Basic Components in VHDL
Binary logic

- Logic “variables” on which logic “functions” work
  - Variables: Binary (0 or 1)
  - E.g. A can have the value 0 or 1
    - $A = 0$, lamp is out
    - $A = 1$, lamp is on
Binary logic

- Logical levels
  - In digital electronics:
    - signal level: HIGH (1) of LOW (0)
  - (e.g.) TTL- logic:
    - LOW: voltage between 0V en 0.8V
    - HIGH: voltage between 2V en 5 V
  - On Spartan 3E: 3.3V, 2.5V and 1.2V
Logical statements

A logical statement can be true or false

The results usually depends on some independent parameters (inputs)

In the Boolean logic:

- **AND** = “.”
- **OR** = “+”
- **NOT** = “’” or “¯” or “ \ ”
- **IF** = “= “
Binary logic

Logic statements & equations

Example: We go for an outdoor swim (S) if the weather is nice (W), the pool is open (P) and it is not crowded (C).

So: "swim" is true (S = 1) if "nice weather" is true (W = 1) and "pool open" is true (P = 1) and "crowded" is not true (C = 0)

Logic equation:
\[ S = W \cdot P \cdot C' \]

Schematic:
Logic gates

Not gate

\[ F = \overline{A} = /A = A' \]
Logic gates

And gate

IEC-SYMBOOL

WAARHEIDSTABEL

All possibilities for 2 inputs

LOGISCHE VERGELIJKING

F = A \cdot B = AB

TIJDSDIAGRAMMA

VHDL - introduction

14/10/2015
Logic gates

Or gate

**Logic symbols**

- **Symbol**: 74XX32
- **Diagram**: [Or gate diagram]

**Truth table**

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>F (OR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Logical equation**

\[ F = A + B \]
Logic gates

NAND gate

**IEC-SYMBOL**

**WAARHEIDSTABLE**

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>AND</th>
<th>F (NAND)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**LOGISCHE VERGELIJKING**

\[ F = \overline{A \cdot B} = \overline{AB} \]

**TIJDSDIAGRAMMA**

\[ \text{Diagram for } A, B, F \]
Logic gates

NOR gate

**IEC-SYMBOOL**

![IEC-SYMBOOL](image)

**WAARHEIDSTABEL**

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>OR</th>
<th>F (NOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**LOGISCHE VERGELIJKING**

\[ F = A + B \]

**TIJDSDIAGRAMMA**

![TIJDSDIAGRAMMA](image)
Logic gates

EXOR gate

IEC-SYMBOOL

WAARHEIDSTABEL

LOGISCHE VERGELIJKING
\[ F = \overline{A}B + \overline{A} \overline{B} \]
\[ F = A \oplus B \]

TIJDSDIAGRAMMA

VHDL - introduction

14/10/2015
Logic gates

EXNOR gate

**IEC-SYMBOOL**

74xx810

A

B

F

**WAARHEIDSTABEL**

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>F (EXNOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**LOGISCHE VERGELEJKING**

\[
F = \overline{AB} + AB
\]

\[
F = A \oplus B
\]

**TIJDSDIAGRAMMA**

A

B

F
Logic gates

American symbols

<table>
<thead>
<tr>
<th>AND-POORT</th>
<th>OR-POORT</th>
<th>NOT-POORT</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Circuit Symbol" /></td>
<td><img src="image" alt="Circuit Symbol" /></td>
<td><img src="image" alt="Circuit Symbol" /></td>
</tr>
<tr>
<td>NAND-POORT</td>
<td>NOR-POORT</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Circuit Symbol" /></td>
<td><img src="image" alt="Circuit Symbol" /></td>
<td></td>
</tr>
<tr>
<td>EXOR-POORT</td>
<td>EXNOR-POORT</td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Circuit Symbol" /></td>
<td><img src="image" alt="Circuit Symbol" /></td>
<td></td>
</tr>
</tbody>
</table>
Logic gates

• schematic
• logic equation
• truth table
• timing diagram

=> they are all connected
Logic gates

- From schematic

- To truth table

All possibilities for 3 inputs

\[
\begin{align*}
K &= \overline{B} \\
L &= K \cdot C \\
F &= A + L
\end{align*}
\]

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>K</th>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Logic gates

- From schematic

\[ F = A + L \]
\[ F = A + (K \cdot C) \]
\[ F = A + (\overline{B} \cdot C) \]

- To logic equation
Logic gates

- From schematic
- To timing diagram
Logic gates

- From truth table

- To logic equation

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>F</th>
<th>productterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$A \cdot \overline{B} \cdot \overline{C}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A \cdot \overline{B} \cdot \overline{C}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A \cdot \overline{B} \cdot C$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$A \cdot \overline{B} \cdot C$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A \cdot B \cdot C$</td>
</tr>
</tbody>
</table>

$$F = A \overline{B} \overline{C} + A \overline{B} \overline{C} + \overline{A} \overline{B} \overline{C} + A \overline{C} + A B C$$
Exercise 1

- Truth table
- Logic equation
- Timing diagram
Exercise 1

• Truth table

• F1 is 1 if
  • D is ‘1’ or
  • H is ‘1’ => if G is ‘0’
  => if B and C both ‘0’ or
  • F is ‘1’ => B is ‘1’ or E is ‘1’ =>
if A and C both ‘1’
Exercise 1

- Logic equation:
  - $F_1 = F + H + D$
  - $= E + B + G’ + D$
  - $= (A.C + B) + (B + C)’ + D$
Logic ICs

Single gates

DIL : dual in line
DIP: dual in line package
### Logic ICs

#### Single gates

<table>
<thead>
<tr>
<th>NOT</th>
<th>AND</th>
<th>OR</th>
<th>NAND</th>
<th>NOR</th>
<th>EXOR</th>
<th>EXNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 input</td>
<td>74xx04</td>
<td>4049</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 inputs</th>
<th>74xx08</th>
<th>74xx32</th>
<th>74xx00</th>
<th>74xx02</th>
<th>74xx86</th>
<th>74xx810</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4081</td>
<td>4071</td>
<td>4011</td>
<td>4001</td>
<td>4070</td>
<td>4077</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 inputs</th>
<th>74xx11</th>
<th>74xx10</th>
<th>74xx27</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4073</td>
<td>4075</td>
<td>4023</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 inputs</th>
<th>74xx21</th>
<th>74xx20</th>
<th>74xx25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4082</td>
<td>4072</td>
<td>4012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6 inputs</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>74xx260</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8 inputs</th>
<th>74xx30</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4068</td>
<td>4078</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12 inputs</th>
<th>74xx134</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13 inputs</th>
<th>74xx133</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Logic ICs

Single gates

- Pin numbering

- Datasheet
Logic ICs

Single gates

- Impossible to use for complex logic equations!
Logic ICs

FPGAs
Logic ICs

FPGAs
Logic ICs

FPGAs
- Configurable Logic Blocks
Logic ICs

FPGAs

• Logic = hardware truth table = Look-Up Table

Xilinx cells are based on the use of SRAM as a look-up table. The truth table for a K-input logic function is stored in a $2^K \times 1$ SRAM.

