Ascend 310

DDK 样例使用指导

文档版本 01

发布日期 2019-03-13

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 DDK 样例概览

表 1-1 DDK 样例列表

模块	子目录	功能简介	使用说明
customop	customop_app	测试数据以及main、Makefile文件写作样例。	-
	customop_caffe_demo	caffe框架的非IR方式进行算子插件开发的代码样例。	
	customop_caffe_ir_demo	caffe框架的IR方式进行算子插件开发的代码样例。	
	customop_tensorflow_demo	tensorflow框架的算子插件开发的代码样例	
	python	caffe_reduction_layer算子代码，完成算子插件代码开发。	
dvpp	-	使用DVPP调用VDEC、VPC、JPEGD、JPEGE、VENC、CMDLIST6个子模块的样例代码。	请参见**6 dvpp**
cert_path	KeyManager.py	用于进行密钥加解密。	请参考《Ascend 310 模型加解密使用指导》

模块	子目录	功能简介	使用说明
detection_ssd_asic	-	asic环境用例，ssd检测网络模型，生成置信度文件。	请参见**5 Detection ssd**
classify_net_asic	resnet-18	asic环境用例，分类网络模型的集合，生成置信度文件。	请参见**4 classfiy net asic**
	resnet-50		
	resnet-101		
	resnet-152		
	resnext-50		
	resnext-101		
	vgg16		
	vgg19		
fasterrcnn_vgg16_asic	----	asic环境用例，faster-rcnn检测网络模型	请参见**3 faster rcnn_vgg16_asic**
hiaiengine	----	自定义engine开发样例。	请参见《HiAI Engine开发指导》。
tvm	----	Tensor Engine工程样例，包含sign以及reduction单算子的样例代码。	请参见**2 tvm**

2 tvm

2.1 工程描述

本Sample是Tensor Engine工程样例，包含单算子sign以及reduction算子的样例代码，以及样例数据生成脚本。

用户可以基于此样例工程更快的进行TE算子的开发。

2.2 工程路径

Sample所在路径：$HOME/tools/che/ddk/ddk/sample/tvm。

说明

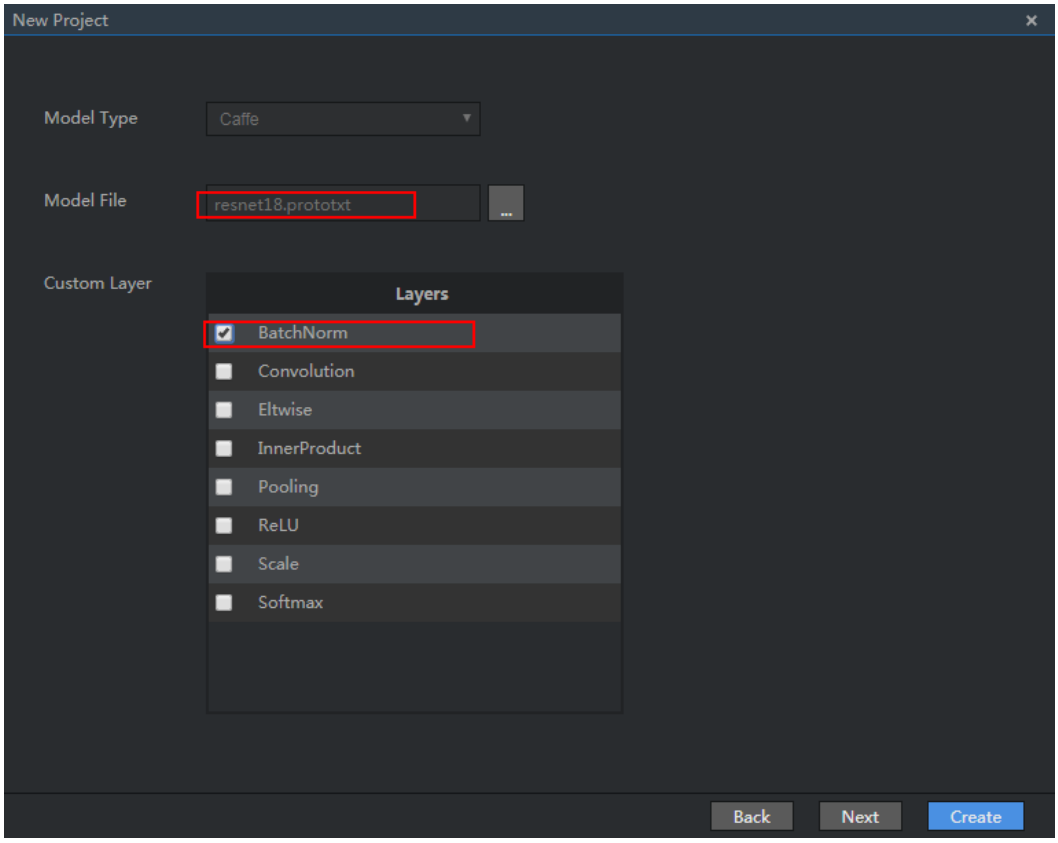
$HOME/tools为DDK的默认安装路径。

2.3 工程导入

步骤1 在Mind Studio新建工程界面选择“Operator Project > Tensor Engine Project”工程类型，在“Source Code From”选择“Local (WebServer)”，“Local File Path”中选择“$HOME/tools/che/ddk > ddk > sample > tvm”。

步骤2 单击“Next”，进入模型配置界面，如图2-1图2-1所示。

图 2-1 网络模型配置



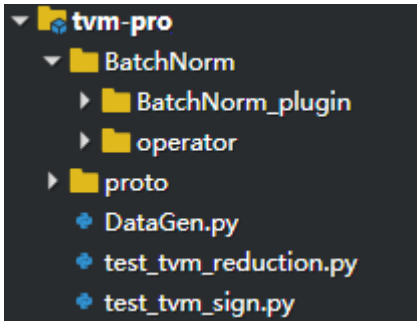
“Model File”导入一个模型文件，“Custom Layer”中选择第一层。

说明

此处可以导入任意工具转化后的模型文件，供后续自定义算子使用其框架。

步骤3 单击“Create”，通过样例导入完成Tensor Engine工程的创建，如图2-2所示。

图 2-2 TVM 工程示例



代码目录结构说明如表2-1所示。

表 2-1 代码目录结构说明

目录	说明
BatchNorm_plugin	自定义算子插件代码目录。 此sample中只进行算子的实现，不涉及算子插件的实现。
operator	算子的实现代码目录
proto	工具内置的算子定义proto文件
DataGen.py	样例数据生成脚本
test_tvm_reduction.py	reduction算子的实现样例
test_tvm_sign.py	sign算子的实现样例

----结束

2.4 样例数据生成

在Mind Studio的服务器后台手动执行样例工程中的数据生成脚本。

步骤1 进入新建工程路径。

例如新建的样例工程名为tvm-pro，则进入的路径为：

cd /projects/tvm-pro

步骤2 执行DataGen.py脚本，生成样例数据。

python DataGen.py

生成以下八个数据文件，sign开头的作为test_tvm_sign.py中算子的输入数据和验证数据，reduction为减法运算的数据，数据文件说明如表2-2所示。

表 2-2 数据文件说明

数据文件	说明
reduction_input_2_3_4_sum_axis_1.data	reduction算子的二进制格式输入数据文件。
reduction_input_2_3_4_sum_axis_1.txt	将reduction算子的二进制格式文件以txt文件的方式显示出来，方便用户查看结果。
reduction_output_2_3_4_sum_axis_1.data	reduction算子二进制格式输出数据验证文件，用于验证算子运行后的输出结果是否正确。
reduction_output_2_3_4_sum_axis_1.txt	用户可见的reduction算子的输出数据文件，用处同上。
sign_input_2_4_cce.data	sign算子的二进制格式输入数据文件。

数据文件	说明
sign_input_2_4_cce.txt	将sign算子二进制格式文件以txt文件的方式显示出来，方便用户查看结果。
sign_output_2_4_cce.data	sign算子二进制格式输出数据验证文件，用于验证算子运行后的输出是否正确。
sign_output_2_4_cce.txt	用户可见sign算子的输出数据文件，用处同上。

以上数据尺寸均为：(2, 4)两行四列。

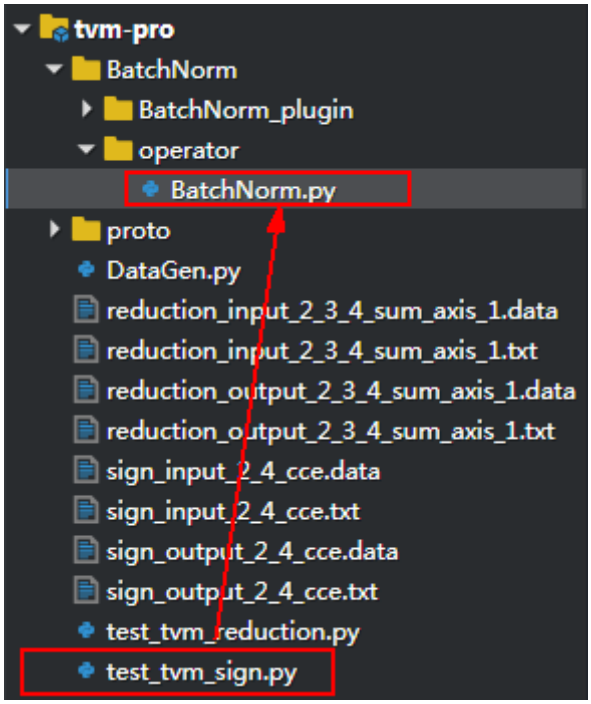
----结束

2.5 代码实现

本示例以编辑运行样例工程中的sign算子为例进行操作说明。

1. 将test_tvm_sign.py中的内容拷贝覆盖到BatchNorm.py文件中，如图2-3所示。

图 2-3 拷贝示例



说明

“BatchNorm_plugin > operator”中的“BatchNorm.py”为编译的入口文件，所以把需要编译的“test_tvm_sign.py”中的代码拷贝到“operator”的“BatchNorm.py”中。

2. 在替换后的BatchNorm.py文件中搜索“kernel name”，将“kernel name”的值修改为文件名“BatchNorm”，如图2-4所示

图 2-4 编辑修改 BathNorm.py

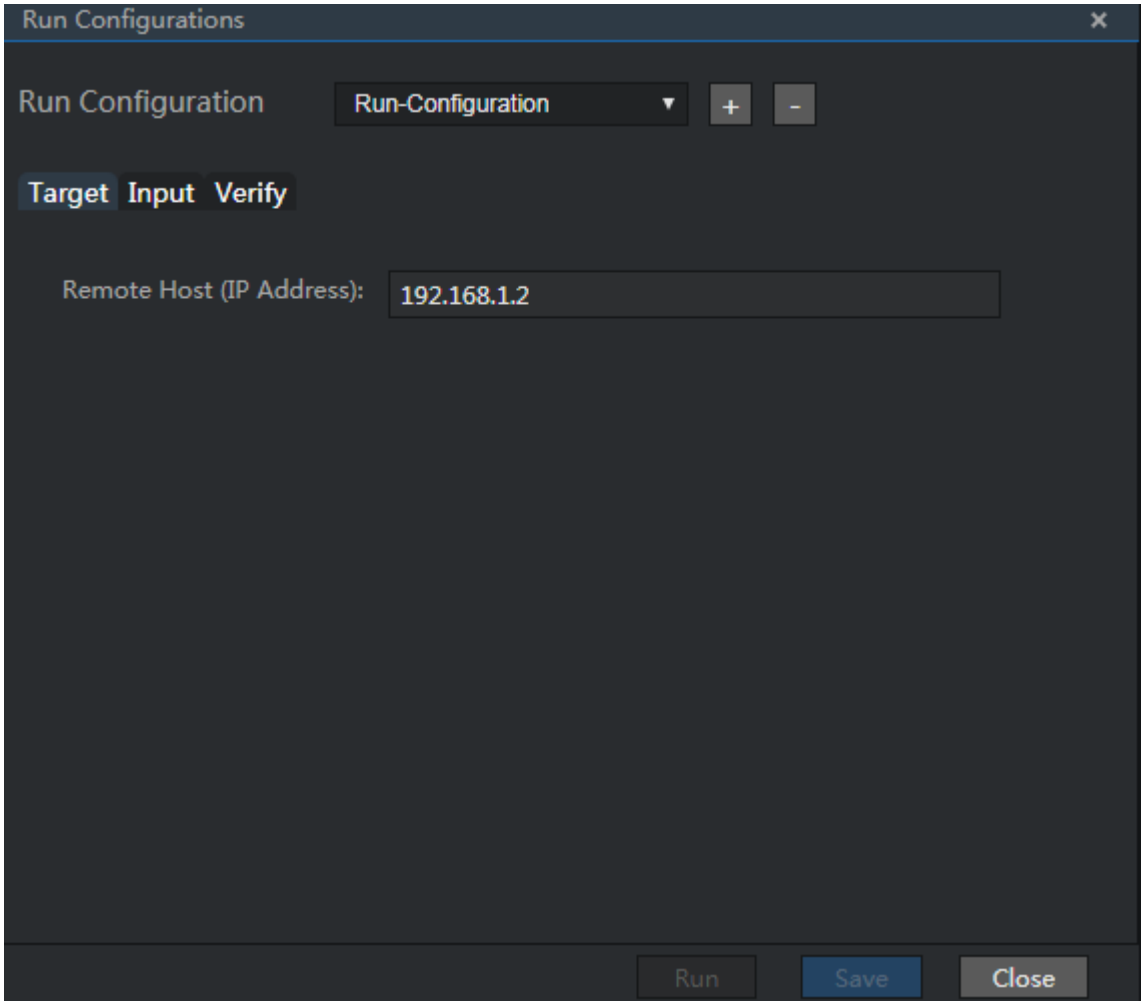
```
import te.lang.cce
import topi

def sign_cce(data_shape, dtype, kernel_name = "BatchNorm", need_build = False, need_print = False):
    """
```

