VHDL for Logic synthesis

A Book by Andrew Rushton

EE 464 Tutorial by Ayman Elnaggar

September 17, 1996

The purpose is to show as many as possible of the features of the language that can be synthesized, and how those features are interpreted by a synthesizer.

Transformation of design involves:

- Transformation from requirements to specification.
- Transformation from specification to design.
- Transformation from design to implementation.

The system model will be a VHDL model which represents the algorithm to be performed without any hardware implementation in mind. The purpose is to create a *simulation model* that can used as a formal specification of the design.

The system model is then transformed into a register-transfer level (RTL) design in preparation for synthesis.

The final stage of the design cycle is to synthesize the RTL design to produce the netlist.

RTL Design

In RTL design, a circuit is described as a set of registers and a set of transfer functions describing the flow of data between the registers.

The registers are implemented directly as flip-flops, whilst the transfer functions are implemented as blocks of combinational circuits.

Note: This division of the design is the main objective of the hardware designer using synthesis.

The RTL Design Stages

- ☐ Identify the data operations.
- ☐
- ☐ Determine the precision of the operations.
- ☐ Decide what data processing resources to provide.
- ☐ Allocate operations to resources.
- ☐ Allocate registers for intermediate results.
- ☐ Design the controller.
- ☐ Design the reset mechanism.

Example

The cross product of two vectors is given by

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = a_0 b_0 + \cdots + a_7 b_7$$

the system model in VHDL is:

```
entity cross_product is
  port (a,b: in in_vector; z :out integer)
end;

architecture rtl of cross_product is
  begin
      z<= a(0)*b(0)+a(1)*b(1)+a(2)*b(2)+
          a(3)*b(3)+a(4)*b(4)+a(5)b(5)+
          a(6)*b(6)+a(7)*b(7);
  end;
```

Combinational Logic

Design Units

- *The entity*: is the design unit that defines the interfaces to a circuit. It also defines the name of the circuit.
- *The architecture*: is the design unit that defines the contents of the circuit.

Entities and architectures therefore exist in pairs- a complete circuit description will generally have both an entity and an architecture.

Example

```
entity full_adder is
    port (a,b,c : in bit ; sum, cout : out bit);
end full_adder;
```

In this case the circuit full_adder has five ports; three input ports and two output ports.

Note: The repeat of the circuit name full_adder after the end is optional.

The structure of the architecture is

```
architecture rtl of full_adder is:
  signal sum1, sum2, c1, c2 :bit;
   begin
     sum1<= a xor b;
     c1<= a and b;
     sum2<= sum1 xor c;
     c2<= sum1 and c;
     sum <= sum2;
     cout <= c1 or c2;
   end rtl;
```

The architecture has two parts.

- The *declarative* part is the part before the keyword begin where you define internal signals. Signals are similar to ports but are internal to the circuit.
- The *statement* part is the part after the begin. This is the description of the circuit itself. In the previous example, the statement part contains only signal assignments.

Note: The ordering of the statements in the previous example are irrelevant. This is because each statement simply defines a relationship between its inputs and its output.

Signals and Ports

signals are the carriers of the data values around the architecture. Ports are the same as signals but also provide an interface through the entity so that the entity can be used as a subcircuit in a hierarchical design.

A signal is declared in the declarative part of an architecture and the declaration has two parts:

```
signal a, b, c : bit;
```

The first part is a list of signal names. The second part, after the colon, is the type of the signals.

A port is a signal and is declared in a similar way within the port specification of an entity. A port declaration has three parts:

```
a, b, c, : in bit
```

The first part is a list of port names. The second part is the mode of the port (in, out, inout). The third part is the type as in the signal.

Note:

- A port in mode *in* cannot be assigned to and so can appear only on the right-hand side of a signal assignment.
- A port in mode *out* can only be assigned to and so can appear only on the left-hand side of a signal assignment.
- If you want to connect the output of a circuit as an input to another, use Signals to read the output first.

