HDL

- HDL stands for **Hardware Description Language**
- Definition : A high level programming language used to model hardware.
- Hardware Description Languages
  - have special hardware related constructs.
  - currently model digital systems, and in future can model analog systems also.
  - can be used to build models for simulation, synthesis and test.
  - have been extended to the system design level.

Why Use HDLs

- Allows textual representation of a design.
- High level language similar to C,C++.
- Can be used for Modeling at the
  - Gate Level
  - Register Level
  - Chip Level
- Can be used for many applications at the
  - Systems Level
  - Circuit Level
  - Switch Level
- Design decomposition is simple with HDLs and hence can manage complexity
- Early validation of designs.

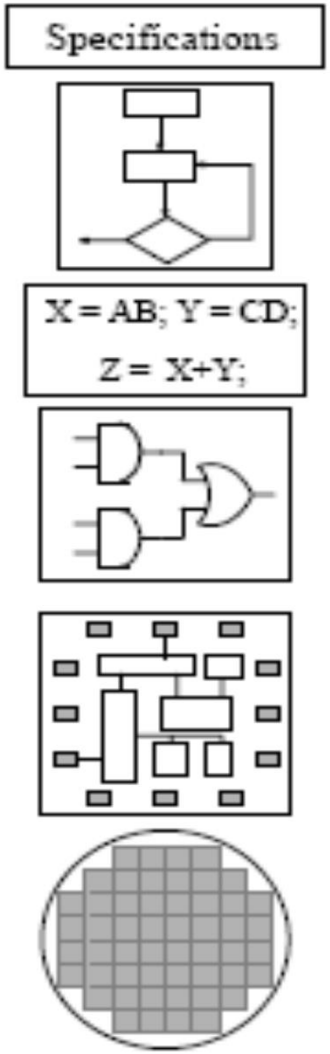
Need for Design Tools

- Current systems are very complex.
- Design abstraction and decomposition is done to manage complexity.
- Tools automate the process of converting your design from one abstraction level to another.
- Design Automation Tools improve productivity.
- Different tools are required in different steps.

Classification of CAD Tools

- Editors
  - Allows specification of the design either textually or graphically.
- Simulators
  - Models the response of a system to input stimuli.
- Analyzers
  - Used at different levels to check for correctness and compliance to rules.
- Synthesis
  - Transformation of representation between different abstraction levels.

Flow and Tools



Specifications

↓

Functional Design

↓

Logic Design

↓

Circuit Design

↓

Physical Design

↓

Fabrication

← System Description Languages

← Hardware Description Languages, Schematic Editors

← Logic Synthesis Tools (Tech mappers)

← Physical Synthesis Tools (Place & Route)

← Tape out and Manufacture

CAD Tools -1 Design entry

- Graphical
  - Silicon Level – To create layouts
    - e.g. Magic
- Other Levels
  - e.g. ViewLogic, Protel
- Text
  - Natural language specification at system level.
  - Hardware Description Languages at Chip, Register and Gate levels.
    - e.g. VHDL, Verilog
- Circuit Level
  - e.g. SPICE

