Virtuosity™, Xen Zynq Distribution

User's Manual

Xilinx-XenZynq-DOC-0001 v1.07 August 21, 2019

Substituting `/dev/ttyUSB#` with the correct device found using the script above. For example:

```
$ sudo screen /dev/ttyUSB0 115200
```

2. If the U-boot environment provided with the XZD release is not being used, you will need to interrupt the startup sequence and enter the boot commands manually:

```
ZynqMP> setenv fdt_addr 0x280000
ZynqMP> setenv fdt_high 0xffffffffffffffff
ZynqMP> saveenv
ZynqMP> fatload mmc 0:1 $fdt_addr xen.dtb
ZynqMP> fdt addr $fdt_addr
ZynqMP> fdt resize 1024
ZynqMP> fatload mmc 0:1 0x300000 Image
ZynqMP> fdt set /chosen/dom0 reg <0 0x300000 0x$filesize>
ZynqMP> fatload mmc 0:1 0x80000 xen.ub
ZynqMP> bootm 0x80000 - $fdt_addr
```

For UltraZed, substitute `mmc 0:1` with `mmc 1:1`.

4.1.2. Booting via JTAG

As an alternative to booting the board with all the boot files on a second partition of the SD card, the board can be booted with using U-Boot and FSBL from the host PC. This section is an alternative to 4.1.1. As with the previous section, a minimum of a Class 10, SD card with a capacity of at least 8GB is recommended. See the target board's Overview document from Xilinx for a block diagram of the board to see where all the ports are.

1. *Set the Boot Mode to JTAG Boot.*
   The ZCU102 does this by setting DIP switch SW6 boot mode pins to 0b0000 (on, on, on, on).
   The UltraZed does this by setting DIP switch SW2 boot mode pins to 0b1111 (on, on, on, on).
2. *Connect the Ethernet port to your network and the USB UART and USB JTAG ports to your host machine.*
3. *Connect the power cord.*
4. *Populate your SD card. This only needs to be done when a change is made to the file system.*
   a. *Insert your SD card into your host machine.*
   b. *Figure out which device the SD card shows up as. It should be the last device that shows up.*

      ```
      $ dmesg
      ```

   c. *Mount the rootfs file system by following the process laid out in 3.1.1 steps 3 through 11*
   d. *Unmount SD card and place it back in the board.*
5. *Install the device tree blob for the board*

   ```
   $ cp $RELEASE_DIR/dist/images/linux/$BOARD/xen.dtb /tftpboot/xen.dtb
   ```

6. *In a new terminal, connect to the board UART, assuming the device is mounted to /dev/ttyUSB2*

   ```
   $ sudo screen /dev/ttyUSB2 115200
   ```

7. *Start the board using the power switch*
8. *In the other terminal, connect to the jtag, load boot images, and run them*

```
$ cd $RELEASE_DIR/dist
$ /opt/Xilinx/SDK/2017.3/bin/xsdb dist_zcu102_boot.tcl
```

9. *In the screen terminal, stop the U-Boot autoboot and set the following environment variables from the TFTP network values from section 2.4 and use an open IP address on your network for* `ipaddr`*:*

```
ZynqMP> setenv serverip  xxx.xx.xxx.xxx
ZynqMP> setenv gatewayip xxx.xx.xxx.xxx
ZynqMP> setenv netmask   xxx.xx.xxx.xxx
ZynqMP> setenv ipaddr    xxx.xx.xxx.xxx
ZynqMP> run xen
```

10. *Log into the system using root/root as the username and password.*

4.2. Running XZD

The following sections explain how to do some very basic domain management on Xen.

4.2.1. Booting a Guest

To test that Xen is running and display diagnostic information, use the 'xl info' command. The expected output is shown on the following page.

```
[root@xilinx-dom0 ~]# xl info
host                   : xilinx-dom0
release                : 4.4.0
version                : #3 SMP Wed Jun 15 12:01:36 EDT 2016
machine                : aarch64
nr_cpus                : 4
max_cpu_id             : 127
nr_nodes               : 1
cores_per_socket       : 1
threads_per_core       : 1
cpu_mhz                : 50
hw_caps                : 00000000:00000000:00000000:00000000:00000000:00000000:00000000:00000000
virt_caps              :
total_memory           : 4096
free_memory            : 3038
sharing_freed_memory   : 0
sharing_used_memory    : 0
outstanding_claims     : 0
free_cpus              : 0
xen_major              : 4
xen_minor              : 7
xen_extra              : .0-rc
xen_version            : 4.7.0-rc
xen_caps               : xen-3.0-aarch64 xen-3.0-armv7l
xen_scheduler          : credit
xen_pagesize           : 4096
platform_params        : virt_start=0x200000
xen_changeset          : Tue Feb 2 14:24:02 2016 -0500 git:398c245
xen_commandline        : console=dtuart dtuart=serial0 dom0_mem=512M bootscrub=0 dom0_vcpus_pin
maxcpus=3 timer_slop=0
cc_compiler            : aarch64-linux-gnu-gcc (crosstool-NG linaro-1.13.1-4.9-2014.09 -
cc_compile_by          : robertvanvossen
cc_compile_domain      :
cc_compile_date        : Wed Jun 15 12:00:58 EDT 2016
build_id               : ccd4c633e7094b77493d12feda5aff138fe5b677
xend_config_format     : 4
```

If Xen is not properly installed or running, you will receive an error similar to:

```
xl: command not found
```

Please re-extract the release image, and try again.

To view the list of running domains, use the 'xl list' command

```
# xl list

Output:
Name                                        ID   Mem VCPUs      State   Time(s)
dom0                                         0   512     1     r-----     133.4
```

The following necessary components for your first guest have already been created and stored in *dom0*'s file system:

Guest Linux Kernel, found at	/root/Dom1-Kernel
Guest File System Image, found at	/root/Dom1.img
Guest Domain Configuration, found at	/etc/xen/dom1.cfg

Staying in the terminal, we will prepare to run a guest domain. The additional domain is already included in the archive we have been working with and will now prepare to run it on the Xen hypervisor. To do that type the following commands in *dom0*'s console:

1. *Mount the guest's file system to a loop device in domain 0.*

```
# losetup /dev/loop0 /root/Dom1.img
```

You should receive no output if the command succeeds.

2. *Boot another domain (dom1), and connect to its console*

```
# xl create -c /etc/xen/dom1.cfg
```

The -c flag will automatically attach *dom1*'s console. That is, once this command is executed, you will be logging into *dom1*.

3. *Try the following at the dom1 command prompt*

Since your guest is not a privileged domain, typing 'xl info' will output less detailed information, and 'xl list' will generate an error as it can only be run in *dom0*.

```
[root@xilinx-dom1 ~]# xl info
host                   : xilinx-dom1
release                : 4.4.0
version                : #3 SMP Wed Jun 15 12:01:36 EDT 2016
machine                : aarch64
libxl: error: libxl.c:5183:libxl_get_physinfo: getting physinfo: Operation not permitted
libxl_physinfo failed.
libxl: error: libxl.c:5752:libxl_get_scheduler: getting current scheduler id: Operation not
permitted
get_scheduler sysctl failed.
xend_config_format     : 4
```

```
[root@xilinx-dom1 ~]# xl list

Output:
libxl: error: libxl.c:670:libxl_list_domain: getting domain info list: Operation not
permitted
libxl_list_domain failed.
```

The system is now running Xen and two domains or virtual machines: *dom0* and *dom1*.

If you want to return to *dom0*'s console while leaving the guest running, you may press CTRL-]. This will close the internal console connection, and bring *dom0* back into focus within the terminal window.

