Signals & Variables (1A)

Young Won Lim
06/15/2025

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Young Won Lim
06/15/2025

Signal Declaration (1)

Declaration ---- used in ----> Architecture

Syntax

signal signal_name : type;

signal signal_name : type := initial_value;

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Signal Declaration (2)

```
signal SUM, CARRY1, CARRY2 : bit;
signal COUNT : integer range 0 to 15;
signal CLK, RESET : std_ulogic := '0';
signal ALARM_TIME : T_CLOCK_TIME := (1,2,0,0);
signal CONDITION : boolean := false;
```

During elaboration, eacg signal is set to an initial value. If a signal is not given an explicit initial value, it will default to the leftmost value ('left) of its declared type:

```
signal I : integer range 0 to 3;
-- I will initialise to 0
signal X : std_logic;
-- X will initialise to 'U'
```

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Signal Declaration (3)

A signal which is driven by more than one process, concurrentstatement or component instance, must be declared with a resolved type, e.g.std_logic or std_logic_vector:

```
architecture COND of TRI_STATE is
  signal TRI_BIT: std_logic;
begin
  TRI_BIT <= BIT_1 when EN_1 = '1'
        else 'Z';
  TRI_BIT <= BIT_2 when EN_2 = '1'
        else 'Z';
end COND;
```

Signals may not be declared in a processor subprogram (except as formal parameters).

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Signal Declaration (4)

Ports declared in an entity are accessible as signals within the associated architecture(s) and do not need to be redeclared.

A signal of a resolved type may be declared as a guarded resolved signal. This is required if all drivers to a signal may be turned off, through guarded assignments.

signal signal_name : resolved_type signal_kind;

The "signal kind" keyword may be register or bus. Guarded resolved signals of kind register retain their current value when drive is turned off, whereas signals of kind bus rely on the resolution function to provide a "no-drive" value.

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Signal Declaration (5)

Synthesis Issues

Signals are supported for synthesis, providing they are of a type acceptable to the logic synthesis tool.

The signal kinds register of bus are usually ignored.

Only certain resolved signal types are supported. Most tools recognise the std_logic_1164 types.

Whats New in '93

Signal Declarations have not changed in VHDL-93.

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration (1)

Declaration ---- used in ----> Process
Procedure
Function

Syntax

variable variable_name : type;

variable variable_name : type := initial_value;

See LRM section 4.3.1.3

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration (2)

Rules and Examples

```
variable HEIGHT : integer := 8;
variable COND : boolean := true;
variable IN_STRING : string(1 to 80);
variable M,N : bit := '1';
variable I : integer range 0 to 3;
variable MAKE_FRAME_STATE :
   T_MAKE_FRAME_STATE := RCV_HIGH;
```

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration (3)

A Variable may be given an explicit initial value when it is declared. If a variable is not given an explicit value, it's default value will be the leftmost value ('left) of its declared type.

```
variable I : integer range 0 to 3;
-- initial value of I is 0
variable X : std_ulogic;
-- initial value of X is 'U'
```

Variables within subprograms (functions and procedures) are initialised each time the subprogram is called:

```
function PARITY (X : std_ulogic_vector)
                 return std_ulogic is
  variable TMP : std_ulogic := '0';
begin
  for J in X'range loop
    TMP := TMP xor X(J);
  end loop; --no need to initialise TMP
  return TMP;
end PARITY;
```

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration

Variables in processes, except for "FOR LOOP" variables, receive their initial values at the start of the simulation time (time = 0 ns)

```
process (A)
   variable TMP : std_ulogic := '0';
begin
   TMP := '0';
   -- in this example we need to reset
   -- TMP to '0' each time the process
   -- is activated
   for I in A'low to A'high loop
      TMP := TMP xor A(I);
   end loop;
   ODD <= TMP;
end process;
```

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration

Synthesis Issues

Variables are supported for synthesis, providing they are of a type acceptable to the logic synthesis tool.

In a "clocked process", each variable which has its value read before it has had an assignment to it will be synthesised as the output of a register.

In a "combinational process", reading a variable before it has had an assignment may cause a latch to be synthesised.

Variables declared in a subprogram are synthesised as combinational logic.

