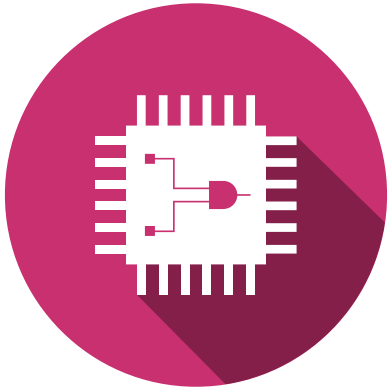
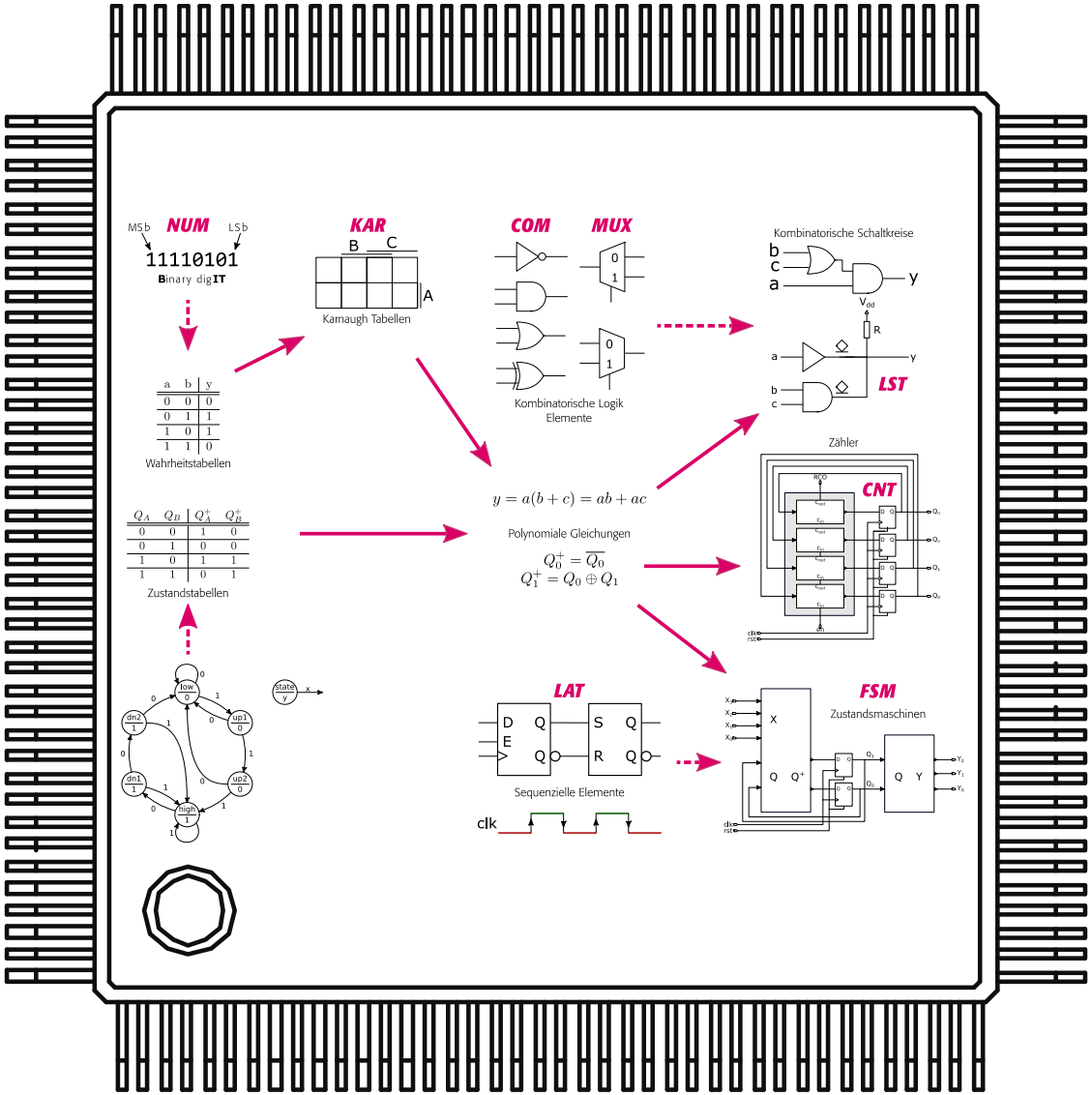
VHISC Hardware Description Language
Course Embedded Systems (SEm)
VHDL

Silvan Zahno / François Corthay



Degree program Systems Engineering
Specialization Infotronics – Embedded Systems

Current content of the topic in the course



VHISC Hardware Description Language (VHDL)

- **Usage**
- Structural description
- Behavioural description
- Usual types and operators
- Generic parameters
- Signals and variables
- Simulation and test

