ZYNQ MPSoC Development Platform

HSL Tutorial

ALINX

Version Record

Version	Date	Release By	Description
Rev1.0	2021-04-10	Rachel Zhou	First Release

We promise that this tutorial is not a permanent, consistent document. We will continue to revise and optimize the tutorial based on the feedback of the forum and the actual development experience.

Preface

vitis HLS enhances the level of abstraction in system design and provides tangible help to designers. vitis HLS improves the level of abstraction in two ways: First, use C/C++ as the programming language to take advantage of the advanced structures available in the language. Second, provide more data primitives that allow designers to build blocks (bit vectors, queues, etc.) using the underlying hardware. These two features help designers to easily solve common protocol system design challenges with vitis HSL compared to using RTL. This simplifies system assembly, simplifies FIFO and memory access, and abstracts the control flow.

vitis HSL facilitates architectural research, and users can pass the required features of the design to the synthesis tool simply by inserting program instructions into the code (such as the Tcl command when using the GUI or batch mode). This allows the user to explore a large number of alternative architectural scenarios without modifying the design code itself. The scope of the research can be fundamental issues such as module pipeline, or it can be a common problem such as FIFO queue depth.

vitis HSL is easy to simulate. C and RTL simulations are another great place for vitis HLS. The design generally uses a two-step process verification: the first step is “C language simulation”. The compilation and execution of C/C++ in this step is the same as the common C/C++ program; the second step is “C/RTL co-simulation”. In this step, vitis HSL will automatically generate an RTL test platform based on the C/C++ test platform, then set up and execute the RTL simulation to check the correctness of the implementation.

If you can make full use of these advantages, this will be of great benefit to the user's system design. This is not only reflected in development time and productivity, but also due to the more compact nature of the vitis HSL code, which is reflected in code maintainability and readability. In addition, through high-level synthesis, users can still effectively control the architecture and its features. Proper understanding and use of the vitis HSL program is fundamental to achieving this control.

Content

Preparation and Precautions

This tutorial is based on the fact that learners are already proficient in Vivado and vitis. It is recommended to be familiar with the course_s2_vitis part before reading this tutorial, and not to explain in detail the Vivado project establishment, vitis bare metal software writing, interrupt handling, etc. HLS requires learners to be proficient in C++, have a certain basic knowledge of hardware, and have a deep understanding of hardware pipeline, combinatorial logic, sequential logic, FIFO, RAM, AXI bus protocol to learn, so as to avoid low-level errors.

Software Environment

This tutorial is based on the software version VITIS HLS. The shortcut icons to be used here are:

vivado Software

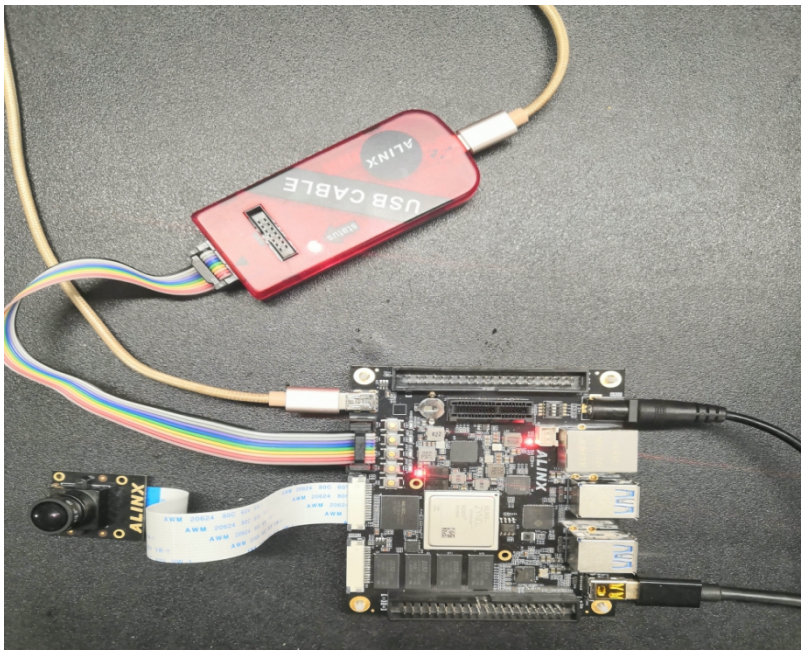
vitis hls Software

Hardware Environment

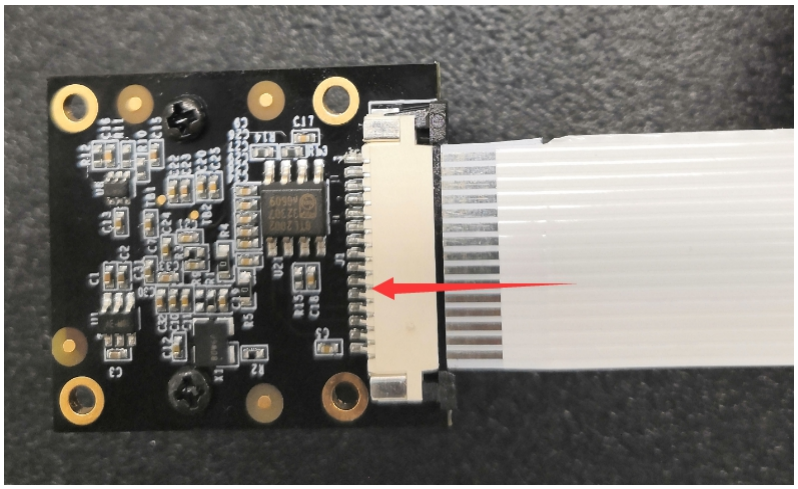
ALINX Brand ZYNQ FPGA Development Board, mipi camera module (Module AN5641). The list of currently supported models is as follows:

FPGA Development Board Module	Mipi Camera Module	ZYNQ Chip Module
AXU2CGA/AXU2CGB	AN5641	Xczu2cgsfvc784-1-i
AXU3EG	AN5641	Xczu3egsfvc784-1-i
AXU4EV-E/AXU4EV-P	AN5641	Xczu4evsfvc784-1-i

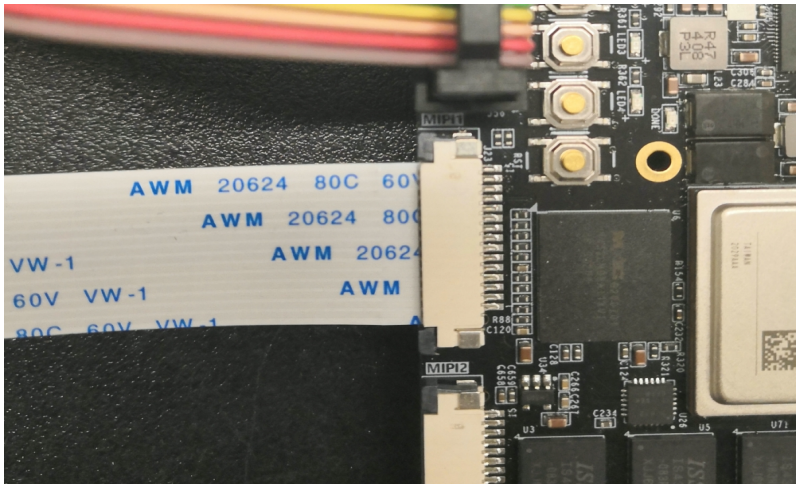
AXU5EV-E/AXU5EV-P	AN5641	Xczu5evsfvc784-1-i
AXU7EV	AN5641	Xczu7evffvc1156-1-i
AXU9EG	AN5641	Xczu9egffvb1156-1-i
AXU15EG	AN5641	Xczu15egffvb1156-1-i



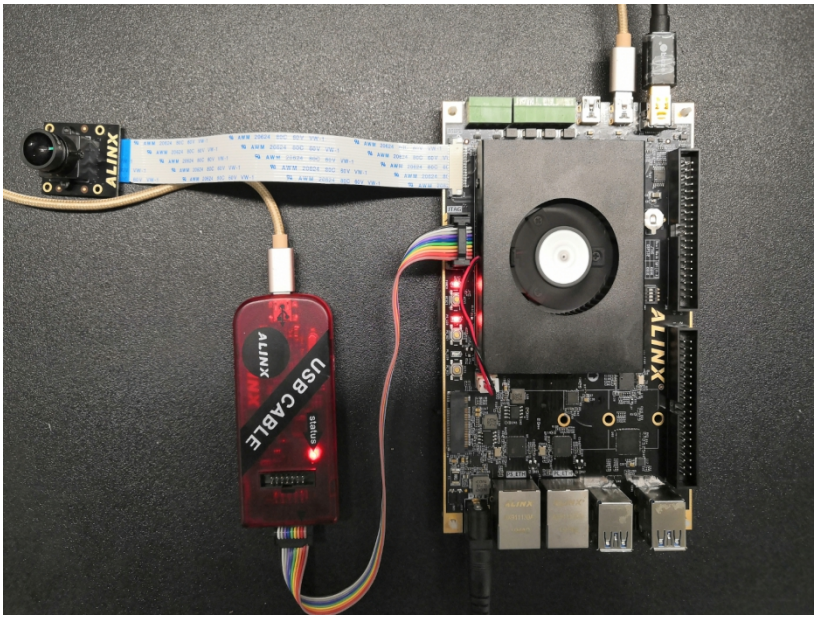
AXU2CGB/A FPGA Development Board



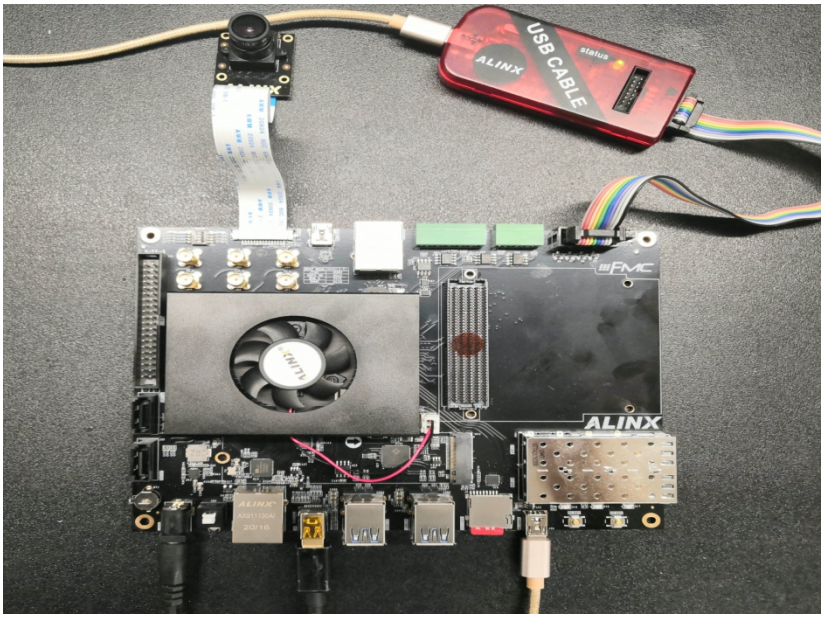
MIPI Camera Connection Method



MIPI Cable Connection



AXU3EG/AXU4EV/AXU5EG



AXU7EV/AXU9EG/AXU15EG

In addition, you need to prepare one HDMI display (supports 1080P60).

Experimental Project and Directory Description

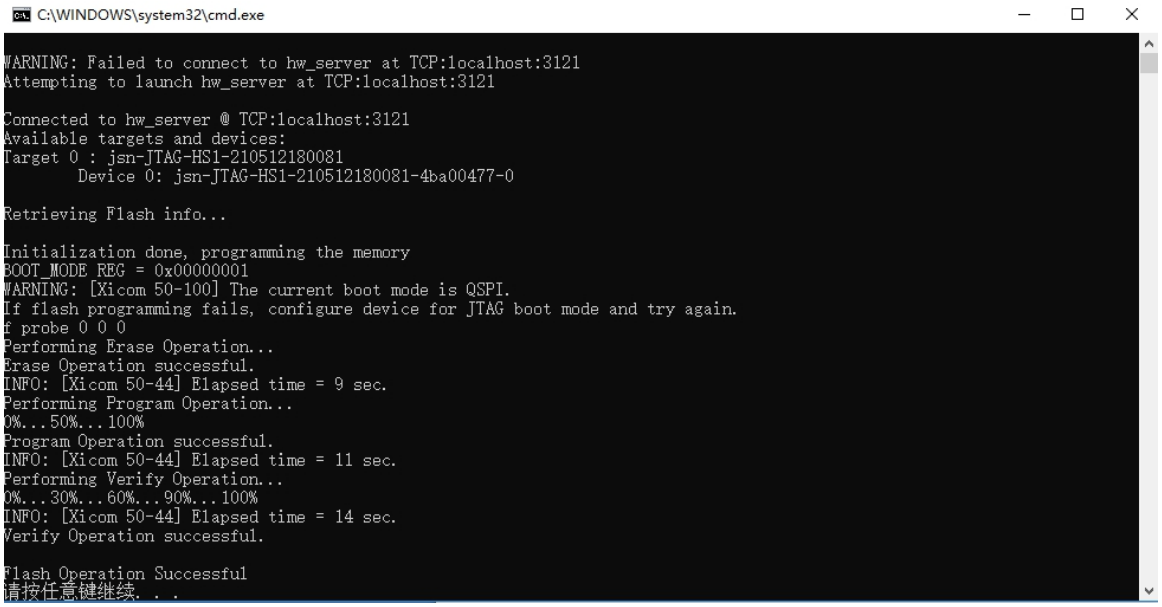
Project Name	Directory path	Description
HLS Project	hls	Each algorithm exports IP, referenced by vivado
vivado Project	vivado	
Vitis Project	vivado/xx_project/xx_.vitis/	software program
Common ip	vivado/ip_project	Verilog package ip
HLS export IP	vivado/xx_project	Add this path when Vivado adds IP core
flash_load_comm	flash_load_comm	qspi burning

Rapid Experiment Recurrence

1) Connect the “jtag” interface
2) Set the boot mode to “QSPI” boot
3) Copy the file “BOOT.bin” in the corresponding SDK project to the “flash_load_comm” directory.

4) Power on the fpga development board

5) Click the program_qspi.bat script to run. After burning successfully, the interface is as follows



6) Re-power the development board

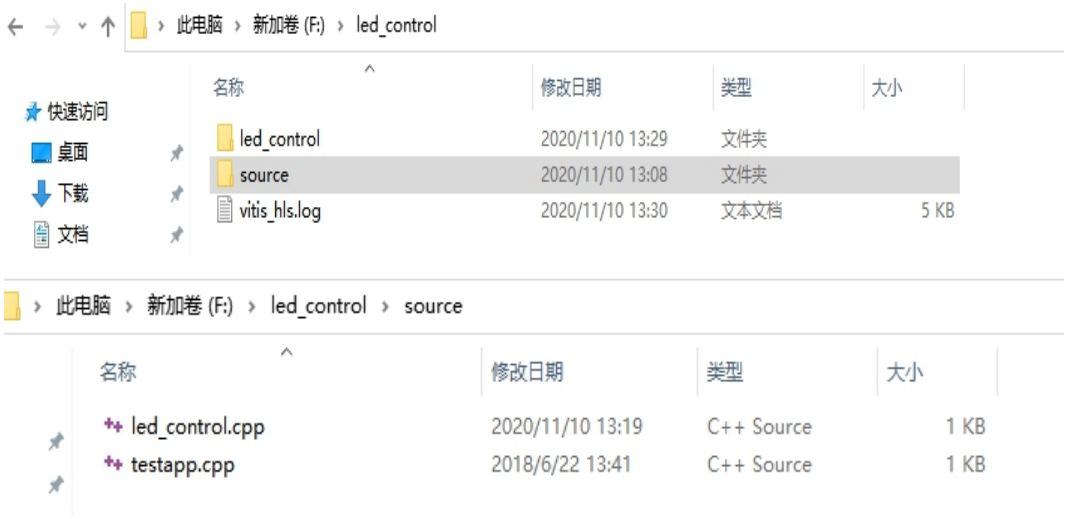
Note: If the “SDx” software installation location is not the default C drive, “program_qspi.bat” will run incorrectly. In this case, you need to modify the command path in “program_qspi.bat”.

Part 1: Getting to Know Vitis HLS

Part 1.1: Basic operation of Vitis HLS

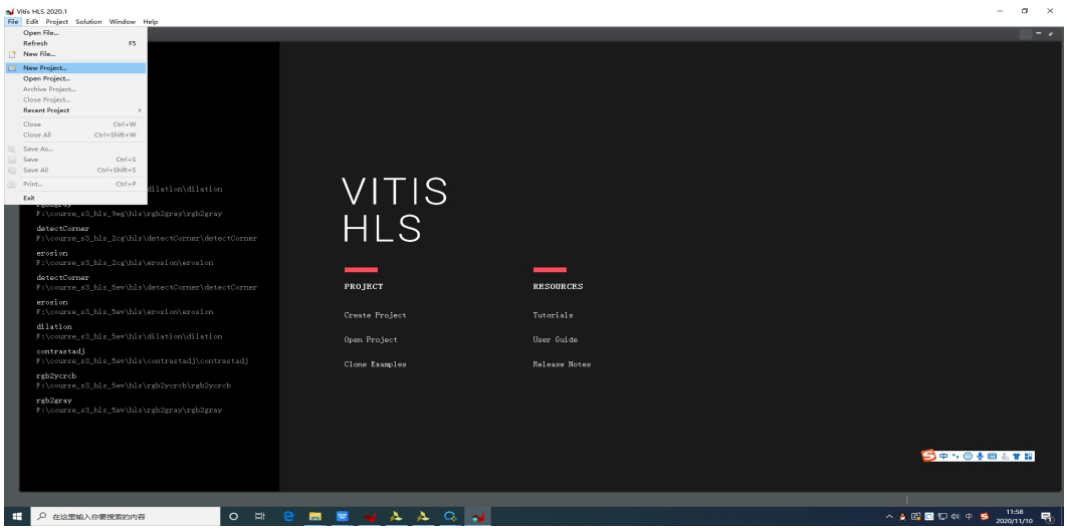
Part 1.1.1: Create a “vitis HSL” Project (Take Part 2 Project as Example)

1) Create a new folder, name it with the project name, create a subfolder source in the folder to store code files



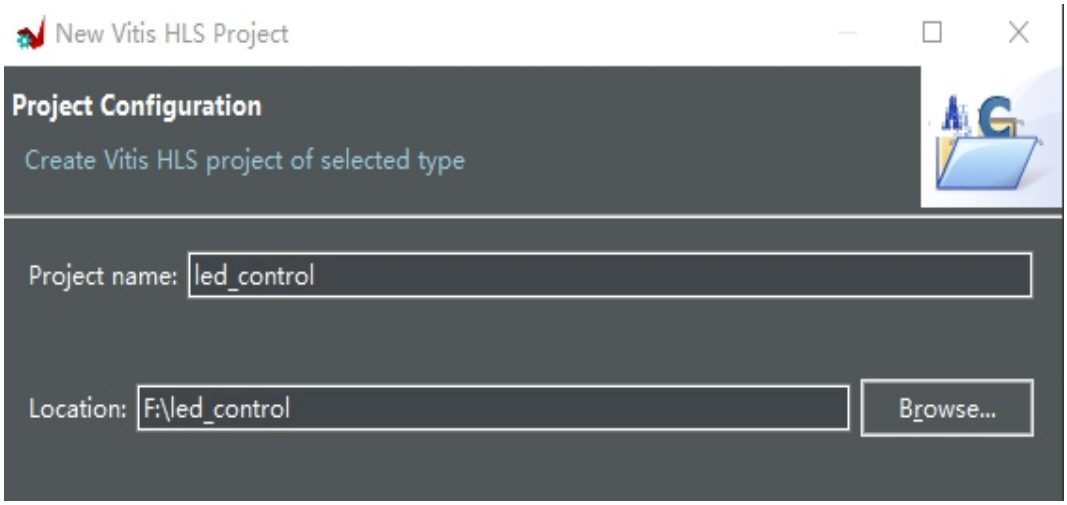
2) Open vitis HLS

3) Select Create Project on the initial interface or “File-> New Project” in the upper left corner

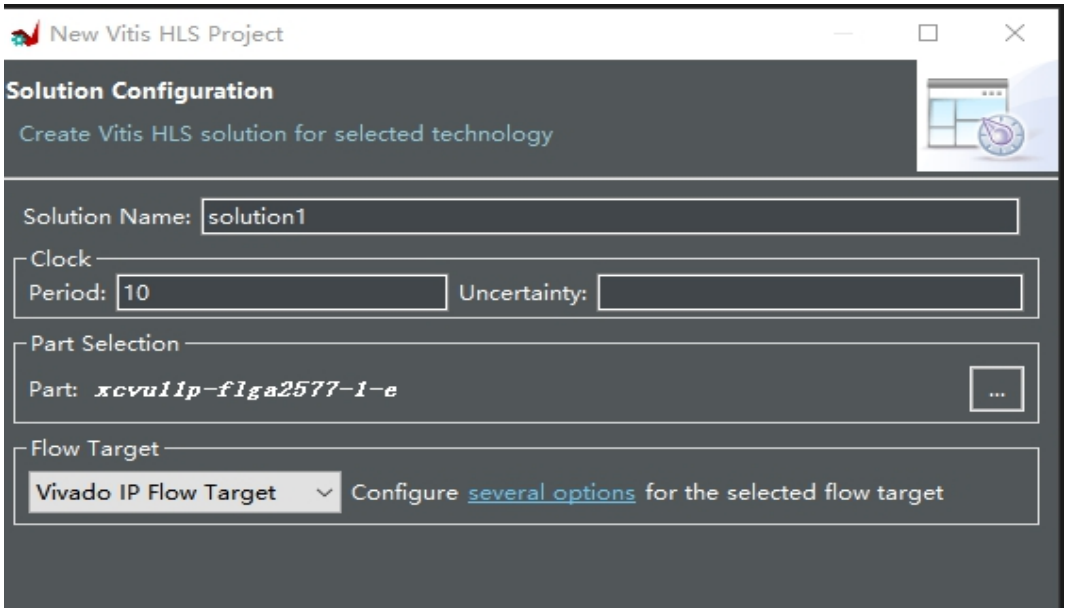


4) Set the project name and select the storage path (here, take the

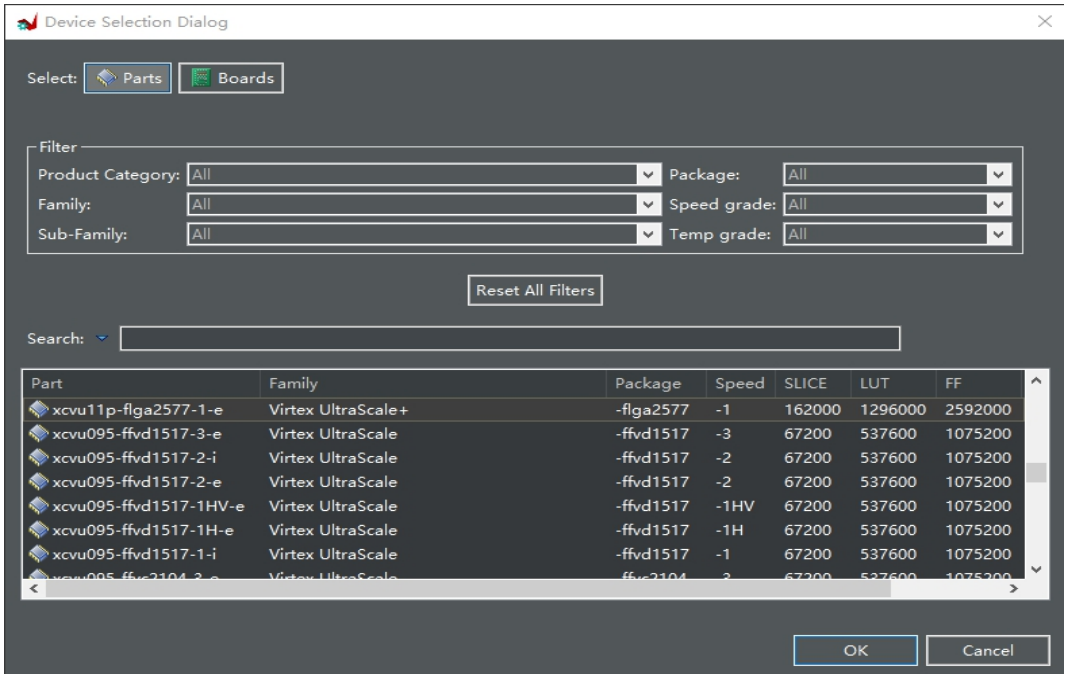
experiment in part 2 as an example), as shown in the figure



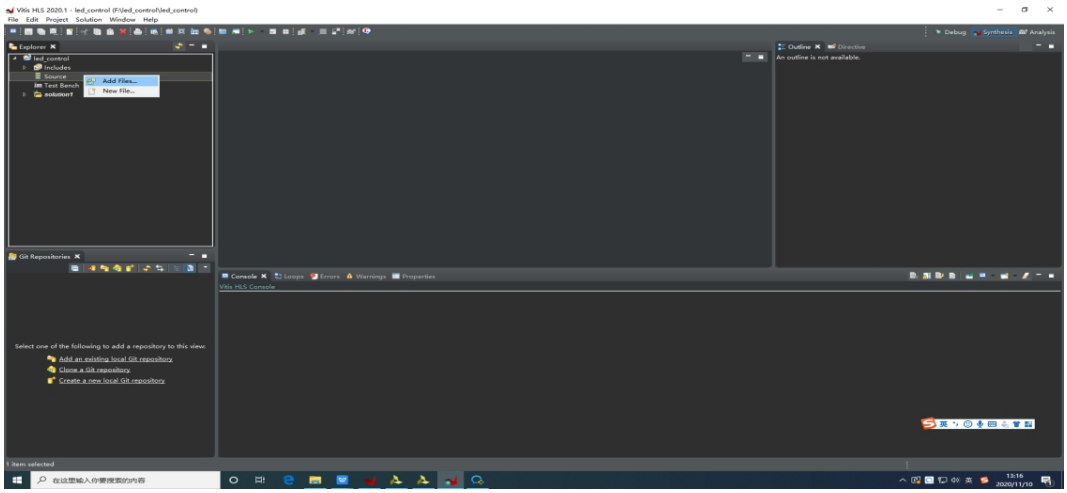
5) Next→Next to the following interface, in the Clock Period column, you can fill in the period of the running clock. The software will optimize the logic according to this clock. You can fill in according to actual needs. The default is 10ns, and the clock uncertainty is 12.5% of the clock period by default.



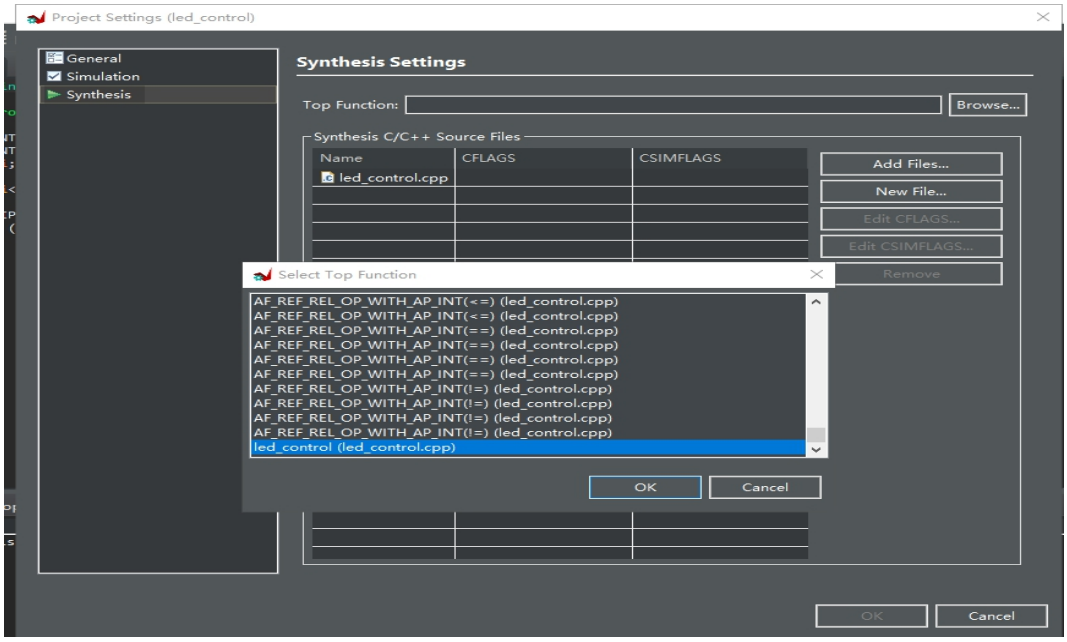
6) Click on the icon ... to go to the following interface



7) Enter the ZYNQ chip model in the Search input box, select the corresponding device in the drop-down list box, and click the "OK"
8) Click the "Finish" to complete the creation of the vitis HLS project
9) Right-click the "Source" icon, select "Add Files...", find the "led_control.cpp" file in the source folder and add it to the project



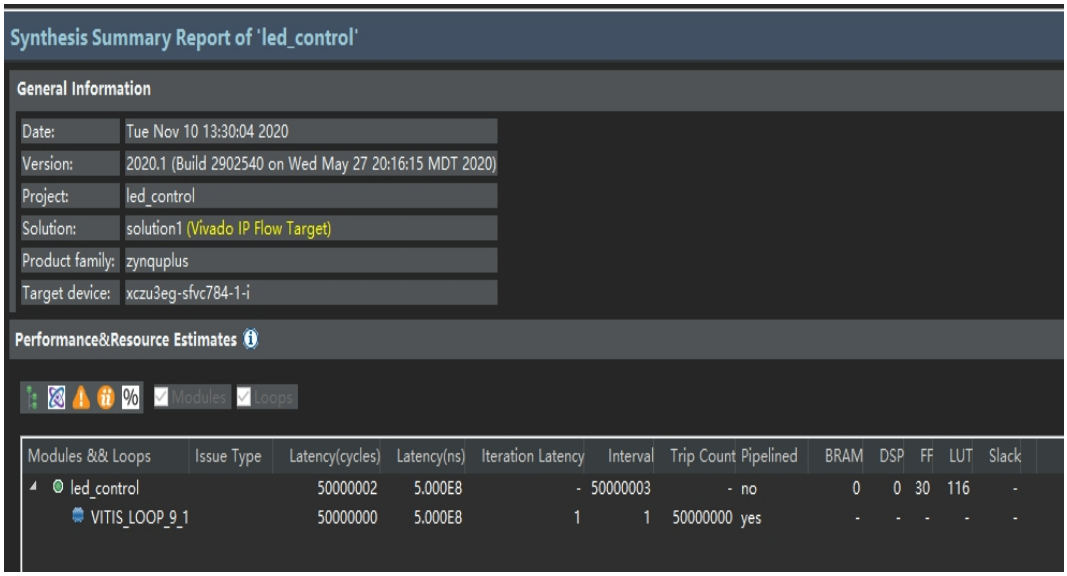
10)Double click to open "led_control.cpp"
11)"Ctrl+S" save the current document content
12)Click the icon  to set top_function



Note: vitis HLS will display all related functions, usually the top-level function we need to set is near the bottom

13) Click the icon  to synthesize, or click Solution→Run C Synthesis→Active Solution in the menu bar

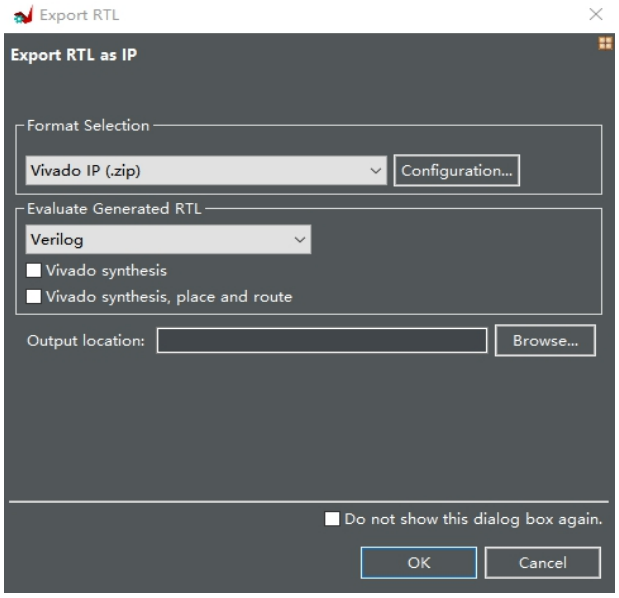
14) After synthesis, we usually pay attention to observe the occupied resources and some timing information to make a preliminary judgment on whether it meets expectations. You can click on the icon  to open the comprehensive evaluation report after synthesis.



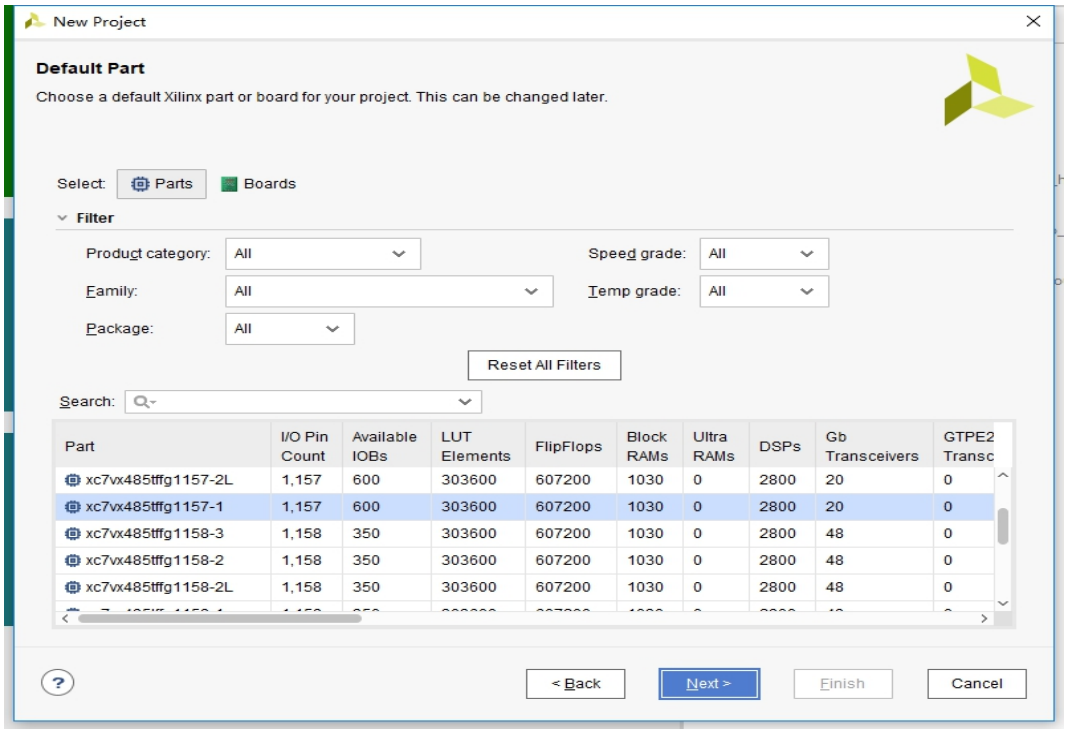
Unlike vivado HLS, vitis HLS displays timing and resource occupancy information together. Latency (cycles) indicates how many clock cycles are required to complete the execution of the function, latency (ns) indicates the execution time of the function in ns. iteration Latency represents the iteration delay in the loop, interval represents the execution interval of the loop iteration, trip Count represents the estimated maximum execution time in the loop, pipelined indicates whether to perform pipeline operation, which will be introduced in detail in the following chapters. Bram represents the block RAM occupancy rate, DSP represents DSP occupancy rate, FF represents the trigger occupancy rate, LUT represents the occupancy rate of the lookup table, Slack indicates whether there is a timing violation. Pay special attention to the look-up table as the most important resource in FPGA. Usually insufficient resources refer to insufficient LUT resources. The following chapters will introduce commonly used area optimization methods to reduce LUT resource consumption.

Part 1.1.2: Vitis HLS Generates IP Core

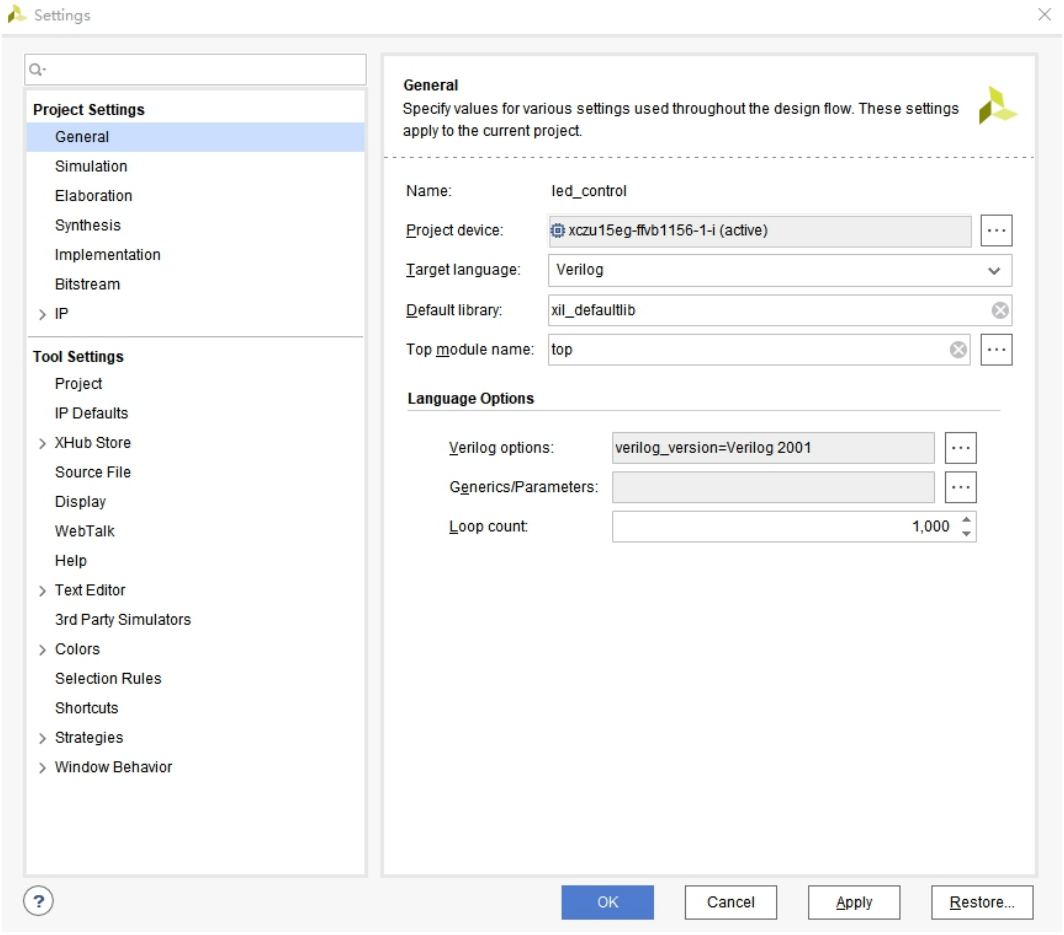
1) Click the icon  to export the IP core



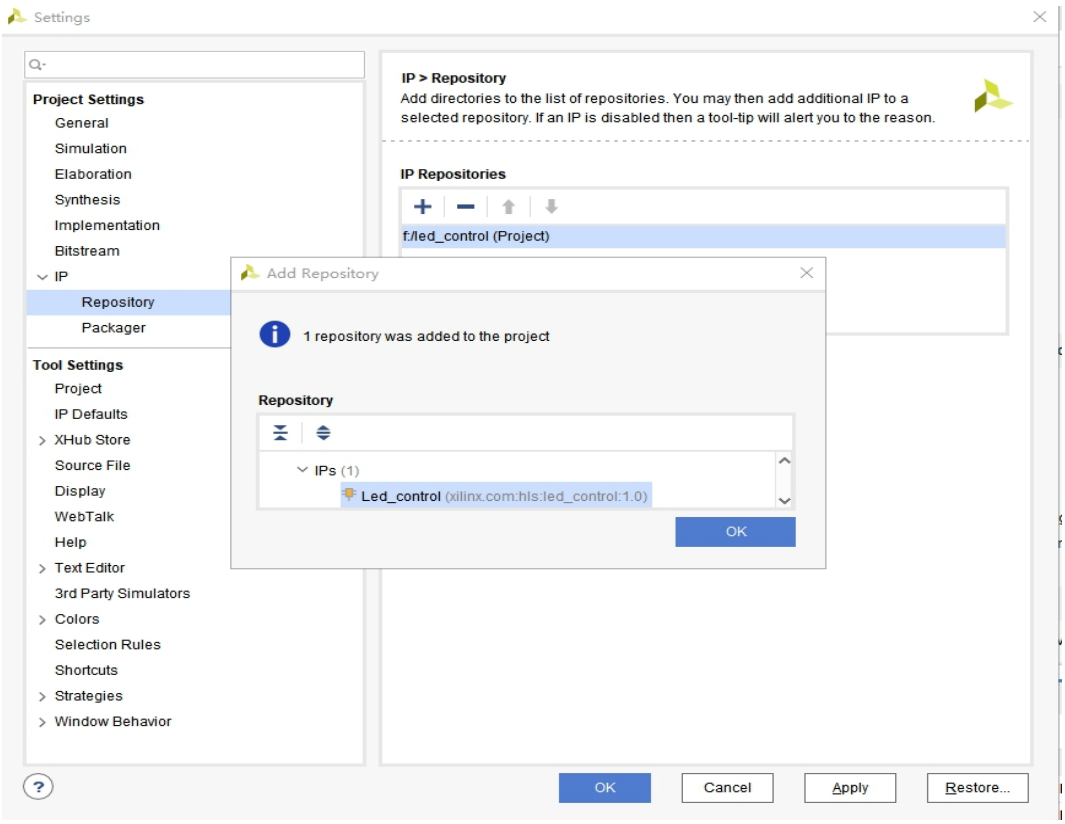
2) Open the vivado software after exporting
3) Click the "Create Project" tab, the project creation wizard pops up
4) Next
5) Set storage path and project name
6) Next→Next→Next→Next, to the following interface



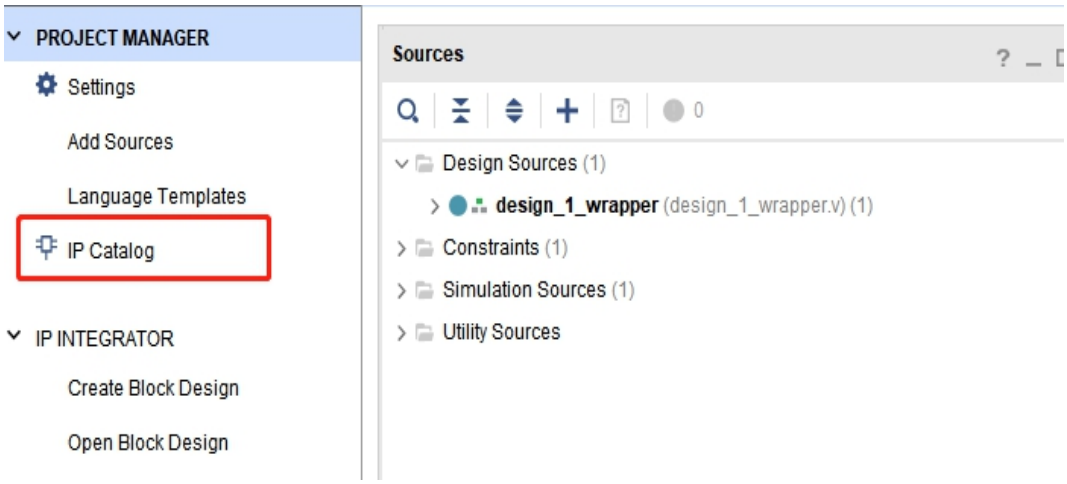
7) Enter the ZYNQ chip model in the "Search" box, and select the device in the drop-down list box. Note that the device must be consistent with the device in "vitis HLS"
8) Next→Finish, complete the creation of the vivado project
9) Enter the vivado software project interface, click the icon  to open the following interface:



