ELEC-H409

Digital Architecture and Design

Implementing AES Encryption on an FPGA

Authors : (Group 4)

Lucas Placentino

Mehrdad Gholamnejad

Teaching Team:

Dragomir Milojevic

Lucas Stefanidis

Navid Ladner

2025

Contents

1 Introduction

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm that is largely used today to protect digital data. It is used for many purposes such as securing communications, data storage, and embedded systems. AES works with a fixed size of data blocks and a secret key, and provides a high level of security with low hardware complexity.

The goal of this project is that to implement an AES-128 encryption architecture using VHDL hardware description language. The encryption is applied on 128 bit plaintext blocks using a 128 bit key. The AES algorithm consists of four main steps which are AddRoundkey, SubBytes, ShiftRows and MixedColumns. In order to generate the last final encrypted output, these four steps will be executed several times in different encryption rounds.

In this project, each AES operation is implemented as a seperate VHDL module. All modules are first validated independently using test benches and provided test vectors. After that, the full AES encryption is built by connecting these modules together and controlling them with finite state machines. The architecture is designed such that one AES operation is executed at each clock cycle, which makes the behavior fast and easy to analyze and verify.

Finally, the AES encryption system is implemented on a Digilent Basys3 FPGA board. The encryption process is started using a push button, and once the encryption is finished, the characters *AES* are shown on a seven-segment display and LEDs are turned on. Both simulation and hardware tests are used to verify that the design works correctly.

2 Design & Implementation

In this chapter, we will show the block diagram of each logic module, their generated schematic after synthesis, and their test-bench design.

2.1 AddRoundKey step (`add_round_key.vhd`)

2.1.1 Logic

Block Diagram

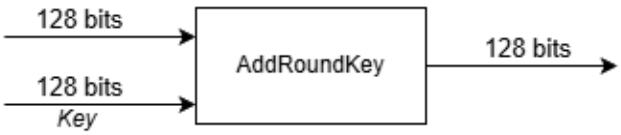

Figure 2.1: AddRoundKey block diagram

The AddRoundKey block has two inputs, one is the input vector and the other the corresponding round key, and one output vector. The block diagram can be seen Figure 2.1.

Generated Schematic

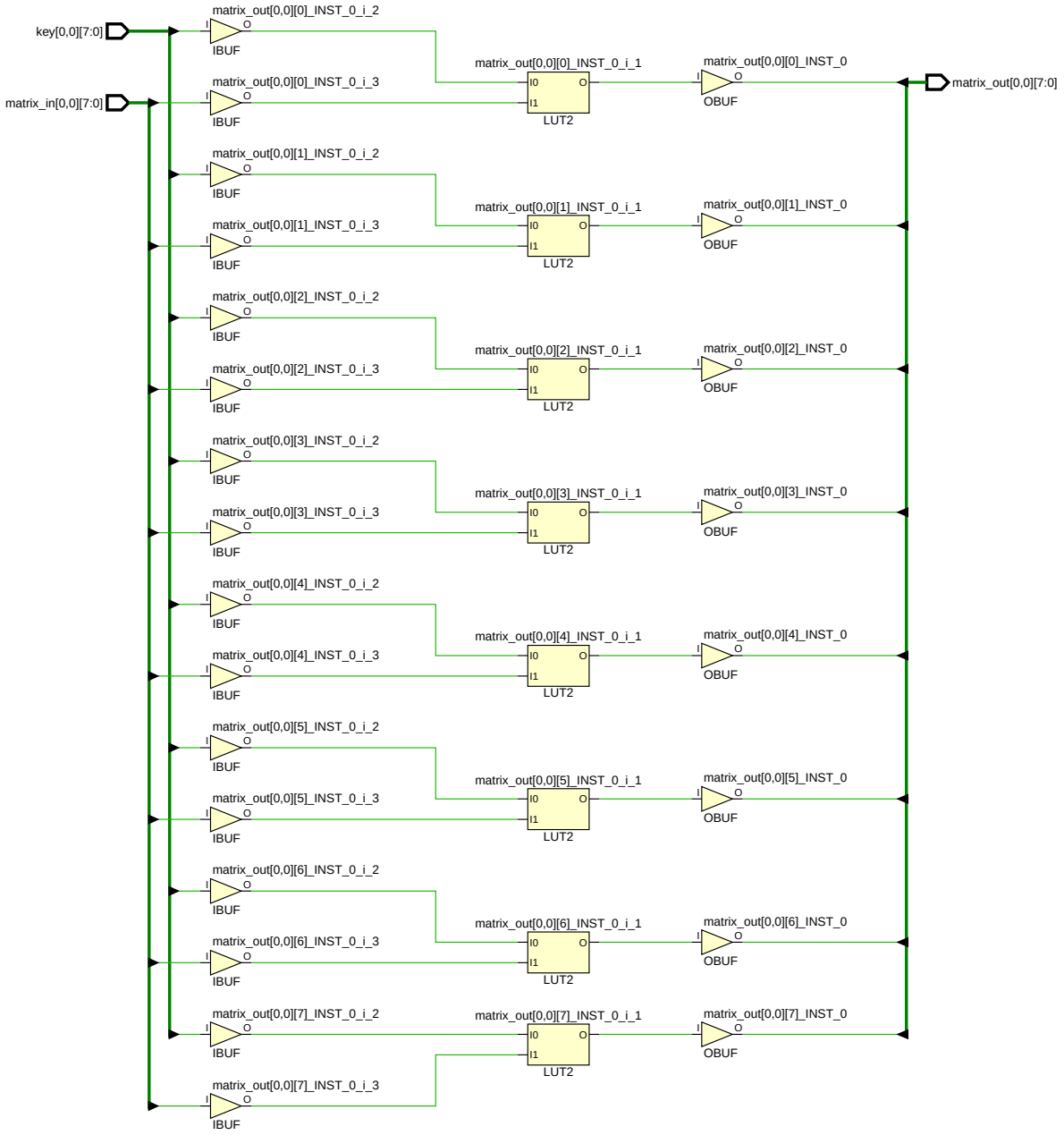

Figure 2.2: Part of synthesized schematic of AddRoundKey (Matrix element i=1,j=1)

After running the synthesis of AddRoundKey as top module, we get the schematic on Figure 2.2. We can clearly see both inputs (input vector and key), and the output. We observe that it simply contains a LUT.

2.1.2 Test-bench

For the test-bench of this module, we first create a sanity check with null matrices as input and key where we expect to get a null matrix as the output. The other checks are done using the keys provided in the project guide and using example input vectors from NIST's AES_Core128.pdf Block Cipher Modes of Operation Electronic Codebook (ECB) document [NIS12].

2.2 SubBytes step (`sub_byte.vhd`)

This uses the provided `s_box.vhd` file as a component.

2.2.1 Logic

Block Diagram

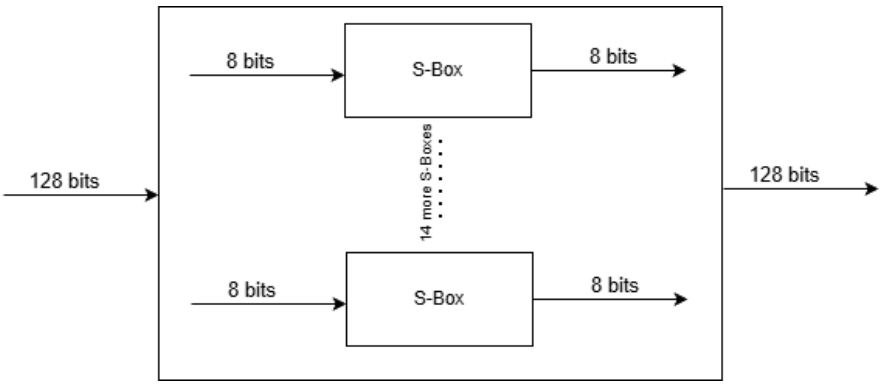

Figure 2.3: SubBytes block diagram

The SubBytes block has one input vector and one output vector. It contains 16 of the provided S_box sub-modules (each having one 8-bit input and one 8-bit output).