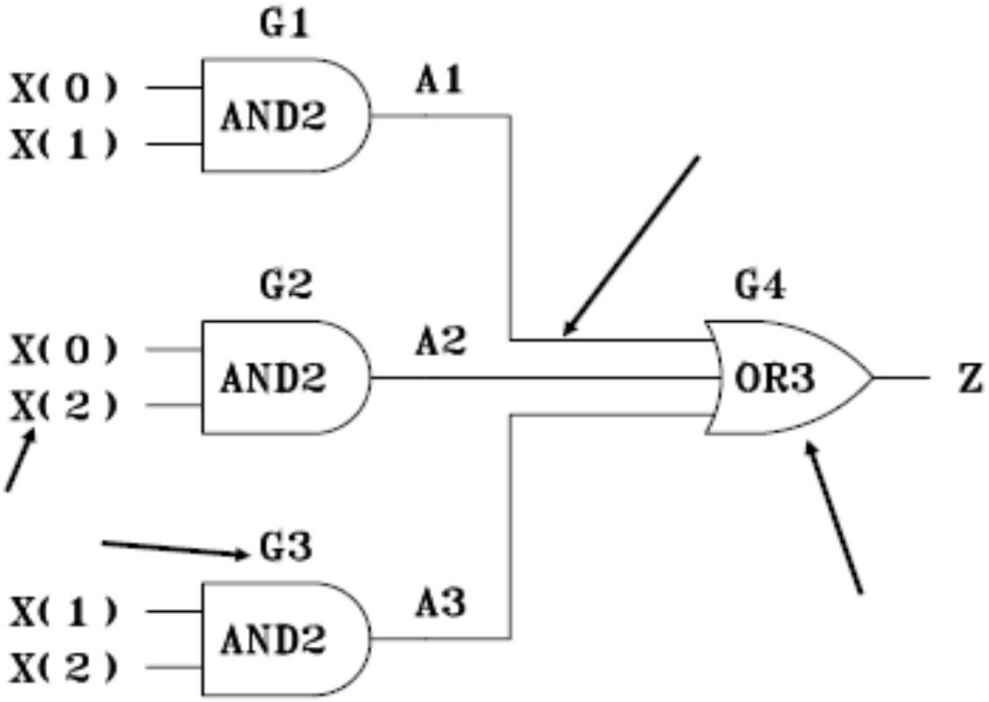
Graphical Editors

- Silicon Level editors are called Layout editors.
  - Draw rectangles describing metal, poly, diffusion etc
  - Library components are also at the same level.
  - Usually has online Design Rule Checking (DRC).
- Graphical Editors at other levels are usually called Schematic editors.
  - Used to create block diagrams and schematics.
  - The process is usually called *Schematic Capture*.

Schematic Editors

- Can create and display graphical components called “tokens”
- Can “interconnect” these tokens.
- Advantage :
  - Gives a structural representation called “netlist” describing the components used and their interconnections.
  - Also provides a simulation model to find the system’s response for different stimuli.

Example of Schematic Entry



Text based Design Entry

- Choose a specific HDL.
- Use text editors to describe the design.
  - e.g. vi, emacs, notepad etc.
  - Some tools have built-in editors
- Enter your design conforming to the language lexicon, syntax and semantics.
- Check for errors.
- “Compile” to get a simulation model.

What makes HDLs Different ?

- Hardware systems are concurrent in nature.
- Hardware systems may be distributed in nature.
  - Many components
  - Different rates for processing data, different clocks.
- Hardware systems are timed.
  - All hardware components have inherent delays and hence managing timing is crucial.
- Traditional software design techniques are insufficient.

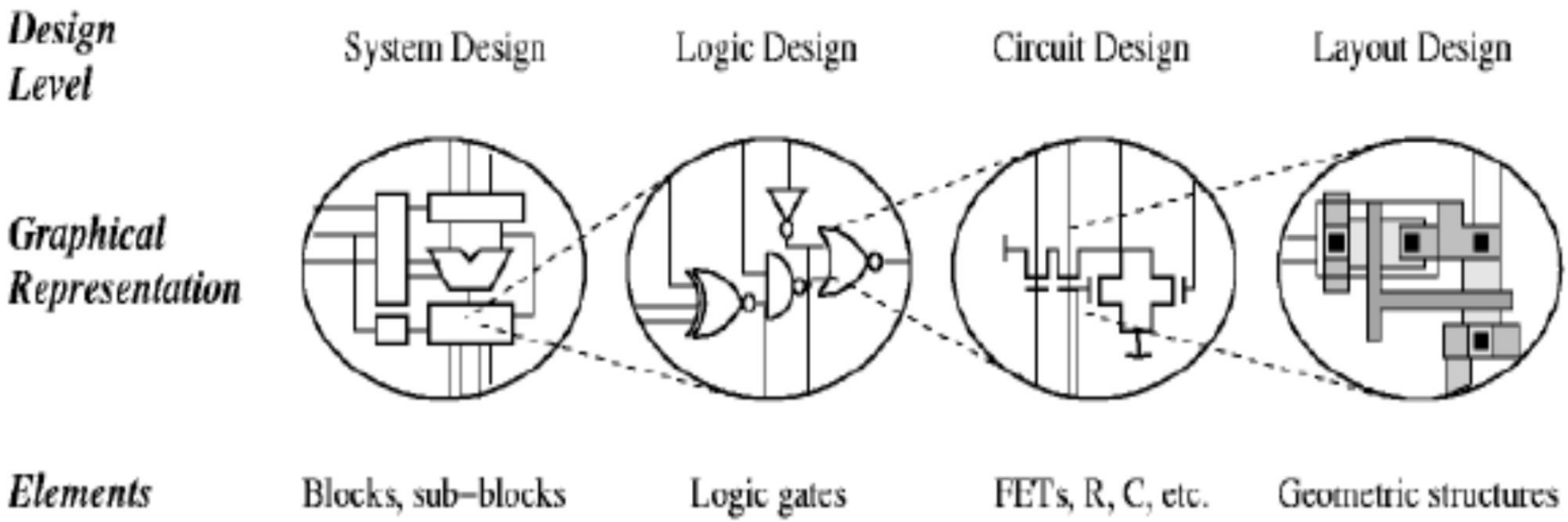
CAD Tools -2 Simulators

- Defn. : A program that models response of a system to the input stimuli.
- Simulation is widely used to establish design correctness.
- Types
  - Deterministic
  - Stochastic

