A DIGITAL SIGNAL PROCESSOR BASED DEVELOPMENT PLATFORM FOR THERMAL IMAGERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA ÖZKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2003

Approval of Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel DEMİREKLER
Head of Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. T. Engin TUNCER
Co-Supervisor

Assoc. Prof. Dr. Cengiz BEŞİKCİ
Supervisor

Examining Committee Members:

Prof. Dr. Nevzat GENCER (Chairman)

Assoc. Prof. Dr. Cengiz BEŞİKÇİ

Assoc. Prof. Dr. T. Engin TUNCER

Assist. Prof. Dr. Tolga ÇİLOĞLU

Assist. Prof. Dr. Atila YILMAZ

ABSTRACT

A DIGITAL SIGNAL PROCESSOR BASED DEVELOPMENT PLATFORM FOR THERMAL IMAGERS

Özkan, Mustafa
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Cengiz Beşikci
Co-Supervisor: Assoc. Prof. Dr. Temel Engin Tuncer

June 2003, 112 pages

The performance of thermal imaging systems depends on the sophistication level of the signal processing algorithms and the careful design of the required electronics. Digital Signal Processor (DSP) boards are very beneficial in designing such systems, and applying high level signal processing algorithms. The aim of this study is to design and implement a development platform for thermal imagers using a double floating point processor DSP board.

A front-end electronics board has been employed as the interface between the detector read-out integrated circuit (ROIC) and the DSP board. The output of the front-end electronics board is fed to the DSP board, which captures the 12-bit digital data and applies the basic signal processing algorithms to the raw image data. Capturing and signal processing software is constructed using the C programming language and loaded to the DSP with the compiler called Code Composer Studio. An image formation program is written with the functions of OpenGL graphics programming language in Visual C/C++ compiler. The implemented system is tested with a highly nonuniform 128x128 InSb detector array on Si substrate, and successful performance is achieved in real time.

The system can work with detector arrays up to a format of 320x256 at 25 Hz frame rate. Higher formats and frame rates are possible, if the delays in the front-end electronics board are reduced through the use of higher speed components in this board.

Keywords: Thermal Imaging, Digital Signal Processors, Normalization

ÖZ

TERMAL GÖRÜNTÜLEYİCİLER İÇİN SAYISAL SİNYAL İŞLEYİCİ TABANLI GELİŞTİRME PLATFORMU

Özkan, Mustafa
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Cengiz Beşikci
Ortak Tez Yöneticisi : Doç. Dr. Temel Engin Tuncer

Haziran 2003, 112 sayfa

Termal görüntüleme sistemlerinin performansı kullanılan sinyal işleme algoritmalarının yetenek düzeylerine ve gerekli elektronik birimlerin dikkatli tasarımına bağlıdır. Sayısal Sinyal İşleme (DSP) kartları bu sistemlerin tasarımında ve yüksek seviyeli sinyal işleme algoritmalarının uygulanmasında oldukça kullanılışlıdır. Bu çalışmanın amacı, çift kayar noktalı işlemciye sahip bir DSP kartı ile termal görüntüleme sistemleri için bir geliştirme platformu tasarlamak ve uygulamaktır.

Dedektör okuma tümdevresi (ROIC) ve DSP kartı arasında arayüzey olarak bir ön-elektronik kartı kullanılmıştır. Ön-elektronik kartının çıkışı, 12-bit sayısal veriyi yakalayıp ham görüntü verisine temel sinyal işleme algoritmalarını uygulayan DSP

kartı girişine verilmiştir. Veri yakalama ve sinyal işleme yazılımları C programlama dili kullanılarak oluşturulmuş ve DSP kartına Code Composer Studio (CCS) adlı derleyici yoluyla yüklenmiştir. Görüntü oluşturma programı, OpenGL grafik programlama dilinin fonksiyonları ile Visual C/C++ derleyicisinde yazılmıştır. Oluşturulan sistem, homojenlik düzeyi düşük olan ve 128x128 formatında olup Si taban üzerinde fabrike edilen bir InSb dedektör dizini ile test edilmiş ve başarılı olarak çalıştığı gözlenmiştir.

Sistem, 25 Hz resim hızında, 320x256 formatına kadar olan dedektör dizinleri ile çalışabilmektedir. Daha yüksek formatlar ve resim hızları daha yüksek hızlı devre elemanları kullanılarak ön-elektronik kartındaki gecikmelerin azaltılmasıyla mümkündür.

Anahtar Kelimeler: Termal Görüntüleme, Sayısal Sinyal İşleyiciler, Düzeltme Metodları

ACKNOWLEDGMENTS

I would like to thank my thesis advisors Assoc. Prof. Dr. Cengiz Beşikçi and Assoc. Prof. Dr. Temel Engin Tuncer for their guidance and support. I also would like to thank Fatih Işık, Selçuk Özer, Abdulkadir Yeler and Murat Tepegöz for their help in various stages of this work. I also would like to thank ASELSAN A.Ş. and my other friends at METU and ASELSAN for their help and support throughout this thesis study.

TABLE OF CONTENTS

ABSTRACT ..... **iii**

ÖZ ..... **v**

ACKNOWLEDGMENTS ..... **vii**

TABLE OF CONTENTS ..... **viii**

LIST OF TABLES ..... **xii**

LIST OF FIGURES ..... **xiii**

CHAPTER

1. INTRODUCTION **1**

1.1 Overview 1

1.2 Organization of the Thesis 2

2. A BRIEF INTRODUCTION TO INFRARED TECHNOLOGY **4**

2.1 Infrared Radiation 4

2.2 IR Imaging Systems 5

2.3 Atmospheric Transmission Window 7

2.4 Infrared Detectors 8

2.5 Infrared Imaging 11

2.6 Summary 12

3. INDIGO SYSTEM' s ISC9806 ROIC 13

3.1 Introduction 13
3.2 Simplified Unit Cell Schematic 13
3.3 Operation Modes 14
3.3.1 Default Mode 14
3.3.2 Command Mode 15
3.4 Integration Modes 19
3.5 Integration Time 22
3.6 Readout Timing 23
3.7 Control Signals for ROIC and Timing Requirements 23
3.8 Summary 26

4. THERMAL IMAGER DEVELOPMENT PLATFORM HARDWARE ..27

4.1 Introduction 27
4.2 DSP Board 29
4.2.1 On-Board Memory 31
4.2.2 On-Board Buses 31
4.2.3 I/O Port Module Site 32
4.3 I/O Module Extender Board 34
4.4 Front-end Electronics Card 34
4.5 Detector Dewar Assembly 39
4.6 Summary 39

5. DATA CAPTURE AND ROIC CONTROL 40

5.1 Introduction 40
5.2 Multi-channel Buffered Serial Port 41
5.3 EMIF 44
5.3.1 EMIF Signal Descriptions 44
5.3.2 EMIF Registers 45

5.3.3 Asynchronous Data Read with EMIF 47
5.4 Serial Data Transmission with DSP 49
5.5 Clock Generation with DSP 50
5.6 Parallel Interface with Front-End Electronics Card 51
5.7 Interrupt Missing 53
5.8 Summary 54

6. IMAGE FORMATION AND PROCESSING 55

6.1 Introduction 55
6.2 Introduction to OpenGL 55
6.3 Image Formation 57
6.4 Non-uniformity Correction 59
6.4.1 One-point Correction 59
6.4.2 Two-point Correction 62
6.5 Frame Averaging and Image Filtering 65
6.6 Mathematical Analysis of the Algorithms 67
6.7 Summary 69

7. CONCLUSION AND FUTURE WORK 70

REFERENCES 72

APPENDICES

A TMS320C6701 ARCHITECTURE 76

A.1 TMS320 Family Evolution 76
A.2 TMS320C6701 Architecture 77
A.3 Development Tools for the C6000 DSP Platform 79
A.3.1 Code Composer Studio 80

B CONFIGURATION OF McBSP 82

C SOURCE CODE FOR EMIF CONFIGURATION.....86

D THE CODE FOR REAL TIME DATA CAPTURE.....89

E SOURCE CODE FOR IMAGE FORMATION.....94

F THE CODES FOR NORMALIZATION COEFFICIENTS.....102

F.1 The Code for Calculating One-point Offset Coefficients..... 102

F.2 The Code for Calculating Two-point Gain Coefficients 103

G THE PROGRAM SEGMENTS FOR SIGNAL PROCESSING APPLICATIONS 106

G-1 One-point Offset Correction..... 106

G.2 Two-point Gain Correction..... 108

G.3 Frame Averaging..... 109

G.4 Sobel Filter Application..... 111

LIST OF TABLES

TABLE

2.1 Infrared Camera Applications 6
3.1.a Window Data Word 18
3.1.b Mode Word Data 18
3.2 Default values of Serial Control Register at Power-up 19
3.3 Clock Timing Requirements 22
3.4 ROIC Timing Signals 24
4.1 DSP Board's I/O Module Site Pinouts 33
5.1 McBSP Interface Signals 41
5.2 McBSP Registers and Addresses 43
5.3 Signal Descriptions of EMIF 45
5.4 EMIF Memory Mapped Registers 46
6.1 Mathematical Analysis of Algorithms 69
A.1 Characteristics of the TMS320C6701 Processors 79

LIST OF FIGURES

FIGURE

2.1 The Electromagnetic Spectrum.....4
2.2 Two infrared cameras available in the market 6
2.3 Examples for infrared images 7
2.4 Atmospheric transmission windows 8
2.5 Photovoltaic detector operation..... 9
2.6 Band diagram of photoconductive detectors. 10
3.1 Simplified unit cell schematic 14
3.2 Serial Control Register 16
3.3 Block diagram for Command Mode operation 17
3.4 Integrate-While-Read timing diagram 20
3.5 Integrate-Then-Read timing diagram 21
3.6 Detailed timing diagram for Integrate-Then-Read 21
3.7 Indigo System` s ISC9806 (ROIC) timing 25
4.1 System layout of thermal imager development platform 28
4.2 PCI/C6600 DSP board 29
4.3 PCI/C6600 Block Diagram 30
4.4 I/O module extender board..... 34
4.5 The front-end electronics board 36
4.6 Block diagram of front-end electronics circuit 37
4.7 Output of ADC 37
4.8 The user interface program 38

4.9 Detector Dewar Assembly (DDA) 39
5.1 McBSP Block Diagram 42
5.2 Block diagram of EMIF for asynchronous interface 44
5.3 C6701 EMIF CE(0/1/2/3) Space Control Register Diagram 46
5.4 Asynchronous Read Timing Example 48
5.5 Data delay 50
5.6 Programmable Frame Period and Width 51
5.7 Signal flow in the system 53
5.8 A snapshot for interrupt missing 54
6.1 The development platform 58
6.2 Raw thermal image of resistance heater captured by the system from a highly nonuniform 128x128 detector array. 58
6.3 One-point offset correction algorithm with pixel replacement. 61
6.4 The image of a funnel and hotplate 61
6.5 The raw and one-point corrected image when the funnel is between the hotplate and the constructed imaging system. 62
6.6 Two-point gain correction algorithm with pixel replacement 64
6.7 The raw and two-point corrected image when the funnel is between the hotplate and the constructed imaging system. 65
6.8 The effect of frame averaging algorithm after two-point gain correction 66
6.9 Sobel filtering with normalization and pixel replacement 67
A.1 TMS320C6701 Internal Architecture 77
A.2 Working diagram for the parts of Code Composer Studio (CCS) 80
A.3 The phases of the development cycle that CCS supports 81

CHAPTER 1

INTRODUCTION

1.1 Overview

All objects, even at room temperature, emit infrared radiation, which is in the invisible region of the electromagnetic spectrum. The infrared detectors convert this radiation into electrical signals, and thermal imaging systems display the infrared image on the screen. [1]

Focal-Plane-Array (FPA) is the common name for an array of detectors. As the number of the rows and columns of FPA increases, the resolution of infrared imaging is improved. There are two types of infrared FPAs: monolithic and hybrid. Monolithic FPAs have both IR-sensitive detectors and signal processing circuit on the same layer, while the hybrid array has the IR-sensitive detectors on one layer and the signal-transmission and processing circuitry on another layer. The signal transmission path and the related electronics are usually integrated on a chip called the ROIC. [2] The analog signal coming from each detector is passed through the signal transmission path and amplifier, and digitized by an ADC (Analog to Digital Converter) before applying signal processing and image formation algorithms. The performance of the thermal imaging systems depends on the sophistication level of the signal processing algorithms, as well as on the careful design of the required electronics. These systems provide vision at day and night. Scientists are working on

constructing and improving algorithms for applications such as target tracking, target identification and detection of cancer cells.

