Project Of Systems-On-Chip (CS-309)

Lab 3: Hybrid & Embedded Linux Systems

MAROUF Imad Eddine
ESER Can

Supervisor : Pr. BEUCHAT René
École Polytechnique Fédéral de Lausanne, Switzerland

***Abstract*—In this report, we present the details and steps used to install a linux system on Cycle V FPGA with customized softwares and Ubuntu Core root file-system. We also discuss, the main parts of the architecture of Cyclone V in order to set up an operating system. During the second part of the laboratory we discuss the differences and advantages of using bare-metal or embedded Linux system in prototype. This report is in the context of "Projet de Systems-on-Chip CS-309" course supervised by Pr. BEUCHAT René.**

INTRODUCTION

The objective of this third laboratory is to have an broad overview about the architecture of FPGA and the main components which are responsible for setting up an operating system.

Cyclone V FPGA provides a hard processor system (HPS) portion and an FPGA and various peripherals portion as illustrated in figure 1.

A key advantage that we learnt in this laboratory is the importance of using Linux's built-in drivers that support a vast array of peripherals, including many of them found on the DE1-SoC board. Considering the USB and Ethernet ports of the DE1-SoC board, writing a driver code for these devices is no easy feat, and would significantly increase the development time of an application that requires them. Instead, a hardware designer can use Linux operating system and its driver support for these devices, allowing them to use the high-level abstraction layers provided by the drivers of the OS to use the devices with minimal effort. Which gives power capabilities and freedom to design robust and cool embedded systems.

I. CYCLONE V ARCHITECTURE

A general block diagram of the DE1-SoC development board is provided in Fig 1. The DE1-SoC contains a Cyclone V device which comprises of two distinct components - an FPGA and Hard Processor System (HPS). The HPS is a hard logic microprocessor unit (MPU) consisting of a dual-core ARM Cortex-A9 processor, on-chip memories, SDRAM, L3 interconnect, and support and interface peripherals. The HPS will be used to execute the software portion of your SoC design.

The Cyclone V's FPGA component consists of FPGA fabric, standard FPGA components (LUTs, CLBs, PLL etc), shared memory controllers, and general peripherals found on a standard FPGA dev board. The FPGA is used to prototype hardware for your SoC design, receiving and sending data to and from the HPS using AXI buses, bridges, and Avalon master-slave devices.

HPS consists of a dual-core Arm Cortex-A9 MPCore processor, a rich set of peripherals, and a shared multiport SDRAM memory controller, as shown in the following figure1. Block diagram shows the three switches inside HPS but the most important one is L3 main switch as is controls the other switches and also the bridges. There are three

