

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Yi Zheng**

**A NEW DESIGN FOR THE X-RAY LINE-SCAN DETECTOR SYSTEM SOFTWARE**

Master’s Thesis

Degree Programme in Computer Science and Engineering

<May 2014>

**Zheng Y. (2014) A new design for the X-ray line-scan detector system software.** University of Oulu, Department of Computer Science and Engineering. Master’s Thesis, 57p.

****ABSTRACT****

**The X-ray technic has been used in a wide range of industrial products, such as medical CT scanners, luggage security scanners, container scanners, industrial CT scanners, and so on. The task of making these X-ray scanners cheaper, but with an acceptable level of performance, is an important subject for the industry.**

**The aim of this thesis is to design a low cost X-ray line-scan detector, with acceptable performance, high flexibility, and excellent reliability. To achieve these requirements, special attention has been given to the design of the detector’s system firmware and software.**

**In this thesis, various software designs have been studied for noise reduction, higher processing speed and increased reliability. In most commercial X-ray line-scan detectors the digital process work is done by FPGA or DSP processors, both of which add to the cost of the detector. In this thesis, in order to meet the cost and flexibility requirements of different customers, we use personal computer software to implement the digital signal process work of the detector. The X-ray line scan detector prototype has been delivered for industrial testing, and the image results are presented and analyzed with mathematical software.**

**Keywords:** **X-ray line-scan detector, Nios II, uCLinux, Calibration, ApponsView, Firmware, Multi-core CPU**

**Zheng Y. (2014) Uudentyyppistä X-ray viivakamerat ilmaisin laiteohjelmiston.** Oulun yliopisto, tietotekniikan osasto. Diplomityö, 57s.

****Tiivistelmä****

**Röntgen kuvannusta käytetään useissa erinlaisissa sovelluksissa, kuten lääketieteen tietokone tomografia (CT) laitteissa, matkalaukkujen läpivalaisussa lentokenttien turvatarkastuksissa, rahtikonttien läpivalaisussa, teollisuuden CT skannerit ja niin edelleen. Näiden kuvantamisjärjestelmien tekeminen kustannustehokkaasti, mutta riittävällä suorituskyvyllä on tärkeää.**

**Tämän opinnäytetyön tarkoituksena on suunnitella edullinen rontgen viivakamera varustettuna riittävän hyvällä suorituskyvyllä, joustavuudella ja erittäin korkealla luotettavuudella. Näiden tavoitteiden saavuttamiseksi suunnittelussa on kiinnitetty huomiota viivakameran sulautettuun ohjelmistoon ja sovellus ohjelmistoon.**

**Tässä työssä on tutkittu erilaisia ohjelmisto osia kuvakohinan pienentämiseksi, kuvadatan käsittelyn nopeuden lisäämiseksi ja järjestelmän korkean luotettavuuden saavuttamiseksi. Useimmissa kaupallisissa röntgen viivakameroissa on käytössä FPGA tai DSP prosessitasoinen ratkaisu kuvadatan kasittelyyn. Jotta voidaan vastata eri asiakkaiden joustavuus- ja kustannus vaatimuksiin, tässä työssä käytetään edullista ja joustavaa ratkaisua, jossa digitaalinen kuvadata prosessoidaan tietokoneessa. Tähän ratkaisuun perustuva röntgen viivakamera prototyyppi on toimitettu teolliseen testaukseen. Testauksessa saadut tulokset on analysoitu ja esitetty matemaattisen sovelluksen avulla.**

**Avainsanat: Röntgen viivakamera, NiosII, uClinux, Kalibrointi, ApponsVIew, Sulautettu ohjelmisto, moniydinprosessori**

# ****TABLE OF CONTENTS****

Abstract

Tiivistelmä

Table of contents

Foreword

Abbreviations

1. INTRODUCTION 8

1.1. Background 8

1.2. Scope of This Thesis Work 9

1.3. The Author’s Contribution 9

1.4. Report Organization 10

2. INTRODUCTION TO THE X-RAY LINE-SCAN DETECTOR 11

2.1. Introduction to the Imaging System 11

2.2. Operation Principle of the X-ray Line-Scan Detector 12

2.3. Block-Diagram and Functional Description 13

2.3.1. Detector Board 14

2.3.2. Front-End Board 14

2.3.3. Control Unit Board 15

2.3.4. Power Supply Unit 16

2.3.5. Host PC 16

3. INTRODUCTION TO THE CONTROL UNIT BOARD 18

3.1. Overview 18

3.2. Design of Control Unit Board 18

3.2.1. FPGA Technology 18

3.2.2. Nios II Processor 20

3.2.3. Embedded Linux / uCLinux 21

3.2.4. Network Model 22

3.2.5. Application layer protocol 23

3.2.6. Socket programming model 25

3.3. Firmware Implementation 26

3.3.1. Setup Cross-compiler Development Environment 27

3.3.2. Firmware flow 27

3.4. Discussion 29

4. DIGITAL SIGNAL PROCESSING 31

4.1. Overview 31

4.2. Re-arrange Pixel Order 32

4.3. Systematic error correction 32

4.3.1. Offset correction by dark frame subtraction 33

4.3.2. Gain calibration and correction 33

4.3.3. Baseline shift 34

4.3.4. Correction result 34

4.4. Image Denoising 35

4.4.1. White noise 35

4.4.2. Moving Average Filter 35

5. HOST PC SOFTWARE DESIGN AND IMPLEMENTATION 37

5.1. General Requirement 37

5.2. Component Architecture 38

5.2.1. ApponsControl Library Architecture 38

5.2.2. ApponsView Application Architecture 39

5.3. ApponsView Implementation 40

*5.3.1.* ApponsControl Library Implementation 40

*5.3.2.* ApponsView Application Implementation 43

5.4. Multi Core Optimization 43

5.4.1. Parallel Programming 45

5.4.2. OpenMP 46

5.4.3. Implementation with OpenMP 47

6. TEST RESULTS AND APPLICATIONS 48

6.1. ApponsView Test Result 48

6.2. Applications 50

7. Conclusions 52

8. References 53

9. Appendices 56

Foreword

This thesis was carried out in Appons Inc., during the year of 2014. I would like to express my deepest gratitude to my supervisor Professor Janne Heikkilä, for his warm advice and continuous encouragement. I would also like to thank Mr. Jian Zheng for his kind help and valuable advice.

This work is supported by Appons Inc. I have benefited significantly over the years from discussions with my gifted colleagues, especially Mr. Jian Zheng, who is the FPGA designer of the X-ray line-scan detector.

Finally, I would like to give special thanks to my wife. Thanks for your support and understanding all the time.

Oulu, 19.05.2014

Yi Zheng

ABBREVIATIONS

AC Alternating Current

AD Analog to Digital

ADC Analog Digital Converter

ARP Address Resolution Protocol

API Application Programming Interface

ASIC Application-specific integrated circuit

CPLD Complicated programmable logic device

CU Control Unit

DC Direct Current

DSP Digital signal processing

FE Front-end

FIR Finite impulse response

FPGA Field programmable gate logic

FPN Fixed pattern noise

GND Ground

HDL Hardware Description Language

IC Integrated circuit

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers, Inc.

IP Internet Protocols

I/O Input/output

LSB Least Significant Byte

LSI Large Scale Integration

MAC Multiplier accumulator

MAC Media Access Control

MACs Multiply-accumulators

MUX Multiplexer

PAL Programmable Array Logic

PC Personal Computer

PCB Printed Circuit Board

PGA Programmable Gain Amplifier

RAM Random Access Memory

RMS Root Mean Square

SoC System-on-Chip

SNI Serial Network Interface

SNR Signal Noise Ratio

SRAM Static Random Access Memory

S/H Sample and hold

S/N Signal/noise ratio

TCP Transmission Control Protocol

TTL Transistor-Transistor Logic

USB Universal serial bus

USART Universal Synchronous Asynchronous Receiver / Transmitter

VHDL VHSIC Hardware Description Language

VLIW Very Long Instruction Word

UDP User Datagram Protocol

# INTRODUCTION

## Background

Today there are many industrial inspections that utilize X-ray as a detect method. Its non-destructive and non-contact features enable it to be used widely in inspecting food for foreign objects, airport security checks, and medical inspections.

The X-ray line-scan detector, which is also known as *linear array*, has become more and more common in the field of non-destructive testing. X-ray line-scan cameras have been available for more than ten years and during these years the technology has developed towards higher scanning speeds, better dynamic range, and smaller pixel sizes[[[1]](#endnote-2)]. The linear array is suitable for inspecting moving objects on a belt, or for fixing the object and moving and mounting the scanner on a shifting plate. They are used to search for missing products in packages, for checking fill-levels, for inspecting the contents of a passenger’s luggage, and for finding foreign objects in food products. Figure 1 is a typical X-ray line-scan detector.



Figure 1 A typical X-ray line scan camera (courtesy of Detection Technology Inc.)

A typical X-ray line-scan detector consists of the following components: the scintillator, the photodiode array, the detector front-end, the data acquisition system, the control unit, the mechanics, the power supply, the accessories, the frame grabber card, and the software.

The scintillator is used to convert the x-ray radiation into visible light, with the photodiode serving to measure the amount of the light transformed by the scintillators [[[2]](#endnote-3), [[3]](#endnote-4), [[4]](#endnote-5)].

The front-end is used to integrate and amplify the very low current output from the photodiode arrays [5]. In the majority of cases, the scintillator[[[5]](#endnote-6), [[6]](#endnote-7), [[7]](#endnote-8), [[8]](#endnote-9)], photodiode array and front-end electronics are made into a detector board.

The control unit board is responsible for controlling the detector by receiving commands through Ethernet from the host computer. Furthermore, it also receives image data from the detector board and sends it to the host computer through Ethernet. In addition to this, it provides the required timing signals for the detector card [[[9]](#endnote-10)].

The Ethernet electronics take care of the communications between the X-ray line-scan camera and computer.

The mechanical parts include the enclosure, the X-ray beam collimator, the X-ray entrance window, and the X-ray shielding for electronics. The X-ray line-scan scanner mechanics are shown in Figure 2.

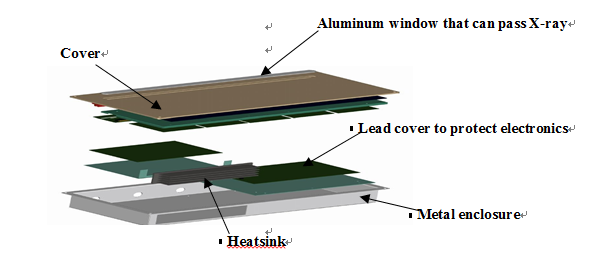


Figure 2 Mechanics of a typical X-ray line-scan detector.

An X-ray line-Scan scanner is suitable for real-time X-ray photography, in which objects move on conveyor belts.

## Scope of This Thesis Work

During the last few years, the desktop personal computer’s CPU performance has progressed greatly, especially the Multi-core CPU technology. This makes it possible to move all the digital image processing work from the detector side to the host PC side. The cost of the hardware is thereby lessened considerably.

The purpose of this thesis is to study how to use software to do the digital signal processing work that reduces system noise and processes reliable customized data for the system, in areas such as normalization and signal enhancement.

In this thesis, a description of the X-ray line-scan detector system is presented first. This is followed by a short introduction to both the digital signal processing concept that is used for the detector, and how this theory is implemented. The final aim of this study is to develop an X-ray line-scan system with a lower price than what is currently available on the market, an acceptable level of performance, and the flexibility to be optimized or upgraded in the future.

## The Author’s Contribution

The X-ray line-scan detector from Appons Inc. has been developed during this current year. The X-ray line-scan detector prototype has now been delivered to the customer, and the production line has been prepared.

This thesis will focus on the firmware and software design part of the X-ray line-scan detector. The author independently designed the firmware and software of the system. Some hardware parts are also covered in this thesis. They consist of the detector board and the front-end board design. The author’s contribution includes the following:

1. Designing the firmware of the control unit board, including the command set of the detector;
2. Designing the communication protocol between the PC and the detector;
3. Designing the PC software, including the library with basic raw data processing filter for the detector and the application to show the functions of the detector;
4. Designing a detector simulator, running under Linux for test purposes;
5. Implementing and testing the above design;
6. Performance tuning of the software and firmware.

## Report Organization

Chapter 1 is an introduction to the work. It describes the scope of the thesis and gives some background knowledge of the X-ray line-scan detector. Chapter 2 provides a more detailed introduction to the X-ray line-scan image system. An introduction to the control board and firmware is given in Chapter 3. Chapter 4 introduces the digital process algorithm used by X-ray line-scan image system. Chapter 5 describes in detail the host side software implementations for the detector. Chapter 6 lists the measured results and discusses different applications of the X-ray line-scan detector. The conclusion is given in Chapter 7.

# INTRODUCTION TO THE X-RAY LINE-SCAN DETECTOR

## Introduction to the Imaging System

The X-ray line-scan detector is designed to be used for the inspection of moving objects that generally travel at a constant speed. It is a type of linear-array detector.

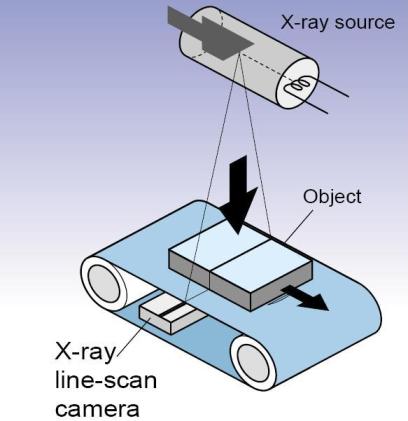


Figure 3 Operation principle of a typical line scan imaging system.

Figure 3 shows how a line-scan detector scans an image. Just as a normal optical scanner captures a single line one at a time, so the moving of the scanner or object will let detector scan a whole object to get the final image. This kind of detector is suitable for the inspection of food, electronic components or luggage, as X-rays can detect – without contact or destruction – defects or foreign objects that are invisible to the naked eye.



Figure 4 A typical X-ray application environment

The main components of a typical imaging system are shown in Figure 4:

1. A conveyor which transports the object across the detector.

2. An X-ray generator and its control system.

3. A personal computer equipped with a network interface.

4. Necessary mechanics for system integration and radiation shielding.

5. An X-ray line-scan detector and a power supply unit.

At the highest point, there is the X-ray generator which is placed on the top of the cabinet. The X-ray line-scan detector with photodiode array is assembled in a line inside the metal enclosure. The control unit board and other electrical board are also assembled in the same metal enclosure. The X-ray line-scan detector is placed under the conveyor belt. When the scanned objects are moving along the belt, the X-ray beam goes through the objects and is captured by the detector. The detector will convert the X-ray signal to a digital signal and send it to the personal computer. The personal computer then receives the image data and processes it according to the requirement.

## Operation Principle of the X-ray Line-Scan Detector

Figure 5 shows the data flow of the X-ray line detector. The key components are the detector board, the front-end board and the control unit board. They are responsible for transforming the X-ray’s radiation to a digital signal which can be used by the computer.

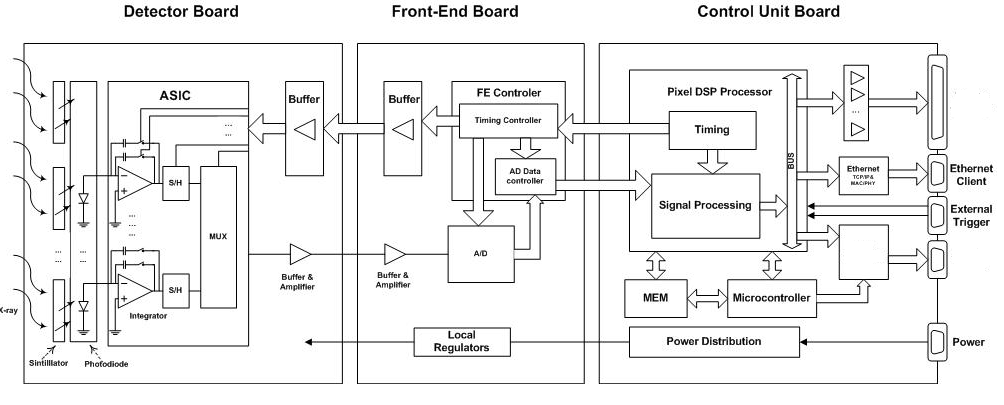


Figure 5 Operation principle of X-ray line-scan detector

The silicon photodiodes can only detect visible light, so the incoming X-ray must be transformed to visible light by scintillators in a crystal array. The photodiode can then accurately measure the visible light.

