HDL for PLD and CPLD Lab: Verilog

http://www.igs.uni-rostock.de/ -> Intranet -> Scripte
or Stud.IP (http://studip.uni-rostock.de/)

HDLs started as verification instruments

Verilog ← Verifying logic

Developed in U.S. since 1984/85 (Fa. Gateway Design Automation) -> Phil Moorby

1989 Takeover by Cadence Design Systems

Proprietary until early 90th
First standard (IEEE-1364) in 1995
New standard in 2001 → Verilog 2001
2002 -> System Verilog ->
IEEE Std 1800.2005 -> Verilog 2005
2009 Verilog + SystemVerilog -> Verilog 2009
Latest revision: IEEE 1800-2017

Note documenting the name search for the simulation language

https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/archive/2016/04/21/phil-moorby

Books

D. Thomas, Phil Moorby : The Verilog Hardware Description Language
Kluwer, ISBN 0-7923-8166-1

Peter J. Ashenden: Digital Design An Embedded Systems Approach using Verilog
Morgan Kaufmans, ISBN 978-0-12-369527-7

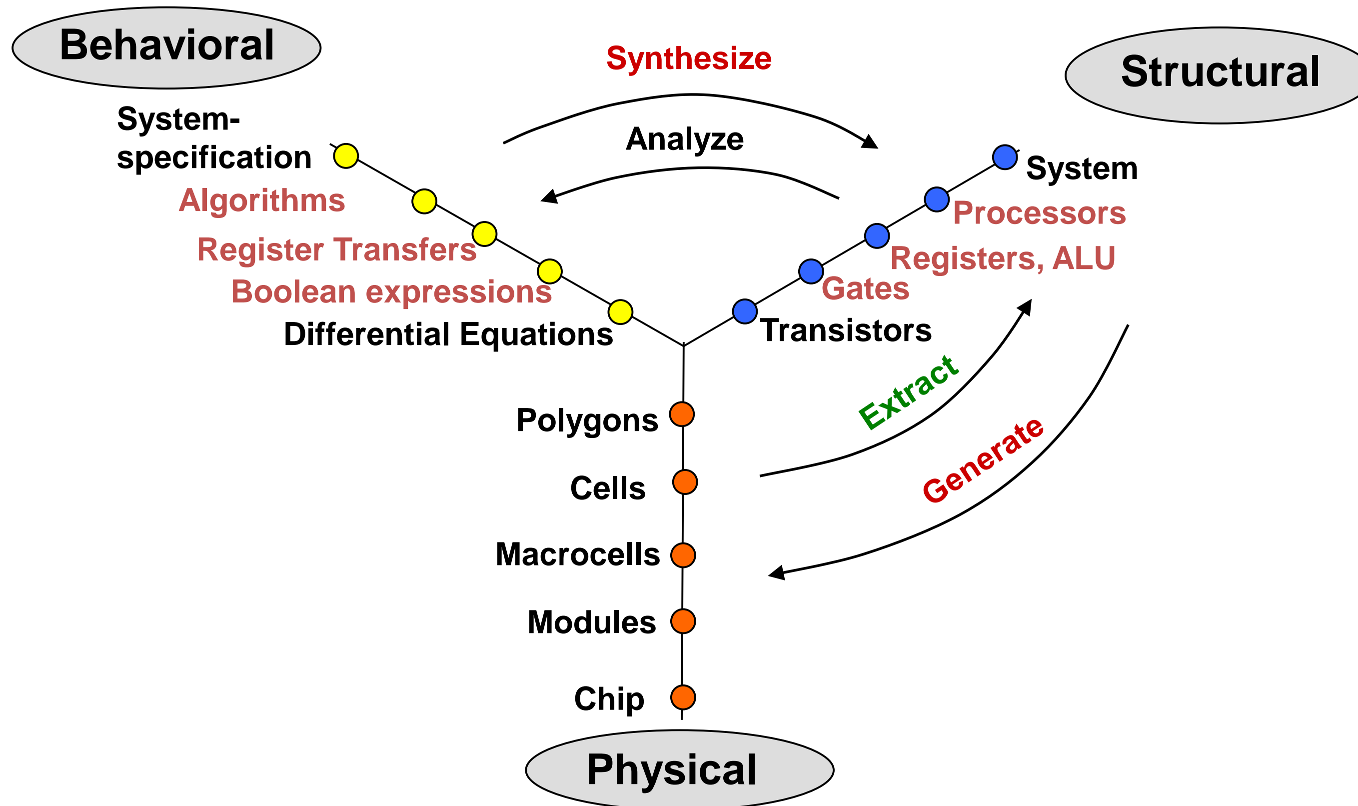
Michael D. Ciletti: Advanced Digital Design with the VERILIOG HDL
Prentice Hall, ISBN 01131678442

