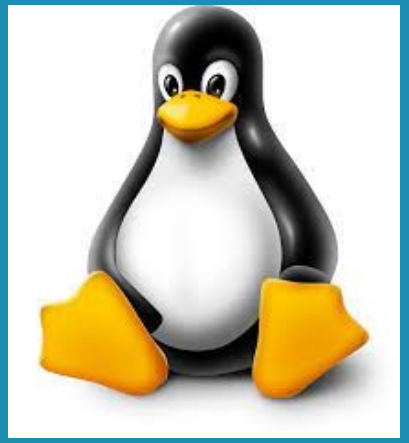
KU LEUVEN

Petalinux demo



Create Virtual Box : create VM

- Name : Petalinux2401
- System type : Linux / Ubuntu
- Disk (vdi) : 200GB (depends on image you create...)
- Add optical drive : iso of Ubuntu 22.04.5 LTS (20.04.6 LTS)
- Boot virtualbox
- Install : Minimal system
- Username/password : xilinx/xilinx

https://discuss.pynq.io/t/stuck-at-extract-yocoto/4134/8

Create Virtual Box : install Ubuntu

- Boot the system
- Devices/insert guest additions cd image…
- Run the guest additions
- **Do not upgrade the system to a newer version of Ubuntu**
- Sudo apt update
- Sudo apt upgrade → this will upgrade packages, not the whole system
- Download Vitis 2024.1 (Unified Web installer Linux)
- Download Petalinux 2024.1

Create Virtual Box

Two options:

Create a Virtual Box with only Petalinux to create basic embedded linux images: https://www.hackster.io/whitney-knitter/vivado-vitis-petalinux-2024-1-install-on-ubuntu-22-04-e76e91

Create a Virtual Box to build PYNQ (This requires Vivado 2022.1 and Ubuntu 20.04 LTS 6)

PYNQ Virtual Box : install packages

- Sudo apt install build-essential gawk micro
- sudo apt install net-tools xterm autoconf libtool texinfo zlib1g-dev gcc-multilib zlib1g-dev:i386 ncurses-dev
- Sudo apt install blt diffstat libegl1-mesa libncurses5-dev libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib libtcl8.6 libtk8.6 pax pylint python-is-python2 python2 python2-minimal python2.7 python2.7-minimal python3-astroid python3-git python3-gitdb python3-isort python3-jinja2 python3-lazy-object-proxy python3-logilab-common python3-markupsafe python3-mccabe python3-smmap python3-tk python3-wrapt screen tftpd tk8.6-blt2.5 xinetd
- Sudo dpkg-reconfigure dash → (select No in the dialog box)
- Sudo apt install git
- wget http://mirrors.kernel.org/ubuntu/pool/main/libi/libidn/libidn11_1.33-2.2ubuntu2_amd64.deb
- Sudo apt install ./libidn11_1.33-2.2ubuntu2_amd64.deb

Create Virtual Box : clone PYNQ.git

- Cd ~
- Git clone https://github.com/Xilinx/PYNQ.git
- Cd ~/PYNQ/sdbuild
- ./scripts/setup_host.sh

PYNQ Virtual Box : install Petalinux

- cd ~/Downloads
- chmod u+x ./petalinux-v2022.1-04191534-installer.run
- ./petalinux-v2022.1-04191534-installer.run -d */home/xilinx/petalinux --platform “arm”*
- *Accept licenses and install*

PYNQ Virtual Box : install Vitis

- Cd ~/Downloads
- chmod u+x ./Xilinx_Unified_2022.1_0420_0327_Lin64.bin
- ./Xilinx_Unified_2022.1_0420_0327_Lin64.bin
- Select Vitis and only the 7000 Series board (same as installation for DDC)
- python3 -m pip install --upgrade pip

Create Virtual Box : tftpboot

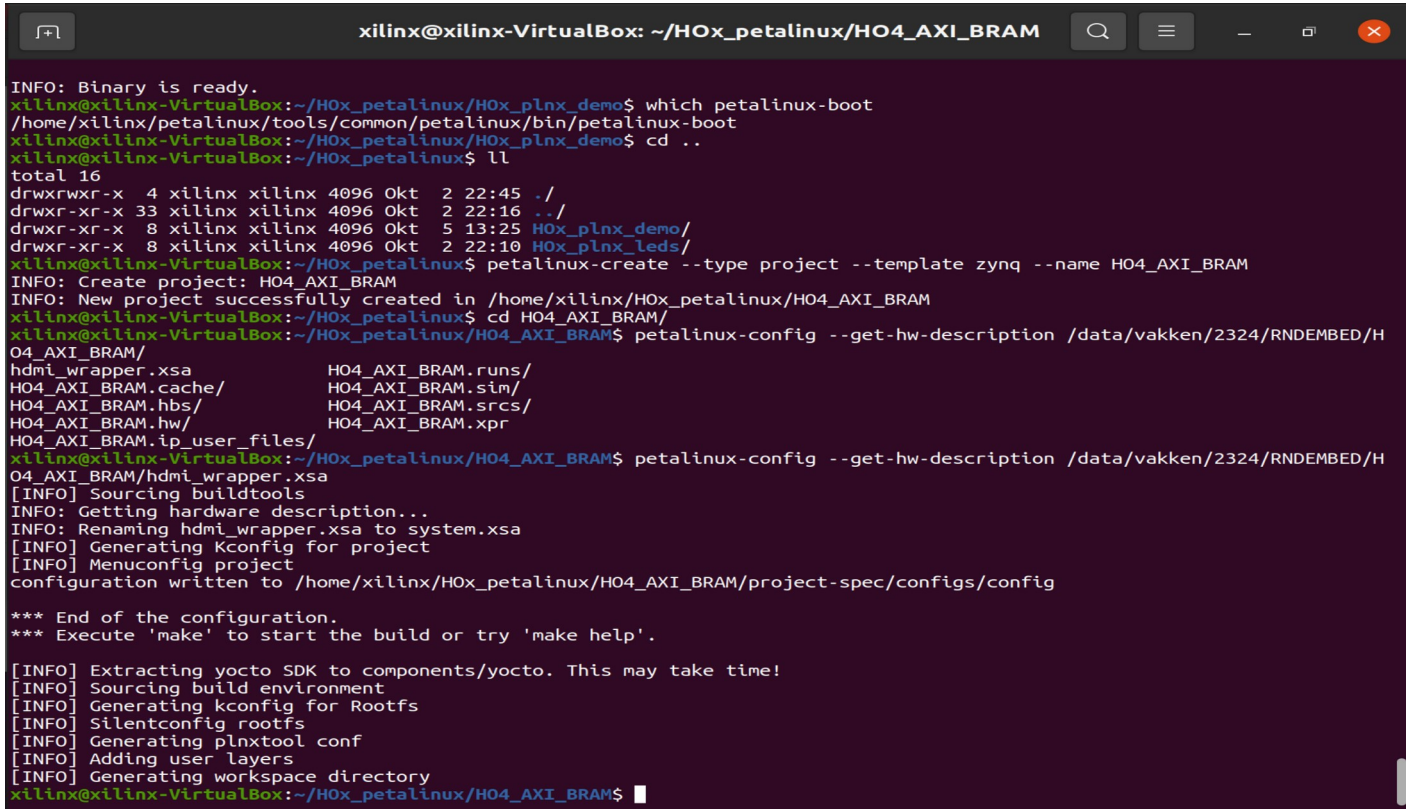
- When using tftpboot make sure you set /tftpboot of your host system as a shared folder with the VM, name it also /tftpboot
- Also share another folder (eg. your Vivado projects)
- Make your xilinx user member of the vboxsf group:
  - Sudo adduser xilinx vboxsf

Open console

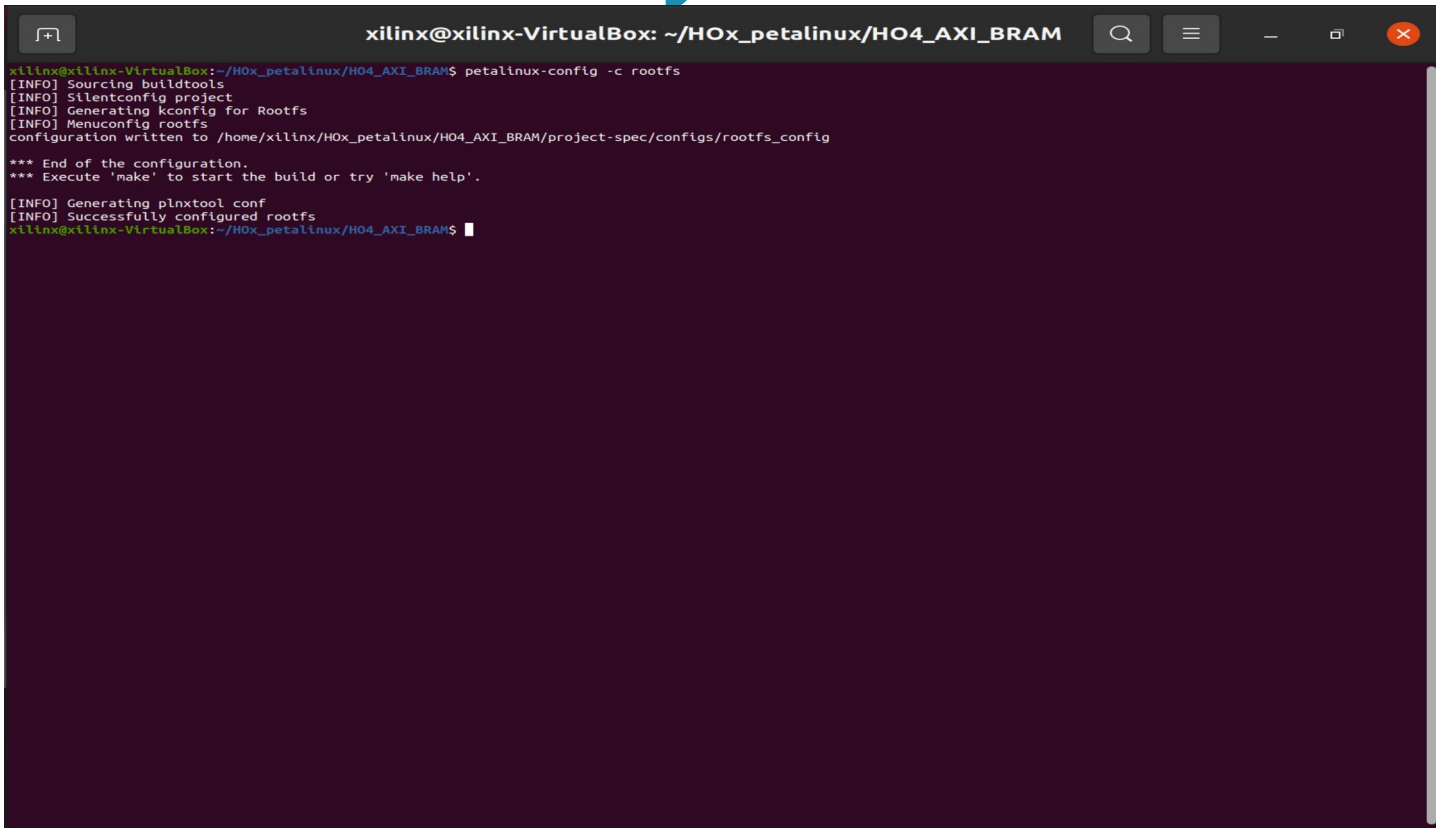
- Source */tools/Xilinx/Vivado/2022.1/settings64.h*
- *Source /tools/Xilinx/Vitis/2022.1/settings64.h*
- *Source /home/xilinx/petalinux/settings.h*

