1、搭建山海关的开发环境

1.1 安装交叉编译工具链

交叉编译工具链在nas服务器上，路径为：

/nas/users/platform/AMBA\_4\_CV25\_linux\_shanhaiguan/tools/Ambarella\_Toolchain\_Linaro\_2020.09.tar.xz。

解压此文件，然后运行安装脚本：

tar pxvf Ambarella\_Toolchain\_Linaro\_2020.09.tar.xz

cd Ambarella\_Toolchain\_Linaro\_2020.09

sudo ./ubuntuToolChain-linux5.4-202008

这样交叉编译工具链就安装完成。

1.2 clone山海关工程的仓库

仓库地址为http://192.168.1.124/liyunlong/shg\_amba\_cv25\_sdk\_3\_0\_2

1.3 source build.sh

进入shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/boards/cv25\_hazelnut目录（Hazelnut代表是CV25开发板，Chestnut是CV2开发板，Walnut是CV22开发板，Cashewnut是CV28开发板），运行source build.sh

如果出现Check env error, please install mkfs.jffs2genext2fsmkcramfs错误。需要安装这些库：

mkfs.jffs2：sudo apt-get install mtd-utils

genext2fs：sudo apt install genext2fs

mkcramfs：sudo apt-get install mkcramfs

如果提示缺少crypto，那么就需要安装crypto python库：

sudo apt-get install python3-crypto

再次source build.sh，直到没有错误出现，如下所示：

============================================================

TOOLCHAIN PATH: /usr/local/linaro-aarch64-2020.09-gcc10.2-linux5.4

TOOLCHAIN NANE: aarch64-linux-gnu-gcc

TOOLCHAIN VERSION: 10.2.1

============================================================

#

# configuration written to .config

#

Build Linux-5.4 with cv25\_emmc\_kernel\_config ...

1.4 编译山海关项目工程

进入shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/boards/cv25\_hazelnut/目录中，运行make命令：

cd shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/boards/cv25\_hazelnuts

make -j8

编译完成后，终端显示信息如下所示：

...

...

/home/public/shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/out/cv25\_hazelnut/rootfs/ext4: clean, 3281/3584 files, 427637/457988 blocks

Build Amboot ...

Build AmBST ...

Build Done

这样就在shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/下生成了out文件夹，里面包含编译生成的镜像和可执行程序，下面烧写步骤中的elf文件就在这个文件夹里。

1.5 编译山海关SDK的doxygen文档

cd shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/boards/cv25\_hazelnuts

make sdk\_doc

这样在shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/document/doc下就会生成doxygen文档，里面是代码相关的文档。

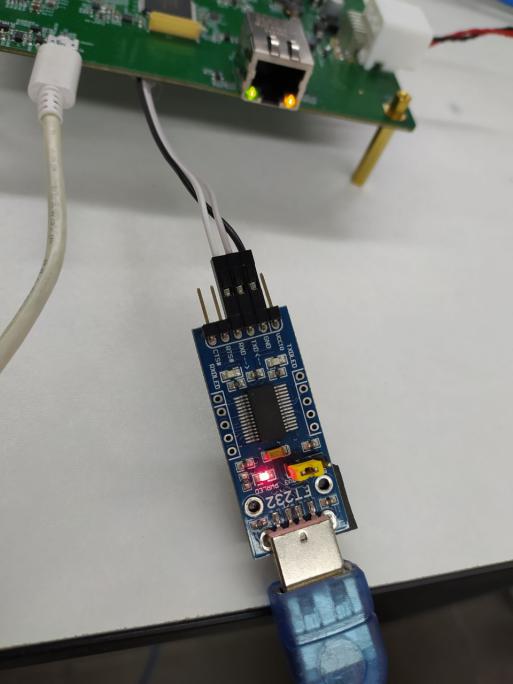
2、针对新拿到的山海关板子烧写

2.1 安装烧写工具

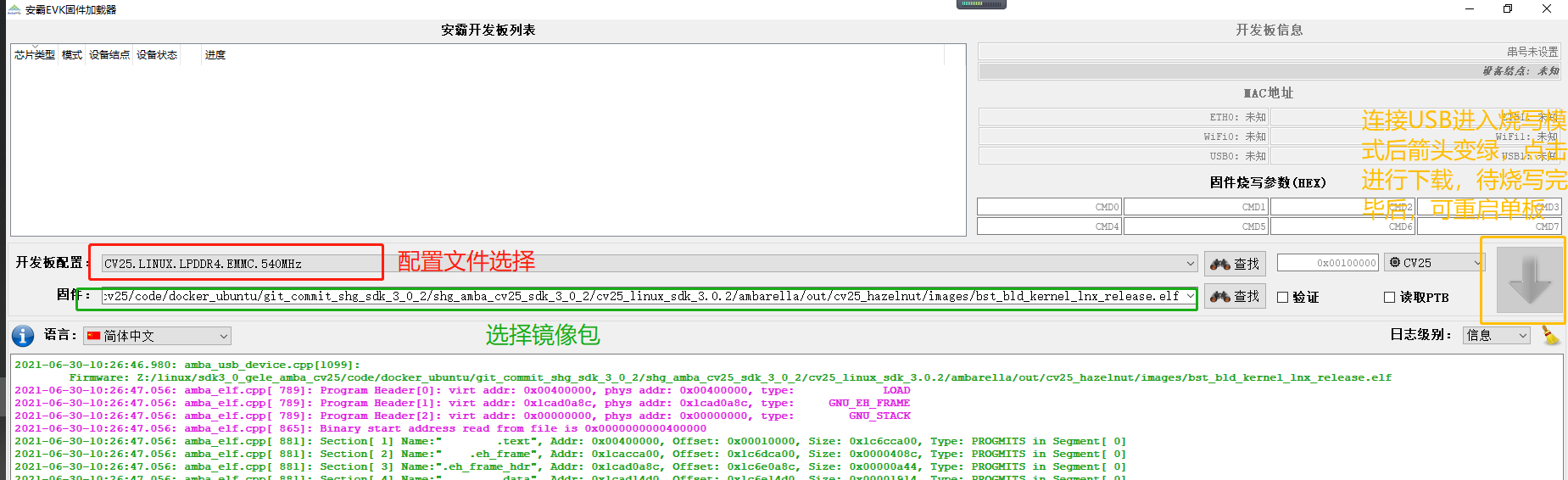
烧写工具的路径为：/nas/users/platform/AMBA\_4\_CV25\_linux\_shanhaiguan/tools/Ambarella\_AmbaUSB\_v4.2.6.zip