Stuart Sutherland u. a.: SystemVerilog for Design
Springer Science+Business Media, ISBN 9780387333991
http://sutherland-hdl.com/online_verilog_ref_guide/verilog_2001_ref_guide.pdf

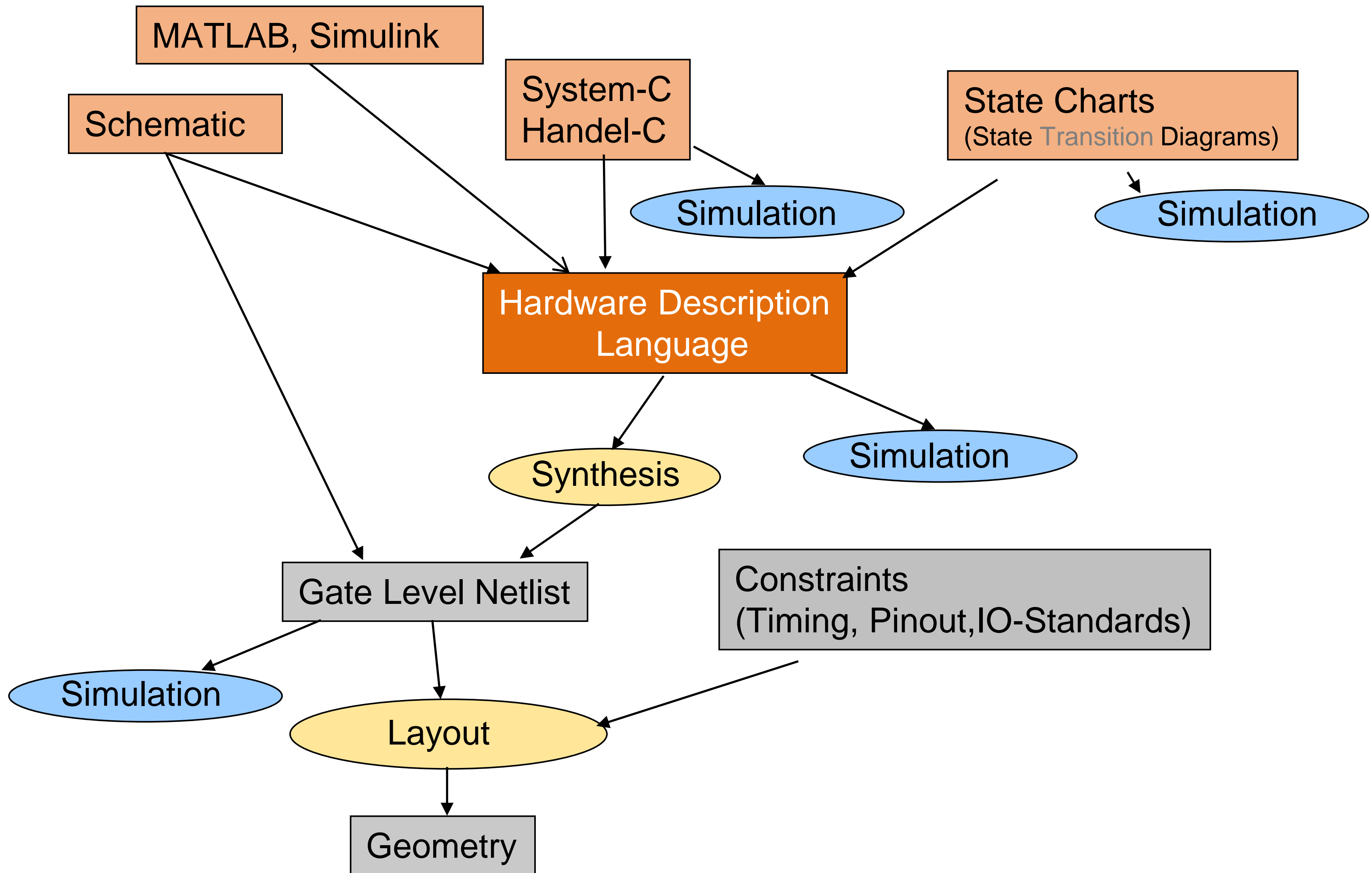
Bernhard Hoppe : Verilog
Oldenbourg 2006, ISBN 9783486580044

