Verilog Cheat Sheet

S Winberg and J Taylor

Comments

```
// One-liner
/* Multiple
   lines */
```

Numeric Constants

```
// The 8-bit decimal number 106:
8'b_0110_1010 // Binary
8'o_152       // Octal
8'd_106       // Decimal
8'h_6A        // Hexadecimal
"j"           // ASCII

78'bZ         // 78-bit high-impedance
```

Too short constants are padded with zeros on the left. Too long constants are truncated from the left.

Nets and Variables

```
wire [3:0]w; // Assign outside always blocks
reg  [1:7]r; // Assign inside always blocks
reg  [7:0]mem[31:0];

integer j; // Compile-time variable
genvar  k; // Generate variable
```

Parameters

```
parameter  N     = 8;
localparam State = 2'd3;
```

Assignments

```
assign Output = A * B;
assign {C, D} = {D[5:2], C[1:9], E};
```

Operators

```
// These are in order of precedence...
// Select
A[N] A[N:M]
// Reduction
&A ~&A |A ~|A ^A ~^A
// Compliment
!A ~A
// Unary
+A -A
// Concatenate
{A, ..., B}
// Replicate
{N{A}}
// Arithmetic
A*B A/B A%B
A+B A-B
// Shift
A<<B A>>B
// Relational
A>B A<B A>=B A<=B
A==B A!=B
// Bit-wise
A&B
A^B A~^B
A|B
// Logical
A&&B
A||B
// Conditional
A ? B : C
```

Module

```
module MyModule
#(parameter N = 8) // Optional parameter
 (input  ipClk, ipReset,
  output [N-1:0]opOutput);
  // Module implementation
endmodule
```

Module Instantiation

```
// Override default parameter: setting N = 13
MyModule #(13) MyModule1(Clk, Reset, Result);
```

Case

```
always @(*) begin
  case(Mux)
    2'd0: A = 8'd9;
    2'd1,
    2'd3: A = 8'd103;
    2'd2: A = 8'd2;
    default:;
  endcase
end

always @(*) begin
  casex(Decoded)
    4'b1xxx: Encoded = 2'd0;
    4'b01xx: Encoded = 2'd1;
    4'b001x: Encoded = 2'd2;
    4'b0001: Encoded = 2'd3;
    default: Encoded = 2'd0;
  endcase
end
```

Synchronous

```
always @(posedge ipClk) begin
  if(Reset) B <= 0;
  else      B <= B + 1'b1;
end
```

Loop

```
always @(*) begin
  Count = 0;
  for(n = 0; n < 8; n++)
    Count = Count + Input[n];
end
```

Function

```
function [6:0]F;
  input [3:0]A;
  input [2:0]B;
  begin
    F = {A+1'b1, B+2'd2};
  end
endfunction
```

Generate

```
genvar n;
wire [12:0]Output[19:0];

generate
  for(n = 0; n < 20; n++)
  begin: Gen_Modules
    MyModule #(13) MyModule_Instance(
      Clk, Reset,
      Output[j]
    );
  end
endgenerate
```

State Machine

```
enum { Start, Idle, Work, Done } State;

reg Reset;

always @(posedge ipClk) begin
  Reset <= ipReset;

  if(Reset) begin
    State <= Start;

  end else begin
    case(State)
      Start: begin
        State <= Idle;
      end
      Idle: begin
        State <= Work;
      end
      Work: begin
        State <= Done;
      end
      Done: begin
        State <= Idle;
      end
      default:;
    endcase
  end
end
```