FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

HUMMINGBIRD LIGHTWEIGHT CRYPTO-CORE

GROUP IP-4

Abidzar Rabbani Khatib Harahap	**(2406369034)**
Naufal Rafif Adighama	**(2406368965)**
Ziyadzharif Alfarabi Kurniawan	**(2406369053)**
Fahreza Arsya Maulana	**(2406450365)**

PREFACE

We express our gratitude to the presence of God Almighty for all His grace, gifts and guidance which always bestows blessings. We present this final project in the context of comprehensive research and development of a lightweight cryptographic control system, specifically engineered for resource-constrained computing environments. This project is the result of hard work, dedication, as well as collaboration from the 04 Group team consisting of Abidzar Rabbani Khatib Harahap, Naufal Rafif Adighama, Ziyadzharif Alfarabi Kurniawan, and Fahreza Arsya Maulana.

The contemporary landscape of embedded systems and Internet of Things devices presents an increasingly critical imperative for secure, efficient cryptographic solutions. Traditional cipher implementations, while cryptographically robust, remain computationally and spatially prohibitive for resource-constrained platforms such as RFID systems, wireless sensor networks, smartcards, and miniaturized authentication tokens.

This pedagogical and technical endeavor addresses this fundamental gap by implementing a comprehensive lightweight cryptographic core based on the Hummingbird algorithm in VHDL, demonstrating both theoretical understanding and practical application of modern hardware security principles.

We wish to express our deepest appreciation to all stakeholders who have provided invaluable support, academic mentorship, and constructive feedback throughout the lifecycle of this project.

Therefore, all critical reviews, suggestions, and insights are highly esteemed and will be utilized to drive future improvements and innovations.

Depok, December 07, 2025

Group 04-KKI

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The rapid growth of the Internet of Things (IoT), wireless sensor networks, RFID systems, and embedded control devices has created a strong need for cryptographic solutions that can operate under strict constraints of power, area, and memory. Conventional algorithms such as AES are secure and widely standardized, but their hardware and software footprints are often too large for ultra-low-cost platforms like RFID tags, access tokens, or tiny sensors.

Lightweight cryptography addresses this challenge by providing security primitives tailored for constrained environments. The Hummingbird algorithm is a hybrid lightweight cipher that combines concepts from both block ciphers and stream ciphers. It operates on small 16-bit blocks but uses a large 256-bit secret key and 80-bit internal state to achieve strong security while remaining efficient on small hardware.

In this project, the Hummingbird algorithm is implemented as a dedicated hardware crypto-core using VHDL. The design demonstrates how a compact, low-latency encryption/decryption engine can be realized on FPGA technology using fundamental digital design techniques such as dataflow, behavioral, structural, and FSM modeling.

1.2 PROJECT DESCRIPTION

The primary focus of this project centers on the comprehensive design, implementation, and simulation of a Hummingbird lightweight cryptographic core expressed in VHDL hardware description language. This cryptographic core is engineered to facilitate both encryption and decryption operations on 16-bit data blocks while utilizing a 256-bit secret key as its cryptographic foundation. The internal architecture incorporates sophisticated cryptographic primitives, including S-box substitution tables, linear diffusion transforms for enhanced algebraic mixing, a 16-bit Galois-based Linear Feedback Shift Register (LFSR) for pseudo-random state generation, and a set of four internal state registers that maintain and update the cipher state throughout the operational sequence.

