ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment V
SEQUENTIAL DESIGN & MEMORY ELEMENTS

In this experiment, different implementation methods of different memory elements will be covered. Students will realize sequential circuits and investigate their properties. Structural and Behavioral designs for different memory elements will be done in HDL code. Designs will be verified using testbenches and real-time FPGA executions. Students will also learn the usage of IPs in Vivado. Some routing and timing constraints will also be introduced.

Preliminary

Students should cover Section III of LogiCORE IP Block Memory Generator v7.3 Product Guide.

Objectives

- Become familiar with different flip-flop and latch types
- To define memory elements with Verilog
- Using testbenches to perform proper simulations
- Using IPs in Vivado.

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, getting design reports

Experiment Report Checklist

1. **SR Latch with NAND Gate**
   - Truth table and Circuit diagram,
   - Characteristic and inverse characteristic functions,
   - Verilog code and Simulation results,
   - RTL and Technology Schematics.
2. **D Flip-Flop**
   - Truth table and Circuit diagram,
   - Characteristic and inverse characteristic functions,
   - Verilog code and Simulation results
   - RTL and Technology Schematics
   - Path with the longest delay and its delay time. Explain if there is a maximum limit of clock frequency in your design.
3. **Master-Slave D Flip-Flop**
   - Add the circuit diagram to your report.
   - Explain how Master-Slave D-Flip Flop works. What is the difference from the previous D-FF.
   - Which edge of the clock signal affects the output? Explain.
   - Verilog code and Simulation results.
4. **D Flip-Flop Behavioral Design**
   - Verilog code and Simulation results.
   - RTL and Technology Schematics.
5. **8-bit Register**
   - Verilog code and Simulation results.
   - RTL and Technology Schematics
   - Explain how you define a 4x8-bit register array in Verilog.
6. **Block RAM**
   - Verilog code and Simulation results.
   - Investigate and explain the Block RAM ports.
   - Investigate and explain the structure of .coe file.
7. **BONUS: Sliding LEDs**
   - Verilog code and Simulation results.
   - If you used a different realization than the one we provide here, briefly explain it (you can use comments as well).
   - Utilization report and critical path slack.

- **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. There will be two different submissions for project arhive folder and a PDF report file.**

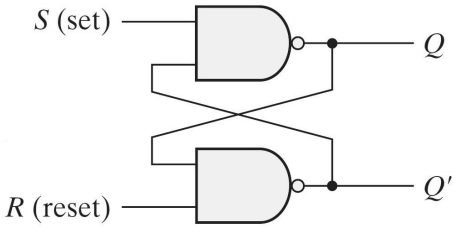

Fig.1: SR Latch with NAND gates

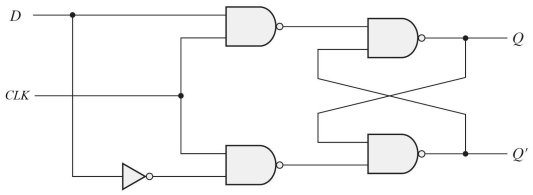

Fig.2: D Latch

SR Latch with NAND Gate

1. Realize the SR Latch **in Fig.1** with **NAND gates** using your **SSI Library**. This module is going to have 1-bit inputs **S**, **R**, and 1-bit outputs **Q**, **Qn**. Connect these ports, **S**, **R**, **Q**, **Qn** to **SW1**, **SW0**, **LED1**, **LED0**, respectively.
2. Create a **testbench** to simulate your design.
3. **Synthesize** and **Implement** your design.
4. Add the followings to your report:
   - Circuit diagram and Truth table of SR Latch,
   - Characteristic and inverse characteristic functions of SR Latch,
   - Verilog code and Simulation results,
   - RTL and Technology Schematics.
5. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

D Flip-Flop

1. Realize the **D Flip Flop in Fig.2** using your **SSI Library**. Connect the ports, **CLK**, **D**, **Q**, **Qn** to **BTN**, **SW0**, **LED1**, **LED0**, respectively.
2. Create a **testbench** to simulate your design.
3. **Synthesize** and **Implement** your design.
4. Add the followings to your report:
   - Circuit diagram and Truth table of D Flip-Flop,
   - Characteristic and inverse characteristic functions of D Latch,
   - Verilog code and Simulation results,
   - RTL and Technology Schematics,
   - Path with the longest delay and its delay time. Is there a maximum limit of clock frequency in your design? Explain.
5. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