烧写工具可以运行在不同的OS上——Ubuntu 18.04或者Windows。安装烧写工具。

2.2 用USB线连接山海关板子和PC机，用串口线连接山海关板子的UART串口

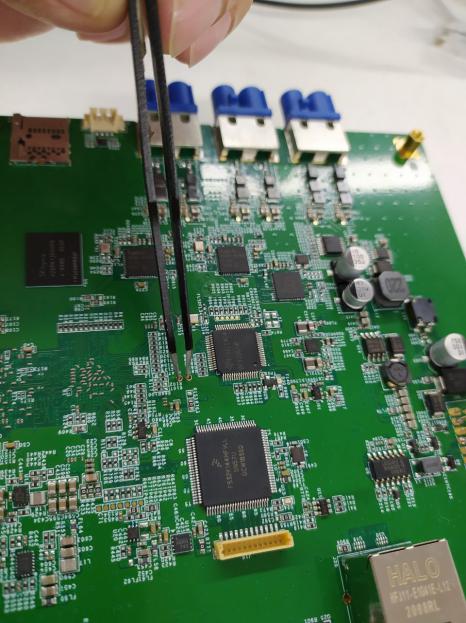


2.3 打开烧写工具，界面如下：



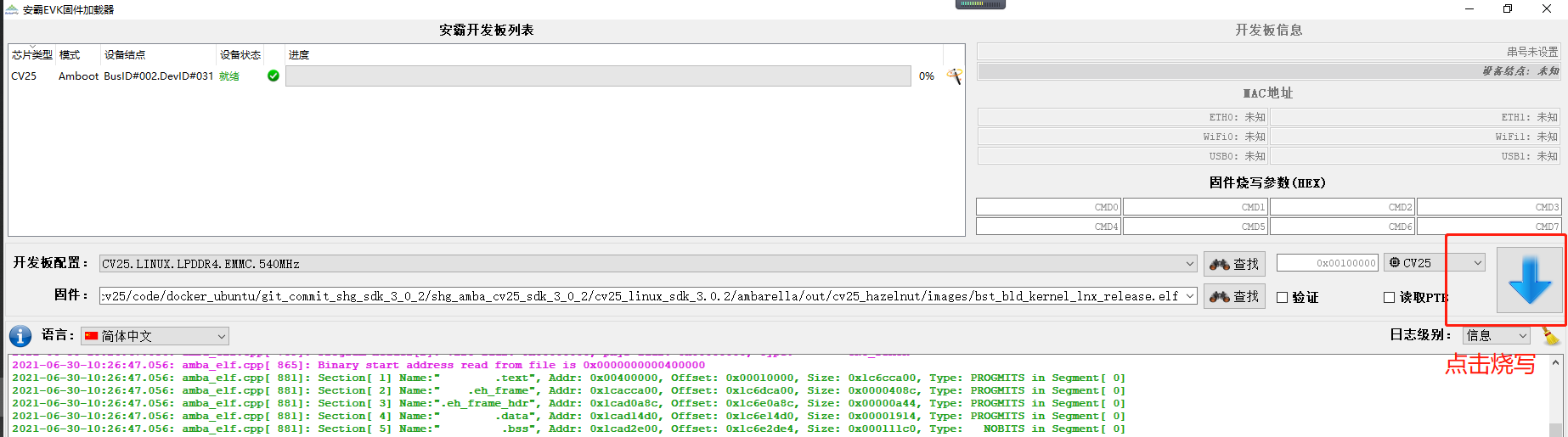
由于我们的SOC为cv25，所以右面选择CV25。

2.4 短接山海关板子上的预留端点（POC[10]）可以进入USB烧写模式。我们这里使用镊子进行短接，如下图所示。基本步骤：POC[10]短接-->板子上电-->插上USB线-->点击烧写按钮。

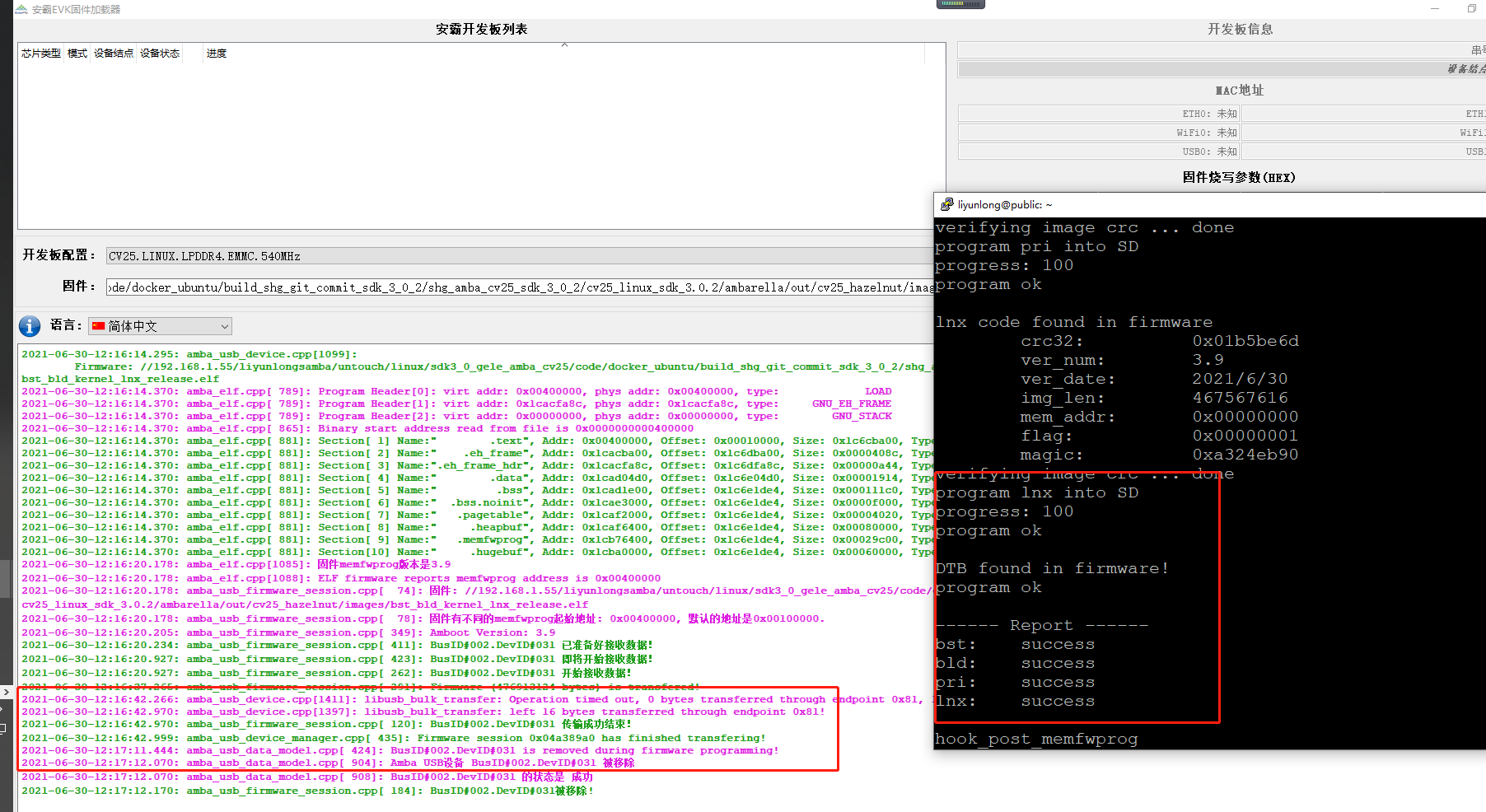


2.5 进行烧写

点击蓝色按钮进行烧写。



2.6 烧写完成（图片两个红框都满足，代表烧写完成）：



3、建立NFS

可以参看《Ambarella\_CV2x\_UG\_Flexible\_Linux\_SDK3.0\_Code\_Building\_and\_Debug\_Environment.pdf》文档的“6.5Network Sharing: NFS”部分。

3.1 通过串口设置IP地址，比如设置为192.168.1.178

vi /etc/network/interfaces

添加如下内容：

auto eth0

iface eth0 inet static

address 192.168.1.178 # IP地址， 要根据自己网段下IP的使用设置，不能和别的IP相冲突

netmask 255.255.255.0

gateway 192.168.1.1

保存后，重新启动板子。然后ping一下其他IP地址，确认网络通信正常。

3.2 在PC机上建立nfs文件夹，并编辑PC机上的/etc/exports文件

确保PC机上已经启动了nfs服务，可以用service nfs-server status命令查看。然后

建立/home/public/shanhaiguan\_nfs文件夹，然后添加nfs信息，即在/etc/exports文件最后添加下面一行：

/home/public/shanhaiguan\_nfs \*(rw,nohide,insecure,no\_subtree\_check,async,no\_root\_squash)

