Android Software Reference Manual

Table of Contents

Warpx Software Reference Manual - This document

Introduction

Warpx is a Wearable Reference Platform, aimed at facilitating development and innovation for wearable products.

The purpose of this document is to help the user to quickly bring up the warpx board with the available software. The software is released in source codes or binary objects. The user can build the entire stack from source or install the binary images as explained in this guide. The latter is strongly recommended before building from sources and customizing the Android Operating System.

At the time of writing some procedures described in this document are not definitive and still under development by the warpx core team. This document will be constantly updated based on the latest software release and procedures on how to build and deploy.

Audience for this Guide

Warpx is a flexible platform that can be interacted with in a variety of different ways. The current software is based on Android 4.3.1.

Android developers, although covering a wide spectrum of users, can be mainly divided in two categories, application developers and embedded system developers.
Application developers focus on app development using the Android SDK. One of the primarily goals of warpx is to provide as many developers as possible the chance to design applications for wearable devices.
What an application developer needs to be operative is:

- A full bootable warpx Android operating system
- All the necessary Android SDK and IDE installed in the development host
- ADB interface working to debug and deploy applications

Embedded developers are more focussed on the development of embedded systems using the warpx based on Android or Linux (or even more on custom operating systems). Embedded developer need:

- a serial console to debug the entire boot sequence
- the proper tools to flash the various operating system images
- Android SDK installed in the host machine
- BSP and Android source code to build and customize the system images.

This guide's purpose is to provide the necessary information to both applications and embedded developers to maximize the potential of warpx by quickly enabling board bring up.

At the time of writing we used a Linux host machine for both application and system development. Windows and MAC OS X are not supported yet in this documentation. You can easily setup a Linux virtual machine using i.e Oracle VirtualBox that is free (https://www.virtualbox.org).

Getting Started

In this section we will go through some simple steps to setup your warpx and use the default Android 4.3 OS and example applications pre-installed.

We assume that your warpx setup is:
- warpx main board
- warpx interposed board
- Touch Screen

Refer to the Hardware Reference Manual for more information on how set up the warpx unboxing.

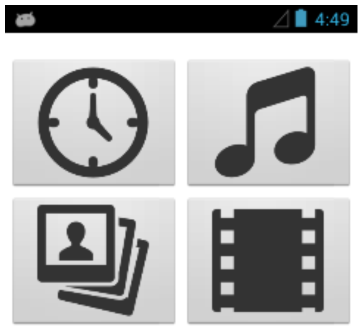
When you power up your warpx, (the first boot may take a while because the data partition is populated).
The device, out of the box, is already in developer debug mode and if the USB OTG cable is connected to the warpx board, the OS should notify you to store the HOST computer RSA key fingerprint to enable ADB access.
From the host machine you should be able to give:

```
$ adb devices
List of devices attached
0123456789ABCDEF     device
$ adb shell
root@warpx:/ #
```

You have root access to the warpx shell!

First encounter with the system

When the boot is completed the warpx Launcher is loaded.
The launcher presents all the demo applications in a vertical scrollable list.

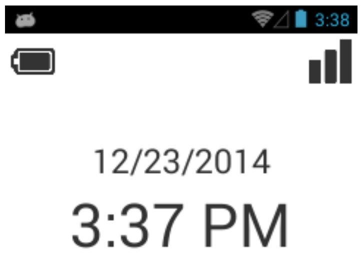


At the time of writing, Android is not configured to use the GPIO buttons on the interposer board as HOME and BACK (however it can be easily done). Android can emulate those buttons using the *input keyevent #*.
Refer to *https://developer.android.com/reference/android/view/KeyEvent.html*

Available applications are:

- Watch: show the time, stopwatch, program alarms
- Music: browse and play music
- Gallery: browse and show images
- Video: browse and play video
- Compass: show cardinal directions
- Freefall: detect when the board is falling (use at own risk!)
- Settings: simplified version of Android Settings. Allow to connect to Wifi, pair with Bluetooth, adjust Sound volumes, change Display settings, modify Date and time-zone.

