Advanced VLSI Project – introduction and stage 1

- Please read the table of contents and the introduction and only then start the assignment.
- The project is done in groups of 4.

Table of contents

1. Introduction

[Taken from chapter 2 in Bitcoin Low Power Case Study]

1.1 Technology

Bitcoin is a popular digital currency that is in widespread use today. The compute servers that make up the Bitcoin Network comprise the most powerful network on the planet, and it's become so just in the past ten years.

A Bitcoin Mining server is essentially a massive set of parallel hashing functions, whose sole purpose is to solve mathematical challenges as quickly as possible, to gain a reward of Bitcoins. Thus, mining is the method in which new Bitcoins are made available.
These Miners in the early days were based on CPUs and later GPUs; but later FPGAs and now specialized ASICs have come into play. A massive collection of Mining Systems, also known as Mining Farms, are now dominant in the Bitcoin Network.

1.2 Motivation

For the Mining Farms, the cost of storage and electricity become deciding factors in their profitability; hence an improvement in the overall efficiency of a system leads directly to its overall cost-effectiveness. For example, in 2014, the state-of-the-art Mining System had a throughput of 2 THash/sec (Tera Hashes per second) and was powered by a 1400W power supply. Two years later in 2016, the upgraded system had a throughput of 14 THash/sec and was powered by a 1375W power supply. That is a 7X improvement in power efficiency in just 2 years!
Clearly, the ASICs in that system were significantly improved, and thus achieved the incredible performance improvement. In other words, the same system could use 7X less electricity with the same throughput. For this and other reasons, Synopsys chose to use Bitcoin, a real-world design with real-life impact, to create a Low Power Case Study. Design The "Bitcoin-inspired" design is implemented using a hierarchical approach. Our reasons for doing so are as follows:

- For Physical Design, we utilize MIMs (multiply instantiated modules), so any changes to the base design will only require a single change to an instance that gets propagated, rather than multiple changes to several instances.
- For many users, a hierarchical approach is the preferred design style. The microarchitecture for the Bitcoin design is shown in Figure 2.1.

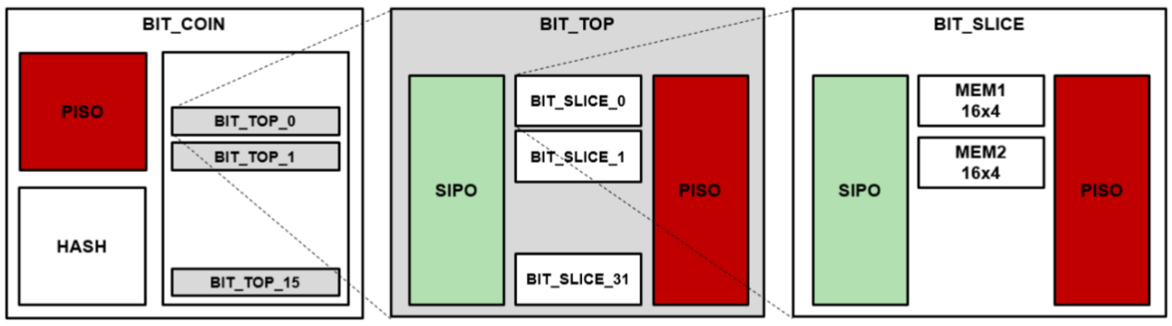


Figure 2.1 - Bitcoin Microarchitecture

Per Figure 2.1, the fundamental building block of the system is "BIT_SLICE", which is comprised of a SIPO (Serial In-Parallel Out) block, two memories, and a PISO (Parallel In-Serial Out) block. The BIT_SLICE block is instantiated 32-times in "BIT_TOP", which also has SIPO and PISO blocks. And finally, BIT_TOP is instantiated 16 times in "BIT_COIN" which has a PISO and HASH block.

Note the Reference Design is “Bitcoin-inspired” because the HASH used is a simplified version of the real hash, a SHA-256 (Secure Hash Algorithm, 256-bit). We made this and other design choices to ensure the runtime of the design through the tools was reasonable and relatively fast while still mimicking a real-world design.

In our project we will implement a simpler version of the Bitcoin case study designed by Synopsys while focusing on the backend side (physical design) and taking the frontend side (RTL implementation and verification) as given.