- ***Do this for each console you open*** *(optionally add it to .bashrc)*

Create a new project from HO4_AXI_BRAM



Create root file system

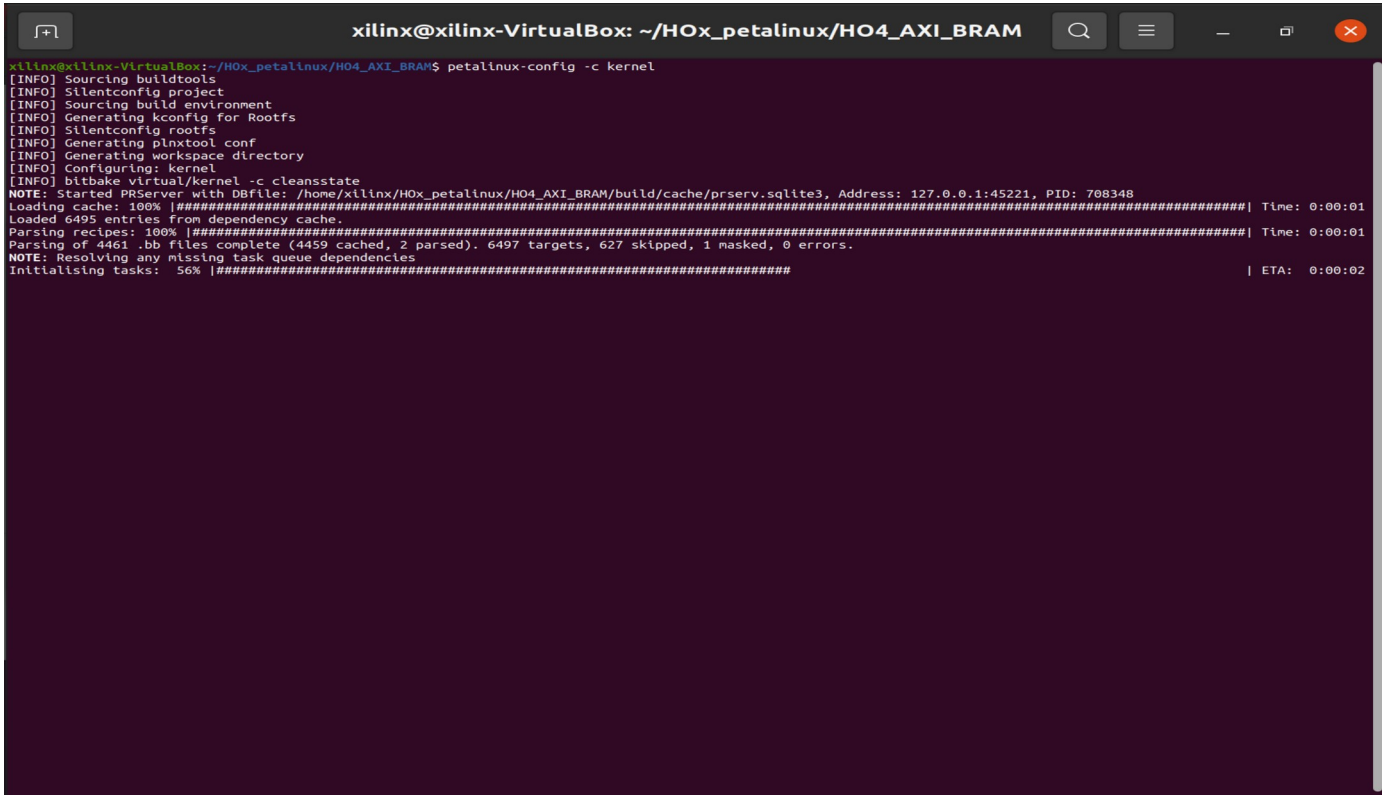


KU LEUVEN

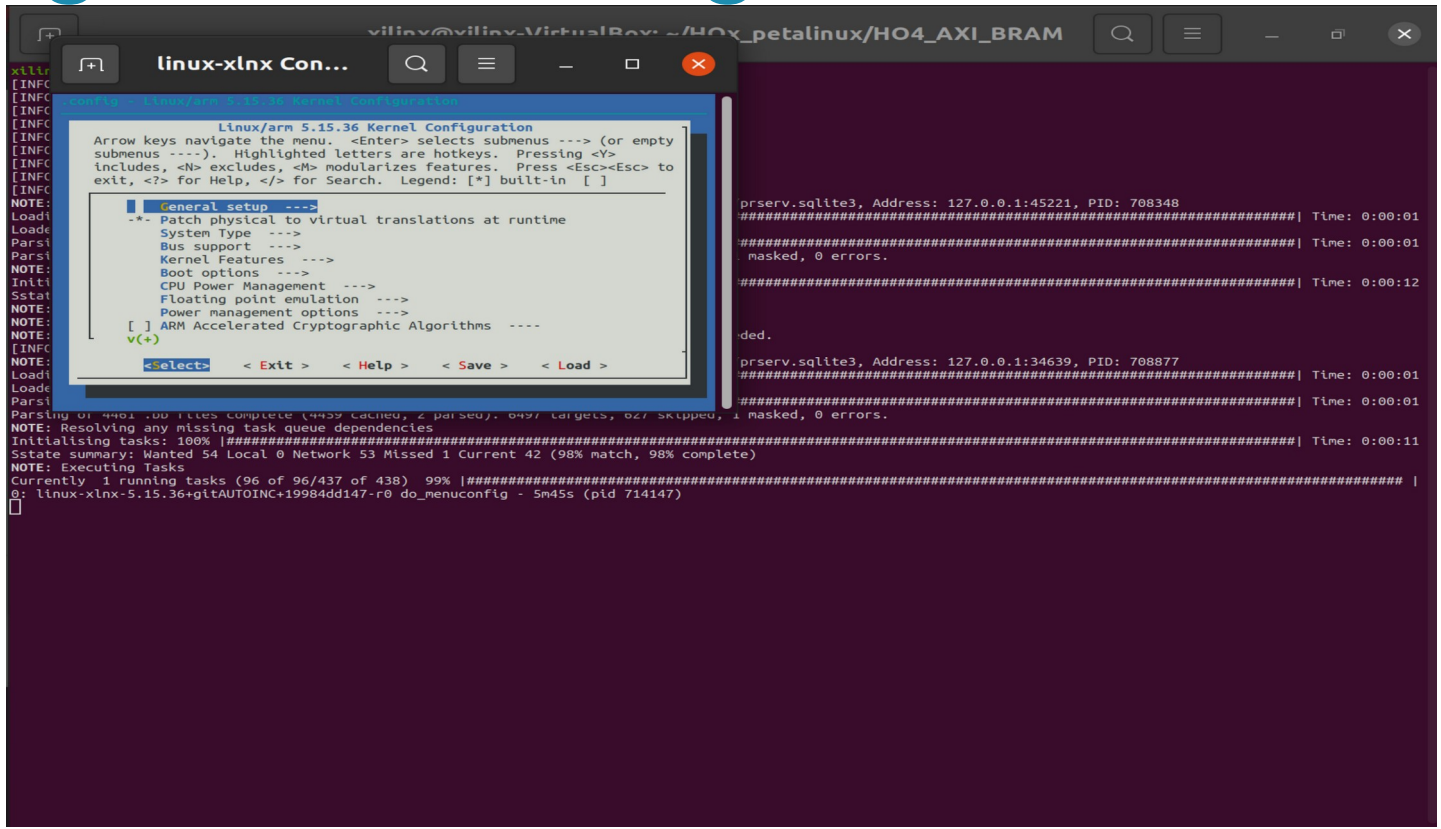
Rootfs interesting packages

- You can search in the packages by pressing '*/*' and your search query
- Filesystem packages → base → dnf
- Imagefeatures → package-management
- Openssh (unselect dropbear)
- Filesystem packages → misc → ca-certificates
- Filesystem packages → misc → packagegroup-core-buildessential

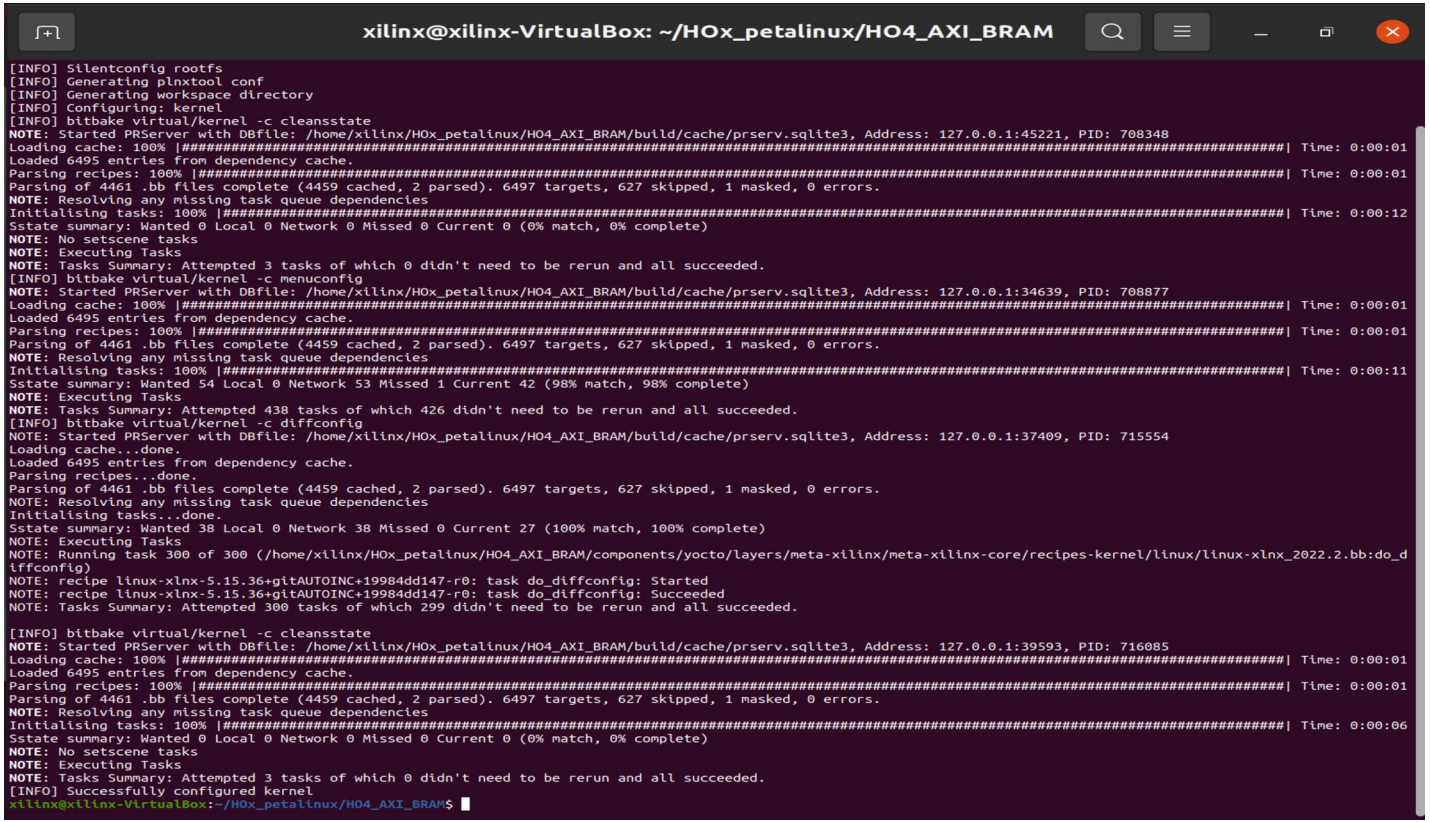
Config Kernel image (get coffee!)



