

**DIGITAL SYSTEM DESIGN APPLICATION**

**EHB436E CRN: 11280**

**Salih Ömer Ongün**

**040220780**

**Experiment 6**

**CLOCK GENERATION**

**Design Source**

**module** clock\_gen

**(**

**input** clk**,**

**input** rst**,**

**output** **[**6**:**0**]** cnt100**,**

**output** **[**6**:**0**]** cnt80**,**

**output** **[**6**:**0**]** cnt60

**);**

**wire** clk80**;**

**wire** clk60**;**

**reg** **[**6**:**0**]** cnt100\_reg**;**

**reg** **[**6**:**0**]** cnt80\_reg**;**

**reg** **[**6**:**0**]** cnt60\_reg**;**

clk\_wiz\_0 instance\_name

**(**

// Clock out ports

**.**clk\_out1**(**clk80**),** // output clk\_out1 80MHz

**.**clk\_out2**(**clk60**),** // output clk\_out2 60MHz

// Status and control signals

**.**reset**(**rst**),** // input reset

**.**locked**(**locked**),** // output locked

// Clock in ports

**.**clk\_in1**(**clk**)** // input clk\_in1

**);**

**always** **@(posedge** clk**)** **begin**

**if(**rst**==**1'b1**)** **begin**

cnt100\_reg **<=** 7'b0**;**

**end**

**else** **if(**locked **==**1'b1**)** **begin**

**if(**cnt100\_reg **<** 100**)** **begin**

cnt100\_reg **<=** cnt100\_reg **+** 1**;**

**end**

**else** **begin**

cnt100\_reg **<=** cnt100\_reg**;**

**end**

**end**

**end**

**assign** cnt100 **=** cnt100\_reg**;**

**always** **@(posedge** clk80**)** **begin**

**if(**rst**==**1'b1**)** **begin**

cnt80\_reg **<=** 7'b0**;**

**end**

**else** **if(**locked **==**1'b1**)** **begin**

**if(**cnt80\_reg **<** 80**)** **begin**

cnt80\_reg **<=** cnt80\_reg **+** 1**;**

**end**

**else** **begin**

cnt80\_reg **<=** cnt80\_reg**;**

**end**

**end**

**end**

**assign** cnt80 **=** cnt80\_reg**;**

**always** **@(posedge** clk60**)** **begin**

**if(**rst**==**1'b1**)** **begin**

cnt60\_reg **<=** 7'b0**;**

**end**

**else** **if(**locked **==**1'b1**)** **begin**

**if(**cnt60\_reg **<** 60**)** **begin**

cnt60\_reg **<=** cnt60\_reg **+** 1**;**

**end**

**else** **begin**

cnt60\_reg **<=** cnt60\_reg**;**

**end**

**end**

**end**

**assign** cnt60 **=** cnt60\_reg**;**

**endmodule**

**Simulation Source**

**module** clock\_gen\_tb**();**

**reg** CLK **=** 1'b0**;**

**reg** RST **=** 1'b0**;**

**wire** **[**6**:**0**]** CNT100**;**

**wire** **[**6**:**0**]** CNT80**;**

**wire** **[**6**:**0**]** CNT60**;**

clock\_gen uut

**(**

**.**clk**(**CLK**),**

**.**rst**(**RST**),**

**.**cnt100**(**CNT100**),**

**.**cnt80**(**CNT80**),**

**.**cnt60**(**CNT60**)**

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;**

**end**

**initial** **begin**

RST **=** 1**;**

**#**50**;**

RST **=** 0**;**

**#**2000**;**

$finish**;**

**end**

**endmodule**

**ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturulduSimulation Wave**

As can be seen in the image, our code is working correctly. The cnt100 counter runs on the rising edge of the clk clock. The cnt60 counter runs on the rising edge of the clk60 clock. The cnt80 counter runs on the rising edge of the clk80 clock. When cnt100 reaches 100, the counter stops. When cnt60 reaches 60, the counter stops. When cnt80 reaches 80, the counter stops.

