

بسمه تعالی



دانشگاه صنعتی شریف  
دانشکده مهندسی کامپیوتر

# پروژه طراحی سیستم های دیجیتال

اعضای گروه:

امیرحسن جعفرآبادی

(۴۰۰۱۰۴۸۷۸)

سید علی جعفری

(۴۰۰۱۰۴۸۸۹)

سید علی طیب

(۴۰۰۱۰۵۲۶)

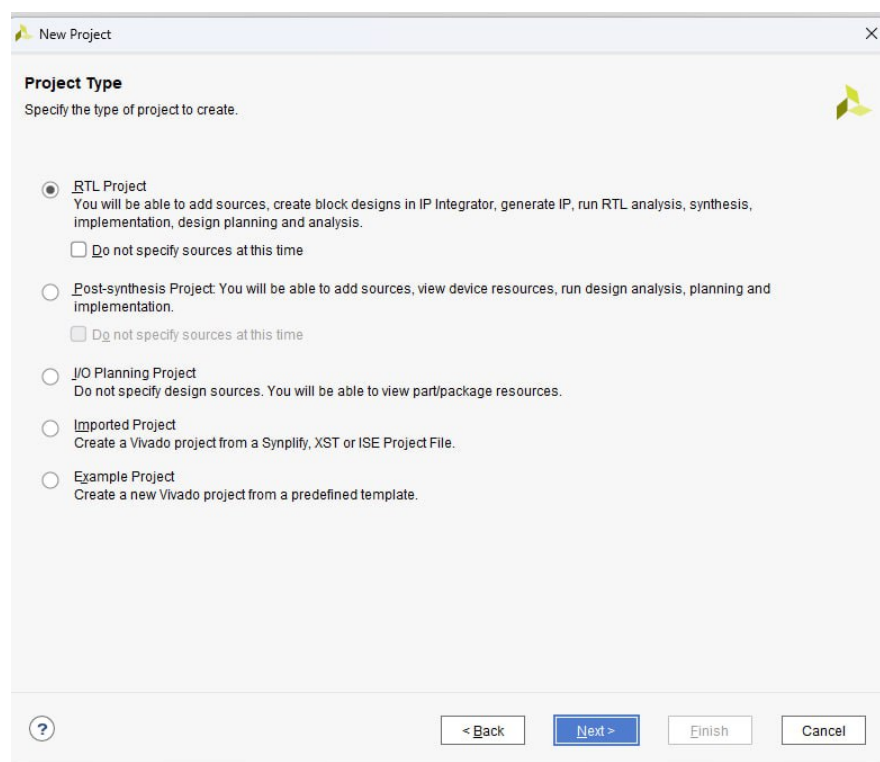
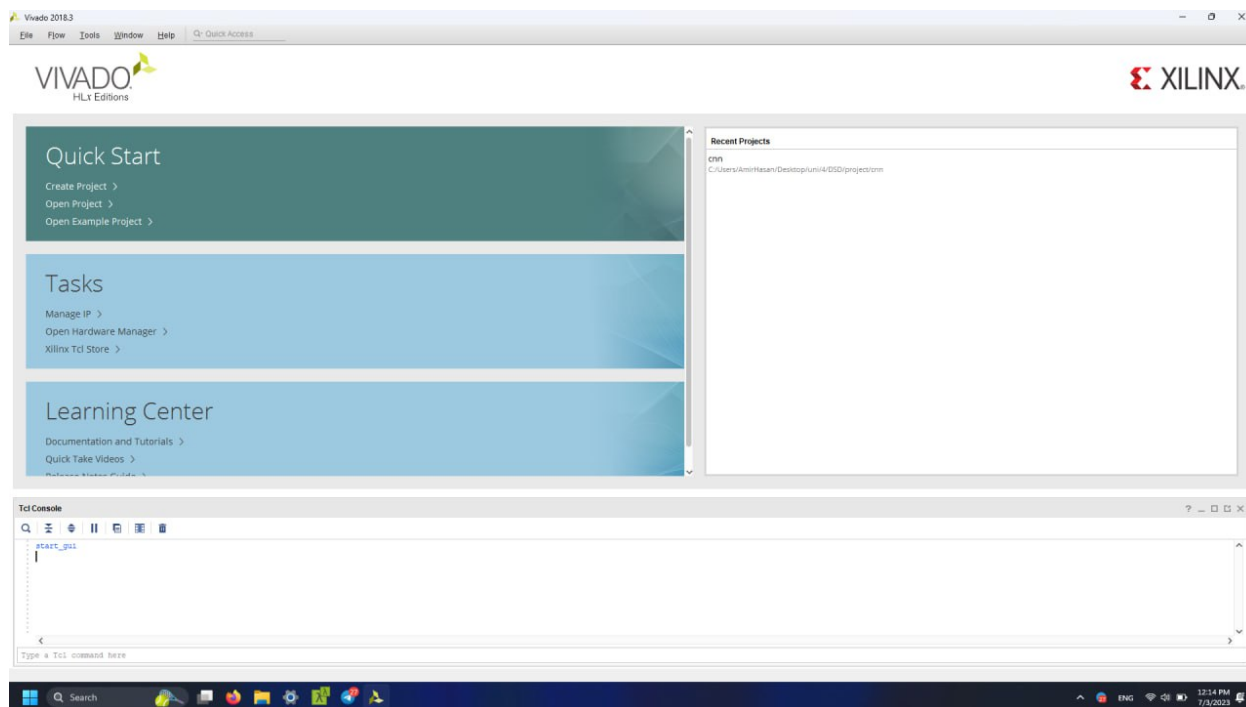
استاد:

