Clearwater Pass Hypervisor

Getting Started Guide

December 2017

Intel Confidential

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: <http://www.intel.com/design/literature.htm>

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at h[ttp://www.intel.com/](http://www.intel.com/) or from the OEM or retailer.

Intel, Graphics Virtualization Technology and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2018, Intel Corporation. All rights reserved.

Contents

[1.0 Introduction 6](#_Toc502810116)

[1.1 Terminology 6](#_Toc502810117)

[2.0 Host System Setup 8](#_Toc502810118)

[2.1 Host Development System 8](#_Toc502810119)

[2.2 Pre-Built Files and Packages 8](#_Toc502810120)

[2.3 Installing the Flash Tools 9](#_Toc502810121)

[3.0 Target System Setup 10](#_Toc502810122)

[3.1 Hardware Introduction 10](#_Toc502810123)

[3.2 CPU Stepping Identification 11](#_Toc502810124)

[3.3 Main Board Fab Revision Identification 11](#_Toc502810125)

[3.4 Debug Adapter Setup 12](#_Toc502810126)

[3.5 Board Connections 13](#_Toc502810127)

[4.0 Software Setup 14](#_Toc502810128)

[4.1 Flashing the I/O Controller (IOC) firmware 14](#_Toc502810129)

[4.2 Flashing the Automotive Boot Loader (ABL) Firmware 15](#_Toc502810130)

[4.3 Installing CWP/SOS/LaaG 16](#_Toc502810131)

[4.4 Installing CWP/SOS/AaaG 18](#_Toc502810132)

[5.0 Boot CWP/Service OS and Start User OS 22](#_Toc502810133)

[5.1 Boot the Hypervisor 22](#_Toc502810134)

[5.2 Commands Supported by the Hypervisor Shell 22](#_Toc502810135)

[5.3 Starting the Linux Guest OS 23](#_Toc502810136)

[6.0 Login to the System 26](#_Toc502810137)

[7.0 Build cwp/sos/Linux Guest Source 27](#_Toc502810138)

[7.1 Install Dependent Packages 27](#_Toc502810139)

[7.2 Download the Source from the Release Server 27](#_Toc502810140)

[7.3 Create the Source Tree Structure 27](#_Toc502810141)

[7.4 Build Source 28](#_Toc502810142)

[8.0 Build Android as a Guest Source 29](#_Toc502810143)

[8.1 Installing Packages 29](#_Toc502810144)

[8.1.1 Install the Latest Versions from Canonical\* 29](#_Toc502810145)

[8.1.2 Install Additional Packages 30](#_Toc502810146)

[8.1.3 Install an Additional Package for Openssl Dependency 30](#_Toc502810147)

[8.1.4 Install OpenJDK 30](#_Toc502810148)

[8.2 Create Workspace Directory 30](#_Toc502810149)

[8.3 Get the Latest Version of the Google Repo Tool 31](#_Toc502810150)

[8.4 Get the Source from our External Release Server (ultramobile-kits.org) 31](#_Toc502810151)

[8.5 Build the Android as a Guest Stack 31](#_Toc502810152)

[8.6 Build Output 31](#_Toc502810153)

[9.0 Build ROOT File System 33](#_Toc502810154)

[9.1 Setting up your Host to use the Yocto Project 33](#_Toc502810155)

[9.1.1 Build Host 33](#_Toc502810156)

[9.1.2 Additional Packages 33](#_Toc502810157)

[9.2 Extracting the DOM0/DOMU Tarball 34](#_Toc502810158)

[9.3 Setting up the Yocto Build Environment 35](#_Toc502810159)

[9.4 Creating the Yocto Build Directory 36](#_Toc502810160)

[9.5 Build Linux Guest Userland 37](#_Toc502810161)

[9.6 Build the Linux Service OS Userland 37](#_Toc502810162)

[9.7 Generate the Final Deployable Image 38](#_Toc502810163)

[10.0 Working with the Application Software 39](#_Toc502810164)

[10.1 glmark Benchmark 39](#_Toc502810165)

[10.2 Prioritized Rendering 39](#_Toc502810166)

[10.3 Hyper\_DMABUF Surface Sharing 39](#_Toc502810167)

[10.3.1 For Android as a Guest Running on CWP 40](#_Toc502810168)

[10.3.2 For Linux as a Guest Running on CWP 40](#_Toc502810169)

[10.4 Planes Restriction 41](#_Toc502810170)

[10.4.1 The i915.domain\_plane\_owners Parameter 42](#_Toc502810171)

[10.4.2 The i915.avail\_planes\_per\_pipe Parameter 42](#_Toc502810172)

[10.4.3 Modifying the Kernel Command Line Parameter Values 43](#_Toc502810173)

[10.4.4 Settings Without eDP Display Support 44](#_Toc502810174)

[10.4.5 Settings with eDP Display Support 45](#_Toc502810175)

[10.5 Direct Display Mode 45](#_Toc502810176)

[10.6 Multi-Planes 46](#_Toc502810177)

Figures

[Figure 1. Photo of GP MRB D0 Board (in Chassis) 11](#_Toc502810240)

[Figure 2. CPU D0 Stepping on Chassis Identification Label 12](#_Toc502810241)

[Figure 3. Main Board Fab on Chassis Identification Label (Fab E) 12](#_Toc502810242)

[Figure 4. Main Board Fab Identification on the MRB (Fab E) 13](#_Toc502810243)

[Figure 5. Debug Adapter Key Components 13](#_Toc502810244)

[Figure 6. MRB Cable Connections 14](#_Toc502810245)

[Figure 7. Example Kernel Command Parameters 45](#_Toc502810246)

Tables

[Table 1. Terminology 7](#_Toc502810247)

[Table 2. Supported Commands in the Hypervisor Shell 23](#_Toc502810248)

Revision History

|  |  |  |
| --- | --- | --- |
| Date | Revision | Description |
| December 2017 | 1.31 | Add eDP display support |
| December 2017 | 1.3 | Updated to support the hypervisor r03 release. |
| October 2017 | 1.2 | Clearwater\_Pass\_Hypervisor\_GSG-r02-2017WW41.5  Updated to support AaaG and hypervisor r02 codebase for GP MRB D0 only. |
| August 2017 | 1.1 | Updates to account for MRBs with a D0 stepping SoC. |
| July 2017 | 1.0 | Initial release. |

# Introduction

Intel delivers regular automotive hypervisor (code named Clearwater Pass) engineering code releases to enable Gordon Peak Modular Reference Board (GP MRB) hardware from Intel for platform evaluation purposes only. For the hypervisor release, Intel is also providing Linux\* Service OS, Linux Guest OS, and Android\* as a Guest as reference software for Clearwater Pass.

Intel is delivering one release to support two configurations:

1. CWP, Service OS, and Linux as a Guest (LaaG)
2. CWP, Service OS, and Android as a Guest (AaaG)

That means customers are able to bring up any one of them with the release.

This Getting Started Guide (GSG) provides information for setting up your host computer, hardware kit, and the tools used for programming and flashing firmware and pre-built binaries. In this GSG, we also provide step-by-step instructions for building source.

Please refer to the release notes for details of the r03 code release.

Release packages are here:

<https://ultramobile-kits.org/dana/fb/smb/wfb.cgi?t=p&v=resource_1423611128.573830.3%2Coa&si=&ri=&pi=&sb=name&so=asc&dir=gordon_peak_cwp%5Ca81%5C20171214_WW50>

## Terminology

Table . Terminology

| Term | Description |
| --- | --- |
| AaaG | Android as a Guest |
| ABL | Automotive Boot Loader |
| APL | Apollo Lake System on Chip. An Intel® architecture SoC that integrates the next generation Intel processor core, graphics, memory controller and I/O interfaces. |
| BIOS | Basic Input Output System |
| BSP | Board Support Package |
| DM | Device Model |
| ELK | Emergency Linux Kernel |
| GP | Gordon Peak, a Modular Reference Board (MRB) for the first cycle of Apollo Lake SoC Engineering Samples |
| IOC | I/O Controller |
| IVE | In Vehicle Entertainment |
| LaaG | Linux as a Guest |
| MRB | Modular Reference Board |
| PFT | Intel® Platform Flash Tool (Intel® PFT) for flashing images |
| SDC | Software Defined Cockpit |
| SoC | System-on-a-Chip |
| SPI | Serial Peripheral Interface |
| VCP | Virtual COM Port |
| Yocto | Yocto project; <http://www.yoctoproject.org> |

§

# Host System Setup

In this section, we’ll explain how to set up your development system and necessary software packages.

