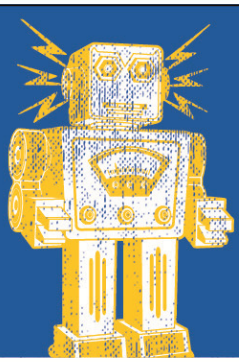
Elektronik

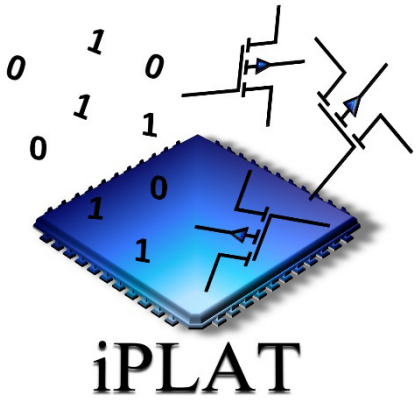


Digital System Design

Distance Learning Letter
VHDL: Combinatorial Logic

Gefördert von
MA23
Wirtschaft, Arbeit Statistik
StaDt+Wien

iPLAT – Competence Team for Innovative Platforms for Mixed Hardware/Software Systems
MA23 Project 18-06



Version: 0.5
Author: R. Höller, A. Puhm, F. Schrön

Copyright Notice

This document or parts of it (text, photos, graphics and artwork) are copyrighted and not intended to be published to the broad public, e.g., over the internet. Any redistribution, publishing or broadcast with permission only. Violation may be prosecuted by law.

Dieses Dokument bzw. Teile davon (Text, Photos, Graphiken und Artwork) sind urheberrechtlich geschützt und nicht für die breite Veröffentlichung, beispielsweise über das Internet, vorgesehen. Jegliche weitere Veröffentlichung nur mit Genehmigung. Zuwiderhandlungen können gerichtlich verfolgt werden.

Introduction

After studying the utilized VHDL simulator and the basics of VHDL constructs in the previous Distance Learning Letters, the most important VHDL syntax for combinatorial logic will be described in this Distance Learning Letter. The example given at the end of the document can be designed and verified with the simulator. It will give you the opportunity to put the acquired knowledge into practice.

Processes

One of the most important constructs in VHDL is the `process` (reserved word). It has to be used after the `begin` keyword of the architecture. A `process` is always executed sequentially (in simulation and also in synthesis). This means that all commands within the process will be executed sequentially depending on their order in the code. The sensitivity list contains signals which determine the recalculation of the whole process. A recalculation is performed if a signal changes its value. Moreover, processes without a sensitivity list are possible. These processes need at least one `wait` statement, otherwise, the process will produce an endless loop.

Statements in processes are executed sequentially. All processes of a design are executed in parallel to each other and to VHDL statements outside of processes. This is a major problem for the simulation of such a design. A CPU with one processing unit could not simulate a VHDL design with, e.g., two processes. The simulator has to determine the order of execution for these processes to simulate the behavior of the actual hardware. This problem is solved by the sensitivity list of a process. Each process of a VHDL design will be executed once at the beginning of the simulation. After that, it will only be executed again if a signal that is part of the sensitivity list changes the value (e.g., for a `std_logic` signal from 0 -> 1, 1 -> X, X -> U, etc.). The sensitivity list of a process is only important for the simulation and will be ignored by the synthesis tool. If the wrong signals are added to the sensitivity list, the behavior of the simulation will severely differ from the behavior of the actual design in the hardware. Therefore, a simple rule has to be followed: Every signal that is read in the process (e.g., part of an if-condition or on the right side of an assignment) has to be part of the sensitivity list. A single exception to this rule is a process that describes synchronous logic. In this case, only the asynchronous reset (if applicable) and the clock signal have to be part of the sensitivity list.

Sequential Statements

Several processes defined in one architecture are running in parallel (true parallel). This is a major difference between a hardware description language and other programming languages (e.g. C). Within a process, only sequential statements can be used. These statements are:

- Conditional statement (`if – else`, `if – elsif – else`, `case`);
- loops (`for – do`, `while`);
- simple assignments for signals and variables; and
- function calls and procedure calls.

The following examples will demonstrate processes and sequential statements. Please note that it is typical for VHDL that there are several possible ways to describe the same circuit.