The address lines of the SRAM function as inputs and the output (data) line of the SRAM provides the value of the logic function.
Logic ICs

FPGAs

- D-flipflop is memory element
- It reads input on clock flank
- Is only used when clock is used
- Clock should drive all flipflops at the same time
Logic ICs

FPGAs

- Interconnection

**SRAM** to make interconnections
Logic ICs

FPGAs
- Interconnection
Logic ICs

FPGAs

- I/O Block
  - Input – output conditioning
  - IO standards
  - Inverting
  - Delay
  - Pull-up / -down
  - Slew rate & drive strength

Configuration with SRAM
Logic ICs

FPGAs
- I/O Block

<table>
<thead>
<tr>
<th>IOSTANDARD</th>
<th>Output Drive Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>LVTTL</td>
<td>✓</td>
</tr>
<tr>
<td>LVCMOS33</td>
<td>✓</td>
</tr>
<tr>
<td>LVCMOS25</td>
<td>✓</td>
</tr>
<tr>
<td>LVCMOS18</td>
<td>✓</td>
</tr>
<tr>
<td>LVCMOS15</td>
<td>✓</td>
</tr>
<tr>
<td>LVCMOS12</td>
<td>✓</td>
</tr>
</tbody>
</table>
Logic ICs

FPGAs

- Spartan 3E: architecture => extra hardware
Logic ICs

FPGAs
- Spartan 3E: architecture

Table 1: Summary of Spartan-3E FPGA Attributes

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>Equivalent Logic Cells</th>
<th>CLB Array (One CLB = Four Slices)</th>
<th>Distributed RAM bits(1)</th>
<th>Block RAM bits(1)</th>
<th>Dedicated Multipliers</th>
<th>DCMs</th>
<th>Maximum User I/O</th>
<th>Maximum Differential I/O Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S100E</td>
<td>100K</td>
<td>2,160</td>
<td>22 × 16</td>
<td>240 × 960</td>
<td>15K</td>
<td>72K</td>
<td>4</td>
<td>2</td>
<td>108</td>
</tr>
<tr>
<td>XC3S250E</td>
<td>250K</td>
<td>5,508</td>
<td>34 × 26</td>
<td>612 × 2,448</td>
<td>38K</td>
<td>216K</td>
<td>12</td>
<td>4</td>
<td>172</td>
</tr>
<tr>
<td>XC3S500E</td>
<td>500K</td>
<td>10,476</td>
<td>46 × 34</td>
<td>1,164 × 4,656</td>
<td>73K</td>
<td>360K</td>
<td>20</td>
<td>4</td>
<td>232</td>
</tr>
<tr>
<td>XC3S1200E</td>
<td>1200K</td>
<td>19,512</td>
<td>60 × 46</td>
<td>2,168 × 8,672</td>
<td>136K</td>
<td>504K</td>
<td>28</td>
<td>8</td>
<td>304</td>
</tr>
<tr>
<td>XC3S1600E</td>
<td>1600K</td>
<td>33,192</td>
<td>76 × 58</td>
<td>3,688 × 14,752</td>
<td>231K</td>
<td>648K</td>
<td>36</td>
<td>8</td>
<td>376</td>
</tr>
</tbody>
</table>

Notes:
1. By convention, one Kb is equivalent to 1,024 bits.
Logic ICs

FPGAs

• Spartan 3E: DCM

Clock

• frequency,
• phase shift
• skew
employs a Delay Locked Loop
Logic ICs

FPGAs

• Spartan 3E: Clock Distribution Network
Logic ICs

FPGAs

- Spartan 3E: Configuration

Where does the bit-file come from?
Basys2: 2 options
  - Master Serial: Platform Flash: XCF02S
  - JTAG: Through USB
VHDL

Spartan 3E: XC3S100E

100K gates!? How in earth can you address them?

VHDL!!!
VHDL

- HDL = Hardware Description Language
- V = VHSIC = Very High Speed Integrated Circuit
  - Language used to describe hardware (not for programming)
  - Result = hardware
    - IC
    - FPGA
Structure of VHDL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin

    led <= sw;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
  Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
        led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin

  led <= sw;

end Behavioral;
Structure of VHDL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
  Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
         led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin
  led <= sw;
end Behavioral;

Entity declaration

Entity <ent_name> is
  Port ();
end <ent_name>;

• Interface description of logic component
• Port signals:
  • Signal name
  • Signal direction
  • Data types
  • Signal width
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
            led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin
    led <= sw;
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
  Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
        led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin
  led <= sw;
end Behavioral;
VHDL

Structure of VHDL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
  Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
       led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin
  led <= sw;
end Behavioral;

Architecture

Architecture <arch_name> of <entity_name> is
begin
end <arch_name>;

• Definition part:
  • signal assignments
  • processes
  • component instantiations
  • ...

VHDL - introduction

14/10/2015
Lab: startup
Project set up

- Start ISE software: 32 bit Project Navigator
Project set up

- File => New Project

Choose meaningful Directory (Name/Project_name)
Project set up

- Project Settings (information is available on the device)
Project set up

- Project Settings (information is available on the device)
  - Family: Spartan3E
  - Device: XC3S100E
  - Package: CP132
  - Speed: -4
  - Synthesis Tool: XST
  - Simulator: ISim
Project set up

- Right click on **project** => **new source** => **VHDL Module** & give meaningful name
Project set up

- Define inputs/outputs, individual ports or busses, input, output or both
Project set up

- Write code in editor

```vhdl
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
-- use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNIISM;
--use UNISIM.VComponents.all;

entity button_led is
  Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
        led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is
begin
  led <= sw;
end Behavioral;
```
Project set up

- Write code in editor
  - This will describe the internal behavior of the logic component:
    - what does it do?
    - what is the interface?
Project set up

• Simulate the design with a testbench

• Click on simulation in the left upper panel:

• Write a testbench to
  - simulate the clock
  - simulate all possible input combinations
Project set up

- Add user constraints, a UCF-file.
  - To which FPGA pins are the ports from the entity connected?
  - How are the ports connected to the outside world?
  - What is the I/O standard?
  - How is the I/O block configured?
Project set up

