2019 年 SEU-Xilinx 国际暑期学校团队项目设计文档

(Project Paper Submission Template)

作品名称	基于 FPGA 的高帧率目标检测系统设计
组员姓名	林晓波 北京科技大学 自动化学院 博士四年级 许运丰 北京大学 软件与微电子学院 硕士一年级 刘杨琳 南京航空航天大学 能源与动力学院 航空工程 硕士一年级 刘扬 东南大学 电子学院 本科三年级
房间号及桌号	720 教室 25 组/桌

第一部分

小组成员分工

（各成员任务分配）

	姓名	任务分配
组长	林晓波	总体策划，petalinux，找 yolo 和 ssd 原始模型
组员 1	许运丰	使用 OpenCV 进行图像处理,安装 AI SDK 以及通过 SDK 和配合项目组生成的模型完成应用的开发
组员 2	刘杨琳	搭建基于 ultra96 的 block design，配置 dpu ip，成.hdf 文件。学习使用 petalinux
组员 3	刘扬	搭建 DNNDK 开发环境，使用 DECENT 和 DNNC 对 caffe 模型进行定点化和编译

第二部分

设计概述 /Design Introduction

1、功能说明

基于机器视觉的智能目标检测系统应用非常广泛，尤其在实际应用中，经常涉及高速目标的实时检测和控制，对目标检测的智能性和实时性提出了更严格的要求。使用深度学习技术实现的目标检测算法，能够准确地将目标检测并识别出来。但为了提升准确率使用大型网络，由此因此很难满足图像信喜传输和处理的实时性要求。

目前很多学者正致力于高帧率目标检测系统的研究。学术界常用来提高识别帧率的方法有，通过网络剪枝减小网络冗余，通过量化减少参数冗余，通过定点化减少计算复杂度和压缩网络，通过并行、流水技术增加计算单元的计算并行度。

本研究以高帧频与实时性作为研究的切入点，设计一种基于 Ultral 96 平台，充分利用芯片上的 FPGA 资源及其硬件并行的优势，进行目标检测识别的任务。本研究主要从两个方面入手实现既定目标：一是从算法上，选取更适合应用场景的网络，再逐步对网络进行压缩，将网络配置到硬件上。二是从硬件上，增加摄像头 OV5640 以及 HDMI 接口，板卡上电后直接对 OV5640 捕获到的图像进行识别，再将识别结果以 HDMI 接口输出，摆脱了对主机的依赖，更加符合 IOT 对终端设备的要求。

2、设备清单

Ultra96*1

网线*1

数据线*1

SD 卡*1

屏幕*1

HDMI 线*1

USB 摄像头*1

PYNQ*1

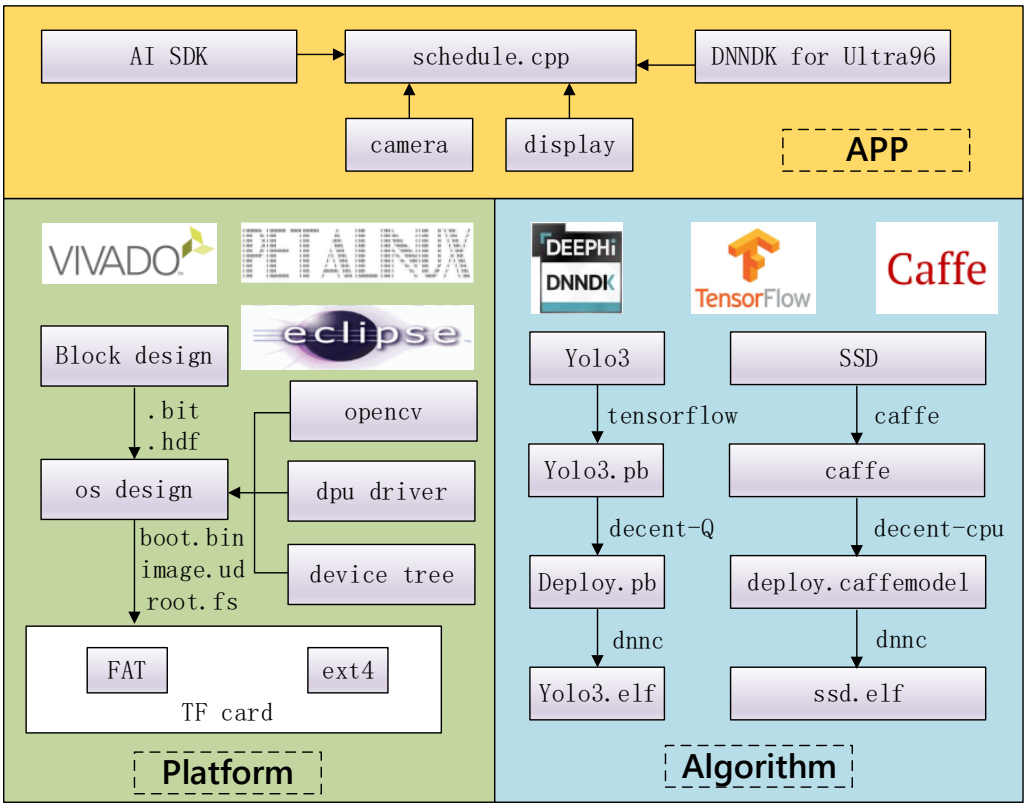
第三部分

详细设计 /Detailed Design

功能说明

Xilinx 提供的 demo 识别速率为 14fps,本研究最终目标是希望能达到对 640*480*3 的图像实现 30fps 的识别效果，包括人、车辆、桌椅、动物等，并将结果通过 HDMI 接口实时显示出来。

