ECE 3401 Lecture 10

More on VHDL

Outline

- More on VHDL
  - Some VHDL Basics
    - Data Types
    - Operators
    - Delay Models
  - VHDL for Simulation
  - VHDL for Synthesis

Data Types

- Every signal has a type, type specifies possible values
- Type has to be defined at signal declaration, either in
  - Entity: port declaration
  - Architecture: signal declaration
  - Package declaration
- The data types on both sides of the assignment operator '**<=**' have to match.

Standard Data Types

```
package STANDARD is       (Ashenden's book p.38, p.265 Appendix A)
 type BOOLEAN is (FALSE,TRUE);
 type BIT is ('0', '1');
 type CHARACTER is (-- Ascii set);
 type INTEGER is range -- implementation_defined
 type REAL is range -- implementation_defined
  -- BIT_VECTOR, STRING, TIME …
end STANDARD;
```

- Every type has a number of possible values
- Standard types are defined by the language
- User can define his own types
- Number values can be communicated via signals of type ' integer ' or ' real '.

Data Type "Time"

```
architecture EXAMPLE of TIME_TYPE is
  signal     CLK  : bit  := '0';
  constant PERIOD : time := 50 ns;

begin
process
 begin
     wait for 50 ns;
          . . .
     wait for PERIOD ;
          . . .
     wait for 5 * PERIOD ;
          . . .
     wait for PERIOD * 5.5;
end process;
. . .

-- concurrent signal assignment
CLK <= not CLK after 0.025 us;
-- or with constant time
-- CLK <= not CLK after PERIOD/2;
end EXAMPLE;
```

- Type declaration:

```
type time is range implementation
defined
        units
        fs;
        ps=1000fs;
        ns=1000ps;
        us=1000ns;
        ms=1000us;
        sec=1000ms;
        min=60sec;
        hr=60min;
end units;
```

- Usage:
  - testbenches
  - gate delays
- Multiplication/division
  - multiplied/divided by integer/real, returns TIME type
  - internally in smallest unit (fs)

'Integer' and 'Bit' Types

```
architecture EX1 of DATATYPES is

 signal SEL : bit ;
 signal A,B,Z : integer range 0 to 3;
begin
  A <= 2;
  B <= 3;

 process(SEL, A, B)
 begin
   if SEL = '1' then
     Z <= A;
   else
     Z <= B;
   end if;
 end process;
end EX1;
```

=

```
architecture EX2 of DATATYPES is

 signal SEL : bit ;
 signal A,B,Z: bit_vector(1 downto 0);
begin
  A <= "10";
  B <= "11";

 process(SEL,A,B)
 begin
  if SEL = '1' then
       Z <= A;
   else
       Z <= B;
  end if;
 end process;
end EX2;
```

- During synthesis, integer signals will be mapped to a number of wires. These wires could be modeled via bit vectors; yet 'bit_vector' signals do not have a numerical interpretation associated with them.
  - Therefore the synthesis result for the two example architectures would be the same.

Types of Assignment for 'bit' Data Types

```
architecture EXAMPLE of ASSIGNMENT is

 signal Z_BUS: bit_vector(3 downto 0);
 signal BIG_BUS: bit_vector(15 downto 0);

begin

  -- legal assignments:
  Z_BUS(3) <= '1';
  Z_BUS <= "1100";

  Z_BUS <= b"1100";
  Z_BUS <= x"c";
  Z_BUS <= X"C";
  BIG_BUS <= B"0000_0001_0010_0011";

end EXAMPLE
```

- Single bit values are enclosed in single quotation mark (')
- Vector values are enclosed in double quotation mark (")
  - Optional base specification (default: binary)
  - Values may be separated by underscores to improve readability
- Valid assignments for the datatype 'bit' are also valid for all character arrays, e.g. 'std_(u)logic_vector'

Assignment with Array Types

- Elements are assigned according to their position, not their number
- The direction of arrays should always be defined the same way

```
architecture EXAMPLE of ARRAYS is
 signal Z_BUS : bit_vector (3 downto 0);
 signal C_BUS : bit_vector (0 to 3);
begin
   Z_BUS <= C_BUS;
end EXAMPLE;
```

Slices of Arrays

```
architecture EXAMPLE of SLICES is
  signal BYTE : bit_vector (7 downto 0);
  signal A_BUS, Z_BUS : bit_vector (3 downto 0);
  signal A_BIT : bit;

begin
  BYTE (5 downto 2) <= A_BUS;
  BYTE (5 downto 0) <= A_BUS;   -- wrong

  Z_BUS (1 downto 0) <= `0` & A_BIT;
  Z_BUS <= BYTE (6 downto 3);
  Z_BUS (0 to 1) <= `0` & B_BIT;   -- wrong
  A_BIT <= A_BUS (0);
end EXAMPLE;
```

- Slices select elements of arrays

The direction of the "slice" and of the "array" must match.

