# DIGITAL SYSTEM DESIGN APPLICATIONS

# (CRN: 11275)

# THE REPORT OF EXPERIMENT – 6


Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

Yusuf Tekin
040200043

# 1. Clock Generation

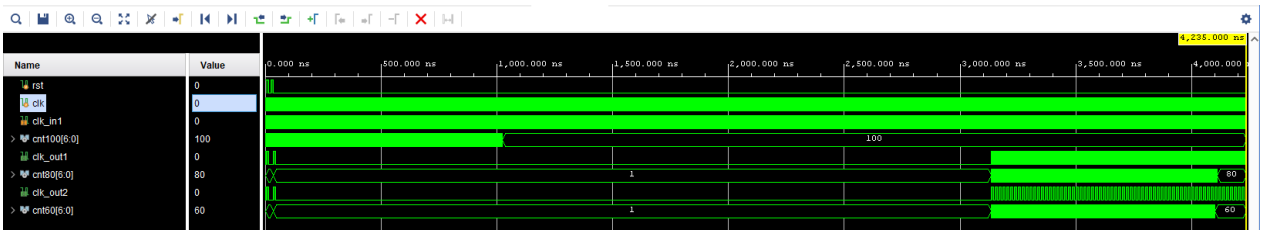
```
`timescale 1ns / 1ps

module top_module(
    input clk,
    input rst,
    output reg [6:0] cnt100,
    output reg [6:0] cnt80,
    output reg [6:0] cnt60
    );

    wire clk80, clk60;

    // Clocking Wizard instantiation
    clk_wiz_0 clknetwork (
        .clk_out1(clk80),
        .clk_out2(clk60),
        .reset(rst),
        .locked(),
        .clk_in1(clk)
    );

    // Counter for 100 MHz clock
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            cnt100 <= 7'b0;
        end else if (cnt100 < 100) begin
            cnt100 <= cnt100 + 1;
        end
    end

    // Counter for 80 MHz clock
    always @(posedge clk80 or posedge rst) begin
        if (rst) begin
            cnt80 <= 7'b0;
        end else if (cnt80 < 80) begin
            cnt80 <= cnt80 + 1;
        end
    end

    // Counter for 60 MHz clock
    always @(posedge clk60 or posedge rst) begin
        if (rst) begin
            cnt60 <= 7'b0;
        end else if (cnt60 < 60) begin
            cnt60 <= cnt60 + 1;
        end
    end

endmodule
```

*Clock Generation Verilog Code*

Yusuf Tekin
040200043

```
`timescale 1ns / 1ps

module top_module_tb;

reg clk, rst;
wire [6:0] cnt100, cnt80, cnt60;

top_module dut (
    .clk(clk),
    .rst(rst),
    .cnt100(cnt100),
    .cnt80(cnt80),
    .cnt60(cnt60)
);

always #5 clk = ~clk;

initial begin

    clk = 0; rst = 1; #15;
    rst = 0; #10; rst = 1;
    #10; rst = 0;
    #4200;
    $finish();
end
endmodule
```

*Clock Generation Testbench Code*



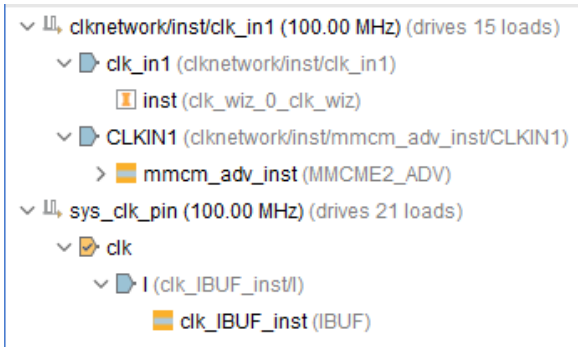

*Clock Generation Behavioral Simulation*

**Primitives**

| Ref Name | Used | Functional Category |
|---|---|---|
| OBUF | 21 | IO |
| FDCE | 20 | Flop & Latch |
| LUT5 | 6 | LUT |
| LUT3 | 6 | LUT |
| LUT4 | 4 | LUT |
| LUT6 | 3 | LUT |
| LUT2 | 3 | LUT |
| LUT1 | 3 | LUT |
| BUFG | 3 | Clock |
| IBUF | 2 | IO |
| MMCME2_ADV | 1 | Clock |

