XEN Hypervisor

Xilinx Wiki

Exported on 06/29/2021

Table of Contents

1 Xen-Based-System Feature Summary

Xilinx provides reference designs which include core capabilities for Xen-based systems. The table below lists important system features and the first Xilinx Xen release that makes them available.

Feature	**Xen release**
Dom0less	2019.1
Cache Coloring	2020.1

2 Additional Commercial Support and Professional Services

Xilinx recommends our Premier Partner: DornerWorks[1] to customers seeking support beyond the example designs described above. DornerWorks has worked with customers to solve complex system problems including new OS support, frontend drivers[2], performance optimization, DMA accesses, and inter-OS communications.

1 http://www.dornerworks.com
2 https://wiki.xenproject.org/wiki/Frontend_Driver

3 Related Links

Non-hypervisor AMP Options and Viability: Unsupervised AMP[3]

3 https://xilinx-wiki.atlassian.net/wiki/display/A/Unsupervised+AMP

4 PDF Versions

Release
Latest[4] (PDF of Current Wiki)

4 https://xilinx-wiki.atlassian.net/wiki/download/attachments/18842530/XEN%20Hypervisor-20210624.pdf?api=v2&cacheVersion=1&modificationDate=1624562505317&version=1

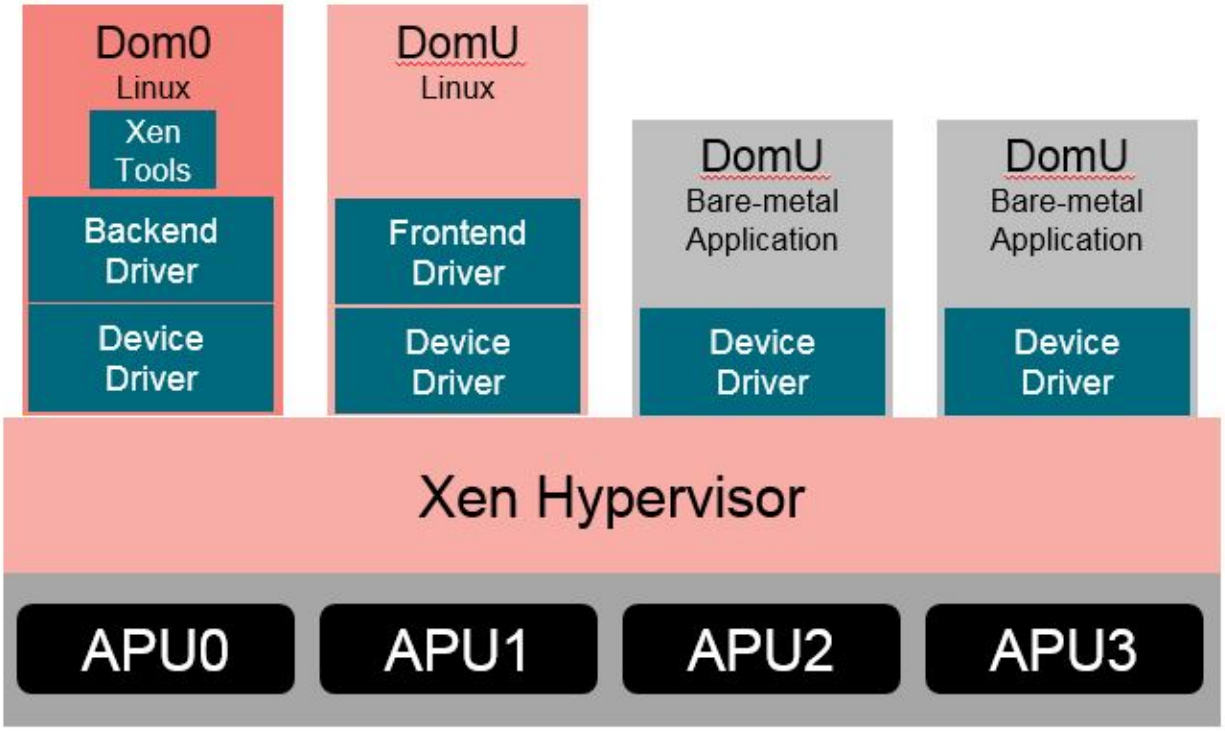
5 Overview

The purpose of this page is to give a high level overview of the Xen Hypervisor as it applies to Xilinx devices.

Xen is a type-1 Hypervisor defined, maintained and provided to the open source community by the Xen Project[5]. Xilinx actively contributes code to the Xen Project to provide Zynq UltraScale+ MPSoC platform support as well as key enhancements which benefit Xilinx customer use-cases.

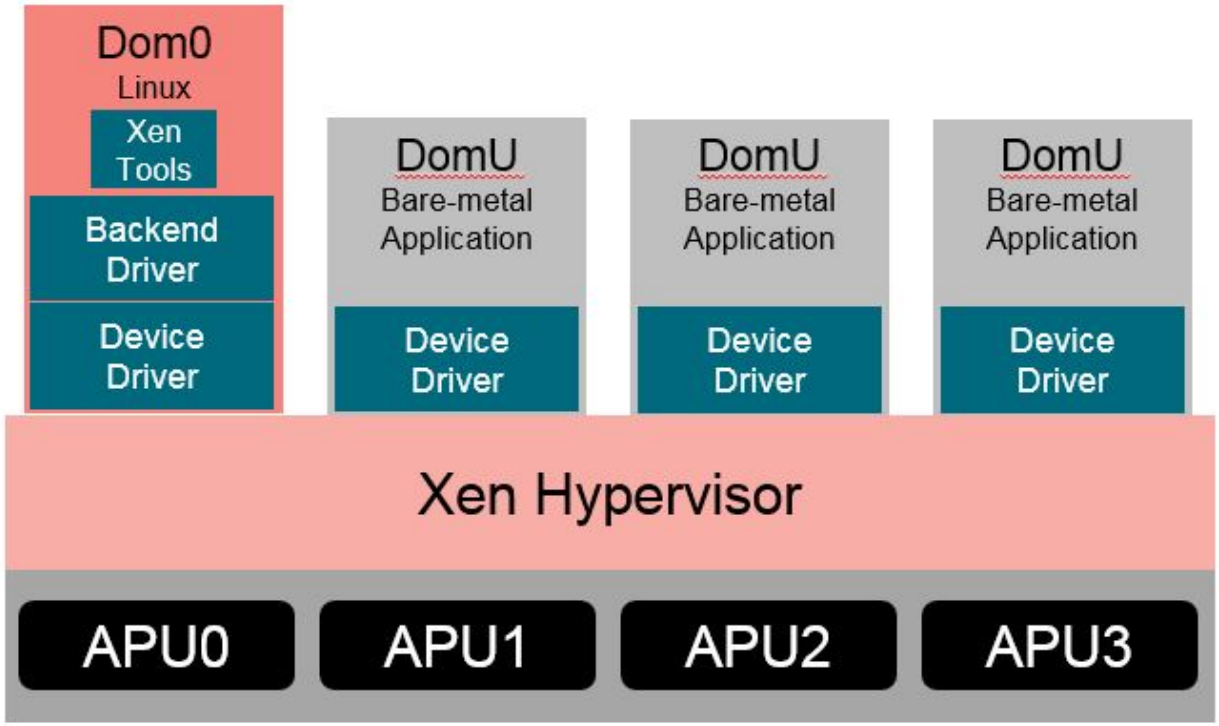
Xen allows multiple instances of operating system(s) or bare-metal applications to execute on Zynq UltraScale+ MPSoC. Additional information on the Xen hypervisor can be found at the Xen Project Getting Started[6] page.

Xilinx provides within the PetaLinux Tools and also in our Git tree, core elements and example designs to enable usage of Linux + bare-metal system configurations across the processing cores of Zynq UltraScale+ MPSoC. Key components of these example designs are described below in order to assist our customers to configure, build and deploy these basic configurations and to also identify current functionality gaps which may need to be further addressed within the customer's final system architecture.

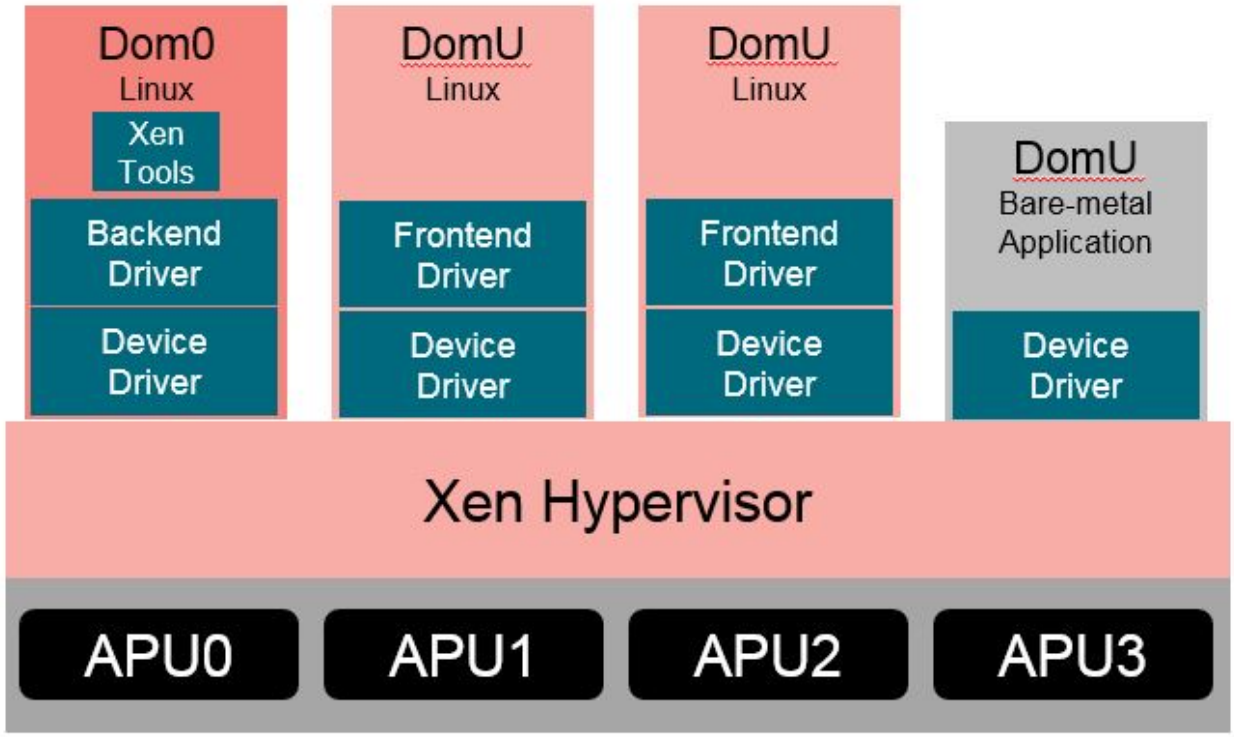


One Linux DomU + two Bare-Metal Applications

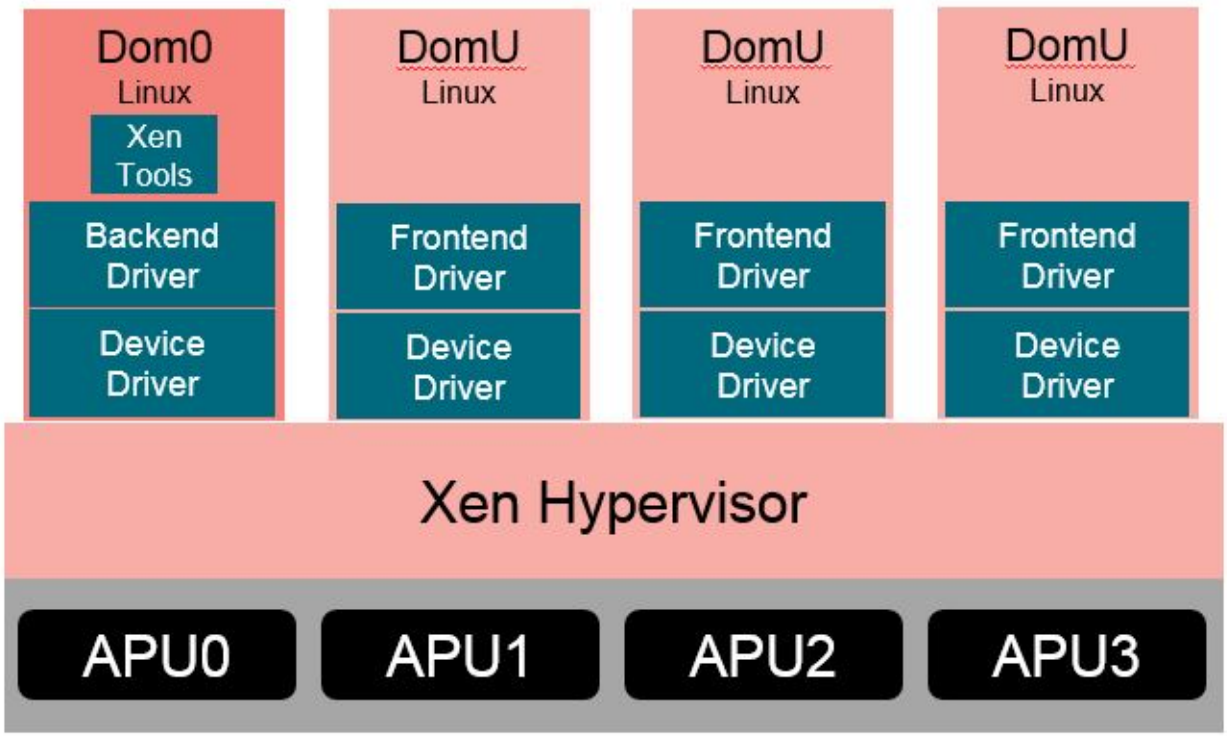
5 https://www.xenproject.org/
6 https://xenproject.org/developers/getting-started-devs/



Linux Dom0 with custom apps + three Bare-Metal Applications



Two Linux DomU + one Bare-Metal Application



Three Linux DomU

6 Building Xen Hypervisor through Yocto Flow

6.1 Table of Contents

6.2 Building the Xen Image

Below are the steps to build the Xen Image using Yocto:

6.2.1 Install the repo

Repo is a tool that enables the management of many git repositories given a single manifest file. The repo will fetch the git repositories specified in the manifest and, by doing so, sets up a Yocto Project build environment for you.

```
# Download the Repo script.
curl -k https://storage.googleapis.com/git-repo-downloads/repo > repo
# Generally, curl is install at /usr/bin/curl. If it is installed at custom location please point it to
correct location.

# Make it executable.
chmod a+x repo

# If it is correctly installed, you should see a Usage message when invoked with the help flag.
repo --help
# If system complaints about repo not found. Please run with './repo --help'
```

6.2.2 Fetch all sources

```
# initialize the repo.
repo init -u git://github.com/Xilinx/yocto-manifests.git -b <release-version>

# repo sync to get all sources
repo sync
```

Example of release-version: rel-v2020.2, rel-v2020.1 and rel-v2019.2 etc.

6.2.3 Source environment

```
# source the environment to build using bitbake.
source setupsdk
```

6.2.4 Add option to generate rootfs image with Yocto build

This option will generate the rootfs in cpio.gz format which will be used in creating boot-scripts(see page 20) step. Rootfs built will contain the useful Xen tools like xl.

```
# Edit conf/local.conf and Add the below line
IMAGE_FSTYPES += "cpio.gz"
```

6.2.5 Build using bitbake

Next, we will build the xen-image-minimal image. This will produce Xen, Linux kernel for Xen and rootfs. If only Xen needs to be build, it can done using MACHINE=vck190-versal bitbake xen.

```
 $ MACHINE=vck190-versal bitbake xen-image-minimal
# Please use appropriate yocto machine name to build for a target hardware. Examples:
#MACHINE=vck190-versal
#MACHINE=zcu102-zynqmp
#MACHINE=zcu104-zynqmp
#MACHINE=ultra96-zynqmp
```

This step can take a lot of time depending upon host machine processing power. It will also required significant disk space, please see Yocto[7] for more details.

6.3 Creating boot scripts

To create boot scripts, we use Imagebuilder[8] which generates a U-Boot script that loads all the necessary binaries and automatically adds the required entries to device tree at boot time. In order to use it, we will need to write a config file first.

First, cd to <current directory>/tmp/deploy/images/$(machine_name). Example: build/tmp/deploy/images/vck190-versal.

Next, create a file named "config" with the contents as illustrated below and save the file.

