

PetaLinux installation and usage instructions

20. lokakuuta 2020

Sisältö

1	Starting Linux	3
2	Starting Vivado	3
3	Design hardware	3
3.1	Download the base HW design	4
3.2	Upgrade and compile the base design	4
4	Make Linux project from board support package	5
5	Make Linux project from own HW using Petalinux 2019.2	5
5.1	Configure and Compile linux	5
5.2	Some usefull configuration options	6
5.3	Build the linux system	7
5.4	Test the system in QEMU emulator	8
5.5	Booting over JTAG	8
5.6	Install linux in SD-card	8
5.7	Upload bitstream and device tree separately	10
5.8	Test booting in a real system	11
5.9	Debugging	12
5.10	Manual boot	12
6	Test the linux	13
7	Debootstrap your own linux	13
8	Use custom IP	13
9	Load a system design to FPGA using FPGA manager	15
10	Install Petalinux	15
10.1	Download	15
10.2	Extract and install	16
10.3	Update the project with your own HW	17

11 Acronyms	17
12 References	18

1 Starting Linux

The Petalinux compilation needs to be made in a Linux environment. The five computers in the middle of the classroom TF4110 and the teachers computer can now be booted to Linux (Ubuntu 18.04) in following way:

1. Restart the computer
2. When the display is still dark, start clicking the **F9** button, until you see a boot method selection menu.
3. From this menu, select **INTEL BOOT OPTION**, it will launch a network boot from the Linux terminal server
4. Then just wait (or click Enter every time possible if you do not want to wait) and the Linux should be up and running about two minutes

2 Starting Vivado

1. Click the the Grid icon in the left bottom corner of the display. You will see a list of application that you can start. Find a Terminal and start it. You can also write terminal in the search field, and you may find the terminal application faster.
2. Write **vivado &** in the terminal window. It will start the vivado in the bacground, and lets you use the terminal for other purposes. Prepare to wait for a couple of minutes, Vivado is huge and takes time to be started from the slow server.
3. If you are running vivado the first time here, run first **TOOLS -> DOWNLOAD LATEST BOARDS**. If you do not do this, the synthesis of designs for Zybo boards will fail. It can take suprisingly long time to run first time (like 5-10 minutes) but next time it will be fast. So if you are uncertain, run it.

3 Design hardware

It is a good idea to implement certain HW components in the FPGA to fully utilize all resources of it from the Linux side. Even the base system providing all needed components is rather complex, and therefore it is good to get it as ready as possible. Unfortunately the Digilent reference design is a bit old, and the IP cores needs upgrading and one of the is totally missing. But it is still pretty good base.

3.1 Download the base HW design

Download first the Digilent reference design, which is a good base for running linux in Zybo. Do not forget the `-recursive` parameter. Otherwise needed IP repositories are not downloaded!

```
1 # This is for Zybo Z7-10 board
2 git clone --recursive https://github.com/Digilent/Zybo
   ↳ -Z7-10-base-linux.git
3
4 # And take this if you have Zybo Z7-20 board
5 git clone --recursive https://github.com/Digilent/Zybo
   ↳ -Z7-20-base-linux.git
```

3.2 Upgrade and compile the base design

The repository does not have a Vivado project at all, but it includes a TCL-script to create one.

1. Create Vivado project
 - (a) Open vivado and select `RUN TCL SCRIPT`
 - (b) Browse and run the `create_project.tcl` script from the `proj` sub-directory
 - (c) The project will be created and opened in Vivado
2. Upgrade IP:
 - (a) Select `REPORTS -> REPORT IP STATUS`
 - (b) From the IP status window, click `UPGRADE ALL SELECTED` to upgrade all old IP
 - (c) It is possible that one IP block cannot be upgraded in Zybo-20 project. In that case delete it. Let's hope that it was not important
3. Create HDL wrapper of the system block diagram, by right-clicking it in the sources-window, and selecting `CREATE HDL WRAPPER`.
4. Synthesize and generate the Bitstream
5. Export the hardware `FILE->EXPORT->EXPORT HARDWARE`, include the Bitstream. Export for example in directory `Linux-Zybo10-HW`

