LUND UNIVERSITY | LTH FACULTY OF ENGINEERING

Introduction to VHDL

KRZYSZTOF KUCHCINSKI
Krzysztof.Kuchcinski@cs.lth.se

Outline

Basic Aspects

VHDL History

- The name: VHSIC Hardware Description Language
- Important dates:
  - 1983: development started with support from US government.
  - 1987: adopted by IEEE as a standard (IEEE Std. 1076 - 1987).
  - 1993: VHDL'92 adopted as a standard after revision of the initial version (IEEE Std. 1076 - 1993).

Main Features

- Supports the whole design process:
  - system level
  - RT level
  - logic level
  - circuit level (to some extent)
- Suitable for specification in
  - behavioral domain
  - structural domain
- Precise simulation semantics is associated with the language constructs

Basic Constructs

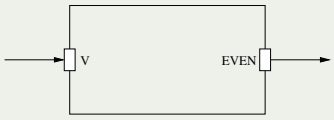
- The basic building block of a VHDL model is the entity
- An entity is described as a set of *design units*:
  - **entity declaration**
  - **architecture body**
  - package declaration
  - package body
  - configuration declaration

An Example

An Example

Example

A four bit parity generator



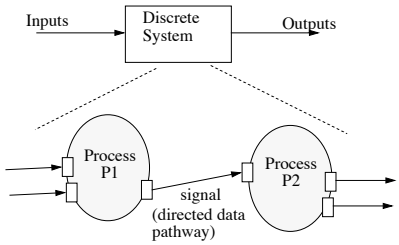

```
entity PARITY is
   port(V:in BIT_VECTOR(3 downto 0);
        EVEN:out BIT);
end PARITY;
```


Architecture body for parity generator – behavioral

```
architecture PARITY_BEHAVIORAL of PARITY is
begin
  process(V)
    variable NR_1: NATURAL;
  begin
    NR_1 := 0;
    for I in 3 downto 0 loop
      if V(I) = '1' then
        NR_1 := NR_1+1;
      end if;
    end loop;
    if NR_1 mod 2 = 0 then
      EVEN <= '1' after 2.5 ns;
    else
      EVEN <= '0' after 2.5 ns;
    end if;
  end process;
end PARITY_BEHAVIORAL;
```

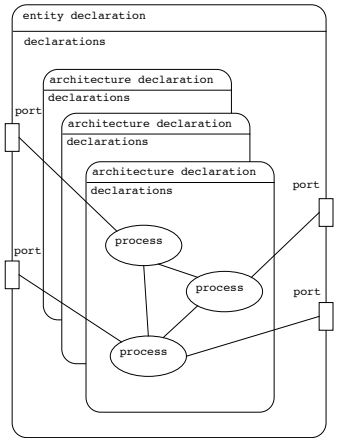
The VHDL Simulation Mechanism

A Model of Behavior



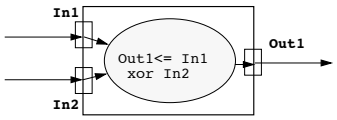
- a process can be view as a *sequential program*,
- a process can call subprogram (procedures and functions),
- processes are *run in parallel*,
- the process can be *suspended* (wait statement) and *reactivated* by receiving a signal or timing condition.

A Model of Behavior (cont'd)



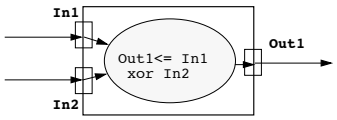
A Model of Behavior (cont'd)

```
entity Xor is
  port (In1, In2: BIT; Out1: out Bit);
end Xor;
```



A Model of Behavior (cont'd)

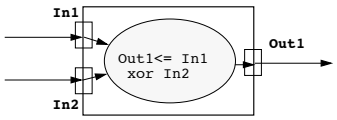
```
entity Xor is
  port (In1, In2: BIT; Out1: out Bit);
end Xor;
```




```
architecture Behavior1 of Xor is
  constant Delay : Time := 5 ns;
begin
  process (In1, In2)
  begin
    Out1 <=In1 xor In2 after Delay;
  end process;
end Behavior1;
```


A Model of Behavior (cont'd)

