Sistemas Digitais I

LESI - 2º ano

Unit 5 - VHDL

João Miguel Fernandes
www.di.uminho.pt/~jmf

DEP. DE INFORMÁTICA
ESCOLA DE ENGENHARIA
UNIVERSIDADE DO MINHO

5. VHDL

- Summary -

- Design flow
- Entities and Architectures
- Types
- Functions and Procedures
- Libraries and Packages
- Structural Design
- Dataflow Design
- Behavioural Design
- Time Dimension
- Simulation

5. VHDL

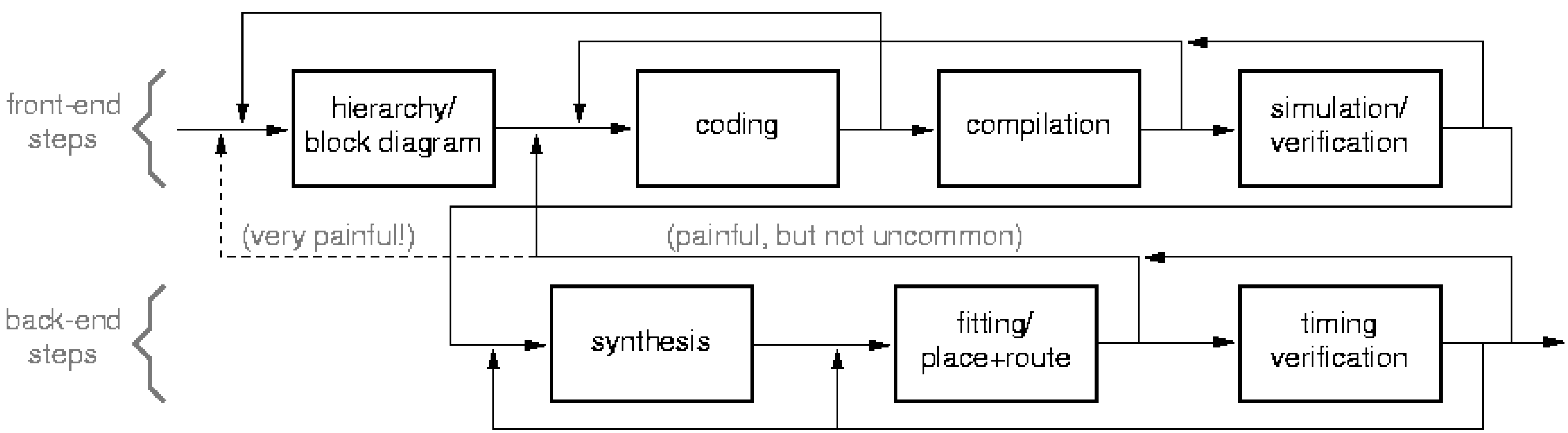
- Introduction -

- VHDL was developed, in the mid-1980s, by DoD and IEEE.
- VHDL stands for VHSIC Hardware Description Language;
  VHSIC stands for Very High Speed Integrated Circuit.
- VHDL has the following features:
  - **Designs may be decomposed hierarchically.**
  - **Each design element has both an interface and a behavioural specification.**
  - **Behavioural specifications can use either an algorithm or a structure to define the element's operation.**
  - **Concurrency, timing, and clocking can all be modelled.**
  - **The logical operation and timing behaviour of a design can be simulated.**

5. VHDL

- Design flow -

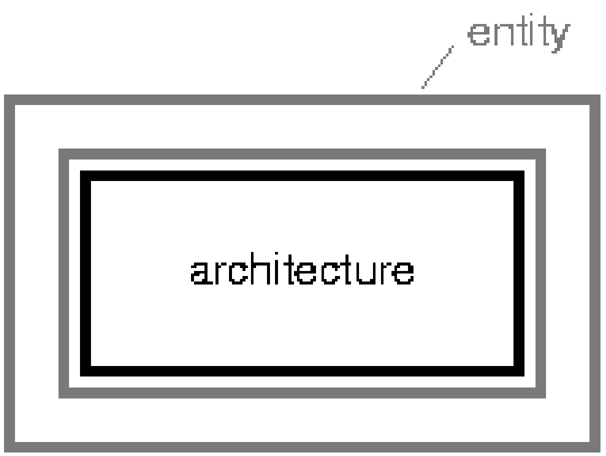
- VHDL started out as a documentation and modelling language, allowing the behaviour of designs to be specified and simulated.
- Synthesis tools are also commercially available. A synthesis tool can create logic-circuit structures directly from VHDL specifications.