*Clock Generation Primitive Usage*

Yusuf Tekin
040200043




*Clock Generation Clock Networks Report*

Yusuf Tekin
040200043

# 2. Clock Gating

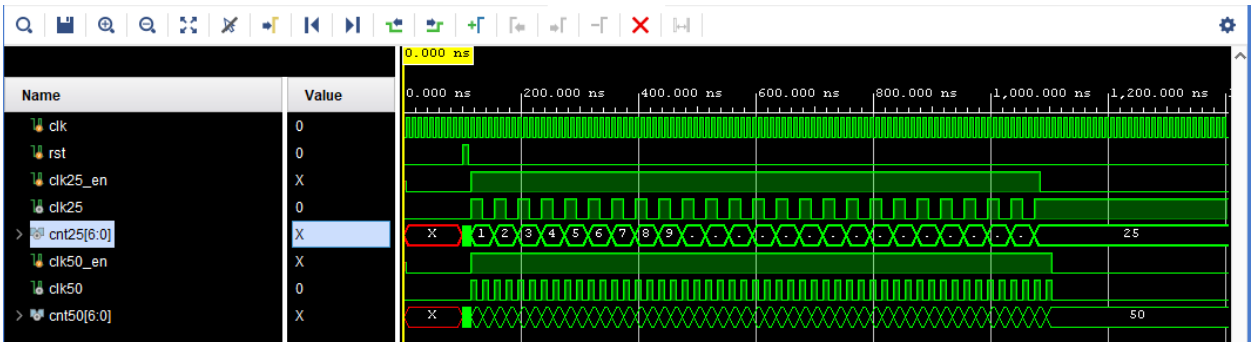
```
`timescale 1ns / 1ps
module clock_gating(
    input clk,
    input rst,
    output reg [6:0] cnt50,
    output reg [6:0] cnt25
    );

reg clk50_en, clk25_en;
wire clk50, clk25;

integer i;

BUFR #(
   .BUFR_DIVIDE("2"),
   .SIM_DEVICE("7SERIES")
)
BUFR_ins_1 (
   .O(clk50),
   .CE(clk50_en),
   .CLR(1'b0),
   .I(clk)
);

BUFR #(
   .BUFR_DIVIDE("4"),
   .SIM_DEVICE("7SERIES")
)
BUFR_inst_2 (
   .O(clk25),
   .CE(clk25_en),
   .CLR(1'b0),
   .I(clk)
);

always @(posedge clk50 or posedge rst) begin
    if(rst) begin
        cnt50 <= 7'b0;
    end else if(cnt50 < 50) begin
        cnt50 <= cnt50 + 1;
    end
end

always @(posedge clk25 or posedge rst) begin
    if(rst) begin
        cnt25 <= 7'b0;
    end else if(cnt25 < 25) begin
        cnt25 <= cnt25 + 1;
    end
end

always @(posedge clk or posedge rst) begin
    if (rst) begin
        clk50_en <= 1'b0;
        clk25_en <= 1'b0;
    end else begin
        if (cnt50 < 50)
            clk50_en <= 1'b1;
        else
            clk50_en <= 1'b0;

        if (cnt25 < 25)
            clk25_en <= 1'b1;
        else
            clk25_en <= 1'b0;
    end
end
endmodule
```

*Clock Gating Verilog Code*

Yusuf Tekin
040200043

```
`timescale 1ns / 1ps

module clock_gating_tb;

wire [6:0] cnt50, cnt25;
reg clk, rst;

clock_gating dut(
    .clk(clk),
    .rst(rst),
    .cnt50(cnt50),
    .cnt25(cnt25)
    );

always #5 clk = ~clk;

initial begin
    clk = 1'b0; rst = 1'b0; #100;
    rst = 1'b1; #10; rst = 1'b0;
    #4200;
    $finish();
end
endmodule
```

