EMBEDEEP
壹 普 智 慧

EEP-TPU FPGA IP evaluation User manual

eep-ug007 (v0.1.0)

2022-11-14-

www.embedeep.com

Revision history:

Version	Date	Describe	Author
0.1.0	2022-11-14	Initial version	Zhou

Contents

1. Overview

EEP TPU is an artificial intelligence tensor processor independently developed by Xiamen EMBEDEEP Technology Co., Ltd. (EMBEDEEP). The EEP-TPU tensor processor has many innovative features and adopts a data flow computing architecture, which has the following advantages:

(1) It can provide efficient, high-performance, and flexible data computing for neural network inference;

(2) It can provide FP16 and INT8 precision calculation and supports mixed precision calculation mode;

(3) It can support SoC mode or standalone mode;

(4) It has no dependence on the performance of the main processor and the operating system environment, and can adapt to a variety of systems in many scenarios;

(5) With the self-developed heterogeneous computing software system and compilation optimization development environment, it can support a variety of mainstream deep learning frameworks, including CAFFE, Darknet, Pytorch, ONNX, NCNN, etc.;

(6) It can support a variety of image algorithms such as classification, detection, and segmentation, including ResNet, Inception, MobileNet, SqueezeNet, YOLOV3, YOLOV4, YOLOV5, SSD, ICNet, etc.

In order to facilitate users to have a more direct performance evaluation and use of EEP TPU IP, we provide an evaluation version of the IP based on Xilinx zynqMPSoC FPGA for free (TPU can work continuously for 1~2 hours). Users can use the evaluation version of the IP based on the provided design demo, quickly build FPGA engineering with EEP TPU, and combine the corresponding bare metal programs to intuitively experience how to use EEP TPU in neural network inference applications.

2. Description of IP Project Package

In the evaluation IP engineering package, we provide the Vivado project based on the ZU15EG FPGA development board. The package includes script files with EEP TPU IP Vivado projects, FPGA pin constraint files, evaluation IP, and IP-matching compilers. This project can be used to run the bare-metal program for testing TPU functionality. For a more detailed introduction, please refer to the “excel” document in the project directory named “Directory Description.xlsx”.

Note: The project is based on the MZU15A FPGA development board from Milianke. If you switch to other ZynqMP SoC FPGAs, you need to modify the corresponding FPGA pin constraints and zynqMP IP settings, and the methods of EEP-TPU IP calling and connection are the same.

3. Vivado project construction

Figure 3.1 shows a block diagram of an SoC system for TPU IP testing, built in the Vivado IPI tool. The system contains:

(1) DVP signal receiving IP (EEP_DVP_Top), which can collect DVP camera data and store it in DDR;

(2) TPU IP (EEP_TPU), which can perform neural network inference calculations;

(3) Zynq MP, which is used for system program operation and other work.

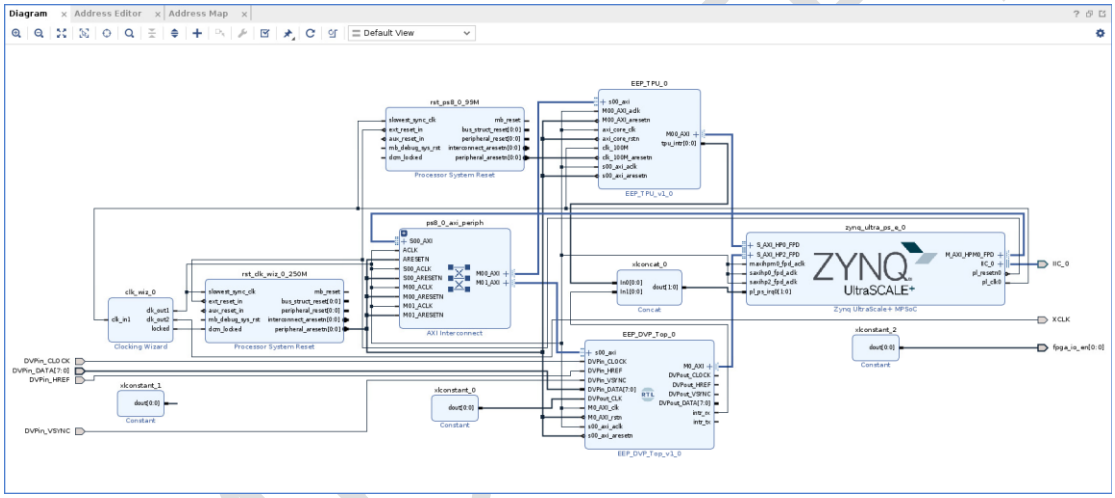

Fig. 3.1

3.1 Linux system

Under the Linux system, users can open the terminal and enter the IP project directory. In the “script” directory, the user can modify the installation path of [Vivado install path] vivado in the “create_prj.sh”, and then execute the command:

```
$ ./create_prj.sh system_rtl_MZU15A_TPU_IP_DVP_DP_v202101.tcl
```

The script will automatically generate vivado projects, perform comprehensive implementations, and generate BIT and related XSA files.