## Host Development System

The preferred (and tested) development host platform is a PC running Ubuntu\* 14.04 or 16.04. You’ll use this system to communicate with the GP MRB evaluation platform. We created this documentation based on validation using Ubuntu 14.04.

## Pre-Built Files and Packages

Pre-built binaries and tools are provided in the release to help you get started quickly with the Automotive Hypervisor, Linux Service OS, Linux Guest OS, and Android Guest OS on the Apollo Lake GP MRB platform. Download the files and packages described below from the release server into your ~/Downloads folder.

| Filename | Description |
| --- | --- |
| gordon\_peak\_cwp\_LaaG.img | This is the root filesystem image for the User OS, known as Linux Guest OS. It has an integrated kernel and userspace. |
| gordon\_peak\_cwp\_AaaG.img | This is the Android as a Guest partition image. In general, the Android OS contains several partition images, but here we build everything for Android into a data partition. |
| ifwi\_gr\_mrb\_b1.bin | This is the Automotive Bootloader (ABL) firmware for GP MRB D0. It’s a single release to support both Linux Guest OS and Android as a Guest. It also includes part of the fastboot protocol to be used for communicating with the fastboot tool running on the host side for flashing partition images. |
| ioc\_firmware\_gp\_mrb\_fab\_e.ias\_ioc | This is I/O Controller (IOC) firmware for GP MRB D0. See the Mainboard Fab Identification section below for details. This release only provides IOC firmware for D0 Feb\_e. |
| partition\_desc.bin | This is the binary image for GPT partitions. |
| sos\_boot.img | This Service OS boot image contains CWP hypervisor and SOSF kernel |
| sos\_rootfs.img | This is the root filesystem image for the Linux Service OS. It contains the Device Models implementation and SOS user space. |
| flash\_AaaG.json | Configuration file for Intel® Platform Flash Tool (Intel® PFT) to flash Android guest OS image + hypervisor/SOS boot image and Linux Service OS userland |
| flash\_LaaG.json | Configuration file for Intel® PFT to flash Linux Guest OS images + hypervisor/SOS boot image + Linux Service OS userland |
| platformflashtool\_5.8.5.0\_linux\_x86\_64.deb | This package provides flashing tools including the ias-spi-programmer needed to flash ABL firmware - automotive bootloader firmware, the ioc\_flash\_server\_app needed to flash the I/O Controller (IOC), fastboot, and Intel PFT for flashing images |
| oecore-x86\_64-corei7-64-toolchain-nodistro.0.sh | The SDK tool used for building device modules (DM). |

You’ll use these files later when you setup the software on the evaluation board in Chapter 4.0, “Software Setup”.

## Installing the Flash Tools

Install the Platform Flash tool version 5.8.5.0 (downloaded from VIP) on your Ubuntu host development system using the following commands:

$ sudo dpkg -i ~/Downloads/platformflashtool\_5.8.5.0\_linux\_x86\_64.deb

# **Note**: If there are any package dependency issues during installation, run the command below to fix them.

$ sudo apt-get -f install

When the installation is complete, you’ll find Intel PFT installed in the /opt/intel/platformflashtool/ folder.

§

# Target System Setup

The evaluation kit includes:

* Gordon Peak Module Reference Board (BP MRP) D0
* Debug adapter (for firmware flashing)
* RS232 communication cable
* Power cable

These items are needed, but not included in the evaluation kit:

* USB-to-micro USB cables
* HDMI Type A monitor cable to connect a monitor to the board

In the following sections, we explore the target system components and how to set them up.

## Hardware Introduction

The Gordon Peak Modular Reference board (GP MRB) is the Apollo Lake Engineering Samples MRB unit provided to selected customers for evaluation purposes only.

Figure . Photo of GP MRB D0 Board (in Chassis)

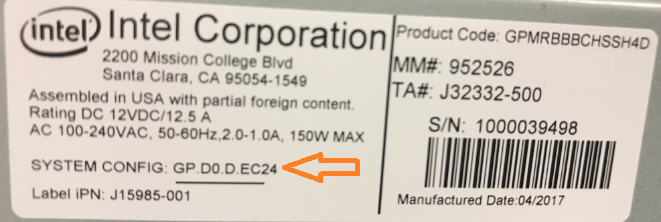


## CPU Stepping Identification

You need to know the CPU stepping and the board Fab revision for your MRB so you can choose the right firmware to load.

Find the label on the metal chassis with the SYSTEM CONFIG identification such as “SYSTEM CONFIG GP.D0.D.EC24” shown in Figure 2.

Figure . CPU D0 Stepping on Chassis Identification Label



The “GP.D0” indicates (for this system) that the CPU stepping is D0. The CPU stepping depends on the specific MRB you are using.

## Main Board Fab Revision Identification

You also need to identify the mainboard Fab revision to choose the right IOC firmware to use. On the same label that identifies the CPU stepping, you can also identify the board Fab revision, encoded in the last three digits of the TA#. In the label shown in Figure 3, it is “-500”.

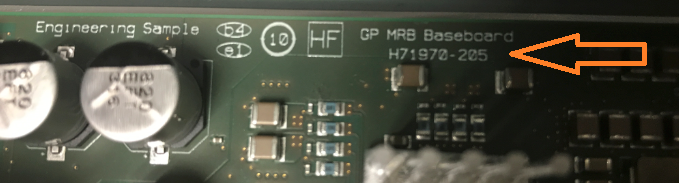
Figure . Main Board Fab on Chassis Identification Label (Fab E)



The first digit of this part of the TA# indicates the Fab: -1xx indicates Fab A, -2xx indicates Fab B, and so on. -5xx (as shown in the figure) indicates Fab E.

The mainboard Fab is also encoded in the product code printed on the board itself. For example, see Figure 4.

Figure . Main Board Fab Identification on the MRB (Fab E)



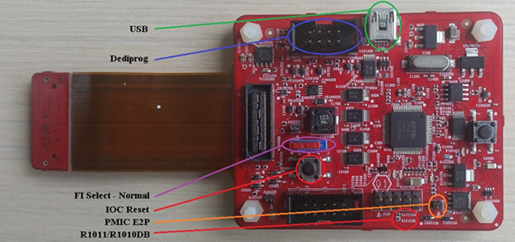
The last three digits encode the Fab, “-205”. In this case, it is the last digit that is important: -xx1 indicates Fab A, -xx2 indicates Fab B, and so on. “-xx5” (as shown in the figure) indicates Fab E.

With the Fab revision information identified, you can now choose the correct I/O controller (IOC) firmware to flash onto the board. For a Fab E board (as shown in the figure), we need to use the ioc\_firmware\_gp\_mrb\_**fab\_e**.ias\_ioc binary image.

## Debug Adapter Setup

A Debug Adapter is included in the evaluation kit for flashing the IOC and ABL firmware. Connect the power cables to the MRB board and an RS232 communication cable (provided) between the board and Debug Adapter.

Figure . Debug Adapter Key Components



When you connect the Debug Adapter to your host computer, you see four USB serial ports available:

$ ls /dev/ttyUSB\*

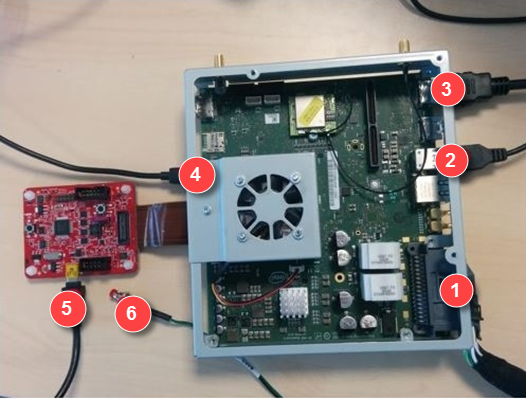
/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3

Device /dev/ttyUSB2 is the IOC serial console and /dev/ttyUSB3 is the ABL console. Configure your serial communication tool (for example, minicom) to use a baud rate of 115200 when using these ports.

## Board Connections

Use Figure 5 to identify where cables are connected to the evaluation board, Debug Adapter and host computer.