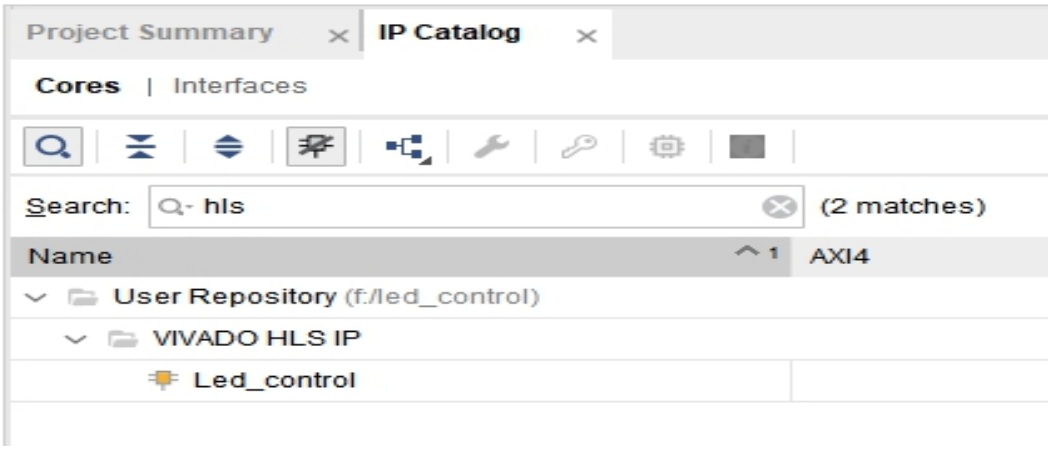
10)Select "IP->repository", add IP core path



11)Select "IP Catalog" on the left



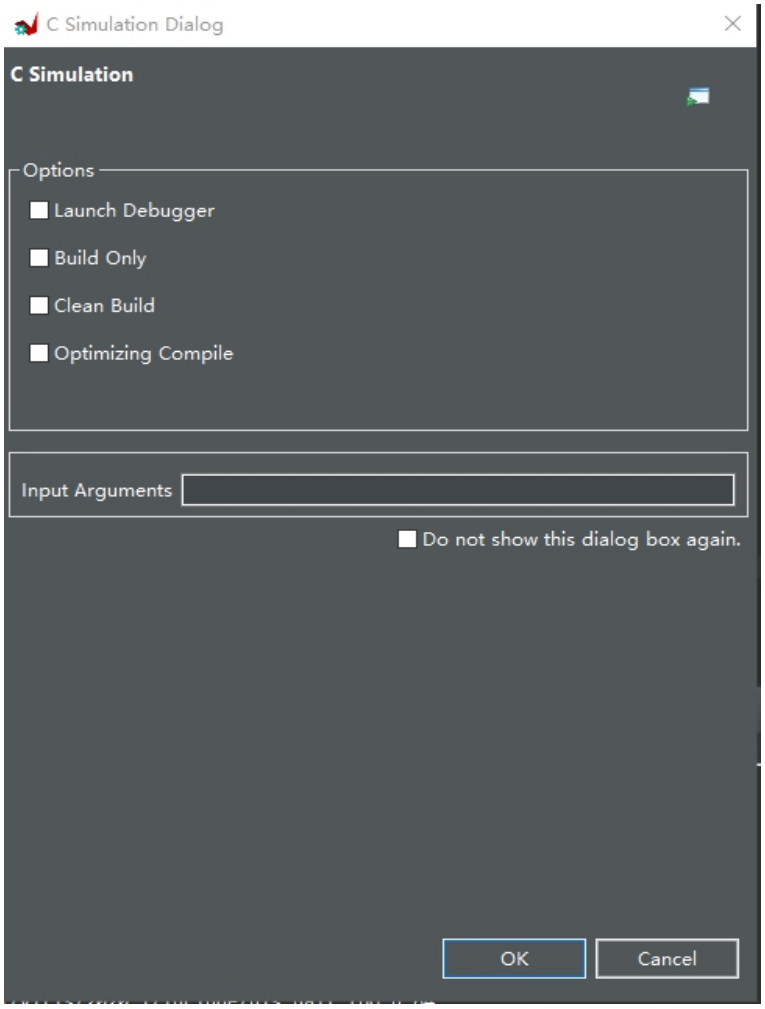
12)In the interface below, enter "hls" to see the added IP core, at this time our IP core has been added



Part 1.1.3: Vitis HLS Simulation

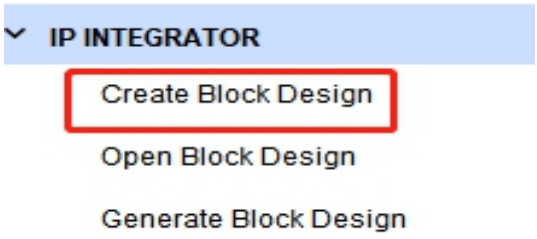
The simulation in Vitis HLS is divided into c simulation and C_RTL cross simulation. Usually, c simulation is performed in HLS first, and IP core is generated after c simulation, and the IP core is added to vivado for simulation to observe the waveform diagram

1) Click the icon  , the pop-up interface is as follows

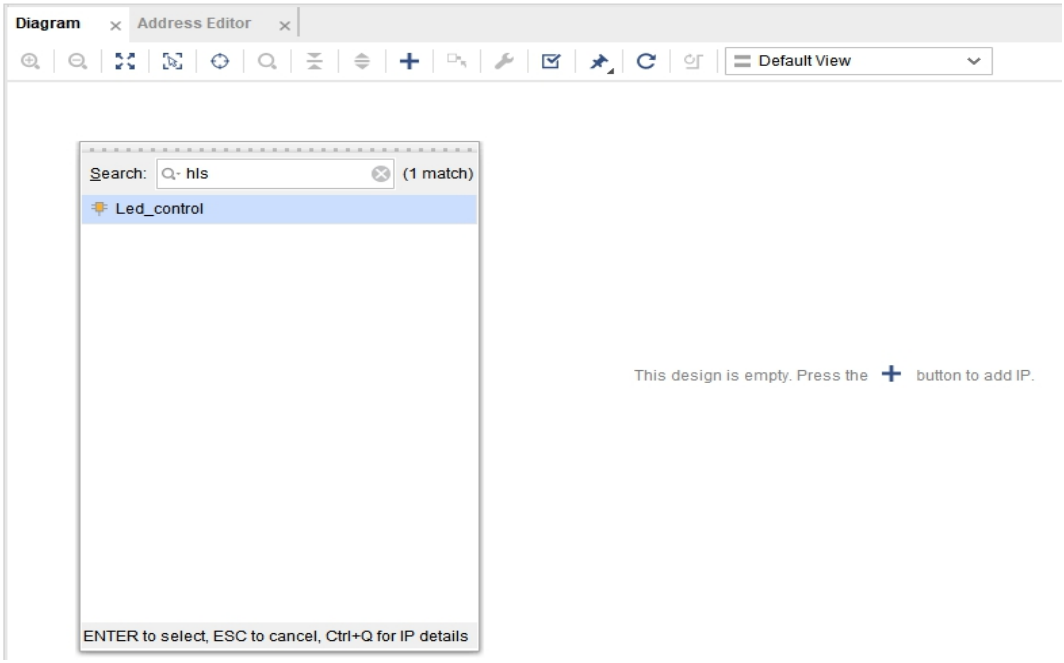


2) If there are no parameters in the main function of the test file, you can directly keep the default settings, click ok to observe the simulation effect

3) After c simulation, perform "c_rtl" cross simulation, add the IP core to vivado according to the previous steps, and open the icon "create block design"



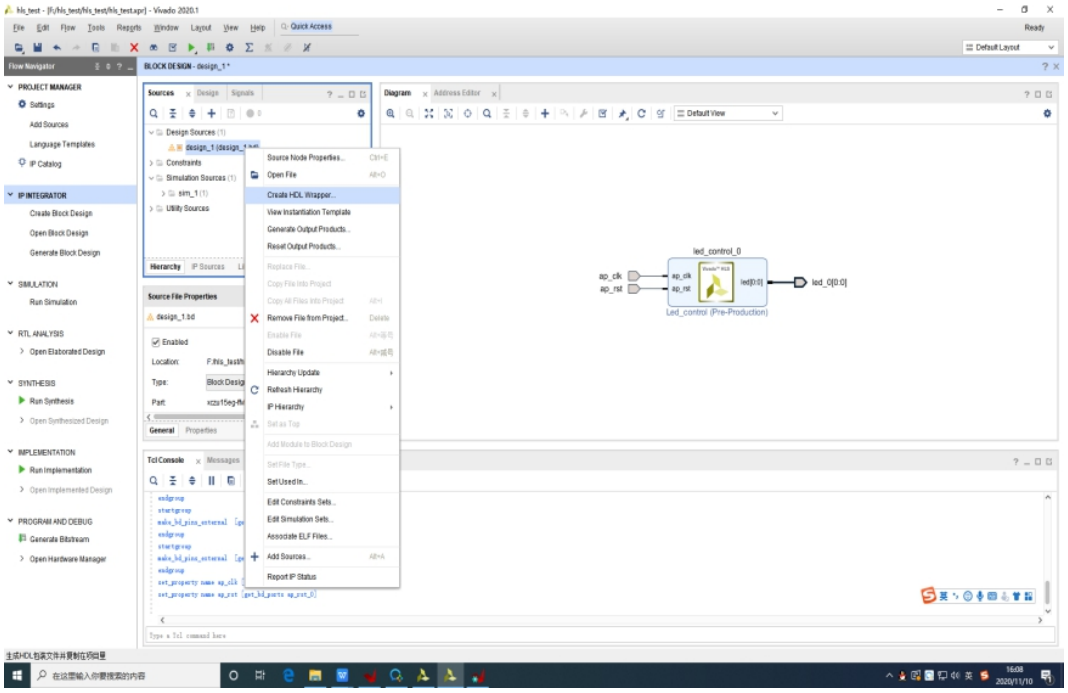
4) Click the plus sign in the displayed "block design" to add the "IP" core (take the entry project in Chapter 2 as an example)



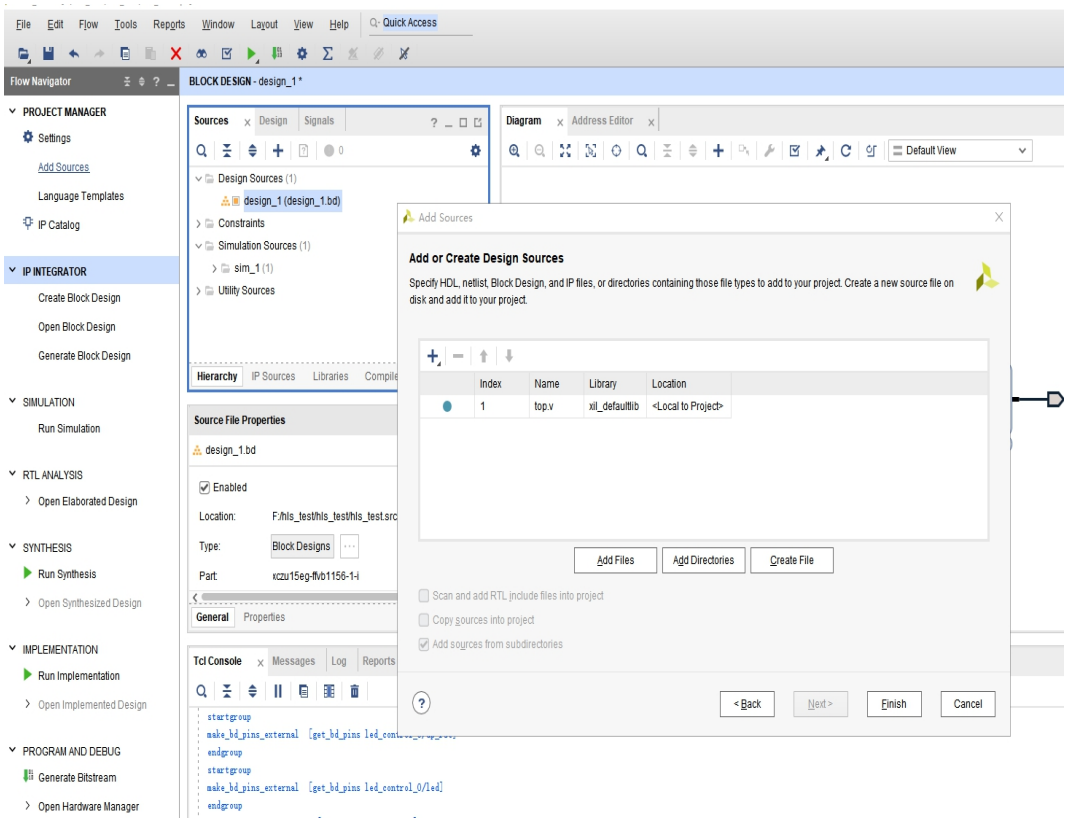
5) At this point, you can see that our IP core is added to the visualization interface

6) Press “ctrl+T” to generate port on ap_clk, ap_rst, led

7) Right click and select “Create HDL Wrapper”



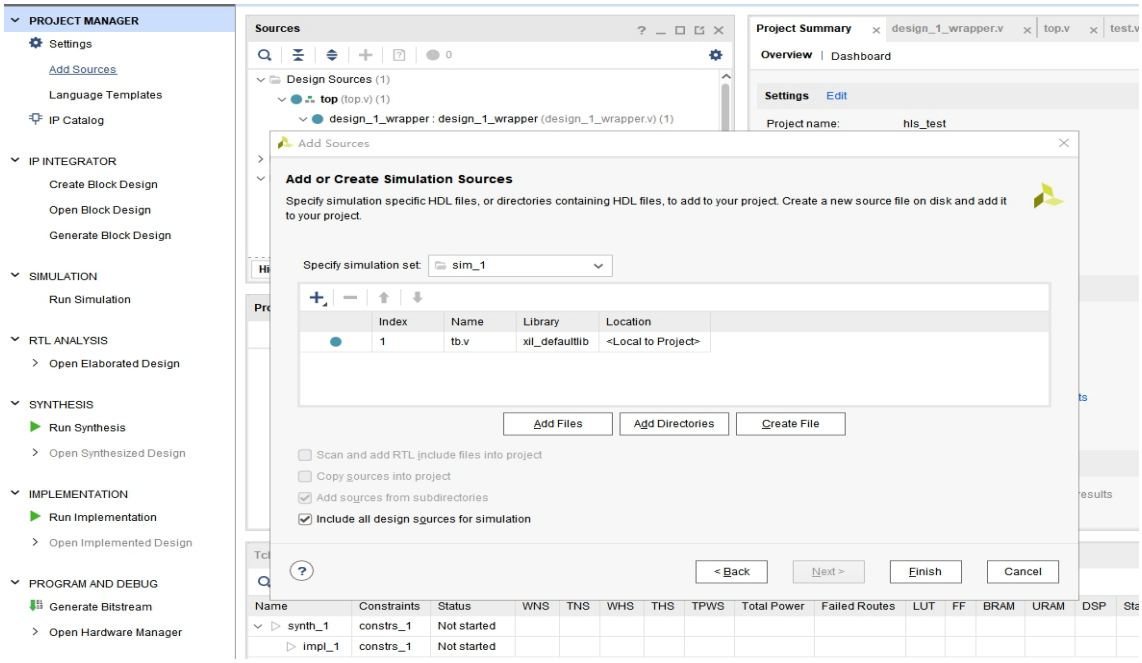
8) Create “top.v” file



9) Enter the following code in the “top.v” file

```
module top(
    input clk ,
    input ap_rst
    );
//internal variable declation

//regs
    reg [31:0]cnt=32'd0 ;
//wires
    wire led ;
//main codes

always @ ( posedge clk )
begin
    if ( ap_rst == 1'b1 ) begin
        cnt <= 32'd0 ;
    end
    else begin
        cnt <= cnt + 32'd1 ;
    end
end

    design_1_wrapper design_1_wrapper(
        .ap_clk( clk ) ,
        .ap_rst ( ap_rst ) ,
        .led_0 ( led )
        ) ;
endmodule
```

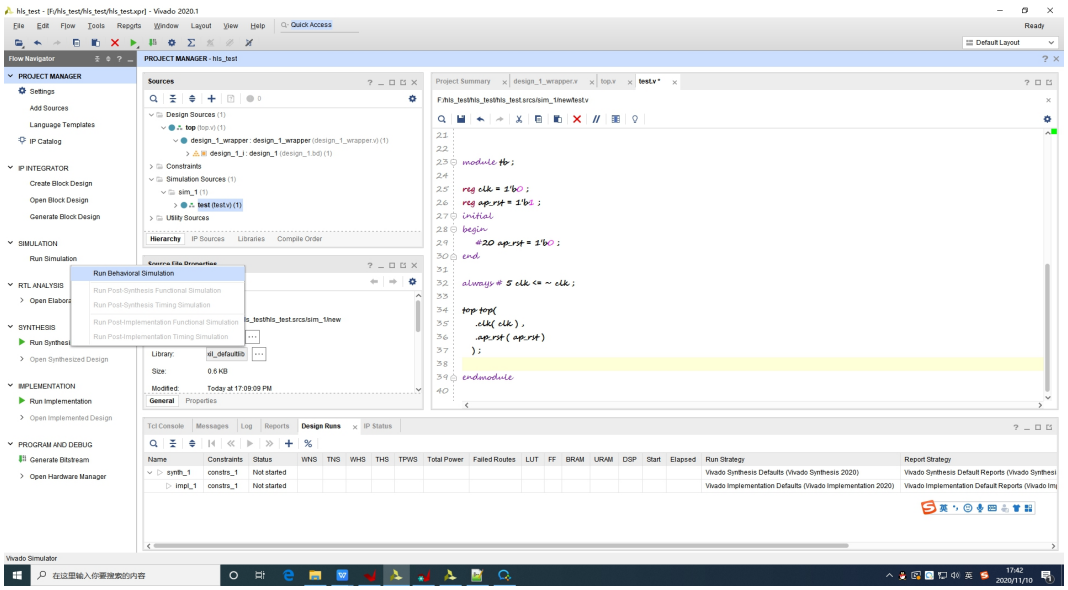
10)Create a new “test.v” simulation file



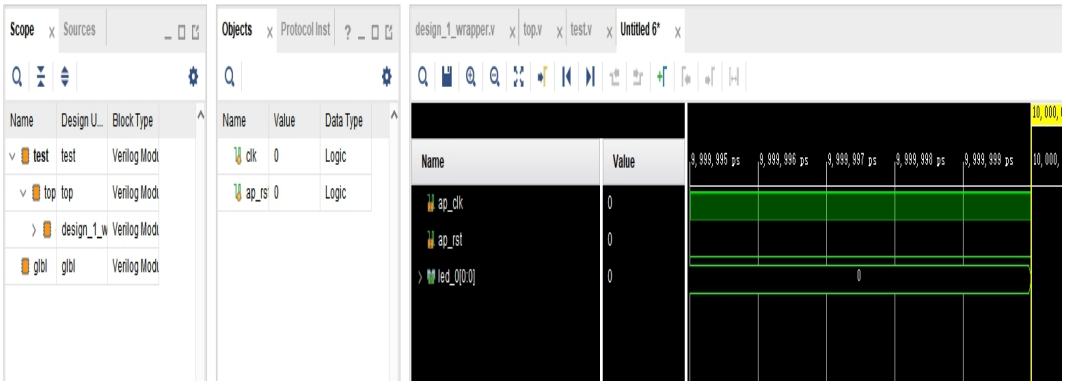
11)Enter the following code in test.v

```
module tb;

  reg clk = 1'b0 ;
  reg ap_rst = 1'b1 ;
initial
begin
    #20 ap_rst = 1'b0 ;
end

  always # 5 clk <= ~ clk ;

  top top(
     .clk( clk ) ,
     .ap_rst ( ap_rst )
    ) ;

endmodule
```

12)Select “simulation->run simulation”



13)You can see the corresponding signal waveform in the waveform diagram to determine whether it meets the expected operating effect

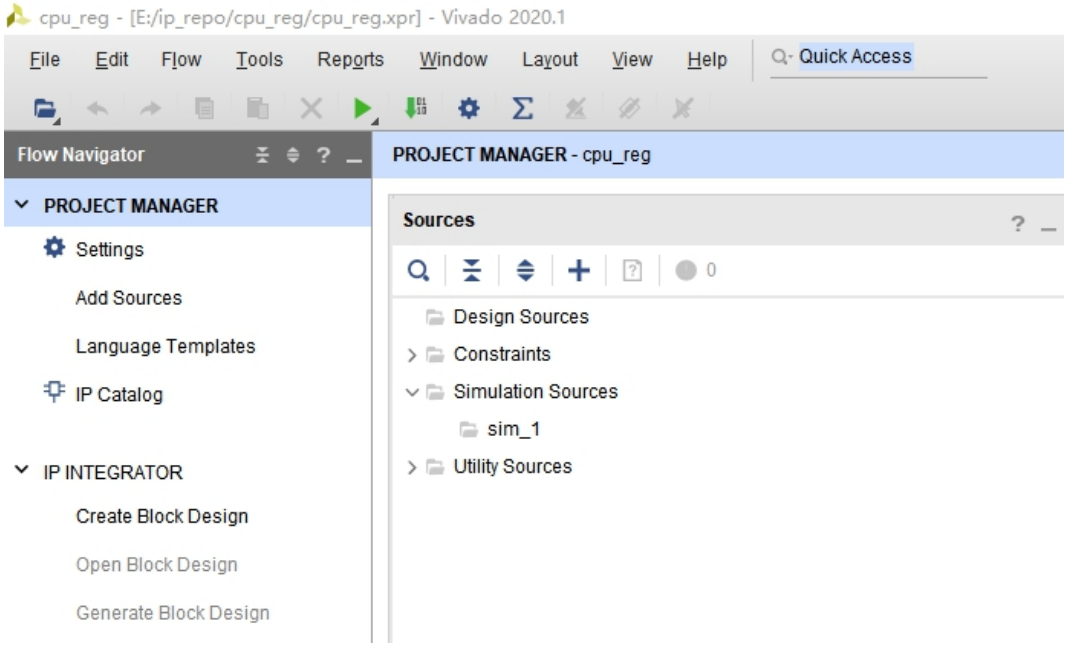


Note: During simulation, input the corresponding signal according to the corresponding interface of the corresponding HLS setting. The following chapters will introduce the interface type and setting method in detail.

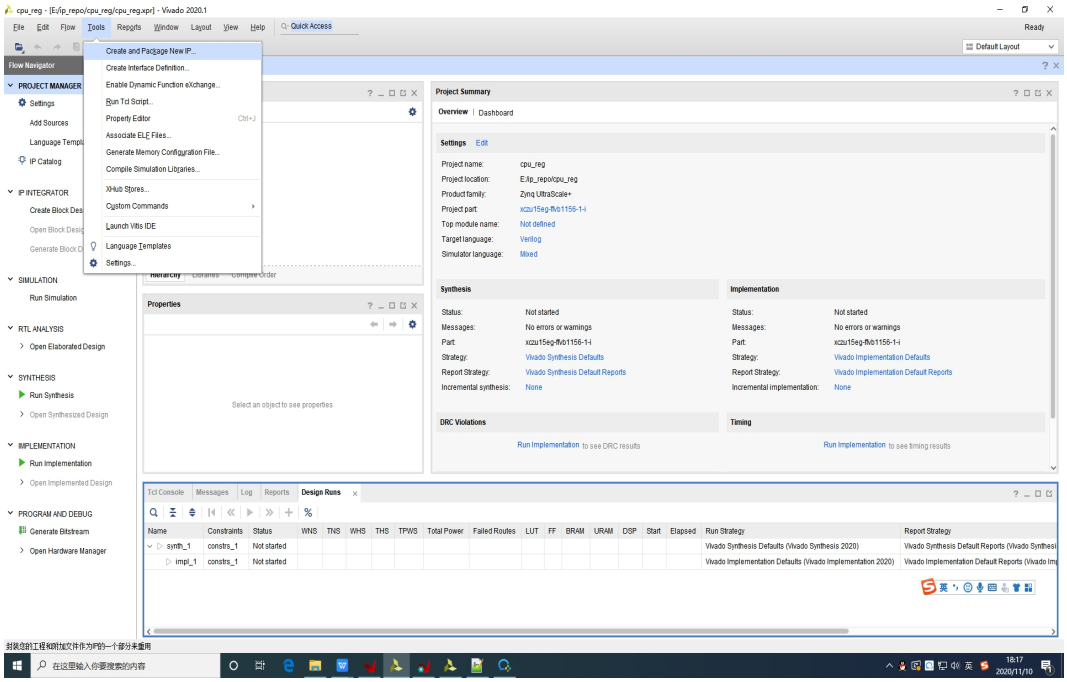
Part 1.2: Vivado Generates CPU Registers

The CPU register is a common way of interaction between PL and PS. PS and PL can exchange data through AXI bus. The main AXI bus interfaces include AXI_FULL type, AXI_LITE type, and AXI_STREAM type. Vivado provides us with a template for interface writing. You only need to use the template and add appropriate content in the user logic part of the module, and you can use the AXI bus interface to achieve your requirements. Under normal circumstances, the input data interface to be processed into the HLS and the processed output data interface will be set to the AXI type to facilitate interaction with the CPU. Here is an example of setting s_axilite to introduce how to use the template of CPU registers and the common methods of interacting with HLS.

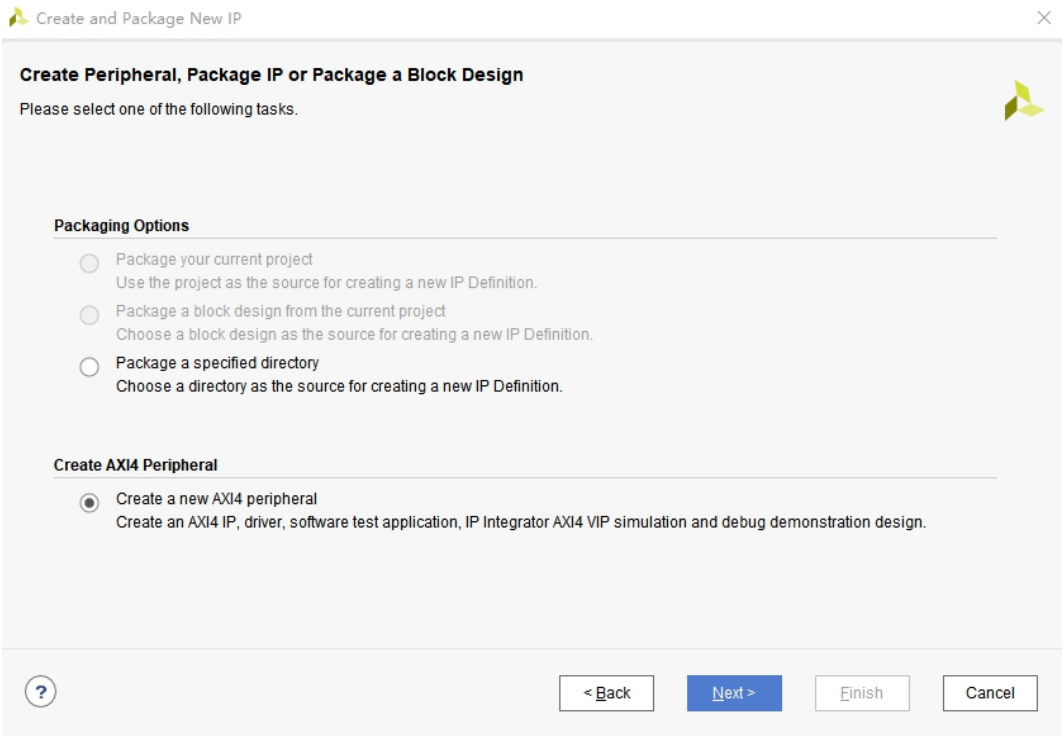
1) Create a blank vivado project and name it cpu_reg



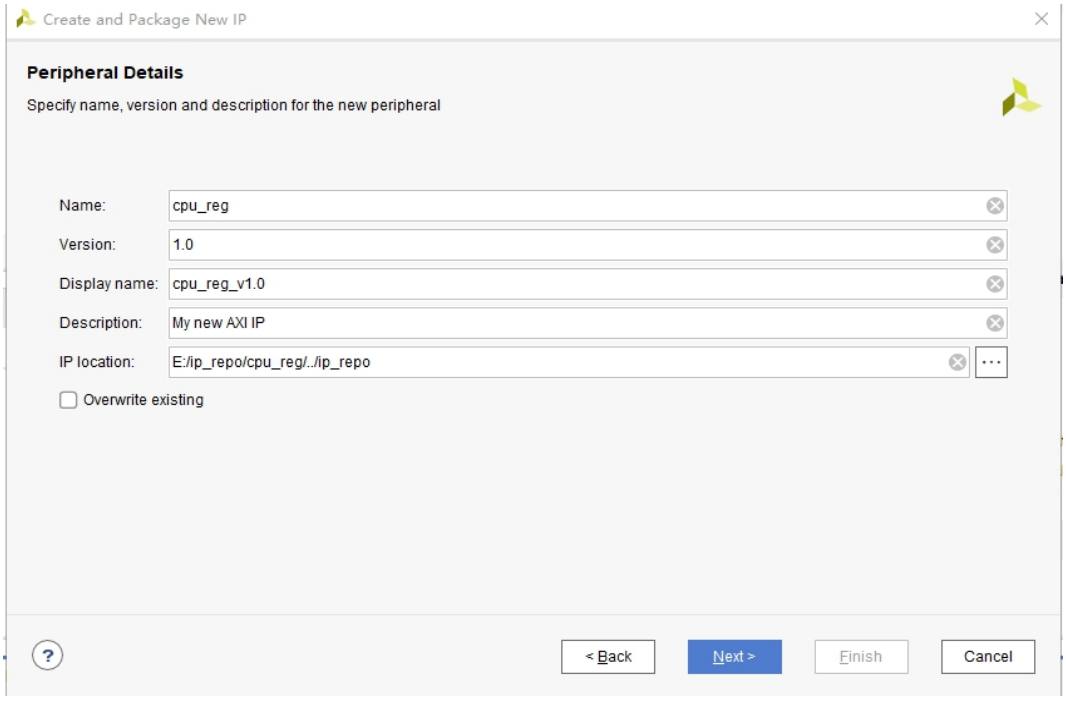
2) Select “tools->package and create new IP”



3) ”next->next“ until the following interface appears, select ”Create a new AXI4 peripheral”

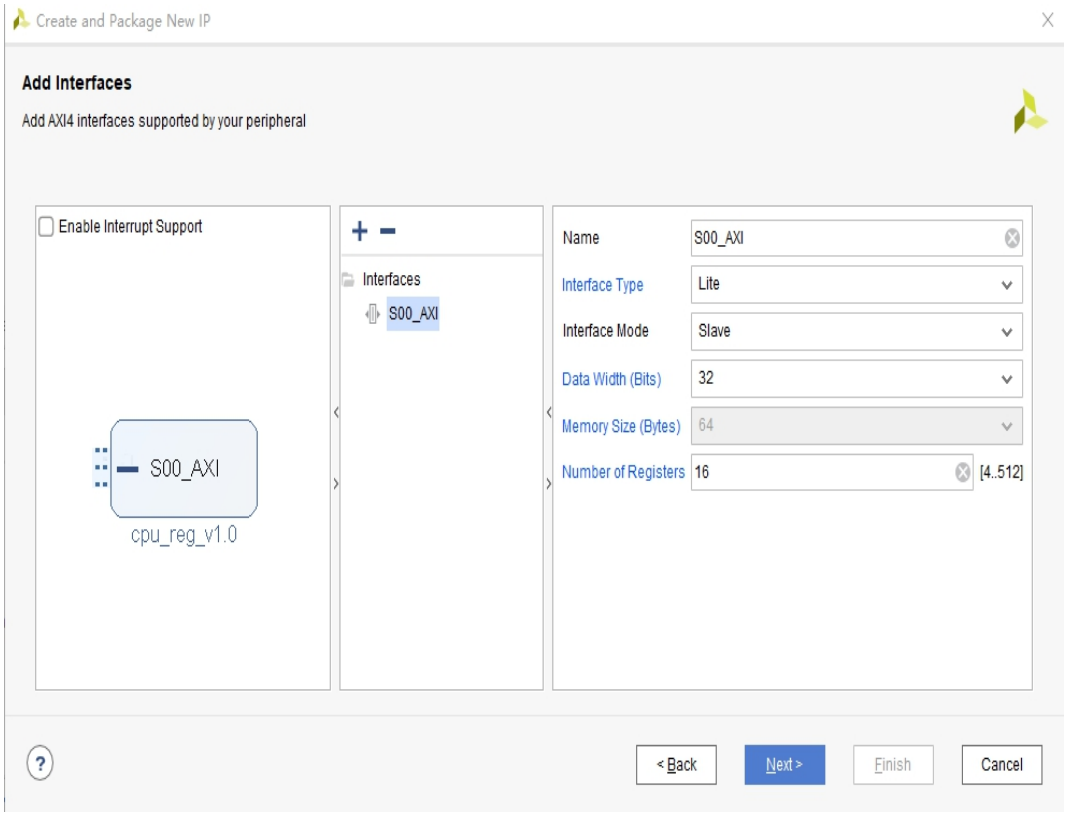


4) Next->Next, change the name to "cpu_reg"

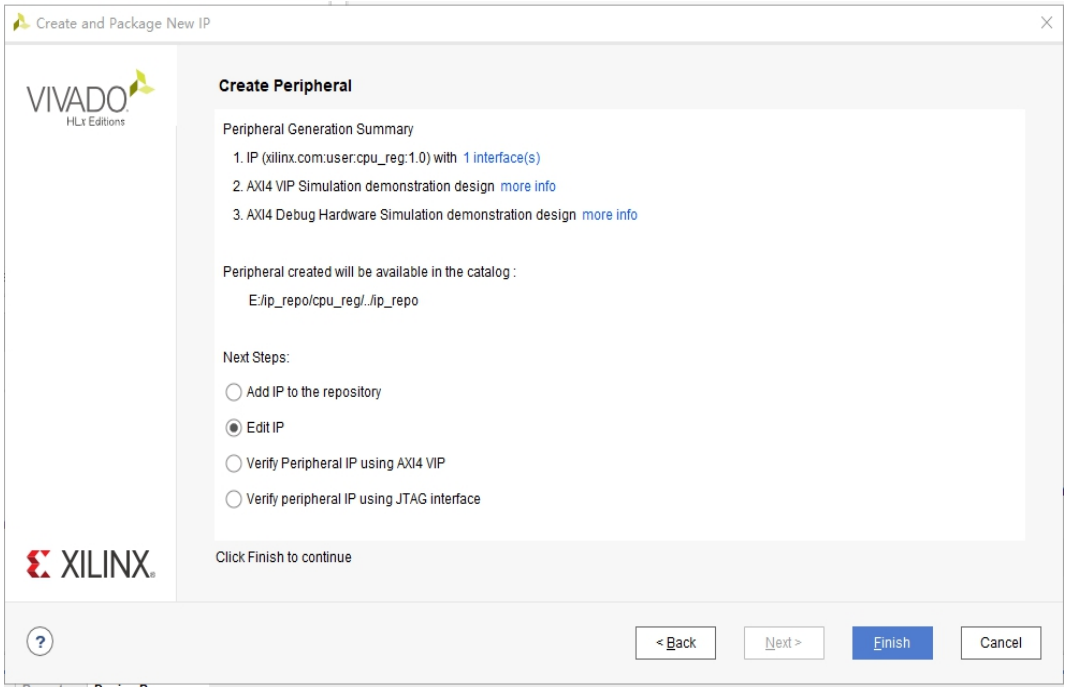


5) Next->Next, according to the following interface configuration, you can select the appropriate interface type in the "Interface Type" column, Divided into "stream", "full" and "lite". Interface Mode selects the master and slave, pay attention to the correct correspondence with the docking module In order to facilitate the

interaction with HLS, we choose "Slave lite"



6) Next->Next select "edit IP" and click "Finish"



7) Open "cpu_reg_v1_0_S00_AXI_inst" on the left, and make some modifications as shown in the figure

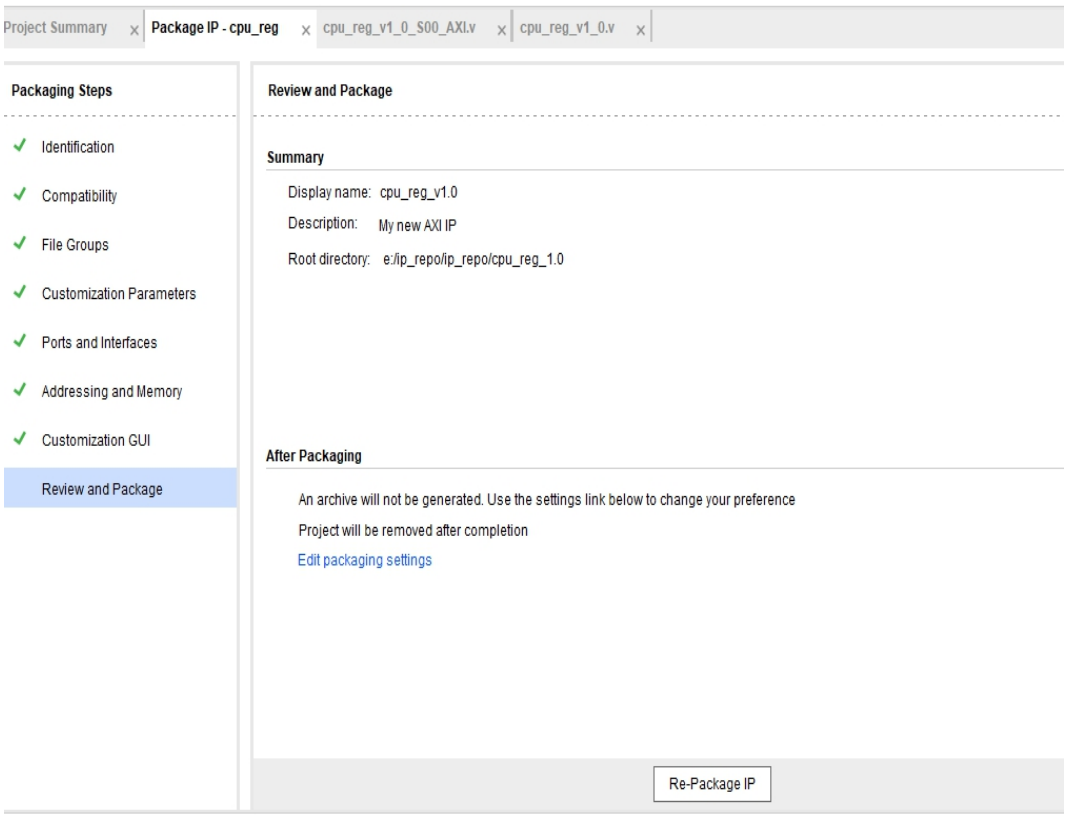
```
// Add user logic here

assign reg_out_0 = slv_reg0 ;
assign reg_out_1 = slv_reg1 ;
assign reg_out_2 = slv_reg2 ;
assign reg_out_3 = slv_reg3 ;
assign reg_out_4 = slv_reg4 ;
assign reg_out_5 = slv_reg5 ;
assign reg_out_6 = slv_reg6 ;
assign reg_out_7 = slv_reg7 ;
assign reg_out_8 = slv_reg8 ;
assign reg_out_9 = slv_reg9 ;
assign reg_out_10 = slv_reg10 ;
assign reg_out_11 = slv_reg11 ;
assign reg_out_12 = slv_reg12 ;
assign reg_out_13 = slv_reg13 ;
assign reg_out_14 = slv_reg14 ;
assign reg_out_15 = slv_reg15 ;

// User logic ends

// Users to add ports here

output wire [31:0] reg_out_0 ,
output wire [31:0] reg_out_1 ,
output wire [31:0] reg_out_2 ,
output wire [31:0] reg_out_3 ,
output wire [31:0] reg_out_4 ,
output wire [31:0] reg_out_5 ,
output wire [31:0] reg_out_6 ,
output wire [31:0] reg_out_7 ,
output wire [31:0] reg_out_8 ,
output wire [31:0] reg_out_9 ,
output wire [31:0] reg_out_10 ,
output wire [31:0] reg_out_11 ,
output wire [31:0] reg_out_12 ,
output wire [31:0] reg_out_13 ,
output wire [31:0] reg_out_14 ,
output wire [31:0] reg_out_15 ,

// User ports ends
```

8) Open “cpu_reg_v1_0” on the left, and make some modifications as shown in the figure

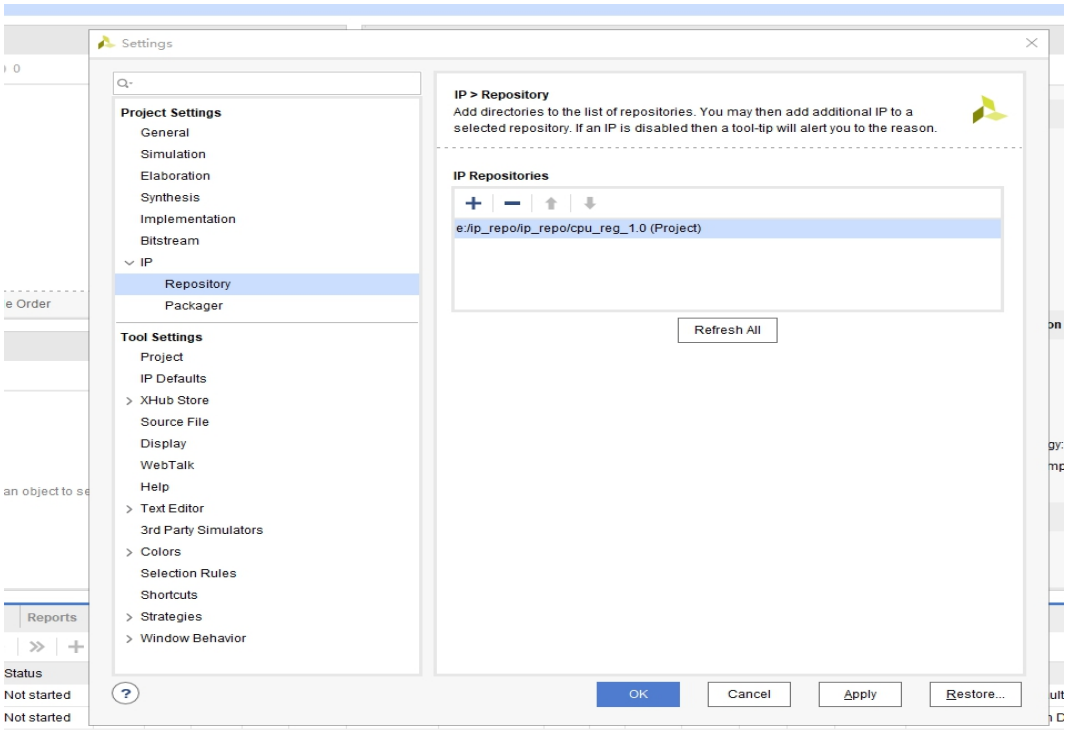
```
// Users to add ports here
output wire [31:0] cpu_reg_0 ,
output wire [31:0] cpu_reg_1 ,
output wire [31:0] cpu_reg_2 ,
output wire [31:0] cpu_reg_3 ,
output wire [31:0] cpu_reg_4 ,
output wire [31:0] cpu_reg_5 ,
output wire [31:0] cpu_reg_6 ,
output wire [31:0] cpu_reg_7 ,
output wire [31:0] cpu_reg_8 ,
output wire [31:0] cpu_reg_9 ,
output wire [31:0] cpu_reg_10 ,
output wire [31:0] cpu_reg_11 ,
output wire [31:0] cpu_reg_12 ,
output wire [31:0] cpu_reg_13 ,
output wire [31:0] cpu_reg_14 ,
output wire [31:0] cpu_reg_15 ,
// User ports ends
```

```
.cpu_reg_0 ( cpu_reg_0 ) ,
.cpu_reg_1 ( cpu_reg_1 ) ,
.cpu_reg_2 ( cpu_reg_2 ) ,
.cpu_reg_3 ( cpu_reg_3 ) ,
.cpu_reg_4 ( cpu_reg_4 ) ,
.cpu_reg_5 ( cpu_reg_5 ) ,
.cpu_reg_6 ( cpu_reg_6 ) ,
.cpu_reg_7 ( cpu_reg_7 ) ,
.cpu_reg_8 ( cpu_reg_8 ) ,
.cpu_reg_9 ( cpu_reg_9 ) ,
.cpu_reg_10 ( cpu_reg_10 ) ,
.cpu_reg_11 ( cpu_reg_11 ) ,
.cpu_reg_12 ( cpu_reg_12 ) ,
.cpu_reg_13 ( cpu_reg_13 ) ,
.cpu_reg_14 ( cpu_reg_14 ) ,
.cpu_reg_15 ( cpu_reg_15 ) ,
 .S_AXI_ACLK(s00_axi_aclk),
 .S_AXI_ARESETN(s00_axi_aresetn),
```

9) Select “review and package”, click “Re-Package IP”



10)Click Settings "->IP->repository" in the project



At this point our CPU register has been successfully generated

Part 1.3: HLS Interface Synthesis

The HLS interface is mainly divided into three categories: clock and reset interface, Block Level interface, Port Level Interface. The main things we can integrate are Block Level Interface and Port Level Interface

Block Level Interface:

Port Level Interface :

The interface is related to the topfunction parameter, and is mainly divided into four types: AXI4 Interface, No I/O Interface, wire handshake, and memory Interface.