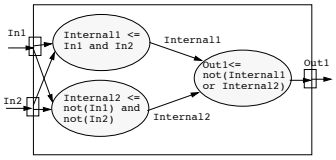
```
entity Xor is
  port (In1, In2: BIT; Out1: out Bit);
end Xor;
```




```
architecture Behavior1 of Xor is
  constant Delay : Time := 5 ns;
begin
  process (In1, In2)
  begin
    Out1 <=In1 xor In2 after Delay;
  end process;
end Behavior1;
```

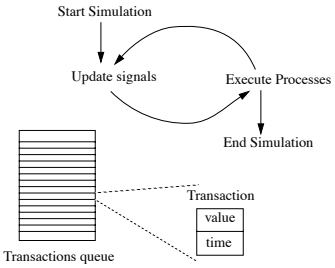
```
architecture Behavior1a of Xor is
  constant Delay : Time := 5 ns;
begin
  process
  begin
    Out1 <=In1 xor In2 after Delay;
    wait on In1, In2;
  end process;
end Behavior1a;
```


A Model of Behavior (cont'd)



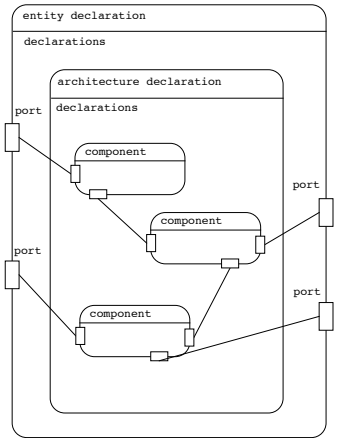

```
architecture Behavior2 of Xor is

  constant Delay : Time := 5 ns;
  signal Internal1, Internal2: Bit;

begin
  process (In1, In2)
  begin
    Internal1 <=In1 and In2 after Delay;
  end process;

  process (In1, In2)
  begin
    Internal2 <= not(In1) and not(In2) after Delay;
  end process;

  process (Internal1, Internal2)
  begin
    Out1 <= not (Internal1 or Internal2) after Delay;
  end process;

end Behavior2;
```


A Model of Time

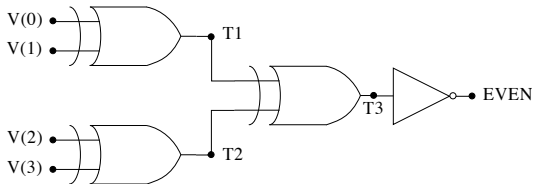


- collection of transactions is called a *signal driver*,
- *two-stage model* of time – a component reacts to activity on its inputs respond through its output connections,
- *delta delay* is used to change a simulation cycle without updating the simulation time.

A Model of Structure



Parity Generator – Structural



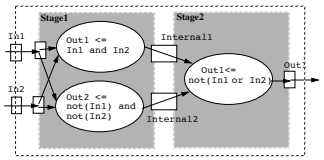
Example (cont'd)

```
use WORK.all;
architecture PARITY_STRUCTURAL of PARITY is
  component XOR_GATE
    port(X,Y: in BIT; Z: out BIT);
  end component;

  component INV
    generic(DEL: TIME);
    port(X: in BIT; Z: out BIT);
  end component;

signal T1, T2, T3: BIT;

begin
  XOR1: XOR_GATE port map (V(0), V(1), T1);
  XOR2: XOR_GATE port map (V(2), V(3), T2);
  XOR3: XOR_GATE port map (T1, T2, T3);
  INV1: INV
    generic map (0.5 ns)
    port map (T3, EVEN);
end PARITY_STRUCTURAL;
```


