Z7-Lite Series

Tutorial : **HLS**

Xilinx ZYNQ-7000 SoC

Rev. 1.0

Article I Version Record

Version	time	describe
Rev. 1.0	2020.7.28	First version released

Article **II** Copyright Notice:

XME/IME series core modules, evaluation boards and related intellectual property rights are owned by Microphase Technology.

Limited company owned.

Article **III** Disclaimer:

Microphase Technology does not guarantee the source code, software, documents, etc. provided in the product.

any warranties of any kind, whether express or implied, including but not limited to fitness for a particular purpose.

There is no warranty for a particular purpose; all risks are borne by the user.

Chapter 1 Introduction to **HLS**

1.1 Introduction to HLS

Vivado HLS (High Level Synthesis) is a high-level synthesis tool provided by Xilinx to developers.

Using HLS can speed up our development efficiency. Developers can use C/C++ or

It can convert high-level languages such as System C into RTL circuits, which is very useful when dealing with some more complex algorithms.

It can provide great convenience for developers.

Vivado HLS provides developers with common libraries, such as OpenCV libraries related to image processing and other

Its mathematical library allows developers to speed up development by calling functions in these libraries.

Vivado's design is centered on IP, and HLS is equivalent to a tool that can be used to create IP.

IP created through HLS can be applied to Vivado by instantiating or using BlockDesign.

You can use HLS to apply the IP created by HLS to your own projects.

Xilinx also provides us with HLS tutorial documents. If you are interested, you can download them yourself.

These documents are also provided in our development kit. Users can download UG902 from the Xilinx official website.

UG871ÿUG1233ÿ

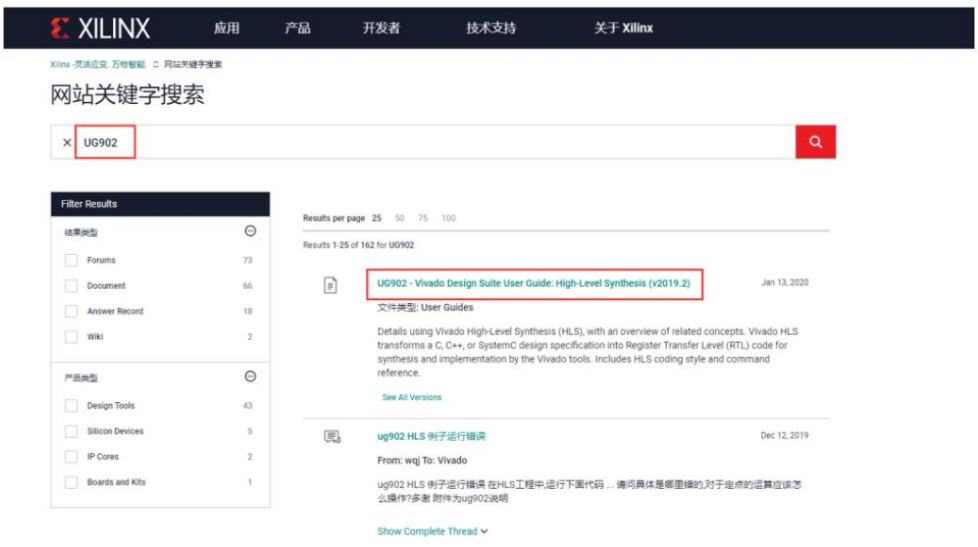


Figure 1- 1

Here we provide the download link of HLS2016.1 version, Xilinx's latest UG902 version

The HLS video related chapters have been moved to a new document, which is more complete than the 2016.1 version.

The 2016 version of the user guide contains more information about HLS graphs.

It is a more detailed description of the processing function, and will not be too specific to the version of Vivado2018.3 we use.

Big impact.

1) UG902

https://china.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug902-vivado-high-level-synthesis.pdf

The latest version of the reference document UG1233 on HLS image processing is linked below

2) UG1233

https://china.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1233-xilinx-opencv-user-guide.pdf

The link to the reference document UG871 on how to use Vivado HLS is as follows

3) UG871

https://china.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug871-vivado-high-level-synthesis-tutorial.pdf

1.2 Introduction to Vivado HLS Development Process

The process of developing with HLS is shown in the following figure:

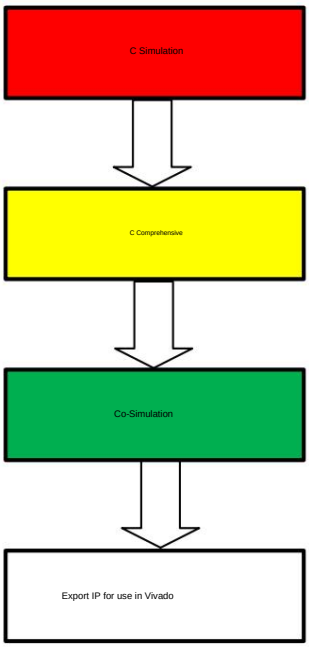


Figure 1- 2

First, you need to perform C/C++ simulation. Anyone who has done FPGA development knows that in FPGA development During the development of HLS, we will use Testbench to verify the functions of the modules we designed. Simulation is also required. In the C simulation stage, we need to add a top-level function in the source file. This function is what we want to map to the RTL circuit in the future. With this function, we need We need a C Testbench to verify the function of this function and check our function at the algorithm level. Whether it can work properly. Algorithm-level simulation can quickly produce results, which helps improve our development efficiency.

Next we need to synthesize the C code, which will generate In the C synthesis stage, HLS will generate the corresponding circuit according to some constraints we have in the function. We will talk about different interfaces and constraints in the future. In HLS, these constraints are called Directives.

After C synthesis is completed, C/RTL joint simulation is required. At this stage, HLS will We use our C Testbench to generate our RTL Testbench and use the simulation tool we choose to After the simulation is completed, we can observe the waveform generated by the joint simulation. If the RTL simulation results are correct, you can proceed to the next step.

As mentioned earlier, HLS is equivalent to an IP generator, which can convert our high-level language code

Mapped as an IP, we can export these IPs to the Vivado integrated development environment as needed.

Apply the IP of these algorithms to actual projects.

The above is an overview of the entire development process of HLS development, which also includes the function interface and optimization.

The following chapters will introduce the content of the flashing LED.

The entire HLS development process.

Chapter 2 HLS Development Process Experience

2.1 Use **HLS** to complete **LED** light flashing

Just like in software development, learning a new language and a new development environment always requires writing some Just like a program that outputs Hello world!, in FPGA development, adapting to a new development environment is also a challenge. It is inevitable to start with controlling the LED lights. Now let's start creating this project.

2.1.1 Create a new **HLS** project

After installing the Vivado suite, HLS has been automatically installed. We only need to open Vivado HLS. Yes, as shown.

Error! Reference source not found.

Figure 2-1

After opening HLS, as shown in the figure below, click Create New Project to create a new project

Figure 2- 2

Next, enter the name of the project, select the save path for the project, and click Next;

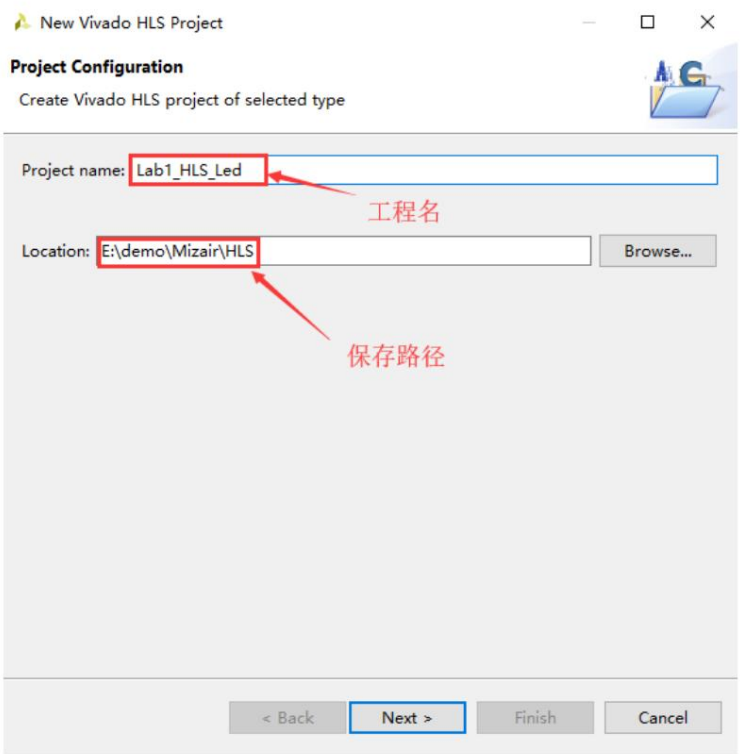


Figure 2- 3

Next, select the defined function. We will not add it here yet, just click Next;

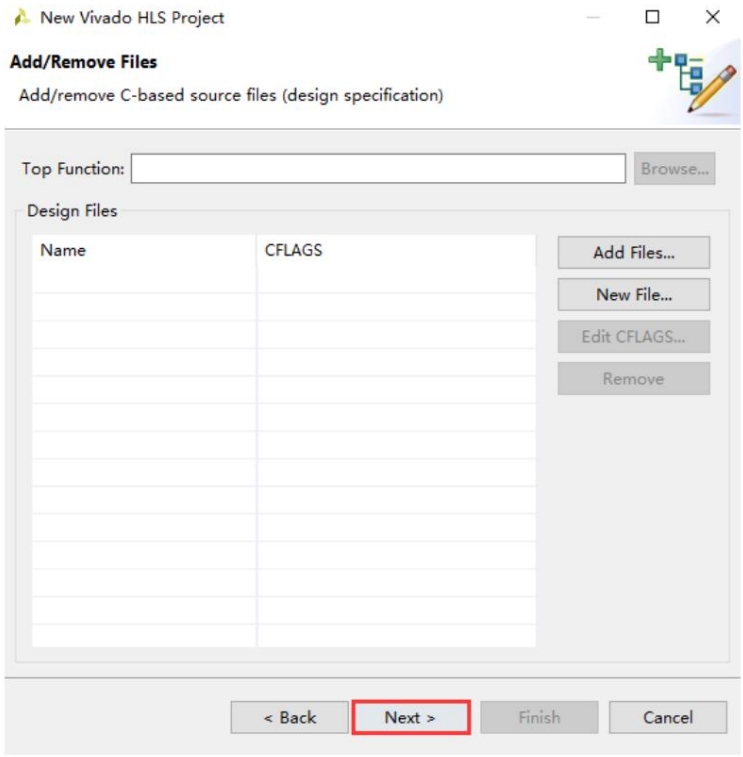


Figure 2- 4

The next step is to add the C simulation file. We will not add it yet. Click Next.

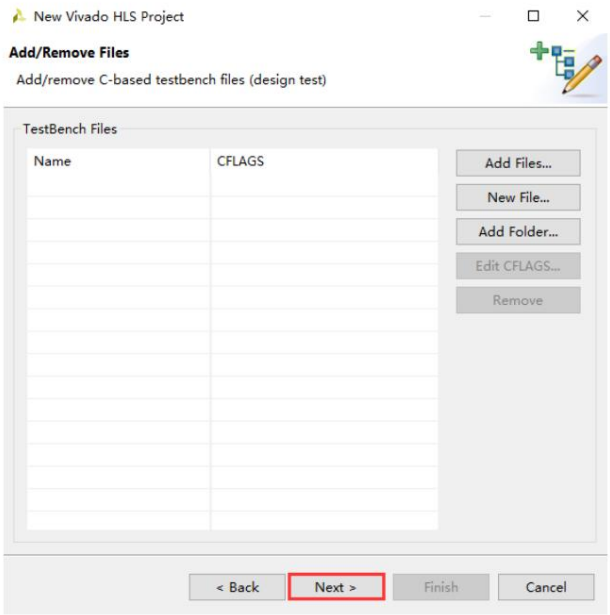


Figure 2-5

Next, we select the device. First, we click on the position shown in Figure 1 to select the component.

During the selection window, users who purchased Z7-Lite7020 selected the device xc7z020clg400-

2. Users who purchased Z7-Lite7010 select the device xc7z010clg400-1 on our development board and click OK.

Click Finish to complete the project creation.

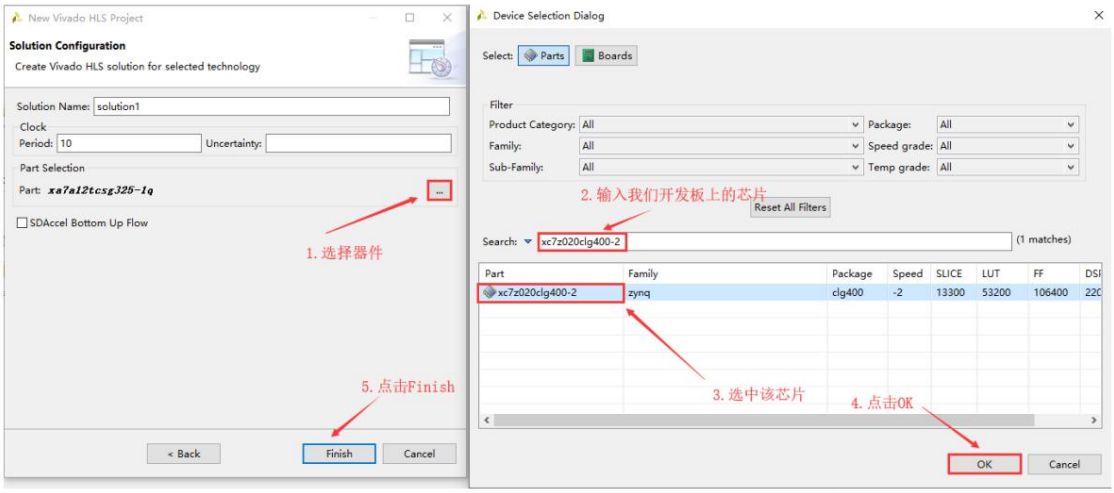


Figure 2-6

The interface after creation is shown below. The red area 1 is the navigation bar, and the green area 2 is the navigation bar.

is the code editing area, blue area 3 is the variable function and Directive area used, purple area 4 is

Output information window.

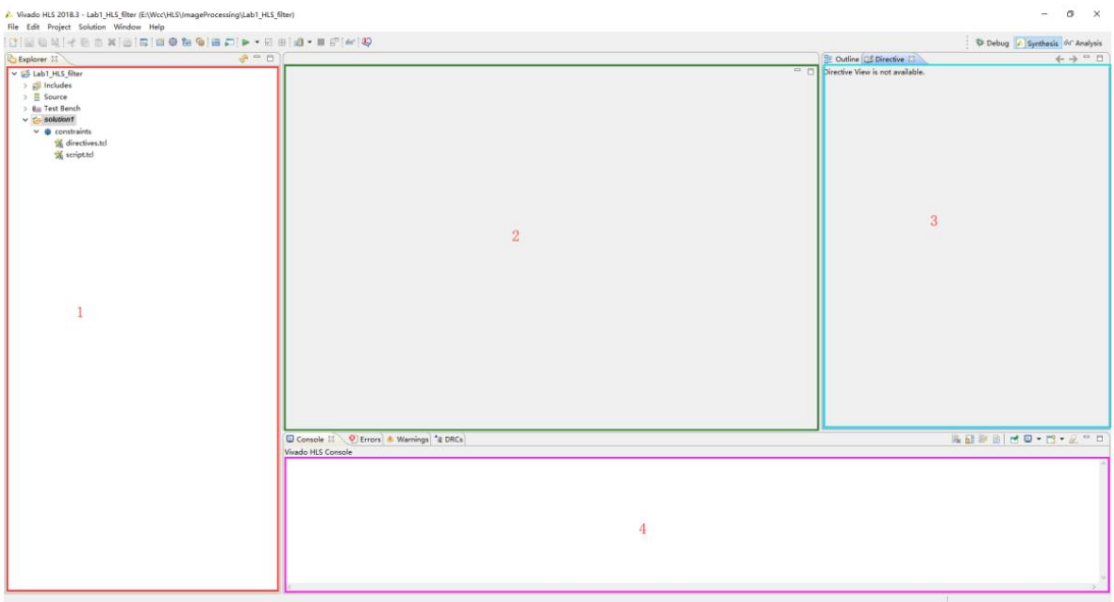


Figure 2- 7

In the navigation bar, the file list is shown as follows:

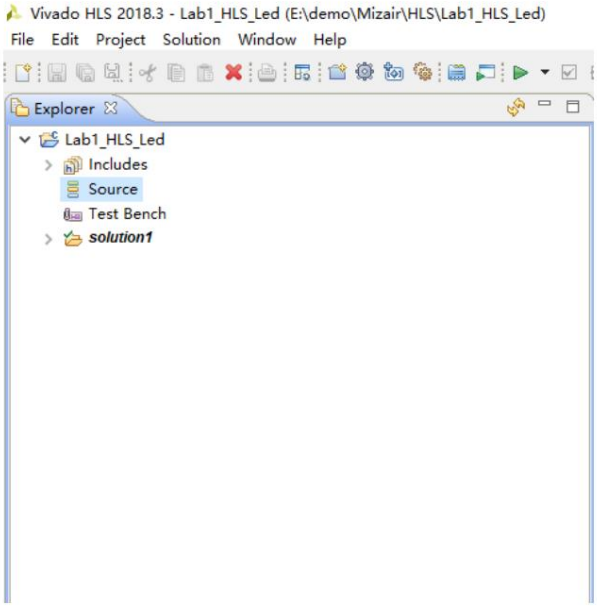


Figure 2- 8

The source column is used to store the source code of the function, and the Test Bench is used to store the C simulation file.

Solution contains the output files during and after the project is run.

2.1.2 Add source files

First, we add the source code of this project, as shown below. Right-click Source and click New file.

In the window that appears, select the directory where we store the source code. Create a new led.cpp file:

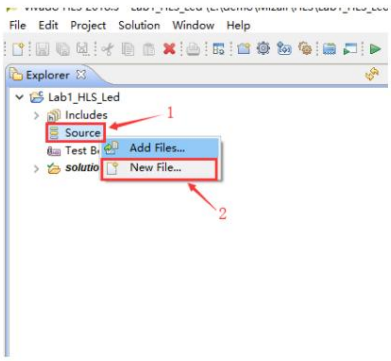


Figure 2- 9

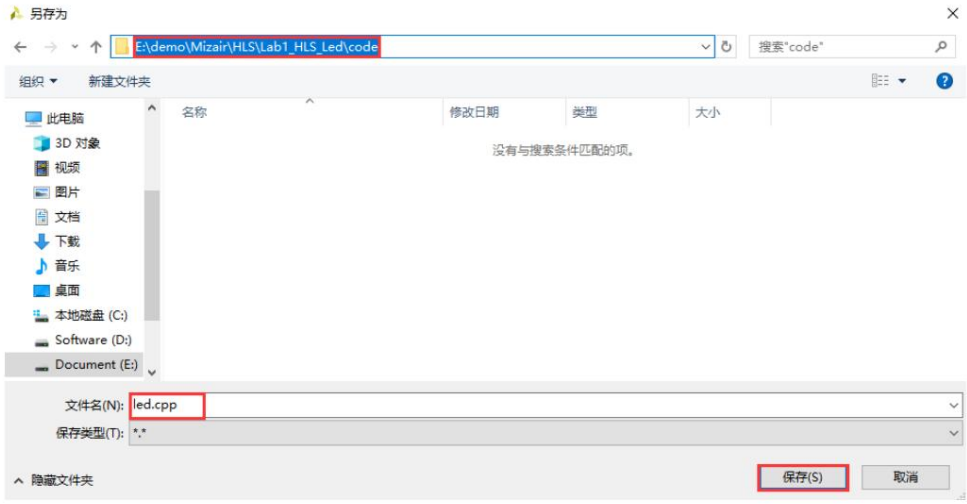


Figure 2- 10

After saving, HLS will automatically open the file we just created. We can write in this file.

The code of the function we want to implement.

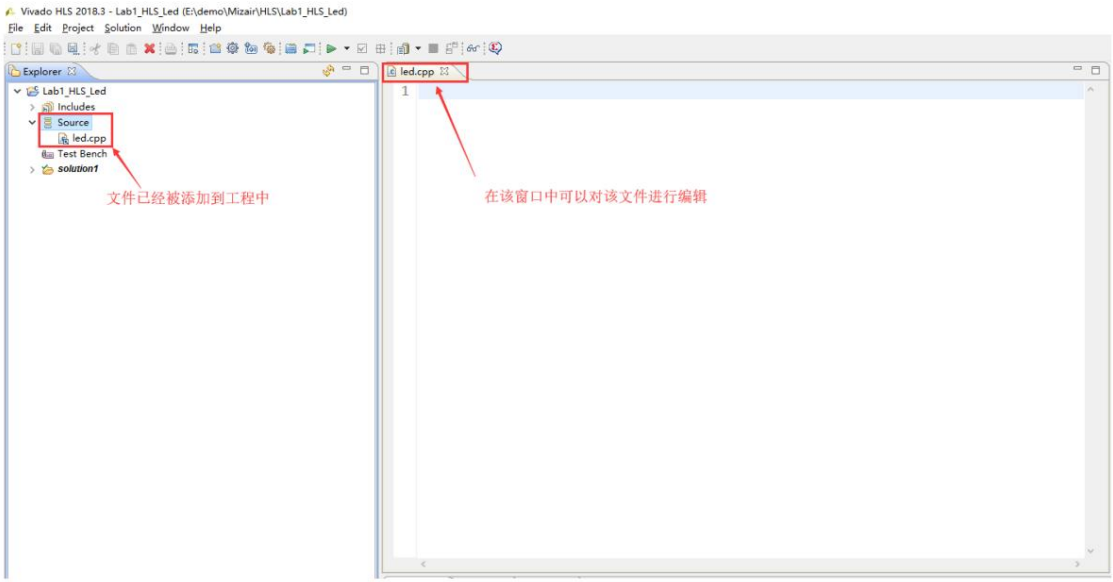


Figure 2- 11

Following the same method, we can add a header file to the cpp file;

Figure 2- 12

Next, we will complete the code design of this experiment. The code design of this experiment is relatively simple.

The code of the header file in the Source file is as follows:

```
1 #ifndef _SHIFT_LED_H_
2 #define _SHIFT_LED_H_
3 #define CNT_MAX 100000000
4 //#define CNT_MAX 100
5 #define FLASH_FLAG CNT_MAX-2
6 typedef        int led_t;
7 typedef        int cnt_t;
8 void flash_led(led_t *led_o , led_t led_i);
9 #endif
```

The maximum count value CNT_MAX 100000000 is the value of one second of counting at a clock frequency of 100M.

The number of counts required, during simulation, we can comment it out and use the next maximum value definition,

This can speed up our simulation. FLASH_FLAG is the LED flashing sign. When the count reaches this value,

The LED changes.

Next is the definition of the int type variable. Finally, a function is declared, which is how we

The fixed function that needs to be designed in this project.

The source file in the Source file is designed as follows:

First, import the header file, then complete the function, using a for loop to count. When the variable i counts

When the count reaches FLASH_FLAG, the state of led_o changes. You can see that this function is very simple.

```
1 #include "led.h"
2
3 void flash_led(led_t *led_o , led_t led_i){
4 cnt_t i; 5
for(i=0;i<CNT_MAX;i++){ if(i==FLASH_FLAG)
6                 { *led_o = ~led_i;
7
8                 }
9       }
10 }
```

2.1.3 Add C simulation file

The steps to add a simulation file are similar to adding a source file. Right-click Test Bench and select Add New File.

In the pop-up window, create a new test_led.cpp file. ;

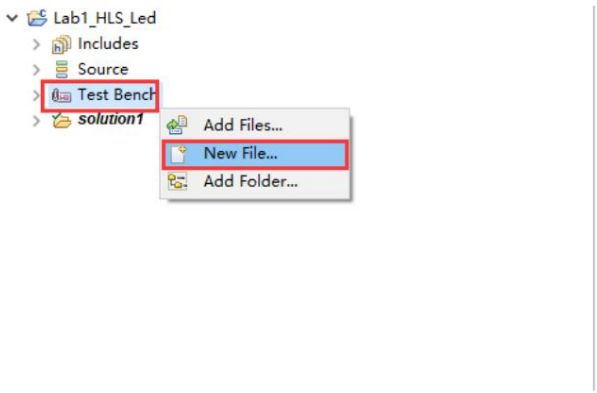


Figure 2- 13

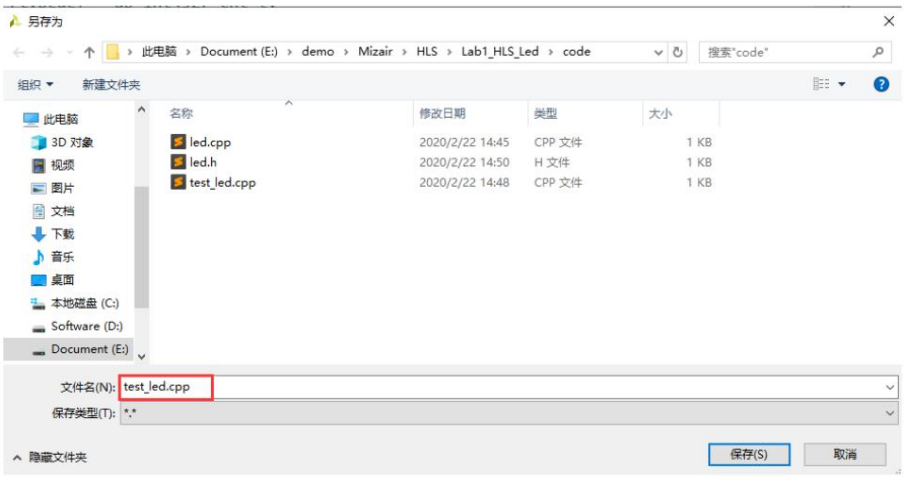


Figure 2- 14

In the C simulation file, we write the following code

```
#include "led.h"
#include <stdio.h>

int main(){

    led_t led_i=0x01;
    led_t led_o;
const int SHIFT_TIME = 4;
    int i;
for(i=0;i<SHIFT_TIME;i++)
            { flash_led(&led_o , led_i); led_i
            = led_o;
            printf("shift_out is %d \n",(int)(led_o&0x01));
}
}
```

In the simulation file, we import the header file led.h and call the flash_led

Function, in the simulation file, we give this output back to the input, and finally output led_o to observe

After completing the C simulation file, we can perform C simulation and C synthesis mapping to RTL circuit

road.

2.1.4 Perform **C** simulation and **C** synthesis

Before simulation, we first select the top-level function we want to simulate, click project ÿ select

project seethings ÿselect synthesis ÿbrowser ÿselect flash_led as the top-level function.

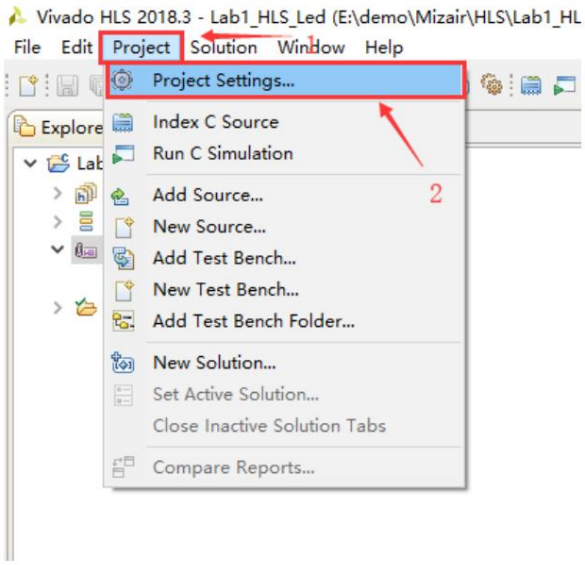


Figure 2- 15

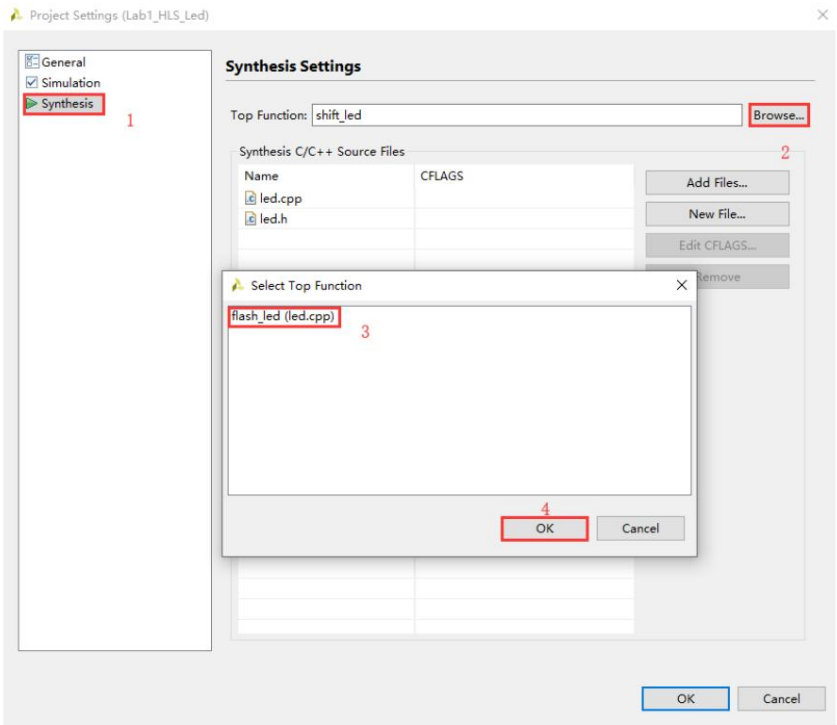


Figure 2- 16

Then click project ÿ Run C simulation

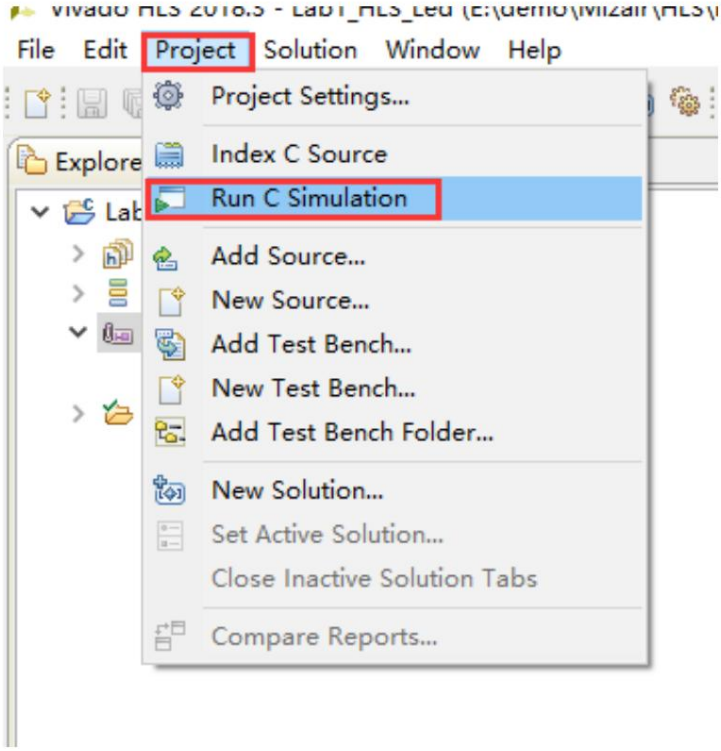


Figure 2- 17

In the Console window, we can see that the output results are alternating between 0 and 1, proving that the C simulation

The result is correct.

Figure 2- 18

Next, perform C synthesis. The compiler will map the C++ code to the RTL circuit. Click Solution.

ÿ Run C Synthesis ÿ Active Solution. After waiting for a while, the compiler will prompt the result of synthesis.

fruit.

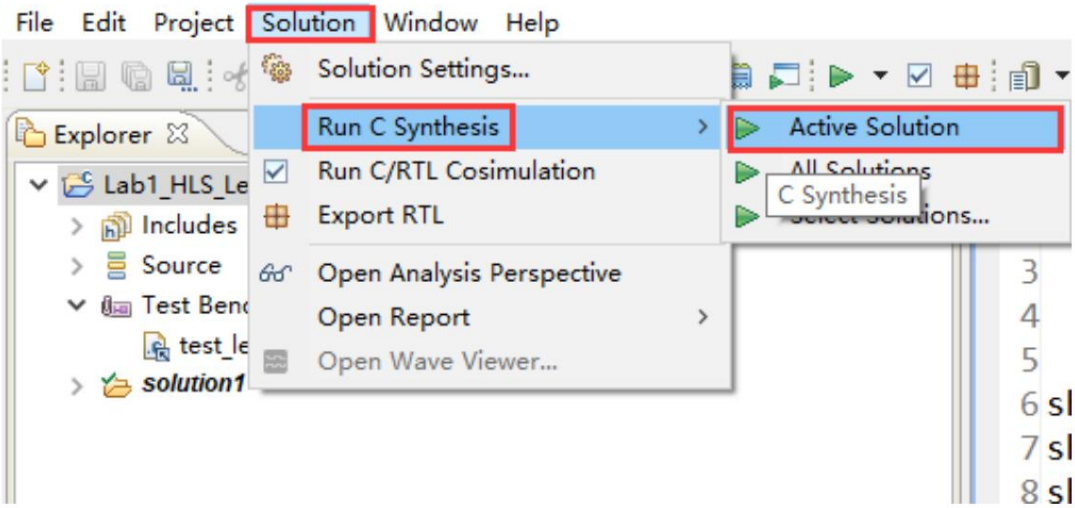


Figure 2- 19

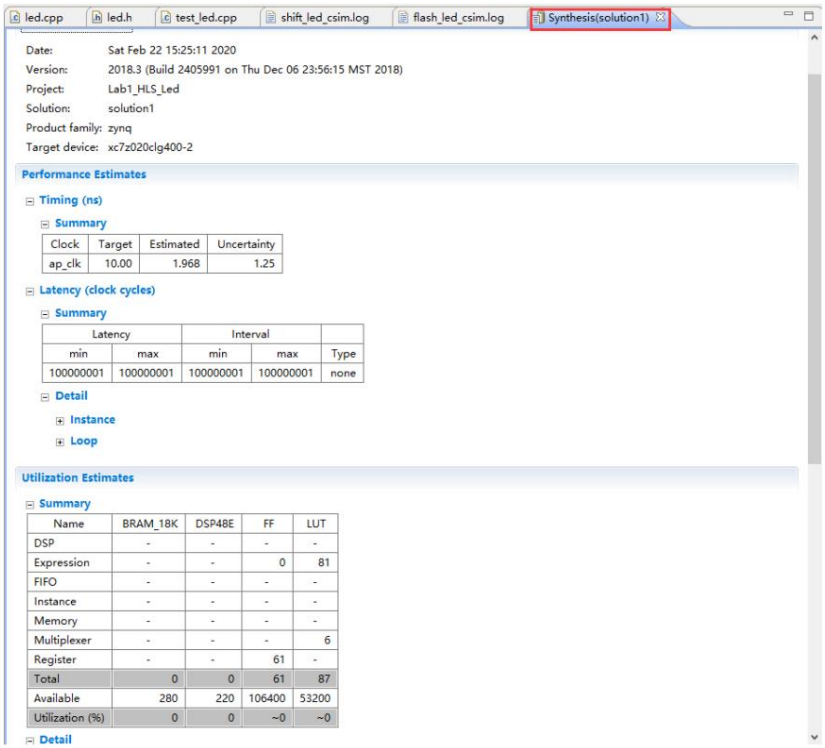


Figure 2- 20

In the C synthesis results, we can view the resources occupied and the latency required for the design.

And the type of interface and so on.

Timing and Latency reports: Latency refers to the time it takes for the designed circuit to complete a task.

Interval refers to the time interval between two tasks.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	1.968	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
100000001	100000001	100000001	100000001	none

Figure 2- 21

Logic resources occupied: As you can see, this design occupies 61 flip-flops and 87 lookup tables;

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	81
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	6
Register	-	-	61	-
Total	0	0	61	87
Available	280	220	106400	53200
Utilization (%)	0	0	~0	~0

Figure 2- 22

The interface of the synthesized module is as follows: We can see that the synthesized signal interface list is as follows

As shown in the figure, we can see that some other signals are generated after synthesis. The generation of these signals can be

Used to mark the working status of this module. It is worth noting that these signals are consistent with the interface protocol adopted.

We will discuss the related interface protocols in the following chapters.

Here we only need to know that the protocols of these interfaces are related to the directives we adopt.

close.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	flash_led	return value
ap_rst	in	1	ap_ctrl_hs	flash_led	return value
ap_start	in	1	ap_ctrl_hs	flash_led	return value
ap_done	out	1	ap_ctrl_hs	flash_led	return value
ap_idle	out	1	ap_ctrl_hs	flash_led	return value
ap_ready	out	1	ap_ctrl_hs	flash_led	return value
led_o	out	32	ap_vld	led_o	pointer
led_o_ap_vld	out	1	ap_vld	led_o	pointer
led_i	in	32	ap_none	led_i	scalar

Figure 2- 23

We can also view the RTL code generated after C synthesis. It is worth noting that the code generated by HLS

The generated Verilog code can be found in the Syn of Solution.

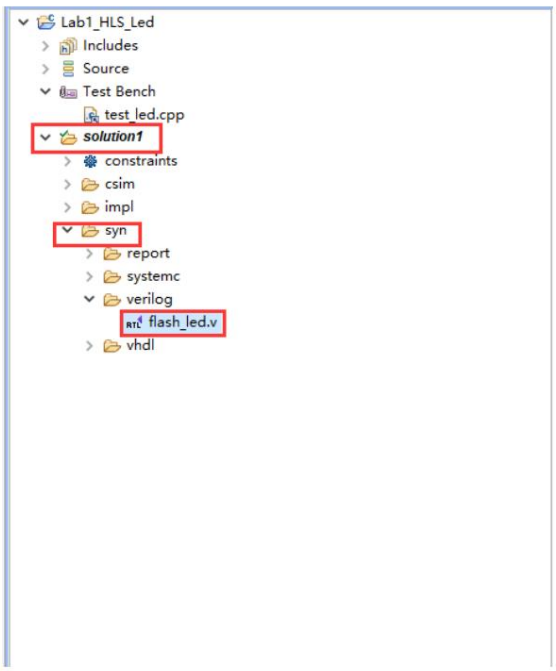


Figure 2- 24

2.1.5 Joint Simulation

Next, we perform C/RTL co-simulation to verify whether the mapped RTL circuit is correct.

It is worth noting that Vivado HLS will use our C Testbench to automatically generate Verilog Testbench.

At the same time, after the joint simulation is completed, we can view the simulation by using Vivado or Modelsim

Waveform.

Click Solution ÿ Run C/RTL Cosimulation

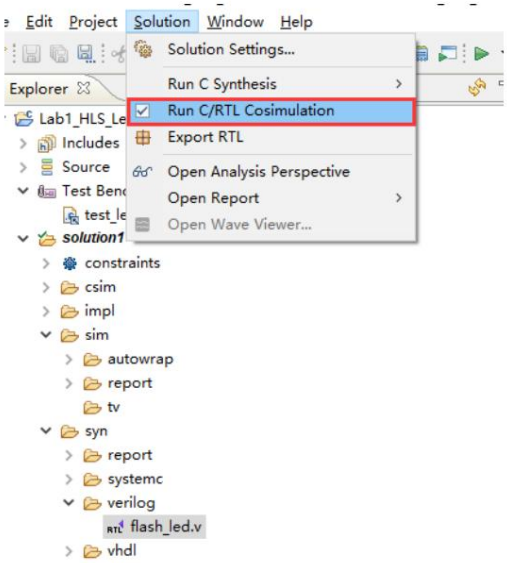


Figure 2- 25

In the pop-up window, first select the simulation tool. I chose Modelsim here. You can also choose

Other simulation tools, as long as the simulation tool is installed on the user's PC and in the environment variable Path,

If there is a path for the simulation tool, HLS will automatically locate that path and start the simulation.

Then select Verilog simulation, Dump Trace All to start the simulation.

Window, observe the printing information.

Since the maximum value in the header file is set too high, the simulation time will be very long, which is consistent with our

In FPGA development, when performing simulation, it is consistent. From this, we can see that HLS is essentially

For circuit development, only high-level languages are used, and the high-level languages are mapped to RTL circuits through tools.

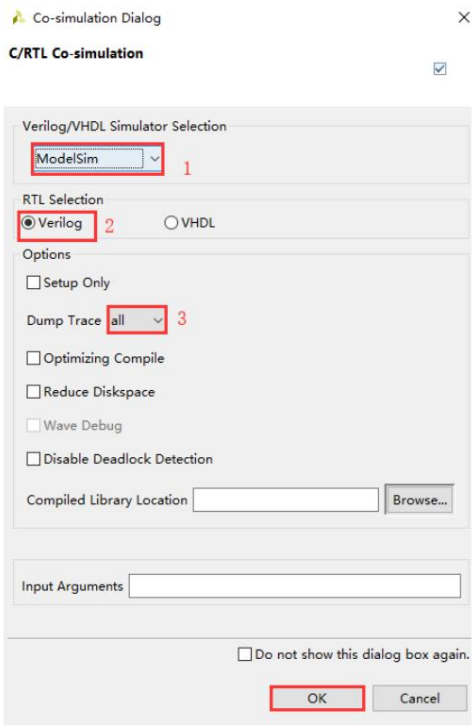


Figure 2- 26

During the joint simulation, we can also observe the results printed by the Console:

The results obtained by C/RTL co-simulation are consistent with those obtained by C simulation.

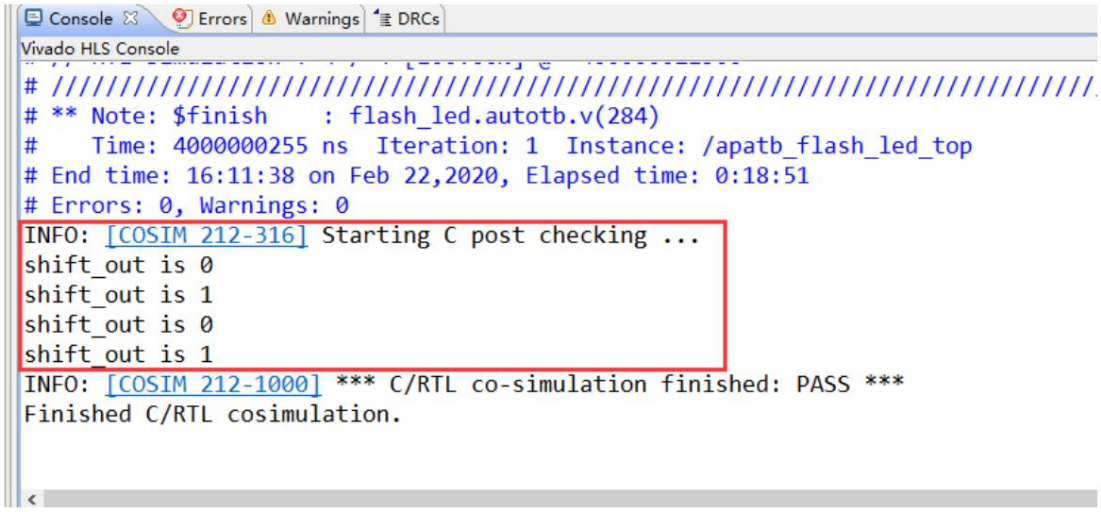


Figure 2- 27

At this point, we can say that the entire design is complete, but we can consider what we have done in

There are still some flaws in the design. For this LED light flashing experiment, we can see that only 1 bit of data is needed.

The bit width can represent this variable, and in this experiment, we still use the int type to define

From the result of C synthesis, the input led_i and output led_o are still 32 bits.

So is there a way in HLS to implement custom bit width like in FPGA development?

The answer is yes. Next, we will modify the design to complete the customized bit-width input and output.

First, we import the ap_int.h header file in the source file header file. This header file is created by

The HLS library provides this header file. By importing this header file, we can call the functions and keywords in it to

Declare data of arbitrary bit width.

The modified header file is as follows:

```
1 #ifndef _SHIFT_LED_H_ 2
#define _SHIFT_LED_H_
3
4 #include "ap_int.h" 5
#define CNT_MAX 100000000 6 //#define
CNT_MAX 100 7 #define
FLASH_FLAG CNT_MAX-2 8 // typedef 9 //
typedef 10              int led_t;
typedef                 int cnt_t;
ap_int<1> led_t; 11 typedef ap_int<32>
cnt_t; 12 void flash_led(led_t *led_o ,
led_t led_i);
13
14 #endif
```

You can see that after we introduced the header file, the definition of the data type has also changed.

Here we use the ap_int template to define the variable type let_t with a data width of 1 bit.

led_t, and variable type cnt_t with a data width of 32 bits.

Next, we add a directive to the function in the source file to determine the type of the interface.

Go to the Directive window and you can see the function we declared, the function parameters and the

In this chapter, we will only add directives and will not further modify them.

Note that we will often use these constraints in subsequent chapters.

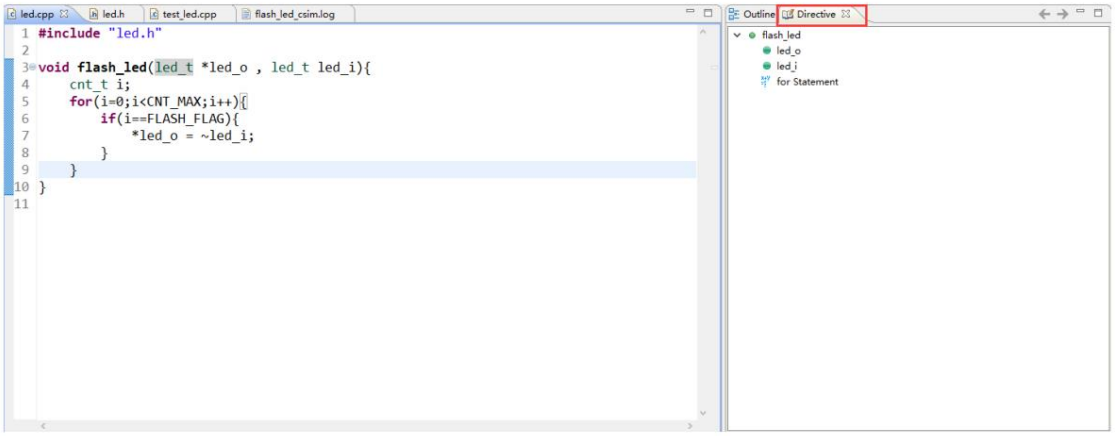


Figure 2- 28

First, constrain the output signal led_o. In the Directive window, right-click led_o and select Insert Directive;

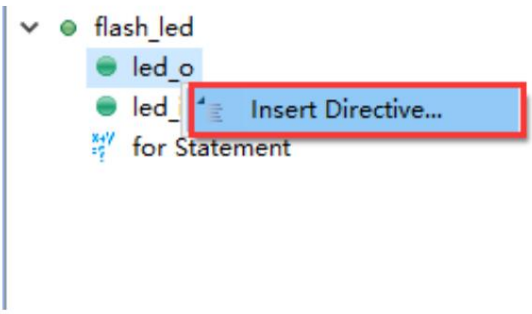


Figure 2- 29

In the pop-up window, select Directive as interface type INTERFACE; Destination select Source file, the tool will add constraint statements to the source file. If you select Directive file, the tool will add the directive Write to a Tcl script, here choose to add to the source file. In the Option option, select ap_ovld This will give the output signal an output valid indication signal. For led_i, use a similar method and select

Select the ap_vld signal in the configurable output section, which will give the input signal an input valid signal.

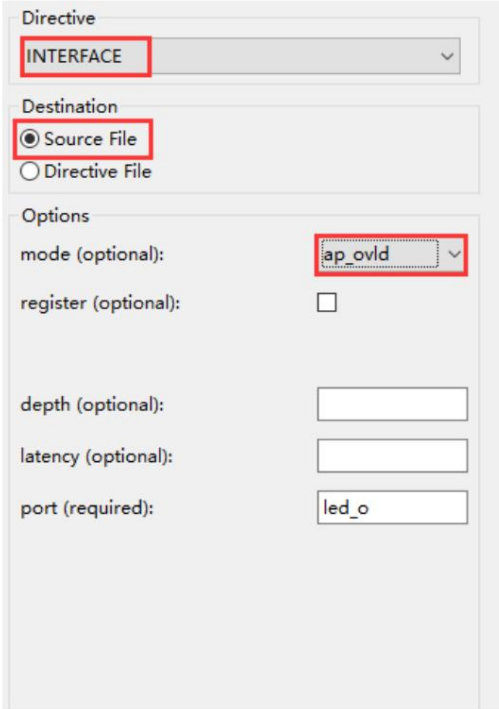


Figure 2- 30

The source file after adding constraints is as follows: As you can see, Directive has been added to the source file.

Then we perform C simulation again.

```
#include "led.h"

void flash_led(led_t *led_o , led_t led_i){ #pragma HLS

INTERFACE ap_vld port=led_i #pragma HLS INTERFACE

ap_ovld port=led_o

        cnt_t i;

        for(i=0;i<CNT_MAX;i++)

                { if(i==FLASH_FLAG){ *led_o

                        = ~led_i;

                }

        }

}
```

After the simulation, we can see that the simulation results are still correct. Next, we will perform C synthesis.

By observing how the comprehensive results after adding Directive differ from the previous results.

led.cpp | led.h | test_led.cpp | flash_led_csim.log

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../code/led.cpp in debug mode
   Generating csim.exe
shift_out is 0
shift_out is 1
shift_out is 0
shift_out is 1
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

Figure 2- 31

It can be seen that after changing the data type to a custom bit width data type, the synthesis generates

The number of required flip-flops is reduced to 32 and the number of lookup tables is reduced to 70.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	52
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	18
Register	-	-	32	-
Total	0	0	32	70
Available	280	220	106400	53200
Utilization (%)	0	0	~0	~0

Figure 2- 32

Next, we look at the interface signal after adding the directive, and we can see that HLS has already provided

The input and output signals are marked with valid input and output flags.

When, you can control its timing.

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	flash_led	return value
ap_rst	in	1	ap_ctrl_hs	flash_led	return value
ap_start	in	1	ap_ctrl_hs	flash_led	return value
ap_done	out	1	ap_ctrl_hs	flash_led	return value
ap_idle	out	1	ap_ctrl_hs	flash_led	return value
ap_ready	out	1	ap_ctrl_hs	flash_led	return value
led_o_V	out	1	ap_ovld	led_o_V	pointer
led_o_V_ap_vld	out	1	ap_ovld	led_o_V	pointer
led_i_V	in	1	ap_vld	led_i_V	scalar
led_i_V_ap_vld	in	1	ap_vld	led_i_V	scalar

Figure 2- 33

Finally, we can run Cosimulation again.

Modelsim or Vivado software opens the waveform generated by the co-simulation.

First open the Modelsim software, in Modelsim, locate Solution, and then open the .wdf

The specific path is Solution/sim/verilog/flash_led.wdf. Use Modelsim to open the file.

You can observe the waveform.

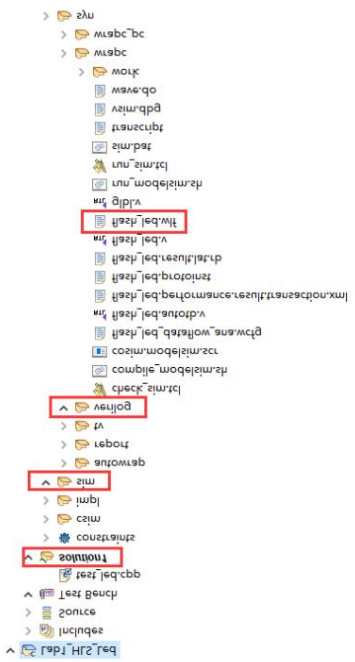


Figure 2- 34

After opening the file, add all the signals defined as modules to the waveform window.

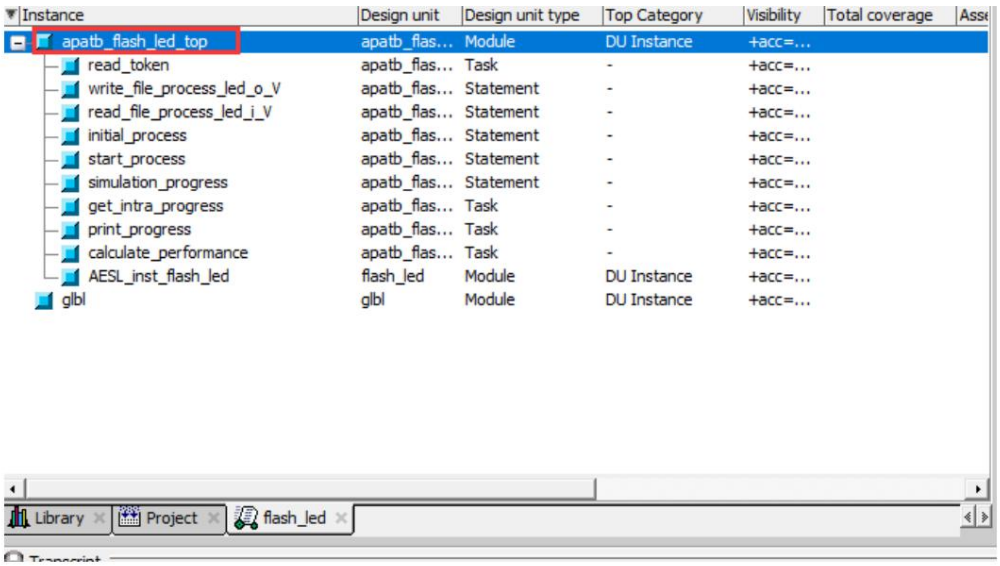


Figure 2- 35

, we can observe the input-input-output signal, and some other, signals, which need to be re-

The signals of interest are those generated after synthesis in C above.