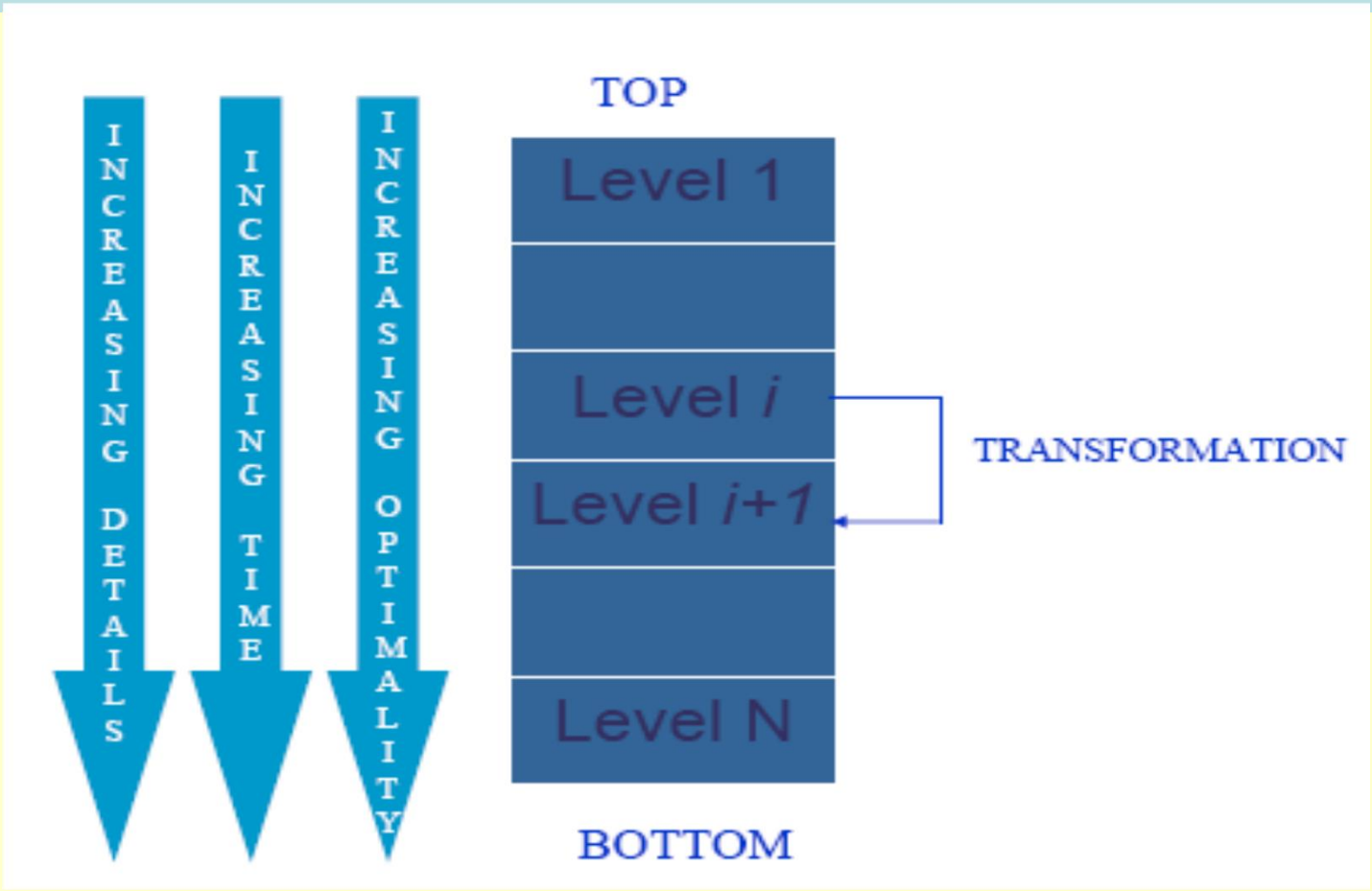
CAD Tools -3 Synthesis Tools

- Synthesis Definition : Transformation of a representation in one hierarchical level to another.
- Different names in different levels :
  - Algorithmic Synthesis – Abstract behavioral to register level or gate level specification
  - Logic Synthesis – RTL specification to gates
  - Physical Synthesis – Structural specification as gates to layout.