Config Kernel image

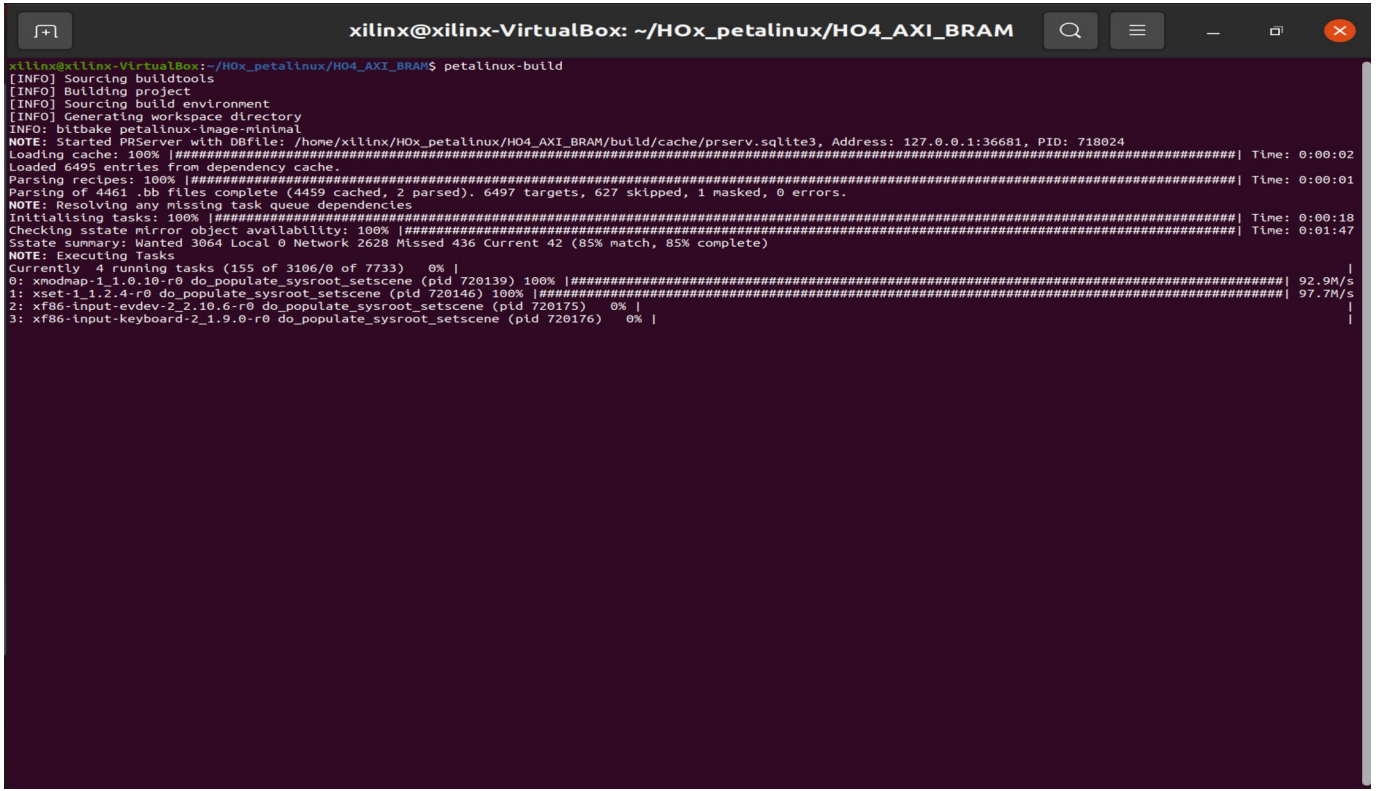


Config Kernel image



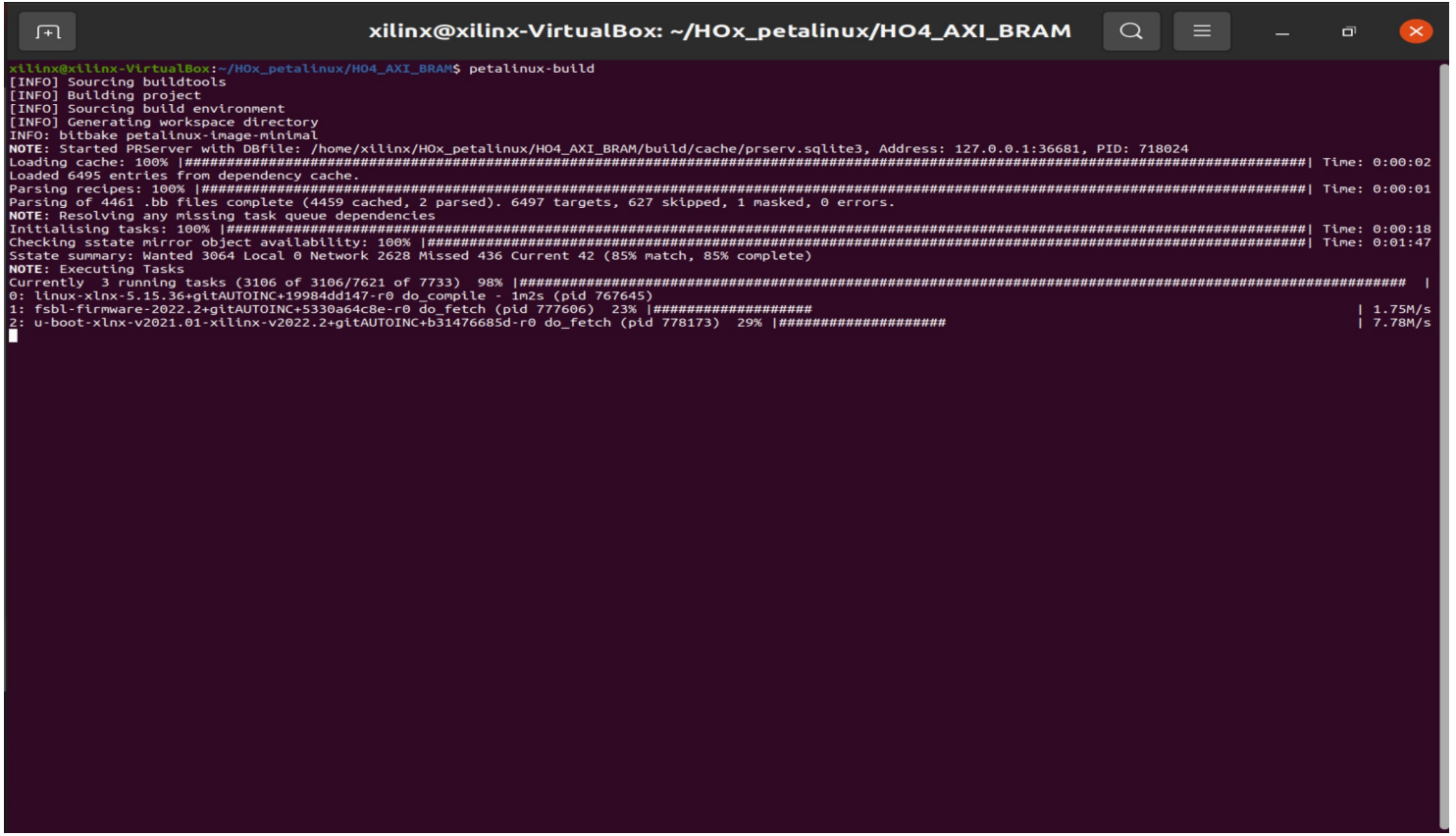
KU LEUVEN

Build petalinux (read a chapter of Hitchhikers Guide to the Galaxy – 25 mins)