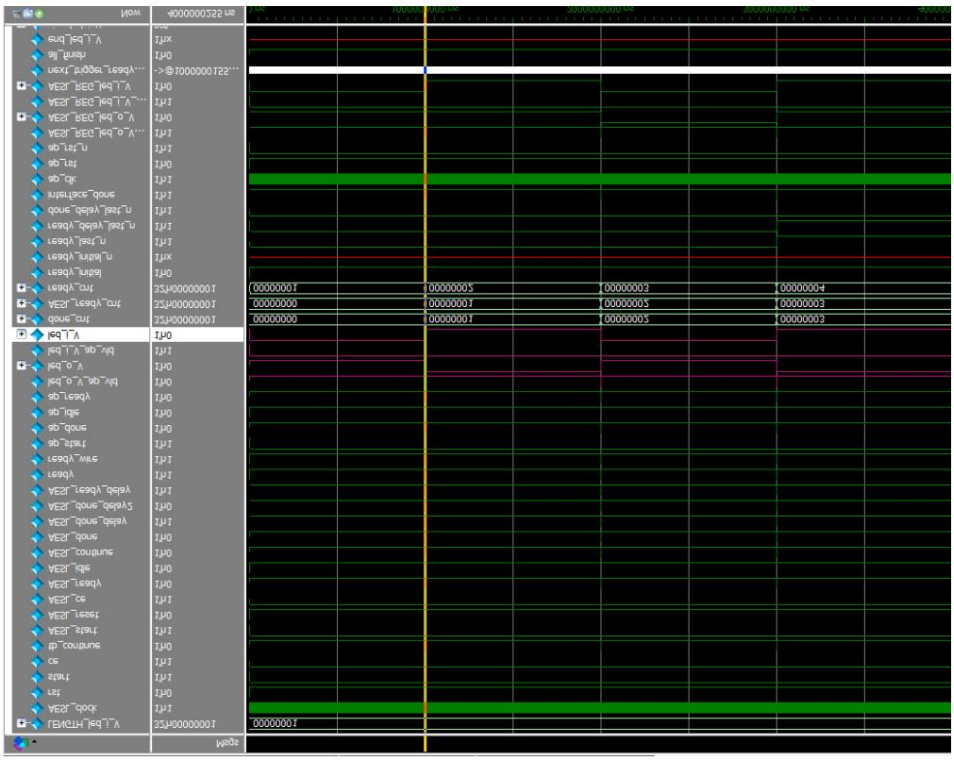


Figure 2- 36

1) Export the **IP** core generated by **the HLS** project

After the above steps, the HLS verification stage has been completed, and the result of the function is known to be

If there is no error, you can then export the generated RTL circuit and call it in Vivado.

Click Solution and select Export RTL to export the RTL circuit generated by HLS as an IP core.

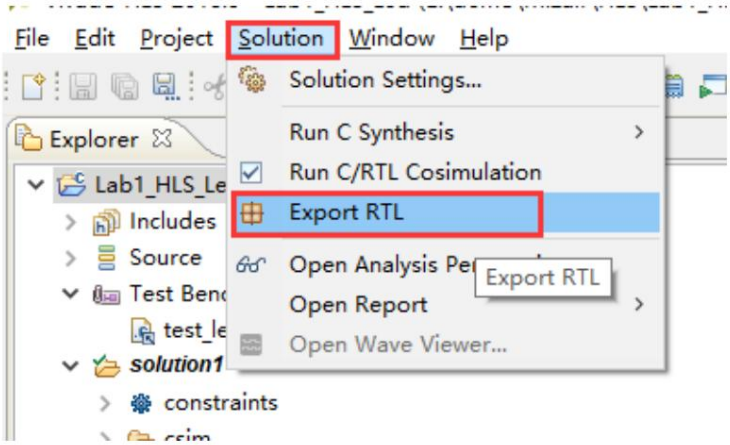


Figure 2- 37

The exported IP core can be found in the Solution folder.

2.2 Create a **Vivado** project and import the **IP** core generated by **HLS**

Next, import the IP core generated by HLS into Vivado. First open Vivado and create a new

In the previous FPGA basic development section, we have introduced how to create a Vivado project.

It is recommended to create a Vivado project and HLS project in the same file.

Folders make it easier for us to manage files.

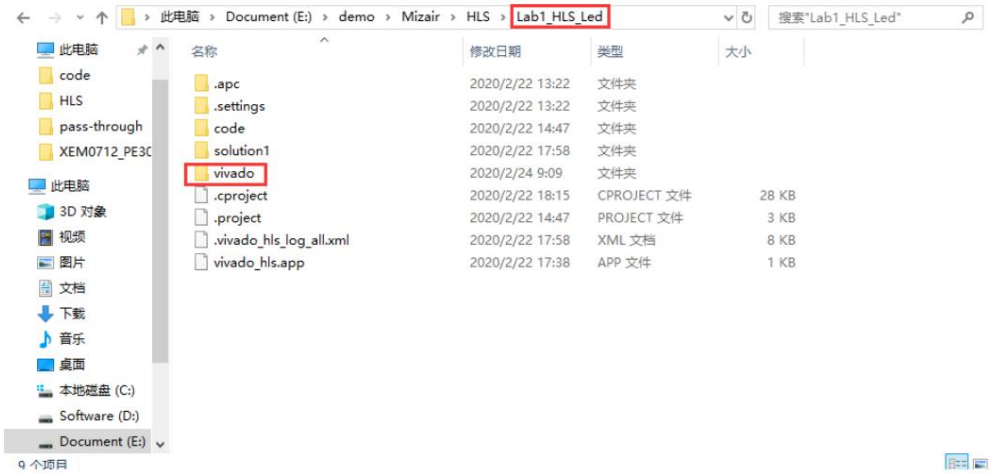


Figure 2- 38

After creating the project, we will add the IP generated by HLS to our IP repository. Click settings

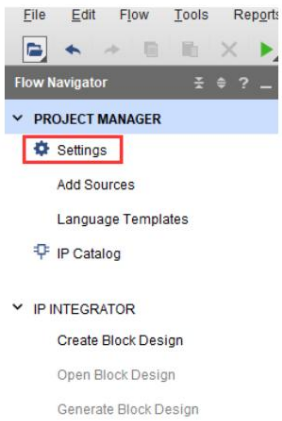


Figure 2- 39

Then click IP, select the warehouse, and click the + sign to add the IP generated by HLS.

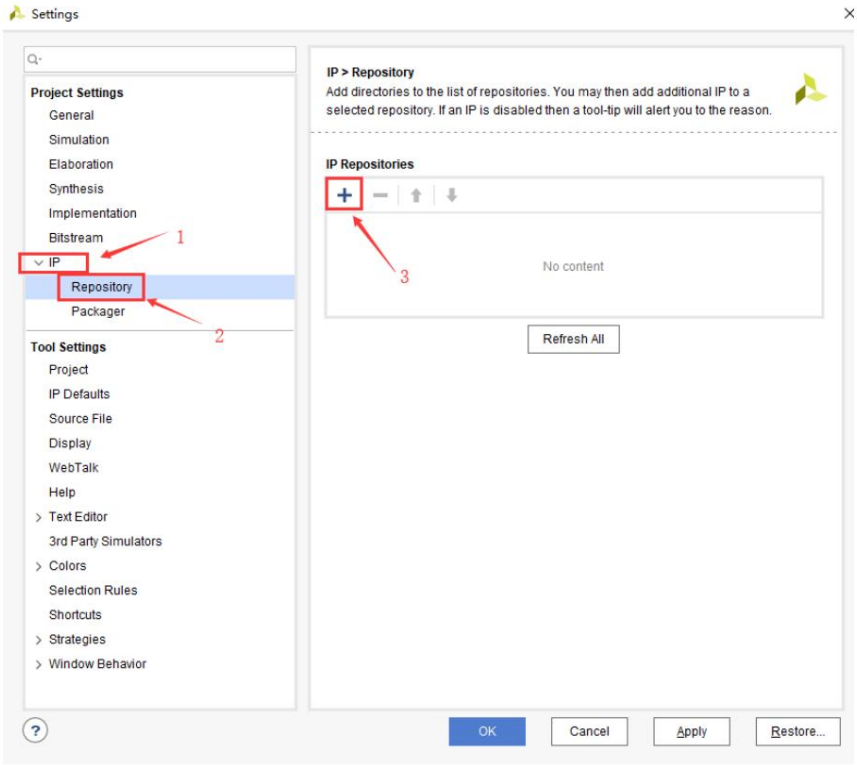


Figure 2- 40

In the pop-up interface, locate Solution and click Select. The system will automatically identify the IP.

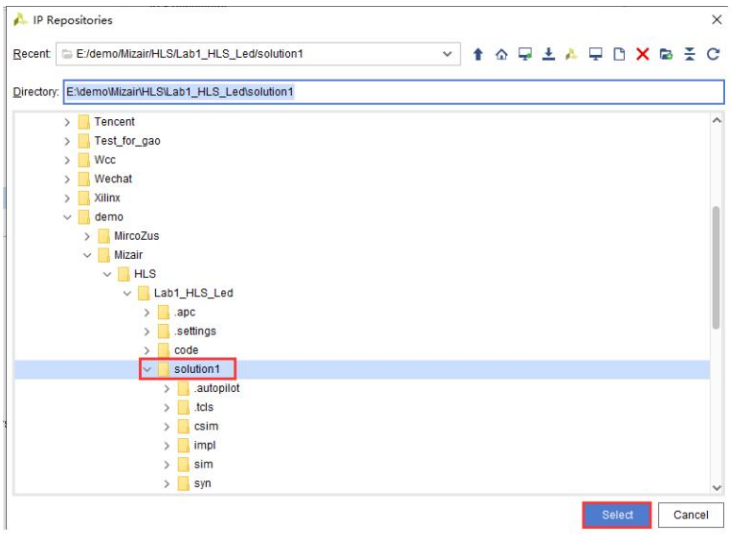


Figure 2- 41

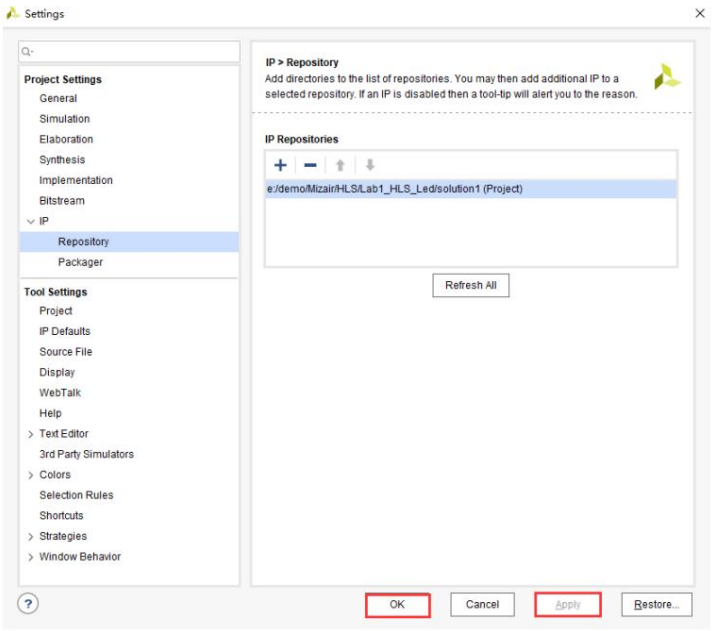


Figure 2- 42

Click Apply ÿ OK to add the IP to the warehouse.

Next, we will verify whether the IP generated by HLS is imported into the Vivado IP repository. Click IP Catalog, you can find that HLS has been added to the IP warehouse;

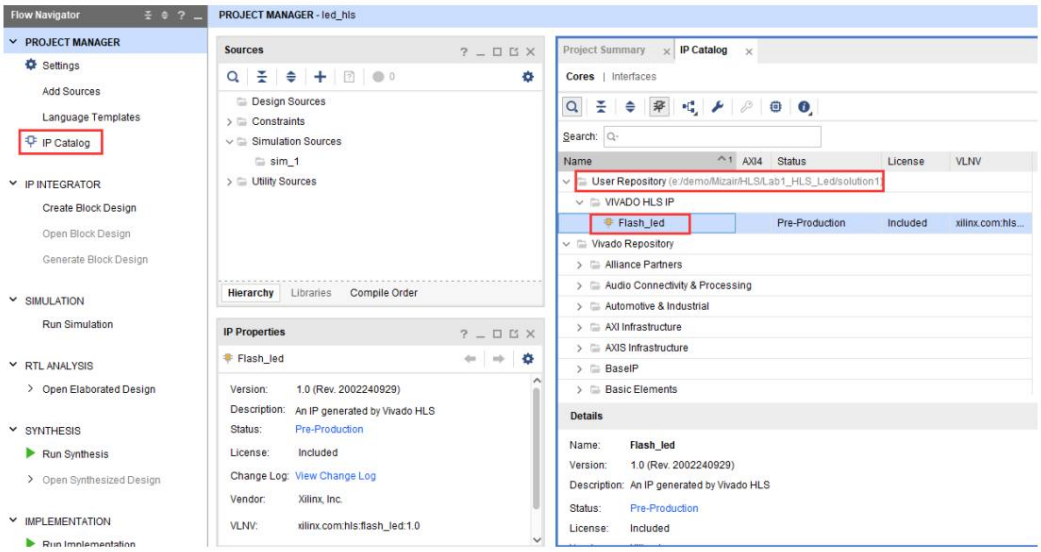


Figure 2- 43

2.3 Add **the IP** generated by **HLS** to the project

Select the IP generated by HLS in IP Catalog, double-click and generate the IP;

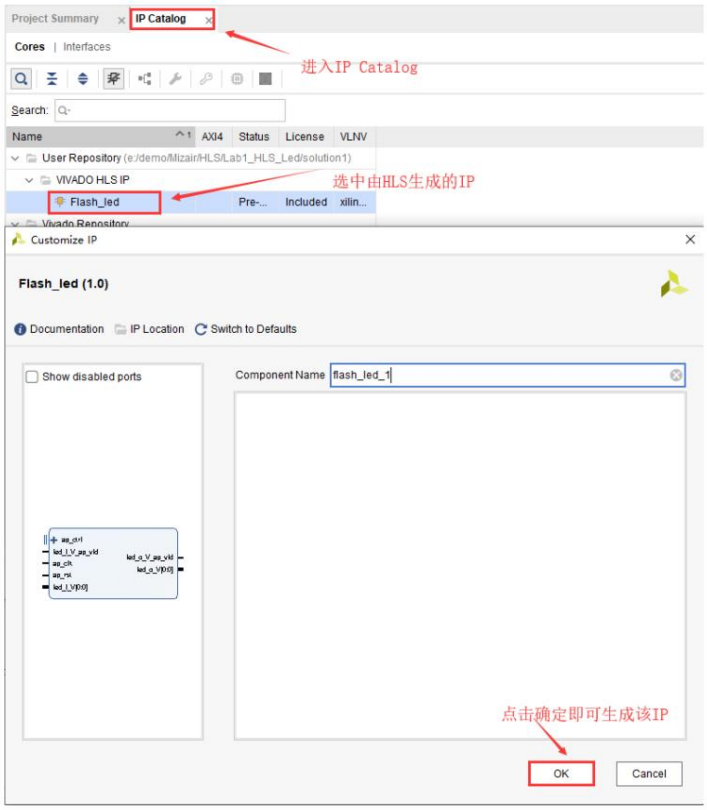


Figure 2- 44

Add a new file to the project to complete this experiment. Click Add file

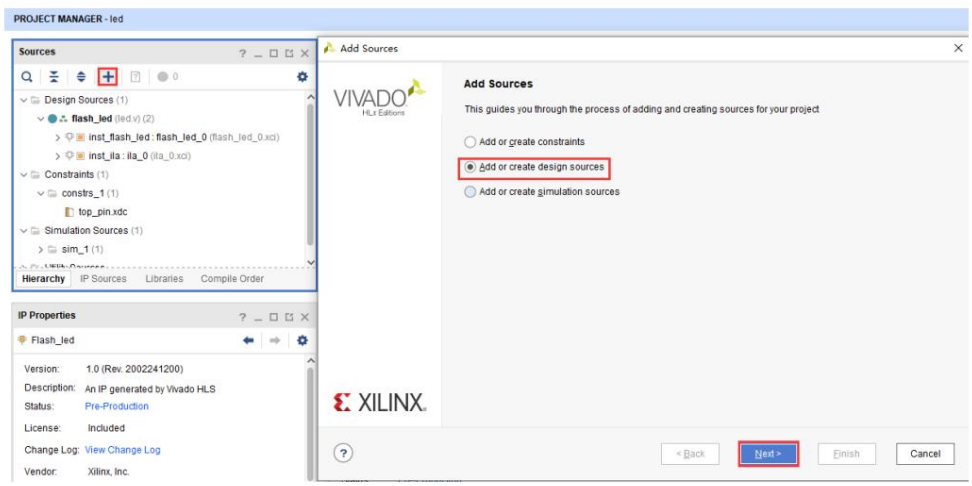


Figure 2- 45

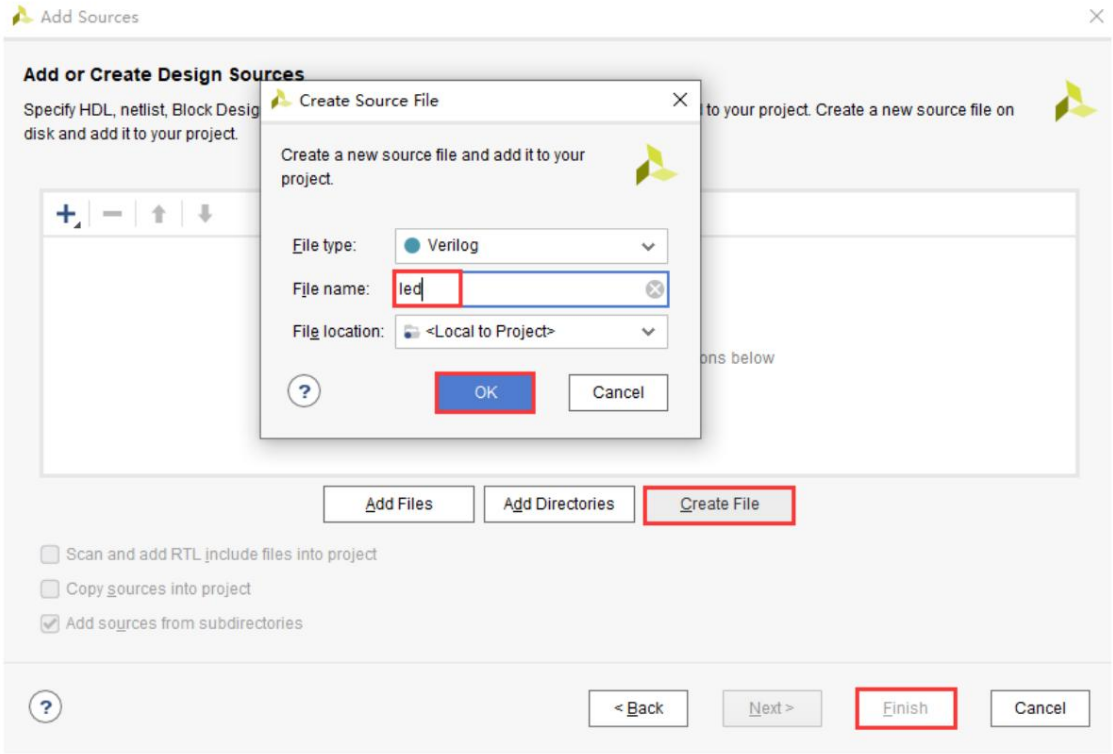


Figure 2- 46

After adding the file, you can write the code. The code for this experiment is relatively simple, so we will not do it here.

There is no need to explain too much, just instantiate the generated HLS IP into the project.

```
1 `timescale 1ns / 1ps
2 module flash_led(
3       input       wire                clk     ,
4 input 5           wire                rst_n       ,
output wire );                          ice_o
6
7
8 wire              rst             ; //Synchronous reset
9 wire              ap_ready            ;//Currently can receive the next data
10 reg 11           ap_start        ;//IP starts working
reg                 led_i_vld           ;//Input data is valid
12 wire                 led_o_vld           ;
13 reg              led_i               ;//Input LED signal
14 wire                 led_o_r         ;
15 wire                 ap_done             ;
16 wire                 ap_idle         ;
17 reg          [1:0]   delay_cnt           ;
```

```
18
19 assign rst = ~rst_n ;
20 assign led_o = led_o_r ;
21
22 //----------------delay_cnt-------------------
23 always @(posedge clk) begin
24             if (rst==1'b1) begin
25                     delay_cnt <= 'd0;
26             end
27             else if(delay_cnt[1]==1'b0) begin
28                     delay_cnt <= delay_cnt + 1'b1;
29             end
30 end
31
32 //----------------ap_start-------------------
33 always @(posedge clk) begin
34             if (rst==1'b1) begin
35                     ap_start <= 1'b0;
36             end
37             else if(delay_cnt[1]==1'b1)begin
38                     ap_start <= 1'b1;
39             end
40 end
41
42 //----------------led_i_vld-----
43 always @(posedge clk) begin
44             if (rst==1'b1) begin
45                     led_i_vld <= 1'b0;
46             end
47             else if(delay_cnt[1]==1'b1)begin
48                     led_i_vld <= 1'b1;
49             end
50 end
51
52 //----------------ap_i-----------------
53 always @(posedge clk) begin
54             if (rst==1'b1) begin
55                     led_i <= 1'b0;
56             end
57             else if(led_o_vld==1'b1)begin
58                     led_i <= led_o_r ;
```

```
59              end
60 end
61
62
63 flash_led_0 inst_flash_led (
64          .led_o_V_ap_vld(led_o_vld), // output wire led_o_V_ap_vld
65          .led_i_V_ap_vld(led_i_vld), // input wire led_i_V_ap_vld
66                                                  // input wire ap_clk
67                                                  // input wire ap_rst
68                                                      // input wire ap_start
69                                                          // output wire ap_done
70                                                      // output wire ap_idle
71                                                          // output wire ap_ready
72          .ap_clk(clk), .ap_rst(rst), .ap_start(ap_start), .ap_done(ap_done), .ap_idle(ap_idle), .ap_ready(ap_ready), .led_o_V(led_o_r),
73          .led_i_V(led_i)                         // input wire [0 : 0] led_i_V
74 );
75
76 endmodule
```

Next, you can add constraint files and perform final board-level verification.

The added constraint files are as follows:

```
###############LED define################### set_property
PACKAGE_PIN P15 [get_ports {led_o}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_o}]

###############Reset define################### set_property
PACKAGE_PIN P16 [get_ports {rst_n}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst_n}]

###############50M CLK define################### create_clock
-period 20.000 -name clk -waveform {0.000 10.000} [get_ports clk]
set_property PACKAGE_PIN N18 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
```

2.4 Download Verification

After adding the constraint file, you can proceed to the final board-level verification. First, generate the bit stream file;

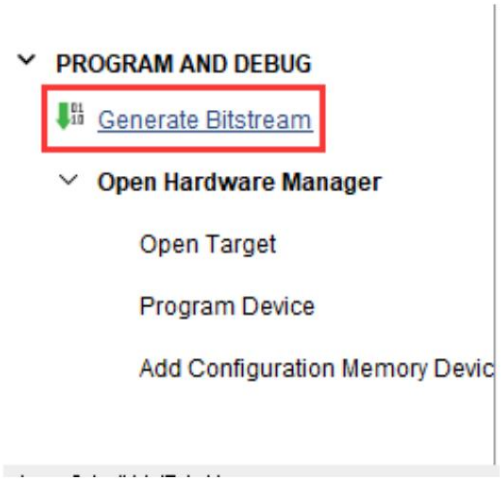


Figure 2- 47

After the bit file is generated, connect the development board as follows:

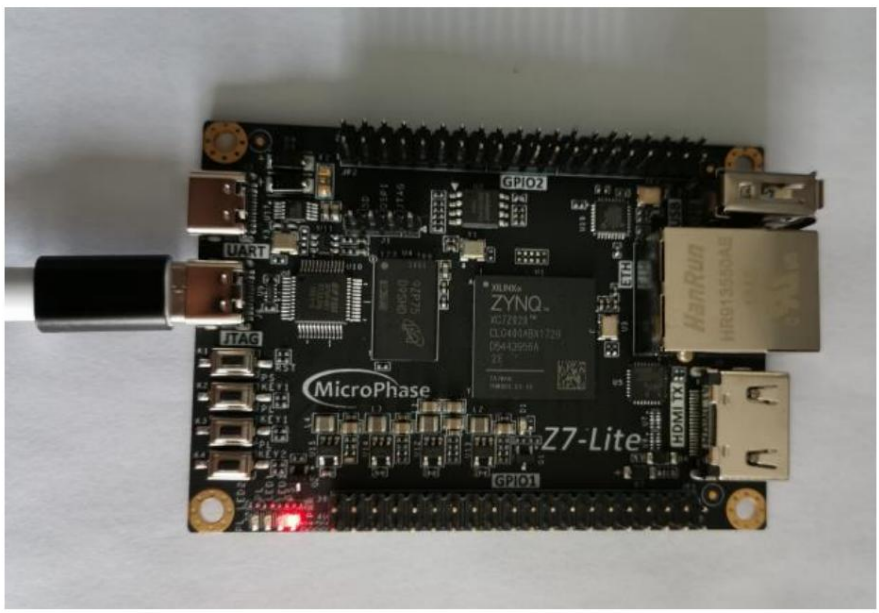


Figure 2- 48

Then download the bit file into FPGA and observe the experimental phenomenon. Finally, it can be found that the development board

The LED light starts to turn on and off continuously.

2.5 Experimental Summary

Through this experiment, we have completed the process of packaging an IP through HLS and applying this IP to the Vivado tool.

During the process, you can see that the whole process is quite complicated. This experiment is very simple.

This section shows the process of HLS development, but does not reflect the advantages of HLS. In the following chapters, we will further

Further deepen the use of HLS software.

To summarize the basic steps of HLS development, first you need C/C++ source files and C/C++ simulation files.

Perform C simulation and then C synthesis to map the functions described in C/C++ language to RTL circuits.

Then, perform C/RTL joint simulation and generate Verliog simulation Testbench through C simulation Testbench.

Verify the correctness of the RTL circuit.

Chapter 3 Building an **HLS** Image Simulation Platform

In this section, we will briefly introduce the OpenCV library included in HLS.

Understand the process of developing image algorithms using HLS, and distinguish between synthesizable code and C simulation code in HLS.

The connections and differences between them.

3.1 Introduction to OpenCV and OpenCV Library in HLS

OpenCV is a library that contains many common computer vision functions for desktop processors and GPUs.

Although many functions in the OpenCV library have been heavily optimized to make many

Many computer vision applications are capable of running in near real time, but an optimized embedded implementation is often preferable.

Vivado HLS includes many video libraries that make it easier for users to build various video processing systems.

These libraries are implemented as synthesizable C++ code that roughly corresponds to the functionality implemented in OpenCV.

Many video concepts and abstractions are similar to those in OpenCV.

In particular, many functions in the OpenCV image processing module have corresponding Vivado HLS libraries.

function.

OpenCV can be used in several places during the design of a video processing system.

OpenCV function calls can be used to design and implement a complete algorithm, either using file access functions to input

The algorithm can be used in embedded systems such as Zynq Base

TRD) using platform-specific function calls to access input and output images. In this case,

Video processing is still implemented using OpenCV function calls executed on the processor (such as the Zynq processor

The Cortex-A9 processor core in the system).

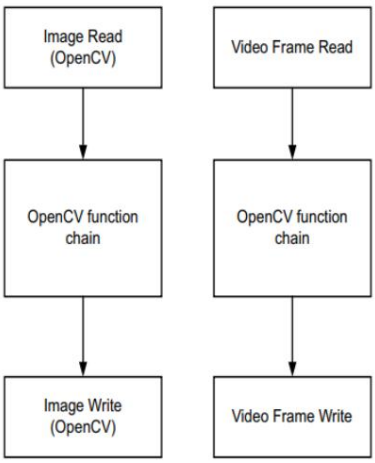


Figure 3- 1

OpenCV function calls can also be replaced by corresponding synthesizer functions in the Xilinx Vivado HLS video library.

OpenCV function calls can be used to access input and output images and provide a golden rule for video processing algorithms.

Reference implementation. The synthesized processing blocks can be integrated into the Zynq programmable logic.

The implemented design, the integrated block can process the video stream created by the processor, such as data read from a file,

Or a live video stream from external input.

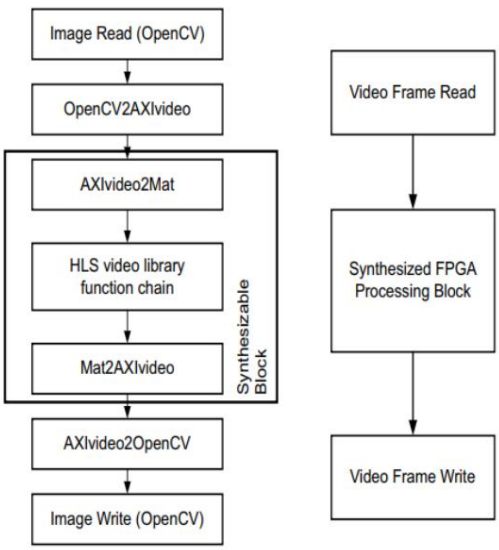


Figure 3- 2

3.2 Commonly used structures and functions in **OpenCV**

In this chapter, we first introduce the commonly used OpenCV functions and understand how to use OpenCV to process images.

These functions will provide reading input images and saving generated images when building a C simulation platform.

Provides golden reference comparison images, etc. Through these functions, we can complete the corresponding

To compare the functions, you can determine whether the functions are correct.

To use this, you need to import the header files hls_opencv.h and opencv2/opencv.hpp.

3.2.1 IplImage Structure

IplImage is the basic data structure in OpenCV, used to represent images, where Ipl is Intel Image

Abbreviation for Processing Library. IplImage is a structure, where the structure definition is:

```
typedef struct _IplImage
{
    int  nSize;             /* sizeof(IplImage) */
    int  ID;                /* version (=0)*/
    int  nChannels;         /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int  alphaChannel;      /* Ignored by OpenCV */
    int  depth;             /* Pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                               IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported.  */
    char colorModel[4];     /* Ignored by OpenCV */
    char channelSeq[4];     /* ditto */
    int  dataOrder;         /* 0 - interleaved color channels, 1 - separate color channels.
                               cvCreateImage can only create interleaved images */
    int  origin;            /* 0 - top-left origin,
                               1 - bottom-left origin (Windows bitmaps style).  */
    int  align;             /* Alignment of image rows (4 or 8).
                               OpenCV ignores it and uses widthStep instead.    */
    int  width;             /* Image width in pixels.                           */
    int  height;            /* Image height in pixels.                          */
    struct _IplROI *roi;    /* Image ROI. If NULL, the whole image is selected. */
    struct _IplImage *maskROI;      /* Must be NULL. */
    void  *imageId;                 /* "             " */
    struct _IplTileInfo *tileInfo;  /* "             " */
    int  imageSize;         /* Image data size in bytes
                               (==image->height*image->widthStep
                               in case of interleaved data)*/
    char *imageData;        /* Pointer to aligned image data.         */
    int  widthStep;         /* Size of aligned image row in bytes.    */
    int  BorderMode[4];     /* Ignored by OpenCV.                     */
    int  BorderConst[4];    /* Ditto.                                 */
    char *imageDataOrigin;  /* Pointer to very origin of image data
                               (not necessarily aligned) -
                               needed for correct deallocation */
}
IplImage;
```

Figure 3- 3

According to the definition of this structure, we can see that the data of this structure type contains the image

The size, number of channels, depth, etc. The part of the IplImage structure that stores image data is char

* imageData, this pointer points to the first address of the image array.

1) Load the image **cvLoadImage**

In image development, it is often necessary to read image files from the outside to the application. In OpenCV, you can Use the cvLoadImage function in OpenCv to read the image data.

The definition of cvLoadImage is as follows: Load a file from a file.

Through parameters, you can change or not the state of the image.

```
enum
{
/* 8bit, color or not */
    CV_LOAD_IMAGE_UNCHANGED =-1,
/* 8bit, gray */
    CV_LOAD_IMAGE_GRAYSCALE =0,
/* ?, color */
    CV_LOAD_IMAGE_COLOR     =1,
/* any depth, ? */
    CV_LOAD_IMAGE_ANYDEPTH =2,
/* ?, any color */
    CV_LOAD_IMAGE_ANYCOLOR =4
};
/* load image from file
   iscolor can be a combination of above flags where CV_LOAD_IMAGE_UNCHANGED
   overrides the other flags
   using CV_LOAD_IMAGE_ANYCOLOR alone is equivalent to CV_LOAD_IMAGE_UNCHANGED
   unless CV_LOAD_IMAGE_ANYDEPTH is specified images are converted to 8bit
*/
CVAPI(IplImage*) cvLoadImage( const char* filename, int iscolor CV_DEFAULT(CV_LOAD_IMAGE_COLOR));
```

The specific usage is as follows:

```
IplImage * img_0; img_0 =
cvLoadImage("fileName.bmp");
```

img_0 will point to the address of the loaded image. In this way, the image can be imported into the application.

Use the program.

2) Create an image **cvCreateImage**

The function cvCreateImage creates the image's first address and allocates storage space.

```
/* Creates IPL image (header and data) */
CVAPI(IplImage*) cvCreateImage( CvSize size, int depth, int channels);
```

The usage is as follows, where size is the size of the image to be created, and depth is the depth of the image to be created.

channels is the number of channels to create the image.

```
IplImage * img_0; img_0 =
cvCreateImage( size, depth,channels);
```

3) Save the image **cvSaveImage**

Its function is to save the image to a file. The function definition is as follows:

```
enum
{
    CV_IMWRITE_JPEG_QUALITY =1,
    CV_IMWRITE_PNG_COMPRESSION =16,
    CV_IMWRITE_PNG_STRATEGY =17,
    CV_IMWRITE_PNG_BILEVEL =18,
    CV_IMWRITE_PNG_STRATEGY_DEFAULT =0,
    CV_IMWRITE_PNG_STRATEGY_FILTERED =1,
    CV_IMWRITE_PNG_STRATEGY_HUFFMAN_ONLY =2,
    CV_IMWRITE_PNG_STRATEGY_RLE =3,
    CV_IMWRITE_PNG_STRATEGY_FIXED =4,
    CV_IMWRITE_PXM_BINARY =32
};

/* save image to file */
CVAPI(int) cvSaveImage( const char* filename, const CvArr* image,
                        const int* params CV_DEFAULT(0) );
```

4) Release the image **cvReleaseImage**

The function is to release the space temporarily used by the image.

The function definition is as follows:

```
/* Releases IPL image header and data */
CVAPI(void) cvReleaseImage( IplImage** image );
```

Here's how to use it:

```
IplImage * img_0;
cvReleaseImage(&img_0);
```

3.2.2 Mat class in OpenCV

In the early OpenCV1.x version, image processing was implemented through the IplImage structure.

The early OpenCV was written in C language, so the interface provided was also a C language interface. Its source code

It is completely C programming style. IplImage structure is the basic data structure of OpenCV matrix operation.

OpenCV2.x version, OpenCV open source library introduced object-oriented programming ideas, a large number of source code in C++

Rewrite, the Mat class (short for Matrix) is a wrapper class introduced by OpenCV for image processing.

Functionally, the Mat class is further enhanced based on the IplImage structure, and due to the introduction of

C++ advanced programming features, the extensibility of the Mat class is greatly improved, and the content of the Mat class is not

If you look at the definition of the Mat class, you will find that its design and implementation are very comprehensive and specific.

Basically covers the basic requirements of computer vision for image processing.

The Mat class is very powerful. In this chapter, we will only introduce the simple initialization of its objects.

If you are interested, you can enter the library file to view the source code.

1) Read the image **imread**

The definition of imread is as follows, by reading an external file,

```
enum
{
        // 8bit, color or not
        IMREAD_UNCHANGED =-1, // 8bit,
        gray
        IMREAD_GRAYSCALE =0,
        // ?, color
        IMREAD_COLOR              =1,
```

```
        // any depth, ?
        IMREAD_ANYDEPTH =2, // ?,
        any color
        IMREAD_ANYCOLOR =4
};
CV_EXPORTS_W Mat imread( const string& filename, int flags=1 );
```

The object of the Mat class can be initialized through imread. The specific usage is as follows:

```
Mat src_rgb = imread("fileName.bmp");
IplImage src = src_rgb;
cvShowImage("src",&src);
```

The above code can read the image and display it.

3.3 Synthesizable Image Processing Modules in **HLS**

In the previous section, the OpenCV functions introduced can be directly called by HLS software, but

However, it is not synthesizable as supported by HLS. For image processing functions that can be synthesized and mapped into RTL circuits,

For further processing, follow the process indicated in Error! Reference source not found.

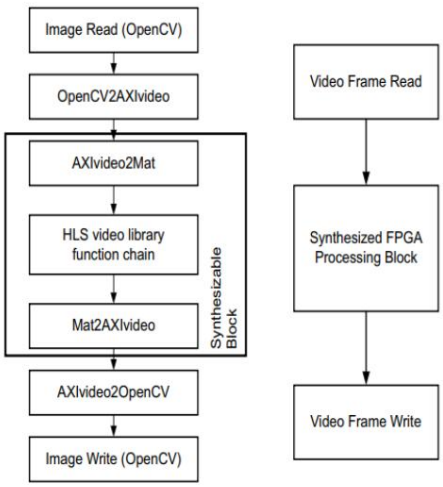


Figure 3-4 HLS process for processing synthesizable modules

From the error! Reference source not found. Error! Reference source not found. it can be seen that the synthesizable modules

The input and output streams are in AXI-Stream format. Xilinx video processing components usually use a common

AXI4 stream protocol is used to transmit pixel data. The protocol describes each row of video pixels as an AXI4 packet.

The last pixel of each row is asserted with the TLAST signal. For the AXI-Stream protocol, please refer to UG761.

https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/la

test/ug761_axi_reference_guide.pdf

From the above flowchart, it can be seen that when using the synthesizable module of HLS, you first need to

The image is converted into data in the AXI-stream format supported by the compiler. When outputting images, the AXI-

The data in stream format is converted into the same data format as OpenCV supported by HLS software.

Among them, the functions used to convert these two data formats are:

IplImage to AXIVideo:

```
Template<int W>
void IplImage2AXIvideo(IplImage* img, hls::stream<ap_axiu<W,1,1,1> >& AXI_video_strm);
```

AXIVideo to IplImage:

```
template<int W>
void AXIvideo2IplImage(hls::stream<ap_axiu<W,1,1,1> >& AXI_video_strm, IplImage* img);
```

hls::stream is an object, which can be understood as a stream (note: personal understanding may not be

Accurate), which contains data of type ap_axiu. ap_axiu is a structure whose data members

It constitutes the signals required by the AXI-stream protocol.

After you have the AXi-stream data stream, you need to convert the data stream to image processing supported by HLS.

In hls, similar to OpenCV, the image matrix Mat is used to represent

Display image.

The following two functions can be used to convert AXI-stream format data into HLS synthesizable Mat

type.

AXIvideo to Mat

```
template<int W, int ROWS, int COLS, int T>
```

```
int AXIvideo2Mat(stream<ap_axiu<W,1,1,1> >& AXI_video_strm,Mat<ROWS, COLS, T>& img);
```

Mat to AXIvideo

```
template<int W, int ROWS, int COLS, int T>
int Mat2AXIvideo(Mat<ROWS, COLS, T>& img,stream<ap_axiu<W,1,1,1> >& AXI_video_strm);
```

In HLS, a simple way to distinguish between synthesizable and non-synthesizable functions is to look at the operations before the function.

If the function begins with hls::function() and ends with hls, then the function is generally synthesizable. If it begins with cv

begins with , then the function is generally not synthesizable.

3.4 Grayscale Conversion Example

Next, let's take a look at the grayscale conversion of an image to illustrate the HLS development process shown above.

To practice, we build a simulation platform for image algorithms. The experiments in this chapter are only simulations.

3.4.1 Create **HLS** Project

Open the HLS software and create a new project according to the steps in the previous experiment;

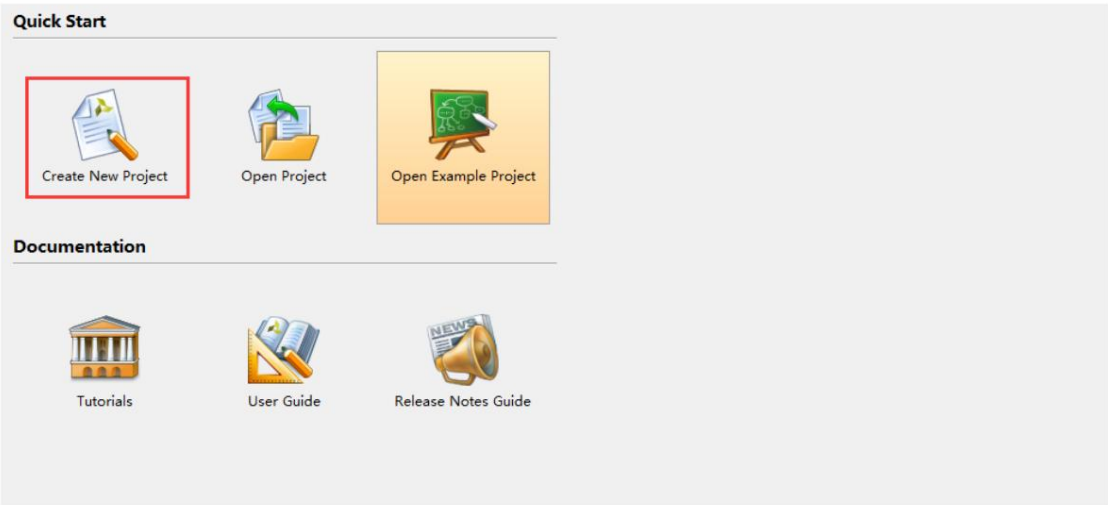


Figure 3- 5

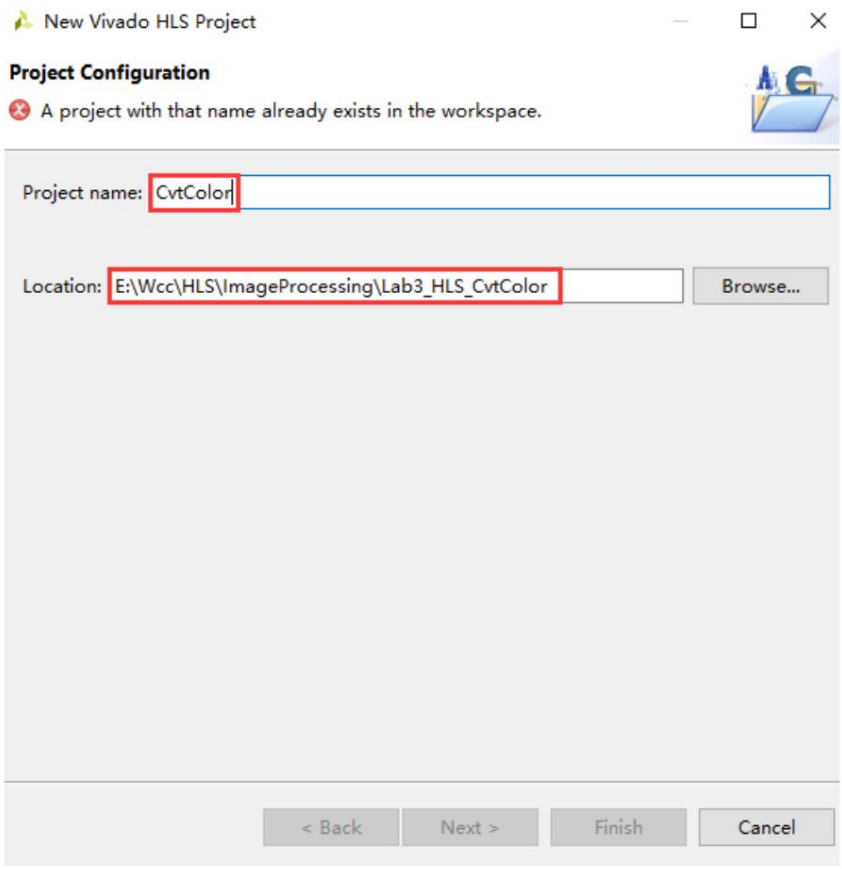


Figure 3-6 Select path and create project

Select the corresponding components: Users who purchase Z7-Lite7020 select the components on our development board xc7z020clg400-2, users who purchase Z7-Lite7010 select the device on our development board xc7z010clg400-1

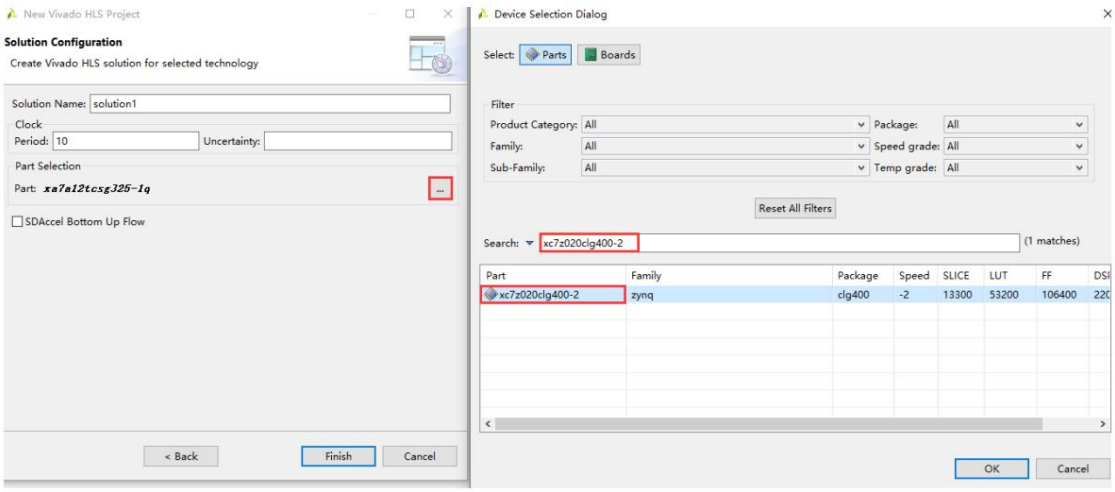


Figure 3-7 Select components

3.4.2 Add files to the project

Right click Source to create a new source file and header file

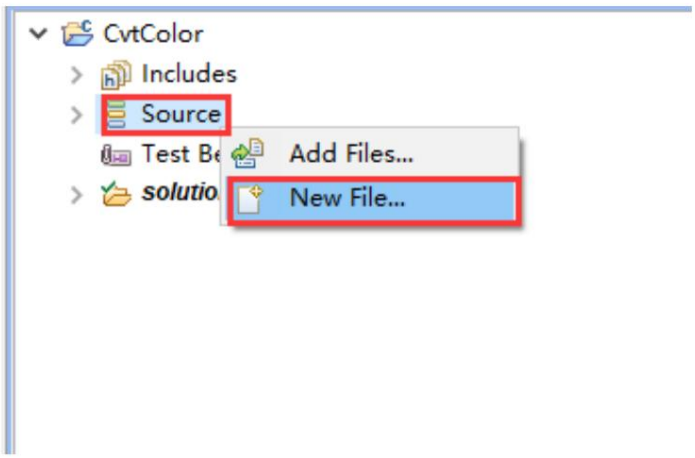


Figure 3- 8

First, add the header file, which defines the AXI-stream stream format, image matrix, and declaration

A synthesizable function.

```
1 #ifndef __CVT_COLOR_HEAD_H_
2 #define __CVT_COLOR_HEAD_H_
3
4 #include "hls_video.h"
5 #include "ap_axi_sdata.h"
6
7 //  Defines the size of the image
8 #define MAX_COLS 1920
9 #define MAX_ROWS 1080
10
11 //  Define input and output images
12 #define SRC_IMAGE "SrcImage.bmp"//13 #define  Input image path
DST_IMAGE "DstImage.bmp"//14 #define GOLD_IMAGE  Output image path
"GoldImage.bmp"//                                  Reference Image Path
15
16 //  Define image data type
17
18 //  definitionAXI-Stream  Data Flow
```

```
19 typedef hls::stream<ap_axiu<32,1,1,1> > AXI_STREAM ;
20
21 //22Define an image matrix, where the pixel format in the image matrix is3 aisle 8 bit unsigned number, RGB image
typedef hls::Mat<MAX_ROWS,MAX_COLS,HLS_8UC3> IMAGE_RGB;
23
24 //25Defines the format of each pixel of the image matrix, 3 aisle 8 Bit unsigned
typedef hls::Scalar<3,unsigned char> PIXEL_RGB;
26
27 //28Define an image matrix, where the pixel format in the image matrix is1 aisle 8 bit unsigned number, grayscale image
typedef hls::Mat<MAX_ROWS,MAX_COLS,HLS_8UC1> IMAGE_GRAY;
29
30 //31Defines the format of each pixel of the image matrix, 1 Channel8bit unsigned
typedef hls::Scalar<1,unsigned char> PIXEL_GRAY;
32
33
34 //top function
35 void image_filter(AXI_STREAM & inStream,AXI_STREAM & outStream, int rows,int cols);
36 #endif
```

Lines 1 and 2 prevent repeated calls to header files; lines 4 and 5 reference the HLS image processing library and definition

The header file of the stream structure; lines 8 and 9 define the size of the image; lines 11 to 14 define the input

Output image and golden reference image path; 19 defines the type of AXI-Stream; 21~30 lines re-define

It defines RGB image matrix, grayscale image matrix, RGB image pixel, grayscale image pixel; line 35 declares

The top-level function.

In the source file, the implementation of the declared function is defined, and the specific code is shown in the following table:

```
1 #include "ChangeColor.h"
2
3 void image_filter(AXI_STREAM & inStream,AXI_STREAM & outStream, int rows,int cols){
4 #pragma HLS INTERFACE s_axilite port=cols bundle=CTRL_BUS
5 #pragma HLS INTERFACE s_axilite port=rows bundle=CTRL_BUS
6 #pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
7 #pragma HLS INTERFACE ap_stable port=rows
8 #pragma HLS INTERFACE ap_stable port=cols
9 #pragma HLS INTERFACE axis register port=outStream
10 #pragma HLS INTERFACE axis register both port=inStream
11 IMAGE_RGB img_0(rows,cols);
12 IMAGE_GRAY img_1(rows,cols);
```

```
13 IMAGE_RGB img_2(rows,cols);
14 #pragma HLS dataflow
15 hls::AXIvideo2Mat(inStream, img_0);
16 hls::CvtColor<HLS_BGR2GRAY>(img_0,img_1);
17 hls::CvtColor<HLS_GRAY2BGR>(img_1,img_2);
18 hls::Mat2AXIvideo(img_2,outStream);
19 }
```

The first line imports the header file we just created, and lines 3 to 19 implement the functions we declared.

Lines 4 to 10 are the added interface (INTERFACE) type directives, which have been introduced in the previous chapter.

In this chapter, we will introduce how to add the above Directives; 11 to 13

The line defines three variables, where img_0 and img_2 are RGB image matrices, and img_1 is a GRAY

The functions of lines 15 and 18 are similar, and their purpose is to perform the conversion between the image matrix and the AXI-stream.

The conversion between the two is the synthesizable function in hls that can process the data transmitted. Lines 16 and 17 complete

It is the conversion of image color. The image conversion is performed through hls::CvtClolor. The definition and usage of the function can be

Refer to the introduction in UG902, which mainly converts one type of image into another type of image.

Next, we will introduce how to add these Directives. First, we will add the AXI-Stream type interface.

Directive, we already know that the input and output images are of AXI-Stream type. In the Directive window, select

In the function interface, right-click and select Add Directive. Here we take inStream as an example:

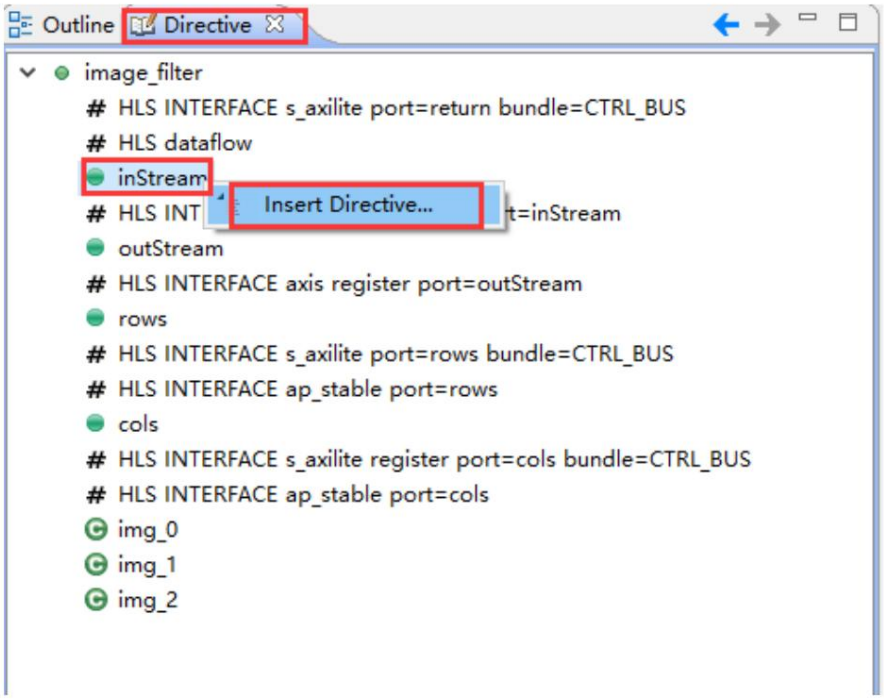


Figure 3- 9

In the pop-up interface, select the interface type according to the Directive type, add it to the source file, and then click Mode.

Click on the axis (i.e. AXI4-stream) and click OK. You can see that AXI-

Interface constraints for Stream types.