Example

```
architecture rtl of example is
  signal temp1, temp2 :  bit;
    begin
      temp1 <= a and b;
      temp2 <= temp1 or c;
       y1 <= temp1;
       y2 <= temp2
      end
```

Initial Values

All signals have an initial value when simulation begins. This initial value either can be given in the signal declaration or will be given by default by the simulator.

Initial values given in the signal declaration look like:

```
signal a : bit := '1';
```

Note:

- For synthesis, there is no hardware interpretation of an initial value. It is not possible to initialize all signals in a circuit with a known value on power up. So, synthesis ignores initial values.
- The point is that this is not a failure of the synthesizer — it is the designer's responsibility to provide a mechanism for putting the circuit into a known state, either in the form of a reset input or by designing circuits so that they can be put into a known state.

Simple Signal assignment

The simple signal assignment looks like:

```
x <= a xor b ;
```

The rules of VHDL insist that the source of an assignment is the same type as the target.

Conditional Signal Assignments

Conditional signal assignments are signal assignments with more than one source expression, where one of the source expressions is chosen by a control condition:

```
sum <= a xor b when c = '1' else
       not (a xor b);
```

In this case, the condition is the test for c='1'. The test can be any boolean expression, which generally means a test for equality or inequality, or a comparison such as less-than.

Note:

- The rule here is that all the source expressions must be the same type as the target of the assignment.
- The hardware mapping of the conditional signal assignment is a *multiplexer* which selects between the source expressions.
- It is possible to have any number of branches; each branch but the last has a different condition. The last branch must be *unconditional else* so that one of the source expressions will always be assigned to the target.

Note: (Cont.)

- The conditions are evaluated in order so that the first condition which is true will be selected. This is equivalent in hardware terms to a series of two way multiplexers, with the first condition controlling the multiplexer nearest to the output so that the condition overrides any of the later conditions.

An example of a multiple-branch is:

```
z <= s when sel1 = '1' else
     b when sel2 = '1' else
     c;
```

Sequential VHDL

Processes

A process is a series of sequential statements that must be executed in order. At this level, VHDL is like many software programming languages. The main difference between a process and software program is that a process runs repeatedly throughout a simulation run.

A process can appear anywhere in the body of an architecture (after the *begin*).

The basic structure of a process is:

process (*sensitivity list*)
  declaration part
begin
  statement part
end process

Sensitivity List

- The sensitivity list is the list of signals to which the process is sensitive. this is a set of signal which the simulator monitors for *events* (changes of value).
- Any events on any of the signals in the sensitivity list cause the process to be executed once. That is, all the statements are executed and then the process stops and waits for further events.
- The sensitivity list is optional. If it is absent, then the process will run continuously. So, the process has should contain a *wait* statement.

Combinational Process

For a process to model combinational logic, it must be sensitive to all the signals that are read in the process. In other words, the process must be re-evaluated every time one of the inputs to the circuit it represent changes.

Note:

If a process is not sensitive to all its inputs and is not a registered (has no FF's) process, then it is unsynthesizable.

Registered Process

Both registered and combinational processes are interpreted in the same way, the only difference is that: the contents of the process, excluding the register template part, are interpreted as a combinational process and a register is added to every output.

```
process
begin
   wait until ck ='1';
    z <= a and b ;
end process;
```

Feedback

- One consequence of the behavior of signal assignments in processes is that an assignment to a signal can be overridden by later assignment.
- The second consequence of this behavior is that any reading of a signal that is also assigned to in the same process will pick up the value assigned to the signal in the previous execution of the process.

Example

```
process
begin
   wait until ck='1';
   if reset ='1' then
      count <= "0000";
   else
       count <= count + "0001";
   end if;
end process
```

Variables

Variables are used to store intermediate calculations within a process. They exist only within sequential VHDL and cannot be declared or used directly in an architecture.