*Clock Gating Testbench Code*

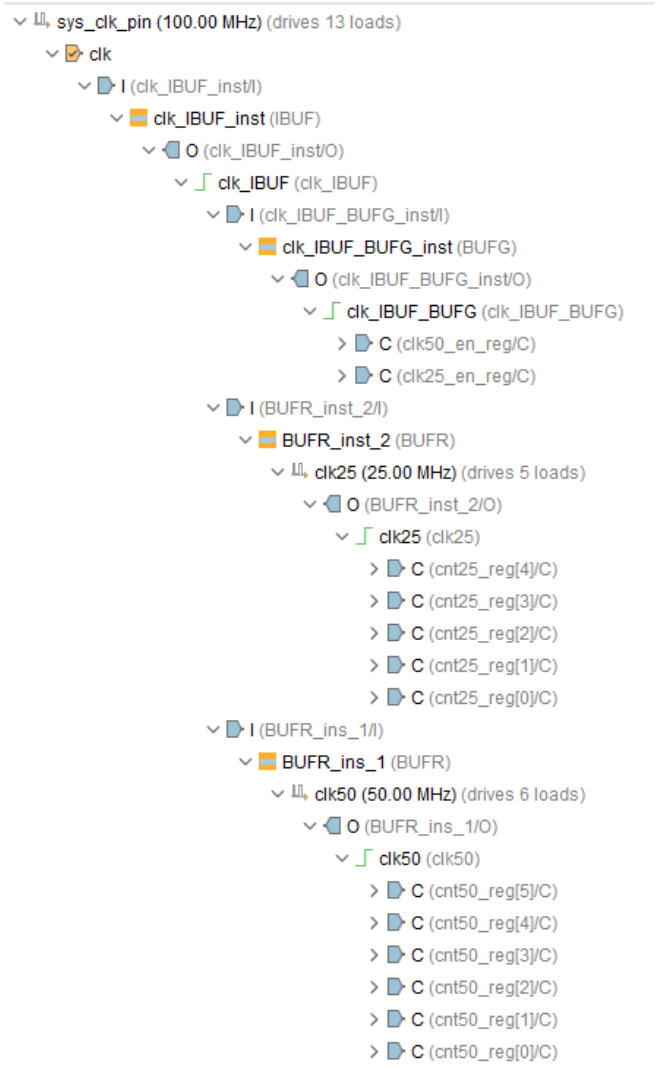



*Clock Gating Behavioral Simulation*

**Primitives**

| Ref Name | Used | Functional Category |
|---|---|---|
| OBUF | 14 | IO |
| FDCE | 13 | Flop & Latch |
| LUT5 | 4 | LUT |
| LUT4 | 2 | LUT |
| LUT3 | 2 | LUT |
| LUT2 | 2 | LUT |
| LUT1 | 2 | LUT |
| IBUF | 2 | IO |
| BUFR | 2 | Clock |
| LUT6 | 1 | LUT |
| BUFG | 1 | Clock |

*Clock Gatin Primitive Usage*




*Clock Gating Clock Network Report*

# 3. Block RAM

```
//Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

//Copyright 2022-2024 Advanced Micro Devices, Inc. All Rights
Reserved.
//-----------------------------------------------------------------------
------------
//Tool Version: Vivado v.2024.1 (win64) Build 5076996 Wed May 22
18:37:14 MDT 2024
//Date        : Mon Dec  9 11:36:38 2024
//Host        : DESKTOP-KMJ9JAC running 64-bit major release  (build
9200)
//Command     : generate_target block_ram_50MHz_wrapper.bd
//Design      : block_ram_50MHz_wrapper
//Purpose     : IP block netlist
//-----------------------------------------------------------------------
------------
`timescale 1 ps / 1 ps

module block_ram_50MHz_wrapper
   (addra_0,
    clk_100MHz,
    dina_0,
    douta_0,
    wea_0);
  input [3:0]addra_0;
  input clk_100MHz;
  input [7:0]dina_0;
  output [7:0]douta_0;
  input [0:0]wea_0;

  wire [3:0]addra_0;
  wire clk_100MHz;
  wire [7:0]dina_0;
  wire [7:0]douta_0;
  wire [0:0]wea_0;

  block_ram_50MHz block_ram_50MHz_i
       (.addra_0(addra_0),
        .clk_100MHz(clk_100MHz),
        .dina_0(dina_0),
        .douta_0(douta_0),
        .wea_0(wea_0));
endmodule
```

