

Previously you heard about the synchronous paradigm for designing digital circuits. However, what you did not learn yet is how the sequential circuit elements that are required for synchronous circuits can be described and used in VHDL. So let's get to it.

HWMoD
WS25

Seq. Elem.

Introduction
D Latches
D Flip-Flop

Hardware Modeling [VU] (191.011) – WS25 – Sequential Circuit Elements in VHDL

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

In synchronous design, circuit elements with a memory allow circuits to store and utilize past states or input values. This storing of information is coordinated by a global clock signal. Based on this signal, the state-changes in memory elements such as latches, and the previously mentioned flip-flops, occur.

Introduction

HWMoD
WS25

Seq. Elem.
Introduction

Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic

Before we start, note that both latches and flip-flops are storage elements that hold a *single* bit of data. We will discuss larger memory elements, such as RAMs, ROMs and FIFOs in an upcoming lecture and now exclusively focus on latches and flip-flops.

Introduction

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements

The major difference between latches and the already familiar flip-flops is how they react to changes of their input signals. While a latch captures its input when the clock is high or low, a flip-flop does so only at the edges of the clock signal. We refer to this as latches being *level-sensitive*, and flip-flops being *edge-triggered*.

Introduction

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements
- Operation principle
 - Latches: level-sensitive
 - Flip-flops: edge-triggered

The level-sensitive nature of latches makes them somewhat problematic in synchronous designs. While we will only discuss this in more detail in an upcoming lecture, for now simply accept that they must be strictly avoided in all designs during this course. Nevertheless, it is important to cover both concepts as they represent fundamental building blocks of digital systems and potentially related problems.

Introduction

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements
- Operation principle
 - Latches: level-sensitive
 - Flip-flops: edge-triggered
- Latches can be **problematic** in synchronous designs!

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements
- Operation principle
 - Latches: level-sensitive
 - Flip-flops: edge-triggered
- Latches can be **problematic** in synchronous designs!
- Common Types
 - Latches: RS, D
 - Flip-flops: JK, T, D

As you already learned in other courses, there exists a range of latches and flip-flops, such as RS or D latches, and JK, T or D flip-flops.

Introduction

HWMoD
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements
- Operation principle
 - Latches: level-sensitive
 - Flip-flops: edge-triggered
- Latches can be **problematic** in synchronous designs!
- Common Types
 - Latches: RS, **D**
 - Flip-flops: JK, T, **D**

In this course only the D-type versions are of importance, where D is an abbreviation for data. Let us now continue with some definitions.

Introduction

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Combinational logic cannot retain any data ⇒ Sequential logic
- Latches and flip-flops are single-bit storage elements
- Operation principle
 - Latches: level-sensitive
 - Flip-flops: edge-triggered
- Latches can be **problematic** in synchronous designs!
- Common Types
 - Latches: RS, **D**
 - Flip-flops: JK, T, **D**
- Relevant for this course: Data (D) type

A D latch is a level-sensitive single-bit storage device featuring an enable input. Whenever this enable is active, the latch continuously transfers the signal level on its input D to its output Q. When the enable input is inactive, the latch holds the last output value.

Introduction (cont'd)

HWMoD
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

Often the state where the latch propagates its input is referred to as the latch being *open*, *enabled*, or *transparent*, since for the input it appears as though the signal passes directly through the latch. The other state is typically referred to as the latch being *closed*, *disabled*, or *opaque*.

Introduction (cont'd)

HWMoD
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

D flip-flops on the other hand only capture their input signal at single moments in time, marked by rising or falling transitions of a clock signal. During all other times the input signals can not affect the output of a D flip-flop.

Introduction (cont'd)

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

D Flip-Flop

D flip-flops are edge-triggered. They capture the data on the input (D) at a specific clock edge, transfer it to the output (Q) and hold it until the next edge.

The specific clock edge the flip-flop reacts to is referred to as the *active clock edge*. Typically, the rising edge is used for this purpose. We will therefore only use this variant in this course.

Introduction (cont'd)

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

D Flip-Flop

D flip-flops are edge-triggered. They capture the data on the input (D) at a specific clock edge, transfer it to the output (Q) and hold it until the next edge.

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

D Flip-Flop

D flip-flops are edge-triggered. They capture the data on the input (D) at a specific clock edge, transfer it to the output (Q) and hold it until the next edge.

Register

Registers are collections of D flip-flops (latches) that hold data that logically belong together.

Finally, let us define the term register, as it is also important in the context of storage elements. A register is a set of single-bit storage elements – usually flip-flops – that are triggered by the same clock signal and work in unison to store multiple bits of data that logically belong together.

Introduction (cont'd)

HWMod
WS25

Seq. Elem.
Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

D Latch

D latches are level-sensitive. They transfer the data on the input (D) to the output (Q) when enabled and retain the data when disabled.

D Flip-Flop

D flip-flops are edge-triggered. They capture the data on the input (D) at a specific clock edge, transfer it to the output (Q) and hold it until the next edge.

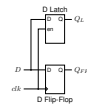
Register

Registers are collections of D flip-flops (latches) that hold data that logically belong together.

Sequential Circuit Elements in VHDL

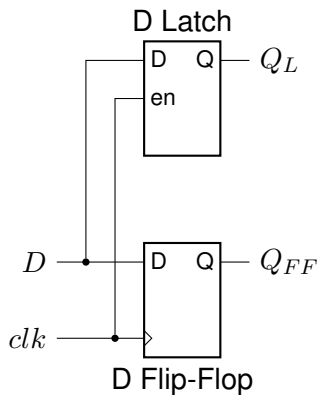
Introduction

D Latches vs. D Flip-Flop



It is crucial that you really understand the difference between latches and flip-flops. Hence, let us take the time to consider this simple circuit consisting of a D latch and a D flip-flop. Both storage elements are connected to the same input signal D . The signal used to clock the flip-flop (clk) is used as the enable signal for the D latch.

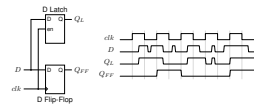
D Latches vs. D Flip-Flop



HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Sequential Circuit Elements in VHDL
 - Introduction
 - D Latches vs. D Flip-Flop**

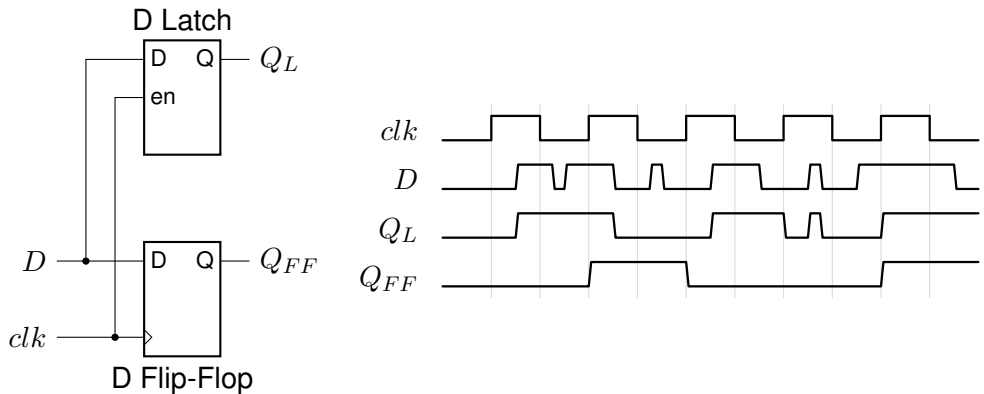


The timing diagram on the right side of the slide shows how the two storage elements react to the same input signals. Note that for the sake of simplicity, we omit the propagation delay of the latch, and the clock-to-output delay of the flip-flop. Let's first consider the D latch.

