Multi-channel Network Video Surveillance Coding System Based on GOWIN FPGAs

Cheng Yu Pan Rui Chen Hongzhao

Part I Design Overview

1.1 design purpose

The GOWIN FPGA-based multi-channel network video surveillance coding system provides efficient and reliable solutions. The system consists of a capture board, an encoding board, and a host computer. The capture board receives one to five camera inputs and converts them into digital signals. The encoding board encodes multiple videos in real-time and efficiently, utilizing the computing power of GOWIN FPGAs to reduce bandwidth and storage requirements. The host computer serves as the control center, providing a user-friendly interface and functions, receiving, and decoding video streams, and displaying, recording, and remotely accessing surveillance images. The system is flexible and scalable to meet the needs of real-time transmission, efficient encoding, and remote access. Utilizes GOWIN FPGAs to provide high-performance, low-latency video processing and transmission for safe and reliable surveillance services.

1.2 Application Areas

This multi-channel network video surveillance coding system based on GOWIN FPGAs is suitable for a wide range of application areas. First, it can be widely used in the field of security surveillance. By supporting multiple camera inputs and efficient encoding rates, the system is able to process and transmit video streams from multiple surveillance points in real-time, providing comprehensive real-time monitoring and recording functions. This is of great significance in security-sensitive places such as public places, enterprises and institutions, and residential neighborhoods. Secondly, the system can also be applied in the field of traffic monitoring. By connecting multiple cameras to the system, it can realize all-round monitoring and real-time video recording of traffic intersections, highways, and other traffic scenes. This helps the traffic management department to grasp the traffic situation in real-time and improve traffic safety and smoothness. In addition, the system can also be used in the field of remote monitoring and video conferencing. By transmitting video streams over the network, users can remotely monitor and manage the situation in multiple locations or conduct remote video conferences. This is very valuable for cross-region collaboration, distance education, telemedicine, and other fields. In short, the system has a wide range of applications in a variety of fields such as security monitoring, traffic monitoring, and remote monitoring. It can provide efficient and reliable video encoding and transmission capabilities to meet the needs of real-time monitoring, video storage, and remote access, providing users with a safe and convenient monitoring solution.

1.3 Main technical features

(1) Efficient video encoding: The system utilizes the powerful computing capability of GOWIN FPGAS to achieve high-quality, lowlatency video encoding. Advanced video coding algorithms are used to provide excellent video quality and reduce network bandwidth consumption and storage space requirements.

(2) Multiple Video Processing and Transmission: The system supports simultaneous processing and transmission of multiple video streams. The capture board is capable of receiving 1 to 5 camera inputs, and the encoding board efficiently encodes multiple videos in real time. This enables the system to meet the needs of real-time monitoring and video storage.

(3) Remote access and management: The upper computer acts as the control and management center of the system, supporting remote access and management of multiple monitoring points. Users can receive and decode the video stream transmitted by the encoding board in real-time, and display, record, and remotely access the monitoring screen. The system also supports the storage and retrieval of video data, as well as the configuration and management of the surveillance system.

1.4 Key Performance Indicators

1. Enter the camera parameters.
2. Video transmission resolution.
3. Network port transmission efficiency.

4, the upper computer operability and logic.

1.5 Key innovations

(1) Modular development and design flexibility using two FPGA development boards in cascade.

(2) Supports 1~5 camera inputs, far ahead of the traditional 1~2.

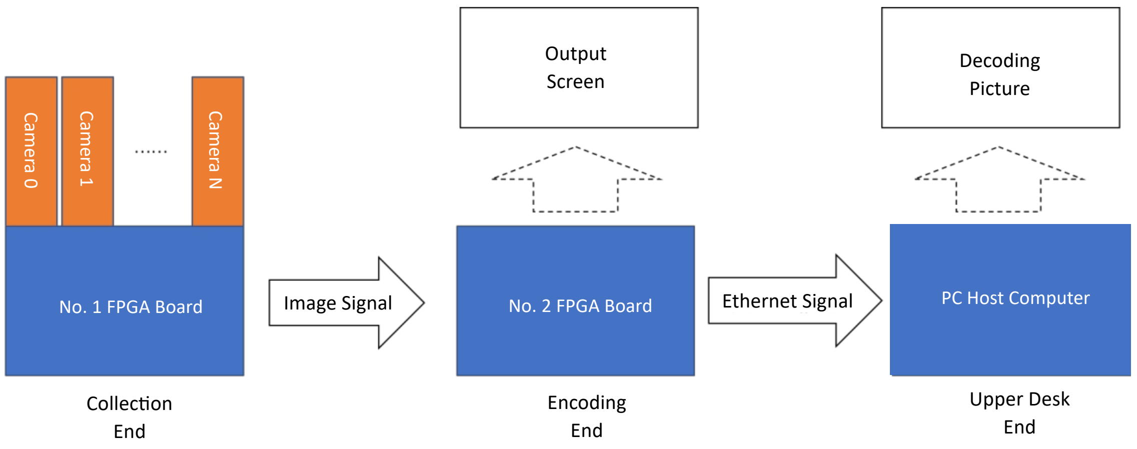
(3) The encoding side supports the original picture ring out, and can display with the upper computer each other.

(4) Development board peripheral circuits, transmission modules, and host computer completely independent design

Part II System Composition and Functional Description

2.1 Overall Introduction

The overall project structure is shown below:



Picture 1 Project structure

The project is divided into the capture side, which is responsible for capturing and fusing multiple camera frames and sending them to the encoding side with standard protocols via GOWIN HDMI TX IP or external HDMI PHY; the encoding side is responsible for receiving and decoding the standard video protocols and ringing out the display. The encoder part is responsible for receiving and decoding the standard video protocol and loop out the display. In addition, through DDR3 and FIFO cache, the encoder module packages the video into a continuous Ethernet packet stream and sends it to the host computer; the host computer part is responsible for receiving and decoding the Ethernet packet stream and outputting the current screen, and provides user-defined functions such as screenshot and screen recording.