To reconnect to a guest terminal, use the "xl console" command

```
[root@xilinx-dom0 ~]# xl console dom1
```

It is important that you are aware which guest you are issuing commands to. Pay careful attention to the hostname listed in the terminal:

```
[root@xilinx-dom0 ~]#        dom0
[root@xilinx-dom1 ~]#        dom1
```

4.2.2. Copying a Guest

Both *dom0* (control) and *dom1* (guest) are included in the archive. You can easily boot a second guest domain, for a total of three domains including *dom0*, by making copies of *dom1*'s components.

1. *Make sure that domain 1 is powered down before we copy its kernel and file system.*

```
[root@xilinx-dom0 ~]# xl console dom1
[root@xilinx-dom1 ~]# poweroff
```

The guest will shutdown, and the system should return to domain0's console automatically.

Make sure that the guest is completely shutdown by using the 'xl list' command. Dom1 should **NOT** have an entry. If you do see an entry for *dom1*, then this means that *dom1* is still shutting down. Wait for approximately 15 or 20 seconds and try the command again. The results should appear similar to the below output.

```
[root@xilinx-dom0 ~]# xl list

Output:
Name                                        ID   Mem VCPUs State  Time(s)
dom0                                         0   512     1     r-----     259.7
```

2. *Copy the dom1 FS image.*

```
[root@xilinx-dom0 ~]# cp /root/Dom1.img /root/Dom2.img
```

This file is 1Gb, and will take a while to copy on the emulated SATA device.

3. *Copy the dom1 kernel.*

```
[root@xilinx-dom0 ~]# cp /root/Dom1-Kernel /root/Dom2-Kernel
```

4. *Copy the dom1 configuration file.*

```
[root@xilinx-dom0 ~]# cp /etc/xen/dom1.cfg /etc/xen/dom2.cfg
```

5. *Edit the new dom2 configuration file.*

You will need to:
a. Rename the guest to be named "dom2".
b. Configure the guest to boot domain 2's kernel.
c. Change the targeted loop device to allow two domains to run simultaneously.

You can accomplish this change using vi or sed expressions.

```
[root@xilinx-dom0 ~]# vi /etc/xen/dom2.cfg
```

or

```
[root@xilinx-dom0 ~]# sed -i 's/om1/om2/' /etc/xen/dom2.cfg
[root@xilinx-dom0 ~]# sed -i 's/loop0/loop1/' /etc/xen/dom2.cfg
```

6. Verify that the changes have been made to the appropriate files.

```
[root@xilinx-dom0 ~]# cat /etc/xen/dom2.cfg

# =====================================================================
# Example PV Linux guest configuration
# =====================================================================

...
```

```
# Guest name
name = "dom2"

...

# Kernel image to boot
kernel = "/root/Dom2-Kernel"

...

# Disk Devices
# A list of `diskspec' entries as described in
# docs/misc/xl-disk-configuration.txt
disk = [ 'phy:/dev/loop1,xvda,w' ]
```

7. Mount the guest file systems to their respective loop devices in domain 0. The losetup command creates a device on which we can mount the file systems created.

```
# If you have already mounted /root/Dom1.img to /dev/loop0,
# there is no need to mount it again
[root@xilinx-dom0 ~]# losetup /dev/loop0 /root/Dom1.img
# The dom2 file system hasn't been mounted yet:
[root@xilinx-dom0 ~]# losetup /dev/loop1 /root/Dom2.img
```

8. *Start the domains (or virtual machines) with the following commands.*

```
[root@xilinx-dom0 ~]# xl create /etc/xen/dom1.cfg
```

When you leave the -c flag off of the domain creation command, you will receive the guests initial boot messages to *dom0*'s standard out, while also keeping *dom0*'s console in focus.

This chatter does not affect your standard input however it does make it a bit hard to type the next command. You might want to wait before issuing the next command. The last message should look similar to the below output (the '3' in the vif3.0 output is variable).

```
...
xenbr0: port 2(vif3.0) entered forwarding state
xenbr0: port 2(vif3.0) entered forwarding state
xenbr0: port 2(vif3.0) entered forwarding state
```

Hit enter to bring up a new line in the console, indicating your hostname. Make sure you are still in *dom0*'s console, and that you didn't attach to the guest.

```
[root@xilinx-dom0 ~]#
[root@xilinx-dom0 ~]# xl create /etc/xen/dom2.cfg
```

You should see similar console chatter from dom2 booting as it reports to standard out.

The 'xl list' command will now show all three domains running. The ID values are sequential and will increase each time a domain is created. The ID numbers here might look different in your output.

```
[root@xilinx-dom0 ~]# xl list

Name                                        ID   Mem VCPUs      State   Time(s)
```

```
dom0                                         0   512     1     r-----     378.0
dom1                                         1   128     1     -b----      67.8
dom2                                         2   128     1     -b----      46.5
```

Guest domains should be shutdown carefully, as their file systems are easily corrupted if they are reset improperly or shutdown in the middle of an IO operation. There are two methods to shut down a guest:

From domain0, you can request the Xen Kernel to send a shutdown signal to a guest:

```
[root@xilinx-dom0 ~]# xl shutdown dom1
```

You could also attach to *dom1*'s console by executing the command "xl console dom1" and execute the poweroff command:

```
# To attach to dom1's console
[root@xilinx-dom0 ~]# xl console dom1
# While in dom1
[root@xilinx-dom1 ~]# poweroff
```

The poweroff command will function as expected within *dom0*'s console as well, but you should make sure that all domains are properly shutdown before doing so.

```
[root@xilinx-dom0 ~]# poweroff
```

4.2.3. Booting Guests with Alternate File Systems

Two alternate guest file systems and matching Xen configuration files have also been provided, these are the Ubuntu Core file system and the Linaro flavored OpenEmbedded file system. Their images can be mounted and the guests booted using commands similar to those found in 4.2.1:

For the Ubuntu Core FS:

1. *Mount the guest's file system to a loop device in domain 0.*
   ```
   # losetup /dev/loop1 /root/ubuntu-core-fs.img
   ```
2. *Boot another domain, and connect to its console*
   ```
   # xl create -c /etc/xen/ubuntu-core-fs.cfg
   ```

For the Linaro OpenEmbedded FS:

1. *Mount the guest's file system to a loop device in domain 0.*
   ```
   # losetup /dev/loop2 /root/linaro-openembedded-fs.img
   ```
2. *Boot another domain, and connect to its console*
   ```
   # xl create -c /etc/xen/linaro-openembedded-fs.cfg
   ```

You can perform all of the same operations with these guests as described in 4.2.1 and 4.2.2.

5.1. Environment Setup and Build Process

If you have not executed the steps in section 2.3, in a terminal, set RELEASE_DIR to the directory path where the archive was decompressed. This variable will be used in several instructions so that you may copy-paste them.

```
$ export RELEASE_DIR=`pwd`/Virtuosity_2019
```

5.2. Build Dom0 Linux Kernel, Xen, U-Boot, & FSBL

The following instructions assume that you are using the provided Yocto binaries.