3.2 Windows system

Under the Windows system, please open the Vivado tool and execute the command in the "tcl console":

source system_rtl_MZU15A_TPU_IP_DVP_DP_v202101.tcl

Note: At the beginning of the "tcl" file, there are variables "PRJNAME", "PRJPATH", "IP_PATH", "constr_PATH". You need to modify the corresponding path name according to the actual situation.

After executing the above command, the vivado project and related bit files will also be generated.

4. Bare metal test of TPU IP

4.1 Hardware connection

Based on the Milianke MZU15A development kit, the hardware connection is shown in Figure 4.1.

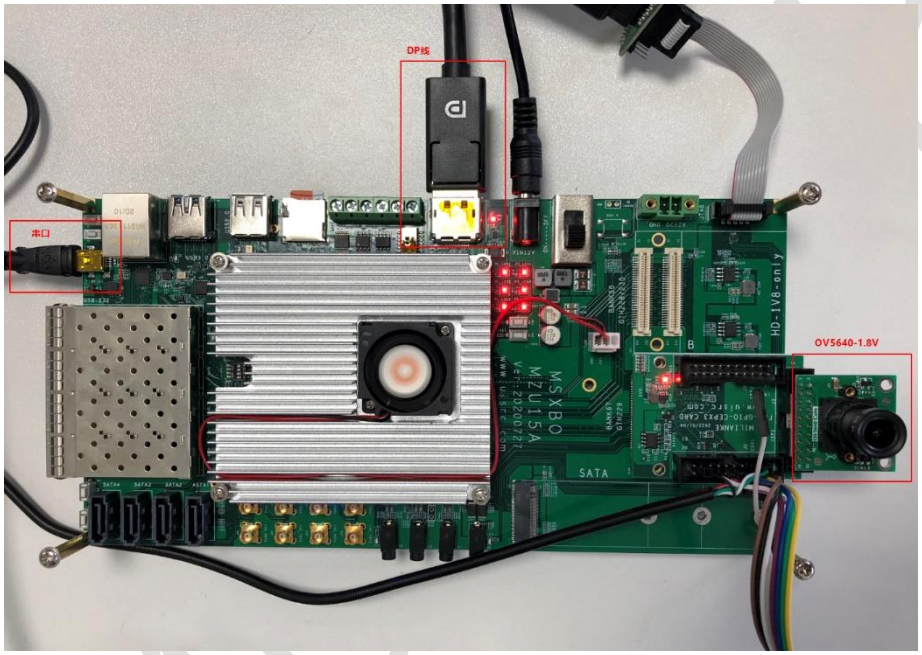

Figure 4.1 Hardware connection

4.2 Vitis engineering construction

Open “Xilinx Vitis 2021.1” and select the folder where the “xsa” file is located as the “workspace”.

Note: The “xsa” file is located in the hardware/xsa directory.

