ELEC6233 – SystemC coursework

Ahmed Soliman
AS8U22
EMECS – Embedded Computing Systems – Part 1
Tomasz Kazmierski

ABSTRACT: *In this coursework, two circuit designs, an Adder and a Counter, were designed and verified using SystemC language. The Adder circuit was extended with additional test cases and its behaviour was validated through simulation and analytical verification. The Counter circuit was designed, tested with a custom testbench, and verified through simulation. The project provided a better understanding of digital circuit design and simulation, and highlighted possible improvements such as adding more advanced features to the Counter circuit or expanding the Adder circuit's functionality.*

1. Introduction

This coursework involved designing and verifying two digital circuits, an Adder and a Counter, using SystemC language. In this report, we will discuss which parts of the assignment were delivered according to the requirements and highlight important issues encountered.
For the Adder circuit, we were required to modify the testbench and validate its behaviour through simulation and analytical verification.
As for the Counter circuit, we needed to write the code based on given comments, simulate it using a custom testbench, and verify its functionality.

2. Design 1: Full-Adder

For the adder there were 2 requirements. Firstly, we were given code files with testbench and We were asked to modify the testbench to validate the code operation. The testbench only contained 3 cases for the 3 inputs (A, B, Cin) so it was missing 5 more cases which I added to test the full functionality of the adder which can be seen in figure 1.

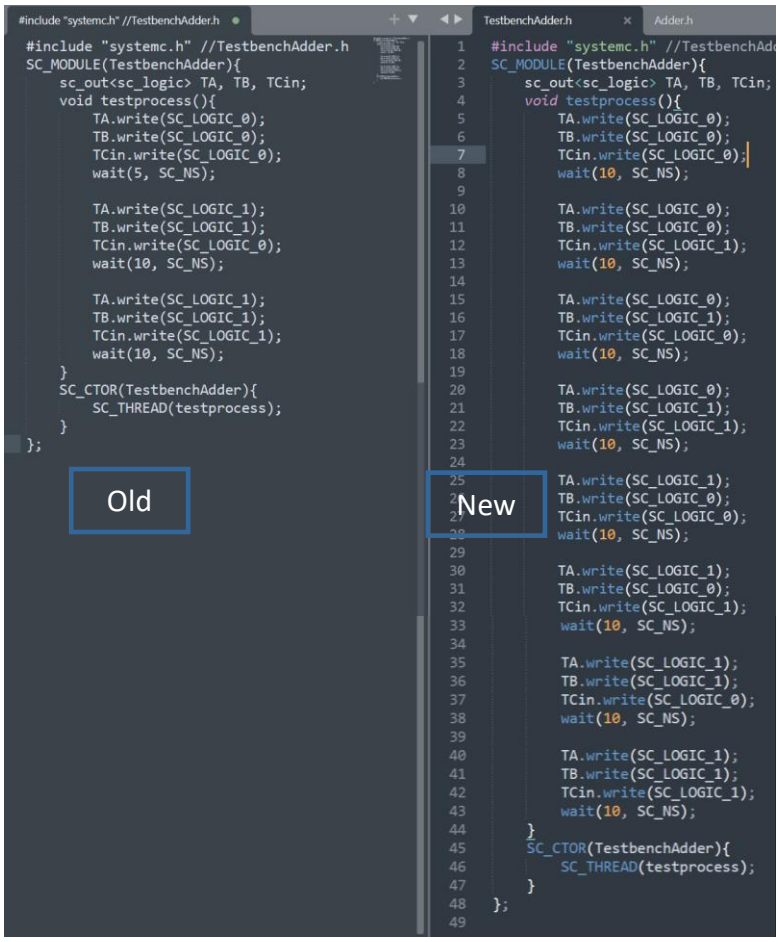

Figure 1

The second requirement was to observe the behaviour of the simulation and report any error. In order to verify the behaviour, we must first know the truth table for a full adder which can be seen in figure 2.

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2

And from the wave form in the figure below for the adder simulation we can see that for every input value (orange line is zero, Blue line is 1) for A, B, Cin the output value for Sum and Cout is the same as the table above which shows that the code behaves correctly without any error. For example, at instance 36 ns input values for A, B, Cin are 0,1,1 respectively and the output for Sum, Cout are 0,1 respectively which is same in truth table above.

