

When writing VHDL code for sequential circuits, two common pitfalls can lead to unintended behavior: undesired latches and combinational loops. Let us discuss how both of these mistakes can happen and what you can do to prohibit their occurrence.

HWMoD
WS25

Latches &
Loops

Latches
Comb. Loops
Summary

Hardware Modeling [VU] (191.011) – WS25 – Latches and combinational loops

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

- └ Latches and combinational loops
 - └ Undesired Latches
 - └ **Undesired Latches**

■ When should you use a latch or a flip-flop?

In previous lectures you learned about latches and flip-flops, and how they differ. Naturally, you might now question when you are supposed to use which.

Undesired Latches

- When should you use a latch or a flip-flop?

HWMod
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired

However, let us make one thing clear straight away: In the context of this course, latches are **always** a mistake. Under no circumstances should you create a circuit that contains a latch. But why is this the case?

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired

HWMod
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks

As we already discussed, the synchronous design style is used in an overwhelming majority of cases. In this style the flip-flop is the state-holding element of choice, as its sampling point directly relates to the time grid created by the clock. Due to the prominent use of this style, many technologies do not even provide latches as primitive building blocks.

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks

HWMod
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead

However, if the description of a circuit still models a latch, it will be built from available combinational logic.

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead

HWMoD
WS25

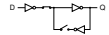
Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Undesired Latches

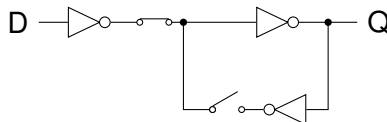
- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead



In its simplest form, such an implementation might look like the one depicted on the slide. Note that for the sake of simplicity we omitted the enable signal that control the switches. Let us quickly discuss how it works.

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead

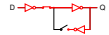


Latches and combinational loops

Undesired Latches

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead



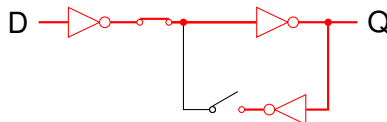
When the first switch is closed, the latch is enabled and forwards the input to its output. Note that this is what we referred to as the latch being *transparent* in the respective lecture.

Undesired Latches

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead

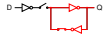


Latches and combinational loops

Undesired Latches

Undesired Latches

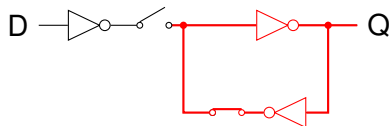
- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead



When the input switch is now open, the latch outputs the previously captured input value as it is stored in its internal feedback loop. The latch is now *opaque*.

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead

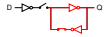


Latches and combinational loops

Undesired Latches

Undesired Latches

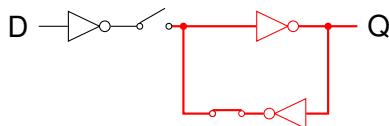
- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead
 - Delay of feedback path can become high



Now the problem with building a latch like the one shown is that the overall delay of the feedback path can become quite high, depending on the placement and routing of the circuit.

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead
 - Delay of feedback path can become high

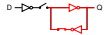


Latches and combinational loops

Undesired Latches

Undesired Latches

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead
 - Delay of feedback path can become high
 - Feedback path can lead to oscillation



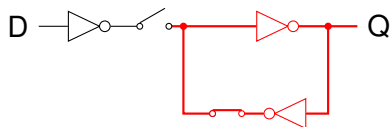
This can result in the feedback path starting to oscillate if the input was not stable for long enough. Obviously, this must never happen as subsequent circuit parts might also oscillate or observe undesired values.