On top of the photodiode, there is a scintillation screen. An individual photodiode is known as a pixel. The spacing between pixels is called pixel pitch, the distance of which generally varies from 0.2 to 2.5 mm.

Each pixel of the detector provides a signal current that is proportional to the flux and energy of the X-rays absorbed in the respective pixels themselves. The photodiode converts the light into photocurrent with the object information. It is at this point that the front-end board converts the current to analog voltage by ASIC chips.

Timing control defines the integration time of the front-end board. The output is an analog voltage which is proportion to the total integrated current. The voltage is put into an A/D converter and the output of the A/D converter is then read into front-end board registers.

The control unit board reads the front-end board’s register to process the pixel’s response data into its own sending buffer. After two lines of data are ready, it will set up a flag to notify the firmware that the data is ready. The firmware will then send the pixel data to the socket buffer.

## Block-Diagram and Functional Description

Figure 6 is a block diagram of the X-ray line-scan detector system.

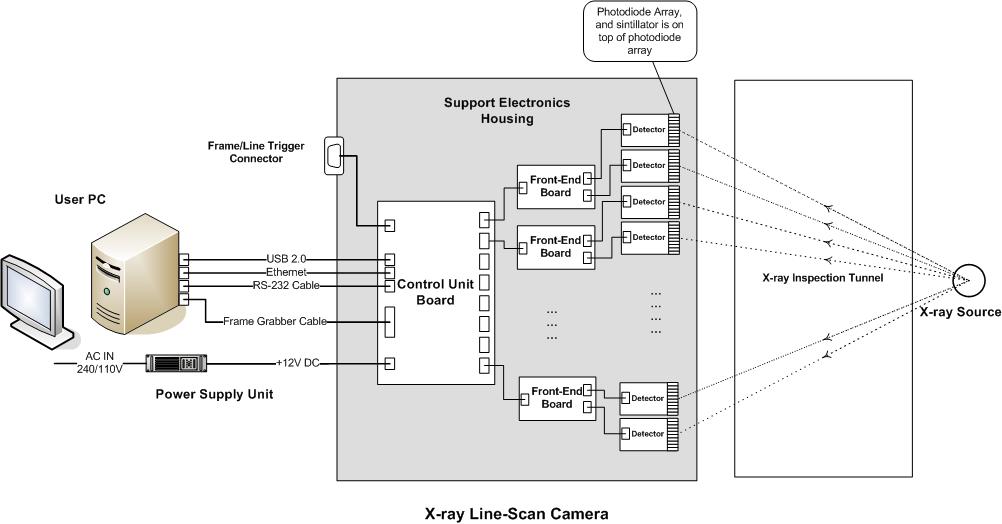


Figure 6 Block diagram of the X-ray line-scan detector system

Normally, an X-ray line-scan detector has detector boards, A/D or Front-End boards (FEs), a control unit board (CU) and, optionally, a DC power supply. These are shown in Figure 6.

The detector boards are mounted in the mechanic. They are under a collimator, along with the detector’s window. The active area plane of the detector elements is under the X-ray beam from the focal spot of the X-ray source. One front-end board connects two detector boards, while all front-end boards are connected to the control unit board. Because of the physical area limitation of the CU board, eight FE boards at most can be connected to one CU board. All detector boards work in parallel. Every FE board multiplexes the analog signals from two detector boards and converts them to digital signals. After that, the FE boards send the digital signals to the CU board. All of the FE boards also work in parallel. The CU board receives the data from these FE boards, and arranges the order of the pixel data according to the physical positions of the pixels.

The computer software performs the necessary digital signal processing, such as the re-sequenced pixel data, average filter and calibration. The customer can then apply their own image processing method to this processed image data.

### *Detector Board*

Figure 7 shows the operation principle of a photodiode. A photodiode is a semiconductor device that converts [light](http://en.wikipedia.org/wiki/Light) into [current](http://en.wikipedia.org/wiki/Electric_current), and a photodiode array consists of several photodiodes arranged in a line on the same plane. It is common for one detector board to have 64 or 128 photodiodes.  When a [photon](http://en.wikipedia.org/wiki/Photon) of sufficient energy strikes the diode, it creates an [electron](http://en.wikipedia.org/wiki/Electron) and a pair of [hole](http://en.wikipedia.org/wiki/Electron_hole)s. This mechanism is also known as the inner photoelectric. If the absorption occurs in the junction's [depletion region](http://en.wikipedia.org/wiki/Depletion_region), or one diffusion length away from it, these carriers are swept from the junction by the built-in electrical field of the depletion region. Thus, holes move toward the [anode](http://en.wikipedia.org/wiki/Anode) and electrons toward the [cathode](http://en.wikipedia.org/wiki/Cathode), and a [photocurrent](http://en.wikipedia.org/wiki/Photocurrent) is produced. The total current through the photodiode is the sum of the dark current (current that is generated in the absence of light) and the photocurrent, so the dark current must be minimized to maximize the sensitivity of the device [[[10]](#endnote-11)].



Figure 7 Operation principle of photodiode array.

### *Front-End Board*

Figure 8 is a block diagram of the front-end board. The front-end board consists of the AD converter, the FE controller, the power circuit, the amplifier circuit and other related buffer circuits.

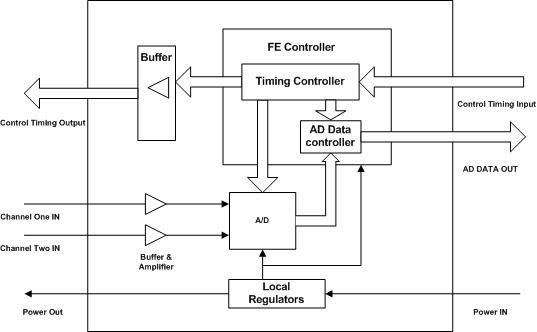


Figure 8 Block diagram of the front-end board.

The output signal of the detector board is an analog signal which cannot be used by the computer. The front-end board converts the analog voltage signals into digital signals. Each front-end board is normally connected to two detector boards

There is a three-channel CCD processor in the front-end board, which is used for AD conversion. The analog video outputs of the two detector boards are multiplexed into an AD converter in the CCD processor, and then multiplexed digital data can be read out from the digital output of the CCD processor. The resolution of the A/D conversion is 14-bit.bits[[[11]](#endnote-12)]. The analog voltage signals are read pixel by pixel from the detector boards. The front-end controller is the main control component in the front-end board. To meet the flexibility requirements, the front-end controller is implemented by a CPLD chip.

The front-end controller generates control timing to control the analog read out speed. It also generates the control timing of the CCD processor. Thus, the front-end controller could control the synchronization between the output of the detector board and A/D sampling timing. Finally, the digital output from the CCD processor is buffered, and is then sent serially to the control unit board using differential links.

### *Control Unit Board*

A block diagram of the control-unit board is shown in Figure 9. It receives the data serially from the front-end board through differential links and converts the serial digital signal to 16-bit data for each pixel. When there are two lines of data, the control-unit board will send the data to the computer for further processing. The control-unit board is also responsible for processing the command from the computer. After the command is processed, the control-unit board sends the feedback to the computer.

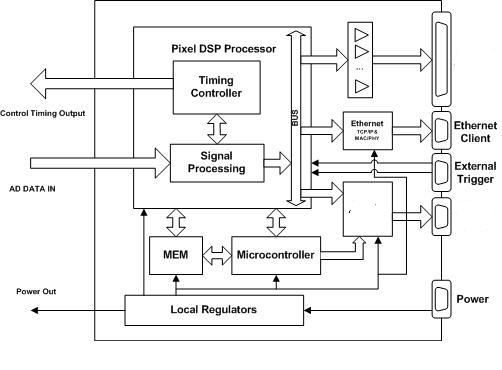


Figure 9 Simplified block diagram of control unit board.

The physical transfer layer between the control unit board and the personal computer is Ethernet. Compared with the frame grabber card and USB, Ethernet has the advantage in cost, flexibility, and transport range. Furthermore, for our customer, the bandwidth of the Ethernet is acceptable for their application, which makes Ethernet the optimum choice for the connection between the computer and the detector. This kind of design lessens the cost of the board and also simplifies the development work for FPGA and PCB design. In reference to this facet of the work, more detail will be provided in Chapter 3 of this thesis.

### *Power Supply Unit*

The power supply unit converts the supply voltage (AC 110/240 V) into +12V DC voltage, which can be used by the electronics of the X-ray line-scan detector. The control-unit board and the front-end boards generate internal voltages by using local regulators from the +12V DC power supply.

### *Host PC*

A remote host PC is needed in our system to control the X-ray line-scan detector and receive image data from the detector. The host PC will require specialized software to cooperate with the X-ray line-scan detector.

To make the detector operational, there are certain parameters – such as integration time and overall gain – that need to be set before the detector starts scanning. The host PC software acts as a control interface for the detector, and its second main function is to decode raw data from the detector. The raw data is packed into DTP format (which will be explained in Chapter 3) and cannot be used directly as image data. It must first be unpacked and re-sequenced in a predefined order before we do any digital image processing work.

# INTRODUCTION TO THE CONTROL UNIT BOARD

## Overview

The control unit board is a very important part of the X-ray line-scan detector. It provides control timing signals for the detector board and transfers data from the front-end board to the computer. Its main functions are:

* To receive the raw data from the AD or front-end board and send it to host PC;
* To provide control timing signals for the whole system;
* To provide power for the control unit board and front-end boards;
* To communicate with the host PC.

Keeping our aims of low cost, acceptable performance and high flexibility in mind, we will discuss how to meet the above targets in the architecture design of the control unit board.

## Design of Control Unit Board

There are several ways to meet the system requirements, such as DSP, ASICs, and general-purpose processors. However, after comparing the cost and flexibility of each option, we opted for the FPGA solution as it provides exceptional performance advantage at a low cost.

### *FPGA Technology*

A field-programmable gate array (FPGA) is an [integrated circuit](http://en.wikipedia.org/wiki/Integrated_circuit) designed to be configured by a customer or a designer after manufacturing, hence the term ‘[field-programmable](http://en.wikipedia.org/wiki/Field-programmable)’. The FPGA configuration is generally specified using a [hardware description language](http://en.wikipedia.org/wiki/Hardware_description_language) (HDL) similar to that employed for an [application-specific integrated circuit](http://en.wikipedia.org/wiki/Application-specific_integrated_circuit) (ASIC). [Circuit diagrams](http://en.wikipedia.org/wiki/Circuit_diagram) were previously used to specify the configuration, as they were for ASICs, but this practice is increasingly rare.

Contemporary FPGAs have large resources of [logic gates](http://en.wikipedia.org/wiki/Logic_gate) and RAM blocks to implement complex digital computations. As FPGA designs employ incredibly quick I/Os and bidirectional data buses, it becomes a challenge to verify the correct timing of valid data within setup time and hold time. [Floor planning](http://en.wikipedia.org/wiki/Floor_planning) enables resource allocation within FPGA to meet these time constraints. FPGAs can be used to implement any logical function that an ASIC could perform. The aforementioned ability to update the functionality after shipping, [partial re-configuration](http://en.wikipedia.org/wiki/Partial_re-configuration) of a portion of the design[[[12]](#endnote-13)] and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain [programmable logic](http://en.wikipedia.org/wiki/Programmable_logic_device) components called *logic blocks*, and a hierarchy of reconfigurable interconnects that allow the blocks to be ‘wired together’. In other words, many (changeable) logic gates can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex [combinational functions](http://en.wikipedia.org/wiki/Combinational_logic), or merely simple [logic gates](http://en.wikipedia.org/wiki/Logic_gate) like [AND](http://en.wikipedia.org/wiki/AND_gate) and [OR](http://en.wikipedia.org/wiki/XOR_gate). In most FPGAs the logic blocks also include memory elements, which may be simple [flip-flops](http://en.wikipedia.org/wiki/Flip-flop_%28electronics%29) or more complete blocks of memory.

Some FPGAs have analog features in addition to digital functions. The most common analog feature is programmable [slew rate](http://en.wikipedia.org/wiki/Slew_rate) and [drive strength](http://en.wikipedia.org/w/index.php?title=Drive_strength&action=edit&redlink=1) on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise [ring](http://en.wikipedia.org/wiki/Electrical_resonance) unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slowly. Another relatively common analog feature is differential comparators on input pins designed to be connected to [differential signaling](http://en.wikipedia.org/wiki/Differential_signaling) channels. A few ‘[mixed signal](http://en.wikipedia.org/wiki/Mixed-signal_integrated_circuit) FPGAs’ have integrated peripheral [analog-to-digital converters (ADCs)](http://en.wikipedia.org/wiki/Analog-to-digital_converter) and [digital-to-analog converters (DACs)](http://en.wikipedia.org/wiki/Digital-to-analog_converter) with analog signal conditioning blocks allowing them to operate as a [system-on-a-chip](http://en.wikipedia.org/wiki/System-on-a-chip) [[[13]](#endnote-14)]. Such devices blur the line between an FPGA, which carries digital ones and zeroes on its internal programmable interconnect fabric, and [field-programmable analog array](http://en.wikipedia.org/wiki/Field-programmable_analog_array) (FPAA), which carries analog values on its internal programmable interconnect fabric.

To define the behavior of the FPGA, the user provides a [hardware description language](http://en.wikipedia.org/wiki/Hardware_description_language) (HDL) or a [schematic](http://en.wikipedia.org/wiki/Schematic) design. The HDL form is more suited to work with large structures because it is possible to simply specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualization of a design [[[14]](#endnote-15)].

Next, a technology-mapped [net list](http://en.wikipedia.org/wiki/Netlist) is generated using an electronic design automation tool. The net list can then be fitted to the actual FPGA architecture using a process called [place-and-route](http://en.wikipedia.org/wiki/Place_and_route), commonly performed by the FPGA company's proprietary place-and-route software. The user will validate the map, place and route results via [timing analysis](http://en.wikipedia.org/wiki/Timing_analysis), [simulation](http://en.wikipedia.org/wiki/Simulation), and other [verification](http://en.wikipedia.org/wiki/Verification_and_validation) methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA. This file is transferred to the FPGA/CPLD via a [serial interface](http://en.wikipedia.org/wiki/Serial_communication) ([JTAG](http://en.wikipedia.org/wiki/Joint_Test_Action_Group)) or to an external memory device like an [EEPROM](http://en.wikipedia.org/wiki/EEPROM)[[[15]](#endnote-16)].

