HOGESCHOOL ROTTERDAM

Hardware Programming HWP01 2020-2021

capturing a

FPGA

design with

VHDL

Planning: theory

- First week
  - Introduction digital systems
  - Introduction to FPGAs
  - Structured digital Design
  - Modeling concepts in VHDL
- Second Week
  - Introduction VHDL
  - Code structure
  - Data types
- Third week
  - Combinational versus sequential design
  - Concurrent and sequential code
  - Signals and variables
- Fourth week
  - Introduction to state machines
- Fifth week
  - Designing state machines
  - Advanced VHDL design

Agenda

- Introduction to VHDL
- Code structure and data types
- Design verification

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

- Remember: VHDL is not a normal programming language

- Everything happens all the time

VHDL Design

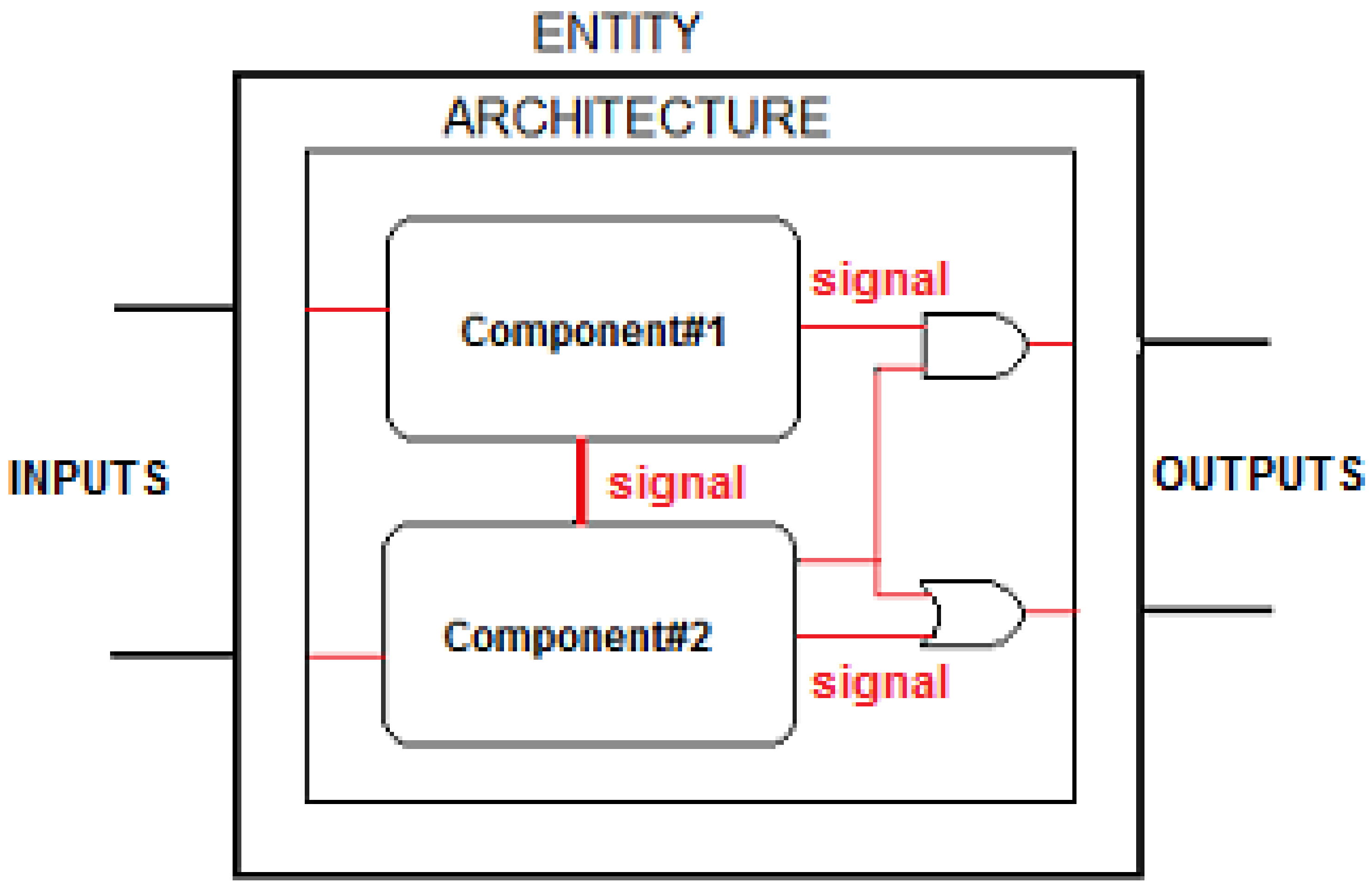


Figure adapted from FPGAcenter.com

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

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

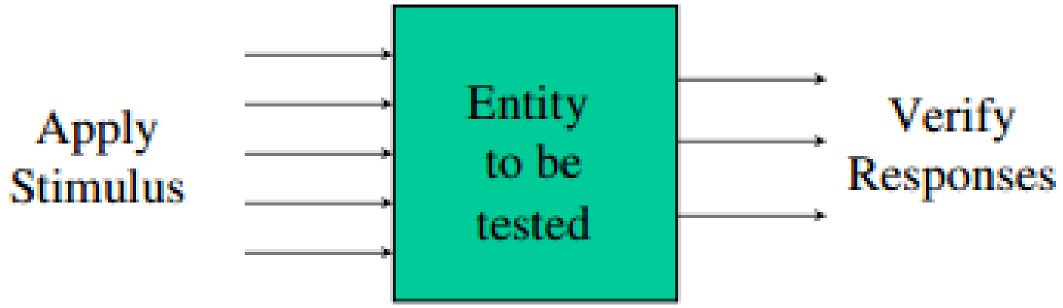


Figure adapted from Duckworth.

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 ***<u>not</u>*** synthesizable

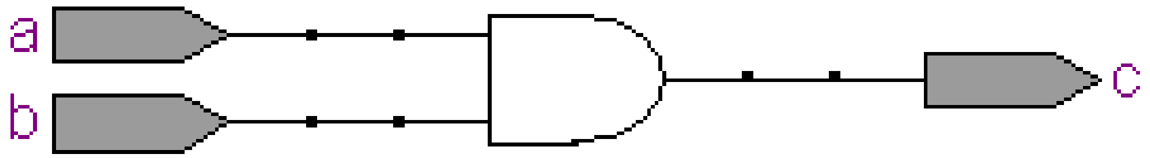
Discussion of example

```vhdl
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 assignment2 IS
                        PORT (
            SW          : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