7 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto
8 https://gitlab.com/ViryaOS/imagebuilder

```
MEMORY_START="0x0"
MEMORY_END="0x80000000"

DEVICE_TREE="system.dtb"
XEN="xen"
DOM0_KERNEL="Image"
DOM0_RAMDISK="xen-image-minimal-vck190-versal.cpio.gz"

NUM_DOMUS=0

UBOOT_SOURCE="boot_xen.source"
UBOOT_SCRIPT="boot_xen.scr"
```

Save the config file.
Now *uboot-script-gen* can be used to generate *boot.scr*:

```
bash uboot-script-gen -c config -d . -t tftp
```

6.4 Running Xen on QEMU

Boot the qemu using below command:

```
MACHINE=vck190-versal runqemu xen-image-minimal
# Please change the machine name appropriatly.
```

Once you get to the u-boot prompt use the below commands to TFTP boot Xen.

```
dhcp
tftpb 0xC00000 boot_xen.scr; source 0xC00000
```

This should start the QEMU boot.

After you enter the above command, the QEMU boot sequence loads the Xen images and boots Linux. At the prompt login, enter root as the username and root as the password.

> To quit the emulation, press **CTRL+A** followed by **X**.

7 Building Xen Hypervisor with Petalinux 2020.1 and 2020.2

This page includes information on how to build and deploy Xen on Xilinx boards, including both **Xilinx Ultrascale+** and **Versal** boards.

7.1 Table of Contents

7.2 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux. Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions. The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit recipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/xen. You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

7.3 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Petalinux Package Groups  ---> packagegroup-petalinux-xen   --->  [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

> ⚠ NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.

You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ { };
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ {
     cpus {
         cpu@1 {
             device_type = "none";
         };
         cpu@2 {
             device_type = "none";
         };
         cpu@3 {
             device_type = "none";
         };
     };
 };
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend
```

and add this line to it: file://xen.dtsi.
The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

Then run petaliux-build:

```
$ petalinux-build
```

7.4 TFTP Booting Xen and Dom0

7.4.1 Run Xen dom0 on QEMU

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/xen"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

7.4.2 Run Xen dom0 on HW

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
u-boot> setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

7.4.3 TFTPing Xen using ImageBuilder

ImageBuilder is a set of Open Source community scripts to automatically configure a Xen system with Dom0 and multiple Dom0-less VMs for booting. ImageBuilder can generate a U-Boot script that loads all of the binaries automatically and boot the full system quickly. ImageBuilder is available here[9]. The main script is *scripts/uboot-script-gen* and its usage is described in details on the Xen Project wikipage[10].

Petalinux prebuilt binaries can be used in a config file as follows for *uboot-script-gen*:

```
MEMORY_START="0x0"
MEMORY_END="0x80000000"

DEVICE_TREE="xen.dtb"
XEN="xen"
DOM0_KERNEL="xen-Image"
DOM0_RAMDISK="xen-rootfs.cpio.gz"

NUM_DOMUS=0

UBOOT_SOURCE="boot.source"
UBOOT_SCRIPT="boot.scr"
```

Now *uboot-script-gen* can be used to generate *boot.scr*:

```
$ bash ./scripts/uboot-script-gen -c config -d . -t tftp
```

Boot the system with the following uboot command (assuming the tftp serverip is 10.0.2.2, which is typically the value for QEMU):

```
u-boot> setenv serverip 10.0.2.2
u-boot> tftpb 0xC00000 boot.scr; source 0xC00000
```

The Xen and Dom0 command line are generated by *uboot-script-gen.* If you would like to change anything, for instance increase the dom0 memory allocation, it is always possible by editing boot.source. Simply do the following:

9 https://gitlab.com/ViryaOS/imagebuilder
10 https://wiki.xenproject.org/wiki/ImageBuilder

- edit boot.source, change dom0_mem to dom0_mem=2G
- regenerate boot.scr with the following command: mkimage -A arm64 -T script -C none -a 0xC00000 -e 0xC00000 -d boot.source boot.scr

7.5 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. xen-Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen
5. xen-rootfs.cpio.gz)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/xen.

7.5.1 Booting with ImageBulider

ImageBuilder's script *uboot-script-gen* can be used to generate a uboot script that loads all the binaries automatically from MMC. Call uboot-script-gen with the following command, assuming that *$sdbootdev* is *0* and *$partid* is *1*:

```
$ bash ./scripts/uboot-script-gen -c config -d . -t "load mmc 0:1"
```

Copy the generated *boot.scr* onto the boot partition of the SD card. Boot the system with the following U-Boot command:

```
$ mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 0xC00000 boot.scr; source 0xC00000
```

7.6 Graphical Desktop in Dom0

ZU+ only.

To get a graphic desktop, e.g. matchbox, working in dom0, it is necessary to add two SMIDs to device tree: the SMID 0xce3 for zynqmp-display@fd4a0000 and SMID 0xce4 for dma@fd4c0000. The attached DTB[11] comes with the necessary modifications.

11 https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/xen-dp.dtb?api=v2&cacheVersion=1&modificationDate=1614215975287&version=1

7.7 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-built/
linux/xen,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/xen/xen-Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/xen/xen-Image root@<board-
ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
 root@plnx_aarch64:/etc/xen# xl list
 Name                                        ID   Mem VCPUs  State Time(s)
 Domain-0                                     0   512     1     r-----      79.8
 Domain-0                                     0   512     1     r-----      79.8
 guest0                                       1   256     2     ------      93.7
 guest1                                       2   256     2     ------      26.6
 guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

7.8 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

7.9 Starting Linux guests with Para-Virtual networking (PV network)

IMPORTANT: due to a bug, it is necessary to rebuild Xen 2020.1 and 2020.2 with this patch[12] applied to Xen to get PV drivers to work, including PV network.
This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-built/
linux/xen,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

2 https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/0001-xen-arm-fix-gnttab_need_iommu_mapping.patch?api=v2&cacheVersion=1&modificationDate=1613168003301&version=1

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

7.10 Starting Linux guests with Pass-through networking

It is possible to directly assign the network peripheral to a domU on both ZU+ and Versal. The following example is for ZU+.

Turn xen.dtb into xen.dts:

```
dtc -I dtb -O dts xen.dtb > xen.dts
```

The, edit xen.dts by adding xen,passthrough; under the node of the device to assign, in this case ethernet@ff0e0000:

```
ethernet@ff0e0000 {
            compatible = "cdns,zynqmp-gem";
            status = "enabled";
            interrupt-parent = <0x2>;
            interrupts = <0x0 0x3f 0x4 0x0 0x3f 0x4>;
            reg = <0x0 0xff0e0000 0x0 0x1000>;
            clock-names = "pclk", "hclk", "tx_clk";
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            #stream-id-cells = <0x1>;
            iommus = <0x6 0x77>;
            power-domains = <0x10>;
            clocks = <0xd 0xd 0xd>;
            phy-mode = "rgmii-id";
            xlnx,ptp-enet-clock = <0x0>;
            local-mac-address = [00 0a 35 00 02 90];
            phy-handle = <0x11>;
            linux,phandle = <0x22>;
            phandle = <0x22>;
            xen,passthrough;

            phy@c {
                reg = <0xc>;
                ti,rx-internal-delay = <0x8>;
                ti,tx-internal-delay = <0xa>;
                ti,fifo-depth = <0x1>;
                linux,phandle = <0x11>;
                phandle = <0x11>;
            };
        };
```

Convert xen.dts back into xen.dtb:

```
dtc -I dts -O dtb xen.dts > xen.dtb
```

7.11 Starting a Guest with a Passthrough SD Card

It is possible to directly assign the MMC controller to a domU on both ZU+ and Versal. The following example is for ZU+ only. It allows a Xen DomU full unmediated access to a SD card plugged into the board.

Add the following lines to the xl config file for the VM:

```
device_tree="/root/passthrough-example-part.dtb"
dtdev=["/amba/mmc@ff170000"]
iomem=["0xff170,1"]
irqs=[81]
```

Where **passthrough-example-part.dtb** is the following attached file:

passthrough-example-part.dtb[13]

Also, you need to add **xen,passthrough;** under the mmc node in the host device tree so that it doesn't get automatically assigned to dom0:

```
    mmc@ff170000 {
[...]

        clock-frequency = <0xb2cbcae>;
        xlnx,mio_bank = <0x1>;
        phandle = <0x24>;
        xen,passthrough; /* add this line */
    };
```

Finally, make sure to run the following commands at boot from xsdb to configure the system so that normal-world MMC DMA goes via the SMMU:

```
    mwr -force 0xff180408 0x20
    mwr -force 0xFF240000 0x100492
    mwr -force 0xFF240004 0x100492
```

7.11.1 Dom0less

For MMC assignment to a dom0less guest, use the attached partial device tree binary:

passthrough-example-part-dom0less.dtb[14]

To use it, just add it to your ImageBuilder config as follows:

```
NUM_DOMUS=1
DOMU_KERNEL[0]="xen-Image"
DOMU_RAMDISK[0]="initrd.cpio"
DOMU_PASSTHROUGH_DTB[0]="passthrough-example-part-dom0less.dtb"
```

Regenerate the boot.scr and boot.source scripts as usual with ImageBuilder's uboot-script-gen.

13 https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/passthrough-example-part.dtb?api=v2&cacheVersion=1&modificationDate=1614210949326&version=2

14 https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/passthrough-example-part-dom0less.dtb?api=v2&cacheVersion=1&modificationDate=1592937638654&version=1

As for the traditional Dom0 case, it is also necessary to add **xen,passthrough;** under the mmc node in the host device tree, and issue the three special writes at boot time to configure the system so that normal-world MMC DMA goes via the SMMU.

7.12 Starting a guest with a passthrough SATA disk

It is possible to directly assign the SATA controller to a domU on both ZU+ and Versal. The following example is for ZU+ only. It allows a Xen DomU full unmediated access to any SATA disks connected to it.

Add the following lines to the xl config file for the VM:

```
# Device passthroughs
dtdev = [ "/amba/ahci@fd0c0000" ]
device_tree = "/path/to/passthrough-example-sata.dtb"
irqs = [ 165 ]
iomem = [ "0xfd0c0,2" ]
```

Where **passthrough-example-sata.dtb** is this attached file[15].

Then, you need to add **xen,passthrough;** under the ahci@fd0c0000 node in the host device tree so that it doesn't get automatically assigned to dom0:

```
    ahci@fd0c0000 {
[...]

        phy-names = "sata-phy";
        phys = <0x35 0x1 0x1 0x1 0x7735940>;
        xlnx,tz-nonsecure-sata0 = <0x0>;
        xlnx,tz-nonsecure-sata1 = <0x0>;
        xen,passthrough; /* add this line */
    };
```

It is also necessary to add the four SMIDs of the SATA controller to device tree: 0x4c0, 0x4c1, 0x4c2, 0x4c3:

```
sata: ahci@fd0c0000 {
[...]

mmu-masters = < ... &sata 0x4c0 &sata 0x4c1 &sata 0x4c2 &sata 0x4c3>;
```

15https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/passthrough-example-sata.dtb?api=v2&cacheVersion=1&modificationDate=1614214588061&version=1

The attached DTB[16] comes with the necessary modifications.

Finally, make sure to run the following commands at boot from xsdb to configure the system so that normal-world SATA DMA goes via the SMMU:

```
mask_write 0xFD690020 0x0000000F 0x0000000F
```

16 https://xilinx-wiki.atlassian.net/wiki/download/attachments/384663561/xen-sata.dtb?api=v2&cacheVersion=1&modificationDate=1614215995310&version=1

8 Building Xen Hypervisor with Petalinux 2019.2

8.1 Table of Contents

8.2 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux
Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

8.3 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Petalinux Package Groups  ---> packagegroup-petalinux-xen   --->  [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ { };
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ {
    cpus {
        cpu@1 {
            device_type = "none";
        };
        cpu@2 {
            device_type = "none";
        };
        cpu@3 {
            device_type = "none";
        };
    };
 };
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend
```

and add this line to it: file://xen.dtsi.
The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

Then run petaliux-build:

```
$ petalinux-build
```

8.4 TFTP Booting Xen and Dom0

8.4.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

8.4.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
u-boot> setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

8.4.3 TFTPing Xen using ImageBuilder

ImageBuilder is a set of Open Source community scripts to automatically configure a Xen system with Dom0 and multiple Dom0-less VMs for booting. ImageBuilder can generate a U-Boot script that loads all of the binaries automatically and boot the full system quickly. ImageBuilder is available here[17]. The main script is *scripts/uboot-script-gen* and its usage is described in details on the Xen Project wikipage[18].

uboot-script-gen takes raw binaries as input (not uboot binaries). Petalinux pre-build images provide u-boot binaries for Xen and the Dom0 rootfs. First, they need to be converted into raw binaries:

```
dd if=xen.ub of=xen bs=64 skip=1
dd if=xen-rootfs.cpio.gz.u-boot of=xen-rootfs.cpio.gz bs=64 skip=1
```

Then, they can be used in a config file as follows for *uboot-script-gen*:

```
MEMORY_START="0x0"
MEMORY_END="0x80000000"

DEVICE_TREE="xen.dtb"
XEN="xen"
DOM0_KERNEL="xen-Image"
DOM0_RAMDISK="xen-rootfs.cpio.gz"

NUM_DOMUS=0

UBOOT_SOURCE="boot.source"
UBOOT_SCRIPT="boot.scr"
```

Now *uboot-script-gen* can be used to generate *boot.scr*:

```
$ bash ./scripts/uboot-script-gen -c config -d . -t tftp
```

Boot the system with the following uboot command (assuming the tftp serverip is 10.0.2.2, which is typically the value for QEMU):

```
u-boot> setenv serverip 10.0.2.2
u-boot> tftpb 0xC00000 boot.scr; source 0xC00000
```

17 https://gitlab.com/ViryaOS/imagebuilder
18 https://wiki.xenproject.org/wiki/ImageBuilder

The Xen and Dom0 command line are generated by *uboot-script-gen.* If you would like to change anything, for instance increase the dom0 memory allocation, it is always possible by editing boot.source. Simply do the following:

- edit boot.source, change dom0_mem to dom0_mem=2G
- regenerate boot.scr with the following command: mkimage -A arm64 -T script -C none -a 0xC00000 -e 0xC00000 -d boot.source boot.scr

8.4.4 TFTPing Xen from pre-built images manually

```
$ tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1400000 xen.ub; tftpb 9000000 xen-
rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

8.4.4.1 Booting with larger binaries and Workaround for ZCU104

Any time you update binaries and they potentially increase in size, please make sure that they don't overlap. For instance, if the kernel Image is loaded at 0x80000 and the dtb at 0x1280000, it means that the maximum size of the kernel is 12MB. More than that and the loading addresses need to be updated to avoid overlaps. Moreover, both address and size of the kernel Image are also specified under the /chosen node in the dtb file.

The ZCU104 BSP comes with a larger Dom0 rootfs (xen-rootfs.cpio.gz.u-boot file). U-boot has issues booting the rootfs as a u-boot image, it needs to be loaded as a raw binary instead. Thus, the following workaround is required for ZCU104.

Convert the xen-rootfs back to a normal binary:

```
$ dd if=xen-rootfs.cpio.gz.u-boot of=xen-rootfs.cpio.gz. bs=64 skip=1
```

Add the following node under /chosen in device tree xen-qemu.dts (you can convert a device tree binary to source and back using dtc):

```
dom0-ramdisk {

    compatible = "xen,linux-initrd", "xen,multiboot-module";

    reg = <0x0 0x9000000 152373016>;

};
```

Making sure that the size of the ramdisk (152373016 in the example) matches the actual size of xen-rootfs.cpio.gz.u-boot. Also add root=/dev/ram0 to the dom0 command line on device tree (/chosen/xen,dom0-bootargs).

Boot the system with the following command:

```
u-boot> tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1400000 xen.ub; tftpb 9000000 xen-
rootfs.cpio.gz; bootm 1400000 - 1280000
```

8.4.4.2 TFTPing Xen from your own images

```
u-boot> tftpb 1280000 system.dtb; tftpb 0x80000 Image; tftpb 1400000 xen.ub; tftpb 9000000
 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

Below is an example of what you will see.

```
[. . .]
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 Image; tftpb 1400000 xen.ub; tftpb 9000000
 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen-qemu.dtb'.
Load address: 0x1280000
Loading: ###
         18.3 MiB/s
done
Bytes transferred = 38412 (960c hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
[. . .]

Bytes transferred = 49329157 (2f0b405 hex)
## Booting kernel from Legacy Image at 01400000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    754000 Bytes = 736.3 KiB
   Load Address: 06000000
   Entry Point:  06000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 09000000 ...
   Image Name:   petalinux-user-image-zcu102-zynq
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    49329093 Bytes = 47 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 01000000
   Booting using the fdt blob at 0x1000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 030f4000, end 05fff3c5 ... OK
   Loading Device Tree to 0000000007ff3000, end 0000000007fff60b ... OK
```

```
Starting kernel ...

 Xen 4.9.2-pre
(XEN) Xen version 4.9.2-pre (@) (aarch64-xilinx-linux-gcc (GCC) 7.2.0) debug=n  Mon Mar 12 16:17:51 EDT
2018
(XEN) Latest ChangeSet: Sat Feb 24 07:47:11 2018 -0800 git:8c11d16-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305106 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Bringing up CPU1
(XEN) Bringing up CPU2
(XEN) Bringing up CPU3
(XEN) Brought up 4 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA and 8-bit VMID
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Loading kernel from boot module @ 0000000000080000
(XEN) Loading ramdisk from boot module @ 00000000030f4000
(XEN) Allocating 1:1 mappings totalling 768MB for dom0:
(XEN) BANK[0] 0x00000020000000-0x00000040000000 (512MB)
(XEN) BANK[1] 0x00000060000000-0x00000070000000 (256MB)
(XEN) Grant table range: 0x0000087fe00000-0x0000087fe5b000
(XEN) Loading zImage from 0000000000080000 to 0000000020080000-0000000023180000
(XEN) Loading dom0 initrd from 00000000030f4000 to 0x0000000028200000-0x000000002b10b3c5
(XEN) Allocating PPI 16 for event channel interrupt
(XEN) Loading dom0 DTB to 0x0000000028000000-0x0000000028008f30
```

```
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 280kB init memory.
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER4
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER8
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER12
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER16
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER20
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[    0.000000] Booting Linux on physical CPU 0x0
[. . .]
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2018.3 xilinx-zcu102-2018_3 /dev/hvc0

xilinx-zcu102-2018_3 login:
```

Login using 'root' as the username and password.