Undesired Latches

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- When should you use a latch or a flip-flop?
- In this course latches are **always** undesired
- Often not available as primitive building blocks
 - Built from combinational elements instead
 - Delay of feedback path can become high
 - Feedback path can lead to oscillation



OK, so latches can be problematic and you should not use them in your designs. Then let's just not use the structure shown in the respective lecture and you are good, right? Not quite. Sometimes they happen by mistake, especially if you are new to describing hardware.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

■ How do latches happen?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12        if a <= b then
13            x <= '1';
14        end if;
15    end process;
16 end architecture;
```

The slide shows a simple combinational comparator as an example.

Pitfalls - Latch Errors (cont'd)

■ How do latches *happen*?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12        if a <= b then
13            x <= '1';
14        end if;
15    end process;
16 end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

How do latches happen?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x   : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12        if a <= b then
13            x <= '1';
14        end if;
15    end process;
16 end architecture;
```

It takes two `unsigned` inputs and computes an output of type `std_ulogic`.

Pitfalls - Latch Errors (cont'd)

How do latches *happen*?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x   : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12        if a <= b then
13            x <= '1';
14        end if;
15    end process;
16 end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

How do latches happen?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10  main : process (all) begin
11
12    if a <= b then
13      x <= '1';
14    end if;
15  end process;
16 end architecture;
```

Whenever a is smaller than b , the module shall output '1'.

Pitfalls - Latch Errors (cont'd)

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

How do latches *happen*?

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10  main : process (all) begin
11
12    if a <= b then
13      x <= '1';
14    end if;
15  end process;
16 end architecture;
```

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

- How do latches happen?
 - Not all paths write to signal

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x   : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12    end if;
13    end process;
14 end architecture;
```

The problem is that `x` is not always assigned when the process is triggered. For a combinational circuit this is problematic, as it should map *any* input combination to a respective output in a deterministic manner. So what will happen?

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal

```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x   : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10    main : process (all) begin
11
12    if a <= b then
13      x <= '1';
14    end if;
15  end process;
16 end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

```
■ How do latches happen?  
■ Not all paths write to signal  
■ Holding state infer latch
```

```
1 entity cmp is  
2 port(  
3   a, b : in  unsigned;  
4   x   : out std_ulogic  
5 );  
6 end entity;  
7  
8 architecture arch of cmp is  
9 begin  
10  main : process (all) begin  
11  
12    if a <= b then  
13      x <= '1';  
14    end if;  
15  end process;  
16 end architecture;
```

If we recall the semantics of signal assignments, the resulting behavior is actually quite intuitive. The signal x will simply hold its value between assignments. Thus, if there is a path through the process that does not write to x , we describe a state-holding behavior.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state infer latch

```
1 entity cmp is  
2 port (  
3   a, b : in  unsigned;  
4   x   : out std_ulogic  
5 );  
6 end entity;  
7  
8 architecture arch of cmp is  
9 begin  
10  main : process (all) begin  
11  
12    if a <= b then  
13      x <= '1';  
14    end if;  
15  end process;  
16 end architecture;
```

Latches and combinational loops

Undesired Latches

Pitfalls - Latch Errors (cont'd)

```
■ How do latches happen?  
■ Not all paths write to signal  
■ Holding state ⇒ infer latch
```

```
entity cmp is  
  port(  
    a, b : in  unsigned;  
    x    : out std_ulogic  
  );  
end entity;  
  
architecture arch of cmp is  
begin  
  main : process (all) begin  
    if a <= b then  
      x <= '1';  
    end if;  
  end process;  
end architecture;
```

In the absence of a clock this corresponds to a latch. In such cases, we say that a latch is *inferred* from the description.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state ⇒ infer latch

```
1 entity cmp is  
2   port (  
3     a, b : in  unsigned;  
4     x    : out std_ulogic  
5   );  
6 end entity;  
7  
8 architecture arch of cmp is  
9 begin  
10  main : process (all) begin  
11  
12    if a <= b then  
13      x <= '1';  
14    end if;  
15  end process;  
16 end architecture;
```

Latches and combinational loops

Undesired Latches

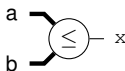
Pitfalls - Latch Errors (cont'd)



On the slide you can find the circuit that the code actually describes. How can the undesired latch be mitigated?

