Sequential Signal Assignment (4A)

Young Won Lim
06/12/2025

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

This document was produced by using OpenOffice and Octave.

Young Won Lim
06/12/2025

Sequential Signal Assignment Statement

```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

Within a process, statements are indeed carried out sequentially.

However, values assigned to signals are not carried out immediately but scheduled to occur at the end of the process.

when the assignment **c** <= **a**; is carried out, the value of **a** is still the value a had at the start of the process.

This is because the assignment **a** <= **b**; has not yet been carried out and **a** has not changed.

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Multiple Assignment

```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

In fact, the assignment **a <= b**;
will never be carried out
because of the fourth assignment **a <= c**;,
which will be scheduled to occur
at the process end instead.

This behaviour reflects the behaviour of real logic circuitry, which is what VHDL was designed to do.

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Think reverse order

```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

try to read statements in reverse order:

a <= **c** -- the **a** register gets whatever was in the **c** register
c <= **a** -- the **c** register gets whatever was in the **a** register
b <= **c** -- the **b** register gets whatever was in the **c** register
a <= **b** -- this statement is ignored.

no two drivers for a single FF input

multiple assignments to a signal in a process statement,
only the last assigned value is considered

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Synthesis Results (1)

```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

b_reg and a_reg are exactly the same
(i.e. they each clock the same signal in and out).

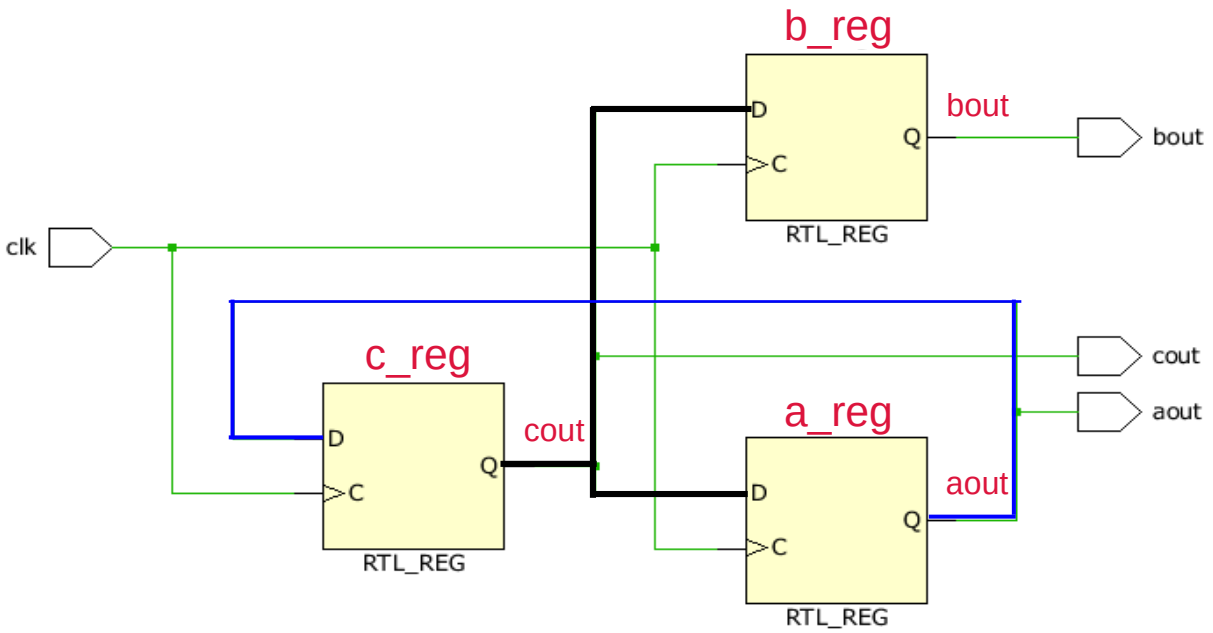
If this simple example were to be synthesized,
one of them would almost certainly be removed,
and bout would be tied to aout.