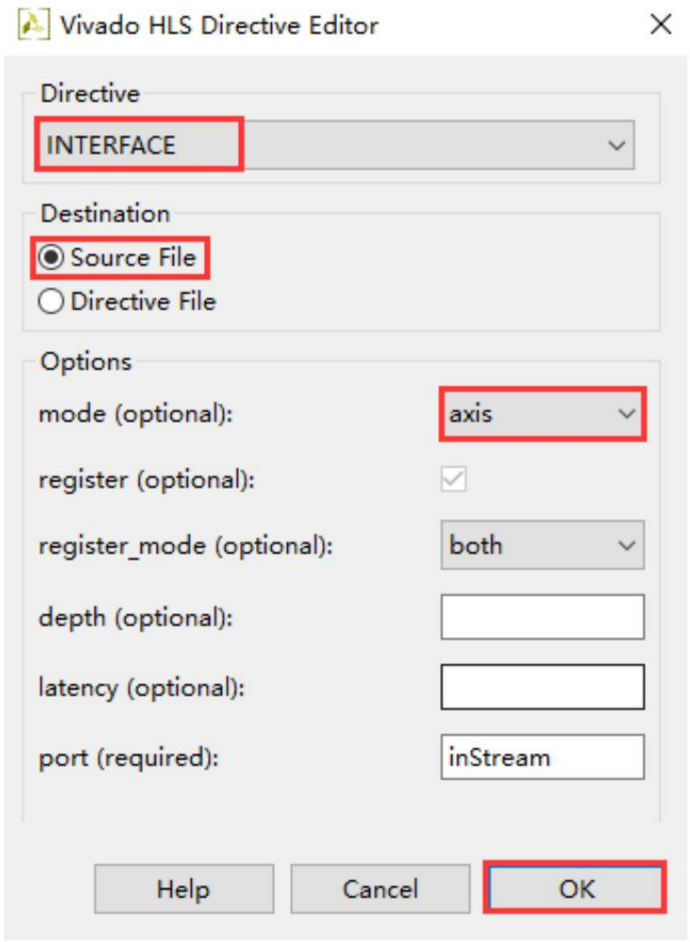


Figure 3- 10 Adding a Directive

Then constrain the basic types. These variables are generally used as control signals. In HLS, they are often used as

The s_axilite interface supports the AXI-Lite protocol and can transmit a relatively large amount of data each time.

Small, field is used to control the module. We follow the method introduced earlier and select the int type variable to add

In Directive, select s_axilite in Option and fill in CTRL_BUS in bundle to indicate that the signal is a

control signal. For the input rows and cols. It is also hoped that it remains stable when the module is working, so

We also need to add constraints to ap_stable to keep it stable.

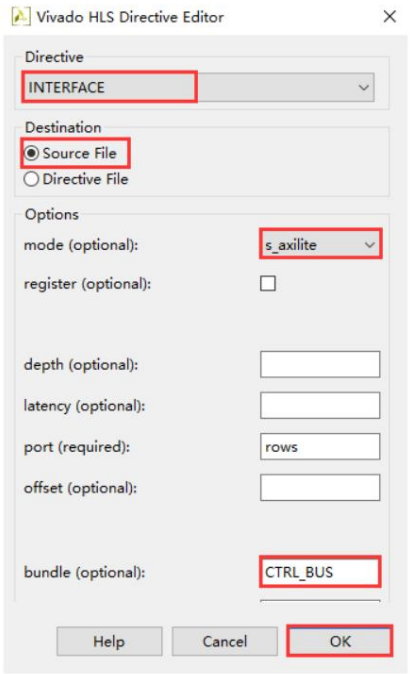


Figure 3- 11

As you can see, there is a directive for return in the interface, but the function is of type void.

There is no return value. What is the purpose of adding the return directive here? This is because

After adding the s_axilite Directive, the SDK can provide configuration information and control for the generated IP.

Signal. Just like the need to initialize peripherals during the SDK development phase, add a Directive to the return

After that, the SDK will provide drivers for these initialization and configuration signals, making it easier for us to develop in the SDK.

How to add Return Directive:

Finally, you can see a Directive #pragma HLS dataflow. This Directive is not

It is an interface type directive, which is used to optimize functions. It adds statements.

The functions inside the data will be executed in parallel, reducing latency and increasing running speed. This Directive is added as follows:

Select the function, right-click and select Directive as DATAFLOW.

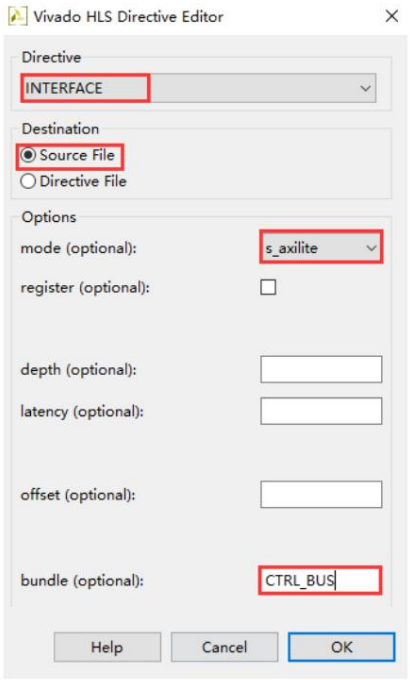


Figure 3- 12

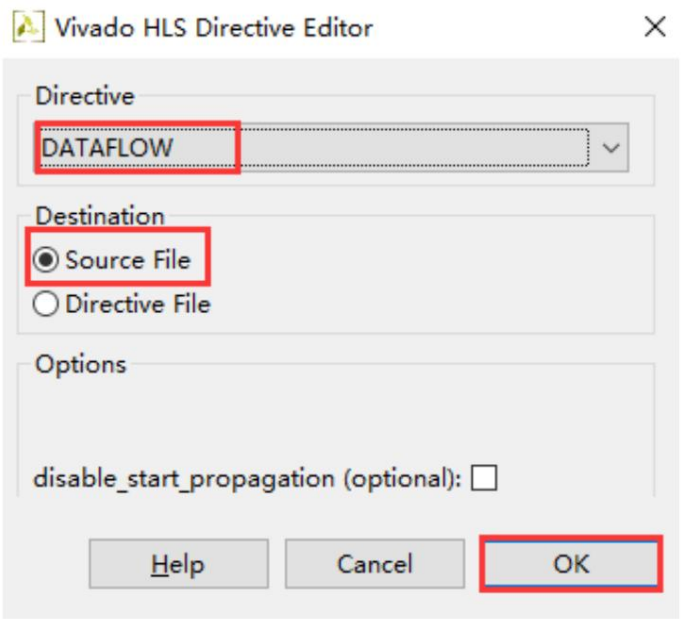


Figure 3- 13 Adding DataFlow

3.4.3 Add simulation file

Next, add a simulation file. Right-click on the Test Bench column to create a new file:

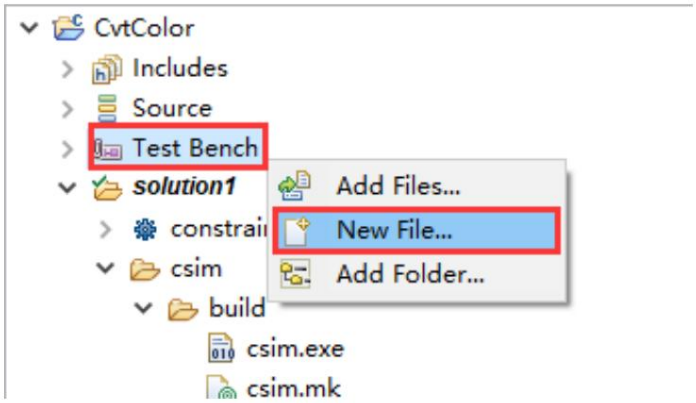


Figure 3- 14 Add simulation file

The contents of the simulation file are as follows:

```
1 #include <stdio.h>
2 #include <opencv2/opencv.hpp>
3 #include "ChangeColor.h"
4 #include "hls_opencv.h"
5 using namespace cv ;
6
7
8 //      Image comparison
9 int image_compare(const char* output_image, const char* golden_image) {
10          if (!(output_image) || !(golden_image)) {
11                  printf("Failed to open images...exiting.\n");
12                  return -1;
13          } else {
14                  Mat o = imread(output_image);
15                  Mat g = imread(golden_image);
16                  assert(o.rows == g.rows && o.cols == g.cols);// assert                  If it is correct, continue to run the following program, otherwise
Report an error
17                  assert(o.channels() == g.channels() && o.depth() == g.depth());
18                      printf("rows = %d, cols = %d, channels = %d, depth = %d\n", o.rows, o.cols, o.channels(),
o.depth());
19                  int flag = 0;
20                  for (int i = 0; i < o.rows && flag == 0; i++) {
21                          for (int j = 0; j < o.cols && flag == 0; j++) {
22                                  for (int k = 0; k < o.channels(); k++) {
23                                          unsigned char p_o = (unsigned char)*(o.data + o.step[0]*i + o.step[1]*j + k);
24                                          unsigned char p_g = (unsigned char)*(g.data + g.step[0]*i + g.step[1]*j + k);
25                                          if (p_o != p_g) {
```

```
26                                      printf("First mismatch found at row = %d, col = %d\n", i, j);
27                                      printf("(channel%2d) output:%5d, golden:%5d\n", k, p_o, p_g);
28                                      flag = 1;
29                                      break;
30                                  }
31                              }
32                          }
33                  }
34                  if (flag)
35                          printf("Test Failed!\n");
36                  else
37                          printf("Test Passed!\n");
38
39                  return flag;
40          }
41 }
42
43 int main(){
44 IplImage          * SrcImage;//       Input image
45 IplImage          * DstImage;//       Output image
46 IplImage          * GoldImage;//      Reference Images
47 IplImage          * GoldImageTmp;//   Reference Images
48 SrcImage = cvLoadImage(SRC_IMAGE,-1);//                Importing images
49
50 //      create opencv    Supported output images
51 DstImage = cvCreateImage(cvGetSize(SrcImage),SrcImage->depth, SrcImage->nChannels);
52 GoldImage = cvCreateImage(cvGetSize(SrcImage),SrcImage->depth, SrcImage->nChannels);
53 GoldImageTmp = cvCreateImage(cvGetSize(SrcImage),SrcImage->depth, 1);
54 //AXI-Stream              Data Flow
55 AXI_STREAM inStream;
56 AXI_STREAM outStream;
57 IplImage2AXIvideo(SrcImage,inStream);
58
59 //      Synthesizable functions
60 image_filter(inStream,outStream,SrcImage->height, SrcImage->width);
61 //      Save the output
62 AXIvideo2IplImage(outStream,DstImage);
63 cvSaveImage(DST_IMAGE,DstImage);
64
65 //      Reference image output
66 cvCvtColor(SrcImage, GoldImageTmp, CV_BGR2GRAY);
```

```
67 cvCvtColor(GoldImageTmp, GoldImage, CV_GRAY2BGR);
68 cvReleaseImage(&GoldImageTmp);
69 cvSaveImage(GOLD_IMAGE, GoldImage);
70 cvShowImage(DST_IMAGE, DstImage);
71 cvWaitKey(0);
72
73 //      Compare the golden reference image and HLS Comprehensive execution results
74 image_compare(DST_IMAGE,GOLD_IMAGE);
75 //      Release pictures
76 cvReleaseImage(&SrcImage);
77 cvReleaseImage(&DstImage);
78 cvReleaseImage(&GoldImage);
79 return 0;
80 }
```

Lines 1 to 4 import the required header files, and line 5 uses the cv namespace. Lines 9 to 41 define

A function is used to compare the image output by using HLS and the image output by using OpenCv

Whether the generated golden reference image is completely consistent.

Lines 43 to 80 are the main function of this policy. You can see the simulation process in the Testbench.

According to the class content in the figure, first import the image from the outside world. The imported image is of the IplImage type:

Line 48; then convert the IplImage type data into the AXI-stream format supported by the HLS synthesizable module.

The data in the format is: 57 lines; then the top-level function of the synthesizable module is called and the data flow is given: 60 lines, the function call

After use, convert the AXI-Stream format data into IplImage format and save the generated image

Save: 62~63 lines; In addition to calling the synthesizable module for verification, the function in OpenCV is also used to generate

A golden reference image is generated: lines 66-69; comparing the golden reference image with the image generated by the synthesizable module,

If there are any differences, print out the first pixel position where the difference occurs. Finally, release the occupied memory.

live.

3.4.4 Simulation Verification

Select an image and name it SrcImage.bmp and save it in the same file path as the source code.

In the middle of the room.

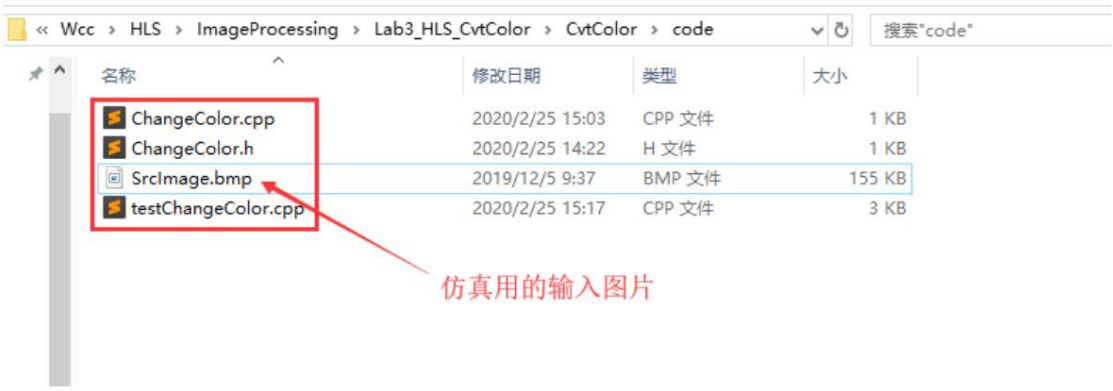


Figure 3- 15

Click C Simulation to start the simulation. After the simulation is finished, you can see the csimÿbulid folder.

Two images are generated, which can be opened and viewed.

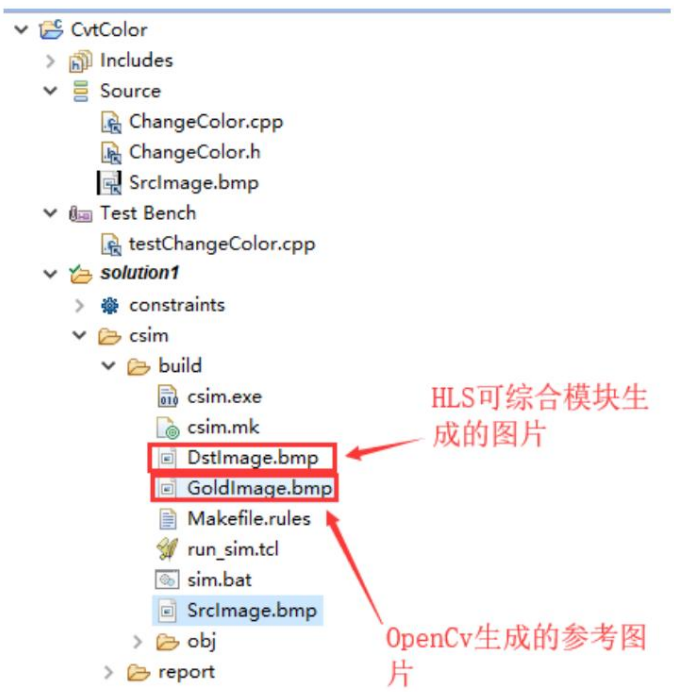


Figure 3- 16

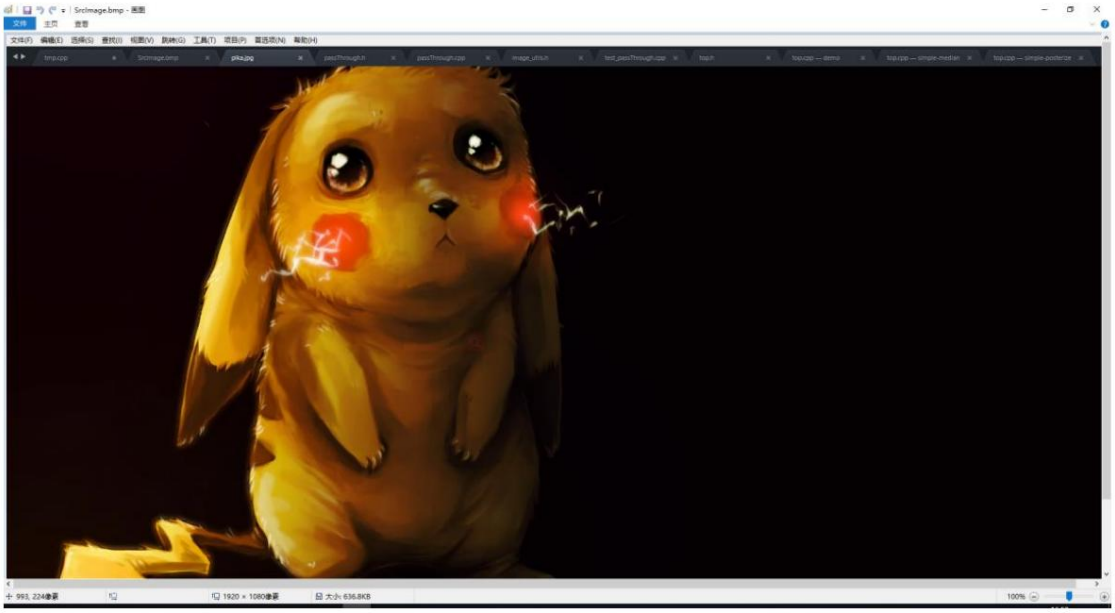


Figure 3-17 Pikachu original image

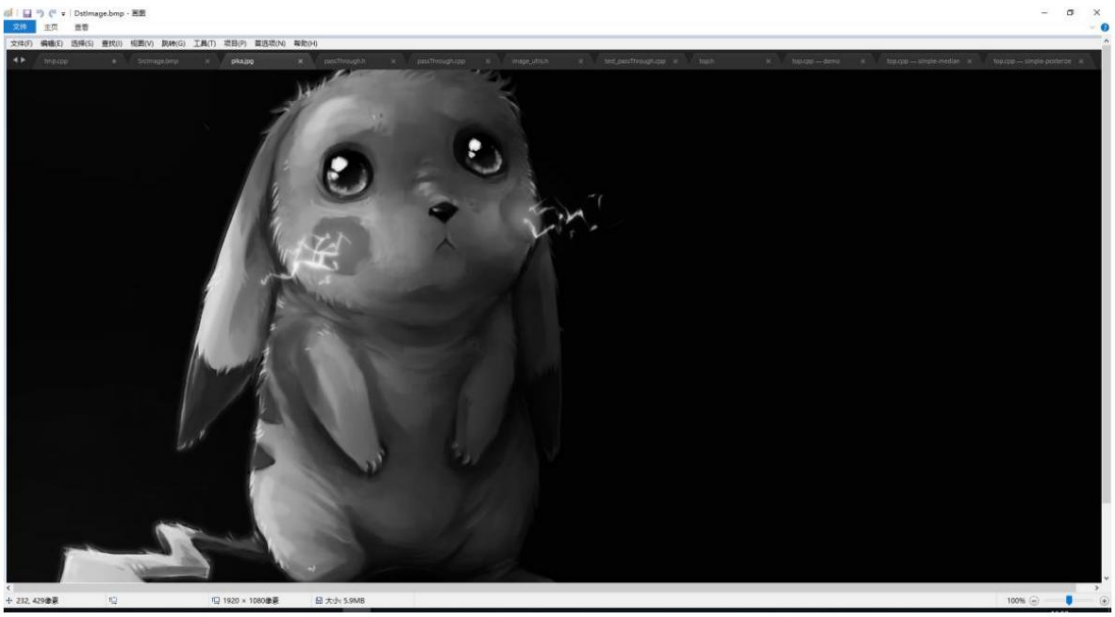


Figure 3-18 Image generated by HLS synthesizable module

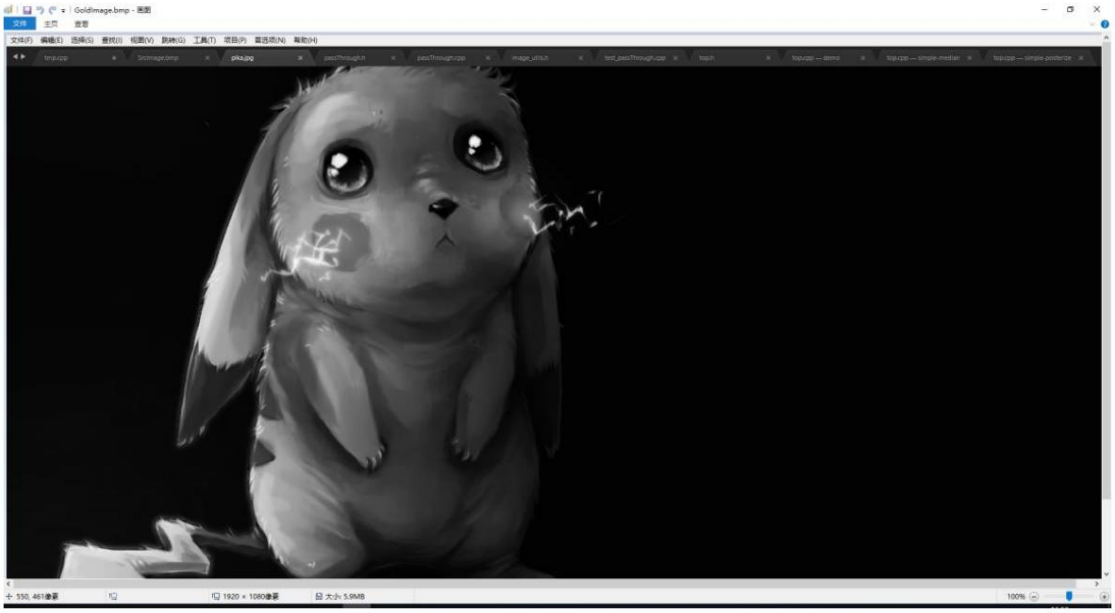


Figure 3-19 Golden reference picture

By observing the two generated pictures, we can see that there is almost no difference between them, but observing the Console

The printed information shows that the two pictures are not the same:

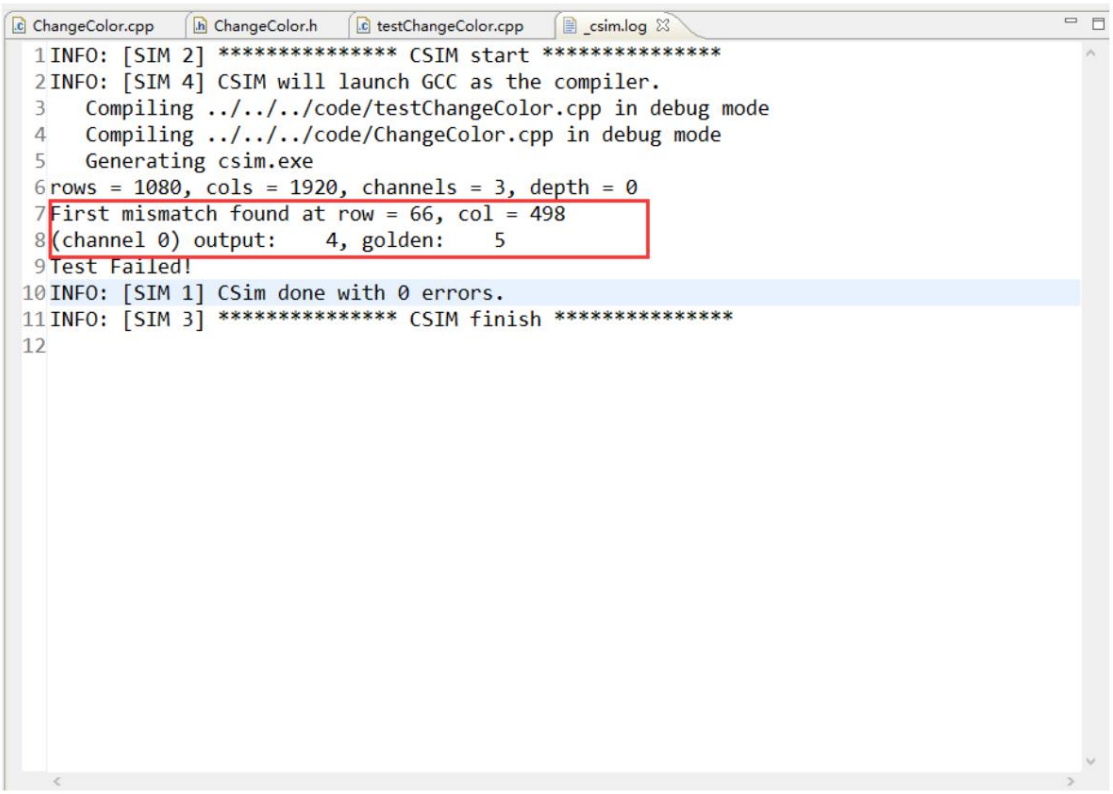


Figure 3- 20

This is because the image processing functions of the synthesizable modules supported by HLS are different from those of OpenCv

The function in the library has inconsistent precision. Through beyond compare, we can observe that the two images

The observation report shows that there are only 35 differences between the pixels.

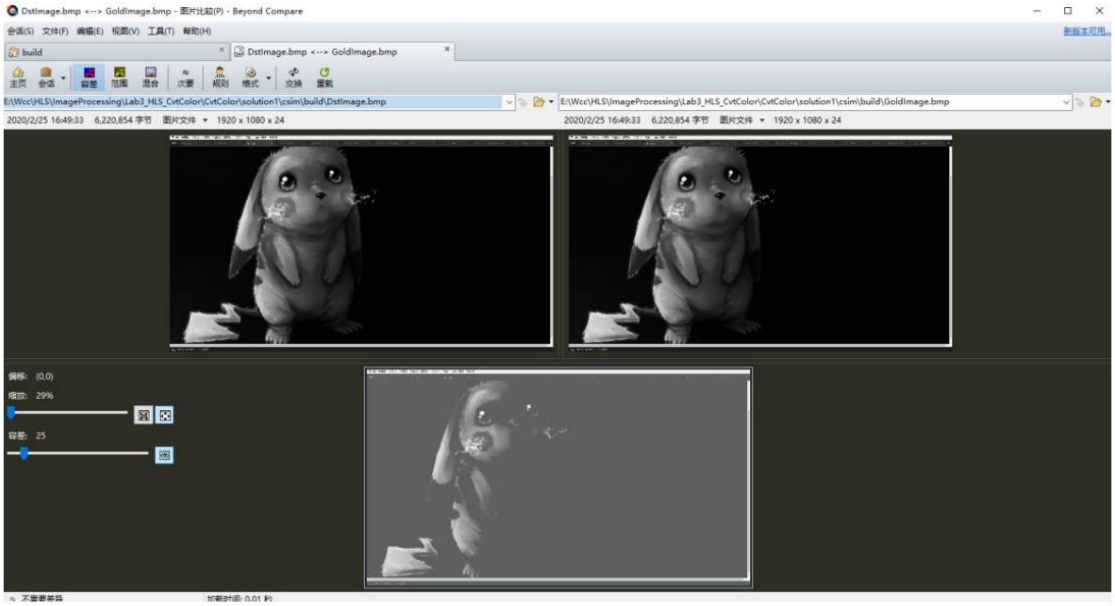


Figure 3-21 Images compared using beyond compare

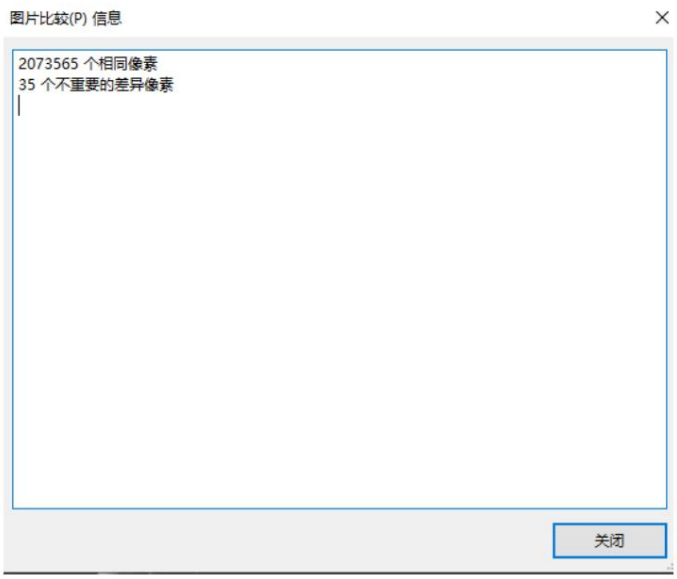


Figure 3- 22 Comparison results

3.5 Experimental Summary

This experiment mainly introduces the process of developing image processing algorithms in HLS, the basic functions in OpenCV

Through this operation, a verification platform for image algorithm processing has been built.

It will be used many times in the following chapters.

In this chapter, it is important to note the difference between the synthesizable HLS code and the code used in simulation.

The path is the same as the simulation in FPGA development. After completing this chapter of experiments, the most important thing is to master a

As shown in the error! Reference source not found, follow the standard route to perform the algorithm

development.

Chapter 4 Floating Point Arithmetic and **BRAM** Interface in **HLS**

Fixed-point FPGA implementations are faster and more efficient than floating-point algorithms, but floating-point algorithms are often needed.

This is because fixed-point numbers have a limited data dynamic range and require in-depth analysis to determine the integer

In order to achieve the optimal QoR, many different types of intermediate data width change patterns need to be introduced.

Fixed-point numbers are used as intermediate variables, while floating-point numbers have a larger dynamic data range, so in many algorithms

Only one data type is required.

In this experiment, we call the math functions of the library to demonstrate how to perform floating-point operations in HLS.

The generated IP is designed as a BRAM interface, generating an IP that can be used for floating-point operations.

Called in Vivado.

Complete the HLS operation of the following formula and generate the BRAM interface.

The generated IP core is verified in the SDK. This formula has no practical significance and is only used for demonstration.

How to use the math library and floating point operations in HLS as a hint.

$$f(x,y) = sinx + \sqrt{exp(y)}/2$$

4.1 Implementation of floating point arithmetic core in HLS

4.1.1 Add source files

After creating the HLS project, create a new file to store the source file. Add the following content to the source file:

Allow:

```
1 #include <hls_math.h>
2
3 void crazyFunction(float x[100],float y,float result[100]){
4 #pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS 5
#pragma HLS INTERFACE s_axilite port=y bundle=CRTL_BUS 6
#pragma HLS INTERFACE bram port=x
7 #pragma HLS INTERFACE bram port=result
```

```
8 int idx;
9 for(idx=0;idx<100;idx++){
10 #pragma HLS PIPELINE
11          result[idx] = (hls::sin(x[idx]) + hls::sqrtf(hls::expf(y)))/2;
12 }
13 }
```

The first line introduces the math function library supported in HLS. By introducing this library, it can be similar to C language

The third line declares the top-level function of this experiment. You can see that there are two

An array of float type and a float type data.

Lines 4 to 7 are Directives, where line 4 is for this function to provide when generating IP.

The interface function can be called in the SDK and the driver function of the IP is generated.

Line 5 declares the variable Y as an AXI_Lite interface, which can be passed in as a parameter.

Variable function.

Lines 6 and 7 declare the two arrays as BRAM interfaces, which are connected to the BRAM in Vivado

In the integrated environment, it can be used to connect to BRAM.

Line 10 is the directive for adding pipelines, which can optimize the for loop and reduce latency.

Line 11 calls the library function to complete the main function of this experimental function.

Let's use one of the directives to demonstrate how to add a BRAM interface type directive:

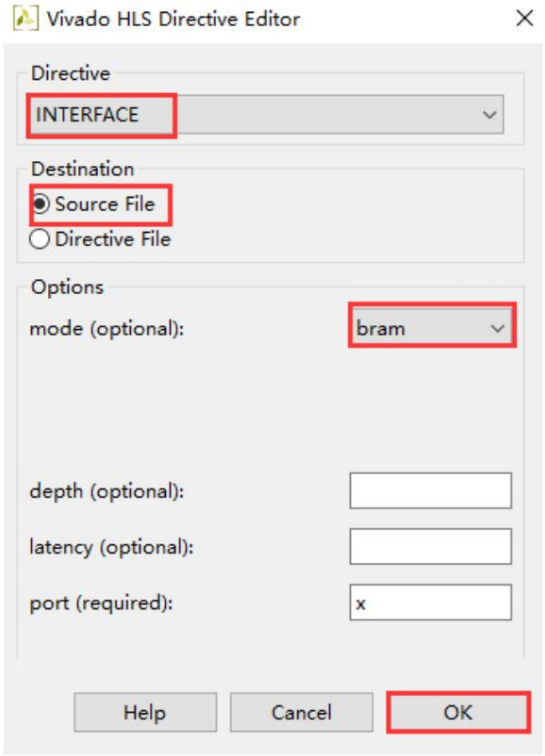


Figure 4- 23

4.1.2 Add C simulation file

Next, add a C simulation file to verify that the top-level function works correctly.

```
1 #include <stdio.h>
2 void crazyFunction(float x[100],float y,float result[100]); 3 int main(){ 4
float xVect[100];
int idx ;
5
6 float y = 2.0f;
7 float result[100]; 8
for(idx=0;idx<100;idx++){
9               xVect[idx] = idx;
10 } 11
crazyFunction( xVect,y,result);
12
13 for(idx=0;idx<100;idx++)
14              { printf("result[%d]=%f\n",idx,result[idx]);
15 }
16 return 0;
17 }
```

The simulation file is also relatively simple, which is to instantiate the top-level function and give the input array to observe

Observe the results printed in the output array.

4.1.3 Running **C** Simulation

After clicking Run C simulation, you can observe that the results are output in the Console.

