ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment V
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. **Simple Memory Element**
   - Circuit Diagram of the Simple Memory Element.
   - Verilog code and Simulation results.
   - Synthesize warning given from Vivado about combinatorial loops.
   - Constraint to prevent the combinatorial loops warning.
   - RTL and Technology Schematic.
2. **SR Latch with NOR Gate**
   - Circuit diagram of the SR Latch using NOR gates.
   - Truth and excitation tables.
   - Characteristic and inverse characteristic functions.
   - Verilog code and Simulation results.
3. **SR Latch with NAND Gate**
   - Circuit diagram of the SR Latch using NAND gates.
   - Truth and excitation tables.
   - Characteristic and inverse characteristic functions.
   - Verilog code and Simulation results.
4. **D Flip-Flop**
   - Circuit diagram of the D FF.
   - Truth and excitation tables.
   - 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.
5. **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.
6. **D Flip-Flop Behavioral Design**
   - Verilog code and Simulation results.
   - RTL and Technology Schematics.
7. **8-bit Register**
   - Verilog code and Simulation results.
   - RTL and Technology Schematics

- Explain how you define a 4x8-bit register array in Verilog.

8. **Block RAM**
   - Verilog code and Simulation results.
   - Investigate and explain the Block RAM ports.
   - Investigate and explain the structure of .coe file.
9. **FIFO**
   - Verilog code and Simulation results.
   - Explanation of FIFO data structure,
   - Utilization report.

- **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.**

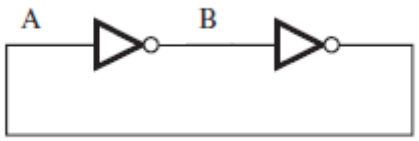

Fig.1: Simple Memory Element

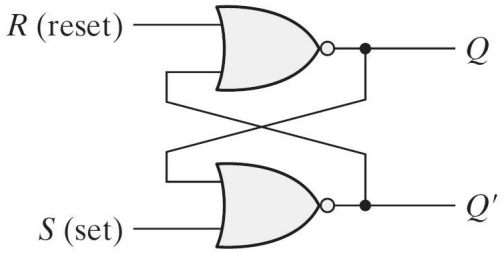

Fig.2: SR Latch with NOR gates

Simple Memory Element

1. Realize the simple memory element **in Fig.1** which consists of two **NOT gates** using your **SSI Library**.
2. Create a Top Module and assign one of the outputs of NOT gates to a **LED**. Assign the input to a **SW**.
3. Since your module does not have any inputs, use **(* dont_touch *)** constraint to prevent trimming unconnected signals. Use **set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets <netName>];** constraint to allow combinatorial loops in your design.
4. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.
5. Add the followings to your report:
   - Circuit Diagram, Verilog code and Simulation results,
   - Synthesize warning regarding combinatorial loops,
   - Constraint to prevent the warning,
   - RTL and Technology Schematics.

SR Latch with NOR Gate

1. Realize the SR Latch **in Fig.2** with **NOR 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. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.
4. Add the followings to your report:
   - Circuit diagram, Verilog code and Simulation results,
   - Truth and excitation table of SR Latch,
   - Characteristic and inverse characteristic function of SR Latch.

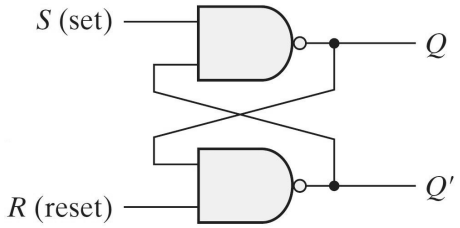

Fig.3: SR Latch with NAND gates

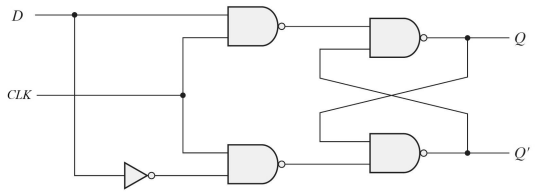

Fig.4: D Latch

SR Latch with NAND Gate

1. Realize the SR Latch **in Fig.3** with **NAND gates** using your **SSI Library**. Connect the ports as in the previous one SR Latch with NOR Gate.
2. Create a **testbench** to simulate your design.
3. **Synthesize** and **Implement** your design.
4. Add the followings to your report:
   - Circuit diagram, Verilog code and Simulation results,
   - Truth and excitation table of SR Latch,
   - Characteristic and inverse characteristic function of SR Latch.

