



Sukkur Institute of Business Administration University

Department of Electrical Engineering

ESE-412: Digital System Design Lab

Handout#02: Verilog Data Flow Modelling for Basic Combinational Circuits

Instructor: Dr. Safeer Hyder Laghari

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Lab Learning Objectives:

After completing this session, student should be able to:

- ◆ Use continuous assignment and Verilog HDL operators to design basic combinational circuits

Lab Hardware and Software Required:

1. Xilinx Vivado 2016.2
2. Digilent NEXYS 4 DDR FPGA Board
3. Desktop/Laptop Computer

Background Theory*Data flow modelling:*

As we know that, Gate-level modelling is highly intuitive, however, it is preferable for small circuits as gate count is low and designer can connect and instantiate every gate easily. Data flow modelling provides higher abstraction for design and is effective for more complex designs. As on every coming day, gate density in digital circuits is increasing therefore higher level of design abstractions are beneficial in every aspect.

Dataflow modelling 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. Synthesis tools have been developed and integrated in the computer aided design tools to convert data flow modelling based designs into gate-level designs.

Dataflow modelling is used mostly for describing the Boolean equations of combinational logic. Combinational logic can be designed with truth tables, Boolean equations, and schematics. Currently, automated tools are used to create a gate-level circuit from a dataflow design description. This process is called logic synthesis. Dataflow modelling has become a popular design approach as logic synthesis tools have become sophisticated. This approach allows the designer to concentrate on optimizing the circuit in terms of data flow. For maximum flexibility in the design process, designers typically use a Verilog description style that combines the concepts of gate-level, data flow, and behavioral design. In the digital design community, the term RTL (Register Transfer Level) design is commonly used for a combination of dataflow modelling and behavioral modelling.

Continuous Assignment:

Dataflow modelling uses continuous assignments and the keyword `assign`. A continuous assignment is a statement that assigns a value to a net. The data type family `net` is used in Verilog HDL to represent a physical connection between circuit elements. A net is declared explicitly by a net keyword (e.g., `wire`) or by declaring an identifier to be an input port. The logic value associated with a net is determined by what the net is connected to. If the net is connected to an output of a gate, the net is said to be driven by the gate, and the logic value of the net is determined by the logic values of the inputs to the gate and the truth table of the gate.

If the identifier of a net is the left-hand side of a continuous assignment statement or a procedural assignment statement, the value assigned to the net is specified by a Boolean expression that uses operands and operators. As an example, assuming that the variables were declared, a two-to-one-line multiplexer with scalar data inputs A and B, select input S, and output Y is described with the continuous assignment


```
assign Y = (A & & S) || (B & & S)
```

The relationship between Y, A, B, and S is declared by the keyword `assign`, followed by the target output Y and an equals sign. Following the equals sign is a Boolean expression. In hardware terms, this assignment would be equivalent to connecting the output of the OR gate to wire Y.

Operators:

Verilog HDL provides about 30 different operators. Table 2.1 lists all of these operators, their symbols, and the operation that they perform. It is necessary to distinguish between arithmetic and logic operations, so different symbols are used for each. The plus symbol `1+2` indicates the arithmetic operation of addition; the bitwise logic AND operation (conjunction) uses the symbol `&`. There are special symbols for bitwise logical OR (disjunction), NOT, and XOR. The equality symbol uses two equals signs (without spaces between them) to distinguish it from the equals sign used with the `assign` statement. The bitwise operators operate bit by bit on a pair of vector operands to produce a vector result. The concatenation operator provides a mechanism for appending multiple operands.

Table 2.1 Verilog HDL Operators

Operator Type	Operator Symbol	Operation Performed	Number of Operands
Table 2.2: Verilog operator precedence			
Arithmetic	<code>+ - ! ~ & ~& ~ ^ ~ ^ ^ ~ (unary)</code>	Highest precedence	
	<code>**</code>		
	<code>* / %</code>		
Logical	<code>+ - (binary)</code>		
	<code><< >> <<< >>></code>		
Relational	<code>< <= > >=</code>		
	<code>== != === !==</code>		
	<code>& (binary)</code>		
Equality	<code>^ ^~ ~^ (binary)</code>		
	<code> (binary)</code>		
	<code>&&</code>		
Bitwise	<code> </code>		
	<code>?: (conditional operator)</code>		
	<code>{ } { } { }</code>	Lowest precedence	
Reduction	<code>^~ or ~^</code>	bitwise xnor	two
	<code>&</code>	reduction and	one
	<code>~&</code>	reduction nand	one
	<code> </code>	reduction or	one
	<code>~ </code>	reduction nor	one
	<code>^</code>	reduction xor	one
Shift	<code>^~ or ~^</code>	reduction xnor	one
	<code>>></code>	Right shift	two
Concatenation	<code><<</code>	Left shift	two
	<code>{ }</code>	Concatenation	any number
Replication	<code>{ { } }</code>	Replication	any number
Conditional	<code>?:</code>	Conditional	three

In table 2.2, we can see the precedence table for all the operators in dataflow modelling. The precedencies are shown in the table from highest to the lowest precedence.

