RESEARCH AND DEVELOPMENT OF DES ENCODER AND DECODER ON FPGA DEVICES

Vuong Thuy Linh

Master, Lecturer, Department of Information Technology, Faculty of General Education, University of Labour and Social Affairs, No. 43, Tran Duy Hung Street, Trung Hoa Ward, Cau Giay District, Hanoi City, Vietnam

Le Ngoc Giang*

PhD, Head of Metrology Department, Faculty of Fundamental Technical,
AD-AF Academy of Viet Nam, Son Tay, Ha Noi, Viet Nam
* Email: lengocgianglinh@gmail.com

Abstract

In this paper, the author researches and develops a set of DES encryption and decryption experiments on FPGA devices. The goal of this research is to design and implement a set of experiments that can be easily used and demonstrated to students and researchers how DES works, as well as evaluate the properties of the test set, such as speed, safety, and energy consumption. First, the author builds and designs a DES encoder and decoder using the VHDL language, a popular hardware description language in the field of FPGA design. Next, the author uses Xilink Vivado software to simulate, synthesize, and realize the design on the FPGA platform. The experimental set includes a main circuit board and a self-designed and manufactured expansion input and output circuit assembled with the FPGA core circuit board. The test set is connected to the computer using the UART asynchronous serial standard through an easy-to-use interface. Based on the DES encoder and decoder loaded into the FPGA, users can perform three experiments on data transmission techniques: UART asynchronous serial data transmission, data encryption with DES, and data decryption with DES. Research results show that the test set operates stably and accurately, has high encoding and decoding speed, relative safety, and low energy consumption. This research not only contributes to improving knowledge and skills in DES encryption and decryption but also opens up new perspectives on applying FPGA technology in security and communication applications.

***Keywords*: DES encoder; DES decoder; FPGA, experiments.**

1. Introduction

Encryption and decryption are two important techniques in the field of information security, as they help protect data from unauthorized access or alteration. Data Encryption Standard (DES) is a data encryption standard developed by IBM in the 1970s and adopted as a US national standard in 1977 [1]. DES uses a 56-bit secret key to encrypt and decrypt 64-bit data blocks, by applying bit transformations, bit permutations, and XOR operations. DES has been widely used in information security applications [2].

Data security is a difficult problem in today's age of information and data transmission. This is especially important in fields related to communications and security, where data encryption solutions are considered essential. In the digital future, FPGA (Field-Programmable Gate Array) devices will be an effective tool for implementing security solutions. In this challenging era, research on data security becomes even more urgent. New technologies, such as FPGAs, open new opportunities for building flexible and effective security solutions. In particular, the development of DES encoders and decoders on FPGA devices not only meets the high demand for information security but also partly solves the problem of applying strong encryption solutions to numerical applications [3].

This paper researches and develops a DES encoder and decoder on FPGA devices. DES is a traditional encryption algorithm, but it still has a role to play in the field of data security. The combination of DES and FPGA not only allows for fast encryption and decryption, but also facilitates integration into digital communication systems [4].

The goal of this research is not only to deeply study the theory of DES but also to develop a set of experiments

on encryption and decryption on FPGAs. This set of experiments not only helps to understand the operation of DES but is also practical and applicable for students and researchers in mastering the field of data security.

This paper not only focuses on the research aspect but also aims at practical application. Implementing DES on FPGAs offers flexibility and efficiency, and the paper emphasizes the need for its integration into digital communication systems. The paper is organized with a clear structure, including an introduction, design and implementation, experimental results, and conclusions.

2. Design of the encoder and decoder using the VHDL language

2.1. *Build a block diagram of the experimental equipment*

The DES encryption and decryption experimental equipment is shown in Figure 1, including three main parts: the communication part, the buffer part, and the core part.

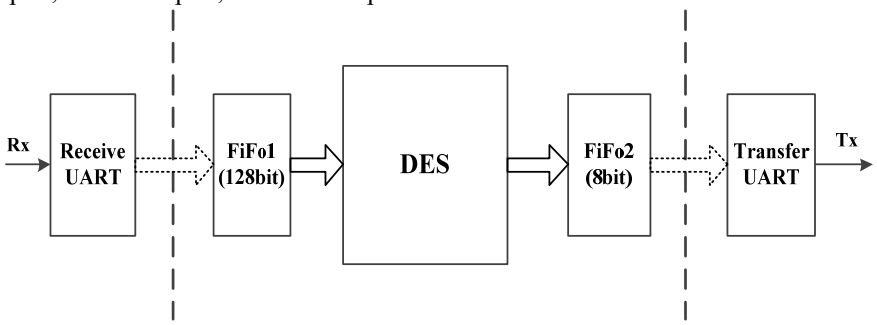

Fig.1. Block diagram of the experimental equipment

2.1.1. Communication part

The communication part has two parts: the input communication part (Receive_UART) and the output communication part (Transfer UART).

The communication part at the input of the experimental device includes a block that implements UART asynchronous serial communication. Serial data is transmitted along a single wire and encoded according to the voltage level on the wire. Data entered into the encryption block includes input data along with the encryption key divided into 8-bit data packets, then framed and put into the receiver_UART block in the form of an Rx line code. The output of the Receiver_UART block will separate the data from the frame.

The output communication part is the block that performs the inverse transformation compared to the input block, that is, generates a line code from the input bit string. The frame structure is the same as that of the Receive_UART section. The input data of this block is 8 bits of data read out from FiFo2 and then framed by inserting a Start bit (value 0) at the beginning and a Stop bit (value 1). The output of this block is done by shifting each bit to the right.

The device is connected and exchanges data with the computer using the UART (Universal Asynchronous Receiver Transmitter) data transmission interface called serial asynchronous data transmission and reception. The method of transmitting and receiving data according to the UART standard can be done in full-duplex or half-duplex mode. Unlike synchronous serial communication, asynchronous serial communication only requires one transmission line for the whole process; the "data frame" has been standardized by the devices, so there is no need for a clock line to signal before the data is transmitted. Therefore, the efficiency of asynchronous information transmission is higher than that of synchronous communication.

The UART transmitter block is responsible for transmitting each bit in the data byte sequentially. The receiving UART block is responsible for assembling these bits into complete bytes.

Each UART block includes two shift registers, used to convert data from serial to parallel and vice versa. The transmitter is responsible for converting 1 byte of data from parallel to serial and adding start and stop bits to form a transmission frame. The receiver is responsible for converting the received serial signal sequence into complete data bytes.

Transmitted data is controlled by a clock pulse (which is the baud rate), and the receiver will sample the signals, regroup, and restore the original signal. Signal sampling mechanism: one bit of information is divided into 16 sampling points; the signal sampling point is the middle point of the information bit.

2.1.2. Buffer part

The buffer part of the experimental device is the registers located between the DES core and the communication block, which are very important when transmitting high-speed data. In this design, two FiFo buffers of 128 bits for input and 8 bits for output are used.

The FiFo memory block operates on the principle of first in, first out (first data comes out first). In FPGA, these blocks are available as IP cores.

In this experimental device, the input needs to create FiFo1, including the following signals: reset, clk, datain (128 bits), fifo_empty, fifo_full, read_fifo, write_fifo, and dataout (128 bits). In which: fifo_empty, fifo_full will notify whether the fifo is empty or full of data in the register.

Specifically:

When fifo_empty = 1, it means that FiFo is empty, and at this time it is only allowed to write data to FiFo, not to read data from FiFo.

When fifo_empty = 0, it indicates that there is still data in FiFo and allows both reading and writing to FiFo.

When data is written but not read out, FiFo will be full, and the fifo_full=1 signal.

When the FiFo is full, it is only allowed to read data from the FiFo, not write data, and is only allowed to write when fifo_full drops to 0.

