ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment VI
CLOCK MANAGEMENT & MEMORY

2024

In this experiment, clock management methods and implementation of memory blocks will be covered. Students will realize sequential circuits and their properties. Structural and Behavioral designs will be applied. Designs will be verified using testbenches and real-time FPGA executions. Students will also learn the usage of IPs in Vivado.

Objectives

- Understanding and managing the Clock Networks
- Defining Memory Blocks
- Using testbenches to perform proper simulations
- Using IPs in Vivado.

Requirements

- 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. **Clock Generation**
   - Verilog and Testbench Codes .
   - Simulation Results.
   - Utilization Report (primitives).
   - Clock Network Report.
2. **Clock Gating**
   - Verilog and Testbench Codes .
   - Simulation Results.
   - Utilization Report (primitives).
   - Clock Network Report.
3. **Block RAM**
   - Verilog and Testbench Codes.
   - Simulation Results.
   - Utilization Report (primitives).
4. **FIFO for Clock Domaing Crossing**
   - Verilog and Testbench Codes.
   - Simulation Results.
   - Utilization Report (primitives).
   - Clock Network Report.
   - Clock Interaction Report.
   - Post-Implementation Timing Simulation.
5. **Research**
   - Research about **Clock Tree Synthesis**.
   - Research about **Dual FF Synchronizers** for Clock Domain Crossing.

- **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.**

Clock Generation

1. In Xilinx 7-series FPGAs, the clock management tiles provide clock frequency synthesis. These tiles include one mixed-mode clock manager (MMCM) and one phase-locked loop (PLL). The external clock input provide a reference clock frequency to these MMCMs and PLLs.
2. Create a new project, open IP Catalog and type **Clocking Wizard**. Use the default settings for clock generation:
   - Primitive: **MMCM**
   - Clocking Features: **Frequency Synthesis, Phase Alignment**
   - Jitter Optimization: **Balanced**
   - Input Clock: **Primary, 100MHz, Single Ended Clock Capable Pin**
3. In the outport clock section, choose two output clocks with the frequencies of **80MHz** and **60MHz**. Leave the other sections default and generate the IP block.
4. Write down a top module which is called **clock_gen**. This module should have 1-bit inputs **clk** and **rst**; 7-bit outputs **cnt100**, **cnt80** and **cnt60** corresponding to counters for 100MHz, 80MHz and 60MHz clocks, respectively. Comment out the clock section in the master .xdc file and rename the clock port as clk.
5. Instantiate the IP block into the top module. Intantiation template for the IP block can be found from the IP sources section. The clock input will be the system clock **(clk)** and reset will be **rst** signal. Leave the locked port blank (if exists). Finally, define wires **clk80** and **clk60** for the newly generated output clocks.
6. There will be three different parallel **always blocks** for the counters **cnt100**, **cnt80** and **cnt60**. These counters will increment on each positive edge of the clock signals. When the cnt100 hits the number 100, the counting should stop. Similarly, cnt80 should stop when the number is 80 and cnt60 should stop when the number is 60.
7. Create a **testbench** to simulate your design. Show the newly generated clocks in the simulation waveform (by dragging the objects to the waveform). Show the cycles that the counters stop. Be careful that the generation of the new clocks will require some time.
8. **Synthesize** and **Implement** your design.
9. Obtain **Utilization Report** and show the usage of primitives.
10. Obtain **Clock Networks Report** and show which clocks are used.

Clock Gating

1. Clock gating is a power optimization method used in digital design. It reduces dynamic power dissipation by stopping the clock signal when it is not in use. In Xilinx 7-series FPGAs, clock gating method is implemented by **BUFR** primitive. This primitive can also be used for clock dividing alongside with clock gating.
2. Create a new project and write down a top module which is called **clock_gating**. This module should have 1-bit inputs **clk** and **rst**; 6-bit outputs **cnt50** and **cnt25** corresponding to counters for 50MHz and 25MHz clocks, respectively. Comment out the clock section in the master .xdc file and rename the clock port as clk.

3. Instantiate the two **BUFR** primitives into the top module. Intantiation template for the primitive can be found from the Ninova. The clock input **(I)** will be the system clock and clear **(CLR)** will be logic-0. Define wires **clk50_en** and **clk25_en** for the control enable **(CE)** inputs. Finally, define wires **clk50** and **clk25** for the newly generated output clocks **(O)**. Choose the **BUFR_DIVIDE** parameter as 2 and 4 for the 50MHz and 25MHz clocks, respectively.

4. There will be two different parallel **always blocks** for the counters **cnt50** and **cnt25**. These counters will increment on each positive edge of the clock signals. When the cnt50 hits the number 50, the counting should stop. Similarly, cnt25 should stop when the number is 25. Obtain this behavior by using control enable signals **clk50_en** and **clk25_en**. Use asynchronous reset to initialize these enable signals.

5. Create a **testbench** to simulate your design. Show the newly generated clocks in the simulation waveform (by dragging the objects to the waveform). Show the cycles that the clocks are gated (stop).

6. **Synthesize** your design. After synthesis, write **create_generated_clock** constraints for the user-defined clocks. To obtain proper analysis and reports from the Vivado, user-defined clocks (not from the Wizard) should be added to the constraint file.
   - create_generated_clock -name clk50MHZ -source [get_pins BUFR_inst/I] -divide_by 2 [get_pins BUFR_inst/O];
   - create_generated_clock -name clk25MHZ -source [get_pins BUFR_inst2/I] -divide_by 4 [get_pins BUFR_inst2/O];
   - Change the name of the instances if it is different in the technology schematic.