Example – testbench

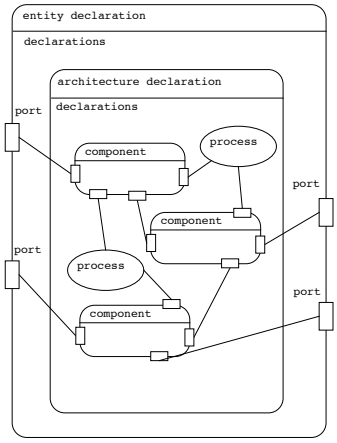
```
entity BENCH is
end BENCH;

use WORK.all;
architecture ARCH_BENCH of BENCH is
  component PARITY
    port(V: in BIT_VECTOR (3 downto 0); EVEN: out BIT);
  end component;
  for PARITY_GENERATOR:PARITY use
    entity PARITY(PARITY_STRUCTURAL);
  signal VECTOR: BIT_VECTOR (3 downto 0);
  signal E: bit;
begin
  VECTOR <= '0010',
     '0000' after 3 ns,
     '1001' after 5.8 ns,
      . . .
     '0111' after 44.5 ns,
     '1101' after 50 ns;
  PARITY_GENERATOR:PARITY port map(VECTOR, E);
end ARCH_BENCH;
```


Model of Structure

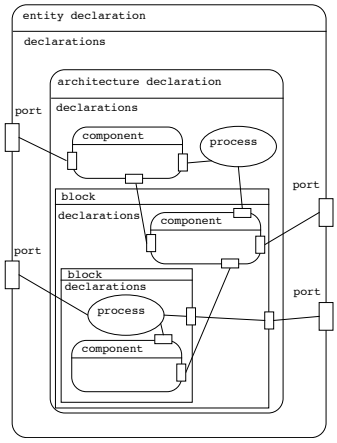



```
architecture Structure of Xor is
signal Internal1, Internal2: Bit;

component Stage1
  port (I1, I2: Bit; O1, O2: out Bit);
end component;
component Stage2
  port (I1, I2: Bit; O1: out Bit);
end component;

begin
  U0: Stage1 port map (In1, In2,Internal1,Internal2);
  U1: Stage2 port map (I1=>Internal1, I2=>Internal2, O1=>Out);
end Structure;
```

Mixed Descriptions



Block Structuring in VHDL



Signal Assignment and Delay Mechanisms

The wait statement

- A process may suspend itself by executing a wait statement:

```
wait on A,B,C until A < 2*B for 100 ns;
```

  - Sensitivity clause
  - Condition clause
  - Time-out clause

The VHDL Simulation Mechanism

- After *elaboration* of a VHDL model results a set of processes connected through signals.
- The VHDL model is simulated under control of an event driven simulation kernel (*the VHDL simulator*).
- Simulation is a cyclic process; each *simulation cycle* consists of a signal update and a process execution phase.
- A global clock holds the *current simulation time*; as part of the simulation cycle this clock is incremented with discrete values.

The VHDL Simulation Mechanism (cont'd)

- current signal values are *only* updated **by the simulator at certain moments during simulation!**

```
. . .
X <= 1;
if X = 1 then
  statement_sequence_1
else
  statement_sequence_2
end if;
. . .
```

- A signal assignment statement only schedules a new value to be placed on the signal at some later time which is specified by the designer as part of the signal assignment

```
S <= 1 after 20 ns, 15 after 35 ns;
```


The VHDL Simulation Mechanism (cont'd)

- **signal driver** contains the *projected output waveform* of a signal
- **projected output waveform** is a set of *transactions*
- **transaction**: pair consisting of a *value* and a *time*

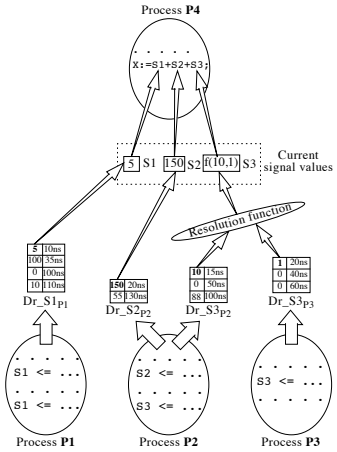
The VHDL Simulation Mechanism (cont'd)

- **signal driver** contains the *projected output waveform* of a signal
- **projected output waveform** is a set of *transactions*
- **transaction**: pair consisting of a *value* and a *time*

Signal assignment

A signal assignment only affects the projected output waveform, by placing one or more transactions into the driver corresponding to the signal and possibly by deleting other transactions.

