An Embedded Encrypted Data Storage System based on FPGA

Jiesi Na [a], Hongtao Qi [b], Jianyu Zhang [c]

The first laboratory, Kunming Shipbuilding Equipment Research and Test Center, Kunming, China

[a] rsw1986@mail.nwpu.edu.cn, [b] qihongtao2021@163.com, [c] zhangjianyu2021@163.com

Abstract. For the embedded storage system with data confidentiality requirements, an embedded storage platform with triple DES as the encryption algorithm, eMMC as the storage medium, UART and UDP as the communication interface, and FPGA as the main control chip is designed. in this design, the data encryption key can be freely set by the host computer during data storage. When reading the data, the host computer can obtain the plaintext data by providing the correct key without decrypting the host computer. The experimental results show that the design can normally complete the encrypted storage and decrypted reading of data, the storage rate can reach about 18MBps, and the reading speed can reach about 16MBps.

Keywords: FPGA; 3DES; eMMC; Data Storage.

1. Introduction

Embedded storage systems (hereinafter referred to as "system") are widely used in various customized measurement and acquisition equipment. For field measurement and acquisition equipment, especially for autonomous mobile acquisition equipment, the security issue of collecting and recording data cannot be underestimated. This paper designs a low-cost embedded data storage system with low-end FPGA as the main control chip and eMMC as the high-speed storage carrier, which can perform triple DES encryption on data[1].

The system can access measurement and acquisition equipment (hereinafter referred to as "equipment"), receive control commands and encryption keys sent by the equipment, and encrypt and record external sampling data. After the measurement and acquisition equipment is completed, the system can work independently from the equipment, connect the system to the upper computer, input the correct key through the upper computer, and control the reading of the data in the storage carrier. The system application scenario is shown in Figure 1.

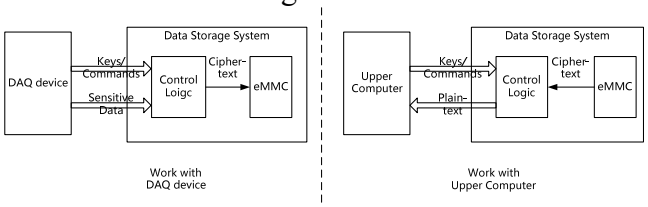

Fig 1. Data storage system application scenarios

2. Overall Design

2.1 Hardware Environment and Software Architecture

All control logic of the data storage system is implemented in FPGA, and the peripheral hardware environment is relatively simple, mainly including some physical driver chips and eMMC memory chips for interface communication to achieve low-cost design architecture.

Figure 2 shows the overall architecture block diagram of the data storage system. The main control FPGA model is Altera-CycloneIV-EP4CE75, including 75,408 Logic Elements (logic resources in Altera FPGA), 2,745Kbits embedded memory, 200 embedded multipliers, and a maximum of 426 user IO.

The eMMC memory model is Micro-MTFC16G, 16GB storage capacity, compatible with eMMC v4.51 protocol[2][3]. The Ethernet PHY chip model is RTL8211E, which supports up to Gigabit

networks and is compatible with the 1000Base-T IEEE 802.3ab protocol[4]. The RS485 bus driver model is TI SN65HVD23, which supports a maximum transmission rate of 25Mbps over a distance of 160m, and is compatible with the TIA/EIA-485 protocol [5].