امین فصحتی

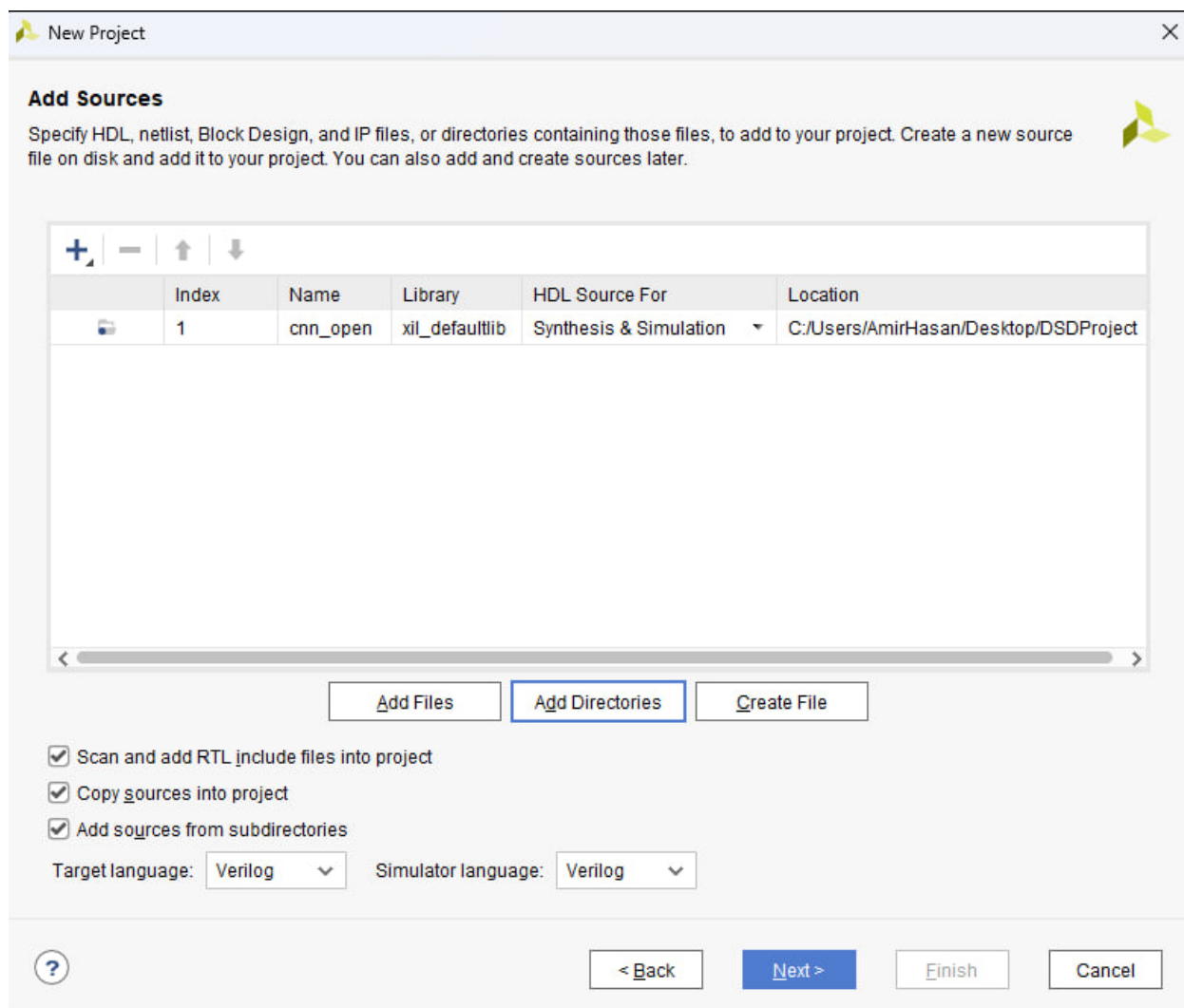
بهار ۱۴۰۲

# 1- شبیه سازی

برای شبیه سازی و سنتز پروژه در نرم افزار vivado باید ابتدا یک پروژه ی خالی از نوع RTL ایجاد کنیم.




سپس در بخش add sources با استفاده از add directory فولدر cnn\_open ریپازیتوری گیت را اضافه می کنیم.



**Add Sources**


Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location
	1	cnn_open	xil_defaultlib	Synthesis & Simulation	C:/Users/AmirHasan/Desktop/DSDProject

☒ Scan and add RTL include files into project  
☒ Copy sources into project  
☒ Add sources from subdirectories


Target language: Verilog Simulator language: Verilog

در بخش انتخاب FPGA مدل xkcu040-ffva1156-2-e را انتخاب می کنیم که مدل استفاده شده در برد KCU105 می باشد.


New Project
×

### Default Part

Choose a default Xilinx part or board for your project.



**Parts** | Boards

[Reset All Filters](#)

Category:
All

Package:
All

Temperature:
All

Family:
All

Speed:
All

Search:
×
(7 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs
xcku040-ffva1156-3-e	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-2-e	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-2-i	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-1-c	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-1-i	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-1L-i	1156	520	242400	484800	600	0	1920
xcku040-ffva1156-1LV-i	1156	520	242400	484800	600	0	1920

<
>

?