FiFo2's output padding is similar to FiFo1's, but FiFo2's input has only 8 bits taken from the output of the DES block's shift register. This FiFo2 block performs the task of storing data and is read out when there is a control signal from the Transfer UART block.

2.1.3. Core part

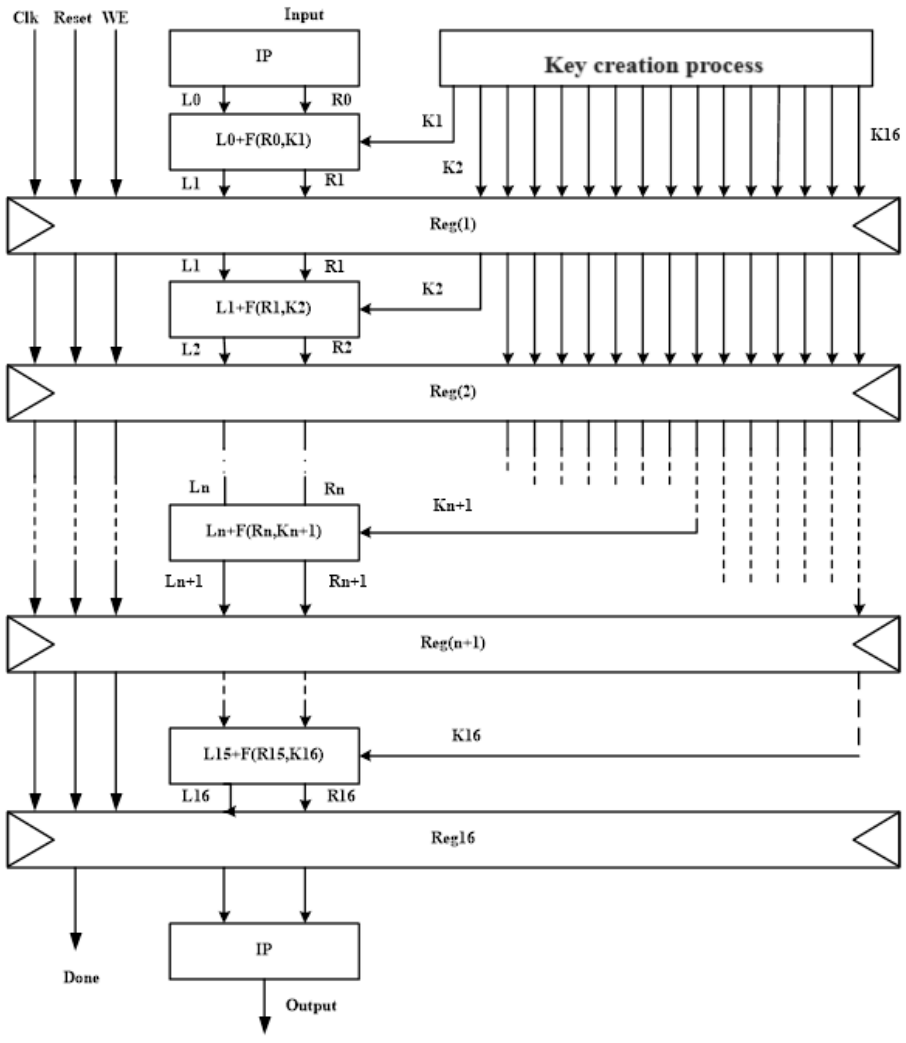

Fig.2. DES encryption block diagram according to pipeline structure

The main design of the core is the DES encryption and decryption block using the pipeline structure. A pipeline structure is a structure where the process of receiving and executing commands is divided into many layers. The

layers execute in parallel, so the system can perform many jobs at the same time. This is a method that is being applied to advanced microprocessors, communication systems, etc. because of its high efficiency and high working speed.

The DES block with pipeline structure is divided into 16 layers separated by registers that operate according to the system's cycles. According to standard principles, the DES algorithm operates in 16 iterations, i.e., in 16 clock cycles. Therefore, if a large number of encryption and decryption times are counted (the data length is large enough), then using a pipeline-style algorithm to complete one encryption and decryption will take one cycle. The clock period thus increases the encoding speed significantly.

The block diagram of the DES encryption block is shown in Figure 2. In Figure 2, the ith register (a total of 16 registers) will store the calculated value from the ith combinational logic block. The combinational logic block is responsible for performing all the work in one loop cycle, including: implementing the F function, the XOR function,... The process of implementing this diagram uses the VHDL language to allow the construction of large designs. from components by describing the architecture in terms of structure. The design includes three sub-blocks: the F function, the key generator function, and the register.

For this pipeline-style structure, for each clock, the layers execute instructions independently of each other, so data on the transmission line is received continuously if there is data, and at the output, we also receive results. corresponding. The done='1' signal tells the operator that the result is available. When there is no output data, this signal drops to 0. The time to complete 1 encryption from the moment there is input data is 16 clock pulses.

Decoding is the exact opposite of encoding. The decryption algorithm of DES is to put the data to be decrypted into the last permutation block, and the order of the keys inserted is key16, key15,... key1. The decoding scheme is shown in Figure 3.

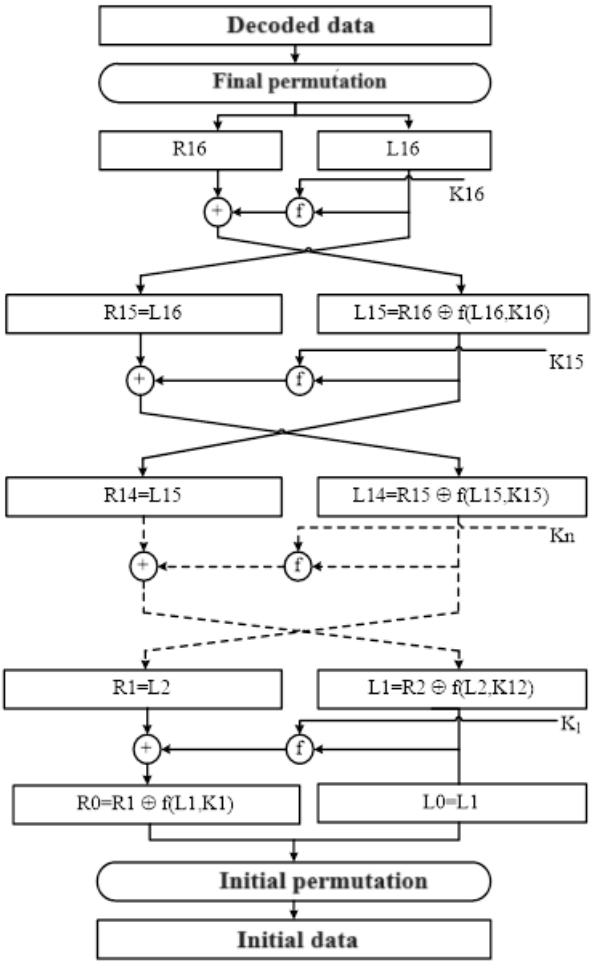

Fig.3. DES decoding block diagram

2.2. *Simulation of functional testing*

After designing the DES encoder and decoder, use ModelSim software to simulate and check the functionality

of the design.

For the input communication block, Receiver_UART is just a 128-bit serial shift register. When 8 bits of data are received, this register shifts. To control the shift register, the Receiver_UART block must generate a signal to let the translator know that a frame has been received for translation. To receive correct data at the output, the width of the signal to be translated correctly is equal to the width of one clock cycle. In this case, we use an additional counter to count the number of translations. When counting 16 translations, it is allowed to write to FiFo1. Normally, when there is no input data, the signal is kept high. To start transmitting information, the Rx signal will switch to a low level for a sufficient period of time. This time is calculated by the time it takes to receive a START bit. After the START bit, data bits are transmitted serially on the Rx, followed by the STOP bit. The transmission frame structure is as follows:

START (1 bit)	DATA (8 bits)	STOP (1 or 1.5 bits)

Data entered into the encryption block including input data and key will be divided into 8-bit data packets, then Start and Stop bits will be added to create a frame as above and put into the receiver_UART block as a line code. is Rx. The output of the receiver_Uart block will strip the data from the frame.

