Computer-aided Design of Digital Circuits: VHDL Fundamentals

by Arthur Strasser and Sean Whitty
a.strasser@tu-bs.de, whitty@ida.ing.tu-bs.de
Institute of Computer and Network Engineering
Technische Universität Braunschweig

1 VHDL: A Hardware Description Language

VHDL (Very-high-speed integrated circuit Hardware Description Language) is a solution for describing digital circuits in computer-aided electronic design automation. VHDL, along with Verilog, is the most common hardware description language used in the development of Field-Programmable Gate Arrays (FPGA) and Application-Specific Integrated Circuits (ASIC). The use of VHDL allows the development of designs with greater levels of complexity than earlier methods used to describe hardware, and increase productivity which can shorten lengthy time to market associated with other design solutions.

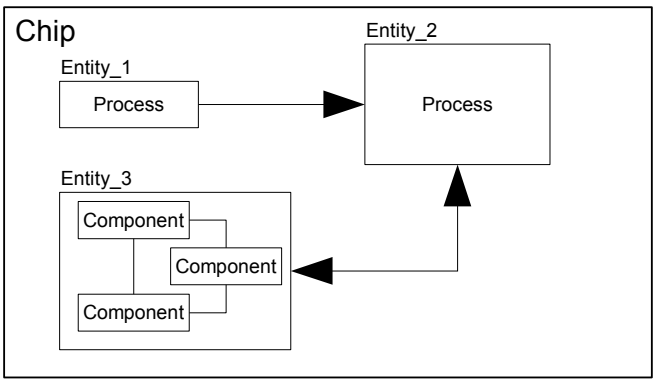
The design process can be split into three different phases:

Specification A hardware description of a given specification in VHDL is prepared

Validation/verification Simulation of a VHDL design and comparison of the results with the specification

Synthesis Automated translation of a hardware description into ready-to-fab logic (ASIC or FPGA)

1.1 System Structure



The general structure of circuits described in VHDL is composed of a heterogeneous description of dataflow and behavior between various entities and their architectures.

entity Definition of signals and the entity name from the external point of view

architecture Contains the internal realization

1.1.1 Entity

The entity can contain parameters called generics, which define specific properties, such as the width of signals and ports.

Listing 1: Example of an entity with generics

```
entity ADDER is
  generic (BIT_WIDTH : integer range 2 to 64 := 16);
  port (A, B         : in  signed(BIT_WIDTH-1 downto 0);
        SUM          : out signed(BIT_WIDTH downto 0));
end entity ADDER;
```

1.1.2 Architecture

The architecture is divided in two sections, which are known as declarations and statements. The declarations define types, constants, and subroutines. Statements compose the essence of the behavior. Three statement style types can be utilized:

Behavioral Constructed by processes with a sequential execution of statements

Dataflow Equivalent to a combinatorial function with logic operators

Structural A description of components (from a netlist)

1.1.3 Behavior

Statements contained inside a VHDL process behave similar to those in a conventional programming language, such as C or Java. **The statements within a process are sequentially executed**, based on control statements and locally defined variables. Furthermore, **all assignments of a process are evaluated at the end of the process**; in other words, new values are available only at the beginning of the next activation. In contrast to other programming languages, in VHDL, all hardware components are active at the same time. Parallelism in a design can therefore be accomplished, for example, with the definition of more than one process. **A process is activiated by signals defined in its sensitivity list**. This means that changes to signals which are defined within the sensitivity list cause the activation of the appropriate process. The following listing shows a process that is activated synchronously by a periodic clock (Listing 2), and a process that is activated asynchronously by signal changes (Listing 3).

Listing 2: Example of a synchronous process

```
architecture rtl of entity_name is
-- place here signal/constant/component declarations
begin
  -- place here statements
  process_synchronous: process(clk)
    begin
      if(rising_edge(clk)) then
        if(A = B) then
          C <= '1';
        else
          C <= '0';
        end if;
      end if;
  end process process_synchronous;
end architecture rtl;
```

Listing 3: Example of an asynchronous process

```
architecture rtl of entity_name is
-- place here signal/constant/component declarations
begin
  -- place here statements
  process_asynchronous: process(A, B)
    begin
      if(A = B) then
        C <= '1';
      else
        C <= '0';
      end if;
  end process process_asynchronous;
end architecture rtl;
```

1.1.4 Dataflow

This statement type is defined by a static assignment of signals in combination with gates, adders, decoders, and multiplexers. Dataflow statements function like a static wiring between hardware components. Every change on the input signals leads to an immediate change on the output signals.

Listing 4: Example of a asynchronous process

```
architecture rtl of entity_name is
-- place here signal/constant/component declarations
begin
  -- place here statements
  C <= not(A xor B);
end architecture rtl;
```

1.1.5 Structure

A hardware description can consist of components from several sources. IPs and libraries from other IP Core vendors are frequently included with self-written code. A so-called structure description instantiates all components and connects them to one another (Listing 5).

Listing 5: Example of a structure description

```
architecture rtl of entity_name is
-- place here signal/constant/component declarations
  -- signal declaration
  signal wire : std_logic;
  -- first component declaration
  component XYZ is
    port(X,Y : in std_logic;
         Z   : out  std_logic);
  end component XYZ;
  --second component declaration
  component INV is
    port(X : in std_logic;
         Z : out  std_logic);
  end component INV;
begin
  -- place here statements
  comp1 : XYZ port map(X => A, Y => B, Z => wire);
  comp2 : INV port map(X => wire, Z => C);
end architecture rtl;
```

1.2 Communication

In VHDL, a developer must distinguish between signals and variables. Signals can be used in a more physical manner to allow communication between various parts of a design, processes, and different levels of a design hierarchy. Signals can be thought of as a direct, wired connection with timing features that are necessary for simulation purposes. On the other hand, variables are used only within processes to save values for sequential evaluation without timing constraints.