The most common HDLs are [VHDL](http://en.wikipedia.org/wiki/VHDL) and [Verilog](http://en.wikipedia.org/wiki/Verilog), although in an attempt to reduce the complexity of designing in HDLs – which have been compared in their level of difficulty to [assembly languages](http://en.wikipedia.org/wiki/Assembly_language) – there have been moves to raise the abstraction level through the introduction of [alternative languages](http://en.wikipedia.org/wiki/Hardware_description_language#HDL_and_programming_languages). [National Instruments](http://en.wikipedia.org/wiki/National_Instruments)' [*LabVIEW*](http://en.wikipedia.org/wiki/LabVIEW) graphical programming language (sometimes referred to as ‘G’) has an FPGA add-in module available to target and program FPGA hardware.[[[16]](#endnote-17)][[[17]](#endnote-18)]

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called [IP cores](http://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core), and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as [OpenCores](http://en.wikipedia.org/wiki/OpenCores) (typically released under [free and open source](http://en.wikipedia.org/wiki/Free_and_open_source_software) licenses such as the [GPL](http://en.wikipedia.org/wiki/GNU_General_Public_License), [BSD](http://en.wikipedia.org/wiki/BSD_license) or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the [RTL](http://en.wikipedia.org/wiki/Register_transfer_level) description in [VHDL](http://en.wikipedia.org/wiki/VHDL) or [Verilog](http://en.wikipedia.org/wiki/Verilog) is created by manufacturing test benches to simulate the system and observe results[[[18]](#endnote-19)]. Then, after the [synthesis](http://en.wikipedia.org/wiki/Logic_synthesis) engine has mapped the design to a net list, the net list is translated to a gate level description where simulation is repeated to confirm that the synthesis proceeded without errors. Finally, the design is mapped into the FPGA, at which point propagation delays can be added and the simulation run again with these values back-annotated onto the net list.

### *Nios II Processor*

*Nios II*, from Altera Inc., is a configurable 32-bit soft processor which replaced the old NIOS in 2004. It can accelerate time-critical software algorithms by adding custom instructions to its instruction set. Complex sequences of standard instructions can be reduced to a single instruction implemented in hardware[[[19]](#endnote-20)]. Figure 10 is a *Nios II* processor block diagram.

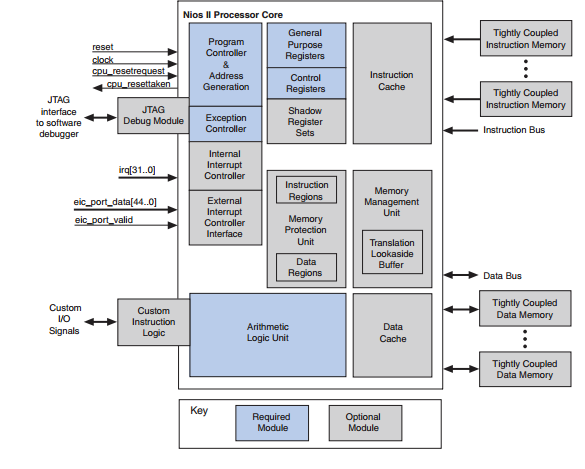


Figure 10 Nios II processor architecture.

*Nios II* custom instructions are custom logic blocks connected directly to the *Nios II* arithmetic logic unit (ALU). This allows for an easy way to tailor the *Nios II* core to meet the needs of a particular application. The simplicity of altering the design of the FPGA-based *Nios II* processor provides a user-friendly path to experiment with hardware/software tradeoffs at any point in the design process.

The *Nios II* supports different features of custom instructions:-

• Combinational: Single clock cycle custom logic blocks;

• Multi-cycle: Multiple clock cycle custom logic blocks of fixed or variable durations;

• Extended: Custom logic blocks that are capable of performing multiple operations;

• Internal Register file: Custom logic blocks that access internal register files for input and/or output;

• External Interface: Custom logic blocks that interface to logic outside of the Nios II processor’s data path.

The software interface is easily implemented through C-like functions. The custom logic block is interfaced including the ‘.h’ files generated by the Altera Quartus II software. In this way, the inclusion of new custom instructions in the *Nios II* processor requires just an updated compilation of the software since the ‘.h’ files are automatically generated by the Altera Quartus II software.

The ALU can be expanded to include a floating-point unit (FPU) that it is included as an option in the *Nios II* configuration profile. This option can be selected in order to perform floating-point instructions in hardware instead of compiling the respective subroutines in software.

The version of *Nios II* used in this thesis does not implement a virtual memory management unit (MMU). This limits the selection of operating systems available for the processor. *uCLinux* was the operating system chosen because it is a Linux version that does not require a processor with a memory management unit. While memory management units support a great deal of devices suitable for embedded applications, their inclusion would require the utilization of hardware resources that cannot be used for the implementation of other hardware devices.

.

### *Embedded Linux / uCLinux*

Most modern embedded applications necessitate the support of an operating system in order to provide flexibility, efficiency and reliability. An operating system allows the processor to be shared among multiple tasks. Consequently, the designer may implement the application as a set of tasks that has to be executed in order to cover all the different functionalities.

Multitasking requires a special management of the memory resources in order to avoid a task gaining access to the memory space of another task, thereby jeopardizing its execution. When the operating system has to take into account that undesirable memory access may be produced due to malicious or untested reasons, then a hardware memory management unit is mandatory. However, when all the tasks of the system are well known, as in most of the embedded applications in which the designer specifies all of the tasks to be executed, the need for a hardware memory management unit is lessened considerably. Consequently, the application can be implemented on more simple processors, which reduces the cost, consumption and complexity of the system.

*uCLinux* is an operating system derivative of Linux, intended for processors without Memory Management Units. It supports the *Nios II* processor with a well-developed collection of user applications, libraries and tool chains, all of which make this operating system a good choice for embedded systems. Different embedded systems have chosen this operating system because of its performance, efficiency and support for diverse hardware devices in soft-core processors [[[20]](#endnote-21), [[21]](#endnote-22)].

Cyclone II edition series CPU have been supported by *uCLinux* so we only need to modify or add transplantation files under linux/arch/Nios2nommu/scripts, which are mainly associated with the specific configuration of the hardware platform. In addition to this, we can use the *Nios II* SOPC Builder software that comes with a corresponding generating platform-specific configuration file. The process of transplantation is relatively simple. The main consideration with the hardware development platform is to modify specific factors, such as memory size, configuration and other peripheral devices [[[22]](#endnote-23)].

### *Network Model*

Communications between computers on a network is done through protocol stack. The most widely used and most widely available protocol suite is the TCP/IP protocol stack. A protocol stack consists of layered architecture where each layer depicts some functionality which can be carried out by a protocol. Each layer usually has more than one protocol option to carry out the responsibility that the layer adheres to. TCP/IP is generally considered to be a 4 layer system. [[[23]](#endnote-24), [[24]](#endnote-25), [[25]](#endnote-26), [[26]](#endnote-27)]

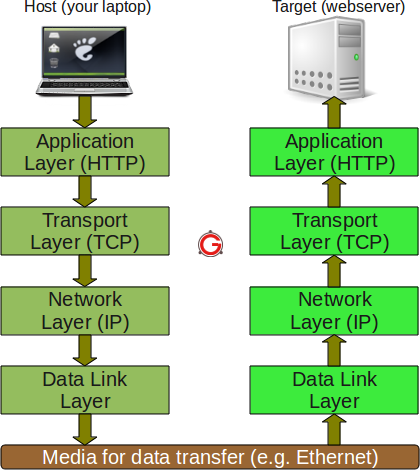


Figure 11 Network layer model.

Figure 11 shows the network layer model.

1. Application layer

This is the top layer of the TCP/IP protocol suite. This layer includes applications or processes that use transport layer protocols to deliver the data to destination computers. In our case, the application layer is the DTP protocol and it will be introduced in the next chapter.

1. Transport layer

This layer provides backbone to data flow between two hosts. The layer receives data from the application layer above it. There are many protocols that work at this layer but the two most commonly used are TCP and UDP. TCP is used where a reliable connection is required, while UDP is used in case of an unreliable connection. For its reliability, we chose TCP as the transport layer.

1. Network layer

This layer is also known as the *Internet layer*. The main purpose of this layer is to organize or handle the movement of data on a network. By ‘movement of data’, we generally mean the routing of data over the network. The main protocol used at this layer is IP. ICMP (used by popular ‘ping’ commands) and IGMP are also used at this layer.

1. Data link layer

This layer is also known as the *network interface layer*. This layer normally consists of device drivers in the OS and a network interface card attached to the system. Both the device drivers and the network interface card take care of the communication details, with the media being used to transfer the data over the network. In most cases, this media is in the form of cables. Some of the well-known protocols that are used at this layer include ARP(Address resolution protocol) and PPP(Point to point protocol).

### *Application layer protocol*

To simplify development work, the communication between host and detector is divided into two logical channels which are implemented as two socket connections; a command channel and an image channel. The logical view is shown in Figure 12.

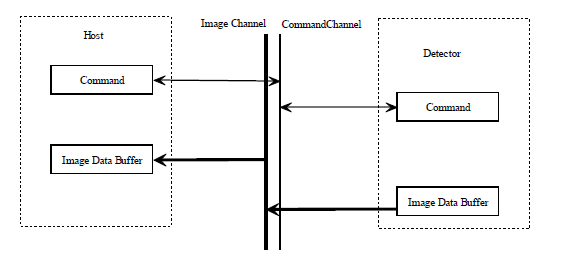


Figure 12 Communication channel logical view.

The reason why we have chosen to use a two-channel design is that the command channel and image channel have different behavior when communicating with a host PC. The command channel’s communication is dual directional. In a typical case, the host sends a command to the detector, after which the command is processed, and the detector sends the feedback to the host PC. Conversely, the image channel’s communication is single directional; the data always runs from the detector to the host PC. To support different pixel format, we will use the DTP protocol to transfer the image data, while the command channel protocol is very simple for debugging purposes.

1. Command Channel Protocol

The command channel is used to transfer control commands to the detector. For the command format, all the commands are in ASCII format and obey the following rules:-

* All the command data is ASCII format. It is easier for development and debugging, as compared to the binary format.
* The command is the capital letter from A to Z in ASCII format. It defines the target register of the detector. For example, the keyword ‘ST’ refers to the integration time register.
* The operation code defines what kind of operation is applied for the register, as defined by the command. There are two operation codes:
  + “W”: write to the register.
  + “R”: read from register.
* The first character of the command body is “[“, which is defined as STX. The last character of the command body is “]”, which is defined as ETX.
* The general format of the command protocol is as following:

STX + command+”,” + operation code + ”,” + parameter in Hex + ETX

Let us examine a theoretical case. If we want to set the integration time of the detector to five hundred microseconds, we would send the string “[ST,W,1F4]” to the detector. ”[“ is the STX of the command and “]” is the ETX. “ST” is the command key word of the integration command, and “W” means set operation. The parameter 1F4 is the value 500 in hex format.

1. Image channel protocol

The image channel is used to receive the image data or to flag error data, which is transferred from the detector to the host in the DTP protocol.

The organization of the DTP protocol is shown in Figure 13:

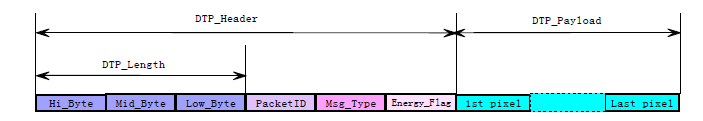


Figure 13 DTP packet format.

* 1st Byte: High byte of DTP packet length, including the header.
* 2nd Byte: Mid byte of DTP packet length, including the header.
* 3rd Byte: Low byte of DTP packet length, including the header.
* 4th Byte: Image packet ID.

If there is no image data line missing, the ID number should be continued. For example, if the current ID is 211, then the next line ID is 212. If the current line ID is 255, then the next line ID should be 0. After the detector gets the start sending frame command ―[SF, 1], the ID number will reset to 0.

* 5th Byte: Message type of the DTP packet.

Currently we just define one type of message:

* FRAME\_IMAGE\_DATA: 0x00

It indicates that the message body is frame data,

* 6th Byte: Reserved for future.

The DTP protocol meets the requirement of flexibility, reliability and efficiency. It is very flexible for a different pixel number detector as it defines the three byte value of DTP length. It also reserves two bytes for future extensions. For example, we have the option of defining more packet type to support dual energy detectors in the future. The image packet ID mechanism ensures that when there are packets missing, the host PC will know that this has occurred. Finally, the total header length is just 6 bytes for one transfer, which normally contains thousands of bytes of payload data.

### *Socket programming model*

A socket is an end point of communication between two systems on a network. To be more precise, a socket is a combination of an IP address and a port on one system. On every system a socket exists for the process of interacting with socket on other systems over a network. In our case, the detector acts as the socket server and the host PC acts as the client [26].

Figure 14 shows the interaction between client and server.

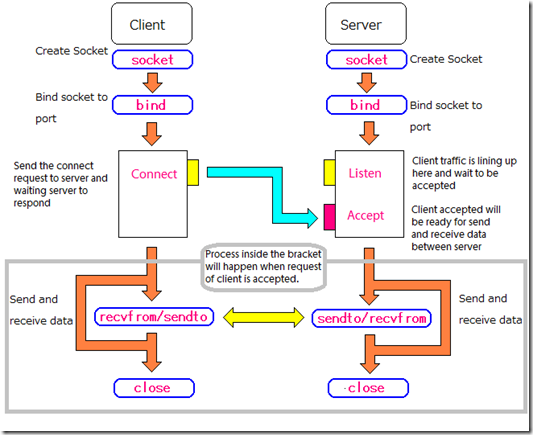
[](http://lh4.ggpht.com/-SMrgQcpKqUk/UYALoswxqrI/AAAAAAAAArU/WtVTD_Wd76s/s1600-h/network4%5b6%5d.png)

Figure 14 Socket Model

For the detector, we need to create a socket server by employing the following process:

1. Call the function ‘socket’ to create an un-named socket inside the kernel and return an integer known as a *socket descriptor*.
2. Call the function ‘bind’ to assign the details specified in the structure ‘serv\_addr’ of the socket created in the step above. The details include the family/domain, and the port on which the server will wait for the client requests to come.
3. Call the function ‘listen’ with the second argument, as the maximum number of client connections to the server will queue for this listening socket.
4. After the call to ‘listen’, this socket becomes a fully functional listening socket.
5. The call to ‘accept’ is run in an infinite loop so that the server is always running and the delay or sleep of one second ensures that this server does not eat up all of the CPU processing.
6. As soon as the server gets a request from the client, it prepares the date and time and writes on the client socket through the descriptor returned by “accept”.

## Firmware Implementation

Originally, firmware meant the contents of a writable [control store](http://en.wikipedia.org/wiki/Control_store) (a small specialized high speed memory), containing [microcode](http://en.wikipedia.org/wiki/Microcode) that defined and implemented the computer's [instruction set](http://en.wikipedia.org/wiki/Instruction_set), and that could be reloaded to specialize or modify the instructions that the [central processing unit](http://en.wikipedia.org/wiki/Central_processing_unit) (CPU) could execute [[[27]](#endnote-28)]. As originally used, firmware contrasted with hardware and software. It was not composed of CPU machine instructions, but of lower-level microcode involved in the implementation of machine instructions. It existed on the boundary between hardware and software; thus the name ‘firmware’. Still later, popular usage extended the word ‘firmware’ to denote anything ROM-resident, including processor machine-instructions for [BIOS](http://en.wikipedia.org/wiki/BIOS), [bootstrap loaders](http://en.wikipedia.org/wiki/Booting), or specialized applications. In our case, the firmware refers to the software running inside the *Nios II* processor.

### *Setup Cross-compiler Development Environment*

Generally speaking, a cross-compiler is a compiler that runs on platform A (the host), but generates executables for platform B (the target) [[[28]](#endnote-29)]. For us, we run the *Nios II* compiler from one desktop Linux environment to generate the executable code for *Nios II* processor. To establish a cross-compiler environment for *uCLinux*, we need to perform the following steps:

1. Download the official uClinux cross-compiler to support *Nios II* Development Kit, including Nios2-linux-uclibc-gcc, Nios2-linux-uclibc-ld, Nios2-linux-uclibc-objdump and so on.
2. Install the cross-compiler kit to Linux PC.
3. Setup the PATH for the cross gcc as “PATH=$PATH:/opt/nios2/bin”, you can add a line at the end of file ~/.bash\_profile (or ~/.profile on Debian/Ubuntu)
4. Logout and login again
5. To verify the cross gcc, try out, nios2-linux-uclibc-gcc -v

It should display the information:

Reading specs from /opt/nios2/lib/gcc/nios2-linux-uclibc/3.4.6/specs

Configured with: /root/buildroot/toolchain\_build\_nios2/gcc-3.4.6/configure --prefix=/opt/nios2 --build=i386-pc-linux-gnu --host=i386-pc-linux-gnu --target=nios2-linux-uclibc --enable-languages=c --enable-shared --disable-\_\_cxa\_atexit --enable-target-optspace --with-gnu-ld --disable-nls --enable-threads --disable-multilib --enable-cxx-flags=-static

Thread model: posix

gcc version 3.4.6

After the setup of the cross compiler environment, we can develop the firmware in the same way that we would a standard Linux program. Appendix 1 is the Make file we used for the firmware.

