Hi3516ev300 移植 Tensorflow Lite 流程

１．安装交叉编译链，本文以 arm-himix100-linux 为例。

２．下载 Tensorflow 源码

```
git clone https://github.com/tensorflow/tensorflow
```

３．安装 Tensorflow 相关依赖包

cd 到 Tensorflow 工程的根目录，然后执行下面的脚本

```
./tensorflow/contrib/lite/download_dependencies.sh
```

４．交叉编译 Tensorflow Lite

先修改以下内容

在../tensorflow/lite/tools/make/build_rpi_lib.sh 对应修改

＃＃

```
set -e

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

cd "$SCRIPT_DIR/../../.."

#change CC_PREFIX if u need

CC_PREFIX=arm-himix100-linux- make -j 3 -f tensorflow/lite/tools/make/Makefile
TARGET=RPI TARGET_ARCH=armv7
```

＃＃

修改完后在 tensorflow 根目录执行

```
./tensorflow/lite/tools/make/build_rpi_lib.sh
```

编译结束，会在 tensorflow/lite/tools/make/gen/lib/PRI_armv7 目录下产生 libtensorflow-lite.a 静态库，至此 libtensorflow-lite 编译成功。

５．编译 Demo 代码

本文中以 label_image 为例，首先修改第４步中的脚本文件 build_rpi_lib.sh，修改方式可以参考 Makefile 里对 MINIMAL Demo 的配置，本文示例如下:

```
# Make uses /bin/sh by default, which is incompatible with the
bashisms seen
# below.
SHELL := /bin/bash

# Find where we're running from, so we can store generated files
here.
ifeq ($(origin MAKEFILE_DIR), undefined)
   MAKEFILE_DIR := $(shell dirname $(realpath $(lastword $
(MAKEFILE_LIST))))
endif

# Try to figure out the host system
HOST_OS :=
ifeq ($(OS),Windows_NT)
   HOST_OS = windows
else
   UNAME_S := $(shell uname -s)
   ifeq ($(UNAME_S),Linux)
      HOST_OS := linux
   endif
   ifeq ($(UNAME_S),Darwin)
      HOST_OS := osx
   endif
endif

HOST_ARCH := $(shell if uname -m | grep -q i[345678]86; then echo
x86_32; else uname -m; fi)
OBJDIR := $(MAKEFILE_DIR)/gen/obj/
BINDIR := $(MAKEFILE_DIR)/gen/bin/
LIBDIR := $(MAKEFILE_DIR)/gen/lib/
GENDIR := $(MAKEFILE_DIR)/gen/obj/

CXX := $(CC_PREFIX)g++
CXXFLAGS := -mcpu=cortex-a7 -mfloat-abi=softfp -mfpu=neon-vfpv4 -
fno-aggressive-loop-optimizations --std=c++11 -O3 -DNDEBUG
```

```
CC := $(CC_PREFIX)gcc
CFLAGS  :=  -mcpu=cortex-a7  -mfloat-abi=softfp  -mfpu=neon-vfpv4  -
fno-aggressive-loop-optimizations -O3 -DNDEBUG
LDOPTS :=
LDOPTS += -L/usr/local/lib
LIBFLAGS                                                           =
-L/home/gzh/Hi3516EV200R001C01SPC010/software/Hi3516EV200R001C01SP
C010/01.software/board/Hi3516EV200_SDK_V1.0.1.0/osdrv/tools/board/
mtd-utils/zlib-1.2.11/zlib_install/lib
ARFLAGS := -r

INCLUDES := \
-I. \
-I$(MAKEFILE_DIR)/../../../ \
-I$(MAKEFILE_DIR)/downloads/ \
-I$(MAKEFILE_DIR)/downloads/eigen \
-I$(MAKEFILE_DIR)/downloads/gemmlowp \
-I$(MAKEFILE_DIR)/downloads/neon_2_sse \
-I$(MAKEFILE_DIR)/downloads/farmhash/src \
-I$(MAKEFILE_DIR)/downloads/flatbuffers/include \
-I$(GENDIR)

# This is at the end so any globally-installed frameworks like
protobuf don't
# override local versions in the source tree.
INCLUDES += -I/usr/local/include

# These are the default libraries needed, but they can be added to
or
#  overridden  by  the  platform-specific  settings  in  target
makefiles.
LIBS := \
-lstdc++ \
-lpthread \
-lm \
-lz \
-ldl \
-lrt

include
/home/gzh/armnn/tensorflow/tensorflow/lite/tools/make/targets/rpi_
makefile.inc
# This library is the main target for this makefile. It will
contain a minimal
# runtime that can be linked in to other programs.
BIN_PATH := $(BINDIR)label_image
TF_LIB_PATH                                                    :=
/home/gzh/armnn/tensorflow/tensorflow/lite/tools/make/RPI_armv7/li
b/libtensorflow-lite.a

# A small example program that shows how to link against the
library.