VHDL

Usage

Basic usage

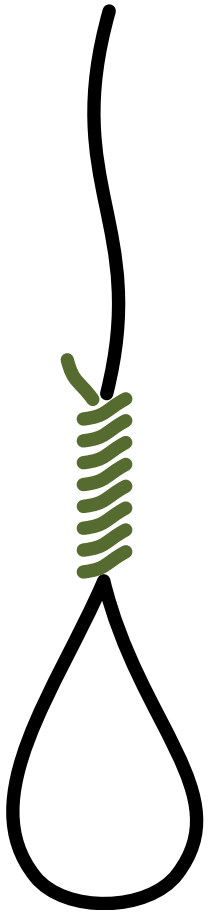
- Documentation
- Simulation
- Design
- Specification

Extended usage

- Automated synthesis
- Delay verification
- Formal equivalence

VHDL
Caveat

Hardware description ! = Software program



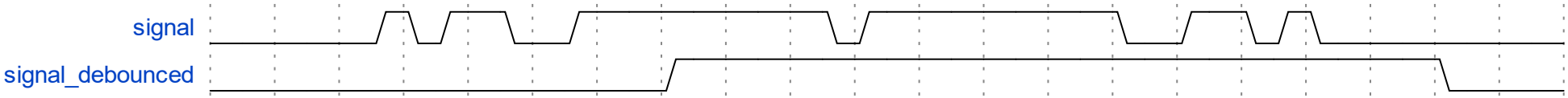
VHISC Hardware Description Language (VHDL)

- Usage
- **Structural description**
- Behavioural description
- Usual types and operators
- Generic parameters
- Signals and variables
- Simulation and test

Structural Description

Debouncing circuit

Circuit must remove bouncing for a given time period:



Will be used as an example throughout the presentation

Structural Design

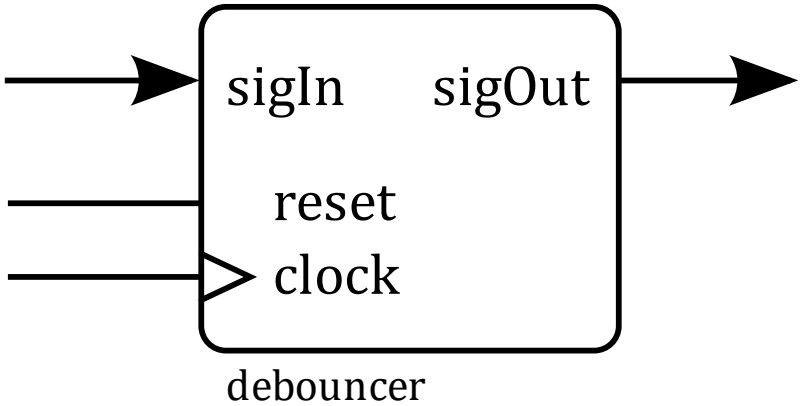
Entity

The entity describes the external view of the circuit

```
library ieee;
  use ieee.std_logic_1164.all;

entity debouncer is
  port(
    reset  : in  std_ulogic;
    clock  : in  std_ulogic;
    sigIn  : in  std_ulogic;
    sigOut : out std_ulogic
  );
end debouncer;

```

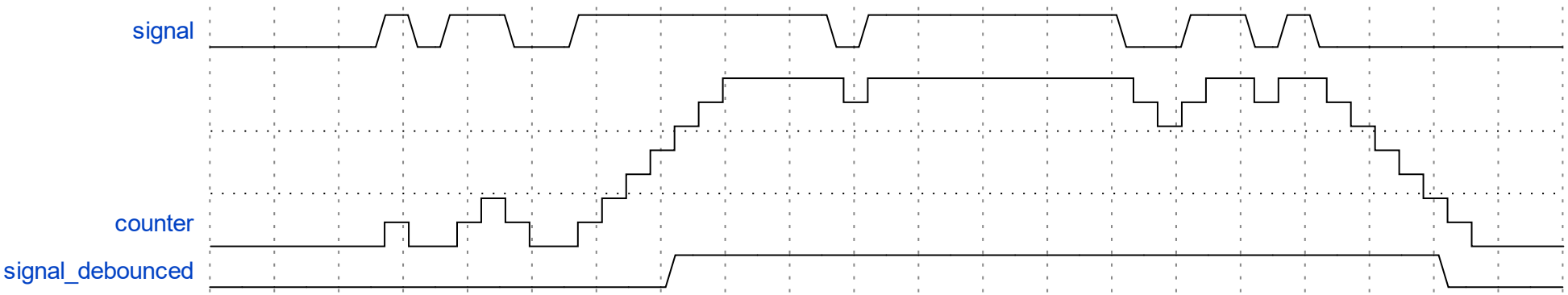


Structural Description

Debouncing design

Requires:

- A counter
- A Schmitt-trigger



Structural Description

Debouncing Architecture

A behavi our al architecture describes components and signals

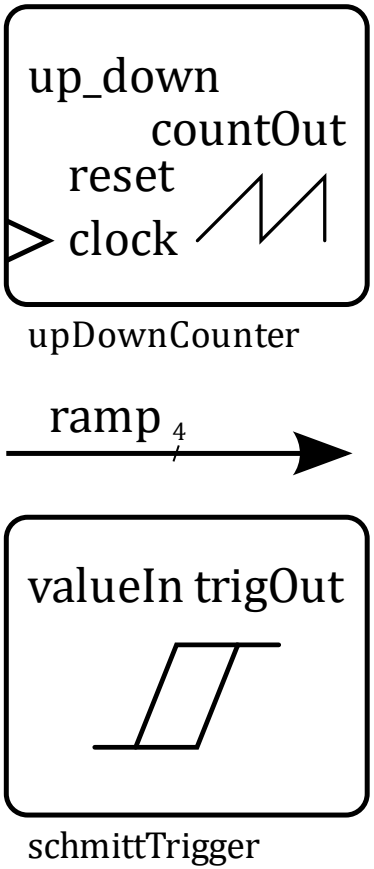
```
architecture struct of debouncer is

  component upDownCounter
    port(
      reset    : in  std_ulogic;
      clock    : in  std_ulogic;
      up_down  : in  std_ulogic;
      countOut : out unsigned(3 downto 0)
    );
  end component;

  signal ramp : unsigned(3 downto 0);

  component schmittTrigger
    port(
      valueIn : in  unsigned(3 downto 0);
      trigOut : out std_ulogic
    );
  end component;

begin
  -- components and connections
end struct;
```

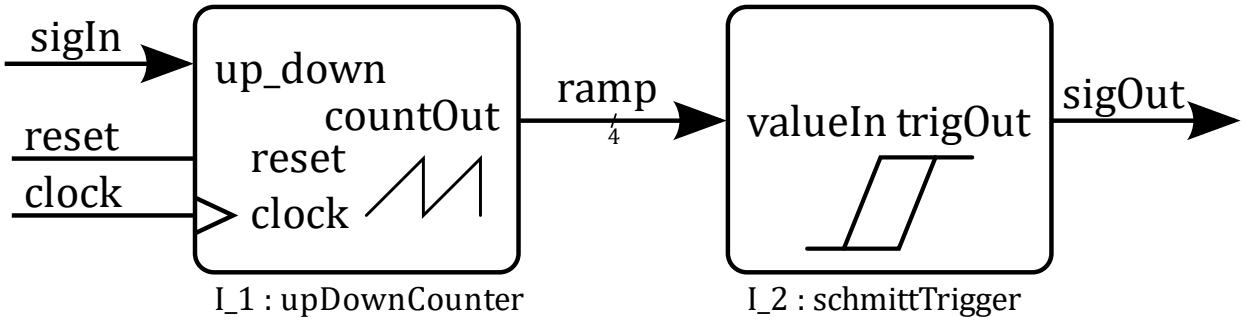


Structural Description

Debouncing Architecture

A behavi our al architecture describes components and signals

```
architecture struct of debouncer is
  -- signal and component declarations
begin
  I_1 : upDownCounter
    port map(
      reset    => reset;
      clock    => clock;
      up_down  => sigIn;
      countOut => ramp;
    );
  I_2 : schmittTrigger
    port map(
      valueIn => ramp;
      trigOut => sigOut;
    );
end struct;
```



VHISC Hardware Description Language (VHDL)

- Usage
- Structural description
- **Behavioural description**
- Usual types and operators
- Generic parameters
- Signals and variables
- Simulation and test

Behavioural Description

Assignments

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;                          -- flipflop I/O
  signal gtHigh, ltLow : std_ulogic;                  -- comparators
  signal DSet : std_ulogic;                 -- when over high threshold
begin
  -- comparisons

  DSet <= Q or gtHigh;                            -- gtHigh forces '1'
  D <= DSet and not ltLow;                         -- ltLow forces '0'

  -- D-flipflop
end RTL;

```

```
D <= DSet and not ltLow;                           -- ltLow forces '0'
DSet <= Q or gtHigh;                            -- gtHigh forces '1'
```

Whatever the order, the functionality is identical!

Behavioural Description

Processes

```
architecture RTL of schmittTrigger is
  -- signal declarations
begin
  -- comparisons and flipflop

  p1: process(Q, gtHigh)
  begin
    DSet <= Q or gtHigh;                          -- gtHigh forces '1'
  end process p1;

  p2: process(DSet, ltLow)
  begin
    D <= DSet and not ltLow;                       -- ltLow forces '0'
  end process p2;

end RTL;
```

- Direct assignments are a shorter way to declare what actually are processes
- Signals in the sensitivity list wake-up processes

Behavioural Description

Sensitivity list

