BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI – K. K. BIRLA GOA CAMPUS
Second Semester 2011-2012
CS C342 Advanced Computer Organization
Reading Material for VERILOG Lab: Published on 16th January 2012

What is Verilog?

Verilog is one of the two-major Hardware Description Languages (HDL) used by hardware designers in industry and academia. VHDL is the other one. Many feel that Verilog is easier to learn and use than VHDL. VHDL was made an IEEE Standard in 1987, and Verilog in 1995. Verilog is very similar to C-like.

Verilog allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i. e., gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication. A primary use of HDLs is the simulation of designs before the designer must commit to fabrication.

The Verilog language provides the digital designer with a means of describing a digital system at a wide range of levels of abstraction, and, at the same time, provides access to computer-aided design tools to aid in the design process at these levels.

Verilog allows hardware designers to express their design with **behavioral constructs**, deterring the details of implementation to a later stage of design. An abstract representation helps the designer explore architectural alternatives through **simulations** and to detect design bottlenecks before detailed design begins. Though the behavioral level of Verilog is a high level description of a digital system, it is still a precise notation. Computer-aided-design tools, i. e., programs, exist which will "compile" programs in the Verilog notation to the level of circuits consisting of logic gates and flip flops. One could then go to the lab and wire up the logical circuits and have a functioning system. And, other tools can "compile" programs in Verilog notation to a description of the integrated circuit masks for **very large scale integration** (VLSI). Therefore, with the proper automated tools, one can create a VLSI description of a design in Verilog and send the VLSI description via electronic mail to a **silicon foundry** and receive the integrated chip. Verilog also allows the designer to specify designs at the logical gate level using **gate constructs** and the transistor level using **switch constructs**.

Our goal in the course is not to create VLSI chips but to use Verilog to precisely describe the *functionality* of *any* digital system, for example, a computer. However, a VLSI chip designed using Verilog's behavioral constructs will be rather slow and be wasteful of chip area. The lower levels in Verilog allow engineers to optimize the logical circuits and VLSI layouts to maximize speed and minimize area of the VLSI chip.

Levels of Abstraction in Verilog Programming:

1) Gate Level Modeling
2) Data Flow Modeling
3) Behavioral Modeling
4) RTL Modeling

For the sake of simplicity, this manual takes you through all the levels of abstraction with a sample code for a 2:1 Multiplexer.

Before looking into the various levels of abstraction, we shall define Modules and Instances.

Module

A Module is a basic building block in Verilog. A module provides the necessary functionality to the higher-level block through its port interface (inputs and output) but hides the internal implementation. This allows the designer to modify module internals without affecting the rest of the design and its functionality.
Please refer Sec2.3 of the book “Verilog HDL” by Samir Palnitkar.

Instance

A module provides a template from which you can create objects/ instances of the module type. When a module is invoked, Verilog creates a unique object from the template. The process of creation objects from module template is called instantiation and the objects are called instance.
Please refer Sec2.4 of the book “Verilog HDL” by Samir Palnitkar.

Gate Level Modeling

This is the basic level of modeling in terms of logic gates and the connections between these gates. Most digital design is now done at the gate level or higher levels of abstractions. At gate level, the circuit is described in terms of gates say AND, OR etc. Hardware design at this level is intuitive for a user who is familiar with the basic knowledge of Digital logic Design. This allows the user to see a direct correspondence between the Verilog Description and the Circuit Diagram.

Example:

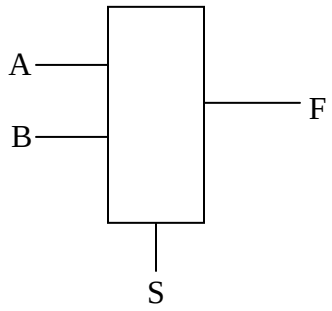


Figure 1. A 2:1 Multiplexer
In terms of Logic Gates, F= OR (AND (S, A), AND ((NOT (S)), B))

Code:

```
module mux2to1_gate (a,b,s,f);
        input a,b,s;
        output f;
        wire c,d,e;
        not n1(e,s);// e=~s
        and a1(c,a,s);
        and a2(d,b,e);
        or o1(f,c,d);
endmodule
```

For Further reading on Gate Level Modeling refer Chapter 5 of the book “Verilog HDL” by Samir Palnitkar.