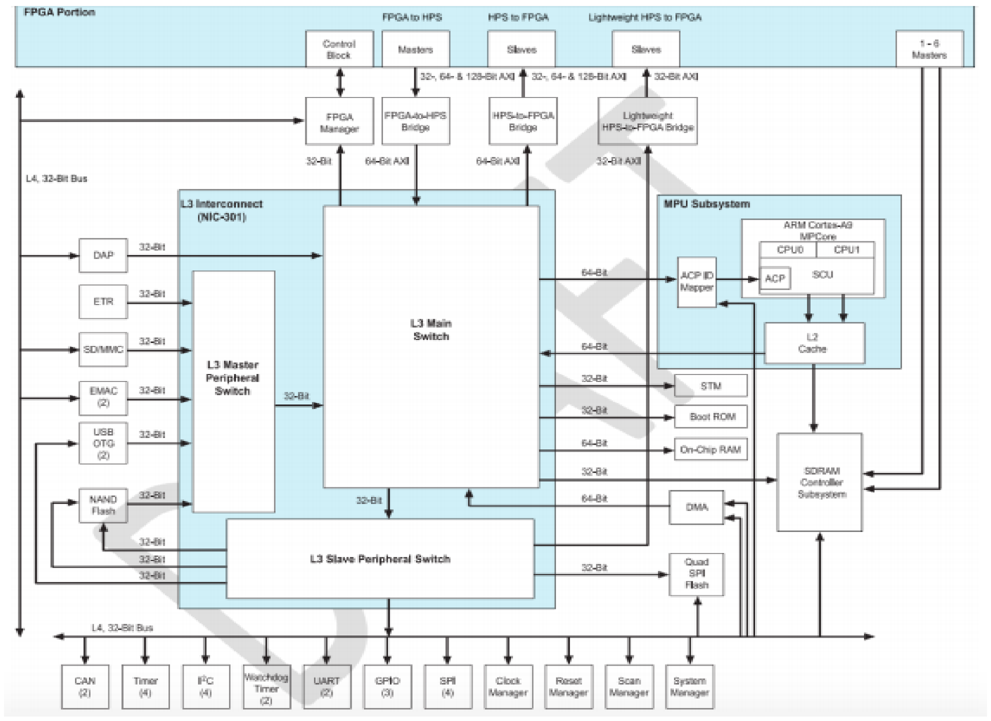


Figure 1: HPS Interconnections inside Cyclone V

bridges for different purposes. FPGA-HPS Bridge is used to control HPS from FPGA and vice versa.

There are two types of HPS-FPGA bridges: One is HPS-FPGA and other one is HPS-FPGA LW (Light Weight). LW Bridge is used when control of FPGA peripherals is needed through FPGA while other one is used when FPGA is configured from HPS software. HPS-FPGA Bridge can be of 128 bits depending upon the specific task. Now to access these bridges, address mapping of these are very important to know. Base addresses are given below: - HPS-FPGA LW (0xFF20_0000) - HPS-FPGA (0xC000_000)

Programs running on the ARM processor of the Cyclone V SoC can communicate with FPGA-side components through the either the HPS-to-FPGA (hps2fpga) or the Lightweight HPS-to-FPGA (lwhps2fpga) bridge built into the chip. These bridges are mapped to regions in the ARM memory space, meaning that communicating with the FPGA effectively boils down to accessing these memory regions. When an FPGA-side component (such as an IP core) is connected to one of these bridges, the component's memory-mapped register interface will be available to the ARM within the bridge's memory region. Qsys is used to connect the component to a bridge, and to set its register interface's address offset within the bridge's memory span.

If we were developing a baremetal ARM program (a program that does not run on top of an operating system), accessing addresses within the bridge memory region would be as simple as de-referencing a pointer to the physical address of interest. For a program running on Linux, things are slightly complicated as Linux programs have access to a virtual memory space rather than the physical memory space.

II. Installation Steps

A. Overview about the bootflow

Booting software on the HPS is a multi-stage process. Each stage is responsible for loading the next stage. The first software stage is the boot-ROM. HPS starts when the processor is released from reset (powers up) and executes code in the internal bootROM at the reset exception adress. Typically, the main responsibilities of the BootROM are: Detect the selected boot source, perform minimal HPS initialization and load the next boot stage (typically the Preloader) from Flash to OCRAM and jump to it.

The preloader is a piece of software that gets called from the Boot ROM with the purpose of setting up the system to a point that the actual bootloader can be run, sometimes is referred to it as ”source”. It performs many tasks like initialize the SDRAM interface, configure the HPS I/O and loading the next boot software into the SDRAM and pass control to it.

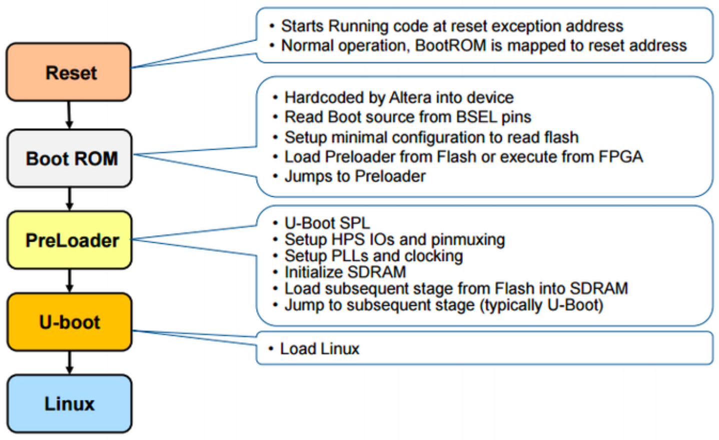

Reset
Boot ROM
PreLoader
U-boot
Linux
• Starts Running code at reset exception address
• Normal operation, BootROM is mapped to reset address
• Hardcoded by Altera into device
• Read Boot source from BSEL pins
• Setup minimal configuration to read flash
• Load Preloader from Flash or execute from FPGA
• Jumps to Preloader
• U-Boot SPL
• Setup HPS IOs and pinmuxing
• Setup PLLs and clocking
• Initialize SDRAM
• Load subsequent stage from Flash into SDRAM
• Jump to subsequent stage (typically U-Boot)
• Load Linux

Figure 2: Cyclone V Bootflow

The purpose of U-Boot is to get the system up to the point the Linux kernel can be started up (in the same way the purpose of the preloader was to set up the system to run U-Boot).