2.6 工程编译

TE算子在工程运行的时候会自动编译，本示例介绍样例工程中sign算子的运行配置。

步骤1 选中样例工程，在菜单栏选择“Run > Edit Run Configuration...”，弹出运行窗口配置界面，如图2-5所示。

图 2-5 运行配置界面



步骤2 配置参数说明。

TE构建配置界面中总共包括三部分配置：

表 2-3 TE 构建配置参数说明

配置参数	参数说明
Target配置	配置运行目标信息。 ● 对于ASIC类型工程，请填写host侧的IP地址。 ● 对于Atlas DK类型工程，请填写开发板的IP地址。
Input配置	描述算子的输入输出数据信息。 ● “Custom Operator Name” 单算子名的配置，工程新建后，名称会自动显示。 ● INPUT_PARAM：输入数据的描述，可根据实际输入数据个数添加，最多允许20条输入数据。 Input0：第一个输入数据。 File Path：输入数据文件路径，选择**2.4 样例数据生成**构造的二进制输入数据文件。 ● OUTPUT_PARAM：输出数据的描述，可根据实际输出数据个数添加，最多允许10条输入数据。 Output0：第一个输出数据。 – File Path：输出数据文件，默认为相对于output文件夹下的out0.data文件。 – Data Size： 输出数据大小，单位字节。 – Data Type：输出数据类型，选择float16或者float32。
Verify配置	该配置页下的参数用于校验Run配置中输出结果和用户预期结果是否一致。此页签参数为可选参数，如果不配置，则运行时不校验输出数据正确性。 配置项意义如下，配置示例如**图2-7**所示。 ● EXPECT_PARAM：用户预期的结果，用于和输出结果检验，数据个数需要与输出数据个数相同，最多允许10条输入数据。 Expect0：第一个验证数据。 File Path：选择验证数据文件（不允许用户手工编辑文件路径）。 ● Precision Deviation：精度偏差，表示单数据相对误差允许范围，数值范围为[0，1），精度偏差越小表示精度越高（默认为0.2）。 ● Statistical Discrepancy：统计偏差，整个数据集中精度偏差不满足门限的数据量占比，数值范围（0，1），统计偏差越小表示精度越高（默认为0.2）。

图 2-6 Input 配置示例

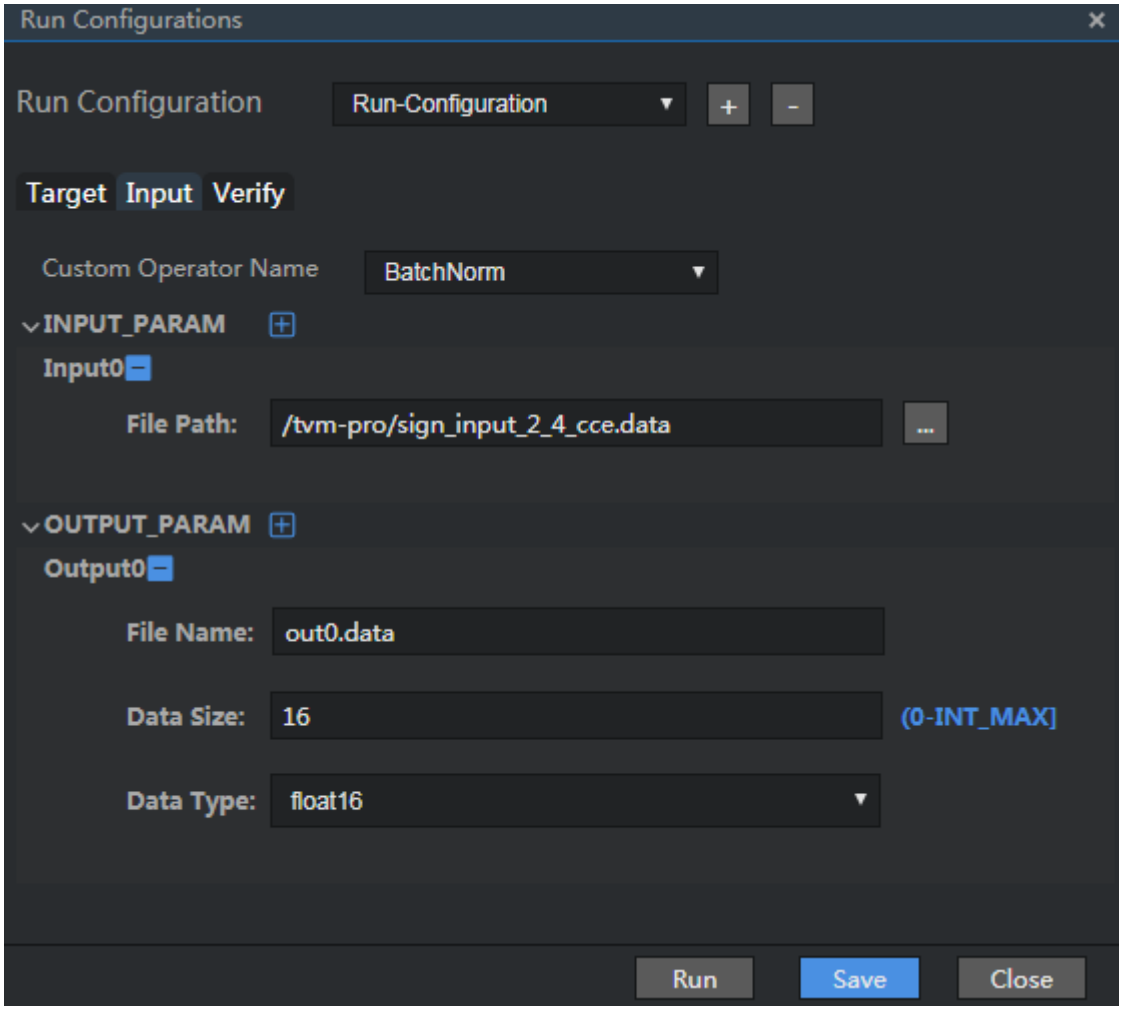
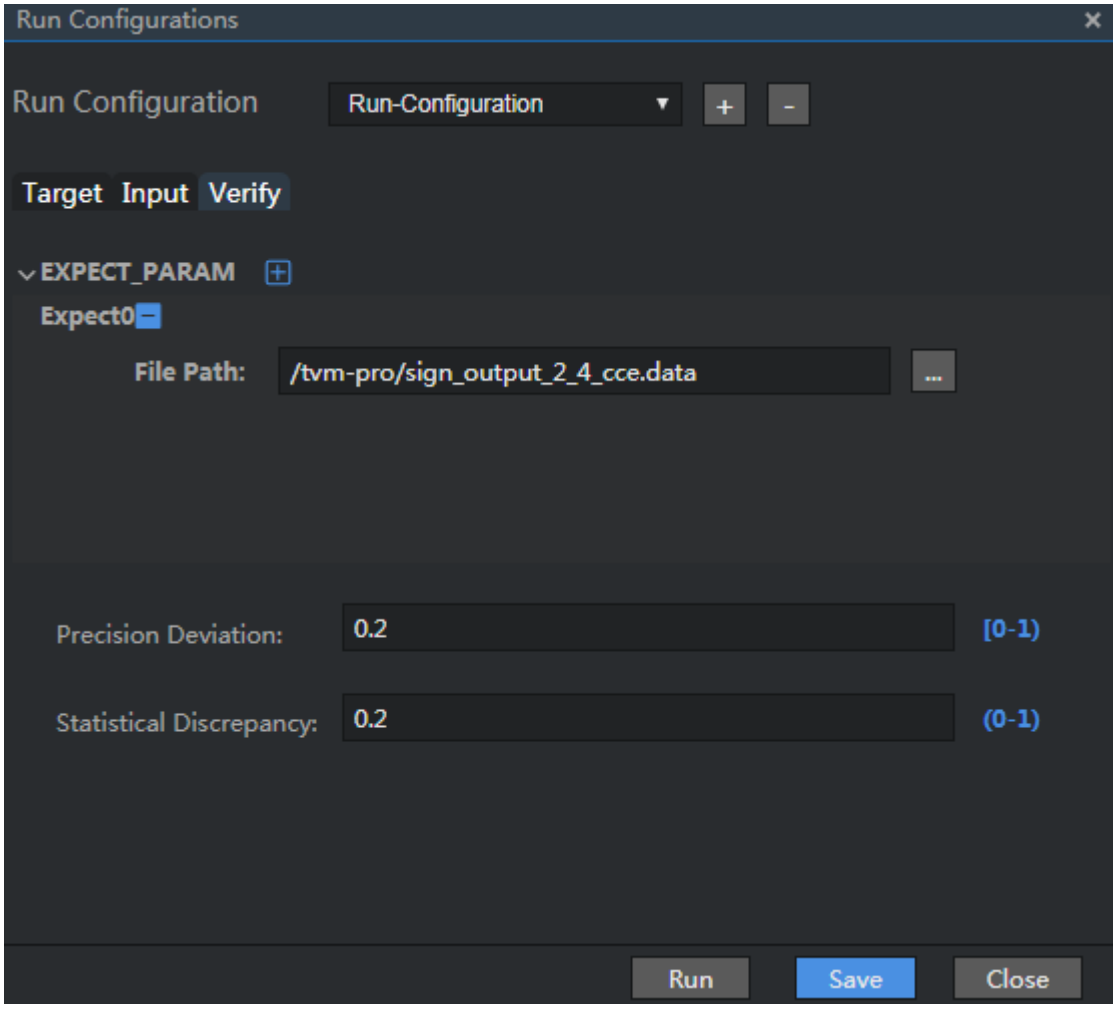


图 2-7 Verify 配置示例



说明

Verify中的EXPECT_PARAM中的个数要与Run配置中的OUTPUT_PARAM中的个数相同，且Verify配置中的配置项不存在空值，Tensor Engine运行时才会进行校验操作。

步骤3 完成配置后，单击“Save”保存。

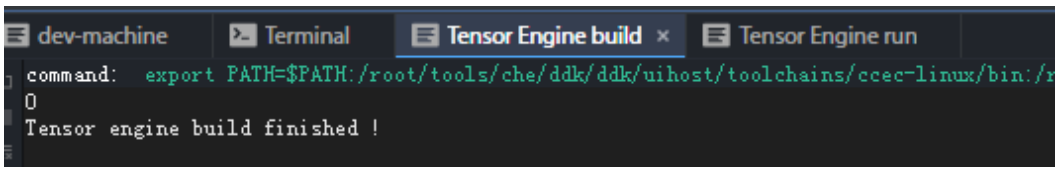
----结束

2.7 工程运行

保存后，单击配置界面中的“Run”，运行算子工程。

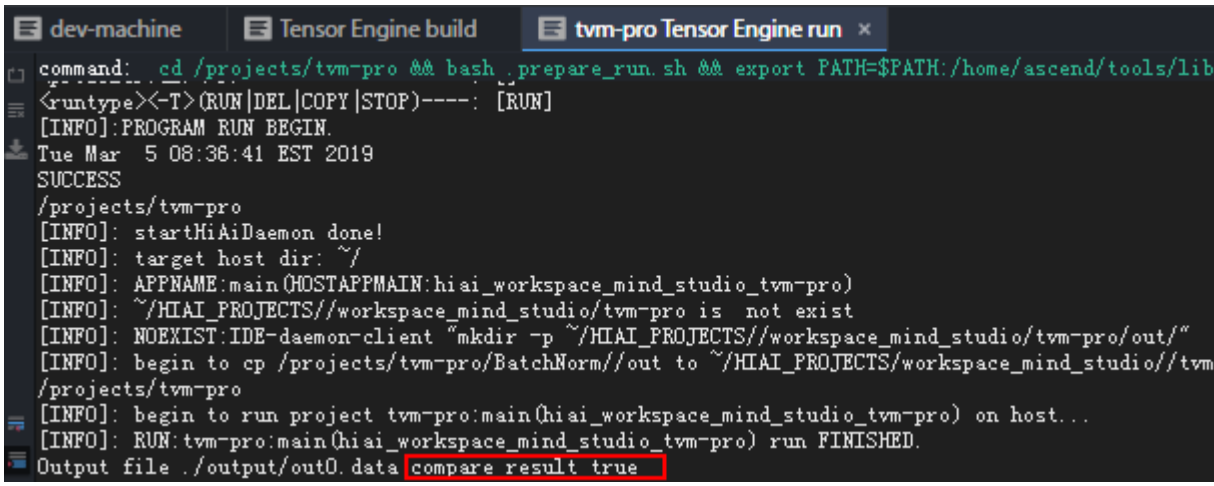
开始构建、运行后，Mind Studio界面依次出现两个窗口，分别是“Tensor Engine build（编译过程）”和“Tensor Engine run（运行过程）”，构建成功如图2-8所示。

图 2-8 Build 成功



进行文件校验后，运行成功终端输出如图2-9所示。

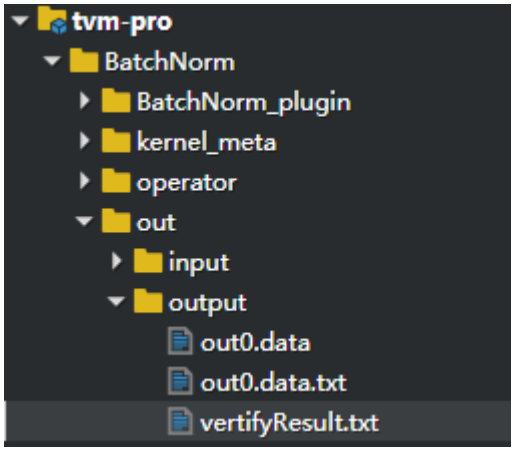
图 2-9 运行成功



2.8 输出验证

运行成功后，在output文件下会生成输出文件*out0.data.txt*与校验结果文件vertifyResult.txt文件，如图2-10所示。

图 2-10 输出文件



3 fasterrcnn_vgg16_asic

3.1 工程描述

fasterrcnn_vgg16_asic 样例工程是基于 fastrcnn(vgg16) 检测网络编写的示例代码，该示例代码主要实现了输入BGR数据给到推理模块， 并输出检测目标的框以及分类的置信度。

3.2 工程路径

Sample所在路径：$HOME/tools/che/ddk/ddk/sample/fasterrcnn_vgg16_asic。

由于fasterrcnn caffe模型的权重文件太大，所以未包含在产品包中，要运行此示例，请先自行下载fasterrcnn_vgg16网络的模型文件及权重文件，并参考**3.4 数据输入**将原始网络模型转化为适配Ascend 310的模型。

