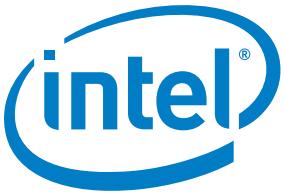


Xen* GVT-g on Gordon Ridge Modular Reference Board (MRB)

Getting Started Guide

December 2017

Intel Confidential

Document Number: 569407-2.

Contents

Revision History

Revision Date	Revision	Description
December 8, 2017	2.2a	Updated for Xen_GVTG_EC1746_RC6_HF1 release
November 16, 2017	2.2	Updated for Xen_GVTG_EC1746_RC6 release
October 13, 2017	2.0	Updated for Xen_GVTG_EC1739_RC1_HF2 release
October 9, 2017	1.9	Updated for Xen_GVTG_EC1739_RC1_HF1 release
September 25, 2017	1.8	Updated for Xen_GVTG_EC1739_RC1 release
September 8,2017	1.7	Updated for GP_Xen_GVTG_ALPHA_WW29.5_RC2-HF2 release
August 8,2017	1.6	Updated for GP_Xen_GVTG_ALPHA_WW29.5_RC2-HF1 release
July 20,2017	1.5	Updated for GP_Xen_GVTG_ALPHA_WW29.5_RC2 release
June 7,2017	1.3	Updated for PRE-ALPHA WW22.5 HF1 release
June 6,2017	1.2	Updated for PRE-ALPHA RC3 WW22.5 release
April 24,2017	1.1	Updated for EC6 RC2 WW17.5 release
February 22,2017	1.0	Updated for POC-5 RC2 WW08.5 release
February 3, 2017	0.9	Updated for POC-5 RC1 WW04.5 release
November 11, 2016	0.8	Updated for POC-4 WW46.5 release
October 12, 2016	0.7	Updated for POC-3 WW42.5 release
September 30, 2016	0.6	Updated for POC-2 WW40.5 release
September 12, 2016	0.5	Initial release POC-1

1.0 Introduction

This document contains detailed instructions for building and installing the Xen GVT-g software release with a Yocto* Project Linux with the Gordon Ridge Board Support Package (BSP).

The topics covered in this document are as follows:

- Build environment setup
- Building Yocto* Linux
- Programming the I/O Controller (IOC)
- Programming the Serial Peripheral Interface (SPI)
- Installing and using the release on the Modular Reference Board (MRB)

1.1 Terminology

Table 1. Terminology

Term	Description
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
eDP	Embedded DisplayPort
ELK	Emergency Linux Kernel
eMMC*	Embedded Multi-Media Controller
FBL	Firmware BootLoader
FPS	Frames Per Second
GGTT	Global Graphic Translation Table
GP	Gordon Peak
GR	Gordon Ridge
GuC	Graphic Micro-Controller
GVT	Graphics Virtualization Technology
HuC	HEVC Micro-Controller
IFWI	Integrated FirmWare Image
IOC	I/O Controller
IOTG	Internet of Things Group

Term	Description
LSB	Least Significant Bit
MRB	Modular Reference Board
MSB	Most Significant Bit
PDK	Platform Development Kit
PLK	Production Linux Kernel
PMIC	Power Management Integrated Circuit
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
VBT	Video Bios Table
VCP	Virtual COM Port
VT	Virtualization Technology

2.0 Yocto BSP Build

This chapter describes the host setup and the steps needed to build the Xen GVT-g software image, the SPI image and the steps to generate and install sdk tools, using Yocto*.

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

2.1 Build Machine Setup

The host Linux PC requires a minimum of 100 GB of free storage space to build the bootable image. The full requirements are detailed in the *Yocto Project Quick Start* available here:

http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

This document describes the setup using a Fedora 21 or higher, 64-bit distribution.

The essential and graphical packages needed for the Fedora distribution can be installed on the host using the command:

```
$ sudo yum install gawk make wget tar bzip2 gzip python \
unzip perl patch diffutils diffstat git cpp gcc gcc-c++ \
glibc-devel texinfo chrpath ccache perl-Data-Dumper \
perl-Text-ParseWords perl-Thread-Queue socat findutils \
which SDL-devel xterm bitbake python-pyasn1-modules \
graphviz glibc.i686
```

Install all the development tools and libraries on the host PC prior to compiling the image. These are also detailed in the Quick Start Guide at the link provided above.

Note: If you intend to build Audio in the Yocto image, ensure that the `pulseaudio` and `pulseaudio-libs-devel` packages have been installed on the host, using:

```
$ sudo yum install pulseaudio pulseaudio-libs \
pulseaudio-libs-devel
```

Note: If you are using a Fedora version higher than 22, downgrade the `pulseaudio-libs` packages to the Fedora 22 version, using :

```
$ sudo yum --releasever=22 --allowerasing downgrade pulseaudio-
libs{,-{devel,glib2}}
```

Note: For Fedora 23, install the python rsa module, using :

```
$ sudo yum install python-pip
$ sudo pip install --proxy=<Your-Proxy:Port> rsa
```

Note: 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.

2.2 Extracting BSP Releases

This section describes the extraction of the GP BSP release on the host:

1. Export the environment variables as follows:

```
$ export REL_VER=GVTG_EC1746_RC6_HF1
$ export GR_YOCTO=/home/${USER}/Xen_${REL_VER}/
```

2. Create a `work` and a `release` directory:

```
$ mkdir -p /home/${USER}/Xen_${REL_VER}
$ mkdir -p /home/${USER}/release_${REL_VER}
```

3. Unpack the release content under the release directory:

```
$ cd /home/${USER}/release_${REL_VER}
$ tar -xvf Xen_${REL_VER}.tar.gz --strip-components=1
```

   A directory called `platform` is created that includes:

   - DomU and Dom0 tarballs – to build DomU and Dom0 images.
   - `ABL` folder – contains ABL binaries and release changlog file.
   - `pdk` folder - contains a collection of tools, created to help customize the platform delivery and also to measure its performance.
   - `ifwi` folder - contains the IFWI tarball to build an SPI image with the ELK.
   - `IOC` folder - contains IOC pre-built firmware for different MRB revisions.
   - `collaterals` folder – contains the list of release components.
   - `Weston` tarball – contains source code for Weston

4. Change the directory to the `platform/` folder :

```
$ cd /home/${USER}/release_${REL_VER}/platform
```

5. Extract `Dom0` and `DomU` tarballs into the `work` directory:

```
$ tar -xf Xen_${REL_VER}-DOM0.tgz -C $GR_YOCTO/
$ tar -xf Xen_${REL_VER}-DOMU.tgz -C $GR_YOCTO/
```

   The following folders are created in the `$GR_YOCTO` directory:

   - `Xen_${REL_VER}-DOM0` - contains all the recipes required for Dom0.
   - `Xen_${REL_VER}-DOMU` - contains all the recipes required for DomU.

Both of these directories have the following sub-directories:

- `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 the 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, for example: e2fsprogs, ias-boot-abldumper, mmc-utils.
- `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.

2.3 Setting the Yocto Environment

The following sections describe the steps needed to set up the Yocto repos used to build the Xen GVT-g software.