1.3 The goal for the project

Helping you, the students, prepare for the VLSI industry.

1. Understanding Hierarchical approach
2. Querying information
3. Practicing teamwork
4. Understanding the backend design of chip implementation

1.4 The objectives of the project

1. Implementing the theoretical concepts from the lectures by acquiring hands on experience with the CAD tools. The concepts are included in the different parts [Grade partition]:

1.1. Synthesis	[15%]
1.2. STA	[30%]
1.3. Floorplan	[20%]
1.4. Placement	[20%]
1.5. CTS	[5%]
1.6. Routing	[5%]
1.7. Signoff	[5%]

2. The tools:
   - 2.1. Provided by our vendor Synopsys: **Fusion Compiler.**
3. Questions about the project:
   - 3.1. We opened a dedicated forum for questions about the project.
     Please specify at the title: the related stage your question is about, for example:
     “Synthesis \\ <Question>”
   - 3.2. Sources of information for you to use:
     - 3.2.1. “man” command in fusion compiler
     - 3.2.2. SolvNet search engine (Synopsys). A tutorial on how to open an account can be found on Moodle.

     Before asking in the forum, we encourage you to search for yourself in those resources.

2. Setup

After you have connected to your VPN and then VNC, open a new terminal (mouse - right click “open in terminal”).

2.1 Course working area:

If you haven’t created a working station already (as the last part in the “connecting to microns” pdf), just type in the console:

```
advvlsi
```

it will take you to your working station, which is in the following template:

/project/advvlsi/users/$user/ws

Enter the following command:

```
cd /project/advvlsi/users/$user/ws
```

2.2 Creating the project workspace

Read the PDF file about the project workspace overview in our Moodle, to get a better understanding before you continue forward.

2.3 Copying the files

copy and paste (click mouse3 (roller)) in your terminal the following commands in orange.

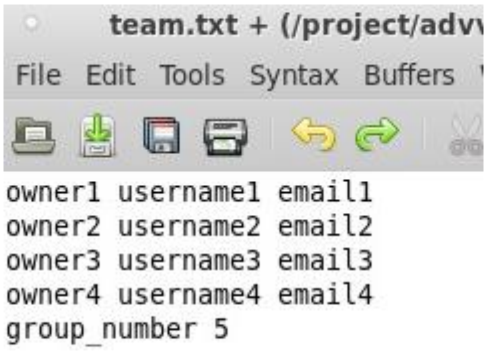
copy the work area template from Elad’s work area.

```
mkdir -p /project/advvlsi/users/$user/ws/bitcoin
cp -r /project/advvlsi/users/eladsimanian/ws/bitcoin/stage1 bitcoin/
```

2.4 Teams documentation

1. Edit the following file:

```
gvim bitcoin/stage1/docs/team.txt
```



```
owner1 username1 email1
owner2 username2 email2
owner3 username3 email3
owner4 username4 email4
group_number 5
```

2. Put your usernames and emails instead. To check for your username, you can use the following command in the terminal:

```
echo $user
```

3. Change the group name according to the "project – groups of 4" in the Moodle.
   3.1. To write in gvim, press `i` it means insert mode
   3.2. To go out of insert mode, press escape.
   3.3. To save in gvim, type (include the :) `:wq` and enter
   3.4. To learn more about gvim you can type `vimtutor` , to quit it do `:exit` and enter

3. Stage 1: Synthesis

As was discussed during the lecture, the Synthesis stage is mainly about converting the RTL code delivered by the Frontend team into logic gates.

First, we need to set the environment for fusion compiler, meaning to configurate the program such that we will familiarize ourselves with the standard cells, external vendors IPs (such as SRAMs), the constraints we wish to meet and so on.

In the directory you copied from Elad's workspace you will find a TCL script named: "bitcoin_stage_1.tcl" that will guide you through this stage of implementation.

For your comfort we marked all tasks needed for submission with:
[#P<Part_ID>_Q<Q_ID>] such as **[#P3.1_Q1]**

To answer the questions, open a word document with the matching question numbers.

3.1 Get familiar with Fusion Compiler and the bitcoin design

1. Open tcl script

   Change directory to the workspace for stage 1:
   (some of the files being used by the scripts relay on relative paths).