*Block RAM Verilog Code*

Yusuf Tekin
040200043

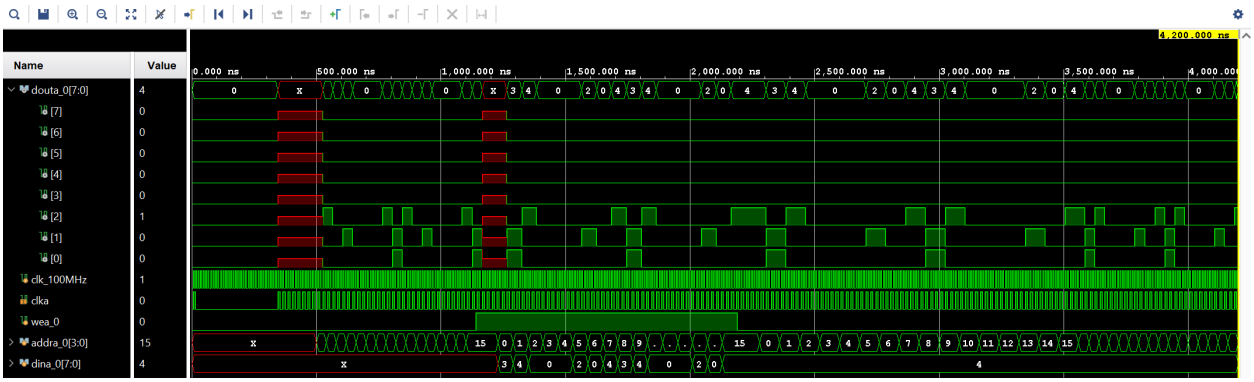
```
`timescale 1ns / 1ps

module block_ram_50MHz_wrapper_tb;

wire [7:0] douta_0;
reg clk_100MHz, wea_0;
reg [3:0] addra_0;
reg [7:0] dina_0;

integer i;

block_ram_50MHz_wrapper dut(
    .addra_0(addra_0),
    .clk_100MHz(clk_100MHz),
    .dina_0(dina_0),
    .douta_0(douta_0),
    .wea_0(wea_0)
    );

always #5 clk_100MHz = ~clk_100MHz;

initial begin
    clk_100MHz = 1'b0; wea_0 = 1'b0; #500;

    $display("\n--- Reading Initial Data ---");
    for (i = 0; i < 16; i = i + 1) begin
        addra_0 = i;
        #40;
        $display("Address %0d: Data = %0d", addra_0, douta_0);
    end

    $display("\n--- Writing School Number in reverse to the  RAM ---");
    wea_0 = 1'b1; #90;
    addra_0 = 4'b0000;  dina_0 = 8'b00000011; #60;
    addra_0 = 4'b0001;  dina_0 = 8'b00000100; #60;
    addra_0 = 4'b0010;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b0011;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b0100;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b0101;  dina_0 = 8'b00000010; #60;
    addra_0 = 4'b0110;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b0111;  dina_0 = 8'b00000100; #60;
    addra_0 = 4'b1000;  dina_0 = 8'b00000011; #60;
    addra_0 = 4'b1001;  dina_0 = 8'b00000100; #60;
    addra_0 = 4'b1010;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b1011;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b1100;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b1101;  dina_0 = 8'b00000010; #60;
    addra_0 = 4'b1110;  dina_0 = 8'b00000000; #60;
    addra_0 = 4'b1111;  dina_0 = 8'b00000100; #60;

    wea_0 = 1'b0; #90;
    addra_0 = 4'b0000; #80;
    addra_0 = 4'b0001; #80;
    addra_0 = 4'b0010; #80;
    addra_0 = 4'b0011; #80;
    addra_0 = 4'b0100; #80;
    addra_0 = 4'b0101; #80;
    addra_0 = 4'b0110; #80;
    addra_0 = 4'b0111; #80;
    addra_0 = 4'b1000; #80;
    addra_0 = 4'b1001; #80;
    addra_0 = 4'b1010; #80;
    addra_0 = 4'b1011; #80;
    addra_0 = 4'b1100; #80;
    addra_0 = 4'b1101; #80;
    addra_0 = 4'b1110; #80;
    addra_0 = 4'b1111; #80;
    $display("\n--- Writing Complete ---");
    $display("\n--- Reading data from RAM ---");
    for (i = 0; i < 16; i = i + 1) begin
        addra_0 = i;
        #40;
        $display("Address %0d: Data = %0d", addra_0, douta_0);
    end

    $display("\nBenchmark finished.\n");
    $finish();

end
endmodule
```

