Homework 1: 8-bit Adders Using Verilog in Structural, Dataflow, and Behavioral Levels

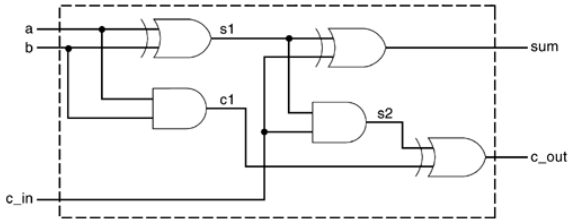
Due Oct. 12, 2015

1. Use Verilog HDL to design an 8-bit adder in different levels:

(1) Structure-Level Modeling

A logic circuit can be designed by use of logic gates. Verilog supports basic logic gates as predefined primitives. These primitives are instantiated like modules except that they are predefined in Verilog and do not need a module definition. All logic circuits can be designed by using basic gates. There are two classes of basic gates: and/or gates and buf/not gates. And/or gates have one scalar output and multiple scalar inputs. Buf/not gates have one scalar input and one or more scalar outputs. Use the primitives (**and, or, xor, nand, nor, xnor, not**) to design an 8-bit adder.

Hint: You can design 1-bit full adder first. The following figure shows the logic diagram for a 1-bit full adder. An 8-bit ripple carry adder can be constructed from eight 1-bit full adders.



(2) Data Flow Modeling

For small circuits, the gate-level modeling approach works very well because the number of gates is limited, and the designer can instantiate and connect every gate individually. However, in complex designs the number of gates is very large. Thus, designers can design more effectively if they concentrate on implementing the function at a level of abstraction higher than gate level. Dataflow modeling provides a powerful way to implement a design. Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than

instantiation of individual gates. A continuous assignment is the most basic statement in dataflow modeling, used to drive a value onto a net. Write the dataflow description for an 8-bit adder by using continuous assignment **assign.**

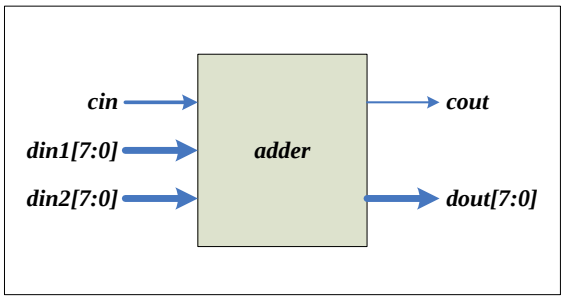
(3) Behavioral Modeling

Verilog provides designers the ability to describe design functionality in an algorithmic manner. In other words, the designer describes the behavior of the circuit. Thus, behavioral modeling represents the circuit at a very high level of abstraction. Design at this level resembles C programming more than it resembles digital circuit design. Behavioral Verilog constructs are similar to C language constructs in many ways. Verilog is rich in behavioral constructs that provide the designer with a great amount of flexibility. Model an 8-bit adder at behavior level.

Hint 1: Structured procedures **initial** and **always** form the basis of behavioral modeling. All other behavioral statements can appear only inside initial or always blocks. An initial block executes once; an always block executes continuously until simulation ends.

Hint 2: Procedural assignments are used in behavioral modeling to assign values to register variables. Blocking assignments (=) must complete before the succeeding statement can execute. They are usually used to describe combinational logic. Non-blocking assignments (<=) schedule assignments to be executed and continue processing to the succeeding statement. They are usually used to describe sequential logic.

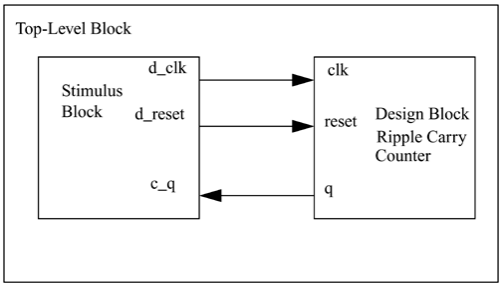
The following figure shows the ports of the adders. All ports declarations should be the same in the adders in this homework.



The module names of the adders should be **adder_struct**, **adder_dataflow**, and **adder_beh**.

2. Simulation of the 8-bit Adders

Once a design block is completed, it must be verified. The functionality of the design block can be verified by applying stimulus and checking results. We call such a block the stimulus block. It is a good practice to keep the stimulus and design blocks separate, like the following figure. A testbench consists of two parts: a stimulus block and a design block. The stimulus block generates input test vectors and compares the simulation results with golden references. Non-synthesizable Verilog code can be used in the stimulus block. The design block uses synthesizable Verilog codes to model the hardware.



Your testbench might use the system task **$random** to generate random numbers as input test vectors.

3. The above three different adder designs are pure combinational logic where the outputs are not latched, i.e, the outputs are not stored in flip flops. A flip-flop can be modeled as below:

```
module D_FF (q, d, clk);
output q;
input d, clk;
reg q;
always @( posedge clk)
                q <= d;
endmodule
```

Use flip-flops to latch the results of the outputs in the three different adder designs in the previous problems. Simulate again these sequential logic designs to verify their functions in RTL (register transfer level).

Report Requirement

The report should clearly explain your design and how you verify your design.

Email your report to TA's mailbox and also upload it to the course website.

Email's title format: HDL HW1 [your student ID number] [name]

Example: HDL HW1 M023040099 王小明

Your attachment should include HDL codes and all the supporting document (such as results from EDA tools and your explanations).