            HEX0        : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
            LEDR        : IN STD_LOGIC_VECTOR(17 DOWNTO 0)
);
            END COMPONENT;
            SIGNAL SW_tb            : STD_LOGIC_VECTOR(17 DOWNTO 0);
            SIGNAL HEX0_tb          : STD_LOGIC_VECTOR(6 DOWNTO 0);
            SIGNAL LEDR_tb          : STD_LOGIC_VECTOR(17 DOWNTO 0);

            BEGIN
TB: assignment2 PORT MAP (SW => SW_tb, HEX0 => HEX0_tb, LEDR => LEDR_tb);
PROCESS
BEGIN
            SW_tb <= "000000000000000000";
            FOR I IN 0 TO 15 LOOP
                        WAIT FOR 10 ns;
                        SW_tb <= STD_LOGIC_VECTOR(UNSIGNED(SW_tb) + 1);
            END LOOP;

            REPORT "Test completed.";
            WAIT;
END PROCESS;

END ARCHITECTURE;
```

Agenda

- Introduction to VHDL
- **Code structure and data types**
- Design verification

Example: AND gate



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

```
•   --------------------------------------------
•   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
•   --------------------------------------------
```

INCLUDE : *IEEE.STD_LOGIC_1164* LIBRARY

https://www.csee.umbc.edu/portal/help/VHDL/std_logic_1164.vhdl

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
```

Defines STD_LOGIC as well as STD_LOGIC_VECTOR (array of STD_LOGIC)

