# Boot the zc706 with the Analog Devices ADRV9371 Images over NFS

Status: Not yet working (SD Card boot does work)

Current issue: kernel panic

## 1) Set up NFS Share and Obtain Filesystem Content

First, we became the user ori and created an NFS share containing the Analog Devices filesystem content. The directory is **/home/ori/exports/adi**.

The goal was to create an easily accessible NFS share with the desired filesystem that volunteers on the VM could use. One can also use these instructions and set up an NFS share independent of the **/home/ori/exports/adi** directory.

Why NFS? We need more space for the root filesystem than can fit in memory. We do not want to be restricted to using an SD card because that requires manual intervention to set up, and remote developers should not have to wait. Booting the Analog Devices ADRV9371 environment from SD card does work and demonstrations have been made. Converting to NFS share would be a very large step forward in ease of use and capability at Remote Labs.

Analog Devices files were obtained from the SD card image provided by Analog Devices.

## Procedure Followed

This procedure is on the host machine.

**su ori**
*<entered password>*
**cd /home/ori/exports**

**mkdir petalinux**
**cd petalinux**

copy an image to this directory. For example, the contents of /home/suoto/exports/petalinux

**sudo cp -r /home/suoto/exports/petalinux/* .**

**sudo nano /etc/exports**

This is a configuration file that lists all the possible places for mounting from NFS.

Make another entry here that looks like these other entries, that points to this new directory.

**/home/ori/exports/petalinux *(rw,nohide,insecure,no_subtree_check, async,no_root_squash)**

“async” is an interesting decision. It makes the NFS faster but is also more dangerous.

Save that file.

Close that editor.

There are some magic words that make the changes to this file take effect.

**sudo exportfs -ra**

**sudo service nfs-server restart** to restart the NFS server

At this point we had a new NFS mount. Testing showed that it was working as expected, with this small Petalinux-built root filesystem. We also needed a root filesystem that mirrored the one provided by Analog Devices in their SD card image. The steps are similar:

**su ori**
*<entered password>*
**cd /home/ori/exports**
**mkdir adi**
**cd adi**

To obtain access to the SD card image, we plugged the SD card itself into a reader and plugged the reader into a USB port bound to the chococat VM. We then mounted the SD card image on the VM in order to copy it:

**sudo mkdir /mnt/sdcard**
**sudo mount /dev/sda2 /mnt/sdcard**
**sudo cp -r /mnt/sdcard/* .**

**sudo nano /etc/exports**

This is a configuration file that lists all the possible places for mounting from NFS.

Make another entry here that looks like these other entries, that points to this new directory.

**/home/ori/exports/adi *(rw,nohide,insecure,no_subtree_check, async,no_root_squash)**

Save that file.

Close that editor.

**sudo exportfs -ra**

**sudo service nfs-server restart** to restart the NFS server

# 2) Build Petalinux Image

Build a petalinux image, with NFS turned on, and files copied to /tftpboot for tftp booting.

## Procedure Followed

This procedure is on the host machine.

Assume successful install of petalinux and Vivado. Assume an account on chococat.sandiego.openresearch.institute.

**source ~/petalinux/settings.sh**

**petalinux-create --type project --template zynqMP --name <test-NFS> --source <path-to-BSP>**

**cd test-NFS**

**petalinux-config --get-hw-description=<path to zc706.xsa>**

**petalinux-config**

there are some settings in this menuconfig that need to be enabled.

NFS is the RootFS type
set the location of the NFS server

**petalinux-config -c kernel**

networking support => IP: kernel level configuration

Confirm that DHCP, BOOTP, and RARP are enabled.

file systems => network file systems => root file system on NFS

**petalinux-configs**

image packaging configuration => copy final images to to tftpboot

For TFTPBOOT to work, a TFTP server must be enabled and started and a /tftpboot directory set up with correct permissions. There's a procedure for setting this up in the petalinux documentation.

**petalinux-build**

**petalinux-package --prebuilt --fpga <path-to-bitstream-from-analog-devices>**

**petalinux-boot --jtag --prebuilt 2 --hw_server-url TCP:chococat:3121**

This stalled out at kyber registration.

Although the fpga file is picked up in petalinux-package and is successfully copied to the correct prebuilt directory, the petalinux-boot command gives a warning that no FPGA bitstream will be loaded.