2.2 Introduction to the modules

2.2.1 capture side

The architecture of this multi-channel network video surveillance coding system with multi-channel video buffer splicing is shown in Fig. 2, the multi-channel video signal enters the FIFO buffer through the DVP interface and initiates the burst interrupt of the corresponding video channel to the AXI4-Core control core when it is stored to one line of pixel data, and the AXI4-Core control core responds to the interrupt signals of the video channels in a sequential arbitration and writes images of various channels into the DDR3 controller by ping-ponging at the corresponding address through the AXI protocol. The AXI4-Core control core arbitrates the video interrupt signals of each channel in sequence and writes the images of each channel into the DDR3 controller through the AXI protocol according to the corresponding address ping-pong, of which the video storage address of each channel is shown in Figure 3.

Considering that GOWIN DDR3 IP does not have a direct AXI interface, we hang a conversion bridge between the original APP interface and AXI interface at the exemplary DDR3 IP interface and use this conversion bridge with the DDR3 IP to realize the whole module's control of reading and writing to DDR3.

For video splicing output, we use a self-driven video output controller to continuously read the hooked FIFO module, while the backward FIFO controller in Figure 21 carries the data of the next line from the DDR to the FIFO when it starts to read a new line, so as to realize the splicing display of the image, and its effect graph is shown in Figure 4.