- Add user constraints, a UCF-file.
  - push “I/O Pin Planning”
Project set up

- Push “Edit Constraints” on the processes panel, left.
- Copy all suitable pin locations from the UCF-file
Project set up

• An example is seen below:

- NET "sw<7>" LOC = "N3"; # Bank = 2, Signal name = SW7
- NET "sw<6>" LOC = "E2"; # Bank = 3, Signal name = SW6
- NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5
- NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4
- NET "sw<3>" LOC = "B4"; # Bank = 3, Signal name = SW3
- NET "sw<2>" LOC = "K3"; # Bank = 3, Signal name = SW2
- NET "sw<1>" LOC = "L3"; # Bank = 3, Signal name = SW1
- NET "sw<0>" LOC = "P11"; # Bank = 2, Signal name = SW0
- NET "btn" LOC = "G12"; # Bank = 0, Signal name = BTN0
- NET "Led<3>" LOC = "P6"; # Bank = 2, Signal name = LD3
- NET "Led<2>" LOC = "P7"; # Bank = 3, Signal name = LD2
- NET "Led<1>" LOC = "M11"; # Bank = 2, Signal name = LD1
- NET "Led<0>" LOC = "M5"; # Bank = 2, Signal name = LD0
Project set up

• **Generate Programming File** runs through: synthesize => implement design
Project set up

- To implement the design, open Digilent’s Adept software
- Browse bit-file and then push program
ISE software

1. Toolbar
2. Sources window
3. Processes window
4. Workspace
5. Transcript window
ISE software

- Language templates
  - Useful tool to find coding examples
ISE software

--The following logical operators are used in conditional
--such as an if statement in order to specify the condition:

    NOT ---- Not True
    AND ---- Both Inputs True
    OR ---- Either Input True
    = ---- Inputs Equal
    /= ---- Inputs Not Equal
    < ---- Less-than
    <= ---- Less-than or Equal
    > ---- Greater-than
    >= ---- Greater-than or Equal
Solutions

There are three design methods in VHDL

1. **Dataflow design**
2. **Behavioural design**
3. **Structural design**
Solutions

Dataflow design

C<=A xor B;

• Defines direct relation (single to couple of gates) between output and input
• Sequence not important (parallelism)
• Asynchronous
• Difficult for complex functions
• Difficult to understand
Lab 0

Make a component which connects all switches to all LEDs
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity button_led is
    Port ( sw : in  STD_LOGIC_VECTOR (3 downto 0);
        led : out STD_LOGIC_VECTOR (3 downto 0));
end button_led;

architecture Behavioral of button_led is

Begin

    led <= sw;

end Behavioral;
Lab 1

Make a logic components to test all the gates

- Led 7 = SW1 and SW2 and SW3
- Led 5 = SW1 or SW2 or SW3
- Led 5 = SW1 and SW2
- Led 4 = SW1 nand SW2
- Led 3 = SW1 or SW2
- Led 2 = SW1 nor SW2
- Led 1 = SW1 xor SW2
- Led 0 = SW1 xnor SW2
Lab 1 - solution

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity gates is
  port(
    sw1 : in STD_LOGIC;
    sw2 : in STD_LOGIC;
    sw3 : in STD_LOGIC;
    led : out STD_LOGIC_VECTOR(7 downto 0)
  );
end gates;

architecture behavior of gates is
begin
  led(7) <= sw1 and sw2 and sw3;
  led(6) <= sw1 or sw or sw32;
  led(5) <= sw1 and sw2;
  led(4) <= sw1 nand sw2;
  led(3) <= sw1 or sw2;
  led(2) <= sw1 nor sw2;
  led(1) <= sw1 xor sw2;
  led(0) <= sw1 xnor sw2;
end gates2;

Parallellism:
Statements run concurrent to each other
Lab 2

Make logic components to implement these schematics
Lab 2 - solutions

entity lab2_1 is
  Port (    sw0 : in  STD_LOGIC;
            sw1 : in  STD_LOGIC;
            sw2 : in  STD_LOGIC;
            sw3 : in  STD_LOGIC;
            led0 : out STD_LOGIC);
end lab2_1;

architecture Behavioral of lab2_1 is
begin
  led0 <= (sw0 and sw1) xor (sw2 or sw3);
end Behavioral;
Lab 2 - solutions

entity lab2_1 is
  Port (  sw0 : in  STD_LOGIC;
          sw1 : in  STD_LOGIC;
          sw2 : in  STD_LOGIC;
          sw3 : in  STD_LOGIC;
          led0 : out  STD_LOGIC);
end eef1_8;

architecture Behavioral of lab2_1 is

begin
  led0 <= (sw0 and sw1) xor (sw0 or sw2);

end Behavioral;
Combinatorial Logic

Multiplexer - principle

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Y0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Y1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Y2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Y3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Y0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Y1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Y2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Y3</td>
</tr>
</tbody>
</table>
Lab 3

4 to 1 MUX

- D0 when A1A0 is “00”
- D1 when A1A0 is “01”
- D2 when A1A0 is “10”
- D3 when A1A0 is “11”
Lab 3 - solutions

4 to 1 MUX – with gates => increasingly complicated

entity mux41 is
  port(
    d : in STD_LOGIC_VECTOR(3 downto 0);
    a : in STD_LOGIC_VECTOR(1 downto 0);
    z : out STD_LOGIC
  );
end mux41;

architecture behavior of mux41 is

begin
  z <= (not a(1) and not a(0) and d(0))
    or (not a(1) and a(0) and d(1))
    or (a(1) and not a(0) and d(2))
    or (a(1) and a(0) and d(3));
end behavior;
Solutions

Behavioural design

Processes

```vhdl
process(sensitivity list: signal-names) begin
  -- sequential code:
  e.g.: if (condition) then
         action1;
  else
         action2;
  end if;
end process;
```
Solutions

Behavioural design

Processes

- **If** (condition) **then** ... **elsif** (condition) **then** ... **else** ... **end if**;
- **Case** (signal_name) **is**
  - **when** condition 1 => ...;
  - **when** condition 2 => ...;
  ```
  end case;
  ```
- **For** signal_name **in** start_value **to** stop_value **loop**
  ```
  ...
  ```
  ```
  end loop;
  ```
Solutions

Behavioural design

- Sequence important
- Asynchronous
- Divides complex functions in manageable parts
- Easier to understand
- For combinatorial: all input signals in the sensitivity list
Lab 3 - solutions