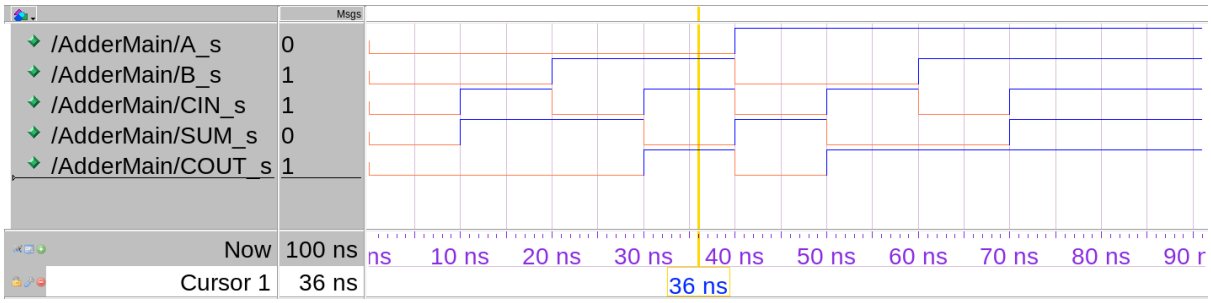

Figure 3

3. Design 2: 8-bit Counter

For the second Design (Counter), there was 2 requirements needed. First we need to write the Counter code using systemC based on the comments written in the file, then we need to simulate and validate the counter through a testbench.

Accordingly I wrote the code for the files as seen in the figures below.

In CounterMain declaired signals needed for the counter to work like enable, reset and output of the counter were created. Also, Instances for the counter and testbench were created. After that binding of the signals created to the signals inside the modules were made.

CounterMain.cpp

```
#include "counter8bit.h"
#include "TestbenchCounter.h"

SC_MODULE(CounterMain){
    sc_clock     clock ;
    sc_signal<bool> reset_s,enable_s;
    sc_signal <sc_uint<8> > counter_out_s;
    counter8bit counter;
    TestbenchCounter test1;

    SC_CTOR(CounterMain): clock("SystemClock", 2, 0.5, true), counter("incr_counter"), test1("testprocess"){

        counter.reset(reset_s);
        counter.enable(enable_s);
        counter.clock(clock);
        counter.counter_out(counter_out_s);


        test1.reset(reset_s);
        test1.enable(enable_s);
    }
 };

SC_MODULE_EXPORT(CounterMain);

```

Figure 4

As for the actual implementation for the counter, it was made in the .h file which can be seen that the counter can only count or reset if enable was true otherwise it retain its value. And in the constructor the counter is senstive to the clock positive edge signal only.
In the testbench several tests were made to validate the code in which it tested when on combinations between enable and reset (True and False) and also tested overflow of the counter which should be in a cycle and return counting from 0.

counter8bit.h

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

SC_MODULE (counter8bit) {
    sc_in    <bool> clock;
    sc_in    <bool> reset;
    sc_in    <bool> enable;
    sc_out   <sc_uint<8> > counter_out;
    sc_uint<8>  count;
    void incr_counter () {
        while(true){
        if(enable.read()){
            if(reset.read()){
                count = 0 ;
                counter_out.write(count);
            }
            else{
                count = count + 1;
                counter_out.write(count);
                }
            }
            wait();
        }
    }

    SC_CTOR(counter8bit) {
        SC_THREAD (incr_counter);
        sensitive << clock.pos();
        count=0;
    }
};
```

CounterMain.cpp

TestbenchCounter.h

```
#include "systemc.h"
SC_MODULE(TestbenchCounter){
    sc_out<bool> reset, enable;
    void testprocess(){
        enable.write(false);
        reset.write(false);
        wait(10, SC_NS);

        enable.write(true);
        reset.write(true);
        wait(10, SC_NS);

        enable.write(true);
        reset.write(false);
        wait(10, SC_NS);

        enable.write(false);
        reset.write(true);
        wait(10, SC_NS);

        enable.write(false);
        reset.write(false);
        wait(10, SC_NS);

        enable.write(true);
        reset.write(true);
        wait(10, SC_NS);

        enable.write(true);
        reset.write(false);
        wait(600, SC_NS);