#MINIMAL_PATH                                                  :=
/home/danale/tensorlite/tensorflow/tensorflow/lite/examples/minima
l
LABEL_IMAGE_PATH :=$(BINDIR)label_image
#MINIMAL_SRCS                                                  :=
/home/danale/tensorlite/tensorflow/tensorflow/lite/examples/minima
l/minimal.cc
#MINIMAL_OBJS := $(addprefix $(OBJDIR), \
#$(patsubst %.cc,%.o,$(patsubst %.c,%.o,$(MINIMAL_SRCS))))

LABEL_IMAGE_SRCS                                               :=
/home/gzh/armnn/tensorflow/tensorflow/lite/examples/label_image/la
bel_image.cc \
/home/gzh/armnn/tensorflow/tensorflow/lite/examples/label_image/
bitmap_helpers.cc
LABEL_IMAGE_OBJS := $(addprefix $(OBJDIR), \
$(patsubst %.cc,%.o,$(patsubst %.c,%.o,$(LABEL_IMAGE_SRCS))))

# What sources we want to compile, must be kept in sync with the
main Bazel
```

```
# build files.

CORE_CC_ALL_SRCS := \
$(wildcard tensorflow/lite/*.cc) \
$(wildcard tensorflow/lite/kernels/*.cc) \
$(wildcard tensorflow/lite/kernels/internal/*.cc) \
$(wildcard tensorflow/lite/kernels/internal/optimized/*.cc) \
$(wildcard tensorflow/lite/kernels/internal/reference/*.cc) \
$(wildcard tensorflow/lite/*.c) \
$(wildcard tensorflow/lite/kernels/*.c) \
$(wildcard tensorflow/lite/kernels/internal/*.c) \
$(wildcard tensorflow/lite/kernels/internal/optimized/*.c) \
$(wildcard tensorflow/lite/kernels/internal/reference/*.c) \
$(wildcard tensorflow/lite/downloads/farmhash/src/farmhash.cc) \
$(wildcard tensorflow/lite/downloads/fft2d/fftsg.c)

# Remove any duplicates.
CORE_CC_ALL_SRCS := $(sort $(CORE_CC_ALL_SRCS))
CORE_CC_EXCLUDE_SRCS := \
$(wildcard tensorflow/lite/*test.cc) \
$(wildcard tensorflow/lite/*/*test.cc) \
$(wildcard tensorflow/lite/*/*/*test.cc) \
$(wildcard tensorflow/lite/*/*/*/*test.cc) \
$(wildcard tensorflow/lite/kernels/*test_util.cc) \
#$(MINIMAL_SRCS) \
$(LABEL_IMAGE_SRCS)

# Filter out all the excluded files.
TF_LITE_CC_SRCS   :=   $(filter-out   $(CORE_CC_EXCLUDE_SRCS),   $
(CORE_CC_ALL_SRCS))
TF_LITE_CC_OBJS := $(addprefix $(OBJDIR), \
$(patsubst %.cc,%.o,$(patsubst %.c,%.o,$(TF_LITE_CC_SRCS))))
LIB_OBJS := $(TF_LITE_CC_OBJS)

# For normal manually-created TensorFlow Lite C++ source files.
$(OBJDIR)%.o: %.cc
```

```
   @mkdir -p $(dir $@)
   $(CXX) $(CXXFLAGS) $(INCLUDES) -c $< -o $@
# For normal manually-created TensorFlow Lite C source files.
$(OBJDIR)%.o: %.c
   @mkdir -p $(dir $@)
   $(CC) $(CCFLAGS) $(INCLUDES) -c $< -o $@

# The target that's compiled if there's no command-line arguments.
all: $(TF_LIB_PATH) $(LABEL_IMAGE_PATH)

# Gathers together all the objects we've compiled into a single
'.a' archive.

$(TF_LIB_PATH): $(LIB_OBJS)
   @mkdir -p $(dir $@)
   $(AR) $(ARFLAGS) $(TF_LIB_PATH) $(LIB_OBJS)

$(LABEL_IMAGE_PATH): $(LABEL_IMAGE_OBJS) $(TF_LIB_PATH)
   @mkdir -p $(dir $@)
   $(CXX) $(CXXFLAGS) $(INCLUDES) \
   -o $(LABEL_IMAGE_PATH) $(LABEL_IMAGE_OBJS) \
   $(LIBFLAGS) $(TF_LIB_PATH) $(LDFLAGS) $(LIBS)

# Gets rid of all generated files.
clean:
   rm -rf $(MAKEFILE_DIR)/gen

# Gets rid of target files only, leaving the host alone. Also
leaves the lib
# directory untouched deliberately, so we can persist multiple
architectures
# across builds for iOS and Android.
cleantarget:
   rm -rf $(OBJDIR)
   rm -rf $(BINDIR)
```

```
$(DEPDIR)/%.d: ;
.PRECIOUS: $(DEPDIR)/%.d

-include $(patsubst %,$(DEPDIR)/%.d,$(basename $(TF_CC_SRCS)))
                                                              #
```