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state \Rightarrow infer latch



```
1 entity cmp is  
2   port (  
3     a, b : in  unsigned;  
4     x    : out std_ulogic  
5   );  
6 end entity;  
7  
8 architecture arch of cmp is  
9   begin  
10    main : process (all) begin  
11  
12      if a <= b then  
13        x <= '1';  
14      end if;  
15    end process;  
16 end architecture;
```

Latches and combinational loops

Undesired Latches

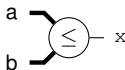
Pitfalls - Latch Errors (cont'd)



The answer is actually simple: If a combinational process writes to a signal, ensure it does so on all possible paths.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state \Rightarrow infer latch
- Always cover all paths/cases!



```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9 begin
10  main : process (all) begin
11
12    if a <= b then
13      x <= '1';
14    end if;
15  end process;
16 end architecture;
```

Latches and combinational loops

Undesired Latches

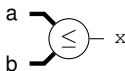
Pitfalls - Latch Errors (cont'd)



To achieve this, you can add a default assignment at the beginning of a combinational process, or make sure all cases are covered using the `others` keyword. Note that the latter of course only for `case` statements.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state \Rightarrow infer latch
- Always cover all paths/cases!
 - Default assignment or `others`

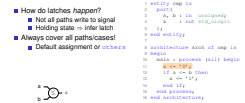


```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9   begin
10    main : process (all) begin
11
12      if a <= b then
13        x <= '1';
14      end if;
15    end process;
16 end architecture;
```

Latches and combinational loops

Undesired Latches

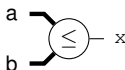
Pitfalls - Latch Errors (cont'd)



In our example, we simply set `x` to `'0'` before the `if` statement. If the condition is met, the previous assignment will be overwritten, if not, the output will be assigned an explicit value.

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state \Rightarrow infer latch
- Always cover all paths/cases!
 - Default assignment or `others`



```

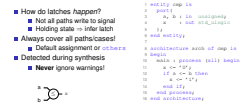
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9   begin
10    main : process (all) begin
11      x <= '0';
12      if a <= b then
13        x <= '1';
14      end if;
15    end process;
16 end architecture;

```

Latches and combinational loops

Undesired Latches

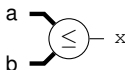
Pitfalls - Latch Errors (cont'd)



Finally, we want to mention that the synthesis tool is capable of detecting latches and it *will* warn you. You must **never** ignore warnings like that!

Pitfalls - Latch Errors (cont'd)

- How do latches *happen*?
 - Not all paths write to signal
 - Holding state \Rightarrow infer latch
- Always cover all paths/cases!
 - Default assignment or *others*
- Detected during synthesis
 - **Never** ignore warnings!



```
1 entity cmp is
2   port (
3     a, b : in  unsigned;
4     x    : out std_ulogic
5   );
6 end entity;
7
8 architecture arch of cmp is
9   begin
10    main : process (all) begin
11      x <= '0';
12      if a <= b then
13        x <= '1';
14      end if;
15    end process;
16 end architecture;
```

So far, we have discussed the problems resulting from undesired latches. A major problem is the possibility of the feedback loop oscillating. However, this is not the only scenario in which such a loop can be modelled by a circuit description.

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

Consider the code shown on the slide. It shows the architecture of a purely combinational 8-bit counter with wrap-around behavior on overflow.

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops

■ Feedback loops not exclusive to latches

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

Due to the use of `all` in its sensitivity list, the shown process will trigger whenever the counter variable is incremented. This directly leads to another increment. How does the circuit look? Is it problematic?

Pitfalls - Combinational Loops

■ Feedback loops not exclusive to latches

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

```

1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3 begin
4   comb : process(all) begin
5     cnt <= cnt + 1;
6   end process;
7 end architecture;

```



Next to the code you can see an abstract view of the circuit. The output of the combinational counter is directly fed back into one of its operand inputs.