系统组成



1 使用 petalinux 定制 linux os

1.1 添加 vivado 输出的板级支持信息，即 hdf 和 bit 文件

1.2 配置 congfig 参数，如 rootfs 从 sd 卡启动、设备树、支持 uvc 驱动等

1.3 添加 dpu 驱动和 opencv

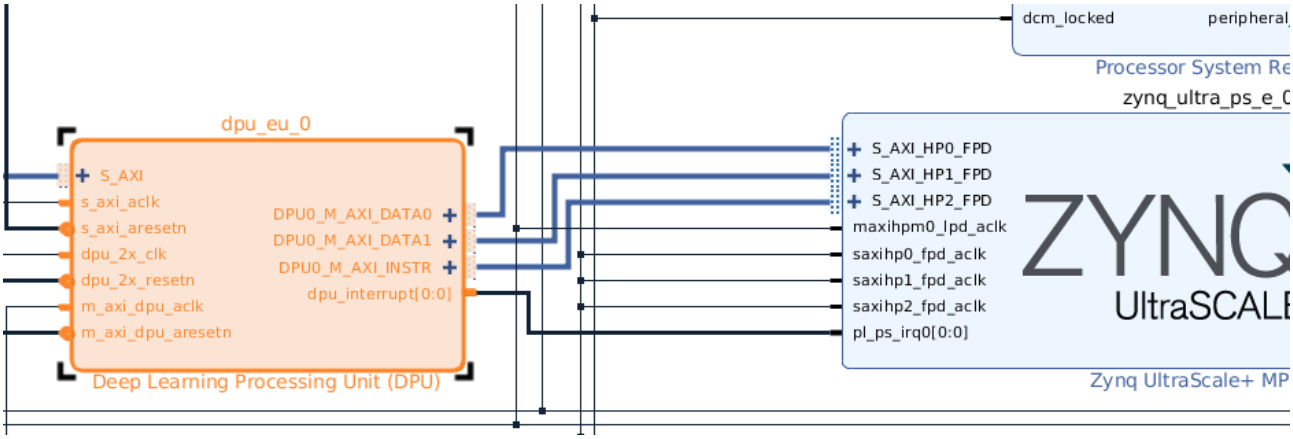
1.4 dpu 中断信号的处理

查阅资料得知 ps 的接口和 linux 中断管教的对应信息

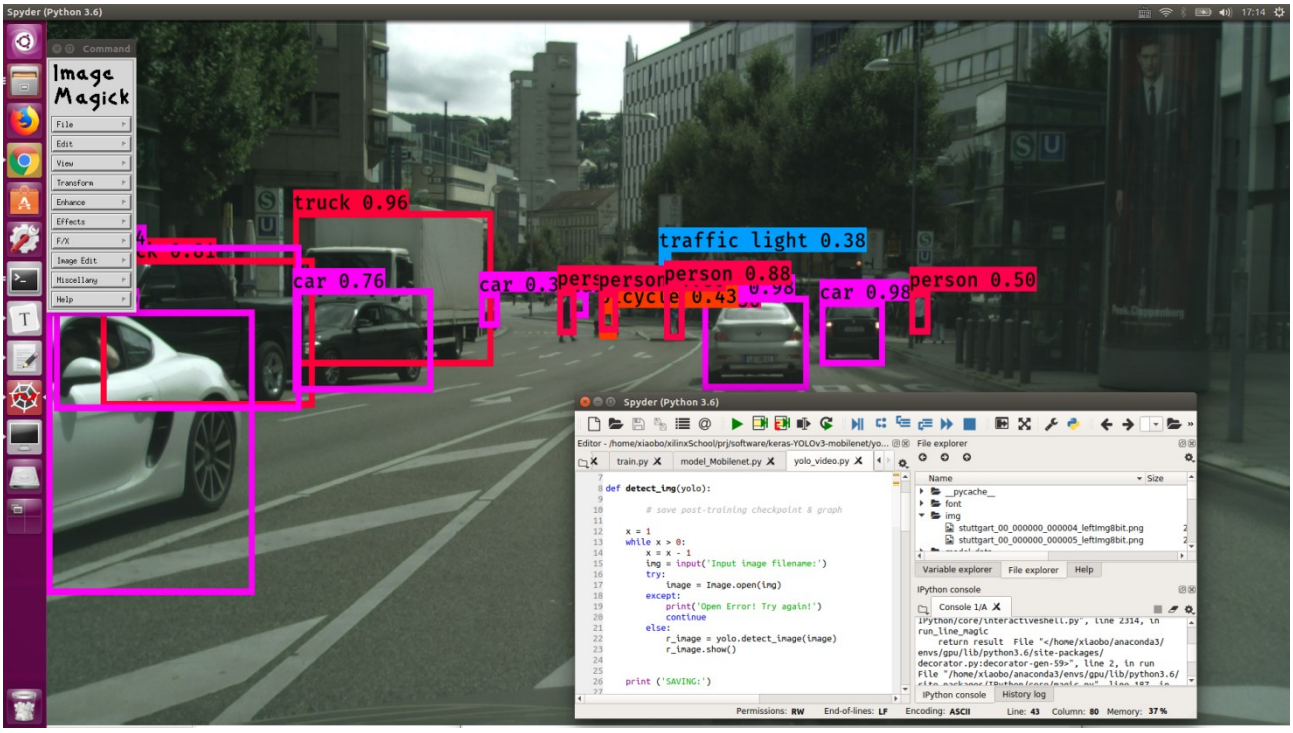
Interrupt Values

PS Interface	GIC IRQ #	Linux IRQ #
PL_PS_IRQ1[7:0]	143:136	111:104
PL_PS_IRQ0[7:0]	128:121	96:89

