EC383 - Mini Project in VLSI Design

Password Cracker

End Sem Evaluation Report

Submitted by
R S Muthukumar 201EC149
Utkarsh Mahajan 201EC164
Devvrat Mukund Ashtaputre 201EC118

Project Github Link

6rd Semester B.tech (ECE)

Course Instructor: Dr. Ramesh Kini M.

Department of Electronics and Communications Engineering,

National Institute of Technology Karnataka, Surathkal

Introduction:

In today's digital age, passwords are an essential part of securing various digital systems and accounts, including online banking, email, social media, and more. However, the security of a system can be easily compromised if an attacker gains access to the passwords. Therefore, passwords are often hashed to protect them from being easily stolen or decrypted. Hashing is a one-way process that converts the password into an encrypted code that cannot be easily reversed. Our Project presents the development of a Verilog-based password cracker that implements a dictionary attack to crack SHA-256 hashed passwords.

Literature Survey:

Password cracking has been a topic of interest for many years, with various techniques developed to crack passwords. In recent years, the use of field-programmable gate arrays (FPGAs) for password cracking has gained popularity due to their ability to perform parallel processing, making password cracking much faster than traditional methods. Additionally, FPGA-based implementations of password cracking are more power-efficient than traditional methods, making them an attractive option for security researchers. ASCII-based password cracking methods have also been explored, but they are much slower than FPGA-based methods. Therefore, the use of Verilog in the development of password crackers has gained popularity in recent years due to its ability to be synthesized into FPGA hardware.

Hashcat is an example of password cracking software that uses a variety of methods, including dictionary attacks, brute-force attacks, and more. It is a popular password cracking tool due to its ability to leverage the power of GPUs to crack passwords quickly. However, software-based password cracking is still slower than FPGA-based methods, making FPGA-based password crackers a more efficient option for security researchers. SHA-256 is a popular cryptographic hash function used for password hashing, and it is considered to be highly secure. It is one of the most commonly used hash functions, with many digital systems and websites using it for password storage. Therefore, we chose SHA-256 for our project as it is a widely used and highly secure hash function.

In our literature survey, we explored various tools and techniques for the design and development of digital chips. One notable tool that we came across is

OpenLANE, which provides a complete RTL-to-GDSII flow for digital chip design. This flow involves converting the design of a chip from a high-level language like Verilog or VHDL to the final physical layout that can be sent for fabrication. OpenLANE includes tools for synthesis, placement and routing, timing analysis, and physical verification, making it a comprehensive solution for digital chip design.

One of the key advantages of OpenLANE is that it is open-source, which means that it is freely available for anyone to use and modify. This makes it an attractive tool for academics, researchers, and small companies who may not have the resources to invest in expensive commercial tools. By leveraging the capabilities of OpenLANE, we were able to effectively design and implement our project in a cost-effective and efficient manner.

Problem Statement:

The RTL-to-GDSII synthesis process is critical for the efficient implementation of digital systems, including password crackers. Password cracking remains a significant security threat in today's digital age, and FPGA-based implementations of password crackers have become increasingly popular due to their parallel processing capability and power efficiency. This project aims to develop a Verilog-based password cracker that uses a dictionary attack to crack SHA-256 hashed passwords, with a focus on the RTL-to-GDSII synthesis process.

Motivation:

The development of a Verilog-based password cracker has several motivations. Firstly, it provides an opportunity to learn about RTL to GDS-2 synthesis while implementing a real-world application. Secondly, it highlights the importance of strong password security measures to prevent unauthorized access to digital systems. Finally, the project provides an opportunity to apply Verilog design skills to develop a faster and more efficient method of password cracking.

Objectives:

The objectives of this project are:

- To develop a Verilog-based password cracker that uses a dictionary attack to crack SHA-256 hashed passwords.
- To synthesize the design using OpenLane.
- To understand and evaluate the synthesized design of the password cracker.

Project Details:

The password cracker consists of several modules, including the top-level SHA-256 module, a memory module, and a control module.

The top-level SHA-256 module is responsible for computing the SHA-256 hash of the passwords. It consists of several submodules, including the message parser, scheduler, and digest modules. The message parser processes the input password message and prepares it for the subsequent stages. The scheduler schedules the message blocks for processing by the digest module, which computes the final hash output.