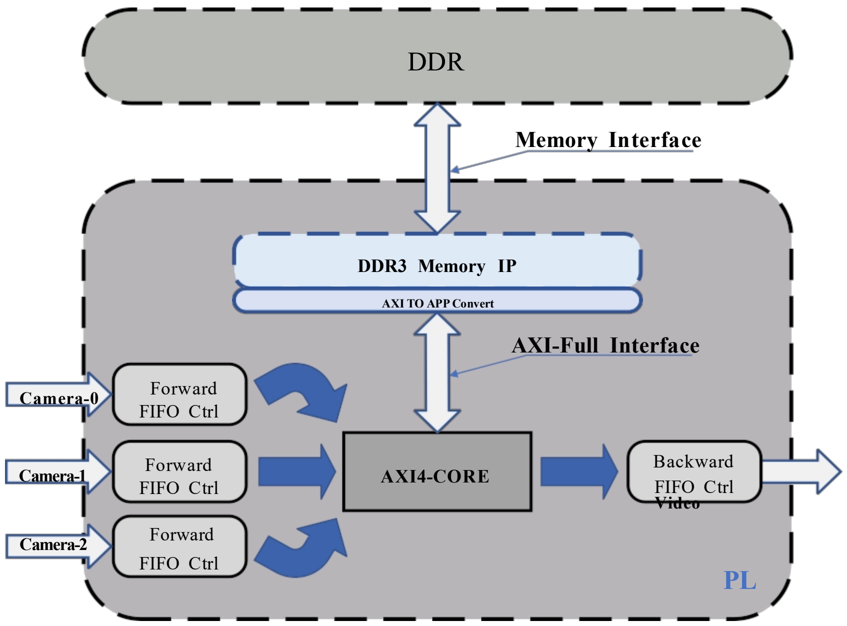


Fig. 2 Multi-channel video cache splicing architecture at the capture side

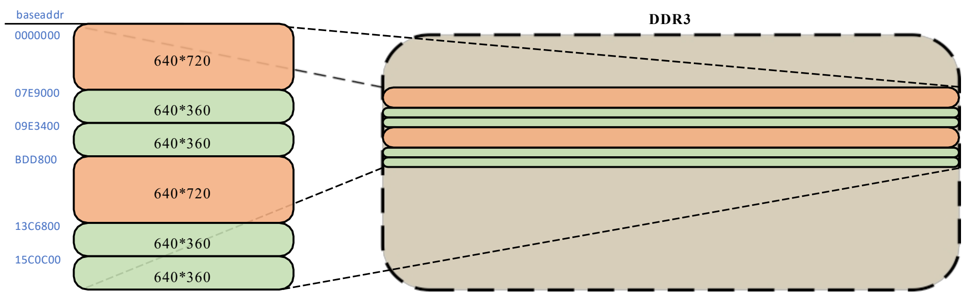


Fig. 3 Storage space diagram of multiple videos on the capture side

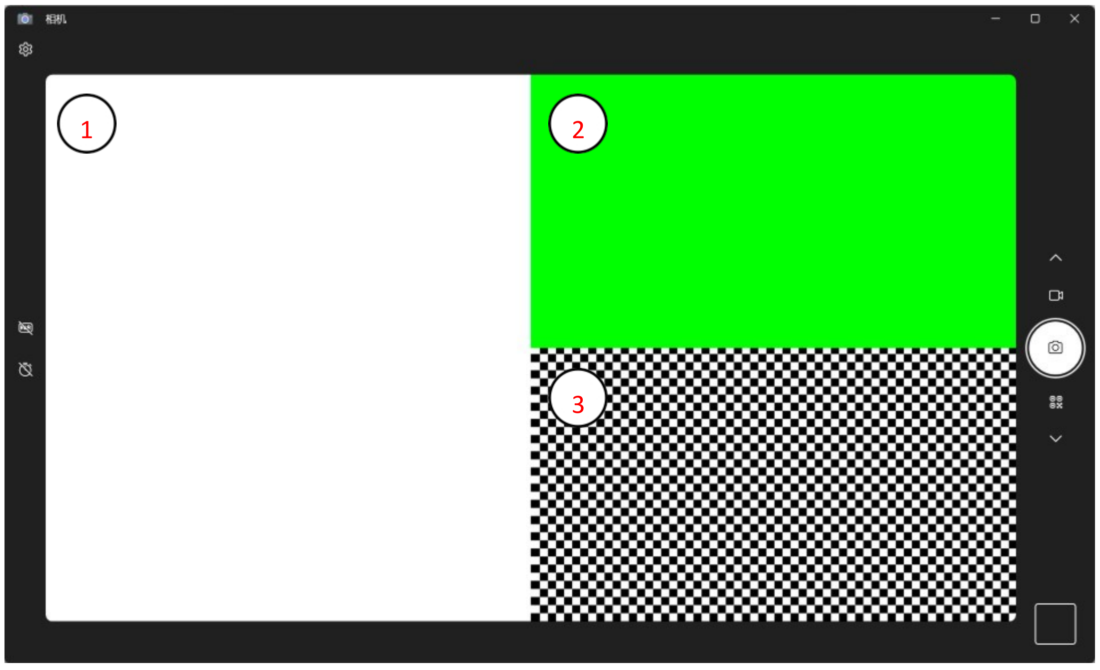


Fig. 4 Capture end output image

2.2.2 code-side

The structure of the encoding terminal is shown in Figure 5 below, which consists of decoding module, FIFO read/write module, loop out module and Ethernet encoding module. In the normal operation of the module, the HDMI decoder module firstly gives the RGB code stream corresponding to the input TMDS, and sends one way to the HDMI TX encoder module for screen loop out; the other way is sent to the FIFO read/write control module. When the number of FIFO memory meets the size of one Ethernet packet, the Ethernet encoding module will be triggered to start working, read the data in the FIFO, convert it from RGB565 to a single byte 8bit for sending, and put a number value on each sent packet for the host computer to count and analyze the network packet loss. Under this design, the maximum support for 720P60 or 1080P28 video streams is sent continuously, static timing analysis passed, the simulation diagram is shown in Figure 6 below, there is no logic error in the simulation of a frame image.

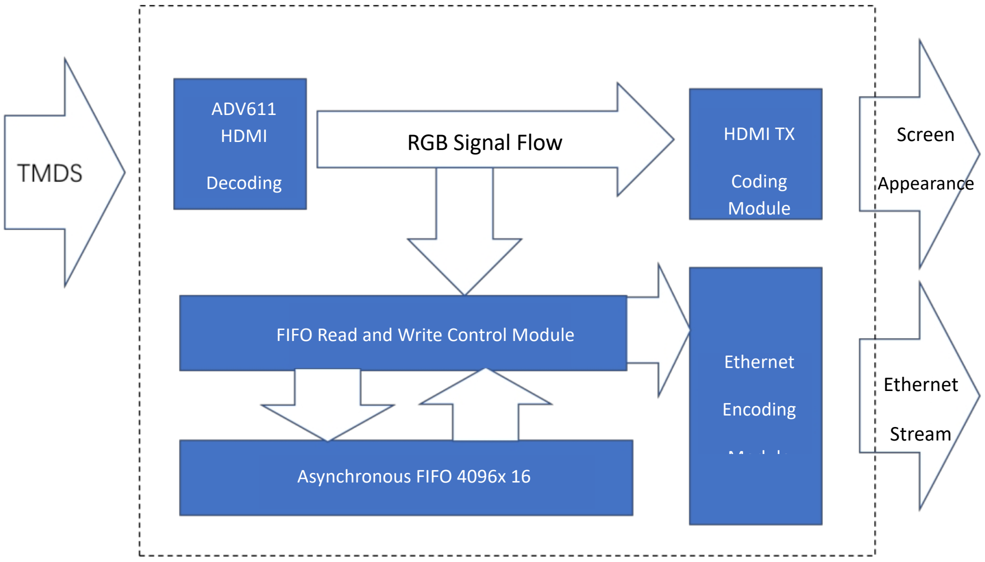


Fig. 5 Structure of the coding side

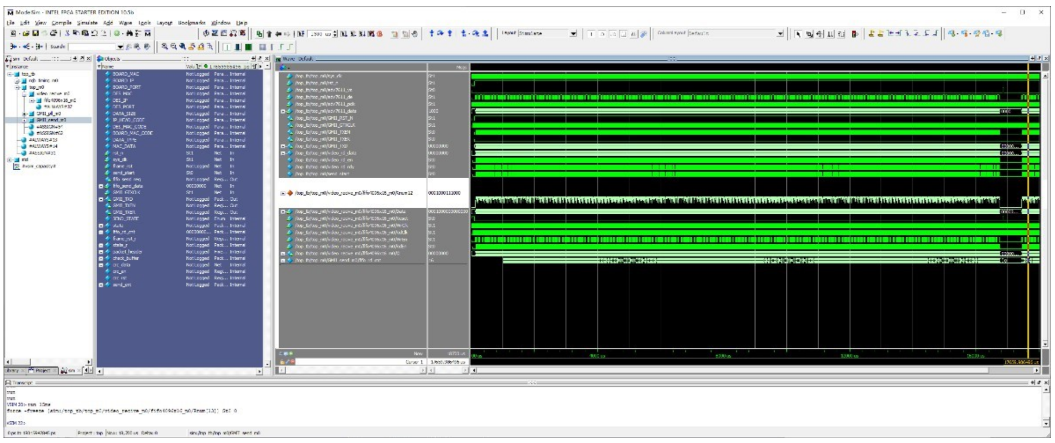


Fig. 6 Simulation of sending frame from encoding side

2.2.3 upper computer

The flowchart of the host computer program is shown in Fig. 7, which consists of three parts: the main process M, the data processing thread T1, and the user interface thread T2. The host computer adopts C++ as the programming language and is written in Visual Studio 2022 environment, which uses the built-in Socket network model and Thread multi-thread model of C++, and adopts OpenCV and CVUI third-party libraries for the design of the user interface, and the program adopts the CMake construction tool, which supports compilation and running under Windows, Linux and other platforms. The program is built with CMake, which supports compiling and running under Windows, Linux and other platforms.