    }
    SC_CTOR(TestbenchCounter){
        SC_THREAD(testprocess);
    }

};
```

Figure 5

After completion of writing the code simulation was done to verify it. And as can be seen in figure 6 in the Blue region enable was set to 0 so counter didn't increment, in the red region enable was 1 but reset was 1 so counter remained at 0, now in the green region counter started counting as enable was 1 while reset is 0.

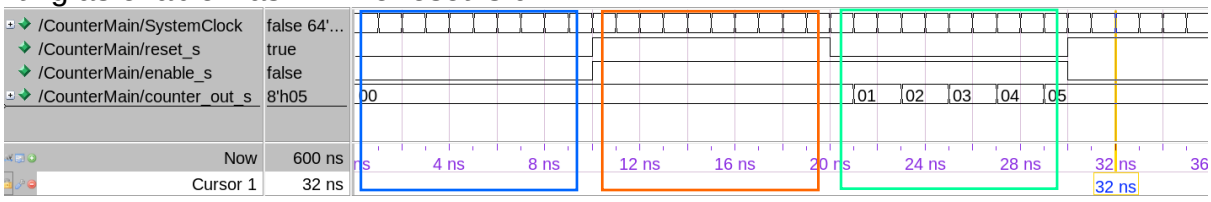

Figure 6

In figure 7, as can be seen in the violet region reset was 1 but counter retained its value as enable was set to 0, but when enable was 1 and reset was 1 counter was reset as seen in the yellow region.

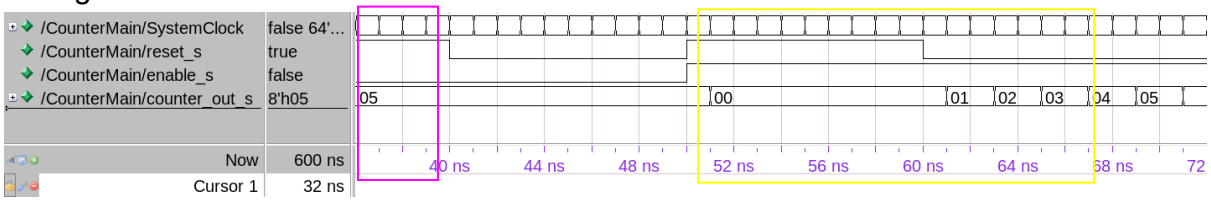

Figure 7

Finally, Figure 8 shows the overflow of the counter when it reached its maximum value then it started counting from the beginning again.

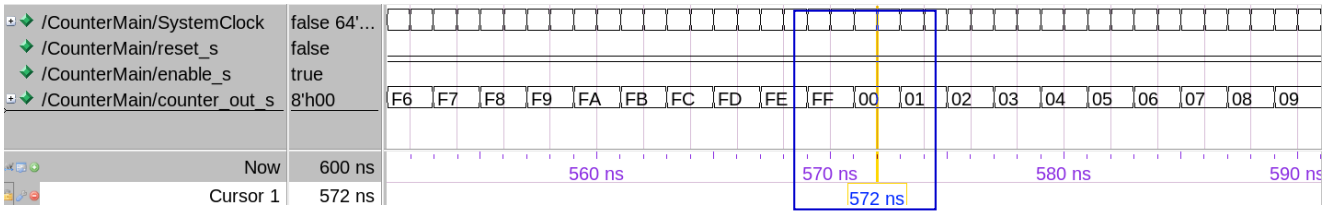

Figure 8

3. Conclusion

Throughout this project, I gained a deeper understanding of digital circuit design and simulation through Systemverilog, as well as the benefits of using SystemC language for such tasks. Specifically, I learned how to modify testbenches to validate circuit operation, observe and analyse simulation waveforms to verify behaviour, and create and simulate custom circuits with specific functionality.

In terms of future improvements, one possible extension would be to add more advanced features to the Counter circuit, such as the ability to count up or down based on user input or to generate specific sequences of values. Additionally, the Adder circuit could be expanded to handle larger input sizes or to be used in more complex arithmetic operations.

Overall, this coursework provided a valuable opportunity to apply theoretical knowledge of digital circuit design and simulation in a practical setting, while also highlighting areas for further exploration and improvement.

4. References

- ASIC World. (n.d.). SystemC Tutorial. Retrieved April 1, 2023, from http://www.asic-world.com/systemc/tutorial.html
- QuestaSim Tutorials provided on secure website