

Programmierbare Integrierte Schaltungen II

Application Specific Integrated Circuits (ASICs)

Me

Klaus-Peter Kirchner

University of Rostock

Faculty of Electrical Engineering and Information Technology

Contact:

Dr. Klaus-Peter Kirchner, Südstadt H III, Room 30

Phone 0381 498 7210

klaus-peter.kirchner@uni-rostock.de

Skript/Materials (several parts):

Stud-IP -> Course 24175 -> Documents

Campus Südstadt H III, Room S08 (WS-Pool)

Drive N:\asic1\...

www.igs.uni-rostock.de

-> Intranet (ITMZ-account) -> Skripte



History and Motivation

VHDL

VHSIC Hardware Description Language

Very High Scale/Speed Integrated Circuits

Developed in U.S. since 1981 (Initiated by US Department of Defense)

First Standard (IEEE-1076) in 1987

Revision in 1993 → VHDL-93

Revision in 2002 → VHDL-2002 (minor changes, relaxed buffer use)

Revision in 2008 → VHDL-2008 (new generics, PSL)

Standard IEEE-1076.1 includes analog circuits (VHDL-AMS)

Property Specification Language
IEEE 1850 (Verification, Assertions)

Analog Mixed Signal

Description of digital hardware for simulation

Usable from gate to system level

Behavioural (and structural) description

Hardware independent

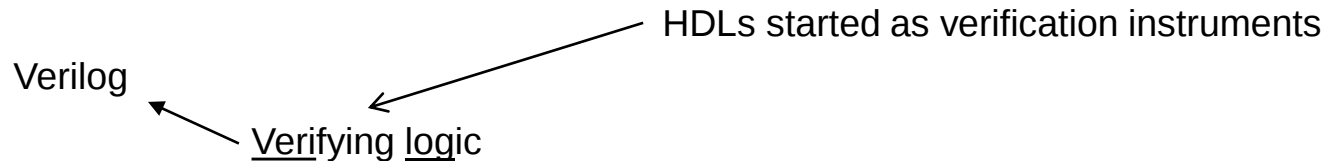
Portable

Top down and bottom up methodology supported

Readable ;-)

Synthesis tools became available (subset of VHDL)

Verilog



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

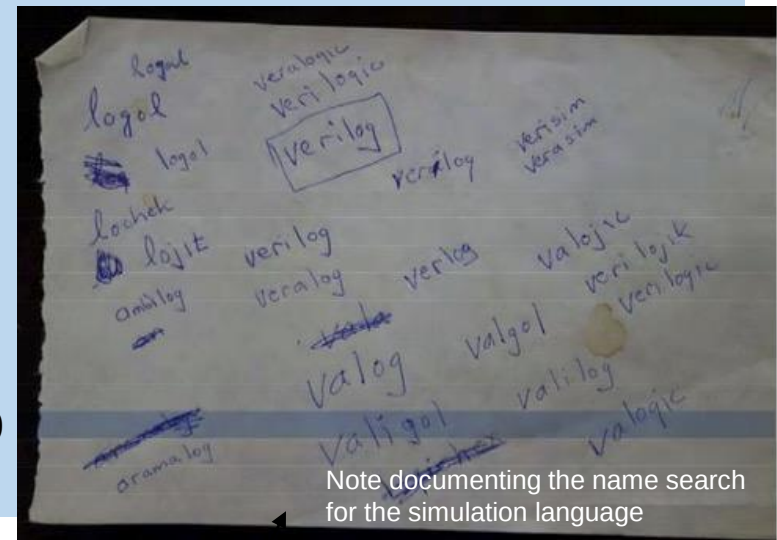
New standard in 2001 → Verilog 2001

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

Peter J. Ashenden: The Designers Guide to VHDL, 3rd edition
Morgan Kaufmann Publishers, ISBN 978-0-12-088785-9

Ashenden, Peterson, Teegarden:
The System Designer's Guide to VHDL-AMS
Morgan Kaufmann Publishers, ISBN 1-55860-893-1

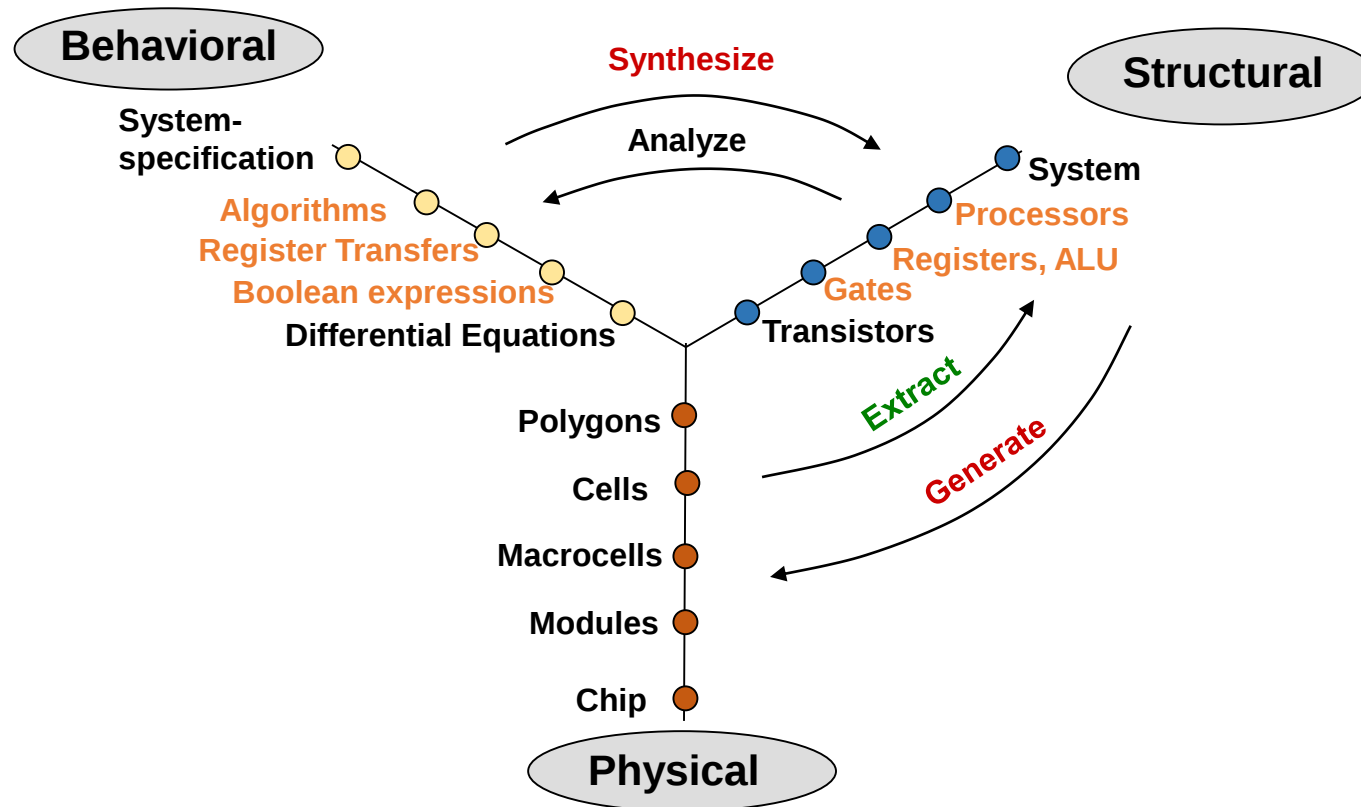
Peter J. Ashenden: VHDL-Cookbook
tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf

Weng Fook Lee : VHDL Coding and Logic Synthesis with SYNOPSYS
Academic Press, ISBN 0-12-440651-3

S. Yalamanchili: Introductory VHDL From Simulation to Synthesis
Prentice Hall, ISBN 0-13-080982-9

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

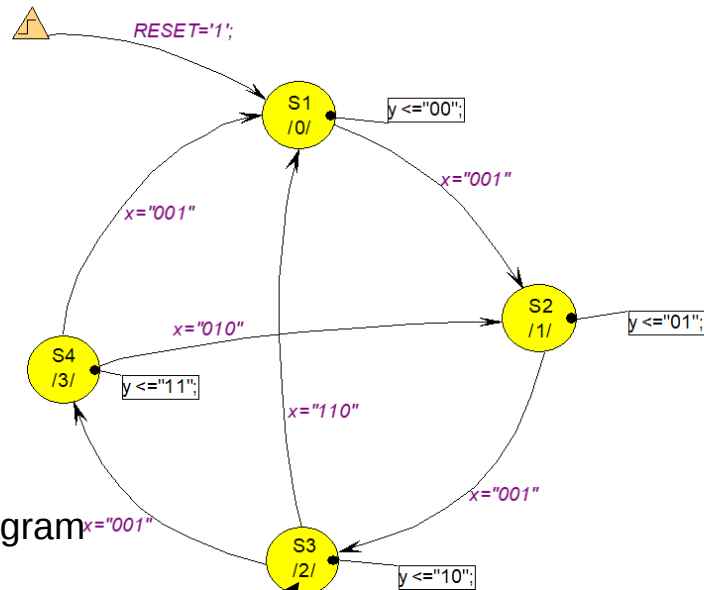
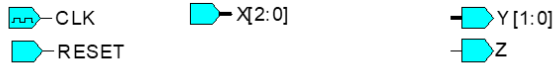
VHDL in the Design Environment



Design Views and Levels of Abstraction
(Gajski and Kuhn)

Simple FSM (Finite State Machine)

$Z \leq X(1) \text{ and } X(2);$



State diagram

Later we learn how to influence encoding:
 ATTRIBUTE enum_encoding : STRING;
 ATTRIBUTE enum_encoding OF Sreg0_type : TYPE IS "00 01 10 11";

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity fsm1 is
    port (CLK, RESET: in STD_LOGIC;
          X: in STD_LOGIC_VECTOR (2 downto 0);
          Y: out STD_LOGIC_VECTOR (1 downto 0);
          Z: out STD_LOGIC);
end entity fsm1;
```

architecture fsm1_arch of fsm1 is

```
type Sreg0_type is (S1, S2, S3, S4); --Enumeration type
signal Sreg0: Sreg0_type; --Signal defined by new type
```

begin

```
Z <= X(1) and X(2);
```

```
Y <= "01" when (Sreg0 = S2) else
      "10" when (Sreg0 = S3) else
      "11" when (Sreg0 = S4) else "00";
```

Sreg0_machine: **process** (CLK)

begin

```
if CLK'event and CLK= '1' then --if rising_edge(CLK)
```

```
if RESET='1' then Sreg0 <= S1;
```

```
else
```

```
case Sreg0 is
```

```
when S1 => if x="001" then Sreg0 <= S2; end if;
```

```
when S2 => if x="001" then Sreg0 <= S3; end if;
```

```
when S3 => if x="110" then Sreg0 <= S1;
```

```
elseif x="001" then Sreg0 <= S4; end if;
```

```
when S4 => if x="010" then Sreg0 <= S2;
```

```
elseif x="001" then Sreg0 <= S1; end if;
```

```
when others => null;
```

```
end case;
```

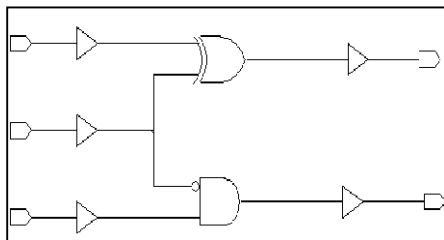
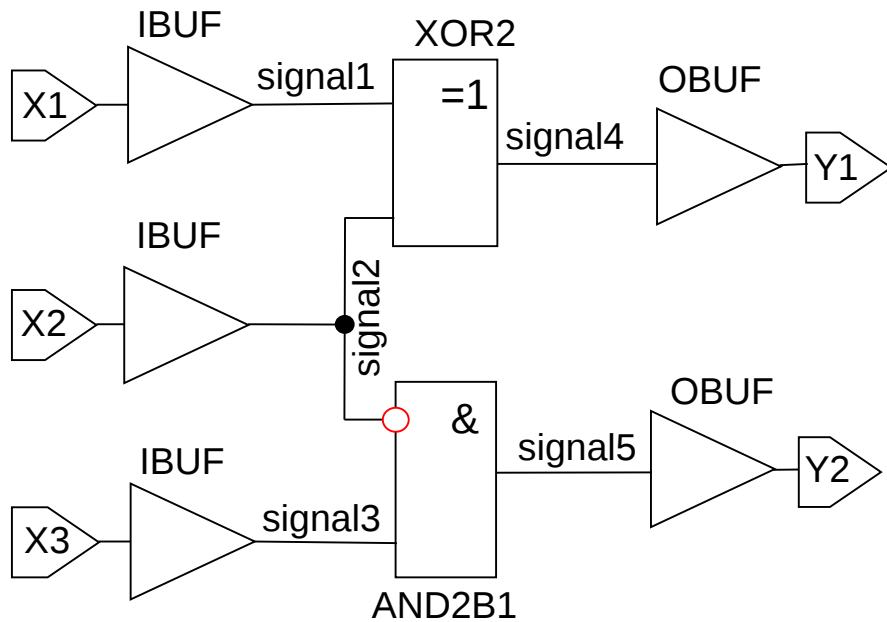
```
end if;
```

```
end if;
```

```
end process;
```

end architecture;

Schematic and Structural VHDL



```
LIBRARY ieee;
LIBRARY UNISIM;
USE ieee.std_logic_1164.ALL;
USE UNISIM.Vcomponents.ALL;
```

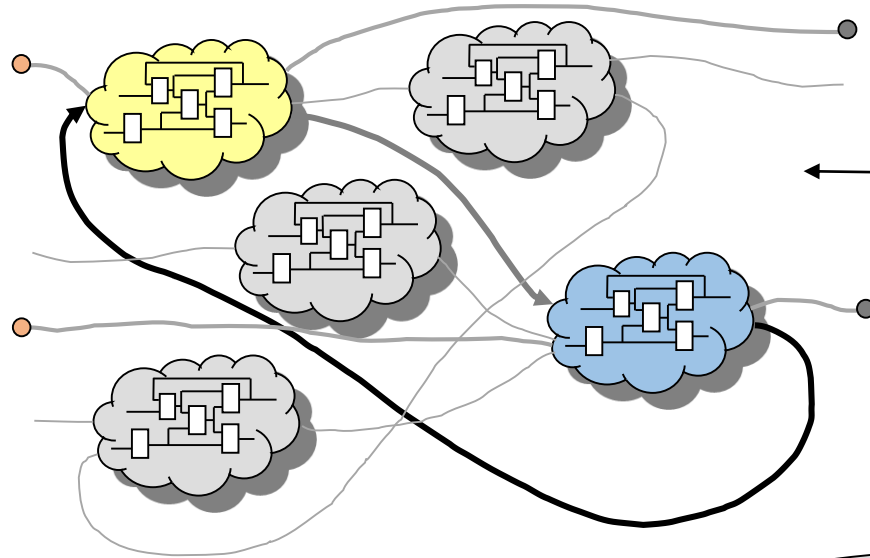
```
ENTITY logic1 IS
  PORT ( X1,X2,X3 : IN STD_LOGIC;
        Y1, Y2 : OUT STD_LOGIC);
end logic1;
```

```
ARCHITECTURE SCHEMATIC OF logic1 IS
  SIGNAL signal1, signal2, signal3, signal4, signal5: STD_LOGIC;
  BEGIN
```

```
  XLXI_6 : AND2B1
    PORT MAP (I0=>signal3, I1=>signal2, O=>signal5)
  XLXI_1 : IBUF
    PORT MAP (I=>X1, O=>signal1);
  XLXI_2 : IBUF
    PORT MAP (I=>X2, O=>signal2);
  XLXI_3 : IBUF
    PORT MAP (I=>X3, O=>signal3);
  XLXI_4 : OBUF
    PORT MAP (I=>signal4, O=>Y1);
  XLXI_5 : OBUF
    PORT MAP (I=>signal5, O=>Y2);
  XLXI_7 : XOR2
    PORT MAP (I0=>signal1, I1=>signal2, O=>signal4);
```

```
END ARCHITECTURE SCHEMATIC;
```

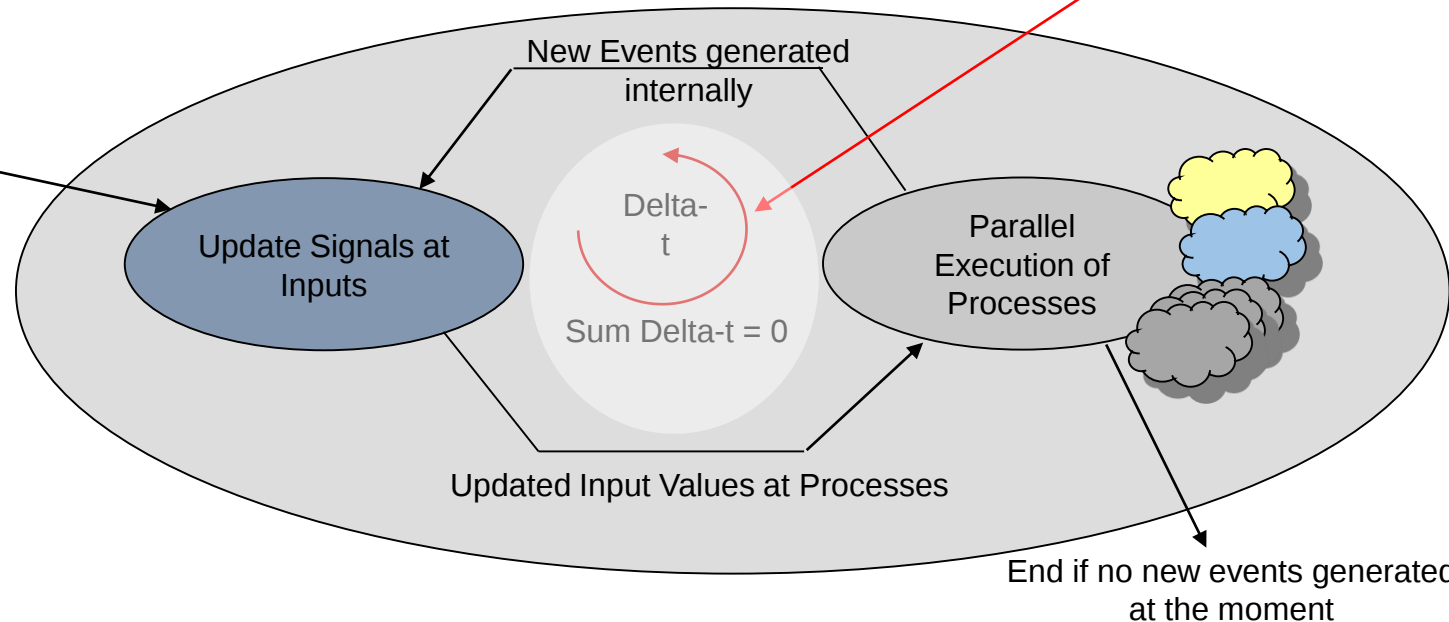
Simulation: Logic Simulation Parallelism and Simulation Cycle



What to specify first ?
yellow or blue module ?

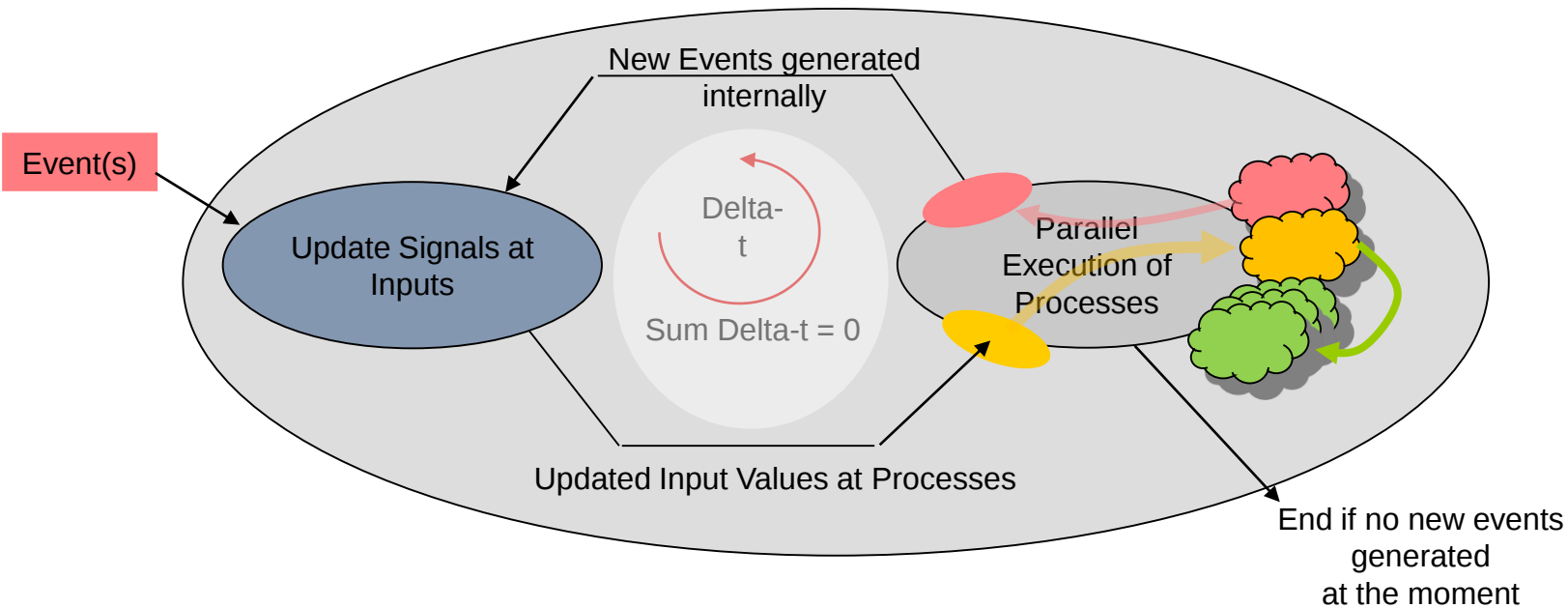
calculating concurrence/simultaneity with
a sequentially operating simulator machine

External Event(s)
occurring in "real time"



Simulation: Logic Simulation Parallelism and Simulation Cycle

Activation of processes and new internal events



Process with new signal value(s) at input (active)



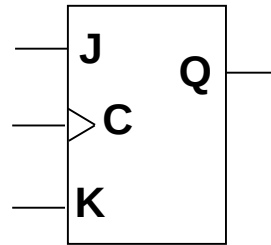
Process does not change its output (quiet)



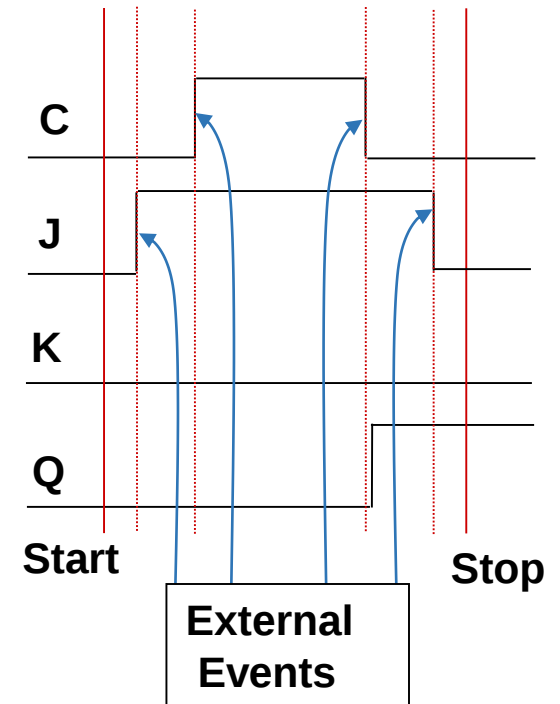
Process does changes output its output (switching)

Example: JK-MS-Flip-Flop

J	K	Q(n+1)
0	0	Q(n)
0	1	0
1	0	1
1	1	/Q(n)

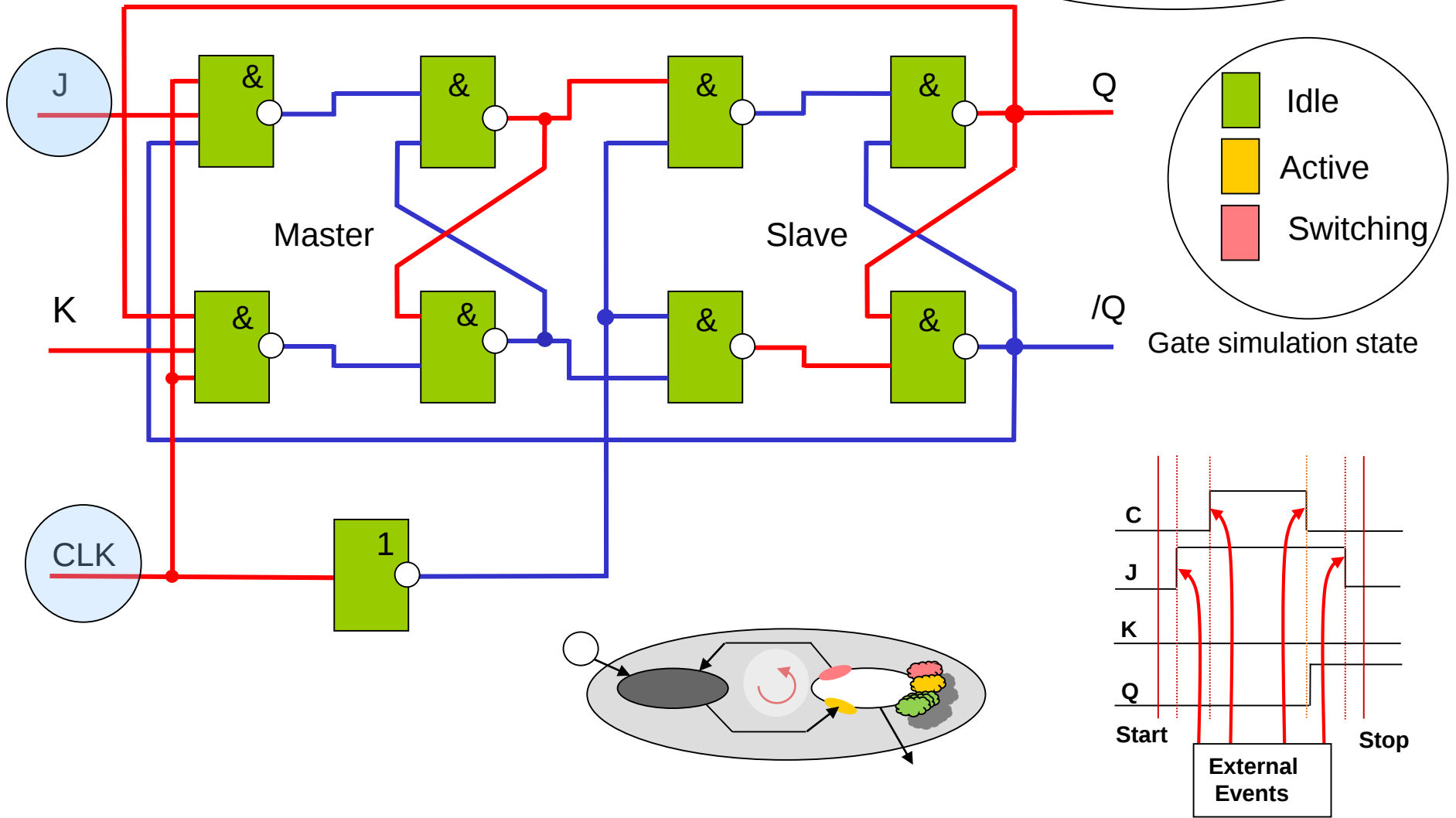


Q(n)	Q(n+1)	J	K
0	0	0	X
0	1	1	X
1	1	X	0
1	0	X	1

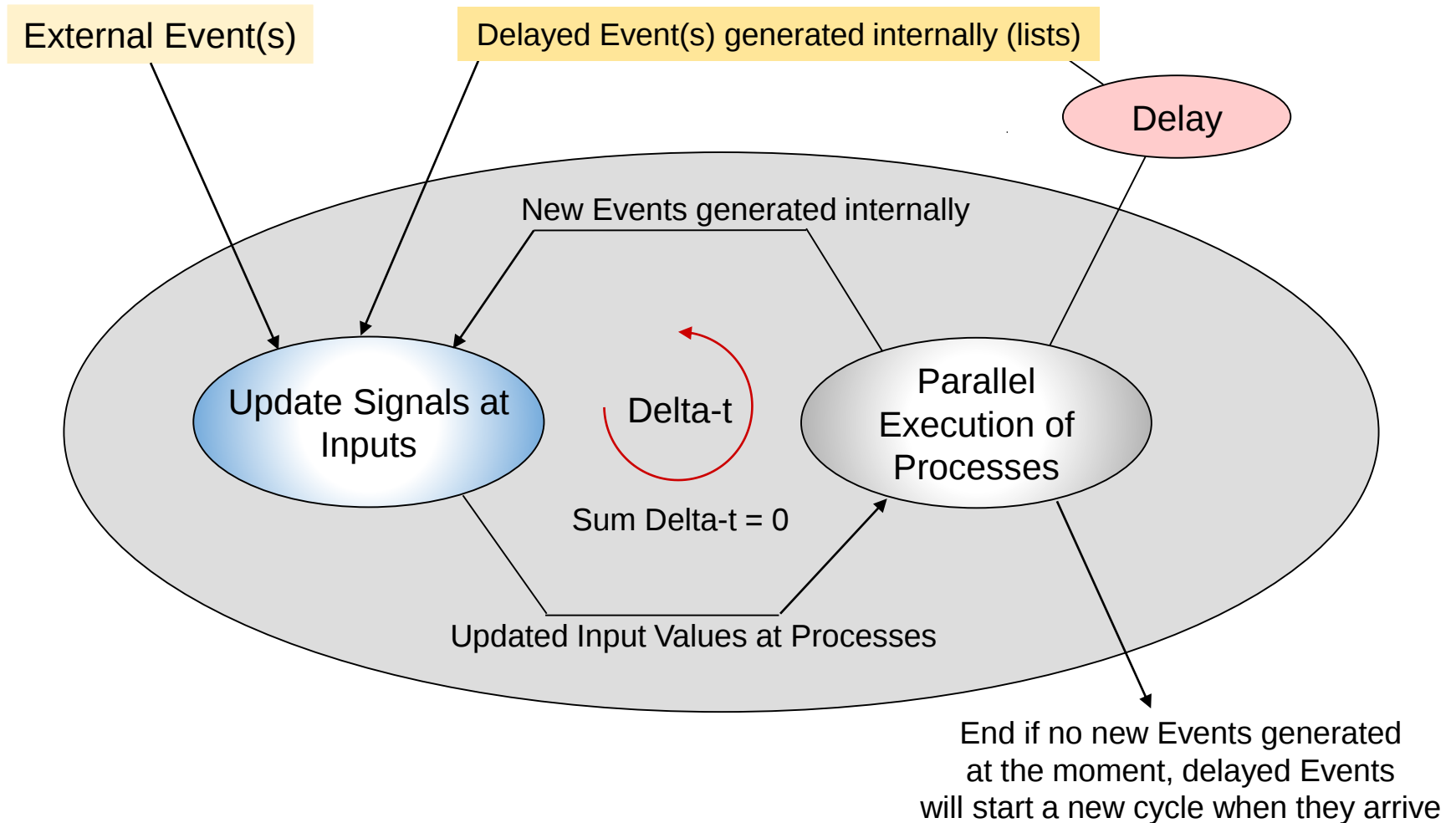


Simulation of a JK-Trigger in an Event Driven Simulator

Startig: JK-FF with Q= 0 in Hold State J,K = 0, CLK= 0

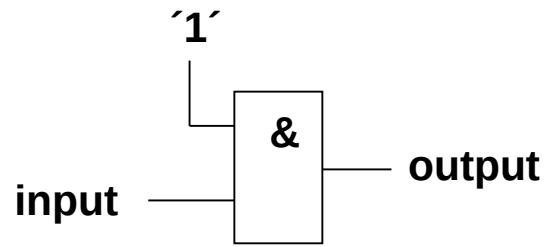


Execution of a simulation cycle in Event Driven Simulation

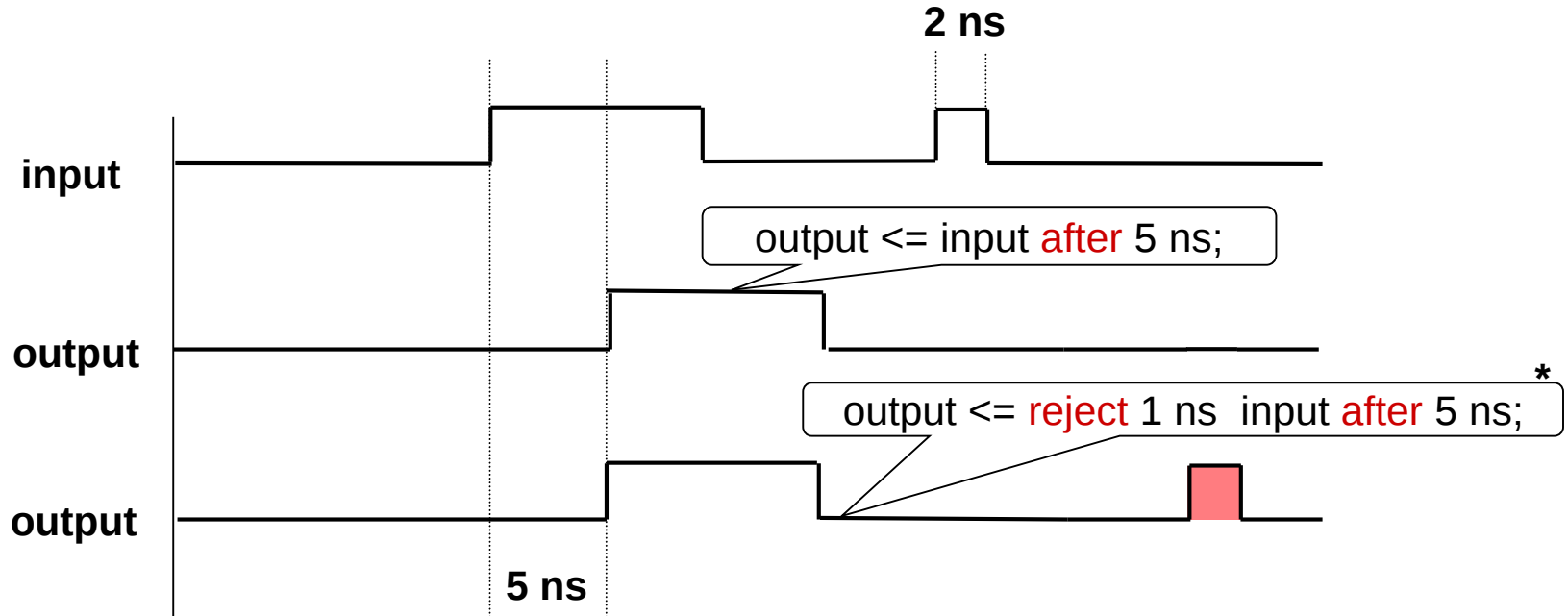


Delay Models (Inertial)

Inertial Delay

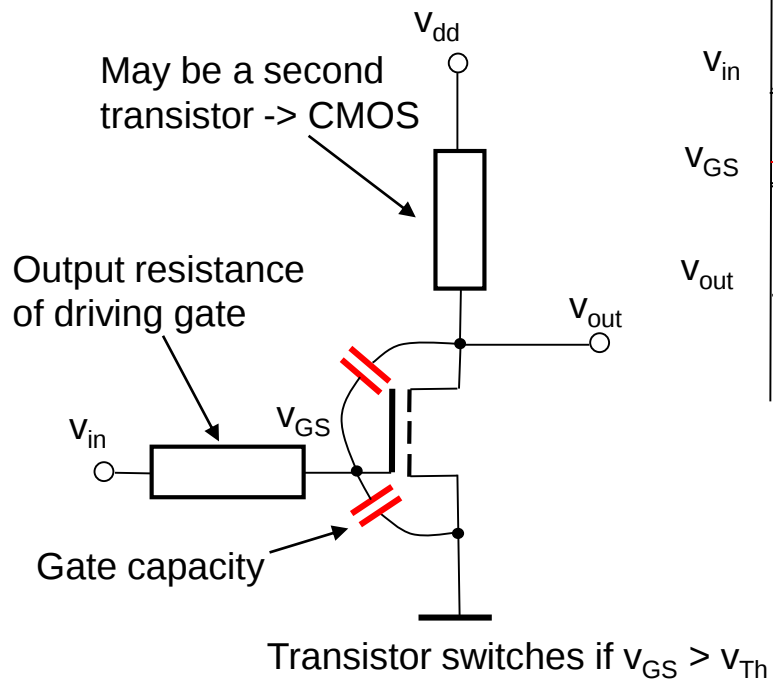


Typical behavior of **GATES**
Any pulse shorter than delay is
not propagated to the output

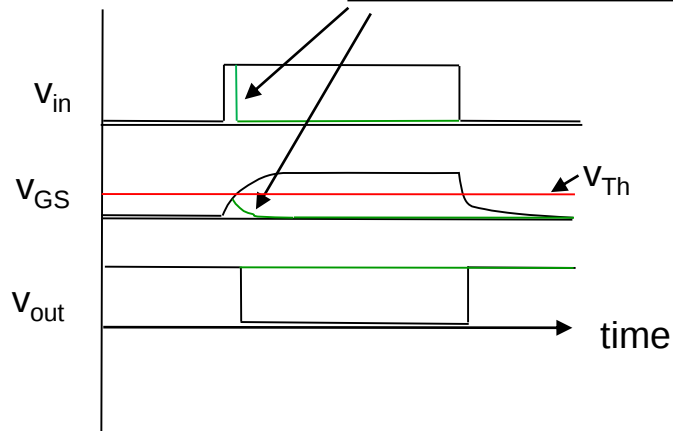


* reject is new in VHDL '93

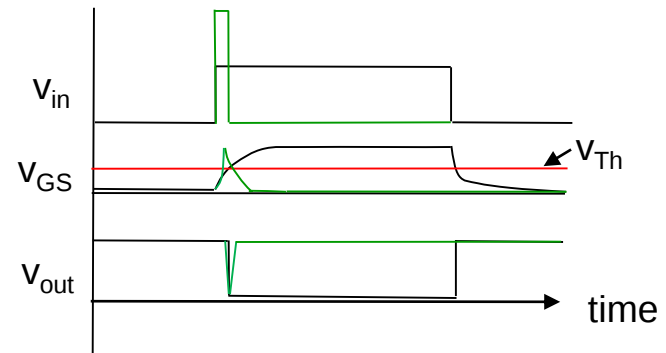
Behavior of Gates -> Inertial Delay



During the short pulse the threshold voltage v_{Th} is not reached and no output pulse is generated



There is a problem beyond digital view:
What does happen with short high energy pulses?

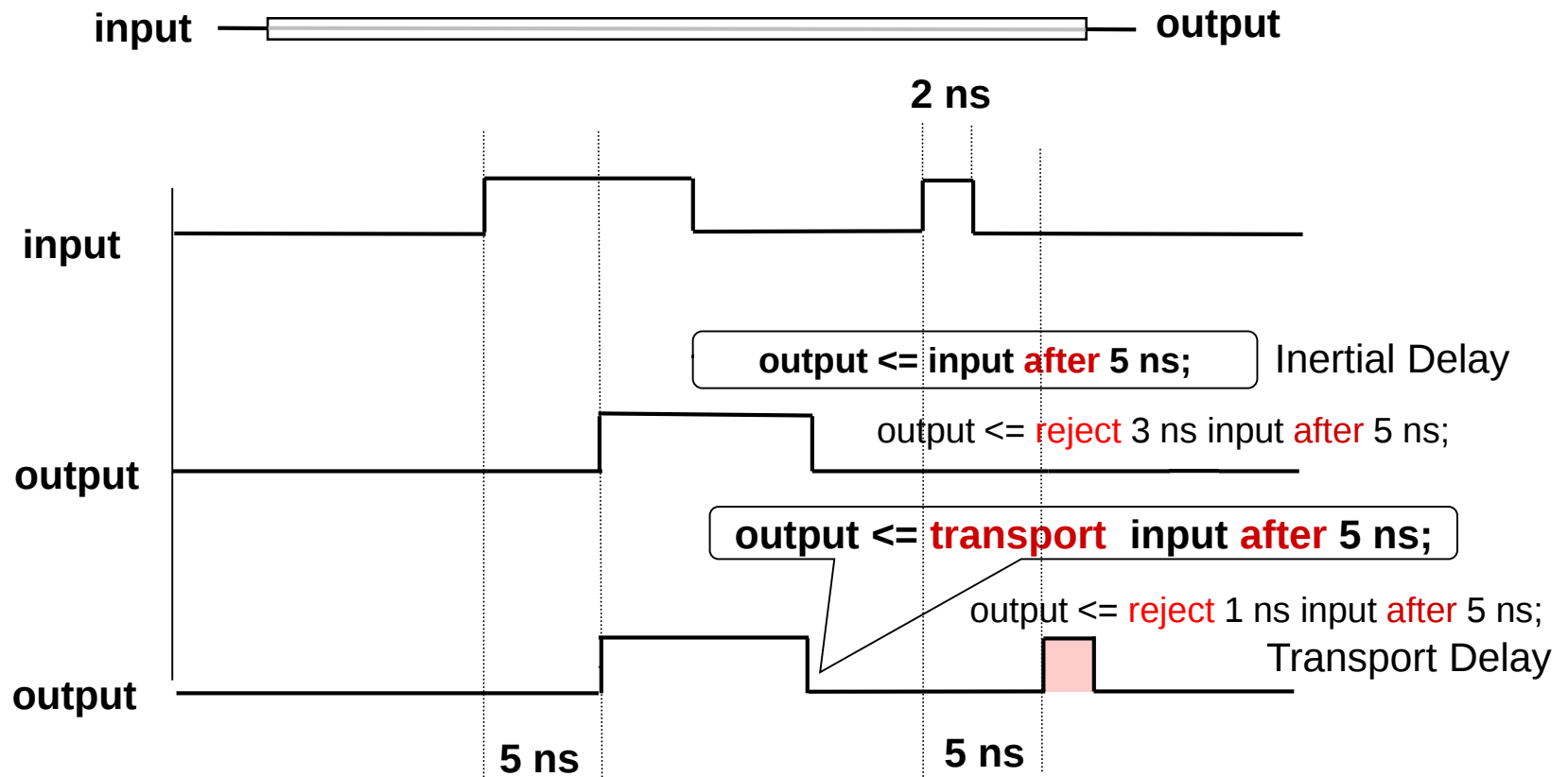


Delay Models (Transport)

Typical behavior of **WIRES**
Pulse shorter than delay is
propagated to the output

e. g. coax cable 5ns/m

Transport Delay



Design Entity

Entity Declaration

An **entity** can be a complex design like a CPU or even a simple gate

