Hi3516ev300 使用ＡＲＭＮＮ流程

１．安装 SCons

这里安装的是 Scons2.4.1，如果已经安装过其他 scons 了，必须卸载。卸载方法如下：

```
rm /usr/local/bin/scons*
rm -r /usr/local/lib/scons*
```

下载完解压，进入解压后的文件夹 scons-2.4.1，在终端里输入 sudo python setup.py install

２．安装 CMake

sh cmake-3.11.1-Linux-x86_64.sh

３．编译 Boost

下载 1.64.0 版本，下载.gz 压缩包，解压之后进入文件夹 boost_1_64_0，运行 sh bootstrap.sh

之后要对该文件夹的 project-config.jam 进行修改。修改如下：

原内容：

```
if ! gcc in [ feature.values <toolset> ]
{
    using gcc ;
}
```

修改后：

```
if ! gcc in [ feature.values <toolset> ]
{
   using gcc : arm : xxxx/arm-himix100-linux-gcc ;# xxxx 是你的海思交叉编译链的位置
}
```

修改后运行：

```
./b2 link=static cxxflags=-fPIC --with-filesystem --with-test --with-log --with-
program_options
```

注意：

如果运行指令以后，显示的是

```
-32-bit         :yes
-arm            :yes
```

证明这个板子是 32 位的。

如果运行指令以后，显示的是

```
-32-bit         :no
-64-bit         :yes
-arm            :yes
```

证明这个板子是 64 位的。

４．普通编译 protobuf

cd BASEDIR/protobuf

./autogen.sh

./configure

make

make check

make install

5．编译 Caffe

请自行参照网上教程

6．安装 tensorflow

cd $BASEDIR/tensorflow

执行　/armnn/scripts/generate_tensorflow_protobuf.sh ../tensorflow-protobuf ../protobuf-host

7．Build the Google FlatBuffers library

下载 Google FlatBuffers library，解压执行

cd BASEDIR/flatbuffers

海思编译链　cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release

make

8．交叉编译 protobuf

cd BASEDIR/protobuf

make clean

./autogen.sh

./configure –prefix=$BASEDIR/protobuf-host -交叉编译链

make

make check

make install

9．使用 scons 编译 ComputeLibrary

修改 SConstruct：

原内容：

```
if env['arch'] == 'armv7a':
    env.Append(CXXFLAGS = ['-march=armv7-a', '-mthumb', '-mfpu=neon'])

    if env['os'] == 'linux':
        prefix = "arm-linux-gnueabihf-"
        env.Append(CXXFLAGS = ['-mfloat-abi=hard'])
```

```
    elif env['os'] == 'bare_metal':
        prefix = "arm-eabi-"
        env.Append(CXXFLAGS = ['-mfloat-abi=hard'])
    elif env['os'] == 'android':
        prefix = "arm-linux-androideabi-"
        env.Append(CXXFLAGS = ['-mfloat-abi=softfp'])
```

修改后:

```
if env['arch'] == 'armv7a':
    env.Append(CXXFLAGS = ['-march=armv7-a', '-mthumb', '-mfpu=neon'])

    if env['os'] == 'linux':
        prefix = "arm-himix100-linux-"
        env.Append(CXXFLAGS = ['-mfloat-abi=softfp'])
    elif env['os'] == 'bare_metal':
        prefix = "arm-eabi-"
        env.Append(CXXFLAGS = ['-mfloat-abi=softfp'])
    elif env['os'] == 'android':
        prefix = "arm-linux-androideabi-"
        env.Append(CXXFLAGS = ['-mfloat-abi=softfp'])
```

原内容:

```
BoolVariable("cppthreads", "Enable C++11 threads backend", True),
```

修改后:

```
BoolVariable("cppthreads", "Enable C++11 threads backend", False),
```

原内容:

```
BoolVariable("neon", "Enable Neon support", False),
```

修改后:

```
BoolVariable("neon", "Enable Neon support", True),
```

运行

```
scons extra_cxx_flags="-fPIC" benchmark_tests=1 validation_tests=1 neon=1 opencl
=1 embed_kernels=1 -j4
```