During the data transmission process, to reduce and avoid the influence of noise that causes erroneous reception of sent data, we perform signal sampling. The sampling process proceeds as follows: divide the data bit into 16 samples and the value of the data bit will be determined at the 8th sample (lowest error probability).

To count the number of samples we use a sample counter, in addition we also use another bit counter to check whether we have received enough data in a transmitted frame or not. When dividing a bit into 16 samples, the maximum frequency of this UART receiving block will be: 16*115200 bits/s=1843.2 kHz. The receiver_UART block operates according to the state diagram in Figure 4.

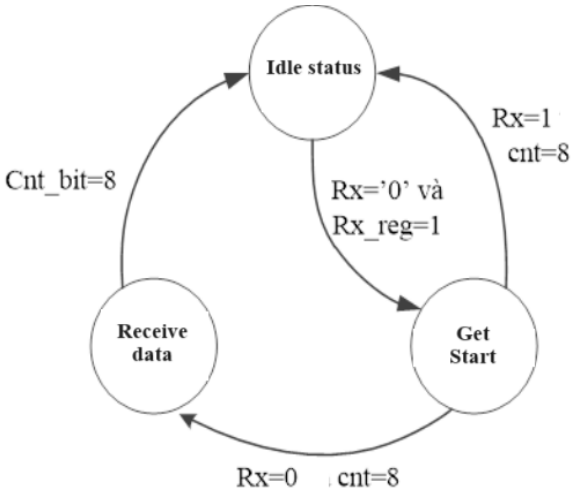

Fig.4. State diagram of the Receiver_UART block

The working principle of the diagram is as follows:

Initially, the receiving block is in an idle state.

When Rx = 0 and Rx_reg = 1, the counter (cnt) will start counting, and at the same time, the system will switch to the Receive start state.

To accurately receive the start pulse, there will be a confirmation at the midpoint, cnt = 8. If at this time Rx = 0, then it is the start pulse and begins to receive input data. In the opposite case, confirm that it is not a start pulse but may just be noise on the transmission line, and the system returns to an idle state, waiting for the start pulse.

Once it is confirmed that there is a start pulse, the system switches to the receive data state and starts the bit counter.

When all 8 bits of data have been received on the transmission line (cnt_bit = 8), the system returns to the idle state, waiting for the next frame.

The wave diagram is shown in Figure 5.

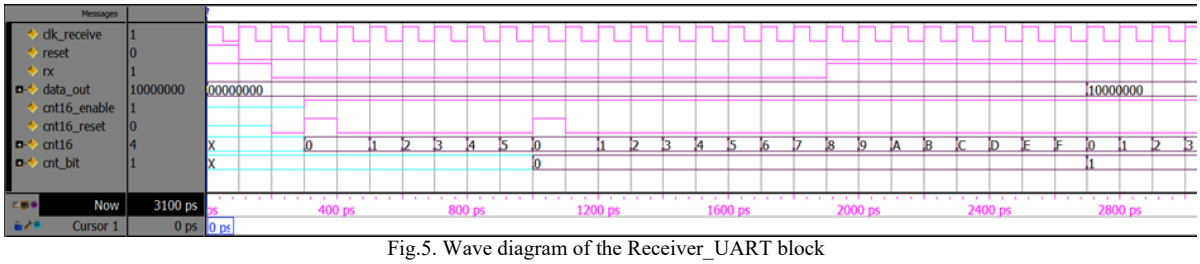

Fig.5. Wave diagram of the Receiver_UART block

The Transfer_UART output communication block has the opposite task compared to the Receiver_UART block, which is to create a line code from the input of a bit string. The input to this block is 8 bits of data read out from FiFo2, which is then framed by inserting a start bit at the beginning and a stop bit at the end. The output of this block is done by shifting each bit to the right.

The operation of the Transfer_UART block is described through the state diagram in Figure 6:

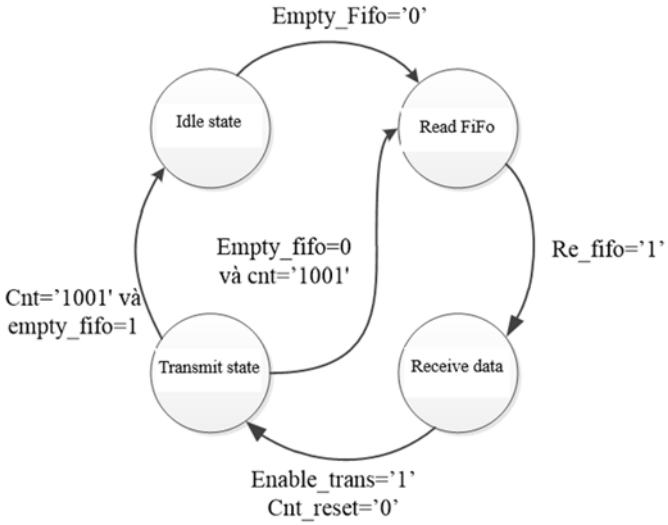

Fig.6. State diagram of the Transfer_UART block

Operating principle of the state diagram:

Normally, the system is in an idle state.

When there is data at the input, if FiFo2 reports that there is data (empty_fifo = 0), then switch to the Read FiFo state.

The transmitting block has signal Re_fifo=1, and the system is now in FiFo reading state. This state lasts one clock pulse, after which the system switches to the Receive Data from Fifo state.

After receiving data and inserting marker bits, start the bit counter and give the transmission permission signal enable_trans=1, then the system switches to the transmit state.

During transmission, count the number of bits received. When receiving enough bits, check the fifo_empty signal to switch to the free state if there is no data on the fifo (cnt = '1001' and empty_fifo = 1).

If there is still data (cnt = '1001' and empty_fifo = 1), switch to the Read FiFo state.

The wave diagram of the Transfer_UART block is shown in Figure 7.

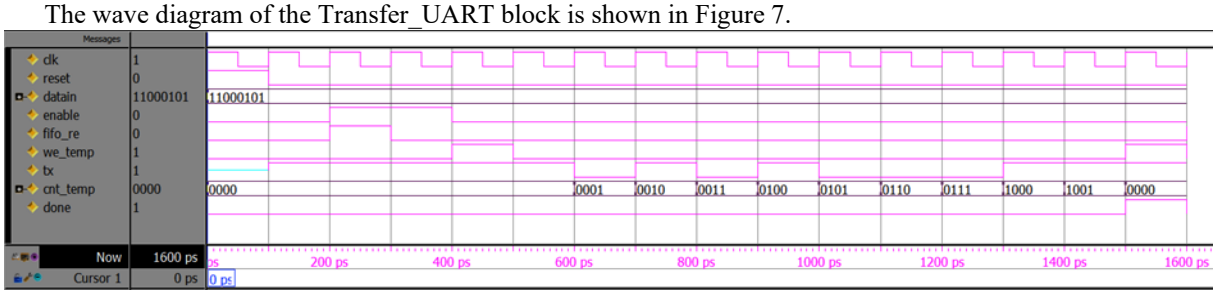

Fig.7. Wave diagram of the Transfer_UART block

The wave diagram of the Transfer_uart block shown in Figure 7 transmits 8 bits with the value 11000101. The transfer block is responsible for inserting the start bit and stop bit. After inserting, its value will be 1110001010 and will be shifted one bit at a time. Therefore, on the output, we receive Tx according to clock pulses (clk), respectively: 0,1,0,1,0,0,0,1,1,1.