Synthesis at different Levels



Synthesis Transformations



Verilog HDL

Lexical Conventions

- Similar to C
- Whitespace
  - Blank spaces, tabs and newlines
  - \b, \t, \n
  - Whitespace ignored except in
    - Strings and token separation
- Comments
  - // - single line;
  - /* … */ - multiple lines

Operators

- Unary,
  - A = ~b;
- Binary
  - a = b&&c;
- Ternary
  - a = b ? c : d;

Number Specifications

- • Sized numbers :

  <size>’<base format><number>

  – 4’b1111; 12’habc

  – <size> - number of bits

- Unsized numbers

  – 23456 // default is decimal

  – ‘hc3

Number Specification

- Negative numbers
  - <size>'<base format><number>
    - -6'd3 // 6-bit negative number stored as two's complement of 3
    - 4'd-2 //illegal specification

Possible Values

- X or Z values
  - 12'h13x; 6'hx; 32'bz;
  - x or z sets 1,3,4 bits in binary, octal and hexadecimal representations respectively
  - Value extension
    - If most significant bit is
      - 0,x,z then 0,x,z respectively
      - 1 then 0
- Data types
  - Value set – 0,1,x, z
  - Signal strength

Readability

- Readability enhancements
  - 12'b1111_1110_0101 //Underscore
- Strings – sequence of ASCII bytes
  - "Hello Verilog Word"; // a string

Identifiers and Keywords

- reg value; // reg – keyword
- input clk; // input – keyword
- More keywords as we progress
- Identifiers made of alphanumeric characters, the underscore (_) and the dollar ($) sign and starts with alphabets or underscore.
- The dollar sign as first character is reserved for system tasks.
  - Ex : $monitor

Nets and Regs

- • Nets
  - Connection between hardware elements
    - wire a; wire b,c; wire d = 1'b0;
    - wand, wor, tri, triand, trior and trireg are other forms of net
- Regs
  - Data storage elements
  - Not equivalent to hardware registers
  - A variable that stores value
  - Unlike a net, it needs no driver
  - Default value for a 'reg' variable is x

Vectors

- Nets and register
- wire [7:0] busA, busB;
- reg [16:0] address;
- [high# : low#] or [low# : high#], but the most significant bit is the left number in the square bracket
- reg [0 : 40] addr1; wire [15 : 12] addr2;

Addressing Vectors

- Address bits are parts of vectors
  - reg [7:0] busA;
    - busA[7]; //stands for 7th bit
    - busA[2:0]; // first 3 LSBs
  - wire [0:15] addr1;
    - addr1[2:1] is illegal;
    - addr1[1:2] is correct;
  - Writing to a vector :
    - busA[2:0] = 3'd6;

Numbers

- Integers
  - datatype storing signed numbers
  - reg stores unsigned numbers
  - Default length is 32-bits
  - integer counter;
  - counter = -1; // stores as 32-bit two's complement number

Real Numbers

- Decimal and Scientific Notations
  - real delta;
    - delta = 4e10;
    - delta = 2.13;
  - integer a;
    - a = delta; /*‘a’ gets the value 2 which is the rounded value of 2.13*/

Time

- Verilog simulation is done with respect to simulation time
- To store simulation time, a special ‘time’ variable is declared in verilog
- The ‘time’ variable is at least 64-bits
- time save_sim_time; save_sim_time = $time;
- Useful for timing measurements
  - Debug

Arrays

- <array_name> [<subscript>]
  - reg bool[31:0]; //compare it with vectors
  - time chk_point [1:100];
  - integer count [0:7];
  - reg [4:0] port_id[0:7];
  - integer matrix_d[4:0][4:0] // illegal
  - count[5]; chk_point[100];
  - How to access say bit 3 of port_id[5];
    - reg [4:0] a;
    - a = port_id[5]; // a[3] is required bit

Arrays vs Vectors

- A vector is a single element that is n-bits wide;
- Arrays are multiple elements that are 1-bit or n-bit wide.