D Flip-Flop

1. Realize the **D Flip Flop in Fig.4** 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. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.
4. Add the followings to your report:
   - Circuit diagram, Verilog code and Simulation results,
   - Truth and excitation table of D Latch
   - Characteristic and inverse characteristic function of D Latch
   - 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.

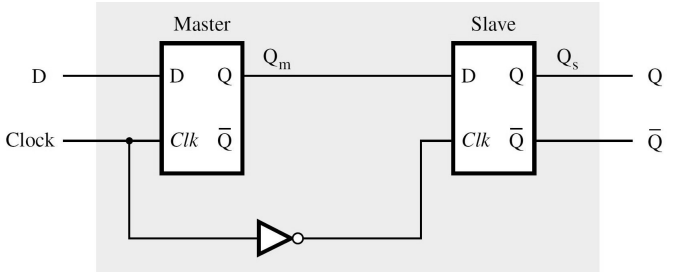

Fig.5: Master-Slave D-FF

Master-Slave D Flip-Flop

1. Realize the **Master-Slave D Flip-Flop in Fig.5** using previously designed **D Latch**.
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. 6, and add it to your report.

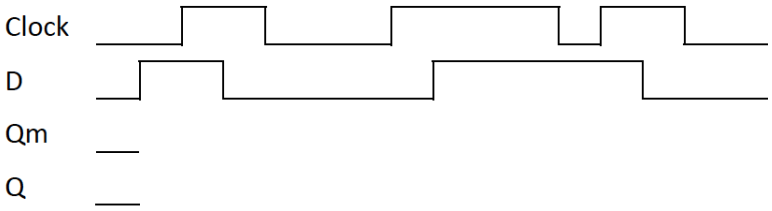

Fig.6: Time diagram for Master-Slave Flip-Flop.

D Flip-Flop Behavioral Design

1. Write down another module which has same ports with **Edge-Triggered D Flip-Flop** . This module is going to have 1-bit **reg** named **FF**. Using **always** block, assign **D input** to **FF** on positive-edge of clock signal. Assign **FF** to **Q** output, and complement of **FF** to **Qn** output.
2. Create a **testbench** to simulate your design.
3. If you get an error during the implementation, use **set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets <netName>];** constraint.
4. **Synthesize** and **Implement** your design.
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. This module is going to 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. Make the connections of the ports **IN**, **CLK**, **CLEAR**, **OUT** to **SW0-7**, **BTN0**, **BTN1**, **LED0-7**. Then, generate **.BIT file** and program your FPGA.
5. Add the followings to your report:
   - Verilog code and Simulation results
   - RTL and Technology Schematics
   - Investigate and explain how 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. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.
7. Add the followings to your report:
    - Verilog Code and Simulation Results of your design
    - Functionalities of Block RAM ports
    - Structure of .coe file,

FIFO

1. You are going to design a FIFO Memory using **FIFO Generator**. Create a FIFO with specifications below:
    - Interface Type: **Native**,
    - Read Mode: **Standart FIFO**,
    - Write Width: **8**,
    - Write Depth: **16**,
    - Status Flags: **Overflow**✓, **Underflow**✓
    - Data Counts: **Data Count**✓
2. Write down a top module for generated FIFO which has the following ports:
    - 1-bit input **clk**,
    - 1-bit input **wr_en**,
    - 1-bit input **rd_en**,
    - 1-bit input **srst** (Synchronous Reset),
    - 8-bit input **din**,
    - 8-bit output **dout**,
    - 4-bit output **data_count**,
    - 1-bit output **emty**,
    - 1-bit output **full**,
    - 1-bit output **overflow**,
    - 1-bit output **underflow**
3. Make the connections as in the Block RAM.
4. Write a module for the simulation. Write arbitrary data to **FIFO** until the **full** and **overflow** flags are set. Then, read the data until the **empty** and **underflow** flags are set. Consider the **wr_en** and **rd_en** signals for the write and read operations.
5. **Synthesize** and **Implement** your design.
6. Explain **FIFO** Data Structure. What is its principle and how it works?
7. Add the followings to your report:
    - Verilog Code and Simulation Results of your design,
    - Explanation of FIFO data structure,
    - Utilization report

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