Activity 02: Modeling and Implementation of Combinational Circuits in Verilog

Objective

Understand Verilog syntax, get familiar with Xilinx ISE Simulator, and test/implement combinational circuits on Nexys 3 FPGA board.

Theory

Verilog is a Hardware Description Language; a textual format for describing electronic circuits and systems. Verilog HDL allows designers to design at various levels of abstraction. The main components of Verilog are as follows.

1. Module:

   A **module** is the basic building block in Verilog. A digital design coded in Verilog consists of one or several modules. The contents of a module are also critical from synthesis perspective. **Modules** are declared and instantiated like classes in C++, but modules declarations cannot be nested. These instances of low-level modules are interconnected. **Modules** have **ports** for these interconnections. Modules start with keyword **module** and end with keyword **endmodule**. The Ports of a module can be **input**, **output** or **inout**. A very simple example of a concept of a module is illustrated as follows.

2. Nets:

   Nets are physical connections between components. Though many types of nets are defined in Verilog. A variable of type **wire** can only be assigned a value or expression once i.e. it appears only once on LHS in the entire design. A wire can be used multiple times in the logic i.e. it can appear multiple times in RHS expressions. This variable is usually an output of logic and always shows the logic value of the driving components. A wire infers a physical wire once synthesized.

3. Registers:

   A **register** type variable is denoted by **reg**. Register variables are used for implicit storage meaning that values should be written on these variables and unless variable is modified it retains previously assigned value. It is important to note that a variable of type register does not necessarily imply a hardware register and it may infer a physical wire once synthesized.

Levels of abstraction

Verilog is a hardware description language. The HW can be described at several levels of details. To capture these details Verilog provides the designer the following four levels of abstractions:

1. Switch level (not covered in this lab)
2. Gate level
3. Dataflow level
4. Behavioral or algorithmic level

1. Gate level

At this abstraction level, a digital circuit is designed using gates such as AND, OR, NOT, NOT, XOR etc. While such modeling is intuitive and efficient for smaller circuits, gate level modeling does not scale well for larger circuits.

2. Dataflow level

Dataflow modeling is a higher level of abstraction compared to gate level modeling. To design a circuit in this abstraction level the designer should be aware of data flow of the design. The gate level modeling becomes very complex for a VLSI circuit; hence dataflow modeling became a very important way of implementing the design. In dataflow modeling most of the design is implemented using continuous assignments, which are used to drive a value onto a net. The continuous assignments are made using the keyword assign.

3. Behavioral level

Behavioral level is the highest level of abstraction in Verilog. This level is characterized with the provision of high-level language constructs. This level contains constructs like for loop, while, repeat, if-else, and case etc.

In behavioral or algorithmic level, all the behavioral statements are enclosed in a procedural block. Variables used in LHS of all these statements must be declared as of type reg. The LHS variables used in the expression may be reg or wire type. There are two types of procedural blocks: always and initial

Practical:

Activity 2a: Half adder

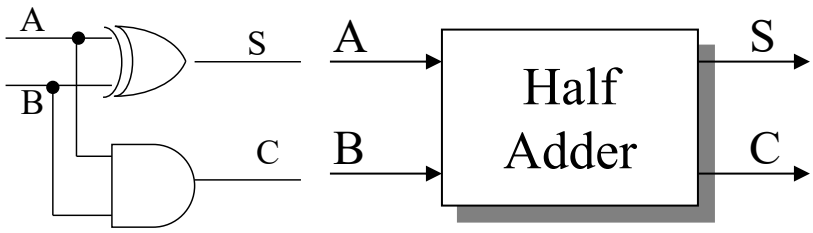
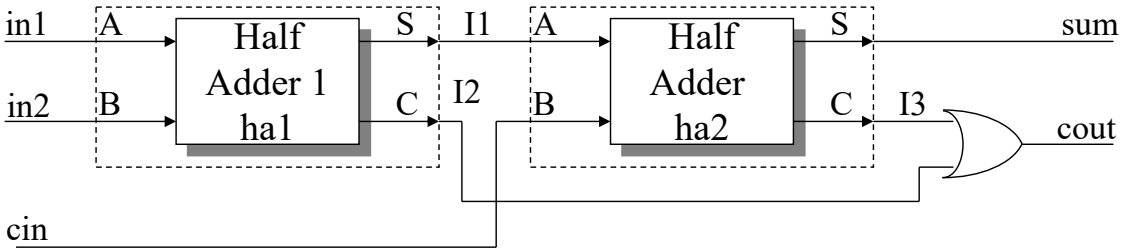


Figure 1

Implement a half adder, as illustrated in Figure 1, in Verilog HDL using gate level abstraction. Write a test bench to verify the functional behavior of the adder for all possible input combinations. Simulate the testbench using Xilinx ISE Simulator. Show the simulation waveforms to TA/LE and include a snapshot in the lab report.

Activity 2b: Full Adder



Use two instances of a half adder and an OR gate to implement a full adder, as illustrated in Figure 2, in Verilog HDL. Write a test bench to verify the functional behavior of the full adder for all possible input combinations. Simulate the testbench using Xilinx ISE Simulator. Show the simulation waveforms to TA/LE and include a snapshot in the lab report.