Figure . MRB Cable Connections



Legend:

1. Power supply connection
2. 2x USB ports used to connect peripherals, such as a keyboard or external USB disk
3. 2x HDMI ports
4. 1x USB On-The-Go (OTG) port
5. 1x USB port on the debug adapter (acting as four virtual serial ports)
6. Reset button used to reset the board
7. The evaluation kit does not include USB cables. These cables must be supplied by the user.

# Software Setup

In Section 2.2, “Pre-Built Files and Packages”, we indicated where to find and download the binary images for flashing the IOC and ABL firmware. In this section, we use those files (placed in the ~/Downloads folder). We recommend flashing both the IOC and ABL firmware when you get the evaluation kit. For this procedure, we used the GP MRB D0 board.

## Flashing the I/O Controller (IOC) firmware

1. Press and hold the IOC-reset button on the Debug Adapter, then run the flashing command:

$sudo /opt/intel/platformflashtool/bin/ioc\_flash\_server\_app -s /dev/ttyUSB2 –grfabd -t ioc\_firmware\_gp\_mrb\_fab\_e.ias\_ioc

------------------------------------------------

IOC Flash Server

------------------------------------------------

Trying to open port: /dev/ttyUSB2

Setting baudrate to 115200 Bit/s

Flash application section

Restarting the IOC into the bootloader (terminal request)

Setting Hardware Handshake to disable

Waiting for IOC bootloader

Checking for unexpected data on serial port / press reset button on debug adapter, if this message is seen for more then 2s ... done

Bootloader request detected.

Waiting for IOC information.

1. Release the IOC-reset button and the IOC Flash Server app continues:

Unknown hardware revision 0x02

Flash boot loader major-minor: 3-3

Starting to flash firmware: ioc\_firmware\_gp\_mrb\_fab\_e.ias\_ioc

Erasing flash.

Data frame number send: 2376|2376

Flashing was successful.

Restarting the IOC

In the example shown above, the system confirmed that IOC firmware flashing completed successfully.

## Flashing the Automotive Boot Loader (ABL) Firmware

The Automotive Bootloader (ABL) firmware supports fastboot to flash images.

1. Unplug the power cable, press and hold the “ignition” button (item 6 in the board connections picture above) before plugging the power cable back in. After five seconds, release the “ignition” button, then run the following command to flash ABL.

$ sudo /opt/intel/platformflashtool/bin/ias-spi-programmer --write ifwi\_gr\_mrb\_b1.bin

Intel IAS SPI Programmer - Linux build. Version 1.1

-------------------------------------------------------------

Programming device information: FTDI, Quad RS232-HS

Flash vendor ID is: 0x20

Flash ID is: 0xBB

Capacity: 0x17 (64MBit)

Data size to write: 8388608 Bytes

Start reading flash for write comparison

Start erasing underlying flash area: ............................

Start writing data to flash: .........10.........20.........30.........40.........50.........60.........70.........80.........90.........100

Skipped pages due empty or equal content: 19519

Start write verification

Write verification completed successfully

Execution time: 143 sec.

In the example shown above, the system confirmed that the ABL firmware was successfully installed.

After installing the IOC firmware and the ABL firmware on the GP MRB board, you can install the hypervisor, Linux Service OS, and Guest OS. In this release, we provide two Guest OS reference partition images, *LaaG* and *AaaG*.

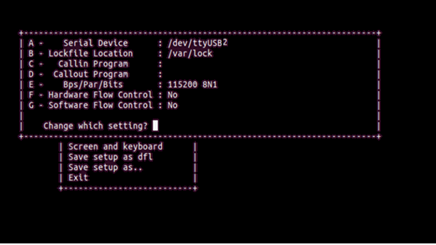
* To install LaaG, follow the instructions in Section 4.3.
* To install AaaG, follow the instructions in Section 4.4.

## Installing CWP/SOS/LaaG

1. Connect a USB cable from the debug board to your Ubuntu host machine, and run the following command to verify that its four USB serial ports are discovered and showing under /dev.

$ ls /dev/ttyUSB\*

/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3

1. Run minicom -s to verify the minicom serial port settings are 115200 8N1 and both HW and SW flow control are turned off. Also, change the Serial Device to /dev/ttyUSB2. An example screen is shown below:  
     
   
2. Leave the session up and open another minicom session, this time connecting to /dev/ttyUSB2 (for the IOC console) with command shown below:

sudo minicom -s

1. In this IOC console session, press ‘r’ to reboot the system. You may notice that the CPU fan stops rotating momentarily. An example of the IOC console log is shown below:

00:00 00s023ms Module cbc\_logic\_core : Init

00:00 00s024ms Module can\_lifecycle : Init

00:00 00s024ms CAN Lifecycle - using NVM values

00:00 00s024ms Lifecycle: PreRuntime: 1000 ms, require 1 message(s) repetitions to start

00:00 00s024ms CAN ID 0x1A, dlc 4, byte[2]

00:00 00s024ms Signal mask 0x7, bit shift: 0 bits

00:00 00s024ms startup value [min/max] = 0x01/0x07

00:00 00s024ms shutdown value [min/max] = 0x00/0x00

00:00 00s024ms S3 value [min/max] = 0xFF/0xFF

00:00 00s025ms Module can\_gw\_filter : Init

00:00 00s025ms Runtime Measurement calibration offset: 000 ns

00:00 00s025ms User CAN matrix is used: 0

00:00 00s025ms Automatic cold start disabled

00:00 00s025ms Max allowed startup temperature 125 C, max allowed temperature during runtime 125 C

00:00 00s025ms Min allowed startup battery voltage 8.00 V, min allowed battery voltage during runtime 6.00 V

00:00 00s025ms Recovery from S3 on SoC timer event supported

00:00 00s025ms ------------------- Logic Core Initialisation done -------------------

1. When the reboot completes, press ‘n’, then ‘4’ and ‘Enter’ to boot the system into fastboot mode. In this mode, the fastboot command is available to communicate from the host to the target, which lets you erase and flash partition images onto the board’s eMMC flash memory. An example of the IOC console log is shown below:

00:00 59s386ms master\_state\_engine -> wait for CBC - start synced peripherals

00:00 59s386ms cbc\_logic\_core -> startup(1 - early)

00:00 59s387ms CBC kmod versionless startup detected.

00:00 59s387ms master\_state\_engine -> wait for all modules

00:00 59s387ms cbc\_logic\_core -> startup(2 - late)

00:00 59s388ms cbc\_logic\_core -> on state

00:00 59s391ms console\_queue -> startup(1)

00:00 59s391ms timestamp -> startup(1)

00:00 59s396ms console\_queue -> on state

00:00 59s396ms timestamp -> on state

00:00 59s629ms abl\_control\_uart -> on state

00:00 59s629ms Startup Counter: 1

00:00 59s629ms System Start completed! (ABL Blob 1)

00:00 59s629ms master\_state\_engine -> on state

00:00 59s629ms ------------------------------------------------------

00:00 59s629ms ACTIVE stay alive Reasons:

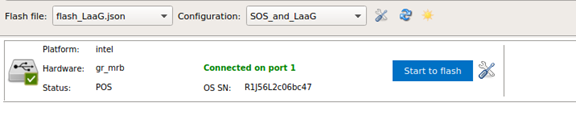
00:00 59s629ms wakeup button (trigger)

00:00 59s629ms UART reset performed

00:00 59s781ms UART reset performed

00:01 19s206ms Temp. (id 0) = 35 C; Temp. (id 1) = 36 C; Supply: 12.00 V atc = 2; FAN 141 rpm; FAN 30 %

Leave this minicom session running; you may use it later to reboot.

1. Ensure the OTG USB cable is connected on the GP board to the host machine.
2. Start Intel® PFT from the desktop of the development computer.
3. Select the target file flash\_LaaG.json from the **Browse** window.
4. Press **Start to flash** to start flashing.  
   The following is an example of the Intel® PFT window flashing LaaG.  
   
5. All the binaries for the configuration (CWP, Linux Service OS, and Linux Guest) have been installed. The system will not reboot automatically, so you must reboot the system by pressing ‘r’, ‘n’, ‘1’, and ‘enter’ on the IOC console.
6. Start a new minicom session on the host computer, this time connected to /dev/ttyUSB3. Ensure the minicom serial port settings are 115200 8N1 and both HW and SW flow control are turned off, and change the Serial Device to /dev/ttyUSB3. This is for the ABL console output:

sudo minicom -s

Here is an example of the log output for the new minicom session:

Welcome to minicom 2.7

OPTIONS: I18n

Compiled on Jan 1 2014, 17:13:19.