AXI4 Interface :

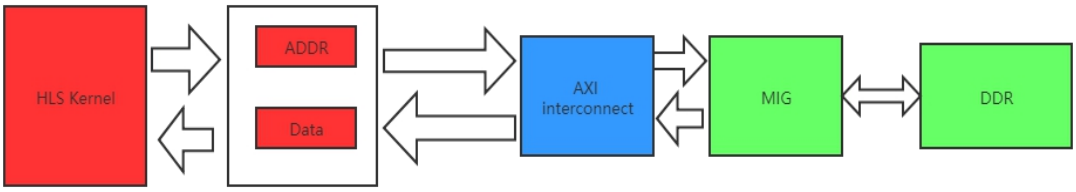
Divided into three types: axi, m_axi, s_axilite, based on the AXI bus protocol, used for data interaction between the PS side and the PL side.

1) **AXI:**

Refers to the axi-stream interface, which includes two types of side-channel and without side-channel. It is usually used to connect data continuously input from the outside world. For example, if we want to send several sets of data from the PS end to the PL end, we can input the IP core Set to axi type interface

2) **m_axi:**

Refers to the axi-full interface in master mode. HLS is used as the master mode to read and write data from the PS side. It is mostly used to send data to the PS side through the AXI-HP interface, and then to the DDR or read from the DDR. , The main reading and writing methods and connection forms are shown in the figure:



Note that if the interface is set to m_axi mode, it will involve a read and write delay problem, which is about 30 cycles overall, which involves an optimization called burst transfer. Using a suitable method of reading and writing DDR can improve the overall speed of data processing to a certain extent

3) s_axilite:

Refers to the axi-lite interface, which is a lightweight axi interface, usually used for data interaction with the PS-side CPU, and is mostly connected to the AXI-GP interface for sending configuration information provided by the CPU or writing configuration information to the CPU .

NO I/O Interface

Divided into two types: ap_none and ap_stable

1) ap_none:

Multiple connections to a single data, without any other related signals, belong to the default parameter interface

2) ap_stable:

Used to configure input parameters, parameter configuration usually only changes in the reset state, not much use

WIRE HANDSHAKE

Divided into three types: ap_vld, ap_ovld, ap_ack

1) ap_vld:

Input signal, usually connected to a single data, including valid signal and input signal

2) ap_ovld:

Output signal, usually connected to a single data, including valid signal and output signal

3) ap_ack:

Contains only valid signals, does not contain data, basically not used

Memory Interface

Contains ap_bram, ap_memory, and ap_fifo

1) ap_bram:

When the input data type is an array, using this type of interface, you can directly connect with Bram

2) ap_memory:

The function is the same as ap_bram, except that ap_bram will pack the relevant signals together when synthesizing, and ap_memory will synthesize into multiple scattered interfaces

3) ap_fifo:

Use this type of interface when the input data type is an array, which is used to connect directly to fifo

Part 1.4: HLS Common Optimization

Divided into four ways to optimize: Throughput, Area, Latency, logic

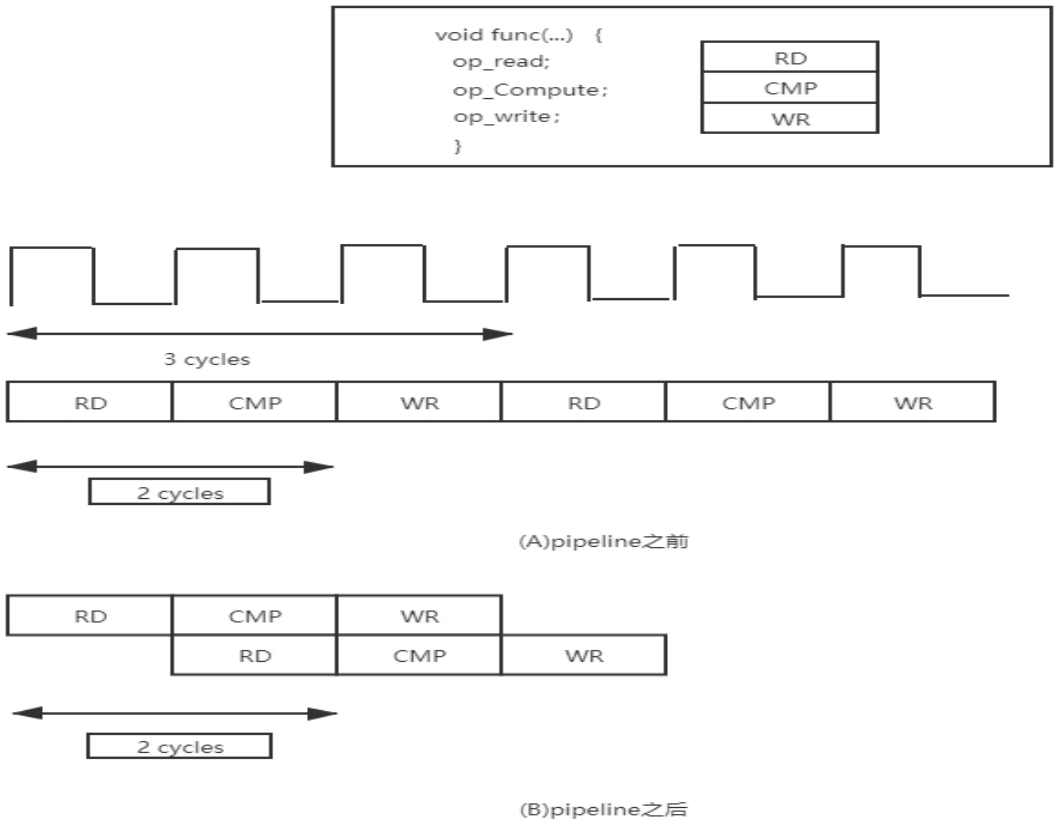
Throughput:

Throughput optimization is mainly divided into the following types

1) Function and Loop PipeLine:

Function or loop pipeline hydration can reduce the corresponding execution interval. After the function is pipelined, the loop in the function body will automatically expand to improve the running speed. After the loop is streamlined, the loop count counter does not need to wait for all the statements in the loop body to be executed before

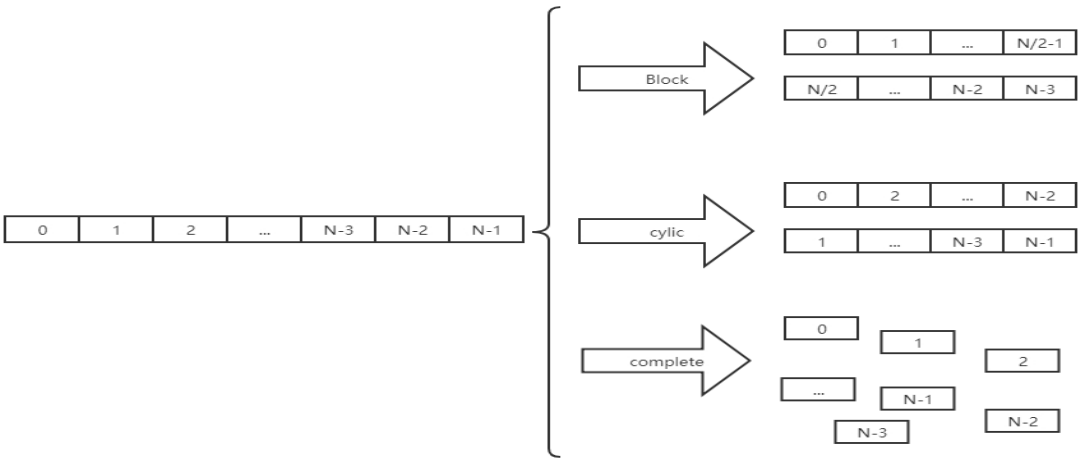
increasing. To increase the circulation of data, take the following figure as an example:



It can be seen that the execution time after pipeline optimization is shortened

2) Partitoning Arrays:

Partitoning Arrays refers to the disassembly of a large array into several small arrays. There are three modes of block, cylic, and complete. The three disassembly methods are shown in the figure:



In HLS, the array can be integrated into two forms of BRAM and FIFO. Both of these devices cannot provide more than two interfaces for data reading and writing. If the array is disassembled, multiple BRAMs can be used to represent the original large array. More ports are available for data reading, which has a great impact on the optimization methods of pipeline and unroll.

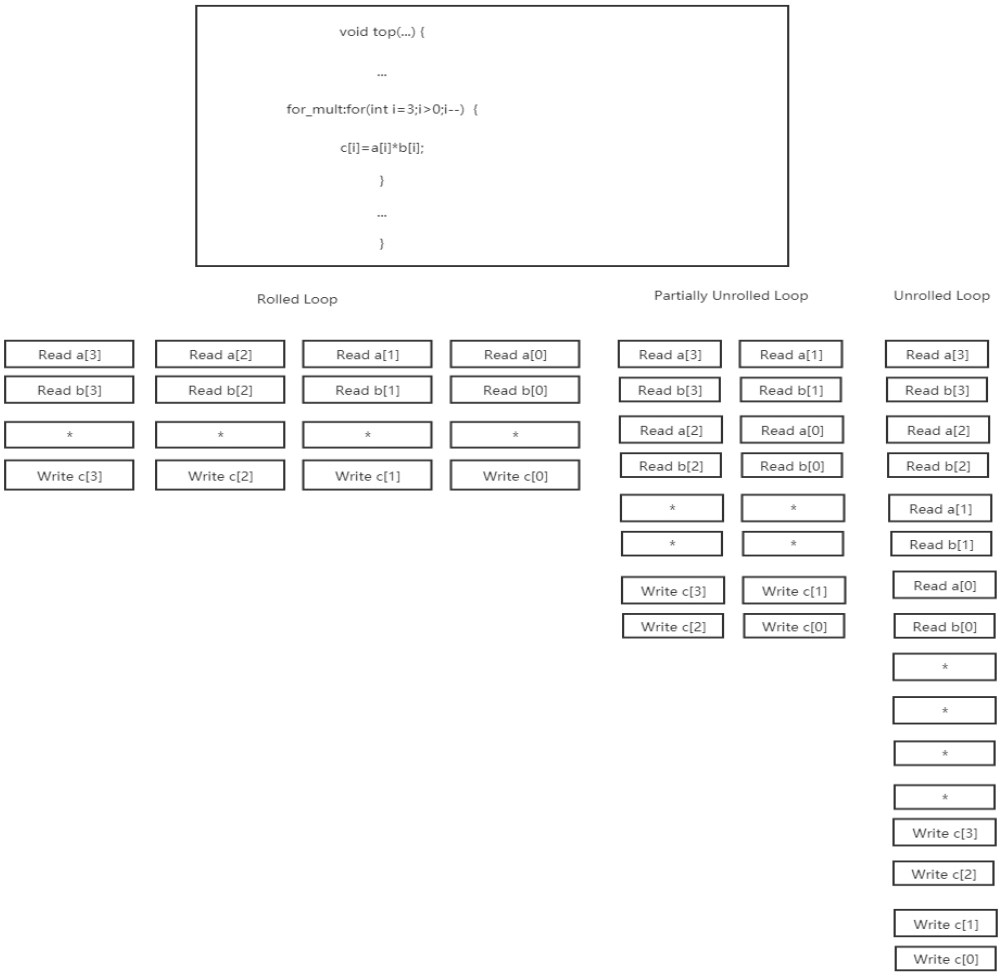
3) Remove False Dependency:

Statement execution in HLS has two modes: sequential execution and parallel execution. When HLS considers that there is no dependency between two sentences, HLS will choose to execute two statements at the same time to increase the speed of operation. This increase in speed sometimes has a very large impact on the overall execution speed. There are four situations between HLS statements, RAW (read and then write), WAR (write and then read), WAW (write and then write), RAR (read and then read). Among them, only in the case of RAR can it be considered that there is no dependency between the two statements at this time and can be executed at the same time. Note that if there is no RAR between the two sentences, forcibly setting "Dependency=false" will cause errors during synthesis or HLS will use a very radical way to meet the requirements, which will have a great impact on the function, so this optimization method must

Use it with caution

4) UNROLLING:

Loop unrolling can reduce the delay of the overall loop. When processing more data in one cycle, the impact will be greater area consumption. The impact of loop unrolling is shown in the following figure:



It can be seen that it takes four cycles to consume one multiplier before expansion, two cycles after partial expansion, two multipliers, and one cycle four multipliers after full expansion. At the same time, loop unrolling must ensure that the corresponding data can be read from bram at the same time As can be seen in the figure, partial expansion needs to read two numbers from bram at the same time, which can be achieved by specifying the array resource as

dual-port-Bram Full expansion requires four numbers to be read from bram at the same time. However, there is no bram with four ports. Therefore, the array must be completely disassembled and turned into a trigger to achieve full expansion. This function consumes a lot of area, so be careful use.

5) Dataflow:

When HLS executes two dependent functions, it will execute the next function after all the statements of the previous function are executed. At this time, The total execution time is the sum of the execution time of the two functions. After DATAFLOW optimization, the total execution time is the longest execution time of the two functions, which greatly reduces the running time. You can think of this optimization method as a pipeline between functions. In many video processing functions, this optimization method is usually used to ensure a better display effect if the resolution is high and the number of frames is high. There are several limitations to using this optimization method:

1) Function parameters should be set to stream type, according to fifo integrated array
2) The output of the previous function can only be used as input to one function, and cannot be used as input to more than two functions at the same time
3) If multiple functions are executed in sequence, the output of the previous function cannot bypass the intermediate function and directly input to the subsequent function.
4) Function parameters cannot be fed back, that is, the function output can only be input to the function after it, not the function before it
5) Function conditional execution, if an intermediate function can only

be executed under certain conditions, DATAFLOW cannot be used for optimization at this time

6) The final function has multiple outputs under multiple conditions, and DATAFLOW cannot be optimized at this time

For the first item, pay special attention. If the array is integrated into a stream, setting the depth to be too small will cause the entire function to appear halt. At this time, the new data cannot be input, and the old data has not been read out, causing the entire IP core to stop running. The entire project may be completely terminated.

Latency

Delay optimization mainly includes the following two methods

1) Flatten:

If a loop is nested within a loop, there will be a clock delay from the outer loop to the inner loop. At this time, Flatten optimization can remove this delay and increase the running speed.

2) Merge:

Usually between two loops, the next loop can only be executed after the previous loop is completed. At this time, if the two loops are integrated into one loop, the overall execution delay can only be reduced.

Area

Area optimization includes the following methods

1) DATA_Types and width:

The data bit width in HLS can be flexibly defined. It needs to be defined according to the required bit width. If the bit width is too large, it will increase resource consumption. If the definition is too small, data overflow is likely to occur. At the same time, when the data calculation result exceeds 8, 16, or 32 bits, Pay attention to avoid the situation of Integer promotion. For example, if two 18-bit numbers are multiplied,

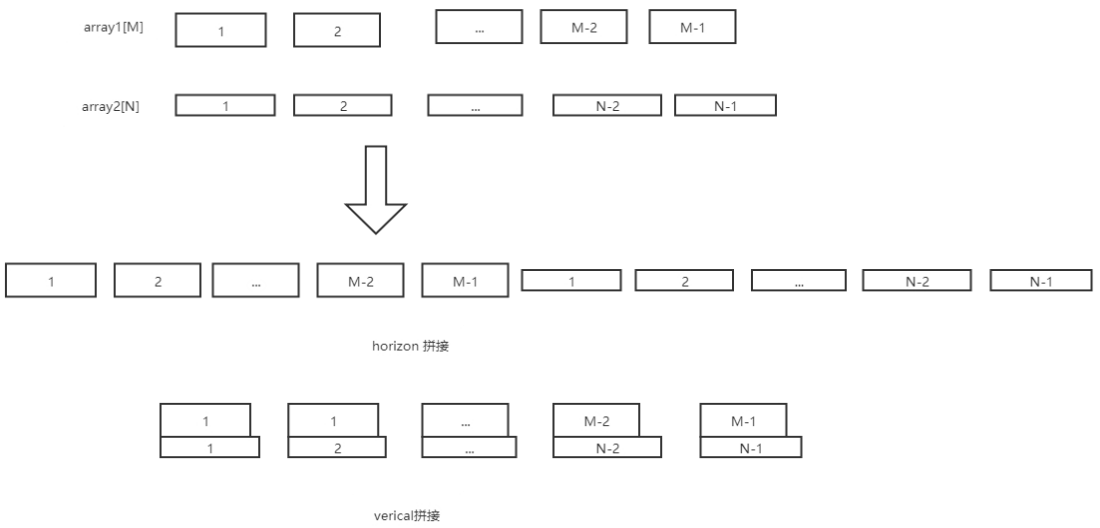
the product is defined as 36 bits, and 32 bits are exceeded. At this time, the compiler will convert two 18-bit numbers into 32-bit numbers, and then multiply two 32-bit numbers. Generate a 32-bit product, then convert the 32-bit product into 36 bits and send it to the previously defined 36-bit product. At this time, the calculated result is not correct.

2) Function Inlining:

Function inlining has a great impact on resource consumption. Function inlining refers to the elimination of function levels. When a function is called by multiple functions and the called function is at the same level, this function can be reused. We can By modifying the function level to achieve the effect of function reuse, There are three ways of inlining. Set inline directly to the function. This kind of inline is inline on top, which means that the function that calls it is told that the function has been considered inline. At this time, the function that is set to inline cannot It is reused, how many times the resource consumption will be generated as many times as it is called, and it is recommended to set inline = off for functions called multiple times This type of region inline inline is the next inline, that is, when this function calls the sub-function, the sub-function is regarded as inline, and it does not need to go through a clock to jump to the sub-function. Note that region inline can only act on the level of it. Lower-level sub-functions, functions called within sub-functions are not considered inline Recursive inline This kind of inlining is for all functions called by it, and the sub-functions of the called function are also regarded as inline, that is, there is no lower-level function under the function at this time, sometimes for the inline method Improper setting may result in larger area consumption, so appropriate inline type setting should be made according to the specific situation

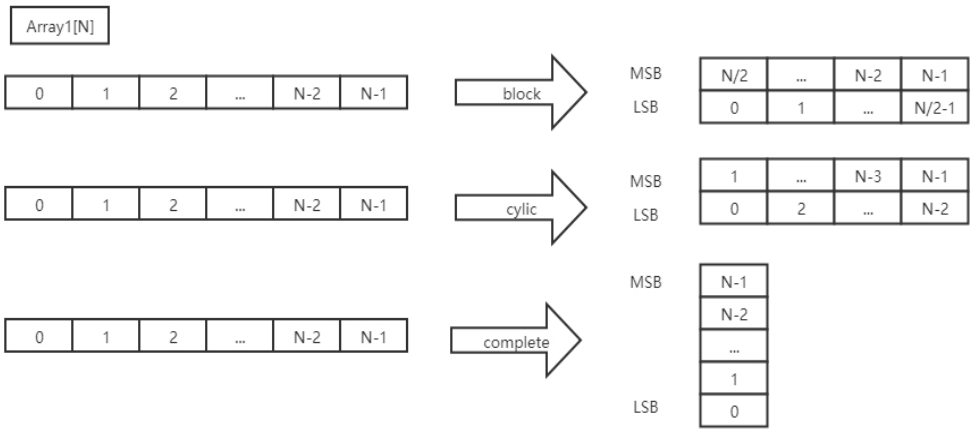
3) Map many arrays to one large array:

Combine multiple small arrays into a larger array. Since each array will be integrated into bram or fifo, too many small arrays will increase the resource consumption of bram. At this point, we can try to combine small arrays into larger arrays to reduce bram consumption. There are two splicing modes: horizon and vertical modes After horizon splicing, the size of the array is the sum of the spliced small arrays. The vertical mode is to increase the data bit width, and the size is the original larger array. The effect is shown in the following figure:



4) Array Reshape:

Array reshape, the effect is similar to array Partition, using veritcal to splice arrays to achieve the effect of parallel output of the array, the array reshape method is shown in the figure:



5) Controlling hardware resources:

Control logic resources, you can request HLS to reuse the resources by specifying the total number of logic resources used, and specify which logic resources to use for a certain operation to implement the corresponding operation. This can directly reduce resource consumption and help the optimization of the area.

LOGIC

Logic optimization is achieved in the following two ways

1) Controlling Operator Pipelining :

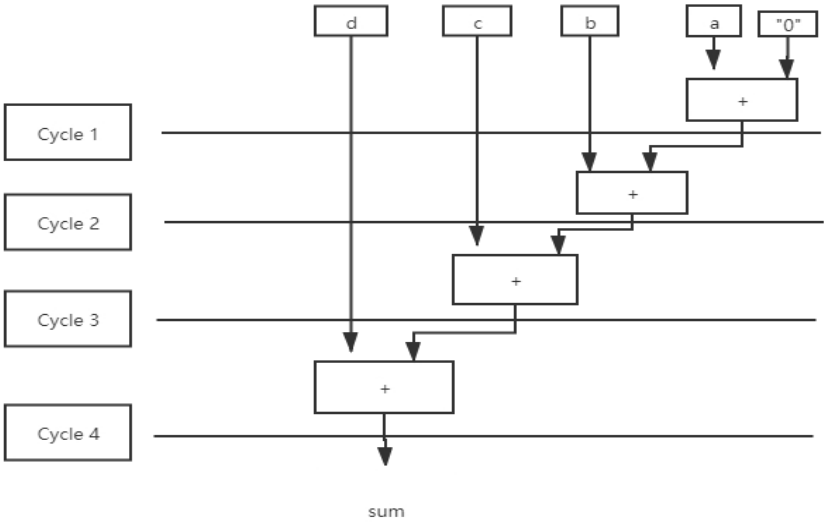
HLS automatically controls the operation pipeline level, sets the latency to adjust the operation delay, improves the timing by sacrificing time lag, and optimizes the logic

2) Optimizing Logic Expression:

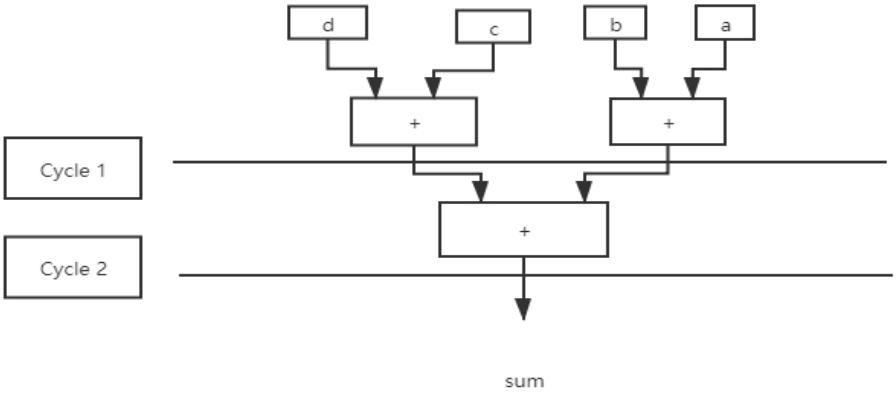
Expression balance, in the default state, it is turned on for fixed-point calculations and turned off for floating-point calculations. It can be adjusted manually. Note that the area consumption will increase when Expression balance is turned on. For example, the following code

```
    data_t foo_top (data_t a, data_t b,
data_t c, data_t d)
    {
    data_t sum;
    sum = 0;
    sum += a;
    sum += b;
    sum += c;
    sum += d;
```

The comprehensive method before setting "Expression balance=on" is as follows:

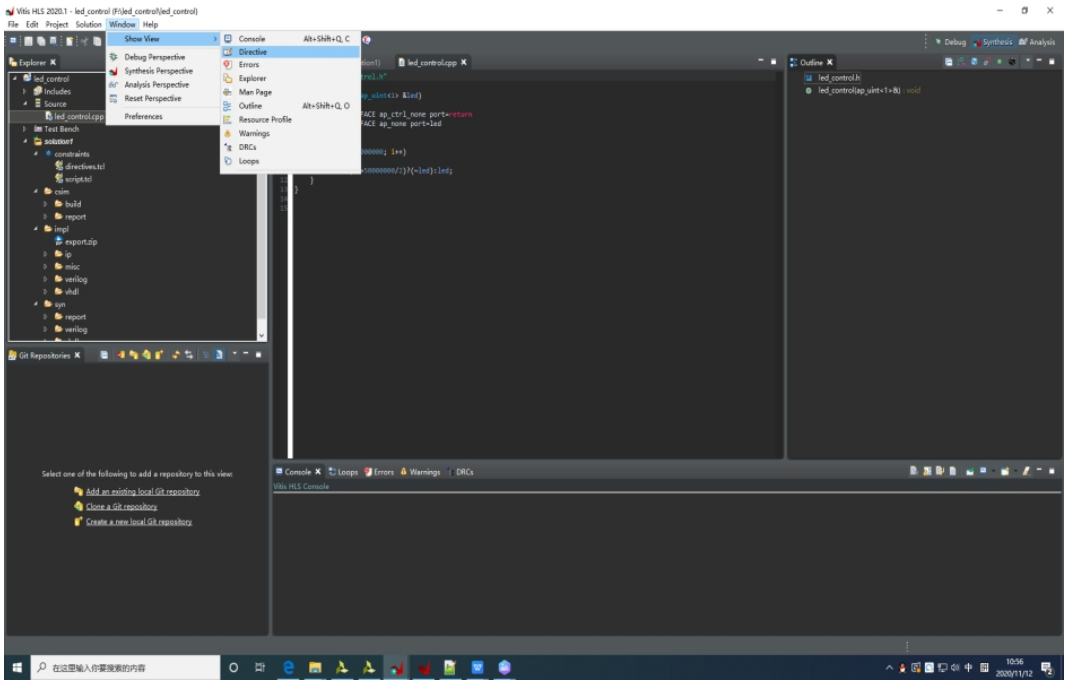


The comprehensive way after setting "expression balance=on" is as follows:

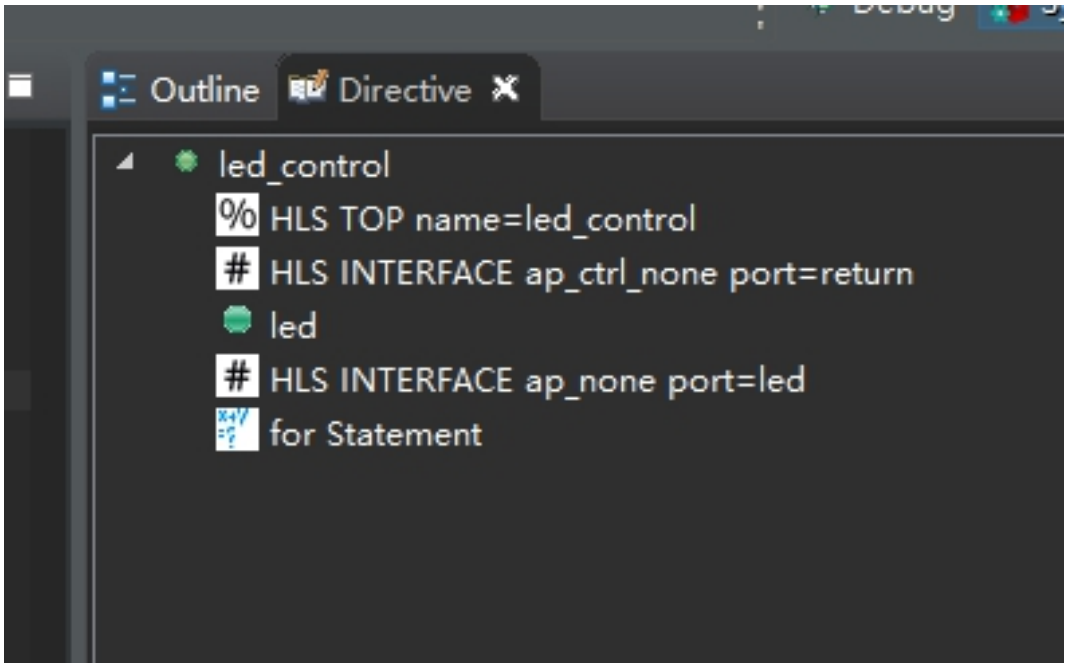


How to do interface synthesis and optimization operations:

Open window->show view->directive



Double-click the corresponding signal in the directive to perform interface synthesis or optimization



Part 1.5: HLS include Library

Arbitrary Precision Data Type	Integer and fixed point (ap_cint.h, ap_int.h and systemc.h)
HLS Flow	Stream data structure model

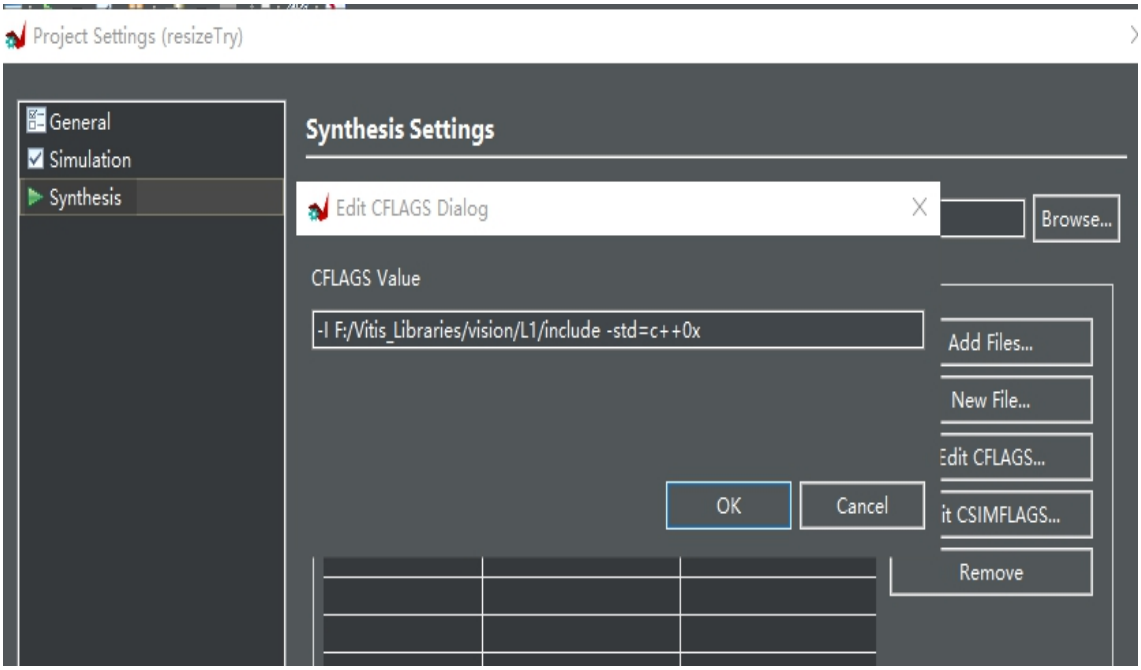
	Designed for optimal performance and area (hls_stream.h)
HLS Math	The synthesis of the standard C (math.h) and C++ (cmath.h) math libraries is widely supported. Support for floating point and fixed point functions: abs, atan, atanf, atan2, atan2, ceil, ceilf, copysign, copysignf, cos, cosf, coshf, expf, fabs, fabsf, floorf, fmax, fmin, logf, fpclassify, isfinite, isinf, isnan, isnormal, log, log10, modf, modff, recip, recipf, round, rsqrt, rsqrtf, 1/sqrt, signbit, sin, sincos, sincosf, sinf, sinhf, sqrt, tan, tanf, trunc
HLS IP	Integrated LogiCORE IP FFT and FIR Compiler (hls_fft.h, hls_fir.h, ap_shift_reg.h)
HLS Linear algebra	cholesky, cholesky_inverse, matrix_multiply, qrf, qr_inverse, svd (hls_linear_algebra.h)
HLS DSP	atan2, awgn, cmpy, convolution_encoder, nco, qam_demod, qam_mod, sqrt, viterbi_decoder (hls_dsp.h)

Note: The video processing function of Vitis HLS uses the function of xfopencv, which can be downloaded on Xilinx official github https://github.com/Xilinx/Vitis_Libraries.

The image processing function of xfopencv is in the include folder of L1. After the download is complete, replace the xf_infra.hpp file with the file with the same name provided in the tutorial

Set the file path before using the xfopencv library file

Click the icon  to select “synthesis->edit cflags”, and enter: “-I <include-path> -std=c++0x” as shown in the figure:



Among them, F:/Vitis_Libraries/vision/L1/include is the path before downloading. The corresponding path should be replaced according to the downloaded address. The hls_video function and hls_opencv function of the previous version have been abandoned in the vivado HLS 2020.1 and vitis HLS 2020.1 versions. It is to pay special attention.

Part 1.6: HLS Official Tutorial

1) *Introduction to FPGA Design with Vivado High-Level Synthesis (UG998)*
2) *Vivado Design Suite Tutorial: High-Level Synthesis (UG871)*
3) *Vivado Design Suite User Guide: High-Level Synthesis (UG902)*
4) *Xilinx Opencv User Guide (UG1233)*
5) *Vitis HLS Migration Guide (UG1391)*

Part 2: Getting Started

Part 2.1: Introduction to the Experiment

LED light flashing is a relatively simple program. Through HLS, we can realize LED light flashing and observe the phenomenon. During this period, we should pay attention to that sometimes the code we write in HLS synthesis may be different from what we expected. It cannot be simply written as C++. For people studying HLS, it is best to have an initial understanding of the working methods and principles of FPGA, and then combine the knowledge of C++ to write HLS code to realize the development of HLS

Part 2.2: Experimental Source Code

```
#include <ap_int.h>

void led_control(ap_uint<1> &led)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=led
unsigned int i;
  led=0;
  for(i=0; i<50000000; i++)
  {
#pragma HLS PIPELINE
       led = (i<50000000/2)?(~led):led;
  }
}
```

Part 2.3: Interface Settings and Optimization

Interface Setting

Here Block level Interface is set to ap_none type to remove redundant control signals, and the parameter is set to ap_none type

Comprehensive Optimization:

Use Pipeline optimization inside the loop to reduce the delay and increase the running speed.

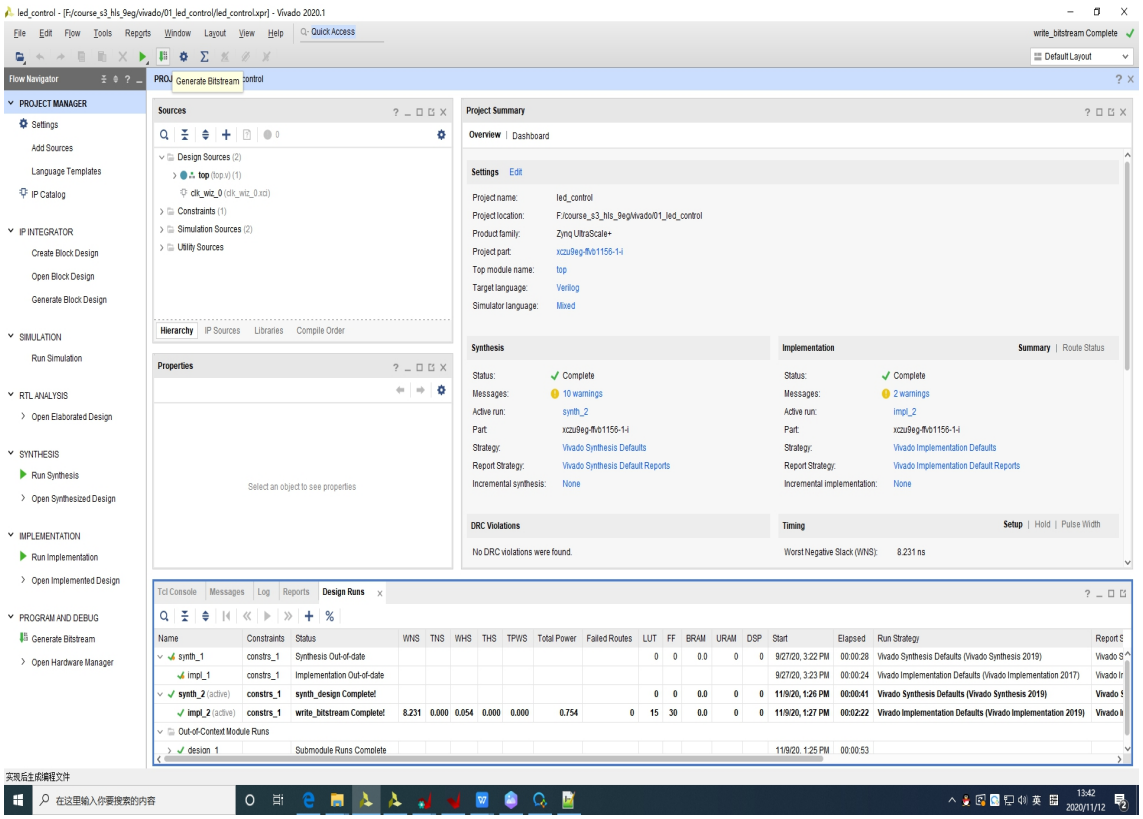
Part 2.4: Project Path

Name	Path
Vivado Project	vivado/led_control
HLS Project	hls/led_control
bit File	/vivado/led_control/led_control.runs/impl_1/top.bit

Part 2.5: Project Realization

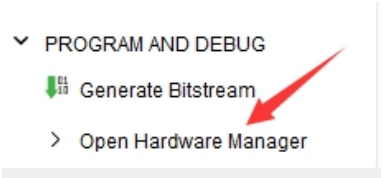
Let's focus on the implementation of the project. The clock required for this entry-level project directly uses the clock on the FPGA development board. According to the clock type on the FPGA development board, the corresponding clock primitive IP core is used to drive our HLS core to achieve functions.

First, we follow the steps in the previous chapter to add the IP core to the vivado project, generate the system wrapper, and create the top-level file. Then click generate bitstream to generate the bitstream. This process may take a few minutes, please be patient, if it is not completed for more than an hour, it is recommended to change to other computer

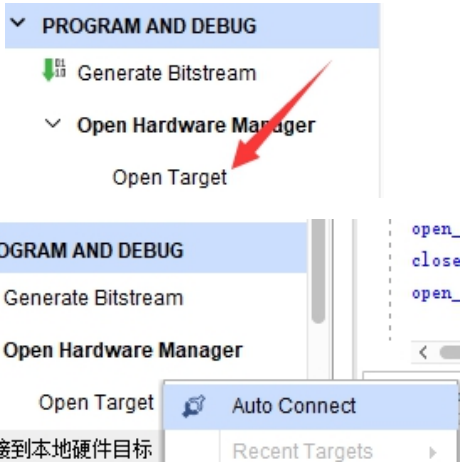


Next, we can load the newly generated bit file to the FPGA Development board, first power on the board, and make sure to connect the JTAG.

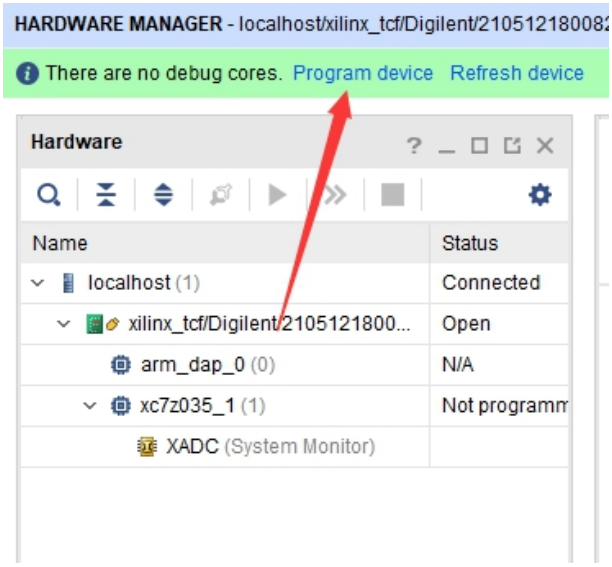
1) Click “Open Hardware Manager”



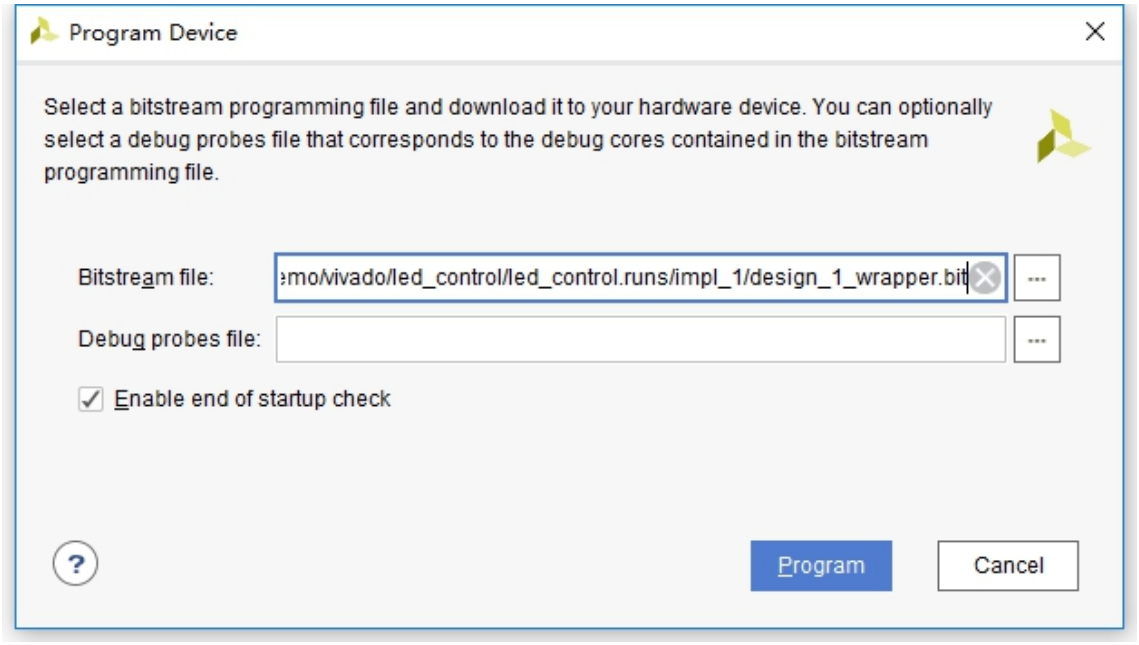
2) Click “Open Target”→“Auto Connect”



3) Click “Program device”



4) Click “Program”



5) After the completion, you can see the LED lights flashing at a frequency of 1 second, so far, the entire experiment is completed.

Part 3: HLS Interacts with CPU Registers

Part 3.1: Introduction to the Experiment

The experimental result in this chapter is similar to the previous chapter. The difference is that in this chapter we use the CPU register to control the flashing time of the light. In addition to using Vivado, we also need to use Vitis to assist. The clock is generated by the PS terminal, and a program must be run on the PS terminal. The circuit clock can be generated only after the PS terminal runs.