3.3 工程导入

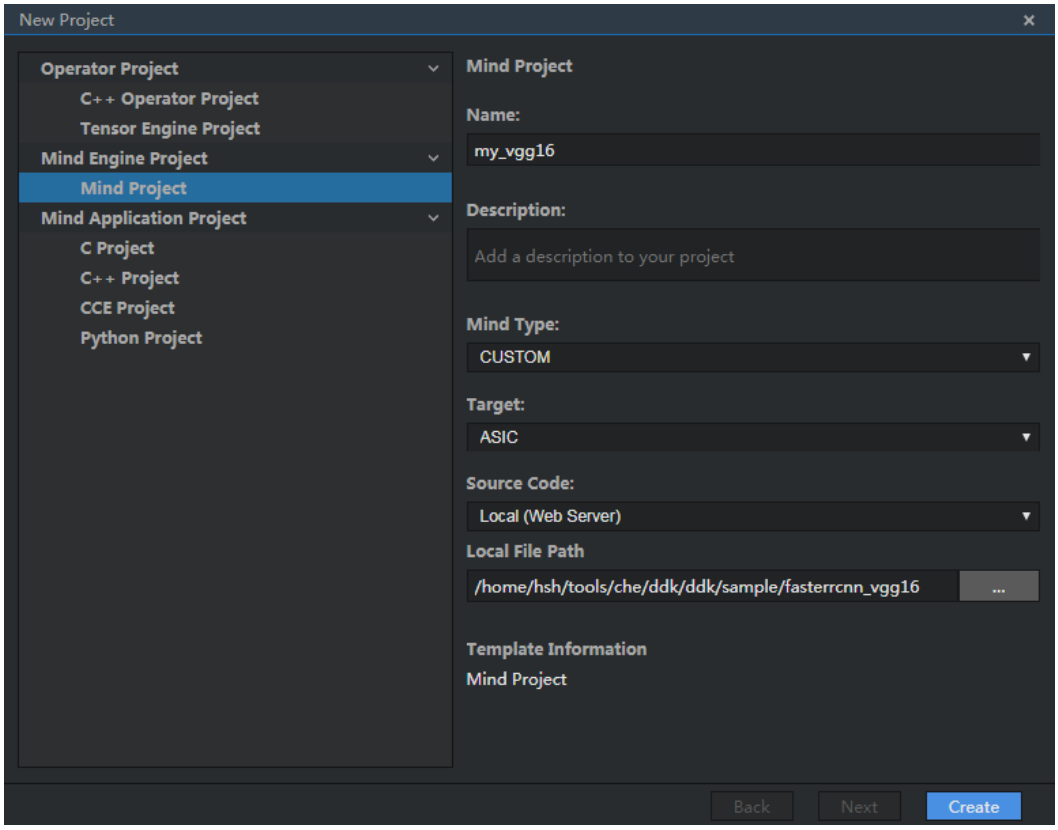
步骤1 在Mind Studio新建工程界面选择“Mind Engine Project > Mind Project”工程类型。

- “Name” 填入“my_vgg16”。
- “Mind Type”选择“CUSTOM”。

- “Target”选择“ASIC”。
- “Source Code From”选择“Local (WebServer)”。
- “Local File Path”中选择“$HOME/tools/che/ddk > ddk > sample > fasterrcnn_vgg16_asic”。

单击“Create”完成样例工程导入。

图 3-1 导入工程示例



步骤2 样例工程导入后，工程代码结构如图3-2所示。

图 3-2 Fasterrcnnvgg16 工程代码结构

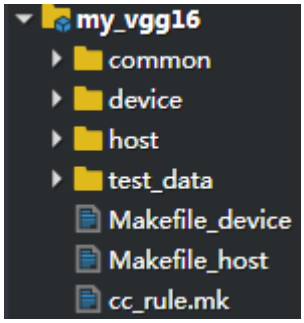


表 3-1 代码结构说明

一级目录	二级目录	说明
common	common_inc	通用头文件夹
device	inner_inc	Device头文件夹
	src	Device源文件夹
host	inner_inc	Host头文件夹
	src	Host源文件夹
test_data	config	配置文件
	data	待分类的BGR测试数据
	model_gen	模型文件及模型转化时用到的AIPP配置文件及算子映射配置文件
Makefile	-	编译脚本
cc_rule.mk	-	Makefile编译脚本所加载的文件

----结束

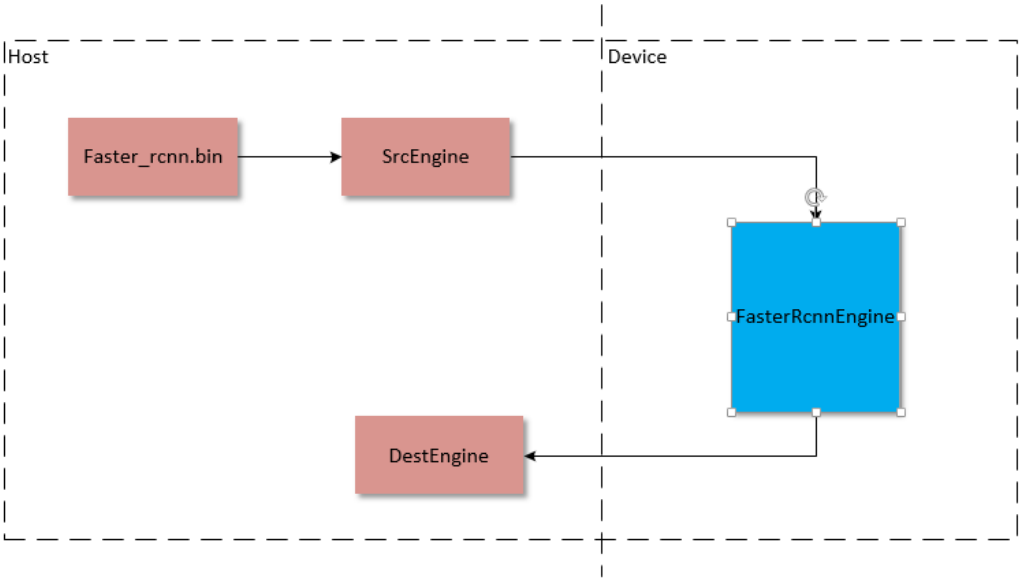
3.4 数据输入

参数	说明
待分类的BGR测试数据	工程目录下/test_data/data/faster_rcnn.bin。（180*300）

参数	说明
faster-rcnn(vgg16)模型	模型名称：FasterRCNN-vgg16.davinci.om 转化方法： 将 faster_rcnn_vgg16 caffemodel 文件放到 test_data/model_gen 目录，并改名为 FasterRCNN_VGG16_180x300_do.caffemodel 进入 test_data/model_gen 目录 1. 执行命令：su root 2. 执行命令：export LD_LIBRARY_PATH=/<安装目录>/tools/che/ddk/ddk/uihost/lib 3. 执行命令：/<安装目录>/tools/che/ddk/ddk/uihost/bin/omg --model=./FasterRCNN_VGG16_180x300_do.prototxt --weight=./FasterRCNN_VGG16_180x300_do.caffemodel --framework=0 --aipp_conf=*aipp_nv12_img.cfg* --output=./FasterRCNN-vgg16.davinci --op_name_map=*./model_face.txt* 其中FasterRCNN_VGG16_180x300_do.prototxt为网络模型文件，aipp_conf为AIPP配置文件，op_name_map是算子映射配置文件（包含DetectionOutput网络时需要指定），这三个文件可从工程目录下的test_data/model_gen目录下获取。 详细的参数说明可以参考《Ascend 310模型加解密使用指导》。 将转化好的 FasterRCNN-vgg16.davinci.om 文件放在测试目录（与 my_vgg16 可执行文件在同一级目录） 说明：也可参考《Ascend 310 Mind Studio基本操作》通过Mind Studio前台界面进行转化。
Graph配置文件	工程目录下/test_data/config/sample.prototxt

3.5 代码实现

该用例主要分为三个Engine（SrcEngine-FasterRcnnEngine-DestEngine）, 详细关系如下图：



Engine名称	源码文件
SrcEngine	工程目录下/host/src/src_engine.cpp
FasterRcnnEngine	工程目录下/device/src/faster_rcnn_engine.cpp
DestEngine	工程目录下/host/src/dest_engine.cpp

3.6 工程编译

无需编写Makefile，通过Mind Studio操作界面提供的编译界面进行编译，详细操作步骤如下。

步骤1 选中要进行配置工程结构的工程，单击依次选择“Build > Edit Build Configuration”，弹出自定义工程的配置页面。

步骤2 配置自定义工程。

自定义配置工程结构分为**Main**、**Host**、**Device**三部分。

每个.so文件需要根据sample.prototxt文件中的每个Engine的side值在相应页签配置。

Main、host与Device的参数描述如表3-2所示。

表 3-2 配置说明

配置项	说明
Main	可执行文件的生成，点击 Build -> Edit Build Configurations -> Main，Files 选择 host 目录下的cpp文件，即 host/src 目录下的所有 cpp 文件。 Build Options 选项，Include Path: 需要填入 common 目录和 host 目录：<project_path>/common <project_path>/host 其中 <project_path> 可以通过左侧工程目录上右击->Show References 查看。多个 include 目录以空格分开。 举例：“/projects/my_vgg16/common /projects/my_vgg16/host” 保存配置后生成CMakeLists.txt文件（路径：工程根目录下build文件夹）。 编译成功生成可执行文件 my_vgg16（路径：out文件夹）。
host	添加需要运行在host侧的Engine，并填写Engine的名称。 本样例此页签无需配置。
Device	device lib 库生成，点击 Build -> Edit Build Configurations -> Device，Engine Name 填入"faster_rcnn_vgg16"（此为生成的 lib so 文件名，不可与其它生成的文件重名），Files 选择 device 目录下的 cpp 文件，即 device/src 目录下的所有 cpp 文件。 Build Options 选项，Include Path: 需要填入 common 目录和 device 目录：<project_path>/common <project_path>/device 其中 <project_path> 可以通过左侧工程目录上右击->Show References 查看。多个 include 目录以空格分开 举例：“/projects/my_vgg16/common /projects/my_vgg16/device” 保存配置生成与Engine Name同名的文件夹，包含CMakeLists.txt文件（路径：工程根目录下build文件夹）。 编译成功生成对应的.so文件（路径：out文件夹）。 **说明** .so文件需要签名，防止文件被篡改，签名方法: 1. 由openssl工具中的RSA_generate_key接口生成公钥pub.pem+私钥pri.pem，密钥长度建议2048bit以上。 2. 通过私钥，利用SHA256算法对.so文件进行签名，生成.so.signature签名文件，与原.so文件放在同一目录。 3. 公钥需要由SetPublicKeyForSignatureEncryption接口传给HiAI Engine。接口信息请参见《HiAIEngine API参考》。

三个页签的配置依据是根据Engine的so文件需要放置在哪一侧运行。本样例中SrcEngine和DestEngine配置在Main页签；faster_rcnn_vgg16配置在Device页签，编译配置示例下图所示。

图 3-3 Main 页签配置示例

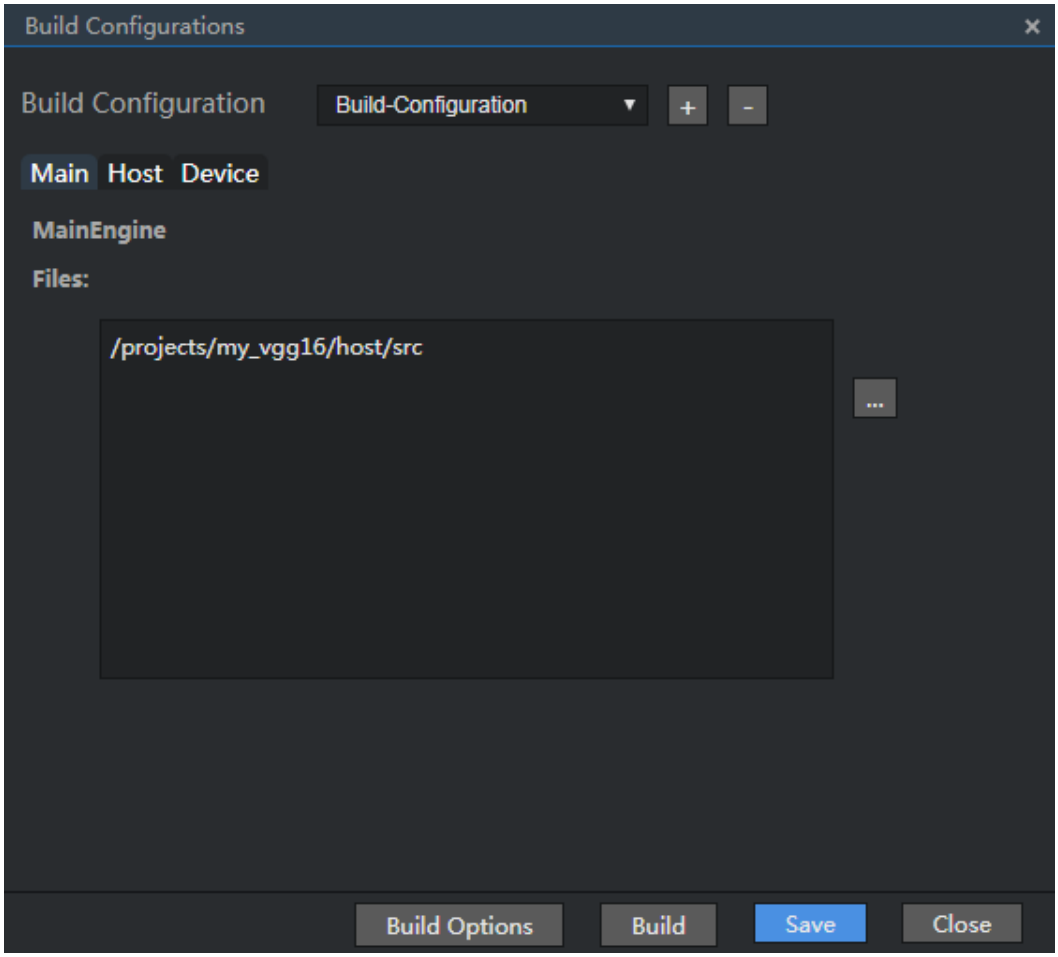
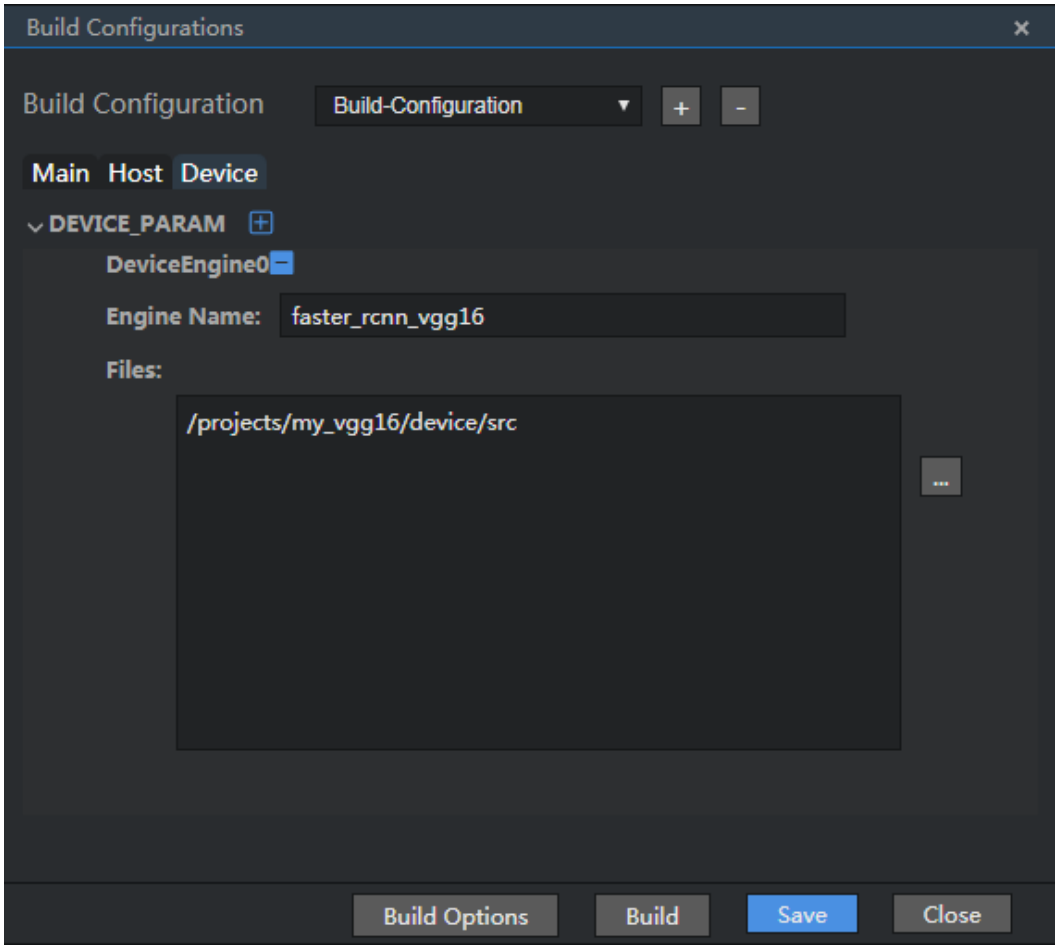