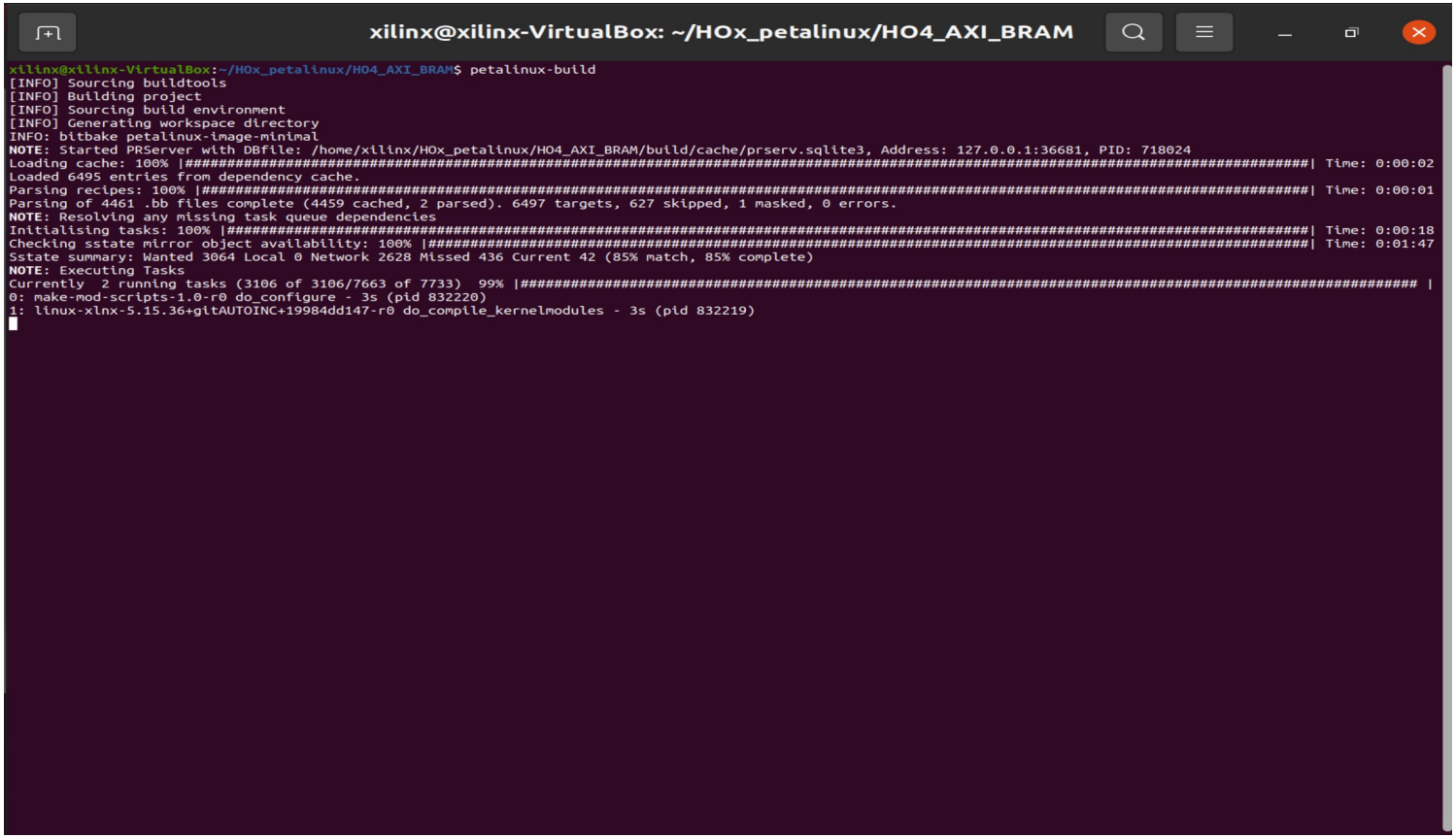
KU LEUVEN

After 15 minutes



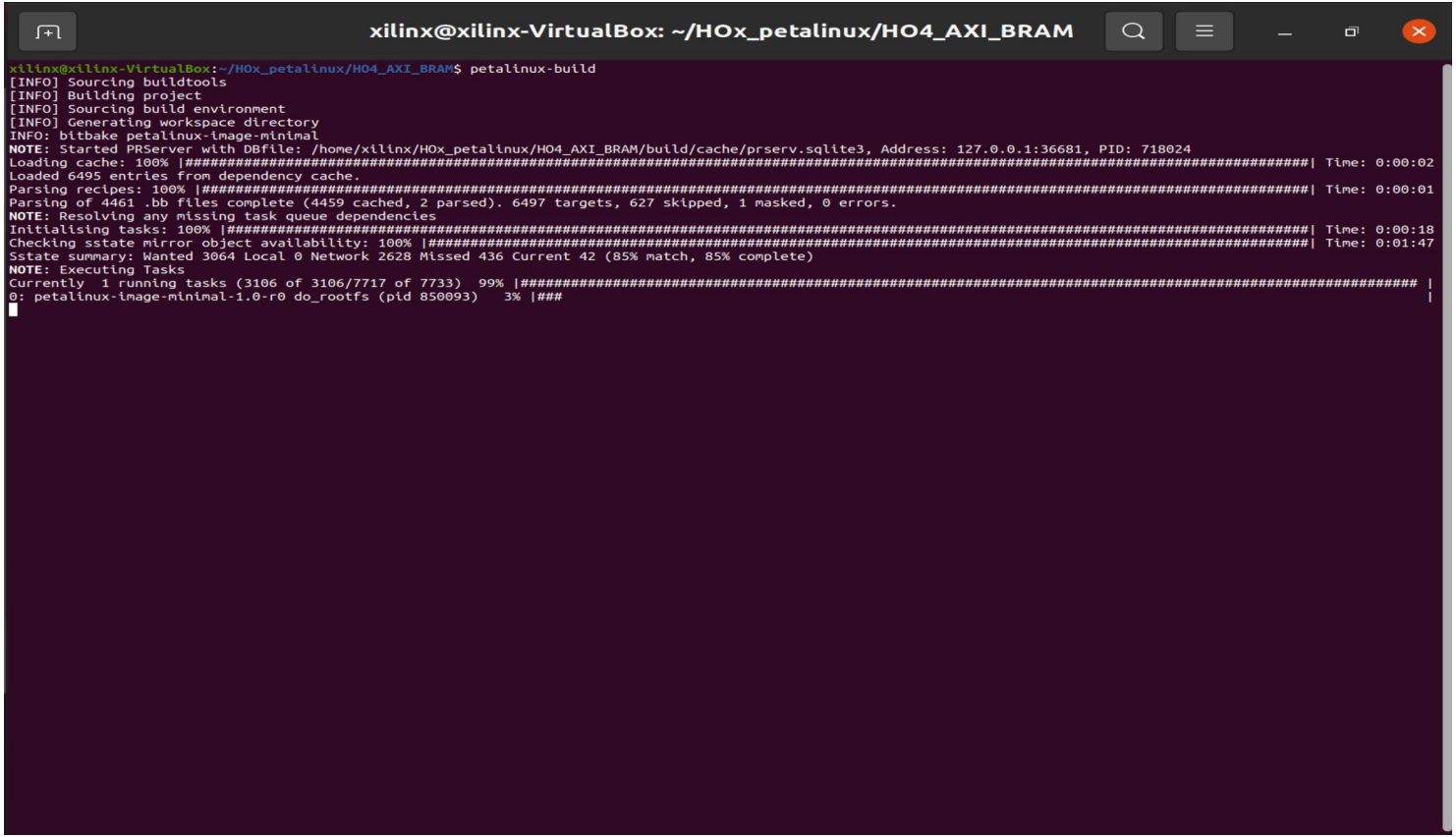
KU LEUVEN

Read...



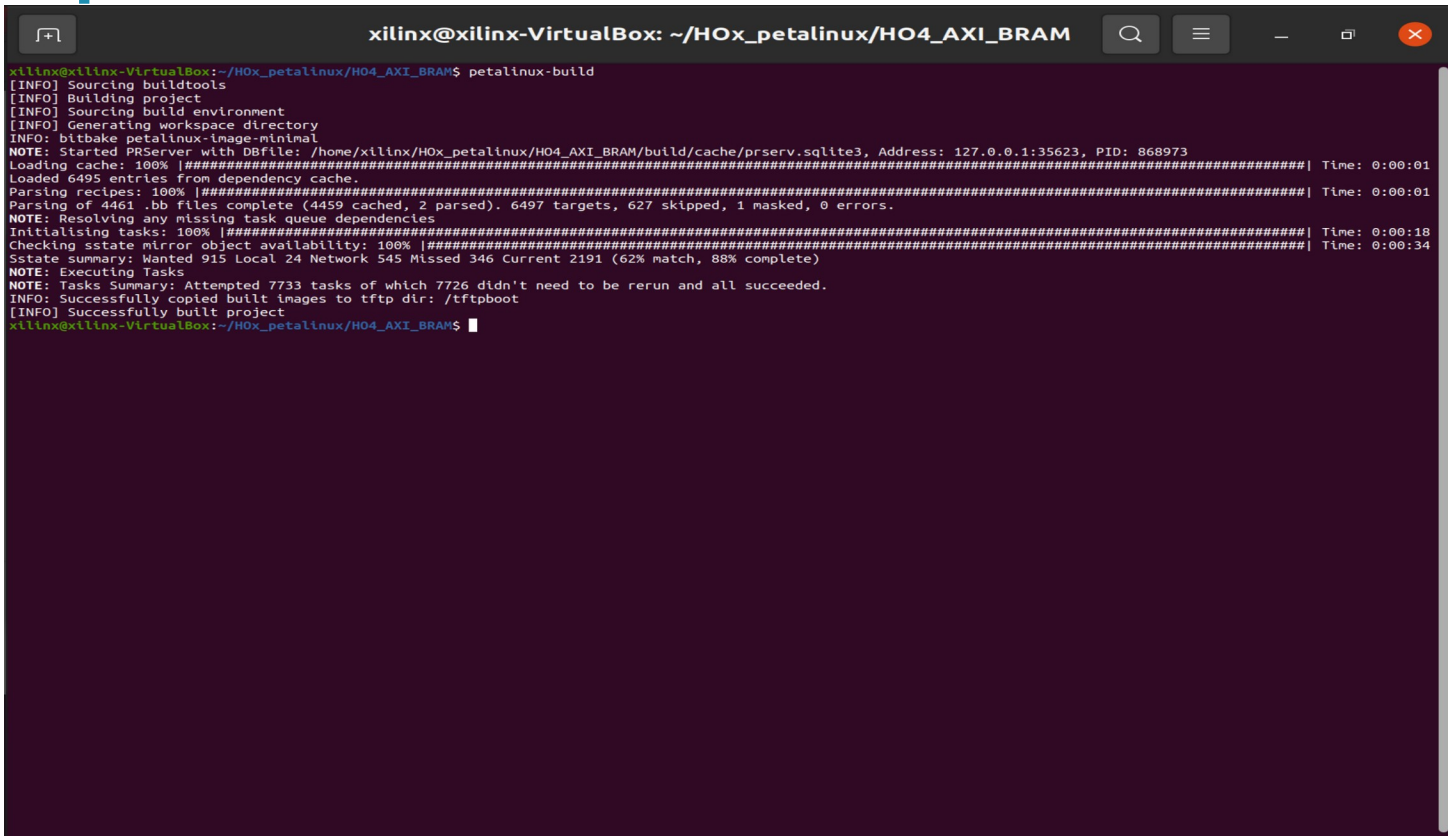
KU LEUVEN

Read... read...



KU LEUVEN

Build petalinux – done!



KU LEUVEN

Check generated images and create BOOT.BIN

```
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$ ls -al images/linux/
total 2343300
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 22:37 .
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 21:59 ..
-rw-rw-r-- 1 xilinx xilinx    5136248 Okt  7 22:22 BOOT.BIN
-rw-rw-r-- 1 xilinx xilinx        313 Okt  7 22:22 bootgen.bif
-rw-r--r-- 1 xilinx xilinx       2776 Okt  7 22:00 boot.scr
-rw-r--r-- 1 xilinx xilinx       7326 Okt  7 22:30 config
-rw-r--r-- 1 xilinx xilinx 203896268 Okt  7 22:37 image.ub
drwxr-xr-x 2 xilinx xilinx       4096 Okt  7 22:00 pxelinux.cfg
-rw-r--r-- 1 xilinx xilinx 525044224 Okt  7 22:36 rootfs.cpio
-rw-r--r-- 1 xilinx xilinx 199334651 Okt  7 22:36 rootfs.cpio.gz
-rw-r--r-- 1 xilinx xilinx 199334715 Okt  7 22:36 rootfs.cpio.gz.u-boot
-rw-r--r-- 1 xilinx xilinx 765291520 Okt  7 22:36 rootfs.ext4
-rw-r--r-- 1 xilinx xilinx 262668288 Okt  7 22:36 rootfs.jffs2
-rw-r--r-- 1 xilinx xilinx     47704 Okt  7 22:35 rootfs.manifest
-rw-r--r-- 1 xilinx xilinx 200900276 Okt  7 22:36 rootfs.tar.gz
-rw-r--r-- 1 xilinx xilinx   4045672 Okt  7 20:31 system.bit
-rw-r--r-- 1 xilinx xilinx     19703 Okt  7 22:33 system.dtb
-rwxr-xr-x 1 xilinx xilinx    966664 Okt  7 22:33 u-boot.bin
-rw-r--r-- 1 xilinx xilinx    986367 Okt  7 22:33 u-boot-dtb.bin
-rw-r--r-- 1 xilinx xilinx   1052328 Okt  7 22:33 u-boot-dtb.elf
-rwxr-xr-x 1 xilinx xilinx   7279024 Okt  7 22:33 u-boot.elf
-rw-r--r-- 1 xilinx xilinx   4539856 Okt  7 22:37 uImage
-rw-r--r-- 1 xilinx xilinx  13922724 Okt  7 22:37 vmlinux
-rw-r--r-- 1 xilinx xilinx   4539792 Okt  7 22:37 zImage
-rw-r--r-- 1 xilinx xilinx    453152 Okt  7 22:01 zynq_fsbl.elf
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$ petalinux-package --boot --fsbl images/linux/zynq_fsbl.elf --u-boot images/linux/u-boot.elf --fpga images/linux/sys
tem.bit --force
[INFO] Sourcing buildtools
INFO: Getting system flash information...
INFO: File in BOOT BIN: "/home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/zynq_fsbl.elf"
INFO: File in BOOT BIN: "/home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/system.bit"
INFO: File in BOOT BIN: "/home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/u-boot.elf"
INFO: File in BOOT BIN: "/home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/system.dtb"
INFO: Generating zynq binary package BOOT.BIN...


****** Xilinx Bootgen v2022.2
  **** Build date : Sep 26 2022-06:24:42
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

[WARNING]: Partition zynq_fsbl.elf.0 range is overlapped with partition system.bit.0 memory range
[WARNING]: Partition system.bit.0 range is overlapped with partition system.dtb.0 memory range

[INFO]   : Bootimage generated successfully

INFO: Binary is ready.
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$ ls -al images/linux/
total 2343300
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 22:38 .
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 21:59 ..
-rw-rw-r-- 1 xilinx xilinx    5136248 Okt  7 22:38 BOOT.BIN
-rw-rw-r-- 1 xilinx xilinx        313 Okt  7 22:38 bootgen.bif
-rw-r--r-- 1 xilinx xilinx       2776 Okt  7 22:00 boot.scr
-rw-r--r-- 1 xilinx xilinx       7326 Okt  7 22:30 config
-rw-r--r-- 1 xilinx xilinx 203896268 Okt  7 22:37 image.ub
```