Part 3.2: Experimental Source Code

```
#include <ap_int.h>

void led_register(ap_int<1> &led, int total_cnt, int high_cnt)
{
#pragma HLS INTERFACE ap_none port=led
#pragma HLS INTERFACE ap_none port=total_cnt
#pragma HLS INTERFACE ap_none port=high_cnt
#pragma HLS INTERFACE ap_ctrl_none port=return
  led=0;
  unsigned int i=0;
  for(i=0;i<total_cnt;i++)
  {
#pragma HLS LOOP_TRIPCOUNT max=50000000 min=50000000
       led = (i==high_cnt)?(~led):led;
  }
}
```

Part 3.3: Interface Settings and Optimization

Interface Setting

The Block level Interface is set to ap_ctrl_none type, because there is no need for additional control signals. In other experiments, all input parameters of Block level Interface are set to ap_none in this way to remove redundant associated signals.

Comprehensive Optimization:

Because the number of loops is a variable, TRIPCOUNT needs to be set otherwise accurate running time and delay estimates cannot be obtained. This type of optimization is required most of the time when the number of loops is a variable.

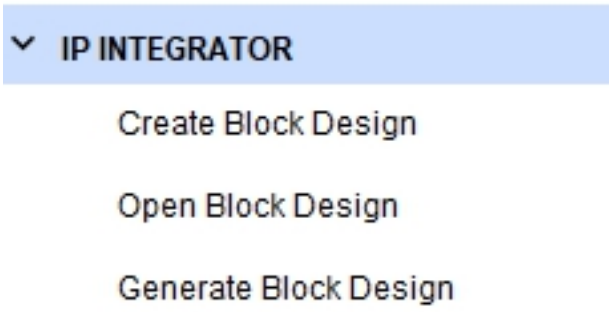
Part 3.4: Project Path

Name	Path
Vivado Project	vivado/led_control
HLS Project	hls/led_control
Bit File	/vivado/led_control/led_control.runs/impl_1/design_1_wrapper.bit

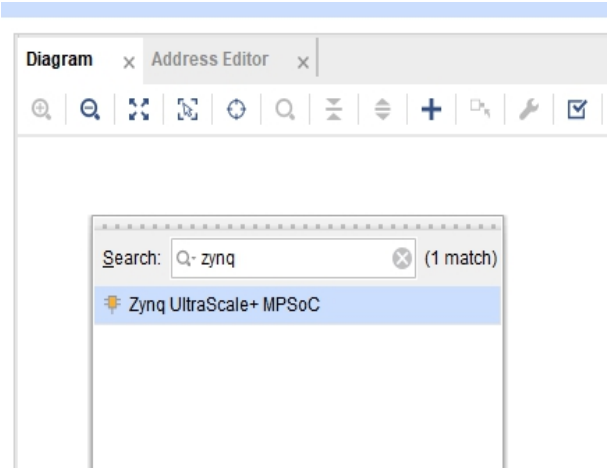
Part 3.5: Project Realization

Let's focus on the project realization, and the project realization should follow the steps below

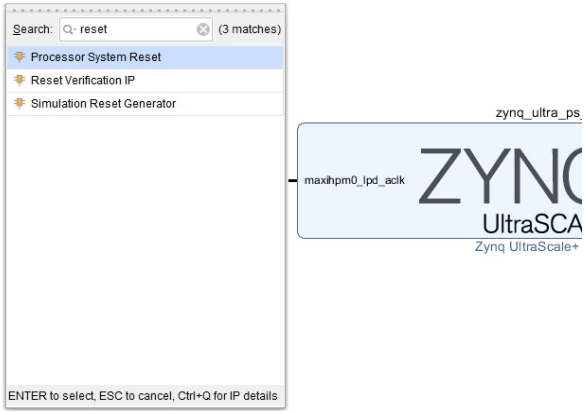
1) Follow the steps described in the previous chapter to add the IP core to the project
2) Select “Create Block Design” on the left



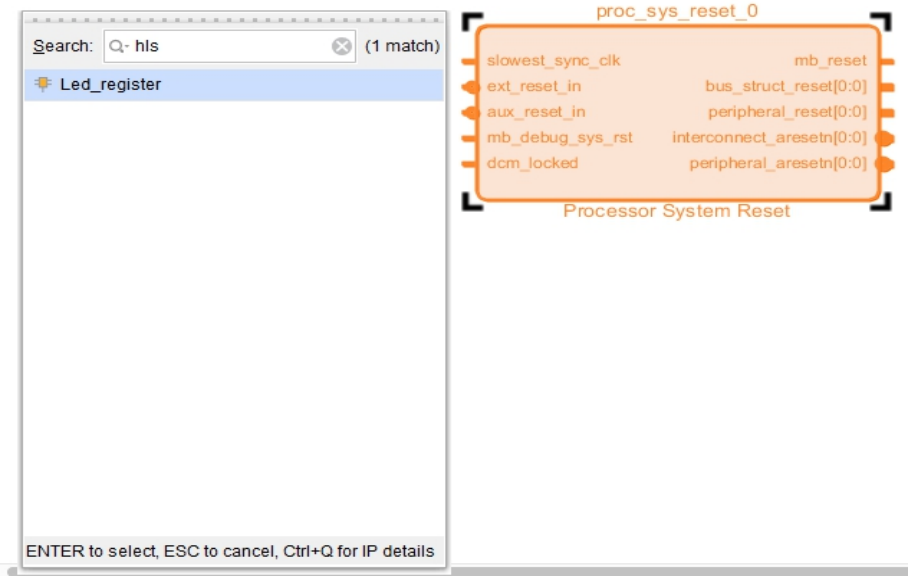
3) Click the plus sign in the interface to enter “zynq”, and configure the IP core according to the Hello world chapter in course_s2



4) Click the plus sign in the interface to enter reset, add reset IP core

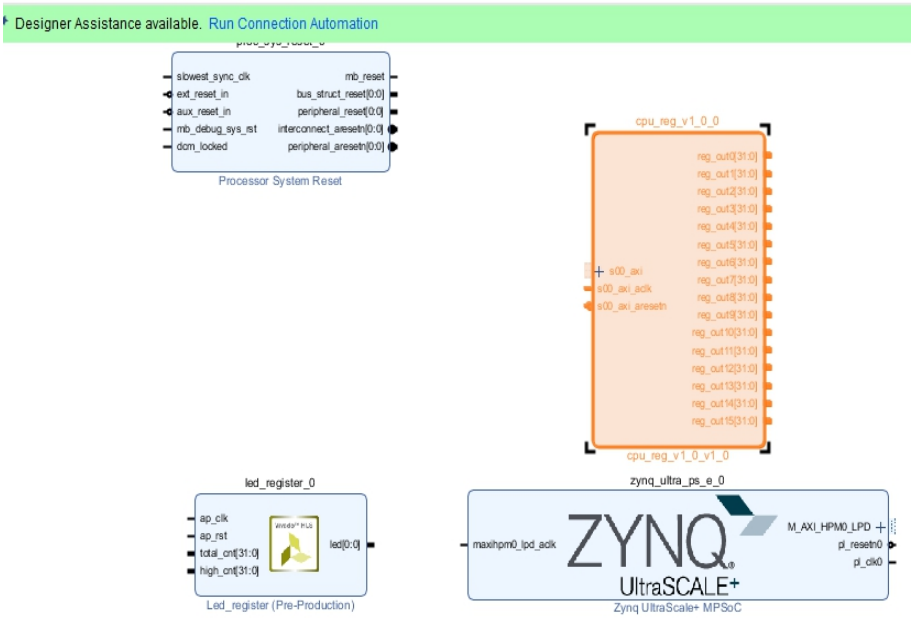


5) Click the plus sign in the interface to enter hls, add the IP core generated by HLS

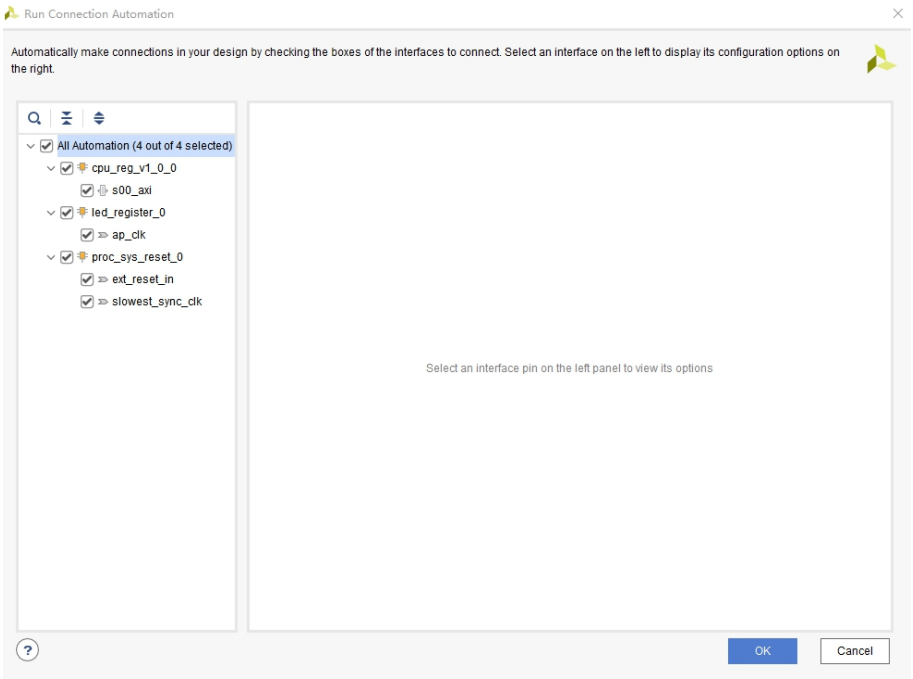


6) In the interface, continue to click the plus sign and enter CPU_reg

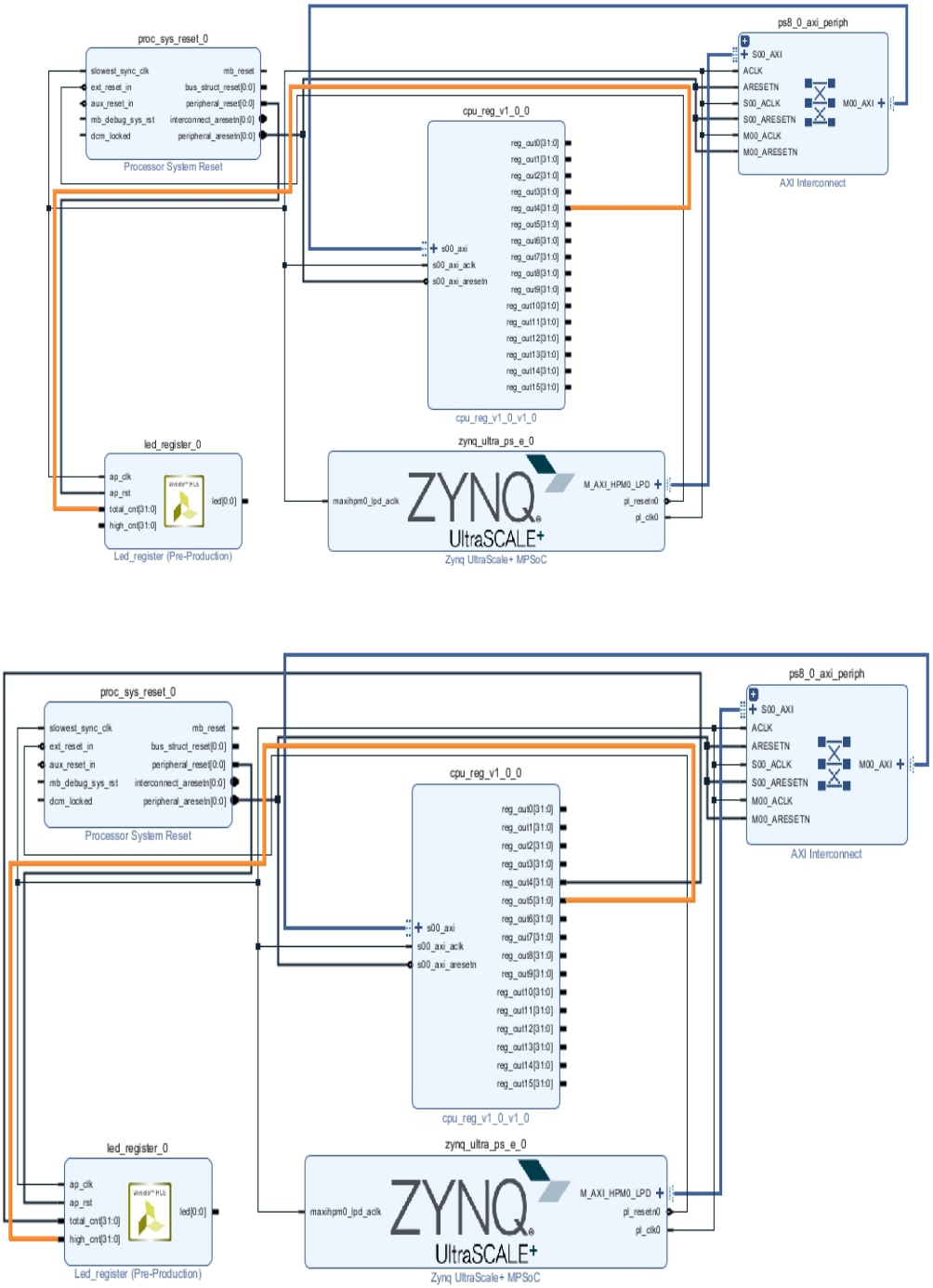
to add the CPU register we generated. Then click run auto connection



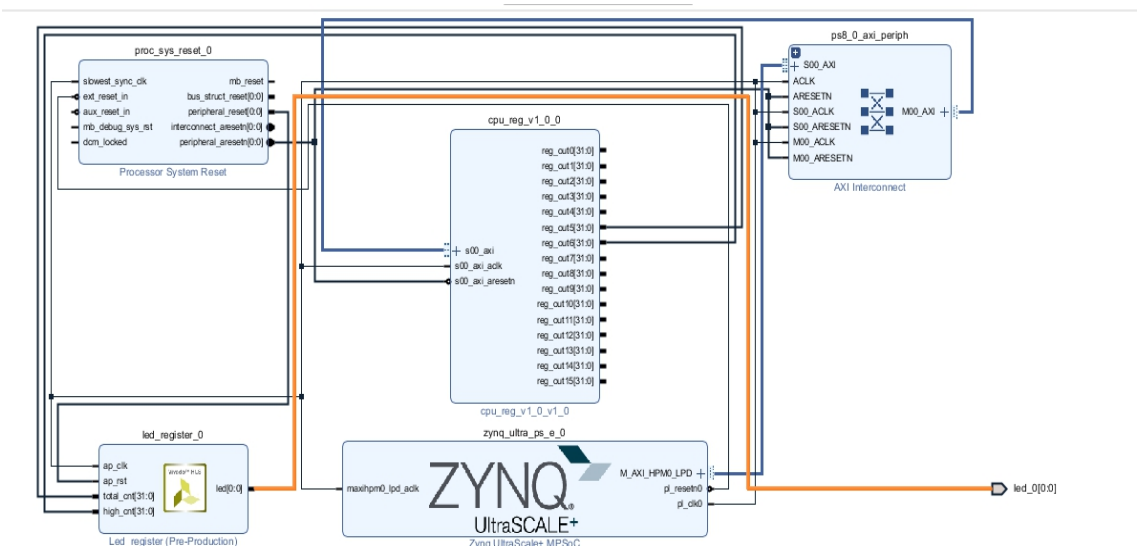
7) Check all



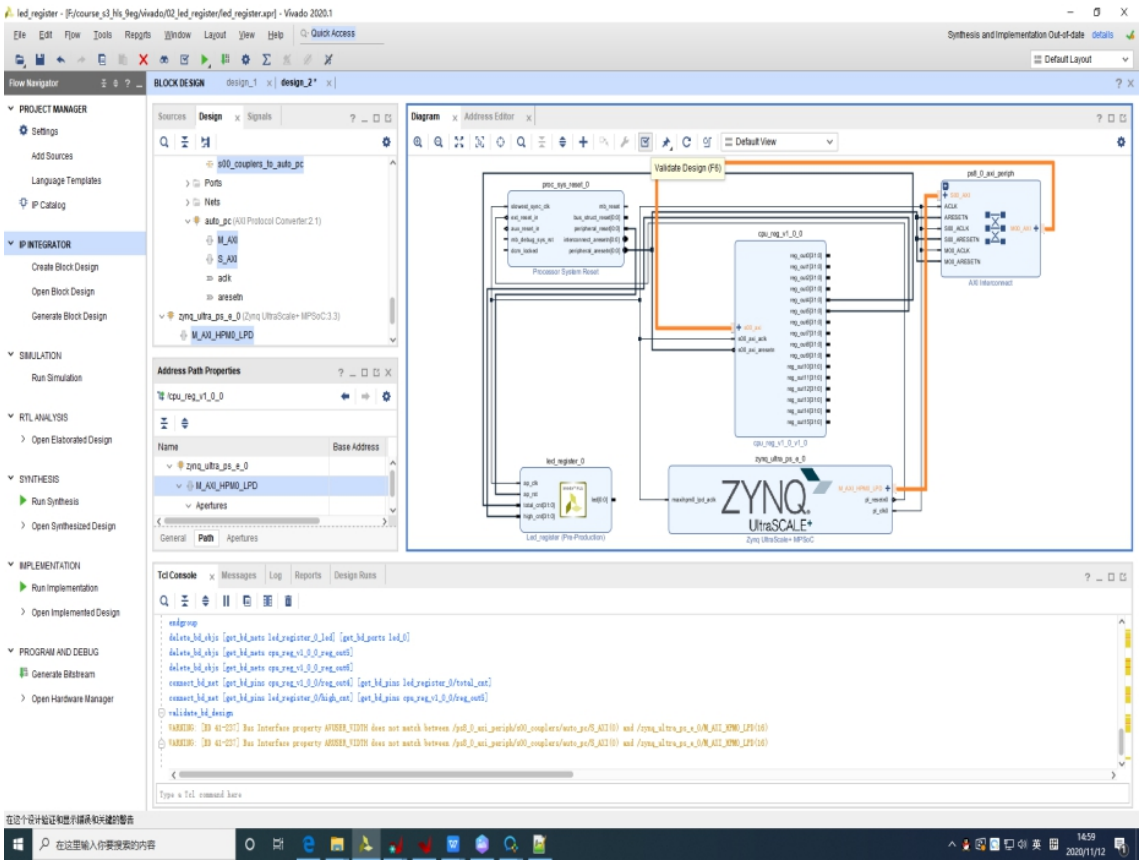
8) After clicking “ok”, connect the IP core as follows

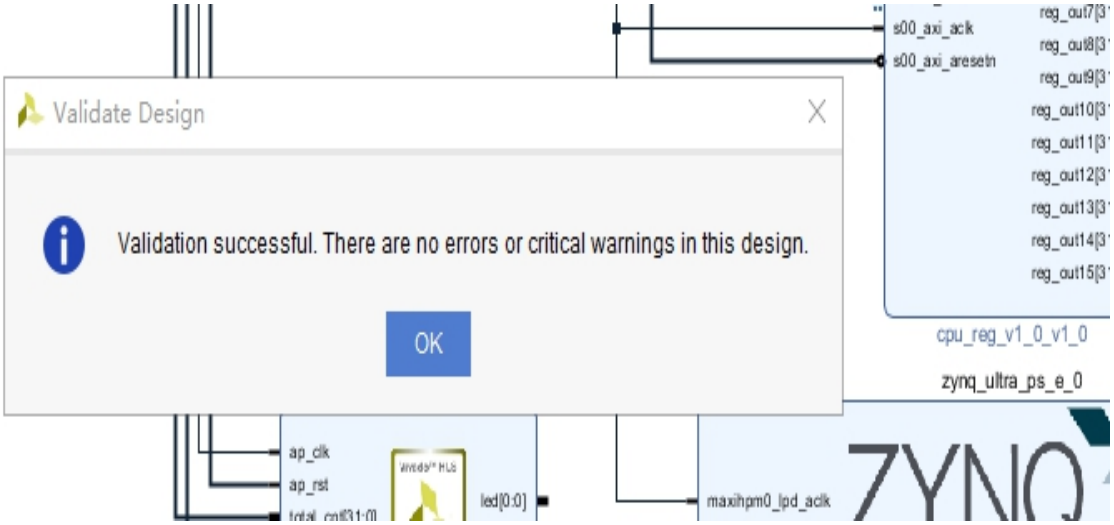


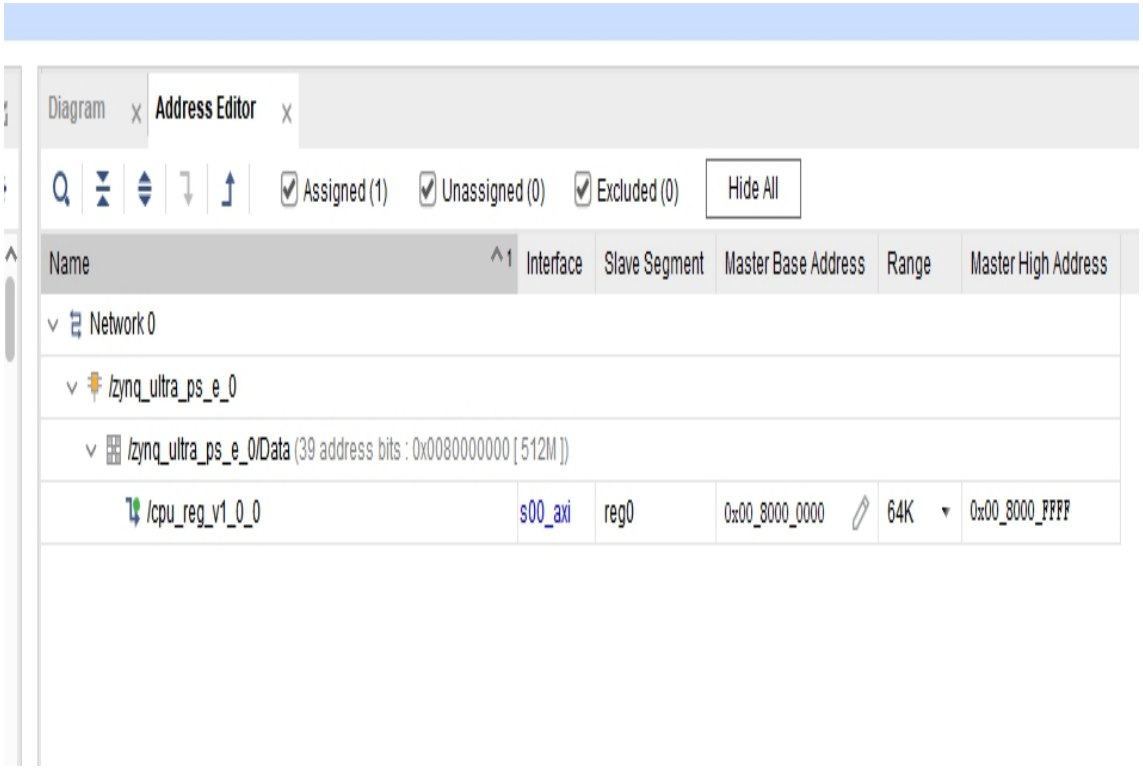
9) In the output port on the right of “led_register_0”, click “ctrl+T” to generate the output port.



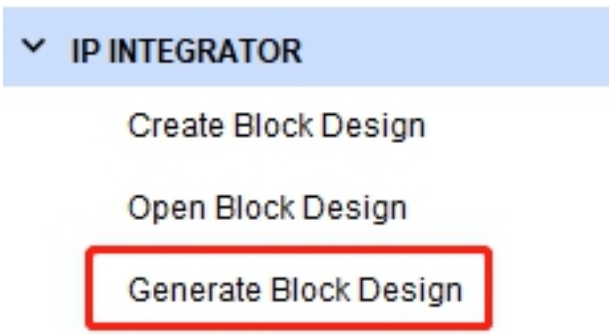
10)Click F6, after confirming that there is no error in the wiring, assign an address to the IP core.



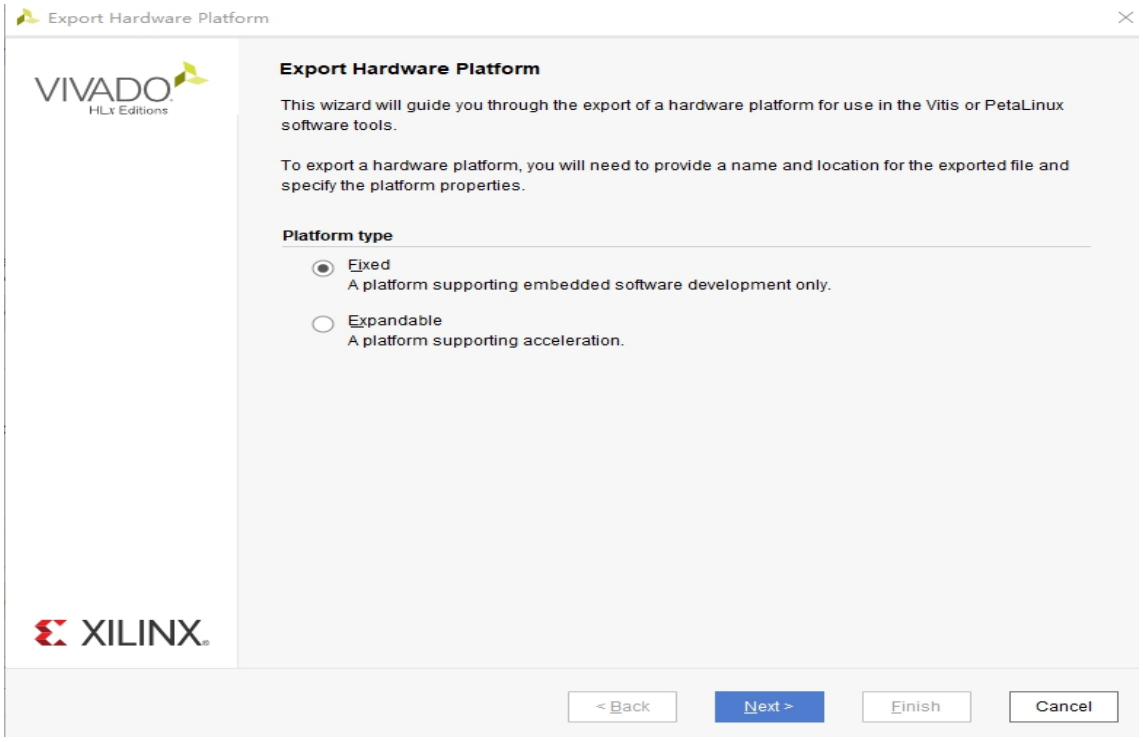




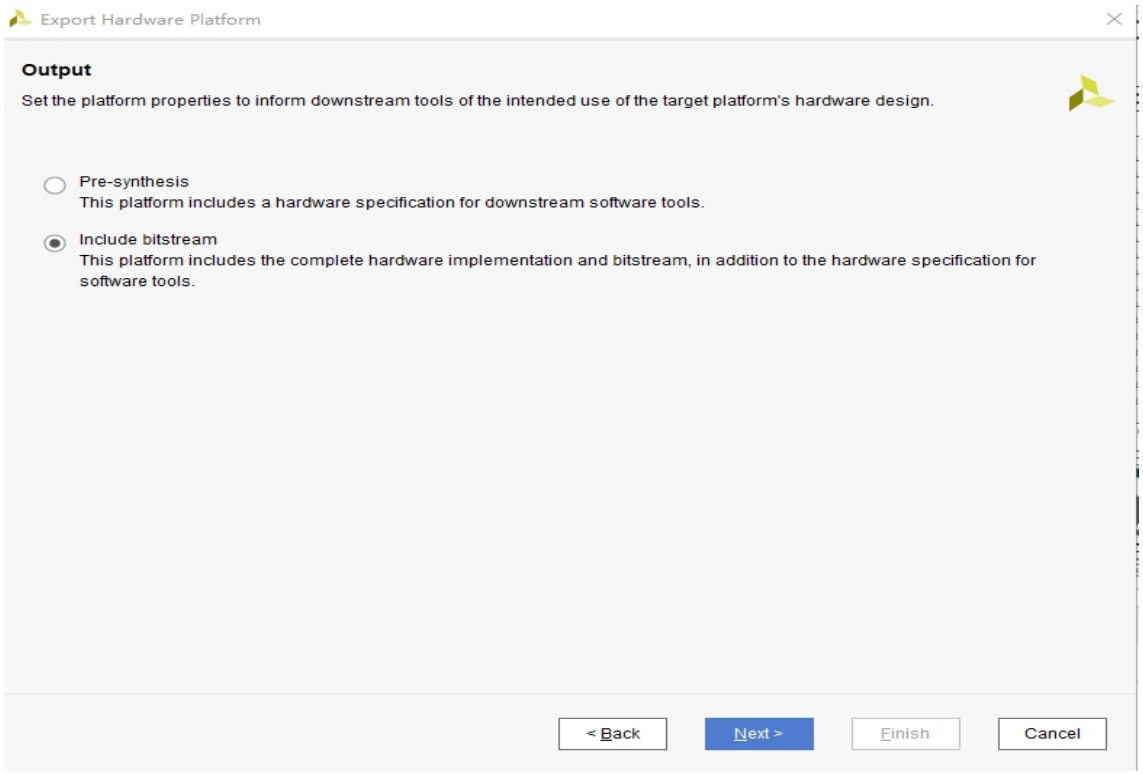
11)Click “generate block design” after finishing



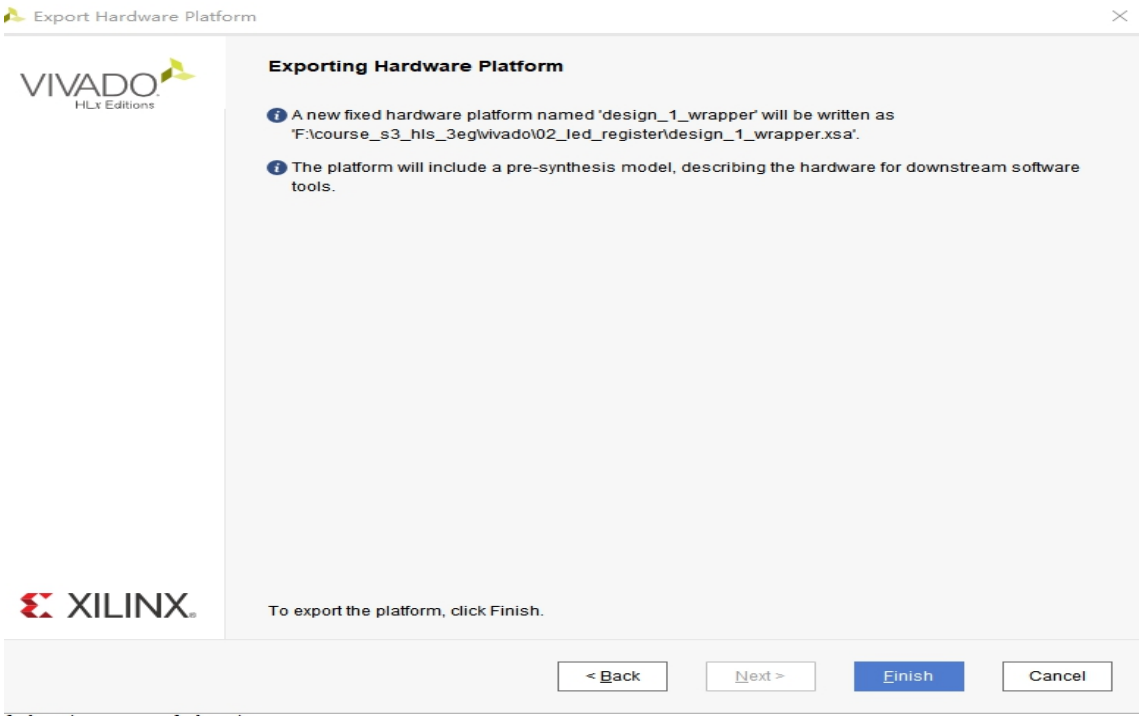
12)Generate a bitstream file, after the completion of the upper left corner “Files->Export->Export Hardware”, select “fixed”



13)Next->Next, Select “include bitstream”

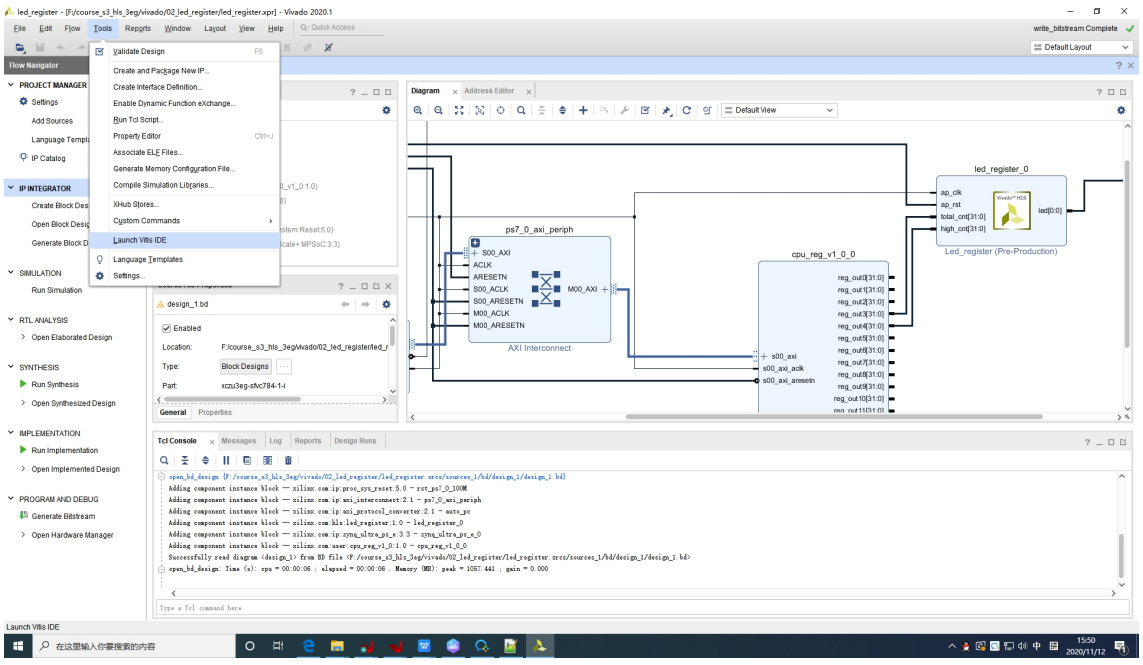


14)Next->Next, Click “finish”

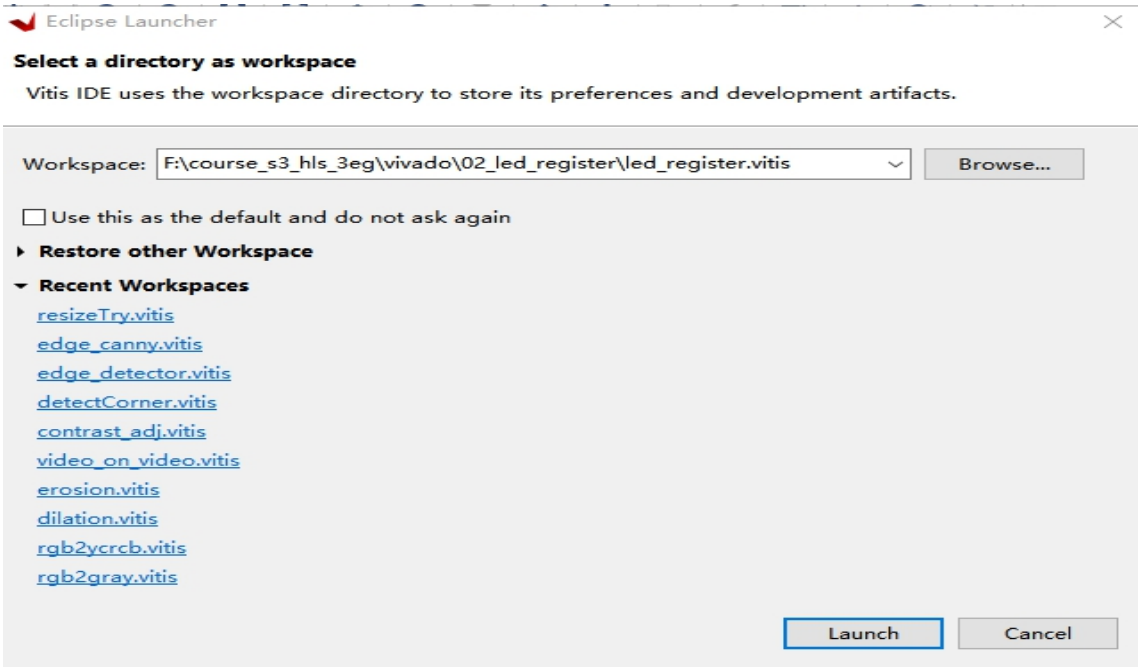


15)Create a new folder “led_register.vitis” in the project

16)Tools->launch vitis

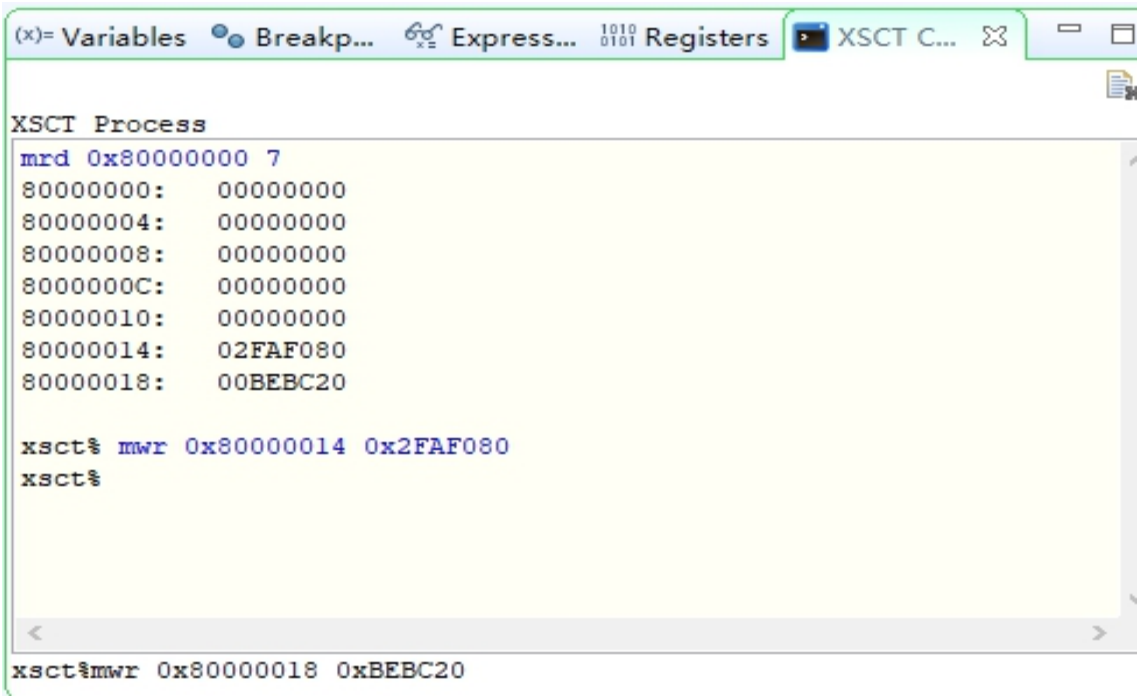


17)Workspace select the folder created



18)Then follow the operation in course_s2 to burn the bitstream into the board

19)After the burning is complete, click “debug” and enter the following command in the “XSTC console”



Note: If we want to read data from the register, enter in the command line: mrd+space+address+read number. If you want to write data from the register, enter in the command line:

mwr+space+address+write data value.

20) You can see that the LED lights continue to flash according to the set flashing cycle and time

Part 4: How to use the built-in Functions in the xfopencv Library

Part 4.1: Introduction to the Experiment

The experiment in this chapter mainly realizes the adjustment of image resolution, which is realized by bilinear interpolation on the image. Xilinx official xfopencv library has this function. What we mainly do is an application-based experiment. Through this experiment, we are familiar with how to use the built-in functions in the official Xilinx xfopencv library to achieve our needs. Readers have the opportunity to practice common functions in the Xilinx library and are familiar with them. Code. Some functions may not work well after being burned. At this time, you need to make some modifications based on your actual needs. The experiments in the subsequent chapters include several routines that directly use the functions in the xfopencv library to find that the effect is not satisfactory. Some have too long delay, and some show black screen or color bars. These functions have been modified and placed in the source folder of the project and can be used directly.

Part 4.2: Experimental Source Code

Top-level function parameter definition

```
#ifndef __resizeTry_h__
#define __resizeTry_h__

#include "ap_int.h"
#include "hls_stream.h"
#include "ap_axi_sdata.h"

#include "common/xf_common.hpp"
#include "common/xf_infra.hpp"
#include "common/xf_utility.hpp"
```

```
#include "xf_resize_nn_bilinear.hpp"

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920

void resizeTry(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst);

#endif
```

Top-level function section of source file

```
#include "resizeTry.h"

void resizeTry(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst)
{
#pragma HLS DATAFLOW
#pragma HLS INTERFACE axis port=dst register_mode=both register
#pragma HLS INTERFACE axis port=src register_mode=both register
#pragma HLS INTERFACE ap_ctrl_none port=return

  xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_src;
#pragma HLS STREAM variable=img_src.data depth=1920 dim=1
  xf::cv::Mat<XF_8UC3,640,640,XF_NPPC1>img_dst;
#pragma HLS STREAM variable=img_dst.data depth=1920 dim=1

  xf::cv::AXIvideo2xfMat(src,img_src);
  xf::cv::resizeNNBilinear<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1,640,640,2,2>(img_src,img_dst);
  xf::cv::xfMat2AXIvideo(img_dst,dst);
}
```

Part 4.3: Interface Synthesis and Project Optimization

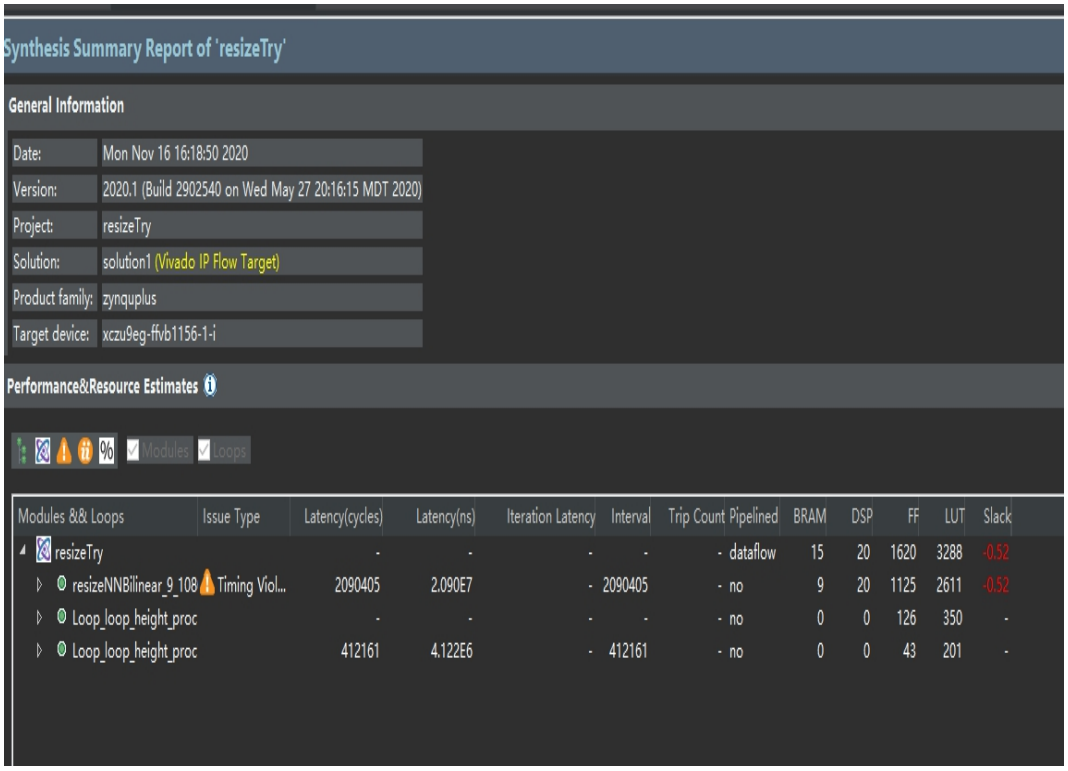
Interface Synthesis

The Block level Interface is set to ap_ctrl_none to remove redundant associated signals. Because the input and output are continuously transformed data, set to axi type. When connecting, it can be directly connected to the previous corresponding IP core.