Memories

- reg mem1bit [0:1023];

  // a memory of 1024, 1-bit words
- reg [7:0] membyte [0:1023];

  // a memory of 1024, 8- bit words
- membyte [511];

  //fetches 1 byte word whose address is 511

Parameters

- Imagine this case
  - You design a 4-bit adder with HAs and FAs
  - You need 8-bit adders too
  - Rewrite is a waste of effort
    - Prone to bugs too
- Parameters
  - Method to define constants inside a module
  - For eg. parameter port_id = 5;
  - Can be overriden at module instantiation time

Strings

- Stored in ‘reg’ variables
  - each character takes 8- bits
- reg [8*19:1] string_value;
- string_value = “Hello Verilog world”;
- string_value is defined as a string that can store 19 characters.

System Tasks

- Vendor specific
- Some generic tasks
  - $display
    - similar to printf and used for displaying values of variables and expressions
  - Values can be printed in decimal (%d), binary (%b), string (%s), hex (%h), ASCII character (%c), octal (%o), real in scientific (%e), real in decimal (%f), real in scientific or decimal whichever is shorter (%g).
  - time format (%t), signal strength (%v) and hierarchy name (%m).

System Tasks

- $monitor
  - Monitor a signal when its value changes
  - Only one active monitor list
    - If more than one then, the last one statement will be the active one.
  - Monitoring turned ON by default at start of simulation and can be controlled during the simulation using **$monitoron** and **$monitoroff**

System Tasks

- $stop – stops simulation and puts it into interactive mode
- $finish – terminates the simulation

Continuous Assignments

- assign out = i1 & i2;
- assign addr[15:0] = addr1[15:0]^addr2[15:0];
- assign {cout,sum[3:0]} = a[3:0]+b[3:0]+c_in;
- wire out; assign out = in1 & in2;
- is equivalent to
- wire out = in1 & in2; //Implicit continuous assignment

Expressions

- Dataflow modeling describes the design in terms of expressions instead of primitive gates.
- Expressions – those that combine operands and operators
- a ^ b; addr1[20:17] + addr2[20:17];
- in1 | in2;

Operands

- Constants, integers, real numbers
- Nets, Registers
- Times
- Bit-select
  - One bit of a vector net or vector reg
- Part-select
  - Selected bits of vector net or vector reg
- Memories

Operators -Types

- Arithmetic
- Logical
- Relational
- Equality
- Bitwise
- Reduction // Not available in software languages
- Shift
- Concatenation
- Replication
- Conditional
- Syntax very similar to C

Arithmetic Operators

- Binary Operators
  * - multiply
  / - division
  + - addition
  - subtraction

- Commercial Verilog gives non ‘x’ values whenever possible

  a = 4’b0x11; b = 4’b1000; a+b = 4’b1x11;

  a = 4’b0x11; b = 4’b1100; a+b = 4’bxx11;

Logical Operators

- logical and (&&), logical or (||), logical not (!).
  - They evaluate to a 1-bit value: 0 (false) 1 (true) or x (ambiguous)
  - If an operand is not equal to zero, it is a logical 1 and if it is equal to zero, it is a logical 0. If any operand bit is x or z, then operand is x and treated by simulators as a false condition
  - Logical operators take variables or expressions as operands.

Logical Operators Examples

- A = 3; B = 0;

  A&&B, A||B evaluates to 0 and 1 resp.

  !A, !B evaluates to 0 and 1 resp.

- A = 2'b0x; B = 2'b10;

  A&&B evaluates to x

- (a==2) && (b == 3) //Expressions

Relational Operators

- Greater-than (>)
- Less-than (<)
- Greater-than-or-equal-to (>=)
- Less-than-or-equal-to (<=)
- Evaluates to 1 or 0, depending on the values of the operands
  - If one of the bits is an ‘x’ or ‘z’, it evaluates to ‘x’

Relational Operators (Example)

- //A = 4, B = 3
- //X = 4'b1010, Y = 4'b1101, Z = 4'b1xxx
- A <= B //returns 0
- A > B //returns 1
- Y >= X //returns 1
- Y < Z //returns x

Equality Operators

- Logical equality (==), logical inequality (!=) :

  if one of the bits is ‘x’ or ‘z’, they output ‘x’ else returns ‘0’ or ‘1’

- Case equality (===), case inequality (!==) :

  compares both operands bit by bit and compare all bits including ‘x’ and ‘z’. Returns only ‘0’ or ‘1’

