Introduction to Digital Design
Using Digilent FPGA Boards
— Block Diagram / Verilog Examples

Richard E. Haskell
Darrin M. Hanna

Oakland University, Rochester, Michigan

NOTE: For the FPGA Labs you will NOT be using the Altec Active HDL software. Instead, you will be using the Xilinx tool chain as described in the Screen-shot document.

LBE Books
Rochester Hills, MI

Example 2

2-Input Gates

In this example we will design a circuit containing six different 2-input gates. ~~Example 2a will show the simulation results using Aldec Active-HDL and Example 2b will show how to synthesize the program to a Xilinx FPGA on a Digilent board.~~

Prerequisite knowledge:

Appendix C – Basic Logic Gates
~~Appendix A – Use of Aldec Active-HDL~~

2.1 Generating the Design File *gates2.bde*

~~Part 4 of the tutorial in Appendix A shows how to connect two inputs *a* and *b* to the inputs of six different gates using the block diagram editor (BDE) in Active-HDL~~. The result is shown in Fig. 2.1. Note that we have named the outputs of the gates the name of the gate followed by an underscore. Identifier names in Verilog can contain any letter, digit, underscore _, or $. The identifier can not begin with a digit or be a keyword. Verilog is *case sensitive*.

The name of this file is *gates2.bde*. When you compile this file the Verilog program *gates2.v* shown in Listing 2.1 is generated. We have modified the module statement to conform to the 2001 Verilog standard as described in Example 1.

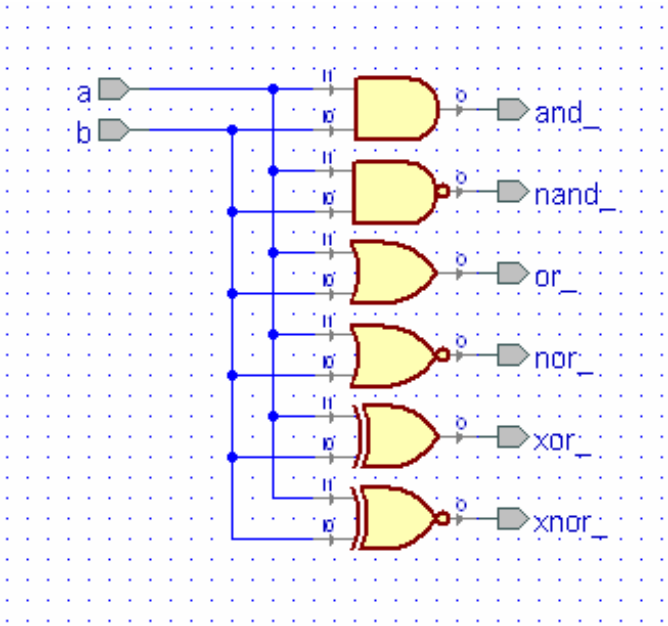

Figure 2.1 Circuit diagram for Example 2

Listing 2.1 gates2.v

```
// Example 2a: gates2
module gates2 (
input wire a,
input wire b,
output wire and_,
output wire nand_,
output wire nor_,
output wire or_,
output wire xnor_,
output wire xor_
) ;

assign and_ = b & a;
assign nand_ = ~(b & a);
assign or_ = b | a;
assign nor_ = ~(b | a);
assign xor_ = b ^ a;
assign xnor_ = ~(b ^ a);

endmodule
```

The logic diagram in Fig. 2.1 contains six different gates. This logic circuit is described by the Verilog program shown in Listing 2.1. The first line in Listing 2.1 is a comment. Comments in Verilog follow the double slash //. All Verilog programs begin with a *module* statement containing the name of the module (*gates2* in this case) followed by a list of all input and output signals together with their direction and type. We will generally use lower case names for signals. The direction of the input and output signals is given by the Verilog statements *input*, *output*, or *inout* (for a bi-directional signal). The type of the signal can be either *wire* or *reg*. In Listing 2.1 all of the signals are of type *wire* which you can think of as a wire in the circuit in Fig. 2.1 where actual voltages could be measured. We will describe the *reg* type in Example 5.

To describe the output of each gate in Fig. 2.1 we simply write the logic equation for that gate preceded by the keyword *assign*. These are *concurrent* assignment statements which means that the statements can be written in any order.

2.2 Simulating the Design *gates2.bde*

~~Part 4 of the tutorial in Appendix A shows how to simulate this Verilog program using Active-HDL.~~ The simulation produced in Appendix A is shown in Fig. 2.2. Note that the waveforms shown in Fig. 2.2 verify the truth tables for the six gates. Also note that two clock stimulators were used for the inputs *a* and *b*. By making the period of the clock stimulator for the input *a* twice the period of the clock stimulator for the input *b* all four combinations of the inputs *a* and *b* will be generated in one period of the input *a*.