**metin, diyagram, plan, şematik içeren bir resim

Açıklama otomatik olarak oluşturulduRTL Schematic**

**diyagram, plan, şematik, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturulduTechnology Schematic**

**diyagram, plan, teknik çizim, şematik içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Utilization Report**

**metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Clock Network Report**

**ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

We use 100MHz, 80MHz and 60MHz clocks.

**CLOCK GATING**

**Design Source**

**module** clock\_gating

**(**

**input** clk**,**

**input** rst**,**

**output** **[**6**:**0**]** cnt50**,**

**output** **[**6**:**0**]** cnt25

**);**

**reg** clk50\_en**,**clk25\_en**;**

**wire** clk50**,**clk25**;**

**reg** **[**6**:**0**]** cnt50\_reg**;**

**reg** **[**6**:**0**]** cnt25\_reg**;**

BUFR **#(** // 50MHz

**.**BUFR\_DIVIDE**(**"2"**),**

**.**SIM\_DEVICE**(**"7SERIES"**)**

**)**

BUFR\_inst50 **(**

**.**O**(**clk50**),**

**.**CE**(**clk50\_en**),**

**.**CLR**(**1'b0**),**

**.**I**(**clk**)**

**);**

BUFR **#(** // 25MHz

**.**BUFR\_DIVIDE**(**"4"**),**

**.**SIM\_DEVICE**(**"7SERIES"**)**

**)**

BUFR\_inst25 **(**

**.**O**(**clk25**),**

**.**CE**(**clk25\_en**),** // 1-bit input: Active high, clock enable (Divided modes only)

**.**CLR**(**1'b0**),** // 1-bit input: Active high, asynchronous clear (Divided modes only)

**.**I**(**clk**)**

**);**

**always** **@(posedge** clk50**,** **posedge** rst**)** **begin**

**if(**rst**==**1'b1**)** **begin**

clk50\_en **<=** 1'b1**;**

cnt50\_reg **<=** 7'b0**;**

**end**

**else** **if(**clk50\_en **==** 1'b1**)begin**

**if(**cnt50\_reg **<** 50**)** **begin**

cnt50\_reg **<=** cnt50\_reg **+** 1**;**

**end**

**else** **begin**

clk50\_en **<=** 1'b0**;**

**end**

**end**

**end**

**assign** cnt50 **=** cnt50\_reg**;**

**always** **@(posedge** clk25**,** **posedge** rst**)** **begin**

**if(**rst**==**1'b1**)** **begin**

clk25\_en **<=** 1'b1**;**

cnt25\_reg **<=** 7'b0**;**

**end**

**else** **if(**clk25\_en **==** 1'b1**)begin**

**if(**cnt25\_reg **<** 25**)** **begin**

cnt25\_reg **<=** cnt25\_reg **+** 1**;**

**end**

**else** **begin**

clk25\_en **<=** 1'b0**;**

**end**

**end**

**end**

**assign** cnt25 **=** cnt25\_reg**;**

**endmodule**

**Simulation Source**

**module** clock\_gating\_tb**();**

**reg** CLK **=** 1'b0**;**

**reg** RST **=** 1'b0**;**

**wire** **[**6**:**0**]** CNT50**;**

**wire** **[**6**:**0**]** CNT25**;**

clock\_gating uut

**(**

**.**clk**(**CLK**),**

**.**rst**(**RST**),**

**.**cnt50**(**CNT50**),**

**.**cnt25**(**CNT25**)**

**);**

**always** **begin**

**#**5 CLK **=** **~**CLK**;**

**end**

**initial** **begin**

RST **=** 1**;**

**#**105**;**

RST **=** 0**;**

**#**800**;**

RST **=** 1**;**

**#**105**;**

RST **=** 0**;**

$finish**;**

**end**

**endmodule**

ekran görüntüsü, çizgi, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**Simulation Wave**

As can be seen in the image, our code is working correctly. Our internal 100MHz clock is available in clk. External clk50(50MHz) and clk25(25MHz) work correctly. The cnt50 counter runs on the rising edge of the clk50 clock. The cnt25 counter runs on the rising edge of the clk25 clock. If we give the rst signal the counters and enable signals are reset. However, the clocks continue to run.

ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturulduAfter the rst signal, counters start counting.

If counters reach their final value, enable signal of clocks have logic zero value. Therefore, counters stop counting and clocks stop.

diyagram, plan, şematik, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu**Technology Schematic**

**Utilization Reports**

**metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Clock Networks**

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

We use 50MHz and 25MHz clocks.

**BLOCK RAM**

**Design Source**

**module** block\_ram

**(**

**input** clka**,**

**input** wea**,**

**input** **[**3**:**0**]** addra**,**

**input** **[**7**:**0**]** dina**,**

**output** **[**7**:**0**]** douta

**);**

**wire** clk\_50**;**

blk\_mem\_gen\_0 your\_instance\_name

**(**

**.**clka**(**clk\_50**),** // input wire clka

**.**wea**(**wea**),** // input wire [0 : 0] wea

**.**addra**(**addra**),** // input wire [3 : 0] addra

**.**dina**(**dina**),** // input wire [7 : 0] dina

**.**douta**(**douta**)** // output wire [7 : 0] douta

**);**

clk\_wiz\_2 instance\_name

**(**

// Clock out ports

**.**clk\_50**(**clk\_50**),** // output clk\_50

// Status and control signals

**.**reset**(**reset**),** // input reset

**.**locked**(**locked**),** // output locked

// Clock in ports

**.**clk\_in1**(**clka**)** // input clk\_in1

**);**

**endmodule**

**Simulation Source**

**module** block\_ram\_tb**();**

**reg** clka **=** 1'b0**;**

**reg** wea **=** 1'b0**;**

//reg ena = 1'b1;

**reg** **[**3**:**0**]** addra **=** 4'b0**;**

**reg** **[**7**:**0**]** dina **=** 8'b0**;**

**wire** **[**7**:**0**]** douta**;**

**integer** i**;**

**reg** **[**7**:**0**]** vary **[**15**:**0**];**

//reg [7:0] vary;

//reg [7:0] vary2;

block\_ram uut

**(**

**.**clka**(**clka**),**

**.**wea**(**wea**),**

**.**addra**(**addra**),**

**.**dina**(**dina**),**

**.**douta**(**douta**)**

**);**

**always** **begin**

**#**5 clka **=** **~**clka**;**

**end**

**initial** **begin**

$display**(**"we are reading"**);**

**#**300**;** // ilk baslangcta LOCKED oluyor

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

addra **=** i**;**

**#**50**;**

$display**(**"Address: %d, Data Read: %h"**,** addra**,** douta**);**

**end**

**#**10

$display**(**"we are writing to array"**);**

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

addra **=** i**;**

**#**50**;**

vary**[**15**-**i**]** **=** douta**;**

$display**(**"array'e yaz Address: %d, Data Output: %h,Vary: %h"**,** addra**,** douta**,**vary**[**15**-**i**]);**

**end**

**#**10**;**

$display**(**"we are writing to address"**);**

wea **=** 1'b1**;**

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

addra **=** i**;**

dina **=** vary**[**i**];**

**#**50

$display**(**"Address: %d, Data Input: %h, Vary: %h, Data Output: %h"**,** addra**,** dina**,**vary**[**i**],**douta**);**

**end**

$display**(**"we are reading number backwards"**);**

wea **=** 1'b0**;**

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

addra **=** i**;**

**#**50**;**

$display**(**"Address: %d, Data Read: %h"**,** addra**,** douta**);**

**end**

$finish**;**

**end**

**endmodule**

**Simulation TCL Console**

**metin, ekran görüntüsü, doküman, belge, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

First, we read the data filled from the memory.coe file. It read my numbers in the correct order. Then, I created an array and created the data that the block ram wrote to the array in reverse order. For example, the last index of the array has the first index data of the block ram.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Then the data in the array is written to the block ram address in order. My ID number has been written to block ram in reverse order. Finally, ıt read value in block ram, the values is inverse order of my student ID.