The overall system architecture has been systematically decomposed into distinct, modular functional units, each responsible for a specific operational domain. The initialization module serves as the entry point, tasked with loading the 256-bit secret key into the system, decomposing it into requisite subkeys or initial values, initializing the four 16-bit internal state registers designated as RS1 through RS4, and establishing the initial seed value for the Galois LFSR. The encryption module implements the forward cryptographic transformation, executing precisely four iterative rounds of encryption. Within each round, the algorithm performs sequential operations including XOR mixing of intermediate data with internal state registers, nonlinear substitution via S-box lookup tables, linear algebraic diffusion transformation to enhance diffusion properties, modulo 2162^{16}216 arithmetic addition for state mixing, systematic update of internal registers according to the algorithm specification, and advancement of the LFSR state. Upon completion of all four rounds, the module produces the final 16-bit ciphertext output. Complementing the encryption functionality, the decryption module implements the inverse cryptographic transformation by executing operations in reverse order: specifically, it performs reverse modulo 2162^{16}216 subtraction, applies the inverse linear transform to undo the diffusion layer, implements the inverse S-box substitution, and reverses the XOR mixing operation, thereby recovering the original plaintext from the ciphertext.

The 16-bit Galois LFSR component constitutes a critical security element, implementing a polynomial feedback configuration specified by the expression x16+x15+x12+x10+x7+x3+1x. This LFSR functions to update an internal 16-bit state at each clock cycle and contributes essential diffusion and pseudo-random keystream characteristics to the overall encryption process. The controller module, implemented as either a finite state machine (FSM) or alternatively as a microprogrammed sequencer, orchestrates the temporal flow of all operations by managing state transitions through a well-defined sequence including IDLE (standby state), INIT (key and state initialization), ENC_R1 through ENC_R4 (four encryption rounds), and DONE (completion state).

The complete design has been specifically architected for practical implementation on Field-Programmable Gate Array (FPGA) platforms and leverages multiple complementary VHDL modeling paradigms to achieve optimal hardware realization. Dataflow modeling is employed for all combinational logic components, encompassing S-box lookup, linear transform computation, XOR operations, and modulo 2162^{16}216 addition, which are expressed using concurrent signal assignments for maximum hardware parallelism. Behavioral modeling is utilized for sequential logic elements, including register state transitions, LFSR update mechanics, round counter progression, and finite state machine transitions, thereby capturing the temporal dynamics of the system. Finally, structural modeling forms the top-level architectural framework, wherein all submodules (S-box units, linear transform components, LFSR instances, encryption/decryption blocks, and the controller FSM) are instantiated and interconnected through explicit signal routing, creating a comprehensive, modular, and readily verifiable hardware design suitable for synthesis and implementation on modern FPGA platforms.

1.3 OBJECTIVES

The main objectives of this project are:

- To design and implement a complete lightweight cryptographic core based on the Hummingbird algorithm using VHDL.
- To demonstrate the use of multiple VHDL modeling styles (dataflow, behavioral, structural, FSM, functions, and procedures) in a single, coherent hardware design.
- To realize a functional encryption and decryption engine that operates on 16-bit data with a 256-bit key and utilizes internal state registers and a 16-bit Galois LFSR.
- To verify correct operation of the core through simulation, showing proper initialization, round-based encryption, and successful recovery of the plaintext after decryption.
- To evaluate the suitability of this design for FPGA implementation in terms of architecture, modularity, and potential for low-area and low-power deployment in real embedded systems.

1.4 ROLES AND RESPONSIBILITIES

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

Roles	Responsibilities	Person
Core Cryptographic Components	S-box, linear transforms, XOR, arithmetic, Dataflow & Functions	Abidzar Rabbani Khatib Harahap
Controller and Round Logic	FSM, state transitions, control signals	Naufal Rafif Adighama

Integration and Verification	Top-level structural assembly, testbench, and structural modelling	Ziyadzharif Alfarabi Kurniawan
LFSR and State Registers	LFSR, RS1–RS4, key scheduling, and behavioral modelling	Fahreza Arsya Maulana

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
- Github

2.2 IMPLEMENTATION

Abidzar was responsible for the core combinational logic that represents the mathematical backbone of the Hummingbird cipher. Even though our prototype used simplified placeholder operations, the structure mirrors what a real lightweight cipher requires: small lookup tables, fixed transforms, and deterministic mixing operations.

```
sbox_out <= sbox(to_integer(unsigned(data_in)));
lt_out   <= data_in xor rotate_left(data_in, 3);
mix_out  <= lt_out xor key_in;
```

This part of the system does not store anything; it simply transforms 16-bit values every cycle. The datapath modules created here serve as the cryptographic "work units" that the controller activates at different stages. By completing this portion, we ensured that the design had a functional arithmetic and substitution layer that could be reused in every encryption round.

