Design of Heterogeneous Environment

Jared Bold

Department of Electrical Engineering

Graduate Research Paper

Rochester Institute of Technology

Contents

List of Figures

List of Tables

1. Abstract

2. Introduction

3. Background

3.1 Hardware - Xilinx ZC702

The hardware platform selected is the Xilinx ZC702 Evaluation Board, based around the Xilinx Zynq XC7Z020-1CLG484C. The board provides many standard peripherals such as 1 GB DDR3 RAM, 128 Mb QSPI flash memory, USB 2.0 transceiver, SD Card interface, USB JTAG interface, HDMI, Ethernet PHY with RJ-45 connector, and additional peripherals and user I/O as shown in Figure **??** [**?**]. The purpose of the evaluation board is primarily to allow for a system design to be tested prior to producing a custom board layout, and makes it an ideal development platform for this work.

The Xilinx Zynq XC7Z020-1CLG484C device combines a Processing System (PS) and Programmable Logic (PL) on the same chip, forming a System on a Chip (SoC). The PS is comprised of two ARM Cortex-A9 MPCore application processors (APUs), each containing a NEON co-processor, a 32 KB L1 instruction cache, and a 32 KB L1 data cache. The APUs share a common 512 KB L2 cache. The PL is designed using the Xilinx Artix-7 fabric, their middle-grade fabric, with 85K Logic Cells (LCs), 53,200 Look-Up Tables (LUTs), 106,400 Flip-Flops (FFs), 560 KB Block RAM, and 220

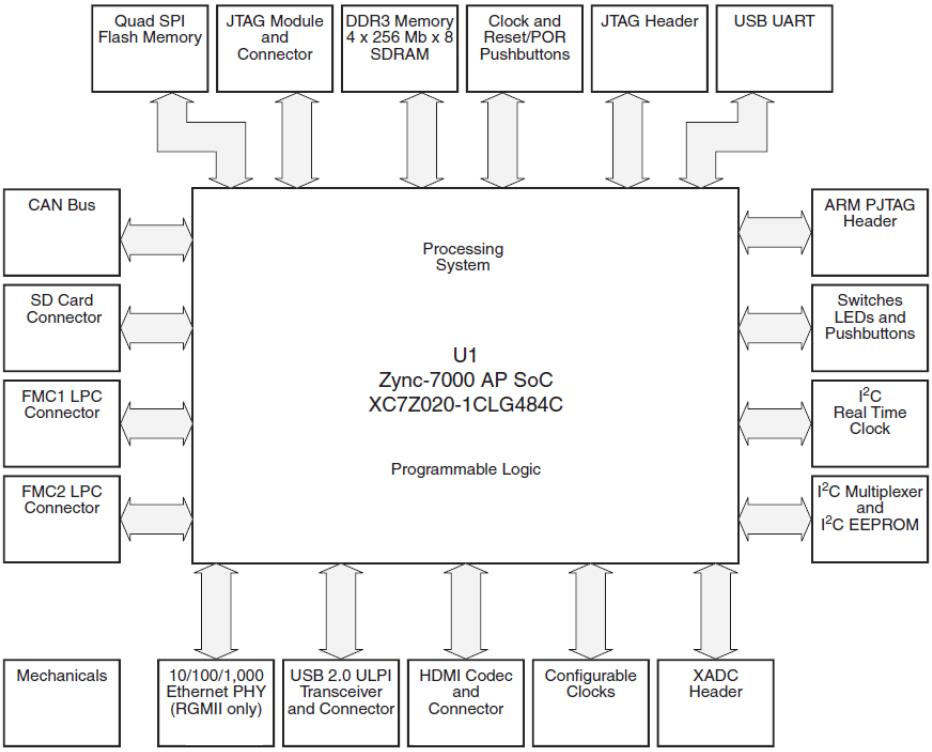

Figure 3.1: Top-Level Diagram of Xilinx ZC702 Evaluation Board

DSP Blocks [**?**]. Figure 3.2 shows the top level view of the composition of the Zynq device. Communication between the two portions of the chip occurs via a subset of the communcation buses specified by the Advanced Microcontroller Bus Architecture (AMBA), known as the Advanced eXtensible Interface (AXI). The PL can be user programmed to provide access to additional peripherals or to offload processing from the APUs.