**Technology Schematic**

**metin, diyagram, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Utilization Reports**

**metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**FIFO**

**Design Source**

**module** FIFO

**(**

**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

**);**

**wire** clk\_50**,**clk\_25**;**

fifo\_generator\_0 your\_instance\_name

**(**

**.**wr\_clk**(**clk\_50**),** // input wire wr\_clk

**.**wr\_rst**(**rst**),** // input wire wr\_rst

**.**rd\_clk**(**clk\_25**),** // input wire rd\_clk

**.**rd\_rst**(**rst**),** // input wire rd\_rst

**.**din**(**din**),** // input wire [7 : 0] din

**.**wr\_en**(**wr\_en**),** // input wire wr\_en

**.**rd\_en**(**rd\_en**),** // input wire rd\_en

**.**dout**(**dout**),** // output wire [7 : 0] dout

**.**full**(**full**),** // output wire full

**.**overflow**(**overflow**),** // output wire overflow

**.**empty**(**empty**),** // output wire empty

**.**underflow**(**underflow**)** // output wire underflow

**);**

clk\_wiz\_3 instance\_name

**(**

// Clock out ports

**.**clk\_50**(**clk\_50**),** // output clk\_50

**.**clk\_25**(**clk\_25**),** // output clk\_25

// Status and control signals

**.**reset**(**reset**),** // input reset

**.**locked**(**locked**),** // output locked

// Clock in ports

**.**clk\_in1**(**clk**)** // input clk\_in1

**);**

**endmodule**

**Simulation Source**

**module** FIFO\_tb**();**

**reg** clk **=** 1'b0**;**

**reg** rst **=** 1'b0**;**

**reg** wr\_en **=** 1'b0**;**

**reg** rd\_en **=** 1'b0**;**

**reg** **[**7**:**0**]** din **=** 1'b0**;**

**wire** **[**7**:**0**]** dout**;**

**wire** empty**;**

**wire** full**;**

**wire** overflow**;**

**wire** underflow**;**

**integer** i **=** 0**;**

FIFO uut

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**wr\_en**(**wr\_en**),**

**.**rd\_en**(**rd\_en**),**

**.**din**(**din**),**

**.**dout**(**dout**),**

**.**empty**(**empty**),**

**.**full**(**full**),**

**.**overflow**(**overflow**),**

**.**underflow**(**underflow**)**

**);**

**always** **begin**

**#**5 clk **=** **~**clk**;**

**end**

**initial** **begin**

**#**200**;**

rst **=** 1'b1**;**

**#**20**;**

rst **=** 1'b0**;**

**#**350**;**

**while(**i**<**65**)** **begin**

wr\_en **=** 1'b1**;**

din **=** i**;**

$display**(**"Data Write: %d"**,**din**);**

rd\_en **=** 1'b1**;**

i**=**i**+**1**;**

**#**10**;**

$display**(**"Data Read: %d"**,**dout**);**

**end**

**end**

**endmodule**

ekran görüntüsü, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**Simulation Wave**

ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

Açıklama otomatik olarak oluşturulduSimulation start to write data.