**petalinux-boot --jtag --prebuilt 2 --fpga --bitstream <path-to-bitstream-from-analog-devices>**

fails. This isn't unexpected given the documentation.

Loading the bitstream first with

**petalinux-boot --fpga --bitstream <path-to-bitstream-from-analog-devices>**

and then

**petalinux-boot --jtag --prebuilt 2 --hw_server-url TCP:chococat:3121**

does get past the kyber registration step, but later on there's a kernel panic.

Something is still wrong at some point in the booting process. There is a lot going right, but it doesn't work yet. We'll keep at it until it works.

# 3) Use TFTP Boot

Assume we have a built petalinux image, with NFS turned on, and files copied to /tftpboot for tftp booting (accomplished using petalinux-config).

Establish a serial terminal to zc706 from the host machine (in this case, the host machine is chococat).

**for example,**

**screen -L -Logfile** <*log file name*> **/dev/ttyUSB0 115200**

**petalinux-boot --jtag --prebuilt 2 --hw_server-url TCP:chococat:3121**

hit any key to stop autobooting.
Zyng> indicates a u-boot prompt on the zc706. We got there by using a screens session from the host machine.

**Zynq> printenv bootargs**

They were incorrect.

**Zynq> editenv bootargs**

**Zynq> edit: console=ttyPS0,115200 earlycon root=/dev/nfs**
**nfsroot=10.73.1.93:/home/ori/exports/petalinux,nfsvers=3,tcp ip=10.73.1.9 rw rootfstype=nfs**

**Zynq> printenv bootargs**
**bootargs=console=ttyPS0,115200 earlycon root=/dev/nfs**
**nfsroot=10.73.1.93:/home/ori/exports/petalinux,nfsvers=3,tcp ip=10.73.1.9 rw rootfstype=nfs**
**Zynq> tftpboot 0x00200000 10.73.1.93:uImage**
**Zynq> bootm 0x00200000 - 0x00100000**

login with root:root unless you have changed it in petalinux-config.

Now, this worked, but we then learned how to properly save the bootargs to flash, which eliminated this re-typing of the bootargs every time through at the u-boot prompt.

“Don’t forget to use **saveenv** or else the changes to environment variables in u-boot will disappear upon development board reset.”

We then created an environment variable called **adi_tftpboot** designed to boot the Analog Devices components. **adi_tftpboot** was parameterized for portability.

```
Zynq> printenv bootargs
bootargs=console=ttyPS0,115200 earlycon root=/dev/nfs nfsroot=${nfs_ip}:$
{nfs_rfs},nfsvers=3,tcp ip=${ipaddr} rw rootfstype=nfs
Zynq> printenv adi_tftpboot
adi_tftpboot=echo Booting with TFTP... && tftpboot 0x02A00000 ${serverip}:${tftproot}/$
{devicetree_image} && tftpboot 0x03000000 ${serverip}:${tftproot}/${kernel_image} && if
tftpboot 0x02000000 ${serverip}:${tftproot}/${ramdisk_image}; then bootm 0x03000000
0x02000000 0x02A00000; else bootm 0x03000000 - 0x02A00000; fi
Zynq> printenv serverip
serverip=10.73.1.93
Zynq> printenv tftproot
tftproot=adiboot
Zynq> printenv devicetree_image
devicetree_image=devicetree.dtb
Zynq> printenv kernel_image
kernel_image=uImage
Zynq> printenv ramdisk_image
## Error: "ramdisk_image" not defined
Zynq> printenv bootcmd
bootcmd=run adi_tftpboot
```

# 4) Appendix

## Reset the zc706

To reset the zc706 over jtag:

assuming bin directory of Vivado is on your path, then:

```
abraxas3d@chococat:~/stillworks$ xsdb
xsdb% connect -url TCP:chococat:3121
tcfchan#0
xsdb% targets
  1  APU
     2  ARM Cortex-A9 MPCore #0 (Running)
     3  ARM Cortex-A9 MPCore #1 (Running)
  4  xc7z045
xsdb% targets 1
xsdb% rst -srst -clear-registers
xsdb% exit
exit
```

Once it was clear this short procedure in xsdb worked to reset the board over JTAG, we put these commands into a script using the autoexpect utility.