Harald Flügel: FPGA-Design mit Verilog
Oldenbourg 2010, ISBN 9783486592344

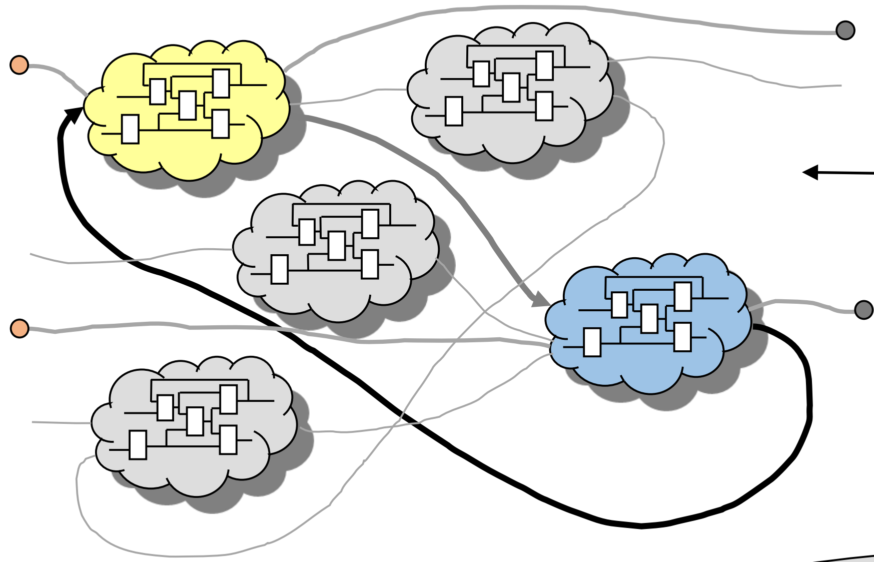
Verilog and System Verilog in the Design Environment



Design Views and Levels of Abstraction (Gajski and Kuhn)


Design Entry
MATLAB, Simulink
Schematic
System-C
Handel-C
State Charts
(State Transition Diagrams)
Simulation
Simulation
Hardware Description Language
Synthesis
Simulation
Gate Level Netlist
Constraints
(Timing, Pinout,IO-Standards)
Simulation
Layout
Geometry

Execution of a simulation cycle in Event Driven Simulation



What to specify first ?
yellow or blue module ?

HDL must describe parallelism

New Events generated internally

External Event(s)

Update Signals at Inputs

Delta-t

Sum Delta-t = 0

Parallel Execution of Processes

Updated Input Values at Processes

End if no new events generated at the moment

Some global remarks on syntax

Identifiers in Verilog are **case sensitive** (in VHDL they are not!)

Identifiers start with a letter or underscore -> Do not use **keywords** even with different case

Comments: // for a line until newline or blocks included in /* */ (in VHDL comment starts with two hyphens)

Size always in bits, not in digits!-> Fill with MSB or truncation possible!

Numbers: size's formatvalue

optional for signed,
unsigned is default
MSB sign extension!

8'b1010_0001
161
8'd161
'ha1 'hA1

Optional underscore for readability
(not at first position -> identifier)

'b , 'B , 'h , 'H , 'd , 'D , 'o , 'O

(both upper- and lowercase possible for base and hex digits)

Strings: "text" may include: \n, \t, \", \\, \ddd -> (newline, tab, quota, backslash, character_octal_code)

Vector slices use squared brackets and colon: a[7:0]

Macros: `define name(parameters) value; Use as: `name(arguments)

Module

A module is a model in Verilog, mainly a piece of (digital) hardware.