Project Optimization:

In order to reduce the processing delay, the main function uses Dataflow optimization. The overall function running time is about 20ms. If the clock cycle is set to 10ns in terms of timing, the function may violate the rules in the synthesis of vitis HLS, but the overall report

after the implementation of vivado is Standard, if there is no timing violation in vivado, there is no timing violation actually, and it can be set to 10ns, which can be set with confidence.



Part 4.4: Project Path

Name	Path
Vivado Project	vivado/rgb2gray
HLS Project	hls/ rgb2gray

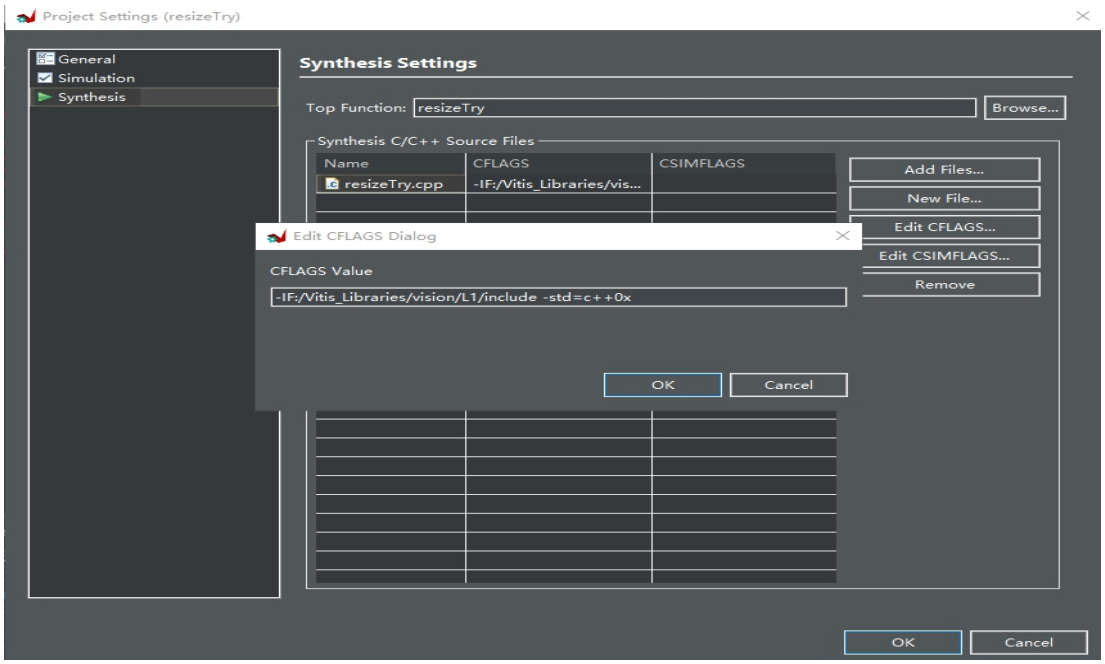
Part 4.5: Project Realization

Let's focus on the project implementation. The connection method and IP core settings refer to the project settings displayed by the mipi camera in course_s2. The data path should be set in the HLS, otherwise there will be an error that the header file cannot be found. Follow the steps below

1) Create an HLS project as mentioned in Part 1 and name it resizeTry
2) Enter the code in the experimental source code in the source file

and header file

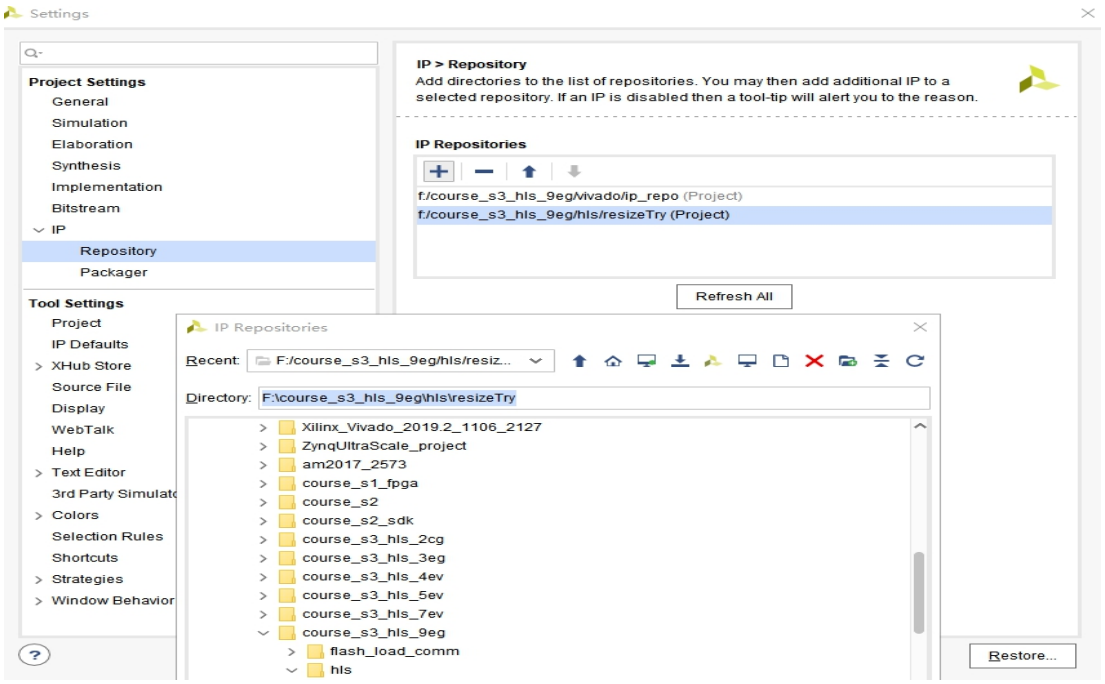
3) Save after input, click  to select the top-level function, select "edit cflags", enter "-I <get-path> -std=c++0x"



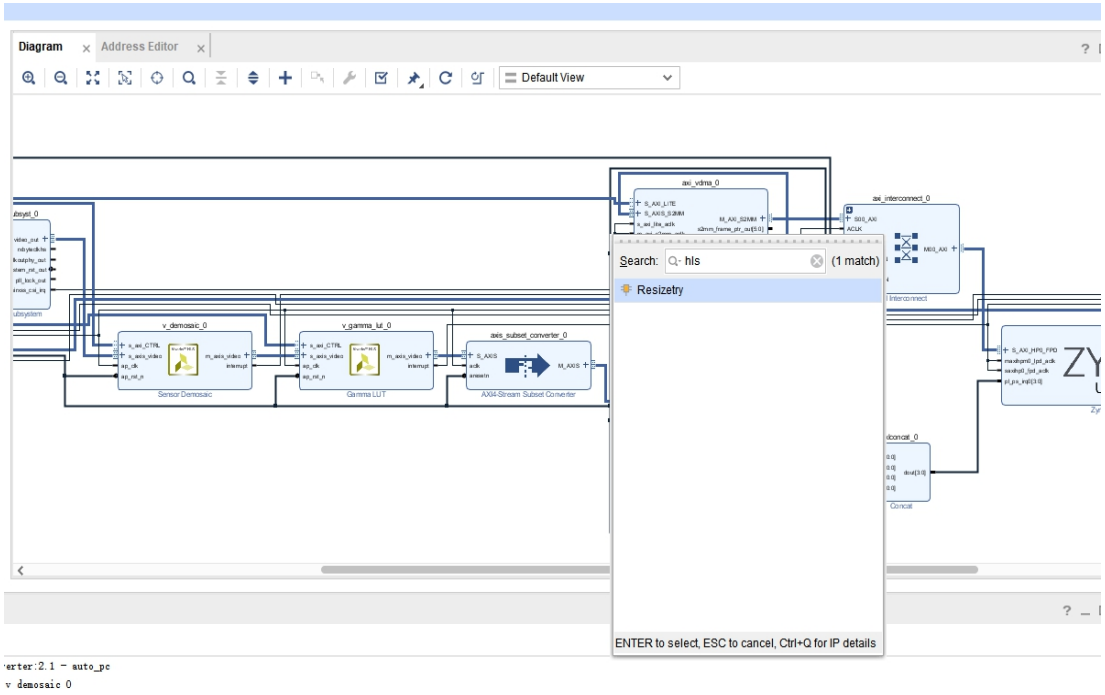
4) Click  to synthesize, and then click  to export IP core

5) Open Vivado, follow the connection method of the mipi project in course_s2 and the IP core setting method

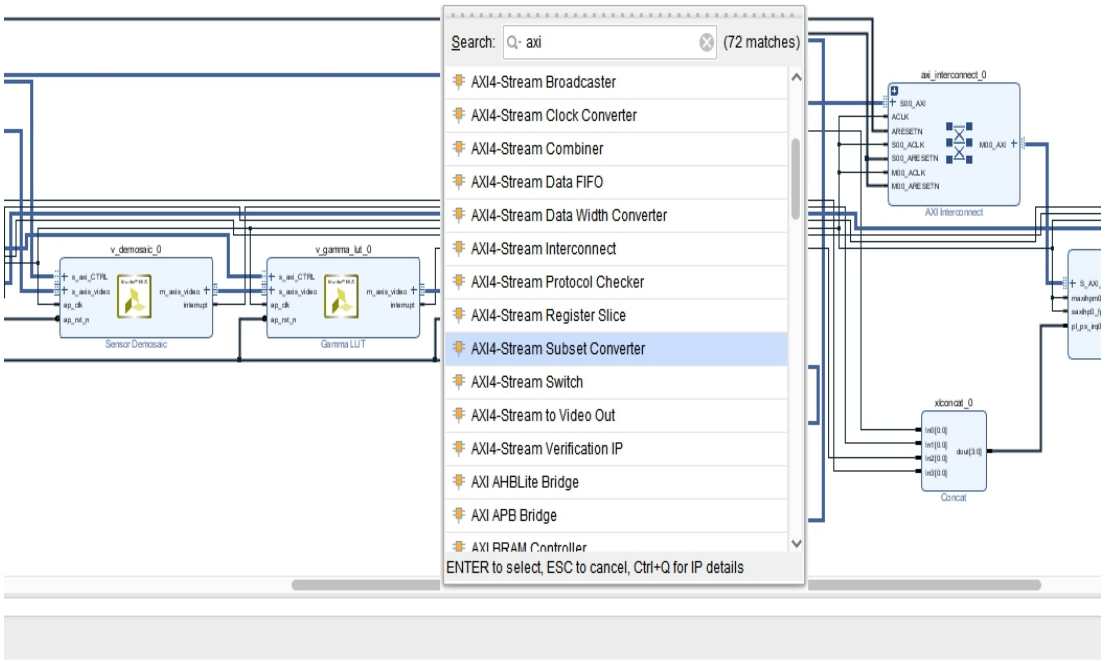
6) Click  to select "IP->repository", add IP core path



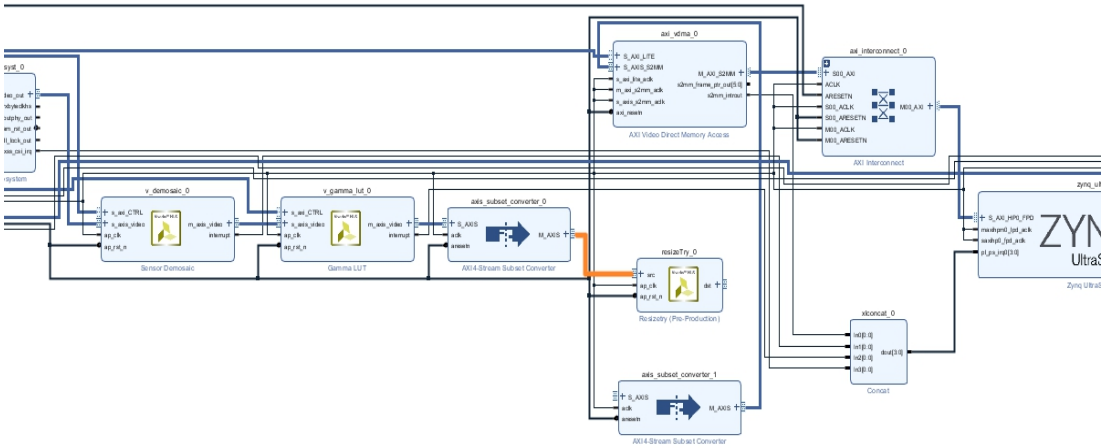
7) Click OK, right-click to add IP core in block design, enter HLS

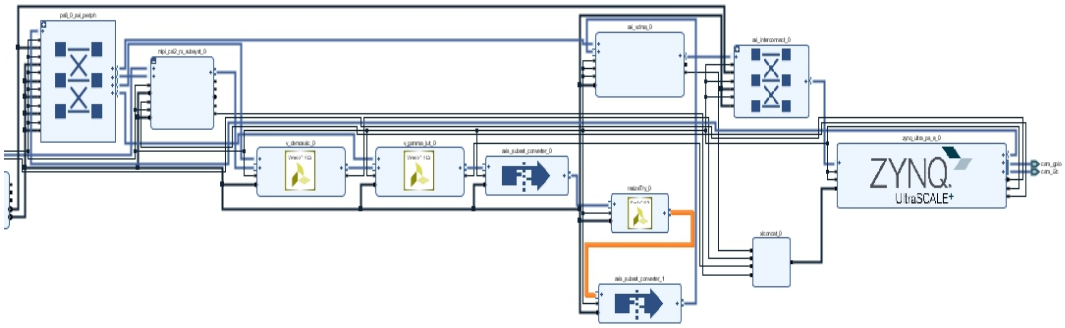


8) Add IP core in block design, enter axi, select axi_subset_convert



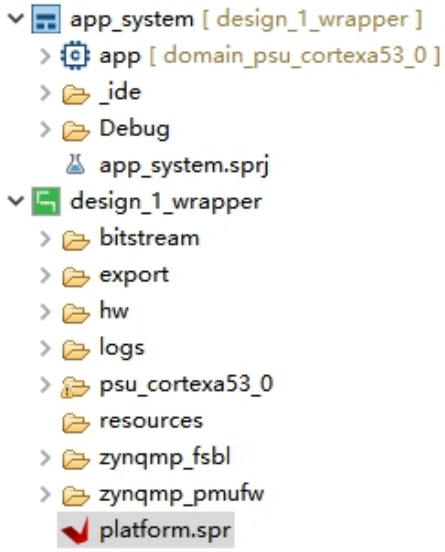
9) Connect and save as shown in the figure, and then generate a bitstream



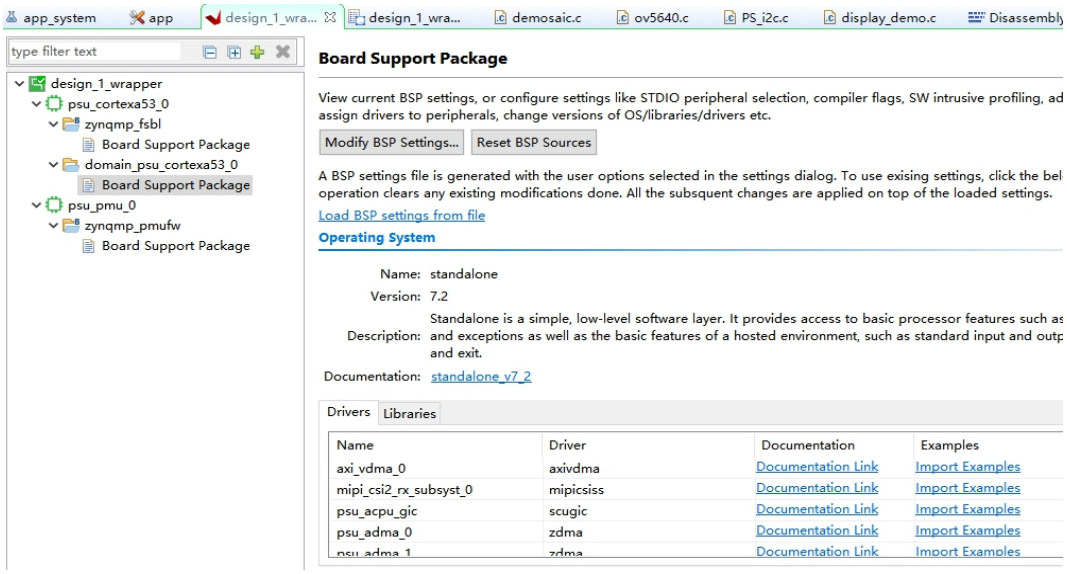


10)Follow the steps in Part 2 to generate the vitis project

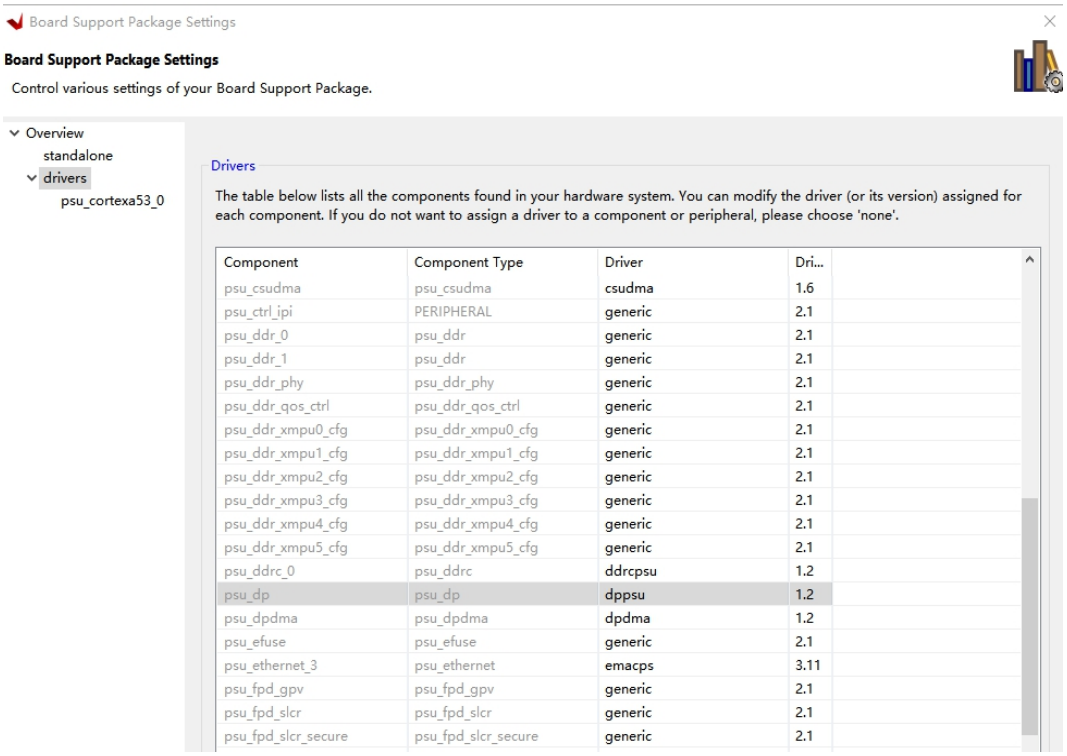
11)Double-click platform.spr to modify the dp driver



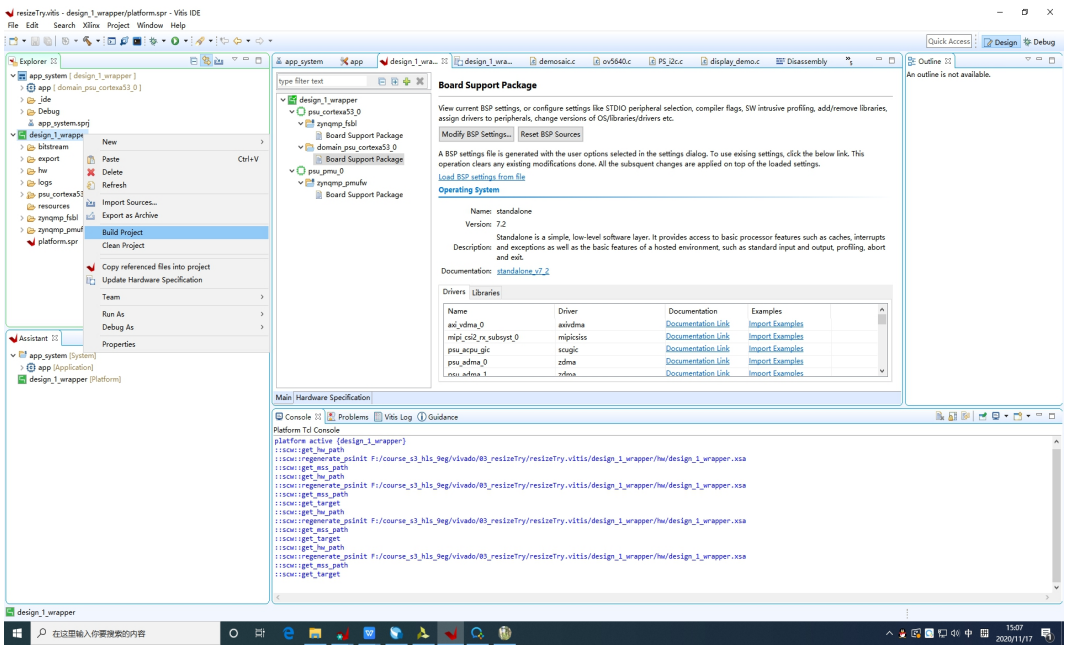
12)Choose the appropriate bsp settings



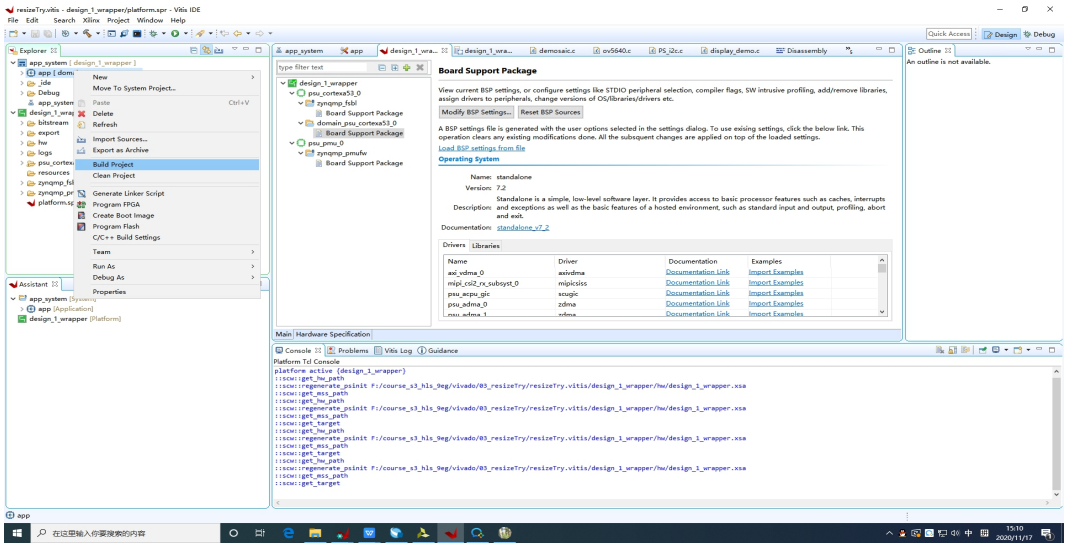
13)Find psu_dp and modify the driver to dppsu



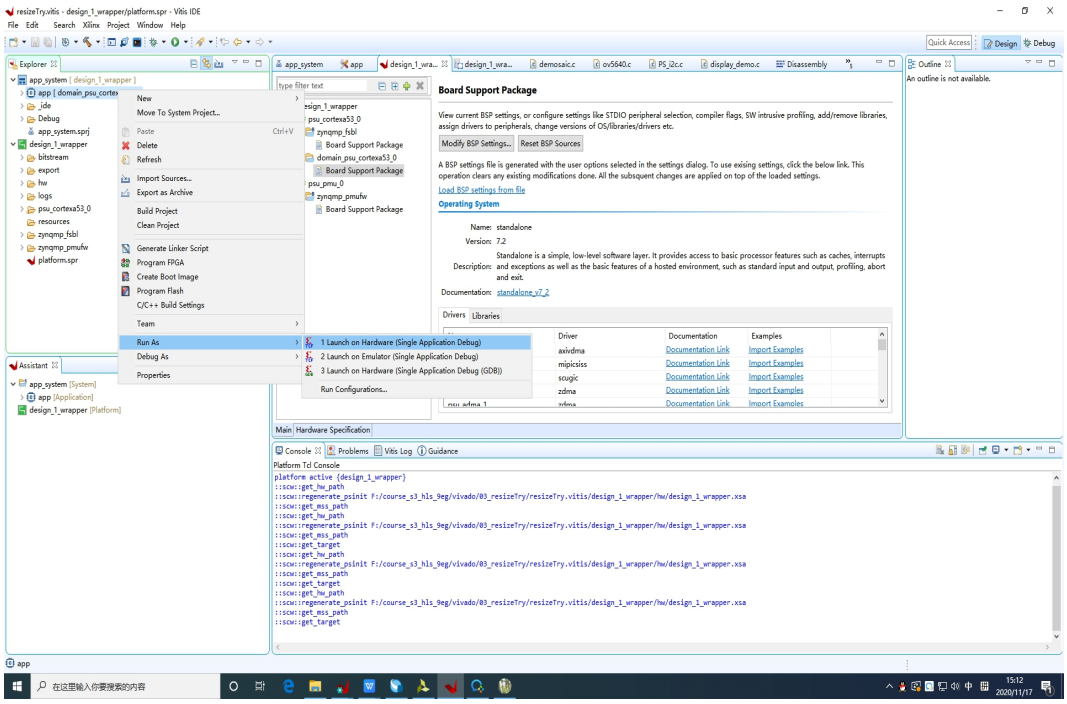
14)Select design_1_wrapper, select build project



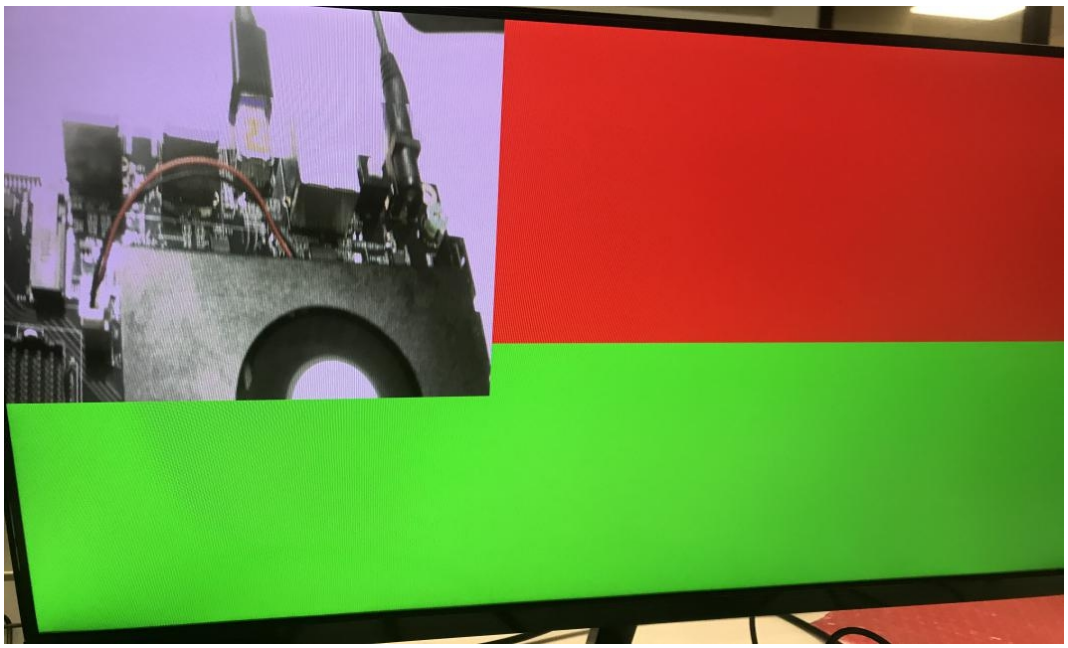
15)After completion, select the created project, select build project



16)After completion, select the created project and select “run as”



17)After a few minutes of burning, you can see the following image displayed on the screen



Part 5: Image RGB to Grayscale Conversion

Part 5.1: Introduction to the Experiment

RGB images are three-channel. Sometimes when doing image processing algorithms, in order to save resources, the input three-channel information is converted into single-channel information. The channel information should contain the original three-channel information. The most common way is to do RGB to grayscale. The conversion method is implemented according to the following formula. After the conversion, the image will become gray and white, and the gray information will be 8 bits. The high eight bits of the input 24-bit data contain B information, the middle eight bits contain G information, and the low eight bits contain R information. After extracting and using the function CalculateGray to do the gray-scale conversion, the converted gray-scale information is spliced into 24 bits, and the 24-bit image information is output, and the display is gray.

Part 5.2: Experimental Source Code

Header File Parameter Definition

```
#ifndef __rgb2gray_h__
#define __rgb2gray_h__

#include "hls_stream.h"
#include "ap_int.h"
#include "ap_axi_sdata.h"

#include "common/xf_common.hpp"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"

#include "imgproc/xf_cvt_color.hpp"

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920
```

```
Void rgb2gray(hls::stream<ap_axiu<24,1,1,1>>&video_in,hls::stream<ap_axiu<24,1,1,1>>&video_out);
    #endif
```

Source File Channel Extraction Part

```
    #include "rgb2gray.h"

    void ExtractPixel(XF_TNAME(XF_8UC3,XF_NPPC1)&src,ap_uint<8>dst[3])
    {
      unsigned int i,j=0;
      for(i=0;i<24;i+=8)
      {
    #pragma HLS UNROLL
            dst[j]=src.range(i+7,i);
            j++;
      }
    }

    template<int ROWS,int COLS>
    void xfrgb2gray(xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&src,xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1>&dst)
    {
      XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
      XF_TNAME(XF_8UC1,XF_NPPC1)gray_packed;
      ap_uint<8>rgb[3];
      ap_uint<8>gray;
      unsigned int i,j=0;
      for(i=0;i<ROWS;i++)
      {
            for(j=0;j<COLS;j++)
            {
    #pragma HLS PIPELINE
                  rgb_packed=src.read(i*COLS+j);
                  ExtractPixel(rgb_packed,rgb);
                  gray=CalculateGRAY(rgb[0],rgb[1],rgb[2]);
                  gray_packed.range(7,0)=gray;
                  dst.write(i*COLS+j,gray_packed);
            }
      }
    }
```

Source File Channel Restoration Part

```
template<int ROWS,int COLS>
void xfgray2rgb(xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1>&src,xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&dst)
{
  XF_TNAME(XF_8UC1,XF_NPPC1)gray_packed;
  XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
  ap_uint<8>gray;
  unsigned int i,j=0;
  for(i=0;i<ROWS;i++)
  {
       for(j=0;j<COLS;j++)
       {
#pragma HLS PIPELINE
            gray_packed=src.read(i*COLS+j);
            gray=gray_packed.range(7,0);
            rgb_packed.range(7,0)=gray;
            rgb_packed.range(15,8)=gray;
            rgb_packed.range(23,16)=gray;
            dst.write(i*COLS+j,rgb_packed);
       }
  }
}
```

Top-level Function Section of Source File

```
void rgb2gray(hls::stream<ap_axiu<24,1,1,1>>&video_in,hls::stream<ap_axiu<24,1,1,1>>&video_out)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS DATAFLOW
#pragma HLS INTERFACE axis port=video_out register_mode=both register
#pragma HLS INTERFACE axis port=video_in register_mode=both register
  xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_rgb_src;
#pragma HLS STREAM variable=img_rgb_src.data depth=1920 dim=1
  xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_gray_src;
#pragma HLS STREAM variable=img_gray_src.data depth=1920 dim=1
  xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_gray_dst;
#pragma HLS STREAM variable=img_gray_dst.data depth=1920 dim=1
  xf::cv::AXIvideo2xfMat(video_in,img_rgb_src);
  xfrgb2gray<IMG_MAX_ROWS,IMG_MAX_COLS>(img_rgb_src,img_gray_src);
  xfgray2rgb<IMG_MAX_ROWS,IMG_MAX_COLS>(img_gray_src,img_gray_dst);
  xf::cv::xfMat2AXIvideo(img_gray_dst,video_out);
}
```

Part 5.3: Interface Settings and Project Optimization

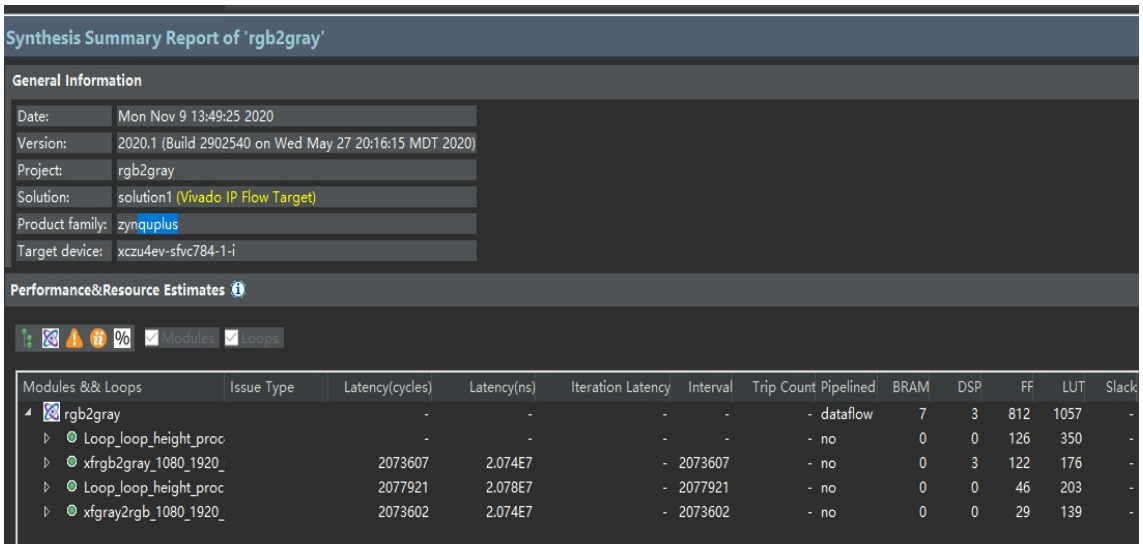
Interface Setting

The Block level Port is set to ap_ctrl_none. The input and output parameters are continuously transmitted data and are set to axi type interfaces.

Comprehensive Optimization:

In order to increase the number of image frames, Dataflow is set

in the top-level function, so the input and output parameters img_rgb_src, img_gray_src and img_gray_dst of the middle two functions are set to stream type, the depth is set to 1920, and pipeline optimization is set in the loop to improve the running speed. The overall resource consumption and delay information is shown in the figure below. It can be seen that the delay is about 20ms, which can meet the number of frames.



Part 5.4: Project Path

Name	Path
Vivado Project	vivado/rgb2gray
HLS Project	hls/ rgb2gray

Part 5.5: Project Realization

After burning, you can see the following image on the monitor with a resolution of 1920*1080

You can see that the image has changed from RGB to gray

Part 6: Image RGB to YCrCb

Part 6.1: Introduction to the Experiment

Since the RGB image mentioned in the previous chapter cannot be restored to an RGB image after being converted to a grayscale image, for some cases where the output is still an RGB image after processing, you can use the RGB to YCrCb image algorithm to convert RGB to YCrCb, After processing, it is restored to RGB image output, where Y refers to the previous grayscale image, and the conversion formula of RGB to YCrCb is as follows:

Y = 0.299*R + 0.587*G + 0.114*B

Cr= (R-Y)*0.713+delta

Cb= (B-Y)*0.564+delta

$$delta = \begin{cases} 128 & for\ 8\text{-}bit\ images \\ 32768 & for\ 16\text{-}bit\ images \\ 0.5 & for\ floating\ point\ images \end{cases}$$

The YCrCb to RGB conversion formula is as follows:

- R= Y+1.403*(Cr-delta)
- G= Y-0.714*(Cr-delta)-0.344*(cb-delta)
- B= Y+1.773+(Cb-delta)

$$delta = \begin{cases} 128 & for\ 8\text{-}bit\ images \\ 32768 & for\ 16\text{-}bit\ images \\ 0.5 & for\ floating\ point\ images \end{cases}$$

In the experiment, the YCrCb image will be displayed first, and then the RGB image will be displayed through the register control

Part 6.2: Experimental Source Code

Parameters in the header file

```
#ifndef __rgb2ycrcb_h__
#define __rgb2ycrcb_h__
```

```
#include "ap_int.h"
#include "hls_stream.h"
#include "ap_axi_sdata.h"

#include "common/xf_common.hpp"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"

#include "imgproc/xf_cvt_color.hpp"

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920

void rgb2ycrcb(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst,ap_uint<1>en);

#endif
```

Extract the channel and integrate the channel part in the source file

```
void ExtractPixel(XF_TNAME(XF_8UC3,XF_NPPC1)&src,ap_uint<8>dst[3])
{
#pragma HLS INLINE off
  unsigned int i,j=0;
  for(i=0;i<24;i+=8)
  {
#pragma HLS UNROLL
        dst[j]=src.range(i+7,i);
        j++;
  }
}

void PackPixel(XF_TNAME(XF_8UC3,XF_NPPC1)&dst,ap_uint<8>src[3])
{
#pragma HLS INLINE off
  unsigned int i,j=0;
  for(i=0;i<24;i+=8)
  {
#pragma HLS UNROLL
        dst.range(i+7,i)=src[j];
        j++;
  }
}
```

RGB to YCrCb part of the source file

```
template<int ROWS,int COLS>
void
xfrgb2ycrcb(xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&src,xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&dst)
{
  XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
```

```
    XF_TNAME(XF_8UC3,XF_NPPC1)ycrcb_packed;
    ap_uint<8>rgb[3];
    ap_uint<8>ycrcb[3];
    unsigned int i,j=0;
    for(i=0;i<ROWS;i++)
    {
        for(j=0;j<COLS;j++)
        {
#pragma HLS PIPELINE
            rgb_packed=src.read(i*COLS+j);
            ExtractPixel(rgb_packed,rgb);
            ycrcb[0]=CalculateGRAY(rgb[0],rgb[1],rgb[2]);
            ycrcb[1]=Calculate_CR(rgb[0],ycrcb[0]);
            ycrcb[2]=Calculate_CB(rgb[2],ycrcb[0]);
            PackPixel(ycrcb_packed,ycrcb);
            dst.write(i*COLS+j,ycrcb_packed);
        }
    }
}
```

YCrCb to RGB part of the source file

```
    template<int ROWS,int COLS>
    void
xfycrcb2rgb(xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&src,xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&dst,ap_uint<1>en
)
    {
      XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
      XF_TNAME(XF_8UC3,XF_NPPC1)ycrcb_packed;
      ap_uint<8>rgb[3];
      ap_uint<8>ycrcb[3];
      unsigned int i,j=0;
      if(en==0)
      {
          for(i=0;i<ROWS;i++)
          {
              for(j=0;j<COLS;j++)
              {
#pragma HLS PIPELINE
                  ycrcb_packed=src.read(i*COLS+j);
                  rgb_packed=ycrcb_packed;
                  dst.write(i*COLS+j,rgb_packed);
              }
          }
      }
      else
      {
          for(i=0;i<ROWS;i++)
          {
              for(j=0;j<COLS;j++)
              {
#pragma HLS PIPELINE
                  ycrcb_packed=src.read(i*COLS+j);
                  ExtractPixel(ycrcb_packed,ycrcb);
```

```
                    rgb[0]=Calculate_Ycrcb2R(ycrcb[0],ycrcb[1]);
                    rgb[1]=Calculate_Ycrcb2G(ycrcb[0],ycrcb[1],ycrcb[2]);
                    rgb[2]=Calculate_Ycrcb2B(ycrcb[0],ycrcb[2]);
                    PackPixel(rgb_packed,rgb);
                    dst.write(i*COLS+j,rgb_packed);
                }
            }
        }
}
```

The top-level function part of the source file

```
    void rgb2ycrcb(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst,ap_uint<1>en)
    {

    #pragma HLS DATAFLOW
    #pragma HLS INTERFACE ap_vld port=en register
    #pragma HLS INTERFACE axis port=dst register_mode=both register
    #pragma HLS INTERFACE axis port=src register_mode=both register
    #pragma HLS INTERFACE ap_ctrl_none port=return
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_src;
    #pragma HLS STREAM variable=img_src.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_gray_src;
    #pragma HLS STREAM variable=img_gray_src.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>img_dst;
    #pragma HLS STREAM variable=img_dst.data depth=1920 dim=1

      xf::cv::AXIvideo2xfMat(src,img_src);
      xfrgb2ycrcb(img_src,img_gray_src);
      xfycrcb2rgb(img_gray_src,img_dst,en);
      xf::cv::xfMat2AXIvideo(img_dst,dst);
}
```

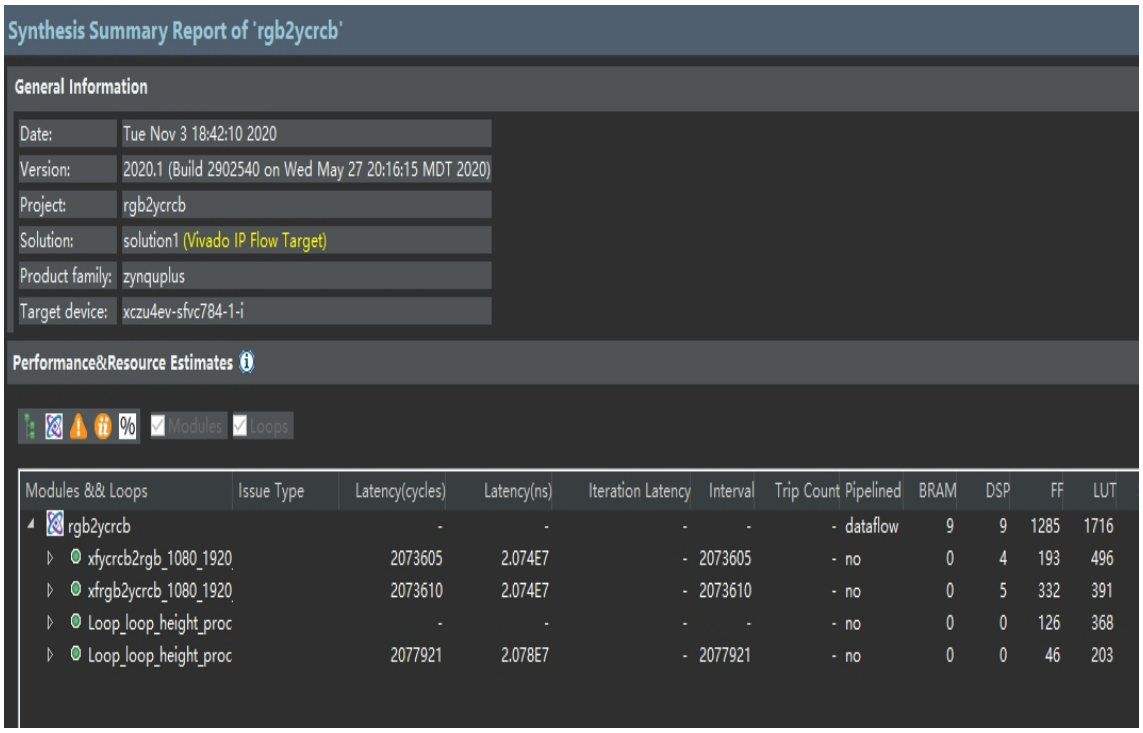
Part 6.3: Interface Synthesis and Project Optimization

Interface Synthesis

The Block Level Interface setting is the same as the previous chapter, the parameter setting is the same as the previous chapter, the parameter en is set to ap_vld type, because this parameter is used to determine whether to perform YCrCb to RGB conversion, and the enable signal is connected to the frame start signal user. And the Parameters are transmitted once per frame

Project Optimization:

The optimization in the project is basically the same as before. Since ExtractPixel and PackPixel are called more frequently, inline=off is set to prevent inline, reduce resource consumption, and expand the loop to reduce delay. The overall resource consumption is shown in the figure



Part 6.4: Project Path

Name	Path
Vivado Project	vivado/rgb2ycrcb
HLS Project	hls/ rgb2ycrcb
BOOT.bin Project	vivado/rgb2ycrcb/rgb2ycrcb.sdk/app/bootimage/BOOT.bin

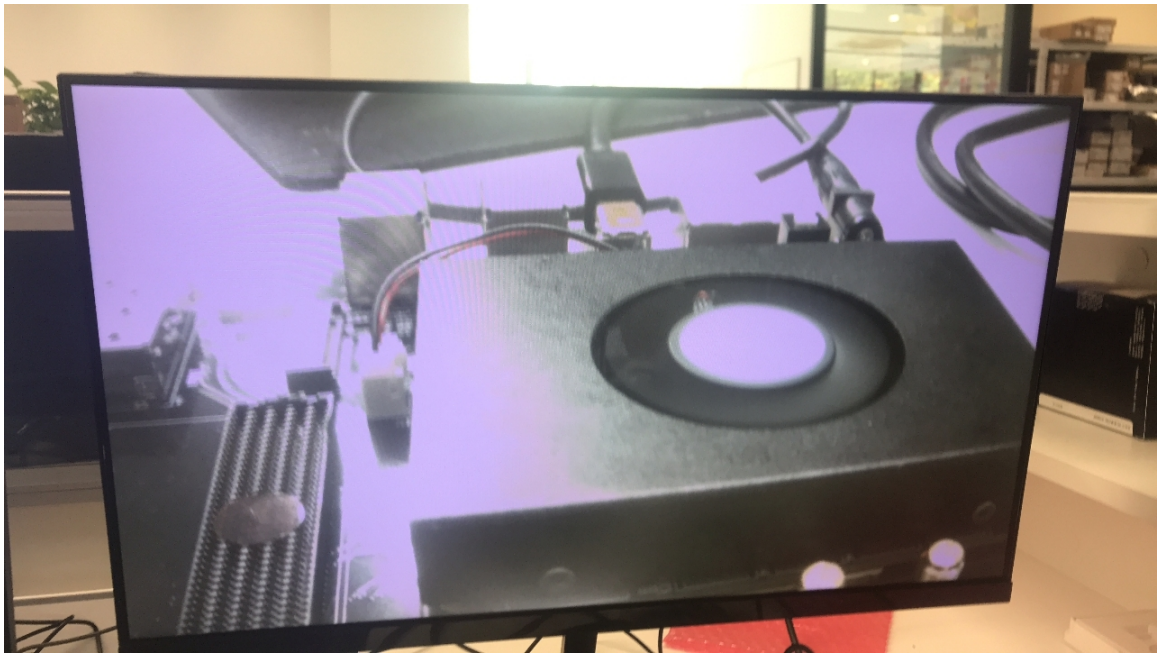
Part 6.5: Project Realization

After burning, the Ycrcb image will be displayed, and the effect is shown in the figure

Enter the following command in vitis

```
XSCT Process
185:    b    _boot
xsct% Info: Cortex-A53 #0 (target 9) Running
mrd 0x80040000 8
80040000:   00000000
80040004:   00000000
80040008:   00000000
8004000C:   00000000
80040010:   00000000
80040014:   00000000
80040018:   00000000
8004001C:   00000000

xsct% mwr 0x8004001c 0x1
xsct%
xsct%
```

The display effect is as follows:



Part 7: Image Morphological Filtering

Part 7.1: Introduction to the Experiment

In this chapter, we try two more common morphological filtering methods, namely corrosion and expansion. Corrosion refers to the pixel with the smallest value at several points around the target pixel and assigns it to the target pixel. The effect is that the low-brightness area of the image increases compared to the original image. Dilation is the opposite of corrosion. The target pixel is assigned to the maximum value around the target pixel. Compared with the original image, the high-brightness area of the image increases, which can be used to enhance the image quality and reduce the discontinuities.