ekran görüntüsü, multimedya yazılımı içeren bir resim

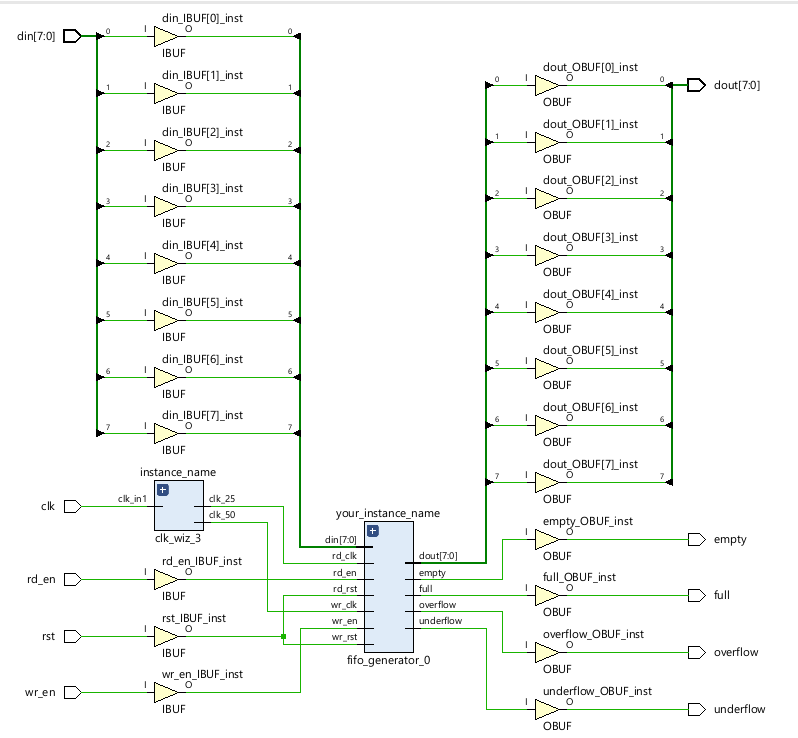
Açıklama otomatik olarak oluşturulduWrite clock has 50MHz and read clock 25MHz frequency. Therefore, writing is faster than reading. Writing reach final value but reading does not.

Writing reach final value but reading does not. Therefore, full flag has logic 1 value. Simulation write data to FIFO although full has logic 1 value. Therefore, overflow flag has logic 1 value.

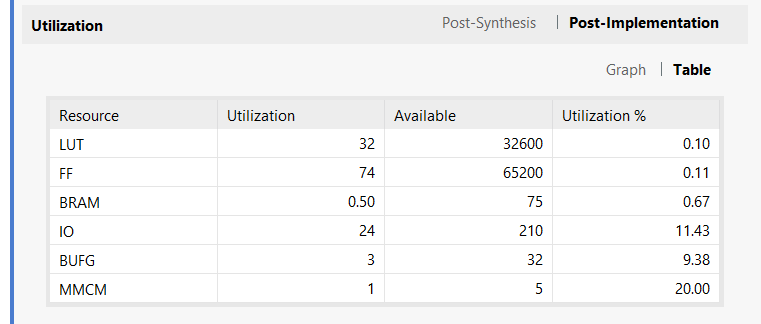
ekran görüntüsü, multimedya yazılımı, yazılım, renklilik içeren bir resim

Açıklama otomatik olarak oluşturulduFinally, reading reach final value but it can not read some intermediate value.