Module includes:
- Name of the model
- Port Declarations (Inputs and Outputs)
- Parameters
- Signal Declarations
- Structural or Behavioural Logic Descriptions

separate portlist, portmode, datatype

```
module  mux (out0, in0, in1, sel);
            output out0;
            input  in0,in1,sel;

wire out0, in0,in1,sel;

assign out0 = (sel==0) ? in0 : in1;

endmodule
```

true : false

Verilog 1995

combined portlist, portmode, datatype

```
module  mux (
            output wire out0,
            input wire in0,in1);

assign out0 = (sel==0) ? in0 : in1;

endmodule
```

Verilog 2001

Signals: Wire and Reg

Nets (**wire**) are connections only (no driver included)
Must: connect outputs of module instantiations
left side of assign
Must not: left side in initial and always@

There exist other net data types (tri, wor, trior, wand …)
-> resolution of parallel drivers, undriven signals

net declaration ->

```
wire  clk;
tri [7:0] dbus;
```

Control of implicite net declaration:
`default_nettype *net_data_type* <- standard is wire
`default_nettype *none* <- no implicit declarations

With Verilog-2001 implicit net declaration available and standard

Registers (**reg**) store a value (include a driver, not necessarily a memory)
Must: **left side in initial and always@**
creation of registers
Must not: left side of assign

System Verilog provides **logic** to overcome the ambiguous term **reg**

reg declaration ->

```
reg [9:0] counter;
```

Other reg types: integer (32 bit, signed), time (64 bit unsigned), real

Constants: Parameter

Example from Lattice 4000ZE Oscillator
OSCOUT: Nominal Clock 5 MHz
TIMEROUT: Divided Clock
DYNOSCDIS: Disable/Save Power
TIMERRES: Reset
TIMER_DIV: 128,1k,1024k

Parameters may be overwritten during module instantiation
Used as constants

```
parameter   width = 8;
parameter   hallo = "Hallo";
parameter   vect1 = 8'b1000_1111;
```

In VHDL constants are static inside an entity.
VHDL uses generics for parameterizing entities

```
module osctimer(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT);
        parameter TIMER_DIV = "128";
        input DYNOSCDIS;
        input TIMERRES;
        output OSCOUT;
        output TIMEROUT;
endmodule
```

Here we use an oscillator module from the hardware library.
Parameter applies to the HW-block.
Parameter applies to the Simulation model.

dot -> .local(actual) <- may remain empty for outputs (no connection)

```
OSCTIMER I1 (.DYNOSCDIS(1'b0), .TIMERRES(RST), .OSCOUT(CLK_F), .TIMEROUT(CLK_S));

defparam    I1.TIMER_DIV = "1024";
```

<- module instance parameter value assignment

named connection -> in VHDL "named association"

```
OSCTIMER #(.TIMER_DIV("1024")) I1 (.DYNOSCDIS(1'b0), .TIMERRES(RST), .OSCOUT(CLK_F), .TIMEROUT(CLK_S));
```

verbose -> safe

ordered connection -> in VHDL "positional association"

```
OSCTIMER #("1024") I1 ( 1'b0, RST, CLK_F, CLK_S);
```

less verbose -> dangerous!

Operators 1

Arithmetic Operators

Fill with zero from right, don't affect sign (MSB)

- +
- -
- /
- *
- **
- % modulo
- <<< shift arithmetic
- >>> shift arithmetic

-3 % 2 // result: -1

Use sign (MSB) to fill from left

Comparasion Operators

- ==
- !=
- >
- <
- >=
- <=
- === Include X,Z
- !== Include X,Z

2'b1x == 2'b1x // result: x
2'b1x === 2'b1x // result: 1

Bitwise Operators (applied separately to each bit position of the operands)

- ~a invert
- a & b and
- a | b or
- ^ xor
- ~^ (^~) xnor
- a << b shift a left b times
- a >> b shift a right b times

Fill vacant positions with 0

Unary Reduction (applied to all bits of unary operand (vector))

- &a
- ~&a
- |m
- ~|m
- ^
- ~^ (^~)

both variants

a[n] & a[n-1] & … & a[0]

XOR/XNOR -> Parity calculation

Operators 2

Logical Operators (Test for true/false)		
! a	a && b both a and b true?	a || b either a or b true ?
Miscellaneous Operators		
sel?a:b conditional : if sel is true return a else b	{a1,a2} concatenation	{b{a}} replicate concatenation, replication (b instances of "a" chained up)

!! VHDL: a1 & a2

[k]

assign

Discuss!: assignment vs. equation

assign continuous statement -> combinatorial logic, simple assignments, (concatenation, slice splitting)

delay — true : false

```
assign  #10 out                          = (sel==0)?  in0 : in1;              //Multiplexer
assign  #(2:3:4),(2:3:4) output_1        = internal_signal2;                  //Simple assignment
assign  #(2:3:4),(2:3:4) ,(5:5:5) output_2 = (enable) ? int_sigl2 : 1'bz;     //Tristate buffer
assign  vec_long                         = {vec_short_1, vec_short_2};        //Concatenation
assign  vec_long                         = {4{vec_short}}                     //Replication
assign  vec_short                        = vec_long[3:0];                     //Slice
```

left side must be a net (e.g. wire)

Delay values for event driven simulation — not for synthesis!