In fact, it took some keep attributes just to tell Vivado
not to eliminate the b_reg for the RTL.

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Synthesis Results (2)

```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```



synthesized results by Vivado

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Simulation Results (1)

In VHDL, statements in process
are executed sequentially.
a, **b**, **c** and **d** are signals

a <= **b**;
c <= **a**;

At the end of the process
old value of **b** assigned to **a**. and
old value of **a** assigned to **c**.

statements in process executed sequentially,
but the signals don't get the new values
before end of the process.

the initial value of the signal before rising edge of clock.

a=1; **b**=1; **c**=0;

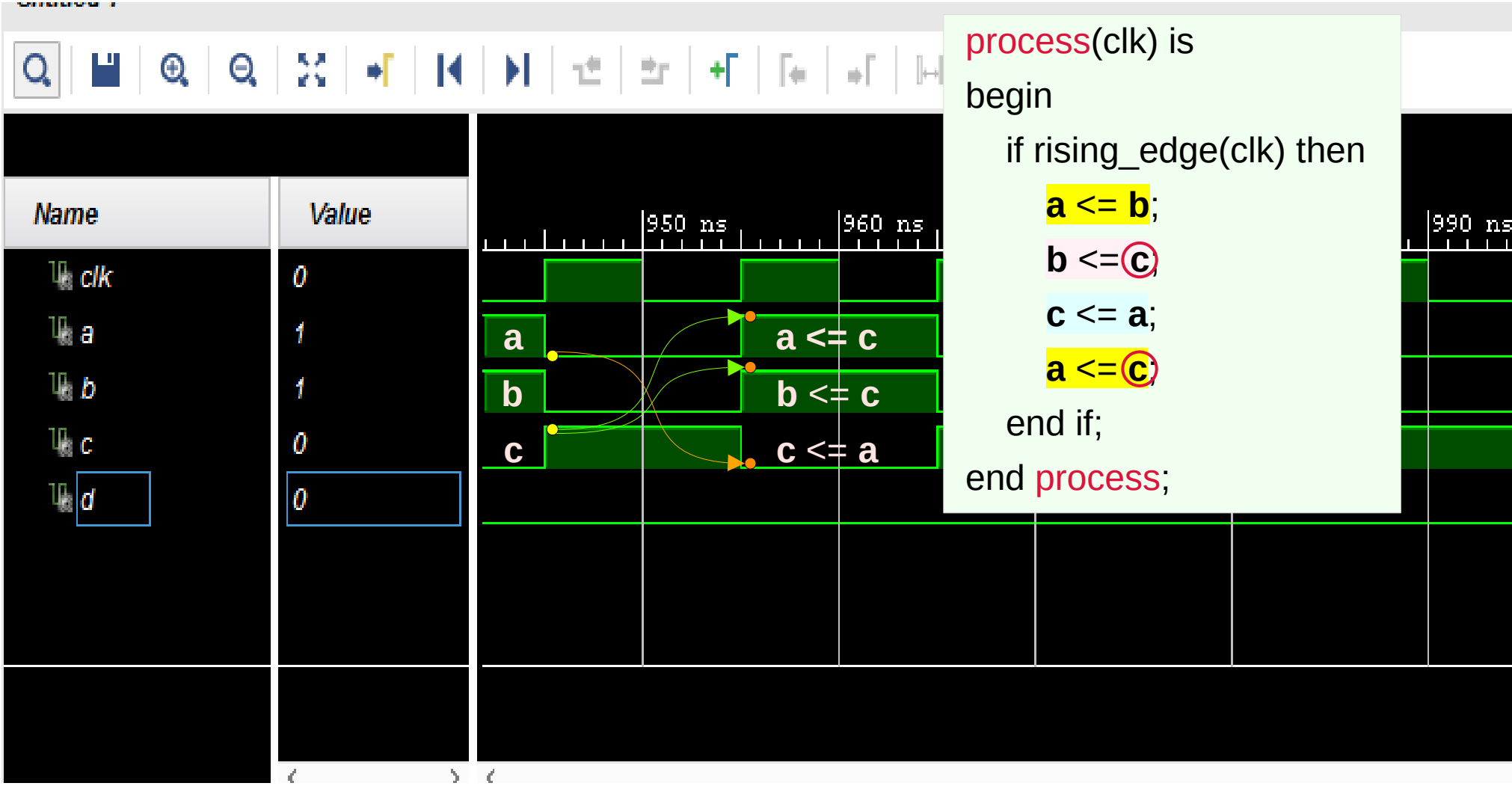
In the rising edge of clock,
the statements in the process executed.
so at the end of process
the new value of signals are:

a = old **c** = 0;
b = old **c** = 0;
c = old **a** = 1;