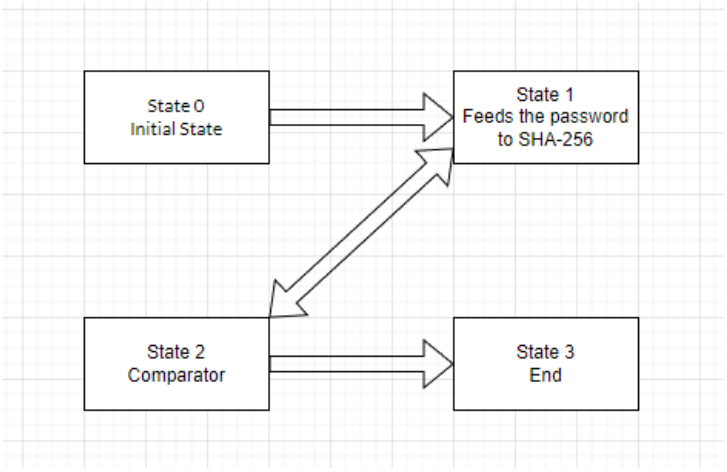
The memory module is responsible for storing the password list that the password cracker uses for the dictionary attack. It uses byte-addressable memory to store the passwords and provides an interface to the control module to read and write from the memory.

The control module (main) is responsible for coordinating the operations of the other modules. It uses a finite state machine to control the flow of data and signals between the modules. It reads the hashed password input and compares it with the hash values of the passwords in the memory to find a match.

The RTL-to-GDS flow for the project uses the SkyWater 130nm open-source process technology. The flow involves several steps, including synthesis, floorplanning, placement, routing, and post-layout simulation. The project uses the open-source OpenLane toolchain to automate the RTL-to-GDS flow.

Implementation details:

The password cracker implementation consists of several modules including main.v, m_digest.v, m_padder_parser.v, m_scheduler.v, top_sha.v, and interative_processing.v. The main control module is main.v, which contains the state machine that controls the overall operation of the password cracker.



As can be seen from the state diagram, The machine has three states:

- State 0: Initial State
  - In state 0, the password cracker initializes various variables and registers, including the count, state, rst, memory_address, start_string, curr_index, password_count, cracked, and done.
- State 1: Feeds the password to SHA256 module
  - In state 1, the password cracker reads the password from memory and feeds it into the SHA256 module through the top_sha.v module. The SHA256 module processes the input password and generates a hash value.
- State 2: Comparator
  - In state 2, the password cracker compares the generated hash value with the target hash value. If the generated hash matches the target hash, the password has been successfully cracked and the cracked flag is set to 1. If the hash does not match, the state machine transitions back to state 1 to process the next password.
- State 3: End
  - In state 3, the password cracker has completed its operation and sets the done flag to 1.

SHA 256 Algorithm:

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that generates a fixed-size output of 256 bits. It is widely used in the field of cryptography for secure data transmission and storage. Here are the steps involved in the SHA-256 algorithm:

Initialization: The SHA-256 algorithm initializes eight 32-bit words called "hash values" with predetermined constants. These hash values are used to compute the final hash output.

Message Padding: The input message is padded with zeros to ensure its length is a multiple of 512 bits. Then a 64-bit representation of the original message length is appended to the padded message.

Breaking the message into chunks: The padded message is divided into 512-bit chunks called "message blocks".

Message Schedule: Each message block is processed through a series of steps to generate a set of 64 words called the "message schedule".

Compression: The message schedule and hash values are used in a compression

function that generates a new set of hash values for each message block.

Output: After all message blocks have been processed, the hash values are concatenated to form the final output, which is a 256-bit hash.

The SHA256 module consists of several sub-modules including m_digest.v, m_padder_parser.v, m_scheduler.v, and interative_processing.v. These modules perform the various stages of the SHA256 hashing algorithm, including padding the input message, parsing it into blocks, scheduling the blocks for processing, and iteratively processing the blocks to generate the hash value.

The memory used in the password cracker is byte-addressable and is implemented using a Verilog array byte_addressable_memory. The memory stores the input passwords and the target hash value.

The implementation is verified using a testbench main_tb.v, which initializes the memory with test passwords and verifies that the password cracker successfully cracks the passwords.