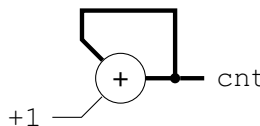
Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches

```

1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3 begin
4   comb : process(all) begin
5     cnt <= cnt + 1;
6   end process;
7 end architecture;

```



```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



Evidently, the circuit features a combinational feedback loop and is prone to the same problems as the feedback loop in the latch we considered before. How can you prevent such loops?

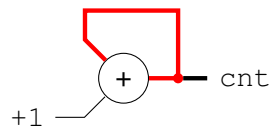
Pitfalls - Combinational Loops

HWMoD
WS25

Latches &
Loops
└─ Latches
└─ Comb. Loops
└─ Summary

■ Feedback loops not exclusive to latches

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read and write to same signal

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



Again, the prevention is quite simple: combinational processes must **never** read and write to the same signal. Such an assignment will always trigger the process which results in another assignment, thus creating a loop.

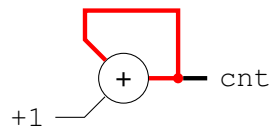
Pitfalls - Combinational Loops

HWMMod
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- Feedback loops not exclusive to latches
- Comb. process must never read **and** write to same signal

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



Latches and combinational loops

- Comb. Loops
- Pitfalls - Combinational Loops**

- Feedback loops not exclusive to latches
- Comb. process must never read and write to same signal
- Hard to spot at interfaces

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

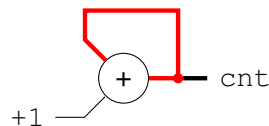


Note that special care must be taken at interfaces between modules, as it can be quite hard to spot such loops. However, feedback loops can be required, for example in control applications. So how do you implement them then?

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read **and** write to same signal
 - Hard to spot at interfaces

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read and write to same signal
- Hard to spot at interfaces
- Feedback paths must contain flip-flops

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3 begin
4   comb : process(all) begin
5     cnt <= cnt + 1;
6   end process;
7 end architecture;
```

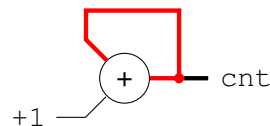


When describing a feedback loop, the key is to include flip-flops in the feedback path. This way, signal transitions propagate on a per-clock-period basis, preventing oscillations. In terms of VHDL this also means that the value generated by a combinational process is only assigned to a signal in a synchronous, and hence different, process.

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read **and** write to same signal
 - Hard to spot at interfaces
 - Feedback paths must contain flip-flops

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3 begin
4   comb : process(all) begin
5     cnt <= cnt + 1;
6   end process;
7 end architecture;
```



Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read and write to same signal
 - Hard to spot at interfaces
 - Feedback paths must contain flip-flops
- Also reported during synthesis

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```

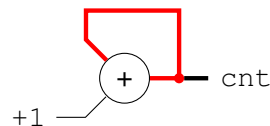


Finally, just as with latches, tools will detect combinational loops and warn you about them. Again: do **not** ignore such warnings!

Pitfalls - Combinational Loops

- Feedback loops not exclusive to latches
- Comb. process must never read **and** write to same signal
 - Hard to spot at interfaces
 - Feedback paths must contain flip-flops
- Also reported during synthesis

```
1 architecture beh of counter is
2   signal cnt : unsigned(7 downto 0);
3   begin
4     comb : process(all) begin
5       cnt <= cnt + 1;
6     end process;
7 end architecture;
```



└ Latches and combinational loops

└ Comb. Loops

└ Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11
12
13   end architecture;
```

Let us finish by considering how the counter can be implemented without a purely combinational feedback loop.

Pitfalls - Combinational Loops (cnt'd)

HWMoD
WS25

Latches &
Loops
└ Latches
└ Comb. Loops
└ Summary

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11
12
13
14
15 end architecture;
```

- └ Latches and combinational loops
 - └ Comb. Loops
 - └ **Pitfalls - Combinational Loops (cnt'd)**

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11
12
13
14   end architecture;
```

The first thing we do is introducing an additional signal with the suffix `_next`.