```
architecture RTL of schmittTrigger is
  -- signal declarations
begin
  -- comparisons and flipflop

  p1: process(Q, gtHigh, ltLow, DSet)
  begin
    D <= DSet and not ltLow;                        -- ltLow forces '0'
    DSet <= Q or gtHigh;                           -- gtHigh forces '1'
  end process p1;

end RTL;
```

- Several assignments can be written inside a process
- The assignments are done in parallel : one can consider that they effectively occur on process exit
- For combinatorial circuits, all signals on the right side of assignments have to be in the sensitivity list

Behavioural Description

Delays

```
architecture RTL of schmittTrigger is
  -- signal declarations
begin
  -- comparisons and flipflop

  DSet <= Q or gtHigh after 2 ns;
  D <= DSet and not ltLow after 1 ns;

end RTL;
```

- Assigned signals change a specified time after the process has been waken-up
- The simulator keeps a list of what is going to happen in the future
- The list can be overwritten by another assignment

Behavioural Description

Delay types

```
DSet <= Q or gtHigh after 2 ns;

DSet <= Q or gtHigh inertial after 2 ns;

DSet <= Q or gtHigh transport 2 ns;

DSet <= Q or gtHigh reject 1 ns after 2 ns;
```

- Different behaviours can be specified for impulses shorter than the delay
- Default is : i ner t i al

Behavioural Description

Conditions if - else

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;                          -- flipflop I/O
  signal gtHigh, ltLow : std_ulogic;                  -- comparators
begin
  -- comparisons and flipflop

  setOrReset: process(gtHigh, ltLow, Q)
  begin
    if gtHigh = '1' then
      D <= '1';
    elsif ltLow = '1' then
      D <= '0';
    else
      D <= Q;
    end if;
  end process setOrReset;

end RTL;
```

- if only within a process, condition in sensitivity list
- Combinatorial => else is necessary

Behavioural Description

Conditions when

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;                          -- flipflop I/O
  signal gtHigh, ltLow : std_ulogic;                  -- comparators
begin
  -- comparisons and flipflop

  D <=   '1' when gtHigh = '1'
    else '0' when ltLow = '1'
    else  Q;

end RTL;
```

- when only outside a process
- Combinatorial => el se is necessary
- Compact but not very legible

Behavioural Description

Conditions case

```
1  architecture RTL of schmittTrigger is
2    signal D, Q : std_ulogic;                          -- flipflop I/O
3    signal gtHigh, ltLow : std_ulogic;                  -- comparators
4    signal conditions : std_ulogic_vector(1 to 2);
5  begin
6    -- comparisons and flipflop
7
8    conditions <= (gtHigh, ltLow);
9
10   setOrReset: process(conditions, Q)
11   begin
12     case conditions is
13       when "00"   => D <=  Q ;
14       when "10"   => D <= '1';
15       when "01"   => D <= '0';
16       when others => D <= '-';
17     end case;
18   end process setOrReset;
19
20 end RTL;
```

- case only inside a process, selector in sensitivity list
- Combinatorial => when others is necessary

Behavioural Description

Conditions with-select

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;                          -- flipflop I/O
  signal gtHigh, ltLow : std_ulogic;                  -- comparators
  signal conditions : std_ulogic_vector(1 to 2);
begin
  -- comparisons and flipflop

  conditions <= (gtHigh, ltLow);

  with conditions select
    D <=
       Q  when "00",
      '1' when "10",
      '0' when "01",
      '-' when others;

end RTL;
```

- with only outside a process
- Combinatorial => when others is necessary
- Compact but less easy to read

Behavioural Description

Synchronous circuit

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;                                  -- flipflop I/O
begin
  -- comparisons
  -- calculation of D

  store: process(reset, clock)
  begin
    if reset = '1' then
      Q <= '0';
    elsif rising_edge(clock) then
      Q <= D;
    end if;
  end process store;

end RTL;
```

- All signals assigned within `rising_edge(clock)` are to be initialized at reset (and no other ones)

Behavioural Description

Sensitivity list

- Combinatorial circuits : all inputs
- Sequential circuits : only reset and clock

VHISC Hardware Description Language (VHDL)

- Usage
- Structural description
- Behavioural description
- **Usual types and operators**
- Generic parameters
- Signals and variables
- Simulation and test

Usual types and operators

Numbers

```
type    integer  is range -2147483648 to 2147483647;
subtype natural  is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;

-- operators:    + - * /    **    abs    mod rem    sla sra

-- comparisons:    =    /=    <    >    <=    >=

```

- Independent of their possible binary coding
- Not compatible with bit (or std_logic) vectors

Usual types and operators

Logic signals