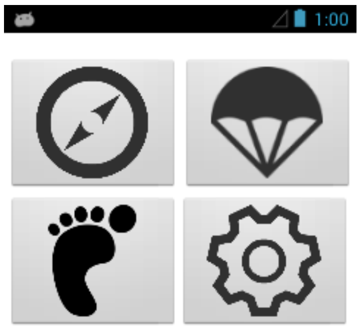
Pressing the BACK[1] button in the main screen brings to the status page.



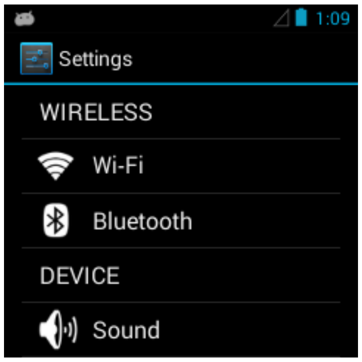
[1] At the present time, the platform does not support GPIO buttons on the interposer board as BACK and HOME Android buttons. It is possible to use software key emulation

Configure Wifi

To configure wifi network go to settings pressing the gear icon in the second launcher page



In the settings page press Wi-Fi



and select your network.
If necessary input[2] the passkey based on your authentication type.
Once the connection is established you should be able to ping the outside network:

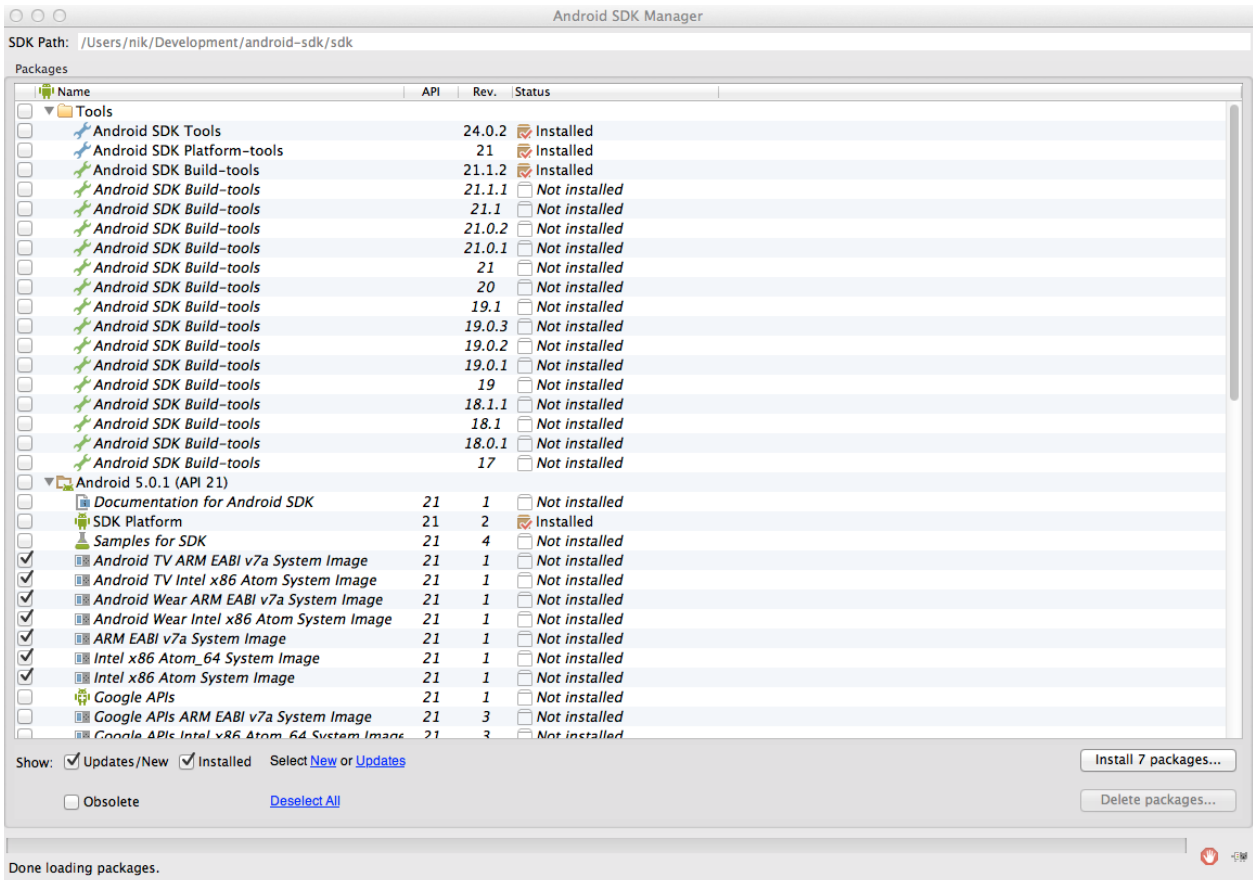
```
root@warpx:/ # ping google.com
PING google.com (173.194.115.66) 56(84) bytes of data.
64 bytes from dfw06s41-in-f2.1e100.net (173.194.115.66): icmp_seq=1 ttl=53 time=72.8 ms
64 bytes from dfw06s41-in-f2.1e100.net (173.194.115.66): icmp_seq=2 ttl=53 time=76.0 ms
64 bytes from dfw06s41-in-f2.1e100.net (173.194.115.66): icmp_seq=3 ttl=53 time=74.5 ms
...
```