8.5 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

8.5.1 Booting with ImageBulider

ImageBuilder's script *uboot-script-gen* can be used to generate a uboot script that loads all the binaries automatically from MMC. First, convert the xen and xen-rootfs binaries into raw binaries and write an appropriate config file, see TFTPing Xen using ImageBuilder(see page 35) above. Then, call uboot-script-gen with the following command, assuming that *$sdbootdev* is *0* and *$partid* is *1*:

```
$ bash ./scripts/uboot-script-gen -c config -d . -t "load mmc 0:1"
```

Copy the generated *boot.scr* onto the boot partition of the SD card. Boot the system with the following U-Boot command:

```
$ mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 0xC00000 boot.scr; source 0xC00000
```

8.5.2 Booting with RootFS in Kernel (initramfs) manually

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image). Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1280000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1280000; load mmc $sdbootdev:$partid 1400000 xen.ub; bootm 1400000 -
1280000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

8.5.3 Booting RootFS mounted on RAM (initrd) manually

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1280000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1280000; load mmc $sdbootdev:$partid 1400000 xen.ub; load mmc $sdbootdev:
$partid 9000000 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

8.6 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/images/Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/images/Image root@<board-
ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
 root@plnx_aarch64:/etc/xen# xl list
 Name                                        ID   Mem VCPUs  State Time(s)
 Domain-0                                     0   512     1     r-----      79.8
 Domain-0                                     0   512     1     r-----      79.8
 guest0                                       1   256     2     ------      93.7
 guest1                                       2   256     2     ------      26.6
 guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

8.7 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

8.8 Starting Linux guests with Para-Virtual networking

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

8.9 Starting Linux guests with Pass-through networking

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1 $ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: "-drive file=qemu_sd.img,if=sd,format=raw,index=1"
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
u-boot> tftpb 1280000 xen-qemu.dtb; fdt addr 1280000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000
 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image;
tftpb 1400000 xen.ub; tftpb 9000000 xen-rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

The full command for booting images you built is shown below:

```
u-boot> tftpb 1280000 system.dtb; fdt addr 1280000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000
 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb
1400000 xen.ub; tftpb 9000000 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[19]
Once you have logged onto the system mount the SD card and copy the image.

```
mount /dev/mmcblk0 /mnt/ cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
cd /etc/xen
```

```
xl create -c example-passnet.cfg
```

19 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

9 Building Xen Hypervisor with Petalinux 2019.1

9.1 Table of Contents

9.2 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux
Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

9.3 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Petalinux Package Groups  ---> packagegroup-petalinux-xen   --->  [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ { };
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ {
     cpus {
         cpu@1 {
             device_type = "none";
         };
         cpu@2 {
             device_type = "none";
         };
         cpu@3 {
             device_type = "none";
         };
     };
 };
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend
```

and add this line to it: file://xen.dtsi.
The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

Then run petaliux-build:

```
$ petalinux-build
```

9.4 TFTP Booting Xen and Dom0

9.4.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

9.4.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

9.4.3 TFTPing Xen from pre-built images

```
$ tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1400000 xen.ub; tftpb 9000000 xen-
rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

9.4.4 Booting with larger binaries and Workaround for ZCU104

Any time you update binaries and they potentially increase in size, please make sure that they don't overlap. For instance, if the kernel Image is loaded at 0x80000 and the dtb at 0x1280000, it means that the maximum size of the kernel is 12MB. More than that and the loading addresses need to be updated to avoid overlaps. Moreover, both address and size of the kernel Image are also specified under the /chosen node in the dtb file.

The ZCU104 BSP comes with a larger Dom0 rootfs (xen-rootfs.cpio.gz.u-boot file). U-boot has issues booting the rootfs as a u-boot image, it needs to be loaded as a raw binary instead. Thus, the following workaround is required for ZCU104.

Convert the xen-rootfs back to a normal binary:

```
$ dd if=xen-rootfs.cpio.gz.u-boot of=xen-rootfs.cpio.gz. bs=64 skip=1
```

Add the following node under /chosen in device tree xen-qemu.dts (you can convert a device tree binary to source and back using dtc):

```
dom0-ramdisk {

    compatible = "xen,linux-initrd", "xen,multiboot-module";

    reg = <0x0 0x9000000 152373016>;

};
```

Making sure that the size of the ramdisk (152373016 in the example) matches the actual size of xen-rootfs.cpio.gz.u-boot. Also add root=/dev/ram0 to the dom0 command line on device tree (/chosen/xen,dom0-bootargs).

Boot the system with the following command:

```
$ tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1400000 xen.ub; tftpb 9000000 xen-
rootfs.cpio.gz; bootm 1400000 - 1280000
```

9.4.5 TFTPing Xen from your own images

```
$ tftpb 1280000 system.dtb; tftpb 0x80000 Image; tftpb 1400000 xen.ub; tftpb 9000000 rootfs.cpio.gz.u-boot;
bootm 1400000 9000000 1280000
```

Below is an example of what you will see.

```
[. . .]
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb 1280000 xen-qemu.dtb; tftpb 0x80000 Image; tftpb 1400000 xen.ub; tftpb 9000000
 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen-qemu.dtb'.
Load address: 0x1280000
Loading: ###
         18.3 MiB/s
done
Bytes transferred = 38412 (960c hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
[. . .]

Bytes transferred = 49329157 (2f0b405 hex)
## Booting kernel from Legacy Image at 01400000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    754000 Bytes = 736.3 KiB
   Load Address: 06000000
   Entry Point:  06000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 09000000 ...
   Image Name:   petalinux-user-image-zcu102-zynq
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    49329093 Bytes = 47 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 01000000
   Booting using the fdt blob at 0x1000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 030f4000, end 05fff3c5 ... OK
   Loading Device Tree to 0000000007ff3000, end 0000000007fff60b ... OK
```

```
Starting kernel ...

 Xen 4.9.2-pre
(XEN) Xen version 4.9.2-pre (@) (aarch64-xilinx-linux-gcc (GCC) 7.2.0) debug=n  Mon Mar 12 16:17:51 EDT
2018
(XEN) Latest ChangeSet: Sat Feb 24 07:47:11 2018 -0800 git:8c11d16-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305106 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Bringing up CPU1
(XEN) Bringing up CPU2
(XEN) Bringing up CPU3
(XEN) Brought up 4 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA and 8-bit VMID
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Loading kernel from boot module @ 0000000000080000
(XEN) Loading ramdisk from boot module @ 00000000030f4000
(XEN) Allocating 1:1 mappings totalling 768MB for dom0:
(XEN) BANK[0] 0x00000020000000-0x00000040000000 (512MB)
(XEN) BANK[1] 0x00000060000000-0x00000070000000 (256MB)
(XEN) Grant table range: 0x0000087fe00000-0x0000087fe5b000
(XEN) Loading zImage from 0000000000080000 to 0000000020080000-0000000023180000
(XEN) Loading dom0 initrd from 00000000030f4000 to 0x0000000028200000-0x000000002b10b3c5
(XEN) Allocating PPI 16 for event channel interrupt
(XEN) Loading dom0 DTB to 0x0000000028000000-0x0000000028008f30
```

```
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 280kB init memory.
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER4
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER8
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER12
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER16
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER20
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[    0.000000] Booting Linux on physical CPU 0x0
[. . .]
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2018.3 xilinx-zcu102-2018_3 /dev/hvc0

xilinx-zcu102-2018_3 login:
```

Login using 'root' as the username and password.

9.5 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

9.5.1 RootFS in Kernel (initramfs)

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image). Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1280000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1280000; load mmc $sdbootdev:$partid 1400000 xen.ub; bootm 1400000 -
1280000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

9.5.2 RootFS mounted on RAM (initrd)

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1280000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1280000; load mmc $sdbootdev:$partid 1400000 xen.ub; load mmc $sdbootdev:
$partid 9000000 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

9.6 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/images/Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/images/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
 root@plnx_aarch64:/etc/xen# xl list
 Name                                        ID   Mem VCPUs  State Time(s)
 Domain-0                                     0   512     1     r-----      79.8
 Domain-0                                     0   512     1     r-----      79.8
 guest0                                       1   256     2     ------      93.7
 guest1                                       2   256     2     ------      26.6
 guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

9.7 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical

CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

9.8 Starting Linux guests with Para-Virtual networking

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

9.9 Starting Linux guests with Pass-through networking

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1 $ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: "-drive file=qemu_sd.img,if=sd,format=raw,index=1"
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb 1280000 xen-qemu.dtb; fdt addr 1280000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb 1400000 xen.ub; tftpb 9000000 xen-rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

The full command for booting images you built is shown below:

```
$ tftpb 1280000 system.dtb; fdt addr 1280000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 1400000 xen.ub; tftpb 9000000 rootfs.cpio.gz.u-boot; bootm 1400000 9000000 1280000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[20]
Once you have logged onto the system mount the SD card and copy the image.

20 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

```
mount /dev/mmcblk0 /mnt/ cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
cd /etc/xen
```

```
xl create -c example-passnet.cfg
```

10 Building Xen Hypervisor with Petalinux 2018.3

10.1 Table of Contents

10.2 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux
Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

10.3 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Petalinux Package Groups  ---> packagegroup-petalinux-xen   --->  [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ { };
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ {
     cpus {
         cpu@1 {
             device_type = "none";
         };
         cpu@2 {
             device_type = "none";
         };
         cpu@3 {
             device_type = "none";
         };
     };
 };
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend
```

and add this line to it: file://xen.dtsi.
The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

Then run petaliux-build:

```
$ petalinux-build
```

10.4 TFTP Booting Xen and Dom0

10.4.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

10.4.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

10.4.3 TFTPing Xen from pre-built images

```
$ tftpb 1000000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1030000 xen.ub; tftpb 2000000 xen-
rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

10.4.4 TFTPing Xen from your own images

```
$ tftpb 1000000 system.dtb; tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot;
bootm 1030000 2000000 1000000
```

Below is an example of what you will see.

```
[. . .]
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb 1000000 system.dtb; tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000
 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen.dtb'.
Load address: 0x1000000
Loading: ###
         18.3 MiB/s
done
Bytes transferred = 38412 (960c hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
[. . .]

Bytes transferred = 49329157 (2f0b405 hex)
## Booting kernel from Legacy Image at 01030000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    754000 Bytes = 736.3 KiB
   Load Address: 06000000
   Entry Point:  06000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 02000000 ...
   Image Name:   petalinux-user-image-zcu102-zynq
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    49329093 Bytes = 47 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 01000000
   Booting using the fdt blob at 0x1000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 030f4000, end 05fff3c5 ... OK
   Loading Device Tree to 0000000007ff3000, end 0000000007fff60b ... OK
```

```
Starting kernel ...

 Xen 4.9.2-pre
(XEN) Xen version 4.9.2-pre (@) (aarch64-xilinx-linux-gcc (GCC) 7.2.0) debug=n  Mon Mar 12 16:17:51 EDT
2018
(XEN) Latest ChangeSet: Sat Feb 24 07:47:11 2018 -0800 git:8c11d16-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305106 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Bringing up CPU1
(XEN) Bringing up CPU2
(XEN) Bringing up CPU3
(XEN) Brought up 4 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA and 8-bit VMID
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Loading kernel from boot module @ 0000000000080000
(XEN) Loading ramdisk from boot module @ 00000000030f4000
(XEN) Allocating 1:1 mappings totalling 768MB for dom0:
(XEN) BANK[0] 0x00000020000000-0x00000040000000 (512MB)
(XEN) BANK[1] 0x00000060000000-0x00000070000000 (256MB)
(XEN) Grant table range: 0x0000087fe00000-0x0000087fe5b000
(XEN) Loading zImage from 0000000000080000 to 0000000020080000-0000000023180000
(XEN) Loading dom0 initrd from 00000000030f4000 to 0x0000000028200000-0x000000002b10b3c5
(XEN) Allocating PPI 16 for event channel interrupt
(XEN) Loading dom0 DTB to 0x0000000028000000-0x0000000028008f30
```

```
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 280kB init memory.
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER4
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER8
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER12
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER16
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER20
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[    0.000000] Booting Linux on physical CPU 0x0
[. . .]
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2018.3 xilinx-zcu102-2018_3 /dev/hvc0

xilinx-zcu102-2018_3 login:
```

Login using 'root' as the username and password.

10.5 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

10.5.1 RootFS in Kernel (initramfs)

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image). Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1000000; load mmc $sdbootdev:$partid 1030000 xen.ub; bootm 1030000 -
1000000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

10.5.2 RootFS mounted on RAM (initrd)

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1000000; load mmc $sdbootdev:$partid 1030000 xen.ub; load mmc $sdbootdev:
$partid 2000000 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

10.6 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/images/Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/images/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
 root@plnx_aarch64:/etc/xen# xl list
 Name                                        ID   Mem VCPUs  State Time(s)
 Domain-0                                     0   512     1     r-----      79.8
 Domain-0                                     0   512     1     r-----      79.8
 guest0                                       1   256     2     ------      93.7
 guest1                                       2   256     2     ------      26.6
 guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

10.7 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical

CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

10.8 Starting Linux guests with Para-Virtual networking

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

10.9 Starting Linux guests with Pass-through networking

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1 $ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: "-drive file=qemu_sd.img,if=sd,format=raw,index=1"
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb D80000 xen-qemu.dtb; fdt addr 1000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb 1030000 xen.ub; tftpb 2000000 xen-rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

The full command for booting images you built is shown below:

```
$ tftpb D80000 system.dtb; fdt addr 1000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[21]
Once you have logged onto the system mount the SD card and copy the image.

21 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

```
mount /dev/mmcblk0 /mnt/ cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
cd /etc/xen
```

```
xl create -c example-passnet.cfg
```

11 Building Xen Hypervisor with Petalinux 2018.1

Table of Contents

11.1 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux
Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

11.2 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Petalinux Package Groups  ---> packagegroup-petalinux-xen   --->  [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ { };
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen.dtsi"
/ {
     cpus {
         cpu@1 {
             device_type = "none";
         };
         cpu@2 {
             device_type = "none";
         };
         cpu@3 {
             device_type = "none";
         };
     };
 };
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend
```

and add this line to it: file://xen.dtsi.
The file should look like this:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://system-user.dtsi"
SRC_URI += "file://xen.dtsi"
```

Then run petaliux-build:

```
$ petalinux-build
```

11.3 TFTP Booting Xen and Dom0

11.3.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

11.3.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

11.3.3 TFTPing Xen from pre-built images

```
$ tftpb 1000000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1030000 xen.ub; tftpb 2000000 xen-
rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

11.3.4 TFTPing Xen from your own images

```
$ tftpb 1000000 system.dtb; tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot;
bootm 1030000 2000000 1000000
```

Below is an example of what you will see.

```
[. . .]
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb 1000000 system.dtb; tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000
 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen.dtb'.
Load address: 0x1000000
Loading: ###
         18.3 MiB/s
done
Bytes transferred = 38412 (960c hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
[. . .]

Bytes transferred = 49329157 (2f0b405 hex)
## Booting kernel from Legacy Image at 01030000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    754000 Bytes = 736.3 KiB
   Load Address: 06000000
   Entry Point:  06000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 02000000 ...
   Image Name:   petalinux-user-image-zcu102-zynq
   Image Type:   AArch64 Linux RAMDisk Image (uncompressed)
   Data Size:    49329093 Bytes = 47 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 01000000
   Booting using the fdt blob at 0x1000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 030f4000, end 05fff3c5 ... OK
   Loading Device Tree to 0000000007ff3000, end 0000000007fff60b ... OK
```

```
Starting kernel ...

 Xen 4.9.2-pre
(XEN) Xen version 4.9.2-pre (@) (aarch64-xilinx-linux-gcc (GCC) 7.2.0) debug=n  Mon Mar 12 16:17:51 EDT
2018
(XEN) Latest ChangeSet: Sat Feb 24 07:47:11 2018 -0800 git:8c11d16-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305106 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Bringing up CPU1
(XEN) Bringing up CPU2
(XEN) Bringing up CPU3
(XEN) Brought up 4 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA and 8-bit VMID
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Loading kernel from boot module @ 0000000000080000
(XEN) Loading ramdisk from boot module @ 00000000030f4000
(XEN) Allocating 1:1 mappings totalling 768MB for dom0:
(XEN) BANK[0] 0x00000020000000-0x00000040000000 (512MB)
(XEN) BANK[1] 0x00000060000000-0x00000070000000 (256MB)
(XEN) Grant table range: 0x0000087fe00000-0x0000087fe5b000
(XEN) Loading zImage from 0000000000080000 to 0000000020080000-0000000023180000
(XEN) Loading dom0 initrd from 00000000030f4000 to 0x0000000028200000-0x000000002b10b3c5
(XEN) Allocating PPI 16 for event channel interrupt
(XEN) Loading dom0 DTB to 0x0000000028000000-0x0000000028008f30
```

```
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 280kB init memory.
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER4
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER8
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER12
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER16
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER20
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[    0.000000] Booting Linux on physical CPU 0x0
[. . .]
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2018.1 xilinx-zcu102-2018_1 /dev/hvc0

xilinx-zcu102-2018_1 login:
```

Login using 'root' as the username and password.