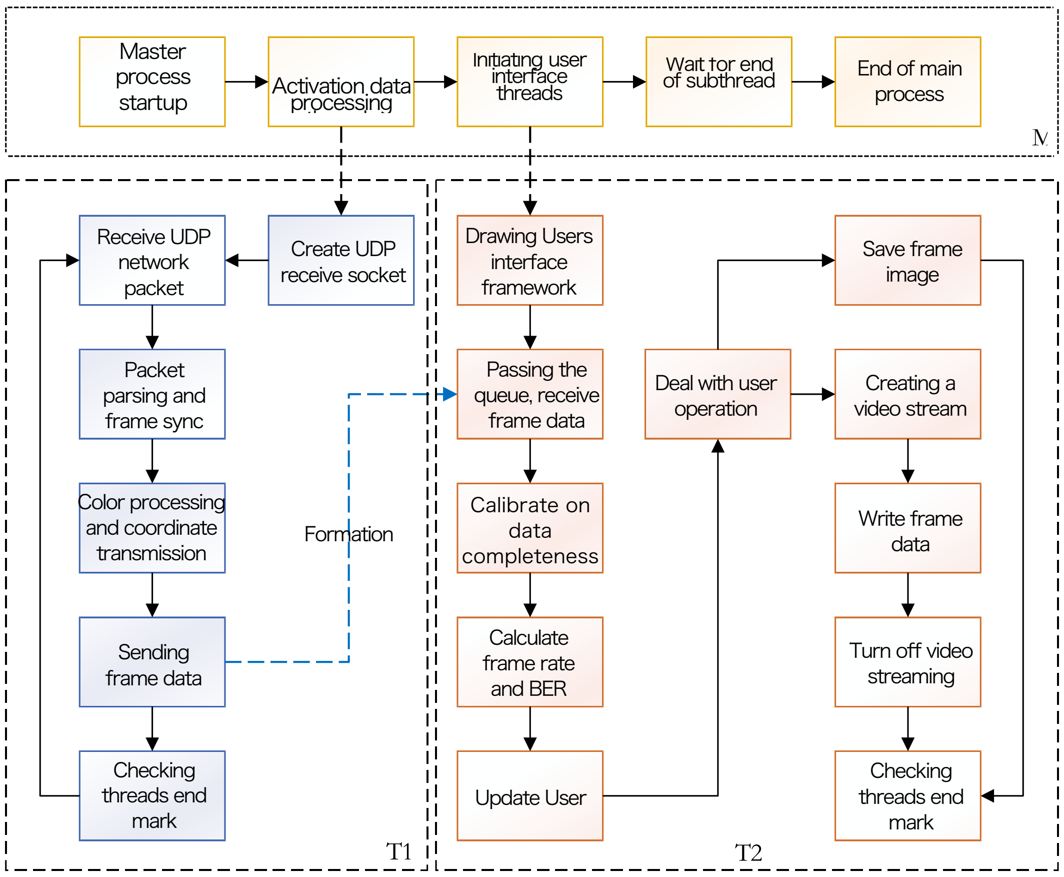

Master process startup
Activation data processing
Initiating user interface threads
Wait for end of subthread
End of main process
M
Receive UDP network packet
Create UDP receive socket
Packet parsing and frame sync
Color processing and coordinate transmission
Sending frame data
Checking threads end mark
Formation
T1
Drawing Users interface framework
Passing the queue, receive frame data
Calibrate on data completeness
Calculate frame rate and BER
Update User
Deal with user operation
Save frame image
Creating a video stream
Write frame data
Turn off video streaming
Checking threads end mark
T2

Figure 7 Upper computer program flow chart

When the host computer starts up, the main process M, as the control unit, runs first and starts two threads, the data processing thread T1 and the user interface thread T2, after initializing the basic operation environment, and waits for the threads to finish running. The main work of the data processing thread T1 is to receive the Ethernet data stream from the encoding terminal, parse the received packets, and synchronize the frame data according to the pre-coded packet index. The synchronized frame data will be written to the frame buffer after color processing and coordinate transformation, and thread T1 sends the frame data to user interface thread T2 through the queue. user interface thread T2 starts to draw the user interface frame first, and then receives the data from thread T1 through the queue. after T2 receives a frame data, it first verifies the integrity of the frame data, and if the frame data is incomplete, the frame will be dropped, and the frame error count will be increased at the same time. If the frame is incomplete, the frame is discarded and the frame error count is increased. Thread T2 then calculates the complete frame count and the packet error rate and prints the information on the user interface.T2 performs video capture and video recording operations based on button click events on the user interface. Thread T1 and Thread T2 will check the end-of-thread flag bit of the master process M after completing their work to determine whether they should continue the loop or terminate it to match the running state of the master process.

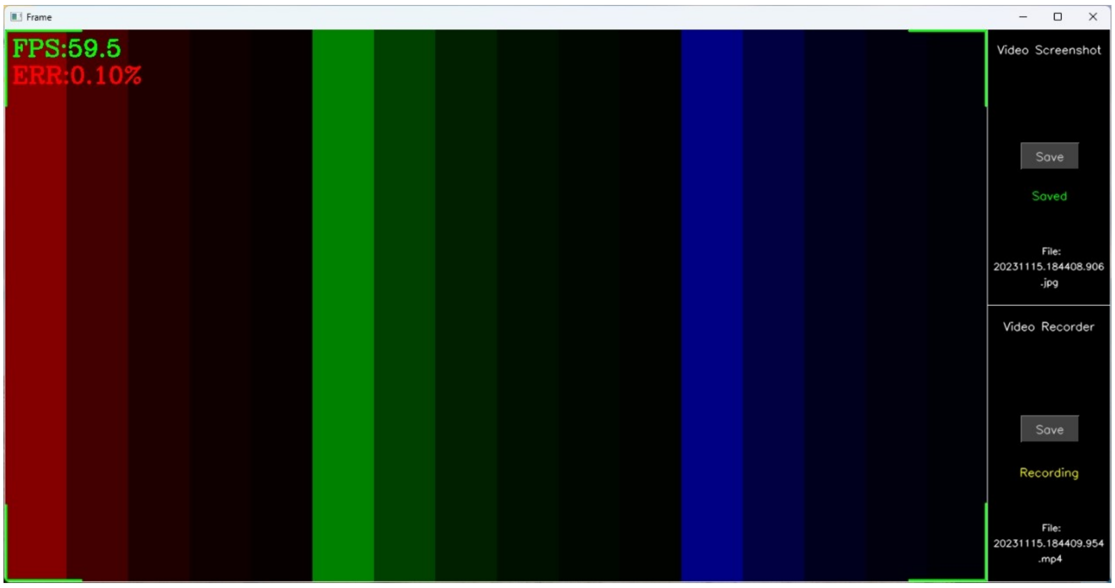


Figure 8 Upper computer user interface diagram

As shown in Figure 8, the user interface of the host computer, the left part is the video screen display area, the upper left corner is the realtime frame rate FPS and packet error rate ERR display, and the right part is the user operation area, which is the video screenshot and video recording buttons, respectively. After the user clicks the video capture button, the host computer encodes the raw frame data corresponding to the current display screen into a JPG image file and saves it. After the user clicks the Video Record button, the status area shows that the video is being recorded and the original video frame is encoded into an MP4 video file and saved until the user clicks the Stop Recording button. The screenshot and video recording files are named with the time stamp of the start of the operation, accurate to 1 millisecond. The host computer supports simultaneous screenshot and video recording, which can quickly capture key video frames.