Part 7.2: Experimental Source Code

Expansion part parameter definition of top-level file

```
#ifndef __dilation_h__
#define __dilation_h__

#include "ap_int.h"
#include "ap_axi_sdata.h"
#include "hls_stream.h"

#include "common/xf_common.hpp"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920

#define NEW_K_ROWS  3
#define NEW_K_COLS 3

void dilation(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst);

#endif
```

Corrosion part parameter definition of top-level file

```
#ifndef __erosion_h__
```

```
#define __erosion_h__

#include "ap_int.h"
#include "ap_axi_sdata.h"
#include "hls_stream.h"

#include "common/xf_common.hpp"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"

#include "imgproc/xf_cvt_color.hpp"
#include "hsv2rgb.hpp"
#include "rgb2hsv.hpp"

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920

#define NEW_K_ROWS  3
#define NEW_K_COLS 3

void erosion(hls::stream<ap_axiu<24,1,1,1>>&src,hls::stream<ap_axiu<24,1,1,1>>&dst);

#endif
```

Core code of source file expansion part

```
template <int PLANES, int DEPTH, int K_ROWS, int K_COLS>
void dilate_function_apply(XF_PTUNAME(DEPTH) * OutputValue,
                           XF_PTUNAME(DEPTH) src_buf[K_ROWS][K_COLS],
                           unsigned char kernel[K_ROWS][K_COLS]) {
// clang-format off
    #pragma HLS INLINE
    // clang-format on

    XF_PTUNAME(DEPTH) packed_val = 0, depth = XF_PIXELWIDTH(DEPTH, XF_NPPC1) / PLANES;

Apply_dilate_Loop:
    for (ap_uint<5> c = 0, k = 0; c < PLANES; c++, k += depth) {
// clang-format off
        #pragma HLS LOOP_TRIPCOUNT min=1 max=PLANES
        #pragma HLS UNROLL
        // clang-format on
        XF_PTNAME(DEPTH) max = 0;
        for (ap_uint<13> k_rows = 0; k_rows < K_ROWS; k_rows++) {
// clang-format off
            #pragma HLS LOOP_TRIPCOUNT min=1 max=K_ROWS
            #pragma HLS UNROLL
            // clang-format on
            for (ap_uint<13> k_cols = 0; k_cols < K_COLS; k_cols++) {
// clang-format off
                #pragma HLS LOOP_TRIPCOUNT min=1 max=K_COLS
                #pragma HLS UNROLL
                // clang-format on
                if (kernel[k_rows][k_cols]) {
```

```
                    if (src_buf[k_rows][k_cols].range(k + (depth - 1), k) > max) {
                        max = src_buf[k_rows][k_cols].range(k + (depth - 1), k);
                    }
                }
            }
        }
        packed_val.range(k + (depth - 1), k) = max;
    }

    *OutputValue = packed_val;
    return;
}
```

Core code of source file corrodes part

```
template <int PLANES, int DEPTH, int K_ROWS, int K_COLS>
void erode_function_apply(XF_PTUNAME(DEPTH) * OutputValue,
                          XF_TNAME(XF_8UC3,XF_NPPC1)&buf_value,
                      XF_PTUNAME(DEPTH) src_buf[K_ROWS][K_COLS]) {
// clang-format off
    #pragma HLS INLINE
    // clang-format on

    XF_PTUNAME(DEPTH) packed_val = 0, depth = XF_PIXELWIDTH(DEPTH, XF_NPPC1) / PLANES;

        // clang-format on
        ap_uint<8> min = 255;
        for (ap_uint<13> k_rows = 0; k_rows < K_ROWS; k_rows++) {
// clang-format off
            #pragma HLS LOOP_TRIPCOUNT min=1 max=K_ROWS
            #pragma HLS UNROLL
            // clang-format on
            for (ap_uint<13> k_cols = 0; k_cols < K_COLS; k_cols++) {
// clang-format off
                #pragma HLS LOOP_TRIPCOUNT min=1 max=K_COLS
                #pragma HLS UNROLL
                // clang-format on
                    if (src_buf[k_rows][k_cols].range(23, 16) < min) {
                        min = src_buf[k_rows][k_cols].range(23, 16);
                    }
            }
        }
        packed_val.range(15,0)=buf_value.range(15,0);
        packed_val.range(23,16) = min;

    *OutputValue = packed_val;
    return;
}
```

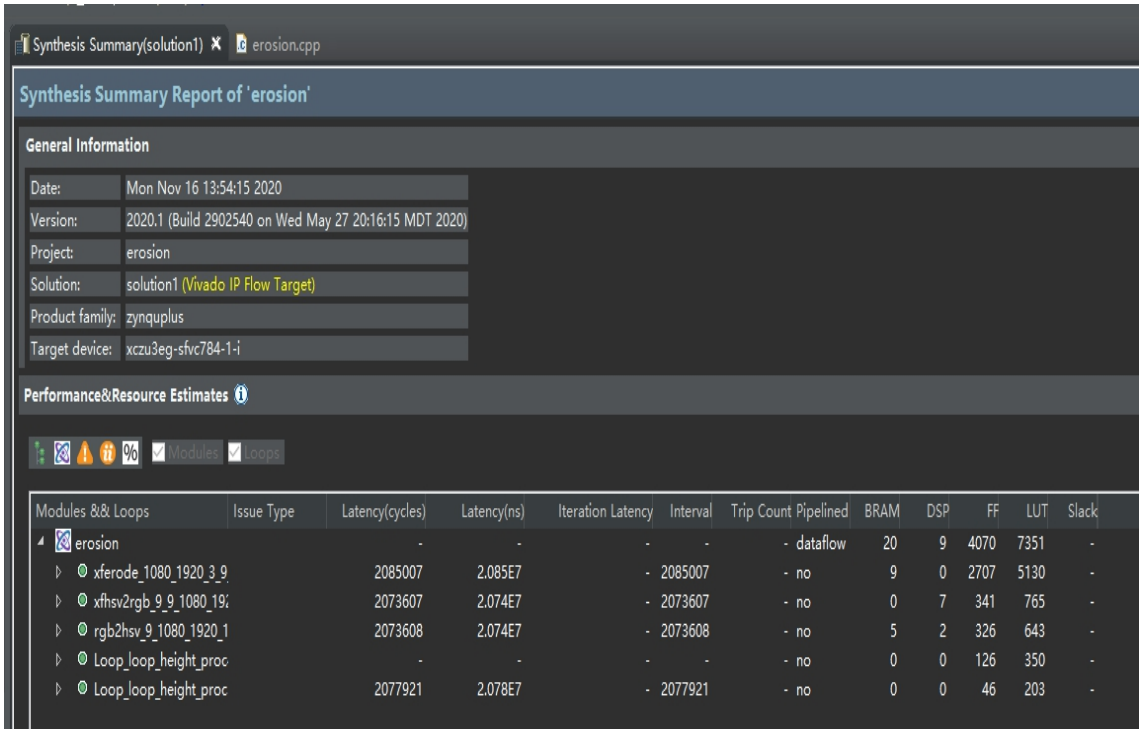
Part 7.3: Interface Synthesis and Project Optimization

Interface Synthesis

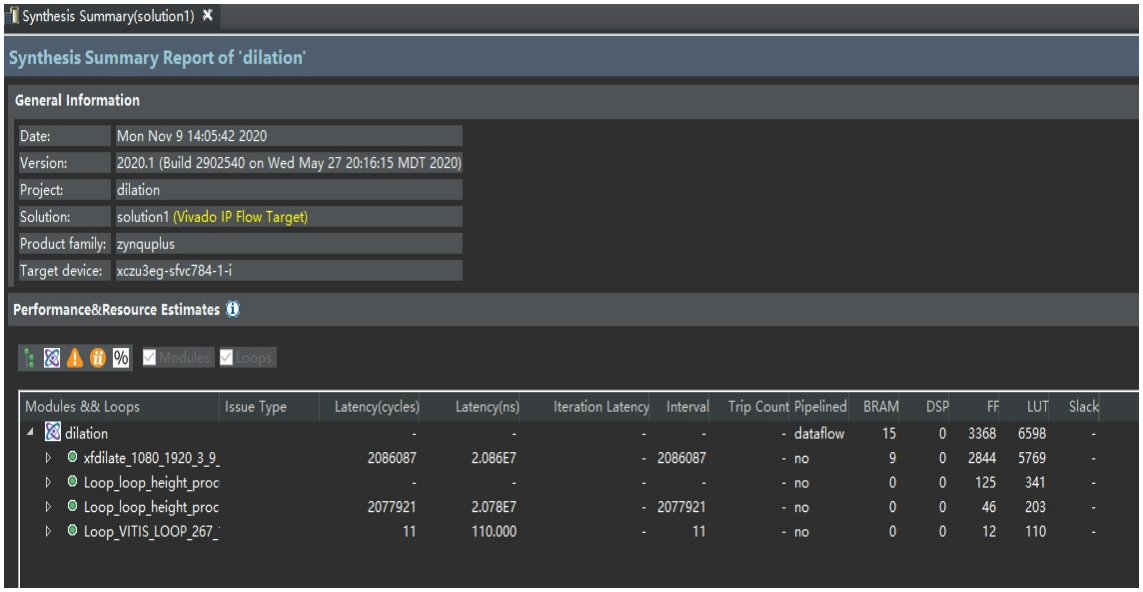
The interface setting is similar to the previous chapter, the Block level Interface is set to ap_ctrl_none to remove redundant associated signals, and the input and output parameters are set to axi type because of continuously changing data.

Project Optimization:

The top-level function is set to Dataflow to reduce the running delay. In the function, because the number of loops is a variable, tripcount is often used in order to estimate the overall running delay. For loops that consume little resources, use loop expansion to improve the overall function running speed. The delay is about 20ms to meet the timing requirements



Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	Slack
erosion		-	-	-	-	-	dataflow	20	9	4070	7351	-
xferode_1080_1920_3_9		2085007	2.085E7	-	2085007	-	no	9	0	2707	5130	-
xfhsv2rgb_9_9_1080_19		2073607	2.074E7	-	2073607	-	no	0	7	341	765	-
rgb2hsv_9_1080_1920_1		2073608	2.074E7	-	2073608	-	no	5	2	326	643	-
Loop_loop_height_proc		-	-	-	-	-	no	0	0	126	350	-
Loop_loop_height_proc		2077921	2.078E7	-	2077921	-	no	0	0	46	203	-

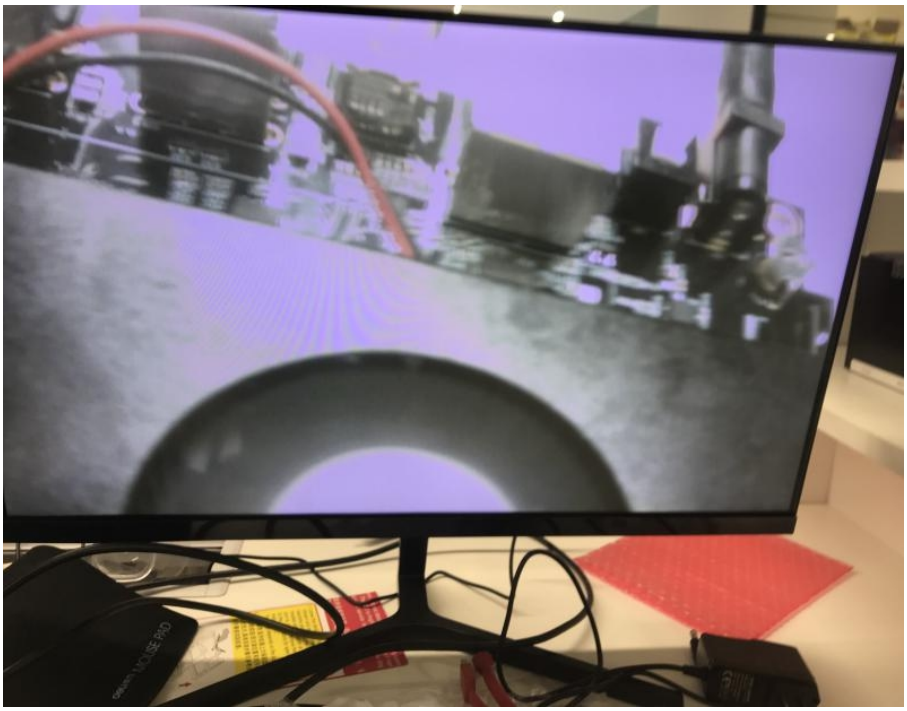


Part 7.4: Project Path

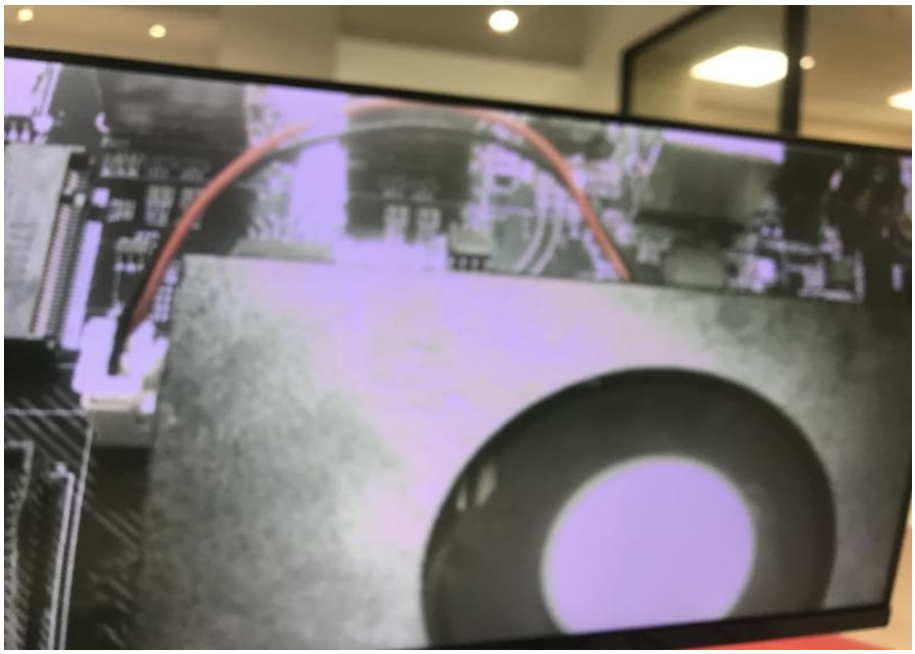
Name	Path
Vivado Project	vivado/dilation
HLS Project	hls/ dilation
Vivado Project	vivado/erosion
HLS Project	hls/ erosion

Part 7.5: Project Realization

Expansion operation display effect



Corrosion operation display effect



Part 8: Image overlay

Part 8.1: Introduction to the Experiment

Perform gray-scale conversion on the video, convert it into a gray-scale image and superimpose it on the original image, control its size, transparency and display position through the register, and observe the phenomenon on the display through DP display

Part 8.2: Experimental Source Code

Top-level file parameter definition

```
#ifndef __overlaystream_h__
#define __overlaystream_h__

#include "ap_int.h"

#include <hls_stream.h>
#include <ap_axi_sdata.h>
#include <string.h>

#include "common/xf_common.hpp"
#include "common/xf_infra.hpp"
#include "common/xf_utility.hpp"
#include "imgproc/xf_resize.hpp"
#include "imgproc/xf_cvt_color.hpp"

typedef xf::cv::ap_axiu<24,1,1,1> pixel_t;
typedef hls::stream<pixel_t> vstream_t;

#define IMG_MAX_ROWS    1080
#define IMG_MAX_COLS    1920

void overlaystream(vstream_t &video_in, vstream_t &video_out,unsigned int overlay_alpha);

#endif
```

Source file overlay

```
template<int ROWS, int COLS>
void overlyOnMat(xf::cv::Mat<XF_8UC3,ROWS, COLS,XF_NPPC1> &img_bottom, xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1>
&img_top, xf::cv::Mat<XF_8UC3,ROWS, COLS,XF_NPPC1>
&img_out,ap_uint<8>overly_alpha,ap_uint<32>overly_x,ap_uint<32>overly_y,ap_uint<32>overly_h,ap_uint<32>overly_w)
{
```

```
    XF_TNAME(XF_8UC3,XF_NPPC1) pixelBottom;
    XF_TNAME(XF_8UC1,XF_NPPC1) pixelTop;
    XF_TNAME(XF_8UC3,XF_NPPC1) pixelMix;
    ap_uint<8> pixelTop_value;
#pragma HLS ARRAY_PARTITION variable=pixelTop_value dim=1 complete
    ap_uint<8> pixelBottom_value[3];
#pragma HLS ARRAY_PARTITION variable=pixelBottom_value dim=1 complete
    ap_uint<8> pixelMix_value[3];
#pragma HLS ARRAY_PARTITION variable=pixelMix_value dim=1 complete
    unsigned int idx=0;
    ap_uint<8>overly_alpha_tmp=0;
    for(int row=0; row<ROWS; row++)
        {
#pragma HLS loop_tripcount min=12 max=1080
            for(int col=0; col<COLS; col++)
            {
#pragma HLS loop_tripcount min=3 max=1920
#pragma HLS pipeline
                pixelBottom=img_bottom.read(row*COLS+col);
                pixelTop=img_top.read(row*COLS+col);
                if((row>=overly_y)&&(row<overly_h)&&(col>=overly_x)&&(col<=overly_w))
                {
                    overly_alpha_tmp=overly_alpha;
                }
                else
                {
                    overly_alpha_tmp=0;
                }
                ExtractPixel(pixelBottom,pixelBottom_value);
                pixelTop_value=pixelTop.range(7,0);

    pixelMix_value[0]=((255-overly_alpha_tmp)*pixelBottom_value[0]+overly_alpha_tmp*pixelTop_value)/255;

    pixelMix_value[1]=((255-overly_alpha_tmp)*pixelBottom_value[1]+overly_alpha_tmp*pixelTop_value)/255;

    pixelMix_value[2]=((255-overly_alpha_tmp)*pixelBottom_value[2]+overly_alpha_tmp*pixelTop_value)/255;
                PackPixel(pixelMix,pixelMix_value);
                img_out.write(row*COLS+col,pixelMix);
            }
        }
}
```

Top-level function section of source file

```
    void overlaystream(vstream_t &video_in, vstream_t &video_out,ap_uint<8>overly_alpha,ap_uint<32>
overly_x,ap_uint<32>overly_y,ap_uint<32>overly_h,ap_uint<32>overly_w)
    {
    #pragma HLS INTERFACE ap_vld port=overly_alpha register
    #pragma HLS INTERFACE ap_vld port=overly_w register
    #pragma HLS INTERFACE ap_vld port=overly_h register
    #pragma HLS INTERFACE ap_vld port=overly_y register
    #pragma HLS INTERFACE ap_vld port=overly_x register
```

```
    #pragma HLS ALLOCATION instances=resize limit=1 function
    #pragma HLS INTERFACE axis port=video_out register_mode=both register
    #pragma HLS INTERFACE ap_ctrl_none port=return
    #pragma HLS INTERFACE axis port=video_in

      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_in;
    #pragma HLS STREAM variable=img_in.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_out;
    #pragma HLS STREAM variable=img_out.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_src1;
    #pragma HLS STREAM variable=img_src1.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1> img_dst1;
    #pragma HLS STREAM variable=img_dst1.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS, XF_NPPC1> img_src2;
    #pragma HLS STREAM variable=img_src2.data depth=1920 dim=1

    #pragma HLS dataflow

      xf::cv::AXIvideo2xfMat(video_in, img_in);
      xfrgb2gray<IMG_MAX_ROWS,IMG_MAX_COLS>(img_in,img_src1,img_src2);
      overlyOnMat<IMG_MAX_ROWS,IMG_MAX_COLS>(img_src2,img_src1,img_out,overly_alpha,overly_x,overly_y,overly_h,o
verly_w);
      xf::cv::xfMat2AXIvideo(img_out, video_out);
}
```

Part 8.3: Interface Synthesis and Project Optimization

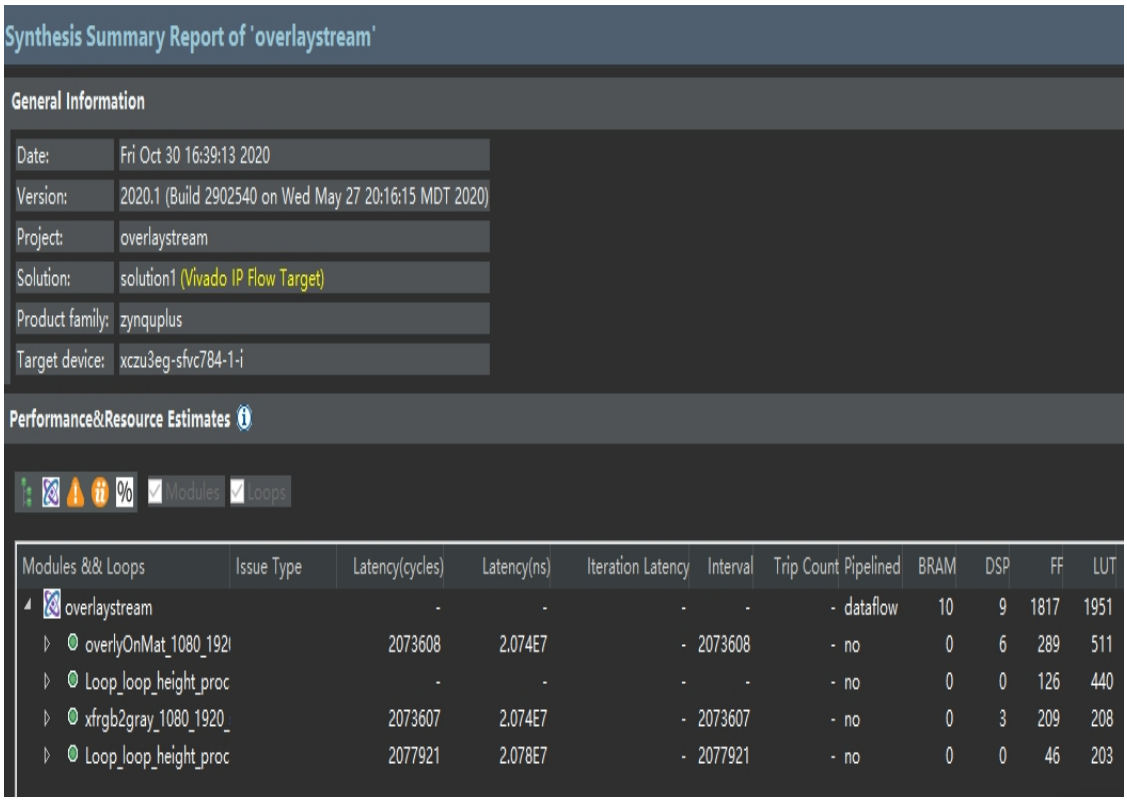
Interface Synthesis

The interface settings are basically the same as the previous chapter. We can also try to read the data from the DDR and do the corresponding processing. When reading the data from the DDR, the interface type needs to be set to m_axi. The parameters of display position and display transparency are sent once per frame, so the parameters are still set to ap_vld type, and the effective signal is connected to the frame start signal.

Project Optimization:

The top-level function in the code sets Dataflow, and the functions in the superimposed part set the Pipeline to the inner function, which reduces the running delay. If the outer function is set to the pipeline, the inner function will be expanded. However, because the top-level function is set to Pipeline function, the parameter transfer

between the functions is integrated into fifo, and there is only one output port. It is actually impossible to expand the inner function, so the outer function cannot be Pipeline optimized. The overall running delay is about 20ms, which can meet the frame number requirement

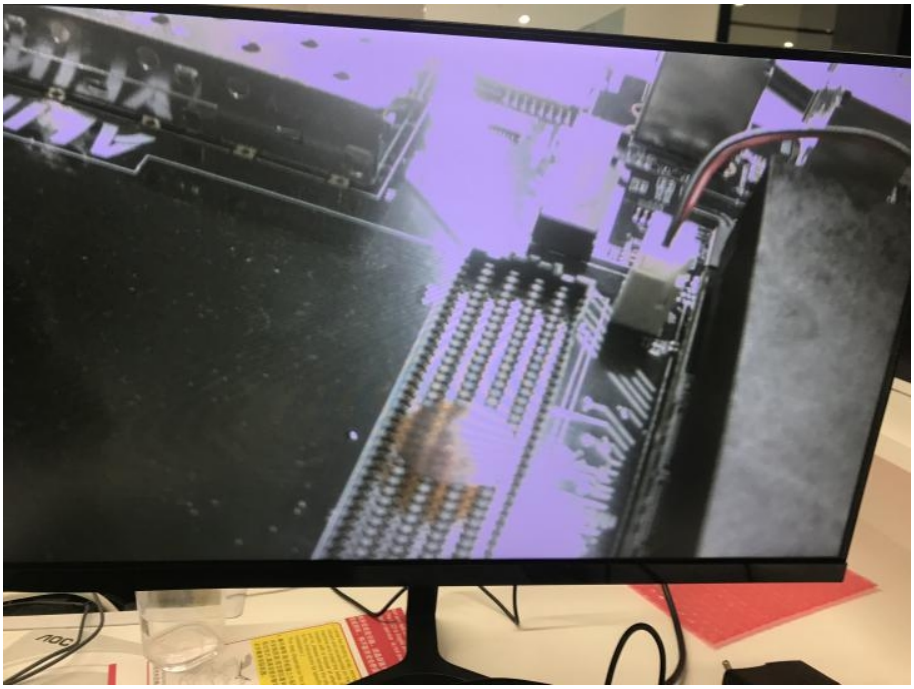


Part 8.4: Project Path

Name	Path
Vivado Project	vivado/video_on_video
HLS Project	hls/ overlaystream

Part 8.5: Project Realization

The image when the contrast is set to 0:



The image when the contrast is set to 255:

The position and transparency of the superimposed image can be adjusted by itself. The maximum transparency is 255 and the minimum is 0. We can set different transparency to observe the results through the register.

Part 9: Image Contrast Adjustment

Part 9.1: Introduction to the Experiment

The image contrast adjustment is realized by equalizing the histogram of the overall image. The histogram is the image grayscale statistical graph, and the overall image brightness distribution statistical graph is obtained according to the grayscale statistical graph of a frame of image. After obtaining the brightness distribution statistical map, we first change the image from RGB to YCrCb, and then according to the statistical value of Y, in order to increase the number of frames. Set the array used to store the equalized results to static, and replace the Y value after the arrival of the next frame with the statistical result of the previous frame. If the statistical value of the same frame is used, the overall processing delay will be increased and the effect will not be much different from the statistical result of the previous frame, so this method is not recommended. After the previous operation, we can enhance the image quality and adjust the contrast of the originally overexposed or underexposed image to a more suitable range.

Part 9.2: Experimental Source Code

Top-level file parameter definition

```
#ifndef __CONTRASTADJ_H__
#define __CONTRASTADJ_H__

#include "ap_int.h"
#include "hls_stream.h"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"
#include "common/xf_common.hpp"
#include "imgproc/xf_cvt_color.hpp"

#define IMG_MAX_HEIGHT  1080
#define IMG_MAX_WIDTH   1920
```

```
    void contrastadj(hls::stream<ap_axiu<24,1,1,1>> &src_axi, hls::stream<ap_axiu<24,1,1,1>> &dst_axi, int
adj);

#endif
```

Source file histogram statistics section

```
    template<int ROWS, int COLS, int N>
    void dualAryEqualize(
           xf::cv::Mat<XF_8UC3,ROWS, COLS,XF_NPPC1>    &_src,
           xf::cv::Mat<XF_8UC3,ROWS, COLS,XF_NPPC1>    &_dst,
           int filter)
    {
      static ap_uint<8> map[N];
      ap_uint<32>hist_out1[N];
    #pragma HLS ARRAY_PARTITION variable=hist_out1 dim=1 complete
      ap_uint<32>hist_out2[N];
    #pragma HLS ARRAY_PARTITION variable=hist_out2 dim=1 complete
      unsigned int i,j=0;
      XF_TNAME(XF_8UC3,XF_NPPC1)src1;
      XF_TNAME(XF_8UC3,XF_NPPC1)src2;
      XF_TNAME(XF_8UC3,XF_NPPC1)dst1;
      XF_TNAME(XF_8UC3,XF_NPPC1)dst2;
      ap_uint<8> value1[3];
      ap_uint<8> value2[3];
      int pos1,pos2;
      for(i=0;i<N;i++)
      {
           hist_out1[i]=0;
           hist_out2[i]=0;
      }


      for(i=0;i<ROWS;i++)
      {
           for(j=0;j<COLS;j+=2)
           {
    #pragma HLS DEPENDENCE variable=src1 intra false
    #pragma HLS DEPENDENCE variable=src2 intra false

    #pragma HLS PIPELINE II=2
                src1=_src.read(i*COLS+j);
                src2=_src.read(i*COLS+j+1);
                ExtractPixel(src1,value1);
                ExtractPixel(src2,value2);
                pos1=(int)value1[0];
                pos2=(int)value2[0];
                hist_out1[pos1]+=1;
                hist_out2[pos2]+=1;
                value1[0]=map[pos1];
                value2[0]=map[pos2];
                PackPixel(dst1,value1);
                PackPixel(dst2,value2);
                _dst.write(i*COLS+j,dst1);
```

```
                _dst.write(i*COLS+j+1,dst2);
            }
        }
        int count_total = 0, tmp_hist;
        float scale;
        for(int i=0;i<N;i++)
        {
    #pragma HLS PIPELINE
            tmp_hist=hist_out1[i]+hist_out2[i];
            count_total += tmp_hist;
            scale = count_total*255/ROWS/COLS;
            scale=(scale-i)*0.01*filter+i;
            if(scale < 16)scale = 16;
            if(scale > 255)scale = 235;
            map[i] =  ap_uint<8>  (scale);
        }
}
```

Top-level function section of source file

```
    void contrastadj(hls::stream<ap_axiu<24,1,1,1>> &src_axi,hls::stream<ap_axiu<24,1,1,1>> &dst_axi,int
adj)
    {

    #pragma HLS INTERFACE ap_ctrl_none port=return
    #pragma HLS INTERFACE ap_vld port=adj register

    #pragma HLS INTERFACE axis port=src_axi
    #pragma HLS INTERFACE axis port=dst_axi

      xf::cv::Mat<XF_8UC3,IMG_MAX_HEIGHT, IMG_MAX_WIDTH, XF_NPPC1> img1;
    #pragma HLS STREAM variable=img1.data depth=1 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_HEIGHT, IMG_MAX_WIDTH, XF_NPPC1> img2;
    #pragma HLS STREAM variable=img2.data depth=1 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_HEIGHT, IMG_MAX_WIDTH, XF_NPPC1> img3;
    #pragma HLS STREAM variable=img3.data depth=1 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_HEIGHT, IMG_MAX_WIDTH, XF_NPPC1> img4;
    #pragma HLS STREAM variable=img4.data depth=1 dim=1

    #pragma HLS dataflow

    // AXIvideoTest<IMG_MAX_HEIGHT,IMG_MAX_WIDTH>(src_axi,dst_axi);
     xf::cv::AXIvideo2xfMat<24,XF_8UC3,IMG_MAX_HEIGHT,IMG_MAX_WIDTH,XF_NPPC1>(src_axi, img1);
      xfrgb2ycrcb<IMG_MAX_HEIGHT,IMG_MAX_WIDTH>(img1, img2);
      dualAryEqualize<IMG_MAX_HEIGHT, IMG_MAX_WIDTH, 256>(img2, img3,adj);
       xfycrcb2rgb<IMG_MAX_HEIGHT,IMG_MAX_WIDTH>(img3, img4);
     xf::cv::xfMat2AXIvideo<24,XF_8UC3,IMG_MAX_HEIGHT,IMG_MAX_WIDTH,XF_NPPC1>(img4,dst_axi);
}
```

Part 9.3: Interface Synthesis and Project Optimization

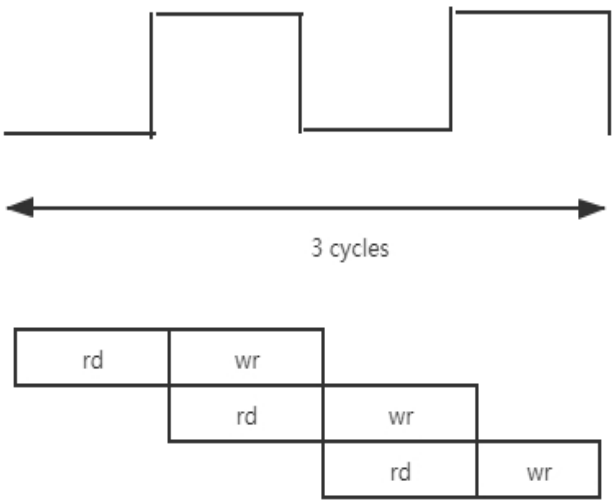
Interface Synthesis

The interface setting is similar to the previous chapter. The Block

level Interface is set to ap_none to remove redundant control signals. Because the input and output are continuously transmitted data, set to axis type, and the contrast parameter is transmitted once per frame.

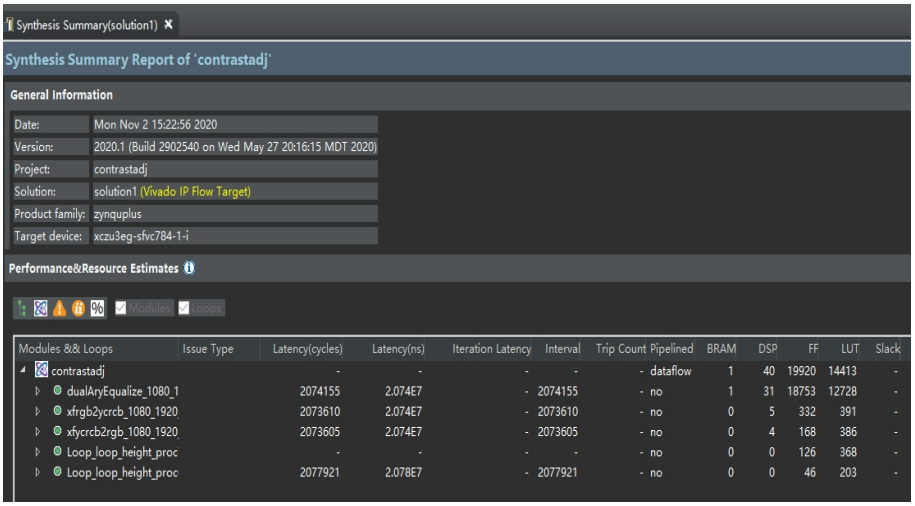
Project Optimization:

Most of the project optimization is similar to the previous chapter. One point to note is that the interval of our Pipeline cannot be set to 1. The main reason can be seen from this picture:



Assuming that we set the interval to 1, we can see from the figure that what we read from the array is the data of the previous clock, although we can set the array to dual_port_ram, But we cannot achieve read and write operations within the same clock, This will bring a special situation that when we perform histogram statistics, the gray values of the two pixels are the same. At this time, according to the code, we will read from the same address and then write, and because our read and write need to be divided into two clocks The data read by the second clock is not the data after the address is written, that is, the increased data, but the data before the increase, which will cause the gray value we counted to be half of the actual gray value One, it will affect the accuracy. The solution to this is to

count two data at a time, use two arrays to store two consecutive pixel statistical results, and finally add the two arrays as a whole to get the actual statistical results of the entire frame, and perform histogram equalization on them. And assign the obtained gray value to the gray value of the next frame. The overall operating delay of the IP core can meet the frame number requirement at about 21ms:



Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	Slack
contrastadj		-	-	-	-	-	dataflow	1	40	19920	14413	-
dualAryEqualize_1080_1		2074155	2.074E7	-	2074155	-	no	1	31	18753	12728	-
xfrgb2ycrcb_1080_1920		2073610	2.074E7	-	2073610	-	no	0	5	332	391	-
xfycrcb2rgb_1080_1920		2073605	2.074E7	-	2073605	-	no	0	4	168	386	-
Loop_loop_height_proc		-	-	-	-	-	no	0	0	126	368	-
Loop_loop_height_proc		2077921	2.078E7	-	2077921	-	no	0	0	46	203	-

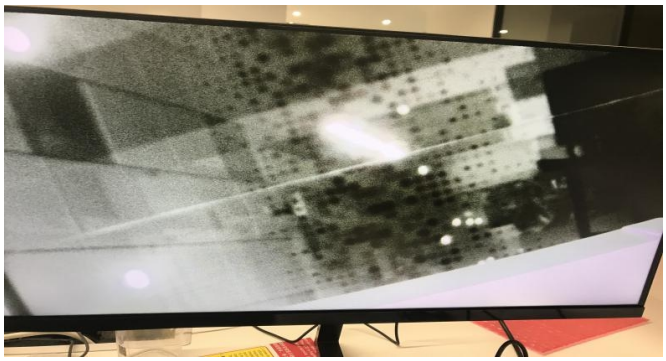
Part 9.4: Project Path

Project Name	Project Path
Vivado Project	vivado/contrast_adj
HLS Project	hls/contrast_adj

Part 9.5: Experimental Result

When the contrast is 0

When the contrast is 100



When the contrast is 100

Part 10: Corner Detection

Part 10.1: Introduction to the Experiment

Corner Detection is a method used to obtain image features in computer vision systems. It is widely used in motion detection, image matching, video tracking, 3D modeling and target recognition. Also known as feature point detec

FAST (Features from Accelerated Segment Test) is a corner detection method originally proposed by Edward Rosten and Tom Drummond. The most outstanding advantage of this algorithm is its computational efficiency. Just like its abbreviated name, it is fast and in fact faster than other well-known feature point extraction methods (such as SIFT, SUSAN, Harris). If applied to machine learning methods, the algorithm can achieve better results. Because of its fast nature, the FAST corner detection method is well suited for real-time video processing.

FAST algorithm steps

1) Select a pixel P from the image, and we will determine if it is a feature point. We first set its brightness value to Ip.
2) Set a suitable threshold t.
3) Consider a discretized Bresenham circle with a radius equal to 3 pixels centered on the pixel point. There are 16 pixels on the boundary of this circle (as shown in Figure 1)
4) Now, if there are “n” consecutive pixels on a circle of 16 pixels, and their pixel values are either larger than “Ip+t” or smaller than “Ip−t”, then it is a corner. (shown by the white dashed line in Figure above). The value of n can be set to “12” or “9”, and experiments have shown that choosing “9” may have a better effect.

In the above algorithm, for each point in the image, we have to traverse the pixels of 16 points on its neighborhood circle, which is inefficient. We present a high-speed test to quickly exclude a large number of non-corner pixels. The method simply checks the pixels at four positions 1, 9, 5 and 13 and first detects position 1 and position 9, and if they are both darker than the threshold or brighter than the threshold, position 5 and position 13 are detected again. If PP is a corner point, then at least 3 of the above four pixels should all be greater than Ip+t or less than Ip−t, because if it is a corner point, more than three-quarters of the circle should satisfy the judgment condition. If not, then p can't be a corner. After doing the preliminary test of all the above points for all the points, the eligible corner points will become the candidate corner points. We will then complete the test for the candidate corner points, that is, detect all the points on the circle.

