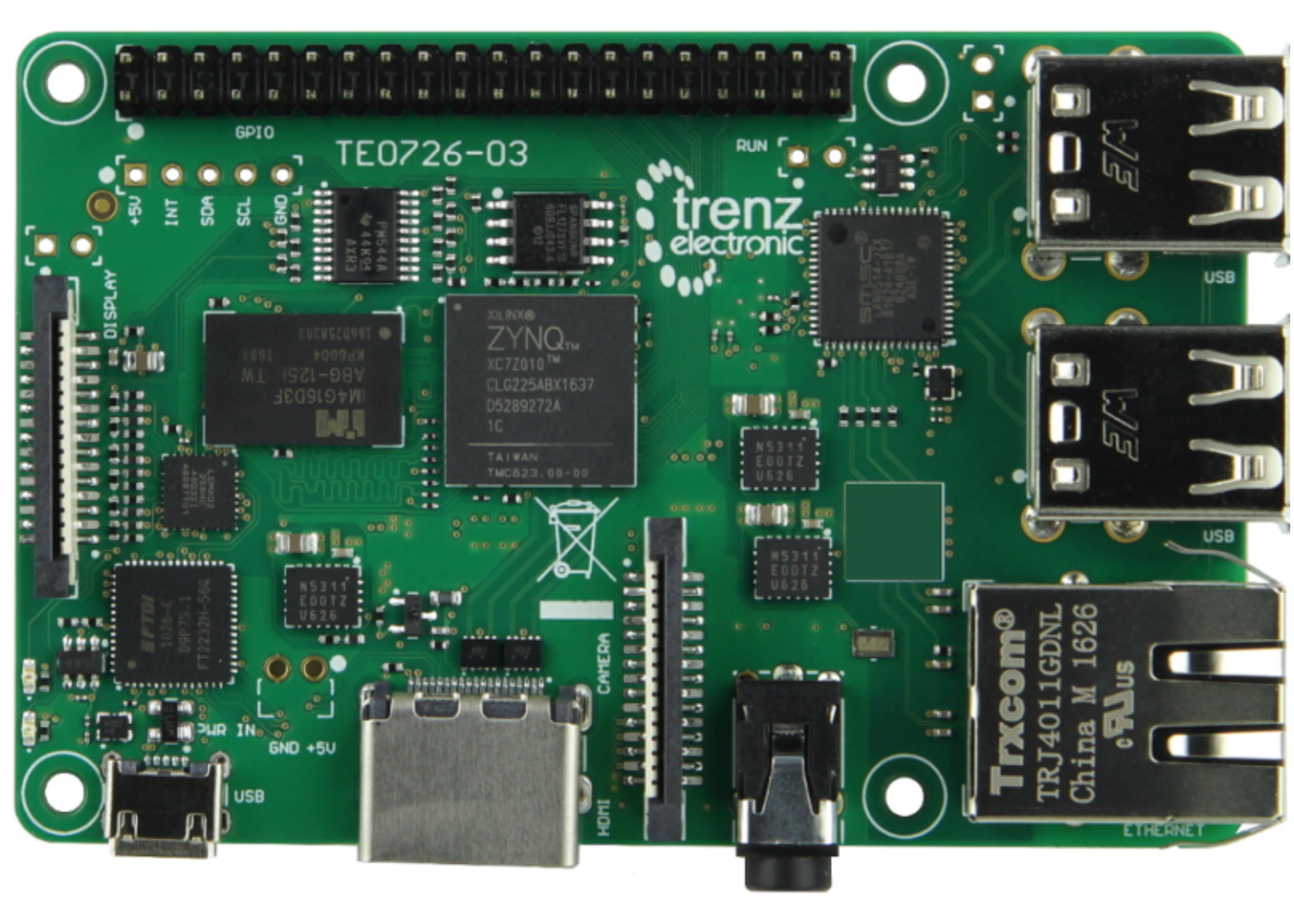
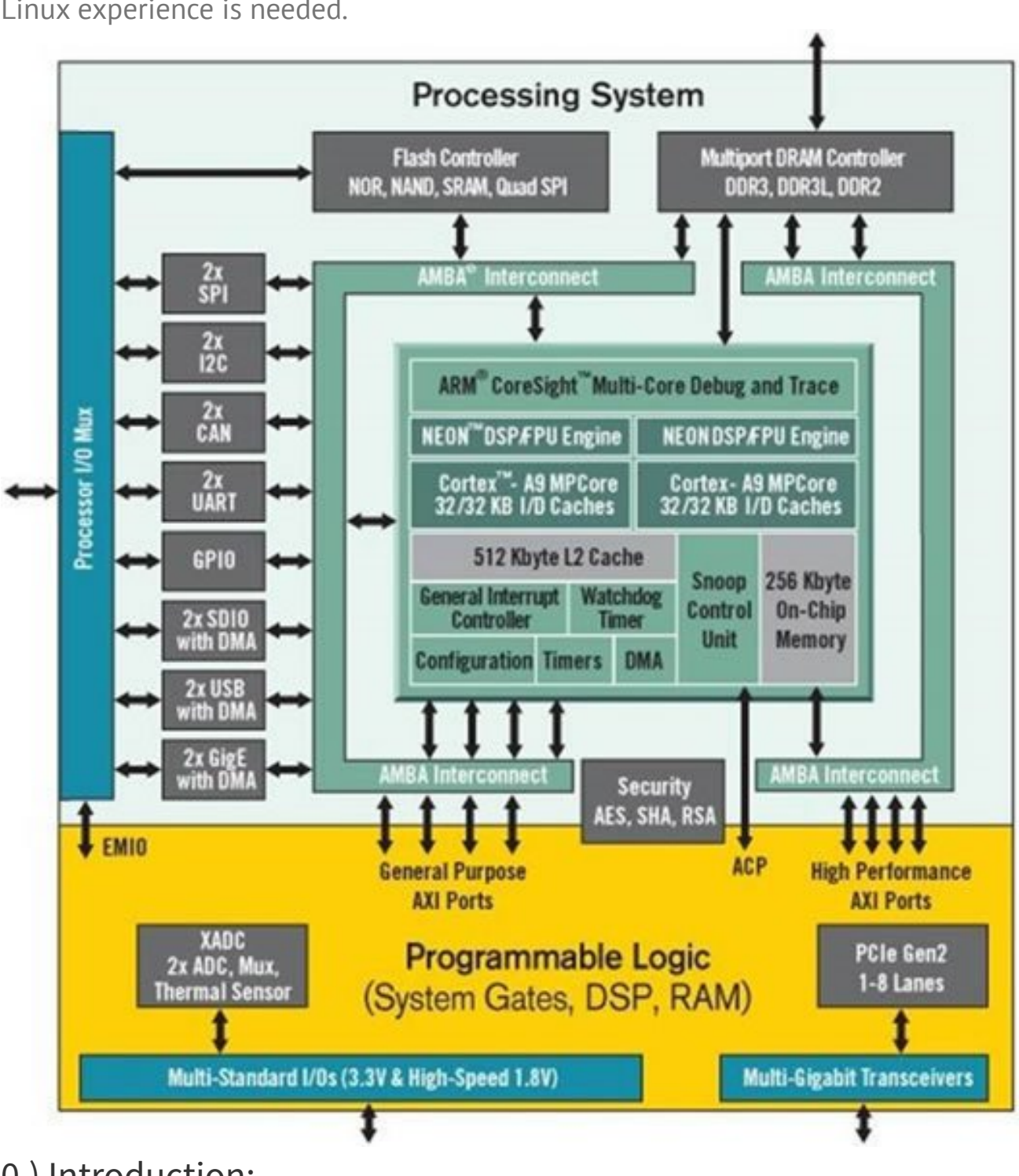
ZynqBerry Linux installation from scratch