通过 vivado 查看中断连接的管脚



2 使用 dnndk x86端对下载到的 tensorflow 模型进行压缩和编译



2.1 下载合适的 yolo-mobile 模型，并使用 python 对其进行测试

```
In [11]: from tensorflow.python.framework import tensor_util
         from google.protobuf import text_format
         import tensorflow as tf
         from tensorflow.python.platform import gfile
         from tensorflow.python.framework import tensor_util


         GRAPH_PB_PATH = 'trans_model/yolo.pb' #path to your .pb file
         with tf.Session() as sess:
             print("load graph")
             with gfile.FastGFile(GRAPH_PB_PATH,'rb') as f:
                 graph_def = tf.GraphDef()
             # Note: one of the following two lines work if required libraries are available
                 #text_format.Merge(f.read(), graph_def)
                 graph_def.ParseFromString(f.read())
                 tf.import_graph_def(graph_def, name='')
                 for i,n in enumerate(graph_def.node):
                     print("Name of the node - %s" % n.name)
```

```
Name of the node - reshape_2/Reshape/shape/2
Name of the node - reshape_2/Reshape/shape/3
Name of the node - reshape_2/Reshape/shape/4
Name of the node - reshape_2/Reshape/shape
Name of the node - reshape_2/Reshape
Name of the node - reshape_3/Shape
Name of the node - reshape_3/strided_slice/stack
Name of the node - reshape_3/strided_slice/stack_1
Name of the node - reshape_3/strided_slice/stack_2
Name of the node - reshape_3/strided_slice
Name of the node - reshape_3/Reshape/shape/1
Name of the node - reshape_3/Reshape/shape/2
Name of the node - reshape_3/Reshape/shape/3
Name of the node - reshape_3/Reshape/shape/4
Name of the node - reshape_3/Reshape/shape
Name of the node - reshape_3/Reshape
Name of the node - output_1
Name of the node - output_2
Name of the node - output_3
```

2.2. 使用 tensorflow frozen 工具将 tf 模型中的权值固化到模型中

2.3 使用 decent_q 工具对模型进行定点化压缩

```
In [11]: from tensorflow.python.framework import tensor_util
         from google.protobuf import text_format
         import tensorflow as tf
         from tensorflow.python.platform import gfile
         from tensorflow.python.framework import tensor_util


         GRAPH_PB_PATH = 'trans_model/yolo.pb' #path to your .pb file
         with tf.Session() as sess:
             print("load graph")
             with gfile.FastGFile(GRAPH_PB_PATH,'rb') as f:
                 graph_def = tf.GraphDef()
             # Note: one of the following two lines work if required libraries are available
                 #text_format.Merge(f.read(), graph_def)
                 graph_def.ParseFromString(f.read())
                 tf.import_graph_def(graph_def, name='')
                 for i,n in enumerate(graph_def.node):
                     print("Name of the node - %s" % n.name)
```

```
Name of the node - reshape_2/Reshape/shape/2
Name of the node - reshape_2/Reshape/shape/3
Name of the node - reshape_2/Reshape/shape/4
Name of the node - reshape_2/Reshape/shape
Name of the node - reshape_2/Reshape
Name of the node - reshape_3/Shape
Name of the node - reshape_3/strided_slice/stack
Name of the node - reshape_3/strided_slice/stack_1
Name of the node - reshape_3/strided_slice/stack_2
Name of the node - reshape_3/strided_slice
Name of the node - reshape_3/Reshape/shape/1
Name of the node - reshape_3/Reshape/shape/2
Name of the node - reshape_3/Reshape/shape/3
Name of the node - reshape_3/Reshape/shape/4
Name of the node - reshape_3/Reshape/shape
Name of the node - reshape_3/Reshape
Name of the node - output_1
Name of the node - output_2
Name of the node - output_3
```

这一步使用教程给的 minist 是成功的，但是使用定点后的 yolo3.pb 模型，没有成功，提示有类型不支持。由于项目时间较短，且我们没有 gpu 服务器，因此没有进行修改后模型的训练

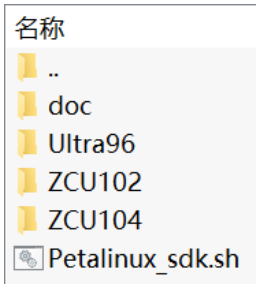
DNNSDK 的使用

工具：Xilinx AI SDK， OpenCV， Make

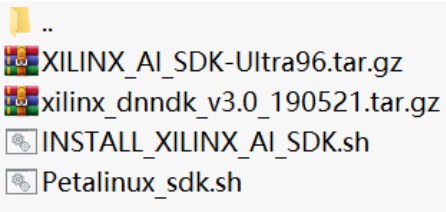
一、环境安装

1. 文件 xilinx-ultra96-prod-dpu1.4-desktop-buster-2019-05-31 解压后是一个镜像文件，用来给 ultra96 烧写 Linux 镜像。使用 Win32DiskImager 工具完成。

2. 文件 xlnx_dnndk_v3.0_190624 下包含两个压缩包 XILINX_AI_SDK-V1.0.0-BUILD-16-2019-05-31.tar.gz 和 xilinx_dnndk_v3.0_190624.tar.gz XILINX_AI_SDK-V1.0.0-BUILD-16-2019-05-31.tar.gz 文件解压后看到目录结构为：