Note: The commit IDs of these meta-layer repositories can be obtained from the file: `$GR_YOCTO/Xen_${REL_VER}-DOMU/oss-meta-layers.conf`

1. Change the directory using:

```
$ cd $GR_YOCTO/Xen_${REL_VER}-DOMU
```

2. Clone and check out 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 ..
```

3. Copy these layers to Dom0 build environment:

```
$ tar cf - meta-intel meta-ivi meta-openembedded \
meta-oracle-java meta-virtualization poky meta-qt5 \
| tar -C $GR_YOCTO/Xen_${REL_VER}-DOM0 -xf -
```

2.4 Build DomU Image

The following steps are required to build the image for the DomU.

1. Change the directory to `$GR_YOCTO/Xen_${REL_VER}-DOMU` using:

```
$ cd $GR_YOCTO/Xen_${REL_VER}-DOMU
```

2. Set up the build environment directory:

```
$ export TEMPLATECONF=../project/conf/
$ source poky/oe-init-build-env
```

3. Make the following changes to the file `conf/local.conf`

   – Set the `MACHINE` variable
   `MACHINE ?= "intel-corei7-64"`

   – Set the `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 the current folder.

4. Build the DomU image:

```
$ bitbake ias-kc-pf-image-domu
```

   After a successful build, the DomU rootfs tarball `ias-kc-pf-image-domu-intel-corei7-64-*rootfs.tar.gz` is located under:

```
$GR_YOCTO/Xen_${REL_VER}-DOMU/build/tmp/deploy/images/intel-corei7-64/
```

Note: 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 machine specifications. For example, refer to the following lines that can be added to your local.conf at `$GR_YOCTO/build/conf/local.conf`.

```
BB_NUMBER_THREADS ?= "xx"
PARALLEL_MAKE ?= "-j xx"
```

2.5 Build Dom0 Image

The following steps are needed to create the Dom0 rootfs and the iasImage with Xen and the Dom0 kernel.

Important: The command line arguments for Xen and the Dom0 kernel are available in the files `xen_cmdline.txt` and `dom0_cmdline.txt`. These files can be found at the following location:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-virtualization/recipes-extended/xen/xen-4.6.0/
```

In our ABL multiboot tests with the Xen Hypervisor, we observed that the first command line parameter from both command line files is removed. Therefore, we added a dummy parameter first - `removedanyhow=1`.

Important:The command line arguments for the DomU kernel is availabe in the `xenguest.cfg` file. This file can be found under :

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-
virtualization/recipes-extended/xenguest-config/xenguest-
config/
```

Important: eDP detection for Dom0 is disabled by default. To enable eDP detection, delete the `video=eDP-1:d` argument in the file:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias
virtualization/recipes-extended/xen/xen-4.6.0/
dom0_cmdline.txt
```

1. Change the directory to `$GR_YOCTO/Xen_${REL_VER}-DOM0` using:

   ```
   $ cd $GR_YOCTO/Xen_${REL_VER}-DOM0
   ```

2. Set up the build environment directory:

   ```
   $ export TEMPLATECONF=../project/conf/
   $ source poky/oe-init-build-env
   ```

3. Make the following changes to the file `conf/local.conf`

   – Ensure that the `MACHINE` variable is set correctly:
   `MACHINE ?= "gr-mrb-64"`

   – Set the `IAS_PACKAGE_PREFIX` variable
   `IAS_PACKAGE_PREFIX ?= "file:///<absolute_path_to-Dom0>/bin/"`

   The `<absolute_path_to_Dom0>` is the full path to the Dom0 folder. It can be retrieved using the `pwd` command from the current folder.

4. Build the Xen and Dom0 image:

   ```
   $ bitbake xen
   $ bitbake ias-kc-pf-image-dom0
   ```

   After a successful build, the Dom0 rootfs tarball `ias-kc-pf-image-dom0-gr-mrb-64-*.tar.gz` and the iasImage (Xen and the Dom0 kernel) `xen_iasImage-gr-mrb-64` are located under:

   ```
   $GR_YOCTO/Xen_${REL_VER}-DOM0/build/tmp/deploy/images/gr-mrb-64/
   ```

Note: 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 machine specifications. For example, refer to the following lines that can be added to your `local.conf` at `$GR_YOCTO/Xen_${REL_VER}-DOMX/build/conf/local.conf`:

```
BB_NUMBER_THREADS ?= "xx"
PARALLEL_MAKE ?= "-j xx"
```

2.6 Generate and Install the SDK Tools

SDK tools refers to the standard collection of Yocto* project SDK tools plus some specific Intel targets tools. The following are the steps to generate and install the tools:

1. Change the directory to `$GR_YOCTO/Xen_${REL_VER}-DOM0`:

   ```
   $ cd $GR_YOCTO/Xen_${REL_VER}-DOM0
   ```

2. Set up the build environment directory:

   ```
   $ export TEMPLATECONF=../project/conf/
   $ source poky/oe-init-build-env
   ```

3. Make the following changes to the `conf/local.conf` file:

   – Ensure that the `MACHINE` variable is set correctly:
   `MACHINE ?= "gr-mrb-64"`

   – Set the `IAS_PACKAGE_PREFIX` variable
   `IAS_PACKAGE_PREFIX ?= "file:///<absolute_path_to-Dom0>/bin/"`

   The `<absolute_path_to_Dom0>` is the full path to the Dom0 folder. It can be retrieved using the `pwd` command from the current folder.

4. Build the sdk tools using:

   ```
   $ bitbake ias-kc-pf-image-dom0 -c populate_sdk
   ```

   After a successful build , the SDK installer shell script file (.sh) is found under:

   ```
   $ GR_YOCTO/Xen_${REL_VER}-DOM0/build/tmp/deploy/sdk
   ```

5. Create the sdk directory directory `$GR_YOCTO/sdk` and install the SDK tools:

   ```
   $ mkdir -p $GR_YOCTO/sdk

   $ sh $GR_YOCTO/Xen_${REL_VER}-
   DOM0/build/tmp/deploy/sdk/oecore-x86_64-corei7-64-toolchain-
   nodistro.0.sh -d $GR_YOCTO/sdk
   ```

The SDK tool files are found under:

```
$GR_YOCTO/sdk/sysroots/x86_64-oesdk-linux/
```

An example of an Intel tool is `ias_image_app` (see Appendix A for usage).

2.7 Build the SPI Image

This section describes the steps required to build the SPI image for the MRB . The SPI image contains the ABL, with stitched ELK, kernel command line, and extra data .

1. Change the directory to the `ifwi` folder in the release location :

   ```
   $ cd ${GR_YOCTO}/platform/ifwi
   ```

2. Extract `Xen_${REL_VER}-ifwi.tgz` using:

   ```
   $ tar -xf Xen_${REL_VER}-ifwi.tgz -C $GR_YOCTO/
   ```

   The following directory is created:

   ```
   $GR_YOCTO/Xen_${REL_VER}-ifwi
   ```

   It contains the following sub-directories:

   - `bin` - contains all the binary deliveries needed.
   - `meta-ias-core` - contains the layer for IAS build tools.
   - `meta-ias-devtools` - contains TSD devtools component layers.
   - `meta-ias-gr-mrb-bsp` – contains kernel and kernel firmware recipes.
   - `meta-ias-ifwi` – contains IFWI receipes.
   - `meta-ias-ivi-adapt` - contains IVI adaptations.
   - `meta-ias-oss` - contains open-source packages that must be modified.
   - `meta-ias-security`- contains the security component layers.
   - `meta-ias-setup` – contains the ias-setup-shutdown component.
   - `meta-ias-vehiclebus` - contains Carrier Board Communications (CBC).
   - `project` - contains image definitions.
   - `oss-meta-layers.conf` – contains meta-layers commit IDs.