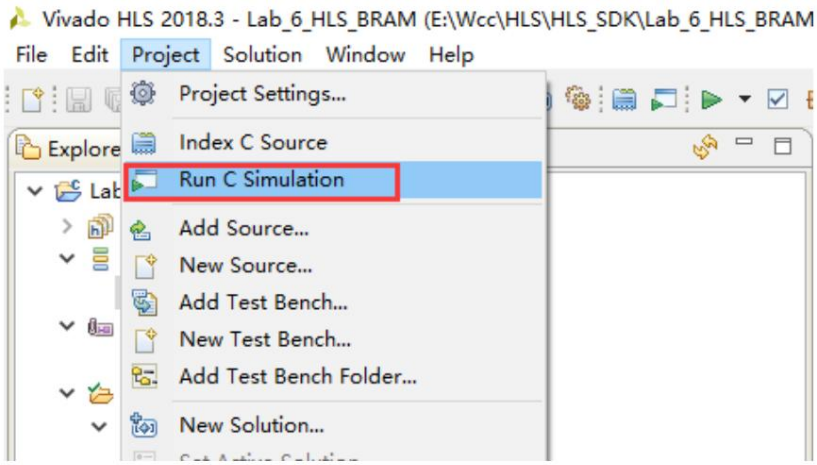


Figure 4- 24

Results of running the C simulation:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../code/test_crazyFunction.cpp in debug mode
   Generating csim.exe
result[0]=1.359141
result[1]=1.779876
result[2]=1.813790
result[3]=1.429701
result[4]=0.980740
result[5]=0.879679
result[6]=1.219433
result[7]=1.687634
result[8]=1.853820
result[9]=1.565200
result[10]=1.087130
result[11]=0.859146
```

Figure 4- 25

4.1.4 Perform **C** Synthesis

First, select the top-level function to be synthesized, select Project Settings in Project, and in the pop-up interface

Select Synthesis and then select the top-level function;

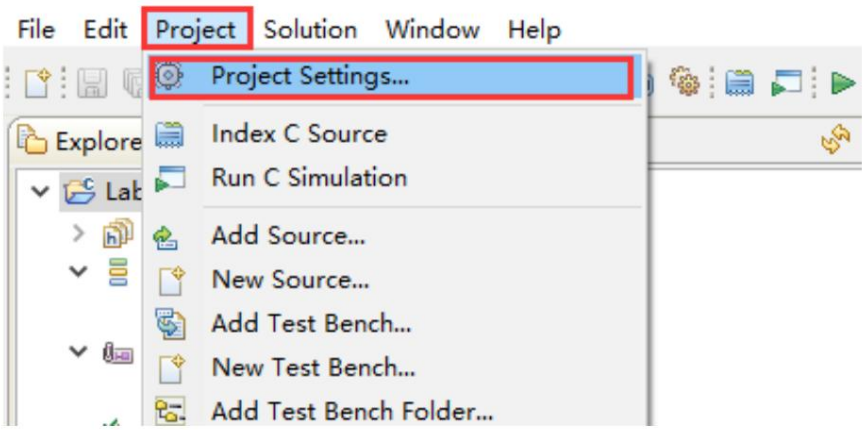


Figure 4- 26

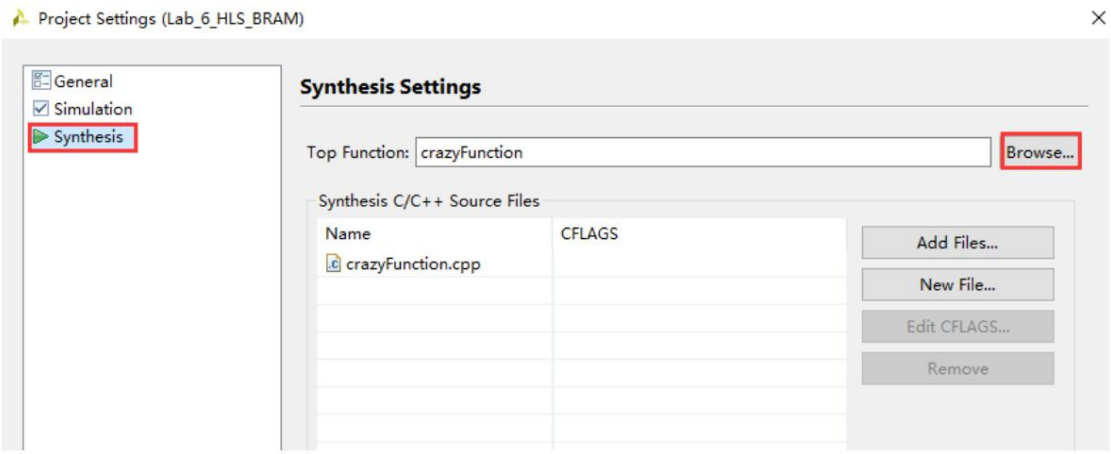


Figure 4- 27

After the settings are completed, click C to start the integration. After the integration is completed, you can view the occupied resources.

From the timing summary, we can see that the delay here is relatively small. If the user is interested, you can find the source file,

Add pipeline directive and remove it to compare the timing difference between the following two.

After the for loop is optimized, the latency can be significantly reduced.

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.682	1.25

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
126	126	126	126	none

⊟ Detail

⊞ Instance

⊞ Loop

Figure 4- 28 Timing Report

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	14
FIFO	-	-	-	-
Instance	0	25	2937	4873
Memory	-	-	-	-
Multiplexer	-	-	-	18
Register	0	-	436	96
Total	0	25	3373	5001
Available	280	220	106400	53200
Utilization (%)	0	11	3	9

⊟ Detail

Figure 4- 29 Resource usage

The main thing to focus on is the interface after synthesis:

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_CRTL_BUS_AWVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_AWREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_AWADDR	in	5	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WDATA	in	32	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_WSTRB	in	4	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARVALID	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARREADY	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_ARADDR	in	5	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RVALID	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RREADY	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RDATA	out	32	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_RRESP	out	2	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BVALID	out	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BREADY	in	1	s_axi	CRTL_BUS	scalar
s_axi_CRTL_BUS_BRESP	out	2	s_axi	CRTL_BUS	scalar

Figure 4-30AXI_Lite interface

x_Addr_A	out	32	bram	x	array
x_EN_A	out	1	bram	x	array
x_WEN_A	out	4	bram	x	array
x_Din_A	out	32	bram	x	array
x_Dout_A	in	32	bram	x	array
x_Clk_A	out	1	bram	x	array
x_Rst_A	out	1	bram	x	array
result_Addr_A	out	32	bram	result	array
result_EN_A	out	1	bram	result	array
result_WEN_A	out	4	bram	result	array
result_Din_A	out	32	bram	result	array
result_Dout_A	in	32	bram	result	array
result_Clk_A	out	1	bram	result	array
result_Rst_A	out	1	bram	result	array

Figure 4-31 BRAM interface

It can be seen that the INTERFACE type Directive added to the source file has played a role.

Data for AXI_Lite and BRAM interface types are not generated.

AXI_Lite complies with the signal format specified by the AXI bus protocol.

The corresponding AXI_Lite interface is generated for each value, which can be used to control the IP work in the future.

For the x, and result interfaces, their respective BRAM interfaces are generated, and their data can be

Stored in BRAM.

4.1.5 Perform joint simulation

After synthesis is completed, perform co-simulation and click solutionÿRun C/RTL Cosimulation

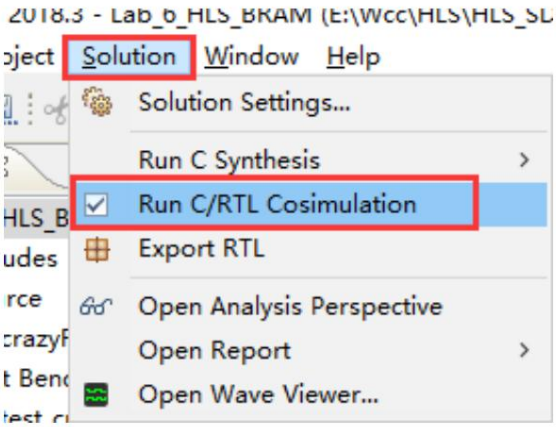


Figure 4- 32

The HLS compiler will generate RTL simulation files based on the C simulation files and perform RTL-level simulation.

The results obtained through C simulation and RTL simulation will be displayed in the Console.

4.1.6 Export **HLS IP**

After the co-simulation is completed, the IP can be exported for calling by the Vivado integrated environment.

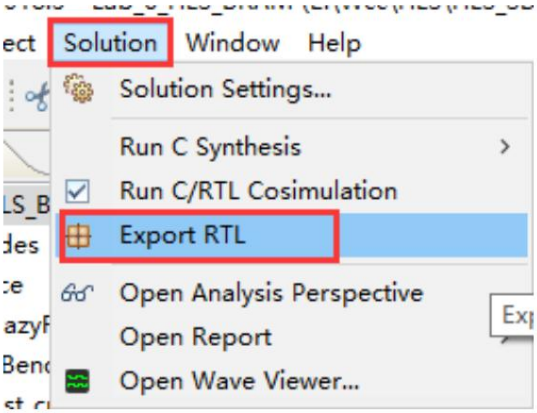


Figure 4- 33

4.2 Build the test hardware platform

4.2.1 Add **HLS IP**

In the Vivado IP repository, add the IP generated by HLS to the Vivado IP repository and click

Setting ÿ IP ÿ Repository ÿ click the plus sign, then locate the HLS solution path, Vivado

The IP will be identified.

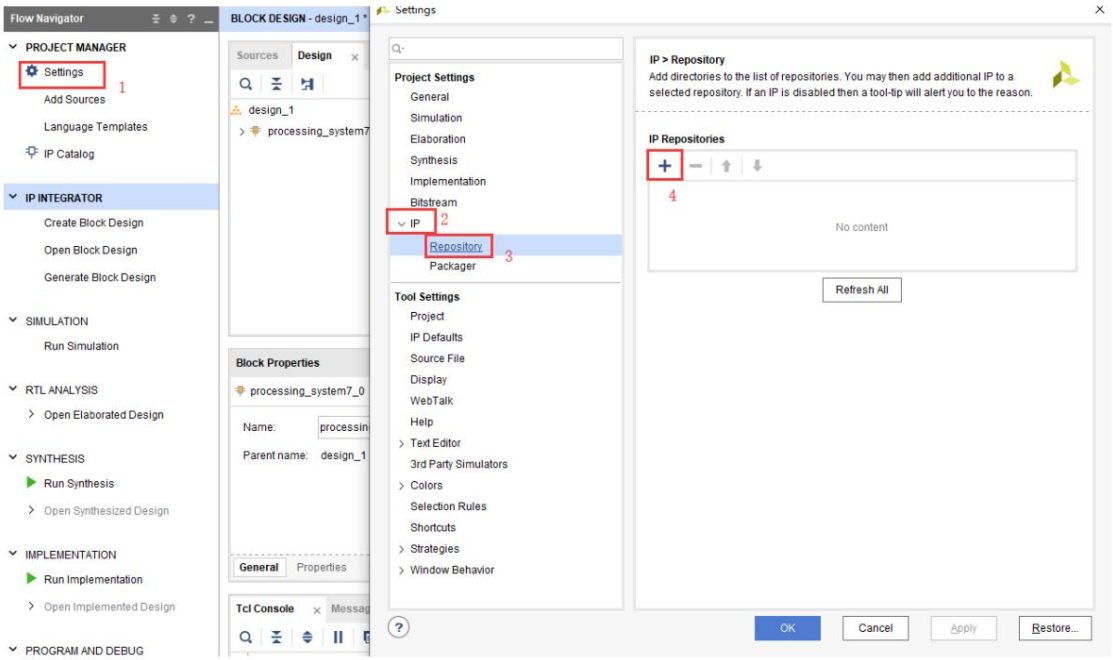


Figure 4- 34

4.2.2 Create **BlockDesign**

Click Create Block Design to build the hardware environment for this experiment.

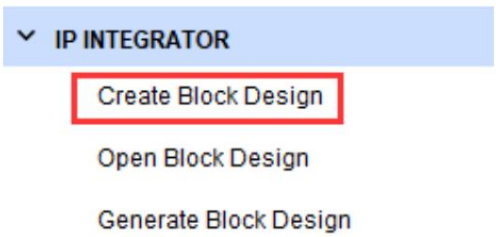


Figure 4- 35

Add ZYNQ IP core to Block design. Double-click ZYNQ to configure.

Set the voltage of Bank1 to LVCMOS 1.8V, and select SD0 and UART1.

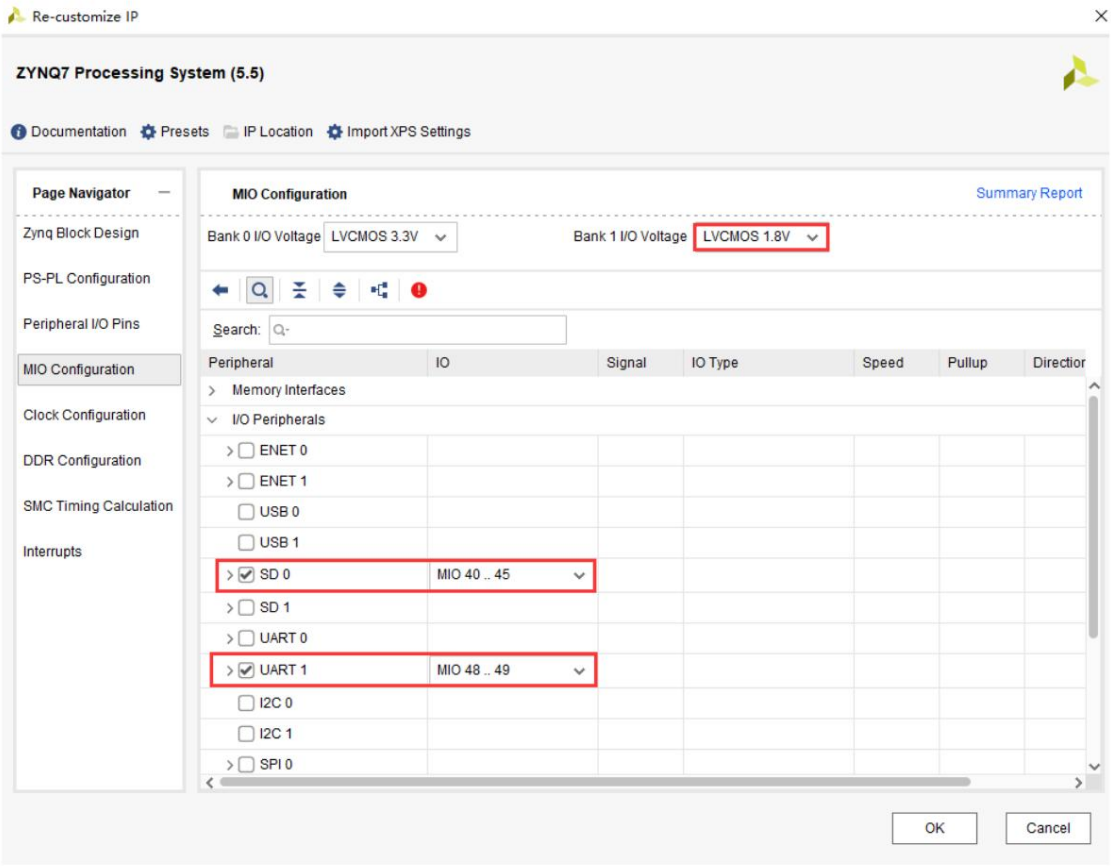


Figure 4- 36

In DDR Configuration, select the model:

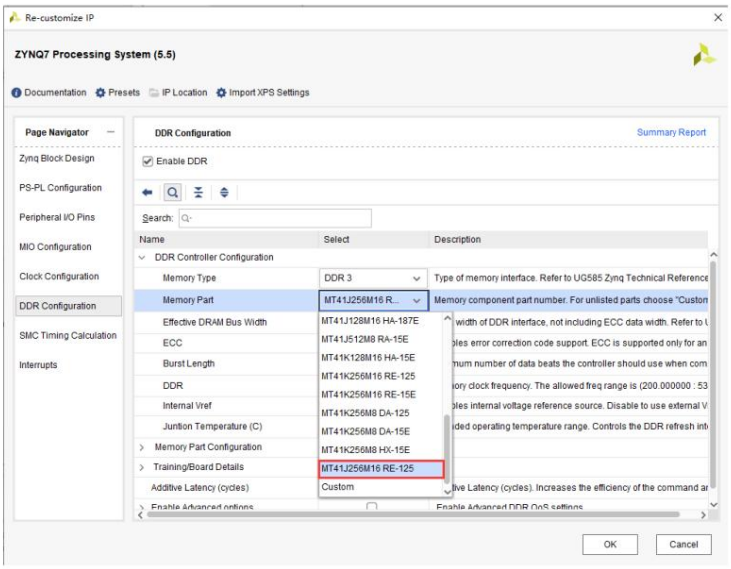
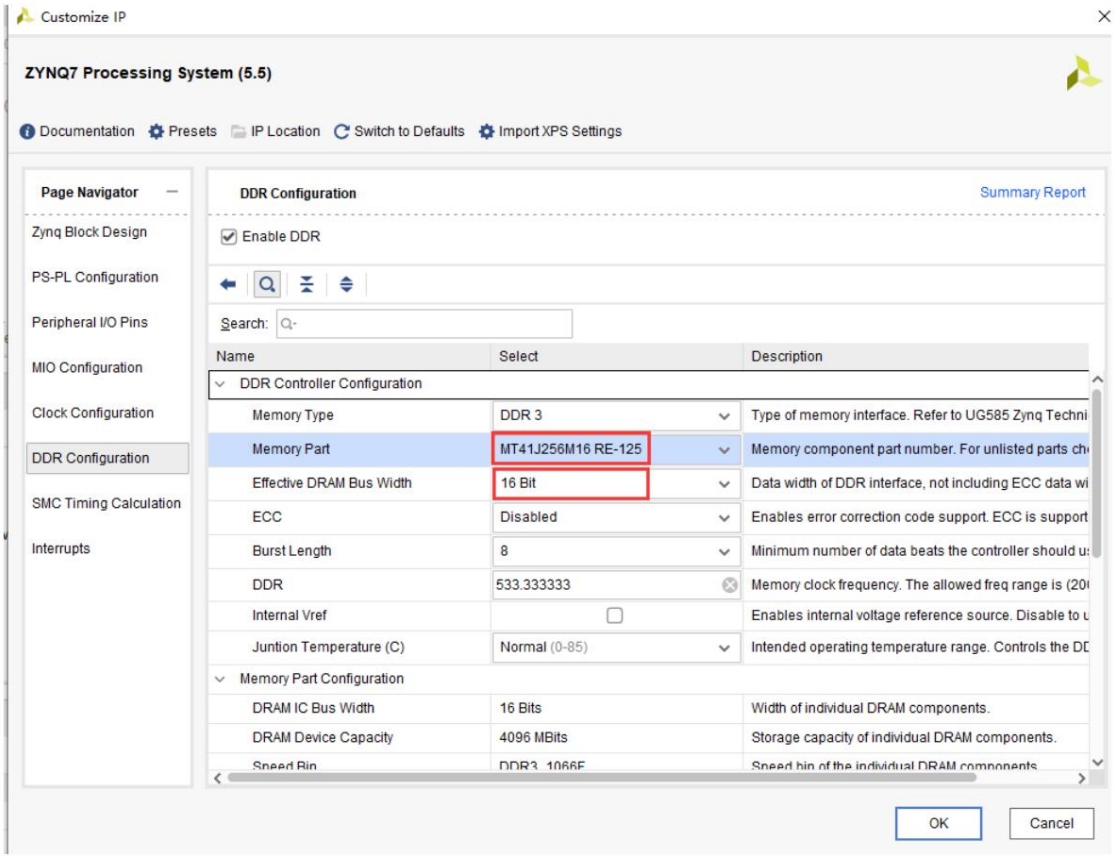


Figure 4- 37



Next, add the floating point operation IP generated by HLS and search for crazyFunction

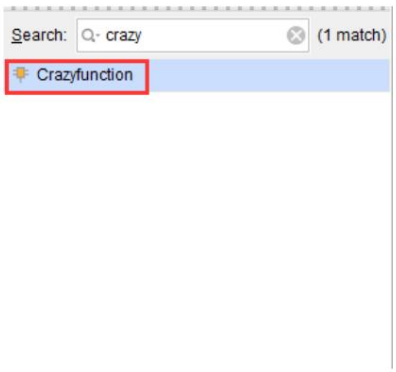


Figure 4- 38

Click Run connection Automation, first select the control bus, click Connect, and then

As shown in the figure below:

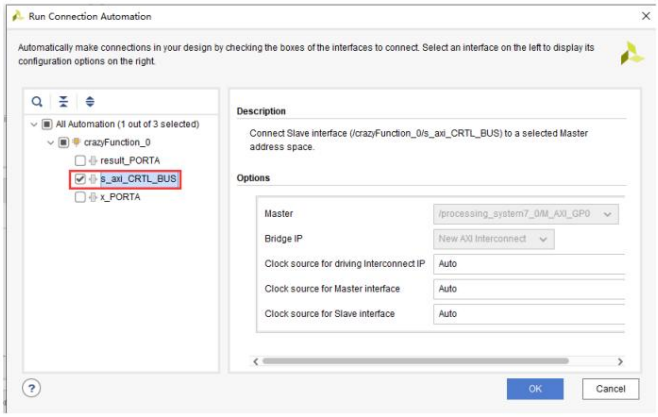


Figure 4- 39

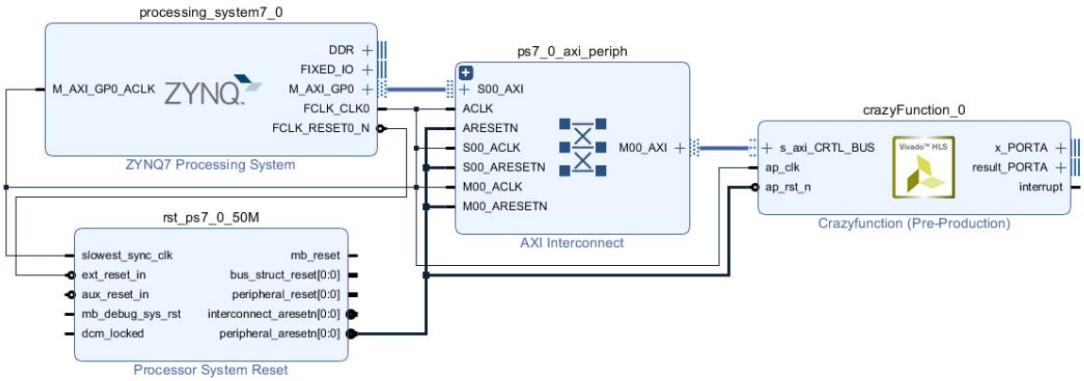


Figure 4- 40

Click Run connection Automation again and select one of the interfaces, as shown below:

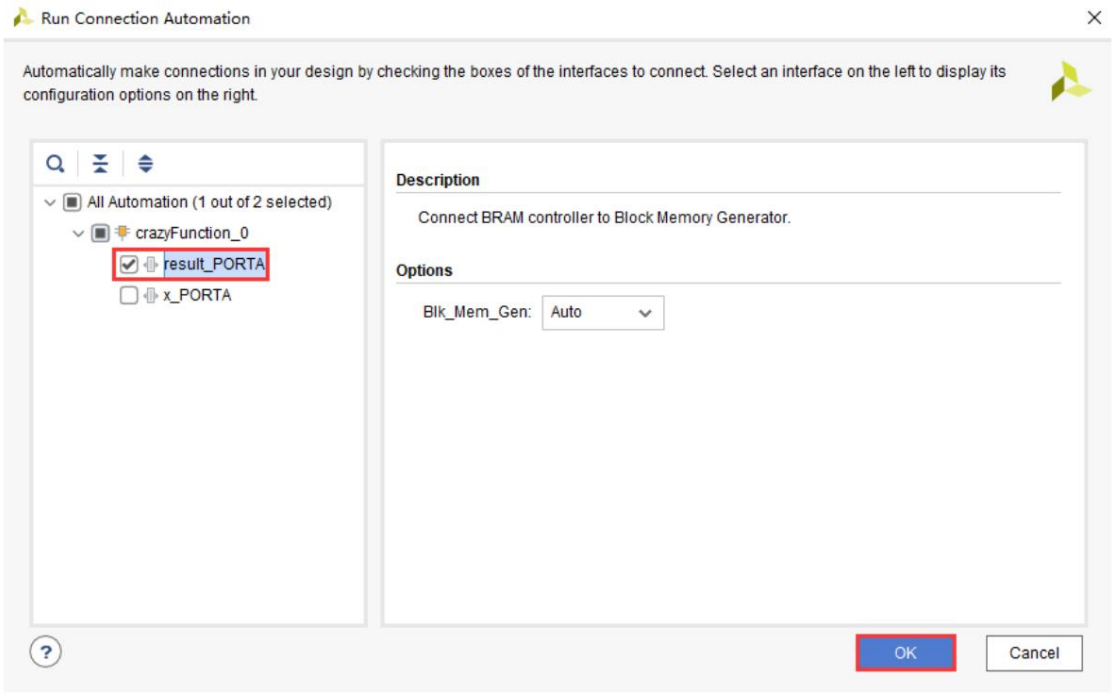


Figure 4- 41

You can see that a BRAM is automatically generated, as shown in the following figure:

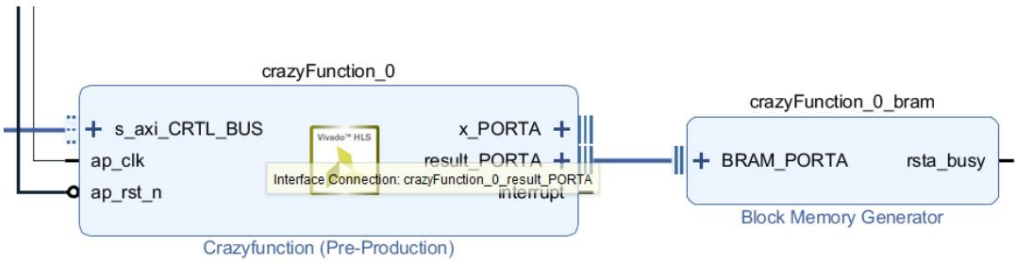


Figure 4- 42

Click on the generated BRAM to change it to a dual-port RAM, as shown below:

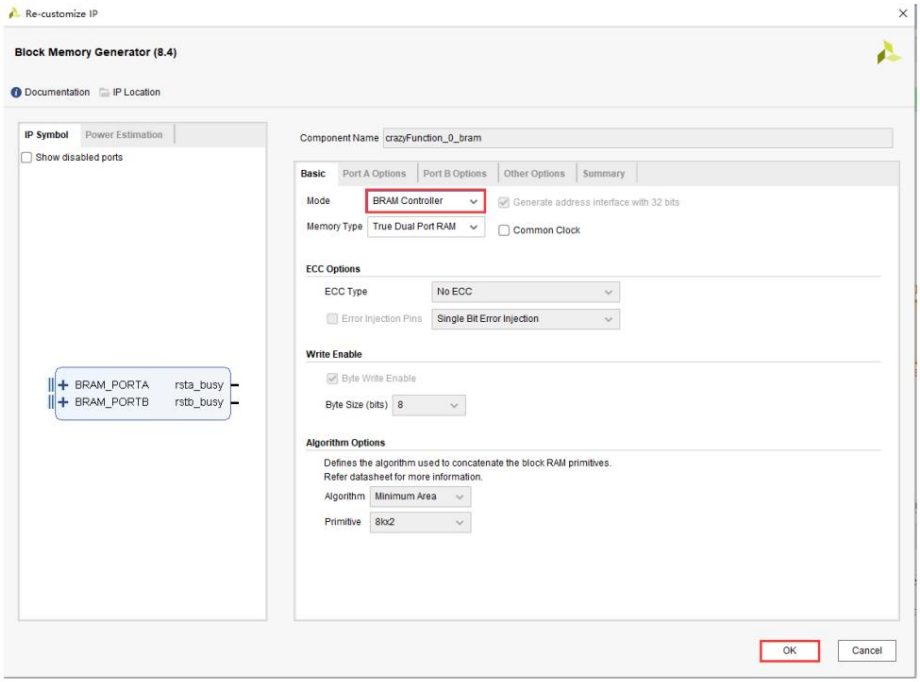
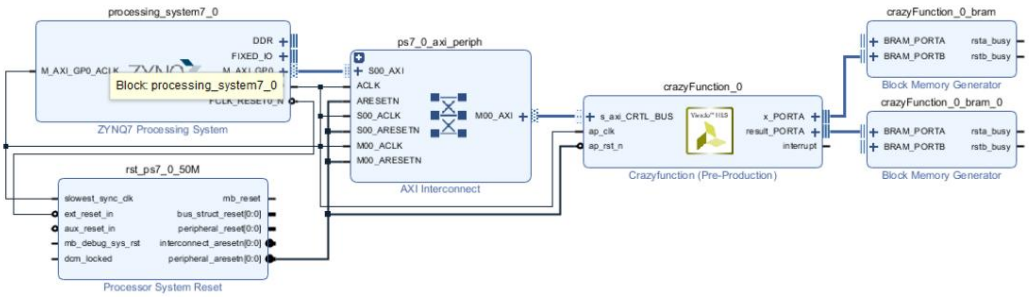


Figure 4- 43 Dual-port RAM

The same operation is used for X-port. The results are as follows:



Add axi_bram_controller, click the + sign, search for bram and you will see axi_bram_controller

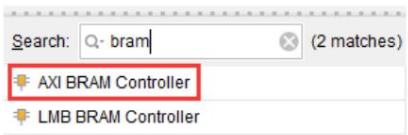


Figure 3- 44

Then set axi_bram_controller to a port

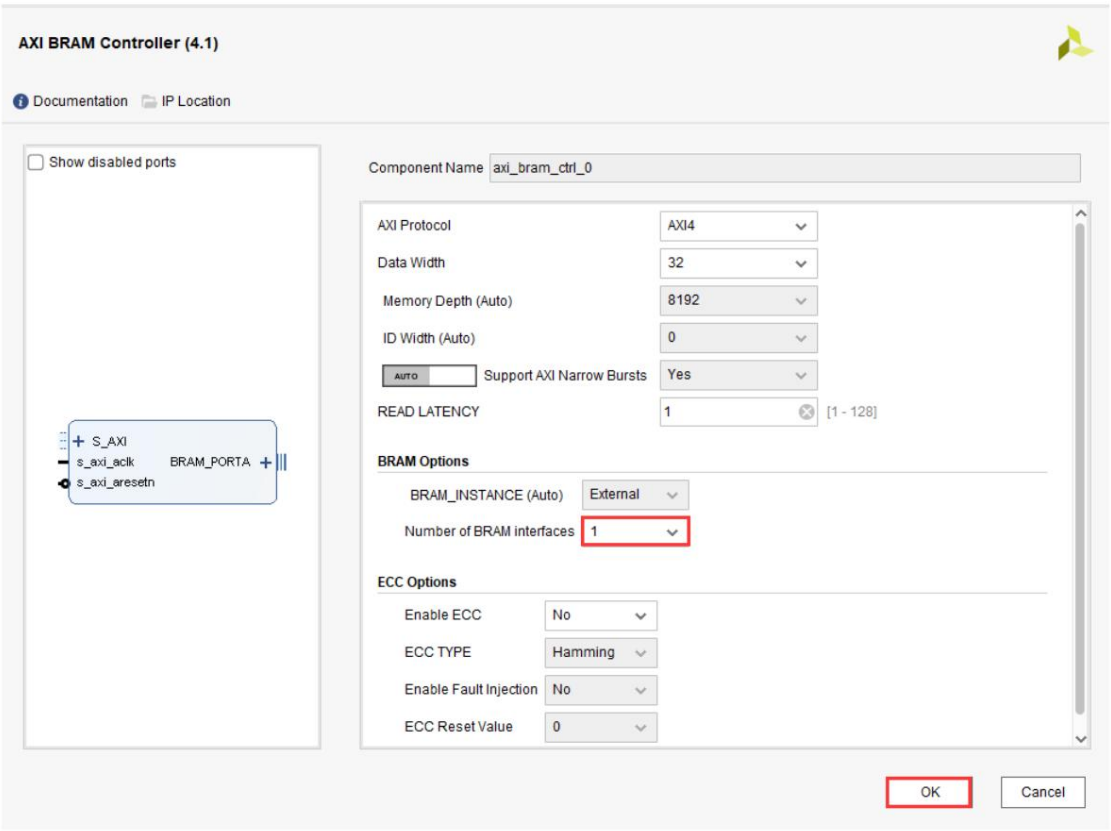


Figure 4- 45

After adding, connect as shown in the following figure:

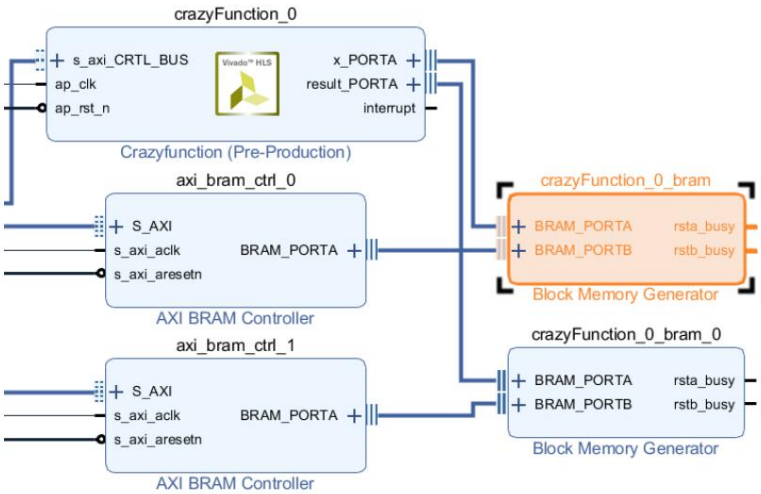


Figure 4- 46

Then click Run connection Automation and select all:

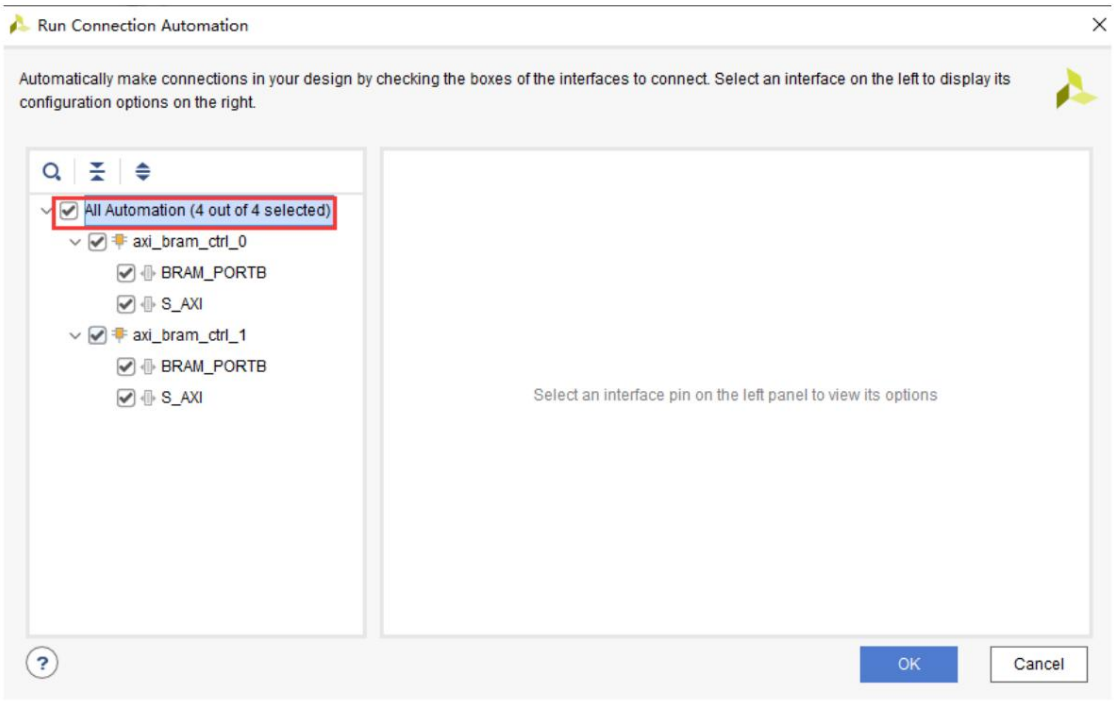


Figure 4- 47

Finally, click Run Block Automation to complete the construction of the hardware module:

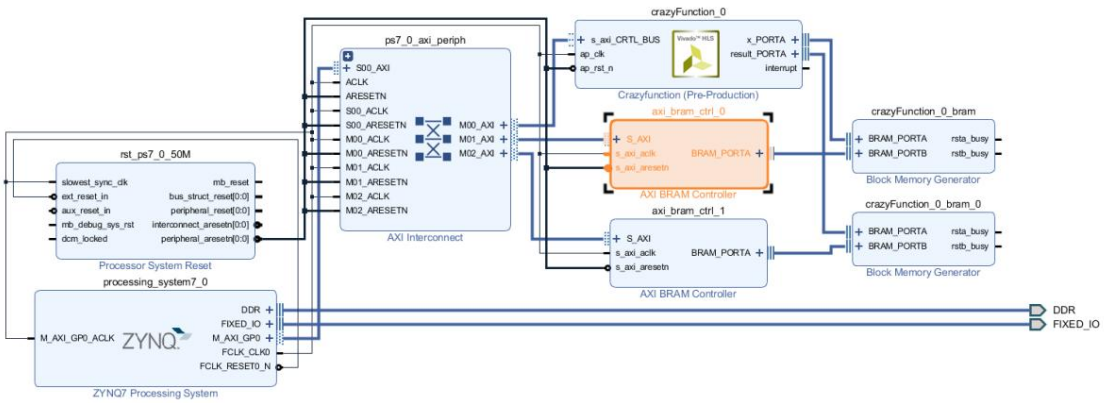


Figure 4- 48

After generating the top-level wrapper, generate the bit file;

4.3 Create **SDK** Project

After the bit file is generated, export the bit file, click export hardware ÿ check include bitstreamÿ

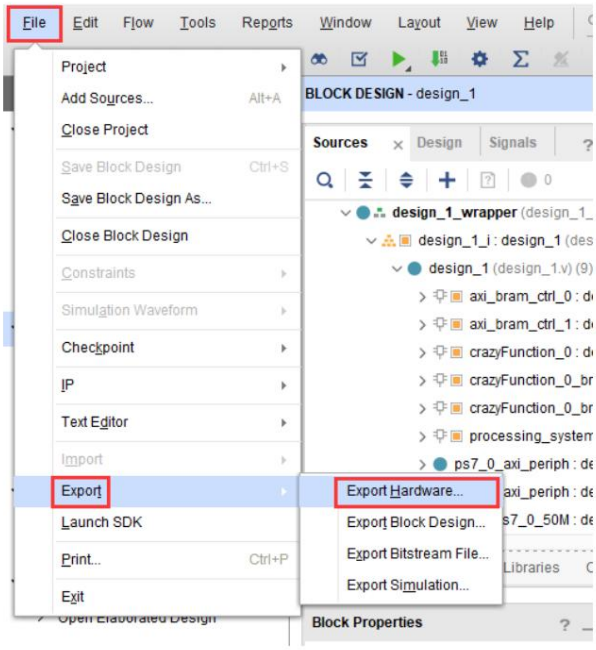


Figure 4- 49

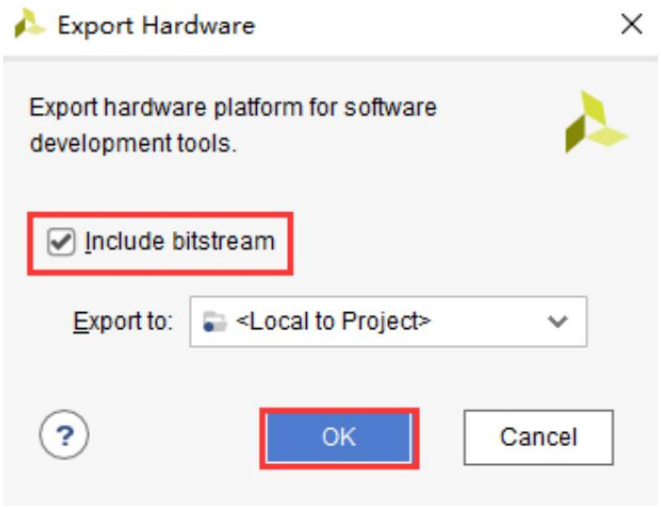


Figure 4- 50

Then click Launch SDK to open the SDK.

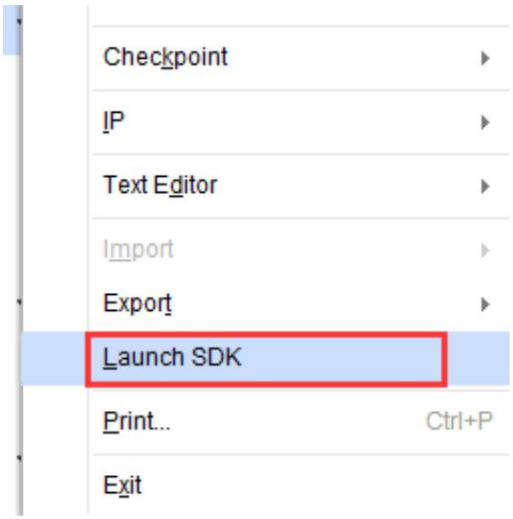


Figure 4- 51

Enter the SDK and create an empty C++ project:

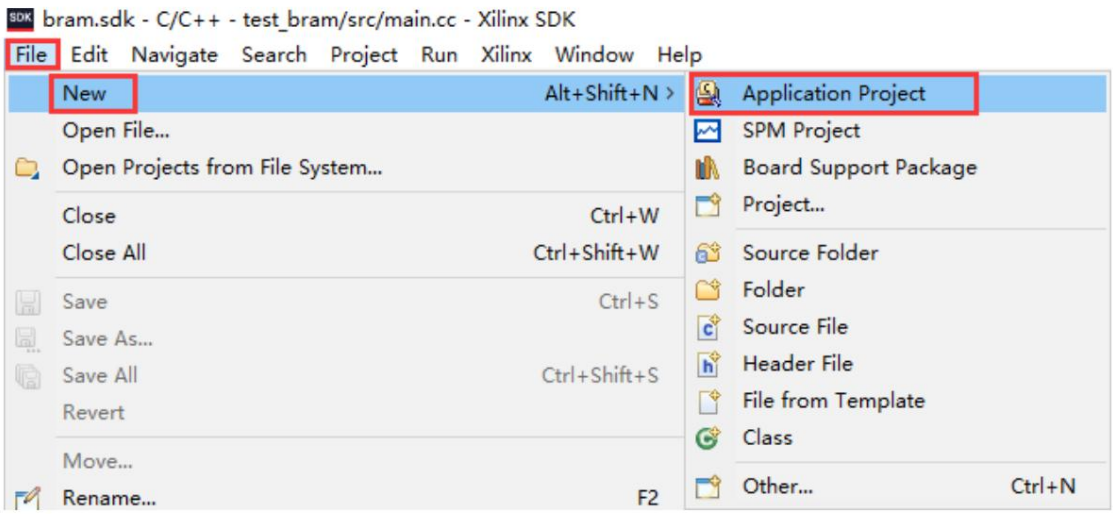


Figure 4- 52

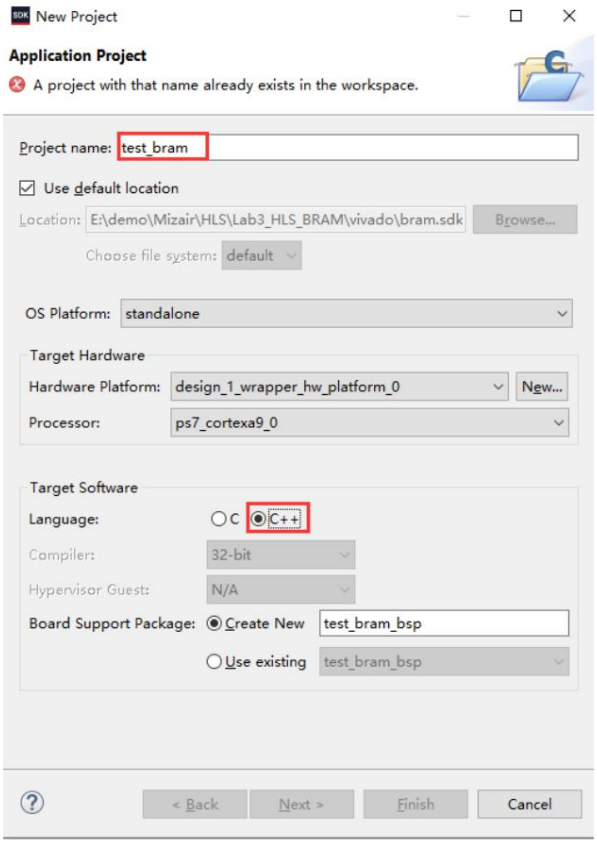


Figure 4- 53

Connect the board as shown:

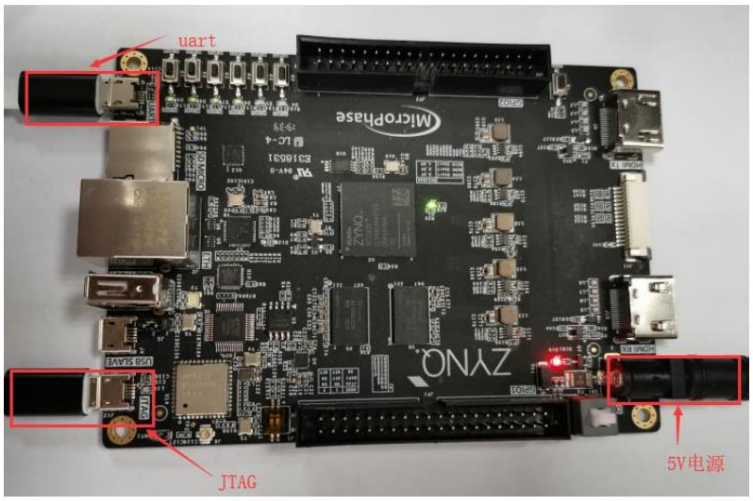


Figure 4- 54

After creating the project, add the following code:

```
1 /*
2         * Empty C++ Application 3 */

4 #include <stdio.h>
5 #include "xil_printf.h"
6 #include "xparameters.h"
7 #include "math.h"
8 #include "xcrazyfunction.h"
9
10
11 #define FUNC_DEV_ID XPAR_CRAZYFUNCTION_0_DEVICE_ID
12
13
14 float * xVect_Hw = (float *) 0x40000000; 15 float *
recVect_Hw = (float *) 0x42000000;
16
17 static XCrazyfunction Crazyfunction; 18 static
XCrazyfunction_Config *Crazyfunction_Cfg;
19
20
21 int initCrazyfunction(){
22 int status;
23 Crazyfunction_Cfg = XCrazyfunction_LookupConfig(FUNC_DEV_ID); 24 status =
XCrazyfunction_CfgInitialize(&Crazyfunction,Crazyfunction_Cfg); 25 if(status!=XST_SUCCESS){

26              printf("initialize function failed\n");
27              return status;
28 }
29 return XST_SUCCESS; 30 }

31
32
33
34 void crazyFunction(float x[100],float y,float result[100]){ 35 int idx; 36

for(idx=0;idx<100;idx++){
37              result[idx] = (sin(x[idx]) + sqrtf(expf(y)))/2;
38 }
39 }
```

```
40
41 u32 float_to_u32(float value){ 42 u32
result; 43 union
float_byte{
44              float v;
45              u8 byte[4];
46 }data;
47 data.v = value;
48 result = (data.byte[3]<<24) + (data.byte[2]<<16)+(data.byte[1]<<8) + (data.byte[0]); 49 return result;


50
51 }
52
53
54 int main() 55
{
56
57 int idx;
58 float xVect_Sw[100]; 59 float
recVect_Sw[100];
60
61 int status; 62
status = initCrazyfunction(); 63 if(status!
=XST_SUCCESS){
64              printf("initialize function failed\n");
65              return status;
66 }
67
68 for(idx=0;idx<100;idx++){ xVect_Sw[idx]
69              = idx; xVect_Hw[idx] = idx;
70
71 } 72
printf("test in software mode\n");
73
74 crazyFunction(xVect_Sw,1.0f,recVect_Sw);
75
76
77 printf("test in hardware mode\n"); 78
XCrazyfunction_Set_y(&Crazyfunction,float_to_u32(1.0f));
79
80 XCrazyfunction_Start(&Crazyfunction);
```

```
81 while(!XCrazyfunction_IsDone(&Crazyfunction)){
82
83 };
84
85 //Output calculation
results 86 for(idx=0;idx<100;idx++) 87
{
88          printf("resVect_Hw[%d]          =          %f,recVect_Sw[%d]=          %f
\n",idx,recVect_Hw[idx],idx,recVect_Sw[idx]); 89 }


90
91 return 0;
92 }
```

The function completed by this code is to calculate the result of function operation by calling HLS IP and

Use the function to directly calculate the result of the function operation. By comparing the values of the two, you can determine whether the IP is working normally.

do.

Select the project for debugging, as shown below:

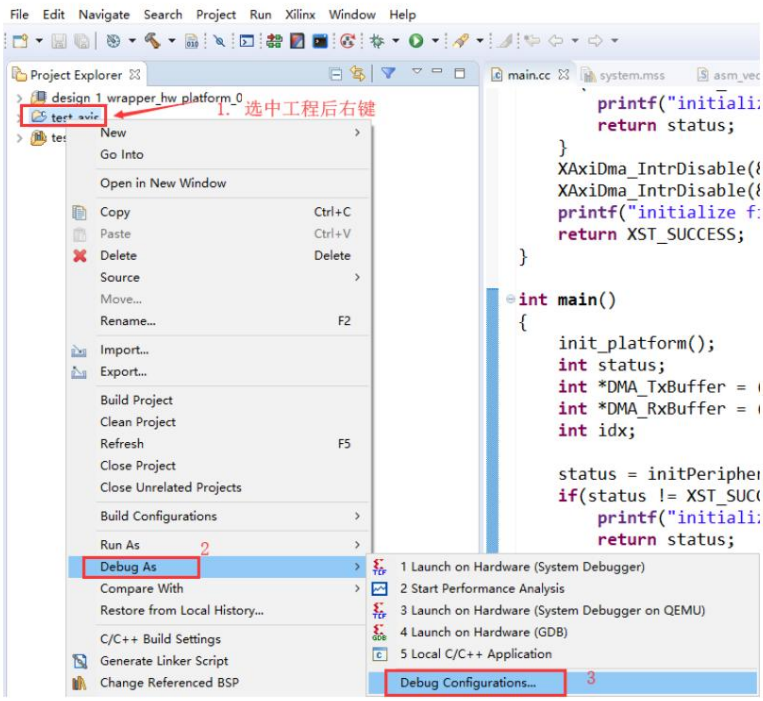


Figure 4- 55

In the pop-up interface, configure as shown below to run and enter SDK debugging

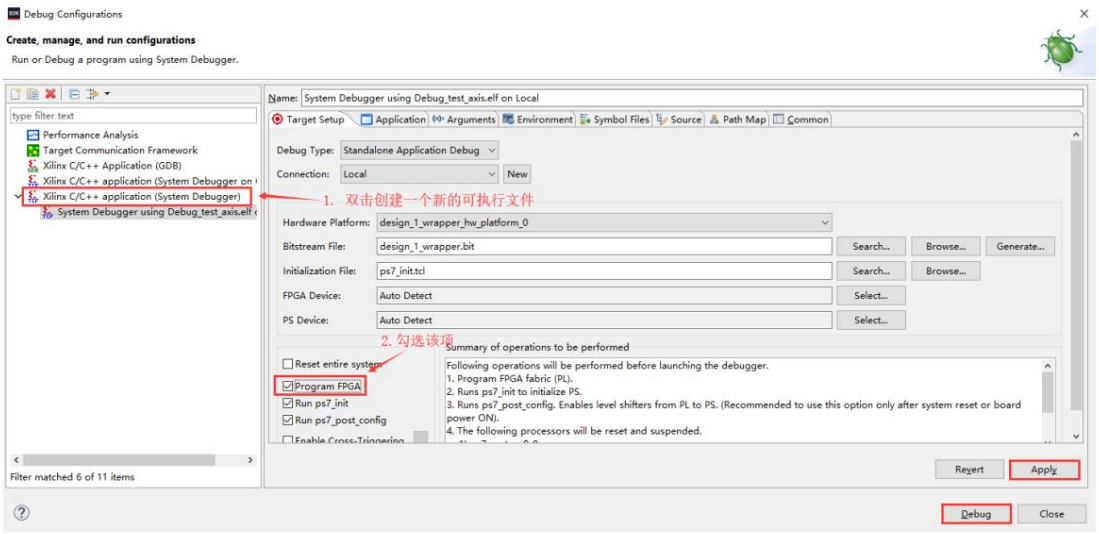


Figure 4- 56

After entering the debugging interface, first connect the serial port to observe the print results:

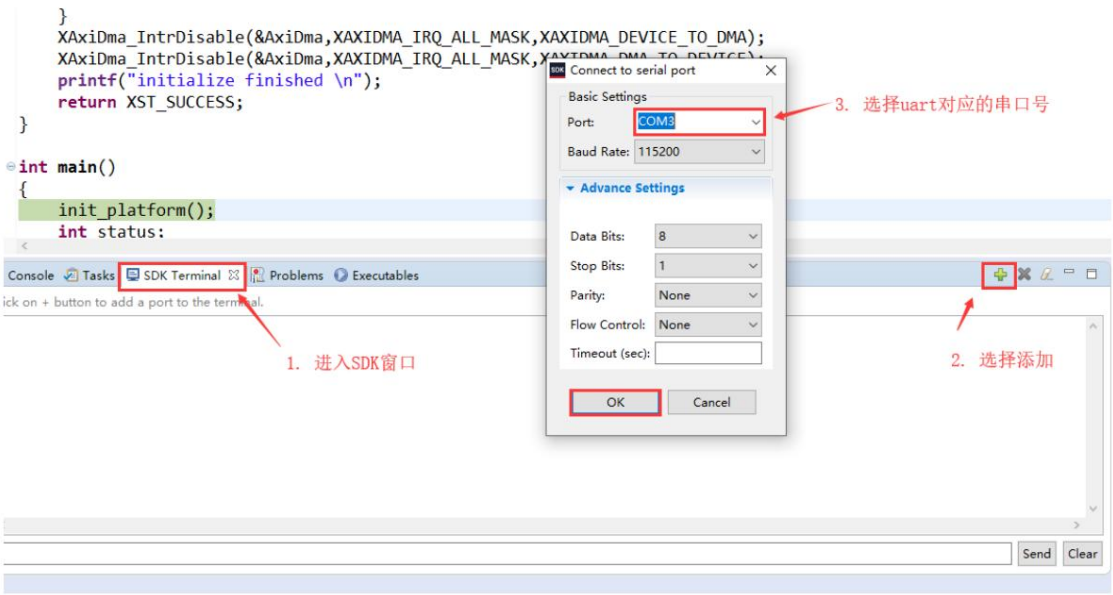


Figure 4-57

After clicking Debug, the complete result of running is shown in the figure:

```
test in software mode
test in hardware mode

resVect_Hw[0] = 0.824361,recVect_Sw[0]= 0.824361
resVect_Hw[1] = 1.245096,recVect_Sw[1]= 1.245096
resVect_Hw[2] = 1.279009,recVect_Sw[2]= 1.279009
resVect_Hw[3] = 0.894921,recVect_Sw[3]= 0.894921
resVect_Hw[4] = 0.445959,recVect_Sw[4]= 0.445959
resVect_Hw[5] = 0.344898,recVect_Sw[5]= 0.344898
resVect_Hw[6] = 0.684653,recVect_Sw[6]= 0.684653
resVect_Hw[7] = 1.152854,recVect_Sw[7]= 1.152854
resVect_Hw[8] = 1.319040,recVect_Sw[8]= 1.319040
resVect_Hw[9] = 1.030420,recVect_Sw[9]= 1.030420
resVect_Hw[10] = 0.552350,recVect_Sw[10]= 0.552350
resVect_Hw[11] = 0.324366,recVect_Sw[11]= 0.324365
resVect_Hw[12] = 0.556074,recVect_Sw[12]= 0.556074

resVect_Hw[13] = 1.034444,recVect_Sw[13]= 1.034444
resVect_Hw[14] = 1.319664,recVect_Sw[14]= 1.319664
resVect_Hw[15] = 1.149505,recVect_Sw[15]= 1.149505
resVect_Hw[16] = 0.680409,recVect_Sw[16]= 0.680409
resVect_Hw[17] = 0.343662,recVect_Sw[17]= 0.343662
resVect_Hw[18] = 0.448867,recVect_Sw[18]= 0.448867
resVect_Hw[19] = 0.899299,recVect_Sw[19]= 0.899299
resVect_Hw[20] = 1.280833,recVect_Sw[20]= 1.280833
resVect_Hw[21] = 1.242688,recVect_Sw[21]= 1.242688
resVect_Hw[22] = 0.819935,recVect_Sw[22]= 0.819935
resVect_Hw[23] = 0.401250,recVect_Sw[23]= 0.401250
resVect_Hw[24] = 0.371571,recVect_Sw[24]= 0.371571
```

Figure 4- 58

It can be seen that the result of directly calling the function in the software is consistent with that of calling the IP.

This shows that IP is currently working properly.

4.4 Experimental Summary

In this experiment, floating point numbers are completed by calling the mathematical function library supported by HLS in HLS.

The calculation is done and a BRAM type interface is generated. Through the hardware environment built in Vivado,

Complete the data interaction between PS and PL. And in the SDK environment, through simple verification code,

The HLS generated IP is verified.

This experiment is relatively simple, mainly to show how to use this complete software chain.

It can be seen that the correlation between Xilinx's tools is still very strong.

Chapter 5 Implementing **the AXI-Stream** Interface in **HLS**

In the previous chapter, we introduced the commonly used interface BRAM. BRAM can be used for data continuity and

In this experiment, we will introduce how to use HLS to generate AXI-stream interface when the data volume is not large.

HLS IP, this experiment is still relatively simple. Through a simple experiment, we can complete the AXI-stream connection.

This interface will be frequently used in the future.

5.1 Simple interface for implementing **AXI-stream** in HLS

5.1.1 Add source files

After creating the HLS project, create a new source file. First, create a new header file

```
1 #ifndef __DO_GANI_H__
2 #define __DO_GAIN_H__
3 #include "hls_stream.h"
4 #include "ap_axi_sdata.h"
5
6 //  Define a  AXI stream  data of type , accompanied by sideChannel
7 typedef ap_axis<32,1,1,1> intSideChannel;
8 typedef hls::stream<intSideChannel> AXI_STREAM;
9
10 // Declaring a function
11 void doGain(AXI_STREAM & inStream,AXI_STREAM & outStream, int gain);
12 #endif
```

The header files include hls_stream.h and ap_axi_sdata.h.

hls_stream contains AXI_Stream related variables. After importing this header file, it can be used to declare

AXI_Stream data in stream format. Ap_axi_sdata.h This header file can be used to define AXI_Stream

The data format in the stream must meet the AXI_stream protocol, such as keep, id, dst, etc.

Contained in axi_stream.

5.1.2 Adding a custom function

The top-level function of this experiment is relatively simple. It just multiplies the input stream data by 5 and gives it to the output stream.

According to the data of the flow, other information remains unchanged.

The source files are as follows:

```
1 #include "doGain.h"
2
3 //AXI stream 4 void    Input and output streams
doGain(AXI_STREAM & inStream,AXI_STREAM & outStream, int gain){
5 #pragma HLS INTERFACE axis register both port=outStream
6 #pragma HLS INTERFACE axis register both port=inStream
7 #pragma HLS INTERFACE s_axilite port=gain bundle=CRTL_BUS
8 #pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
9       int idx;
10 for(idx=0;idx<1000;idx++){
11 #pragma HLS PIPELINE
12              intSideChannelValueIn ;
13              intSideChannelValueOut ;
14              ValueIn = inStream.read();
15              ValueOut.data = ValueIn.data * gain;
16              ValueOut.keep = ValueIn.keep;
17              ValueOut.id = ValueIn.id;
18              ValueOut.last = ValueIn.last;
19              ValueOut.user = ValueIn.user;
20              ValueOut.dest = ValueIn.dest;
21              ValueOut.strb = ValueIn.strb;
22              outStream.write(ValueOut);
23 }
24 }
```

Lines 5 to 8 are directives of the interface. After adding these interfaces, you can use AXI_STREAM type

The data interface is converted to the AXI_STREAM interface type.

The way to add a directive of the AXI_STREAM interface type is the same as that of the BRAM interface type.

The adding method is similar, as shown in the figure Error! Reference source not found.

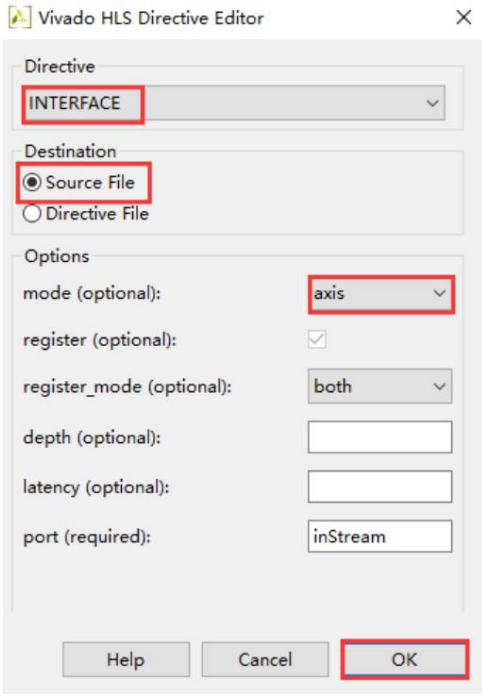


Figure 5- 1

In the defined function, the function is just to multiply the data in the input stream by an integer and give it to the output stream.

Outgoing data flow.

5.1.3 Add **C** simulation file

In the C simulation file, all that is done is to provide an input stream to the function to be tested and observe the results.

To determine whether the top-level function is working properly, the << and >> in the simulation file are overloaded operators, which are equivalent to

Data is pushed into or received from a data flow pipeline.

```
1 #include "stdio.h"
2 #include "doGain.h"
3
4 int main(){ 5 //
Declare input and output streams
6 AXI_STREAM inStream;
7 AXI_STREAM outStream; 8 int idx;
9 //Fill data into
the input stream 10
for(idx=0;idx<1000;idx++){
11          intSideChannel valIn;
12          valIn.data = idx;
```

```
13          valIn.dest = 0;
14          valIn.id = 0;
15          valIn.keep = 1;
16          valIn.last = 1;
17          valIn.strb = 1;
18          inStream << valIn;
19 }
20 //Call the top-level function
21 doGain(inStream, outStream,5);
22
23 //Print the results of the output stream
24 for(idx=0;idx<1000;idx++){
25          intSideChannel valOut;
26          outStream.read(valOut);
27          printf("value out is %d\n",(int)valOut.data);
28 }
29
30 return 0;
31 }
```

After adding the C simulation file, perform C simulation. The result of C simulation is shown in the figure:

```
INFO: [SIM 211-2] *************** CSIM start ***************
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
make: 'csim.exe' is up to date.
value out is 0
value out is 5
value out is 10
value out is 15
value out is 20
value out is 25
value out is 30
value out is 35
value out is 40
value out is 45
value out is 50
value out is 55
value out is 60
value out is 65
value out is 70
value out is 75
value out is 80
value out is 85
value out is 90
```

Figure 5- 2

As you can see, the output data flow is 5 times the input data flow.

5.1.4 Perform **C** Synthesis

After the C simulation is completed, perform C synthesis to map the RTL circuit. After the C synthesis is completed, check the interface list.

In the table, you can see that the AXI_Stream interface has been generated, as shown in the figure

inStream_TDATA	in	32	axis	inStream_V_data_V	pointer
inStream_TVALID	in	1	axis	inStream_V_dest_V	pointer
inStream_TREADY	out	1	axis	inStream_V_dest_V	pointer
inStream_TDEST	in	1	axis	inStream_V_dest_V	pointer
inStream_TKEEP	in	4	axis	inStream_V_keep_V	pointer
inStream_TSTRB	in	4	axis	inStream_V_strb_V	pointer
inStream_TUSER	in	1	axis	inStream_V_user_V	pointer
inStream_TLAST	in	1	axis	inStream_V_last_V	pointer
inStream_TID	in	1	axis	inStream_V_id_V	pointer
outStream_TDATA	out	32	axis	outStream_V_data_V	pointer
outStream_TREADY	in	1	axis	outStream_V_data_V	pointer
outStream_TVALID	out	1	axis	outStream_V_dest_V	pointer
outStream_TDEST	out	1	axis	outStream_V_dest_V	pointer
outStream_TKEEP	out	4	axis	outStream_V_keep_V	pointer
outStream_TSTRB	out	4	axis	outStream_V_strb_V	pointer
outStream_TUSER	out	1	axis	outStream_V_user_V	pointer
outStream_TLAST	out	1	axis	outStream_V_last_V	pointer
outStream_TID	out	1	axis	outStream_V_id_V	pointer

Figure 5- 3

Next, we will conduct joint simulation and export IP, which will not be described in detail here.

5.2 Building a hardware test platform

Add the IP generated by HLS in the Vivado IP Repository and find the IP in the HLS project directory.

solution is enough.

Then create a Block Design, add the ZYNQ IP to the Block Design, and configure

ZYNQ IP core, as shown in the figure:

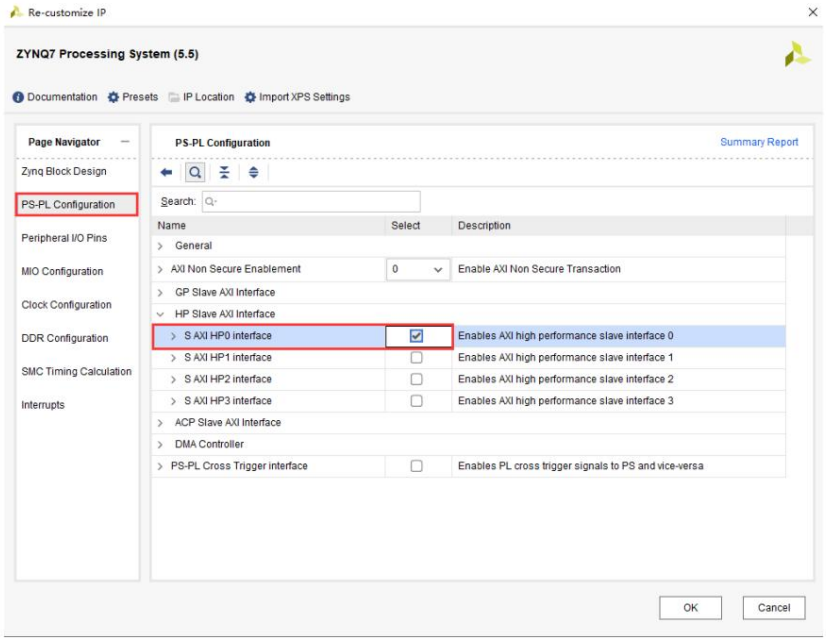


Figure 5- 4

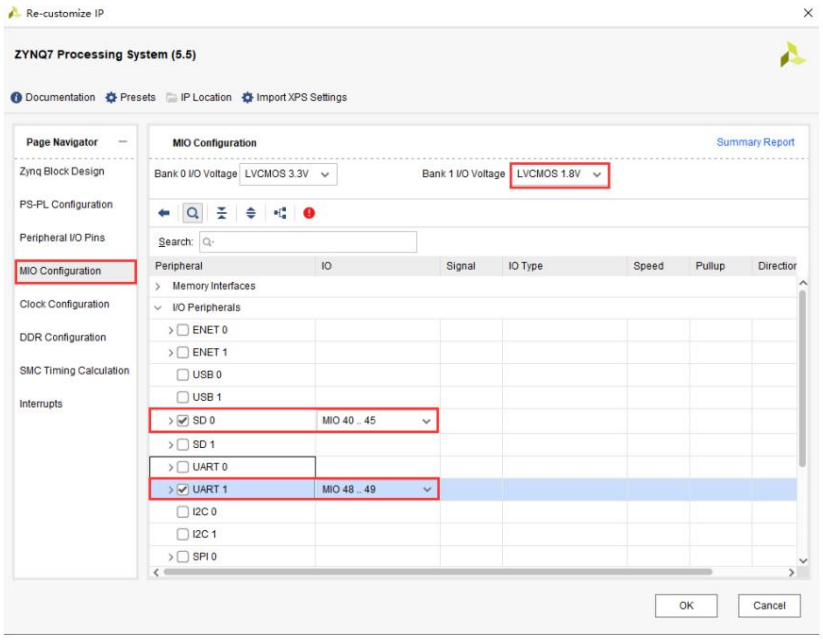


Figure 5- 5

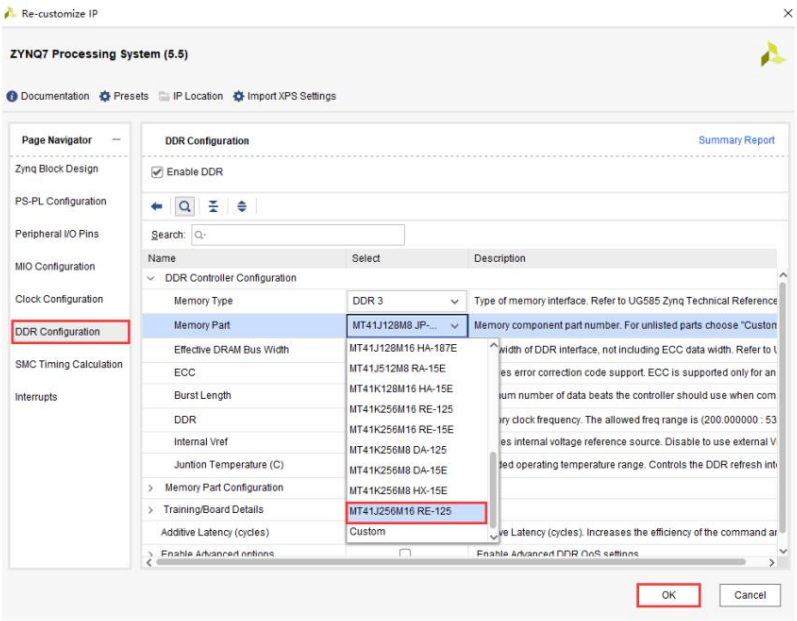


Figure 5-6

Next, add the AXI DMA IP and configure it

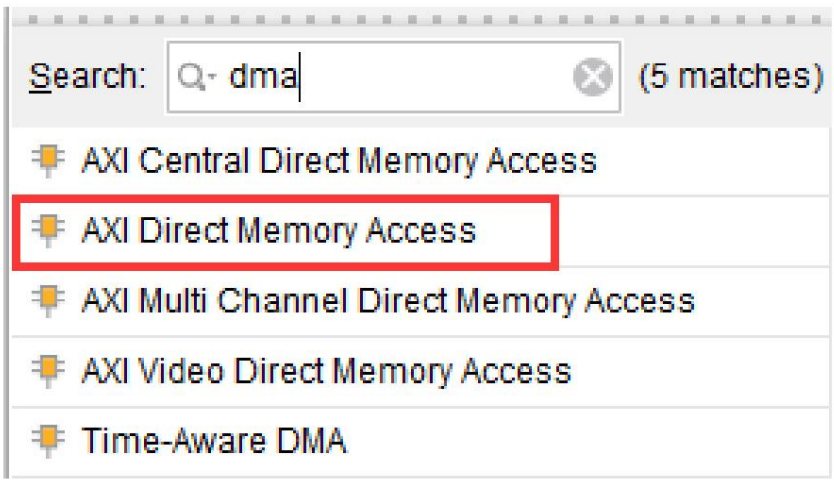


Figure 5-7 Add DMA IP

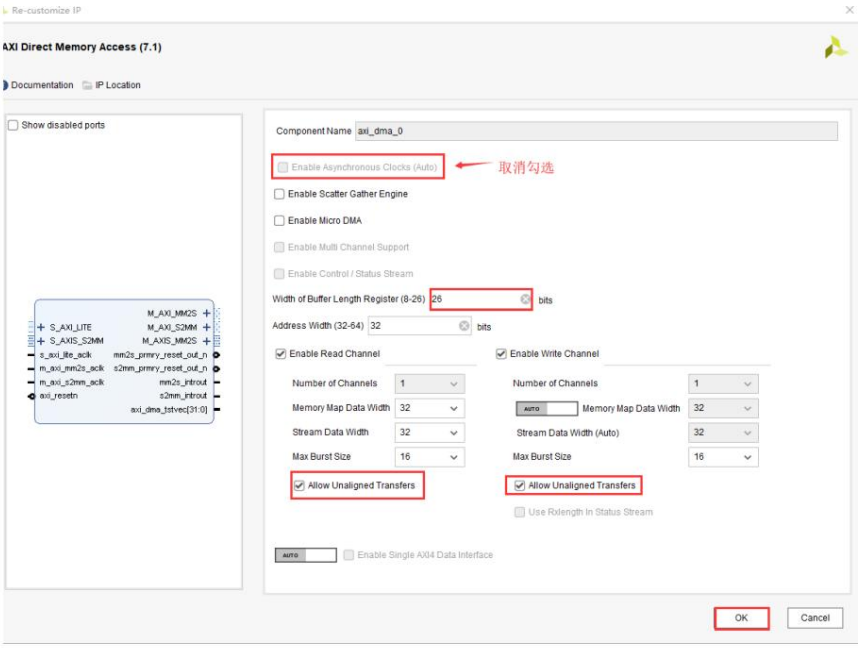


Figure 5-8 Configuring DMA

Next, add the IP with AXI_Stream interface generated by HLS. After adding the IP, the HLS IP

Connect the AXI_Stream interface to the AXI_Stream interface on the DMA and connect the clock line, as shown in the figure

Show:

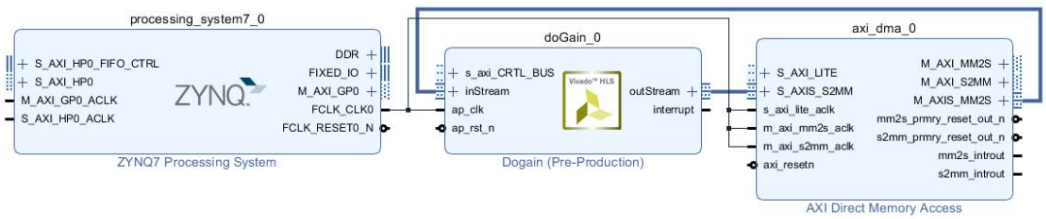


Figure 5- 9

Then click Auto-Connect to complete the construction of Block Design. The final Block Design is as shown in the figure:

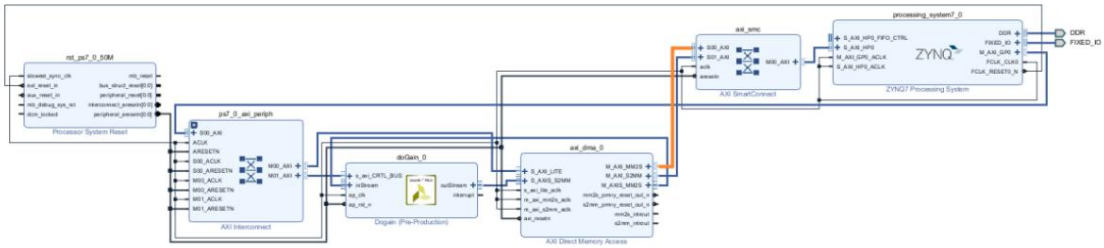


Figure 5- 10

Then create a wrapper to generate a bit stream file. After exporting the bit stream, launch the SDK and perform

Line development.

5.3 Create **SDK** Project

After exporting the bitstream, Launch SDK creates a new project. In the main.c file of the project,

The completed code is as follows:

```
1 #include <stdio.h>
2 #include "platform.h" 3 #include
"xil_printf.h"
4 #include "xaxidma.h"
5 #include "xparameters.h" 6 #include
"xdogain.h"
7
8
9 #define GAIN_DEV_ID XPAR_DOGAIN_0_DEVICE_ID 10 #define
DMA_DEV_ID XPAR_AXI_DMA_0_DEVICE_ID
11
12 #define MEM_BASE_ADDR 0x01000000 13 #define
TX_BUFFER_ADDR (MEM_BASE_ADDR + 0x00100000)
14 #define RX_BUFFER_ADDR (MEM_BASE_ADDR + 0x00300000)
15
16 #define ARR_SIZE 1000
17
```

```
18
19
20 int inStream[ARR_SIZE];
21
22 static XDogain DoGian;
23 static XDogain_Config 24 static * Dogain_Cfg;
XAxiDma AxiDma;
25 static XAxiDma_Config * AxiDma_Cfg;
26
27 int initPeripherals(){
28          int status;
29          printf("initialize doGain core\n");
30          Dogain_Cfg = XDogain_LookupConfig(GAIN_DEV_ID);
31          status = XDogain_CfgInitialize(&DoGian,Dogain_Cfg);
32          if(status != XST_SUCCESS){
33                  printf("initialize failed\n");
34                  return status;
35          }
36          printf("initialize AXIDMA core\n");
37          AxiDma_Cfg = XAxiDma_LookupConfig(DMA_DEV_ID);
38          status = XAxiDma_CfgInitialize(&AxiDma,AxiDma_Cfg);
39          if(status != XST_SUCCESS){
40                  printf("initialize failed\n");
41                  return status;
42          }
43
    XAxiDma_IntrDisable(&AxiDma,XAXIDMA_IRQ_ALL_MASK,XAXIDMA_DEVICE_TO_DMA);
44
    XAxiDma_IntrDisable(&AxiDma,XAXIDMA_IRQ_ALL_MASK,XAXIDMA_DMA_TO_DEVICE);
45          printf("initialize finished \n");
46          return XST_SUCCESS;
47 }
48
49 int main()
50 {
51          init_platform();
52          int status;
53          int *DMA_TxBuffer = (int *) TX_BUFFER_ADDR;
54          int *DMA_RxBuffer = (int *) RX_BUFFER_ADDR;
55          int idx;
56
```

```
57          status = initPeripherals();
58          if(status != XST_SUCCESS){
59                  printf("initialize failed\n");
60                  return status;
61          }
62          // Initialize Send buffer
63          for(idx=0;idx<ARR_SIZE;idx++){
64                  inStream[idx] = idx;
65          }
66
67          while(1){
68                  int gain;
69                  printf("Enter a gain:");
70                  scanf("%d",&gain);
71                  // gain set up Value
72                  XDogain_Set_gain(&DoGian,gain);
73                  // Start Testing IP nuclear
74                  XDogain_Start(&DoGian);
75                  // refresh cache so that it can be observedcache Data in
76                  Xil_DCacheFlushRange((u32)&inStream,(ARR_SIZE * sizeof(int)));
77                  Xil_DCacheFlushRange((u32)DMA_RxBuffer,(ARR_SIZE printf("sending data * sizeof(int)));
78                  to IP\n");
79                  // fromDMA Send data to IP nuclear
80                  XAxiDma_SimpleTransfer(&AxiDma,(u32)inStream,ARR_SIZE *
sizeof(int),XAXIDMA_DMA_TO_DEVICE);
81
82                  printf("getting data from IP \n");
83                  // fromIP Get data
84                  XAxiDma_SimpleTransfer(&AxiDma,(u32)DMA_RxBuffer,ARR_SIZE *
sizeof(int),XAXIDMA_DEVICE_TO_DMA);
85                  while(XAxiDma_Busy(&AxiDma,XAXIDMA_DEVICE_TO_DMA)){
86
87                  };
88
89                  Xil_DCacheInvalidateRange((u32)DMA_RxBuffer,ARR_SIZE while(! * sizeof(int));
90                  XDogain_IsDone(&DoGian)){
91
92                  };
93                  printf("calculate is done \n");
94                  for(idx=0;idx<ARR_SIZE;idx++){
95                          printf("Rx_Buffer[%d]=%d\n",idx,DMA_RxBuffer[idx]);
```

```
96                  }
97              }
98
99              return 0;
100 }
```

The function of this part of the code is that the user enters a number for the IP gain by pressing a key.

The data is passed to the IP core through DMA, and the IP core multiplies the incoming data stream by the gain and outputs

The data flow is transferred to the memory by DMA, thus completing the data flow in the form of AXI_Stream.

After connecting the development board, click debug to start running; in the SDK Terminal, you will be prompted to enter a

The value is entered on the keyboard and then sent to the FPGA through the serial port.

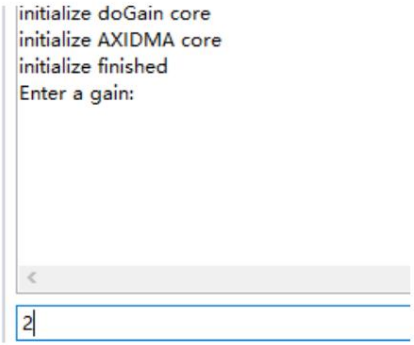


Figure 5- 11

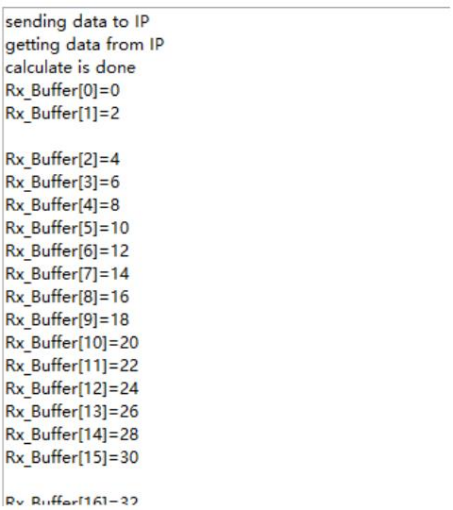


Figure 5- 12

You can observe that the output data is printed on the console through the serial port.

5.4 Experimental Summary

In this chapter's experiment, a simple example was used to complete the HLS generation of an AXI_Stream interface. The IP core is generated, and the task completed by this IP is relatively simple. In the vivado integrated environment, through DMA connects the IP core generated by HLS to DDR through DMA, allowing ZYNQ to access the HLS generated IP.

Supports AXI_Stream protocol data, which will be used when using HLS for image processing related development Often used. In subsequent experiments, AXI_Stream data will be used extensively.

Chapter 6 Image Histogram

6.1 Experimental Introduction

For a grayscale image, it can be expressed as I(x,y). I is the intensity of the image.

It is the statistical distribution probability of different gray levels. For example, for an image with a pixel bit of 8, its gray level

The range is from 0 to 255. The grayscale histogram is to count the pixel ratio of each gray level.

It can directly reflect the brightness of the image. For example, if the probability of 0 is large, it means that the image is dark, and 255 is

The higher the probability, the brighter the image.

The following is a brief explanation of the meaning of the grayscale histogram through a grayscale image:

Enter the following code in Matlab to convert a color RGB image into a grayscale image.

And display the histogram of the grayscale image.