This thesis work involves the design and implementation of a development platform for thermal imagers using a DSP board having two high speed processors, TMS320C6701 whose architecture is given in Appendix A. The implemented development platform consists of a DSP board, a front-end electronics circuitry, a PC running under MS Windows NT 4.0 OS and two compilers: Code Composer Studio 1.2 and Microsoft Visual C/C++. The front-end electronics system sends clock signals to ROIC, performs low noise detector bias and reference generation, filtering and AC coupling, amplification, automatic gain control (AGC), coarse offset correction, DC restoration, 12 bit digitization, and FPA temperature measurement. Parallel data capture and signal processing applications are performed with the DSP board. The image formation is implemented with the OpenGL graphic programming language.

The work implemented in this thesis consists of the initial steps in the construction of a sophisticated development platform for high performance thermal imagers capable of implementing high level image processing algorithms. In the extent of future study, the system will be enhanced by constructing additional sophisticated processing algorithms such as target tracking and target identification.

1.2 Organization of the Thesis

This thesis consists of seven chapters. The second chapter gives information on the basics of infrared detectors and infrared imaging systems. Chapter 3 describes the Indigo System's ROIC, ISC9806. The operation modes of this multiplexer and timing requirements are discussed. Chapter 4 presents the hardware of the developed system. All of the components used and how they are connected to each other are

described in detail. Chapter 5 describes the methods adopted for controlling the ROIC with the DSP Board. Parallel data capture with the parallel port of the DSP and the required timing calculations for capturing data from the ADC asynchronously are described in detail. The image formation and signal processing algorithms are presented in chapter 6. Finally, conclusion and future work are given in chapter 7.

CHAPTER 2

A BRIEF INTRODUCTION TO INFRARED TECHNOLOGY

2.1 Infrared Radiation

Electromagnetic radiation is available in many different forms in nature. Visible light, ultraviolet light, radio waves, and X-rays having different wavelengths are all examples of electromagnetic radiation. The region of infrared (IR) radiation in the electromagnetic spectrum is between visible light and microwaves. Figure 2.1 shows the major divisions of the electromagnetic spectrum.

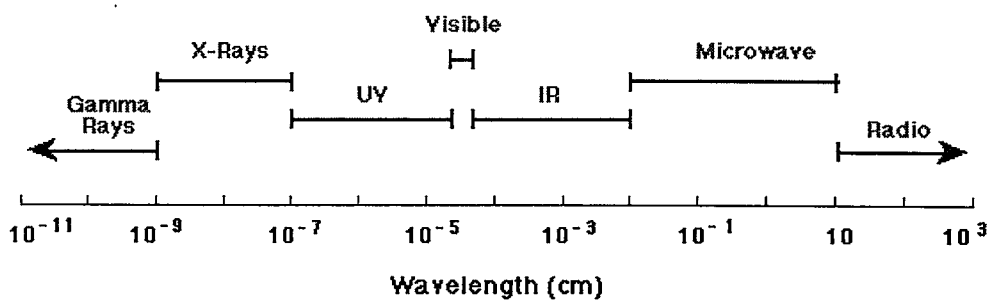


Figure 2.1 The Electromagnetic Spectrum [3]

All objects emit temperature-dependent infrared radiation. The hotter an object is, the more intense infrared radiation it sends. At moderate temperatures (above 200°F), the intensity of the radiation is enough for human body to detect that radiation as heat. [1]

2.2 IR Imaging Systems

The infrared energy cannot be seen by naked eye, but can be detected electronically. The infrared light can be converted to an electrical signal with optoelectronic components. This invisible energy can be monitored and analyzed with infrared cameras and computers. Unlike a conventional video camera, an infrared camera works as well in total darkness as it does in normal daylight. [4] Infrared cameras are used in many applications. These are summarized in Table 2.1. [5]

In IR imaging systems, resolving fine detail (spatial resolution) and small temperature differences (thermal resolution) are important performance parameters. A high quality system should respond to a rapidly changing scene without blurring and should view large temperature span without saturating. [6-8] Figure 2.2 shows some examples for such infrared cameras. Figure 2.3 gives some examples for infrared images taken by such infrared cameras.

Table 2.1 Infrared camera applications [5]

Military	**Industrial**	**Scientific**	**Medical**	**Civil**
Missile guidance	Plant maintenance	Pollution control	Mammography	Law enforcement
Target acquisition	Quality control	Energy conversation	Detection of arterial constriction	Fire fighting
Weapon guidance	Non-destructive Testing	Resource development	Evaluation of soft tissue injury	Surveillance
Surveillance				Border patrol
Reconnaissance				
Navigation				

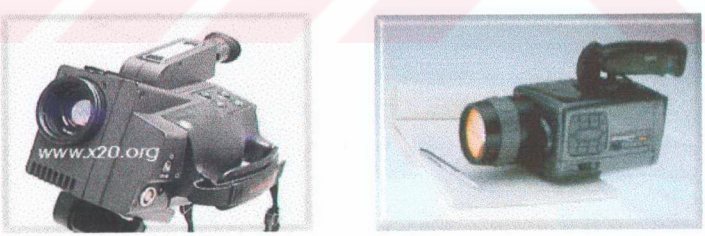
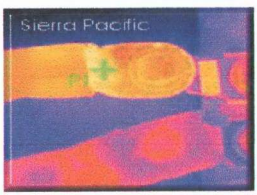
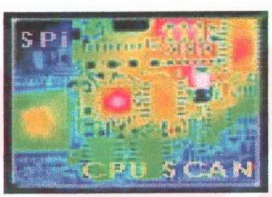


Figure 2.2 Two infrared cameras available in the market [5]

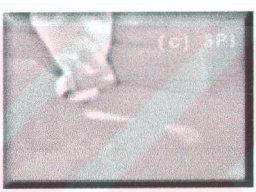
a) Thermal Medical Imaging



b) Electrical hot connection



c) Image of PC board



d) Infrared_footprints

Figure 2.3 Examples for infrared images [5]

2.3 Atmospheric Transmission Window

The chemicals in the atmosphere absorb infrared radiation generating four different atmospheric windows. The 0.7-1.1 μm range is the Near Infrared region. Short Wavelength Infrared region is in the wavelength ranges of 1.5-1.8 and 2.0-2.4 μm. Other regions are Medium Wavelength Infrared (3-5 μ range) and the Long Wavelength Infrared (8-14 μm range), which are the primary bands for thermal imaging. In these windows atmospheric transmission is maximum or equivalently the

absorption is minimum. Figure 2.4 shows the atmospheric windows described above. [9]

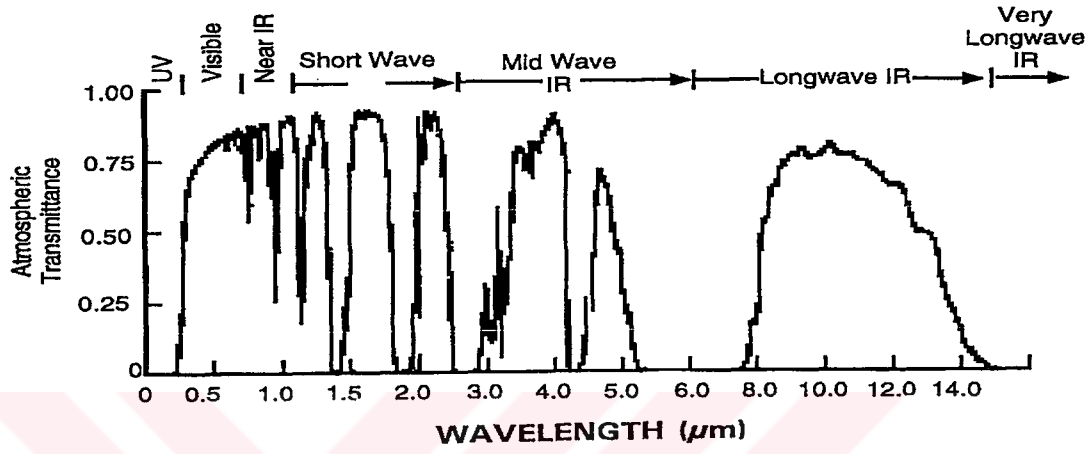


Figure 2.4 Atmospheric transmission windows [9]

2.4 Infrared Detectors

An infrared detector converts the radiation coming from the objects to electrical signal, which depends on the amount of radiation hitting the active region of the detector. Infrared detectors can be classified as thermal or photon detectors. Photon type detectors operate on the principles of direct electron-photon interaction. They provide superior sensitivity and response speed. However, photon detectors require to be cooled to cryogenic temperatures to minimize the noise, which is a costly operation. Thermal detectors convert incident radiation into heat, thereby raising the temperature of the detector element. This change in temperature is then converted to an electrical signal that can be amplified and displayed. However, the sensitivity is

lower and the response time is longer than for photon detectors. They do not need to be cooled to cryogenic temperatures. Some examples for thermal detectors are thermopile, bolometer, Golay cell , and pyroelectric detectors. [10,11]

Photo-detectors are classified as photoconductive and photovoltaic detectors. In photovoltaic ones when a photon strikes an electron in the valence band (filled orbitals), the electron is excited to the conduction band (unfilled orbitals) creating an electron (-) - hole (+) pair. The concentration of these electron-hole pairs is dependent on the amount of light striking the semiconductor. Photovoltaic detectors contain a p-n junction, which causes the electron-hole pairs to separate by the built-in electric field to produce a voltage that can be measured. The operation of photovoltaic detector is shown in Figure 2.5.

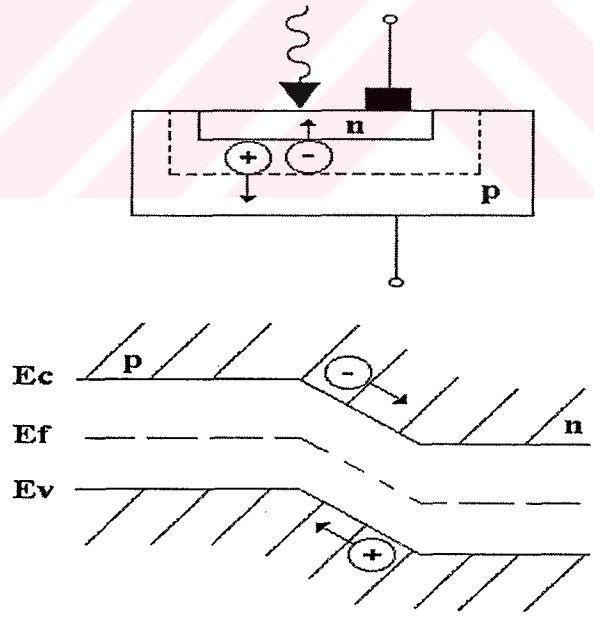


Figure 2.5 Photovoltaic detector operation [12]

Examples of photovoltaic infrared detector materials are indium antimonide (InSb), mercury cadmium telluride (MCT), platinum silicide (PtSi) -silicon Schottky barrier. [10]

Photogeneration of charge carriers (electrons, holes or electron-hole pairs) is the basic principle of photoconductive detectors. The conductivity of the device material is increased with these charge carriers and an electric circuit is used to measure this decrease in resistance. Detector materials possible to use for photoconductive detectors are indium antimonide (InSb), mercury cadmium telluride (mercad, MCT), lead sulfide (PbS), and lead selenide (PbSe). Band diagram of photoconductive detectors is shown in Figure 2.6. [10]

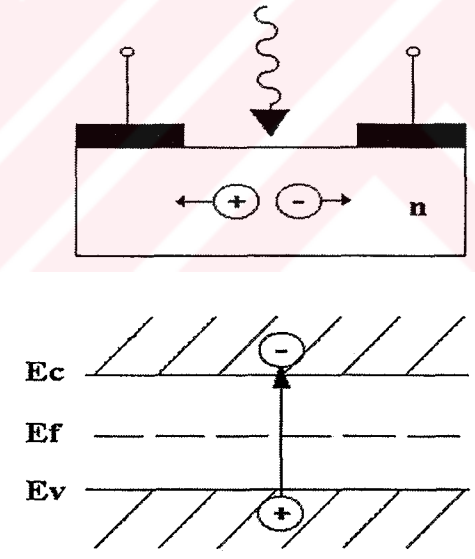


Figure 2.6 Band diagram of photoconductive detectors.[12]

If the photo-transitions take place across the fundamental band gap of the infrared material, photon detectors are classified as intrinsic. If the photo-transition is from impurity states to either of the valence or the conduction band, the detectors are extrinsic type.