图 3-4 Device 页签配置示例



单击“Build COnfiguration”窗口中的“Build Options”，弹出如图3-5所示界面。

图 3-5 Build Options （Main）配置界面

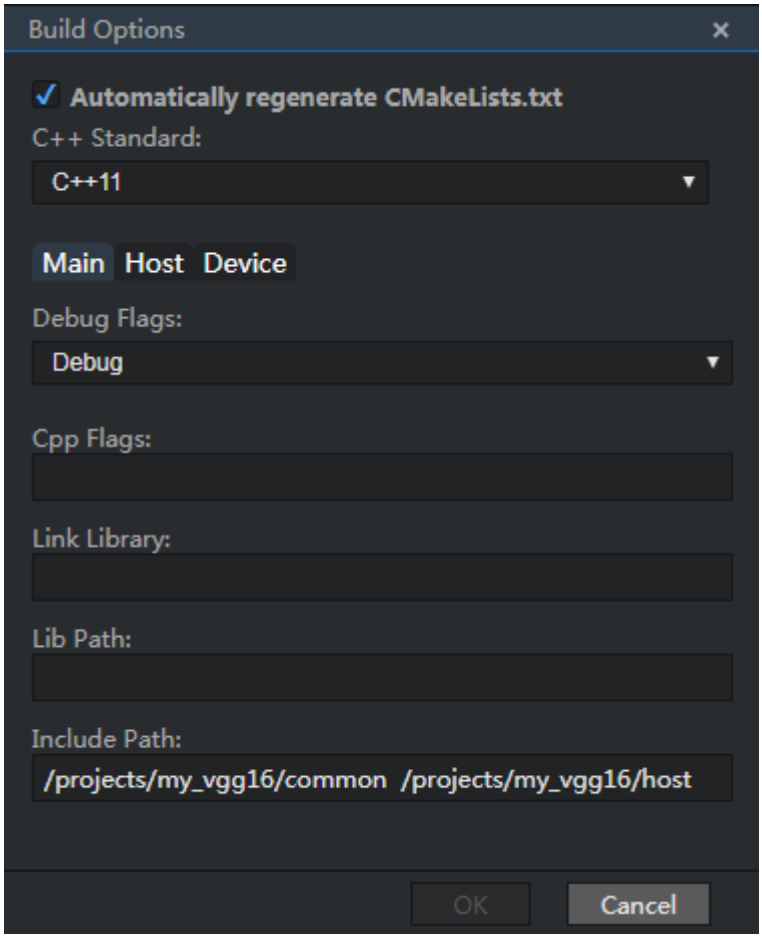
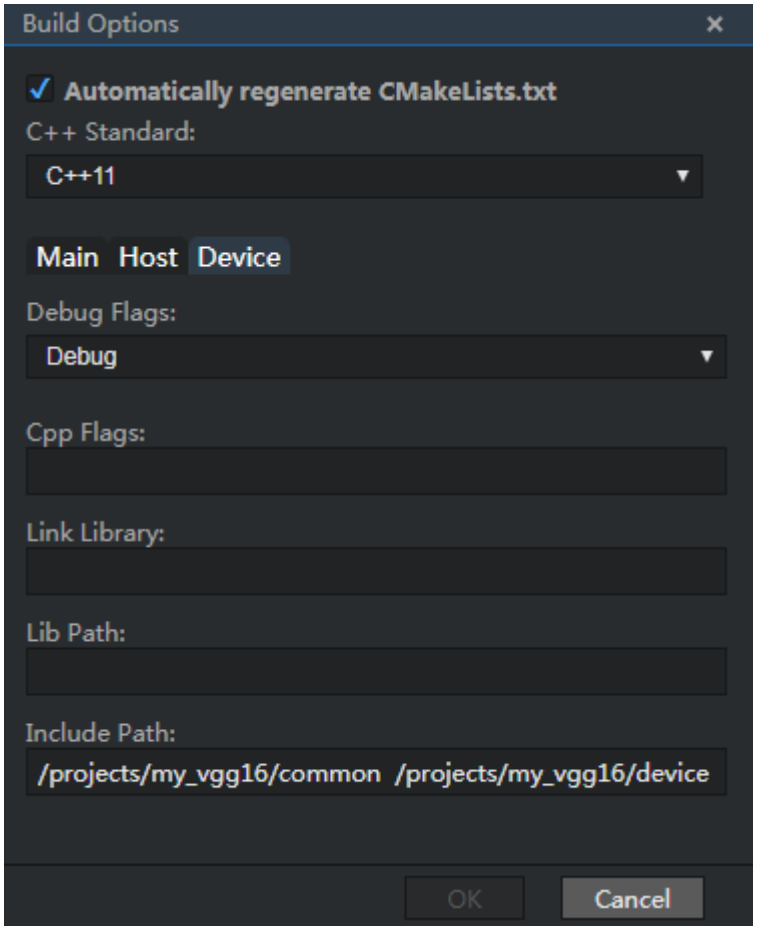


图 3-6 Build Options （Device）配置界面



说明

- Main页签的Include Path中配置“/projects/my_vgg16/common /projects/my_vgg16/host”。
- Device页签中的Include Path中配置“/projects/my_vgg16/common /projects/my_vgg16/device”。

Build Options配置页面中各个页签的配置参数如表3-3所示。

表 3-3 Build Options 配置参数说明

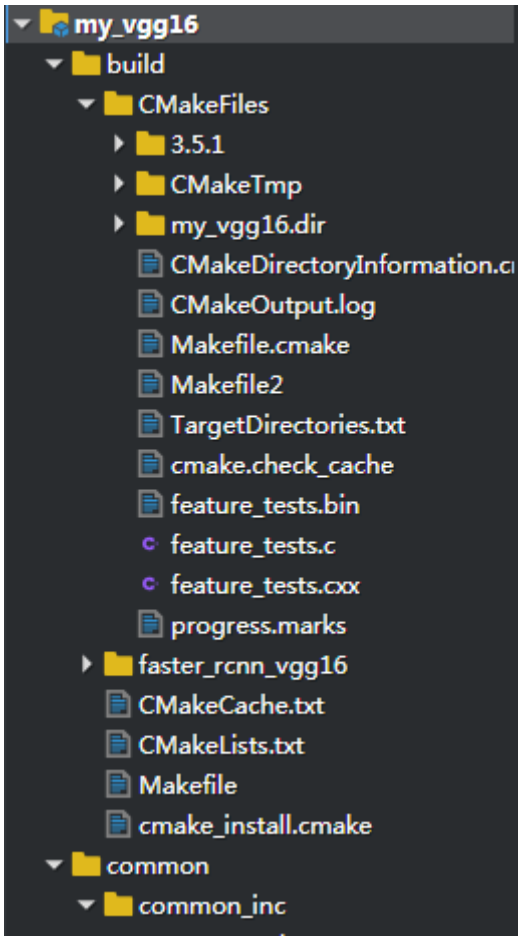
Build Options配置项	说明
Automatically regenerate CMakeLists.txt	默认为勾选状态。 ● 若为勾选状态，每次保存配置时会自动覆盖CMakeLists.txt文件。（如果用户对Makefile文件有修改，也会覆盖。） ● 若为未勾选状态，保存配置时不自动覆盖CMakeLists.txt文件。

Build Options配置项	说明
C++ Standard	C++标准，分为如下两种。 ● C++ 11 ● C++ 98 默认为C++ 11。
Debug Flags	编译模式，分为如下两种。 ● Debug ● Release 默认Debug模式。
Cpp Flags	编译选项，用户配置。
Link Library	引用的链接库，用户可配置。 链接的动态库，对应编译命令中-l参数的值。
Lib Path	链接库路径，目前使用缺省路径，用户可增加配置。 动态库查找路径，对应编译命令中-L参数的值。
Include Path	头文件路径，目前使用缺省路径，用户可增加配置。 头文件查找路径，对应编译命令中-I参数的值。

步骤3 配置完成后，单击“Save”，保存工程结构的配置，单击“Build”进行编译。

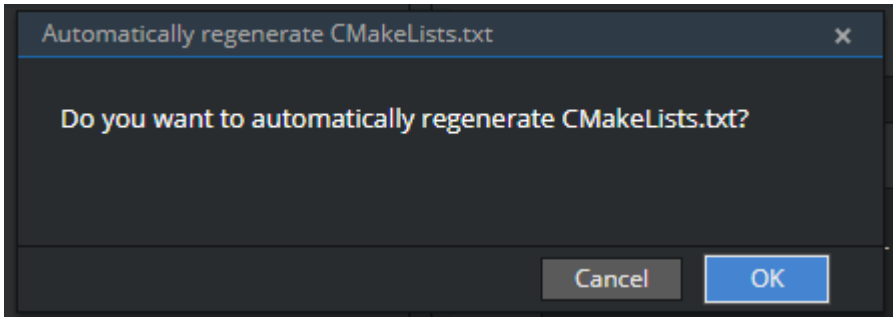
- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为勾选状态，界面下方的“build”窗口内会显示CMakeLists.txt文件生成信息，工程根目录下自动生成保存CMakeLists.txt文件的build文件夹，如图3-7所示。

图 3-7 工程目录示例



- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为未勾选状态。
  - 当工程根目录下不存在build文件夹或工程根目录下的build文件夹不包含CMakeLists.txt文件，则会自动生成包含CMakeLists.txt文件的build文件夹，如图3-7所示。
  - 当工程根目录下存在包含CMakeLists.txt文件的build文件夹，将会弹出Automatically regenerate CMakeLists.txt窗口，如图3-8所示。

图 3-8 弹出窗口



若单击“OK”，则覆盖原有CMakeLists.txt文件。

若单击“Cancel”，则使用当前工程目录下存在的CMakeLists.txt文件进行编译，不再重新生成CMakeLists.txt文件。

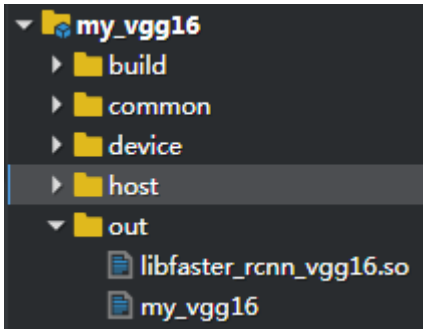
工程编译按成后，build窗口编译结果如图3-9所示。

图 3-9 编译输出窗口

```
CMakeLists.txt generated to /projects/my_vgg16/build/
CMakeLists.txt generated to /projects/my_vgg16/build/faster_rcnn_vgg16
CMakeLists.txt generated
-- Configuring done
-- Generating done
-- Build files have been written to: /projects/my_vgg16/build
[ 12%] Linking CXX executable /projects/my_vgg16/out/my_vgg16
[ 75%] Built target my_vgg16
[ 87%] Linking CXX shared library /projects/my_vgg16/out/libfaster_rcnn_vgg16.so
[100%] Built target faster_rcnn_vgg16
```

编译成功后，会在工程根目录下生成out文件夹，如图3-10所示。

图 3-10 编译后 out 文件



out文件夹中的“my_vgg16”文件为host端的可执行文件，libfaster_rcnn_vgg16.so为Device端的SO文件。

说明

1. 工程中未被Main Engine、Host Engines、Device Engines选中路径的文件夹以及文件在编译时会被忽略。
2. 如果对未配置过的工程进行一键式编译，若工程根目录下存在包含CMakeLists.txt文件的build文件夹，该工程会使用build文件夹进行编译，否则会使用选中的配置对该工程进行编译。
3. 重命名后的工程，需修改之前的Main Engine/Host Engines/Device Engines中Files配置项的路径值。若运行之前的配置，生成器会根据原来的Files内的路径值生成原工程名的无效工程。