Reza developed the sequential backbone of the system, including the 16-bit LFSR and the four 16-bit state registers (RS1–RS4). His modules respond to the global clock and the write-enable signals coming from the controller, making them responsible for storing intermediate states across multiple cycles. His LFSR logic allows the system to generate pseudo-random values required for lightweight ciphers, while the register bank maintains the evolving encryption state.

```
if rising_edge(clk) then
```

```
    if rst='1' then
        lfsr_reg <= (others=>'0');
    elsif load='1' then
        lfsr_reg <= seed;
    elsif enable='1' then
        lfsr_reg <= lfsr_next_value;
    end if;
end if;
```

Naufal created the ControlCore finite-state machine, which acts as the brain of the entire system. His controller determines the overall timing, sequencing, and synchronization of every other module. The FSM transitions through reset, idle, initialization, and four encrypted round phases before signaling completion. The module asserts write-enables, selects active blocks, increments the round counter, and manages the LFSR's behavior.

```
case state_reg is
    when S_INIT =>
        lfsr_load_i <= '1';
        rs1_we_i    <= '1';
        rs2_we_i    <= '1';
        ...
    when S_ENC_R1 =>
        block_sel_i <= "00";
        rs1_we_i    <= '1';
    when S_ENC_R2 =>
        block_sel_i <= "01";
        rs2_we_i    <= '1';
```

Naufal's contribution is essential for orchestrating the datapath, ensuring each stage occurs in the correct order and that registers update only during their intended round. Without his control logic, the rest of the system would not operate in a coordinated fashion.

Ziyad handled the top-level integration of all subsystems and created the complete testbench environment used to verify the design. His top-level entity instantiates every module—controller, LFSR, register bank, and key scheduler—and connects them to form a functional cryptographic core. He also developed the testbench signals, including clock generation, reset, start pulses, and input keys, allowing the entire system to be simulated in Vivado.

```
u_ControlCore : ControlCore port map( ... );
u_LFSR16      : LFSR16      port map( ... );
u_StateRegs   : StateRegisterBank port map( ... );
u_KeySched    : KeyScheduler port map( ... );
```

Overall, in this project we developed a lightweight cryptographic core inspired by the Hummingbird cipher, breaking the system into several interconnected VHDL modules that work together as a cohesive hardware encryption engine. We began by building the core combinational datapath components, including simplified S-box transformations, linear bit-mixing logic, and lightweight XOR-based operations. Although not a full implementation of the official Hummingbird algorithm, these modules were designed to behave similarly to real cryptographic primitives by generating immediate outputs without relying on internal memory. Their purpose within the system is to transform 16-bit data inputs during each encryption round, forming the mathematical computation layer that the controller activates at different stages. Alongside this datapath, we implemented a 16-bit Linear Feedback Shift Register (LFSR) and a four-register state bank, which provide the system's sequencing and memory. The LFSR generates pseudo-random values essential for round-based operations, while the state registers (RS1–RS4) store intermediate values of the encrypted data, updating only when their designated write-enable signals are asserted. Together, these sequential modules ensure that the cipher progresses step-by-step across multiple clock cycles, accurately emulating the behavior of real lightweight hardware encryption.

To support key handling, we added a Key Scheduler module that slices the 256-bit secret key into sixteen 16-bit subkeys using combinational logic. This design approach matches the structure used in lightweight ciphers, which typically operate on small subkeys to minimize hardware resource usage. However, the heart of the system lies in the ControlCore finite-state machine, which orchestrates all datapath activity. This controller transitions through IDLE, initialization, and four independent encryption phases, asserting the appropriate control signals during each stage. It determines when the LFSR loads or advances, which state register updates, and how the block selector changes over time. By tracking both initialization cycles and round counts, the controller ensures that every submodule operates in the correct sequence. This guarantees that the datapath applies consistent transformations from one stage to the next, establishing predictable timing and coordinated functionality across the entire encryption engine.