2.5 Infrared Imaging

Infrared imaging systems are classified as mechanical scanning systems and starring array systems. A mechanical scanner uses one or more moving mirrors for sampling the object plane sequentially in a row-wise manner and directing these onto the detector. The advantage is that only a line array of detectors may be sufficient. The disadvantages are that high precision and expensive opto-mechanical parts should be used, and the detector response time has to be short. [10]

Detector arrays, which are fabricated as a matrix consisting of a large number of detectors, are called staring arrays. The advantage is that moving mechanical parts are not needed and the detector can be slow. The disadvantage is that the detector array is more complex to fabricate. Electronic circuitry needs to multiplex the detector signals to one or more outputs. The output from the circuit can be either in analog or digital form. In the analog case, analog to digital conversion is usually done externally. The electronic component multiplexing or reading out the signals from the detector elements is called ROIC or multiplexer. In hybrid type of FPAs the detector chip is flip-chip bonded to the ROIC chip. In this process metal bumps are put onto contact holes, one per pixel, of both the detector chip and the ROIC. The two chips are first aligned to each other by using special equipment. Then the chips are put in contact by applying heat and mechanical force. The two chips become electrically connected to each other via the metal bumps after this process. Usually indium is used for the bumps. [10]

There are two basic types of readout devices such as charge-coupled-device (CCD) and complementary metal-oxide semiconductor (CMOS). In the CCD ROIC, the electrons of one detector is transferred to the next at the same row. The signal is read out and determined for each detector when the electron transfer reaches the end column. The CCD transfer process has the disadvantage of losing charges along the way.

The CMOS ROIC is generated by a series of metal-oxide-silicon field-effect transistors that provide direct access to the signal from each detector. In a CMOS ROIC, the signal from each detector is read out column by column and row by row. The ROIC used in our system is CMOS type and hybridized to detector array with indium bumps.

One of the less desirable characteristics of modern FPAs is their relative non-uniformity from detector to detector in response to temperature. This results from variations in the manufacturing process and the detector material itself. All FPA cameras generally have some type of non-uniformity correction built into the camera.

2.6 Summary

A brief introduction to infrared technology was given in this chapter. The types and working principles of infrared detectors and the infrared imaging systems were illustrated. The next chapter will discuss the ROIC used in the designed development platform.

CHAPTER 3

INDIGO SYSTEM' s ISC9806 ROIC

3.1 Introduction

A 128x128 FPA coupled to Indigo System's ISC9806 ROIC has been employed in this work. However, the developed system can be compatible with other ROICs after performing straightforward modifications. This chapter introduces Indigo System's 128x128 ROIC. This ROIC is hybridized to the detector array by indium bumps. The ISC9806 is compatible with p on n detectors such as InSb, HgCdTe and InGaAs. This ROIC is suitable for applications from portable infrared imagers to high speed scientific and industrial imaging systems.

3.2 Simplified Unit Cell Schematic

The ROIC uses a direct injection input circuit as shown in Figure 3.1. Detector current flows through the input gate transistor and charges up the integration capacitor. The anti bloom keeps the input circuit from saturating. The voltage on the integration capacitor is sampled and multiplexed to the column amplifier, which provides amplification and skimming functions. Detector bias can be controlled by applying bias to Vdet_adj pad in default mode and by configuring serial control register in command mode. [13]

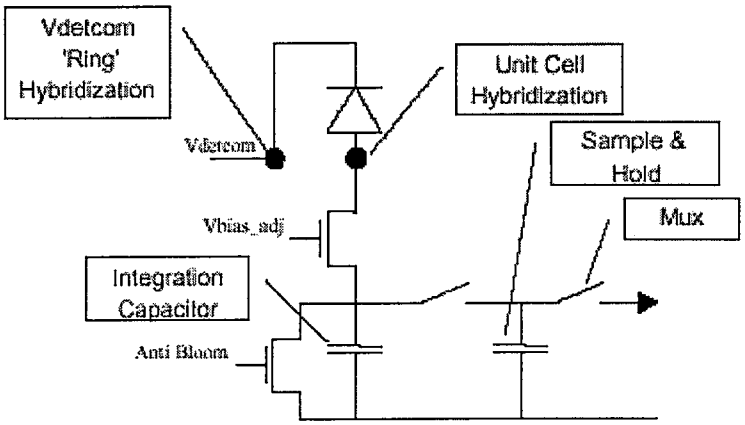


Figure 3.1 Simplified unit cell schematic [13]

3.3 Operation Modes

The ISC9806 has two operation modes; Default Mode and Command Mode. Command mode uses serial control register and has better control on the ROIC operation.

3.3.1 Default Mode

The Default Mode operates with a single output, variable gain, full window, normal scan order, no reference output, supporting NTSC and PAL. The on chip serial control register is not used in this mode, hence advanced functions of the ROIC, such as windowing, invert/revert and multiple data outputs, are not allowed. At power up, the ROIC starts its operation with a single output, full window (128x128) mode. The amplifier gain, detector bias, power and master bias current are adjusted by applying bias to the required pad of ROIC.

3.3.2 Command Mode

The command mode uses the on chip serial control register (Figure 3.2) to control the advanced features of the ROIC. The settings in this register determine the gain state, detector bias, power bias, master current bias, skimming setting, output mode, window size, window position, image transposition and test mode selection.

The block diagram for Command Mode operation is shown in Figure 3.3. There are 4 digital input signals, FSYNC, LSYNC, CLK and DATA which is used to load the Serial Control Register. VPOS, VPOSOUT, VPD, VNEG, VNEGOUT, VND, VREF, and VOUTREF are the supply and reference voltages. VDETCOM is the detector substrate connection. VOS is the skimming control voltage. TEMP is a temperature monitoring pad and OUT A-D, OUTR are the output pads.

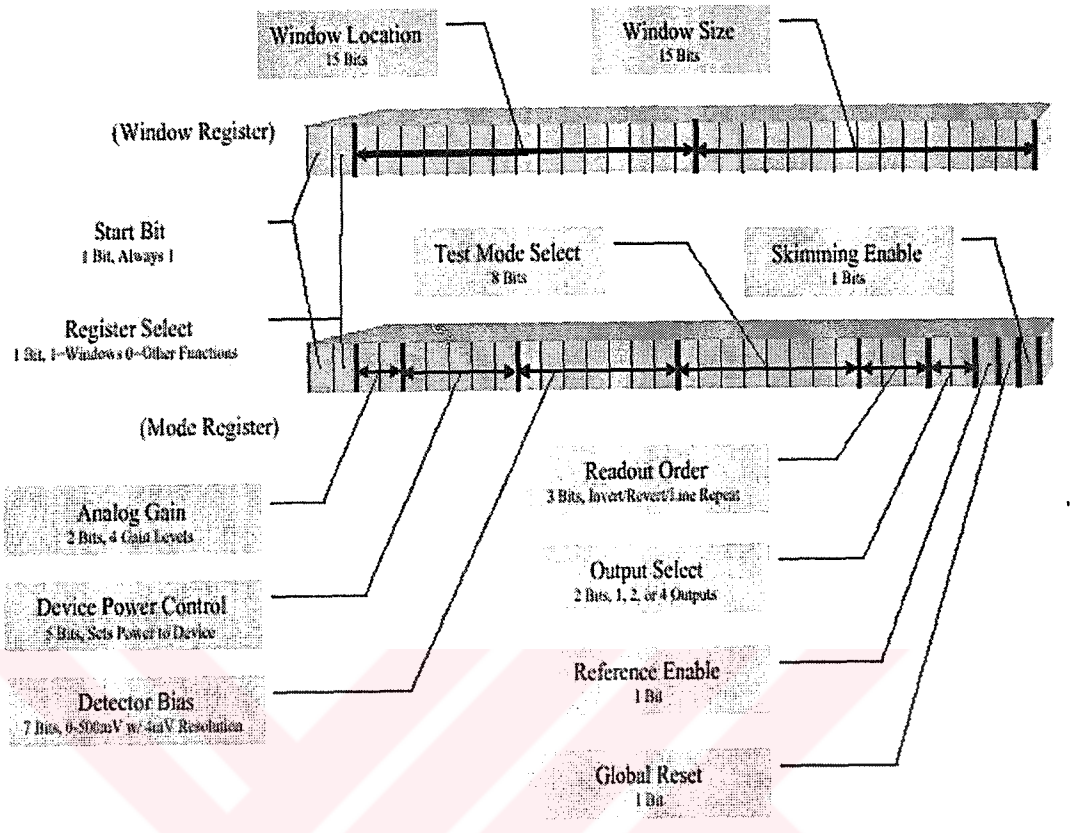


Figure 3.2 Serial Control Register [13]

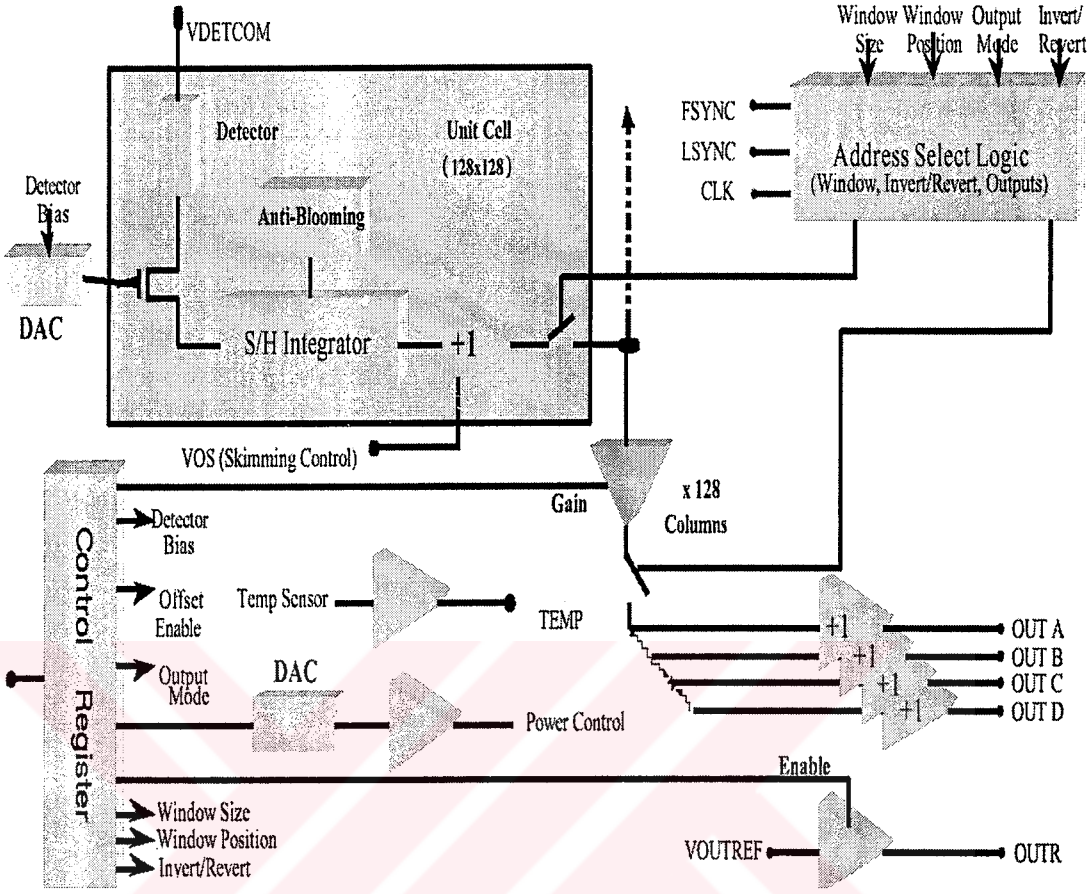


Figure 3.3 Block diagram for Command Mode operation [13]

Two types of data words can be loaded to Serial Control Register, window word or mode word. The window word contains the row and column starting address along the size of window. The mode word contains configuration parameters and adjustments for controlling the readout. The first bit of any word is the Start Bit which is always 1. The second bit is the Register Select Bit and it determines whether a window or mode word is loaded in the register. Table 3.1 illustrates the 30 bits of the Serial Control Register with the field definitions for the window word and the mode word. At power up, these registers are loaded to default values as shown in Table 3.2.

Table 3.1.a Window Data Word [13]

Register Bit	Field	Function
D29	Start Bit	Start of control word, always 1 for valid word
D28	Register Select	Determines window or mode word (1 for window word)
D27-D24	Unused Bits	Not Used
D23-D18	WAX(5-0)	Window start address for column
D17-D12	WAY(5-0)	Window start address for row
D11-D6	WSX(5-0)	Readout window size, number of columns
D5-D0	WSY(5-0)	Readout window size, number of rows

Table 3.1.b Mode Word Data [13]

Register Bit	Field	Function
D29	Start Bit	Start of control word, always 1 for valid word
D28	Register Select	Determines window or mode word (0 for mode word)
D27-D26	GC(1-0)	Sets the amplifier gain state
D25-D24	PW(1-0)	Power control adjustment
D23-D21	I(2-0)	Master current adjustment
D20-D14	DE(6-0)	Sets the detector bias
D13-D8	TS(5-0)	Test functions for factory characterisation
D7-D5	RO(2-0)	Selects readout order
D4-D3	OM(1-0)	Selects number of outputs
D2	RE	Reference output, 0 = disable, 1 = enable
D1	RST	Reset to power up conditions
D0	OE	Skimming, 0 = disable, 1 = enable

Table 3.2 Default values of Serial Control Register at power up [13]

BITS(s)	DEFAULT VALUE	COMMENT
WAX(5-0)	000000	Address 0
WAY(5-0)	000000	Address 0
WSX(5-0)	111111	Full Window
WSY(5-0)	111111	Full Window
GC(1-0)	00	Lowest Gain
PW(1-0)	00	Lowest Power
I(2-0)	100	Mid Current
DE(6-0)	0000000	Minimum Reverse Bias
TS(5-0)	000000	Normal Operation
RO(2-0)	000	Normal Mode
OM(1-0)	00	Single Output
RE	0	Reference Output Disabled
RST	N/A	
OE	1	Skimming Enabled

3.4 Integration Modes

The ISC9806 ROIC integrates all the pixels at the same time and there are two modes for this snapshot integration, Integrate-While-Read and Integrate-Then-Read Modes.