1.2.1 Signal

Signals connect entities and establish communication between processes. In contrast to variables, the value of a signal is modified after a given period of time and not immediately. Inside a process, **a new value is assigned after the entire sequence of statements is completely processed and next iteration begins**. The following listings shows some signal declarations (6) and some language characteristics (7).

Listing 6: Example of signal declarations

```
signal COUNT     : integer range 1 to 50;
signal DATA_BUS : std_logic_vector(10 downto 0);
```

Listing 7: Example of typical language characteristics

```
VHDL     JAVA       | P1: process(A, B) is
A <= B   A = B      |   begin
C <= A   C = A      |     A <= B;
-- not equivalent   |     A <= C;
                    |     -- result: A = C
                    | end process P1;
```

1.2.2 Variable

In contrast to signals, **variables value assignments occur immediately** (Listing 9). Therefore, the results are visible to subsequent sequential statements within the same process. Communication between processes with variables is not possible. The following listing (8) shows some examples of variable declarations. One of them is used to initialize and constrain the range of a variable (INDEX).

Listing 8: Example of a variable declaration

```
variable INDEX   : integer range 1 to 50 := 10;
variable REG     : std_logic_vector(7 downto 0);
variable X, Y    : integer;
```

Listing 9: Example of a variable assignment

```
P2: process(A, B) is
  begin
    variable a, b, c : integer;
    a:= a and b;
    a:= a and c;
    -- result : a:= a and b and c
end process P2;
```

1.3 Datatypes

Every signal or variable must receive a data type at declaration time. A compiler checks every assignment and type for consistency; therefore, it is often necessary to use type conversions when making certain assignments (Listing10). The following subsections distinguish

between scalars and complex types, which are both frequently used. It should be noted that not all types are suitable for synthesis, but can nonetheless be helpful during simulation.

Listing 10: Example of type conversion

```
1     signal a : integer;
      signal b : std_logic_vector(31 downto 0);
3     -- conversion: integer to std_logic
      -- second parameter is equivalent to the width of the target
5     b <= std_logic_vector(to_unsigned(a, 32));
      -- conversion std_logic to integer
7     a <= to_integer(to_unsigned(b, 32));
```

1.3.1 Scalars

std_logic Defined by IEEE package 1164 with nine different values

boolean Boolean values: false/true

integer Values with a range between -2^{31}-1 and $+2^{31}$-1 described in dec, oct or hex

1.3.2 Complex

std_logic_vector An array of std_logic

signed/unsigned Usable with numeric_std package for numbers and computations in twos complement

1.4 Target System Architectures

Many use cases exist for VHDL, such as emulation, simulation, rapid prototyping, and automatic synthesis.
In general, however, certain design steps must be considered in advance in order to realize a hardware design. First, one must describe the hardware design in VHDL. This includes the use of one's own intellectual property and is often combined with reusable components from other vendors or recent projects. In the next step, the designer must compile and map the design to the specified technology. In most cases, the compilation and synthesis is accomplished with tools like Mentor Graphics ModelSim and Xilinx ISE, or other similar tools from technology vendors for FPGAs, ASICs or CPLDs (Complex Programmable Logic Devices). The following sections provide a short introduction of FPGAs and ASICs.

1.4.1 FPGA: Field-Programmable Gate Array

The FPGA is a programmable device which consists of LUTs (Lookup Tables) and SRAM cells. The lookup tables are realized as logic elements like AND and OR gates. A complex design is composed of many LUTs which are connected by switches to map a design onto the target FPGA hardware. The switches are equivalent to SRAM cells. These cells must be programmed by a bitstream file from a synthesis tool. FPGAs can be programmed multiple times and are very cost-effective when small volumes of a design are to be produced.

1.4.2 ASIC: Application-Specific Integrated Circuit

In contrast to FPGAs, ASICs are integrated circuits with fixed functionality that cannot be reprogrammed after design time. They offer a significant cost reduction for vendors of high volume products, such as mobile phones and other embedded systems. Moreover, ASICs allow the designer to develop resource efficient platforms by reducing the number of ICs.

1.5 VHDL Example: Gates and Processes

Listing 11: Example of an entire VHDL code file

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

entity gates is
  port (i_a, i_b, i_c, i_d, i_e, i_f : in  std_logic;
        o_x, o_y, o_z                : out std_logic);
end gates;

architecture rtl of gates is
  signal internal_out    : std_logic;
  signal i, j, k, l, m   : std_logic;
begin
  -- some gates
  i <= i_a or i_b;
  j <= i_b and i_c;
  l <= not ( i_d and i_e );
  o_z <= i when (i_f = '1') else j;

  -- some processes
  MUX : process (k, l, i_a)
  begin
    if i_a = '1' then
      m <= k;
    else
      m <= l;
    end if;
  end process MUX;

  FF : process (i_f)
  begin
    if rising_edge(i_f) then
      internal_out <= i_e;
    end if;
  end process FF;
  o_y <= internal_out;

  LATCH : process (i_f, m)
    if ( i_f = '1' ) then
      o_x <= m;
    end if;
  end process LATCH;
end rtl;
```