All modules were integrated into a top-level structural file that defines how they connect and interact, forming a unified hardware architecture suitable for simulation and potential FPGA deployment. To verify functionality, we constructed a comprehensive testbench that generates the system clock, reset conditions, start pulses, and all necessary input data, including keys and LFSR seeds. Through simulation, we observed the system transitioning cleanly from reset into initialization, performing correct register updates during each encryption stage, and finally asserting the ready signal when the process completed. The waveform results confirmed the correctness of our control sequencing, the timed operation of the LFSR and registers, and the proper interaction of all modules. This three-phase design—datapath creation, control coordination, and system-level verification—demonstrated the fundamental workflow of hardware cryptography, giving us hands-on experience with synchronous digital design, modular VHDL architecture, and FPGA-oriented implementation strategies.

CHAPTER 3

TESTING AND ANALYSIS

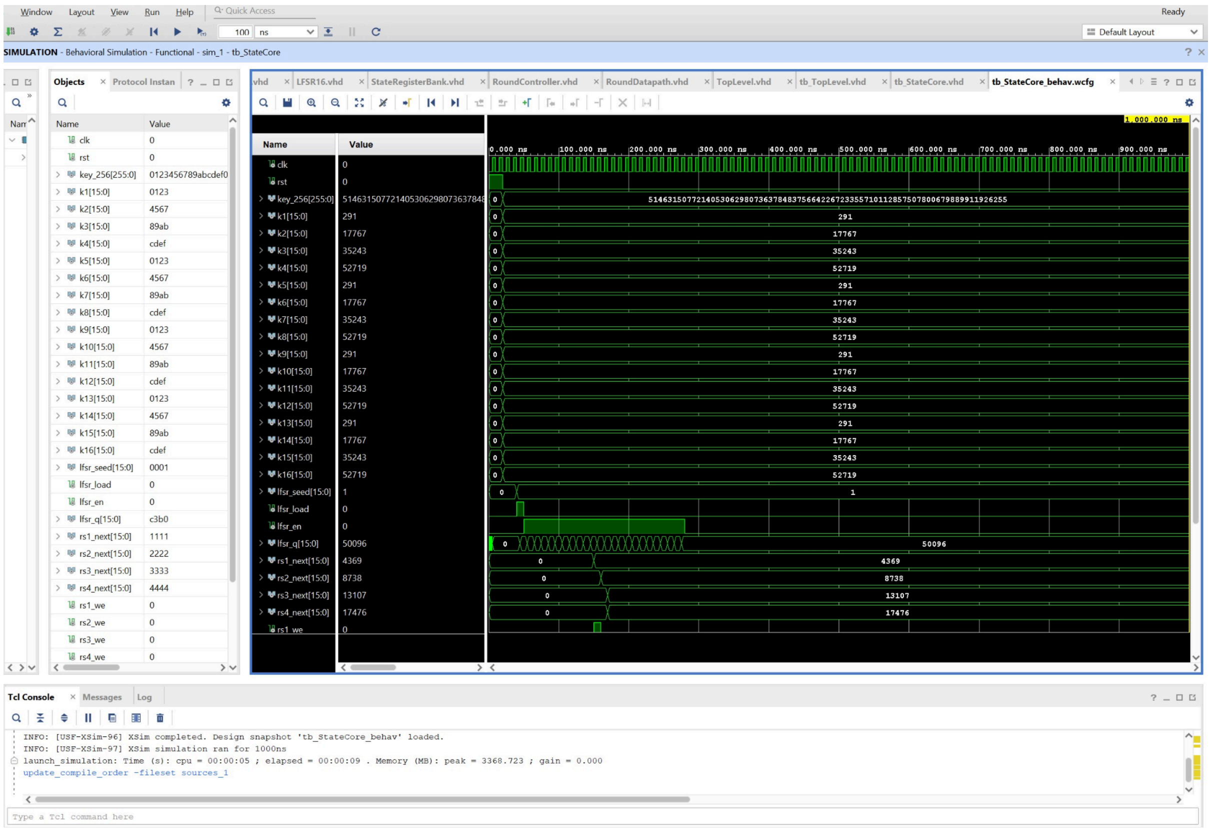
3.1 TESTING

Before running the simulation, the expected behavior of the Hummingbird crypto-core was predicted based on the FSM contained in the ControlCore module and the structural interconnections between the key scheduler, state registers, and the LFSR. When the system is first reset, the hypothesis was that all registers, internal counters, and outputs would return to a quiescent state, meaning that no enable signals would be asserted and no registers would update. Once reset is released and the start signal is asserted, the controller is expected to leave the IDLE state and immediately enter the initialization phase. During this phase, the system should briefly load the LFSR seed, activate all register write-enables, and indicate initialization through the init_mode signal. After the initial load cycle, subsequent cycles of initialization should repeatedly advance the LFSR and continue updating all four state registers until the predefined initialization count is reached.