11.4 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

11.4.1 RootFS in Kernel (initramfs)

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image). Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1000000; load mmc $sdbootdev:$partid 1030000 xen.ub; bootm 1030000 -
1000000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

11.4.2 RootFS mounted on RAM (initrd)

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 1000000; load mmc $sdbootdev:$partid 1030000 xen.ub; load mmc $sdbootdev:
$partid 2000000 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

11.5 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/images/Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/images/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
 root@plnx_aarch64:/etc/xen# xl list
 Name                                        ID   Mem VCPUs  State Time(s)
 Domain-0                                     0   512     1     r-----      79.8
 Domain-0                                     0   512     1     r-----      79.8
 guest0                                       1   256     2     ------      93.7
 guest1                                       2   256     2     ------      26.6
 guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

11.6 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical

CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

11.7 Starting Linux guests with Para-Virtual networking

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

11.8 Starting Linux guests with Pass-through networking

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1 $ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: "-drive file=qemu_sd.img,if=sd,format=raw,index=1"
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb D80000 xen-qemu.dtb; fdt addr 1000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb 1030000 xen.ub; tftpb 2000000 xen-rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

The full command for booting images you built is shown below:

```
$ tftpb D80000 system.dtb; fdt addr 1000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 1030000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[22]
Once you have logged onto the system mount the SD card and copy the image.

22 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

```
mount /dev/mmcblk0 /mnt/ cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
cd /etc/xen
```

```
xl create -c example-passnet.cfg
```

12 Building the Xen Hypervisor with PetaLinux 2017.3

12.1 The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux.

Table of Contents

12.2 Introduction

Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

12.3 Configuring and building XEN from source using PetaLinux

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Filesystem Packages ---> misc ---> packagegroup-petalinux-xen ---> [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configuration that expects the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS to boot DomU.

We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen-overlay.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen-overlay.dtsi"
/ {
};
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen-overlay.dtsi"
/ {
    cpus {
        cpu@1 {
            device_type = "none";
        };
        cpu@2 {
            device_type = "none";
        };
        cpu@3 {
            device_type = "none";
        };
    };
};
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend
```

and add this line to it: file://xen-overlay.dtsi.
The file should look like this:

```
SRC_URI_append ="\
    file://system-user.dtsi \
    file://xen-overlay.dtsi \
"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

Then run petaliux-build:

```
$ petalinux-build
```

12.4 TFTP Booting Xen and Dom0

12.4.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

12.4.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

12.4.3 TFTPing Xen from pre-built images

```
$ tftpb D80000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 1000000 xen.ub; tftpb 2000000 xen-
rootfs.cpio.gz.u-boot; bootm 1000000 2000000 D80000
```

12.4.4 TFTPing Xen from your own images

```
$ tftpb D80000 system.dtb; tftpb 0x80000 Image; tftpb 1000000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot;
bootm 1000000 2000000 D80000
```

Below is an example of what you will see.

```
[...]

BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP>
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb D80000 system.dtb; tftpb 0x80000 Image; tftpb 1000000 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-
boot; bootm 1000000 2000000 D80000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'system.dtb'.
Load address: 0xd80000
Loading: #########
   5.8 MiB/s
done
Bytes transferred = 42866 (a772 hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################

[...]

   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##################################################################
   ##########
   7 MiB/s
done
Bytes transferred = 38596692 (24cf054 hex)
## Booting kernel from Legacy Image at 01000000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    721216 Bytes = 704.3 KiB
   Load Address: 06000000
   Entry Point:  06000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 02000000 ...
   Image Name:   petalinux-user-image-plnx_aarch6
   Image Type:   AArch64 Linux RAMDisk Image (gzip compressed)
   Data Size:    38596628 Bytes = 36.8 MiB
   Load Address: 00000000
```

```
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 00d80000
   Booting using the fdt blob at 0xd80000
   Loading Kernel Image ... OK
   Loading Ramdisk to 7b9de000, end 7dead014 ... OK
   Loading Device Tree to 0000000007ff2000, end 0000000007fff771 ... OK

Starting kernel ...

 Xen 4.8.2-pre
(XEN) Xen version 4.8.2-pre (xbrbbot@) (aarch64-xilinx-linux-gcc (Linaro GCC 6.2-2016.11) 6.2.1 20161016)
debug=n  Thu Oct  5 21:45:37 MDT 2017
(XEN) Latest ChangeSet: Sun Mar 26 23:37:13 2017 +0200 git:89dceb9-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305006 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Brought up 1 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Loading kernel from boot module @ 0000000000080000
(XEN) Loading ramdisk from boot module @ 000000007b9de000
(XEN) Allocating 1:1 mappings totalling 768MB for dom0:
(XEN) BANK[0] 0x00000020000000-0x00000040000000 (512MB)
(XEN) BANK[1] 0x00000840000000-0x00000850000000 (256MB)
```

```
(XEN) Grant table range: 0x0000007fe00000-0x0000007fe56000
(XEN) Loading zImage from 0000000000080000 to 0000000020080000-0000000023180000
(XEN) Loading dom0 initrd from 000000007b9de000 to 0x0000000028200000-0x000000002a6cf014
(XEN) Allocating PPI 16 for event channel interrupt
(XEN) Loading dom0 DTB to 0x0000000028000000-0x0000000028008eb6
(XEN) Std. Loglevel: Errors and warnings
(XEN) Guest Loglevel: Nothing (Rate-limited: Errors and warnings)
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 272kB init memory.
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER4
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER8
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER12
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER16
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER20
(XEN) d0v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[    0.000000] Booting Linux on physical CPU 0x0

[...]

Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2017.3 xilinx-zcu102-2017_3 /dev/hvc0

xilinx-zcu102-2017_3 login:
```

Login using 'root' as the username and password.

12.5 SD Booting Xen and Dom0

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

12.5.1 RootFS in Kernel (initramfs)

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image). Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid D80000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr D80000; mmc $sdbootdev:$partid 1000000 xen.ub; bootm 1000000 - D80000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

12.5.2 RootFS mounted on RAM (initrd)

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid D80000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr D80000; load mmc $sdbootdev:$partid 1000000 xen.ub; load mmc $sdbootdev:
$partid 2000000 rootfs.cpio.gz.u-boot; bootm 1000000 2000000 D80000
```

12.6 Starting simple additional guests

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 pre-built/linux/images/Image
root@localhost:/boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no pre-built/linux/images/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
# cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
# xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
# xl list
```

To get back to the guests console:

```
# xl console guest0
```

You can create further guests by for example running:

```
# xl create example-simple.cfg name=\"guest1\"
# xl create example-simple.cfg name=\"guest2\"
root@plnx_aarch64:/etc/xen# xl list
Name                                        ID   Mem VCPUs  State Time(s)
Domain-0                                     0   512     1     r-----      79.8
Domain-0                                     0   512     1     r-----      79.8
guest0                                       1   256     2     ------      93.7
guest1                                       2   256     2     ------      26.6
guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
# xl destroy guest0
```

12.7 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical

CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

```
xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'
```

12.8 Starting Linux guests with Para-Virtual networking

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image (see earlier steps) we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  165.460858] xenbr0: port 1(eth0) entered blocking state
[  165.461819] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
# cd /etc/xen
```

The start DomU

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
$ ssh -p 2322 root@localhost
```

12.9 Starting Linux guests with Pass-through networking

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1
$ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Pass-through does not work in PetaLinux 2017.3 with the example pass-through device tree included in PetaLinux!
You will need to copy this device tree into Dom0 for 2017.3. PetaLinux 2017.4 does not require any changes from the included device tree:

```
/dts-v1/;

/ {
    #address-cells = <0x2>;
    #size-cells = <0x1>;

    passthrough {
        compatible = "simple-bus";
        ranges;
        #address-cells = <0x2>;
        #size-cells = <0x1>;

        misc_clk: misc_clk {
            #clock-cells = <0x0>;
            clock-frequency = <0x7735940>;
            compatible = "fixed-clock";
        };

        gem3: ethernet@ff0e0000 {
            compatible = "cdns,zynqmp-gem";
            status = "okay";
            interrupt-parent = <0x1>;
            interrupts = <0 63 4>, <0 63 4>;
            reg = <0x0 0xff0e0000 0x1000>;
            clock-names = "pclk", "hclk", "tx_clk", "rx_clk";
            #address-cells = <1>;
            #size-cells = <0>;
            clocks = <&&misc_clk &&misc_clk &&misc_clk &&misc_clk>;
            phy-mode = "rgmii-id";
            xlnx,ptp-enet-clock = <0x0>;
            local-mac-address = [00 0a 35 00 22 01];
            phy-handle = <&&phy1>;

            phy1: phy@c {
                reg = <0xc>;
                ti,rx-internal-delay = <0x8>;
                ti,tx-internal-delay = <0xa>;
                ti,fifo-depth = <0x1>;
                ti,rxctrl-strap-worka;
            };
        };
    };
};
```

We recommend compiling it to a DTB on the host machine and then copying it onto the SD card:

```
mcopy -i qemu_sd.img ./passthrough-example-part.dtb ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: "-drive file=qemu_sd.img,if=sd,format=raw,index=1"
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-
built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive
file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive
file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb D80000 xen-qemu.dtb; fdt addr D80000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status
"disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb
1000000 xen.ub; tftpb 2000000 xen-rootfs.cpio.gz.u-boot; bootm 1000000 2000000 D80000
```

The full command for booting images you built is shown below:

```
$ tftpb D80000 system.dtb; fdt addr D80000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status
"disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 1000000
 xen.ub; tftpb 2000000 rootfs.cpio.gz.u-boot; bootm 1000000 2000000 D80000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[23]
Once you have logged onto the system mount the SD card and copy the image.

```
mount /dev/mmcblk0 /mnt/
cp /mnt/Image /boot/
```

You also need to replace the included passthrough-example-part.dtb device tree. If it's on the SD card you can run this

```
rm /etc/xen/passthrough-example-part.dtb
cp /mnt/passthrough-example-part.dtb /etc/xen/passthrough-example-part.dtb
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
cd /etc/xen
```

```
xl create -c example-passnet.cfg
```

23 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

Unless you have manually built the latest commit of QEMU from master DomU will have before you reach the login prompt.

13 Building the Xen Hypervisor with PetaLinux 2017.1

13.1 The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux.

Table of Contents

13.2 Introduction

Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

13.3 Configuring and building XEN from source using PetaLinux 2017.1

First let's enable Xen to be built by default.

```
$ petalinux-config -c rootfs
```

Now let's enable Xen:

```
Filesystem Packages ---> misc ---> packagegroup-petalinux-xen ---> [*] packagegroup-petalinux-xen
```

Now we need to change the rootFS to be an INITRD

```
$ petalinux-config
```

And change

```
Image Packaging Configuration  --->  Root filesystem type (INITRAMFS)  ---> (X) INITRD
```

NOTE: This means that any images built will NOT have the rootFS in the Image that is built by PetaLinux. This means you will need to edit any scripts or configs that expect the rootFS to be included. This includes the Xen configs mentioned later.
You can still use the prebuilt Image file which does still include the rootFS.
We also want to edit the device tree to build in the extra Xen related configs.
Edit this file

```
project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi
```

and add this line: /include/ "xen-overlay.dtsi".
It should look like this for harware:

```
/include/ "system-conf.dtsi"
/include/ "xen-overlay.dtsi"
/ {
};
```

or like this for QEMU:

```
/include/ "system-conf.dtsi"
/include/ "xen-overlay.dtsi"
/ {
    cpus {
        cpu@1 {
            device_type = "none";
        };
        cpu@2 {
            device_type = "none";
        };
        cpu@3 {
            device_type = "none";
        };
    };
};
```

NOTE: There is a bug on QEMU where the CPUs running in SMP sometimes cause hangs. To avoid this we only tell Xen about a single CPU.

Also edit this file:

```
project-spec/meta-user/recipes-bsp/device-tree/device-tree-generation_%.bbappend
```

and add this line to it: file://xen-overlay.dtsi.
The file should look like this:

```
SRC_URI_append ="\
    file://system-user.dtsi \
    file://xen-overlay.dtsi \
"
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

Then run petaliux-build:

```
$ petalinux-build
```

13.4 TFTP Booting Xen and Dom0 2017.1

13.4.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

13.4.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

13.4.3 TFTPing Xen from pre-built images

```
$ tftpb 4000000 xen-qemu.dtb; tftpb 0x80000 xen-Image; tftpb 6000000 xen.ub; tftpb 0x1000000 xen-
rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
```

13.4.4 TFTPing Xen from your own images

```
$ tftpb 4000000 system.dtb; tftpb 0x80000 Image; tftpb 6000000 xen.ub; tftpb 0x1000000 rootfs.cpio.gz.u-
boot; bootm 6000000 0x1000000 4000000
```

Below is an example of what you will see.

```
[...]

BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Hit any key to stop autoboot:  0
ZynqMP>
ZynqMP> setenv serverip 10.0.2.2
ZynqMP> tftpb 4000000 system.dtb; tftpb 0x80000 Image; tftpb 6000000 xen.ub; tftpb 0x1000000
 rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'system.dtb'.
Load address: 0x4000000
Loading: #########
         5.8 MiB/s
done
Bytes transferred = 42748 (a6fc hex)
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'Image'.
Load address: 0x80000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################

[...]

         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ################################
         8.2 MiB/s
done
Bytes transferred = 43688238 (29aa12e hex)
## Booting kernel from Legacy Image at 06000000 ...
   Image Name:
```

```
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    721216 Bytes = 704.3 KiB
   Load Address: 05000000
   Entry Point:  05000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 01000000 ...
   Image Name:   petalinux-user-image-plnx_aarch6
   Image Type:   AArch64 Linux RAMDisk Image (gzip compressed)
   Data Size:    43688174 Bytes = 41.7 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 04000000
   Booting using the fdt blob at 0x4000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 7b539000, end 7dee30ee ... OK
   Loading Device Tree to 0000000007ff2000, end 0000000007fff6fb ... OK

Starting kernel ...

