VERILOG
History

• Developed by Gateway Design Automation
• Bought by Cadence Design Systems
• Now an IEEE standard: IEEE-1364

• Pure language definition

• Website of links see www.esperan.com
Module

- This is the equivalent of the ENTITY and ARCHITECTURE pair in VHDL

```
module HALFADD(A, B, SUM, CARRY);
    output SUM, CARRY;
    input A, B;
    assign SUM = A ^ B;
    assign CARRY = A & B;
endmodule
```

- Verilog is case sensitive

^ is XOR
& is AND
Instantiation

- Like VHDL named and inferred port mapping allowed.
- Unlike VHDL no configuration file is supported. However, order of compilation is not important.

```verilog
module FULLADD( A, B, CIN, SUM, CARRY);
input A, B, CIN;
output SUM, CARRY;
wire N_SUM, CARRY1, CARRY2;
HALFADD u1(A, B, N_SUM, CARRY1);
HALFADD u2(.B(N_SUM), .A(CIN), .SUM(SUM), .CARRY(CARRY2));
or  u3(CARRY, CARRY2, CARRY1);
endmodule
```

Similar to a signal definition

The or is a built-in gate primitive within Verilog
Sequential Statements

• In VHDL one has the PROCESS statement. In VERILOG one has the ALWAYS and INITIAL statements.

module ZOR(A, B, Z_OR);
input A, B;
output Z_OR;
reg Z_OR;
always @(A or B)
begin
if ((A == 1) | (b == 1))
    Z_OR <= 1;
else
    Z_OR <= 0;
end
endmodule

If output is generated in a sequential statement block it must be of a register type.

Sensitivity list.

All sequential statements are enclosed by a begin and an end (unless there is only one statement in the branch)
Sequential Statements (cont)

- The initial statement is run once only at the start of the simulation.

```
initial
begin
  OUT_A=0;
  OUT_b=1;
end
```
Data Types

• Verilog objects communicate using variables (this is only a different name as from VHDL)
• All variables have a type
• Verilog has built in types (i.e. enumerated types are not supported)

• There are two classes of data types
  • Register
  • Net

• The type must be defined when the variable is declared.
• Module ports are wire by default. Need to be redefined in certain conditions.
Net Data Types

- Unlike VHDL resolution functions are built in to the net definitions
- List of some net types
  - wire
  - tri
  - wand (Wired AND so a resolved type)
  - wor (Wired OR)
- Possible values
  - 1'b1, 1'b0, 1'bx, 1'bz
  - 4'b1000

1 is the width of the bus
' is the base indicator
b is binary
z is value
Register Data Types

• Need to use this type to store data.
• List of types
  • reg : unsigned variable of any type
  • integers : 32 bit signed (twos compilment) number
  • real : 64 bit unsigned number
  • time : 64 bit unsigned number

• NOTE: Not a strongly typed language as is VHDL. One can assign an integer to a reg. However, as one is signed and the other is not the value will change (though the binary representation will not.)
Vectors

- Example of vector definition. Bit order is defined when the vector is declared.
  - `reg [3:0] UP_BUS;`
  - `wire [15:4];`

- Vector assignment
  - Vector to vector. Elements are assigned by position not element number. Therefore, remember to number in a consistent manner.
  - Vector values can be decimal (default), hexadecimal, octal or binary
    - `bus <= 8'o67` (Note that the leading elements, if any, are stuffed to zero)
Arrays

- Verilog support for more complex data structures is limited. Only two dimensional arrays are supported. Records and larger arrays are not supported.

- Definition of array
  - `reg [7:0] MEM[255:0]` This gives a 8 bit wide 256 locations deep array
  - `integer NUM[99:0]` An integer is already a vector.
Parameter

- This is similar to the VHDL GENERIC.
- Parameters are local to a module.