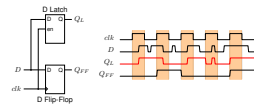
D Latches vs. D Flip-Flop

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop



- Sequential Circuit Elements in VHDL
 - Introduction
 - D Latches vs. D Flip-Flop**

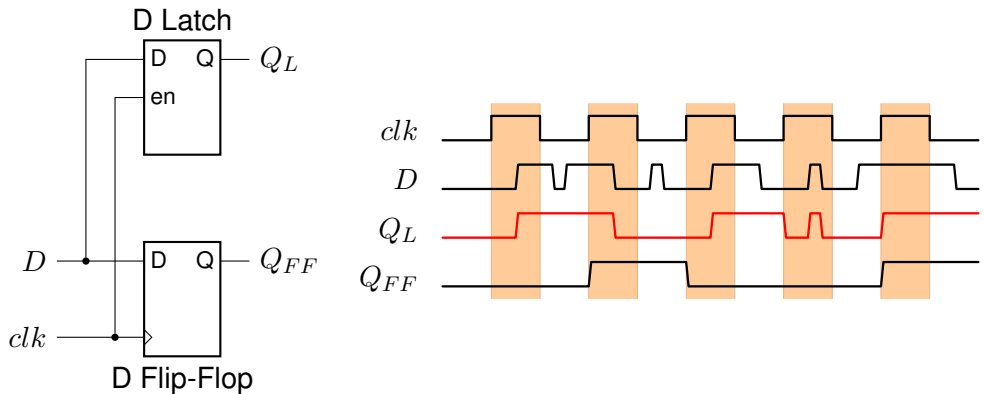


As you already heard, it is transparent when its enable signal is active. In this case, this is during the high period of the clock, which is highlighted in the timing diagram. During these periods the output signal Q_L directly follows the input signal D .

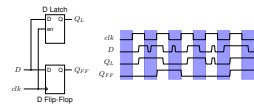
D Latches vs. D Flip-Flop

HWMoD
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop



- Sequential Circuit Elements in VHDL
 - Introduction
 - D Latches vs. D Flip-Flop**

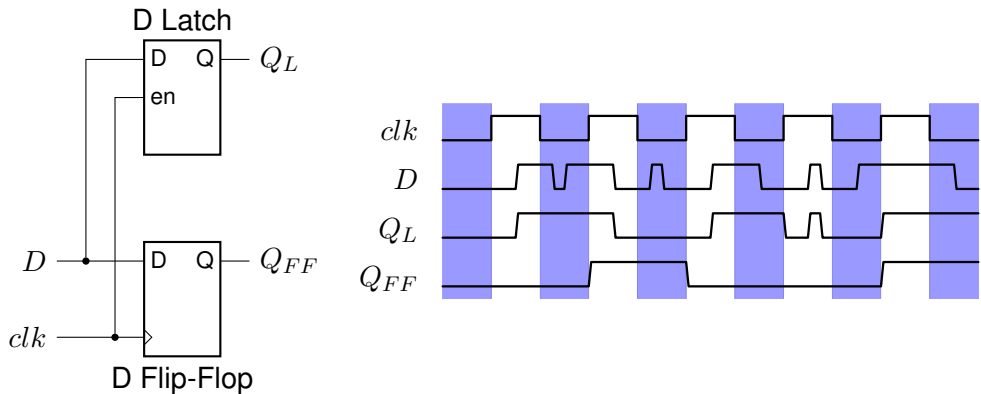


In contrast, during the low-phase, Q_L is kept at the last value it had during the high period. Changes of D during this phase have no effect.

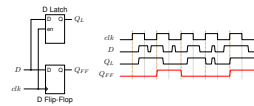
D Latches vs. D Flip-Flop

HWMoD
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

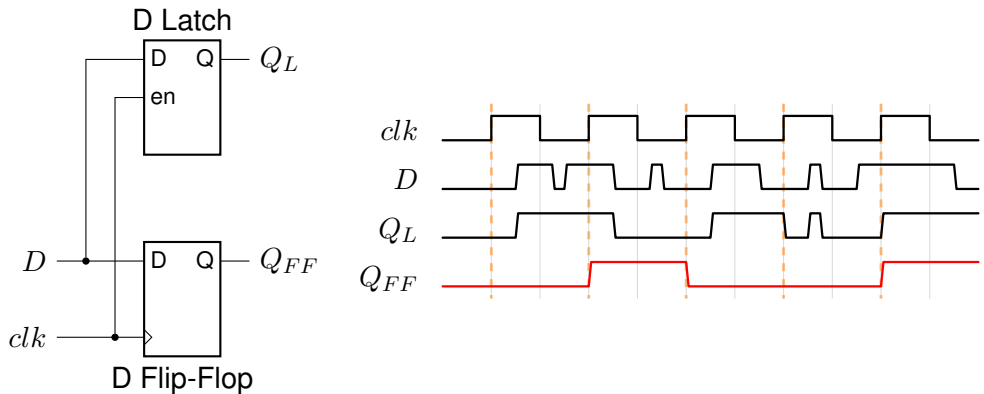


- Sequential Circuit Elements in VHDL
 - Introduction
 - D Latches vs. D Flip-Flop**



For the D flip-flop only the rising clock edges are relevant. Only at these points in time D is captured and transferred to Q_{FF} . For the rest of the time the value of D is completely irrelevant.

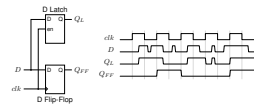
D Latches vs. D Flip-Flop



HWMoD
WS25

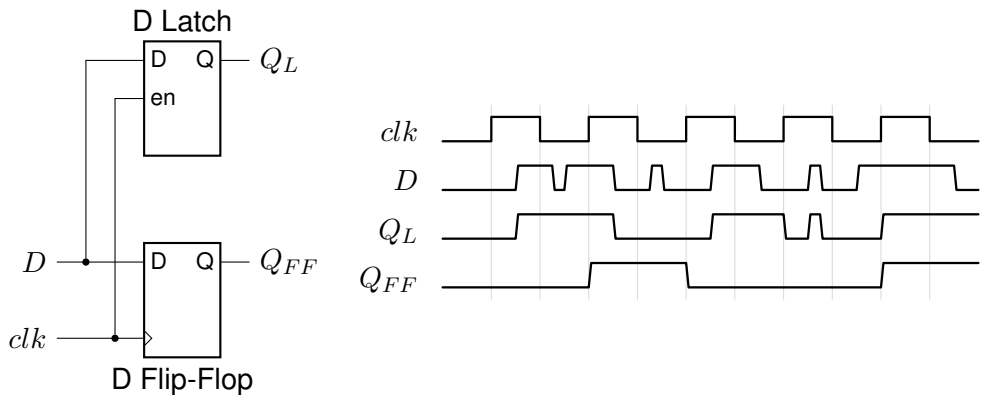
Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Sequential Circuit Elements in VHDL
 - Introduction
 - D Latches vs. D Flip-Flop**



Please do not continue watching this lecture if this paramount difference is not clear to you at this point! If necessary, pause the video to really understand the shown timing diagram.