4 to 1 MUX – with case statement

architecture behavior of mux41 is

begin
  p1: process(d, a) begin
    case a is
      when "00" => z <= d(0);
      when "01" => z <= d(1);
      when "10" => z <= d(2);
      when "11" => z <= d(3);
      when others => z <= d(0);
    end case;
  end process;
end behavior;
Combinatorial Logic

Demultiplexer - principle

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>Z3</th>
<th>Z2</th>
<th>Z1</th>
<th>Z0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>Z3</th>
<th>Z2</th>
<th>Z1</th>
<th>Z0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
1 to 4 DEMUX

- Y is Z0 when A1A0 is “00”
- Y is Z1 when A1A0 is “01”
- Y is Z2 when A1A0 is “10”
- Y is Z3 when A1A0 is “11”

⇒ Difficulty: what about the other outputs?
⇒ Do not leave them un-assigned: otherwise latching (= unintended memory synthesis)

Use concatenate: ‘&’ => binds different signals together
e.g. Z <= ‘0’ & ’0’ & ’0’ & Y;
Lab 4 - solutions

1 to 4 DEMUX

entity demux14 is
    port(
        y : in STD_LOGIC;
        a : in STD_LOGIC_VECTOR(1 downto 0);
        z : out STD_LOGIC_VECTOR(3 downto 0)
    );
end demux14;
architecture behavior of demux14 is

begin
p1: process(y, a) begin
    case a is
    when "00" => z <= '0'&'0'&'0'&y;
    when "01" => z <= '0'&'0'&y&'0';
    when "10" => z <= '0'&y&'0'&'0';
    when "11" => z <= y&'0'&'0'&'0';
    when others => z <= '0'&'0'&'0'&'0';
    end case;
end process;
end behavior;
Solutions

Behavioural design

• Inter-process communication is done with signals

• Signals are declared between “architecture” and “begin”

• Design can be structured with different processes, connected with signals
Lab 5

Make logic components to implement these schematics with the use of processes and signals.
Lab 5 - solutions

entity lab5_1 is
Port (sw0 : in STD_LOGIC;
     sw1 : in STD_LOGIC;
     sw2 : in STD_LOGIC;
     sw3 : in STD_LOGIC;
     led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
signal and_sig: std_logic;
signal or_sig: std_logic;
begin
    and_gt: process (sw0, sw1) begin
        and_sig <= sw0 and sw1;
    end process;
    or_gt: process (sw2, sw3) begin
        or_sig <= sw2 or sw3;
    end process;
    xor_gt: process (or_sig, and_sig) begin
        led <= or_sig xor and_sig;
    end process;
end Behavioral;
Lab 5 - solutions

entity lab5_1 is
Port (  sw0 : in STD_LOGIC;
        sw1 : in STD_LOGIC;
        sw2 : in STD_LOGIC;
        sw3 : in STD_LOGIC;
        led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
signal and_sig: std_logic;
signal or_sig: std_logic;
begin
  and_gt: process (sw0, sw1) begin
    and_sig <= sw0 and sw1;
  end process;
  or_gt: process (sw2, sw3) begin
    or_sig <= sw2 or sw3;
  end process;
  xor_gt: process (or_sig, and_sig) begin
    led <= or_sig xor and_sig;
  end process;
end Behavioral;
entity lab5_1 is
    Port (
        sw0 : in STD_LOGIC;
        sw1 : in STD_LOGIC;
        sw2 : in STD_LOGIC;
        sw3 : in STD_LOGIC;
        led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
    signal and_sig: std_logic;
    signal or_sig: std_logic;
    begin
        and_gt: process (sw0, sw1) begin
            and_sig <= sw0 and sw1;
        end process;
        or_gt: process (sw2, sw3) begin
            or_sig <= sw2 or sw3;
        end process;
        xor_gt: process (or_sig, and_sig) begin
            led <= or_sig xor and_sig;
        end process;
    end Behavioral;
Lab 5 - solutions

entity lab5_1 is
Port (  sw0 : in STD_LOGIC;
  sw1 : in STD_LOGIC;
  sw2 : in STD_LOGIC;
  sw3 : in STD_LOGIC;
  led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
signal and_sig: std_logic;
signal or_sig: std_logic;
begin
  and_gt: process (sw0, sw1) begin
    and_sig <= sw0 and sw1;
  end process;
  or_gt: process (sw2, sw3) begin
    or_sig <= sw2 or sw3;
  end process;
  xor_gt: process (or_sig, and_sig) begin
    led <= or_sig xor and_sig;
  end process;
end Behavioral;
Lab 5 - solutions

entity lab5_1 is
Port (  sw0 : in STD_LOGIC;
    sw1 : in STD_LOGIC;
    sw2 : in STD_LOGIC;
    sw3 : in STD_LOGIC;
    led : out STD_LOGIC);
end lab5_1;

architecture Behavioral of lab5_1 is
  signal and_sig: std_logic;
  signal or_sig: std_logic;
begin
  and_gt: process (sw0, sw1) begin
    and_sig <= sw0 and sw1;
  end process;
  or_gt: process (sw2, sw3) begin
    or_sig <= sw2 or sw3;
  end process;
  xor_gt: process (or_sig, and_sig) begin
    led <= or_sig xor and_sig;
  end process;
end Behavioral;
Solutions

Structural design

• Design can be done with the use of components
• Different components are connected with signals
• Both components and signals are declared in the declarative part between “architecture” and “begin”
• In the definition part the actual component is used, instantiated
• Divide problems into sub-problems
• Reuse – one component can be used several times

⇒ the difference in actual component used is made in instance name and connected signals – naming of ports remain the same
Structural design

--component declarations
component <entity-name>
port (
    <signal-names : mode signal-type>)
end component;
...

--signals to connect blocks
begin
<component-name> : <entity-name>
    port map (
        <signal-mapping>
    );
...