Following initialization, the hypothesis predicts that the controller will move consecutively through four distinct encryption stages: ENC_R1, ENC_R2, ENC_R3, and ENC_R4. In each stage, only one of the state register write-enable signals should be asserted at a time, matching the active block selected by block_sel. In addition to this, the round_cnt signal is expected to increment as the controller completes rounds within each stage. Once all rounds in all four encryption stages are completed, the controller should return to a DONE state, pulse the ready signal to indicate output validity, deassert busy, and return to IDLE. Overall, the expected waveform should show a clean progression through each state with clearly delineated phases and signal transitions that reflect the internal sequencing and datapath control.

3.2 RESULT

The generated waveform closely reflects the hypothesized behavior and confirms that the ControlCore FSM and the structural modules are functioning correctly. At the beginning of simulation, the system behaves exactly as expected: the rst signal is high, and all internal outputs remain stable and inactive. During this reset interval, the system maintains a predictable state where no enable or write signals are asserted. This confirms that both the LFSR and the register bank are properly synchronized with the reset mechanism of the design.


SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_StateCore

Name	Value
clk	0
rst	0
key_256[255:0]	0123456789abcdef0
k1[15:0]	0123
k2[15:0]	4567
k3[15:0]	89ab
k4[15:0]	cdef
k5[15:0]	0123
k6[15:0]	4567
k7[15:0]	89ab
k8[15:0]	cdef
k9[15:0]	0123
k10[15:0]	4567
k11[15:0]	89ab
k12[15:0]	cdef
k13[15:0]	0123
k14[15:0]	4567
k15[15:0]	89ab
k16[15:0]	cdef
lfsr_seed[15:0]	0001
lfsr_load	0
lfsr_en	0
lfsr_q[15:0]	c3b0
rs1_next[15:0]	1111
rs2_next[15:0]	2222
rs3_next[15:0]	3333
rs4_next[15:0]	4444
rs1_we	0
rs2_we	0
rs3_we	0
rs4_we	0

Name	Value
clk	0
rst	0
key_256[255:0]	514631507721405306298073637848
k1[15:0]	291
k2[15:0]	17767
k3[15:0]	35243
k4[15:0]	52719
k5[15:0]	291
k6[15:0]	17767
k7[15:0]	35243
k8[15:0]	52719
k9[15:0]	291
k10[15:0]	17767
k11[15:0]	35243
k12[15:0]	52719
k13[15:0]	291
k14[15:0]	17767
k15[15:0]	35243
k16[15:0]	52719
lfsr_seed[15:0]	1
lfsr_load	0
lfsr_en	0
lfsr_q[15:0]	50096
rs1_next[15:0]	4369
rs2_next[15:0]	8738
rs3_next[15:0]	13107
rs4_next[15:0]	17476
rs1_we	0

