Hardware Programming HWP1

capturing a
FPGA
design with
VHDL

Planning: theory

- First week
  - Introduction digital systems
  - Introduction to FPGAs
  - Structured digital Design
  - Modelling concepts in VHDL
- Second week
  - Introduction VHDL
  - Design verification
  - Code structure and data types
- Third week
  - Combinational versus sequential design
  - Concurrent and sequential code
  - Signals and variables
- Fourth week
  - Components
  - Generics
- Fifth week
  - Designing state machines

Agenda

- Discussion of previous week
- Introduction to VHDL
- Design verification
- Code structure and data types

Agenda

- Discussion of previous week
- Introduction to VHDL
- Design verification
- Code structure and data types

Introduction to VHDL

- VHSIC was a 1980s U.S. government program to develop very-high-speed integrated circuits. The United States Department of Defense launched the VHSIC project in 1980 as a joint tri-service project. The project led to advances in integrated circuit materials, lithography, packaging, testing, and algorithms, and created numerous computer-aided design tools. A well-known part of the project's contribution is VHDL, a hardware description language.

- Standard defined by IEEE in 1987, revisions in 2008 and 2019

- Remember: VHDL is not a normal programming language. It describes digital hardware.

- Everything happens all the time

Entity and architecture keywords

- **ENTITY**
  - Define a function block and specify the interface to the outside world with **PORTS**
- **ARCHITECTURE**
  - Define the **implementation** of your entity. Either **behavioral** or **structural**