The difference between the working frequency of the Transfer_uart block and the Receiver_uart block is that the working frequency of the Transfer_uart block is exactly equal to the baud rate f baud = 9600 bps, 16 times smaller than the working frequency of the Receiver_uart block.

For the encryption process, when giving data and encryption keys as shown in Table 1, the wave diagram of the pipeline structure for DES encryption is as shown in Figure 8.

Input data	1111111111111111	1000000000000001
Key	1111111111111111	3000000000000000
Data after encryption	F40379AB9E0EC533	958E6E627A05557B

Table 1. Data and keys included in DES encryption

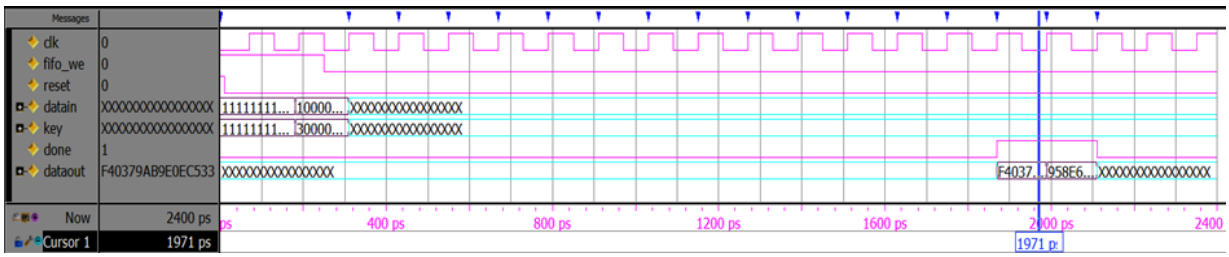

Fig.8. Wave diagram of pipelined DES encryption

For the decoding process, we take the encrypted result above and put it into decryption, and the output we get is the same result as the encrypted input data using the same key. The data and key used for decryption are as shown in Table 2.

Input data	F40379AB9E0EC533	958E6E627A05557B
Key	1111111111111111	30000000000000000
Output data	1111111111111111	10000000000000001

Table 2. Data and encryption keys included in DES decryption

The wave diagram simulation results of the decoding process have the form shown in Figure 9.

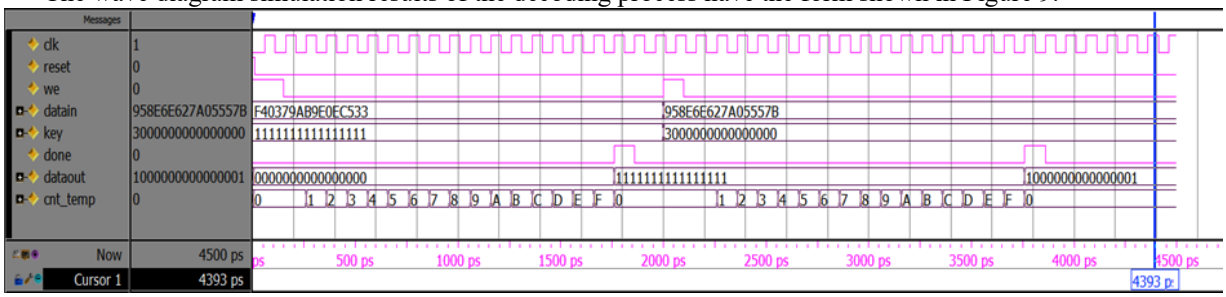

Fig.9. Wave diagram simulating the decoding process

To check the accuracy of the design, we use DevC++ software and then compare these two results with each other. If the same input data and key in both designs give the same result, the comparison table will be shown as "ok"; otherwise, if it is wrong, the result will be shown as "false." The comparison results are shown in the file output, as shown in Figure 10.

File Edit Format View Help

```
455 datain : 00000000000003AA     key : 0123456789ABCDEF     dataout_vhd : 72AFFAC9D1C09461 =  dataout_c : 72AFFAC9D1C09461  ok
456 datain : 00000000000003AB     key : 0123456789ABCDEF     dataout_vhd : D2ADD15A93D60EE0 =  dataout_c : D2ADD15A93D60EE0  ok
457 datain : 00000000000003AC     key : 0123456789ABCDEF     dataout_vhd : FF11CA7075136E07 =  dataout_c : FF11CA7075136E07  ok
458 datain : 00000000000003AD     key : 0123456789ABCDEF     dataout_vhd : 9FAD52584F5C3E04 =  dataout_c : 9FAD52584F5C3E04  ok
459 datain : 00000000000003AE     key : 0123456789ABCDEF     dataout_vhd : E97F431DCF85CADB =  dataout_c : E97F431DCF85CADB  ok
460 datain : 00000000000003AF     key : 0123456789ABCDEF     dataout_vhd : 4706E58CD7ED889B =  dataout_c : 4706E58CD7ED889B  ok
461 datain : 00000000000003B0     key : 0123456789ABCDEF     dataout_vhd : 1E29A6BF987CA7D7 =  dataout_c : 1E29A6BF987CA7D7  ok
462 datain : 00000000000003B1     key : 0123456789ABCDEF     dataout_vhd : DC0115B45195F88D =  dataout_c : DC0115B45195F88D  ok
463 datain : 00000000000003B2     key : 0123456789ABCDEF     dataout_vhd : 59726408DA257F9C =  dataout_c : 59726408DA257F9C  ok
464 datain : 00000000000003B3     key : 0123456789ABCDEF     dataout_vhd : 90B447B4535303D0 =  dataout_c : 90B447B4535303D0  ok
465 datain : 00000000000003B4     key : 0123456789ABCDEF     dataout_vhd : E7F2B98CA3EC8005 =  dataout_c : E7F2B98CA3EC8005  ok
466 datain : 00000000000003B5     key : 0123456789ABCDEF     dataout_vhd : 3E516263DB41EDDD =  dataout_c : 3E516263DB41EDDD  ok
467 datain : 00000000000003B6     key : 0123456789ABCDEF     dataout_vhd : 89307E631B4C2343 =  dataout_c : 89307E631B4C2343  ok
468 datain : 00000000000003B7     key : 0123456789ABCDEF     dataout_vhd : 6AC6DC34431F4894 =  dataout_c : 6AC6DC34431F4894  ok
469 datain : 00000000000003B8     key : 0123456789ABCDEF     dataout_vhd : A53F7288CF02353D =  dataout_c : A53F7288CF02353D  ok
470 datain : 00000000000003B9     key : 0123456789ABCDEF     dataout_vhd : FB7B7DF842155627 =  dataout_c : FB7B7DF842155627  ok
471 datain : 00000000000003BA     key : 0123456789ABCDEF     dataout_vhd : 96C86960119B53B7 =  dataout_c : 96C86960119B53B7  ok
472 datain : 00000000000003BB     key : 0123456789ABCDEF     dataout_vhd : 21B6EE36CCBE5A94 =  dataout_c : 21B6EE36CCBE5A94  ok
473 datain : 00000000000003BC     key : 0123456789ABCDEF     dataout_vhd : F9A069DE4BEEF27D =  dataout_c : F9A069DE4BEEF27D  ok
474 datain : 00000000000003BD     key : 0123456789ABCDEF     dataout_vhd : 0F5C4F223EC22378 =  dataout_c : 0F5C4F223EC22378  ok
475 datain : 00000000000003BE     key : 0123456789ABCDEF     dataout_vhd : 9E1727568D5AD5DD =  dataout_c : 9E1727568D5AD5DD  ok
476 datain : 00000000000003BF     key : 0123456789ABCDEF     dataout_vhd : E2E438C716C309CD =  dataout_c : E2E438C716C309CD  ok
477 datain : 00000000000003C0     key : 0123456789ABCDEF     dataout_vhd : 65A3AB3A717CF9DF =  dataout_c : 65A3AB3A717CF9DF  ok
478 datain : 00000000000003C1     key : 0123456789ABCDEF     dataout_vhd : 6D5231FF7E641B3D =  dataout_c : 6D5231FF7E641B3D  ok
479 datain : 00000000000003C2     key : 0123456789ABCDEF     dataout_vhd : 82668F2FA14E9496 =  dataout_c : 82668F2FA14E9496  ok
```

Fig.10. Check the accuracy of the design using DevC++ software.

