CSE 371/372
(Structural) Verilog Primer

Amir Roth

And now … Verilog

- Structural Verilog: use for actual designs
  - Wires and wire assignment
  - Combinational primitives
  - Hierarchical modules
  - Timing
- Behavioral Verilog: use for wrappers and testing only
  - I.e., things you don't want to write gate-level designs for
  - Registers and memories
  - Behavioral assignment and sensitivity lists
- Verilog system features
  - Pre-processor
  - VPI

Basic Structural Datatype: Wire

```
wire W;
tri T;
```

- Identical to `wire`, meant to suggest multiple drivers
- We will not use (very much)

- Wire vectors: buses

```
wire [7:0] W1, W2;      // 2 8-bit buses
```

  - Ranges
    - Bound to type, not variable
    - Can be variables themselves
    - Don't have to be N–1:0 for N bit vector

```
wire [0:7] W4;
wire [15:8] W5;
```

Wire Assignment

- Wire assignment: "continuous assignment"
  - Connect combinational logic block or other wire to wire input
  - When RHS changes, RHS re-evaluated and re-assigned
  - Designated by the keyword **assign**

```
wire a, b, c;
assign c = a | b;
wire a, b, c = a | b;      // same thing
```

- Some new operations
  - Wire bit/range select
  - Concatenate: {<expr>[,<expr>]*}
  - e.g., swap high and low-order bytes of 16-bit vector

```
wire [15:0] w1, w2;
assign w2 = {w1[7:0], w1[15:8]}
```

Hardware Values

- A hardware signal can have four values
  - `0`
  - `1`
  - `x`: 0, 1, 0.5, floating, don't know, don't care, error
  - `z`: high-impedance (no current flowing)
- Vector constants

```
<size>'[<format>]<value>
reg [7:0] r;     // 8-bit reg
r = 8'b100;      // constant 4
r = 8'b1;        // constant 1
r = 8'h4;        // constant 4
r = 8'bz;        // high impedance on all 8 wires
```

Structural Primitives

- Gate-level
  - N-ary boolean operators: `and`, `or`, `xor`, `not`, `nand`, `nor`, `xnor`
    - E.g., C = A+B

```
or (C, A, B);
```

    - E.g., C= A+B+D

```
or (C, A, B, D);
```

  - That's about it

- Transistor-level primitives too
  - Will not use

Modules: User-Defined Blocks

- Hardware structure interface/implementation bundle

```
module what_is_this (S, A, B, O);
  input S, A, B;
  output O; wire O;

  wire S_, AnS_, BnS;

  not (S_, S);
  and (AnS_, A, S_);
  and (BnS, B, S);
  or (O, AnS_, BnS);
endmodule // what_is_this
```

Three Module Components

- Interface specification

```
module mux2to1(S, A, B, O);
  input S, A, B;
  output O; wire O;
```

  - Outputs must be re-declared `wire` (so we know to use wire assignment)
  - Can also have `inout`: bidirectional wire

- Internal wires, i.e., "local" variables

```
  wire S_, AnS_, BnS;
```

- Implementation

```
  not (S_, S);
  and (AnS_, A, S_);
  and (BnS, B, S);
  or (O, AnS_, BnS);
```

Hierarchical Modules

```
module mux2to1_4(S, A, B, O);
  input [3:0] A, B;
  input S;
  output [3:0] O; wire [3:0] O;

  mux2to1 mux0 (S, A[0], B[0], O[0]);
  mux2to1 mux1 (S, A[1], B[1], O[1]);
  mux2to1 mux2 (S, A[2], B[2], O[2]);
  mux2to1 mux3 (S, A[3], B[3], O[3]);
endmodule
```

- Note: user-defined modules must be named, primitives do not
- Tease: module instantiation arrays
  - Part of Verilog, but NYI by ICARUS

```
mux2to1 mux [3:0] (S, A, B, O);
```

Connections by Name

- Can (should) specify module connections by name
  - Helps keep the bugs away
  - Order doesn't matter

```
mux2to1 mux0 (.S(S), .A(A[0]), .B(B[0]), .O(O[0]));
mux2to1 mux1 (.S(S), .O(O[1]), .A(A[1]), .B(B[1]));
mux2to1 mux2 (.A(A[2]), .B(B[2]), .O(O[2]), .S(S));
mux2to1 mux3 (.O(O[3]), .A(A[3]), .B(B[3]), .S(S));
```

Timing

- In addition to functionality, can specify latency
  - `#<const>`: number of internal simulator ticks
  - E.g., wire with 5-tick propagation delay
    - All assignments to this wire will take 5 ticks to propagate

    ```
    wire #5 w;
    ```
  - E.g., and gate with 2-tick propagation delay

    ```
    and #2 (O,A,B);
    ```
  - No explicit delays for user-defined modules
  - Without explicit delays, changes are instantaneous
- Timing methodology
  - For now, set structural timing to instantaneous
  - Later, more detailed timing methodology (maybe)

Basic Behavioral Datatype: Register

```
reg R;
```

- Interface-less storage bit, not a "register"

- Another useful construct: M*N bit array (M N-bit words)
  - Referred to as a "memory"

```
reg [N-1:0] mem [M-1:0];
```

  - Can access individual memory words as register vectors

```
mem[idx];  // N-bit word at idx
```