Generated Schematic

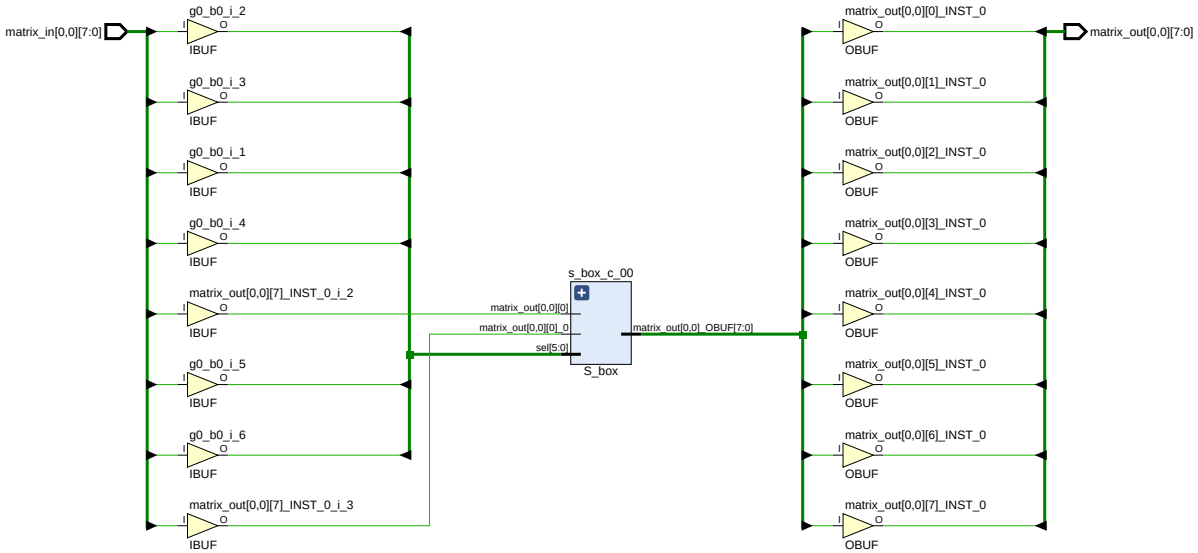

Figure 2.4: Part of synthesized schematic of SubBytes, S_box submodule on Figure 2.5 (Matrix element i=1,j=1)

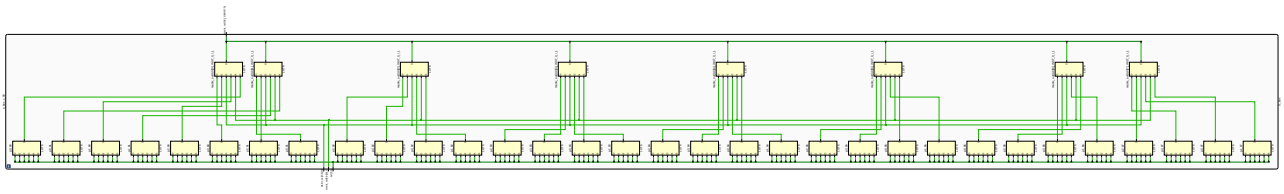

Figure 2.5: Synthesized schematic of S_box

After running the synthesis of SubBytes as top module, we get the schematic on Figure 2.4. We can clearly see both the vector input and the vector output. We observe that it simply connects them to S_box submodules (which itself can be seen Figure 2.5).

2.2.2 Test-bench

For the test-bench of this module, we first create a sanity check with null matrix as input where we expect to get a `x"63636363_63636363_63636363_63636363"` vector (converted back to a matrix) as the output. The other checks are done using example input vectors from NIST's AES_Core128.pdf Block Cipher Modes of Operation Electronic Codebook (ECB) document [NIS12].

2.3 ShiftRows step (`shift_rows.vhd`)

2.3.1 Logic

Block Diagram

Figure 2.6: ShiftRows block diagram

The ShiftRows block simply has one input vector and one output vector.

Generated Schematic

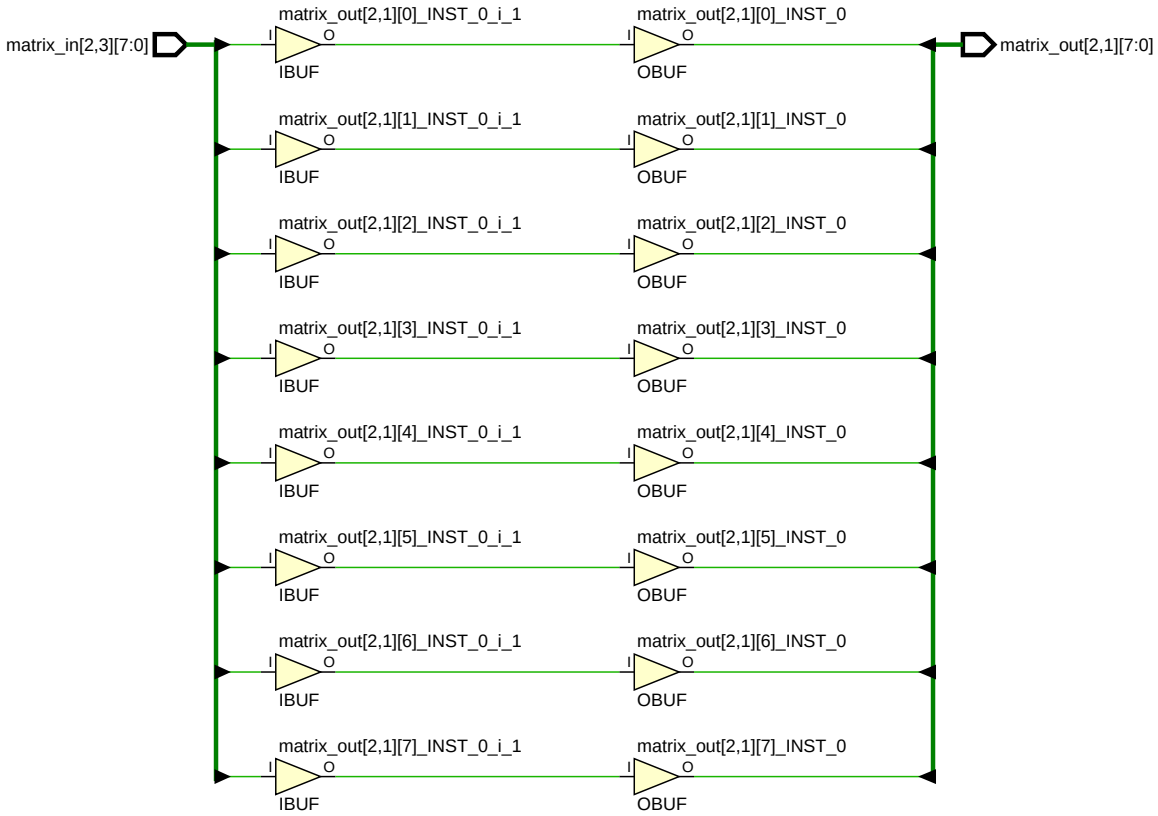

Figure 2.7: Part of synthesized schematic of ShiftRows (Matrix element i=3,j=4 to Matrix element i=3,j=2)

After running the synthesis of ShiftRows as top module, we get the schematic on Figure 2.7. We can clearly see both the vector input and the vector output. We observe that it simply connects an input cell to a different output cell, depending on the row.

2.3.2 Test-bench

For the test-bench of this module, we first create a sanity check with null matrix as input where we expect to get a null matrix as the output. The other checks are done using example input vectors from NIST's AES_Core128.pdf Block Cipher Modes of Operation Electronic Codebook (ECB) document [NIS12].

2.4 MixColumns step (`mix_columns.vhd`)

This uses the provided `LUT_mul2.vhd` and `LUT_mul3.vhd` files as components.

2.4.1 Logic

Block Diagram

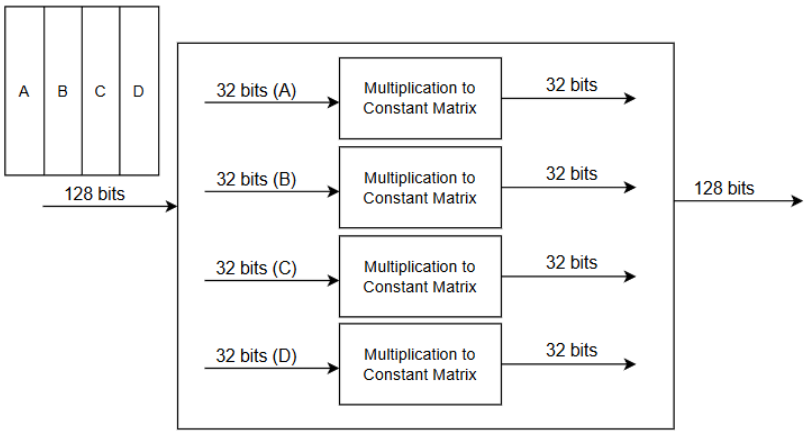

Figure 2.8: MixColumns block diagram

The MixColumns block has one input vector and one output vector. It contains 4 submodules, one for each column (each having one 32-bit input and one 32-bit output). This submodule is made from both the `LUT_mul2.vhd` and `LUT_mul3.vhd` provided modules.