[2] Today the standard Android keyboard is available.

Toolbox for Application Developers

An application developer can quickly start working with the warpx.
The board comes with Android 4.3 operating system preinstalled and applications examples released under GPL license. It can be helpful to start from these examples and customize the code, build and deploy on the warpx.
To develop software you can use your prefered OS. All development tools are available for all the platforms: Windows, Mac OSX, Linux.

First, download and install the Android SDK on your development host. All the information is available here: *http://developer.android.com/sdk/installing/index.html*.
To download the SDK for your platform go to: http://developer.android.com/sdk/index.html#Other.
The Android SDK Manager (you can find the binary, named *android*, in the "tool" directory of the SDK) will provide you all the available packages:


Android SDK Manager

SDK Path: /Users/nik/Development/android-sdk/sdk

Packages

	Name	API	Rev.	Status
☐	Tools			
☐	Android SDK Tools		24.0.2	Installed
☐	Android SDK Platform-tools		21	Installed
☐	Android SDK Build-tools		21.1.2	Installed
☐	*Android SDK Build-tools*		*21.1.1*	*Not installed*
☐	*Android SDK Build-tools*		*21.1*	*Not installed*
☐	*Android SDK Build-tools*		*21.0.2*	*Not installed*
☐	*Android SDK Build-tools*		*21.0.1*	*Not installed*
☐	*Android SDK Build-tools*		*21*	*Not installed*
☐	*Android SDK Build-tools*		*20*	*Not installed*
☐	*Android SDK Build-tools*		*19.1*	*Not installed*
☐	*Android SDK Build-tools*		*19.0.3*	*Not installed*
☐	*Android SDK Build-tools*		*19.0.2*	*Not installed*
☐	*Android SDK Build-tools*		*19.0.1*	*Not installed*
☐	*Android SDK Build-tools*		*19*	*Not installed*
☐	*Android SDK Build-tools*		*18.1.1*	*Not installed*
☐	*Android SDK Build-tools*		*18.1*	*Not installed*
☐	*Android SDK Build-tools*		*18.0.1*	*Not installed*
☐	*Android SDK Build-tools*		*17*	*Not installed*
☐	Android 5.0.1 (API 21)			
☐	*Documentation for Android SDK*	*21*	*1*	*Not installed*
☐	SDK Platform	21	2	Installed
☐	*Samples for SDK*	*21*	*4*	*Not installed*
✓	*Android TV ARM EABI v7a System Image*	*21*	*1*	*Not installed*
✓	*Android TV Intel x86 Atom System Image*	*21*	*1*	*Not installed*
✓	*Android Wear ARM EABI v7a System Image*	*21*	*1*	*Not installed*
✓	*Android Wear Intel x86 Atom System Image*	*21*	*1*	*Not installed*
✓	*ARM EABI v7a System Image*	*21*	*1*	*Not installed*
✓	*Intel x86 Atom_64 System Image*	*21*	*1*	*Not installed*
✓	*Intel x86 Atom System Image*	*21*	*1*	*Not installed*
☐	*Google APIs*	*21*	*1*	*Not installed*
☐	*Google APIs ARM EABI v7a System Image*	*21*	*3*	*Not installed*
☐	*Google APIs Intel x86 Atom_64 System Image*	*21*	*3*	*Not installed*