The timing pattern for Integrate-While-Read is shown in Figure 3.4. The rising edge of FSYNC (Frame Synchronization) points to the beginning of frame and FSYNC is followed a by series of LSYNC signals which controls the synchronization

of each individual line on the array. In this case, the integration time and readout time occurs simultaneously. Minimum FSYNC width, minimum LSYNC width and FSYNC to LSYNC delay should be 60 CLK cycles, 1 CLK cycle and 0.5 CLK cycle, respectively.

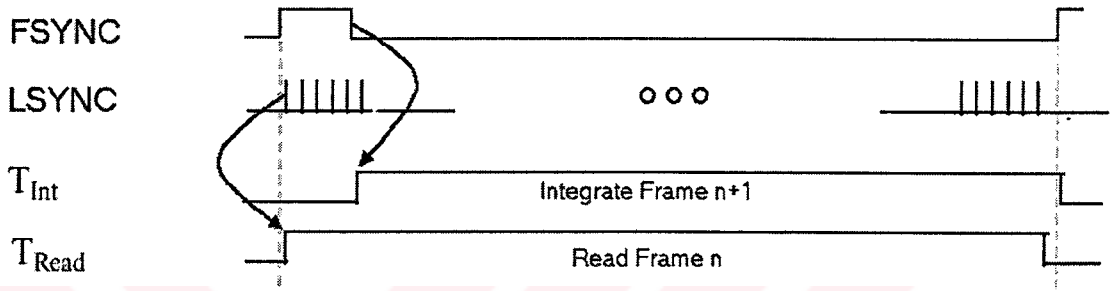


Figure 3.4 Integrate-While-Read timing diagram [13]

Figure 3.5 shows the timing pattern for Integrate-then Read Mode operation. The difference between Integrate-then Read and Integrate-While-Read is that LSYNC signals are sent during the high level of FSYNC clock, which means FSYNC clock remains high until the readout sequence. In this case, integration time occurs after the readout time, resulting in a frame time that is approximately equal to readout time plus integration time. Minimum FSYNC width depends on the size of detector array, and minimum LSYNC width and FSYNC to LSYNC delay do not change. Detailed timing diagram for the clocks is shown in Figure 3.6 and Table 3.3.

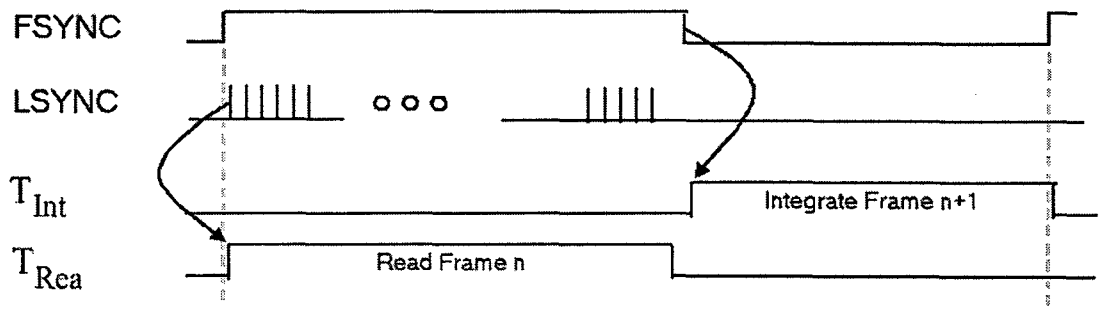


Figure 3.5 Integrate-Then-Read timing diagram [13].

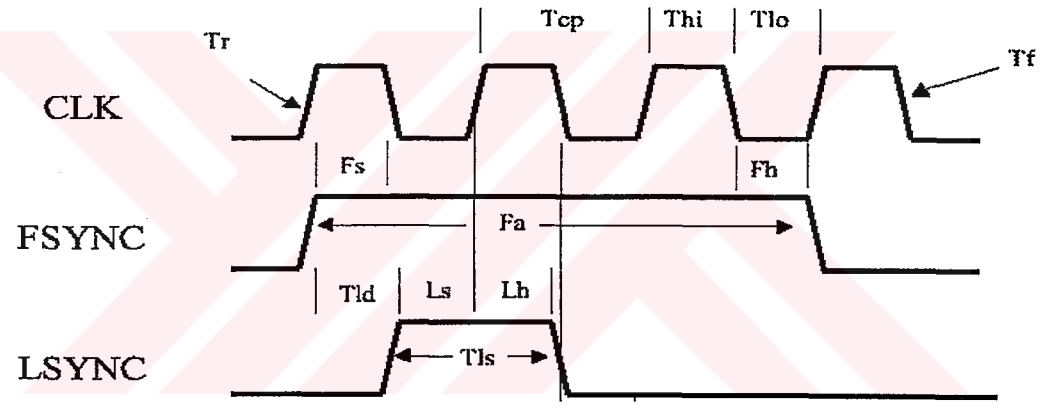


Figure 3.6 Detailed timing diagram for Integrate-Then-Read [13]

Table 3.3 Clock Timing Requirements [13]

Parameter	Min	Typ	Max	Comments
Tr	-	-	10ns	10-90% (All signals)
Tf	-	-	10ns	10-90%(All signals)
Tcp	200ns	-	-	10 MHz Pixel Rate
Thi	-	0.5*Tcp	-	Clock Duty Cycle
Tlo	-	0.5*Tcp	-	Clock Duty Cycle
Tls	-	1*Tcp	-	LSYNC width =min. 1 clock cycle
Tld	0.5*Tcp	-	-	FSYNC to LSYNC delay

3.5 Integration Time

Integration time depends on the clock frequency, frame time and the width of the FSYNC signal. The rising edge of FSYNC marks the beginning of the frame and occurs at a regular frequency frame rate. The location of the falling edge of the FSYNC clock pulse is varied to control the integration time. The integration time is approximated as [13]

$$T_{Int} = T_{Frame} - (T_{FSYNC} - 25.5 \text{ clock cycles}) \quad (3.1)$$

where

T_{int} : Integration time, measured in MC cycles

T_{Frame} : Frame time, measured in MC cycles

T_{FSYNC} : Width of FSYNC clock pulse, measured in MC cycles

3.6 Readout Timing

The readout time depends on the pixel rate, number of outputs, window size, line repeat mode, and line dead times. Line time is made up of the analog setup time plus the active pixel time plus any line dead time. The minimum analog setup time is 16 clock cycles; active pixel time is dependent on the number of outputs and window size. For our case, we have 128 columns and 1 output. Therefore, the active pixel time is 64 clock signals. If the minimum_line_dead_time is added, minimum line time can be calculated as 99 clock cycles. In our case, the line time is 129 clock cycles. The readout time for a window size of M is calculated as: [13]

Readout time: (M+2)*line time + 1 clock signal + line dead time (3.2)

For a 128x128 detector array, readout time is (130*129+1+line dead time) clock cycles.

3.7 Control Signals for ROIC and Timing Requirements

128x128 architecture is used in Command Mode and three clock signals and one serial data are sent for proper operation. The clock signals are sent to FSYNC, LSYNC and CLK pads. FSYNC marks the beginning of a frame. Since we use the ROIC in Integrate-then-Read Mode operation, minimum FSYNC width should be 16800 master clock cycles as calculated above. After 0.5 master clock cycle delay from the rising edge of FSYNC, LSYNC signal, which controls the readout synchronization of each individual row on the array, should be sent. The minimum line width of this clock is 1 master clock. The master clock in our system is 0.7202 MHz and each pixel is read on each rising and falling edge of the master clock during active pixel readout time. The serial data is used for the configuration of readout

circuit. This 30 bit serial data is loaded to Serial Control Register after 4 master clock signals from the rising edge of every FSYNC clock. After the serial data has been loaded, it is held and applied on the next rising edge of FSYNC. A new control world is only required when settings need to be changed. If a new control world is not loaded before the rising edge of FSYNC, the existing settings are applied. The detailed timing diagram is shown in Table 3.4

Table 3.4 ROIC timing signals.

Signal Definitions	Time (MC Cycles)	Time (ms)
Pixel_Clock	1	0.0013
Frame_Sync_ON_min	16800	23.32
Frame_Sync_OFF_min	1250	1
Frame_Sync_OFF_max	7700	10
Frame_Sync_Period	24500	33.3
Line_Sync_Period	129	0.1677
Imaging ADC_Clock	0.5	0.00065

The clocks sent to the ROIC and the serial data contents are shown in Figure3.7. ADC clock is sent to the ADC card, which digitizes the analog signal coming from ROIC on every rising edge. Therefore the frequency of this clock should be twice the master clock of ROIC.

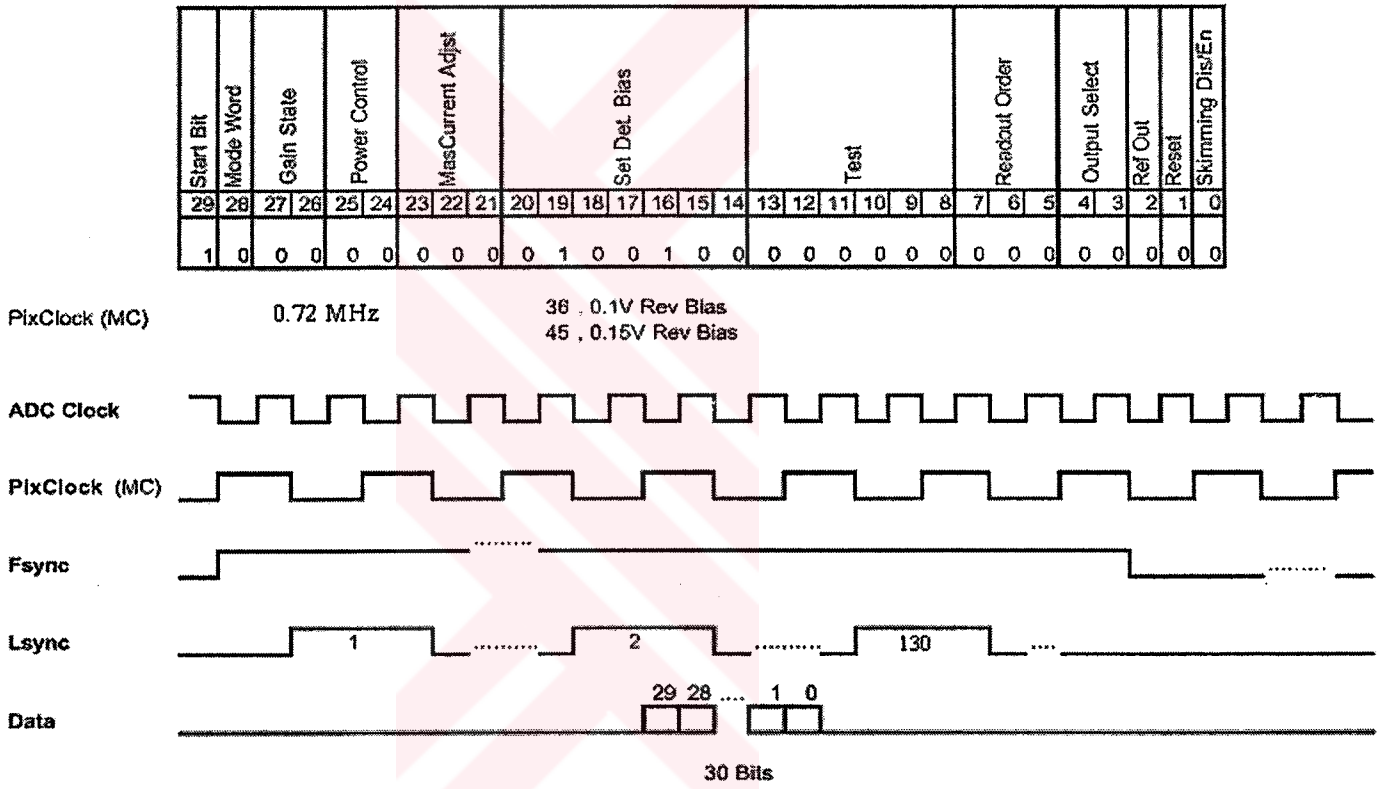


Figure 3.7 Indigo System' s ISC9806 (ROIC) timings.