Port /dev/ttyUSB3, 14:13:31

Press CTRL-A Z for help on special keys

CWPHV:\>

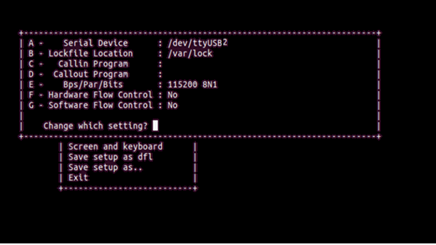
By default, after the system reboots, hypervisor and SOS were booted automatically, but the Linux guest must be started manually.

## Installing CWP/SOS/AaaG

1. Connect a USB cable from the debug board to your Ubuntu host machine, and run the following command to verify that its four USB serial ports are discovered and showing under /dev.

$ ls /dev/ttyUSB\*

/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3

1. Run minicom -s to verify the minicom serial port settings are 115200 8N1 and both HW and SW flow control are turned off. Also, change the Serial Device to /dev/ttyUSB2. An example screen is shown below:  
     
   
2. Leave the session up and open another minicom session, this time connecting to /dev/ttyUSB2 (for the IOC console) using the command shown below:

sudo minicom -s

1. In this IOC console session, press ‘r’ to reboot the system. You may notice that the CPU fan stops rotating momentarily. An example of the IOC console log is shown below:

00:00 00s023ms Module cbc\_logic\_core : Init

00:00 00s024ms Module can\_lifecycle : Init

00:00 00s024ms CAN Lifecycle - using NVM values

00:00 00s024ms Lifecycle: PreRuntime: 1000 ms, require 1 message(s) repetitions to start

00:00 00s024ms CAN ID 0x1A, dlc 4, byte[2]

00:00 00s024ms Signal mask 0x7, bit shift: 0 bits

00:00 00s024ms startup value [min/max] = 0x01/0x07

00:00 00s024ms shutdown value [min/max] = 0x00/0x00

00:00 00s024ms S3 value [min/max] = 0xFF/0xFF

00:00 00s025ms Module can\_gw\_filter : Init

00:00 00s025ms Runtime Measurement calibration offset: 000 ns

00:00 00s025ms User CAN matrix is used: 0

00:00 00s025ms Automatic cold start disabled

00:00 00s025ms Max allowed startup temperature 125 C, max allowed temperature during runtime 125 C

00:00 00s025ms Min allowed startup battery voltage 8.00 V, min allowed battery voltage during runtime 6.00 V

00:00 00s025ms Recovery from S3 on SoC timer event supported

00:00 00s025ms ------------------- Logic Core Initialisation done -------------------

1. When the reboot completes, press ‘n’, then ‘4’ and ‘Enter’ to boot the system into fastboot mode. In this mode, the fastboot command is available to communicate from the host to the target, which lets you erase and flash partition images onto the board’s eMMC flash memory. An example of the IOC console log is shown below:

00:00 59s386ms master\_state\_engine -> wait for CBC - start synced peripherals

00:00 59s386ms cbc\_logic\_core -> startup(1 - early)

00:00 59s387ms CBC kmod versionless startup detected.

00:00 59s387ms master\_state\_engine -> wait for all modules

00:00 59s387ms cbc\_logic\_core -> startup(2 - late)

00:00 59s388ms cbc\_logic\_core -> on state

00:00 59s391ms console\_queue -> startup(1)

00:00 59s391ms timestamp -> startup(1)

00:00 59s396ms console\_queue -> on state

00:00 59s396ms timestamp -> on state

00:00 59s629ms abl\_control\_uart -> on state

00:00 59s629ms Startup Counter: 1

00:00 59s629ms System Start completed! (ABL Blob 1)

00:00 59s629ms master\_state\_engine -> on state

00:00 59s629ms ------------------------------------------------------

00:00 59s629ms ACTIVE stay alive Reasons:

00:00 59s629ms wakeup button (trigger)

00:00 59s629ms UART reset performed

00:00 59s781ms UART reset performed

00:01 19s206ms Temp. (id 0) = 35 C; Temp. (id 1) = 36 C; Supply: 12.00 V atc = 2; FAN 141 rpm; FAN 30 %

Leave this minicom session running; you may use it later to reboot.

1. Ensure the OTG USB cable is connected on the GP board to the host computer.
2. Start Intel PFT from the desktop of the development computer.
3. Select the target file flashAaaG.json from the **Browse** window.
4. Press **Start to flash** to start flashing.  
   The following is an example of the Intel PFT window flashing AaaG:  
   
5. All the binaries for the configuration (CWP, Linux Service OS, and AaaG) have been installed. The system will not reboot automatically, so you must reboot the system by pressing ‘r’, ‘n’, ‘1’, and ‘Enter’ on the IOC console.
6. After rebooting the system, the hypervisor, SOS, and Android are booted automatically.
7. If you have installed all images, then want to update LaaG or AaaG only, you can select LaaG or AaaG in configuration option menu. Also, we provides one option to flash SOS separately.

§

# Boot CWP/Service OS and Start User OS

In this section, you use minicom sessions on the Linux host computer to communicate with the GP MRB board over the USB serial ports on the debug board. You’ll use the two minicom sessions that you used in previous steps. If the sessions are not already open, open them now.

## Boot the Hypervisor

1. After you rebooted the system successfully in the previous steps, the hypervisor and SOS were booted automatically and the ABL console automatically switched to the hypervisor shell output from /dev/ttyUSB3. To confirm that the console switched, press the **Enter** key. You should see the prompts shown below::

CWPHV:\>

CWPHV:\>

1. By default, the hypervisor and Linux Service OS were booted up automatically. If you installed Linux Guest, the Linux Guest OS must be started manually. You can check the status of VMs by running the command shown below:

CWPHV:\>vm\_list

VM NAME VM ID VM STATE

======= ===== ========

vm\_0 0 ON

vm\_1 1 OFF

## Commands Supported by the Hypervisor Shell

You can interact with the hypervisor through the hypervisor shell, using the following commands.

Table . Supported Commands in the Hypervisor Shell

| Command | Command Structure | Description |
| --- | --- | --- |
| help | help | Provides helpful description about all of the available commands in the hypervisor shell. |
| vm\_list | vm\_list | Lists the available VMs and displays the VM name, ID, and running state (“ON” for a running VM and “OFF” otherwise). |
| vm\_boot | vm\_boot <vm-id> | Boots a halted VM and moves it into a running state. |
| vm\_console | vm\_console <vm-id> | Switches the console from the hypervisor shell to the VM’s shell. |

## Starting the Linux Guest OS

1. The Service OS was booted automatically, which is running as VM0. To start the Linux guest, which is VM1, you must log on to VM0’s shell first, using the command shown below:

CWP:\> vm\_console 0

1. After running the hypervisor command, the hypervisor shell is switched to the VM0 shell. An example of the log output is shown below:

See 'systemctl status systemd-modules-load.service' for details.

Starting Apply Kernel Variables...

[ OK ] Started Create Static Device Nodes in /dev.

Starting udev Kernel Device Manager...

[ OK ] Reached target Local File Systems (Pre).

Mounting /var/volatile...

[ OK ] Started Fastboot setup.

[ OK ] Started Apply Kernel Variables.

[ OK ] Mounted /var/volatile.

[ OK ] Started udev Kernel Device Manager.

Starting Load/Save Random Seed...

[ OK ] Reached target Local File Systems.

Starting cbc attach...

Starting Weston...

Starting early\_video application...

[ OK ] Started Load/Save Random Seed.

[ OK ] Started Weston.

...

[ OK ] Started Telephony service.

[FAILED] Failed to start OEM Master Key Injection.

See 'systemctl status oemmasterkey.service' for details.

[DEPEND] Dependency failed for OEM Keystore Key Injection.

[ OK ] Started Login Service.

[ OK ] Started UFIPC daemon for channel IASBus.

[ OK ] Started Connection service.

[ OK ] Started NodeStateManager to provide锟斤拷鈥s, session and shutdown handling.

[ OK ] Reached target Remote File Systems (Pre).

[ OK ] Reached target Remote File Systems.

[ OK ] Reached target Network.

Starting Permit User Sessions...

[ OK ] Reached target Network is Online.

Starting Wait for valid IP using connman...

[ OK ] Started OpenAVB GPTP daemon.

