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

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 16:04 (f8a58e9)

Introduction

- Combinational logic cannot retain any data $\Rightarrow$ Sequential logic

Introduction

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

Introduction

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

Introduction

- Combinational logic cannot retain any data $\Rightarrow$ 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!

Introduction

- Combinational logic cannot retain any data $\Rightarrow$ 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**

Introduction

- Combinational logic cannot retain any data $\Rightarrow$ 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

Introduction (cont'd)

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.

Introduction (cont'd)

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.

Introduction (cont'd)

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.

Introduction (cont'd)

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.

Introduction (cont'd)

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.

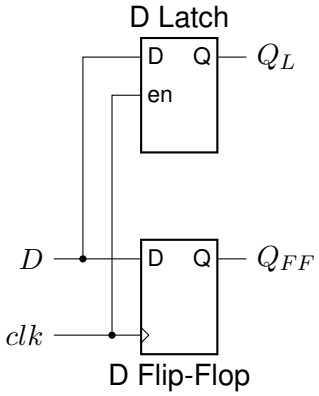
D Flip-Flop

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.

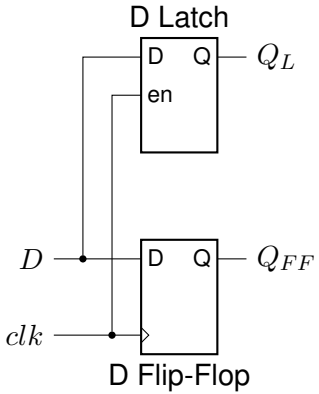
Register

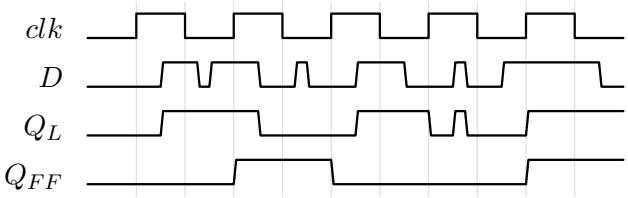
Registers are collections of D flip-flops (latches) that hold data that logically belong together.

