

# Verilog HDL, an introduction

## An IEEE Standard

Egbert Molenkamp  
Department of Electrical Engineering, Mathematics and Computer Science  
University of Twente  
PO Box 217  
7500 AE Enschede  
the Netherlands  
email: [molenkam@cs.utwente.nl](mailto:molenkam@cs.utwente.nl)



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

1

## Contents

- ➔ History
- ➔ Global differences between Verilog HDL and VHDL
- ➔ Modeling styles
- ➔ Test environment
- ➔ Data types
- ➔ Operators



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

2



## History

- ➡ 1983-1984: Verilog HDL was designed by Phil Moorby in only one Winter, at Gateway Design Automation.
- ➡ 1990: Gateway Design Automation is taken over by Cadence
  - Cadence's ownership → others support VHDL
- ➡ 1990: language is placed in public domain by OVI (Open Verilog International).
- ➡ 1992: Start of the standardization process as an IEEE standard
- ➡ 1995: Verilog HDL an IEEE standard, Std. 1364-1995.
- ➡ 2001: Verilog HDL; Std. 1364-2001
- ➡ 2005: Verilog HDL; Std. 1364-2005



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

3

## Global differences: Verilog versus VHDL

Verilog HDL	VHDL	
C-like	ADA-like (PASCAL, MODULA)	Verilog and VHDL are both suitable for RT Level
Loose typing	Strong typing	Better supports descriptions at a higher level of abstraction
No user defined type	User-defined types	
Delay mechanism support backannotation via SDF	Backannotation not in a standardized way.	<b>VITAL</b>
Switch level gates (pmos, etc)	No switch level gates	Better supports gate level simulation
Most ASIC design in Verilog	Most FPGA design in VHDL	



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

4

## An example

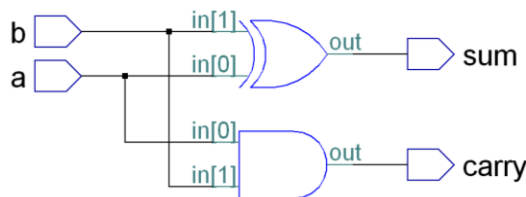
compiler directive: time unit is 1 ns, and precision 100 ps

```
// an example
`timescale 1ns/100ps
module HalfAdder(a,b,carry,sum);
  input a, b;
  output carry, sum;

  assign #2 sum = a ^ b;
  assign #3 carry = a & b;
endmodule
```

- case sensitive!
- keywords in lower case
- no separation between external and internal view

continuous assignment statement



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

5

## Verilog

```
// an example
`timescale 1ns/100ps
module HalfAdder(a,b,carry,sum);
  input a, b;
  output carry, sum;

  assign #2 sum = a ^ b;
  assign #3 carry = a & b;
endmodule
```

Verilog 1995

```
// an example
`timescale 1ns/100ps
module HalfAdder(
  input a,b,
  output carry,sum);

  assign #2 sum = a ^ b;
  assign #3 carry = a & b;
endmodule
```

Added in Verilog 2001



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

6

## Modeling styles

- ➔ Dataflow
  - Continuous assignment statements; order independent (see previous slide for an example)
- ➔ Behavioral
  - Initial statement; executes only once
  - Always statement; executes repeatedly
- ➔ Structural
  - Built-in gate primitives (gate level)
  - Switch-level primitives (transistor level)
  - User-defined primitives (UDP, at gate level)
  - Module instantiations
- ➔ Mixture allowed

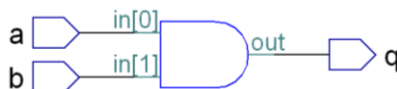


© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

7

## behavioral style

- ➔ Always statement is sensitive to a change of 'a' or 'b'
- ➔ Only *reg* data types can be assigned a value in an *initial* and an *always* statement. This data type remains its value until a new value is assigned
- ➔ Added in Verilog 2001
  - Comma separated sensitivity list allowed
  - More efficient I/O interface



```

module demo(a,b,q);
  input a, b;
  output q;
  reg z,q;

  always @(a or b) begin
    z = a & b;
    q = z;
  end
endmodule
  
```

```

module demo_verilog2001_style(
  input a, b,
  output reg q);
  reg z;

  always @(a, b) begin
    ...
  end
endmodule
  
```

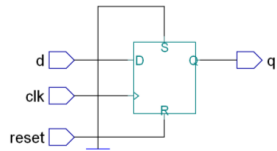


© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

8

## behavioral style

- ➔ @(posedge clk or reset)  
Mixing of edge and level sensitive not supported for synthesis
- ➔ @(posedge clk or posedge reset)  
OK for synthesis

