Verilog Session :
General Introduction to Verilog HDL

EE282

Fall Quarter, 2001

Verilog Hardware Description Language

Motivation for HDLs in general:

- Increased hardware complexity
- Design space exploration
- Inexpensive alternative to prototyping

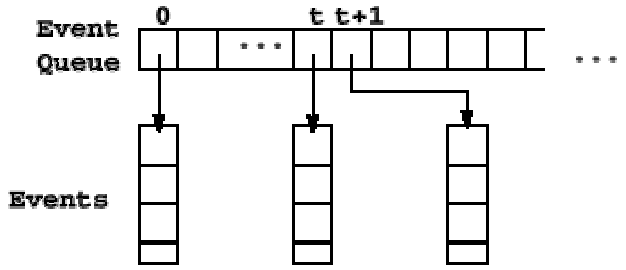
General Features:

- Support for describing circuit connectivity
- High-level programming language support for describing behavior
- Support for timing information (constraints, etc.)
- Support for concurrency

Verilog Simulation Model

- Verilog uses *event-driven* simulation.

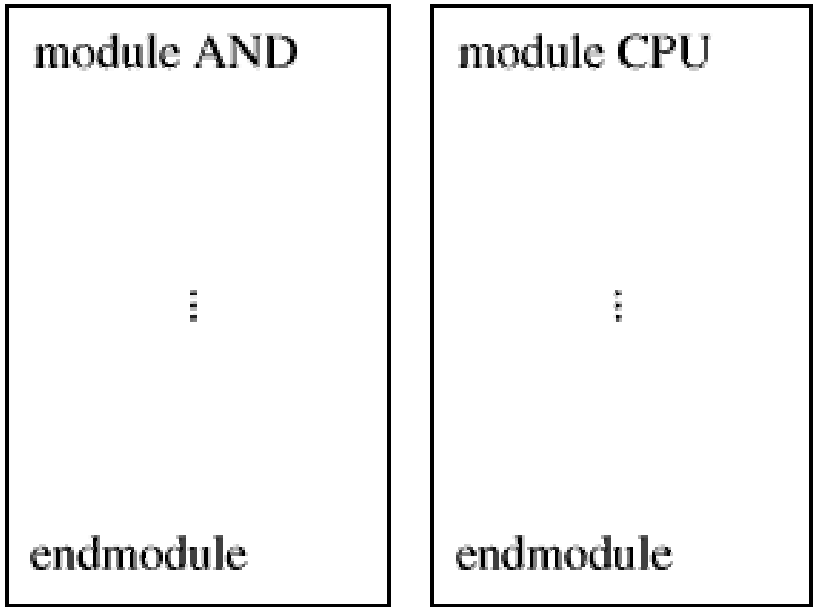
Event Driven Simulation:



- Simulation starts at time 0.
- When all the events at time *t* have been scheduled, the simulation clock advances to *t+1*.
- Simulation completes when there are no more events to process.

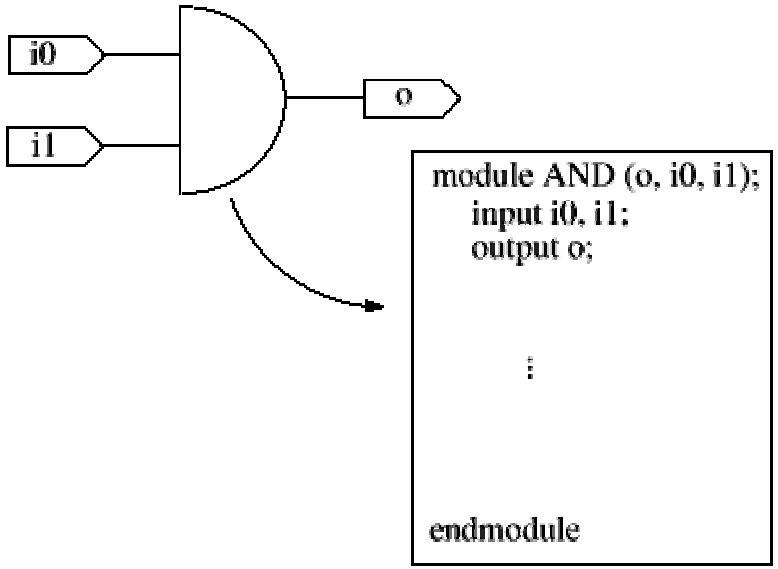
Key Verilog Features: Modules

- Modules are basic building blocks of a design description.
- Modules start with keyword *module* followed by the module name and end with the keyword *endmodule*.



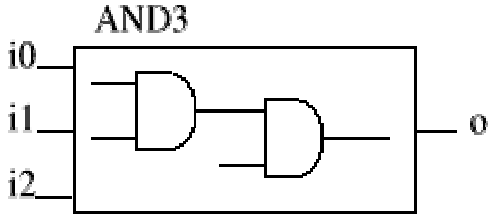
Key Verilog Features: Module Ports

- Module ports are similar to hardware pins: they provide means of communication between the module and the outside world.
- Ports can be *input*, *output*, or *inout* (bidirectional).