```
entity half_adder is  
  generic (td: time := 2 ns);  
  port (a, b: in bit;  
        sum, carry: out bit);  
end entity half_adder;
```

generics (parameters)

ports (signals)

type

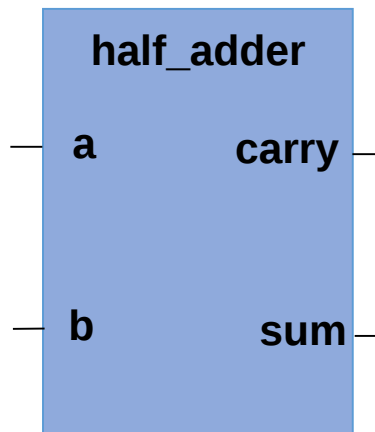
mode

in
out
inout
buffer

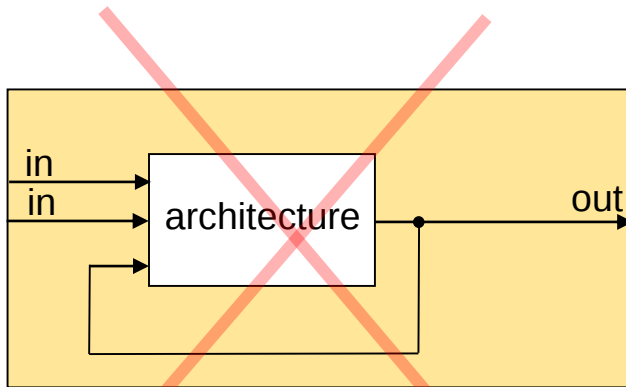
Signal not readable inside entity!
Use buffer or new internal signal.

>= VHDL 93

symbol

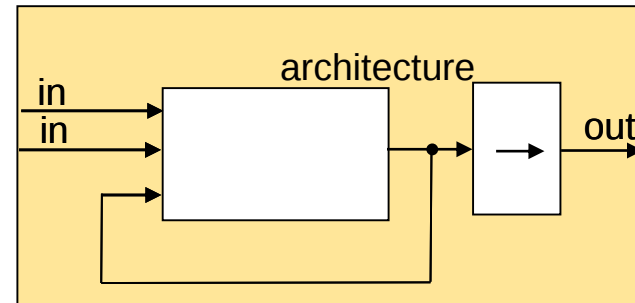


Port Modes

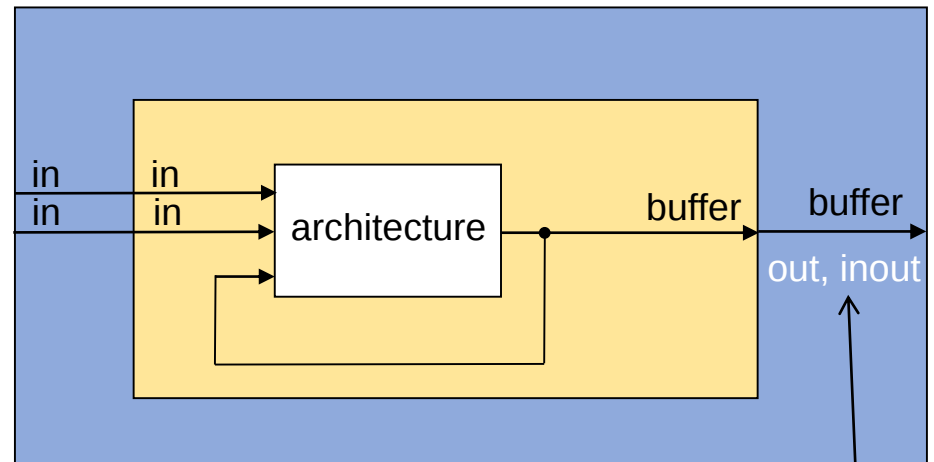


Port-Signal declared as „out“ cannot be read as an input

No longer restricted in VHDL 2008



Defining a new (internal) signal for internal feedback

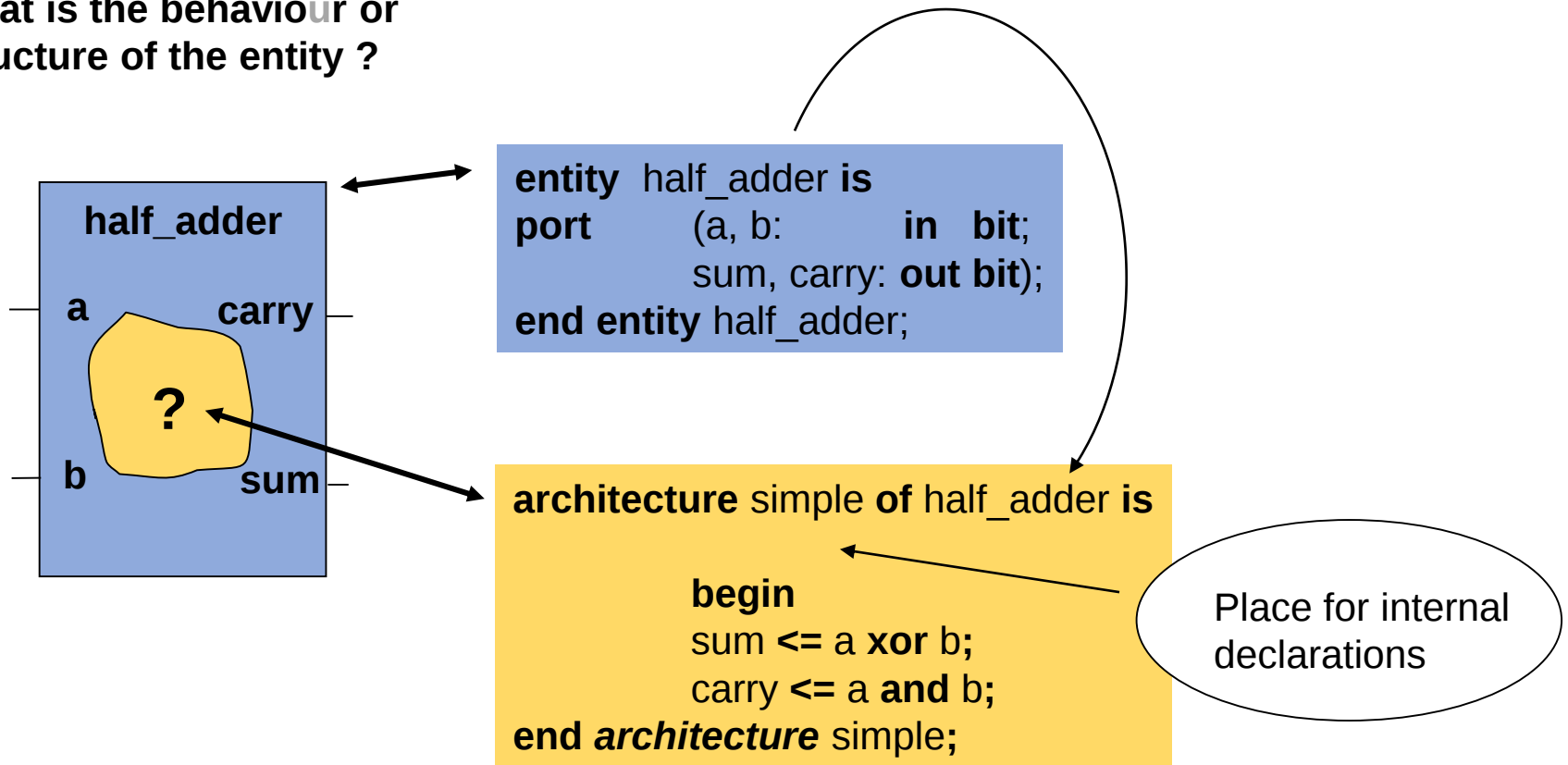


Mode „buffer“ must be used through hierarchy.

Beginning with VHDL-2002 connection to out, inout possible.

Architecture

What is the behaviour or structure of the entity ?



An entity can have more than one architecture. When the entity is used the architectures may be selected by [configurations](#). The architecture is the internal implementation of an entity. Synthesis tools will implement exactly one architecture.

Some Syntactical Rules: Identifiers, Comments, Instructions

VHDL is not case-sensitive, but remember that UNIX is !

and Verilog is!

HALF_ADDER and **half_adder** are the same in VHDL, but when in an UNIX-environment files are created using these identifiers UNIX scripts will distinguish between files **half_adder** and **HALF_ADDER** !

Scripts may become confused ..

x, **x1**, **x_1** are legal identifiers

x_, **1x**, **x-1** are illegal identifiers

VHDL key words must not be object identifiers

Example: **in** and **out** are illegal identifiers because they are port modes defined in VHDL, but **in0** or **out_1** are legal.

-- (two horizontal bars, hyphens) start a comment in VHDL

Some programs use special commands for flow control

Example: **-- pragma translate off/on** in SYNOPSYS

VHDL instructions may extend a line

Instructions are closed by the **;** semicolon character

/* */ in VHDL 2008 possible like in C and other languages

Some Syntactical Rules: Extended Identifier (VHDL 93)

VHDL 93 provides „Extendend Identifiers“

Enclosed in backslash

Case sensitive!

May contain special characters

May be reserved VHDL keywords

May start with a number

Allows keeping identifiers after translation from other languages or netlists

Examples:

\signal number 1\	with space
\1signal\	starting with a number
\in\	keyword in VHDL
\flip&flop\, \FLIP&FLOP\	special character included

Some Syntactical Rules: Literals

Values directly specified in the description

INTEGERS:	DECIMAL	2561
	HEXA	16#A01#
	OCTAL	8#5001#
	BINARY	2#1010_0000_0001# (underscore for readability)

With Exponent: $= 2\#11\#E8 = 3 \cdot 2^8 = 768$

FLOATING POINT:	3.141	(period indicates: floating point)
	1.23E-3	(exponential)

With Exponent: $= 16\#A.01\#E2 = \left(10 + \frac{1}{256}\right) \cdot 16^2 = 2561$

CHARACTERS: 'A' (single quotation marks)

STRINGS: "Abcd efg" (quotation marks)

String in string: Double quotation marks "ab"abc""

BIT STRINGS **B"10100010"** or simply **"10100010"**
X"A2" (hexa) or **O"242"** (octal)

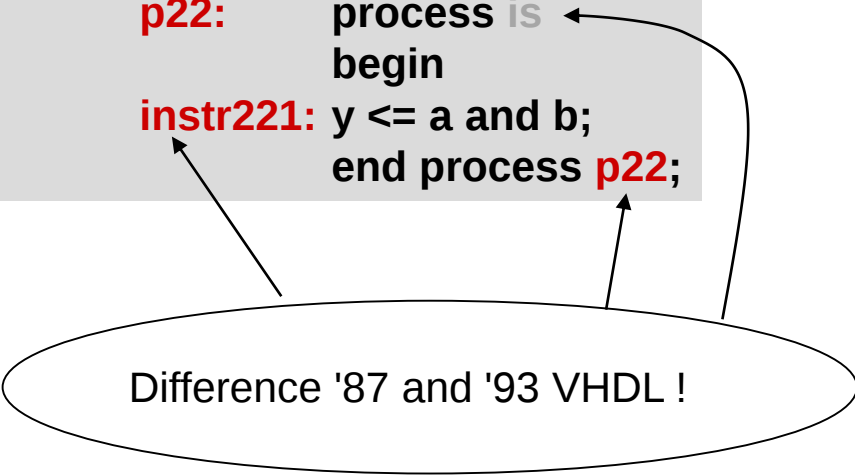
PHYSICAL: **127 ns** (value, space, unit)
 230 V

Some Syntactical Rules: Labels

Many VHDL units like process, loop, procedure, but also simple instructions in VHDL93 and newer can have a label for identification:

Example:

```
p22:      process is  
          begin  
instr221: y <= a and b;  
          end process p22;
```



Difference '87 and '93 VHDL !

And sometimes it is advisable to use labels. Helps to identify code during debugging in simulation.

Objects

Classes of Objects in VHDL (digital)

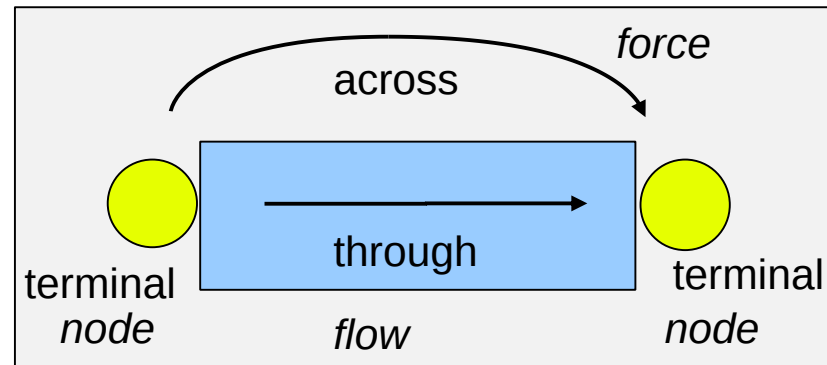
Signals Representation of wires in hardware.
New value is always assigned "in the future",
when all processes are finished for the
actual simulation cycle

Variables (including files VHDL'87)
Useful for computation
Get their new value immediately after
assignment.

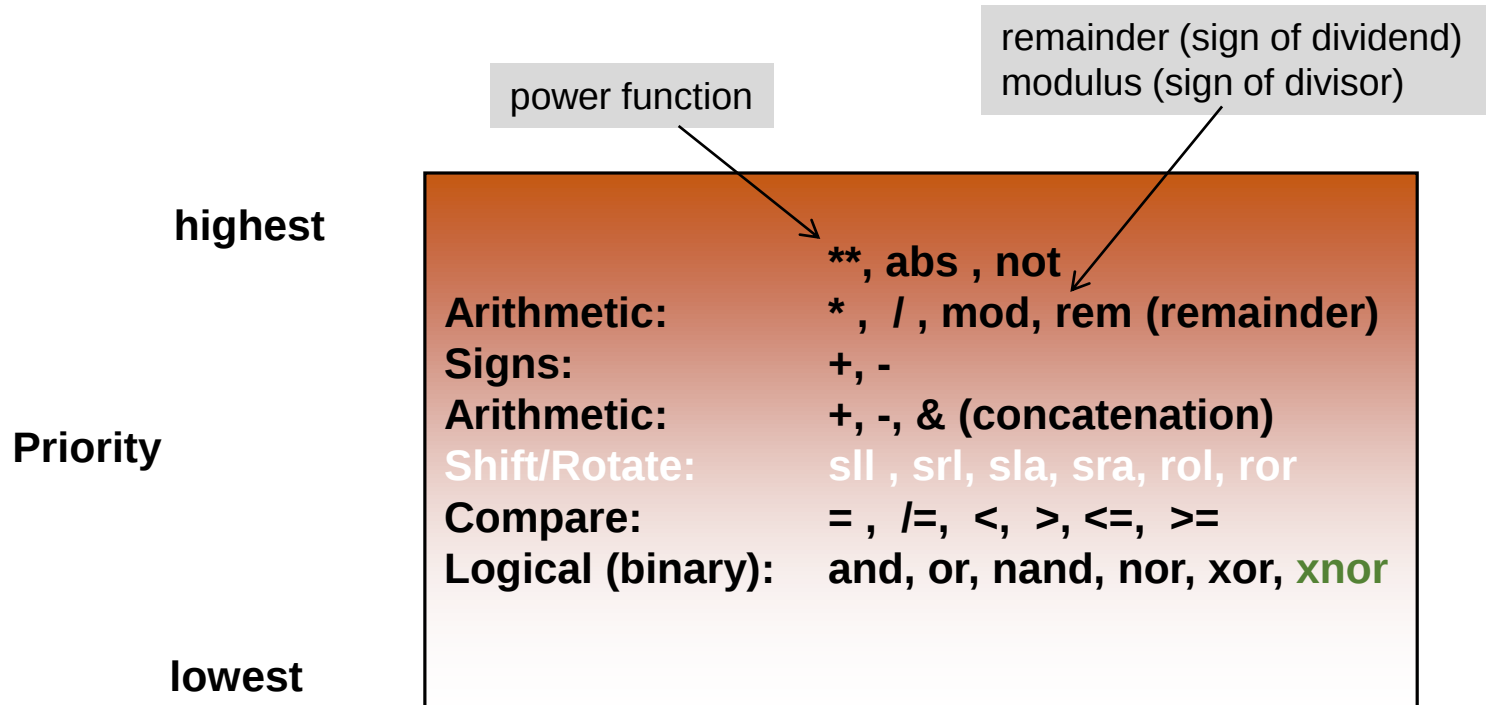
Constants Value is assigned at declaration only
Files (VHDL'93 and newer)

Quantities (VHDL-AMS)
floating TYPE
voltage or current
Terminals (VHDL-AMS)

electrical – voltage, current
thermal – temperature, heat(power) flow
translational – displacement, force
fluidic – pressure, flow rate



Operators



sll/srl: Shift logical (include MSB)
sla/sra: Shift arithmetical (excl. MSB)
rol/ror: Rotate left/right



Do not use the shift operators.
Use functions like **shift_right**, **shift_left**
from package IEEE.Numeric_STD or use slices.

concatenation operator is & (for 1-dimensional arrays incl. strings)

"abc" & 'd' is "abcd" and ('0','1') & B"001111" is "0100_1111" or ('0','1','0','0','1','1','1','1')

Join strings Join bit vectors

Operators

New operators in VHDL 2008

Matching Compare
Don't care '-' at one input
possible

```
cs >= '1' when addr(19 downto 12)=X"0F" else '0';  
cs <= addr ?= X"0F---";
```

Matching Compare: `?=` , `?/=`, `?<`, `?>`, `?<=`, `?>=`
Logical (unary): `and`, `or`, `nand`, `nor`, `xor`, `xnor`
Condition Conversion: `??`
`maximum`, `minimum`, `to_string`, `rising_edge`, `falling_edge`

Confusing, because often seen in older descriptions ->
available from additional libraries -> now native VHDL

Type: bit

`Y <= and X;`

If `??(a and b)` then

`Y = x ?>= "1001";`

If X is a bit-vector, the result is the and-function of its elements. The result type is bit, std_ulogic.

If a, b are of bit or std_logic type,
?? converts result to boolean -> **See next slide!**

If x is greater than "1001" the result Y will be '1', not "true"
Argument types: bit, std_ulogic or vector of them only

`Podd <= xor Y;` -- Podd = 1, when bit or std_logic vector Y has odd parity (odd number of '1' values)

Bit and Boolean

Predefined types in VHDL

type bit is ('0','1');

signal s1: bit;

WORKING: if s1= '1' then ..

NOT WORKING: if s1 then ..

type boolean is (false,true);

signal s2: boolean;

WORKING: if s2 then ..

WORKING: if s2 = true then

NOT WORKING: if s2 = '1' then ..



Condition conversion in VHDL 2008

?? sig1 converts '0' and '1' to false and true

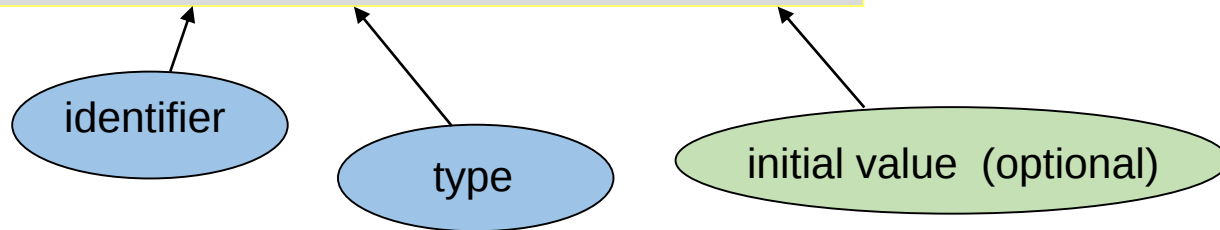
Automatically without operator in newer versions?

Operators (Comparison VHDL <-> Verilog)

	VHDL	Verilog
logical	not, and, or, xor ..	~, &, , ^
shift	sll, sla ...	a<<1, a<<<1
concatenation	a & b	{a,b}
unary reduction	y <= and x;	y <= &x;

Object Declaration (Signals)

```
signal z_internal: bit;  
signal state: main_state_type := idle;  
signal counter: integer range 0 to 127 := 0;
```



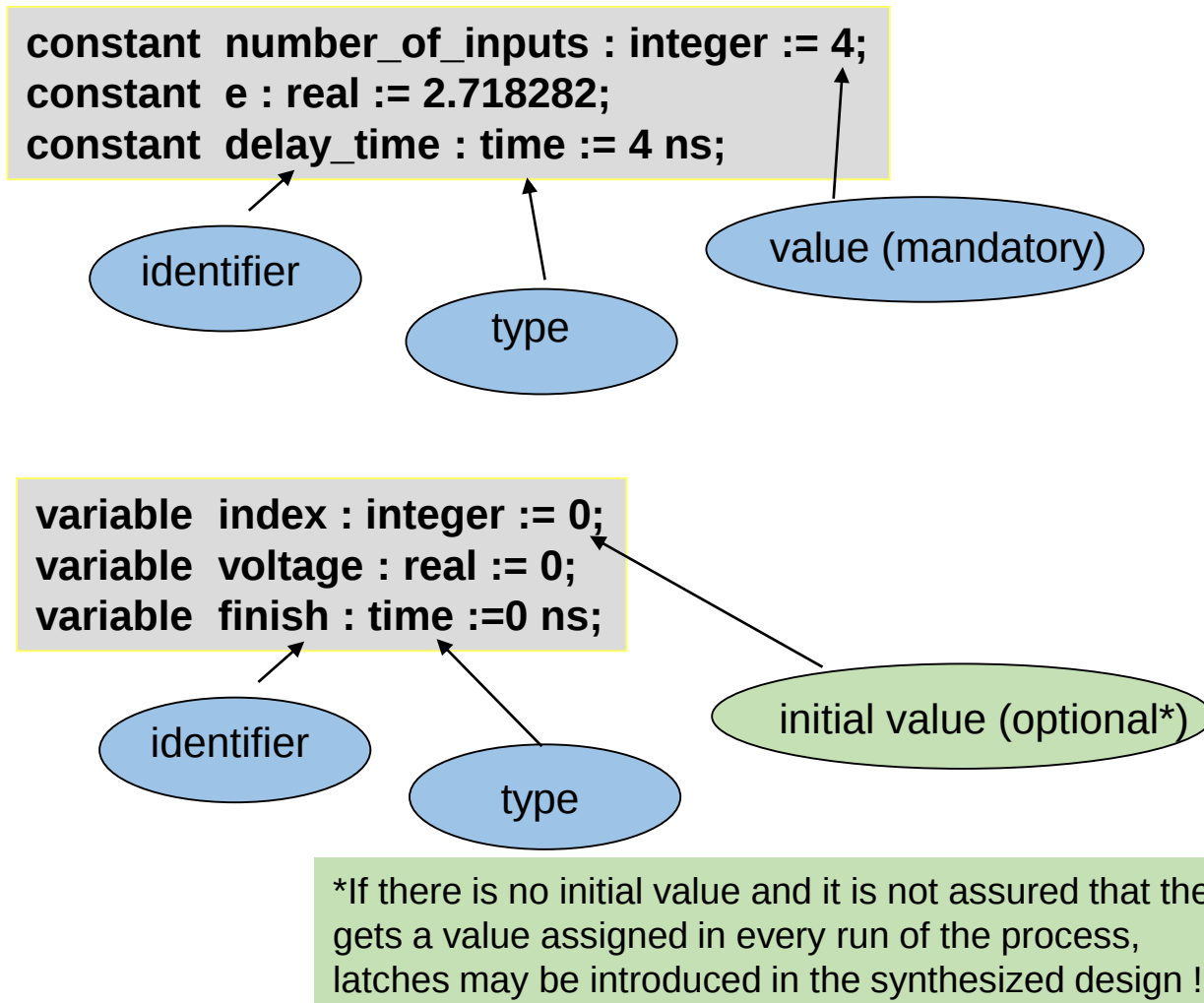
Initial values for simulation,
Declaring an initial value will NOT
include reset hardware in synthesis !!

If we **know** that the hardware (FPGA) sets all flip-flops to '0' at poweron it may be a good idea to presume this fact in our design and to help the simulator to start with the proper value, using an initial value!

The popular type "std_logic" for signals will start with value 'X' in simulation!
You can derive this behaviour from the declaration of the type where 'X' is the first value.

Remember: Signals may also be declared in entity declarations (inputs and outputs)

Object Declaration (Constants and Variables)



Concurrent and Sequential View

Concurrent View

Signal Assignments

Simple Signal Assignment

Conditional Signal Assignment

Selected Signal Assignment

PROCESS

PROCEDURE

ASSERT

Sequential View

Signal Assignment

Conditional Signal Assignment

Selected Signal Assignment



VHDL 2008

Variable Assignment

Flow Control

IF Statement

CASE Statement

Loops

Simple Loop

FOR Loop

WHILE Loop

NEXT

EXIT

Procedure Calls

WAIT

ASSERT

RETURN

NULL

Concurrent Signal Assignment - CSA

Simple Signal Assignment

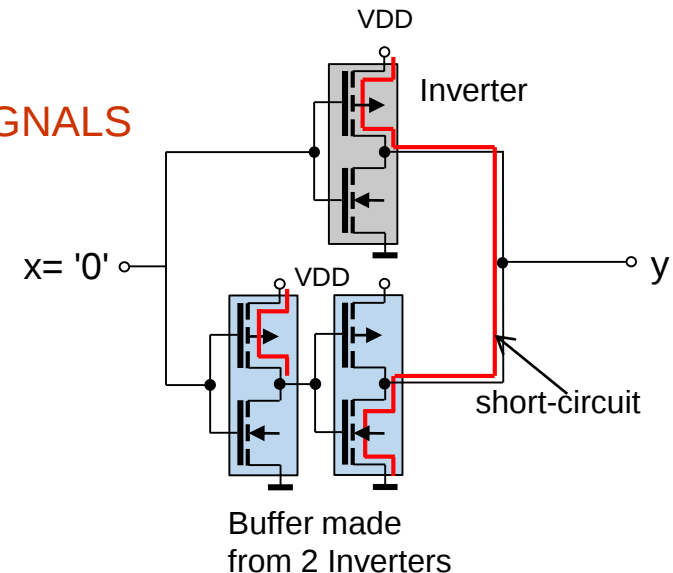
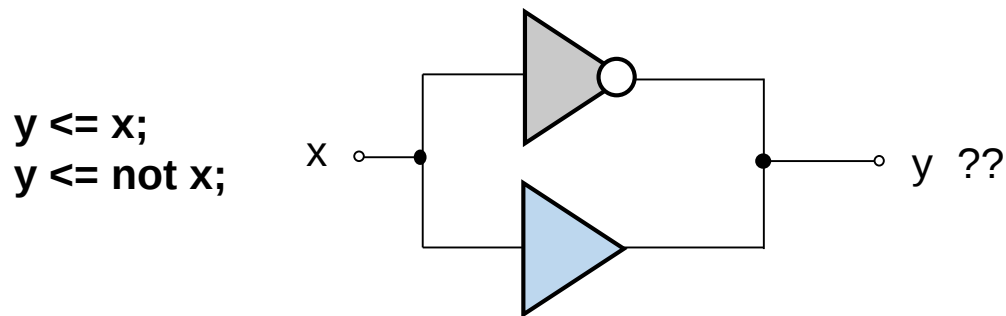
```
u <= a and b;  
v <= (a or c) after 5 ns;  
w <= '1', '0' after 10 ns, '1' after 20 ns;
```

$\leq$
"is defined to be"

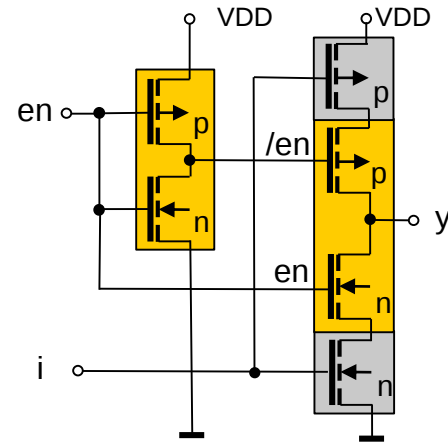
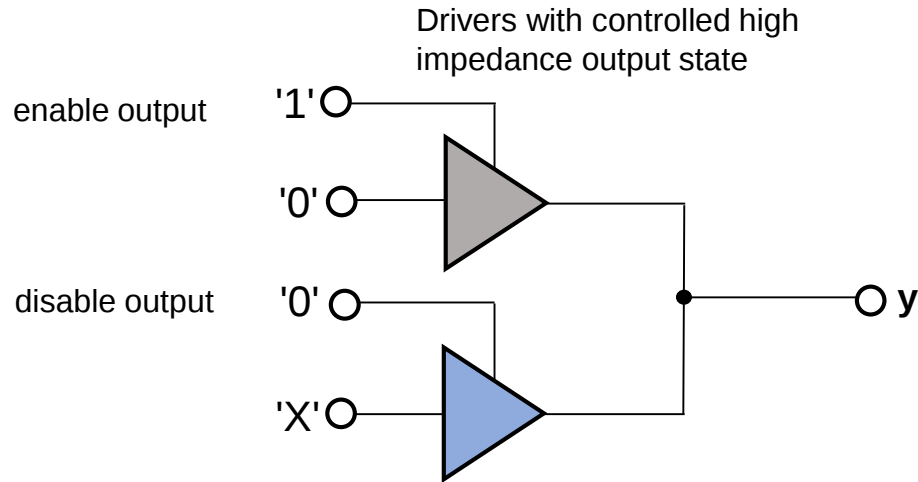
Not an equation. We cannot swap/transpose both sides. Hardware is (mostly) unidirectional.
Exception: Transmission Gates (No operator, but instantiation possible)

Usually a signal must not have more than one driver,
Only one CSA can assign a value to a signal.

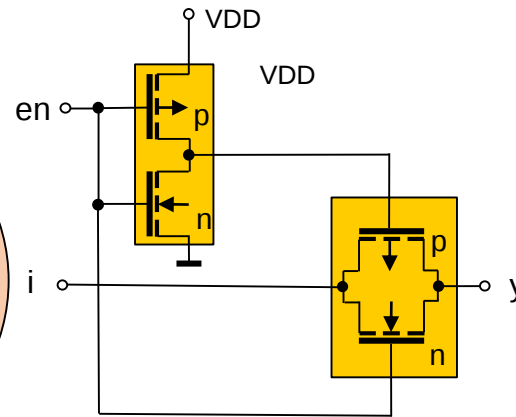
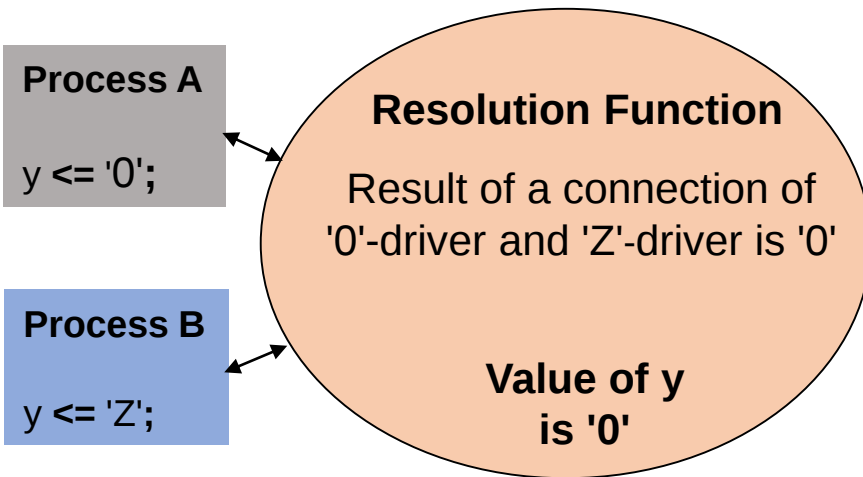
Signals with more than one driver --> **RESOLVED SIGNALS**



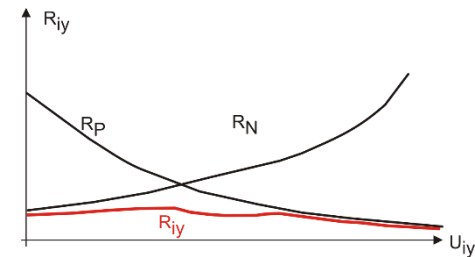
Resolved Signals



Inverter with tristate control



Transmission Gate
nonrestoring (noise from "in" is passed to "out")



Concurrent Signal Assignment - CSA

Conditional Signal Assignment