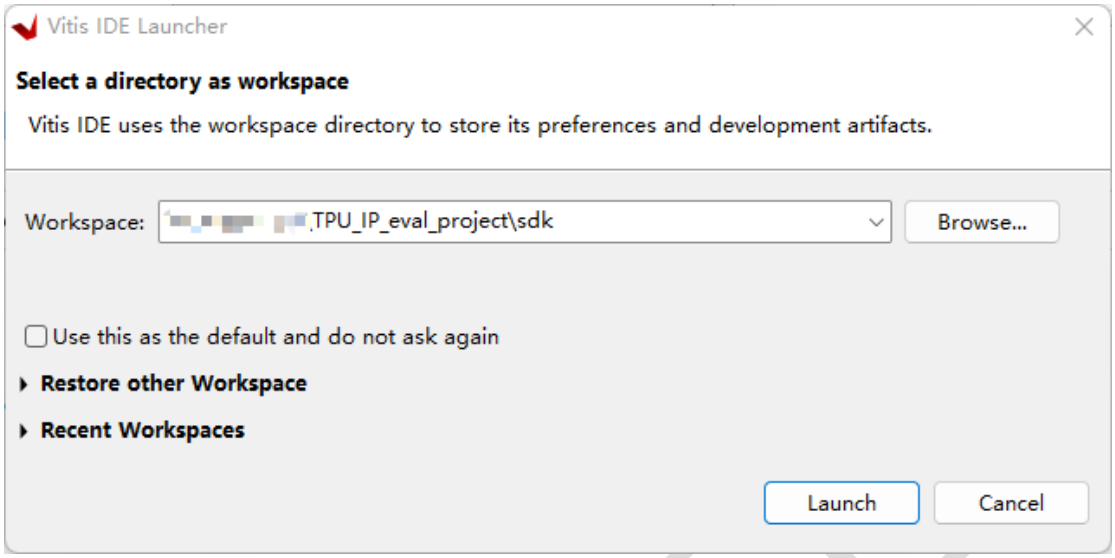

Figure 4.2 Construction flow chart 1

File->New->Application Project...->Next, select the “xsa” file.

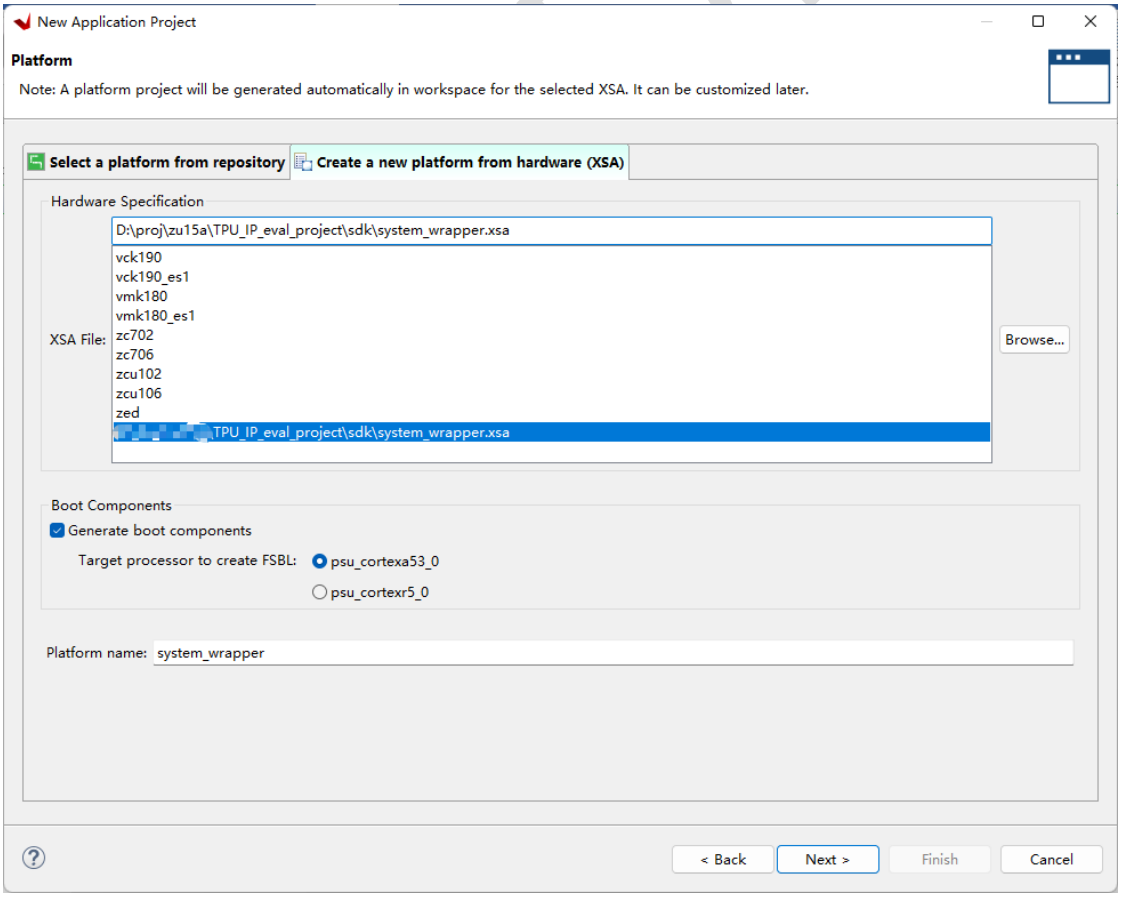

Figure 4.3 Construction flow chart 2

Enter the project name, Next->Next, create an empty C++ project, and complete the project construction.