5. VHDL

- Entities and Architectures (1) -

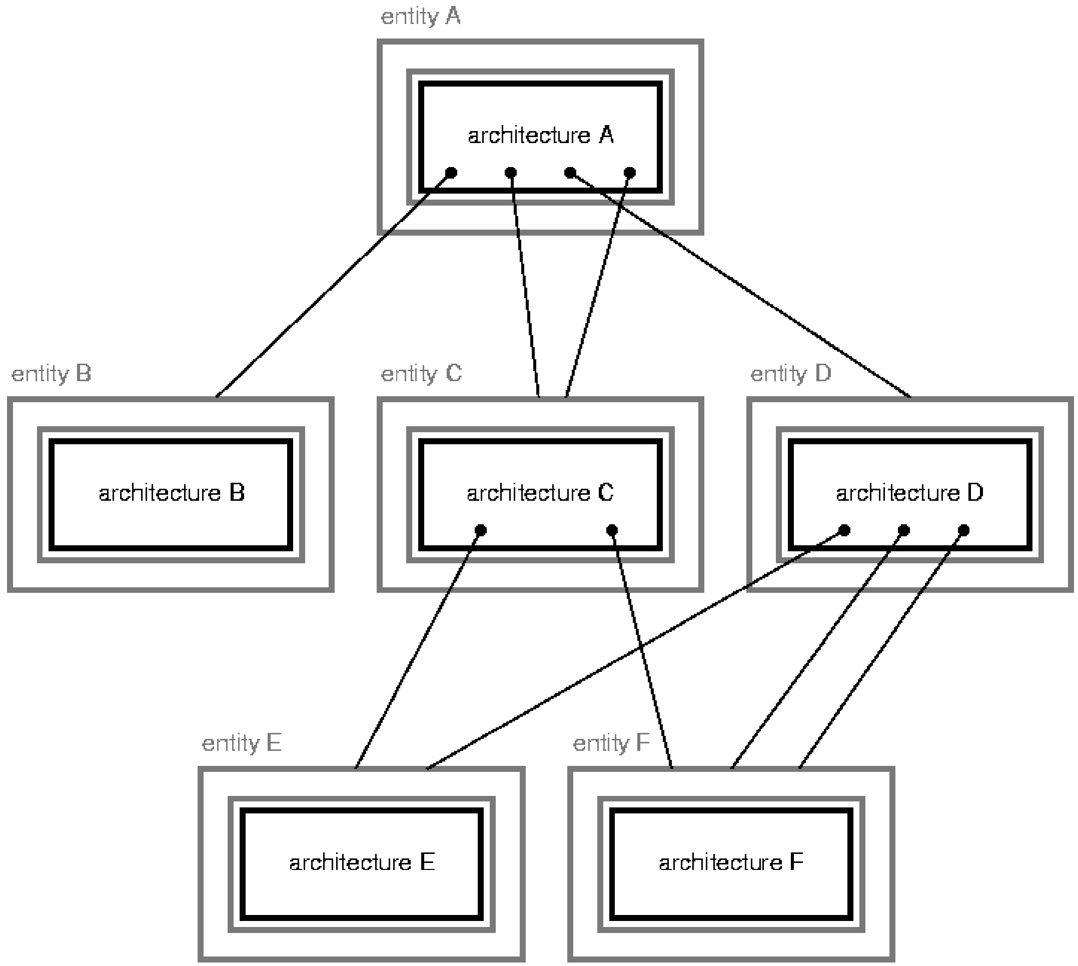
- VHDL was designed with the principles of structured programming.
- Pascal and Ada influenced the design of VHDL.
- An interface defines the boundaries of a hardware module, while hiding its internal details.
- A VHDL <u>entity</u> is a declaration of a module's inputs and outputs.
- A VHDL <u>architecture</u> is a detailed description of the module's internal structure or behaviour.