4 Make Linux project from board support package

```
1 # Import the settings from the Petalinux configuration
  ↳ file
2 source /opt/petalinux/2017.4/settings.sh
3
4 # Create new petalinux project
5 petalinux-create --type project --name Linux-Zybo10-SW
  ↳ -s /opt/data/BSP/Petalinux-Zybo-Z7-10-2017.4-1.
  ↳ bsp
6 cd Linux-Zybo10-SW
7
8 # Build the Linux kernel, boot manager and root file
  ↳ system with default settings
9 petalinux-build
10
11 # Package the binaris redy to be installed in SD-card
12 petalinux-package --boot --force --fsbl ./images/linux
  ↳ /zynq_fsbl.elf --fpga --u-boot
```

5 Make Linux project from own HW using Petalinux 2019.2

To use the PetaLinux-tools, you need to first configure paths and other environment by running the setup script:

```
1 # Import the settings from the Petalinux configuration
  ↳ file
2 source /opt/petalinux/2019.2/settings.sh
3
4 # Create new petalinux project
5 petalinux-create --type project --template zynq --name
  ↳ Linux-Zybo10-SW
6 cd Linux-Zybo10-SW
7
8 # Now import your latest HW into your Linux design
9 petalinux-config --get-hw-description=./Linux-Zybo10-
  ↳ HW
```

5.1 Configure and Compile linux

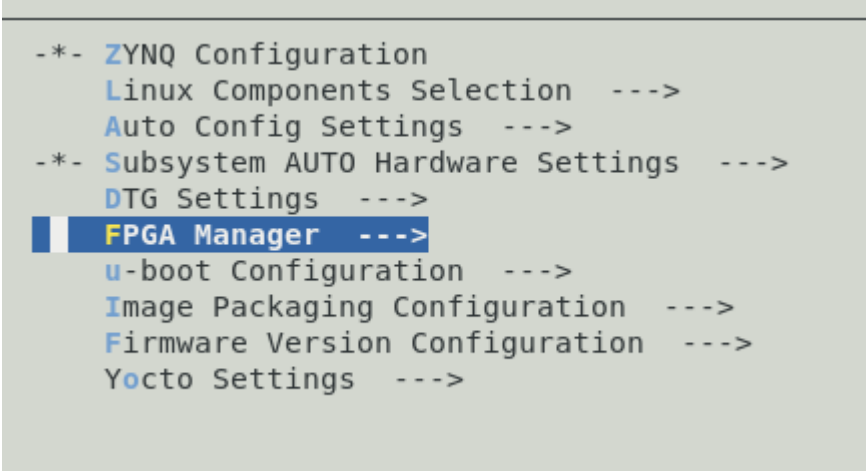
If you do this first time, skip all additional configurations listed below and continue from build in subsection 5.3 . Just go with the defaults and try to make

build successful and your system to boot. You can run further configurations later when you start needing them.

```
1 # Subsystem level configuration (Select FPGA manager)
2 $ petalinux-config
3
4 # Configure boot process (Skip)
5 $ petalinux-config -c u-boot
6
7 # Configure kernel: (Skip, already done in the first
  ↳ stage)
8 $ petalinux-config -c kernel
9
10 # Configure rootfs:
11 $ petalinux-config -c rootfs
12
13 # Configure the device tree (Skip)
14 $ petalinux-config -c device-tree
```

5.2 Some useful configuration options

Activate FPGA-manager from the main config. It allows the configuration of the FPGA from the Linux side.



```
-*- ZYNQ Configuration
  Linux Components Selection --->
  Auto Config Settings --->
  *- Subsystem AUTO Hardware Settings --->
    DTG Settings --->
    FPGA Manager --->
    u-boot Configuration --->
    Image Packaging Configuration --->
    Firmware Version Configuration --->
    Yocto Settings --->
```



```
[*] Fpga Manager
  ( ) Specify hw directory path (NEW)
```