Show: [✓ Updates/New] [✓ Installed] Select New or Updates ☐ Obsolete Deselect All

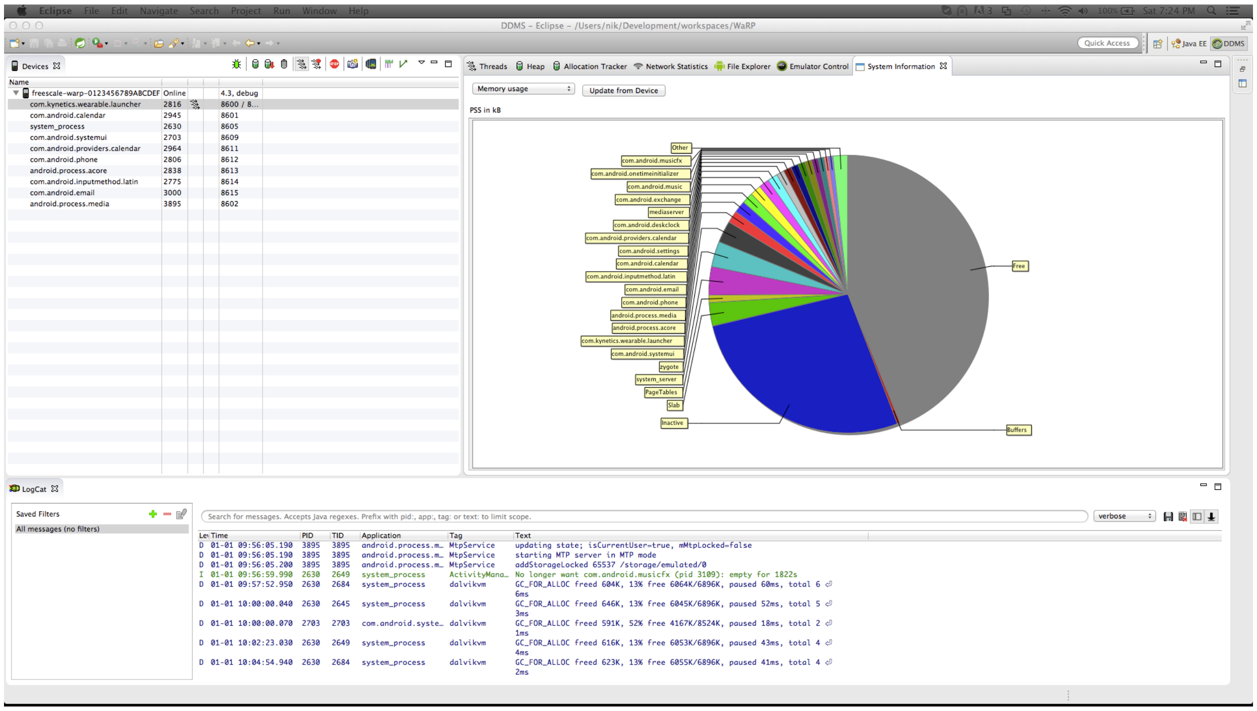
Done loading packages.

