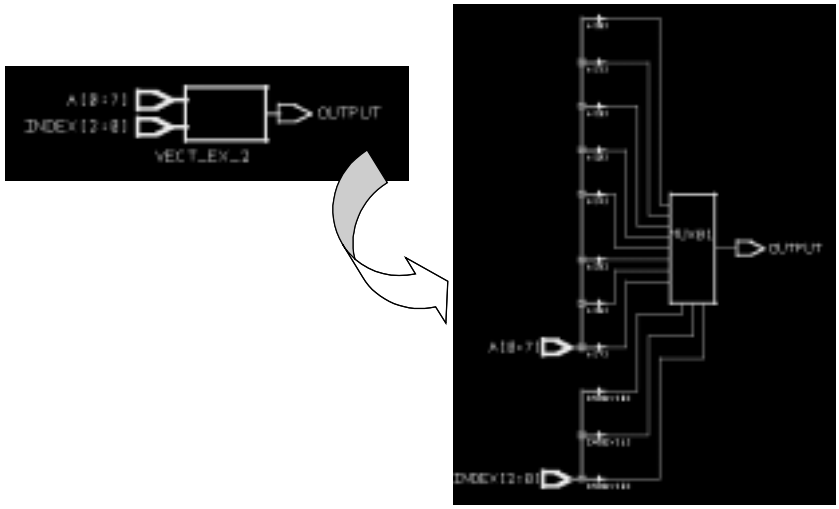
VHDL: vector indexes - 3



VHDL: vectors

- Types and subtypes
  - Supported types are:
    - Scalar
      - Enumerated
      - Integer
    - Composite
  - One dimensional arrays

VHDL Synthesis Subset

VHDL: Operators & Operands

- Arithmetic
  - abs, **, /, mod, rem are not supported
  - * supported only if:
    - both operands are constant
    - the second operand is a power of two
  - physical, real, string and null literals are not supported

VHDL: Operators & Operands

- Relational

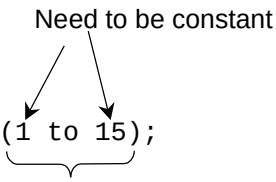
```
entity ex is
  port(A, B: in Integer range 0 to 15;
       Z: out boolean);
end ex;
architecture arch of ex is
begin
  Z <= (A < B);
end arch;
```

Usually supported
<, >, <=, >=, =, /=

! These operators may require a large amount of area

VHDL Slices

```
package ops is
  subtype WORD is bit_vector(1 to 16);
  function asr (INP: WORD)
      return WORD;
end ops;
package body ops is
  function asr (INP: WORD)
      return WORD is
      variable RESULT: WORD;
  begin
      RESULT(1) := INP(1);
      RESULT(2 to 16) := INP(1 to 15);
      return RESULT;
  end;
end ops;
```



VHDL functions

```
entity EX is
  port(INPUT : in WORD ;
       OUTPUT : out WORD) ;
end EX ;
architecture EX_1 of EX is
begin
      OUTPUT <=   asr(asr(asr(INPUT))) ;
end EX_1 ;
```

- The synthesizer creates hardware each time the function is used.
- Recursion is bound to a constant and not all tools support it.

Some restrictions

- The synthesis subset suggests some restrictions on the following elements:
  - WAIT statement
  - LOOP statement
  - BLOCK statement
  - GENERATE statement

WAIT restrictions

- The UNTIL clause is the only supported clause:

```
WAIT UNTIL sig'EVENT AND sig= value;
```

- This statement has to be the first statement of the process.
- Not supported:
  - timeout clause (WAIT FOR 7ns)

LOOP restrictions

- The only allowed loop statements are FOR loops.

```
FOR … IN … TO … LOOP
     sequence of statements
END LOOP;
```

- The discrete range in a FOR iteration scheme has to be static.

GENERATE & BLOCK

- They are not supported.
  - No GUARDED BLOCK
  - BLOCK statement is ignored
- The user must spend time to explicit what these constructs would immediately implement.
- Instead of the block statement hierarchy should be used.

Synthesized circuits

- How to obtain the “desired” kind of circuit:
  - Combinational
  - Sequential
- Main problem:
  - Be sure not to write code that “introduces” memory elements.