Generated Schematic

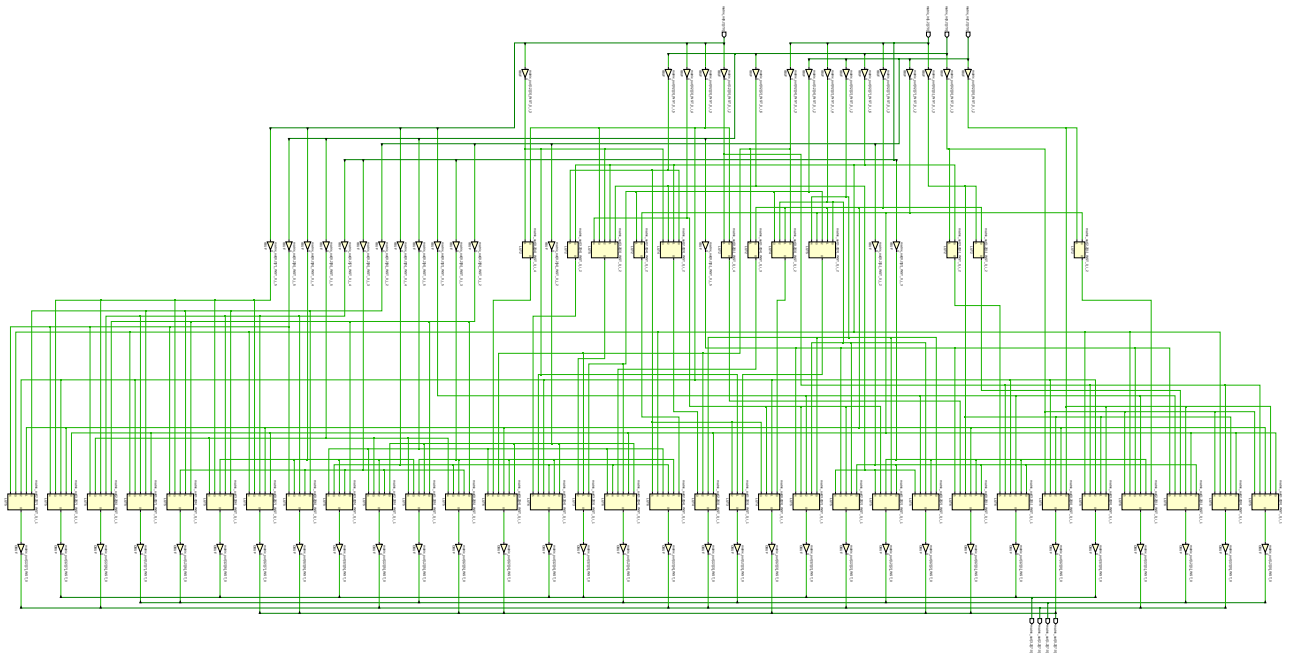

Figure 2.9: Part of synthesized schematic of MixColumns (Matrix column 3)

After running the synthesis of MixColumns as top module, we get the schematic on Figure 2.9. We can clearly see both the vector input (here a single column) and the vector output (also here a single column. We observe that it simply connects the input to both LUT_mul2 and LUT_mul3 to the output, for each column.

2.4.2 Test-bench

For the test-bench of this module, we first create a sanity check with null matrix as input where we expect to get a null matrix as the output. The other checks are done using example input vectors from NIST's AES_Core128.pdf Block Cipher Modes of Operation Electronic Codebook (ECB) document [NIS12].

2.5 AES Encryptor (`aes_enc.vhd`)

2.5.1 Logic

Block Diagram

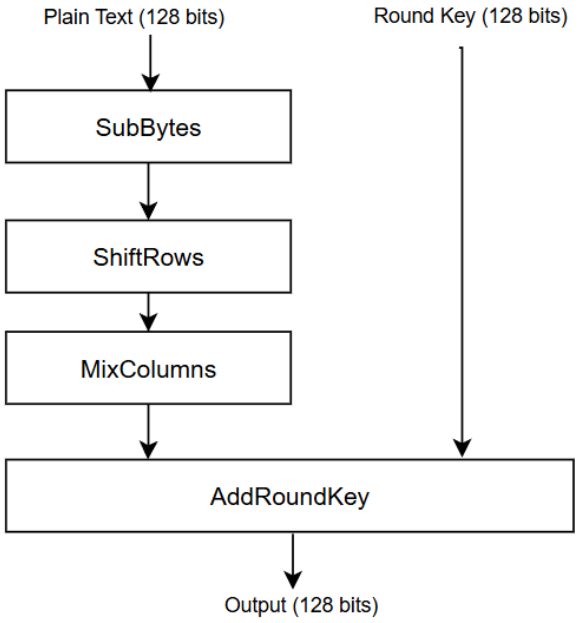

Figure 2.10: One Round of AES Block

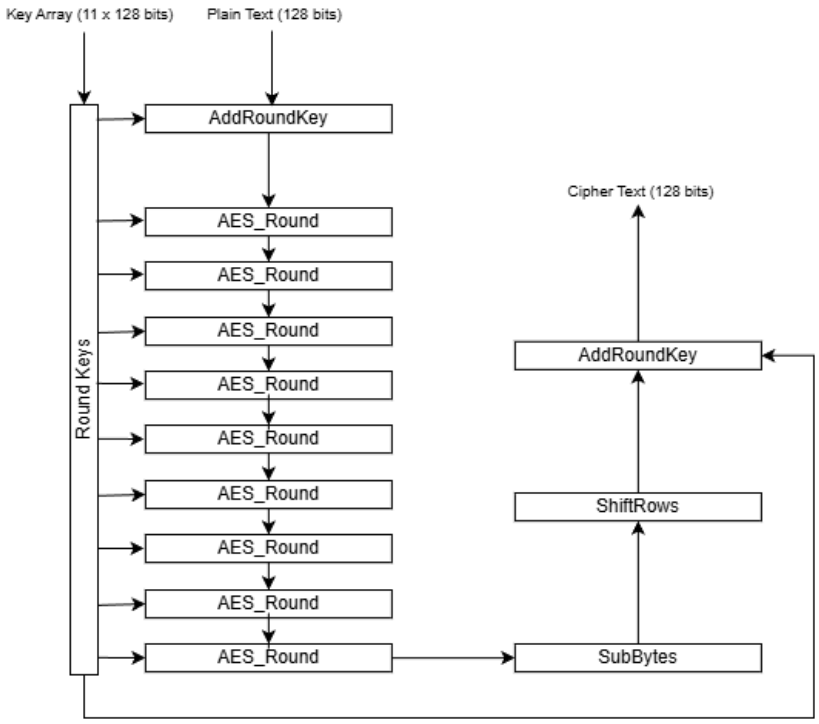

Figure 2.11: AES block diagram

The AES Encryptor block has two inputs, one plain text vector input and one array of 11 keys, and one output vector (the cipher text). It contains the first AddRoundKey step, then 9 rounds of 4 steps, followed by the last 3 steps (SubBytes, ShiftRows, then the final AddRoundKey). Every AddRoundKey step (including the on in each round) gets a key from the key array.

Generated Schematic

This schematic is so big that it is separated into 11 pages, Figure 2.12 shows the 9th page, which includes the `sub_byte` component.