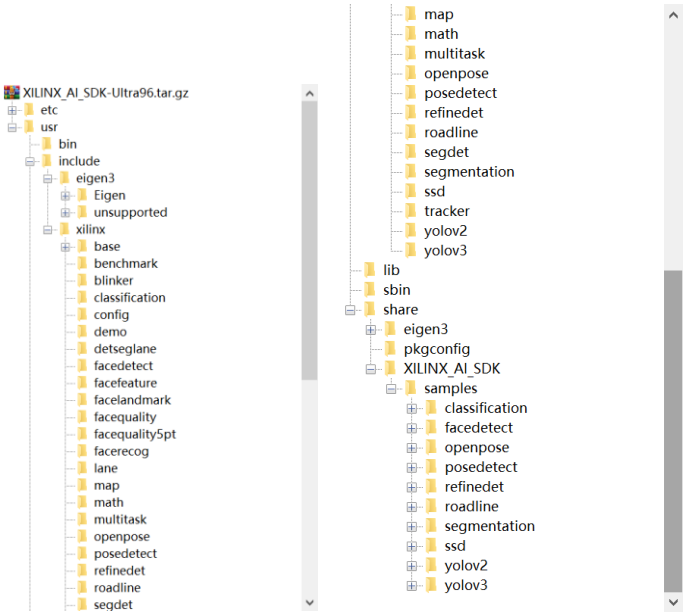


Petalinux_sdk.sh 用于给主机端安装 petalinx sdk 工具，Ultra96 下文件夹由目标执行



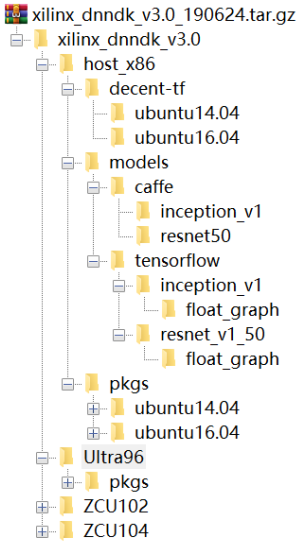
4. INSTALL_XILINX_AI_SDK.sh 主要完成两个功能

1. 安装 xilinx_ai_sdk



xilinx_ai_sdk 安装后这个文件被解压放到对应目录,share/XILINX_AI_SDK 下有对应程序

2. xilinx_dnndk_v3.0_190624.tar.gz 文件打开后



可以看到是主机端的安装文件用于完成定点化、编译等功能，ultral96 文件夹下是对应的 DPU 的驱动

二、安装后可以看到 SDK 各个模块被放到对应的文件夹

```
Model and Library files are stored in /usr/lib
The header files are stored in /usr/include/Xilinx
Model profiles are stored in /etc/XILINX_AI_SDK.conf.d
Samples are stored in /usr/share/XILINX_AI_SDK/samples
Documents are stored in /usr/share/XILINX_AI_SDK/doc
```

Build.sh 用来生成目标文件

三、SDK 的使用

1. 使用YOLO完成目标检测

1.新建文件yolo_test.cpp

2. 包含头文件.

```
#include <xilinx/yolov3/yolov3.hpp>//Necessary head file
#include <iostream>
#include <map>
#include <string>
//The necessary opencv lib
#include <opencv2/opencv.hpp>
```

3. 加载图片

```
Mat img = cv::imread(argv[1]);
```

4. 实例化模型

```
auto yolo = xilinx::yolov3::YOLOv3::create_ex("yolov3_voc_416", true);
```

5. 运行程序

```
auto results = yolo->run(img);
```

6 显示结果

```
for(auto &box : results.bboxes){
    int label = box.label;
    float xmin = box.x * img.cols + 1;
    float ymin = box.y * img.rows + 1;
    float xmax = xmin + box.width * img.cols;
```

```
        float ymax = ymin + box.height * img.rows;
        if(xmin < 0.) xmin = 1.;
        if(ymin < 0.) ymin = 1.;
        if(xmax > img.cols) xmax = img.cols;
        if(ymax > img.rows) ymax = img.rows;
        float confidence = box.score;
        cout << "RESULT: " << label << "/t" << xmin << "/t" << ymin << "/t"<< xmax << "/t"
    << ymax << "/t" << confidence << "/n";
        rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0),1, 1, 0);
    }
```

7 .make产生可执行程序之后运行

2. 加入摄像头模块

1. 打开摄像头

```
VideoCapture cap(0);
```

2.将摄像头的画面输入网络中。

```
Mat frame;
Cap>>frame;
auto results = yolo->run(img);
```

3. 改用视频流

1.打开文件

```
VideoCapture cap(“test.avi”);
```

2.将视频的画面输入网络中。

```
Mat frame;
Cap>>frame;
auto results = yolo->run(img);
```

定点化、编译

工具：DNNDK、linux/虚拟机（本组使用 Oracle VM VirtualBox）

训练出来的两个文件为权重参数和网络模型，都是浮点数型的。为了节约资源并加快计算，需要将上述的浮点数据转化为定点数，即量化/定点化（quantization）操作。得到定点数型的文件后，即可开始编译（compilation）。编译的作用是将权重和网络模型转化为可执行的“.elf”文件。量化和编译的详细操作如下：

搭建 DNNDK 的开发环境，安装 DECENT 和 DNNC 编译器。

DECENT 用于量化操作：输入为浮点数型的权重和网络模型文件。输出为定点数型的权重和网络模型。使用其专用量化指令，并设置输入、输出文件及其路径、工作模式等，即可在 DNNDK 中进行量化操作。需要注意的是，需要根据实际开发环境来选择 decent/decent-cpu 指令，否则会报错。

下图：decent 指令

```
decent      quantize                                    \
            -model float.prototxt                       \
            -weights float.caffemodel                   \
            -output_dir decent_output                   \
             -method 1
```

下图：量化进程

```
I0717 00:18:25.502142 14571 net.cpp:266] res2a_relu needs backward computation.
I0717 00:18:25.502152 14571 net.cpp:266] res2a needs backward computation.
I0717 00:18:25.502162 14571 net.cpp:266] res2a_branch2c_fixed needs backward computation.
I0717 00:18:25.502172 14571 net.cpp:266] res2a_branch2c needs backward computation.
I0717 00:18:25.502182 14571 net.cpp:266] res2a_branch2b_fixed needs backward computation.
I0717 00:18:25.502197 14571 net.cpp:266] res2a_branch2b_relu needs backward computation.
I0717 00:18:25.502208 14571 net.cpp:266] res2a_branch2b needs backward computation.
I0717 00:18:25.502218 14571 net.cpp:266] res2a_branch2a_fixed needs backward computation.
I0717 00:18:25.502225 14571 net.cpp:266] res2a_branch2a_relu needs backward computation.
I0717 00:18:25.502806 14571 net.cpp:266] res2a_branch2a needs backward computation.
I0717 00:18:25.503108 14571 net.cpp:266] res2a_branch1_fixed needs backward computation.
I0717 00:18:25.503259 14571 net.cpp:266] res2a_branch1 needs backward computation.
I0717 00:18:25.503274 14571 net.cpp:266] pool1_pool1_fixed_0_split needs backward computation.
I0717 00:18:25.503283 14571 net.cpp:266] pool1_fixed needs backward computation.
I0717 00:18:25.503298 14571 net.cpp:266] pool1 needs backward computation.
I0717 00:18:25.503309 14571 net.cpp:266] conv1_relu needs backward computation.
I0717 00:18:25.503317 14571 net.cpp:266] conv1 needs backward computation.
I0717 00:18:25.503327 14571 net.cpp:268] data_fixed does not need backward computation.
I0717 00:18:25.503340 14571 net.cpp:268] data does not need backward computation.
I0717 00:18:25.503355 14571 net.cpp:310] This network produces output prob
I0717 00:18:25.503768 14571 net.cpp:330] Network initialization done.
I0717 00:18:25.568043 14571 decent.cpp:198] Start Calibration
I0717 00:20:41.776150 14571 decent.cpp:222] Calibration iter: 1/100 ,loss: 0.155767
```

