

TDC_Lab_3. Designing sequential circuits

3.1. Flip-Flop D (FFD)

Testbenches with a clock signal

PROJECT_13. How to create a testbench to test a Flip-Flop D. Add a new simulation source to the project ("**FFD_BASIC_TB**") and generate the testbench template as it was explained in Lab_2:

<http://vhdl.lapino.net/testbench/tb.php>

https://www.doulos.com/knowhow/perl/testbench_creation/

Testbenches with a clock signal

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity FFD_BASIC_tb is
end;

architecture bench of FFD_BASIC_tb is

    component FFD_BASIC
        Port ( D_i : in STD_LOGIC;
              Q_o : out STD_LOGIC;
              CLK_i : in STD_LOGIC);
    end component;

    signal D_i: STD_LOGIC;
    signal Q_o: STD_LOGIC;
    signal CLK_i: STD_LOGIC;

    constant clock_period: time := 10 ns;
    signal stop_the_clock: boolean;
```

The frequency system clock is 100
MHz in Nexys4

Testbenches with a clock signal

begin

```
uut: FFD_BASIC port map ( D_i    => D_i,  
                           Q_o    => Q_o,  
                           CLK_i => CLK_i );
```

stimulus: **process**

begin

-- Put initialisation code here

-- Put test bench stimulus code here

stop_the_clock <= true;

wait;

end process;

Two process executing
simultaneously: clocking and
stimulus

clocking: **process**

begin

while not stop_the_clock **loop**

CLK_i <= '0', '1' **after** clock_period / 2;

wait for clock_period;

end loop;

wait;

end process;

A while-loop generates a periodical
clock signal

end;

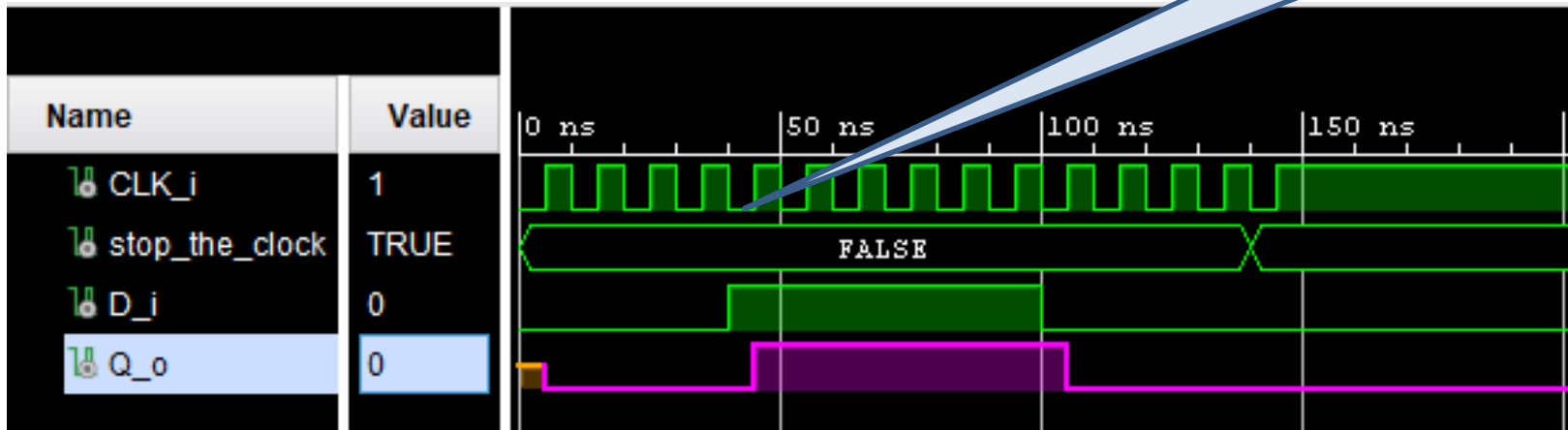
Testbenches with a clock signal

```
stimulus: process
begin
    D_i <= '0';
    wait for 40 ns;
    D_i <= '1';
    wait for 60 ns;
    D_i <= '0';
    wait for 40 ns;
    stop_the_clock <= true;
    wait;
end process;
```

The only input is assigned with a different value every 60 ns

The periodical clock keeps while new stimulus are provided.

The output is updated in the rising edge clock



Constraints file with a clock

Project_13. Add a new source to create a flip-flop D with ENABLE and RESET control lines.

Entity name: **FFD**
Inputs : D_i, RST_i, ENA_i, CLK_i
Output: Q_o

- Add a testbench to check the performance (**FFD_TB**)
- Add a constraint file to verify the design in NEXYS4

Constraints file with a clock

PROJECT_13. How to add a clock to the constraints file (XDC)

From de reference
manual

6 Oscillators/Clocks

The Nexys4 DDR board includes a single 100 MHz crystal oscillator connected to pin E3 (E3 is a MRCC input on bank 35). The input clock can drive MMCMs or PLLs to generate clocks of various frequencies and with known phase relationships that may be needed throughout a design. Some rules restrict which MMCMs and PLLs may be driven by the 100 MHz input clock. For a full description of these rules and of the capabilities of the Artix-7 clocking resources, refer to the “7 Series FPGAs Clocking Resources User Guide” available from Xilinx.

Xilinx offers the Clocking Wizard IP core to help users generate the different clocks required for a specific design. This wizard will properly instantiate the needed MMCMs and PLLs based on the desired frequencies and phase relationships specified by the user. The wizard will then output an easy-to-use wrapper component around these clocking resources that can be inserted into the user's design. The clocking wizard can be accessed from within the Project Navigator or Core Generator tools.

The master XDC includes a line declaring the CLOCK, erase the comment in this line in order to add this constraint.
(Remember erase the comment for the inputs and outputs needed.)

Constraints file with a clock

PROJECT_13. How to add a clock to the constraints file (XDC)

Master

```
1 ## This file is a general .xdc for the Nexys4 DDR Rev. C
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6 ## Clock signal
7 #set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8 #create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { CLK100MHZ }];
9
10
11 ##Switches
12
```

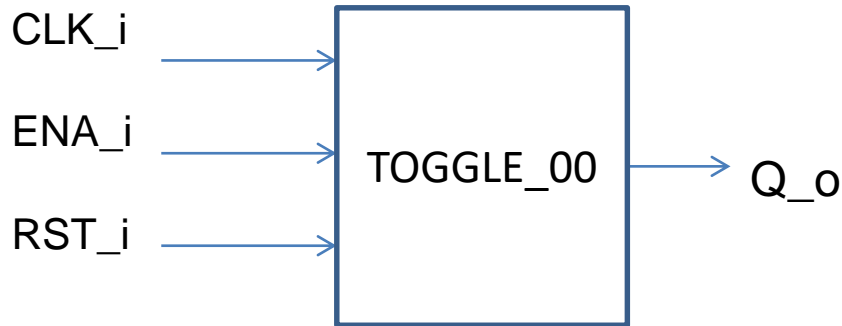
FFD_basic's XDC file