VHDL Design

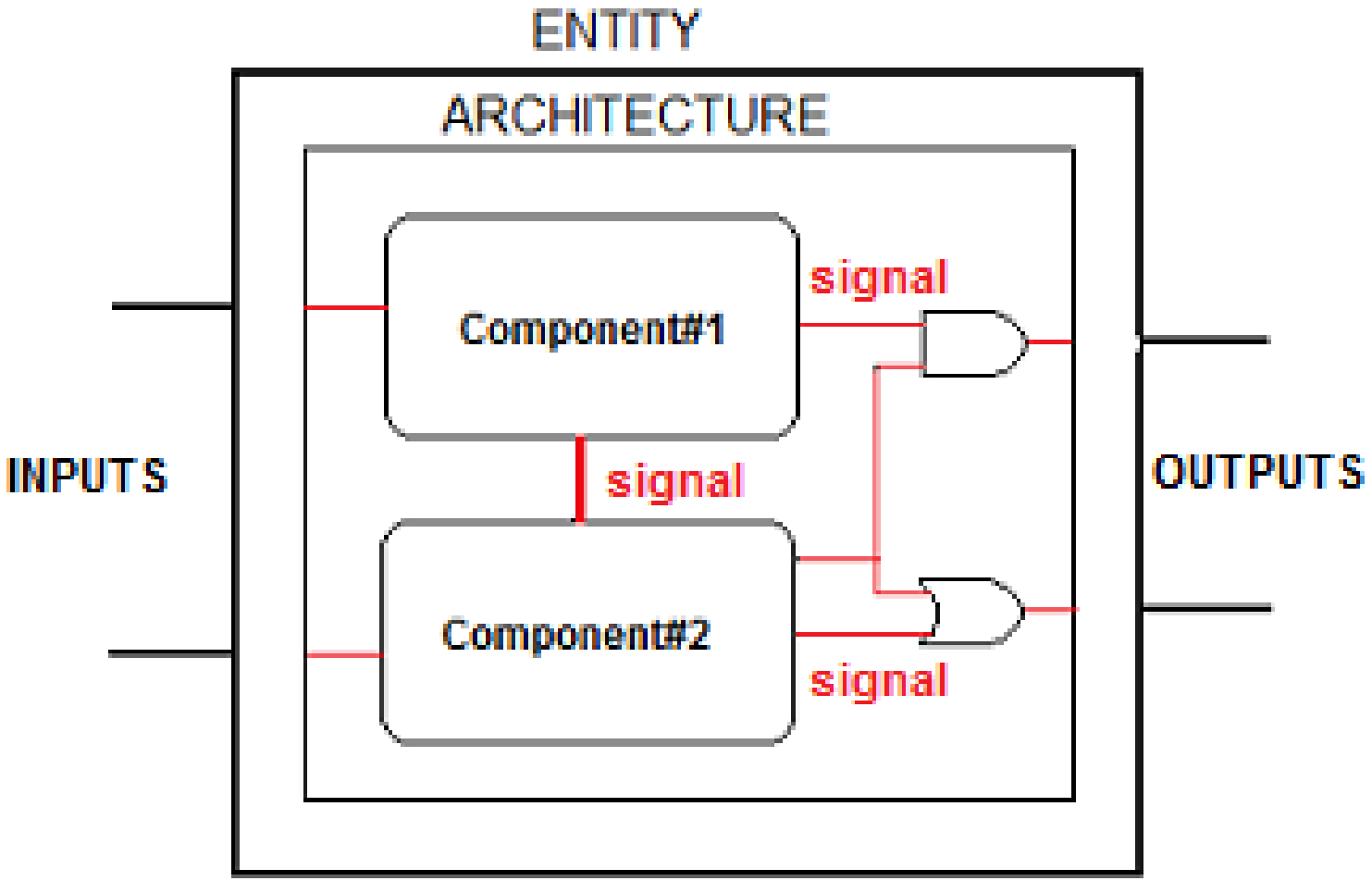


Figure adapted from FPGAcenter.com

Agenda

- Discussion of previous week
- Introduction to VHDL
- **Design verification**
- Code structure and data types

Design verification: test bench

- Functional verification of your design

- In this course you will have to create a test bench for every assignment you hand in

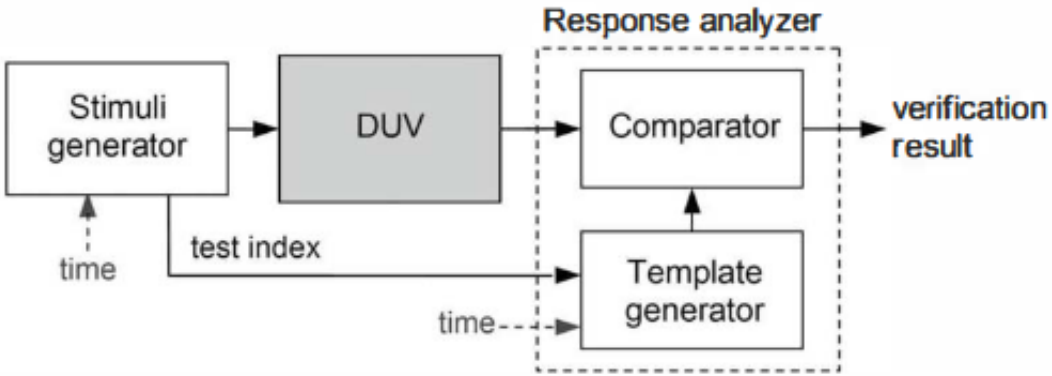


Figure adapted from Pedroni.

Test bench overview

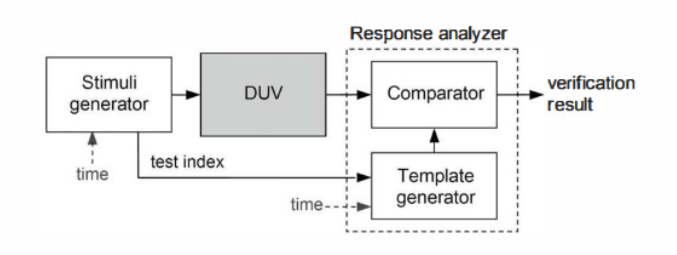
- A test bench entity has no ports
- In the architecture you create port maps to instantiate your design
- Generate a clock signal as stimulus
- In this course we only cover functional verification; no timing verification

Delay models

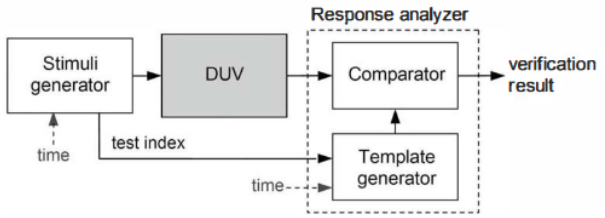
- Inertial delay:
  - Models delays in gates with the after clause: **a <= b AFTER 1 ns;**
- Transport delay:
  - Models delays in wires
- Delta delay:
  - Delay added automatically by the simulator if no delay is explicitly prescribed
- Remember: delay statements are ***not*** synthesizable

