lab 02

Process

Digital design – laboratory exercises

assistant: Nejc Ilc

Process

```
<label>: process (<sensitivity_list>)
begin
   -- statements - sequential order
end
```

In `sensitivity_list` we put all signals that can cause changes in the outputs of the process.

"if" statement

- Use it only inside a process
- Syntax

```
if condition then
  -- statements
else
  -- statements
end if;
```

```
if condition_1 then
  -- statements
elsif condition_2 then
  -- statements
else
  -- statements
end if;
```

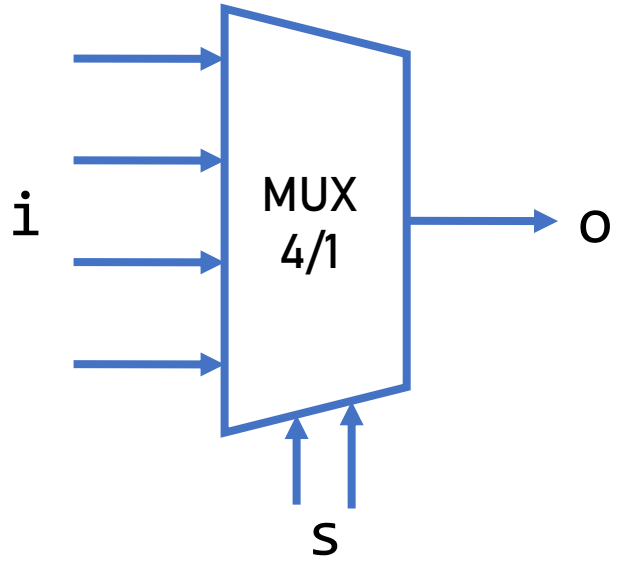
"case" statement

- Use it only inside a process
- Syntax

```
case s is
  when value_1 => output <= expression_1;
  when value_2 => output <= expression _2;
  ...
  when others => output <= expression _dfl;
end case;
```

- Last case (`when others`) is mandatory.

Example



Using "case":

```
process (i, s)
begin
  case s is
    when   "00" => o <= i(0);
    when   "01" => o <= i(1);
    when   "10" => o <= i(2);
    when   "11" => o <= i(3);
    when others => o <= i(0);
  end case;
end process;
```

Using conditonal assignment:

```
o <= i(0) when s="00" else
     i(1) when s="01" else
     i(2) when s="10" else
     i(3);
```

Using "select":

```
with s select
  o <= i(0) when "00",
       i(1) when "01",
       i(2) when "10",
       i(3);
```

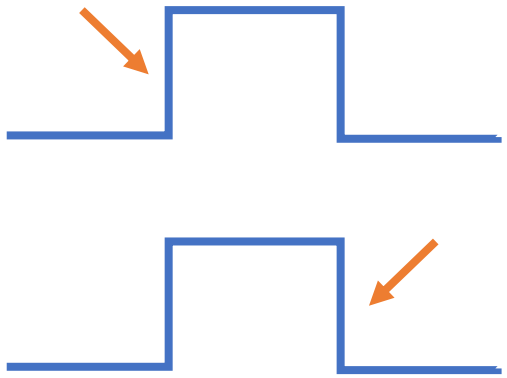
Sequential circuits

- In sequential circuits, changes occur at clock events (at the first/positive/rising or last/negative/falling edge)
- We detect an event on a clock signal `clk` using `clk'event`
  - `clk'event` is true, when there is an edge on signal
- Rising edge detection

  ```
  clk'event and clk='1'
  ```

- Falling edge detection

  ```
  clk'event and clk='0'
  ```



Example: D flip-flop

```
process(clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
end process;
```

Example: counter

```
process(clk)
  begin
    if clk'event and clk = '1' then
      if reset = '1' then
        count <= (others =>'0'); -- reset all bits on '0'
      else
        count <= count + 1;
      end if;
    end if;
end process;
```

Library IEEE: package STD_LOGIC_1164

- Package IEEE.STD_LOGIC_1164 med defines (among others) types STD_LOGIC and STD_LOGIC_VECTOR and functions as:
  - rising_edge
    instead of: clk'event and clk = '1'
  - falling_edge
    instead of: clk'event and clk = '0'

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Library IEEE: package NUMERIC_STD

```
use IEEE.NUMERIC_STD.all;
```

- Package IEEE.NUMERIC_STD defines types `signed` and `unsigned` and the corresponding arithmetic and logical operations on them:

  `+, -, *`

  `=, /=, <, <=, >, >=`

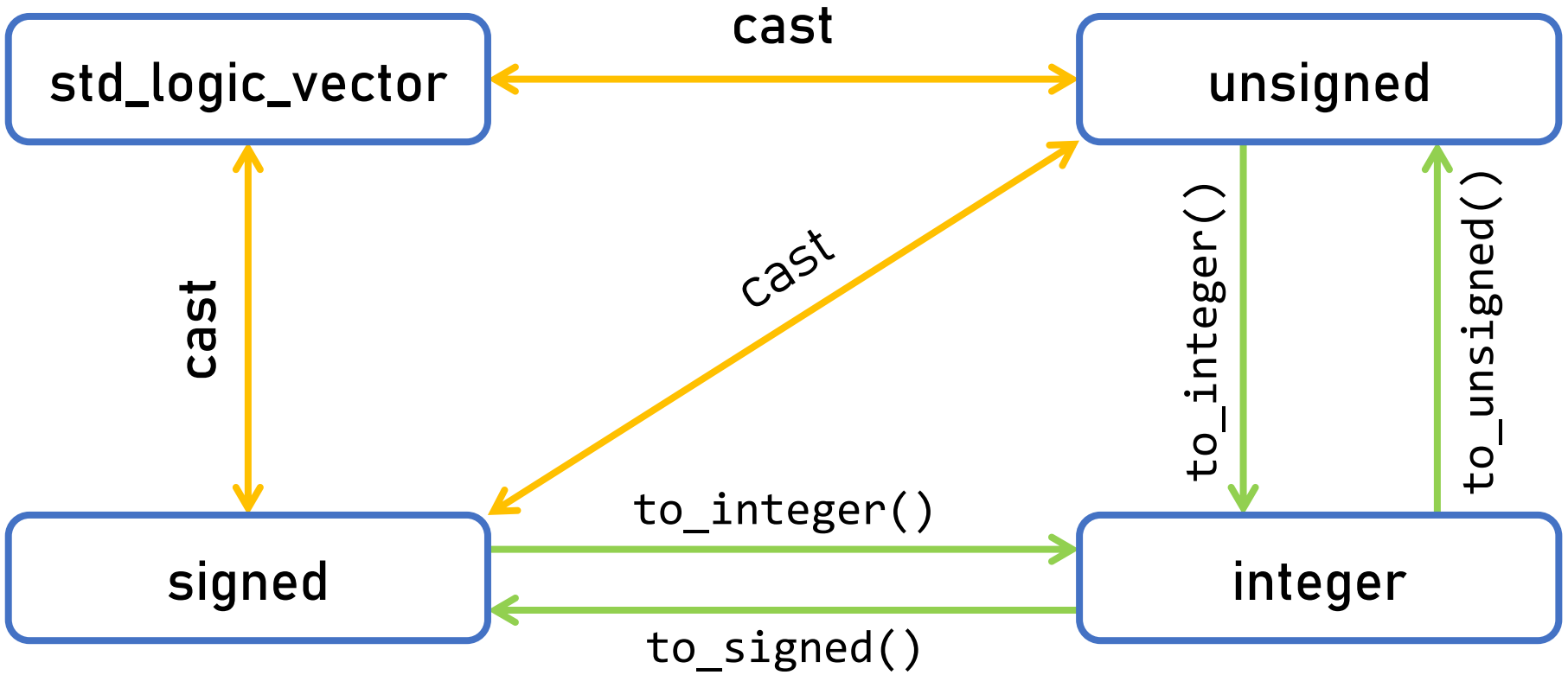
  `shift_left(op1, op2), shift_right(op1, op2)`

  - here, `op1` is of type `(un)signed` and `op2` is of type `integer`

Library IEEE: package NUMERIC_STD
Type conversion

- When using arithmetic operators, we often have to use type conversion, e.g.:
  - std_logic_vector ←→ signed/unsigned
  - std_logic_vector ←→ integer
  - integer ←→ signed/unsigned
- We use
  - casting, e.g.:
    - unsigned(signal_std_logic_vector)
    - std_logic_vector(signal_unsigned)
  - and conversion functions, e.g.:
    - to_integer(signal_unsigned)
    - to_signed(signal_integer, bit_width)
- Code examples: https://nandland.com/common-vhdl-conversions

Library IEEE: package NUMERIC_STD
Type conversion



Challenge

Describe a module in VHDL that will generate the "scrolling LEDs" pattern:

- a block of four LEDs is scrolling (four LEDs are on)
- the first switch determines the direction of scrolling
  - when the switch is on, the direction of the scrolls is reversed (left to right)
- the second switch changes the scroll speed
  - when the switch is on, the pattern moves every 0.5 seconds, otherwise every 1 second
- the middle button (BNTC) resets the circuit