ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment V
SEQUENTIAL DESIGN

2024

In this experiment, students will realize sequential circuits and learn their properties. Behavioral designs for clock dividers will be done in HDL code. Designs will be verified using testbenches and real-time FPGA executions.

Objectives

- Design sequential circuits with Verilog
- Using testbenches to perform proper simulations

Requirements

Students are expected to be able to

- The basic concepts of sequential circuits
- Structural and Behavioral Design Methods with Verilog
- Usage of Vivado: synthesize, implementation, design reports

Experiment Report Checklist

1. **D Flip-Flop**
   - Logic Diagram, truth table, characteristic function and behavior of SR NAND Latch.
   - Logic Diagram, truth table, characteristic function and behavior of Gated D Latch.
   - Logic Diagram, truth table, characteristic function and behavior of Edge-Triggered D Flip-Flops.
   - Verilog code and Simulation results of DFF with synchronous reset.
   - RTL Schematics, Technology Schematics and the usage of primitives.
   - Verilog code and Simulation results of DFF with active-low asynchronous reset.
   - RTL Schematics, Technology Schematics and the usage of primitives.
   - Comparison of both type DFFs in terms of usage of primitives.
2. **8-bit Shift Register**
   - Verilog code and Simulation results.
   - RTL and Technology Schematics
   - Utilization report, usage of primitives.
3. **Clock Divider with Stopwatch Example**
   - Verilog code and Simulation results of the Clock Divider.
   - Verilog code and Simulation results of the Stopwatch Example.
4. **Sliding LEDs**
   - Verilog code and Simulation results.
   - Post-Implementation Timing Simulation result.
   - Timing report and critical path delay.
5. **Research**
   - Research about the **Static Timing Analysis** in digital design.
   - Research about the **Metastability** concept in digital design.

- **Projects and reports are to be done INDIVIDUALLY. High amounts of points will be deducted from similar works.**
- **Reports must be written in a proper manner. Divide your text to sections and subsections if needed, label your figures and connect your sections with proper explanations of your works. Reports filled with imprecisely placed tables and figures, with no verbal explanations in workflow, will not fare well.**
- **Check homeworks section in Ninova for submission dates.**

D Flip-Flop

1. Add the logic diagram and truth table of the **SR NAND Latch** to your report. Derive the characteristic function of **SR NAND Latch** and explain its behavior. Which operation of the SR NAND Latch is not allowed (forbidden state)?

2. Add the logic diagram and truth table of the **Gated D Latch** to your report. Derive the characteristic function of **Gated D Latch** considering the "Enable" input.

3. Do research about the **Edge-Triggered D Flip-Flops** (such as Master-Slave DFF), add their logic diagram to your report and explain their behavior. What are the concepts of "edge-sensitivity" and "level sensitivity"? (Ref: S. Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed.)

4. Add a new source to your project named **registers.v** .

5. Write down a module which is called **DFF_sync**. This module should have 1-bit inputs **clk**, **rst**, **D** and 1-bit output **Q**. Behavior of this module should be **DFF with synchronous reset**. Obtain this behavior using an **always** block. This module will be triggered when the **positive-edge** of **clk** signal. Assign **logic-0** to output if **rst** is high; otherwise, assign **D** input to output **Q**.

6. Create a **testbench** to simulate your design.

7. **Synthesize** and **Implement** your design. Obtain the utilization report and show which primitives are used.

8. Write down another module which is called **DFF_async**. This module should also have 1-bit inputs **clk**, **rst**, **D** and 1-bit output **Q**. Behavior of this module should be **DFF with active-low asynchronous reset**. Obtain this behavior using an **always** block. This module will be triggered when the **negative-edge** of **rst** signal or the **positive-edge** of **clk** signal. Assign **logic-0** to output if **rst** is high; otherwise, assign **D** input to output **Q** .

9. Create a **testbench** to simulate your design.

10. **Synthesize** and **Implement** your design.

11. View RTL and Technology Schematics.

12. Obtain the utilization report and show which primitives are used. Compare the primitives used for both flip-flops, are they different?

8-bit Shift Register

1. Create a new project and write down a module which is called **shift8**. This module should have 1-bit inputs **clk**, **rst**, **D** and 8-bit output **Q**. Behavior of this module should be **serial-in parallel-out 8-bit shift register**. Obtain this behavior using an **always** block which will be triggered on the **positive-edge** of **clk** signal. In each clock cycle, the output Q will be left-shifted. Assign **logic-0** to the all outputs if **rst** is high; otherwise, assign **D** input to output **Q[0]** and shift the other **Q** outputs to the left (**Hint:** Use concatenation operator).

2. Create a **testbench** to simulate your design.

3. Make your pin configurations. Assign **clk** and **rst** inputs to the buttons, **D** input to a switch and **Q** outputs to the LEDs.

