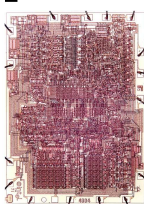


Verilog

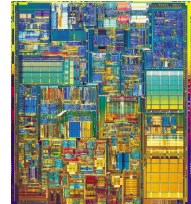
Prof. Abel Guilhermino

Aula X

Advancements over the years

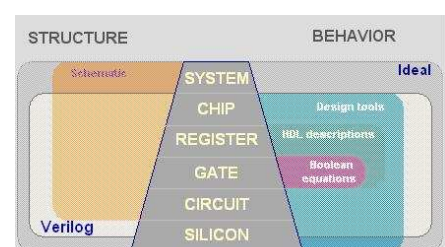


- © Intel 4004 Processor
- Introduced in 1971
- 2300 Transistors
- 108 KHz Clock



- © Intel P4 Processor
- Introduced in 2000
- 40 Million Transistors
- 1.5GHz Clock

System Design Pyramid



The diagram illustrates the System Design Pyramid, which is divided into two main categories: STRUCTURE and BEHAVIOR. The pyramid levels from top to bottom are: SYSTEM, CHIP, REGISTER, GATE, CIRCUIT, and SILICON. The left side (STRUCTURE) includes Schematic and Verilog. The right side (BEHAVIOR) includes Design tools, HDL descriptions, and Boolean equations. The top level is labeled 'Ideal'.

Introduction

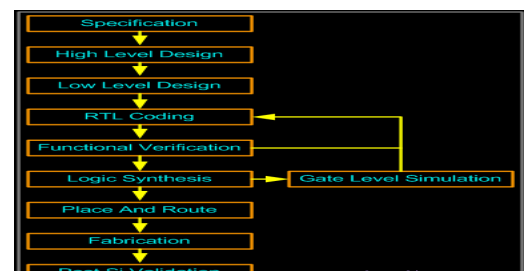
Purpose of HDL:

1. Describe the circuit in algorithmic level (like c) and in gate-level (e.g. And gate)
2. Simulation
3. Synthesis
4. Words are better than pictures

History:

- Need: a simple, intuitive and effective way of describing digital circuits for modeling, simulation and analysis.
- Developed in 1984-85 by Philip Moorby
- In 1990 Cadence opened the language to the public
- Standardization of language by IEEE in 1995

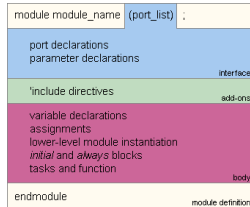
Top-Down Design Approach



The flowchart shows the following steps in a top-down design approach: Specification, High Level Design, Low Level Design, RTL Coding, Functional Verification, Logic Synthesis, Place And Route, Fabrication, and Post SI Validation. A feedback loop labeled 'Gate Level Simulation' connects the Logic Synthesis step back to the RTL Coding step.

Definition of Module

- Interface: port and parameter declaration
- Body: Internal part of module
- Add-ons (optional)

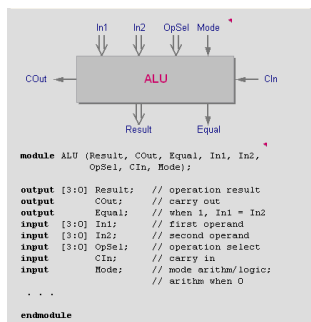


Some points to remember

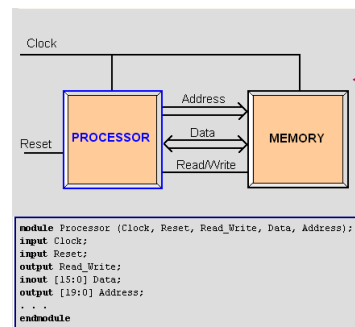
- The name of Module
- Comments in Verilog
 - One line comment (`//`)
 - Block Comment (`/*.....*/`)
- Description of Module (optional but suggested)

The Module Interface

- Port List
- Port Declaration



Specifications of Ports



Data Types

- Data Values:
 - 0,1,x,z
- Wire
 - Synthesizes into wires
 - Used in structural code
- Reg
 - May synthesize into latches, flip-flops or wires
 - Used in procedural code
- Integer
 - 32-bit integer used as indexes
- Input, Output, inout
 - Defines ports of a module (wire by default)

```

module sample (a,b,c,d);
    input a,b;
    output c,d;

    wire [7:0] b;

    reg c,d;

    integer k;
    
```

Data Values

- Numbers:
 - Numbers are defined by number of bits
 - Value of 23:
 - 5'b10111
 - 5'd23
 - 5'h17
- Constants:
 - wire [3:0] t,d;
 - assign t = 23;
 - assign d= 4'b0111;
- Parameters:
 - parameter n=4;
 - wire [n-1:0] t, d;
 - `*define` Reset_state = 0, state_B =1, Run_state =2, finish_state = 3;
 - if(state== Run_state)