Required packages are:

- Android SDK Tools
- Android Platform Tools

- Android SDK build Tools
- Android 4.3.1 API 18

To debug the warpx, ADB is the primary command line tool (http://developer.android.com/tools/help/adb.html).
To write the warpx software Eclipse IDE (www.eclipse.org) has been used with the ADT plugin. Eclipse is available for all platforms.
Refer to the on line guide http://developer.android.com/tools/help/adt.html to install the ADT plugin.
Once you install the required packages and Eclipse + ADT, it's possible to easily debug the warpx inside the IDE environment.



Name				
freescale-warp-0123456789ABCDEF	Online			4.3, debug
com.kynetics.wearable.launcher	2816			8600 / 8...
com.android.calendar	2945			8601
system_process	2630			8605
com.android.systemui	2703			8609
com.android.providers.calendar	2964			8611
com.android.phone	2806			8612
android.process.acore	2838			8613
com.android.inputmethod.latin	2775			8614
com.android.email	3000			8615
android.process.media	3895			8602

```
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
Le Time               PID   TID   Application           Tag             Text
D  01-01 09:56:05.190 3895  3895  android.process.m…    MtpService      updating state; isCurrentUser=true, mMtpLocked=false
D  01-01 09:56:05.190 3895  3895  android.process.m…    MtpService      starting MTP server in MTP mode
D  01-01 09:56:05.200 3895  3895  android.process.m…    MtpService      addStorageLocked 65537 /storage/emulated/0
I  01-01 09:56:59.990 2630  2649  system_process        ActivityMana…   No longer want com.android.musicfx (pid 3109): empty for 1822s
D  01-01 09:57:52.950 2630  2684  system_process        dalvikvm        GC_FOR_ALLOC freed 604K, 13% free 6064K/6896K, paused 60ms, total 6
                                                                        6ms
D  01-01 10:00:00.040 2630  2645  system_process        dalvikvm        GC_FOR_ALLOC freed 646K, 13% free 6045K/6896K, paused 52ms, total 5
                                                                        3ms
D  01-01 10:00:00.070 2703  2703  com.android.syste…    dalvikvm        GC_FOR_ALLOC freed 591K, 52% free 4167K/8524K, paused 18ms, total 2
                                                                        1ms
D  01-01 10:02:23.030 2630  2649  system_process        dalvikvm        GC_FOR_ALLOC freed 616K, 13% free 6053K/6896K, paused 43ms, total 4
                                                                        4ms
D  01-01 10:04:54.940 2630  2684  system_process        dalvikvm        GC_FOR_ALLOC freed 623K, 13% free 6055K/6896K, paused 41ms, total 4
                                                                        2ms
```

The device, out of the box, is already in developer debug mode and if the USB OTG cable is connected to the warpx board, the OS should notify you to store the HOST computer RSA key fingerprint to enable ADB access.

Using Eclipse + ADT you can deploy your application directly on the warpx.
Another popular IDE is Android Studio: https://developer.android.com/training/basics/firstapp/index.html.

Today the most popular Android IDE is Android Studio. This guide is still referring to Eclipse + ADT plugin.

Toolbox for Embedded Developers

Warpx uses u-boot as bootloader with **fastboot** support.
Embedded developer first need to have access to the device console used to debug in the early stage of the boot process.
Warp_0x01 does not come with a physical serial console out of the box, therefore it's necessary to have the **Development Interposer Board (DIP)**. This is a development and debugging board that provides all the necessary amenities to the early stage board bring up.

Warpx kernel supports the g_multi gadget (using the USB OTG) which provide a serial interface once the module is loaded by the system. However this gadget support is not provided by uboot.

Different programs and utilities will be part of your toolbox depending on what you are trying to accomplish, this guide provide all this information.
To start deploying existing pre-build images you need:

- linux host machine with all the build essentials installed (compiler, linker, dev libraries, etc.)
- the Android SDK and Platform tools *http://developer.android.com/sdk/installing/index.html*
- Fastboot (you can find it in the Platform Tools)
- Minicom or other terminal emulation software

Update Android Binaries Packages

Binary images can be provided, over the time, to be flashed in the warpx. These images may contain the latest bugfix or new features. Update the entire system is very simple and straightforward.

