Verilog

Logic Synthesis
- Verilog and VHDL started out as simulation languages but soon programs were written to automatically convert Verilog code into low-level circuit descriptions (netlists)
- Synthesis converts Verilog (or other HDL) descriptions to a logic mapping by using technology-specific primitives:
  * For FPGAs: LUTs, flip-flops, and BRAMs
  * For ASICs: standard cells and memory macros
- In addition, synthesis algorithms optimize the implementation for delay and power

Verilog Modules and Instantiation
- Modules define circuit components.
- Instantiation defines hierarchy of the design.
- Example:
  ```verilog
  module addr_cell (a, b, cin, s, cout);
  input     a, b, cin;
  output    s, cout;
  endmodule
  ```
  * Name: `addr_cell`
  * Port list: `a, b, cin, s, cout`
  * Similar to C, but without a call and return mechanism.

Structural Verilog
- `module xor_gate (out, a, b );`
- `input a, b;`;
- `output out;`;
- `wire aBar, bBar, t1, t2;`
- `not invA (aBar, a);`
- `not invB (bBar, b);`
- `and and1 (t1, a, bBar);`
- `and and2 (t2, b, aBar);`
- `or  or1 (out, t1, t2);`
- Notes:
  * The instantiated gates are not “executed”. They are active always.
  * xor gate already exists as a built-in, so there’s no need to define it.
  * Undeclared variables assumed to be wires. Don’t let this happen to you!
/* 2-input multiplexer in gates */
module mux2 (in0, in1, select, out);
input in0, in1, select;
output out;
wire s0, w0, w1;
not (s0, select);
and (w0, s0, in0),
(w1, select, in1);
or (out, w0, w1);
endmodule // mux2

module mux4 (in0, in1, in2, in3, select, out);
input in0, in1, in2, in3;
input [1:0] select;
output out;
wire w0, w1;
mux2
m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
m3 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4

Aside: Netlists
- **Netlist** is a description of the connectivity of an electronic circuit
- A netlist consists of a list of components in a circuit and a list of the nodes they are connected to. A wire (net) connects two or more components
- Structural Verilog is one form of describing a netlist
- Most common netlist format is EDIF (Electronic Design Exchange Format)
  - Established in 1985, still in use

Simple Behavioral Model

module foo (out, in1, in2);
input         in1, in2;
output        out;
assign out = in1 & in2;
endmodule

Short-hand for explicit instantiation of bit-wise "and" gates (in this case).

Example - Ripple Adder

module FullAdder(a, b, ci, r, co);
input a, b, ci;
output r, co;
assign r = a ^ b ^ ci;
assign co = a & ci | a & b | b & ci;
endmodule

module Adder(A, B, R);
input [3:0] A;
input [3:0] B;
output [4:0] R;
wire c1, c2, c3;
FullAdder
add0(.a(A[0]), .b(B[0]), .ci(1),   .co(c1),   .r(R[0]) ),
add1(.a(A[1]), .b(B[1]), .ci(c1),   .co(c2),   .r(R[1]) ),
add2(.a(A[2]), .b(B[2]), .ci(c2),   .co(c3),   .r(R[2]) ),
add3(.a(A[3]), .b(B[3]), .ci(c3),   .co(R[4]), .r(R[3]) );
endmodule

Verilog Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Logical negation</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Logical exclusive or</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left shift</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right shift</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>==</td>
<td>Equality</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal</td>
</tr>
<tr>
<td>(</td>
<td>Open parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>Close parenthesis</td>
</tr>
<tr>
<td>[</td>
<td>Open index brackets</td>
</tr>
<tr>
<td>]</td>
<td>Close index brackets</td>
</tr>
<tr>
<td>{</td>
<td>Open range brackets</td>
</tr>
<tr>
<td>}</td>
<td>Close range brackets</td>
</tr>
</tbody>
</table>

Example:

```
module FullAdder(a, b, ci, r, co);
input a, b, ci;
output r, co;
assign r = a ^ b ^ ci;
assign co = a & ci | a & b | b & ci;
endmodule
```
Verilog Numbers

Constants:
14 ordinary decimal number
-14 2’s complement representation
12’b0000_0100_0110 binary number ("_" is ignored)
12’b046 hexadecimal number with 12 bits

Signal Values:
By default, values are unsigned
12’b0000_0100_0110 wire signed [31:0] x;
By default, values are unsigned
12’b0000_0100_0110

Continuous Assignment Examples

Common same value

Non-Continuous Assignments

Always Blocks
Always blocks give us some constructs that are impossible or awkward in continuous assignments.