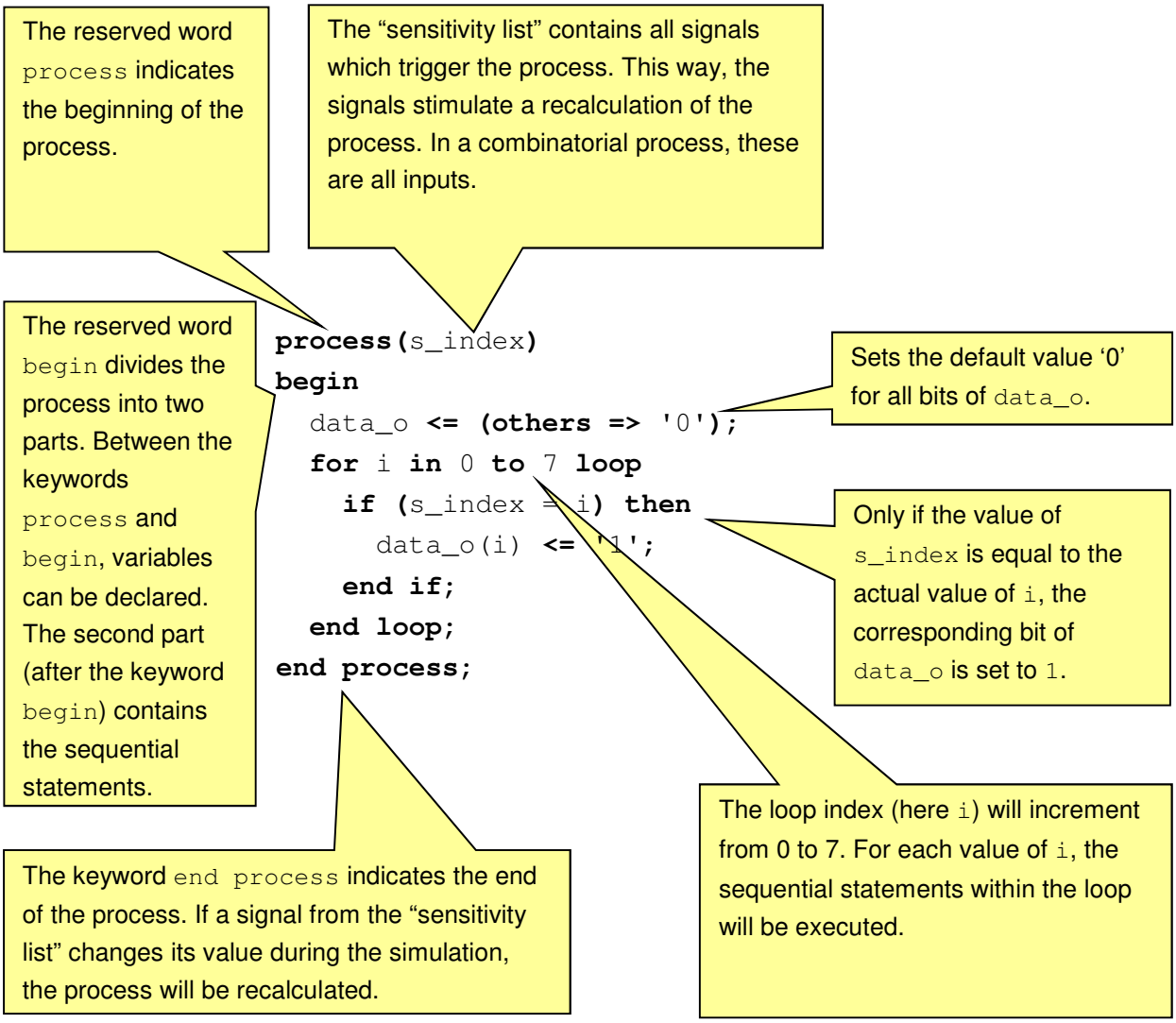


Figure 1: Decoder with `for-loop`

The figure above can be programmed in an easier way. The signal `data_i` is converted to `s_index`.

```
process(data_i)
begin
  s_index <= to_integer(unsigned(data_i));
end process;
```

In order to start the process, the signal `s_index` is used in the "sensitivity list".

```
process(s_index)
begin
  data_o <= "00000000";
  data_o (s_index) <= '1';
end process;
```

Sets all bits of `data_o` to '0'

Finally, the integer value of `s_index` is utilized to set a specific bit of `data_o` to '1'.

Figure 2: Decoder with vector index

Several combinatorial problems can be solved with a simple `case` statement. In this example, the `case` statement supports good readability of the VHDL code.

```
process(data_i)
begin
  case data_i is
    when "000" => data_o <= "00000001";
    when "001" => data_o <= "00000010";
    when "010" => data_o <= "00000100";
    when "011" => data_o <= "00001000";
    when "100" => data_o <= "00010000";
    when "101" => data_o <= "00100000";
    when "110" => data_o <= "01000000";
    when others => data_o <= "10000000";
  end case;
end process;
```

If there are missing binary possibilities within the `when` condition, an `others` branch is absolutely necessary.