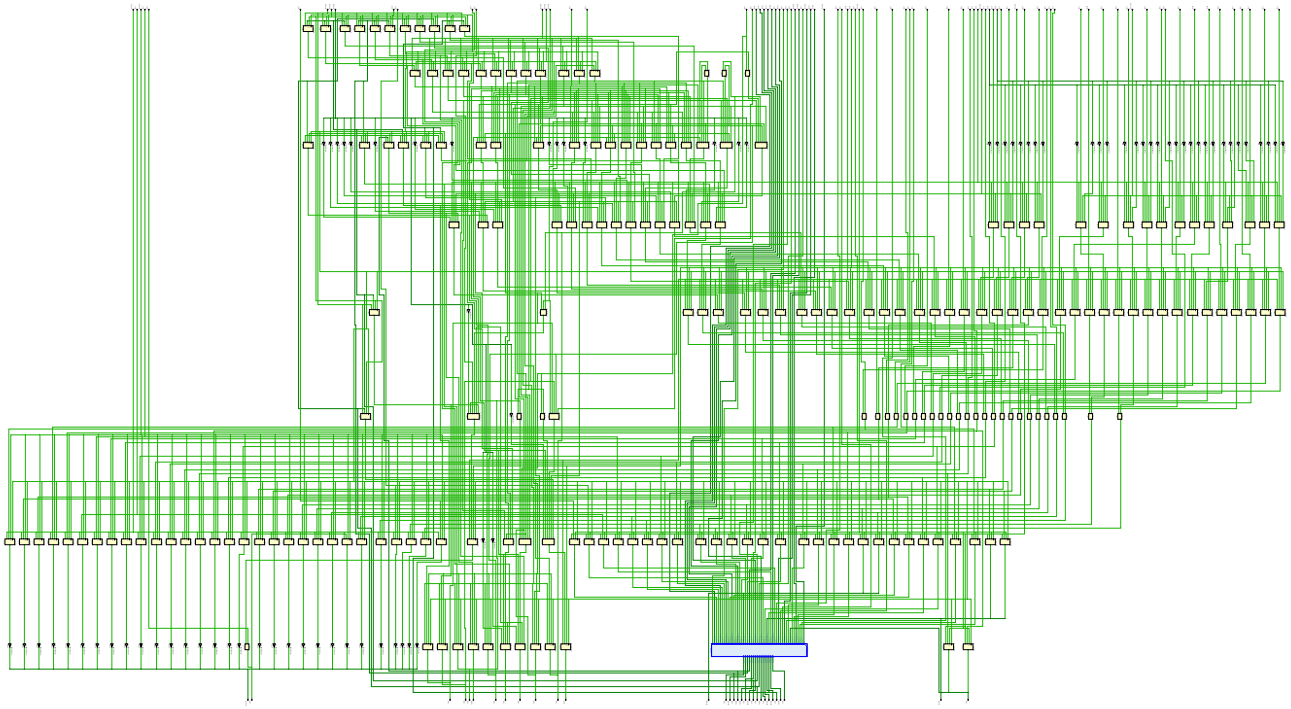

Figure 2.12: Page 9/11 of synthesized schematic of the AES Encryptor

2.5.2 Test-bench

2.6 7-segment Display (`display.vhd`)

This additional module handles the generation of the "AES" text signal for the display.

2.6.1 Block Diagram

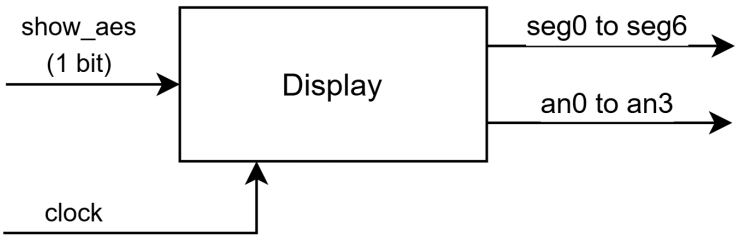

Figure 2.13: Display block diagram

The Display block has one boolean input and a clock input. It has two series of outputs, one for the display anodes and one for the display segments. The clock input will be used to make a subclock in order to have the multiplexing needed to show different characters on each digit of the display.

2.6.2 Generated Schematic

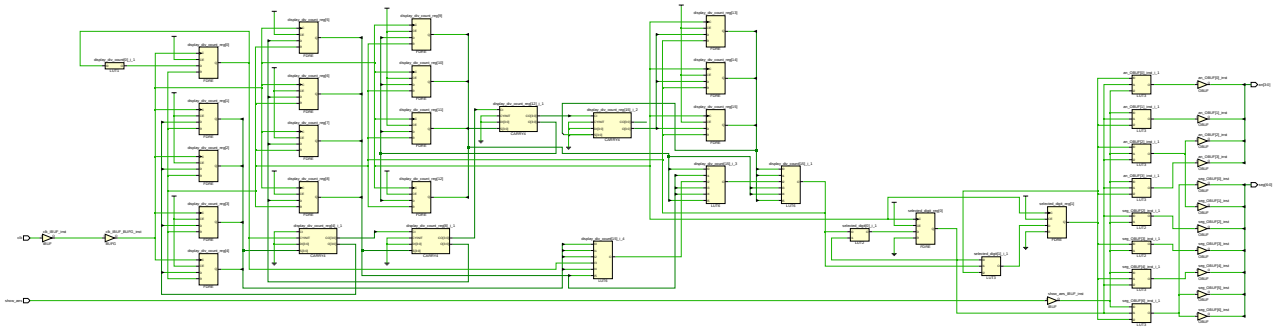

Figure 2.14: Synthesized schematic of Display driver

After running the synthesis of Display as top module, we get the schematic on Figure 2.14. We can clearly see both the input and the anodes and segments outputs. We also observe that the big subclock part of the schematic, and the rather simple part made to show each digit on each subclock count tick when the input is high.

2.6.3 Hardware Test

No test-bench was created for this module as it interfaces to the hardware i/o of the Basys3 board. It was directly tested on hardware (with a quick project to show "AES" when btnD was pressed, seen on Figure 2.15).

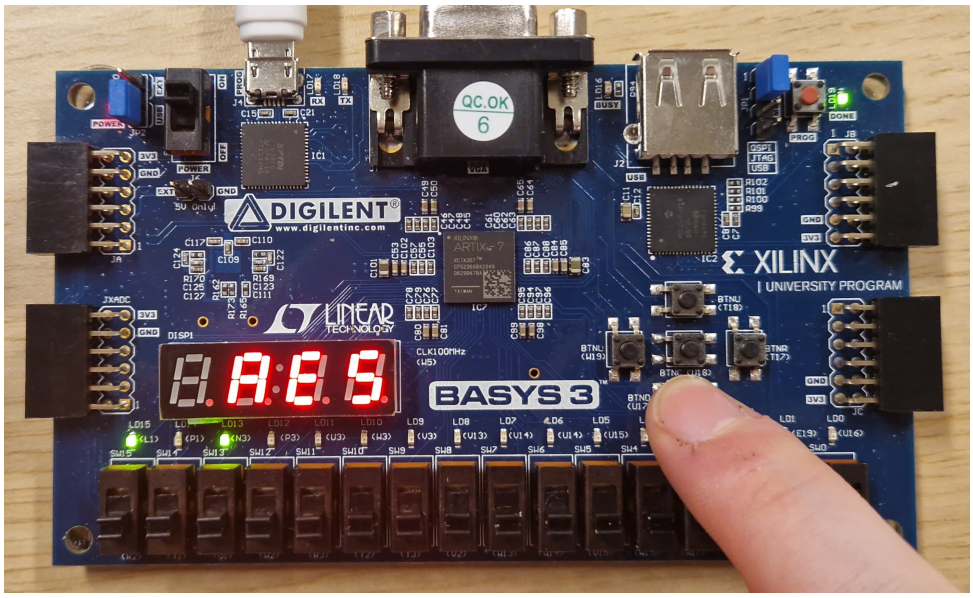

Figure 2.15: Photo of display test showing "AES" text when pressing down `btnD`

2.7 Custom Types (`matrix_type_pkg.vhd`)

This is simply a utility to convert 128-bit vectors to 4x4-byte matrices (and vice-versa) and define this byte_matrix type and a type key_array of 11 128-bit keys, used for easier development.
No test-bench needed for this simple module.

2.8 Top Module (`top_io.vhd`)

This top module handles the I/O (buttons, LEDs and 7-segment display), the main clock, and interfaces them with the AES Encryptor module. It is the module that is loaded onto the FPGA board.