```
img_rgb = imread('Lena.jpg');
imtool(img_rgb);
img_gray = rgb2gray(img_rgb);
imtool(img_gray);
imhist(img_gray);
```

Figure 6-1 Original image

Figure 6-2 Grayscale image

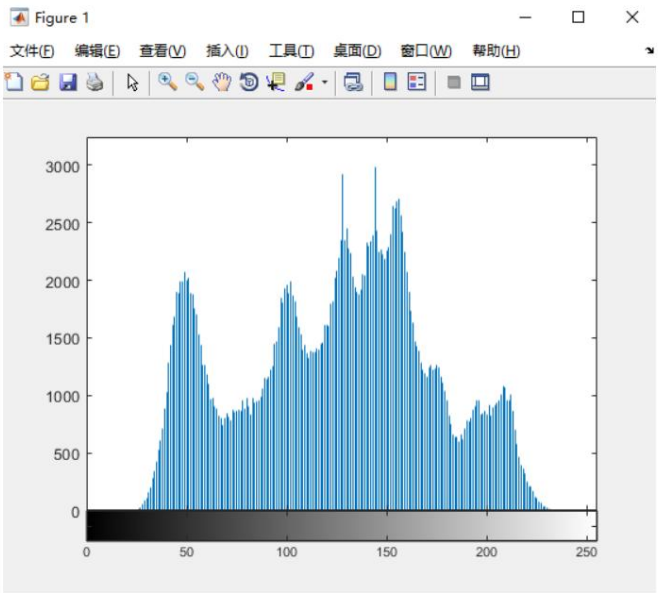


Figure 6-3 Grayscale image histogram

From the above grayscale image histogram, we can see that the grayscale image histogram is the histogram of each pixel in the image.

The grayscale values of the pixels are counted. Different bins are divided according to a certain grayscale interval.

If the value falls within the bin interval, the statistical value of the bin interval is increased by 1.

The range is 1, that is, each different gray value is counted. From the grayscale image, we can see that the image

The grayscale value distribution is moderate, which means that the brightness and darkness of the image are moderate.

6.2 **HLS** Implementation of Image Histogram

In this experiment, the histogram of an image is counted in HLS. After creating the project,

First add the source files and C simulation files.

6.2.1 Defining header files

The header file is defined as follows:

```
1 #ifndef __HISTO_HEAD_H__
2 #define __HISTO_HEAD_H__
3 //      Introductionstream     kind
4 #include <hls_stream.h>
5 //6    use     axi stream side channel(TLAST,TKEEP,TUSER,TID and so on)
#include <ap_axi_sdata.h>
7
8 //9    Define input and output files
#define INPUT_IMAGE "Lena.bmp"
10 #define HISTOGRAM_FILE "histogram.txt"
11 //8    bit unsigned  side_channel       Data Types
12 typedef ap_axiu<8,1,1,1> uint_8_side_channel;
13 typedef hls::stream<uint_8_side_channel> AXI_STREAM;
14 void Histogram(AXI_STREAM &inStream,int HistOut[256]);
15
16 #endif
```

In the header file, the format of each data in the AXI_Stream stream data is redefined;

The data stream format of AXI_Stream declares the top-level function and defines the input and output files.

6.2.2 Add source files

Define the top-level function in the source file. The definition of the top-level function is as follows:

```
1 #include "histogram.h"
2
3 void Histogram(AXI_STREAM &inStream, int HistoOut[256]){
4 #pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
5 #pragma HLS INTERFACE axis register port=inStream
6 #pragma HLS INTERFACE bram port=HistOut
7
```

```
8 int idx;
9 uint_8_side_channel recPixel;
10 //11 Before each frame comes, clear bram Contents
for(idx=0;idx<256;idx++){
12 #pragma HLS PIPELINE
13          HistoOut[idx]=0;
14 }
15 //    Receive image data and calculate histogram
16 for(idx=0;idx<(512*512);idx++){
17          recPixel = inStream.read();
18          HistoOut[recPixel.data] +=1;
19 }
20 }
```

Lines 4 to 6 are the Directives for adding interfaces, where the input AXI_Stream format data is defined.

Define the data type as an AXI_Stream interface, define the array type as a BRAM interface, and

The return value of the function is defined as the AXI_Lite interface, which is convenient for calling the IP in the subsequent SDK.

Lines 11 to 14 clear the data in the array so that it can be reset each time an image arrives.

Start counting again.

Lines 16 to 19 read the data in the input data stream and count the grayscale values of the image into a histogram.

middle.

6.2.3 Add simulation file

In the simulation file, add the following:

```
1 #include "stdio.h"
2 #include "histogram.h"
3 #include <hls_opencv.h>
4 #include <opencv2/core/core.hpp>
5
6 int histogram[256];
7
8 void saveHistgram(const char * file, int *histgram){
9       FILE *pFile;
10 pFile = fopen(file,"w");
11 int idx;
12 if(pFile!=NULL){
```

```
13              for(idx=0;idx<256;idx++){
14                      fprintf(pFile,"Bin[%d]=%d\n",idx,histgram[idx]);
15              }
16 }
17 }
18
19 int main(){
20 printf("Load Image %s \n",INPUT_IMAGE);
21 cv::Mat imageSrc;
22 imageSrc = cv::imread(INPUT_IMAGE);
23 //      Convert the image to grayscale
24 cv::cvtColor(imageSrc, imageSrc,CV_BGR2GRAY);
25 printf("image row=%d,image col= %d\n",imageSrc.rows,imageSrc.cols);
26 //      Define input and output image data flow
27 AXI_STREAM inStream;
28 for(int idxRows=0;idxRows<imageSrc.rows;idxRows++){
29              for(int idxCols=0;idxCols<imageSrc.cols;idxCols++){
30                      uint_8_side_channel valIn;
31                      valIn.data = imageSrc.at<unsigned char>(idxRows, idxCols);
32                      valIn.dest = 1;
33                      valIn.keep = 1;
34                      valIn.last = 1;
35                      valIn.strb = 1;
36                      valIn.id = 0;
37                      valIn.user = 1;
38                      inStream << valIn;
39              }
40 }
41 Histogram(inStream, hsitogram);
42 saveHistgram(HISTOGRAM_FILE,hsitogram);
43 return 0;
44 }
```

In the main function, a different method from the second experiment is used to import the image from the external file into

Import it into the project and convert it into the data format supported by HLS.

The << in line 38 is an overload of the operator, which is equivalent to pushing data into the pipeline.

The grayscale image imitation map is output to an external txt file.

6.2.4 Perform simulation and synthesis

After adding the source files, you can use them for C simulation, C synthesis, and C/RTL joint simulation.

After the C simulation is finished, you can see that the Console reports no error, as shown in the figure:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../code/test_histogram.cpp in debug mode
   Compiling ../../../code/histogram.cpp in debug mode
   Generating csim.exe
Load Image Lena.bmp
image row=512,image col= 512
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

Figure 6- 4

At this time, in the Csim folder of the Solution, there will be an output txt text data, which records

The histogram statistics of this image are recorded.

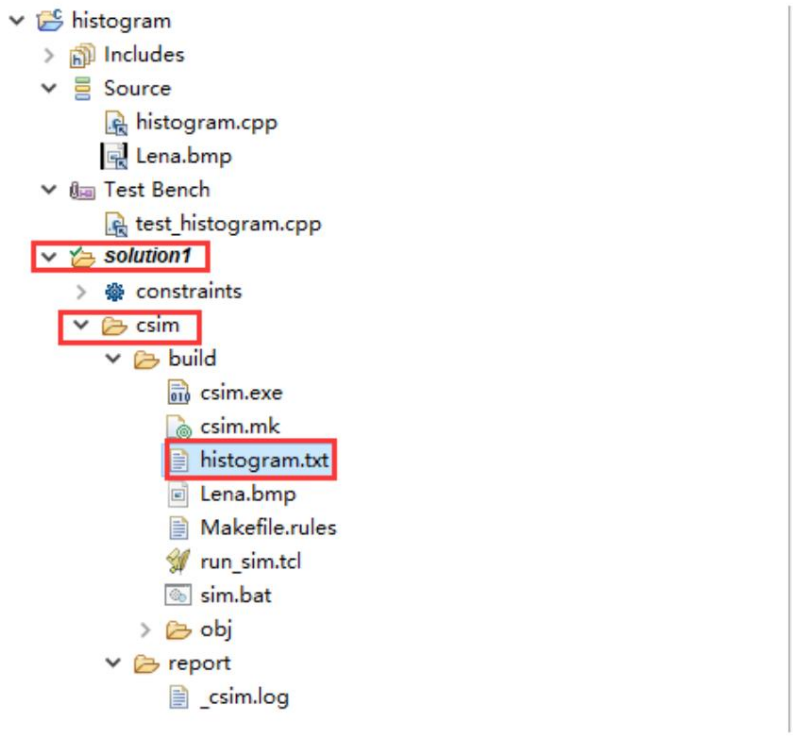


Figure 6- 5

Open it and you can see the statistics of the image histogram, as shown in the figure:

```
Bin[20]=1
Bin[21]=0
Bin[22]=0
Bin[23]=2
Bin[24]=4
Bin[25]=8
Bin[26]=15
Bin[27]=24
Bin[28]=55
Bin[29]=85
Bin[30]=112
Bin[31]=159
Bin[32]=202
```

Figure 6- 6

After the synthesis is completed, you can view the interface and resource usage of the IP mapped to the RTL circuit.

The occupancy is shown in the figure. It can be seen that the histogram statistics occupy very little resources.

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	77
FIFO	-	-	-	-
Instance	0	-	36	40
Memory	-	-	-	-
Multiplexer	-	-	-	31
Register	-	-	82	-
Total	0	0	118	148
Available	280	220	106400	53200
Utilization (%)	0	0	~0	~0

Figure 6-7 Resource usage

The interface after synthesis is shown in the figure: You can see that after the synthesis is completed, AXI_Lite,

AXI_Stream and BRAM interfaces.

s_axi_CRTL_BUS_RDATA	out	32	s_axi	CRTL_BUS	return void
s_axi_CRTL_BUS_RRESP	out	2	s_axi	CRTL_BUS	return void
s_axi_CRTL_BUS_BVALID	out	1	s_axi	CRTL_BUS	return void
s_axi_CRTL_BUS_BREADY	in	1	s_axi	CRTL_BUS	return void
s_axi_CRTL_BUS_BRESP	out	2	s_axi	CRTL_BUS	return void
ap_clk	in	1	ap_ctrl_hs	Histogram	return value
ap_rst_n	in	1	ap_ctrl_hs	Histogram	return value
interrupt	out	1	ap_ctrl_hs	Histogram	return value
inStream_TDATA	in	8	axis	inStream_V_data_V	pointer
inStream_TVALID	in	1	axis	inStream_V_dest_V	pointer
inStream_TREADY	out	1	axis	inStream_V_dest_V	pointer
inStream_TDEST	in	1	axis	inStream_V_dest_V	pointer
inStream_TKEEP	in	1	axis	inStream_V_keep_V	pointer
inStream_TSTRB	in	1	axis	inStream_V_strb_V	pointer
inStream_TUSER	in	1	axis	inStream_V_user_V	pointer
inStream_TLAST	in	1	axis	inStream_V_last_V	pointer
inStream_TID	in	1	axis	inStream_V_id_V	pointer
HistoOut_Addr_A	out	32	bram	HistoOut	array
HistoOut_EN_A	out	1	bram	HistoOut	array
HistoOut_WEN_A	out	4	bram	HistoOut	array
HistoOut_Din_A	out	32	bram	HistoOut	array
HistoOut_Dout_A	in	32	bram	HistoOut	array
HistoOut_Clk_A	out	1	bram	HistoOut	array
HistoOut_Rst_A	out	1	bram	HistoOut	array

Figure 6-8 Integrated interface

After completing the C/RTL co-simulation, we can find the RTL in the Sim folder under Solution

The output results after the simulation.

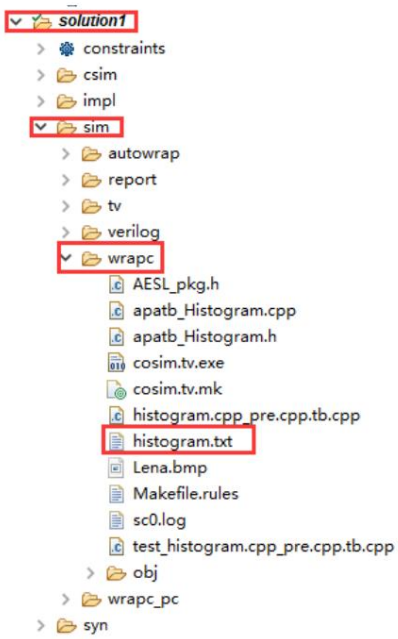


Figure 6- 9

Open the histogram.txt file in the folder and you can get the result obtained by RTL simulation.

It can be seen that the results obtained by RTL and C simulation are consistent.

Next, you can export the IP core and call it in the Vivado integrated development environment.

6.3 Build the test hardware platform

After exporting the IP, create a Vivado project and add the HLS-generated IP to Vivado's IP Repository

middle.

First add the ZYNQ processor in Block Design. The configuration method is similar to the experiment in the previous chapter.

Then add the Histogram IP, click Run Connection Automation after adding, and first perform BRAM

For the port side configuration, select HistoOut_PORTA and click OK to automatically generate a BRAM.

Then, according to the experiment in Chapter 3, add axi_bram_controller so that the PS side can access the BRAM.

As shown in the figure:

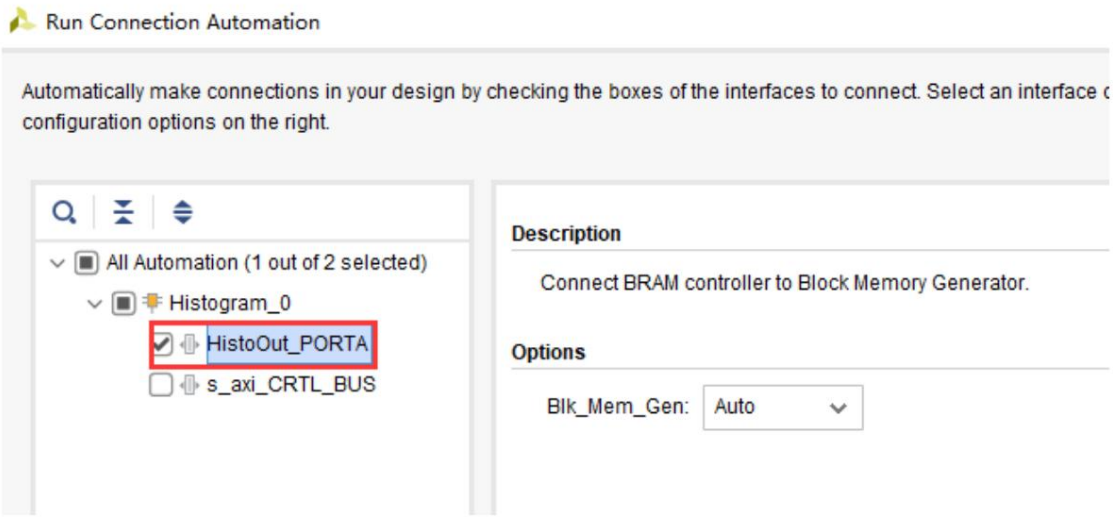


Figure 6- 10

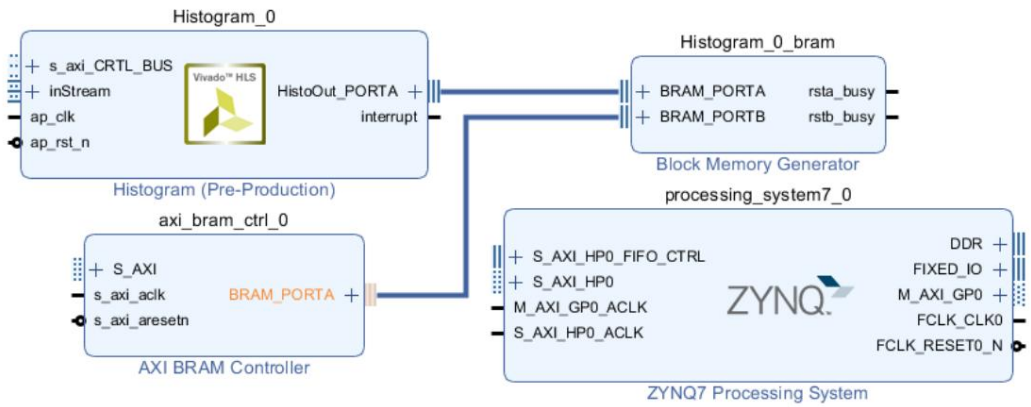


Figure 6- 11

Next, add DMA to allow the IP to access the contents of the DDR. The configuration method of DMA is the same as that in Section 4.

The configuration method in the chapter experiment is similar, only the read channel needs to be used.

Connect the IPs as shown. Then click Run Connection Automation to complete the interconnection between IPs.

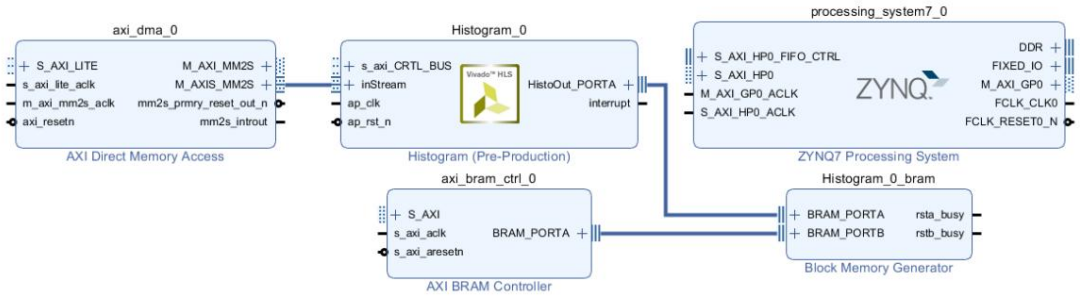


Figure 6- 12

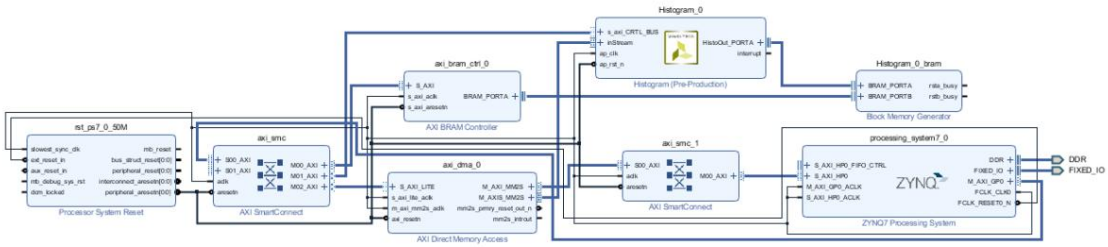


Figure 6- 13

After all connections are completed, generate the wrapper and then generate the bitstream.

After completion, you can start development in SDK.

6.4 Create **SDK** Project

Before creating a Vivado project, use MATLAB to generate a grayscale image in hexadecimal format.

txt text file.

Open MATLAB and run the following script in the project path.

```
img_rgb = imread('Lena.bmp');
img_gray = rgb2gray(img_rgb); fid0 =
fopen('Lena.txt', 'wt'); fprintf(fid0,
'0x%02x, ', img_gray'); fid0 = fclose(fid0);
```

After the operation is completed, a text file named Lena.txt will be generated.

Change the content to an array, and then add the array to the SDK project.

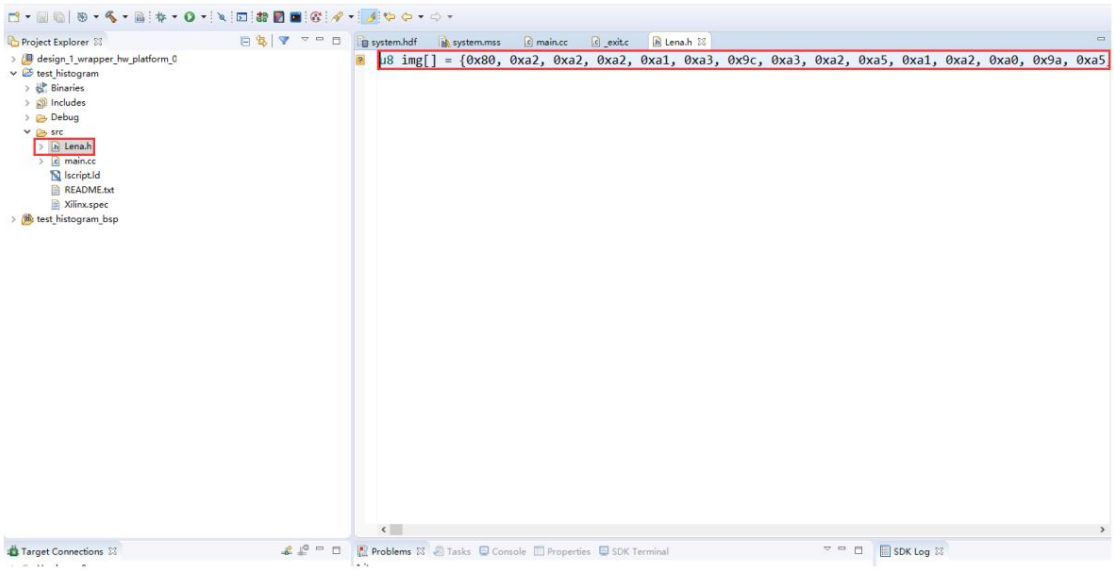


Figure 6- 14

In the SDK, the following design is performed to complete the image histogram through software and calling IP core respectively.

Statistics, and find the maximum and minimum values in the histogram, and finally compare the histogram statistics of the image

Are the results consistent?

The code design is as follows:

```
1 #include <stdio.h>
2 #include <xaxidma.h>
3 #include <xhistogram.h>
4 #include "Lena.h"
5 #include "xparameters.h"
6
7 //IP 8       Device address
#define HISTO_DEV_ID XPAR_HISTOGRAM_0_DEVICE_ID 9 #define
AXIDMA_DEV_ID XPAR_AXIDMA_0_DEVICE_ID
10
```

```
11 //      Image size
12 #define ARR_SIZE 512*512
13
14 //DMA         Memory address used
15 #define MEM_BASE_ADDR 0x01000000
16 #define memRxBuffer_Base (MEM_BASE_ADDR + 0x00300000)
17
18 //BRAM          Address
19 #define BRAM_BASE_ADDR 0x40000000
20
21 //      Defines the input and output buffer     Pointer
22
23 ;unsigned char * RxBuffer = (unsigned char *) memRxBuffer_Base;
24
25 //26   An array of histograms calculated by the software
unsigned int hist_Sw[256];
27
28 unsigned int * hist_Hw = (unsigned int *)BRAM_BASE_ADDR;
29 //30   Software output image matrix
unsigned char imageOut_Sw[ARR_SIZE];
31 //      Input is a pixel matrix such as an image
32 unsigned char imageIn_Hw[ARR_SIZE];
33
34 //      definition IP  Configuration and initialization structure
35 static XAxiDma AxiDma;
36 static XAxiDma_Config *  AxiDma_Cfg;
37 static XHistogram Histo;
38 static XHistogram_Config *Histo_Cfg;
39
40 int initperipheral(){
41          int status;
42
43          // initialization Histogram IP
44          printf("initialize Histogram...\n");
45          Histo_Cfg = XHistogram_LookupConfig(HISTO_DEV_ID);
46          status = XHistogram_CfgInitialize(&Histo,Histo_Cfg);
47          if(status !=XST_SUCCESS){
48                   printf("initialize failed\n");
49          }
50
51
```

```
52          // initialization AXI DMA
53
54          printf("initialize AXIDMA core...\n");
55          AxiDma_Cfg = XAxiDma_LookupConfig(AXIDMA_DEV_ID);
56          status = XAxiDma_CfgInitialize(&AxiDma,AxiDma_Cfg);
57          if(status != XST_SUCCESS){
58                  printf("initialize failed\n");
59                  return status;
60          }
61
      XAxiDma_IntrDisable(&AxiDma,XAXIDMA_IRQ_ALL_MASK,XAXIDMA_DEVICE_TO_DMA);
62
      XAxiDma_IntrDisable(&AxiDma,XAXIDMA_IRQ_ALL_MASK,XAXIDMA_DMA_TO_DEVICE);
63          printf("initialize finished \n");
64          return XST_SUCCESS;
65 }
66
67
68 //69   Calculate the image histogram by software
void doHistSw(unsigned char *img,unsigned int *Array){
70          // Clear the data in the histogram array
71          for(int idx=0;idx<256;idx++){
72                  Array[idx] = 0;
73          }
74
75          for(int idx=0;idx<ARR_SIZE;idx++){
76                  // Fill the histogram array with data by determining the value of each pixel in the image
77                  Array[img[idx]] += 1;
78          }
79 }
80
81 int main()
82 {
83
84          int status;
85          unsigned char min;
86          unsigned char max;
87          // Initialize peripherals
88          status = initperipheral();
89          if(status!=XST_SUCCESS){
90                  printf("initialize timer failed\n");
```

```
91              return status;
92          }
93          // Calculate image histogram using software method
94          printf("Doing histogram on software...\n");
95          doHistSw(img,hist_Sw);
96          // Get the maximum and minimum grayscale values calculated by the software
97          for(int idx=0;idx<256;idx++){
98              min = idx;
99              if(hist_Sw[idx])
100                 break;
101         }
102
103         // Get the maximum value calculated by the software
104         for(int idx=255;idx>=0;idx--){
105             max = idx;
106             if(hist_Sw[idx])
107                 break;
108         }
109         printf("The histogram in Software is between %d~%d ÿ \n",min,max);
110
111         // pass FPGA Complete histogram processing
112         printf("Doing Histogram in hardware...\n");
113         for(int idx=0;idx<ARR_SIZE;idx++){
114             imageIn_Hw[idx] = img[idx];
115         }
116
117         XHistogram_Start(&Histo);
118         // Computed using hardware Histogram
119
120         // refresh cache In order to be able to observe the changes of data in memory
121         Xil_DCacheFlushRange((u32)imageIn_Hw,(ARR_SIZE * sizeof(unsigned char)));
122         Xil_DCacheFlushRange((u32)RxBuffer,(ARR_SIZE*sizeof(unsigned char)));
123
124         //DMA Data transfer
125
126         // Transfer image data to IP
127         XAxiDma_SimpleTransfer(&AxiDma,(u32)imageIn_Hw,(ARR_SIZE*sizeof(unsigned
char)),XAXIDMA_DMA_TO_DEVICE);
128
129         // Wait for data transfer to complete
130         while(XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE)){
```

```
131
132         };
133
134         // Get the maximum and minimum grayscale values calculated by the software
135         for(int idx=0;idx<256;idx++){
136                 min = idx;
137                 if(hist_Hw[idx])
138                         break;
139         }
140
141         // Get the maximum value calculated by the software
142         for(int idx=255;idx>=0;idx--){
143                 max = idx;
144                 if(hist_Hw[idx])
145                         break;
146         }
147         printf("The histogram in Hardware is between %d~%d \n",min,max);
148
149         // Compare results calculated by software and hardware
150         printf("compare the result in Software and Hardware...\n");
151         int error=0;
152         for(int idx=0;idx<256;idx++){
153                 if(hist_Hw[idx]!=hist_Sw[idx]){
154                         error=1;
155                 }
156         }
157         if(error == 1){
158                 printf("The result do not match !\n");
159         }
160         else{
161                 printf("The result do matches well!\n");
162         }
163
164         return 0;
165 }
```

Click Debug to run the project. The results are shown in the figure.

The range of the image histogram is 20~245, which is calculated by software method and by calling IP core.

The results are consistent.

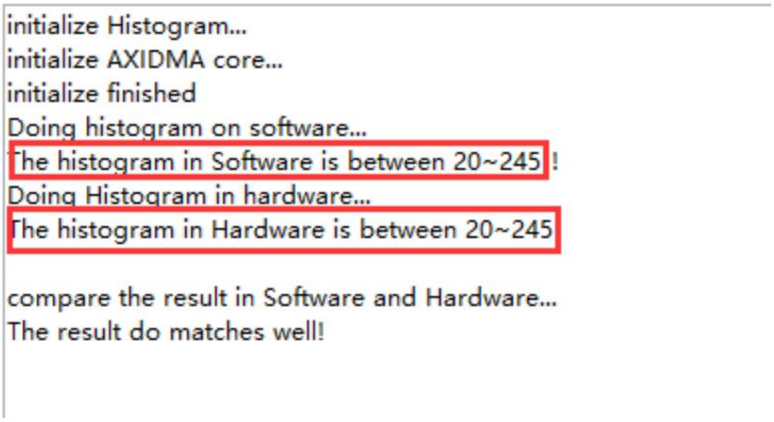


Figure 6- 15

6.5 Experimental Summary

In this experiment, we learned how to process the histogram of a grayscale image and generate it through HLS.

An IP core for calculating histograms was developed and called in Vivado. The functions of the IP core were then tested using the SDK.

Verified.

Histograms are often used in image processing. In applications such as image segmentation, histograms are often used in image processing.

Through this experiment, we can only briefly understand what a histogram is.

The application still requires users to explore it themselves.

Chapter 7 Image Histogram Stretching

7.1 Experimental Introduction

In the previous chapter, we briefly analyzed the statistics of the histogram. In this chapter, we will analyze the statistics of the histogram.

Simple expansion is used to stretch the histogram and adjust the contrast of the image.

First, let's use Matlab to give a simple and intuitive explanation of image stretching. There are two images below

As shown in the figure, one of the images is the original image without histogram stretching, and the other image is the original image with histogram stretching.

The result after histogram stretching. From the result, we can see that the stretched image is clearer than the original image.

The image is brighter and the contrast is higher than before. Use matlab to check the histogram distribution of the two.

The histogram distribution is shown in the figure:

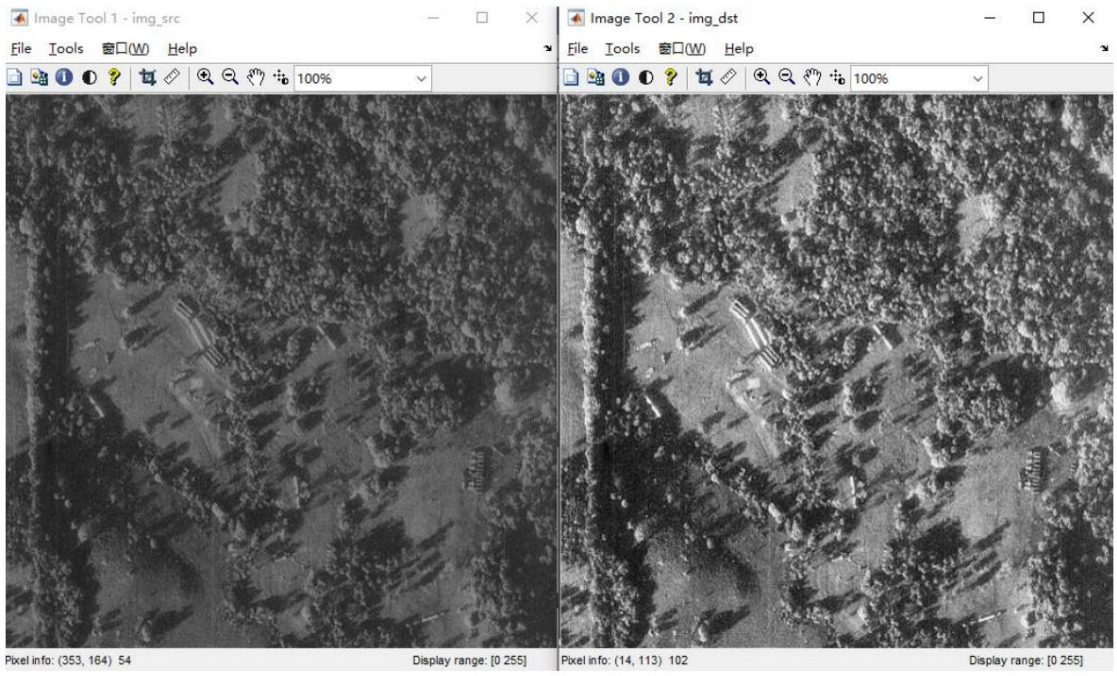


Figure 6-1 Image comparison

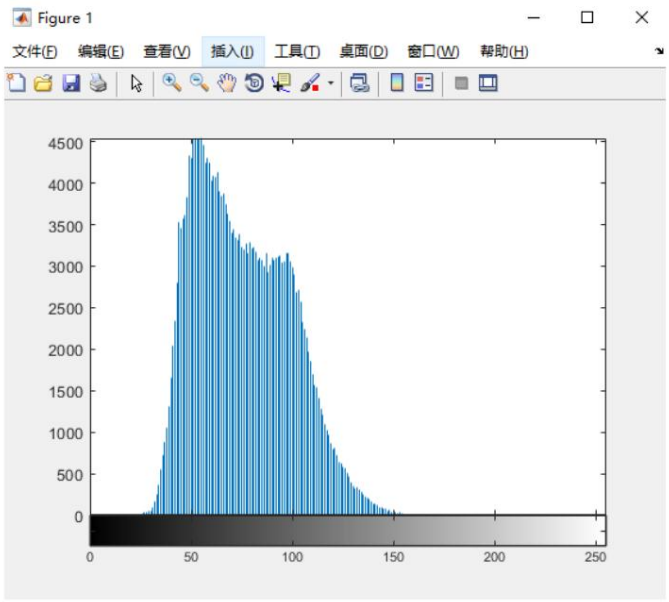


Figure 7-2 Histogram distribution of the original image

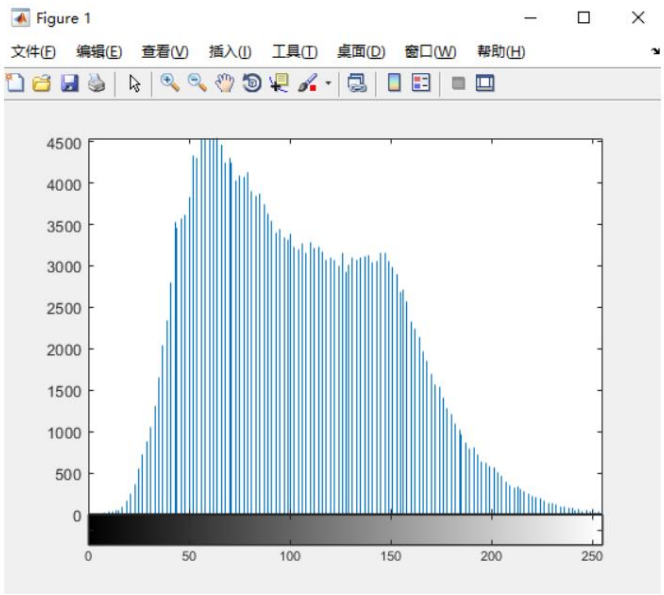


Figure 7-3 Histogram distribution after stretching

By comparing the histograms, we can find that the grayscale of the original image is mainly concentrated in the areas with smaller grayscale values.

area, so it appears darker. After stretching, the grayscale distribution of the image is more balanced, so the image also appears

Brighter than before.

The matlab script for this experiment is as follows: First, customize a function for stretching the histogram

```
function [imgOut] = change(imgIn) yMax
= single(255); xMin =
single(min(min(imgIn))); xMax =
single(max(max(imgIn))); imgOut = imgIn;
for idx=1:numel(imgIn)

    x = single(imgIn(idx));
    imgOut(idx)=((x-xMin))/(xMax-xMin) * yMax;
end
end
```

Then stretch the histogram of the input image

```
img_src = imread('tree.png'); img_src
= rgb2gray(img_src); imtool(img_src);
img_dst =
change(img_src); imtool(img_dst);
```

The simple histogram stretching calculation formula used in this experiment is as follows: where Ymax is set to

The maximum grayscale value is 255, X is the grayscale value of the current pixel, Xmin is the minimum grayscale value of the image, and Xmax is

The maximum gray value of the image.

$$y = \frac{(X - Xmin) * Ymax}{(Xmax - Xmin)}$$

7.2 HLS Implementation of Histogram Stretching

After creating the project, add the source file and C simulation file. The definition of the header file is as follows:

It is almost the same as the content of the experiment in the previous chapter.

```
1 #ifndef __STRETCH_HEAD_H__
2 #define __STRETCH_HEAD_H__ 3 //4
#includetroductionstream    kind
<hls_stream.h> 5 //
        use    axi stream side channel(TLAST,TKEEP,TUSER,TID and so on)
6 #include <ap_axi_sdata.h>
```

```
7
8 //9    Define input and output files
#define INPUT_IMAGE "Tree.png"
10 #define OUTPUT_IMAGE "TreeWithAdjust.bmp"
11
12 //8    bit unsigned  side_channel      Data Types
13 typedef ap_axiu<8,1,1,1> uint_8_side_channel;
14 typedef hls::stream<uint_8_side_channel> AXI_STREAM;
15
16 void Stretch(AXI_STREAM &inStream,AXI_STREAM &outStream, unsigned char min,
unsigned char max);
17
18 #endif
```

Next is the source file for the top-level function, which looks like this:

```
1 #include "HistgramStretch.h"
2
3 void Stretch(AXI_STREAM & inStream,AXI_STREAM & outStream, unsigned char min,
unsigned char max){
4 #pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
5 #pragma HLS INTERFACE s_axilite port=max bundle=CRTL_BUS
6 #pragma HLS INTERFACE s_axilite port=min bundle=CRTL_BUS
7 #pragma HLS INTERFACE axis port=outStream
8 #pragma HLS INTERFACE axis port=inStream
9
10 //histogram channel 11 float xWidth Width
= max - min;
12 //13   Start processing incoming data
uint_8_side_channel recPixel;
14 for(int idx=0;idx <(512*512);idx++){
15 #pragma HLS PIPELINE
16            //  Reading Data
17             uint_8_side_channel currentPixel = inStream.read();
18
19             uint_8_side_channel dataOut;
20            // Get histogram unsigned    The value in
21             char x_t = currentPixel.data;
22            // stretch histogram
23             float y_t_float = ((x_t-min)/xWidth) * 255;
24             unsigned char y_t = y_t_float;
25            //  Pass the data through AXI stream      Output
```

```
26          dataOut.data = y_t;
27          dataOut.dest = currentPixel.dest;
28          dataOut.id = currentPixel.id;
29          dataOut.keep = currentPixel.keep;
30          dataOut.last = currentPixel.last;
31          dataOut.strb = currentPixel.strb;
32          dataOut.user = currentPixel.user;
33          //send the stream
34          outStream.write(dataOut);
35 }
36 }
```

Lines 4 to 8 are the directive declarations of the interface type, which enable the generation of

Corresponding interface.

Inside the function, the image histogram is stretched by the maximum value of the grayscale passed in and

The minimum value is used to calculate the original interval of the calculation formula mentioned above, and then the new gray value of the current pixel is calculated.

Finally, the result is given to the output data stream.

The C simulation file is as follows:

```
1 #include "HistgramStretch.h"
2 #include <hls_opencv.h>
3 #include <stdio.h>
4 #include <opencv2/core/core.hpp>
5
6 char outImage[512][512];
7
8 void saveImages(std::string path, cv::InputArray inArr){
9 double min;
10 double max;
11 cv::minMaxIdx(inArr, &min, &max);
12 cv::Mat adjMap;
13 cv::convertScaleAbs(inArr, adjMap,255/max);
14 cv::imwrite(path,adjMap);
15 }
16
17
18 int main(){
19 printf("Load Image %s",INPUT_IMAGE);
```