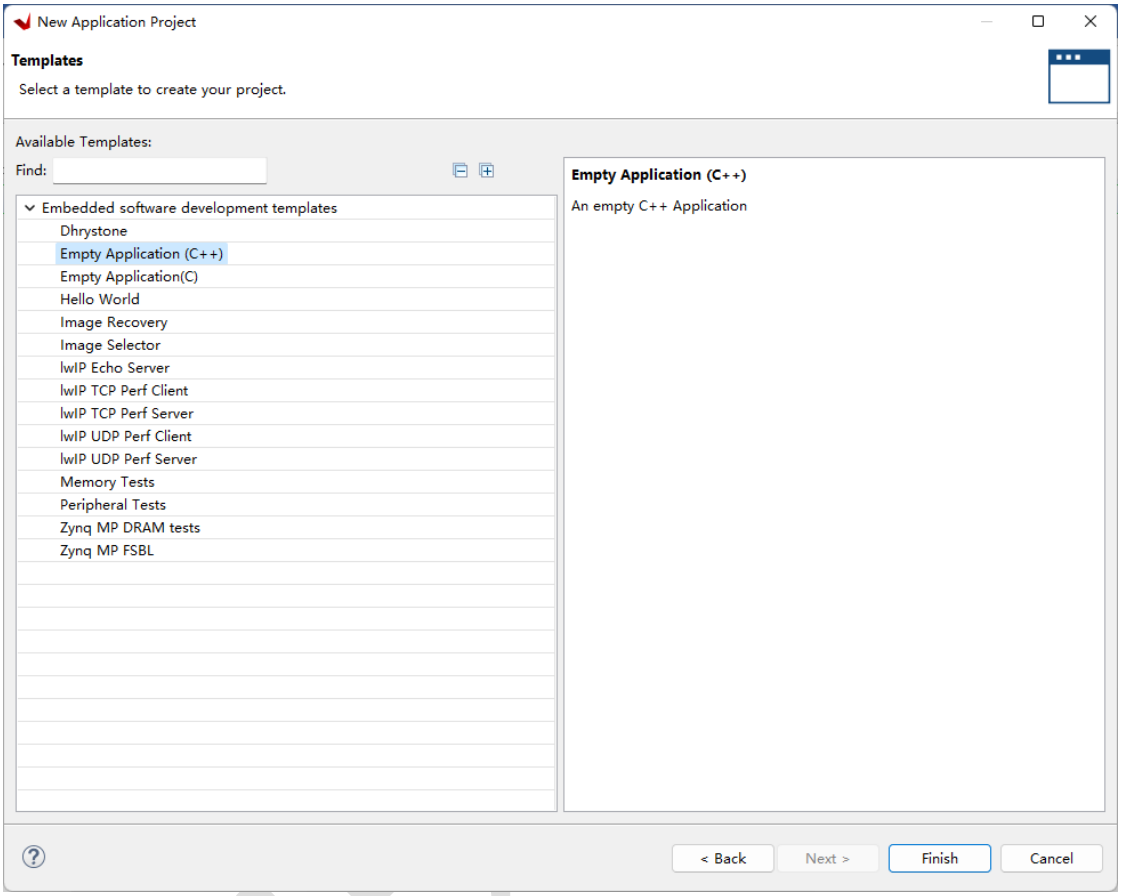

Figure 4.4 Construction flow chart 3

(1) Add “XILFFS” file system support

Select “platform.sqr” to modify the settings of the “BSP”, add system support for the “xilffs” file, and then recompile the “system_wrapper”.