*Block RAM Testbench Code*

```
--- Reading Initial Data ---
Address 0: Data = 4
Address 1: Data = 0
Address 2: Data = 2
Address 3: Data = 0
Address 4: Data = 0
Address 5: Data = 0
Address 6: Data = 4
Address 7: Data = 3
Address 8: Data = 4
Address 9: Data = 0
Address 10: Data = 2
Address 11: Data = 0
run 4500 ns
Address 12: Data = 0
Address 13: Data = 0
Address 14: Data = 4
Address 15: Data = 3

--- Writing School Number in reverse to the  RAM ---

--- Writing Complete ---

--- Reading data from RAM ---
Address 0: Data = 3
Address 1: Data = 4
Address 2: Data = 0
Address 3: Data = 0
Address 4: Data = 0
Address 5: Data = 2
Address 6: Data = 0
Address 7: Data = 4
Address 8: Data = 3
Address 9: Data = 4
Address 10: Data = 0
Address 11: Data = 0
Address 12: Data = 0
Address 13: Data = 2
Address 14: Data = 0
Address 15: Data = 4

Benchmark finished.

$finish called at time : 4200 ns : File "C:/Users/jsp
```

*Block RAM Simulation TCL Console Log*

Yusuf Tekin
040200043

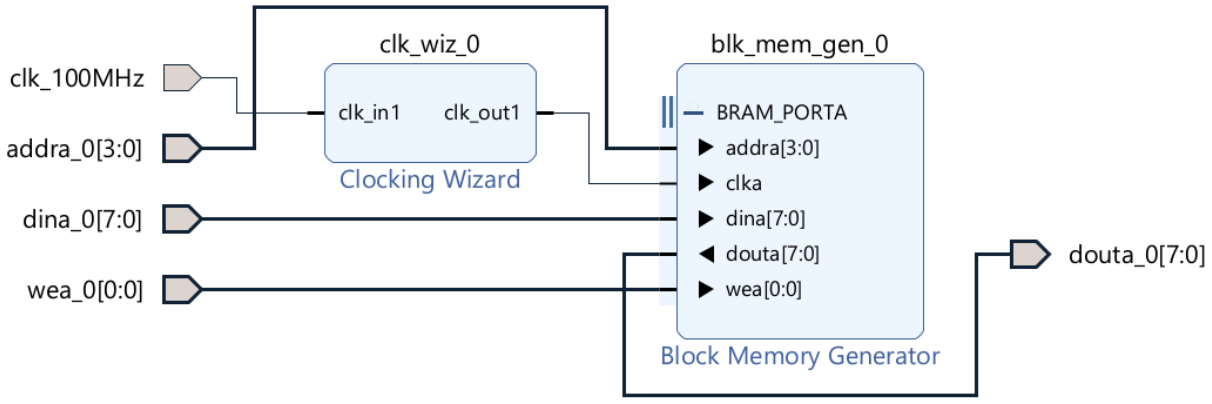



*Block RAM Behavioral Simulation*

**Primitives**

| Ref Name | Used | Functional Category |
|---|---|---|
| IBUF | 14 | IO |
| OBUF | 8 | IO |
| BUFG | 2 | Clock |
| RAMB18E1 | 1 | Block Memory |
| MMCME2_ADV | 1 | Clock |