7. **Resynthesize** and **Implement** your design.

8. Obtain **Utilization Report** and show the usage of primitives.

9. Obtain **Clock Networks Report** and show which clocks are used.

Block RAM

1. At this section, a Block RAM will be designed by using **Block Memory Generator** from the IP Catalog. 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 and instantiate the generated Block RAM. The instantiation template can be found from the IP sources. Block RAM has the following ports:
   - 1-bit input **clka** : Clock
   - 1-bit input **wea** : Write Enable

- 1-bit input **ena** : Enable
- 4-bit input **addra** : Address to write/read
- 8-bit output **dina** : Data Input
- 8-bit output **douta** : Data Output

3. The top module should have the same ports with the Block RAM except **ena** input. Connect **ena** input to logic-1 and make the proper connections of the Block RAM with the top module ports.

4. Generate a **50Mhz clock** with an any method, and add the **create_generated_clock** constraint if necessary.

5. In the constraints file; connect **clka** to generated **50Mhz clock**, **wea** to **BTN0**, **addra** to **SW[3:0]**, **dina** to **SW[15:7]** and **douta** to **LED[7:0]**.

6. 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).

7. Create a **testbench** to simulate your design. Read the data from the Block RAM for each address first. Then write your **student ID** upside down (inverse order) for each address. After that, read the data from the Block RAM again. Use **$display()** function to show the simulation results in Tcl Console.

8. **Synthesize** and **Implement** your design.

9. Obtain **Utilization Report** and show the usage of primitives.

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

FIFO for Clock Domain Crossing

1. **Clock Domain Crossing** is the process of passing a signal or multiple data from one clock domain to another clock domain. Passing signals from one clock domain into another one may cause metastability issues. In order to prevent it, synchronizer circuits inluding dual flip-flops with an handshake mechanism can be used for single bit signals. For the large amount of data or consecutive signals, **FIFO** structures are more efficient.

2. In the **FIFO** structures, data is pushed from the source clock domain and is popped out from the destination clock domain. The depth of the FIFO should be determined according to frequency rate of the clock domains so that it may not overflow or underflow the data.

3. Create a new project, open IP Catalog and type **FIFO Generator**. Create a FIFO with specifications below:
   - FIFO Implementation: **Independent Clocks Block RAM**,
   - Synchronization Stages: **2**,
   - Read Mode: **Standard FIFO**,
   - Write Width: **8**, Write Depth: **32**,
   - Read Width: **8**, Read Depth: **32**,
   - No ECC, No Output Registers,

- **Reset Pin**✓, Asynchronous Reset, no Reset Synchronization,
- Status Flags: **Overflow**✓, **Underflow**✓, **Active High**,
- Leave the other parts default

4. Write down a top module and instantiate the generated FIFO. The instantiation template can be found from the IP sources. FIFO should have the following ports:
   - 1-bit input **wr_clk**,
   - 1-bit input **rd_clk**,
   - 1-bit input **wr_rst**,
   - 1-bit input **rd_rst**,
   - 1-bit input **wr_en**,
   - 1-bit input **rd_en**,
   - 8-bit input **din**,
   - 8-bit output **dout**,
   - 1-bit output **empty**,
   - 1-bit output **full**,
   - 1-bit output **overflow**,
   - 1-bit output **underflow**
5. The top module should have the same ports with the FIFO except **wr_clk, rd_clk, wr_rst** and **rd_rst** inputs. The top module should have one reset **(rst)** and one system clock **(clk)**. Connect **rst** input to **wr_rst** and **rd_rst**.
6. Generate a **50Mhz clock** and a **25Mhz clock** with an any method in the top module. They should be synchronous (no phase difference). Connect **50Mhz clock** to **wr_clk** and connect **25Mhz clock** to **rd_clk**.
7. Create a **testbench** to simulate your design. The writing and reading operations will be done simultaneously but within different clock domains. Push integer numbers to the FIFO by an increasing order (0-64). These numbers will be popped out at the same time with a slower frequency. Keep in mind that there is a latency for reading operations and flag assertions.
8. Run **Behavioral Simulation** and show the full and overflow flags assertions.
9. **Synthesize** and **Implement** your design.
10. Obtain **Utilization Report** and show the usage of primitives.
11. Obtain **Clock Networks Report** and show which clocks are used.
12. Obtain **Clock Networks Interaction** and show the regions that source and destination clocks are different.
13. Run **Post-Implementation Timing Simulation** and show the full and overflow flags assertions.

Research

1. Do research about the **Clock Tree Synthesis** in digital design. Give a few examples about **Clock Tree Architectures** and explain. What are the key parameters of a clock tree? Explain the concepts of clock latency, clock skew, jitter and clock slew.
2. Do research about the **Dual FF Synchronizers** for Clock Domain Crossing. Show the logic diagram and explain.

References:

1. Nexys4 Reference Manual
2. 7 Series FPGAs Clocking Resources User Guide
3. LogiCORE IP Block Memory Generator v8.4 Product Guide
4. FIFO Generator v13.2 LogiCORE IP Product Guide