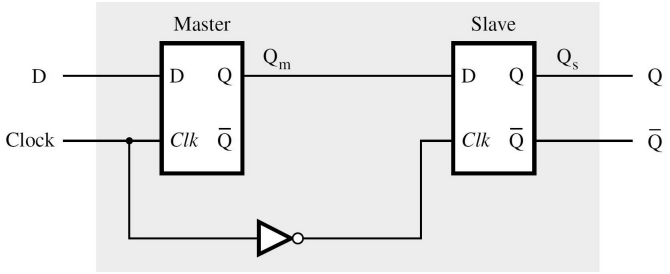

Fig.3: Master-Slave D-FF

Master-Slave D Flip-Flop

1. Realize the **Master-Slave D Flip-Flop in Fig.3** using previously designed **D Flip-Flop**.
2. Create a **testbench** to simulate your design.
3. Add the followings to your report:
   - Circuit diagram, Verilog code and Simulation results,
   - Explanation of how Master-Slave D Flip-Flop works,
   - Which edge of the clock signal effects the output?
4. Draw **Qm** and **Q** signals on the given time diagram in Fig. 4, and add it to your report.

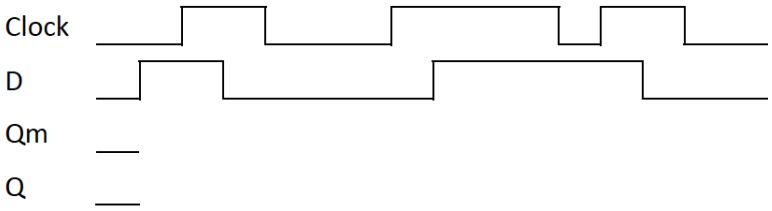

Fig.4: Time diagram for Master-Slave Flip-Flop.

D Flip-Flop Behavioral Design

1. Write down another module which has same ports with **D Flip-Flop** . This module also have 1-bit **reg** named **FF**. Using **always** block, assign **D input** to **FF** on positive-edge of clock signal. After that assign **FF** to **Q** output, and complement of **FF** to **Qn** output using **assign** keyword (continuous assignment).
2. Create a **testbench** to simulate your design.
3. **Synthesize** and **Implement** your design.
4. If you get an error during the implementation, use **set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets <netName>];** constraint.
5. Add the followings to your report:
   - Verilog code,
   - RTL and Technology Schematics.

8-bit Register

1. Write down another module which has the following ports:
   - 8-bit input **IN**
   - 1-bit input **CLK**
   - 1-bit input **CLEAR**
   - 8-bit output **OUT**
2. Using **always** block, assign values of **IN** to **OUT** on the positive-edge of CLK input.
3. When **CLEAR** input equals to 1, **OUT** must be 0. This operation must be independent from positive-edge of **CLK** input.
4. **Synthesize** and **Implement** your design.
5. Add the followings to your report:
   - Verilog code and Simulation results,
   - RTL and Technology Schematics,
   - Investigate and explain how can you define 4x8-bit of register array in Verilog.

Block RAM

1. You are going to design a Block RAM using **Block Memory Generator**. Create a block RAM with specifications below:
   - Memory Type: **Single Port RAM**,
   - Algorithm: **Minimum Area**,
   - Write Width: **8**,
   - Write Depth: **16**,
   - Operating Mode: **Write First**,
   - Enable: **Always Enabled**,
   - Load Init File: **memory.coe** (it is provided in Ninova),
   - Leave other options unchanged.
2. Write down a top module for generated block RAM which has the following ports:
   - 1-bit input **clka**
   - 1-bit input **wea**
   - 4-bit input **addra**
   - 8-bit output **douta**
3. From your constraints file; connect **clka** to **50Mhz clock**, **wea** to BTN0, **addra** to **SW0-3**, and **douta** to **LED0-7**. Connect **dina** input of your generated block RAM to a 8-bit wire in the top module.
4. Examine **memory.coe**, and place your 8-digit-length **student ID** number to addresses. Fill the all addresses (Example: 40130075-40130075, 16 digits for 16 addresses)
5. Write a module for the simulation. Read the data from the block RAM for each address.