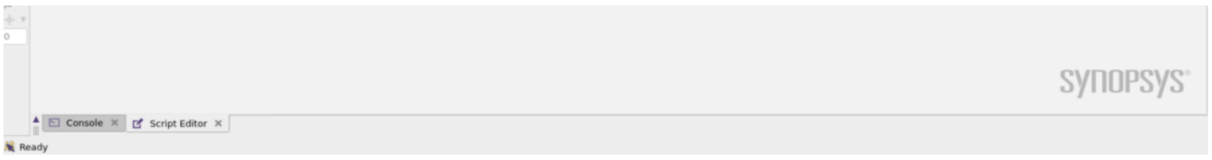
```
cd bitcoin/stage1/workspace/stage1/
```

1.1. Open the Fusion Compiler program using the

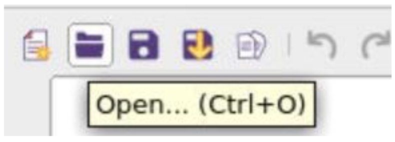
```
fc_shell -gui
```

Note: Common error can be because you are not working on advvlsi.

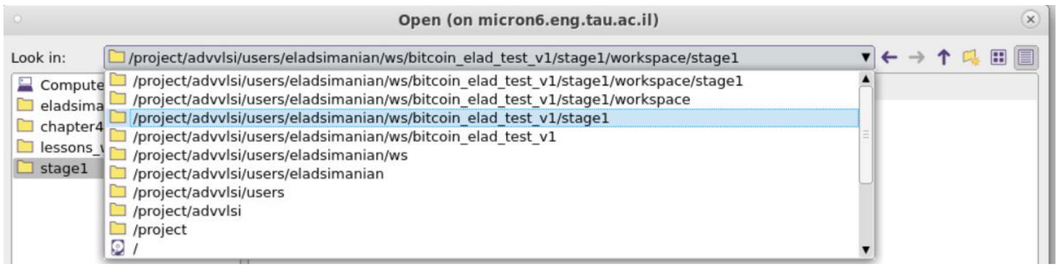
1.2. While you are in Fusion Compiler program, look for the script editor (bottom panel near the console).



1.3. Open the bitcoin_stage1.tcl file by using:



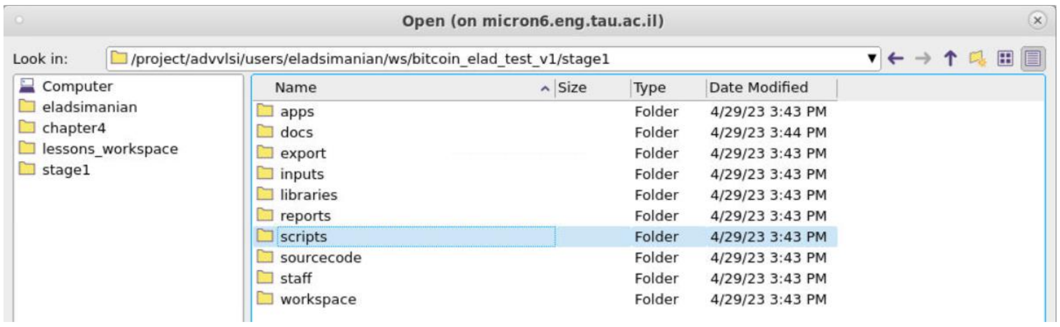
1.4. Travel 2 hierarchies back to stage1 (make sure that the path is yours, and NOT eladsimanian's).


Open (on micron6.eng.tau.ac.il)

Look in: /project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1/stage1/workspace/stage1

```
/project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1/stage1/workspace/stage1
/project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1/stage1/workspace
/project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1/stage1
/project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1
/project/advvlsi/users/eladsimanian/ws
/project/advvlsi/users/eladsimanian
/project/advvlsi/users
/project/advvlsi
/project
/
```

1.5. Enter the scripts directory:


Open (on micron6.eng.tau.ac.il)

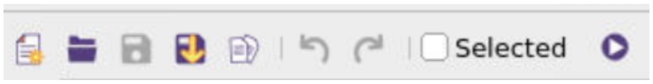
Look in: /project/advvlsi/users/eladsimanian/ws/bitcoin_elad_test_v1/stage1