Key Verilog Features: Module Instances

- A Verilog model generally consists of a hierarchy of module *instances*.
- A module instance is not the same as a function call in high-level languages: each instance is a complete, independent, concurrent copy of a module.




```
module AND3 (o, i0, i1, i2);
        input i0, i1, i2;
        output o;
        wire temp;
        AND a0 (temp, i0, i1);
        AND a1 (o, temp, i2);
endmodule
```

Verilog Logic System

- *0* : zero, low, false, logic low, ground...
- *1* : one, high, true, logic high, power...
- *X* : unknown...
- *Z* : high impedance, unconnected, tri-state...

Verilog Data Types

Nets:

- Nets are physical connections between different devices.
- Nets always reflect the value of the driving device.
- Type of nets, we will be using exclusively - wire.

Registers:

- Contain implicit storage - unless a variable of this type is explicitly assigned/modified, it holds its previously assigned value.
- Register variables do not imply hardware registers.
- Main register type is *reg*. Another less important register type is *integer* which simply is equivalent to a 32 bit *reg*.

Verilog Variable Declaration

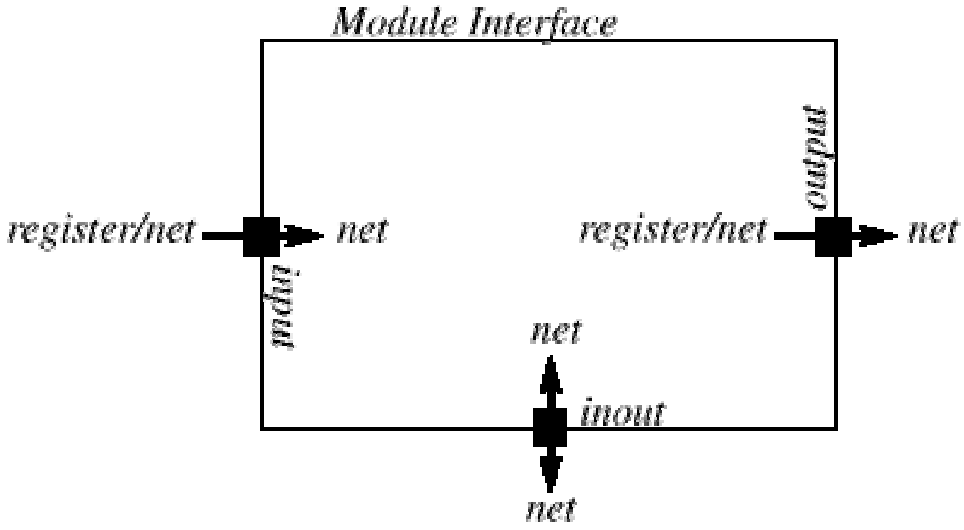
- **Declaring a *net***

  wire [<range>] <net_var> [<, net_var>*]
- **Declaring a *register***

  reg [<range>] <reg_var> [<, reg_var>*]
- **Samples**

  reg r; // 1-bit reg variable.
  wire w1, w2; // 2 1-bit wire variables.
  reg [7:0] vreg; // 8-bit register; least significant bit is 0, i.e. vreg[0].
  wire [0:11] vw1, vw2; // 2 12-bit nets; LSB is 11, i.e. vw1[11].
- Range information is specified as [MSB:LSB]. If no range exists, the corresponding variable has a bitwidth of one

Correct Data Typing for Port Connectivity

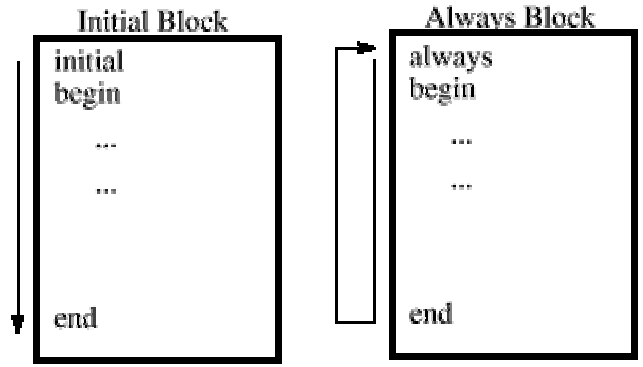


Behavioral Modelling in Verilog

- Involves system description at a high level of abstraction - using high-level language constructs.
- Emphasizes functionality primarily - independent of implementation.
- Specifies a set of concurrent *procedural* blocks, namely:
  1. *Initial* blocks, and
  2. *Always* blocks.
- Procedural blocks are built using the following constructs:
  1.Timing controls, and
  2. Procedural assignments,
  3. High-level language programming constructs.