```
process(clk) is
begin
   if rising_edge(clk) then
      a <= b;
      b <= c;
      c <= a;
      a <= c;
   end if;
end process;
```

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Simulation Results (2)




```
process(clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Sequential Statement

https://electronics.stackexchange.com/qauestions/372181/process-statements-and-sequential-execution-in-vhdl

Updating signal values and running processes

signals represent the interface
between the concurrent domain of a VHDL model
and the sequential domain within a process.

A VHDL model is composed of several processes
that communicate via signals.

At simulation, all hierarchy of the model is removed and
only the processes and signals remain.

The simulator alternates
between updating signal values and then
running processes activated by changes
of the signals listed in their sensitivity lists

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

Signal Assignment

Signal assignment may be performed
using a sequential statement or a concurrent statement.

The sequential statement may
only appear inside a process,
while the concurrent statement may
only appear outside processes.

The sequential signal assignment has a single form,
the simple one, which is an unconditional assignment.

The concurrent signal assignment has,
in addition to its simple form, two other forms:
the conditional assignment and
the selective assignment .

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

Sequential Signal Assignment Statement

The sequential signal assignment has the same syntax
as the simple form of the concurrent signal assignment;

the difference between them results from context.

the sequential signal assignment has the following syntax:

signal <= expression [after delay];

as a result of executing this statement in a process,
the expression on the right-hand side of the assignment symbol
is evaluated and an event is scheduled to change the value of the signal.

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

When the process suspends

The simulator will only change the value of a signal
when the process suspends, and,
if the after clause is used,
after the delay specified from the current time.

Therefore, in a process the signals will be updated
only after executing all the statements of the process or
when a wait statement is encountered.

Typically, synthesis tools do not allow to use after clauses,
or they ignore these clauses.

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

Delay and synthesis

The after clauses are ignored not only because
their interpretation for synthesis is not specified by standards,
but also because it would be difficult to guarantee the results of such delays.

For example, it is not clear whether the delay
should be interpreted as a minimum or maximum propagation delay.

Also, it is not clear how the synthesis tool should proceed
if a delay specified in the source code cannot be assured

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

Multiple assignments

A consequence of the way in which
signal assignments within processes are executed is that
when more than one value is assigned to the same signal,
only the last assignment will be effective.

Thus, the two processes are equivalent.

```
proc7: process (a)
begin
        z <= '0';
        z <= a;
end process proc7;
```

```
proc8: process (a)
begin
z <= a;
end process proc8;
```

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

When the process suspends

In conclusion, the following important aspects
should be taken into consideration
when signal assignment statements are used inside processes:

- Any signal assignment becomes effective
  only when the process suspends.
  Until that moment, all signals keep their old values.

- Only the *last assignment* to a signal
  will be effectively executed.
  Therefore, it would make no sense
  to assign more than one value to a signal
  in the same process

https://cse.usf.edu/~haozheng/teach/cda4253/doc/vhdl-stmt.pdf

Zero time

No event will ever occur while a process is running!

When a process is woken by an event,
it runs to completion ("end process")
or an explicit "wait" statement,
and goes to sleep.