#(2:3:4),(2:3:4),(5:5:5) min:typ:max
rising_edge, falling_edge, to_tristate

see also SDF-files !

Wire Assignment = Combination of wire declaration and assignment: wire y123 = x1 & x2;
Beginning with Verilog 2001 there is an implicit wire declaration (can be disabled by compiler directive)

Processes (initial, always)

always

Behavioral description of parallel hardware

Sensitivity list: wait for an event at …

Whenever there happens something in the sensitivy list do the following …

```
reg [7:0] clk_count;

//Clock counter
always @(posedge clk)
begin
        if (clk_count == 203) clk_count <= 0;
        else clk_count <= clk_count + 1;
end
```

```
reg y;
wire a,b;

//Combinatorial logic
always @ (a,b)   // @* possible
begin
        y <= a & b;
end
```

y must be a reg type

"always" contains sequential statements -> sum of execution times is 0

initial

Initialization for simulation, description of waveforms

```
.timescale 1ns/1ns;   // time unit/precision

initial
 begin
     x1 = 0;
     #10 x1 = 1;
     #10 x1 = 0;
 end
```

times: relative (delays)

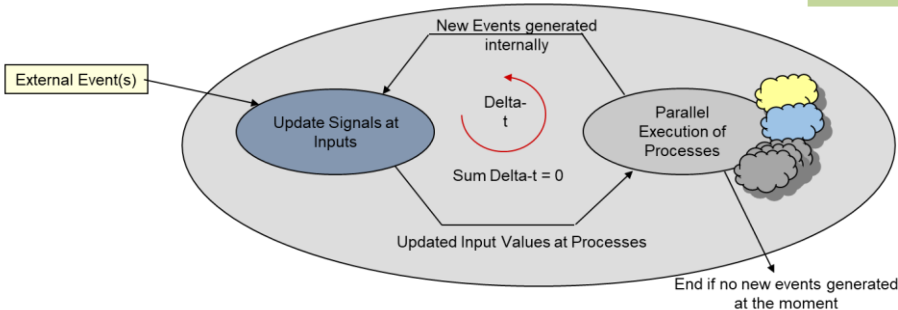
Blocking and Nonblocking Assignments

Use complete sensitivity lists here (including all "right side wires") -> * is a good idea

```
always @(*)
a[1] = a[0];
a[2] = a[1];
end
```

Blocking assignment evaluates and assigns before continuing.
Evaluation and assignment is a single step.
a[2] has the value of a[0] after clock edge

Use for combinatorial logic (including inertial delay)



VHDL: Behaviour of Variables

```
always @(posedge clk)
a[1] <= a[0];
a[2] <= a[1];
end
```

value after clk-edge before

Nonblocking assignment evaluates but assigns when all other evaluations are made in the actual time step.
Evaluation and assignment done in two steps
a[2] gets the value of a[1] after clock edge

Use it for sequential logic.
Use for combinatorial logic (including transport delay)

VHDL: Behaviour of Signals

Parallel processes fork-join

Especially for testbenches

```
fork : main
          begin : T1
              code of thread1
          end
          begin
              code of thread2
          end
           …
join
```

join - wait for completion of all threads
join_any - waits for 1st thread in time
join_none - starts threads only and continues

System Verilog

wait fork - wait for all threads to finish
disable fork t_name - kill threads (all or by name)

Creates parallel execution
Can be made from multiple initial- or always- processes

Control Structures

```
reg state [1:0];
parameter [1:0] zero =2'b00, one =2'b01, two =2'b10, three =2'b11;
```

```
if   (input1==2'b01)
              begin
                  y1 <= x1 + x2;
                  y2  = x1 + x2;
              end
else if  (input1==2'b10)  y1 <= X3;

else          y1 <= X4;
```

```
case (state)
  one:  begin
            if (key_pressed)  state <= two;
              else state <= zero;
        end
  two:     state <= three;


  default:  state <= one;
endcase
```

values only
no relational operators
no intervals "from to"

more: casez, casex
z, resp. z and x as don't care for choices
(See also operators == and ===)