5. VHDL

- Entities and Architectures (2) -

- An architecture may use other entities.
- A high-level architecture may use a lower-level entity multiple times.
- Multiple top-level architectures may use the same lower-level entity.
- This forms the basis for hierarchical system design.



5. VHDL

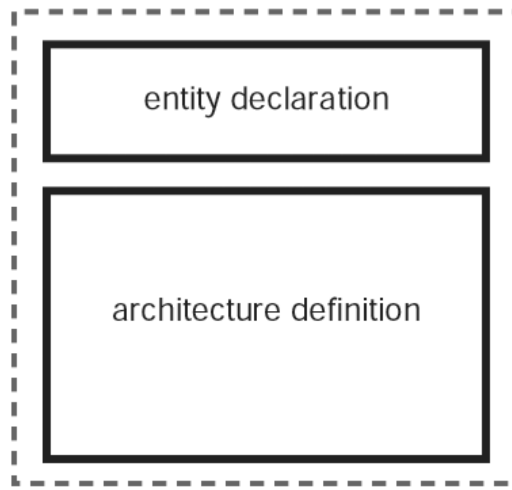
- Entities and Architectures (3) -

- In the text file of a VHDL program, the entity declaration and the architecture definition are separated.

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

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

text file (e.g., `mydesign.vhd`)



- The language is not case sensitive.
- Comments begin with 2 hyphens (--) and finish at the end of the line.
- VHDL defines many reserved words (`port`, `is`, `in`, `out`, `begin`, `end`, `entity`, `architecture`, `if`, `case`, ...).

5. VHDL

- Entities and Architectures (4) -

- Syntax of an entity declaration:

```
entity entity-name is
   port(signal-names  : mode signal-type;
        signal-names  : mode signal-type;
          ...
        signal-names  : mode signal-type);
end entity-name;
```

- mode specifies the signal direction:
  - `in`: **input to the entity**
  - `out`: **output of the entity**
  - `buffer`: **output of the entity (value can be read inside the architecture)**
  - `inout`: **input and output of the entity.**
- signal-type is a built-in or user-defined signal type.

5. VHDL

- Entities and Architectures (5) -

- Syntax of an architecture definition:

```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
    ...
   concurrent-statement
end architecture-name ;
```

- The declarations can appear in any order.
- In signal declarations, internal signals to the architecture are defined:
  `signal signal-names : signal-type;`

5. VHDL

- Types (1) -

- All signals, variables, and constants must have an associated type.
- A <u>type</u> specifies the set of valid values for the object and also the operators that can be applied it ⇒ ADT (similar concept to OO class).
- VHDL is a strongly typed language.
- VHDL has the following pre-defined types:

`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`

- `integer` includes the range -2 147 483 647 through +2 147 483 647.
- `boolean` has two values, true and false.
- `character` includes the characters in the ISO 8-bit character set.

5. VHDL

- Types (2) -

- Built-in operators for `integer` and `boolean` types.