TIP: *If you have not yet setup your host, please follow the steps in Chapter 2.*

1. Clone the Yocto source and create the default configuration file for the *dom0* FS.

```
$ cd $RELEASE_DIR
$ mkdir Xocto
$ cd Xocto
$ repo init -u git://github.com/dornerworks/xzd-yocto-manifests.git -m xzd.xml -b
Virtuosity_2019
$ repo sync
```

2. Setup the Yocto environment

```
$ source setupsdk
```

3. Configure Yocto to target the correct tool chain.
   Open *'conf/local.conf'* in an editor, then update Xilinx tool options to point at your SDK or Petalinux installation. Replace *{BOARD}* with the value of ${BOARD}.

```
MACHINE = “{BOARD}-zynqmp”
XILINX_VER_MAIN = “2017.3”
EXTERNAL_TOOLCHAIN_zynq = “/opt/Xilinx/SDK/2017.3/gnu/aarch32/lin/gcc-arm-linux-gnueabi”
EXTERNAL_TOOLCHAIN_microblaze =
“/opt/Xilinx/SDK/2017.3/gnu/microblaze/linux_toolchain/lin64_le”
EXTERNAL_TOOLCHAIN_aarch64 = “/opt/Xilinx/SDK/2017.3/gnu/aarch64/lin/aarch64-linux”
XILINX_SDK_TOOLCHAIN = “/opt/Xilinx/SDK/2017.3”
```

4. Optional: The default Linux kernel configuration will be sufficient to run as the *dom0* kernel for this exercise. You can use the following command to enable or disable kernel options as desired, but any configuration changes made will not be permanent unless added to the Yocto recipe.

```
$ bitbake -c menuconfig linux-xlnx
```

Added kernel configurations are beyond the scope of this document and are the responsibility of the user to understand their interactions and consequences if used.

5. Build the XZD image.

```
$ bitbake xzd-image-minimal
```

Depending on computer and network speeds this step can take over 2 hours. You may occasionally see a failure duiring a compile step, these can be caused intermittent connections and/or parallelzation issues on systems using multicore processors. In some cases it may be necessary to repeat this command 4 or 5 times before it will succeed.

6. View the files just created by Yocto in the 'Xocto/build/tmp/deploy/images/${BOARD}-zynqmp' directory

```
$ ls $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp
```

5.2.1. Customizing the Image

Refer to the Yocto Documentation for image customization.

http://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html

5.3. Installing and Using Built Images

1. Setup a two partition SD according to the instructions in section 3.1.1 steps 1 through 10.
2. Copy the files 'BOOT.bin', 'xen.ub', 'xen.dtb', 'uboot.env', and 'Image' to the fat partition of the SD card.

```
$ sudo cp $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp/boot.bin
/media/$USER/BOOT/
$ sudo cp $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp/xen.ub
/media/$USER/BOOT/
$ sudo cp $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp/xen.dtb
/media/$USER/BOOT/
$ sudo cp $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp/Image
/media/$USER/BOOT/
$ sudo cp $RELEASE_DIR/dist/images/linux/${BOARD}/uboot.env /media/$USER/BOOT/
```

3. Create the overlay directory and add any additional files into the locations you need them on the EXT4 partition. The example below will place a copy of the *dom0* Linux kernel generated by Yocto into to the root directory which can be used as a basis for any number of Linux domUs.

```
$ mkdir /media/$USER/overlayfs/overlay
$ mkdir /media/$USER/overlayfs/overlay/root
$ cp $RELEASE_DIR/Xocto/build/tmp/deploy/images/${BOARD}-zynqmp/Image
/media/$USER/rootfs/overlay/root/Dom1-Kernel
```

4. Unmount your SD card. Follow the steps in section 4.1.1 to boot your built system from the SD card
5. Once at the *dom0* prompt, you can use the following commands to start a domU.

Use a text editor to create a configuration file with the following contents for your domain in '/etc/xen/`:

```
# =====================================================================
# Example PV Linux guest configuration
```

```
# =====================================================================
#
# This is a fairly minimal example of what is required for a
# Paravirtualised Linux guest. For a more complete guide see xl.cfg(5)

# Guest name
name = "dom1"

# Kernel image to boot
kernel = "/root/Dom1-Kernel"

# Kernel command line options
extra = "console=hvc0 earlyprintk=xenboot root=/dev/loop0 rw"

# Initial memory allocation (MB)
memory = 512

# Number of VCPUS
vcpus = 1

# Network devices
# A list of 'vifspec' entries as described in
# docs/misc/xl-network-configuration.markdown
vif = [ 'bridge=xenbr0' ]
```

Now use this configuration file to start the new domain:

```
[root@xilinx-dom0 ~]# xl create /etc/xen/dom1.cfg -c
```

TIP: *There are times when the domain is created in a paused state. To correct this problem, enter the following commands below:*

```
[root@xilinx-dom0 ~]# xl unpause dom1
[root@xilinx-dom0 ~]# xl console dom1
```

6. Verify that the new domain is running.

```
[root@xilinx-dom0 ~]# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
dom0                                         0   512     1     r-----      36.7
dom1                                         1   128     1     -b----      10.3
```

5.4. Creating More Guests

The file systems and Linux kernels are built by Yocto in section 5.2. It is no longer necessary to build the guest domain file systems and guest domain kernel individually. Refer to section 5.2.1 for customization of the file system image.

If you would like to create an additional guest that has a different command prompt name, simply open 'conf/local.conf' in an editor, then add the option below with your desired <hostname>:

```
hostname_pn-base-files = "<hostname>”
```

Then re-run the build steps above in 5.2 step 5 to regenerate the file system with the new hostname.

Chapter 6 Xen on Zynq

6.1. Xen Boot Process

Running Xen requires booting two kernels: the Xen kernel and the *dom0* (or control) kernel, which at this point is a Linux kernel. Both kernels are loaded into the proper memory locations by the boot loader. Once the boot loader has finished the initialization of the system, it passes control of the boot process over to Xen.

Xen then performs some additional hardware initialization such as initializing the Zynq hardware so that Xen can map and handle requests from the device drivers used by the *dom0* kernel. More details on the Xen boot process can be found on the Xen Wiki (http://www.xenproject.org/help/wiki.html).

Once Xen performs its initialization, it then loads the *dom0* (usually Linux) kernel and the RAM disk into memory. The *dom0* kernel is then booted by Xen inside the privileged virtual machine domain; from the perspective of the kernel and the user this boot process is identical to Linux booting directly on the system hardware. Once *dom0* has booted, access to Xen can be configured for each unprivileged domain via the *dom0* interface to Xen. *dom0* has special privileges allowing it to perform this configuration, among them being the ability to access the system hardware directly. It also runs the Xen management toolstack, briefly described below.

6.2. xl – Interfacing to Xen

xl provides an interface to interact with Xen. It is the standard Xen project supported toolstack provided with the Xen hypervisor for virtual machine configuration, management, and debugging.

6.2.1. Listing Domains

'xl list' is used to list the running domains and their states on the system.

```
# xl list

Output:
Name                        ID   Mem VCPUs      State  Time(s)
Domain-0                     0  2048     2     r-----    32.0
dom1                         1  1024     2     r-----     7.3
```

6.2.2. Creating a Guest Domain

The following command will start a new domain using the provided configuration file argument. The name of the domain is determined by the value set in the configuration file.

```
# xl create -c /etc/xen/dom1.cfg
```

6.2.3. Shutting Down or Destroying a Guest Domain

'xl shutdown' is the command that should be used to shut down a running guest domain on the system. It performs a graceful shutdown of the domain using the built-in shutdown features of the domain's operating system, allowing the domains files system to close safely, and releasing any hardware resources (RAM, CPU Time, etc.) back to the system.

```
# xl shutdown dom1
```

The 'xl destroy' command should only be used as a last resort on unresponsive domains that are not removed when given the 'xl shutdown' command. The destroy command does not perform a graceful shutdown and potentially could corrupt the guest domain's file system as well as any I/O devices to which the domain has been given access.

```
# xl destroy dom1
```

6.2.4. Switching Between Domains

To access the command-line interface of a running domain, use the following command:

```
# xl console <domain-name>
```

where <domain-name> is the name of the specific domain of interest (the names of all running domains can be viewed using the `xl list` command detailed above).