3. Copy the required upstream layers from DomU to the IFWI build environment:

   ```
   $ cd $GR_YOCTO/Xen_${REL_VER}-DOMU
   $ tar cf - meta-intel meta-ivi meta-openembedded \
    poky | tar -C $GR_YOCTO/Xen_${REL_VER}-ifwi -xf -
   ```

   Important: The list of required layers are available under `$GR_YOCTO/Xen_${REL_VER}-ifwi/oss-meta-layers.conf`

4. Change the directory to `$GR_YOCTO/Xen_${REL_VER}-ifwi`:

```
$ cd $GR_YOCTO/Xen_${REL_VER}-ifwi
```

5. Set up the build environment directory:

```
$ export TEMPLATECONF=../project/conf-ifwi/
$ source poky/oe-init-build-env
```

6. Make the following changes in the `conf/local.conf` file:

   – Ensure that the `MACHINE` variable is set correctly:
   `MACHINE ?= "gr-mrb-64"`

   – Set the `IAS_PACKAGE_PREFIX` variable:
   `IAS_PACKAGE_PREFIX ?= "file:///<absolute_path_to_ifwi>/bin/"`

   The `<absolute_path_to_ifwi>` is the full path to the IFWI folder. It can be retrieved using the `pwd` command from the current folder.

7. Build the the SPI image using:

```
$ bitbake ifwi
```

The SPI image `ifwi-apl-spi-(version).bin` is created under:

```
$GR_YOCTO/Xen_${REL_VER}-ifwi/build/tmp/deploy/images/gr-mrb-64/ifwi-apl-*/
```

The SPI image includes the ABL with VT enabled , the stitched ELK and the kernel command line.

The stitching is achieved during the build procedure. Details are in the following receipe:

```
$GR_YOCTO/Xen_${REL_VER}-ifwi/meta-ias-ifwi/recipes-bsp/ifwi/ifwi.bb
```

The receipe invokes a stitching python script "`IFWIStitch.py`", from the following package :

```
$GR_YOCTO/Xen_${REL_VER}_ifwi/bin/ifwi-toolchain-native/ifwi-toolchain-native-3.1.50.2227a-r0.tar.gz
```

The VT configuration is enabled using the `smip_iafw.bin` binary file fed to the stitching command through the –`ablcfg` option.

Note: The SPI image needs to be updated only if there is a new ABL for the concerned release. Check the Release Notes document for the ABL version.

3.0 Updating Firmware

This chapter describes how to program the SPI and IOC using the tools provided in the pdk and sdk components of the release package.

PDK tools are available under the platform `/pdk/host` folder.

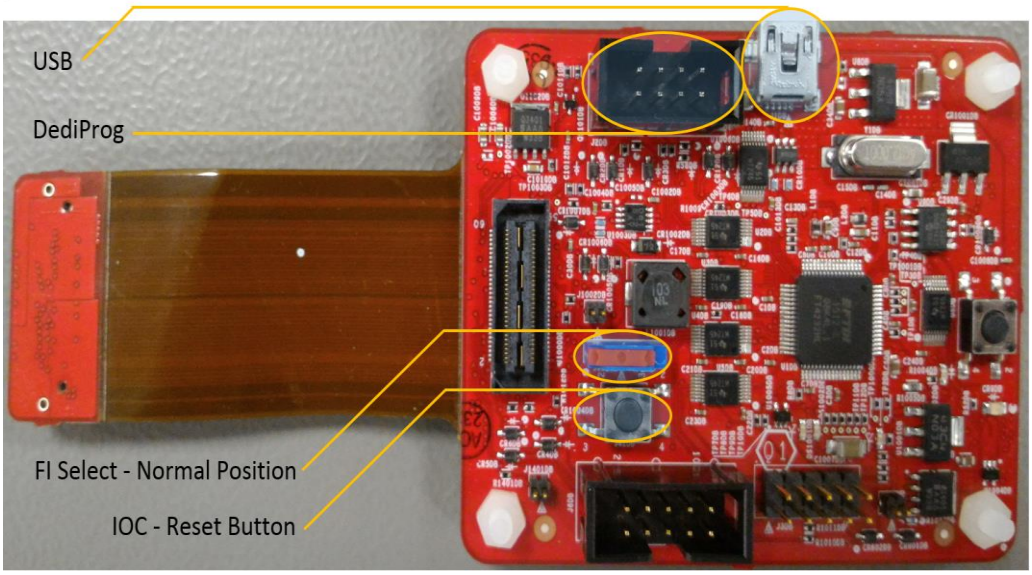
SDK tools are installed on the host machine by following the steps in Section 2.6.

Note that the instructions in this chapter needs to be followed only if there are changes to the IOC firmware. Refer to the Release Notes for any such changes.

3.1 Debug Adapter

Figure 1 shows a top view of the Debug Adapter.

Figure 1. Debug Adapter Top View



If you connect the Debug Adapter, you can see four USB serial ports.

- The 3rd USB serial port is the IOC serial console.
- The 4th USB serial port is the BXT serial console.
- Use a 115200 baud rate

Microsoft Windows* Drivers: Use the Virtual COM Port (VCP) driver directly from the manufacturer.

3.2 Programming the IOC Firmware

This section describes how to update the IOC firmware using a Microsoft Windows* host.

3.2.1 Prerequisites

1. Copy the release package to a Microsoft Windows* host.
2. The new firmware image is available in: `<release_package>/platform/IOC/ioc_firmware_gp_mrb_fab_x.ias_ioc`

   To choose which fab, look at the TA# number on the metal chassis: the last three numbers (after the '-' sign) are important:

-20x	`ioc_firmware_gp_mrb_fab_b.ias_ioc`
-30x	`ioc_firmware_gp_mrb_fab_c.ias_ioc`
-40x	`ioc_firmware_gp_mrb_fab_d.ias_ioc`
-50x	`ioc_firmware_gp_mrb_fab_e.ias_ioc`

Note: Only firmware images that have been explicitly built for use with the IOC flasher can be used with the IOC flasher. These firmware images have a file name ending with `.ias_ioc`.

3.2.2 Steps

1. Connect the Debug Adapter to the IOC debug port and to USB.
2. Connect the Debug Adaptor to the host using the USB cable supplied.
3. Power off the target.
4. Disconnect all power.
5. Apply power.
6. On the Microsoft Windows* host, open a command prompt window.
7. Navigate to the Microsoft Windows* folder of the IOC Flash Server Windows Package in the release package (`<release_package>/platform/pdk/host/windows/iocflashserver-win/`), unzip the ioc flash tool and execute:

   ```
   ioc-flash-server-x86_64.exe –s COMX –t
   <release_package>/platform/IOC/ioc_firmware_gp_mrb_fab_x.ias_ioc
   ```

Note: Replace “x” with the actual COM port number that is needed. By connecting the Debug Adapter to your host, four USB ports are added to the existing *n* USB ports on the host in the device manager. Use the third port from the added USB ports to update the IOC firmware.

8. Wait for IOC flashing to finish.

9. On success, the output should look like the following:

```
Trying to open port: COMX
   Restarting the IOC into the bootloader (terminal request)
   Waiting for IOC bootloader
   Checking for unexpected data on serial port / press reset
   button on debug adapter, if this message is seen for more
   than 2s ... done
   Bootloader request detected.
   Waiting for IOC information.

   Hardware revision: GR FAB C
   Flash boot loader major-minor: 3-3
   Starting to flash firmware:
   ioc_firmware_gp_mrb_fab_c.ias_ioc
   Erasing flash.
   Data frame number send: xxxx|xxxx

   Flashing was successful.
   Restarting the IOC
```

3.3 Programming the IOC Bootloader (FBL)

This section describes how to update the IOC bootloader using a Microsoft Windows* host and the FR flash tools.

