

ENCS 533 – Advanced Digital Design

Lecture 4

How VHDL deals with time

1 Introduction

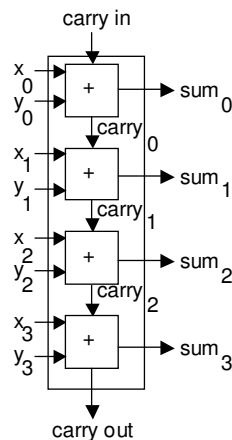
VHDL descriptions of designs, whether behavioural or structural, must be simulated in order to confirm that they behave as required. Simulation allows us to apply inputs, and then trace how the rest of the circuit evolves with time as the influence of the new inputs propagates through towards the outputs. This means that VHDL must be able to deal how signals change with time. In this lecture we will look in depth at how VHDL treats time.

2 Synthesis and modelling in VHDL

Although we haven't highlighted the fact, the code listings that we have looked at have been quite different in their intent. For example

```
c <= a + b;
```

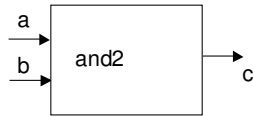
is intended as the input to a synthesis tool. We are saying what we want the design to do, and the synthesis tool will come up with some interconnection of gates that will do this function. By contrast, this is a structural description.



```
ARCHITECTURE structural OF adder IS
    SIGNAL carry: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    g0:  entity work.fulladd(structural)
        PORT MAP (x(0),y(0),cin,sum(0),carry(0));
    g1:  entity work.fulladd(structural)
        PORT MAP (x(1),y(1),carry(0),sum(1),carry(1));
    g2:  entity work.fulladd(structural)
        PORT MAP (x(2),y(2),carry(1),sum(2),carry(2));
    g3:  entity work.fulladd(structural)
        PORT MAP (x(3),y(3),carry(2),sum(3),cout);
END ARCHITECTURE structural;
```

It is the sort of thing that a synthesis tool would hand back to us after it had finished synthesis of the behavioural description of addition.

Although we didn't flag the fact at the time, we've also seen another type of description. For example, what is this?



```
ARCHITECTURE simple OF and2 IS
BEGIN
    c <= a AND b;
END ARCHITECTURE simple;
```

It isn't a structural description, because we're not connecting more primitive blocks together. It also isn't the input to a synthesis tool, since there is nothing to synthesise: this already is one of the primitive building blocks.

Essentially, this is a *model* of one of our building blocks. Ultimately when we synthesise a design, it must be resolved down to real logic gates that might, for example, exist as chips that we have in our cupboard. Also, a structural design produced by a synthesis tool will be a set of instructions as to how to connect together real logic gates that we might have in our cupboard. What we need is a way to tell VHDL about the properties of the real world devices that are kept in our cupboard, and will ultimately be used to build our designs.

That's the purpose of the description of the AND gate. It tells VHDL about the real physical properties of the real devices that I am going to use to build my designs.

The above description is a fairly basic model. It says what combinations of inputs will give rise to 1's and 0's at the outputs. A really useful model would give us extra information, for example what the delay of the gate is. If we know the delays of the individual gates, then we can estimate the overall delay of a system built up from these gates.

The AFTER keyword

Suppose we did some measurements on the gates that we have available to build our system, and we find that the AND gate has a delay of 10 ns. We could modify our description of the gate to this:

```
ARCHITECTURE with_timing_info OF and2 IS
BEGIN
    c <= a AND b AFTER 10 NS;
END ARCHITECTURE with_timing_info;
```

Now, when a or b change, the value of c will be re-computed, but c will not get its new value until 10 ns *after* the change in the input. VHDL knows about the following units of time:

Unit	Name	Meaning
PS	picosecond	10^{-12} seconds
NS	nanosecond	10^{-9} seconds
US	microsecond	10^{-6} seconds
MS	millisecond	10^{-3} seconds
S	second	
MIN	minute	60 seconds
HR	hour	60 minutes

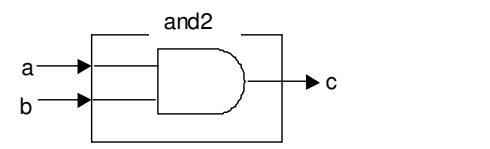
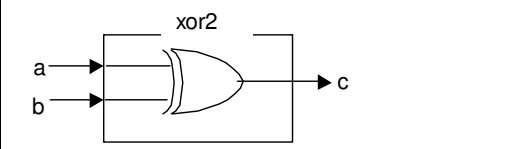
There is also another interval of time that is important in VHDL jargon. Delta is the smallest interval of time that the simulator can deal with. You can think of delta as meaning “a moment” or “an instant”.