The reason the ZC702 was selected is two-fold. The first being the use of the Xilinx Zynq device, which provides an ideal environment for building and testing homogeneous processing environments without limitations of using seperate CPU and FPGA chips. Secondly, the Zynq device and the ZC702 board have a large amount of support from the open source community. The

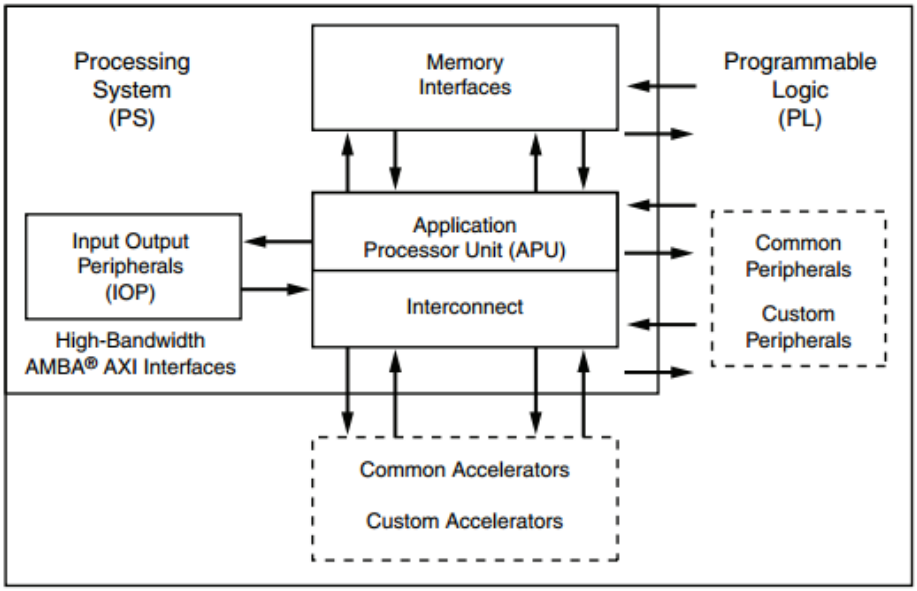

Figure 3.2: Top-Level Diagram of Xiliinx Zynq device

wealth of information and example code provide an excellent starting place.

3.2 Operating System - Linux 3.14

For purposes of this work, the Linux operating system has been selected to run on the ZC702 evaluation board. This Linux kernel is compiled directly from source maintained by Xilinx and is based off of the mainstream Linux kernel 3.14. The Linux kernel separates the operating system into two distinct spaces, the User Space and the Kernel Space. User applications run in the User Space and use calls to the system call interface and the GNU C Library to communicate and interact with the Kernel Space, as shown in 3.3. In the Kernel Space, system resources such as memory and peripherals are accessible and can be interacted with. The mememory seen within the User Space is known as a virtual memory space which maps to a physical memory

in the Kernel Space. The Linux kernel provides several features including process management, memory management, a virtual file system, a network stack, and device drivers [**?**]. These features allow for easier development and interaction between User Space applications and the hardware running underneath.