Any system update is provided by the following images:

1. boot.img
2. recovery.img
3. system.img

Configure udev to support fastboot by defining a rule for the device ID **18d1:0d02** Google Inc.
This allows you to handle the attached device by a regular user, not root.

```
SUBSYSTEM=="usb",ATTR{idVendor}=="18d1",ATTR{idProduct}=="0d02",MODE="0640",OWNER="user"
```

where user is *your user* in the host machine.
Setup the warpx for the update:

1. connect the warpx USB cable to your host machine (this will transfer the file over fastboot)
2. connect the DIP USB cable to your host machine (this will carry the serial interface)
3. power up the DIP with 12V power supply

Power up the board and press the spacebar within 3 seconds to stop u-boot to load the operating system:

```
U-Boot 2013.04-00158-gf6b5254-dirty (Oct 13 2014 - 19:38:04)

CPU:   Freescale i.MX6SL rev1.2 at 792 MHz
CPU:   Temperature 45 C, calibration data: 0x55f4e45f
Reset cause: WDOG
Board: warpx Board
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0
MMC Device 1 not found
No MMC card found
Using default environment

In:        serial
Out:   serial
Err:   serial
MMC Device 1 not found
no mmc device at slot 1
Configuring display bridge
check_and_clean: reg 0, flag_set 0
Fastboot: Normal
flash target is MMC:0
RUNNING ESDHC INIT
Setting reg 0x021940C0 to 0x00000002
Normal Boot
Hit any key to stop autoboot:  0
=>
=>
=>
```

then launch fastboot (server) from the serial console.

```
=> fastboot
fastboot is in init......USB Mini b cable Connected!
fastboot initialized
USB_SUSPEND
USB_RESET
USB_PORT_CHANGE 0x4
USB_RESET
USB_PORT_CHANGE 0x4
USB_RESET
USB_PORT_CHANGE 0x4
```

In the host machine verify the fast boot connection with:

```
$ fastboot devices
12345    fastboot
```

and then flash the three images:

```
$ fastboot flash boot boot.img
sending 'boot' (4266 KB)...
OKAY [  0.628s]
writing 'boot'...
OKAY [  0.197s]
finished. total time: 0.825s

$ fastboot flash recovery recovery.img
sending 'recovery' (4806 KB)...
OKAY [  0.708s]
writing 'recovery'...
OKAY [  0.225s]
finished. total time: 0.933s

$ fastboot flash system system.img
sending 'system' (286720 KB)...
OKAY [ 40.104s]
writing 'system'...
OKAY [ 11.470s]
finished. total time: 51.576s
```

In the meantime you will see output messages in the console that shows the writing process (that we omit here)

Now exit from fastboot mode in the host machine with[3]:

```
$ fastboot continue
resuming boot...
OKAY [  0.006s]
finished. total time: 0.007s
```

Now press the BT1 on the DIP board for 3 seconds and the system will reboot.

[3] Because in u-boot if you press 'enter' you actually *repeat* the last command you gave (i.e the fastboot command), **be aware to not press enter when in fastboot mode** or you will open another fastboot session. Here you need to give fastboot continue many times as you pressed enter.

Board Bring Up

In this section we will go through the process of flashing a pre-built Android image on the warpx from scratch. This may be applied if you want to update the bootloader, custom board bring up or severe system failure.

The board is shipped with a bootloader so this procedure can be skipped by the reader if not necessary or you are an application developer.

It's is assumed you know the basic of AOSP, embedded systems and at least have a good handle on how Linux works and how to interact with its command line. A serial console is required in order to go through the contents of this section. To have access to the serial console you need to have the warpx Interposer board.