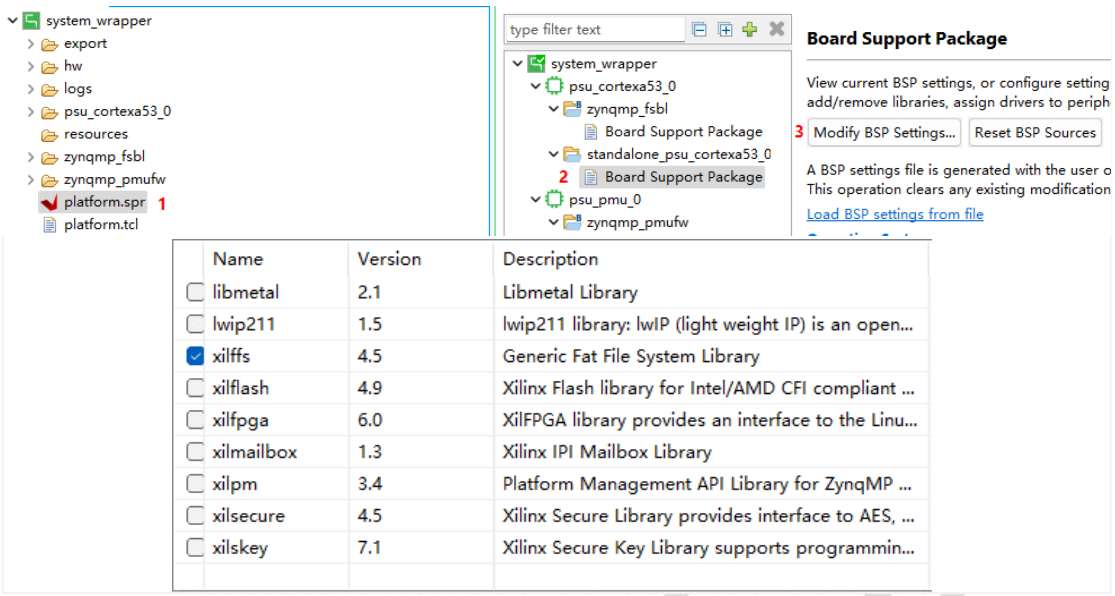

Figure 4.5 Xilffs configuration

(2) Set the heap size

Right-click the “Generate Linker Script” button to set the heap size to 10000 KB. The actual size can be adjusted according to the application.

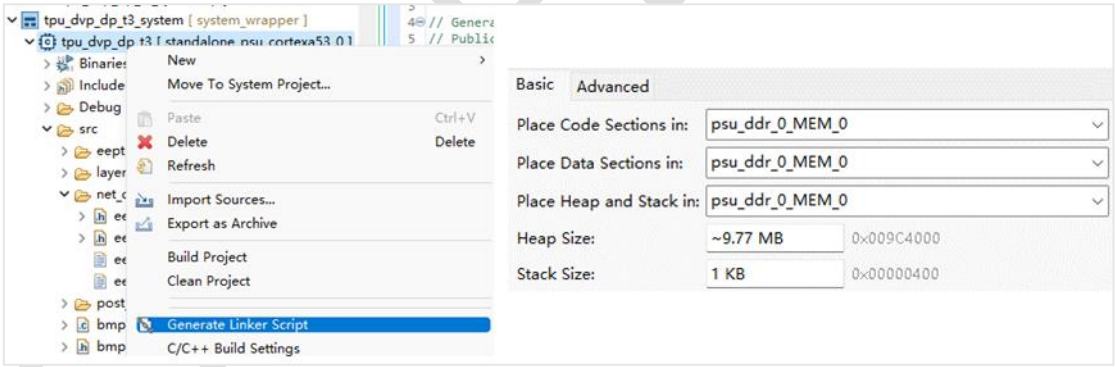

Figure 4.6 Heap Settings

Copy the files in the “sdk/standalone/src” folder to the “src” folder of the built sdk project to complete the construction of the project.

4.3 Demonstration of TPU test results

The loading of the program can be downloaded by JTAG or written to the SD card with our pre-compiled “BOOT .bin”, and the FPGA can be loaded through the SD card boot.

(1) JTAG download

Copy the “eepnet.mem” and “eepinput.mem” files to the SD card, and set the boot mode to “JTAG”, that is, the mode switch is set to “ON ON ON ON”. Right-click the project > Debug As-> Launch Hardware (Single Application Debug) in Xilinx Vitis, complete the download of bit and elf files, and run the program in the Debug interface.