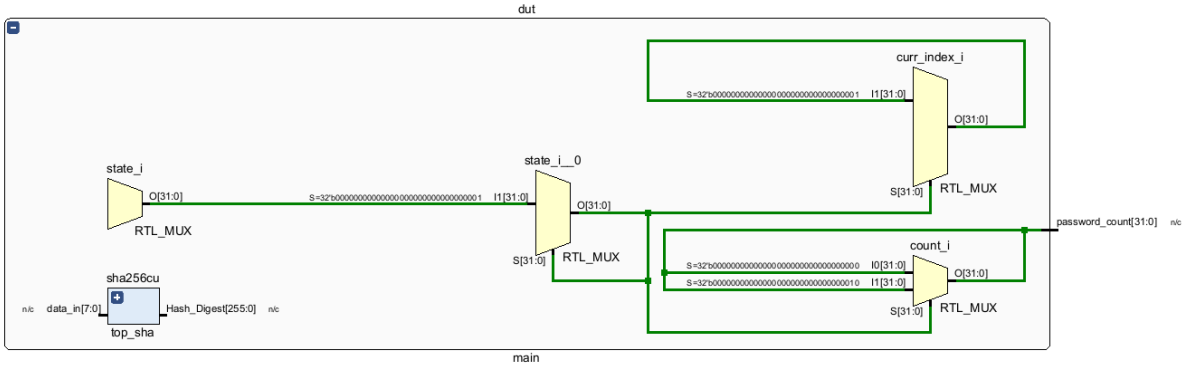
RTL to GDS-2 Flow

To first add a new design, we need a config file (config.json). Config file is the entry point to using OpenLane. We can specify a lot of things here. For instance, we have specified the clock period, port, and by default unless mentioned, openlane assumes we do Area based synthesis.

```
{
   "DESIGN_NAME": "password_cracker",
   "VERILOG_FILES": "designs/password_cracker/password_cracker.v
                     designs/password_cracker/m_digest.v
                     designs/password_cracker/m_pader_parser.v
                     designs/password_cracker/m_scheduler.v
                     designs/password_cracker/top_sha.v
                     designs/password_cracker/interative_processing.v",
   "CLOCK_PORT": "clk",
   "CLOCK_PERIOD": 10.0,
   "DESIGN_IS_CORE": true
}
```

Results:

Xilinx Vivado Synthesis Results:



Exploration Analysis Report

This report summarizes the results of the exploration analysis for the design. The objective of the analysis is to find the optimal strategy that minimizes the area, delay and number of gates for the design.

Best Results:

The table below shows the best results achieved for each metric:

Best Area	Best Gate Count	Best Delay
57244.9	6711.0	3295.53
AREA 2	AREA 1	AREA 3

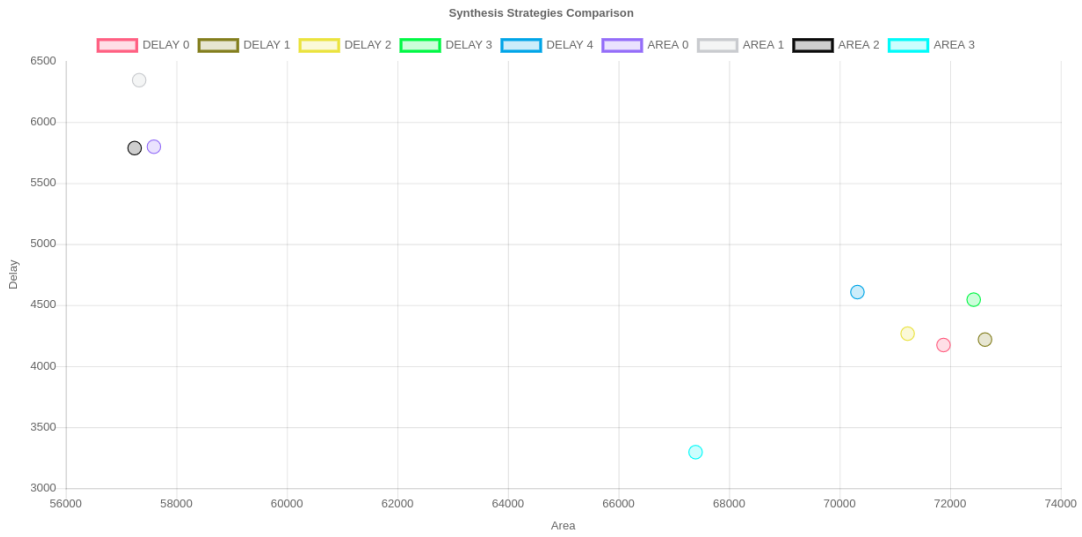
Strategy Comparison:

The table below compares the different strategies that were analyzed:

Strategy	Gate Count	Area (um^2)	Delay (ps)	Gates Ratio	Area Ratio	Delay Ratio
DELAY 0	8105.0	71880.19	4173.14	1.207	1.255	1.266
DELAY 1	8078.0	72630.91	4217.85	1.203	1.268	1.279
DELAY 2	7958.0	71230.81	4266.25	1.185	1.244	1.294
DELAY 3	8097.0	72425.71	4544.68	1.206	1.265	1.379
DELAY 4	8235.0	70322.45	4606.56	1.227	1.228	1.397
AREA 0	6743.0	57593.99	5797.69	1.004	1.006	1.759
AREA 1	6711.0	57327.48	6342.52	1.0	1.001	1.924
AREA 2	6774.0	57244.9	5786.28	1.009	1.0	1.755
AREA 3	9321.0	67394.63	3295.53	1.388	1.177	1.0

Scatter Chart

The scatter chart below shows the relationship between area and delay for each strategy:

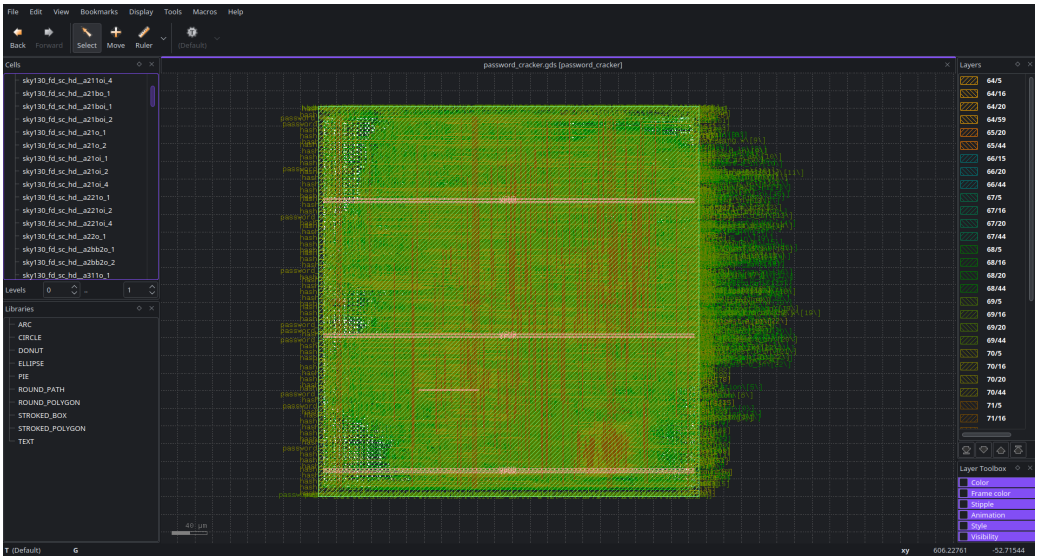


The chart shows that strategies with lower delay tend to have higher area, and vice versa. However, We can see that the AREA 2 strategy stands out as having both low delay and low area.

Conclusion

Based on the results of the exploration analysis, the best strategy for minimizing area, delay, and gate count is AREA 2. However, other strategies with higher delay may also be considered depending on the design requirements.

GDS layout of the fabricated chip:



The GDS file used for this project was generated by OpenLane, an automated RTL to GDSII flow for ASIC design. This binary file format represents the geometric layout of the integrated circuit design, including the placement of components, interconnects, and other physical details. It was generated as part of the final stage of the flow, which involves layout synthesis and optimization using various EDA tools. The GDS file was visualized using Klayout, a specialized layout editor software.