Lab Examples:

Example 2.1 – Two-to-Four line decoder with enable using dataflow modelling

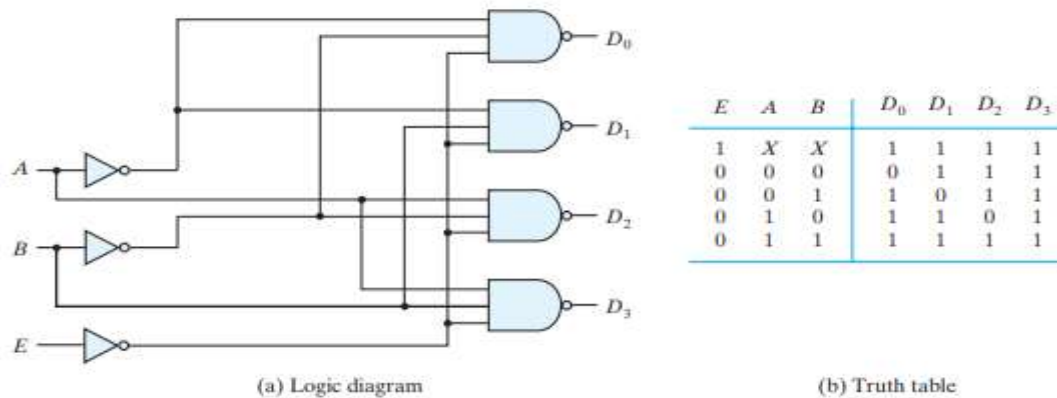


Figure 1 Active-LOW 2-to-4 line decoder with enable input

Listing 2.1 – Active-LOW 2-to-4 line decoder with enable input

```

module decoder_2x4_df (
    output [3:0] D,
    input A, B, enable );

    assign    D[0] = ~((~A) & (~B) & (~enable)),
             D[1] = ~((~A) & B & (~enable)),
             D[2] = ~(A & (~B) & (~enable)),
             D[3] = ~(A & B & (~enable));

endmodule

```

Example 2.2: Four-bit adder using arithmetic and concatenation operators

The dataflow description of the four-bit adder is shown below in Example 2.2 code. The addition logic is described by a single statement using the operators of addition and concatenation. The plus symbol (+) specifies the binary addition of the four bits of A with the four bits of B and the one bit of C_{in}. The target output is the concatenation of the output carry C_{out} and the four bits of Sum. Concatenation of operands is expressed within braces and a comma separating the operands. Thus, {C_{out}, Sum} represents the five-bit result of the addition operation.

Listing 2.2: Four-bit adder

```

module four_bit_adder(
    output [3:0] Sum,
    output      C_out,
    input [3:0] A, B,
    input      C_in );

    assign {C_out, Sum} = A + B + C_in;

endmodule

```

Example 2.3: 4-to-1 Multiplexer using dataflow bitwise operators

Listing 2.3: 4-to-1 multiplexer

```
module mux4_to_1(  
    output out,  
    input i0, i1, i2, i3, s1, s0);  
  
    assign out =      (~s1 & ~s0 & i0) |  
                      (~s1 & s0 & i1) |  
                      (s1 & ~s0 & i2) |  
                      (s1 & s0 & i3);  
  
endmodule
```

Example 2.4: 1 bit comparator using relational and equality operators

Listing 2.3: 1-bit comparator

```
Module mag_comparator(  
    output A_lt_B, A_eq_B, A_gt_B,  
    input A, B);  
  
    assign A_lt_B = (A<B);  
    assign A_gt_B = (A>B);  
    assign A_eq_B = (A==B);  
  
endmodule
```

Lab Exercises:

1. A *magnitude comparator* checks if one number is greater than or equal to or less than another number. A 4-bit magnitude comparator takes two 4-bit numbers *A*, and *B* as input.
 - a. Drive Boolean expressions for the design. Provide the handwritten work.
 - b. Write the Verilog description of the module *magnitude_comparator* using bitwise operators. Provide the codes with name signature.
 - c. Test the design using Verilog test bench. Provide the codes with the name signature, along with tcl console and timing diagram results.
 - d. Make use of comments and indentations while coding.
2. Design 16-bit comparator by cascading method.
 - a. Provide the design schematic
 - b. Implement the design and provide the codes.
 - c. Write down the test bench and provide tcl console and timing diagrams.
 - d. Make use of comments and indentations while coding.

3. A full subtracter has three 1-bit inputs x,y, and z (previous borrow) and two 1-bit outputs D (difference) and B (borrow).
 - a. Draw the truth table for the given circuit.
 - b. Drive the simplified Boolean expression.
 - c. Write the Verilog description for the subtracter module.
 - d. Instantiate the design module in the test module and provide the tcl console and timing diagram outputs.

Detailed readings and codes:

Verilog HDL A Guide to Digital Design and Synthesis by Samir Palnitkar, 2nd Edition, Chapter 6

<https://github.com/SAFEERHYDER/Digital-System-Design>