In this example, each branch assigns a value to `data_o`. A `when` branch can also contain several statements.

Figure 3: Decoder with `case` statement

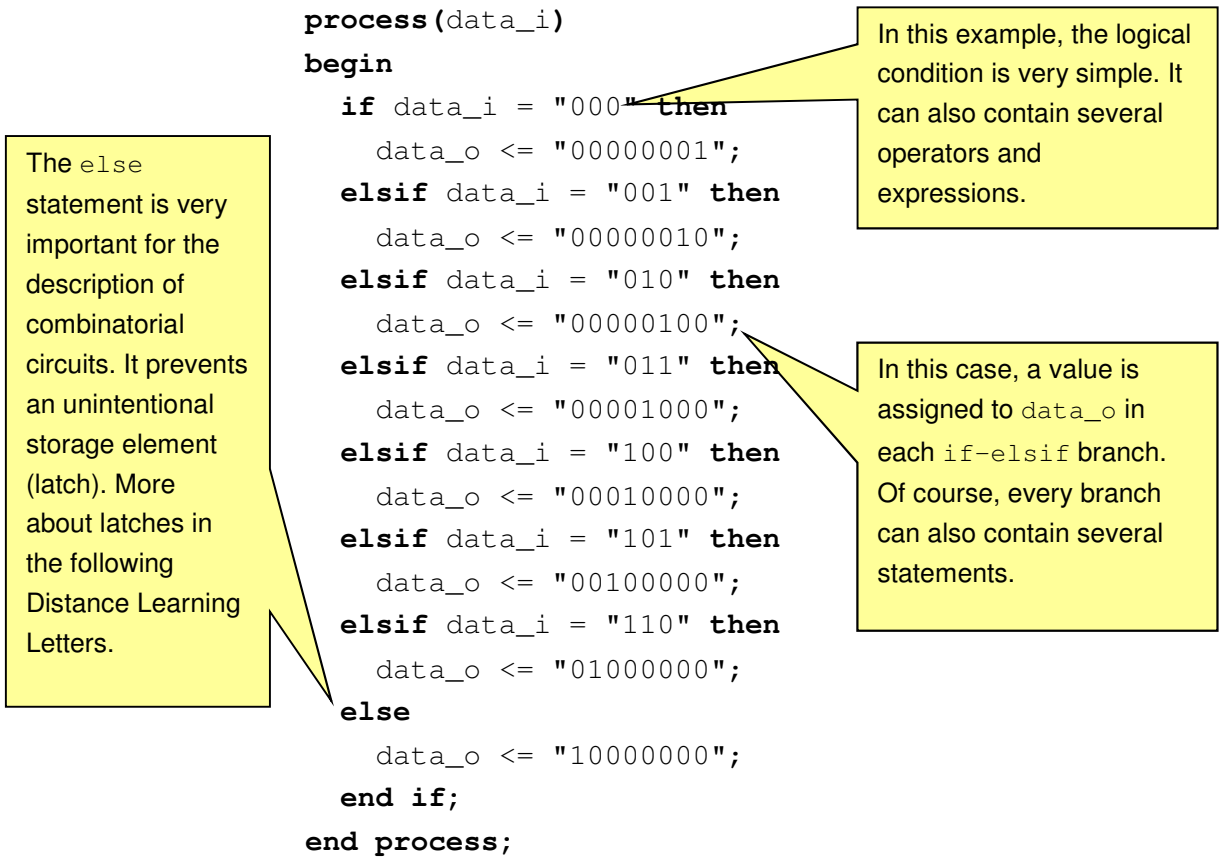


Figure 4: Decoder with `if - elsif` statements

Sometimes, the description with several `if` statements can cause confusion. Sometimes, however, it can have advantages, especially in case of a complex state machine.

