Department of Electrical and Computer Engineering
Stony Brook University

ESE 356 Digital System Specification and Modeling (Fall 2019)

Project No. 0
Due Date: September 12, 2018

This assignment is to be done individually. Submit both the design, test bench, and simulation results.

Problem 1: Design and verify a sequence detector. Output will set to "high" if "1011" bits are received. Output is "low" otherwise. Your hardware must meet the following specification:

Inputs:

```
sc_in<bool> clock;
sc_in<bool> reset; // clears all internal registers
sc_in<bool> clear; // clears output registers
sc_in<sc_bit> data_in; // single bit
```

Outputs:

```
sc_out<sc_bit> data_out; // single bit
```

Problem 2: Design logic to do the following:

- There are two 8-bit counters. Counter 1 is loaded from in1[7:0] when the signal load1 is high, and is decremented by 1 when dec1 is high. Counter 2 is loaded from in2[7:0] and is decremented by the amount specified in in3[7:0] when dec2 is high.
- Whenever the contents of the two 8-bit counters become the same as each other, then the output flag ended goes high and the counter contents must remain the same until one of them are reloaded from in1 or in2.
- Also, if either counter overflows (the counter contains unsigned numbers), then ended should go high and the counter contents must remain the same until one of them are reloaded from in1 or in2. (Overflow is indicated simply by a carry out = 1 in this case).
- The value on in3 will be held constant for you. You do not need to register it. Design and verify your design, as well as a timing diagram. Also, make your solution consistent with the input/output declaration below.

Inputs:

```
sc_in<bool> clock;
sc_in<sc_unit<8> > in1, in2, in3;
sc_in<sc_bit> load1;  // when load1 goes high,
                      //in1 is registered into count1
sc_in<sc_bit> load2;  // when load2 goes high,
                      // in2 is registered into count2
sc_in<sc_bit> dec1;  // decrement count1 by 1
sc_in<sc_bit> dec2;  // decrement count2 by in3
```

Outputs:

```
sc_out<sc_bv<8> > count1;  // contents of counter 1
sc_out<sc_bv<8> > count2;  // contents of counter 2
sc_out<sc_bit> ended;  // goes high when count1==count2,
                       // or either counter experiences
                       // an unsigned overflow
```

Problem 3: Design and verify a communications interface. Often data sent over a data link are organized as packets of data, each packet containing some identification bits, data, and some check bits used to determine if a transmission error has occurred. Your hardware must meet the following specification:

Inputs:

```
sc_in<bool>  clock;
sc_in<bool>  reset; //reset is active low
sc_in<bool>  clear; //clears output registers – active high
sc_in<sc_uint<12> > inData[11:0]; // input data:
                    // inData[11:8] contains the header
                    // inData[7:4] contains the data payload
                    // inData[3:1] are not used
                    // inData[0] is a parity bit.
                    // It is 1 if inData[7:4] is meant
                   // to be even parity.
                    // A new inData arrives every clock
```

Outputs:

All outputs are registered and are cleared when `reset' is low or `clear' is high.

```
sc_out<sc_uint<4> > payload; // when inData is of type1.
sc_out<sc_uint<8> > count;   // the total count of type1
                             // payload
sc_out<sc_uint<8> > error;   // the number of type1 payload
                             //with wrong parity
```

Clock 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

Clear 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

InData 1F1 0E0 171 0E0

(i.e., 1F1 : type1 == true, payload == F, parity should be even

0E0 : type1 == false, payload == E, parity should be odd

170 ; type1 == true, payload == 7, parity should be even

Payload 0 F F 7 7

The piece of hardware checks inData on each clock cycle. If inData[11:8] = 1, then it transfers the middle four bits of inData to payload and increments count. At the same time, it checks the parity of the middle four bits and sees if it is as expected. If it is not, then there is a transmission error, and error is incremented.