然后运行exportfs -a命令，使修改后的/etc/exports立即生效。

3.3 在板子上进行mount

在板子上建立nfs对应的（对应主机的/home/public/shanhaiguan\_nfs）文件夹：/nfs\_zhuhaijun。然后运行如下mount命令：

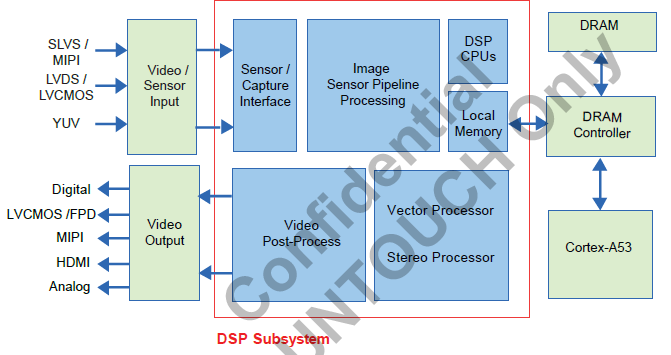
mount -t nfs -o nolock,vers=3 192.168.1.110:/home/public/shanhaiguan\_nfs /nfs\_

zhuhaijun

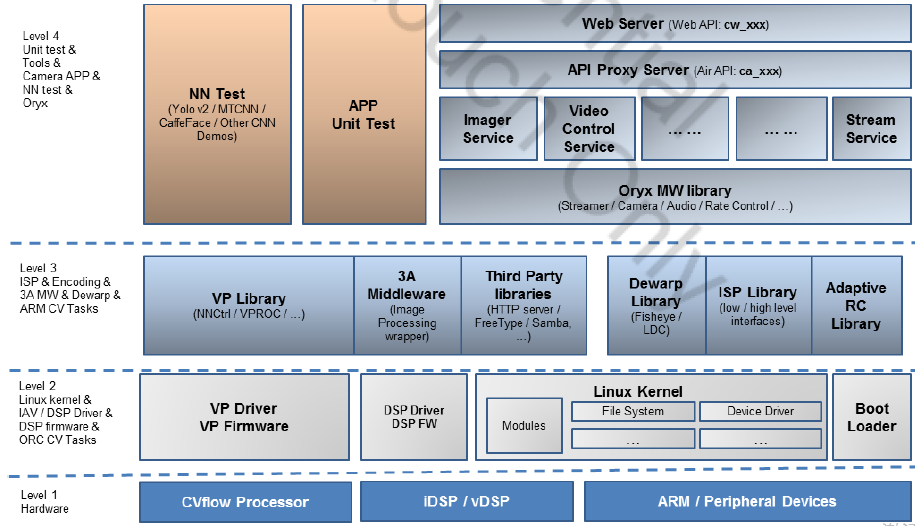
这样就完成了nfs的挂载，可以看到PC机上的文件可以在板子上显示。

4、安霸SDK的特点

4.1 CV25 DSP部分的架构



4.2 SDK软件架构



4.3 SDK目录结构

SDK的顶层目录为shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella，现在对目录里包含的内容进行简要说明：

AMBoot/ Boot loader, board support toolkit, and file system information about the partitions.

app/ Demo applications.

boards/ The compile configuration files, DDR configuration, and boot strap.

build/ The make file rules and AmbaConfig rules.

document/ The Doxygen content and configuration rules.

include/ Header files.

kernel/ Linux kernel, DSP firmware, IAV / DSP / sensor and peripheral devices drivers.

external Codes of external Wi-Fi modules

linux-5.4 Codes of Linux kernel and patches.

private Codes and binaries of Ambarella propriety drivers or modules.

license/ The license files of each component.

out/ The build target and intermediate objects.

packages/ The libraries that assist in video processing, streaming, or transferring.

data\_transfer Codes of data storage over NFS and data transferring over USB/TCP.

img\_algo (Source codes are not released) Image algorithm library.The prebuilt binary is in ambarella/prebuild/imgproc.

utils Utility library codes

nnctrl Initial cavalry binary which is generated by cavalry\_gen

vproc Some pre-defined DAGs used to convert the date format

prebuild/ The prebuilt binaries including Ambarella libraries and third-party libraries.

ambarella/library Binary format of various Ambarella proprietary library

imgproc Binary format library file and image tuning parameter files

third-party Binary format library files from the third party

rootfs/ The root file system.

unit\_test/ Unit test tools.

4.4 Linux驱动和应用库

IAV driver (main interface for the application to call CODEC functions)

Cavalry driver (main interface for the application to call VP functions)

DSP, VIN, VOUT drivers (GPLv2 Linux modules)

Sensor driver (Proprietary Linux modules)

Image processing driver (Proprietary Linux modules)

Image processing (AE / AWB) library ( libimg\_algo\_v5.a / libimg\_dsp\_v5\_hf.a )

Image processing flow library ( libimg\_aaa\_flow\_v5.so )

Multi-channel Lua parser library ( libmcl.so )

Neural network flow control library ( libnnctrl.so )

Neural network data processing library ( libvproc.so )

Neural network memory management library ( libcavalry\_mem.so )

CNNGen Pre-Processing library ( libcgpp.so )

Easy AI library ( libeazyai.so )

Smart VOUT framebuffer library ( libsmartfb.so )

Electronic image stabilization library ( libeis.so )

Lens distortion correction library ( libdewarp.so )

Multi-region warp library ( libdewarp.so )

Smart rate controll library ( libsmartrc\_3\_0.so )

Text insertion library ( libtextinsert.a )

Arbitrary Blur library ( libblur.so )

Audio codec library (ALSA, libasound.a )

4.5 demo和测试应用程序

Unit test to capture / parse DSP log ( dsplog\_cap )

Unit test to test encode and streaming ( test\_encode, test\_stream )

Unit test to exercise Digital Pan / Tilt / Zoom ( test\_dptz )

Unit test to exercise privacy mask functionality ( test\_privacymask )

Unit test to exercise overlay insert functionality ( test\_overlay, test\_textinsert )

Unit test to exercise blur insert functionality ( test\_blur )

Unit test to capture RAW / YUV / ME1 / ME0 data ( test\_yuvcap )

Unit test to enable MB / CTB level smart encoding ( test\_frame\_sync, test\_sync\_roi, test\_sync\_stream, test\_qproi )

Unit test to run 3A process ( test\_tuning, test\_aaa\_service )

Unit test to encode / decode AAC audio ( test\_aacdec, test\_aacenc )

Unit test to exercise video decode functionality ( test\_decode )

Unit test to exercise encode from memory (EFM) functionality ( test\_efm )

Unit test to exercise Lens Distortion Correction (LDC) functionality ( test\_ldc )

Unit test to run multi-channel functionality ( test\_multi\_chan )

Unit test to exercise VIN signal suspend / recovery functionality ( test\_vin\_cap )

Unit test to exercise pyramid manual feed functionality ( test\_pyramid\_mf )

Unit test to exercise canvas manual feed functionality ( test\_manual\_feed )

Unit test to exercise stream fps suggestion functionality ( test\_fps\_suggest )

Unit test to exercise NetLink functionality ( test\_netlink )

Unit test to exercise IDSP/encoder synchronization functionality ( test\_idsp\_enc\_sync )

Unit test to exercise Electronic Image Stabilization (EIS) functionality ( test\_eis\_warp )

Unit test to exercise fisheye dewarp functionality ( test\_dewarp )

Unit test to run Smart Rate Control (SRC) ( test\_smartrc\_pa )

Unit test to exercise image scale functionality ( test\_img\_scale )

Unit test to exercise customized SEI functionality ( test\_custom\_sei )