KU LEUVEN

Create image for sd card

```
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$ petalinux-package --wic
INFO: Sourcing build environment
INFO: Extracting rootfs, This may take time!
INFO: Creating wic image...
INFO: wic create /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/build/rootfs.wks --rootfs-dir /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/build/wic/rootfs --bootimg-dir /home/xilinx
/HOx_petalinux/HO4_AXI_BRAM/images/linux/ --kernel-dir /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/ --outdir /tmp/tmp.G3dvEk0t3x -n /home/xilinx/HOx_petalinux/HO
4_AXI_BRAM/build/tmp/work/cortexa9t2hf-neon-xilinx-linux-gnueabi/wic-tools/1.0-r0/recipe-sysroot-native
INFO: Creating image(s)...

WARNING: bootloader config not specified, using defaults

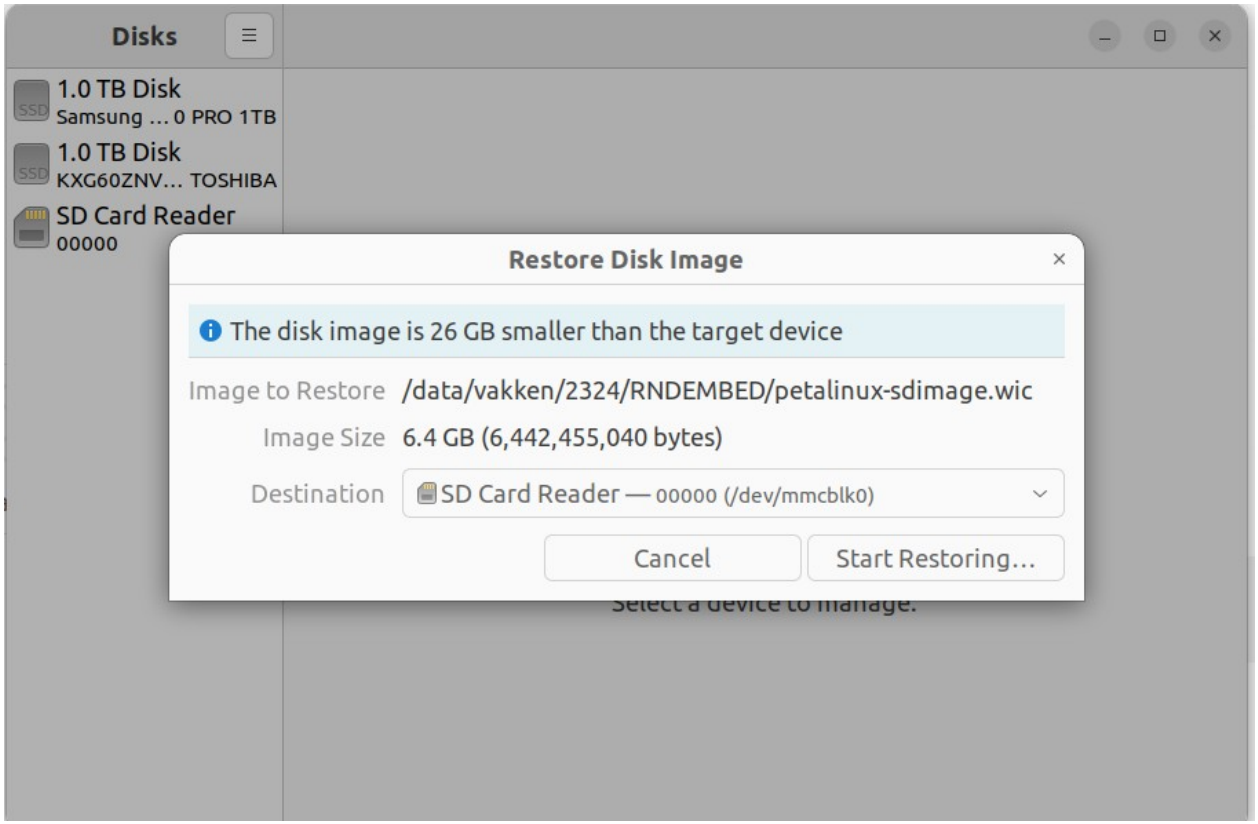
INFO: The new image(s) can be found here:
  /tmp/tmp.G3dvEk0t3x/rootfs-202310072040-mmcblk0.direct

The following build artifacts were used to create the image(s):
  ROOTFS_DIR:                   /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/build/wic/rootfs
  BOOTIMG_DIR:                  /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/
  KERNEL_DIR:                   /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/images/linux/
  NATIVE_SYSROOT:               /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/build/tmp/work/cortexa9t2hf-neon-xilinx-linux-gnueabi/wic-tools/1.0-r0/recipe-sysroot-native

INFO: The image(s) were created using OE kickstart file:
  /home/xilinx/HOx_petalinux/HO4_AXI_BRAM/build/rootfs.wks
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$ ls -al images/linux/
total 3000796
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 22:41 .
drwxr-xr-x 3 xilinx xilinx       4096 Okt  7 21:59 ..
-rw-rw-r-- 1 xilinx xilinx    5136248 Okt  7 22:38 BOOT.BIN
-rw-rw-r-- 1 xilinx xilinx        313 Okt  7 22:38 bootgen.bif
-rw-r--r-- 1 xilinx xilinx       2776 Okt  7 22:00 boot.scr
-rw-r--r-- 1 xilinx xilinx       7326 Okt  7 22:30 config
-rw-r--r-- 1 xilinx xilinx  203896268 Okt  7 22:37 image.ub
-rw-r--r-- 1 xilinx xilinx 6442455040 Okt  7 22:41 petalinux-sdimage.wic
drwxr-xr-x 2 xilinx xilinx       4096 Okt  7 22:00 pxelinux.cfg
-rw-r--r-- 1 xilinx xilinx  525044224 Okt  7 22:36 rootfs.cpio
-rw-r--r-- 1 xilinx xilinx  199334651 Okt  7 22:36 rootfs.cpio.gz
-rw-r--r-- 1 xilinx xilinx  199334715 Okt  7 22:36 rootfs.cpio.gz.u-boot
-rw-r--r-- 1 xilinx xilinx  765291520 Okt  7 22:36 rootfs.ext4
-rw-r--r-- 1 xilinx xilinx  262668288 Okt  7 22:36 rootfs.jffs2
-rw-r--r-- 1 xilinx xilinx      47704 Okt  7 22:35 rootfs.manifest
-rw-r--r-- 1 xilinx xilinx  200900276 Okt  7 22:36 rootfs.tar.gz
-rw-r--r-- 1 xilinx xilinx    4045672 Okt  7 20:31 system.bit
-rw-r--r-- 1 xilinx xilinx      19703 Okt  7 22:33 system.dtb
-rwxr-xr-x 1 xilinx xilinx     966664 Okt  7 22:33 u-boot.bin
-rw-r--r-- 1 xilinx xilinx     986367 Okt  7 22:33 u-boot-dtb.bin
-rw-r--r-- 1 xilinx xilinx    1052328 Okt  7 22:33 u-boot-dtb.elf
-rwxr-xr-x 1 xilinx xilinx    7279024 Okt  7 22:33 u-boot.elf
-rw-r--r-- 1 xilinx xilinx    4539856 Okt  7 22:37 uImage
-rw-r--r-- 1 xilinx xilinx   13922724 Okt  7 22:37 vmlinux
-rw-r--r-- 1 xilinx xilinx    4539792 Okt  7 22:37 zImage
-rw-r--r-- 1 xilinx xilinx     453152 Okt  7 22:01 zynq_fsbl.elf
xilinx@xilinx-VirtualBox:~/HOx_petalinux/HO4_AXI_BRAM$
```


KU LEUVEN

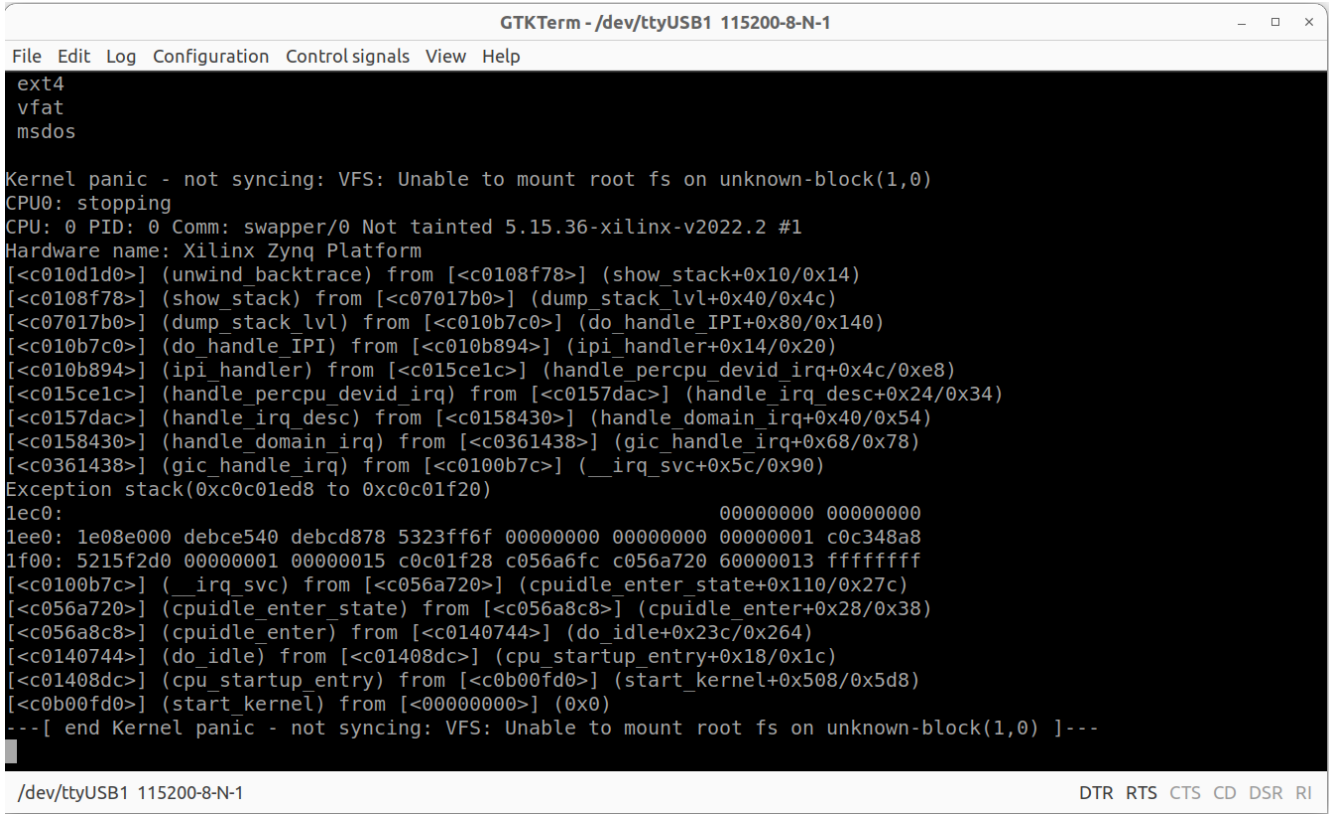
Write image to sd card (14 mins)



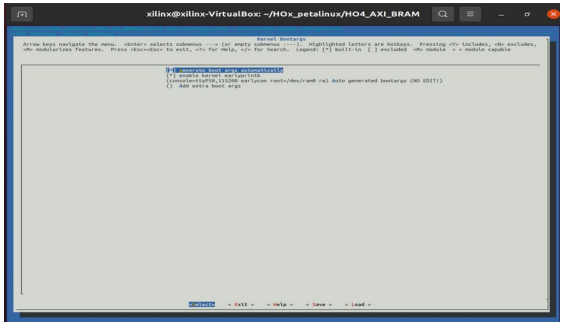
Boot Pynq-Z2 from sd card

- Insert sd card
- Change bootmode jumper to SD
- Turn on board
- Connect serial terminal to */dev/ttyUSB1 115200 baud*

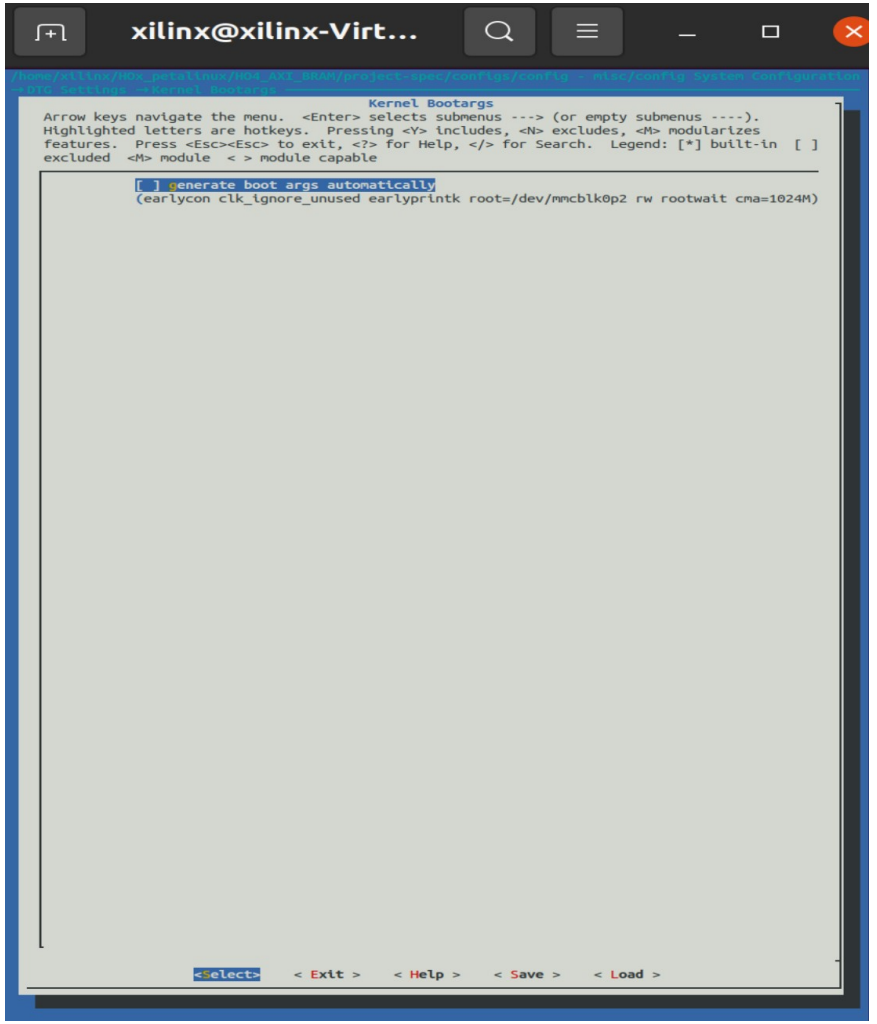
Kernel Panic!