Solutions

Structural design

- Entity declaration of the component used and the component declaration in the top-component are very similar

⇒ The entity declaration of the sub-module can be used for the component declaration in the top-module

⇒ The component declaration of the top-module can be used for the entity declaration of the sub-module

- Copy – paste

- Change “component” to “entity” and visa versa
Lab 6

Use a top-module to connect a MUX and a DEMUX

• Declare components
• Declare signals
• Instantiate – use components
Lab 6 - solution

entity muxdemux is
  port(
    d : in STD_LOGIC_VECTOR(3 downto 0);
    a : in STD_LOGIC_VECTOR(1 downto 0);
    z : out STD_LOGIC_VECTOR(3 downto 0)
  );
end muxdemux;
Lab 6 - solution

architecture behavior of muxdemux is

component mux41 is
    port(
        d : in STD_LOGIC_VECTOR(3 downto 0);
        a : in STD_LOGIC_VECTOR(1 downto 0);
        z : out STD_LOGIC
    );
end component mux41;

component demux14 is
    port(
        y : in STD_LOGIC;
        a : in STD_LOGIC_VECTOR(1 downto 0);
        z : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component demux14;

signal y : in STD_LOGIC;

Copy – paste
Entity => component

Signal name can be the same as the port name, but is not obliged
Lab 6 - solution

Begin

Mux1: mux41 is
port map(d => d, a => a, z => y);

Demux1: demux14 is
port map(y => y, a => a, z => z);

end behavior;

- Instantiate – use component
- It is possible to use a second Mux2 if desired
- Connect ports, *left*, to signals, *right*
- Signals can be signals declared in the *entity declaration* or signals declared in the *architecture*
- Ports and signals can have the same name
Simulation

Testbench

How do you test a design?
Answer: implement the hardware on an FPGA-board and test all input options.
This process can be automated in a testbench

Why?

Synthesis (= the process of translating VHDL to the proper hardware) consumes a lot of time, up to hours.
Solution: simulate first, synthesize later
Simulation

Testbench

• A testbench is a VHDL module with one component, the component we want to test.
• It has not interface with the outside world: empty entity
  1. The component is declared
  2. The input and output signals are declared
  3. The component is instantiated
  4. The input and output signals are connected to the instantiated component
  5. In two processes all input signals are asserted at a certain moment of time
    • One process for the clock
    • One process for the other inputs
  6. The output is analyzed in a waveform or in an output txt-file.
• It is possible to look at internal signals
Simulation

Testbench
Lab 7

Simulate Lab 0 – 1 – 2 – 3 - 4

- Open the project
- Click on simulation in the left upper panel
- Add new source to the project
- Write a testbench to
  - simulate the clock
  - simulate all possible input combinations
ENTITY button_led_TB IS
END button_led_TB;

ARCHITECTURE behavior OF button_led_TB IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT button_led
PORT(
    sw : IN std_logic_vector(3 downto 0);
    led : OUT std_logic_vector(3 downto 0)
);
END COMPONENT;

--Inputs
signal sw : std_logic_vector(3 downto 0) := (others => '0');

--Outputs
signal led : std_logic_vector(3 downto 0);
BEGIN

-- Instantiate the Unit Under Test (UUT)

library work;
use work.button_led;

uut: button_led PORT MAP ( 
   sw => sw, 
   led => led 
);

Instantiate component
uut= name
connect signals to ports
BEGIN
  -- Stimulus process
  stim_proc: process
  begin
    wait for 100 ns;
    sw <= "0000";
    wait for 100 ns;
    sw <= "0001";
    wait for 100 ns;
    sw <= "0010";
    wait for 100 ns;
    sw <= "0011";
    wait for 100 ns;
    sw <= "0100";
    wait for 100 ns;
    sw <= "0101";
    wait for 100 ns;
    sw <= "0110";
    wait;
  end process;
BEGIN

-- Instantiate the Unit Under Test (UUT)
  uut: oef1_1 PORT MAP (  
      sw => sw,  
      led => led  
  );
-- Stimulus process
  stim_proc: process
  begin
    wait for 100 ns;
    for i in 0 to 20 loop
      wait for 40 ns;
      sw <= sw + 1;
    end loop;
    wait;
  end process;

Problem:
+ - operant is not declared in the libraries
=> extra libraries needed
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

Edit => Language Templates...
VHDL => Common Constructs => Library Declarations/Use => Commonly Used
Lab 7 – solution Lab 0 – variants

ARCHITECTURE behavior OF button_led_TB IS
...
--Inputs
    signal sw : std_logic_vector(3 downto 0) := (others => '0');
    signal sw : std_logic_vector(3 downto 0);
...
BEGIN
...

What happens in simulation?
Lab 7 – solution Lab 0 – variants

Possibilities for std_logic (not only ‘0’ and ‘1’)

type STD_ULOGIC is (  
  `U`, -- uninitialized  
  `X`, -- strong 0 or 1 (= unknown)  
  `0`, -- strong 0  
  `1`, -- strong 1  
  `Z`, -- high impedance  
  `W`, -- weak 0 or 1 (= unknown)  
  `L`, -- weak 0  
  `H`, -- weak 1  
  `-`, -- don’t care);

By default first value in enumeration
Combinatorial Logic

BIN/DEC decoder

<table>
<thead>
<tr>
<th>BIN/DEC</th>
<th>B1, B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, 1, 3, 2</td>
</tr>
<tr>
<td>1</td>
<td>4, 5, 7, 6</td>
</tr>
<tr>
<td>2</td>
<td>12, 13, 15, 14</td>
</tr>
<tr>
<td>3</td>
<td>8, 9, 11, 10</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|cccc}
0 = \overline{B_3}B_2B_1B_0 & 4 = \overline{B_3}B_2B_1B_0 & 8 = \overline{B_3}B_2B_1B_0 & 12 = \overline{B_3}B_2B_1B_0 \\
1 = B_3B_2B_1B_0 & 5 = B_3B_2B_1B_0 & 9 = B_3B_2B_1B_0 & 13 = B_3B_2B_1B_0 \\
2 = B_3B_2B_1B_0 & 6 = B_3B_2B_1B_0 & 10 = B_3B_2B_1B_0 & 14 = B_3B_2B_1B_0 \\
3 = B_3B_2B_1B_0 & 7 = B_3B_2B_1B_0 & 11 = B_3B_2B_1B_0 & 15 = B_3B_2B_1B_0 \\
\end{array}
\]
Lab 8

Make a 3 to 8 Decoder in VHDL

- With gates
- With a for loop
- Simulate
entity decode38 is
  port(
    a : in STD_LOGIC_VECTOR(2 downto 0);
    y : out STD_LOGIC_VECTOR(7 downto 0)
  );
end decode38;

architecture behavior of decode38 is
begin
  y(0) <= not a(2) and not a(1) and not a(0);
  y(1) <= not a(2) and not a(1) and a(0);
  y(2) <= not a(2) and a(1) and not a(0);
  y(3) <= not a(2) and a(1) and a(0);
  y(4) <= a(2) and not a(1) and not a(0);
  y(5) <= a(2) and not a(1) and a(0);
  y(6) <= a(2) and a(1) and not a(0);
  y(7) <= a(2) and a(1) and a(0);
end decode38a;
architecture behavior of decode38 is
  begin
    process(a)
      variable j: integer;
      begin
        j := conv_integer(a);
        for i in 0 to 7 loop
          if(i = j) then
            y(i) <= '1';
          else
            y(i) <= '0';
          end if;
        end loop;
      end process;
  end behavior;
Combinatorial Logic