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Variable Declaration

Whats New in '93

In VHDL-93, shared variables may be declared within an architecture, block, generate statement, or package:

shared variable variable_name : type;

Shared variables may be accessed by more than one process. However, the language does not define what happens if two or more processes make conflicting accesses to a shared variable at the same time.

https://ics.uci.edu/~jmoorkan/vhdlref/var_dec.html

Signals and Variables (I)

Variables are used when you want to create serialized code, unlike the normal parallel code. (Serialized means that the commands are executed in their order, one after the other instead of together). A variable can exist only inside a process, and the assignment of values is not parallel. For example, consider the following code:

```
signal a,b : std_logic_vector(0 to 4);

process (CLK)
    begin
        if (rising_edge(clk)) then
            a <= '11111';
            b <= a;
        end if;
end process;
```

will put into b the value of a before the process ran, and not '11111'.

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

```
signal a,b : std_logic_vector(0 to 4);

process (CLK)
    variable var : std_logic_vector(0 to 4);
    begin
        if (rising_edge(clk)) then
            var := '11111';
            a <= var;
            b <= var;
        end if;
end process;
```

will put the value '11111' into both a and b.

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

Variables are intended to be a used for storing a value within a process. As such It's scope is limited. There tends to be a less direct relationship to synthesized hardware.

Variables also get a value immediately, whereas signals don't. the following two processes have the same effect:

```
signal IP, NEXTP : STD_LOGIC_VECTOR(0 to 5);

process (CLK)
   Variable TEMP : STD_LOGIC_VECTOR(0 to 5);
   begin
      if (rising_edge(clk)) then
         TEMP := IP;
         IP <= NEXTP;
         NEXTP <= TEMP(5) & TEMP(0 to 4);
      end if;
end process;
```

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

```
signal IP, NEXTP : STD_LOGIC_VECTOR(0 to 5);

process (CLK)

    begin
        if (rising_edge(clk)) then
            IP <= NEXTP;
            NEXTP <= IP(5) & IP(0 to 4);
        end if;
end process;
```

This is because the updates get scheduled, but haven't actually changed yet. the <= includes a temporal element.

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

variables: Temporary location; they are used to store intermediate values within "process".

signals: Update signal values. Run process activated by changes on signal.While process is running all signals in system remain unchanged.

Differences:

variables: They are local; no delay; declared within process

signals: They are global (before begin); delay due to wire; declared before key word begin

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

On a side note variables can't just live in processes (but also e.g. in procedures), furthermore they can be shared variables accessible from multiple processes

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (I)

Variables - they are local to a process, their value is updated as soon as the variable gets a new value.

Shared Variables- are like variables but they can be accessed from different processes.

Signals- Their scope is bigger, every process can access signals declared in the architecture or a specific block(if there is). There value updates after the process is suspended or encounters a wait statement.

https://stackoverflow.com/questions/15485749/vhdl-variable-vs-signal

Signals and Variables (II)

Every programming language has objects for storing values. VHDL too have them. Two of these object types are called Signals and Variables. They might look very similar for a beginner, but there are few fundamental differences between them.

Variables are assigned using the := operator. And signals are assigned with the <= operator.

Variables can be declared and used only within a process/function/procedure but Signals can be declared and used anywhere.

https://vhdlguru.blogspot.com/2020/11/signals-and-variables-in-vhdl.html

Signals and Variables (II)

In a block of statements, the statements with variables immediately take their values. Very similar to how it works in programming languages like C. But in a group of statements with Signals on the left hand side, the signals does not take it's new value until the process has suspended (either hit the bottom or hit a wait statement).

This can be further explained with the following example scenario.

Suppose I want to implement a swapping function in VHDL using Signals.

I can simply write,

```
signal x,y : std_logic := 0;

process(Clk)
begin
if(rising_edge(Clk)) then
  x <= y;
  y <= x;
end if;
end process;
```

https://vhdlguru.blogspot.com/2020/11/signals-and-variables-in-vhdl.html

Signals and Variables (II)

What happens above is that, though the 'x' is assigned the value of 'y' sequentially first, the new value isn't updated to 'x' until we "exit" the process. So from a practical point of looking at it, it looks like they happen in parallel.

