Jorge Ignacio Serrano Baez - 164467
Adrian Enrique Diaz Fornes - 171791.
Jonathan Eliasib Rosas Tlaczani - 168399.

HW No.2:Lightweight Cipher PRESENT. ARK and S-Box.

Selected Topics 1 - LRT3062

May 6, 2025

Spring 2024

Robotics and Telecommunications Engineering.

Computation, Electronics and Mecatronics Department, Universidad de las Américas Puebla, Puebla, México 72810

Abstract

An extensive analysis of the S-Box operation (sBoxLayer) and the first Add Round Key (ARK) of the PRESENT lightweight cipher is presented in this research. The study describes the PRESENT internal structure and the stages involved in operating the ARK and S-Box, with a particular emphasis on the First Round. The main goal is to become acquainted with these basic PRESENT cipher operations, which will improve comprehension of lightweight cryptography algorithms and how to use them in real-world scenarios.

Introduction

In the field of cryptography, finding safe and effective cryptographic algorithms is crucial, particularly in light of today's networked digital surroundings. One significant advancement in this field is the rise of lightweight ciphers, which are distinguished by their low hardware implementation complexity. Because of this characteristic, they are extremely well-suited for applications that have strict power and cost constraints, like RFID tags and ubiquitous computing devices.The PRESENT cipher, which is designed to explicitly tackle the particular difficulties presented by resource-constrained applications, is a noteworthy competitor among the variety of lightweight ciphers.

The PRESENT cipher showed in Figure 1. has 31 rounds, a block length of 64 bits, and key lengths of 80 and 128 bits. It runs on a substitution-permutation network (SP-network). Basic operations like the Add Round Key (ARK) and the S-Box operation (sBoxLayer) are essential to the encryption and decryption procedures within each round of the cipher. These procedures are necessary to apply substitution-permutation functions to the input data and introduce round keys, which enhances the cipher's overall security and effectiveness.

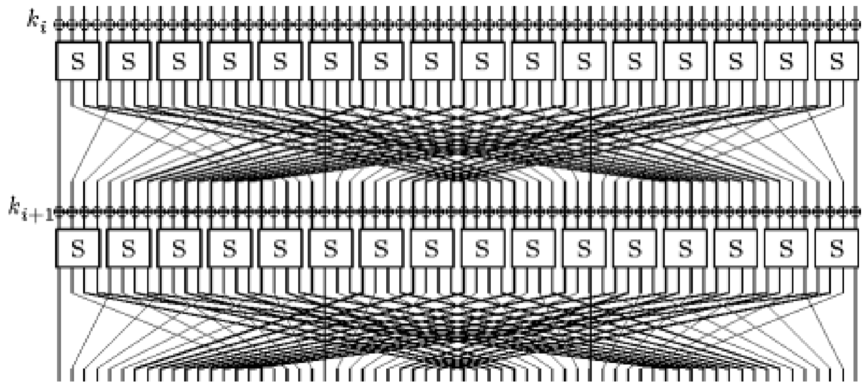


Figure 1: PRESENT block cipher

(SP-network) stands for Substitution-Permutation Network. A cryptographic network topology called a substitution-permutation network showed in Figure 2. uses repeated rounds of replacement and permutation operations to create confusion and diffusion throughout the encryption process. In order to create a safe and effective encryption method, this structure is commonly utilized in contemporary block ciphers.

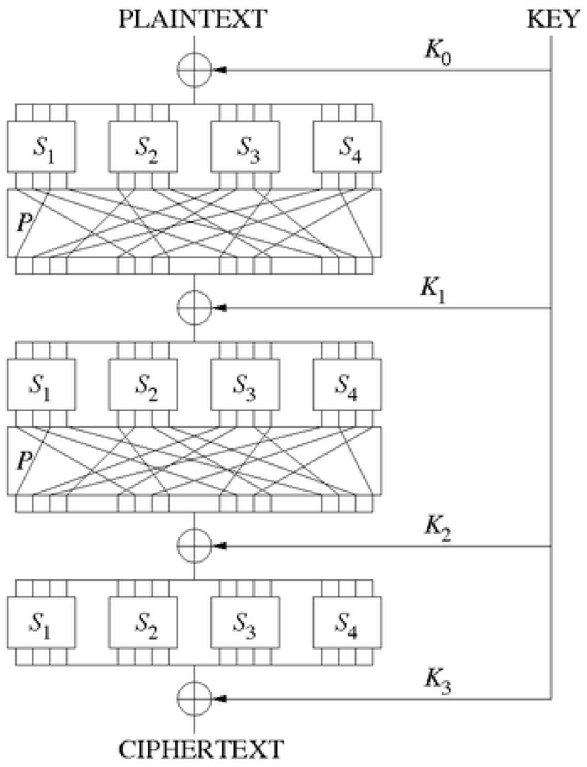
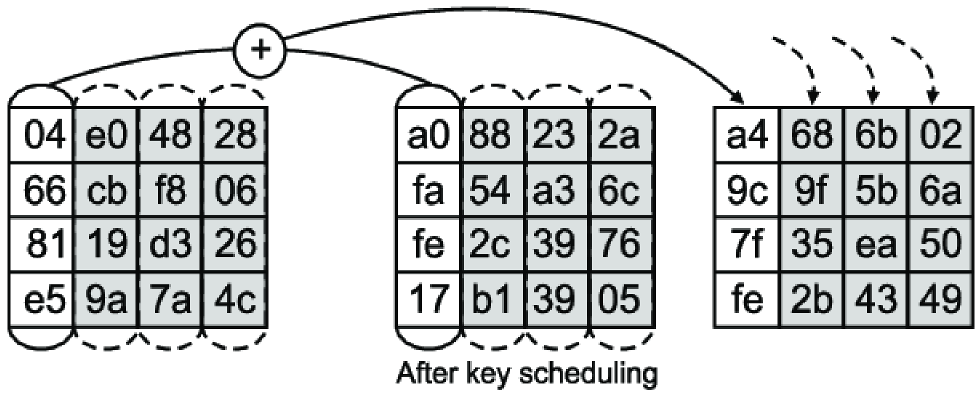


