Introduction to VHDL

• VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
• Current standard is IEEE 1076-1993 (VHDL-93). Some tools still only support VHDL-87. Tools used in the lab support VHDL-93
• ADA like syntax, strictly typed language, concurrent
• Feature rich language for modeling digital systems at system level down to gate level.
• Only a subset of the language is supported for synthesis. Only RTL VHDL code is synthesizable with most tools
• The goal with this course is not that you should learn the complete language. You should learn how to write RTL VHDL code, but also some behavioral “stuff” for test benches

Main language concepts

• Concurrency
  – VHDL can describe activities that are happening in parallel
• Structure, hierarchy
  – VHDL allows to structure a design in a hierarchical manner
• Sequential statements
  – VHDL also allows sequential execution of statements. Just like any other programming language
• Time
  – VHDL allows modeling of time

VHDL design units

• Entity declaration
  Specifies the interface of an entity
• Architecture body
  Describes the function of an entity. An entity can have more than one architectures.
• Configuration declaration
  Used to bind entity statements to particular architecture bodies.

• Package declaration
  Used to store a set of common declarations such as components, types, procedures and functions
• Package body
  Used to store the definition of functions and procedures declared in the package declaration

The entity declaration

The entity specifies the interface of a design unit. May be seen as a “black box” description

library ieee;
use ieee.std_logic_1164.all;

type std_ulogic_vector is array (natural range <>) of std_ulogic;

entity Adder is
  port ( A, B : in std_logic_vector(3 downto 0);
    Cin  : in std_logic;
    Sum  : out std_logic_vector(3 downto 0);
    Cout : out std_logic);
end Adder;

addition: architecture add of Adder is
begin
  Sum <= A xor B xor Cin;
  Cout <= A and B or (A xor Cin) or (B and Cin);
end add;

configuration CFG of Ent1 is
end CFG;

library ieee;
use ieee.std_logic_1164.all;

type std_ulogic_vector is array (natural range <>) of std_ulogic;

entity Adder is
  port ( A, B : in std_logic_vector(3 downto 0);
    Cin  : in std_logic;
    Sum  : out std_logic_vector(3 downto 0);
    Cout : out std_logic);
end Adder;

addition: architecture add of Adder is
begin
  Sum <= A xor B xor Cin;
  Cout <= A and B or (A xor Cin) or (B and Cin);
end add;

configuration CFG of Ent1 is
end CFG;
Port modes

Three most often used port modes:

- **in**
- **out**
- **inout**

Mode **in**

- Entity
- Driver
- S
- Port signal

Signal cannot be read inside entity

Mode **out**

- Entity
- Driver
- I
- S
- Port signal

Signal I can be read inside entity

Mode **inout**

- Entity
- Driver
- S
- Driver
- Port signal

Signal can be read inside entity

We will not use buffer mode!

The architecture

The architecture defines the contents of the “black box”

Modeling styles - behavioral

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity eqcomp4 is
  port
    (A, B : in std_logic_vector(3 downto 0);
     Equals : out std_logic);
end eqcomp4;

architecture behavioral1 of eqcomp4 is
begin
  process (A, B)
  begin
    if A = B then
      Equals <= '1';
    else
      Equals <= '0';
    end if;
  end process;
end behavioral1;
```

Modeling styles - dataflow

```vhdl
architecture dataflow of eqcomp4 is
begin
  Equals <= '1' when (A = B) else '0';
end dataflow;
```

```vhdl
architecture dataflow_bool of eqcomp4 is
begin
  Equals <= not(A(0) xor B(0)) and
    not(A(1) xor B(1)) and
    not(A(2) xor B(2)) and
    not(A(3) xor B(3));
end dataflow_bool;
```
Modeling styles - structural

library ieee;
use ieee.std_logic_1164.all;
use work.gates.all;

entity eqcomp4 is
  port (A, B : in std_logic_vector(3 downto 0);
        Equals : out std_logic);
end eqcomp4;

architecture structure of eqcomp4 is
  signal X : std_logic_vector(3 downto 0);
begin
  u0 : xnor2 port map (A => A(0), B => B(0), O => X(0));
  u1 : xnor2 port map (A => A(1), B => B(1), O => X(1));
  u2 : xnor2 port map (A => A(2), B => B(2), O => X(2));
  u3 : xnor2 port map (A => A(3), B => B(3), O => X(3));
  u4 : and4 port map (A => X(0), B => X(2), C => X(3),
                      D => X(4), O => Equals);
end structure;

Concurrent statements

The architecture body contains concurrent statements. Sequential statements are not allowed in the architecture body.

library ieee;
use ieee.std_logic_1164.all;

entity Test is
  port (A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Concurrent of Test is
begin
  X <= A xor B;
  with A select
    Y <= '1' when '1',
    '2' when '0',
    '-' when others;
end Concurrent;

Internal signals

Internal signals can be declared in the declarative region of the architecture.

library ieee;
use ieee.std_logic_1164.all;

entity Test is
  port (A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Internal of Test is
  signal Int : std_logic;
begin
  Int <= A xor B;
  X <= not Int;
  Y <= Int and A;
end Internal;

Processes and sequential statements

library ieee;
use ieee.std_logic_1164.all;

entity Test is
  port (A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Process of Test is
begin
  P1: process (A, B)
  begin
    if A = '1' and B = '0' then
      X <= A;
      Y <= '1';
    else
      X <= B;
      Y <= '0';
    end if;
  end process P1;
  end Process;

  X and Y is not readable!
Multiple processes interact concurrently

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity Test is
    port (
        A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Proc of Test is
    signal Internal : std_logic;
begin
    P1: process (A, B)
        begin
            if A = '1' and B = '0' then
                X <= A;
                Internal <= '0';
            else
                X <= B;
                Internal <= '1';
            end if;
        end process P1;
    P2 : process (A, B, Internal)
        begin
            if Internal = '1' then
                Y <= A;
            else
                Y <= B;
            end if;
        end process P2;
end Proc;
```