All bits of `data_o` are set to the default value '0'. This will prevent an unintentional storage element (latch).

```
process(data_i)
begin
  data_o <= "00000000";
  if data_i = "000" then
    data_o <= "00000001";
  end if;
  if data_i = "001" then
    data_o <= "00000010";
  end if;
  if data_i = "010" then
    data_o <= "00000100";
  end if;
  if data_i = "011" then
    data_o <= "00001000";
  end if;
  if data_i = "100" then
    data_o <= "00010000";
  end if;
  if data_i = "101" then
    data_o <= "00100000";
  end if;
  if data_i = "110" then
    data_o <= "01000000";
  end if;
  if data_i = "111" then
    data_o <= "10000000";
  end if;
end process;
```

Figure 5: Decoder with several `if` statements

Processes can also be named; this name is called label in VHDL.

```
run : process
begin
  data_i <= "000";
  wait for 100 ns;
  data_i <= "001";
  wait for 100 ns;
  data_i <= "010";
  wait for 100 ns;
  data_i <= "011";
  wait for 100 ns;
  data_i <= "100";
  wait for 100 ns;
  data_i <= "101";
  wait for 100 ns;
  data_i <= "110";
  wait for 100 ns;
  data_i <= "111";
  wait for 100 ns;
end process run;
```

This process does not have a sensitivity list. It will therefore run the whole time during simulation and will only be stopped by a `wait` statement. If the process reaches the end, it will start from the beginning.

It is easy to model a signal this way. The modeled signal will be used for the verification of the digital circuit. Every 100 ns, the value of `data_i` changes. Furthermore, the correct functionality can be verified in a simulator.

Figure 6: Process to generate a timed signal

Please note the implied priority of the `if - elsif` statement. Only the first statement which fulfills its condition will be executed. The other statements are not executed.

```
process(data1_i, data2_i)
begin
data_o <= "00000000";
  if data1_i = '1' then
    data_o(0) <= '1';
  elsif data2_i = '1' then
    data_o(1) <= '1';
  end if;
end process;
```

Figure 7: Implied Priority

Concurrent Statements

As described before, the processes defined in an architecture are executed in parallel (true parallel). This is also true for instances (as described in previous Distance Learning Letters). In addition to processes and instances, there are a few other statements which are executed in parallel. The `when` and `with - select` statements will be described in the following examples.