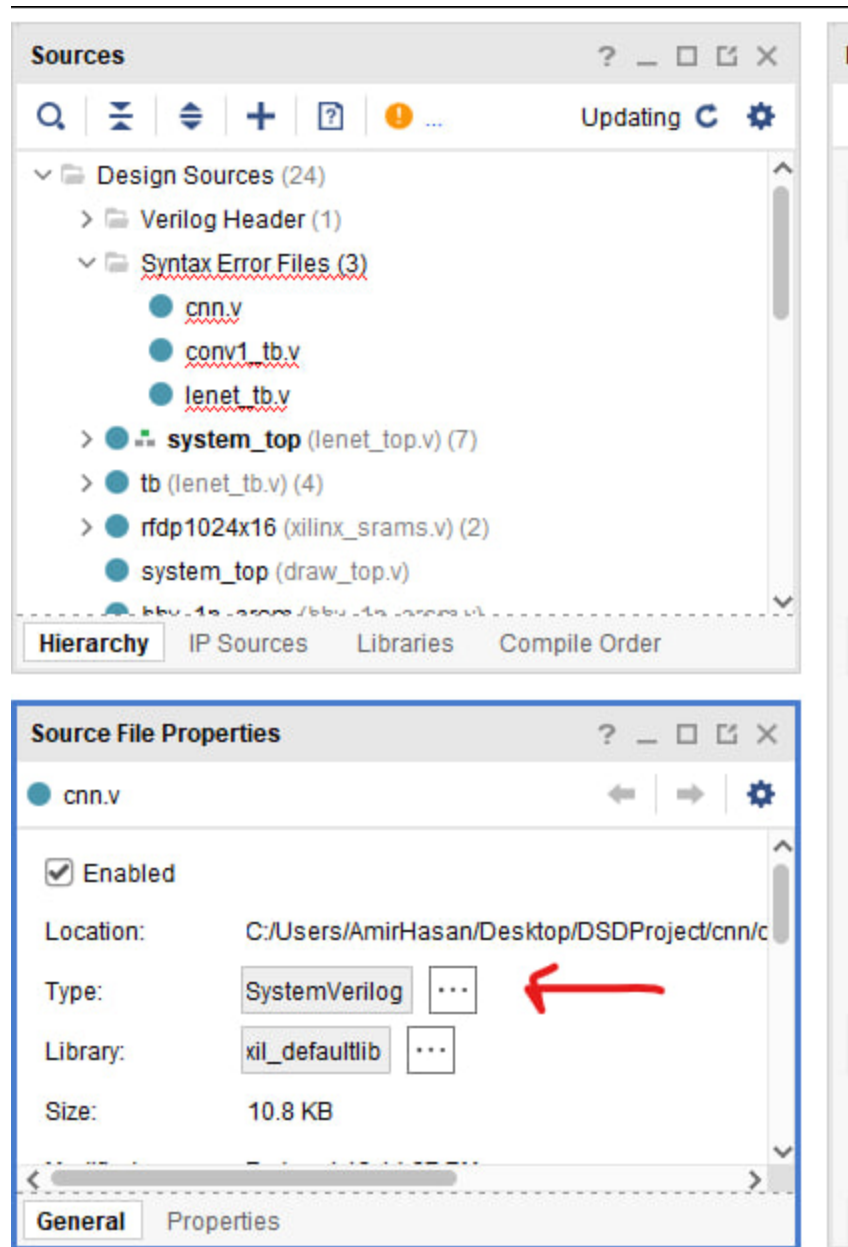
< Back

Next >

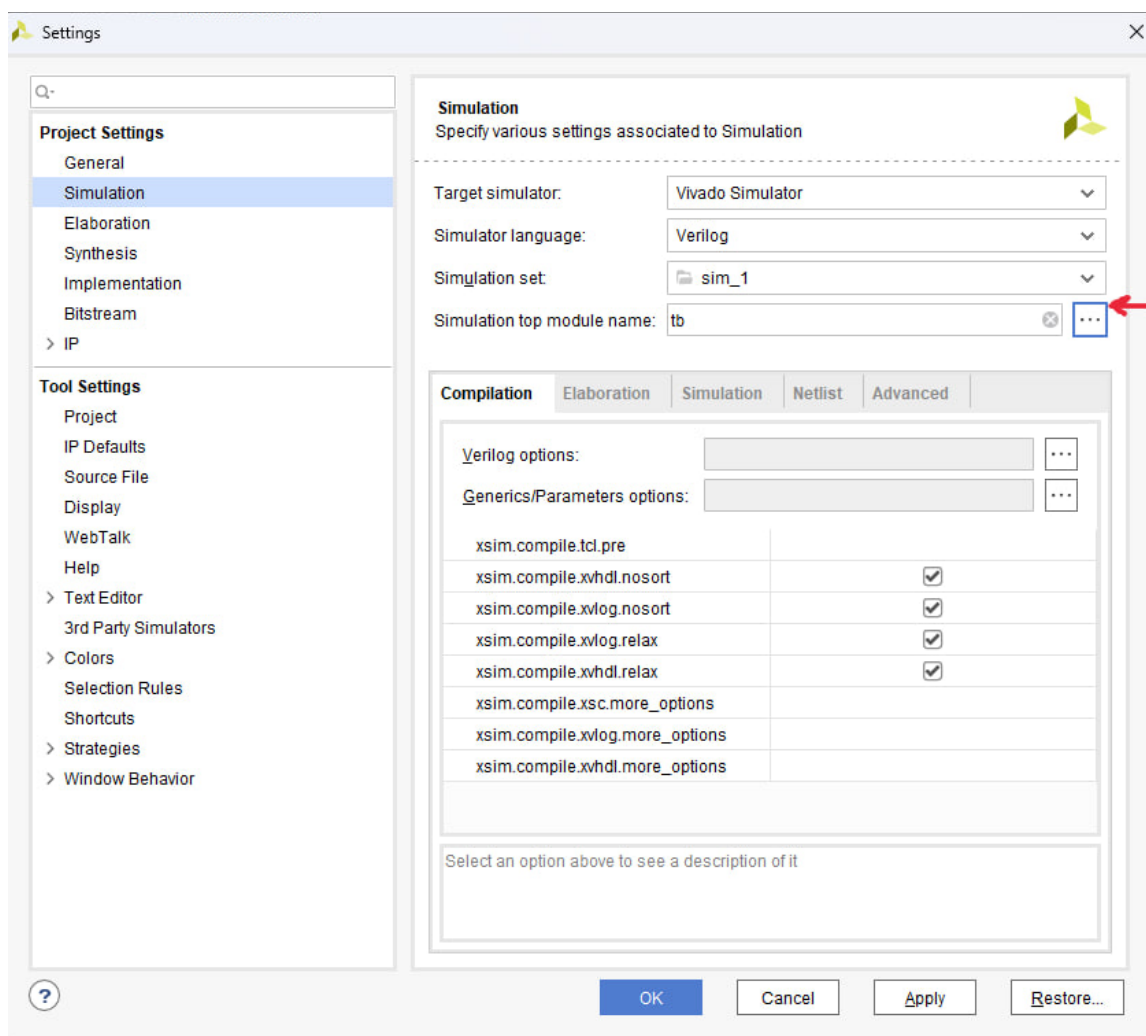
Finish

Cancel

پس از ساخته شدن پروژه چند ارور وجود دارند که به دلیل این است که vivado فایل ها را به عنوان verilog خوانده است. ما باید در بخش source file properties نوع فایل ها را به system verilog تغییر دهیم.