 Xen 4.8.1-pre
(XEN) Xen version 4.8.1-pre (alistai@) (aarch64-xilinx-linux-gcc (Linaro GCC 6.2-2016.11) 6.2.1 20161016)
debug=n  Wed Mar 22 14:11:36 MDT 2017
(XEN) Latest ChangeSet: Wed Feb 22 15:46:19 2017 +0100 git:e9e1b9b-dirty
(XEN) Processor: 410fd034: "ARM Limited", variant: 0x0, part 0xd03, rev 0x4
(XEN) 64-bit Execution:
(XEN)   Processor Features: 0000000000002222 0000000000000000
(XEN)     Exception Levels: EL3:64+32 EL2:64+32 EL1:64+32 EL0:64+32
(XEN)     Extensions: FloatingPoint AdvancedSIMD
(XEN)   Debug Features: 0000000010305006 0000000000000000
(XEN)   Auxiliary Features: 0000000000000000 0000000000000000
(XEN)   Memory Model Features: 0000000000001122 0000000000000000
(XEN)   ISA Features:  0000000000011120 0000000000000000
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001231:00011011
(XEN)     Instruction Sets: AArch32 A32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 03010066
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01260000 02102211
(XEN)  ISA Features: 02101110 13112111 21232042 01112131 00011142 00011121
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27 Freq: 50000 KHz
(XEN) GICv2 initialization:
(XEN)         gic_dist_addr=00000000f9010000
(XEN)         gic_cpu_addr=00000000f9020000
(XEN)         gic_hyp_addr=00000000f9040000
(XEN)         gic_vcpu_addr=00000000f9060000
(XEN)         gic_maintenance_irq=25
(XEN) GICv2: Adjusting CPU interface base to 0xf902f000
(XEN) GICv2: 192 lines, 4 cpus (IID 00000000).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) Bringing up CPU1
(XEN) Bringing up CPU2
(XEN) Bringing up CPU3
```

```
(XEN) Brought up 4 CPUs
(XEN) P2M: 40-bit IPA with 40-bit PA
(XEN) P2M: 3 levels with order-1 root, VTCR 0x80023558
/amba@0/smmu0@0xFD800000: Decode error: write to 6c=0
(XEN) I/O virtualisation enabled
(XEN)  - Dom0 mode: Relaxed
(XEN) Interrupt remapping enabled

[...]

Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2017.1 plnx_aarch64 /dev/hvc0

INIT: Id "PS0" respawning too fast: disabled for 5 minutes

PetaLinux 2017.1 plnx_aarch64 /dev/hvc0

plnx_aarch64 login:
```

Login using 'root' as the username and password

13.5 SD Booting Xen and Dom0 2017.1

To boot Xen from an SD card you need to copy the following files to the boot partition of the SD card:

1. BOOT.bin
2. Image
3. the compiled device tree file renamed to system.dtb (xen.dtb or xen-qemu.dtb for QEMU from the pre-built images, system.dtb from a Petalinux build)
4. xen.ub
5. rootfs.cpio.gz.u-boot (Only if using initrd instead of initramfs for the rootfs)

When using the pre-built images from the BSP, copy these files from <project-dir>/pre-built/linux/images. The prebuilt images are built to support both Linux without Xen and with Xen such that some of the Xen based image file names are different than in a normal Petalinux build. The prebuilt Linux kernel image includes an initramfs rootfs. Petalinux builds (rather than prebuilt images) require an initrd rootfs such that another file for the rootfs must also be used as described below.

13.5.1 RootFS in Kernel (initramfs)

This method allows the use of a Linux kernel with an initramfs (such as the prebuilt image) . Boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 4000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 4000000; mmc $sdbootdev:$partid 6000000 xen.ub; bootm 6000000 - 4000000
```

This would also allow a rootfs mounted on the SD card to be used. It requires that you extract the rootFS (cpio file) to the root partition on the SD card and setup the kernel to expect the rootfs on the SD card in the kernel command line.

13.5.2 RootFS mounted on RAM (initrd)

This method is required when using a Linux image which is initrd based and does not include a rootfs. The rootfs.cpio.gz.u-boot file will be loaded in memory from u-boot. Then boot the SD card on hardware or QEMU and stop the u-boot autoboot. At the u-boot prompt run:

```
mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 4000000 system.dtb &&&& load mmc $sdbootdev:
$partid 0x80000 Image; fdt addr 4000000; load mmc $sdbootdev:$partid 6000000 xen.ub; load mmc $sdbootdev:
$partid 9000000 rootfs.cpio.gz.u-boot; bootm 6000000 9000000 4000000
```

13.6 Starting simple additional guests (PetaLinux 2016.4 or later)

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 images/linux/Image root@localhost:/
boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no images/linux/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
$ cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
xl list
```

To get back to the guests console:

```
xl console guest0
```

You can create further guests by for example running:

```
xl create example-simple.cfg name=\"guest1\"
xl create example-simple.cfg name=\"guest2\"
root@plnx_aarch64:/etc/xen# xl list
Name                                        ID   Mem VCPUs  State Time(s)
Domain-0                                     0   512     1     r-----      79.8
Domain-0                                     0   512     1     r-----      79.8
guest0                                       1   256     2     ------      93.7
guest1                                       2   256     2     ------      26.6
guest2                                       3   256     2     ------       1.8
```

To destroy a guest:
xl destroy guest0

13.7 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.

To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'

13.8 Starting Linux guests with Para-Virtual networking (PetaLinux 2016.4 or later)

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  186.459495] xenbr0: port 1(eth0) entered blocking state
[  186.461194] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
$ cd /etc/xen
```

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
ssh -p 2322 root@localhost
```

13.9 Starting Linux guests with Pass-through networking (PetaLinux 2017.1)

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.

You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1
$ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: -drive file=qemu_sd.img,if=sd,format=raw,index=1
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/
linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive
file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive
file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb 4000000 xen-qemu.dtb; fdt addr 4000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status
"disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb
6000000 xen.ub; tftpb 0x1000000 xen-rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
```

The full command for booting images you built is shown below:

```
$ tftpb 4000000 system.dtb; fdt addr 4000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status
"disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 6000000
 xen.ub; tftpb 0x1000000 rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[24]
Once you have logged onto the system mount the SD card and copy the image.

```
# mount /dev/mmcblk0 /mnt/
# cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
$ cd /etc/xen
```

```
# xl create -c example-passnet.cfg
```

24 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

14 Building the Xen Hypervisor with PetaLinux 2016.4 and newer

Table of Contents

14.1 Overview

The guide below shows you how to build Xen, boot Xen and then run some example configurations on ZU+. The steps below use PetaLinux and assume you have some knowledge of using PetaLinux.
Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.
The default PetaLinux configuration has images ready to do boot Xen, these are the pre-built images. You can use those or you can manually edit receipes and build Xen yourself. The pre-built images can be found in this directory (inside a PetaLinux project) pre-built/linux/images/ and prefixed with "xen-". You can either use the pre-builts or follow the next section to configure and build Xen yourself. If you are using the pre-builts you can skip to the booting Xen section for your release version.

14.2 Configuring and building XEN from source using PetaLinux 2016.4

To build the XEN tools and XEN hypervisor you will need to edit the following two files:

```
$ vim project-spec/meta-plnx-generated/recipes-core/images/petalinux-user-image.bb
```

Remove or comment out '#' the following line

```
EXTRA_USERS_PARAMS = "usermod -P root root;"*
```

The second file that needs to be edited is:

```
$ vim build/conf/local.conf
```

Add the extra line at the end:

```
EXTRA_IMAGE_FEATURES = "debug-tweaks"
```

NOTE: You might need to have already run the build command below to be able to edit this file.

```
$ petalinux-build -c xen-image-minimal
```

NOTE: As of PetaLinux 2016.4 the petalinux-build command no longer builds Xen by default, so the above command is required

You have to manually copy the xen-qemu.dtb from the pre-built directory:

```
$ cp ./pre-built/linux/images/xen-qemu.dtb ./images/linux/
```

You will also need to manually copy the recently built Xen images

```
$ cp build/tmp/deploy/images/plnx_aarch64/xen.ub ./images/linux/
$ cp build/tmp/deploy/images/plnx_aarch64/Image ./images/linux/xen-Image
$ cp build/tmp/deploy/images/plnx_aarch64/xen-image-minimal-plnx_aarch64.cpio.gz.u-boot ./images/linux/xen-rootfs.cpio.gz.u-boot
```

NOTE: If you re-build anything else after Xen the following images will be over-written

14.3 TFTP Booting Xen and Dom0 2016.4

14.3.1 Run Xen dom0 on QEMU:

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-built/linux/images"
```

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

14.3.2 Run Xen dom0 on HW:

To use the prebuilt Xen on hardware:

```
$ petalinux-boot --jtag --prebuilt 2
```

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

```
$ tftpb 4000000 xen-qemu.dtb
$ tftpb 0x80000 xen-Image
$ tftpb 6000000 xen.ub
$ tftpb 0x1000000 xen-rootfs.cpio.gz.u-boot
$ bootm 6000000 0x1000000 4000000
```

Below is an example of what you will see.

```
ZynqMP> tftpb 4000000 xen-qemu.dtb
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen-qemu.dtb'.
Load address: 0x4000000
Loading: ######
    4 MiB/s
done
Bytes transferred = 29417 (72e9 hex)
ZynqMP> tftpb 0x80000 xen-Image
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen-Image'.
Load address: 0x80000
Loading: #################################################################
    #################################################################
    #################################################################

[...]

    #################################################################
    ###############################################################
    5.4 MiB/s
done
Bytes transferred = 12297216 (bba400 hex)
ZynqMP> tftpb 6000000 xen.ub
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen.ub'.
Load address: 0x6000000
Loading: #################################################################
    #################################################################
    ############
    4.2 MiB/s
done
Bytes transferred = 722528 (b0660 hex)
ZynqMP> tftpb 0x1000000 xen-rootfs.cpio.gz.u-boot
Using ethernet@ff0e0000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'xen-rootfs.cpio.gz.u-boot'.
Load address: 0x1000000
Loading: #################################################################
    #################################################################
    #################################################################
    #################################################################
    #################################################################

[...]

    #################################################################
    #################################################################
    #####
    5.2 MiB/s
done
```

```
Bytes transferred = 36571940 (22e0b24 hex)
ZynqMP> bootm 6000000 0x1000000 4000000
## Booting kernel from Legacy Image at 06000000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    722464 Bytes = 705.5 KiB
   Load Address: 05000000
   Entry Point:  05000000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 01000000 ...
   Image Name:   xen-image-minimal-plnx_aarch64-2
   Image Type:   AArch64 Linux RAMDisk Image (gzip compressed)
   Data Size:    36571876 Bytes = 34.9 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 04000000
   Booting using the fdt blob at 0x4000000
   Loading Kernel Image ... OK
   Loading Ramdisk to 05d1f000, end 07fffae4 ... OK
   Loading Device Tree to 0000000005d14000, end 0000000005d1e2e8 ... OK

Starting kernel ...

[...]


Starting OpenBSD Secure Shell server: sshd
  generating ssh RSA key...
  generating ssh ECDSA key...
  generating ssh DSA key...
  generating ssh ED25519 key...
done.
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]

PetaLinux 2016.4 plnx_aarch64 /dev/hvc0

plnx_aarch64 login:
```

Login using 'root' as the username

14.4 Starting simple additional guests (PetaLinux 2016.4 or later)

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 has finished booting, we'll need to copy a guest Image into dom0's filesystem.
We'll use the base prebuilt PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:
To target QEMU run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -P 2222 images/linux/Image root@localhost:/
boot/
```

If running on hardware run the following on the host:

```
scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no images/linux/Image root@<board-ip>:/boot/
```

If you would prefer to load DomU's kernel to the guest via SD card, you can follow the instructions in the "Starting Linux guests with Pass-through networking" section.

The xen-image-minimal rootFS includes some prepared configurations that you can use. These are located in '/etc/xen/'

```
$ cd /etc/xen
```

To start a simple guest run the following from the dom0 prompt

```
xl create -c example-simple.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
Once at the dom0 prompt you can list the guests from dom0:

```
xl list
```

To get back to the guests console:

```
xl console guest0
```

You can create further guests by for example running:

```
xl create example-simple.cfg name=\"guest1\"
xl create example-simple.cfg name=\"guest2\"
root@plnx_aarch64:/etc/xen# xl list
Name                                        ID   Mem VCPUs  State Time(s)
Domain-0                                     0   512     1     r-----      79.8
Domain-0                                     0   512     1     r-----      79.8
guest0                                       1   256     2     ------      93.7
guest1                                       2   256     2     ------      26.6
guest2                                       3   256     2     ------       1.8
```

To destroy a guest:
xl destroy guest0

14.5 CPU Pinning

The following will only work on QEMU with multi-core enabled or on real HW.
When running multiple guests with multiple Virtual CPUs, Xen will schedule the various vCPUs onto real physical CPUs.
The rules and considerations taken in scheduling decisions depend on the chosen scheduler and the configuration.
To avoid having multiple vCPUs share a single pCPU, it is possible to pin a vCPU onto a pCPU and to give it exclusive access.

To create a simple guest with one Virtual CPU pinned to Physical CPU #3, you can do the following:

```
xl create example-simple.cfg 'name="g0"' 'vcpus="1"' 'cpus="3"'
```

Another way to pin virtual CPUs on to Physical CPUs is to create dedicated cpu-pools.
This has the advantage of isolating the scheduling instances.

By default a single cpu-pool named Pool-0 exists. It contains all the physical cpus.
We'll now create our pool named rt using the credit2 scheduler.

```
xl cpupool-create 'name="rt"' 'sched="credit"'
xl cpupool-cpu-remove Pool-0 3
xl cpupool-cpu-add rt 3
```

Now we are ready to create a guest with a single vcpu pinned to physical CPU #3.

xl create /etc/xen/example-simple.cfg 'vcpus="1"' 'pool="rt"' 'cpus="3"' 'name="g0"'

14.6 Starting Linux guests with Para-Virtual networking (PetaLinux 2016.4 or later)

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt and have copied a domU image we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
# killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
# ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
# ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
# brctl addbr xenbr0
# brctl addif xenbr0 eth0
# /sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.24.1) started
[  186.459495] xenbr0: port 1(eth0) entered blocking state
[  186.461194] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Similar to before we will use the pre-defined examples in '/etc/xen/'

```
$ cd /etc/xen
```

```
# xl create -c example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU from the host running Para-Virtual networking:

```
ssh -p 2322 root@localhost
```

14.7 Starting Linux guests with Pass-through networking (PetaLinux 2017.1)

The difficulty with using pass through networking is that the steps above use Dom0 networking to load the DomU boot image onto the guest. This won't work with pass through networking as Dom0 never has any networking avaliable.
You will need to find a way to get the kernel and rootFS (the pre-built Image file) onto the guest. The steps below are used to get the Image file onto a SD card image and attach it to QEMU. Similar steps can be followed for hardware, excpet just copy the Image file to a formated SD card and insert it into the board.
Create and format the file we will be using on your host:

```
$ dd if=/dev/zero of=qemu_sd.img bs=128M count=1
$ mkfs.vfat -F 32 qemu_sd.img
```

Copy the Image file onto the card.
NOTE: We are using the pre-built Image which contains a kernel and rootFS. If you use the Image you built above then no rootFS is included. You will need to copy the rootFS onto the SD card and edit the Xen config file later to specify a rootFS.

```
$ mcopy -i qemu_sd.img ./pre-built/linux/images/Image ::/
```

Now boot QEMU with this extra option appened inside the --qemu-args: -drive file=qemu_sd.img,if=sd,format=raw,index=1
The full command should look something like this for your prebuilt images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./pre-built/linux/images/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

The full command should look something like this for your own images:

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -nic -net nic -net user,tftp=./images/linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22 -drive file=qemu_sd.img,if=sd,format=raw,index=1"
```

Then boot Dom0 following the steps above, with one difference. You will need to make sure that you tell Xen about the network passthrough. To do this you will need to edit the device tree. We are going to use u-boot to edit the device tree.
After loading the device tree to memory you will need to run this: fdt addr $fdt_addr && fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" && fdt set /amba/ethernet@ff0e0000 xen,passthrough "1"
The full command for booting prebuilt images you built is shown below:

```
$ tftpb 4000000 xen-qemu.dtb; fdt addr 4000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 xen-Image; tftpb 6000000 xen.ub; tftpb 0x1000000 xen-rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
```

The full command for booting images you built is shown below:

```
$ tftpb 4000000 system.dtb; fdt addr 4000000 &&&& fdt resize 128; fdt set /amba/ethernet@ff0e0000 status "disabled" &&&& fdt set /amba/ethernet@ff0e0000 xen,passthrough "1" &&&& tftpb 0x80000 Image; tftpb 6000000 xen.ub; tftpb 0x1000000 rootfs.cpio.gz.u-boot; bootm 6000000 0x1000000 4000000
```

NOTE: If running on hardware you will need to make a change to allow the DMA transactions. See here for more details: Passthrough Network Example[25]
Once you have logged onto the system mount the SD card and copy the image.

```
# mount /dev/mmcblk0 /mnt/
# cp /mnt/Image /boot/
```

Similar to before we will use another pre-defined examples in '/etc/xen/'

```
$ cd /etc/xen
```

```
# xl create -c example-passnet.cfg
```

25 https://xilinx-wiki.atlassian.net/wiki/display/A/Xen+Hypervisor+through+Yocto+Flow#x-Passthrough+Network+Example

15 Building the Xen Hypervisor with PetaLinux 2016.3

Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.

15.1 Boot Xen and dom0 on QEMU or Hardware

To run Xen you can either use the prebulit version shipped with PetaLinux or build it yourself using PetaLinx. See the section below for steps about buildling Xen yourself.