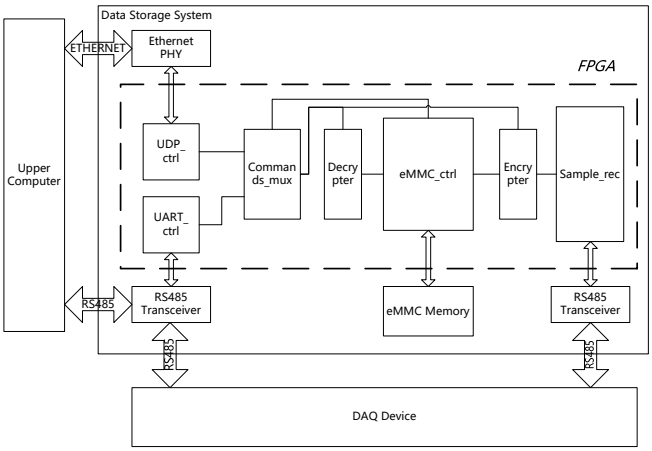

Fig 2. Overall architecture of the data storage system

This paper mainly focuses on the logic design inside the FPGA. The dashed box in Figure 2 divides the functional modules of the internal control logic of the FPGA from the perspective of software, and the lines represent the interfaces between the functional modules within the software. The main module functions:

- UDP_ctrl(UDP communication controller): implements standard UDP protocol, the maximum bandwidth in the project can be 16MBps, and the high bandwidth requirement is mainly used for fast reading of stored data.
- UART_ctrl(UART communication controller): realizes standard UART communication with verification, the baud rate is 115200bps, mainly used to transmit control commands[6].
- Encrypter & Decrpter(Encryption and decryption module): realize the standard 3DES encryption and decryption algorithm, the algorithm key is input by the host computer, when the host computer does not input the key, the system default key is used for encryption and decryption.
- eMMC_ctrl(eMMC controller): compatible with eMMC v4.51 protocols, supports single-address read and write, multi-address read and write, stream read and write commands, the maximum operating clock is 25MHz, and the read and write rate can reach about 18MBps.
- Sample_rec(Sampling Data Receiver): use RS485 bus, adopt customized protocol, receive data from measurement and acquisition equipment, RS485 bus baud rate is 10Mbps.

2.2 System Workflow

After the data storage system is powered on, all functional modules work in parallel. The eMMC controller will initialize and authenticate the eMMC. If the initialization fails, it will report an error and stop working. If the initialization is successful, it will wait for the control command.

The sampling data receiving module starts to work, receives the data sampled from the measuring equipment, and loads the data into the FIFO buffer.

The UART module waits for control commands. If a 3DES key is received, the encryption module updates the key for data encryption; if a write command is received, the data is read from the FIFO, and the ciphertext is written into eMMC after encryption; if a read command is received, Then read the ciphertext from eMMC, and send the decrypted plaintext data to the PC through the UDP interface.

The workflow of the data storage system is shown in Figure 3.

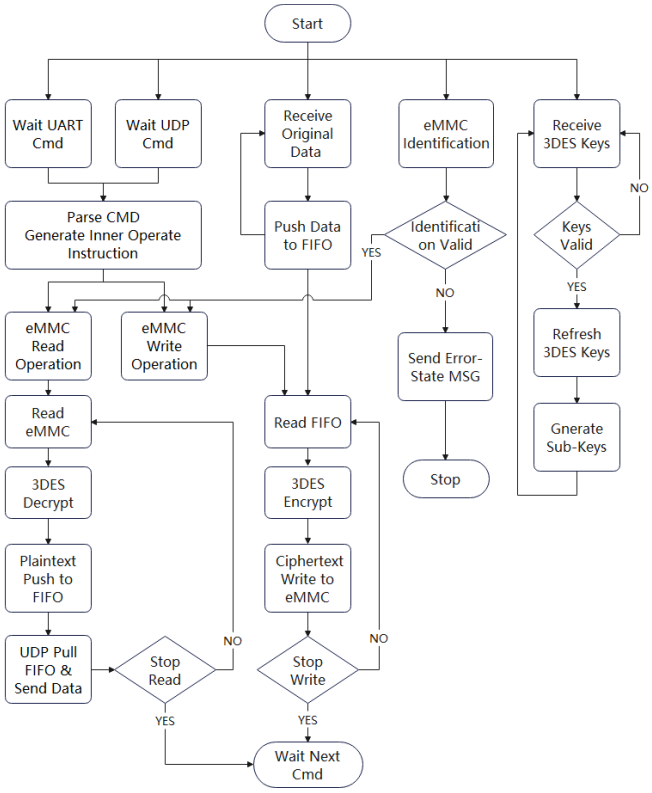

Fig 3. System workflow

3. Detailed Design

3.1 Dual External Interface

For compatibility, the "system" implements external communication interfaces conforming to the standard UART protocol and UDP protocol respectively. In order to share the data frame decoding module within the logic, the UART interface and the UDP interface use a unified high-level communication protocol. When the data packet is sent by UART, each byte is encapsulated as the payload of a UART frame and sent, and when it is transmitted by UDP, the data packet is sent as the payload segment of the Ethernet UDP protocol.