`integer` ***Operators***		`boolean` ***Operators***	
+	addition	and	AND
–	subtraction	or	OR
*	multiplication	nand	NAND
/	division	nor	NOR
mod	modulo division	xor	Exclusive OR
rem	modulo remainder	xnor	Exclusive NOR
abs	absolute value	not	complementation
**	exponentiation		

5. VHDL

- Types (3) -

- User-defined types are common in VHDL programs.
- Enumerated types are defined by listing the allowed values.

```
type type-name is (value-list);

subtype subtype-name is type-name start to end;
subtype subtype-name is type-name start downto end;

constant constant-name : type-name := value;
```

```
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
subtype STD_LOGIC is resolved STD_ULOGIC;
```

- `type traffic_light is (reset, stop, start, go);`
- `subtype bitnum is integer range 31 downto 0;`
- `constant BUS_SIZE: integer := 32;`

5. VHDL

- Types (4) -

- <u>Array types</u> are also user-defined.

```
type type-name is array(start to end) of element-type;

type type-name is array(start downto end) of element-type;

type type-name is array(range-type) of element-type;

type type-name is array(range-type range start to end) of element-type;

type type-name is array(range-type range start downto end) of element-type;
```

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

5. VHDL

- Types (5) -

- Array literals can be specified by listing the values in parentheses:
  `xyz := ('1','1','0','1','1','0','0','1');`
  `abc := (0=>'0', 3=>'0', 9=>'0', others=>'1');`
- Strings can be used for STD_LOGIC arrays:
  `xyz := "11011001";`
  `abc := "011011110111111";`
- Array slices can be specified:
  `xyz(2 to 4)`   `abc(9 downto 0)`
- Arrays and array elements can be combined with the concatenation operator (`&`):
  `'0'&'1'&"1Z"` is equivalent to `"011Z"`.
  `B(6 downto 0)&B(7)` represents a 1-bit left rotate of the B array.

5. VHDL

- Functions and Procedures (1) -

- A function accepts a set of arguments and returns a result.
- The arguments and the result must have a type.
- Syntax of a function definition.

```
function function-name (
      signal-names  : signal-type;
      signal-names  : signal-type;
       ...
      signal-names  : signal-type
)  return return-type is
   type declarations
   constant declarations
   variable declarations
   function definitions
   procedure definitions
begin
   sequential-statement
    ...
   sequential-statement
end function-name;
```

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

5. VHDL

- Functions and Procedures (2) -

- It is often necessary to convert a signal from one type to another.
- Assume that the following <u>unconstrained array type</u> is defined:

```
type
STD_LOGIC_VECTOR is
array (natural range
<>) of STD_LOGIC;
```

```
function CONV_INTEGER (X: STD_LOGIC_VECTOR) return INTEGER is
  variable RESULT: INTEGER;
begin
  RESULT := 0;
  for i in X'range loop
    RESULT := RESULT * 2;
    case X(i) is
      when '0' | 'L'  => null;
      when '1' | 'H'  => RESULT := RESULT + 1;
      when others     => null;
    end case;
  end loop;
  return RESULT;
end CONV_INTEGER;
```

```
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
    return STD_LOGIC_VECTOR is
  variable result: STD_LOGIC_VECTOR (SIZE-1 downto 0);
  variable temp: integer;
begin
  temp := ARG;
  for i in 0 to SIZE-1 loop
    if (temp mod 2) = 1 then result(i) := '1';
    else result(i) := '0';
    end if;
    temp := temp / 2;
  end loop;
  return result;
end;
```

5. VHDL

- Functions and Procedures (3) -

- A procedure is similar to a function, but it does not return a result.
- Whereas a function call can be used in the place of an expression, a procedure call can be used in the place of a statement.
- Procedures allow their arguments to be specified with mode `out` or `inout`, so it is possible for a procedure to “return” a result.

5. VHDL

- Libraries and Packages (1) -

- A library is a place where the VHDL compiler stores information about a particular design project.
- For any design, the compiler creates and uses the `work` library.
- A design may have multiple files, each containing different units.
- When a file is compiled, the results are placed in the `work` library.
- Not all information needed in a design must be in the `work` library. A designer may rely on common definitions or functions across a family of different projects.
- A project can refer libraries containing shared definitions:

```
library ieee;
```