Pitfalls - Combinational Loops (cnt'd)

HWMoD
WS25

Latches &
Loops
└ Latches
└ Comb. Loops
└ Summary

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11
12
13
14
15   end architecture;
```

- └ Latches and combinational loops
 - └ Comb. Loops
 - └ **Pitfalls - Combinational Loops (cnt'd)**

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11   comb : process(all) begin
12     cnt_next <= cnt + 1;
13   end process;
14 end architecture;
```

We use this signal to refer to the output of the combinational adder and hence the combinational process will assign the incremented value to it.

Pitfalls - Combinational Loops (cnt'd)

HWMMod
WS25

Latches &
Loops
└ Latches
└ Comb. Loops
└ Summary

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4
5
6
7
8
9
10
11
12   comb : process(all) begin
13     cnt_next <= cnt + 1;
14   end process;
15 end architecture;
```

Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture type of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   comb
4   sync : process(clk, res_n) begin
5     if res_n = '0' then
6       cnt <= (others => '0');
7     elsif rising_edge(clk) then
8       cnt <= cnt_next;
9     end if;
10  end process;
11
12  comb : process(all) begin
13    cnt_next <= cnt + 1;
14  end process;
15 end architecture;
```

We use the other signal, `cnt`, for describing a register.

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4     sync : process(clk, res_n) begin
5       if res_n = '0' then
6         cnt <= (others => '0');
7       elsif rising_edge(clk) then
8         cnt <= cnt_next;
9       end if;
10    end process;
11
12    comb : process(all) begin
13      cnt_next <= cnt + 1;
14    end process;
15  end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4     sync : process(clk, res_n) begin
5       if res_n = '0' then
6         cnt <= (others => '0');
7       elsif rising_edge(clk) then
8         cnt <= cnt_next;
9       end if;
10    end process;
11
12    comb : process(all) begin
13      cnt_next <= cnt + 1;
14    end process;
15  end architecture;
```

Note that the respective synchronous process adheres to the structure we introduced in a previous lecture, featuring an asynchronous reset and sampling `cnt_next` exclusively on rising clock edges.

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4     sync : process(clk, res_n) begin
5       if res_n = '0' then
6         cnt <= (others => '0');
7       elsif rising_edge(clk) then
8         cnt <= cnt_next;
9       end if;
10    end process;
11
12    comb : process(all) begin
13      cnt_next <= cnt + 1;
14    end process;
15  end architecture;
```

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

Latches and combinational loops

Comb. Loops

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3 begin
4   sync : process(clk, res_n) begin
5     if res_n = '0' then
6       cnt <= (others => '0');
7     elsif rising_edge(clk) then
8       cnt <= cnt_next;
9     end if;
10  end process;
11
12  comb : process(all) begin
13    cnt_next <= cnt + 1;
14  end process;
15 end architecture;
```

Note how the respective combinational process is almost the same as before, except for one detail. Instead of having the same signal on *both* sides of the assignment, the current register output is incremented and assigned to the signal the flip-flop will sample next. Observe how the combinational process now no longer reads from and writes to the same signal! This is also reflected by the corresponding circuit.

Pitfalls - Combinational Loops (cnt'd)

```
1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3 begin
4   sync : process(clk, res_n) begin
5     if res_n = '0' then
6       cnt <= (others => '0');
7     elsif rising_edge(clk) then
8       cnt <= cnt_next;
9     end if;
10  end process;
11
12  comb : process(all) begin
13    cnt_next <= cnt + 1;
14  end process;
15 end architecture;
```

HWMMod
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- └ Latches and combinational loops
- └ Comb. Loops
- └ Pitfalls - Combinational Loops (cnt'd)

