Multivalued Logic and Signal Resolution

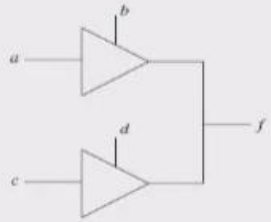
- Till now we have used mostly used two-valued bit logic in our Verilog code.
- In order to represent tristate buffers and buses, it is necessary to be able to rep- resent a third value, *Z*, which represents the high-impedance state.
- It is also at times necessary to have a fourth value, *X*, to represent an unknown state. This unknown state may occur if the initial value of a signal is unknown, or if a signal is simulta- neously driven to two conflicting values, such as 0 and 1. If the input to a gate is *Z*, the gate output may assume an unknown value, *X*.

A 4-Valued Logic System

Signals in a 4-valued logic can assume the four values: *X*, 0, 1, and *Z*, where each of the symbols represent the following:

- *X* Unknown
- 0 0
- 1 1
- *Z* High impedance
- The high impedance state is used for modeling tristate buffers and buses. This unknown state can be used if the initial value of a signal is unknown, or if a signal is simultaneously driven to two conflicting values, such as 0 and 1. Verilog uses the 4-valued logic system by default.

Two tristate buffers with their outputs tied together



. Data type X01Z, which can have the four values *X*, 0, 1, and *Z*, is assumed. The tristate buffers have an active-high output enable, so that when *b* 5 1 and *d* 5 0, *f* 5 *a*; when *b* 5 0 and *d* 5 1, *f* 5 *c*; and when *b* 5 *d* 5 0, the *f* output assumes the high-Z state. If *b* 5 *d* 5 1, an output conflict can occur. Tristate buffers are used for bus management whereby only one active-high output should be enabled to avoid the output conflict.

```
module t_buff_exmpl (a, b, c, d, f);

    input a;
    input b;
    input c;
    input d;
    output f;
    reg f;

    always @(a or b)
    begin : buff1
        if (b == 1'b1)
            f = a ;
        else
            f = 1'bz ; //"drive" the output high Z when not enabled
    end

    always @(c or d)
    begin : buff2
        if (d == 1'b1)
            f = c ;
        else
            f = 1'bz ; //"drive" the output high Z when not enabled
    end
endmodule
```

```
module t_buff_exmpl2 (a, b, c, d, f);
   input a;
    input b;
    input c;
    input d;
    output f;

    assign f = b ? a: 1'bz ;
    assign f = d ? c: 1'bz ;

endmodule
```

The operation of a tristate bus with the 4-valued logic, is specified by the following table:

	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Built-in Primitives

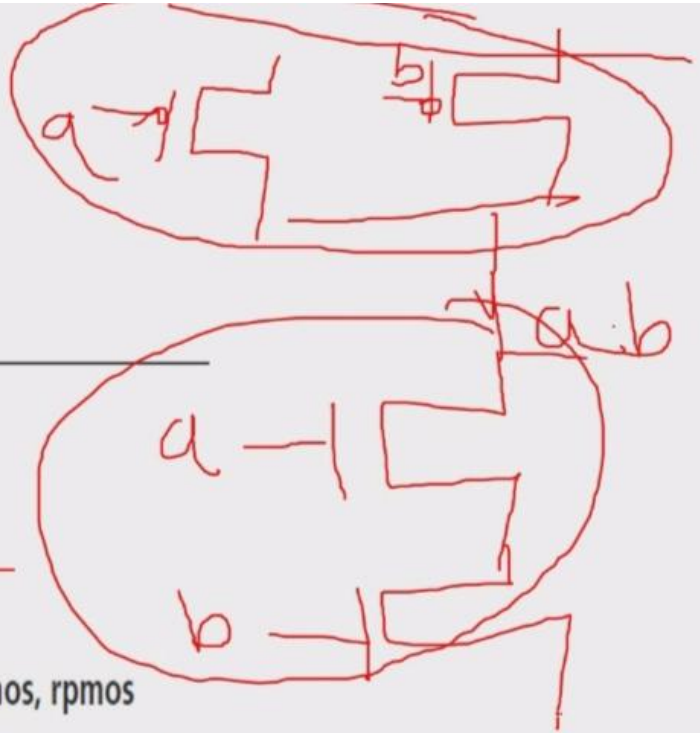
- The focus of this book is on behavioral-level modeling, and hence we focused on Verilog constructs that allow that. However, Verilog allows modeling at switch- level details and has several predefined primitives for that. It also has predefined
- gate-level primitives with drive strength among other things. There are 14 prede- fined logic gate primitives and 12 predefined switch primitives to provide the gate- and switch-level modeling facility. Modeling at the gate and switch level has several advantages:
- Synthesis tools can easily map it to the desired circuitry since gates provide a very close one-to-one mapping between the intended circuit and the model.
- There are primitives such as the bidirectional transfer gate that cannot be otherwise modeled using continuous assignments.
- The Verilog module in Figure 2-7 is defined in Figure 8-7 using built-in primi- tives. The **and** and **or** are built-in primitives with one output and multiple inputs.
- The output terminal must be listed first followed by inputs as in
- **and** (out, in_1, in_2, ..., in_n);