Note: If the current IOC firmware flashed to the target is of build ID 13 or higher, you can skip this section as it will detect an outdated fbl on startup and update it automatically.

3.3.1 Prerequisites

1. Copy the release package to a Microsoft Windows host.

2. FLASH MCU Programmer (pcwfr.zip) available for download from: http://www.spansion.com/support/microcontrollers/developmentenvironment/Pages/mcu-download.aspx

3. The new IOC bootloader image iavailable in:
   `<release_package>/platform/IOC/ioc_production_image_gp_mrb_fab_x.mhx`

   To choose which fab, look at the TA# number on the metal chassis.

The last three numbers (after the '-' sign) are important:

-20x	`ioc_production_image_gp_mrb_fab_a_b.mhx`
-30x	`ioc_production_image_gp_mrb_fab_c.mhx`
-40x	`ioc_production_image_gp_mrb_fab_d.mhx`
-50x	`ioc_production_image_gp_mrb_fab_e.mhx`

3.3.2 Steps

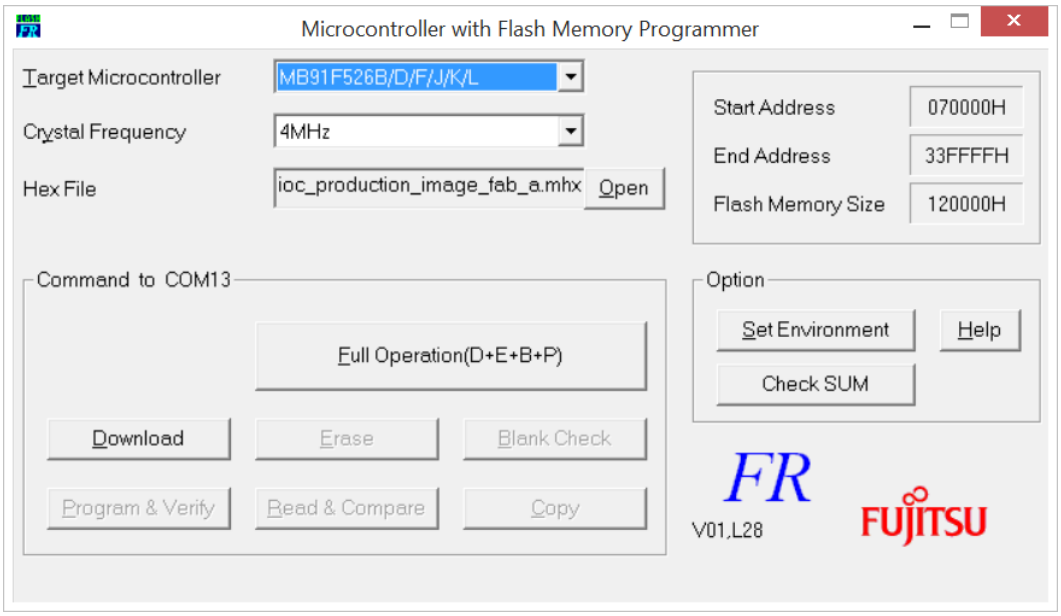
1. Connect the Debug Adapter to the IOC debug port and to USB.
2. Connect the Debug Adaptor to the host using the USB cable supplied.
3. Power off the target.
4. Disconnect all power.
5. Move the **FI Select** switch on the Debug Adapter, away from the ribbon cable (the opposite to the “Normal” position – See Figure 1).
6. Apply power.
7. Start the installed Fujitsu* flasher on the Microsoft Windows machine (see Figure 2 following).
8. Select **Target Microcontroller** "MB91F526B/D/F/J/K/L”.
9. Set **Crystal Frequency** to "4MHz".
10. Use the **Set Environment** button to select the IOC COM port of the Debug Adapter (3rd USB serial port – check the device manager).
11. Select the **Hex File** to program (.mhx file).
12. Click on the **Full operation (D+E+B+P)** button.
13. Press the (IOC) reset button on the Debug Adapter when prompted.
14. Acknowledge the request to press reset on the microcontroller on the user board with "ok".

15. Flashing starts. Wait for an end message. If you get an error message, for example, “timeout error number 003”, disconnect the power supply and start again from step 3. You might also want to disconnect/reconnect the USB cable of the Debug Adapter.

16. Move the **FI Select** switch back to original (“Normal”) position.

17. Press the reset button on Debug Adapter to start the flash boot loader.

Figure 2. Fujitsu* Flash Memory Programmer Interface



3.4 Programming the SPI

This section describes the steps needed to update the SPI with a new image.

3.4.1 Prerequisites

- The SDK tools are installed as per Section 2.6.
- The SPI image is successfully built as per Section 2.7.

3.4.2 Steps

1. Connect the Debug Adapter to the IOC debug port and to USB.

2. Connect the Debug Adaptor to the host using the USB cable supplied.

3. Ensure that the MRB does not have power applied to it.

4. Change the directory to `$GR_YOCTO/sdk/sysroots/x86_64-oesdk-linux/usr/bin/`:
   `$ cd $GR_YOCTO/sdk/sysroots/x86_64-oesdk-linux/usr/bin/`

5. Press and hold the “ignition” button, then apply power to the MRB while holding the “ignition” button.

6. Release the “ignition” button and erase the SPI by executing:

   ```
   $ sudo ./ias-spi-programmer --erase
   ```

   The output of erase operation is similar to the following:

   ```
   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)

   Start full chip erase: ..............................
   Execution time: xx sec.
   ```

7. Flash the new SPI image by executing:

   `$ sudo ias-spi-programmer --write ifwi-apl-spi-3.0.20.1182d+1735-r0-(date_of_build).bin-A.bin` (for the A stepping)
   or
   `$ sudo ./ias-spi-programmer --write ifwi-apl-spi-3.0.20.1182d+1735-r0-(date_of_build).bin-B.bin` (for B, C or D steppings)

   Following a successful flash operation, the output is similar to the following:

   ```
   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: xxxxx
   Start write verification
   Write verification completed successfully
   Execution time: xxx sec.
   ```

8. After a successful flash operation, disconnect the power and restart the target.

4.0 Booting Xen GVT-g Software Image

This section describes the steps required to flash the Xen GVT-g software image generated in Section 2.5 on the MRB.

4.1 Booting with ELK

Assumption : The SPI is flashed as per the instructions in Section 3.4.

Power cycle the target, then press and hold the Ignition button for more than 20 seconds and release it to boot the target into the ELK.

When the MRB boots in ELK mode, the functionality is minimal.

4.2 Programming the Xen GVT-g Software Image on the MRB

This section describes the steps required to flash the Xen GVT-g software image on eMMC.

4.2.1 Prerequisites

- The MRB boots successfully with the SPI image as in Section 4.1.
- A successful execution of the build process as in Section 2.0.
- The resulting iasImage, Dom0 and DomU rootfs are copied to a USB stick.