Aggregates

```
architecture EXAMPLE of AGGREGATES is
  signal BYTE : bit_vector (7 downto 0);
  signal Z_BUS : bit_vector (3 downto 0);
  signal A_BIT, B_BIT, C_BIT, D_BIT : bit;

begin
  Z_BUS <= ( A_BIT, B_BIT, C_BIT, D_BIT ) ;
  ( A_BIT, B_BIT, C_BIT, D_BIT ) <= bit_vector'("1011");
  ( A_BIT, B_BIT, C_BIT, D_BIT ) <= BYTE(3 downto 0);
  BYTE <= (7 => '1', 5 downto 1 => '1', 6 => B_BIT, others => '0');
end EXAMPLE;
```

- Aggregates bundle signals together, may be used on both sides of an assignment
- keyword **'others'** selects all remaining elements
- Some aggregate constructs may not be supported by your synthesis tool

Outline

- More on VHDL
  - Some VHDL Basics
    - Data Types
    - Operators
    - Delay Models
  - VHDL for Simulation
  - VHDL for Synthesis

Operators

- **Logical**: NOT, AND, OR, NAND, NOR, XOR, XNOR
- **Relational**: =, /=, <, <=, >=, >
- **Shift**: sll, srl, sla, sra, rol, ror
- **Arithmetic:** +, -, *, /, mod, rem, **, abs, &

Logical Operators

```
entity LOGIC is
port (A, B, C, D : in std_logic;
        Z1   : out std_logic;
      EQUAL : out boolean);
end LOGIC;

architecture ARCH of LOGIC is
begin

Z1 <= A and (B or (not C xor D)));
EQUAL <= A xor B;     -- wrong

end ARCH;
```

- Priority:
  - NOT (top priority)
  - AND, OR, NAND, NOR, XOR, XNOR (equal priority)
- Brackets must be used to define the order of evaluation
- Data types have to match.

Logical Operations with Arrays

```
architecture ARCH of LOGIC is

signal A_BUS, B_BUS : bit_vector (3 downto 0);
signal Z_BUS : bit_vector (4 to 7);


begin
    Z_BUS <= A_BUS and B_BUS;
end EXAMPLE
```

Operands of the same length and type

Assignment via the position of the elements (according to range definition)

Shift Operators: Examples

- Defined only for one-dimensional arrays of bit or boolean!

```
Signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);
  Z_BUS <= A_BUS sll 2;
  Z_BUS <= B_BUS sra 1;
```

Example

- Shift left logical:

  B"1111" **sll** 1 = B"1110"
  B"1100" **sll** 4 = B"0000"

- Shift right logical:

  B"1111" **srl** 1 = B"0111"
  B"1100" **srl** 5 = B"0000“

- Shift left arithmetic: (filled with rightmost bit)

  B"1100" **sla** 1 = B"1000"
  B"0011" **sla** 2 = B"1111"

- Shift right arithmetic: (filled with leftmost bit)

  B"1100" **sra** 1 = B"1110"
  B"1100" **sra** -1 = B"1000"

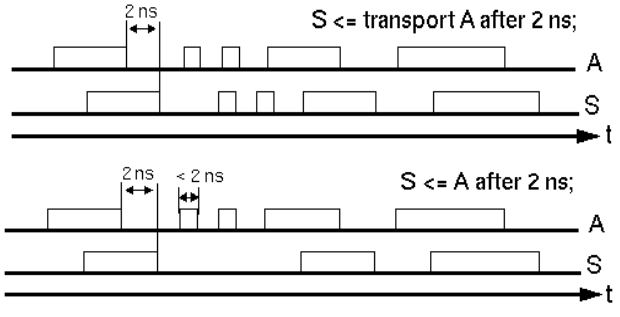
Concatenation

- Operator: &

```
architecture EXAMPLE_1 of CONCATENATION is
signal BYTE : bit_vector (7 downto 0);
signal A_BUS, B_BUS : bit_vector (3 downto 0);
begin
    BYTE <= A_BUS & B_BUS;
end EXAMPLE;
```

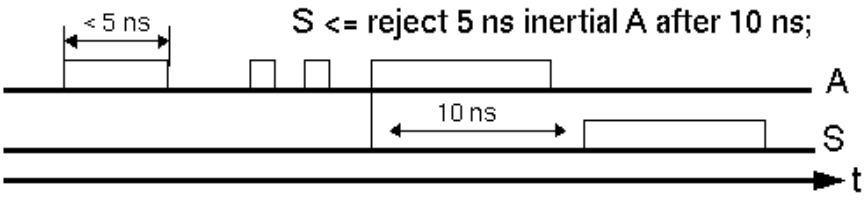
Delay Models

- **Transport delay model:**
  - everything is transferred via the signal
- **Inertial delay model:** (default delay mechanism)
  - signal transitions are only transferred when the new value remains constant for a minimum amount of time, thus spikes are suppressed.