2.3. *Realized on FPGA*

2.3.1. DES encryption and decryption experiment set on the FPGA platform

The specialized digital design circuit built on the FPGA board is used to load the DES encryption and decryption core to perform digital communication experiments. This set of experiments serves the learning and teaching needs of data transmission techniques and the scientific research of lecturers and students. The device includes two boards: The motherboard and the core board are arranged on two different floors.

The QMTECH_ARTIX-7_XC7A35T motherboard measures 108.71mm x 134.62mm, providing several peripheral interfaces to meet different connectivity and application needs, as shown in the image shown in Figure 11.

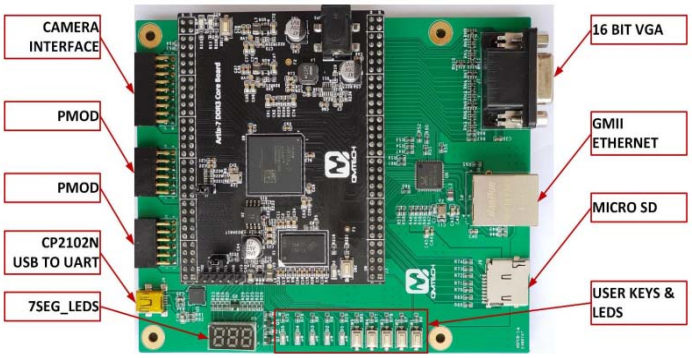

Fig.11. Actual image of motherboard, QMTECH_ARTIX-7_XC7A35T

The details of these user interfaces are as follows:

+ MiniUSB to UART serial conversion port, using the Silicon Labs CP2102N-A01-GQFN28R chip designed by Silicon Labs, including a full-speed USB 2.0 function controller, USB transceiver, oscillator, and UART.
+ 16-bit VGA display interface (RGB565), using resistor dividers.
+ GMII Ethernet interface, using Realtek's RTL8211EG chip.
+ MicroSD/TF card interface.
+ CMOS/CCD camera interface, using an 18-pin female connector.

The QMTech® XC7A35T DDR3 core board uses a Xilinx Artix® 7 FPGA chip with a MicroBlaze™ soft processor. With a relatively rich amount of logic resources and large DDR3 memory (MT41J128M16, 1066 MB/s), The core board is capable of meeting high-speed hardware design problems while still ensuring low power consumption and low cost.

The core board has 108 non-multiplexed FPGA IOs to expand custom modules, such as UART modules, CMOS/CCD camera modules, LCD display modules, HDMI/VGA modules, etc. The actual image of the QMTech® XC7A35T DDR3 core board is shown in Figure 12.

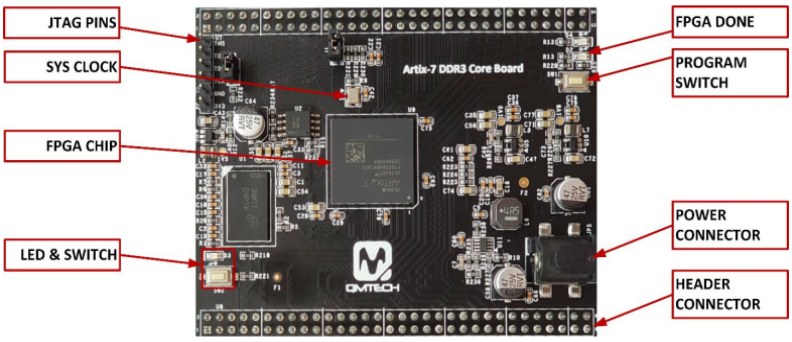

Fig. 12. Actual image of the QMTECH_ARTIX-7_XC7A35T core board

The specifications of the QMTech® XC7A35T DDR3 core board are as follows:

+ System clock frequency: 50 MHz;
+ Block RAM resource: up to 1,800 KB;
+ Logic Cell Resources: 33,280;
+ On-board MT28QL128 SPI Flash, 16M bytes for user configuration code;
+ Internal memory: 256MB Micron DDR3, MT41J128M16JT-125: K;
+ 3.3V power supply, using the MP2315 wide DC/DC input range;

+ 64p socket, 2.54mm pitch headers for expanding input/output (user IOs);
+ 2 user switches;
+ 3 LEDs (user LEDs);
+ JTAG interface, using a 6p socket with a 2.54mm pitch header;
+ Printed circuit size: 67mm x 84mm;
+ Power source: 1A, 5V DC, plug type DC-050, 5.5mm x 2.1mm;

Besides the peripheral interfaces already on the QMTECH_ARTIX-7_XC7A35T motherboard, to add important interfaces such as ADC converters, DACs, and audio interfaces (microphone, headphone), another peripheral circuit is installed. further developed and connected to the motherboard via standard Digilent PMOD female jack assemblies. The hardware design diagram of the peripheral circuit is shown in Figure 13.

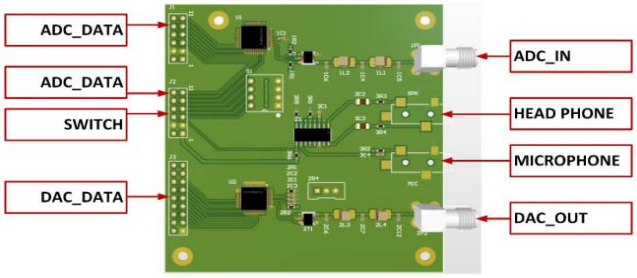

Fig. 13. Hardware structure diagram of the peripheral connection circuit

The specific specifications of some component modules on the peripheral board are listed below:
+ ADC AD9218, 10 bit, 80 Mbps, SMA.
+ DAC AD9763, 10 bit, 125 Mbps, SMA.
+ Audio codec SI3000 4-8 Kbps, 16 bit, 3.5mm mic, 3.5mm headphone.
+ DP-04 2.54mm 4-pin bit switch.

For the digital circuit design process using the Xilinx Vivado tool, after creating a project and adding source code files, we perform design simulation using the SIMULATION function. Next, perform circuit design analysis (RTL analysis) to receive the circuit diagram in Netlist form. The results of the encryption circuit analysis are shown in Figure 14, and the mahoa_des block is shown in Figure 15.