6. **Synthesize** and **Implement** your design.
7. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.
8. Add the followings to your report:
   - Verilog Code and Simulation Results of your design
   - Functionalities of Block RAM ports
   - Structure of .coe file,

BONUS DESIGN: Sliding LEDs

- **Setting up:** At this section, a sliding LED application with multiple speed configurations will be designed (See the video "exp5_bonus_demo.mp4" at Ninova Class Files). First, create a module within a new source file, both named "sliding_leds". Your module will have two single-bit inputs, "clk and "rst"; a two-bit input named "SW", and a 16-bit output named "LED". Also, 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 100kHz internal oscillator (Meaning your system will work with 100MHz clock frequency).
- **Expected Functionality:** You are tasked to create the sliding LED behaviour seen at the video "exp5_bonus_demo.mp4" at Ninova Class Files.
  - Circuit clock frequency will be **constant 100MHz**, this will be applied internally within the board, on "clk" input.
  - Clock triggering should always happen at **rising edge**.
  - The system should contain an **asynchronous active-1 reset**, applied over "rst" input.
  - When **SW=0** system will be paused.
  - When **SW=1** LED should slide at 10Hz.
  - When **SW=2** LED should slide at 20Hz.
  - When **SW=3** LED should slide at 50Hz.
- **Step 1: Frequency Selector**
  Since the system clock is 100MHz, you will need a way to detect 10Hz ,20Hz ,50Hz periods. The circuit will include a counter for this, but first; required count destination values must be obtained.
  - Calculate how many 100MHz clock rising edges you need to count to emulate 10Hz, 20Hz, and 50Hz clock rising edges. Show your calculations.
  - Go back to your module definition. Define a global module parameter (written inside module parameter list "#()") named **"MAX_CNT_DEST"**. Set default value of this parameter to be the highest count value you've calculated previously.
  - Write a **combinational logic** that selects 10Hz's count value when SW=1 or SW=0, 20Hz's value when SW=2, and 50Hz's count value when SW=3. When defining the wire that will have the selection value, use **$clog2** function (learn about it first) for declaring wire size, so that it can change according to **"MAX_CNT_DEST"** parameter.

- **Step 2: Edge Counter**
  Here you will design the counter section of the circuit, which will count up to the selected destination value, and set a flag once it is reached there. After one clock cycle from the point where flag is set, flag value and counter value should be set to 0. This operation must be repeated continuously.
  - Create an always block that is triggered by the rising edge of the clock or the rising edge of reset. Code the described counter behavior here (Use if-else constructs). You will need regs that hold the count value (use $clog2 again), and a single bit reg that will act as your detection flag.
  - Code in the **asynchronous active-1** reset behavior for the regs you're changing within this particular always block.

- **Step 3: LED Control**
  At this part, you will code how the LEDs will be controlled depending on detection flag's value.
  - Create an always block that is triggered by the rising edge of the clock or the rising edge of reset. Code the desired sliding LED behavior here. You should only need to change LED values, so define them as regs if you haven't already done so. Remeber that the system should be paused when SW=0 (TIP: Consider using shift operator).
  - Code in the **asynchronous active-1** reset behavior for the regs you're changing within this particular always block.

- **Step 4: Simulation**
  Get the testbench provided in Ninova. Do not change anything here. This testbench is configured to work with a slower system clock so that you can observe the results faster. Perform behavioral simulation. Add waveform to your report (Tip: Collapse LED signal in order to get a better waveform visual). If you want to perform post-synthesis and post-implementation simulations, synthesize your design with **MAX_CNT_DEST=500**.

- **Step 5: Seeing is Believing**
  Implement the design with it's original **MAX_CNT_DEST** value (if you changed it to 500 before, for post-implement simulations). Add the design's LUT and FF count to your report. Generate a timing report and obtain the slack at the critical path in your design. Include this in your report as well. Generate bitstream. You will have a chance to try your design out during the lab session.

References:

1. Nexys4 Reference Manual
2. Xilinx Constraints Guide
3. Brown&Vrasenic, "Fundamentals of Digital Logic with Verilog Design", McGraw-Hill, p.349-369
4. M. Mano, "Digital Design", Chapters 6.2, 6.3, p.203-219
5. LogiCORE IP Block Memory Generator v7.3 Product Guide
6. FIFO Generator v13.1 LogiCORE IP Product Guide