The process that we describe is useful to anyone who wants to start from scratch in case of some kind of board early stage bring up. In practice this means that when the device is powered down,and u-boot must be transferred into RAM

Configure udev

There are some minor tweaks that are required in the host machine to properly configure the various USB devices. The different USB devices you will be interacting with are:

1. ID **0403:6010** Future Technology Devices International, Ltd FT2232C Dual USB-UART/FIFO IC
2. ID **15a2:0063** Freescale Semiconductor, Inc. (USB OTG on the warpx)
3. ID **18d1:0d02** Google Inc. (when using fastboot)
4. ID **18d1:4e42** Google Inc. (ADB interface when Android is loaded)

To operate properly (2),(3),(4) udev rules should be defined in order to have the right permission to access those devices. For this purpose a set of udev rules should be created in particular:

```
SUBSYSTEM=="usb",ATTR{idVendor}=="15a2",ATTR{idProduct}=="0063",MODE="0640",OWNER="user"
SUBSYSTEM=="usb",ATTR{idVendor}=="18d1",ATTR{idProduct}=="0d02",MODE="0640",OWNER="user"
SUBSYSTEM=="usb",ATTR{idVendor}=="18d1",ATTR{idProduct}=="4e42",MODE="0640",OWNER="user"
```

where *user* is the user that will interact with the system (not root)

U-boot flash and eMMC partitioning

There are two way to load a new uboot on the system. The fastest way is to use the NXP MFC Tools modified for warpx. This can be downloaded from warpx.io. NXP MFC runs only under Windows.

If you are using Linux and you DON'T want to use Windows (or you don't simply use Windows) you need to load Uboot, Kernel and a Linux ramdisk in memory using an external tool called **usb-loader**.

Refer to warpx.io/resources for the binary kernel image and the RAM disk necessary to this operation.

To be able to manually load a prebuilt U-Boot the Interposer Board Boot mode header (BM0 jumper) should be closed: this forces the serial bootloader mode. See separate "Interposer Guide" to identify the Boot mode header on your Interposer Board.
The following procedure describes how to load into RAM a new u-boot.
For this purpose we need use imx_usb tool.
The **imx_usb** tools allows to easily load binaries in memory and start U-Boot.
To download and build imx_usb from sources:

```
$ git clone https://github.com/warpxboard/imx_usb_loader.git -b warpx/master
$ cd imx_usb_loader
$ make
```

You will need the libusb1 development package installed into your linux host.
Now plug the warpx board in the warpx Interposer Board set the BM0 jumper and connect its serial (micro usb) port to the host computer. You should see

```
$ lsusb
Bus 001 Device 023: ID 15a2:0063 Freescale Semiconductor, Inc.
```

You could open your minicom pointing to the USB serial port

```
$ minicom -D /dev/ttyUSB0
```

You need now the warpx u-boot binary, and the *rootfs.img* binary (downloaded from warpx.io or built from sources) to be loaded into warpx using imx_usb. If you place the u-boot binary in the same directory of the imx_usb binary you should just give[4]

```
$ ./imx_usb
```

Once the u-boot is loaded, on the client side give (your host machine) give:

[4] The u-boot binary version is only indicative.

```
$ fastboot devices
12345 fastboot usb:3-2.3
```

```
$ fastboot boot rootfs.img
downloading 'boot.img'...
OKAY [  0.352s]
booting...
OKAY [  0.009s]
finished. total time: 0.361s
```

When the linux system is loaded, login as root (empty password) and if you need to partition the eMMC for Android just give the following command:

```
# sfdisk -L /dev/mmcblk0 < part_table
```

File part_table is preloaded in the Linux ramdisk.
Now three partitions need then to be formatted with:

```
# mkfs.ext4 -L data /dev/mmcblk0p4
# mkfs.ext4 -L cache /dev/mmcblk0p6
# mkfs.ext4 -L vendor /dev/mmcblk0p7
```

To flash the u-boot just create the following shell script (if not already present in the root dir):

```
#!/bin/bash
#make boot partition writable
echo 0 > /sys/block/mmcblk0boot0/force_ro && sync
#clear uboot params
dd if=/dev/zero of=/dev/mmcblk1 bs=512 seek=1536 count=16
#write uboot
dd if=uboot.imx of=/dev/mmcblk1boot0 bs=512 seek=2 && sync
```

And execute it after setting the 755 permissions.
Then you can reboot the system.