<pre>always @(posedge clk or reset) begin   if (reset)     q = 0;   else     q = d; end</pre> <p style="text-align: right; color: blue;"><b>Is this a flipflop?</b></p>	<pre>always @(posedge clk or posedge reset) begin   if (reset)     q = 0;   else     q = d; end</pre> 
---	--

Synthesis tool reports:  
Error, Illegal mixing of level and edge triggers

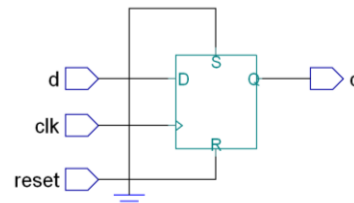
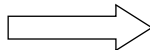


© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

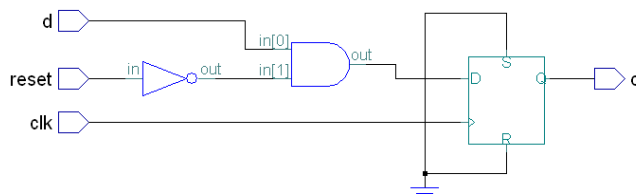
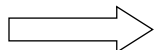
9

## Asynchronous/synchronous reset

```
always @(posedge clk or posedge reset) begin
  if (reset)
    q = 0;
  else
    q = d;
end
```



```
always @(posedge clk) begin
  if (reset)
    q = 0;
  else
    q = d;
end
```



Note: 'begin' and 'end' not needed if there is one statement



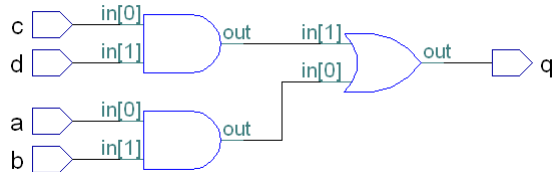
© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

10

## Sensitive to all variables read (Verilog 2001)

```
`timescale 1ns/100ps
module all_in_list (a,b,c,d,q);
  input a,b,c,d;
  output q;
  reg z1,z2,q;

  always @* begin
    z1 = a & b;
    z2 = c & d;
    q = z1 | z2;
  end
endmodule
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

11

## delays in behavioral style

### Delayed assignment

z = a & b;  
#2 p = z | c;

and

or

**VHDL:**  
z <= a and b;  
wait for 2 ns;  
p <= z or c;

The 2 time unit specifies that the second statement is executed after 2 time units after the first statement.

### Intra-assignment delay

z = a & b;  
p = #2 z | c;

**VHDL:**  
z <= a and b;  
p <= z or c after 2 ns;

The second statement is executed At the same time as the first statement. The assignment to p is delayed 2 time units.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

12

```

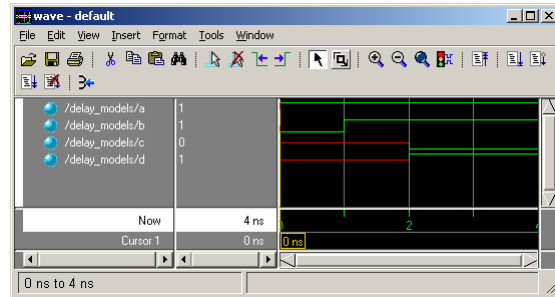
module delay_models (a,b,c,d);
  output a,b,c,d;
  reg a=1, b = 0, c, d;

  initial b = #1 1;

  initial c = #2 a & b;

  initial begin
    #2 d = a & b;
  end
endmodule

```



Added in Verilog 2001: “reg a=1” initial value assigned to object a  
“begin end pair ” not needed if there is 1 statement.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

13

## Structural

- ⇒ Built-in gates, e.g. and, nand, xor. The first entry is the output. These built-in gates can have 2 or more inputs.
- ⇒ Label is optional.
- ⇒ After the name of built-in gates multiple instantiations may occur. Separated with ',', and the last with ','.
- ⇒ Gate instantiations can appear in any order.
- ⇒ 'wire' is optional.
- ⇒ Comment:
  - Single line //
  - Multiple lines /\*..\*/
- ⇒ Verilog 2001
  - default net types can be disabled with:  
'default\_nettype none'