Now if I have to use variables for implementing a swapping function, I need three statements. Like below:

```
process(Clk)
variable x,y,temp : std_logic := 0;
begin
if(rising_edge(Clk)) then
  temp := x;
  x := y;
  y := x;
end if;
end process;
```

https://vhdlguru.blogspot.com/2020/11/signals-and-variables-in-vhdl.html

Signals and Variables (II)

Since variables take the values assigned to them right away, we need a temporary variable to hold the value of 'x' before assigning 'y' to it.

Variables declared in different processes cannot communicate with each other. They are local to the process. On the other hand signals declared in a VHDL entity can be used anywhere in the entity.

You cannot declare or use a Signal inside a VHDL Function. Functions are purely combinatorial in VHDL and thus you have to have use variables.

If you want the code to be synthesised, then beware of the consequences of using a variable. Variables often create latches when implemented on a FPGA and synthesis tools often pass a warning to notify. If not needed its good to avoid latches in your design.

Though using variables might seem make the work easier, it might not pass the synthesis stage. For many, who come to VHDL from a C background, using variables is very tempting.

https://vhdlguru.blogspot.com/2020/11/signals-and-variables-in-vhdl.html

Signals and Variables (II)

Check this Matrix Multiplication code using Variables to see some of the dangers involved with them. Multiplication of two matrices requires a large number of multipliers and adders. In C, you would use some nested "for" loops to achieve this. And with the use of variables you can do the same thing in VHDL too like you can see from the link.

But using this same piece of code in a real FPGA is impossible to achieve. Either the design wont pass the synthesis stage or it will take days to get it done.

All those individual additions and multiplications gets done in "one" clock cycle. None of the adders and multipliers get reused and the loops get unfolded into a concatenated series of resources.

In such a case its necessary to use signals and spilt the whole operation over many clock cycles. This reduces the resource usage and more importantly you have a chance to get your design synthesised.

https://vhdlguru.blogspot.com/2020/11/signals-and-variables-in-vhdl.html

Variable

The previous article in this series discussed that sequential statements allow us to describe a digital system in a more intuitive way. Variables are useful objects that can further facilitate the behavioral description of a circuit. This article will discuss the important features of variables. Several examples will be discussed to clarify the differences between variables and signals. Let’s first review VHDL signals.

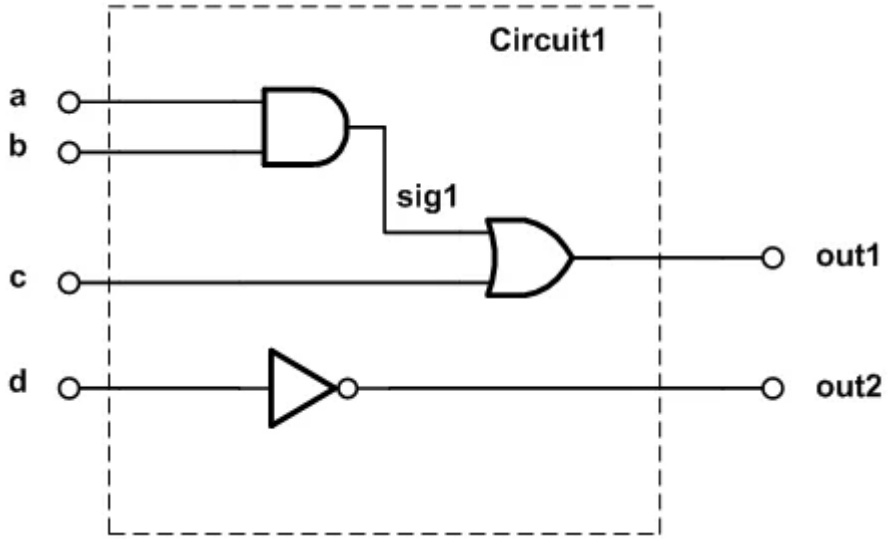
https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

```
1     architecture Behavioral of circuit1 is
2           signal sig1: std_logic;
3     begin
4           sig1 <= ( a and b );
5           out1 <= ( sig1 or c );
6           out2 <= (not d);

7     end Behavioral;
```

As you can see, a signal has a clear mapping into hardwar
make sense to have multiple assignments to a signal? For