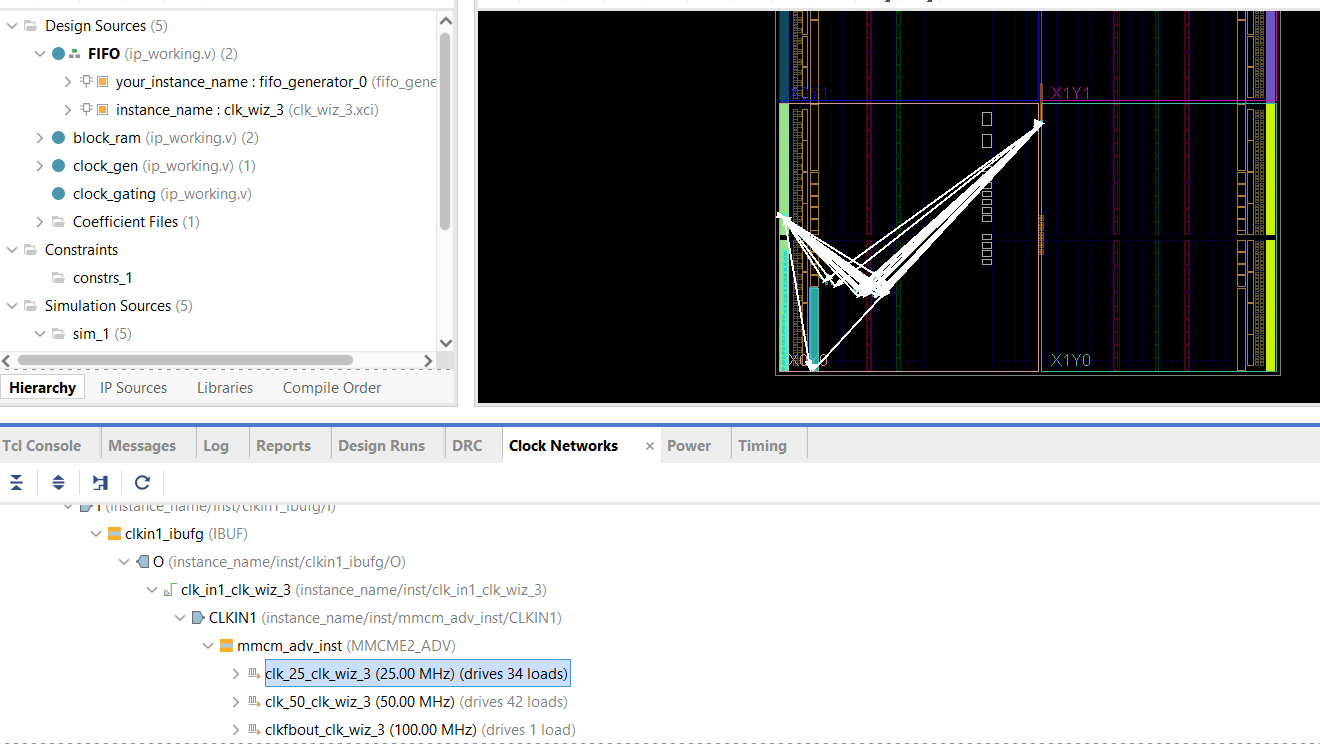
**RTL Schematic**

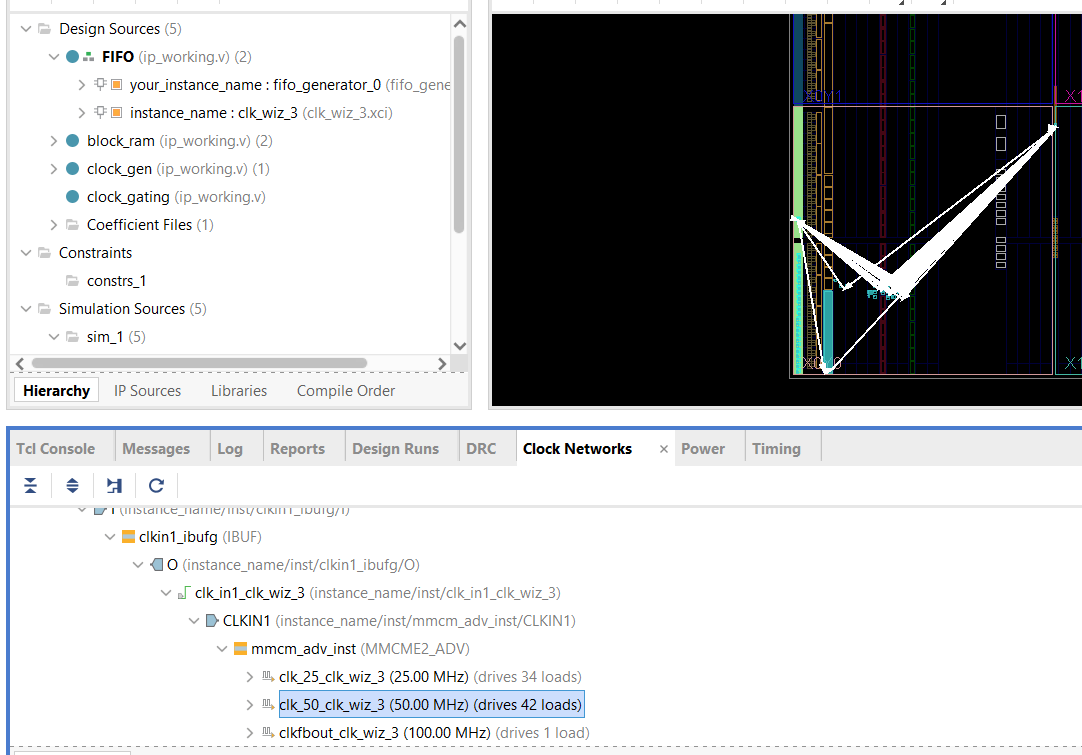


**Utilization Report**



**Clock Network Reports**





We use 50MHz and 25MHz clocks.

**Clock Network Interaction**



**Post-Implementation Timing Simulation**

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

**RESEARCH**

1. Clock Tree Synthesis is one of most important technique in digital desgin. It aims distribute clock signal balanced. It insert buffers/inverters along the clock routes of an ASIC design. ***[1]***

Clock Latency : Clock latency refers to the arrival time of the clock signal from the clock source to flip flops. The clock latency from the port up to the clock pin is referred to as the network latency.

Clock Skew : Clock Skew refers to the difference in the clock arrival time between two registers.

Clock Slew (or transition time): The time that a given signal takes to rise from a level of 10% of the rail voltage to the level of 90% of the rail voltage is referred to as rise slew. Similarly, the time that a given signal takes to fall from a level of 90% of the rail voltage to the level of 10% of the rail voltage is referred to as fall slew**.*[2]***

Clock Jitter: Normally, clocks have specific period. However, noise, temperature and other effects cause disruption in period of clock cycle. It is called clock jitter.

Conventional CTS/Single point CTS:  
Single point CTS is the default choice for most of the designers having lower frequency & lesser no of sinks. As name suggested having single clock source which distribute clock to every corner of design.

çizgi, diyagram, origami, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Clock Mesh Structure  
As the name suggests it create a dense mesh of shorted wires which is being driven by mesh drivers to distribute clock in every corner of the design.  