D Latches vs. D Flip-Flop



HWMod
WS25

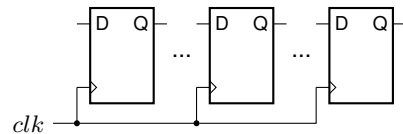
Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop



Typically, when we design a digital circuit, we want the storage elements to be initialized with a specific value upon power-up. Furthermore, when the circuit entered a faulty state from which a circuit cannot recover by other means, it can become necessary to reset the states of all storage elements to their initial value. This is the purpose of the reset signal.

Reset Signal

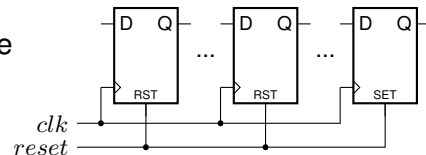
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault



Consider the circuit featuring a reset on the slide. Like the clock, the reset is also a global signal that is usually connected to all storage elements in a design or module.

Reset Signal

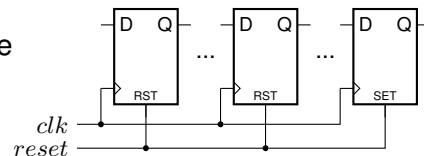
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module



On many chips the reset is a physical external pin that can, for example, be connected to a push button.

Reset Signal

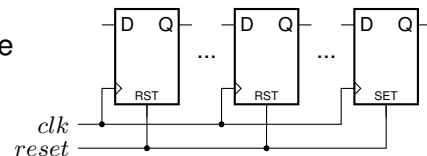
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button



The reset signal ensures that all storage elements are set to a predefined value, allowing the system to start from a known, stable state. A latch or flip-flop without a reset can power-up to an arbitrary value.

Reset Signal

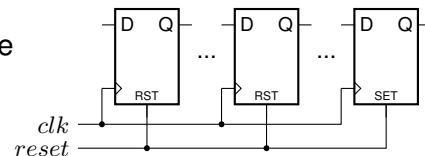
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button
- Prevents power-up to an arbitrary state



Usually this is undesired, which is why it is good practice to reset all sequential elements, even if you consider it unnecessary. This also minimizes the potential for divergence between the behavior of the simulation and hardware.

Reset Signal

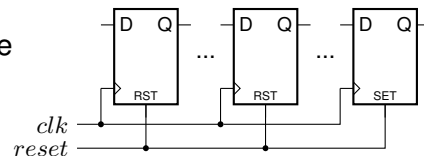
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button
- Prevents power-up to an arbitrary state
- Include reset for all registers!



Typically, storage elements are reset to hold a logical zero. However, this is not a strict rule. Sometimes it might be necessary to initialize storage elements to one. This is shown for the right-most flip-flop in the drawing. Note that we will mark flip-flops and latches whose reset value is one, by labelling their reset inputs with SET instead of RST.

Reset Signal

- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button
- Prevents power-up to an arbitrary state
- Include reset for all registers!
- Typical reset value is zero/low (sometimes different values are necessary)



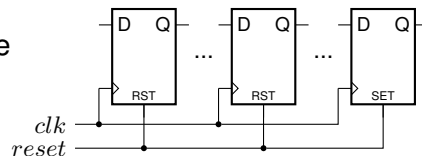
- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button
- Prevents power-up to an arbitrary state
- Include reset for all registers!
- Typical reset value is zero/low (sometimes different values are necessary)
- Testbenches must **always** activate the UUT's reset upon startup



At this point we want to mention that testbenches must always start by activating the reset. This brings the UUT into a defined state before applying test stimuli. Furthermore, while the reset is active all input signals to the UUT should also be set to defined values. As a result, when the reset signal is released the design only sees valid input values.

Reset Signal

- Purpose: Bring a circuit into a defined state
 - after power-up
 - in case of a fault
- Global signal
 - Connects to the reset inputs of all registers in design or module
 - Often connected to an external button
- Prevents power-up to an arbitrary state
- Include reset for all registers!
- Typical reset value is zero/low (sometimes different values are necessary)
- Testbenches must **always** activate the UUT's reset upon startup



Before we look at some VHDL code examples, we have to introduce the important concept of active signal levels. This refers to the specific logic level at which a signal is considered *active* or *asserted*.

Active Signal Levels

HWMoD
WS25

Seq. Elem.

Introduction

Latches vs. FFs

Reset Signal

Active Signal Levels

D Latches

D Flip-Flop

In binary digital systems, a signal can either be *active-high* or *active-low*. An active-high signal is active when it exhibits logical 1, typically represented by a high voltage level. Conversely, an active-low signal is considered active when its logical value is 0.

Active Signal Levels

HWMMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

To clearly distinguish active-low from active-high signals, we mark active-low signals in circuit diagrams with a horizontal bar above the signal name and identifiers in source code with the suffix `_n`. Examples are shown on the slide.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)

The first time you will come into touch with an active-low signal is the reset signal. In digital designs the reset is often implemented in an active-low way, as this comes with several benefits.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., $\overline{0T}$)
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*

This design choice ensures that the system starts in a predictable state during power-up, as logic lines tend to default to low voltage, often referred to as *ground*.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., $\overline{0T}$)
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity

Additionally, active-low resets are more robust in noisy environments, because electrical interference is less likely to falsely trigger a low signal than a high one.

Active Signal Levels

HWMod
WS25

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., $\overline{0T}$)
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility

These characteristics, combined with the historical adoption of active-low conventions in TTL and CMOS technologies, make it a reliable practice and industry-standard.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., $\overline{0T}$)
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility
 - Our naming convention: `res_n`.

We will therefore exclusively use active-low resets, referring to this signal as `res_n`.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility
 - Our naming convention: `res_n`.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., $\overline{0T}$)
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility
 - Our naming convention: `res_n`.

With that being said, we have everything we need to implement our first D latch in VHDL. We start with the latch as it is arguably the simpler state holding element.

Active Signal Levels

HWMod
WS25

Seq. Elem.
Introduction
Latches vs. FFs
Reset Signal
Active Signal Levels
D Latches
D Flip-Flop

Active Signal Levels

A signal is considered active-high (low) when a high (low) logic level activates the signal or causes the intended action to occur.

- Notation for active-low signals:
 - Circuit diagrams: a bar above the signal name (e.g., \overline{en})
 - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
 - Power-up default often *ground*
 - Noise immunity
 - Widely adopted in industry for reliability and compatibility
 - Our naming convention: `res_n`.

Sequential Circuit Elements in VHDL

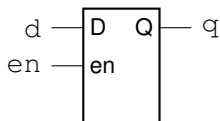
D Latches

D Latches



The entity declaration of a D latch, shown on the slide, is quite straight-forward. It contains two inputs, `d` and `en`, as well as a single output `q`.

