**CHAPTER III**

**METHODOLOGY**

**3.1 System Requirements**

The research required the development of a System that for any compatible Input Image, an Output Image is produced for which the Face Detection filter would be applied. An indication of the Face that appears on the Image could be by way of the putting rectangle overlay over the Face that appears in the image. A very simple diagram of this is shown in **Figure 9**. In this research, in order to have fair treatment of multiple tests, the Face Detection sample program of the OpenCV Library was used as a standard algorithm. In addition, for statistical test, the standard test Image is a picture of Lena Söderberg for which many other Face Detection System Studies are using as a standard test image.

**Figure 9.** Top Level block diagram of the Entire System. The face detection system component is expanded in Appendix A.

**3.2 Top Level Face Detection System Design Overview**

The system design involved hardware and software abstraction and a clear separation between systems that interact with each other. At the bottom is the hardware layer in which the board and its peripherals such as the SoC, and IC Controllers are located. There is then an integration layer that is the Raw Binary File (RBF) which configures the FPGA fabric of the Hardware Layer. The OS Layer which lives in the Memory Card image is then configured to work in conjunction with the RBF which is also called by uBoot during System Boot. Refer to the diagram below (**Figure 10**) for the overall system design.



**Figure 10.** Top Level System Overview (See APPENDIX A for the Expanded Diagram)

**3.3 Hardware Preparation**

In order to begin the research the board which has different configuration has to be properly prepared by way of setting the proper BOOTSEL and CLKSEL settings header found on its TOP and BOTTOM side of the PCB. These headers are intended for configurability of the board boot process, boot source and clock distribution. Please Refer to **Appendix D** for the proper Board Configuration.

**3.4 Development Environment Preparation**

The researcher used the Altera Web-edition 13.1 Release of the Altera Design Suite. It is a free version of the Alter Design suite with Quartus and QSsys System. In addition the researcher developed a Linux Build Environment by way of a Virtual Desktop environment implemented on Virtual Box Open Source Virtualization Software. The build environment is based on the Open Source Ubuntu Linux. There are many more tools and utilities used in this research but some of them have trivial purposes and even some are an integral part of either the Windows OS or the Ubuntu based Linux OS. Shown in **Figure 11** are the development tools required for both Windows and Linux environments.



**Figure 11.** Development Tools

**3.5 Integration Layer**

**3.5.1 Golden Hardware Reference Design**

Using the QSYS tools, the IP Blocks necessary to emulate the Golden Hardware Reference Design was compiled in to Verilog. After generation, the produced Verilog file with references to the required Altera and 3rd Party IP Blocks was Analysed and Synthesized using Quartus. In addition an included PIN Placement TCL script was utilized which was produced by QSYS. After Analysis and Synthesis, a Full Quartus Compile was done. Using Altera’s BSP Editor which was included in SoC EDS, the preloader source files were produced. After that they are compiled using Altera’s Built in *make* and *GCC*.

**3.6 OS Layer**

**3.6.1 Modified Linux Kernel Source**

Altera provides an example Linux Kernel through Rocket Boards Org, however the Kernel available is very limited and required extensive modification. This opted the researcher to clone the Linux Kernel Source they have made available at Github instead. The main advantages with Altera’s Kernel source are the following:

1. Improved Frame buffer using the source copied from NIOS2 tree with the addition of some minor modifications in order to make it compatible with Cyclone V SoC.
2. The Kernel Build configuration was already modified to make sure some of the key systems are by default enable. This caused only a few minor adjustments to be made with the menuconfig utility after the socfpga\_defconfig was ran.
3. The kernel device tree is now based on the one produced from the Altera Golden Hardware Reference Design (GHRD).

Using menuconfig, several modifications to the Linux Kernel configuration was made. Among the vital modifications is making sure the USB and UVC Kernel Modules are properly configured. Using the Linaro GCC the Linux Kernel Source and the Kernel Modules was compiled.

**3.6.2 U-Boot Configuration**

Using the PuTTY console, The Board’s U-Boot was also configured. The Linux kernel requires that the FPGA be configured with the video pipeline prior to the video frame buffer driver being implemented. The FPGA configuration is implemented by the u-boot fpgaload command. And the notes on Appendix E was added to U-BOOT. This makes sure that U-Boot understands and detects the location of the appropriate Kernel Drivers and Kernel Compiled code when booting.

**3.6.3 SD Card Image**

**3.6.3.1 The Debian Based Linaro Ubuntu Image**

The Debian based Image of Linaro Ubuntu was downloaded from the Linaro Org. This was extracted and stored on a specified location on the Linux Build environment.

**3.6.3.2 Partitioning of the SD Card Image**

Using the Ubuntu Linux Based Software Development OS, the MicroSD Card was identified and partitioned. An excerpt from RocketBoards is on Appendix F on how to partition an SD Card using Linux.

**3.6.3.3 System Files**

The preloader, was placed in the binary partition. The Linux kernel, Device Tree and FPGA RBF respectively the soc\_system.dtb, soc\_system.rbf, uImage are placed on the FAT 32 Partition. And finally, the entire Linaro Root Filesystem is copied to the Linux partition. In summary the Boot Process operates as described in the image in **Figure 12**.



**Figure 12.** Boot Process

**3.6.4 Utilities**

The Linux System now installed the Board is capable of booting in to Linux. After booting, the important utilities are installed and compiled. Key utilities are the SSH for network connection, compiler utilities, and the X11, Synergy and XServer family of utilities for the user interface.

**3.6.5 OpenCV Library**

Finally, in this stage, the OpenCV library source and its associated dependencies are downloaded through Ethernet directly to the SD Card. After installing all the development tools necessary within the SoCKit Linux, OpenCV and its associated libraries are compiled. Key libraries are FFMPEG, V4L2 and QT. After compilation the Face Detect example program is modified to fit the environment. After modification the Face Detect program is ran and tested.