```
y <= a after 5 ns when input = 0 else  
      b when input = 1 else  
      c when input = 2 else  
      d when input = 3  
      else 'X' ;
```

choice selector: integer

Selected Signal Assignment

```
with input select y <= a after 5 ns when "00" ,  
                     b when "01",  
                     c when "10",  
                     d when "11",  
                     'X' when others ;
```

choice selector: bit_vector

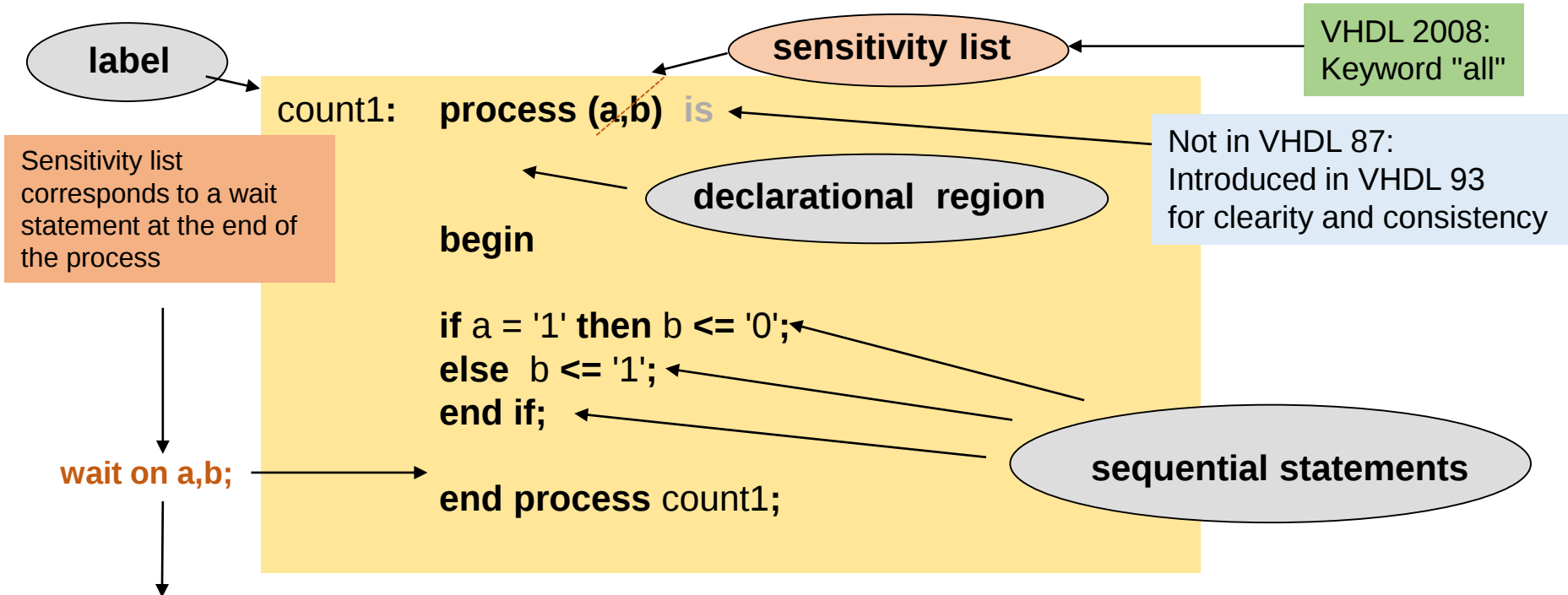
Possible keyword **unaffected** in VHDL '93

sequential logic (latches)!

Process (combinatorial, sensitivity list / wait)

Process construct allows sequential statements inside.

Process is one CSA in the description (or: every CSA is a process)



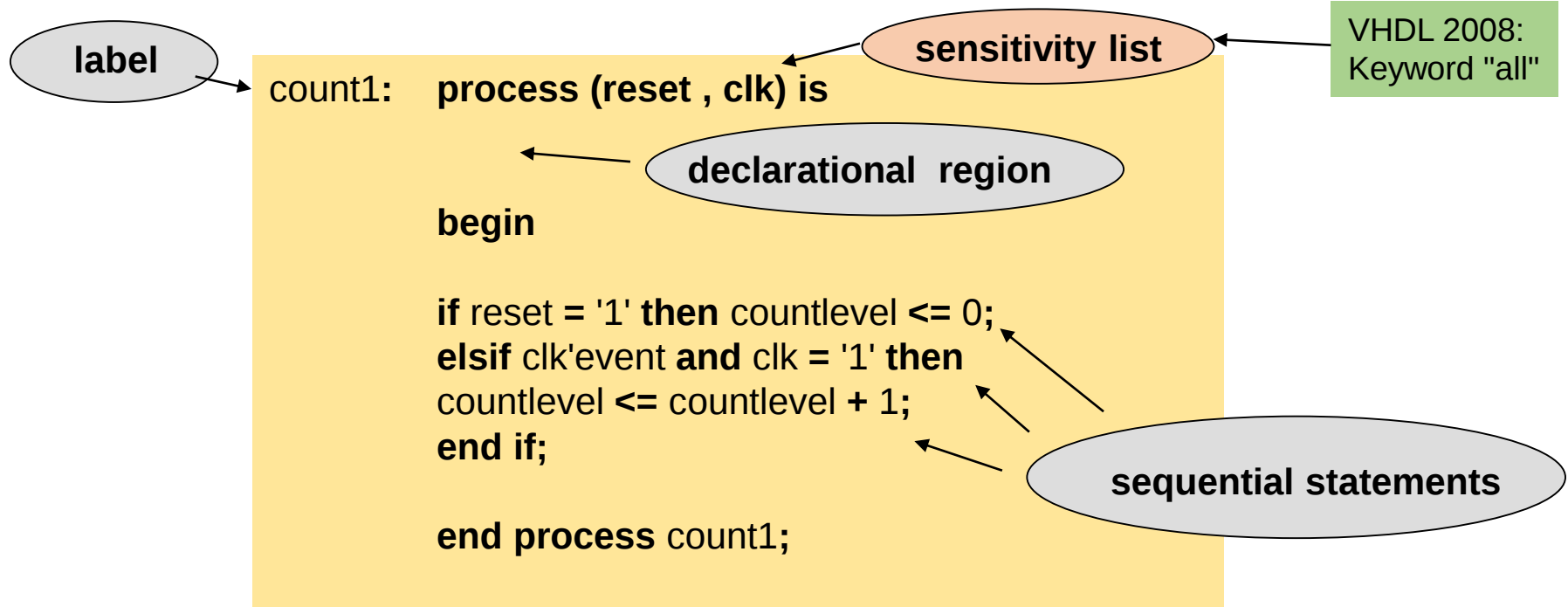
Process is executed whenever an event at a signal mentioned in the sensitivity list occurs. (and waits just before the end statement)

Process must have a sensitivity list **or** wait statements for execution control, but not both together !!

Process (sequential, sensitivity list)

Process construct allows sequential statements inside.

Process is one CSA in the description (or: every CSA is a process)



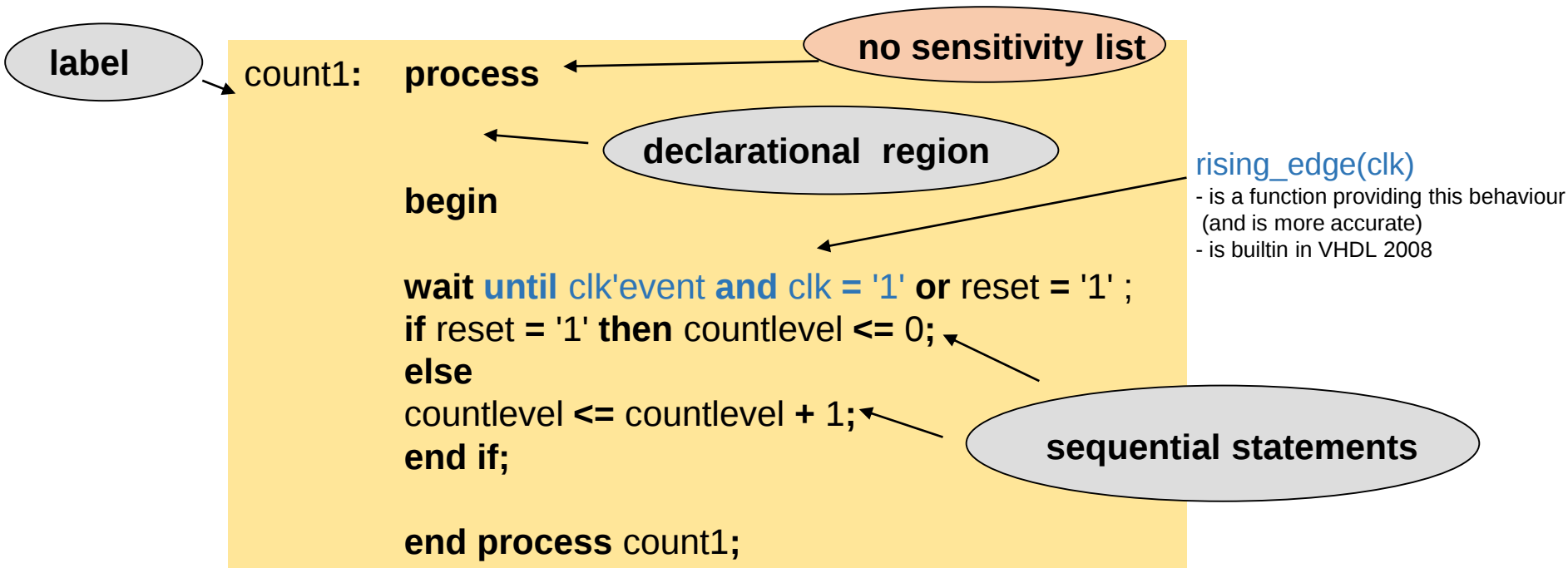
Process is executed whenever an event at a signal mentioned in the sensitivity list occurs (and waits just before the end statement)

Process must have a sensitivity list **or** wait statements for execution control, but not both together !!

Process (sequential, wait statement)

Process construct allows sequential statements inside.

Process is one CSA in the description (or every CSA is a process)



Process with no sensitivity list is executed forever.

To achieve an advance in time there must be a wait statement if there is no sensitivity list.

Process must have sensitivity list **or** wait statements for execution control, but not both together !!

Sensitivity list is equivalent to a wait statement at the end of the process (if they concern the same signals, events).

Signal Assignment

VHDL 2008:
Selected and Conditional allowed

Signal Assignment (simple only)

```
u  <=  a and b;  
v  <=  (a or c) after 5 ns;  
w  <=  '1' , '0' after 10 ns, '1' after 20 ns;
```

Inside a process multiple assignments to the same signal are possible.
Last assignment executed will give the result (after end of all processes).

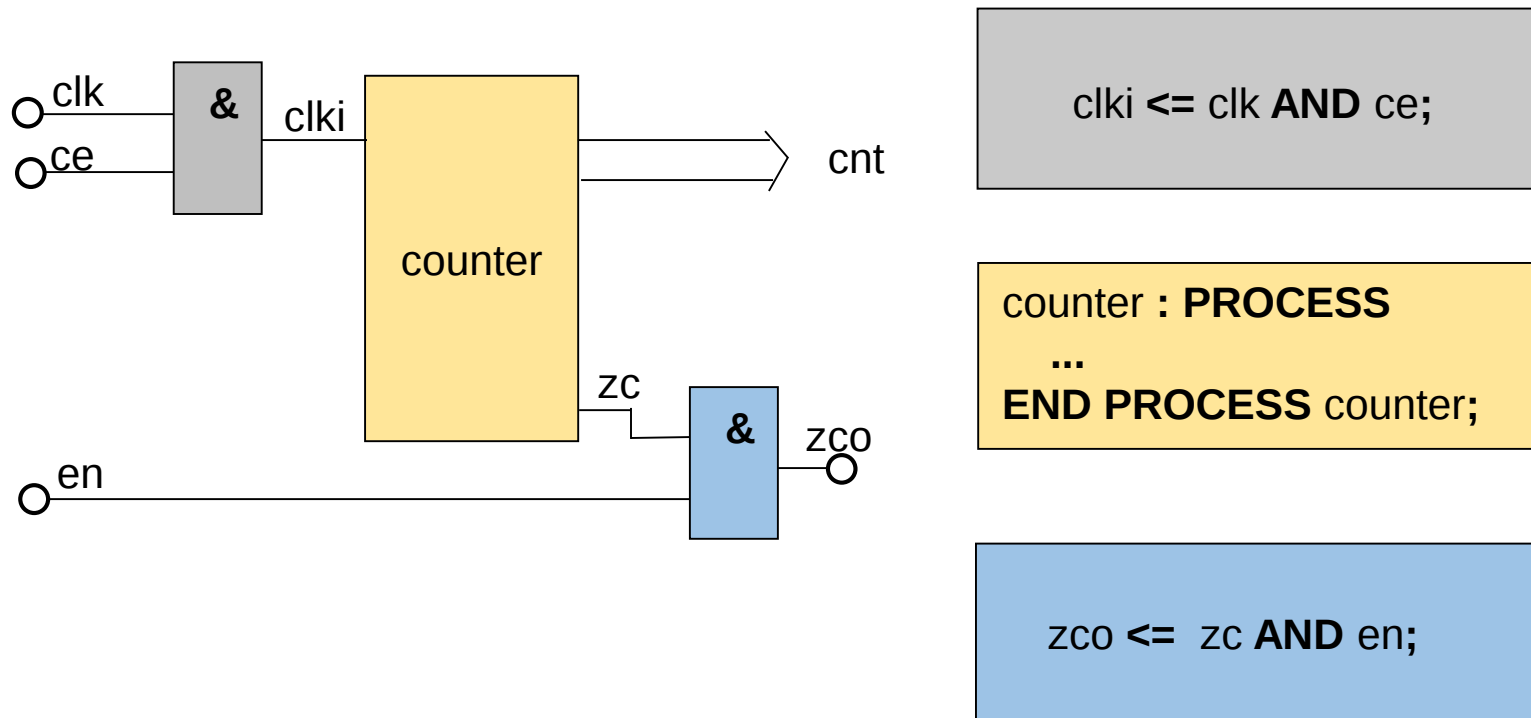
Variable Assignment

```
v := a and b;
```

Only inside a process*.

Assignment is processed immediately. Value stored from process call to process call. -> If not initialized before first use in process -> Latch or Flip-Flop

Process as Concurrent Statement/ Concurrent Statement as Process



IF (sequential)

type: boolean
extended in VHDL 2008 (bit, std_logic)

VHDL'93

label: IF condition THEN statements;
END IF **label;**

if a='1' and b='1' then
if ?? a and b then
if a and b then

IF condition THEN statements;
ELSE statements;
END IF;

IF condition THEN statements;
ELSIF condition THEN statements;
ELSIF condition THEN statements;
ELSE statements;
END IF;

Conditions may overlap
Priority by the order in code

ELSIF

END IF

Nesting of IF-statements allowed,
but not recommended excessively,
especially in synthesis

Target hardware: priority encoder

Case (sequential)

Control flow by the value of an expression

```
label: CASE      object      IS
    WHEN value1      => statements;
    WHEN value2      => statements;
    WHEN value3 | value4  => statements;
    WHEN value5 TO value7  => statements;
    WHEN OTHERS      => statements;
END CASE label;
```

VHDL 2008: expressions instead of single objects allowed

VHDL 2008: Matching case case? expression is when "--00" =>

Choices mustn't overlap. Must be "locally static".
All possible values for expression must be represented as choices!

Different behavior in Verilog (default is optional, overlap is allowed)

Value1 or value 2
Using a RANGE of choices
Referencing all values, not used yet

value1 | value2
value1 TO / DOWNTO value2
OTHERS

Range must be defined.
Is not for bit-vectors!
Is for integer, enumeration

If you need a branch with no action use the **NULL**-statement

Target hardware: multiplexer

Case (example)

Signal count runs cyclic from 0 to 137 with an output pulse of '1' at zero_out while the signals value is 0.

```
SIGNAL count :    INTEGER RANGE 0 TO 137;  
SIGNAL zero_out : BIT;
```

```
WAIT UNTIL clk'EVENT and clk='1';
```

```
counter138:  
CASE    count IS  
    WHEN    0          => count <= count + 1;  
                zero_out <= '0';  
    WHEN 1 TO 136      => count <= count + 1;  
    WHEN    137        => count <= 0;  
                zero_out <= '1';  
    WHEN OTHERS      => count <= 0;  
                zero_out <= '1';  
                report ("unexpected value for count");  
END CASE counter138;
```

Loop (sequential)

Simple Loop

Label: **LOOP**
 statements
 EXIT label WHEN condition;
 END LOOP label;

In all loops

EXIT statement must be
executed or WAIT included

For Loop

Label: **FOR loop_variable IN range LOOP**
 statements
 END LOOP label;

Integer, implicit declaration

Stepsize 1

0 TO 10 15 DOWNTO 1
V'LOW TO 15 V'HIGH DOWNTO 0
V'RANGE V'REVERSE_RANGE
A'RANGE(2) A'REVERSE_RANGE(1)

While Loop

Label: **WHILE condition LOOP**
 statements
 END LOOP loop_label;

Condition must become
FALSE during loop execution
or EXIT statement must be
executed or WAIT included

!Synthesis!

Loops (example)

The signals input and output are vectors with elements of the (self defined) type bit4v.

TYPE bit4v **IS** ('X', '0', '1', 'Z');

example: PROCESS (input)

VARIABLE result : bit4v := '1';

BEGIN

result := '1';

FOR i **IN** input'RANGE **LOOP**

CASE input(i) **IS**

WHEN '0' **=>** result := '0' **EXIT**;

WHEN 'X' **=>** result := 'X' **NEXT**;

WHEN 'Z' | '1' **=>** **NULL**;

END CASE;

END LOOP;

output <= result;

END PROCESS example;

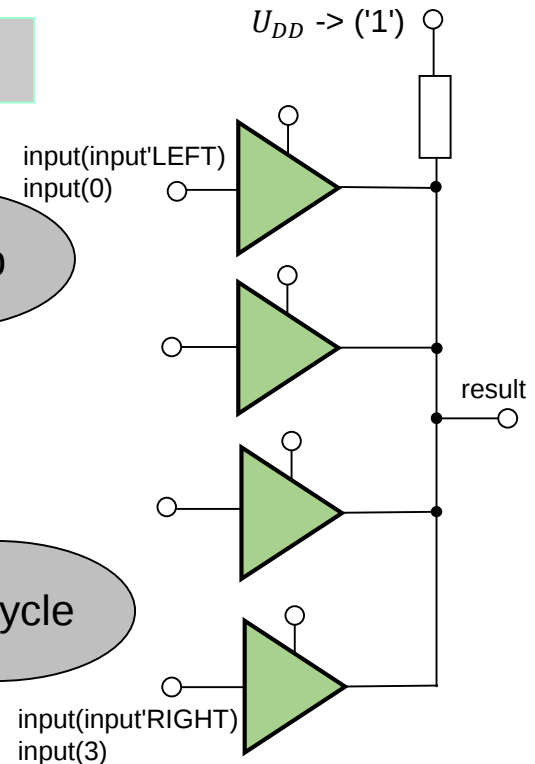
Loop works in area,
not in time !

'0' is dominant value

exit loop

next cycle

change nothing



-> input(input'RANGE is: 0 to 3)

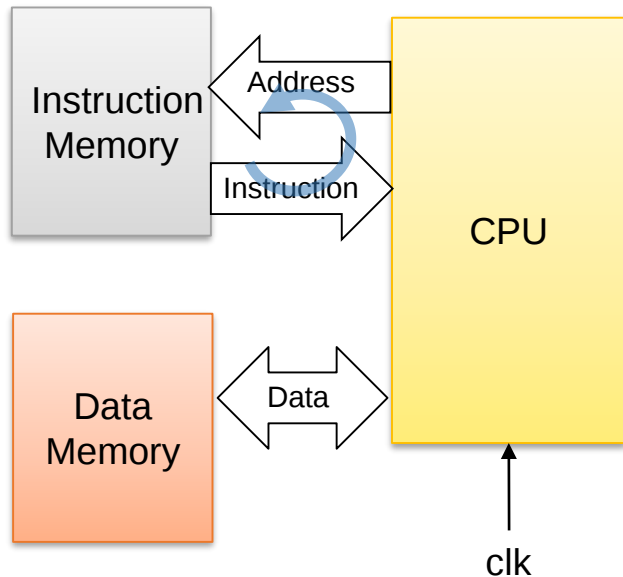
Multiplexing the inputs with a wired_or FUNCTION using open-drain outputs.

Loop in Programming and Hardware Description Languages

Programming Language

```
for i in 0 to 3 loop
if dig_array(i) = 9 then dig_array(i) <= 0;
  next;
else dig_array(i) <= dig_array(i) + 1;
  exit;
end if;
end loop;
```

time

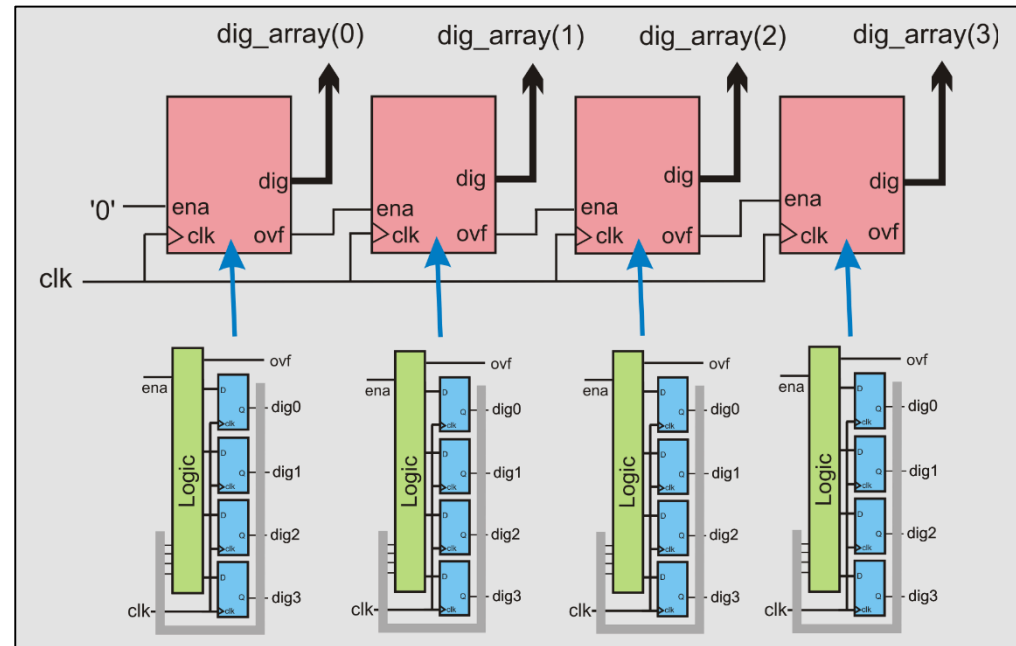


multiple clock cycles, dependent on state
(overflow or not)

Hardware Description Language

```
if rising_edge(clk) then
for i in 0 to 3 loop
if dig_array(i) = 9 then dig_array(i) <= 0;
  next;
else dig_array(i) <= dig_array(i) + 1;
  exit;
end if;
end loop;
end if;
```

area



One clock cycle

Wait Statement (Sequential)

WAIT suspends a process

WAIT; suspends until end of simulation

WAIT ON list of signals; an event at one of the listed signals reactivates the process

WAIT UNTIL condition; process is suspended until condition = TRUE

WAIT FOR time; process is suspended for the given time

WAIT UNTIL sig1 = '1' **FOR** 1 us; (condition TRUE or TIMEOUT)
WAIT ON a **UNTIL** b = '1'; (b tested whenever event at a)
WAIT UNTIL (clk'event AND clk = '1'); (wait for rising clock edge)

Procedure Call

- Subroutine with sequential statements
- Procedures can change the values of their input objects

label: identifier (association list);

Examples with different kinds of association lists:

Positional: add (x1 , x2 , y , c);

Named: add (term1 => x1, carry => c, sum => y, term2 => x2);

```
PROCEDURE add
  (term1, term2: IN bit_vector;
   sum:      OUT bit_vector;
   carry:    OUT bit) IS ....
```

Named association: locals => actuals

In synthesis every procedure call invokes a new instance of hardware !

Procedure

- Subroutine with sequential statements
- Can change the values of its input objects

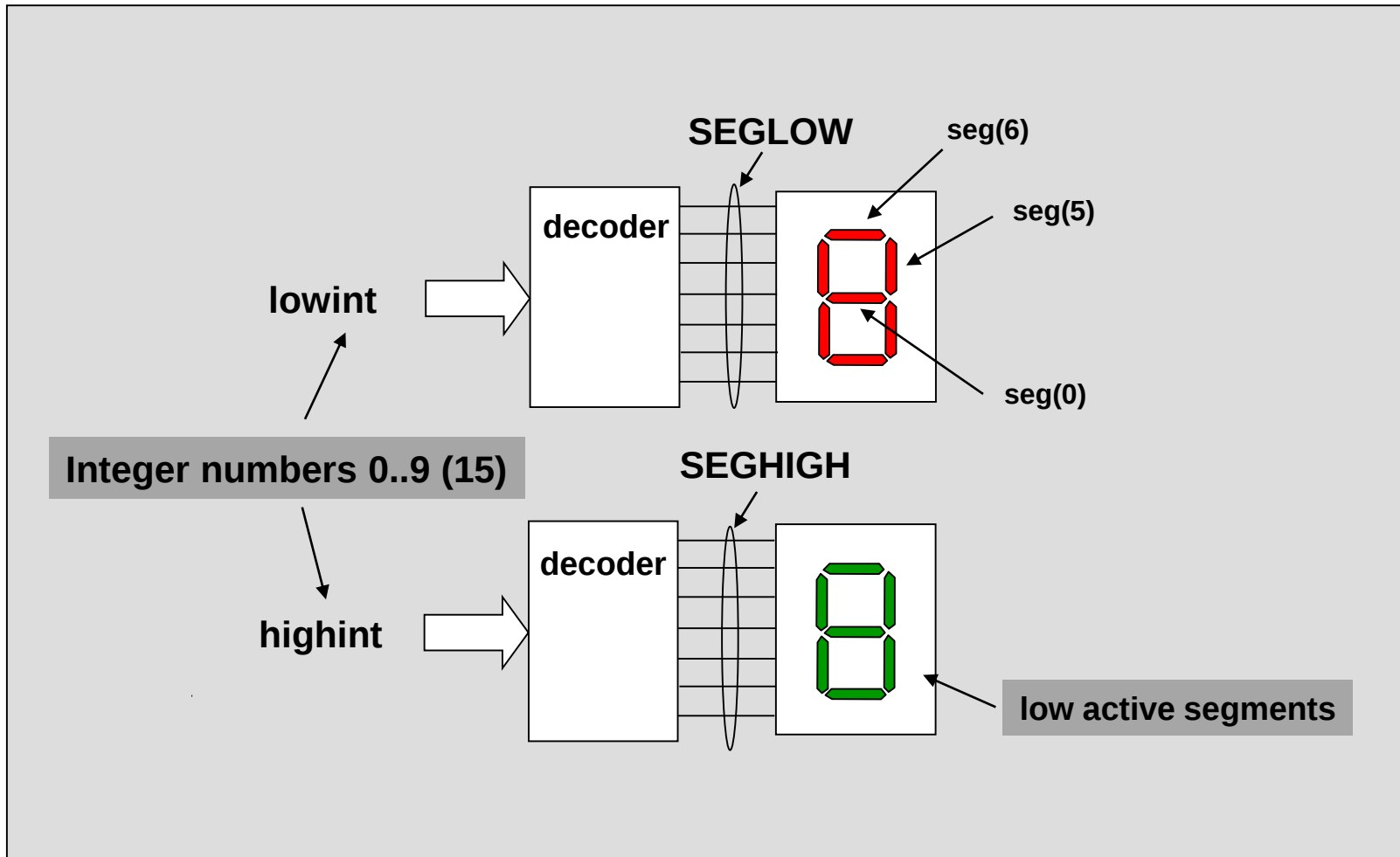
```
PROCEDURE add (term1, term2: IN bit_vector;  
               sum: OUT bit_vector;  
               carry: OUT bit) IS  
Variable c: bit := '0';  
BEGIN  
    FOR i IN term1'REVERSE_RANGE LOOP  
        sum(i) <= term1(i) XOR term2(i) XOR c;  
        c := (term1(i) AND term2(i)) OR ... ;  
    END LOOP;  
    carry <= c;  
END PROCEDURE add;
```

term1, term2, sum must be of same dimension in this example.

If *term1* is declared 7 downto 0 the loop is running from 0 to 7

Unconstrained ARRAY parameter possible.
Default values possible.

Procedure Example



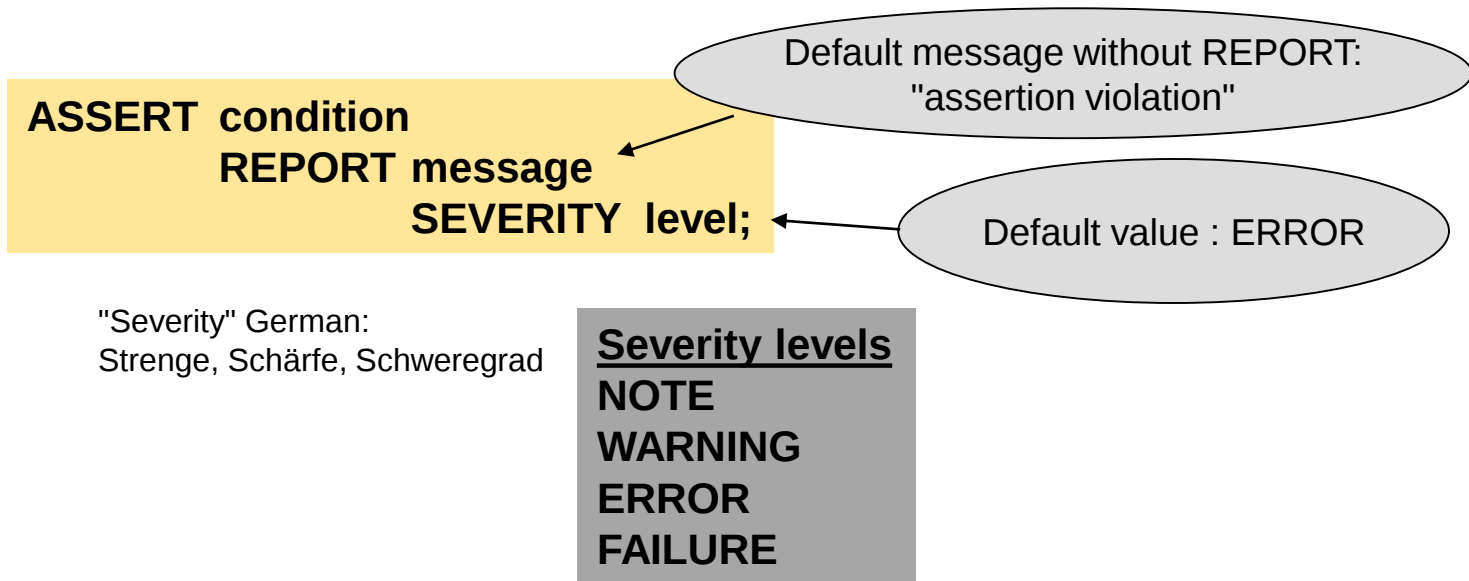
Procedure Example

```
procedure decode(signal dig: in integer range 0 to 15;  
                signal seg: out std_logic_vector(6 downto 0)) is  
begin  
  case dig is  
    when 0 => seg <= "0000001";  
    when 1 => seg <= "1001111";  
    ..  
    when 8 => seg <= "0000000";  
    when 9 => seg <= "0000100";  
    when others => seg <= "1111111";  
  end case;  
end procedure decode;
```

```
decode (lowint, SEGLOW);  
decode (highint, SEGHIGH);
```

Assert and Report

ASSERT is used to check expressions during simulation.

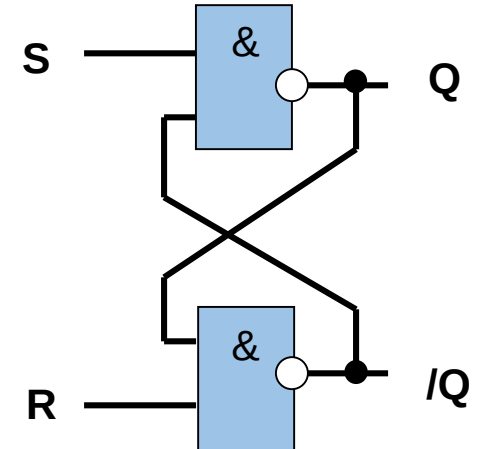


If condition is TRUE then do nothing, else print the message when the severity level is equal or higher than the level selected in the simulator.

Report can be used standalone.

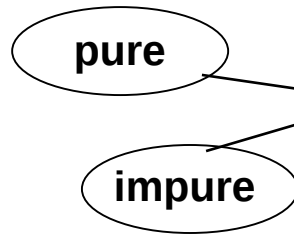
Assert (example)

```
ASSERT   R or S = '1'  
          REPORT  "R and S are both 0 "  
          SEVERITY warning;
```



```
ASSERT input_char >= '0' and input_char <= '9'  
          REPORT  "Input is not a digit!"  
          SEVERITY note;
```

pure : Does not depend on external values. Every call with same inputs gives the same output. (**default**)



impure : FUNCTION depends on external values. Every call can give different output even with the same inputs

Function

A Function is used in place of an expression or value.