Unit test to exercise stream statistics functionality ( test\_statistics )

Unit test to exercise HEVC scaling list functionality ( test\_hevc\_scalelist )

Unit test to exercise IAV data poll functionality ( test\_poll )

Unit test to exercise custimized VOUT functionality ( test\_custom\_vout )

Unit test to load CVflow firmware ( cavalry\_load )

Unit test to capture / parse CVflow log ( cavalry\_log )

Unit test to exercise NN flow control functionality ( test\_nnctrl, test\_nnctrl\_live )

Unit test to exercise FMA functionality ( test\_cavalry\_fma, test\_cavalry\_fma\_live )

Unit test to exercise NN memory management functionality ( test\_cavalry\_mem )

Application to run RTSP streaming ( rtsp\_server )

Application to load DSP microcode ( load\_ucode )

Application to run classification ( Yolo-v2, Yolo-v3 )

Application to run face detection ( MTCNN )

**CNN-CNNGen开发流程文档**

1.概述

本文档旨在帮助用户了解Ambarella CNNGen工具的开发流程。CNNGen是CVflow®矢量处理器（VP）的代码生成工具，CVflow®矢量处理器是在Ambarella的机器视觉（CV）芯片系列中高效执行神经网络的关键视觉处理模块。

CNNGen为用户提供了一个高级描述接口，用于指定卷积神经网络（CNN），这是一种在图像识别和目标检测中非常有效的神经网络。

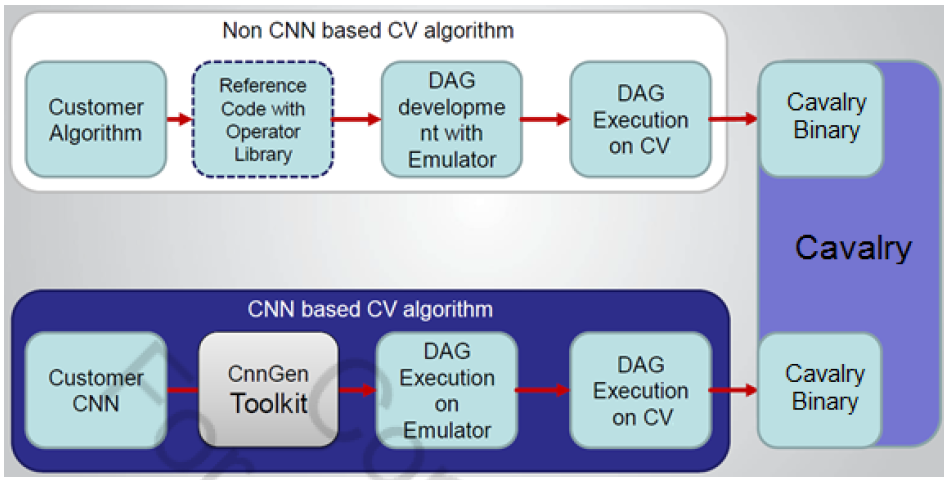
CV2 CVflow Vector Processor引擎的性能是CV22的5倍以上，CV25的性能是CV22的一半。CV2x CV工具支持CNN框架，例如Caffe、TensorFlow、TensorFlow Lite和ONNX。CV2x也支持GoogleNet, Resnet, Inception, Mobilenet, SSD等。

安霸提供了profile工具来分析CVflow处理器的性能，例如CVflow的pipeline efficiency和瓶颈。

CV25/CV22/CV2支持8 bit定点计算，CNNGen工具能够使用DRA来减少由此带来的精度损失。

安霸使用Linaro工具链编译ARM程序，使用CV工具链为CVflow处理器编译程序。

CNNGen开发流程：



2 工作流程

安霸CNNGen工具在PC上转换神经网络，然后在CV2x板子上运行神经网络。在Ubuntu 16.04中，用户向CNNGen提供模型文件，CNNGen随后转换文件并生成可以在CV2x板上运行的DAG。

基本流程：

（1）冻结并从CNN框架输出protobuf和模型；

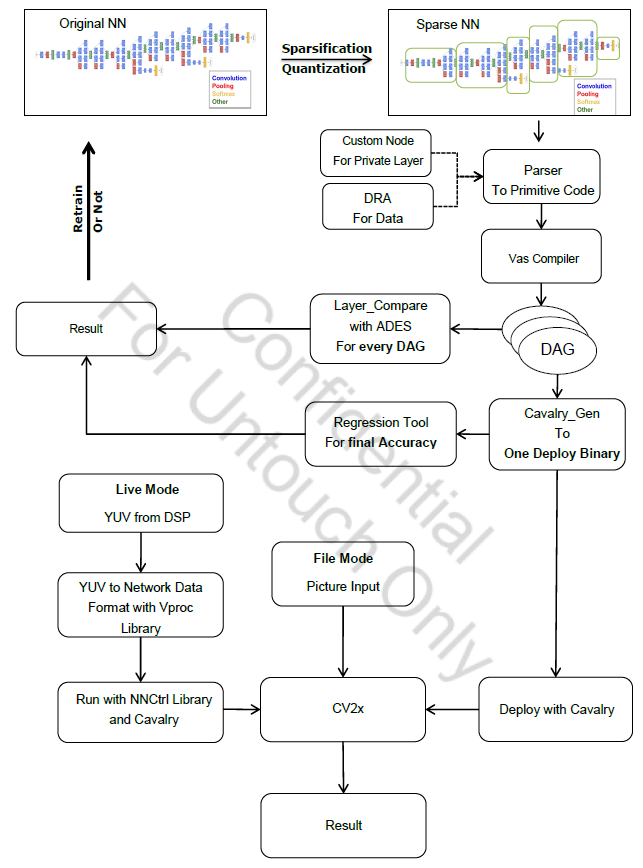
（2）通过Python解析器运行protobuf和模型，该解析器命令AmbaCNN API在CNNGen中构造节点图；

（3）CNNGen将节点图扩展为原始图，并执行量化和计算缩减；

（4）VAS将原始图扩展为运算符图（DAG），并执行低级优化和DAG拆分；

（5）在ADES上运行VAS生成的DAG以验证结果和准确性；

（6）使用cavalry\_gen脚本生成最终的执行二进制文件，并使用安霸接口在CV2x板上运行它。

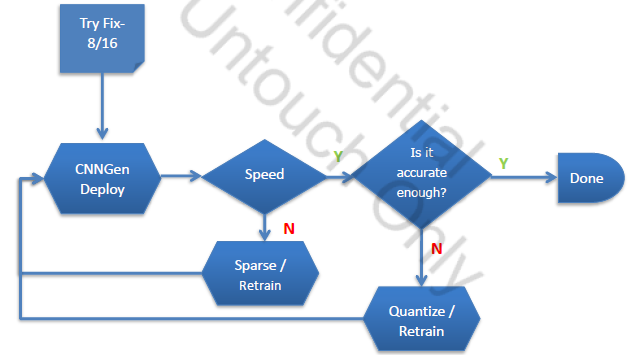


2.1 准备CNN模型

为了在CV2x上实现最佳性能，安霸建议对CNN模型进行剪枝（pruning）。CV2x包含一个新的、更快的卷积引擎，称为inception卷积引擎（ICE）。但是，由于ICE专门支持定点数据格式，因此数据格式必须为FX16或FX8。CNNGen 2.0将模型参数转换为FX16或FX8数据格式。如果原始模型基于FX32，则在转换过程中可能会丢失精度（16位的精度损失较小）。另外，在使用8位量化实现最佳性能的同时，可能会丢失精度。因此，在使用CNNGen之前，用户应该首先剪枝、执行量化，然后重新训练程序。