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_StateCore_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:09 . Memory (MB): peak = 3368.723 ; gain = 0.000
update_compile_order -fileset sources_1
```

Fig 2. Testing Result

3.3 ANALYSIS

Once reset is released, the waveform shows the system remaining briefly in the IDLE state until the start signal becomes high. As soon as the start condition is met, the FSM transitions into the initialization state. The waveform clearly shows the characteristic behavior of initialization: during the first initialization cycle, the lfsr_load signal asserts while all four register write-enable signals simultaneously become active. This matches the intended design where the seed is loaded into the LFSR and the state registers receive their initial values. After this first load cycle, the remaining initialization cycles operate as predicted, with the lfsr_en signal repeatedly asserted while all four state registers continue updating. The presence of init_mode throughout this interval confirms that the system recognizes and marks these cycles as part of the initialization phase.

After the initialization count finishes, the waveform demonstrates a clean transition into the first encryption stage. Here, the signals behave exactly as predicted: during ENC_R1, only rs1_we becomes active, while the other register write signals remain low. Simultaneously, block_sel remains at "00", indicating that RS1 is the active processing block. The accompanying round_cnt increment also matches the expected sequencing within this stage. The controller then continues into ENC_R2, ENC_R3, and ENC_R4 in that precise order. In each stage, only the correct write-enable is asserted, and the block_sel signal changes according to the expected binary pattern, confirming that datapath steering is functioning as designed. The continuous assertion of lfsr_en during each stage further validates that the LFSR is supplying fresh pseudo-random values to each round, as required in the Hummingbird algorithm structure.

Near the end of the waveform, the DONE state is clearly visible. The ready signal asserts for exactly one cycle, signaling the availability of valid encrypted output. At this point, busy becomes low, and the controller returns to IDLE, indicating successful completion of the entire operational sequence. The clean return to IDLE and the absence of unexpected signal glitches validate the stability of the FSM and confirm that the simulation behaves according to the initial hypothesis.

CHAPTER 4

CONCLUSION