3.2 UDP Controller

The UDP controller implements the standard UDP protocol. The UDP protocol is relatively simple. The sending and receiving logic can be encoded and decoded according to the UDP protocol. The main difficulty lies in the IO design and timing constraints of the FPGA and PHY chips[7]. In the PHY Gigabit Ethernet working mode, the working clock of the UDP module is provided by the PHY chip, which is a 125MHz clock, and a true dual-port RAM is used to process the UDP module and other logic across the clock. The communication between the UDP module and the PHY chip is the GMII protocol, which is a source-synchronous transmission model with 8-bit data. Setting the correct IO timing constraints can guide the direction of the timing-driven compiler and verify the timing of the design.

```
#**************************************************************
# Create Generated Clock
#**************************************************************
create_clock -name "e_rxc" -period 8.000 [get_ports {e_rxc}] -waveform {0.000 4.000}
create_generated_clock -name {e_gtxc} -source [get_ports {e_rxc}]
    -master_clock {e_rxc}  [get_ports {e_gtxc}]

#**************************************************************
# set input_delay & output_delay
#**************************************************************
set_input_delay -clock { e_rxc } -max -add_delay 5.5 [get_ports e_rxd\[*\]]
set_input_delay -clock { e_rxc } -min -add_delay 0.5 [get_ports e_rxd\[*\]]
set_output_delay -clock { e_gtxc } -max -add_delay 2 [get_ports e_txd\[*\]]
set_output_delay -clock { e_gtxc } -min -add_delay 0 [get_ports e_txd\[*\]]
```

Fig 4. GMII bus timing constraints

The timing constraints are set as shown in Figure 4. In the figure, “e_rxc” is the bus data receiving clock, and “e_gtxc” is the bus data sending clock.

3.3 eMMC Controller

As a popular embedded memory chip in recent years, eMMC is widely used in smart consumer electronic products and mobile multimedia devices due to its advantages of simple interface, large storage capacity, fast transmission speed and high integration.

The eMMC protocol stipulates that in order to be backward compatible with old devices, eMMC first works in the low-speed mode to initialize the device, and after the initialization is completed, it enters the high-speed mode[8][9]. The two operating modes in this paper use 100kHz and 25Mhz operating clocks respectively.

eMMC controller uses the asynchronous FIFO to perform cross-clock processing between the controller and other logic.

The communication between the eMMC controller and the eMMC device is a source-synchronous transmission model with 8-bit data. Setting the correct IO timing constraints can guide the direction of the timing-driven compiler and verify the timing of the design.

The timing constraint settings are shown in Figure 5. In the figure, “emmc_clk_bus_o_low” is the clock for low-speed mode, and “emmc_clk_bus_o_high” is the clock for high-speed mode.

```
#**************************************************************
# Create Generated Clock
#**************************************************************
create_generated_clock -name {emmc_clk_bus_o_high} -source [get_pins {*|pll1|clk[1]}]
    -master_clock {*|pll1|clk[1]} [get_ports {emmc_clk_bus_o}]
create_generated_clock -name {emmc_clk_bus_o_low} -source [get_pins {*|pll1|clk[2]}]
    -master_clock {*|pll1|clk[2]} [get_ports {emmc_clk_bus_o}] -add

#**************************************************************
# set input_delay & output_delay
#**************************************************************
set_input_delay -clock { emmc_clk_bus_o_low } -max -add_delay 13.7 [get_ports emmc_dat_bus_io\[*\]]
set_input_delay -clock { emmc_clk_bus_o_low } -min -add_delay 2.5  [get_ports emmc_dat_bus_io\[*\]]
set_input_delay -clock { emmc_clk_bus_o_high } -max -add_delay 13.7[get_ports emmc_dat_bus_io\[*\]]
set_input_delay -clock { emmc_clk_bus_o_high } -min -add_delay 2.5 [get_ports emmc_dat_bus_io\[*\]]
set_output_delay -clock { emmc_clk_bus_o_low } -max -add_delay  3  [get_ports emmc_dat_bus_io\[*\]]
set_output_delay -clock { emmc_clk_bus_o_low } -min -add_delay -3  [get_ports emmc_dat_bus_io\[*\]]
set_output_delay -clock { emmc_clk_bus_o_high } -max -add_delay 3  [get_ports emmc_dat_bus_io\[*\]]
set_output_delay -clock { emmc_clk_bus_o_high } -min -add_delay -3 [get_ports emmc_dat_bus_io\[*\]]
```