Part III Completion and performance parameters

1、 Input camera parameters: 1~5 way DVP OV5640 (1 main 0 vice, 1 main 2 vice, 1 main 4 vice).

2、 Video transmission resolution: 1280*720@60hz in line with standard video timing.

3. Network port transmission protocol: GMII 1000M transmission based on RTL8211 (actual 900+M).
4. The upper computer supports screenshot and recording.

Part IV Summary

4.1 Scalability

Capture side: improve the versatility of the access camera, the bandwidth of the output screen
Encoding side: to improve the versatility of the access screen, Ethernet sending efficiency, video encoding compression rate
host computer: to improve the logic and operability of the user's operation

4.2 experience

In the process of developing a multi-channel network video surveillance coding system based on GOWIN FPGAs, we deeply realized the importance and wide application of video coding technology. By fully utilizing the computing power of GOWIN FPGAs, we have achieved efficient video encoding and transmission to meet the needs of real-time monitoring, video storage and remote access. The system has a wide range of applications in the fields of security, transportation and remote monitoring. We also recognize the importance of flexibility and scalability in the system design so that it can be adapted to different scenarios and needs. All in all, this project has given us an in-depth understanding of the practical application and value of video coding technology, and has contributed to the provision of secure and reliable surveillance solutions.

Part V References

[1]. LI Yong,WANG Lei,QIAN Hanlin. Design and realization of panoramic video image stitching based on FPGA[J]. Electronic Design Engineering,2018,26(2):80-83. DOI:10.3969/j.issn.1674-6236.2018.02.018.

[2]. Wan Quan,Li Shaofu. F P G A - b a s e d image overlay and crossscreen splicing fusion method[J]. Liquid Crystal and Display,2020,35(10):1066-1072. DOI:10.37188/YJYXS20203510.1066. [3]. YANG Lei, REN Long, LIU Qing, et al. Research and realization of real-time splicing technology for large field of view images based on FPGA[J]. Infrared and Laser Engineering,2015(6):1929-1935. DOI:10.3969/j.issn.1007-2276.2015.06.045.

[4]. ZHENG R,XIAO SHUNWEN,WANG YONG. Design of F P G A - b a s e d

Gigabit Ethernet real-time image acquisition and transmission system [J]. Journal of West China Normal University (Natural Science Edition) ,2022,43(3):349-354. DOI:10.16246/j.issn.1673-5072.2022.03.017.

[5]. ZHENG R,XIAO SHUNWEN,WANG YONG. Design of F P G A - b a s e d Gigabit Ethernet real-time image acquisition and transmission system [J]. Journal of West China Normal University (Natural Science Edition) ,2022,43(3):349-354. DOI:10.16246/j.issn.1673-5072.2022.03.017.

[6]. Yu P.W., Ren Y., Feng P., et al. Design of Gigabit Ethernet CMOS Image Data Transmission System Based on FPGAs [J]. J]. Foreign Electricity Electronic Measurement Measurement Technology Techniques ,2016,35(11):76-81. DOI:10.3969/j.issn.1002-8978.2016.11.018.

[7]. Gao Shangshang,Wang Xinyu,Wang Xiaoya,et al. Design and FPGA Implementation of Image Processing System Based on Ethernet Transmission [J]. Computing Computer Computer Measurement Measurement and Control control ,2022,30(7):213-218. DOI:10.16526/j.cnki.11-4762/tp.2022.07.032.