3.8 Summary

This chapter described the Indigo System` s ISC9806, which is a CMOS type ROIC. The unit cell architecture of this readout was given and the operating modes were illustrated. The specific information such as the mode we used, the control of detector properties with serial data and the timings for digital clocks for proper operation were also described. The ROIC is the important part of this development system, which reads and amplifies the detector signal. The following chapter discusses the other parts of the development platform.

CHAPTER 4

THERMAL IMAGER DEVELOPMENT PLATFORM HARDWARE

4.1 Introduction

This chapter describes the hardware of the thermal imager development platform implemented in this work . It consists of a front-end electronics board having 12-bit, 10 MSPS ADC, a DSP board, and a Detector Dewar Assembly (DDA). System layout is shown in Figure 4.1.

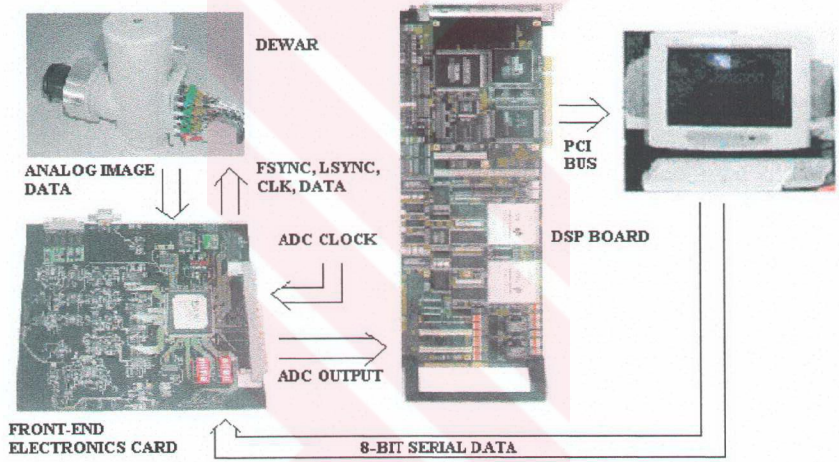


Figure 4.1 System layout of thermal imager development platform

4.2 DSP Board

The DSP Board (Blue Wave Systems PCI/C6600) shown in Figure 4.2 is PCI Bus based and has two TMS320C6701 floating point Digital Signal Processors on it. These 167 MHz processors can perform 1000 Million floating-point operations per second. The board communicates with external boards via I/O module site or PMC module site using 32 bit data busses. Each processor can access to 1Mbyte (256Kx32) flash memory, 16Mbyte (4M x 32) SDRAM and 1Mbyte (256K x 32) SRAM via local busses and shared busses. The layout of the DSP board is shown in Figure 4.3.

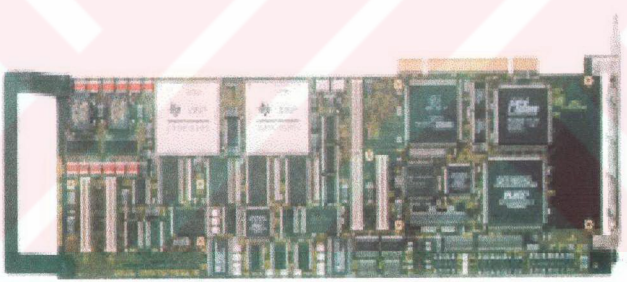


Figure 4.2 PCI/C6600 DSP board [14]

As shown in Figure 4.3, each processor accesses the memory units and I/O module site via EMIF (External Memory Interface). The board can communicate with the PC via PCI Bus and the processors are connected to PCI Bus via HPI (Host Port Interface).

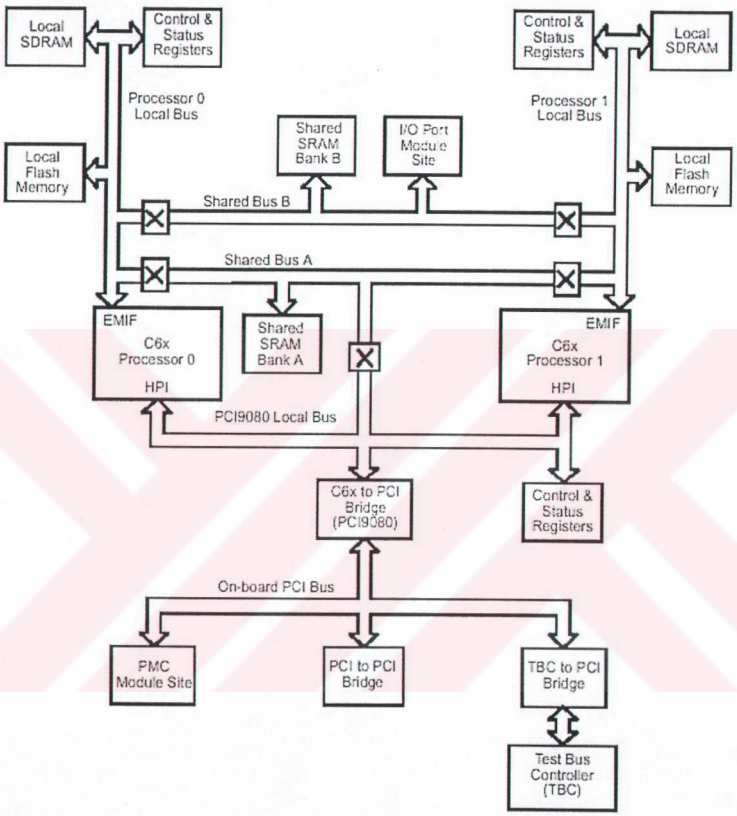


Figure 4.3 PCI/C6600 Block Diagram [14]

4.2.1 On-Board Memory

Each C6x includes two 64-Kbyte blocks of on-chip memory, 32-bit wide internal data RAM and 256-bit wide internal program RAM. The C6x DSPs can each be provided with one or two 16-Mbyte banks of SDRAM (Synchronous Dynamic RAM), which provides up to 32-M byte per processor for bulk data or program storage. The SDRAMs can also be accessed by the Host via HPI. Another memory unit is the Shared SRAM which increases the data transfer rates between the processors. Flash memories are dedicated to embedded code storage. The Blue Wave System's bootcode is written to the address range extending from 0140 0000h to 0141 FFFFh and from 014E 0000h to 014F FFFFh in the processor' s memory map. Other memory addresses in the Flash memory are available for user application codes.

4.2.2 On-Board Buses

PCI/C6600 contains six on-board busses. Processor0 Local Bus and Processor1 Local Bus have access to Flash memory, SDRAM and Control and Status registers. The Shared Bus A provides access to Shared SRAM A from each C6x via its EMIF and the PLX PCI9080 bridge device. The Shared Bus B provides access to Shared SRAM B from each C6x via its EMIF and the I/O Port Module site. The PCI9080 Local Bus links the PLX PCI9080 device to the HPI and Shared SRAM Bank A on Shared Bus A. On-board PCI bus provides data transfers between the host and each DSP's Host Port Interface. The data transfers between Host and Shared SRAM A, Host and the PMC site, PMC site and each DSP's HPI, and PMC site and Shared SRAM A can also be performed by on-board PCI bus.

4.2.3 I/O Port Module Site

This site provides connection to the I/O Modules. This provides two PMC type 64-way connectors, IOJ1 and IOJ2, which enable parallel and serial data transfer. McBSP 0 of each processor is directly connected to I/O Port Module site and provides serial data communication between the processor and I/O Port Module. The pin orientation of the connectors in I/O Port Module Site is given in Table 4.1. The IO_A[], IO_D[] and IO_BE[] signals are the buffered versions of the C6's EA[], ED[] and BE[] signals. The I/O module address bus is numbered 19..0, whereas the C6x address bus is numbered 21..2, so IO_A [0] corresponds to EA [2]. IO_RD#, IO_WR# and IO_WR/RD are gated and buffered versions of the C6's /ARE, /AWE and /AOE signals. They have the same meaning and timing (with the addition of buffer delays) as the C6's /ARE, /AWE and /AOE signals.

Table 4.1 DSP Board's I/O Module Site Pinouts [14]

Pin	Signal	Pin	Signal
1	+5V	2	+3.3V
3	IO_CLK	4	IO_RST#
5	IO_REQ	6	IO_GNT
7	GND	8	GND
9	IO_WR	10	IO_RD
11	IO_RDY#	12	IO_TERM
13	IO_WR/RD	14	IO_PRESENT
15	GND	16	GND
17	A0	18	A1
19	A2	20	A3
21	A4	22	A5
23	GND	24	GND
25	A6	26	A7
27	A8	28	A9
29	A10	30	A11
31	GND	32	GND
33	A12	34	A13
35	A14	36	A15
37	A16	38	A17
39	GND	40	GND
41	A18	42	A19
43	BE0	44	BE1
45	BE2	46	BE3
47	GND	48	GND
49	SP0_CLKR*	50	SP0_CLKX*
51	SP0_DR*	52	SP0_DX*
53	SP0_FSR*	54	SP0_FSX*
55	GND	56	GND
57	SP0_CLKS*	58	+12V
59	-	60	-
61	-	62	-
63	+3.3V	64	+5V

IOJ1

Pin	Signal	Pin	Signal
1	+5V	2	+3.3V
3	IO_INTUP#	4	IO_INTDW#
5	D0	6	D1
7	GND	8	GND
9	D2	10	D3
11	D4	12	D5
13	D6	14	D7
15	GND	16	GND
17	D8	18	D9
19	D10	20	D11
21	D12	22	D13
23	GND	24	GND
25	D14	26	D15
27	D16	28	D17
29	D18	30	D19
31	GND	32	GND
33	D20	34	D21
35	D22	36	D23
37	D24	38	D25
39	GND	40	GND
41	D26	42	D27
43	D28	44	D29
45	D30	46	D31
47	GND	48	GND
49	SP1_CLKR*	50	SP1_CLKX*
51	SP1_DR*	52	SP1_DX*
53	SP1_FSR*	54	SP1_FSX*
55	GND	56	GND
57	SP1_CLKS*	58	-12V
59	-	60	-
61	-	62	-
63	+3.3V	64	+5V

IOJ2

4.3 I/O Module Extender Board

In order to provide data communication with external devices, an I/O Module or a custom extender board is necessary. For this purpose, an extender board is designed and implemented in this work. The board is shown in Figure 4.4.

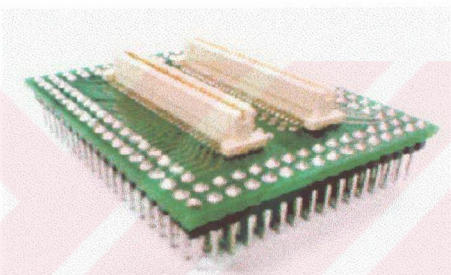


Figure 4.4 I/O module extender board

4.4 Front-end Electronics Card

The analog signal coming from the ROIC should be digitized by analog to digital converters. A front-end electronics board performing this function was designed and implemented by F. Işık [15]. This card provides the interface between detector readout and the DSP board. The card has four 12-bit 10 MSPS ADC chips and a Xilinx XC4036EX-HQ304 Field Programmable Gate Array chip. The board is shown in Figure 4.5. The front-end electronics system has analog and digital parts.

The analog section performs low-noise readout bias and reference generation, detector signal unity gain buffering, digital to analog converter (DAC) controlled global gain and offset level generation, clamping, differential ADC driving, 12-bit video digitization, DAC controlled gain offset and readout temperature monitoring. The digital section implements the functions of sampling and multiplexing of digital data, PC communication via RS232 with HPVEE control program for user interface, DAC control for global gain/offset levels and readout biases, serial ADC control for pot-controlled settings, and integration time setting. The block diagram of the board is given in Figure 4.6. The Xilinx XC4036EX-HQ304 Field Programmable Gate Array generates FSYNC, LSYNC and DATA signals synchronized to the ADC clock. The ADC clock is provided externally. However, it can also be generated internally. All of the configuration data for generating digital signals is stored in a serial EEPROM and downloaded when power is on. The analog circuits are fed with 15V, -15V, 5V, -5V supplies, and the digital circuits are biased with +5V supply.

The 12-bit output of the front-end electronics board is fed to the parallel port of the DSP board for capturing the raw image data. In our system, the front-end electronics card samples the analog signal at every 712 nanosecond, which requires 1.404 MHz ADC clock. The ADC clock can be varied from 1.4 MHz to 2.25 MHz. This limitation is due to the architecture of the front-end electronics card and to the sampling frequency of the DSP board. The data coming from ROIC reaches the ADC chip with a delay of 220 nanosecond due to the delays of the analog circuits, which corresponds a maximum ADC clock of 2.25 MHz. The lower limit can be varied according to the data capture program, which will be explained in the following sections. The timing for ADC conversion with respect to pixel clock and readout output is shown in Figure 4.7. The first real image data is the 10^{th} converted data after the rising edge of pixel clock.

