EE345

Intro Verilog

What's an HDL?

- Hardware Description Language
  - Use text to describe hardware
  - Alternative to schematic capture
  - Top HDL's: VHDL and Verilog
    - Some SystemC also
- VHDL
  - Looks like Ada programming language
- Verilog
  - Looks like 'C' programming language
- System C
  - Built on C++ programming language

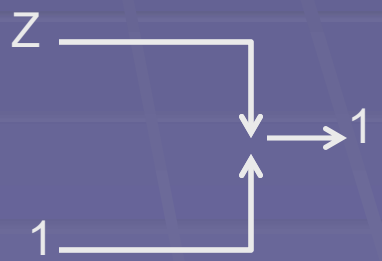
Verilog

- In my experience, Verilog is more common than VHDL for CPU design
- We'll use the latest variant: SystemVerilog
- IEEE standard 1800 (published 2012)
  - http://standards.ieee.org/getieee/1800/download/1800-2012.pdf
- This is a "big" language
  - Extends Verilog (IEEE 1364, 2005)
  - We'll only be using parts of it

Verilog Values and Expressions

Verilog Logic Values

- Four-value logic
  - 0: logic zero
  - 1: logic one
  - Z: high-impedance (float)
  - X: contention or uninitialized
- These combine intuitively



Single-bit or Multi-bit Logic Values

- "logic" is the 4-value type
  - To "declare" is to name something, assign a type

```
          // This is a comment
logic output;           // Single-bit "output"
logic [7:0] counter;    // 8-bit "counter"
```

- Verilog code will determine what the names do
  - Flop, wire, etc.
  - We'll see this later

Values

- **Single-bit: just do it**
  - **X, Z, 0, 1**

```
output = 1;   // drive ‘1’ onto output
triSig = Z;   // Float triSig
```

- **Multi-bit: can specify length and radix**

```
bus = 32’bZ;       // 32 bits of Z,‘b’ means binary
value = 8’hA5;     // 8b, hex
```

Expressions

- There are *many* operators
  - We typically just use a handful of them
- The usual math expressions are available

```
x = x + 2;          // add 2 to x
abc = def * 4;      // def multiplied by 4
```

- Note that you can imply a lot of hardware
  - E.g., * operator: may turn into a multiplier
  - Think about what will “come out”

Expressions: Bitwise

- ~ (negation)

```
logic [7:0] x;
x = 8'b00110101;
x = ~x;      // x gets 11001010
```

- &, |, ^ (and, or, exclusive-or)

```
logic [7:0] x, y, z;
x = 8'b11000011;
y = 8'b00000010;
z = x & y;          // z gets 00000010
z = x | y;          // z gets 11000011
z = x ^ y;          // z gets 11000001
```

Expressions: Relational

```
logic rel;

rel = A == B;   // rel is '1' if A equals B
rel = A != B;   // rel is '1' if A not equal B
rel = A < B;    // rel is '1' if A less than B
…
```

- Equality (and inequality) are cheaper
  - Basically XOR or wide gate
    - E.g., “x == 0” is a NOR gate
- Less-than (etc) can be more expensive

Expressions: Conditional Operator

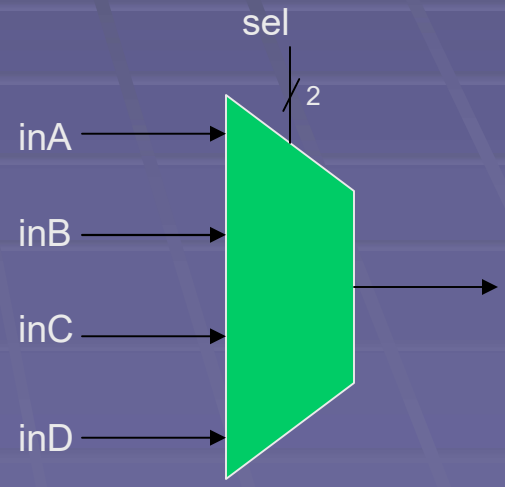
- Expression can evaluate if/else condition

```
x = a ? b : c; // if a, then x gets b, else x gets c
```

- Often used to implement a mux

```
logic [31:0] inA, inB, inC, inD;
logic [1:0] sel;      // mux select

      // 4:1 mux
output =    sel == 2'b00 ? inA :
            sel == 2'b01 ? inB :
            sel == 2'b10 ? inC :
                           inD;
```



Ranges & Concatenation

- Select a portion of a vector

```
logic [31:0] bus
logic [7:0] field = bus[31:24]; // MSB of bus
logic top = field[7];           // single bit
```

- Combine items: enclose in curly brackets

```
logic [31:0] bus;
logic [7:0] B = 8'h1_2;     // Can use "_" in values

bus = {B, 8'h5A, B, 8'h34}; // 32'h12_5A_12_34
```