To return to the *dom0* command-line interface, press the key combination: CTRL-].

6.3. xentop – Analyzing Domain Resource Utilization

xentop is part of the Xen tools and is located on the privileged domain, *dom0*. It can be used to analyze the performance of the various domains present on the system.

The default xentop display gives you some basic stats on the overall system such as number of domains, domain state, total memory, memory being used, CPU core count and speed, and Xen version number. It also provides some similar domain specific information like CPU usage and memory usage, network utilization (if using the default para-virtualized network bridge), and physical CPU (pCPU) usage per domain virtual CPU (vCPU).

To start xentop issue the following command in the *dom0* terminal

```
# xentop
```

xentop also accepts several command line arguments that can alter is behavior and the information that is displays. Those command line arguments are as follows:

-h, --help
Show the help message and exit

-V, --version
Show version information and exit

-d, --delay=SECONDS

Sets the number of seconds between updates (default 3)

-n, --networks
Displays network information in xentop console

-x, --vbds
Show virtual block device usage data

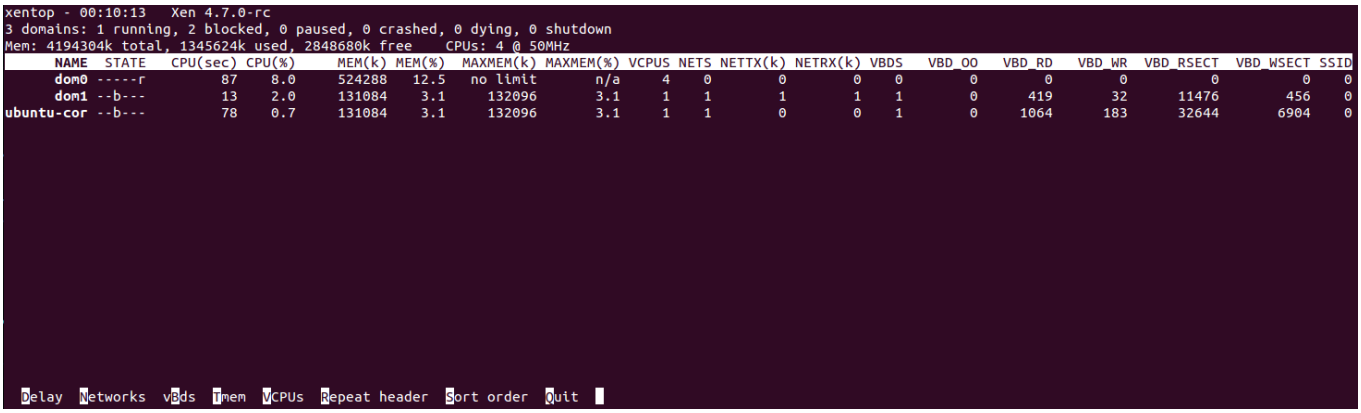
-r, --repeat-header
Repeat table header before each domain in the list

-v, --vcpus
Shows each vCPU's execution time in seconds below its assigned domain

-f, --full-name
Forces xentop to output the full domain name (default is truncated)

-b, --batch
Redirects xentop output data to stdout (batch mode)

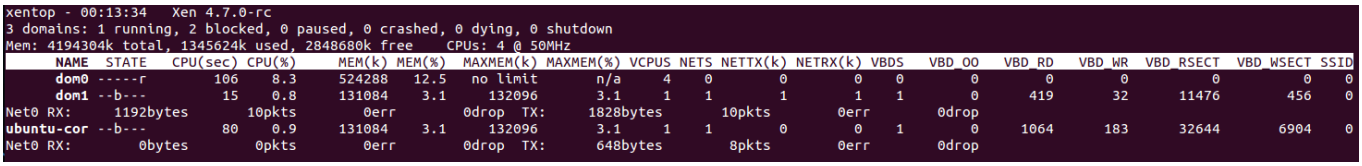
-i, --iterations=ITERATIONS
Sets the maximum number of metric updates that xentop should produce before ending
Following is an example of the xentop console with no additional commandline arguments:



In addition to the commandline arguments, the display of xentop can be changed while it is running by pressing the following letters (also displayed in the menu at the bottom of the xentop console):

D – Prompts you to set the delay, in seconds, between performance metric updates

N - Toggles display of network information for each domain. Example:



B – Toggles display of virtual block device information for each domain. Example:

```
xentop - 00:14:52   Xen 4.7.0-rc
3 domains: 1 running, 2 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 4194304k total, 1345624k used, 2848680k free    CPUs: 4 @ 50MHz
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VBD_WSECT SSID
      dom0 -----r        114    8.4     524288   12.5   no limit       n/a     4    0        0        0    0        0        0        0          0          0    0
      dom1 --b---         16    0.8     131084    3.1     132096       3.1     1    1        1        1    1        0      419       32      11476        456    0
VBD BlkBack 51712 [ca: 0]  OO:        0   RD:      419   WR:       32   RSECT:      11476   WSECT:        456
ubuntu-cor --b---         80    0.8     131084    3.1     132096       3.1     1    1        0        0    1        0     1064      183      32644       6904    0
VBD BlkBack 51712 [ca: 0]  OO:        0   RD:     1064   WR:      183   RSECT:      32644   WSECT:       6904
```

V - Toggles display of vCPU run time in seconds. Example of a domain with 2 vCPUs (0 and 1):

```
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS
      dom1 --b---         18    2.1     131084    3.1     132096       3.1     2
VCPUs(sec):    0:         10s  1:          8s
```

R - Toggles printing the table headers before each domain listing. Example:

```
xentop - 00:17:04   Xen 4.7.0-rc
3 domains: 1 running, 2 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 4194304k total, 1345624k used, 2848680k free    CPUs: 4 @ 50MHz
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VBD_WSECT SSID
      dom0 -----r        127    8.4     524288   12.5   no limit       n/a     4    0        0        0    0        0        0        0          0          0    0
VCPUs(sec):    0:        127s
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VBD_WSECT SSID
      dom1 --b---         17    2.5     131084    3.1     132096       3.1     1    1        1        1    1        0      419       32      11476        456    0
VCPUs(sec):    0:         17s
Net0 RX:     1192bytes       10pkts        0err        0drop  TX:     1828bytes       10pkts        0err        0drop
VBD BlkBack 51712 [ca: 0]  OO:        0   RD:      419   WR:       32   RSECT:      11476   WSECT:        456
      NAME  STATE    CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR  VBD_RSECT  VBD_WSECT SSID
ubuntu-cor --b---         81    1.3     131084    3.1     132096       3.1     1    1        0        0    1        0     1064      183      32644       6904    0
VCPUs(sec):    0:         81s
Net0 RX:        0bytes        0pkts        0err        0drop  TX:      648bytes        8pkts        0err        0drop
VBD BlkBack 51712 [ca: 0]  OO:        0   RD:     1064   WR:      183   RSECT:      32644   WSECT:       6904
```

S – Cycles through which table header to use for sorting (default is to sort by domain name)

Arrow Keys - Scroll through the domain display

Q, Esc – Quit xentop

6.4. Shared Memory

The Xen Project Hypervisor uses what it calls “grant tables” to provide a generic mechanism for memory sharing between domains. This shared memory interface is used to implement several Xen features, including PV (split) drivers for various I/O such as block devices and network interface cards.