### *Firmware flow*

The main functions of the firmware are:

* To initialize system parameters;
* To setup the socket listening port;
* To parse, execute and respond to commands from the host computer;
* To send raw image data to the host computer.

Figure 15 is a flow chart of the firmware. It shows the process of how firmware works.

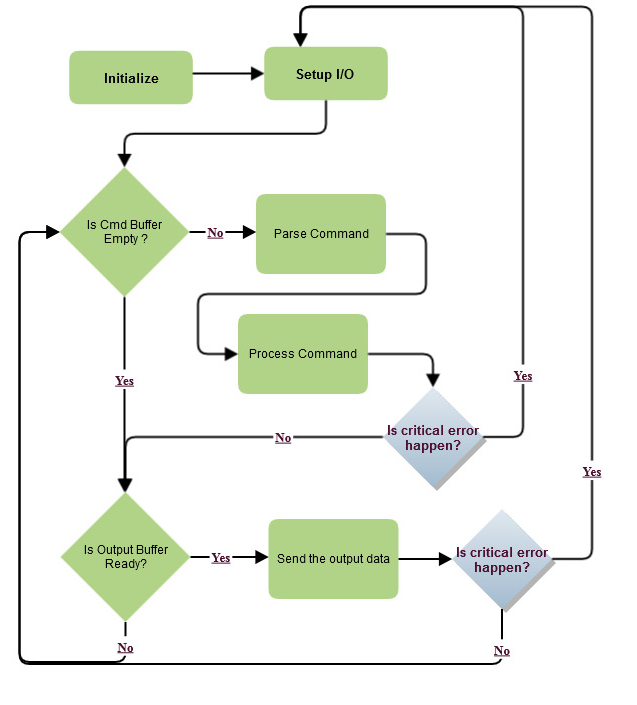


Figure 15 The firmware flow chart.

1. Detector Initialization

After the detector is turned on, FPGA loads the uClinux image file from flash into the *Nios II* memory, and starts up uClinux. Then uClinux initiated scripts will execute our firmware, and the firmware will start to initialize the detector with predefined parameters such as integration time and overall gain.

1. Setup I/O

After detector initialization, the firmware will set up the socket connection for communication. The firmware will then enter an endless loop to await the incoming socket connection request. When the socket connection request arrives, the firmware will accept the request and put the connection into a connection list, before checking the connections of the list in the endless loop. The socket programming details will be introduced in the next chapter.

1. Communication with host PC

If the output data from the detector board is ready to send, FPGA will set up a flag to indicate this fact. If the output mode is network, firmware will call socket lib to send the output data.

1. Critical error handler

If a critical error occurs when trying to send or receive data, firmware will need to reset the I/O interface. For example, if the network cable plug is out, the TCP connection will be interrupted. The detector must be in listening mode to accept the connection. Otherwise, the host PC cannot reconnect to the detector.

## Discussion

The target of this thesis is to design a detector with low cost, acceptable performance, high flexibility, and excellent reliability. The control unit board is a key component of the whole system, so it must meet the above requirements. Based on the design and implementation of the control unit board in this chapter, we will now discuss how the design meets the requirement.

1. Lower Cost

Based on the *Nios II* processor solution, the low cost and high density Cyclone FPGA is employed in the design. Furthermore, because all the digital signal processing works are moved to the host computer side, the FPGA we chose for the project is much cheaper than other available options. The design removes the MCU chip and DSP chip from the control unit board. It also lessens the cost in the PCB area.

On the host PC side, the network card replaces the expensive commercial frame grabber board. Also, if we compare the programming difficulty of the former with the latter, we are clearly saving money in the area of software development.

1. Acceptable performance

The advantage of the frame grabber card is that it provides a much faster data transfer speed (at more than 40Mbytes/s), than the current Ethernet chip we chose, which only supports a maximum of 10 Mbytes/s for reasons of cost. However, if we take into consideration our current customer application for a maximum bandwidth request of about 5Mbytes, the performance of the latter is more than acceptable.

1. High Flexibility

The DTP protocol provides high flexibility on application layer, as discussed in Section 3.2.4. Furthermore, the network interface also allows for the possibility of wireless connection for some special applications. We need only to connect the detector with a wireless router, and it would become a wireless detector.

1. Excellent Reliability

We choose TCP rather than UDP for our transport layer protocol. TCP use [a sliding window mechanism](http://www.tcpipguide.com/free/t_TCPSlidingWindowDataTransferandAcknowledgementMech.htm) that controls the flow of data between devices. TCP not only manages the basic data transfer process, but also ensures that data is sent reliably, and manages the flow of data between devices to guarantee that data is transferred efficiently without either device sending it faster than the other can receive it.

From the previous discussion, we can see that the control-unit board design fully meets our target requirements. Also, and as we mentioned briefly in this chapter, the digital signal processing work is moved from FPGA to host PC side to lessen the overall cost. We will introduce how the digital signal can be processed in the next chapter.

# DIGITAL SIGNAL PROCESSING

## Overview

The original signals from detector cards are analog current signals which need to be translated into digital signals before they can be recognized by a computer. Furthermore, the raw digital signal contains systematic errors and random noise due to non-ideals in sensor channels. Digital signal processing, such as channel correction, noise filtering, and image enhancement, is required to improve the quality of the final images [[[29]](#endnote-30), [[30]](#endnote-31)].

Depending on the speed and computational requirements of the application, the digital signal processor may be realized by a general-purpose processor, FPGA, custom ASIC or special purpose DSP device and host PC. In the last chapter, we reached the conclusion that to decrease the cost of FPGA; we should move the digital signal process work to the host PC side. Nowadays, the CPU performance is fast enough to handle the work which used to be done by FPGA or other hardware.

The raw data flow diagram of the pixel processor is presented in Figure 16.

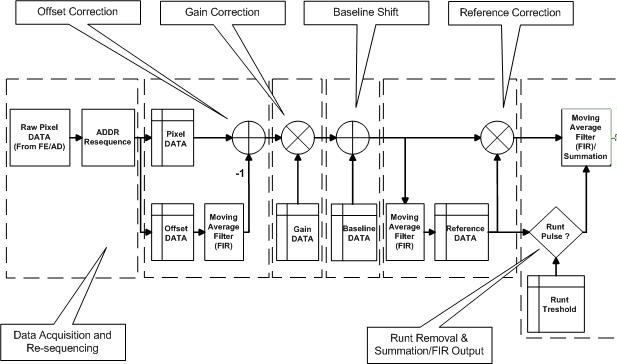


Figure 16 Data flow diagram of the pixel data processor.

The DSP data flow could be divided into six parts: they are data acquisition and re-sequencing, offset correction, gain correction, baseline shift and summation/FIR output.

The following symbols are used.

* Pxx[n,p] Pixel signal data, line number n, pixel value p
* Pxx[n] Pixel signal data , line number n (all pixel values)
* Oxx[n] Offset data, line number n

## Re-arrange Pixel Order

Raw data from the detector modules is not in the correct order for an image. The first step of digital image processing is to reorder the raw data, depending on the reading sequence of the reading circuits. Until then, we can get an image visible to human eyes, and we are able to apply further image processing in the spatial domain.

## Systematic error correction

Systematic error, or fixed pattern noise (FPN), refers to non-temporal spatial noise. [[[31]](#endnote-32)]FPN is constant when the read out conditions and the integration time are not changed. Thus, it is possible to suppress the system error from the measured signal by calibration.

FPN is mostly caused by offset and gain mismatch between sensor channels, including variation of dark current and gain of photo detectors, offset and gain variation in channel amplifiers and ADCs. Internal switching during reading out might also introduce some FPN.

This kind of error can be non-coherent or coherent. Non-coherent noise does not change with signal strength, and is dominated by the mismatch of the leakage currents among photodiodes, especially with long integration time. The non-coherent offset error can be eliminated by dark frame subtraction. Coherent offset components can generally be suppressed by reference frame subtraction.

Gain errors come from scintillator crystal material variations, crystal-to-photodiode coupling variations, channel gain mismatch in amplifiers and ADCs. These errors are nonlinear. They are the most problematic FPN to correct.

The fixed pattern noise in the X-ray line-scan detector can be and must be removed; otherwise they could be wrongly identified as valid objects in a scanned image by computer software or human eyes. The figure 17(a) shows the ideal response when a detector receives X-ray radiation, and figure 17(b) shows the same response when offset and gain errors are applied.

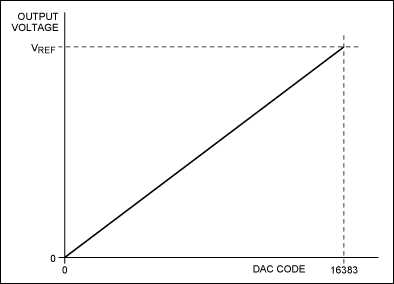


Figure 17(a) Ideal response for pixel.

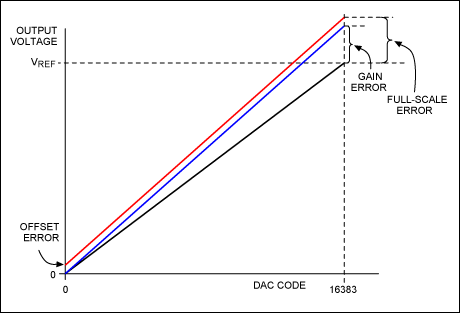


Figure 17(b) Offset and gain error.

### *Offset correction by dark frame subtraction*

The dark frame subtraction method assumes the FPN is constant and equal to the value of the dark frame. The FPN can then be subtracted from subsequent captured images. In our implementation, the moving average of the offset value of each pixel is used for the offset correction of that individual channel (Oc[n]).

The offset corrected value is calculated as follows:



where, M is the size of the moving average filter, n refers to the line number of the image data, and p is the pixel index of the detector.

The offset corrected pixel value will therefore be:



where, Pr is the raw data from the detector.

### *Gain calibration and correction*

Gain variations should be corrected by calibration using the following steps:

1. Take an offset scan with radiation off to get the pixel offset value *P0[p]*.
2. Take an air scan with radiation on, without the object, so we can get the pixel value *P1[p].*
3. The gain of each pixel is calculated as *G0[p] = P1[p] – P0[p].*
4. Using the average gain of the complete image line as the reference gain, gain correction factors for the each pixel are calculated as Gc[p] = G0[p]/AVG(G0(p)).
5. Store the correction values.

So for the gain correction output, we have:

PC2[n,p] = PC1[n,p]\*G[p]

where , p = 0… 1023 (for 1024 channels).G[p]is the correction factor.

### *Baseline shift*

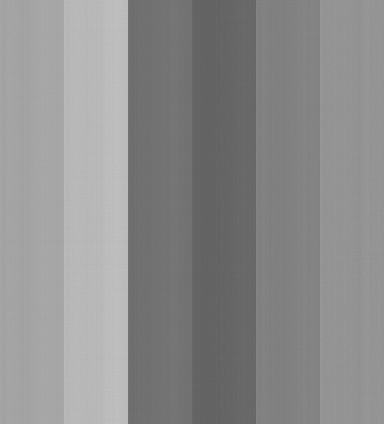
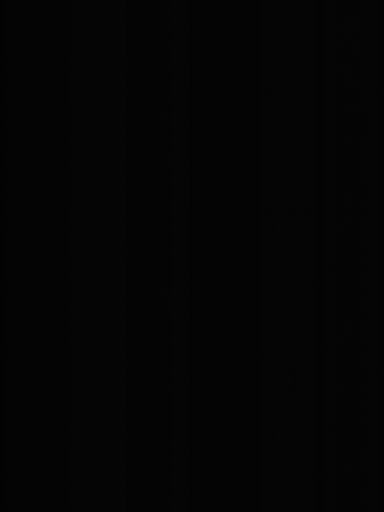
The calibrated pixel data without X-ray would drop to around zero in the digital system after the processing block. The zero data would be outputted when the calibrated pixel data is less than zero, so the baseline shift is needed when the calibrated pixel data without X-ray has to be analyzed in the software. The baseline value could be specified by the user employing the special register in FPGA. Then the shifted line of pixel data is:

PC3[n,p] = PC2[n,p] + B

where, B is a constant for baseline shift, but it can be changed by the user.

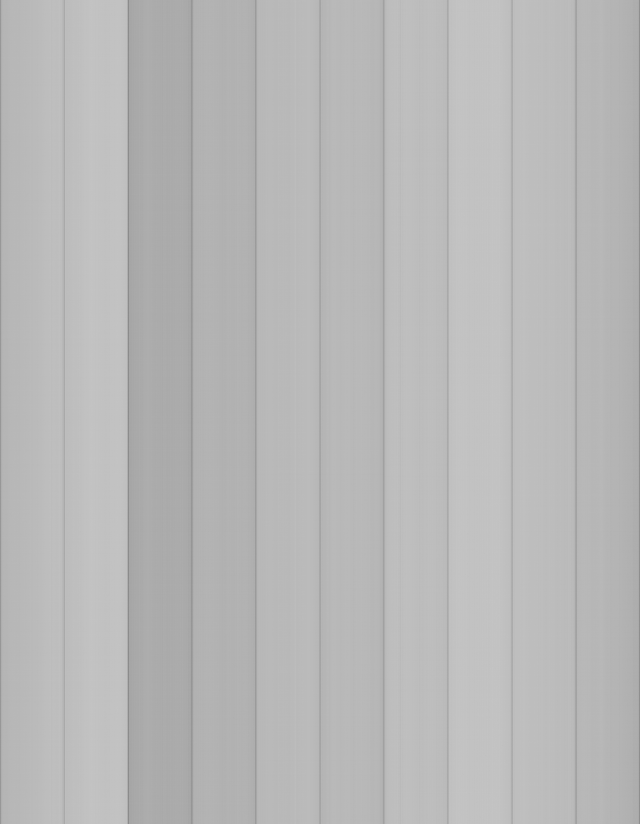
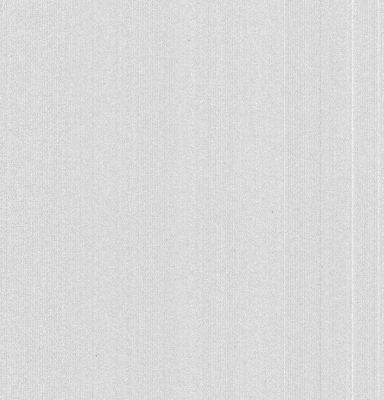
### *Correction result*

Figure 18 shows the result of offset correction and Figure 19 shows the result of gain correction. From the pictures it can be seen that the stripes of the uncorrected background are completely removed by calibration. All the non-uniformities caused by the sensors are now corrected and invisible to human eyes.

1. (b)

Figure 18 Offset calibration (a) a blank image without offset calibration (b) a blank image with offset calibration.

1. (b)

Figure 19 Gain calibration (a) a blank image without gain calibration (b) a blank image with gain calibration.

## Image Denoising

Image denoising is an important image processing task, both as a process in itself, and as a component in other processes. There are very many ways to denoise an image or a set of data. The main properties of a good image denoising model are that it will remove noise while preserving edges.

Consider an image contaminated by noise:

*I(x) = f(x) + n(x)*

where, *n(x)* is the undesired noise component of the signal. Our goal is to remove that noise, resulting in minimal damage to the image.