```
'U',  -- Uninitialized
'X',  -- Unknown
'0',  -- LOGIC  0
'1',  -- LOGIC  1
'Z',  -- High Impedance  (used for tri-state I/O's)
'W',-- Weak  signal
'L',  -- Weak signal with logic  0   as preference
'H',  -- Weak signal with logic  1   as preference
'-'   -- Don't care
```

Defines various FUNCTIONS, such as EDGE DETECTION

```
    ------------------------------------------------------------------
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
```

INCLUDE : *IEEE.STD_LOGIC_1164* LIBRARY

https://www.csee.umbc.edu/portal/help/VHDL/std_logic_1164.vhdl

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
```

Defines STD_LOGIC as well as STD_LOGIC_VECTOR (array of STD_LOGIC)

```
'U',  -- Uninitialized
'X',  -- Unknown
'0',  -- LOGIC  0
'1',  -- LOGIC  1
'Z',  -- High Impedance  (used for tri-state I/O’s)
'W',-- Weak  signal
'L',  -- Weak signal with logic  0   as preference
'H',  -- Weak signal with logic  1   as preference
'-'   -- Don't care
```

Defines various FUNCTIONS, such as EDGE DETECTION

```
-------------------------------------------------------------------
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
```

Data Types: IEEE 1164 Standard Logic

- IEEE's **std_logic_1164** examples:

- STD_LOGIC

```
SIGNAL a: STD_LOGIC;

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

- STD_LOGIC_VECTOR

```
SIGNAL b: STD_LOGIC_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: >, <, <=, >=, =, SHIFT_LEFT, SHIFT_RIGHT

Normal STD_LOGIC_VECTORS: can't calculate with those!

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

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,8));
h <= STD_LOGIC_VECTOR(TO_SIGNED(j,8));
```

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
```

Signal Attributes

- ***Most*** signal attributes are ***not synthesizable***. They are used primarily for simulation.
- Attributes of enumeration types should not be used for synthesis, nor should the attributes POS, VAL, SUCC, PRED, LEFTOF, RIGHTOF.
- Many synthesis tools do not support these particular attributes, and attributes of enumeration types can become invalid during optimization.
- Use standard expressions **rising_edge(clk)**.

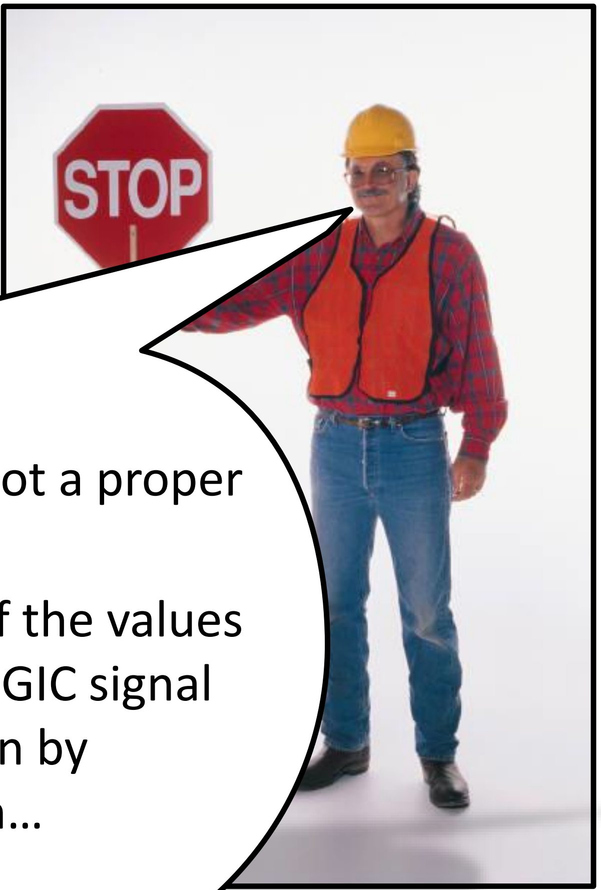
Simple Signal Assignment statement

- 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!