The output of all the mesh drivers will be shorted using a metal mesh, which will carry the clock signal across the block using horizontal and vertical metal stripes

diyagram, çizgi, metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu ***[3]***

1. If the signal is not in a stable state at a certain time before and after the clock signal, the problem of metastability occurs. One of the methods to solve metastability is the Double Flop Synchroniser method. This method consists of two cascaded D flip-flops that sample the input signal with the destination clock. Flip-flops work as buffers to solve the metastability problem. Signals that cannot reach a stable state in the first flip-flop become stable due to the delay that occurs until they reach the second flip-flop. Therefore, it can solve metastability problem. [4]

metin, diyagram, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

***[5]***

**REFERENCE**

**[1]** B, D. M. (2024, March 27). *What is Clock Tree Synthesis?* ChipEdge VLSI Training Company. https://chipedge.com/what-is-clock-tree-synthesis/

**[2]** Anysilicon. (2022, September 24). *Ultimate Guide: Clock Tree Synthesis - AnySilicon*. AnySilicon. https://anysilicon.com/clock-tree-synthesis/

**[3]** Abhishek. (n.d.). *Different types of clock tree structure*. https://vlsiconceptsforyou.blogspot.com/2020/07/different-types-of-clock-tree-structure.html?m=1

**[4]** Satheesh, M. (2024, April 3). *Double Flip-Flop Synchronizer for CDC*. Digital System Design. https://digitalsystemdesign.in/double-flip-flop-synchronizer-for-cdc/?srsltid=AfmBOooDFmO3E6dyGGnOqOVNQIvss2JmlYAyPtwuCR7QY3uE-IxriopT

**[5]** *File:2FF Synchronizer.gif - Wikimedia Commons*. (2020, January 18). https://commons.wikimedia.org/wiki/File:2FF\_synchronizer.gif