2.2 模型优化

下图显示了部署前CNN模型的推荐优化流程：



用户可以利用AmbaCaffe执行剪枝和量化，详细信息参考《Ambarella CV2x UG AmbaCaffe》。

2.2.1剪枝pruning

剪枝方法包括连接剪枝（connection pruning）、系数剪枝（coefficient pruning）等。

CVflow只能通过系数剪枝（稀疏化）提高性能，因为它包含高密度的零值系数。由于CVflow执行引擎可以绕过已知结果而不执行MAC，因此当系统受到MAC限制时，会有直接加速。用户可以实现低非零系数密度，同时在许多神经网络上保持准确性并获得3-4倍的速度提升。

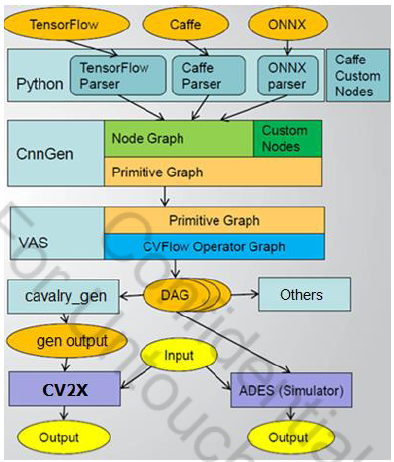
注:某些加速引擎可能无法从该技术所需的并行体系结构中获益。连接剪枝(Connection pruning)应用不同的剪枝技术。通过减少已处理的卷积核的数量，连接剪枝可以帮助CVflow和其他引擎。

2.2.2量化

虽然浮点（32位）用于训练和推理，但定点8 bit和定点16 bit速度更快、更节约资源。因此，所有加速引擎在边缘使用定点8 bit和定点16 bit进行推理。

关于AmbaCaffe量化，可以参考《Ambarella CV2x UG AmbaCaffe》。

2.3移植流程



cavalry\_gen和ADES工具具有不同的功能：

——cavalry\_gen是Cavalry框架中使用的输入工具，当用户必须开发真实案例时，建议使用Cavalry，因为它可以生成最终的Cavalry DAG，并包含一组丰富的API以与DAG一起运行。

——ADES能够仿真，使用户能够验证转换过程并生成输出，以便与EVK板上的运行进行比较。

移植流程如下：

——CNNGen解析器将节点图扩展为原始图，并使用DRA和自定义节点执行量化和计算缩减。（DRA通过数据分布确定数据格式，数据分布由设计师提供的示例图像计算得出。Custom node提供了一系列特定的API，使用户能够实现自己的层。）

