Installing Linux on a DE10-Nano with support for OpenCL, HDMI, and Webcams

Mackenzie Hauck

Table of Contents

1. Overview

This document will go over all of the steps needed to get embedded Linux with OpenCL drivers, HDMI video out, and webcam drivers running on the Terasic DE10-Nano board.

NOTE	If you do not need a Linux image with both HDMI output, AND OpenCL to be running, you can use an official SD card image provided by Terasic or Intel.

1.1. Prerequisites

Hardware:

- Terasic DE10 Cyclone V development board
- Mini-USB to USB A cable
- 4GB+ MicroSD card
- Ethernet cable
- SD card reader
- A PC that can run Quartus (preferably running Linux, see apendix for running on Windows)
- Optional: webcam, USB OTG to USB A adapter

This document assumes you are at least somewhat familiar with the following topics:

- Linux (navigating via the commandline, and commandline tools)
- Quartus, and Qsys (now known as Platform Designer in 17.1 and up)
- Basics of VHDL or Verilog

No prior knowledge is required in:

- OpenCL
- Compiling the Linux kernel
- Linux drivers

1.2. Hardware

The DE10-Nano includes a single CycloneV SoC (system on chip) composed of a dual core ARM Cortex A9 CPU and a FPGA with 110k logic elements, a MicroSD card slot, USB host via an OTG (on-the-go) port, and ethernet. Intel supports running Linux on the SoC which then requires drivers to communicate with ethernet, USB webcams, FPGA fabric, and OpenCL custom components that *run in* the FPGA fabric. Most of these drivers are available in the Linux kernel source, so including them is as simple as enabling them in the kernel configuration. The one not supported directly by the kernel is the OpenCL

driver. Intel provides the source code of this driver which can be compiled to be used with our kernel.

Our operating system (Linux) is stored on the MicroSD card, and the setup is very similar to that of a Raspberry Pi. The architecture of the A9 cores is `armv7` (the same as Raspbery Pi 2 and up), and includes floating point hardware, so looking at tutorials for those Pi's may be useful to get cross compilers or other software installed. The prefix for this architecture is `arm-linux-gnueabihf` where the `hf` stands for "hard float", i.e., dedicated floating point hardware. The processor also supports the NEON SIMD instructions.

1.3. Embedded Software

In order to more easily take advantage of the FPGA logic from Linux, Intel provides an OpenCL offline compiler allowing us to write "high-level" OpenCL code that then gets compiled down into a bitstream that can be loaded onto the FPGA. The Linux kernel communicates with the custom logic on the FPGA by way of an Intel provided kernel driver and the matching Qsys component. With this driver loaded, we can run what's known as an "OpenCL host program" which uses the standard OpenCL interface to talk to the FPGA. This host program is run in userland in Linux. No FPGA specific code is required in the host program, and it can be coded in many languages with OpenCL support like C++, C, Rust, Python, etc. Theoretically the host program could be written in a way to be backend agnostic, where the OpenCL device could be a GPU, FPGA, etc. However, in practice, the workflows are slightly different:

- The FPGA requires pre compiled OpenCL code compared to GPUs using the OpenCL source code
- FPGA code may contain Intel specific OpenCL extensions which will only run on Intel FPGAs

1.4. SD Card layout

Our operating system is stored on the SD card, split into several partitions. The two partitions we are interested in are the root file system partition (EXT3), and the "boot" partition that contains the bootloader, u-boot script, and the compressed kernel image file. The boot partition is FAT32, so we can access its contents from Windows, or another operating system to modify the files that are used during boot.

Due to the difficulty of of compiling everything from scratch and creating an SD card image, we will instead be modifying an existing SD card image from the user thinkoco on github, then updating the kernel with webcam drivers enabled, and installing some software in Linux.

Much of the content in this guide has been adapted from thinkoco's guide which you can view here.