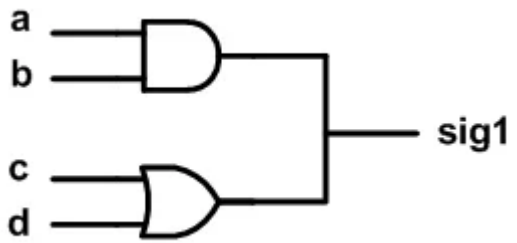
```
sig1 <= ( a and b );
sig1 <= (c or d);
```



https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

If these two assignments are in the concurrent part of the code, then they are executed simultaneously.
We can consider the equivalent hardware of the above code as shown in Figure 2.



https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Figure 2 suggests that multiple assignments to a signal in the concurrent part of the code is not a good idea because there can be a conflict between these assignments. For example, if A=C=0 and B=D=1, the first line would assign sig1 = (0 and 1) =0, while the second would attempt to assign sig1 = (0 or 1) = 1. That's why, in the concurrent part of the code, VHDL doesn't allow multiple assignments to a signal. What if these two assignments were in the sequential part of the code? A compiler may accept multiple assignments inside a process but, even in this case, only the last assignment will survive and the previous ones will be ignored. To explain this, note that a process can be thought of as a black box whose inner operation may be given by some abstract behaviour description. This description uses sequential statements. The connection between the process black box and the outside world is achieved through the signals. The process may read the value of these signals or assign a value to them. So VHDL uses signals to connect the sequential part of the code to the concurrent domain. Since a signal is connected to the concurrent domain of the code, it doesn't make sense to assign multiple values to the same signal. That's why, when facing multiple assignments to a signal, VHDL considers only the last assignment as the valid assignment.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

The black box interpretation of a process reveals another important property of a signal assignment inside a process: When we assign a value to a signal inside a process, the new value of the signal won't be available immediately. The value of the signal will be updated only after the conclusion of the current process run. The following example further clarifies this point. This example uses the VHDL "if" statements. Please note that we'll see more examples of this statement in future articles; however, since it is similar to the conditional structures of other programming languages, the following code should be readily understood. You can find a brief description of this statement in a previous article.

Example: Write the VHDL code for a counter which counts from 0 to 5.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

One possible VHDL description is given below:

```
1       library IEEE;
2       use IEEE.STD_LOGIC_1164.ALL;

3       entity SigCounter is
4           Port ( clk : in  STD_LOGIC;
5                  out1: out integer range 0 to 5);
6       end SigCounter;

7       architecture Behavioral of SigCounter is

8              signal sig1 : integer range 0 to 6;

9       begin
```

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

```
10          process(clk)
11          begin
12          if (clk'event and clk='1') then
13                  sig1 <= sig1+1;
14                  if (sig1=6) then
15                          sig1 <= 0;
16                  end if;
17          end if;
18          out1 <= sig1;
19          end process;

20    end Behavioral;
```

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

In this example, sig1 is defined as a signal of type integer in the declarative part of the architecture. With each rising edge of clk, the value of the signal sig1 will increase by one. When sig1 reaches 6, the condition of the “if” statement in line 14 will be evaluated as true and sig1 will take the value zero. So it seems that sig1, whose value is eventually passed to the output port out1, will always take the values in the range 0 to 5. In other words, it seems that the “if” statement of line 14 will never let sig1 take the value 6. Let’s examine the operation of the code more closely.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

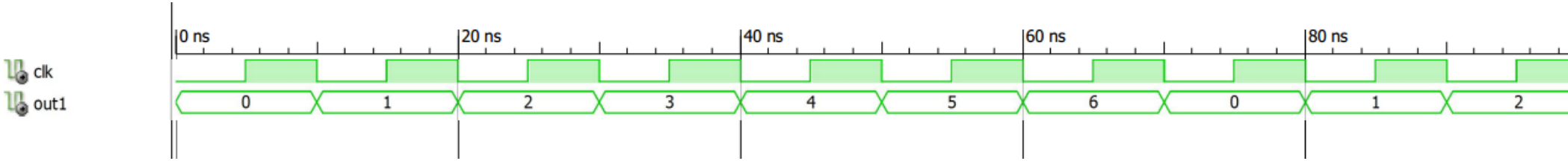
Variable

Assume that a previous run of the process sets sig1 to 5. With the next rising edge of clk, the statements inside the “if” statement of line 12 will be executed. Line 13 will add one to the current value of sig1, which is 5, and assign the result to sig1. Hence, the new value of sig1 will be 6; however, we should note that the value of the signal sig1 will be updated only after the conclusion of the current process run. As a result, in this run of the process, the condition of the “if” statement in line 14 will be evaluated as false and the corresponding “then” branch will be bypassed. Reaching the end of the process body, the value of sig1 will be updated to 6. While we intended sig1 to be in the range 0 to 5, it can take the value 6!