```
FUNCTION identifier (interface_list) RETURN TYPE IS  
    declarations;  
BEGIN  
    sequential statements;  
    RETURN value;  
END FUNCTION identifier;
```

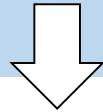
```
FUNCTION limit (actual, max, min: integer) RETURN integer IS  
BEGIN  
    IF actual > max THEN RETURN max;  
    ELSIF actual < min THEN RETURN min;  
    ELSE RETURN actual;  
    END IF;  
END FUNCTION limit;
```

```
Y <= limit (x, 126 , 17 );
```

Function Example from Cnt9999 Lab

```
FUNCTION int7seg (digit:integer range 0 to 9) RETURN std_logic_vector
```

```
variable dout: std_logic_vector(0 to 6);  
begin  case digit is  -- "abcdefg"  
  when 0    => dout := "0000001";  
  when 1    => dout := "1001111";  
  when 2    => dout := "0010010";  
  when 3    => dout := "0000110";  
  when 4    => dout := "1001100";  
  when 5    => dout := "0100100";  
  when 6    => dout := "0100000";  
  when 7    => dout := "0001111";  
  when 8    => dout := "0000000";  
  when 9    => dout := "0000100";  
  when others => dout := "1111111";  
end case;  
return dout;  
end function int7seg;
```



```
SEGLOW <= int7seg(lowint);  
SEGHIGH <= int7seg(highint);
```

Compare with
procedure call:

```
decode (lowint, SEGLOW);  
decode (highint, SEGHIGH);
```

NEXT, EXIT, RETURN, NULL

By using **NEXT** execution continues at the bottom of the LOOP-statement.

By using **EXIT** a loop will be left. Simple loops can only left by this command. Loop_level allows to leave outer loops from internal loops in nested loop constructions.

```
EXIT loop_label WHEN condition;  
EXIT loop_label;  
EXIT WHEN condition;  
EXIT;
```

RETURN is used to leave subroutines. A Procedure can be left by simply using **RETURN**;

A Function cannot alter the value of its input objects. The RETURN statement has to return a value.

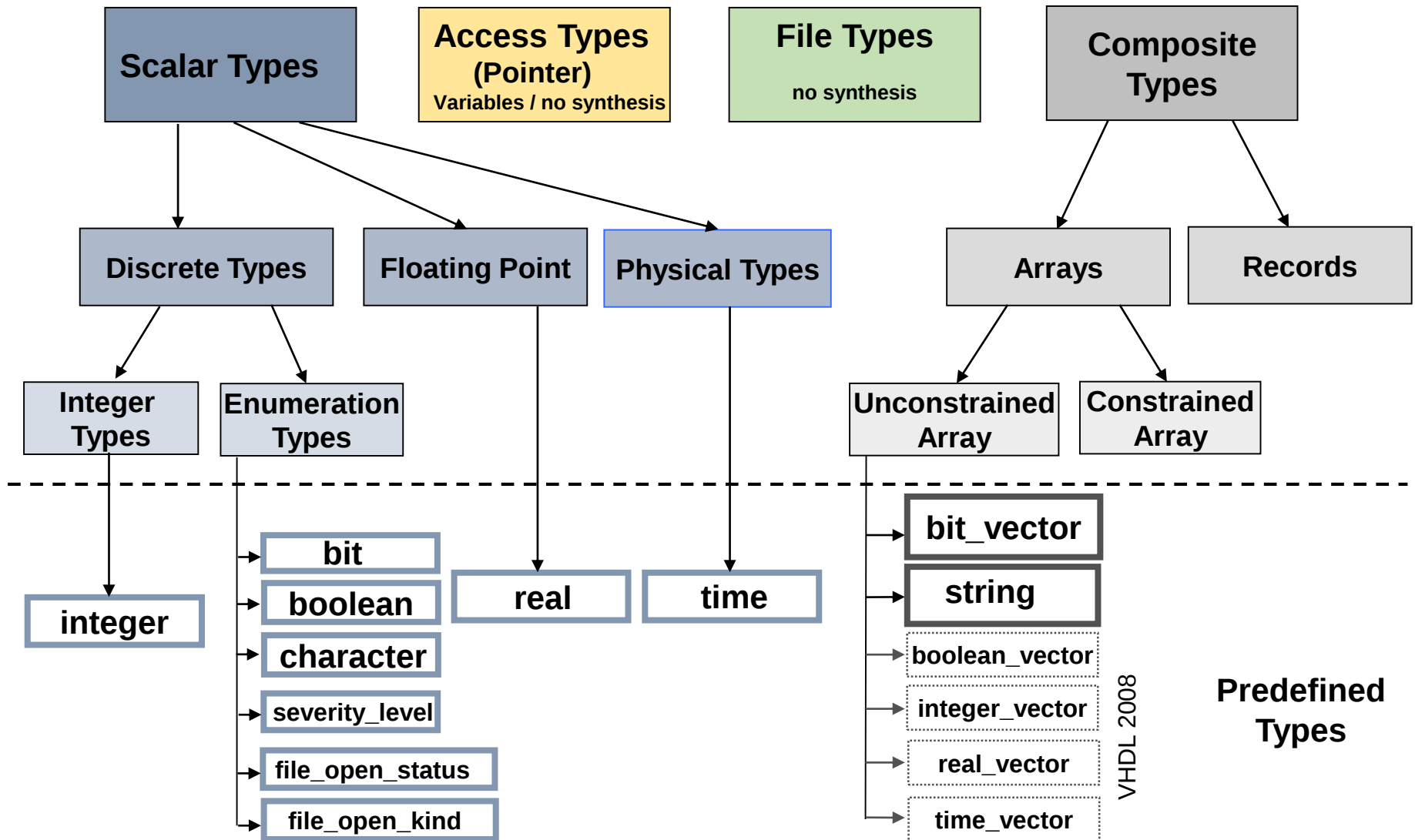
```
RETURN value;
```

The **NULL** statements can be used as a placeholder in arms of trees where no activity is wished:

```
NULL;
```

Types

VHDL is a strongly typed language => "verbose" language => "facilitated" with VHDL 2008



Scalar Types

Integer Floating Point Enumeration Physical (Files)

Type Declaration:

TYPE type_identifier **IS** type_definition;

Examples:

TYPE	my_real	IS	Range -5.0 to 5.0;	
TYPE	my_integer	IS	Range -128 to 127;	
TYPE	my_logic	IS	('X','Z','0','1');	
TYPE	color	IS	(red, green, blue);	} Enumeration
TYPE	state_type	IS	(reset, receive, send, wait);	
TYPE	severity_level	IS	(note, warning, error, failure);	

Integer is predefined (min.) -2,147,438,648 to 2,147,483,647 -> 32 Bit

Real is predefined (min.) -1.0E+38 to 1.0E+38 (32 Bit with 6 digits precision)

! A signal declared as an integer type has 32 bits --> Synthesis !

Scalar Types (Time)

Physical Type **TIME** is predefined

```
TYPE time IS RANGE       $-(2^{31})$  to  $2^{31}-1$ 
```

```
units
```

```
    fs;
```

```
    ps = 1000 fs;
```

```
    ns = 1000 ps;
```

```
    us = 1000 ns
```

```
    ms = 1000 us;
```

```
    sec = 1000 ms;
```

```
    min = 60 sec;
```

```
    hr  = 60 min;
```

```
end units;
```

tool dependent

! There is a space between value and unit

127 ns

0.127 us

Scalar Types (Physical Types)

```
TYPE resistance IS RANGE 0 to 1E9  
units
```

```
    ohm;  
    kohm = 1000 ohm;  
    Mohm = 1000 kohm;  
end units;
```

Integer values only!

```
TYPE length IS RANGE 0 to 1E9  
units
```

```
    um;  
    mm = 1000 um;  
    m   = 1000 mm;  
    inch = 25400 um;  
    ft   = 12 inch;  
end units;
```

-- primary unit is micron

Composite Types (Arrays I)

Arrays contain elements of unique type

Element Type

Constrained Array:

```
TYPE word IS ARRAY (31 DOWNT0 0) OF bit;

TYPE color IS (red, green, blue, magenta, yellow, cyan);
TYPE color_table_type IS ARRAY (color) OF bit;
TYPE three_color IS ARRAY (color RANGE red TO blue) OF bit;
```

```
TYPE intmatrix IS ARRAY (1 TO 10, 1 TO 20) OF integer;
```

```
TYPE raw, col IS RANGE 1 TO 1024;
```

```
TYPE pix_mat IS ARRAY (raw, col) OF bit;
```

2-dimensional

Unconstrained Array:

```
RANGE <> -- <> box, replaced by a range in object declaration
```

Predefined in VHDL:

```
TYPE bit_vector IS ARRAY (natural RANGE <>) of bit;
```

```
TYPE string IS ARRAY (positive RANGE <>) of character;
```

Composite Types (Arrays II)

Declaration of objects:

Constrained Array:

```
SIGNAL my_word_sig :    word;  
SIGNAL color_table:    color_table_type;
```

Unconstrained Array:

```
SIGNAL vec_sig1:        bit_vector ( 0 TO 31);  
SIGNAL vec_sig2:        bit_vector (7 DOWNT0 0);  
SIGNAL my_string:       string (1 TO 20);
```

Box <> replaced by
range specification

Referencing Arrays, Slices, Elements:

bit

```
y          <= vec_sig1(13);           -- A single element;  
z          <= color_table( green );    -- Index is enumeration type;  
vec_sig2   <= vec_sig1( 0 TO 7 );      -- A slice of the array;  
vec_sig1   <= vec_sig2;                -- Vectors of same dimension;  
vec_sig1( 0 TO 9) <= vec_sig2 ( 9 DOWNT0 0); -- Reverse index connection;
```

Composite Types (Usage of unconstrained Arrays)

-- Two constrained array types

TYPE intarray10 IS ARRAY (0 TO 9) OF integer;

TYPE intarray20 IS ARRAY (0 TO 19) OF integer;

-- Declaring signals using two different types

SIGNAL signal1, signal2 : intarray10;

SIGNAL signal3 : intarray20;

--Error in assignment, type mismatch

signal3 <= signal1 & signal2;

-- One unconstrained array type

TYPE intarray IS ARRAY (natural range <>) of integer;

--Declaring signals using the same type with different constraints (length)

SIGNAL signal1, signal2 : intarray (0 to 9);

SIGNAL signal3 : intarray (0 to 19);

--OK., because all operands are of the same type

signal3 <= signal1 & signal2;

Composite Types (Arrays III)

Content of Arrays (Aggregates):

Positional Association:

SIGNAL my_vector: bit_vector (0 TO 7);

my_vector <= ('0', '1', '0', '0', '0', '0', '0', '0');
my_enumerable <= (red , red, red, blue, yellow, green);
my_int_vector <= (1 , 25 , 34 , 112);

Position 0

Position 7

For 1-dimensional bit vectors: Bit strings

my_vector <= "01000000"; -- or B"01000000"
my_vector <= X"40"; -- 0100 0000
my_vector <= O"100" -- (0)01 000 000

Extensions (VHDL-2008):
7X"0F" - 7Bit - "0001111"
7UX"F" -Unsigned -"0001111"
7SX"F" -Signed - "1111111"
7SX"7" -Signed - "0000111"
my_vector <= 8D"64";

Named Association:

my_vector <= (0=>'0', 7=>'0', 6=>'0', 4=>'0', 3=>'0', 5=>'0', 2=>'0', 1=>'1');
my_vector <= (1=>'1', 0 | 2 | 3 | 4 | 5 | 6 | 7 => '0');
my_vector <= (0=>'0', 1=>'1', 2 TO 7 => '0');
my_vector <= (1=>'1', OTHERS => '0');

position => value

New in VHDL 2008 (Bitstring in named (mixed??) association):

my_vector <= ("0100" , OTHERS => '0');

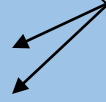
0100 at left side

Composite Types (Records)

```
TYPE    alu_instr IS (add, sub, adc, sbb);  
TYPE    address IS RANGE 0 TO 127;
```

```
TYPE    instruction  
RECORD  
opcode: alu_instr;  
op1, op2: address;  
END RECORD;
```

Elements of
different types



```
SIGNAL  code_bus:      instruction;  
SIGNAL  alu_control:   alu_instr;  
SIGNAL  src, dst:      address;
```

```
alu_control <= code_bus.opcode;  
src         <= code_bus.op1;  
dst         <= code_bus.op2;
```

record identifier



element identifier



Subtypes

```
TYPE    type1    IS RANGE 1 TO 12;  
TYPE    type2    IS RANGE 1 TO 12;
```

```
SIGNAL sig1:      type1;  
SIGNAL sig2, sig3: type2;
```

```
sig1      <= sig3; -- ERROR !  
sig2      <= sig3; -- OK.
```

Signals must be of the
same type at both sides

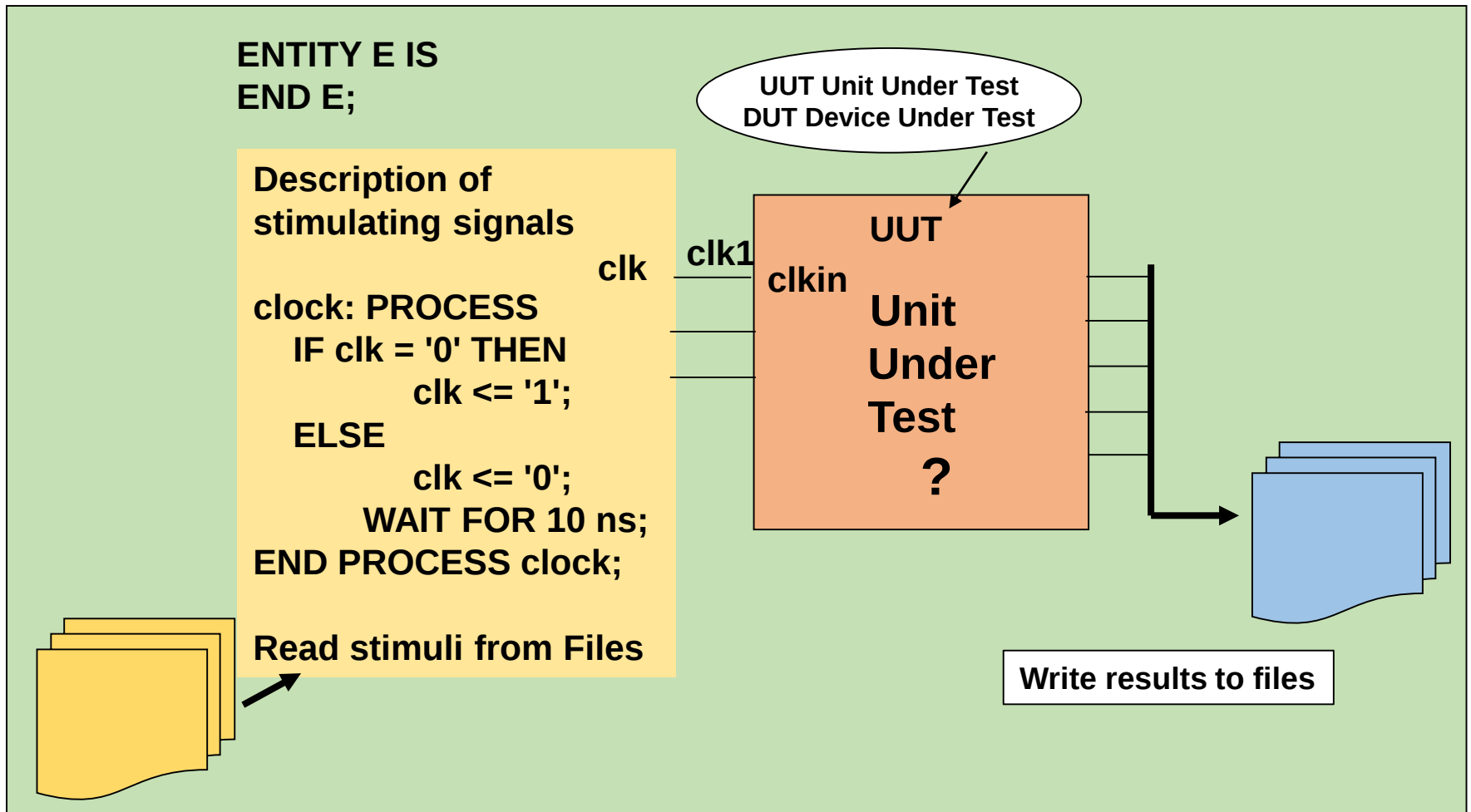
Subtypes are compatible with each other

```
SUBTYPE    reg_adr  IS integer RANGE 0 TO 127;  
SUBTYPE    word     IS bit_vector ( 15 DOWNT0 0);  
SUBTYPE    ascii    IS character;          -- synonym
```

Predefined subtypes in VHDL:

```
SUBTYPE positive IS RANGE 1 TO integer'high;  
SUBTYPE natural  IS integer RANGE 0 TO integer'high;
```

Testbench



Example for Synthesis Problems (Type Conversion) I

Simple Problem: Text defined as a constant with type „string“ shall be transferred as a byte or later bitwise by a serial interface. We need the characters as 8 bit vectors. Logically there is no conversion necessary because, for example Character-code for 'a' is 41h "converted" to std_logic_vector 41h. But we can't access single bits or slices from a character.

```
constant line1: string(1 to 16) := "abcdefghijklmnop";  
signal index: integer range 1 to 16;  
signal databyte: std_logic_vector (7 downto 0;
```

```
databyte <= CONV_STD_LOGIC_VECTOR(character'pos(line1(index)),8);
```

Worked well within Modelsim, but failed without comment in Xilinx Synthesis Tool ISE

Example for Synthesis Problems (Type Conversion) II

Solution : `type T is array(CHARACTER range NUL to DEL) of STD_LOGIC_VECTOR(7 downto 0);`

```
constant CHAR2STD: T :=
  ("00000000", "00000001", "00000010", "00000011",
   "00000100", "00000101", "00000110", "00000111",
   "00001000", "00001001", "00001010", "00001011",
   "00001100", "00001101", "00001110", "00001111",
   "00010000", "00010001", "00010010", "00010011",
   "00010100", "00010101", "00010110", "00010111",
   "00011000", "00011001", "00011010", "00011011",
   "00011100", "00011101", "00011110", "00011111",
   "00100000", "00100001", "00100010", "00100011",
   "00100100", "00100101", "00100110", "00100111",
   "00101000", "00101001", "00101010", "00101011",
   "00101100", "00101101", "00101110", "00101111",
   "00110000", "00110001", "00110010", "00110011",
   "00110100", "00110101", "00110110", "00110111",
   "00111000", "00111001", "00111010", "00111011",
   "00111100", "00111101", "00111110", "00111111",
   "01000000", "01000001", "01000010", "01000011",
   "01000100", "01000101", "01000110", "01000111",
   "01001000", "01001001", "01001010", "01001011",
   "01001100", "01001101", "01001110", "01001111",
   "01010000", "01010001", "01010010", "01010011",
   "01010100", "01010101", "01010110", "01010111",
   "01011000", "01011001", "01011010", "01011011",
   "01011100", "01011101", "01011110", "01011111",
   "01100000", "01100001", "01100010", "01100011",
   "01100100", "01100101", "01100110", "01100111",
   "01101000", "01101001", "01101010", "01101011",
   "01101100", "01101101", "01101110", "01101111",
   "01110000", "01110001", "01110010", "01110011",
   "01110100", "01110101", "01110110", "01110111",
   "01111000", "01111001", "01111010", "01111011",
   "01111100", "01111101", "01111110", "01111111");
end CHAR2STD;
```

Implementation in Synthesis ?

```
databyte <= CHAR2STD(line1(index));
```

Attributes I

Annotation of additional Information

Attributes for Types, Arrays, Signals, Entities
Predefined and User-defined Attributes

Identifier'attribute_identifier

Predefined Attributes for Scalar Types

type'LEFT	-	Value of the leftmost position in type declaration
type'RIGHT	-	Value of the rightmost position in type declaration
type'LOW	-	Least value of the type
type'HIGH	-	Greatest value of the type

TYPE color IS (red, green, blue, magenta, cyan, yellow);

color'LEFT = color'LOW = red
color'RIGHT = color'HIGH = yellow

In enumeration types values
are ordered from left to right

Attributes II

TYPE address IS RANGE 0 TO 127;

address'low = address'left = 0

address'high = address'right = 127

TYPE address IS RANGE 127 DOWNTO 0;

address'low = address'right = 0

address'high = address'left = 127

type'POS(X)

Position of value X in type declaration (starting left with 0)

type'VAL(P)

Value at the position P in type

TYPE color IS (red, green, blue, magenta, cyan, yellow);

color'POS(blue) = 2

color'VAL(3) = magenta

type'SUCC(X)

type'PRED(X)

Next value (greater/smaller)

type'LEFTOF(X)

type'RIGHTOF(X)

Value at Position +-1

color'LEFTOF(green) = color'PRED(green) = red

Attributes III

type'ASCENDING

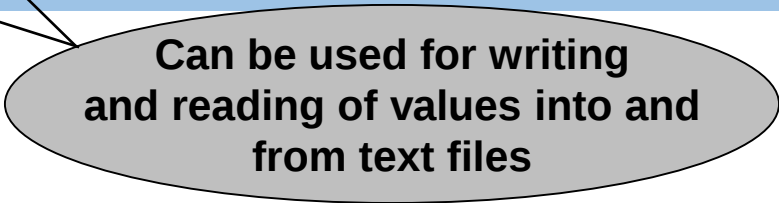
TRUE if type has an ascending range

type'IMAGE(x)

textual representation of the value x in type

type'VALUE(s)

value in type represented by the string s



**Can be used for writing
and reading of values into and
from text files**

Attributes IV

Attributes of (constrained) Arrays:

array'LEFT(n)	left bound of index range in dimension n
array'RIGHT(n)	right bound of index range in dimension n
array'LOW(n)	lowest index value in dimension n
array'HIGH(n)	highest index value in dimension n

array'LENGTH(n)	number of elements in dimension n
array'RANGE(n)	range definition in dimension n
array'REVERSE_RANGE(n)	reverse range definition in dimension n
array'ASCENDING	true if range in dimension n is accending

```
SIGNAL : vec_1 IS bit_vector ( 7 DOWNT0 0);  
vec_1'LEFT = vec1'HIGH          7  
vec_1'RIGHT = vec_1'LOW         0  
vec_1'LENGTH                   8  
vec_1'RANGE                     7 DOWNT0 0  
vec_1'REVERSE_RANGE             0 TO 7  
vec_1'ASCENDING                 FALSE
```

Default dimension is (1)

Attributes V

Attributes of Signals:

Attribute is a new signal

signal'DELAYED(time)	Original signal delayed by time
signal'STABLE(time)	New signal of type boolean TRUE if signal hasn't changed for time, else FALSE
signal'QUIET(time)	New signal of type boolean TRUE if no assignment has been made for time, else FALSE
signal'TRANSACTION	New signal of type BIT changing its value between '0' and '1' whenever a new assignment for the original signal was made

Attribute is a value (no new signal)

signal'EVENT	TRUE when during the last simulation cycle a change of the value occurred
signal'ACTIVE	TRUE when during the last simulation cycle an assignment to the signal was made
signal'LAST_EVENT	time from the last change of signal to actual time
signal'LAST_ACTIVE	time from the last assignment to signal to actual time
signal'LAST_VALUE	value of the signal before the last event (diff. '87/'93)

signal'DRIVING	True if the containing process drives the signal
signal'DRIVING_VALUE	value contributed by the containing process

VHDL'93 only

Attributes VI

Avoid them -> Portability

User defined attributes:

(Interesting because we find "software vendor defined attributes" sometimes)

Declaration:

ATTRIBUTE attribute_identifier : type;

Assignment:

ATTRIBUTE attribute_identifier OF target_identifier:target_kind IS value;

Signals, types components
can have identical identifiers

```
ATTRIBUTE pin_number : positive;  
TYPE logic_levels IS (cmos,ttl);  
ATTRIBUTE input_kind : logic_levels;
```

```
ATTRIBUTE pin_number OF in1:SIGNAL IS 12;  
ATTRIBUTE input_kind OF in1, in3, in4 : SIGNAL IS cmos;
```

```
in1'input_kind = cmos
```

Attribute FOREIGN

The attribute Foreign enables interfaces to non VHDL tools and libraries

Content is **tool dependent !! → Not portable !!**

String-attribute for architectures and subroutines

Example:

```
PROCEDURE create_window (size_x, size_y: IN positive;  
                          status: OUT boolean);
```

```
ATTRIBUTE foreign OF create_window: PROCEDURE IS
```

```
"bind to open_window" &
```

```
"parameter x_size maps to size_x: IN positive" &
```

```
"parameter y_size maps to size_y: IN positive" &
```

```
"parameter state maps to status: OUT boolean";
```

```
END PROCEDURE create_window;
```



Non VHDL

It depends on the tool, what is done with the attribute string in the environment.
Architectures or Procedures are replaced by implementations in other languages
or special external hardware ...

Attributes (SYNOPSISYS-Example)

ATTRIBUTE enum_encoding

TYPE color IS (red, yellow, black);

ATTRIBUTE enum_encoding : STRING;

ATTRIBUTE enum_encoding OF color : TYPE IS "01 10 11";



Synthesis Software (Design Compiler) reads the attributes and encodes the abstract enumeration values as binary representations. (There exist also other ways to control encoding.)

File Types and Files

It is possible to store values in files.
Files contain values of one type only.

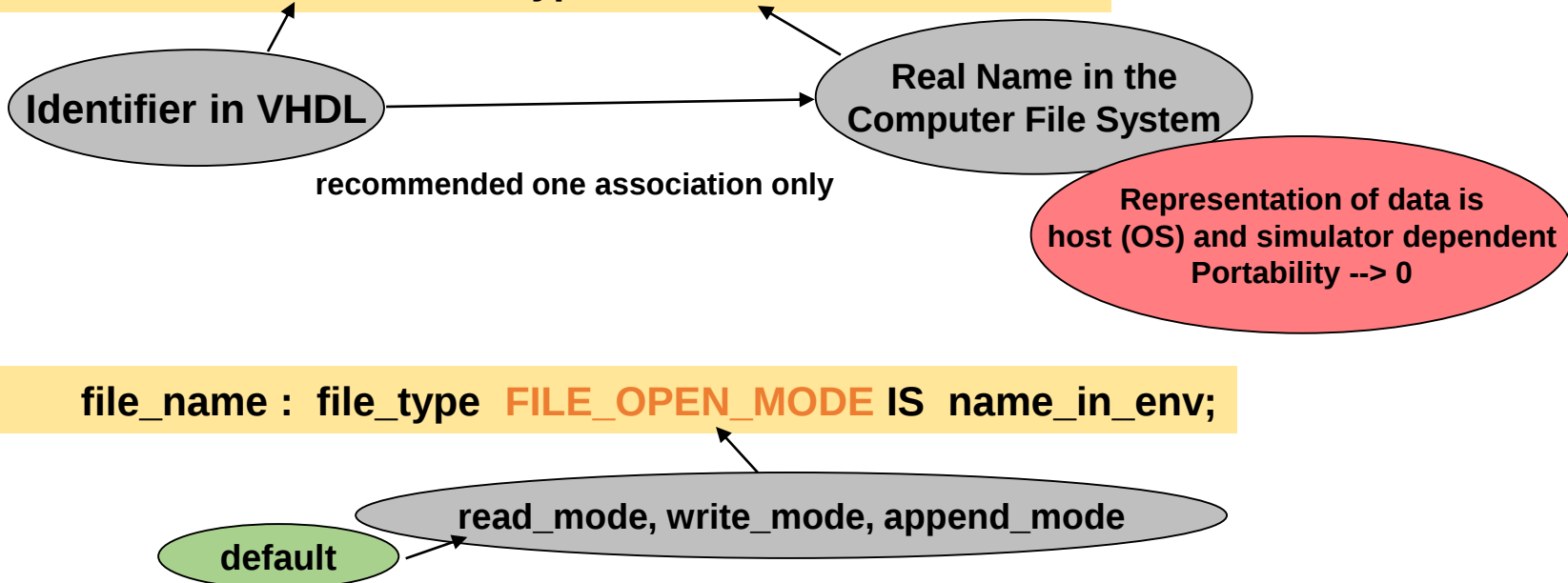
VHDL-87 uses
different syntax!
-> incompatible

Declaration of a file type:

```
TYPE    file_type IS FILE OF element_type;
```

Declaration of a file using file_type:

```
FILE    file_name : file_type IS "name_in_env";
```



Using Files (VHDL 93 +)

Read:

```
TYPE file_type    IS FILE OF element_type;  
FILE  file_name : file_type READ_MODE IS myfile;
```

```
PROCEDURE read (FILE myfile: file_type; value: OUT element_type);
```

```
FUNCTION endfile (FILE myfile: file_type) RETURN boolean;
```

Usage example:

```
if not endfile(myfile) then  
  read(myfile, mysignal);  
end if;
```

function with additional
parameter "length" available

Write:

```
TYPE file_type    IS FILE OF element_type;  
FILE  file_name : file_type WRITE_MODE IS myfile;
```

```
PROCEDURE write (FILE myfile: file_type; value: IN element_type);
```

Usage example: write(myfile, mysignal)

For advanced usage of files:

Procedures : "file_open" , "file_close" , Type: "file_open_status"

TEXTIO

Way out of the host and simulator dependency of data representation in files

**Textfiles (type "text") contain elements of type string.
Elements (strings) are accessed by a pointer line.**

input, output predefined for
simulators command-line
(keyboard, display)

```
FILE input : text OPEN read_mode IS "std_input";  
FILE output : text OPEN write_mode IS "std_output";
```

Use your (VHDL-) name here

Use your (filesystem-) name here

```
PROCEDURE read (L: INOUT line; value OUT bit_vector );  
PROCEDURE read (L: INOUT line; value OUT bit_vector, good: OUT boolean);
```

Procedures for
different types available

bit, bit_vector, boolean, character,
integer, real, string, time

```
PROCEDURE write (L: INOUT line; value IN bit_vector ;  
justified: IN side:=right; field: IN width := 0 .. );
```

right " 001"
left "001 "

Minimal number
of characters
(spaces included)

additional:
"unit" for time, "digits" for real

use IMAGE and VALUE attributes
for user defined types

