

**Department of Electrical and Computer Engineering
Stony Brook University**

ESE 501 System Specifications and Modeling (Fall 2009)

Assignment No. 1

This assignment is to be done individually. Submit both the design, test bench and simulation results.
This assignment is **due September 17, 2009 in class**. No electronic copy will be accepted.

Problem 1: 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.

```
module count_compare (clock, in1, in2, in3, load1, load2, dec1, dec2, count1, count2, ended);
input  clock;
input  [7:0] in1, in2, in3;
input  load1; // when load1 goes high, in1 is registered into count1
input  load2; // when load2 goes high, in2 is registered into count2
input  dec1; // decrement count1 by 1
input  dec2; // decrement count2 by in3
output [7:0] count1; // contents of counter 1
output [7:0] count2; // contents of counter 2
output ended; // goes high when count1==count2, or either counter experiences an unsigned overflow
              // note: count1 and count2 will not change after same goes high until load1 or load2 forces
              // one of the to be reloaded.
```

Problem 2: Design and verify a communications interface in Verilog. 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:

```

Clock;
Reset; //reset is active low
Clear; //clears output registers – active high
InData[11:0]; // Input data, organized as follows:
              // 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.
Payload[3:0]; // is changed to contain the `data payload` when `InData` is of type 1.
Count[7:0]; // total count of type 1 data
Error[7:0]; // number of type 1 data of 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
Clear 1 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 : `type 1`, payload=F, parity should be even
        0E0 : `not type 1`, payload=E, odd parity
        170 ; `type 1`, payload=7, payload should have been even parity
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.