While building Linux kernel in x86, the kernel parameters are not set in *${ANDROID\_PATH}/kernel/.config* but in *${ANDROID\_PATH}/device/Advantech/hit\_w121/hit\_w121\_defconfig*. You should *make menuconfig* in kernel and then merge *.config* to *hit\_w121\_defconfig*. Mind using *make mrproper* to clean the intermediate files in *kernel*.

Android雖包含kernel tree但kernel是以pre-built binary方式加進最後產生的鏡像文件中的。這做法在手機上沒有問題，反正要功能都預先知道。Take TI for example, the Linux kernel folder is independent from Android source tree and should be compiled beforehand。然對X86來說需求不同kernel也就不同。X86在編譯Android時加入kernel編譯，並將編譯結果加入鏡像文件中。原始碼加入kernel編譯規則(build/core/kernel.mk)，給X86預設的config存檔放在kernel/arch/x86/configs/android-x86\_defconfig。*hit\_w121\_defconfig* is the kernel configuration of *HIT\_W121* and located in *Android/device/Advantech/hit\_w121* folder. The folder named “kernel” is the Linux kernel source code。在編譯過程中會以這個config為主，產生kernel和對應子組塊。也可以單獨編譯kernel：$ *make kernel TARGET\_PRODUCT=eeepc*

也可用自己的config存檔。將它放入kernel/arch/x86/configs中，並指定變量TARGET\_KERNEL\_CONFIG(假設config 存檔名叫my\_defconfig)：*$ make iso\_img TARGET\_PRODUCT=eeepc TARGET\_KERNEL\_CONFIG=my\_defconfig*

You can compile the Linux driver (.ko) independently by assigning the path of kernel to Linux kernel of android source tree like *…/out/target/product/hit\_w121/obj/kernel* (Not the folder “kernel” in Android source tree, since we should choose the compiled binary files).

There are some problems while creating an Android usb image, use ISO image instead.

Mind that you should also add the parameters even if you just want to clean the intermediate files in Android source tree.

*# make clean TARGET\_PRODUCT=hiw\_w121 TARGET\_ARCH\_VARIANT\_NAME=x86-atom*.

Since the SmartCard API is part of the Android platform, the Eclipse development environment needs to be configured in order to use the compiled SDK with the integrated SmartCard API interface instead of the official SDK from Google. If not yet done, adapt the SDK location in Eclipse in order to register the API extension available in the development environment. Mind that you must build the android kernel (${ANDROID\_PATH}/out/target/product/hit\_w121…) successfully before building android sdk since some files (especially the header files in *${ANDROID\_PATH}/out/…/obj/include*) in android kernel are required while building sdk.  
The output of *make PRODUCT-sdk-sdk TARGET\_PRODUCT=x86* can be found under *$ANDROID\_ROOT/out/host/linux-x86/sdk/...*

* Open Eclipse and go to Windows -> Preferences -> Android
* Specify the SDK Location as *$ANDROID\_ROOT/out/host/linux-x86/sdk/android-sdk\_eng.root\_linux-x86*

Caution 1: Two files: *emulator* and *hardware-properties*.ini doesn’t exist while the compiler uses *tools.atree* compiling sdk. The probable root cause I guess is because some files are not complete in android source code. I just mark these two files and copy the files from android sdk released from google (*/home/price/andoird/android-sdk-linux\_x86*).

Caution 2: After building the android source tree successfully, you should start to compile the corresponding sdk for this version. For example, if the version of android source tree is 2.3.4, then the API version should be 2.3.4, too.

When using JNI in android, use C instead of CPP file because the dalvik virtual machine can’t locate the export symbol properly in CPP file.

There are two kinds of smart card: Crypto Card and Java Card. Crypto Card is developed by C land Java Card by Java.

Readers send APDU (Application Protocol Data Unit) command to smartcards and then readers send ATR (Answer To Request) back including the card information like card properties, companies and protocol.

Most smartcards just contain memory inside which are called memory card also. Lately, more advanced smartcards contain not only memory but also CPU so that they have calculation abilities. This kind of cards are called CPU card.

Smartcards exploit PKCS (Public Key Cryptography Standards) to encrypt.

Android module build stores the intermediate files in specified folder. If the product is *hit\_w121*, module name, a shared library, is *ezusb.so*, then all intermediate files are in *${ANDROID\_PATH}/out/target/product/hit\_w121/obj/SHARED\_LIBRARIES/ezussb\_intermediates/src*. The library exists *in ${ANDROID\_PATH}/out/target/product/hit\_w121/system/ezusb.so*.

In hit-w121, the brand of smart card reader is CASTLE tech, the user mode driver: ezusb.so is used to read the data from the device node as mentioned above. This driver is used to communicate the card (ICC) with reader (IFD). In android, *${ANDROID\_PATH}/external/pcsc* is the project to implement the pcsc functions. This project provides a set of APIs (IFD Handler) so that other components can control the smartcard system too. The pcsc daemon (running *pcscd –f –d*) reads the usb information (*/dev/bus/usb* for android 2.3 and */sys/bus/usb/devices* for android 2.2) at the fixed time interval.