Don’t panic - 42



- petalinux-config
- DTG Settings
- Kernel bootargs
- Turn off generate boot args automatically
- earlycon clk_ignore_unused earlyprintk root=/dev/mmcblk0p2 rw rootwait cma=1024M

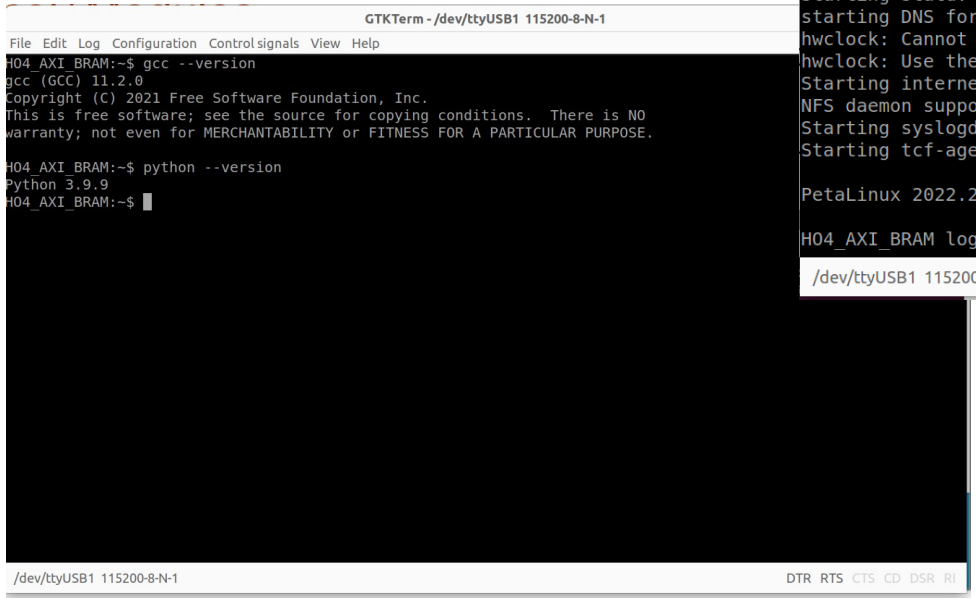


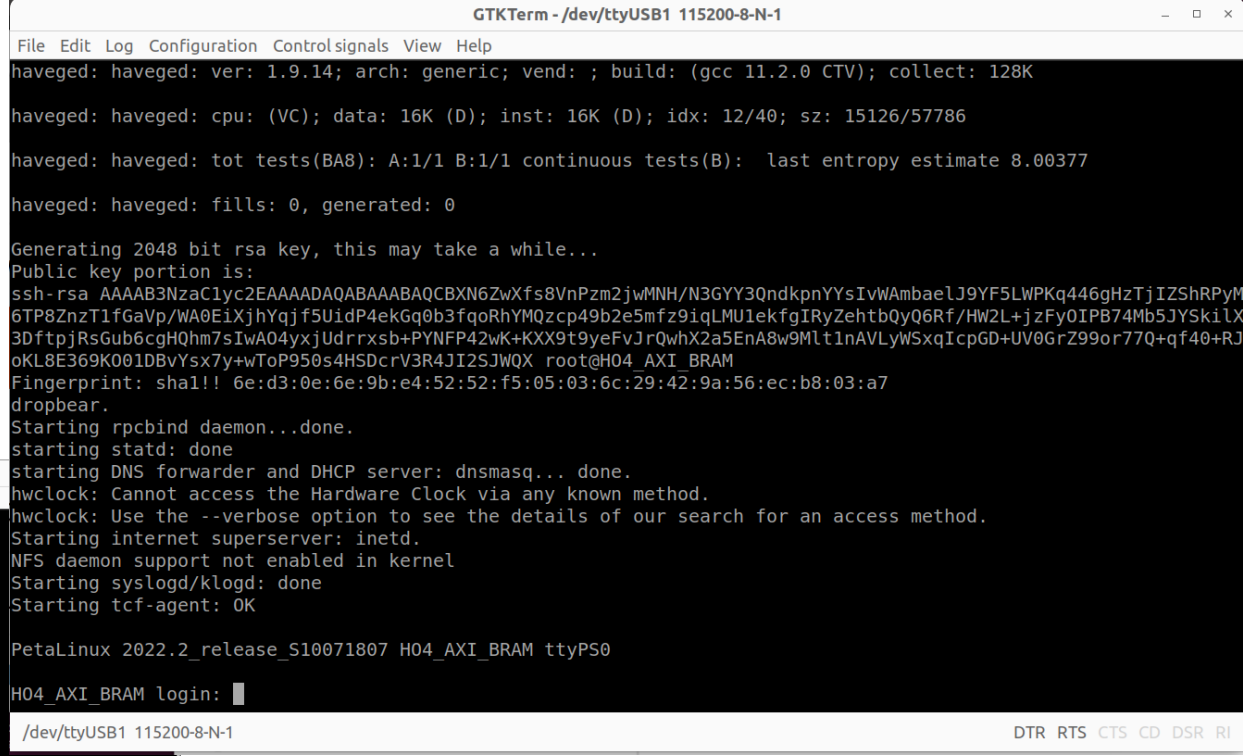
Don’t panic - 42

- petalinux-build
- petalinux-package --boot ...
- Copy BOOT.bin, boot.scr and uImage to SD card and overwrite
- Insert SD card in Pynq-Z2
- Turn on board

Login

- Petalinux
- Choose password





KU LEUVEN

Install some extra tools

- Add dnf and vim to rootfs
- https://support.xilinx.com/s/question/0D54U00006FYXOtSAP/dnf-update-fails-on-petalinux-20222?language=en_US
- Add nano in rootfs
  https://support.xilinx.com/s/question/0D52E00006xnpfBSAQ/add-nano-editor-to-petalinux-20202?language=en_US
- If you did not select openssh or git in the rootfs config:
  - sudo dnf --allowerasing install openssh
  - sudo dnf install git

Install some extra tools

- micro : if CA Certificate (you can select this in rootfs) is not yet installed, a dirty quickfix to ignore certificates:
  - curl -k https://getmic.ro -o install_micro.sh (if you would like to install the micro text editor)
  - Edit the script on line 61 so curl has "-k" added to it and git has -c http.sslVerify=false (not a good idea, I told you it was dirty… it is better to select CA certificate in rootfs)
  - sudo ./install_micro.sh
  - sudo mv ./micro */usr/bin*
  - git -c http.sslVerify=false clone https://gitlab.kuleuven.be/…
- lsiuo :
  - cd ~/Downloads
  - lp -o lsuio-0.2.0.tar.gz
  - tar xzvf lsuio-0.2.0.tar.gz
  - cd lsuio-0.2.0/
  - cat INSTALL
  - ./configure
  - make
  - make check
  - sudo make install
  - lsuio --help

Demo time : embedded linux

- Boot board
- ls
- ip a
- In a terminal ssh petalinux@10.42.0.128
- cat */proc/cpuinfo*
- *uname -a*
- *ls /boot*

devmem

vitis_bd_wrappe | vitisbd_hw_plat | xgpio_example.c | HO4 | dino_swap_syste | dino_swap | helloworld.c

```
#include "xil_printf.h"
#include "sleep.h"
#include "xparameters.h"

#include "xil_io.h"

const u32 dino_mem[2][24] =
{
    {
        0x00000000,0x0000FFE0,0x0001BFF0,0x0001FFF0,0x0001FFF0,0x0001FFF0,
        0x0001FFF0,0x0001FC00,0x0001FF80,0x0001F800,0x0803F000,0x080FF700,
        0x0C3FFD00,0x0E7FF800,0x0FFFF800,0x0FFFF800,0x07FFF000,0x03FFC000,
        0x01FF0000,0x00FE0000,0x00778000,0x00600000,0x00400000,0x00600000
    },
    {
        0x00000000,0x0000FFE0,0x0001BFF0,0x0001FFF0,0x0001FFF0,0x0001FFF0,
        0x0001FFF0,0x0001FC00,0x0001FF80,0x0001F800,0x0803F000,0x080FF700,
        0x0C3FFD00,0x0E7FF800,0x0FFFF800,0x0FFFF800,0x07FFF000,0x03FFC000,
        0x01FF0000,0x00FE0000,0x00660000,0x00320000,0x00020000,0x00030000
    }
};

int main()
{
    u32 a[24];
    int i, j;

    init_platform();

    xil_printf("\n\r ----- Initia
    for(i=0; i<24; i++)
    {
        a[i] = Xil_In32( XPAR_AXI
        xil_printf("\n\r%08x",(un
    }

    j = 0;
```

Problems | Vitis Log | Gu

'CF Debug Virtual Terminal - ARM Cortex-A9 M

- stdio.h
- platform.h
- xil_printf.h
- sleep.h
- xparameters.h
- xil_io.h
- dino_mem : const u32[
- main() : int

GTKTerm - /dev/ttyUSB1 115200-8-N-1

File Edit Log Configuration Control signals View Help

```
HO4_AXI_BRAM:~$ ls /sys/firmware/devicetree/base/
#address-cells  __symbols__  amba_pl  chosen      cpus           fpga-full  name          replicator
#size-cells     aliases      axi      compatible  fixedregulator  memory     pmu@f8891000
HO4_AXI_BRAM:~$ ls /sys/firmware/devicetree/base/amba_pl/
#address-cells  #size-cells  axi_bram_ctrl@40000000  compatible  name  phandle  ranges
HO4_AXI_BRAM:~$ sudo devmem 0x40000000 32
0x00000000
HO4_AXI_BRAM:~$ sudo devmem 0x40000004 32
0x0000FFE0
HO4_AXI_BRAM:~$ sudo devmem 0x40000008 32
0x0001BFF0
HO4_AXI_BRAM:~$ sudo devmem 0x40000050 32
0x00778000
HO4_AXI_BRAM:~$ sudo devmem 0x40000050 32 0x00660000
HO4_AXI_BRAM:~$ sudo devmem 0x40000050 32
0x00660000
HO4_AXI_BRAM:~$
```

/dev/ttyUSB1 115200-8-N-1 DTR RTS CTS CD DSR RI

KU LEUVEN

Write to memory with devmem/memmap

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>

// Make the SDK console work in the debugger
#define printf(...) \
 fprintf(stdout, __VA_ARGS__); \
 fflush(stdout);

typedef long long int u64;
typedef long int u32;



const u32 dino_mem[2][24] =
{
{
0x00000000,0x0000FFE0,0x0001BFF0,0x0001FFF0,0x0001FFF0,0x0001FFF0,
0x0001FFF0,0x0001FC00,0x0001FF80,0x0001F800,0x0803F000,0x080FF700,
0x0C3FFD00,0x0E7FF800,0x0FFFF800,0x0FFFF800,0x07FFF000,0x03FFC000,
0x01FF0000,0x00FE0000,0x00778000,0x00600000,0x00400000,0x00600000
},
{
0x00000000,0x0000FFE0,0x0001BFF0,0x0001FFF0,0x0001FFF0,0x0001FFF0,
0x0001FFF0,0x0001FC00,0x0001FF80,0x0001F800,0x0803F000,0x080FF700,
0x0C3FFD00,0x0E7FF800,0x0FFFF800,0x0FFFF800,0x07FFF000,0x03FFC000,
0x01FF0000,0x00FE0000,0x00660000,0x00320000,0x00020000,0x00030000
}
};
```