```
type std_ulogic is
  (
    'U',  -- uninitialized
    'X',  -- forcing unknown
    '0',  -- forcing 0
    '1',  -- forcing 1
    'Z',  -- high impedance
    'W',  -- weak unknown
    'L',  -- weak 0
    'H',  -- weak 1
    '-',  -- don't care
  );

-- operators:    not    and    nand    or    nor    xor

-- functions:    rising_edge    falling_edge

```

- Defined in ieee package std_logic_1164
- Signals always start with leftmost value : here 'U'

Usual types and operators

Logic signals

```
type std_ulogic is
  (
    'U',  -- uninitialized
    'X',  -- forcing unknown
    '0',  -- forcing 0
    '1',  -- forcing 1
    'Z',  -- high impedance
    'W',  -- weak unknown
    'L',  -- weak 0
    'H',  -- weak 1
    '-',  -- don't care
  );

-- operators:    not    and    nand    or    nor    xor

-- functions:    rising_edge    falling_edge

```

- Defined in ieee package std_logic_1164
- Signals always start with leftmost value : here 'U'

Usual types and operators

Logic arrays

```
type std_ulogic_vector is array ( natural range <> ) of std_ulogic;

-- concatenation operator for arrays:    &
```

```
type unsigned is array ( natural range <> ) of std_logic;
type signed   is array ( natural range <> ) of std_logic;
```

```
-- array assignments:
myString <= "food";
myString <= ('f', 'o', 'o', 'd');
myString <= (1 => 'f', 4 => 'd', 2|3 => 'o');
myString <= (4 => 'd', 1 => 'f', others => 'o');
myString <= (others => 'X');
```

- Types defined in ieee packages std_logic_1164 and numeric_std
- Operators to be examined in the corresponding package declarations

Usual types and operators

Type casting

```
architecture RTL of upDownCounter is
  signal counter : unsigned(3 downto 0);
  constant maxValue : signed(3 downto 0) := to_signed(-1,
countOut'length);
begin
  countAndSaturate: process(reset, clock)
  begin
    if reset = '1' then
      counter <= (others => '0');
    elsif rising_edge(clock) then
      if (up_down = '1') and (counter <= unsigned(maxValue)) then
        counter <= counter + 1;
      elsif (up_down = '0') and (counter >= 0) then
        counter <= counter - 1;
      end if;
    end if;
  end process countAndSaturate;
  countOut <= counter;
end RTL;
```

- VHDL is very strict
- signed maxValue has to be cast to unsigned
- Types must be compatible (here: array of std_logic)

Usual types and operators

Type qualification

```
architecture RTL of upDownCounter is
  subtype counterType is unsigned(3 downto 0);
  signal counter : counterType;
begin
  countAndSaturate: process(reset, clock)
  begin
    if reset = '1' then
      counter <= (others => '0');
    elsif rising_edge(clock) then
      if (up_down = '1') and (counter+1 /= 0) then
        counter <= counter + 1;
      elsif (up_down = '0') and (counter >= counterType'(others = '0')) then
        counter <= counter - 1;
      end if;
    end if;
  end process countAndSaturate;
  countOut <= counter;
end RTL;
```

- Indeed, VHDL is very strict
- `(others => '0')` is not qualified : it may be of various lengths and of type string, unsigned, signed, …

Usual types and operators

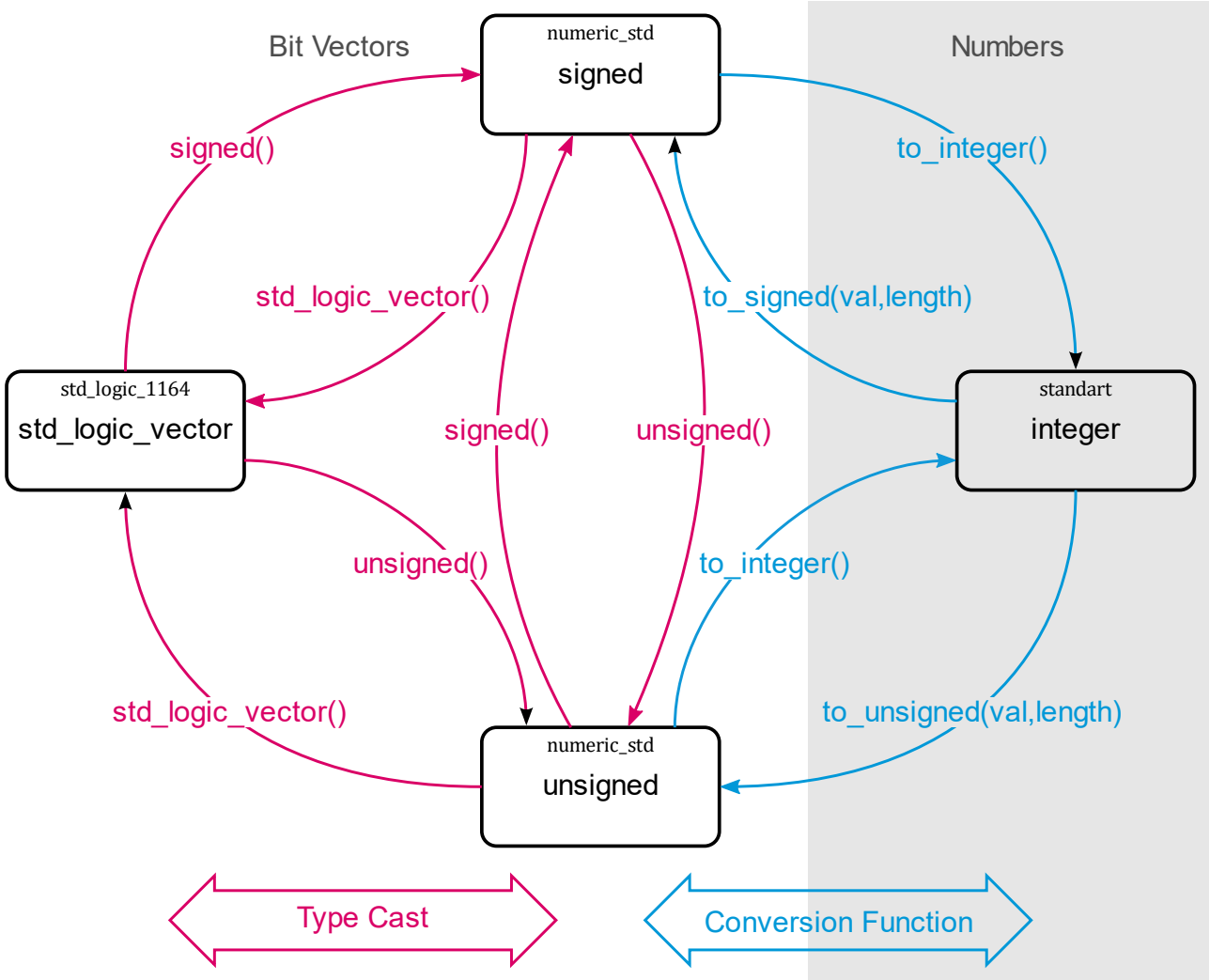
Type conversion