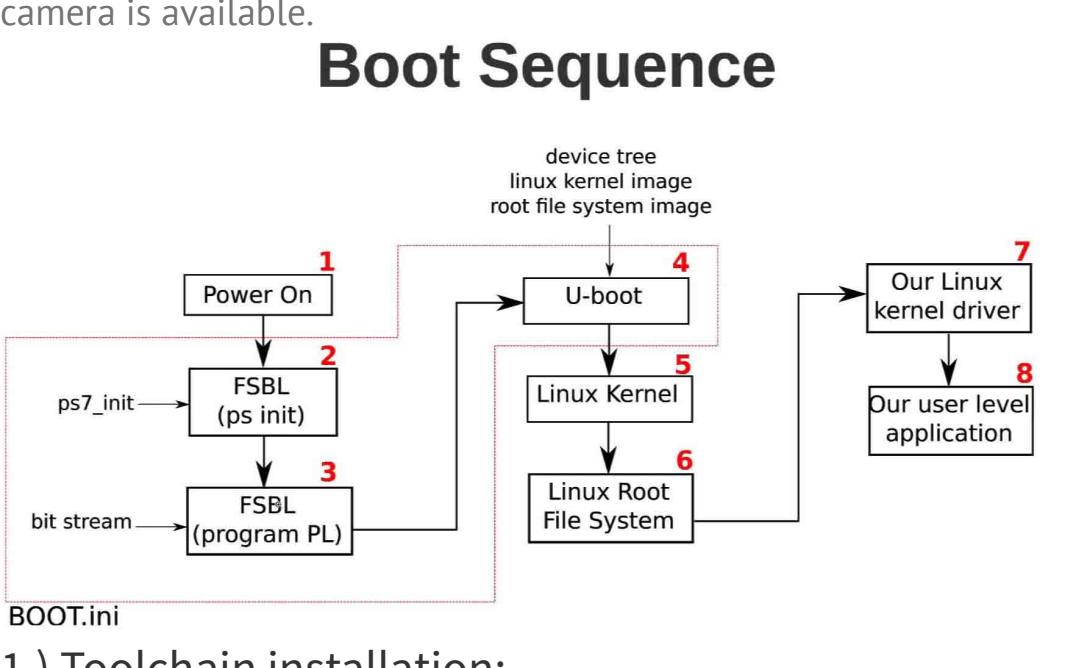


The ZynqBerry from Trenz-Electronic(see here) is a perfect playground for signal processing on the FPGA. The ZynqBerry has the same form factor as the Raspberry Pi and the same peripheral connectors. The ZYNQ-SoC combines a FPGA device with an ARM processor, a wonderful combination! The ARM is called processing system (**PS**), the FPGA programmable logic (**PL**). The toolchain for the PL development is Vivado by Xilinx. The build tools for the PS are Xilinx-SDK under Windows and PetaLinux under Linux which can be installed in a virtual machine. Please note that this tutorial, the scripts and the listed commands are only valid for the tools at the time this tutorial was written. All tools to build the System are free available. Linux experience is needed.



0.) Introduction:

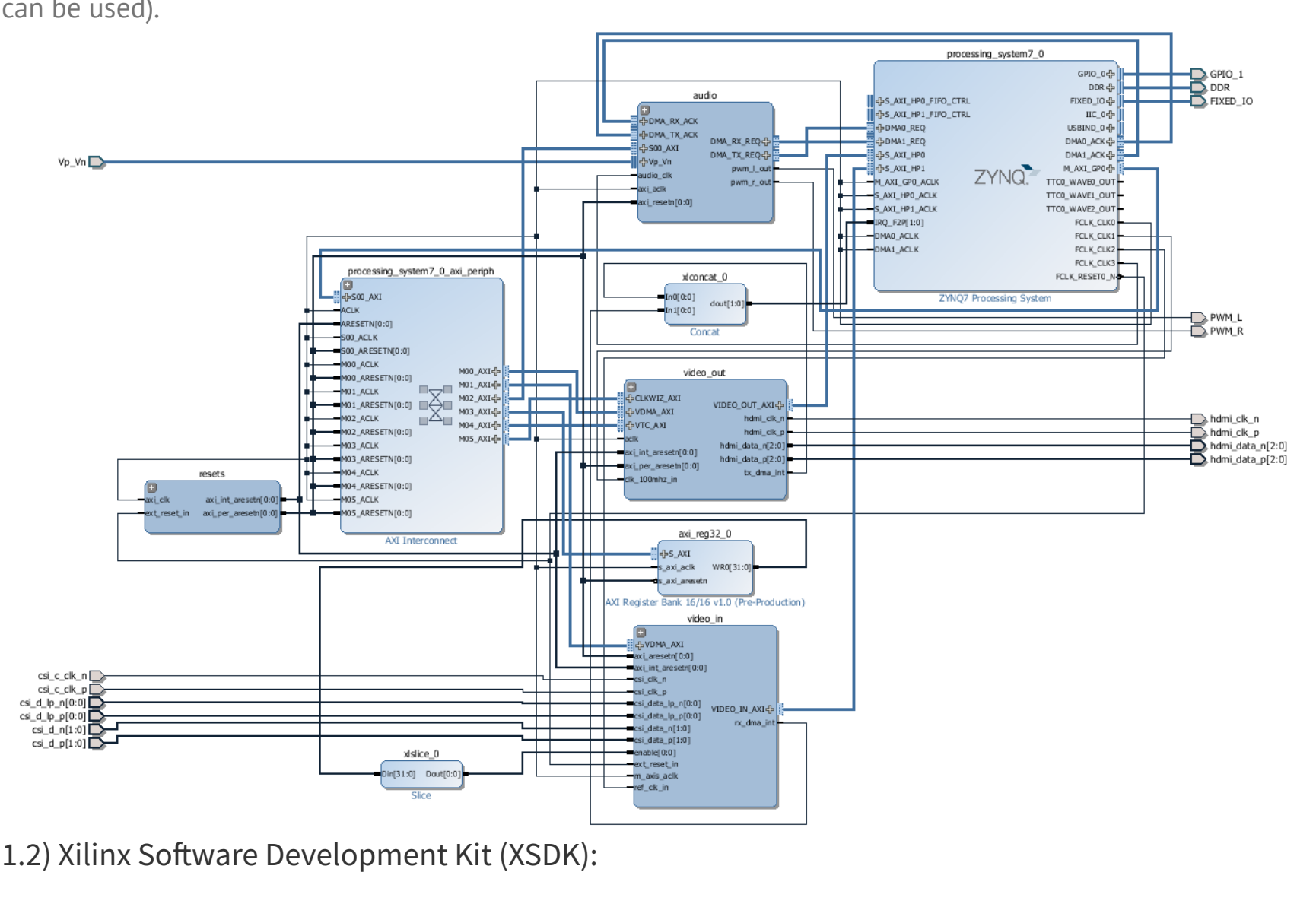
In order to build the system the following files are required.
1.) **FSBL.elf** –> the first stage boot loader.
2.) **zsys_wrapper.bit** –> FPGA bitstream file.
3.) **u-boot.elf** –> universal boot loader which loads the kernel from the SD-Card.
4.) **image.ub** –> Kernel Image.
5.) **BOOT.bin** –> contains the FSBL.elf, u-boot.elf and the zsys_wrapper.bit. Uploaded to the Flash.
6.) **debian.img** –> Linux Image.
It is not necessary to build all from scratch, a pre configured project for a device with video, sound an camera is available.



1.) Toolchain installation:

1.1) Vivado:

The Windows HDL tool Vivado can be downloaded here: Vivado Design Suite (the free ISE-WebPack Edition can be used).



1.2) Xilinx Software Development Kit (XSDK):

For embedded applications XSDK can be downloaded here: Xilinx Software Development Kit (only needed to build the FSBL).

1.3) PetaLinux:

PetaLinux is the Xilinx embedded linux distribution for the ZYNQ. It contains the Kernel, u-boot, rootfs, applications…

1.3.1) Install Ubuntu Linux in the VM:

Windows host, Linux guest. Install virtual box: Virtual Box with the pre configured Ubuntu 16.04 from www.osboxes.org –> OSBoxes (all cores, 4GB RAM). Password for user osboxes is "osboxes.org". Change the sudoers file (see here, add `osboxes "ALL=NOPASSWD: ALL"` to the `/etc/sudoers` file). Enable file exchange with "shared folders" in virtual box to Windows (enable "Auto-Mount" and "Make-Permanent"). In order to access the shared folders from Linux add the user `osboxes` to the group `vboxsf` in Ubuntu:
`> groupadd vboxsf`