```
1 ## This file is a general .xdc for the Nexys4 DDR Rev. C
2 ## To use it in a project:
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK_i }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
8 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { CLK_i }];
9
10
11 ##Switches
12
```

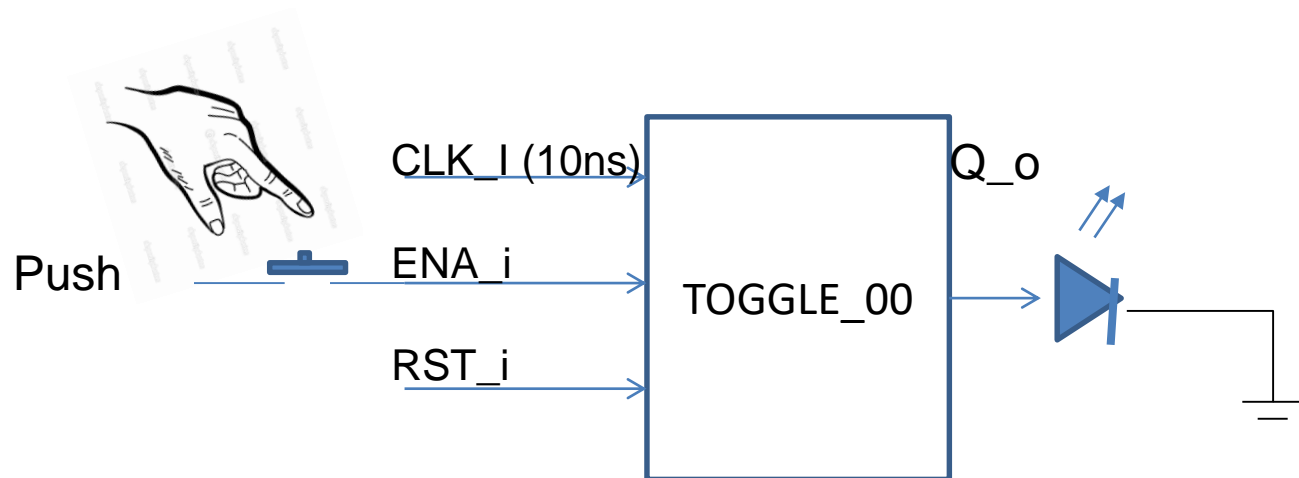
3.2. Toggling LED circuit

Toggling LED circuit

Project_14. Design a sequential circuit to toggle the output every time a pushbutton (“ENA_i”) is pushed. (Entity name: “**TOGGLE_00**”)



CLK	RESET	ENA	Q(t)	Q(t+1)
↑	0	1	0	1
↑	0	1	1	0
↑	1	X	X	0



Toggling LED circuit

Project_14. Design a sequential circuit which toggles the output value every time a pushbutton is pushed

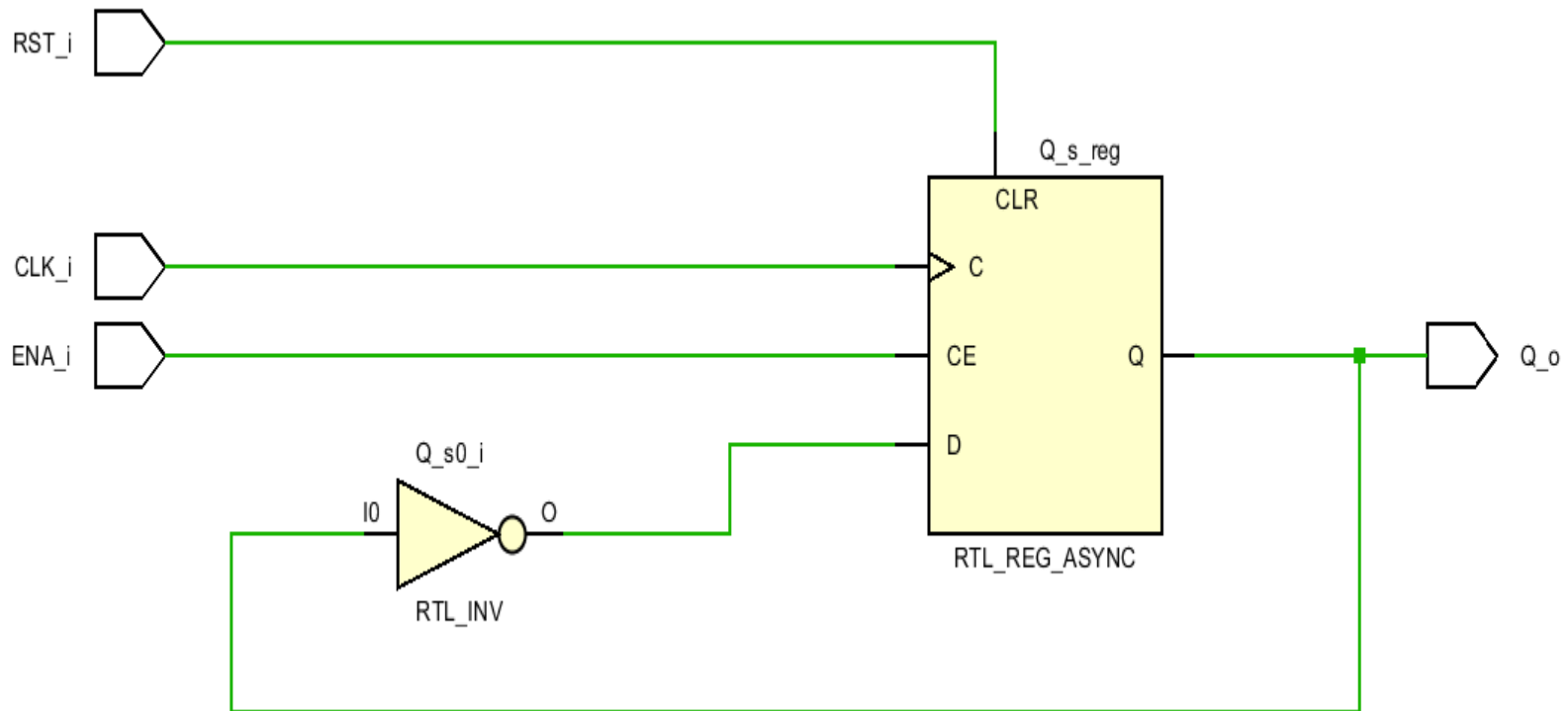
```
architecture Behavioral of TOGGLE_00 is
    signal Q: std_logic; -- Dummy signal to read Q
begin
    process (CLK_i,RST_i)
    begin
        if RST_i='1' then
            Q <= '0';
        elsif rising_edge(CLK_i) then
            if ENA_i='1' then
                Q <= not Q; -- Read the output "Q"
            end if;        -- If '0' keeps the previous value
        end if;
    end process;

    Q_o <= Q;