D Latches

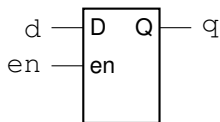


```
1 entity d_latch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic
6   );
7 end entity;
8
9 architecture arch of d_latch is
10 begin
11
12
13
14
15
16
17 end architecture;
```



In the architecture the first thing that needs to be changed is that the process is sensitive to both inputs.

D Latches



```

1 entity dlatch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic
6   );
7 end entity;
8
9 architecture arch of dlatch is
10  begin
11    process(en, d)
12    begin
13
14
15    end process;
16 end architecture;

```



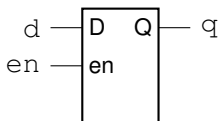
```

1 entity dlatch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic
6   );
7 end entity;
8
9 architecture arch of dlatch is
10  begin
11    process(en, d)
12    begin
13
14
15    end process;
16 end architecture;

```

For the implementation of the desired behavior of the D latch, recall the behavioral specification from earlier.

D Latches



```

1 entity dlatch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic
6   );
7 end entity;
8
9 architecture arch of dlatch is
10  begin
11    process(en, d)
12    begin
13
14
15    end process;
16 end architecture;

```



Operation Principle

While en is high, q is assigned the value of d , otherwise q is not updated and simply holds its last value.

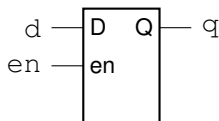
```

1 entity dlatch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic);
6 end entity;
7
8 architecture arch of dlatch is
9   signal q_reg : std_ulogic;
10  begin
11   q <= q_reg;
12   process(en, d)
13   begin
14     if en = '1' then
15       q_reg <= d;
16     end if;
17 end process;
18 end architecture;

```

Whenever en is high, the latch is supposed to propagate d to q . When it is not enabled, the output should hold its previous value.

D Latches



```

1 entity dlatch is
2   port (
3     d : in std_ulogic;
4     en : in std_ulogic;
5     q : out std_ulogic);
6 end entity;
7
8 architecture arch of dlatch is
9 begin
10  process(en, d)
11  begin
12    if en = '1' then
13      q <= d;
14    end if;
15  end process;
16 end architecture;

```

Operation Principle

While en is high, q is assigned the value of d , otherwise q is not updated and simply holds its last value.



```

1 entity d_latch is
2     port (
3         d : in std_ulogic;
4         en : in std_ulogic;
5         q : out std_ulogic
6     );
7 end entity;
8
9 architecture arch of d_latch is
10    signal q_reg : std_ulogic;
11 begin
12    process(en, d)
13        if en = '1' then
14            q_reg <= d;
15        end if;
16    end process;
17 end architecture;

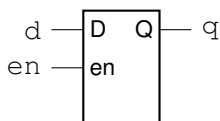
```

Operation Principle

While `en` is high, `q` is assigned the value of `d`, otherwise `q` is not updated and simply holds its last value.

To implement this in the process body we need a single if-condition. The condition checks whether the enable signal is asserted and assigns the signal `d` to `q` if that's the case. Should this condition be false, the signal value of `q` is not updated, meaning that it will retain its previous value.

D Latches



```

1 entity d_latch is
2     port (
3         d : in std_ulogic;
4         en : in std_ulogic;
5         q : out std_ulogic
6     );
7 end entity;
8
9 architecture arch of d_latch is
10    begin
11        process(en, d)
12            begin
13                if en = '1' then
14                    q <= d;
15                end if;
16            end process;
17        end architecture;

```

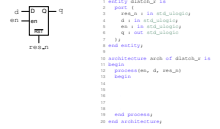
Operation Principle

While `en` is high, `q` is assigned the value of `d`, otherwise `q` is not updated and simply holds its last value.

Sequential Circuit Elements in VHDL

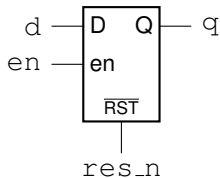
D Latches

D Latch - Reset



To implement a reset mechanism for the D latch, we first extend its interface by a reset input. Adhering to what we discussed before, we use an active-low reset.

D Latch - Reset



```
1 entity d_latch_r is
2     port (
3         res_n : in std_ulogic;
4         d : in std_ulogic;
5         en : in std_ulogic;
6         q : out std_ulogic
7     );
8 end entity;
9
10 architecture arch of d_latch_r is
11 begin
12     process(en, d, res_n)
13     begin
14
15
16
17
18
19     end process;
20 end architecture;
```

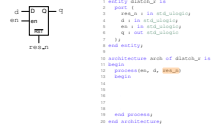
HWMoD
WS25

Seq. Elem.
Introduction
D Latches
Reset
D Flip-Flop

Sequential Circuit Elements in VHDL

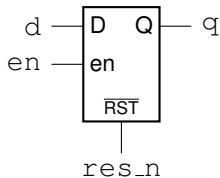
D Latches

D Latch - Reset



In our latch process we first have to extend the sensitivity list to also include the reset input.

D Latch - Reset



```
1 entity d_latch_r is
2     port (
3         res_n : in std_ulogic;
4         d : in std_ulogic;
5         en : in std_ulogic;
6         q : out std_ulogic
7     );
8 end entity;
9
10 architecture arch of d_latch_r is
11 begin
12     process(en, d, res_n)
13     begin
14
15
16
17
18
19     end process;
20 end architecture;
```

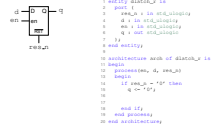
HWMoD
WS25

Seq. Elem.
Introduction
D Latches
Reset
D Flip-Flop

Sequential Circuit Elements in VHDL

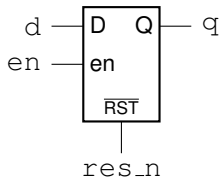
D Latches

D Latch - Reset



Then we introduce the reset condition. Since the reset signal is active-low, we need to test it for the value '0'. If this condition is true, the output q is simply set to '0'.

D Latch - Reset



```
1 entity d latch_r is
2   port (
3     res_n : in std_ulogic;
4     d : in std_ulogic;
5     en : in std_ulogic;
6     q : out std_ulogic
7   );
8 end entity;
9
10 architecture arch of d latch_r is
11 begin
12   process(en, d, res_n)
13   begin
14     if res_n = '0' then
15       q <= '0';
16
17
18     end if;
19   end process;
20 end architecture;
```



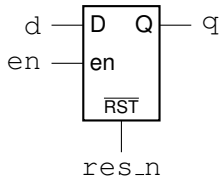
```

1 entity dlatcl_r is
2     port (
3         res_n : in std_ulogic;
4         d : in std_ulogic;
5         en : in std_ulogic;
6         q : out std_ulogic);
7 end entity;
8
9 architecture arch of dlatcl_r is
10    process(en, d, res_n)
11    begin
12        if res_n = '0' then
13            q <= '0';
14        elsif en = '1' then
15            q <= d;
16        end if;
17    end process;
18 end architecture;

```

Next, the code that tests the enable signal goes in the else-if-branch.