How VHDL processes delays

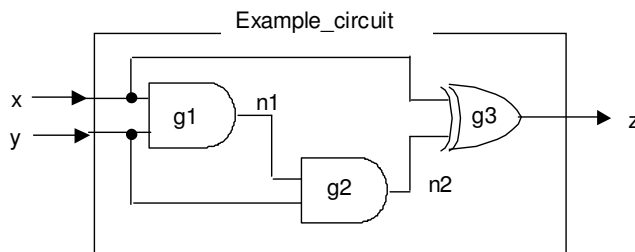
Let's assume that we have gates with the following delays:

- AND gate: 10 ns
- XOR gate: 15 ns

Here are the descriptions of the gates

	
<pre>LIBRARY ieee; USE ieee.std_logic_1164.ALL; ENTITY and2 is PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC); END ENTITY and2; ARCHITECTURE delayed OF and2 IS BEGIN c <= a AND b AFTER 10 NS; END ARCHITECTURE delayed;</pre>	<pre>LIBRARY ieee; USE ieee.std_logic_1164.ALL; ENTITY xor2 is PORT (a, b: IN STD_LOGIC; c: OUT STD_LOGIC); END ENTITY xor2; ARCHITECTURE delayed OF xor2 IS BEGIN c <= a OR b AFTER 15 NS; END ARCHITECTURE delayed;</pre>

Lets take a simple example circuit built from these gates:



```
ENTITY example_circuit is
    PORT ( x, y: IN STD_LOGIC;
          z: OUT STD_LOGIC);
END ENTITY example_circuit;
```

```

ARCHITECTURE structural OF example_circuit IS
    SIGNAL n1, n2: STD_LOGIC;
BEGIN
    g1: ENTITY work.and2(delayed) PORT MAP (x,y,n1);
    g2: ENTITY work.and2(delayed) PORT MAP (y,n1,n2);
    g3: ENTITY work.xor2(delayed) PORT MAP (x,n2,z);
END ARCHITECTURE structural;

```

Now, suppose this design is being simulated:

- At time zero, x='0' and y='1'. (So n1='0', n2='0' and z='0'.)
- At time 100 ns, x becomes '1'.

What happens next?

The event queue

The idea of an event queue is central to the way VHDL simulates code. For our particular circuit, it initially looks like this:

Time = 0

Signal Name:	x	y	n1	n2	z
Present value:	0	1	0	0	0
Next value:	1				
Event time:	100				

It has a list of the present value for each signal, any new value that has been scheduled to take place in future, and the time at which the signal must assume this new value. In this case then next event is that x will transition from '0' to '1' at time 100 ns.

Once the event queue is set up, the simulator proceeds by looking down the event queue to find the time of the next pending event. It then jumps forward to the time of the next event (100 ns), giving x its new value. It then looks at the netlist of the circuit

```

g1: ENTITY work.and2(delayed) PORT MAP (x,y,n1);
g2: ENTITY work.and2(delayed) PORT MAP (y,n1,n2);
g3: ENTITY work.xor2(delayed) PORT MAP (x,n2,z);

```

and finds which gates have x as an input. These gates (g1 and g3) have their outputs recomputed. The new values, and the scheduled time for these new values to happen, are placed on the event queue.

Time = 100

Signal Name:	x	y	n1	n2	z
Present value:	1	1	0	0	0
Next value:			1		1
Event time:			110		115

Note that the delay of the AND gate is 10 ns, so n1 is scheduled to get its new value at 110 ns, and the XOR delay is 15 ns, so z is scheduled to get its value at time 115 ns.

Now that all processing associated with time 100 ns is complete, the simulator again looks down the event queue to find the next scheduled event. The simulator advances its time pointer to 110 ns and updates the value of n1.

