Tutorial on

How to use Simulink Algorithms in Android Applications

I. Overview

This brief tutorial describes how to use algorithms developed in Simulink in Android applications. The whole idea is to use Simulink tool to develop sophisticated mathematical functions that can be used for data processing. The developed Simulink models can be turned into c codes using the code generation function inherent to Simulink. The generate c codes are then incorporated in the Android project and built into a library, which can be called by the Java codes in the Android project. The main advantages of this method, compared to developing everything in Java, are that: (i) Simulink is a high-level design tool, which is much more convenient to design algorithms than in Java code. (ii) Matlab/Simulink has many out-of-the-box mathematics functions (e.g., the Fourier Transform), where in Java, using third-party libraries would be cumbersome and writing the math functions oneself is effort-demanding and error-prone.

The whole process can be divided into the following steps:

1. Algorithm development in Simulink
2. Code generation in Simulink
3. Configuration in Android Studio
4. Move generated codes into Android project
5. Write C wrapper code
6. Call functions from Java

Necessary tools for include:

- Matlab/Simulink
- Android Studio
- Android NDK

II. Code Generation in Simulink

Algorithm Development: We skip the first step of algorithm development in Simulink, which is quite intuitive and come to the step of code generation. One thing to note is that in the Simulink model, always use *Inport* and *Outport* as data sinks and sources – the Matlab workspace is not available for Android devices.

Simulink Project Settings: The Simulink model needs to be properly configured for the code generation. For this purpose, press the gear icon in the toolbar and wait for the '*Configuration Parameters*' window to appear. Then set the following fields:

- In *Solver*, set *Solver options->Type* as *Fixed-step, Solver options->Solver* as *discrete* and *Solver options->Fixed-step* size as *1*.
- In *Code Generation*, set *Target selection->System target file* as *ert.tlc (Embedded Coder)* , uncheck the box *Generate Makefiles* and check the box *Generate code only*.

Code Generation: Then press the *build* button to generate codes. You can see the build process in the Matlab command window. The c codes will be saved in the current Matlab path.

III. Configuration in Android Studio

NDK: For the generated code to be properly built und incorporated in the Android projects, Android NDK is necessary. The NDK can be downloaded from https://developer.android.com/ndk/index.html. After downloading and extracting the NDK to a certain path (*NDK_PATH*), go to the *local.propertie*s file in the Android Project and added the NDK path to the existing SDK path. (*ndk.dir=NDK_PATH*). The NDK is already installed in the diskless clients. For the path of the NDK, please refer to the example apps.

Creating Library Module: Go to *File->New->New Module…*, select the new module to be *Android Library* and name it (e.g., *ndkmodule*). Then, under the directory *ndkmodule/src/main/* create a new directory and by right click on *main*, then *New->Folder->JNI Folder*.

IV. Move Generated Codes into Android Project

*Copy all *.c and *.h files:* Copy all the source and header files generated from Simulink model to the *jni* folder created in the last step. The file *ert_main.c* is not needed.

V. Write C Wrapper Code

In order to the generated c code to be incorporated into the Android project, some wrapper codes are necessary. The functions that need to be wrapped include (e.g., if the Simulink model name is *add*) *add_initialize()*, *add_terminate()*, *add_step()* and some functions to set the input values and retrieve the output values. These functions can be found in the generated c codes. The NDK needs a special function signature to recognize the generated c functions. For example *real_T Java_com_rcsexample_btfft2_MainActivity_getModelOutput(JNIEnv* env, jobject javaThis, jint index)* is a function that takes two integer variables, i.e., index and channel and returns a *real_T* variable (double). The long name is necessary for the *jni* to find the functions. It starts with *Java,* then the package name (in this case *com.rcsexample.btfft2*), then the name of the calling java class (e.g., *MainActivity*), and after the last underscore the actual function name.

VI. Call Functions from Java

The next step is to declare and use the functions from the c codes in the Java codes. For each Java module that uses the c functions, first go to *buid.gradle* file of the module add *compile project(' :ndkmodule')* under *dependencies.* Then in the main Java code, load the library by calling *static {System.loadLibrary("ndkmodule")}* ("ndkmodule" here should correspond to the module name of the c codes). Then declare the functions to be used individually as native (e.g., *private native void stepModel()*). After this steps, the functions can be called in the Java code using the short name (e.g., *stepModel()*).

Appendix: Copy Scripts for the C Codes

Two scripts are provided to ease the copying process of the c codes. If manual copying is preferred. Then the scripts here are not needed. The two scripts are *copysimulinkfiles.m* and *copyfiles.sh*. The first is used in Matlab to find all the related files. The second copies these files to the *jni* folder.

Android Studio Setup

In Android Studio, go to *File->Settings…*, then under *Tools->External Tools*, add a new tool. Set the *Name* of the tool (e.g., Copy Simulink Files), then set *Tool settings->Program* as the script *copyfiles.sh* with the full path, set *Tool settings->Parameters* as the *jni* folder with full path.

Copy Files

Set the working directory in Matlab as the same one holding the Simulink models. Then run the script *copysimulinkfiles.m* in Matlab. This will find all the necessary files and save their paths and names in a file. Then in Android Studio, go to *Tools->External Tools*, and clicked the external tool added, which runs the *copyfiles.sh* script and copies all the files to the *jni* folder.