```
write (line, state_variable_type'image(state_variable));
```

TEXTIO - Example from Lab "cnt9999"

use std.textio.all; -- At top of VHDL description

Generic (Short_time: integer range 0 to 25000 := 25000;
Long_time: integer range 0 to 1000 := 1000); -- Parameter in entity declaration

file integer_file: text open write_mode is "integer_file.txt"; --Declaration in architecture

Writetofile: process(dig_array) - - Process to store the value if a digit changes

Variable simtime: time;

Variable integer_line: line;

begin

if reset_int'last_value = '1' then --Start time with falling edge of reset

simtime := (25000/Short_time * 1000/Long_time) * reset_int'last_active; --Correction of acceleration

else simtime := now; --Before reset ends use actual time

end if;

write(integer_line, simtime, justified => right, field=> 20, unit => ms);

write(integer_line, string'(": "));

for i in 3 downto 1 loop

write(integer_line, dig_array(i));

- -We can use the 8-bit character set: here Spanish upside down exclamation mark

write(integer_line, string'(" " & character'val(161) & " "));

end loop;

write(integer_line, dig_array(0));

writeline(integer_file, integer_line);

end process Writetofile;

Result in "integer_file.txt

0 ms: 0 | 0 | 0 | 0

0 ms: 9 | 9 | 9 | 5

1000 ms: 9 | 9 | 9 | 6

2000 ms: 9 | 9 | 9 | 7

TEXTIO - Example from Lab "vga"

We want to see the vga-output as an image during simulation.

Solution: Generate a Bitmap File

Simple structure: Header + $n \times 3$ Bytes (blue,green,red)

Problem:

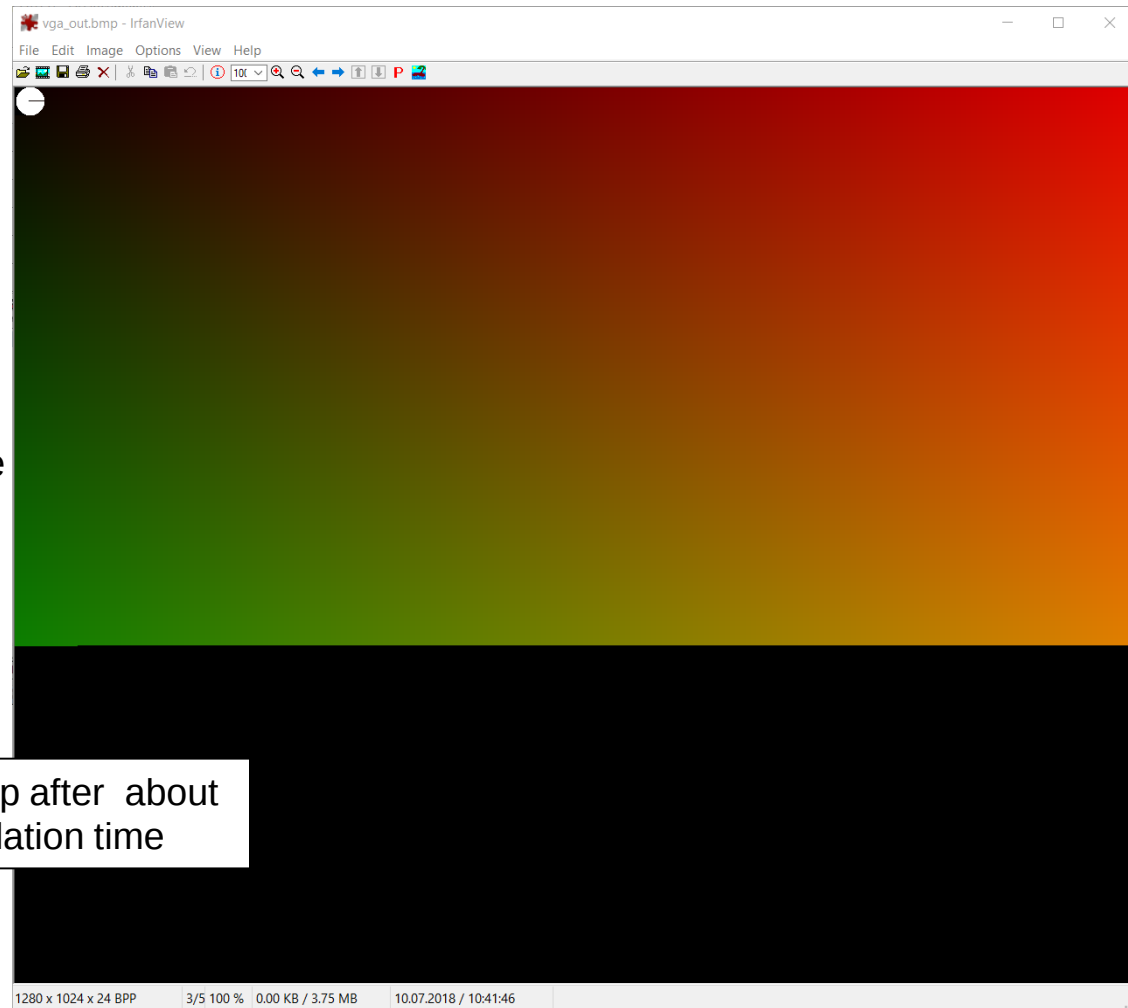
Long simulation times

16.7 ms simulation time per picture

Modelsim_64Bit Windows10,
i7-4790 CPU @ 3.60GHz, 32 GB

188 s /picture -> 11.25 s / ms simulation time
(same time for project on network or local)

vga_out.bmp after about
10 ms simulation time



TEXTIO - Example from Lab "vga"

We want to see the vga-output as an image during simulation.

Solution: Generate a Bitmap File

Simple stucture: Header + n x 3 Bytes (blue,green,red)

Bitmap: block

```
type bitmap_hecker_type is array (0 to 53) of integer;
constant bitmap_header: bitmap_hecker_type :=
( 16#42#, 16#4D#, --"BM"
  16#36#, 16#00#, 16#3C#, 16#00#, -- Size of BMP: 1280x1024x3+54 = 3932214 = 003C0036H
  16#00#, 16#00#, 16#00#, 16#00#,
  16#36#, 16#00#, 16#00#, 16#00#, -- Offset of pixel array to start of bitmap file (standard 54 Bytes = 36H)
  --DIB (Device Independent Bitmap) Header
  16#36#, 16#00#, 16#00#, 16#00#, -- 40 Bytes DIB header length
  16#00#, 16#05#, 16#00#, 16#00#, -- Width of bitmap 1280 = 500H
  16#00#, 16#FC#, 16#FF#, 16#FF#, -- Height of bitmap 1024 = 400H, top to bottom negative FFFFC00H
  16#01#, 16#00#, -- one color plane
  16#18#, 16#00#, -- 24 bits per pixel (one byte/color blue/red/green)
  16#00#, 16#00#, 16#00#, 16#00#, -- no compression
  16#36#, 16#00#, 16#3C#, 16#00#, -- Size of raw bitmap data: 1280x1024x3 = 3932160 = 3C0000
  16#13#, 16#0B#, 16#00#, 16#00#, -- Print resolution horizontal 72 DPI (2835 pixel/metre)
  16#13#, 16#0B#, 16#00#, 16#00#, -- Print resolution vertical 72 DPI (2835 pixel/metre)
  16#00#, 16#00#, 16#00#, 16#00#, -- 0 colors in palette
  16#00#, 16#00#, 16#00#, 16#00#); -- 0 important colors

type bitmap_file_type is file of character;
file bitmap_file: bitmap_file_type open write_mode is "vga_out.bmp";
signal header_flag : boolean := true;
signal virgin: boolean := true;
```

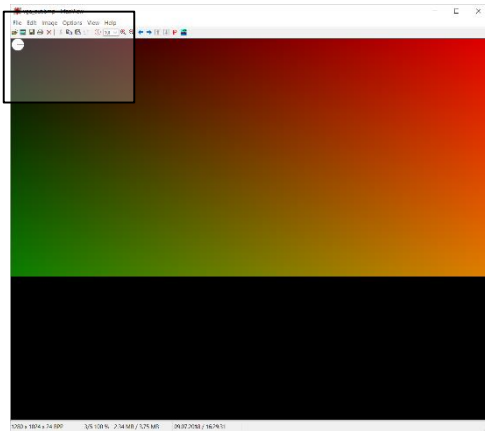
Files: Example from Lab "VGA"

```
begin
Write_BMP: process(VIDEOCLK)
  begin
    if rising_edge(VIDEOCLK) then
      if VSYNC = '0' or virgin= true then
        virgin <= false;
        if header_flag = true then
          header_flag <= false;
          file_close(bitmap_file);
          file_open(bitmap_file,"vga_out.bmp",write_mode);
          for i in 0 to 53 loop
            write(bitmap_file, character'val(bitmap_header(i)));
          end loop;
        end if;
      end if;
      if BLANK = '1' then
        header_flag <= true;
        --write pixels
        write(bitmap_file,character'val(to_integer(unsigned(blue))));
        write(bitmap_file, character'val(to_integer(unsigned(green))));
        write(bitmap_file, character'val(to_integer(unsigned(red))));
      end if;
    end if;
  end process Write_BMP;
end block Bitmap;
```

Here we must look into the textio library
What types are available?
What is their format in the files?
Example: Integer always Bytes (32 Bit)

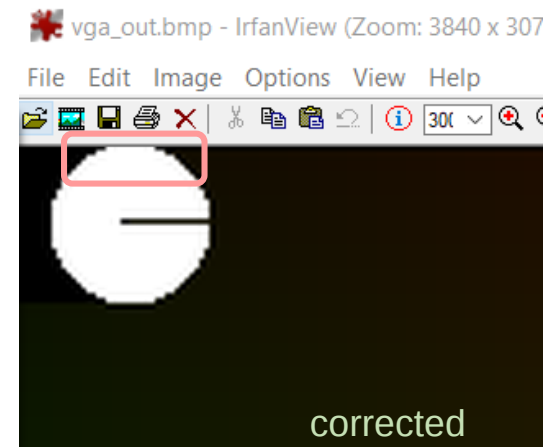
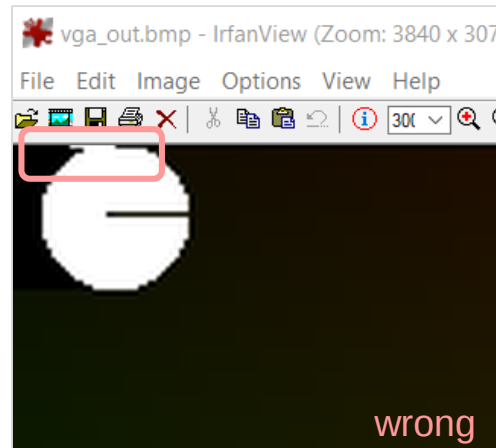
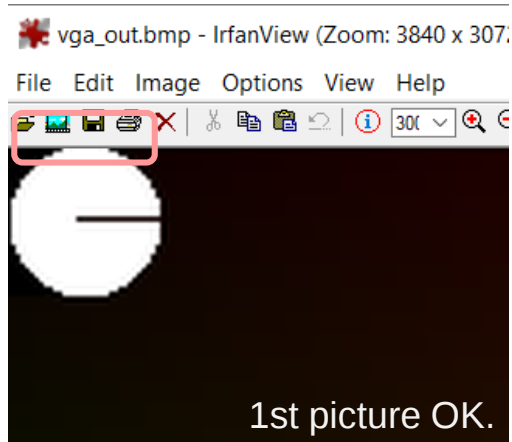
T'VAL(X) is the value of discrete type T at integer position X.

Find Errors in VGA-Lab Design analyzing Bitmap File



Increment of the pacman position was too late, effective in the second line for the first time.

Changed to an earlier point of time: start of vblank instead of end of vblank



```

newballpos:
  process (vclk)
begin
  if (vclk'event and vclk = '1') then
    vblankold <= vblank;
    if (((vblankold xor vblank) = '1') and ( vblank = '1' )) then --detect end of picture
    ....
    ball_hpos <= ball_hpos + 5;
  
```

changed from '0' to '1'

↙

Structural Description

Component Declaration

```
COMPONENT      identifier IS  
    GENERIC (parameter_interface_list);  
    PORT      (port_interface_list);  
end COMPONENT identifier;
```

optional

Component Instantiation

```
inst_label: COMPONENT identifier  
    GENERIC MAP (locals => actuals)  
    PORT MAP (locals => actuals);
```

no ;

or "others" or "all"

Component Specification

```
FOR inst_label : identifier USE ENTITY  
library_name.entity_name(architecture_name);
```

(Optional: Without specification the component with the entity_name identical to the identifier is taken from library work. The most recently analyzed (newest) architecture is used)

Direct Component Instantiation

```
entity    and_gate is
    port ( a, b in bit; y out bit);
end entity and_gate;

architecture arch1 of and_gate is
begin
    y <= a and b after td;
end architecture arch1;
```

```
U112:    entity work.and_gate(arch1)
        port map ( a => ext1 , b => ext2, y => ext3);
```

Instantiation_label: entity library_name.entity_name(architecture_name)
.....

Not included in VHDL'87

Generics

Generics are parameters in VHDL

Generics are constants with mode "IN".

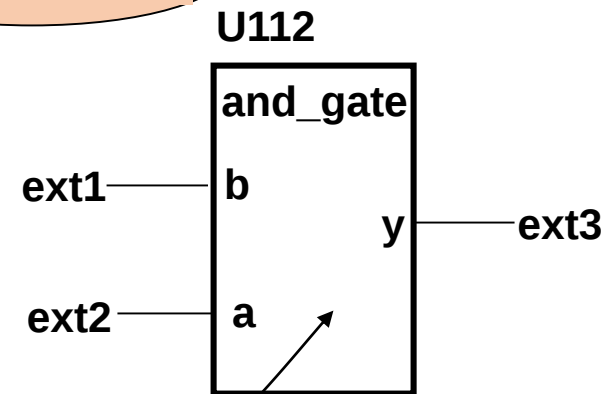
Generics are used to parametrize behaviour/structure of an entity

```
entity    and_gate is
    generic (td: time := 2 ns);
    port ( a, b in bit; y out bit);
end entity and_gate;
```

```
architecture behav of and_gate is
begin
```

```
    y <= a and b after td;
end architecture behav;
```

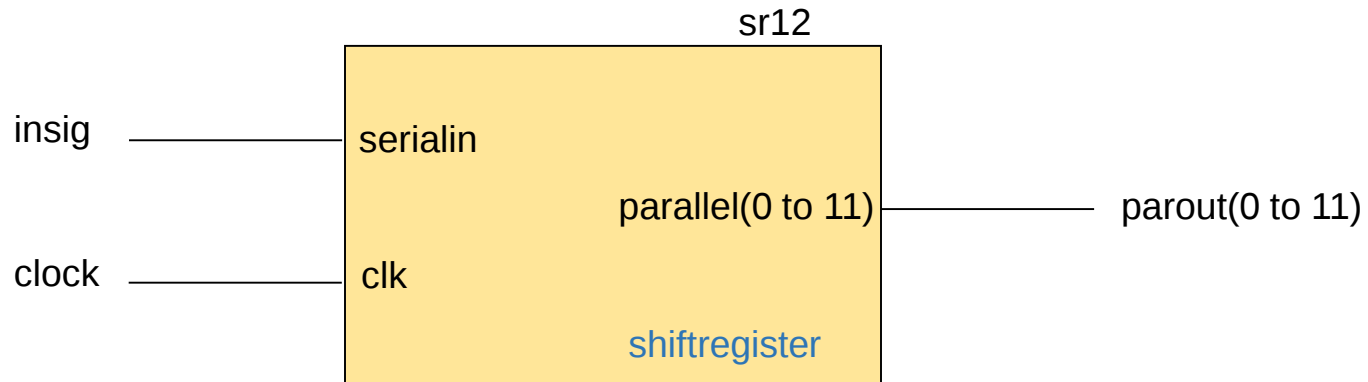
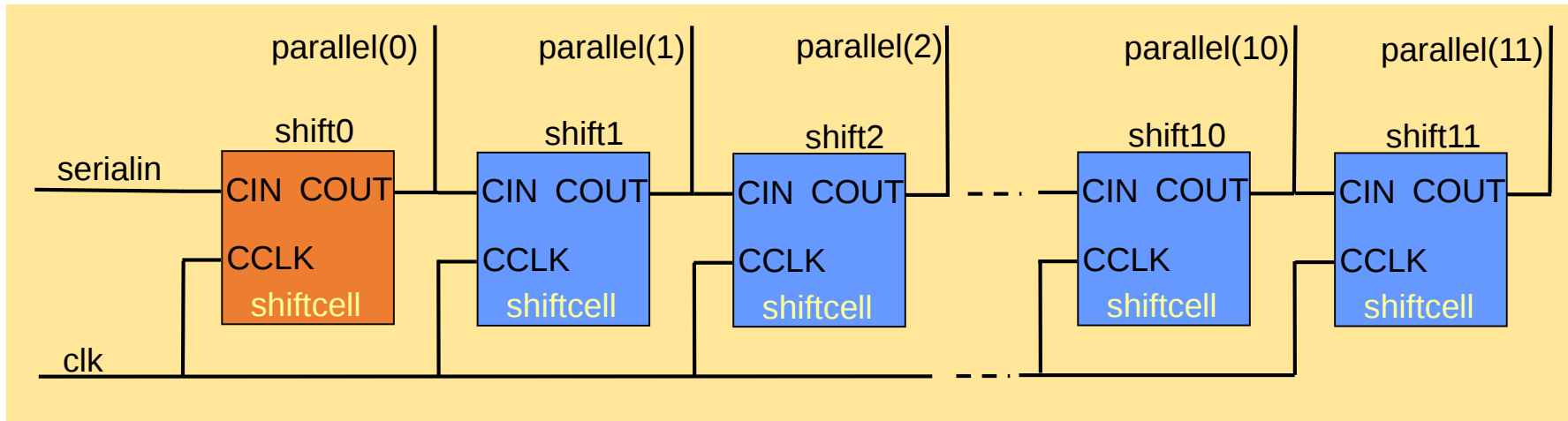
default value



actual value

```
U112:    entity work.and_gate(behav)
    generic map (td => 3 ns)
    port map ( a => ext2 , b=> ext1, y=> ext3);
```

Shift Register I



Generate Statement

Description of replicated subsystems (processes or component instantiations)

Label: **generation scheme**

FOR identifier IN discrete_range
IF condition

GENERATE
concurrent statements
END GENERATE label;

VHDL-2008 IF with
ELSIF, ELSE branches
dto. CASE ..

Will become:
shift1, shift2 ...

ENTITY shiftregister IS

PORT (clk, serialin: IN std_logic;

 parallel: BUFFER std_logic_vector(0 TO 11));

END shiftregister;

ARCHITECTURE structure OF shiftregister IS

COMPONENT shiftcell

PORT (cclk; cin : IN std_logic;

 cout : OUT std_logic);

END COMPONENT shiftcell;

shift0 : shiftcell (clk, serialin, parallel(0));

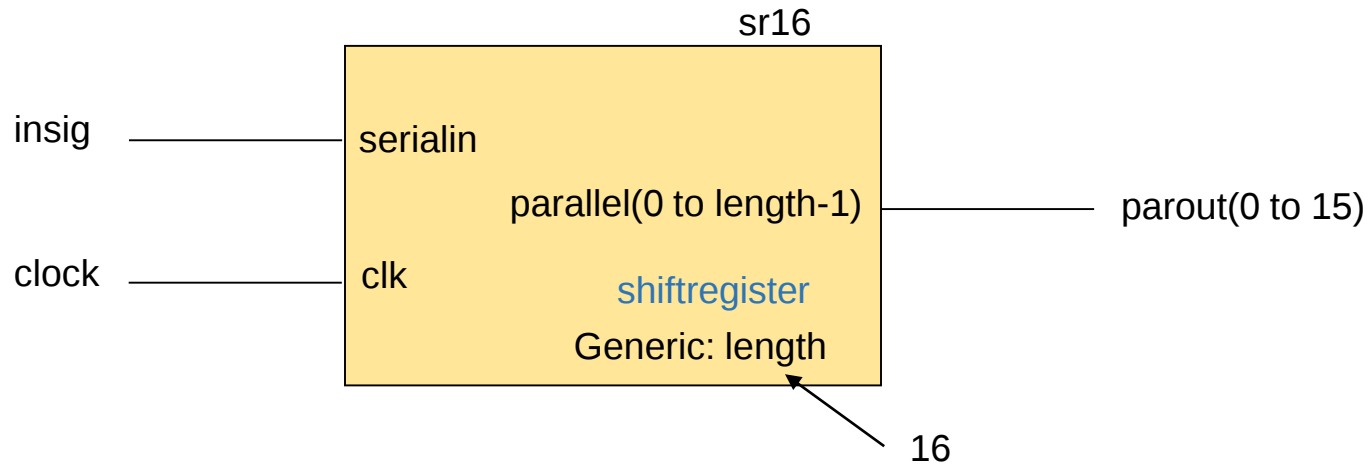
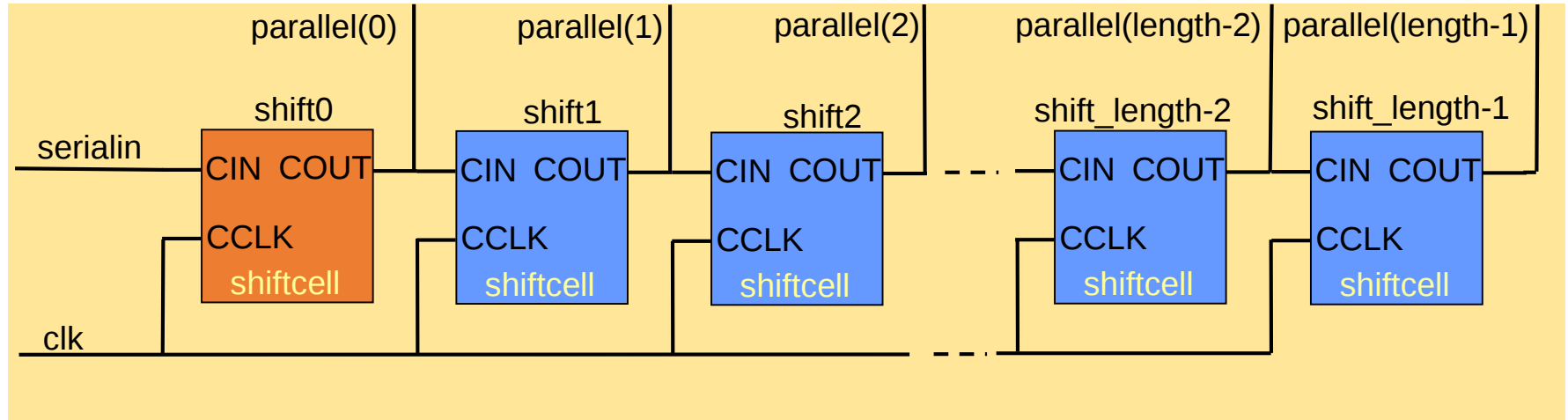
gen_shift: **FOR** i in 1 TO 11 **GENERATE**

 shift: shiftcell **PORT MAP** (clk, parallel(i-1), parallel(i));

END GENERATE gen_shift;

END ARCHITECTURE structure;

Shift Register II



Generate and Generic

ENTITY shiftregister IS

GENERIC (length : positive := 12);

PORT (clk, serialin: IN std_logic;

parallel: BUFFER std_logic_vector(0 TO length-1));

END shiftregister;

ARCHITECTURE structure OF shiftregister IS

COMPONENT shiftcell

PORT (cclk; cin : IN std_logic;

cout : OUT std_logic);

END COMPONENT shiftcell;

shift0 : shiftcell (clk, serialin, parallel(0));

gen_shift: **FOR** i in 1 TO length-1 **GENERATE**

shift: shiftcell **PORT MAP** (clk, parallel(i-1), parallel(i));

END GENERATE gen_shift;

END ARCHITECTURE structure;

sr16: shiftregister
GENERIC MAP (length=>16)
PORT MAP (clock, insig, parout);

Generate - Conditional Code

Exclude code from synthesis

```
if Simulation = '1' generate
    Code
end generate;
```

Better than metacomments (because portable)
Disadvantage: Parameter Simulation must be handled

```
-- synthesis translate_off
    Code
--synthesis translate_on
```

metacomments

Write Code depending on the value of static parameter_x

```
if Parameter_x = '1' generate
    Code 1
end generate;
```

```
if Parameter_x = '0' generate
    Code 2
end generate;
```

VHDL-2008 IF with
ELSIF, ELSE branches
dto. CASE ..

Resolution Function

```
TYPE    bit4v    IS ( 'X' , '0' , '1' , 'Z' );  
TYPE    bit4v_vec IS ARRAY (integer RANGE <> OF bit4v);
```

```
FUNCTION wired_or (input: bit4v_vec) RETURN bit4v IS
```

```
    VARIABLE result : bit4v := '1';
```

```
BEGIN
```

```
    FOR i IN input'RANGE LOOP
```

```
        IF input(i) = '0' then
```

```
            result := '0'; EXIT ;
```

```
        ELSIF input(i) = 'X' THEN
```

```
            result := 'X'; NEXT;
```

```
        END IF,
```

```
    END LOOP;
```

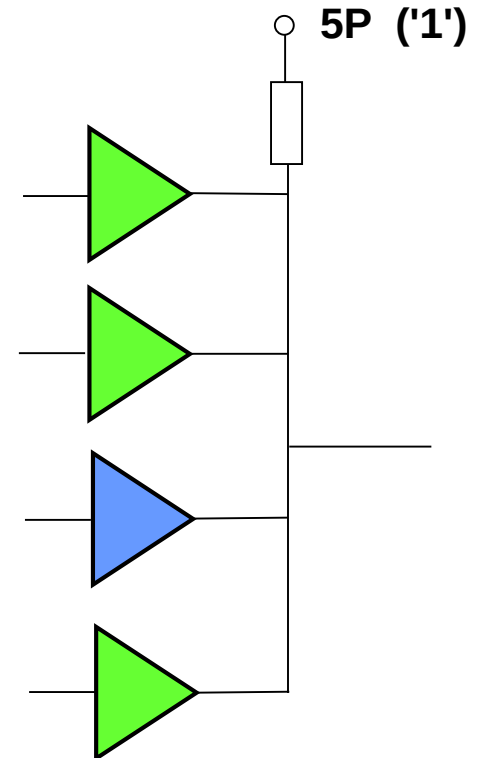
```
    RETURN result;
```

```
END FUNCTION wired_or;
```

```
SUBTYPE bit4v_res IS wired_or bit4v;
```

```
SIGNAL sig27:    wired_or bit4v;
```

```
SIGNAL sig28:    bit4v_res;
```



Alias

By help of an alias we can declare a new name for an object

```
SIGNAL ProgRegister: std_logic_vector(7 DOWNTO 0);  
ALIAS EnableInterrupt:std_logic IS ProgRegister(7);  
ALIAS Bitrate:std_logic_vector(4 downto 0) IS ProgRegister(6 DOWNTO 4);  
ALIAS Prescaler:std_logic_vector(3 downto 0) IS ProgRegister(3 DOWNTO 0);
```

**VHDL allows the simple alias without a subtype (red in the Examples),
but some software (SYNOPTSYS) needs the subtype declaration**

Package

Packages contain declarations and subroutine implementations

```
PACKAGE mypack IS
```

```
CONSTANT    frame_length : POSITIVE := 1500;  
TYPE        instruction IS (add , sub, adc, sbb );  
SUBTYPE     word IS bit_vector (31 DOWNTO 0);
```

```
FUNCTION ">" (a,b : in WORD) RETURN boolean;  
PROCEDURE add (term1, term2: IN bit_vector;  
               sum: OUT bit_vector;  
               carry: OUT bit)
```

```
END PACKAGE mypack;
```

Package Body

Necessary only if the package contains subprograms

Operator
Overloading

Repeated from
Declaration

```
PACKAGE BODY    mypack IS
  FUNCTION ">" (a,b : in WORD) RETURN boolean IS
  VARIABLE result: BOOLEAN := FALSE;
  BEGIN
    FOR i IN 31 DOWNTO 0 LOOP
      IF a(i) = '1' AND b(i) = '0' THEN result := TRUE;  EXIT;
      IF a(i) = '0' AND b(1) = '1' THEN result := FALSE; EXIT;
      END IF;
    END LOOP;
    RETURN result;
  END FUNCTION ">"

  .... Bodies of other Functions and Procedures

END PACKAGE BODY mypack;
```

Analyze package declaration before package body !

Configurations

Specify which description
will be used actually

If an entity "**counter**" with one architecture "**simple**" exists in the library **WORK** the following configurations are equivalent for an architecture "**testbench**" in the entity "**E**"

```
CONFIGURATION count_test2 OF E IS
  FOR testbench
    FOR UUT:counter USE entity WORK.counter(simple);
  END FOR;
END FOR;
END count_test2;
```

ALL

Without a given architecture use
the newest architecture

```
CONFIGURATION count_test4 OF E IS
  FOR testbench
  END FOR;
END count_test4;
```

The configuration can be part of a description file or a separate file.

Equivalent Configurations

If an entity "**counter**" with one architecture "**simple**" exists in the library **WORK** the following configurations are equivalent for an architecture "**testbench**" in the entity "**E**"

```
CONFIGURATION count_test1 OF e IS
  FOR testbench
    FOR ALL:counter USE entity WORK.counter(simple);
  END FOR;
END FOR; END count_test1;
```

```
CONFIGURATION count_test2 OF e IS
  FOR testbench
    FOR UUT:counter USE entity WORK.counter(simple);
  END FOR;
END FOR; END count_test2;
```

```
CONFIGURATION count_test3 OF e IS
  FOR testbench
    FOR UUT:counter USE entity WORK.counter;
  END FOR;
END FOR; END count_test3;
```

```
CONFIGURATION count_test4 OF e IS
  FOR testbench
  END FOR; END count_test4;
```

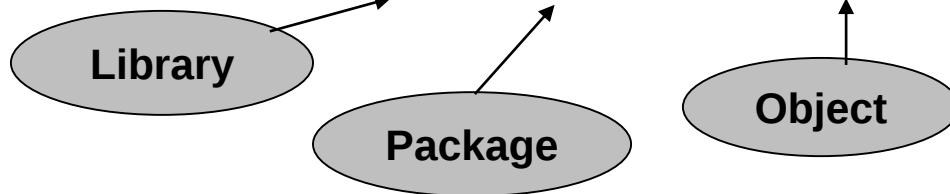
Libraries

```
LIBRARY library_identifier;
```

Defined in
Tool Setup

**WORK and STD are predefined .
WORK users working library
STD contains Packages STANDARD and TEXTIO**

Objects can be located by long names like
SIGNAL a: IEEE.std_logic_1164.std_logic;



```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.all
```

```
Signal a: std_logic;
```

```
LIBRARY mylib;  
USE mylib.pack1;           -- pack1.objectname  
USE mylib.pack1.sreg;      -- sreg from pack1 is visible directly  
USE mylib.pack1.ALL;       -- all objects in pack1 are visible
```

IEEE1164

TYPE std_ulogic -> std_logic (is a **resolved type**)

'U'	uninitialized
'X'	forced unknown
'0'	forced 0
'1'	forced 1
'Z'	high impedance
'W'	weak unknown
'L'	weak 0
'H'	weak 1
'-'	don't care

forced : 0 ... n x 100 Ohm

high impedance : n x MOhm

weak : n x KOhm

Library includes type declarations for std_logic and std_logic_vector and functions (overloading operators like AND, OR)

IEEE1164

Example: Exact function for rising clock edge

A rising edge in std_logic may be: '0' to '1', '0' to 'H', 'L' to '1' or 'L' to 'H'

```
SUBTYPE X01 IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
FUNCTION To_X01(s : std_ulogic) RETURN X01;
FUNCTION rising_edge(SIGNAL s : std_ulogic) RETURN BOOLEAN;
```

```
FUNCTION rising_edge(SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
RETURN(s'EVENT AND (To_X01(s) = '1') AND (To_X01(s'LAST_VALUE) = '0'));
END;
```



Event on s
detected



Value is '1' or 'H' now



Value was '0' or 'L' before

Compare with often used: `clk'event and clk = '1'`
Includes 'Z','X','U' to '1', but does not include 'L' to 'H' .., when std_logic is used

Operator Overloading

Operators must match the types of input and output signals

No problem for predefined types:

Outbit <= Inbit1 AND Inbit2;

But there can exist other types:

**Out_IEEE_bit <= In_IEEE_bit1 AND In_IEEE_bit2;
OUT_my_bit <= IN_my_bit1 AND In_my_bit1;**

**Compiler searches for an operator declaration for the given types.
Operators are declared as functions.**

IEEE1164 Operator Overloading

Example: AND-function

```
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;  
  
FUNCTION "and" (l : std_ulogic; r : std_ulogic) RETURN UX01 IS  
BEGIN  
  RETURN(and_table(l, r));  
END "and";
```

```
CONSTANT and_table : stdlogic_table := (
```

```
--  -----  
--  | U  X  0  1  Z  W  L  H  -  | |  
--  -----  
  ('U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U'), -- | U |  
  ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'), -- | X |  
  ('0', '0', '0', '0', '0', '0', '0', '0', '0'), -- | 0 |  
  ('U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'), -- | 1 |  
  ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'), -- | Z |  
  ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'), -- | W |  
  ('0', '0', '0', '0', '0', '0', '0', '0', '0'), -- | L |  
  ('U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'), -- | H |  
  ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X') -- | - |
```

```
FUNCTION "and"(l : std_ulogic; r : std_ulogic) RETURN UX01 IS  
BEGIN  
  RETURN(and_table(l, r));  
END "and";
```

Qualified Expressions

Sometimes the search for an Operator returns ambiguous results.

Example:

```
Y <= my_function ('0', x);
```

If my_function is declared for the type of y as the result, x as second operand and for the types BIT and CHARACTER of the first operand the operator cannot be selected.

Qualified Expression adds the type of the object:

```
Y <= my function ( Bit('0'),x);
```

Use it, when something may be ambiguous. Mostly when you analyzed (compile) the VHDL-description and got an error.

Package std_logic_arith

```
TYPE unsigned IS ARRAY (natural RANGE <>) OF std_logic;  
TYPE signed   IS ARRAY (natural RANGE <>) OF std_logic;
```

```
signed'("0101") -- represents +5  
signed'("1011") -- represents -5
```

```
SIGNAL S_SIG: signed (5 downto 0); -- bit 5 is the sign  
SIGNAL S_SIG: signed (0 to 5);      -- bit 0 is the sign
```

`conv_signed(signed'("110"), 8)` gives `"11111110"` as result

```
FUNCTION conv_std_logic_vector (ARG: unsigned; SIZE: integer) RETURN std_logic_vector;
```

FUNCTION "+" (L: unsigned; R: unsigned)	RETURN unsigned;
FUNCTION "+" (L: signed; R: signed)	RETURN signed;
FUNCTION "+" (L: unsigned; R: signed)	RETURN signed;
FUNCTION "+" (L: signed; R: unsigned)	RETURN signed;
FUNCTION "+" (L: unsigned; R: integer)	RETURN unsigned;
FUNCTION "+" (L: integer; R: unsigned)	RETURN unsigned;
FUNCTION "+" (L: unsigned; R: integer)	RETURN std_logic_vector;

Package numeric_std

```
TYPE unsigned IS ARRAY (natural RANGE <>) OF std_logic;  
TYPE signed   IS ARRAY (natural RANGE <>) OF std_logic;
```

```
signed'("0101") -- represents +5  
signed'("1011") -- represents -5
```

```
SIGNAL S_SIG: signed (5 downto 0); -- bit 5 is the sign  
SIGNAL S_SIG: signed (0 to 5);      -- bit 0 is the sign
```