The front-end electronics board communicates with PC via RS232 port for user interface. The PC sends 8-bit serial data and the FPGA chip combines every four 8-bit serial data to generate the 30-bit DATA command. The user interface program is written with HPVEE and provides the control of the detector gain and bias, power, master current, VDETCOM bias, video offset, video gain, detector skimming and the number of ROIC outputs. It also monitors the detector temperature. The user interface program display on PC monitor is shown in Figure 4.8.

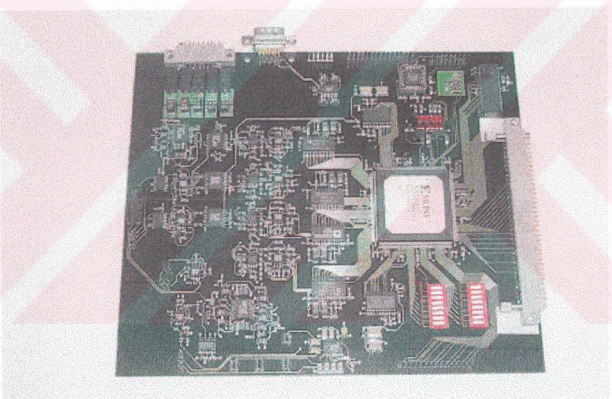


Figure 4.5 The front-end electronics board [15]

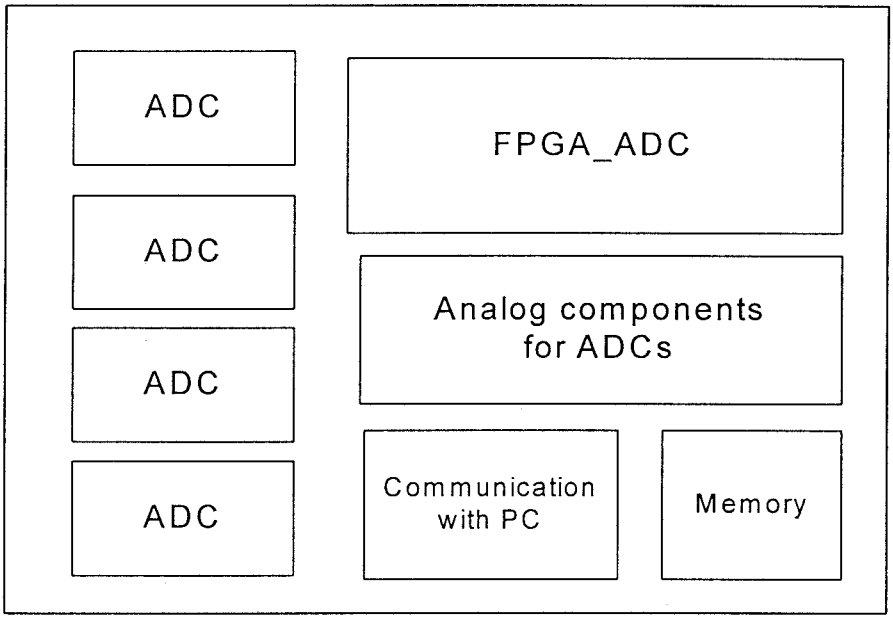


Figure 4.6 Block diagram of front-end electronics circuit [15]

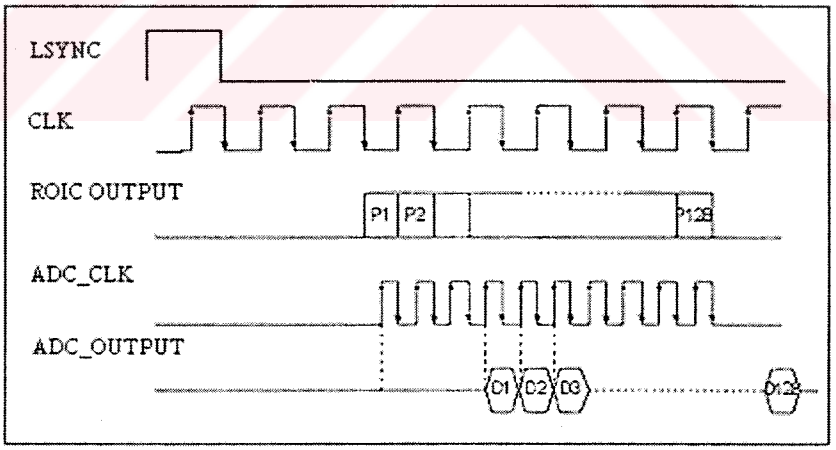


Figure 4.7 Output of ADC [15]

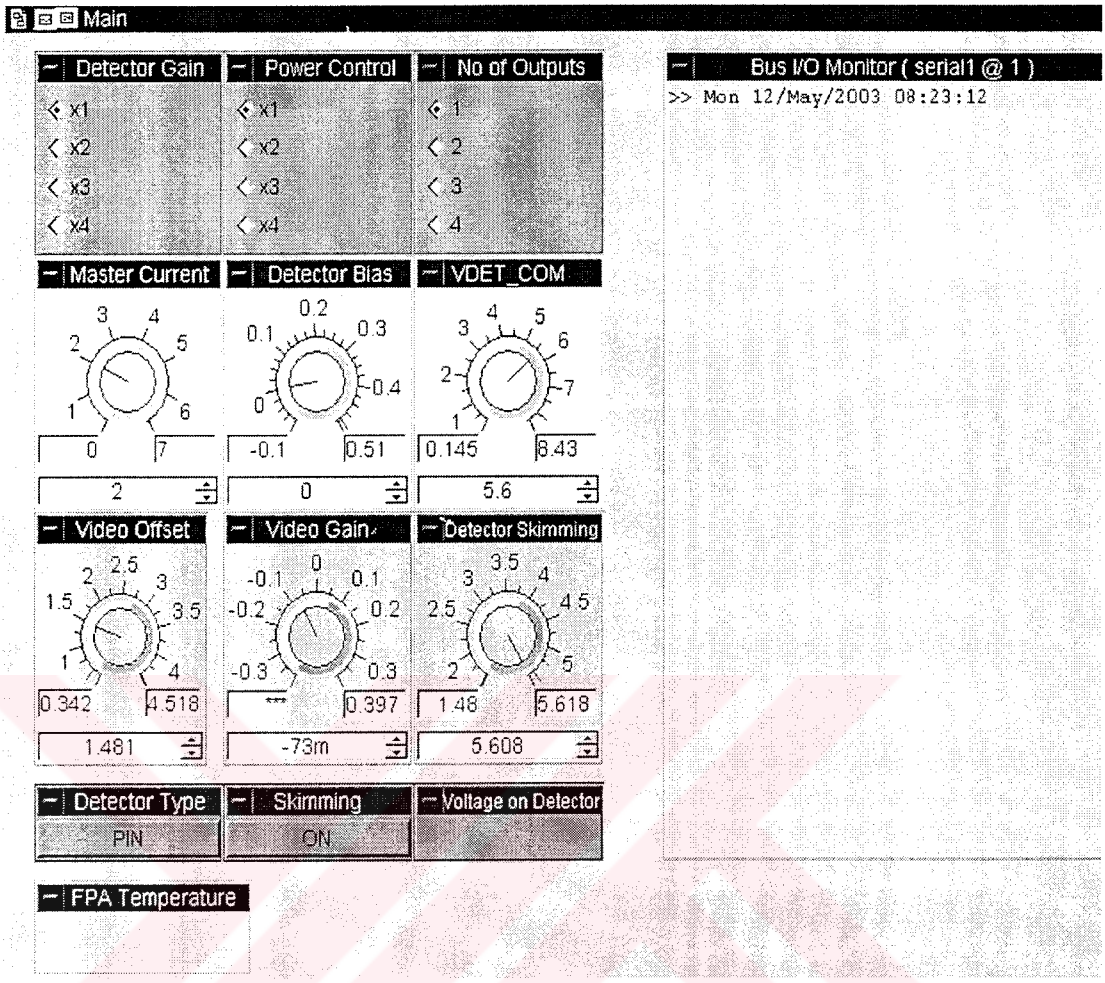


Figure 4.8. The user interface program [15]

4.5 Detector Dewar Assembly

DDA shown in Figure 4.9 consists of a pour filled liquid nitrogen dewar, 50mm f2/3 3-5 μm germanium lens and sensor interface connections. The IR FPA hybridized to ISC9806 is placed on 84 pin LCC and clamped to the cold finger using indium foil in between the ceramic carrier and the cold finger.

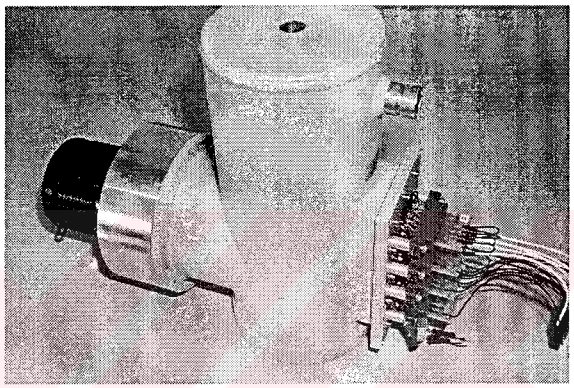


Figure 4.9 Detector Dewar Assembly (DDA)

4.6 Summary

The hardware of the designed development platform was illustrated in this chapter. The architecture and the properties of the DSP board and the front-end electronics card were given. The limitations for the operating conditions of this system were also expressed. The next chapter will discuss the method adopted to synchronize the DSP board and front-end electronics card and capture the image data at the output of ADCs on the front-end electronics card.

CHAPTER 5

DATA CAPTURE AND ROIC CONTROL

5.1 Introduction

The analog signal coming from each pixel in the detector array is digitized with the ADC chips on the front-end electronics card, and the DSP board captures the 12-bit digital data at the output of this card with its parallel port, EMIF, to form the processed real time image. The DSP board uses interrupts to capture the real time image data. Since the DSP board has only one external interrupt, the transmitter interrupts generated by the serial port of the DSP, McBSP, are employed to provide synchronization through feeding the FSYNC and LSYNC signals to two McBSPs of the DSP. This chapter presents the approach adopted to implement the above functions using the serial and parallel ports of the DSP. The DSP can also be used to generate the digital clocks and send the serial data for ROIC. If the DSP is used to implement these functions, the required period for the digital clocks can not be generated and data loss occurs while capturing the real image data. Therefore the front-end electronics card is used to generate digital clocks and send serial data to ROIC. Before presenting our approach for generating real time processed thermal images, Multi-channel Buffered Serial Port (McBSP) and the EMIF of the DSP, the method of generating digital clocks, and sending serial data and capturing data asynchronously with the DSP will be introduced in the following sections.

5.2 Multi-channel Buffered Serial Port

Multi-channel Serial Ports can be used for serial data transmission and clock generation. McBSP uses a data path and control path for these operations. Data path connects the device to outer devices and McBSP communicates with interfacing devices via the ports defined in Table 5.1. Control path is used for synchronization purposes such as sending and receiving interrupts.

The internal structure of the related DSP section is shown in Figure 5.1. The serial data transmitted or received is written to or read from the required registers by CPU or DMA of the processor. The processor uses 32 bit-wide control registers for configuring the McBSP' s registers.