5. VHDL

- Libraries and Packages (2) -

- Specifying a library gives access to any previously analysed entities and architectures, but does not give access to types and the like.
- A package is a file with definitions of objects (signals, types, constants, functions, procedures, component declarations) to be used by other programs.
- A design can use a package:
  `use ieee.std_logic_1164.all;`
- Within the ieee library, the definitions are on file std_logic_1164.

```
package package-name is
   type declarations
   signal declarations
   constant declarations
   component declarations
   function declarations
   procedure declarations
end  package-name ;
package body package-name is
   type declarations
   constant declarations
   function definitions
   procedure definitions
end  package-name ;
```

5. VHDL

- Structural Design (1) -

- The body of an architecture is a series of concurrent statements.
- Each concurrent statement executes simultaneously with the other concurrent statements in the same architecture body.
- Concurrent statements are necessary to simulate the behaviour of hardware.
- The most basic concurrent statement is the <u>component statement</u>.

```
label: component-name port map (signal1, signal2, ...,signaln);

label: component-name port map (port1=>signal1,port2=>signal2,  ...,portn=>signaln);
```

- `component-name` is the name of a previously defined entity.
- One instance of the entity is created for each component statement.

5. VHDL

- Structural Design (2) -

- Before being instantiated, a component must be declared in the `component declaration` in the architecture's definition.
- A component declaration is essentially the same as the port declaration part of an entity declaration.

```
component component-name
   port (signal-names  :  mode signal-type;
         signal-names  :  mode signal-type;
          ...
         signal-names  :  mode signal-type);
end component;
```

- The components used in an architecture may be those previously defined as part of a design, or they may be part of a library.

5. VHDL

- Structural Design (3) -

```
library IEEE;
use IEEE.std_logic_1164.all;
entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0); F: out STD_LOGIC );
end prime;
architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port(I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port(I0,I1,I2,I3: in STD_LOGIC; O:out STD_LOGIC);end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

5. VHDL

- Structural Design (4) -

- An architecture that uses components is a structural description, since it describes the structure of signals and entities that realise the entity.
- The `generate` statement allows repetitive structures to be created.

```
label: for identifier in range generate
        concurrent-statement
      end generate;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
entity inv8 is
    port ( X: in STD_LOGIC_VECTOR (1 to 8);
           Y: out STD_LOGIC_VECTOR (1 to 8) );
end inv8;
architecture inv8_arch of inv8 is
component INV port (I: in STD_LOGIC; O:out STD_LOGIC); end component;
begin
  g1: for b in 1 to 8 generate
        U1: INV port map (X(b), Y(b));
      end generate;
end inv8_arch;
```

5. VHDL

- Structural Design (5) -

- Generic constants can be defined in an entity declaration.

```
entity entity-name is
   generic (constant-names : constant-type;
            constant-names : constant-type;
              ...
            constant-names : constant-type);
   port (signal-names  : mode signal-type;
         signal-names  : mode signal-type;
           ...
         signal-names  : mode signal-type);
end entity-name;
```

- Each constant can be used within the respective architecture and the value is deferred until the entity is instantiated in another architecture, using a component statement.
- Within the component statement, values are assigned to the generic constants using a `generic map` clause.

5. VHDL
- Structural Design (6) -

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv is
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end businv;

architecture businv_arch of businv is
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  g1: for b in WID-1 downto 0 generate
        U1: INV port map (X(b), Y(b));
      end generate;
end businv_arch;
```

5. VHDL

- Structural Design (7) -

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv_example is
    port ( IN8: in STD_LOGIC_VECTOR (7 downto 0);
           OUT8: out STD_LOGIC_VECTOR (7 downto 0);
           IN16: in STD_LOGIC_VECTOR (15 downto 0);
           OUT16: out STD_LOGIC_VECTOR (15 downto 0);
           IN32: in STD_LOGIC_VECTOR (31 downto 0);
           OUT32: out STD_LOGIC_VECTOR (31 downto 0) );
end businv_example;

architecture businv_ex_arch of businv_example is
component businv
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end component;
begin
U1: businv generic map (WIDTH=>8) port map (IN8, OUT8);
U2: businv generic map (WIDTH=>16) port map (IN16, OUT16);
U3: businv generic map (WIDTH=>32) port map (IN32, OUT32);
end businv_ex_arch;
```

5. VHDL

- Dataflow Design (1) -

- Other concurrent statements allow circuits to be described in terms of the flow of data and operations on it within the circuit.
- This gives origin to the dataflow description style.
- Syntax of concurrent signal assignments statements.