```
to_signed(signed'("110"), 8) gives "11111110" as result
```

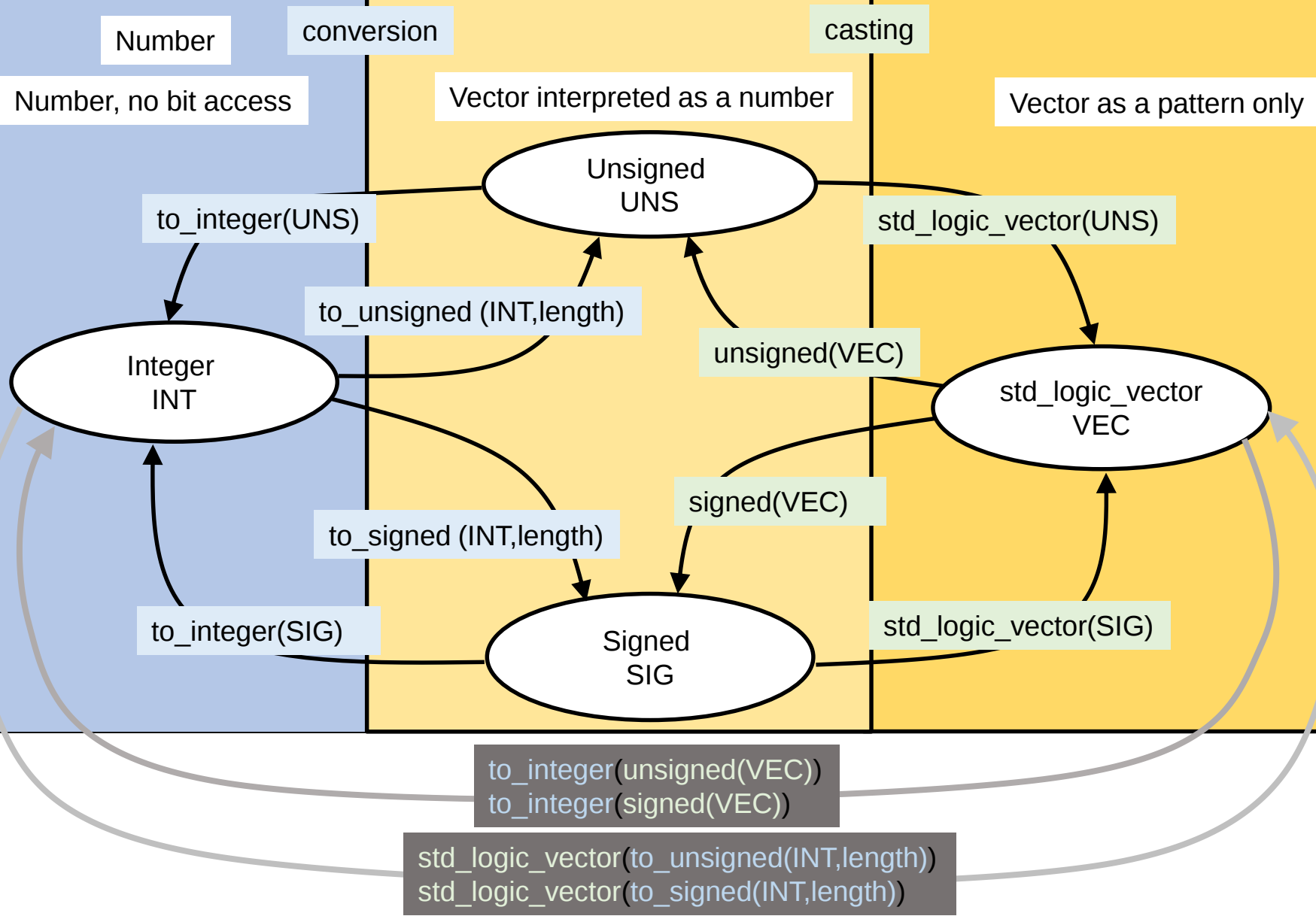
```
std_logic_vector (ARG: unsigned) RETURN std_logic_vector;
```

FUNCTION "+" (L: unsigned; R: unsigned)	RETURN unsigned;
FUNCTION "+" (L: signed; R: signed)	RETURN signed;
FUNCTION "+" (L: unsigned; R: natural)	RETURN unsigned;
FUNCTION "+" (L: natural; R: unsigned)	RETURN unsigned;
FUNCTION "+" (L: signed; R: integer)	RETURN signed;
FUNCTION "+" (L: integer; R: signed)	RETURN signed;

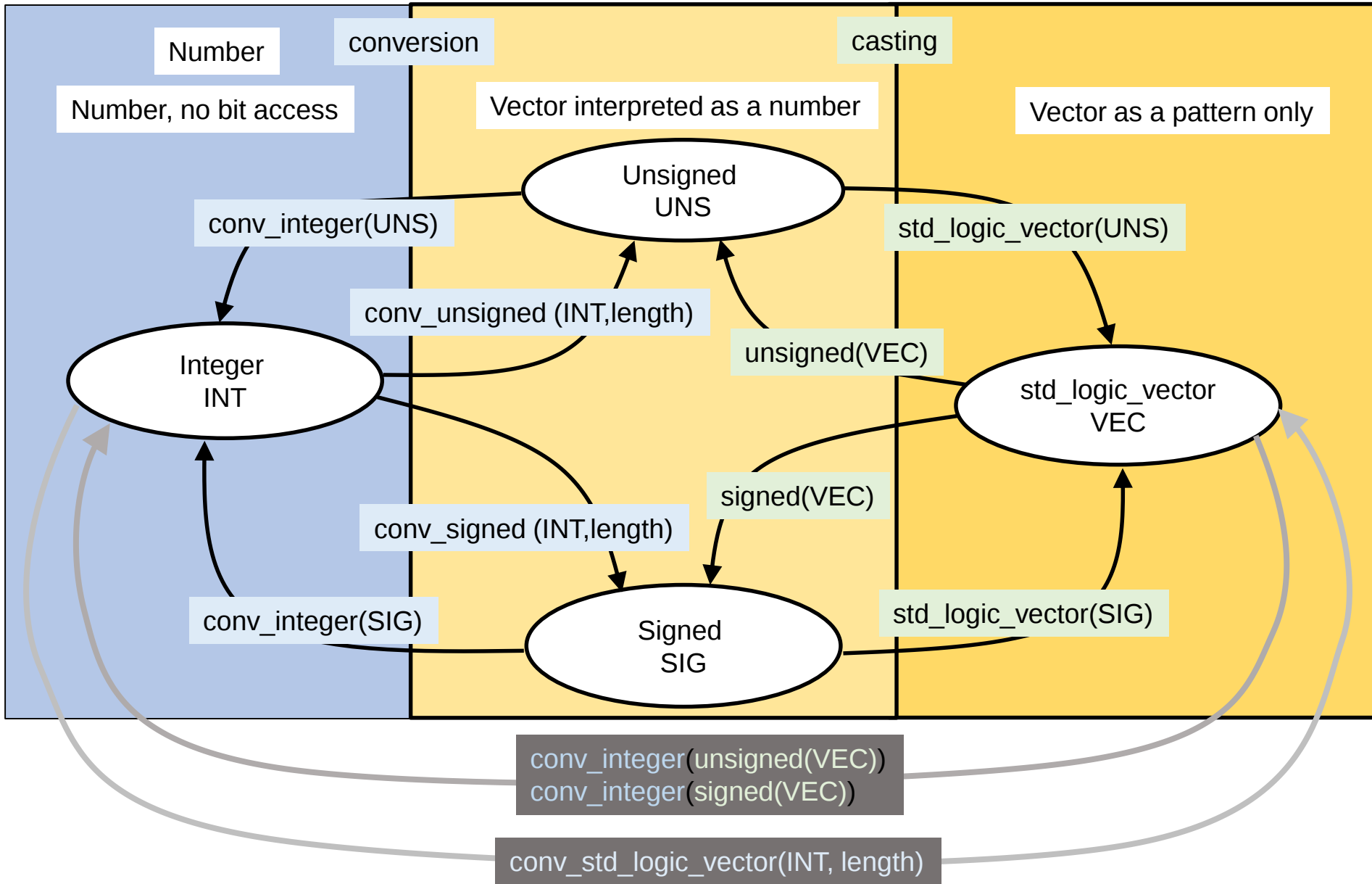
numeric_std vs. std_logic_arith

	numeric_std	std_logic_arith
Type Conversions		
std_logic_vector -> unsigned	unsigned(arg)	unsigned(arg)
std_logic_vector -> signed	signed(arg)	signed(arg)
unsigned/signed-> std_logic_vector	std_logic_vector(arg)	std_logic_vector(arg)
integer -> unsigned	to_unsigned(arg,size)	conv_unsigned(arg,size)
integer -> signed	to_signed(arg,size)	conv_signed(arg,size)
unsigned/signed -> integer	to_integer(arg)	conv_integer(arg)
integer -> std_logic_vector	std_logic_vector(to_unsigned(arg),size)	conv_std_logic_vector(arg,size)
unsigned + unsigned -> std_logic_vector	std_logic_vector (arg1 + arg2)	arg1 + arg2
signed + signed -> std_logic_vector	std_logic_vector (arg1 +arg2)	arg1 + arg2
Resize		
unsigned	resize(arg,size)	conv_unsigned(arg,size)
signed	resize(arg,size)	conv_signed(arg,size)

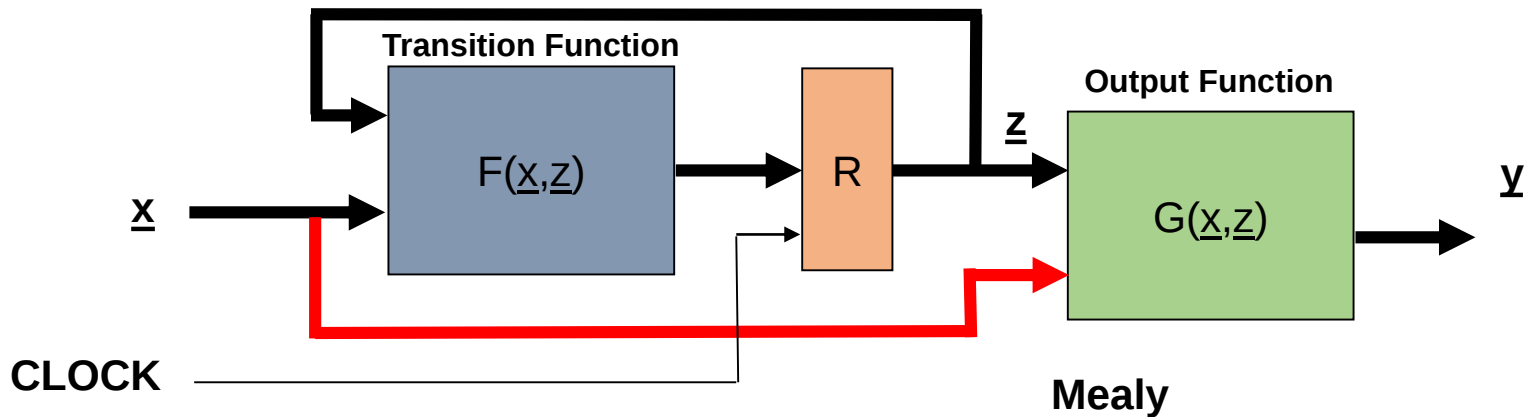
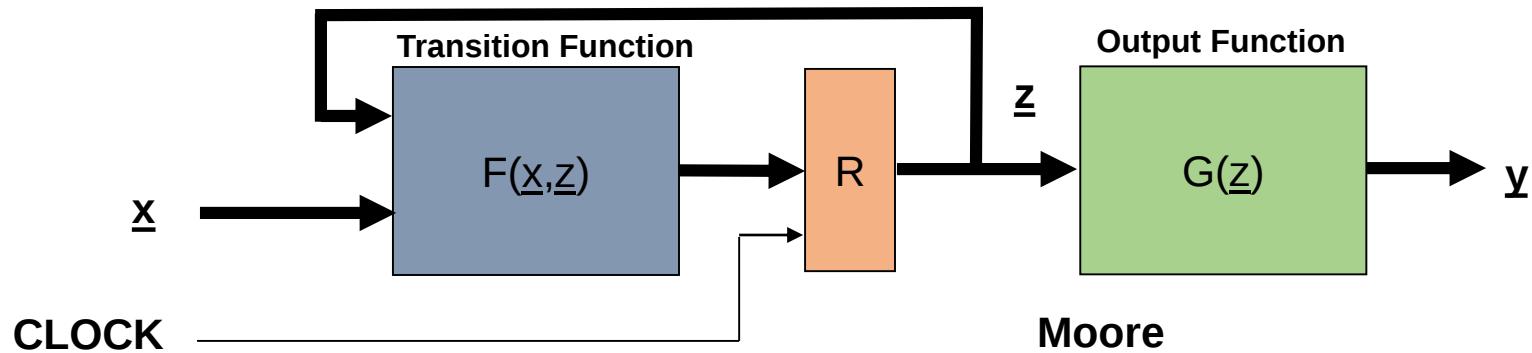
VHDL Type Conversion and Casting (**numeric_std** notation)



VHDL Type Conversion and Casting (`std_logic_arith` notation)

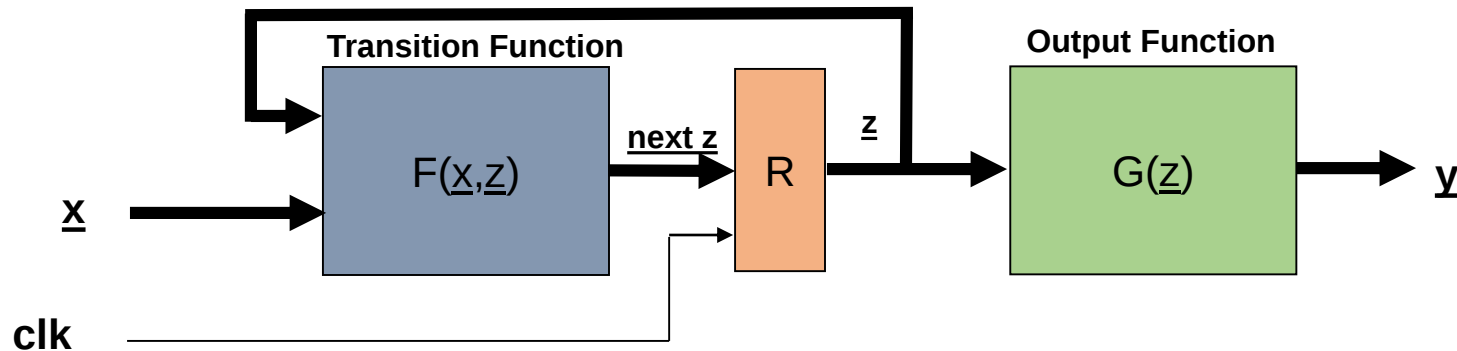


State Machines: Moore and Mealy



State Machines: Moore 3 Processes

$\underline{z}$ - current state
 $\underline{\text{next_}z}$ - next state



1st process (combinatorial)

```
process (z,x)
begin
  case z is
    when S1 => if x = x1 then next_z <= S2;
               end if;
    ...
  end case;
end process
```

2nd process (clocked)

```
process (clk)
begin
  if rising_edge(z) then
    z <= next_z;
  end if;
end process
```

3rd process (combinatorial)

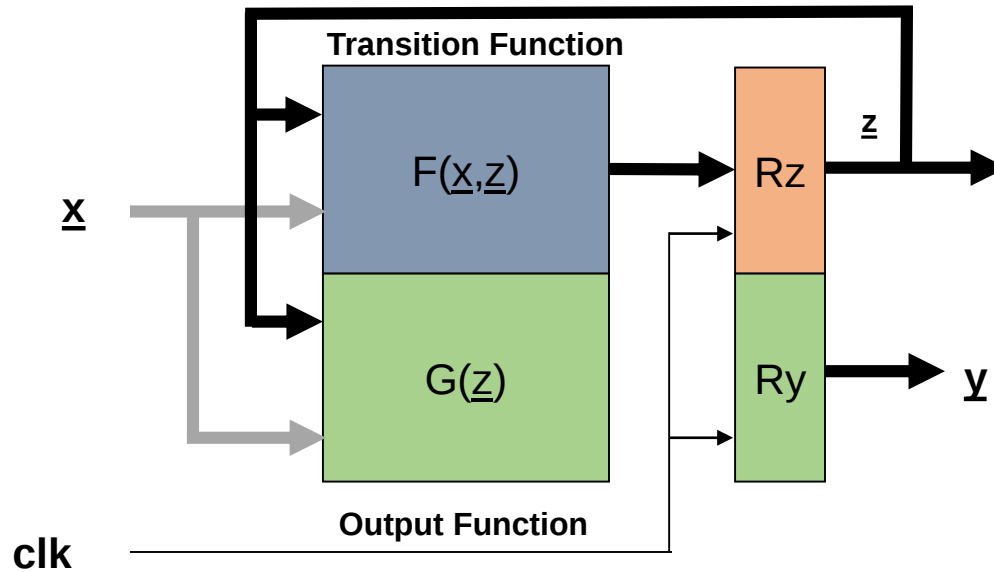
```
y <= "010" when z=S1 else
     "100" when z=S2 else
     "001";
```

Mealy includes input $\underline{x}$ here

Inside declarational part of architecture:

```
type z_type is (S1,S2,S3);
signal z, next_z : z_type;
```

State Machines: Moore Single Process



Single process (clocked)

```
process (clk)
begin
  if rising_edge(clk) then
    case z is
      when S1 => if x = x1 then z <= S2;
                                y <= "010";
                        end if;
    ...
    end case;
  end if;
end process;
```

y is registered

State Machine: Implicit

Process runs from WAIT to WAIT and restarts from the top (like a loop)
There is no explicit declaration of a state signal

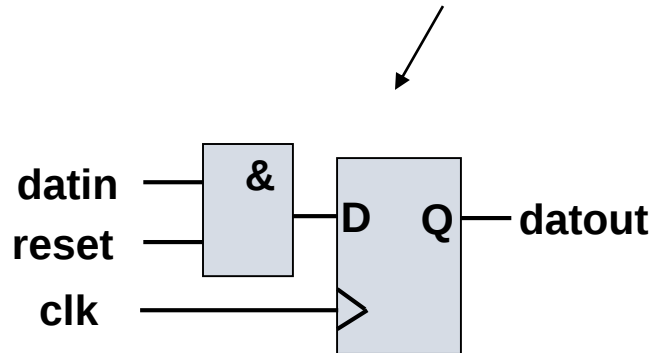
```
process
begin
    y <= "000";
wait until rising_edge(clk);
    y <= "101";
wait until rising_edge(clk) ;
    if x = x1 then    y <= "100";
    elsif x = x2 then y <= "010";
    end if;
wait until rising_edge(clk);
    y <= "100";
wait until rising_edge(clk)
...
end process;
```

Do **not** use a sensitivity list for processes containing WAIT-statements!

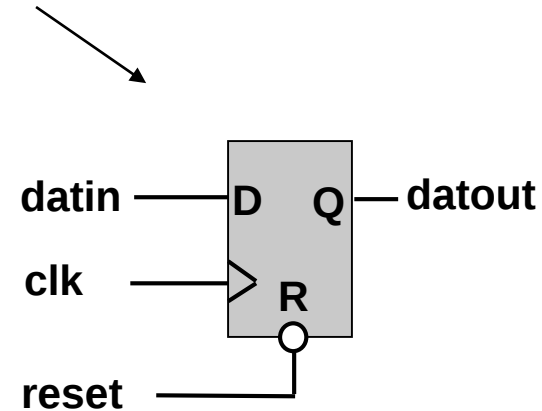
Synthesis?

Not all synthesis programs may support implicit state machine descriptions

Synchronous and Asynchronous Reset

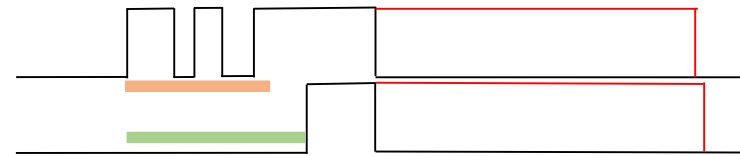
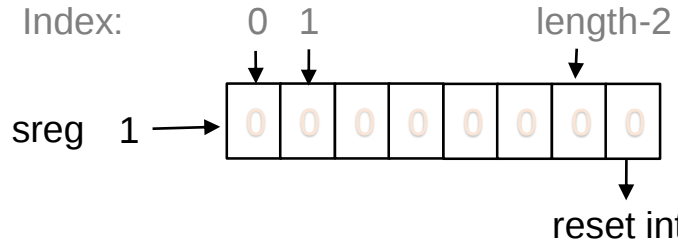


```
sr: PROCESS
  WAIT UNTIL (clk'event and clk='1');
  IF reset = '0' THEN datout <= '0';
  ELSE datout <= datin;
  END IF;
END PROCESS sr;
```

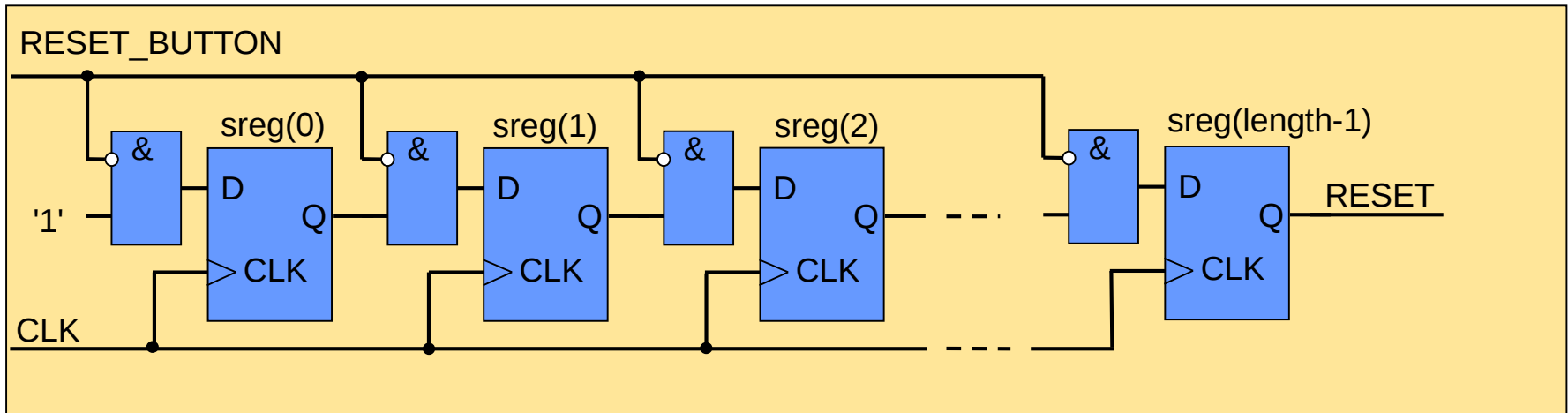


```
ar: PROCESS (clk , reset)
  IF reset = '0' THEN datout <= '0';
  ELSIF (clk'event and clk='1') THEN
    datout <= datin;
  END IF;
END PROCESS ar;
```

Example: Debouncing



$\text{clock_period} \times \text{reset_length} > \text{bouncing time}$



```

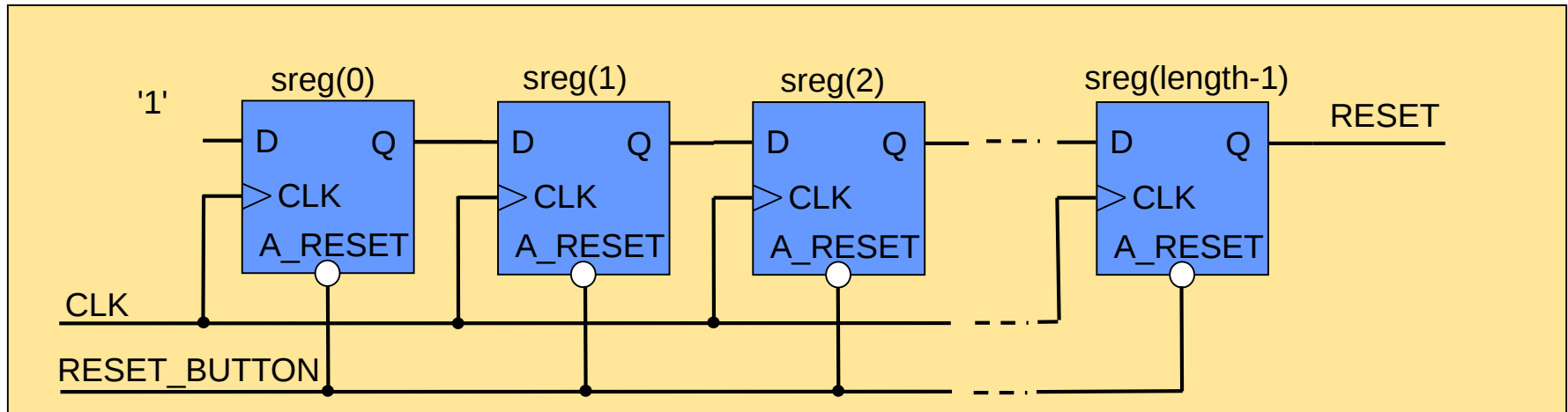
rstgen:  Process (CLK)
begin
  if rising_edge(CLK) then
    if RESET_BUTTON = '1' then sreg <= (others => '0');
    else sreg <= '1' & sreg(0 to sreg'length-2);
    end if;
  end if;
end process rstgen;

```

Only pulses at sample moment
initiate and prolong RESET output

RESET <= not sreg(sreg'length-1);

Example: Debouncing



```
rstgen:    Process (CLK, RESET_BUTTON )
begin
  if RESET_BUTTON = '1' then sreg <= (others => '0');
  elsif rising_edge(CLK) then else sreg <= '1' & sreg(0 to sreg'length-2);
  end if;
end process rstgen;
```

```
RESET <= not sreg(sreg'length-1);
```

Very short pulses initiate
and prolong RESET output

Synthesis: Memory Insertion (Latch, Register)

In all processes (including clocked processes):

Be sure that all variables get a value assigned whenever the process is executed!

Otherwise in clocked processes registers(flip-flops), in combinatorial processes latches for the variables inserted.

In combinatorial processes:

Assign a value to any output of the process in all branches (if, case) when the process is executed (see sensitivity list)!

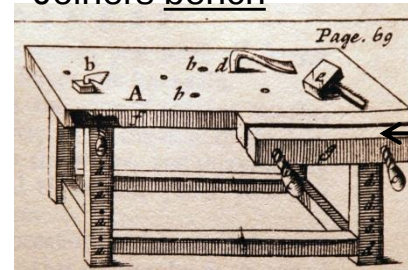
```
if inp = '1' then outp <= '0';  
end if;
```

Says: When inp = '0' hold the old value → Latch insertion

Look for warnings from the synthesis program indicating latch insertion and check your code! If your intention was to make a synchronous design with edge triggered flip-flops there should not occur any Latch insertion.

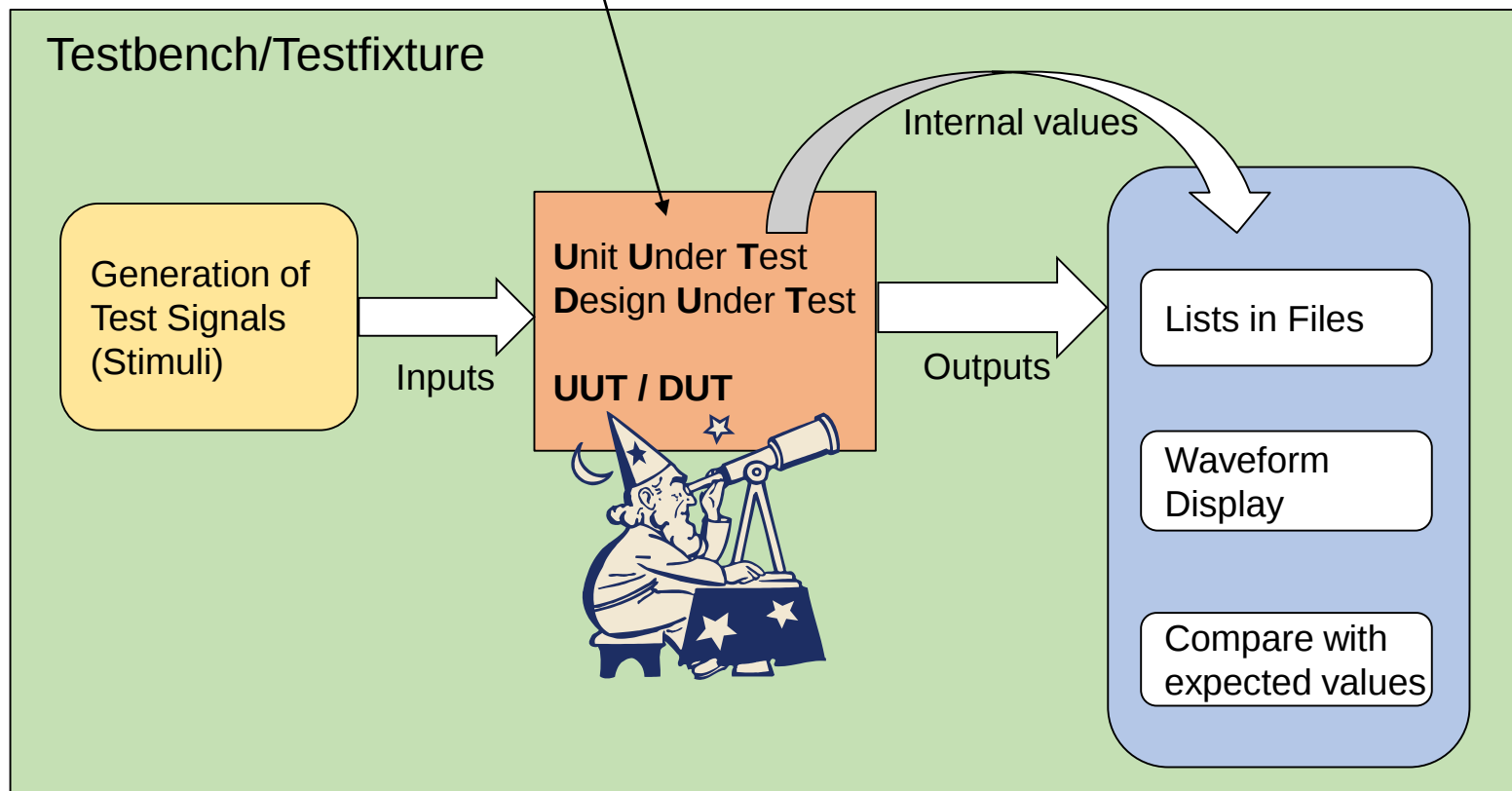
Simulation: Testbench for Logic Simulation

Joiners bench

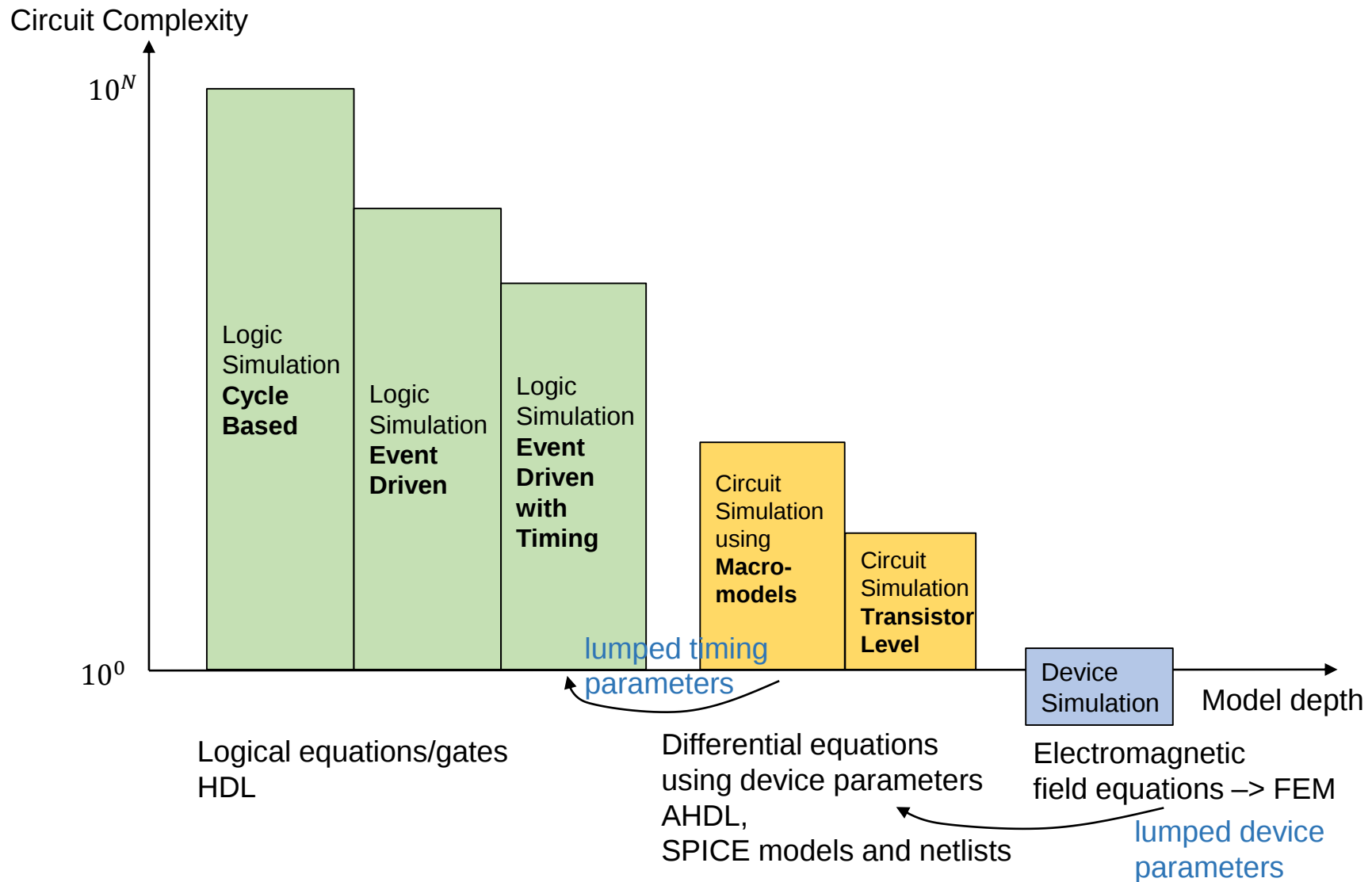


Fixture (vise_{am.}/vice_{br.})

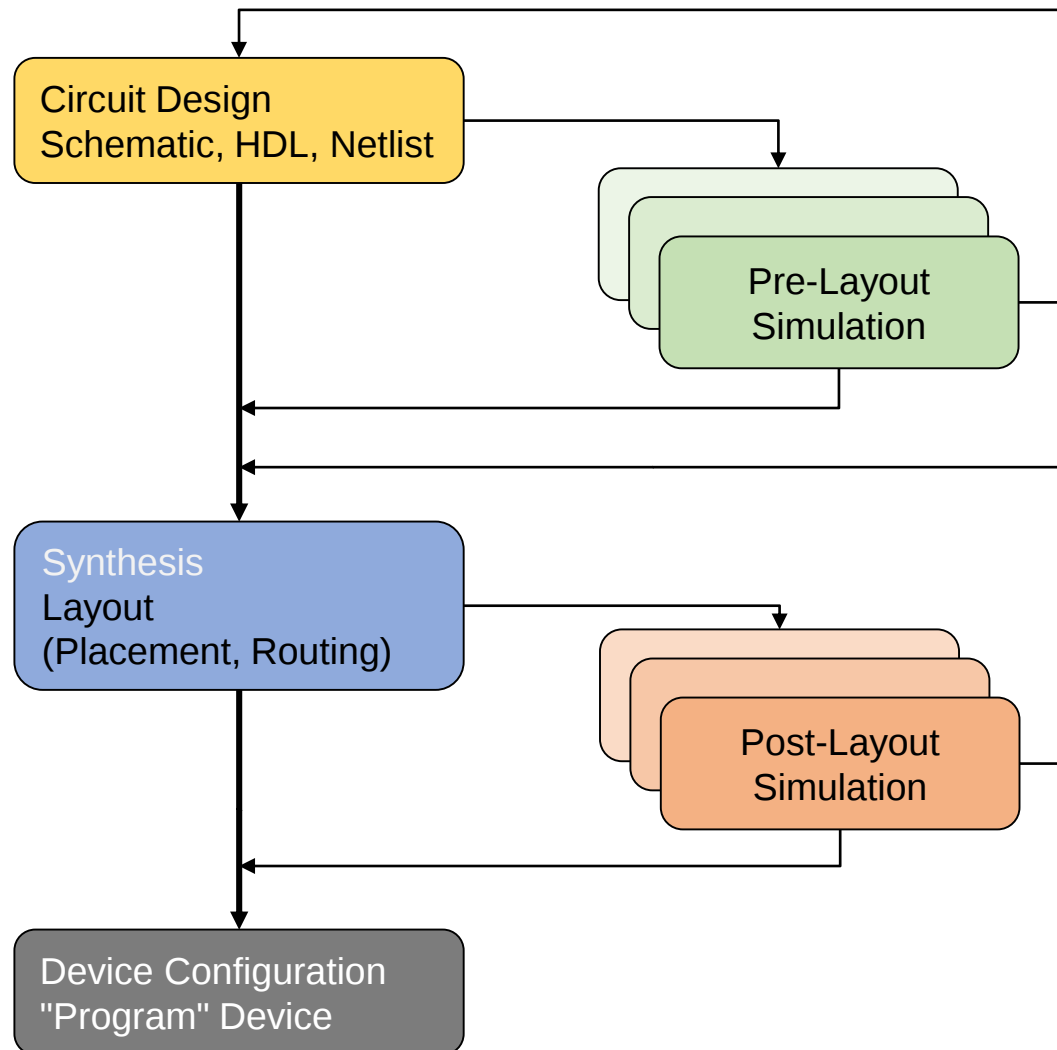
Netlist, HDL, SDF for timing



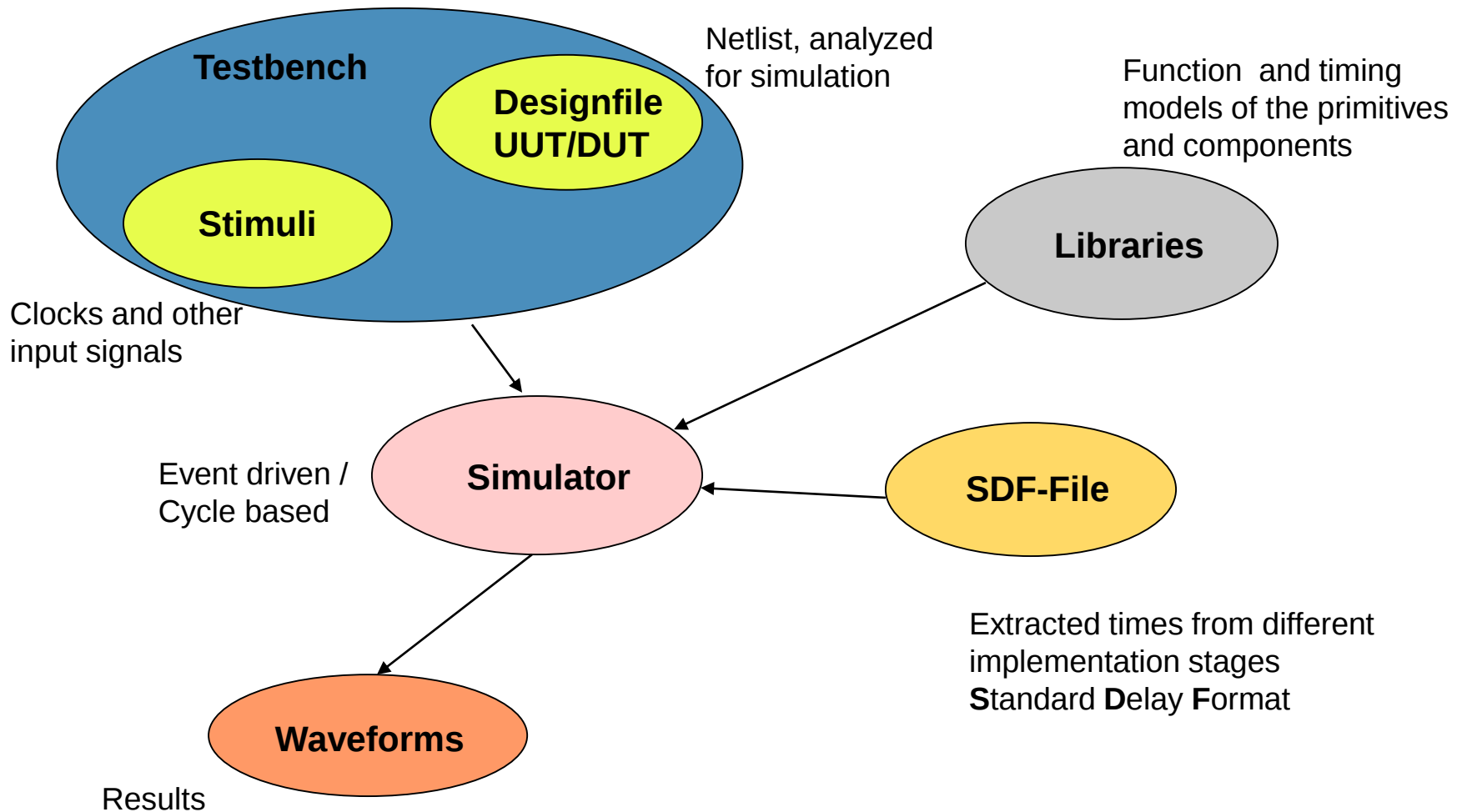
Simulation: Depth of Simulation Models and Circuit Complexity



Simulation as Part of the Design Process



Simulation



Simulation Libraries (Example: Xilinx)

Functional Simulation (VHDL and Gate Level Netlist)