Combinational synthesis

- Combinational networks may be obtaineed by means of:
  - logical/arithmetical relations (data flow)
  - behavioral descritpion
- Attention is focused on:
  - Assignment of all target signals in any possible situation, otherwise memory elements are necessary to store “old” values

Data flow

- Concurrent signal assignments
- It produces combinational networks UNLESS:
  - the waveform depends on the target signal, or
  - the signal assignment originate a combinational loop

Asynchronous circuit

Data flow - 1

```
library IEEE;
use IEEE.Std_logic_1164.all;

entity MUX21 is
      port(load, data1, data2: in std_logic;
           s: out std_logic);
end MUX21;

architecture data_flow of MUX21 is
begin
      s <= data1 when (load='1') else data2;
end data_flow;
```

Multiplexer 2x1

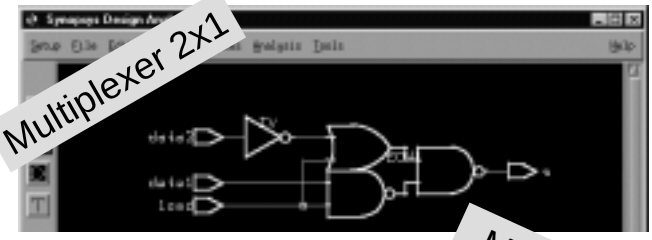
Data flow - 2

Waveform depends on target signal

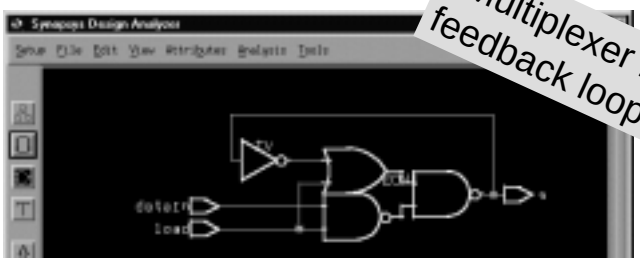
```
library IEEE;
use IEEE.Std_logic_1164.all;

entity LATCH is
   port(load, datain:    in std_logic;
         s: buffer std_logic);
end LATCH;

architecture data_flow of LATCH is
begin
   s <= datain when (load='1') else s;
end data_flow;
```

Multiplexer 2x1 with a feedback loop or a latch

Data flow - 1a/2a



Multiplexer 2x1



Multiplexer 2x1 with a feedback loop or a latch

Data flow - 3

```
library IEEE;
use IEEE.Std_logic_1164.all;
entity COMB is
      port(load, in1, in2, data2: in std_logic;
           s: out std_logic);
end COMB;
architecture data_flow of COMB is
      signal data1: std_ulogic;
begin
      data1 <= in1 or in2;
      s <= data1 when (load='1') else data2;
end data_flow;
```