A normal way to extract *f(x)* from *I(x)* is to design an appropriate filter *H(z),* which removes the noise component *n(x)* and at the same time lets the desired signal *f(x)* go through unchanged. This process can be expressed as:

*y(x) = yf(x) + yn(x)*

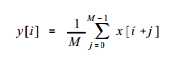
where *yf(x)* is the output due to *f(x)* and *yn(x)* is the output due to *n(x)*.

### *White noise*

In [signal processing](http://en.wikipedia.org/wiki/Signal_processing), white noise is a [discrete signal](http://en.wikipedia.org/wiki/Discrete_signal) whose [samples](http://en.wikipedia.org/wiki/Sample_(signal)) are regarded as a sequence of [serially uncorrelated](http://en.wikipedia.org/wiki/Serial_correlation) [random variables](http://en.wikipedia.org/wiki/Random_variable) with zero [mean](http://en.wikipedia.org/wiki/Mean_(statistics)) and finite [variance](http://en.wikipedia.org/wiki/Variance). Depending on the context, one may also require that the samples be [independent](http://en.wikipedia.org/wiki/Statistical_independence) and have the same [probability distribution](http://en.wikipedia.org/wiki/Probability_distribution). In particular, if each sample has a [normal distribution](http://en.wikipedia.org/wiki/Normal_distribution) with zero mean, the signal is said to be Gaussian white noise [[[32]](#endnote-33)].

### *Moving Average Filter*

The simple moving average filter averages recent values of the filter input for a given number of inputs. This is the most common example of the ‘[moving average](http://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Filtering/Filter-Memory/filter-memory.htm#ARMA)’ (MA) category of filters, also called [finite impulse response](http://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Filtering/Filter-Memory/filter-memory.htm#IIR) (FIR) filters. Each recent input is multiplied by a coefficient for all linear moving average filters, and the coefficients are all the same for this simple moving average. The sum of the coefficients is 1.0, so that the output eventually matches the input when the input does not change. Its output merely depends on recent inputs, unlike the exponential filter that also reuses its previous output. The only parameter is the number of points in the average - the ‘window size’ [[[33]](#endnote-34)].

****

The above is the equation for the moving average filter. In this equation, x[ ] is the input signal, y[ ] is the output signal, and M is the number of points used in the moving average. This equation only uses points on one side of the output sample being calculated. It reduces the standard deviation by M.

# HOST PC SOFTWARE DESIGN AND IMPLEMENTATION

In Chapter 4, we discussed the digital signal processing algorithm used by PC software. This chapter describes ApponsView, the host PC software architecture design, and the manner in which it can be implemented so as to fulfill the detector’s functions. Section 5.1 states the general requirements of ApponsView.

## General Requirement

*ApponsView* shall run with in Microsoft Windows XP/Vista/Win7 environment, and it shall contain a DLL (dynamic link library) for the customer to perform development work. *ApponsView* should be a real-time image processing system with detector control interface. It should be run on a standard personal computer equipped with network card or mainboard integrated network interface. All the signal process work is done by personal computer, so dedicated hardware such as a frame grabber card is not necessary.

A variety of digital signal process techniques shall be utilized in order to provide a high quality object scanning image. The process time should satisfy the real-time restriction, and the customer can design their own process module to meet their specific requirements.

Another important requirement is controlling the detector. *ApponsView* should provide a graphic user interface for the user to control the detector, and show the status of the detector. For example, when data is lost or the connection is interrupted, *ApponsView* should notify the user that something is wrong.

In summary, *ApponsView* should support the following features:

1. Detector controlling

Commands can be sent to the detector through the command channel, such as speed, gain, and offset. To control the detector, the user can send raw commands in ASCII format to the detector. A high level wrapper of the command is also necessary.

1. Receive image data from detector

The image data can be taken from the detector though the network connection. It should provide interface to access the frame buffer which receives the image data from the detector. The user can define frame buffer size, and be notified when one frame is ready.

1. Basic processing and analysis of the image

Implementing the digital filter described in the previous chapter and mapping 16- bit data to 8-bit data for display, *ApponsView* can also save the original image data in an 8-bit or 16-bit format. *ApponsView* should, in addition, provide basic image manipulators such as brightness and contrast adjustment functions for the image.

1. User interaction

*ApponsView* shall read user-definable preference information from the configuration file and adapt itself to those settings during initialization. It shall also initialize the detector by predefined parameters which are stored in the configuration file.

File save function should also be supported by *ApponsView*. The user can define the size of the file.

## Component Architecture

.

At the top level, the entire host PC software is divided into library and application. *ApponsControl* is a dynamic link libraray (DLL), which is a single module that can be loaded into memory on demand. All the necessory functions such as detector control and image data receiving from the detector and digitial filter are packed in this DLL. The *ApponsView* application is an executable program which provides user interface for *ApponsControl* library.

The *ApponsControl* library is packed in ActiveX control format. ActiveX control is a [control](http://www.webopedia.com/TERM/C/control.html) that ultilizes Microsoft [ActiveX](http://www.webopedia.com/TERM/A/ActiveX.html) technologies. An ActiveX control can be automatically [downloaded](http://www.webopedia.com/TERM/D/download.html) and [executed](http://www.webopedia.com/TERM/E/execute.html) by a [Web browser](http://www.webopedia.com/TERM/B/browser.html). ActiveX is not a [programming language](http://www.webopedia.com/TERM/P/programming_language.html), but rather a set of rules for how applications should share information. Programmers can develop ActiveX controls in a variety of languages, including [C](http://www.webopedia.com/TERM/C/C.html), [C++](http://www.webopedia.com/TERM/C/C_plus_plus.html), Visual Basic, and [Java](http://www.webopedia.com/TERM/J/Java.html). Then the customer can choose the tools suitable for them to do the development work. For simple applications that merely show a scanned image without advance image process work, the customer can use Visual Basic to accelerate the development. If the customer requires image data processing that need highly efficient memory access, they can choose C++ as development tools [[[34]](#endnote-35)].

*ApponsView* is developed on Qt platform. Qt is the leading C++ based cross-platform UI and software application development framework, and is widely used for developing [application software](http://en.wikipedia.org/wiki/Application_software) with a [graphical user interface (GUI)](http://en.wikipedia.org/wiki/Graphical_user_interface) (in which cases Qt is classified as a [widget toolkit](http://en.wikipedia.org/wiki/Widget_toolkit)). Using Qt also gives the possiblity of porting the application to the Linux platform.

### *ApponsControl Library Architecture*

The component architecture of the *ApponsControl* library is shown in Figure 20

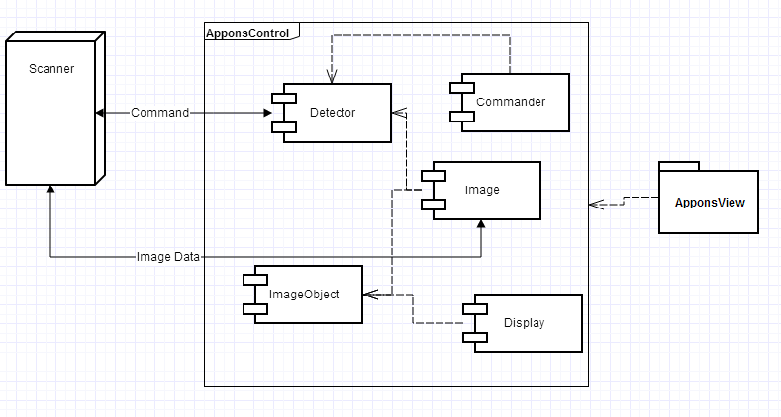


Figure 20 Component View of *ApponsControl*

The *ApponsControl* library includes several ActiveX control components. They are detector control, commander control, image control, and display control. The controls cooperate to help fulfill the detector’s functions.

Detector control is responsible for command channel communication, including sending commands and receiving command feedback from the scanner. It provides a string command interface for the user to send commands. Commander control is a wrapper interface of detector control. In other words, it provides a more user-friendly function to perform commands by function calling. For most customers, the commander control is enough for them to control the detector. However, more advanced users may prefer to use the detector directly if they are familiar with the detector command list. There are also some commands which are not available in an API document, and the only way to access these functions is to use detector control to send the ASCII command string.

Image control is responsible for the image channel operation, including initializing the frame buffer and triggering frame ready events for application. Image control uses detector control to send frame commands to the detector. ImageObject control is a frame buffer tool class which provides quick accessibility to the frame buffer. Image control employs ImageObject to manage the internal frame buffer. The display control is used to show the image data in the frame buffer. The original image data is 16-bit, so the display control needs to convert it to 8-bit in accordance with the user’s settings.

### *ApponsView Application Architecture*

The component architecture of the *ApponsView* application can be seen in Figure 21.

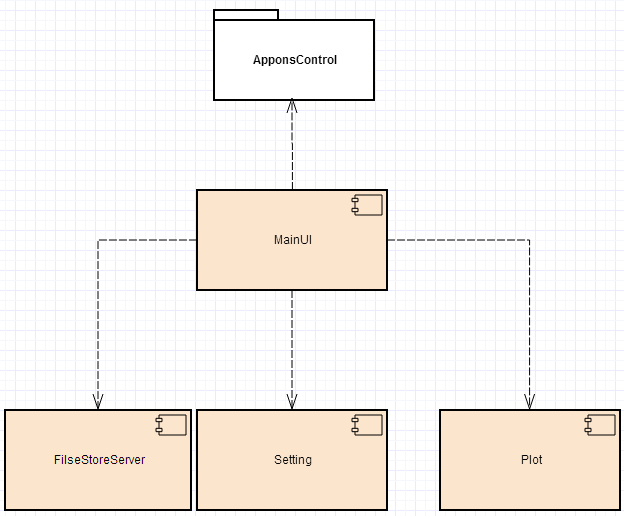


Figure 21 Component View of *ApponsControl.*

The MainUI module is responsible for the user interaction and the main thread of application. There are several support modules to help MainUI perform the work. When MainUI is starting, it calls the Setting module to load the pre-defined configuration of the detector and *ApponsView* itself. As described in the previous section, *ApponsControl* library is used for communication between *ApponsView* and the detector. FileStoreServer is used to save the image data coming from the detector. Plot module is responsible for the image data statistical profile plot drawing.

## ApponsView Implementation

Based on the architecture design, the development work is also divided into two parts: *ApponsControl* library and *ApponsView* application. To meet the real-time requirements, we chose C++ as the development language, but we used different tools to develop the *ApponsView* and *ApponsControl* library for the different requirements.

### *ApponsControl Library Implementation*

According to the customer requirement, the *ApponsControl* library should support ActiveX control standards, so we chose Visual C++ 2010 as the tool to develop *ApponsControl* library. The figure 22 is the class diagram of the *ApponsControl* library.

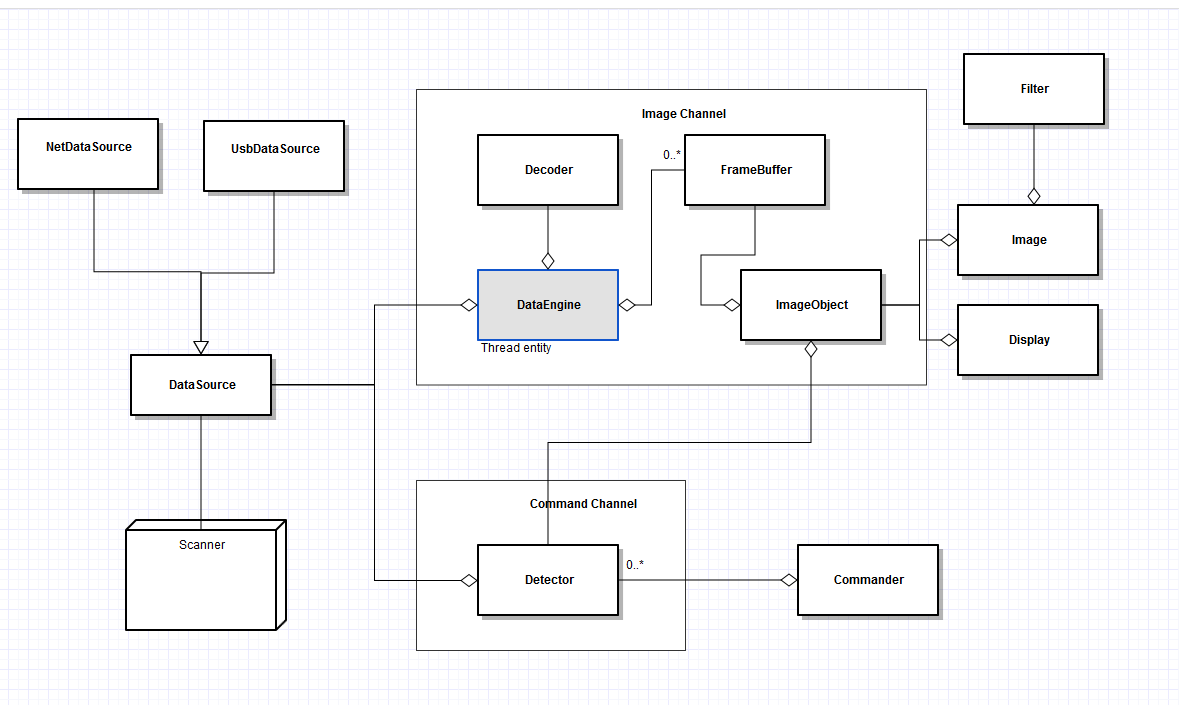


Figure 22 Class View of *ApponsControl*

* DataSource class

DataSource class is an abstract interface of the data stream I/O operation. It isolates the future change of the low level I/O. Thus, in future, if we need to change another I/O library or support a new interface, we only need to implement a new DataSource class.

* DataEngine class

Data Engine is a thread entity class. It owns a thread which reads the data from the data source, and calls the Decoder to decode the scanner data into raw image data.

* Decoder class

Decoder is responsible for decoding DTP packet into raw image data, and putting the raw image data into a frame buffer. It also checks the packet sequence number to make sure that there is no packet lost.

* FrameBuffer and ImageObject

FrameBuffer is a memory block which stores the image data, and ImageObject is a wrapper class which provides a directly accessible interface to it. ImageObject pre-calculates all the row addresses of the FrameBuffer and stores them in xtable. Thus, when there is an access requirement such as img[x,y], there is no need to do the multiple operation x\*width+y. It merely requires a one-time add operation xtable[x]+y.

* Filter

Filter is the digital process component described in Chapter 4. The figure 23 show the class diagram of the filter design.

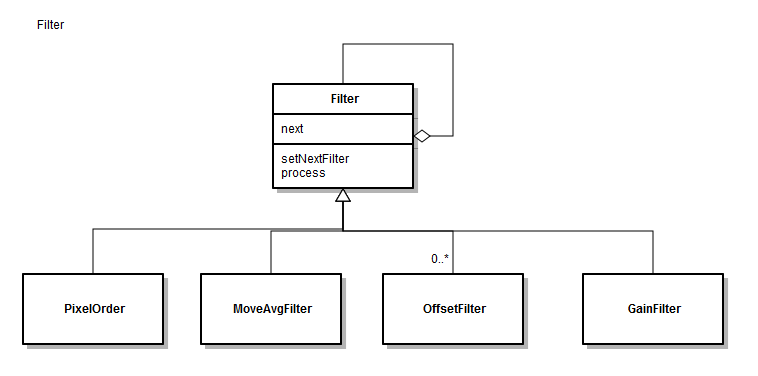


Figure 23 Class View of Filter

There are currently four kinds of filter:

1. PixelOrder Filter

The PixelOrder filter is responsible for adjusting the pixel order or assigning each pixel to the correct order. The original order is mixed between the left and right channels of the control unit board. which described in section 4.2

1. MoveAvg Filter

The MoveAvg filter implements the FIR moving Average Filter described in section 4.4.2

1. Offset Filter

The Offset filter is responsible for the offset correction described in Section 4.3.1

1. Gain Filter

The Gain filter is responsible for the gain correction described in Section 4.3.2

The filters are organized in a chain mode. When one filter finished the process work, it will transfer the processed data to next filter. This kind of design provides the flexible combination of the filter to finish different job.