end Behavioral;
```

Toggling LED circuit

Project_14. Design a sequential circuit which toggles the output value every time a pushbutton is pushed



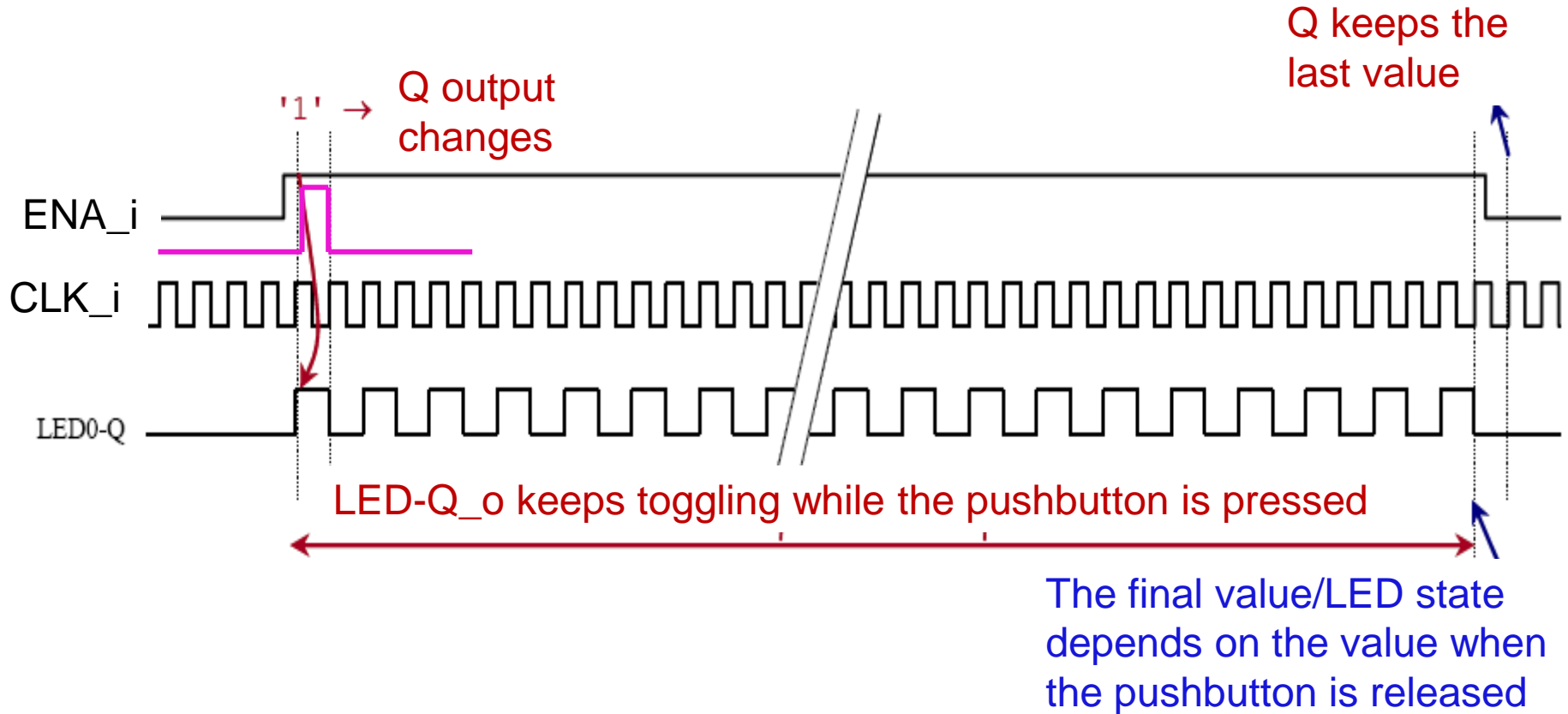
Toggling LED circuit

Project_14. Design a sequential circuit which toggles the output value every time a pushbutton is pushed

1. Run the circuit in Nexys4
 - What do you think about the performance of the circuit?
Is it right? What is happening?
2. Simulate the Toogle circuit
 - Look at the simulation waveform, could you determine why the circuit has a wrong behaviour?

Toggling LED circuit

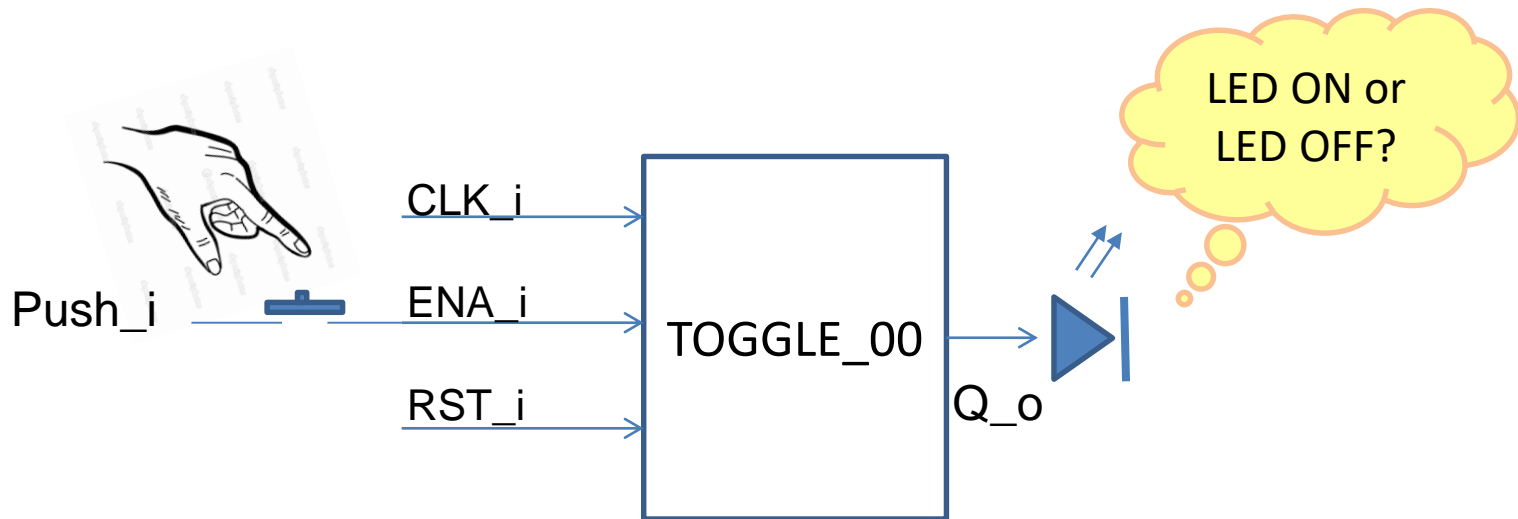
Project_14. Simulation



Toggling LED circuit

Watch the simulation waveform and notice how many periods take the human's action of pressing a pushbutton. → **More than one cycle and remember:**

...for every rising edge clock a new toggle has taken place



3.3. Toggle circuit with rising edge detection

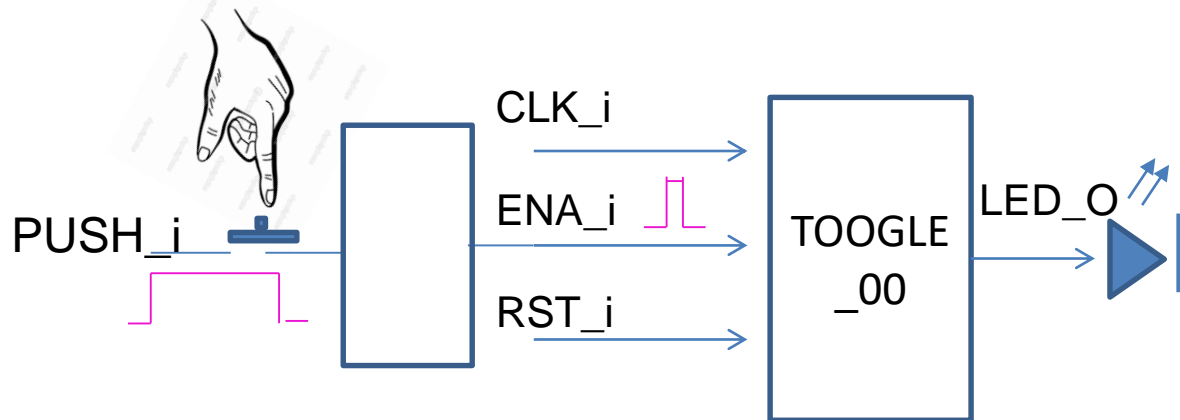
Toggle circuit with rising edge detection

Project_14. Modify the previous design to detect the rising edge in “PUSH_i” asserting the output for a clock cycle (pulse lasting 10 ns)

Entity name: **TOGGLE_01**

Inputs: **RST_i, PUSH_i, CLK_i**

Outputs: **LED_o**



PUSH_i (t)	PUSH_i (t+1)	LED_o (t +1)
0	0	LED_o (t)
0	1	$\overline{\text{LED_o (t)}}$
1	0	LED_o (t)
1	1	LED_o (t)

Positive edge

Toggle circuit with rising edge detection

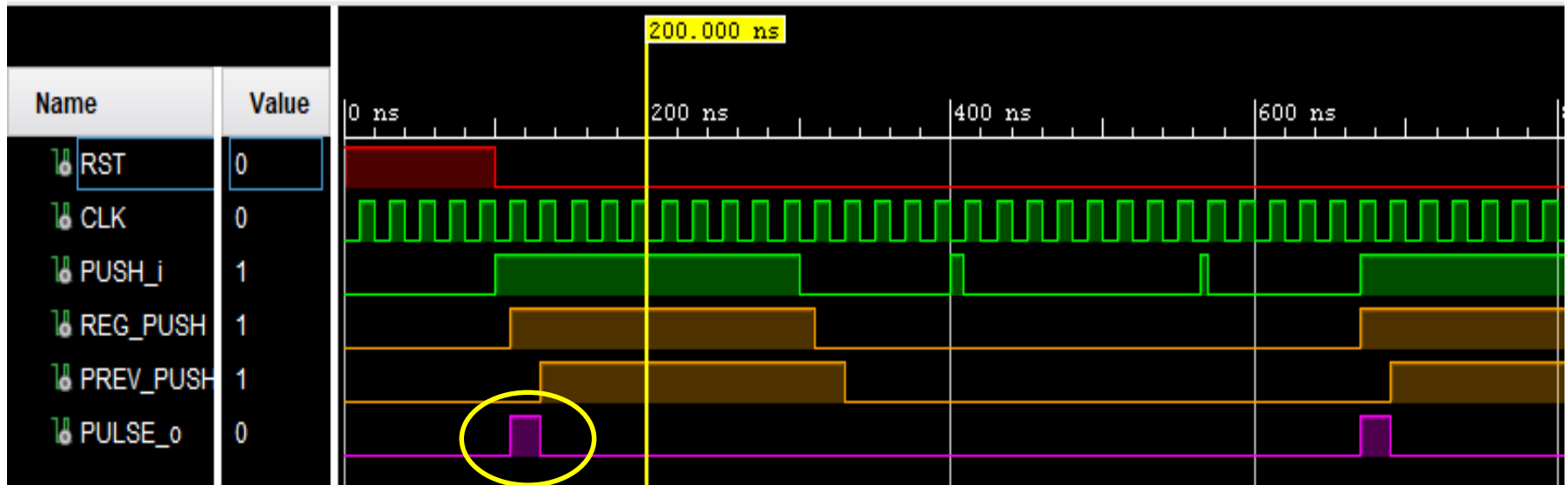
-- Process to synchronize the input PUSH_i, storing the
-- PREV_PUSH and toggle the output every time PUSH_i
-- is pressed.