The Rootfs configuration selects which applications will be included in the ready made Linux system. If you want to have very minimal system, you do not need to activate any of these. I activated plenty:

- *matchbox*: This is a simple desktop environment for Linux
- *networking-stack*: Activate this if you need networking.
- *OpenCV*: This is a common image processing module.
- *python-modules*: Activate this if you need python
- *utils*: Miscellaneous utilities
- *v4lutils*: Video for linux, activate this if you need camera
- *x11*: X11 windowing system, activate this if you need graphical desktop

```

packagegroup-petalinux --->
packagegroup-petalinux-display-debug --->
packagegroup-petalinux-lmsensors --->
packagegroup-petalinux-matchbox --->
packagegroup-petalinux-networking-debug --->
packagegroup-petalinux-networking-stack --->
packagegroup-petalinux-openamp --->
packagegroup-petalinux-opencv --->
packagegroup-petalinux-python-modules --->
packagegroup-petalinux-qt --->
packagegroup-petalinux-qt-extended --->
packagegroup-petalinux-self-hosted --->
packagegroup-petalinux-utils --->
packagegroup-petalinux-v4lutils --->
packagegroup-petalinux-x11 --->

```

5.3 Build the linux system

```

1 #Build the PetaLinux system (kernel, uboot and rootfs)
2   petalinux-build
3
4 #Package it ready to be installed
5 petalinux-package --boot --force --fsbl ./images/linux
   ↳ /zynq_fsbl.elf --fpga ./images/linux/system.bit
   ↳ --u-boot --dtb ./images/linux/system.dtb
6
7 # Package prebuilt directrory
8 petalinux-package --prebuilt --fpga images/linux/
   ↳ system.bit --force

```

5.4 Test the system in QEMU emulator

Leave from QEMU by pressing CTRL-A-x!

```
1 # Test the boot loader
2     petalinux-boot --qemu --u-boot
3
4 # Test the Linux kernel
5     petalinux-boot --qemu --kernel
6
7 # Test the whole prebuild system in emulator
8     petalinux-boot --qemu --prebuilt 3
```

5.5 Booting over JTAG

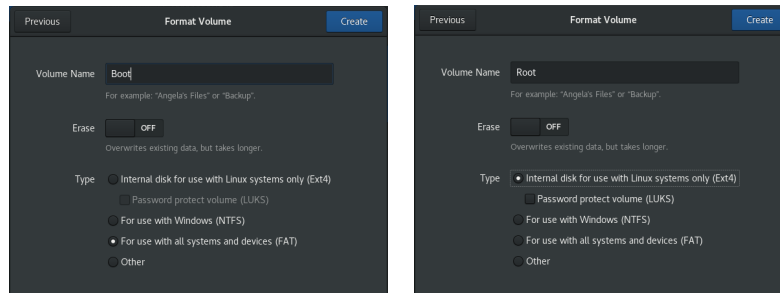
```
1 # Load the boot loader in the system and boot it
2     petalinux-boot --jtag --u-boot
3
4 # Load the kernel in the system and try to use that
5     petalinux-boot --jtag --kernel
6
7 # Load the whole prebuild system and boot it
8     petalinux-boot --jtag --prebuilt 3
```