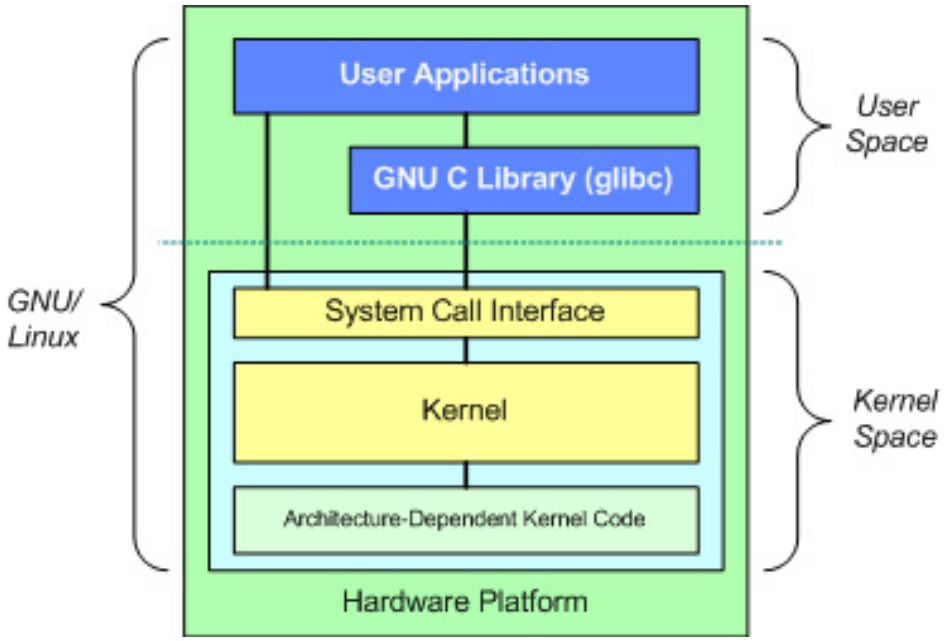

Figure 3.3: View of Linux kernel

Using Linux has several advantages over writing a bare-metal application. One of the greatest benefits comes from the portability and ease of access to both development tools and open source applications. An application that had previously been compiled for a Linux operating system can be recompiled to run in the embedded environment. These applications can range from image processing libraries (libtiff) to communicaitons (ssh) to file transfer (ftp) to any number of other things. The tools needed to do this, such as gcc, are readily available and open source, making them free to use. In

addition to applications, Linux provies device drivers for much of the hardare in use today, and if a driver does not exist, one can be created based on an existing driver allowing for easy access and interaction with hardware peripherals. This is of great benefit when compared with having to write to device registers by hand in a bare-metal application.

In addition to device drivers, the Linux kernel provides a method of directly mapping physical memory in the Kernel Space to virtual memory in a User Space application using the mmap system call. This proves very useful in SoC applications where the peripherals are memory mapped and may change based on the configuration of the device. Using this mapping, the registers of an external device can be directly written to from the User Space, bypassing the need to develop a custom device driver and allowing for faster development and testing.

3.3 AXI Standards

3.3.1 Stream Interface

3.3.2 Memory Mapped Interface

3.4 Image Filtering

3.5 Bilinear Interpolation

Bilinear interpolation (BI) is the process by which an unknown value is interpolated based on a 2 dimensional interpolation process using 4 known

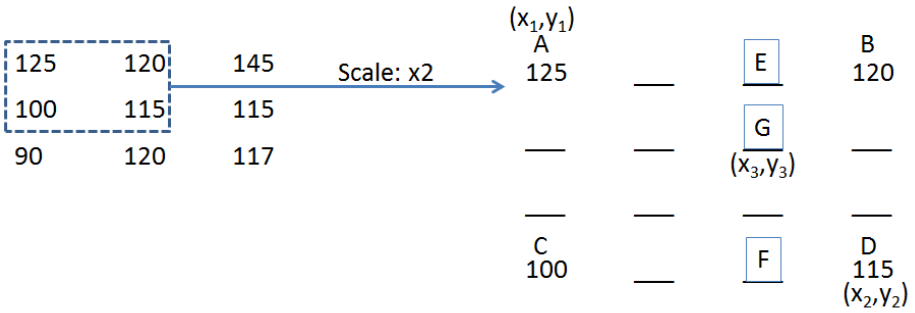

Figure 3.4: Bilinear Interpolation of Grayscale Image

values. In its application in image processing, BI is used to scale images up and down. Unlike nearest-neighbor interpolation, which relies on only one known value resulting in blocky images, BI has creates smoother images when scaling due to its estimation of intermediate values.

The process of bilinear interpolation begins by expanding the known pixels to their scaled locations. This expansion process generates a window of unknown pixels, bounded by four corners of known pixels, as shown in 3.4. In order to calculate the the unknown pixel values, three equations must solved. To generate an final equation for the value, the variables shown in 3.4 will be used. The first equation is the horizontal interpolation of the value at E.

$$E = \frac{x_3 - x_1}{x_2 - x_1} B + \frac{x_2 - x_3}{x_2 - x_1} A \tag{3.1}$$

Next, the horizontal interpolation of the value at F is calculated.

$$F = \frac{x_3 - x_1}{x_2 - x_1} D + \frac{x_2 - x_3}{x_2 - x_1} C \tag{3.2}$$

Finally, the vertical interpolation of value at G is calculated.