Multiplexer 2x1

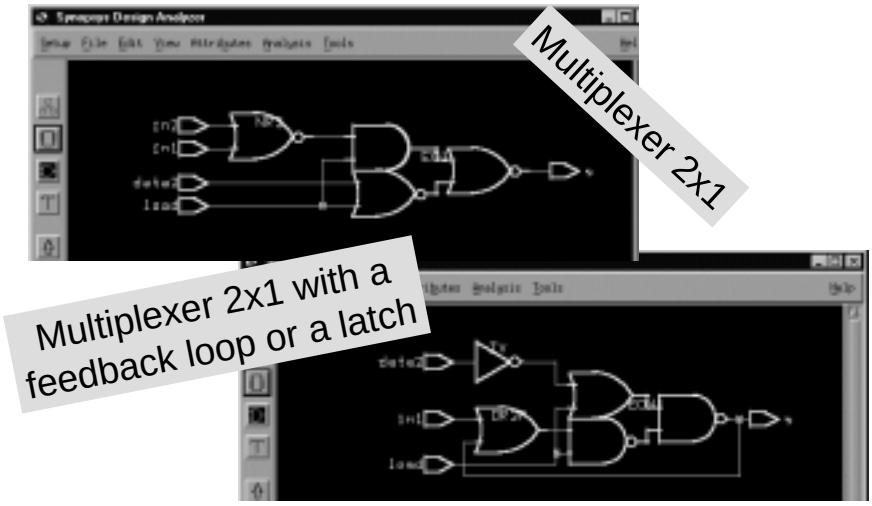
Data flow - 4

Waveform depends on target signal

```
library IEEE;
use IEEE.Std_logic_1164.all;
entity COMB is
      port(load, in1, in2, data2: in std_logic;
           s: buffer    std_logic);
end COMB;
architecture data_flow of COMB is
      signal data1: std_ulogic;
begin
      data1 <= in1 or s;
      s <= data1 when (load='1') else data2;
end data_flow;
```

Multiplexer 2x1 with a feedback loop or a latch

Data flow - 3a/4a



Behavioral

- Statements contained in (one) process
- A process which model pure combinational logic has two main characteristics:
  - The sensitivity list contains all signals that are read inside the process
  - All signal and variables are assigned in all conditional branches of the process

Sequential synthesis

- Synchronous circuits
- One clock signal is identified and eventually a reset signal (synchronous or asynchronous)
- Behavioral description ⇨ process

Process styles

- Four styles of processes are envisioned:
  - Processes with a sensitivity list including all read signals and assigning all signals and variables in all conditional branches.
  - Processes with a sensitivity list including all read signals and assigning all variables in all conditional branches.
  - Processes with a wait statement for detecting clock edges.
  - Processes with a sensitivity list including a clock signal and optionally an asynchronous reset signal.

Process “style 1”

- Processes with a sensitivity list including all read signals and assigning all signals and variables in all conditional branches.
  - The sensitivity list will contain all the intermediate signals
  - It models pure combinational logic

Process “style 1”

```
ENTITY ao IS
  port(i0, i1, i2: IN BIT;
       out0: OUT  BIT);
END ao;

ARCHITECTURE rtl OF ao IS
      SIGNAL result: BIT;
BEGIN
      PROCESS(i0, i1, i2, result)
      BEGIN
            result <= i0 AND i1;
            out0 <= result OR i2;
      END PROCESS;
END rtl;
```

Intermediate signal

If result is not inserted in the sensitivity list the synthesis would provide (when the tool provides a synthesis) a complex asynchronous sequential circuit with an event-triggered flip• flop.

Process “style 1”

- Signal assignments are updated at the end of the process execution.
- Therefore result and out0 are updated at the same time, out0 accessing the old value of result.
- To access the correct value of result, the process must be re-executed following an update of result to update out0 to the correct value.
  This is done by placing the intermediate signal in the process sensitivity list.

Process “style 1”

- Use of variables would resolve the problem of inserting in the sensitivity list intermediate signals.
- Intermediate signals with no other specific functionality are inefficient.

```
...
PROCESS(i0, i1, i2)
      VARIABLE result: BIT
BEGIN
      result := i0 AND i1;
      out0 <= result OR i2;
...
```

Process “style 2”

- Processes with a sensitivity list including all read signals and assigning all variables in all conditional branches.

Model a mixture of pure combinational logic and asynchronous latches.

- Latches are inferred when signals are not assigned in a conditional branch.

Process “style 2”

```
entity ANDGATE is
      port(in1, in2: in std_logic;
           outp: out std_logic);
end ANDGATE;
architecture correct of ANDGATE is
begin
      process(in1, in2)
            variable x: std_logic;
      begin
            if (in1='1') then
                  x:=in2;
            else
                  x := '0';
            end if;
            outp <= x;
      end process;
end correct;
```

AND gate!

Process “style 2”

```
entity ANDGATE is
      port(in1, in2: in std_logic;
              outp: out std_logic);
end ANDGATE;
architecture incorrect of ANDGATE is
begin
      process(in1, in2)
            variable x: std_logic;
      begin
            if (in1='1') then
                  x:=in2;
            end if;
            outp <= x;
      end process;
end correct;
```

A level-sensitive latch

Process “style 3”

- Processes with a WAIT statement as the first statement of a process.
- It is a clocked circuit (synchronous sequential machine)
  - Finite State Machine (FSM)
    - Moore & Mealy

```
WAIT UNTIL clk’EVENT and clk = ‘1’
```