Example: DFF (bad)

- In many code snippets people use 'EVENT like this:

```
-------------------------------------
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 implementation OF dff IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
     IF (rst='1') THEN
        q <= '0';
     ELSIF (clk'EVENT AND clk='1') THEN
        q <= d;
     END IF
   END PROCESS;
          qi <= NOT(q);
END implementation;
```



Why is this not a proper DFF?

Hint: think of the values a STD_LOGIC signal can obtain by definition...

Example: DFF (good)

- 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;
```

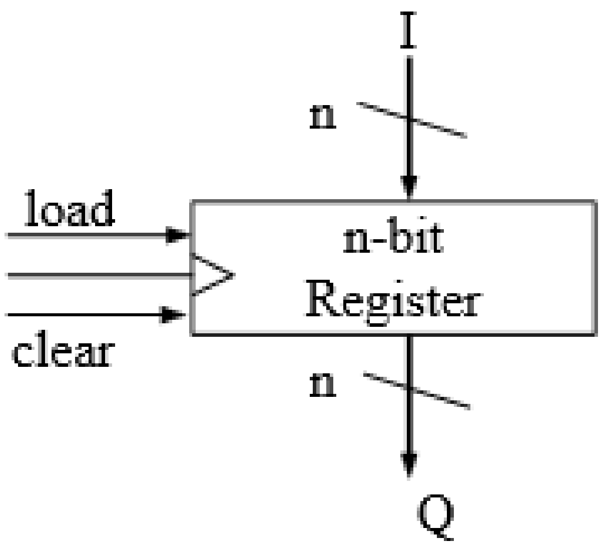
- Looks like this in the library:

```
FUNCTION rising_edge (SIGNAL s : std_ulogic)
RETURN BOOLEAN IS
BEGIN
   RETURN (s'EVENT AND (To_X01(s) = '1')
      AND (To_X01(s'LAST_VALUE) = '0'));
END;
```

- Make sure it is a real rising edge from 0 to 1, and not from X or Z.

D flip-flop

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



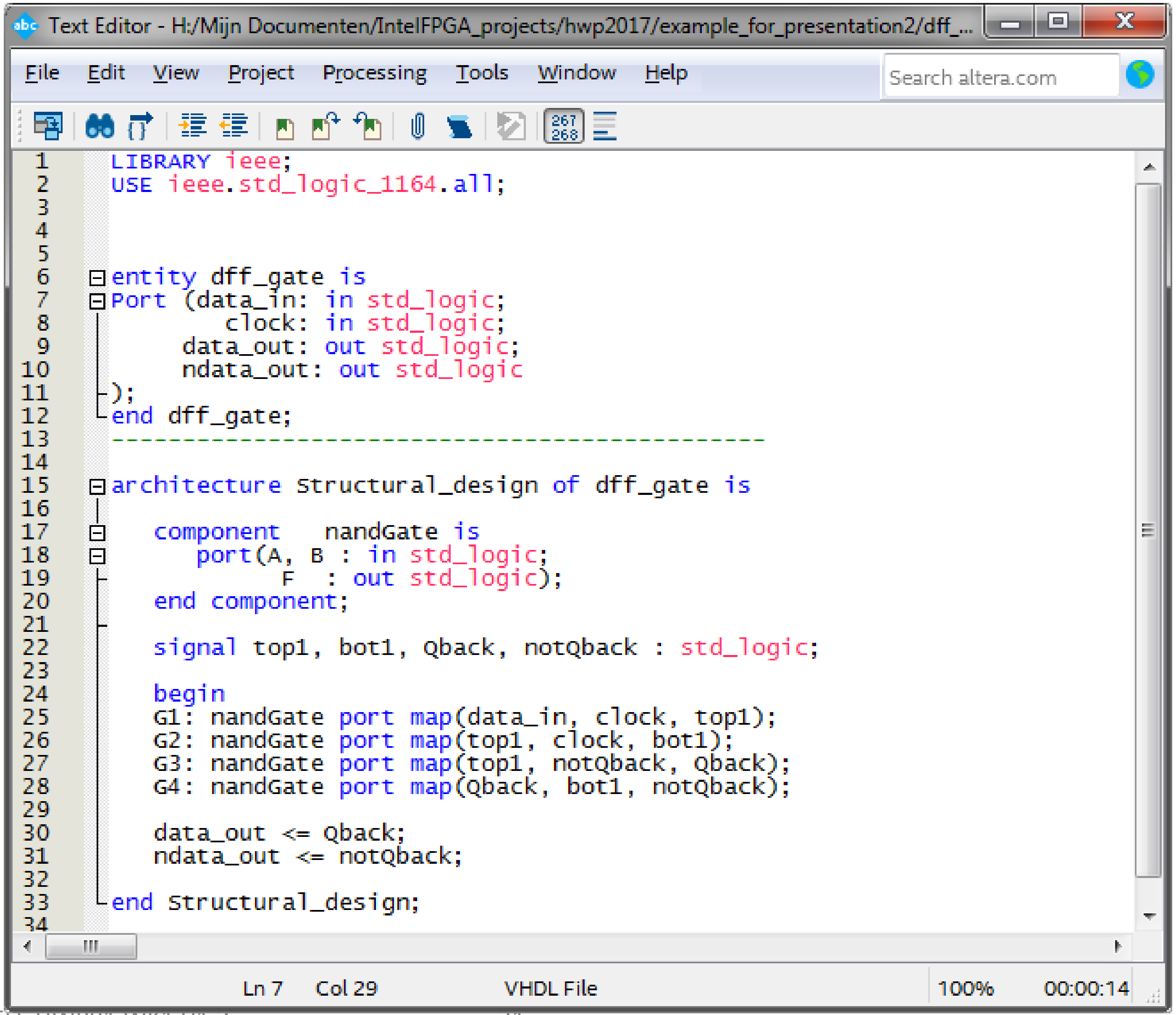
Q =
  0 if clear=1,
  I if load=1 and clock=1,
  Q(previous) otherwise.

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

Behavioral D flip-flop