——VAS将原始图扩展为算子图（DAG），并执行低级优化和DAG拆分。

——用户在ADES上仿真运行VAS生成的DAG，以此来确定结果是否正确。

——用户使用cavalry\_gen脚本生成最后在板子上可以运行的可执行文件，然后使用安霸的接口在板子上运行。

2.3.1 Dynamic Range Analysis(DRA)

2.3.2 Debug

2.4 部署网络

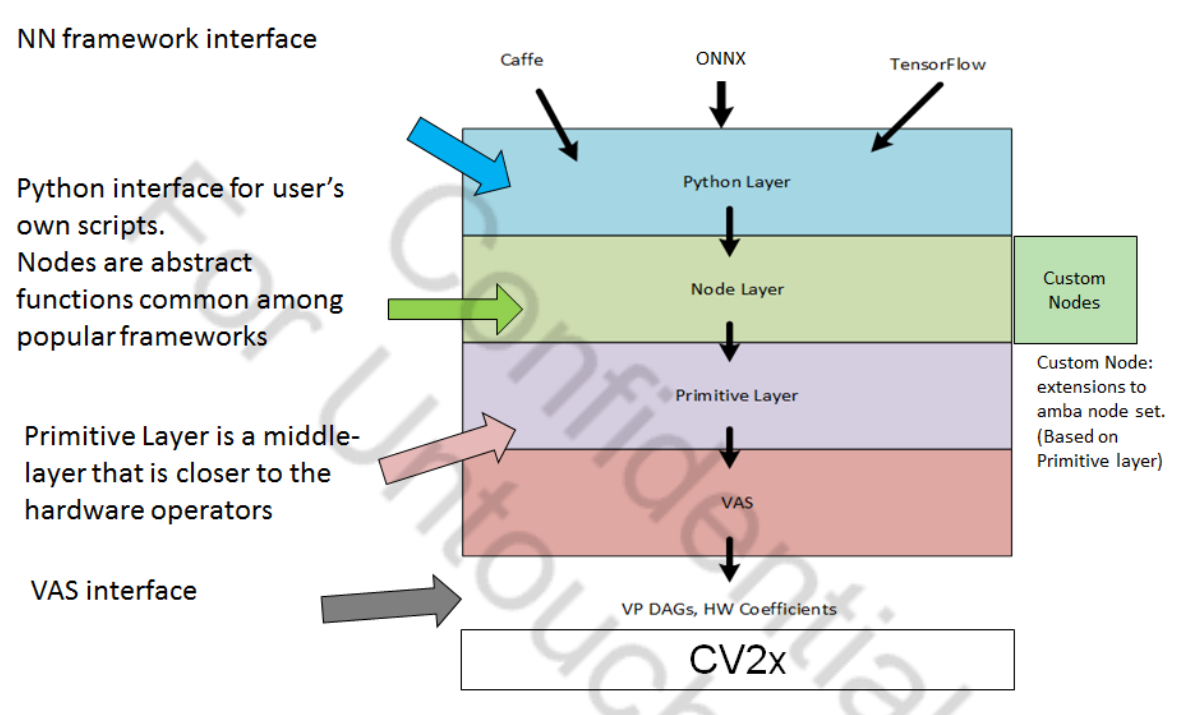
2.5 框架选择

3 CNNGen

本章描述运行CNNGen所需的环境设置和工具。

3.1 接口

下图描述了CNNGen的接口



3.3 环境设置

3.4 输入数据格式

3.5 DRA List Generation

3.6 Parsers

3.7 Caffe Parsers

3.8 TensorFlow Parsers

3.9 ONNX Parsers

3.10 ADES

3.11 cavalry\_gen

3.12 layer\_compare.py

3.13 Pre-Process Model Tools

3.14 eval\_surgery.py

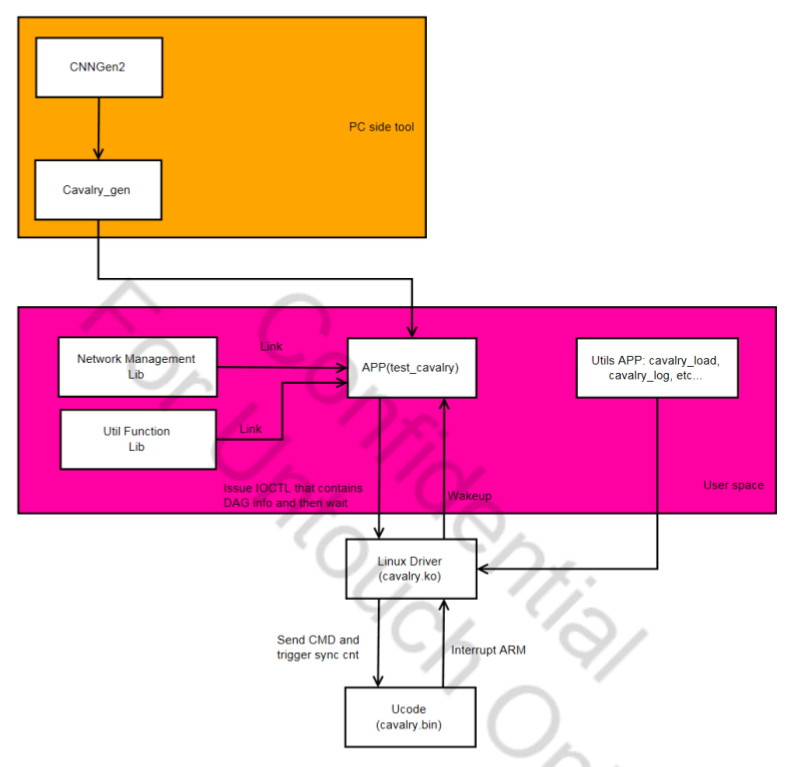
3.15 Supplementary Documents

3.16 External Tools

4 自定义节点

5 部署

下图说明了部署的流程：



5.1 准备二进制文件

用户必须准备一个能够运行Cavalry的二进制文件，下面将对此进行介绍。

5.2 文件模式

5.3 Live模式

5.5 Accuracy Tool

评估性能和准确性是Ambarella CV芯片开发的关键。性能测试显示CNN在Ambarella CV芯片上的执行速度，而精度测试显示将CNN移植到CV芯片后的精度结果。由于x86平台的架构差异，CVflow以定点数据类型（如Fix-8/Fix-16）执行CNN。因此，由于CVflow精度结果与基准精度之间存在微小差异，因此在将CNN部署到CV芯片时，有必要进行精度回归测试。

精度工具生成两种类型的网络输出。因此，它允许用户比较：

(1)原始网络输出

(2)Cavalry工具输出

5.6 CNNGen例程包

5.7 CNNGen Demos

5.8 PVANET

随着深度神经网络（DNN）的发展，一些网络采用两步结构进行目标检测，这在大多数实际应用中降低了计算成本并提高了检测精度。由于效率的原因，诸如Fast-RCNN和Faster-RCNN之类的网络在实际场景中变得流行。

本节以PVANet为例，使用Ambarella提供的工具链，在Ambarella CV2x平台上部署多步骤结构网络。以下部分介绍了PVANet网络结构和转换所需的工具，解释了执行单个网络任务的步骤，然后介绍了在Ambarella CV2x平台上PVANet的性能。

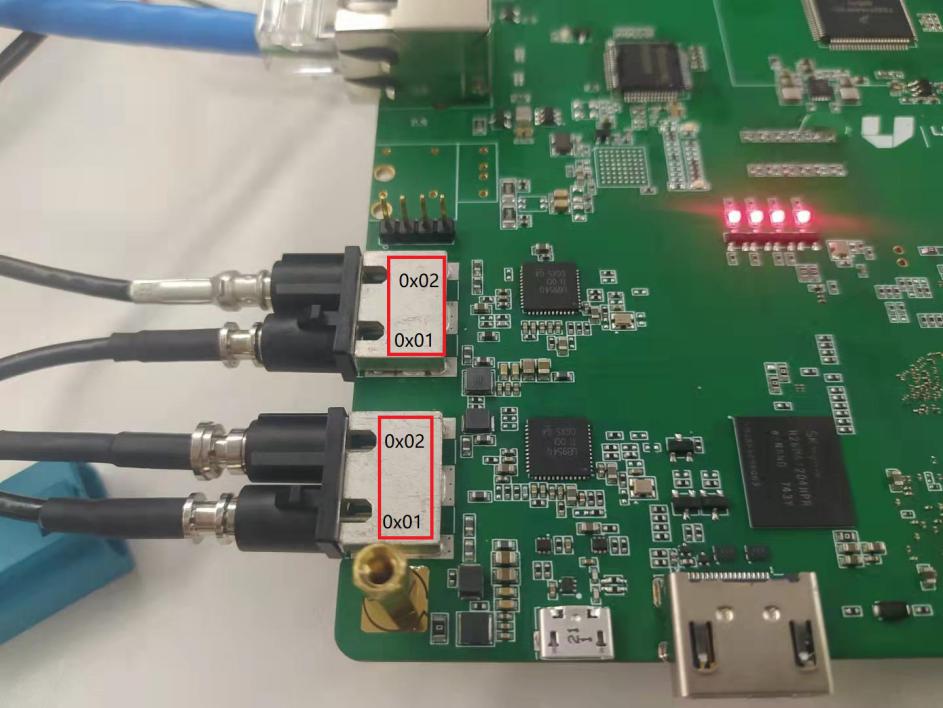
5.8.1 下载模型

git clone --recursive https://github.com/sanghoon/pva-faster-rcnn.git

1. RTSP and H.264

6.1 摄像头的区分情况

区分情况如下图所示，



6.2 rtsp\_server

rtsp\_server例程在目录下

When the RTSP Server is running, view live streams using VLC media player as an RTSP client, 也可以用rtsp的python脚本在PC机运行来显示从板子传过来的视频图像。PC机rtsp.py代码如下：

#!/usr/bin/env python

# coding=utf-8

import cv2

print(cv2.\_\_version\_\_)

cap = cv2.VideoCapture('rtsp://192.168.1.145/stream1')

print(cap)

ret,frame = cap.read()

while ret:

ret,frame = cap.read()

cv2.imshow("current frame",frame)

#cv2.imwrite('frame.jpg', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

cap.release()

Initialize the board.

CV2x # init.sh --imx274\_mipi

CV2x # test\_tuning -a &

Configure board resource through a Lua script.

Here we take CV22\_ Walnut for example,

other platform should have the similar Lua script locate in

ambarella\unit\_test\private\iav\_test\arch\_v5\scripts\.

CV2x # test\_encode --hdmi 720p --resource-cfg

/usr/local/bin/scripts/cv22\_vin0\_1080p\_linear.lua

Start the RTSP server and begin to encode.

CV2x # rtsp\_server &

CV2x # test\_encode -A -h1080p -e

rtsp\_server相关代码在shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/packages/lw\_media/source/streaming/下(需要注意的是这里的代码仅仅是形成lw\_media库.so)，里面包含：

streaming\_if.cpp

rtsp\_auth\_accounts.cpp

streaming\_server\_manager.h

streaming\_server\_manager.cpp

rtsp\_server.h

rtsp\_server.cpp

rtsp\_server的main()函数在shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/packages/lw\_media/unit\_test/test\_rtsp.cpp中，

编译后形成的是test\_rtsp可执行文件，但是shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/unit\_test/private/package\_test/arch\_v5/make.inc这个Makefile中又将test\_rtsp可执行程序链接成rtsp\_server：

ifeq ($(BUILD\_AMBARELLA\_APP\_IPCAM\_RTSP), y)

else

@-rm -rf $(UNIT\_TEST\_PATH)/rtsp\_server

@-rm -rf rtsp\_server

@-ln -s $(TEST\_RTSP\_TARGET) rtsp\_server

@-cp -dpRf rtsp\_server $(UNIT\_TEST\_PATH)/

@-rm -rf rtsp\_server

endif

故此我们才能知道文档中rtsp\_server的来源。

视频编码test\_encode可执行文件的源码路径为：

shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/unit\_test/private/iav\_test/arch\_v5/test\_encode.c

初始化板子的脚本init.sh：shg\_amba\_cv25\_sdk\_3\_0\_2/cv25\_linux\_sdk\_3.0.2/ambarella/out/cv25\_hazelnut/fakeroot/usr/local/bin/init.sh，可以通过不同的参数来使用，使用方法为：init.sh --<sensor\_name>，例如：

init.sh --imx274\_mipi

脚本运行后的打印输出示例如下：

/bin:/sbin:/usr/bin:/usr/sbin:/usr/bin/X11:/usr/local/bin:/usr/local/sbin

[ 1968.206711] ambarella-i2c e400b000.i2c: No ACK from address 0x34, 0:0!