```
process (CLK_i,RST_i)
begin
    if RST_i='1' then
        REG_PUSH <= '0';
        PREV_PUSH <= '0';
        Q <= '0';--
    elsif rising_edge(CLK_i) then
        REG_PUSH <= PUSH_i;
        PREV_PUSH <= REG_PUSH
        -- Check the rising edge condition in the PUSHBUTTON
        if (PREV_PUSH = '0' and REG_PUSH = '1') then
            Q <= not Q;
        end if;
    end if;
end process;
```

Toggle circuit with rising edge detection

- Add a testbench
- Watch the waveform and determine how many cycles is the output delayed . What is the reason for this delay? Add a constraints file to test on NEXYS4

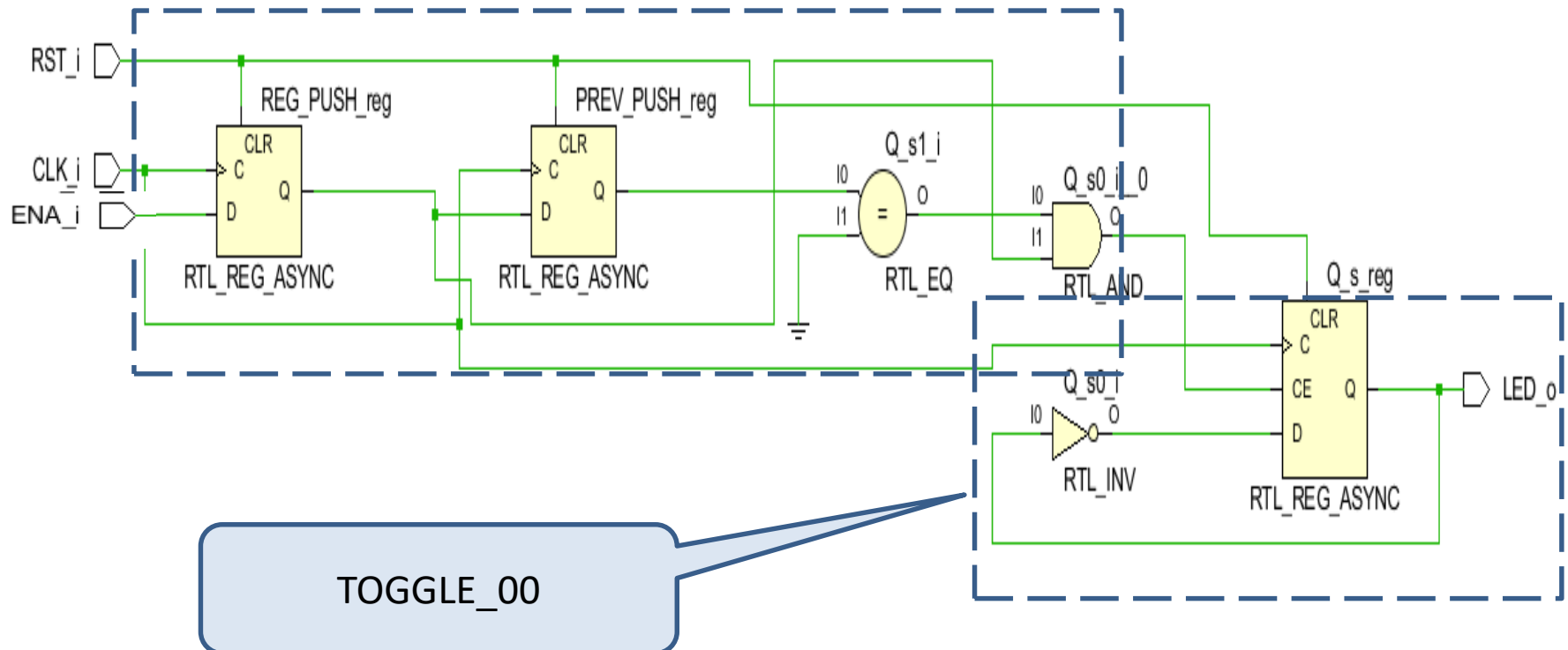
Toggle circuit with rising edge detection



One PULSE lasting one clock-cycle. It is the new ENABLE for the toggle

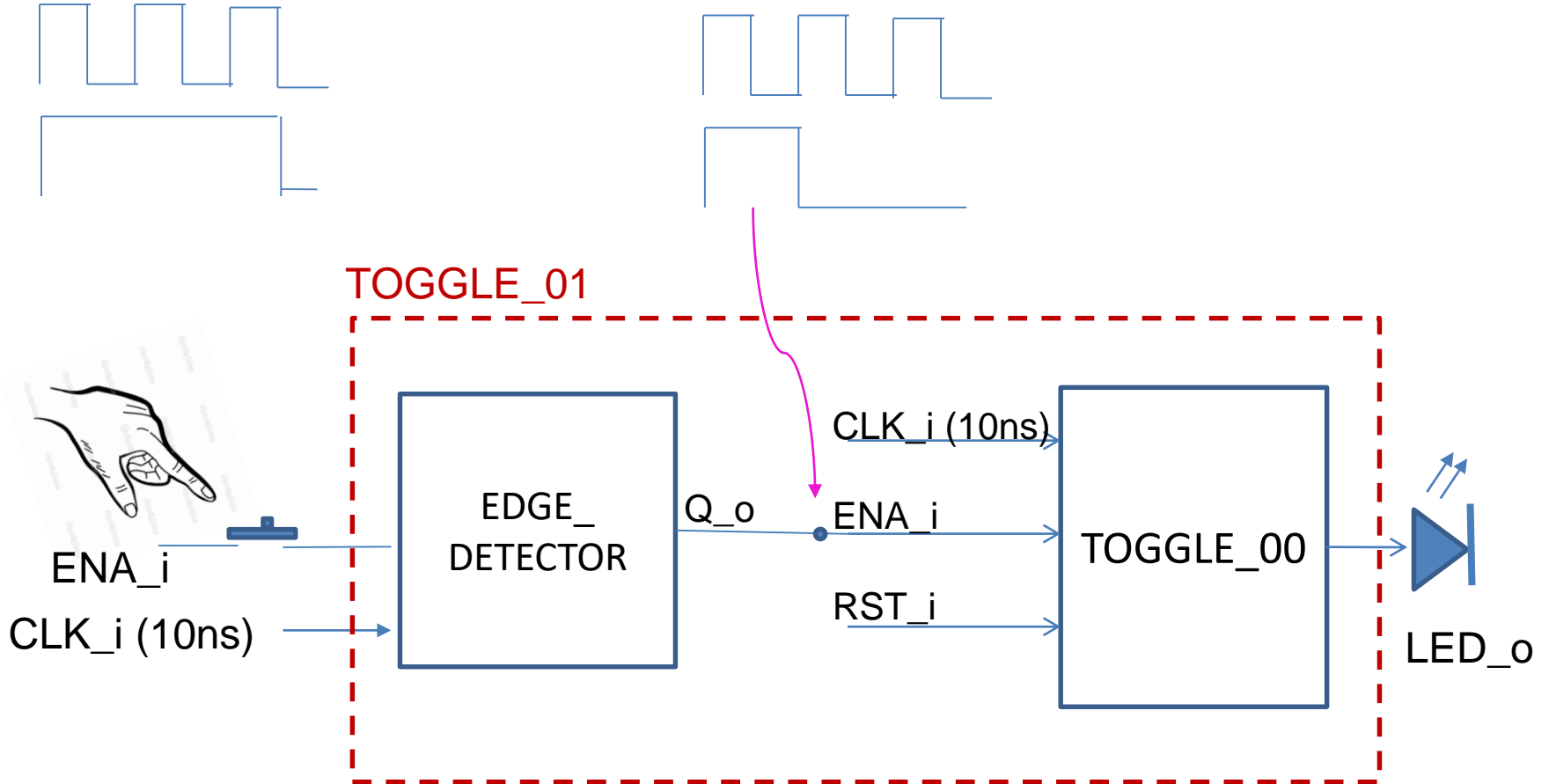
Toggle circuit with rising edge detection

EDGE_DETECTION



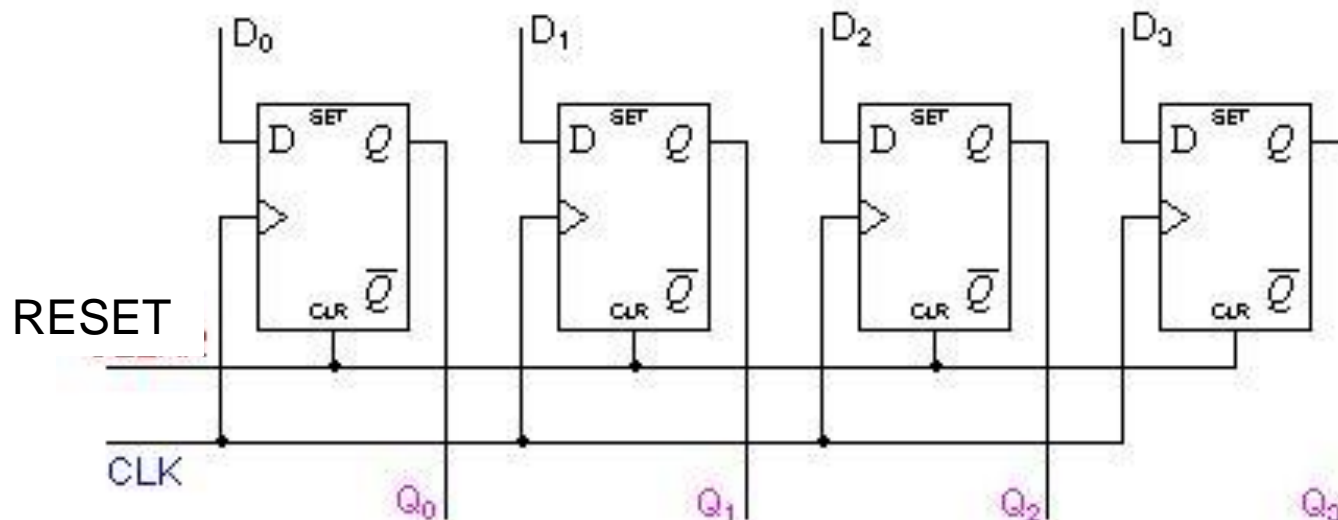
TOGGLE_00

Edge detector circuit and toggling LED



3.4. Increasing the width of the FFD: Registers

Registros



Project_16

- Basing upon the FFD VHDL description, design a **generic** register by using a behavioral description. **Not use a structural description describing four FFD in chain!!!**
- Verify it is correctly working setting the parameter to 5 bits width.
 - **Entity name:** REG_N_bits
 - **Inputs:** RST_i, CLK_i, ENA_i, DATA_IN_i
 - **Outputs:** DATA_OUT_o

3.5. Counters

Counters

- Create a new Project called **Project_17** and add a VHDL module describing a generic synchronous counter (**COUNTER_N_bits**)
- Add a testbench to check that it is working correctly.
- Add a constraints file and verify the working onto Nexys4
- Is the behaviour right on Nexys4? Why?

Counters