برای شبیه‌سازی در تنظیمات شبیه‌سازی simulation top module name را تغییر داده و از system top به tb تغییر می‌دهیم. (تصویر مربوطه در صفحه بعد می‌باشد)



پس از لانچ سیمولیشن به دو خطا برمیخوریم. خطای اول دوبار تعریف شدن متغیر cnt\_kx در فایل cnn.v است که با کامنت کردن تعریف دوم بر طرف می شود. خطای دوم اشکال در آدرس فایل test\_1000f.yuv (عکس های ورودی برای تست) در تست بنچ است که با تبدیل آدرس از relative به absolute برطرف می شود. برای بررسی بهتر تعداد تست های اجرا شده را از 20 به 100 تغییر می دهیم. این کار با تغییر ورودی تسک drive\_frame در فایل lenet\_tb انجام می شود. پس از لانچ کردن شبیه سازی و اجرای تست بنچ خروجی های خط به خط مثل خط زیر می گیریم:

T 186618==process a frame 0, digit 7 =====

این خط می گوید که frame اول توسط ماژول عدد 7 تشخیص داده شده است. برای بررسی دقت ماژول باید label های صحیح را داشته باشیم. می توان از فایل PNG test\_900.png که در کنار پروژه قرار داشت استفاده کرد اما برای تسریع کار و محاسبه ی دقت به صورت اتوماتیک و غیر دستی نیاز به label ها به صورت غیر متنی داریم. برای این کار با کد پایتون زیر 1000 برچسب اول را به دست می آوریم.

```
import tensorflow as tf
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
test_labels[:1000]
```

سپس برای محاسبه ی دقت خروجی از کد زیر استفاده می کنیم که خروجی testbench را در output قرار می دهیم و در نهایت درصد دقت چاپ می شود.

```
import pandas as pd
from io import StringIO

output = ""

data = pd.read_csv(StringIO(output), sep = "\s+", header = None)
right = 0
wrong = 0
for index, value in enumerate(data[6].tolist()):
    if value == labels[index]: right = right+1
    else : wrong=wrong+1

print((right/(right+wrong)) *100)
```

## 2- مشخص کردن ماژول های ضرب کننده و جمع کننده

واحد ضرب کننده و جمع کننده در فایل cnn.v موجود است. پس از مشاهده کد و بررسی عملکرد ماژول های مختلف، به این نتیجه رسیدیم که کار جمع کننده را ماژول acc و کار ضرب کننده را ماژول mac انجام می دهد.

## ضرب‌کننده (mac):

ورودی و خروجی های این ماژول در پایین قابل مشاهده است:

```
module mac#(
    parameter INPUT_NUM = 1 // input plane_num
)(
    input                                clk,
    input                                rstn,
    input                                d_en,
    input  [`WDP*INPUT_NUM-1:0]         d,
    input  [`WDP*INPUT_NUM-1:0]         w,
    input                                q_en,
    input                                q_en_b1,
    output  [`WDP*2-1:0]                 q
);
```

همانطور که در تصویر بالا مشاهده می‌شود، پارامتر INPUT\_NUM داریم که به طور پیش فرض 1 است. اما می‌توان آن را هنگام نمونه گیری از این ماژول تغییر داد.

یکسری سیگنال های کلاک و ریست و enable موجود است.

قسمت مهم دو آرایه d و w است. که آرایه ای است که طول آن به اندازه  $WDP * INPUT\_NUM$  است. ( $WDP = WD + 1$ )

WD : تعداد بیت طول داده

در نهایت هم متغیر q داریم که تک بیت خروجی ما خواهد بود. (گویی این ماژول بعد از ضرب نظیر به نظیر بیت های آرایه، همه آنها را باهم جمع می‌کند.)

عملکرد : به طور کلی این ماژول درایه های آرایه d و w را نظیر به نظیر در هم ضرب میکند. و در هر مرحله نتایج را باهم جمع می‌کند.



## جمع‌کننده (acc):

ورودی و خروجی های این ماژول در پایین قابل مشاهده است:

```
module acc #(
    parameter INPUT_NUM = 1,
    parameter WIGHT_SHIFT = 8
)(
    input                                clk,
    input                                rstn,
    input                                go,
    input                                en,
    input                                first_data,
    input                                last_data,
    input                                [^WDP*INPUT_NUM-1:0] data_i,
    input                                [^WD_BIAS:0] bias,
    input                                [^WDP*INPUT_NUM-1:0] weight,
    output reg                            q_en,
    output reg                            [^WD:0] q
);
```

همانطور که در تصویر بالا مشاهده می‌شود، پارامتر INPUT\_NUM داریم که به طور پیشفرض 1 است. همچنین پارامتر WIGHT\_SHIFT داریم که به طور پیشفرض برابر 8 می‌باشد. این پارامتر برای تبدیل نتیجه ضرب به 16 بیت استفاده شده است.

سپس مقداری بیت کلاک و ریست موجود است.

سایر ورودی ها در توضیح عملکرد، توضیح خواهم داد.

عملکرد: در ابتدا این ماژول مقادیر data\_i و weight را در هم ضرب می‌کند. (این کار را با استفاده از ماژول mac انجام می‌دهد)

```

mac #(
    .INPUT_NUM          (INPUT_NUM)
)mac(
    .clk                (clk),
    .rstn               (rstn),
    .d_en               (en_d1),
    .d                   (data_i_d1),
    .w                   (weight_d1),
    .q_en_b1            (q_mac_en_b1),
    .q_en               (q_mac_en),
    .q                   (q_mac)
);

```

تصویر بالا نمونه گرفته شده از ماژول mac است.

به این صورت که متغیر data\_i و weight ابتدا با کمک ماژول mac در هم ضرب می شوند و سپس حاصل آن با bias جمع می گردد.

در نهایت هم مقدار پارامتر WIGHT\_SHIFT اثر داده می شود و حاصل در q قرار می گیرد.

### 3- شبیه سازی با کاهش تعداد بیت ها

با توجه به نتیجه گیری انجام شده در قسمت قبل ابتدا مقدار wd را کاهش دادیم. اما مشاهده شد که دقت پس از این کار از 100 درصد به حدود 10 درصد افت می کند.

فرضیه ی اولیه این بود که پس از تغییر wd باید مقدار wd\_bias را هم تغییر داد. به این منظور wd های مختلف را با wd\_bias های مختلف تست کردیم.

برای انجام سریعتر و دقیق تر این تست از پایتون و اسکریپت های TCL استفاده کردیم.