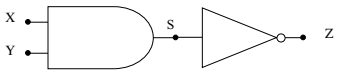
Signal Assignment



The VHDL Simulation Cycle

- The current time T_c is set to T_n
- Each active signal is updated; as result of signal updates events are generated
- Each process that was suspended waiting on signal events that occurred in this simulation cycle resumes; processes also resume which were waiting for a certain, completed, time to elapse
- Each resumed process executes until it suspends
- The time T_n of the next simulation cycle is determined as the earliest of the following three time values:
  1. TIME'HIGH
  2. The next time at which a driver becomes active
  3. The next time at which a process resumes

Delta Delay and Delta Cycle

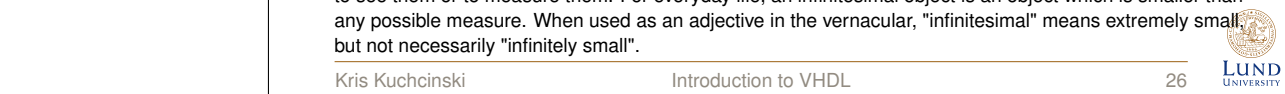



```
entity DELTA_DELAY_EXAMPLE is
  port(X, Y:in BIT; Z: out BIT);
end DELTA_DELAY_EXAMPLE;

architecture DELTA of DELTA_DELAY_EXAMPLE is
  signal S: BIT;
begin
  AND_GATE: process(X,Y)
  begin
    S <= X and Y;
  end process;
  INVERTER: process(S)
  begin
    Z <= not S;
  end process;
end DELTA;
```

Delta Delay

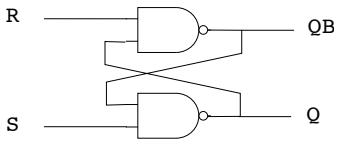
- If no delay time is specified, a delta delay is assumed for any signal assignment.
- Delta delay represents an infinitesimal[1] delay, less than any measurable time (i.e., femtoseconds), but still larger than zero.

[1] Infinitesimals (from a 17th century Modern Latin coinage infinitesimus, originally referring to the "infinite-th" member of a series) have been used to express the idea of objects so small that there is no way to see them or to measure them. For everyday life, an infinitesimal object is an object which is smaller than any possible measure. When used as an adjective in the vernacular, "infinitesimal" means extremely small, but not necessarily "infinitely small".



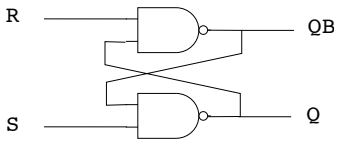
Delta Delay Example

```
QB <= R nand Q;
Q <= S nand QB;
```

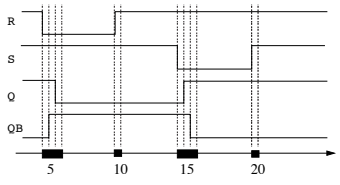


Delta Delay Example

```
QB <= R nand Q;
Q <= S nand QB;
```




```
R <= '0' after 5ns, '1' after 10ns;
S <= '0' after 15ns, '1' after 20ns;
```



Signal Assignment Statement

The projected output waveform stored in the driver of a signal can be modified by a *signal assignment statement*.

```
signal_assignment_statement ::=
   target <= [transport |
              [reject time_expression] inertial] waveform;

waveform ::= waveform_element {, waveform_element}

waveform_element ::= value_expression [after time_expression]
```

Signal Assignment Statement

The projected output waveform stored in the driver of a signal can be modified by a *signal assignment statement*.

```
signal_assignment_statement ::=
   target <= [transport |
              [reject time_expression] inertial] waveform;

waveform ::= waveform_element {, waveform_element}

waveform_element ::= value_expression [after time_expression]

S <= transport 100 after 20 ns, 15 after 35 ns;
S <= 1 after 20 ns,15 after 35 ns;
```

The concrete way a driver is updated as result of a signal assignment depends on the *delay mechanism*.

Transport Delay

- Transport delay models devices that exhibit nearly infinite frequency response: any pulse is transmitted, no matter how short its duration.
- Update rule:
  1. All old transactions scheduled to occur at the *same time or after* the first new transaction are deleted from the projected waveform.
  2. The new transactions are appended to the end of the driver.