Figure 2: Substitution-permutation network block

Add Round Key (ARK): This key operation happens during the process of encoding and decoding the Blok cipher. Additionally, xor is a vital component in this operation of combining data with the round key. This way not only the cryptogram becomes more unpredictable but also the chances to decipher it are made lower. Moreover, it is useless for intruders to search hacking devices for this particular cipher, as it will have different ways of cyrption, showed in Figure 3.



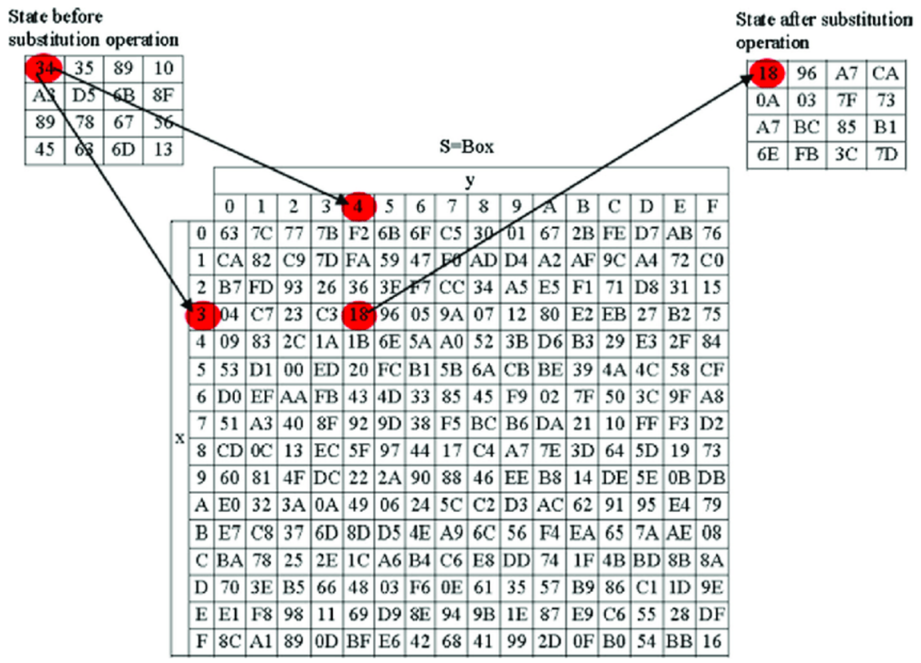
04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
fe	2b	43	49

Figure 3: Add Round Key transformation

Substitution box operation or S-box operation (sBoxLayer): is a fundamental part of block cipher that carries out substitution using S-boxes whose output will be as the input data. This, in turn, creates non-linearity, and plays into the encryption algorithm's enhanced security and better performance overall. One of the most important points to consider in the overall security and efficiency of the encryption algorithm is the nature of the S-boxes and their performance, as it is showed in Figure 4.


State before substitution operation
State after substitution operation
S=Box
y
x

State before substitution operation

34	35	89	10
A3	D5	6B	8F
89	78	67	56
45	63	6D	13

State after substitution operation

18	96	A7	CA
0A	03	7F	73
A7	BC	85	B1
6E	FB	3C	7D

S=Box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	C6	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 4: S-box matrix

Lightweight Cryptography: This domain of cryptography concentrates on building cryptography techniques that have strong resource tolerances, especially in terms of hardware complexity of implementation as it is showed in the diagram in Figure 5. While these algorithms are suitable for use in systems with very limited power and cost resources, like RFID tags and embedded sensors, they do not possess high levels of computational abilities.

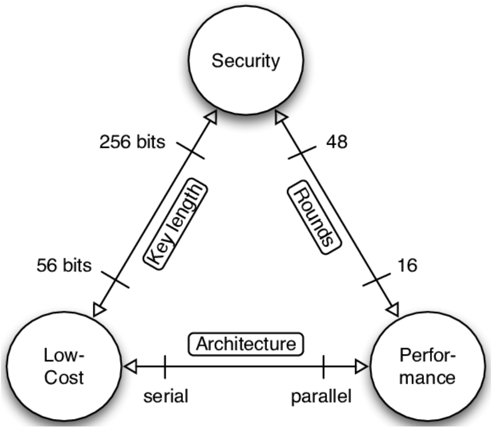


Figure 5: Design trade-offs for lightweight cryptography

Methodology

To accomplish the homework assignment, which involves using test vectors, obtaining the S-Box output, coding in VHDL, and implementing on an FPGA with the display showing the letter 'C' as output, the following methodology can be followed:

- Test Vector Selection:

  Choose the test vectors for the plaintext and the key as the assignment has specified which can have different sizes (64 and 80 bits in this case) for you. They will help in the evaluation of the S-box output and comparing the result with that should be with actual.