The above algorithm efficiency is actually very high, but there are some disadvantages

1) When we set n<12, we can't use the fast algorithm to filter the non-corner points.
2) The detected corner point is not optimal because its efficiency depends on the ordering of the problem and the distribution of the corner points;
3) The result of corner analysis was discarded
4) Multiple feature points are easy to squeeze together

Robustness can be enhanced by machine learning and non-maximum suppression. Since the related functions in opencv do not use machine learning, we will only introduce methods for non-maximum suppression. Since the segmentation test does not calculate the corner response function, the conventional non-maximum suppression method is not applicable to the FAST

algorithm. The following is the non-maximum suppression method of FAST:

1) Calculating a score function whose value V is the maximum value of the minimum of all consecutive 10 pixels in the absolute difference between the feature point and its 16 pixels on the circumference, and the value is greater than the threshold t;
2) Compare V in the neighborhood of 3 × 3 feature points (instead of image neighborhood)
3) Eliminate non-maximal feature points

In opencv, there are two core functions for implementing the FAST algorithm. Their prototypes are:

1) void FAST(InputArray image, vector<KeyPoint>& keypoints, int threshold, bool nonmaxSuppression=true)
2) void FASTX(InputArray image, vector<KeyPoint>& keypoints, int threshold, bool nonmaxSuppression, int type)

"Image" is the input image, the requirement is grayscale image, "keypoints" is the detected feature point vector, "threshold" is threshold "t", "nonmaxSuppression" is non-maximum suppression, "true: means non-maximum suppression, "type" is to select circumferential pixel, is 8 (FastFeatureDetector::TYPE_5_8), or 12 (FastFeatureDetector::TYPE_7_12),or 16 (FastFeatureDetector::TYPE_9_16). This parameter is the difference between the "FAST" function and the "FASTX" function. In fact, the FAST function calls the "FASTX" function, and the incoming "type" value is "FastFeatureDetector::TYPE_9_16".

Here we superimpose the detected corners on the same frame of image, first make a copy of the image, then convert one of the images into a single-channel grayscale image and perform corner detection

on it, the other image remains unchanged , After the corner detection is completed, judge whether one of the points is a corner point. If it is a corner point, mark it in green to clearly see the superposition effect. If we want to compare the difference and the impact on the corner detection effect, we can adjust the threshold appropriately, but the threshold should not exceed 10, at this time, corners cannot be detected basically.

Part 10.2: Experimental Source Code

Top file parameter settings

```
    #ifndef DETECTCORNER_H
    #define DETECTCORNER_H

    #include "hls_stream.h"
    #include "ap_int.h"
    #include "string.h"

    #include "common/xf_common.hpp"
    #include "common/xf_utility.hpp"
    #include "common/xf_infra.hpp"

    #include "imgproc/xf_cvt_color.hpp"
    #include "xf_fast.hpp"
    #include "imgproc/xf_dilation.hpp"

    typedef ap_axiu<24,1,1,1> pixel_t;
    typedef hls::stream<pixel_t> stream_t;

    #define IMG_MAX_ROWS    1080
    #define IMG_MAX_COLS    1920
    #define NEW_K_ROWS 16
    #define NEW_K_COLS 16

    void detectCorner(unsigned int *pImgRGB888, unsigned int *pCorner, int rows, int cols, int threhold);

    #endif

#endif
```

Source file copy and integration part

```
    template<int ROWS,int COLS>
    void
xfrgb2gray(xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&src,xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1>&dst1,xf::cv::Mat<
XF_8UC3,ROWS,COLS,XF_NPPC1>&dst2)
```

```
    {
      XF_TNAME(XF_8UC3,XF_NPPC1)pixel_src;
      XF_TNAME(XF_8UC1,XF_NPPC1)pixel_dst1;
      XF_TNAME(XF_8UC3,XF_NPPC1)pixel_dst2;
      ap_uint<8>rgb[3];
      ap_uint<8>gray;
      unsigned int i,j=0;
      for(i=0;i<ROWS;i++)
      {
            for(j=0;j<COLS;j++)
            {
                  pixel_src=src.read(i*COLS+j);
                  ExtractPixel(pixel_src,rgb);
                  gray=CalculateGRAY(rgb[0],rgb[1],rgb[2]);
                  pixel_dst1.range(7,0)=gray;
                  pixel_dst2=pixel_src;
                  dst1.write(i*COLS+j,pixel_dst1);
                  dst2.write(i*COLS+j,pixel_dst2);
            }
      }
    }

    template<int ROWS,int COLS>
    void
xfgray2rgb(xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1>&src1,xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1>&src2,xf::cv::Mat
<XF_8UC3,ROWS,COLS,XF_NPPC1>&dst)
    {
      unsigned int i,j=0;
      XF_TNAME(XF_8UC1,XF_NPPC1)pixel_src1;
      XF_TNAME(XF_8UC3,XF_NPPC1)pixel_src2;
      XF_TNAME(XF_8UC3,XF_NPPC1)pixel_dst;
      for(i=0;i<ROWS;i++)
      {
            for(j=0;j<COLS;j++)
            {
                  pixel_src1=src1.read(i*COLS+j);
                  pixel_src2=src2.read(i*COLS+j);
                  if(pixel_src1==255)
                  {
                  pixel_dst.range(7,0)=0x00;
                  pixel_dst.range(15,8)=pixel_src1;
                  pixel_dst.range(23,16)=0x00;
                  }
                  else
                  {
                        pixel_dst=pixel_src2;
                  }
                  dst.write(i*COLS+j,pixel_dst);
            }
      }
    }
```

At the same time, change the FastProc function in xf_fast.hpp to

this paragraph

```
template <int NPC, int WORDWIDTH, int DEPTH, int WIN_SZ, int WIN_SZ_SQ>
void xFfastProc(XF_PTNAME(DEPTH) OutputValues[XF_NPIXPERCYCLE(NPC)],
                XF_PTNAME(DEPTH) src_buf[WIN_SZ][XF_NPIXPERCYCLE(NPC) + (WIN_SZ - 1)],
                ap_uint<8> win_size,
                uchar_t _threshold,
                XF_PTNAME(DEPTH) & pack_corners) {
// clang-format off
    #pragma HLS INLINE
    // clang-format on

    uchar_t kx = 0, ix = 0;

    // XF_SNAME(WORDWIDTH) tbuf_temp;
    XF_PTNAME(DEPTH) tbuf_temp = 0;

    ////////////////////////////////////////////////
    // Main code goes here
    // Bresenham's circle score computation
    short int flag_d[(1 << XF_BITSHIFT(NPC))][NUM] = {0}, flag_val[(1 << XF_BITSHIFT(NPC))][NUM] = {0};

// clang-format off
    #pragma HLS ARRAY_PARTITION variable=flag_val dim=1
    #pragma HLS ARRAY_PARTITION variable=flag_d dim=1
    // clang-format on

    for (ap_uint<4> i = 0; i < 1; i++) {
// clang-format off
        #pragma HLS LOOP_TRIPCOUNT MAX=1
        #pragma HLS LOOP_FLATTEN off
        #pragma HLS PIPELINE II=1
        // clang-format on
        // Compute the intensity difference between the candidate pixel and pixels on the Bresenham's circle
        flag_d[i][0] = src_buf[3][3 + i] - src_buf[0][3 + i];  // tbuf4[3+i] - tbuf1[3+i];
        flag_d[i][1] = src_buf[3][3 + i] - src_buf[0][4 + i];  // tbuf4[3+i] - tbuf1[4+i];
        flag_d[i][2] = src_buf[3][3 + i] - src_buf[1][5 + i];  // tbuf4[3+i] - tbuf2[5+i];
        flag_d[i][3] = src_buf[3][3 + i] - src_buf[2][6 + i];  // tbuf4[3+i] - tbuf3[6+i];
        flag_d[i][4] = src_buf[3][3 + i] - src_buf[3][6 + i];  // tbuf4[3+i] - tbuf4[6+i];
        flag_d[i][5] = src_buf[3][3 + i] - src_buf[4][6 + i];  // tbuf4[3+i] - tbuf5[6+i];
        flag_d[i][6] = src_buf[3][3 + i] - src_buf[5][5 + i];  // tbuf4[3+i] - tbuf6[5+i];
        flag_d[i][7] = src_buf[3][3 + i] - src_buf[6][4 + i];  // tbuf4[3+i] - tbuf7[4+i];
        flag_d[i][8] = src_buf[3][3 + i] - src_buf[6][3 + i];  // tbuf4[3+i] - tbuf7[3+i];
        flag_d[i][9] = src_buf[3][3 + i] - src_buf[6][2 + i];  // tbuf4[3+i] - tbuf7[2+i];
        flag_d[i][10] = src_buf[3][3 + i] - src_buf[5][1 + i]; // tbuf4[3+i] - tbuf6[1+i];
        flag_d[i][11] = src_buf[3][3 + i] - src_buf[4][0 + i]; // tbuf4[3+i] - tbuf5[0+i];
        flag_d[i][12] = src_buf[3][3 + i] - src_buf[3][0 + i]; // tbuf4[3+i] - tbuf4[0+i];
        flag_d[i][13] = src_buf[3][3 + i] - src_buf[2][0 + i]; // tbuf4[3+i] - tbuf3[0+i];
        flag_d[i][14] = src_buf[3][3 + i] - src_buf[1][1 + i]; // tbuf4[3+i] - tbuf2[1+i];
        flag_d[i][15] = src_buf[3][3 + i] - src_buf[0][2 + i]; // tbuf4[3+i] - tbuf1[2+i];
        // Repeating the first 9 values
        flag_d[i][16] = flag_d[i][0];
        flag_d[i][17] = flag_d[i][1];
        flag_d[i][18] = flag_d[i][2];
        flag_d[i][19] = flag_d[i][3];
        flag_d[i][20] = flag_d[i][4];
```

```
            flag_d[i][21] = flag_d[i][5];
            flag_d[i][22] = flag_d[i][6];
            flag_d[i][23] = flag_d[i][7];
            flag_d[i][24] = flag_d[i][8];

            // Classification of pixels on the Bresenham's circle into brighter, darker or similar w.r.t.
            // the candidate pixel
            for (ap_uint<4> j = 0; j < 8; j++) {
    // clang-format off
                #pragma HLS unroll
                // clang-format on
                if (flag_d[i][j] > _threshold)
                    flag_val[i][j] = 1;
                else if (flag_d[i][j] < -_threshold)
                    flag_val[i][j] = 2;
                else
                    flag_val[i][j] = 0;

                if (flag_d[i][j + 8] > _threshold)
                    flag_val[i][j + 8] = 1;
                else if (flag_d[i][j + 8] < -_threshold)
                    flag_val[i][j + 8] = 2;
                else
                    flag_val[i][j + 8] = 0;
                // Repeating the first 9 values
                flag_val[i][j + PSize] = flag_val[i][j];
            }
            flag_val[i][PSize / 2 + PSize] = flag_val[i][PSize / 2];
            flag_d[i][PSize / 2 + PSize] = flag_d[i][PSize / 2];

            // Bresenham's circle score computation complete

            // Decision making for corners
            uchar_t core = 0;
            uchar_t iscorner = 0;
            uchar_t count = 1;
            for (ap_uint<5> c = 1; c < PSize + PSize / 2 + 1; c++) {
    // clang-format off
                #pragma HLS LOOP_TRIPCOUNT MAX=25
                #pragma HLS UNROLL
                // clang-format on
                if ((flag_val[i][c - 1] == flag_val[i][c]) && flag_val[i][c] > 0) {
                    count++;
                    if (count > PSize / 2) {
                        iscorner = 1; // Candidate pixel is a corner
                    }
                } else {
                    count = 1;
                }
            } // Corner position computation complete
            // NMS Score Computation
            if (iscorner) {
                xFCoreScore(flag_d[i], _threshold, &core);
                pack_corners.range(ix + 7, ix) = 255;
            } else
                pack_corners.range(ix + 7, ix) = 0;
```

```
            ix += 8;
            // Pack the 8-bit score values into 64-bit words
            tbuf_temp.range(kx + 7, kx) = core; // Set bits in a range of positions.
            kx += 8;
        }
        // return tbuf_temp;

        OutputValues[0] = pack_corners; // array[(WIN_SZ_SQ)>>1];
        return;
    }
```

Top-level function section of source file

```
    void detectCorner(hls::stream<ap_axiu<24,1,1,1>>&video_in, hls::stream<ap_axiu<24,1,1,1>>&video_out,int
threshold)
    {
    #pragma HLS INTERFACE ap_vld port=threshold register
    #pragma HLS INTERFACE axis port=video_out register_mode=both register
    #pragma HLS INTERFACE axis port=video_in register_mode=both register
    #pragma HLS INTERFACE ap_ctrl_none port=return
    #pragma HLS DATAFLOW


      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_in;
    #pragma HLS STREAM variable=img_in.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_out;
    #pragma HLS STREAM variable=img_out.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_rgb_src;
    #pragma HLS STREAM variable=img_rgb_src.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_rgb_dst;
    #pragma HLS STREAM variable=img_rgb_dst.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_gray_src;
    #pragma HLS STREAM variable=img_gray_src.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> img_gray_dst;
    #pragma HLS STREAM variable=img_gray_dst.data depth=1920 dim=1


      unsigned char kernel[NEW_K_ROWS][NEW_K_COLS];
      #pragma HLS array_partition variable=kernel dim=0
      // clang-format on
      for (unsigned char i = 0; i < NEW_K_ROWS; i++) {
            for (unsigned char j = 0; j < NEW_K_COLS; j++) {
                  kernel[i][j] = 1; // _kernel[i*NEW_K_COLS+j];
            }
      }

      xf::cv::AXIvideo2xfMat(video_in,img_in);
      xfrgb2gray<IMG_MAX_ROWS,IMG_MAX_COLS>(img_in,img_gray_src,img_rgb_src);
      xf::cv::fast<0,XF_8UC1,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>(img_gray_src,img_rgb_src,img_gray_dst,img_rgb_d
st,threshold);
    //      xf::cv::xfdilate<IMG_MAX_ROWS,IMG_MAX_COLS, XF_CHANNELS(XF_8UC1, XF_NPPC1), XF_8UC1, XF_NPPC1, 0,
(IMG_MAX_COLS >> XF_BITSHIFT(XF_NPPC1)) + (NEW_K_ROWS >> 1),
    //                      NEW_K_ROWS,
NEW_K_COLS>(mask,dmask,matGray.rows,matGray.cols>>XF_BITSHIFT(XF_NPPC1),kernel);
```

```
    //
      overlyOnMat<IMG_MAX_ROWS,IMG_MAX_COLS>(img_rgb_dst,img_gray_dst,img_out,overly_alpha,overly_x,overly_y,ove
rly_h,overly_w);
      xfgray2rgb<IMG_MAX_ROWS,IMG_MAX_COLS>(img_gray_dst,img_rgb_dst,img_out);
      xf::cv::xfMat2AXIvideo(img_out,video_out);
}
```

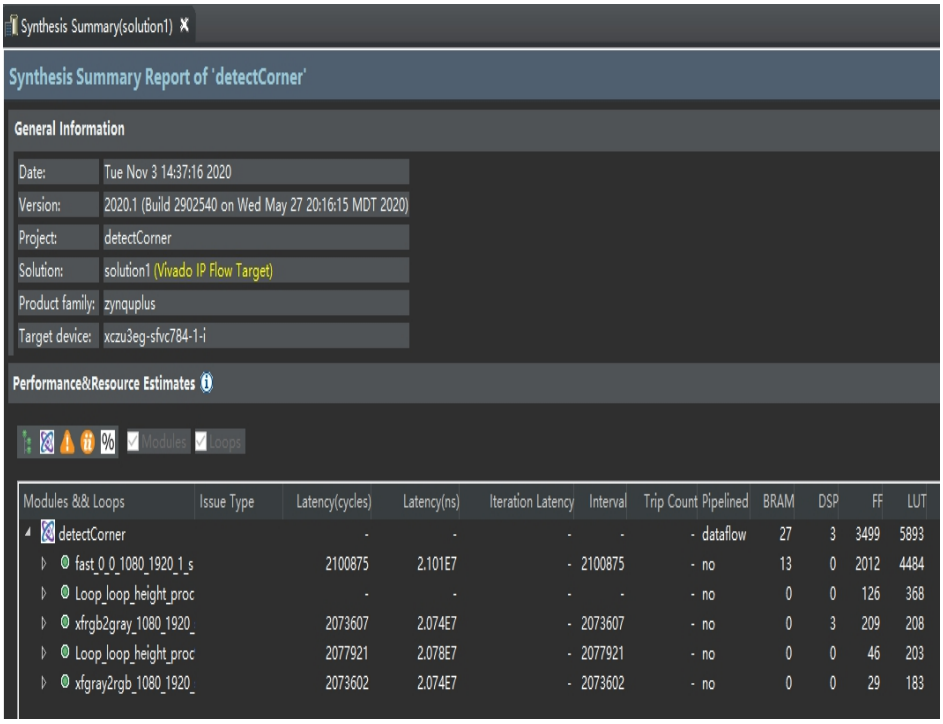
Part 10.3: Interface Setting and Project Optimization

Interface Setting

The interface setting is similar to the previous chapter. Because the input and output are continuous data, it is set to axi type, and the parameter is set to ap_vld type. Each frame is transmitted once to ensure a better display effect of the image.

Project Optimization:

The engineering optimization is similar to the previous chapter. In order to increase the number of image frames, Dataflow optimization is used. It should be noted that when using Dataflow optimization, the function output parameters cannot bypass a function in the middle and make the next function input. At this time, in order to achieve the superimposition effect, the cv::xf::fast function needs to be modified, and input and output are added to it. The xf_fast.hpp file that we have modified is in the source folder of the project. The original xf_fast.hpp function can only achieve the effect of detecting the focus but not the effect of superimposing. After the overall optimization, the running time is about 21ms, which can meet the requirements of the number of frames



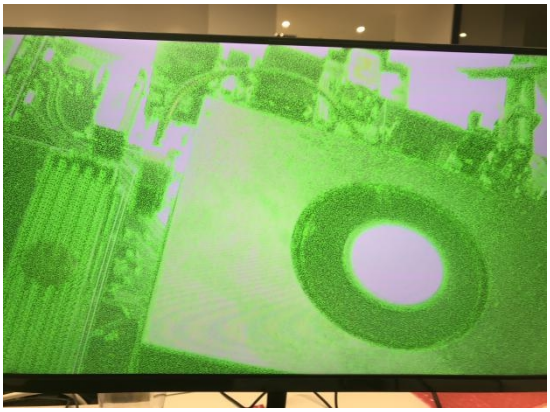
Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
detectCorner		-	-	-	-	-	dataflow	27	3	3499	5893
fast_0_0_1080_1920_1_s		2100875	2.101E7	-	2100875	-	no	13	0	2012	4484
Loop_loop_height_proc		-	-	-	-	-	no	0	0	126	368
xfrgb2gray_1080_1920_		2073607	2.074E7	-	2073607	-	no	0	3	209	208
Loop_loop_height_proc		2077921	2.078E7	-	2077921	-	no	0	0	46	203
xfgray2rgb_1080_1920_		2073602	2.074E7	-	2073602	-	no	0	0	29	183

Part 10.4: Project Path

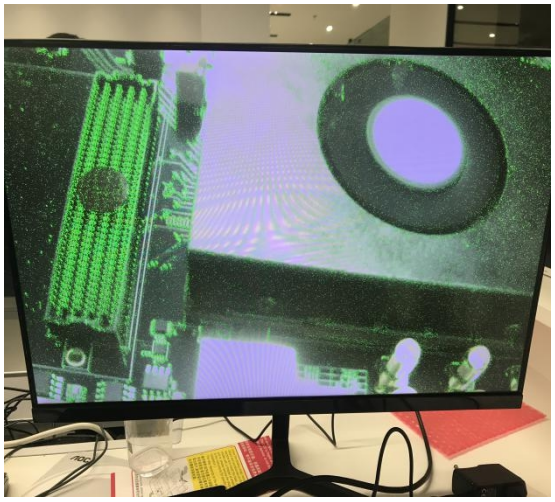
Name	Path
Vivado Project	vivado/detectCorner
HLS Project	hls/ detectCorner

Part 10.5: Project Realization

After burning, the effect will be displayed when the threshold is set to 0



The effect is displayed when the threshold is set to 5



The effect is displayed when the threshold is set to 8

Part 11: SOBEL operator realizes edge detection

Part 11.1: Introduction to the Experiment

Sobel operator, also known as Sobel-Feldman operator, or Sobel filter, is an image edge detection algorithm widely used in image processing and computer vision. It is named after Irwin Sobel and Gray Feldman of the Stanford University Artificial Intelligence Laboratory (SAIL).

The Sobel operator uses two (3x3) matrices to convolve the original image to calculate an estimate of the gray-scale difference (bias) in both directions (one horizontal direction, one vertical direction). We assume that A is the original image (the color image needs to be converted to a grayscale image first), and Gx and Gy are the approximate values of the grayscale partial derivatives in the horizontal and vertical directions, respectively (ie, the planar convolution results of the original image in both directions).

The mathematical expression is as follows: The Sobel operator uses two (3x3) matrices to convolve the original image to calculate an estimate of the gray-scale difference (bias) in both directions (one horizontal direction, one vertical direction). We assume that A is the original image (the color image needs to be converted to a grayscale image first), and Gx and Gy are the approximate values of the grayscale partial derivatives in the horizontal and vertical directions, respectively (ie, the planar convolution results of the original image in both directions). The mathematical expression is as follows:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

The corresponding calculation process is as follows:

Gx = [f(x+1,y-1)+2*f(x+1,y)+f(x+1,y+1)] - [f(x-1,y-1)+2*f(x-1,y)+f(x-1,y+1)]

Gy = [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] - [f(x-1, y+1) + 2*f(x,y+1)+f(x+1,y+1)

In the above formula, f(x, y) is the gray value at (x, y) in the image A. From this you can calculate the Gx and Gy for each point.

For each point in the image, the estimated value G of the gradient can be derived from the gradients Gx and Gy in both directions by:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

Part 11.2: Experimental Source Code

Header file parameter definition

```
#ifndef __edge_detector_h__
#define __edge_detector_h__

#include "ap_axi_sdata.h"
#include "hls_stream.h"
#include "ap_int.h"

#include "common/xf_common.hpp"
#include "common/xf_utility.hpp"
#include "common/xf_infra.hpp"

#include "imgproc/xf_cvt_color.hpp"
#include "imgproc/xf_sobel.hpp"

typedef ap_axiu<24, 1, 1, 1> us_pixel_t;
typedef hls::stream<us_pixel_t> ustream_t;

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920

void edge_detector(ustream_t &src, ustream_t &dst, unsigned char threshold);

#endif
```

The main part of the source file

```
template<int ROWS,int COLS>
void thresholding(xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> &src,
        xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> &dst,
```

```
        ap_uint<8>&threshold)
{
  XF_TNAME(XF_8UC1,XF_NPPC1) pixel_src, pixel_dst;
  unsigned int i,j=0;
  ap_uint<8>value;
  if(threshold==0)
  {
      for(i=0;i<ROWS;i++)
      {
          for(j=0;j<COLS;j++)
          {
#pragma HLS PIPELINE
              pixel_src=src.read(i*COLS+j);
              dst.write(i*COLS+j,pixel_src);
          }
      }
  }
  else
  {
      for(i=0;i<ROWS;i++)
      {
          for(j=0;j<COLS;j++)
          {
#pragma HLS PIPELINE
              pixel_src=src.read(i*COLS+j);
              if(pixel_src>threshold)
              {
                  pixel_dst=0;
              }
              else
              {
                  pixel_dst=255;
              }
              dst.write(i*COLS+j,pixel_dst);
          }
      }
  }
}


template<int ROWS,int COLS>
void thresholding(xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> &src,
        xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> &dst,
        ap_uint<8>&threshold)
{
  XF_TNAME(XF_8UC1,XF_NPPC1) pixel_src, pixel_dst;
  unsigned int i,j=0;
  ap_uint<8>value;
  if(threshold==0)
  {
      for(i=0;i<ROWS;i++)
      {
          for(j=0;j<COLS;j++)
          {
#pragma HLS PIPELINE
              pixel_src=src.read(i*COLS+j);
```

```
                    dst.write(i*COLS+j,pixel_src);
                }
            }
        }
        else
        {
            for(i=0;i<ROWS;i++)
            {
                for(j=0;j<COLS;j++)
                {
    #pragma HLS PIPELINE
                    pixel_src=src.read(i*COLS+j);
                    if(pixel_src>threshold)
                    {
                        pixel_dst=0;
                    }
                    else
                    {
                        pixel_dst=255;
                    }
                    dst.write(i*COLS+j,pixel_dst);
                }
            }
        }
}
```

Top-level function section of source file

```
    void edge_detector(ustream_t &src, ustream_t &dst, ap_uint<8> threshold)
    {
    #pragma HLS DATAFLOW
    #pragma HLS INTERFACE axis port=src
    #pragma HLS INTERFACE axis port=dst
    #pragma HLS INTERFACE ap_vld port=threshold register
    #pragma HLS INTERFACE ap_ctrl_none port=return

      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> srcImg;
    #pragma HLS stream variable=srcImg.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> rgbSobel;
    #pragma HLS stream variable=rgbSobel.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> grayImg;
    #pragma HLS stream variable=grayImg.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> sobelImg_x;
    #pragma HLS stream variable=sobelImg_x.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> sobelImg_y;
    #pragma HLS stream variable=sobelImg_y.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> sobelImg;
    #pragma HLS stream variable=sobelImg.data depth=1920 dim=1
      xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS,XF_NPPC1> thresholdImg;
    #pragma HLS stream variable=thresholdImg.data depth=1920 dim=1

      xf::cv::AXIvideo2xfMat(src, srcImg);
      xfrgb2gray<IMG_MAX_ROWS,IMG_MAX_COLS>(srcImg, grayImg);
        xf::cv::xFSobelFilter3x3<XF_8UC1, XF_8UC1,IMG_MAX_ROWS, IMG_MAX_COLS, XF_CHANNELS(XF_8UC1,XF_NPPC1),
XF_DEPTH(XF_8UC1,XF_NPPC1), XF_DEPTH(XF_8UC1,XF_NPPC1),
```

```
                    XF_NPPC1, XF_WORDWIDTH(XF_8UC1,XF_NPPC1), XF_WORDWIDTH(XF_8UC1,XF_NPPC1), (IMG_MAX_COLS >>
XF_BITSHIFT(XF_NPPC1)),false>(
          grayImg,sobelImg_x,sobelImg_y,grayImg.rows,grayImg.cols>>XF_BITSHIFT(XF_NPPC1));
     AddWeightedKernel<IMG_MAX_ROWS,IMG_MAX_COLS>(sobelImg_x,0.5f,sobelImg_y,0.5f,0.0f,sobelImg);
     thresholding<IMG_MAX_ROWS,IMG_MAX_COLS>(sobelImg, thresholdImg,threshold);
     xfgray2rgb<IMG_MAX_ROWS,IMG_MAX_COLS>(thresholdImg, rgbSobel);
     xf::cv::xfMat2AXIvideo(rgbSobel, dst);
}
```

Part 11.3: Interface Synthesis and Project Optimization

Interface Synthesis

The interface settings are the same as before, the Block Level Interface is set to ap_none type, the input and output parameters are set to axi type because of continuously changing data, and the threshold setting parameter is set to ap_vld type, which is transmitted once per frame to achieve better display effect

Project Optimization:

The project optimization is similar to the previous chapter. Note that because the output result is special when the threshold value is 0, it is judged whether it is 0 outside the current loop, reducing the parameter judgment in the loop, and improving the running speed. Because the Dataflow is used, in order to avoid In the case of deadlock, we should try our best to ensure that the delay in the function is as close as possible, and the overall project running delay of about 20ms can meet the frame number requirement

Synthesis Summary Report of 'edge_detector'

General Information

Date:	Mon Nov 2 14:21:28 2020
Version:	2020.1 (Build 2902540 on Wed May 27 20:16:15 MDT 2020)
Project:	edge_detector
Solution:	solution1 (Vivado IP Flow Target)
Product family:	zynquplus
Target device:	xczu3eg-sfvc784-1-i

Performance&Resource Estimates

Modules Loops

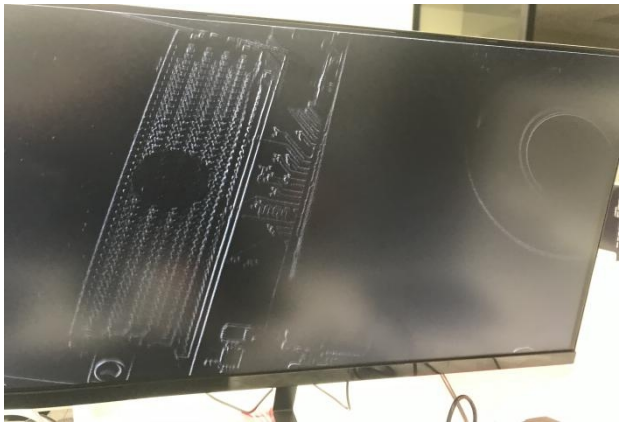
Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
edge_detector		-	-	-	-	-	dataflow	14	3	2102	3159
xFSobelFilter3x3_0_0_10		2082003	2.082E7	-	2082003	-	no	3	0	242	1104
Loop_loop_height_proc		-	-	-	-	-	no	0	0	126	350
xfrgb2gray_1080_1920_		2073607	2.074E7	-	2073607	-	no	0	3	122	176
thresholding_1080_192(		2073602	2.074E7	-	2073602	-	no	0	0	62	275
Loop_loop_height_proc		2077921	2.078E7	-	2077921	-	no	0	0	46	203
AddWeightedKernel_10		2073602	2.074E7	-	2073602	-	no	0	0	28	154

Part 11.4: Project Path

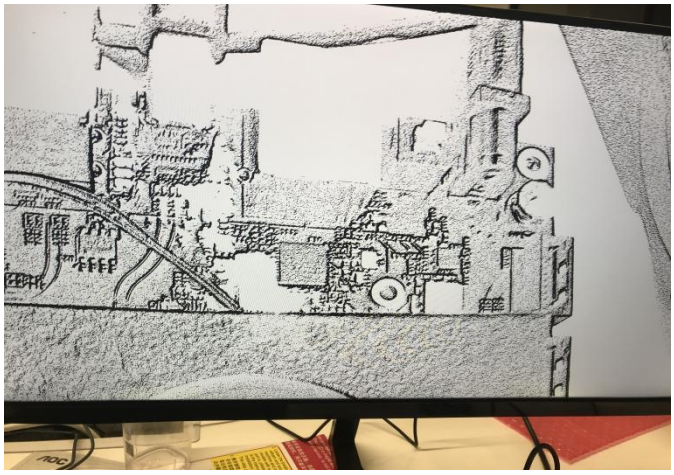
Name	Path
Vivado Project	vivado/edge_detector
HLS Project	hls/ edge_detector

Part 11.5: Project Realization

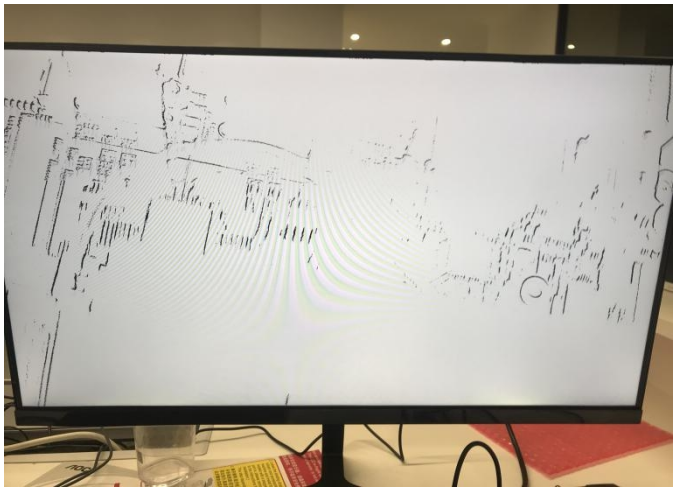
After burning, the effect will be displayed when the threshold is set to 0



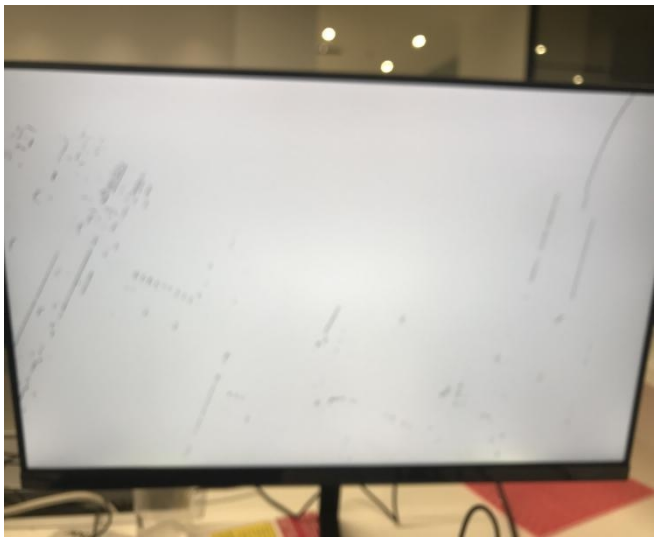
The effect is displayed when the threshold is set to 10



The effect is displayed when the threshold is set to 50



The effect is displayed when the threshold is set to 100



Part 12: Canny Operator Realizes Edge Detection

Part 12.1: Introduction to the Experiment

Compared with the Sobel operator in the previous chapter to achieve edge detection, Canny operator has strong anti-interference against noise and is more accurate for edge detection. The frame outlined by the canny operator is more detailed and can clearly show the weak edges in the picture. The operation is mainly divided into the following steps. First, perform 3x3 Gaussian filtering on the image, then perform first-order partial derivative finite difference calculation on the screen to calculate the gradient amplitude and direction, and perform non-maximum suppression to remove false edges after completion. Finally, double thresholds are used to judge the connected edges. In the experiment, the gradient amplitude and direction are calculated by Sobel. Usually, the edge points we call are points with large changes in gray value. After these points are obtained, the maximum point is searched around. Double threshold judgment means that if the target point is less than the smaller threshold, it is considered to be not an edge and removed. If If it is greater than the smaller threshold and smaller than the larger threshold, set a number between 0 and 255 for the point, which is considered a weak edge. If it is greater than the larger threshold and set it to 0, the point is painted black, and it is considered a strong edge.

Part 12.2: Experimental Source Code

Header file parameter definition

```
#ifndef __edge_canny_detector_h__
#define __edge_canny_detector_h__

#include "ap_int.h"
#include "hls_stream.h"
#include "ap_axi_sdata.h"

#include "common/xf_common.hpp"
#include "common/xf_infra.hpp"
#include "common/xf_utility.hpp"

#include "xf_canny.hpp"
#include "imgproc/xf_cvt_color.hpp"

typedef ap_axiu<24,1,1,1> pixel;
typedef hls::stream<pixel> pixel_t ;

#define IMG_MAX_ROWS 1080
#define IMG_MAX_COLS 1920
#define FILTER_TYPE 3

void edge_canny_detector(pixel_t &src,pixel_t &dst,ap_int<32> lowthreshold,ap_int<32> highthreshold);

#endif
```

The main part of the source file

```
template<int ROWS,int COLS>
void xfrgb2gray(xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1> &src,xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1> &dst)
{
  XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
  XF_TNAME(XF_8UC1,XF_NPPC1)gray_packed;
  ap_uint<8>rgb[3];
  ap_uint<8>gray;
  unsigned int i,j=0;
  for(i=0;i<ROWS;i++)
  {
        for(j=0;j<COLS;j++)
        {
#pragma HLS PIPELINE
            rgb_packed=src.read(i*COLS+j);
            ExtractPixel(rgb_packed,rgb);
            gray=CalculateGRAY(rgb[0],rgb[1],rgb[2]);
            gray_packed.range(7,0)=gray;
            dst.write(i*COLS+j,gray_packed);
        }
  }
}

template<int ROWS,int COLS>
void xfgray2rgb(xf::cv::Mat<XF_8UC1,ROWS,COLS,XF_NPPC1> &src,xf::cv::Mat<XF_8UC3,ROWS,COLS,XF_NPPC1> &dst)
{
#pragma HLS INLINE off
```

```
    XF_TNAME(XF_8UC3,XF_NPPC1)rgb_packed;
    XF_TNAME(XF_8UC1,XF_NPPC1)gray_packed;
    ap_uint<8>rgb[3];
    ap_uint<8>gray;
    unsigned i,j,k=0;
    for(i=0;i<ROWS;i++)
    {
        for(j=0;j<COLS;j++)
        {
#pragma HLS PIPELINE
            gray_packed=src.read(i*COLS+j);
            gray=gray_packed.range(7,0);
            rgb_packed.range(7,0)=gray;
            rgb_packed.range(15,8)=gray;
            rgb_packed.range(23,16)=gray;
            dst.write(i*COLS+j,rgb_packed);
        }
    }
}

template<int ROWS,int COLS>
void
duplicate(xf::cv::Mat<XF_16SC1,ROWS,COLS,XF_NPPC1>&src1,xf::cv::Mat<XF_16SC1,ROWS,COLS,XF_NPPC1>&src2,xf::cv::Ma
t<XF_16SC1,ROWS,COLS,XF_NPPC1> &dst1,xf::cv::Mat<XF_16SC1,ROWS,COLS,XF_NPPC1>
&dst2,xf::cv::Mat<XF_16SC1,ROWS,COLS,XF_NPPC1> &dst3,xf::cv::Mat<XF_16SC1,ROWS,COLS,XF_NPPC1> &dst4)
{
    XF_TNAME(XF_16SC1,XF_NPPC1)pixel_src1;
    XF_TNAME(XF_16SC1,XF_NPPC1)pixel_src2;
    unsigned int i,j=0;
    for(i=0;i<ROWS;i++)
    {
        for(j=0;j<COLS;j++)
        {
#pragma HLS PIPELINE
            pixel_src1=src1.read(i*COLS+j);
            pixel_src2=src2.read(i*COLS+j);
            dst1.write(i*COLS+j,pixel_src1);
            dst2.write(i*COLS+j,pixel_src1);
            dst3.write(i*COLS+j,pixel_src2);
            dst4.write(i*COLS+j,pixel_src2);
        }
    }
}
```

Top-level function section of source file

```
void edge_canny_detector(pixel_t &src,pixel_t &dst,ap_int<8>&lowthreshold,ap_int<8>&highthreshold)
{

#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_vld port=highthreshold register
#pragma HLS INTERFACE ap_vld port=lowthreshold register
#pragma HLS INTERFACE axis port=dst register_mode=both depth=16 register
#pragma HLS INTERFACE axis port=src register_mode=both depth=16 register
#pragma HLS DATAFLOW
```

```
    xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>rgb_img_src;
  #pragma HLS STREAM variable=rgb_img_src.data depth=1920 dim=1
    xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>gray_img_src;
  #pragma HLS STREAM variable=gray_img_src.data depth=1920 dim=1
    xf::cv::Mat<XF_8UC1,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>gray_img_dst;
  #pragma HLS STREAM variable=gray_img_dst.data depth=1920 dim=1
    xf::cv::Mat<XF_8UC3,IMG_MAX_ROWS,IMG_MAX_COLS,XF_NPPC1>rgb_img_dst;
  #pragma HLS STREAM variable=rgb_img_dst.data depth=1920 dim=1

    int img_height=rgb_img_src.rows;
    int img_width=rgb_img_src.cols;
      xf::cv::Mat<XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> gaussian_mat(img_height, img_width);
  #pragma HLS STREAM variable=gaussian_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> gradx_mat(img_height, img_width);
  #pragma HLS STREAM variable=gradx_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> gradx1_mat(img_height, img_width);
  #pragma HLS STREAM variable=gradx1_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> gradx2_mat(img_height, img_width);
  #pragma HLS STREAM variable=gradx2_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1,IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> grady_mat(img_height, img_width);
  #pragma HLS STREAM variable=grady_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> grady1_mat(img_height, img_width);
  #pragma HLS STREAM variable=grady1_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> grady2_mat(img_height, img_width);
  #pragma HLS STREAM variable=grady2_mat.data depth=1920 dim=1
      xf::cv::Mat<XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1, 1920*3> magnitude_mat(img_height, img_width);
  #pragma HLS STREAM variable=magnitude_mat.data depth=5760 dim=1
      xf::cv::Mat<XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1, 1920*3> phase_mat(img_height, img_width);
  #pragma HLS STREAM variable=phase_mat.data depth=5760 dim=1
      xf::cv::Mat<XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1> nms_mat(img_height, img_width);
  #pragma HLS STREAM variable=nms_mat.data depth=1920 dim=1

  // clang-format off
  // clang-format on

  // clang-format off

      // clang-format on

    xf::cv::AXIvideo2xfMat(src,rgb_img_src);
    xfrgb2gray<IMG_MAX_ROWS,IMG_MAX_COLS>(rgb_img_src,gray_img_src);
      xFAverageGaussianMask3x3<XF_8UC1, XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_DEPTH(XF_8UC1,XF_NPPC1),
XF_NPPC1, XF_WORDWIDTH(XF_8UC1,XF_NPPC1), (IMG_MAX_COLS >> XF_BITSHIFT(XF_NPPC1))>(
            gray_img_src, gaussian_mat, img_height, img_width);
      xFSobel<XF_8UC1, XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_DEPTH(XF_8UC1,XF_NPPC1),
XF_DEPTH(XF_16SC1,XF_NPPC1),XF_NPPC1,XF_WORDWIDTH(XF_8UC1,XF_NPPC1), XF_WORDWIDTH(XF_16SC1,XF_NPPC1), 3, false>(
          gaussian_mat, gradx_mat, grady_mat, XF_BORDER_REPLICATE, img_height, img_width);
      duplicate<IMG_MAX_ROWS,IMG_MAX_COLS>(gradx_mat,grady_mat,gradx1_mat,gradx2_mat,grady1_mat,grady2_mat);
      xf::cv::magnitude<1, XF_16SC1,XF_16SC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_NPPC1, 1920*3>(gradx1_mat,
grady1_mat, magnitude_mat);
      xFAngle<XF_16SC1, XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_16SP, XF_8UP, XF_NPPC1, XF_16UW, XF_8UW, 1920*3>(
           gradx2_mat, grady2_mat, phase_mat, img_height, img_width);
      xFSuppression3x3<XF_16SC1, XF_8UC1, XF_8UC1, IMG_MAX_ROWS, IMG_MAX_COLS, XF_16SP, XF_8UP, XF_8UP, XF_NPPC1,
XF_16UW, XF_8UW, XF_8UW, (IMG_MAX_COLS >> XF_BITSHIFT(XF_NPPC1)), 1920*3, 1920*3> (magnitude_mat, phase_mat, nms_mat,
lowthreshold,highthreshold, img_height, img_width);
  //    nPackNMS<IMG_MAX_ROWS,IMG_MAX_COLS>(nms_mat,gray_img_dst);
```

```
    //    AddWeightedKernel<IMG_MAX_ROWS,IMG_MAX_COLS>(gradx_mat,0.5f,grady_mat,0.5f,0.0f,gray_img_dst);
      xfgray2rgb<IMG_MAX_ROWS,IMG_MAX_COLS>(nms_mat,rgb_img_dst);
      xf::cv::xfMat2AXIvideo(rgb_img_dst,dst);
}
```

Part 12.3: Interface Synthesis and Project Optimization

Interface Synthesis

The interface settings are the same as the previous chapters. The Block Level Interface is set to ap_ctrl_hs to eliminate redundant associated signals. The input and output parameters are set to axi type interfaces because of continuously changing data, and low_threshold and high_threshold are set to ap_vld type interfaces. The valid signal is connected to the frame start signal. If the threshold is sent during the operation of a frame, the display of the upper and lower edges of the screen will be inconsistent.