- Design a circuit which increments a counter every time a pushbutton is pressed. (Binary Count: 0 to 9 shown in 7-segment display)

Project's name: Project_18

Entity's name: YOUR_TURN

Inputs: CLK_i, RST_i, INC_i

Outputs: ANODE_o, CATHODE_o

- Write a testbench for checking the right performance

Reuse as many as designed modules as you need. It is recommended to add the “Edge_detector” designed in the Project_15 to filter the pushbutton.

3.6. Frequency divider (a.k.a. pre-scaler)

Counters

- Add to the **Project_19** a second file describing a generic prescaler (Application 1).
- Default value of the desired frequency 1 MHz.

Project's name: **Project_19**

Entity's name: **CE_N_Hz**

Inputs: CLK_i, RST_i

Outputs: CLK_N_Hz_o

Add a constant ("**CLK_100MHz**") for defining the number of cycles (Hz) of the system clock (100MHz). Then, add a generic ("**F_Hz**") to set the desired frequency in hertz. Later, define a new constant ("**PRESCALER_FACTOR**") with a value depending on the clock-system frequency and the desired one.

- Write a testbench for checking the right performance.

Pre-scaler: First Application (Clock Enable)

- According to the 2_3 lesson, design a pre-scaler (50% duty cycle) to provide a 1 second clock.

(Remember: clock system frequency 100 MHz)

Project's name: Project_20

Entity's name: PRESCALER_1Hz

Inputs: CLK_i, RST_i

Outputs: CLK_1Hz_o

- Checking by means of the simulator
- Modify the simulation options to achieve more than 1 second in the waveform (Next slide)
- Checking by using a LED as output in Nexys4

Pre-scaler: First Application (Clock Enable)

- Use the Doulos TB generator to create the testbench
https://www.doulos.com/knowhow/perl/testbench_creation/
- Change the elapsed time until “stop_the_clock <= true;”