Each process execute its statements sequentially. Each process execute when there is an event on one of the signals on its sensitivity list. This may cause an event on another signal that triggers another process.

Concurrent vs. sequential execution

```vhdl
architecture Concurrent of Test is
begin
    Y <= A or B;
    Y <= C or D;
end Concurrent;
```

A signal that is assigned to within a process is not updated until the process is suspended.

```vhdl
architecture Sequential of Test is
begin
    process (A, B, C, D)
        begin
            Y <= A or B;
            Y <= C or D;
        end process;
end Sequential;
```

A signal that is assigned to within a process is not updated until the process is suspended.

Sensitivity lists

For a process that models combinational logic, the sensitivity list must be complete!

What does this process model?

```vhdl
process (A)
begin
    Y <= A or B or C or D;
end process;
```

Synplify will assume that the sensitivity list is complete. The function of the synthesized logic will not match the function of the VHDL model you simulated.

Event based simulation

Delta “time” is orthogonal to simulation time

Advance in time when no more processes are scheduled to execute at current simulation time.
Simulation - an example

architecture sim of Test is

    signal A, B, C, D : std_logic := '0';
    signal S1, S2, Y : std_logic;

    begin
        A <= '0' after 5 ns, '1' after 10 ns;
        B <= '1' after 5 ns;
        C <= '0' after 5 ns, '1' after 10 ns;
        D <= '1' after 5 ns;
        S1 <= A xor B;
        S2 <= C xor D;
        Y <= S1 and S2;
    end Sim;

Combination feedback loops

In a synchronous design combinational feedback loops should ("must") be avoided. There are some rare exceptions though.

Assume S = 0 and A = 1.
What will happen in simulation?
Simulation will never advance in time!

Data objects - constants

A constant can hold a single value of a given type. Must be declared in package, entity, architecture or process declarative region. Can improve maintainability and readability of code.

constant Mult : std_logic_vector := "0001"; -- Opcode multiply
constant Width : integer := 12;

Data objects - signals

Holds a list of values, which include the current value, past value and a set of possible scheduled values that are to appear on the signal. Future values can be assigned to the signal using the signal assignment operator.

signal ShiftReg : std_logic_vector(7 downto 0);
shiftreg <= shiftreg(6 downto 0) & Input;

May be assigned initial values when declared:

signal Count : std_logic_vector(3 downto 0) := "0101";

But this is not meaningful for synthesis!

Signals can represent wires and memory holders.
Data objects - variables

Can hold a single value of a given type, but different values can be assigned to the variable at different times using a variable assignment statement. A variable is locally declared in a process or subprograms and can only be used locally. Variables are more abstract compared to signals. Variable assignments immediately and not scheduled.

```vhdl
variable ShiftReg : std_logic_vector(7 downto 0);
shiftreg := shiftreg(6 downto 0) & Input;
```

Skahill gives the advice to avoid variables, but I disagree. Use them whenever possible since a variable uses less simulation resources than a signal. A beginner may however find working with signals easier.

Variables in processes

A variable is declared inside the process and is not visible outside the process. A variable is updated immediately. Retains its value through the simulation.

```vhdl
process(A, B, C, D)
variable Temp : std_logic;
begn
    temp := '0';
    temp := temp xor A;
    temp := temp xor B;
    temp := temp xor C;
    temp := temp xor D;
    Y <= temp;
end process;
```

Data objects - files

Files are only useful for simulation. Obviously a file data object does not belong in synthesis.

You will learn more about files when you write your own test benches in future labs.