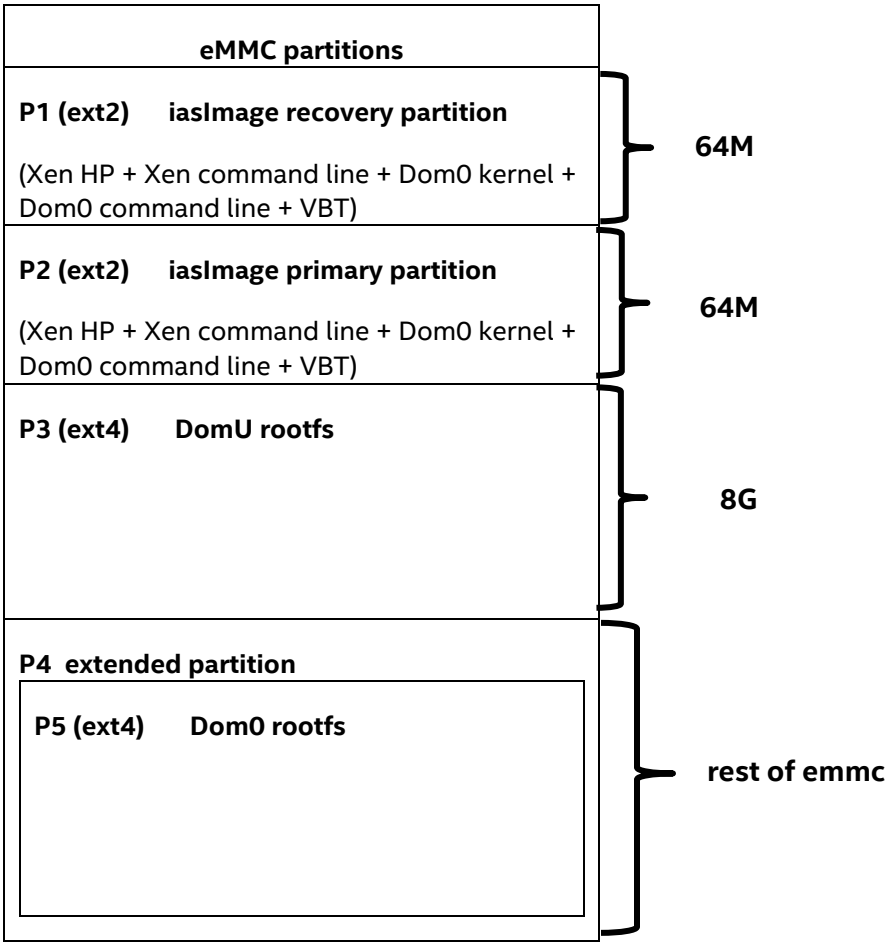
4.2.2 Preparing the eMMC to Load the Kernel Image from a User Partition

The ABL expects:

- The PLK (iasImage) to be in `mmcblk0p1` or `mmcblk0p2`
- The `rootfs` to be in any storage location. For the Xen GVT-g software, the following have been selected:
  - `mmcblk0p5` as the storage location for Dom0 rootfs
  - `mmcblk0p3` as the storage location for DomU rootfs

The following figure shows the eMMC partitions layout on the MRB.

Figure 3. eMMC Layout for Xen GVT-g Software Release



The following are the steps required to create these partitions, from the ELK target prompt:

1. Ensure that the eMMC is blank by writing zeros to it using `dd`:

   ```
   $ dd if=/dev/zero of=/dev/mmcblk0 bs=4096 count=1
   ```

2. Create partitions using `fdisk`:

   ```
   $ fdisk /dev/mmcblk0
   ```

   - Enter "`n`" to create a new partition
   - Enter "`p`" to select primary partition
   - Enter "`1`" to create partition 1
   - Enter to select the start of the first cylinder
   - Enter "`+64M`" to select the sizel

- Enter "n" to create a new partition
- Enter "p" to select primary partition
- Enter "2" to create partition 2
- Enter to select the start of the first cylinder
- Enter "+64M" to select the size

- Enter "n" to create a new partition
- Enter "p" to select primary partition
- Enter "3" to create partition 3
- Enter to select the start of the first cylinder
- Enter "+8GB" to select the size

- Enter "n" to create next partition
- Enter "e" to create Extended partition
- Partition 4 will be created
- Enter to select default start cylinder
- Enter to select last cylinder

- Enter "n" to create next partition
- Logical partition 5 will be created
- Enter to select default start cylinder
- Enter to select last cylinder

- Enter "w" to store the partition table

3. When `fdisk` exits, the following partitions are created :

   `mmcblk0p1, mmcblk0p2, mmcblk0p3, mmcblk0p4` and `mmcblk0p5`.

4. Format `mmcblk0p1` and `mmcblk0p2` as an `ext2` file system using:

```
$ mke2fs -b 4096 /dev/mmcblk0p1
$ mke2fs -b 4096 /dev/mmcblk0p2
```

5. Format `mmcblk0p3 & mmcblk0p5` as an `ext4` file system using:

```
$ mkfs.ext4 /dev/mmcblk0p3
$ mkfs.ext4 /dev/mmcblk0p5
```

4.2.3 Flashing the Boot Image and rootfs on the eMMC

The following steps are needed to update the `iasImage` , `DomU and Dom0 rootfs` on the target.

1. Mount the USB stick containing the iasImage , DomU and Dom0 rootfs on a mounting point on the target

```
$ mkdir -p usb
$ mount /dev/sda1 usb
$ cd usb/<iasImage_and_rootfs_location>/
```

2. Copy `iasImage` to `/dev/mmcblk0p1` and `/dev/mmcblk0p2`:

```
$ mkdir -p /mnt

$ mount /dev/mmcblk0p1 /mnt

$ cp xen_iasImage-gr-mrb-64 /mnt/iasImage

$ umount /mnt

$ mount /dev/mmcblk0p2 /mnt

$ cp xen_iasImage-gr-mrb-64 /mnt/iasImage

$ umount /mnt
```

3. Extract the DomU rootfs tar to the /dev/mmcblk0p3:

```
$ mount /dev/mmcblk0p3 /mnt

$ tar -xvf ias-kc-pf-image-domu-intel-corei7.tar.gz -C /mnt

$ umount /mnt
```

4. Extract the Dom0 rootfs tar to the /dev/mmcblk0p5:

```
$ mount /dev/mmcblk0p5 /mnt

$ tar -xvf ias-kc-pf-image-dom0-gr-mrb-64.tar.gz -C /mnt

$ umount /mnt
```

4.3 Booting the Xen GVT-g Software Image

Once you power cycle the target, it loads the Xen and dom0 kernel image from the eMMC partition `/dev/mmcblk0p2` and boots the kernel with its rootfs mounted at the `/dev/mmcblk0p5` partition.

DomU is automatically started by Dom0 via the systemd service `xenguest` and boots the kernel with its rootfs mounted at the `/dev/mmcblk0p3` partition.

Note: `mmcblk0p3` is mounted on `/mnt/xenguest` as Read Only.

5.0 Working with the Xen GVT-g Software

5.1 Useful Commands

- On Dom0, use the following command to view the list of domains. This command can also be used to find out the ID and Name of each domain:

  ```
  $ xl list
  ```

- To access the console of any VM from Dom0, use following command:

  ```
  $ xl console <VM Name>
  ```

  For example: `xl console DomU`

- To exit the DomU console, you can use : **CTRL+]**.

- To determine the GVT kernel being used, issue the command:

  $ uname –a

  `Linux gr-mrb-64 4.1.27-gvt-yocto-standard-gvt-bxt` would be visible on the console for the above command.

Note: Information about FPS should be displayed by default on screen. This option is enabled in the `/lib/systemd/system/glmark2_build.service` file. The option responsible for FPS display is –-annotate (in both Doms).

5.2 glmark Benchmark

By default, two glmark subtests are started on Dom0 and DomU after boot.

glmark is a non vsync bound (free-running) open-source benchmark and with preemption enabled on Dom0 (see Section 5.3), the GPU can be overwhelmed quickly and fps can drop on DomU side.