----结束

3.7 工程运行

EVB板： 将生成的可执行文件 my_vgg16 及 libfaster_rcnn_vgg16.so 以及生成的模型文件 FasterRCNN-vgg16.davinci.om 放到同一目录，当前目录创建test_data, test_data目录创建data及 config目录， 将faster_rcnn.bin放入data, sample.prototxt放入config目录，执行FasterrcnnVgg16。

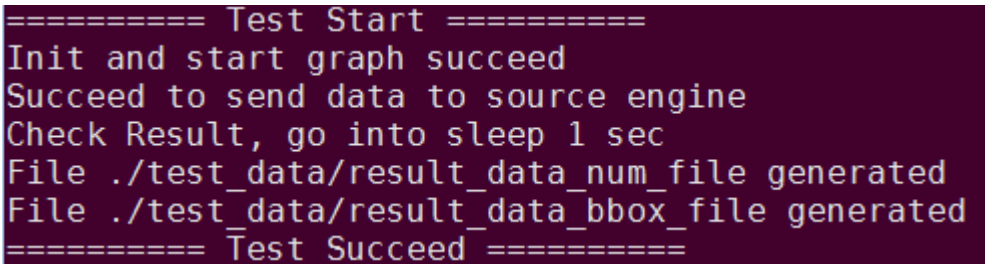
一级目录	二级目录	说明
my_vgg16		可执行文件

一级目录	二级目录	说明
libfaster_rcnn_vgg16.so		device 端so文件
FasterRCNN-vgg16.davinci.om		模型文件
test_data	data/faster_rcnn.bin	数据文件
	config/sample.prototxt	配置文件

3.8 输出验证

进入测试目录，执行 ./my_vgg16，输出如下：

图 3-11 执行结果




```
// 生成data_num和data_bbox信息，32个int32类型数，表示每个目标检测的框的数目
std::shared_ptr<hiai::AINeuralNetworkBuffer> output_data_num =
std::static_pointer_cast<hiai::AINeuralNetworkBuffer>(output_data_vec[0]);
/*
|--1---2---3---4---5-----------32---------|
|  0 | 0 | 1 | 2 | 0 | ..... | 0 |
|-----------------------------------------|
*/
表示label3 有1个框， label4 有两个框， label不包含background

// 生成box_data,维度为(32,608,8)
std::shared_ptr<hiai::AINeuralNetworkBuffer> output_data_bbox =
std::static_pointer_cast<hiai::AINeuralNetworkBuffer>(output_data_vec[1]);
/*
---------|-------------------------------------------------------------|
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox1
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox2
label1   |
         |
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox608
---------|-------------------------------------------------------------|
.
.
.
---------|-------------------------------------------------------------|
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox1
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox2
label32  |
         |
         | xmin | ymin | xmax | ymax | score | reserve | reserve | reserve |-----bbox608
---------|-------------------------------------------------------------|
```



```
box[i, j, 0] 表示第i个分类的第j个框 box的 xmin
box[i, j, 1] 表示第i个分类的第j个框 box的 ymin
box[i, j, 2] 表示第i个分类的第j个框 box的 xmax
box[i, j, 3] 表示第i个分类的第j个框 box的 ymax
box[i, j, 4] 表示第i个分类的第j个框 score
*/
```


4 classfiy net asic

4.1 工程描述

classify_net_asic样例工程是集合了多种分类网络的的示例代码，该示例代码主要实现了输入BGR数据到推理模块， 并输出分类的置信度， 它包含了resnet-18, resnet-50, resnet-101, resnet-152, resnetx-50, resnext-101, vgg16, vgg19等网络， 运行时通过参数区分，如：./classify_net resnet-18。

本样例通过resnet-18网络为例进行执行。

4.2 工程路径

Sample所在路径：$HOME/tools/che/ddk/ddk/sample/classify_net_asic。

由于classify_net_asic模型太大，所以未包含在产品包中，要运行此示例，请先自行下载classify_net_asic网络模型文件以及权重文件，并参考**4.4 数据输入**将网络模型转化为适配Ascend 310的模型。

4.3 工程导入

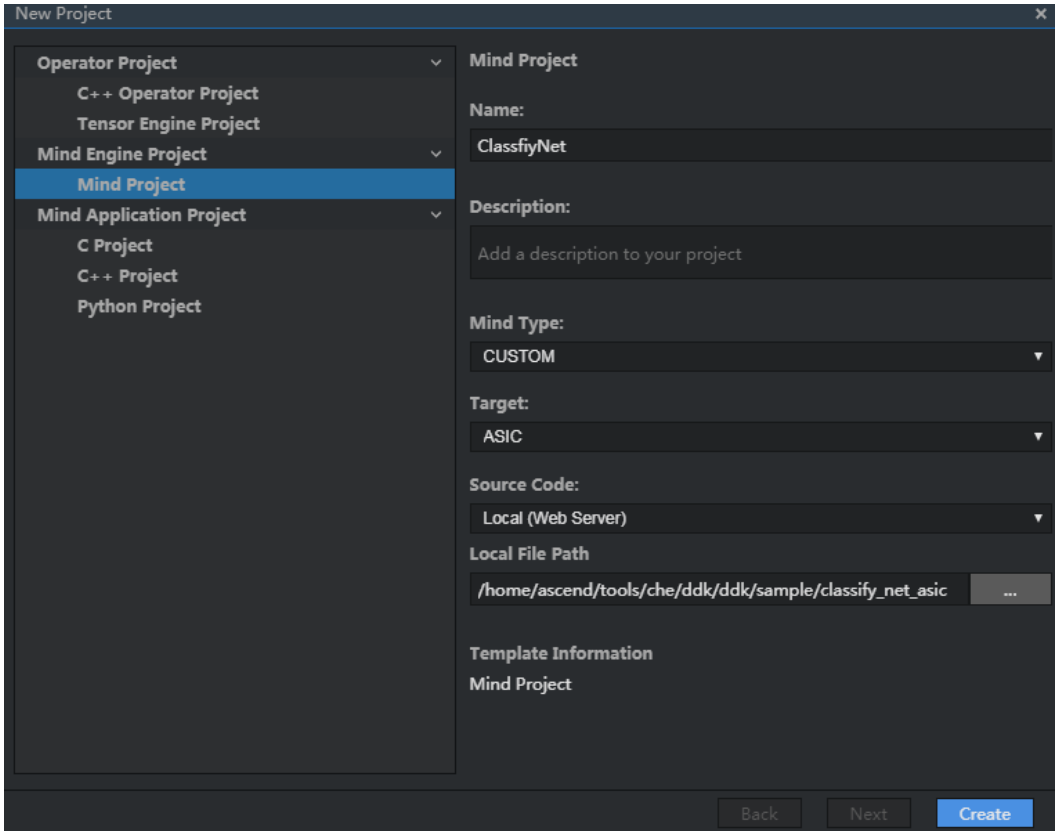
步骤1 在Mind Studio新建工程界面选择“Mind Engine Project > Mind Project”工程类型。

- “Mind Type”选择“CUSTOM”。

- “Target”选择“ASIC”。
- “Source Code From”选择“Local (WebServer)”。
- “Local File Path”中选择“$HOME/tools/che/ddk > ddk > sample > classify_net_asic”。

单击“Create”完成样例工程导入。

图 4-1 导入工程示例



步骤2 样例工程导入后，工程代码结构如下图所示

图 4-2 自定义 Engine 工程代码结构

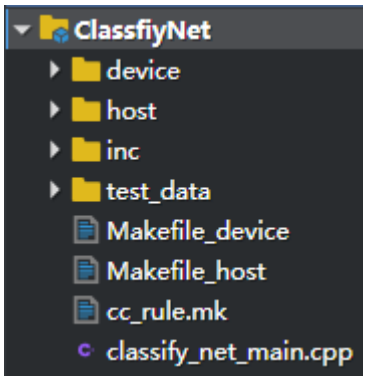


表 4-1 代码结构说明

一级目录	二级目录	说明
device		device侧文件夹
host		host侧文件夹
inc		共用头文件夹
test_data	config	配置文件
	data	测试数据
Makefile	-	编译脚本
cc_rule.mk	-	Makefile编译脚本所加载的文件
classify_net_main.cpp	-	主程序文件

----结束

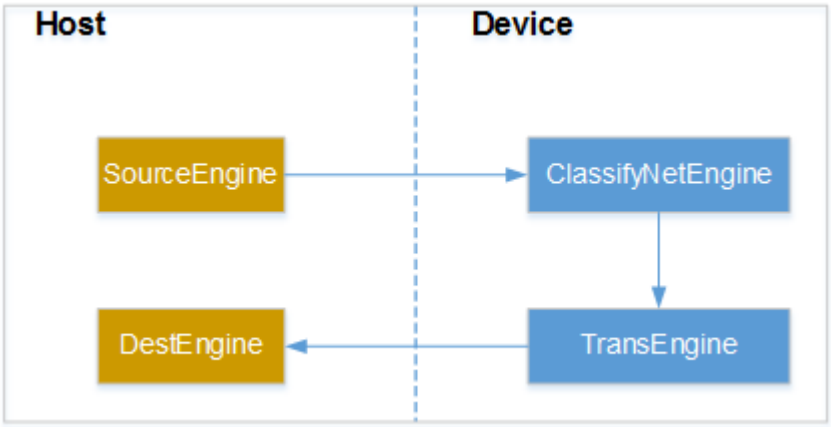
4.4 数据输入

参数	说明
待分类的BGR数据	工程目录下/test_data/resnet-18（每个网络不一样）/data/src_test.bin
Resnet-18模型	模型名称：ResNet-18-model.om(根据网络不一样，模型名称不一，具体的模型名称可查看/config/graph_sample.prototxt文件中的配置)。 执行如下模型转换命令，指定转化后模型名称为ResNet-18-model.om： ./omg --model=*ResNet_18_deploy.prototxt* --weight=*ResNet-18-model.caffemodel* --framework=0 --output=./ResNet-18-model 执行前需要执行如下命令添加环境变量： export LD_LIBRARY_PATH=$HOME/tools/che/ddk/ddk/uihost/lib/ 详细的参数含义可参考《Ascend 310 模型加解密使用指导》。 说明：也可参考《Ascend 310 Mind Studio基本操作》通过Mind Studio前台界面进行转化。
Graph配置文件	工程目录下/test_data/resnet-18（每个网络不一样）/sample.prototxt

4.5 代码实现

该用例主要分为三个Engine（SrcEngine-ClassifyNetEngine-DestEngine）, 详细关系如下图：

图 4-3 engine 实现图



Engine名称	源码文件
SourceEngine	工程目录下/classify_net_host.cpp
ClassifyNetEngine	工程目录下/device/classify_net_ai_engine.cpp
TransEngine	工程目录下/device/classify_net_ai_engine.cpp
DestEngine	工程目录下/classify_net_host.cpp

4.6 工程编译

通过Mind Studio操作界面提供的编译界面进行编译。此种编译方式，用户无需编写Makefile文件，详细操作步骤如下。

步骤1 选中要进行配置工程结构的工程，单击依次选择“Build > Edit Build Configuration”，弹出自定义工程的配置页面。

步骤2 配置自定义工程。

自定义配置工程结构分为**Main**、**Host**、**Device**三部分。

每个.so文件需要根据sample.prototxt文件中的每个Engine的side值在相应页签配置。

Main、host与Device的参数描述如表4-2所示。

表 4-2 配置说明

配置项	说明
Main	“MainEngine”的实现文件，选择执行文件的cpp文件，包含： ● classify_net_main.cpp ● /host/classify_net_host.cpp ● /host/data_recv.cpp ● /host/util.cpp ● /device/sample_data.cpp文件。 保存配置后生成CMakeLists.txt文件（路径：工程根目录下build文件夹）。 编译成功生成可执行文件（路径：out文件夹）。
Host	添加需要运行在host侧的Engine，并填写Engine的名称。 本样例此页签无需配置。
Device	添加运行在Device侧的Engine，并填写Engine的库名称为“ai_engine”。 选择执行文件 cpp文件： /device/classify_net_ai_engine.cpp /device/sample_data.cpp **说明** .so文件需要签名，防止文件被篡改，签名方法: 1. 由openssl工具中的RSA_generate_key接口生成公钥pub.pem+私钥pri.pem，密钥长度建议2048bit以上。 2. 通过私钥，利用SHA256算法对.so文件进行签名，生成.so.signature签名文件，与原.so文件放在同一目录。 3. 公钥需要由SetPublicKeyForSignatureEncryption接口传给HiAI Engine。接口信息请参见《HiAIEngine API参考》。

三个页签的配置依据是根据Engine的so文件需要放置在哪一侧运行。本样例中SourceEngine和DestEngine配置在Main页签；ClassifyNetEngine与TransEngine配置在Device页签，编译配置示例下图所示。