15.1.1 Configurng and building XEN from source using PetaLinux (Optional)

The default PetaLinux configuration has all the necessary options set by default. The XEN tools and XEN hypervisor will be built by default. So simply run:

```
$ petalinux-build
```

You have to manually copy the xen-qemu.dtb from the pre-built directory:

```
$ cp pre-built/linux/images/xen-qemu.dtb images/linux/
```

15.1.2 Run Xen dom0 on QEMU:

To use the Xen you built yourself run:

```
$ petalinux-boot --qemu --u-boot
```

To use the prebuilt Xen run:

```
$ petalinux-boot --qemu --prebuilt 2 --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=pre-
built/linux/images"
```

15.1.3 Run Xen dom0 on HW:

To use the Xen you built yourself run:

```
$ petalinux-boot --jtag --u-boot
```

To use the prebuilt Xen run:

```
$ petalinux-boot --jtag --prebuilt 2
```

You should eventually see something similar to this, when you do press any key to stop the autoboot.

```
Hit any key to stop autoboot:
```

If u-boot wasn't able to get an IP address from the DHCP server you may need to manually set the serverip (it's typically 10.0.2.2 for QEMU):

```
$ setenv serverip 10.0.2.2
```

Now to download and boot Xen, if running on QEMU, use xen-qemu.dtb otherwise use xen.dtb. Example:

```
$ tftpb 4000000 xen.dtb
$ tftpb 0x80000 Image
$ tftpb 6000000 xen.ub
$ bootm 6000000 - 4000000
```

Below is an example of what you will see.

```
U-Boot-PetaLinux> tftpb 4000000 xen.dtb
Using ethernet@ff0e0000 device
TFTP from server 10.10.70.101; our IP address is 10.10.70.1
Filename 'xen.dtb'.
Load address: 0x4000000
Loading: ##
         970.7 KiB/s
done
Bytes transferred = 28852 (70b4 hex)
U-Boot-PetaLinux> tftpb 0x80000 Image
Using ethernet@ff0e0000 device
TFTP from server 10.10.70.101; our IP address is 10.10.70.1
Filename 'Image'.
Load address: 0x80000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ######################
```

```
         6.9 MiB/s
done
Bytes transferred = 37524480 (23c9400 hex)
U-Boot-PetaLinux> tftpb 6000000 xen.ub
Using ethernet@ff0e0000 device
TFTP from server 10.10.70.101; our IP address is 10.10.70.1
Filename 'xen.ub'.
Load address: 0x6000000
Loading: #################################################################
         7.6 MiB/s
done
Bytes transferred = 788064 (c0660 hex)
U-Boot-PetaLinux> bootm 6000000 - 4000000
## Booting kernel from Legacy Image at 06000000 ...
   Image Name:
   Image Type:   AArch64 Linux Kernel Image (uncompressed)
   Data Size:    788000 Bytes = 769.5 KiB
   Load Address: 05000000
   Entry Point:  05000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 04000000
   Booting using the fdt blob at 0x4000000
   Loading Kernel Image ... OK
   Loading Device Tree to 0000000007ff5000, end 0000000007fff0b3 ... OK

Starting kernel ...



Starting QEMU as disk backend for dom0
/etc/rc5.d/S70xencommons: line 102: /usr/bin/qemu-system-i386: No such file or directory
Warning unable to detect baudrate, use 115200!
Running dynamic getty on hvc0/115200

PetaLinux 2016.3 Xilinx-ZCU102-DA7-ES2-2016_3 /dev/hvc0
Xilinx-ZCU102-DA7-ES2-2016_3 login:
```

15.2 Starting additional guests (Tested on QEMU with the 2016.3 ZCU102 BSP)

PetaLinux has no pre-built or automatic support for starting additional guests.
We'll go through the manual steps needed to do so.

First, we'll enable SSH. This is done by configuring the rootfs.

```
$ petalinux-config -c rootfs
```

We navigate to Filesystem Packages->console/network->dropbear and enable all of the options there.

Once enabled exit petalinux-config and ensure that the new selection is saved. Then rebuild PetaLinux

```
$ petalinux-build
```

If running on QEMU, we'll need to setup a port mapping for port 22 (SSH) in our VM.
In this example, we forward the hosts port 2222 to the VM's port 22.

```
$ petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=images/
linux,hostfwd=tcp:127.0.0.1:2222-22"
```

Once you hit the u-boot prompt, follow the steps in the earlier section on how to run Xen dom0.
When dom0 is done starting, we'll need to copy a guest Image into dom0's filesystem.
We'll simply reuse the PetaLinux Image as our domU guest.

If running on QEMU, we use scp's -P option to connect to our hosts port 2222 where QEMU will forward the connection to the guests port 22:

```
scp -P 2222 images/linux/Image root@localhost:/boot/
```

If running on HW:

```
scp images/linux/Image root@<board-ip>:/boot/
```

Now we need an xl configuration file that describes the VM. We'll create it in /etc/xen/example1.cfg. Here's a small example of the contents of such a file:

```
name = "guest0"
kernel = "/boot/Image"
extra = "console=hvc0 rdinit=/sbin/init"
memory = 256
vcpus = 2
```

Next step is to start the VM. This is done from a dom0 prompt:

```
xl create -c /etc/xen/example1.cfg
```

You'll see another instance of Linux booting up.
At any time you can leave the console of the guest and get back to dom0 by pressing **ctrl+]**.
You can list the guests from dom0:

```
xl list
```

To get back to the guests console:

```
xl console guest0
```

You can create further guests by for example running:

```
xl create example-simple.cfg name=\"guest1\"
xl create example-simple.cfg name=\"guest2\"
root@plnx_aarch64:/etc/xen# xl list
Name                                        ID   Mem VCPUs  State Time(s)
Domain-0                                     0   512     1     r-----      79.8
Domain-0                                     0   512     1     r-----      79.8
guest0                                       1   256     2     ------      93.7
guest1                                       2   256     2     ------      26.6
guest2                                       3   256     2     ------       1.8
```

To destroy a guest:

```
xl destroy guest0
```

15.3 Starting Linux guests with Para-Virtual networking

In this section we'll describe how to run Linux guests that use para-virtual network connections with dom0 and external networks.
First, we need to reconfigure the PetaLinux kernel.

```
petalinux-config -c kernel
```

Enable Device Drivers, Network devices, Xen backend network device. Save and exit.
Now, we'll reconfigure the rootfs.

```
petalinux-config -c rootfs
```

Enable dropbear as described in the previous section nr 2.
We also need to enable the bridge-utils. These are found under Filesystem Packages, net, bridge-utils.

This time we will run QEMU slightly different. We'll create two port mappings. One for dom0's SSH port and another for the Para-Virtual domU.
The default IP addresses assigned by QEMUs builtin DHCP server start from 10.0.2.15 and count upwards.
Dom0 will be assigned 10.0.2.15, the next guest 10.0.2.16 and so on.
So here's the command line that maps host port 2222 to dom0 port 22 and 2322 to domUs port 22.

```
petalinux-boot --qemu --u-boot --qemu-args "-net nic -net nic -net nic -net nic -net user,tftp=./images/
linux/,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22,hostfwd=tcp:127.0.0.1:2322-10.0.2.16:22"
```

Now, follow the instructions from section 1 on how to boot Xen dom0.
Once you are at the dom0 prompt, we'll need to setup the networking.
In this example, we will configure the guests to directly join the external network by means of a bridge.

First of all, we need to de-configure the default setup.
Kill the dhcp client for eth0:

```
killall -9 udhcpc
```

List and remove existing addresses from eth0:

```
ip addr show dev eth0
```

In our example the address is 10.0.2.15/24:

```
ip addr del 10.0.2.15/24 dev eth0
```

Then, create the bridge and start DHCP on it for dom0:

```
brctl addbr xenbr0
brctl addif xenbr0 eth0
/sbin/udhcpc -i xenbr0 -b
```

You should see something like the following:

```
udhcpc (v1.23.2) started
[ 146.546689] xenbr0: port 1(eth0) entered forwarding state
[ 146.550165] xenbr0: port 1(eth0) entered forwarding state
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
```

Now we need an xl config file describing the VM. The contents are very similar to the example in section 2, we only add one line.
In this example we create /etc/xen/example-pvnet.cfg in dom0's filesystem:

```
name = "guest0"
kernel = "/boot/Image"
extra = "console=hvc0 rdinit=/sbin/init"
memory = 256
vcpus = 2
vif = [ 'bridge=xenbr0' ]
```

Now we are good to go:

```
xl create -c /etc/xen/example-pvnet.cfg
```

You should see a new linux instance boot up.
Now we'll ssh into the domU running Para-Virtual networking:

```
ssh -p 2322 root@localhost
```

16 Linux - Xen Dom0

The pages describes the process for configuring Xen for Linux Dom0.

16.1 Table of Contents

16.2 General Build Information (Linux Dom0)

16.2.1 Linux Configurations Requirements

Linux needs to be built with the following options enabled:

```
CONFIG_XEN
CONFIG_HVC_DRIVER
CONFIG_HVC_XEN
```

These options are all enabled by default in PetaLinux projects.

16.2.2 PetaLinux rootfs Configuration Requirements

Petalinux rootfs options that need to be enabled (these are enabled by default):

```
CONFIG_ROOTFS_COMPONENT_APPS_NAME_XENTOOLS
CONFIG_ROOTFS_COMPONENT_APPS_NAME_XEN
CONFIG_ROOTFS_COMPONENT_APPS_NAME_GETTYBAUD
```

These options are all enabled by default in PetaLinux projects.

16.2.3 XEN Device Tree Requirements

The DTB file for XEN, should be the same as for plain Linux, with a few entries added to the chosen node. A working XEN DTB will be created by PetaLinux.
The xen,dom0-bootargs corresponds to the Linux Kernel command line.
Here is an example:

```
chosen {
    #address-cells = <0x1>;
    #size-cells = <0x1>;
    bootargs = "console=ttyPS0,115200 earlyprintk=cdns,uart,0xFF000000 rdinit=/sbin/init";
    xen,xen-bootargs = "console=dtuart dtuart=serial0 dom0_mem=256M bootscrub=0 timer_slop=0";
    xen,dom0-bootargs = "rdinit=/bin/sh console=hvc0 earlycon=xen earlyprintk=xen";
    dom0 {
        compatible = "xen,linux-zimage", "xen,multiboot-module";
        reg = <0x00080000 0x3100000>;
    };
};
```

If editing the DTB is required, convert the xen.dtb to .dts and back to dtb, using the following:

```
dtc -I dtb -O dts -o xen.dts xen.dtb
dtc -I dts -O dtb -o xen.dtb xen.dts
```

For more details on running the Xen Hypervisor, please refer http://dornerworks.com/services/xilinxxen

16.3 Additional Commercial Support and Professional Services

For more complex Dom0 topics such as the introduction of alternative non-Linux Dom0 OS, or optimizations in Dom0 Linux to improve solution boot time, footprint or performance; Xilinx recommends our Premier Partner DornerWorks[26].

16.4 Related Links

Petalinux Tools Reference Guide[27]
http://dornerworks.com/services/xilinxxen

26 http://www.dornerworks.com
27 http://www.xilinx.com/support/documentation/sw_manuals/petalinux2014_4/ug1144-petalinux-tools-reference-guide.pdf

17 Linux - Xen DomU

The pages describes the process for configuring Xen for Linux DomU.

17.1 Table of Contents

17.2 General Build Information (Linux DomU)

17.2.1 Linux Configurations Requirements

Linux needs to be built with the following options enabled:

```
CONFIG_XEN
CONFIG_HVC_DRIVER
CONFIG_HVC_XEN
```

These options are all enabled by default in PetaLinux projects. They can remain enabled without issue for native Linux systems running without Xen hypervisor.

CONFIG_XEN enables critical hooks for Xen interface from DomU
CONFIG_HVC_DRIVER includes a driver for the hypervisor (paravirtual) console within the guest. This avoids having to manually assign a dedicated serial console for use by the guest.

This Hypervisor console is an input/output console point to point between DomU and Dom0. It can be run over SSH, physical UART. When multiple DomU instantate a console, each is unique.

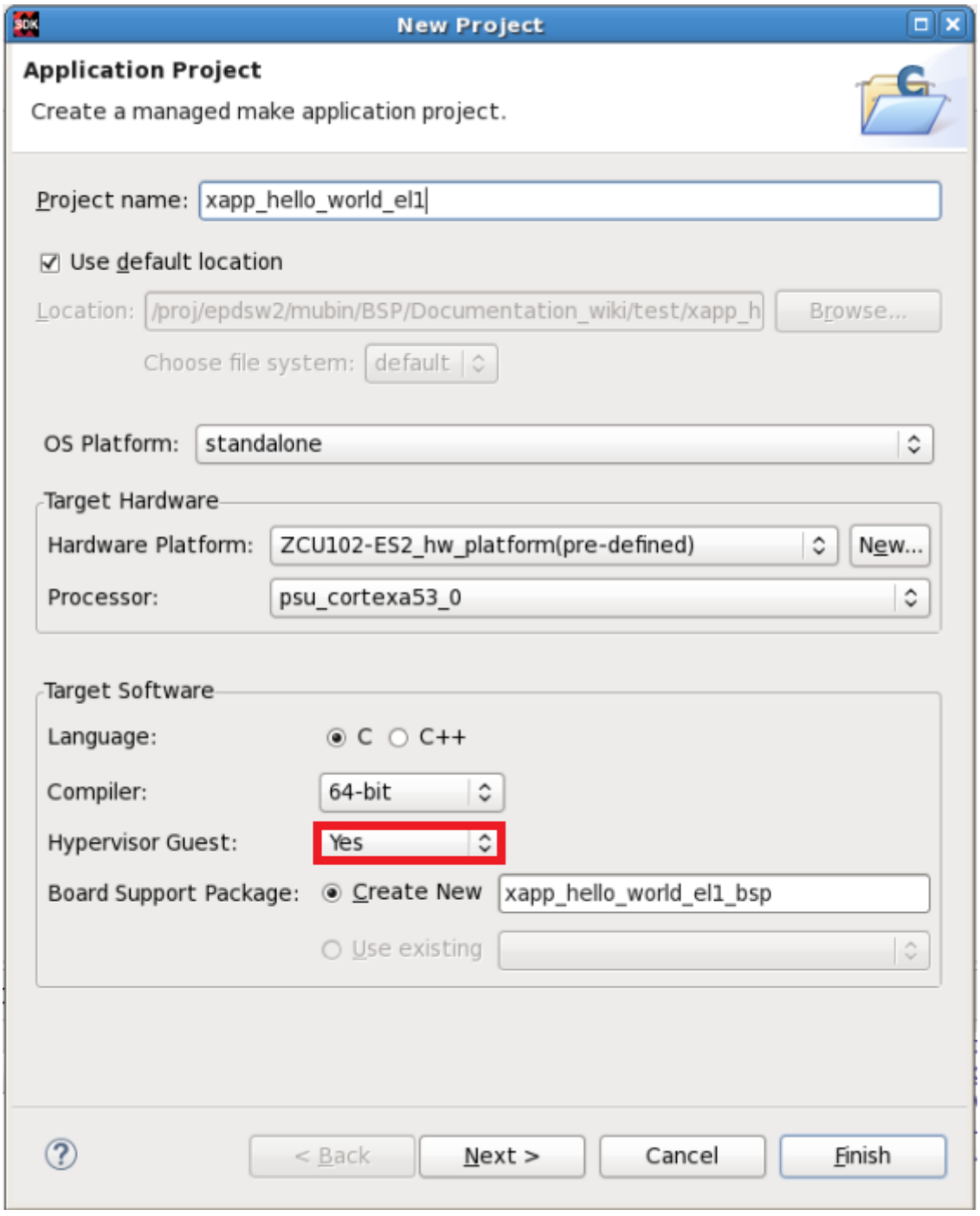
17.3 XEN EL1 Baremetal DomU

The purpose of this page is to describe the build and execution of the Hello World EL1 Baremetal application as a DomU on Zynq UltraScale+ MPSoC ,using PetaLinux and Xilinx SDK Tools.

17.3.1 Table of Contents

17.3.2 Build EL1 Baremetal Application Using SDK

- Create Hello World standalone application in SDK by selecting Hypervisor Guest as "yes", Refer image given below,



- Once application is created, please confirm following changes in application/BSP
  - BSP settings: stdin/stout has to pointed at psu_uart_1

    - Application linker file:Application has to be built with starting address: 0x40000000
- Build the application if settings mentioned above are configured correctly. Bin file will be created in the "<app_project>/Debug/" directory, after successful compilation of application.
- If Bin file is not generated automatically, execute this command from the SDK application project Debug directory: aarch64-none-elf-objcopy -O binary --gap-fill 0 <elf file name> <bin file name>

17.3.3 Changes Required in XEN DTS Using Petalinux

Before starting you need to create a PetaLinux project. It is assumed that a default PetaLinux reference design is used unchanged in these instructions.

