor)

//Notice how Modelsim expects the constructor of Top-level Module

**SC\_CTOR(CounterMain):clock("SystemClock", 2, 0.5, true), counter("Counter1"), test1("Test1"){**

//Interconnect 'reset', 'enable', 'clock', 'counter\_out' of 'counter' with the signals necessary

**counter.clock(clock);**

**counter.reset(reset\_s);**

**counter.enable(enable\_s);**

**counter.counter\_out(counter\_out\_s);**

**test1.reset(reset\_s);**

**test1.enable(enable\_s);**

**test1.clk(clock);** //Interconnect 'reset', 'enable' of 'test1' with the signals necessary

**}**

**};** **SC\_MODULE\_EXPORT(CounterMain);** //Notice this is most crucial part in making Top-Level Module of Modelsim

**Design2, File Testbench.h**

**#include "systemc.h"**

**SC\_MODULE(Testbench)**{//Start 'TestbenchDesign2' Module

**sc\_out<bool> reset;**//Declare an output port, named 'reset'

**sc\_out<bool> enable;**//Declare an output port, named 'enable'

**sc\_in\_clk clk;**

**void testprocess()**{//Declare a void return type function with no parameters, named 'testprocess', -start of 'testprocess'

**enable.write(false);**

**reset.write(false);**//Write enable as false and reset and false

**wait(10,SC\_NS);**//Wait for 10 ns

**enable.write(true)**;

**reset.write(true)**;//Write enable as true and reset and true

**wait(10,SC\_NS);**//Wait for 10 ns

**enable.write(true);**

**reset.write(false)**;//Write enable as true and reset and false

**wait(10,SC\_NS);**//Wait for 10 ns

**enable.write(false);**

**reset.write(true);**//Write enable as false and reset and true

**wait(10,SC\_NS);**//Wait for 10 ns

**enable.write(false);**

**reset.write(false);**//Write enable as false and reset and false

**wait(10,SC\_NS);**//Wait for 10 ns

**}**//-end of 'testprocess'

**SC\_CTOR(Testbench){**//Start Constructor

**SC\_THREAD(testprocess);**//Register 'testprocess' as SC\_THREAD process

**sensitive << clk.pos();**

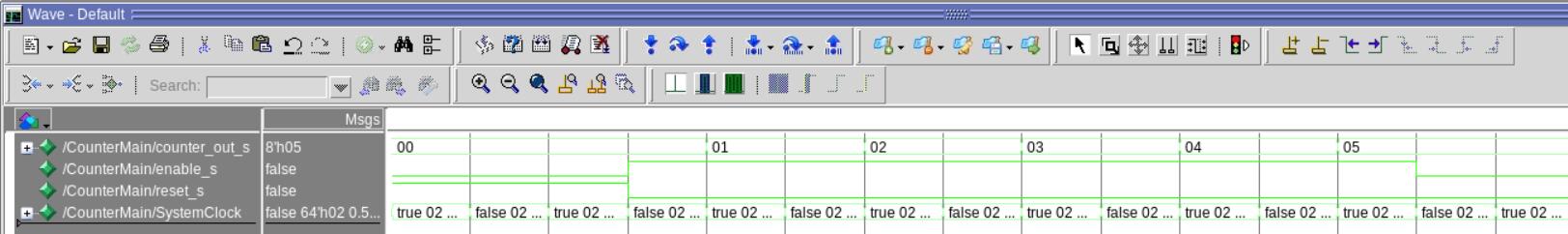
// Initialize count as ZERO

**}**//End Constructor

**}**;//End ' Testbench' Module

*The simulation result is shown below: The testbench provides a square wave signal with a 50% duty cycle (2ns) as a clock stimulus. When the enable signal is 1 and the reset signal is 0, the counter starts to record the rising edge of the clock, and the counter increases by one for each clock transition.*

*During that time, according to the testbench, there are 5 clock cycles. Therefore, the result of counter\_out\_s is 5. So, the waveform indicates the required 8-bit counter work functionally correct.*



**3.3 Design 2b: Combine the design2 with original design1 to count the number of error output.**

*In order to implement this counter that counts errors, the top-level file and testbench file should be modified as shown below:*

*The error signal output by the full adder is used as the counter input enable signal.*