```
49 stimulus: process
50 begin
51
52     -- Put initialisation code here
53
54     RST_i <= '1';
55     wait for 5 ns;
56     RST_i <= '0';
57     wait for 5 ns;
58
59     -- Put test bench stimulus code here
60     wait for 2000 ms;
61     stop_the_clock <= true;
62     wait;
63 end process;
64
65 clocking: process
66 begin
67     while not stop_the_clock loop
68         CLK_i <= '0', '1' after clock_period / 2;
69         wait for clock_period;
70     end loop;
71     wait;
72 end process;
73
```


Pre-scaler

- Modify the simulation options to achieve more than 1 second in the waveform

The screenshot shows the Vivado Project Manager interface for 'PROJECT_16'. The 'Flow Navigator' on the left has a blue arrow pointing to 'Run Simulation' under the 'SIMULATION' section. A yellow box with the text 'Right-click' is positioned next to this arrow. The 'Run Simulation' dialog box is open, showing the 'Simulation' tab. The 'xsim.simulate.runtime' option is highlighted with a red oval and set to '1.5 s'. The 'Simulation' section of the dialog includes fields for 'Target simulator' (Vivado Simulator), 'Simulator language' (VHDL), 'Simulation set' (sim_1), and 'Simulation top module name' (TOGGLE_LED_tb). The 'Tool Settings' section on the left lists various project settings, with 'Simulation' selected. The 'Simulation' tab in the dialog shows a table of simulation options, with 'xsim.simulate.runtime' set to '1.5 s'. The 'xsim.simulate.runtime' option is also listed in the 'Specify simulation run time' section at the bottom of the dialog.

Right-click

Simulation
Specify various settings associated to Simulation

Target simulator: Vivado Simulator
Simulator language: VHDL
Simulation set: sim_1
Simulation top module name: TOGGLE_LED_tb

Tool Settings

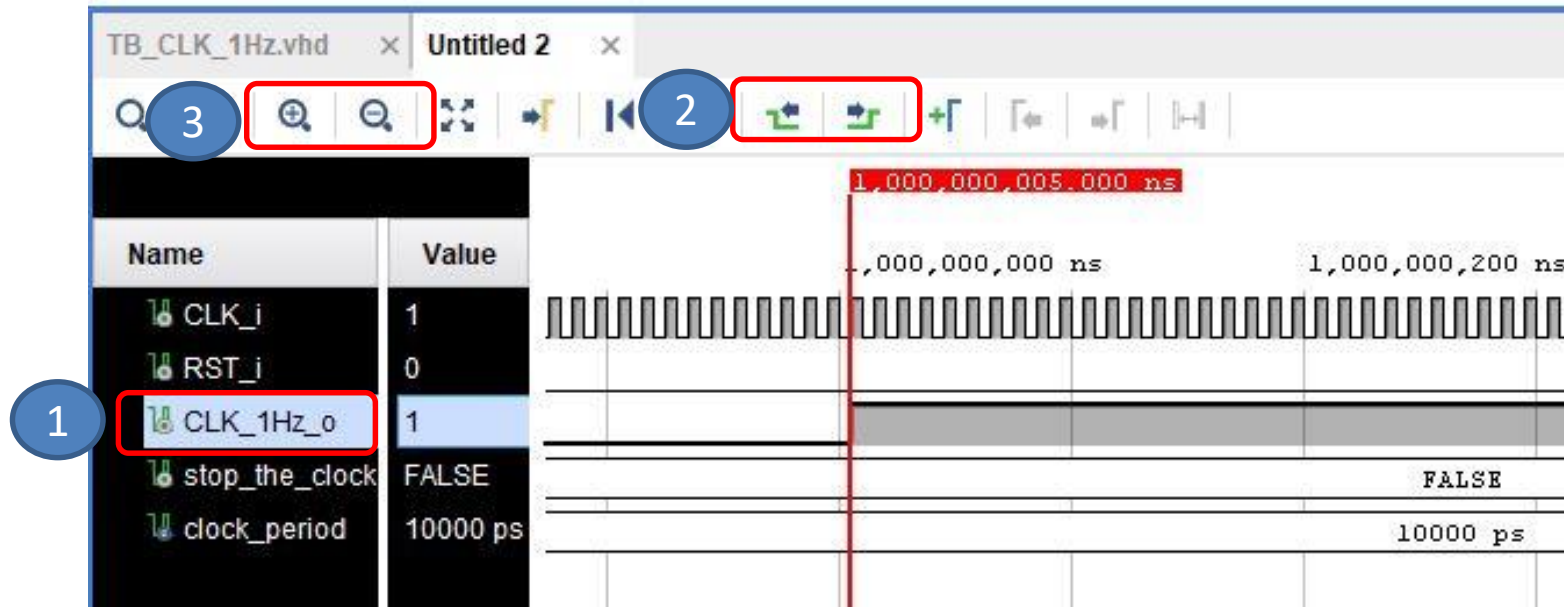
Compilation	Elaboration	Simulation	Netlist	Advanced
xsim.simulate.tcl_post				
xsim.simulate.runtime		1.5 s		
xsim.simulate.log_all_signals				<input type="checkbox"/>
xsim.simulate.custom_tcl				
xsim.simulate.wdb				
xsim.simulate.saif_scope				
xsim.simulate.saif				
xsim.simulate.saif_all_signals				<input type="checkbox"/>
xsim.simulate.xsim.more_options				

xsim.simulate.runtime
Specify simulation run time

OK Cancel Apply Restore...

Pre-scaler

- The simulation takes a long time
- When finish find the rising edge in output CLK_1Hz_o following these tips:
 - 1.- Select the output