### *ApponsView Application Implementation*

Due to the fact that in the future *ApponsView* is needed to be ported on a Linux platform, we chose Qt as the development platform. Qt uses standard [C++](http://en.wikipedia.org/wiki/C%2B%2B) but makes extensive use of a special [code generator](http://en.wikipedia.org/wiki/Code_generator) (called the Meta Object Compiler, or MOC), together with several macros to enrich the language. It runs on the major desktop platforms and some of the mobile platforms. It also has extensive [internationalization](http://en.wikipedia.org/wiki/Internationalization_and_localization) support [[[35]](#endnote-36)].

*ApponsView* uses *ApponsControl* components to do the work related to the detector, such as command communication and image transfers. *ApponsView* creates the components and connects them with pre-defined parameters. It also employs a Qt graphic view framework for image display and manipulation. Figure 24 shows the class relationship in the *ApponsView* application

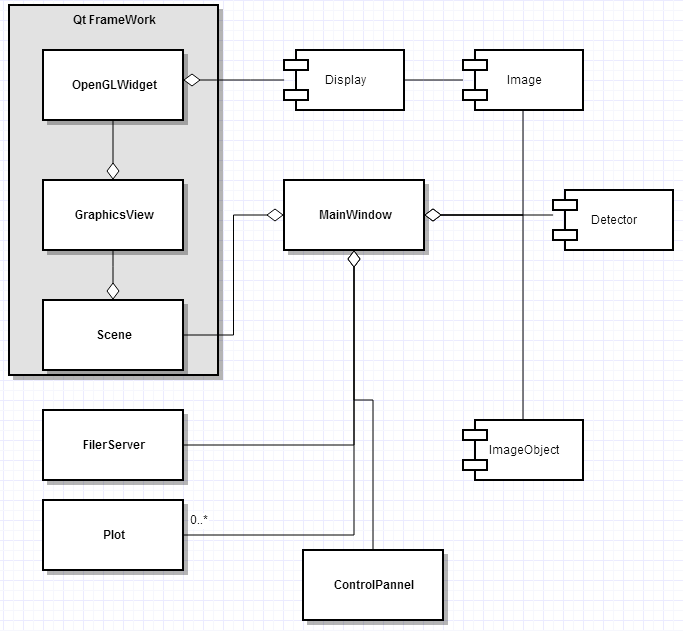


Figure 15 Class View of *ApponsView*

## Multi Core Optimization

A multi-core processor is a single [computing](http://en.wikipedia.org/wiki/Computing) component with two or more independent actual [central processing units](http://en.wikipedia.org/wiki/Central_processing_unit) (called ‘cores’), which are the units that read and execute [program instructions](http://en.wikipedia.org/wiki/Instruction_(computer_science)) [[[36]](#endnote-37)]. The instructions are ordinary [CPU instructions](http://en.wikipedia.org/wiki/Instruction_set) such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to [parallel computing](http://en.wikipedia.org/wiki/Parallel_computing). Manufacturers typically integrate the cores onto a single [integrated circuit](http://en.wikipedia.org/wiki/Integrated_circuit) [die](http://en.wikipedia.org/wiki/Die_(integrated_circuit)) (known as a chip multiprocessor, or CMP), or onto multiple dies in a single [chip package](http://en.wikipedia.org/wiki/Chip_carrier).

Processors were originally manufactured with only one core. Multi-core processors were developed in the early 2000s by [Intel](http://en.wikipedia.org/wiki/Intel), [AMD](http://en.wikipedia.org/wiki/AMD) and others. Multicore processors may have two cores (dual-core CPUs, for example [AMD Phenom II X2](http://en.wikipedia.org/wiki/List_of_AMD_Phenom_microprocessors#.22Callisto.22_.28C2.2FC3.2C_45_nm.2C_Dual-core.29) and [Intel Core Duo](http://en.wikipedia.org/wiki/Intel_Core_Duo)), four cores (quad-core CPUs, for example [AMD Phenom II X4](http://en.wikipedia.org/wiki/List_of_AMD_Phenom_microprocessors#.22Zosma.22_.28E0.2C_45_nm.2C_Quad-core.29), Intel's [i5](http://en.wikipedia.org/wiki/Intel_Core_i5) and [i7](http://en.wikipedia.org/wiki/Intel_Core_i7) processors), six cores (hexa-core CPUs, for example [AMD Phenom II X6](http://en.wikipedia.org/wiki/List_of_AMD_Phenom_microprocessors#.22Thuban.22_.28E0.2C_45_nm.2C_Hexa-core.29) and [Intel Core i7 Extreme Edition 980X](http://en.wikipedia.org/wiki/Gulftown)), eight cores (octo-core CPUs, for example [Intel Xeon E7-2820](http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors#.22Westmere-EX.22_.2832_nm.29) and [AMD FX-8350](http://en.wikipedia.org/wiki/List_of_AMD_FX_microprocessors)), ten cores (for example, [Intel Xeon E7-2850](http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors#.22Westmere-EX.22_.2832_nm.29_Expandable)), or more.

A multi-core processor implements [multiprocessing](http://en.wikipedia.org/wiki/Multiprocessing) work in a single physical package. Designers may couple cores in a multi-core device tightly or loosely. For example, cores may or may not share [caches](http://en.wikipedia.org/wiki/CPU_cache), and they may implement [message passing](http://en.wikipedia.org/wiki/Message_passing) or [shared memory](http://en.wikipedia.org/wiki/Shared_memory) inter-core communication methods. Common [network topologies](http://en.wikipedia.org/wiki/Network_topology) to interconnect cores include [bus](http://en.wikipedia.org/wiki/Bus_topology), ring, two-dimensional mesh, and [crossbar](http://en.wikipedia.org/wiki/Crossbar_switch). [Homogeneous](http://en.wikipedia.org/w/index.php?title=Homogeneous_computing&action=edit&redlink=1) multi-core systems include only identical cores; [heterogeneous](http://en.wikipedia.org/wiki/Heterogeneous_computing) multi-core systems have cores that are not identical. Just as with single-processor systems, cores in multi-core systems may implement architectures such as [superscalar](http://en.wikipedia.org/wiki/Superscalar), [VLIW](http://en.wikipedia.org/wiki/VLIW), [vector processing](http://en.wikipedia.org/wiki/Vector_processor), [SIMD](http://en.wikipedia.org/wiki/SIMD), or [multithreading](http://en.wikipedia.org/wiki/Multithreading_(computer_hardware)). Multi-core processors are widely used across many application domains, including general-purpose, [embedded](http://en.wikipedia.org/wiki/Embedded_processor), [network](http://en.wikipedia.org/wiki/Network_processor), [digital signal processing](http://en.wikipedia.org/wiki/Digital_signal_processing) (DSP), and [graphics](http://en.wikipedia.org/wiki/Graphics_processing_unit).

The improvement in performance gained by the use of a multi-core processor depends very much on the software algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can be [run in parallel](http://en.wikipedia.org/wiki/Parallel_processing) simultaneously on multiple cores; this effect is described by [Amdahl's law](http://en.wikipedia.org/wiki/Amdahl%27s_law). In the best case, so-called [embarrassingly parallel](http://en.wikipedia.org/wiki/Embarrassingly_parallel) problems may realize speedup factors near the number of cores, or even more if the problem is split up enough to fit within each core's cache(s), avoiding use of much slower main system memory. Most applications, however, are not accelerated quite so much, unless programmers invest a prohibitive amount of effort in re-factoring the whole problem [[[37]](#endnote-38)]. The parallelization of software is a significant ongoing topic of research.

In our case, *ApponsView* employs three threads to finish the real-time work. The first of these is I/O thread, which is responsible for receiving data from the detector. Due to the limitations of the detector buffer, only a hundred lines of data can be held at any one time. If the host PC cannot read the image data in time, the data will be lost. The second one is the DSP thread. The DSP thread continues running when there is data input from the detector, and it therefore must process the image data as quickly as possible. Finally, there is the main UI thread which responsible is for the interaction of the end user. All of the above threads need to have a very rapid response; otherwise, the system cannot legitimately be called a ‘real-time system. To achieve this target, we must utilize all of the multi-core’s resources through parallel programming.

### *Parallel Programming*

Parallel processing has always been an interesting method to improve program performance. Lately, there are more and more computers with multi processors or multi-core CPUs, thus making parallel processing available to the masses. However, merely having multiple processing units on the hardware does not alone imply that existing programs will take advantage of it.[[[38]](#endnote-39)] Programmers must take the initiative to implement parallel processing capabilities in their programs to fully utilize the hardware available. *OpenMP* is a set of programming APIs which includes several compiler directives and a library of support functions. It was first developed for use with FORTRAN and is now available for C and C++ respectively.as well.[[[39]](#endnote-40), [[40]](#endnote-41), [[41]](#endnote-42)]

Parallel processing can be divided into two groups: task based and data based.

1. Task based: Divides different tasks to different CPUs to be executed in parallel. For example, a data receiving thread and a DSP processing thread running simultaneously in *ApponsView*. Each thread is a separate task.
2. Data based: Executes the same task, but divides the work load of the data over several CPUs. For example, to convert a 16-bit image to grayscale, we can convert the top half of the image on the first CPU, while the lower half is converted on the second CPU (or as many CPUs as the user has), thus processing the whole thing in half the time.

There are several methods of doing parallel processing:

1. Using *OpenMP*: *OpenMP* is suitable for shared memory systems such as those we have on our desktop computers. Shared memory systems are systems with multiple processors, but each of them has a single memory subsystem. Using *OpenMP* is just like writing one’s own smaller threads but letting the compiler do it. Available in Visual Studio 2010 Professional and Team Suite.
2. Using *Cilk*: *Cilk* is a [general-purpose](http://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language](http://en.wikipedia.org/wiki/Programming_language) designed for [multi-threaded](http://en.wikipedia.org/wiki/Thread_(computer_science)) [parallel computing](http://en.wikipedia.org/wiki/Parallel_computing). The [C++](http://en.wikipedia.org/wiki/C%2B%2B)incarnation is called [*Cilk Plus*](http://en.wikipedia.org/wiki/Cilk_Plus).

The biggest principle behind the design of the *Cilk* language is that the programmer should be responsible for exposing the parallelism by identifying elements that can safely be executed in parallel. It should then be left to the run-time environment, particularly the [scheduler](http://en.wikipedia.org/wiki/Scheduling_(computing)), to decide during execution how to actually divide the work between processors. It is because these responsibilities are separated that a *Cilk* program can run without rewriting on any number of processors, including one [[[42]](#endnote-43)].

1. Using *SIMD* intrinsic: Single Instruction Multiple Data (SIMD) has been available on mainstream processors such as Intel's MMX, SSE, SSE2, SSE3, Motorola's (or IBM's) Altivec and AMD's 3DNow!. SIMD intrinsic are primitive functions to parallelize data processing on the CPU register level. For example, the addition of two unsigned char will take the whole register size, although the size of this data type is just 8-bit, leaving 24-bit in the register to be filled with zeroes and wasted. Using *SIMD* (such as MMX), we can load eight unsigned chars (or four shorts, or two integers) to be executed in parallel on the register level. Available in Visual Studio 2005 using SIMD intrinsics or with Visual C++ Processor Pack with Visual C++ 6.0 [[[43]](#endnote-44)].

Considering the programming platform and time limitations, we chose *OpenMP* as our parallel computing framework.

### *OpenMP*

*OpenMP* (Open Multi-Processing) is an [API](http://en.wikipedia.org/wiki/Application_programming_interface) that supports multi-platform [shared memory](http://en.wikipedia.org/wiki/Shared_memory) [multiprocessing](http://en.wikipedia.org/wiki/Multiprocessing) programming in [C](http://en.wikipedia.org/wiki/C_%28programming_language%29), [C++](http://en.wikipedia.org/wiki/C%2B%2B), and [Fortran](http://en.wikipedia.org/wiki/Fortran),[[4]](http://en.wikipedia.org/wiki/OpenMP#cite_note-OSConcepts-4) on most [processor architectures](http://en.wikipedia.org/wiki/Processor_architecture) and [operating systems](http://en.wikipedia.org/wiki/Operating_system), including [Solaris](http://en.wikipedia.org/wiki/Solaris_%28operating_system%29), [AIX](http://en.wikipedia.org/wiki/IBM_AIX), [HP-UX](http://en.wikipedia.org/wiki/HP-UX), [GNU/Linux](http://en.wikipedia.org/wiki/GNU/Linux), [Mac OS X](http://en.wikipedia.org/wiki/Mac_OS_X), and [Windows](http://en.wikipedia.org/wiki/Microsoft_Windows) platforms. It consists of a set of [compiler directives](http://en.wikipedia.org/wiki/Compiler_directive), [library routines](http://en.wikipedia.org/wiki/Library_%28computing%29), and [environment variables](http://en.wikipedia.org/wiki/Environment_variable) that influence run-time behavior [[[44]](#endnote-45)].

OpenMP is an implementation of [multi-threading](http://en.wikipedia.org/wiki/Thread_%28computer_science%29), a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and a task is divided among them. The threads then run concurrently, with the [runtime environment](http://en.wikipedia.org/wiki/Runtime_environment) allocating threads to different processors.

The section of code that is meant to run in parallel is marked accordingly, with a [pre-processor directive](http://en.wikipedia.org/wiki/Preprocessor_directive) that will cause the threads to form before the section is executed. Each thread has an id attached to it which can be obtained using a [function](http://en.wikipedia.org/wiki/Function_%28computer_science%29) (called omp\_get\_thread\_num()). The thread id is an integer, and the master thread has an id of 0. After the execution of the parallelized code, the threads join back into the master thread, which continues onward to the end of the program. Figure 25 demonstrates the process.

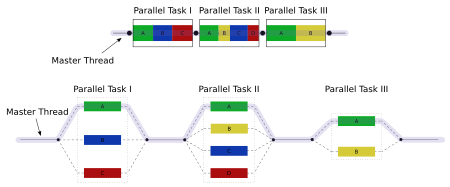
****

Figure 25 *OpenMP*

By default, each thread executes the parallelized section of code independently. Work-sharing constructs can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both [task parallelism](http://en.wikipedia.org/wiki/Task_parallelism) and [data parallelism](http://en.wikipedia.org/wiki/Data_parallelism) can be achieved using *OpenMP* in this way.

The runtime environment allocates threads to processors depending on usage, machine load and other factors. The number of threads can be assigned by the runtime environment based on [environment variables](http://en.wikipedia.org/wiki/Environment_variables) or in code using functions. The *OpenMP* functions are included in a [header file](http://en.wikipedia.org/wiki/Header_file) labelled omp.h in [C](http://en.wikipedia.org/wiki/C_%28programming_language%29)/[C++](http://en.wikipedia.org/wiki/C%2B%2B).

### *Implementation with OpenMP*

*OpenMP* uses the fork-and-join parallelism model. In fork-and-join, parallel threads are created and branched out from a master thread to execute an operation and will only remain until the operation has finished, after which all the threads are destroyed, thus leaving only one master thread. So first, we need to know how many threads we require for our system.

1. Determine the thread number

The number of threads required to solve a problem is generally limited to the number of CPUs one has. Whenever threads are created, a little time is taken to create a thread and later to join the end result and destroy the threads. When the problem is small, and the number of CPUs are less than the number of threads, the total execution time will be longer (slower) because more time has been spent to create threads, and later switch between the threads (due to preemptive behavior) then to actually solve the problem. Whenever a thread context is switched, data must be saved or loaded from the memory. This takes time.

The rule is simple, since all the threads will be executing the same operation (hence the same priority), one thread is sufficient per CPU (or core). The more CPUs one has, the more threads one can create. Most compiler directives in *OpenMP* use the Environment Variable OMP\_NUM\_THREADS to determine the number of threads to create.

