DIGITAL SYSTEM DESIGN PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

Crypto-Hash Engine with Brute-Force Nonce Search

GROUP 7

Muhammad Fairuz Dzaki	**2406368864**
Muhammad Naufal Gilardino	**2406450434**
Raihan Hermuhadzib	**2406369002**
Raul Sumaryada	**2406450466**

PREFACE

First of all, we would like to express our gratitude to God Almighty for being able to complete the final project for the Statistics course. We would like to thank Mr Astha Ekadiyanto, as a lecturer who teaches this digital system design course who has provided guidance and direction to us.

We would also like to thank all the lab assistants who have helped in writing this final project paper. The purpose of writing this paper is to fulfill the final project assignment in the digital system design course.

In compiling this paper, all parties, both directly and indirectly related to the making of this report, have made good and meaningful contributions.

Therefore, we hope that this paper can provide benefits by providing additional knowledge to readers. We realize that the writing of this paper and the making of this final project are not perfect, therefore we welcome criticism and suggestions that are given to help us improve our knowledge in the future. Finally, hopefully this paper can provide significant benefits to readers

Depok, December 07,12, 2025

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Blockchain is an innovative technology that forms the basis for digital or cryptocurrencies, including Bitcoin. Among the important concepts in blockchain systems is mining, a process of transaction validation and adding new blocks to the chain under the mechanism of Proof-of-Work.

Proof of Work is the consensus algorithm in cryptocurrency that requires miners to find a particular value of nonce that results in a hash with specific characteristics. In Bitcoin, a valid hash must have a certain number of leading zeros that define the difficulty level in mining.

Hashing by the SHA-256 algorithm is done in a cycle during the mining process. Miners keep trying different values of nonce until they stumble upon the combination that will yield a valid hash. This computational complexity becomes the main security mechanism of blockchain, as modifying historical data would require recalculating the entire chain.

This project will implement a Bitcoin miner system in hardware using VHDL, or VHSIC Hardware Description Language. The implementation of hardware provides several advantages over software: speed and energy efficiency are two desired reasons, as most industrial mining uses ASICs, Application-Specific Integrated Circuits.

1.2 PROJECT DESCRIPTION

This project is an implementation of a functional Bitcoin mining system that utilizes VHDL for digital system design. It showcases the core ideas of blockchain technology, with the implementation of the Proof-of-Work consensus mechanism used in cryptocurrency mining.

Mining Bitcoins involves the confirmation of transactions and the addition of new blocks into the blockchain. Miners race to solve a computational puzzle: to find a nonce-a number used once-that, when combined with block data and hashed using SHA-256, produces a hash value with the target number of leading zeros. The first miner to identify a valid nonce is rewarded with Bitcoin.

1.3 OBJECTIVES

The objectives of this project are as follows:

1. implementing algorithms in VHDL
2. Design State machine for mining controllers
3. perform multiple modules (behaviorial, dataflow, structural,FSM, functions) into one system
4. understanding the work concept for blockchain system
5. implementing a mining algorithm in VHDL within the modular architecture

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Core components of the algorithms	implementing the main algorithm on using hash Engine	Muhammad Naufal Gilardino
documenting the structure of logic	Writing the Report on various logic and test	Muhammad Fairuz Dzaki

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Vivado (for implementing VHDL as well as creating the schematic)
- Github (for documentary regarding the project)

2.2 IMPLEMENTATION

The Hash core implements a lightweight hash function for VHDL simulation and this implementation is a simplified algorithm that maintains its core cryptographic principles while being practical for hardware description and testing.

The hash function operates on a state based architecture with mixing operations. At initialization the state register is loaded with upper 32 bits of input data,establishing the initial mixing state. the algorithm then puts through 8 compression rounds each contributed to the final hash value through a series of transformations.

```
picked := extract_byte(data_reg, round_cnt);
 shift := 3 + (round_cnt mod 5);
 rot_val := (state_reg xor picked) rol shift;
state_reg <= rot_val + CONSTS(round_cnt);
```

The byte extraction mechanism selects differents part of the input data for each round, making surethat all input bits influence the final output.The XOR operation combines the extracted byte with the current state, introducing non-linearity. Variable rotation amounts (ranging from 3 to 7 bit positions) prevent simple pattern recognition and ensure thorough bit mixing. Finally, round-specific constants prevent symmetrical states and add additional diffusion.

```
x"9E3779B1", x"C2B2A3D9", x"7F4A7C15", x"F39A2D57",
```

```
x"A5E91C23", x"3C6E5F8B", x"1B2C0D9F", x"6DA98B31"
```

These constant ensures that different rounds produce distinct transformations even with similar input patterns. The values are chosen to have good bit distribution and are derived from mathematical constants to avoid accidental patterns.

```
when IDLE =>
    hash_start <= '0';
    nonce_en <= '0';
    nonce_load <= '1';
    done_found <= '0';
    latch_nonce <= '0';
    if start_search = '1' then
        state <= LOAD;
    end if;
```

The controller FSM implements the core mining algorithm through a well-defined sequence of states that orchestrate all system components. This finite state machine serves as the central coordination unit, managing the timing and sequencing of nonce generation, hash computation, and result verification.