Each domain has its own grant table. This is a data structure that is shared with Xen (via xenstore) and allows the domain to communicate to Xen what permissions other domains have on its memory pages. Entries in the grant table are identified by grant references. A grant reference acts as a “capability” which the grantee (client) can use to perform operations on the granter’s (server’s) memory.

This capability-based system allows shared-memory communications between unprivileged domains. A grant reference also encapsulates the details of a shared page, removing the need for a domain to know the real machine address of a page it is sharing.

6.4.1. libvchan

The virtual channel library or libvchan, included with Xen since version 4.2, is designed to make it easier to setup a shared memory ring between two guest domains. This implementation of shared memory uses Xen grant tables and event channels to provide options for both streaming-based communication and packet-based communication. Even if your application requires a different communication implementation than those provided, libvchan serves as a good example of how to setup a robust interdomain communications method.

As mentioned above, libvchan relies on grant table and event channel drivers. Therefore you will need to make sure you have a kernel in each guest domain with the appropriate xenstore, grant table and event channel drivers enabled including: gntdev, gntalloc and evtchn. All of these drivers are already enabled in the default Dom1-Kernel included with XZD.

Shared memory communication using libvchan involves a server that grants access and a client that makes use of the granted access. The server offers the memory used for communication, and advertises its grant references and event channel information via xenstore. The client must know the server's domain id and service path so that it can then obtain the server's information from the xenstore. Using this information, the client then maps the server's shared memory pages into an identically sized memory buffer in its own virtual address space. The two domains can now begin to communicate with one another.

The server and client each have their own separate dedicated ring buffer for writes. This avoids any race conditions during concurrent writes. The server reads from client's ring buffer and vice versa. The credentials for each ring is stored in a shared data control structure.

Shared memory size is based on page granularity. libvchan allows users to specify the size of the ring, which is bounded by the grant table size limit configured for each domain. The server and client can also be configured to have blocking or non-blocking behavior.

TIP: *More specific documentation can be found in the libvchan library code located here: https://github.com/dornerworks/xen/tree/dw-v2017.3/tools/libvchan. The example programs used in 6.4.2 are built by the Makefile in that folder as well.*

6.4.2. Example: Using libvchan for Inter-domain Communication

TIP: *This example assumes you have completed the steps in Chapter 3 and Chapter 4 and have booted either a QEMU setup or the setup described in Chapter 3 and Chapter 4. If you have not completed these steps, please do so before continuing.*

This section will walk you through setting up and using a basic program that utilizes libvchan to send character strings between two domains. This example makes use of several Xen libvchan example programs which are already included in the XZD filesystem.

To run the example boot into Xen, per section 4.1, and start up dom1 according the instructions in section 4.2.1. We will be using the program "vchan-node1" located in the /root/libvchan-example folder to allow dom1 to receive libvchan messages from dom0. Please use the following steps:

1. Open the dom1 console:

```
root@xilinx-dom0:~# xl console dom1
```

2. Change into the libvchan-example directory:

```
root@zcu102-zynqmp:~# cd /root/libvchan-example
```

3. Next, execute the vchan-node1 program in server mode, and tell it to read from dom0

```
root@zcu102-zynqmp:/root/libvchan-example# ./vchan-node1 server read 0 data/vchan
```

4. Use the keyboard shortcut *Ctrl +]* to return to the dom0 console, then enter the libvchan-example directory located there

```
root@xilinx-dom0:~# cd /root/libvchan-example
```

5. Look up the domain ID for dom1 using the following command

```
root@xilinx-dom0:/root/libvchan-example# xl domid dom1
```

6. We can now pass that ID as a commandline parameter to the vchan-node1 program in order to write a message to dom1. Fill in the domid reported in the previous step for ${domid} in the two locations in the command below

```
root@xilinx-dom0:/root/libvchan-example# ./vchan-node1 client write ${domid} \
 /local/domain/${domid}/data/vchan
```

7. Now we can type a message to dom1 and press return to send it

```
#Hello dom1, this is dom0!
#
```

8. Now press Ctrl + C to end the program and switch back to the dom1 console to verify the message was received in dom1

```
root@xilinx-dom0:/root/libvchan-example# xl console dom1
```

If everything worked correctly, you will the message you typed in dom0 has appeared on the dom1 console. Below is an example run through with all the associated output:

```
root@xilinx-dom0:~# xl create /etc/xen/dom1.cfg
Parsing config from /etc/xen/dom1.cfg
(XEN) Physical Timer Value (D1): 4086042332
[   33.173921] PLL: enable
(XEN) eemi: fn=19 No access to MMIO write fd1a0074
(XEN) eemi: fn=19 No access to MMIO write fd1a0074
[   33.179844] PLL: shutdown
[   33.344590] PLL: enable
(XEN) eemi: fn=19 No access to MMIO write fd1a0074
(XEN) eemi: fn=19 No access to MMIO write fd1a0074
[   33.350517] PLL: shutdown
[   33.419744] xenbr0: port 2(vif1.0) entered blocking state
[   33.419798] xenbr0: port 2(vif1.0) entered disabled state
[   33.425215] device vif1.0 entered promiscuous mode
[   33.431835] IPv6: ADDRCONF(NETDEV_UP): vif1.0: link is not ready
root@xilinx-dom0:~# (XEN) d1v0: vGICD: unhandled word write 0xffffffff to ICACTIVER0
[   36.915935] xen-blkback: backend/vbd/1/51712: using 1 queues, protocol 1 (arm-abi)
persistent grants
[   37.045678] vif vif-1-0 vif1.0: Guest Rx ready
[   37.045788] IPv6: ADDRCONF(NETDEV_CHANGE): vif1.0: link becomes ready
[   37.051176] xenbr0: port 2(vif1.0) entered blocking state
[   37.056534] xenbr0: port 2(vif1.0) entered forwarding state
(XEN) mm.c:1302:d0v0 gnttab_mark_dirty not implemented yet

root@xilinx-dom0:~# xl console dom1
[    0.000000] Booting Linux on physical CPU 0x0
.
```

```
.
.
[    4.363443] udevd[1429]: starting eudev-3.2

root@zcu102-zynqmp:~# modprobe xen-gntalloc
root@zcu102-zynqmp:~# modprobe xen-gntdev
root@zcu102-zynqmp:~# cd /root/libvchan-example/
root@zcu102-zynqmp:/root/libvchan-example# ./vchan-node1 server read 0 data/vchan
seed= 1507555061
##### Press Ctrl + ]
root@xilinx-dom0:~# xl domid dom1
1
root@xilinx-dom0:~# cd /root/libvchan-example
root@xilinx-dom0:/root/libvchan-example# ./vchan-node1 client write 1 \
/local/domain/1/data/vchan
seed=1507555418xl con
Hello dom1, this is dom0!
#^C
root@xilinx-dom0:/root/libvchan-example# xl console dom1
#Hello dom1, this is dom0!
^C
root@zcu102-zynqmp:/root/libvchan-example#
```

7.1. Introduction

In order to simplify the process of porting a "standalone" application, one that runs on bare metal without an operating system, to run as a Xen guest, DornerWorks has developed a Bare Metal Container (BMC). The BMC provides the libraries and tools for allowing bare metal applications developed using other tool flows, such as Xilinx's SDK (XSDK), to run as guests under Xen on the Xilinx Zynq UltraScale+ MPSoC. The container provides virtual memory mapping, stack, fault handling, and an API to print to Xen's console. After setting up the environment, the XZD Bare Metal container loads the payload application into virtual memory and then passes control to it.