NOTE

The site RocketBoards.org (run by Intel) has what is known as the "Golden System Reference Design (GSRD)" which is a good starting point for working with Intel FPGAs. https://rocketboards.org/foswiki/Documentation/GSRD

Here is the documentation on the SD card layout if you want to learn more.

2. Preparing our donor SD card image

We will be using the SD card image provied by the user thinkoco on Github which uses kernel version 3.18 with a root file system containing Ubuntu 16.04.

NOTE	The file system also already includes the Intel RTE (a subset of the FPGA SDK for OpenCL). The RTE contains diagnostic tools, a compiled kernel driver to communicate with the OpenCL FPGA logic, and the OpenCL dynamic libraries (`.so` files) that the host program is linked against.

2.1. Download the image

The .img file can be downloaded from [here]

2.2. Load the image onto a MicroSD card

On Windows use Win32DiskImager, on Mac or Linux use `dd`.

You can follow Intel's guide on writing the image with these tools here.

Connect the DE10-Nano to your development PC with the Mini-USB cable and open a serial connection with baud rate 115200.

NOTE	During boot this serial port will display the Linux boot sequence, and after booting, display a login prompt. The default username is `root` and there is no password.

Insert the SD card into the DE10-Nano, and apply power. If you get a login prompt, you are ready for the next step.

3. Compiling our custom Linux kernel

The Linux kernel provided by thinkoco does not have USB webcams enabled, so we will build our own kernel with the required drivers. In the FAT32 boot partition, the kernel is the `zImage` file.

NOTE	A `zImage` is a compressed kernel image, signified by the 'z'. An uncompressed kernel image is named `uImage`.

Our built kernel (`zImage`) will also disable strict checking for loadable modules. This means that even if the version magic of a module does not match exactly, it will still be loaded by the kernel.

NOTE	Our OpenCL Linux kernel driver is loaded as module into the kernel, hence the need for loadable module support.

WARNING	Loading kernel modules that weren't compiled against the *exact* kernel version (3.18) may lead to kernel panics with this setting disabled. Be sure to re-enable for non-development builds.

The following steps were done on a server running Ubuntu 16.04.

3.1. Installing dependencies

```
$ sudo apt update
$ sudo apt install u-boot-tools gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf
libncurses5-dev make lsb uml-utilities git
```

3.2. Downloading kernel source

Apparently Intel does not like keeping source code public that is vulnerable to some exploits, so they have removed their 3.18 branch from github possibly due to the Meltdown / Spectre exploits. Instead, we will download the kernel source from thinkoco's github repository.

We can either clone the whole repository (~1.4GB), or just a single branch as shown below. If you clone the entire repo, be sure to checkout the 3.18 branch.

```
$ cd ~
$ git clone --single-branch -b socfpga-opencl_3.18 https://github.com/thinkoco/linux-
socfpga.git
$ cd linux-socfpga
```

3.3. Building with a modified config

We will use a config already modified to include support for webcams.

NOTE	The `.config` file holds the configuration for building the kernel and should only be modified by certain tools. See the appendix for how to modify it with a `ncurses` frontend.

```
$ cp 3.18_usbcam_config .config
# setup compiler options
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ export LOADADDR=0x8000

# setting LOCALVERSION blank means that kernel modules will
# not have to match their version exactly to the kernel
$ export LOCALVERSION=

# build with 24 threads. replace 24 with the number of threads on your machine
$ make -j24 zImage

# copy the compiled kernel
$ cp arch/arm/boot/zImage ~/output/
```

Now that we have our modified `zImage`, we can insert the MicroSD card into our computer and overwrite the original `zImage` in the FAT32 partition. Once again, boot the DE10 with the USB serial connected and verify you get to the login prompt.

4. Installing and running OpenCL diagnostic tools

5. Quartus setup

6. Compiling and running our own host program, and OpenCL code

6.1. Writing the code

6.2. Compiling the OpenCL code

6.3. Modifying OpenCL libraries

7. Appendix

Other information that may be useful.

7.1. The device source tree

7.2. Modifying Linux kernel config

7.3. Troubleshooting