To overcome this, a patch has been developed to make glmark benchmark vsync bound.

The patch is available under:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-graphics/recipes-benchmark/glmark2/files/
```

File name:

```
0001-GLStateEGL-Added-vsync-option.patch
```

The default glmark2 binaries provided with the Xen GVT-g release, include this patch already.

The following is an example of how to run glmark with vsync:

```
$ glmark2-es2-wayland --vsync
```

Note: Information about FPS can be displayed on screen. In Xen GVT-g, 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).

5.3 Prioritized Rendering

Prioritized rendering , or “preemption”, allows high priority hardware GPU access to one or multiples workloads. By default, all applications run at normal graphics priority. At normal priority on Linux, the kernel driver submits commands in dependency order.

Preemption is driven by two Dom0 kernel command line arguments:

- `i915.enable_preemption`: Acts as a global switch for the feature. It can take one of two values:
  - 1 – Preemption is on.
  - 0 – Preemption is off.
- `i915.context_priority_mode`: Defines the feature policy. Can take one of two values:
  - 2 - All Dom0 workload is prioritized over DomU (this is default option).
  - 0 – Prioritized is disabled.

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

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-virtualization/recipes-
extended/xen/xen-4.6.0/dom0_cmdline.txt
```

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

```
$ cat /proc/cmdline
```

For more information about preemption and prioritized rendering, refer to the *Xen* GVT-g Preemption for Apollo Lake (APL-I) Developer Guide* (id: 573782).

5.4 Display Ready Notification

With the current release, display readiness is notified through a `uevent` with the property GVT_DISPLAY_READY=1.

You can capture this event by using the following command in Dom0:

```
$ udevadm monitor -p -k -s drm
```

You see messages similar to the following:

```
KERNEL[128.075490]
add      /devices/pci0000:00/0000:00:02.0/drm/card0 (drm)
ACTION=add
DEVNAME=/dev/dri/card0
DEVPATH=/devices/pci0000:00/0000:00:02.0/drm/card0
DEVTYPE=drm_minor
GVT_DISPLAY_READY=1
MAJOR=226
MINOR=0
SEQNUM=2806
SUBSYSTEM=drm
VMID=1
```

5.5 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` – Valid for Dom0 only. Specifies planes owners. 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).

  A value of 1 in the nibble means that the concerned plane is owned by DomU.

  A value of 0 in the nibble means that the concerned plane is owned by Dom0.

  The default value of each nibble is 0.

  For example, if: `i915.domain_plane_owners=0x11011100001`, then:
  - DomU owns plane 1 of pipe A, planes 2, 3 and 4 of pipe B, and planes 2 and 3 of pipe C.
  - Dom0 owns planes 2, 3 and 4 of pipe A, plane 1 of pipe B, and plane 1 of pipe C.

  Note: The maximum number of planes per pipe for Apollo Lake is different:
  - Pipe A and pipe B each support **four** planes
  - Pipe C supports **three** planes

  In runtime, you can check the owner of the planes using the following command on Dom0:

  ```
  $ cat /sys/kernel/vgt/control/plane_owner_show
  ```

- `i915.avail_planes_per_pipe` – Valid for DomU and Dom0. Specifies the number of planes exposed for use on the concerned 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 Dom0 is `i915.avail_planes_per_pipe=0x010102`.

This means that Dom0 exposes plane 1 of pipe C , plane 1 of pipe B and plane 2 of pipe A.

The default value for DomU is `i915.avail_planes_per_pipe=0x060E01`, where:

06 – in binary code is 0110 means that DomU exposes plane 2 and 3 on pipe C.

0E – in binary code is 1110 means that DomU exposes plane 2,3 and 4 of pipe B.

01 – in binary code is 0001 means that DomU exposes plane 1 of pipe A.

These flags can be adjusted at build time. They can be changed in the following file for Dom0:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-virtualization/recipes-extended/xen/xen-4.6.0/dom0_cmdline.txt
```

and the following file for DomU :

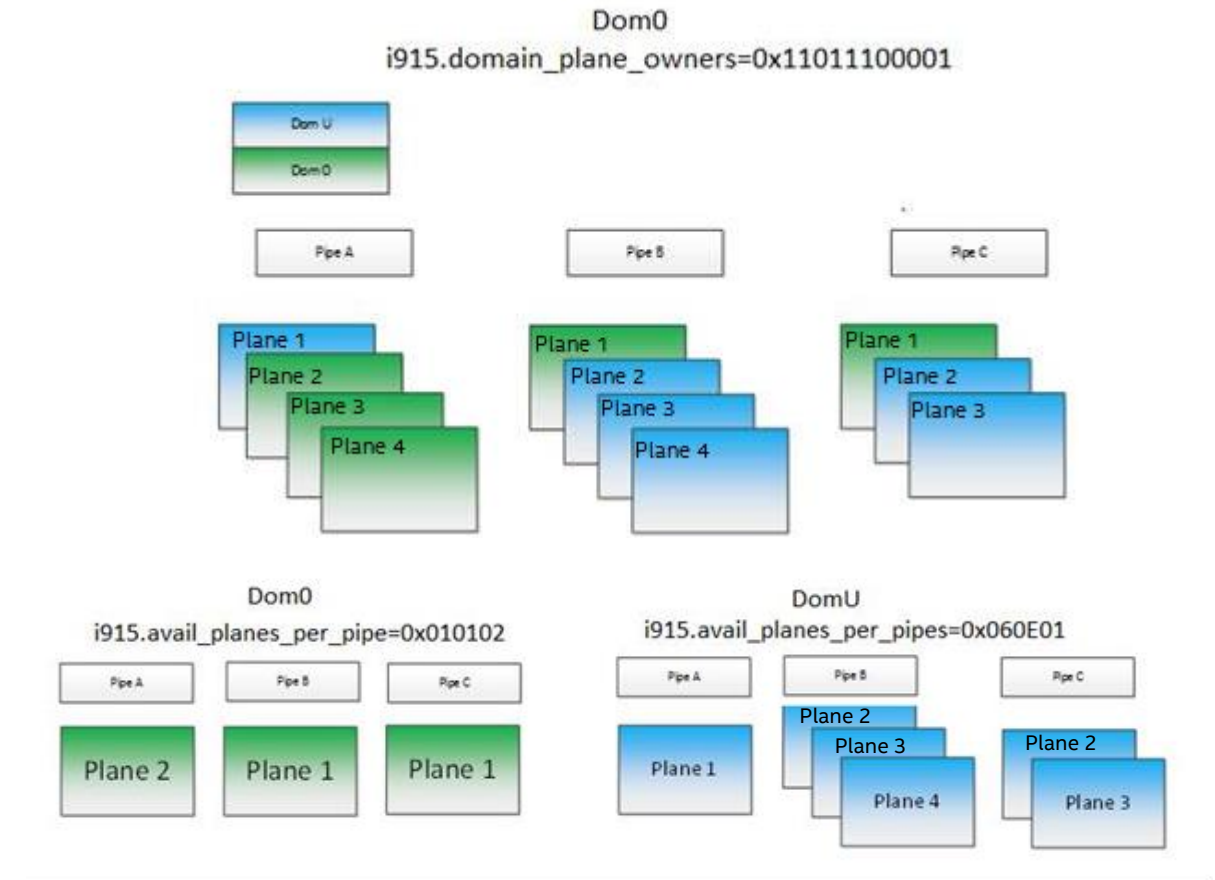
```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-virtualization/recipes-extended/xenguest-config/xenguest-config/xenguest.cfg
```

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 parameters value for different configuration of planes restriction.

Figure 4. Example of Kernel Command Line Parameter Values for Plane Restriction



5.6 Direct Display Mode

In the default `ias.conf` of DomU and Dom0, two displays are enabled in direct display mode.

Depending on 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 Dom0, the usual display names are: HDMI1, HDMI2 and eDP-1.

On DomU, the usual display names are: HDMI1, HDMI2 and HDMI3.

The following steps describe how to enable a 3rd monitor in direct display mode:

1. Connect two HDMI monitors and a 3rd eDP monitor.

2. Enable eDP in the Dom0 kernel command line (see the eDP important note in Section 2.5), which requires recreating the iasimage.

3. Ensure the appropriate plane ownership and exposure in the kernel command line (see Section 5.4).

   The following is an example of how to expose eDP planes for DomU:

   – Dom0 kernel command line options:

   ```
   i915.domain_plane_owners=0x11011101110
   i915.avail_planes_per_pipe=0x010101
   ```

   – DomU kernel command line options:

   ```
   i915.avail_planes_per_pipe=0x060E0E
   ```

4. Update the `/home/root/.config/ias.conf` file of Dom0 or DomU to enable eDP planes.

   The following is an exmaple of how to enable a third display with respect to above kernel options for planes restriction configuration and considering the usual connectors names, referred to earlier in this section:

   – On Dom0, add the following line in the `ias.conf` file:

   ```
   <crtc name='eDP1-0' model='classic' mode='preferred'>
     <output name='eDP1-0' size='inherit' position='rightof'
   target='HDMI2-0' />
   </crtc>
   ```

   - On DomU, 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 (Dom0 and DomU) to boot and start Weston and the glmark2 benches.

6. By default, the glmark2 bench surface is displayed on the plane declared as the “origin” in the `ias.conf` configuration. To place the display on a different plane, use the `surfctrl` command.

   For example:

   ```
   $ surfctrl  --surfname="glmark2"  --pos=3840x0
   ```

5.7 Multi-planes

The Xen GVT-g software allows up to four planes on pipe A and pipe B and three planes on pipe C.

See Section 5.5 for planes ownership and exposure management.

The default `ias.conf` of Dom0 defines one classic plane on HDMI1 and one flexible plane on HDMI2.

The default `ias.conf` of 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 DomU).

To move the display of a given Dom between different planes of a given display pipe, you need to:

1. Grant the plane ownership to the concerned Dom by changing kernel parameters (see Section 5.5).
   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>
...
```