Data types (some of them)

- **Enumeration data type:** Contains a set of user defined values
  ```vhdl
type MyBit is ('0', '1');
type Beer is (Pripps, Falcon, KeyBeer, Guiness);
```
- **Integer data type:** Defines a range of integer numbers. Default is a 32-bit integer
  ```vhdl
type CountValue is range 0 to 10;
```
- **Array data type:**
  ```vhdl
type MyBitVector is array (natural range <>) of MyBit;  -- unconstrained array
type MyByte is array (natural range 7 downto 0) of MyBit;  -- constrained array
```
- **Record data type:**
  ```vhdl
type FloatType is record
    Sign : Mybit;
    Exponent : MyBitVector(7 downto 0);
    Fraction : MyBitVector(15 downto 0);
  end record;
```
- **Subtype:**
  ```vhdl
  subtype Byte is std_logic_vector(7 downto 0);
```
package STANDARD is
    type boolean is (false, true);
    type bit is ('0', '1');
    type character is (ASCII chars...);
    type severity_level is (note, warning, error, failure);
    type integer is range -2147483648 to 2147483647;
    type real is range -1.0E308 to 1.0E308;
    type time is range -2147483647 to 2147483647 units
        fs;  ps = 1000 fs;  ns = 1000 ps;  us = 1000 ns;  ms = 1000 us;  sec = 1000 ms;  min = 60 sec;
    end units;
    subtype delay_length is time range 0 fs to time'high;
    import function now return delay_length;
    subtype natural is integer range 0 to integer'high;
    subtype positive is integer range 1 to integer'high;
    type string is array (positive range <>) of character;
    type bit_vector is array (natural range <>) of bit;
    type file_open_kind is (read_mode, write_mode, append_mode);
    type file_open_status is (open_ok, status_error, name_error, mode_error);
    attribute foreign : string;
end STANDARD;

What types are meaningful
for synthesis???

The std_logic_1164 package

The predefined type bit is defined as

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

Can not model, high impedance, don’t cares etc. So in std_logic_1164 a new type, std_ulogic, is defined:

type std_ulogic is ( 'U', -- Uninitialized
    'X', -- Forcing Unknown
    '0', -- Forcing 0
    '1', -- Forcing 1
    'Z', -- High Impedance
    'W', -- Weak Unknown
    'L', -- Weak 0
    'H', -- Weak 1
    '-' -- Don’t care
    );

std_ulogic_vector is defined as

type std_ulogic_vector is array (natural range <>) of std_ulogic;

The resolved type: std_logic

A signal that has multiple drivers must be of a resolved type. std_ulogic is not resolved but std_logic is a resolved type that is derived from std_ulogic. The resolved vector type is called std_logic_vector

Resolution table for std_logic

<table>
<thead>
<tr>
<th>U</th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
<th>W</th>
<th>L</th>
<th>H</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Driver 1

Driver 2

Attributes

There are many predefined attributes defined in VHDL. Not all can be used for synthesis. Attributes not supported for synthesis either relate to timing or are not necessary to model the physical structure of logic.

Some examples:

clock'event returns true if an event occurred on the signal clock

signal A : unsigned(3 downto 0)

A'left returns 3
A'right returns 0

See text book for further information!
Functions in std_logic_1164

A set of overloaded logic functions and conversion functions are defined in std_logic_1164. The logic functions are overloaded so they can be used for the std_logic and std_ulogic (and vector) types.

Logic functions:
AND, NAND, OR, XOR, XNOR, NOT

Conversion functions:
To_bit, To_bitvector, To_StdULogic, To_StdULogicVector, ToStdLogicVector

Also in the package, edge detecting functions:
rising_edge() and falling_edge()

The std_logic_1164 package does not contain any functions for arithmetic operations.

The numeric_std package

In the IEEE synthesis package, numeric_std, the types unsigned and signed are defined.

type unsigned is array (natural range <>) of std_logic;
type signed is array (natural range <>) of std_logic;

In the package a set of arithmetic functions are defined as well as conversion functions.

Both std_logic_1164 and numeric_std will be used in the labs. You will see examples in the first lab.

The source for the packages may be found as files in the NC VHDL and Synplify installation directories.

Type conversions

Because VHDL is a strongly typed language type conversions are unavoidable. Closely related types may be converted using the syntax:
target_type_name(expression)

Types that are not closely related need a type conversion function.

Assigning values to arrays

architecture assign of Examples is
-- Initial value, not supported for synthesis
signal Byte : std_logic_vector(7 downto 0) := "01010101";
signal Word : std_logic_vector(15 downto 0);
begin
-- Assign a string literal
Byte <= "00001111";
-- Positional association
Byte <= ('1', '0', '1', '0', '1', '0', '1', '0');
-- Assign hexadecimal value (octal is also supported)
Byte <= X"0F";
-- Set all to ones
Byte <= (others => '1');
-- Named association
Byte <= (7 | 6 => '1',
        4 => '1',
        3 downto 1 => '0',
        others => '0');
-- Concatenation
Word <= X"FF" & Byte;
-- Signextend
Word(15 downto 0) <= (others => Byte(7));
Word(7 downto 0) <= Byte;
end Examples;