DNNC 用于编译操作：输入为定点化的权重和网络模型文件，输出为可执行的“.elf”文件。使用其专用编译指令，并设置输入输出文件及其路径、名称，并设置工作模式、CPU 类型、GPU 类型等，即可在 DNNDK 中进行编译操作。需要注意的是 CPU 和 DPU 类型的选择需要慎重，对于 Ultra96，CPU 类型选择 arm64，对于 GPU 类型的选择，需要与底层硬件电路设计共同完成,本组选用 2304FA 型号。

下图：dnnc 指令

```
dnnc    --prototxt=deploy.prototxt                  \
        --caffemodel=deploy.caffemodel              \
        --output_dir=dnnc_output                    \
        --net_name=resnet50                         \
        --dpu=4096FA                                \
        --mode=debug                                \
       --cpu_arch=arm64                            \
        --abi=0
```

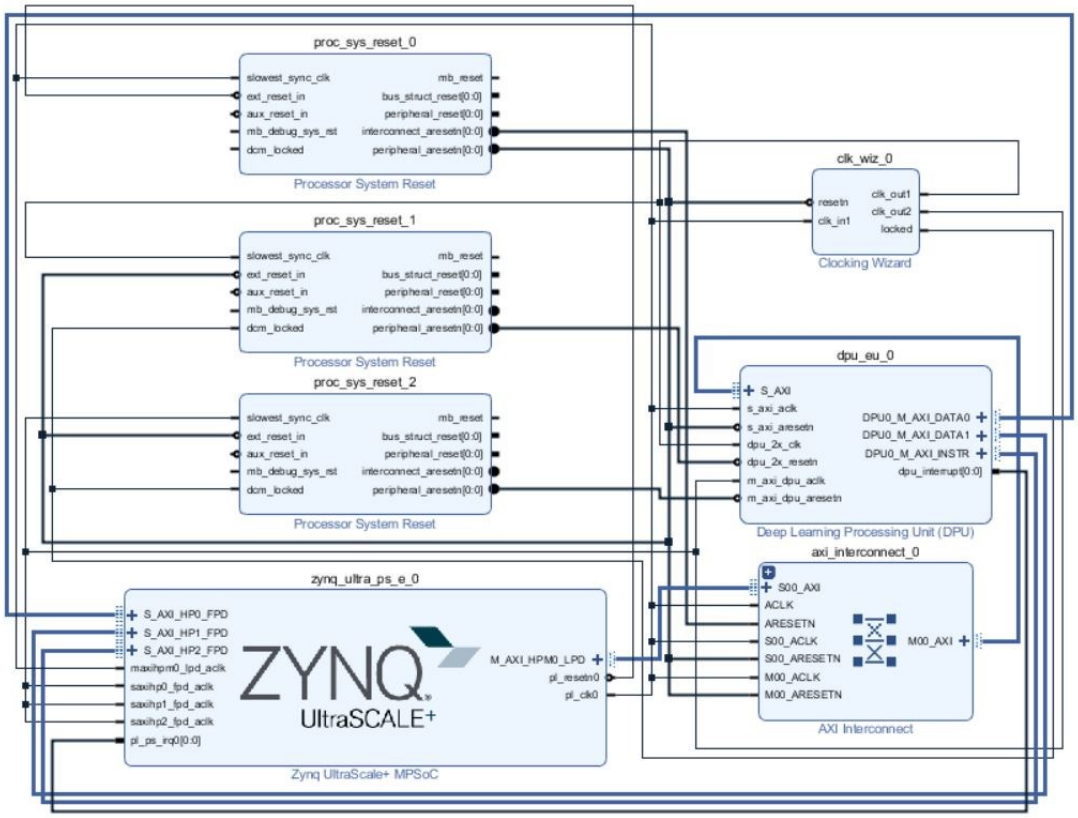
下图：编译进程

```
sh dnnc.sh
DNNDK Version: v3.0
Board Name: Ultra96
DPU Arch: 2304FA
CPU Arch: arm64
DNNC: dnnc-dpu1.4.0
DNNC Mode: debug
Compiling Network: resnet50
[DNNC][Warning] layer [prob] is not supported in DPU, deploy it in CPU instead.

DNNC Kernel Information

1. Overview
kernel numbers  : 2
kernel topology : resnet50_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : resnet50_0
type            : DPUKernel
nodes           : NA
input node(s)   : conv1(0)
output node(s)  : fc1000(0)

kernel id       : 1
kernel name     : resnet50_1
type            : CPUKernel
nodes           : NA
input node(s)   : prob
output node(s)  : prob
```