Table 5.1. McBSP Interface Signals [16]

Pin	I/O/Z	Description
CLKR	I/O/Z	Receive clock
CLKX	I/O/Z	Transmit clock
CLKS	I	External clock
DR	I	Received serial data
DX	O/Z	Transmitted serial data
FSR	I/O/Z	Receive frame synchronization
FSX	I/O/Z	Transmit frame synchronization

Note: I = Input, O = Output, Z = High Impedance

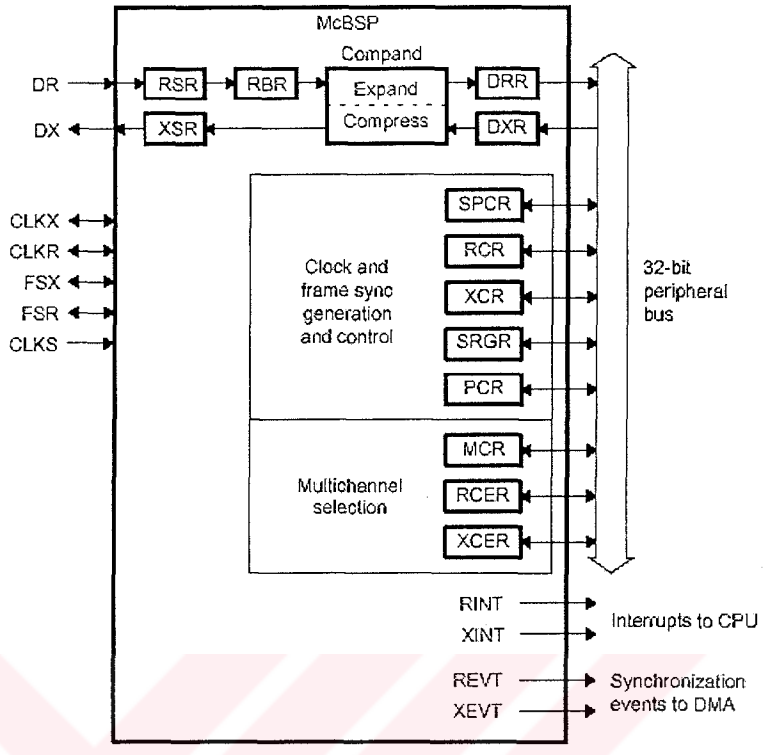


Figure 5.1 McBSP Block Diagram [16]

The transmission through the McBSP occurs in a double-buffered way. The data is first written to DXR (Data Transmit Register) by CPU or by DMA. The contents of DXR are then copied to XSR (Transmit Shift Register), if there is no data in the XSR or the last bit of data is shifted out from DX of McBSP. The transfer starts when the transmit frame sync (FSX) is detected. One bit of data is transmitted on every transmit clock (CLKX). The transmit clocks and frame sync signals are generated by the Sample Rate Generator Register (SRGR). The CPU clock or an external clock via CLKS input can generate the transmit clock. Similarly FSX can also be generated internally or an external signal can be used.

The ready state for transmission can be synchronized by polling XRDY bit or by the interrupts to the CPU. If XRDY bit in the SPCR (Serial Port Configuration Register) is 1, DXR contents have been copied to the XSR and DXR is ready to be loaded with new data. When CPU loads new data, XRDY bit becomes 0. When this data is copied from the DXR to the XSR, the XRDY bit turns from 0 to 1 again. XRDY also drives the transmit interrupt to the CPU. Interrupt is sent to CPU on every serial element by tracking the XRDY bit.

The registers in McBSP are described in Table 5.2. These registers are initialized and configured for the specific purposes. The initialization procedure is described in [17]. The program segment for McBSP configuration is given in Appendix B.

Table 5.2 McBSP Registers and addresses [16]

Hex Byte Address				
McBSP 0	McBSP 1	McBSP 2§	Abbreviation	McBSP Register Name†
–	–	–	RBR	Receive buffer register
–	–	–	RSR	Receive shift register
–	–	–	XSR	Transmit shift register
018C 0000	0190 0000	01A4 0000	DRR	Data receive register
018C 0004	0190 0004	01A4 0004	DXR	Data transmit register
018C 0008	0190 0008	01A4 0008	SPCR	Serial port control register
018C 000C	0190 000C	01A4 000C	RCR	Receive control register
018C 0010	0190 0010	01A4 0010	XCR	Transmit control register
018C 0014	0190 0014	01A4 0014	SRGR	Sample rate generator register
018C 0018	0190 0018	01A4 0018	MCR	Multichannel control register
018C 001C	0190 001C	01A4 001C	RCER	Receive channel enable register
018C 0020	0190 0020	01A4 0020	XCER	Transmit channel enable register
018C 0024	0190 0024	01A4 0024	PCR	Pin control register

5.3 EMIF

EMIF is the section of the DSP providing interface between the external memories and the internal units of the C6000. These memory devices can be SRAMs, SDRAMs, and asynchronous devices including asynchronous SRAM, ROM and FIFOs.

5.3.1 EMIF Signal Descriptions

Block diagram of the external memory interface and signal descriptions are described in Figure 5.2 and Table 5.3 for asynchronous interface.

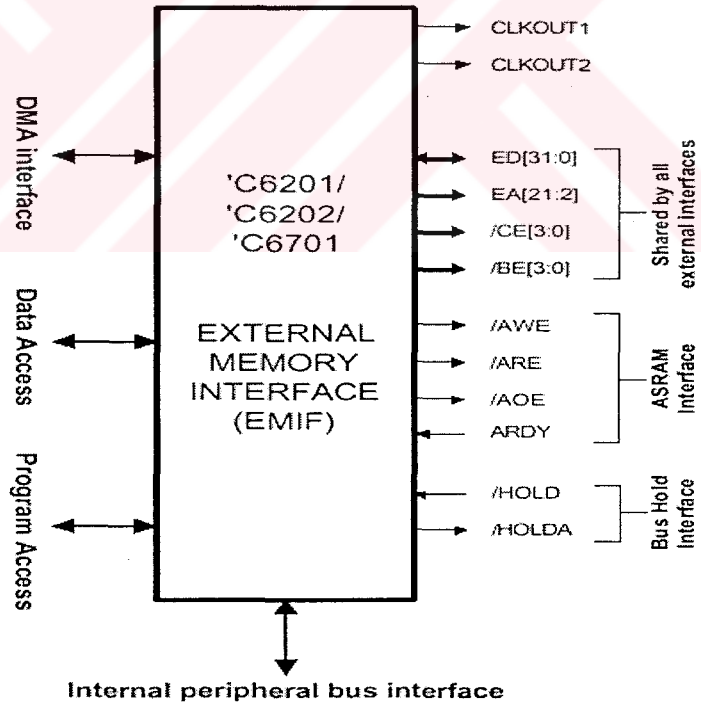


Figure 5.2 Block diagram of EMIF for asynchronous interface [16]

Table 5.3 Signal descriptions of EMIF [16]

Pin	(I/O/Z)	Description
CLKOUT1	O	Clock output. The CPU clock rate
CLKOUT2	O	Clock output.1/2 the CPU clock rate
ED[31:0]	I/O/Z	32-bit data I/O
EA[21:2]	O/Z	External address output
CE[3:0]	O/Z	Active-low chip select
BE[3:0]	O/Z	Active-low byte enables
ARDY	I	Active-high asynchronous ready input
AOE	O/Z	Active-low output enable for asynchronous memory interface
AWE	O/Z	Active-low write strobe for asynchronous memory interface
ARE	O/Z	Active-low read strobe for asynchronous memory interface
HOLD	I	Active-low external bus hold request
HOLDA	O	Active-low external bus hold acknowledge

5.3.2 EMIF Registers

EMIF registers controls the EMIF and specifies the type of memory unit it is communicating to. These memory mapped registers are given in Table 5.4.

Table 5.4 EMIF memory mapped registers [16]

Byte Address	Name
0x01800000	EMIF global control
0x01800004	EMIF CE1 space control
0x01800008	EMIF CE0 space control
0x0180000C	Reserved
0x01800010	EMIF CE2 space control
0x01800014	EMIF CE3 space control

The bit-fields in the CE space control registers specify the memory type EMIF is interfaced to and the timing requirements for reading data from memory or ADC or writing data to memory. The CE space control register diagram is shown in Figure 5.3.

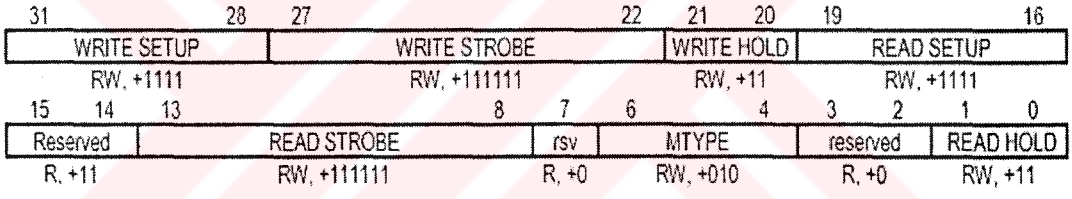


Figure 5.3 C6701 EMIF CE(0/1/2/3) Space Control Register Diagram [16]

Setup time is the time between the beginning of a memory cycle and the activation of the read (ARE) or write (AWE) strobe. Strobe time is the time between the activation and deactivation of the read or write strobe. Hold time is the time between the deactivation of the read or write strobe and the end of the cycle. The setup, strobe and hold parameters are programmed in terms of CPU clock cycles via fields in the EMIF CE space control registers. MTYPE specifies the memory type for the corresponding CE space.

5.3.3 Asynchronous Data Read with EMIF

Figure 5.4 illustrates an asynchronous read cycle with a setup/strobe/hold timing of 1/2/1 [16]. An asynchronous read proceeds as follows: at the beginning of the setup period, /CE (Chip Enable) and /AOE (Asynchronous Output Enable) become active low, /BE [3:0] and EA become valid, C6201/C6202/C6701 setup has a minimum value of 2 for the first access, and after the first access, setup has a minimum value of 1. At the beginning of a strobe period, /ARE (Asynchronous Read Enable) becomes active low. /ARE becomes active high at the beginning of a hold period. Data is sampled on the clock rising edge concurrent with the beginning of the hold period (end of the strobe period) just prior to the /ARE low-to-high transition. At the end of the hold period /AOE becomes inactive as long as another read access to the same /CE space is not scheduled for the next cycle.

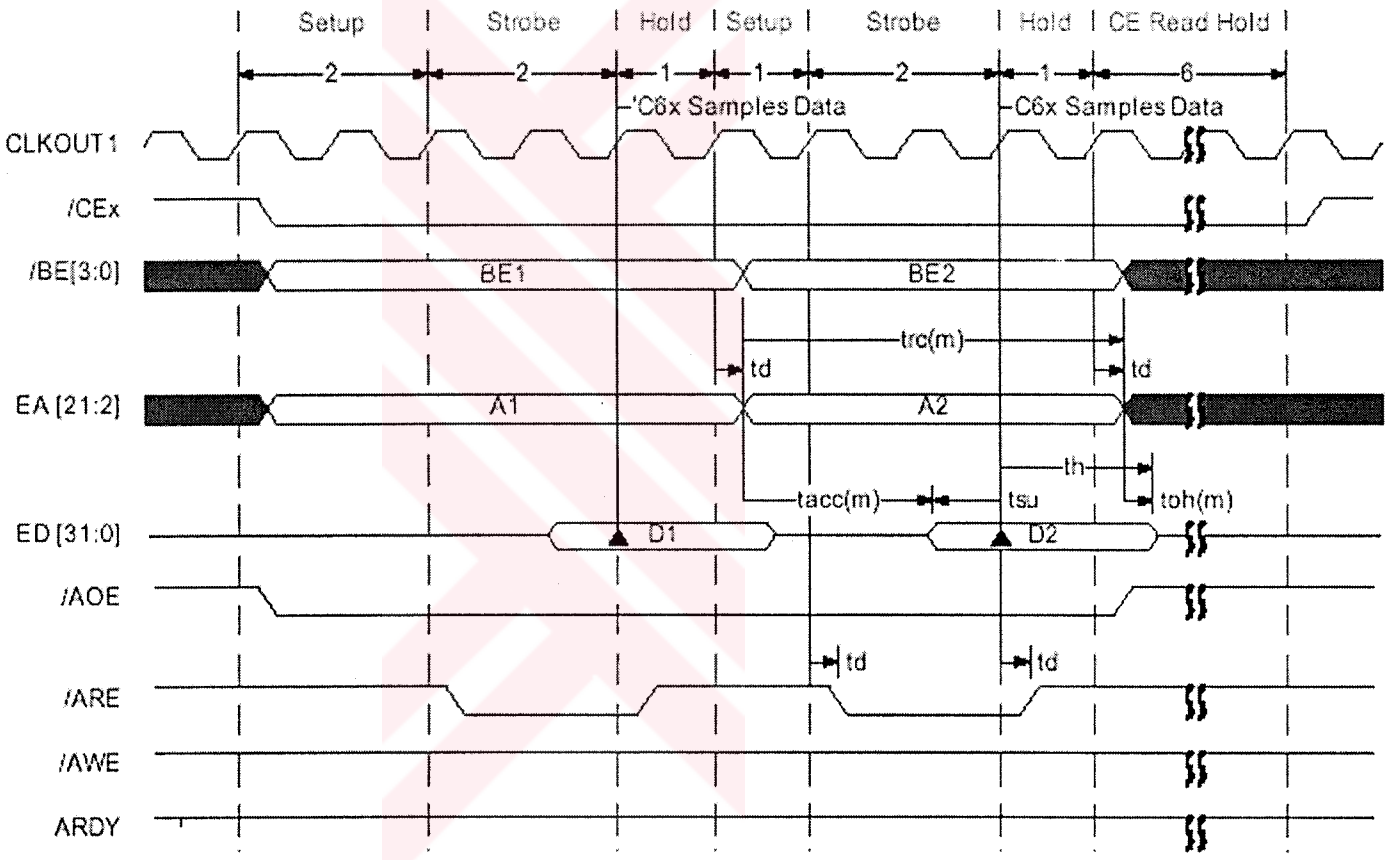


Figure 5.4 Asynchronous Read Timing Example [16]

5.4 Serial Data Transmission with DSP

In this section the possibility of serial data transmission with the DSP will be investigated. In our thermal imaging system the serial data should be sent to ROIC 4 clock cycles after the rising edge of the FSYNC signal. This data should be loaded once at every FSYNC signal and the serial control register of the ROIC is loaded with this data when the start bit of serial data is detected. Every bit needs one pixel clock cycle to be loaded into the serial control register. Therefore if the serial data is to be sent with the DSP board, the transmit clock of SRGR should be synchronized to pixel clock. The required register, CLKSM, for sample rate generator clock mode is configured such that sample rate generator clock is derived from an output clock via CLKS input pin. So the pixel clock is connected to CLKS input pin of McBSP. Since this serial data should be loaded at every FSYNC signal, the FSX signal should also be synchronized to FSYNC signal. The only solution is to use FSYNC signal as FSX signal, which will start the serial data transmission.