```
Library UNISIM;  
use UNISIM.all  
use UNISIM.VCOMPONENTS.ALL
```

```
Library XilinxCoreLib;
```

Timing Simulation

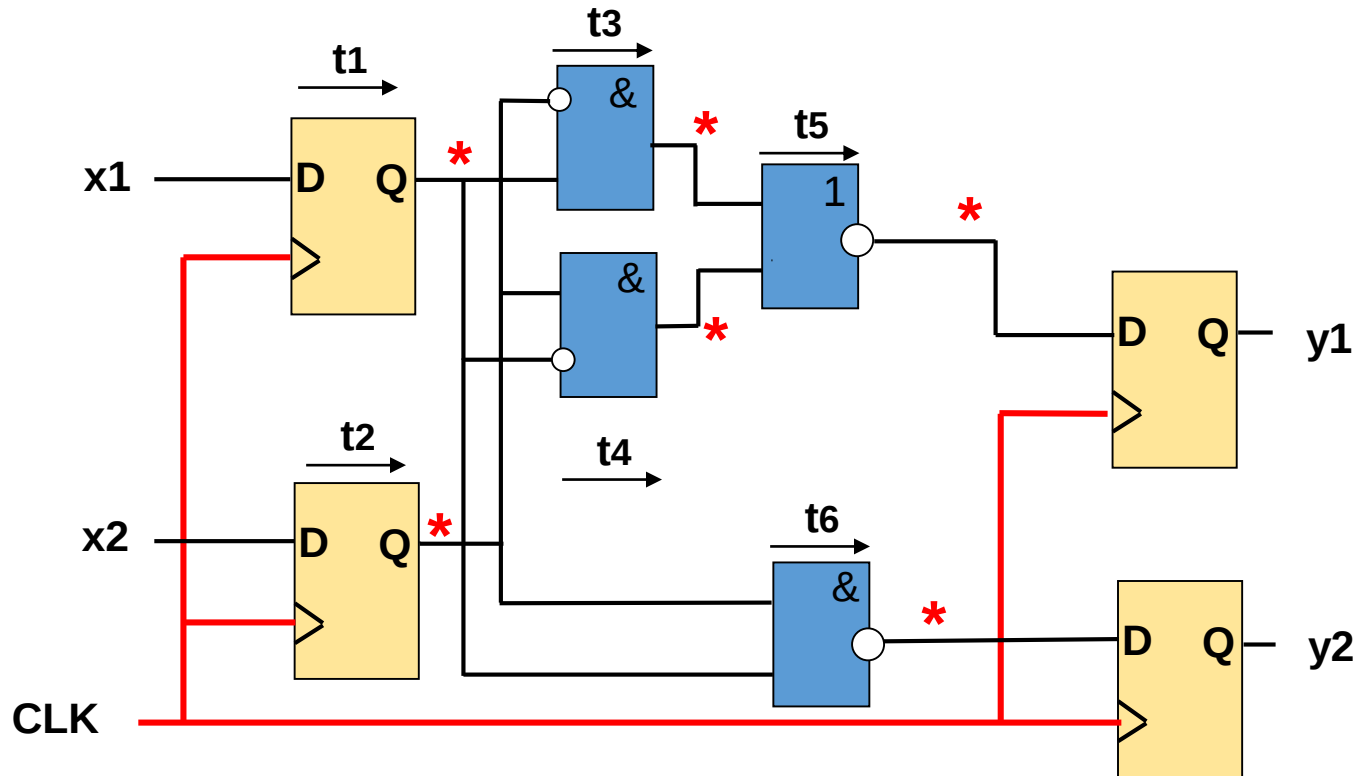
VITAL : VHDL Initiative Towards ASIC-Libraries (IEEE 1076.4)

```
Library SIMPRIM;  
use SIMPRIM.VCOMPONENTS.ALL;  
use SIMPRIM.VPACKAGE.ALL;
```

Cores are implemented as gates ...

!! Libraries have to be compiled by the actual version of imulator for the actual technology !! (Example: Modelsim 6.0a -> XILINX 7.1)
Setup files must point to the appropriate directory !

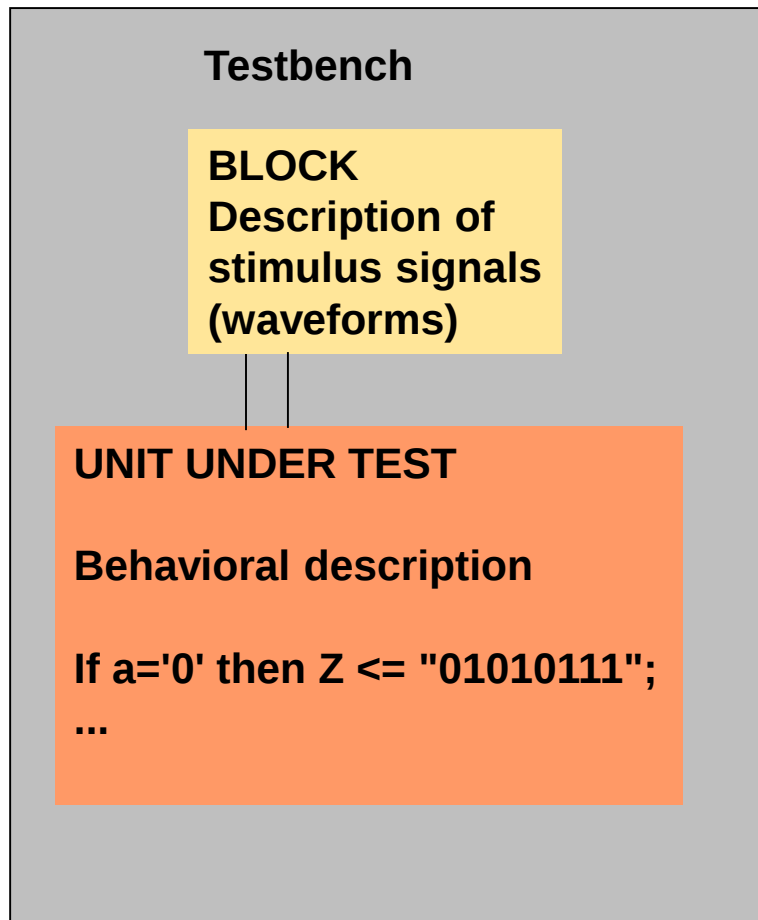
Event Driven and Cycle Based Simulation



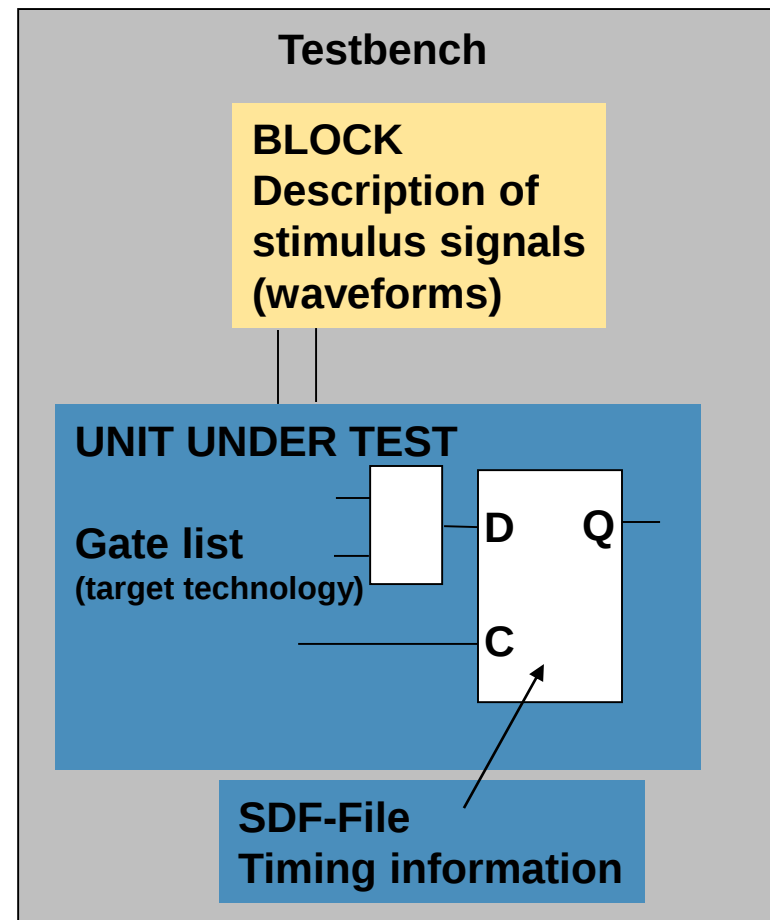
Event Driven: Whenever a delay time runs out a new simulation cycle is started
We can see what and when something happens at the * points.

Cycle Based: Only once per cycle the result is calculated

Testbench and Backannotation

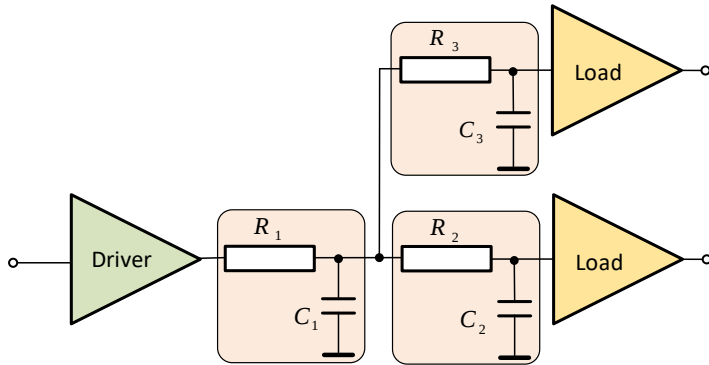


Functional

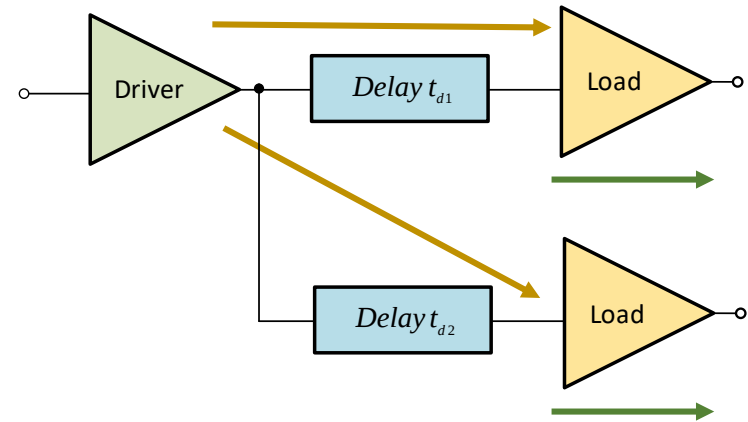


Timing

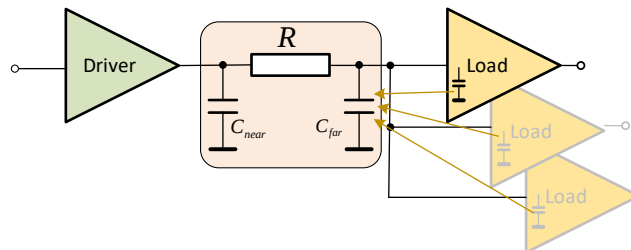
Delay Calculation



Detailed Parasitic Model -> DSPF (Detailed Standard Parasitic Format)



Lumped Model (extracted delay times)



Reduced Parasitic Model / Pi-Model -> RSPF Reduced Standard Parasitic Format

To discuss: RLC-Models

Backannotation (Timing)

SDF-Standard IEEE 1497

VHDL-File

SDF-File

```
(CELL (CELLTYPE "X_FF")
(INSTANCE redcnt_reg_3_Q)
(DELAY
  (ABSOLUTE
    (PORT CE (1881:1881:1881)(1881:1881:1881))
    (PORT CLK (1333:1333:1333)(1333:1333:1333))
    (PORT RST (14400:14400:14400)(14400:14400:14400))
    (IOPATH CLK O (2820:2820:2820)(2820:2820:2820))
    (IOPATH SET O (2820:2820:2820)(2820:2820:2820))
    (IOPATH RST O (5000:5000:5000)(5000:5000:5000))
  ) )
(TIMINGCHECK
  (HOLD (posedge I) (posedge CLK) (0:0:0))
  (HOLD (negedge I) (posedge CLK) (0:0:0))
  ....
  (SETUP (posedge I) (posedge CLK) (1050:1050:1050))
  (SETUP (negedge CE) (posedge CLK) (3000:3000:3000))
  (WIDTH (posedge RST) (4000:4000:4000))
) )
```

min, typ, max in ps *

01-edge 10-edge

```
redcnt_reg_3_Q : X_FF
port map (
  I => n1600,
  CE => n345(8),
  CLK => n1206,
  SET => GND,
  RST => GSR,
  O => redcnt(3) );
```

Internal Delays

Wire Delays
allocated at inputs

Check values

*Sometimes there is only "worst case delay" available, filling all delays (min, typ, max) -> XILINX

Configurations for pre- and post-layout simulation

Configuration for the functional simulation using **behavioral VHDL-model**:

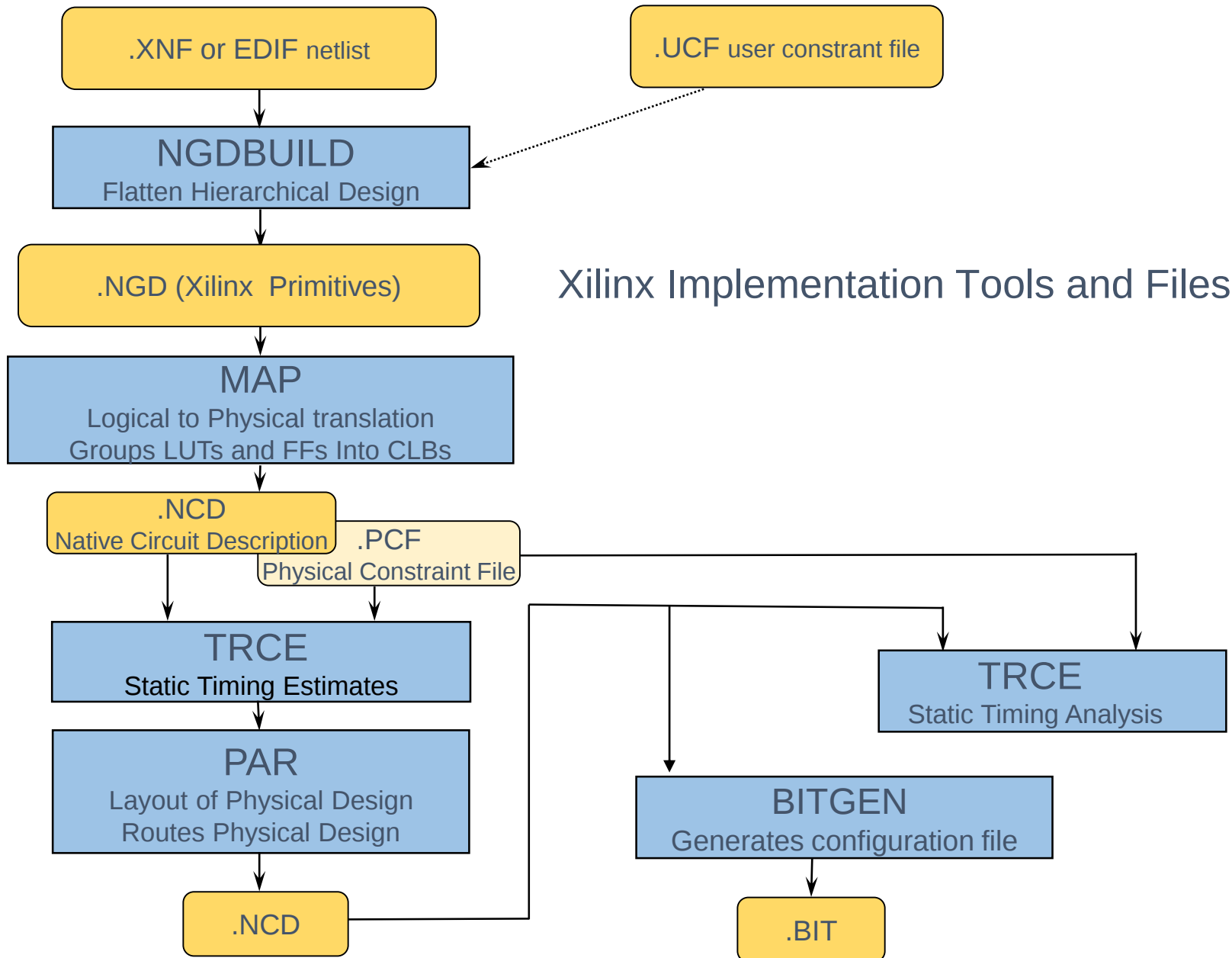
```
CONFIGURATION cfg_design1_tb_behavior OF e IS
  FOR testbench
    FOR UUT: design1 USE entity WORK.design1(behavior);
    END FOR;
  END FOR;
END cfg_design1_tb_behavior;
```

Designers architecture name

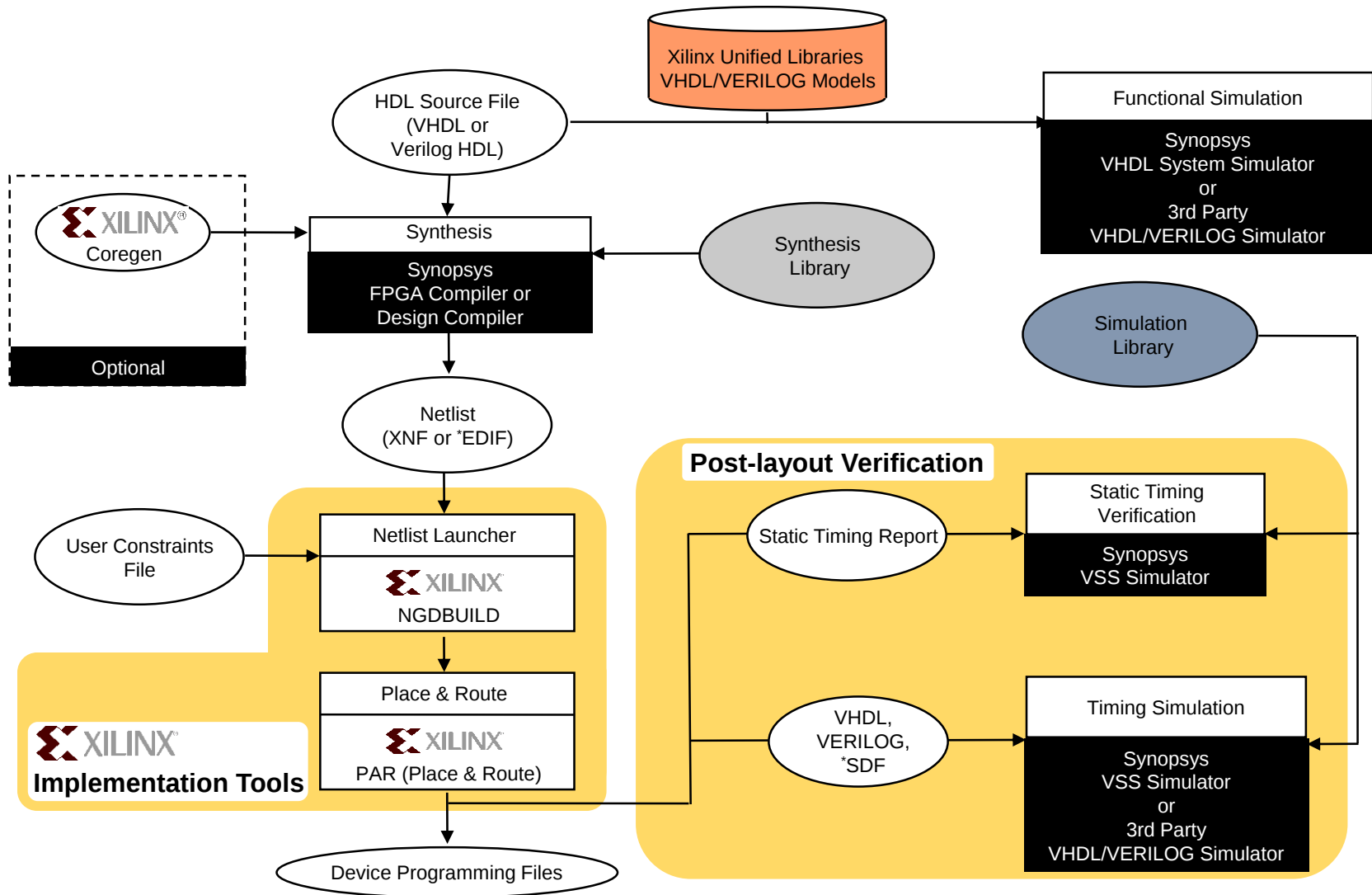
Configuration for the timing simulation using the placed and routed structure:

```
CONFIGURATION cfg_design1_tb_structure OF e IS
  FOR testbench
    FOR UUT: design1 USE entity WORK.design1(structure);
    END FOR;
  END FOR;
END cfg_design1_tb_structure;
```

Design Tools architecture name
for post-layout VHDL description



Synopsys FPGA Design Flow



Standard Cells vs. FPGA

Standard Cells

Floorplanning

- Logic rows, modules

- Padring

- Power rings and stripes

Placement

- Generation of clock nets (trees)

- Routing clock, logic

- Files for mask layout

FPGA

- All cell locations are ready

Mapping

- Power nets are ready

- Clock nets are ready

- Placement is mapping of user logic to the cells on the chip

- Routing with given nets and programmable interconnection points

- Files for programming

Files for Synthesis

Design Compiler
Setup File

VHDL Description

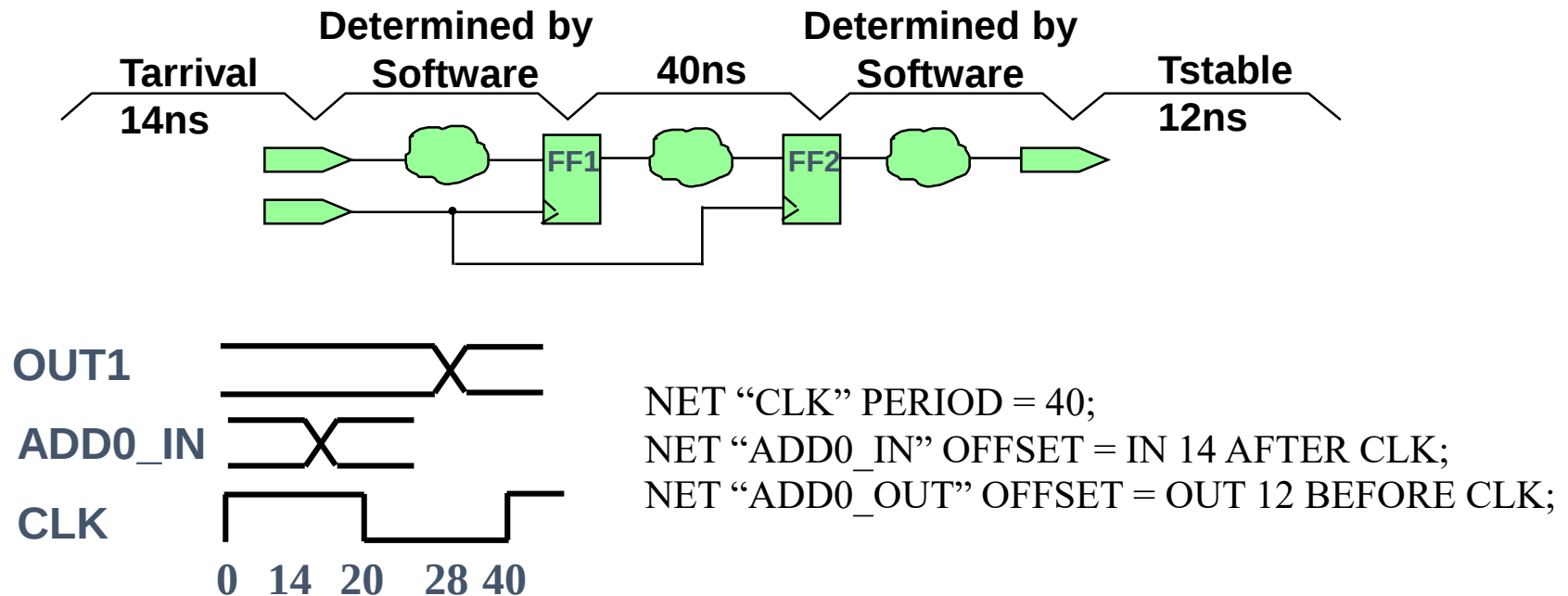
Design Constraints

Design Compiler
Script

Synchronous Constraint Example

(From Xilinx: "Advanced Software Design with Xilinx M1-Based Software")

OFFSET defines the delay of a signal external to the chip, relative to a clock. Internal clock delays are determined by Software



Practical Exercise: Controlling a VGA-Monitor

Task:

One part of a VGA-circuit is the creation of signals for synchronization of the monitor and the DAC (digital to analog converter). Depending of the resolution and the picture rate of the monitor it has to create a distinct pattern for the signals:

HSYNC - horizontal synchronization signal (row)

VSYNC - vertical synchronization signal (picture) und

BLANK - blanking of the beam (horizontal and vertical), signal to the DAC

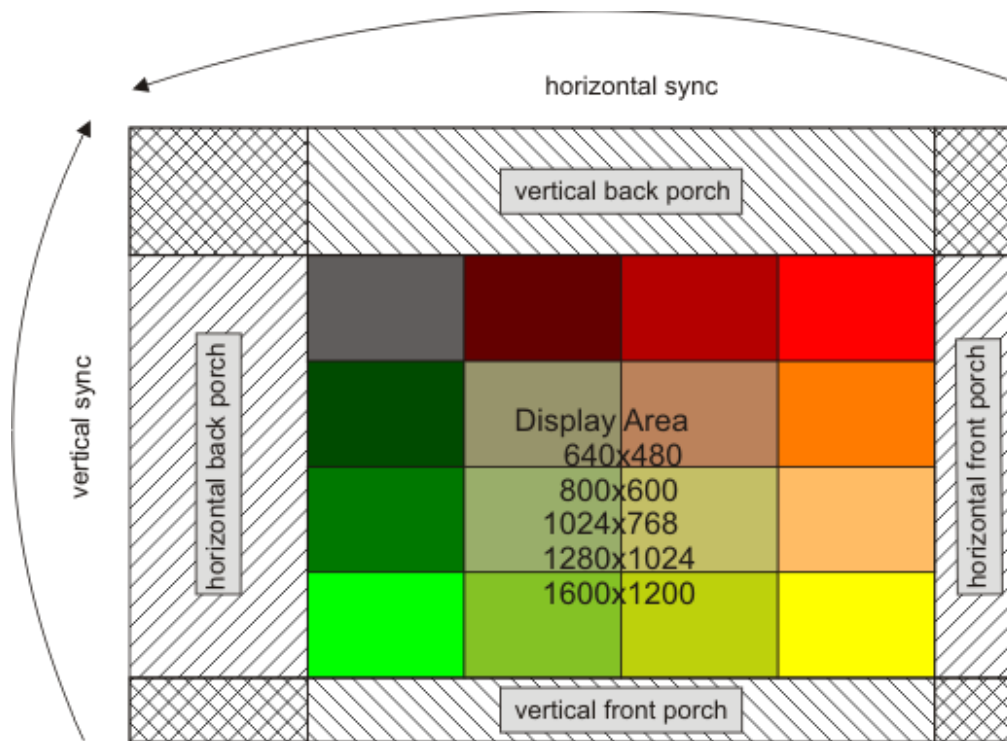
This part of a VGA controller will be designed and tested in this exercise.

All signals are derived from the pixel clock fp:

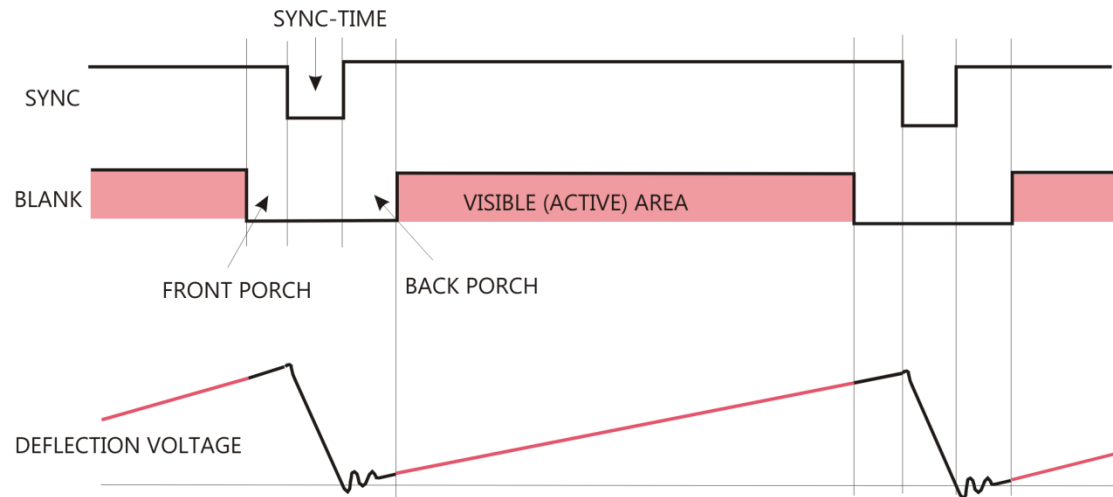
$$fp = \text{picture rate} * \text{number of rows} * \text{number of pixels per row}$$

Practical Training: Controlling a VGA-Monitor

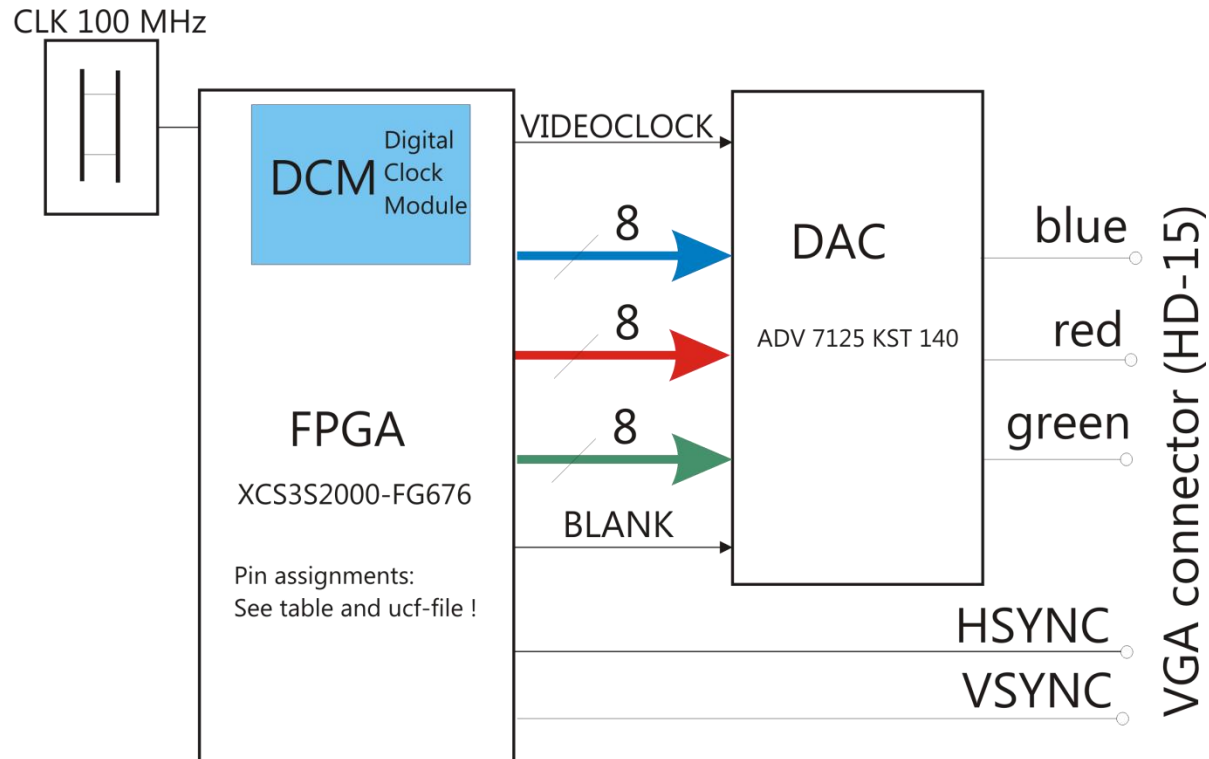
The numbers of rows and pixels per row include the time (expressed in numbers of rows/pixels) for blanking outside the border of the visible picture and for the horizontal and vertical retrace of the beam.



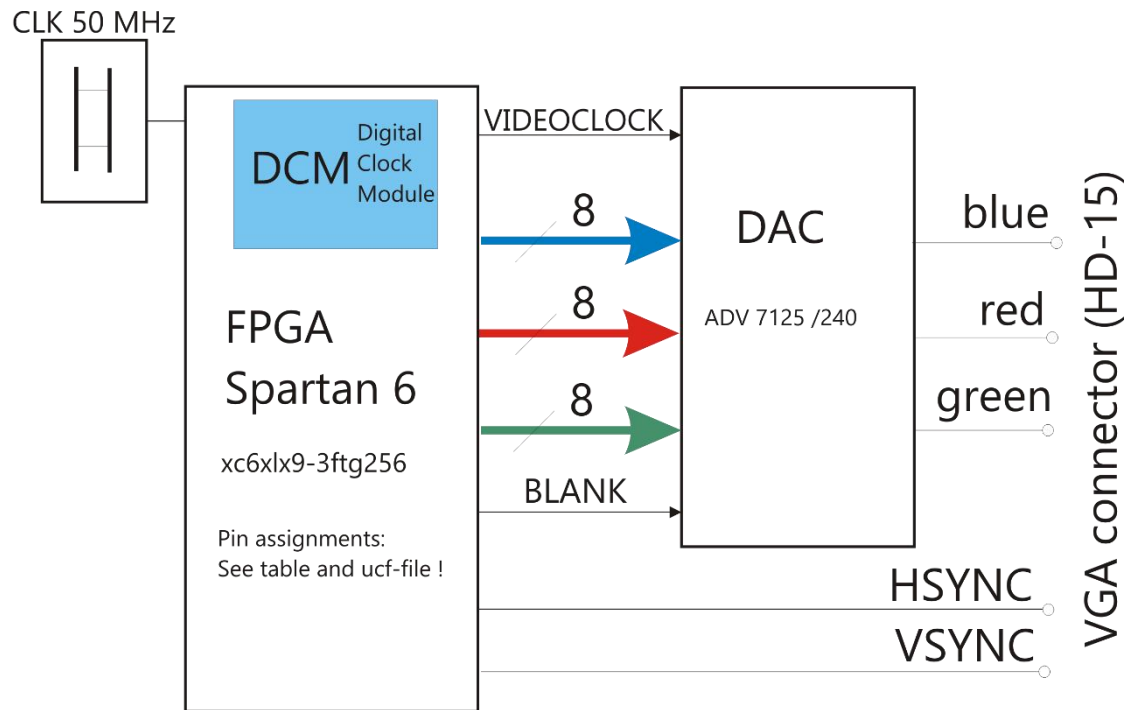
Resolution Refresh	Clock M/D for DCM	H-Active	H-Front-Porch	H-SYNC	H-Back-Porch	V-Active	V-Front-Porch	V-SYNC	V-Back-Porch
640x480 @60Hz	25.175 2/8	640	16	96 (low)	48	480	10	2 (low)	33
640x480 @75Hz	31.5 5/16	640	16	64 (low)	120	480	1	3 (low)	16
800x600 @72Hz	50 2/4	800	56	120 (high)	64	600	37	6 (high)	23
1024x768 @60Hz	65 13/20	1024	24	136 (low)	80	768	3	6 (low)	29
1024x768 @75Hz	78,75 11/14	1024	16	96 (high)	176	768	1	3 (high)	28
1280x1024 @60Hz	108 27/25	1280	48	112 (low)	248	1024	1	3 (low)	38



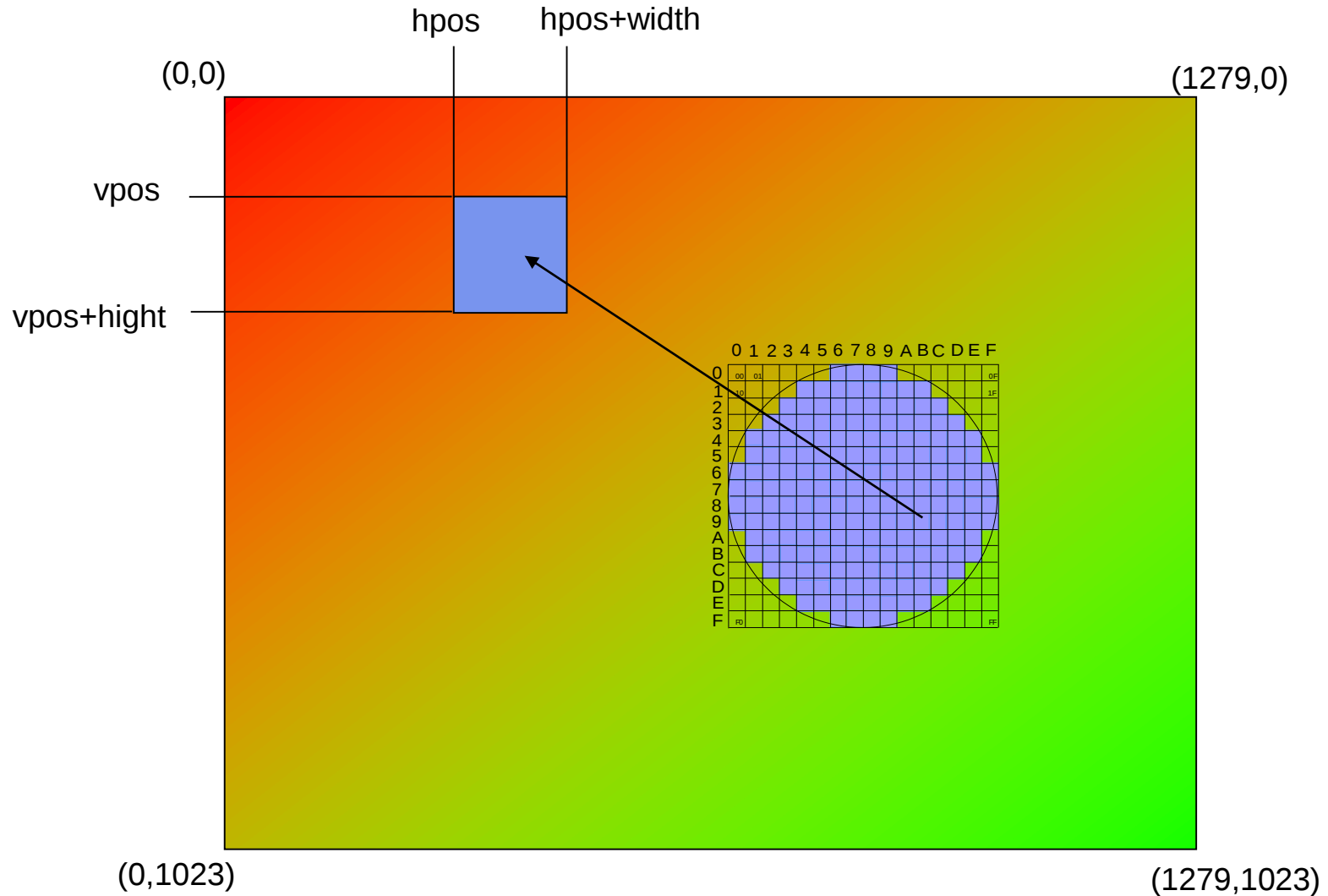
VGA-part of demonstration board with XCS3S2000-FG676:



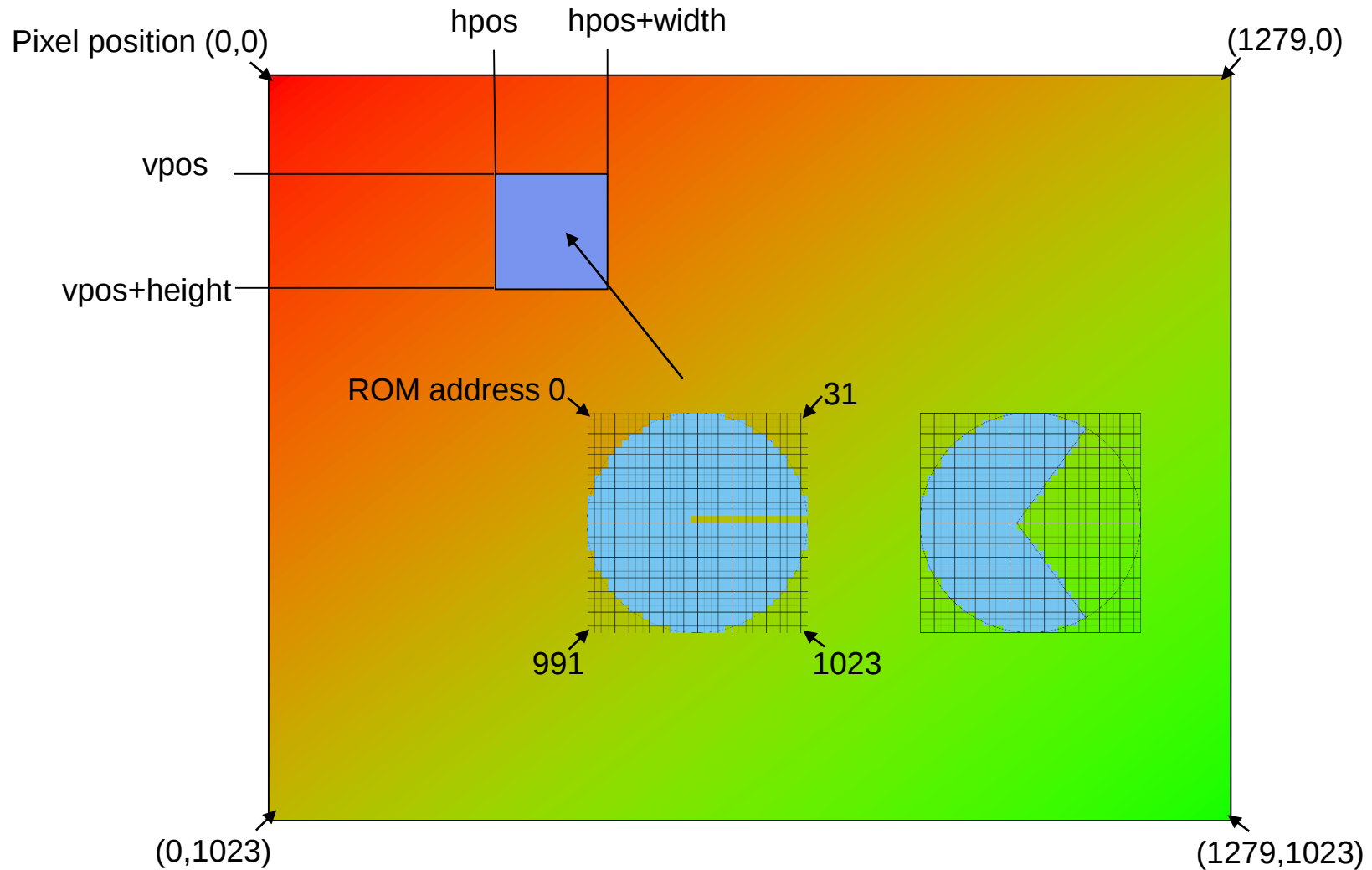
VGA-part of demonstration board with XCS3S2000-FG676:



Displaying objects on display 1



Displaying objects on display 1



Displaying objects on display 2

Object:

If **hpos** $\leq$ **horizontal_counter** $<$ (**hpos** + **width**) and **vpos** $\leq$ **vertical_counter** $<$ (**vpos** + **height**) then replace background color by foreground.

Object with pattern:

If **hpos** $\leq$ **horizontal_counter** $<$ (**hpos** + **width**) and **vpos** $\leq$ **vertical_counter** $<$ (**vpos** + **height**) then look into the memory and increase the memory address by 1.

If memory content is a '1' then replace background color by foreground.

Memory address is set to 0 when a new picture starts.

Memory size is **width*height**. Memory width is 1 bit.

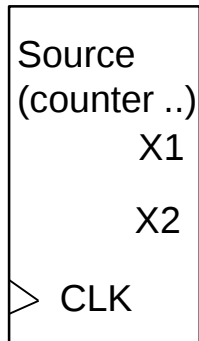
Moving object with pattern:

Change hpos and/or vpos values with every new picture displayed.

Object with variable pattern:

Use memory with n bit width. Use every bit in the word for another pattern and display depending upon actual position.

Avoiding (supressing) Hazards

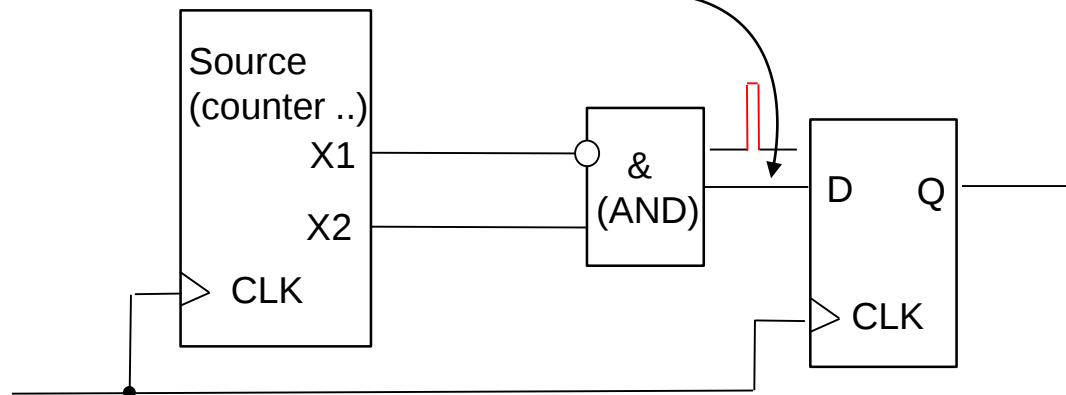


Theory and Case 1 -> no Output
Case 2 -> Glitch, One Hazard

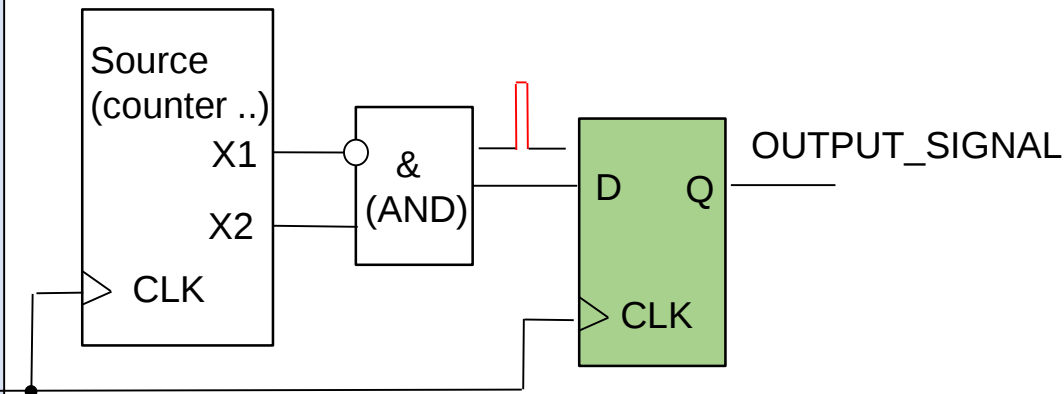
May be avoided by additional logic,
special coding

or D-FF

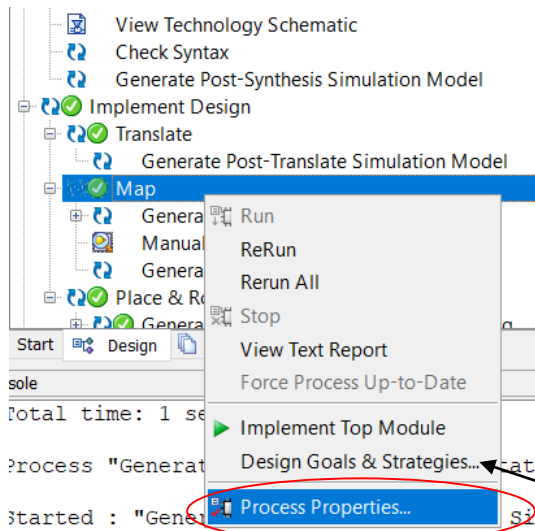
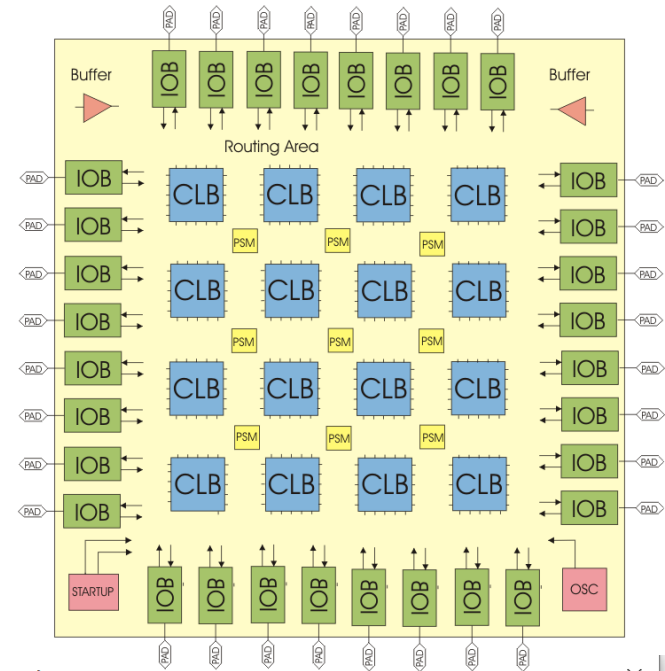
	Theory		Real Case 1		Real Case 1	
	X1	X2	X1	X2	X1	X2
Time ↓	0	0	0	0	0	0
	1	1	1	0	0	1
			1	1	1	1



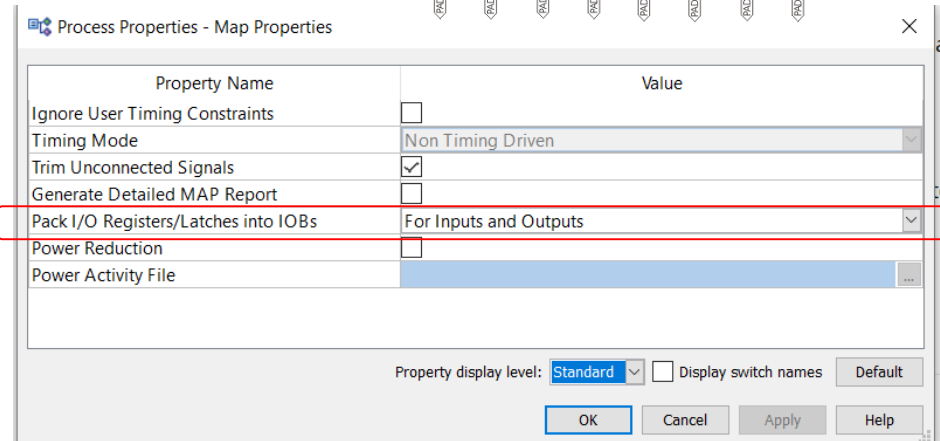
Synchronous Output Changes (and avoid hazards at outputs)



Use a FF in the Output Cell here1



Design Goals and Strategies must not be locked anywhere



Synchronous Output Changes (and avoid hazards at outputs) Example VGA control

Avoid direct output from logic functions (gates)

Use Flip-Flops in the IOBs of the FPGA as last memory

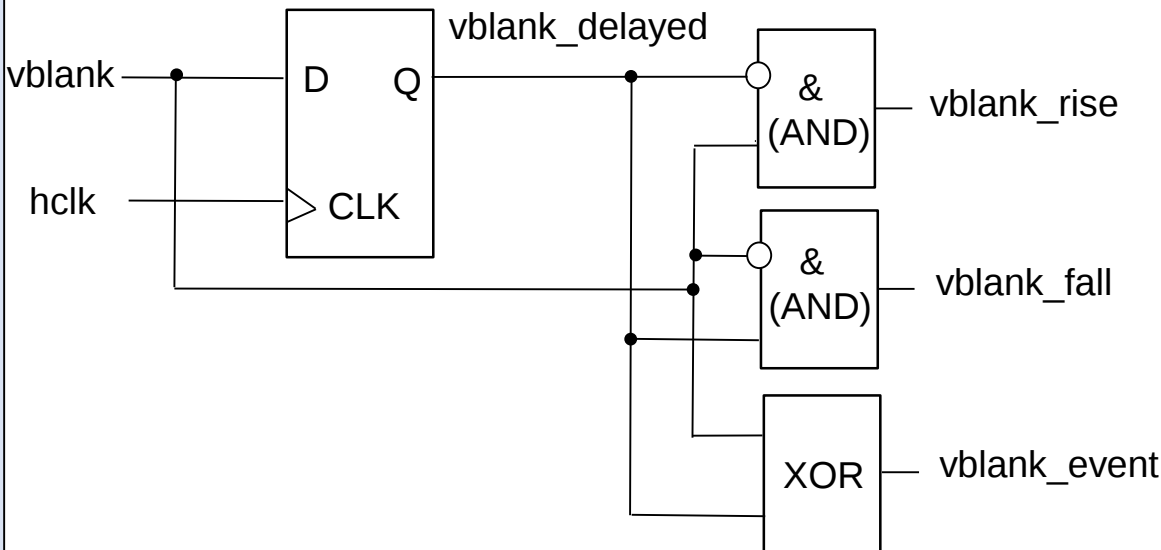
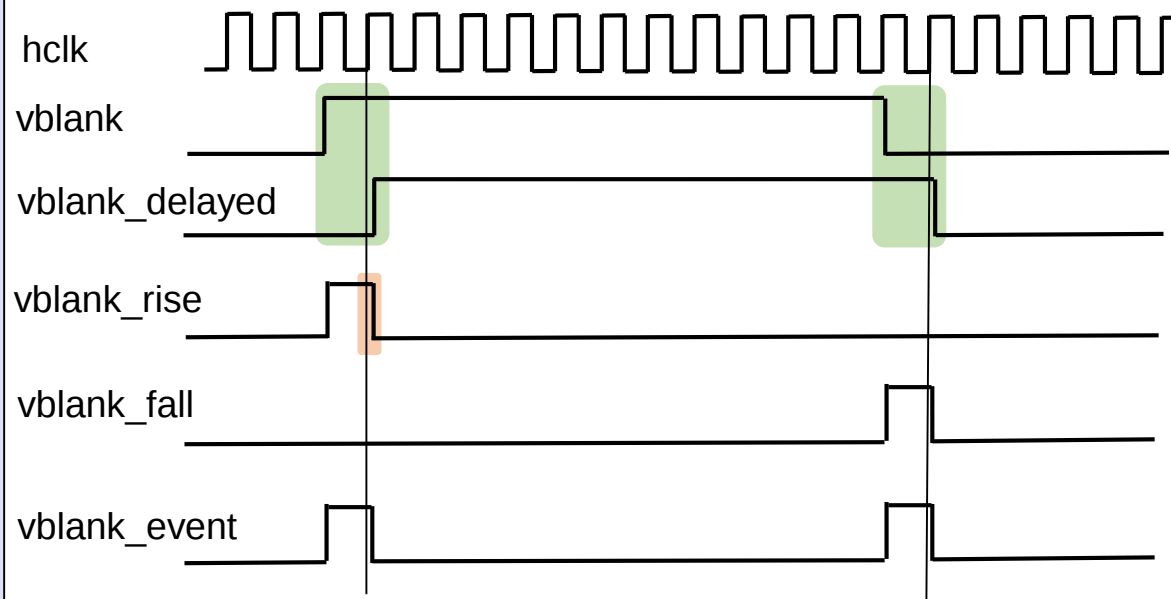
```
hsync_int <= some assignment; -- decoding counter values  
red_int <= another assignment; -- maybe multiplexer background, black, pacman  
.....
```

```
assign_outputs : process(hclk)  
  if rising_edge(hclk) then  
    HSYNC <= hsync_int;  
    VSYNC <= vsync_int;  
    BLANK <= blank_int;  
    RED <= red_int;  
    GREEN <= green_int;  
    BLUE <= blue_int;  
  end if;  
end process assign_outputs;
```

Possible to have expressions here:
BLANK <= hblank and vblank;

Phase relationship hclk to VIDEOCLK ? Expected: VIDEOCLK delayed like the other outputs

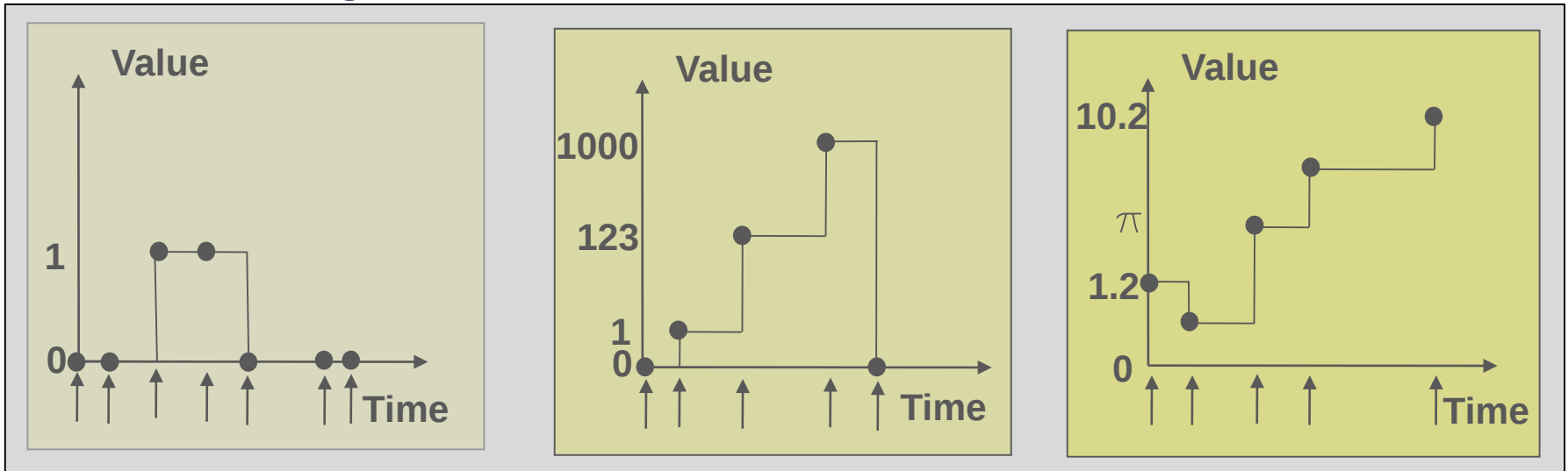
Derivative of a Signal - Building a Pulse of one Period



Position incremented once per picture
(every 1/60 second) :

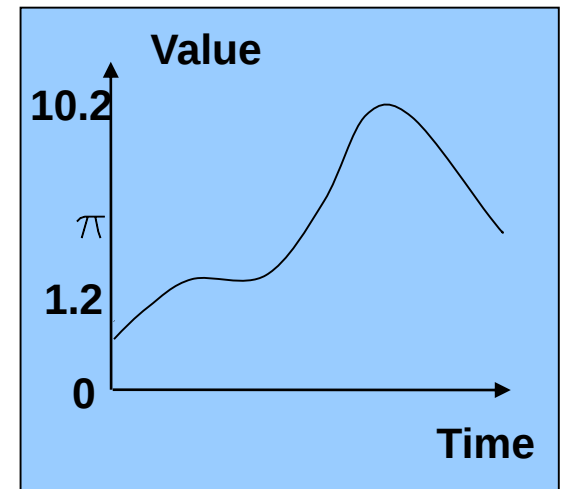
```
useit: process (hclk)
  if rising_edge(hclk) then
    if vblank_rise = '1' then
      position <= position + 1;
      ....
    end if;
  end if;
```

Analog Extensions to VHDL: VHDL-AMS

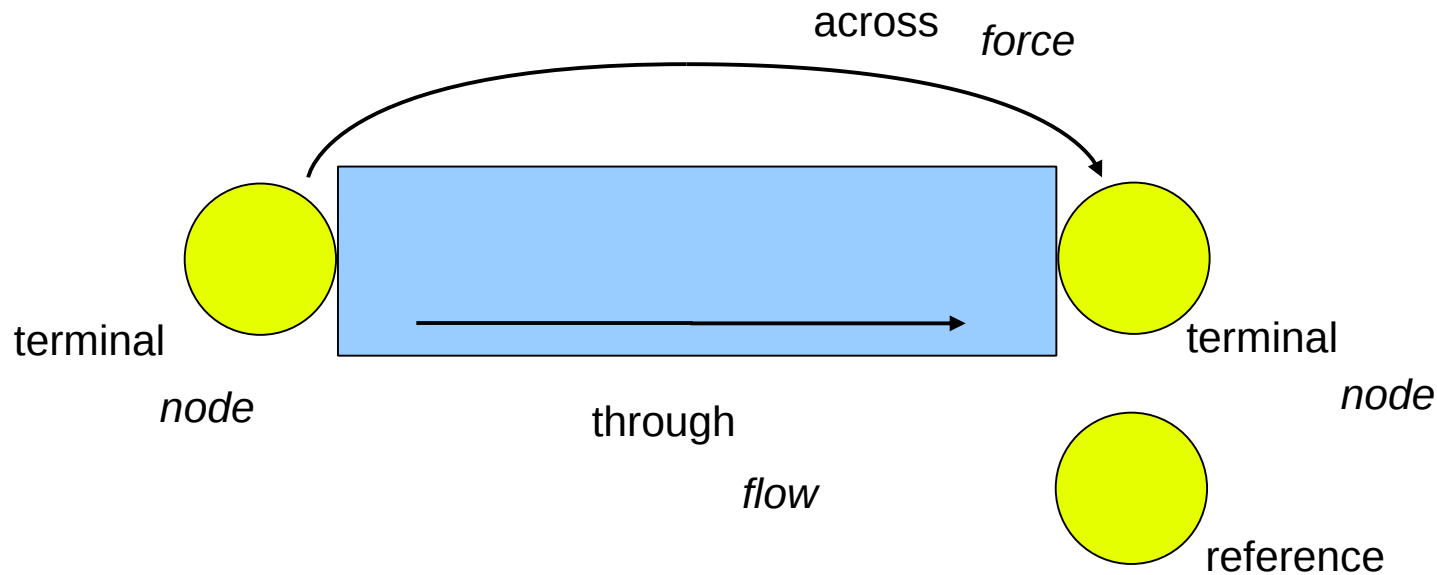


Event driven systems can be modeled with VHDL 1076

Continuous signals need a solver for Differential Algebraic Equations (DAE) -> VHDL-AMS (1076.1)



Energy Domains, Terminals, Natures



subtype voltage **is** real **tolerance** "default_voltage";

...

nature electrical **is**
voltage **across**
current **through**
electrical_ref **reference**;

nature thermal **is**
temperature **across**
heat_flow **through**
thermal_ref **reference**;

Standard Natures

External packages, not included in VHDL (like std_logic);

```
library ieee_proposed;  
use ieee_proposed.electrical_systems.all;
```

Proposed standard packages for modelling of various energy domains

nature	across type, through type	package
electrical	voltage, current	electrical_systems
magnetic	mmf (ampere turns), flux (weber)	
translational_v	velocity, force (dynamic systems)	mechanical_systems
translational	displacement, force (static systems)	
rotational_v	velocity, torque (Nm)	
rotational	angle, torque	
fluidic	pressure, volumeflow rate (m ³ /s)	fluidic_systems
thermal	temperature, heat flow rate (joule/s)	thermal_systems
radiant	illuminance (candela), optic flux (lumen)	radiant_systems

Quantity

New object : Quantity -> Unknown of differential algebraic equations

$$F(x, dx/dt, t)$$

Free Quantities: Signal flow modeling

Branch Quantities. Modeling conservative energy systems

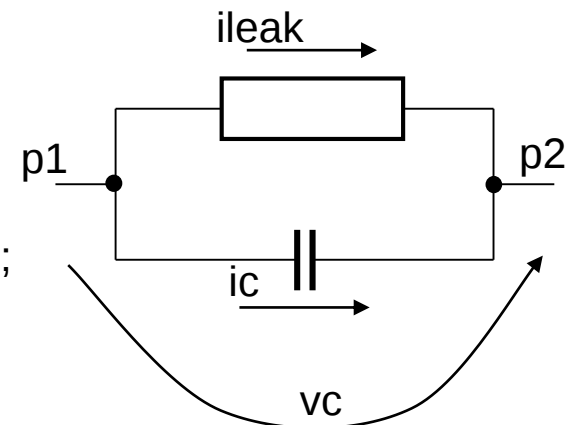
Source Quantities: Modeling sources and noise

Example: Capacitor with leakage

(one across, two through branches) p1

Terminal p1,p2: **electrical**;

Quantity v_C across i_C, i_{leak} through p1 to p2;



Attributes of Quantities

Q`tolerance	Tolerance group of the quantity String – "accurate_voltage", "default_voltage"
Q'dot	Derivative of Q with respect to time (dQ/dt)
Q'integ	Integral of Q with respect to time
Q'above/expr)	Boolean signal, true if $Q > \text{expr}$, else false
Q'delayed(T)	Used to generate events for digital simulation
Q'slew(max_rising,max_falling)	New quantity with the value of Q delayed by T
	Quantity that follows Q but the edges/slopes are limited by the values of max_rising, max_fallin

$\int Q dt$

Q'ltf (num, den) Laplace-domain transfer function of Q, where num and den are the numerator and denominator coefficients of the transforming system as real vectors

Q'zoh(t_sample, t_initial) Sampled version of Q with a sample rate of t_sample and first sample at t_initial (omitted if first sample at 0.0)

Q'ztf(num, den, t_sample, t_initial) Z-transfer domain function of Q, where num and den are the numerator and denominator of the transform as real vectors

Attributes of Signals that are Quantities

Attributes of Signals that are only available for AMS
Interface between digital and analog world

$S_{\text{ramp}}(t_{\text{rise}}, t_{\text{fall}})$

Quantity following S with the given
rise and fall times (no discontinuity)
S must be of a real type

$S_{\text{slew}}(\text{rising_slope}, \text{falling_slope})$

Quantity following S with given
(not max.) rising and falling slopes