```
20 cv::Mat imageSrc;
21 imageSrc = cv::imread(INPUT_IMAGE);
22 //23   Convert the image to grayscale
cv::cvtColor(imageSrc, imageSrc,CV_BGR2GRAY);
24 printf("image row=%d,image col= %d\n",imageSrc.rows,imageSrc.cols);
25 //      Define input and output image data flow
26 AXI_STREAM inStream;
27 AXI_STREAM outStream;
28 //      Define a      CV  The image matrix
29 CV::Mat
imgCvOut(cv::Size(imageSrc.rows,imageSrc.cols),CV_8UC1,outImage,cv::Mat::AUTO_STEP);
30 //31   Fill the input image data into          input stream      middle
for(int idxRows=0;idxRows<imageSrc.rows;idxRows++){
32                  for(int idxCols=0;idxCols<imageSrc.cols;idxCols++){
33                      uint_8_side_channel valIn;
34                      valIn.data = imageSrc.at<unsigned char>(idxRows, idxCols);
35                      valIn.dest = 1;
36                      valIn.keep = 1;
37                      valIn.last = 1;
38                      valIn.strb = 1;
39                      valIn.id = 0;
40                      valIn.user = 1;
41                      inStream << valIn;
42                  }
43          }
44 //      stretch histogram
45 Stretch(inStream, outStream, (unsigned char)22, (unsigned char)154);
46 //47   Fill the stretched image into the image matrix
for(int idxRows=0;idxRows<imageSrc.rows;idxRows++){
48          for(int idxCols=0;idxCols<imageSrc.cols;idxCols++){
49              uint_8_side_channel valOut;
50              valOut = outStream.read();
51              outImage[idxRows][idxCols]=valOut.data;
52          }
53 }
54 saveImages(std::string(OUTPUT_IMAGE), imgCvOut);
55 return 0;
56 }
```

The simulation file is used to import external input images and then convert the images into data streams to be simulated.

The real function, after processing by the function, outputs the result and saves the image.

Then perform C simulation, C synthesis and C/RTL joint simulation in sequence. After the C simulation is completed, you can

In the Csim directory, you can see that the converted images have been generated.

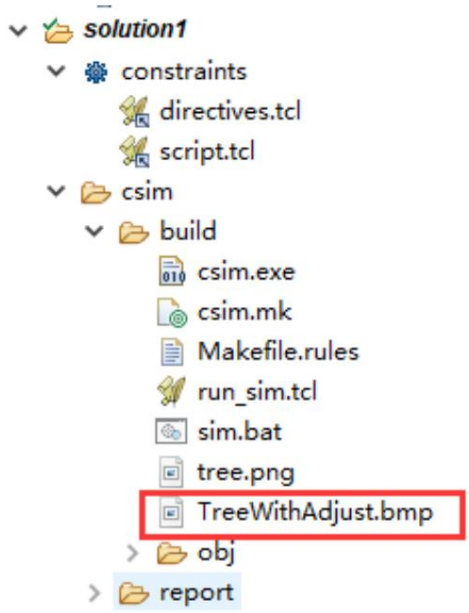


Figure 7- 4

Follow the above steps in sequence. If there are no errors, you can export the IP, and then build this in Vivado.

The hardware platform for the experiment.

7.3 Building the Hardware Platform

In this experiment, we will use the Histogram IP generated in the previous experiment.

Import it into this project, and then create a new ip_repo folder under the Vivado project to save the user

Your own IP.

First, you need to pay attention to the save path of the IP generated by HLS. In the HLS Solution interface, you can

You can see that there is an IP folder under the impl directory under the Solution folder, which contains the files generated by HLS.

If you want to use the generated IP in other projects, just copy this folder to other projects.

In this experiment, we will add the IP generated by HLS in the previous chapter and this chapter to

Add to the ip_repo directory of the vivado project.

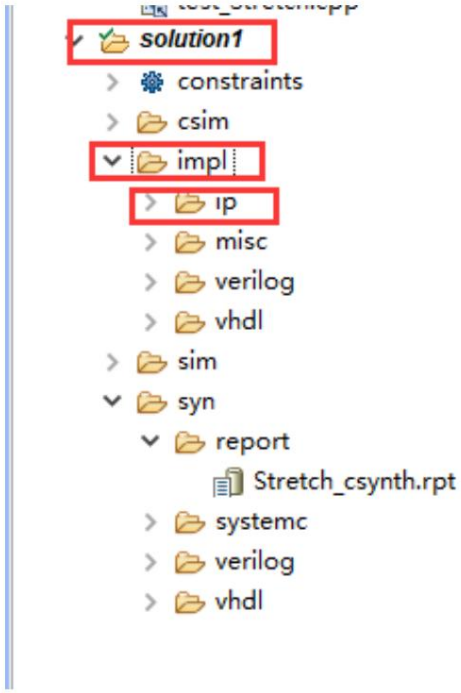


Figure 7- 5

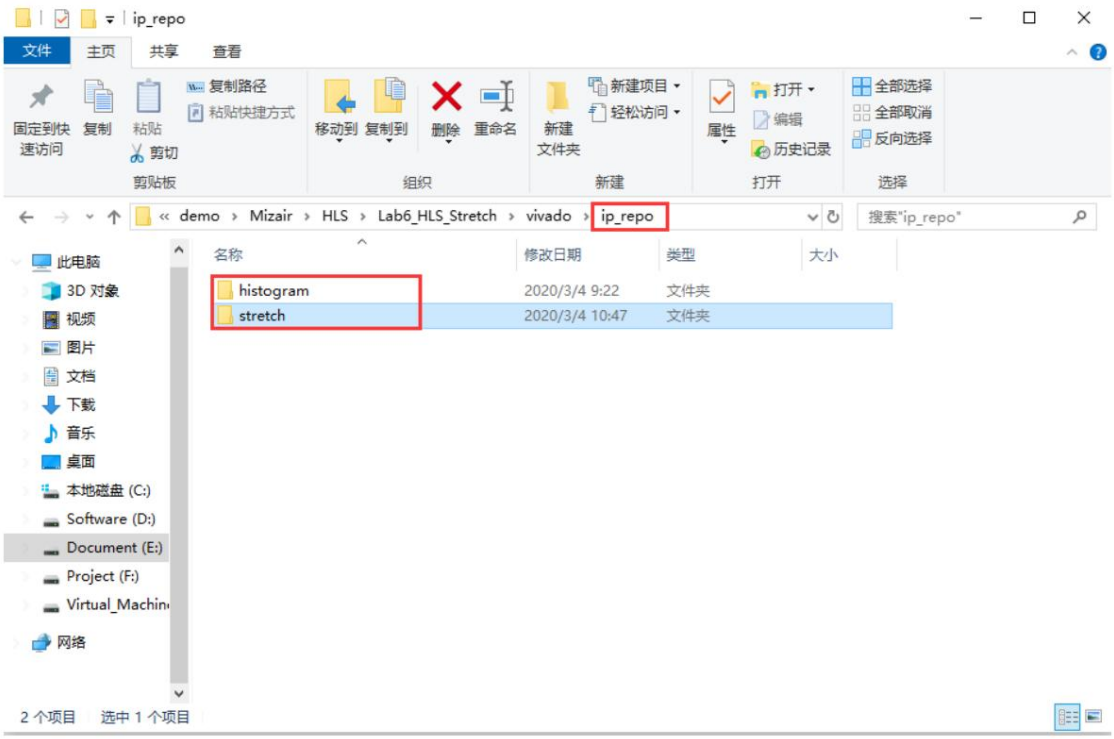


Figure 7- 6

After creating the vivado project, proceed to build the Block design.

The construction method is similar to that in the previous chapter. Here we only briefly explain the connection of several IPs.

In the DMA configuration, you need to enable the read and write channels, as shown in the figure:

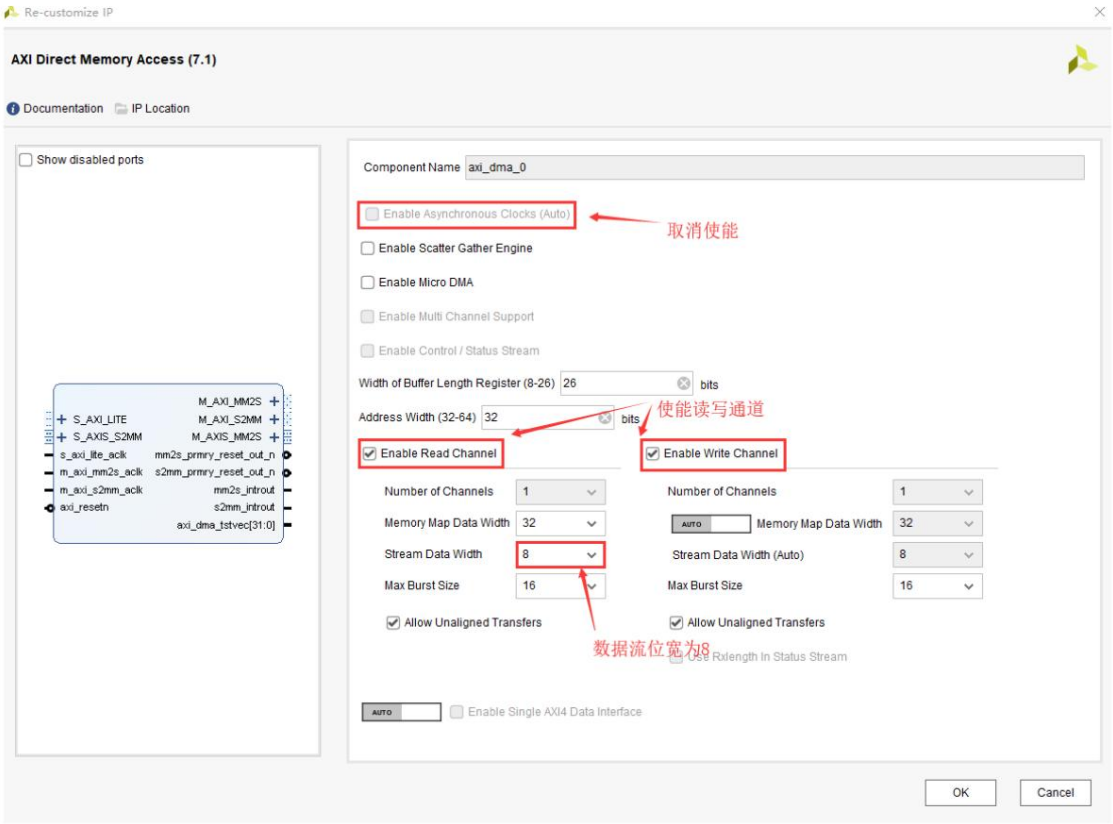


Figure 7- 7

This experiment will use the axi_broadcaster IP, which is used to convert AXI_Stream

The stream can be divided into two completely identical stream signals, similar to broadcasting, outputting the AXI_Stream data stream to

The two HLS IPs are shown in the figure.

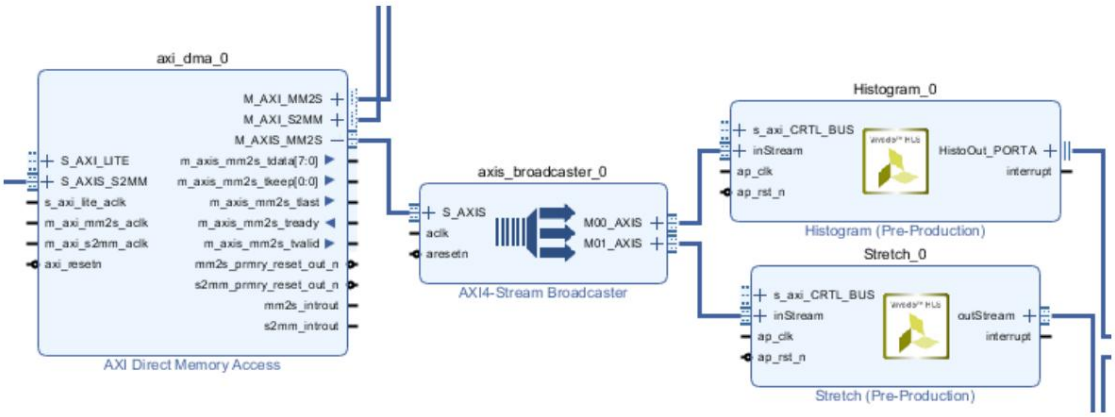


Figure 7- 8

In addition, this experiment adds an axi_timmer IP to calculate the time for comparison through software

And the time required to call IP to calculate the image histogram. After all the connections are completed, it is shown in the figure:

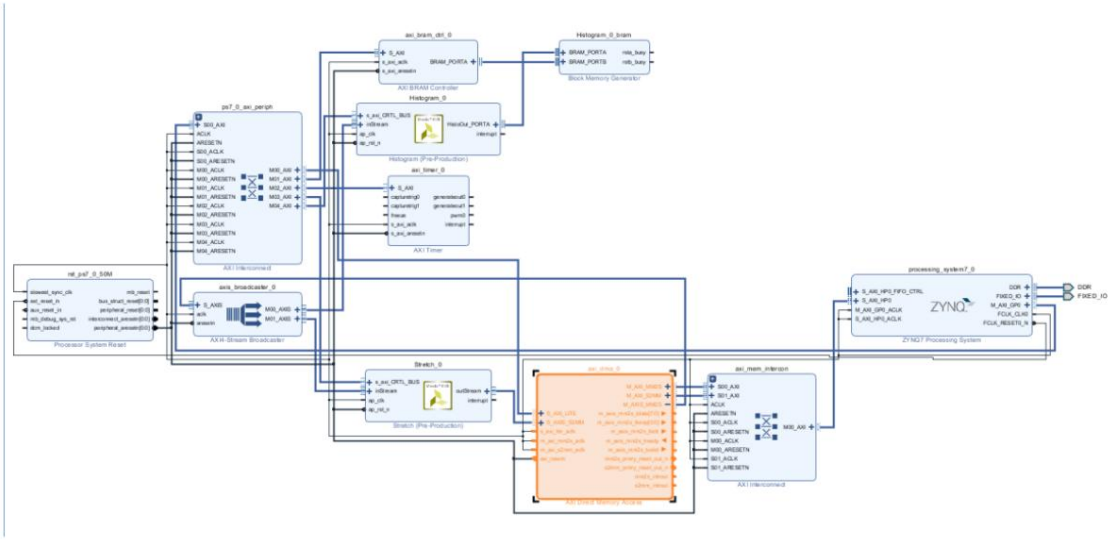


Figure 7- 9

7.4 Create **SDK** Project

In SDK design, firstly, the histogram is obtained by software, and the maximum and minimum grayscale values are found.

The software then stretches the histogram and calculates the time required for the software to complete these operations.

After that, the IP is called to complete the operation to compare the running time of hardware and software.

The running results are shown in the figure: From the results, it can be seen that the histogram calculated by the software and the calling IP

The image interval is consistent, and the pixels of the stretched image are also consistent, but compared with the software, the image generated by calling HLS

IP can complete the histogram stretching much faster than software methods.

Figure 7- 10

7.5 Experimental Summary

In this experiment, a simple histogram stretching experiment was completed. By stretching the histogram, the image

This experiment was completed on the basis of the previous experiment. At the same time, the histogram was also completed.

Finally, the SDK completed the image processing speed of software and hardware.

Compared with the previous example, it is foreseeable that the efficiency of using FPGA for processing larger amounts of data will be much higher.

The efficiency can be greatly improved, which is also the advantage of FPGA.

Chapter 8 HLS Implementation of **2D Convolution**

8.1 Experimental Introduction:

Two-dimensional convolution is widely used in image processing and deep learning.

Mathematical principles and derivations. Interested users can consult relevant materials to understand the origin of two-dimensional convolution.

This is very helpful for understanding the role of convolution. In this experiment, we mainly introduce it in a simple way.

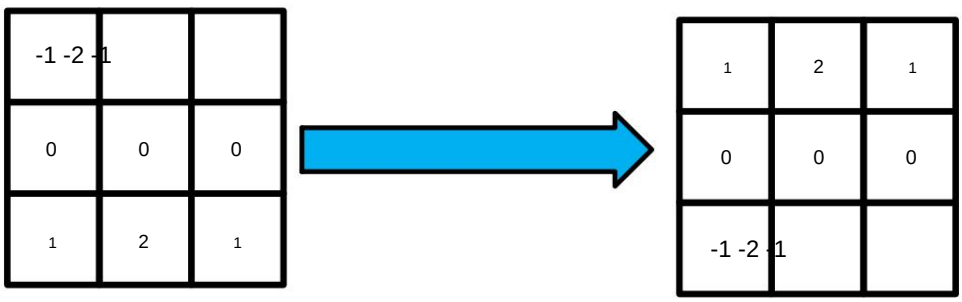
Let's take a look at the implementation process of two-dimensional convolution.

The calculation of two-dimensional convolution can be divided into two major steps: the first is preprocessing, flipping the convolution kernel 180°, and then

The first step is to expand the boundary, usually by padding the boundary with zeros; the second step is to use a sliding convolution kernel to perform convolution budgeting.

As shown in the figure, it is a horizontal flip diagram of a 3x3 Kernel for sobel edge detection.

The kernel is a very important part of the convolution operation.



3X3 convolution kernel, sobel

Flip 180° horizontally

Figure 8- 1

As shown in the figure, this is the padding operation of the original image. The simplest method is to fill a circle of 0s on the periphery.

The purpose of zero padding is to keep the size of the image after the convolution operation consistent with the original image. As shown in the figure,

Fill a 5x5 image with a circle of zeros.

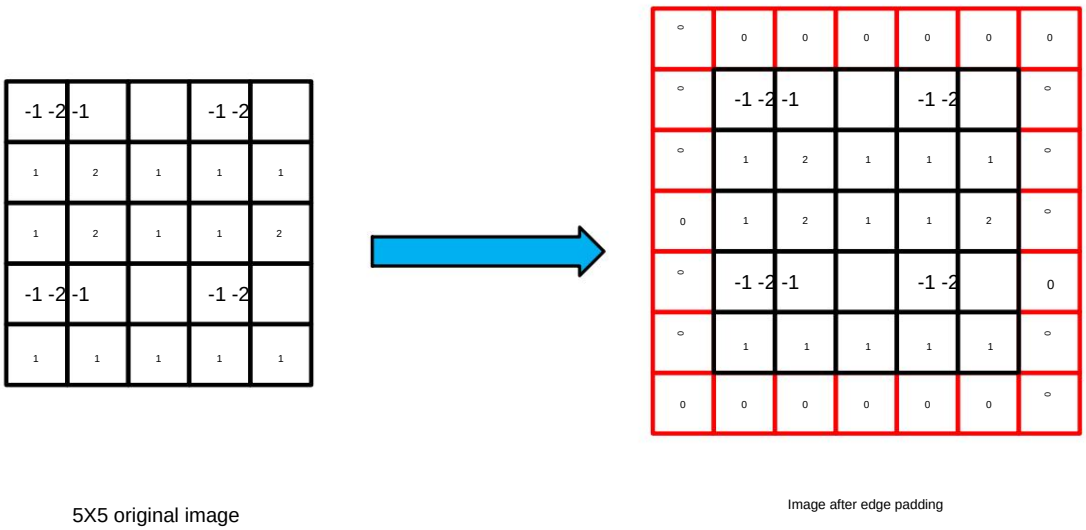


Figure 8- 2

The sliding convolution operation is actually to place the convolution kernel in the image and

The convolution kernel is continuously slid so that the data of the entire image can be affected by the convolution window.

The figure below shows the process of Sobel convolution. The convolution kernel (window) moves on the image in sequence.

The convolution is performed at the places where the kernel overlaps with the image, and then the result is filled in the middle pixel position.

As shown in the figure, because a circle of 0s is filled in the periphery of the original image, the image after convolution is consistent with the original image.

Keep it the same size, still 5x5.

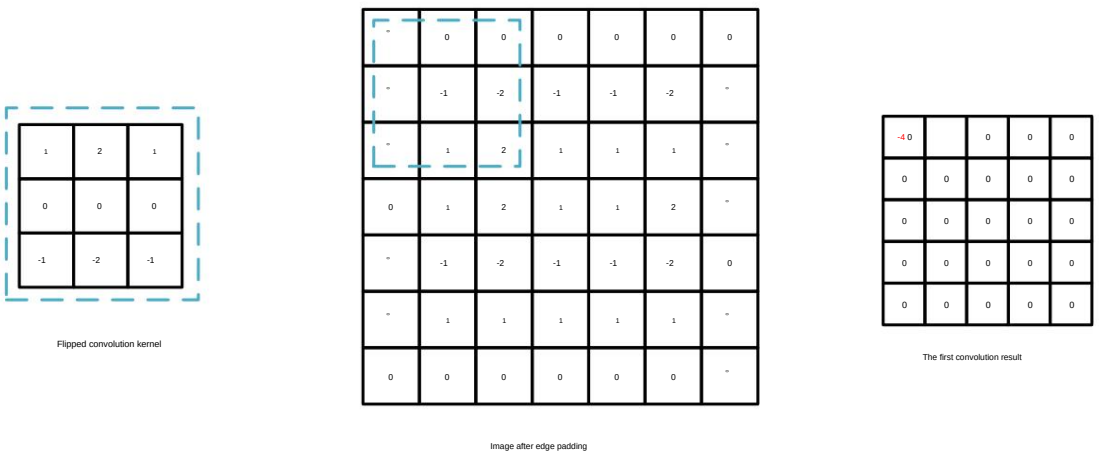


Figure 8-3 The first convolution operation

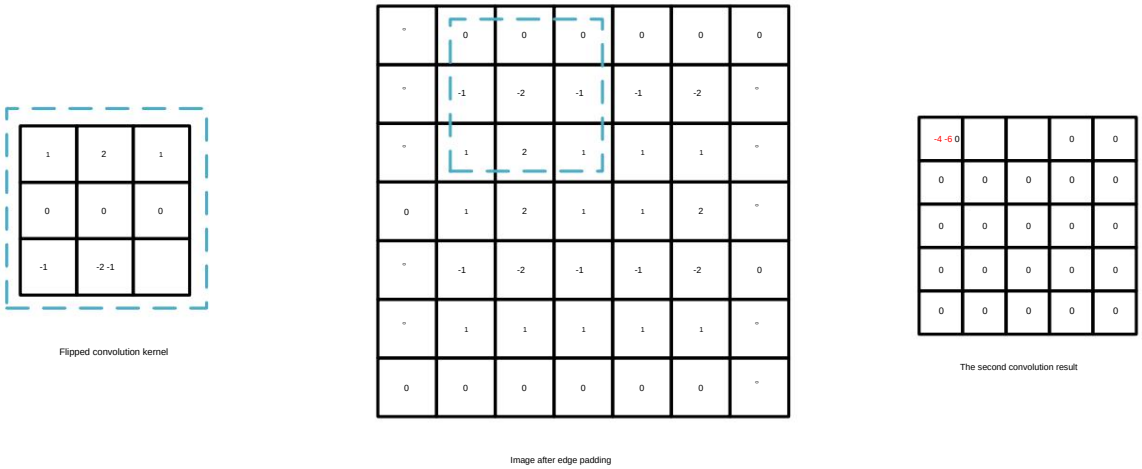


Figure 8-4 The second convolution operation

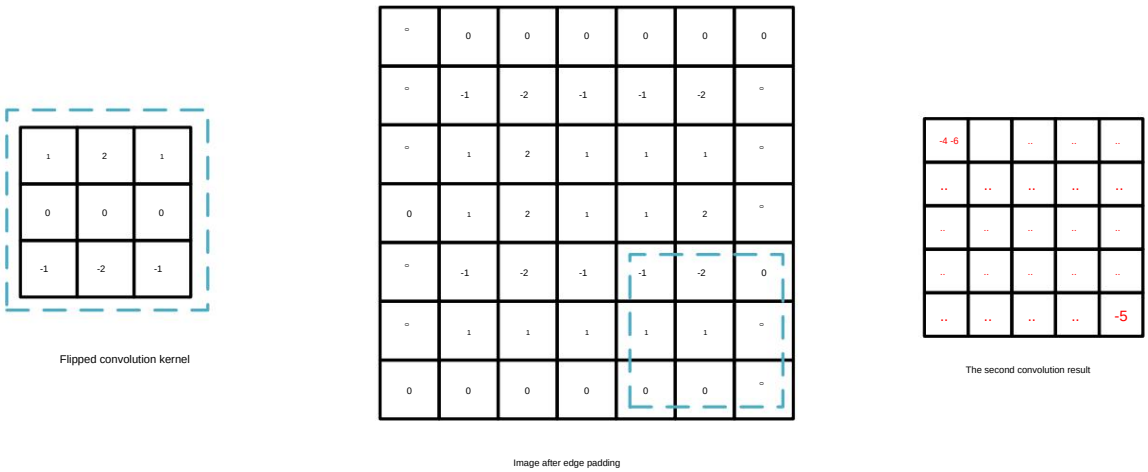


Figure 8-5 The last convolution operation

8.2 A brief introduction to image erosion and dilation

The following is a brief introduction to the knowledge related to image erosion and expansion, and uses Matlab to directly

The effects of dilation and erosion are as follows:

sound, isolate independent image elements, and join adjacent elements to find the clear

The obvious maximum or minimum area.

8.2.1 Image Erosion

Image erosion is simply to find the smallest minimum value in the area. The brightness of the image after erosion is

Before the corrosion, the image is reduced. Use the following script in matlab to quickly observe the corrosion of the image.

The result afterwards.

```
img_rgb=imread('Lena.bmp'); %Load the image in the path img_gray=rgb2gray(img_rgb);
imtool(img_gray); se=strel('disk',1); %Generate a
circular structural element %Use
the generated structural element to erode the image
I2=imerode(img_gray,se); imtool(I2);
```

The result after running is shown in the figure. The left side is the original image, and the right side is the image after corrosion.

It is still possible to see the difference between the eroded image and the original image. Compared with the original image, the noise of the image

Significantly reduced and brightness reduced.

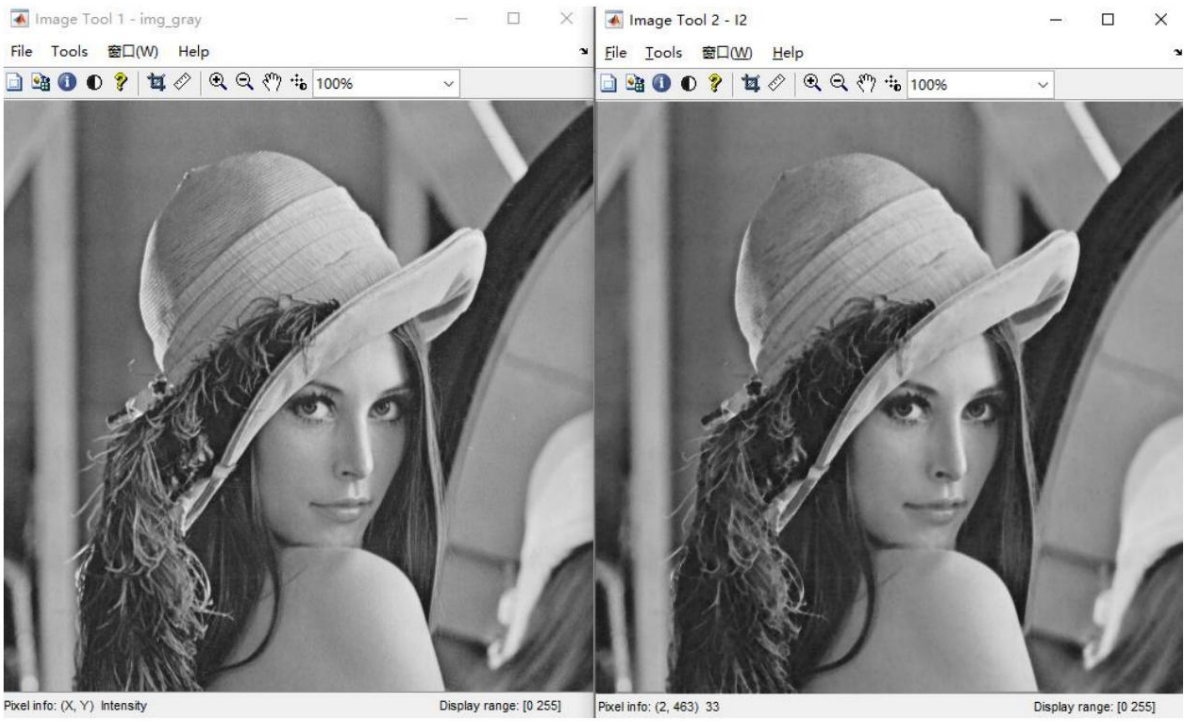


Figure 8-6 Image erosion

In this experiment, it is equivalent to sliding a 3x3 convolution window onto the image and returning this convolution window.

The minimum value in the mouth is used as the value of the middle pixel.

8.2.2 Image Dilation

Dilation has the opposite effect to erosion, which is to find the maximum value in the area. After dilation, the brightness of the image

It will be larger than the original image. You can quickly get the expanded result by running the script in Matlab.

```
img_rgb=imread('Lena.bmp'); %Load the image in the path
img_gray=rgb2gray(img_rgb);
imtool(img_gray);
se=strel('disk',1); %Generate a circular structural element
I2=imdilate(img_gray,se);                %Use the generated structural element to erode the image
imtool(I2);
```

The result after running is shown in the figure: You can see that the white part of the image expands outward, and the brightness of the image

An improvement compared to before.

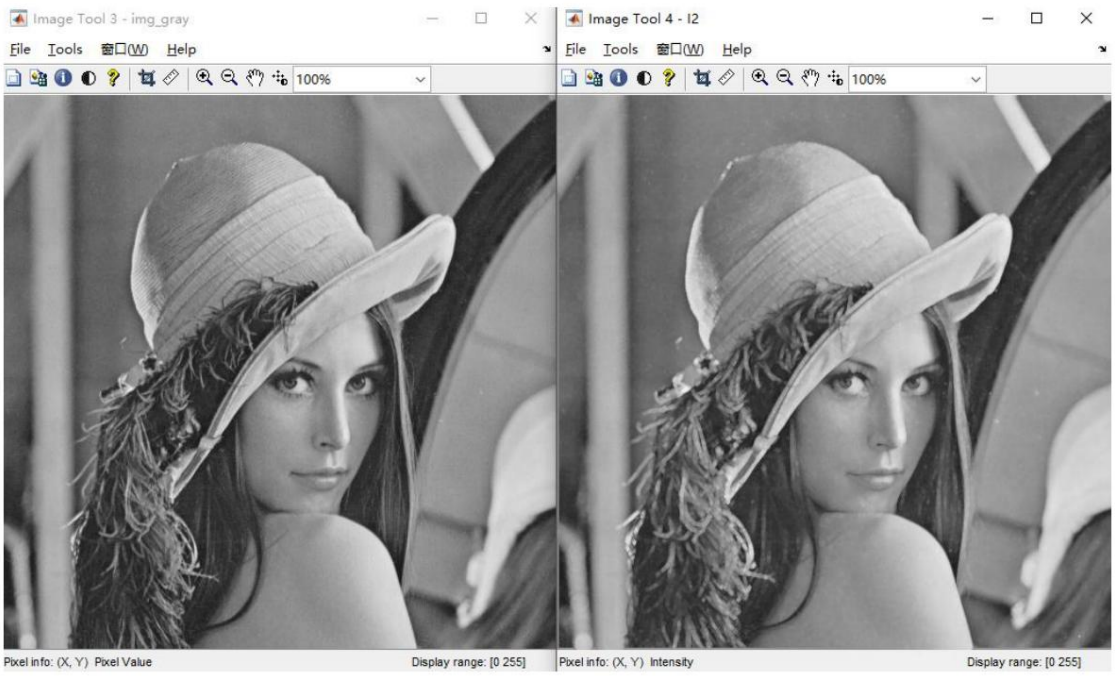


Figure 8-7 Image expansion

In this experiment, it is equivalent to sliding a 3x3 convolution window onto the image and returning this convolution window.

The minimum value in the mouth is used as the value of the middle pixel.

8.3 Sobel Edge Detection

Edge detection is a method often used in industrial production, often used for positioning, extracting wheels

In this chapter, we will use FPGA to complete a simple edge detection algorithm - Sobel edge detection

Algorithm. First, you need to understand the following Sobel operator.

Sobel Operator

The Sobel operator is mainly used for edge detection. Technically, it is a discrete

The difference operator is used to calculate the approximate grayscale value of the image brightness function. Use this operator at any point in the image.

The corresponding grayscale vector or its normal vector will be generated.

The convolution factor of the Sobel operator is as follows:

Sobel卷积因子为：

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

When using this operator, multiply the operator with the corresponding pixel in the image and then sum them up to get

To the gradient value of the image in the horizontal and vertical directions. For example, use this operator with the following 3X3 pixel window

Row Convolution

A1	A2	A3
B1	B2	B3
C1	C2	C3

The gradient calculation results in the horizontal and vertical directions are as follows:

Dx= (A3-A1) + 2*(B3-B1) + (C3-C1);

Dy= (A1-C1) + 2*(A2-C2) + (A3-C3);

The value of the pixel window of the image in the figure represents the grayscale value of the image. The gradient obtained by using this operator is

The final result is relatively simple. After obtaining the gradient, the following formula needs to be used to determine the gradient of the point:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

In general, for the convenience of calculation, the following formula is generally used to calculate the gradient of the image:

$$|G| = |Gx| + |Gy|$$

If the final gradient G obtained is greater than a certain threshold, it can be determined that the point is the edge of the image.

8.4 **Linebuffer** and **Window** Classes in the HLS Video Library

In this experiment, we will not use the edge detection and dilation erosion functions that come with HLS, but instead use

Use the Linebuffer and Window classes in the HLS video library to complete the development of related image algorithms.

You can also directly call HLS ready-made functions to complete image processing related operations.

8.4.1 A brief introduction to Linebuffer

You can get the Xininx introduction to Linebuffer by referring to the manual UG902.

The LineBuffer class is a C++ class that allows users to easily declare and manage line buffers in algorithmic code.

Buffer. This class provides all the methods needed to instantiate and use a line buffer. The LineBuffer class can

Handles all data types.

Linebuffer mainly has the following 5 methods:

- shift_pixels_up()
- shift_pixels_down()
- insert_bottom_row()
- insert_top_row()
- getval(row,column)

Figure 8-8 Common methods of Linebuffer

It is easy to understand the function of each method based on the names of these methods. Shift_pixel_up() is to The pixels in the specified column move up by one. Insert_bottom_row() adds a pixel to the last row of the specified column. Getval(row,column) can return the value of the specified pixel.

The following example briefly explains how to declare a Linebuffer object and use the above method.

hls::LineBuffer<rows, columns, type> variable; is the template of Linebuffer class. Use this template to generate a line buffer.

For example, hls::LineBuffer<3,5,unsigned char> Buff_A; declares a buffer with 3 rows and 5 columns. The buffer contains data of type unsigned char.

Here is a simple example of using the shift_down and insert_top methods on this image:

```
Buff_A.shift_pixels_down(2);
```

```
Buff_A.insert_top_row(100,2);
```

Table 2-8: **Data Set After Shift Down and Insert Top Classes Used**

Line	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	1	2	100	4	5
Row 1	6	7	3	9	10
Row 2	11	12	8	14	15

Figure 8- 9

As shown in the figure above, after using the shift_pixels_down and insert_top_row methods, in the zeroth row and

A new number 100 is added to the second column.

Other methods related to Linebuffer are also described in detail in the manual. Users can refer to UG902

Linebuffer This section will show you other methods. In fact, you can also easily find them by the names of the method functions.

It is easy to understand what the function does.

8.4.2 A brief introduction to the Window class

In addition to the row buffer, HLS also provides a window buffer to facilitate users to perform volume

Similar to Linebuffer, the window class also has the following methods:

- shift_pixels_up()
- shift_pixels_down()
- shift_pixels_left()
- shift_pixels_right()
- insert_pixel(value,row,colum
- insert_row()
- insert_bottom_row()
- insert_top_row()
- insert_col()
- insert_left_col()
- insert_right_col()
- getval(row, column)

It is also easy to know the function of the function by the method name. The following is a more commonly used one

Method, insert_pixel(value,row,column); This method is used to fill a pixel at the specified position in the window.

number.

First, define a variable of the window class using the following template:

```
hls::Window<row, column, type> variable;
```

The following takes a 3x3 char type window as an example:

```
hls::window<3,3,char> window_B;
```

The result after window_B calls the method to insert data is as follows:

```
Buff_B.insert_pixel(100,1,1);
```

Table 2-15: **Memory Window Data Set After Insertion Operation at Location 1,1**

Column 0	Column 1	Column 2	Row
1	2	3	Row 0
6	100	8	Row 1
11	12	13	Row 2

Figure 8-10 Inserting the value 100 in row 1, column 1

I won't explain the other methods of the window class in detail, but they are easy to understand.

If the user has any questions, just refer to the manual UG902.

8.5 HLS Implementation of 2D Convolution

8.5.1 Adding header files

Next, we implement two-dimensional convolution in HLS. First, we need to create a header file, which contains some commonly used numbers.

Redefine by institution type.

```
1 #ifndef __TEST_HEAD_H__
2 #define __TEST_HEAD_H__
3
4 #include <hls_video.h>
5 #include <ap_axi_sdata.h>
6
7 #define KERNEL_DIM 3
8 #define IMG_WIDTH 512
9 #define IMG_HEIGHT 512
10 #define INPUT_IMAGE "Lena.bmp"
11 #define OUTPUT_IMAGE "LenaWithAdjust.bmp"
12
13 //14Defining the data in a data flow
typedef ap_axiu<8,1,1,1> 15 typedef          uint8_side_channel;
ap_axis<8,1,1,1>                              int8_side_channel;
16 //17definitionAXIÿStream      Data Flow
typedef hls::stream<uint8_side_channel> 18 //19 typedef                    AXI_STREAM;
      definitionchar   Type of convolution window
hls::Window<KERNEL_DIM,KERNEL_DIM,char>                                    CHAR_WINDOW;
```

```
20 //  definitionshort  Type of convolution window
21 typedef hls::Window<KERNEL_DIM,KERNEL_DIM,short> 22 //  SHORT_WINDOW;
       definitionLineBuffer 23 For delay
typedef hls::LineBuffer<KERNEL_DIM,IMG_WIDTH,unsigned char> BUFFER;
24
25 //  Reference Function
26 short convolved2d(CHAR_WINDOW * window,char kernel[KERNEL_DIM * KERNEL_DIM]);
27
28
29 //  Find the convolution sum
30 short sumWindow(SHORT_WINDOW *window);
31 //  Find the minimum value in the convolution window
32 short minWindow(SHORT_WINDOW *window);
33 //  Find the maximum value in the convolution window
34 short maxWindow(SHORT_WINDOW *window);
35
36 //  Top-level functions
37 void imagePrco(AXI_STREAM & inStream,AXI_STREAM & outStream, char
kernel[KERNEL_DIM*KERNEL_DIM], int operation);
38
39
40 #endif
```

Line 7 defines the size of the convolution kernel, lines 8 and 9 define the size of the image to be processed, and lines 10 and

Line 11 defines the path of the input and output images, lines 14 and 15 define the pixels in the stream, and line 17 defines

The type of data stream, lines 19 and 21 define two window class objects, and line 23 defines linebuffer

Lines 29 to 34 declare the operations on the data in the convolution kernel, where sumWindow

Used to find the sum in the convolution window. minWindow is used to find the minimum value of the convolution window (Erode erosion).

maxWindow is used to return the maximum value in the convolution window (dilate expansion).

Line 37 declares the top-level function for this experiment. To change the functionality of the top-level function, you only need to change the input

Just input the convolution kernel and the selected operation to the top-level function.

8.5.2 Add source files

In the source file, add the following content to implement the top-level function.

```
1 #include "kernel.h"
2
```

```
3           void imagePrco(AXI_STREAM & inStream,AXI_STREAM & outStream, char
kernel[KERNEL_DIM*KERNEL_DIM], int operation)
4 {
5 #pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
6 #pragma HLS INTERFACE s_axilite port=operation bundle=CTRL_BUS
7 #pragma HLS INTERFACE s_axilite port=kernel bundle=KERNEL_BUS
8 #pragma HLS INTERFACE axis register both port=outStream
9 #pragma HLS INTERFACE axis register both port=inStream
10
11          BUFFER lineBuf;
12          SHORT_WINDOW window;
13
14          //  Index values used to label rows and columns
15          int idxCol = 0;
16          int idxRow = 0;
17          int pixConvolved = 0;
18
19          //  Calculate delay, fill      line_buffer
20          int waitPixel = (IMG_WIDTH *(KERNEL_DIM-1) + KERNEL_DIM)/2;
21          int cntWait = 0;
22          int sentPixel = 0;
23
24          uint8_side_channel dataOutChannel;
25          uint8_side_channel currentChannel;
26
27          //  Iterate over all input pixels
28          for(int idxPix=0;idxPix<(IMG_HEIGHT*IMG_WIDTH);idxPix++)
29          {
30 #pragma HLS PIPELINE
31
32                  currentChannel = inStream.read();
33                  unsigned char inPix = currentChannel.data;
34
35                 //put data on the line buffer
36                  lineBuf.shift_pixels_up(idxCol);
37                  lineBuf.insert_bottom_row(inPix, idxCol);
38
39                 //put data on the window, and multiply by kernel
40                  for(int idxWinRow = 0; idxWinRow < KERNEL_DIM; idxWinRow++)
41                  {
42                          for(int idxWinCol=0; idxWinCol < KERNEL_DIM; idxWinCol++)
```

```
43                          {
44                              // idxWinCol and pixConvolved will slide the window
45                              short val = (short)lineBuf.getval(idxWinRow, idxWinCol +
pixConvolved);
46                              //  The data in the sampling window is Kernel(  Convolution operator )multiplication
47                              val = (short)kernel[(idxWinRow                 *  KERNEL_DIM) + idxWinCol]                 *  val;
48                              window.insert(val,idxWinRow,idxWinCol);
49                          }
50                  }
51
52
53                  short valOut=0;
54
55                  //  Avoid calculating image boundaries)
56                  if((idxRow >= KERNEL_DIM-1) && (idxCol >= KERNEL_DIM -1))
57                  {
58
59                          switch(operation)
60                          {
61                          case 0:
62                          {
63                              valOut = sumWindow(&window);
64                              if(valOut < 0 ){
65                                      valOut = 0;
66                              }
67                              break;
68                          }
69
70                          case 1:
71                          {
72                              valOut = minWindow(&window);
73                              break;
74                          }
75
76                          case 2:
77                          {
78                              valOut = maxWindow(&window);
79                              break;
80                          }
81                          default:;
82                          }
```

```
83                      pixConvolved ++;
84                  }
85
86                  cntWait++;
87                  if(cntWait > waitPixel){
88                      dataOutChannel.data = (unsigned char)valOut;
89                      dataOutChannel.dest = currentChannel.dest;
90                      dataOutChannel.id = currentChannel.id;
91                      dataOutChannel.keep = currentChannel.keep;
92                      dataOutChannel.strb = currentChannel.strb;
93                      dataOutChannel.user = currentChannel.user;
94                      dataOutChannel.last = 0;
95                      outStream.write(dataOutChannel);
96                      sentPixel++;
97                  }
98                  //  Calculate row and column index values
99                  if(idxCol < IMG_WIDTH - 1){
100                     idxCol ++;
101                 }
102                 else{
103                     idxCol = 0;
104                     idxRow++ ;
105                     pixConvolved = 0;
106                 }
107                 //send data to the stream
108
109             }
110             for(cntWait = 0; cntWait < waitPixel; cntWait++){
111             dataOutChannel.data = 0;
112             dataOutChannel.dest = currentChannel.dest;
113             dataOutChannel.id = currentChannel.id;
114             dataOutChannel.keep = currentChannel.keep;
115             dataOutChannel.strb = currentChannel.strb;
116             dataOutChannel.user = currentChannel.user;
117             if(cntWait < waitPixel - 1)
118                 dataOutChannel.last = 0;
119             else
120                 dataOutChannel.last = 1;
121             outStream.write(dataOutChannel);
122             sentPixel++;
123             }
```

```
124
125 }
126
127
128 //  Find the convolution sum
129 short sumWindow(SHORT_WINDOW *window){
130         short sum=0;
131         for(int idxRow = 0; idxRow < KERNEL_DIM; idxRow++){
132                 for(int idxCol = 0;idxCol < KERNEL_DIM;idxCol++){
133                         sum += (short)window->getval(idxRow, idxCol);
134                 }
135         }
136         return sum;
137
138 }
139 //  Find the minimum value in the convolution window
140 short minWindow(SHORT_WINDOW *window){
141         unsigned char minVal = 255;
142         for(int idxRow = 0; idxRow < KERNEL_DIM; idxRow++){
143                 for(int idxCol = 0; idxCol < KERNEL_DIM; idxCol++){
144                         unsigned char valInWindow;
145                         valInWindow = window->getval(idxRow, idxCol);
146                         if(valInWindow < minVal){
147                                 minVal = valInWindow;
148                         }
149                 }
150         }
151         return minVal;
152 }
153 //  Find the maximum value in the convolution window
154 short maxWindow(SHORT_WINDOW *window){
155         unsigned char maxVal = 0;
156         for(int idxRow=0;idxRow<KERNEL_DIM;idxRow++){
157                 for(int idxCol=0;idxCol<KERNEL_DIM;idxCol++){
158                         unsigned char valInWindow;
159                         valInWindow = window->getval(idxRow, idxCol);
160                         if(valInWindow > maxVal){
161                                 maxVal = valInWindow;
162                         }
163                 }
164         }
```

```
165         return maxVal;
166 }
```

In the source file, lines 5 to 9 declare the interface type that defines the function. When generating IP through synthesis,

The for loop in line 28 performs a loop on all input image pixels.

Iterate, lines 36 and 37 call the linebuffer method to fill the input data into the Linebuffer

In the buffer, lines 40 to 50 complete the convolution window corresponding to the kernel and the corresponding pixel multiplication, and

Fill the product in the corresponding position of the convolution window. Lines 56 to 84 perform the convolution operation to prevent the calculation graph from

Lines 87 to 97 give the result of the convolution calculation to the output data stream.

Lines 110 to 125 complete the missing images.

The final image will have two rows and two columns less than the original image.

When the pixels are completed, a warning will be given as shown in the figure, which will eventually lead to calling the generated IP in the SDK

It will not work. The warning shown in the figure is usually caused by the mismatch of pixels in the image.

The input and output data streams need to have signals such as tlast and tuser. If they are not specified, this problem will occur.