In this thesis work, the filter is used to process frame data when one frame is ready. Each pixel is rendered independently of every other pixel, and it does not matter in which order they are processed since there are no data dependencies which are embarrassingly parallel. For this, the *OpenMP* API can really help make it easy to write parallel programs. Instead of spawning a thread, just tell the compiler which sections of code can be made parallel, and optionally tell it which variables can be shared between threads, which need to be private, and a whole myriad of other optional information and the compiler generates the right code.

1. Parallel for calibration

As described in section 4.2 and 4.3, we can get offset and gain through the calibration process and store it in local array. Then we can correct the input Iinput [n] as

Ioutput[n] =( Iinput [n] – O[n]) \* G[n]

where O[n] is pixel[n]’s offset value; G[n] is pixel[n]’s gain value.; and n is the pixel index of detector. Figure 26 shows the parallelized code for calibration.

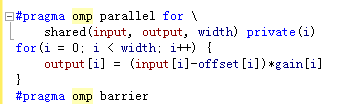
****

Figure 26 Parallelized code for calibration.

# TEST RESULTS AND APPLICATIONS

This chapter describes the measurements that were conducted on the *ApponsView* system. First, moving average filter was tested. Then, the gain and offset correct were verified. Finally, some X-ray applications from the customer were introduced.

## ApponsView Test Result

We tested a detector which has 256 pixels with *ApponsView*, and the result was measured by Mathcad. Figure 27(a) shows the noise of the offset before using moving average filter. The average RMS value is 768 ADC counts. With a 4-order average filter applied, the noise value is 396 ADC counts, as shown in Figure 27(b). When an 8-order moving average filter is used, the filtered noise decreased to 306 ADC counts, as shown in Figure 27(c). The theoretical value and the measured results are normalized and listed in Table 1. It shows that the results match well with what the theory predicted.

Table 1 Comparison between the theoretical values and the measured results

|  |  |  |
| --- | --- | --- |
| Filter order N | Theoretical RMS noise | Measured noise results |
| N=0 (No filtering) | 1 | 1 |
| N=4 | 0.500 | 0.513 |
| N=8 | 0.354 | 0.374 |

.



**ADC counts**

**Channel number**

**(a)**

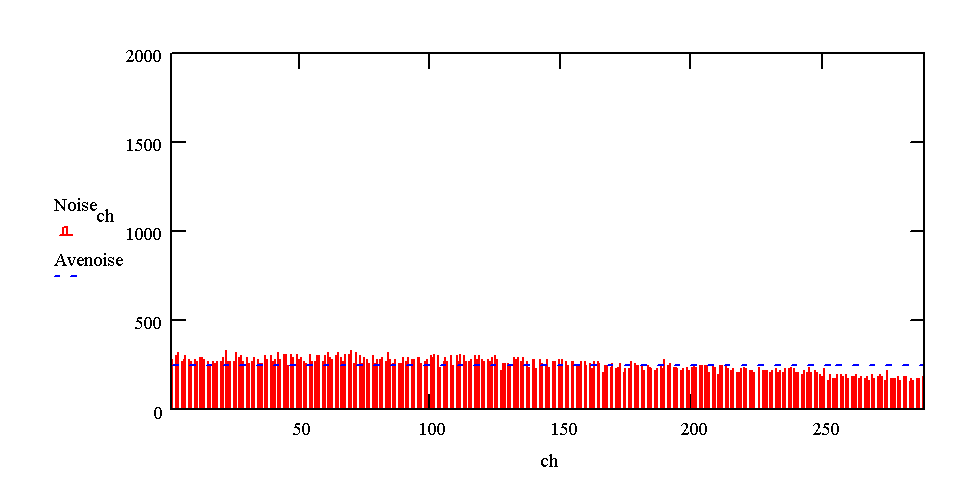
**ADC counts**



**Channel number**

**(b)**

**ADC counts**

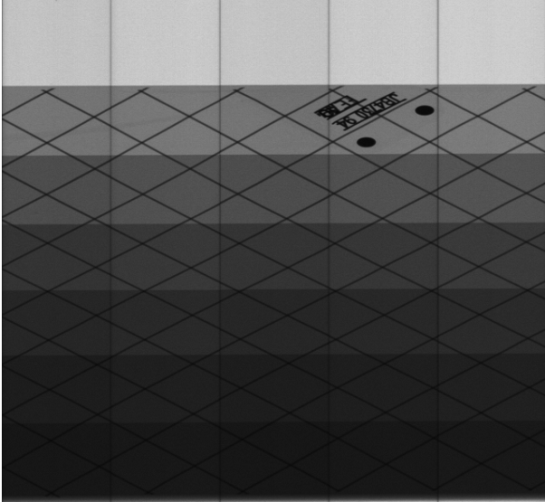
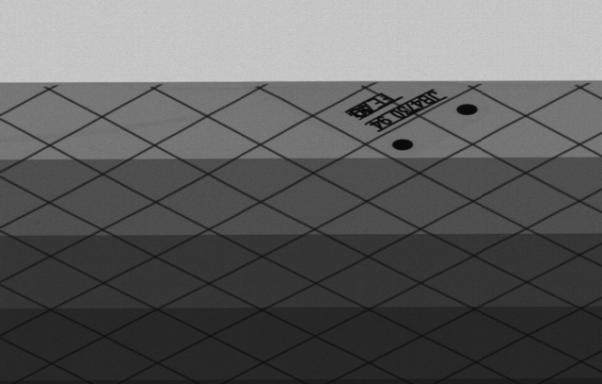


**Channel number**

**(c)**

Figure 16 The noise of the offset values (a) without filtering (b) A four order moving average filter applied; (c) An eight order moving average filter applied.

Figure 28Figure 17 is a sample image made with the X-ray line-scan camera. Figure 28Figure 17 (a) is a telephone’s X-ray image made without calibrations. Figure 28Figure 17 (b) is the image made with calibration. From the comparison it can be seen that the fixed pattern noise of the detectors is totally removed by the digital correction.

(a) (b)

Figure 17 X-ray scanning images (a) without calibration (b) with calibration.

The X-ray line-scan detector could reach 20 cm/s without any geometric distortion. It can meet the requirement of the customer requirement.

## Applications

Luggage inspection is one of the main applications of the X-ray detector. Weapons or dangerous tools such as knives can be easily seem by the detector without opening the suitcase. Pseudo color can also be applied to the image. The target material is normally different than that of the foreign objects, so a threshold value can be set for the gray scale. When the gray scale value is larger than certain value, then the computer will show a predefined color to the pixels. Figure 29 shows a suitcase X-ray image.

In electronic assembly industry, the detector can be used for the inspection of the connector, metal wires, and buttons, etc. Figure 19 shows one application of it.

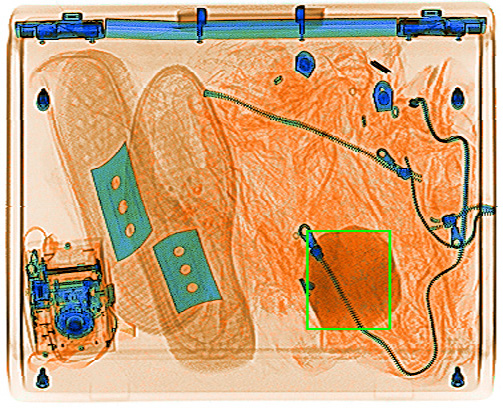


Figure 18 Custom security inspection.

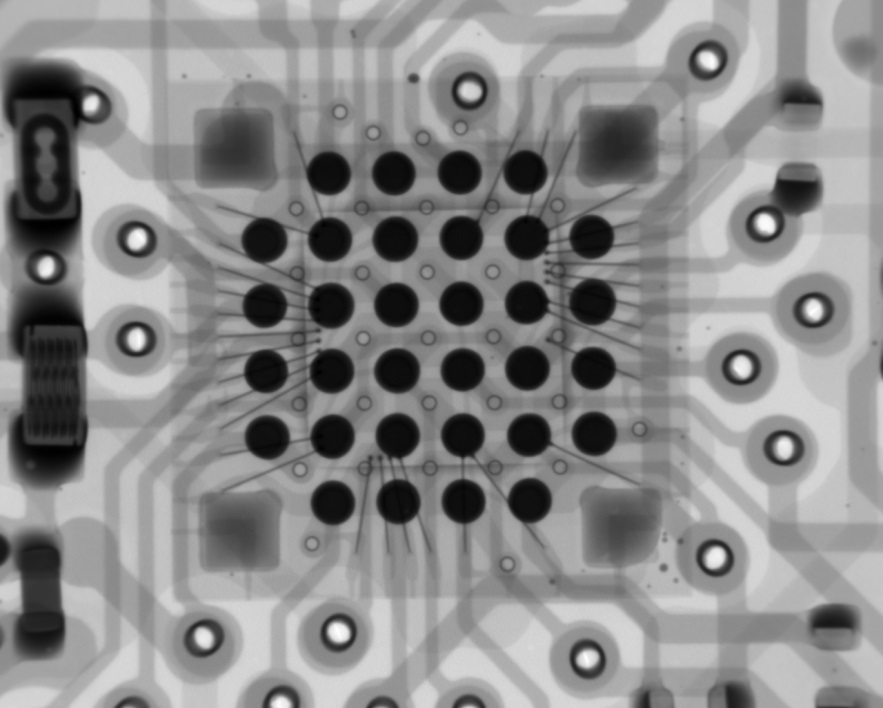


Figure 19 Electronic assembly industry quality inspection.

# Conclusions

The X-ray line-scan detector system studied in this thesis is a complicated real-time system. The whole system is introduced in this thesis, and the software design of the system is presented. To meet the target requirements of low-cost, acceptable-performance, high-flexibility, and excellent-reliability, the architecture of firmware and software is discussed and described in detail.

Traditionally, the X-ray line-scan detector uses the FPGA or dedicated DSP processor for signal processing because of hardware limitations. Neither of these would be flexible or economically viable most application environments.

To make the system more flexible and cheaper, the Ethernet interface is implemented in the X-ray line-scan detector. It makes for the possibility of utilizing the detector without the expensive frame trigger card.

Fixed pattern noise (offset and gain) are totally removed by the software corrections. White noise is also reduced by the moving average filter presented in this thesis. The scanning speed reaches 20cm/s, which meets the design goal. Parallel optimization of the processing algorithm is used to improve the system performance.

# References

[] Koskinen J., “LDA technology today and possibilities in the future,” in Proceedings of the ASNT Digital Imaging IV conference, August 2001.

[2] Holland S., “Fabrication of detectors and transistors on high-resistivity silicon,” Nuclear Instruments and Methods in Physics Research, A275: 537-541, 1989

[3] Kemmer J., “Improvement of detector fabrication by the planar process,” Nuclear Instruments and Methods in Physics Research, A226: 89-93, 1984.

[4] Betta G., Pignatel G. U., Verzellesi G., and Boscardin, M., “Si-PIN X-Ray detector technology,” Nuclear Instruments and Methods in Physics Research, A395: 344-348, 1997.

[5] Hoe D. and Ribner D. “An auto-ranging photodiode preamplifier with 114dB dynamic range,” IEEE Journal of Solid-State Circuits, vol.31, pp. 187-194, February 1996.

[6] Heijne E., “A 1006 element hybrid silicon pixel detector with strobed binary output,” European organization for nuclear research CERN/ECP91-26, 13th, pp. 1-10, December 1991

[7] Matsumoto H. and Watanabe K., “Spike-free switched-capacitor circuits,” Electronics letters, vol. 23, pp. 428-429, April, 1987

[8] Thelen D. and Chu D. “A low noise readout detector circuit for nanoampere sensor applications,” IEEE Journal of Solid-State Circuits, vol. 32, pp. 337-348, March 1997

[9] Orfanidis S. Introduction to signal processing. Prentice Hall, New York: 1996.

[0] James F. Cox (26 June 2001). [Fundamentals of linear electronics: integrated and discrete](http://books.google.com/books?id=FbezraN9tvEC&pg=PA91). Cengage Learning. pp. 91–. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-7668-3018-9](http://en.wikipedia.org/wiki/Special:BookSources/978-0-7668-3018-9). Retrieved 20 August 2011.

[1] Basics of Spectral Measurement JETI Technische Instrumente Line Arrays for Miniaturized Spectrometers GmbH 2005

[2] Wisniewski, Remigiusz (2009). Synthesis of compositional microprogram control units for programmable devices. Zielona Góra: University of Zielona Góra. p. 153. ISBN 978-83-7481-293-1.

[3] Mike Thompson. "Mixed-signal FPGAs provide GREEN POWER". EE Times, 2007-07-02.

[4] Ciletti, Michael D. (2010). Advanced Digital Design with Verilog HDL. Prentice Hall.

[5] Oshana, Rob (29 October 2002). ["Introduction to JTAG"](http://www.embedded.com/electronics-blogs/beginner-s-corner/4024466/Introduction-to-JTAG). Embedded Systems Design. Retrieved 5 April 2007.

[6] Thomas, Donald, Moorby, Phillip "The Verilog Hardware Description Language" Kluwer Academic Publishers, Norwell, MA. ISBN 0-7923-8166-1

[7] Janick Bergerdon, "Writing Testbenches: Functional Verification of HDL Models", 2000, ISBN 0-7923-7766-4. (The HDL Testbench Bible)

[8] Frank Vahid (2010). [Digital Design with RTL Design, Verilog and VHDL](http://books.google.com/books?id=-YayRpmjc20C&pg=PA247) (2nd ed.). John Wiley and Sons. p. 247. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-470-53108-2](http://en.wikipedia.org/wiki/Special:BookSources/978-0-470-53108-2).

[9] NIOS II Processor Overview

Url: <http://www.altera.com/devices/processor/nios2/ni2-index.html>

[20] István Vassányi. "Implementing processor arrays on FPGAs". 1998.

[21] FPGA Architectures from 'A' to 'Z'" by Clive Maxfield 2006

<http://www.embedded.com/columns/showArticle.jhtml?articleID=192700615>

[22] SOPC Bulder

<http://www.altera.com/literature/lit-sop.jsp>

[23] Joseph G. Davies and Thomas F. Lee. Microsoft Windows Server 2003 TCP/IP Protocols and Services. ISBN 0-7356-1291-9

[24] Ian McLean. Windows(R) 2000 TCP/IP Black Book. ISBN 1-57610-687-X

[25] Forouzan, Behrouz A. (2003). TCP/IP Protocol Suite (2nd ed.). McGraw-Hill. ISBN 0-07-246060-1.

[26] W. Richard Stevens. TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols. ISBN 0-201-63495-3

[27] The Authoritative Dictionary of IEEE Standards Terms. p. 438. doi:10.1109/IEEESTD.2000.322230. ISBN 0738126012.

[28] Cross compiler wiki

<http://en.wikipedia.org/wiki/Cross-compiler>

[29] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck : Discrete-Time Signal Processing, Prentice Hall, ISBN 0-13-754920-2

[30] Richard G. Lyons: Understanding Digital Signal Processing, Prentice Hall, ISBN 0-13-108989-7

[3] Electronic Shuttering for High Speed CMOS Machine Vision Applications <http://www.automaatioseura.fi/jaostot/mvn/mvn2007/parameter.html>

[32] Carter,Mancini, Bruce,Ron (2009). Op Amps for Everyone. Texas Instruments. p. 10-11. ISBN 0080949487.

[33] Rabiner, Lawrence R., and Gold, Bernard, 1975: Theory and Application of Digital Signal Processing (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.) ISBN 0-13-914101-4

[34] Introduction to ActiveX Controls at microsoft.com, accessed 18 January 2008

<http://msdn2.microsoft.com/en-us/library/aa751972(VS.85).aspx>

[35] Summerfield, Mark (23 August 2010). "Advanced Qt Programming: Creating Great Software with C++ and Qt 4" (1st ed.). Addison-Wesley. ISBN 978-0-321-63590-7.

[36] Margaret Rouse (March 27, 2007). "Definition: multi-core processor". TechTarget. Retrieved March 6, 2013.