This takes, notionally, ZERO time.

if you have loops in your process,
they are effectively unrolled completely, and
when you synthesise, you will generate enough hardware
to run EVERY iteration in parallel.

Also, any procedures, functions etc, take zero time
- unless they contained an explicit "wait" statement

(in which case the process suspends at the "wait",
as if the procedure had been inlined).

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

No event when a process is running

No event will ever occur while a process is running!

When a process is woken by an event,
it runs to completion ("end process")
or an explicit "wait" statement,
and goes to sleep.

This takes, notionally, ZERO time.

Throughout this process,
all signals have the value
they originally had when the process woke up,
and any signal assignments are stored up,
to happen later.

Variables update immediately;
later statements in the process see the new value).

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Restarting a process

When the process suspends (at "wait" or "end process"),
nothing happens until ALL the other processes also suspend.

(But remember they all take zero time!).

If a process suspends at "end process"
it will restart from the beginning
when its sensitivity list wakes it up.

If it suspends at an explicit "wait",
that "wait" will specify an event or future time,
which will restart it after the "Wait".

NOTES:
1 : do not mix the sensitivity list and Wait styles in the same process!
2: Wait Until some event is synthesisable (though some tools may object) ;
Wait for some time is simulation only

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Performing all the signal assignments

THEN all the signal assignments are performed.

Since all processes are asleep,
this eliminates all race conditions and timing hazards.

Some of these assignments (like '1' to a clock)
will cause events to be scheduled on processes sensitive to them.

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Delta cycle

After all the signal assignments are done,
the time steps forward one infinitely short tick (called a delta cycle),
and then all the processes with scheduled events are woken.

This continues until a delta cycle occurs
in which NO new events are scheduled, and
finally the simulation can advance by a real time step.

```
process(clk)
begin
if rising_edge(clk) then
   A <= B;
   B <= A;
end if;
end process;
```

is hazard-free in VHDL.

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Verilog

If you ever need to use Verilog,
be aware that some of this happens differently there,
and you cannot rely on the same level of predictability in simulation results.

In synthesis, of course, we generate hardware
which will take some real time to execute this process.

However, the synthesis and back-end tools (place and route)
guarantee to either obey this model faithfully,
or fail and report why they failed.

For example, they will add up all the real delays and
verify that the sum is less than your specified clock period.
(Unless you have set the clock speed too high!).

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Zero time

So the upshot is, as long as the tools report success
(and you are setting the timing constraints like clock speed correctly)
you can pretend the above "zero time" model is true,
and the real hardware behaviour will match the simulation.
Guaranteed, barring tool bugs!

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Reentrant?

When starting out using VHDL (or any other HDL for that matter),
it is hugely important to discard all notions of sequential code,
and instead focus on the flow of data through the hardware.

In hardware, everything is inherently parallel
(everything happens simultaneously),
but uses constantly changing data (input signals)
to calculate constantly changing results (output signals)!

Without going into more advanced topics
such as variables, wait commands etc.,
everything within a process happens simultaneously.

If conflicting things occur within the same process
(multiple writes to the same signal),
the last statement in the process wins,
which is often where confusion about "sequential" code in VHDL comes from.

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Delta

This works due to the way that values are assigned to signals.

When assigning a value to a signal,
the value of the signal does not immediately change!

Instead, the assigned value is remembered and
will be committed as the actual signal value later
(in preparation for the next delta cycle,
which is effectively the next quantum of time).

Since the next delta cycle will not begin
until all processes from the previous delta cycle have completed,
signal values will only change when no process is running.

Once all signals have changed, the next delta cycle begins
and any process sensitive to one of the changed signals will be executed.

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

Combinatorial loop

If a process is sensitive to a signal it also writes,
you have what is known as a combinatorial loop,
e.g., a gate where the output feeds an input.

This is (almost) always an error in your circuit,
and will usually cause simulators to enter an infinite delta-cycle loop.

https://stackoverflow.com/questions/13954193/is-process-in-vhdl-reentrant/

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/