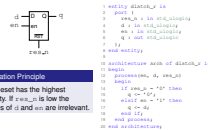
D Latch - Reset



```

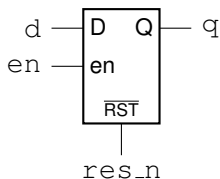
1 entity dlatcl_r is
2     port (
3         res_n : in std_ulogic;
4         d : in std_ulogic;
5         en : in std_ulogic;
6         q : out std_ulogic);
7 end entity;
8
9
10 architecture arch of dlatcl_r is
11 begin
12     process(en, d, res_n)
13     begin
14         if res_n = '0' then
15             q <= '0';
16         elsif en = '1' then
17             q <= d;
18         end if;
19     end process;
20 end architecture;

```



Note that this means that the enable signal is only evaluated, if the reset is not active. Hence, the reset always overrides the internal state of the latch.

D Latch - Reset



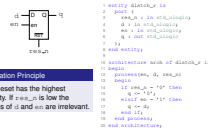
```

1 entity dlatcl_r is
2   port (
3     res_n : in std_ulogic;
4     d : in std_ulogic;
5     en : in std_ulogic;
6     q : out std_ulogic
7   );
8 end entity;
9
10 architecture arch of dlatcl_r is
11 begin
12   process(en, d, res_n)
13   begin
14     if res_n = '0' then
15       q <= '0';
16     elsif en = '1' then
17       q <= d;
18     end if;
19   end process;
20 end architecture;

```

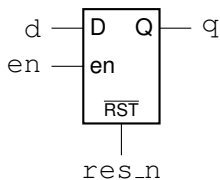
Operation Principle

The reset has the highest priority. If `res_n` is low the values of `d` and `en` are irrelevant.



Let us now turn our attention to the D flip-flop.

D Latch - Reset



Operation Principle

The reset has the highest priority. If `res_n` is low the values of `d` and `en` are irrelevant.

```

1 entity dlatcl_r is
2     port (
3         res_n : in std_ulogic;
4         d : in std_ulogic;
5         en : in std_ulogic;
6         q : out std_ulogic
7     );
8 end entity;
9
10 architecture arch of dlatcl_r is
11 begin
12     process(en, d, res_n)
13     begin
14         if res_n = '0' then
15             q <= '0';
16         elsif en = '1' then
17             q <= d;
18         end if;
19     end process;
20 end architecture;
    
```

Sequential Circuit Elements in VHDL

D Flip-Flop

D Flip-Flop



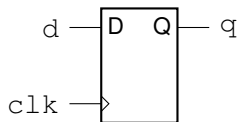
```
1  
2 entity dff is  
3   port ( d : in  std_ulogic;  
4         q : out std_ulogic;  
5         clk : in  std_ulogic;  
6         reset : in std_ulogic;  
7         enable : in std_ulogic;  
8 end entity;
```

Like with the D latch, the entity needs an input, *d*, as well as the output, *q*.

D Flip-Flop

HWMMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable



```
1  
2 entity dff is  
3   port (  
4     clk : in  std_ulogic;  
5     d   : in  std_ulogic;  
6     q   : out std_ulogic  
7   );  
8 end entity;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

D Flip-Flop



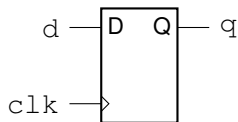
```
1 entity dff is
2   port (
3     d : in  std_ulogic;
4     q : out std_ulogic;
5     clk : in  std_ulogic;
6   );
7 end entity;
```

However, instead of the enable signal, the D flip-flop needs a clock input to trigger its operation.

D Flip-Flop

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable



```
1
2 entity dff is
3   port (
4     clk : in  std_ulogic;
5     d   : in  std_ulogic;
6     q   : out std_ulogic
7   );
8 end entity;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

D Flip-Flop



```
1  
2 entity dff is  
3     port (  
4         clk : in  std_ulogic;  
5         d   : in  std_ulogic;  
6         q   : out std_ulogic  
7     );  
8 end dff;
```

Problem

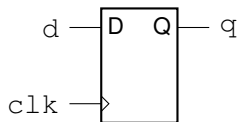
How can we detect the event of a signal transition?

Furthermore, the flip-flop needs to react to signal events, that is transitions, rather than the state of a signal. This requires the introduction of a new VHDL language feature. So let's have a look at what VHDL is offering for that purpose.

D Flip-Flop

HWMMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable



```
1  
2 entity dff is  
3     port (  
4         clk : in  std_ulogic;  
5         d   : in  std_ulogic;  
6         q   : out std_ulogic  
7     );  
8 end entity;
```

Problem

How can we detect the event of a signal transition?

For the moment, let's assume that we have a helper function `rising_edge` that takes a signal of type `std_ulogic` and returns a `boolean`. The function shall only return `true` when a rising edge has just occurred on signal `S` and `false` otherwise.

D Flip-Flop

■ Helper function

```
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];
4 end function;
```

```

■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];
4 end function;

■ D flip-flop architecture
9 architecture arch of dff is
10 begin
11   process (clk)
12     begin
13       if rising_edge(clk) then
14         q <= d;
15       end if;
16     end process;
17 end architecture;

```

Using this helper function we can then implement the architecture of the D flip-flop in a straight-forward manner.

D Flip-Flop

HWMoD
WS25

■ Helper function

```

1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];
4 end function;

```

■ D flip-flop architecture

```

9 architecture arch of dff is
10 begin
11   process (clk)
12     begin
13       if rising_edge(clk) then
14         q <= d;
15       end if;
16     end process;
17 end architecture;

```

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

```
■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return s'falling_edge;
4 end rising_edge;

■ D flip-flop architecture
9 architecture arch of dff is
10 begin
11
12
13
14
15
16
17 end architecture;
```

As with the D latch we first create a process.

D Flip-Flop

■ Helper function

```
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];
4 end function;
```

■ D flip-flop architecture

```
9 architecture arch of dff is
10 begin
11
12
13
14
15
16
17 end architecture;
```

```

■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3     return s'event and s = '1';
4 end function;

■ D flip-flop architecture
5 architecture arch of dff is
6     signal q;
7     process (clk)
8     begin
9         if rising_edge(clk) then
10            q <= d;
11        end if;
12    end process;
13 end architecture;

```

However, now, we only need the clock signal in the sensitivity list. However, since the output of the flip-flop must only change on rising edges of the clock signals, it suffices to make the process sensitive to `clk`.

D Flip-Flop

■ Helper function

```

1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3     return [...];
4 end function;

```

■ D flip-flop architecture

```

9 architecture arch of dff is
10 begin
11     process (clk)
12     begin
13         if rising_edge(clk) then
14             q <= d;
15         end if;
16     end process;
17 end architecture;

```

```

■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3     return s'falling_edge;
4 end function;

■ D flip-flop architecture
9 architecture arch of dff is
10 begin
11     process (clk)
12     begin
13         if rising_edge(clk) then
14             q <= d;
15         end if;
16     end process;
17 end architecture;
    
```

Then we use an if-condition and our helper function to check if a rising clock edge occurred.

D Flip-Flop

HWMoD
WS25

■ Helper function

```

1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3     return [...];
4 end function;
    
```

■ D flip-flop architecture

```

9 architecture arch of dff is
10 begin
11     process (clk)
12     begin
13         if rising_edge(clk) then
14             q <= d;
15         end if;
16     end process;
17 end architecture;
    
```

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

```
■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return s'event and s = '1';
4 end function;

■ D flip-flop architecture
5 architecture arch of dff is
6 begin
7   process (clk)
8   begin
9     if rising_edge(clk) then
10      q <= d;
11    end if;
12   end process;
13 end architecture;
```

If this is the case we assign d to q.