5.6 Install linux in SD-card

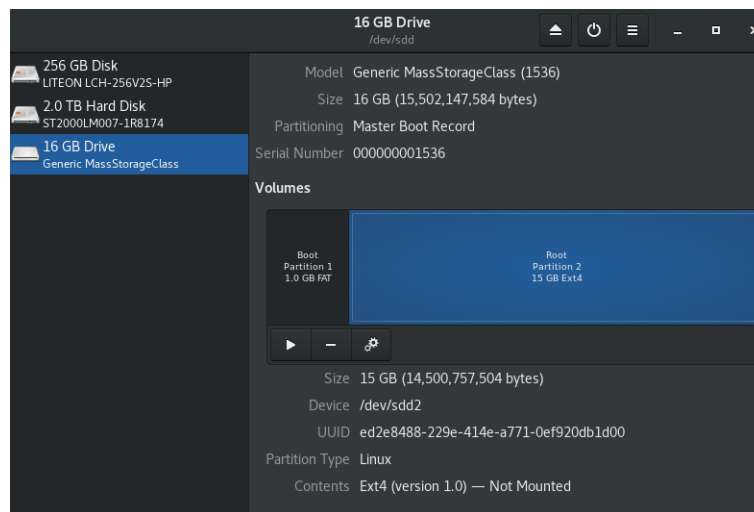
First you need to prepare the SD card by partitioning and formatting it, if you haven't done it already. The new SD-card is prepared like this:

1. Open the disk management tool `gnome-disks` from the terminal. Or you can also open it from the list of applications if you prefer. You have permissions to edit only the USB-disks, but be still carefull. Edit only the 16 GB size disk, and not any other hard disks you see!
2. Delete existing partition from the USB disk.
3. Create a 1 GB FAT formatted partition for boot images, as shown in Figure (1).
4. Create a Ext4 formatted root partition, extending over the rest of the disk for filesystem room, as shown in Figure (1).

Now the USB disk is ready, and should look like in Figure (2). You should see the mounted partitions Boot and Root in the file manager window. If you do not see them, remove the USB disk and insert it back.



Kuva 1: Boot and root partition



Kuva 2: The resulting USB stick partitions

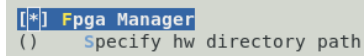
When you have one file manager window open, go with a terminal window to the directory where your petalinux project is. Start another file manager into that directory by entering `nautilus . &` in the terminal. Copy following files:

1. `images/linux/BOOT.BIN` into the `Boot` partition of the USB disk
2. `images/linux/image.ub` into the `Boot` partition of the USB disk
3. `images/linux/rootfs.cpio` into the `Root` partition of the USB disk

Now the SD image is ready. Unmount the memory stick, plug it in to the Zybo and boot a system from it.

5.7 Upload bitstream and device tree separately

Activate the FPGA manager under Activate FGPA manager:



More instructions from Xilinx wiki

```
1
2 # Make first a system.bif file, with following
   ↳ content
3 all:
4 {
5     system.bit /* Bitstream file name */
6 }
7
8 # Convert Bitstream to bin
9 bootgen -image system.bif -arch zynq -
   ↳ process_bitstream bin
10
11 # Compile device tree overlay
12 dtc -O dtb -o pl.dtbo -b 0 -@ pl.dtsi
```

Configure the system with new FPGA overlay, according to instructions.

```

1 echo 0 > /sys/class/fpga_manager/fpga0/flags
2 cp /media/design_1_wrapper.bit.bin /lib/firmware/
   ↪ design_1_wrapper.bit.bin
3 cp /media/pl.dtbo /lib/firmware/
4
5 mkdir /configfs
6 mount -t configfs configfs /configfs
7 cd /configfs/device-tree/overlays/
8 mkdir full
9 echo -n "pl.dtbo" > full/path
10
11
12 ## Or use FPGA utils
13 fpgautil -o /lib/firmware/base/pl.dtbo -b /lib/
   ↪ firmware/base/design_1_wrapper.bit.bin