DEC/BCD encoder without priority

<table>
<thead>
<tr>
<th>DEC</th>
<th>BCD3</th>
<th>BCD2</th>
<th>BCD1</th>
<th>BCD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

BCD0 = 1 + 3 + 5 + 7 + 9
BCD1 = 2 + 3 + 6 + 7
BCD2 = 4 + 5 + 6 + 7
BCD3 = 8 + 9
Combinatorial Logic

DEC/BCD encoder with priority

- **HIGH PRIORITY**
  - The highest number appears on the output

- **LOW PRIORITY**
  - The lowest number appears on the output
Combinatorial Logic

DEC/BCD encoder with priority

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>BCD3</th>
<th>BCD2</th>
<th>BCD1</th>
<th>BCD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

BCD0 = 12468 + 3468 + 568 + 78 + 9
BCD1 = 24589 + 34589 + 689 + 789
BCD2 = 489 + 589 + 689 + 789
BCD3 = 8 + 9
Lab 9

Make a 8 to 3 Encoder in VHDL

• With gates
• With a for loop
• Simulate
entity encode83 is
port(
x : in STD_LOGIC_VECTOR(7 downto 0);
y : out STD_LOGIC_VECTOR(2 downto 0);
valid: out STD_LOGIC
);
end encode83;
Lab 9 – solution - gates

architecture behavior of encode83 is
begin

process(x)
variable valid_var: STD_LOGIC;
begin

y(2) <= x(7) or x(6) or x(5) or x(4);
y(1) <= x(7) or x(6) or x(3) or x(2);
y(0) <= x(7) or x(5) or x(3) or x(1);
valid_var := '0';
for i in 7 downto 0 loop
    valid_var := valid_var or x(i);
end loop;
valid <= valid_var;
end process;

end behavior;

Check if there was an input
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_unsigned.all;

entity encode83 is
  port(
    x : in STD_LOGIC_VECTOR(7 downto 0);
    valid : out STD_LOGIC;
    y : out STD_LOGIC_VECTOR(2 downto 0)
  );
end encode83;
Lab 9 – solution – for loop

architecture behavior of encode83 is
begin
process(x)
variable j: integer;
begin
y <= "000";
valid <= '0';
for j in 0 to 7 loop
if x(j) = '1' then
y <= conv_std_logic_vector(j,3);
valid <= '1';
end if;
end loop;
end process;
end behavior;

Converts integer to binary

Priority?
Combinatorial Logic

BCD/7-segment decoder

BCD to 7 Segment Decoder

7-Segment LED Display
Combinatorial Logic

BCD/7-segment decoder
Combinatorial Logic

BCD/7-segment decoder

7-segment decoder active low outputs → COMMON ANODE DISPLAY

<table>
<thead>
<tr>
<th>DEC</th>
<th>BCD3</th>
<th>BCD2</th>
<th>BCD1</th>
<th>BCD0</th>
<th>g</th>
<th>f</th>
<th>e</th>
<th>d</th>
<th>c</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Lab 10

Make a BCD/7-segment decoder

• Connect to the 7-segment display: active low
• Anodes are also connected active low
• Simulate
Lab 10

VHDL - introduction

14/10/2015
Lab 10 - solution

architecture behavior of hex7seg is
begin
  process(x)
  begin
    case x is
      when "0000"  => a_to_g  <= "0000001";  --0
      when "0001"  => a_to_g  <= "1001111";  --1
      when "0010"  => a_to_g  <= "0010010";  --2
      when "0011"  => a_to_g  <= "0000110";  --3
      when "0100"  => a_to_g  <= "1001100";  --4
      when "0101"  => a_to_g  <= "0100100";  --5
      when "0110"  => a_to_g  <= "0100000";  --6
      when "0111"  => a_to_g  <= "0001101";  --7
      when "1000"  => a_to_g  <= "0000000";  --8
      when "1001"  => a_to_g  <= "0000100";  --9
      when "1010"  => a_to_g  <= "0001000";  --A
      when "1011"  => a_to_g  <= "1100000";  --b
      when "1100"  => a_to_g  <= "0110001";  --C
      when "1101"  => a_to_g  <= "1000100";  --d
      when "1110"  => a_to_g  <= "0110000";  --E
      when others  => a_to_g  <= "0111000";  --F
    end case;
  end process;
end behavior;
## Combinatorial Logic

### Comparator

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A &lt; B</th>
<th>A &gt; B</th>
<th>A = B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram of a comparator:

```
  COMP
     0  P  P<Q
     2  P=Q
     0  Q  P>Q
     2
```
Combinatorial Logic

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &lt; B</th>
<th>A &gt; B</th>
<th>A = B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
(A < B) = \overline{A}B \\
(A > B) = A\overline{B} \\
(A = B) = \overline{A}B + AB = A \oplus B
\]
Combinatorial Logic

2-bit comparator

<table>
<thead>
<tr>
<th>$A_1, B_1$</th>
<th>$A_0, B_0$</th>
<th>$A &gt; B$</th>
<th>$A &lt; B$</th>
<th>$A = B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 &gt; B_1$</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_1 &lt; B_1$</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_1 = B_1$</td>
<td>$A_0 &gt; B_0$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_1 = B_1$</td>
<td>$A_0 &lt; B_0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_1 = B_1$</td>
<td>$A_0 = B_0$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

4-bit comparator

<table>
<thead>
<tr>
<th>$A_3, B_3$</th>
<th>$A_2, B_2$</th>
<th>$A_1, B_1$</th>
<th>$A_0, B_0$</th>
<th>$A &gt; B$</th>
<th>$A &lt; B$</th>
<th>$A = B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_3 &gt; B_3$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 &lt; B_3$</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 &gt; B_2$</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 &lt; B_2$</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 = B_2$</td>
<td>$A_1 &gt; B_1$</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 = B_2$</td>
<td>$A_1 = B_1$</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 = B_2$</td>
<td>$A_1 = B_1$</td>
<td>$A_0 &gt; B_0$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 = B_2$</td>
<td>$A_1 = B_1$</td>
<td>$A_0 &lt; B_0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$A_3 = B_3$</td>
<td>$A_2 = B_2$</td>
<td>$A_1 = B_1$</td>
<td>$A_0 = B_0$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Lab 11