Example
- parameter WIDTH = 8;
- wire [WIDTH-1:0] DATA;
Operators

- Verilog has a full set of operators. (Rather C type that VHDL)
- Arithmetic operators
  - + Add, - Subtract, * Multiply, / Divide, % Modulus
- Bit-wise operators. Normally used on vectors. Each element considered separately
  - ~ Not, & And, | or, ^ xor, ~^ xnor
- Logical operators. Return a 1, 0 or X
  - ! Not, && And, || Or
- Relational and Equalities
  - > greater than, < less than, >= greater than or equal, == logical equality
- Other more "esoteric" operators exist.
Conditional statements

- The IF and CASE statements are supported
- **IF**
  
  if (CONDITION) begin
  // sequential statements
  end
  
  else if (CONDITION) begin
  // sequential statements
  end
  
  else begin
  // sequential statements
  end
Conditional statements (cont)

• **CASE**
  
  ```
  case (EXPRESSION)
  VALUE_1 : statements
  VALUE_2 : statements
  default : statements
  endcase
  ```

• Two more different forms of the CASE exist. CASEX and CASEZ. CASEX treats both z and x as don't care, while CASEZ treats only z as don’t' care.
  
  • Casex (DATA)
  • `4'b0xx1 : Z <= A;`

  **Don't care values not used for comparison**
FOR LOOP

• Unlike VHDL the loop variable must be declared

    integer I;
    .
    .
    for (I = 0; I < 4; I = I + 1)
    begin
        Z <= I;
    end
Repeat and While loops

• WHILE
  while ($time < 800)
  begin
    @(negedge CLK);
    A <= B;
  end

• REPEAT
  repeat (12)
  begin
    @(posedge CLK);
    A <= B;
  end
Tasks and Functions

- Subprograms supported. However, unlike in VHDL a procedure call calls the same procedure not a copy there of. Control may come from a different process.
- VERILOG does not support packages.
- The function is the same as in VHDL. Returns only one value.
- A procedure is called a task. May have multiple outputs.
function integer COUNT;
input [7:0] IN_BUS;
integer I;
begin
    COUNT = 0;
    for (I = 0; I < 8; I = I + 1)
        if (!IN_BUS[I])
            COUNT = COUNT + 1;
end
endfunction

Function calls:
wire [2:0] COUNT_B = COUNT(BUS);
if (COUNT(BUS) == 0 )
Task

```verbatim
task CLOCK;
input [31:0] NUMBER;
begin
repeat (NUMBER)
  @(negedge CLK);
end
endtask

TASK calls
CLOCK (counter)
```

Visible from the calling function
Simulation Cycle

- The simulation cycle is the same as in VHDL. However, the treatment of variables (VHDL meaning) is different.
- All Verilog signals are globally visible including variables. However, variable assignment is blocking.
- Example
  
  ```verilog
  always @(A or B)
  #5 A = 1;
  B <= 3;
  ```

  This statement only executed after five time units as the sequential statements are blocked by the previous assignment.

- If no timing information then race conditions will occur where execution order will affect the result.
WAIT statement

- Limited when compared to VHDL.
- The wait statement is level sensitive.
Timing in Verilog

• Unlike VHDL VERILOG has timing tightly linked to the language.
• Simple delays
  • assign #DELAY Z = A ^ B
  • Q <= #10 D;
• Rise and fall delays
  • assign #(RISE_T, FALL_T, TO_Z) Z = A ^ B
  • Q <= #(10, 11, 35) D;
• Min, max, typical delays
  • assign #(MIN:TYP:MAX) Z = A ^ B
  • Q <= #(10:11:12, 2:3:4) D;
Adding real time units

- specify statement
  ```
  specify
  (A => OUT ) = 12,
  ( B => OUT ) = 14,
  (C,D => OUT) = 15;
  endspecify
  ```

- 'timescale 1ns/100ps

- Make sure vendor libraries have same scale and resolution
IO

- The file I/O is limited when compared to VHDL.
- Can only read binary or hexadecimal values.
- Can only read into a two-dimensional array.
- Can only open 32 files.