1.3.2) Install PetaLinux:

Install the required Ubuntu packages (Xilinx UserGuide UG1144):
```
> sudo apt-get install tofrodos iproute2 gawk gcc git make net-tools
libncurses5-dev tftp zlib1g-dev libssl-dev flex bison libselinux1 lib32z1
lib32ncurses5
```
Download PetaLinux –> PetaLinux Download, install with:
`> ./petalinux-v2016.2-final-installer.run /opt/pkg`

2.) Download the Prebuild Project:

Download the pre configured project from Trenz-Elektronik –> **Reference Design** (download *build* package, not the *noprebuild*). Vivado has a limitation of 256 characters for file names –> don't unpack the project folder to deep into the file system. The `prebuild` folder contains the binaries, the `os` folder contains the PetaLinux project files. Included are also Windows scripts to open the preconfigured Vivado and XSDK projects.

3.) FPGA:

Set the correct paths in the **design_basic_settings.cmd** file and open the script **Vivado_create_project_guimode.cmd**. Did not change anything here and "Generate Bitstream", this can take up to 30 minutes! Export the Hardware with the menu **File->Export->Export Hardware** to the `os/petalinux/Subsystems/linux/hw-description` folder.

5.) XSDK, FSBL:

Open **sdk_create_prebuilt_project_guimode.cmd** script in the project root folder (XSDK with the pre configured hardware platform specification). FSBL can be build with:
–> Menu **File->New->Application Project**.
–> Project Name: FSBL, do not Change anything here, press Next
–> Select Zynq FSBL (te modified app…)
–> Open FSBL->src->fsbl_hooks.c and make sure `#define DIRECT_CAMERA_VIEW` is disabled (no direct HW copy of the camera stream to HDMI).
–> Save modifications, FSBl will be automatically rebuild.
–> Select the FSBL.elf file (FSBL->Binaries) with the mouse and copy the file to the `prebuild` folder.

6.) Petalinux Subsystem and the Kernel:

Build the PetaLinux system in Ubuntu:
`> mkdir ~/Development`
`> cd ~/Development`
`> cp -r /media/sf_PATH_TO_PROJECT_FOLDER.../os/petalinux/ .` –> from Windows shared folder….
`> cd petalinux`
`> source /opt/pkg/petalinux-v2016.2-final/settings.sh` –> initialize PetaLinux
`> export CROSS_COMPILE=arm-xilinx-linux-gnueabi-`
`> export ARCH=arm`
`> petalinux-config --get-hw-description=./hw-description/`
`> vi subsystems/linux/config`
Change the following lines:
SUBSYSTEM_MEMORY_PS7_DDR_0_BANKLESS_SIZE [=0x1F700000]
SUBSYSTEM_ROOTFS_SD [=y]
`> vi subsystems/linux/configs/device-tree/system-top.dts` –> only look, do not Change anything here
`> petalinux-config` boot from SD-card: **–>Image Packaging…–>Root filesystem…–>SD-Card**
`> petalinux-config -c kernel` –> not Change anything
`> petalinux-config -c rootfs` –> root file system is not build because direct boot from SD –> disable anything in **Libs**, **Apps** and **Modules**.
`> petalinux-build` –> can take some time
`> cp -t /media/sf_PATH_TO_PROJECT_FOLDER../prebuild images/linux/zsys_wrapper.bit images/linux/u-boot.elf images/linux/image.ub` –> copy build results to Windows

7.) The Debian Linux Image:

The script `mkdebian.sh` builds a Debian (Jessie, ARM, armhf) distribution image. Two Ubuntu packages must be installed:
`> apt-get install debootstrap qemu`
Download this mkdebian.sh script and copy into `~/Development/petalinux` folder in Ubuntu.
`> sudo ./mkdebian.sh`
The script generates the `te0726-debian.img` linux image file. Copy the image to Windows.
`> cp te0726-debian.img /media/sf_PATH_TO_PROJECT_FOLDER../prebuild/`

8.) Install everything on the ZynqBerry:

8.1) Write the Linux Image to the SD-Card:

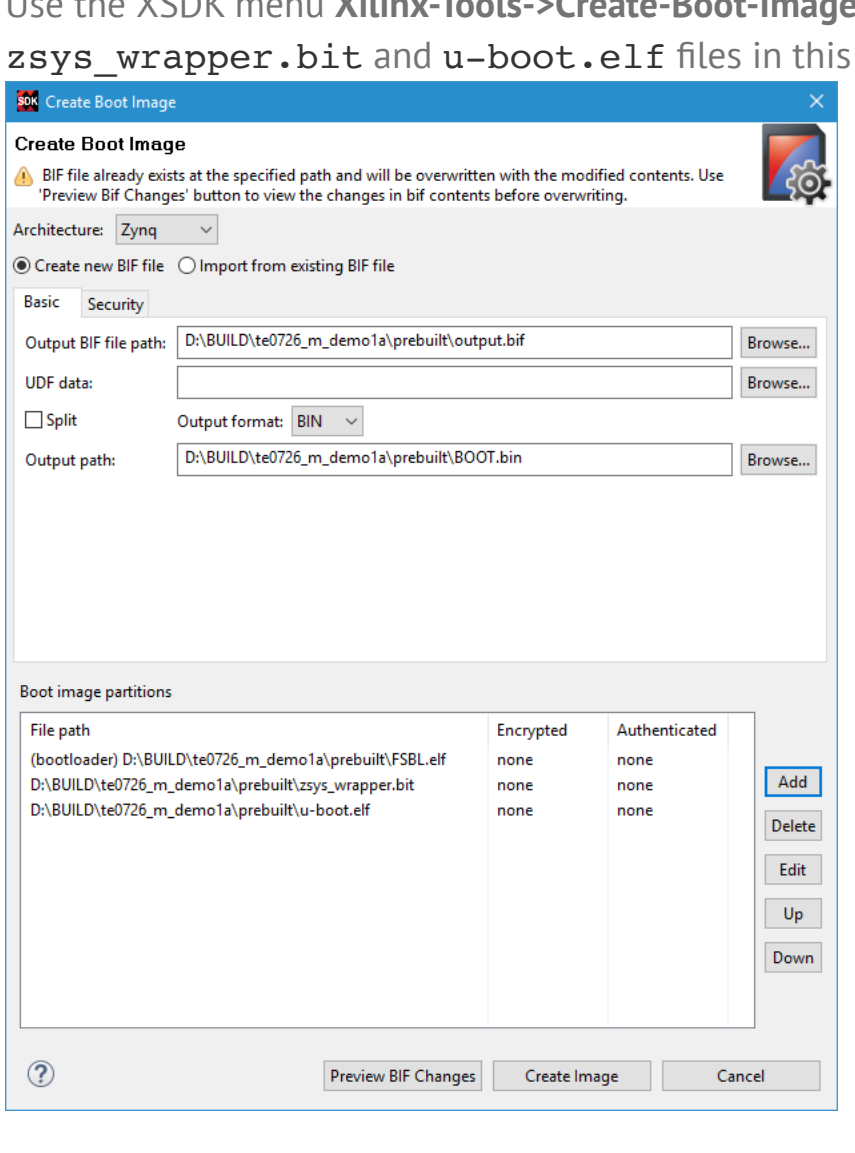
Use Win32DiskImageMaker to write the `te0726.img` to the SD-Card. After the image has been written a partition is visible in Windows.

8.2) Copy the Kernel to the SD-Card:

Copy the `image.ub` from the prebuild folder to the first partition in Windows.
Copy the `/misc/img/prebuild/u-boot.rgba` file also to the first partition in Windows.

8.3) Create the BOOT.bin image:

Use the XSDK menu **Xilinx-Tools->Create-Boot-Image** to create a boot image. Add the `FSBL.el`, the `zsys_wrapper.bit` and `u-boot.elf` files in this order. Save the BIN file to `prebuild` folder.



8.4) Flash the BOOT.bin file:

Connect the ZynqBerry with micro USB cable to the PC. Use the XSDK menu **Xilinx-Tools->Program-Flash** and program the `BOOT.bin` file to the ZynqBerry (flash type "qspi_single").