Fig 5. eMMC bus timing constraints

3.4 Triple DES Encrypter and Decrypter

The 3DES algorithm is a symmetric encryption algorithm, which is an extension of the DES algorithm. It increases the length of the key and replaces the already insecure DES algorithm. The essence of the 3DES algorithm is to perform three DES operations on the data block with different keys. Each DES uses a different key (it can also be the same, but the encryption strength will be weakened). The key length of the 3DES algorithm is 64bit*3=192bit. So far, the 3DES algorithm has not been proven to be cracked[10][11].

The encryption and decryption process of 3DES algorithm can be expressed by formula (1):

$$\begin{cases} C = E_{k3}(D_{k2}(E_{k1}(P))) \\ P = D_{k1}(E_{K2}(D_{k3}(C))) \end{cases} \tag{1}$$

In formula (1) :

C—Ciphertext.

P—Plaintext.

$D_{kn}(P)$—DES encryption operation with Kn as the key and P as the plaintext.

$E_{kn}(C)$—DES decryption operation with Kn as the key and C as the ciphertext.

There are only logical operations and table lookup operations in the encryption and decryption and key generation process in the operation process, and there is no complex mathematical operation, which is more suitable for FPGA implementation.

4. Test Evaluation

Figure 6 shows the STA (Static Timing Analysis) results after compiler placing and routing. In the figure (a) is the setup slack, (b) is the hold slack, and all clocks meet the timing requirements. In the figure, the “e_gtxc” and “e_rxc” are the clocks of the UDP controller; The “emmc_clk_bus_o_high” and “emmc_clk_bus_o_low” are the clocks of the eMMC controller; The “pll[0]”, “pll[1]”, “pll[2]” are the clocks of other modules .

Summary (Setup)

	Clock	Slack
1	altera_reserved_tck	39.760
2	e_gtxc	2.766
3	e_rxc	0.705
4	emmc_clk_bus_o_high	14.654
5	emmc_clk_bus_o_low	9974.654
6	pll_top_inst0|altpll_component|auto_generated|pll1|clk[0]	1.390
7	pll_top_inst0|altpll_component|auto_generated|pll1|clk[1]	13.739
8	pll_top_inst0|altpll_component|auto_generated|pll1|clk[2]	4998.899

(a)

Summary (Hold)

	Clock	Slack
1	altera_reserved_tck	0.410
2	e_gtxc	1.129
3	e_rxc	0.339
4	emmc_clk_bus_o_high	15.823
5	emmc_clk_bus_o_low	15.823
6	pll_top_inst0|altpll_component|auto_generated|pll1|clk[0]	0.303
7	pll_top_inst0|altpll_component|auto_generated|pll1|clk[1]	0.257
8	pll_top_inst0|altpll_component|auto_generated|pll1|clk[2]	0.257

(b)

Fig 6. STA(a)setup slack and (b)hold slack

Figure 7 shows the bus data waveform during eMMC write operation. In the figure, ch_data is the original data before encryption, the data value is the accumulated value data for testing, ch_data_encpt is the encrypted data, emmc_dat_bus_io is the bus data written to eMMC, It is the encrypted ciphertext. The end of the least significant bit of the bus (“emmc_dat_bus_io[0]”) is the CRC status returned by the eMMC device. "0010" in the figure indicates that the bus data is correct and the data is ready to be written into the device.