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Similarly, at the next rising edge of clk, line 13 will assign 7 to sig1. However, the signal value update will be postponed until we reach the end of the process body. In this run of the process, the condition of the “if” statement in line 14 returns true and, hence, line 15 will set sig1 to zero. As you see, in this run of the process, there are two assignments to the same signal. Based on the discussion of the previous section, only the last assignment will take effect, i.e. the new value of sig1 will be zero. Reaching the end of this process run, sig1 will take this new value. As you see, sig1 will take the values in the range from 0 to 6 rather than from 0 to 5! You can verify this in the following ISE simulation of the code.



https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Hence, when using signals inside a process, we should note that the new value of a signal will be available at the end of the current run of the process. Not paying attention to this property is a common source of mistake particularly for those who are new to VHDL.

To summarize our discussion so far, a signal models the circuit interconnections. If we assign multiple values to a signal inside a process, only the last assignment will be considered. Moreover, the assigned value will be available at the end of the process run and the updates are not immediate.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

As discussed in a previous article, sequential statements allow us to have an algorithmic description of a circuit. The code of such descriptions is somehow similar to the code written by a computer programming language. In computer programming, “variables” are used to store information to be referenced and used by programs. With variables, we can more easily describe an algorithm when writing a computer program. That’s why, in addition to signals, VHDL allows us to use variables inside a process. While both signals and variables can be used to represent a value, they have several differences. A variable is not necessarily mapped into a single interconnection. Besides, we can assign several values to a variable and the new value update is immediate. In the rest of the article, we will explain these properties in more detail.

Before proceeding, note that variables can be declared only in a sequential unit such as a process (the only exception is a “shared” variable which is not discussed in this article). To get more comfortable with VHDL variables, consider the following code segment which defines variable var1.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

```
1       process(clk)
2               variable var1 : integer range 0 to 5;
3       begin
4       var1 := 3;
5       ...
6       end process;
```

Similar to a signal, a variable can be of any data type (see the previous articles in this series to learn more about different data types). However, variables are local to a process. They are used to store the intermediate values and cannot be accessed outside of the process. Moreover, as shown by line 4 of the above code, the assignment to a variable uses the “:=” notation, whereas, the signal assignment uses “<=”.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Multiple Assignments to a Variable

Consider the following code. In this case, a variable, var1, of type std_logic is defined. Then in lines 12, 13, and 14, three values are assigned to this variable.

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;

3     entity VarTest1 is
4         Port ( in1, in2, in3 : in  STD_LOGIC;
5                out1 : out  STD_LOGIC);
6     end VarTest1;
```

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

```
7     architecture Behavioral of VarTest1 is

8     begin

9           process(in1, in2, in3)
10          variable var1: std_logic;
11          begin
12                var1 := in1;
13                var1 := (var1 and in2);
14                var1 := (var1 or in3);
15                out1 <= var1;
16          end process;

17    end Behavioral;
```

Figure 4 shows the RTL schematic of the above code which is generated by Xilinx ISE.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

It’s easy to verify that the produced schematic matches the behavior described in the process; however, this example shows that mapping variables into the hardware is somehow more complicated than that of signals. This is due to the fact that the sequential statements describe the behavior of a circuit. As you can see, in this example, each variable assignment operation of lines 13 and 14 have created a different wire though both of these two assignments use the same variable name, i.e. var1.

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Variables are updated immediately. To examine this, we'll modify the code of the above counter and use a variable instead of a signal. The code is given below:

```
1    library IEEE;
2    use IEEE.STD_LOGIC_1164.ALL;