7.1.1. Bare Metal Guest Bootup

dom0 starts up the bare metal guest using the Xen tools, xl toolstack. As part of initializing the guest, Xen allocates the configured amount of memory for it, and then copies the bare metal image to that memory space, using information provided in the image header. At that point, Xen starts execution at Exception Level 1 at offset into the newly allocated memory space indicated by the image header.

The bare metal guest enters into src/head.S and initializes the low level register settings that the guest will need to be set. Currently this includes setting up a direct guest physical address(GPA) to virtual address (VA) mapping of UART1 (1 page at 0xFF010000) as well as 1 page of potentially shared memory at 0x7FFFF000. This file should be modified to add additional MMU mappings if other memory regions are shared or I/O devices are passed through to the guest. This file should also be modified, along with src/main.c if the 4MB limit needs to be changed or a different application execution space is desired.

Next, the bare metal container branches to the `arch_init` function in src/setup.c. This is the first C code that gets executed. Here the physical offset is saved to a global variable so it can be used for any direct memory writes. Then a message gets printed on Xen's console to indicate the bare metal guest has booted to a point where it is about to transfer control to the payload application. This is done using the console_io hypercall. This call places a length and a character buffer in the correct registers and then interrupts into Xen. Xen then prints that buffer onto its own console.

After this, the bare metal container calls the `main` function in src/main.c. Here the bare metal guest loads the 4MB of the payload application to memory at 0x40400000 and then passes control to the payload application. This is where the location and size of the payload application can be changed as long as it remains consistent with src/head.S and the application's link map.

7.1.2. Payload Application

The bare metal container is designed to accept a payload application in the form of an ELF file, such as a standalone application generated by XSDK. The ELF file is converted to its binary equivalent and built with the bare metal container, creating a resulting binary image that can be run from *dom0*'s command line, by using the `./build_it` utility included in the repository.

Alternatively code can be modified in the src/setup.c and/or src/main.c to include the functionality directly. Application should begin at the `main` function. Any new C file can be added to the src directory and the Makefile will include it in the build without modifying any Makefiles or config files. The include directory is the only include directory, so that is where all header files should be placed.

Because of the nature of Xen, the bare metal guest only has access to what it is assigned to in the configuration file that is used to create the guest. There are a couple of options to get access to a device:

- **Direct Passthrough** - This gives a guest direct and usually exclusive access to a device. Information on Ethernet device passthrough can be found in section 9.2 *Passthrough*. The example provided at http://xzdforums.dornerworks.com/attachment.php?aid=2 includes passthough of UART1 to the guest(s). Note in this case since UART1 does not require interrupts or SMMU for proper use, it can be passed through to multiple guests. However, care must be taken to coordinate access of that device between the guests using it, or unspecified behavior could result.
- **Paravirtualized Devices** - *dom0* provides back-end drivers for some devices. To communicate with these drivers, front-end drivers need to be implemented for the bare metal application. This requires support for the XenStore, event channels, shared memory, and all of the related hypercalls. Support for Xen's virtual console has been added for FreeRTOS and bare metal guests.

7.2. Building the Bare Metal Guest

The following instructions assume you have followed Chapter 5 already.

7.2.1. Creating Payload Application

You are free to create your own payload application using whatever toolchain or workflow you like, as long as the resulting application meets these requirements:

- Runs at EL1 or EL0,
- Is contained in a single ELF file,
- Uses less than 4MB of memory, and
- Is linked to, or can otherwise run at, address 0x40400000.

The first requirement is to ensure the guest runs at EL lower than Xen, which runs at EL2.

The second requirement can be mitigated with changes to the `build_it` script.

The last two requirements can be increased or altered with changes to the BMC source code.

An example application is provided in the XZD that can be built using Xilinx's SDK (XSDK). Use the correct $PLATFORM corresponding to your board. For the ZCU102 use *ZCU102_hw_platform* and for the Ultrazed use *zed_hw_platform*. Create the payload application by following the following steps:

1. Create a `Hello World` application in XSDK targeting the $PLATFORM$ called $PROJECT_NAME in the $XSDK_WORKSPACE. This should create three projects: the application project called $PROJECT_NAME, a BSP project called $PROJECT_NAME"_bsp", and a hardware platform project called $PLATFORM.

```
$ source /opt/Xilinx/SDK/2017.3/settings64.sh
$ xsdk
```

2. Copy the files from the XZD to overwrite those in the XSDK workspace:

```
$ cp $RELEASE_DIR/misc/examples/baremetal/app/src/* $XSDK_WORKSPACE/$PROJECT_NAME/src/
$ cp $RELEASE_DIR/misc/xzd_bmc/xzd_bmc.h  $XSDK_WORKSPACE/$PROJECT_NAME/src/
$ cp -r $RELEASE_DIR/misc/examples/baremetal/bsp/* $XSDK_WORKSPACE/$PROJECT_NAME"_bsp"/
```

3. Clean and rebuild the XSDK project.

7.2.2. Building the Guest Image

To build the bare metal image, run the following command, targeting the ELF file you want to be your payload application:

```
$ source /opt/Xilinx/petalinux-v2017.3-final/settings.sh
$ cd $RELEASE_DIR/misc/xzd_bmc
$ ./build_it $XSDK_WORKSPACE/$PROJECT_NAME/Debug/$PROJECT_NAME.elf
```

Example: if you created your XSDK workspace called "workspace" in your home directory, and named your XSDK project "hello_world", then the resulting command would be:

```
$ ./build_it ~/workspace/hello_world/Debug/hello_world.elf
```

This generates an `xzd_bare.elf` file and an `xzd_bare.img` binary image. The binary image is the file that will be used as the `kernel` for the Xen guest. The `xzd_bare.elf` file can be used for debugging the bare metal container portion and $PROJECT_NAME.elf can be used to debug the payload application.

7.3. Installing and Running the Guest Image in the XZD

7.3.1. Guest Image

The `xzd_bare.img` file needs to be transferred to *dom0*'s file system, typically by adding it to the file system or by transferring it using TFTP or other network transfer protocol.

SD Card File System

This method is to simply mount the file system found on your SD card and copy the `xzd_bare.img` file directly to it. This is the easiest method if you already have your File System populated to its own partition on an SD card.

1. Insert SD card.

a. Mount SD partition if it does not automatically do so.

2. Copy files to the SD partition.

```
$ sudo cp xzd_bare.img $YOUR_MEDIA_MOUNT_PATH/root/.
```

3. Gracefully eject SD card using your host system's recommended method.

```
$ umount $YOUR_MEDIA_MOUNT_PATH
```

Network Transfer

This method is to use network transfer protocol, such as TFTP, to move the `xzd_bare.img` file to the target. Copy xzd_bare.img to the TFTP server's base directory.

```
$ cp xzd_bare.img /tftpboot
```

1. From *dom0*, issue the command to retrieve the file.

```
$ tftp -g -r xzd_bare.img <HOST_IP>
```

7.3.2. Guest Configuration

Create a configuration file for the bare metal guest. An example Configuration File (/etc/xen/bare.cfg), which passes UART1 through to the guest, is shown below:

```
name = "bare"
kernel = "/root/xzd_bare.img"
memory = 8
vcpus = 1
iomem = ["0xff010,1"]
```

7.3.3. Running the Guest

Boot up the XZD and in *dom0* start up the bare metal guest with the following command:

```
$ xl create /etc/xen/bare.cfg
```

Terminate the bare metal guest with the following command:

```
$ xl destroy bare
```

7.4. XSDK Example

For the XSDK example, additional steps are needed to create a region of RAM for guests to share in the device tree. An easy way to do this is to add a reserved-memory node to the end of the AMBA node definition in the DTS file appropriate for your target (e.g., xen-zcu102.dts):

```
+               reserved-memory {
+                       #address-cells = <2>;
+                       #size-cells = <2>;
+                       ranges;
+                       guest_shared: guest@7ffff000{
+                               reg = <0 0x7ffff000 0 0x1000>;
+                               no-map;
```

```
+                        };
+                };
```

To recompile the dts to dtb:

```
$ sudo apt install device-tree-compiler # if necessary
$ dtc -I dts -O dtb -o xen.dtb xen-${BOARD}.dts
```

An example of the DTB file can be found in `$RELEASE_DIR/misc/examples/baremetal/config/`. The resulting xen.dtb will need to be added to the SD card or /tftpboot directory depending on your target and method of booting it; see Chapter 3 Target Setup for more details.