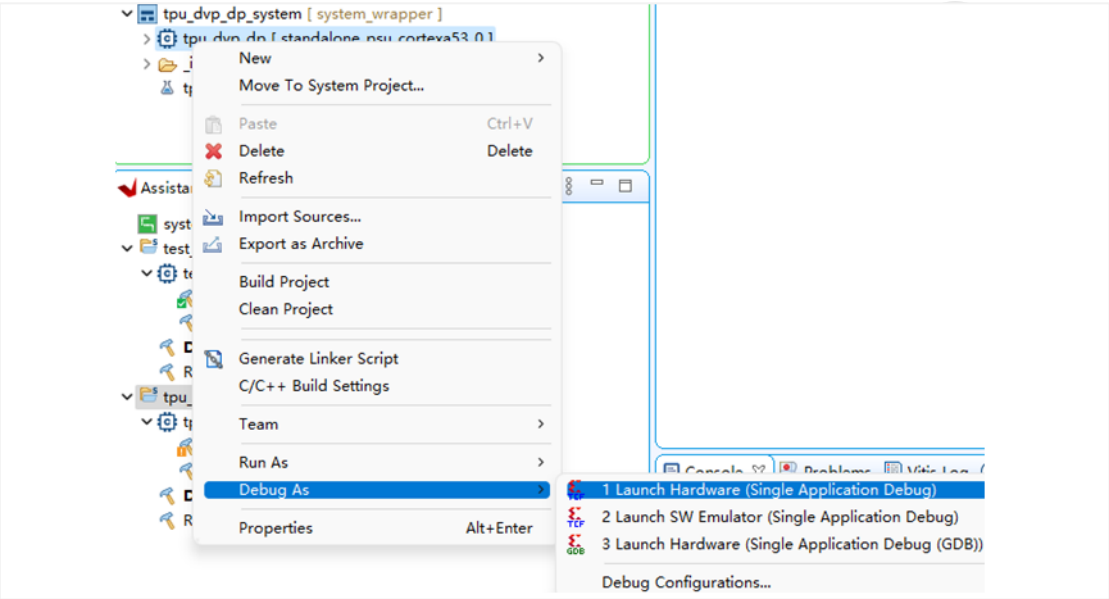

Figure 4.7 Program running

(2) SD card boot

Copy the “BOOT.bin, eepnet.mem and eepinput.mem” files in the “demo/boot” folder to the SD card, and set the boot mode to “SD card boot”, that is, the mode switch is set to “ON ON OFF OFF”.

After the development board is powered on, the following print information can be seen from the serial terminal:

```
Enter the Number:Xilinx Zynq MP First Stage Boot Loader
Release 2021.1   Nov 18 2022  -  16:24:22
PMU-FW is not running, certain applications may not be supported.
Loading program ......
HPD event .......... ! Connected.
Lane count =    2
Link rate =     10

Starting Training...
          ! Training succeeded.
DONE!
.......... HPD event
output cnt = 2
out[0]: hwaddr 0x320f44c0, shape: 1,255,13,13
out[1]: hwaddr 0x321096c0, shape: 1,255,26,26
in: hwaddr 0x31bac4c0, shape: 1,3,416,416
mem_base = 0x31000000
tpureg_addr = 0xa0000000
hwbase0 = 0x31000000
hwbase1 = 0x31bac4c0
hwbase2 = 0x3215dec0
hwbase3 = 0x320f44c0
memsize = 0x01bcde00
addr_alg = 0x31b8c400
TPU hardware version: 0x00810b32 (0.8.4-r11-p50)
Loading data
Load Net Data From SD Card...
Load Net Done
################################################
Choose Feature to Test:
1: Get 1 Frame
2: Forward Result
3: Save Image to SD Card
4: Read Test Image
5: Run Demo
0: Exit
Enter the Number:
```

Figure 4.8 Initial menu

At this time, the DP monitor displays the initial interface shown in Fig.4.9, indicating that the DP monitor can be displayed normally.