```
signal-name  <= expression;

signal-name  <= expression when boolean-expression else
                expression when boolean-expression else
                  ...
                expression when boolean-expression else
                expression;
```

5. VHDL

- Dataflow Design (2) -

```
architecture prime2_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= not N(3)                              and N(0);
  N3L_N2L_N1 <= not N(3) and not N(2) and     N(1)          ;
  N2L_N1_N0  <=              not N(2) and     N(1) and N(0);
  N2_N1L_N0  <=                  N(2) and not N(1) and N(0);
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

```
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0  <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0  <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

5. VHDL

- Dataflow Design (3) -

- Another concurrent statement is the selected signal assignment, which is similar to a typical CASE constructor.
- Syntax of selected signal assignments.

```
with expression select
  signal-name <= signal-value when choices,
                 signal-value when choices,
                    ...
                 signal-value when choices;
```

```
architecture prime4_arch of prime is
begin
  with N select
    F <= '1' when "0001",
         '1' when "0010",
         '1' when "0011" | "0101" | "0111",
         '1' when "1011" | "1101",
         '0' when others;
end prime4_arch;
```

```
architecture prime5_arch of prime is
begin
  with CONV_INTEGER(N) select
    F <= '1' when 1|2|3|5|7|11|13,
         '0' when others;
end prime5_arch;
```

5. VHDL

- Behavioural Design (1) -

- The main behavioural construct is the <u>process</u> which is a collection of sequential statements that executes in parallel with other concurrent statements and processes.
- A process simulates in zero time.
- A VHDL process is a concurrent statement, with the syntax:

```
process (signal-name, signal-name, ..., signal-name)
   type declarations
   variable declarations
   constant declarations
   function definitions
   procedure definitions
begin
   sequential-statement
    ...
   sequential-statement
end process;
```

5. VHDL

- Behavioural Design (2) -

- A process can not declare signals, only variables, which are used to keep track of the process state.
- The syntax for defining a variable is:
  `variable` *`variable-names`* `:` *`variable-type`*`;`
- A VHDL process is either running or suspended.
- The list of signals in the process definition (<u>sensitivity list</u>) determines when the process runs.
- A process is initially suspended. When a sensitivity list's signal changes value, the process resumes, starting at the 1st statement until the end.
- If any signal in the sensitivity list change value as a result of running the process, it runs again.

5. VHDL

- Behavioural Design (3) -

- This continues until the process runs without any of these signals changing value.
- In simulation, this happens in zero simulation time.
- Upon resumption, a properly written process will suspend after a couple of runs.
- It is possible to write an incorrect process that never suspends.
- Consider a process with just one sequential statement "`X <= not X;`" and a sensitivity list of "`(X)`".
- Since X changes on every pass, the process will run forever in zero simulated time.
- In practice, simulators can detect such behaviour, to end the simulation.

5. VHDL

- Behavioural Design (4) -

- The <u>sequential signal assignment</u> statement has the same syntax as the concurrent version (but it occurs within the body of a process):
  signal-name `<= expression;`
- The <u>variable assignment</u> statement has the following syntax:
  variable-name `:= expression;`

```
architecture prime6_arch of prime6 is
begin
  process(N)
    variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
  begin
    N3L_N0     := not N(3)                             and N(0);
    N3L_N2L_N1 := not N(3) and not N(2) and     N(1)          ;
    N2L_N1_N0  :=              not N(2) and     N(1) and N(0);
    N2_N1L_N0  :=                  N(2) and not N(1) and N(0);
    F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
  end process;
end prime6_arch;
```

5. VHDL

- Behavioural Design (5) -

- Other sequential statements include popular constructs, such as `if`, `case`, `loop`, `for`, and `while`.