The FPGA fabric and HPS in the SoC are powered independently. You can reduce the clock frequencies or gate the clocks to reduce dynamic power, or shut down the entire FPGA fabric to reduce total system power.

• You can configure the FPGA fabric and boot the HPS independently, in any order, providing you with more design flexibility

• You can boot the HPS independently. After the HPS is running, the HPS can fully or partially reconfigure the FPGA fabric at any time under software control. The HPS can also configure other FPGAs on the board through the FPGA configuration controller.

• You can power up both the HPS and the FPGA fabric together, configure the FPGA fabric first, and then boot the HPS from memory accessible to the FPGA fabric.

In conclusion, the booting process for Cyclone V SoC when OS is present can be summarized as the following. The preloader is loaded from the bootROM program (saved during chip manufacturing) inside the 64k On-Chip RAM. It configures external RAM memory. Afterward, it loads u-boot in RAM and jumps to it. The u-boot loads some basic drivers needed to launch OS. It executes some commands to initialize the system, reads ".dtb" file and loads some basic drivers to work with some devices. The u-boot can configure the FPGA. In the end, it loads the kernel image in RAM and passes execution to it.

B. Main Steps

As requested in the lab manual we can divide the roadmap to install the linux operating system into seven main steps explained below:

- *1- Compile the preloader: The preloader is used to sets up the HPS. We used the preloader generator on Quartus to generate and compile the preolader and copie its binary to the sdcard target directory.*

- *2- Compile the bootloader: The bootloader allows to load the linux kernel. We downloaded the official U-Boot sources online and cross-compiled it.*

- *3- Compile mainline linux kernel: We downloaded the latest version of Linux kernel and cross-compiled it.*

- *4- Compile the device tree of DE0-Board: In this step, we need to compile the device tree blob for the DEO-Nano-SoC and copy the linux zImage binary and the linux device tree blob to the sdcard target directory.*

- *5- Set up a root file system: After having a fully-working linux machine, then we installed a linux distribution in order to have more tools and functionality. We needed to install Ubuntu Core on our DE0-Nano-SoC which is is the minimal root filesystem (rootfs) needed to run Ubuntu. It consists of a very basic command-line version of the distribution, and can be customized to eventually ressemble the desktop version of Ubuntu. Most importantly, it comes with a package manager. The final step was to configure the rootfs directly on the DE0-Nano-SoC when it boots the operating system for the first time. For this, we used a provided shell script "config_system.sh" on our host machine and place it in the extracted rootfs and configure the rootfs to launch our script when it boots for the first time.*

- *6- Partition and format SD card: After having all the files needed to create the final sdcard, we mounted the sdcard partitions and wrote the preloader to the "a2"* *partition, FPGA .rbf file, U-Boot .img file, U-Boot .scr file, linux zImage file, and linux .dtb file to the FAT32 partition and the customized Ubuntu Core root filesystem to the EXT3 partition.*

- *7- Boot the Linux system: Finally, to boot the Linux system, we wired up the DE0-Nano-SoC by connecting the Power cable, USB-Blaster cable, Ethernet cable and UART cable, then, we pluged in the microSD card. At this step, we should be sure to set the MSEL switch on the top side of the DE0-Nano-SoC to "00000". By using minicom, we should disable the hardware flow control, and we can Power-on the DE0-Nano-SoC. Finally, once we are logged in, we should call the post-installation configuration script to install the required tools from the package manager. We need to be sure to be connected to the internet and share properly the host pc's internet connection with the DE0-Nano-SoC trough the Ethernet cable.*

The tutorial [3] provided, explains in much more details each step above in order to complete the setup.

III. Bare-Metal Application

Embedded systems may be running a fully operating system, or a minimal system that just provides some real-time functions and scheduling. In cases when there's no operating system at all, the system is said to be bare-metal, and consequently bare metal programming is programming directly for a device that lacks an operating system. Bare metal programming can be both highly challenging and very involving with regards to programming it. Because we will deal directly with hardware no available abstraction layers are in between the programmer and the device.