D Latches vs. D Flip-Flop

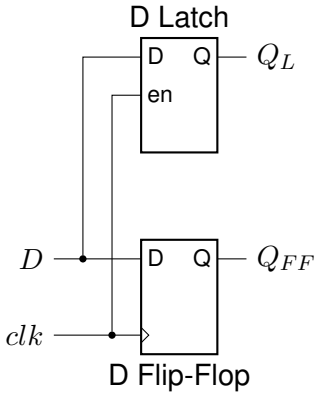


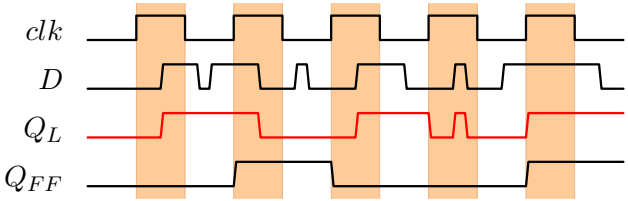
D Latches vs. D Flip-Flop



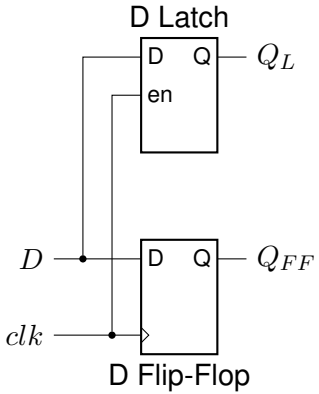


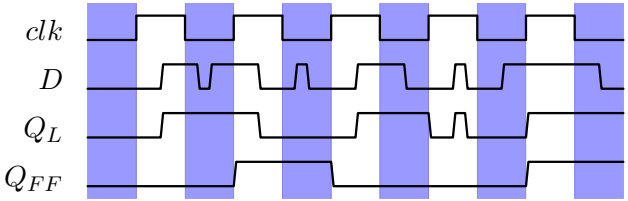
D Latches vs. D Flip-Flop



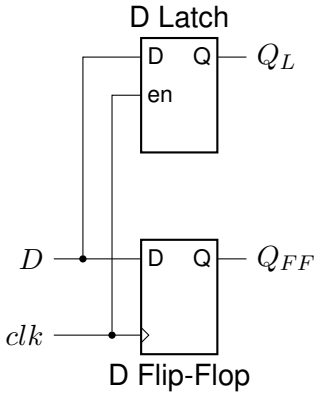


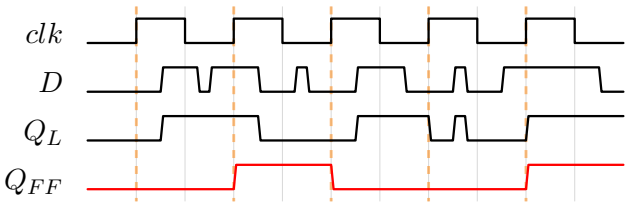
D Latches vs. D Flip-Flop



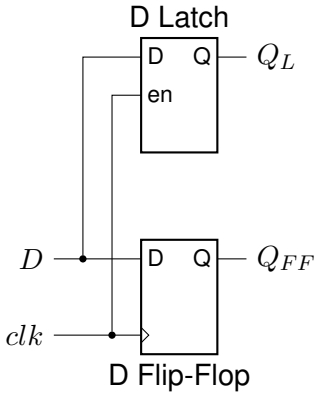


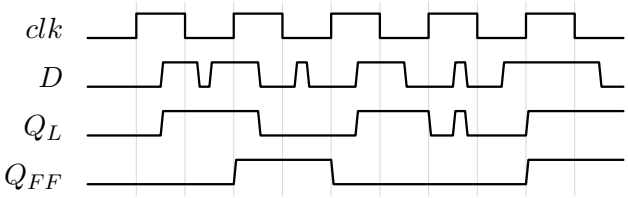
D Latches vs. D Flip-Flop





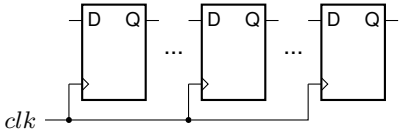
D Latches vs. D Flip-Flop





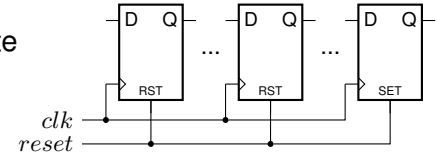
Reset Signal

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



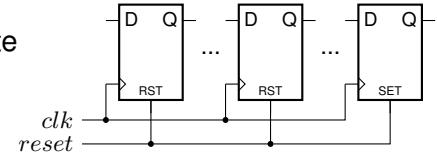
Reset Signal

- 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



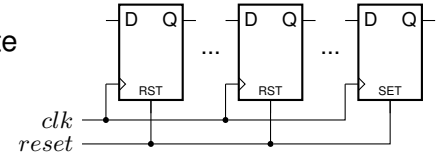
Reset Signal

- 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



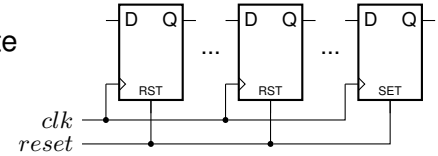
Reset Signal

- 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

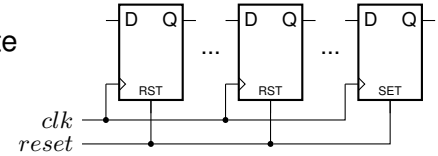


Reset Signal

- 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!

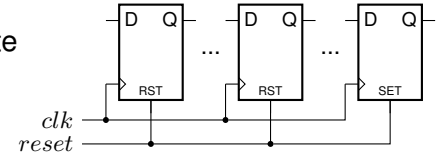


Reset Signal



- 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)

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

Active Signal Levels

Active Signal Levels

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.

Active Signal Levels

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., $\overline{\text{en}}$)
  - Code: the suffix `_n` (e.g., `en_n`)

Active Signal Levels

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., $\overline{\text{en}}$)
  - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets

Active Signal Levels

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., $\overline{\text{en}}$)
  - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
  - Power-up default often *ground*

Active Signal Levels

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., $\overline{\text{en}}$)
  - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
  - Power-up default often *ground*
  - Noise immunity

Active Signal Levels

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., $\overline{\text{en}}$)
  - 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

Active Signal Levels

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., $\overline{\text{en}}$)
  - 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`.

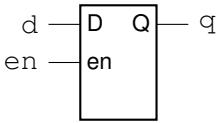
Active Signal Levels

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.