2.8.1 Block Diagram

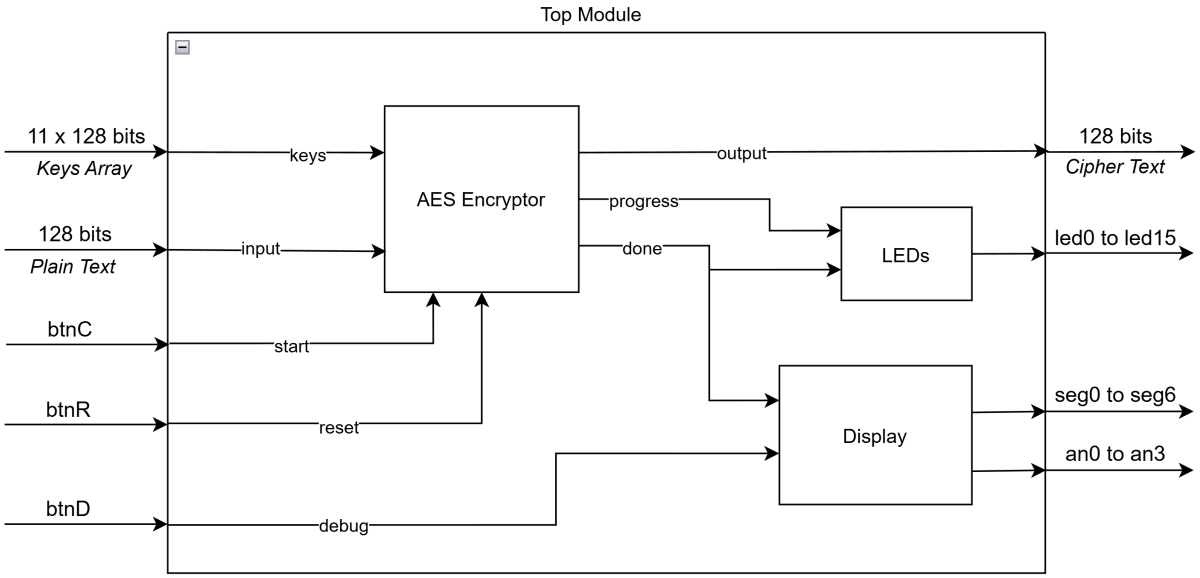

Figure 2.16: Top Module block diagram

The Top Module block has 5 inputs, one plain text vector input, one array of 11 keys, the *Center* button, the *Right* button, and the *Down* button, and 4 groups of outputs, one output vector (the cipher text), one group of outputs to every LED, one group of output of every display segment, and one group of output of every display anode. It contains the AES Encryptor module and the Display module, as well as small built-in logic regarding the LEDs.

Note: The clock was omitted in this diagram. The button inputs are being synchronized (but not debounced as it was unecessary for our design) in our implementation.

2.8.2 Generated Schematic

Synthesis

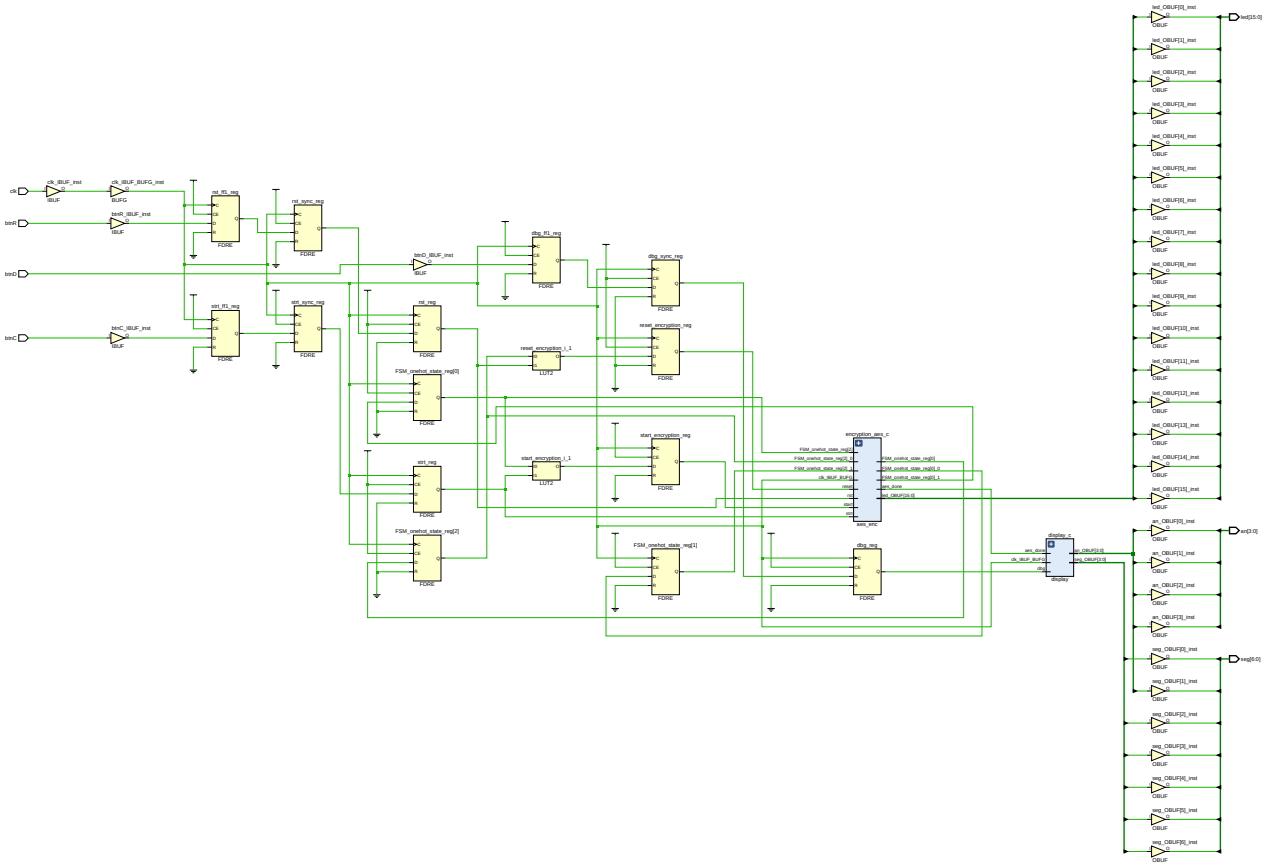

Figure 2.17: Synthesized schematic of top module

Implementation

The implementation on the FPGA can be seen in Appendix B.2 (Figure B.3, and a close-up Figure B.4).

2.8.3 Hardware Test

No test-bench was created for this module as it interfaces to the hardware I/O of the Basys3 board. It was directly tested on the hardware.

In order to test the hardware I/O, we created early on a small implementation where pressing the *Down* button would light up an LED and show "AES" on the 7-segment display.

2.9 State diagrams

Here are described the states and transitions of each Finite State Machine (FSM) used.

2.9.1 Top FSM

State machine for the FPGA's implemented top module.

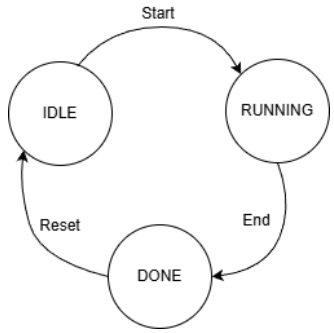

Figure 2.18: Top module state diagram, used in `top_io.vhd`

2.9.2 AES encryptor FSM

"Macro state machine" in the AES Encryptor module.

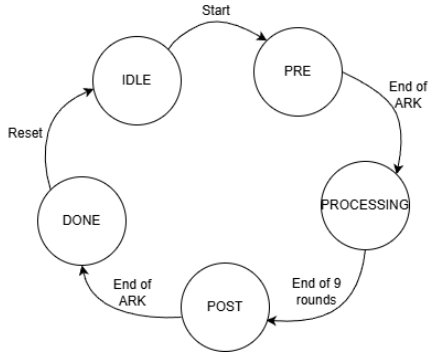

Figure 2.19: AES encryptor state diagram, used in `aes_enc.vhd`

2.9.3 AES steps FSM

"Micro state machine" in the AES Encryptor module.

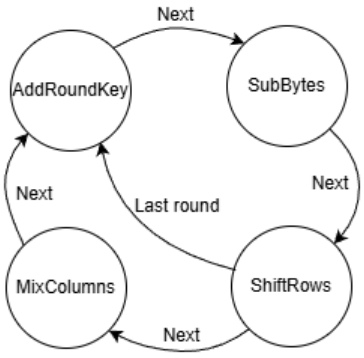

Figure 2.20: AES steps state diagram, used in `aes_enc.vhd`

2.10 Extras

We have implemented additional functionalities:

- Status LEDs:

  LED15 shows that the Top FSM is in IDLE or DONE state, LED13 shows that the AES Encryptor FSM is in the IDLE or DONE state, LED0 shows that the Top FSM is in the DONE state (encryption finished, "AES" shown on screen), and LED7 lights up (when in Top FSM DONE state) if the encryption result matches the expected (pre-computed) result.