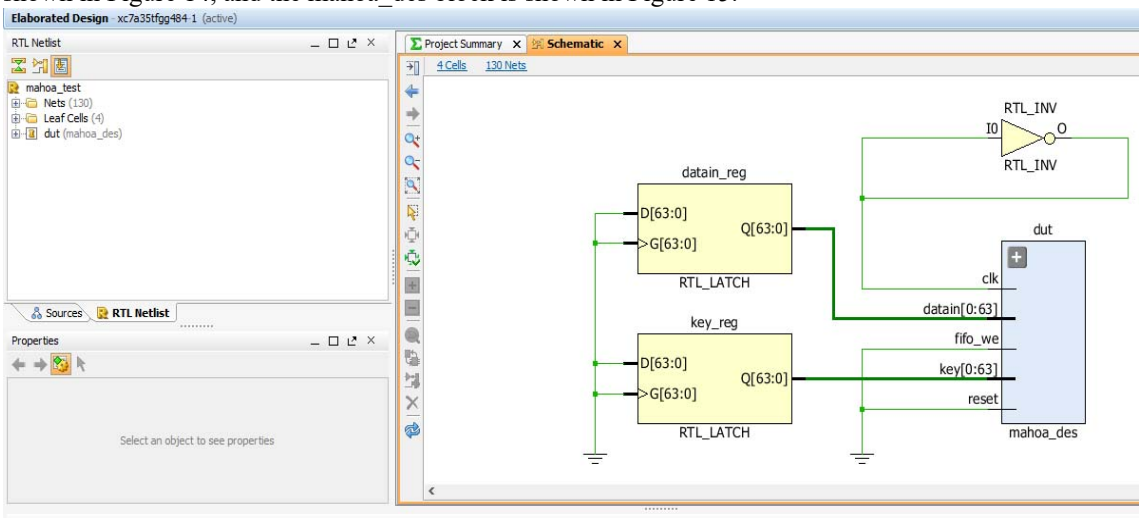

Fig.14. Block diagram of the designed DES encryption circuit

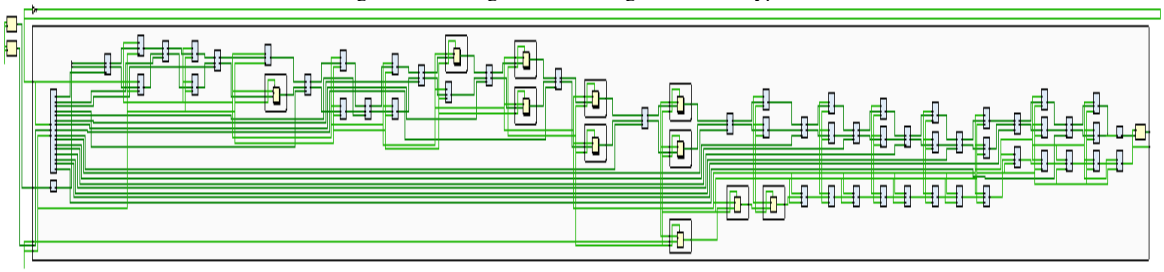

Fig.15. Mahoa_des (dut) encoding block structure

Performing the same process for the decoding block, we get the results of analyzing the design of the DES

decoding block on Xilinx Vivado, shown in Figure 16.

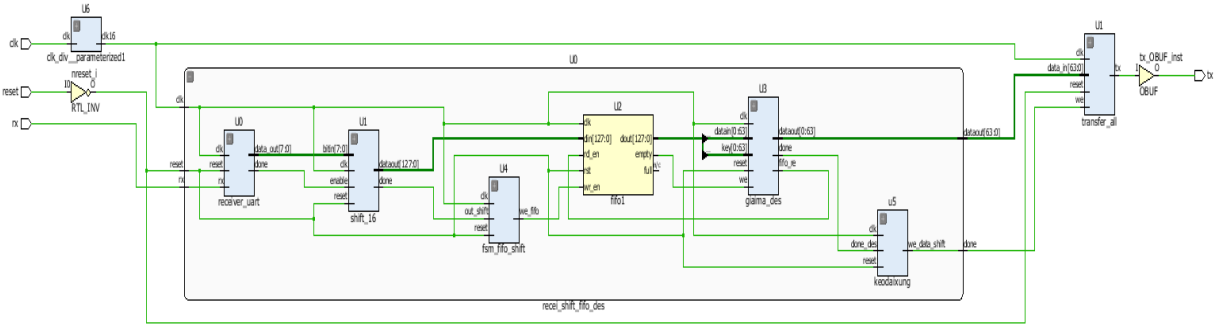

Fig. 16. DES decoding block structure

After having the Netlist design of DES encryption and decryption, we proceed to realize the design (implementation). The results of this stage are receiving the configuration file for the FPGA and providing results in the tool's reports (about resources and speed).

2.3.2. Connect the experiment set

In data transmission systems, there are two ways to put signals on the transmission line: serial and parallel. Parallel transmission is usually carried out over short distances, while serial transmission is often carried out when the transmission distance is large.

In asynchronous mode, clock pulses are generated separately at the transmitter and receiver based on the nominal frequency corresponding to the bit rate, or baud rate.

To perform this experiment, students first need to understand the asynchronous serial information transmission and reception protocol. Serial information is transmitted along a single wire, and the information bits are encoded according to the voltage level on the wire, as shown in Figure 17.

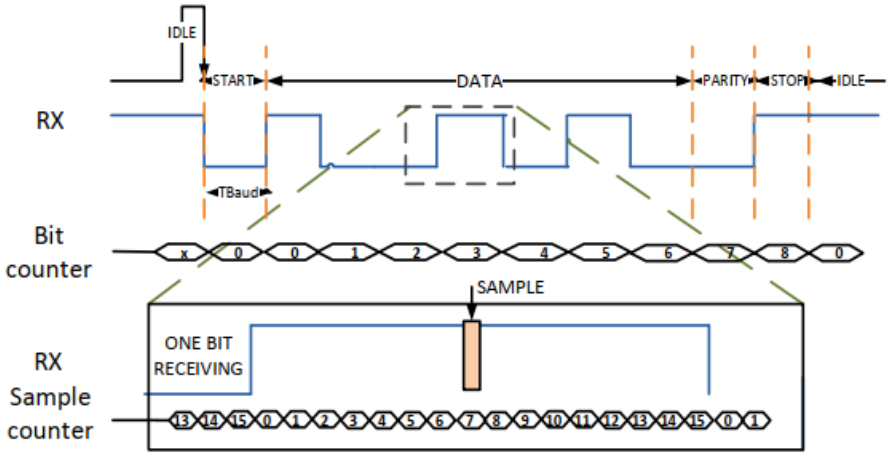

Fig. 17. Asynchronous serial communication protocol

Figure 17 depicts the input signal of a block that receives signals from a serial line. In the idle (IDLE) state with no data, the signal is held high. To start transmitting information, the Rx signal will switch to a low level for a large enough period of time; this time is equal to the time it takes to receive 1 bit of information corresponding to the transmission speed of the port, called TBaud = 1/FBaud.

After this START bit, data bits are transmitted serially on Rx, corresponding to the DATA state. The COM port can be configured to transmit and receive 6, 7, or 8 bits of information. After the end of transmission, there may be an additional parity check bit: PARITY. A STOP bit (logic level 1) is held for a time equal to 1; 1.5; or 2 TBaud. When the transmission ends, the Rx returns to its idle state at high voltage.

The serial transmission line is simple and not high-speed; usually, the baud rate varies from 1200 to 115200 bit/s. This transmission line uses devices that are independent of each other and are affected by significant interference. On the other hand, the receiver and splitter operate asynchronously (not at the same system clock), so a reception mechanism is needed. to avoid errors arising.

One bit of information is divided into 16 sample points, which are determined by a sample counter. The probability of error at the first and last sample positions is highest, and the probability of error at the middle sample position is lowest, so we choose the middle sampling point, meaning the information bit is received by Rx in state

counter = 8.

According to this design, the highest speed can reach 115,200 bits/s, and the sampling counter frequency is equal to 115,200 x 16 = 1,843,200 Hz < 2 MHz. In reality, it is not always possible to divide information bits into 16 samples, which can be divided larger or smaller but still comply with the sampling theorem.

Based on the above protocol, it is possible to design a block that receives information from the COM port with the following configuration:

- Default transmission rate: Baud rate = 9600 bit/s;
- 7-bit ASCII character transmission structure;
- Parity bit support;
- One STOP bit.

The UART asynchronous serial information transmission and reception design diagram is depicted in Figure 18.