 - 2.- Use the button to find the rising edge, the cursor will locate there
 - 3.- Use the zoom around the cursor position until the waveform looks like below



Symmetric Pre-scaler : Second application

- Design a circuit to show 8 different data onto the seven-segment displays.

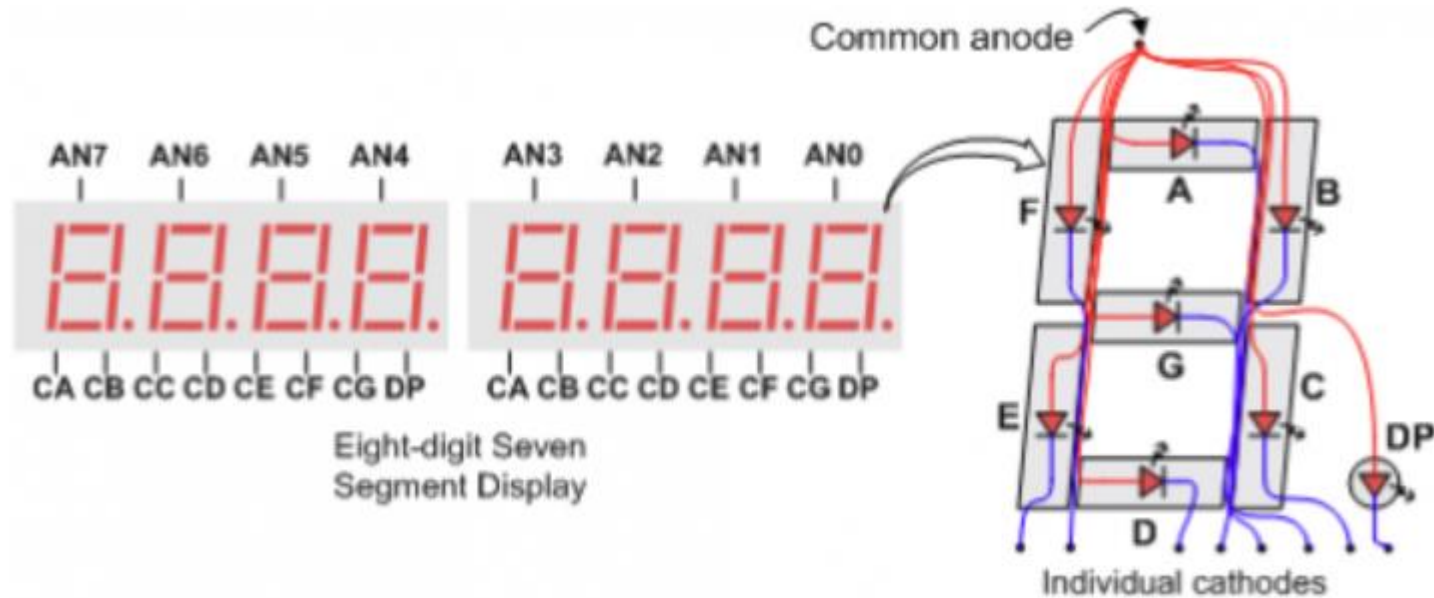
Project's name: Project_21

Entity's name: DISP7SEG_8ON

Inputs: CLK_i, RST_i, DATA0_i, ..., DATA7_i

Outputs: ANODE_o, CATHODE_o

Symmetric Pre-scaler : Second application (Slower CLK)



Every **cathode** (CA, CB...) and **anode** (AN7, AN6,) is controlled by a FPGA pin

Symmetric Pre-scaler : Second application (Slower CLK)

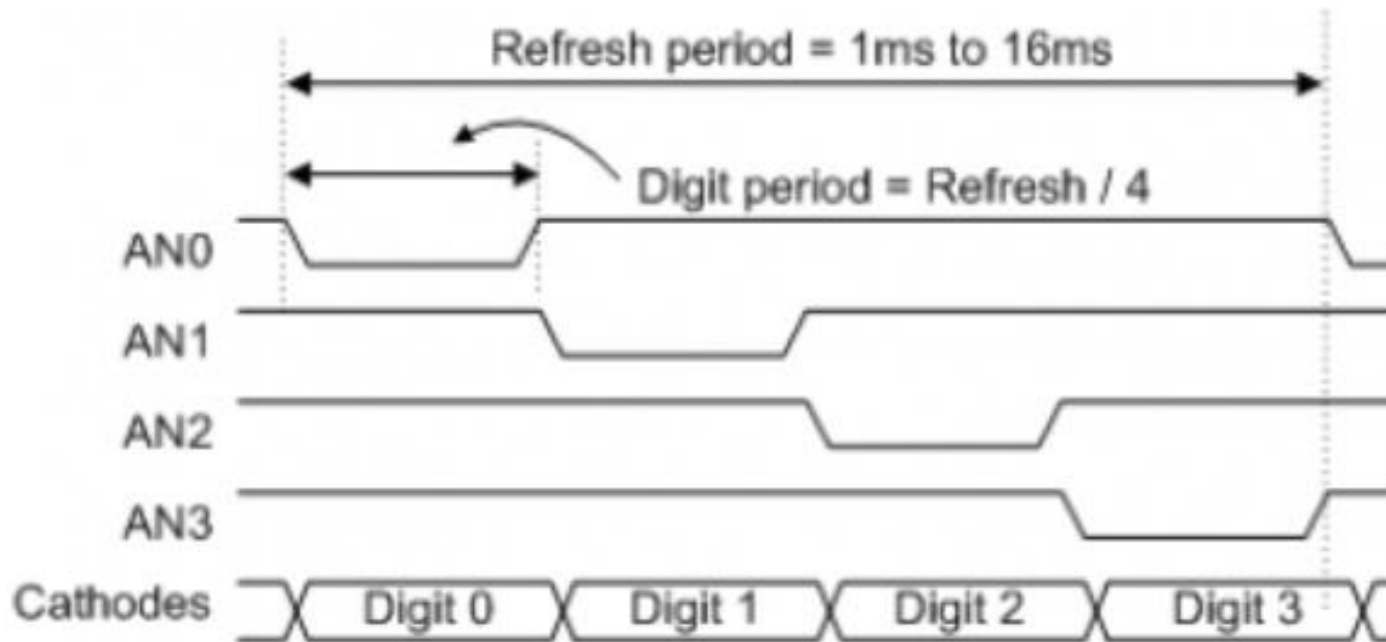


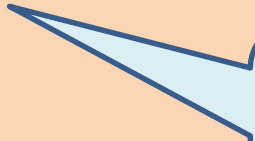
Figure 19. Four digit scanning display controller timing diagram.

Symmetric Pre-scaler : Second application

Project_21

To take into account:

- The eight 7-segment displays are connected in a multiplexed way, where the cathode signals are common to all digits but they can only illuminate the segments of the digit whose corresponding anode signal is asserted.



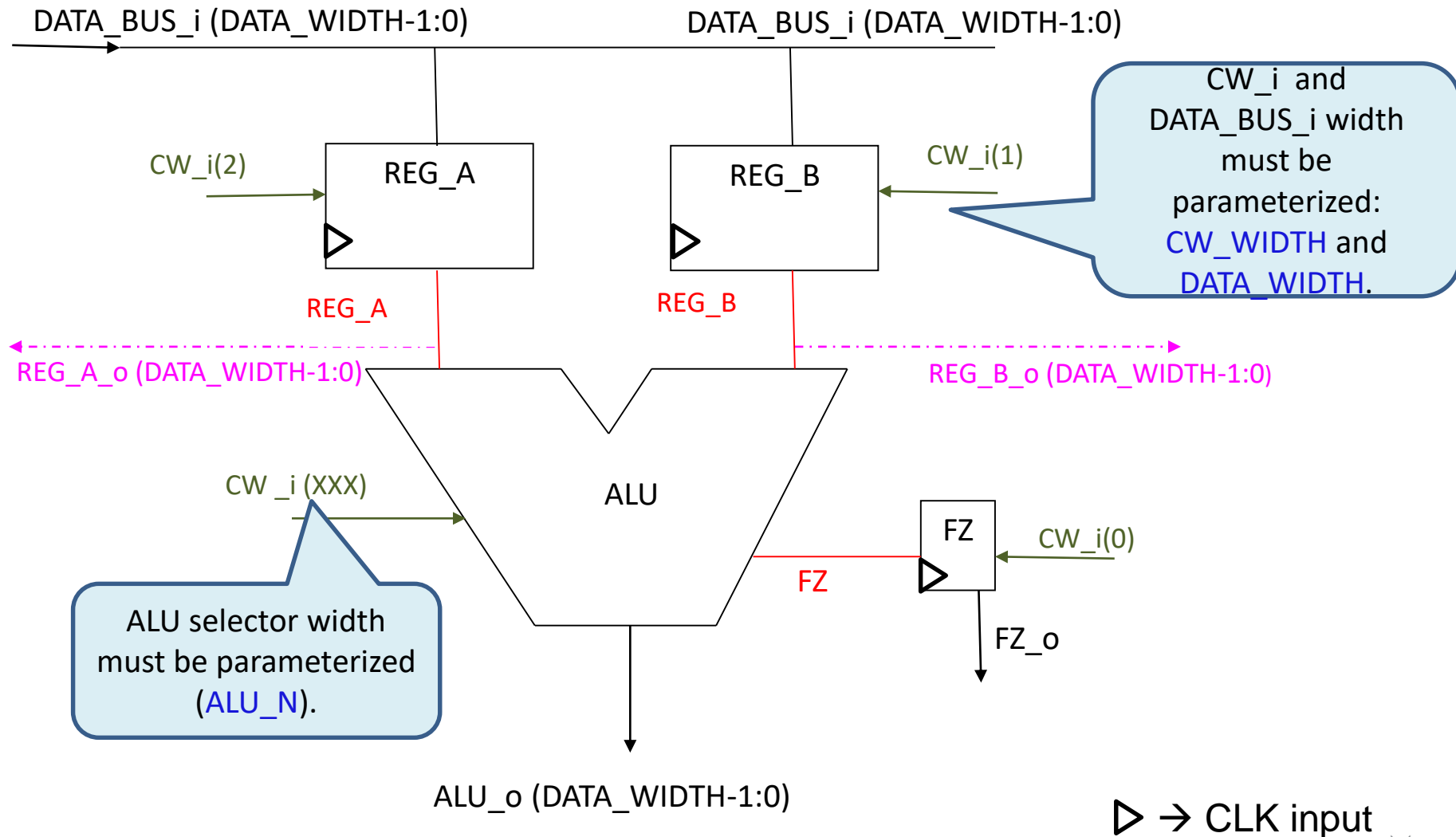
This circuit takes advantage of the human's eyes error, incapable to see fast movements. Actually the first data is shown onto the first display for 1 ms, then the second data onto the second display, and so on...

- Use a **structural** style
- **Reuse** verified modules, custom **libraries** and packages if desired.
- A 1KHz **pre-scaler** will provide a control line (**ENABLE**) to a **counter** in order to select sequentially each display showing a different data.
- It is recommended to start the exercise **drawing a block diagram** to depict the components and the signals.

3.7. Datapath

Datapath

Project_22. Designing a generic width-data datapath



Datapath

- Project name: **P22_DATAPATH_0_RA_RB_ALU**

- File name: **DATAPATH_0**

- Entity name: **DATAPATH_0**

- Inputs:

* DATA_BUS_i (DATA_WIDTH bits)

* RST_i

* CLK_i

* CW_i (CW_WIDTH bits)

-Outputs:

* ALU_o (DATA_WIDTH bits)

* REG_A_o (DATA_WIDTH bits)

* REG_B_o (DATA_WIDTH bits)

* FZ_o

OP(1)	OP(0)	ALU Functions
0	0	MOV A
0	1	INC A
1	0	A + B
1	1	A - B

Guides and Tips:

- ✓ Use only a VHDL file to describe the DATAPATH circuit. (**Not structural**)
- ✓ If prefer, use several processes, and keep in mind that every process is like a component in a structural approach. The processes (components) are connected each other by means of signals.