硬件部分

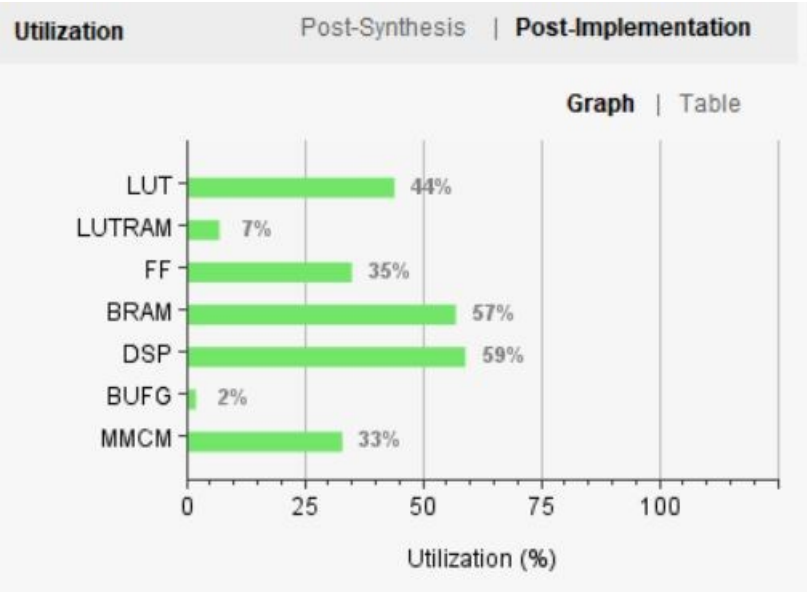
工具：vivado2018.2

1.搭建基于 ultra96 的 block design，配置 DPU IP，下图为 DPU 核数为 1 的 block design 图



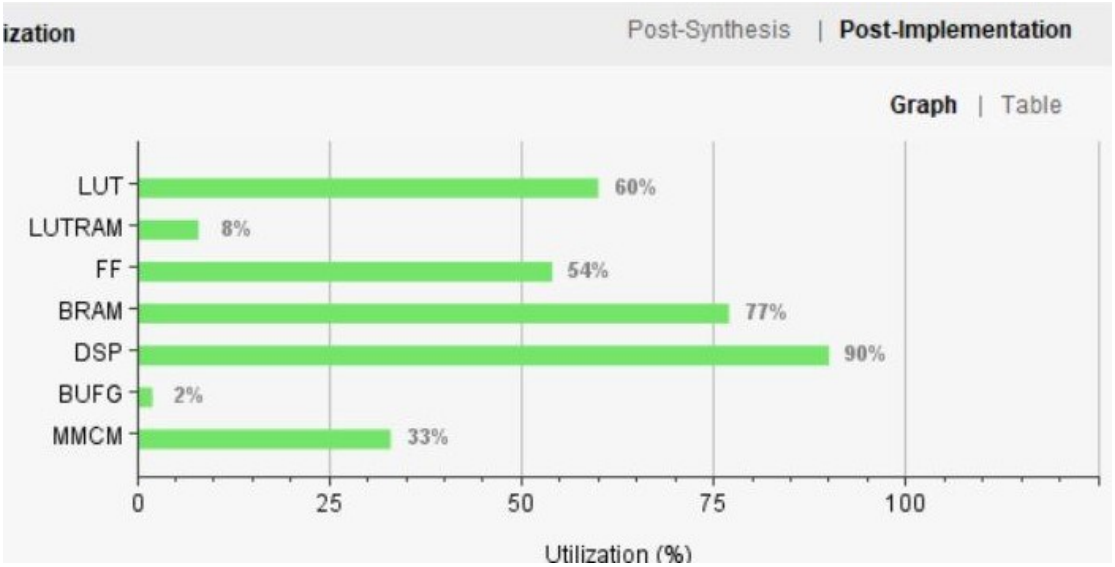
2. DPU IP 可以配置有与卷积单元的并行性相关的不同卷积架构。DPU IP 的可选架构包括 B512、B800、B1024、B1152、B1600、B2304、B3136 和 B4096。分别配置 DPU 核的模式为 B1152、B2304，综合、实现、生成比特流

配置 DPU 单核模式为 B1152 时资源利用率如下图



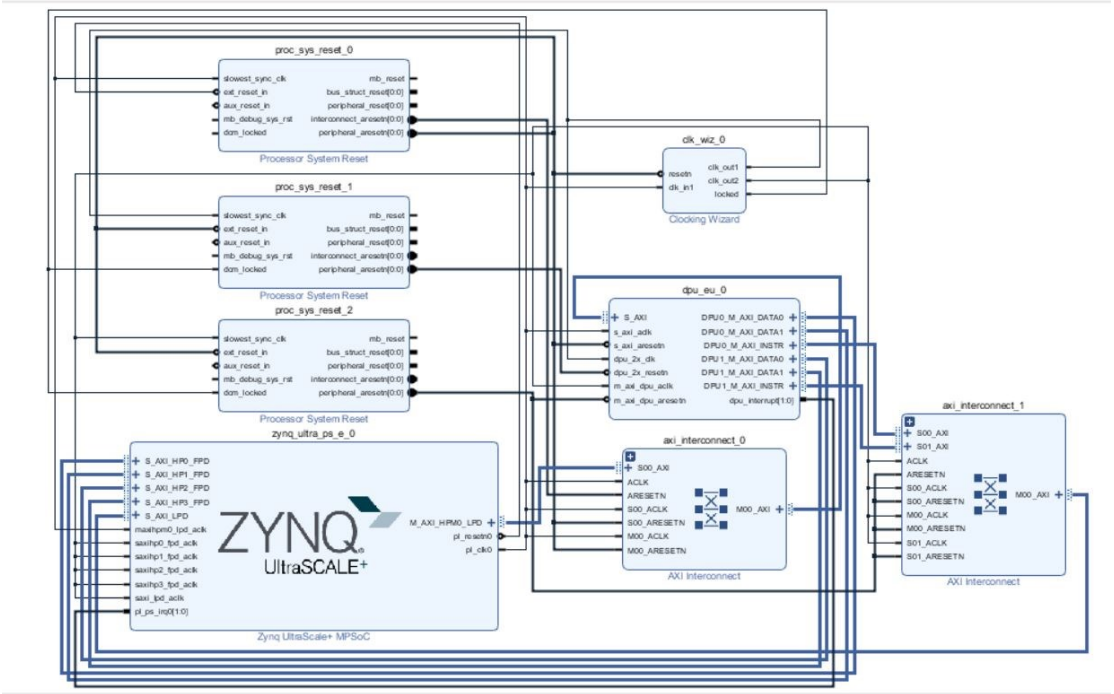
配置 DPU 单核模式为 B2304 时出现 DSP 资源不够报错，查阅《DPU IP Product Guide》，在高 DSP 使用模式下，DSP 片将用于乘法和累加。因此，较高的 DSP 使用率消耗了更多的 DSP 片和更少的 LUT。如果选择较低的 DSP 使用率，则 DPUIP 将仅在卷积中使用 DSP 片进行乘法。考虑到 LUT 资源还剩余较多，修改 DPU 核的 DSP Usage 模式为 Low DSP Usage