$$G = \frac{y_3 - y_1}{y_2 - y_1} F + \frac{y_2 - y_3}{y_2 - y_1} E \tag{3.3}$$

$$\alpha = x_3 - x_1, \beta = x_2 - x_3, \gamma = y_3 - y_1, \omega = y_2 - y_3 \tag{3.4}$$

By substituting (3.1) and (3.2) into (3.3) and allowing variables to be renamed as in (3.4), the BI equation for a pixel is determined.

$$G = \frac{1}{(\omega + \gamma)(\beta + \alpha)} [\gamma(\alpha D + \beta C) + \omega(\alpha B + \beta A)] \tag{3.5}$$

By applying (3.5) to every unknown pixel in every window that is generated from the scaling process, an image is enlarged or shrunk with minimal loss of quality. This technique is easily extended to multiple color channel images such RGB by operating on each colorplane individually and ensuring that pixel offsets are correctly calculated.

4. Design

4.1 Common Design Components

4.1.1 Tagged Image File Format

All images used throughout this research are Tagged Image File Format (TIFF) images. TIFF images consist of an image header that describes the byte order, TIFF version number, and offset to the image file directory[2]. The image file directory stores the offsets to the directory entries and varies in size based on the number of images contained within the TIFF image. The directory entry is then broken down into several sections including a Tag section which consists of a number of tags which give information about the image including the bits per pixel, compression scheme, image length and height, and many more. The directory entry is lastly completed with the offset to the image data. Figure 4.1 shows a more indepth representation of the TIFF structure for multiple image TIFFs.

For the purposes of this research, only uncompressed TIFF images are used in order to eliminate the need for resource intensive decompression and compression of input and output images. To support the reading and writing

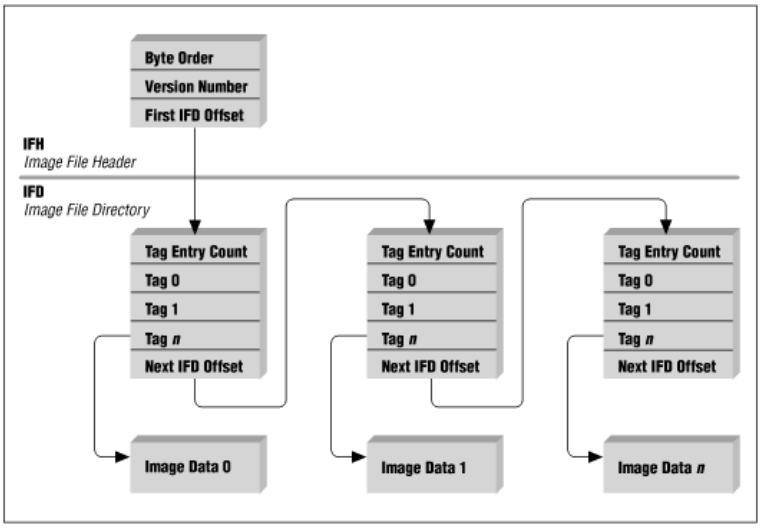

Figure 4.1: TIFF File Structure[1]

of these image files, the LibTiff libary is cross-compiled for use on the ZC702's PetaLinux OS.

4.1.2 Cross-Compilation

4.1.3 User Interface

4.2 Processor Based Processing

4.3 Programmable Logic Based Processing

4.4 Homogeneous Processing

5. Results

5.1 3x3 Gaussian Filter Results

5.2 9x9 LoG Filter Results

5.3 Bilinear Interpolation Results

6. Conclusion

Bibliography

[1] James D. Murray and William Van Ryper. *Encyclopedia of Graphics File Formats, 2nd Edition.* O'Reilly Media, 1996.

[2] Richard H. Wiggins, H. Christian Davidson, H. Ric Harnsberger, Jason R Lauman, and Patricia A. Goede. Image file formats: Past, present, and future. *RadioGraphics*, 21:789–798, 2000.

Appendices

A. Useful Resources

A.1 Programmable Logic

A.2 Operating System

A.3 Vivado

B. Verilog Sources

C. C Source

D. Reproduction of Work