```
int main()
{
   unsigned int bram_size = 0x2000;
   off_t bram_pbase = 0x40000000; // physical base address
   u64 *bram64_vptr;
   int fd;

   // Map the BRAM physical address into user space getting a virtual address for it
   if ((fd = open("/dev/mem", O_RDWR | O_SYNC)) != -1) {

      bram64_vptr = (u64 *)mmap(NULL, bram_size, PROT_READ|PROT_WRITE,
                                MAP_SHARED, fd, bram_pbase);

    int f = 0;
    while(1){
              for (int i=0; i<24; i++){
                 u64 low, high;
                 low = dino_mem[f][i++];
                 high = dino_mem[f][i];
                 u64 hnew = low | (high<<32);
                 bram64_vptr[i/2] = hnew;
              }
              sleep(1);
              f^=0x01;
    }

    close(fd);

  }

 }
```

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/ho4_hdmi_devmem.git

Demo time : /dev/mem

- cd ho4_hdmi_devmem
- micro src/main.cpp
- make
- ./build/dino_run

Code Review /dev/mem

- What are the issues when using devmem/mmap?
- You need sudo
- You can write everywhere in memory
- This is not safe!
- Better to use a specific device which is memory mapped to a restricted area of the memory, this is why we will use UIO

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842412/Accessing+BRAM+In+Linux

Device-tree

https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.4

- On a target linux machine:
  - ls /sys/firmware/devicetree/base
    (contains a representation of the devicetree contents)
- Generate on a live system with:
  - dtc -I fs /sys/firmware/devicetree/base

```
auart0: serial@8006a000 {
        Defines the "programming model" for the device. Allows the
        operating system to identify the corresponding device driver.
        compatible = "fsl,imx28-auart", "fsl,imx23-auart";
        Address and length of the register area.
        reg = <0x8006a000 0x2000>;
        Interrupt number.
        interrupts = <112>;
        DMA engine and channels, with names.
        dmas = <&dma_apbx 8>, <&dma_apbx 9>;
        dma-names = "rx", "tx";
        Reference to the clock.
        clocks = <&clks 45>;
        The device is not enabled.
        status = "disabled";
};
```

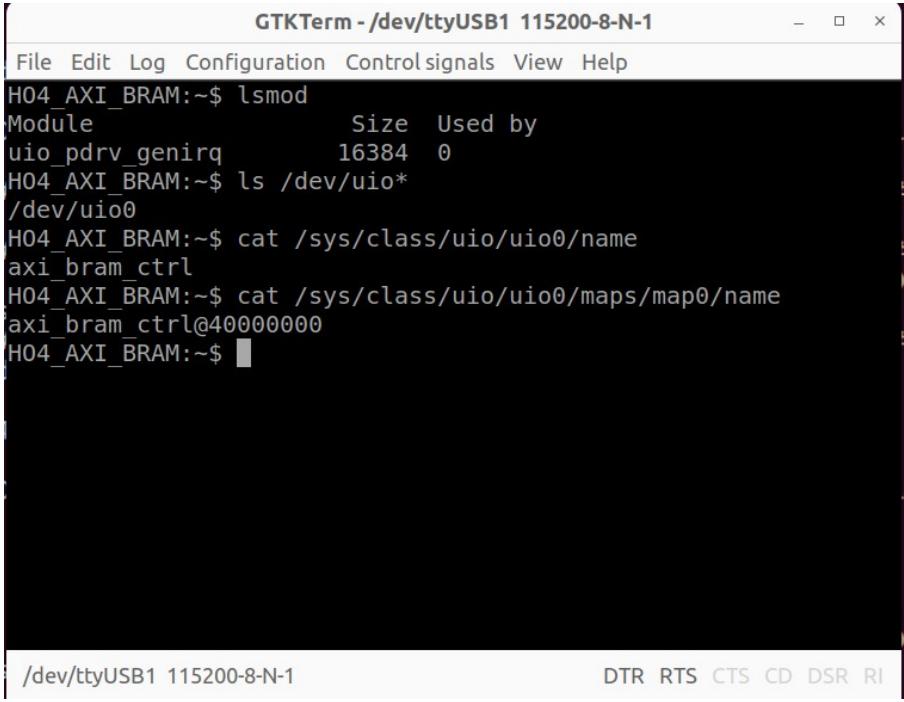
uio

https://www.kernel.org/doc/html/v4.17/driver-api/uio-howto.html#using-uio-pdrv-genirq-for-platform-devices

- Check if uio kernel driver is enabled in kernel config
- Rebuild and boot

- Decompile and inspect devicetree
  dtc -I dtb -O dts -o system.dts system.dtb

  micro system.dts
- cat /proc/device-tree/chosen/bootargs
  - earlycon clk_ignore_unused root=/dev/mmcblk0p2 rw rootwait cma=1024M uio_pdrv_genirq.of_id=generic-uio
- lsmod
- Manual loading: modprobe uio_pdrv_genirq.ko of_id="generic-uio"
- This will load the generic kernel driver that allows us to use a uio device.



uio

- In the previous code:
  - Change /dev/mem to */dev/uio0*
  - *Change bram_pbase address to 0*

- *Compile and run without sudo (quickfix to test, will not work after reboot)*
  - *sudo chmod 666 /dev/uio0*
- *Permanent (this is the right way) solution for chmod:*
  - *create the file /etc/udev/rules.d/10-local.rules and insert the line:*

    KERNEL==”uio0", MODE=”666"

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/ho4_hdmi_uio.git

Demo time : /dev/uio0

- lsuio
- cd ho4_hdmi_uio
- micro src/main.cpp
- make
- ./build/dino_run

Code review /dev/uio

- This is better, no administrator rights needed
- MMU keeps us in safe memory area
- What happens when we do try to access another part of the memory? → SEGFAULT
- Not for realtime data retrieval, in that case we need kernel modules

https://fpga.org/2013/05/28/how-to-design-and-access-a-memory-mapped-device-part-two/

Questions?

Level vs Edge interrupts

- https://discuss.pynq.io/t/axi-intc-output-level-or-edge-for-pynq/947
- https://www.garystringham.com/level-triggered-vs-edge-triggered-interrupts/

IRQ ID#	Name	PPI#	Type	Description
26:16	Reserved	~	~	Reserved
27	Global Timer	0	Rising edge	Global timer
28	nFIQ	1	Active Low level (active High at PS-PL interface)	Fast interrupt signal from the PL: CPU0: IRQF2P[18] CPU1: IRQF2P[19]
29	CPU Private Timer	2	Rising edge	Interrupt from private CPU timer
30	AWDT{0, 1}	3	Rising edge	Private watchdog timer for each CPU
31	nIRQ	4	Active Low level (active High at PS-PL interface)	Interrupt signal from the PL: CPU0: IRQF2P[16] CPU1: IRQF2P[17]

Source	Interrupt Name	IRQ ID#	Status Bits (mpcore Registers)	Required Type	PS-PL Signal Name
PL	PL [2:0]	63:61	spi_status_0[31:29]	Rising edge/ High level	IRQF2P[2:0]
	PL [7:3]	68:64	spi_status_1[4:0]	Rising edge/ High level	IRQF2P[7:3]
Timer	TTC 1	71:69	spi_status_1[7:5]	High level	~
DMAC	DMAC[7:4]	75:72	spi_status_1[11:8]	High level	IRQP2F[27:24]
IOP	USB 1	76	spi_status_1[12]	High level	IRQP2F[7]
	Ethernet 1	77	spi_status_1[13]	High level	IRQP2F[6]
	Ethernet 1 Wake-up	78	spi_status_1[14]	Rising edge	IRQP2F[5]
	SDIO 1	79	spi_status_1[15]	High level	IRQP2F[4]
	I2C 1	80	spi_status_1[16]	High level	IRQP2F[3]
	SPI 1	81	spi_status_1[17]	High level	IRQP2F[2]
	UART 1	82	spi_status_1[18]	High level	IRQP2F[1]
	CAN 1	83	spi_status_1[19]	High level	IRQP2F[0]
PL	PL [15:8]	91:84	spi_status_1[27:20]	Rising edge/ High level	IRQF2P[15:8]

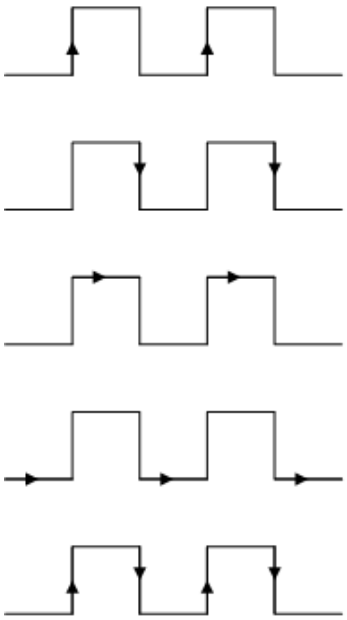
Level vs Edge interrupts

```
The #interrupt-cells property is set to 2 and the first cell defines the
index of the interrupt within the controller, while the second cell is used
to specify any of the following flags:
  - bits[3:0] trigger type and level flags
      1 = low-to-high edge triggered
      2 = high-to-low edge triggered
      4 = active high level-sensitive
      8 = active low level-sensitive

Example:

      i2c@7000c000 {
              gpioext: gpio-adnp@41 {
                      compatible = "ad,gpio-adnp";
                      reg = <0x41>;