In Android, the parameters for each different board/product are setting in *${ANDOIRD\_PATH}/device/${VENDOR\_NAME}/${PRODUCT\_NAME}/*. Generally, copy a similar board/product (Like same CPU, similar peripheral) as reference and then modify the parameters inside. Take HIT\_W121 for example, hit\_w121\_defconfig is the kernel menuconfig and use the files of ASUS and Viewsonic as reference. *AndroidBoard.mk* setup the parameters of kernel. This file calls the *hit\_w121\_info* file which defines the necessary drivers. For example, *FB0DEV=i915* means requiring framebuffer driver: *i915.ko*. In addition, *AndroidBoard.mk* also points out that the menuconfig file of kernel is *hit\_w121\_defconfig*. Same boards can have different products.

*AndroidProduct.mk* tells the Android building system to read *hit\_w121.mk* which defines: (1) Additional package/module. Add the package/module name into *PRODUCT\_PACKAGES*. The *PRODUCT\_PACKAGE* in *hit\_w121.mk* builds the external modules and packages which are not included in *build/target/product/core.mk*. The package/module name is from *LOCAL\_PACKAGE\_NAME*/*LOCAL\_MODULE*. (2) The name of vendor, device and product. (3) The file to be copied into target. In current case, *init.hit\_w121.sh* is copied. This file is a shell script so you can define some action in this file.

*BoardConfig.mk* setup the parameters of board. The most important setting in this file is *BOARD\_KERNEL\_CMDLINE* which will be added into Linux kernel command line. Kernel command line is sent from bootloader to kernel when the mission of bootloader is done and ready to start the kernel.

After kernel is loaded successfully, the first AP, *init*, reads *init.rc* and starts to run. In x86, the system reads *init.hit\_w121.rc* before *init.rc*. In *init.hit\_w121.rc*, define the service the Android is required and finally run *hit\_w121.sh*.

*System.prop* setups the parameters of Java object (The object in framework level) in Android like dpi…etc.

If you would like to copy a file (mfd-core.ko, for example) to the target, add the following command in AndroidBoard.mk.

*SUSI\_FOLDER:=susi*

*$(call add-prebuilt-target,$(TARGET\_OUT\_DATA\_ETC),$(SUSI\_FOLDER)/mfd-core.ko)*

For Android, *mfd-core.ko* locates in *android/device/Advantech/hit\_w121/susi* and is copied into *android/out/target/product/hit\_w121/system/etc/susi.* Then while installing to target, it is located in */system/etc/susi*.

*TARGET\_OUT\_DATA\_ETC* is defined in *android/build/envsetup.sh*. It fails to copy the file to some folders like */lib/modules*. So it doesn’t work to type the command: *$(call …,$(TARGET\_OUT)/lib/modules, mfd-core.ko)*. The probable reason may be that some folders are mounted to read-only partitions.

The touch screen library (ts\_lib) is integrated into Android to calibrate touch screen. To adjust the parameters of touch screen, do the following steps:

*# stop zygote /\* Stop Android UI \*/*

*# cd /data/system/tslib*

*# ll pointercal /\* The file of keeping track of touch screen parameters \*/*

*# ts\_calibrate /\* Start calibrate touch panel \*/*

*# cat pointercal /\* Read the content of the parameters to check if it is correct \*/*

*# start zygote /\* Restart Android UI \*/*

Android does not use *.config* in Kernel folder while building the Linux kernel of Android. After finishing setting the configuration of kernel, you have to update it to the config file in *device* folder. (For example: */device/Advantech/hit\_w121/hit\_w121\_defconfig*). Android uses the configuration file in *device* to build kernel.

In order that the GPIO can work properly in hit\_w121, three modules must be installed *mfd\_core.o*, *lpc\_sch.o* and *sch\_gpio.o*. In *mfd\_core.o*, two export symbols: *mfd\_add\_devices()* and *mfd\_remove\_devices()* are defined (These are defined as platform device). *lpc\_sch.c* is the driver of GPIO interface for Intel Poulsbo SCH. In *lpc\_sch.o*, *pci\_register\_driver()/pci\_unregister\_driver()* are used to register/unregister as PCI drivers (*/sys/bus/pci/driver/lpc\_sch*). Besides, in *XXX\_probe()*/*XXX\_remove()*, this module also calls *mfd\_add\_devices()*/*mfd\_remove\_devices()* to register/unregister the device (/sys/bus/pci/devices/0000:00:1f.0). You can see a link (*/sys/bus/pci/drivers/lpc\_sch/0000:00:1f.0/sch\_gpio*) to it. *sch\_gpio.c* is LPC interface for Intel Poulsbo SCH LPC bridge function of the Intel SCH contains many other functional units, such as Interrupt controllers, Timers, Power Management, System Management, GPIO, RTC, and LPC Configuration Registers. In *sch\_gpio.o*, *platform\_driver\_register()*/*platform\_driver\_unregister()* are called (*/sys/bus/plaftform/driver/sch\_gpio*) to register the platform driver. Then, in *XXX\_probe()*/*XXX\_remove()*, this module calls *request\_region()*/*release\_region()* to register/unregister the IO space (Use *platform\_get\_resource()* to get the resource of IO, memory,…, etc) so that it can use *inb()/outb()* to access the register. Besides, after you install *lpc\_sch.ko* and *sch\_gpio.ko*, the device nodes (/sys/module/lpc\_sch and /sys/module/sch\_gpio) are created and after calling gpio\_export() and gpio\_export\_link(), the device nodes like */sys/devices/susi/led\_left\_right/direction* and */sys/devices/susi/led\_left\_right/value* are created.