Part VI Appendix

6.1 Top-level code for the collection side

```
          input                          clk
,       input                          rst_n
,   output             cmos_scl
,   inout              cmos_sda
,   input              cmos_pclk
,   input              cmos_vsync
,   input              cmos_href
,   input    [7:0]    cmos_db
,   output             cmos_rst_n
,   output             cmos_pwdn

,   output             cmos1_scl
,   inout              cmos1_sda
,   input              cmos1_pclk
,   input              cmos1_vsync
,   input              cmos1_href
,   input    [7:0]    cmos1_db
,   output             cmos1_rst_n
,   output             cmos1_pwdn

,   output             cmos2_scl
,   inout              cmos2_sda
,   input              cmos2_pclk
,   input              cmos2_vsync
,   input              cmos2_href
,   input    [7:0]    cmos2_db
,   output             cmos2_rst_n
,   output             cmos2_pwdn

,   output [14-1:0]   ddr_addr
,   output [3-1:0]    ddr_bank
,   output             ddr_cs
,   output             ddr_ras
,   output             ddr_cas
,   output             ddr_we
,   output             ddr_ck
,   output             ddr_ck_n
,   output             ddr_cke
,   output             ddr_odt
,   output             ddr_reset_n
,   output [2-1:0]    ddr_dm
,   inout [16-1:0]    ddr_dq
,   inout [2-1:0]     ddr_dqs
,   inout [2-1:0]     ddr_dqs_n

,   output             hdmi_clk
```

```
	,	output[23:0]		hdmi_d
	,	output			hdmi_de
	,	output			hdmi_hs
	,	output			hdmi_vs
	,	output			hdmi_nreset
	,	inout			hdmi_scl
	,	inout			hdmi_sda
);

//memory interface
wire                         memory_clk          ;
wire                         dma_clk            ;
wire                         DDR_pll_lock        ;
wire                         cmd_ready           ;
wire[2:0]                    cmd                ;
wire                         cmd_en              ;
wire[5:0]                    app_burst_number   ;
wire[ADDR_WIDTH-1:0]      addr                 ;
wire                         wr_data_rdy        ;
wire                         wr_data_en          ;
wire                         wr_data_end         ;
wire[DATA_WIDTH-1:0]      wr_data              ;
wire[DATA_WIDTH/8-1:0] wr_data_mask          ;
wire                         rd_data_valid      ;
wire                         rd_data_end        ;
wire[DATA_WIDTH-1:0]      rd_data              ;
wire                         init_calib_complete   ;

//According to IP parameters to choose
`define      WR_VIDEO_WIDTH_16
`define  DEF_WR_VIDEO_WIDTH 16

`define      RD_VIDEO_WIDTH_16
`define  DEF_RD_VIDEO_WIDTH 16

`define  USE_THREE_FRAME_BUFFER

`define  DEF_ADDR_WIDTH 28
`define  DEF_SRAM_DATA_WIDTH 128
//
//=========================================================
//SRAM parameters
parameter  ADDR_WIDTH              = `DEF_ADDR_WIDTH;      //存储单元是
byte ， 总 容 量 =2^27*16bit = 2Gbit, 增 加 1 位 rank 地 址 ，
```

```
{rank[0],bank[2:0],row[13:0],cloumn[9:0]}
parameter  DATA_WIDTH                    = `DEF_SRAM_DATA_WIDTH;     //与生成
DDR3IP 有关，此 ddr3 2Gbit, x16，   时钟比例 1:4  ，  则固定 128bit
parameter WR_VIDEO_WIDTH              = `DEF_WR_VIDEO_WIDTH;
parameter RD_VIDEO_WIDTH              = `DEF_RD_VIDEO_WIDTH;

//-------------------
//syn_code
wire                               syn_off0_re;    // ofifo read enable signal
wire                               syn_off0_vs;
wire                               syn_off0_hs;

wire                               off0_syn_de    ;
wire [RD_VIDEO_WIDTH-1:0]          off0_syn_data;

wire[15:0]                        cmos_16bit_data;
wire                               cmos_16bit_clk;

wire[9:0]                         lut_index;
wire[31:0]                        lut_data;

wire                               cmos_frame_clk    ;
wire                               cmos_frame_vsync;
wire                               cmos_frame_href ;
wire                               cmos_frame_de      ;
wire        [23:0]                cmos_frame_data ;

wire                               cmos1_frame_clk    ;
wire                               cmos1_frame_vsync;
wire                               cmos1_frame_href ;
wire                               cmos1_frame_de      ;
wire        [23:0]                cmos1_frame_data ;

wire                               cmos2_frame_clk    ;
wire                               cmos2_frame_vsync;
wire                               cmos2_frame_href ;
wire                               cmos2_frame_de      ;
wire        [23:0]                cmos2_frame_data ;

wire                               video_clk            ;
wire                               video_vsync          ;
wire                               video_href           ;
wire                               video_de             ;
wire        [23:0]                video_data           ;
wire                                    ila_clk             ;


// //generate the CMOS sensor clock and the SDRAM controller clock
// cmos_pll cmos_pll_m0(
//     .clkin                          (clk                        ),
//     .clkout                         (cmos_clk                     )
// );

mem_pll mem_pll_m0(
        .clkin                          (clk                        ),
        .clkout                         (memory_clk             ),
         .clkoutd                        (ila_clk                   ),
        .lock                         (DDR_pll_lock             )
);

Gowin_rPLL_lcd u_Gowin_rPLL_lcd(
        .clkout                         (video_clk                 ), //output clkout
        .lock                            (                            ), //output lock
        .clkin                          (clk                        ) //input clkin
);

//IIC  延时约 1s 复位
reg [31:0] clk_delay = 0;
wire iic_rst = clk_delay != 65_000_000;
always@(posedge video_clk, negedge rst_n) begin
        if (!rst_n) begin
                clk_delay = 0;
        end
        else begin
                clk_delay <=( clk_delay == 65_000_000)? clk_delay : clk_delay + 1;
        end
end


iic_ctrl#(
        .CLK_FRE                        (27                     ),
        .IIC_FRE                      (100                    ),
        .IIC_SLAVE_REG_EX              (1                      ),
        .IIC_SLAVE_ADDR_EX            (0                      ),
        .IIC_SLAVE_ADDR               (16'h78               ),
        .INIT_CMD_NUM                  (303                    ),
        .PIC_PATH                      ("init_640x720.txt")
```

```
)iic_ctrl_m0(
        .clk                          (clk                  ),
        .rst_n                        (~iic_rst            ),
        .iic_scl                      (cmos_scl            ),
        .iic_sda                      (cmos_sda            )
);

iic_ctrl#(
        .CLK_FRE                      (27                   ),
        .IIC_FRE                      (100                  ),
        .IIC_SLAVE_REG_EX             (1                    ),
        .IIC_SLAVE_ADDR_EX            (0                    ),
        .IIC_SLAVE_ADDR               (16'h78               ),
        .INIT_CMD_NUM                 (303                  ),
        .PIC_PATH                     ("init_640x360.txt")
)iic_ctrl_m1(
        .clk                          (clk                  ),
        .rst_n                        (~iic_rst            ),
        .iic_scl                      (cmos1_scl           ),
        .iic_sda                      (cmos1_sda           )
);

iic_ctrl#(
        .CLK_FRE                      (27                   ),
        .IIC_FRE                      (100                  ),
        .IIC_SLAVE_REG_EX             (1                    ),
        .IIC_SLAVE_ADDR_EX            (0                    ),