Starting Bluetooth service...

[ OK ] Started Permit User Sessions.

[ OK ] Started Bluetooth service.

Starting Hostname Service...

[ OK ] Started Getty on tty1.

[ OK ] Started Serial Getty on ttyS2.

[ OK ] Started Serial Getty on ttyS0.

[ OK ] Reached target Login Prompts.

Starting WPA supplicant...

[ OK ] Started WPA supplicant.

[ OK ] Started Hostname Service.

[ OK ] Started GLMark2 Benchmark -- build.

Yocto GENIVI Baseline (Poky/meta-ivi) 10.0+snapshot-20171115 gr-mrb-64 ttyS0

gr-mrb-64 login: root

root@:DOM0 ~ $

1. To start the Linux Guest OS, run a script located on the Service OS side:

root@:DOM0 ~ $ cd /home/root

root@:DOM0 ~ $ ./linux.sh

The VM0 console is switched to the VM1 shell automatically. An example of the log output of Linux Guest OS is shown below.

...

Display on second monitor...

[ OK ] Started cbc attach.

[ OK ] Started Permit User Sessions.

[ OK ] Started WPA supplicant.

[ OK ] Started Bluetooth service.

Starting Hostname Service...

[ OK ] Started Serial Getty on ttyS0.

[ OK ] Started Getty on tty1.

[ OK ] Reached target Login Prompts.

[ OK ] Reached target Multi-User System.

[FAILED] Failed to start Hostname Service.

See 'systemctl status systemd-hostnamed.service' for details.

[ OK ] Stopped cbc attach.

Starting cbc attach...

[ OK ] Started cbc attach.

[ OK ] Stopped cbc attach.

Starting cbc attach...

[ OK ] Started cbc attach.

[ OK ] Stopped cbc attach.

Starting cbc attach...

[ OK ] Started cbc attach.

[ OK ] Stopped cbc attach.

[FAILED] Failed to start cbc attach.

See 'systemctl status cbc\_attach.service' for details.

[ OK ] Started GLMark2 Benchmark -- texture.

Yocto GENIVI Baseline (Poky/meta-ivi) 10.0+snapshot-20171115 intel-corei7-64 ttyS0

intel-corei7-64 login: root

root@:DOMU ~ $

1. Press **Ctrl**+**Space** to leave the VM running and return control to the hypervisor shell.

# Login to the System

1. Log into the IOC console using the command shown below:

sudo minicom -s /dev/ttyUSB2

1. Log into the ABL console using the command shown below:

sudo minicom -s /dev/ttyUSB3

After the system boots successfully, the ABL console automatically switched to the hypervisor console.

1. Log into SOS / DOM0 using the command shown below:

CWPHV:\>vm\_console 0

1. Press **Ctrl**+**Space** to switch to the hypervisor console from SOS/DOM0 or UOS/DOMU.
2. Log into Android using the command shown below:

adb shell

After installing AaaG, the Android guest OS boots automatically. You can log in with adb. Ensure the OTG USB cable is connected.

1. Once the console is assigned to the UOS, it cannot be switched back to the SOS console with **Ctrl**+**Space**. You have to use below command to connect to SOS from another terminal ssh root@ipaddress of SOS.

# Build cwp/sos/Linux Guest Source

In addition to pre-built binaries, Intel provides source code, such as the Linux kernel tree, to support the Linux Service OS, the Linux Guest OS, Clearwater Pass hypervisor, and device modules (DM). To build source easily and conveniently, the source release contains some tools and scripts.

## Install Dependent Packages

1. You may need to install additional packages with apt-get before building the source code. In our validation environment with Ubuntu 14.04, we installed the following packages:

sudo apt-get install liblz4-tool device-tree-compiler \

u-boot-tools default-jre

1. To build DM, you must install the SDK tool released by Intel, oecore-x86\_64-corei7-64-toolchain-nodistro.0.sh. This SDK tool is delivered with pre-build CWP binaries:

cd ~/Downloads

chmod a+x oecore-x86\_64-corei7-64-toolchain-nodistro.0.sh

./oecore-x86\_64-corei7-64-toolchain-nodistro.0.sh

Keep the default target directory for SDK installation as /usr/local/oecore-x86\_64.

## Download the Source from the Release Server

Download the file shown below to ~/Downloads:

CWP\_DEC\_SRC\_RELEASE.tar.gz

This source code package contains Clearwater Pass hypervisor source, DM source, SOS kernel source, Linux guest OS kernel source, all the configuration files, scripts, and tools.

## Create the Source Tree Structure

1. Create the following directory:

~/workspace\_cwp

Use this directory for creating source tree structures of hypervisor, DM, SOS kernel, UOS kernel, and binaries tools.

1. Change to the directory you created and extract the source release package:

cd ~/workspace\_cwp

tar xvfz ~/Downloads/CWP\_DEC\_SRC\_RELEASE.tar.gz -C ~/workspace\_cwp

When you extracted the source release package, the folders shown below were generated.

| File Name | Purpose |
| --- | --- |
| ~/workspace\_cwp/bin | for necessary tools, pre-build binaries, configuration, scripts, and so on |
| ~/workspace\_cwp/hv | for placing and building the Intel hypervisor binary image from source |
| ~/workspace\_cwp/out | for temp files output when building source |
| ~/workspace\_cwp/pub | for final binaries output |
| ~/workspace\_cwp/sos | for building dm and sos kernel from source |
| ~/workspace\_cwp/uos | for building uos from source |
| Makefile |  |

## Build Source

1. Build all the components with the makefile provided by Intel, using the following commands:

cd ~/workspace\_cwp

make env

make

After compiling successfully, all the binaries are located at ~/workspace\_cwp/pub.

# Build Android as a Guest Source

In this section, we’ll explain how to set up your development system and how to build Android source. For more information about the requirements, or if you have any issues about the host development environment, go to <https://source.android.com/source/requirements>.

## Installing Packages

### Install the Latest Versions from Canonical\*

Run the following commands on an Ubuntu 64-bit system:

$ sudo apt-get update

$ sudo apt-get install git gnupg flex bison gperf build-essential zip curl \

libc6-dev libncurses5-dev:i386 x11proto-core-dev libx11-dev:i386 \

libreadline6-dev:i386 libgl1-mesa-dev g++-multilib mingw32 tofrodos \

python-markdown libxml2-utils xsltproc zlib1g-dev:i386

$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 \

/usr/lib/i386-linux-gnu/libGL.so

If you encounter an error, you must download and install some packages manually, or change the sources.list to a third source. As we found with Ubuntu 16.04, mingw32 is not in the repository by default, so you must download and install it manually.

$wget <http://archive.ubuntu.com/ubuntu/pool/universe/m/mingw32/mingw32_4.2.1.dfsg-2ubuntu1_i386.deb>

$wget <http://archive.ubuntu.com/ubuntu/pool/universe/m/mingw32-binutils/mingw32-binutils_2.20-0.2ubuntu1_i386.deb>

$wget <http://archive.ubuntu.com/ubuntu/pool/universe/m/mingw32-runtime/mingw32-runtime_3.15.2-0ubuntu1_all.deb>

#install the package

sudo dpkg -i \*.deb

# if it shows dependencies then use command, and again install

$sudo apt-get install -f

$sudo dpkg -i \*.deb

$ sudo apt-get install libssl-dev

### Install Additional Packages

Run the following command:

$ sudo apt-get install squashfs-tools bc ccache tesseract-ocr imagemagick \

gettext python-libxml2 unzip dosfstools mtools dos2unix

### Install an Additional Package for Openssl Dependency

Run the following command:

$ sudo apt-get install libssl-dev

### Install OpenJDK

Download and Install Oracle\* JDK.

Please download and install Java SE Development Kit 8 from: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Steps to install <http://openjdk.java.net/install/>:

1. Create the following directory:

$ sudo apt-get install openjdk-8-jdk

1. If the repository is not found, then proceed with the following steps.

$ sudo –E add-apt-respository ppa:openjdk-r/ppa

sudo apt-get update

sudo apt-get install openjdk-8-jdk

1. If you have more than one java version installed on your system. Run the below command and set the right version of JDK:

$ sudo update-alternatives –-config java

## Create Workspace Directory

On your Linux build computer, create a workspace directory in which to clone the Android sources and build the image:

$ mkdir ~/workspace\_android\_as\_guest

## Get the Latest Version of the Google Repo Tool

$ mkdir ~/bin

$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

$ chmod a+x ~/bin/repo

## Get the Source from our External Release Server (ultramobile-kits.org)

This step assumes that you already have access to the Intel customer release server (ultramobile-kits.org) and that you have configured your system accordingly (especially your SSH keys). Contact your support team if you need assistance with this.