- The peripheral devices used as part of EL1 Baremetal application need to be passthrough in xen-overlay.dtsi (<petalinux_project>/project-spec/meta-user/recipes-bsp/device-tree/files/xen-overlay.dtsi)
- For example , The "Hello world" standalone application uses uart1 to print messages, so passthrough the uart1 device in xen-overlay.dtsi, as shown below,
- ```
  &&uart1 {
     xen,passthrough = <0x1>;
  };
  ```
- Build the images for XEN with above change using the petalinux. Please refer section "Configuring and building XEN from source using PetaLinux 2017.1" in XEN Hypervisor(see page 10) for steps.

17.3.4 Xen Config file

- xEN needs xl configuration file, which describes the DomU guest.
- Here is a content of hello_world.cfg, an configuration file used to run Hello World application as an domU guest
- ```
  #Guest name
    name = "hello_world"
  # Kernel image to boot
    kernel = "xapp_hello_world_el1.bin"
  # Kernel command line options - Allocate 8MB
    memory = 8
  # Number of VCPUS
    vcpus = 1
  # Pin to CPU 0
    cpus = [1]
    irqs = [ 54 ]
    iomem = [ "0xff010,1" ]
  ```
- As shown in configuration file, bin file generated through SDK (i.e. xapp_hello_world_el1.bin generated through SDK in previous steps)
  has to be mentioned as kernel image in "kernel" field of xen guest configuration file. Also, "irqs"

and "iomem" fields are updated
with uart1 interrupt id and base address receptively. Similar way these fields in config needs to be updated for the other peripherals
as well, if application is using any.

- More information on config file can be found at
https://xenbits.xen.org/docs/unstable/man/xl.cfg.5.html

17.3.5 How to Execute an EL1 DomU Baremetal Application

- Copy the bin file xapp_hello_world_el1.bin generated in previous steps and hello_world.cfg to SD card or else use tftp to get them
in dom0's files system after booting dom0.
- Boot XEN dom0 with the images built in earlier steps. Refer section "TFTP Booting Xen and Dom0 2017.1" in XEN Hypervisor(see page 10), for dom0 boot up steps.
- Once XEN dom0 is up and running, copy the config file hello_world.cfg and bin file xapp_hello_world_el1.bin from SD card to dom0 file system/use tftp to get them in dom0 file system
- Go to that path in dom0 file system and Execute below command
    - **xl create hello_world.cfg**
- You'll see hello world baremetal application running as XEN domU , prints in bare metal application would be available on uart1 console
- More detail on xl command can be found at https://xenbits.xen.org/docs/unstable/man/xl.1.html

17.3.6 Related Links

18 Xen and PL Masters

The purpose of this page is to discuss the hardware and software requirements needed to allow PL Masters to operate within a Xen Hypervisor system configuration.

18.1 Table of Contents

18.2 1 Introduction

The Programmable Logic (PL) of the FPGA provides the flexibility to move data using AXI Masters such as DMA or custom IP. When Xen is running on ZynqMP the SMMU is used control data movement from the PL to any guest domains. This page describes the details needed to make an AXI Master in the PL work with a guest running in Xen. There is a focus on bare metal guests running on Xen with testing based on the 2017.1 release.

This information is based on early prototyping such that there are still some remaining questions and changes are likely. This page also assumes the user understands Xen and the process of passing through a device to a guest domain.

18.3 2 TBUs

The SMMU implements a TBU for sets of masters based on the PS port. Refer to the Zynq MPSOC TRM for more TBU details. The following table specifies the TBU number associated with each port of the Zynq MPSOC PS.

TBU Number	PS Port
0	S_AXI_HPC[0]_FPD
0	S_AXI_HPC[1]_FPD
3	S_AXI_HP[0]_FPD
4	S_AXI_HP[1]_FPD
4	S_AXI_HP[2]_FPD
5	S_AXI_HP[3]_FPD

18.4 3 Stream IDs

Many of the complexities of a PL Master revolve around stream IDs. The details are explained in the following sub-paragraphs.

18.4.1 3.1 AXI IDs and Master IDs

The AXI protocol includes two kinds of AXI transaction identifiers, AXI IDs and Master IDs. The Master ID is used to uniquely identify the master that initiated a transaction. AXI IDs are used to identify separate transactions that must be processed in order, for example when having multiple contexts or threads within a single master.

18.4.2 3.2 Derived Master IDs

On the ZynqMP, some masters have fixed Master IDs. Others have their Master IDs derived by combining a fixed part with AXI IDs generated from the master itself. This may for example allow blocks to differentiate DMA read channels from write channels just by looking at the AXI Master ID.

For transactions crossing from the PL into the PS, at each port a Master ID will be derived from a fixed part combined with the AXI ID. So for multiple PL masters, using the same PL to PS AXI port, the AXI ID is the part that uniquely identifies the PL masters.

For Masters in the PS, partially derived Master IDs are created immediately outside of the respective master. For the PL, this derivation happens at the AXI port at the PL to PS boundary. This has some side-effects that need to be considered.

When connecting multiple masters to slaves using an interconnect, depending on the implementation of the interconnect and its ability to handle out of order traffic, the interconnect may compress the AXI IDs to optimize away unneeded signals. Therefore, when instantiating interconnects in the PL, AXI IDs may be compressed by the interconnect and it may become difficult or even impossible to identify individual PL

masters once transactions reach the PL to PS port.

When a transaction reaches the SMMU, the SMMU will derive a ***Stream ID*** from the AXI transaction in an implementation specific way. On the ZynqMP this is done by combining the AXI Master ID with the TBU number.

18.4.3 3.3 PL Stream IDs

The Master ID for each AXI master and each port of the PS for the PL are listed in a table titled "Master IDs List" in chapter 16 of the Zynq MPSOC Technical Reference Manual. Each PS port to the PL has a fixed master ID that is used together with the TBU number and an AXI ID to create the stream ID.

TBU Number Bits	**Master ID Bits**	**AXI ID Bits**
14-10	9-6	5-0

The following example illustrates the stream ID of 0x200 for the first AXI master of the HPC0 port of the PS.

TBU Number Bits	**Master ID Bits**	**AXI ID Bits**
00000b	1000b	000000b

18.4.4 3.4 Stream IDs in the Device Tree

For Xen based systems the stream IDs of PL masters must be added to the Linux Dom0 device tree. Automated device tree generation does not generate the stream IDs for the PL masters so that the user must add them into the device tree manually. The device tree is used by both Xen and Linux as Dom0 and DomU guests. Linux 4.9 has moved to newer bindings while Xen has not as of 2017.1 and this can be confusing.

18.4.4.1 3.4.1 Xen Specific Device Tree Bindings

Xen uses old bindings which include ***#stream-id-cells*** at each AXI master node and ***mmu-masters*** at the SMMU node. These are required for any DMA device passed through by Xen, regardless of whether it’s for Linux or bare-metal guests.

Linux 4.9 has deprecated the ***mmu-masters*** property in the smmu node, and if it sees one, it will disable the SMMU driver. This is the reason that a native Linux device tree does not include the ***mmu-masters*** property in the smmu node.

The SMMU device node ***mmu-masters*** property contains the stream IDs for each AXI master. The device tree snippet below contains two CDMA IP cores (PL AXI masters) connected to HPC of the PS.

```
        &&smmu {
            status = "okay";
            mmu-masters = < &&gem0 0x874
                &&gem0 0x874
                    ...
                &&nand0 0x872
                &&axi_cdma_0 0x200
                &&axi_cdma_0 0x201
                >;
        };
```

18.4.4.2 3.4.2 Linux Specific Device Tree Bindings

The ***iommus*** property within a node is used by Linux natively (without Xen) as of 4.9. The ***iommus*** property is contained in each node that is an AXI master.
Xen does not know about the ***iommus*** property and requires the ***mmu-masters*** property to use the SMMU.

```
        gem0: ethernet@ff0b0000 {
                  ...
            #address-cells = <1>;
            #size-cells = <0>;
            #stream-id-cells = <1>;
            iommus = <&&smmu 0x874>;
        };
```

18.4.5 3.5 Multiple Stream IDs in the PL

When there are multiple masters in the PL, such as two CDMAs connected to an AXI Interconnect then connected to the HPC0 port of the PS, the AXI Interconnect creates the AXI IDs that map into the stream ID. At the current time there is not a clear way to determine the AXI ID of a specific master. It is also not clear yet if the AXI ID of a master might change when system changes are done in Vivado.

18.4.6 3.6 Determining the Stream ID of an AXI Master

The dtdev line of the Xen guest domain configuration file is used to setup the SMMU for the device that is being passed through to a guest. It is only required for AXI masters rather than AXI slaves. When this line is left out of the configuration file for an AXI master Xen will generate a fault when the AXI master performs a transfer as illustrated below.

```
root@plnx_aarch64:/mnt# (XEN) smmu: /amba/smmu@fd800000: Unexpected global fault, this could be serious
(XEN) smmu: /amba/smmu@fd800000: GFSR 0x80000002, GFSYNR0 0x00000000, GFSYNR1 0x00000201, GFSYNR2
0x00000000
```

The fault indicates an unidentified stream ID has been received by the SMMU as shown in the GFSR register. The GFSYNR1 register contains the stream ID that was unidentified. A stream ID of 0x201 was unidentified in the above fault. The description of the SMMU registers is contained in the ARM System Memory Management Unit Architecture Specification.

18.5 4 Coherent Transfers

There are only minor details that differ for Xen with coherency. As with a baremetal application without Xen, the following must be done for coherent transfers with an AXI master in the PL for Xen

1. The memory, typically DDR, must be outer shareable
2. Coherent transactions must be generated by the AXI Master by setting the AxCACHE signals to a coherent value, typically 0xF or 0xB
3. The AXI master must be connected to a coherent interface (port) on MPSOC such as HPC0 or HPC1

The CCI of the ZynqMP requires that snooping be enabled for the APU to support coherency. This is accomplished by ATF in a Linux based system and a Xen environment such that the bare metal application should not attempt to alter the CCI register as the register cannot be altered from EL1.

18.6 5 Non-Coherent Transfers

There is no difference for non-coherent transfers as the bare metal application is still responsible for cache operations.

18.7 6 Central Direct Memory Access (CDMA) Prototype Testing

The CDMA IP core was used extensively for testing with one or two instances. It provides an easy prototyping platform when used without scatter gather for simple transfers. The hardware system was configured for simple transfers such that each CDMA IP core only requires one AXI master. The goals of the prototyping were to show working AXI masters in the PL that are interfacing with baremetal applications running on top of Xen.

Note: For multiple CDMA IP cores the IP core should also be configured to use the store and forward feature to allow both IP cores to perform transfers simultaneously. Without this feature transfer hangs will be created when both cores transfer simultaneously. Larger transfers with CDMA were desired for testing. The CDMA has a maximum transfer length of 2 ^ 23 - 1 bytes.

18.7.1 6.1 Xilinx SDK Details

The SDK bare metal application built on top of a hypervisor only allocates 1 MB of DDR for the application. It can be safely increased in the linker script for the application making sure that the amount of memory is not more than specified in the Xen configuration file for the guest. The drivers in the SDK for the IP core along with the example applications were used.

18.7.2 6.2 Hardware System Details

For more than one master in the PL connected to a single PS port (HP, HPC) an AXI interconnect is required. There are now two AXI interconnect IP blocks in Vivado. A newer AXI interconnect called AXI SmartConnect is available while the older AXI Interconnect is still available. Vivado may use the AXI SmartConnect as the default interconnect with connection automation.

18.7.3 6.3 AXI SmartConnect

Do not use AXI SmartConnect if there is more than one master in the PL connected to a PS port (HP, HPC) while running with the SMMU in a Xen environment. This IP does not support AXI IDs that are needed for SMMU support and Xen. AXI SmartConnect has been tested with one PL master when AXI IDs are not an issue.

18.7.4 6.4 AXI Interconnect

The AXI Interconnect provides the required AXI IDs needed for using the SMMU in a Xen environment when configured properly. This is the preferred solution.

18.7.4.1 6.4.1 AXI Data Width

The AXI Interconnect incorporates data sizers when the data bus width varies between connected masters and slaves. The data sizers can remove IDs from the AXI interface. The best practice is to configure the master IP and the slave IP to have the same data width such that the AXI Interconnect is configured with only a crossbar connect without any data size changes.

18.7.5 6.5 AXI Transactions for the SMMU

The SMMU is designed to be used in a non-secure system such that AXI transactions generated by a master in the PL must be non-secure. The master IP may not allow the type of the transaction, secure or not, to be specified such that the AXI signals will need to be tied off manually. Security is specified in the AxPROT signals with a value of 0x2 being non-privileged, non-secure, and data. There are still some questions that remain around privileged vs non-privileged transactions and early prototyping has shown non-privileged to function correctly.

Note: non-secure transactions will not work for bare metal applications that not running on Xen. The Xilinx SDK bare metal (standalone) BSP executes at EL3 which drives this requirement. If you desire to create two different configurations, one running bare-metal as a Xen DomU, and the other running bare metal directly

on the hardware at EL3, (without Xen) then a GPIO can be used to control the AxPROT signals.

18.8 7 Checklist for AXI Masters

1. Using AXI Interconnect rather than AXI SmartConnect if multiple AXI masters connected to an AXI interconnect?
2. The AXI master generating unsecure transactions when running on Xen or secure transactions for baremetal without Xen?
3. The AXI interconnect configured for the same data width for the slaves and masters when multiple masters are connected to a single interconnect?
4. Using the correct stream ID in the device tree for Xen?
5. Is the dtdev line of the xen configuration correct referring the correct node?
6. Is the iomem line of the xen configuration correct for the address and range of the device being passed through?

19 Xen Hypervisor internals for Zynq UltraScale+

This page covers various topics on how Xen works internally when running on the ZynqMP.

19.1 Table of Contents

19.2 VM Memory Map

19.2.1 Without Xen

When running Operating Systems or bare-metal applications natively (without a hypervisor) on the ZynqMP, SW needs to target the native memory map of the ZynqMP as described in the TRM.
This includes adhering to the base addresses of each device, linking code towards the base addresses of memory (DDR, OCM or TCM) and using the interrupt numbers allocated and described in the ZynqMP TRM.

19.2.2 Guests on top of Xen - Dom0:

When running Operating Systems on top of Xen, things work a little bit differently.
Xen will create a memory map for every guest.

For Dom0, Xen will create a memory map that is essentially the same memory map as for the native host platform. I.e, on the ZynqMP, dom0 sees the same memory map as it would when running without Xen.
A few differences are that dom0 will not see any memory not allocated to it. E.g, memory used by Xen or by other guests.
Another special thing for dom0 is the fact that all devices and all memory gets mapped 1:1.

This means that a Guest Physical Address (IPA in ARM terminology) will map to the same Physical Address.

19.2.3 Guests on top of Xen - DomU:

For any additional guests (DomU), Xen will create an artificial memory map and interrupt map.
This Xen VM memory map will look the same on all ARMv8 platforms running a given version of Xen, thus making any OS or bare-metal program that runs on top of it portable to any ARMv8 platform that runs on top of Xen (excluding the use of device pass-through).

In 4.8, here are the important parts for us:

GICv2 Distributor and CPU interface:

GUEST_GICD_BASE xen_mk_ullong(0x03001000)
GUEST_GICC_BASE xen_mk_ullong(0x03002000)

2 banks of memory, low and high:
GUEST_RAM0_BASE 0x40000000 /* 3GB of low RAM @ 1GB */
GUEST_RAM0_SIZE 0xc0000000
GUEST_RAM1_BASE 0x0200000000 /* 1016GB of RAM @ 8GB */
GUEST_RAM1_SIZE 0xfe00000000

All the details of the Xen VM memory map are described here[28].

Users extend the Xen VM memory map by mapping in memory areas from the host platform using iomem lines in the VM configuration.

19.3 Share-ability and Memory Attributes

Xen uses the 2-stage MMU in ARMv8 cores to create the virtualized memory map used by guests.

The 2nd translation stage allows Xen to:

1. Isolate guests from each other
2. To create arbitrary memory maps that don't need to map 1:1 to physical memory
3. To control Share-ability and Memory attributes for memory regions

When ARMv8 cores process memory transactions through the MMU, page-table fields control whether memory is shared within the inner or outer domains and the semantics of the transaction in terms of strict ordering, gathering, early acking and so on. Both the stage-1 (the guests page-tables) and the stage-2 (Xens page-tables) can specify the share-ability and memory attributes.
When translation is done, the MMU will merge the stage-1 and stage-2 attributes (basically selecting the most "compatible" option).
Details on the rules for combining/merging the attributes can be found in the ARMv8 specs, version A.e, section D4.5.3 "Combining the stage1 and stage2 attributes..."

Examples:
If either stage specifies Outer shareable, the MMU will use Outer shareable.
If either stage specifies Device memory, the MMU will use Device memory.
If either stage specifies Non-Gathering, the MMU will use non-Gathering.
etc

In Xen 4.8, all mappings created for dom0 area created with the most relaxed settings, i.e Normal Memory, Cached and Outer Shareable. This essentially gives the dom0 guest full control over the attributes with its own stage-1 page-tables.