Delay Models

- **Inertial delay with pulse rejection model:**
  - Models spike-proof behavior. A value is transmitted as long as the pulse is longer than the rejection limit.



Outline

- More on VHDL
  - Some VHDL Basics
  - VHDL for Simulation
  - VHDL for Synthesis

Sequence of Compilation

- Main components are analyzed before side- or sub-components
  - entity before architecture
  - package before package body
- The component which is referred to in another one has to be analyzed first
  - package before entity/architecture
  - configuration after entity/architecture

Simulation Flow

- The simulation of a VHDL model operates in three phases.
  1) First, the simulation model is created in the **elaboration** phase.
  2) In the **initialization** phase, a starting value is assigned to all signals.
  3) The model itself is executed in the **execution** phase.

Testbench

- Used to verify the specified functionality of a design
  - Provides the stimuli (test vectors) for the Unit Under Test (UUT), analyzes the UUT's response or stores the values in a file.
  - Simulation tools visualize signals by means of a waveform which the designer compares with the expected response. Debug if does not match.
- Does not need to be synthesizable
- No ports to the outside, self-contained

Testbench

- Simple testbench responses can be analyzed by waveform inspection
- Sophisticated testbenches may require more complicated verification techniques
  - Can take >50% of project resources

Structure of a VHDL Testbench

```
entity TB_TEST is
end TB_TEST;

architecture BEH of TB_TEST is
  -- component declaration of UUT
  -- internal signal definition
begin
  -- component instantiation of UUT

  -- clock and stimuli generation
  wait for 100 ns;
  A <= 0;
  CLK <= 1;
  …
end BEH;

configuration CFG1 of TB_TEST is
   for BEH;
      -- customized configuration
   end for;
end CFG_TB_TEST;
```

- Declaration of the UUT
- Connection of the UUT with testbench signals
- Stimuli and clock generation (behavioral modeling)
- Response analysis
- A configuration is used to pick the desired components for simulation
  - May be a customized configuration for testbench simulation

Simple Testbench Example

```
entity ADDER is
   port (A,B : in bit;
      CARRY,SUM : out bit);
end ADDER;


architecture RTL of ADDER is
begin
  ADD: process (A,B)
  begin
     SUM <= A xor B;
     CARRY <= A and B;
  end process ADD;
end RTL;
```

```
entity TB_ADDER IS -- empty entity is defined
end TB_ADDER;      -- No need for interface

architecture TEST of TB_ADDER is
   component ADDER
     port (A, B: in bit;
           CARRY, SUM: out bit);
   end component;
   signal A_I, B_I, CARRY_I, SUM_I : bit;

begin
  UUT: ADDER port map(A_I, B_I, CARRY_I, SUM_I);

  STIMULUS: process
    begin
            A_I <= ´0´; B_I <= ´0´; wait for 10 ns;
            A_I <= ´1´; B_I <= ´1´; wait for 10 ns;
            A_I <= ´1´; B_I <= ´0´; wait for 10 ns;
            A_I <= ´1´; B_I <= ´1´; wait for 10 ns;
            wait;
     -- and so on ...
    end process STIMULUS;
end TEST;

configuration CFG_TB_ADDER of TB_ADDER is
     for TEST
     end for;
end CFG_TB_ADDER;
```

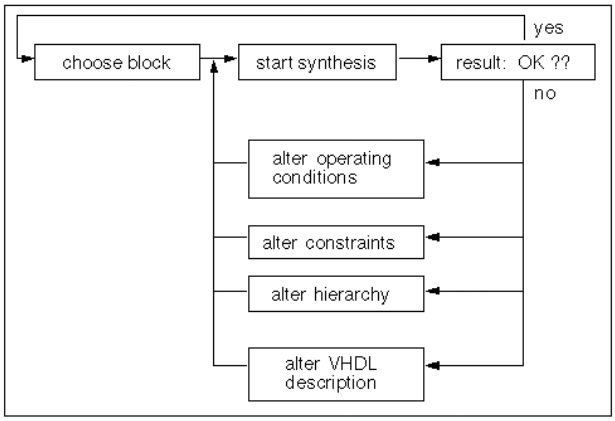
Outline

- More on VHDL
  - Some VHDL Basics
  - VHDL for Simulation
  - VHDL for Synthesis

How to do Synthesis?