$ cd ~/workspace\_android\_as\_guest

$repo init -u ssh://127.0.1.10/manifest -b oa/gordon\_peak\_cwp/a81/rel -m manifest-full-20171214-2017\_ww50.xml

$ repo sync

## Build the Android as a Guest Stack

Execute the following commands to build the Android stack.

$ source build/envsetup.sh

$ lunch gordon\_peak\_cwp-userdebug

$ make droid dist publish ALLOW\_MISSING\_DEPENDENCIES=true –j $(nproc)

# This will automatically use as many threads as cores on your host system

## Build Output

The Android guest partition image for flashing will be created under:

$~/workspace\_android\_as\_guest/out/target/product/gordon\_peak\_cwp/gordon\_peak\_cwp\_AaaG.img

1. After building is completed, you need to copy the image file over to ~/workspace\_cwp/pub to flash with Intel® PFT.

# Build ROOT File System

This section explains how to build the user space of Linux Service OS and Linux Guest OS, and provides step-by-step instructions for building rootfs source. Intel provides two Yocto\* Project meta layers and recipes structure for customers, so that customers are able to download source code from internet and build the rootfs tarball.

To build the Yocto Project structure, you must set up the Linux development environment on Ubuntu 14.04 or Ubuntu 16.04. The full requirements for building with Yocto are detailed in the Yocto Project Quick Start available here:

<http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.htm>

## Setting up your Host to use the Yocto Project

### Build Host

To build host packages, your host must have the following:

* A minimum of 50 Gbytes of free disk space
* A release of the Yocto Project
* Git 1.8.3.1 or greater
* tar 1.24 or greater
* Python 3.4.0 or greater

### Additional Packages

The following items show the required packages by function given a supported Ubuntu Linux distribution:

**Essential packages**: The following packages are required to build an image on a headless system:

sudo apt-get install gawk wget git-core diffstat unzip texinfo \

gcc-multilib build-essential chrpath socat cpio python python3\

python3-pip python3-pexpect xz-utils debianutils iputils-ping

**Graphics support and Eclipse plug-in extras**: The following packages are recommended if the host system has graphics support or if you are going to use the Eclipse IDE.

sudo apt-get install libsdl1.2-dev xterm

Ensure that the build computer has all the necessary network configuration set up per your IT requirements because the build computer uses HTTP, GIT, and FTP to access external sources.

1. The build process downloads source files from multiple repositories, so a fast network connection is a necessity. Also, the build tools can take advantage of multi-core CPUs to speed up compile times.

## Extracting the DOM0/DOMU Tarball

In the following steps, you’ll use two working directories:

* ~/workspace\_download/src

Where files downloaded from the releaser server are placed.

* ~/workspace\_rootfs

For cloning and building the rootfs of Service OS and User OS from source.

1. Export the RF\_YOCTO variable:

export RF\_YOCTO=~/workspace\_rootfs/cwp\_linux\_rootfs

1. Create a directory ~/workspace\_rootfs/cwp\_linux\_rootfs using the variable:

mkdir -p ${RF\_YOCTO}

1. Unpack the contents of the release:

tar -xvf ~/workspace\_download/src/ROOTFS\_EC1746\_RC6\_FOR\_CWP.tar.gz \ -C ${RF\_YOCTO}

1. Change the directory to the ${RF\_YOCTO} folder:

$ cd ${RF\_YOCTO}

The following directories are created:

* GVTG\_EC1746\_RC6-DOM0 - contains all the recipes required for SOS/DOM0.
* GVTG\_EC1746\_RC6-DOMU - contains all the recipes required for UOS/DOMU

Both of these directories have the following subdirectories:

| Subdirectory | Description |
| --- | --- |
| bin | Contains all the binary deliveries needed |
| meta-ias-audio | Contains the audio component layers |
| meta-ias-boot | Contains TSD boot component layers |
| meta-ias-core | Contains the layer for IAS build tools |
| meta-ias-devtools | Contains TSD devtools component layers |
| meta-ias-earlyapplication | Contains the early application component |
| meta-ias-graphics | Contains the Graphics component layers |
| meta-ias-gr-mrb-bsp | Contains kernel and kernel firmware recipes |
| meta-ias-ivi-adapt | Contains IVI adaptations |
| meta-ias-kc-reference | Contains several component references, such as: e2fsprogs, ias-boot -abldumper, mmc-utils, and others |
| meta-ias-lifecycle | Contains Lifecycle components |
| meta-ias-mediasdk | Contains the Media SDK component layers |
| meta-ias-mediatransport | Contains the AVB streamhandler, example configuration for MRB, igb\_avb kernel module, and ptp daemon recipes |
| meta-ias-network | Contains the network component layers |
| meta-ias-oss | Contains open-source packages that must be modified |
| meta-ias-security | Contains the security component layers |
| meta-ias-setup | Contains ias-setup-shutdown component |
| meta-ias-systembus | Contains system bus components |
| meta-ias-targetutilities | Contains target utilities components |
| meta-ias-vehiclebus | Contains Carrier Board Communications (CBC) |
| meta-ias-virtualization | Contains the virtualization component layers |
| project | Contains image definitions |
| oss-meta-layers.conf | Contains meta-layers commit IDs |

## Setting up the Yocto Build Environment

The following sections describe the steps to set up the repos used to build the rootfs.

1. The commit ID of these meta-layer repositories can be obtained from the file:   
   oss-meta-layers.conf.
2. Change the directory:

cd ${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOMU

1. Clone and checkout the upstream layers versions:

git clone git://git.yoctoproject.org/meta-intel && \

cd meta-intel/ && git checkout \

8c15de8dbaa1414ffd0f69bea7227e7285816f85 && cd ..

git clone git://git.yoctoproject.org/meta-ivi && \

cd meta-ivi && git checkout \

e42777ef74b476ed6702b036818bcffd4fe0d768 && cd ..

git clone git://git.openembedded.org/meta-openembedded && \

cd meta-openembedded && git checkout \

dc5634968b270dde250690609f0015f881db81f2 && cd ..

git clone git://git.yoctoproject.org/meta-oracle-java && \

cd meta-oracle-java && git checkout \

f7c98706f80a2488ad2628e5b0c6d3f9f80c24fd && cd ..

git clone git://git.yoctoproject.org/meta-virtualization \

&& cd meta-virtualization && git checkout \

bd3386b597c70b89386b1b884db43d05963ca69b && cd ..

git clone git://git.yoctoproject.org/poky && \

cd poky && git checkout \

dade0e68c645473d94e1b05020b064df40677e81 && cd ..

git clone git://github.com/meta-qt5/meta-qt5 && \

cd meta-qt5 && git checkout \

9bfcf79fcd824efb9f2a9bd72ecbedfee1315c96 && cd ..

1. Copy these layers to the SOS/DOM0 build environment:

tar cf - meta-intel meta-ivi meta-openembedded \

meta-oracle-java meta-virtualization poky meta-qt5 \

| tar -C ${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOM0 -xf -

## Creating the Yocto Build Directory

1. This step is session-specific and must be set each time you login to your host computer.

**Important**: When creating your build configuration, use the supplied sample local.conf.sample and bblayers.conf.sample files. These files can be found in the following location:

${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOMX/project/conf/

## Build Linux Guest Userland

The following steps are required to build the rootfs of UOS/DOMU.

1. Change the directory:

cd ${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOMU

1. Set up the build environment directory:

export TEMPLATECONF=../project/conf/

source poky/oe-init-build-env

1. Make the following changes to the file:

#build/conf/local.conf

#Set the MACHINE variable

MACHINE ?= "intel-corei7-64"

#Set IAS\_PACKAGE\_PREFIX variable

IAS\_PACKAGE\_PREFIX ?= "file:///<absolute\_path\_to

DOMU>/bin/"

The <absolute\_path\_to\_DOMU> is the full path to the DOMU folder. It can be retrieved using the pwd command from within the current folder.

1. Build the UOS/DOMU rootfs:

$ bitbake ias-kc-pf-image-domu

This step builds the root file system -rootfs ias-kc-pf-image-domu-\*rootfs.tar.gz is located at $build/tmp/deploy/images/intel-corei7-64/.