Equality Operators Examples

- //A = 4;B = 3;X = 4'b1010;Y = 4'b1101
- //Z = 4'b1xxz;M = 4'b1xxz;N = 4'b1xxx
- A == B // result is 0
- X != Y //result is 1
- X == Z // result is x
- Z === M //result is 1
- Z === N //result is 0
- M !== N //result is 1

Bitwise operators

- negation (~), and (&), or (|), xor (^), xnor(^~, ~^).
- ‘z’ is treated as ‘x’ in the bitwise operations

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

buf	in	out	not	in	out
	0	0		0	1
	1	1		1	0
	x	x		x	x
	z	x		z	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

Bitwise Operators Examples

- //X = 4'b1010;Y = 4'b1101;Z = 4'b10x1
- ~X // result is 4'b0101
- X & Y // result is 4'b1000
- X | Y // result is 4'b1111
- X ^ Y // result is 4'b0111
- X ^~ Y // result is 4'b1000
- X & Z // result is 4'b10x0

Point to Note

- We distinguish between bitwise operators and logical operators
- //X = 4’b1010;Y = 4’b0000
- X | Y // result is 4’b1010
- X || Y // result is 1

Reduction Operators

- Reduction operators perform a bitwise operation on a single vector operand and yield a 1-bit result.
- Reduction operators work bit by bit from right to left.

Reduction Operators Examples

- and (&), nand (~&), or (|), nor (~|), xor (^); and xnor (~^,^~)
- This is a UNARY operation on vectors
- Let X = 4'b1010
- &X is 1'b0 //0 & 1 & 0 & 1 = 1'b0
- |X is 1'b1
- ^X is 1'b0
- A reduction xor or xnor can be used for even or odd parity generation of a vector.

Shift Operators

- Right shift (>>) and left shift (<<)
- //X = 4'b1100
- Y = X >> 1; // Y is 4'b0110
- Y = X << 1; // Y is 4'b1000
- Y = X << 2; // Y is 4'b0000
- Very useful for modeling shift-and-add algorithms for multiplication.

Concatenation Operators

- Denoted by ({,})
- Append multiple ‘sized’ operands. Unsized operands are NOT allowed as size of each operand should be known to compute size of the result
- //A=1’b1;B=2’b00;C=2’b10;D=3’b110;
- Y = {B,C} // Y is 4’b0010
- Y = {A,B,C,D,3’b001} // Y = 11’b10010110001
- Y = {A, B[0],C[1]} // Y is 3’b101

Replication Operators

- Repetitive concatenation of the same number can be represented using a replication constant
- A = 1'b1; B=2'b00;
- Y = { 4{A} }; //Y is 4'b1111
- Y = {4{A},2{B}}; //Y is 8'b11110000

Conditional Operator

- Usage: condition_expr? true_expr : false_expr;
- If condition evaluates to ‘x’, then both expressions are evaluated and compared bit by bit to return for each bit position, an ‘x’ if the bits disagree, else the value of the bit.
- The conditional expression models a 2-to-1 multiplexer
- assign out = control ? in1 : in0;

Compiler Directives

- Usage : `<keyword>
- `define WORD_SIZE 32; compared with #define in C
- `include header.v compared with #include in C.
- `ifdef and `timescale

Modules

- General Structure
  - Module name, port declarations, parameters (optional)
  - Declaration of wire and reg variables
  - Data flow statements (assign)
  - Instantiation of lower level modules
  - always and initial blocks (all behavioral statements)
  - Tasks and Functions
  - endmodule

Ports

- Port declarations – input, output, inout
- Port connection rules
  - there are two ends to a port with respect to a module, one internal and another external
  - Inputs – input ports are to be Nets inside the module and can be reg or net external to the module

Port Connection Rules

- Outputs – Internal reg or net and external should be a net
- Inouts – Both internal and external to be connected to a net
- Width matching – width may not match – only warning issued
- Unconnected ports – again only warning
  - fulladd4(SUM, , A,B, C_IN); the C_OUT is not included

Ports and external Connections

- Connecting by ordered list
- Connecting by name
  - wire C_OUT; wire [3:0] SUM;
  - reg [3:0] A,B; wire C_IN;
  - Fulladd4 f1 (.c_out(C_OUT), .sum(SUM), .a(A), .b(B), .c_in(C_IN));
- Syntax
  - .< name in instantiated module>(name in instantiating module)

Connecting by Name

- Advantages
  - Remembering order of say, 50 ports is difficult
  - Can drop any port during instantiation.
  - Can rearrange the port list of a module without modifying the code that instantiates it.