- Progress LEDs:

  LED12 to LED2 shows the progress of each encryption round (1+9+3 rounds), while running.

- Debug Button:

  The *Down* button makes the 7-segment display show the "AES" characters when pressed, regardless of state.

These additional features allowed us to make sure our implementation works as intended.

3 Simulation & Results

3.1 Simulation

3.1.1 AddRoundKey step

On Figure 3.1, we can see that the test bench of the AddRoundKey module shows that our implementation of this step works as expected.

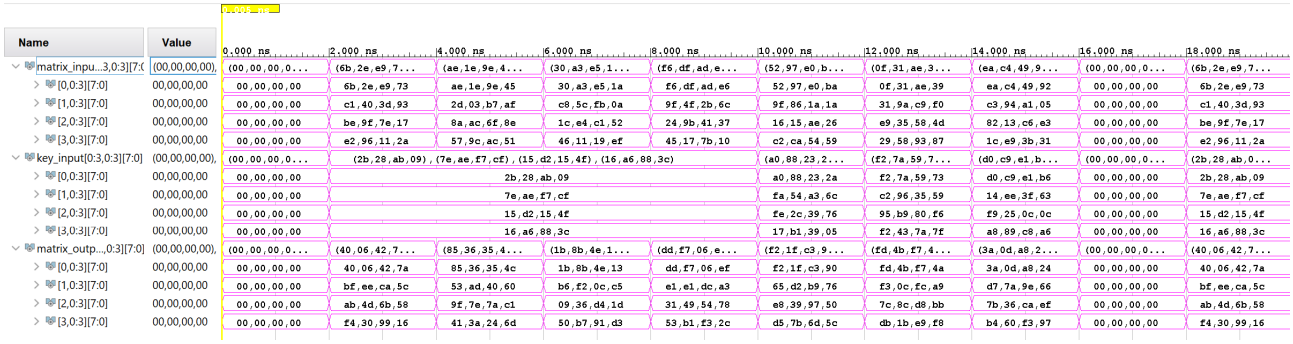

Figure 3.1: Waveform result for AddRoundKey test bench

3.1.2 SubBytes step

On Figure 3.2, we can see that the test bench of the SubBytes module shows that our implementation of this step works as expected.

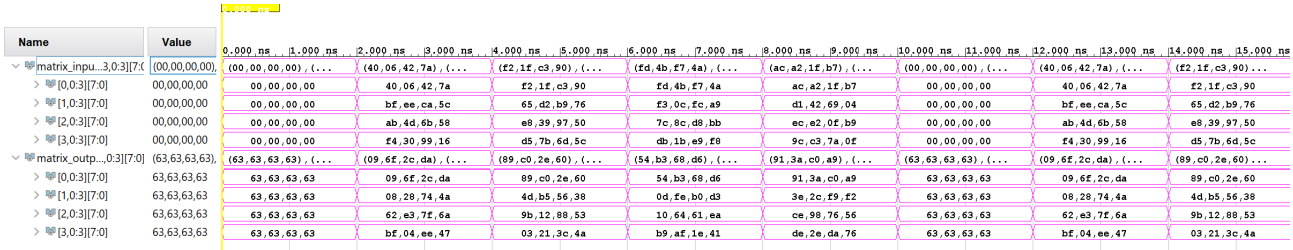

Figure 3.2: Waveform result for SubBytes test bench

3.1.3 ShiftRows step

On Figure 3.3, we can see that the test bench of the ShiftRows module shows that our implementation of this step works as expected.

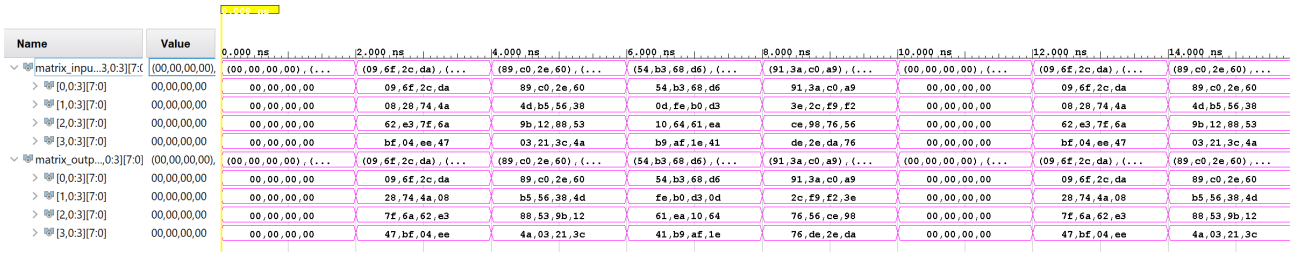

Figure 3.3: Waveform result for ShiftRows test bench

3.1.4 MixColumns step

On Figure 3.4, we can see that the test bench of the MixColumns module shows that our implementation of this step works as expected.

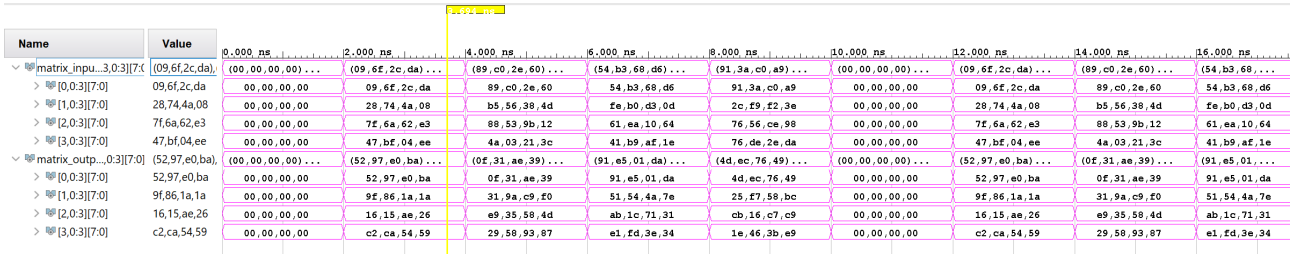

Figure 3.4: Waveform result for MixColumns test bench

3.1.5 AES Encryption

Let's analyse the waveform we get from our test bench (Figure 3.5), the encryptor gets reset between each test.

Figure 3.5: Waveform result for AES Encryption test bench

The *result* signal is meant to be kept at the value of the output (to see better on the waveforms). We see that each test's result matches the expected result, and that the *done* signal indeed goes high when the encryption finishes (that's where we get the output).

Let's take a closer look at the first (non-zero) input to the encryptor (Figure 3.6).

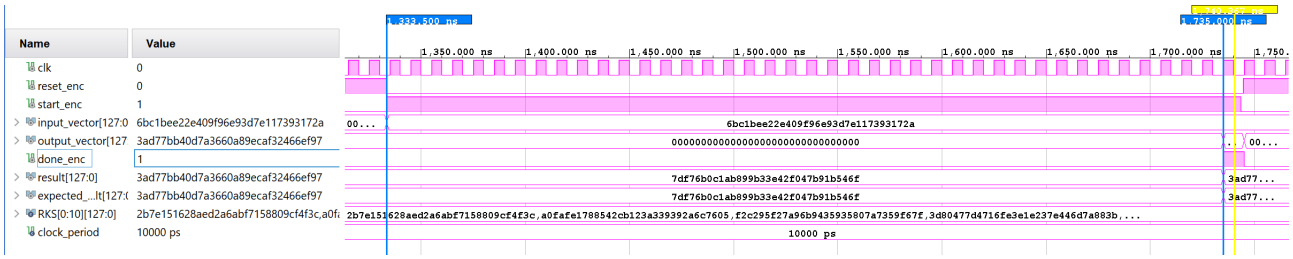

Figure 3.6: Close-up of first non-zero input encryptor test bench waveform

On this Figure 3.6, we can see that the test bench of the AES Encryptor module shows that our design completely works as expected.

It indeed takes 40 clock cycles (400ns) to encrypt the input, as expected ($1 \text{ step} + 4 \text{ steps} \times 9 \text{ rounds} + 3 \text{ steps} = 40 \text{ steps} = 40$ clock cycles).

Now that we know our entire AES encryptor works, we are ready to make a complete test on hardware.