```

5.8 Test booting in a real system

1. Insert the SD-card in the ZYbo
2. Switch the Boot selector jumper in the leftmost (SD-card) position
3. Connect USB cable
4. Open terminal, eg minicom or putty to /dev/ttyUSB1, 115200 baud by using the command `minicom -s` in a terminal.
 - (a) Select Serial port setup from the menu
 - (b) Press **A** and edit the serial device string to `/dev/ttyUSB1`, and then press Enter.
 - (c) Exit from configuration by selecting **Exit** from the menu
 - (d) Now you are in the Linux shell. Hit Enter if you see nothing
 - (e) Log in using username `root` and password `root`

5.9 Debugging

```
1  # Start gdb session
2      petalinux-util --gdb
3
4  # Starts a debug session with QEMU
5      petalinux-util --xsdb-connect HOST:PORT
6
7  # Open a console to display stdout from linux kernel
   ↪ in the hw (printk output)
8      petalinux-util --jtag-logbuf -i <kernelimg>
```

5.10 Manual boot

If the kernel does not boot automatically, connect to the console with minicom and boot it manually

```
1  mmcinfo
2  fatload mmc 0 0x1000000 image.ub
3  bootm 0x1000000
```

6 Test the linux

```
1  # Check network status
2      ifconfig
3
4  # Try the leds behind GPIO interface
5  # Find out GPIO chips supported
6  ls /sys/class/gpio
7      export      gpiochip906  unexport
8
9  cd /sys/class/gpio/gpiochip906/
10 cat label base ngpio
11     zynq_gpio
12     906
13     118
14 # This shows that the zynq-gpio controls 118 io pins
15 echo 906 > export
16 cd gpio906
17 cat direction
18 cat value
19 echo out > direction
20 cat 1 > value
21
22 gpioutil -d
```

7 Debootstrap your own linux

```
1 debootstrap --arch=ARM stable /mnt/target/ http://www
   ↪ .nic.funet.fi/debian/
```

8 Use custom IP

1. Add custom IP with Vivado.
2. Synthesize
3. Generate bitstream
4. Export HW as usual

Petalinux should be able to read your custom HW as well and generate a device tree from it, but there seems to be problems with it. Therefore you need to exclude the device tree from compilation and compile device tree separately as follows:

1. Configure Petalinux as instructed before, but select "exclude PL DTD"
2. Build petalinux

```
1 git clone https://github.com/Xilinx/device-tree-xlnx.  
    ↪ git  
2 /opt/Vitis/2019.2/bin/xsct  
3 #/opt/petalinux/2019.2/tools/xsct/bin/xsct  
4 xsct% hsi open_hw_design ../ownAdder/  
    ↪ lohkaavio_wrapper.xsa  
5 xsct% hsi set_repo_path ../device-tree-xlnx  
6 xsct% hsi create_sw_design device-tree -os  
    ↪ device_tree -proc ps7_cortexa9_0  
7 xsct% hsi generate_target -dir my_dts  
8  
9 make ARCH=arm CROSS_COMPILE=arm-none-eabi-  
    ↪ imx_v6_v7_defconfig  
10 make ARCH=arm CROSS_COMPILE=arm-none-eabi- dtbs  
11  
12 gcc -I my_dts -E -nostdinc -undef -D__DTS__ -x  
    ↪ assembler-with-cpp -o my_dts/system-top.dts.tmp  
    ↪ my_dts/system-top.dtsst  
13 dtc -I dts -O dtb -o system-top.dtb system-top.dts  
14  
15 ./build/tmp/work/plnx_zynq7-xilinx-linux-gnueabi/  
    ↪ linux-xlnx/4.19-xilinx-v2019.2+gitAUTOINC+  
    ↪ b983d5fd71-r0/linux-plnx_zynq7-standard-build/  
    ↪ scripts/dtc
```