Project Optimization:

The overall optimization method is similar to the previous chapter. Using Dataflow to execute the function, the screen color bar appears when running the xf_canny function that comes with the xfopencv library, and the hpp file related to the xf_canny function is modified. The function in the xf_canny.hpp file used in the program is a rewritten file, put it in the source folder, and three folders have been modified accordingly. Verify that the display effect is normal, and the overall function running time is about 20ms to meet the frame number requirement:

Synthesis Summary(solution1)

Synthesis Summary Report of 'edge_canny_detector'

General Information

Date:	Mon Nov 2 13:48:47 2020
Version:	2020.1 (Build 2902540 on Wed May 27 20:16:15 MDT 2020)
Project:	edge_canny_detector
Solution:	solution1 (Vivado IP Flow Target)
Product family:	zynquplus
Target device:	xczu3eg-sfvc784-1-i

Performance&Resource Estimates

Modules Loops

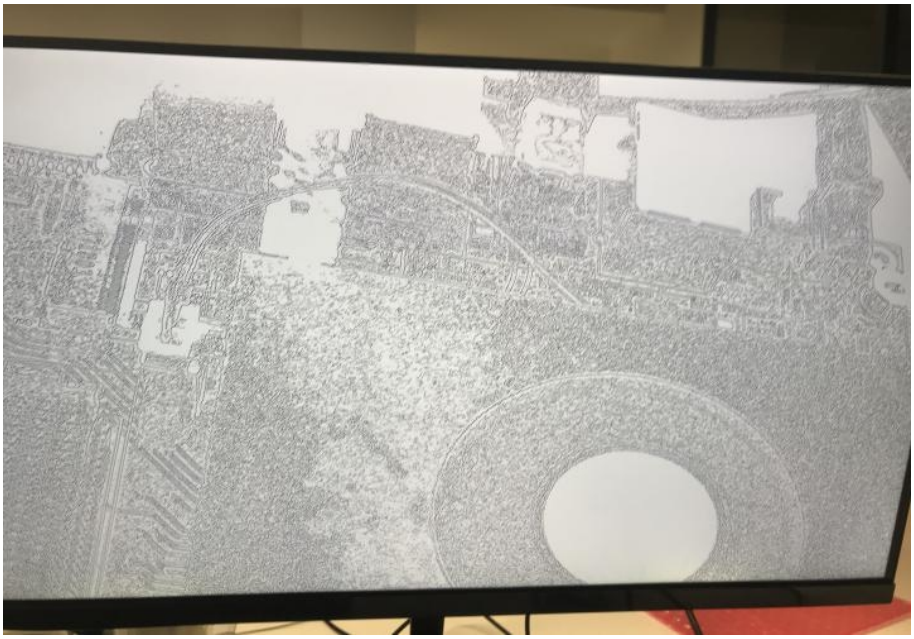
Modules && Loops	Issue Type	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
edge_canny_detector		-	-		-	-	dataflow	46	6	4969	8076
xFMagnitudeKernel_2_2		2085481	2.086E7	-	2085481	-	no	0	2	1043	2266
xFSuppression3x3_2_0_(		2082003	2.082E7	-	2082003	-	no	8	0	378	1227
xFSobel_0_2_1080_1920		2088859	2.089E7	-	2088859	-	no	3	0	425	1218
xFAverageGaussianMas		2082003	2.082E7	-	2082003	-	no	3	0	291	949
xFAngleKernel_2_0_108(		2077921	2.078E7	-	2077921	-	no	0	1	163	380
Loop_loop_height_proc		-	-	-	-	-	no	0	0	126	386
xfrgb2gray_1080_1920_		2073607	2.074E7	-	2073607	-	no	0	3	122	176
Loop_loop_height_proc		2077921	2.078E7	-	2077921	-	no	0	0	46	203
duplicate_1080_1920_s		2073602	2.074E7	-	2073602	-	no	0	0	29	175
xfgray2rgb_1080_1920_		2073602	2.074E7	-	2073602	-	no	0	0	29	139

Part 12.4: Project Path

Name	Path
Vivado Project	vivado/edge_canny
HLS Project	hls/ edge_canny_detector

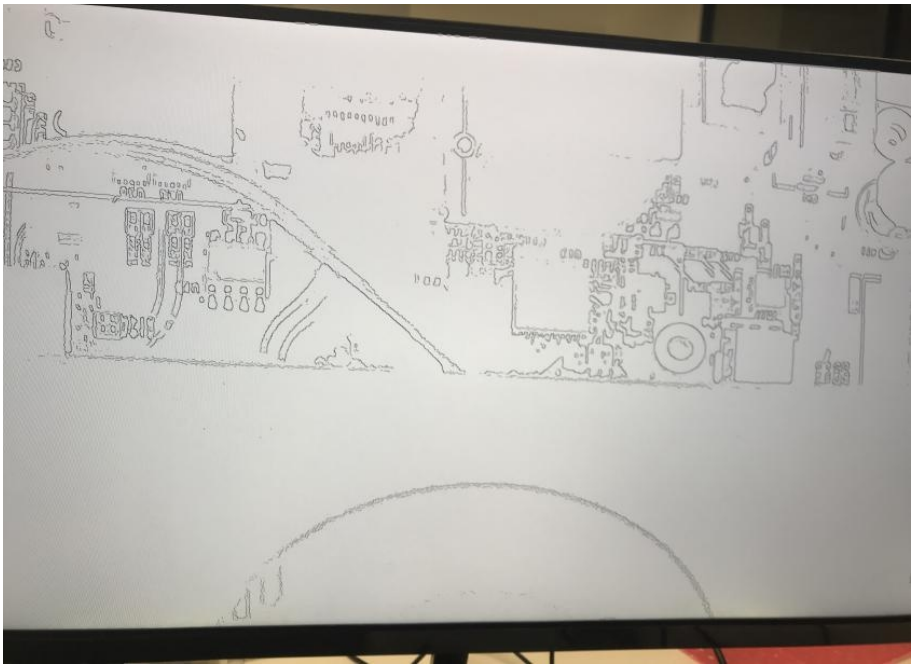
Part 12.5: Project Realization

When the low threshold is set to 10, and the high threshold is set to 20, the screen will be displayed



When the low threshold is set to 20, and the high threshold is set to 30, the screen will be displayed

When the low threshold is set to 50, and the high threshold is set to 100, the screen will be displayed

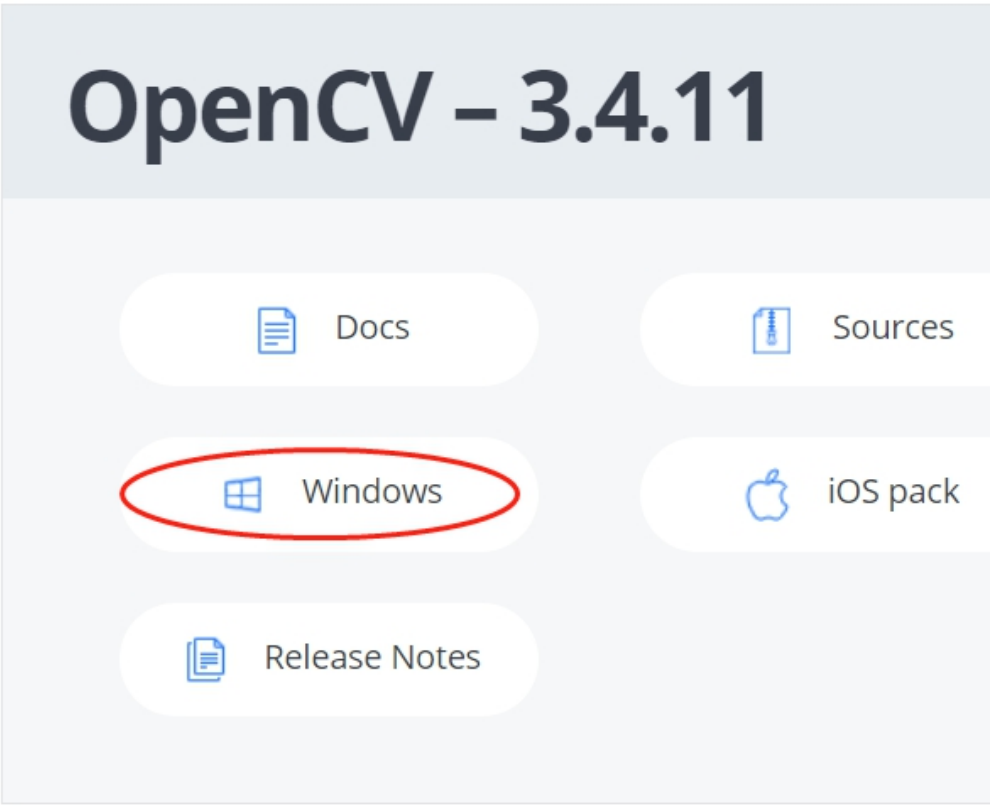


Part 13: How to Implement opencv Simulation with vitis HLS

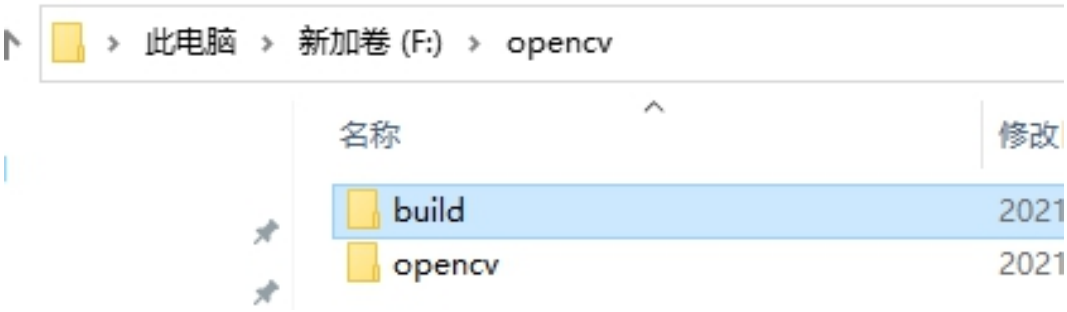
Part 13.1: Build opencv Simulation Environment

In the version above 2020.1, Xilinx removed the opencv library function file, and the simulation of the image is very different from the previous version. If you want to perform image simulation, you need to manually build an opencv simulation environment and install the corresponding library functions, otherwise it will not be able to compile. Although the functions can be implemented on the board without affecting the synthesis, the problem of inability to simulate will cause great inconvenience to the development. We build the opencv simulation environment according to the following steps

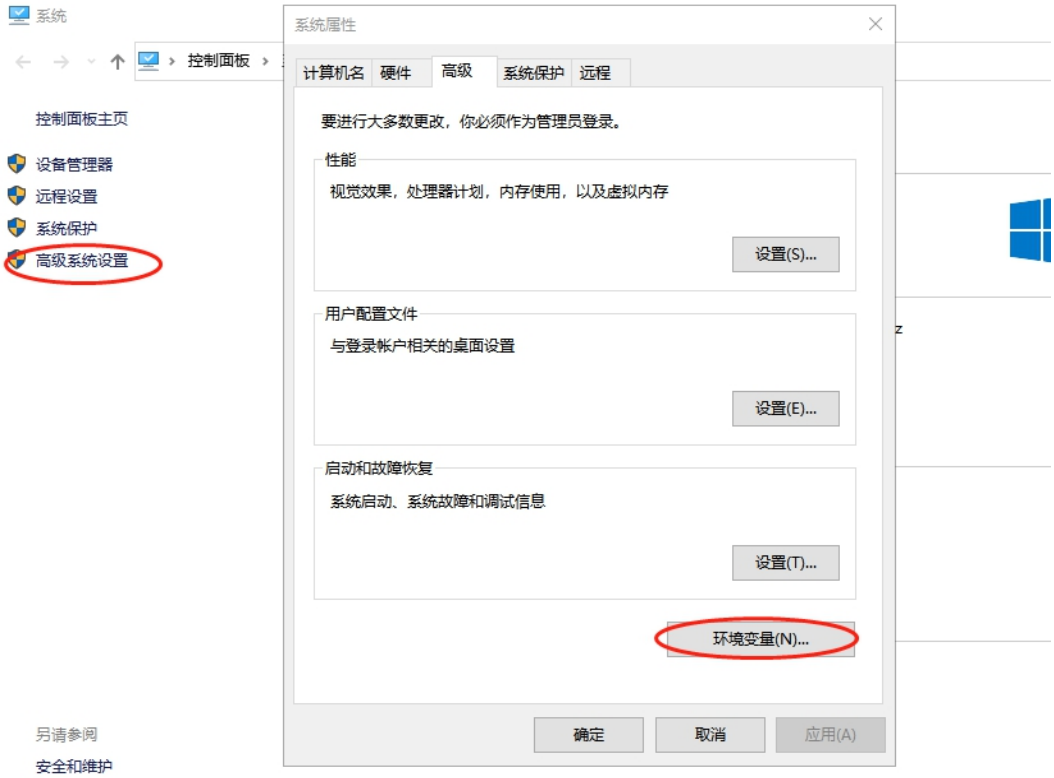
1) Log in to opencv official website: https://opencv.org/releases/ Install opencv3.4.11



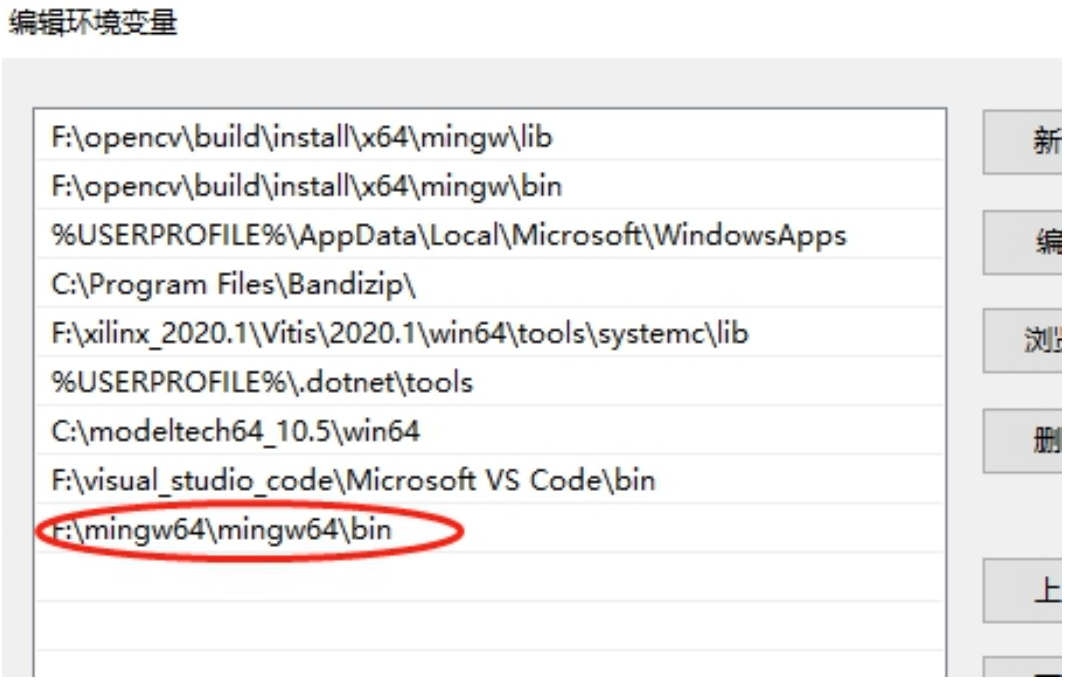
2) Select windows, download the windows version, create a new folder after downloading, unzip the file in it, and create a new build file in the newly created folder



3) Configure environment variables after completion, right click on my computer properties -> advanced system settings -> environment variables



4) Add the installed “mingGW-GW64” path to the path

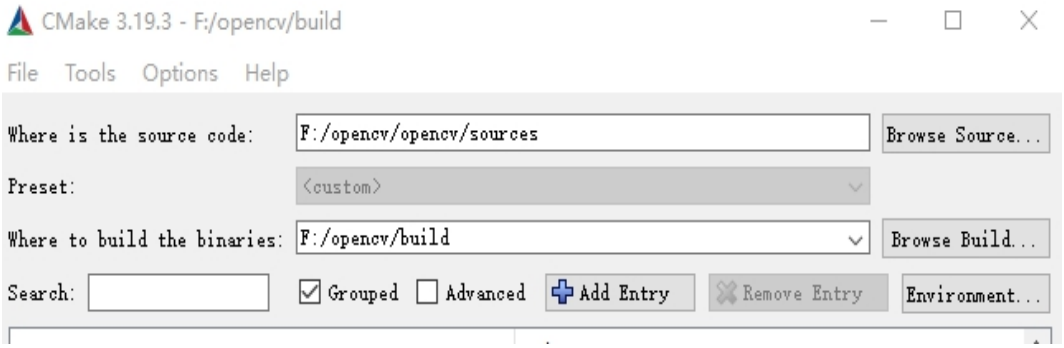


5) Open “cmd”, enter “gcc -v”, the following screen is displayed, indicating that the configuration is successful

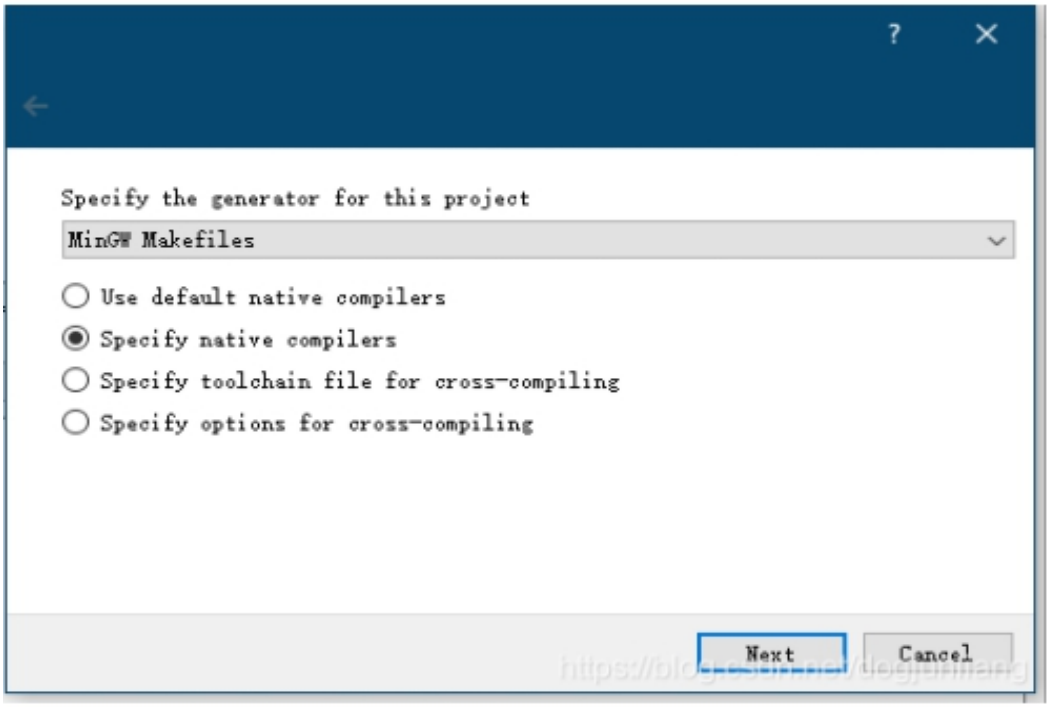
6) Download and install the Cmake URL as follows:

https://cmake.org/download/, the options during the installation process do not need to be modified

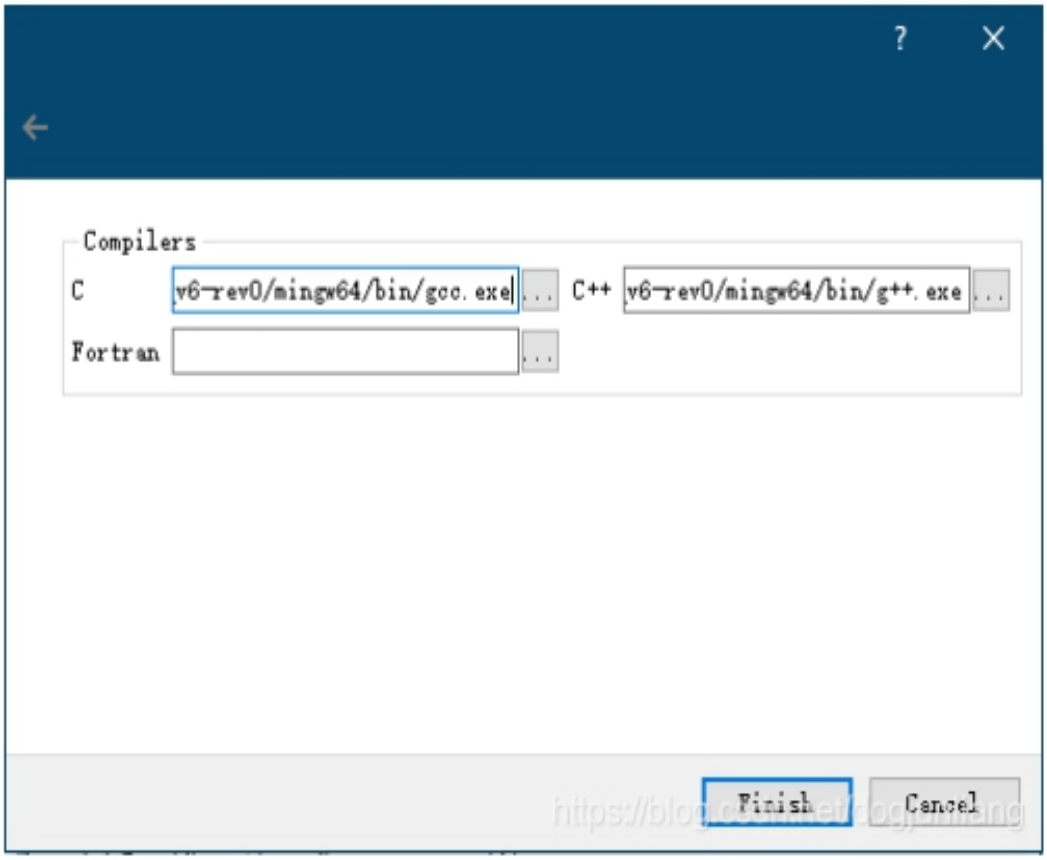
7) After completion, open cmake-gui, select the source folder path of the installed opencv in where is the source code, and select the build folder path created in the first step in where to build the binary:



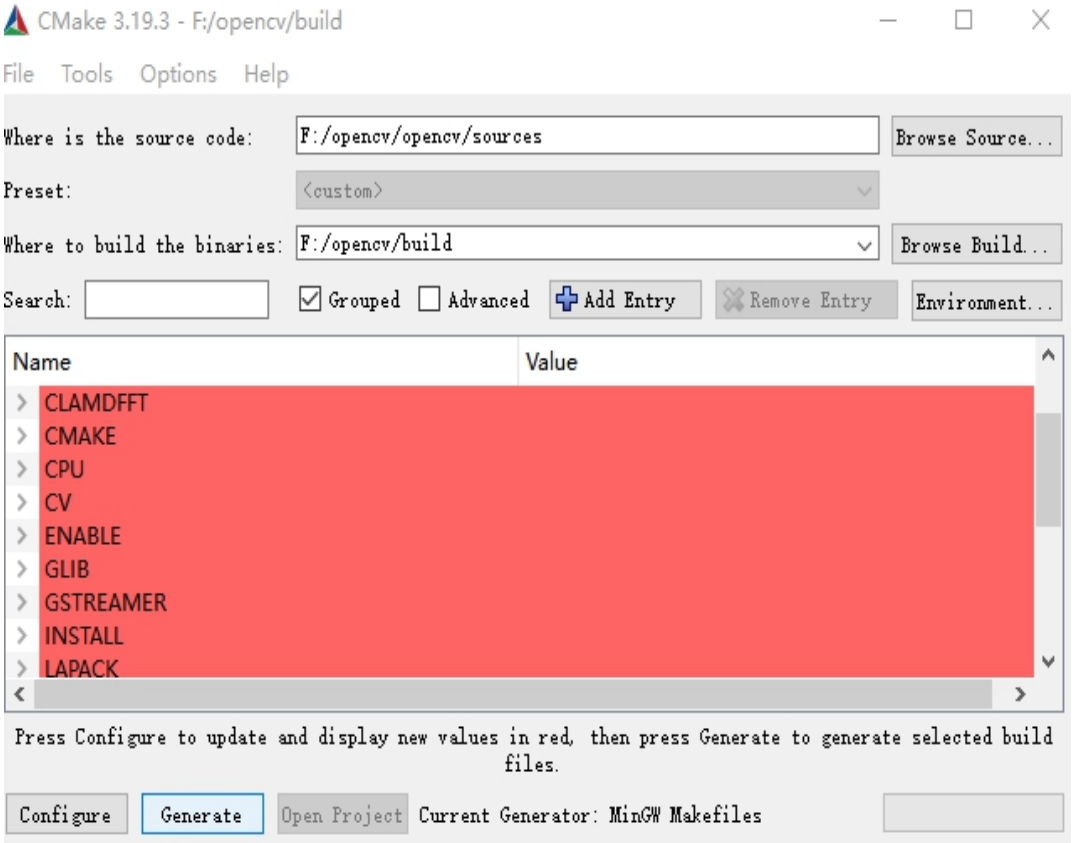
8) Click configure and set as shown in the figure below



The compiler selects the compiler in the installed mingGW-GW-64



9) Click configure, you will see the following interface



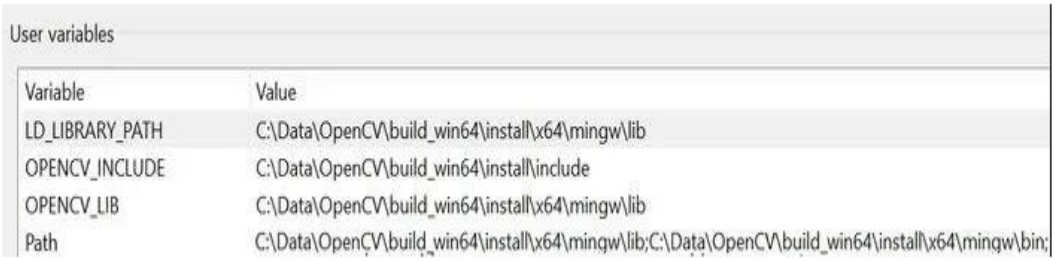
Check "ENABLE_CXX11", check "WITH_OPENGL", do not check "OPENCV_ENABLE_ALLOCATOR_STATS", and keep the rest as default

10) Click "configure", click "Generate" after "config done" appears, close "cmake-gui" after "generate done" appears

11) Right-click the start menu, enter "cmd", enter "cd +" folder path under "windows", enter the "build" folder created in the first step, and enter "mingw32-make" to start compiling.

12) When the compilation is complete, enter "mingw32-make install", and then an "install" folder will be generated in the "build" folder.

13) Close "cmd", follow the steps in step 4 to open the environment variables, create three new environment variables, add two paths to the "path", the configuration method is as shown in the figure:



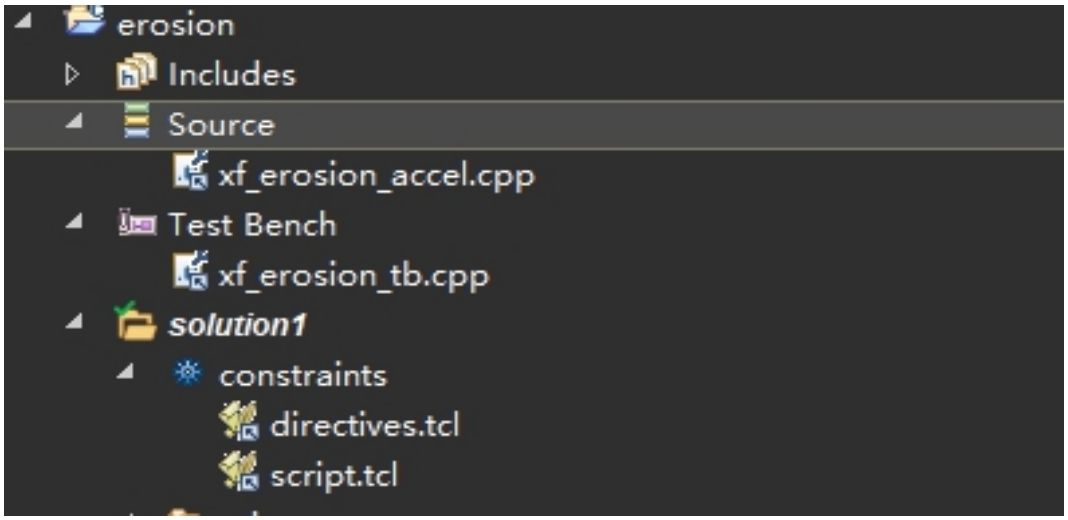
14) Restart the computer, at this time our "opencv" environment has been set up.

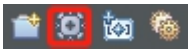
Part 13.2: Vitis HLS Configuration

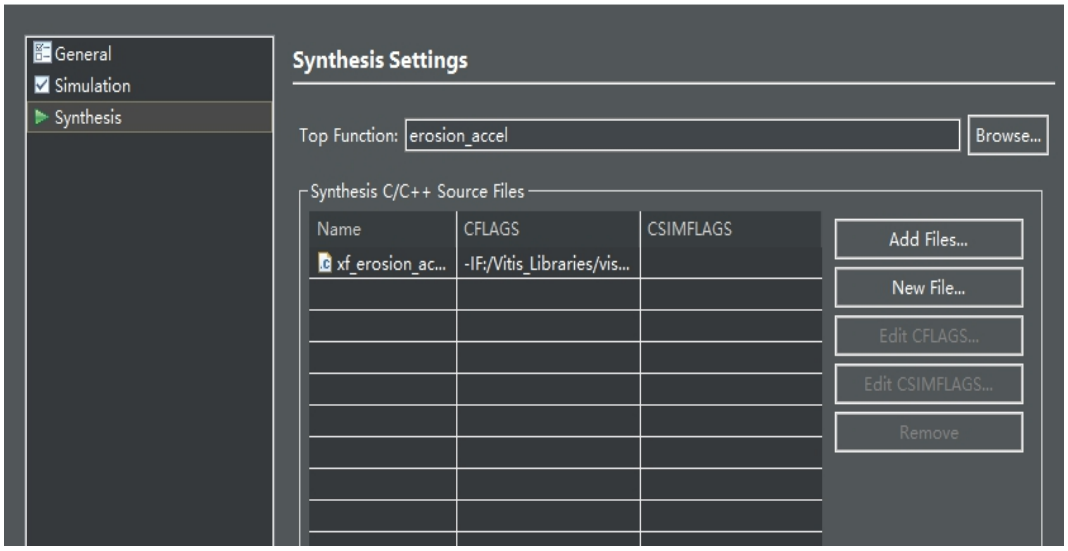
Vitis hls is configured as follows:

1) Create a new project and name it erosion. Follow the steps described in 1.1. After completion, copy the files in the erosion folder in the opencv library function compression package to the source folder, and copy xf_config_param.h in the build folder to the outside. Then add the xf_erosion_accel.cpp file to the source, and then add xf_erosion_tb.cpp to the test bench, the final effect is

shown in the figure:



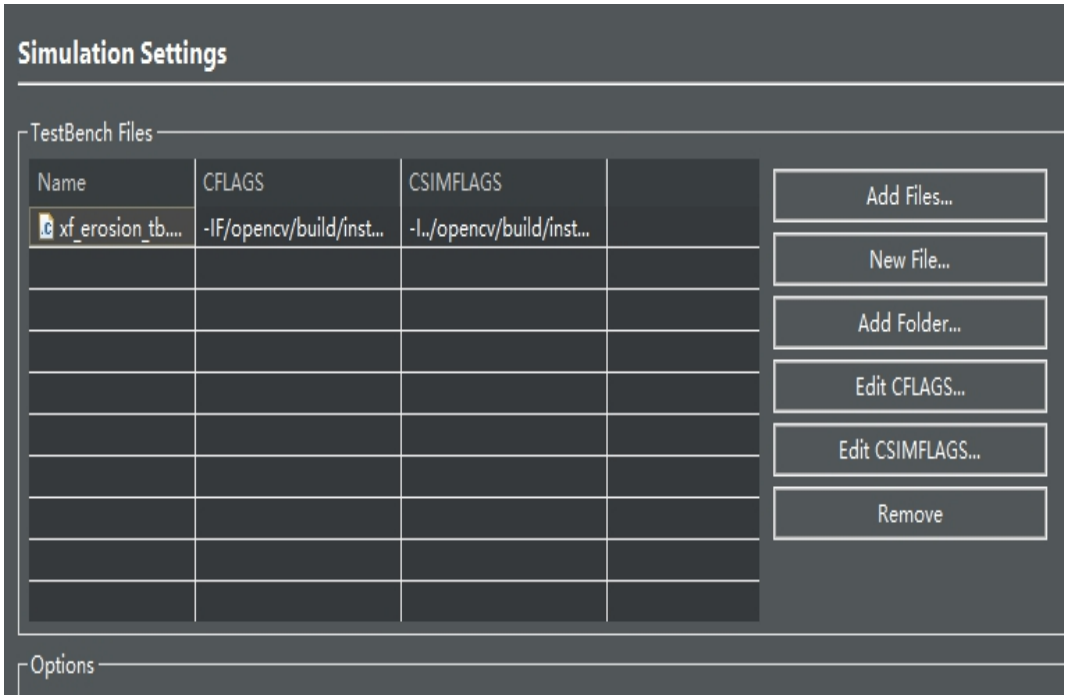
2) Click project setting  , select "synthesis", select "edit_cflags" and enter "-I <get_path> -std=c++0x"



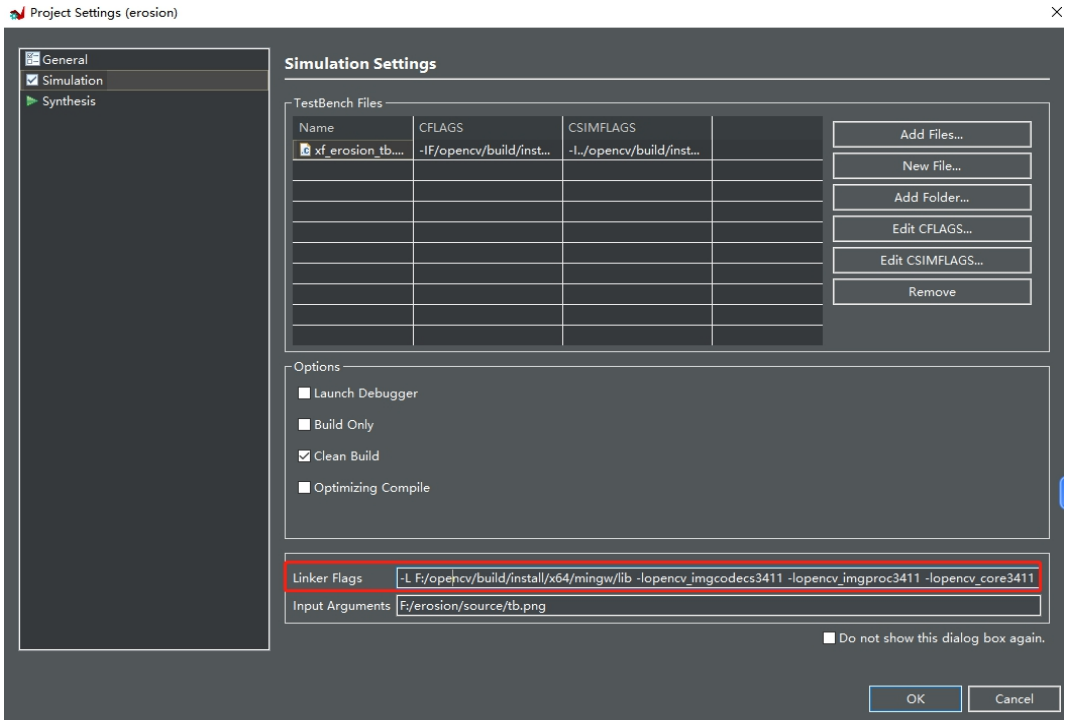
Enter the "opencv" library function path provided by Xilinx installed on the computer in "get_path"

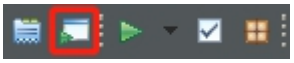
3) Select "simulation", select "edit_csimflags" and enter "-I <get_path_opencv> -I <get_path> -std=c++0x", in the "get_path_opencv" enter the opencv library function installation path downloaded from the official opencv. Note that it is the "include" folder path in the install folder, "get_path" Enter the opencv path provided by Xilinx in the installation (the opencv

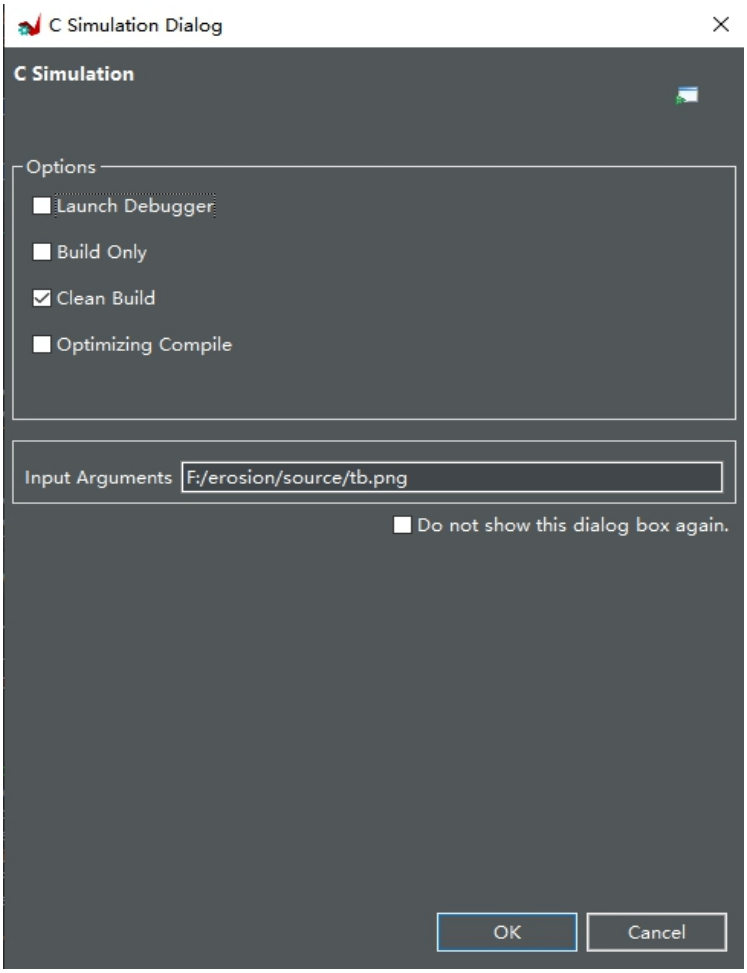
library functions provided by Xilinx are synthesizable, and the functions provided by opencv can only be used for simulation, and there is a big difference between the two)



Enter the following code in the Linker flags "-L F:/opencv/build/install/x64/mingw/lib -lopencv_imgcodecs3411 -lopencv_imgproc3411 -lopencv_core3411 -lopencv_highgui3411 -lopencv_flann3411 -lopencv_features2d3411" (3411 refers to the opencv version, if you install other versions, you need to make corresponding changes)

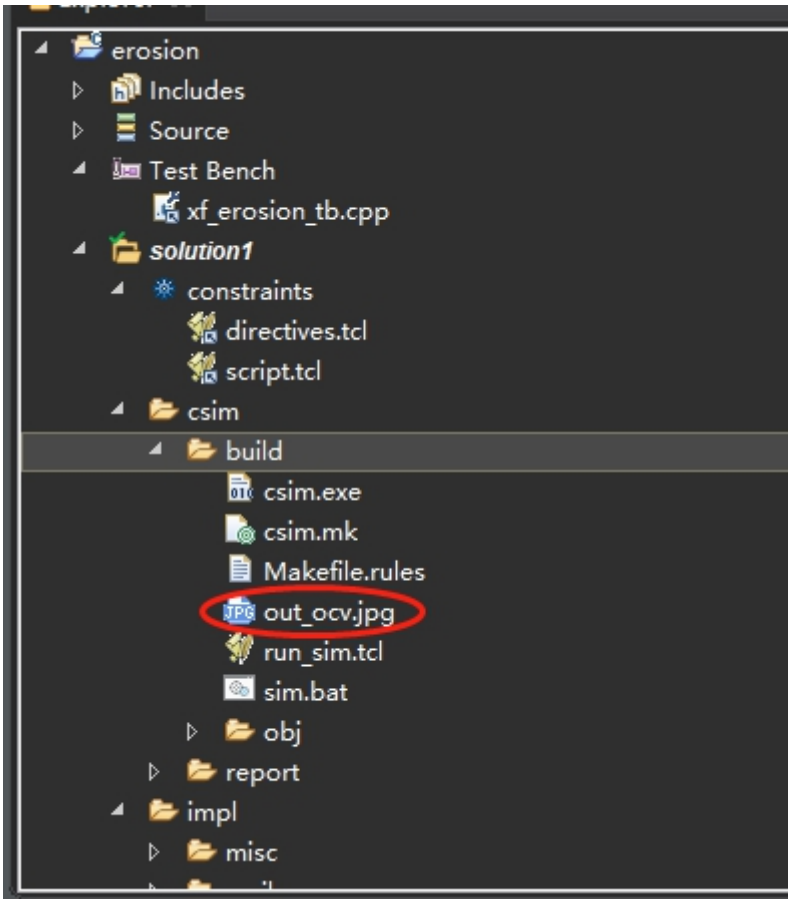


4) Click the icon  and enter the path of the simulation image in "input arguments"

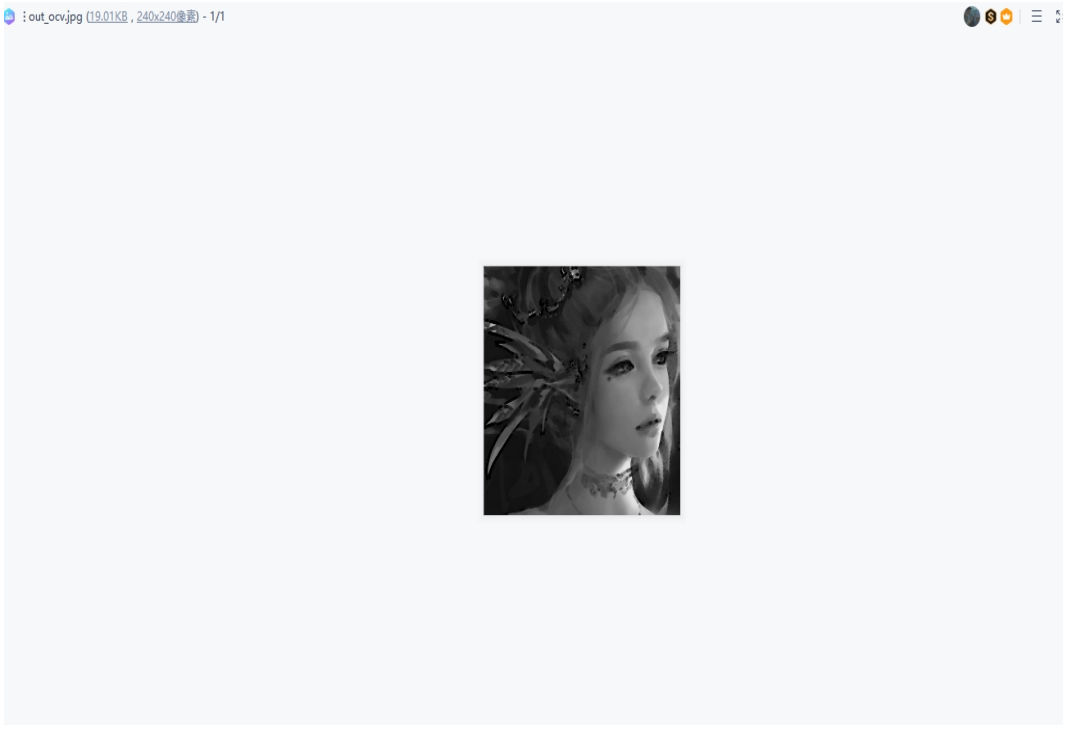


Note that the image resolution should be the same as the image resolution set in the code.

5) After passing the simulation, double-click to open the "out_ocv.jpg" picture on the left



6) Observe the experimental result as below



Part 13.3: Remarks

We provide a project named erosion as a simulation example. After downloading, change the path to the path that you installed and the path of the picture to the path of the picture you downloaded. Then select c simulation to see the effect after running. If there is an error in operation, check whether the path is mismatched and whether opencv is installed correctly.