D Flip-Flop

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

■ Helper function

```
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];
4 end function;
```

■ D flip-flop architecture

```
9 architecture arch of dff is
10 begin
11   process (clk)
12   begin
13     if rising_edge(clk) then
14       q <= d;
15     end if;
16   end process;
17 end architecture;
```

```

■ Helper function
1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return s'edge;
4 end function;

■ D flip-flop architecture
5 architecture arch of dff is
6 begin
7   process (clk)
8     if rising_edge(clk) then
9       q <= d;
10    end if;
11  end process;
12 end architecture;

```

The only question that remains is how the body of the `rising_edge` function is supposed to be implemented. In other words: What expression must be inserted instead of the place-holder in the function body?

D Flip-Flop

HWMoD
WS25

■ Helper function

```

1 function rising_edge(signal s : std_ulogic) return boolean is
2 begin
3   return [...];?
4 end function;

```

■ D flip-flop architecture

```

9 architecture arch of dff is
10 begin
11   process (clk)
12   begin
13     if rising_edge(clk) then
14       q <= d;
15     end if;
16   end process;
17 end architecture;

```

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

Sequential Circuit Elements in VHDL

D Flip-Flop

Signal Edges – event Attribute

■ Special predefined signal attribute: s'event ◆

Luckily for us, the VHDL standard defines the `event` attribute for signals which we can use for the implementation of the `rising_edge` function.

Signal Edges – event Attribute

278

HWMod
WS25

Seq. Elem.

Introduction

D Latches

D Flip-Flop

Signal Edges

Reset

Enable

- Special predefined signal attribute: s'event ◆

Sequential Circuit Elements in VHDL

D Flip-Flop

Signal Edges – event Attribute

- Special predefined `signal` attribute: `s'event` ◆
 - VHDL standard
- "s'event returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`."*

As stated by the short excerpt from the VHDL standard that's shown on the slide, the `event` attribute returns the Boolean value `true` whenever there occurred an event during the current simulation cycle. If no event occurred, the attribute will return `false`.

Signal Edges – event Attribute

278

HWMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- Special predefined `signal` attribute: `s'event` ◆
- VHDL standard

"s'event returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`."

Sequential Circuit Elements in VHDL

D Flip-Flop

Signal Edges – event Attribute

- Special predefined `signal` attribute: `s'event` ◆
- VHDL standard
“`s'event` returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`.”
- Possible edge detection expression
`s'event and s = '1'`

We can harness this attribute to express a simple condition for detecting rising edges. This is shown on the slide. The Boolean expression will evaluate to true whenever the signal `s` transitioned to '1' in the current cycle. You might want to pause the video at this point and think about whether this expression leads to undesired behavior.

Signal Edges – event Attribute

278

HWMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- Special predefined `signal` attribute: `s'event` ◆
- VHDL standard

“`s'event` returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`.”

- Possible edge detection expression
`s'event and s = '1'`

Sequential Circuit Elements in VHDL

D Flip-Flop

Signal Edges – event Attribute

- Special predefined `signal` attribute: `s'event` ◆
- VHDL standard
- `s'event` returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`.
- Possible edge detection expression
- What if `clk` changes from, e.g., '0' to '1'?

As you probably detected yourself, a potential issue of this expression is that it detects arbitrary changes to the value '1'. For example, it will also evaluate to `true` when the clock signal changes from '0' to '1'. Naturally, this does not correspond to a rising clock edge. How can we deal with this issue?

Signal Edges – event Attribute

278

HWMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- Special predefined `signal` attribute: `s'event` ◆
- VHDL standard

“`s'event` returns the value `true` if an event has occurred on `s` during the current simulation cycle; otherwise, it returns the value `false`.”

- Possible edge detection expression
- What if `clk` changes from, e.g., '0' to '1'?

Again, the VHDL standard comes to the rescue with an attribute. In case an event already occurred on a signal s , the last_value attribute returns the value of the signal before the most recent event.

Signal Edges – last_value Attribute

279

HWMod
WS25

Seq. Elem.

Introduction

D Latches

D Flip-Flop

Signal Edges

Reset

Enable

■ VHDL standard

“For a signal s , if an event has occurred on s in any simulation cycle, s ' last_value returns the value of s prior to the update of s in the last simulation cycle in which an event occurred; otherwise, s ' last_value returns the current value of s .”

■ VHDL standard

“For a signal s , if an event has occurred on s in any simulation cycle, s' last_value returns the value of s prior to the update of s in the last simulation cycle in which an event occurred; otherwise, s' last_value returns the current value of s .”

■ Improved edge detection expression

`s'event and (s = '1') and (s'last_value = '0')`

We can use this attribute to refine our edge detection expression such that it only evaluates to `true` whenever an event occurred on the respective signal and this event led to a transition from the value '0' to '1'. So, are we done? Is this expression valid for our purpose? Unfortunately not.

Signal Edges – last_value Attribute

279

HWMoD
WS25

Seq. Elem.

Introduction

D Latches

D Flip-Flop

Signal Edges

Reset

Enable

■ VHDL standard

“For a signal s , if an event has occurred on s in any simulation cycle, s' last_value returns the value of s prior to the update of s in the last simulation cycle in which an event occurred; otherwise, s' last_value returns the current value of s .”

■ Improved edge detection expression

`s'event and (s = '1') and (s'last_value = '0')`

- VHDL standard
 - “For a signal *s*, if an event has occurred on *s* in any simulation cycle, *s*' last_value returns the value of *s* prior to the update of *s* in the last simulation cycle in which an event occurred; otherwise, *s*' last_value returns the current value of *s*.”
- Improved edge detection expression
 - s*'event and (*s* = '1') and (*s*'last_value = '0')
- What if *clk* changes from 'L' to 'H'?

The improved edge detection expression still comes with potential issues. Recall that our signals are in general actually not binary, as they are instances of the nine-valued logic. Within these nine values, the transition from '0' to '1' is not the only valid rising edge. For example, a transition from 'L' to 'H' should also be detected.

Signal Edges – last_value Attribute

279

HWMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

■ VHDL standard

*“For a signal *s*, if an event has occurred on *s* in any simulation cycle, *s*' last_value returns the value of *s* prior to the update of *s* in the last simulation cycle in which an event occurred; otherwise, *s*' last_value returns the current value of *s*.”*

■ Improved edge detection expression

s'event and (*s* = '1') and (*s*'last_value = '0')

■ What if *clk* changes from 'L' to 'H'?

A simple way to handle this problem is by reducing the set of values we need to consider in our `rising_edge` function. We do so by mapping the nine values to their binary equivalents if possible, and otherwise to 'X'.

Signal Edges

- First convert the signal values to '0' or '1' (or 'X' if not possible)

For that purpose the `std_logic_1164` package provides the `two_X01` function, performing the mapping shown on the slide.

Signal Edges

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- First convert the signal values to '0' or '1' (or 'X' if not possible)
- `two_X01` function IEEE SA
OPEN
 - 'U', 'X', 'Z', 'W', '-' → 'X'
 - '0', 'L' → '0'
 - '1', 'H' → '1'

Using this function and the previously introduced attributes, we can now finally write down an expression to detect rising edges of a signal. This expression is simply the one from the previous slide, but using the new mapping function.