For domUs, Xen will map the vGIC as Device nGnRE and allocated memory as Normal Memory, Cached and Outer Shareable.
Any memory regions mapped in by the user (via iomem lines in the VM config) will be mapped as Device nGnRE.

---

28 https://xenbits.xen.org/gitweb/?p=xen.git;a=blob;f=xen/include/public/arch-arm.h;h=bd974fb13d1a319e9672e8015da91891874805e4;hb=HEAD

20 Xen Shared Memory

Memory can be shared between guests and between Dom0 and guests. This page describes the details needed for sharing memory.

20.1 Table of Contents

20.2 Introduction

The following description is one way to accomplish the shared memory and there may be others. The following technique was last tested with the 2017.1 Xilinx release of Petalinux. The following paragraphs describe the process by which some memory is taken from Linux in the device tree and then added as a UIO device in the device tree. It is assumed the user has basic device tree skills understanding how to make changes to existing nodes and adding new nodes together with a basic understand of Xen guest configuration files.

The memory is accessed in Dom0 and the guest as I/O memory rather than normal memory such that performance is not a primary concern. This is the only easy configuration that is supported by default with Xen.

20.3 Dom0 Device Tree Changes

20.3.1 Adding Shared Memory

The UIO driver in Linux is used to access the shared memory from Dom0. Note that the *iomem* statement in the Xen guest configuration file allocates memory in 4K pages so that a 4K minimum is used. The following device tree snippet illustrates how to add the shared memory node.

```
shared_mem {
#stream-id-cells = <0x1>;
compatible = "uio-dev";
reg = <0x0 0x7FFFF000 0x0 0x1000>;
};
```

20.3.2 Alter Linux Memory

Alter the amount of memory for Linux removing a 4K region which will be used by the UIO driver in Dom0. The following device tree snippet illustrates the memory with 4K less.

memory {
device_type = "memory";
reg = <0x0 0x0 0x0 **0x7FFFF000**>;
};

20.3.3 Linux Kernel Command Line

The command line of Linux is altered to allow the UIO driver to work with the shared memory node that was added. The following command line illustrates the addition of the UIO kernel module parameter initialization to allow the UIO driver to be compatible with the shared memory node in the device tree.

xen,dom0-bootargs = "console=hvc0 earlycon=xen earlyprintk=xen maxcpus=1 clk_ignore_unused **uio_pdrv_genirq.of_id=uio-dev**";

Note that the string "*uio-dev*" only has to match the compatible property of the shared memory node and could be any string.

20.4 Guest Configuration

The Xen guest configuration file is altered to use the shared memory as Device I/O memory. The following line illustrates the I/O mem line that allows the guest access to the shared memory together with the 2nd UART.

iomem = [**"0x7FFFF,1",** "0xFF010,1"]

More details can be found here:
https://xenbits.xen.org/docs/unstable/man/xl.cfg.5.html

The general syntax is:
iomem=["IOMEM_START,NUM_PAGES[@GFN]", "IOMEM_START,NUM_PAGES[@GFN]", ...]

For example, to map a page starting at physical address 0xFFFF0000 into a guest at address 0:
iomem = ["0xFFFF0@0x0,1"]

21 Xen on ARM: Share memory between guests

21.1 iomem and shared-memory

iomem is a xl config file option that can be used to map memory into a domU. Typically, it is used to map device memory into a domU as part of device assignment. However it can also be used to setup a cacheable shared memory region between dom0 and a domU. This section explains how to do that.

First, we have to add a reserved-memory node to the host device tree to advertise the special memory region to dom0, so that it won't use it to allocate memory as any other pages. For that, we can make use of the newly introduced "xen,shared-memory-v1" compatible string. For example:

```
    reserved-memory {
        #address-cells = <0x2>;
        #size-cells = <0x2>;
        ranges;

        xen-shmem@70000000 {
            compatible = "xen,shared-memory-v1";
            reg = <0x0 0x70000000 0x0 0x1000>;
        };
    };
```

This node tells dom0 that one page at 0x70000000 is to be use as reserved memory. Then, we need to do the same for DomU. We can do that by adding a device tree fragment to the DomU VM config file. The device tree fragment could be for example:

```
/dts-v1/;


/ {
    /* #*cells are here to keep DTC happy */
    #address-cells = <2>;
    #size-cells = <2>;


    passthrough {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;


        reserved-memory {
            #address-cells = <2>;
            #size-cells = <2>;
            ranges;


            xen-shmem@70000000 {
                compatible = "xen,shared-memory-v1";
                reg = <0x0 0x70000000 0x0 0x1000>;
            };
        };


        memory {
            device_type = "memory";
            reg = <0x0 0x70000000 0x0 0x1000>;
        };
    };
};
```

Similarly to the dom0 example, it tells the domU kernel that the page at 0x70000000 is to be used as reserved memory. Note that we also added the range to a regular memory node, because it is required by device tree that all reserved-memory ranges are also covered by the regular memory nodes. We add the device tree fragment to the DomU device tree using the device_tree option in the VM config file, the same way we use it for device assignment:

```
device_tree = "/root/snippet.dtb"
```

Finally, we only need to map the page into the DomU address space at the right address, which in this example is 0x70000000. We can do that with the iomem VM config option. It is possible to specify the cacheability of the mapping, "memory" means normal cacheable memory:

```
iomem = ["0x70000,1@0x70000,memory"]
```

In this example, we are asking to map one page at physical address 0x70000000 into the guest pseudo-physical address space at 0x70000000. We are also asking to make the mapping a normal cacheable memory mapping.

21.2 Cacheable Shared Memory between Linux and Bare-metal guests

Please refer to the following docs in the Xen tree:

> docs/man/xl-static-shm-configuration.pod.5[29]
> docs/man/xl.cfg.pod.5.in[30]

This feature allows sharing one or more pages in memory across VMs, simply by adding one config option to the VM config file. Memory can be shared as cacheable memory. Here is an example:

```
# Add following to the config file of VM1:
static_shm = ["id=ID1, begin=0x40000000, size=0x1000, role=owner"]

# Add following to the config file of VM2:
static_shm = ["id=ID1, begin=0x48000000, size=0x1000, role=borrower"]
```

In this example we are sharing one page owned by VM1 with VM2. The owner of the page is the "owner" VM (VM1), while the other VM is called "borrower" (VM2). The memory of Xen VMs always starts at 0x40000000 (guest physical address), so in this example we are sharing the first page of VM1 with VM2. Make sure to choose a valid allocated address of the owner VM.

0x48000000 is the address where the page will appear in VM2. Make sure to choose an free address in VM2. In this example, if we assigned 128M of RAM to VM2, then the shared page will be mapped right at the end of memory.

The shared page is advertised as "reserved-memory" in the guest device tree. This is a sample device tree node of a guest with a static_shm configuration:

```
reserved-memory {
                #address-cells = <0x2>;
                #size-cells = <0x2>;
                ranges;

                xen-shmem@40000000 {
                        compatible = "xen,shared-memory-v1";
                        id = "ID1";
                        reg = <0x0 0x40000000 0x0 0x1000>;
                };
```

The guest is free to use the reserved-memory region as it wishes. For instance, in the case of the Linux kernel, it is possible to export the memory as cacheable memory to userspace with a dmabuf driver. See this proof of concept of Linux driver and user-space program:

Xen-dmabuf Linux driver[31]

xen-dmabuf user space program[32]

29 https://github.com/Xilinx/xen/blob/master/docs/man/xl-static-shm-configuration.pod.5
30 https://github.com/Xilinx/xen/blob/master/docs/man/xl.cfg.5.pod.in
31 https://git.kernel.org/pub/scm/linux/kernel/git/sstabellini/xen.git/log/?h=xen-dmabuf
32 https://github.com/sstabellini/dmabuf-user/blob/master/dmabuf-us-test.c

22 Troubleshooting Xen issues

This page contains problems that may be encountered when using Xen, and their solutions.

If you encounter problems you don't see solutions for on this page, navigate to the Xilinx Developer Forums[33] for additional help.

22.1 Kernel call traces

When running Linux on top of Dom0, Linux can go in kernel panic due to insufficient memory for dom0. Example:

```
[   12.747244]  dump_backtrace+0x0/0x190
[   12.747750]  show_stack+0x18/0x30
[   12.748176]  dump_stack+0xd4/0x110
[   12.748530]  dump_header+0x48/0x1e8
[   12.748902]  out_of_memory+0x2c4/0x310
[   12.749289]  __alloc_pages_nodemask+0x72c/0xc64
```

22.1.1 Solution

Increase the dom0 allocated memory. Check xen_boot_tftp.source or xen_boot_sd.source relevant file and see “dom0_mem=1G”, increase this memory appropriately to > 1G. Save the file and recreate the xen_boot_tftp.scr using below command:

```
mkimage -A arm64 -T script -C none -a 0xC00000 -e 0xC00000 -d xen_boot_tftp.source xen_boot_tftp.scr
```

Reboot the machine with latest scripts.

22.2 Xen ethernet passthrough memory map failures

When creating a new DomU using example-passnet.cfg file, if DomU creation fails like below example:

```
xl create -c xl create -c /etc/xen/example-passnet.cfg
ERROR:
Parsing config from /etc/xen/example-passnet.cfg
(XEN) memory_map:fail: dom1 gfn=ff0e0 mfn=ff0e0 nr=1 ret:-38
```

22.2.1 Solution

Update the xen.dts file as per instruction from Starting Linux guests with Pass-through networking (see page 30)page and re-boot the machine(dom0) and then create DomUs.

33 https://forums.xilinx.com/

22.3 Xen ethernet passthrough interrupt mapping failures

When creating a new DomU using example-passnet.cfg file, if DomU creation parts fails like below:

```
root@xilinx-vck190-2021_1:/etc/xen# xl create -c example-passnet.cfg
Parsing config from example-passnet.cfg
(XEN) IRQ 95 is already used by domain 0
libxl: error: libxl_create.c:1637:domcreate_launch_dm: Domain 1:failed give domain access to irq 95: Device
or resource busy
libxl: error: libxl_domain.c:1182:libxl__destroy_domid: Domain 1:Non-existant domain
libxl: error: libxl_domain.c:1136:domain_destroy_callback: Domain 1:Unable to destroy guest
libxl: error: libxl_domain.c:1063:domain_destroy_cb: Domain 1:Destruction of domain failed
```

22.3.1 Solution

Correct the “irqs = [95]” in example-passnet.cfg with correct value. To find the correct value. convert xen.dtb into xen.dts and see the appropriate interrupt value for passthrough device.

23 Xen device passthrough examples

Passthrough allows you to give control of physical devices to guests domus. This page has a few examples for device passthroughs.

23.1 PSUART assignment

Turn xen.dtb into xen.dts:

```
dtc -I dtb -O dts xen.dtb > xen.dts
```

The, edit xen.dts by adding xen,passthrough; under the node of the device to assign, in this case serial@ff000000:

```
serial@ff010000 {
            compatible = "cdns,uart-r1p12", "xlnx,xuartps";
            status = "okay";
            interrupt-parent = <0xfde8>;
            interrupts = <0x0 0x16 0x4>;
            reg = <0x0 0xff010000 0x1000>;
            device_type = "serial";
            clock-names = "uart_clk", "pclk";
            power-domains = <0x26 0x22>;
            clocks = <0x1e 0x1f>;
            cts-override;
            port-number = <0x1>;
            xen,passthrough;
        };
```

Convert xen.dts back into xen.dtb:

```
dtc -I dts -O dtb xen.dts > xen.dtb
```

To enable the PSUART assignment to a domU. Below is one example for the dom create config file.

```
irqs = [ 54 ]
iomem = [ "0xff010,1" ]
device_tree = "/root/passthrough-serial.dtb"
```

In above config file, irqs and iomem entries are added. Here is detailed explanation for these entries:

23.1.1 IRQS calculation

To calculate irqs corresponding to the guest domain, it needs to find the following line in UART settings described in xen.dts file:

```
serial@ff010000 {
            compatible = "cdns,uart-r1p12", "xlnx,xuartps";
            status = "okay";
            interrupt-parent = <0xfde8>;
            interrupts = <0x0 0x16 0x4>;
```

take the interrupt which is device specific, in this case interrupt number is 0x16, add 32 which is the base address of interrupt handler.

0x16 = 22
irqs = 22 + 32 = 54

23.1.2 Setup IOMEM field

iomem = ["0xff010,1"]
1 page starting at "0xFF010000" base address of UART PS.

Creation of passthrough-serial.dts:

```
/dts-v1/;

/ {
    #address-cells = <0x2>;
    #size-cells = <0x1>;

    passthrough {
        compatible = "simple-bus";
        ranges;
        #address-cells = <0x2>;
        #size-cells = <0x1>;

                misc_clk1 {
                        #clock-cells = <0x0>;
                        clock-frequency = <0x7735940>;
                        compatible = "fixed-clock";
                        linux,phandle = <0x1f>;
                        phandle = <0x1f>;
                };

                misc_clk2 {
                        #clock-cells = <0x0>;
                        clock-frequency = <0x5f5b9f0>;
                        compatible = "fixed-clock";
                        linux,phandle = <0x1e>;
                        phandle = <0x1e>;
                };

        serial@ff010000 {
            compatible = "cdns,uart-r1p12", "xlnx,xuartps";
            status = "okay";
            interrupt-parent = <0xfde8>;
            interrupts = <0x0 0x16 0x4>;
            reg = <0x0 0xff010000 0x1000>;
            device_type = "serial";
            clock-names = "uart_clk", "pclk";
            power-domains = <0x26 0x22>;
            clocks = <0x1e 0x1f>;
            cts-override;
            port-number = <0x1>;
        };
    };
};
```

passthrough-serial.dts (see page 158)

Copy the above device tree code snippet and save it as passthrough-serial.dts or download the attached passthrough-serial.dts file. And convert the above dtc into dtb using dtc compiler:

```
dtc -I dts -O dtb -o passthrough-serial.dtb passthrough-serial.dts
```

Finally, create a new guest using ‘xl create -c configfile'.

Check Starting Linux guests with Pass-through networking(see page 30) for another passthrough example.

24 Booting XEN on ZCU102 using SD card

24.1 Table of Contents

24.2 Pre-requisites

- Build Xen image using bitbake for the target.
  - Example: MACHINE=zcu102-zynqmp bitbake xen-image-minimal
- Go to build/deploy/images/<target> and copy the below files to the boot partition of the SD card. Refer this guide to prepare your SD card.[34]
  - BOOT.bin
  - Image
  - system.dtb (rename Image-zynqmp-zcu102-revB.dtb to system.dtb)
  - xen.ub

Extract rootfs from xen-image-minimal-zcu102-zynqmp.cpio to the root partition of the SD card.

24.3 Booting from SD card

- Press any key when prompted to stop autoboot.
- Enter the below command and press enter

```
mmc dev $sdbootdev && mmcinfo && run uenvboot || run sdroot$sdbootdev; load mmc $sdbootdev:$partid $fdt_addr system.dtb && load mmc $sdbootdev:$partid 0x80000 Image; fdt addr $fdt_addr && fdt resize 128 && fdt set /chosen \#address-cells <1> && fdt set /chosen \#size-cells <1> && fdt mknod /chosen dom0 && fdt set /chosen/dom0 compatible "xen,linux-zimage" "xen,multiboot-module" && fdt set /chosen/dom0 reg <0x80000 0x3100000> && fdt set /chosen xen,xen-bootargs "console=dtuart dtuart=serial0 dom0_mem=768M bootscrub=0 maxcpus=1 timer_slop=0" && fdt set /chosen xen,dom0-bootargs "console=hvc0 earlycon=xen earlyprintk=xen maxcpus=1 clk_ignore_unused root=/dev/mmcblk0p2 rootwait"; load mmc $sdbootdev:$partid 1000000 xen.ub; bootm 1000000 - $fdt_addr
```

34 https://xilinx-wiki.atlassian.net/wiki/display/A/How+to+format+SD+card+for+SD+boot

```
rdf0428-zcu106-vcu-trd-2018-2
├── apu
│   ├── apps
│   └── vcu_petalinux_bsp
├── images
│   ├── hdmirx_vcu
│   ├── sdirx_vcu
│   ├── sdirx_vcu_sditx
│   └── vcu_trd
├── pl
│   ├── constrs
│   ├── pre-built
│   ├── Readme.txt
│   ├── scripts
│   └── srcs
└── README.txt
```

Related Links

- How to format SD card for SD boot[35]
- Xilinx Yocto[36]

35 https://xilinx-wiki.atlassian.net/wiki/display/A/How+to+format+SD+card+for+SD+boot
36 https://xilinx-wiki.atlassian.net/wiki/display/A/Yocto