## Build the Linux Service OS Userland

The following steps are required to build the rootfs of DOM0, also known as the Service OS.

1. Change the directory:

cd ${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOM0

1. Set up the build environment directory:

export TEMPLATECONF=../project/conf/

source poky/oe-init-build-env

1. Make the following changes to the file:

#build/conf/local.conf

#Set the MACHINE variable

MACHINE ?= "gr-mrb-64"

#Set IAS\_PACKAGE\_PREFIX variable

IAS\_PACKAGE\_PREFIX ?= "file:///<absolute\_path\_to

DOM0>/bin/"

1. Build the SOS/DOM0 rootfs:

bitbake ias-kc-pf-image-dom0

This step builds root file system - ias-kc-pf-image-dom0-\*.rootfs.tar.gz under build/tmp/deploy/images/gr-mrb-64/.

1. If you encounter “do fetch” or “do fetch timeout” errors when running bitbake, you may need to limit the number of parallel bitbake threads and set make parallelism. These values are 100% dependent on your host computer specifications. For example, refer to the following lines which can be added to your local.conf at ${RF\_YOCTO}/GVTG\_EC1746\_RC6-DOMX/build/conf/local.conf.

BB\_NUMBER\_THREADS ?= "xx"

PARALLEL\_MAKE ?= "-j xx”

## Generate the Final Deployable Image

After you complete building SOS/DOM0 and UOS/DOMU, you should have both rootfs, namely:

* Linux Guest OS rootfs: ias-kc-pf-image-domu-\*rootfs.tar.gz
* Linux Service OS rootfs: ias-kc-pf-image-dom0-\*rootfs.tar.gz

However, neither of them are deployable directly. They must be converted to partition disk images.

1. Copy both files into Clearwater Pass source build tree /bin/rootfs folder.
2. Run the make command.
3. Generate final images by default.

# Working with the Application Software

## glmark Benchmark

If you are running the configuration of CWP, the Linux Service OS, and the Linux Guest OS, by default, two glmark subtests are started on SOS and Linux Guest after boot.

glmark is a non-vsync bound (free-running), open-source benchmark. With preemption enabled on SOS/DOM0, the GPU can be overwhelmed quickly and the frame rate can drop on the UOS/DOMU side.

1. Information about frames per second can be displayed on the screen. In the GPU Mediator, this option is enabled by default in the /lib/systemd/system/glmark2\_build.service file. The option responsible for FPS display is -annotate (in both Doms).

## Prioritized Rendering

Prioritized rendering, or “preemption”, allows high-priority hardware GPU access to one or multiple workloads. By default, all applications run at normal graphics priority. At normal priority on Linux, the kernel driver defines the running order, and preemption is driven by two SOS/DOM0 kernel command line arguments, explained below:

* i915.enable\_preemption:

Acts as a global switch for the feature. Can have one of two values:

1 – Prioritized rendering is on.

0 – Prioritized rendering is off.

* i915.context\_priority\_mode:

Defines the feature policy. Can have one of two values:

2 - All SOS/DOM0 GPU workloads are prioritized over UOS/DOMU GPU workloads (this is default option).

0 – Prioritization is disabled.

## Hyper\_DMABUF Surface Sharing

hyper\_DMA buf surface sharing is a key feature in this release. But to run validation, you must install several applications and RPM packages and start manually.

The usage is different for Linux as a Guest compared to Android as a Guest. The following sections provide separate instructions for each.

### For Android as a Guest Running on CWP

1. Download the application package from the release server.
2. Extract the release package- HYPERDMA.tar.gz
3. Copy libraries libxenctrl.so.4.6, libxenstore.so.3.0, libxenvchan.so.1.0 to /usr/lib of SOS/DOM0.
4. Copy these applications, vmdisplay-server-aaag and vmdisplay-wayland-aaag to /usr/bin of SOS/DOM0.
5. Run hyper DMA surface sharing:
   1. On SOS/DOM0, run the following command:

vmdisplay-server-aaag 1 <DOMU IP> 2345 &

* 1. In another terminal connected to SOS/DOM0, run the following command:

LIBGL\_DRIVERS\_PATH=/usr/lib/mesadri/ vmdisplay-wayland-aaag 1 test

* 1. On the Android guest UI, open the following APP: **Kitchen Sink > inst cluster > START NAV ACTIVITY**.  
     You will see a picture show in the window on DOM0. This is HyperDMA surface sharing on AaaG.

### For Linux as a Guest Running on CWP

1. Download the application package from the release server.
2. Extract the release package- HYPERDMA.tar.gz.
3. Uninstall the pre-installed vmdisplay on SOS/DOM0 and weston on UOS/DOMU by running the following command:

rpm -e --nodeps weston vmdisplay

1. Install the dedicated six rpm packages that are provided in HYPERDMA.tar.gz. On the SOS/DOM0 side, install only vmdisplay-0.7-r2.corei7\_64.rpm. On UOS/DOMU, install the other five Weston related rpm packages with the following command:

rpm –i vmdisplay #Dom0

rpm –i weston #DomU

1. Modify the configuration file on UOS/DOMU:

#.config/ias.conf

<backend raw\_keyboards='1' use\_nuclear\_flip='1' vm="1" vm\_plugin\_path='/usr/lib/weston/[vm-comm-network.so](http://vm-comm-network.so/)' vm\_plugin\_args='0:5554' >

<output name='HDMI2-0' size='inherit' position='origin' vm="1" />

1. Reboot UOS/DOMU:

sync

reboot

1. Run hyper DMA sharing:
   1. On the UOS/DOMU shell, run surfctrl to check the running graphical apps:

surfctrl

If no apps are running, you can start glmark with the following command:

systemctl start glmark2\_texture

* 1. Make the app shareable with surfctrl --surfname=glmark2 --shareable=1:

surfctrl --surfname=glmark2 --shareable=1

* 1. On the SOS/DOM0 shell, start the vmdisplay server:

vmdisplay-server 1 --net <DOMU IP>:5554" --net "0:5555" &

* 1. On the SOS/DOM0 shell, start vmdisplay surface sharing:

LIBGL\_DRIVERS\_PATH=/usr/lib/mesadri vmdisplay-wayland 1 test

* 1. Then on the SOS/DOM0 side, you will see glmark2 shared from UOS/DOMU.

## Planes Restriction

This feature allows the restriction of the owners of each plane and the numbers of the planes exposed on each Dom, in the kernel command line parameters.

The feature is driven by two kernel command line parameters:

* i915.domain\_plane\_owners
* i915.avail\_planes\_per\_pipe

### The i915.domain\_plane\_owners Parameter

The parameter i915.domain\_plane\_owners is valid for SOS/Dom0 only. It tells SOS/Dom0 which planes are owned by each domain. It contains 12 nibbles, where each nibble represents one plane of a pipe, going from pipe A (first four LSB nibbles) up to pipe C (last four MSB nibbles).

The value of each nibble defines the owner of the corresponding plane represented by that nibble.

Value 0 is for SOS/Dom0. Values from 1 to 0xF represents the ID of the guest (UOS/DOMU) in the order in which it is created.

So if there are three guests, the first guest's ID is 1, the second guest’s is 2, and third guest’s is 3.

For Apollo Lake, which has three pipes:

* Pipes A and B have four planes and so four nibbles are used for each.
* Pipe C has only has three planes so only three nibbles are used. The last MSB nibble is unused.

For example:

i915.domain\_plane\_owner = 0x021001200021

pipeC pipeB pipeA

X210 0120 0021

SOS/DOM0 - planes 3A, 4A, 1B, 4B, 1C

UOS/DOMU-1 - planes 1A, 3B, 2C

UOS/DOMU-2 - planes 2A, 2B, 3C

In runtime, you can check the owner of the planes using the following command on SOS/DOM0:

cat /sys/kernel/gvt/control/plane\_owner\_show

### The i915.avail\_planes\_per\_pipe Parameter

The parameter i915.avail\_planes\_per\_pipe is valid for UOS/DOMU and SOS/DOM0 and specifies the number of planes exposed for use on the respective Dom.

It contains three bytes, where each byte indicates the available planes for one pipe. The LSB represents pipe A planes, the second byte represents pipe B planes, and the MSB represents pipe C planes.

