Lab 4: Booting Linux on the Zybo board

Samuel Huang
ECEN 449

1. Introduction

The purpose of the lab is to boot-up Linux on the ZYBO Z7-10 board. This is applicable for future applications, by creating an embedded processor environment that runs an Operating System. This lab utilizes Vivado to Build a Zynq based microprocessor system that can run linux and compile a linux kernel with PetaLinux tools. With a First Stage Boot Loader and a Universal Boot Loader, we create a Zynq Boot Image which loads the operating system and other files to start the Zynq device. Where the First Stage Boot loader initializes the Processing System with the configuration data, the Universal Boot Loader holds the instructions to boot the Linux Kernel. This Zynq Boot Image is then loaded onto the FPGA utilizing an SD card.

2. Procedure

The first part of the lab creates the ZYNQ7 Processing System using block automation in Vivado. This process also adds the multiply IP from the previous lab to the base system. With this the bitstream is generated.

With the bitstream generated the next step is to create the Linux boot files using PetaLinux. PetaLinux is an embedded Linux Software Development Kit which manages the First Stage Boot Loader and Universal Boot Loader. After downloading PetaLinux and sourcing the environment settings we configured the generated bitstream from the first part of the lab. With that we generate the First Stage Boot Loader and Universal Boot Loader by building the PetaLinux project.

With the First Stage Boot Loader and Universal Boot Loader we then generate the ZYNQ boot image by combining the First Stage Boot Loader, Universal Boot Loader, and FPGA bitstream and load the resulting files onto the FPGA with a SD card. Once booted onto the FPGA we can open a terminal that controls the created Operating System to configure some Linux settings on the board.

3. Results

The lab goes through the steps of generating a bitstream that outlines the Zynq based microprocessor system, which is used by the Linux Software Development Kit to create the First Stage Boot Loader and Universal Boot Loader. This kit brings in all the necessary additional files to completely flesh out the Linux based system. With the bitstream, FSBL, and UBL we create a boot image which maps out and allows for us to run a Linux Kernel on a FPGA. Ultimately this lab sets up a processor system which can be used for further lab applications where Linux is helpful.

4. Conclusion

The overall process of the Lab was not too complicated since PetaLinux handled the configuration and creation of the FSBL,UBL, and Boot Image but was prone to errors and complications. Most of the problems came from the fact that the necessary files and programs to create the Linux Processor System were too big and had to be stored on the

Lab computers tmp section. This created many problems with files and PetaLinux directories being created by other students which caused naming conflicts and general confusion. Along with this the process to configure and build the project took up most of the time, causing mistakes and problems taking longer to fix. With these problems the lab still taught me the process of setting up a Linux Software Development Kit and using it to create a processor system that can be embedded onto the FPGA.

5. Questions

The local memory serves as the cache for the program providing fast, temporary data storage that can be easily accessed. This will store frequently used data, so that the system takes less time when trying to access it. For a standard motherboard this local memory exists as the CPU cache, which is embedded directly into the CPU.

Ater going through this process, if a file was created in the non volatile memory it would save after restarting the ZYBO Z7-10 board. Since the files created however are in the volatile memory they would not persist after restarting the board.

If we added another peripheral to the system after compiling the kernel we would have to regenerate the bitstream for the FPGA since that would change. We would have to reconfigure the project with this new bitstream to properly implement the added peripheral. We would have to rebuild the project, and recreate the boot image since they have to be based on this new FPGA system bit stream. Finally we would have to reload the boot image onto the FPGA. Ultimately most of the steps would have to be repeated so that the kernel can properly interact with the added peripherals.