3.2 Results on hardware

3.2.1 Basys3 FPGA board execution

The pre-programmed test plain text input gets successfully encrypted in the desired 40 clock cycles, which at the Basys3's 100MHz clock, makes it not possible to see the encryption progress with a naked eye. Indeed the board encrypts so fast that when the center button is pressed (meaning start encryption), we instantly see "AES" shown on the 7-segment display and LED0 turned on (meaning *done*). LED7 being lit means that the encypted text output matches the expected cipher text result.

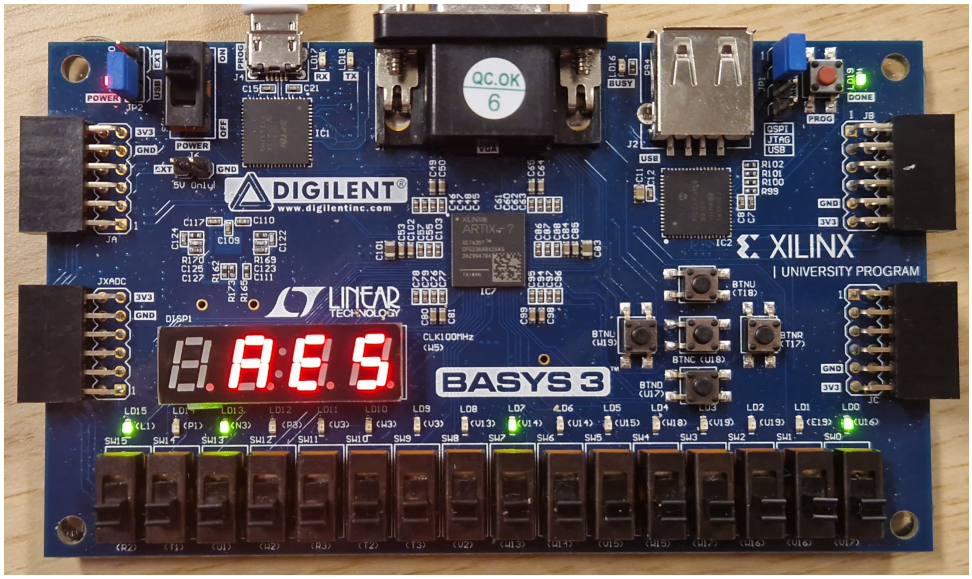

Figure 3.7: Photo of FPGA board done encrypting

Note: A clever way to see every round being executed is by pressing simultaneously both the center and right buttons. By doing that, what we would see is that the LEDs 12-to-2 turn on (each LED represents a round of encryption) and also we would see the first (and the last) LED has slightly less brightness because of the number of steps in this round is less, meaning the LED is on during less clock cycles (see photo in Appendix A, Figure A.1).

4 Conclusion

In this project, an AES-128 encryption FPGA system was designed by separating the algorithm into different modules. Each module was implemented in VHDL and developed using AMD's Vivado software. This modular approach made the design easier to understand and to test during the development process.

For every module, a specific test bench was written in order to verify its behavior. Several simulations were executed using these test benches, and the obtained waveforms showed that the modules are working correctly.

After that, all modules were connected together into a top module to build the complete AES encryption system interfacing with hardware I/O. The final design was synthesized and implemented on the Digilent Basys3 FPGA board. The system was then executed on the hardware and the expected behavior was observed during the tests.

Based on the simulation and FPGA results, it can be concluded that the project was successfully completed and that a working AES-128 encryption implementation was obtained.

Appendix

A Photo of FPGA board

A.1 Trick to see all rounds running

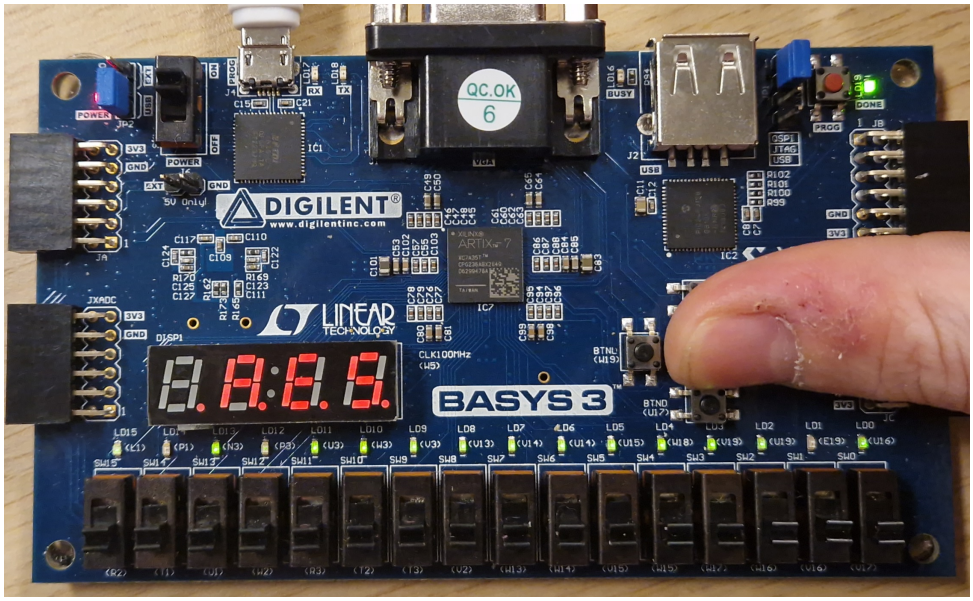

Figure A.1: Photo of FPGA board when `btnC` and `btnR` are pressed simultaneously

B Generated Schematics

B.1 Synthesis Schematics

Figure B.1 shows the schematic output after running the synthesis of the `aes_enc.vhd` as top module. We can see that it is separated into 11 pages of schematics. The page 5 shown here features 322 cells and 1868 nets.

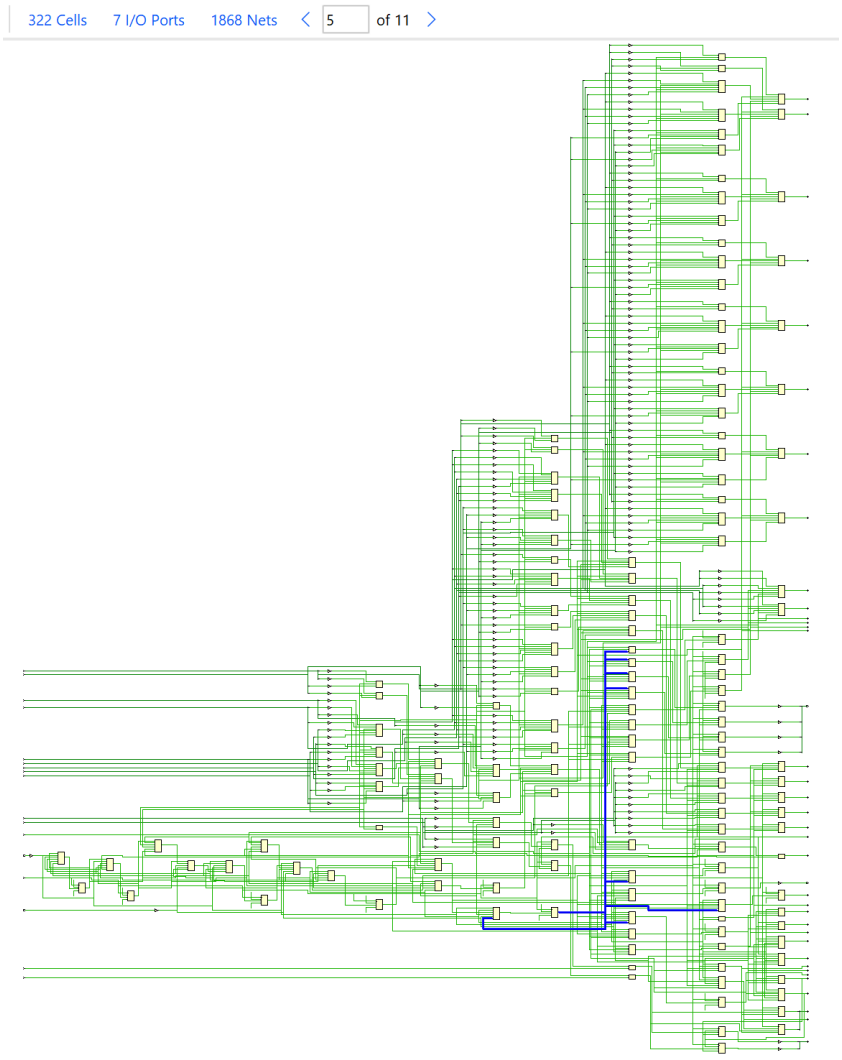

Figure B.1: Part of `aes_enc` module synthesis schematic

Figure B.2 shows the schematic output of the after running the synthesis of the `top_io.vhd` as top module. We can see that it included all submodules (the AES steps) into this one monolithic schematic (and a single submodule for the `sub_bytes` step). It features an enormous 1959 cells and 2127 nets.