Make a comparator
  • Simulate
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp is
  generic (N:integer := 8);
  port(
    x : in STD_LOGIC_VECTOR(N-1 downto 0);
    y : in STD_LOGIC_VECTOR(N-1 downto 0);
    gt : out STD_LOGIC;
    eq : out STD_LOGIC;
    lt : out STD_LOGIC
  );
end comp;
Lab 11 - solution

architecture behavior of comp is
begin
    process(x, y)
    begin
        gt <= '0';
        eq <= '0';
        lt <= '0';
        if (x > y) then
            gt <= '1';
        elseif (x = y) then
            eq <= '1';
        elseif (x < y) then
            lt <= '1';
        end if;
    end process;
end behavior;
Sequential Logic

Sequential vs Combinatorial

New state determined by:
• A number of independent inputs

New state determined by:
• A number of independent inputs
• The current state of the system (through a memory element)
Sequential Logic

Sequential vs Combinatorial

• In a combinatorial system: **all inputs** in the sensitivity list of a process

• In a sequential system: **only clock and asynchronous inputs** in the sensitivity list of a process
FLIPFLOP

- FLIPFLOP = one bit memory = base element of sequential circuits
- Amount of FF’s wordt determined by amount of states
  - One lamp on / of: one FF
  - Four bit binary counter: 4 FFs.
SR-FLIPFLOP with NAND

Active low inputs!
**SR-FLIPFLOP with NAND**

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>QT</th>
<th>QNT</th>
<th>/QNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Forbidden state

Set

Reset

Forbidden state

**Memory**

**Set**

**Reset**

**Forbidden state**
SR-FLIPFLOP with NAND

- Excitation table
  - Which inputs is needed to go from one state to the other?
  - For design!

<table>
<thead>
<tr>
<th>QT</th>
<th>QNT</th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>
JK-FLIPFLOP with asynchronous set and reset
<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>J</th>
<th>K</th>
<th>Q_{NT}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q_T Geheugen</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>/Q_T Synchrone toggle</td>
</tr>
</tbody>
</table>

JK-FLIPFLOP with asynchronous set and reset
### JK-FLIPFLOP

- **Excitation table**

<table>
<thead>
<tr>
<th>$Q_T$</th>
<th>$Q_{NT}$</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>
D-FLIPFLOP

On rising edge clock

<table>
<thead>
<tr>
<th>CLK</th>
<th>D</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D-FLIPFLOP

On rising edge clock
**D-FLIPFLOP**

**Exitation table**

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>D</th>
<th>QNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>? Verboden toestand</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>1 Asynchrone set</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>x</td>
<td>0 Asynchrone reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0 Synchrone reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 Synchrone set</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qt</th>
<th>QNT</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

D = 0: => J = 0 and K = 1

D = 1: => J = 1 and K = 0
Lab 12

Make a edge-triggered D-flipflop
  • Simulate: use clock process

```
-- Clock period definitions
constant clk_period : time := 20 ns;

-- Clock process definitions
clk_process : process
begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
end process;
```
entity Dff is
  port(
    clk : in STD_LOGIC;
    clr : in STD_LOGIC;
    D : in STD_LOGIC;
    q : out STD_LOGIC
  );
end Dff;

architecture behavior of Dff is
begin
  process(clk, clr)
  begin
    if(clr = '1') then
      q <= '0';
    elsif(rising_edge(clk)) then
      q <= D;
    end if;
  end process;
end behavior;
2-bit synchronous counter

CLK

+5V

Q0

01

0

+5V

Q1

00

0

D0

1

S

1D

0

1

C1

ID

R

D1

0

0

1
2-bit synchronous counter

![Diagram of 2-bit synchronous counter]

<table>
<thead>
<tr>
<th>CLK</th>
<th>Q0</th>
<th>Q1</th>
<th>D0</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2-bit synchronous counter

![2-bit synchronous counter diagram](image)

**Table: Clock and Outputs**

<table>
<thead>
<tr>
<th>Clock</th>
<th>Q0</th>
<th>Q1</th>
<th>D0</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lab 13

Make a Divide-by 2 Counter

• Simulate
• Add a clock divider to slow down the clock
entity div2cnt is
  port(
    clk : in STD_LOGIC;
    clr : in STD_LOGIC;
    q0 : out STD_LOGIC
  );
end div2cnt;
architecture behavior of div2cnt is
signal D, q: STD_LOGIC;
begin

D <= not q;

-- D Flip-flop
process(clk, clr)
begin
    if(clr = '1') then
        q <= '0';
    elsif (clk'event and clk = '1') then
        q <= D;
    end if;
end process;

q0 <= q;
end behavior;
4-bit shift register

<table>
<thead>
<tr>
<th>Clear</th>
<th>act. flank</th>
<th>Data in</th>
<th>Q_{FF1}</th>
<th>Q_{FF2}</th>
<th>Q_{FF3}</th>
<th>Data out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1e</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2e</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4e</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5e</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6e</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>7e</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Lab 14

Make a 4-bit shift register with ‘clr’
  • Simulate
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ShiftReg is
    port(
        clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        data_in : in STD_LOGIC;
        q : out STD_LOGIC_VECTOR(3 downto 0)
    );
end ShiftReg;
architecture behavior of ShiftReg is
signal qs: STD_LOGIC_VECTOR(3 downto 0);
begin

-- 4-bit shift register
process(clk, clr)
begin
if clr = '1' then
    qs <= "0000";
elsif clk'event and clk = '1' then
    qs(3) <= data_in;
    qs(2 downto 0) <= qs(3 downto 1);
end if;

end process;
q <= qs;
end behavior;
Ring counter
## Ring counter

<table>
<thead>
<tr>
<th>Clock</th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
<th>O₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Lab 15

Make a ring counter
  • Simulate
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ring4 is
    port(
        clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        q : out STD_LOGIC_VECTOR(3 downto 0)
    );
end ring4;
architecture behavior of ring4 is
signal qs: STD_LOGIC_VECTOR(3 downto 0);
begin
  -- 4-bit ring counter
  process(clk, clr)
  begin
    if clr = '1' then
      qs <= "0001";
    elsif clk'event and clk = '1' then
      qs(3) <= qs(0);
      qs(2 downto 0) <= qs(3 downto 1);
    end if;
  end process;
  q <= qs;
end behavior;
4-bit register