To run multiple guests, the guest name needs to be unique. Furthermore, it is possible to pin guests to specific CPU cores with the `cpu` attribute. These examples assume that the `xzd_bare.img` file has been renamed to `bm.img`. A prebuilt version of `bm.img` can be found in `$RELEASE_DIR/misc`.

bm0.cfg:

```
name = "bm0"
kernel = "/root/bm.img"
memory = 8
vcpus = 1
cpus = [0]
iomem = [ "0x7ffff,1", "0xff010,1"]
```

bm1.cfg:

```
name = "bm1"
kernel = "/root/bm.img"
memory = 8
vcpus = 1
cpus = [1]
iomem = [ "0x7ffff,1", "0xff010,1"]
```

bm2.cfg:

```
name = "bm2"
kernel = "/root/bm.img"
memory = 8
vcpus = 1
cpus = [2]
iomem = [ "0x7ffff,1", "0xff010,1"]
```

bm3.cfg:

```
name = "bm3"
kernel = "/root/bm.img"
memory = 8
vcpus = 1
cpus = [3]
iomem = [ "0x7ffff,1", "0xff010,1"]
```

All four guests can be brought up from *dom0*'s command line:

```
$ xl create bm0.cfg
$ xl create bm1.cfg
$ xl create bm2.cfg
$ xl create bm3.cfg
```

Examples of these configuration files can be found in `$RELEASE_DIR/misc/examples/baremetal/config/`.

Please note that whenever multiple guests have access to the same resource, which could be shared memory or an I/O peripheral device, some method should be used to ensure proper coordination between the different threads of execution. The example XSDK project contains code for a spin lock to provide mutual exclusion for both the shared memory and the UART.

Chapter 8 Other Guests

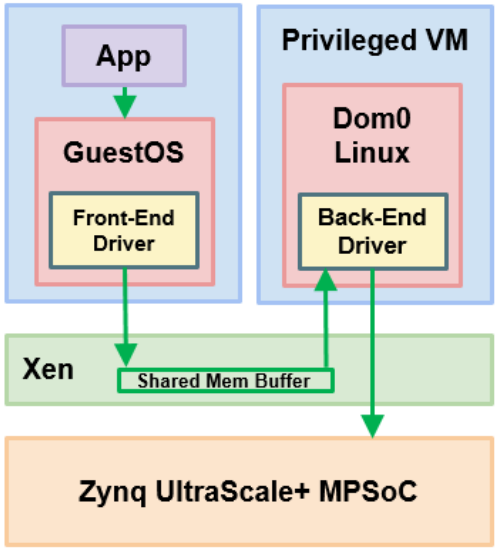
DornerWorks is constantly working on getting new operating systems to run as guests on Xen.

In addition to Linux and bare metal guests, the XZD also provides libraries for making guests using FreeRTOS run on Xen. The latest documentation can be found in the distribution at $(RELEASE_DIR)/docs/XZD FreeRTOS Guest Guide.pdf.

Chapter 9 Interacting with I/O Devices

9.1. Paravirtualization

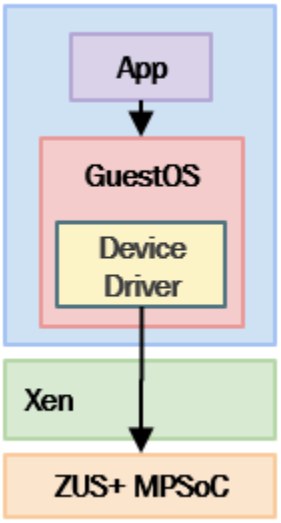
Typically, *dom0* is a full featured OS like Linux, and provides a plethora of device drivers. Xen takes advantage of the availability of drivers by providing a means for guests to use those drivers through *dom0*. This requires modification of the drivers in the guest, and is an example of paravirtualization. I/O Paravirtualization uses software to share a device from a privileged guest, typically *dom0*, to any other guest that needs to access the device. The privileged guest is the only one that has direct access to the device and contains the normal device driver to interact with the device. Then what Xen calls a split driver is used to share the data from the privileged guest to the other guests. A split driver is made up of a backend driver in the privileged guest and a frontend driver in the other guests that want to access the device. The backend driver sets up a shared ring buffer, and an event channel (a notification) for each guest that needs to access the device. The frontend driver in each guest then connects via a wrapper Application Program Interface (API) to those sharing mechanisms.



Since the privileged guest arbitrates access to the device, the data from the device can be shared across virtual machines without breaking partitioning. This is useful if multiple guests need to access the same I/O channel. Another advantage is that the frontend driver presents an abstraction of the specific device, so that guests can be more generic and thus more portable. This can be an initial drawback, because if the guest OS does not support that frontend driver, it needs to be developed. Paravirtualization also adds another layer to the device driver stack, therefore the performance will not be as fast as native OS usage of the device. If multiple guests are sharing the same device the privileged guest must implement an allocation scheme to prevent a guest from monopolizing that device. Since this method takes advantage of the strict memory sharing infrastructure of Xen, it is a safe and secure method for handling I/O.

9.2. Passthrough

Xen has the capability to pass access to peripheral devices through to guest domains, allowing that guest direct and unfettered access. For peripherals requiring high performance, Xen configures the system memory management unit (SMMU) to handle the necessary translations for any direct memory address (DMA) transactions initiated by that peripheral as well as configures the Generic Interrupt Controller (GIC) to pass interrupts to the guest domain. This allows the guest to use the device as if it was the only software running in the system. This passthrough capability allows for practically native performance and increases overall flexibility and stability in the system as the SMMU enforces the memory mapping, preventing a guest from using its DMA-capable peripheral to access data in another guest's memory space.



Section 9.2.2 provides a link to a guide for passing a UART device to a guest, while section 9.2.3 provides instructions for configuring the Xen system installed on the Xilinx UltraScale+ MPSoC (MPSoC) to pass an Ethernet peripheral device through to a guest.

9.2.1. UART Passthrough

For instructions for passing a UART device to a guest domain, *dom1*, see the guide in the distribution at $(RELEASE_DIR)/docs/XZD UART Pass Through How To Guide.

9.2.2. Ethernet Passthrough

The instructions documented here are specific for the Ethernet device, however can be generalized to pass through any DMA-capable device via the SMMU.

It is assumed that the reader has an Ubuntu 14.04 or 16.04 host system and has downloaded the release image mentioned in Chapter 2 Host Setup.