اسکریپت TCL استفاده شده که پروژه را باز کرده شبیه‌سازی را انجام می‌دهد:

```
cd PROJECT_PATH
```

```
open_project cnn.xpr
```

```
launch_simulation
```

```
run all
```

```
quit
```

اسکریپت پایتون زیر ابتدا فایل global.v را باز می‌کند و متغیر های wd, wdp, wd\_bias, wight\_shift را تغییر می‌دهد. سپس فایل tcl را اجرا می‌کند و نتیجه را بر می‌گرداند.

```
import subprocess

with open(r"GLOBAL.V_PATH","r") as f:
    lines = f.readlines()

for wd in range(12,16):
    for wd_bias in range(wd+4,24):
        lines[13] = r"define WD_BIAS      " + str(wd_bias) + "\n"
        lines[14] = r"define WDP_BIAS      " + str(wd_bias + 1) + "\n"
        lines[15] = r"define WD      " + str(wd) + "\n"
        lines[16] = r"define WDP      " + str(wd + 1) + "\n"
        lines[18] = r"define WIGHT_SHIFT  " + str(wd) + "\n"
        with open(r"GLOBAL.V_PATH","w") as f:
            f.writelines(lines)

        process = subprocess.Popen(r"vivado -mode Tcl -source TCL_PATH",
                                    stdout=subprocess.PIPE,
                                    shell=True)

        print ("wd is " ,wd, "wd_bias is " ,wd_bias, "      ", end = "")
        print([int(i.decode("utf-8").split()[6]) for i in process.stdout.readlines() if i.decode("utf-8").startswith("TPR")])
```

نتیجه ی اجرا:

	wd	wdp	labels	percentage
0	12	16	[1,0,1,3,3,1,1,5,9,2,5,5,3,9,2,1,0,5,5,0]	10.0
1	12	17	[1,9,5,0,1,3,3,7,5,5,5,2,2,9,2,3,9,5,5,3]	15.0
2	12	18	[3,5,1,3,1,9,6,1,5,9,0,3,1,3,2,3,5,2,0,2]	20.0
3	12	19	[1,3,4,3,5,6,3,5,5,2,2,2,7,5,2,7,3,3,9,6]	5.0
4	12	20	[1,2,5,9,5,3,1,2,9,3,5,9,1,2,1,2,9,5,8,5]	15.0
5	12	21	[5,7,5,0,9,1,4,2,2,1,9,3,5,2,0,3,5,2,2,3]	15.0
6	12	22	[3,2,4,5,8,3,1,1,5,5,9,6,1,7,9,2,2,1,9,5]	15.0
7	12	23	[2,5,2,5,2,5,9,1,5,1,2,5,3,2,1,3,1,6,6,1]	10.0
8	13	17	[5,5,0,9,8,7,6,9,6,4,6,9,3,2,0,5,8,5,8,8]	10.0
9	13	18	[7,5,2,1,9,0,6,4,2,3,1,3,2,1,8,0,4,9,8,6]	5.0
10	13	19	[8,9,8,7,1,8,6,7,7,3,1,8,6,2,4,5,4,5,2,2]	5.0
11	13	20	[9,5,7,8,8,0,0,4,1,5,0,0,4,6,3,8,5,3,2,3]	5.0
12	13	21	[8,2,9,1,1,1,4,1,4,3,0,5,9,3,6,8,1,4,6,2]	25.0
13	13	22	[8,3,5,6,8,8,7,4,2,0,6,7,0,8,9,7,4,1,0,7]	0.0
14	13	23	[2,4,1,1,4,0,0,0,2,8,7,8,7,7,8,7,8,1,1,9]	10.0
15	14	18	[5,1,5,0,5,5,1,5,2,5,2,1,5,5,5,2,5,1,5,5]	5.0
16	14	19	[5,1,5,0,5,5,1,5,2,5,2,1,5,5,5,2,5,1,5,5]	5.0
17	14	20	[5,5,5,0,5,5,1,5,2,5,1,1,5,5,5,5,5,1,6,5]	10.0
18	14	21	[5,1,5,0,1,1,2,1,1,5,6,5,5,5,5,6,5,1,6,5]	10.0
19	14	22	[1,5,5,9,5,5,2,5,1,5,4,5,5,5,5,5,1,1,6,5]	5.0
20	14	23	[5,1,5,0,5,5,1,5,2,5,2,5,5,5,5,1,5,1,6,5]	5.0
21	15	19	[7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4]	100.0
22	15	20	[7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4]	100.0
23	15	21	[7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4]	100.0
24	15	22	[7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4]	100.0
25	15	23	[7,2,1,0,4,1,4,9,5,9,0,6,9,0,1,5,9,7,3,4]	100.0

همانطور که در جدول مشخص است در همه ی حالاتی که تعداد بیت های wd کاهش یافته اند دقت اصلا قابل قبول نیست. همچنین تغییر bias تاثیر قابل توجهی ندارد. پس از مطالعه ی کد lenet\_roms.v فهمیدیم که باید دقت ضرب های موجود در این فایل را نیز کاهش دهیم.

برای این منظور کد پایتون زیر نوشته شد. این کد تمام وزن های 16 بیتی را یافته و به تعداد بیت کاهش یافته ی مد نظر ما تبدیل می کند.

```
import re

pattern = re.compile(r'16\d(\d+)')
def replace_number(match,bits):
    decimal_value = int(match.group(1))
    return f"{bits}'d{decimal_value//(2**(16 - bits))}"

with open(r'LENET_ROMS.V_PATH', 'r') as file:
    verilog_code = file.read()

new_verilog_code = re.sub(pattern, lambda x :replace_number(x,15), verilog_code)

with open('lenet_roms_15bit.v', 'w') as file:
    file.write(new_verilog_code)
```

دقت شبیه سازی:

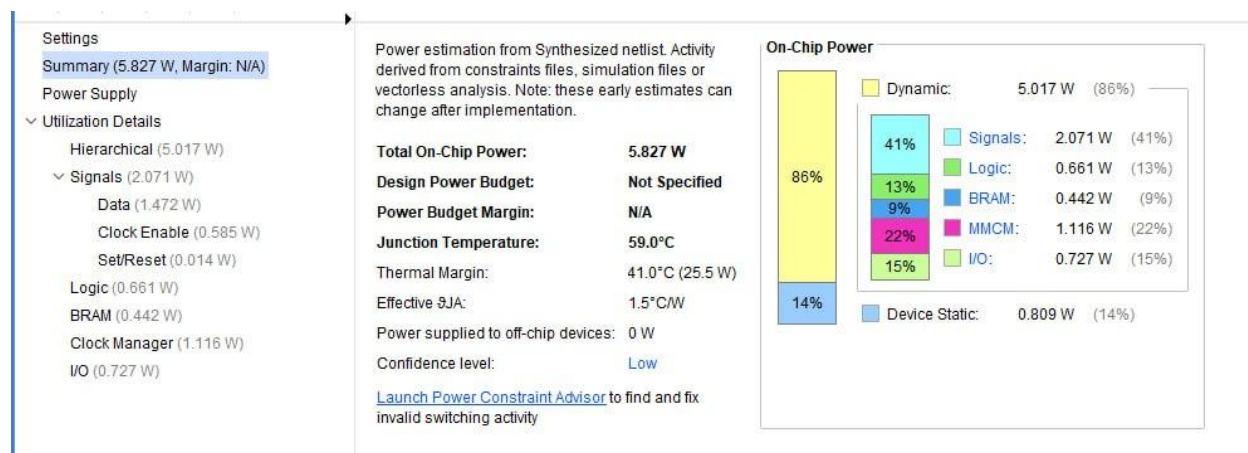
تعداد بیت	دقت
16	100
15	99
14	85
13	15

همانطور که مشاهده می شود می توان با کاهش 1 یا 2 بیت به دقت معقولی رسید اما با کاهش 3 بیت یا بیشتر دیگر قابل استفاده نمیباشد.

## ۴- میزان توان مصرفی و منابع

نرم افزار vivado ابزار قدرتمندی برای طراحی و توسعه سخت افزار های شرکت Xilinx می باشد. یکی از قابلیت هایی که این نرم افزار در اختیار ما قرار می دهد، اندازه گیری میزان توان و منابع مصرفی مدار طراحی شده می باشد. برای پی بردن به این مقادیر گزینه های Report Utilization و Report Power را از زیر منوی SYNTHESIS انتخاب می کنیم.

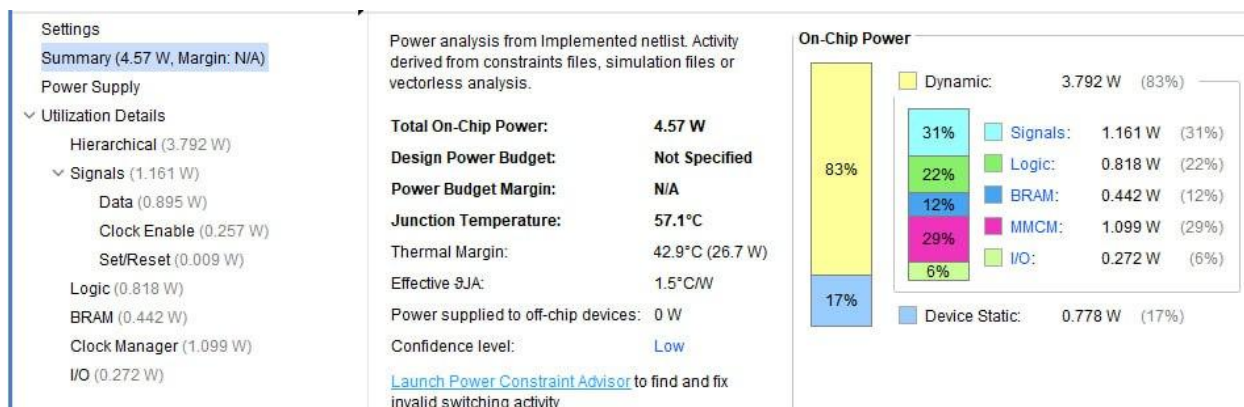
خلاصه ای از مصرف توان در حالت سنتز (Synthesis) را می توانید در عکس زیر مشاهده کنید:



همچنین مصرف منابع در سنتز به شرح زیر است:

Name	CLB LUTs (242400)	CLB Registers (484800)	CARRY8 (30300)	F7 Muxes (121200)	F8 Muxes (60600)	Block RAM Tile (600)	Bonded IOB (520)	GLOBAL CLOCK BUFFERS (480)	MMCME3_ADV (10)
system_top	53374	15132	3627	2376	816	23.5	69	5	1
capture_lenet (capture_lenet)	61	67	2	0	0	0	0	0	0
digit_osd (digit_osd)	84	69	0	4	1	0	0	0	0
dlyRst0 (dlyRst)	9	27	3	0	0	0	0	0	0
draw_rectangle (draw_rectangle)	102	147	2	0	0	1	0	0	0
go_CDC_go_capture_ready (go_CDC_go)	2	4	0	0	0	0	0	0	0
lenet (lenet)	50293	14709	3154	2372	815	22	0	0	0
pll_main (pll_main)	0	0	0	0	0	0	0	3	1
src_buf (rfdp1024x8)	0	0	0	0	0	0.5	0	0	0

اما در حالت پیاده‌سازی (Implementation) مصرف توان را این گونه خروجی گرفتیم:

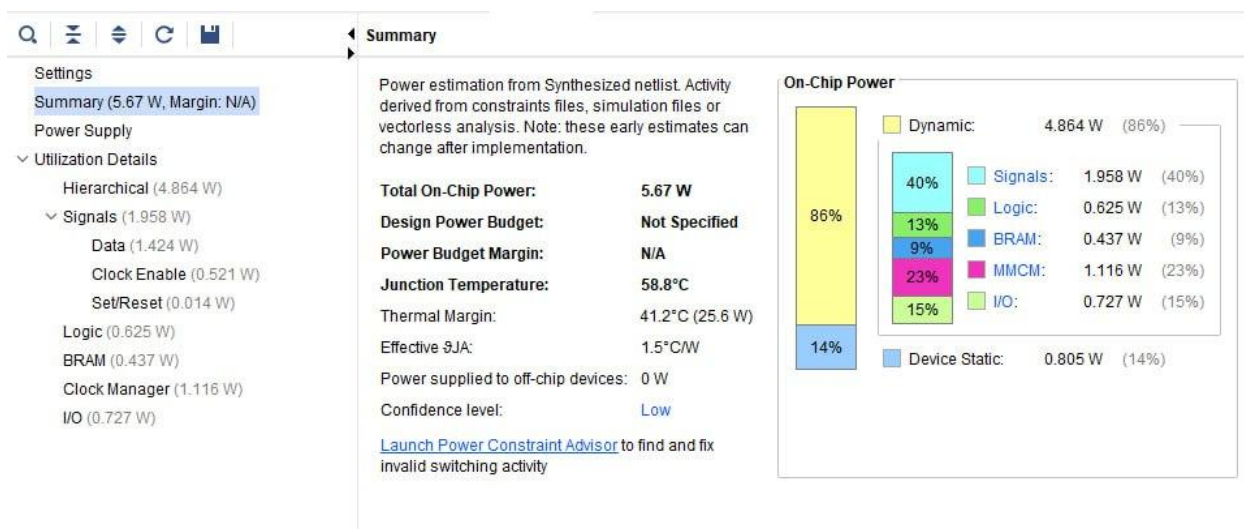


و مصرف منابع در همین حالت، برابر مقادیر زیر بود:

[illegible]

## ۵- میزان توان مصرفی و منابع در طراحی بهینه‌شده:

در حالت **۱۵ بیتی** مانند قسمت ۴ عمل می‌کنیم. در حالت سنتز میزان توان و منابع مصرفی این گونه بود:



Name	CLB LUTs (242400)	CLB Registers (484800)	CARRY8 (30300)	F7 Muxes (121200)	F8 Muxes (60600)	Block RAM Tile (600)	Bonded IOB (520)	GLOBAL CLOCK BUFFERS (480)	MMCME3_ADV (10)
system_top	49883	14433	3119	1743	862	23	69	5	1
capture_ienet (capture_ienet)	61	67	2	0	0	0	0	0	0
digit_osd (digit_osd)	84	69	0	4	1	0	0	0	0
dlyRst0 (dlyRst)	9	27	3	0	0	0	0	0	0
draw_rectangle (draw_rectangle)	102	147	2	0	0	1	0	0	0
go_CDC_go_capture_ready (go_CDC_go)	2	4	0	0	0	0	0	0	0
ienet (ienet)	47667	14010	2882	1739	861	21.5	0	0	0
pll_main (pll_main)	0	0	0	0	0	0	0	3	1
src_buf (rfdp1024x8)	0	0	0	0	0	0.5	0	0	0



و در حالت پیاده‌سازی این گونه بود:

**Settings**

Summary (4.403 W, Margin: N/A)

Power Supply

Utilization Details

- Hierarchical (3.629 W)
  - Signals (1.032 W)
    - Data (0.796 W)
    - Clock Enable (0.231 W)
    - Set/Reset (0.005 W)
  - Logic (0.788 W)
  - BRAM (0.437 W)
  - Clock Manager (1.099 W)
  - I/O (0.272 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

**Total On-Chip Power: 4.403 W**

**Design Power Budget: Not Specified**

**Power Budget Margin: N/A**

**Junction Temperature: 56.8°C**

Thermal Margin: 43.2°C (26.9 W)

Effective  $\theta_{JA}$ : 1.5°C/W

Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

**On-Chip Power**

Category	Power (W)	Percentage (%)
Dynamic	3.629 W	82%
Device Static	0.774 W	18%
Signals	1.032 W	28%
Logic	0.788 W	22%
BRAM	0.437 W	12%
MMCM	1.099 W	30%
I/O	0.272 W	8%

[illegible]

همچنین در حالت **۱۴بیتی** میزان توان و منابع در حالت سنتز اینگونه بود:

Settings

Summary (5.586 W, Margin: N/A)

Power Supply

Utilization Details

Hierarchical (4.783 W)

Signals (1.904 W)

Data (1.366 W)

Clock Enable (0.524 W)

Set/Reset (0.014 W)

Logic (0.603 W)

BRAM (0.433 W)

Clock Manager (1.116 W)

I/O (0.727 W)

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power: 5.586 W

Design Power Budget: Not Specified

Power Budget Margin: N/A

Junction Temperature: 58.6°C

Thermal Margin: 41.4°C (25.7 W)

Effective 9JA: 1.5°C/W

Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

Dynamic: 4.783 W (86%)

40%

13%

9%

23%

15%

Signals: 1.904 W (40%)

Logic: 0.603 W (13%)

BRAM: 0.433 W (9%)

MMCM: 1.116 W (23%)

I/O: 0.727 W (15%)

Device Static: 0.803 W (14%)

[illegible]

و در حالت پیاده‌سازی:

Settings

Summary (4.403 W, Margin: N/A)

Power Supply

Utilization Details

Hierarchical (3.629 W)

Signals (1.043 W)

Data (0.774 W)

Clock Enable (0.26 W)

Set/Reset (0.008 W)

Logic (0.782 W)

BRAM (0.433 W)

Clock Manager (1.099 W)

I/O (0.272 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 4.403 W

Design Power Budget: Not Specified

Power Budget Margin: N/A

Junction Temperature: 56.8°C

Thermal Margin: 43.2°C (26.9 W)

Effective  $\theta_{JA}$ : 1.5°C/W

Power supplied to off-chip devices: 0 W

Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

Dynamic: 3.629 W (82%)

29% Signals: 1.043 W (29%)

22% Logic: 0.782 W (22%)

12% BRAM: 0.433 W (12%)

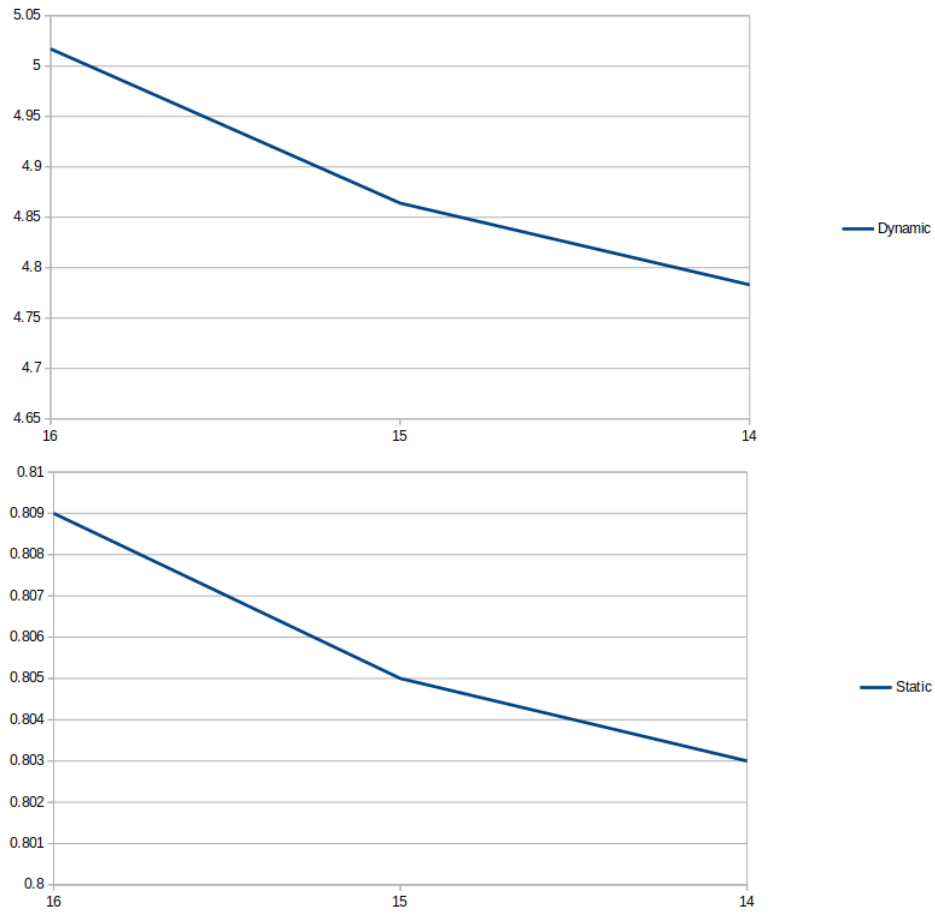
30% MMCM: 1.099 W (30%)