```

1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4     sync : process(clk, res_n) begin
5       if res_n = '0' then
6         cnt <= (others => '0');
7       elsif rising_edge(clk) then
8         cnt <= cnt_next;
9       end if;
10    end process;
11
12    comb : process(all) begin
13      cnt_next <= cnt + 1;
14    end process;
15  end architecture;

```



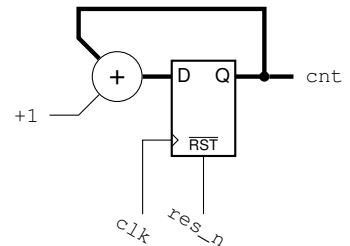
As could be expected, it consists of an adder and a flip-flop. The flip-flop intercepts the path from the adder's output to its input, eliminating the combinational loop.

Pitfalls - Combinational Loops (cnt'd)

```

1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3   begin
4     sync : process(clk, res_n) begin
5       if res_n = '0' then
6         cnt <= (others => '0');
7       elsif rising_edge(clk) then
8         cnt <= cnt_next;
9       end if;
10    end process;
11
12    comb : process(all) begin
13      cnt_next <= cnt + 1;
14    end process;
15  end architecture;

```



- └ Latches and combinational loops
- └ Comb. Loops
- └ Pitfalls - Combinational Loops (cnt'd)

```

1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3 begin
4   sync : process(clk, res_n) begin
5     if res_n = '0' then
6       cnt <= (others => '0');
7     elsif rising_edge(clk) then
8       cnt <= cnt_next;
9     end if;
10  end process;
11
12  comb : process(all) begin
13    cnt_next <= cnt + 1;
14  end process;
15 end architecture;

```



We want to remark that the general structure of the code is an important design pattern used in many VHDL designs. A synchronous process models flip-flops that sample `_next` signals that are produced by combinational processes. Keep this example in mind when writing your own code.

Pitfalls - Combinational Loops (cnt'd)

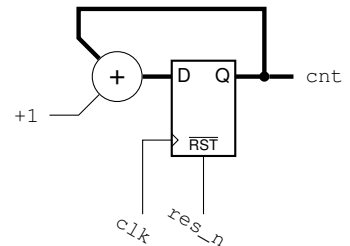
HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

```

1 architecture sync of counter is
2   signal cnt, cnt_next : unsigned(7 downto 0);
3 begin
4   sync : process(clk, res_n) begin
5     if res_n = '0' then
6       cnt <= (others => '0');
7     elsif rising_edge(clk) then
8       cnt <= cnt_next;
9     end if;
10  end process;
11
12  comb : process(all) begin
13    cnt_next <= cnt + 1;
14  end process;
15 end architecture;

```



Let us finish this lecture by highlighting its key takeaways. Undesired latches often originate from missing default assignments in combinational processes. To mitigate this, use default assignments or ensure you cover all possible outcomes of conditions.

Summary

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- Latches
 - Often due to missing default values

Combinational loops result from combinational logic reading from and writing to the same signal. This creates feedback paths without flip-flops. To mitigate this, ensure your combinational processes never read from and write to the same signal. If feedback is required, introduce flip-flops in the path.

Summary

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- Latches
 - Often due to missing default values
- Comb. loops
 - Never read **and** write same signal

Finally, **never** ignore warnings produced by the tools. In this course, such warnings always highlight fatal issues in your design. The slide shows examples of how such warnings look.

Summary

HWMoD
WS25

Latches &
Loops
Latches
Comb. Loops
Summary

- Latches
 - Often due to missing default values
- Comb. loops
 - Never read **and** write same signal
- Never ignore tool warnings!

10631 VHDL Process Statement warning at top_arch.vhd(13): inferring latch(es) for signal or variable "abc", which holds its previous value in one or more paths through the process

332125 Found combinational loop of 2 nodes

Lecture Complete!

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

HWMod
WS25

Latches &
Loops

Latches

Comb. Loops

Summary

Lecture Complete!