```

module HalfAdderS(a,b,carry,sum);
  input a, b;
  output carry, sum;
  wire carry; //optional, is default

  xor
    (sum,a,b);
  and
    a1 (carry,a,b);
endmodule

```

built-in gate

label

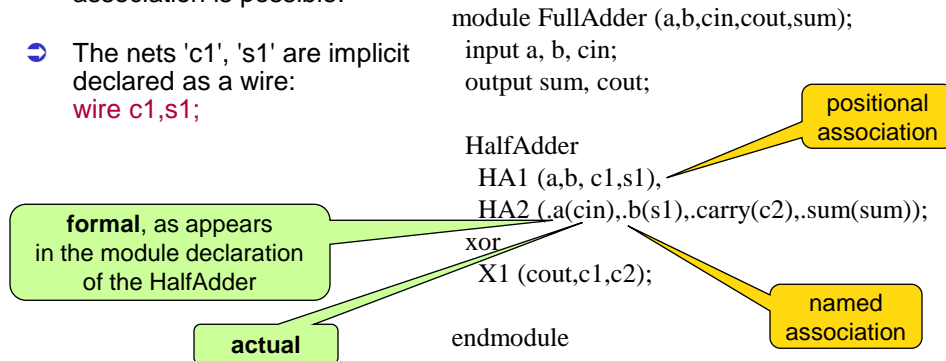


© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

14

## Structural description of a full adder

- *Named and positional association is possible.*
- The nets 'c1', 's1' are implicit declared as a wire:  
**wire c1,s1;**



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

15

## Test environment for the full adder

- An *initial* or *always* statement requires a label if there are *local declarations*.
- Implicit type conversion for 'inp'. It has three bits, but has also integer interpretation.
- Language has 'system tasks' and 'system functions'. They start with the character '\$'.
- This example uses an initial statement. When will it stop?

```

module TestFullAdder;
  reg a,b,cin;
  wire carry,sum;

  FullAdder
    FA(a,b,cin,carry,sum);

  initial
    begin: label_required_local_declaration
      reg [2:0] inp;
      for (inp=0;inp<=7;inp=inp+1)
        begin
          {a,b,cin}=inp;
          #5 $display ("a,b,cin=%b%b%b",a,b,cin,
            " carry,sum=%b%b",carry,sum);
        end
      end
end
endmodule

```

**formatting**



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

16



## Test environment for the full adder /2

- ➔ The for loop:  
(inp=0;inp<=7;inp=inp+1)  
increments the 3 bits  
register 'inp'. After "111" it  
will be "000" again!

- ➔ Work around:  
Change the declaration of  
inp in "reg [3:0] inp;"

```
# a,b,cin=000 carry,sum=00
# a,b,cin=001 carry,sum=01
# a,b,cin=010 carry,sum=01
# a,b,cin=011 carry,sum=10
# a,b,cin=100 carry,sum=01
# a,b,cin=101 carry,sum=10
# a,b,cin=110 carry,sum=10
# a,b,cin=111 carry,sum=11
# a,b,cin=000 carry,sum=00
# a,b,cin=001 carry,sum=01
# a,b,cin=010 carry,sum=01
```

Why doesn't  
it stop?



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

17

## Types

- ➔ Built-in value set
  - 0, 1, x, z
  - x and z are case insensitive
- ➔ Integer
  - simple decimal, e.g. 30, -30
  - base format, e.g. 4'H6 (hex), 5'O23 (octal), 3'B01 (binary)
    - signed number (e.g. 4'H-6) is not allowed
    - in case of extension, zero's are used (or x or z in case left bit is x resp. z)
- ➔ Time
  - Time c; c=\$time; //current simulation time
- ➔ Real
- ➔ String
  - a character is represented as an 8-bit ASCII value (interpreted as an unsigned number).

5 bit octal



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

18

## arrays

reg[31:0] data1;

a register of 32 bits.  
data1=32'b0101

reg data2 [31:0];

a memory with 32 addresses each 1 bit.  
data2=32'b0101 **ILLEGAL**

reg[15:0] memory [0:63];

memory[0]=16'b1;  
smart built-in function to fill memory from file  
\$readmemb("filename",memory);

- Arrays can have not more than 2 dimensions. In Verilog 2001 multi dimensional array are allowed.
- Are interpreted as unsigned number.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

19

## integer

- An integer can not be accessed as a bit vector.  
Solution, assign to a reg register.  
integer int;  
reg [4:0] tmp;  
tmp=int;
- Integers are implemented using 2-complement representation.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

20

## Signed/unsigned (added in Verilog 2001)

➡ In Verilog 2001

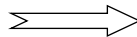
- Objects can be declared as **signed**
- system function **\$signed** and **\$unsigned** added (casting)