                      interrupt-parent = <&gpio>;
                      interrupts = <160 1>;
```



https://github.com/Xilinx/linux-xlnx/blob/master/Documentation/devicetree/bindings/interrupt-controller/interrupts.txt

Level vs Edge interrupts

- Enable GIE, bit 31 of Global IRQ (offset 0x11C)
- Enable Channel 1 IE of IRQ_CONTROL(offset 0x128)
- Clear the status flag IRQ_STATUS (offset 0x120) by writing 0x1 to the register
  See AXI GPIO Product Guide:

Table 2-4: **Registers**

Address Space Offset[3]	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER[1]	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER[1]	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR[1]	R/TOW[2]	0x0	IP Interrupt Status Register.

Global Interrupt Enable Register (GIE)

The Global Interrupt Enable register is a 32-bit read/write register (This Figure). The Global Interrupt Enable Register provides the master enable/disable for the interrupt output (IP2Intc_Irpt signal) to the processor. The bit definition of this register is shown in Table: Global Interrupt Enable Register (0x07F8) .

Figure: Global Interrupt Enable

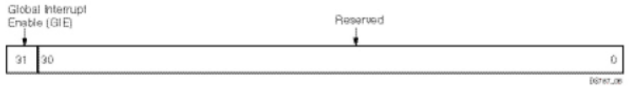
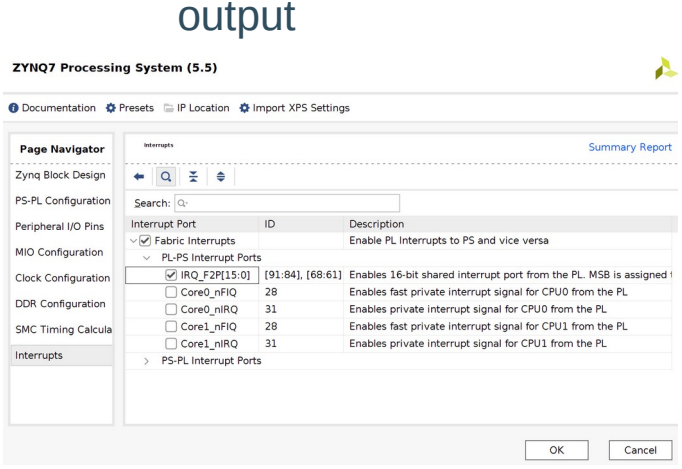


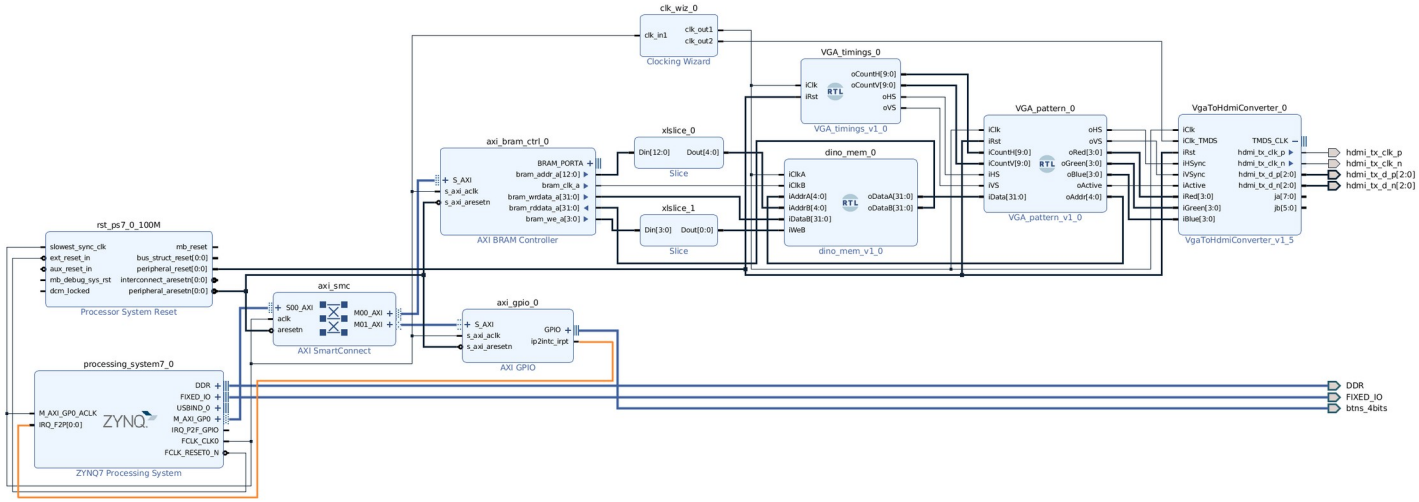
Table: Global Interrupt Enable Register (0x07F8)

Bits	Name	Access	Reset value	Description
31	GIE	Read/Write	0	Global Interrupt Enable bit
30:0	Reserved	N/A	N/A	Reserved

uio using interrupts

- Use a gpio button to create interrupts in the zynq processing system that are processed using axi gpio. Add AXI Gpio IP. Enable ZynqPS PL-PS interrupt.
- Enable the zynq interrupts in your hardware design and connect a wire to the gpio interrupt output





git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/ho4_hdmi_uio_int.git

uio using interrupts

- In petalinux, change the devicetree by editing the file in your petalinux project micro project-spec/meta-user/recipes-bsp/device-tree/filesystem-conf.dtsi
  - Add the uio_pdrv_genirq to load the generic uio kernel driver on bootup (same as previous example)
  - Set generic-uio as the compatible string of the axi_gpio_o device
  - Change the interrupt so it is triggered on edge instead of level, the third parameter of the interrupts attribute indicates what is triggering the interrupt.

```
/include/ "system-conf.dtsi"
/ {
    chosen {
        bootargs = "earlycon clk_ignore_unused earlyprintk root=/dev/mmcblk0p2 rw rootwait cma=1024M uio_pdrv_genirq.of_id=generic-uio";
    };
};

&axi_gpio_0 {
    compatible = "generic-uio";
    interrupts = <0 29 1>;
};
```

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/ho4_hdmi_uio_int.git

uio using interrupts

- Git clone the project
- Make
- ./build/dino_run

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/ho4_hdmi_uio_int.git

uio using interrupts – further reading

- https://forum.digilent.com/topic/4147-uio-interrupts-on-zynq/
- https://r4nd0m6uy.ch/interrupt-driven-user-space-application-with-the-uio-driver.html
- https://xillybus.com/tutorials/device-tree-zynq-4

At last! Kernel modules

- https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/96960682/Linux+Loadable+Kernel+Modules

- Different ways to add kernel modules to your system:
  - Build in a running system
  - Build from within petalinux
  - Cross compile on your laptop

Kernel modules – running system

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/hox_myleds_kernel.git

(This is an example of a kernel module **without interrupts**)

- Building on a live systen (not by default possible with redhat flavors because it lacks complete kernel source code, download linux-xlnx for this, see down here):

- Add kernel-devsrc to your petalinux rootfs by editing project-spec/meta-user/conf/user-rootfsconfig
  - Add a line that contains : CONFIG_kernel-devsrc
  - Add another line that contains : CONFIG_myleds
- Then in petalinux-config -c rootfs → goto user-packages and select kernel-devsrc, in modules select myleds
- petalinux-build/package, write sd card and boot the system
- Download linux-xlnx source code
  - cd ~/Downloads && Wget https://github.com/Xilinx/linux-xlnx/archive/refs/heads/xlnx_rebase_v5.15_LTS_2022.1_update.zip
  - unzip xlnx_rebase_v5.15_LTS_2022.1_update.zip
  - mv xlnx_rebase_v5.15_LTS_2022.1_update/ linux-xlnx/
  - cd linux-xlnx && ln -s /lib/modules/5.15.19-xilinx-v2022.1/build ./build
- Git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/hox_myleds_kernel.git
- make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-

Kernel modules – running system

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/hox_myleds_kernel.git

- Test the system:
  - lsmod
  - sudo rmmod myled && sudo rmmod myleds
  - sudo insmod myled.ko
  - ls /proc # you should see myled in the listing
  - dmesg
  - Press and hold a button
  - cat /proc/myled # Value of the pressed button is printed to the console
  - echo *0x1 > /proc/myled # Led turns on*
  - *cp ./myled.ko /lib/modules/5.15.19-xilinx-v2022.1/extra/ # let the system load the module on bootup*

https://docs.xilinx.com/api/khub/documents/FelAnL4Ho13avhukyj2vLw/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=4.2.12#page=51&zoom=100,122,333

p54, p58

Kernel modules – petalinux-create

(Example of a kernel module **with interrupts**)

- Creating and building a kernel module from within petalinux:
- Create a petalinux application of type module
  - petalinux-create -t modules --name myleds --enable
- This gives you an example of an interrupt driven kernel module
  - Look at project-spec/meta-user/recipes-modules/myleds/files/
- Add the correct device-tree node with the correct compatible attribute (compatible = “vendor,myleds”)
- Petalinux-build -c myleds -x do_install
- Petalinux-build
- Package and write to sd card
- Boot the system

https://docs.xilinx.com/api/khub/documents/FelAnL4Ho13avhukyj2vLw/content?Ft-Calling-App=ft%2Fturnkey-portal&Ft-Calling-App-Version=4.2.12#page=51&zoom=100,122,333

p54, p58

Kernel modules – petalinux-create

(Example of a kernel module **with interrupts**)

- Test the system
  - lsmod # shows all loaded kernel modules, myleds should be there
  - ls /lib/modules/5.15.19-xilinx-v2022.1/extra/ # Lists the kernel modules built by petalinux, myleds.ko should be there
  - rmmod myleds # remove a kernel module
  - insmod /lib/modules/5.15.19-xilinx-v2022.1/extra/myleds.ko # Loads the kernel module
  - Check the status with dmesg
- Enable interrupts:
  - sudo devmem 0x4120011C 32 0x80000000 # This enables the Global Interrupt Controller
  - sudo devmem 0x41200128 32 0x1 # This enables the GPIO Interrupt Controller
  - sudo devmem 0x41200120 32 0x1 # This clears the interrupt
  - dmesg # Look at system messages
  - Push a button
  - dmesg
  - sudo devmem 0x41200120 32 0x1 # Clear interrupt

Kernel modules – cross-compile

git clone git@gitlab.kuleuven.be:groep-t/courses/rndembed/2324/petalinux/hox_myleds_kernel.git

- Cross compiling
  - Clone the git repository
  - Download linux-xlnx git switch to correct version (2022.1_update)
  - apt install gcc-arm-xilinx-linux-gnueabi
  - make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
- Copy the myled.ko file to the sd card and boot the system
- insmod myled.ko
- echo 0x0F > /proc/myled
- Read more : p88 UG1144 [Creating and adding custom kernel modules]

User application that uses Kernel Module

- Let’s build a user application that uses this kernel module.

  Cd ~/hox_myleds_kernel/led_blinks
  vim led_blink.c # Look at the source code

  vim Makefile # Look at the Makefile

  Compile led_blink:
  make
  ls

  lsmod # Make sure you have loaded the correct kernel module, if not rmmod and insmod to fix it.

  ./led_blink # The led should start blinking
  ^C

AXI DMA

- https://support.xilinx.com/s/question/0D52E00006iI3fFSAS/axidma-driver-using-uio?language=en_US

Device Tree Overlays (Pynq)

- https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841645/Solution+Zynq+PL+Programming+With+FPGA+Manager

tftpboot

- Build a u-boot image that always boots via tftpboot when available

https://support.xilinx.com/s/article/763309?language=en_US

https://support.xilinx.com/s/article/000033588?language=en_US

tftpboot – uEnv.txt

- bin=u-boot
- [illegible] fdt_addr=0x88f00000
- uenvcmd=fdt addr $
-
- kernel_image=Image.gz
- kernel_load_address=0x80080000
- devicetree_image=dtb/am62x-[illegible].dtb
- devicetree_load_address=0x88000000
- fdtdevice_image=[illegible].dtb
- boot_image=boot.scr
- loadaddr [illegible]=0x80800000
- loadfdtaddr [illegible]=0x88000000
- kernel_boot=0x80080000
- devicetree_boot=0x88000000
- boot_boot=0x80800000
- overlay_image=overlay.gz
- overlay_load_address=0x88080000
- overlay_boot=0x88080000
-
- tftp_path=0x88d80000
-
- tftp_addr=0x90100000
-
- tftp_kernel=Image.gz
- tftp_dtb=dtb/[illegible].dtb
-
- loadtftpoverlay
-
- tftp_boot=fdtaddr $fdt_addr && tftpboot $kernel_addr $kernel_image && tftpboot $fdt_addr $devicetree_image && run loadtftpoverlay && fdt addr $fdt_addr && fdt resize 0x100000 && for overlay in $overlays ; do tftpboot $overlay_addr $overlay_image && fdt apply $overlay_addr ; done && booti $kernel_addr - $fdt_addr
- tftp_load=tftpboot $kernel_load_address $kernel_image && tftpboot $devicetree_load_address $devicetree_image && run loadtftpoverlay && fdt addr $devicetree_load_address && fdt resize 0x100000 && for overlay in $overlays ; do tftpboot $overlay_load_address $overlay_image && fdt apply $overlay_load_address ; done && booti $kernel_load_address - $devicetree_load_address
- uenvcmd=setenv autoload no; tftp ${tftp_dtb_prefix}${fdtfile}; run tftp_boot

If you want to use tftpboot, change your zynq-common.h:

```
"sdboot=echo Copying Linux from Network to RAM... && " \
        "tftpboot ${kernel_load_address} 10.10.70.101:${kernel_image} && " \
        "tftpboot ${devicetree_load_address} 10.10.70.101:${devicetree_image} && " \
        "tftpboot ${ramdisk_load_address} 10.10.70.101:${ramdisk_image} && " \
        "bootm ${kernel_load_address} ${ramdisk_load_address} ${devicetree_load_address};\0" \
```

Interesting tools for your host

- sudo apt-get install git
- sudo apt-get install libssl-dev
- sudo apt-get install gcc-arm-linux-gnueabi
- sudo apt-get install device-tree-compiler
- sudo apt-get install device-tree-overlay