[37] Margaret Rouse (March 27, 2007). "Definition: multi-core processor". TechTarget. Retrieved March 6, 2013.

[38] Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved 2007-11-09.

[39] [Using OpenMP - Portable Shared Memory Parallel Programming](http://openmp.org/wp/2009/04/download-book-examples-and-discuss/)

<http://openmp.org/wp/2009/04/download-book-examples-and-discuss/>

[40] Quinn Michael J, Parallel Programming in C with MPI and OpenMP McGraw-Hill Inc. 2004. ISBN 0-07-058201-7

[4] R. Eigenmann (Editor), M. Voss (Editor), OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2001, West Lafayette, IN, USA, July 30–31, 2001. (Lecture Notes in Computer Science). Springer 2001. ISBN 3-540-42346-X

[42] "Intel Cilk Plus: A quick, easy and reliable way to improve threaded performance", Intel. Retrieved on 2010-09-14.

<http://software.intel.com/en-us/articles/intel-cilk-plus/>

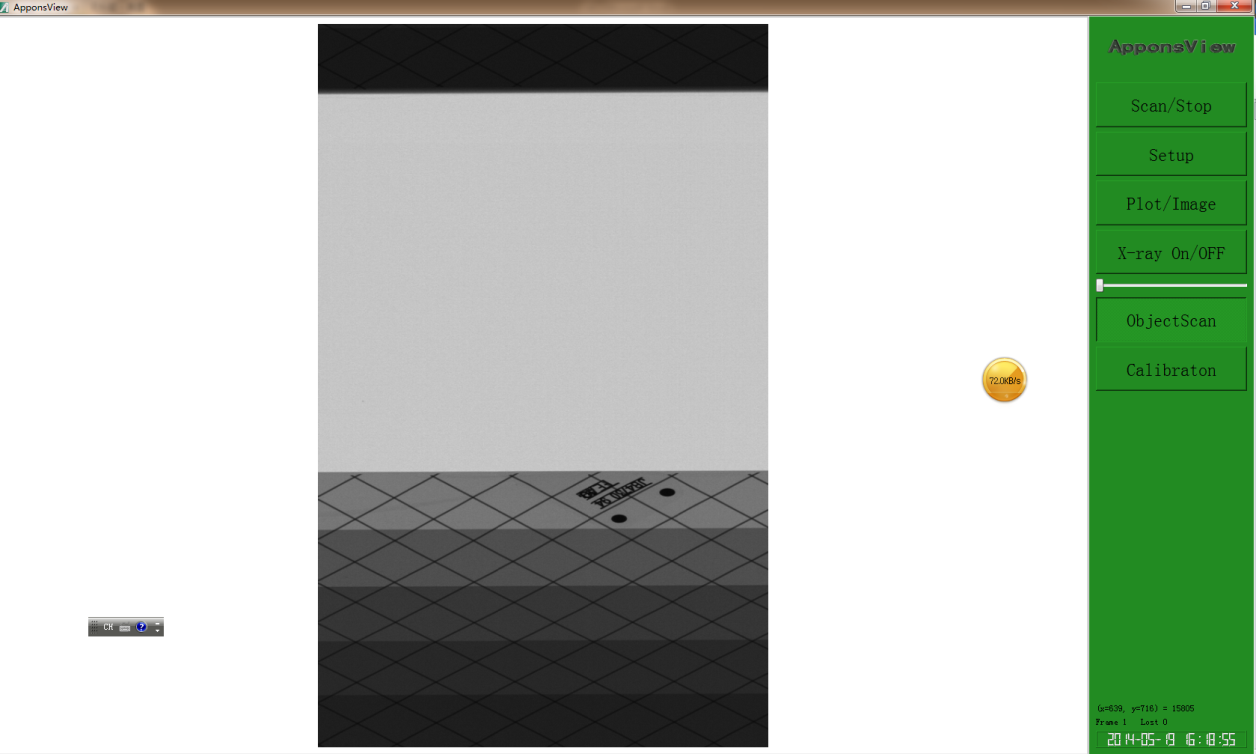
[43] David A. Patterson and John L. Hennessey, Computer Organization and Design: the Hardware/Software Interface, 2nd Edition, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998, p.751

[44] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP. Morgan Kaufmann, 2000. ISBN 1-55860-671-8

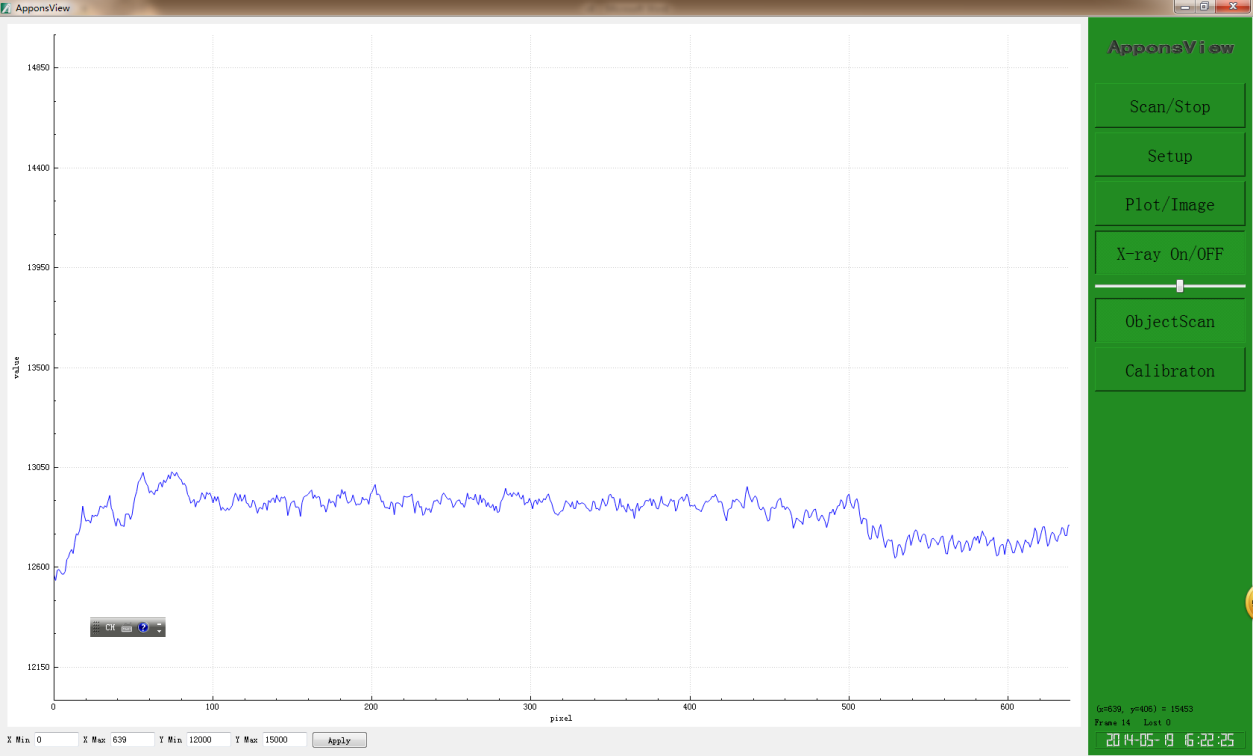
# Appendices

**Appendix 1**. ApponsView UI

Appendix 1. ApponsView User Interface



Scan test object



Show the average column value of the image

1. [] Koskinen J., “LDA technology today and possibilities in the future,” in Proceedings of the ASNT Digital Imaging IV conference, August 2001. [↑](#endnote-ref-2)
2. [] Holland S., “Fabrication of detectors and transistors on high-resistivity silicon,” Nuclear Instruments and Methods in Physics Research, A275: 537-541, 1989 [↑](#endnote-ref-3)
3. [] Kemmer J., “Improvement of detector fabrication by the planar process,” Nuclear Instruments and Methods in Physics Research, A226: 89-93, 1984. [↑](#endnote-ref-4)
4. [] Betta G., Pignatel G. U., Verzellesi G., and Boscardin, M., “Si-PIN X-Ray detector technology,” Nuclear Instruments and Methods in Physics Research, A395: 344-348, 1997. [↑](#endnote-ref-5)
5. [] Hoe D. and Ribner D. “An auto-ranging photodiode preamplifier with 114dB dynamic range,” IEEE Journal of Solid-State Circuits, vol.31, pp. 187-194, February 1996. [↑](#endnote-ref-6)
6. [] Heijne E., “A 1006 element hybrid silicon pixel detector with strobed binary output,” European organization for nuclear research CERN/ECP91-26, 13th, pp. 1-10, December 1991 [↑](#endnote-ref-7)
7. [] Matsumoto H. and Watanabe K., “Spike-free switched-capacitor circuits,” Electronics letters, vol. 23, pp. 428-429, April, 1987 [↑](#endnote-ref-8)
8. [] Thelen D. and Chu D. “A low noise readout detector circuit for nanoampere sensor applications,” IEEE Journal of Solid-State Circuits, vol. 32, pp. 337-348, March 1997 [↑](#endnote-ref-9)
9. [] Orfanidis S. Introduction to signal processing. Prentice Hall, New York: 1996. [↑](#endnote-ref-10)
10. [] James F. Cox (26 June 2001). [Fundamentals of linear electronics: integrated and discrete](http://books.google.com/books?id=FbezraN9tvEC&pg=PA91). Cengage Learning. pp. 91–. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-7668-3018-9](http://en.wikipedia.org/wiki/Special:BookSources/978-0-7668-3018-9). Retrieved 20 August 2011. [↑](#endnote-ref-11)
11. [] Basics of Spectral Measurement JETI Technische Instrumente Line Arrays for Miniaturized Spectrometers GmbH 2005 [↑](#endnote-ref-12)
12. [] Wisniewski, Remigiusz (2009). Synthesis of compositional microprogram control units for programmable devices. Zielona Góra: University of Zielona Góra. p. 153. ISBN 978-83-7481-293-1. [↑](#endnote-ref-13)
13. [] Mike Thompson. "Mixed-signal FPGAs provide GREEN POWER". EE Times, 2007-07-02. [↑](#endnote-ref-14)
14. [] Ciletti, Michael D. (2010). Advanced Digital Design with Verilog HDL. Prentice Hall. [↑](#endnote-ref-15)
15. [] Oshana, Rob (29 October 2002). ["Introduction to JTAG"](http://www.embedded.com/electronics-blogs/beginner-s-corner/4024466/Introduction-to-JTAG). Embedded Systems Design. Retrieved 5 April 2007. [↑](#endnote-ref-16)
16. [] Thomas, Donald, Moorby, Phillip "The Verilog Hardware Description Language" Kluwer Academic Publishers, Norwell, MA. ISBN 0-7923-8166-1 [↑](#endnote-ref-17)
17. [] Janick Bergerdon, "Writing Testbenches: Functional Verification of HDL Models", 2000, ISBN 0-7923-7766-4. (The HDL Testbench Bible) [↑](#endnote-ref-18)
18. [] Frank Vahid (2010). [Digital Design with RTL Design, Verilog and VHDL](http://books.google.com/books?id=-YayRpmjc20C&pg=PA247) (2nd ed.). John Wiley and Sons. p. 247. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-470-53108-2](http://en.wikipedia.org/wiki/Special:BookSources/978-0-470-53108-2). [↑](#endnote-ref-19)
19. [] NIOS II Processor Overview

    Url: <http://www.altera.com/devices/processor/nios2/ni2-index.html> [↑](#endnote-ref-20)
20. [] István Vassányi. "Implementing processor arrays on FPGAs". 1998. [↑](#endnote-ref-21)
21. [] FPGA Architectures from 'A' to 'Z'" by Clive Maxfield 2006

    <http://www.embedded.com/columns/showArticle.jhtml?articleID=192700615> [↑](#endnote-ref-22)
22. [] SOPC Bulder

    <http://www.altera.com/literature/lit-sop.jsp> [↑](#endnote-ref-23)
23. [] Joseph G. Davies and Thomas F. Lee. Microsoft Windows Server 2003 TCP/IP Protocols and Services. ISBN 0-7356-1291-9 [↑](#endnote-ref-24)
24. [] Ian McLean. Windows(R) 2000 TCP/IP Black Book. ISBN 1-57610-687-X [↑](#endnote-ref-25)
25. [] Forouzan, Behrouz A. (2003). TCP/IP Protocol Suite (2nd ed.). McGraw-Hill. ISBN 0-07-246060-1. [↑](#endnote-ref-26)
26. [] W. Richard Stevens. TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols. ISBN 0-201-63495-3 [↑](#endnote-ref-27)
27. [] The Authoritative Dictionary of IEEE Standards Terms. p. 438. doi:10.1109/IEEESTD.2000.322230. ISBN 0738126012. [↑](#endnote-ref-28)
28. [] Cross compiler wiki

    <http://en.wikipedia.org/wiki/Cross-compiler> [↑](#endnote-ref-29)
29. [] Alan V. Oppenheim, Ronald W. Schafer, John R. Buck : Discrete-Time Signal Processing, Prentice Hall, ISBN 0-13-754920-2 [↑](#endnote-ref-30)
30. [] Richard G. Lyons: Understanding Digital Signal Processing, Prentice Hall, ISBN 0-13-108989-7 [↑](#endnote-ref-31)
31. [] Electronic Shuttering for High Speed CMOS Machine Vision Applications <http://www.automaatioseura.fi/jaostot/mvn/mvn2007/parameter.html> [↑](#endnote-ref-32)
32. [] Carter,Mancini, Bruce,Ron (2009). Op Amps for Everyone. Texas Instruments. p. 10-11. ISBN 0080949487. [↑](#endnote-ref-33)
33. [] Rabiner, Lawrence R., and Gold, Bernard, 1975: Theory and Application of Digital Signal Processing (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.) ISBN 0-13-914101-4 [↑](#endnote-ref-34)
34. [] Introduction to ActiveX Controls at microsoft.com, accessed 18 January 2008

    <http://msdn2.microsoft.com/en-us/library/aa751972(VS.85).aspx> [↑](#endnote-ref-35)
35. [] Summerfield, Mark (23 August 2010). "Advanced Qt Programming: Creating Great Software with C++ and Qt 4" (1st ed.). Addison-Wesley. ISBN 978-0-321-63590-7. [↑](#endnote-ref-36)
36. [] Margaret Rouse (March 27, 2007). "Definition: multi-core processor". TechTarget. Retrieved March 6, 2013. [↑](#endnote-ref-37)
37. [] Margaret Rouse (March 27, 2007). "Definition: multi-core processor". TechTarget. Retrieved March 6, 2013. [↑](#endnote-ref-38)
38. [] Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved 2007-11-09. [↑](#endnote-ref-39)
39. [] [Using OpenMP - Portable Shared Memory Parallel Programming](http://openmp.org/wp/2009/04/download-book-examples-and-discuss/)

    <http://openmp.org/wp/2009/04/download-book-examples-and-discuss/> [↑](#endnote-ref-40)
40. [] Quinn Michael J, Parallel Programming in C with MPI and OpenMP McGraw-Hill Inc. 2004. ISBN 0-07-058201-7 [↑](#endnote-ref-41)
41. [] R. Eigenmann (Editor), M. Voss (Editor), OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2001, West Lafayette, IN, USA, July 30–31, 2001. (Lecture Notes in Computer Science). Springer 2001. ISBN 3-540-42346-X [↑](#endnote-ref-42)
42. [] "Intel Cilk Plus: A quick, easy and reliable way to improve threaded performance", Intel. Retrieved on 2010-09-14.

    <http://software.intel.com/en-us/articles/intel-cilk-plus/> [↑](#endnote-ref-43)
43. [] David A. Patterson and John L. Hennessey, Computer Organization and Design: the Hardware/Software Interface, 2nd Edition, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998, p.751 [↑](#endnote-ref-44)
44. [] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP. Morgan Kaufmann, 2000. ISBN 1-55860-671-8 [↑](#endnote-ref-45)