```
architecture RTL of upDownCounter is
  signal counter : unsigned(3 downto 0);
begin
  countAndSaturate: process(reset, clock)
  begin
    if reset = '1' then
      counter <= to_unsigned(0, 4);
    elsif rising_edge(clock) then
      if (up_down = '1') and (counter+1 /= 0) then
        counter <= counter + 1;
      elsif (up_down = '0') and (counter >= 0) then
        counter <= counter - 1;
      end if;
    end if;
  end process countAndSaturate;
  countOut <= counter;
end RTL;
```

- counter is unsigned, 0 is integer (or natural) : the types are not compatible
- Conversion function found in package numeric_std

Usual types and operators

Usual castings and conversions



Usual types and operators

Physical types

```
type time is range -2147483647 to 2147483647
  units
    fs;
    ps = 1000 fs;
    ns = 1000 ps;
    us = 1000 ns;
    ms = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr = 60 min;
  end units;
```

- Mostly used for the type time
- Could be used for analog and mechanical simulations and others...

VHISC Hardware Description Language (VHDL)

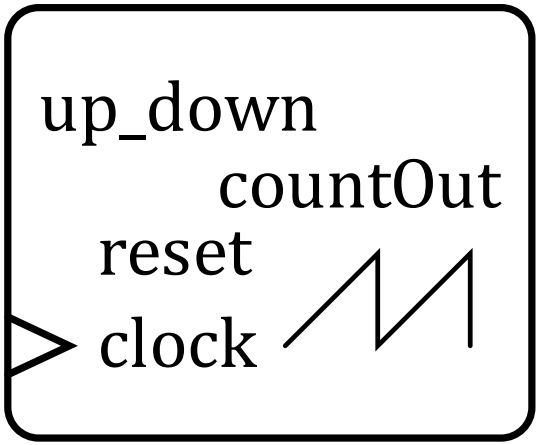
- Usage
- Structural description
- Behavioural description
- Usual types and operators
- **Generic parameters**
- Signals and variables
- Simulation and test

Generic parameters

Generic declaration

```
library ieee;
  use ieee.std_logic_1164.all;

entity upDownCounter is
  generic(
    bitNb : integer := 8
  )
  port(
    reset    : in  std_ulogic;
    clock    : in  std_ulogic;
    up_down  : in  std_ulogic;
    countOut : out unsigned(bitNb-1 downto 0)
  );
end upDownCounter;
```



upDownCounter
bitNb=8

- The generic value is known in the port mapping and inside the circuit

Generic Parameters

Attributes