        .IIC_SLAVE_ADDR               (16'h78               ),
        .INIT_CMD_NUM                 (303                  ),
        .PIC_PATH                     ("init_640x360.txt")
)iic_ctrl_m2(
        .clk                          (clk                  ),
        .rst_n                        (~iic_rst            ),
        .iic_scl                      (cmos2_scl           ),
        .iic_sda                      (cmos2_sda           )
);

ov5640_capture_data    u_ov5640_capture_data(
        .rst_n                             (rst_n                    ),

        .cam_pclk                          (cmos_pclk               ),
        .cam_vsync                         (cmos_vsync              ),
        .cam_href                          (cmos_href               ),
        .cam_data                          (cmos_db                 ),
        .cam_rst_n                        (cmos_rst_n              ),
        .cam_pwdn                           (cmos_pwdn               ),

        .cmos_frame_clk                   (cmos_frame_clk         ),
       .cmos_frame_vsync                  (cmos_frame_vsync       ),
        .cmos_frame_href                  (cmos_frame_href       ),
        .cmos_frame_de                     (cmos_frame_de          ),
        .cmos_frame_data                  (cmos_frame_data        )
);

ov5640_capture_data    u_ov5640_capture_data1(
        .rst_n                            (rst_n                   ),

        .cam_pclk                          (cmos1_pclk              ),
        .cam_vsync                          (cmos1_vsync             ),
        .cam_href                          (cmos1_href              ),
        .cam_data                          (cmos1_db                ),
        .cam_rst_n                        (cmos1_rst_n             ),
        .cam_pwdn                           (cmos1_pwdn              ),

        .cmos_frame_clk                   (cmos1_frame_clk        ),
       .cmos_frame_vsync                  (cmos1_frame_vsync    ),
        .cmos_frame_href                  (cmos1_frame_href      ),
        .cmos_frame_de                     (cmos1_frame_de         ),
        .cmos_frame_data                  (cmos1_frame_data      )
);

ov5640_capture_data    u_ov5640_capture_data2(
        .rst_n                            (rst_n                   ),

        .cam_pclk                          (cmos2_pclk              ),
        .cam_vsync                          (cmos2_vsync             ),
        .cam_href                          (cmos2_href              ),
        .cam_data                          (cmos2_db                ),
        .cam_rst_n                        (cmos2_rst_n             ),
        .cam_pwdn                           (cmos2_pwdn              ),

        .cmos_frame_clk                   (cmos2_frame_clk        ),
       .cmos_frame_vsync                  (cmos2_frame_vsync    ),
        .cmos_frame_href                  (cmos2_frame_href      ),
        .cmos_frame_de                     (cmos2_frame_de         ),
        .cmos_frame_data                  (cmos2_frame_data      )
);
```

```
video_stiching_top u_video_stiching_top(
//--------------------------------------------------
// Cmos port
        .cmos0_clk              (cmos_frame_clk       )
    ,   .cmos0_vsync            (cmos_frame_vsync     )
    ,   .cmos0_href             (cmos_frame_href      )
    ,   .cmos0_clken            (cmos_frame_de        )
    ,   .cmos0_data
    ({cmos_frame_data[7-:8],cmos_frame_data[15-:8],cmos_frame_data[23-:8]})

    ,   .cmos1_clk              (cmos1_frame_clk      )
    ,   .cmos1_vsync            (cmos1_frame_vsync    )
    ,   .cmos1_href             (cmos1_frame_href     )
    ,   .cmos1_clken            (cmos1_frame_de       )
    ,   .cmos1_data
    ({cmos1_frame_data[7-:8],cmos1_frame_data[15-:8],cmos1_frame_data[23-:8]})

    ,   .cmos2_clk              (cmos2_frame_clk      )
    ,   .cmos2_vsync            (cmos2_frame_vsync    )
    ,   .cmos2_href             (cmos2_frame_href     )
    ,   .cmos2_clken            (cmos2_frame_de       )
    ,   .cmos2_data
    ({cmos2_frame_data[7-:8],cmos2_frame_data[15-:8],cmos2_frame_data[23-:8]})

//--------------------------------------------------
// Video port
    ,   .video_clk                (video_clk            )
    ,   .video_vsync              (video_vsync          )
    ,   .video_href               (video_href           )
    ,   .video_de                 (video_de             )
    ,   .video_data               (video_data           )

//--------------------------------------------------
// DDR native port
    ,   .ref_clk                  (clk                  )
    ,   .sys_rst_n                (rst_n                )
    ,   .init_calib_complete    (init_calib_complete)
    ,   .c0_sys_clk               (memory_clk           )
    ,   .c0_sys_clk_locked      (DDR_pll_lock         )
    ,   .ddr_addr                 (ddr_addr             )
    ,   .ddr_bank                 (ddr_bank             )
    ,   .ddr_cs                   (ddr_cs               )
    ,   .ddr_ras                  (ddr_ras              )
    ,   .ddr_cas                  (ddr_cas              )
```

```
        ,     .ddr_we                           (ddr_we                 )
        ,     .ddr_ck                           (ddr_ck                 )
        ,     .ddr_ck_n                         (ddr_ck_n               )
        ,     .ddr_cke                          (ddr_cke                )
        ,     .ddr_odt                          (ddr_odt                )
        ,     .ddr_reset_n                      (ddr_reset_n            )
        ,     .ddr_dm                           (ddr_dm                 )
        ,     .ddr_dq                           (ddr_dq                 )
        ,     .ddr_dqs                          (ddr_dqs                )
        ,     .ddr_dqs_n                        (ddr_dqs_n              )
);

//--------------------------------------------------
//Sil9134

wire[9:0]                   lut_index;
wire[31:0]                  lut_data;
wire                        clk_100mhz;
wire                        pll_iic_locked;


Gowin_rPLL_iic u_Gowin_rPLL_iic(
        .clkout         (clk_100mhz), //output clkout
        .lock           (pll_iic_locked), //output lock
        .clkin          (clk) //input clkin
);

//I2C master controller
i2c_config i2c_config_m0(
        .rst                            (~pll_iic_locked              ),
        .clk                            (clk_100mhz                   ),
        .clk_div_cnt                    (16'd499                      ),
        .i2c_addr_2byte                 (1'b0                         ),
        .lut_index                      (lut_index                    ),
        .lut_dev_addr                   (lut_data[31:24]              ),
        .lut_reg_addr                   (lut_data[23:8]               ),
        .lut_reg_data                   (lut_data[7:0]                ),
        .error                          (                             ),
        .done                           (                             ),
        .i2c_scl                        (hdmi_scl                     ),
        .i2c_sda                        (hdmi_sda                     )
);
//configure look-up table
lut_si9134 lut_si9134_m0(
    .lut_index                    (lut_index              ),
    .lut_data                     (lut_data               )
);


assign  hdmi_clk        =    video_clk          ;
assign  hdmi_vs         =    video_vsync        ;
assign  hdmi_hs         =    video_href         ;
assign  hdmi_de         =    video_de           ;
assign  hdmi_d          =    video_data         ;
assign  hdmi_nreset     =   pll_iic_locked   ;


endmodule
```