The state machine operates synchronously with the system clock, transitioning between states based on input conditions and internal status signals. Each state has specific responsibilities and generates appropriate control signals for other module

```
when LOAD =>
    nonce_load <= '1';
    hash_start <= '1';
    nonce_en <= '0';
    latch_nonce <= '0';
    state <= HASHING;
```

During the HASHING state, the controller waits for the hash core to complete its computation. This state monitors the hash_busy and hash_done signals from the hash core. The hash

computation requires approximately 9 clock cycles (8 rounds plus output), during which the controller remains in HASHING state. Once hash_done asserts, indicating completion, the controller advances to CHECK.

```
when HASHING =>
   hash_start <= '0';
   nonce_load <= '0';
   if hash_busy = '1' then
      state <= HASHING;
   elsif hash_done = '1' then
      state <= CHECK;
   end if;
```

The CHECK state represents the critical decision point in the mining algorithm. Here, the controller evaluates the cmp_less signal from the comparator to determine if the current hash meets the target requirement. If cmp_less is high, indicating hash < target, mining has succeeded. The controller asserts latch_nonce to capture the winning nonce value and done_found to signal completion, then transitions to FOUND state. If the hash is insufficient, the controller asserts nonce_en to increment the nonce counter and hash_start to begin a new hash computation, returning to HASHING state for another attempt.

```
when CHECK =>
   if cmp_less = '1' then
      done_found <= '1';
      latch_nonce <= '1';
      nonce_en <= '0';
      state <= FOUND;
   else
      nonce_en <= '1';
      latch_nonce <= '0';
      hash_start <= '1';
      state <= HASHING;
   end if;
```

The FOUND state maintains the success condition, keeping done_found asserted so external systems can read the result. The controller waits in this state until start_search is deasserted, indicating acknowledgment of the result, before transitioning to STOP state.

```
when FOUND =>
    latch_nonce <= '0';
    hash_start <= '0';
    nonce_en <= '0';
    if start_search = '0' then
        state <= STOP;
    end if;
```

The nonce generator serves as the systematic search mechanism for the mining system. It maintains a 32-bit counter that sequences through potential nonce values, enabling the exhaustive search required for Proof-of-Work mining.

The nonce generator operates with two primary modes: load and increment. In load mode, activated by the controller during initialization, the counter is set to a specific starting value (typically zero for a new search). In increment mode, the counter advances by one each clock cycle, producing the next candidate nonce. This simple but effective mechanism ensures systematic coverage of the nonce space.

```
process(clk, rst)
begin
    if rst = '1' then
        cnt <= (others => '0');
    elsif rising_edge(clk) then
        if load = '1' then
            cnt <= unsigned(load_val);
        elsif enable = '1' then
            cnt <= cnt + 1;
        end if;
    end if;
end process;
```

The synchronous design ensures predictable timing and clean integration with other system components. Reset immediately clears the counter to zero, providing a known initial state. The priority structure (reset > load > enable) prevents conflicting operations and ensures deterministic behavior.

The comparator module implements the fundamental test that determines mining success. Its purpose is elegantly simple: evaluate whether a computed hash is numerically less than the target threshold, thereby validating if the current nonce produces acceptable Proof-of-Work.

```
less <= '1' when unsigned(hash_in) < unsigned(target) else '0';
```

This single line of VHDL really captures the essence of how Bitcoin's difficulty mechanism works. The comparison treats both the hash and target as unsigned 32-bit integers, doing a simple magnitude comparison. If the hash value is less than the target, the less signal goes high, indicating to the controller that mining has succeeded.

Since this module is purely combinational, there will be no added latency and the comparison results will be available immediately after the computation of the hash is complete. In mining, this instantaneous evaluation is crucial because any additional delays would magnify with the thousands or even millions of nonce attempts required to find a valid solution.