*Block RAM Primitive Usage*




*Block RAM Block Design*

Yusuf Tekin
040200043

# 4. FIFO for Clock Domain Crossing

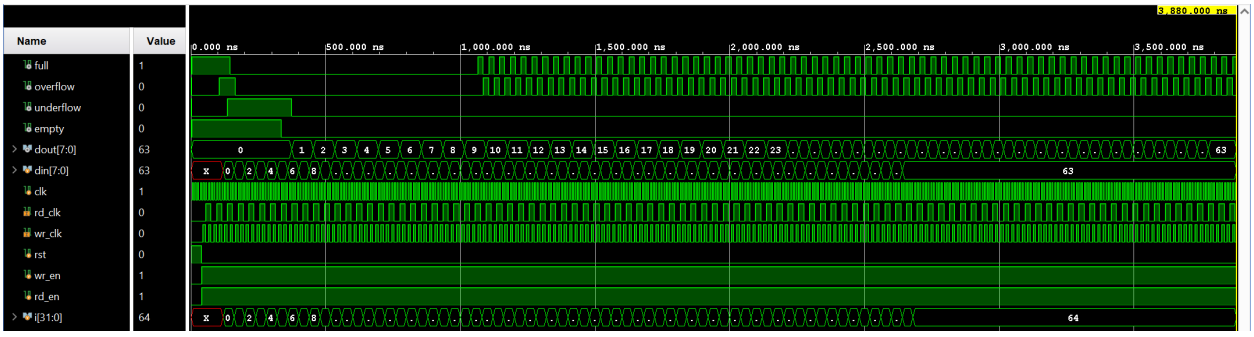
```
`timescale 1ns / 1ps

module top_module(
    input clk,
    input rst,
    input wr_en,
    input rd_en,
    input [7:0] din,
    output [7:0] dout,
    output empty,
    output full,
    output overflow,
    output underflow
    );

reg [2:0] counter = 3'b0;
wire wr_clk, rd_clk;

assign wr_clk = counter[0];
assign rd_clk = counter[1];

always @(posedge clk or posedge rst) begin
     if(rst) counter <= 3'b0;
     else counter <= counter + 1;
end

fifo_generator_0 memory1(
    .wr_clk(wr_clk),
    .wr_rst (rst),
    .rd_clk(rd_clk),
    .rd_rst(rst),
    .din(din),
    .wr_en(wr_en),
    .rd_en(rd_en),
    .dout(dout),
    .full(full),
    .overflow(overflow),
    .empty(empty),
    .underflow(underflow));

endmodule
```

*FIFO Verilog Code*

Yusuf Tekin
040200043

```
`timescale 1ns / 1ps

module top_module_tb;

wire full, overflow, underflow, empty;
wire [7:0] dout;
reg [7:0] din;
reg clk, rst, wr_en, rd_en;
integer i;

top_module dut(
    .clk(clk),
    .rst(rst),
    .wr_en(wr_en),
    .rd_en(rd_en),
    .din(din),
    .dout(dout),
    .empty(empty),
    .full(full),
    .overflow(overlow),
    .underflow(underflow));

always #5 clk = ~clk;

initial begin
    clk = 1'b0; rst = 1'b1; wr_en = 1'b0; rd_en = 1'b0; #40;
    rst = 1'b0; wr_en = 1'b1; rd_en = 1'b1;  #80;

    for (i = 0; i < 64; i = i + 1) begin
        din <= i; #40;
    end
    #1200;
    $finish();
end
endmodule
```

*FIFO Testbench Code*




*FIFO Behavioral Simulation*

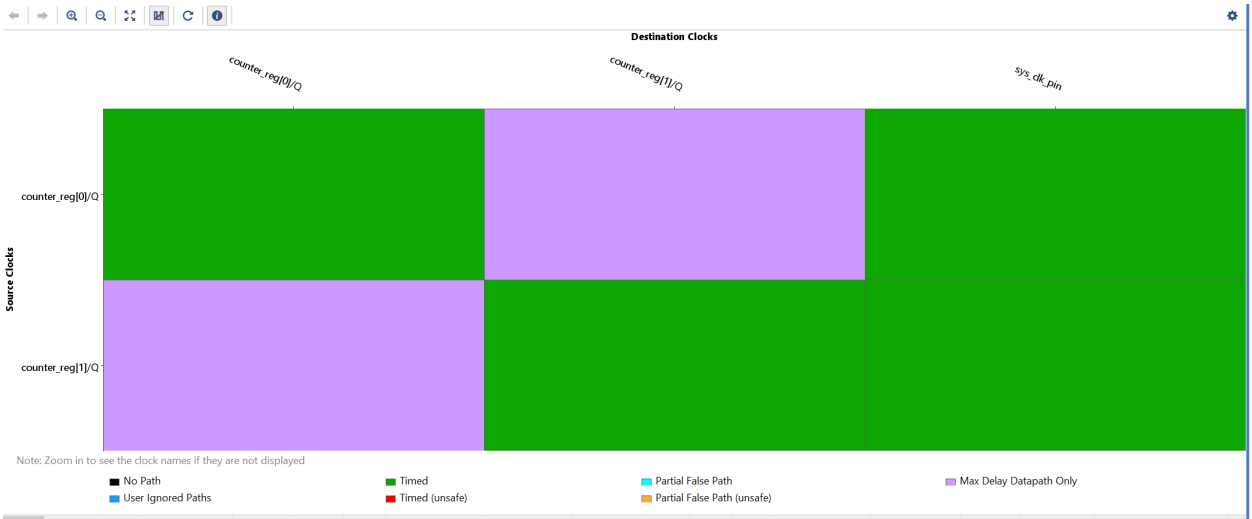
**Primitives**

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 42 | Flop & Latch |
| FDCE | 24 | Flop & Latch |
| LUT2 | 17 | LUT |
| OBUF | 12 | IO |
| IBUF | 12 | IO |
| FDPE | 10 | Flop & Latch |
| LUT4 | 8 | LUT |
| LUT6 | 6 | LUT |
| LUT3 | 5 | LUT |
| LUT5 | 4 | LUT |
| LUT1 | 3 | LUT |
| BUFG | 3 | Clock |
| RAMB18E1 | 1 | Block Memory |

