Overview of Verilog – Part 1

- Objectives
- Verilog Basics
  - Notation
  - Keywords & Constructs
  - Operators
- Types of Descriptions
  - Structural
  - Dataflow
    - Boolean Equations
    - Conditions using Binary Combinations
    - Conditions using Binary Decisions
  - Behavioral & Hierarchical
  - Process (Procedural)
Objectives

- To become familiar with the hardware description language (HDL) approach to specifying designs
  - Be able to read a simple Verilog HDL description
  - Be able to write a simple Verilog HDL description using a limited set of syntax and semantics
  - Understanding the need for a “hardware view” when reading and writing an HDL

Verilog Notation - 1

- Verilog is:
  - Case sensitive
  - Based on the programming language C

- Comments
  - Single Line
    // [end of line]
  - Multiple Line
    /* */

- List element separator: ,
- Statement terminator: ;
Verilog Notation - 2

- Binary Values for Constants and Variables
  - 0
  - 1
  - x, x - Unknown
  - Z, z – High impedance state (open circuit)
- Constants
  - n'b[integer]: 1'b1 = 1, 8'b1 = 000000001, 4'b0101 = 0101, 8'bxxxxxxx, 8'bxxxx = 0000xxxx
  - n'h[integer]: 8'hA9 = 10101001, 16'hf1 = 0000000011110001
- Identifier Examples
  - Scalar: A, C, RUN, stop, m, n
  - Vector: sel[0:2], f[0:5], ACC[31:0], SUM[15:0], sum[15:0]

Verilog Keywords & Constructs - 1

- Keywords are lower case
- **module** – fundamental building block for Verilog designs
  - Used to construct design hierarchy
  - Cannot be nested
- **endmodule** – ends a module – not a statement
  => no “;”
- Module Declaration
  - **module** module_name (module_port, module_port, ...
  - Example: module full_adder (A, B, c_in, c_out, S);
Verilog Keywords & Constructs - 2

- **Input Declaration**
  - **Scalar**
    - `input` list of input identifiers;
    - Example: `input A, B, c_in;`
  - **Vector**
    - `input [range]` list of input identifiers;
    - Example: `input[15:0] A, B, data;`

- **Output Declaration**
  - **Scalar Example:** `output c_out, OV, MINUS;`
  - **Vector Example:** `output [7:0] ACC, REG_IN, data_out;`

Verilog Keywords & Constructs - 3

- **Primitive Gates**
  - `buf, not, and, or, nand, nor, xor, xnor`
  - **Syntax:** `gate_operator instance_identifier (output, input_1, input_2, …)`
  - **Examples:**
    - `and A1 (F, A, B); // F = A B`
    - `or O1 (w, a, b, c)`
      - `O2 (x, b, c, d, e); // w=a+b+c, x=b+c+d+e`
Verilog Operators - 1

• Bitwise Operators
  ~  NOT
  &  AND
  |  OR
  ^  XOR
  ^~ or ~^  XNOR

• Example:
  input [3:0] A, B;
  output [3:0] Z ;
  assign Z = A | ~B;

Verilog Operators - 2

• Arithmetic Operators
  +, −,  (plus others)

• Logical & Relational Operators
  !, &&, | |, = =, !=, >=, <=, >, < (plus others)

• Concatenation & Replication Operators
  {identifier_1, identifier_2, …}
  {n{identifier}}
  ▪ Examples: {REG_IN[6:0], Serial_in},
              {8 {1′b0}}
Structural Verilog

- Circuits can be described by a netlist as a text alternative to a diagram - Example (See Figure 3-59 in text):

```verilog
module fig359s (A0, B0, C0, C1, S0);
    input A0, B0, C0;
    output C1, S0;
    //Seven internal wires needed
    wire[1:7] N;
    //Ports on primitive gates listed output port first
    not G1 (N[3],C0), G2 (N[5],N[2]), G3 (N[6],N[3]);
    nand G4 (N[1],A0,B0);
    nor G5 (N[2],A0,B0), G6 (C1,N[2],N[4]);
    and G7 (N[4],N[1],N[3]), G8 (N[7],N[1],N[5]);
    xor G9 (S0,N[6],N[7]);
endmodule
```

Dataflow Verilog - 1

- Circuit function can be described by assign statements using Boolean equations (See Figure 3-59 in text):

```verilog
module fig359d (A0, B0, C0, C1, S0);
    input A0, B0, C0;
    output C1, S0;
    wire[1:2] N;
    assign N[1] = ~(A0 & B0); /*Note: Cannot write ~& for NAND */
    assign N[2] = ~(A0 | B0);
    assign C1 = ~(N[1] & ~C0 | N[2]);
    assign S0 = (~N[2] & N[1])^(~(~C0));
endmodule
```
Dataflow Verilog - 2

- Circuit function can be described by `assign` statements using the `conditional` operator with binary combinations as in a truth table (See Figure 3-14 in text):

```verilog
module fig314dm (A, E_n, D_n);
    input [1:0] A;
    input E_n;
    output [3:0] D_n;
    //Conditional: (X) ? Y: Z - if X is true, then Y, else Z
    assign D_n = {4{E_n}} & (
        (A == 2'b00) ? 4'h1110:
        (A == 2'b01) ? 4'h1101:
        (A == 2'b10) ? 4'h1011:
        (A == 2'b11) ? 4'h0111:
        4'bxxxx);
endmodule
```

Dataflow Verilog - 3

- Circuit function can be described by `assign` statements using the `conditional` operator for binary decisions on inputs (See Figure 3-14 in text):

```verilog
module fig314dc (A, E_n, D_n);
    input [1:0] A;
    input E_n;
    output [3:0] D_n;
    /* Conditional: (X) ? Y: Z - if X is true, then Y, else Z */
endmodule
```
Behavioral & Hierarchical Verilog

- Circuit function can be described by assign statements at higher than the logic level (See Figure 3-31 in text):

```verilog
module addsub (A, B, R, sub);
    input [3:0] A, B;
    output [3:0] R; //See Fig. 3-51 for carry out
    input sub;
    wire [3:0] data_out;
    add A1 (A, data_out, sub, R);
    M1comp C1 (B, data_out, sub);
endmodule
```

```verilog
module add (X, Y, C_in, S);
    input [3:0] X, Y;
    input C_in;
    output [3:0] S;
    assign S = X + Y + {3'b0, C_in};
endmodule
```

```verilog
module M1comp (data_in, data_out, comp);
    input [3:0] data_in;
    input comp;
    output [3:0] data_out;
    assign data_out = {4{comp}} ^ data_in;
endmodule
```