[ 1968.230715] imx274: probe of 3-0001 failed with error -5

Default init without lens driver

Use default settings

map\_size = 0x500000, nr\_item = 4

addr\_offset = 0x00000000, filename = orccode.bin

addr\_offset = 0x00300000, filename = orcme.bin

addr\_offset = 0x003a0000, filename = orcmdxf.bin

addr\_offset = 0x003e0000, filename = default\_binary.bin

mmap returns 0x7f8ce34000

loading /lib/firmware/orccode.bin...addr = 0x7f8ce34000, size = 0x17efb3

loading /lib/firmware/orcme.bin...addr = 0x7f8d134000, size = 0x35ca8

loading /lib/firmware/orcmdxf.bin...addr = 0x7f8d1d4000, size = 0x24254

loading /lib/firmware/default\_binary.bin...addr = 0x7f8d214000, size = 0x6b9c0

===============================================

ucode (CV22) version = 2018/4/20 281098.281100

===============================================

Lua脚本说明：

Lua脚本在板子上的路径为：/usr/local/bin/scripts/

vsrc\_0 = {

vsrc\_id = 0,

mode = "1080p",

hdr\_mode = "linear", -- options: "linear", "2x" or "3x"

fps = 30,

}

chan\_0 = {

id = 0,

vsrc = vsrc\_0,

......

main = {

max\_output = {1920, 0},

input = {0, 0, 1920, 1080},

output = {0, 0, 1920, 1080},

},

second = {

max\_output = {720, 0},

input = {0, 0, 1920, 1080},

output = {0, 0, 720, 480},

},

third = {

max\_output = {1280, 0},

input = {0, 0, 1920, 1080},

output = {0, 0, 1280, 720},

},

......

pyramid = {

......

},

}

stream\_0 = {

id = 0,

max\_size = {1920, 1080},

max\_M = 1,

max\_N = 30,

fast\_seek\_enable = 0,

two\_ref\_enable = 0,

max\_svct\_layers\_minus\_1 = 0,

max\_num\_minus\_1\_ltrs = 0,

codec\_enable = 0, -- 0: H264/H265/MJPEG; 1: H265/MJPEG; 2: H264/MJPEG; 3: MJPEG

}

stream\_1 = {

id = 1,

max\_size = {1920, 1080},

max\_M = 1,

max\_N = 30,

fast\_seek\_enable = 0,

two\_ref\_enable = 0,

max\_svct\_layers\_minus\_1 = 0,

max\_num\_minus\_1\_ltrs = 0,

codec\_enable = 0, -- 0: H264/H265/MJPEG; 1: H265/MJPEG; 2: H264/MJPEG; 3: MJPEG

}

stream\_2 = {

id = 2,

max\_size = {720, 480},

max\_M = 1,

max\_N = 30,

fast\_seek\_enable = 0,

two\_ref\_enable = 0,

max\_svct\_layers\_minus\_1 = 0,

max\_num\_minus\_1\_ltrs = 0,

codec\_enable = 0, -- 0: H264/H265/MJPEG; 1: H265/MJPEG; 2: H264/MJPEG; 3: MJPEG

}

\_resource\_config\_ = {

version = 1,

channels = {

chan\_0,

chan\_1,

},

canvas = {

{

type = "encode",

size = {1920, 1080},

source = {"chan\_0.main",},

extra\_dram\_buf = 0,

},

{

type = "encode",

size = {720, 480},

source = {"chan\_0.second",},

extra\_dram\_buf = 0,

},

{

type = "prev",

size = {1280, 720},

source = {"chan\_0.third", },

vout\_id = 1,

extra\_dram\_buf = 0,

},

},

streams = {

stream\_0,

stream\_1,

stream\_2,

},

}

config配置文件在ambarella/boards/cv25\_hazelnut/config目录下。这里面有针对Linux内核的config配置，也有针对应用的config配置。

通过RTSP在PC机上存储（H.264/H.265）板子上摄像头的图像：

1. init.sh --imx274\_mipi //Initialize the board

CV2x # test\_tuning -a & //应该是ISP部分的3A处理

CV2x # test\_encode --hdmi 720p --resource-cfg

/usr/local/bin/scripts/cv25\_vin0\_1080p\_linear.lua

(3)

rtsp\_server &

test\_encode -A -h 1080p -e //Start the encoding stream(s) in H.264/H.265 format.

test\_encode -A -h 1080p -e中的-h选项为H.264编码，-H选项为H.265编码，-m为MJPEG编码。根据MJPEG、H.264、H.265的标准得知，H.265的编码最流畅，延时最小。

可以在Windows系统上打开VLC。另外，PC机的视频采集查看还可以通过另外的方式：python脚本实现。

如果是4路视频，则RTSP的地址如下：

rtsp://10.0.0.2/stream1 identifies the first stream.

rtsp://10.0.0.2/stream2 identifies the second stream.

rtsp://10.0.0.2/stream3 identifies the third stream.

rtsp://10.0.0.2/stream4 identifies the fourth stream.

1. Demo:Run CNN

OpenNet和BiSeNet网络是用安霸的数据集训练而得到的，其他模型是通过公共数据集训练得到的。

1. DRAM System

DRAM系统包括4个部分：Linux, IAV, DSP和CV。The Linux partition operates the

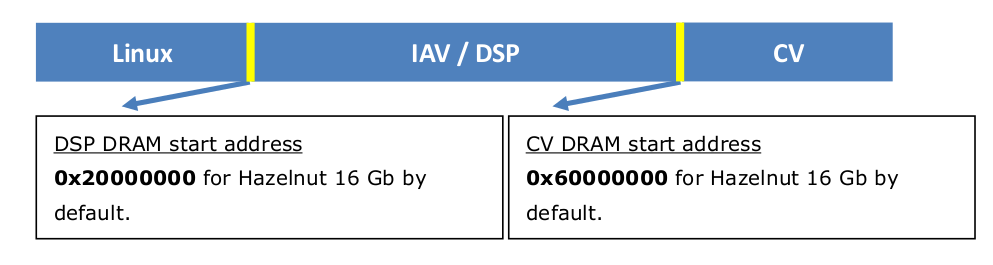
Linux system; the IAV partition is shared by the IAV driver and DSP; the DSP partition is used by DSP;and the CV partition is used by the Vector Processor (VP).

For 16 GB (2 GB) DDR on the CV2x:

Linux memory size = DSP DRAM start address – 0 = 0x20000000 = **512 MB**

DSP CMA memory pool size = CV DRAM start address - DSP DRAM start address = 0x60000000 -0x20000000 = **1 GB**

CV CMA memory pool size = 0x80000000 - CV DRAM start address = 0x80000000 - 0x60000000 =**512 MB**