Time = 110

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	0	0
Next value:				1	1
Event time:				120	115

n1 acts as an input g2, so its output n2 is scheduled to get a new value (1) after the delay of the AND gate (10 ns).

Now that the time 110 ns has completed processing, the simulator jumps to the time of the next scheduled event, i.e. 115 ns and z gets its new value.

Time = 115

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	0	1
Next value:				1	
Event time:				120	

z isn't an input to any other gate, so the transition on z does not cause any further transitions to be scheduled. However, there is still a transition on n2 pending. The simulator jumps on to the time of this event and gives n2 its new value.

Time = 120

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	1	1
Next value:					0
Event time:					135

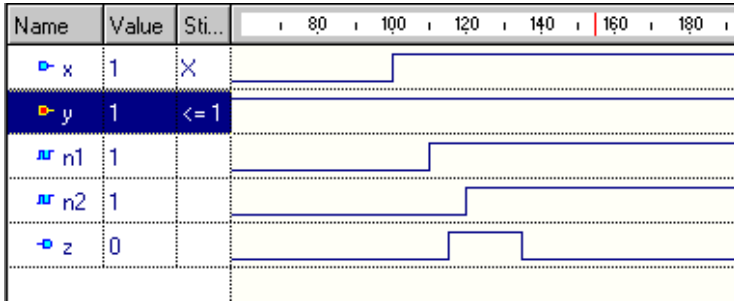
When n2 gets its new value, the output of g3 is re-computed, and scheduled to take its new value after the gate delay of the XOR gate. At time 135 ns, z gets its new value.

Time = 135

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	1	0
Next value:					
Event time:					

When we get to time 135, there is nothing left in the event queue, so the simulator terminates.

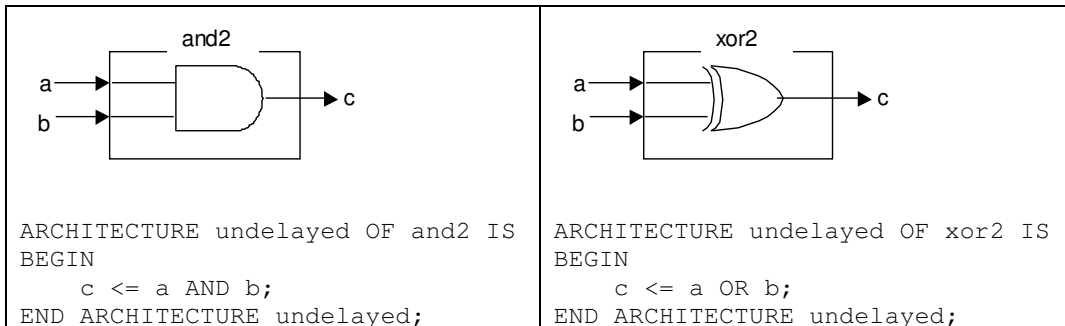
The appearance of the simulation is shown below.



This example illustrates an interesting situation. If we look at the initial values $x=0$, $y=1$, these dictate that $z=0$. Also the final values $x=1$, $y=1$ dictate that $z=0$. Without doing the detailed simulation, we might be tempted to assume that the results would be a constant zero for z . However, due to the gate delays, z glitches briefly up to '1' before settling down to its final value of '0'. Z has briefly gone through the "wrong" value before settling down to the "right" response to the inputs. If some other system is monitoring z , and responding to its value, this glitch could cause the system to do the wrong thing. This type of situation¹ can be very important, and it is important to be able to identify this situation.

Signal assignment always has delay.