Signal Edges

- First convert the signal values to '0' or '1' (or 'X' if not possible)
- to_X01 function IEEE SA
OPEN
 - 'U', 'X', 'Z', 'W', '-' → 'X'
 - '0', 'L' → '0'
 - '1', 'H' → '1'
- Final edge detection expression
`s'event and (to_X01(s) = '1') and`
`(to_X01(s'last_value) = '0')`

Note that this expression is exactly the one used in the IEEE implementation and is not subject to any of the issues we mentioned before. For the sake of completeness, note that the IEEE also defines a similar function for detecting falling edges.

Signal Edges

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- First convert the signal values to '0' or '1' (or 'X' if not possible)
- to_X01 function IEEE SA OPEN
 - 'U', 'X', 'Z', 'W', '-' → 'X'
 - '0', 'L' → '0'
 - '1', 'H' → '1'
- Final edge detection expression


```
s'event and (to_X01(s) = '1') and
              (to_X01(s'last_value) = '0')
```
- rising/falling_edge as defined in std_logic_1164 package IEEE SA OPEN

- First convert the signal values to '0' or '1' (or 'X' if not possible)
- to_X01 function
 - to_X01('0') = '0'
 - to_X01('1') = '1'
 - to_X01('X') = 'X'
 - to_X01('U') = 'X'
 - to_X01('Z') = 'X'
 - to_X01('L') = '0'
 - to_X01('H') = '1'
- Final edge detection expression


```
s'event and (to_X01(s) = '1') and
              (to_X01(s'last_value) = '0')
```
- rising/falling_edge as defined in std_logic_1164 package
- The standard package defines rising/falling_edge for the types
 - bit
 - boolean

Finally, we want to mention that the rising and falling edge functions are not only defined for signals of the type `std_logic`. In particular, the `standard` package also defines versions for the `bit` and `boolean` types.

Signal Edges

- First convert the signal values to '0' or '1' (or 'X' if not possible)
- to_X01 function IEEE SA OPEN
 - 'U', 'X', 'Z', 'W', '-' → 'X'
 - '0', 'L' → '0'
 - '1', 'H' → '1'
- Final edge detection expression


```
s'event and (to_X01(s) = '1') and
              (to_X01(s'last_value) = '0')
```
- rising/falling_edge as defined in std_logic_1164 package IEEE SA OPEN
- The standard package defines rising/falling_edge for the types
 - bit
 - boolean

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset

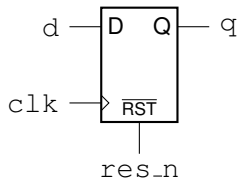


Just as for the latch, resetting a flip-flop is usually necessary. Again, we use a dedicated reset input for that. The slide shows the respective symbol and entity declaration. However, whereas the semantics of the reset are rather obvious for a latch, they are less clear for a flip-flop.

Reset

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable



```
1 entity dff_r is
2   port (
3     clk      : in  std_ulogic;
4     res_n    : in  std_ulogic;
5     d        : in  std_ulogic;
6     q        : out std_ulogic
7   );
8 end entity;
```

Sequential Circuit Elements in VHDL

- D Flip-Flop
- Reset



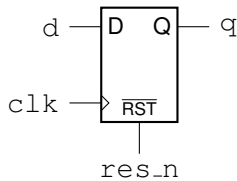
Reset Condition
When is the reset evaluated?

In particular: Is the reset evaluated only at the active clock edges of the flip-flop, or is it level-sensitive and resets the flip-flop independent of the clock?

Reset

HWMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable



```
1 entity dff_r is
2   port (
3     clk : in  std_ulogic;
4     res_n : in std_ulogic;
5     d : in  std_ulogic;
6     q : out std_ulogic
7   );
8 end entity;
```

Reset Condition

When is the reset evaluated?

The answer is that both interpretations of the reset are possible and sensible. Hence, depending on the desired behavior, a flip-flop can in general be equipped with either reset type.

Reset (Cont'd)

Synchronous Reset

Asynchronous Reset

HWMMod
WS25

Seq. Elem.

Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

A reset that is only evaluated at the active clock edge is called a *synchronous reset* for obvious reasons.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge

```
9 architecture sync of dff_r is
10 begin
11
12
13
14
15
16
17
18
19
20
21 end architecture;
```

Asynchronous Reset

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)



The other reset flavor is the *asynchronous reset*, which will reset the flip-flop *whenever* the reset is active, independently of the clock. Let us consider how the two kinds of resets can be described in VHDL.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge

```
9 architecture sync of dff_r is
10 begin
11
12
13
14
15
16
17
18
19
20
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive

```
9 architecture async of dff_r is
10 begin
11
12
13
14
15
16
17
18
19 end architecture;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)

Synchronous Reset	Asynchronous Reset
■ Reset signal is only evaluated at the active clock edge.	■ Reset signal is level-sensitive

```
1 architecture sync of dff_r is
2   process(clk)
3   begin
4     if rising_edge(clk) then
5       if res_n = '0' then
6         q <= '0';
7       else
8         q <= d;
9       end if;
10    end if;
11  end process;
12 end architecture;
```

```
1 architecture async of dff_r is
2   process
3   begin
4     if res_n = '0' then
5       q <= '0';
6     else
7       q <= d;
8     end if;
9   end process;
10 end architecture;
```

We'll start with the synchronous one.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge

```
9 architecture sync of dff_r is
10 begin
11   process(clk)
12   begin
13     if rising_edge(clk) then
14       if res_n = '0' then
15         q <= '0';
16       else
17         q <= d;
18       end if;
19     end if;
20   end process;
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive

```
9 architecture async of dff_r is
10 begin
11
12
13
14
15
16
17
18
19 end architecture;
```

HWMoD
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)

```
Synchronous Reset
■ Reset signal is only evaluated at
  the active clock edge
■ res_n not in sensitivity list

Asynchronous Reset
■ Reset signal is level-sensitive
```

When describing a synchronous reset in VHDL, it is important to omit the reset from the respective sensitivity list. The reason is of course that we only want clock edges to result in an output change and hence to trigger the process.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge
- `res_n` **not** in sensitivity list

```
9 architecture sync of dff_r is
10 begin
11   process (clk)
12   begin
13     if rising_edge(clk) then
14       if res_n = '0' then
15         q <= '0';
16       else
17         q <= d;
18       end if;
19     end if;
20   end process;
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive

```
9 architecture async of dff_r is
10 begin
11
12
13
14
15
16
17
18
19 end architecture;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)

```
Synchronous Reset
■ Reset signal is only evaluated at
  the active clock edge
■ res_n not in sensitivity list

Asynchronous Reset
■ Reset signal is level-sensitive
```

Also note that the reset condition check is nested within the condition for the rising clock edge. Let us now implement the asynchronous reset.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge
- `res_n` **not** in sensitivity list

```
9 architecture sync of dff_r is
10 begin
11   process (clk)
12   begin
13     if rising_edge(clk) then
14       if res_n = '0' then
15         q <= '0';
16       else
17         q <= d;
18       end if;
19     end if;
20   end process;
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive

```
9 architecture async of dff_r is
10 begin
11
12
13
14
15
16
17
18
19 end architecture;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)