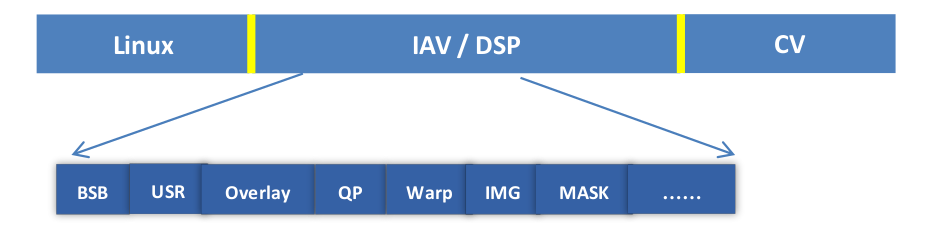


8.1 Linux DRAM

在分给Linux的512MB中，内核的启动地址为0x280000，0-0x280000（2560KB）是给AMBoot的。

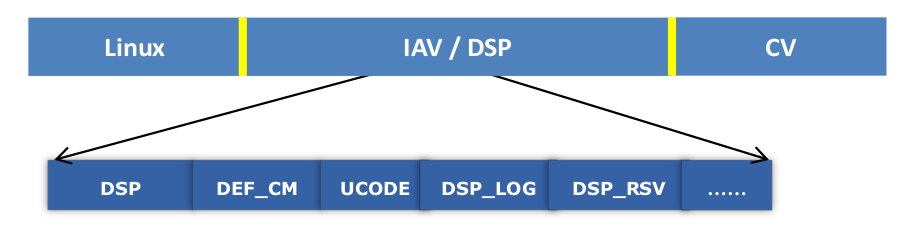
8.2 IAV DRAM

IAV DRAM4KB字节对齐。



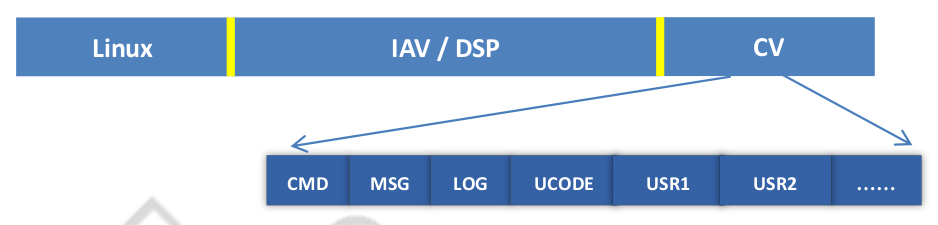
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 指令和使用方法 | 是否可以重新设置大小 | 默认大小 | 是否必需 |
| BSB | Bit stream buffer stores the encoding data (such as H.265, H.264 and MJPEG).The size depends on the total stream bitrate and bit stream data  cache time. | Yes | 10 MB | 必需 |
| USR | User data buffer is used to encode from the raw data (both YUV and RGB).Its size depends on raw data size and number. | Yes | 0 MB | 可选 |
| MV | Motion vector buffer | Yes | 0 MB | 可选 |
| OVERLAY | Overlay buffer is used for OSD.Its size depends on overlay area size and number. | Yes | ~8 MB | 可选 |
| QPMATRIX  QPMATRIX\_RAW | QP ROI matrix buffer  QP matrix raw buffer | Yes | ~8 MB | 可选 |
| WARP | The warp buffer is used for multi-  region Dewarp and LDC | Yes | ~4MB x N(N: cChannel number) | 可选 |
| QUANT | JPEG quant matrix table is used to  improve the MJPEG quality | No | 8 KB | 必需 |
| IMG | Image buffer used for 3A | Yes | (5 x N + 1) MB(N: cChannel number) | 必需 |
| MASK | Single-color / multi-color privacy  mask buffer.  The size depends on the maximum supported VIN resolution.Single-color: (maximum VIN resolution / 8) x N  Multi-color: Maximum VIN resolution x N  (N: Channel number) | Yes | Single-color: 10MBx N  Multi-color: 0 MB(N: channel number) | 可选 |
| INTRA\_PB | Intra playback buffer for decoder(only in the decode mode) | Yes | 0 MB | 可选 |
| BLEND\_ALPHA | Alpha blend buffer for the cross-channel stitching.  The size depends on stitching area size and number. | Yes | 0 MB | 可选 |
| PYRAMID\_POOL | Pyramid buffer pool for the pyramid manual-feed mode.  The size depends on the pyramid buffer size and number. | Yes | 0 MB | 可选 |
| CANVAS\_POOL | Canvas buffer pool for the canvas manual-feed mode.  The size depends on the canvas buffer size and number. | Yes | 0 MB | 可选 |
| BLUR | Blur buffer for the stream privacy protection.  The size depends on the maximum blur area size. | Yes | 0 MB | 可选 |
| ARB\_BLUR | Arbitrary blur buffer for the stream privacy protection.  The size depends on the maximum blur area size and maximum stream number. | Yes | 0 MB | 可选 |
| HEVC\_SCALELIST | HEVC scalelist buffer for HEVC scale.  The size depends on the scalelist number, set as 1 KB by default. | Yes | 0 MB | 可选 |
| MULTI\_COLOR\_MASK | MULTI\_COLOR buffer for HEVC scale.  The size depends on the color mask pitch, height, clut size, and maximum channels. | Yes | 0 MB | 可选 |
| INT\_BSB | Internal bit stream buffer stores the encoding data (such as H.265,H.264, and MJPEG).  The size depends on the total stream bitrate and bitstream data cache time. | Yes | 0 MB | 可选 |
| EXTRA\_RAW | Extra raw buffer for store the embedded extra raw data. The size depends on the customized extra raw data size. | Yes | 0 MB | 可选 |

8.3 DSP Partition



DSP partition = DSP buffer + 6 MB (DEF\_CMD + UCODE + DSP\_LOG + DSP\_RSV ...)

8.4 CV Partition



附录A

安霸2021年1月份宣布推出CVflow®系列最新芯片CV5，该款人工智能视觉处理器可支持8K视频录制或4路独立图像输入的4K视频流录制。新SoC芯片CV5将推动智能汽车摄像系统、消费级无人机、运动相机和 360°全景相机，以及机器人视觉系统的进一步发展。安霸半导体CVflow AI引擎与双核Arm®A76处理器的完美集成为各种主流人工智能算法提供卓越性能。CV5拥有高性能图像信号处理器（ISP），可为视频编码优化以提高人眼观感，同时为机器视觉算法优化以提升准确度。CV5采用5纳米先进制程，拍摄8Kp30视频所需功耗低于2瓦。

在汽车视频流远程处理应用中，CV5可支持多路视频流编码，涵盖前置ADAS、驾驶员监控、车舱监控及侧视摄像头。借助于CVflow人工智能引擎，CV5可同时运行高级驾驶员辅助系统（ADAS）算法（如车道偏离、前方碰撞预警）以及驾驶员监控算法（如驾驶员疲劳驾驶检测）等。多路高分辨率视频捕捉与先进的人工智能处理的完美结合，可支持ADAS摄像头在远距离以更高精度识别目标物体。

CV5与安霸半导体其它CVflow系列芯片共享同一套SDK和计算机视觉算法优化（CV）工具，简化了各个价格区间和不同性能选项的相机开发流程。一套完整的机器视觉工具包括了编译器、调试器，并支持行业标准的PyTorch、ONNX、Caffe和TensorFlow等机器学习框架，以及卷积神经网络（CNN）性能优化完全指南，可帮助客户将自己的神经网络快速移植到CV5上。

CV5 芯片关键特征：

——支持 DNN 的 CVflow架构；

——双核 1.6GHz Arm®Cortex®-A76 附带 NEON™DSP扩展和 FPU；

——高速 SLVS-EC, MIPI-CSI (C/D PHY）接口，可接入多达14个摄像头；

——多通道 ISP，处理能力高达 8KP60；

——原生支持 RGGB、RCCB、RCCC、RGB-IR 和单色传感器；

——多重曝光高动态范围（HDR）处理；

——实时硬件加速的鱼眼镜头畸变校正（LDC）；

——支持多码流，高达 8Kp60 的 AVC 和 HEVC 编码；

——多种外设接口，包括 4 通道 PCIe、CAN FD、千兆网口、USB 3.1（主/从模式），三个 SD 卡控制器，MIPI DSI/CSI-2和HDMI输出；

——支持高达32GB的LPDDR4x / LPDDR5 / LPDDR5x，64位数据总线；

——可实现设备信息安全包括安全启动，支持TrustZone®、TRNG、OTP、内存隔离，内存加密和虚拟化；

——5nm制程；

——16×16 FBGA封装，球间距0.5mm。