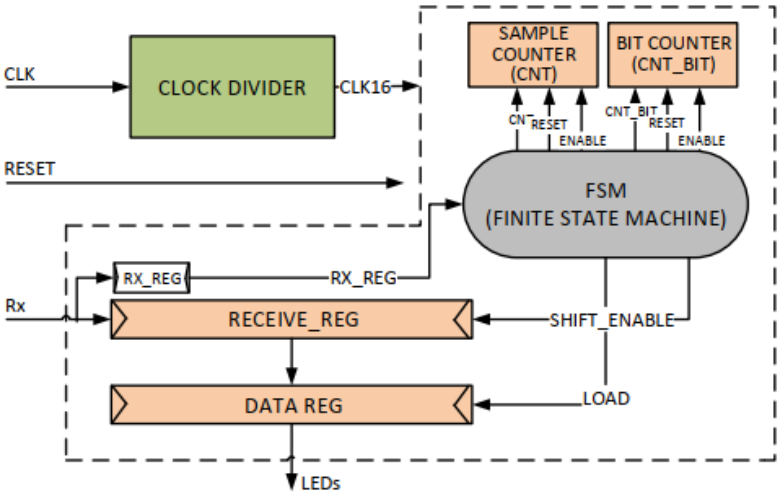

Fig. 18. Block diagram of serial information transmission and reception on the FPGA Kit

The Rx signal is an input from the computer's COM port. The base oscillation frequency of the circuit is taken from a crystal oscillator with a frequency of CLK = 50 MHz; this frequency is divided by a frequency divider with a division factor of 9600 x 16 before being used as a clock. CLK16 system for the entire block, in which 9600 is the Baud rate.

The reset signal is also taken from a control pin of the FPGA. The two signals CLK16 and RESET are common to all other blocks in the receiver. To check the status of receiving input data, information bits will be assigned to the corresponding LEDs on the circuit.

Experimental circuit connection diagram as shown in Figure 19.

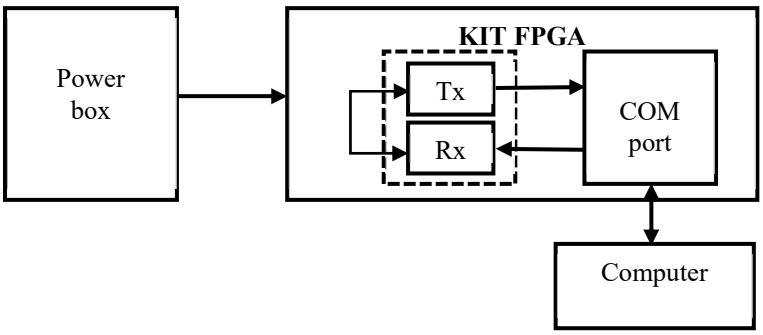

Fig. 19. Experimental circuit connection diagram

In Figure 19, data is transmitted and received between the computer and the FPGA circuit via the UART protocol. In particular, data is transmitted from the computer to the FPGA circuit via the Rx line, and data is transmitted from the FPGA circuit to the computer via the Tx line. Data is displayed partially or completely on the LEDs of the FPGA circuit and through the data transmission and reception interface, as shown in Figure 20. The power box has the function of converting from 220V/50Hz AC power to DC power. Stable direction: 1A, 5V DC.

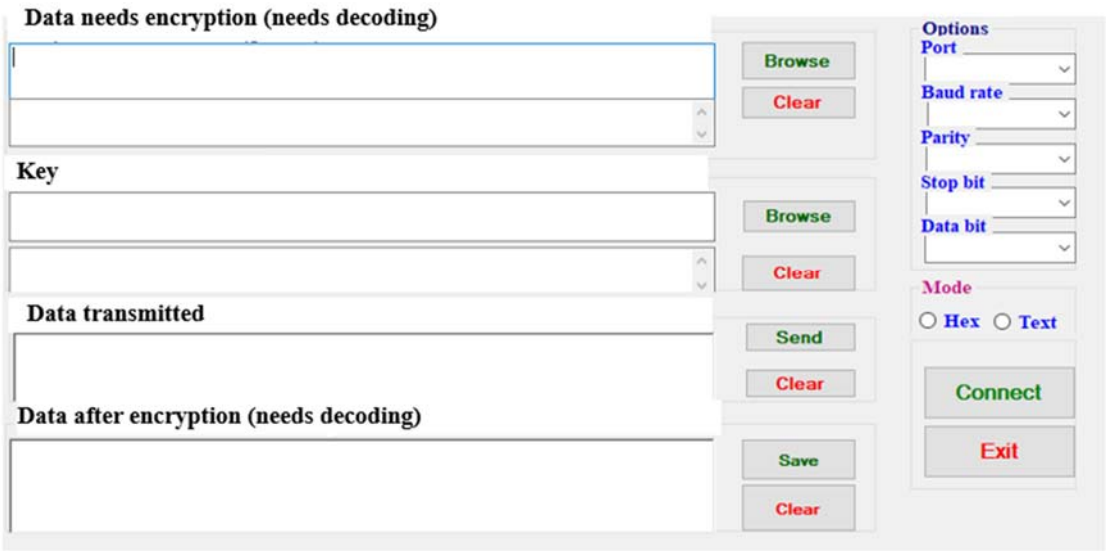

Fig.20. Data transmission and reception interface

The data transmission and reception interface includes the following:

+ Input: data input window and code key input window. In these windows, data can be entered directly or taken from the file. Correspondingly, each window has buttons to search data files or delete entered data.

+ Transmission mode options: include COM port selection window, baud rate selection window (default is 9600baud), parity check mode selection window, STOP bit length selection window, and length selection window data.

+ The display part includes a window displaying data before transmission and a window displaying data received from the device. These data can be displayed in text or hexadecimal format via the display mode button.

+ Control button: button to send data, clear display window, store data, button to connect computer to device, and button to exit program.

The DES encryption and decryption experiment set after fabrication and editing is shown in Figure 21.

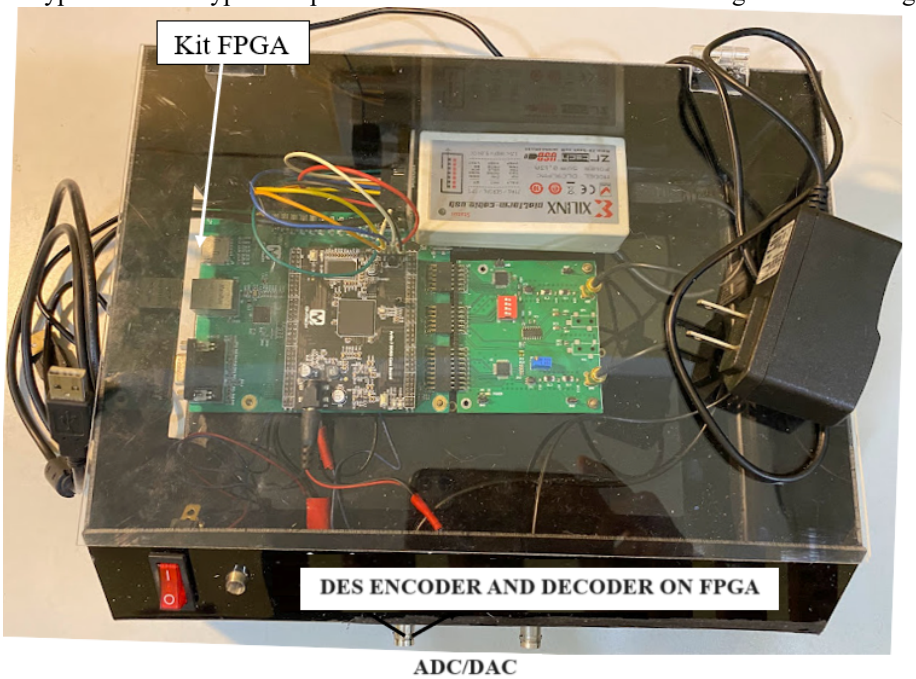

Fig. 21. Product image

The experimental equipment set includes the following components: a power converter from 220V/50Hz to 5V/DC; a FPGA kit; a chip loader; connection cables; a power switch; a power light; and an ADC/DAC input/output conversion port. The components of the device are covered with a protective cover made of mica to avoid dust, mold, and impact.