- ✓ Set 4 bits as default value for the generic (**DATA_WIDTH**).
- ✓ Describe **the Control Word** input (**CW_i**) as a parameterized array.
- ✓ Set 5 bits as width CW (**CW_WIDTH**) for testing purposes. To select the ALU functions use the MSB in CW array.
- ✓ Add a generic (**N_ALU**) to set the the size of the ALU selection input.

Camino de datos

1.- Add a testbench (**TB_Datapath_0.vhd**) to verify the circuit is working properly by checking all the operations provided by the ALU. The TB has to take the following steps :

- Clock-cycle 1. Set a value in DATA_BUS (Operand A) and load it in **REG_A** → Enable REG_A with **CW_i(2)**
- Clock-cycle 2. Set a new value in DATA_BUS (Operand B), if needed, and load it in **REG_B** → Enable REG_B with **CW_i(1)**
- Clock-cycle 3. Select an operation in ALU and update **FZ** → Select ALU operation (**CW_i(4:3)**) and enable FZ with **CW_i(0)**

2.- In the TB, each enable signal to update the registers must last only one clock-cycle (**10ns**). In order to achieve it, write the complete value for **CW_i** in each cycle-clock.

3.- The following table shows the value for **CW_i** for each micro-instruction.

Camino de datos

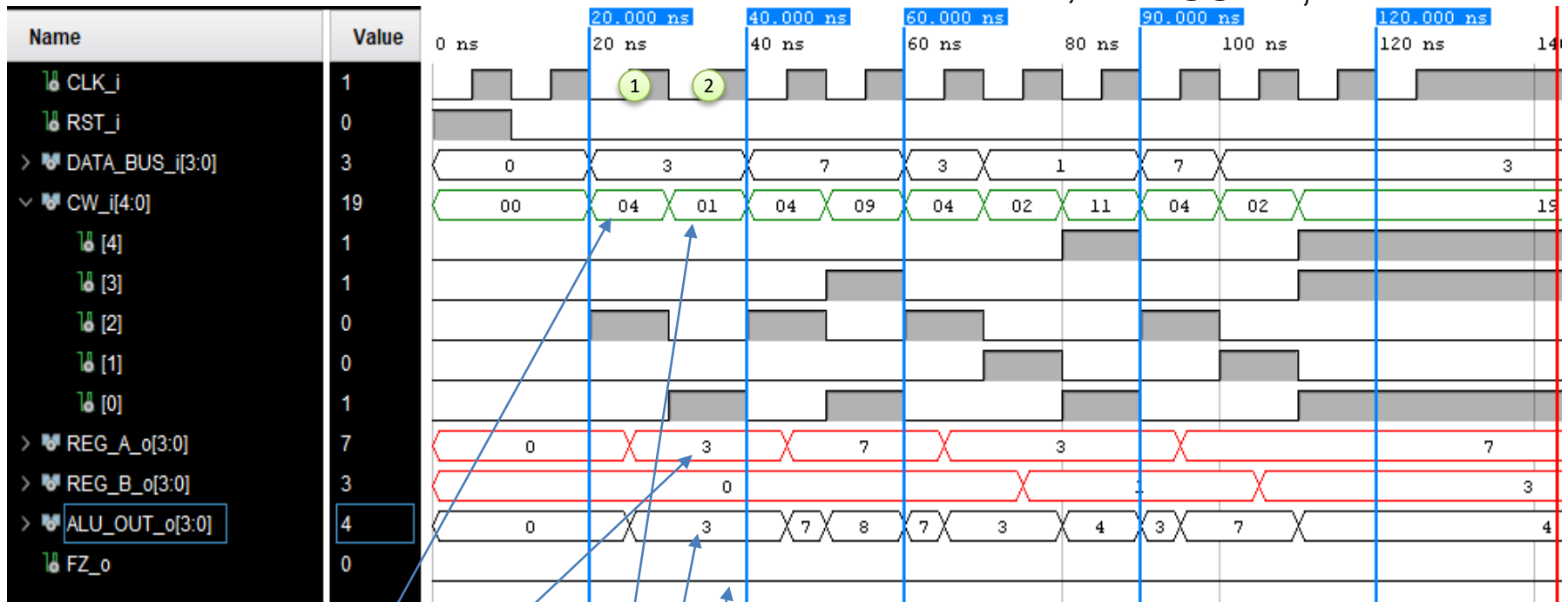
		ALU_OP	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(2)	CW(1)	CW(0)
1	<i>Load OPE_A</i>	00	1	0	0
2	<i>MOV A operation and FZ</i>	00	0	0	1

		ALU_OP	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(2)	CW(1)	CW(0)
1	<i>Load OPE_A</i>	00	1	0	0
2	<i>INC A operation and FZ</i>	01	0	0	1

		ALU_OP	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(2)	CW(1)	CW(0)
1	<i>Load OPE_A</i>	00	1	0	0
2	<i>Load OPE_B</i>	00	0	1	0
3	<i>A + B operation and FZ</i>	10	0	0	1

	Micro-instructions	ALU_OP	REG_A	REG_B	FZ
CLK	Micro-instructions	CW (4:3)	CW(2)	CW(1)	CW(0)
1	<i>Load OPE_A</i>	00	1	0	0
2	<i>Load OPE_B</i>	00	0	1	0
3	<i>A - B operation and FZ</i>	11	0	0	1

Camino de datos



1 Load "3" in REG_A

2 Select MOV_A and Update FZ

References

[1] Introducing the Spartan3E FPGA and VHDL (Ch.11)

<http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Dev/FPGA/IntroToSpartanFPGABook.pdf>

[2] Free Range VHDL (Ch.5)

http://www.freerangefactory.org/dl/free_range_vhdl.pdf

[3] Diseño de circuitos digitales con VHDL (Ch.5)

<http://eciencia.urjc.es/handle/10115/4045>

[4] <https://surf-vhdl.com/vhdl-syntax-web-course-surf-vhdl/vhdl-syntax-coding-style/>