VS680 Software Overview

Andrew Duggan
22 August 2023
Overview

- VS680 runs the open source version of Android 12 (AOSP).
- We provide an SDK called VSSDK which contains the board specific components which make Android run.
  1. Kernel
  2. Bootloader
  3. Native processes for supporting media
  4. Trustzone OS and apps
  5. Android HAL implementations
Architectural Overview

VSSDK contains the code for the Linux Kernel, Native Daemons and Libraries, and HAL implementations.
Development on VS680

Android provides well defined APIs for interfacing with the Camera, NPU, and GPU.

Applications can be developed using Android Studio the Android NDK can be used to develop the native components.

Example:
https://github.com/android/camera-samples
AOSP vs Android

AOSP is the purely open source version of Android and it lacks Google Play Services. Developing Android apps for AOSP cannot depend on these services.
Interacting with the NPU

It can be accessed through the Android [NNAPI interface](https://www.tensorflow.org/lite/guide/nnapi). The NNAPI uses “online” model conversion which optimizes and converts the model at runtime into a native format understood by the NPU hardware.
Offline Models

Offline conversion is done beforehand and allows for greater optimization. The output is a native binary format compatible with the NPU hardware.
SynAP

Synaptics provides SynAP to support offline modes.

SynAP consists of a framework for running offline models and a toolkit for quantizing, optimizing, and profiling offline models.

Both are documented in the SDK in syna-release/synap/release/doc/SynAP.pdf
Working with VS680
Accessing the SDK with Git

Accessing our external SDK is restricted. We will create one or two accounts which have access to it. Accounts should use an @ucf.edu email address.

Access requires:
- SSH RSA public / private key pair for the account.
- The external IP address of the server being used to download the code.

A local mirror can be setup to distribute the SDK throughout the class.

Instructions are on page 8 of the User Guide.
Build Environment

A build server with 16GB RAM and 300 – 600GB storage. We recommend Ubuntu 16.04 as the OS.

Build server can be a VM or docker container.

List of package dependencies is in the user guide.
**Downloading the SDK**

The SDK and Android both use repo to manage all of the Git repositories.

In this example the manifest branch name is set with the `-b` parameter and is:

```
$ mkdir s_aosp_dolphin_sl_noip
$ cd s_aosp_dolphin_sl_noip
$ repo init -u ssh://debugithub.synaptics.com:29418/manifest -b m/cd/reference_auto_sync/sdk/rel_branch/vssdk/v1.10.1/202301101805/s_aosp_dolphin_sl_noip/v202305160610/202305171142
```

This will varies based on the release being downloaded.

```
$ repo sync
Fetching project DEBU/android/vendor/imagination
remote: Counting objects: 7109, done
remote: Finding sources: 100% (7109/7109)
```
SDK Directory Structure

The Android directory contains Android HAL components and will be copied into the Android Tree.
The syna-release directory low level components for the system.

```
[aduggan@jcluvp-bld96 s_aosp_dolphin_sl_noip]$ ls
android  syna-release
[aduggan@jcluvp-bld96 s_aosp_dolphin_sl_noip]$ ls syna-release/
ampsdk  application  boot  build  configs  drm  external  factory  fw_enc  linux_5.4  Makefile  osal  security  synap  sysroot  ta_app  ta_enc  tee  toolchain
[aduggan@jcluvp-bld96 s_aosp_dolphin_sl_noip]$ ls android/
device  vendor
[aduggan@jcluvp-bld96 s_aosp_dolphin_sl_noip]$ 
```
Downloading Android

Downloads branch android-12.0.0_r32 of AOSP

```bash
[aduggan@sjc1uvp-bld06 build]$ cd aosp-s
[aduggan@sjc1uvp-bld06 aosp-s]$ repo init -u https://android.googlesource.com/platform/manifest -b android-12.0.0_r32
Get https://gerrit.googlesource.com/git-repo/clone.bundle
Get https://gerrit.googlesource.com/git-repo
Checking out files: 100% (137/137), done.
```

```bash
[aduggan@sjc1uvp-bld06 aosp-s]$ repo sync
remote: Counting objects: 464, done
remote: Finding sources: 100% (208/208)
remote: Total 208 (delta 62), reused 149 (delta 62)
Receiving objects: 100% (208/208), 690.91 KiB | 23.82 MiB/s, done.
Resolving deltas: 100% (62/62), completed with 9 local objects.
```
Android Directory

Then copy SDK’s android directory into the AOSP tree.

This integrates all of the Synaptics vendor specific files into Android.
Building the Android Image

The steps to build the SDK and an entire AOSP image are performed by the build_androidtv script.

```bash
$ ./vendor/synaptics/build/build_androidtv -p ./vendor/synaptics/dolphin/configs/aosp_dolphin_sl_noip -m /data/build/s_aosp_dolphin_sl_noip/syna-release
```

The script also allows only building specific steps. The command below will only build the SDK.

```bash
$ ./vendor/synaptics/build/build_androidtv -p ./vendor/synaptics/dolphin/configs/aosp_dolphin_sl_noip -m /data/build/s_aosp_dolphin_sl_noip/syna-release -s prepare_sdk
```
The Android Image

Once the build is complete the Android image will be located in `.vendor/synaptics/dolphin/build/aosp_dolphin_sl_noip/release/eMMCimg`

Unzip the dolphin-syna-image.zip file. The images will be extracted into a new eMMCimg directory. Then copy the factory_setting.subimg to the new eMMCimg. Now the resulting eMMCimg directory can be copied to a flash drive for installation onto the board.
Setting up the console

Debug board connects to the board with the white cable into the UART connector. (Gray cable not provided)
Flashing the Android Image

Flashing requires booting from an external SPI flash board. It contains a version of u-boot with support for reading the image from USB and writing it to the internal eMMC.

Run usb2emmc eMMCimg at the prompt.
Debugging

Android Studio and the Android NDK support debugging of apps.

To debug system level issues Android’s logcat is very useful.

[link to Logcat documentation]
Working with Cameras

The MIPI-CSI camera module and external USB cameras will be detected by the Android Camera HAL and will be accessible to Android applications.

The MIPI-CSI camera module plugs into the MIPI-CSI0 x4 Lane connector.
Support Requests using Jira

We will use Jira to handle support requests and questions.

One or two accounts associated with an @ucf.edu email will have access to Jira. When filing Jiras please provide as much information as possible. For example, please include console output, logcat output, sample code or binaries, and any other information which could be relevant.

Meet Patel will be the main support contact.

https://synacsm.atlassian.net/servicedesk/customer/user/login?destination=portals