Cloud FPGA

bit.ly/cloudfpga
Lecture: User-Defined Primitives

Prof. Jakub Szefer
Dept. of Electrical Engineering, Yale University

EENG 428 / ENAS 968
Cloud FPGA
This lecture is mostly based on contents of Chapter 9, from “The Verilog Hardware Description Language” book [1], 5th edition. Example figures and (modified) code are from the textbook unless otherwise specified.

**Topics covered:**

- User-defined primitives
- Combinatorial primitives
- Sequential primitives
User-Defined Primitives
Gate Level Primitives

- Verilog provides a set of 26 built-in gate level primitives:

<table>
<thead>
<tr>
<th>n_input gates</th>
<th>n_output gates</th>
<th>tristate gates</th>
<th>pull gates</th>
<th>MOS switches</th>
<th>bidirectional switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>buf</td>
<td>bufif0</td>
<td>pullup</td>
<td>nmos</td>
<td>tran</td>
</tr>
<tr>
<td>nand</td>
<td>not</td>
<td>bufif1</td>
<td>pulldown</td>
<td>pmos</td>
<td>tranif0</td>
</tr>
<tr>
<td>nor</td>
<td>notif0</td>
<td>cmos</td>
<td></td>
<td></td>
<td>tranif1</td>
</tr>
<tr>
<td>or</td>
<td>notif1</td>
<td>rmos</td>
<td></td>
<td></td>
<td>rtran</td>
</tr>
<tr>
<td>xor</td>
<td></td>
<td>rpmos</td>
<td></td>
<td></td>
<td>rtranif0</td>
</tr>
<tr>
<td>xnor</td>
<td></td>
<td>rcmos</td>
<td></td>
<td></td>
<td>rtranif1</td>
</tr>
</tbody>
</table>

- Potential need for other primitives:
  - Compact representation of possibly arbitrary logic
  - Reduce the pessimism with respect to the undefined x value
  - Better simulation speed by modeling more complex operations using single primitive

- Typically, custom primitives cannot be used of synthesis to hardware (FPGA or ASIC)
Combinatorial Primitives

- User-defined primitives are defined using the `primitive` keyword and a `table`:
  - Similar to a module in structure, but actually are more like a function

- Rules for user-defined primitives:
  - Primitives can have multiple input ports, but only one output port
  - They may not have bidirectional inout ports
  - The output port must be the first port in the port list
  - All primitive ports are scalar, and no vector ports are allowed
  - Only logic values of 1, 0, and `x` are allowed on input and output

```plaintext
primitive carry  
(output carryOut,  
 input carryIn, aIn, bIn  
);

table  
// inputs : output  
  0 0 0 : 0;  
  0 0 1 : 0;  
  0 1 0 : 0;  
  0 1 1 : 1;  
  1 0 0 : 0;  
  1 0 1 : 1;  
  1 1 0 : 1;  
  1 1 1 : 1;  
endtable
endprimitive
```
The combinatorial primitive description can be expanded with use of undefined values $x$ and don't care values:

```cpp
primitive carryX
{
    output carryOut,
    input carryIn, aIn, bIn
};

table
    // inputs : output
    0 0 0 : 0;
    0 0 1 : 0;
    0 1 0 : 0;
    0 1 1 : 1;
    1 0 0 : 0;
    1 0 1 : 1;
    1 1 0 : 0;
    1 1 1 : 1;
    0 0 x : 0;
    0 x 0 : 0;
    x 0 0 : 0;
    1 1 x : 1;
    x 1 1 : 1;
endtable
endprimitive
```

```cpp
primitive carryAbbrev
{
    output carryOut,
    input carryIn, aIn, bIn
};

table
    // inputs : output
    0 0 ? : 0;
    0 ? 0 : 0;
    ? 0 0 : 0;
    ? 1 1 : 1;
    1 ? 1 : 1;
    1 1 ? : 1;
endtable
endprimitive
```