9.) Boot Linux:

Connect the ZynqBerry with micro USB cable to the PC. Connect mouse and keyboard and HDMI monitor. Use putty and listen to the serial console (COM4). Boot up the device. Login with user:`root` and the password:`root`. Start the X-Server with `startx`. This could be automatically done with an entry in the `/etc/rc.local`.
`> echo 'startx &' >> /etc/rc.local`
Open terminal on the ZynqBerry and update the system:
`> apt-get update`
`> apt-get upgrade`

10.) (Optional) Hello Qt World:

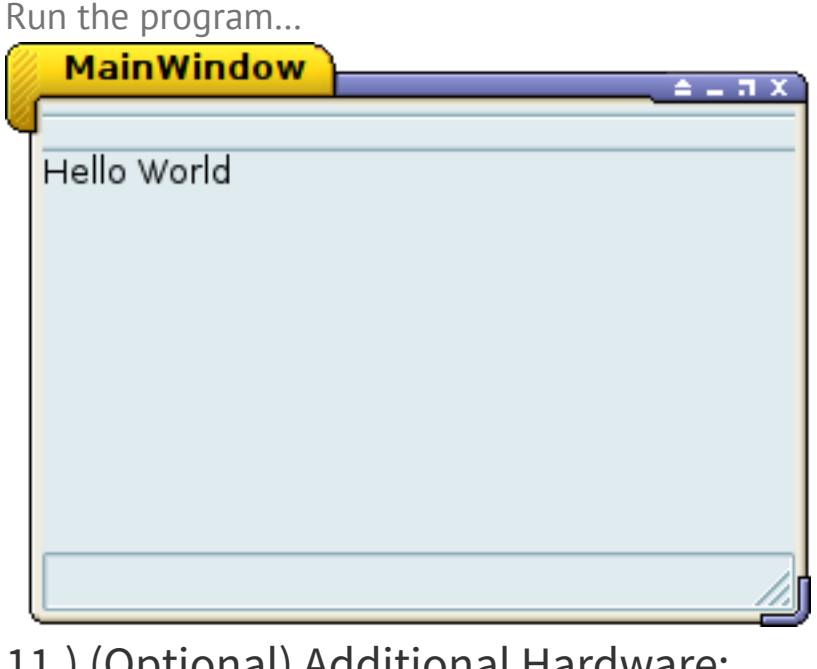
Install Qt-Creator:
`> apt-get install qtcreator`
Open qt-creator and create a new "Qt Widgets Application" Project, select QMainWindow as base class. Make sure your `mainwindow.cpp` look like:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "qlabel.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent),
  ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QLabel* l = new QLabel("Hello World", this->centralWidget());
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

Run the program…



11.) (Optional) Additional Hardware:

10.1) Cable Network:

The LAN9514 ethernet controller is supported in Debian.

11.1) USB Sticks:

`> mkdir /media/UsbStick`
`> mount /dev/sda1 /media/UsbStick`

11.2) The Framebuffer Camera:

The ZynqBerry has the Raspberry Pi CSI = Camera Serial Interface. For the **Raspberry Pi camera in Version 1.3** the I2C initialization code is available. It can be found in the PetaLinux folder under `../petalinux/components/apps/rpi-camera`. Copy the files to the ZynqBerry and compile with:
`> gcc rpi-camera.c -o rpi-camera`
Do the same for the `reg` program in `../petalinux/components/apps/reg`
The camera can now be initialized with:
`> rpi-camera /dev/i2c-5`
`> reg 0x43C10040 1`
After the initialization the camera is available as framebuffer device `/dev/fb1`. Test the camera by copying the camera image directly to the Screen:
`> cat /dev/fb1 > /dev/fb0`
Record a camera stream with:
`> ffmpeg -y fbdev -i /dev/fb1 -framerate 24 -s 320x240 test.mpg`
Look here if you want to use the framebuffer camera image in Qt –> Qt Framebuffer Camera

11.3) The Sound Driver:

The kernel module for the alsa sound driver must be compiled with PetaLinux in Ubuntu:
`> petalinux-config` –> set **Image-Packaging…->Root-filesystem->INITRAMFS**
`> petalinux-config -c rootfs` –> make sure **Modules->te-audio-codec** is selected
The compiled kernel module `./build/Linux/rootfs/modules/te-audio-codec/te-audio-codec.ko` can now be copied to the ZYNQ at `/lib/modules/4.0.0-xilinx/extra/te-audio-codec.ko`. Add the following line to the `/etc/tc.local`:
`> insmod /lib/modules/4.0.0-xilinx/extra/te-audio-codec.ko`
`> echo 'insmod /lib/modules/4.0.0-xilinx/extra/te-audio-codec.ko' >> /etc/rc.local`
Reboot and test the alsa sound driver while playing an audio file with VLC.

11.4) Change the Display Resolution:

The resolution is fixed to 1280×720@60Hz, and can be changed with:
a.) In the Vivado block design open the video-out IP and open the video timing generator. Change the settings for the resolution and note the frame-sizes. The values in the clocking wizard IP must also be set. The base clock frequency is `horiz_frame_size x vert_frame_size x FPS`. CLK2 is the double base frequency, and CLK3 the tenfold base frequency.
b.) Open `fsbl_hooks.c` in XSDK and Change the values for the VDMA.
c.) Change the resolution in the PetaLinux device-tree with `vi subsystems/linux/configs/device-tree/system-top.dts`.
d.) Rebuild everything.

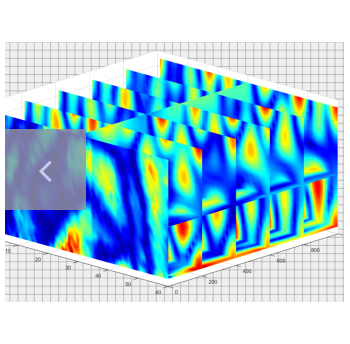
12.) Troubleshooting:

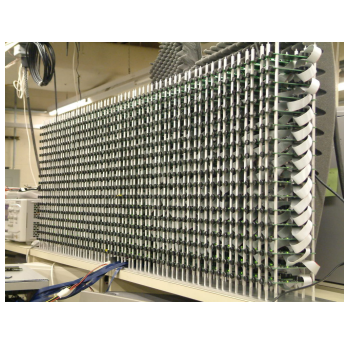
12.1) Reset ZYNQ via the XMD console:

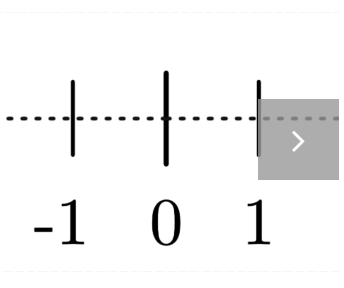
In case the ZYNQ does not answer while programming or the Flash is damaged the ZYNQ can be reseted with the XMD console: XSDK menu **Xilinx-Tools->XMD-Console**:
```
XMD% connect arm hw
XMD% rst -debug sys
XMD% targets
XMD% disconect 64
```

By Christoph Lauer | November 20th, 2016 | Categories: Generic | Tags: C/C++, Embedded, FPGA, Hardware, Linux, Network, QT, SHELL, Toolkit, Video, Windows, ZYNQ | 0 Comments

Related Posts







Leave A Comment

Comment...

Name (required) Email (required) Website

POST COMMENT

Popular Recent

Realtime-Audio-DSP Tutorial with the ARM STM32F4-Discovery Board
May 11th, 2015

A CAN Driver Library for the ARM Cortex-M4 based STM32F4
May 24th, 2015

A Realtime Audio Demonstration of an Adaptive Filter with the ARM Cortex-M4 based STM32F4-Discovery Board
May 30th, 2015

Tag Cloud

3D Acoustic Algorithm Artificial Intelligence Artwork Audio beamforming C/C++ CAN Dashboard Widget Design Documentation Download DSP Library Embedded FFT FPGA Hardware Java KDE Linux Mac OS X Microphone Array Music Network OpenGL Paper Photo Physical Simulation Programming QT SHELL Signal Processing Software Sonogram Spectrum Speech Recognition Tool Toolkit University Video Wavelet Wavelets Wigner-Ville-Distribution Windows