```
if boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
else sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
else sequential-statement
end if;
```

```
case expression is
   when choices => sequential-statements
   ...
   when choices => sequential-statements
end case;
```

```
loop
   sequential-statement
   ...
   sequential-statement
end loop;
```

```
for identifier in range loop
   sequential-statement
   ...
   sequential-statement
end loop;
```

```
while boolean-expression loop
   sequential-statement
   ...
   sequential-statement
end loop;
```

5. VHDL

- Behavioural Design (6) -

```
architecture prime7_arch of prime is
begin
  process(N)
    variable NI: INTEGER;
  begin
    NI := CONV_INTEGER(N);
    if NI=1 or NI=2 then F <= '1';
    elsif NI=3 or NI=5 or NI=7 or NI=11 or
          NI=13 then F <= '1';
    else F <= '0';
    end if;
  end process;
end prime7_arch;
```

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
    case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end prime8_arch;
```

5. VHDL

- Behavioural Design (7) -

```
architecture prime9_arch of prime9 is
begin
  process(N)
  variable NI: INTEGER;
  variable prime: boolean;
  begin
    NI := CONV_INTEGER(N);
    prime := true;
    if NI=1 or NI=2 then null; -- boundary cases
    else for i in 2 to 253 loop
           if NI mod i = 0 then
             prime := false; exit;
           end if;
         end loop;
    end if;
    if prime then F <= '1'; else F <= '0'; end if;
  end process;
end prime9_arch;
```

5. VHDL

- Time Dimension (1) -

- None of the previous examples deals with the time dimension of the circuit operation: everything happens in zero simulated time.
- VHDL has excellent facilities for modelling the time.
- VHDL allows a time delay to be specified by using the keyword `after` in any signal-assignment statement.
- ```
  Z <= '1' after 4ns when X='1' else
       '0' after 3ns;
  ```
- This models a gate that has 4ns of delay on a 0-to-1 output transition and only 3ns on a 1-to-0 transition.
- With these values, a VHDL simulator can predict the approximate timing behaviour of a circuit.
  ```

5. VHDL

- Time Dimension (2) -

- Another way to invoke the time dimension is with `wait`.
- This sequential statement can be used to suspend a process for a specified time period.
- A `wait` statement can be used to create simulated input waveforms to test the operation of a circuit.

```
entity InhibitTestBench is
end InhibitTestBench;

architecture InhibitTB_arch of InhibitTestBench is
component Inhibit port (X,Y: in BIT; Z: out BIT); end component;
signal XT, YT, ZT: BIT;
begin
  U1: Inhibit port map (XT, YT, ZT);
  process
  begin
    XT <= '0'; YT <= '0';
    wait for 10 ns;
    XT <= '0'; YT <= '1';
    wait for 10 ns;
    XT <= '1'; YT <= '0';
    wait for 10 ns;
    XT <= '1'; YT <= '1';
    wait; -- this suspends the process indefinitely
  end process;
end InhibitTB_arch;
```

5. VHDL

- Simulation (1) -

- Once we have a VHDL program whose syntax and semantics are correct, a simulator can be used to observe its operation.
- Simulator operation begin at simulation time of zero.
- At this time, the simulator initialises all signals to a default value.
- It also initialises any signals and variables for which initial values have been explicitly declared.
- Next, the simulator begins the execution of all processes (and concurrent statements) in the design.
- The simulator uses a time-based event list and a signal-sensitivity matrix to simulate the execution of all the processes.

5. VHDL

- Simulation (2) -

- At simulation time zero, all processes are scheduled for execution.
- One of them is selected and all of its sequential statements are executed, including any looping behaviour that is specified.
- When the execution of this process is completed, another one is selected, and so on, until all processes have been executed.
- This completes one simulation cycle.
- During its execution, a process may assign new values to signals.
- The new values are not assigned immediately. They are placed on the event list and scheduled to become effective at a certain time.

5. VHDL

- Simulation (3) -

- If the assignment has an explicit simulation time (`after` clause), then it is scheduled on the event list to occur at that time.
- Otherwise, it is supposed to occur "immediately".
- It is actually scheduled to occur at the current simulation time plus one delta delay.
- The <u>delta delay</u> is an infinitesimally short time, such that the current simulation time plus any number of delta delays still equals the current simulation time.
- The delta delay concept allows processes to execute multiple times (if necessary) in zero simulated time.
- After a simulation cycle completes, the event list is scanned for the signals that change at the next earliest time on the list.

5. VHDL

- Simulation (4) -

- This may be as little as one delta delay, or it may be a real delay, in which case the simulation time is advanced.
- In any case, the scheduled signal changes are made.
- Some processes may be sensitive to the changing signals.
- All the processes that are sensitive to a signal that just changed are scheduled for execution in the next simulation cycle.
- The simulator's operation goes on indefinitely until the list is empty.
- The event list mechanism makes it possible to simulate the operation of concurrent processes in a uni-processor system.
- The delta delay mechanism ensures correct operation even though a set of processes may require multiple executions.

5. VHDL

- Simulation (5) -