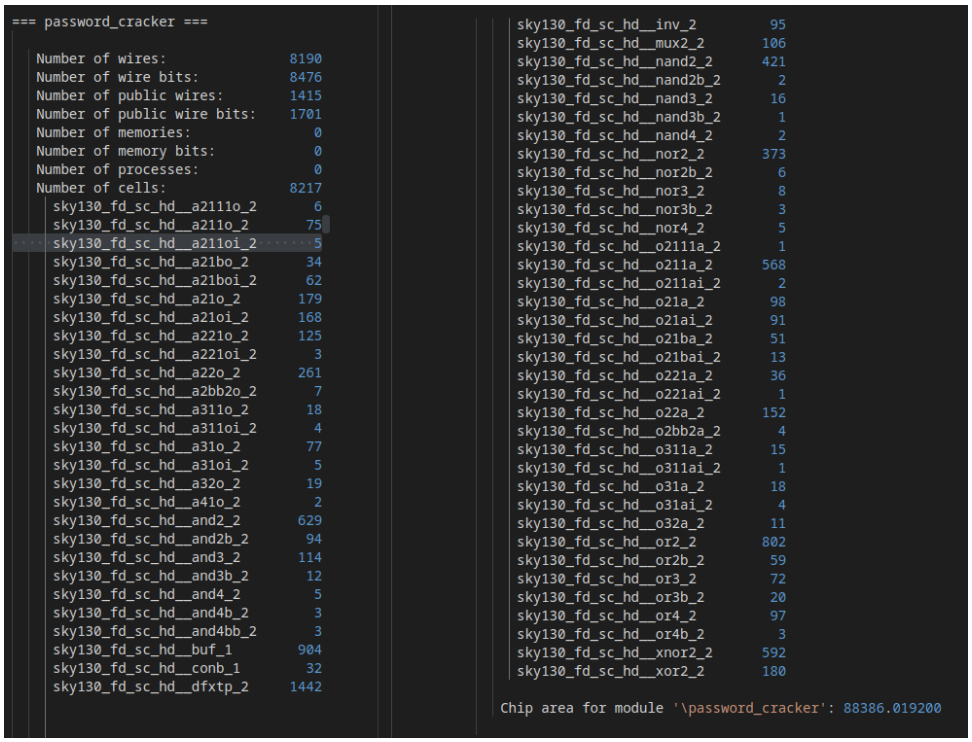
Initial and post-technology mapping statistics of the "password_cracker" circuit:

Initial statistics:

```
=== password_cracker ===

   Number of wires:                 11638
   Number of wire bits:             15376
   Number of public wires:            172
   Number of public wire bits:       3089
   Number of memories:                  0
   Number of memory bits:               0
   Number of processes:                 0
   Number of cells:                 13513
     $_ANDNOT_                       3102
     $_AND_                           206
     $_DFFE_PP_                       515
     $_DFF_P_                          36
     $_MUX_                          2961
     $_NAND_                          135
     $_NOR_                           370
     $_NOT_                           326
     $_ORNOT_                         918
     $_OR_                           2845
     $_SDFFE_PN0P_                    735
     $_SDFFE_PN1P_                    138
     $_SDFFE_PP0P_                      9
     $_SDFF_PN0_                        9
     $_XNOR_                          577
     $_XOR_                           631
```

After technology mapping, using AREA 0 strategy, using ABC:



```
=== password_cracker ===

   Number of wires:                 8190
   Number of wire bits:             8476
   Number of public wires:          1415
   Number of public wire bits:      1701
   Number of memories:                 0
   Number of memory bits:              0
   Number of processes:                0
   Number of cells:                 8217
     sky130_fd_sc_hd__a2111o_2         6
     sky130_fd_sc_hd__a211o_2         75
     sky130_fd_sc_hd__a211oi_2         5
     sky130_fd_sc_hd__a21bo_2         34
     sky130_fd_sc_hd__a21boi_2        62
     sky130_fd_sc_hd__a21o_2         179
     sky130_fd_sc_hd__a21oi_2        168
     sky130_fd_sc_hd__a221o_2        125
     sky130_fd_sc_hd__a221oi_2         3
     sky130_fd_sc_hd__a22o_2         261
     sky130_fd_sc_hd__a2bb2o_2         7
     sky130_fd_sc_hd__a311o_2         18
     sky130_fd_sc_hd__a311oi_2         4
     sky130_fd_sc_hd__a31o_2          77
     sky130_fd_sc_hd__a31oi_2          5
     sky130_fd_sc_hd__a32o_2          19
     sky130_fd_sc_hd__a41o_2           2
     sky130_fd_sc_hd__and2_2         629
     sky130_fd_sc_hd__and2b_2         94
     sky130_fd_sc_hd__and3_2         114
     sky130_fd_sc_hd__and3b_2         12
     sky130_fd_sc_hd__and4_2           5
     sky130_fd_sc_hd__and4b_2          3
     sky130_fd_sc_hd__and4bb_2         3
     sky130_fd_sc_hd__buf_1          904
     sky130_fd_sc_hd__conb_1          32
     sky130_fd_sc_hd__dfxtp_2       1442
     sky130_fd_sc_hd__inv_2           95
     sky130_fd_sc_hd__mux2_2         106
     sky130_fd_sc_hd__nand2_2        421
     sky130_fd_sc_hd__nand2b_2         2
     sky130_fd_sc_hd__nand3_2         16
     sky130_fd_sc_hd__nand3b_2         1
     sky130_fd_sc_hd__nand4_2          2
     sky130_fd_sc_hd__nor2_2         373
     sky130_fd_sc_hd__nor2b_2          6
     sky130_fd_sc_hd__nor3_2           8
     sky130_fd_sc_hd__nor3b_2          3
     sky130_fd_sc_hd__nor4_2           5
     sky130_fd_sc_hd__o2111a_2         1
     sky130_fd_sc_hd__o211a_2        568
     sky130_fd_sc_hd__o211ai_2         2
     sky130_fd_sc_hd__o21a_2          98
     sky130_fd_sc_hd__o21ai_2         91
     sky130_fd_sc_hd__o21ba_2         51
     sky130_fd_sc_hd__o21bai_2        13
     sky130_fd_sc_hd__o221a_2         36
     sky130_fd_sc_hd__o221ai_2         1
     sky130_fd_sc_hd__o22a_2         152
     sky130_fd_sc_hd__o2bb2a_2         4
     sky130_fd_sc_hd__o311a_2         15
     sky130_fd_sc_hd__o311ai_2         1
     sky130_fd_sc_hd__o31a_2          18
     sky130_fd_sc_hd__o31ai_2          4
     sky130_fd_sc_hd__o32a_2          11
     sky130_fd_sc_hd__or2_2          802
     sky130_fd_sc_hd__or2b_2          59
     sky130_fd_sc_hd__or3_2           72
     sky130_fd_sc_hd__or3b_2          20
     sky130_fd_sc_hd__or4_2           97
     sky130_fd_sc_hd__or4b_2           3
     sky130_fd_sc_hd__xnor2_2        592
     sky130_fd_sc_hd__xor2_2         180

Chip area for module '\password_cracker': 88386.019200
```