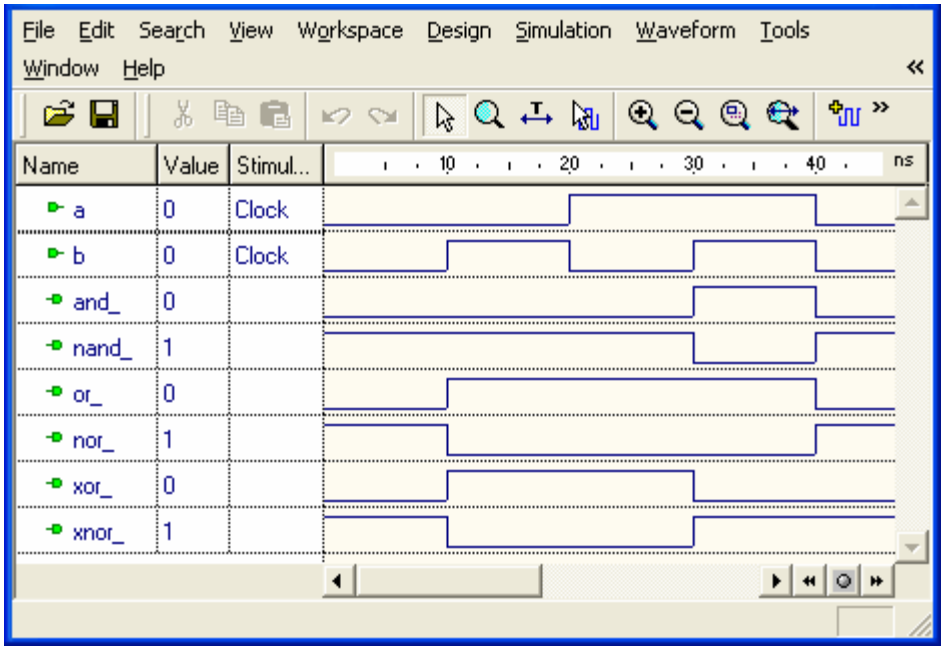

Figure 2.2 Simulation of logic circuit in Fig. 2.1

2.3 Generating a Top-Level Design

Part 5 of the tutorial in Appendix A shows how to create a top-level design for the *gates2* circuit. In order to use the constraint files *basys2.ucf* or *nexys2.ucf* described in Example 1 we must name the switch inputs *sw*[*i*] and the LED outputs *ld*[*i*]. This top-level design, as created in Part 5 of Appendix A is shown in Fig. 2.3. The module *gates2* in Fig. 2.3 contains the logic circuit shown in Fig. 2.1. Note that each wire connected to a bus must be labeled to identify its connection to the bus lines.

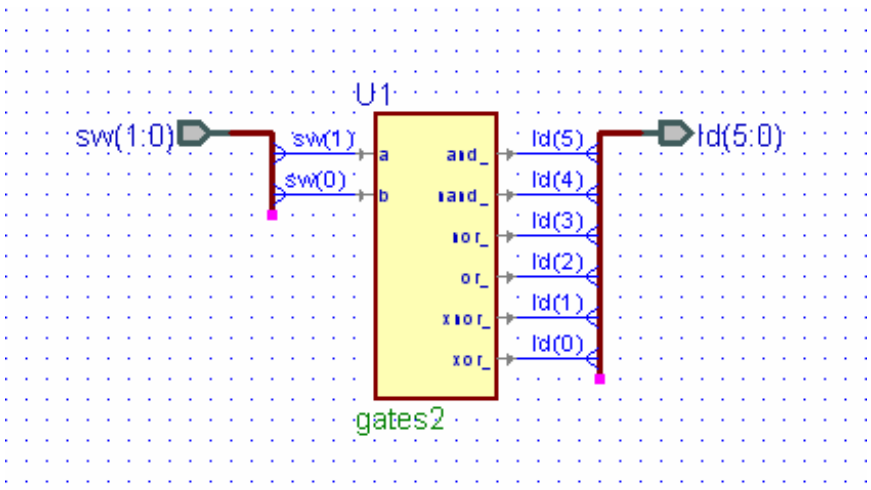

Figure 2.3 Top-level design for Example 2

Compiling the top-level design shown in Fig. 2.3 will generate the Verilog program shown in Listing 2.2. The inputs are now the two rightmost slide switches, *sw*[1:0], and the outputs are the six right-most LEDs *ld*[5:0]. To associate these inputs and outputs with the inputs *a* and *b* and the six output in the *gates2* module in Fig. 2.1 and Listing 2.1 we use the Verilog instantiation statement

```
gates2 U1
(     .a(sw[1]),
      .and_(ld[5]),
      .b(sw[0]),
      .nand_(ld[4]),
      .nor_(ld[3]),
      .or_(ld[2]),
      .xnor_(ld[1]),
      .xor_(ld[0])
);
```

This Verilog instantiation statement begins with the name of the module being instantiated, in this case *gates2* from Listing 2.1. This is followed by an arbitrary name for this module in the top-level design. Here we call it U1. Then in parentheses the inputs and outputs in Listing 2.1 are associated with corresponding inputs and outputs in the top-level design in Fig. 2.3. Note that we connect the input *a* in Listing 2.1 to the input *sw*[1] on the FPGA board. The input *b* in Listing 2.1 is connected to *sw*[0] and the outputs *and_*, *nand_*, *or_*, *nor_*, *xor_*, and *xnor_* are connected to the corresponding LED outputs *ld*[5:0].

Follow the steps in the tutorial in Appendix A and implement this design on the FPGA board. Note that when you change the settings of the two right-most slide switches the LEDs will indicate the outputs of the six gates.

Listing 2.2 gates2_top.v

```
// Example 2b: gates2_top
module gates2_top (sw,ld) ;
input wire [1:0] sw;
output wire [5:0] ld;

gates2 U1
(     .a(sw[1]),
      .and_(ld[5]),
      .b(sw[0]),
      .nand_(ld[4]),
      .nor_(ld[3]),
      .or_(ld[2]),
      .xnor_(ld[1]),
      .xor_(ld[0])
);
```