```
architecture RTL of upDownCounter is
  signal counter : unsigned(countOut'range);
begin
  countAndSaturate: process(reset, clock)
  begin
    if reset = '1' then
      counter <= (others => '0');
    elsif rising_edge(clock) then
      if up_down = '1' then
        if (counter(counter'high downto counter'high-1) /= "11") then
          counter <= counter + 1;
        end if;
      elsif up_down = '0' then
        if (counter(counter'high downto counter'high-1) /= "00") then
          counter <= counter - 1;
        end if;
      end if;
    end if;
  end process countAndSaturate;
  countOut <= counter;
end RTL;
```

- Signal and type attributes allow to write robust code
- Often independent of generic parameters

Generic Parameters

Attributes

```
architecture RTL of upDownCounter is
  signal counter : unsigned(7 downto 0);
  signal debugString : string(1 to 10);
begin
  -- counter'length = 8
  -- counter'high   = 7
  -- counter'low    = 0
  -- counter'left   = 7
  -- counter'right  = 0
  -- counter'range  = (7 downto 0)
  -- counter'reverse_range = (0 to 7)

  -- counter MSB = counter(7) = counter(counter'high)

  -- debugString'left   = 1
  -- debugString'right  = 10
end RTL;
```

- Never use constants within the code : prefer the attributes
- The code must function with different generic values

Generic Parameters

Iterative structures

```
architecture structural of shiftRegister is

  component dFlipFlop
    port(
      clock, reset, d : in std_ulogic;
      Q               : out std_ulogic
    );
    end component;

  signal c: std_ulogic_vector(1 to bitNb+1);  -- 1 bit more

begin

  c(c'low) <= dataIn;

  shReg: for i in c'low to c'high-1 generate
    dFF: dFlipFlop port map(clock, reset, c(i), c(i+1));
  end generate shReg;

  dataOut <= c(c'high);

end structural;
```

- Describe circuits with sizes based on generic values

VHISC Hardware Description Language (VHDL)

- Usage
- Structural description
- Behavioural description
- Usual types and operators
- Generic parameters
- **Signals and variables**
- Simulation and test

Signals and variables

```
architecture RTL of schmittTrigger is
  signal DSet : std_ulogic
  -- other signal declarations
begin
  -- comparisons and flipflop

  withSignal: process(Q, gtHigh, ltLow, DSet)
  begin
    D <= DSet and not ltLow;                                -- ltLow forces '0'
    DSet <= Q or gtHigh;                                    -- gtHigh forces '1'
  end process withSignal;

  withVariable: process(Q, gtHigh, ltLow)
    variable DSet_var : std_ulogic;
  begin
    DSet_var := Q or gtHigh;                                -- gtHigh forces '1'
    D <= DSet_var and not ltLow;                            -- ltLow forces '0'
  end process withVariable;

end RTL;
```

- Signals declared at architecture level
- Variables declared inside a process (and not seen outside it)

Signals and variables

Variables to signals

```
architecture RTL of upDownCounter is
begin
  countAndSaturate: process(reset, clock)
    variable counter : unsigned(countOut'range);
  begin
    if reset = '1' then
      counter := (others = '0');
    elsif rising_edge(clock) then
      if up_down = '1' then
        counter := counter + 1;
        if counter = 0 then                                    -- if overflow
          counter := (others = '1');                        -- back to max value
        end if;
      elsif (up_down = '0') and (counter >= 0) then
        counter := counter - 1;
      end if;
    end if;
    countOut <= counter;                  -- inside the process, best at the end
  end process countAndSaturate;
end RTL;
```

- Like signals, variables keep their values between process activations

Signals and variables

Iterative functions

```
architecture RTL of upDownCounter is
  signal counter : unsigned(countOut'range);
  signal allZero : std_ulogic
begin

  checkIfAllZero: process(counter)
    variable allZeroInt : std_ulogic;
  begin
    allZeroInt := '1';
    for i in counter'range loop
      if counter(i) = '1' then
        allZeroInt := '0';
      end if;
    end loop;                                        -- loop with zero delay !
    allZero <= allZeroInt;
  end process checkIfAllZero;

end RTL;
```

- Here is a very good reason to use variables
- Loops are done over array elements, not in time: only the clock lets us advance in time

Signals and variables

Usage of variables

- Variables allow programming-like code
- Remain unchanged between process activiations
- Are fine for describing iterative structures
- Only known inside processes (not totally true)
- Cannot have a delay attached

VHISC Hardware Description Language (VHDL)

- Usage
- Structural description
- Behavioural description
- Usual types and operators
- Generic parameters
- Signals and variables
- **Simulation and test**

Simulation and test

Simple testbench example

```
architecture test of upDownCounter_tester is
                                                    -- clock definitions
  constant clockFrequency : real       := 100.0E6;
  constant clockPeriod    : time       := (1.0/clockFrequency) * 1 sec;
  signal   sClock         : std_ulogic := '1';

begin
  ------------------------------------------------------------------
                                                      -- clock and reset
  sClock <= not sClock after clockPeriod/2;
  clock <= transport sClock after clockPeriod*9/10;
  reset <= '1', '0' after 2*clockPeriod;