```
module Adder(a,b,s1,s2,s3);
  input signed [1:0] a,b;
  wire [2:0] c;
  output [2:0] s1,s2,s3;
```

```
  assign c=b;
  assign s1 = a + b; // signed + signed
  assign s2 = a + c; // signed + unsigned
  assign s3 = a + $signed(c); // signed + signed
```

```
endmodule
```

If A=11 and B=00 then  
S1=111  
S2=011  
S3=111



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

21

## Operators

lower  
precedence  
↓

+	Unary add
-	Unary minus
!	Unary logical negation
~	Unary bit-wise neg.
&	Reduction and
~&	Reduction nand
~^ of ^~	Reduction xnor
^	Reduction xor
	Reduction or
~	Reduction nor
<hr/>	
*	Multiply
/	Divide
%	Modules
<hr/>	
+	Binary plus
-	Binary minus
<hr/>	
<<	Left shift
>>	Right shift

<	Less than
<=	Less or equal
>	Greater than
>=	Greater or equal
<hr/>	
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
<hr/>	
&	Bit wise and
<hr/>	
^	Bit wise xor
<hr/>	
~^ or ^~	Bit wise xnor
<hr/>	
	Bit wise or
<hr/>	
&&	Logical and
<hr/>	
	Logical or
<hr/>	
?	Conditional operator



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

22

## Delay

- Verilog has a detailed delay mechanism, e.g.
  - and #4 (o,i1,i2);  
rise and fall delay are 4 time units.
  - and #(3,5) (o,i1,i2);  
rise delay is 3, and fall delay is 5. Transition to x is minimum of the values.
  - bufif1#(3,5,4) (o,i,sel);  
a buffer gate with tri-state, rise delay is 3, fall delay is 5 and turn-off delay is 4. Transition to x is minimum of the values.
- Also min, typ, or max can be used, e.g.
  - and #(1:2:3,...) and (o,i1,i2);  
rise delay: minimal 1, typical is 2 and max is 3.
  - the selection is an option of the simulator.



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

23

## blocking, non-blocking procedural assignment

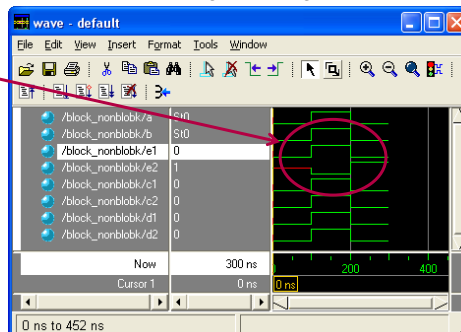
```
module block_nonblock (a,b,e1,e2);
  input a,b;
  output e1,e2;
  reg c1,c2,d1,d2,e1,e2;
```

```
  always @ (a,b) begin
    c1=a;
    d1=b;
    e1=c1&d1;
  end
```

```
  always @ (a,b) begin
    c2<=a;
    d2<=b;
    e2=c2&d2;
  end
```

```
endmodule
```

- In a blocking (=) assignment the statement has to be finished before the next statement is executed. E.g. in the left example. A and b will both have the value of a.
  - VHDL: similar to (shared) variable assignment
- In case of a non-blocking (<=) assignment all statements are executed 'at the same time'.
  - VHDL: similar to signal assignment



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

24

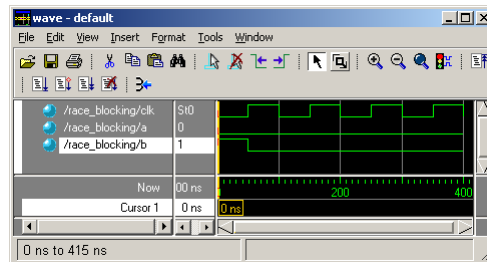
## Race condition