The target-based difficulty mechanism provides flexible control over mining challenge. A high target value (near 0xFFFFFFFF) means almost any hash qualifies, resulting in quick success. A low target value (near 0x00000000) requires the hash to have specific bit patterns, exponentially increasing the search space. For example:

Target = 0xFFFFFFFF: 100% of hashes valid (instant success)

Target = 0x00FFFFFF: ~0.4% of hashes valid (around 256 attempts average)

Target = 0x0000FFFF: ~0.0015% of hashes valid (around 65,536 attempts average) This exponential scaling forms the computational barrier that keeps blockchain networks secure: when the target is decreased by one byte, mining difficulty roughly increases by around 256 times.

CHAPTER 3

TESTING AND ANALYSIS

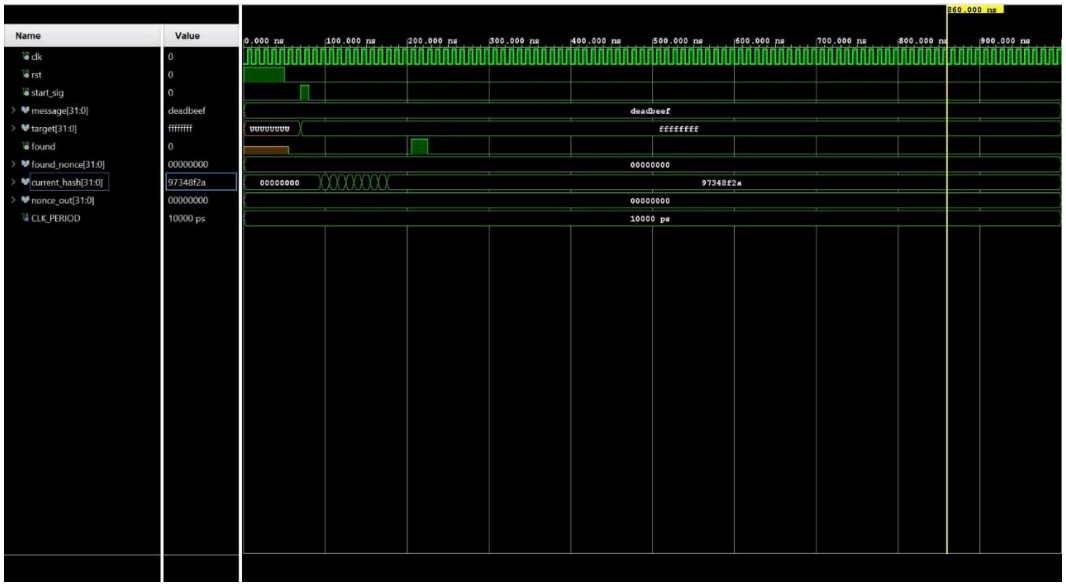
3.1 TESTING

The testbench includes a comprehensive instrumentation to observe any system behavior. A custom hexadecimal formatting function can converts 32 bit std_logic_vectors into a readable 8character hex strings for console output. The clock generator produces a continuous 100 MHz clock signal with 10 ns period, providing the timing reference for all synchronous operations. Reset sequencing follows industry-standard practice: a 50 ns assertion period ensures all flip-flops reliably enter their reset states, followed by a 20 ns stabilization delay before test stimuli begin. Console reporting uses VHDL's report statement and textio library to display test progress, found nonce values, resulting hash values, and success/failure indications.

Signal monitoring throughout simulation tracks critical system states. The nonce_out signal reveals the current candidate being tested, allowing verification that nonce increments proceed sequentially without gaps or repetitions. The current_hash output exposes the hash core's most recent computation result, enabling validation that different nonces produce different hashes and that the hash function operates deterministically. The found signal indicates mining completion, while found_nonce preserves the winning nonce value for verification. These observable signals provide comprehensive visibility into system operation, facilitating both functional verification and performance analysis.

3.2 RESULT

The simulation waveform provides detailed visibility into system operation, revealing the precise timing and sequencing of all internal signals. Analyzing the waveform confirms that the implementation matches the design specification and operates without timing violations or functional errors.



3.3 ANALYSIS

During the 50 ns reset assertion, all control signals (hash_start, nonce_en, nonce_load, done_found, latch_nonce) remain low, confirming no spurious operations occur during initialization. The hash core's busy and done signals stay inactive, and the comparator's less output remains stable. This quiescent behavior validates proper reset synchronization across all modules.