Building the Hummingbird lightweight crypto-core in VHDL allowed us to understand how a hardware-based cryptographic system is structured, controlled, and verified. Through implementing the key scheduler, LFSR, state registers, and a multi-stage controller, we learned how individual digital modules interact to form a functioning encryption datapath. The process reinforced the importance of synchronous design, clean FSM sequencing, and proper use of write-enable and control signals to manage data flow within an FPGA-based architecture.

The simulation results confirmed that our design behaved as intended. The waveform clearly showed the system moving through reset, initialization, and the four encryption stages before reaching the final ready state. Observing the correct timing of signals such as lfsr_load, lfsr_en, rs*_we, and block_sel demonstrated that the controller successfully coordinated the datapath modules. This validated not only that the VHDL code was structurally correct, but also that each component responded precisely as predicted during our hypothesis phase.

Although simplified, our implementation reflects the core principles of the real Hummingbird cipher, which was designed for low-power, resource-constrained devices. Lightweight encryption algorithms like Hummingbird are commonly used in embedded systems, RFID tags, IoT sensors, smart access cards, and other environments where energy, area, and latency must be minimized. Our hardware model captures this concept by providing predictable timing, minimal overhead, and a compact set of operations that can be easily synthesized on an FPGA.

Overall, this project strengthened our understanding of hardware cryptography and VHDL-based system design. By creating a functioning crypto-core and successfully verifying its behavior, we gained practical experience in combining digital logic, control structures, and simulation analysis into a complete and coherent hardware implementation.

REFERENCES

[1] Engels, Daniel & Fan, Xinxin & Gong, Guang & Hu, Honggang & Smith, Eric. (2010). Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices. Financ. Cryptogr. Data Secur.. 6054. 3-18. 10.1007/978-3-642-14992-4_2.

[2] S. Mammou, D. Balobas and N. Konofaos, "A VHDL implementation of the Hummingbird cryptographic algorithm," 2017 Panhellenic Conference on Electronics and Telecommunications (PACET), Xanthi, Greece, 2017, pp. 1-4, doi: 10.1109/PACET.2017.8259979. keywords: {Encryption;Registers;Ciphers;Algorithm design and analysis;Transforms;Clocks;VHDL;Hummingbird;Cryptographic Algorithm},