Discussion of testbench example

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

entity assignment2_tb is
end entity;

architecture testbench of assignment2_tb is
            component seven_segment_decoder is
            port (
                        hex_nibble: in std_ulogic_vector(3 downto 0);
                        blank: in std_ulogic;
                        seven_segment_code: out std_ulogic_vector(6 downto 0)
            );
            end component;
            signal hex_nibble_tb                    : std_ulogic_vector(17 downto 0);
            signal blank_tb                         : std_ulogic_vector(6 downto 0);
            signal seven_segment_code_tb            : std_ulogic_vector(17 downto 0);

begin
duv: seven_segment_decoder port map(
            seven_segment_code => seven_segment_code_tb,
            blank => blank_tb,
            seven_segment_code => seven_segment_code_tb
);
process
begin
            sw_tb <= "000000000000000000";
            for i in 0 to 15 loop
                        wait for 10 ns;
                        sw_tb <= std_ulogic_vector(unsigned(sw_tb) + 1);
            end loop;

            report "test completed.";
            wait;
end process;

end architecture;
```



```
process
        type int_array is array(0 to 15) of integer;
        constant expexted_seven_segment_code: int_array := (
                16#40#, 16#79#, 16#24#, 16#30#,
                16#19#, 16#12#, 16#02#, 16#78#,
                16#00#, 16#10#, 16#08#, 16#03#,
                16#46#, 16#21#, 16#06#, 16#0E#
        );
        begin
                report "Testing entity assignment2.";
                -- Initialize signals.
                hex_nibble_tb <= "0000";

                blank_tb <= '1';
                wait for 10 ns;
                -- Check blank.
                assert seven_segment_code_tb = "1111111"
                        report "test failed for blank = 1" severity error;

                blank_tb <= '0';
                -- Loop through all possible values of switches.
                for i in 0 to 15 loop
                        hex_nibble_tb <= std_ulogic_vector(to_unsigned(i, hex_nibble_tb   'length));
                        wait for 10 ns;

                        -- Check result.
                        assert seven_segment_code_tb = std_ulogic_vector(to_unsigned(
                                expexted_seven_segment_code(i), seven_segment_code_tb'length
                        ))
                        report "test failed for i = " & to_string(i)
                        severity error;
                end loop;

                report "Test completed.";
                std.env.stop;
        end process;
end architecture;
```



Agenda

- Discussion of previous week
- Introduction to VHDL
- Design verification
- **Code structure and data types**

Logic values

- See book chapter 7.6.2
- There are 4 synthesizable logic values
  - ‘0’, ‘1’, ‘-’ and ‘Z’.
- There are 5 other useful logic values
  - ‘L’ = (weak ‘0’), ‘H’ = (weak ‘1’), ‘X’, ‘W = (weak ?)’, ‘U’.
- You should not use any other then the 4 synthesizable logic values during this course.

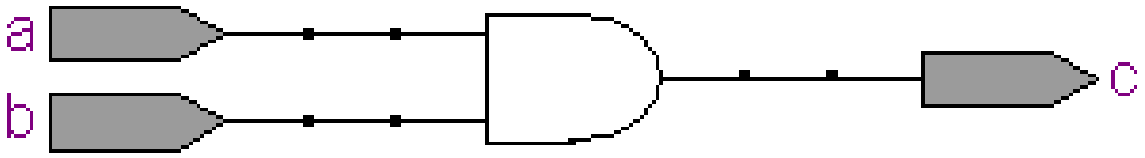
Resolved and Unresolved

- Multiple drivers are driving a logic signal
  - Is this correct?
  - What happens?

- Unresolved
- Resolved

- Detailed explanation

Example: AND gate




```
-----------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------
ENTITY andgate IS
   PORT (a,b: IN  STD_ULOGIC;
         c: OUT STD_ULOGIC);
END andgate;
-----------------------------------------------
ARCHITECTURE voorbeeld OF andgate IS
BEGIN
        c <= A AND B;
END voorbeeld;
-----------------------------------------------
```

Libraries