*FIFO Primitive Usage*

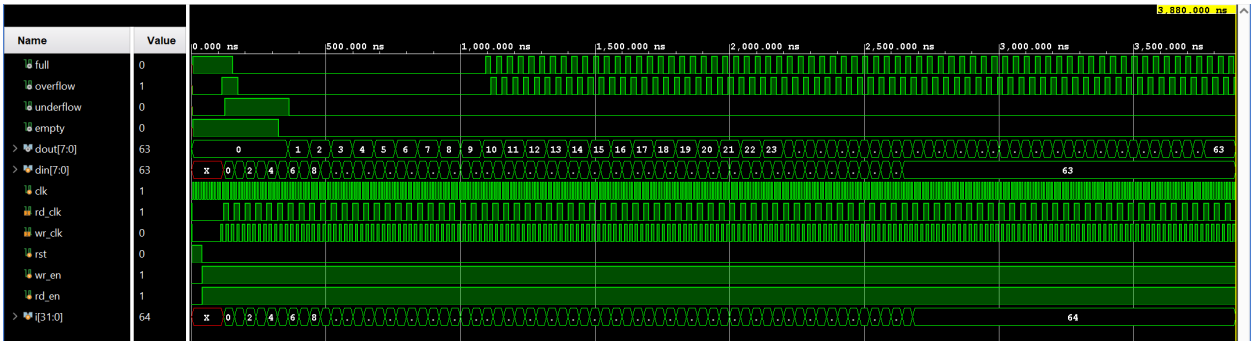

*Clock Network Reports*

Yusuf Tekin
040200043




| Id | Source Clock | Destination Clock | Edges (WNS) | WNS (ns) | TNS (ns) | Failing Endpoints (TNS) | Total Endpoints (TNS) | Path Req (WNS) | Edges (WHS) | WHS (ns) | THS (ns) | Failing Endpoints (THS) | Total Endpoints (THS) | Path Req (WHS) | Clock Pair Classification | Inter-Clock Constraints |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | counter_reg[0]/Q | counter_reg[0]/Q | rise - rise | 16.661 | 0.000 | 0 | 62 | 20.000 | rise - rise | 0.122 | 0.000 | 0 | 62 | 0.000 | Clean | Timed |
| 2 | counter_reg[0]/Q | counter_reg[1]/Q | rise - rise | 998.821 | 0.000 | 0 | 5 | 1000.000 | | | | 0 | 5 | | Ignored | Max Delay Datapath Only |
| 3 | counter_reg[0]/Q | sys_clk_pin | rise - rise | 6.406 | 0.000 | 0 | 2 | 10.000 | rise - rise | 0.838 | 0.000 | 0 | 2 | 0.000 | Clean | Timed |
| 4 | counter_reg[1]/Q | counter_reg[0]/Q | rise - rise | 998.675 | 0.000 | 0 | 5 | 1000.000 | | | | 0 | 5 | | Ignored | Max Delay Datapath Only |
| 5 | counter_reg[1]/Q | counter_reg[1]/Q | rise - rise | 36.554 | 0.000 | 0 | 44 | 40.000 | rise - rise | 0.114 | 0.000 | 0 | 44 | 0.000 | Clean | Timed |
| 6 | counter_reg[1]/Q | sys_clk_pin | rise - rise | 6.272 | 0.000 | 0 | 1 | 10.000 | rise - rise | 0.890 | 0.000 | 0 | 1 | 0.000 | Clean | Timed |