Case statement example:

Review – Ripple-Carry Adder Example

Verilog and SystemVerilog Data Types

Verilog Assignment Types

Verilog and SystemVerilog Data Types

Verilog Assignments Examples

Always Blocks

Module and_or_gate (out, in1, in2, in3);
input in1, in2, in3;
output out;
wire [1:0] select;
wire r, co;
input a, b, ci;
output out;
reg [1:0] c;
assign out = a ^ b ^ ci;
assign co = a & ci + a & b + b & cin;

SystemVerilog logic – generic data type
net – can have multiple drivers
High-level primitives enable direct synthesis of behavioral descriptions (functions such as additions).

Modularity is essential to the success of large designs.

Example - Ripple Adder Generator
Parameters give us a way to generalize our designs. A module becomes a "generator" for different variations. Enables design/module reuse. Can simplify testing.

\[
\text{Declaration of the ALU Module:}
\]

\[
\text{module alu}(a, b, f, r);\]
\[
\text{input [31:0] a, b; input [2:0] f; output [31:0] r;}
\]
\[
\text{endmodule}
\]

\[
\text{Example: A 32-bit ALU}
\]

Function Table

<table>
<thead>
<tr>
<th>F2</th>
<th>F1</th>
<th>F0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A - B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A - 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A + 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>A * B</td>
</tr>
</tbody>
</table>

Defining Processor ALU in 5 mins

- Modularity is essential to the success of large designs
- High-level primitives enable direct synthesis of behavioral descriptions (functions such as additions, subtractions, shifts (< and >>), etc.)

Example - Ripple Adder Generator

Parameters give us a way to generalize our designs. A module becomes a "generator" for different variations. Enables design/module reuse. Can simplify testing.

\[
\text{Declaration of the ALU Module:}
\]

\[
\text{module alu}(a, b, f, r);\]
\[
\text{input [31:0] a, b; input [2:0] f; output [31:0] r;}
\]
\[
\text{endmodule}
\]

More on Generate Loop

Permits variable declarations, modules, user-defined primitives, gate primitives, continuous assignments, initial blocks and always blocks to be instantiated multiple times using a for-loop.

```
// Gray-code to binary-code converter
module gray2bin1 (bin, gray);
    // Gray-code to binary-code converter
    // this is a magnitude multiplier
    output [31:0] prod;
    input [15:0] i0,i1;
    // signed arithmetic later
    assign prod = i0 * i1;
    endmodule
```

More on Generate Loop

Permits variable declarations, modules, user-defined primitives, gate primitives, continuous assignments, initial blocks and always blocks to be instantiated multiple times using a for-loop.

```
// Gray-code to binary-code converter
module gray2bin1 (bin, gray);
    // Gray-code to binary-code converter
    // this is a magnitude multiplier
    output [31:0] prod;
    input [15:0] i0,i1;
    // signed arithmetic later
    assign prod = i0 * i1;
    endmodule
```

Top-Level ALU Declaration

- Given submodules:
  
  \[
  \text{module max32two}(l, r, sel);\]
  \[
  \text{input [31:0] l, r; input [1:0] sel; output [31:0] out;}
  \]
  \[
  \text{endmodule}
  \]

- Declaration of the ALU Module:
  
  \[
  \text{module alu}(a, b, f, r);\]
  \[
  \text{input [31:0] a, b; input [2:0] f; output [31:0] r;}
  \]
  \[
  \text{endmodule}
  \]

Latches and Flip-Flops

- Flip-flop is edge-triggered, latch is level-sensitive
  
  \[
  \text{D Flip-flop:}
  \]
  \[
  \text{\hspace{1cm} Rising edge}
  \]
  \[
  \text{\hspace{1cm} Signifies 'edge triggered'}
  \]

- D Latch
  
  \[
  \text{\hspace{1cm} Transparent}
  \]
  \[
  \text{\hspace{1cm} HIGH}
  \]
  \[
  \text{\hspace{1cm} Level sensitive if there is no 'edge'}
  \]

Sequential Logic, Take 2

---

**Note:** this is not a port. Acts like a (unique) names wire/regs in module alu
Timing

- Combinational logic timing

A is arriving late (is in the critical path)

B is arriving late (is in the critical path)

HL and LH transition differ

tA-Out and tB-Out differ

In CMOS, propagation delay depends on:
- Gate type, size (output resistance)
- Capacitive loading
- Input slope

Timing

- Flip-flop timing
  (latch timing will be covered later)

Setup and hold times
- Data cannot change in the interval of setup time before the clock edge to hold time after the clock edge

Register

- 4-bit register

* Accumulator

Summary

- Verilog is the most-commonly used HDL
- We have seen combinatorial constructs
  - Assign statement
  - Always blocks
  - Sequential – next week
- Practice is the best way to learn a new language...