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.

1.1. Prerequisites

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

- Linux (navigating via the commandline, and commandline tools)
- gcc / Makefiles
- 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
- Rust

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 source files for a GPU
- 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, but it is split into several partitions. 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.

I will give a very brief, and incomplete, overview as we are not too concerned with the specifics of how the SD card is setup.

2. Preparing our donor SD card image

3. Compiling our custom Linux kernel

4. Installing and running 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