Use $x$ to match undefined inputs
Use $? to match for don't care values, don't care is 0, 1, or $x$
LUT Modules in Xilinx

- The **primitives** may be similar and/or confusing with LUT modules
- Xilinx allows users to directly specify values store in Look-Up Tables in the FPGA fabric
  - Users can write any values, effectively creating their own custom primitives
  - Many Look-Up Tables modules are available, one example is the LUT6:

```c
// LUT6: 6-input Look-Up Table with general output
// 7 Series
LUT6 #(
  .INIT(64’h0000000000000000) // Specify LUT Contents
)
LUT6_inst
(
  .0(0), // LUT general output
  .I0(I0), // LUT input
  .I1(I1), // LUT input
  .I2(I2), // LUT input
  .I3(I3), // LUT input
  .I4(I4), // LUT input
  .I5(I5) // LUT input
);
// End of LUT6_inst instantiation
```

- More in future lecture on Series 7 devices…
Sequential Primitives

- User-defined primitives may be used to describe sequential devices which exhibit level- and edge-sensitive properties
  - They have internal state that, modeled with a register variable, and a state column used to specifying the behavior of the primitive
  - Effectively model the primitive as a state machine

- Textbook example of level-sensitive latch:

```verbatim
class latch
{
  output reg q,
  input clock, data
);
  table
    // inputs : state : output
    0 1 : ? : 1;
    0 0 : ? : 0;
    1 ? : ? : -;
endtable
endprimitive
```

- Use `?` to match for don't care values, don't care is 0, 1, or x
- Can also use `x` to match undefined inputs
- Use `-` to denote that the output doesn't change
- Output is a register, like in sequential logic
- Specify operation as a table: inputs, current state, next state (it is also the output)
Sequential Primitives

• For edge-sensitive primitives, the input clock needs to specify the edge
  • Positive or negative edge
  • Can include undefined and don’t care values

• D Flip-Flop form textbook:

```plaintext
primitive dEdgeFF
(
  output reg q,
  input clock, data
);

table
  // inputs : state : output
  (01) 0 : ? : 0;
  (01) 1 : ? : 1;
  (0x) 1 : 1 : 1;
  (0x) 0 : 0 : 0;
  (?0) ? : ? : -;
  (? (??)) ? : ? : -;
endtable
endprimitive

primitive dEdgeFFShort
(
  output reg q,
  input clock, data
);

table
  // inputs : state : output
  r 0 : ? : 0;
  r 1 : ? : 1;
  (0x) 0 : 1 : 1;
  (0x) 1 : 1 : 1;
  (?0) ? : ? : -;
  (? * : ? : -;
endtable
endprimitive
```

Table of the short-hand notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Interpretation</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Logic 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Logic 1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>Iteration of 0, 1, and x</td>
<td>Cannot be used in output field</td>
</tr>
<tr>
<td>b</td>
<td>Iteration of 0 and 1</td>
<td>Cannot be used in output field</td>
</tr>
<tr>
<td>-</td>
<td>No change</td>
<td>May only be given in the output field of a sequential primitive</td>
</tr>
<tr>
<td>(vw)</td>
<td>Change of value from v to w</td>
<td>v and w can be any one of 0, 1, x, or b</td>
</tr>
<tr>
<td>*</td>
<td>Same as (??)</td>
<td>Any value change on input</td>
</tr>
<tr>
<td>r</td>
<td>Same as (01)</td>
<td>Rising edge on input</td>
</tr>
<tr>
<td>f</td>
<td>Same as (10)</td>
<td>Falling edge on input</td>
</tr>
<tr>
<td>p</td>
<td>Iteration of (01), (0x), and (x1)</td>
<td>Positive edge including x</td>
</tr>
<tr>
<td>n</td>
<td>Iteration of (10), (1x), and (x0)</td>
<td>Negative edge including x</td>
</tr>
</tbody>
</table>
References