2. 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=3840x0
```

5.8 Surface Sharing

The Xen GVTG-g software release allows DomU to share its display with Dom0 to get it displayed on a Dom0-owned plane.

Two mechanisms for surface sharing are:

- **Hyper_DMABUF sharing**: A genereic mechanism that allows the sharing of objects between both Doms. This is the default mechanism .
- **GGTT surface sharing**: A specific mechanism that allows only the sharing of display buffers between both Doms.

To enable surface sharing on DomU, two steps are required :

1. Enable the surface sharing in the `ias.conf` file with one of above sharing mechanism (see Section 5.8.1 or Section 5.8.2 following).

2. Declare the required surface to be shared, as sharable using the `surfctrl` command :

   To enable surface sharing for a given surface on DomU:

   ```
   $ surfctrl  --surfid=<surface_id> --shareable=1
   ```

   To disable surface sharing for a given surface on DomU:

   ```
   $ surfctrl  --surfid=<surface_id> --shareable=0
   ```

   To check the shareable flag (for example, id) for all existing surfaces on DomU:

   ```
   $ surfctrl
   ```

Note: Each individual surface is managed separately through `surfctrl` and needs to be declared shareable if sharing is required.

Note: By default, the shareable flag is set to 0 for a new surface, it has to be enabled for each new application started on DomU if sharing is required.

To display a shared surface on a Dom0-owned plane, `vmdisplay_wayland` application is used, see Section 5.8.1 or Section 5.8.2 following.

5.8.1 Hyper_DMABUF Surface Sharing

1. To enable a DomU shared surface display on a Dom0-owned plane using the Hyper_DMABUF surface sharing mechanism, two applications are involved:

   – `vmdisplay-server` (running in Dom0 / importer side). This application can be found under`/usr/bin/`

For example usage, enter:

```
$ /usr/bin/vmdisplay-server --help
```

– `vmdisplay-input` (running in DomU / exporter side). This application can be found under `/usr/bin/`

For example usage, enter:

```
$ /usr/bin/vmdisplay-input --help
```

Note: In the Xen image reference tree, a dedicated script is responsible for starting these applications automatically. The script can be found in: `/usr/bin/vmdisplay_daemon.py`. If a surface sharing issue is detected, it is worth checking if these processes are running in the background by using:
`$ ps –aux | grep vmdisplay`

If not, try running the following process manually.

To run `vmdisplay-server` on Dom0 use:
`$ systemctl start vmdisplay_importer`

To run `vmdisplay-input` on DomU use:
`$ systemctl start vmdisplay_exporter`

2. In DomU's `/home/root/.config/ias.conf` file, append:

- `vm='1'` flag to backend and output lines
- Optional attribute: `vm_share_only='1'` or `'0'` to backend line:
  - `vm_share_only='1'` - the surface is shared only, this is default value
  - `vm_share_only='0'` - the surface is shared and displayed simultaneously
- `vm_plugin_path` and `vm_plugin_args` flags to the backend line.
  - `Vm_plugin_path` – Should point to the library that implements the communication channel. In the Xen GVT-g reference tree, we are using `/usr/lib/Weston/vm-comm-net-work.so`, which is provided with Weston.
  - `vm_plugin_args` – Plugin-specific parameters.
    For the vm-comm-network, the arguments are `<ip address>:<port>` – These define the network address and port that the surface metadata will be available on (0 is the default IP of DomU). Standard port is 5554.

The modified `ias.conf` file of DomU should look like the following:

```
<backend raw_keyboards='1' use_nuclear_flip='1' vm='1' vm_share_only='0'
vm_plugin_path='/usr/lib/weston/vm-comm-network.so'
vm_plugin_args='0:5554'>
…
<crtc name='HDMI2' model='classic' mode='preferred'>
```

```
  <output name='HDMI2-0' size='inherit' position='origin' vm='1' />
<crtc name='HDMI1' model='classic' mode='preferred'>
       <output name='HDMI1-0' size='inherit' position='rightof'
target='HDMI2-0' vm='1'/>….
```

3. Stop/Start weston and the `glmark2_texture` application on DomU:

```
$ systemctl stop weston
$ systemctl stop glmark2_texture

$ systemctl start weston
$ systemctl start glmark2_texture
```

   At this point, DomU stops flipping the framebuffer and starts flipping the surface information table instead.

4. Declare the DomU surface as shareable, using:

```
$ surfctrl  --surfid=<surface_id> --shareable=1
```

5. Run the `vmdisplay-wayland` application on Dom0:

```
$ LIBGL_DRIVERS_PATH=/usr/lib/mesadri/ vmdisplay-wayland
<Domid> <surface name>
```

   For example:

```
$ LIBGL_DRIVERS_PATH=/usr/lib/mesadri/ vmdisplay-wayland 1
DomUonDom0
```

Note: Currently, `vmdisplay-wayland` is provided as a sample application with source code. It demonstrates the usage of indirect display and the surface sharing features.

5.8.2 GGTT Surface Sharing

Display sharing with the GGTT mechanism can be enabled using the following steps:

1. Change the DomU `/home/root/.config/ias.conf` file as follows :
   - Append the `vm='1'` flag and the `vm_use_ggtt='1'` flag to the backend line. For example:

```
  <backend raw_keyboards='1' use_nuclear_flip='1' vm='1'
vm_use_ggtt='1'>
```

   - Append the `vm='1'` flag to the output line. For example:

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

2. Stop/Start `weston` and the `glmark2_texture` application on DomU:

```
$ systemctl stop weston
$ systemctl stop glmark2_texture
```

```
$ systemctl start weston
$ systemctl start glmark2_texture
```

At this point, DomU stops flipping the framebuffer and starts flipping the surface information table instead.

3. Declare the DomU surface as shareable , using :

```
$ surfctrl  --surfid=<surface_id> --shareable=1
```

4. Run the `vmdisplay-wayland` application with an extra 'L' parameter:

```
$ LIBGL_DRIVERS_PATH=/usr/lib/mesadri/ vmdisplay-wayland <Dom id> <surface name> -L
```

For example:

```
$ LIBGL_DRIVERS_PATH=/usr/lib/mesadri/ vmdisplay-wayland 1 DomUonDom0 -L
```

6. At this point, the DomU and Dom0 display are visible on same Dom0 plane, one next to the other.

5.8.3 GGTT Surface Sharing with Constant Pinning

To improve GGTT surface sharing performance, a new option is available for the user, that is, constant pinning.

This option allow you to keep the GGTT offset of a shared frame pinned, instead of a pin/unpin each time.

Constant pinning is disabled by default and can be enabled by appending the `vm_pin='1'` flag to the backend line of DomU `ias.conf` file.

For example :

```
<backend raw_keyboards='1' use_nuclear_flip='1' vm='1' vm_use_ggtt='1' vm_pin='1'>
```

Then, follow the same steps to enable the usual GGTT-based surface sharing.

Note: Currently, `vmdisplay-wayland` is provided as a sample application with source code. It demonstrates the usage of indirect display and the surface sharing features.

Note: <Dom id> used in above `vmdisplay-wayland` command is the ID of the Dom, retrieved with the `xl list` command run from Dom0 prompt

Appendix A

A.1 Create and Customize the IasImage

We can customize the final generated IasImage thanks to the `ias_image_app` tool. This tool enables, among other things, the following:

- The creation of an IAS Image
- The stitching of one or multiple firmware files to an IAS Image (for example, in the case of Guc/Huc firmware filess)
- The stitching of an ACPI-wrapped binary into the IAS Image (for example, in the case of a vbt.bin file)
- The signing of an IasImage (for example, for ABL authentication)

The `ias_image_app` tool is available in sdk tools. Refer to Section 2.6.

Launching the tool with no extra parameters displays help text.

The basic prototype of an `ias_image_app` command is as follows:

```
$ ias_image_app -o <name_of_output_iasimage> -i image_type
<input_files>
```

In the Yocto* Xen reference release, the tool is used to create the final signed multi-boot iasImage, with the stitched binaries needed. The syntax can be found in the following receipe:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-virtualization/recipes-
extended/xen/xen_4.6.0.bbappend
```

The following are examples of the usage of the `ias_Image_app` tool:

- Create a non-signed multi-boot iasImage:

```
$ ./ias_image_app -o iasImage -i 0x40000 \
  xen_cmdline.txt xen-gr-mrb-64 dom0_cmdline.txt bzImage
```

- Create a signed multi-boot iasImage :

```
$ ./ias_image_app -o iasImage -i 0x40300 \
  xen_cmdline.txt xen-gr-mrb-64 dom0_cmdline.txt bzImage
```

  In the Yocto Xen reference release, bzImage and the zipped Xen image can be found under:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/build/tmp/deploy/images/gr-mrb-64/
```

- Create a signed iasImage with stitched Guc/Huc firmware:

```
$ echo -n "bxt_guc_ver8_7.bin" > guc.txt
$ echo -n "bxt_huc_ver01_07_2013.bin" > huc.txt
$ ./ias_image_app -o iasImage_fws -i 0x40000 \
        --page-aligned=1 \
        ./guc.txt ./bxt_guc_ver8_7.bin \
        ./huc.txt ./bxt_huc_ver01_07_2013.bin
$ ./ias_image_app -o iasImage -i 0x40000 \
        --page-aligned=6 \
        ./xen_cmdline.txt ./xen-gr-mrb-64 \
        ./dom0_cmdline.txt ./bzImage \
        ./iasImage_fws
```

  In the Yocto Xen reference release, the Guc and Huc firmware can be found under:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/build/tmp/sysroots/gr-mrb-64/lib/firmware/i915/
```

A.2 How to Configure SATA in Pass-through

This section explains how to configure the SATA device in pass-through so it can be owned by DomU instead of Dom0.

The procedure is valid for any PCI device.

Below are the steps to configure SATA in pass-through :

1. Determine the PCI ID of the SATA device using

   For example:

```
lspci | grep SATA
```

```
00:12.0 SATA controller: Intel Corporation
Atom/Celeron/Pentium Processor N4200/N3350/E3900 Series SATA
AHCI Controller (rev 0b)
```

2. Hide the PCI device by appending the `dom0_cmdline.txt` with `xen-pciback.hide=(00:12.0)`

3. Build Xen and the Dom0 images. See section 2.5.

4. Flash the new iasImage on the target. See section 4.2.

5. From Dom0 prompt, adapt the DomU `xenguest.cfg` file on the target (MRB) to assign the SATA Controller using PCI passthrough by adding the following line:

   ```
   pci=['00:12.0@12']
   ```

   to the `/opt/ias/xenguest/xenguest.cfg` file.

6. Power off the target and restart the MRB to use the newly installed kernel image.

7. Check whether the SATA (PCI) device is assigned to DomU by invoking `lspci` on the DomU console, as follows:

   ```
   xl console DomU
   lscpi | grep SATA
   00:12.0 SATA controller: Intel Corporation
   Atom/Celeron/Pentium Processor N4200/N3350/E3900 Series SATA
   ```

Note: The previous steps requires the Xen PCI-device backend driver to be compiled into kernel (by enabling the kernel config flag: `CONFIG_XEN_PCIDEV_BACKEND=y`.

A.3 How to start DomU on a D0 Low Board

The default `xenguest.cfg` file is not suitable to boot the DomU for Low boards with limited memory and CPU cores. To do so, a couple of changes are required in the `xen_cmdline.txt` file and the `xenguest.cfg` DomU configuration file.

The `xen_cmdline.txt` file is available under:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-
virtualization/recipes-extended/xen/xen-4.6.0/
```

The `xenguest.cfg` file is available under:

```
$GR_YOCTO/Xen_${REL_VER}-DOM0/meta-ias-
virtualization/recipes-extended/xenguest-config/xenguest-
config/
```

1. In the `xen_cmdline.txt` file, the Dom0 memory and vcore number must be reduced as follows:

   ```
   dom0_mem=1024M
   dom0_max_vcpus=1
   ```

2. In the `xenguest.cfg`, the DomU memory size needs to be reduced as below:

   ```
   memory = 850
   ```

A Dom0 rebuild or an iasImage re-creation with a re-flashing of the new iasImage is required to make use of the change in `xen_cmdline.txt`.

The `xenguest.cfg` file can be updated at run-time from Dom0 at the following location:

```
/opt/ias/xenguest/xenguest.cfg
```

Then, DomU needs to be started from Dom0 prompt, as follows:

```
systemctrl start xenguest
```