9.2.2.1. Modifying the Xen Device Tree

The first step in passing through an Ethernet device to a guest domain is to edit the xen.dts file located on the host computer in the XZD development system. The general process for enabling device passthrough is to disable the device from access to *dom0* and to enable it for passthrough to a guest domain. We will disable the GEM3 Ethernet controller and add an attribute that will allow it to be passed through. This will disable network connectivity for *dom0*.

We need to convert the xen.dtb into a xen.dts file, which is a text file that we can edit. You can start with $RELEASE_DIR/dts/xen-${BOARD}.dts. You can also use the the `dtc` command to create the dts file from the dtb:

```
$ dtc -I dtb -O dts -o $RELEASE_DIR/xen.dts /tftpboot/xen.dtb
```

Open the xen.dts file and find the fourth Ethernet controller located at address 0xFF0E0000. The text found in the xen.dts should look similar to that below.:

```
            ethernet@ff0e0000 {
                  compatible = "cdns,zynqmp-gem";
                  status = "okay";
                  interrupt-parent = <0x4>;
                  interrupts = <0x0 0x3f 0x4 0x0 0x3f 0x4>;
                  reg = <0x0 0xff0e0000 0x0 0x1000>;
                  clock-names = "pclk", "hclk", "tx_clk", "rx_clk";
                  #address-cells = <0x1>;
                  #size-cells = <0x0>;
                  #stream-id-cells = <0x1>;
                  iommus = <0x8 0x877>;
                  power-domains = <0x10>;
                  clocks = <0x3 0x1f 0x3 0x34 0x3 0x30 0x3 0x34>;
                  phy-handle = <0x11>;
                  phy-mode = "rgmii-id";
                  pinctrl-names = "default";
                  pinctrl-0 = <0x12>;
                  linux,phandle = <0x2a>;    ethernet@ff0c0000 {

                  phy@c {
                        reg = <0xc>;
```

```
                ti,rx-internal-delay = <0x8>;
                ti,tx-internal-delay = <0xa>;
                ti,fifo-depth = <0x1>;
                ti,rxctrl-strap-worka;
                linux,phandle = <0x11>;
                phandle = <0x11>;
        };
};
```

Once the Ethernet controller is located , edit the xen.dts to change status to "disabled" and add the attribute "xen,passthrough = <0x1>; ", resulting in something similar to the following:.

```
        ethernet@ff0e0000 {
                compatible = "cdns,zynqmp-gem";
                status = "disabled";
                xen,passthrough = <0x1>;
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x3f 0x4 0x0 0x3f 0x4>;
                reg = <0x0 0xff0e0000 0x0 0x1000>;
                clock-names = "pclk", "hclk", "tx_clk", "rx_clk";
                #address-cells = <0x1>;
                #size-cells = <0x0>;
                #stream-id-cells = <0x1>;
                iommus = <0x8 0x877>;
                power-domains = <0x10>;
                clocks = <0x3 0x1f 0x3 0x34 0x3 0x30 0x3 0x34>;
                phy-handle = <0x11>;
                phy-mode = "rgmii-id";
                pinctrl-names = "default";
                pinctrl-0 = <0x12>;
                linux,phandle = <0x2a>;

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

If you have built from source using Chapter 5, an example of the above changes can be seen by decompiling the 'Image-xen-zcu102-enet_pt.dts' file in your '$RELEASE_DIR/Xocto/build/tmp/deploy/images/zcu102-zynqmp/' folder. After the dts has been modified, we need to compile the dts back into the dtb and ensure that it is the saved in the correct location depending on your booting methods (/tftpboot/ or on the BOOT partition of the SD card). Enter the following command to perform this task:

```
$ dtc -I dts -O dtb -o /tftpboot/xen.dtb $RELEASE_DIR/xen.dts
```

or

```
$ dtc -I dts -O dtb -o /media/$USER/BOOT/xen.dtb $RELEASE_DIR/xen.dts
```

9.2.2.2. Modifying the Domain Configuration File

The next two files that require modification are done on the *dom0* file system from the XZD development system. Locate the configuration file for the guest domain that will be using the passed-through Ethernet device. Edit your dom1.cfg, located in /etc/xen, by adding the text below to the bottom of the file. In addition to adding the text below, make sure that you either comment out or delete the virtual device at approximately line 39. This line begins with 'vif'.

```
# Ethernet Device
# This will set up the Ethernet so that it is
# accessible to this guest domain.
dtdev = [ "/amba/ethernet@ff0e0000" ]
device_tree = "/etc/xen/xen-partial.dtb"
irqs = [ 95 ]
iomem = [ "0xff0e0,1" ]
```

Figure 1: Editing the guest domain configuration file

The lines above add gem3 exclusively to the guest domain, in this case, domain 1. The options needed for passthrough are defined below:

dtdev : The absolute path of the device to passthrough in the device tree
device_tree : *dom0* path to partial device tree to be passed to the guest
irqs : IRQ number to be given to the guest
iomem : The physical pages to be passed in to the guest

9.2.2.3. Creating a Partial Device Tree

The last step in setting up the system for passthrough involves creating a device tree for the domain called a *partial device tree*.

Create a new file and name it *xen-partial.dts*. Ensure that this file is located in the **device_tree** path indicated in the configuration file shown in Figure 1. The entire *xen-partial.dts* should be saved on your SD card in the *dom0* extended FS at '/media/$USER/rootfs/overlay/etc/xen' , or on the target at '/etc/xen', and look like the one below:

```
/dts-v1/;

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

    passthrough {
        compatible = "simple-bus";
```

```
        ranges;
        #address-cells = <0x2>;
        #size-cells = <0x1>;

        misc_clk {
            #clock-cells = <0x0>;
            clock-frequency = <0x17d7840>;
            compatible = "fixed-clock";
            linux,phandle = <0x2>;
            phandle = <0x2>;
        };

        ethernet@ff0e0000 {
            #stream-id-cells = <0x1>;
            compatible = "cdns,gem";
            reg = <0x0 0xff0e0000 0x1000>;
            interrupts = <0x0 0x3f 0x4 0x0 0x3f 0x4>;
            clock-names = "pclk", "hclk", "tx_clk";
            clocks = <0x2 0x2 0x2>;
            #address-cells = <0x1>;
            #size-cells = <0x0>;
            phy-handle = <0x1>;
            phy-mode = "rgmii-id";

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

Figure 2: xen-partial.dts

The *xen-partial.dts* now needs to be compiled into a binary file known as a device tree blob (dtb). We will use the host's dtc compiler, mentioned above, to compile the dts file into a dtb file.

Generate the dtb file by entering the following command:

```
$ dtc -I dts -O dtb -o xen-partial.dtb xen-partial.dts
```

9.2.2.4. Communicating with the Domains

You are now ready to boot both domain 0 and domain 1, each one is assigned a unique Ethernet device. To test the Ethernet device passthrough with the ZCU102, one can ping a domain when logged into another domain and visa-versa.

One can also use the *nc,* or *netcat,* command that tests the ability to communicate between the domains and the host computer. To log into the host computer from a domain, enter the following command on your host computer.

```
$ nc -l 4321
```

This will create a quick server that can be used to communicate to other systems via an IP address. On one of the domains, enter the following command.

```
# telnet 10.0.2.2 4321
```

This will set up a connection with the host and allow you to send characters between the domain and the host computer. Entering ctrl-c in the domain will break the connection. Now follow the same steps in the other domain to test the other Ethernet device.

10.1. Current Support Options

For more information and support, please visit our web site.

http://dornerworks.com/services/xilinxxen

Details on the Xilinx Zynq UltraScale+ MPSoC can be found here:

http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html