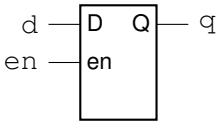
- Notation for active-low signals:
  - Circuit diagrams: a bar above the signal name (e.g., $\overline{\text{en}}$)
  - Code: the suffix `_n` (e.g., `en_n`)
- Active-low resets
  - Power-up default often *ground*
  - Noise immunity
  - Widely adopted in industry for reliability and compatibility
  -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

D Latches



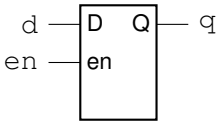

```vhdl
entity dlatch is
  port (
    d : in std_ulogic;
    en : in std_ulogic;
    q : out std_ulogic
  );
end entity;

architecture arch of dlatch is
begin




end architecture;
```

D Latches



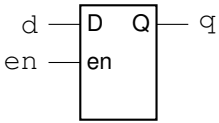

```
1  entity dlatch is
2    port (
3      d : in std_ulogic;
4      en : in std_ulogic;
5      q : out std_ulogic
6    );
7  end entity;
8
9  architecture arch of dlatch is
10 begin
11   process(en, d)
12   begin
13
14
15
16   end process;
17 end architecture;
```

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
16   end process;
17 end architecture;
```

D Latches

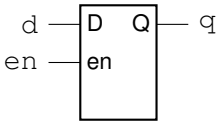


Operation Principle

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    );
7  end entity;
8
9  architecture arch of dlatch is
10 begin
11   process(en, d)
12   begin
13
14
15
16   end process;
17 end architecture;
```

D Latches

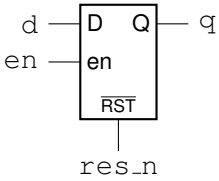


Operation Principle

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

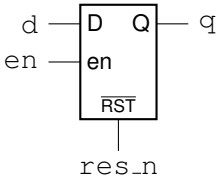
```
1  entity dlatch is
2    port (
3      d : in std_ulogic;
4      en : in std_ulogic;
5      q : out std_ulogic
6    );
7  end entity;
8
9  architecture arch of dlatch is
10 begin
11   process(en, d)
12   begin
13     if en = '1' then
14       q <= d;
15     end if;
16   end process;
17 end architecture;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

D Latch - Reset



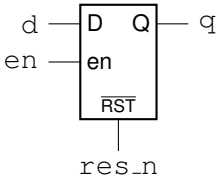

```
1  entity dlatch_r is
2    port (
3      res_n : in std_ulogic;
4      d : in std_ulogic;
5      en : in std_ulogic;
6      q : out std_ulogic
7    );
8  end entity;
9
10 architecture arch of dlatch_r is
11 begin
12   process(en, d, res_n)
13   begin
14
15
16
17
18
19   end process;
20 end architecture;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

D Latch - Reset



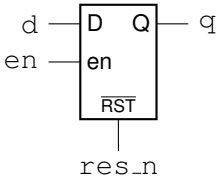

```vhdl
entity dlatch_r is
  port (
    res_n : in std_ulogic;
    d : in std_ulogic;
    en : in std_ulogic;
    q : out std_ulogic
  );
end entity;

architecture arch of dlatch_r is
begin
  process(en, d, res_n)
  begin




  end process;
end architecture;
```

D Latch - Reset



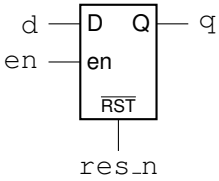

```
1  entity dlatch_r is
2    port (
3      res_n : in std_ulogic;
4      d : in std_ulogic;
5      en : in std_ulogic;
6      q : out std_ulogic
7    );
8  end entity;
9
10 architecture arch of dlatch_r is
11 begin
12   process(en, d, res_n)
13   begin
14     if res_n = '0' then
15       q <= '0';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

D Latch - Reset




```
1  entity dlatch_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 dlatch_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;
```

D Latch - Reset

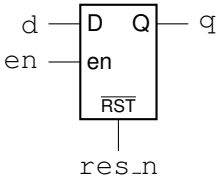


Operation Principle

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