Operators

- **Arithmetic:**
*,+,-,/,%
- **Relational**
<,<=,>,>=,==,!=
- **Bit-wise Operators**
 - Not: ~
 - XOR: ^
 - And: & 5'b11001 & 5'b01101 ==> 5'b01001
 - OR: |
 - XNOR: ~^ or ^~
- **Logical Operators**
Returns 1or 0, treats all nonzero as 1
 - ! : Not
 - && : AND 27 && -3 ==> 1
 - || : OR

```

reg [3:0] a, b, c, d;
wire[7:0] x,y,z;
parameter n =4;

c = a + b;
d = a * n;

if(x==y) d = 1; else d =0;

d = a ^~ b;

if ((x>=y) && (z)) a=1;
else a = !x;
    
```

Operators

- **Reduction Operators:**
Unary operations returns single-bit values
 - & : and
 - | : or
 - ~& : nand
 - ~| : nor
 - ^ : xor
 - ~^ :xnor
- **Shift Operators**
Shift Left: <<
Shift right: >>
- **Concatenation Operator**
{ } (concatenation)
{ n(item) } (n fold replication of an item)
- **Conditional Operator**
Implements if-then-else statement
(cond) ? (result if cond true) : (result if cond false)

```

module sample (a, b, c, d);
input [2:0] a, b;
output [2:0] c, d;
wire z,y;

assign z = ~| a;
c = a * b;
if(a==b) d = 1; else d =0;

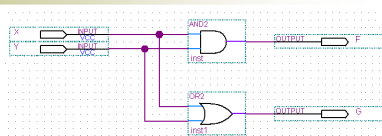
d = a ^~ b;

if ((a>=b) && (z)) y=1;
else y = !x;

assign d << 2; //shift left twice
assign (carry, d) = a + b;
assign c = {2{carry},2{1'b0}};
// c = {carry,carry,0,0}

assign c = (inc==2)? a+1:a-1;
    
```

Examples



```

module exmplo ( X, Y, F, G);

input X;
input Y;
output F,G;

assign F = X & Y;
assign G = X | Y;

endmodule
    
```

- Assigns are executed in parallel

Verilog Structure

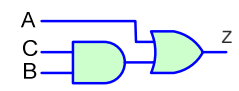
- All code are contained in modules
- Can invoke other modules
- Modules cannot be contained in another module

Verilog Structure

```

module gate(Z,A,B,C);
input A,B,C;
output Z;

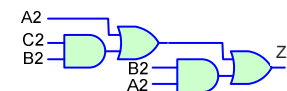
assign Z = A|(B&C);
endmodule
    
```



```

module two_gates(Z2,A2,B2,C2)
input A2,B2,C2;
output Z2;
wire G2;

gate gate_1(G2,A2,B2,C2);
gate gate_2(Z2,G2,A2,B2);
endmodule
    
```



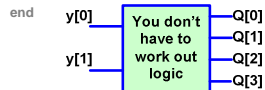
Structural Vs Procedural

| | |
|--|--|
| <p>Structural</p> <ul style="list-style-type: none"> ■ textual description of circuit ■ order does not matter ■ Starts with assign statements ■ Harder to code ■ Need to work out logic <pre style="font-size: small; margin-top: 10px;"> wire c, d; assign c = a & b; assign d = c b; </pre> | <p>Procedural</p> <ul style="list-style-type: none"> ■ Think like C code ■ Order of statements are important ■ Starts with initial or always statement ■ Easy to code ■ Can use case, if, for <pre style="font-size: small; margin-top: 10px;"> reg c, d; always@ (a or b or c) begin assign c = a & b; assign d = c b; end </pre> |
|--|--|

Structural Vs Procedural

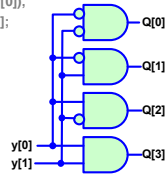
Procedural

```
reg [3:0] Q;
wire [1:0] y;
always@(y)
begin
  Q=4'b0000;
  case(y) begin
    2'b00: Q[0]=1;
    2'b01: Q[1]=1;
    2'b10: Q[2]=1;
    2'b11: Q[3]=1;
  endcase
end
```



Structural

```
wire [3:0]Q;
wire [1:0]y;
assign
  Q[0]=(!y[1])&(!y[0]),
  Q[1]=(!y[1])&y[0],
  Q[2]=y[1]&(!y[0]),
  Q[3]=y[1]&y[0];
```



Blocking Vs Non-Blocking

Blocking

- <variable> = <statement>
- Similar to C code
- The next assignment waits until the present one is finished
- Used for combinational logic

Non-blocking

- <variable> <=> <statement>
- The inputs are stored once the procedure is triggered
- Statements are executed in parallel
- Used for flip-flops, latches and registers



Do not mix both assignments in one procedure

Blocking Vs Non-Blocking

Initial

begin

#1 e=2;

#1 b=1;

#1 b<=0;

 e<=b; // grabbed the old b

 f=e; // used old e=2, did not wait

 e<=b