Register Assignment

- Register Assignment: "procedural assignment"
  - One-shot deal, value remains until next assignment
  - Designated by absence of **`assign`** keyword

```
reg a, b, c;
c = a | b;
c <= a | b;
```

  - Blocking assignment (**`=`**)
    - Evaluated sequentially (like in C)
  - Non-blocking (**`<=`**)
    - Evaluated in parallel (using values from beginning of block)
    - Try to use non-blocking assignments within **`always`** block
  - What's the difference? Consider a shift-register

Behavioral Operations

- Very much like C
  - Arithmetic: `+`, `-`, `*`, `/`
  - Logical: `&&`, `||`, `!`
  - Bitwise: `&`, `|`, `^`, `~`, `&~`, `~|`
  - Comparison: `==`, `<>`, `>`, `<`, `<=`, `>=`
  - Conditional assignment: `?  :`

Behavioral Statements

- Like in C, but use `begin-end` instead of `{-}` to group

```
if (<expr>) <stmt> else if <stmt>

for (<stmt>;<expr>;<stmt>) <stmt>
```

  - Careful: No ++ operator in Verilog

```
case (<expr>)
   <match-constant1>:<stmt>
   <match-constant2>,<match-constant3>:<stmt>
   default: <stmt>
endcase
```

Behavior Invocation: Initial

```
initial
    begin
        <stmt>*
    end
```

- Initializes all local state
  - Otherwise initial values are X
  - Multiple `initial` sections are allowed

Behavior Invocation: Always

```
always @(<sensitivity><or sensitivity>*)
    begin
      <stmt>*
    end
```

- Defines reaction of module to changes in input
  - sensitivity list: signals or signal edges that trigger change
  - Keyword **or**: disjunction of multiple sensitivity elements
  - Multiple **always** sections are allowed
    - Careful: don't know order in which signals arrive
    - Best to use one

Signal and Signal Edge Sensitivity

- Signal sensitivity: evaluate block on any signal change

```
always @(CLK)
```

- Edge sensitivity: evaluate block on particular signal change

```
always @(posedge CLK)
```

- Quiz: what's the difference?

```
always @(D or CLK) if (CLK) Q <= D;
always @(posedge CLK) Q <= D;
```

Auxiliary Variables

- C style variables that are used procedurally
  - Understood to be "program helpers", not pieces of hardware

```
integer i;  // signed 32-bit (not int)
time t;     // unsigned 64-bit
real r;     // double precision FP
```

  - memory (i.e., C) like array syntax

```
integer iarray[63:0]; // array of 64 integers
```

  - E.g.,

```
integer i;
for (i = 0; i < N; i = i + 1)
   memory[i] <= 0;
```

  - E.g.,

```
time sim_num_insn; // retired instructions
```

Behavior Modules: Tasks

```
module memory ();
    reg [7:0] memory [1023:0];
    integer i;
    task clear;
      begin
         for (i = 0; i < 1024; i = i + 1)
            memory[i] <= 8'b0;
      end
    endtask
endmodule

memory mem();
initial mem.clear;
```

- Tasks: module "methods"
  - Can be invoked from `always` and `initial` blocks

Verilog Pre-Processor

- Like the C pre-processor
  - But uses ` (back-tick) instead of #
  - Constants: `define
    - No parameterized macros
    - Use ` before expanding constant macro

```
`define bus_width 16
wire [`bus_width-1:0] w;
```

  - Conditional compilation: `ifdef, `endif
  - File inclusion: `include

Useful VPI Calls

- Start with `$`

`$time`

  - Simulator's internal clock (64-bit unsigned)
  - Can be used as both integer and auto-formatted string

`$finish`

  - Terminate simulation

`$readmemh(<fname>,<mem>,<start>,<end>);`

`$writememh(<fname>,<mem>,<start>,<end>);`

  - Load contents of ASCII file to memory array (and vice versa)
  - Parameters `<start>`,`<end>` are optional
  - Useful for loading initial images, dumping final images…

More Useful VPI Calls

- For output

```
$display(<fmtstring><,signal>*);
$fdisplay(<fhandle>,<fmtstring><,signal>*);
```

  - Signal printf/fprintf

```
$monitor(<fmtstring><,signal>*);
```

  - Non-procedural printf, prints out when a signal changes

```
$dumpvars(1<,signal>*);
```

  - Similar to monitor
  - VCD format for waveform viewing (gtkwave)
  - Output is in dumpfile.vcd

An Example Test Module

```
`include "mux.v"
module main;
  reg [3:0] A, B;
  wire [3:0] O;
  reg S;

  mux2to1_4 mux (S, A, B, O);

  initial
   begin
     $monitor ($time,,"S=%b,A=%d,B=%d,O=%d", S, A, B, O);
     $dumpvars(1, S, A, B, O);

     #5 A=4'b1010; B=4'b0010; S=1'b0;
     #5 S=1'b1;
     #5 $finish;
   end
endmodule
```

A Test Module With Clock

```
`include "fsm.v"
module main;
  reg clk, in;
  wire out;

  fsm fsm1 (clk, in, out);

  always #5 clk <= ~clk;

  initial
   begin
     clk = 0;
     $monitor ($time,,"CLK=%b", clk, fsm1.state);
     $dumpvars(1, clk, fsm1.state);
     #100 $finish;
   end
endmodule
```