```
Synchronous Reset
■ Reset signal is only evaluated at
the active clock edge
■ res_n not in sensitivity list

Asynchronous Reset
■ Reset signal is level-sensitive
■ res_n in sensitivity list
```

The first difference is the sensitivity list. Since an asynchronous reset should take immediate effect, regardless of the clock, the process must be sensitive to the reset as well.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge
- `res_n` **not** in sensitivity list

```
9 architecture sync of dff_r is
10 begin
11     process (clk)
12     begin
13         if rising_edge(clk) then
14             if res_n = '0' then
15                 q <= '0';
16             else
17                 q <= d;
18             end if;
19         end if;
20     end process;
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive
- `res_n` in sensitivity list

```
9 architecture async of dff_r is
10 begin
11     process (clk, res_n)
12     begin
13         if res_n = '0' then
14             q <= '0';
15         elsif rising_edge(clk) then
16             q <= d;
17         end if;
18     end process;
19 end architecture;
```

Sequential Circuit Elements in VHDL

D Flip-Flop

Reset (Cont'd)

```
Synchronous Reset
■ Reset signal is only evaluated at
  the active clock edge
1 architecture sync of dff_r is
2   process(clk)
3   begin
4     if rising_edge(clk) then
5       if res_n = '0' then
6         q <= '0';
7       else
8         q <= d;
9       end if;
10    end if;
11  end process;
12 end architecture;

Asynchronous Reset
■ Reset signal is level-sensitive
1 architecture async of dff_r is
2   process(clk, res_n)
3   begin
4     if res_n = '0' then
5       q <= '0';
6     elsif rising_edge(clk) then
7       q <= d;
8     end if;
9   end process;
10 end architecture;
```

Furthermore, the reset condition must not be nested within the condition for the rising edge, as otherwise the process would be triggered but would not reset the flip-flop independent of the clock.

Reset (Cont'd)

Synchronous Reset

- Reset signal is only evaluated at the active clock edge
- `res_n` **not** in sensitivity list

```
9 architecture sync of dff_r is
10 begin
11   process(clk)
12   begin
13     if rising_edge(clk) then
14       if res_n = '0' then
15         q <= '0';
16       else
17         q <= d;
18       end if;
19     end if;
20   end process;
21 end architecture;
```

Asynchronous Reset

- Reset signal is level-sensitive
- `res_n` in sensitivity list

```
9 architecture async of dff_r is
10 begin
11   process(clk, res_n)
12   begin
13     if res_n = '0' then
14       q <= '0';
15     elsif rising_edge(clk) then
16       q <= d;
17     end if;
18   end process;
19 end architecture;
```

What if we want a flip-flop that does not update its value in each clock cycle? For example, a circuit might be in a state where some of its flip-flops or registers must hold their values, although their inputs might change.

Enable Input

- What if a flip-flop should not be updated each clock cycle?

HWMMod
WS25

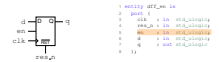
Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

Sequential Circuit Elements in VHDL

D Flip-Flop

Enable Input

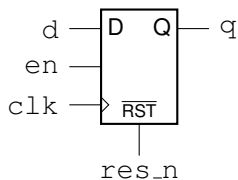
- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



We can achieve this by using a dedicated enable input in addition to the clock, reset and data inputs. A respective flip-flop symbol and VHDL entity are shown on the slide.

Enable Input

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



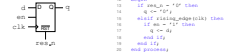
```
1 entity dff_en is
2   port (
3     clk : in  std_ulogic;
4     res_n : in  std_ulogic;
5     en : in  std_ulogic;
6     d : in  std_ulogic;
7     q : out std_ulogic
8   );
```

Sequential Circuit Elements in VHDL

D Flip-Flop

Enable Input

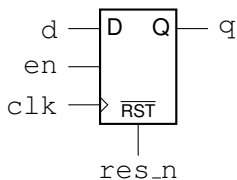
- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



Let us now quickly look at a possible implementation of the desired behavior.

Enable Input

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



```

11 process(clk, res_n)
12 begin
13   if res_n = '0' then
14     q <= '0';
15   elsif rising_edge(clk) then
16     if en = '1' then
17       q <= d;
18     end if;
19   end if;
20 end process;

```

HWMMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

Sequential Circuit Elements in VHDL

D Flip-Flop

Enable Input

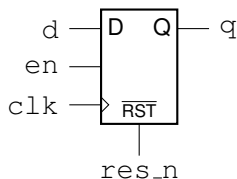
- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



In the shown process, the condition for the enable signal is contained within the body of the `elsif` branch. This results in the enable only taking effect at rising clock edges. Note that this is not the only way such a behavior can be implemented, but it is the one we recommend.

Enable Input

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal



```
11 process(clk, res_n)
12 begin
13   if res_n = '0' then
14     q <= '0';
15   elsif rising_edge(clk) then
16     if en = '1' then
17       q <= d;
18     end if;
19   end if;
20 end process;
```

HWMMod
WS25

Seq. Elem.
Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal
- Structure matters!
 - Always use the patterns shown in this lecture



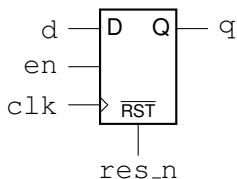
```

11 process (clk, res_n)
12 begin
13     if res_n = '0' then
14         q <= '0';
15     elsif rising_edge (clk) then
16         if en = '1' then
17             q <= d;
18         end if;
19     end if;
20 end process;
    
```

This recommendation does not only hold for the currently shown, but for all the different flavors of sequential elements you just saw. It is vital that you strictly stick to the code patterns for latches and flip-flops presented in this lecture. Do not experiment with custom flip-flop descriptions!

Enable Input

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal
- Structure matters!
 - **Always** use the patterns shown in this lecture



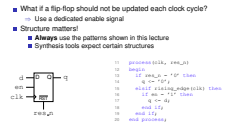
```

11 process (clk, res_n)
12 begin
13     if res_n = '0' then
14         q <= '0';
15     elsif rising_edge (clk) then
16         if en = '1' then
17             q <= d;
18         end if;
19     end if;
20 end process;
    
```

Sequential Circuit Elements in VHDL

D Flip-Flop

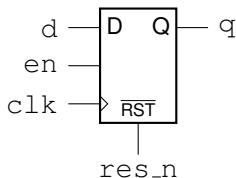
Enable Input



It might be the case that such structures work in simulation but fail to synthesize or, even worse, result in a circuit that does not match the intended design. Synthesis tools look for the code patterns presented here and will, if you stick to them, generate correct circuits! Please always keep that in mind when writing VHDL code!

Enable Input

- What if a flip-flop should not be updated each clock cycle?
 - ⇒ Use a dedicated enable signal
- Structure matters!
 - Always** use the patterns shown in this lecture
 - Synthesis tools expect certain structures



```
11 process (clk, res_n)
12 begin
13   if res_n = '0' then
14     q <= '0';
15   elsif rising_edge(clk) then
16     if en = '1' then
17       q <= d;
18     end if;
19   end if;
20 end process;
```

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

Seq. Elem.

Introduction
D Latches
D Flip-Flop
Signal Edges
Reset
Enable

Lecture Complete!