6.2 编码端顶层代码：

```
module top(
    input               sys_clk,
    input               rst_n,

    // 视频源输入
     output               adv7611_rst_n,
     inout                adv7611_sda,
     inout                adv7611_scl,
     input                adv7611_hs,
     input                adv7611_vs,
     input                adv7611_de,
     input                adv7611_pclk,
     input    [15:0]     adv7611_data,

     //环出模块
     output             hdmi_clk_p  ,hdmi_clk_n ,
    output    [2:0]     hdmi_data_p,hdmi_data_n,

    // 串口监测端口
    output              ws2812_io,

    // GMII 输出
    output              GMII_RST_N,
    output              GMII_GTXCLK,
    output              GMII_TXEN,
    output              GMII_TXER,
    output    [7:0]     GMII_TXD
    );
```

```
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
///////////////////                  接收模块                  ///////////////////////////
////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
assign adv7611_rst_n = rst_n;

wire[9:0]       adv7611_lut_index;
wire[31:0]      adv7611_lut_data;
wire            cfg_done,cfg_error;
i2c_config i2c_config_m0(
   .rst_n                   (rst_n                        ),
   .clk                     (sys_clk                      ),
   .clk_div_cnt             (16'd270                      ),
   .i2c_addr_2byte          (1'b0                         ),
   .lut_index               (adv7611_lut_index            ),
   .lut_dev_addr            (adv7611_lut_data[31:24]      ),
   .lut_reg_addr            (adv7611_lut_data[23:8]       ),
   .lut_reg_data            (adv7611_lut_data[7:0]        ),
   .error                   (cfg_error                    ),
   .done                    (cfg_done                     ),
   .i2c_scl                 (adv7611_scl                  ),
   .i2c_sda                 (adv7611_sda                  )
);

lut_adv7611 lut_adv7611_m0(
   .lut_index               (adv7611_lut_index            ),
   .lut_data                (adv7611_lut_data             )
);


wire [ 7:0] video_rd_data;
wire        video_rd_en;

reg              test_de;
reg      [15:0]  test_data;

always@(posedge adv7611_pclk)begin
      test_de <= adv7611_de;
      test_data <= adv7611_data;
end

video_recive video_recive_m0(
      .Reset               (adv7611_vs          ),
	.video_clk		(adv7611_pclk		),
	.video_de		(test_de	),
	.video_data
	({test_data[13:11],test_data[15:14],test_data[6:5],test_data[10:7],test_data[2:0],te
st_data[4:3]}),

	.video_rd_clk		(GMII_GTXCLK		),
	.video_rd_rdy		(video_rd_rdy		),
	.video_rd_en		(video_rd_en		),
	.video_rd_data		(video_rd_data		)
	);

//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
///////////////////			压缩模块			////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
//未使用

//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
///////////////////			发送模块			////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////////
parameter DATA_SIZE = 16'D1442; // 2 字节编号 + 1440 字节数据
assign GMII_RST_N = rst_n;

GMII_pll GMII_pll_m0(
	.clkin		(sys_clk		),
	.clkout		(GMII_GTXCLK	) // 125MHz
	);

wire send_start;
assign send_start	= video_rd_rdy; // video_rd_num >= 1440;//(DATA_SIZE - 2);
//当存满的数据足够一次发送，就开始发送(-2 编号字节)
GMII_send #(
	.BOARD_MAC	(48'h00_11_22_33_44_55			),//开发板 MAC 地
址
	.BOARD_IP	({8'd192,8'd168,8'd2,8'd123}	),//开发板 IP 地址
	.BOARD_PORT	(16'd8000				),
	.DES_MAC	(48'hff_ff_ff_ff_ff_ff			),//目的 MAC 地址
	.DES_IP		({8'd192,8'd168,8'd2,8'd102}	),//目的 IP 地址
	.DES_PORT	(16'd8001				), //DES_PORT
```

```
    .DATA_SIZE    (DATA_SIZE                  )  // 数 据 包 长 度
50~1500 B
    )GMII_send_m0(
    .rst_n            (rst_n             ),

    .sys_clk          (sys_clk           ),
    .frame_rst        (adv7611_vs        ),

    .send_start       (send_start       ),
    .fifo_send_req        (video_rd_en        ),
    .fifo_send_data   (video_rd_data        ),

    .GMII_GTXCLK          (GMII_GTXCLK        ),
    .GMII_TXD             (GMII_TXD           ),
    .GMII_TXEN            (GMII_TXEN          ),
    .GMII_TXER            (GMII_TXER          )
    );

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//////////////////                  环出模块                 ///////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
DVI_TX_Top your_instance_name(
    .I_rst_n          (rst_n                         ), //input I_rst_n
    .I_rgb_clk            (adv7611_pclk                   ), //input I_rgb_clk
    .I_rgb_vs             (adv7611_vs                     ), //input I_rgb_vs
    .I_rgb_hs             (adv7611_hs                     ), //input I_rgb_hs
    .I_rgb_de             (adv7611_de                     ), //input I_rgb_de
    .I_rgb_r          ({adv7611_data[15:11],3'd0}  ), //input [7:0] I_rgb_r
    .I_rgb_g          ({adv7611_data[10: 5],2'd0}  ), //input [7:0] I_rgb_g
    .I_rgb_b          ({adv7611_data[ 4: 0],3'd0}  ), //input [7:0] I_rgb_b
    .O_tmds_clk_p         (hdmi_clk_p                     ), //output O_tmds_clk_p
    .O_tmds_clk_n         (hdmi_clk_n                     ), //output O_tmds_clk_n
    .O_tmds_data_p        (hdmi_data_p                    ),        //output        [2:0]
O_tmds_data_p
    .O_tmds_data_n        (hdmi_data_n                    )         //output        [2:0]
O_tmds_data_n
);

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//////////////////                  监测模块                 ///////////////////////////
////////////////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////////////////////

ws2812_top ws2812_top_m0(
        .clk            (sys_clk                                ),
        .key            ({cfg_done,cfg_error}                   ),
        .WS2812_Di      (ws2812_io                              )
);

endmodule
```