Name	Size	Type	Date Modified
apps		Folder	4/29/23 3:43 PM
docs		Folder	4/29/23 3:44 PM
export		Folder	4/29/23 3:43 PM
inputs		Folder	4/29/23 3:43 PM
libraries		Folder	4/29/23 3:43 PM
reports		Folder	4/29/23 3:43 PM
scripts		Folder	4/29/23 3:43 PM
sourcecode		Folder	4/29/23 3:43 PM
staff		Folder	4/29/23 3:43 PM
workspace		Folder	4/29/23 3:43 PM

1.6. Open the bitcoin_stage1.tcl file.

1.7. Press the script editor once again at the bottom panel.

1.8. Enable the "Selected" button so you could mark commands in the file and run it using the "play" button one by one:



(If it's in grey and unclickable, double click the first line of the code, to make it colored in blue and try again).

2. Read RTL

2.1. Run the commands in this part of the script one at the time; double click line 4, press play, double click line 5, press play …
Make sure there aren't any errors along the way.

```
#####                 Part 1 Read RTL                 #####
.
.
.
##############################################################
```

Don't continue yet to

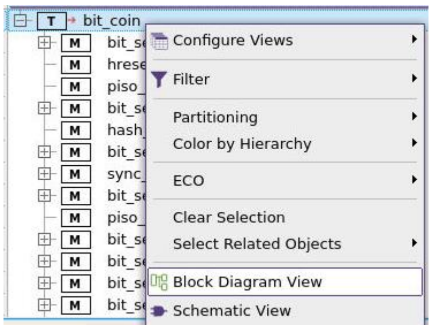
##### Part 2 Compile flow #####

2.2. As we discussed, the implementation of the bitcoin chip is hierarchical. We want to get a general idea on how the different modules look like and how they are connected to one another.

2.3. Useful tips:

2.3.1. By right clicking the "bit_coin" item in the Logical Hierarchy window, open a "Block Diagram View".

2.3.2. By double clicking a certain block we can “expand” the selected item



2.4. **[#P3.1_Q1]** Locate and attach a few print screens of the major items from the architecture image (page 2) including:

2.4.1. Bit_coin: hash, piso and 16 bit_top (named also bit_secure)

2.4.2. Bit_slice: piso, sipo, 2 nibbles

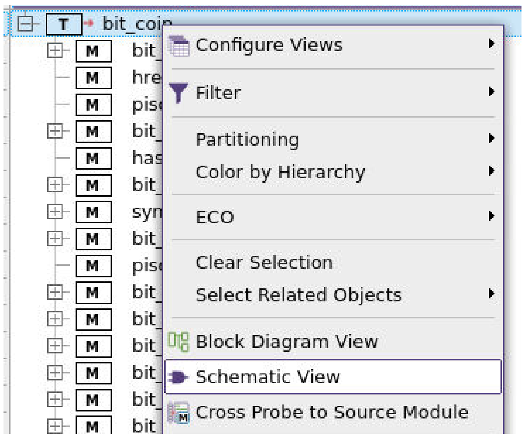
Add titles (such as “hash”, “nibble” etc. to your prints) with the name of the modules onto the print screen.

Important note:

The RTL of the bitcoin project requires some delving into to understand, and some background knowledge regarding its architecture. **You are not expected to understand the functionality of RTL. Instead, only the general composition of the modules.** In this part of the course, our goal is to understand the backend implementation, for that reason we will get the RTL as given and skip straight to the physical design.

On a personal note, in the industry, having a good understanding of the data flow could go a long way and encourage smarter decisions during the implementation.

For additional exploration of the design, you can also select “Schematic View”.