```
1  entity dlatch_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 dlatch_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;
```

D Latch - Reset

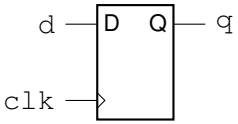


Operation Principle

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

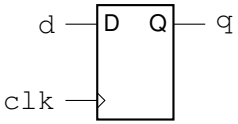
```
1  entity dlatch_r is
2    port (
3      res_n : in std_ulogic;
4      d : in std_ulogic;
5      en : in std_ulogic;
6      q : out std_ulogic
7    );
8  end entity;
9
10 architecture arch of dlatch_r is
11 begin
12   process(en, d, res_n)
13   begin
14     if res_n = '0' then
15       q <= '0';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

D Flip-Flop



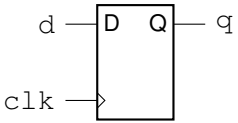

```
1
2 entity dff is
3   port (
4     clk : in  std_ulogic;
5     d   : in  std_ulogic;
6     q   : out std_ulogic
7   );
8 end entity;
```

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 entity;
```

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 entity;
```

Problem

How can we detect the event of a signal transition?

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

- 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;
```

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;
```

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;
```

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;
```

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;
```

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;
```

Signal Edges – `event` Attribute

- Special predefined `signal` attribute: `s'event` 

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`."

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'
```

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'
```

- What if `clk` changes from, e.g., `'U'` to `'1'`?

Signal Edges – `last_value` Attribute

- 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`."

Signal Edges – `last_value` Attribute

- 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')
```

Signal Edges – `last_value` Attribute

- 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'`?

Signal Edges

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

Signal Edges

- First convert the signal values to `'0'` or `'1'` (or `'X'` if not possible)
- `to_X01` function
  - `'U','X','Z','W','-'` → `'X'`
  - `'0','L'` → `'0'`
  - `'1','H'` → `'1'`

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')
```

Signal Edges

- First convert the signal values to `'0'` or `'1'` (or `'X'` if not possible)
- `to_X01` function
  - `'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

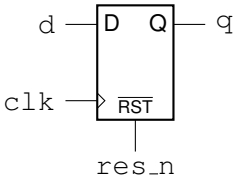
Signal Edges

- First convert the signal values to `'0'` or `'1'` (or `'X'` if not possible)
- `to_X01` function
  - `'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')
```

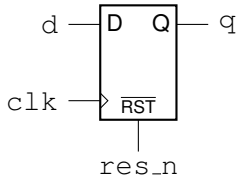
- `rising/falling_edge` as defined in `std_logic_1164` package
- The `standard` package defines `rising/falling_edge` for the types
  - `bit`
  - `boolean`

Reset




```
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




```
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?

Reset (Cont'd)

Synchronous Reset

Asynchronous Reset

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 (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;
```

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;
```

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;
```

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;
```

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;
```

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

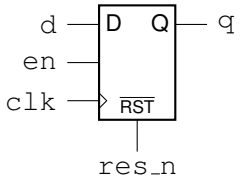
```
 9 architecture async of dff_r is
10 begin
11   process(clk, res_n)
12   begin
13     if res_n = '0' then
14       q <= '0';
15     elsif rising_edge(clk) then
16       q <= d;
17     end if;
18   end process;
19 end architecture;
```

Enable Input

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

Enable Input

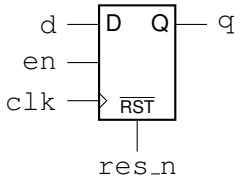
- What if a flip-flop should not be updated each clock cycle?
  - $\Rightarrow$ Use a dedicated enable signal




```
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   );
```

Enable Input

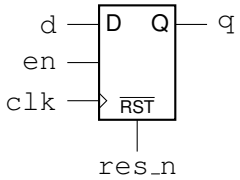
- What if a flip-flop should not be updated each clock cycle?
  - $\Rightarrow$ Use a dedicated enable signal




```
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;
```

Enable Input

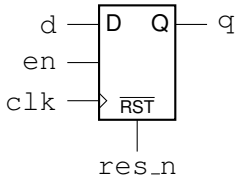
- What if a flip-flop should not be updated each clock cycle?
  - $\Rightarrow$ Use a dedicated enable signal




```
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;
```

Enable Input

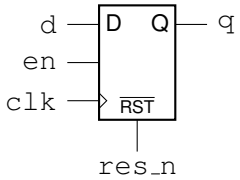
- What if a flip-flop should not be updated each clock cycle?
  - $\Rightarrow$ Use a dedicated enable signal
- Structure matters!
  - **Always** use the patterns shown in this lecture




```
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;
```

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;
```

Lecture Complete!