- Constraints
  - Speed
  - Area
  - Power
- Macrocells
  - Adder
  - Comparator
  - Bus interface
- Optimizations
  - Boolean: mathematical
  - Gate: technological
  - The **optimization phase** requires quite a lot of iterations before the software reports its final result.

Synthesis Process in Practice



- In most cases synthesis has to be carried out several times in order to achieve an optimal synthesis result

Synthesis Strategy

- Consider the effects of **different coding styles** on the inferred hardware structures
  - If Then Else vs. Case vs. ...
- Appropriate design partitioning
  - Critical paths should not be distributed to several synthesis blocks
  - Automatic synthesis performs best at module sizes of several 1000 gates
  - Different optimization constraints used for separate blocks
    - High speed parts can be synthesized with very stringent timing constraints
    - Non-critical parts should consume the least amount of resources (area) possible.

Combinational Process

Example: Multiplexer

```
process (A, B, SEL)
begin
 if (SEL = '1') then
    OUT <= A;
 else
    OUT <= B;
 end if;
end process
```

- In simulation a process is activated when an event occurs on one of its sensitivity list's signal.
- Sensitivity list is usually ignored during synthesis
- Equivalent behavior of simulation model and hardware:
  sensitivity list has to contain all signals that are read by the process

If the signal SEL was missing, synthesis would create exactly the same result, namely a multiplexer, but simulation will show a completely different behavior.

Incomplete Assignment

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity INCOMP_IF is
  port (A,SEL: in  std_logic;
        Z: out std_logic);
end INCOMP_IF;

architecture RTL of INCOMP_IF is
begin
 process (A, SEL)
  begin
   if SEL = '1' then
      Z <= A;
   end if;
  end process;
end RTL;
```

- What is the value of Z, if SEL = `0` ?
  - The old value of Z will be maintained in the simulation, that means no change will be carried out on Z.
- What hardware would be generated during synthesis ?
  - The synthesis tools create a latch, in which the SEL signal is connected with the clock entry. It is an element very difficult to test in the synchronous design, and therefore it should not be used.

Rules for Synthesizing Combinational Logic

- Complete sensitivity list
  - RTL behavior has to be identical with hardware realization
  - An incomplete sensitivity list can cause warnings or errors
- No incomplete If-statements are allowed
  - transparent latches

Combinational Logic

```
architecture EXAMPLE of FEEDBACK is
  signal B,X : integer range 0 to 99;
begin
  process (X, B)
  begin
    X <= X + B;
  end process;

  . . .
end EXAMPLE;
```

- **Do not create combinational feedback loops!**
  - A feedback loop triggers itself all the time.
  - X is increased to its maximum value. So simulation quits at time 0 ns with an error message because X exceeds its range.

Coding Style Influence

Direct Implementation

```
process (SEL,A,B)
begin

  if SEL = `1` then
      Z <= A + B;
  else
      Z <= A + C;
end if;

end process;
```

Manual resource sharing

```
process (SEL,A,B)
variable TMP : bit;
begin
  if SEL = `1` then
      TMP := B;
  else
      TMP := C;
  end if;
  Z <= A + TMP;
end process;
```

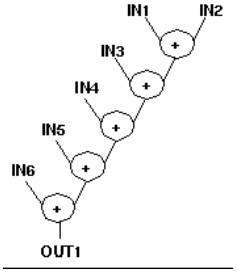
Manual resource sharing is recommended as it leads to a better starting point for the synthesis process.

Adder is shared.

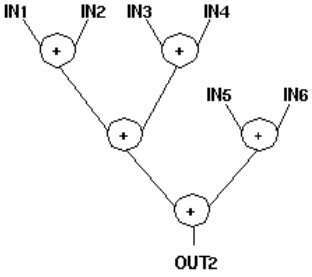
Source Code Optimization

An operation can be described very efficiently for synthesis, e.g.:

OUT1 <= IN1+IN2+IN3+IN4+IN5+IN6



OUT2 <= (IN1+IN2)+(IN3+IN4)
+(IN5+IN6)



- In one description the longest path goes via five, in the other description via three addition components - some optimization tools automatically change the description according to the given constraints.