Suppose we don't give the gates a delay, and just write our descriptions like this



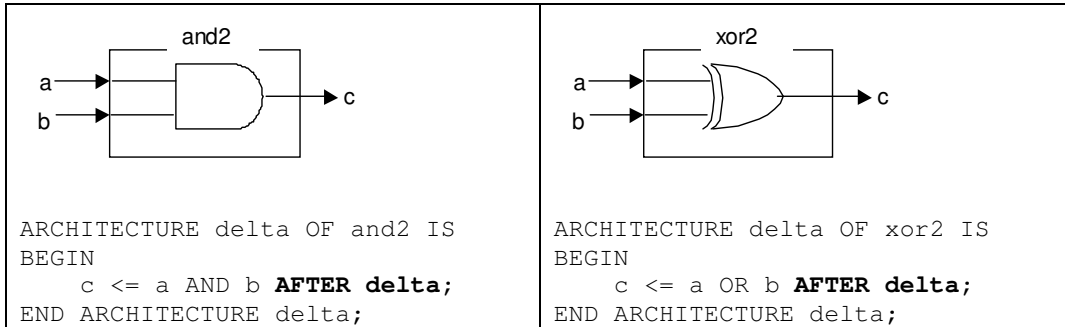
If the gates really were allowed to have zero delay, then we could miss the fact that our example circuit's output is glitchy. There are many other situations where the effect of gate delay is important².

In order to ensure that our descriptions behave in a way that realistically represent what the hardware is doing, it is a rule of VHDL that Signal assignments always have delay.

If we don't say what the delay is, then VHDL will insert a delay of *delta*. So our two gates would be interpreted by VHDL as if we had written this

¹ This is called a *static timing hazard*

² For example, many types of flip-flop circuit couldn't work properly if the gates had exactly zero delay.



This delta delay ensures that our descriptions behave realistically. So let's see how the simulation progresses using our gates with delta delay. This is the initial condition of the event queue:

Time = 0

Signal Name:	x	y	n1	n2	z
Present value:	0	1	0	0	0
Next value:	1				
Event time:	100				

The transition on x occurs at time 100, and causes n1 and z to be updated. The update *doesn't* happen immediately, but is scheduled to happen delta later.

Time = 100

Signal Name:	x	y	n1	n2	z
Present value:	1	1	0	0	0
Next value:			1		1
Event time:			100 + δ		100 + δ

The simulation then progresses as follows

Time = 100 + δ

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	0	1
Next value:				1	
Event time:				100 + 2 δ	

Time = 100 + 2 δ

Signal Name:	x	y	n1	n2	z
Present value:	1	1	0	1	1
Next value:					0
Event time:					100 + 3 δ

Time = 100 + 3 δ

Signal Name:	x	y	n1	n2	z
Present value:	1	1	0	1	0
Next value:					
Event time:					

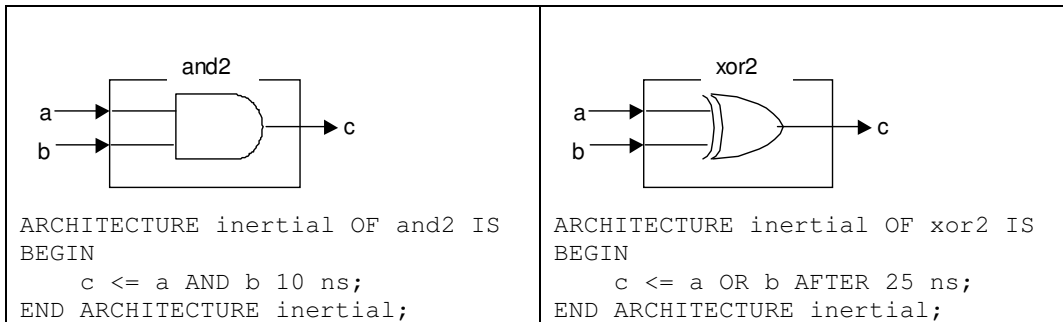
Observe that the simulation has correctly captured the fact that z is glitchy.

Inertial delay

Now let's change the delay of the gates to some slightly different values, in order to illustrate a further feature of VHDL:

- AND gate: 10 ns
- XOR gate: 25 ns

Here are the descriptions of the gates



Now let's see what happens

Time = 0

Signal Name:	x	y	n1	n2	z
Present value:	0	1	0	0	0
Next value:	1				
Event time:	100				

The transition on x causes n1 and z to have updated values scheduled.