4. **Synthesize** and **Implement** your design. If you get an error during the implementation, use **set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets <netName>];** constraint.

5. View RTL and Technology Schematics and obtain utilization summary. Show which primitives are used in the circuit.

6. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

Clock Divider with Stopwatch Example

1. Nexys A7 board includes a 100 MHz crystal oscillator connected to pin E3 which can be used to generate various frequencies of clocks. Vivado supports the "Clocking Wizard IP core" which drives MMCMs or PLLs to generate clocks. Other method is to design a clock divider that can achieve to obtain rates of 100 MHz reference clock by using the counters.

2. In order to use the system clock (100MHz), the "Clock signal" section in the master .xdc file should be commented out. Do not forget to rename the port.

3. Create a new project and write down a module which is called **clk_divider**. This module should have 1-bit input **clk_in**, 1-bit input **rst**, and 1-bit output **clk_out**; besides a 28-bit parameter **CLK_DIV** with the deafult value 100. This module will count to the desired rate **CLK_DIV** on each positive-edge of the **clk_in** signal and output the **clk_out**.

4. Define a 28-bit counter and increment it inside an **always** block on the positive-edge of **clk_in** signal. When the counter reaches the desired rate **CLK_DIV**, invert the **clk_out** signal to obtain newly generated clock. Consider the initial value of **clk_out** signal and the 28-bit counter, they can be initialized by reset **(rst)**.

5. Create a **testbench** to simulate your design. Emulate 1MHz, 100Hz, and 1Hz (1sec) output clocks by changing the value of the parameter. Run your simulations at least 2 seconds.

6. **Synthesize** and **Implement** your design.

- **Stopwatch:**
    1. At this section, a stopwatch application using 7-segment displays will be designed (Look at the "stopwatch_demo.mp4" and the related design files in Ninova).
    2. There will be 2 different clock divider in order to design a stopwatch using 7-segment displays. One of them is timer 1Hz (1sec), other one is the 100 Hz used for refreshing 7-segment displays. Instantiate your **clk_divider** design with the parameter of **CLK_DIV** as 100Hz and 1Hz (1sec). The other parts of the stopwatch design code can be found from the Ninova.
    3. The values of the timer should be converted into the binary-coded-decimal (BCD) format to show the numbers properly on the display. The double-dabble algorithm is used for the BCD conversion. There are two always blocks corresponding to 100Hz and 1Hz clocks. The one triggering with 1Hz clock counts the time (seconds), and the other one triggering with 100Hz clock refreshes the 7-segment displays with the proper BCD values.

4. After instantiating your **clk_divider** design, make the proper pin configurations according to your board.
5. **Synthesize** and **Implement** your design.
6. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

Sliding LEDs

1. At this section, a **sliding LED** application with multiple speed configurations (frequencies) will be designed (See the video "sliding_leds_demo.mp4" in Ninova). There will be 3 different clock frequencies selectable with user input. The **SW** inputs will determine the clock frequencies which are **10Hz, 20Hz and 50Hz**. The circuit should also have a pausing feature. One of the 16 LEDs will be illuminated on each clock cycle and shifted.

2. Create a new project and write down a module which is called **sliding_leds**. This module will have two single-bit inputs, **clk** and **rst**; a two-bit input named **SW**, and a 16-bit output named **LED**. There will be also a parameter named **CLK_DIV** specifying the minimum rate of the clock (10Hz).

3. Define a 24-bit counter signal to count the clock cycles and a 24-bit desired rate signal determined by the selectable input. When the counter signal reaches the desired rate, the value of the LEDs will be shifted left. The value of the desired rate signal should be calculated beforehand and assigned by the SW inputs. The configuration will be SW=2'b00 -> pausing, SW=2'b01 -> 10Hz, SW=2'b10 -> 20Hz and SW=2'b11 -> 50Hz.

4. Prepare a constraint file that will connect outputs to LEDs, "SW" inputs to two of the switches, "rst" input to one of the buttons, and "clk" input to 100MHz oscillator (comment out the clock signal section and rename the port).

5. Create a **testbench** to simulate your design. Run your simulations at least 1 second.

6. **Synthesize** and **Implement** your design.

7. Perform **Post-Implementation Timing Simulation** and verify your design. Obtain timing report and show the critical path delay.

8. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

Research

1. Do research about the **Static Timing Analysis** in digital design. What are the hold time and setup time delays? Explain these terms with the timing diagrams. Explain how the maximum clock frequency can be calculated in a sequential circuit.

2. Do research about the **Metastability** concept in digital design. How does a metastable state occur? How can these metastable states be avoided? Explain.

References:

1. Nexys4 Reference Manual
2. Xilinx Constraints Guide
3. Brown&Vrasenic, "Fundamentals of Digital Logic with Verilog Design", McGraw-Hill, p.349-369