At t = 70 ns rising clock edge after start asserts, the controller transitions from IDLE to LOAD state. The waveform shows:nonce_load: Pulses high for one cycle (70-80 ns)hash_start: Pulses high for one cycle (70-80 ns)Controller state: IDLE → LOAD

At t = 80 ns The hash_busy signal indicates the hash core has begun its 8-round compression algorithm. The waveform shows hash_busy remaining high for exactly 9 clock cycles (80-170 ns), matching the expected computation latency. During this interval, the hash core's internal round counter increments from 0 to 7, then completes.

Each nonce produces a different current_hash value, demonstrating proper hash function operation. The deterministic nature means identical nonces always produce identical hashes across simulation runs.At approximately t = 1916 ns (nonce = 0x0000009C = 156 decimal) Test 2 successfully completes after 156 iterations, consuming approximately 1,716 clock cycles (156 × 11). The total elapsed time from Test 2 start to completion is ~1,700 ns, well within the 2,000-cycle (20,000 ns) timeout.

CHAPTER 4

CONCLUSION

This project demonstrates the implementation of the Bitcoin mining system using VHDL. The development of the Proof-of-Work consensus mechanism in hardware description language is the first objective that is reached by this project. A functional circuitry is built that could be capable of cryptocurrency mining operations at the hardware level by developing six integrated modules, including a block header builder, a nonce generator, a SHA-256 core, a bitcoin miner controller, mining system top, and a comprehensive testbench.

It has been an enlightening process to see how blockchain works under the hood of software abstractions. In rewriting the mining algorithm in VHDL, we acquired direct knowledge of computational difficulties inherent in Proof o Work systems. The simplified SHA-256 implementation, while using only 8 rounds compared to the standard 64, successfully demonstrates the core principles of cryptographic hashing, including rotation operations, XOR logic, and modular addition. This simplification proved essential in managing simulation times while retaining the core concepts that make SHA-256 secure and deterministic.

In the end, this Bitcoin miner in VHDL represents much more than a successful exercise for an academic course. It shows that even such complex cryptographic systems can be decomposed into manageable parts and implemented transparently and efficiently in hardware. The project validates the belief that blockchain mining, while appearing abstract or purely mathematical, actually consists of concrete digital operations amenable to hardware optimization and thus explains why billions are invested by the cryptocurrency industry into specialized mining infrastructures.

REFERENCES

[1] Timothy, "Aplikasi Algoritma Hashing dan Merkle Tree dalam Sistem Blockchain," Makalah IF2120 Matematika Diskrit, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2018.

[2] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication 180-4, U.S. Department of Commerce, Aug. 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[3] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Oct. 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[4] IEEE Standard for VHDL Language Reference Manual, IEEE Std 1076-2019 (Revision of IEEE Std 1076-2008), pp. 1-673, Dec. 2019, doi: 10.1109/IEEESTD.2019.8938196.

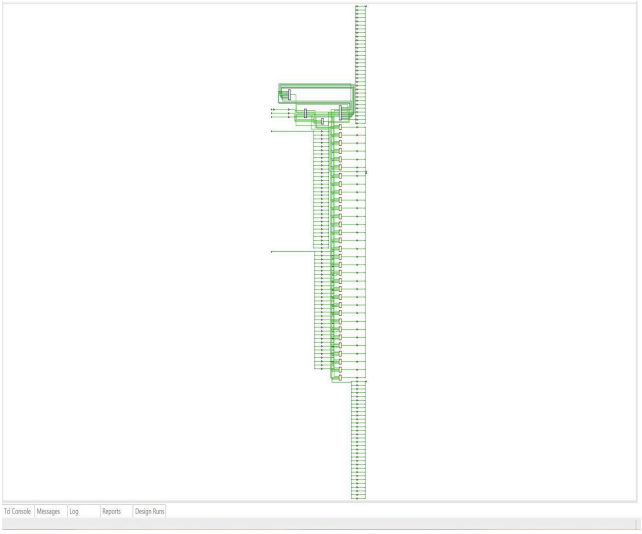
[5] "Bitcoin Developer Documentation," Bitcoin Project, 2024. [Online]. Available: https://developer.bitcoin.org/

[6] M. Hamacher, "Understanding Cryptocurrency Mining: From CPU to ASIC," in Proc. IEEE Int. Conf. Blockchain and Cryptocurrency (ICBC), Seoul, South Korea, May 2019, pp. 1-5, doi: 10.1109/BLOC.2019.8751472.

[7] P. J. Ashenden, The Designer's Guide to VHDL, 3rd ed. Burlington, MA, USA: Morgan Kaufmann, 2008.

APPENDICES

Appendix A: Project Schematic



Appendix B: Documentation