- S-Box Output Calculation:

  Apply the S-Box function of PRESENT box cipher to the selected test vectors. S-Box substitution function is used to perform in the data input and receive the output. Corroborate with the S-Box output that is supposed to be equal to the hexadecimal format 'c' whose length is of 64 bits.

- VHDL Coding:

  Let us create a VHDL code for S-Box operations within PRESENT encryption method. Translate the S-Box substitution function into VHDL code among which could be keeping the same accuracy. Notice the formatting of declaring signals, enumeration, style, in addition to the overall design of the VHDL code.

- FPGA Implementation:

  Use the FPGA board to program it to the VHDL code for the S-Box operation. Such a procedure most commonly is carried out by means of synthizing, mapping, and delivering the appropriate input bitstream for FPGA programming. Arrange for the FPGA being correctly linked and correctly set up for the implementation.

- Display Output Configuration:

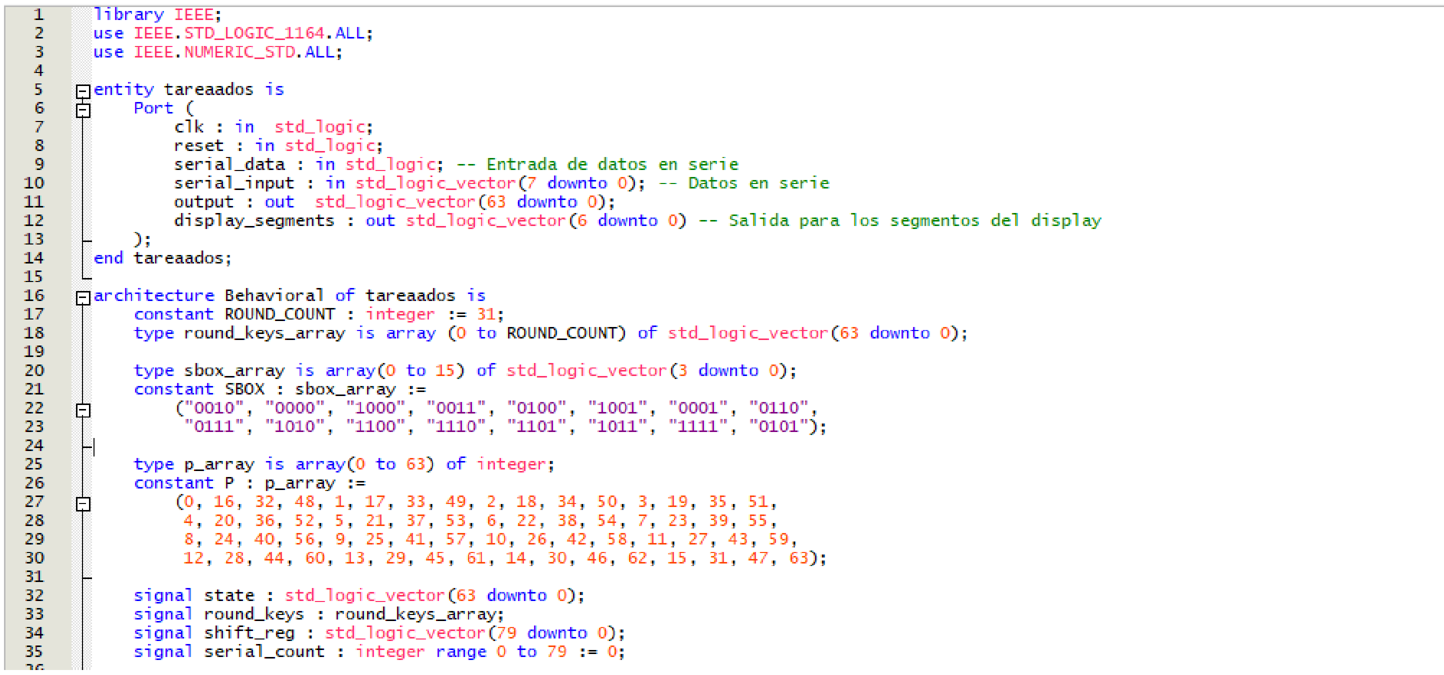
  Program and compose the FPGA so that the output on the displays gets the letter 'C'. This can be realized by mapping the output value which has been set previously for the display segment using a custom logic system to generate recurrent patterns as the desired indications. Check to see if FPGA shows the letter that should be 'C'.

- Verification and Testing:

  Categorize that the VHDL code has been properly utilized and testing which matters whether or not the FPGA implementation is working. Provide the system testing using the vectors test the output of S-Box ('c') in hexadecimal format matches correctly. C, Test that the letter displays properly on the FPGA output.