```
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
WARNING: Hls::stream 'hls::stream<ap_axiu<8, 1, 1, 1> >.2' is read while empty, which may result in RTL simulation hanging.
```

Figure 7- 11

In lines 117 to 120, the last pixel of the entire data stream is judged and marked.

This is to match the AXI-stream interface protocol.

8.5.3 Add **C** simulation file

The simulation files for this experiment are similar to those in the previous chapters.

There are several convolution kernels used that are briefly explained.

1) Impulse Kernel

0	0	0
0	1	0
0	0	0

This kernel does not process the image in any way and simply outputs the original input image.

Select the Kernrl and set operation to 0. The image after two-dimensional convolution is the grayscale image of the original image.

As shown in the figure:

Figure 8-12 Image after convolution

Figure 8-13 Grayscale image

A closer look reveals that the simulated image of C has two fewer rows and two fewer columns. This is consistent with the previous

The code is consistent with what is mentioned above, because no zero padding operation is performed on the original image, which will result in a smaller output image.

The original image is missing two rows and two columns.

2) Kernel for Erode or Dilate

1	1	1
1	1	1
1	1	1

If you select this kernel and set the operation to 1 or 2, you can dilate and erode the image.

The function of this kernel is to find the minimum and maximum values in the convolution domain.

The image after corrosion and the image after expansion:

Figure 8-14 Image after corrosion

Figure 8-15 Image after dilation

3) Sobel kernel

-1 -2 -1		
0	0	0
1	2	1

The vertical sobel kernel is shown above. If you select this kernel and set the operation to

0, that is, to obtain the sum in the convolution window and determine the edge of the image once.

The C simulation results after using this kernel and setting operation to 0 are as follows:

Figure 8-16 Convolution result of sobel kernel

4) Edge detect Kernel

In addition to the Sobel edge detection operator introduced above, here is another edge detection operator.

-1 -1 -1		
-1 8 -1		
1 -1 1		

Using this operator and setting operation to 0, the results of C simulation are as follows:

Figure 8- 17 Edge detection

It can be seen that compared with Sobel edge detection, this operator is better at detecting edges.

The Sobel operator for a surface is only an operator in the vertical direction.

8.6 Experimental Summary

In this chapter, the HLS implementation of some simple image algorithms is completed, including image erosion and dilation.

And simple edge detection, etc. Introduced two important classes in HLS image processing, Linebuffer

And Window, by using these two classes, it can help users complete the development of algorithms related to two-dimensional convolution.

After completing this experiment, you should export the IP. In the next experiment, we will use the IP generated in this experiment.

The IP is created and called in Vivado and SDK to display the image on the monitor through HDMI.

In this experiment, the most important part is the theoretical implementation of the two-dimensional convolution introduced at the beginning.

By understanding the process of convolution, we can improve our understanding of image processing algorithms.

efficiency.

There are still some optimization issues in this experiment, and the author will gradually improve them in subsequent versions.

By adding Directives, the array for loop is optimized to improve the efficiency of the IP core.

The IP can be used normally.

Chapter 9 Comprehensive Experiments on Image Erosion, Dilation, and Edge Detection

9.1 Experimental Introduction

In the last experiment, the algorithm development was completed in HLS and the IP core was generated.

In the experiment, the IP generated in the last experiment is called, so that the image processing operations can be intuitively displayed on the

on an HDMI display.

In this experiment, VDMA will be used to quickly display images on the monitor.

9.2 Building **Block Design**

9.2.1 Video output section

1) Add **VDMA**

Search VDMA in the search bar, and then add VDMA to Block Design.

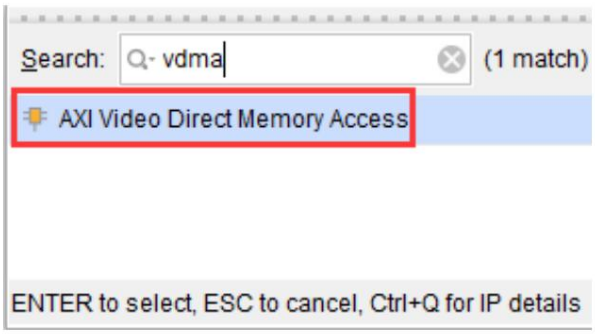


Figure 9-1

After adding VDMA, configure VDMA as follows:

Cancel the write channel enable, set the read channel data stream width to 32 bits, and set the linebuffer depth to

2048ÿ

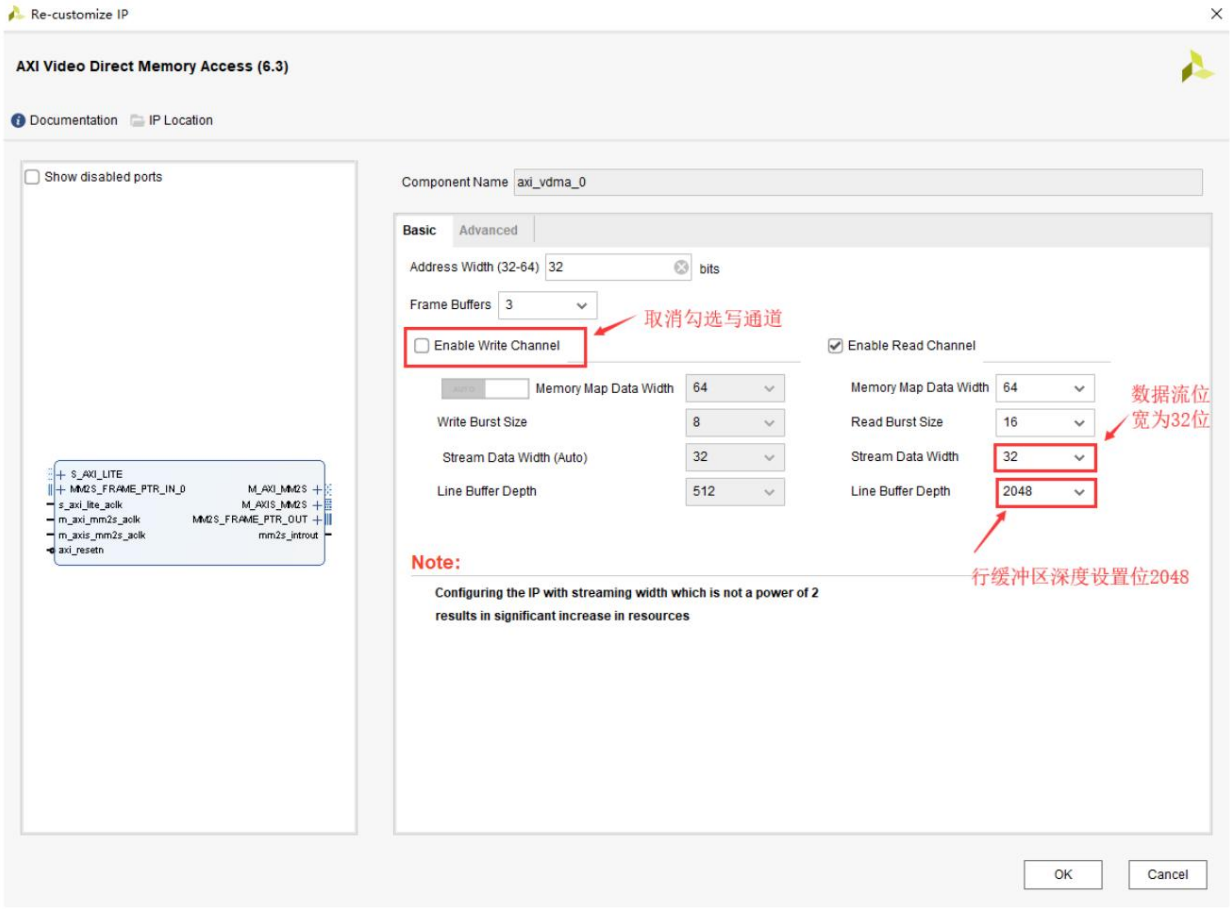


Figure 9-2 VDMA Configuration

2) Video out IP

Add video out IP to provide output video signal through this IP and set it to independent clock mode.

Mode.

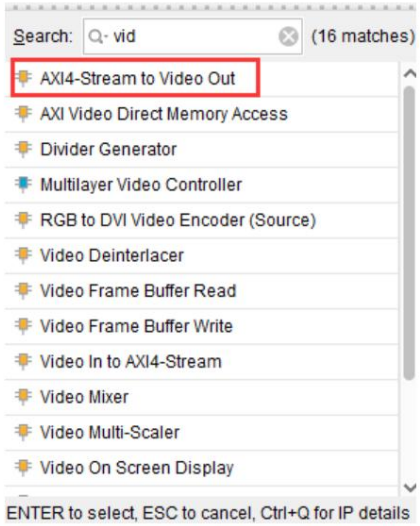


Figure 9- 3

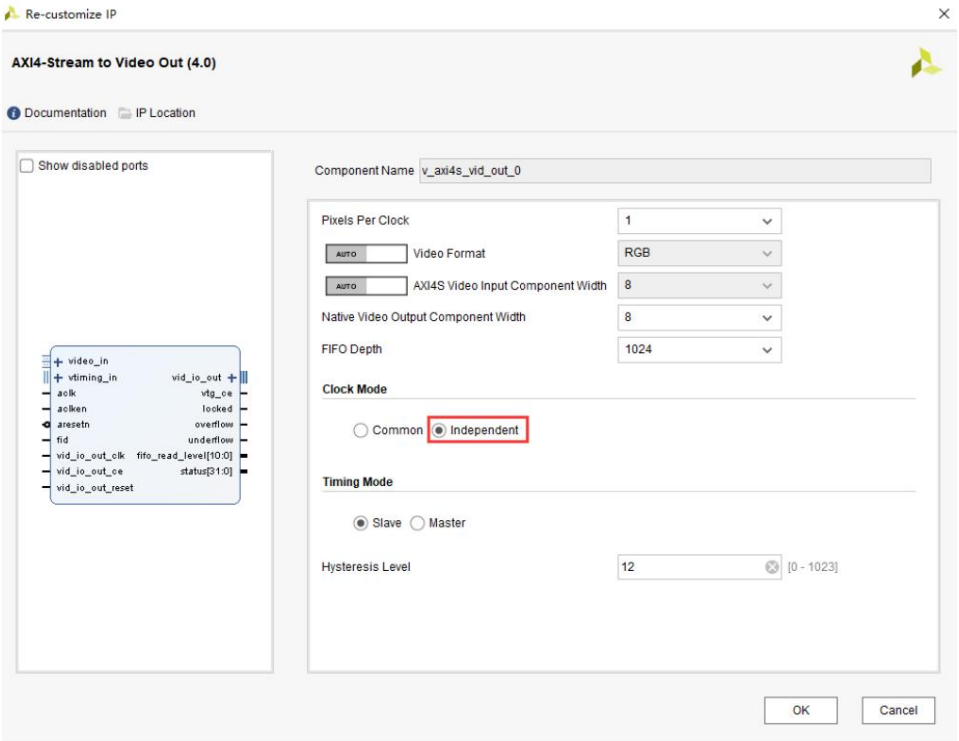


Figure 9- 4

Then connect the VDMA domain video out IP.

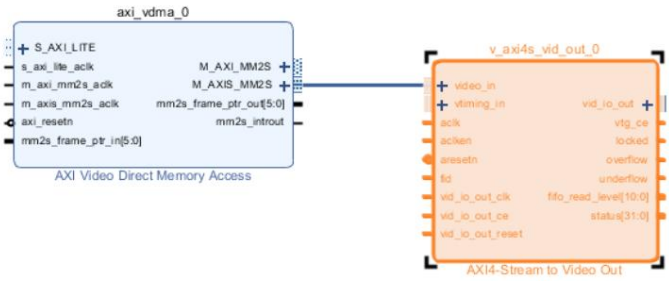


Figure 9-5

3) Video timing IP

Add Video Timing Controller IP to provide synchronization signals for output video output.

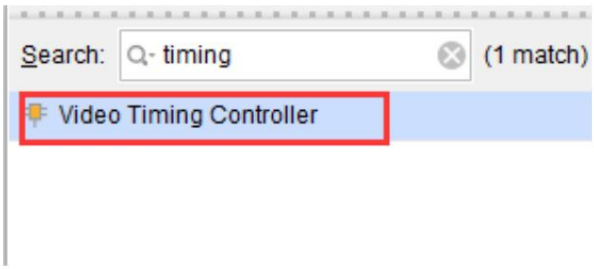


Figure 9-6

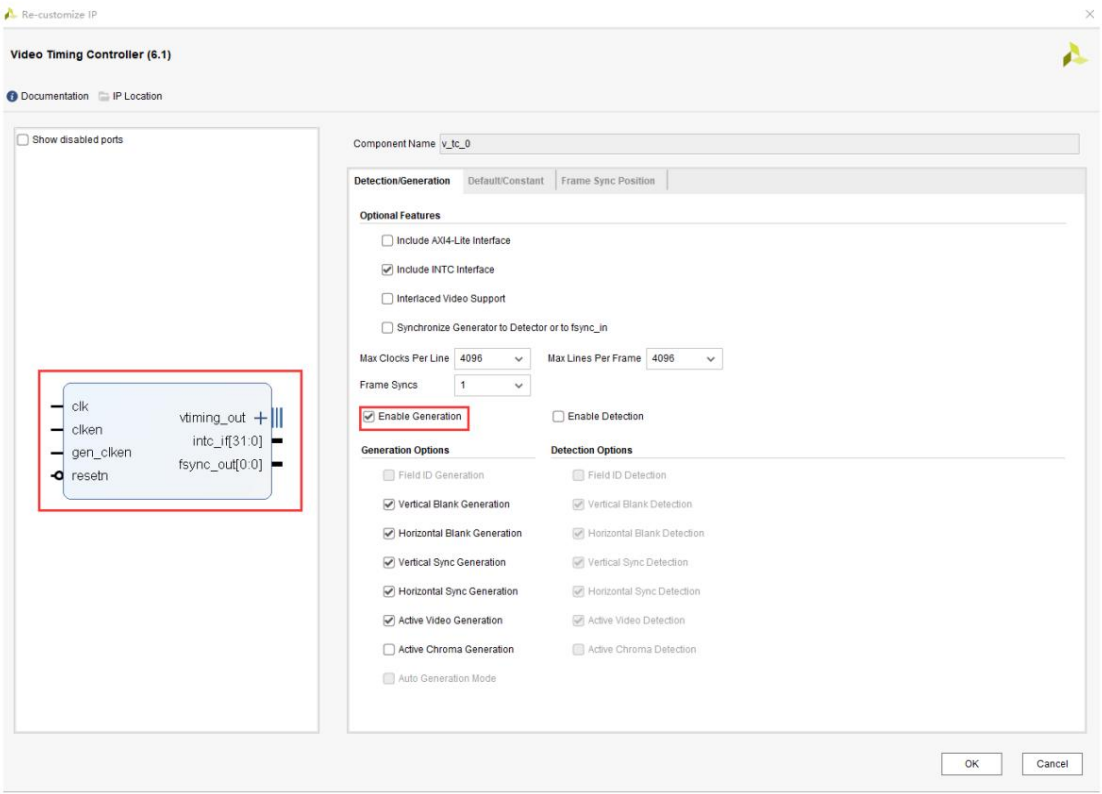


Figure 9-7

Then connect the output timing signal to the timing interface of the Video Out IP.

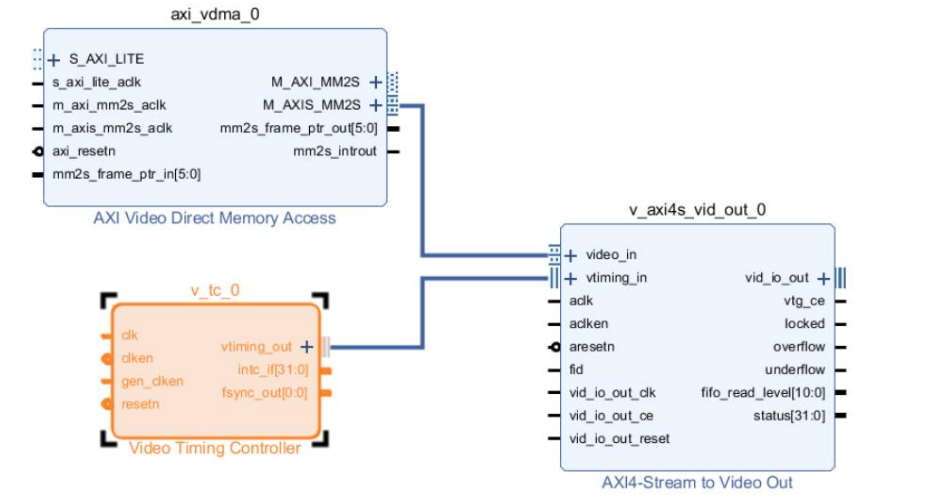


Figure 9- 8

4) Add **HDMI** output **IP**

HDMI output IP completes the encoding of parallel data and sends it to the display in serial mode

This is a custom IP. In this experiment, a pre-packaged IP is used.

In the directory, users only need to add the IP to the warehouse to use the IP.

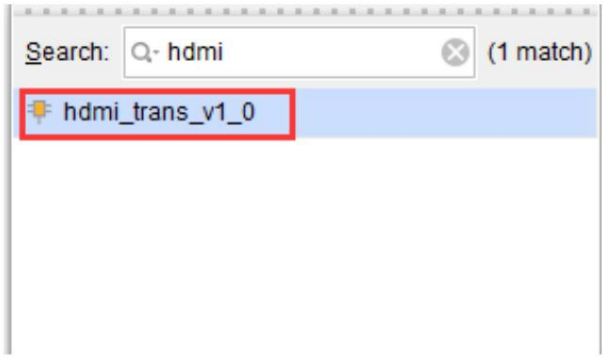


Figure 9- 9

Then the signal output by the video out IP is sent to the HDMI output module.

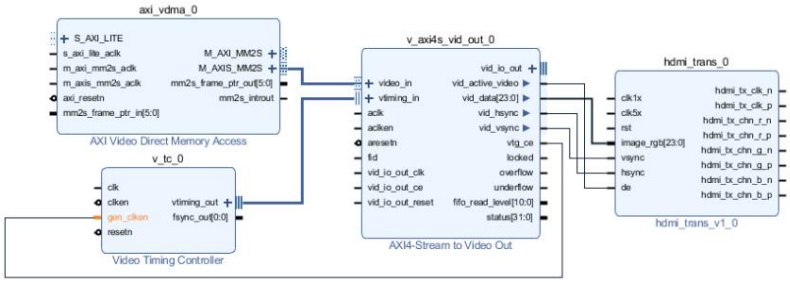


Figure 9- 10

5) Add **PLL**

Since the HDMI output module requires two clocks, one is the pixel clock and the other is the serial output

The required clock, so a PLL is needed to generate a clock of the corresponding frequency.

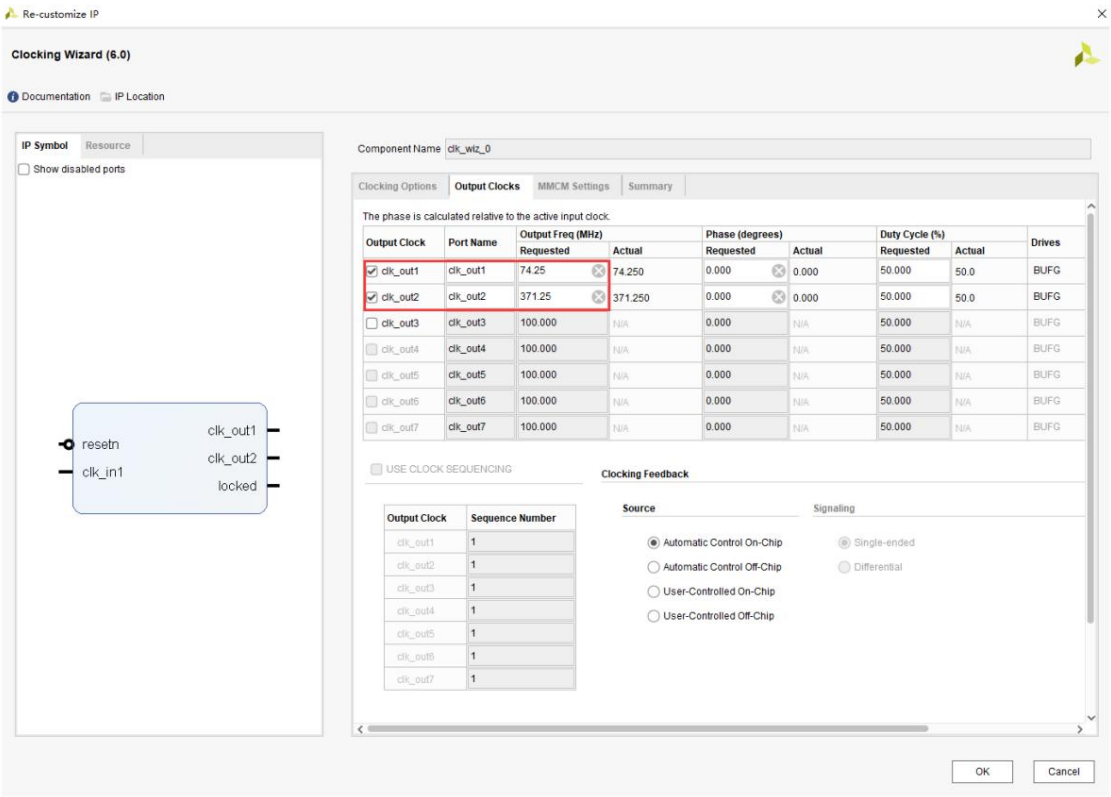


Figure 9-11

Next, connect the clock and reset signals of the IP related to image output

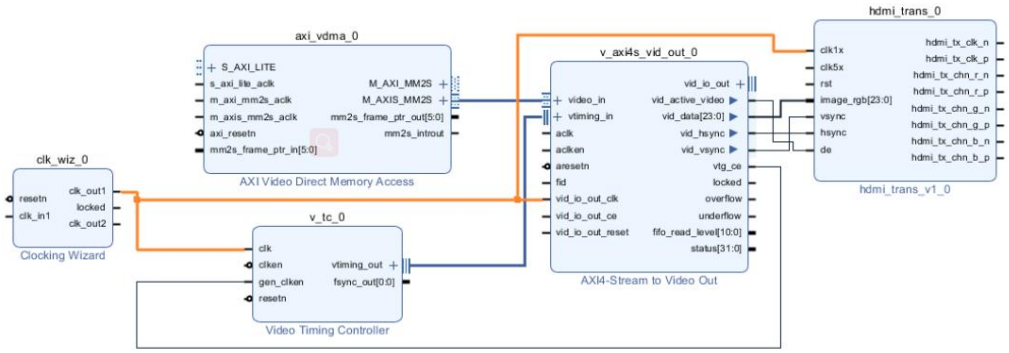


Figure 9-12 Pixel clock connection

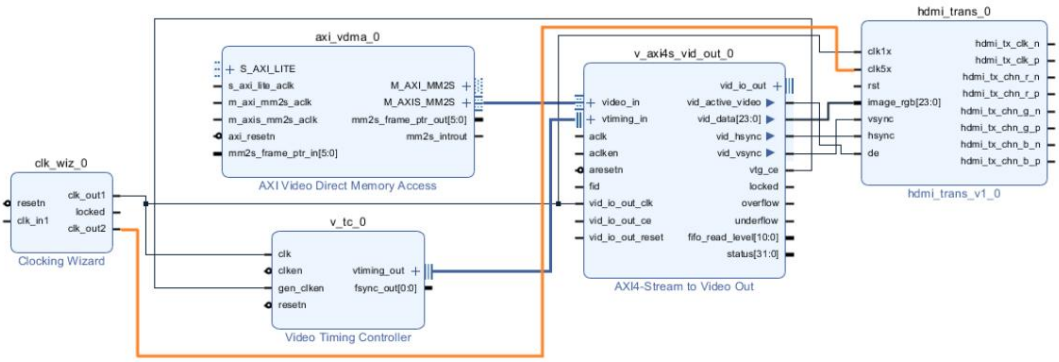


Figure 9-13 Serial clock connection

Various reset signals related to image output, some modules require high reset, some modules require Low reset, add a vector IP to convert simple level signals.

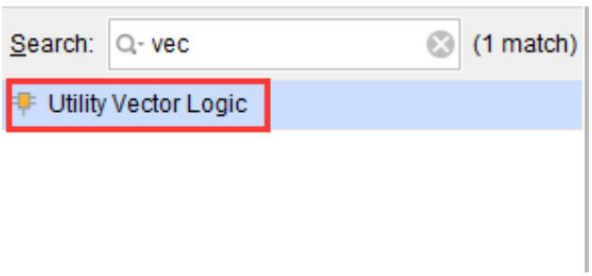


Figure 9- 14

Set the IP to a one-bit inversion operation.

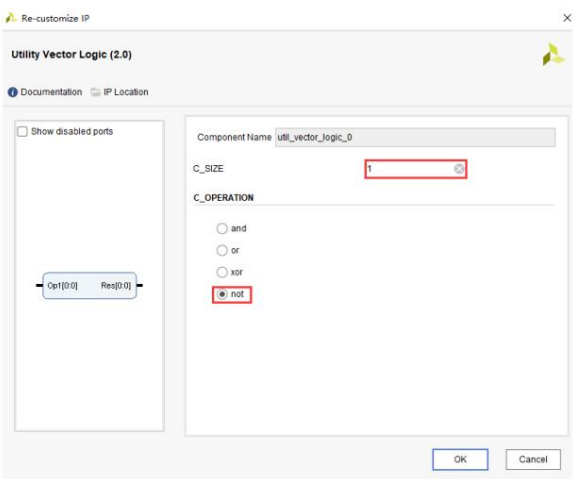


Figure 9-15

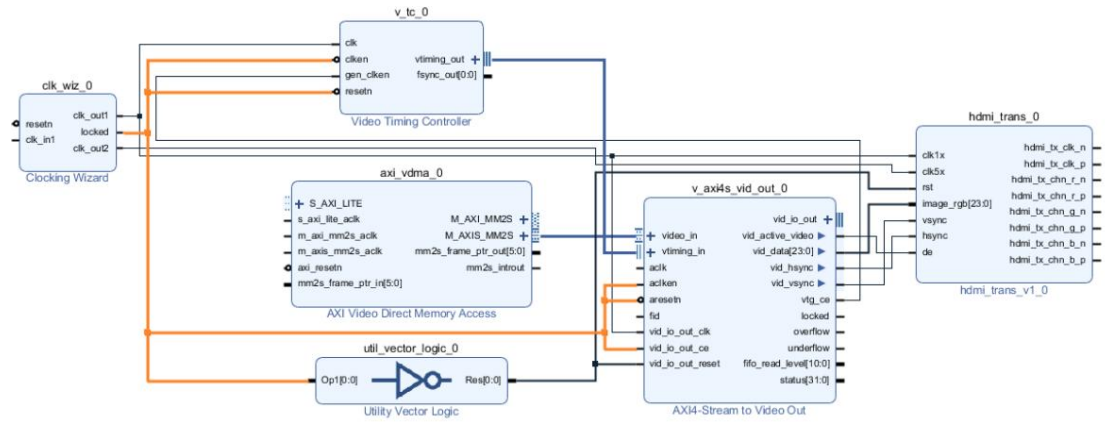


Figure 9-16 Low level reset connection

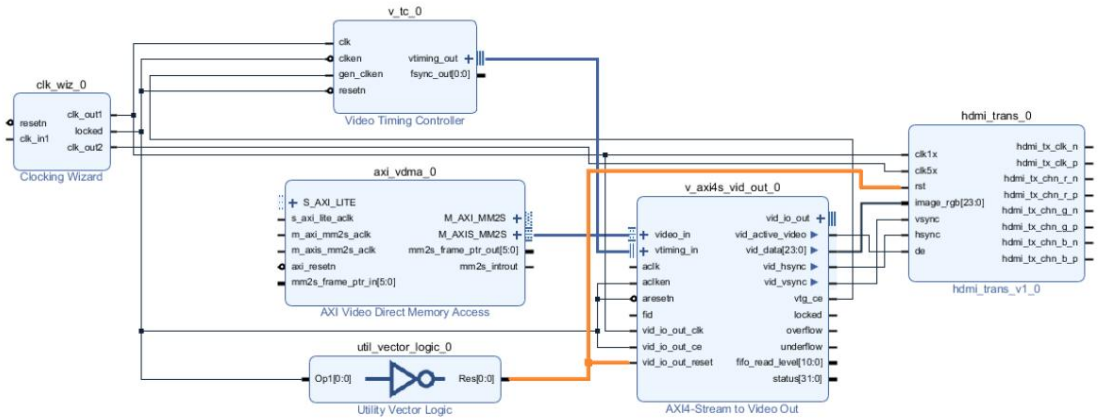


Figure 9-17 High level reset connection

9.2.2 **Zynq** Processor

Next, add the ZYNQ processor and configure it in a similar way to the previous experiment.

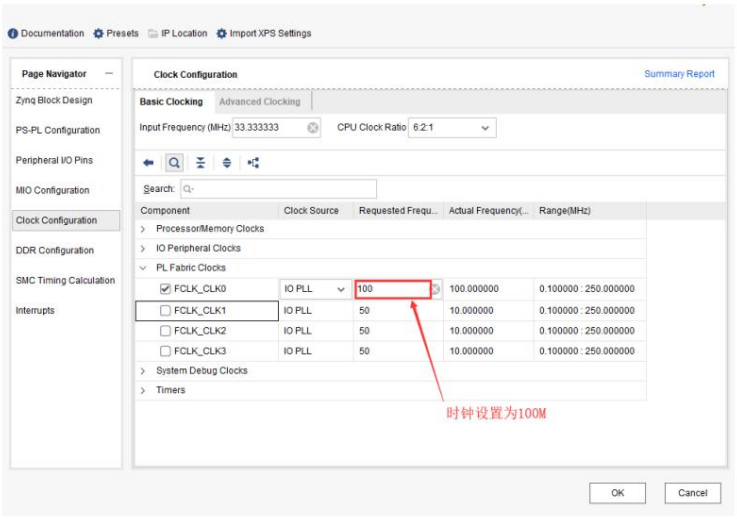


Figure 9- 18

The clock is then given to the PLL for generating the pixel clock.

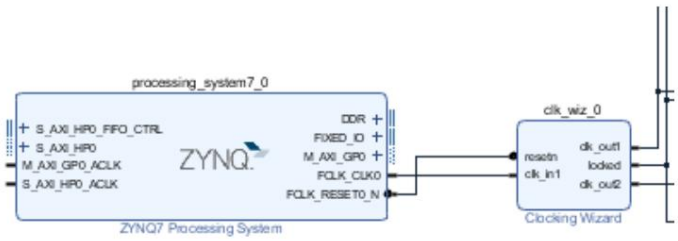


Figure 9- 19

9.2.3 **HLS IP** Part

Next, add the 2D convolution operation IP generated by HLS, which has also been added to the ip_repo path

Next, you can use the IP by searching directly.

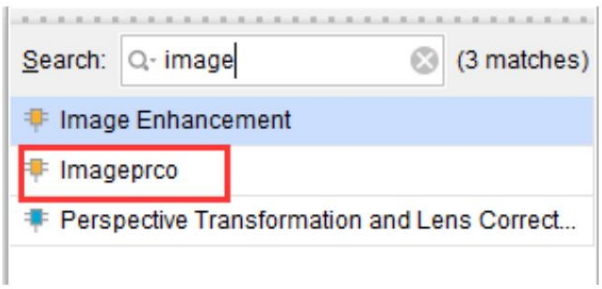


Figure 9- 20

The usage of this IP is similar to that in the previous chapter, except that DMA is added to connect to it.

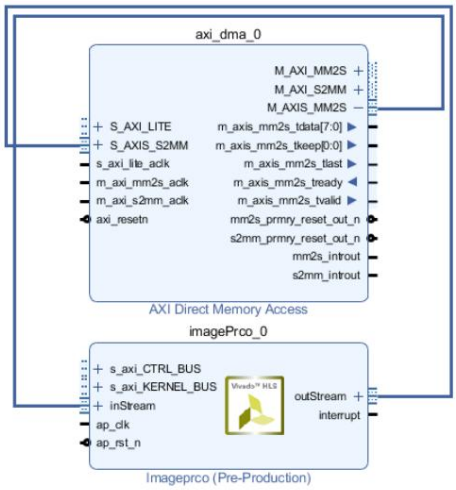


Figure 9- 21

9.2.4 Create **Block Design** by Wiring

After all the above IPs are added, click Automatic Connection to generate the final Block Design for this experiment.

The Block Design after the connection is completed is shown in the figure:

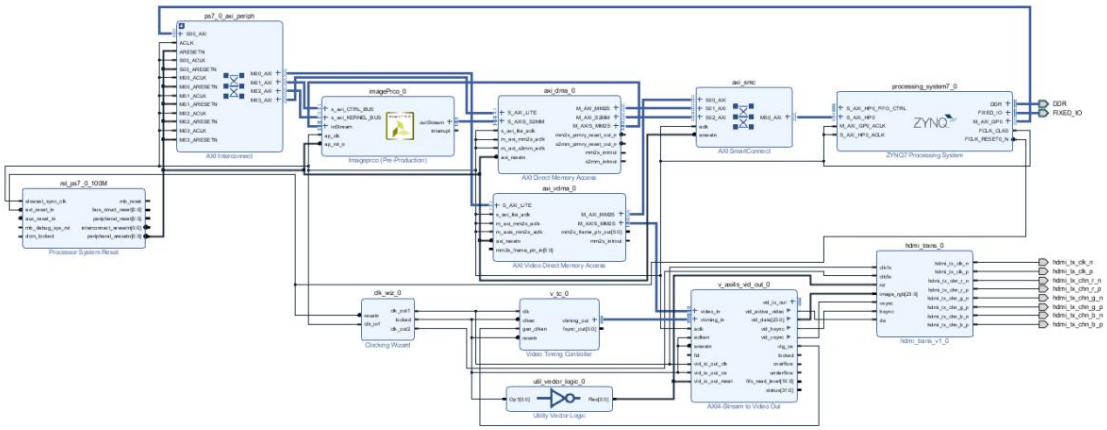


Figure 9- 22

Add a constraint file and add the pin constraints for this experiment in the constrian column:

```
set_property PACKAGE_PIN K17 [get_ports hdmi_tx_clk_p] set_property PACKAGE_PIN
G19 [get_ports {hdmi_tx_chn_r_p}] set_property PACKAGE_PIN F19 [get_ports {hdmi_tx_chn_g_p}]


set_property PACKAGE_PIN D19 [get_ports {hdmi_tx_chn_b_p}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {hdmi_oen[0]}]
set_property PACKAGE_PIN M20 [get_ports {hdmi_oen[0]}]
set_property IOSTANDARD TMDS_33 [get_ports hdmi_tx_chn_r_p]
set_property IOSTANDARD TMDS_33 [get_ports hdmi_tx_chn_g_p]
set_property IOSTANDARD TMDS_33 [get_ports hdmi_tx_chn_b_p]
set_property IOSTANDARD TMDS_33 [get_ports hdmi_tx_clk_p]
```

Then create a wrapper and generate a bit file.

9.3 Create **SDK** Project

After the bit file is generated, you can develop it in the SDK. This experiment is completed by inputting 0,

1, 2, 3 to select the operation to be performed on the image, and the operation can be viewed intuitively on the display.

Observe the changes after performing different convolution operations on the image.

The SDK codes are not explained in detail here, but only the meanings of several important codes are introduced.

First, the initialization and configuration of VDMA is completed by calling the Xilinx official API function.

The official API function is also provided in our project. In this experiment, we only need to call its

The configuration of VDMA can be completed by using an API function in .

```
run_vdma_frame_buffer(&Vdma,VDMA_DEV_ID,1280,720,VDMA_MM2S_ADDR,0,
0,ONLY_READ);
```

Use this function to set the size of the cached image in the VDMA image buffer to 1280*720.

Set to 720P format.

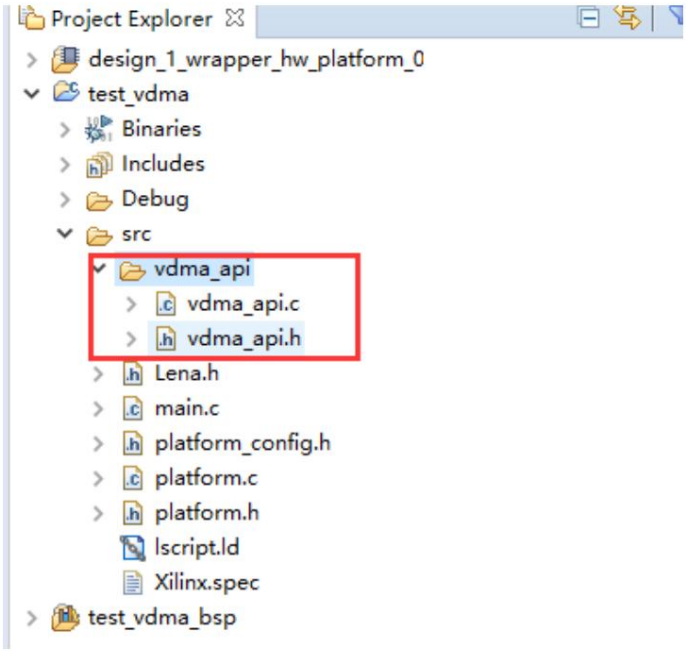


Figure 9- 23 VDMA API function

The following table shows different kernels. These kernels have different functions.

Erode_or_dilatekernel is used to erode and dilate the image, while impulse Kernel only performs

Output without any changes, in order to be able to compare with the image after the operation.

Sobel_kernel is used for sobel edge detection, and edge_kernel is also an edge detection operator.

These kernels cooperate with the keyboard input operations to complete the final image display.

```
char Erode_or_Dilate_kernel[9] = {
                1, 1, 1,
                1, 1, 1,
                1, 1, 1,
};

// impulse kernel (char  Just import the image without making any changes
impulse_kernel[9] = {
                0, 0, 0,
                0, 1, 0,
                0, 0, 0,
};

// kernel for sobel
char sobel_kernel[9] = {
                -1, -2, -1,
```

```
                0, 0, 0,
                1, 2, 1,
};

// kernel for edge
char edge_kernel[9] = {
                -1, -1, -1,
                -1, 8, -1,
                -1, -1, -1,
};
```

Click debug to enter the debugging interface. Connect the serial port first to observe the printed information.

Click Run.

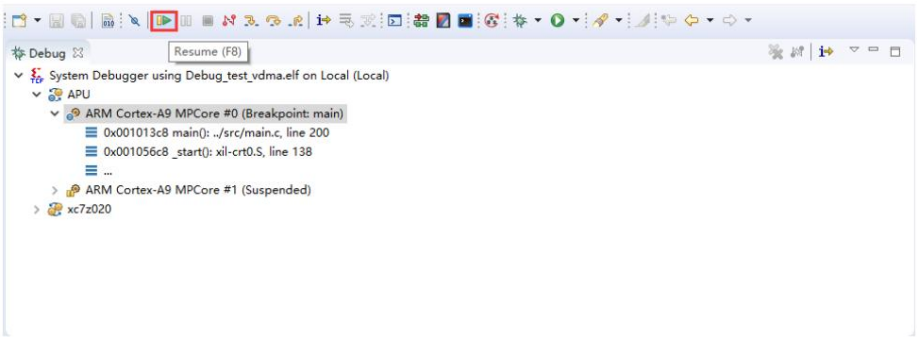


Figure 9- 24

After the serial port outputs the prompt information, connect the HDMI cable first.

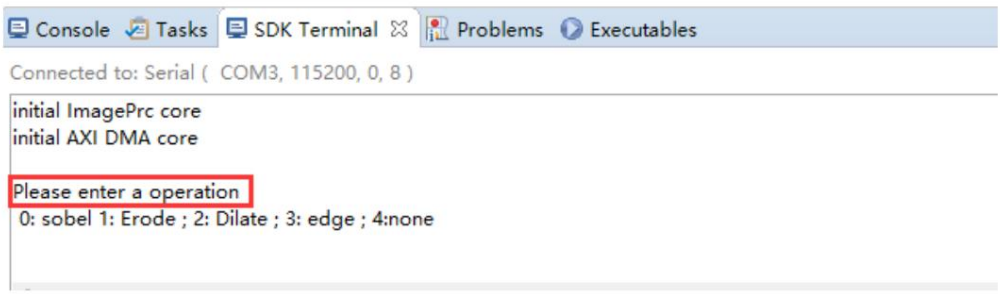


Figure 9- 25

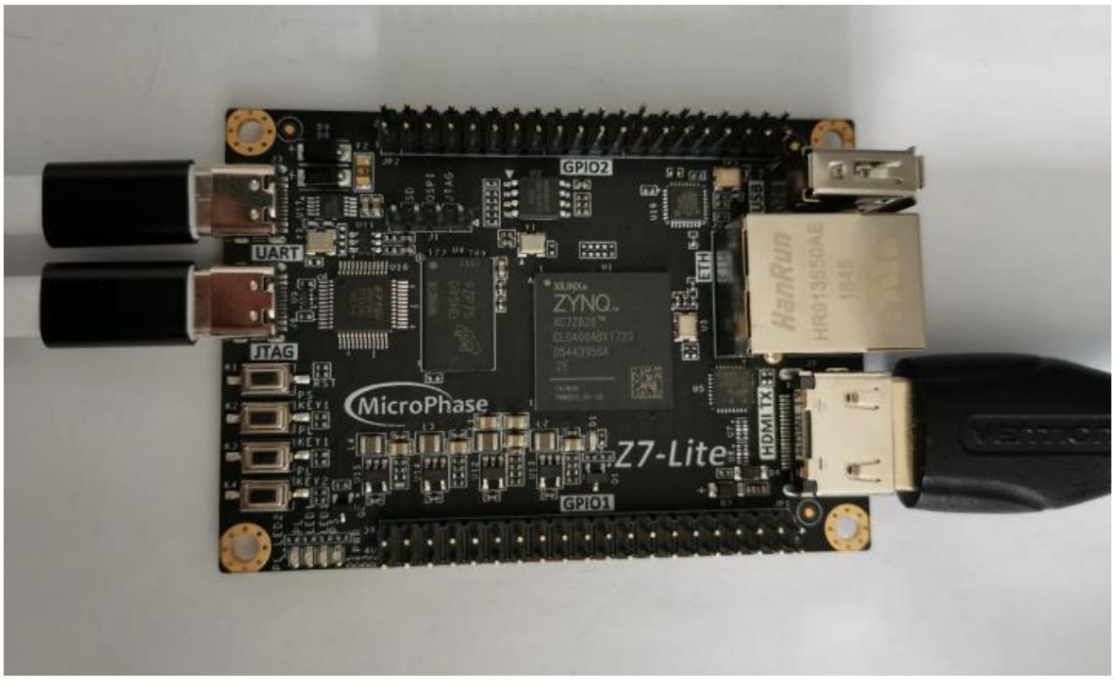


Figure 9- 26

At this time, you can observe different processing of the image by inputting 0~4 in the serial port. For example,

4. That is, no processing is done on the image, and the original image is displayed on the screen.

Figure 9-27 Displaying the original image on the screen

At this time, input 1 to observe the eroded image.

Figure 9-28 Image after corrosion

At this time, input 2 to observe the expanded image.

Figure 9-29 Image after expansion

In the document, you may not be able to see a clear difference. In practice, if you quickly switch between erosion and expansion,

If you look at the picture above, you can see a clear difference.

At this time, input 0 to see the image after Sobel edge detection.

Figure 9-30 Sobel edge detection image

At this time, input 3 to get another image after edge detection.

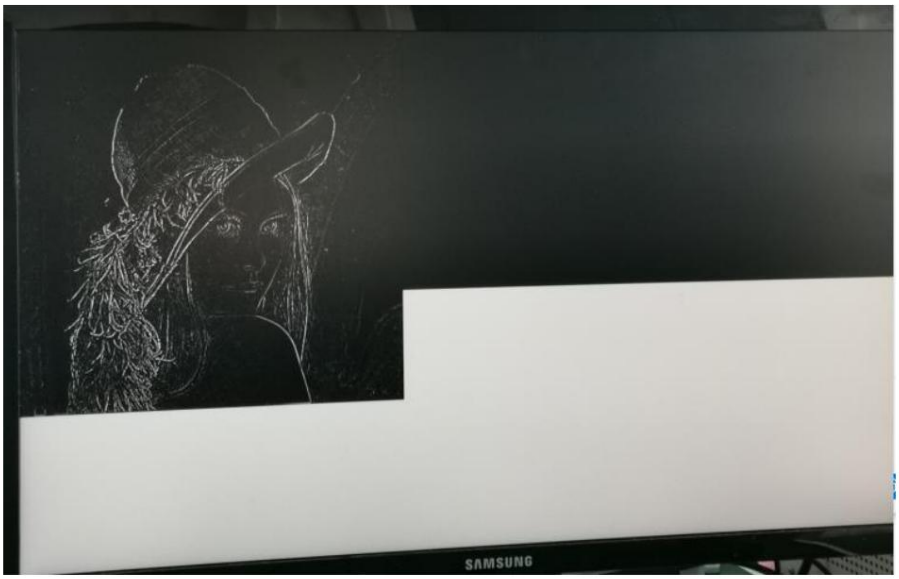


Figure 9-31 Image after edge detection

From now on, the entire demonstration of this experiment is completed. In the SDK, different convolution kernels (Kernel)

The convolution operation is completed by calling the IP core generated by HLS.

Compared with traditional methods, the efficiency is greatly improved.

9.4 Experimental Summary

In this experiment, by calling the IP generated by HLS in the previous experiment,

Block Design uses VDMA to manage output images, which can simplify DDR operations.

In the SDK, the image is processed differently by pressing the corresponding key.

Simple erosion, dilation, and edge detection functions.

In this experiment, it is important to note that when building block design in Vivado, you need to pay attention to each module.

There are connections between them, and some signals are easy to be missed due to negligence.

Chapter 10 Sobel Edge Detection of **OV5640**

10.1 Experimental Introduction

In the experiment of the previous chapter, we introduced how to use HLS for two-dimensional convolution, and finally completed

The image after two-dimensional convolution is displayed on the monitor. In this chapter, microphotography will be used.

One of the cameras on the OV5640 binocular camera is based on the experiment in the previous chapter.

The edge detection of the image data collected by the camera is completed.

10.2 OV5640 Hardware Introduction

The OV5640 binocular camera daughter card is shown in the figure below:

Figure 10- 1

OV5640 is a 1/4-inch single-chip image sensor with a photosensitive array of 2592*1944 (i.e.

500W pixels), can achieve the fastest 15fps QSXVGA (2592*1944) or 90fps VGA (640*480)

The sensor uses OmniBSI (Backside Illumination) technology introduced by OmniVision.

The sensor achieves higher performance, such as high sensitivity, low crosstalk and low noise.

Image processing functions, including automatic exposure control (AEC), automatic white balance (AWB), etc.

The device supports LED fill light, MIPI (Mobile Industry Processor Interface) output interface and DVP (Digital Video Parallel)

Output interface selection, ISP (image signal processing) and AFC (auto focus control) and other functions.

10.2.1 OV5640 Overall Architecture

By looking at the datasheet of OV5640, we can see the overall architecture of OV5640, as shown in the figure below:

figure 2-1 OV5640 block diagram

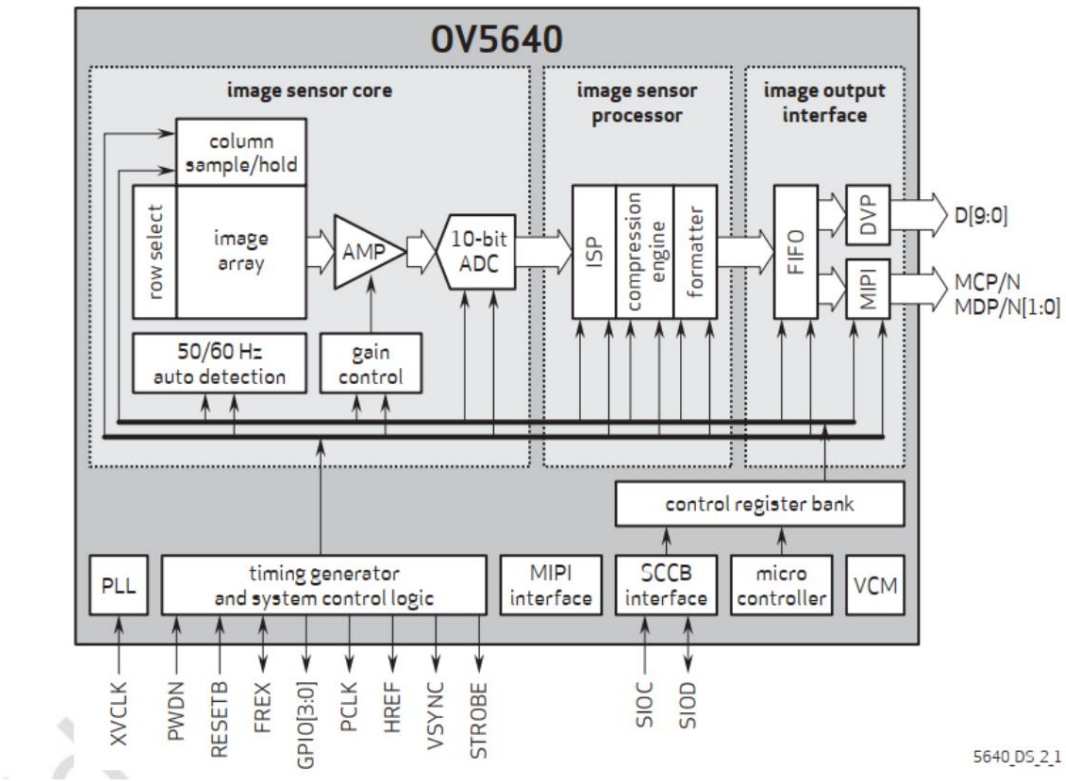


Figure 10- 2

Reading the datasheet, we can know that the general working process of OV5640 is:

In the image sensor core, raw analog image data is obtained in the image array through exposure and sampling.

The image is amplified and corrected by the AMP, and then the corrected image is passed through a 10-bit ADC chip.

The digital signal is converted into a digital signal, and the digital signal of the image is obtained through the image sensor processor.

The signal is cached in the image output interface and finally output as a 10-bit data stream through the DVP or MIPI interface.

The timing generator and system control logic controls the photosensitive array, AMP

and ADC, and generates the image timing signals VSYNC, HREF, PCLK through this module, and the external clock passes

XVCLK passes through PLL and enters the system as the system's working clock.

How to determine the working status of each module and the format and output timing of the output image are

This is achieved by configuring the registers of OV5640. The registers of OV5640 are configured through the SCCB interface.

achieved.

10.2.2 Introduction to SCCB

SCCB (serial camera control bus) is a standard developed by OmniVision.

A three-wire serial bus defined and developed, through which the camera's internal registers can be configured.

Design the camera's functions, including image data format, resolution, and image processing parameters.

In order to reduce the packaging of sensor pins, most SCCB buses now use a two-wire interface bus.

SCCB is very similar to the I2C protocol in form, so in this chapter we will only briefly introduce it.

SCCB protocol. SCCB mainly consists of two lines SCL and SDA. By adjusting the timing of these two signal lines,

Transmission of data.

1) SCCB write transfer timing:

Figure 10- 3

Similar to I2C protocol transmission, SCCB also generates a start bit when transmitting data, and then

Given a 7-bit device address and a write operation, wait for the first eight bits of data to be transmitted before an unused

Pay attention to the bit, which is different from the I2C protocol. In the I2C transmission protocol, after a byte of data is sent,

There is a one-byte response bit, which is a don't care in the SCCB protocol.

Then a byte of register address is transmitted. Similarly, there is a don't care bit after the transmission is completed.

This is followed by a byte of data written to the register, followed by a don't care bit, and finally a stop bit.

The transmission ends.

It can be seen that SCCB data transmission is very similar to I2C. With the previous I2C protocol foundation,

The SCCB protocol here is not difficult to understand.

2) SCCB read transfer timing

The SCCB read transfer timing goes through two stages, which is similar to the I2C protocol, but it should be noted that

The first part of the SCCB read sequence is the write device address and the write register address, and then

A stop bit. In the second stage, a start bit is generated, followed by reading the register and the read data.

The last bit is the stop bit. The read timing of SCCB is quite different from that of I2C.

The START signal and STOP signal always appear in pairs, but they are not used in the I2C protocol.

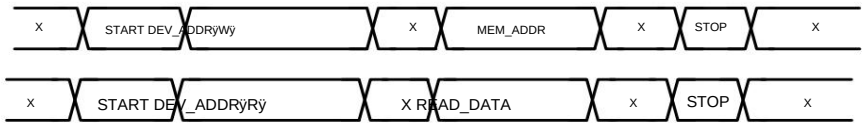


Figure 10- 4

In addition, it is also important to note that in the SCCB transmission protocol, continuous reading and writing are not possible.

That is to say, the SCCB can only read and write one register at a time, and cannot read and write continuously.

It is also important to note that the device OV5640 used in this experiment has a 16-bit address.

When transmitting the register address, it must be divided into two bytes for transmission. The first byte transmits the register address.

The high eight bits, the second byte transfers the register address low eight bits.

10.2.3 Configuration Registers

To determine the working mode of OV5640, we need to configure the registers of OV5640.

The configuration can refer to the OV5640 user manual and software manual. We can find the configuration in the src\doc directory under the project directory.

The relevant manuals about OV5640 are provided in the catalog.

Here is a brief introduction to the more important register configurations.

Image window configuration:

The image window is used to configure the size and scaling of the output image.

Change the window's starting position, window size, window offset relative to the starting position, and zoom ratio.

Setting these parameters can achieve the size of the output image. For example, in this experiment, we will set the window

The effective size is 1024*768, which can be configured through the following registers.

'igure 4-3 image windowing configuration

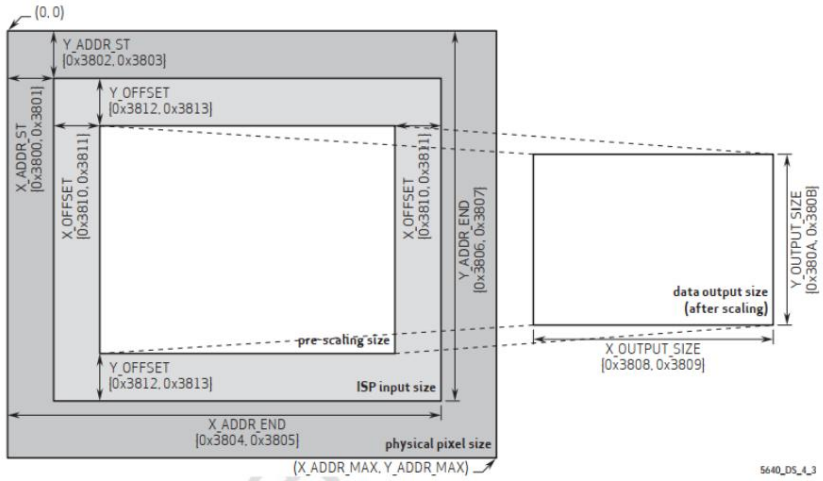


Figure 10- 5

table 4-2 image windowing registers (sheet 1 of 2)

address	register name	default value	R/W	description
0x3800	TIMING HS	0x00	RW	Bit[3:0]: X address start high byte[11:8] high byte
0x3801	TIMING HS	0x00	RW	Bit[7:0]: X address start low byte[7:0] low byte
0x3802	TIMING VS	0x00	RW	Bit[2:0]: Y address start high byte[10:8] high byte
0x3803	TIMING VS	0x00	RW	Bit[7:0]: Y address start low byte[7:0] low byte
0x3804	TIMING HW	0x0A	RW	Bit[3:0]: X address end high byte[11:8] high byte
0x3805	TIMING HW	0x3F	RW	Bit[7:0]: X address end low byte[7:0] low byte
0x3806	TIMING VH	0x07	RW	Bit[2:0]: Y address end high byte[10:8] high byte
0x3807	TIMING VH	0x9F	RW	Bit[7:0]: Y address end low byte[7:0] low byte
0x3808	TIMING DVPHO	0x0A	RW	Bit[3:0]: DVP output horizontal width[11:8] high byte

Figure 10- 6

table 4-2 image windowing registers (sheet 2 of 2)

address	register name	default value	R/W	description
0x3809	TIMING DVPHO	0x20	RW	Bit[7:0]: DVP output horizontal width[7:0] low byte
0x380A	TIMING DVPVO	0x07	RW	Bit[2:0]: DVP output vertical height[10:8] high byte
0x380B	TIMING DVPVO	0x98	RW	Bit[7:0]: DVP output vertical height[7:0] low byte
0x380C	TIMING HTS	0x0B	RW	Bit[3:0]: Total horizontal size[11:8] high byte
0x380D	TIMING HTS	0x1C	RW	Bit[7:0]: Total horizontal size[7:0] low byte
0x380E	TIMING VTS	0x07	RW	Bit[7:0]: Total vertical size[15:8] high byte
0x380F	TIMING VTS	0xB0	RW	Bit[7:0]: Total vertical size[7:0] low byte
0x3810	TIMING HOFFSET	0x00	RW	Bit[3:0]: ISP horizontal offset[11:8] high byte
0x3811	TIMING_HOFFSET	0x10	RW	Bit[7:0]: ISP horizontal offset[7:0] low byte
0x3812	TIMING VOFFSET	0x00	RW	Bit[2:0]: ISP vertical offset[10:8] high byte
0x3813	TIMING VOFFSET	0x04	RW	Bit[7:0]: ISP vertical offset[7:0] low byte

Figure 10- 7

1) Output image format configuration:

The output format of the image needs to be configured in register 0x4300. In this experiment, we use

Configure this register to finally let the camera output data in RGB565 format.

Set the bit[7:4] of the register to 0x6. After setting the output format of 0x4300 to rgb565,

Like the data output order, here we set the fourth bit [3:0] of 0x4300 to 0x1, so that

When the output order is specified, the output is in the order of red, green and blue.

Format control converts the internal data format into the desired output format including YUV, RGB, or RAW.

table 6-5 FORMAT control registers (sheet 1 of 5)

address	register name	default value	R/W	description
0x4300	FORMAT CONTROL 00	0xF8	RW	Format Control 00 Bit[7:4]: Output format of formatter module 0x0: RAW Bit[3:0]: Output sequence 0x0: BGBG... / GRGR... 0x1: GBGB... / RGRG... 0x2: GRGR... / BGBG... 0x3: RGRG... / GBGB... 0x4~0xF: Not allowed 0x1: Y8 Bit[3:0]: Does not matter 0x2: YUV444/RGB888 (not available for full resolution) Bit[3:0]: Output sequence 0x0: YUVYUV..., or GBRGBR... 0x1: YVUYVU..., or GRBGRB... 0x2: UYVUYV..., or BGRBGR... 0x3: VYUVYU..., or RGBRGB... 0x4: UVYUVY..., or BRGBRG... 0x5: VUYVUY..., or RBGRBG... 0x6~0xE: Not allowed 0xF: UYVUYV..., or BGRBGR... 0x3: YUV422 Bit[3:0]: Output sequence 0x0: YUYV... 0x1: YVYU... 0x2: UYVY... 0x3: VYUY... 0x4~0xE: Not allowed 0xF: UYVY... 0x4: YUV420 Bit[3:0]: Output sequence 0x0: YYYY... / YUYV... 0x1: YYYY... / YVYU... 0x2: YYYY... / UYVY... 0x3: YYYY... / VYUY... 0x4: YUYV... / YYYY...

table 6-5 FORMAT control registers (sheet 2 of 5)

address	register name	default value	R/W	description
				0x5: YVYU... / YYYY... 0x6: UYVY... / YYYY... 0x7: VYUY... / YYYY... 0x8~0xE: Not allowed 0xF: YYYY... / UYVY... 0x5: YUV420 (for MIPI only) Bit[3:0]: Output sequence 0x0~0xD: Not allowed 0xE: VYYVYY... / UYYUYY... 0xF: UYYUYY... / VYYVYY... 0x6: RGB565 Bit[3:0]: Output sequence 0x0: {b[4:0],g[5:3]}, {g[2:0],r[4:0]} 0x1: {r[4:0],g[5:3]}, {g[2:0],b[4:0]} 0x2: {g[4:0],r[5:3]}, {r[2:0],b[4:0]} 0x3: {b[4:0],r[5:3]}, {r[2:0],g[4:0]} 0x4: {g[4:0],b[5:3]}, {b[2:0],r[4:0]} 0x5: {r[4:0],b[5:3]}, {b[2:0],g[4:0]} 0x6~0xE: Not allowed 0xF: {g[2:0],b[4:0]}, {r[4:0],g[5:3]} 0x7: RGB555 format 1 Bit[3:0]: Output sequence 0x0: {b[4:0],g[4:2]}, {g[1:0],1'b0,r[4:0]} 0x1: {r[4:0],g[4:2]}, {g[1:0],1'b0,b[4:0]} 0x2: {g[4:0],r[4:2]}, {r[1:0],1'b0,b[4:0]} 0x3: {b[4:0],r[4:2]}, {r[1:0],1'b0,g[4:0]} 0x4: {r[4:0],b[4:2]}, {b[1:0],1'b0,g[4:0]} 0x5: {g[4:0],b[4:2]}, {b[1:0],1'b0,r[4:0]} 0x6~0xE: Not allowed 0xF: {g[1:0],1'b0,b[4:0]}, {r[4:0],g[4:2]} 0x8: RGB555 format 2 Bit[3:0]: Output sequence 0x0: {1'b0,b[4:0],g[4:3]}, {g[2:0],r[4:0]}

Here is a brief description of the output format. The output image is in RGB565 format.

The data is two bytes, but when outputting data, the output data bit width of our DVP interface is

10 bits, namely D[9:0], in order to transmit data correctly, the data of one pixel will be divided into two clock cycles.

Send, only D[9:2] is used as the valid bit of data. According to the configuration given in the previous 0x4300, we can see that

The first clock cycle outputs D[9:2]={r[4:0],g[5:3]}; the output r component and g component are high 3

Bit.

The second clock cycle outputs D[9:2]={g[2:0],b[4:0]}; the lower three bits of the g component and the b component are output.

This way the image can be transmitted correctly.

For other register configurations, users can refer to the document OV5640_CSP3_DS_2.01.

The register descriptions in the documentation are adjusted.

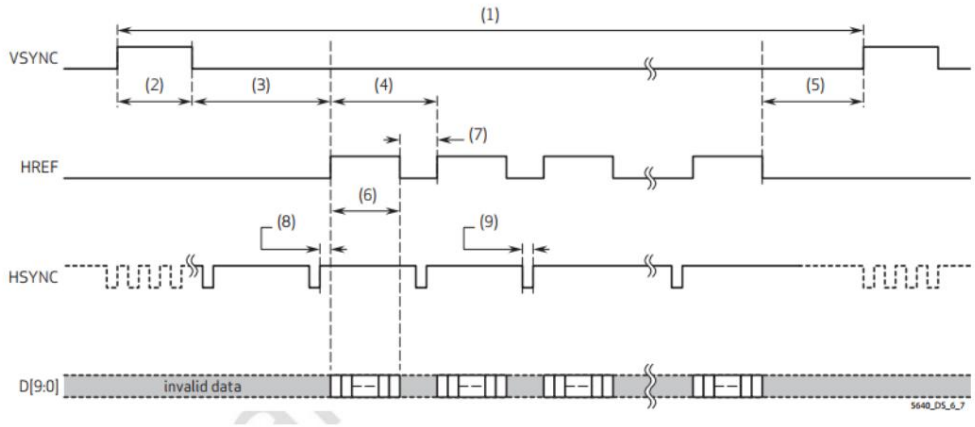
10.2.4 Image output timing:

By referring to the user manual OV5640_CSP3_DS_2.01, you can see the timing of the output image, as follows

As shown in the figure:

6.6.2 DVP timing

figure 6-7 DVP timing diagram



Next, let's briefly introduce this timing diagram. Taking a 1024*768 resolution image as an example, the following figure is

Timing parameters at this resolution:

XGA 1024x768	(1) 5596992 tp (2) 5688tp (3) 2825588 tp (4) 2844 tp (5) 583344tp (6) 1024 tp (7) 1820 tp (8) 0 tp (9) 1820 tp

For the output timing mode of the DVP interface, there are generally two modes to choose from, one is VSYNC

and HSYNC mode, which is similar to the VGA mode we are familiar with; the other mode is

VSYNC and HREF mode. In this mode, when the HREF signal is high, it means that the data is valid.

When OV5640 collects images, we generally use this method to output images, so that users can receive

It can provide convenience when receiving image data. When the HREF signal is valid, it means that the current data

efficient.

It is worth noting that the timing parameter ÿp in the timing diagram is not necessarily the same as that of OV5640.

The system clock of the output pixel format. This depends on the pixel format we output. For example, the output format of rgb565

The data is tp refers to the pixel clock, which is the clock that outputs a pixel. As mentioned earlier,

It takes two system clocks to output one pixel in the RGB565 format, so the Tp of this path is 2*Tsys.

In this experiment, the brief introduction about OV5640 ends here. If users want to learn more

To learn about the working principle of OV5640, you can refer to the user manual and its corresponding software and hardware manuals.

In this experiment, we will provide a packaged camera acquisition IP.

The output format of the header video is 1024x768.

10.3 Design of 2D Convolution in HLS

Compared with the previous experiment, this experiment requires very few changes in the HLS part. The source code is as follows:

```
1 #include "kernel.h"
2
3           void doKernel(AXI_STREAM & inStream,AXI_STREAM & outStream, char
kernel[KERNEL_DIM*KERNEL_DIM], int operation) 4 { 5 #pragma
HLS
INTERFACE s_axilite port=return bundle=CTRL_BUS 6 #pragma HLS INTERFACE
s_axilite port=operation bundle=CTRL_BUS
7 #pragma HLS INTERFACE s_axilite port=kernel bundle=KERNEL_BUS 8 #pragma HLS
INTERFACE axis register both port=outStream 9 #pragma HLS INTERFACE axis
register both port=inStream
10
11 BUFFER lineBuf;
12 SHORT_WINDOW window;
13
14 //15   Index values used to label rows and columns
int pixConvolved = 0;
16
17 //line_buffer.val[2][j] = 18 int waitPixel = IMG_WIDTH
+ 1;
19 int cntWait = 0;
20 int sentPixel = 0;
21 unsigned char pix;
22
23 uint_side_channel dataOutChannel;
```

```
24 uint_side_channel currentChannel;
25
26
27 for(int i = 0; i < (IMG_HEIGHT); i++ )
28 {
29          for(int j = 0; j < (IMG_WIDTH); j++)
30          {
31 #pragma HLS PIPELINE
32                  currentChannel = inStream.read();
33                  unsigned char R = currentChannel.data & 0xFF;
34                  unsigned char G = (currentChannel.data >> 8) & 0xFF;
35                  unsigned char B = (currentChannel.data >> 16) & 0xFF;
36                  pix = (R*76 + G*150 + B*30) >> 8;
37                  lineBuf.shift_pixels_up(j);
38                  lineBuf.insert_bottom_row(pix,j);
39                 //put data on the window, and multiply by kernel
40                  for(int idxWinRow = 0; idxWinRow < KERNEL_DIM; idxWinRow++)
41                  {
42                          for(int idxWinCol=0; idxWinCol < KERNEL_DIM; idxWinCol++)
43                          {
44                                  //idxWinCol and pixConvolved will slide the window
45                                  short val = (short)lineBuf.getval(idxWinRow, idxWinCol +
pixConvolved);
46                                  //  The data in the sampling window is Kernel( Convolution operator )multiplication
47                                  val = (short)kernel[(idxWinRow * KERNEL_DIM) + idxWinCol] * val;
48                                  window.insert(val,idxWinRow,idxWinCol);
49                          }
50                  }
51                  short valOut;
52                  if((i >= KERNEL_DIM-1) && (j >= KERNEL_DIM -1))
53                  {
54                          valOut = sumWindow(&window);
55                          if(valOut < 0 )
56                          {
57                                  valOut = 0;
58                          }
59                          pixConvolved++;
60                          dataOutChannel.data = (valOut << 16) | (valOut << 8) | valOut;
61                          dataOutChannel.dest = currentChannel.dest;
62                          dataOutChannel.id = currentChannel.id;
63                          dataOutChannel.keep = currentChannel.keep;
```

```
64                      dataOutChannel.strb = currentChannel.strb;
65                      dataOutChannel.user = currentChannel.user;
66                      dataOutChannel.last = currentChannel.last;
67                      outStream.write(dataOutChannel);
68                      sentPixel++;
69
70                  }
71                  else
72                  {
73                      dataOutChannel.data = 0;
74                      dataOutChannel.dest = currentChannel.dest;
75                      dataOutChannel.id = currentChannel.id;
76                      dataOutChannel.keep = currentChannel.keep;
77                      dataOutChannel.strb = currentChannel.strb;
78                      dataOutChannel.user = currentChannel.user;
79                      dataOutChannel.last = currentChannel.last;
80                      outStream.write(dataOutChannel);
81                      sentPixel++;
82                  }
83              }
84              pixConvolved = 0;
85 }
86 }
87
88 //  Find the convolution sum
89 short sumWindow(SHORT_WINDOW *window){
90 short sum=0;
91 for(int idxRow = 0; idxRow < KERNEL_DIM; idxRow++){
92              for(int idxCol = 0;idxCol < KERNEL_DIM;idxCol++){
93                  sum += (short)window->getval(idxRow, idxCol);
94              }
95 }
96 return sum;
97
98 }
```

The work done is to convert the input RGB image into a grayscale image, and then perform a two-dimensional convolution on the grayscale image.

The result of the multiplication is output to obtain the image after edge detection.

The main function of the C simulation file is as follows.