图 4-4 Main 页签配置示例

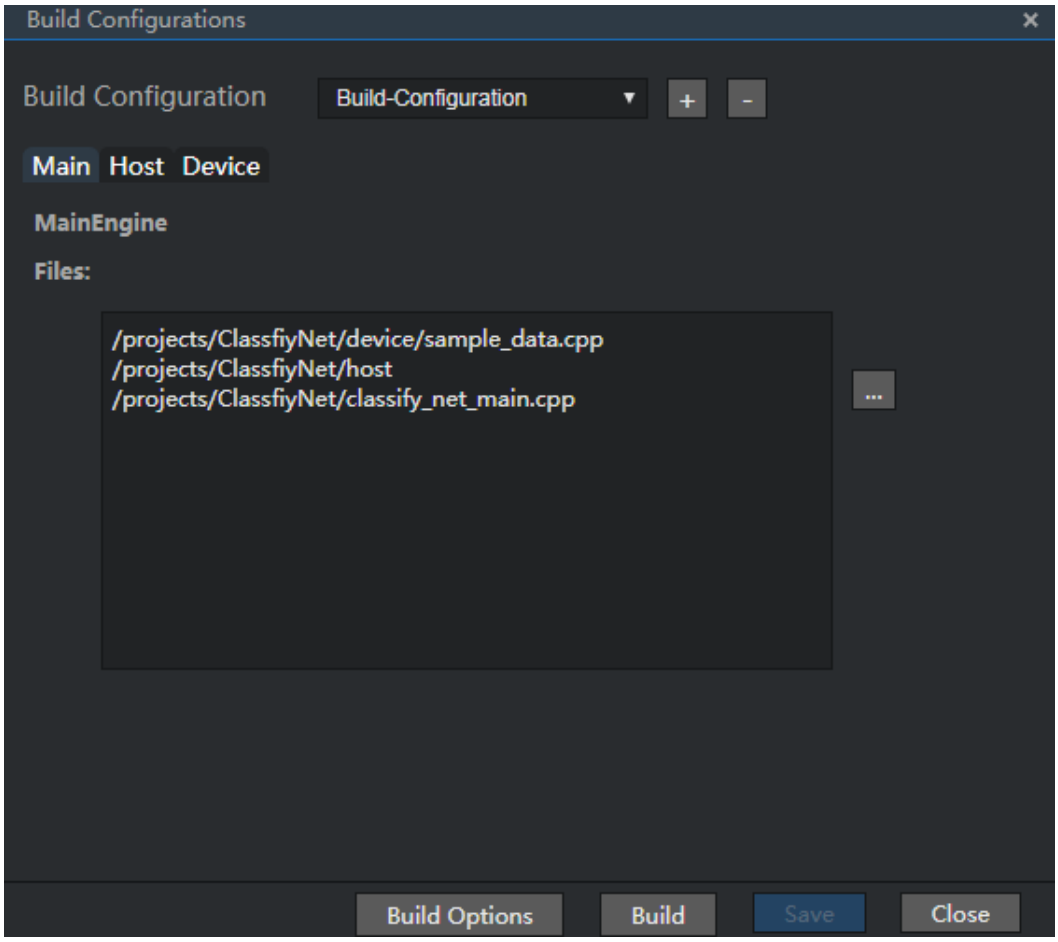
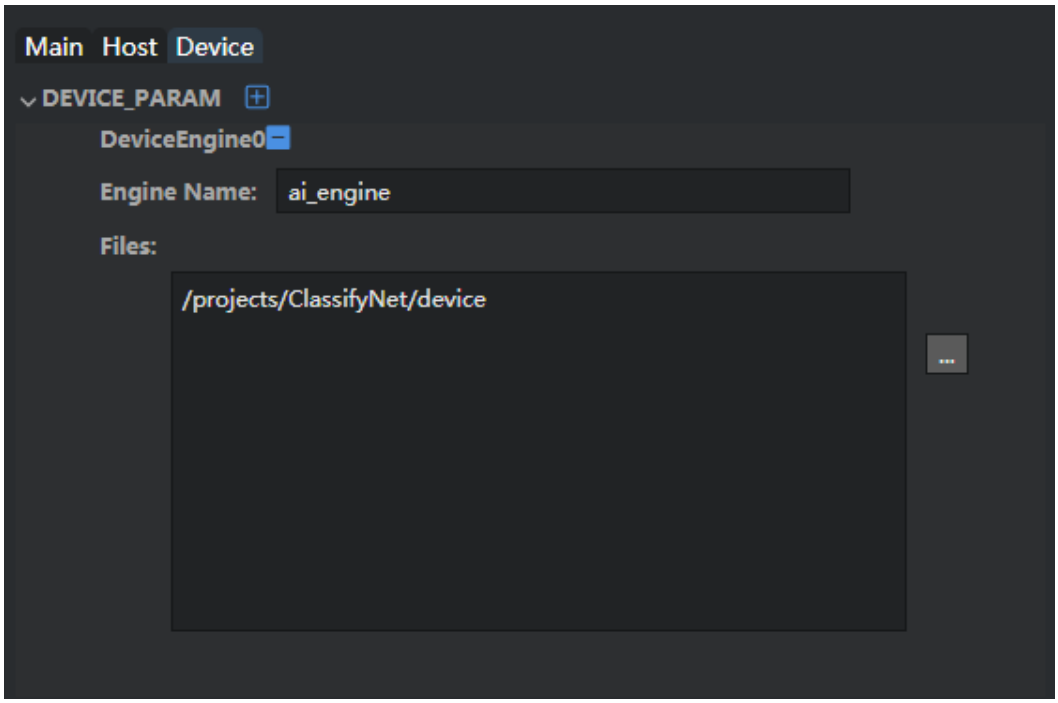
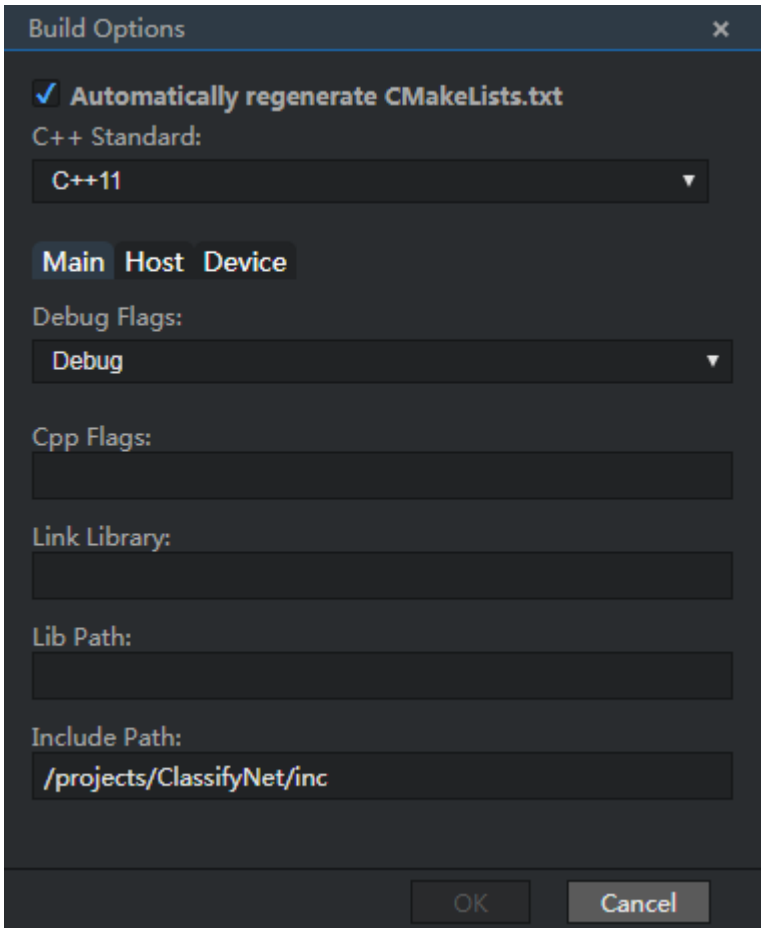


图 4-5 Device 页签配置示例



单击“Build Configurations”中的“Build Options”，弹出如图4-6所示界面。

图 4-6 Build Options 配置界面



说明

- Main页签与Device页签中的Include Path中都配置为“/projects/ClassifyNet/inc”。

Build Options配置页面中各个页签的配置参数如表4-3所示。

表 4-3 Build Options 配置参数说明

Build Options配置项	说明
Automatically regenerate CMakeLists.txt	默认为勾选状态。 ● 若为勾选状态，每次保存配置时会自动覆盖CMakeLists.txt文件。（如果用户对Makefile文件有修改，也会覆盖。） ● 若为未勾选状态，保存配置时不自动覆盖CMakeLists.txt文件。

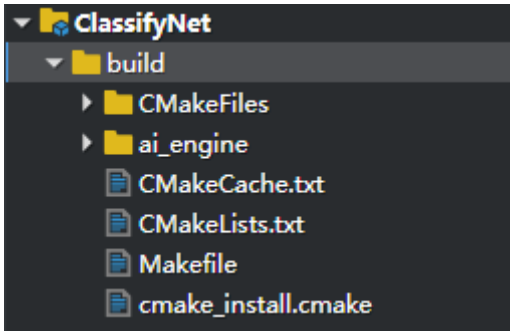
Build Options配置项	说明
C++ Standard	C++标准，分为如下两种。 ● C++ 11 ● C++ 98 默认为C++ 11。
Debug Flags	编译模式，分为如下两种。 ● Debug ● Release 默认Debug模式。
Cpp Flags	编译选项，用户配置。
Link Library	引用的链接库，用户可配置。 链接的动态库，对应编译命令中-l参数的值。
Lib Path	链接库路径，目前使用缺省路径，用户可增加配置。 动态库查找路径，对应编译命令中-L参数的值。
Include Path	头文件路径，目前使用缺省路径，用户可增加配置。 头文件查找路径，对应编译命令中-I参数的值。

步骤3 配置完成后，单击“Save”，保存工程结构的配置，单击“Build”进行编译。

保存工程结构配置分两种情况。

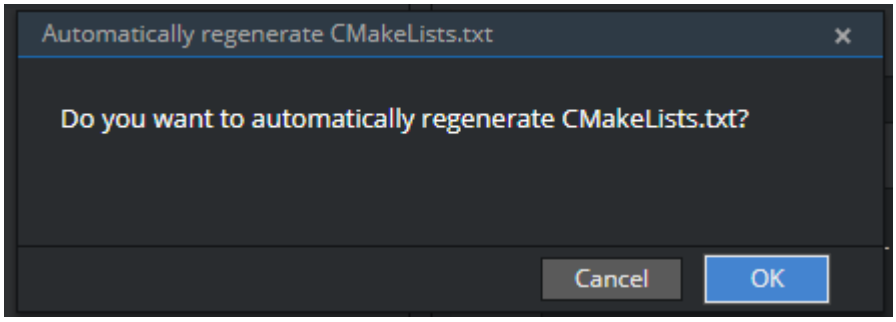
- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为勾选状态，界面下方的“build”窗口内会显示CMakeLists.txt文件生成信息，工程根目录下自动生成保存CMakeLists.txt文件的build文件夹，如图4-7所示。

  图 4-7 工程目录示例

- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为未勾选状态。
  - 当工程根目录下不存在build文件夹或工程根目录下的build文件夹不包含CMakeLists.txt文件，则会自动生成包含CMakeLists.txt文件的build文件夹，如图4-7所示。
  - 当工程根目录下存在包含CMakeLists.txt文件的build文件夹，将会弹出Automatically regenerate CMakeLists.txt窗口，如图4-8所示。

图 4-8 弹出窗口

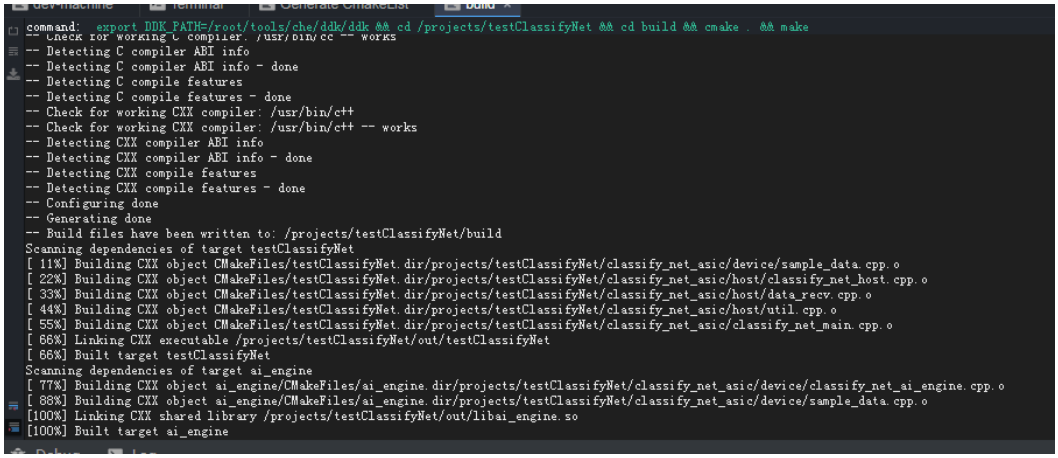


若单击“OK”，则覆盖原有CMakeLists.txt文件。

若单击“Cancel”，则使用当前工程目录下存在的CMakeLists.txt文件进行编译，不再重新生成CMakeLists.txt文件。

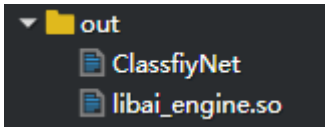
工程编译按成后，build窗口编译结果如图4-9所示。

图 4-9 编译输出窗口



编译成功后，会在工程根目录下生成out文件夹。

图 4-10 编译生成文件



out文件夹中的“ClassifyNet”文件为Main Engine生成的可执行文件。

说明

1. 工程中未被Main Engine、Host Engines、Device Engines选中路径的文件夹以及文件在编译时会被忽略。
2. 如果对未配置过的工程进行一键式编译，若工程根目录下存在包含CMakeLists.txt文件的build文件夹，该工程会使用build文件夹进行编译，否则会使用选中的配置对该工程进行编译。
3. 重命名后的工程，需修改之前的Main Engine/Host Engines/Device Engines中Files配置项的路径值。若运行之前的配置，生成器会根据原来的Files内的路径值生成原工程名的无效工程。

----结束

4.7 工程运行

EVB板： 将生成的可执行文件ClassifyNet及libai_engine.so放到同一目录， 当前目录创建test_data, test_data目录创建执行模型对应的文件夹，如resnet-18, resnet-18下面创建data,mode及 config目录， 将src_test.bin放入data, sample.prototxt放入config目录, ResNet-18-model.om放入mode目录，执行ClassifyNet resnet-18;运行成功后将会生成结果文件result_file

一级目录	二级目录	说明
ClassfiyNet		可执行文件
libai_engine.so		device 端so文件
test_data	resnet-18/data/src_test.bin	数据文件
	resnet-18/mode/ ResNet-18-model.om	模型文件
	resnet-18/config/ graph_sample.prototxt	配置文件

4.8 输出验证

```
将output转换为AINeuralNetworkBuffer
shared_ptr<AINeuralNetworkBuffer> output_tensor =
static_pointer_cast<AINeuralNetworkBuffer>(output_data_vec[0]);
//取出结果的buffer转换为float类型
float * result =(float *)output_tensor->GetBuffer();
int label_index = 0;
float max_value = 0.0;
//遍历查找最大的分类下标和对应的置信度值
for(int i=0; i< output_tensor->GetSize()/sizeof(float) ; i++)
{
    if(*(result + i) > max_value)
    {
        max_value = *(result + i);
        label_index = i;
    }
}
//结果展示
printf("label index:%d, Confidence :%f\n", label_index, max_value);
```


5 Detection ssd

5.1 工程描述

detection_ssd样例工程是基于ssd检测网络编写的示例代码，该示例代码主要实现了输入BGR数据给到推理模块， 并输出分类的置信度以及检测目标的框。

5.2 工程路径

Sample所在路径：$HOME/tools/che/ddk/ddk/sample/detection_ssd_asic。

由于ssd caffe模型太大，所以未包含在产品包中，要运行此示例，请先自行下载ssd网络模型的模型文件及权重文件，参考**5.4 数据输入**将其转换为适配Ascend 310的模型。