Each bit in a given byte determines if a plane is exposed or not:

* A bit value = 1 means that the plane is exposed for use.
* A bit value = 0 means that the plane is not exposed for use, even if it is owned by the Dom.

For example:

The default value for SOS/DOM0 is i915.avail\_planes\_per\_pipe=0x010102.

This means that SOS/DOM0 exposes plane 1 of pipe C, plane 1 of pipe B, and plane 2 of pipe A.

The default value for UOS/DOMU is i915.avail\_planes\_per\_pipe=0x060E01, where:

* 06 – In binary code is 0110 means that UOS/DOMU exposes plane 2 and 3 on pipe C.
* 0E – In binary code is 1110 means that UOS/DOMU exposes plane 2, 3, and 4 on pipe B.
* 01 – In binary code is 0001 means that UOS/DOMU exposes plane 1 on pipe A.

### Modifying the Kernel Command Line Parameter Values

These flags can be adjusted at build time. They can be changed in the following file for SOS/DOM0:

~/workspace\_cwp/hv/bsp/mrb/target\_support/guests/linux-vm0/bootargs.txt

For UOS/DOMU, flags can be changed in these files:

~/workspace\_cwp/bin/configuration/sos\_rootfs/home/root/linux.sh

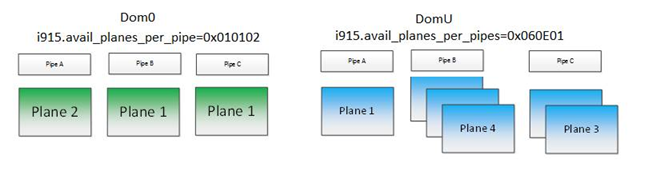
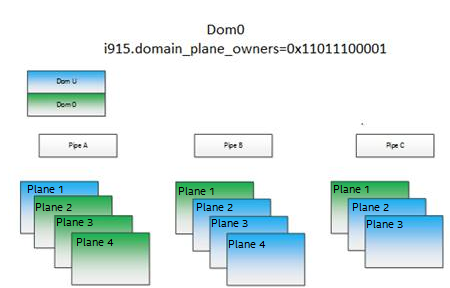
~/workspace\_cwp/bin/configuration/sos\_rootfs/home/root/android.sh

The actual value of these kernel command flags can be checked using the following command on the target:

cat /proc/cmdline

The following figure shows an example of the kernel command parameter values for different plane restriction configurations.

Figure . Example Kernel Command Parameters



### Settings Without eDP Display Support

The following are the current settings without eDP support.

* SOS pipe/plane setting:

i915.avail\_planes\_per\_pipe=0x01010F

i915.domain\_plane\_owners=0x011011110000

* UOS pipe/plane setting:

i915.avail\_planes\_per\_pipe=0x060F00

In the simplest configuration, SOS owns all pipe A planes and UOS owns all pipe B planes, and SOS and UOS display on different HDMI monitors: HDMI-1 for SOS, and HDMI-2 for UOS. There is no plane crossover between SOS and UOS. For this simple configuration, the value of i915.avail\_planes\_per\_pipe can be set to 0x00000F.

However, the current settings shown above use a value of 0x01010F, which makes the first plane of pipe B available to SOS instead of UOS. This enables pipe B to be initialized if the default value of i915.enable\_initial\_modeset=1 is changed.

Likewise, if i915.enable\_initial\_modeset=1, pipe B can be initialized by SOS whether i915.avail\_planes\_per\_pipe equals 0x00000F or 0x01010F.

### Settings with eDP Display Support

eDP support is delivered as supplemental patches. Apply these two patches in eDP\_patches.zip using the following command:

cd ~/workspace\_cwp/hv

git apply –p1 < <path of patches>/0001-hv-enable-eDP-support.patch

cd ~/workspace\_cwp/bin

git apply –p1 < <path of patches>/0001-bin-enable-eDP-support.patch

After applying these two patches, rebuild and flash new sos\_rootfs.img and sos\_boot.img to test eDP.

The SOS pipe/plane default setting is:

i915.avail\_planes\_per\_pipe=0x00000F

i915.domain\_plane\_owners=0x011111110000

The UOS pipe/plane setting is:

i915.avail\_planes\_per\_pipe=0x070F00

This indicates that SOS owns all pipe A planes and displays on the eDP panel, and that UOS owns all pipe B and C planes and displays on HDMI-1 and HDMI-2 in clone mode.

On SOS/DOM0, the usual display names are: HDMI1, HDMI2 and eDP-1.

On UOS/DOMU, the usual display names are: HDMI1, HDMI2 and HDMI3.

**Note**: The customer can get these two patches from the release server:

<https://ultramobile-kits.org/dana/fb/smb/wfb.cgi?t=p&v=resource_1423611128.573830.3%2Coa&si=&ri=&pi=&sb=name&so=asc&dir=gordon_peak_cwp%5Ca81%5C20171214_WW50>

## Direct Display Mode

In the default ias.conf of UOS/DOMU and SOS/DOM0, two displays are enabled in direct display mode. Depending on which displays are connected to the target and the pipe-to-display connection, this can be two HDMI displays or one HDMI display and one eDP display.

To get the name of the connectors to use in the ias.conf file and the pipe-to-display connection, always refer to the debug entry i915\_display\_info for each Dom:

cat /sys/kernel/debug/dri/0/i915\_display\_info

On SOS/DOM0, the usual display names are: HDMI1, HDMI2 and eDP-1.

On UOS/DOMU, the usual display names are: HDMI1, HDMI2 and HDMI3.

The following steps describe how to enable a third monitor in direct display mode:

* + 1. Connect two HDMI monitors and a third eDP monitor.
    2. Enable eDP display support by following the steps defined in [Sec 10.4.2](#_Settings_with_eDP).
    3. Ensure the appropriate plane ownership and exposure in the kernel command line. Refer to [Sec 10.4.2](#_Settings_with_eDP).
    4. Update the /home/root/.config/ias.conf file of UOS/ DOMU to enable planes on the three displays.

The following is an example of how to enable a third display on UOS/DOMU for planes restriction configuration and considering the usual connector names, referred to earlier in [Sec 10.4.2](#_Settings_with_eDP):

On UOS, add the following lines in the ias.conf file:

<crtc name='HDMI3-0' model='classic' mode='preferred'>

<output name='HDMI3-0' size='inherit' position='rightof' target='HDMI2-0' />  
</crtc>

5. Reboot the target and wait for both the VMs (SOS/DOM0 and UOS/DOMU) to boot and start Weston and the glmark2 benches.

6. By default, the glmark2 bench surface is displayed on the plane declared as positioned at the “origin” in the ias.conf configuration.

7. To place the display on a different plane, use the surfctrl command and specify the new coordinates of the surface in respect to the plane position where the surface is to be placed.

For example:

surfctrl --surfname="glmark2" --pos=1920x0

## Multi-Planes

The CWP GVT-g software allows up to four planes on pipe A and pipe B and three planes on pipe C. The default ias.conf of DOM0 defines one classic plane on HDMI1 and one flexible plane on HDMI2.

The default ias.conf of UOS/DOMU defines one classic plane on HDMI1 and one classic plane on HDMI2.

By default, the display is placed on the first plane of HDMI1 for DOM0 (and the first plane of HDMI2 for UOS).

1. If three displays are connected to MRB, the UOS HDMI2 could refer to the physical HDMI1 on the MRB, which will result on UOS and SOS outputs to overlap on the same HDMI1 connected display.

To move the display of a given Dom between different planes of a given display pipe:

* + 1. Grant the plane ownership to the concerned Dom by changing kernel parameters.
    2. Have the ias.conf customized to define a multiple flexible planes model for the given display pipe.  
         
       For example:  
       In the /home/root/.config/ias.conf file, change the model classic to flexible and then you can define multiples planes per pipe:

<crtc name='HDMI2' model=**'flexible'** mode='preferred'>

<output name='HDMI2-0' size='inherit' position='rightof' target='HDMI1-0' />

<output name='left' size='inherit' position='rightof' target='HDMI2-0' plane\_size='900x800' plane\_position='50,50'/>

<output name='right' size='inherit' position='rightof'

target='left' plane\_size='900x800' plane\_position='1000,50'/>

</crtc>

* + 1. Place the Dom display surface on that plane. This is achieved from the concerned Dom prompt using the surfctrl command.  
         
       For example :

surfctrl --surfname="glmark2" --pos=3860x0