Flashing Android

The system is now ready to be flashed with fastboot as explained in here.

Android booting

Android now can be booted from U-Boot with the command:

```
=> booti mmc0
```

and in the console terminal we should have

```
booti mmc0
kernel   @ 80808000 (4126948)
ramdisk  @ 81800000 (237385)
kernel cmdline:
        use boot.img command line:
        console=ttymxc0,115200 init=/init androidboot.console=ttymxc0 androidboot.hardware=freescale csi

Starting kernel ...

Initializing cgroup subsys cpuset
Initializing cgroup subsys cpu
Linux version 3.0.35-06447-gd615ab2 (developer@droidbake-vm) (gcc version 4.6.x-google 20120106
(prerelease)4
CPU: ARMv7 Processor [412fc09a] revision 10 (ARMv7), cr=10c53c7d
CPU: VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: warpx Board Wearable Reference Platform
Memory policy: ECC disabled, Data cache writeback
CPU identified as i.MX6SoloLite, silicon rev 1.2
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 117760
Kernel command line: console=ttymxc0,115200 init=/init androidboot.console=ttymxc0
androidboot.hardware=freei
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 464MB = 464MB total
Memory: 457316k/457316k available, 66972k reserved, 0K highmem
…
Freeing init memory: 204K
adb_open
…
root@warpx:/ #
130|root@warpx:/ #
```

Functional Test for key Devices.

The following section covers some functional test for key devices

This section will be updated with more test cases

Checking the Wifi

To check if wifi is functional first look to see if the interface loads correctly by command line:

```
root@warpx:/ # svc wifi enable

## wifi_probe
wifi_set_power = 1
root@warpx:/ # wifi_set_carddetect = 1
F1 signature read @0x18000000=0x16844330
DHD: dongle ram size is set to 294912(orig 294912)
wl_create_event_handler thr:da9 started
p2p0: P2P Interface Registered
dhd_attach thr:dae started
dhd_attach thr:daf started
dhd_attach thr:db0 started
Broadcom Dongle Host Driver: register interface [wlan0] MAC: 00:90:4c:11:22:33

Dongle Host Driver, version 5.90.195.104
Compiled in drivers/net/wireless/bcmdhd on Nov 19 2014 at 17:20:29
wifi_set_power = 0
=========== WLAN placed in RESET ========

Dongle Host Driver, version 5.90.195.104
Compiled in drivers/net/wireless/bcmdhd on Nov 19 2014 at 17:20:29
wl_android_wifi_on in
wifi_set_power = 1
=========== WLAN going back to live  ========
sdio_reset_comm():
dhdsdio_write_vars: Download, Upload and compare of NVRAM succeeded.
add wake up source irq 104
Firmware up: op_mode=4, Broadcom Dongle Host Driver mac=00:37:6d:16:82:fa
p2p0: p2p_dev_addr=02:37:6d:16:82:fa
```

Then we can check the signal strength by loading the wifi settings by command line:

```
root@warpx:/ # am start -a android.intent.action.MAIN -n \ com.android.settings/.wifi.WifiSettings
```

Checking FXOS chip (accelerometer)

```
root@warpx:/ # dmesg | grep fxos
```

A fxos that is functioning correctly will return:

```
root@warpx:/ # dmesg | grep fxos
<4>fxos8700_device_init succ
```

A fxos that is not functioning correctly will return:

```
root@warpx:/ # dmesg | grep fxos
<3>fxos 8700 read chip ID 0x0 is not equal to 0xc7 or 0xc4
<4>fxos8700: probe of spi1.0 failed with error -22
```