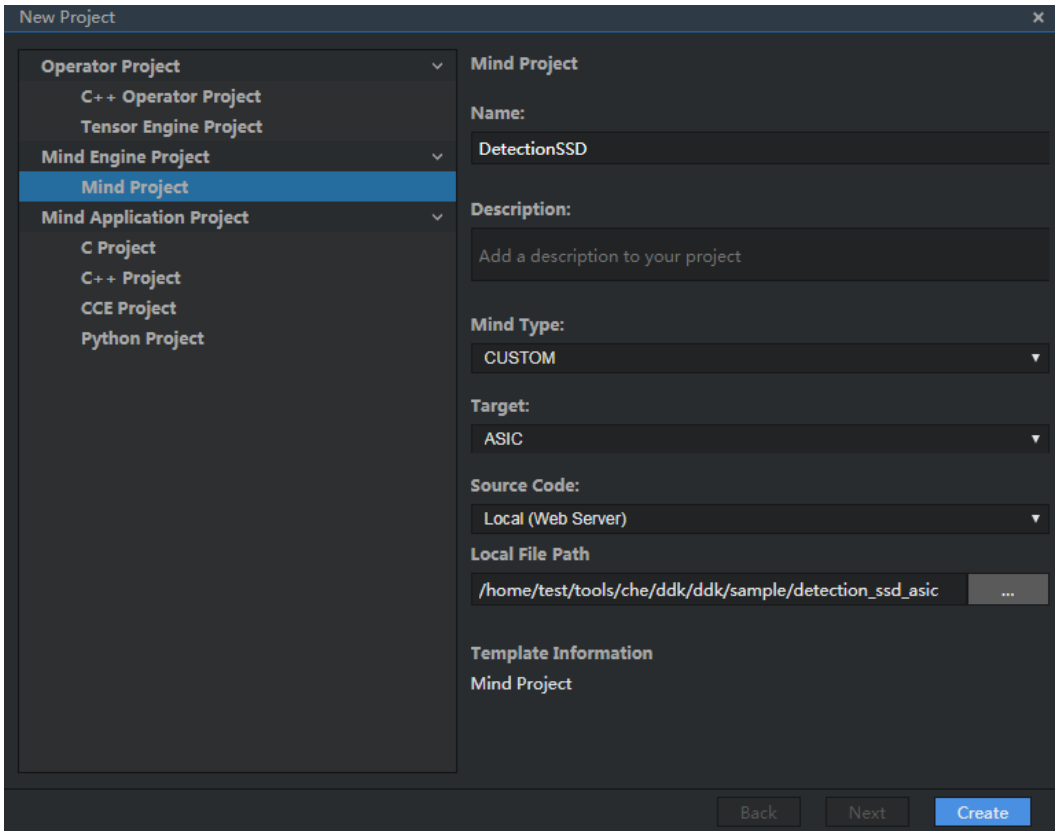
5.3 工程导入

步骤1 在Mind Studio新建工程界面选择“Mind Engine Project > Mind Project”工程类型。

- “Mind Type”选择“CUSTOM”。
- “Target”选择“ASIC”。
- “Source Code From”选择“Local (WebServer)”。
- “Local File Path”中选择“$HOME/tools/che/ddk > ddk > sample > detection_ssd_asic”。

单击“Create”完成样例工程导入。

图 5-1 导入工程示例



步骤2 样例工程导入后，工程代码结构如图5-2所示。

图 5-2 DetectionSSD 样例工程代码结构

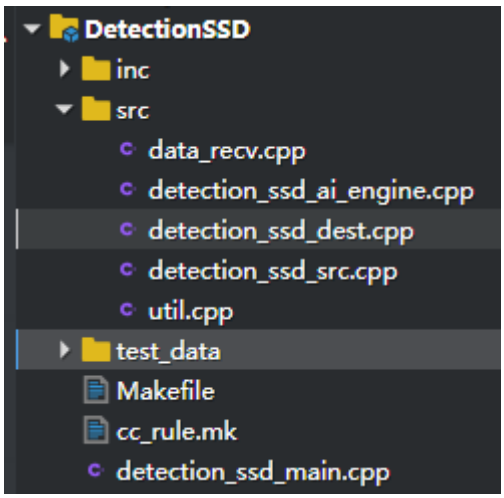


表 5-1 代码结构说明

一级目录	二级目录	说明
inc		头文件夹
src	detection_ssd_src.cpp	destEngine的实现
	detection_ssd_ai_engine.cpp	Device AIEngine的实现
	detection_ssd_dest.cpp	Src Engine的实现
test_data		测试数据及配置文件
Makefile	-	编译脚本
cc_rule.mk	-	Makefile编译脚本所加载的文件
detection_ssd_main.cpp	-	主程序

----结束

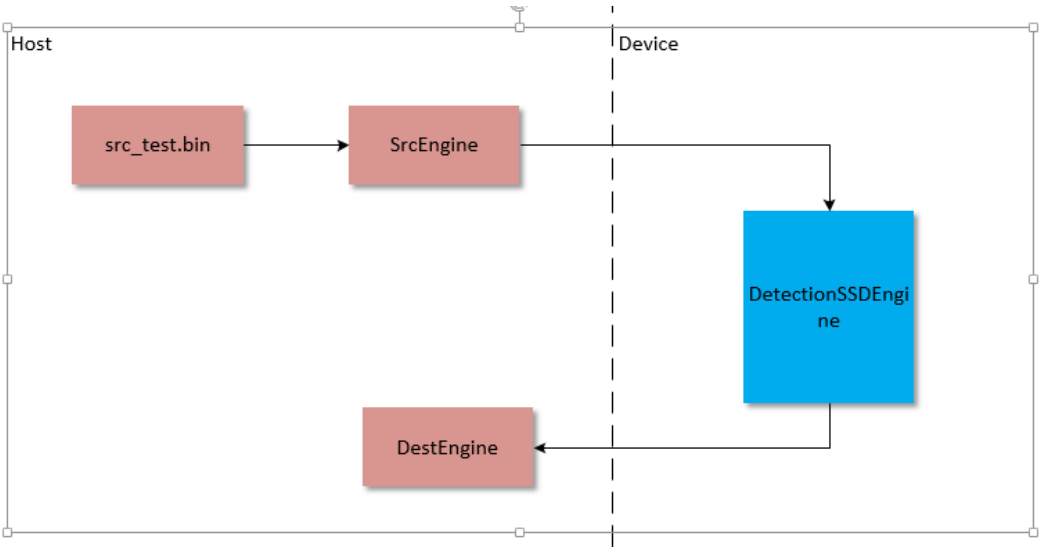
5.4 数据输入

参数	说明
待分类的BGR数据	工程目录下/test_data/ssd/data/src_test.bin。
ssd模型	将转化好的模型文件放入工程目录下：/test_data/ssd/mode/SSD-model.om 执行如下模型转换命令，指定转化后模型名称为SSD-model.om： ./omg --model=*./ssd_deploy.prototxt* --weight=*./ssd.caffemodel* --output=*./mode/SSD-model* --framework=0 --op_name_map=*./sdd-op.mapping* 执行omg命令前需要执行如下命令添加环境变量： export LD_LIBRARY_PATH=$HOME/tools/che/ddk/ddk/uihost/lib/ 详细的参数含义可参考《Ascend 310 模型加解密使用指导》。 说明：也可参考《Ascend 310 Mind Studio基本操作》通过Mind Studio前台界面进行转化。
Graph配置文件	工程目录下/test_data/ssd/config/graph_sample.prototxt

5.5 代码实现

该用例主要分为三个Engine（SrcEngine-DetectionSSDEngine-DestEngine）, 详细关系如下图：

图 5-3 engine 实现图



Engine名称	源码文件
SrcEngine	工程目录下/src/detection_ssd_src.cpp。
DetectionSSDEngine	工程目录下/src/detection_ssd_ai_engine.cpp。
DestEngine	工程目录下/src/detection_ssd_dest.cpp。

5.6 工程编译

通过Mind Studio操作界面提供的编译界面进行编译。此种编译方式，用户无需编写Makefile文件，详细操作步骤如下。

步骤1 选中要进行配置工程结构的工程，单击依次选择“Build > Edit Build Configuration”，弹出自定义工程的配置页面。

步骤2 配置自定义工程。

自定义配置工程结构分为**Main**、**host**、**Device**三部分。

每个.so文件需要根据graph_sample_mini文件中的每个Engine的side值在相应页签配置。

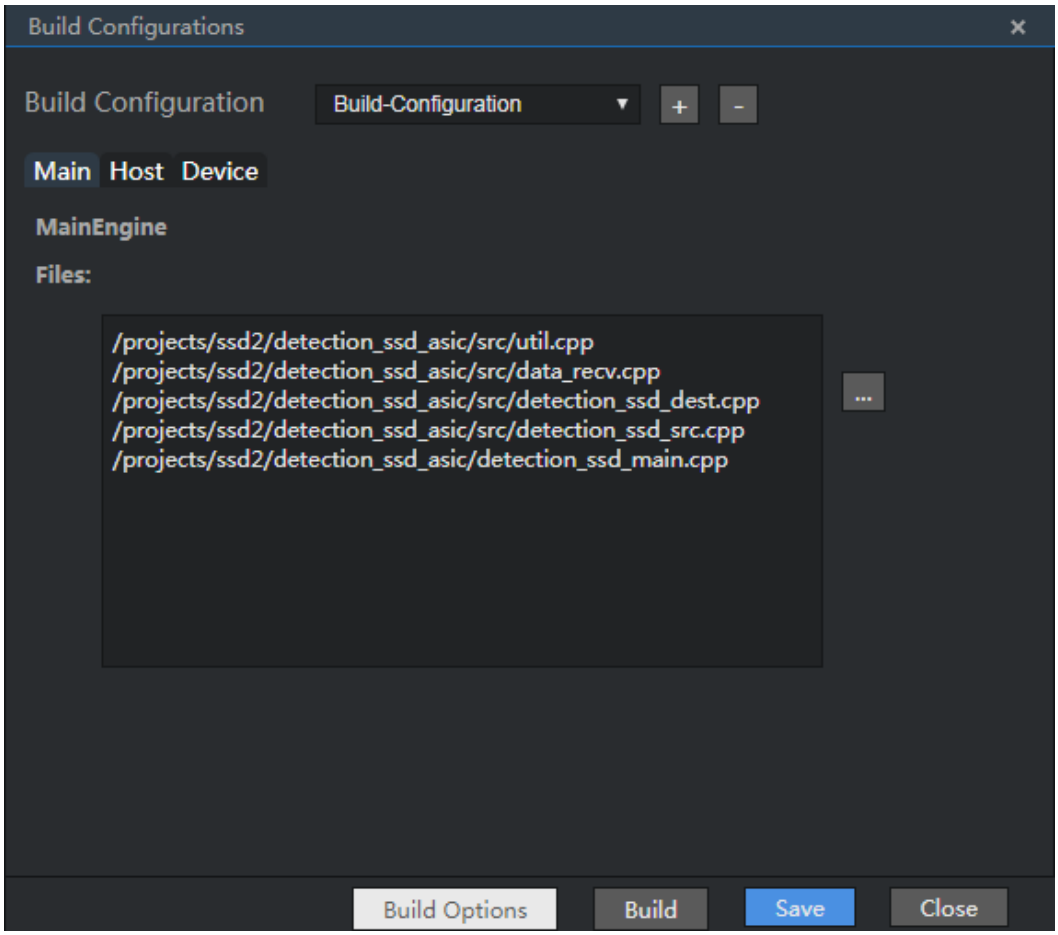
Main、host与Device的参数描述如表5-2所示。

表 5-2 配置说明

配置项	说明
Main	“MainEngine”的实现文件，选择执行文件的cpp文件： ● detection_ssd_main.cpp ● src/data_recv.cpp ● src/util.cpp ● src/detection_ssd_src.cpp ● src/detection_ssd_dest.cpp
Host	添加需要运行在host侧的Engine。 本样例在host页不配置。
Device	添加运行在Device侧的Engine，并填写Engine的名称为“DetectionSSDEngine”。 选择执行文件 cpp文件： /src/detection_ssd_ai_engine.cpp 编译成功生成与Engine Name同名的.so文件（路径：out文件夹）。 **说明** .so文件需要签名，防止文件被篡改，签名方法: 1. 由openssl工具中的RSA_generate_key接口生成公钥pub.pem+私钥pri.pem，密钥长度建议2048bit以上。 2. 通过私钥，利用SHA256算法对.so文件进行签名，生成.so.signature签名文件，与原.so文件放在同一目录。 3. 公钥需要由SetPublicKeyForSignatureEncryption接口传给HiAI Engine。接口信息请参见《HiAIEngine API参考》。