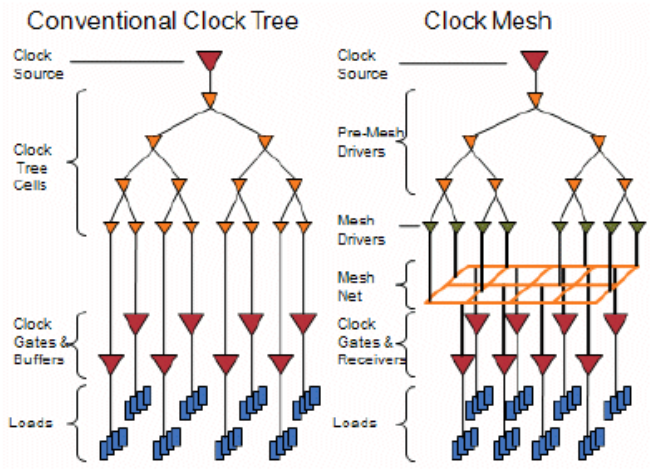
*Clock Interaction Report*




*FIFO Post-Implementation Timing Simulation*

# 5. Research

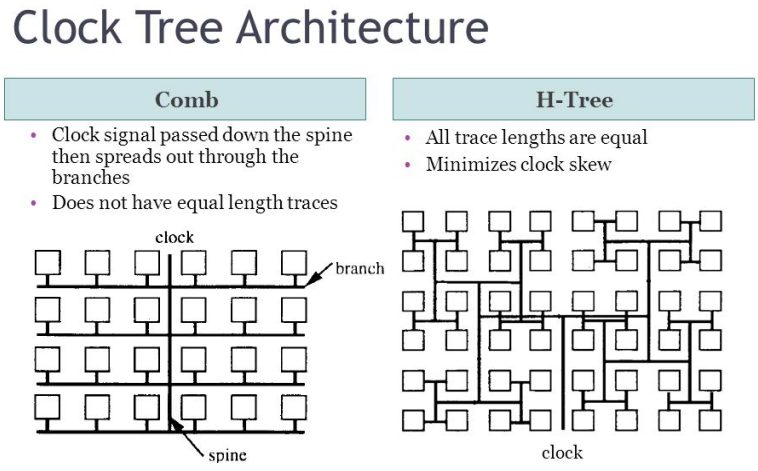
**Clock Tree Synthesis (CTS)** aims to distribute the clock signal from a single clock source to all clocked elements in a design while minimizing skew and maintaining signal integrity.




*CTS Figure*[1]

A clock tree is a hierarchical network of buffers, inverters, and interconnect wires that distribute the clock signal. It starts at the clock source and branches out to clock sinks. The topology can be:

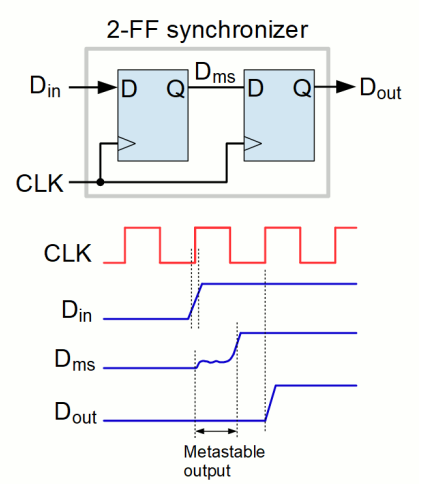
- H-Tree: Balanced tree structure for equal latency.
- Spine or Mesh: Grid-based clock distribution, typically used in high-frequency designs.
- Buffered Tree: Uses buffers at branching points to strengthen the clock signal.




*Clock Tree Architecture*[2]

**Dual FF Synchronizer:**

Clock Domain Crossing (CDC) occurs when signals pass from one clock domain to another domain with a different clock frequency or phase relationship. CDC introduces the risk of metastability, where a flip-flop enters an unstable state and delays the signal propagation. The **dual flip-flop synchronizer** is a common technique used to address metastability. It ensures that data crossing clock domains stabilizes before it is used in the new clock domain.




*Dual FF Figure with Clocks & Outputs*[3]

- **First Flip-Flop:** Captures the signal from the source clock domain and allows metastability to resolve over one clock cycle.
- **Second Flip-Flop:** Further stabilizes the signal, ensuring that it is safe to use in the destination clock domain.
- The two flip-flops act as a buffer, allowing metastability to settle while ensuring signal integrity.

# References

1- https://www.design-reuse.com/articles/21019/clock-mesh-benefits-analysis.html
2- https://slideplayer.com/slide/4442868/
3- https://commons.wikimedia.org/wiki/File:2FF_synchronizer.gif