Data Flow Modeling

The design at this level specifies how the data flows between the hardware registers and how the data is processed. For small circuits the gate level modeling works well as the number of gates is limited. However, in complex designs the designers may have to concentrate on implementing the function than bother about the gates. 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 gates using expressions (=), operators like (&,|, ?) etc.. and continuous assignments(the assign statement).

The 2:1 Multiplexer can be written as

```
F= (S&A)|(~S&B);
        Or
F = S ? A : B;
```

Code:

```
module mux2to1_df (a,b,s,f);
        input a,b,s;
        output f;
        assign f = s ? a : b;
endmodule
```

For Further reading on Data Flow Modeling refer Chapter 6 of the book “Verilog HDL” by Samir Palnitkar.

Behavioral Modeling

This is the highest level of abstraction provided by Verilog. The design at this level is similar to an algorithm. This design is very similar to ‘C’ programming. A module can be implemented in terms of the desired design algorithm without looking into the hardware details using structured procedures (like *always* and *initial*), conditional statements (like *if* and *else*) and multiway branching (like *case*, *casex* and *casez*).

[Note: THOUGH THE CODING STYLE IS SIMILAR TO C, THE DIFFERENCE IS THAT, THE VERILOG PROGRAM CONSISTS OF MODULES THAT MAY RUN CONCURRENTLY. Remember, that you need to simulate hardware and not software.]

The 2:1 Multiplexer can be written as

```
if(s= = 1)
        F=A;
else
        F=B;
```

Code:

```
module mux2to1_beh(a,b,s,f);
        input a,b,s;
        output f;
        reg f;
        always@(s or a or b)
                if(s==1) f = a;
                else f = b;
endmodule
```

For Further reading on Behavioral Modeling refer Chapter 7 of the book “Verilog HDL” by Samir Palnitkar.

RTL Modeling

The term Register Transfer Level Modeling refers to the Verilog description that uses a combination of both Behavioral and Data Flow constructs that is synthesizable. More on this later.
For the input and output connections for the ports, please refer Sec 4.2.3 Port Connection Rules.

Testbench or Stimulus

Once a design block is completed, it must be tested for its correctness. The functionality of a design block can be tested by applying stimulus and checking the results. We call such a block Stimulus or TestBench. It is recommended to keep the design and the stimulus blocks separate. Given below is an example to test the 2:1 Multiplexer we have designed so far.

Code:

```
module testbench;
        reg a,b,s;
        wire f;
        mux2to1_gate mux_gate (a,b,s,f);
        initial
                begin
                        $monitor(,$time," a=%b, b=%b, s=%b f=%b",a,b,s,f);
                        #0 a=1'b0;b=1'b1;
                        #2 s=1'b1;
                        #5 s=1'b0;
                        #10 a=1'b1;b=1'b0;
                        #15 s=1'b1;
                        #20 s=1'b0;
                        #100 $finish;
                end
endmodule
```

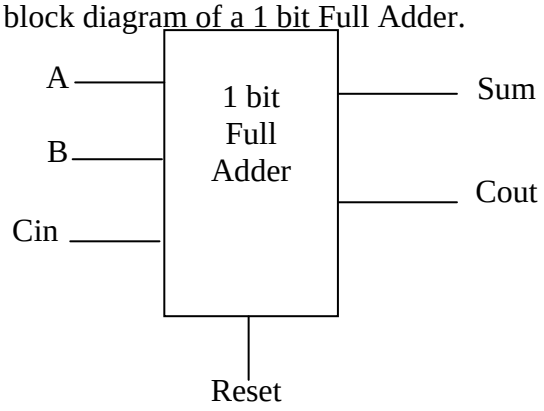
Note: To test out the data flow model and the behavioral model of the 2:1 Multiplexer, replace

```
mux2to1_gate mux_gate (a,b,s,f);
     with
mux2to1_df mux_df (a,b,s,f);
     or with
mux2to1_beh mux_beh (a,b,s,f);
```