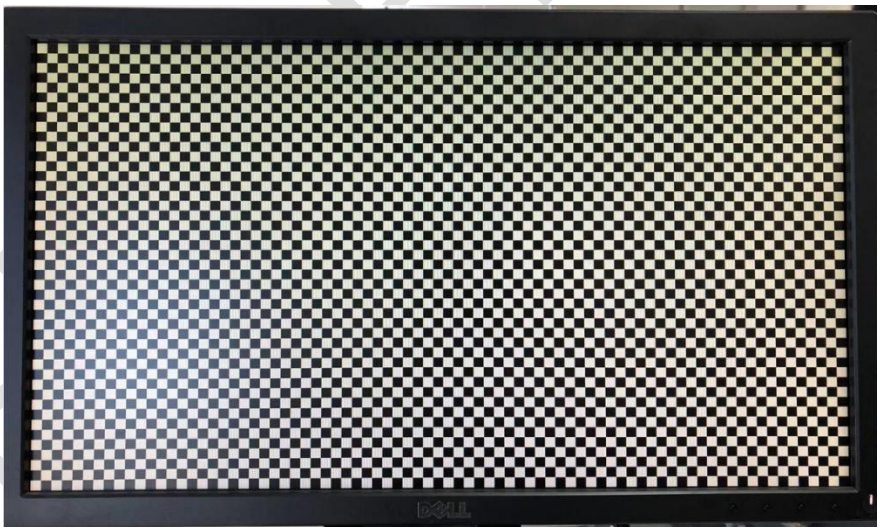

Figure 4.9 Initial display image

Test menu description:

(1) Acquire a frame of image and displays it by DVP;

(2) Inference the acquired images;

(3) Save image data into SD card;

(4) Read the test picture data (the data is saved in the SD card to test whether the inference is correct);

(5) Continuous DVP acquisition of images and inference.

Test items:

(1) In the serial port menu, first enter "4" to load the test image data from the SD card, and then enter "2" to inference test images, you can see the serial port printing results shown in Figure 4.10:

```
Choose Feature to Test:
1: Get 1 Frame
2: Forward Result
3: Save Image to SD Card
4: Read Test Image
5: Run Demo
0: Exit
Enter the Number:4
################################################
Choose Feature to Test:
1: Get 1 Frame
2: Forward Result
3: Save Image to SD Card
4: Read Test Image
5: Run Demo
0: Exit
Enter the Number:2
Forwarding...
read output[0]: addr 0x320f44c0, shape: 255,13,13
epmat_size = 0x15200(86528)
read output[1]: addr 0x321096c0, shape: 255,26,26
epmat_size = 0x54800(346112)
forward time is 38679 us
detection output time is 62124 us
Obj[0]:  8(     truck)  0.910327; At (0.007392,0.276477)  0.251595 x 0.515154
Obj[1]:  1(    person)  0.846338; At (0.857661,0.324374)  0.899836 x 0.472777
Obj[2]: 18(     horse)  0.821468; At (0.443775,0.124901)  0.829803 x 0.885698
Obj[3]: 17(       dog)  0.600459; At (0.283588,0.547477)  0.388209 x 0.892048
Obj[4]:  1(    person)  0.325858; At (0.487803,0.033946)  0.695681 x 0.658211
```

Figure 4.10 Test picture inference

(2) In the serial port menu, first enter 1 to acquire a frame of image through DVP, and then enter 2 to inference the acquired image and print the result.

```
################################################
Choose Feature to Test:
1: Get 1 Frame
2: Forward Result
3: Save Image to SD Card
4: Read Test Image
5: Run Demo
0: Exit
Enter the Number:1
Image Capture Done
################################################
Choose Feature to Test:
1: Get 1 Frame
2: Forward Result
3: Save Image to SD Card
4: Read Test Image
5: Run Demo
0: Exit
Enter the Number:2
Forwarding...
read output[0]: addr 0x320f44c0, shape: 255,13,13
epmat_size = 0x15200(86528)
read output[1]: addr 0x321096c0, shape: 255,26,26
epmat_size = 0x54800(346112)
forward time is 38684 us
detection output time is 62145 us
Obj[0]:  1(    person)  0.808453; At (0.007016,0.121290)  0.850179 x 1.035409
```

Figure 4.11 Practical graphical reasoning

(3) In the serial port menu, enter 5 to make the DVP continuously collect images and reason.

5. Instructions for embedded linux

The compiled BOOT.BIN and image.ub files are stored in the folder hardware/BOOTbin, through which the embedded Linux system can be booted. In the Linux system, users can test the TPU accordingly (note: the TPU is an evaluation version, so the system needs to be restarted every hour). For the Linux system, please refer to the “eep-ug050” and “eep-ug053” documents in the doc directory for the use instructions of TPU. The corresponding demo program is stored in the “sdk” directory.

6. Appendix

6.1 Compilation and transformation of neural networks models

Refer to the use document of the compiler and compile the neural network model into TPU executable bin file. Then through the “eepBincvt” tool, the bin file is converted into the relevant C language header file and model binary file. These header files and binary model files are the network model and related configuration information required for bare metal testing.

(1) Compilation of neural networks models

Run the “sdk/standalone/net_model/scripts/b_yolo4tiny.sh” script to generate “eeptpu_s2.pub .bin” in the “sdk/standalone/net_model/binRoot/yolov4tiny” folder.