注：整个编译过程Ｍａｋｅｆｉｌｅ非常关键，需要认真分析后仔细修改，避免意想不到的问题。

修改完成后再次执行 ./tensorflow/lite/tools/make/build_rpi_lib.sh ，此时在 /tensorflow/lite/tools/make/gen/bin/RPI_armv8 目录下会产生编译好的 label_image 二进制文件，至此 Demo 编译完成。

６．找到以下文件并将其拷贝到板子上，运行 Tensorflow Lite Demo

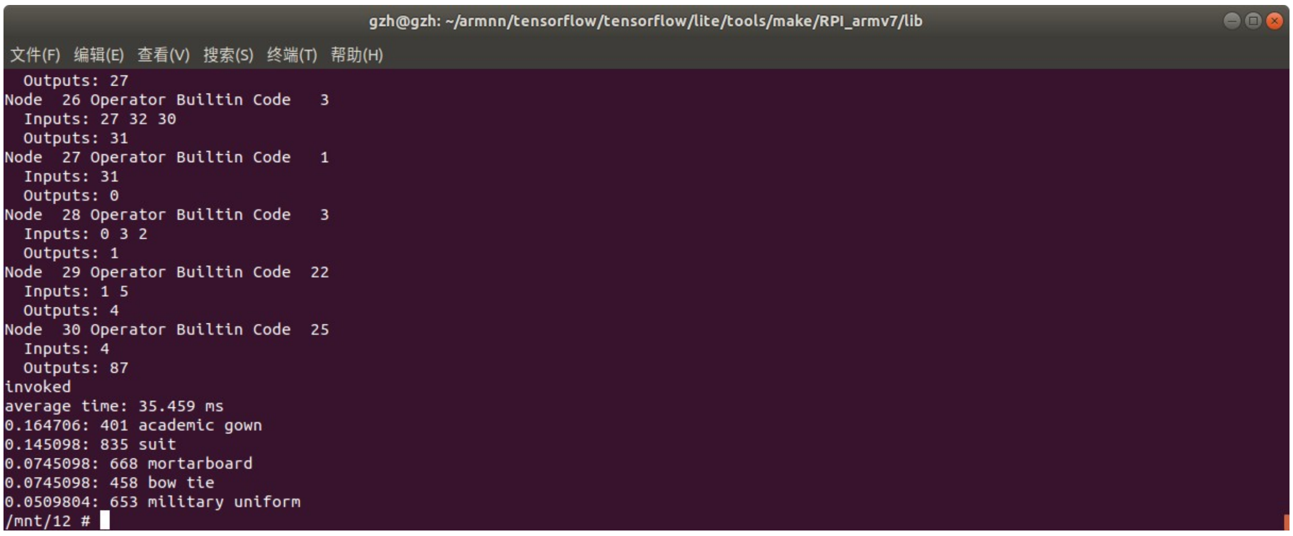
```
grace_hopper.bmp  //测试图像
label_image       //Demo 可执行文件
labels.txt        //标签文件
mobilenet_v1_1.0_224_quant.tflite/mobilenet_v1_1.0_224.tflite  //测试模型
```

准备好之后在板子上执行

```
./label_image -v 1 -m ./mobilenet_v1_0.25_128_quant.tflite -i ./grace_

hopper.bmp -l ./labels_mobilenet_quant_v1_224.txt -t 1
```

运行结果如下:


gzh@gzh: ~/armnn/tensorflow/tensorflow/lite/tools/make/RPI_armv7/lib
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
  Outputs: 27
Node  26 Operator Builtin Code   3
  Inputs: 27 32 30
  Outputs: 31
Node  27 Operator Builtin Code   1
  Inputs: 31
  Outputs: 0
Node  28 Operator Builtin Code   3
  Inputs: 0 3 2
  Outputs: 1
Node  29 Operator Builtin Code  22
  Inputs: 1 5
  Outputs: 4
Node  30 Operator Builtin Code  25
  Inputs: 4
  Outputs: 87
invoked
average time: 35.459 ms
0.164706: 401 academic gown
0.145098: 835 suit
0.0745098: 668 mortarboard
0.0745098: 458 bow tie
0.0509804: 653 military uniform
/mnt/12 #
```