Verilog also provides built-in primitives for tristate gates. The **bufif0** is a non- inverting buffer primitive with active-low control input while **bufif1** has active-high control input. The **notif0** and **notif1** are inverting buffers with active-low and active- high controls, respectively. These primitives can support multiple outputs. The outputs must be listed first followed by input and finally the tristate control input. An array of four inverting tristate buffers can be created as shown in Figure

```
module tri_driver (in, out, tri_en);
    input [3:0] in;
    output [3:0] out;
    input tri_en;

    bufif0 buf_array[3:0] (out, in, tri_en); // array of three-state buffers
endmodule
```

Built-in Primitive Type	Primitives
n-input gates	and, nand, nor, or, xnor, xor
n-output gates	buf, not
Three-state gates	bufif0, bufif1, notif0, notif1
Pull gates	pulldown, pullup
MOS switches	cmos, nmos, pmos, rcmos, rnmos, rpmos
Bidirectional Switches	rtran, rtranif0, rtranif1, tran, transif0, tranif1



- The built-in primitives can have an optional delay specification. A delay spec- ification can contain up to three delay values, depending on the gate type. For a three-delay specification,
- the first delay refers to the transition to the rise delay (i.e., transition to 1),
- the second delay refers to the transition to the fall delay (i.e., transition to 0), and
- the third delay refers to the transition to the high-impedance value (i.e., turn-off).

If only one delay is specified, it is rise delay. If there are only two delays specified, they are rise delay and fall delay. If turn-off delay must be specified, all three delays must be specified. The **pullup** and **pulldown** instance declarations must not include delay specifications.

- The following are examples of built-in primitives with one, two, and three delays:

```
and #(10) a1 (out, in1, in2);                 // only one delay
                                              // (so rise delay=10)
and #(10,12) a2 (out, in1, in2);              // rise delay=10 and
                                              // fall delay=12
bufif0 #(10,12,11) b3 (out, in, ctrl);        // rise, fall, and
                                              // turn-off delays
```

User-Defined Primitives

- The predefined gate primitives in Verilog can be augmented with new primitive elements called user-defined primitives (UDPs). UDPs define the functionality of the primitive in truth table or state table form. Once these primitives are specified by the user, instances of these new UDPs can be created in exactly the same manner as built-in gate primitives are instantiated. While the built-in primitives are only combinational, UDPs can be combinational or sequential.
- A *combinational UDP* uses the value of its inputs to determine the next value of its output. A *sequential UDP* uses the value of its inputs and the current value of its output to determine the value of its output. Sequential UDPs provide a way to model sequential circuits such as flip-flops and latches. A sequential UDP can model both level-sensitive and edge-sensitive behavior

- A UDP can have multiple input ports but has exactly one output port. Bidirectional inout ports are not permitted on UDPs. All ports of a UDP must be scalar—that is, vector ports are not permitted. Each UDP port can be in one of three states: 0, 1, or X. The tristate or high-impedance state value Z is not supported. If Z values are passed to UDP inputs, they shall be treated the same as X values. In sequential UDPs, the output always has the same value as the internal state.
- A UDP begins with the keyword **primitive**, followed by the name of the UDP. The functionality of the primitive is defined with a truth table or state table, starting with the keyword **table** and ending with the keyword **endtable**. The UDP definition then ends with the keyword **endprimitive**. The truth table for a UDP consists of a section of columns, one for each input followed by a colon and finally the output column. The multiplexer we defined in Figure 2-36 using continuous assign state- ments is redefined as a UDP in Figure

- Designers can create higher-level from the lower-level models
- This is the lowest level of abstraction
- Gate Level modelling is also called structural model
- However, detail entry of coding is time consuming.
- Verilog HDL has gate primitives for all basic gates.
- Gate primitives are predefined in Verilog, which are ready to use. They are instantiated like modules.
- There are two classes of gate primitives: Multiple input gate primitives and Single input gate primitives.
- Multiple input gate primitives include **and, nand, or, nor, xor, and xnor.**
- Gate Level(Structural netlist) is appropriate where components are available, just need to interconnect.
- Single input gate primitives include **not, buf, notif1, bufif1, notif0, and bufif0.**

```
not not_1 (out, in);              //Inverting gate
buf buf_1 (out0, out1, in);       //two output buffer gate
notif1 notif1_1 (out, in, ctrl);  //tristate inverter
```

- Array of instances:

```
wire [3:0] out, in0, in1;                 //signal declaration
and and_array[3:0] (out, in0, in1); //4 'and' gate instantiation for two 4 bit vector inputs
```