```
•  ------------------------------------------
•  LIBRARY library_name;
•  USE library_name.package_name.all;
•  ------------------------------------------
The most frequently used packages are:
•  Package standard, from the library std (visible by default)
•  Library work (where the projects are saved is visible by default)
•  Package std_logic_1164, from the library ieee (when needed, must be
explicitly declared)

•  ------------------------------------------
•  LIBRARY std;           -- optional declaration
•  USE std.standard.all; --optional
•  LIBRARY work;           -- optional
•  USE work.all;  -- optional; also not needed if defined as below
•  USE work.my_package.all;  --if an extra user made package is needed
•  LIBRARY ieee;
•  USE ieee.std_logic_1164.all;
•  ------------------------------------------
```

Data Types: IEEE 1164 Standard Logic

- IEEE's **std_logic_1164** examples:

- STD_ULOGIC

```
SIGNAL a: STD_ULOGIC;

a <= '0';       -- a is zero
a <= '1';       -- a is one
a <= 'Z';       -- a is high-impedance
                -- used for bidi ports
```

- STD_ULOGIC_VECTOR

```
SIGNAL b: STD_ULOGIC_VECTOR (7 DOWNTO 0);

b <= "1100ZZZZ";   -- array of 8 std_logic's
                   -- in MSB representation
```

Data Types: *IEEE.numeric_std.all* LIBRARY

```
-------------------------------------------
LIBRARY ieee;
USE ieee.numeric_std.all;
-------------------------------------------
```

Allows for: unsigned, signed, integer, natural

Contains many **functions** such as: abs(), + addition, - subtraction, * multiplication, / division, etc…, that can be used with these types

As well as: >, <, <=, >=, =, sll, srl

Normal STD_LOGIC_VECTORS: can't calculate with those!

Additional **Type conversion functions** needed to overcome this…..

Zie H10.7 van het boek.

https://www.csee.umbc.edu/portal/help/VHDL/numeric_std.vhdl

Conversion of Common Types

```
SIGNAL a: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL b: UNSIGNED (7 DOWNTO 0);
SIGNAL c: SIGNED (7 DOWNTO 0);
```

(un)signed → std_logic_vector

```
a <= STD_LOGIC_VECTOR(b);
a <= STD_LOGIC_VECTOR(c);
```

std_logic_vector → (un)signed

```
b <= UNSIGNED(a);
c <= SIGNED(a);
```

Conversion of Common Types

```
SIGNAL d: UNSIGNED (7 DOWNTO 0);
SIGNAL e: SIGNED (7 DOWNTO 0);
SIGNAL f: INTEGER RANGE 0 TO 255;
SIGNAL g: INTEGER RANGE -128 TO 127;
```

integer → (un)signed

```
d <= TO_UNSIGNED(f, 8);     -- d is an array of 8
e <= TO_SIGNED(g, 8);       -- e is an array of 8
```

(un)signed → integer

```
f <= TO_INTEGER(d);
g <= TO_INTEGER(e);
```

Conversion of Common Types

```
SIGNAL h: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL i: INTEGER RANGE 0 TO 255;
SIGNAL j: INTEGER RANGE -128 TO 127;
```

integer → std_logic_vector

```
h <= STD_LOGIC_VECTOR(TO_UNSIGNED(i,i'length);
h <= STD_LOGIC_VECTOR(TO_SIGNED(j,j'length);
```

std_logic_vector → integer

```
i <= TO_INTEGER(UNSIGNED(h));
j <= TO_INTEGER(SIGNED(h));
```

Warning

STD_LOGIC_ARITH

is

OBSOLETE

Don’t use it, no matter what older books or websites/forums/code snippets online say!

Use IEEE 1164

ieee.numeric_std.all

instead for unsigned and signed types!

Assignment Operators

<= for a ***signal***

:= for a ***variable*** (covered later)

=> for individual elements of a vector

Example:

```
signal   a: std_logic;
variable b: integer range 0 to 255;
signal   c: std_logic_vector(3 downto 0);

a <= '1';                                    -- assign single bits with '
b := 10;                                     -- assign an integer
c <= "1100";                                 -- assign a vector with "
c <= (3 => '1', 2 => '1', OTHERS => '0');    -- same as previous line
a <= c(0);
```

Other Functions available with numeric_std

- Logical

  not, and, or, nand, nor, xor, xnor

- Arithmetic

  +, -, *, /, **, mod, rem, abs

  NOTE: Not all synthesizable

- Comparison

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

- Shift operators:

  sll, srl -- shift left logical, shift right logical

  sla, sra -- shift right arithmetic, shift right arithmetic

  rol, ror -- rotate left, rotate right

Attributes

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);

d'LOW     = 0                          -- laagste array index
d'HIGH    = 7                          -- hoogste array index
d'LEFT    = 7                          -- meest linkse array index
d'RIGHT   = 0                          -- meest rechtse array index
d'LENGTH = 8                           -- lengte van de vector
d'RANGE   = (7 DOWNTO 0)               -- range of the vector
d'REVERSE_RANGE = (0 TO 7) -- reverse range of the vector
```

Zie H9 van het boek

Simple Signal Assignment statement

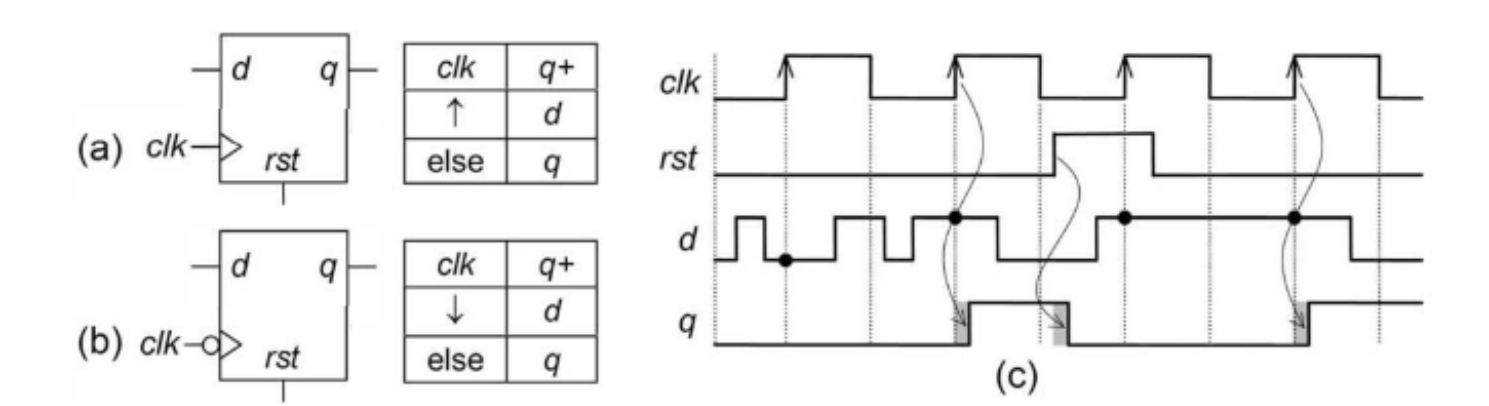
- When the Right Hand Side (RHS) of a signal assignment changes, the signal assignment statement is executed

- Signals in a circuit are modeled as signal statement assignments in VHDL

- Order in text is *not* preserved!

D flip-flop

- We'll show you a behavioral and structural description of a simple D flip-flop



Figures adapted from Embedded Systems Design: A Unified Hardware/Software Introduction

Example: DFF

- Use rising_edge() function (or falling_edge())

```
-------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------
ENTITY dff IS
          PORT(d, clk, rst: IN STD_LOGIC;
         q, qi: OUT STD_LOGIC)
END dff;
-------------------------------------
ARCHITECTURE dff_arch OF dff IS
BEGIN
          PROCESS (rst, clk)
          BEGIN
                    IF (rst='1') THEN
                              q <= '0';
                    ELSIF rising_edge(clk)
                    THEN
                              q <= d;
                    END IF;
          END PROCESS;

          qi <= NOT(q);

END dff_arch;
```

Summary

- Introductie in VHDL
- Verify the functional behavior of your design with test benches
- Use standard functions for conversion of data types

Homework

- Covered today:
  - Discussion of previous week
  - Introduction to VHDL
  - Code structure and data types
  - Design verification

- Next week:
  - Chapter 1 and 2 “Review of Combinational and Sequential Circuits”
  - Chapter 10 (11) “Concurrent Code”
  - Chapter 12 (13) “Sequential Code”