```
module race_blocking (clk,a,b);
input clk;
output a,b;
reg a=0,b=1;
```

```
always @ (posedge clk)
b = a;
```

```
always @ (posedge clk)
a = b;
endmodule
```

- A “race condition” can occur when blocking assignment are used.
  - In this example b is first assigned the value of a before a is assigned the value of b  
→ order dependent !



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

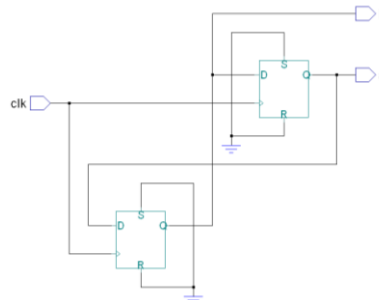
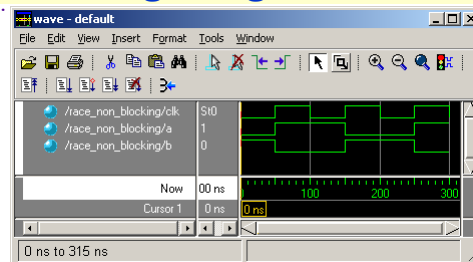
25

## No race condition with non blocking assignment

```
module race_non_blocking (clk,a,b);
input clk;
output a,b;
reg a=0,b=1;
```

```
always @ (posedge clk)
b <= a;
```

```
always @ (posedge clk)
a <= b;
endmodule
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

26

## Generic description added in Verilog 2001

- ➔ Parameter: constant that can be changed when the module is instantiated
- ➔ Localparams are constants that can not be redefined
- ➔ Generate/endgenerate added

```
module multiplier (a,b,q);
  parameter twocomplement = 1;
  parameter width = 4;
  localparam product_width = 2*width;
  input [width-1:0] a,b;
  output [product_width-1 : 0] q;
  reg [product_width-1 : 0] q;

  generate
    if (twocomplement==1)
      always @(a,b) q=$signed(a)*$signed(b);
    else
      always @(a,b) q=a*b;
  endgenerate

endmodule
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

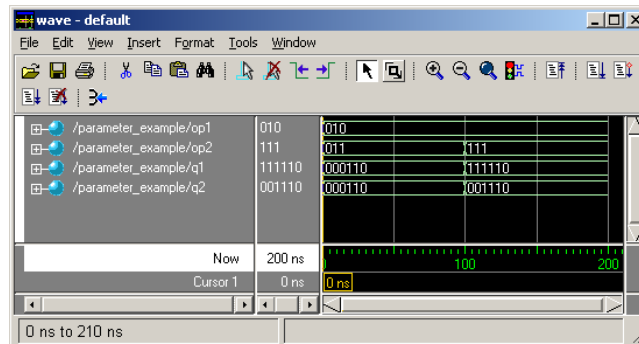
27

## Instantiation of generic modules

```
module parameter_example (op1,op2,q1,q2);
  localparam w = 3;
  input [w-1:0] op1,op2;
  output [2*w-1:0] q1,q2;
```

- ➔ <module><parameters><label><I/Os>
- ➔ Named and positional association possible

```
  multiplier #(.twocomplement(1),.width(w)) sign(op1,op2,q1);
  multiplier #(0,w) unsign(op1,op2,q2);
endmodule
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

28

## Other topics, e.g.

- Case statement: don't care values are interpreted correctly ("casex").
- Loop construct: forever, repeat, while, for
- User-defined primitives (UDP)
- Programming Language Interface (PLI). Interface with a foreign language.
- Functions (Verilog 2001 also supports recursive functions)

length of the result

```
function [nbits-1:0] number_of_ones;
input [nbits-1:0] inp;
integer J;
begin
    number_of_ones = 0;
    for (J=0; J<nbits; J=J+1)
        if (inp[J]) number_of_ones = number_of_ones + 1;
    end
endfunction
```



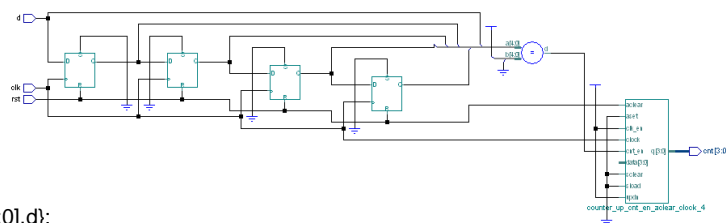
© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

29

## Pattern recogniser

```
module patreq (d,clk,rst,cnt);
input d,clk,rst;
output [3:0] cnt;
reg [3:0] cnt;
```

```
always @(posedge(clk) or posedge(rst))
begin:label_required_local_decl
    reg [4:0] pattern;
    if (rst)
    begin
        pattern=5'b0;
        cnt=4'b0;
    end
    else
    begin
        pattern={pattern[3:0],d};
        if (pattern==5'b11100) cnt=cnt+1;
    end
    end
endmodule
```



© 1989-2012 E. Molenkamp, University of Twente, the Netherlands, Verilog

30