下图为 DPU 单核模式为 B2304 时实现后的 report

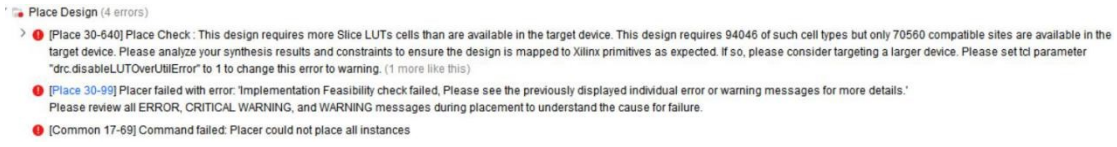


可以明显看出 DPU 的资源利用率提高

3. 可以在一个 IP 中设置三个 DPU 核。多个 DPU 核可以实现更高的性能。修改 DPU 核数为 2，配置 DPU 双核的数据和指令输出，其中双核的指令输出通过 axi_interconnect IP 连到 ultra96 的 PS 端。block design 如下图



但是双核模式配置为 B1152 时资源不够报错



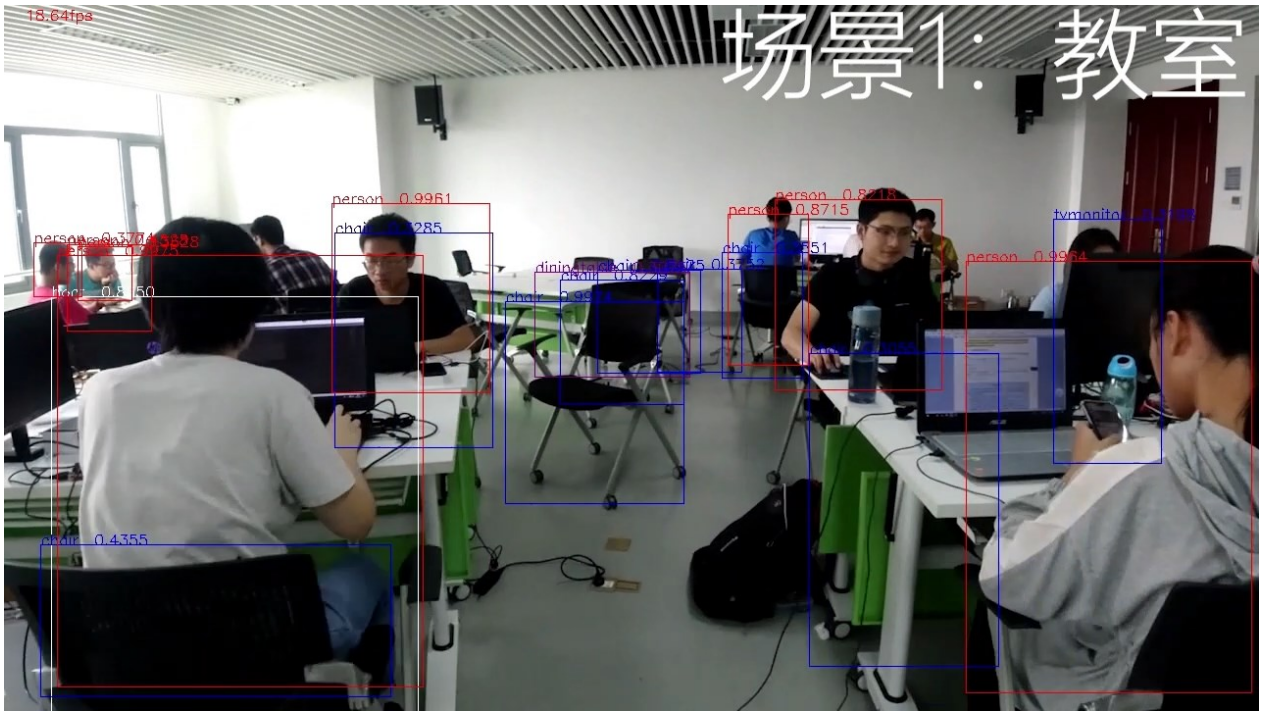
查阅《DPU IP Product Guide》中不同配置下最佳性能表，综合考量我们所需性能要求，最终选择 DPU IP 为单核 B2304 模式，且 DSP 使用模式为 Low DSP Usage

Device	DPU Configuration	Frequency (MHz)	Peak Performance
Z7020	B1152x1	200	230 Gops
ZU2	B1152x1	370	426 Gops
ZU3	B2304x1	370	852 Gops
ZU5	B4096x1	350	1.4 Tops
ZU7EV	B4096x2	330	2.7 Tops
ZU9	B4096x3	333	4.1 Tops