Initial and Always Blocks

- Initial blocks execute *only once*.
- Always blocks execute continuously in a loop.
- Blocks can be a single statement or a compound statement. A compound statement is one or more single statements enclosed within a *begin...end* construct.



Timing Control in Procedural Blocks

- **Delays:**

  Used to delay the subsequent statement by a specified amount of time.

  #10 clk = 1;
  #10 clk = 0;

- **Triggering control:** @ (*trigger_event*)

  Delays execution until *trigger_event* changes or transitions. The *trigger_event* can be a signal/expression or multiple expressions linked using the keyword *or*. It is also possible to detect for particular transitions by using the keywords *posedge* or *negedge*.

  always @(posedge CLK) q = d;
  always @(i0 or i1) o = i0 & i1;

Procedural Assignments

- Regular assignments inside procedural blocks:

  lhs_expr = rhs_expr;
- The lhs_expr *must* be a variable of the type register.
- There are no constraints on data types included in the rhs_expr.

- *Common Error*:

  If a variable is not declared, it defaults to a 1-bit wire.

  Thus, if an undeclared variable appears on the left-hand-side of a procedural assignment, the Verilog compiler will generate an error message, “Illegal left-hand-side assignment”.

  So, use defensive code and declare all signals you read/write.

Operators in Verilog

- Arithmetic: + , - , * , / , % (2's complement)
- Binary bit-wise: ~ , & , | , ^ , ~^
- Unary reduction: & , ~& , | , ~| , ^ , ~^
- Logical: ! , && , || , == , === , != , !==
- Relational: < , < , >= , <=
- Logical shift: >> , <<
- Conditional: ?:
- Concatenation: {}

- **Common Error:**
  A lot of confusion arises between the "==" and "===" operators:
  == returns "x" if either if the input bits is "x" or "z" while === does compare "x"s and "z"s.

Conditional Statements

- **If, If-Else Statements:**

```
if (branch_flag > 0)
   begin
     PC = PCbr;
   end
else
   PC = PC + 4;
```

- **Case Statements.**

```
case (opcode)
    6'b001010 : write_mem = 1;
    6'b100011 : enable_alu = 1;
    default :
       begin
          $display("Unknown opcode: %h", opcode);
       end
endcase
```

- Could also use *casez* (compare *z* values) and *casex* (compare *z* and *x*).

Loop Constructs in Verilog

Repeat Loops:

- Repeats a block of statements a fixed number of times:

  repeat (<size>) <block>

For Loops:

- Same as in C:

  for (<loop_initialization>; <loop_condition>; <loop_update>) <block>

  for (memaddr = 0; memaddr < memsize; memaddr = memaddr + 1)
  $display("Memory at address 0x%h is 0x$h.", memaddr, memory[memaddr]);

- Loop executes while loop_condition evaluates to TRUE.

Continuous Assignments

- Procedural assignments are used to assign values into register type variables.
- Continuous assignments do the same for net type variables.
- Whenever the *right-hand-side* of the assignment changes, the *left-hand-side* is *automatically* and immediately updated to reflect that change.
- Generally, continuous assignments are used to model combinational logic or make a simple connection.

Sample Assignment.

```
assign o = i0 & i1;
```

Modelling Memory in Verilog

- Declaring memory:
  reg [<MSB>:<LSB>] <memory_var> [<start_addr>:<end_addr>];

- Memory addressing is done using indexing into the previously-defined memory array:

  L1Cache[16]...

- Verilog does not provide support for bit accesses into memory arrays.

- Loading memory arrays - using *$readmem* system task:

  $readmem<base> (“<file_name>”, <mem_var>[,<start>, <finish>]);
  <base> - refers to ‘b’ or ‘h’, i.e. binary or hex.
  <start> and <finish> - denote first and last addresses of memory array.

Verilog Lexical Conventions

Comments.

- // - begins with a // and ends with a newline (like C++).
- /* */ - like C.

Integers.

- Integers can be sized or unsized (unsized defaults to 32 bits).
- Sized integers have the following representation:

  <bit_size>’<base><value>
- Examples: 3’b101, 32’b1, 11’d97, 16’h1ff...

Identifiers.

- Identifiers provide user-defined names for Verilog objects.
- Identifiers must begin with a letter (a-z,A-Z) or underscore and can contain any alphanumeric character, underscore, or dollar sign.
- Examples: _bus8, b$c01...
- Illegal: 8bus, out_a%b...

Verilog Lexical Conventions

System Tasks and Functions.

- Representation: $<identifier>.
- $time - returns the current simulation time.
- $display - used for formatted printing like printf in C.
- $stop - stops simulation.
- $finish - ends simulation.
- $readmemh - load memory array from user's text file in hex format.

Compiler Directives.

- A compiler directive is immediately preceded by a grave accent (‘).
- ‘define - defines a compile-time constant or macro.
- ‘ifdef - ‘else - ‘endif - provide support for conditional compilation.
- ‘include - simple text inclusion.