ComputeLibrary 是一个很大的库，编译接近１小时，过程中如果出现问题，只要仔细分析就能解决。

１０．编译 ARMNN-SDK

```
git clone https://github.com/ARM-software/armnn.git
```

进入下载的 armnn 的文件夹中。修改 CMakeLists.txt。在首行加

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -mcpu=cortex-a7 -mfloat-abi=softfp -mfpu=neo
n-vfpv4 -fno-aggressive-loop-optimizations -mfpu=neon -std=c++11 -Wall -Werror -
Wold-style-cast -Wno-missing-braces -Wconversion -Wsign-conversion -pthread" )
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mcpu=cortex-a7 -mfloat-abi=softfp -mfpu
=neon-vfpv4 -fno-aggressive-loop-optimizations -mfpu=neon -std=c++11 -Wall -Werr
or -Wold-style-cast -Wno-missing-braces -Wconversion -Wsign-conversion -pthread"
)
```

```
SET(BOOST_ROOT /home/danale/armnn/boost_1_64_0)
SET(CMAKE_C_COMPILER xxx/arm-himix100-linux-gcc)
SET(CMAKE_CXX_COMPILER xxx/arm-himix100-linux-g++)
```

执行

```
cmake -DARMCOMPUTE_ROOT=/home/danale/armnn/ComputeLibrary -DARMCOMPUTE_BUILD_DIR
=/home/danale/armnn/ComputeLibrary/build -DBOOST_ROOT=/home/danale/armnn/boost_1
_64_0 -DTF_GENERATED_SOURCES=/home/danale/armnn/protobuf/src -DCAFFE_GENERATED_S
OURCES=/home/danale/armnn/caffe/build/include  -DTF_GENERATED_SOURCES=$BASEDIR/t
ensorflow-protobuf -DPROTOBUF_ROOT=$BASEDIR/protobuf-host -DBUILD_TF_LITE_PARSER
=1 -DTF_LITE_GENERATED_PATH=$BASEDIR/tensorflow/tensorflow/lite/schema -DFLATBUF
FERS_ROOT=$BASEDIR/flatbuffers -DFLATBUFFERS_LIBRARY=$BASEDIR/flatbuffers/libfla
tbuffers.a -DBUILD_CAFFE_PARSER=1 -DARMCOMPUTENEON=1 -DBUILD_TF_PARSER=1 -DPROTO
BUF_LIBRARY_RELEASE=/home/danale/armnn/protobuf/src/.libs/libprotobuf.so -DPROTO
BUF_LIBRARY_DEBUG=/home/danale/armnn/protobuf/src/.libs/libprotobuf.so -DPROTOBU
F_INCLUDE_DIRS=/usr/local
```

执行

```
make
```

编译成功以后，会得到以下文件：

libarmnn.so

libarmnnTfLiteParser.so

libarmnnTfParser.so

libarmnnUtils.a

１１．在 hi3516ev300 跑简单 Mnist 样例

将以上生成的库文件拷贝到板子

下载 git clone https://github.com/ARM-software/ML-examples.git

编译 Mnist 项目，此ｓａｍｐｌｅ是基于ｃａｆｆｅ的。编译成功生成可执行文件 mnist_caffe。

在板子上执行如下：

```
root@imx8mqevk:~/mnist# time ./mnist_caffe
Predicted: 7
   Actual: 7

real    0m0.084s
user    0m0.040s
sys     0m0.016s
root@imx8mqevk:~/mnist#
root@imx8mqevk:~/mnist# time ./mnist_tf
Predicted: 7
   Actual: 7

real    0m0.992s
user    0m0.048s
sys     0m0.032s
root@imx8mqevk:~/mnist#
```