4. 输出 petalinux 所需.hdf 文件

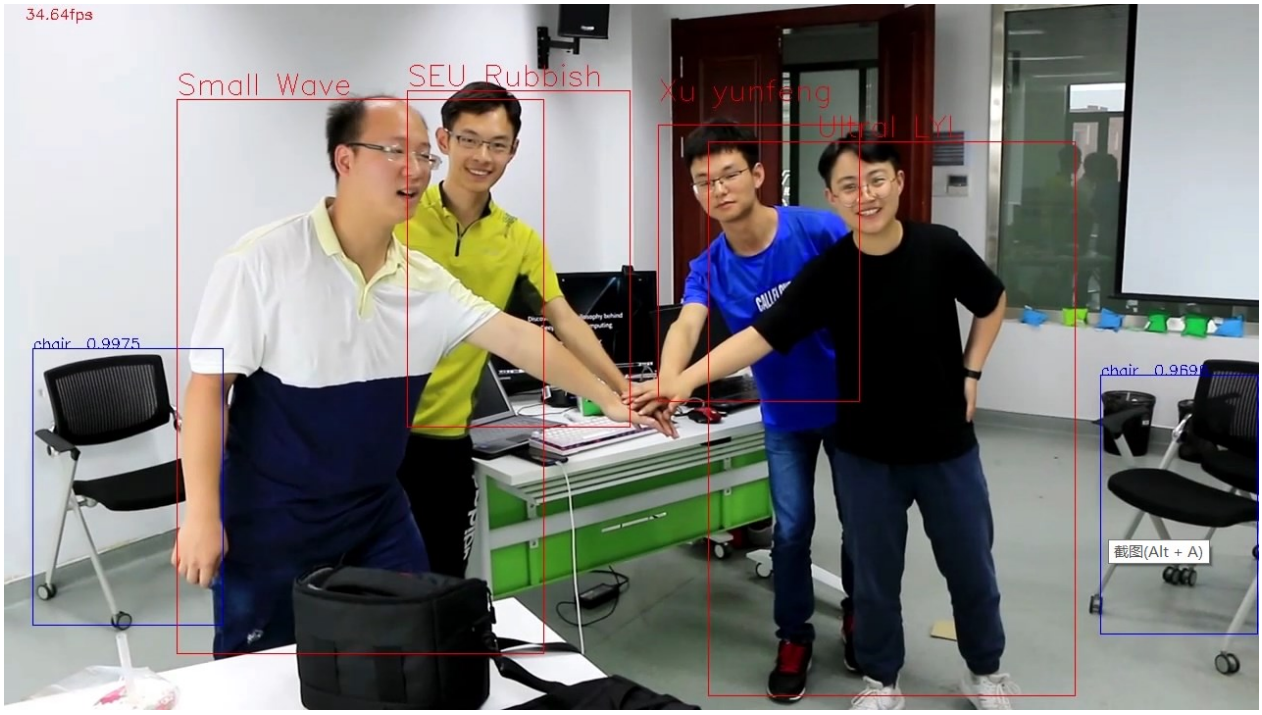
第四部分

完成情况及性能参数 /Final Design & Performance Parameters




场景2：食堂
person 0.8173
person 0.9941
person 0.8501
person 0.7055
person 0.7719


20.72fps
3：马路
car 0.5395
car 0.4065
car 0.6947



视频链接

https://github.com/xilinxfairchild/xilinxSummer_presentation/blob/master/videoList.md

第五部分

项目总结 /Conclusions

林晓波：

所用知识点：

1、使用 petalinux 根据 vivado 生成的 hdf 和 bit 文件编译 os，使用配置定制 os。

2、学习 dpu 1.4 的配置和 dnndk 3.0 的指南，使用 dnndk for x86 对得到的模型进行压缩和转换

3、 学习 dnndk for ultra96 编程模型，在 ultra96 开发板上编写调用生成 elf 的目标检测程序。

收获：对 dpu 和 dnndk 有了一定的了解

心得：在短短的几天里，通过队友间的分工合作，大致完成了预期的目标。感受到团队的力量是巨大的，感谢靠谱的队友们！可惜活动就要结束了，可惜，可惜。。

刘杨琳：

所用知识点：

1、查阅 ultra96 原理图完成 ultra96 的 ps 端配置

2、查阅 dpu 指南完成 dpu ip 核的配置

3、学习 petaliux 环境下生成 boot 文件和 image 文件

收获：

1. 在配置环境的时候一定要多看 xilinx 官方文档。

2. 遇到问题解决问题的过程要记录下来，好记性不如烂笔头。

心得：

在短短的几天里跟着团队一起做项目，什么都懂的林博士，严谨的许运丰同学，认真的刘扬同学，每一个人都值得学习值得信赖，非常感谢他们的鼓励和信任，以后要学的东西还有很多，还要继续加油

刘扬：

所用知识点：

1、CNN 的原理及实现方法

2、Linux 系统使用方法

3、使用 DNNDK 进行定点化和剪枝的方法

收获：

1、科学检索资料的方法、学会了使用 github 等技术网站

2、对 Xilinx 开发工具（Vivado、HLS、SDK 等）运用熟练度提升，学会使用 Xilinx ZYNQ 系列器件（PYNQ-Z2、Ultra96），对 FPGA 应用潜力的认识加深

3、对深度学习技术理解加深

心得：

理论与实现要相得益彰。有价值的理论才有实现的意义，发挥出工具的价值；而先进的工具也是理论是否能实现的先决条件。所以个人的发展要兼顾两者。这次暑期学校也让我认识了很多优秀的同学，让我坚定了一直努力下去的信念。也接触了一些奇思妙想又功能强大的器件。所以我以后想要在最前沿发挥自己的价值，无论理论还是应用。

许运丰：

所用知识点：

1. 安装 AI SDK 以及通过调用 SDK 完成应用的开发

2. OpenCV 图像处理
3. 嵌入式及 linux 基本指令

收获：

1. xilinx 的官方文档有很多详细的介绍，查阅资料对于开发有很大的左右
2. 要把自己学到的和不懂的东西记下来，方便以后再查阅
3. 学习的时候，要从上而下，从顶至下，先掌握整体流程再扣细节

心得：

这次项目锻炼了自己项目开发的能力，和队友们相处也很愉快。不能做一个只会在擅长领域默默 Coding 的程序猿，更要学会与队友沟通，在遇到问题的时候不能心浮气躁，一定要有耐心才能找到问题的根源。

第六部分

源代码/Source code

https://github.com/xilinxfairchild

第七部分（技术总结/心得/笔记等分享）

详情请见 Github