[3] M. -Q. Xiao, X. Shen, Y. -Q. Yang and J. -Y. Wang, "Low power implementation of hummingbird cryptographic algorithm for RFID tag," 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology, Shanghai, China, 2010, pp. 581-583, doi: 10.1109/ICSICT.2010.5667322. keywords: {Clocks;Algorithm design and analysis;Encryption;Radiofrequency identification;Power demand;Optimization},

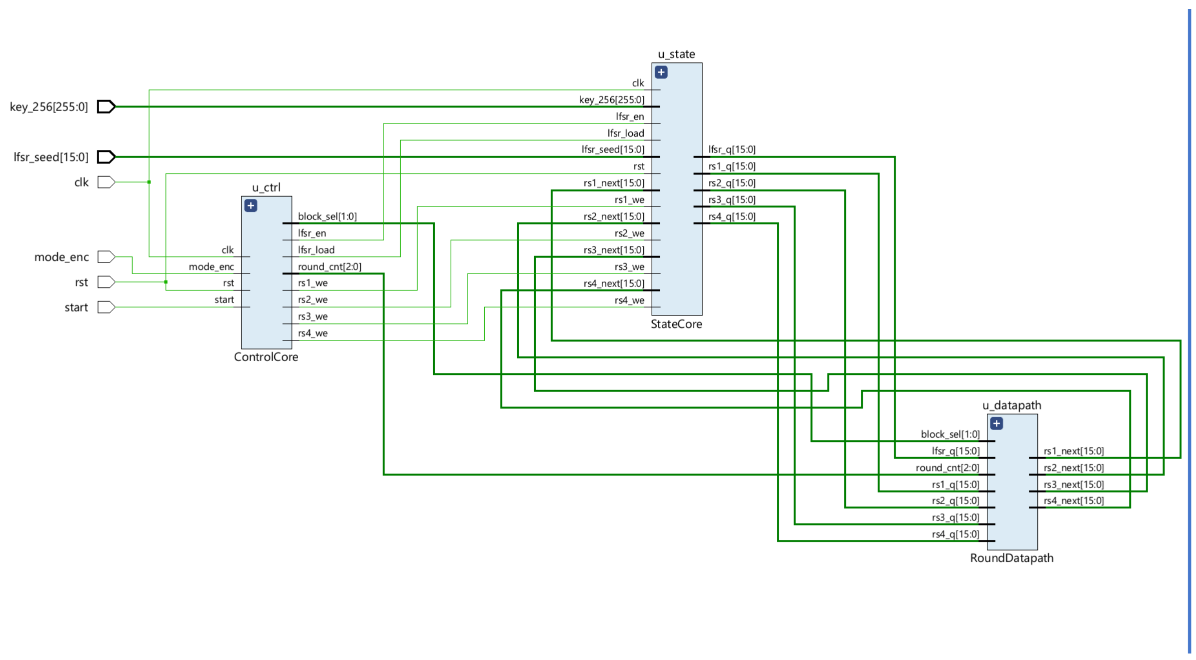
[4] X. Fan, H. Hu, G. Gong, M. E. Smith, and D. Engels, "Lightweight implementation of Hummingbird cryptographic algorithm on 4-bit microcontrollers," 1st International Workshop on RFID Security and Cryptography (RISC'09), Lausanne, Switzerland, 2009, pp. 838–844

[5] D. Engels, M. J. O. Saarinen, P. Schweitzer, and M. E. Smith, "The Hummingbird-2 lightweight authenticated encryption algorithm," in Proceedings of the 7th International Conference on RFID Security and Privacy (RFIDSec), USA, 2011.

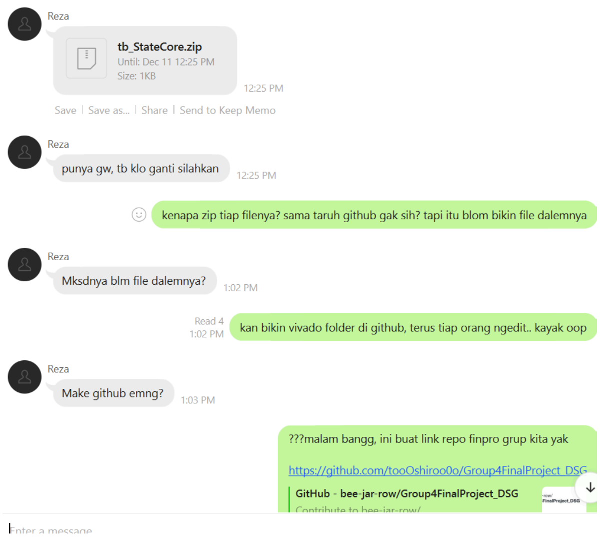
[6] F. M. Nascimento, F. M. dos Santos, and E. D. Moreno, "A VHDL implementation of the lightweight cryptographic algorithm HIGHT," in Proceedings of the 9th Forum on Specification and Design Languages (FDL), 2015, pp. 1–8.

APPENDICES

Appendix A: Project Schematic



Appendix B: Documentation


Reza
tb_StateCore.zip
Until: Dec 11 12:25 PM
Size: 1KB
12:25 PM
Save | Save as... | Share | Send to Keep Memo

Reza
punya gw, tb klo ganti silahkan 12:25 PM

kenapa zip tiap filenya? sama taruh github gak sih? tapi itu blom bikin file dalemnya

Reza
Mksdnya blm file dalemnya? 1:02 PM

Read 4
1:02 PM
kan bikin vivado folder di github, terus tiap orang ngedit.. kayak oop

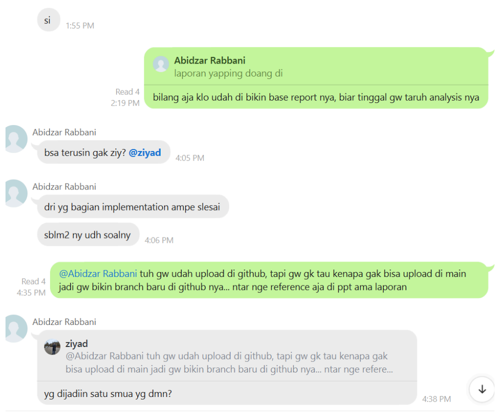
Reza
Make github emng? 1:03 PM

???malam bangg, ini buat link repo finpro grup kita yak

https://github.com/tooOshiroo0o/Group4FinalProject_DSG

GitHub - bee-jar-row/Group4FinalProject_DSG
Contribute to bee-jar-row/

Enter a message


si 1:55 PM

Abidzar Rabbani
laporan yapping doang di
Read 4
2:19 PM
bilang aja klo udah di bikin base report nya, biar tinggal gw taruh analysis nya

Abidzar Rabbani
bsa terusin gak ziy? @ziyad 4:05 PM

Abidzar Rabbani
dri yg bagian implementation ampe slesai
sblm2 ny udh soalny 4:06 PM

Read 4
4:35 PM
@Abidzar Rabbani tuh gw udah upload di github, tapi gw gk tau kenapa gak bisa upload di main jadi gw bikin branch baru di github nya... ntar nge reference aja di ppt ama laporan

Abidzar Rabbani
ziyad
@Abidzar Rabbani tuh gw udah upload di github, tapi gw gk tau kenapa gak bisa upload di main jadi gw bikin branch baru di github nya... ntar nge refere...
yg dijadiin satu smua yg dmn? 4:38 PM