  ------------------------------------------------------------------
                                                 -- input signal waveform
  up_down <=
    '0',
    '1' after 10*clockPeriod,
    '0' after 50*clockPeriod;

end test;
```

- Weird types and assignments, not usable in synthesizable code

Simulation and test

Inital values

```
architecture RTL of upDownCounter is
  signal counter : unsigned(countOut'range) := (others => '0');
  signal isHigh : std_ulogic := '0';
begin

  countAndSaturate: process(reset, clock)
  begin
    if reset = '1' then
      counter <= (others => '0');
    elsif rising_edge(clock) then
      if (up_down = '1') and (counter+1 /= 0) then
        counter <= counter + 1;
      elsif (up_down = '0') and (counter >= 0) then
        counter <= counter - 1;
      end if;
    end if;
  end process countAndSaturate;

  isHigh <= '1' when counter(counter'high downto counter'high-1) = "11"
    else '0';

end RTL;
```

- Only flipflops are initialized, with the r eset signal
- Combinatorial logic can't be initialized

Simulation and test

Memory effect (latch)

```
architecture RTL of schmittTrigger is
  signal D, Q : std_ulogic;
  signal gtHigh, ltLow : std_ulogic;
begin
  -- comparisons and flipflop

  setOrReset: process(gtHigh, ltLow, Q)
  begin
    if gtHigh = '1' then
      D <= '1';
    elsif ltLow = '1' then
      D <= '0';
    end if;
  end process setOrReset;

end RTL;
```

- No else condition => D might not be assigned => has to keep the same value
- The synthesizer will try to implement a latch
- The circuit will not always work (maybe sometimes)

Simulation and test

Test vs synthesizable code

- Testbench is not going into a (programmable) circuit
- Some signal types and functions are not synthesizable
- Delays are not synthesizable
- Initializations are only synthesizable with the help of a Power-On Reset (POR)

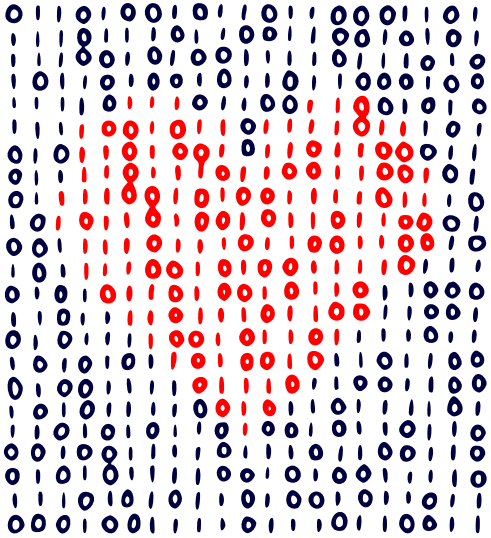
Simulation and test

Real world

- Simulation ensures proper functionality as long as:
  - the stimuli correspond to the real world signals
  - the circuit is synchronous
  - the inputs are properly synchronized
  - the synthesis has not reported severe problems

VHDL

- We now know everything about:
  - Structural description
  - Behavioural description
  - Usual types and operators
  - Generic parameters
  - Signals and variables
  - Simulation and test



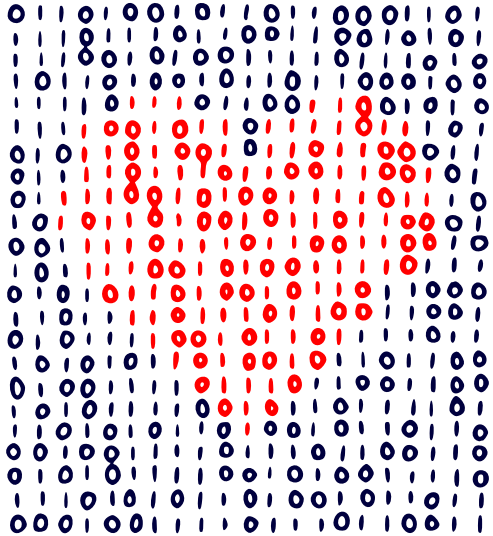
References

- [War17] (Englisch) FPGA Designer Warrior
  http://blog.aku.edu.tr/ismailkoyuncu/files/2017/04/01_ebook.pdf

- [Int19] (Englisch) Intel FPGA Website
  https://www.intel.com/content/www/us/en/products/programmable.html

- [Xil19] (Englisch) Xilinx FPGA Website
  https://www.xilinx.com/

- [Act19] (Englisch) Actel FPGA Website
  https://www.microsemi.com/product-directory/1636-fpga-soc




:
Σ
π
≈
&