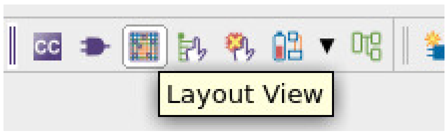


3.2 Warmup Questions

1. **[#P3.2_Q1]** Constraints:
   1.1. Write at least 5 different constraint examples, write who supplies them and their domain. For example, we have the clocks' periods constraint from the timing domain. And the architecture team is responsible for providing them.
   1.2. Name and explain 3 techniques that we (and maybe by extension the tool) can implement during the synthesis stage to meet low power constraints?
   1.3. What are the tradeoffs with the solutions that you offered?
2. **[#P3.2_Q2]** False paths:
   2.1. What are False paths?
   2.2. When should we use them?
3. **[#P3.2_Q3]** Corners, modes, and scenarios:
   3.1. Look at bitcoin_stage_1.tcl and locate the part related to the corners, modes, and scenarios (MCMM). Which scenario \ scenarios are active?
   3.2. What kind of corners, modes, and scenarios would we test in a typical design and why?
4. **[#P3.2_Q4]** What are Multibit Registers?

3.3 Part 3: Compile flow

1. Open the Layout view by selecting (upper panel - 1 o'clock on the screen):



2. Run the commands found in the tcl script "Part 2 Compile flow" one at a time (ignore the false path missing code for now), so you can see the changes being made.

While executing the command "read_sdc ../../sdc/bit_coin.sdc" you will see some error rising – that's ok, just ignore it.

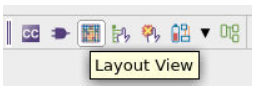
1.1. Proc "collect_reports":
   1.1.1. For defining the proc properly you should select all of it together and run the command.
   1.1.2. Before calling the proc you should make sure a directory /reports/stage1 exists.

1.2. Simplified design: You might notice we use the same "PVT" [Process, Voltage, Temperature] named: "tt0p85v125c", for all corners and scenarios. Also, we haven't activated all the scenarios that were defined.
In a real design that won't be the case and **typically we would check a lot of corner cases (about 32 corner cases)** regarding different combinations of processes, voltages, temperatures.

1.3. Compile_fusion – When you get to the "complile_fusion" commands start with reading about the options and flags this command has (Run: "man compile_fusion").
Note: Some of the "Compile_fusion" commands will take some time to finish.
Be patient and remember the project you had at "Intro to VLSI".

3. After executing the compile_fusion commands until "initial_place" stage:

2.1. Add a layout view (upper panel - 1 o'clock on the screen):



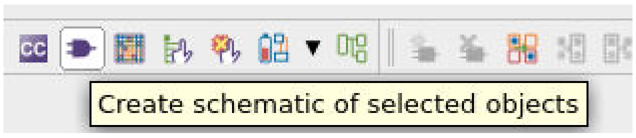
2.2. Then, look at the timing report at this stage in the flow, you can open the file located in the "reports_stage_1" directory (open another terminal and navigate to that report, read it using the less command) or just re-run the report_timing commands (for SETUP and HOLD violations) and look at the results in the prompt.

2.3. **[#P3.3_Q1]** What is the slack for the SETUP and HOLD constraint? Attach a print screen for both.

2.4. **[#P3.3_Q2]** Read the reports and find the starting and ending point of the critical path. If needed use the "-through" option to specify the specific path between those cells.
Fill the code for the command: (the through is optional, you can delete the flag if you don't use it).

```
set tp1 [get_timing_paths -from XXX -to XXX -through XXX]
change_selection $tp1
```

2.5. **[#P3.3_Q3]** While we still in the context of the path (done by section d) select the "Schematic View" and take a snapshot of the critical paths for SETUP and HOLD:



2.6. **[#P3.3_Q4]** Find an additional timing path with a major negative slack. To do that, use the report_timing command with the following flag: "-max_path". Try to find a pattern to those paths.
Hint: Maybe you can find a common factor that can cause the problematic slack.
Use this pattern to fill the code for the "set_false_path" command.

2.7. Finish running the tcl commands until the end of file.

2.8. **[#P3.3_Q5]** After you ran the "final_opto" compilation, change your selection again to the same path you chose for the SETUP path in [#P3.3_Q3].

```
change_selection $tp1
```

Select again "Schematic View" and take another snapshot.
Explain the difference between those snapshots.