***CounterMain3.cpp***

*///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*

**#include “Counter8bit.h”**

**#include “Testbench3.h”**

**#include”Adder.h”**

**SC\_MODULE(CounterMain3){**

**//Declare signals to be tied to the modules**

**sc\_clock clock;**

**sc\_signal<sc\_logic> A\_s, B\_s;**

**sc\_signal<bool> reset\_s; //enable\_s;**

**sc\_signal<bool> error\_s;**

**sc\_signal<sc\_logic> CIN\_s, SUM\_s, COUT\_s;**

**sc\_signal<sc\_uint <8> > counter\_out\_s;**

**Counter8bit counter;**

**Adder adder;**

**Testbench3 test1;**

**//Instantiate Adder and Bind Ports**

**SC\_CTOR(CounterMain3): clock(“SystemClock”, 10, 0.5, true), counter(“Counter1”), test1(“Test1”), adder(“Adder1”)**

**{**

**adder.A(A\_s);**

**adder.B(B\_s);**

**adder.Cin(CIN\_s);**

**adder.sum(SUM\_s);**

**adder.Cout(COUT\_s);**

**counter.clock(clock);**

**counter.reset(reset\_s);**

**counter.enable(error\_s); // error signal enable counter starts to work**

**counter.counter\_out(counter\_out\_s);**

**//Instantiate Testbench And Bind Ports**

**test1.reset(reset\_s);**

**test1.clk(clock);**

**test1.error(error\_s);**

**test1.TA(A\_s);**

**test1.TB(B\_s);**

**test1.TCin(CIN\_s);**

**test1.TSum(SUM\_s);**

**test1.TCout(COUT\_s);**

**}**

**};**

**SC\_MODULE\_EXPORT(CounterMain3);**

**///////////////////////////////////////////////////////////////////////////////////////////////////////**

***Testbench3.h***

**#include "systemc.h"**

**SC\_MODULE(Testbench3){**

**sc\_out<bool> reset;**

**sc\_out<bool> error;**

**sc\_in\_clk clk;**

**sc\_out<sc\_logic> TA, TB, TCin;**

**sc\_in<sc\_logic> TSum, TCout;**

**void testprocess(){**

**reset.write(false);**

**error.write(false);**

**//validation code for combination 000**

**TA.write(SC\_LOGIC\_0);**

**TB.write(SC\_LOGIC\_0);**

**TCin.write(SC\_LOGIC\_0);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_0 && TCout.read()==SC\_LOGIC\_0)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 001**

**TA.write(SC\_LOGIC\_0);**

**TB.write(SC\_LOGIC\_0);**

**TCin.write(SC\_LOGIC\_1);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_1 && TCout.read()==SC\_LOGIC\_0)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 010**

**TA.write(SC\_LOGIC\_0);**

**TB.write(SC\_LOGIC\_1);**

**TCin.write(SC\_LOGIC\_0);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_1 && TCout.read()==SC\_LOGIC\_0)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 011**

**TA.write(SC\_LOGIC\_0);**

**TB.write(SC\_LOGIC\_1);**

**TCin.write(SC\_LOGIC\_1);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_0 && TCout.read()==SC\_LOGIC\_1)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 100**

**TA.write(SC\_LOGIC\_1);**

**TB.write(SC\_LOGIC\_0);**

**TCin.write(SC\_LOGIC\_0);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_1 && TCout.read()==SC\_LOGIC\_0)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 101**

**TA.write(SC\_LOGIC\_1);**

**TB.write(SC\_LOGIC\_0);**

**TCin.write(SC\_LOGIC\_1);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_0 && TCout.read()==SC\_LOGIC\_1)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 110**

**TA.write(SC\_LOGIC\_1);**

**TB.write(SC\_LOGIC\_1);**

**TCin.write(SC\_LOGIC\_0);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_0 && TCout.read()==SC\_LOGIC\_1)**

**error.write(false);**

**else error.write(true);**

**//validation code for combination 111**

**TA.write(SC\_LOGIC\_1);**

**TB.write(SC\_LOGIC\_1);**

**TCin.write(SC\_LOGIC\_1);**

**wait(10, SC\_NS);**

**if(TSum.read()==SC\_LOGIC\_1 && TCout.read()==SC\_LOGIC\_1)**

**error.write(false);**

**else error.write(true);**

**}**

**SC\_CTOR(Testbench3){**

**SC\_THREAD(testprocess);**

**sensitive << clk.pos();**

**}**

**};**

**///////////////////////////////////////////////////////////////////////////////////////////////////////**

*There are 3 inputs that lead to error in the original Design1. So, the error counter should keep the value of 3. The waveform of it is show below.*