Time = 100

Signal Name:	x	y	n1	n2	z
Present value:	1	1	0	0	0
Next value:			1		1
Event time:			110		125

At time 110 n1 gets its new value

Time = 110

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	0	0
Next value:				1	1
Event time:				120	125

thus causing a new value for n2 to be scheduled. At time 120, n2 gets its new value. *Notice that z still hadn't got the new value caused by the event on x, but in the mean time the change in n2 has caused another change for z. We have two events written onto the event queue for z.*

Time = 120

Signal Name:	X	y	n1	n2	z
Present value:	1	1	1	1	0
Next value:					1,0
Event time:					125,145

By default, the way that VHDL responds to this is that the second event over-writes the first, so the event queue looks like this:

Time = 120

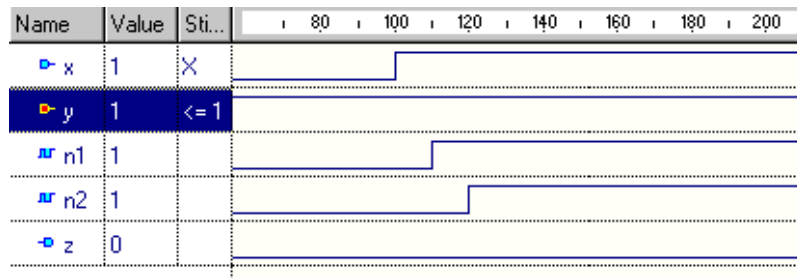
Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	1	0
Next value:					0
Event time:					145

So z stays at zero, and never changes up to 1.

Time = 135

Signal Name:	x	y	n1	n2	z
Present value:	1	1	1	1	0
Next value:					
Event time:					

So the simulation looks like this



This is called *inertial* delay, and is the normal behaviour in VHDL. Essentially what it means is that if the inputs of a gate change in such a way as to try to produce an output pulse that is shorter than the gate delay, then the output pulse is ironed out and never happens. Instead the output of the gate remains constant.

The reason why VHDL behaves in this way is that real hardware also tends to behave like this. Every electrical device has a certain amount of reluctance to change, or inertia³. In our simulation, the inputs to g3 were trying to cause a pulse 20 ns wide at the output of a gate whose delay was 25 ns. The inertia of the gate was too much, so the output simply stayed constant at zero.

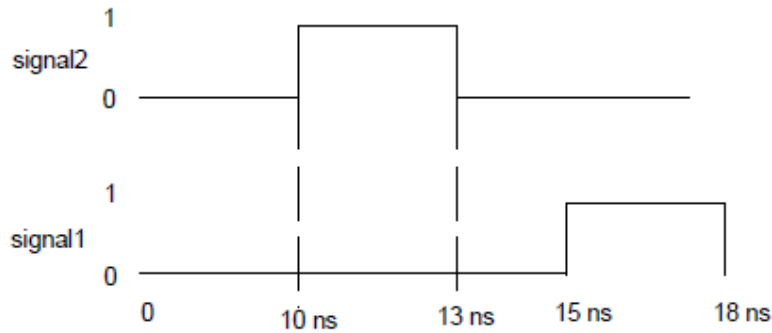
- **Transport delay**

The second type of delay is called *transport delay*. This occurs if we write

³ In this context inertia usually means electrical capacitance

```
signal1 <= TRANSPORT signal2 AFTER 5 NS;
```

With a transport delay, signal1 will simply follow signal2, delayed by 5 ns, as shown:



Sometimes transport delay is what we need in order to make our descriptions work. If a new value arrives on signal2 while an old value is propagating through the block, then there will be no mutual interference between the old value and the new value. In terms of the event queue, this has the following appearance.

Initially

Present value	0
Next value	
Event time	

Time=10

Present value	0
Next value	1
Event time	15

Time=13

Present value	0
Next value	1,0
Event time	15,18

Summary

We have seen how we can associate delays with assignments so that the timing of circuits can be correctly modelled by VHDL. It is important to understand that in VHDL, the assignment of a signal *always* has delay, and if we don't specify a delay then VHDL will insert a delta delay for us. The normal behaviour in VHDL is that assignments have inertial behaviour, i.e. if the inputs change so as to try to produce an

output transition that is shorter than the delay of the gate, then that transition will be smoothed out and will not occur

You should now know...

How to use the AFTER clause to assign delays to transitions

How VHDL uses an event queue to carry out simulations

How inertial delay irons out the effect of brief transient changes in signal

How Transports delay works