The circuit has 11638 wires, 15376 wire bits, 172 public wires, and 3089 public wire bits in its initial form. The circuit comprises 13513 cells of different types, such as AND gates, D flip-flops, multiplexers, etc.

After technology mapping with ABC, using the AREA 0 strategy, the circuit's size reduced significantly. The circuit has 8190 wires, 8476 wire bits, 1415 public wires, and 1701 public wire bits after technology mapping. The number of cells also reduced from 13513 to 8217.

The report provides a breakdown of the cells used in the circuit after technology mapping. The circuit mainly comprises sky130_fd_sc_hd__a* cells, such as a211o_2, a21o_2, a221o_2, etc. These cells have different functionalities, such as AND gates, OR gates, MUXes, and inverters.

Overall, the report shows that technology mapping can significantly reduce circuit size and the number of cells required to implement a circuit while maintaining its functionality. This reduction can lead to improved performance, reduced power consumption, and lower costs.

One crucial line in the previously presented data indicates the usage of delay flops as follows: "sky130_fd_sc_hd__dfxtp_2 1442." This indicates that 1442 delay flops were used. It is important to note that each line in the data represents a type of gate utilized. For instance, o221a represents a 2-input OR into the first two inputs of a 3-input AND.

Metrics:

design	design_name	config	flow_status	total_runtime	routed_runtime	(Cell/mm^2)/Cor	DIEAREA_mm^2	CellPer_mm^2	OpenDP_Util	Peak_Memory_(	cell_count
/openlane/designs/pa	password_cracker	RUN_2023.03.2	flow completed	0h28m35s0ms	0h23m32s0ms	86131.86962	0.190800456	43065.93481	51.44	1237.57	8217

Conclusion

In conclusion, our project to implement SHA-256 on OpenLane was successful. We were able to design and simulate a circuit that can perform the SHA-256 algorithm, which is commonly used for secure hashing of data. OpenLane provided us with a powerful open-source toolchain for implementing our design, allowing us to efficiently automate the design flow from RTL synthesis to layout and final verification.

The design process involved creating a Verilog RTL code for the SHA-256 algorithm, which we then synthesized and optimized using OpenLane's Synthesis tool. We also performed floorplanning, placement, and routing to generate the final layout for the circuit. We used OpenLane's DRC and LVS tools to verify the correctness of the layout and ensure that it meets the necessary design rules.

Overall, our project demonstrates the power and flexibility of OpenLane as a tool for designing and implementing complex circuits. We were able to achieve our goal of implementing a secure hashing algorithm and gain valuable experience in the process. The skills and knowledge we gained from this project will undoubtedly be useful in future projects and research endeavors in the field of digital design.