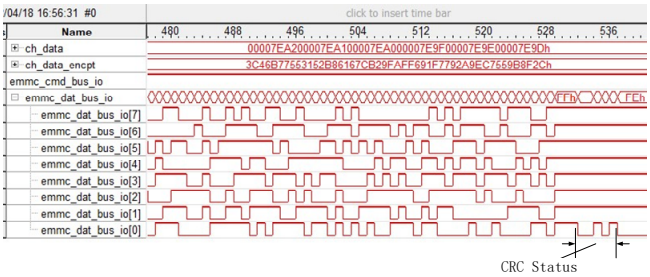

Fig 7. Bus data waveform during eMMC write operation

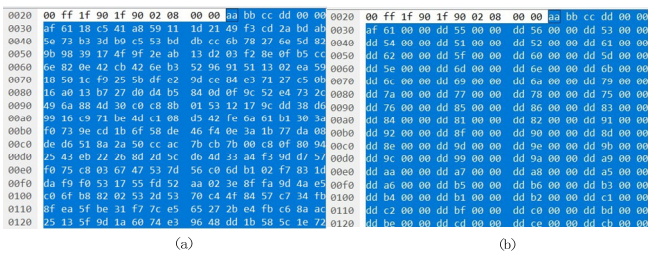

Fig 8. Get(a)incorrect data and (b)correct data via Ethernet

In Figure 8, when the wrong keys are entered (a), the incorrect storage data obtained from the UDP port in the “Wireshark” (network packet capture software). When the valid keys are entered (b), the correct storage data is obtained, and the data is the accumulated value data for testing. In the figure, 4 bytes of "AABBCCDD" are the data packet header, and the following 4 bytes are the packet number. These 8 bytes are not encrypted.

5. Conclusion

The FPGA-based embedded data encryption storage system designed in this paper has the characteristics of flexible use and low cost. It can realize reliable and secure encrypted storage of data, that is, high-speed reading, and can be used in multiple scenarios. The test results show that the system can correctly realize the encrypted storage and decrypted reading of data, which meets the design requirements.

References

[1] Zhang Wei, “Design and implementation of high-speed Ethernet interface based on FPGA,” University of Electronic Science and Technology of China, 2016.

[2] Micron, “4GB, 8GB, 16GB: e·MMC Features DATASHEET,” 2013.

[3] JEDEC Solid State Technology Association, “JESD84-B451-Embedded Multimedia Card (e•MMC) Electrical Standard,” 2012.

[4] Realtek, “Integrated 10/100/1000M Ethernet Transceiver DATASHEET,” 2013.

[5] Texa Insturments, “SN65HVD2x Extended Common-Mode RS-485 Transceivers DATASHEET,” 2016.

[6] Yang Yang, Ye Peng, Li Li, “Design and implementation of UART based on FPGA,” Electronic Measurement Technology, 2011, 34(07): 80-82.

[7] Dong Yongji, Wang Yu, Yuan Zheng, “Design and implementation of 10 Gigabit Ethernet UDP_IP hardware protocol stack based on FPGA,” Computer Application Research: 1-4 [2022-04-18].

[8] Liu Baowen, “Development of high-speed and large-capacity memory card based on eMM,” Harbin Institute of Technology, 2015.

[9] Zhang Yaojun, “Design and implementation of high-speed eMMC array controller based on FPGA,” Xidian University, 2015.

[10] Zhu Xinxin, Li Shuguo, “Implementation of high-performance 3DES algorithm based on FPGA,” Microelectronics and Computer, 2015, 32(09): 54-59.

[11] Huang Hui, Jiang Rongrong, Tan Min, Hu Xueyou, “Implementation of encryption algorithm based on FPGA,” Journal of Hefei University (Natural Science Edition), 2015,25(01):35-38.