```
1 int main(){
2 IplImage * SrcImage;// Input image
3 IplImage * DstImage;// Output image
4 IplImage * GoldImage;// Reference Images
5 IplImage * GoldImageTmp;// Reference Images
6 SrcImage = cvLoadImage(INPUT_IMAGE,-1);// Importing images
7
8 // create opencv Supported output images
9 DstImage = cvCreateImage(cvGetSize(SrcImage),SrcImage->depth, SrcImage->nChannels);
10 GoldImage cvCreateImage(cvGetSize(SrcImage),SrcImage->depth,
SrcImage->nChannels);
11 GoldImageTmp = cvCreateImage(cvGetSize(SrcImage),SrcImage->depth, 1);
12 //AXI-Stream Data Flow
13 AXI_STREAM inStream;
14 AXI_STREAM outStream;
15 IplImage2AXIvideo(SrcImage,inStream);
16
17 // Synthesizable functions
18 doKernel(inStream,outStream,kernel,0);
19 // Save the output
20 AXIvideo2IplImage(outStream,DstImage);
21 cvSaveImage(OUTPUT_IMAGE,DstImage);
22
23 // Reference image output
24 cvCvtColor(SrcImage, GoldImageTmp, CV_BGR2GRAY);
25 cvCvtColor(GoldImageTmp, GoldImage, CV_GRAY2BGR);
26 cvReleaseImage(&GoldImageTmp);
27 cvSaveImage(GOLD_IMAGE, GoldImage);
28 cvShowImage(OUTPUT_IMAGE, DstImage);
29 cvWaitKey(0);
30
31 //32 Compare the golden reference image and HLS Comprehensive execution results
image_compare(OUTPUT_IMAGE,GOLD_IMAGE);
33 // Release pictures
34 cvReleaseImage(&SrcImage);
35 cvReleaseImage(&DstImage);
36 cvReleaseImage(&GoldImage);
37 return 0;
38
39
40 }
```

When simulating in C, the impulse kernel is used for testing, and the output image is as shown below:

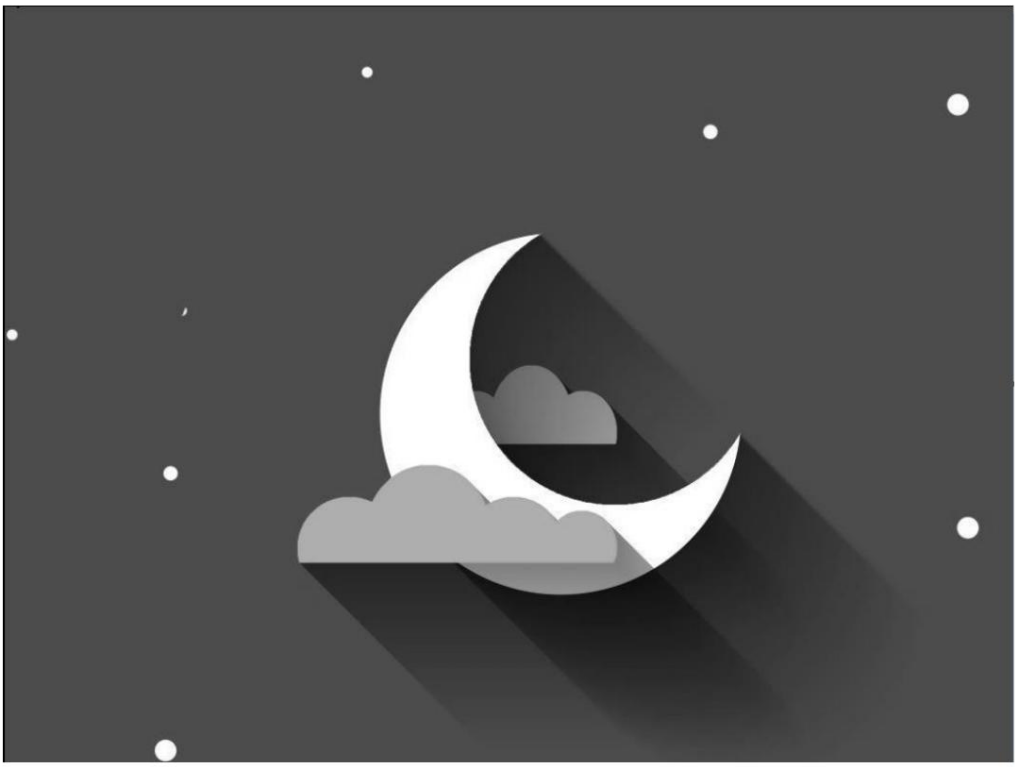


Figure 10- 8

As you can see, because Linebuffer is used in the source file, the entire image is offset.

After that, you can perform C synthesis and C simulation. After the simulation is correct, you can export the IP and call it in Vivado.

10.4 Building **Block Design**

After generating the HLS IP, first add the IP to the Vivado IP repository.

The IP collected by OV5640 is an IP used by Microphase in other developments.

The IP setting of ov5640 outputs images in the format of 1024x768@60Hz.

In HLS, the generated IP is also used to process the size of 1024x768.

For reference.

In the user's IP repository, you can see the IP generated by HLS in this experiment and the user-defined IP.

Figure 10- 9

Next, we will build the Block Design for this experiment.

10.4.1 Camera acquisition part

1) Add user-defined camera acquisition **IP**

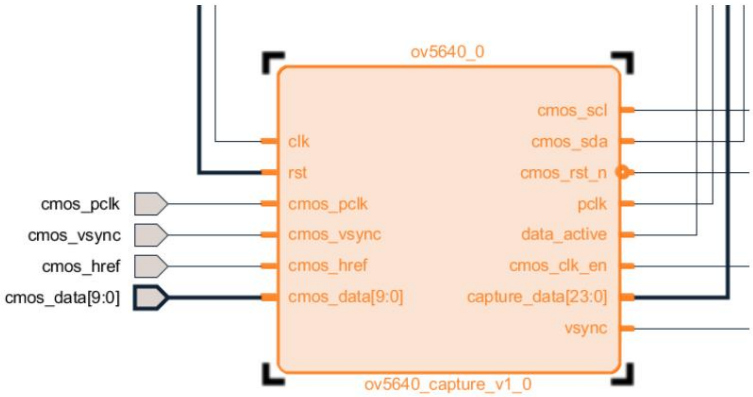


Figure 10- 10

2) Add **VDMA** for video input

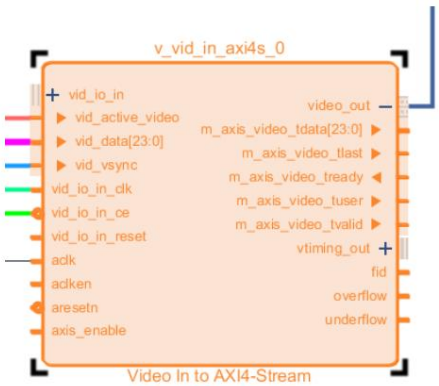


Figure 10-11

Configure the video input VDMA as follows: enable the read and write channels, and set the data flow of the read channel

The bit width is 24 bits.

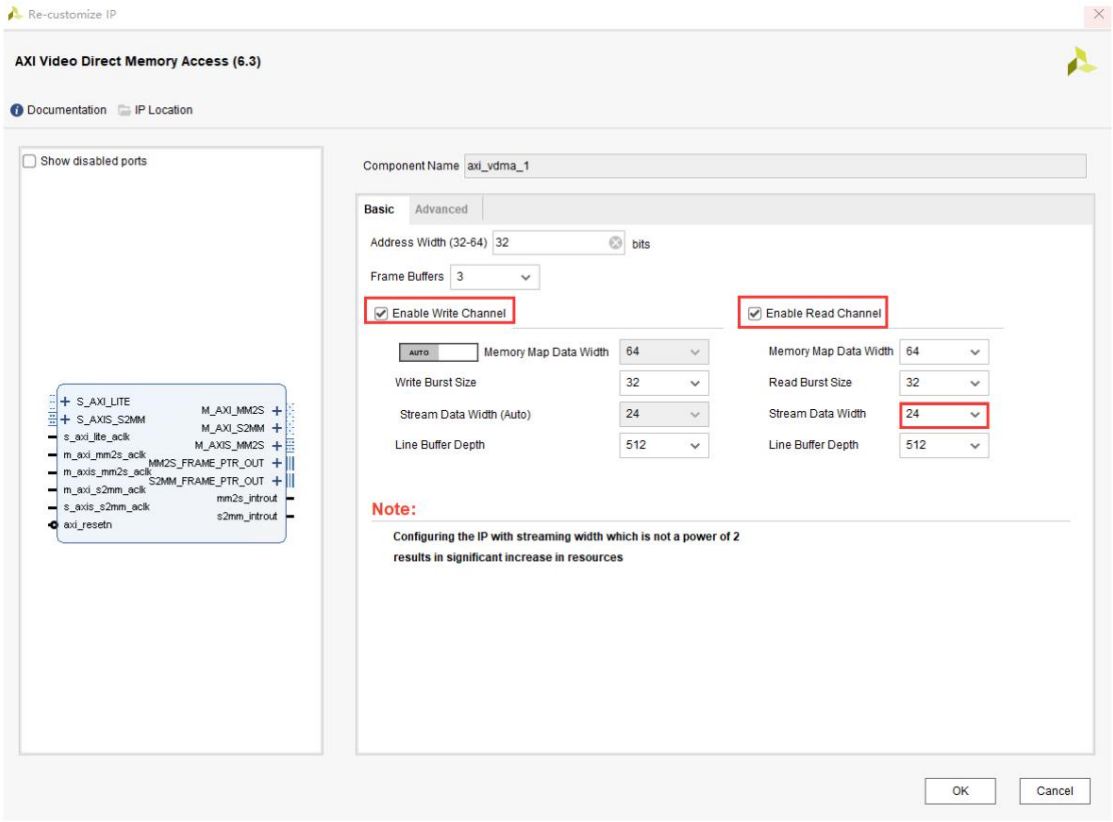


Figure 10-12

Connect the video input VDMA and the camera acquisition module as follows:

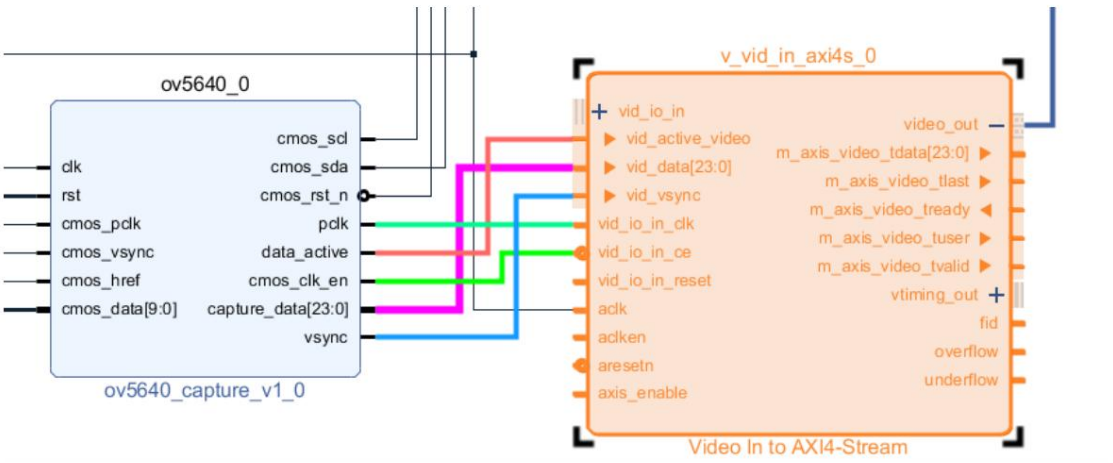


Figure 10- 13

In the camera acquisition module, pclk is the pixel clock, data_active is the data valid signal,

cmos_clk_en is the clock enable signal, capture_data is the captured image data. Vsynac is the field

After the connection is completed, the camera acquisition IP needs to lead out the remaining pins for data collection and configuration.

Set OV5640.

10.4.2 Video Output

For the video output path, the construction method is similar to the previous experiment. Users only need to add the video output

The VDMA of the output can be configured in the same way as the input VDMA.

10.4.3 Adding Image Processing **IP**

Next, add the IP for image processing generated by HLS. Connect the IP to the two VDMAs according to

Connect as follows.

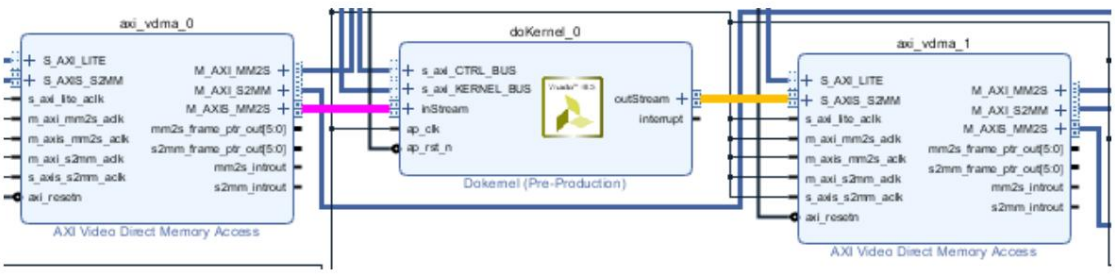


Figure 10- 14

Next, click Auto Connect to complete the construction of the entire block design.

10.5 Create **SDK** Project

After generating the bit file, you can export the bit file and develop it in the SDK.

```
1 #include <stdio.h>
2 #include "platform.h"
3 #include "xil_printf.h"
4 #include "vdma_api/vdma_api.h"
5 #include "xdokernel.h"
6 #include "xparameters.h"
7 #include "xil_cache.h"
8
```

```
9
10 #define DISPLAY_VDMA_DEV_ID XPAR_AXI_VDMA_1_DEVICE_ID
11 #define HLS_VDMA_DEV_ID                 XPAR_AXI_VDMA_0_DEVICE_ID
12 #define KERNEL_DEV_ID                       XPAR_DOKERNEL_0_DEVICE_ID
13
14 #define DISP_BASE_ADDR                  0x08000000
15 #define HLS_BASE_ADDR                   0x03000000
16 #define SCREEN_X                        1024
17 #define SCREEN_Y                        768
18
19
20 static XAxiVdma                         VDMA;
21 static XDokernel                    doKernel ;
22 static XDokernel_Config             *doKernel_Cfg;
23
24
25 //impulse kernel (            Just import the image without making any changes
26 char impulse_kernel[9] = {
27             0, 0, 0,
28             0, 1, 0,
29             0, 0, 0,
30 };
31
32 //kernel for sobel
33 char sobel_kernel[9] = {
34             -1, -2, -1,
35               0, 0, 0,
36               1, 2, 1,
37 };
38
39 //kernel for edge
40 char edge_kernel[9] = {
41             -1, -1, -1,
42             -1, 8, -1,
43             -1, -1, -1,
44 };
45
46
47 int initdockernel(){
48 int status;
49 doKernel_Cfg = XDokernel_LookupConfig(KERNEL_DEV_ID);
```

```
50 status = XDokernel_CfgInitialize(&doKernel,doKernel_Cfg);
51 if(status != XST_SUCCESS){
52                 printf("initialize failed! \n");
53                 return status;
54 }
55
56
57 return status;
58 }
59
60
61
62
63 //      Set the display background
64 void setBackground(){
65 u32 *memAddr;
66 int idxRow;
67 you are a fool;
68 memAddr = (u32 *) DISP_BASE_ADDR;
69 for(idxRow = 0;idxRow < SCREEN_Y; idxRow++){
70                 for(idxCol = 0;idxCol < SCREEN_X; idxCol++){
71                         if(idxRow < SCREEN_Y/2){
72                                 memAddr[idxCol + idxRow*SCREEN_X] = 0;
73                         }
74                         else{
75                                 memAddr[idxCol + idxRow*SCREEN_X] = 0xFFFFFF;
76                         }
77                 }
78 }
79 Xil_DCacheFlush();
80 }
81
82
83
84
85
86 int main()
87 {
88 int status;
89             //  initialization VDMA   And configure
90 run_vdma_frame_buffer(&Vdma, HLS_VDMA_DEV_ID, SCREEN_X, SCREEN_Y,HLS_BASE_ADDR,0,
```

```
0,BOTH);
91 run_vdma_frame_buffer(&Vdma, DISPLAY_VDMA_DEV_ID, SCREEN_X, SCREEN_Y,DISP_BASE_ADDR,0,
0,BOTH);
92 status = initdockernel();
93 if(status != XST_SUCCESS){
94              printf("initialize failed! \n");
95              return status;
96 }
97 setBackground();
98 //      set up Kernel_bus,     Used to set the convolution operator used for the image
99 XDokernel_Write_kernel_Bytes(&doKernel,0,edge_kernel,9);
100
101 //     Set the operation type operation 0:     Find the convolution sum
102 XDokernel_Set_operation(&doKernel,0);
103 //
104 while(1){
105             XDokernel_Start(&doKernel);
106             while(!XDokernel_IsDone(&doKernel)){
107
108             }
109 }
110          print("Hello World\n\r");
111
112          return 0;
113 }
```

The main program of SDK is as follows, which completes the initialization of VDMA and the image processing IP.

initialization.

Line 99 is used to set the convolution kernel to be used, and line 102 is used to set the summation operation.

Loops 104 to 109 are used to enable the image processing IP to start working.

After preparation, connect the development board as shown in the figure below and insert the OV5640 module into the JP1 interface:

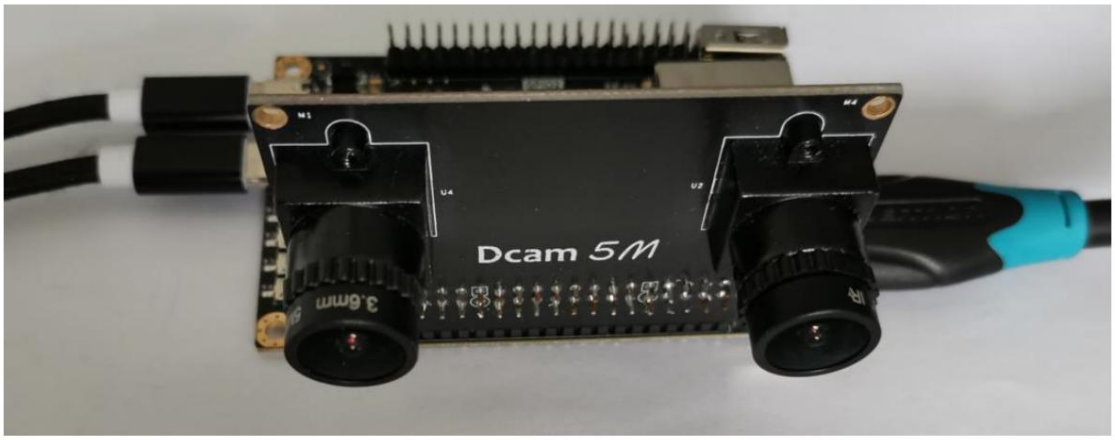


Figure 10- 15

After the compilation is complete, click Run and you can observe the output results after connecting to the monitor. First select impulse_kernel, observe that the grayscale image of the acquired image appears on the display. This is because In HLS, the first thing to do is to convert the image into a grayscale image.

Figure 10-16 The image captured by the camera after grayscale conversion

Next, select Sobel Kernel to observe the output image results. You can see that the output image is an edge.

Figure 10- 17

Next, select edgeKernel to observe the output image results. You can also see that the output image is

Edge, but this edge information is more delicate.

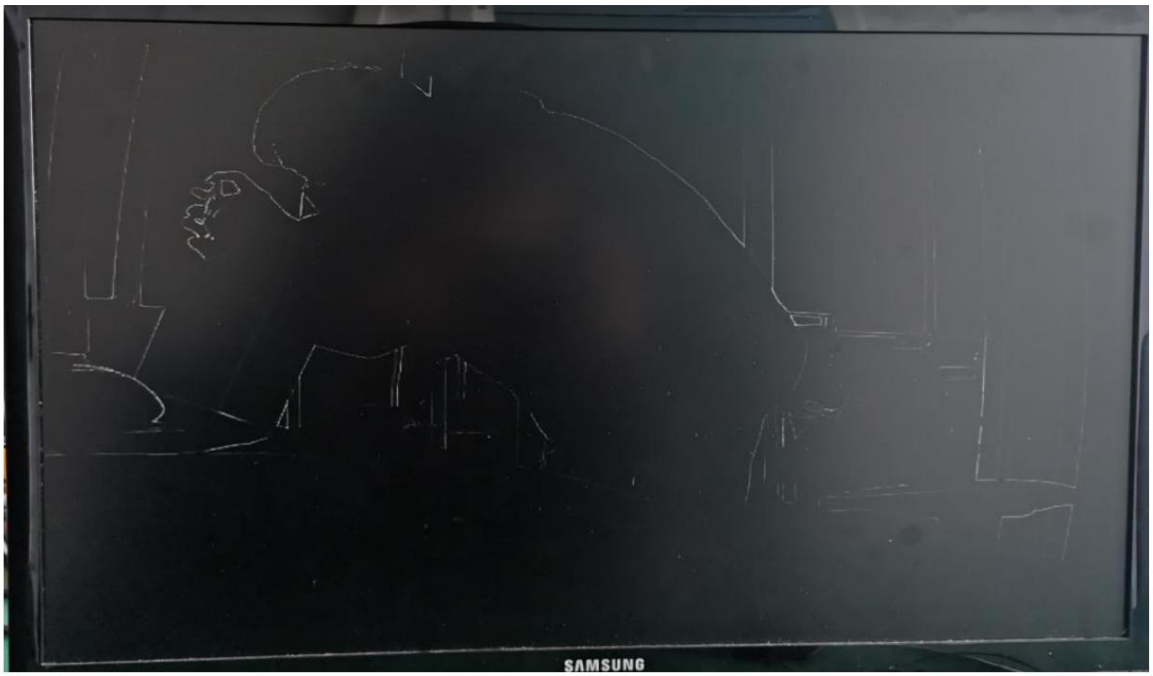


Figure 10- 18

The edge detection function is now complete. It can be seen that the IP can be further improved in HLS.

The entire system can achieve better display effects.

10.6 Experimental Summary

Based on the experiment in the previous chapter, this experiment further uses OV5640 to collect images.

The acquired image is processed and simple edge detection is completed.

It is relatively simple and easy to implement some image algorithms on FPGA.

There are still some problems in the experiment, such as the image has two rows and two columns offset due to LInebuffer.

The data collected by the camera is converted from RGB565 format to RGB888 format, which causes the loss of accuracy.

wait.

In general, this experiment is a relatively comprehensive experiment, from HLS to Vivado to SDK.

Need to master.

Chapter 11 Fast Corner Detection

11.1 Experimental Background

In the previous chapter, we have introduced Sobel edge detection, which is used to detect edges.

Edge is used to judge the change of gradient in horizontal or vertical direction.

The obvious place is the edge.

Corner detection is similar to Soebl edge detection. For edges, as long as there are two directions

If the gradient of is large, it can be judged as an edge. For corners, it is necessary to satisfy its gradient in all directions.

Only when the changes are obvious can it be determined as a corner point.

There is a rigorous mathematical derivation for corner detection. Interested customers can try to read it.

In this experiment, the function provided by Xilinx HLS will be used to complete the corner detection function.

able.

This experiment refers to the demo XAPP1167 provided by Xilinx, which can be found on the Xilinx official

Download from the website, you can also find its functions in the reference files we provide, including Xilinx official

Some other demos are provided for users' reference.

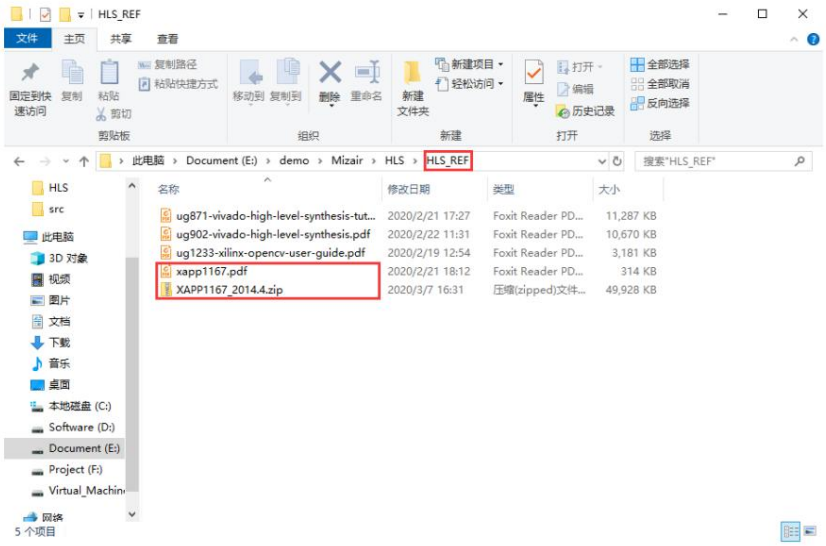


Figure 11- 1

The goal of this experiment is to convert the image collected by OV5640 into a grayscale image and

In the captured image, the positions of the corner points are marked and can be intuitively displayed on the monitor.

The final effect is as follows: You can see that the image collected by OV5640 appears on the display.

The corner points are marked.

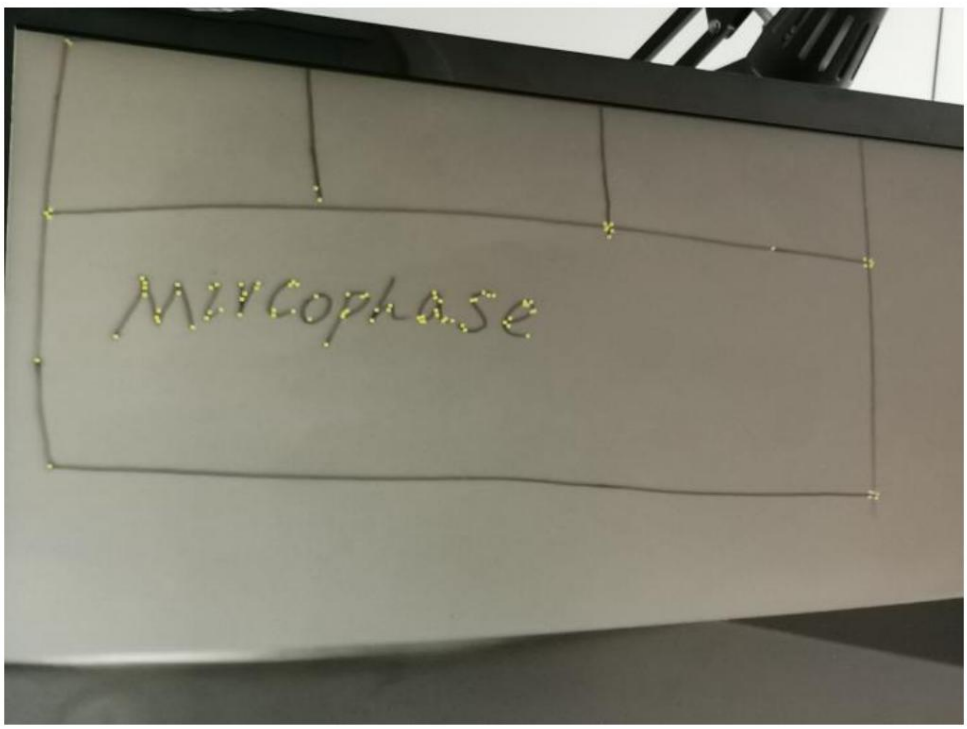


Figure 11- 2

11.2 Corner Detection Using HLS

The definition of the header file is as follows: The header file defines commonly used image formats and pixel formats.

```
1 #ifndef __CORNER_HEAD_H_
2 #define __CORNER_HEAD_H_
3
4 #include "hls_video.h"
5 #include "ap_axi_sdata.h"
6
7 //      Defines the size of the image
8 #define WIDTH 1024
9 #define HEIGHT 768
10
11 //12Define input and output images
#define SRC_IMAGE "screen.bmp"//13 #define        Input image path
DST_IMAGE "DstImage.bmp"//14 #define GOLD_IMAGE Output image path
"GoldImage.bmp"//                                  Reference Image Path
15
16 //   Define image data type
17
18 //19definitionAXI-Stream      Data Flow
typedef ap_axiu<24,1,1,1> int_sideChannel;
20 typedef hls::stream<ap_axiu<24,1,1,1> > AXI_STREAM ;
21
22 //   Define an image matrix, where the pixel format in the image matrix is3  aisle   8  bit unsigned number,  RGB  image
23 typedef hls::Mat<HEIGHT,WIDTH,HLS_8UC3> IMAGE_RGB;
24
25 //   Defines the format of each pixel of the image matrix,  3  aisle   8  Bit unsigned
26 typedef hls::Scalar<3,unsigned char> PIXEL_RGB;
27
28 //29Define an image matrix, where the pixel format in the image matrix is1  aisle   8  bit unsigned number, grayscale image
typedef hls::Mat<HEIGHT,WIDTH,HLS_8UC1> IMAGE_GRAY;
30
31 //32Defines the format of each pixel of the image matrix,  1  aisle   8  Bit unsigned
typedef hls::Scalar<1,unsigned char> PIXEL_GRAY;
33
34
35 //top function
36 void rgb2gray(IMAGE_RGB & imgIn, IMAGE_RGB & imgOut_3C, IMAGE_GRAY &
```

```
imgOut_1C);
37 void doCorner(AXI_STREAM & inStream,AXI_STREAM & outStream);
38 #endif
```

In the top-level function of corner detection, a grayscale conversion function is called, which is responsible for

Convert the input RGB image into a three-channel grayscale image and a single-channel grayscale image.

The grayscale image has a grayscale value on each channel. This is based on the principle of grayscale image display.

To display the grayscale image on the monitor in 24-bit, you need to convert the RGB image to grayscale first.

The grayscale image is then converted to grayscale.

The grayscale image after .

The single-channel grayscale image output is responsible for converting the RGB image into a grayscale image for the next step.

Step processing.

It is worth noting that the HLS library also provides an image conversion function hls::cvtColor, but in this

In this experiment, the reason for writing a grayscale conversion function by yourself is that calling the system's built-in function

The color conversion of the line image will cause the generated IP to fail to meet the timing requirements when it is mapped into the RTL circuit.

Therefore, completing a grayscale conversion function by yourself can effectively avoid the problems caused by timing violations.

```
void rgb2gray(IMAGE_RGB & imgIn, IMAGE_RGB & imgOut_3C, IMAGE_GRAY & imgOut_1C){
    PIXEL_RGB pixIn;
    PIXEL_RGB pixOut_3C;
    PIXEL_GRAY pixOut_1C;

    for(int idxRow = 0; idxRow < HEIGHT; idxRow++){
        for(int idxCol = 0; idxCol < WIDTH; idxCol++){ //The
            input image is a three-channel RGB image
            imgIn >> pixIn;
            unsigned short R = pixIn.val[0];
            unsigned short G = pixIn.val[1];
            unsigned short B = pixIn.val[2];

            // Output three-channel image, each channel is grayscale value, the final displayed image is
            grayscale image pixOut_3C.val[0] = (unsigned char) ((R*76 + G*150 + B*30)
            >> 8); pixOut_3C.val[1] = (unsigned char) ((R*76 + G*150 + B*30) >> 8);
```

```
                    pixOut_3C.val[2] = (unsigned char) ((R*76 + G*150 + B*30) >> 8);

                    // Output single channel image, only one channel, grayscale image
                    pixOut_1C.val[0] = (unsigned char) ((R*76 + G*150 + B*30) >> 8); imgOut_3C <<
                    pixOut_3C; imgOut_1C <<
                    pixOut_1C;
            }
        }
}
```

After defining the function, we can then implement the top-level function. In this experiment, we only need to call the HLS graph

Just like the functions in the processing function library. As follows:

```
1 void doCorner(AXI_STREAM & inStream,AXI_STREAM & outStream){ 2 #pragma HLS
INTERFACE axis port=outStream
3 #pragma HLS INTERFACE axis port=inStream
4 #pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
5
6
7 IMAGE_RGB img_0;
8 IMAGE_RGB img_1;
9 IMAGE_GRAY img_2;
10 IMAGE_RGB img_3;
11 IMAGE_GRAY mask;
12 IMAGE_GRAY dmask;
13
14 #pragma HLS dataflow 15
#pragma HLS stream depth=20000 variable=img_1_.data_stream 16
hls::AXIvideo2Mat(inStream, img_0); 17 PIXEL_RGB
color(255,255,0); 18 rgb2gray(img_0,
img_1, img_2); 19 hls::FASTX(img_2, mask,
20, true);//20 hls::Dilate(mask, dmask);//21              Fast corner
hls::PaintMask(img_1, dmask, img_3,     detection dilates the detected points
color);//22 hls::Mat2AXIvideo(img_3, outStream); 23 }              and adds the dilated points to the image
```

The functions used in this paper can be found in UG902. The following is a brief introduction to these functions.

When I was doing this experiment, I referred to the fast_corner demo in Xapp1167.

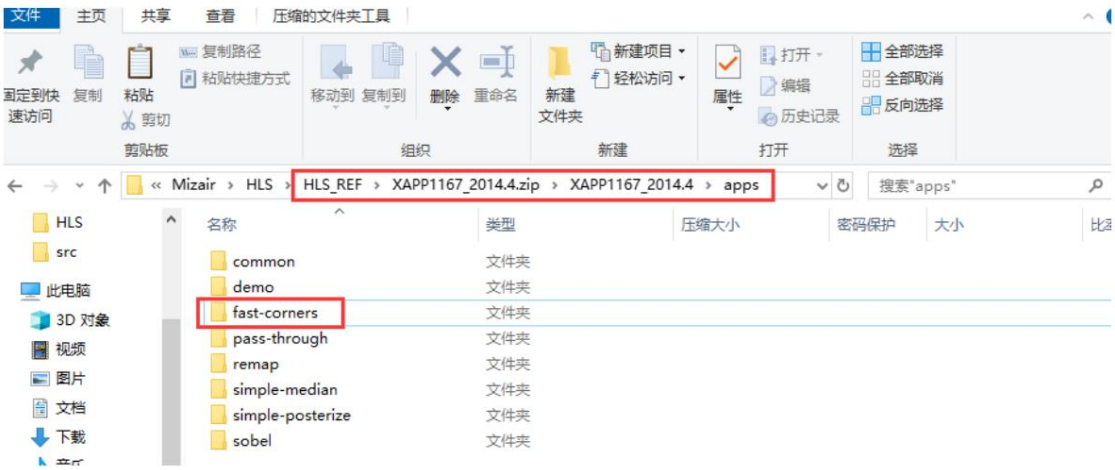


Figure 11- 3

11.2.1 Fast Corner Detection Function

The purpose of the quick corner function: implement a fast corner detector and generate a corner mask or coordinate array

The template of the fast corner detection function is as follows: You can see that this function has two forms. In this experiment

The first form is used.

hls::FASTX

Synopsis

```
template<int SRC_T,int ROWS,int COLS>
void FASTX(
        hls::Mat<ROWS,COLS,SRC_T>    &_src,
        hls::Mat<ROWS,COLS,HLS_8UC1> &_mask,
        int    _threshold,
        bool   _nomax_supression);

template<typename T, int N, int SRC_T,int ROWS,int COLS>
void FASTX(
        hls::Mat<ROWS,COLS,SRC_T>    &_src,
        Point_<T> (&_keypoints)[N],
        int    _threshold,
        bool   _nomax_supression);
```

Figure 11-4 Fast corner detection function

The description of the relevant parameters is shown in the figure below: where src is the original input image.

The mask will be drawn in white at the corner points on a grayscale image of the same size as the input image.

Color marked.

Threshold is the threshold used to determine whether it is a corner point. The larger the value, the more obvious the corner point detection result is.

The area shown as a corner point is smaller.

Nomax_supression: If true, non-maximum suppression is enabled. This parameter is used for corner point detection.

is derived from the mathematics of , and holds true in this experiment.

Table 4-31: **Parameters**

Parameter	Description
src	Input image
mask	Output image with value 255 where corners are detected
keypoints	Array of the coordinates of detected corners
threshold	FAST detector threshold. If a pixel differs from the center pixel of the window by more than this threshold, then it is either a light or a dark pixel.
nomax_supression	If true, then enable suppression of non-maximal edges

Figure 11-1 Parameter meaning

11.2.2 Dilation Function

As shown in line 20 of the code , hls::Dilate(mask, dmask); performs a dilation operation on the image.

In the previous chapter, we have introduced the concept of expansion. In this experiment, we call the built-in function of HLS to

Dilate the image, mask is the input image, dmask is the image after dilation.

It is to make the corners more obvious.

hls::Dilate

Synopsis

Default:

```
template<int ROWS, int COLS, int SRC_T, int DST_T>
void hls::Dilate (
     hls::Mat<ROWS, COLS, SRC_T>& src,
     hls::Mat<ROWS, COLS, DST_T>& dst);
```

Custom:

```
template<int ROWS, int COLS, int SRC_T, int DST_T, int K_ROWS, int K_COLS, typename
K_T, int Shape_type, int ITERATIONS>
void hls::Dilate (
     hls::Mat<ROWS, COLS, SRC_T>&      src,
     hls::Mat<ROWS, COLS, DST_T>&      dst,
     hls::Window<K_ROWS, K_COLS, K_T> & kernel);
```

Figure 11- 2

11.2.3 Drawing the Mask

As shown in line 21 , hls::PaintMask(img_1, dmask, img_3, color) is completed.

The positions of the mask and the input image are marked with a special color. Color has been defined in line 17.

A color pixel is defined in line 17: PIXEL_RGB **color(255,255,0);**

The definition of the paint mask function is as follows:

hls::PaintMask

Synopsis

```
template<int SRC_T,int MASK_T,int ROWS,int COLS>
void PaintMask(
        hls::Mat<ROWS,COLS,SRC_T>        &_src,
        hls::Mat<ROWS,COLS,MASK_T>       &_mask,
        hls::Mat<ROWS,COLS,SRC_T>        &_dst,
        hls::Scalar<HLS_MAT_CN(SRC_T),HLS_TNAME(SRC_T)> _color);
```

Figure 11- 3

Table 4-47: **Parameters**

Parameter	Description
src	Input image
mask	Input mask
dst	Output image
color	Color for marking

Figure 11- 4

Finally, the position corresponding to the corner point is marked with a specific color.

For the functions used this time, we will briefly introduce them here. You can see that the library provided by HLS is used.

Can greatly speed up development efficiency.

11.2.4 Adding **Directives**

In this experiment, one of the directives that needs to be added is very important.

directive, the RTL simulation stage of C/RTL co-simulation always fails. After referring to xapp1167

Adding this directive finally solved the problem.

```
#pragma HLS stream depth=20000 variable=img_1_.data_stream
```

It is basically able to buffer the data stream, which is equivalent to delaying the data stream. Otherwise, if it is not used temporarily,

If you perform other operations while using the data in this stream, you will no longer be able to access the data in this stream.

The data in the format is one-way flow similar.

11.3 HLS Simulation and Synthesis

The original image looks like this:

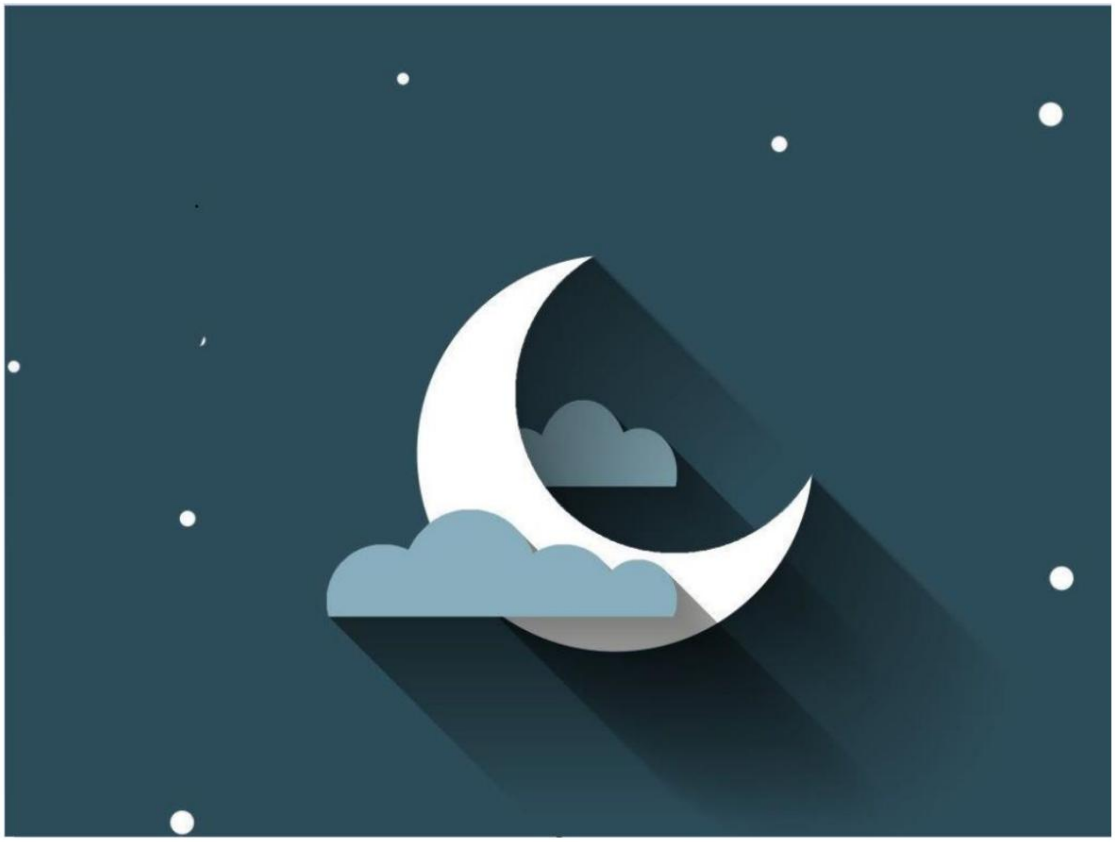


Figure 11-5 Original image

After adding the C simulation file, you can start C simulation. The result of the simulation is shown in the figure:

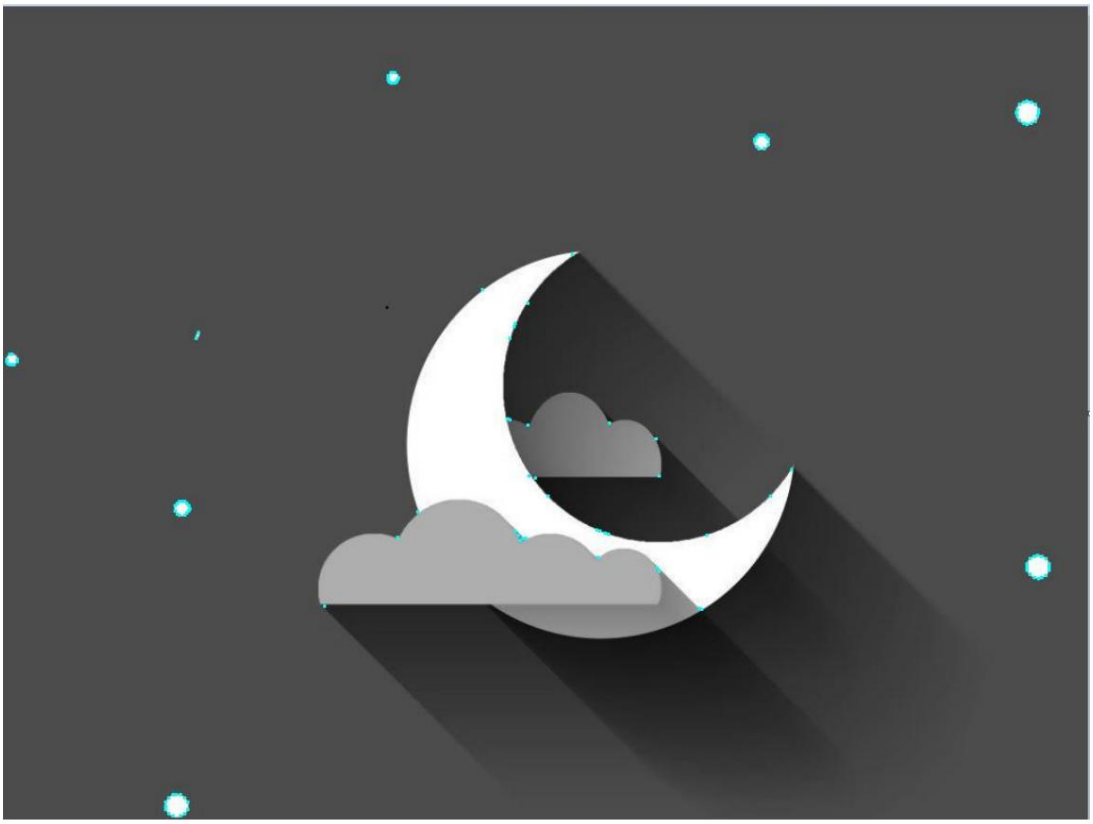


Figure 11- 6

You can see the results of the C simulation. The image has been converted to a grayscale image and the corner points have been Mark out.

Next, you can perform C synthesis and C simulation. After the simulation is completed, you can export the IP to Vivado and Used in SDK.

11.4 Building **Block Design**

Compared with the previous experiment, the Block design of this experiment only needs to change the IP generated by HLS The other modules can remain unchanged. The final Block Design is as follows:

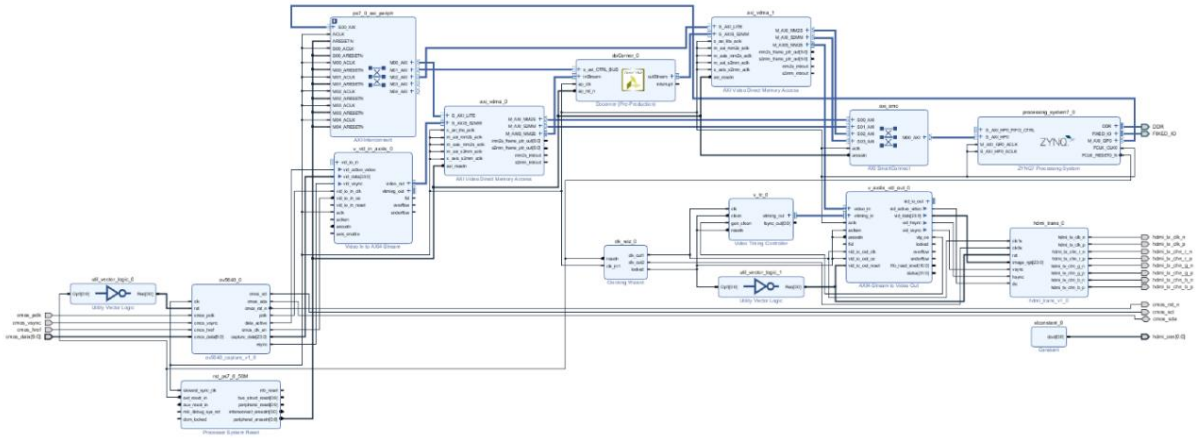


Figure 11- 11

After generating the bit file, import it into the SDK for SDK development. The SDK development for this experiment is also very simple.

No more further explanation here.

11.5 Download Verification

When everything is ready, connect the development board as shown below:

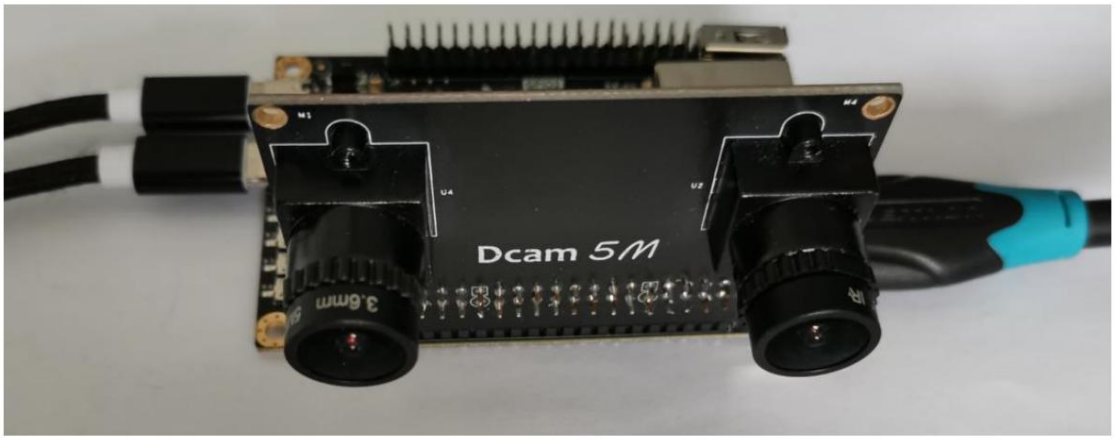


Figure 11- 12

Then start the program through the SDK and click Run

You can see the display results on the monitor as shown below:

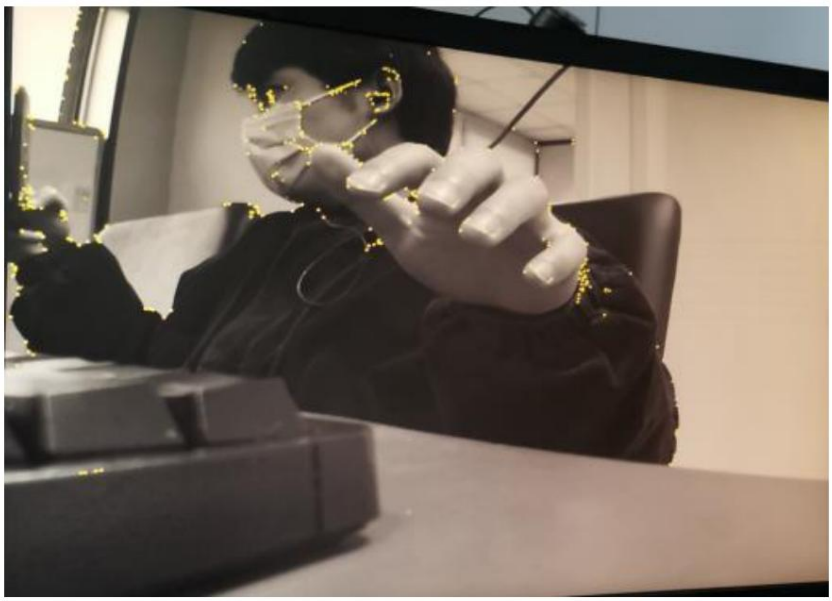


Figure 11- 13

You can see that the mark point has appeared in the screen displayed on the monitor. Next, draw a square box to see if the points in the box can be detected.

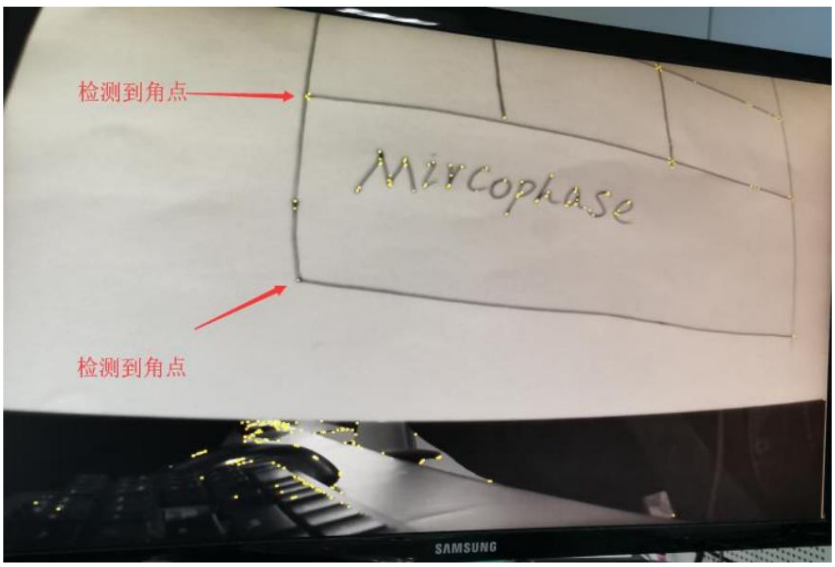


Figure 9- 19

It can be seen that the position of the corners in the picture has been successfully detected.

11.6 Experimental Summary

This experiment uses the image processing function in HLS to complete the function of fast corner detection.

Using built-in functions can speed up development efficiency. It should also be noted that fast corner detection is not very effective for noise indices.

Therefore, there are many points in the picture that are not detected correctly. This can be corrected by using Gaussian filtering or other methods.

This method can reduce noise and improve the accuracy of corner detection.

Summary of the first edition

Thank you for choosing the products of Microphase Technology (Shanghai) Co., Ltd. This document is intended to help users use Z7-Lite ZYNQ7000 series development board for HLS development quick start, I believe that users have read this complete

After the tutorial, you will have a preliminary understanding of using HLS for development.

Supporting source code, users can observe the experimental phenomena according to the source code and project, as the first time I wrote HLS technical documentation is bound to contain errors. If you find any, please contact us immediately. I will

This document will be modified in a timely manner.

The content of this document will not be static and will be updated in the future with more information about HLS development.

Project examples, please continue to pay attention to it, and finally I wish you all the best in your work.