(2) Transformation of neural networks models

The purpose of Bin file conversion is to convert the bin file generated by the compiler into a file format suitable for running bare metal programs. The “eepBinCvt” runtime environment is “Ubuntu 18.04”.

Running the “sdk/standalone/net_model/scripts/cvt .sh” script will generate four files: “eepinput.h”, “eepinput.mem”, “eepnet.h” and “eepnet.mem” in the current directory. Copy the. “.h” file to “src/net_ In the data” folder, copy the .mem file to the SD card, and the application program loads the neural network model data from the SD card to the DDR memory.

```
(base)                                                    NNmodel/eepBinCvt_v2.1.0# ./cvt.sh
./eepBinCvt --bin ../scripts/binRoot/yolov4tiny/eeptpu_s2.pub.bin --input ../models/images/ssd/004545.bmp --outp
ut header

[ eepBinCvt v2.1.0 ]

Bin file: ../scripts/binRoot/yolov4tiny/eeptpu_s2.pub.bin
Input file: ../models/images/ssd/004545.bmp
Base address: 0x00000000
Output type: header
Generating header file...
Bin file is 'public-bin'.
Bin use compiler v2.3.13
Write net data to: ./eepnet.h
Write input data to: ./eepinput.h
Generate ok: cost 834.332 ms

./eepBinCvt --bin ../scripts/binRoot/yolov4tiny/eeptpu_s2.pub.bin --input ../models/images/ssd/004545.bmp --outp
ut mem

[ eepBinCvt v2.1.0 ]

Bin file: ../scripts/binRoot/yolov4tiny/eeptpu_s2.pub.bin
Input file: ../models/images/ssd/004545.bmp
Base address: 0x00000000
Output type: mem
Generating mem file...
Bin file is 'public-bin'.
Bin use compiler v2.3.13
Write ini to: ./eepnet.ini
Write net data to: ./eepnet.mem
Write input data to: ./eepinput.mem
Generate ok: cost 28.615 ms
```

Fig. 6.1 Model compilation example

6.2 FPGA resource usage

The “TPU v3+” design is implemented in the FPGA device “xczu15eg-ffvb1156-2-I”, and the FPGA resource occupation is shown in Figure 6.2:

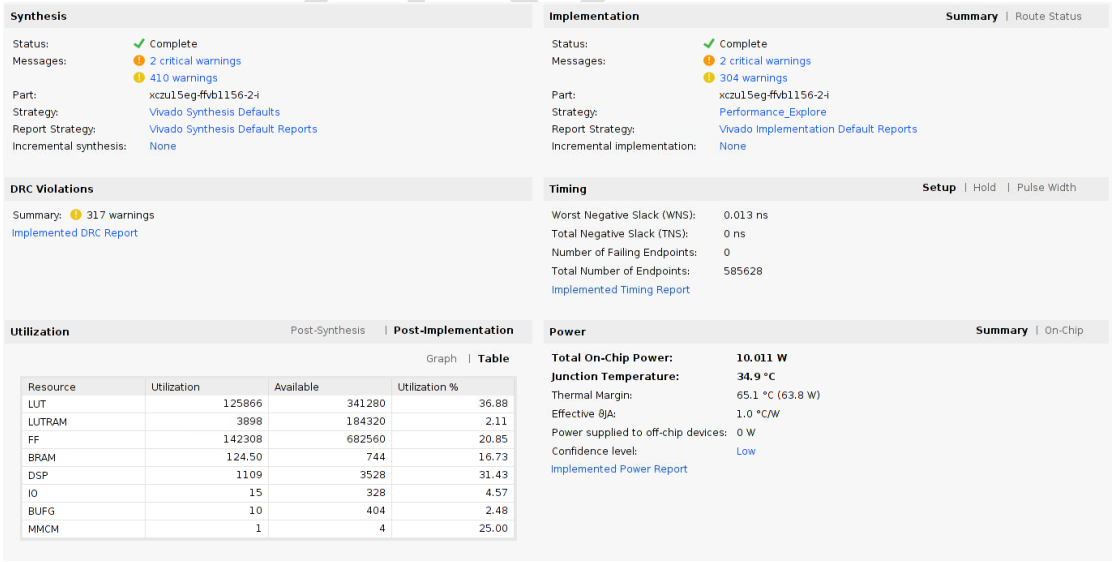

Fig. 6.2 FPGA resource occupation