3. In the tcl script we should have some reports for every stage of compile_fusion (inserted to the directory by the collect_reports proc).
4. **[#P3.3_Q6]** Analyze those reports and answer the following questions:
   4.1. Setup timing: What is the first stage in the compile_fusion flow that fixes the SETUP violations?
   4.2. Hold timing: From which stage in the general chip implementation (Synthesis, STA, Floorplan, Placement…) should we check the hold violations and why?
   4.3. Area: What can we derive out of the following information from the area report?
      4.3.1. Number of ports
      4.3.2. The ratio between number of the combinational cells and the sequential cells
      4.3.3. Number of macros
      4.3.4. Number of buffers and inverters
      4.3.5. Area of cells + area of macros
   4.4. Design: What can we derive out of the following information from the design report?
      4.4.1. ICG - integrated clock gates
      4.4.2. Latches
      4.4.3. Number of Power Domains
   4.5. Power: What can we derive out of the following information from the power report?
      4.5.1. What is the static, dynamic and total power after all compile_fusion command?
      4.5.2. What does each of the power types mean?

4 Submission

1. If the team approves the design, the team lead (and **only him/her**), which has the updated work area with all the teams' files working, should submit the assignment as follows.

2. Submit to Moodle a zip: bitcoin_stage1_group**<group_num>**.zip

   The zip must include two files:

   1.1. PDF file: bitcoin_stage1_group**<group_num>**.pdf, include the following:

   1.1.1. The stage name and number (Stage 1 – Synthesis)

   1.1.2. Workarea path, such as: /project/advvlsi/users/$user/ws/bitcoin

   1.1.3. Team's info: members name, id, usernames (in micron servers), tau emails

   1.1.4. Answers for all questions and tasks marked with "[#P<>Q<>]"

   See below all the questions for this assignment

   1.2. TCL file: attach the modified TCL script after filling in the missing lines.

Good Luck.

P.S the next stage will discuss the STA stage!

Questions only

[#P3.1_Q1] Locate and attach a few print screens of the major items from the architecture image (page 2) including:

- Bit_coin: hash, piso and 16 bit_top (named also bit_secure)
- Bit_slice: piso, sipo, 2 nibbles

[#P3.2_Q1] Constraints:

- Write at least 5 different constraint examples, write who supplies them and their domain. For example, we have the clocks' periods constraint from the timing domain. And the architecture team is responsible for providing them.
- Name and explain 2 techniques that we (and maybe by extension the tool) can implement during the synthesis stage to meet low power constraints?
- What are the tradeoffs with the solutions that you offered?

[#P3.2_Q2] False paths:

- What are False paths?
- When should we use them?

[#P3.2_Q3] Corners, modes, and scenarios:

- Look at bitcoin_stage_1.tcl and locate the part related to the corners, modes, and scenarios (MCMM). Which scenario \ scenarios are active?
- What kind of corners, modes, and scenarios would we test in a typical design and why?

[#P3.2_Q4] What are Multibit Registers?

[#P3.3_Q1] What is the slack for the SETUP and HOLD constraint? Attach a print screen for both.

[#P3.3_Q2] Read the reports and find the starting and ending point of the critical path. If needed use the "-through" option to specify the specific path between those cells.
Fill the code for the command: (the through is optional, you can delete the flag if you don't use it).

[#P3.3_Q3] While we still in the context of the path (done by section d) select the "Schematic View" and take a snapshot of the critical paths for SETUP and HOLD:

[#P3.3_Q4] Find an additional timing path with a major negative slack. To do that, use the report_timing command with the following flag: "-max_path". Try to find a pattern to those paths.
Hint: Maybe you can find a common factor that can cause the problematic slack.
Use this pattern to fill the code for the "set_false_path" command.

[#P3.3_Q5] After you ran the "final_opto" compilation, change your selection again to the same path you chose for the SETUP path in [#P3.3_Q3].

[#P3.3_Q6] Analyze those reports and answer the following questions:

- Setup timing: What is the first stage in the compile_fusion flow that fixes the SETUP violations?
- Hold timing: From which stage in the general chip implementation (Synthesis, STA, Floorplan, Placement…) should we check the hold violations and why?
- Area: What can we derive out of the following information from the area report?

  - Number of ports
  - The ratio between number of the combinational cells and the sequential cells
  - Number of macros
  - Number of buffers and inverters
  - Area of cells + area of macros
- Design: What can we derive out of the following information from the design report?
  - ICG - integrated clock gates
  - Latches
  - Number of Power Domains
- Power: What can we derive out of the following information from the power report?
  - What is the static, dynamic and total power after all compile_fusion command?
  - What does each of the power types mean?