Parallel data in

Clock pulses

Parallel data out
Lab 16

Make a 4-bit register with ‘clr’ and ‘load’
  • Simulate
entity reg4bit is
  port(
    load : in STD_LOGIC;
    inp0 : in STD_LOGIC_VECTOR(3 downto 0);
    clk : in STD_LOGIC;
    clr : in STD_LOGIC;
    q0 : out STD_LOGIC_VECTOR(3 downto 0)
  );
end reg4bit;
architecture behavior of reg4bit is begin
-- 4-bit register with load
process(clk, clr)
begin
  if clr = '1' then
    q0 <= "0000";
  elsif clk'event and clk = '1' then
    if load = '1' then
      q0 <= inp0;
    end if;
  end if;
end process;
end behavior;
Lab 16: N-bit register

entity reg is
  generic(N: integer := 8);
  port(
    load: in STD_LOGIC;
    clk: in STD_LOGIC;
    clr: in STD_LOGIC;
    d: in STD_LOGIC_VECTOR(N-1 downto 0);
    q: out STD_LOGIC_VECTOR(N-1 downto 0)
  );
end reg;
Synchronous Counter

- Counts number of clock pulses
- The value appears on the outputs
- All outputs change on the same clock edge
Synchronous Counter

Synchronous BCD counter with T-FF’s: (J&K connected)
Lab 17

Make a 10-teller with ‘en’, ‘rst’ & ‘co’ (carry out)
- Simulate
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity count10 is
    Port ( clk : in STD_LOGIC;
          rst : in STD_LOGIC;
          en : in STD_LOGIC;
          co : out STD_LOGIC;
          count : out STD_LOGIC_VECTOR (3 downto 0));
end count10;
architecture Behavior of count10 is
  signal count_int: STD_LOGIC_VECTOR (3 downto 0);
begin
  process (clk, rst) begin
    if (rst = '1') then
      count_int <= "0000";
    elsif (rising_edge (clk) and en = '1') then
      count_int <= count_int + 1;
      if count_int = "1001" then
        count_int <= "0000";
      end if;
    end if;
  end process;
  count <= count_int;
  co <= count_int(3) and count_int(0);
end Behavior;
Finite State Machines

- Runs through a finite amount of states
- E.g. BCD-counter – Traffic light
- Next state is determined by the present state, the clock and possibly the value on the inputs
FSM: block diagram Moore
FSM: block diagram Mealy
FSM: block diagram

- State register
  - A certain amount of flip-flops (D or JK), triggered by the clock
  - Amount FFs is determined by the amount of states
FSM: block diagram

- Next state decoder
  - Next state is determined by the current state (fed back) and possibly an input
FSM: block diagram

- Output decoder
  - Output is determined by current state and possibly the inputs (Mealy)
  - By involving inputs in the output decoder, the system is not synchronous anymore (Mealy)
FSM: block diagram

- Next state and output decoder are pure combinatorial components
- Synchronisation is in state register
FSM: state diagram

(S0QT1QT0)

S0 → 00

S1 → 01

LED 1

S2 → 11

LED 2
FSM: state diagram

START  00

MODE

0 1

L1 01 L2 10

LED 1 LED 2
FSM: state diagram

De verandering van "MODE" wordt nu pas verwerkt!

<table>
<thead>
<tr>
<th>CLK</th>
<th>MODE</th>
<th>QT1</th>
<th>QT0</th>
<th>LED1</th>
<th>LED2</th>
</tr>
</thead>
</table>

| START | L1 | START | L1 | START | L2 | START | L2 | START |
FSM:

S0 00

LED 1

MODE 0 1

S1 01

LED 1

S2 10

LED 2
FSM: state diagram

[State diagram with CLK, MODE, QT1, QT0, LED1, LED2 signals and state transitions marked as S1, S0, S1, S0 (1), S2, S0 (2).]
FSM: state diagram

- Other representation:
  - Circles with name and state
  - Coding of the state
  - Arrows give the change of state
FSM: example

• Railway signalization
  o Two sensors, one before and one behind the railway crossing.
  o When the train arrives, the first sensor (SF) gives an impulse to activate the red light.
  o This state remains until the train passes the next sensor (SB) and activates the white light.
Lab 17

Make a railway crossing system
  • Simulate
Lab 17 - solution

entity Railway is
  Port ( clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        SF : in STD_LOGIC;
        SB : in STD_LOGIC;
        W : out STD_LOGIC;
        R1 : out STD_LOGIC;
        R2 : out STD_LOGIC);
end Railway;
architecture Behavioral of Railway is

type state_type is (Nothing, White, Red1, Red2);
signal present_state, next_state: state_type;

One type definition:
type <name_type> is (all possible states);

present_state and next_state can have the values:
Nothing, White, Red1, Red2
Possible statements:
• present_state <= Nothing;
• present_state <= next_state;
Lab 17 – state register

begin

sreg: process(clk, clr)
begin
  if clr = '1' then
    present_state <= Nothing;
  elsif clk'event and clk = '1' then
    present_state <= next_state;
  end if;
end process;

Lab 17 – next state decoding

ns_dec: process (present_state, SF, SB)
begin
    case present_state is
        when Nothing =>
            if SF = '0' then
                next_state <= White;
            else
                next_state <= Red1;
            end if;
        when White =>
            next_state <= Nothing;
        when Red1 =>
            next_state <= Red2;
        when Red2 =>
            if SB = '0' then
                next_state <= Red1;
            else
                next_state <= White;
            end if;
    end case;
end process;
Lab 17 – output decoding

```
out_dec: process (present_state)
begin
  case present_state is
    when Nothing =>
      W <= '0';
      R1 <= '0';
      R2 <= '0';
    when White =>
      W <= '1';
      R1 <= '0';
      R2 <= '0';
    when Red1 =>
      W <= '0';
      R1 <= '1';
      R2 <= '0';
    when Red2 =>
      W <= '0';
      R1 <= '0';
      R2 <= '1';
  end case;
end process;
```
Final lab

Make a stopwatch

- Use the following components
  - Debouncers for Start – Stop – Reset
  - SR Flipflop for Start – Stop
  - Two 10 Counters for 1’s and 10’s
  - A clock divider for slowing down the clock from 50 MHz to 1 Hz
Contact

Ing. Dirk Van Merode MSc.
Project Coordinator DESIRE
Thomas More | Campus De Nayer
Technology & Design
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 496 26 84 15
dirk.vanmerode@thomasmore.be
Skype dirkvanmerode
www.thomasmore.be

Dr. Ing. Peter Arras MSc.
International Relations Officer
KU Leuven | Campus De Nayer
Faculty of engineering technology
J. P. De Nayerlaan 5
2860 Sint-Katelijne-Waver

Belgium
Tel. + 32 15 31 69 44
Gsm + 32 486 52 81 96
peter.arras@kuleuven.be
Skype pfjlarras
www.iiw.kuleuven.be