7% I/O: 0.272 W (7%)

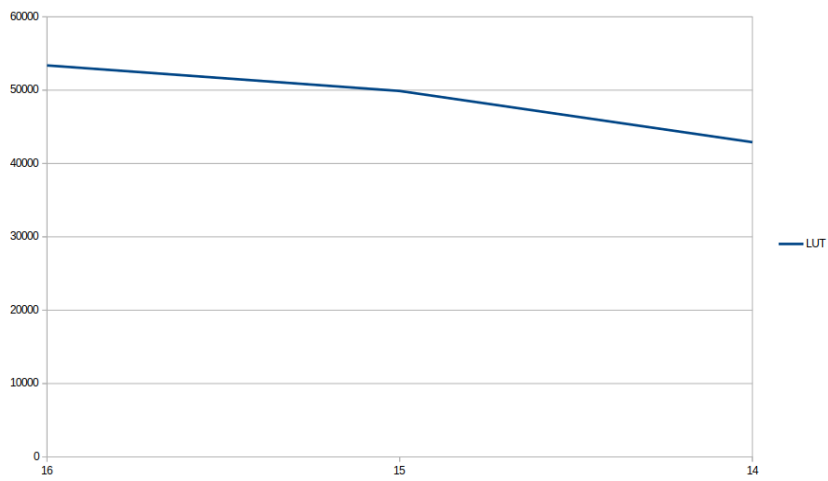
18% Device Static: 0.774 W (18%)

[illegible]

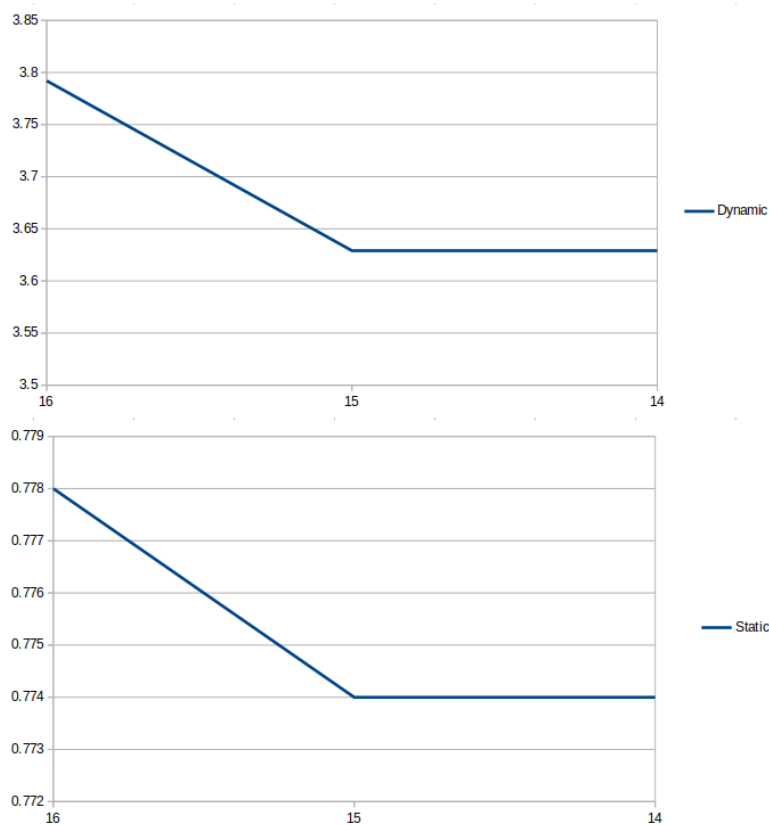
## مصرف توان در حالت سنتز:



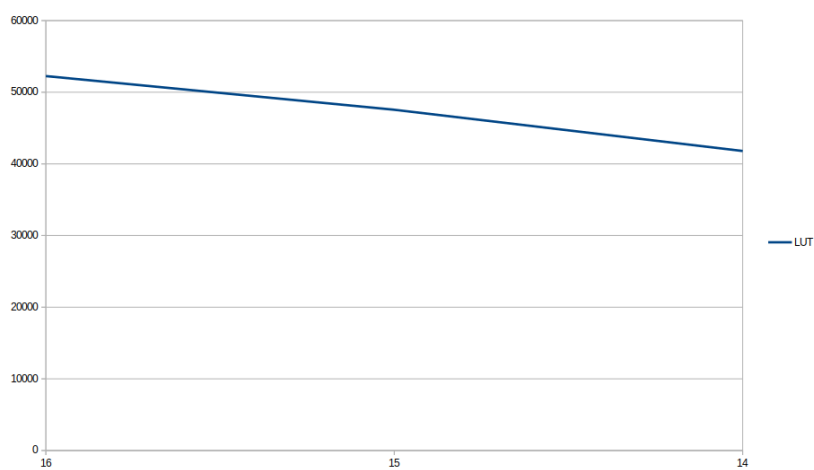
## مصرف منابع (شاخص LUT) در حالت سنتز:



## مصرف توان در حالت پیاده‌سازی:



## مصرف منابع (شاخص LUT) در حالت پیاده‌سازی:



😊 پایان