Loops

for-loop

The loop runs through the structure here

```
always @ (x1,x2)
        for  (i=1; i<=8; i=i+1)
        y[i] = x1[i] ^ x2[8-i];
```

"i" may be manipulated inside loop

forever

```
forever  #10 clock = ~clock
```

repeat

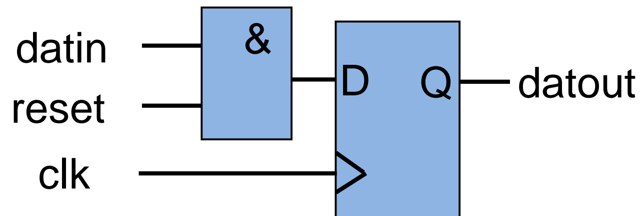
```
repeat (8)  #10 clock = ~clock
```

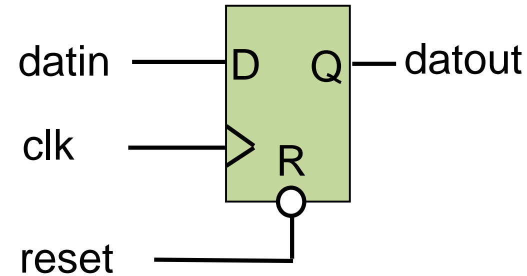
while

```
while (counter <=8)
        begin
        #10 clock = ~clock
```

Exit from loop: disable

Flip-Flops asynchronous and synchronous Reset






```
always @ (posedge clk )
        begin
                if (!reset)
                datout <= 0;
                else
                datout <= datin;
        end
```

```
always @ (posedge clk , negedge reset)
        begin
                if (!reset)
                datout <= 0;
                else
                datout <= datin;
        end
```

Module Instantiation (Structural Descriptions)

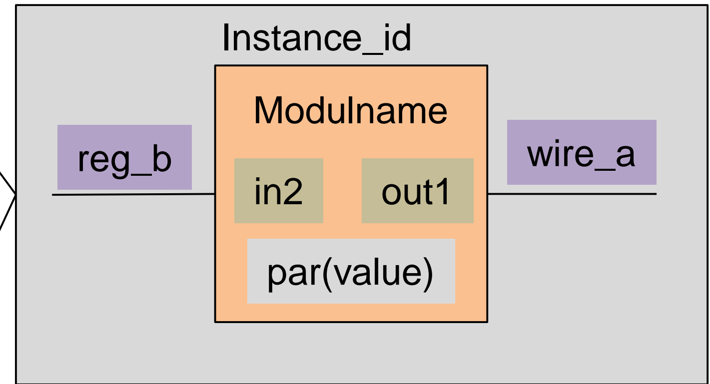
#(.par(value),..)

Named association — local — actual

Modulname Instance_Id (.out1(wire_a), .in2(reg_b), …)

Port out1 of the module connected to actual wire_a

outputs always connected to wires!



Positional association

Modulname Instance_Id (wire_a, reg_b, …)

actual

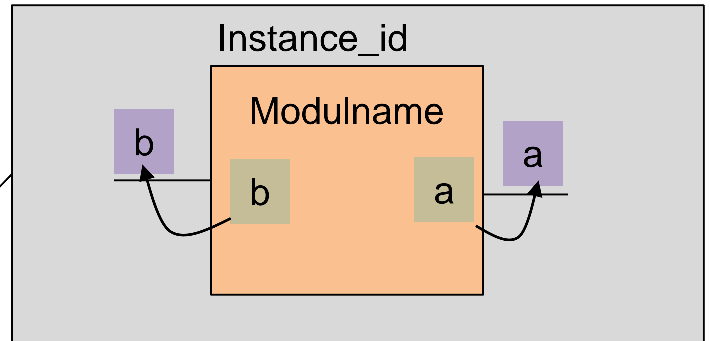
Often used in text books, but not recommended

First port of the module connected to actual wire_a

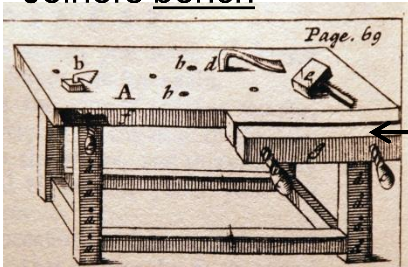
Implicit Names — local = actual — System Verilog

Modulname Instance_Id (.a, .b)

Identical local and actual names possible to mix with named ass.
Very short: Instance_Id (.*)



Joiners bench



Fixture (vise am./vice br.)