Please refer section 2.5 and 2.6 of the book “Verilog HDL” by Samir Palnitkar.

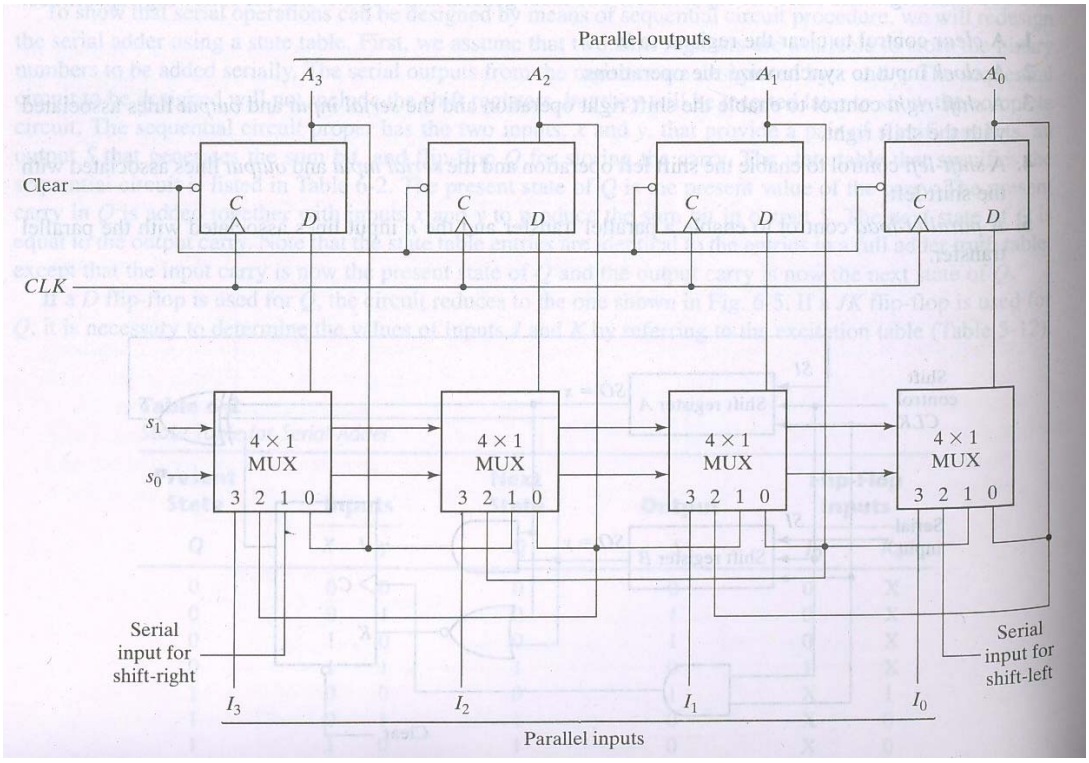
Exercises

1. Given below the block diagram of a 1 bit Full Adder.

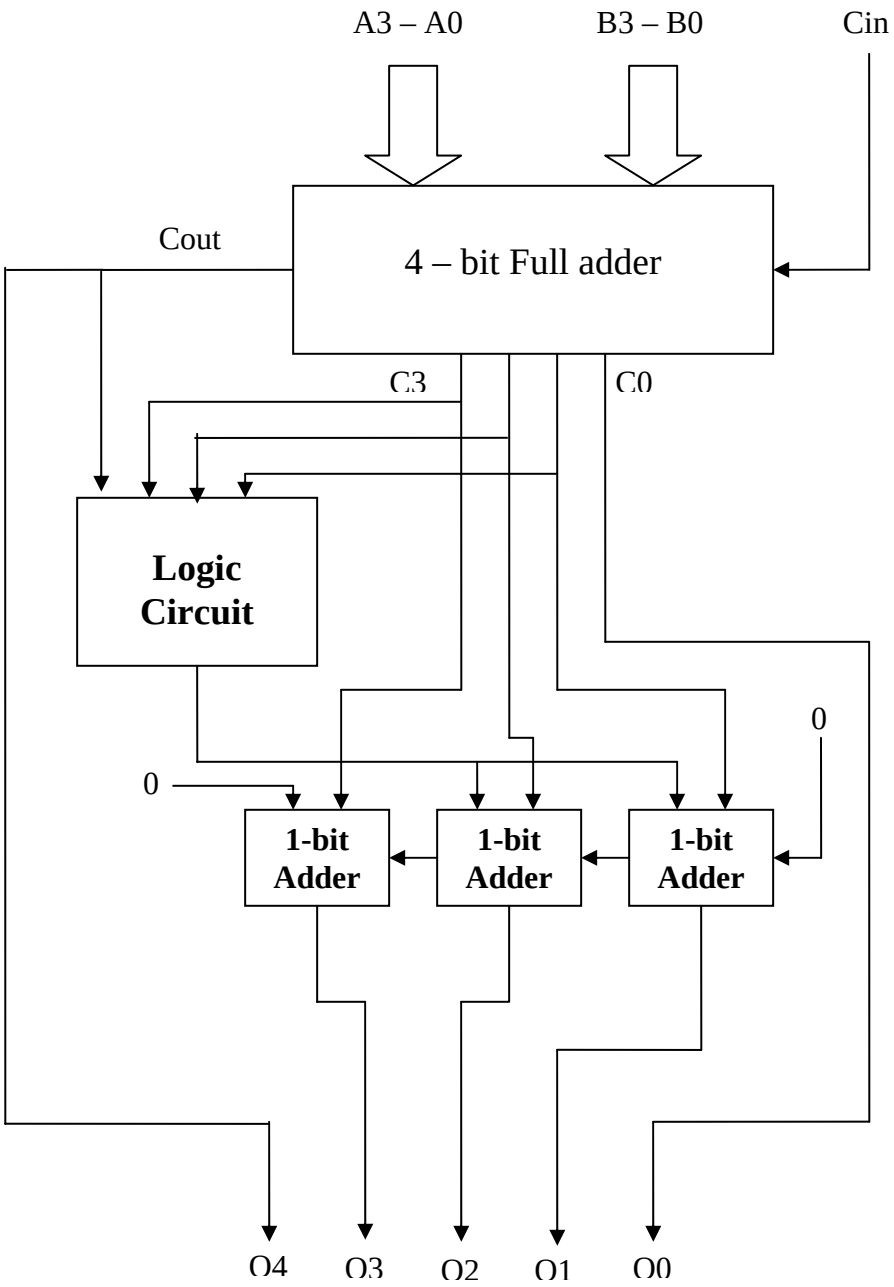


   Implement the 1 – bit Full Adder using the following models and write testbench for all
   (A) Gate level model
   (B) Data flow model and
   (C) Behavioral model
2. Design a 4-bit adder by using 1-bit full adder. Run your code with test bench.
3. Design and simulate a 4-bit universal shift register by using D flip flop as shown in Figure 1. Write a test bench to verify the design. The Universal shift register function is defined by the given table.

S1 S0	Action
0 0	Previous Output
0 1	Shift Right
1 0	Shift Left
1 1	Parallel Load



4. Design a 4 – bit BCD adder as shown in figure below. Run the code with a test bench.



4 – bit BCD Adder