Transport Delay – Example

Consider the following assignments executed at simulation time 100 ns (the projected waveform consists of a single transaction with value 0):

```
S <= transport 100 after 20 ns, 15 after 35 ns;
S <= transport 10 after 40 ns;
S <= transport 25 after 38 ns;
```

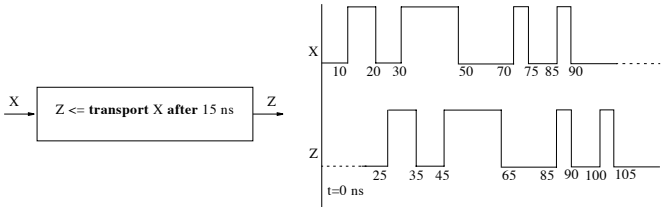
Driver for S after first two assignments:

0	100	15	10
100 ns	120 ns	135 ns	140 ns

Driver for S after last assignment:

0	100	15	25
100 ns	120 ns	135 ns	138 ns

Transport Delay – Example



Inertial Delay

- Inertial delay models the timing behaviour of current switching circuits: an input value must be stable for a duration before the value propagates to the output.
- Additional update rule (after operations have been performed like for transport delay):
  - all old transactions scheduled to occur *before* the first new transaction are deleted from the projected waveform
  - accepted are those transactions which are immediately preceding the first new transaction and have the same value with it

Inertial Delay – Example

Consider the following assignments executed at simulation time 100 ns (the projected waveform consists of a single transaction with value 0):

```
S <= 1 after 20 ns, 15 after 35 ns;
S <= 8 after 40 ns, 2 after 60 ns,
     5 after 80 ns, 10 after 100 ns;
S <= inertial 5 after 90 ns;
```

Inertial Delay – Example

Consider the following assignments executed at simulation time 100 ns (the projected waveform consists of a single transaction with value 0):

```
S <= 1 after 20 ns, 15 after 35 ns;
S <= 8 after 40 ns, 2 after 60 ns,
     5 after 80 ns, 10 after 100 ns;
S <= inertial 5 after 90 ns;
```

1st assignment:

0	1	15
100 ns	120 ns	135 ns

Inertial Delay – Example

Consider the following assignments executed at simulation time 100 ns (the projected waveform consists of a single transaction with value 0):

```
S <= 1 after 20 ns, 15 after 35 ns;
S <= 8 after 40 ns, 2 after 60 ns,
     5 after 80 ns, 10 after 100 ns;
S <= inertial 5 after 90 ns;
```

1st assignment:

0	1	15
100 ns	120 ns	135 ns

2nd assignment:

0	8	2	5	10
100 ns	140 ns	160 ns	180 ns	200 ns

Inertial Delay – Example

Consider the following assignments executed at simulation time 100 ns (the projected waveform consists of a single transaction with value 0):

```
S <= 1 after 20 ns, 15 after 35 ns;
S <= 8 after 40 ns, 2 after 60 ns,
     5 after 80 ns, 10 after 100 ns;
S <= inertial 5 after 90 ns;
```

1st assignment:

0	1	15
100 ns	120 ns	135 ns

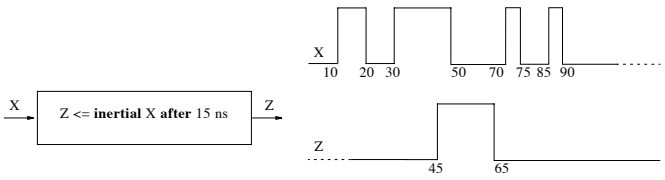
2nd assignment:

0	8	2	5	10
100 ns	140 ns	160 ns	180 ns	200 ns

3rd assignment:

0	5	5
100 ns	180 ns	190 ns

Inertial Delay – Example



VHDL for System Synthesis

VHDL For System Synthesis

- Semantic of VHDL is simulation based
- VHDL widely used for synthesis
- Problems:
  1. VHDL has the rich capabilities of a modern programming language $\Rightarrow$ some facilities are not relevant for hardware synthesis.
  2. Some features are semantically explained in terms of simulation (process interaction, timing model).
- Solutions:
  1. subsetting
  2. modeling guidelines

VHDL For System Synthesis (cont'd)

- VHDL synthesis tools at logic and RT level are commonly available today.
- Industrial use of high-level synthesis with VHDL is at the beginning.