```
---------------------------------------------
entity dff is
Port (  data_in:          in std_logic;
        clock:            in std_logic;
        data_out:         out std_logic
);
end dff;
----------------------------------------------
architecture behv of dff is
begin
    process(clock)
    begin
        -- clock rising edge

            if rising_edge(clock) then
             data_out <= data_in;
            end if;

    end process;
end behv;
---------------------------------------------
```

Structural D flip-flop


Text Editor - H:/Mijn Documenten/IntelFPGA_projects/hwp2017/example_for_presentation2/dff_...
File Edit View Project Processing Tools Window Help
Search altera.com

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;



entity dff_gate is
Port (data_in: in std_logic;
         clock: in std_logic;
      data_out: out std_logic;
      ndata_out: out std_logic
);
end dff_gate;
----------------------------------------------

architecture Structural_design of dff_gate is

   component   nandGate is
      port(A, B : in std_logic;
            F   : out std_logic);
   end component;

   signal top1, bot1, Qback, notQback : std_logic;

   begin
   G1: nandGate port map(data_in, clock, top1);
   G2: nandGate port map(top1, clock, bot1);
   G3: nandGate port map(top1, notQback, Qback);
   G4: nandGate port map(Qback, bot1, notQback);

   data_out <= Qback;
   ndata_out <= notQback;

end Structural_design;
```

Agenda

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

VHDL Golden Reference

- The VHDL Golden Reference Guide by DOULOS
- Use it as a reference
- Link: http://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl_golden_reference_guide.pdf

Summary

- Behavioral and structural design
- Use standard functions for conversion of data types
- Verify the functional behavior of your design with test benches

Homework

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

- Homework:
  - 2.1, 2.2a, 3.1, 3.20, 3.22, 3.24

- Next week:
  - Combinational versus sequential design
  - Concurrent and sequential code
  - Signals versus variables