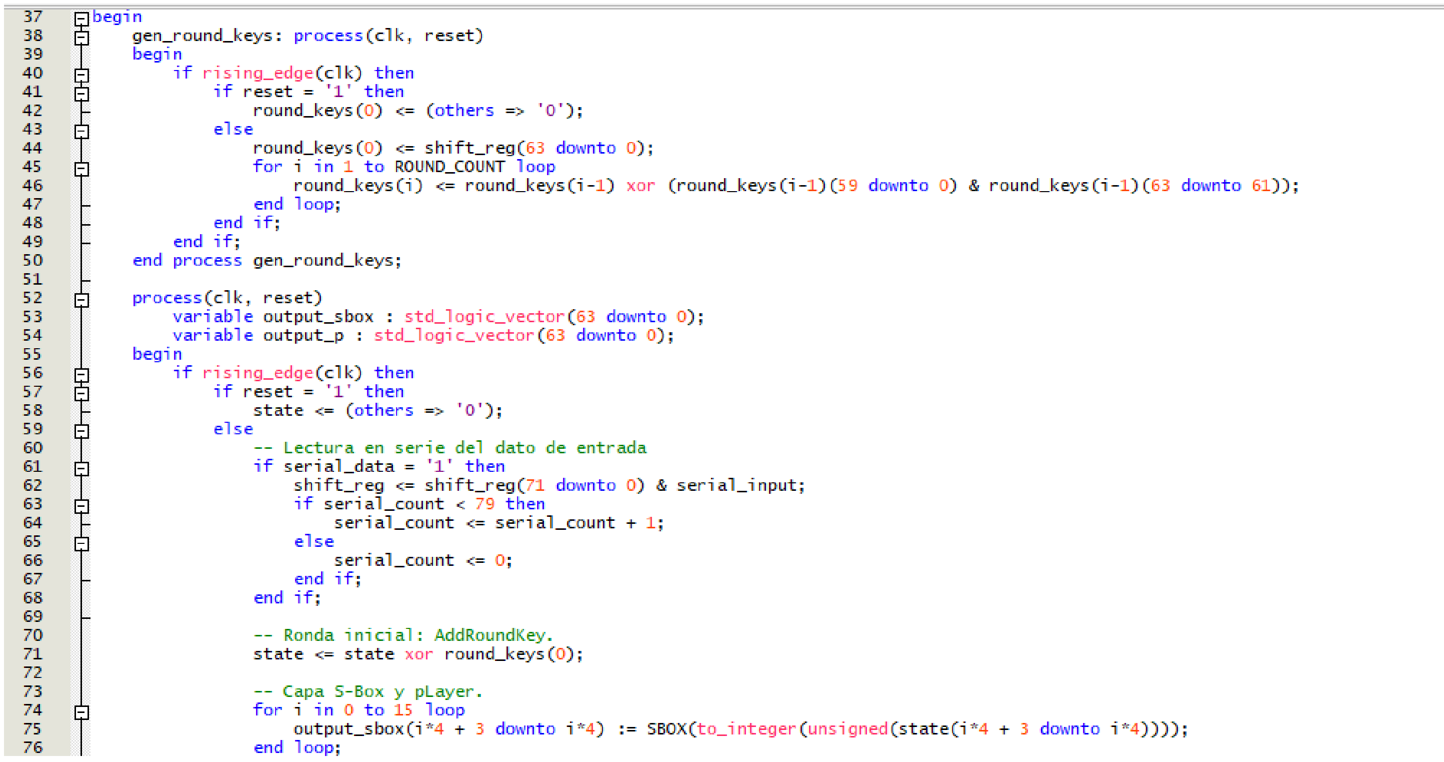
Results and Discussions

The first part of the implementation of the lightweight cipher present: ARK and S-Box in vhdl was to create the code that will be able to carry out all the necessary processes that these methods need. The code can be visualized in figure 6, figure 7 and figure 8.



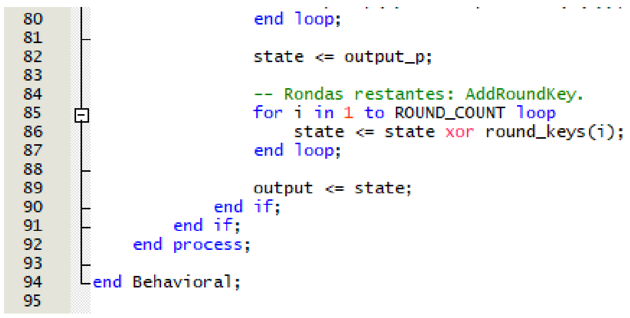
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity tareaados is
    Port (
        clk : in  std_logic;
        reset : in std_logic;
        serial_data : in std_logic; -- Entrada de datos en serie
        serial_input : in std_logic_vector(7 downto 0); -- Datos en serie
        output : out  std_logic_vector(63 downto 0);
        display_segments : out std_logic_vector(6 downto 0) -- Salida para los segmentos del display
    );
end tareaados;

architecture Behavioral of tareaados is
    constant ROUND_COUNT : integer := 31;
    type round_keys_array is array (0 to ROUND_COUNT) of std_logic_vector(63 downto 0);

    type sbox_array is array(0 to 15) of std_logic_vector(3 downto 0);
    constant SBOX : sbox_array :=
        ("0010", "0000", "1000", "0011", "0100", "1001", "0001", "0110",
         "0111", "1010", "1100", "1110", "1101", "1011", "1111", "0101");