Replication

- # of times to repeat concatenation

```
X = {5{1'bZ}};                    // X gets ZZZZZ
Y = { {4{1'b1}}, {4{1'b0}} };  // Y gets 11110000
```

- Here's code from one of the EE345 CPU's

```
data = (drv & ~rst) ? value : {12{1'bZ}};
```

- It means…
  - If "drv" (drive-the-value) and not "rst" (reset)
    - Then port "data" on tri-state bus gets "value"
  - Else
    - "data" gets floated

Verilog Combinatorial Blocks

Combinatorial Logic

- Also known as
  - Random logic
  - Combinational logic
- AND, OR, etc…
  - Equations, or expressions
- Verilog has certain places where you can put these

always_comb

- Used to contain logic that is *only* combinatorial
  - Not flip-flops or latches
- Example (from another EE345 CPU)

```
always_comb begin
    valid = memAccess.op.size != szNone;     // non-0  # of bytes to access

    memReq.memRd = valid & memAccess.op.isLoad;
    memReq.memWr = valid & ~ memAccess.op.isLoad;
        // ...
end
```

Start of block

Intermediate signal

End of block

Combinatorial equations

Note: “blocking” assignment uses “=“. Effect is for assignments to happen in order.

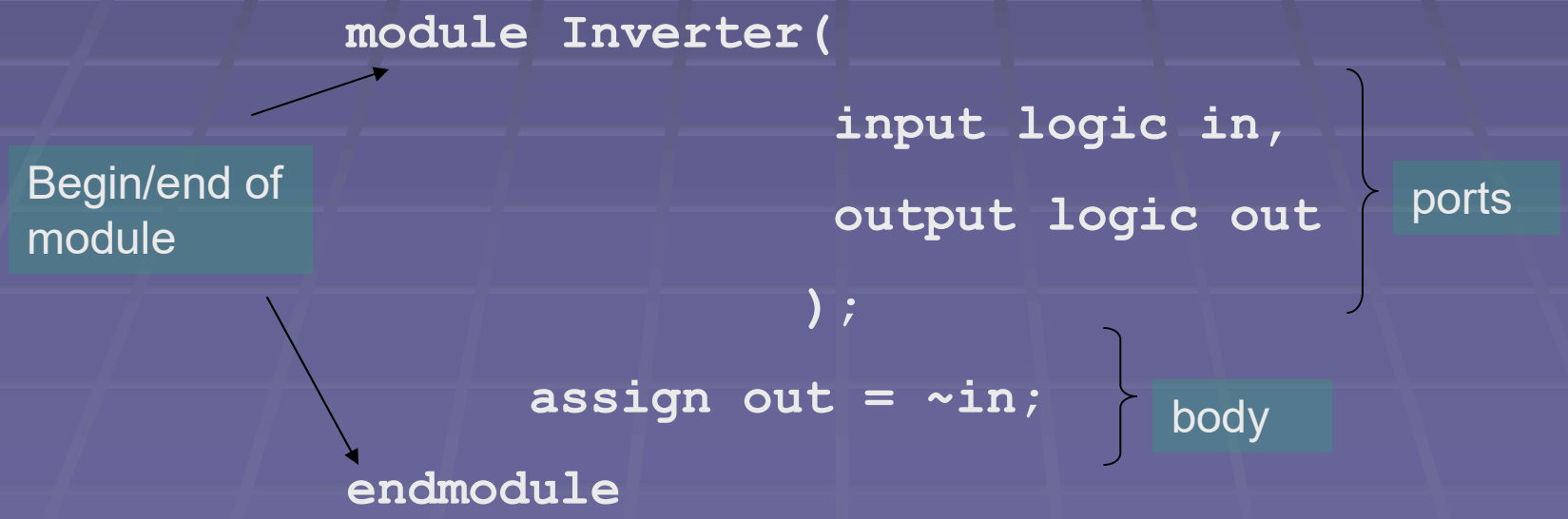
Assign

- Always_comb may be used for 1 or more equations
- “assign” may be used for a single equation
  - But you can always instead use always_comb
  - Older Verilog didn’t have always_comb

```
assign isZero = (value == 0);
```

Modules

- A “module” is like an IC for Verilog
  - Collection of logic
  - Defined inputs/outputs (ports)
  - May be instantiated multiple times



Using Modules

- Create an instance, connect ports

```
logic sigInput, sigOutput;
Inverter inv1(
              sigInput,   // in
              sigOutput   // out
              );
```

Positional port connections


```
Inverter inv2(
              .in(sigInput),
              .out(sigOutput)
              );
```

Named port connections
(usually preferred)

Assignment