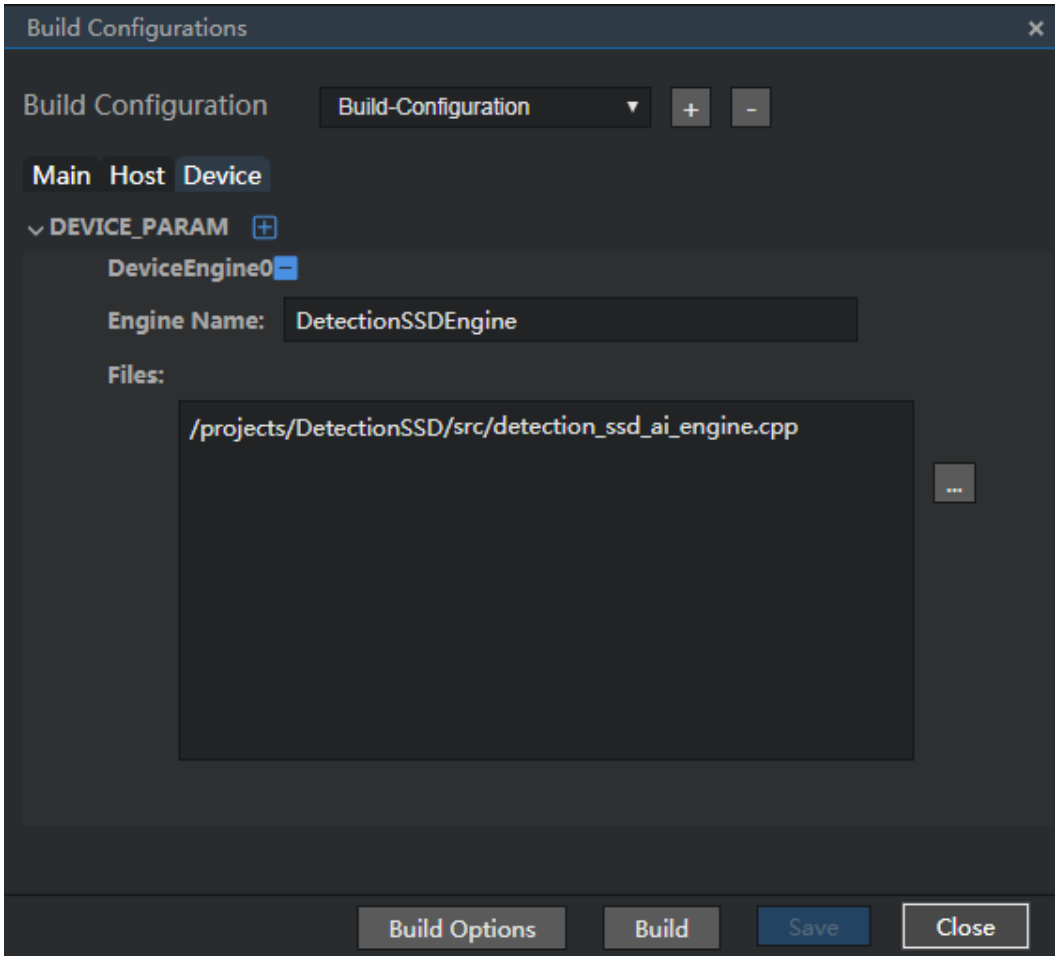
三个页签的配置依据是根据Engine的so文件需要放置在哪一侧运行。由于样例是ASIC工程，样例中SrcEngine和DestEngine在host侧运行,但是不生成so，在main侧编译；DetectionSSDEngine在Device侧运行，编译配置示例下图所示。

图 5-4 Main 页签配置示例



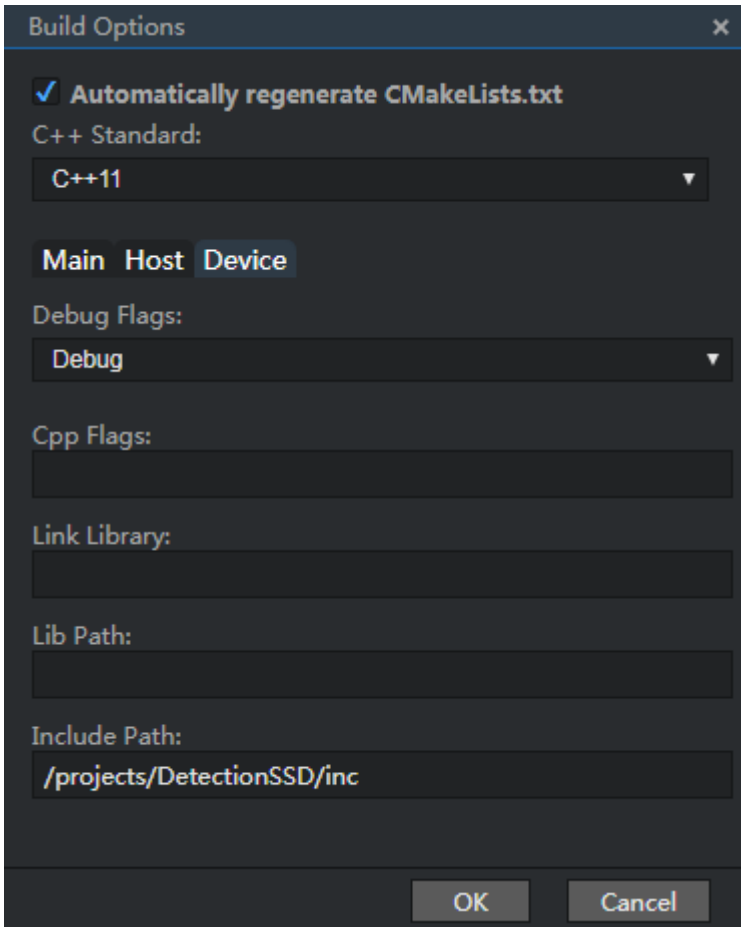
Device页签配置示例如图5-5所示。

图 5-5 Device 页签配置



单击“Build Configuration”中的“Build Options”，弹出如图5-6所示界面。

图 5-6 Build Options 配置界面



说明

Main页签、Host页签与Device页签中的Include Path中都配置为“/projects/DetectionSSD/inc”。

Build Options配置页面中各个页签的配置参数如表5-3所示。

表 5-3 Build Options 配置参数说明

Build Options配置项	说明
Automatically regenerate CMakeLists.txt	默认为勾选状态。 ● 若为勾选状态，每次保存配置时会自动覆盖CMakeLists.txt文件。（如果用户对Makefile文件有修改，也会覆盖。） ● 若为未勾选状态，保存配置时不自动覆盖CMakeLists.txt文件。
C++ Standard	C++标准，分为如下两种。 ● C++ 11 ● C++ 98 默认为C++ 11。

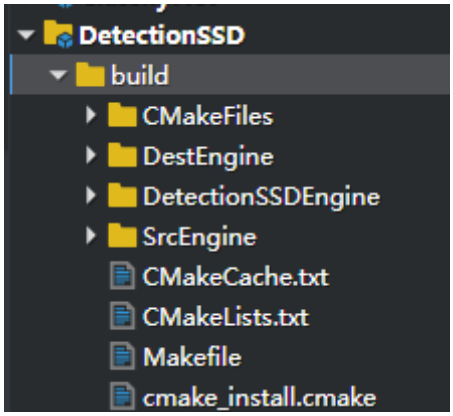
Build Options配置项	说明
Debug Flags	编译模式，分为如下两种。 ● Debug ● Release 默认Debug模式。
Cpp Flags	编译选项，用户配置。
Link Library	引用的链接库，用户可配置。 链接的动态库，对应编译命令中-l参数的值。
Lib Path	链接库路径，目前使用缺省路径，用户可增加配置。 动态库查找路径，对应编译命令中-L参数的值。
Include Path	头文件路径，目前使用缺省路径，用户可增加配置。 头文件查找路径，对应编译命令中-I参数的值。

步骤3 配置完成后，单击“Save”，保存工程结构的配置，单击“Build”进行编译。

保存工程结构配置分两种情况。

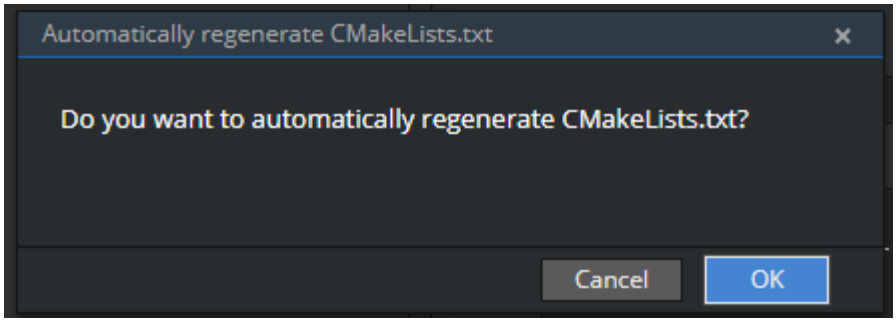
- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为勾选状态，界面下方的“build”窗口内会显示CMakeLists.txt文件生成信息，工程根目录下自动生成保存CMakeLists.txt文件的build文件夹，如图5-7所示。

  图 5-7 工程目录示例

- 若“Build Options”配置窗口中的“Automatically regenerate CMakeLists.txt”配置为未勾选状态。
  - 当工程根目录下不存在build文件夹或工程根目录下的build文件夹不包含CMakeLists.txt文件，则会自动生成包含CMakeLists.txt文件的build文件夹，如图5-7所示。
  - 当工程根目录下存在包含CMakeLists.txt文件的build文件夹，将会弹出Automatically regenerate CMakeLists.txt窗口，如图5-8所示。

图 5-8 弹出窗口

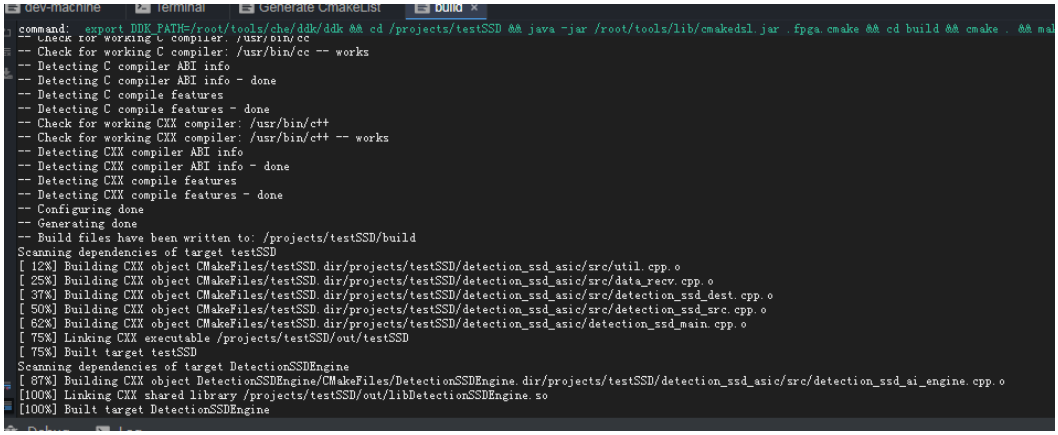


若单击“OK”，则覆盖原有CMakeLists.txt文件。

若单击“Cancel”，则使用当前工程目录下存在的CMakeLists.txt文件进行编译，不再重新生成CMakeLists.txt文件。

工程编译按成后，build窗口编译结果如图5-9所示。

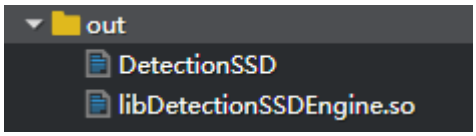
图 5-9 编译输出窗口



编译成功后，会在工程根目录下生成out文件夹，如图5-10所示。

注：目前该样例多文件使用common.h，出现如上warning为正常，不影响程序

图 5-10 编译后 out 文件



out文件夹中的“DetectionSSD”文件为Main Engine生成的可执行文件。

说明

1. 工程中未被Main Engine、Host Engines、Device Engines选中路径的文件夹以及文件在编译时会被忽略。
2. 如果对未配置过的工程进行一键式编译，若工程根目录下存在包含CMakeLists.txt文件的build文件夹，该工程会使用build文件夹进行编译，否则会使用选中的配置对该工程进行编译。
3. 重命名后的工程，需修改之前的Main Engine/Host Engines/Device Engines中Files配置项的路径值。若运行之前的配置，生成器会根据原来的Files内的路径值生成原工程名的无效工程。

----结束

5.7 工程运行

EVB板： 将生成的可执行文件DetectionSSD,libDetectionSSDEngine.so放到同一目录，当前目录创建test_data, test_data目录创建ssd目录，ssd下面创建data,mode及 config目录， 将src_test.bin放入data, graph_sample.prototxt放入config目录, SSD-model.om放入mode目录，执行DetectionSSD，运行成功后，将生成大小为4字节的result_data_num_file文件，表示检测到的框数，以及大小为5600字节的result_data_bbox_file文件，表示框的具体信息。

一级目录	二级目录	说明
DetectionSSD		可执行文件
libDetectionSSDEngine		Device侧推理Engine so文件
test_data	ssd/data/src_test.bin	数据文件
	ssd/mode/SSD-model.om	模型文件
	ssd/config/graph_sample.prototxt	配置文件

5.8 输出验证

```
// 生成data_num和data_bbox信息
IMAGE_HEIGHT = 300;
IMAGE_WIDTH = 300;

//box_num 结果大小为4个字节，为一个float32的数，表示网络中检测到N个框
std::shared_ptr<hiai::AINeuralNetworkBuffer> output_data_num =
std::static_pointer_cast<hiai::AINeuralNetworkBuffer>(output_data_vec[1]);

// box_data 检测框的结果信息，shape(200,7)，数据类型为float32
std::shared_ptr<hiai::AINeuralNetworkBuffer> output_data_bbox =
std::static_pointer_cast<hiai::AINeuralNetworkBuffer>(output_data_vec[0]);
/*
|-----0-------1-------2------3------3------4------5--------6----|
| image_id| Label | score | xmin | ymin | xmax | ymax | reserve |----------bbox1
| image_id| Label | score | xmin | ymin | xmax | ymax | reserve |----------bbox2
|-------------------------------------------------------------|
*/
取对应的前N个框
```


6 dvpp

6.1 工程描述

此样例工程提供了使用DVPP处理各种视频和图片的功能样例，用户可以根据样例使用VDEC、VENC、VPC、JPEGD、JPEGE、PNGD。

6.2 sample 代码路径

Sample所在路径：$HOME/tools/che/ddk/ddk/sample/dvpp。

6.3 数据输入

数据输入依赖用户提供的视频和图像文件。

视频和格式支持限制请参考《 Ascend 310 DVPP API 参考》中各功能的使用限制。

6.4 代码实现

详细见代码sample_dvpp.cpp文件

6.5 工程编译

本样例支持通过后台命令进行编译，方法如下：

进入DDK安装路径$HOME/tools/che/ddk/ddk/sample/dvpp

执行make -j命令，编译出来的文件在out目录下。

6.6 工程运行

步骤1 将编译生成的可执行文件“sample_dvpp_hlt_ddk”复制到device侧任意路径。

说明

依赖SO为： libc_sec.so libDvpp_api.so libDvpp_jpeg_decoder.so libDvpp_jpeg_encoder.so libDvpp_png_decoder.so libDvpp_vpc.so libOMX_common.so libOMX_hisi_vdec_core.so libOMX_hisi_video_decoder.so libOMX_hisi_video_encoder.so libOMX_Core.so libOMX_Core_VENC.so libslog.so

这些so默认已存在deivce侧的/usr/lib64目录，用户无需单独拷贝。

步骤2 将用户自行准备的输入数据文件放入到可执行文件所在路径下，运行./sample_dvpp_hlt_ddk paramList，其中paramList请见《Ascend 310 DVPP API 参考》附录中的DVPP执行器工具使用说明

----结束

6.7 输出验证

各个模块的输出文件，详细见《 Ascend 310 DVPP API 参考》附录中的DVPP执行器工具使用说明章节。