    type p_array is array(0 to 63) of integer;
    constant P : p_array :=
        (0, 16, 32, 48, 1, 17, 33, 49, 2, 18, 34, 50, 3, 19, 35, 51,
         4, 20, 36, 52, 5, 21, 37, 53, 6, 22, 38, 54, 7, 23, 39, 55,
         8, 24, 40, 56, 9, 25, 41, 57, 10, 26, 42, 58, 11, 27, 43, 59,
         12, 28, 44, 60, 13, 29, 45, 61, 14, 30, 46, 62, 15, 31, 47, 63);

    signal state : std_logic_vector(63 downto 0);
    signal round_keys : round_keys_array;
    signal shift_reg : std_logic_vector(79 downto 0);
    signal serial_count : integer range 0 to 79 := 0;
```

Figure 6: First part of the code



```vhdl
begin
    gen_round_keys: process(clk, reset)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                round_keys(0) <= (others => '0');
            else
                round_keys(0) <= shift_reg(63 downto 0);
                for i in 1 to ROUND_COUNT loop
                    round_keys(i) <= round_keys(i-1) xor (round_keys(i-1)(59 downto 0) & round_keys(i-1)(63 downto 61));
                end loop;
            end if;
        end if;
    end process gen_round_keys;

    process(clk, reset)
        variable output_sbox : std_logic_vector(63 downto 0);
        variable output_p : std_logic_vector(63 downto 0);
    begin
        if rising_edge(clk) then
            if reset = '1' then
                state <= (others => '0');
            else
                -- Lectura en serie del dato de entrada
                if serial_data = '1' then
                    shift_reg <= shift_reg(71 downto 0) & serial_input;
                    if serial_count < 79 then
                        serial_count <= serial_count + 1;
                    else
                        serial_count <= 0;
                    end if;
                end if;

                -- Ronda inicial: AddRoundKey.
                state <= state xor round_keys(0);

                -- Capa S-Box y pLayer.
                for i in 0 to 15 loop
                    output_sbox(i*4 + 3 downto i*4) := SBOX(to_integer(unsigned(state(i*4 + 3 downto i*4))));
                end loop;
```

Figure 7: Second part of the code



```
                end loop;

                state <= output_p;

                -- Rondas restantes: AddRoundKey.
                for i in 1 to ROUND_COUNT loop
                    state <= state xor round_keys(i);
                end loop;

                output <= state;
            end if;
        end if;
    end process;