3    entity VarCounter is
4        Port ( clk : in  STD_LOGIC;
5               out1: out integer range 0 to 5);
6    end VarCounter;
```

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

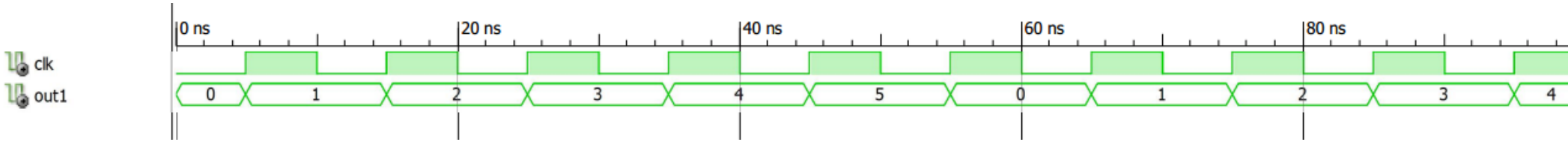
Variable

```
7   architecture Behavioral of VarCounter is

8   begin

9       process(clk)
10      variable var1 : integer range 0 to 6;
11      begin
12      if (clk'event and clk='1') then
13          var1 := var1+1;
14          if (var1=6) then
15              var1 := 0;
16          end if;
17      end if;
18      out1 <= var1;
19      end process;

20  end Behavioral;
```

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

Since the new value of a variable is immediately available, the output will be in the range 0 to 5. This is shown in the following ISE simulation result.



https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Variable

A signal models the circuit interconnections. If we assign multiple values to a signal inside a process, only the last assignment will be considered. Moreover, the assigned value will be available at the end of the current process run and the updates are not immediate.

A single variable can produce several circuit interconnections.

We can assign multiple values to the same variable and the assigned new values will take effect immediately.

Similar to a signal, a variable can be of any data type.

Variables are local to a process. They are used to store the intermediate values and cannot be accessed outside of the process.

The assignment to a variable uses the ":=" notation, whereas, the signal assignment uses "<=".

https://www.allaboutcircuits.com/technical-articles/variable-valuable-object-in-sequential-vhdl/

Internal Signals

Internal signals

Shown below is a second architecture V2 of AOI (remember that the architecture name V2 is completely arbitrary - this architecture is called V2 to distinguish it from the earlier architecture V1). Architecture V2 describes the AOI function by breaking it down into the constituent boolean operations. Each operation is described within a separate concurrent signal assignment. In hardware terms we can think of each assignment as a die in a hybrid package or a multi-chip module. The signals are the bonding wires or substrate traces between each die.

VHDL: Internal signals of an AOI gate

https://www.doulos.com/knowhow/vhdl/internal-signals/

Internal Signals

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity AOI is
  port (A, B, C, D: in STD_LOGIC;
  F : out STD_LOGIC);
end AOI;

architecture V2 of AOI is
  signal AB, CD, O: STD_LOGIC;
begin
  AB <= A and B after 2 NS;
  CD <= C and D after 2 NS;
  O <= AB or CD after 2 NS;
  F <= not O after 1 NS;
end V2;
```

https://www.doulos.com/knowhow/vhdl/internal-signals/

Internal Signals

Signals

The architecture contains three signals AB, CD and O, used internally within the architecture. A signal is declared before the begin of an architecture, and has its own data type (eg. STD_LOGIC). Technically, ports are signals, so signals and ports are read and assigned in the same way.

Assignments

The assignments within the architecture are concurrent signal assignments. Such assignments execute whenever a signal on the right hand side of the assignment changes value. Because of this, the order in which concurrent assignments are written has no effect on their execution. The assignments are concurrent because potentially two assignments could execute at the same time (if two inputs changed simultaneously). The style of description that uses only concurrent assignments is sometimes termed dataflow.

https://www.doulos.com/knowhow/vhdl/internal-signals/

Internal Signals

Delays

Each of the concurrent signal assignments has a delay. The expression on the right hand side is evaluated whenever a signal on the right hand side changes value, and the signal on the left hand side of the assignment is updated with the new value after the given delay. In this case, a change on the port A would propagate through the AOI entity to the port F, with a total delay of 5 NS.

https://www.doulos.com/knowhow/vhdl/internal-signals/

References

[1] http://en.wikipedia.org/
[2] J. V. Spiegel, VHDL Tutorial,
http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html
[3] J. R. Armstrong, F. G. Gray, Structured Logic Design with VHDL
[4] Z. Navabi, VHDL Analysis and Modeling of Digital Systems
[5] D. Smith, HDL Chip Design
[6] http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html
[7] VHDL Tutorial - VHDL onlinewww.vhdl-online.de/tutorial/