The Bare Metal application can be invoked in one of many ways. In the following three scenarios 3. It is invoked after the Preloader boot stage has completed the system hardware initialization and verified the Bare-Metal image or has been built as a Boot Module.

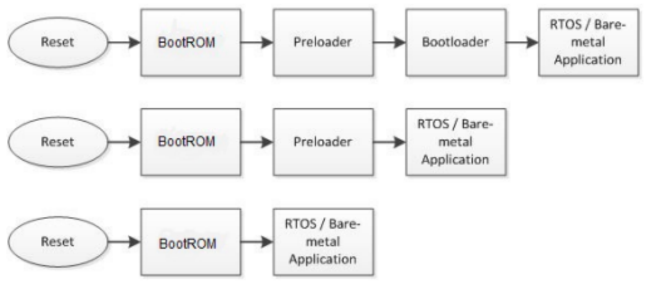
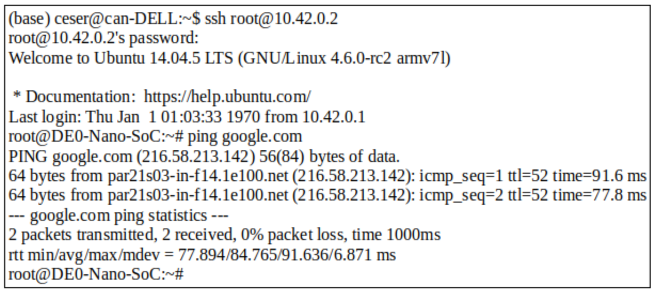

Reset
BootROM
Preloader
Bootloader
RTOS / Bare-metal Application
Reset
BootROM
Preloader
RTOS / Bare-metal Application
Reset
BootROM
RTOS / Bare-metal Application

Figure 3: Bare-Metal Boot Options

For Lab 3.0, the goal was to try having bare-metal application running on Cyclone V, but due to lack of time. We were not able to complete it. The goal of the laboratory was to try re-create previous lab where we used soft components like UART, and NIOS II Processor. As well, the macros used to address memory were suitable only for NIOS processor after being converted to assembly, which not readable by ARM processor within HPS.

IV. Results

Once our full system is fully configured, we can boot the Linux system through the Secure Shell (SSH) without using minicom: *ssh root@10.42.0.2*. We made a Ping test to check the internet connection sharing by Ethernet cable in order to have access to internet with our booted Linux system. Now that our full system is booted and fully configured, we can move on towards building a Linux application.



```
(base) ceser@can-DELL:~$ ssh root@10.42.0.2
root@10.42.0.2's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.6.0-rc2 armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Thu Jan  1 01:03:33 1970 from 10.42.0.1
root@DE0-Nano-SoC:~# ping google.com
PING google.com (216.58.213.142) 56(84) bytes of data.
64 bytes from par21s03-in-f14.1e100.net (216.58.213.142): icmp_seq=1 ttl=52 time=91.6 ms
64 bytes from par21s03-in-f14.1e100.net (216.58.213.142): icmp_seq=2 ttl=52 time=77.8 ms
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 77.894/84.765/91.636/6.871 ms
root@DE0-Nano-SoC:~#
```

Figure 4: Boot the linux system and Ping test.

Conclusion

The Lab 3 of "Projet de Systems-on-Chip CS-309" allowed us to learn about the mechanics involved in installing a Linux operating system on Cyclone V. Moreover, we released the main differences and advantages provided by running bare-metal application or application on top of a Linux system. It seems that this later provides more flexibility and robustness to run powerful and easy to maintenance application on Cyclone V like multi-threading code, running code with different programming languages (must be ARM compatible) and easy access to peripherals comparing to bare-metal application where we have to code drivers from scratch in order to set-up connections between peripherals.

Additionally, the crucial role of the documentation and design tools manuals to learn more about the tools which can make the production process more easy and very effective using RTFM technique to be a competent engineer.

References

Lab Handouts :
[1] R. Beuchat, Hybrid Systems.
[2] R. Beuchat, Embedded Linux Systems.

Tutorials:
[3] SoC-FPGA Design Guide [DE0-Nano-SoC Edition].
[4] Cyclone V SoC Examples: SD Operating system.