Now the device

9 Load a system design to FPGA using FPGA manager

```
1 # Install the default design to FPGA
2 # First copy it to /lib/firmware (in your Zybo linux
  ↳ command line)
3 cp /lib/firmware/base/Linux-Zybo20-HW.bit.bin /lib/
  ↳ firmware/
4
5 # Then just ask the FPGAmanger to upload it to FPGA
  ↳ PL side, byt
6 # just echoing it's filename for the manager
7 root@uusi1702:~# echo Linux-Zybo20-HW.bit.bin > /sys/
  ↳ class/fpga_manager/fpga0/firmware
8 fpga_manager fpga0: writing Linux-Zybo20-HW.bit.bin
  ↳ to Xilinx Zynq FPGA Manager
9 root@uusi1702:~#
```

Bin file can be generated using the following TCL-command in Vivado
write_bitstream -force -bin_file /tmp/

10 Install Petalinux

I you do not have the PetaLinux environment yet, here is how you can install it. The instructions are for Linux, since I am not aware of any method how to use PetaLinux in Windows. I have used plenty of these really good instructions [Getting Started With PetaLinux](#)

10.1 Download

Download the package from Xilinx PetaLinux web site [Xilinx PetaLinux web site](#) and the Reference manual.

```

1 wget https://xilinx-ax-dl.entitlenow.com/dl/ul
   ↳ /2019/10/28/R210258808/petalinux-v2019.2-final-
   ↳ installer.run/dd0431f7077334eff09cf0a270a0a588
   ↳ /5E38488F?akdm=0&filename=petalinux-v2019.2-
   ↳ final-installer.run
2 mv 5E38488F\?akdm\=0 petalinux-v2019.2-final-
   ↳ installer.sh
3 md5sum petalinux-v2019.2-final-installer.sh
4 > 74b27ae60cf50bccae4ca907ff0b97ec petalinux-
   ↳ v2019.2-final-installer.sh

```

Optionally you can also download the Board Support Package (BSP) for zendboard

```

1 wget https://xilinx-ax-dl.entitlenow.com/dl/ul
   ↳ /2019/10/28/R210258787/avnet-digilent-zedboard-
   ↳ v2019.2-final.bsp/301
   ↳ b5b0a0255a994916f7585eb873570/5E3887FB?akdm=0&
   ↳ filename=avnet-digilent-zedboard-v2019.2-final.
   ↳ bsp
2
3 mv 5E3887FB\?akdm\=0 avnet-digilent-zedboard-v2019.2-
   ↳ final.bsp

```

10.2 Extract and install

`./petalinux-v2019.2-final-installer.sh /opt/petalinux/2019.2/`


```

1  # Install dependencies
2  sudo apt install chrpath libtool texinfo zlib1g-dev
   ↳ gcc-multilib zlib1g ncurses-bin gawk ncurses-
   ↳ dev libncurses5-dev lib32z1 lib32z1-dev zlib1g-
   ↳ dev:i386
3  # Install petalinux tools
4  # If you see error awk: read error (Bad address),
   ↳ installa gawk
5
6  sudo mkdir -p /opt/petalinux/2019.2
7  sudo chown ltsp.ltsp /opt/petalinux/2019.2/
8  ./petalinux-v2019.2-final-installer.sh /opt/petalinux
   ↳ /2019.2/

```

10.3 Update the project with your own HW

See DMA-for linuxDMA-for linux video

The PetaLinux project includes the following features by default:

- Ethernet with Unique MAC address and DHCP support (see known issues)
- USB Host support
- UIO drivers for onboard switches, buttons and LEDs
- SSH server
- Build essentials package group for on-board compilation using gcc, etc.
- HDMI output with kernel mode setting (KMS)
- HDMI input via UIO drivers
- U-boot environment variables can be overridden during SD boot by including uEnv.txt in the root directory of the SD card (see u-boot documentation).

11 Acronyms

FSBL First Stage Boot Loader

KERNEL Operating system core

BOOTLOADER A Program which is started first and loads and starts the operating system

ELF Binary file format

DeviceTree Linux specific system hardware description, does not include drivers

BSP Board Support Package: A vendor specific system board hardware description, including the drivers

12 References

1. Petalinux command reference