```
library IEEE;
use     IEEE.std_logic_1164.all;

entity testAlu1bit is
end entity test_alu1bit;

architecture tst of testAlu1bit is

  component alu1bit is
    port (
    a, b, c  : in std_logic;
    sel : in std_logic_vector (1
    downto 0);
    res, f   : out std_logic);
  end component alu1bit;

  signal i1  : std_logic := '0';
  signal i2  : std_logic := '0';
  signal ci  : std_logic := '0';
  signal op  : std_logic_vector
       (1 downto 0) := "00";
  signal res : std_logic;
  signal co  : std_logic;
```

```
begin
  -- instanciar o sistema
  -- a testar
  ALU1: alu1bit
    port map (
      a      => i1  ,
      b      => i2  ,
      c      => ci  ,
      sel    => op  ,
      res    => res ,
      f      => co  );

  process (i1) is
  begin
    if i1='1' then
      i1 <= '0' after 10ns;
    elsif i1='0' then
      i1 <= '1' after 10ns;
    end if;
  end process;

  process (i2) i
  begin
    if i2='1' then
      i2 <= '0' after 20ns;
    elsif i2='0' then
      i2 <= '1' after 20ns;
    end if;
  end process;
```

```
  process (ci) is
  begin
    if ci='1' then
      ci <= '0' after 40ns;
    elsif ci='0' then
      ci <= '1' after 40ns;
    end if;
  end process;

  process (op) is
  begin
    if op="00" then
      op <= "01" after 80ns;
    elsif op="01" then
      op <= "10" after 80ns;
    elsif op="10" then
      op <= "11" after 80ns;
    elsif op="11" then
      op <= "00" after 80ns;
    end if;
  end process;
end architecture;
```

5. VHDL

- Synthesis (1) -

- VHDL was originally conceived as a description and simulation language.
- It was later adopted also for synthesis purposes.
- The language has many features and constructs that can NOT be synthesized.
- The subset of the language and the style of the programs presented so far are generally synthesizable by most commercial tools.
- The code that is written can have a major impact on the quality of the synthesized circuits.
- Serial control structures, like `if-elsif-elsif-else` can result in a corresponding serial chains of logic gates to test conditions.
- It is better to use a `case` or `select` statement if the conditions are mutually exclusive.

5. VHDL
- Synthesis (2) -

- Loops in processes are usually unwound to create multiple copies of combinational logic to execute the statements in the loop.
- If one wants just one copy of the combinational logic to execute the statements in the loop, then a sequential circuit must be designed.
- When using conditional statements in a process, failing to include all the input combinations will cause the compiler to introduce a latch to hold the old value that might otherwise change.
- Such latches are typically not intended.
- Finally, some language features and constructs are simply unsynthesizable, depending on the tool being used.
- Typical examples include dynamic memory, files, and pointers.