2.4. *Develop experimental exercises*

Based on the DES encoder and decoder loaded into the FPGA, the author built three reliable data transmission experiments by designing the connection interface between the computer and experimental equipment according to the serial wireless standard. UART sync.

2.4.1. Experiment 1: UART asynchronous serial data transmission

Experiment 1 aims to consolidate knowledge about the UART asynchronous serial data transmission method, how to convert parallel and serial signals using shift registers, and the process of creating data frames with START bits, STOP, and PARITY. In the experiment, students use a computer experiment program to transmit and receive data in text or an available file format, with different options for transmission speed, odd or even check mode, and stop bit length. Students need to save the transmission and reception results to compare, comment on, and write reports according to the form. The experiment helps students master the principles of serial data transmission, how to frame data, and laboratory operation skills.

Data sent	COM port	Speed (Kbps)	Parity	BitStop	Data length	Data received
11011011	COM1	9600	Odd	first	8	11011011

2.4.2. Experiment 2: Encrypt data with DES

Experiment 2 aims to test the data encryption process with the DES (Data Encryption Standard) standard for data security and authentication. In the experiment, students use a computer experiment program to encrypt and decrypt data in the form of text or an existing file with a given encryption key. Students need to enter data and an encryption key into the program, set parameters such as parity, stop bit, and data length, then press the send button to put the data into the FPGA to perform the encryption process and receive the ciphertext. . Students need to save the coding results to compare, comment on, and write reports according to the template. The experiment helps students understand the importance of data encryption and master the principles of encryption according to the DES standard.

Data	Key code	Parity	BitStop	Data length	Code
100000011111111	3000000000000000	Odd	first	8	

2.4.3. Experiment 3: Decrypt data with DES

Experiment 3 aims to perform data decryption with the DES standard to restore original data from the ciphertext. In the experiment, students use a computer experiment program to decrypt data in the form of available ciphertext with a given encryption key. Students need to enter the ciphertext and encryption key into the program, set parameters such as parity, stop bit, and data length, then press the send button to put the data into the FPGA to perform the decoding process and receive the results. Students need to save the decoding results to compare, comment on, and write reports according to the sample. The experiment helps students understand the importance of data encryption and master the principles of decryption according to the DES standard.

Code	Key code	Parity	BitStop	Data length	Plain copy
100000011111111	3000000000000000	Odd	first	8	

3. Conclude

This paper has researched and developed a set of DES encryption and decryption experiments on FPGA devices with the goal of improving information security knowledge and skills for students and researchers. Using VHDL language and Xilink Vivado software, the author has designed and realized a DES encoder and decoder on an FPGA platform and connected it to a computer via the UART serial standard. The experiment set is used in three experiments on data transmission techniques, allowing users to perform data encryption and decryption with DES. Research results show that the test set operates stably and accurately, has high speed, relative safety, and low energy consumption. The experiment not only illustrates how DES works but can also be applied to future security and communications applications. This paper has contributed to the development of FPGA technology in the field of information security and opened up new research directions for scientists.

References

[1] Aitkhozhayeva, Y. and Tynymbayev, S. (2014). Aspects of hardware modulation in asymmetric cryptography. Journal Vestnik NAS RK, 5, 88–93. ISSN-1991-3494.
[2] Barrera, A., Cheng, C.-W. and Kumar, S. (2020). A fast implementation of the Rijndael substitution box for cryptographic AES. In Proceedings - 2020 3rd international conference on data intelligence and security, pp. 20–25. South Padre Island, TX: IEEE. https://doi.org/10.1109/icdis50059.2020.00009
[3] Gnatyuk, S., Okhrimenko, A., Kovtun, M., Gancarczyk, T. and Karpinskyi, V. (2016). Method of algorithm building for modular reducing by irreducible polynomial. In Proceedings of the 16th international conference on control, automation and systems, pp. 1476–1479. Gyeongju, Korea (South): IEEE. https://doi.org/10.1109/iccas.2016.7832498
[4] Kalimoldayev, M., Tynymbayev, S., Gnatyuk, S., Ibraimov, M. and Magzom, M. (2020). The device for multiplying polynomials modulo an irreducible polynomial. News of the National Academy of Sciences of the Republic of Kazakhstan Series of Geology and Technical Sciences, 2(434), 199–205. https://doi.org/10.32014/2020.2518-170X.60
[5] Kalimoldayev, M., Tynymbayev, S., Gnatyuk, S., Magzom, M., Khokhlov, S. and Kozhagulov, Y. (2020). Matrix multiplier of polynomials modulo analysis starting with the lower order digits of the multiplier NEWS of the Academy of Sciences of the Republic of Kazakhstan Series of Geology and Technical Sciences. News of the National Academy of Sciences of the Republic of Kazakhstan, 4(436), 181–187. https://doi.org/10.32014/2019.2518-170X.113
[6] Kalimoldayev, M., Tynymbayev, S., Magzom, M., Ibraimov, M., Khokhlov, S., Abisheva, A. and Sydorenko, V. (2019). Polynomials multiplier under irreducible polynomial module for highperformance cryptographic hardware tools. In CEUR workshop proceedings. 15th international conference on ICT in education, research and industrial applications. Integration, harmonization and knowledge transfer, ICTERI 2019, Vol. 2393, pp.729–737.
[7] Kumar, A., Vishnoi, P. and Shimi, S. L. (2019). Smart grid security with cryptographic chip integration. EAI Endorsed Transactions on Energy Web, 19(23), 6. https://doi.org/10.4108/eai.13-7-2018.157037
[8] Kumar, N., Mishra, V. M. and Kumar, A. (2021). Smart grid and nuclear power plant security by integrating cryptographic hardware chip. Nuclear Engineering and Technology, 53(10), 3327–3334. https://doi.org/10.1016/j.net.2021.05.006
[9] Liu, W., Fan, S., Khalid, A., Rafferty, C. and O'Neill, M. (2019). Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA. In IEEE transactions on Very Large Scale Integration (VLSI) systems, Vol. 27(10), pp. 2459–2463. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/tvlsi.2019.2922999
[10] Saini, R., Jain, M. and Suneja, K. (2020). FPGA Based Design of Speech Encryption and Decryption for Secure Communication. Communications in Computer and Information Science, 1170, 37–49. ISSN: 18650929.

Authors Profile

Vuong Thuy Linh was born in Hanoi, Vietnam, on March 9, 1977. She holds a Master's degree in Information Technology. Her primary research interests include voice and image processing, machine learning, smart control, smart grid demand response, building automation systems, renewable energy, and applications of the Internet of Things. Currently, she has authored or co-authored more than 10 international journal papers.
She is currently a lecturer at the Department of Information Technology, Faculty of General Education, University of Labor and Social Affairs, located at No. 43, Tran Duy Hung Street, Trung Hoa Ward, Cau Giay District, Hanoi City, Vietnam.

Le Ngoc Giang was born in Hanoi, Vietnam, on July 21, 1975. He has a PhD in Electric Power System and its Automation. His main research interests are power system control and stability analysis, smart grid demand response, building automation systems, renewable energy, and applications of the IoT, as well as designing automatic control systems for flying devices. At present, he is the author or coauthor of more than 80 international journal papers and the owner of two invention patents. Currently, he is the Head of Metrology Department, Faculty of Fundamental Technical, AD-AF Academy of Vietnam, 12713 Kim Son, Son Tay, Ha Noi, Viet Nam.
Email: lengocgianglinh@gmail.com
Scopus 56071251000
https://orcid.org/0000-0002-0198-2857