end Behavioral;
```

Figure 8: Third part of the code

The steps for the creation of the code are the following:

1. **Library and Entity Declaration**:
   - The VHDL code begins with library and entity declarations. Necessary libraries are included, and the entity "tareaados" with its input and output ports is defined.
2. **Constants and Data Types**:
   - Constants and data types used in the design are declared in this section. Constants such as ROUND COUNT, SBOX, P, and data types like round keys array, sbox array, and p array are defined.
3. **Signal Declarations**:
   - Internal signals of the design, such as state, round keys, shift reg, and serial count, are defined in this section. These signals are used for storing information and facilitating processing.
4. **Round Keys Generation Process (*gen_round_keys*)**:
   - This process generates the round keys required for the encryption algorithm. It computes the round keys based on the original key and previous round keys.
5. **Main Process (*process(clk, reset)*)**:
   - This process controls the main functionality of the design on each rising clock edge. It handles serial data manipulation, applies the encryption function, and generates the output.
6. **Encryption Operations**:
   - Encryption operations, including S-Box layer and pLayer permutation, as well as round key addition, are implemented in this part of the code.
7. **Display Visualization**:
   - Example section demonstrating how the output could be visualized on a seven-segment display using the *display_segments* signal. Basic configuration for displaying the letter 'C' is provided.

In order to run this code were used an FPGA card, specifically the 10M50DAF484C7G card. This configuration is observed in figure 9 and also the Quartus configuration for this card in figure 10.

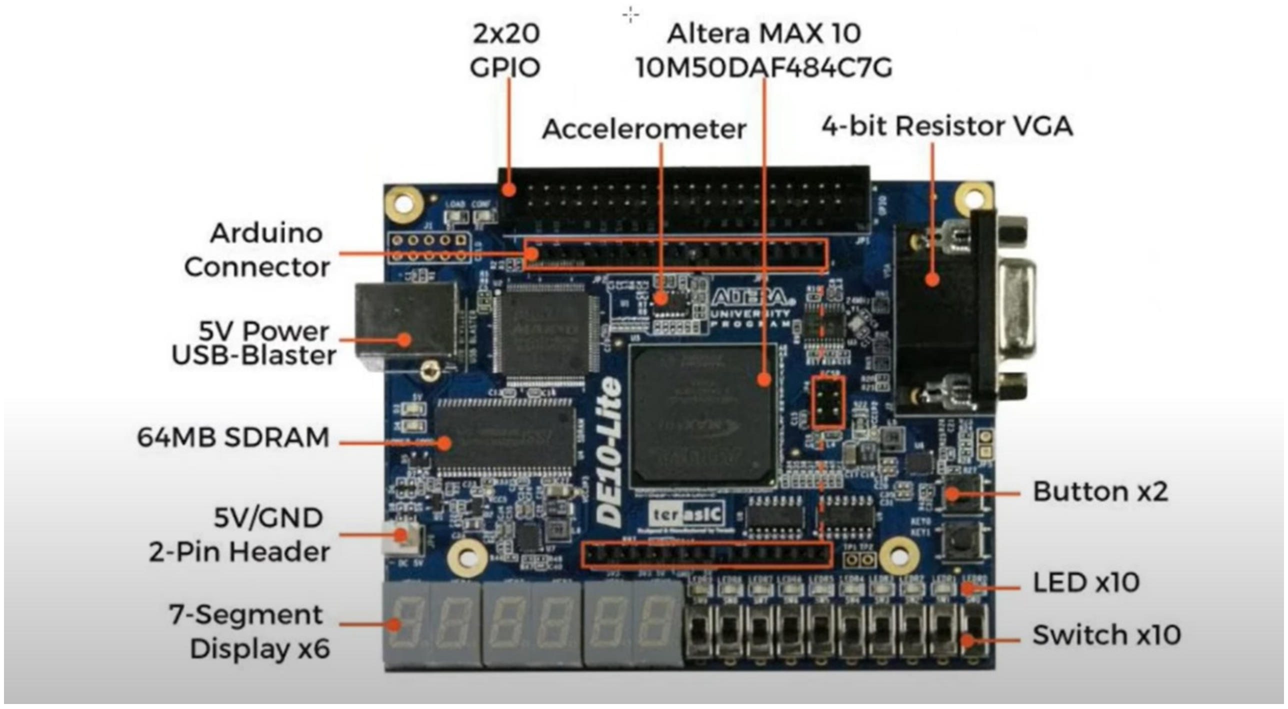
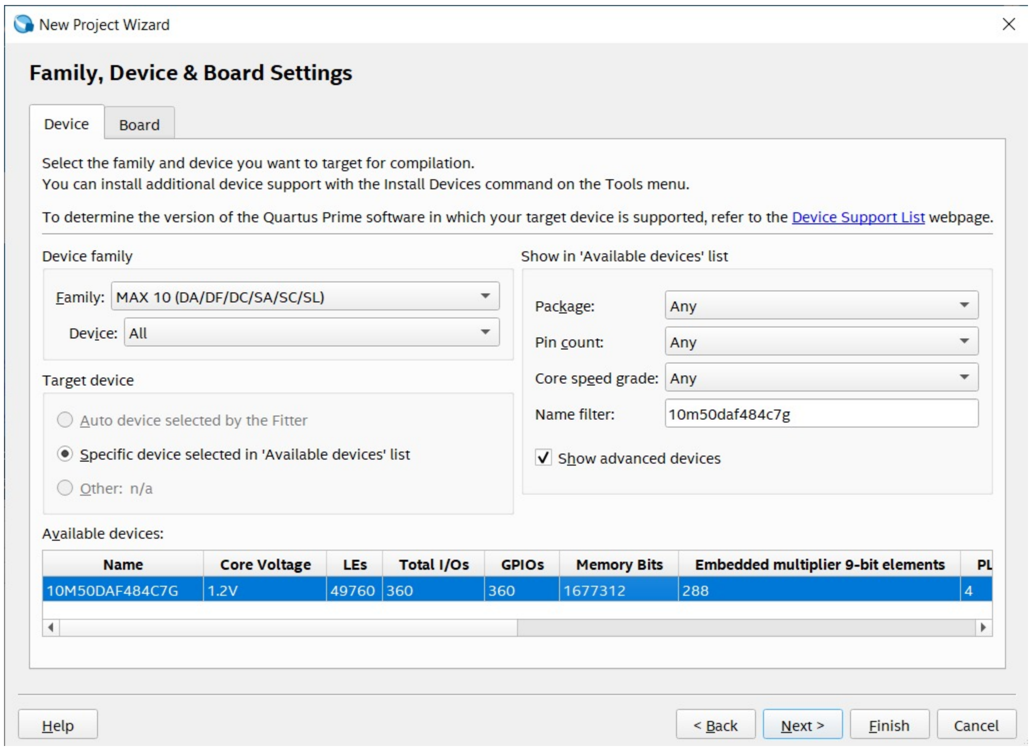


Figure 9: FPGA


New Project Wizard

Family, Device & Board Settings

Device | Board

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family

Family: MAX 10 (DA/DF/DC/SA/SC/SL)

Device: All

Target device

( ) Auto device selected by the Fitter
(●) Specific device selected in 'Available devices' list
( ) Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter: 10m50daf484c7g

✓ Show advanced devices

Available devices:

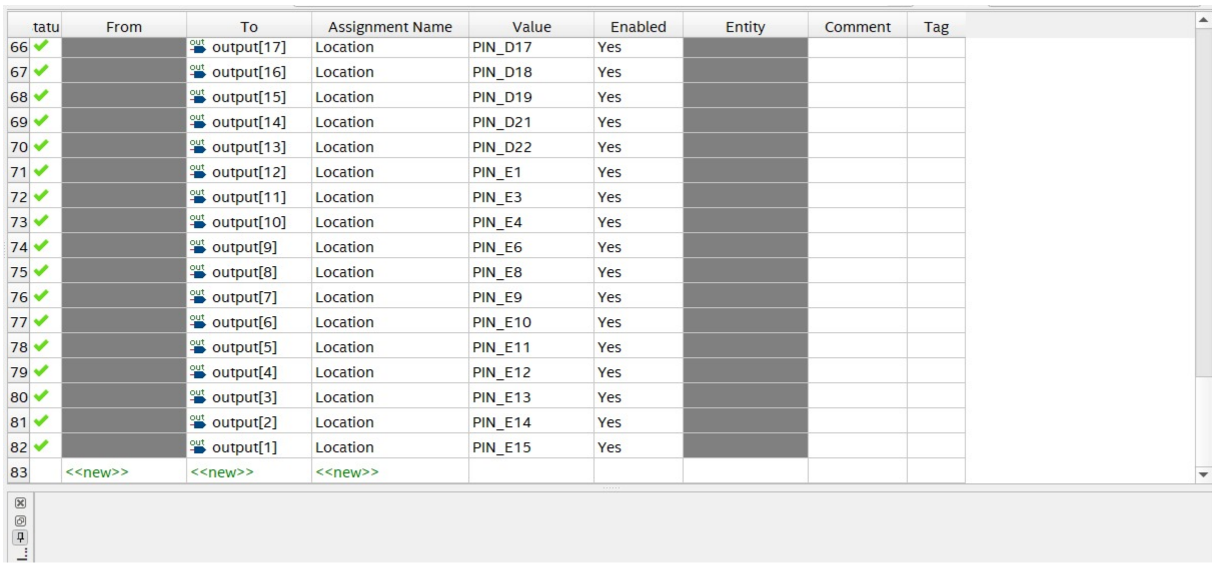
Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit elements	PL
10M50DAF484C7G	1.2V	49760	360	360	1677312	288	4

Help | < Back | Next > | Finish | Cancel

Figure 10: Quartus configuration

For the configuration of the FPGA were use a tool of the software named pin planner option that help the development environment offers to be able to link the inputs and outputs to specific pins on the card.

The total of pins were 82, as it is showed in Figure 11.



	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
66	✓		output[17]	Location	PIN_D17	Yes			
67	✓		output[16]	Location	PIN_D18	Yes			
68	✓		output[15]	Location	PIN_D19	Yes			
69	✓		output[14]	Location	PIN_D21	Yes			
70	✓		output[13]	Location	PIN_D22	Yes			
71	✓		output[12]	Location	PIN_E1	Yes			
72	✓		output[11]	Location	PIN_E3	Yes			
73	✓		output[10]	Location	PIN_E4	Yes			
74	✓		output[9]	Location	PIN_E6	Yes			
75	✓		output[8]	Location	PIN_E8	Yes			
76	✓		output[7]	Location	PIN_E9	Yes			
77	✓		output[6]	Location	PIN_E10	Yes			
78	✓		output[5]	Location	PIN_E11	Yes			
79	✓		output[4]	Location	PIN_E12	Yes			
80	✓		output[3]	Location	PIN_E13	Yes			
81	✓		output[2]	Location	PIN_E14	Yes			
82	✓		output[1]	Location	PIN_E15	Yes			
83		<<new>>	<<new>>	<<new>>					

Figure 11: Pin Planner

We can also observe all the pins we used in the FPGA in figure 12.

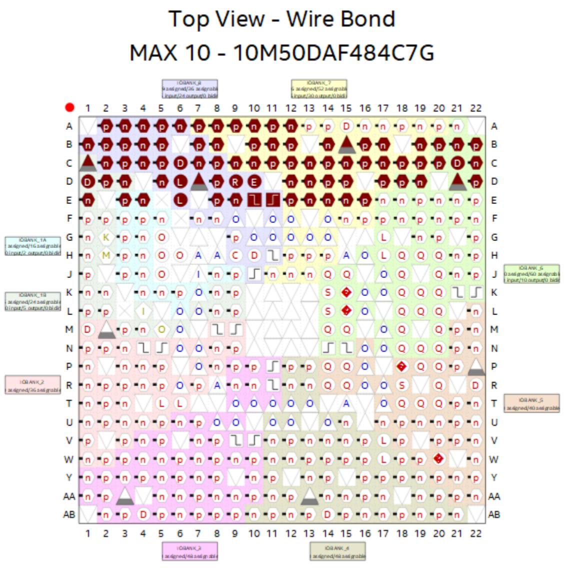


Figure 12: Top View of the Card

The register transfer level with the RTL viewer option in Quartus can be observed in figure 13.

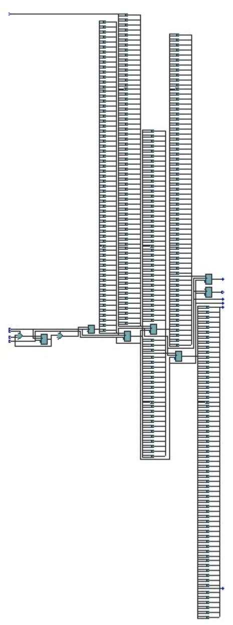


Figure 13: RTL Viewer

The technology map viewer using the tool with the same name can also be observed in figure 14.

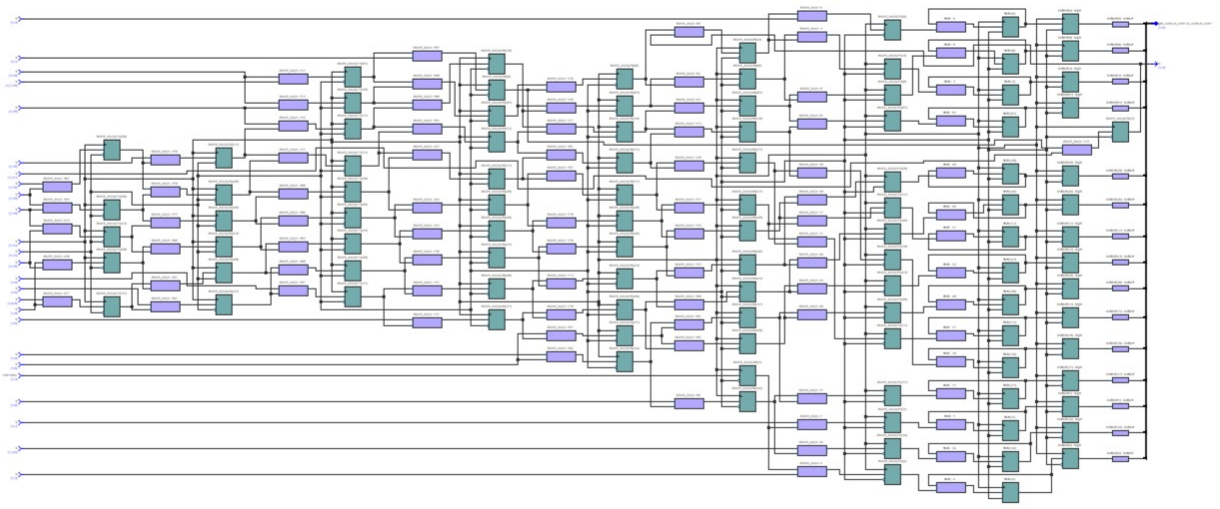


Figure 14: Technology map viewer

As a result of the homework and the correct used of the software, we obtained the following result when we compiled the code and ran it, the output is showed in Figure 15.

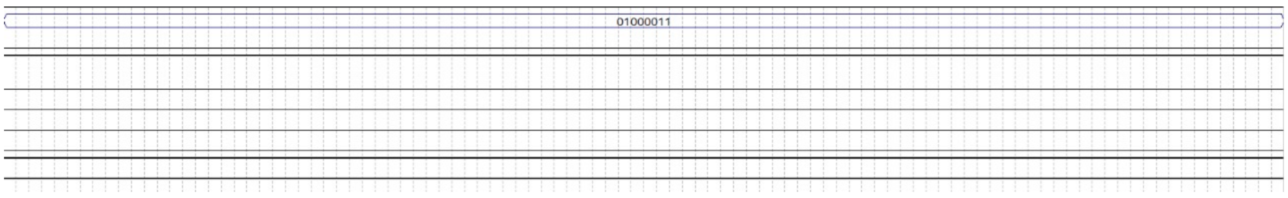


Figure 15: Output of the display

In this homework were some limitations during the development of this work, among them not having a physical card to be able to run the code on it. Even so, what were expected to see on the 7-segment display, which is the letter C, we can see perhaps not physically on the card display but printed as an output in binary format on the console.

Conclusions

