Lab 4.0 – Embedded Linux Systems

Lab 3 – Hybrid Systems (recap)

The goal of lab 3 was to explore how hybrid systems involving both *hard* and *soft* components interact together. To this end, you re-implemented the FPGA-only thermal camera acquisition system you had previously developed in lab 2, by replacing the *embedded* Nios II processor controlling the system by the *general-purpose* ARM processor.

To simplify the transition and to keep the code changes minimal, you used the ARM processor to run *bare-metal* code, similarly to how you were using the Nios II processor.

Lab 4.0 – Embedded Linux Systems

Bare-metal limitations

Running *bare-metal* code on the ARM processor is essentially similar to having a high-frequency Nios II processor, but there are many downsides:

1. The ARM processor available on Cyclone V SoC devices is a dual-core CPU. However, the preloader does not wake CPU1 up from reset, so your code is running on only one of the cores. You cannot use the second core unless *you write the code* needed for waking it up from reset.
2. Interfacing with the sdcard is impossible unless *you write the code* needed for interacting with the multiple filesystems currently on the drive.
3. Interfacing with the Ethernet port is impossible unless *you write the code* needed to implement a complete TCP/IP stack.
4. …

As you can see, the phrase "*unless you write the code needed for <xyz>*" comes up often in the previous list. Actually, the situation is much worse than in the Nios II systems you have built until now. Indeed, whenever you create a software project with the Nios II software toolchain, you see that 2 projects are always created:

- Application project
- Board Support Package (BSP) project

The BSP project contains all the information relative to the system on which the Nios II processor is instantiated, but it also contains code needed for the processor to interact with its environment (standard I/O, file I/O, networking …). This "bridge" code is called the *Hardware Abstraction Layer (HAL)*. All systems

implement a HAL in some form in order to ease user programming related to interfacing with hardware devices. Whenever you compile your Nios II application code, the software toolchain also automatically compiles and links its associated HAL into a single binary.

However, when writing bare-metal programs for the HPS, you don't have access to a HAL unless you explicitly include it yourself[1]. Even if we include this HAL, you still haven't solved the numerous problems listed previously, namely you still have to write all the code needed to access system interfaces, *even standard ones that exist on all machines* (multi-core CPUs, filesystems, networking stack …).

This situation arises all the time, and hence has an expression coined specifically to describe it. We usually call this *re-inventing the wheel*.

In this lab, we will explore one way of getting around this issue: by installing an *operating system (OS)*.

Getting Started

The operating system we will install in this lab is *Linux*. In order to gain access to more than what the Linux kernel alone provides, we will further customize our system by installing the *Ubuntu Core root filesystem*.

In lab 3, you saw that getting the HPS up and running is a much more involved process compared to the Nios II processor, even for a simple bare-metal application. As you can expect, the steps needed to get a complete Linux system up and running are even longer…

As for the last lab, we would normally tell you to **RTFM**, but given that no single document exists that explains how to build such a Linux system from scratch, we have written a step-by-step tutorial that explains how this is done for the Cyclone V SoC-based devices. You can follow the tutorial by reading the SoC-FPGA Design Guide.

The tutorial was written for the DE1-SoC board, which is a much larger Cyclone V SoC-based device compared to the DE0-Nano-SoC that we are using for this course. However, the steps needed to get a Linux system running are very similar for both devices, and you can apply what you'll learn about the DE1-SoC to the DE0-Nano-SoC with only minor changes.

The tutorial is quite long, but you don't need to read all of it, as it includes material that we cover in future labs. The chapters you should read are the following:

- Chapter 7: Cyclone V Overview
  - 7.2: Features of the HPS
  - 7.4: HPS-FPGA Interfaces
  - 7.5: HPS Address Map
  - 7.6: HPS Booting and FPGA Configuration
- Chapter 8: Using the Cyclone V – General Information
- Chapter 9: Using the Cyclone V – Hardware

[1] Altera provides a minimal HAL under the name HWLIB. It is mentioned in the SoC-FPGA Design Guide for reference.

    - 9.3: System Design with Qsys – HPS
    - 9.4: Generating the Qsys System
    - 9.5: Instantiating the Qsys System
    - 9.6: HPS DDR3 Pin Assignments
    - 9.7: Wiring the DE1-SoC
    - 9.8: Programming the FPGA
- Chapter 11: Using the Cyclone V – HPS – ARM – General
    - 11.2: Generating a Header File for HPS Peripherals
    - 11.3: HPS Programming Theory
- Chapter 13: Using the Cyclone V – HPS – ARM – Linux

As in lab 3, if you understand how the system is built and how the different components interact together, you will see that installing Linux is not that complicated. All the information you need can be found in the tutorial, but whenever in doubt, don't forget that we have a Q&A Forum for this course where you can ask questions!

Your Task