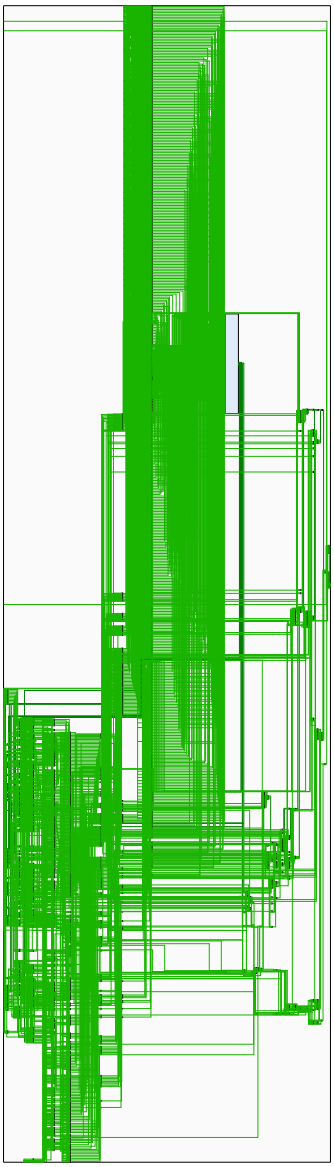

Figure B.2: `aes_enc` module synthesis monolithic schematic

B.2 Implementation on FPGA

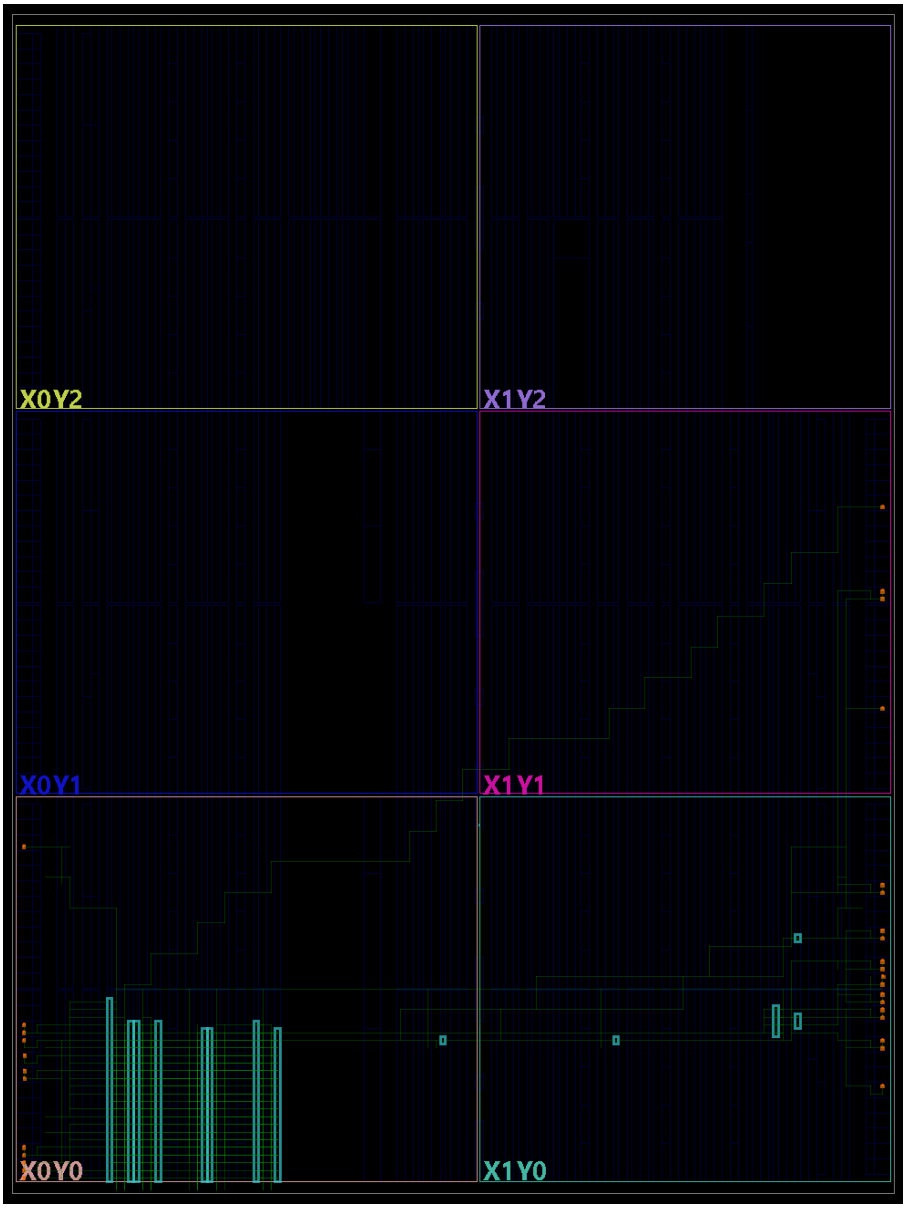

Figure B.3: Whole design implementation on the FPGA

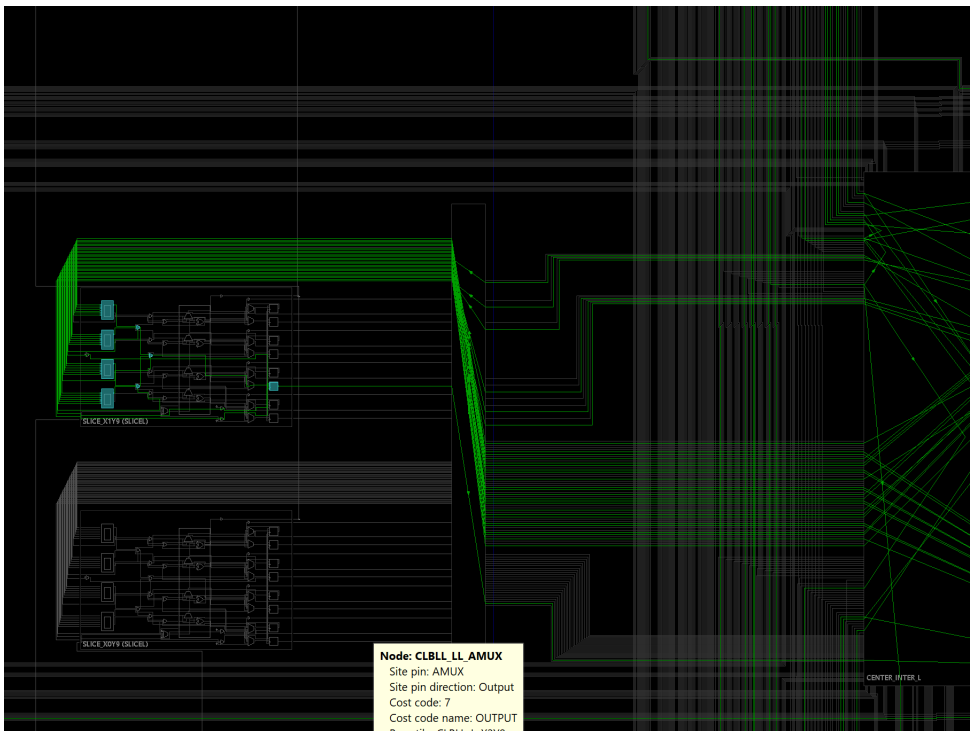

Figure B.4: Close-up of Figure B.3

C

Project Files

All of the project files can be found in the project's Github repository:

`https://github.com/LucasPlacentino/AES128-FPGA`

References

[NIS12] NIST Computer Security Division (2012). *Block Cipher Modes of Operation - Electronic Codebook (ECB)*. Accessed 14 September 2025. URL: https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/examples/AES_Core128.pdf.