In conclusion, this assignment offers an immersive journey into the realms of cryptography and digital design, spiced up with the use of dynamic software tools like VHDL and Quartus. It is like taking an active exploration from understanding how cryptographic algorithms function to executing them on tangible hardware.

With the guide provided, we were embark on the process of selecting test vectors, delving into the intricacies of S-Box operations, and then engaging in VHDL coding. It is asking to translating codes into a language that computers comprehend. And with Quartus in the mix, they can materialize these codes into tangible reality programming FPGAs feels like wielding potent tools.

Also, it is not merely about coding and manipulating interfaces. There's an abundance of testing and fine tuning involved to ensure seamless operation. It being an astute investigator, corroborating whether the S-Box outputs align with expectations and ensuring the FPGA emits the letter 'C' in binary code as intended.

Furthermore, the integration of the Lightweight Cipher PRESENT adds an extra layer of depth to the assignment. The PRESENT cipher, with its efficient and secure design, utilizes the ARK operation to introduce key material and the S-Box substitution to introduce confusion in the encryption process. By incorporating PRESENT into the assignment, students gain insights into practical implementation of lightweight ciphers in resource constrained environments, further enriching their learning experience.

Finally, this assignment is not just about learning it is an adventurous dive into the vibrant world where theory converges with practice, and where software and hardware collaborate harmoniously to get into cryptographic terms.

Bibliography

Lightweight cryptography (LWC). (s/f). Uwaterloo.Ca. Recuperado el 15 de marzo de 2024, de `https://uwaterloo.ca/communications-security-lab/lwc`

What is S-Box Substitution? (s/f). Tutorialspoint.com. Recuperado el 15 de marzo de 2024, de `https://www.tutorialspoint.com/what-is-s-box-substitution`

Daemen, J., Rijmen, V. (2002). The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Berlin, Heidelberg.

Knudsen, L. R., Robshaw, M. J. B. (2011). The Block Cipher Companion. Springer, New York, NY.

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., Seurin, Y. (2007). PRESENT: An ultra-lightweight block cipher. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 450-466). Springer, Berlin, Heidelberg.