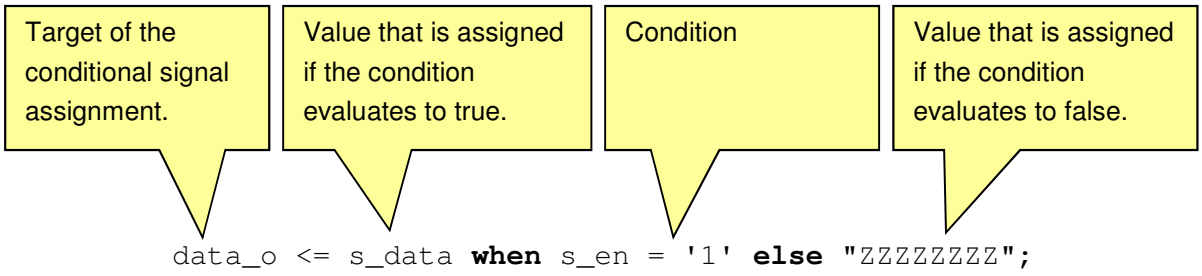


Figure 8: Simple `when` statement

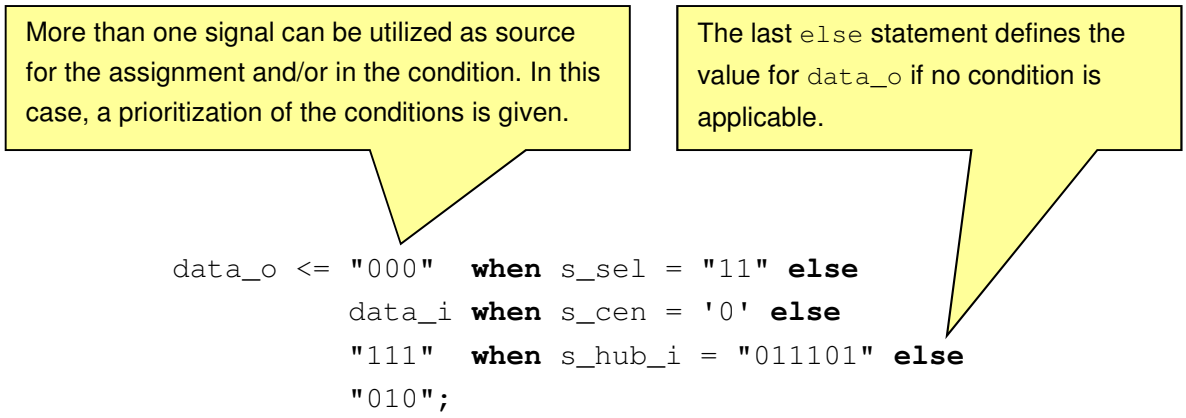


Figure 9: Multiple `when` statements

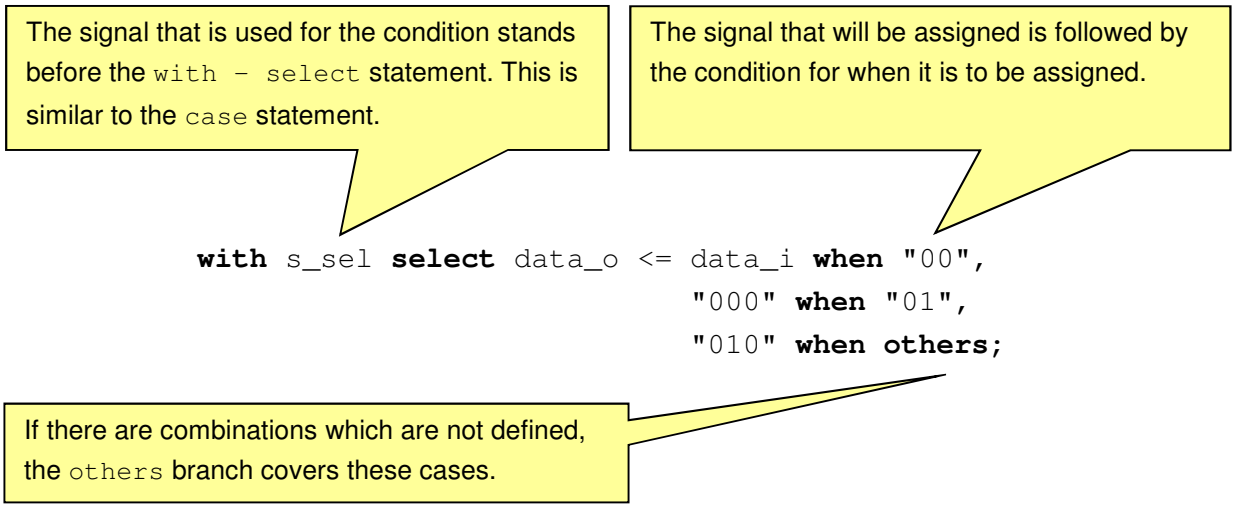


Figure 10: `with - select` statement

Abbreviations

VHDL Very High Speed Integrated Circuit Hardware Description Language

References

There are numerous websites on this topic, e.g. http://www.vhdl-online.de/.

The library of UAS Technikum Vienna has several books about VHDL in English and German. Worth noting are the following:

- Digital Design and Modeling with VHDL and Synthesis, K. C. Chang. Los Alamitos, CA, USA: Wiley – IEEE Computer Soc. Pr., 1997. ISBN 0-8186-7716-3
- Essential VHDL. RTL Synthesis Done Right, Sundar Rajan. USA, 1998. ISBN 0-966959-0-0
- HDL Programming Fundamentals. VHDL and Verilog, N. M. Botros. Hingham, MA, USA. Da Vinci Engineering Press, 2006. ISBN 1-58450-855-8.

Questions

1. Use your knowledge from the previous exercises and scripts to independently describe the following task in VHDL. In order to verify your design, create your own testbench. The testbench shall be loaded and compiled with the ModelSim simulator without errors. Your entity has three inputs and one output of the data type `std_logic_vector`. The block diagram is shown in Figure 11.

Figure 11: Schematic representation of the entity

Import the IEEE library and the `std_logic_1164.all` package. Moreover, describe the dependency of the input signal `sel_i` in order to set the 8-bit output signal `out_o`. The dependency of the input signal shall be described in VHDL with the appropriate constructs of a combinatorial circuit. More details are shown in Figure 11**Fehler! Verweisquelle konnte nicht gefunden werden.**.

sel_i	out_o (MSB ... LSB)
0000	data1_i
0001	data0_i(7), data1_i(7), 000000
1111	1111, data0_i(3 downto 0)
0010	not(data0_i)
0100	data0_i
others	high impedance ("ZZZZZZZZ")

Table 1: Details for Task 1

2. In which way are instructions executed within a VHDL process?
3. Which VHDL statements can be used within a VHDL process?
4. Which branch instructions can be used outside a VHDL process?
5. What is the difference between a VHDL process with and without sensitivity list?
6. How do multiple VHDL processes behave in an architecture?

Version

Version 0.1, 2006-08-21	Create document.
Version 0.2, 2011-06-03	Document translated and updated.
Version 0.3, 2015-08-31	Minor Changes (for example new TW Logo).
Version 0.4, 2016-02-26	Update document.
Version 0.5, 2016-08-17	Proofreading.

If you find errors or inconsistencies, please report them to the supervisors of this lecture via e-mail. Thank you!