Testfixture/Testbench

Simulation environment
Neither inputs nor outputs
-> VHDL: Testbench

module design_tf ();

UUT Unit Under Test
DUT Device Under Test
DUV Device Under Verification

Description of stimulating signals

clk

always

#10 clk = ~clk;

clk

clkin

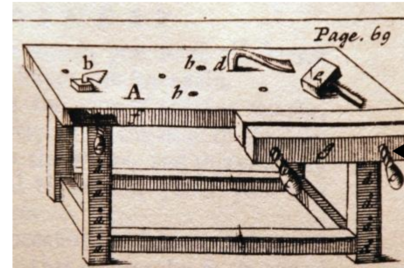
UUT

Unit Under Test

?

?

Testfixture/Testbench



Fixture (vise$_{am.}$/vice$_{br.}$)

Joiners bench

Simulation environment
Neither inputs nor outputs
-> VHDL: Testbench

Netlist, HDL, SDF for timing

Testfixture

Generation of Test Signals (Stimuli)

Inputs

Unit **U**nder **T**est
Design **U**nder **T**est

UUT / DUT

Internal values

Outputs

Value lists in files

Waveform display

Compare with expected values

Testfixture Example

time_unit / precision

To connect primary inputs From TF to UUT

To connect primary outputs From UUT to TF

named connection

local actual

```
`timescale 1 ns / 1 ns

module my_design_tf();

reg clk, input1, input2 ;
wire output1, output2 ;

my_design UUT (  clk(clk),
                .local_input1(input1),
                .local_input2(input2),
                .local_output1(output1),
                .local_output2(output2));
initial     begin
            clk=0; input1=0; input2=1;
            #500  input1= 1;
            #1000 $finish
            end;
always      begin
            #10 clk=~clk;
            end
endmodule
```

using positional connection

```
my_design UUT (clk ,input1, input2,
               output1, output2);
```

Waveform Generation

Stop simulation and exit simulator

$... - System task
($display, $stop, $write, $monitor, $random …
File Handling: $fopen, $fwrite .

Pin Assignment in Verilog

Special comments evaluated by tool chain

Portability violation

Better: Use (tool dependent) constraint files and/or constraint editor

Exemplar from Mentor

```
input         inA3; //exemplar attribute inA3 LOC PA3
output [0:2]  sout; //exemplar attribute sout LOC PF8PA2PB3
```

"Comment appended to statement"

Simple space to separate multiple attributes
Do NOT repeat "synthesis" keyword

NONE if drive strength selection not supported
or a number like 4, 8, 12 … for drive current in mA, if supported

Synplicity/Symplify from SYNOPSYS

```
input         inA3  /* synthesis LOC= "PA3" " PULL="OFF"*/;
output [0:2]  sout  /* synthesis LOC= "PF8PA2PB3" IO_TYPES="LVTTL, NONE" SLEW="SLOW"*/ ;
```

Line comment should work too: // synthesis …

Used by Lattice ispLEVER in our Lab

Tasks and Functions

VHDL: Procedures and Functions

Task is instantiated as a statement by use of its name
Replaces parts in a procedure

my_task (x1, x2, y);

```
task my_task;
        input-declarations;
        output-declarations;
        statements;
endtask
```

statements may change output values

no "return", compare VHDL

Function is used as an operand in an expression
Returns a value
Function name is pseudovariable of appropriate type

y <= my_function (x1, x2);

```
function  my_function;
        input-declarations;
        statements;
endfunction
```

pseudovariable with the type of y

Software Download

www.latticesemi.com

Register -> login

Support -> licensing -> "ispLEVER Classic Software"
You need your computers MAC-ID xx-yy-zz-aa-bb-cc

Products -> Software Tools -> ispLever Classic (Only "ispLEVER 2.0 Base Module" and "Active-HDL simulation libraries")

www.xilinx.com

Register -> login