4 clock cycles delay between the FSYNC clock rising edge and the rising edge of serial data is needed for a valid control of ROIC. The serial data sent from DSP is 32-bit and the serial data for serial control register is 30-bit, so we achieve 2 clock cycles delay by default, if the first two MSB of serial data sent are loaded with zero. Remaining 2 clock cycles delay is achieved by configuring XDATDLY (Transmit Data Delay) bits in the XCR (Transmit Control Register). The beginning of actual data transmission with respect to the start of the frame can be delayed up to 2 data bit clocks. This configuration is clarified in Figure 5.5.

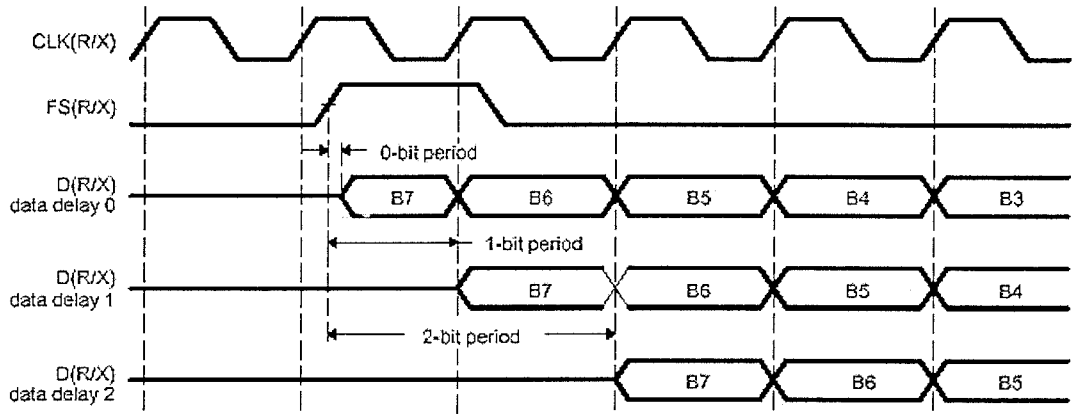


Figure 5.5 Data delay [16]

This approach is not useful. The first real image data occurs at the 10^{th} conversion after the rising edge of LSYNC, which equals 5 pixel clock cycles. The serial data is sent out in 32 pixel clock cycles in this approach. So when the transmitter interrupt occurs, the conversion for first real image data has already been done. This approach has the drawback of data loss in each line of the frame. Therefore, the serial data to ROIC is sent by the front-end electronics card.

5.5 Clock Generation with DSP

In this section the possibility of generating the clock signals with the DSP will be investigated. Our system needs ADC conversion clock, ROIC clock, FSYNC and LSYNC clocks. ADC clock is 1.404 MHz, Master ROIC clock is 0,702 MHz, FSYNC period is 24500 MC cycles, and LSYNC period is 129 MC cycles for integrate then read mode of ROIC.

The internal frame synchronization signal and sample rate generator output clock are shown in Figure 5.6. FPER field in SRGR is 12 bit, which means frame

width supports maximum 4096 sample rate generator clocks. Therefore, if the sample rate generator output clock is set to pixel clock, the maximum period of frame synchronization signal, which is to be used as FSYNC and LSYNC, is 4096 pixel clocks. So we can generate LSYNC signals with DSP; but we can not generate FSYNC signals. Therefore the front-end electronics card is used to generate both the digital clocks and the serial data that should be sent to ROIC. In the next sections, synchronization of the front-end electronics card and the DSP board for data capture will be decribed.

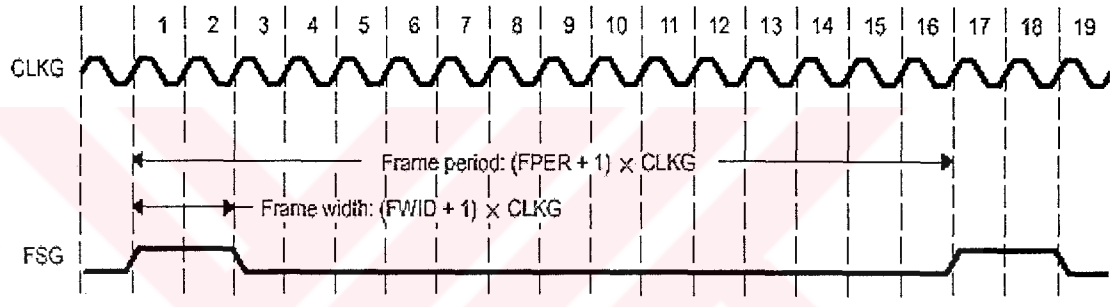


Figure 5.6 Programmable Frame Period and Width [16]

5.6 Parallel Interface with Front-End Electronics Card

Capturing parallel data from the front-end electronics card is compatible with asynchronous mode of external memory interface. The MTYPE in CE space control register is configured as 32 bit-wide asynchronous mode; since the data bus of DSP supports 32 -bit. Since asynchronous mode is used, other fields in the space control register should be programmed independently. The programmable parameters are setup, strobe and hold times. These timing parameters and the instructions in the data

capturing loop specify the time that the DSP receives samples from the front-end electronics card. Since our ADC clock is 1.404 MHz, it sends out parallel data every 712 nanosecond. Therefore to get samples at every 712 nanosecond, the CE1 space control register in our DSP is configured such that the read strobe, setup, hold fields are loaded by 60, 10, and 2, respectively. The software for EMIF configuration is given in Appendix C.

Valid data information is an important parameter in capturing the real data. Synchronization signals are needed to capture the ADC data. The transmitter interrupts sent by McBSP0 and McBSP1 provide the required synchronization. LSYNC and FSYNC signals coming from the front-end electronics circuit are used as FSX. A transmitter interrupt is sent to CPU at every LSYNC detection on McBSP0 and DSP takes 140 samples from ADC output in the interrupt service routine. The captured data is stored in an SDRAM used as buffer. Each converted data is stored in one memory location in SDRAM and the pointer address is increased by one on every storage. When FSYNC is detected on McBSP1, which means a new frame is started, the transmission interrupt is sent to CPU. In the interrupt service routine, the pointer address is refreshed to its initial value, the frame data is transferred to a second buffer, and a mailbox interrupt is sent to PC meaning that the data is ready for transmission to PC memory. The software for real time data capturing is given in Appendix D. The block diagram given below shows the complete signal flow in the system.

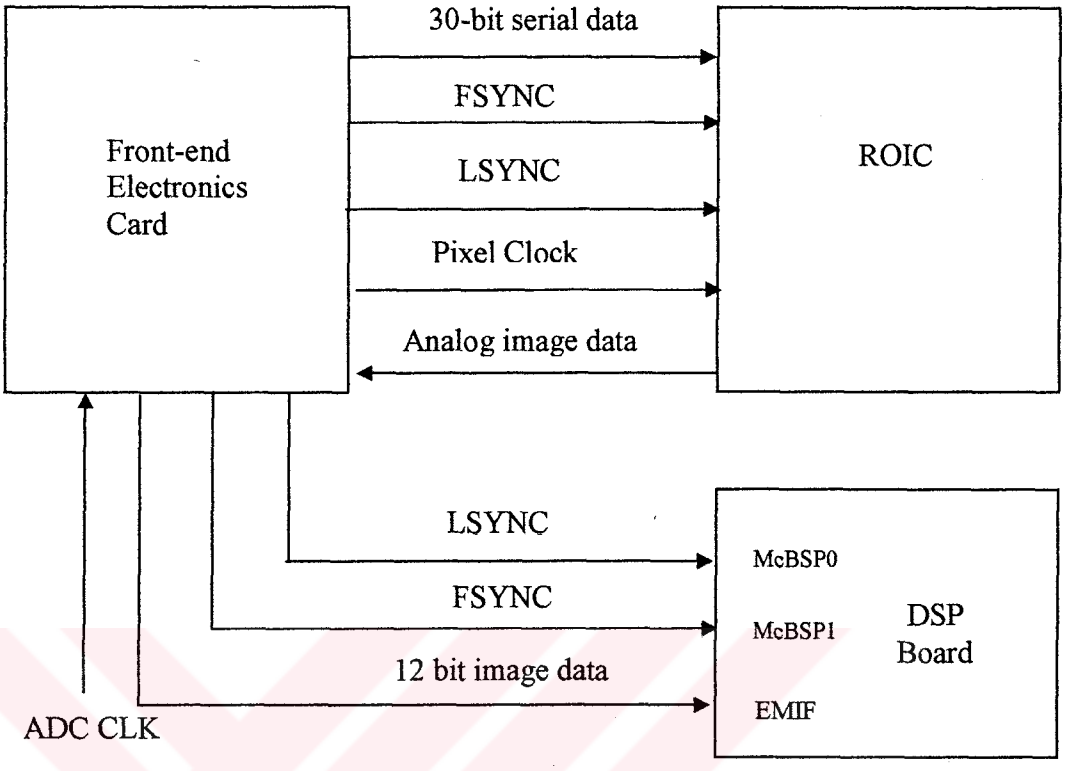


Figure 5.7 Signal flow in the system

5.7 Interrupt Missing

It has been observed that the DSP in our system may occasionally miss interrupts while detecting LSYNCs. The minimum period for DSP to detect a LSYNC signal and generate transmitter interrupt should be 32 CPU clock cycles, which corresponds to 192 nanoseconds. The period between two LSYNC signals is 167.7 microseconds and interrupt missing may still occur randomly. The DSP reads the data at the output of front-end electronics card at every transmitter interrupt generation. When the transmitter interrupt is missed, the image data on that line is also missed. This causes a shift in the image data. An image taken from an oscilloscope screen is

given in Figure 5.8. It is a snapshot taken when the DSP misses an interrupt. Asynchronous Read Enable (ARE) is a reference signal sent from DSP, which indicates that the DSP is taking samples from the front-end electronics card at every rising edge. If interrupt missing occurs during the capture of a frame data, this affects the signal processing applications due to data loss and data shift in frame buffer.

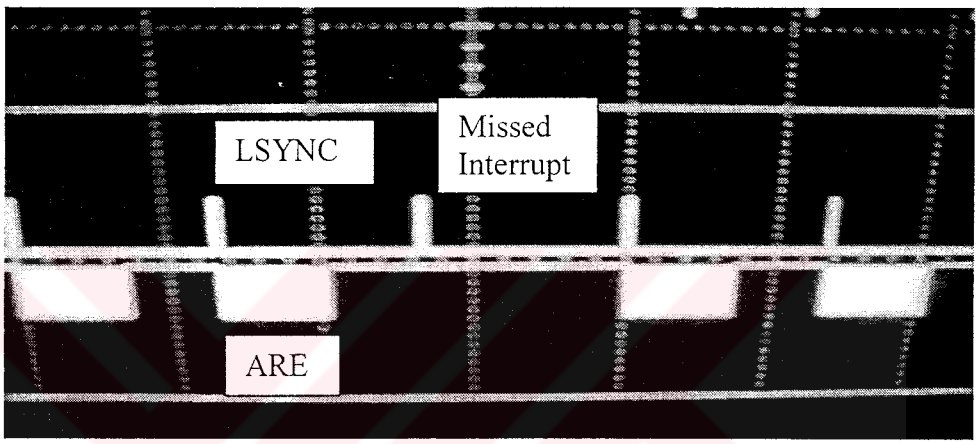


Figure 5.8 A snapshot for interrupt missing

5.8 Summary

The method for synchronizing the DSP board and the front-end electronics card was described in this chapter. The sychronization was done with the interrupts generated in serial port of the DSP and the data at the front-end electronics card was captured via the parallel port of the DSP. The following chapter will introduce the approach adopted to display thermal images on computer screen in real time, and some basic signal processing algorithms that are applied to raw image data will be described.

CHAPTER 6

IMAGE FORMATION AND PROCESSING

6.1 Introduction

Once the image data is captured by the parallel port and written into the DSP memory, the next step is to transfer the data to the PC memory. This is accomplished with the interface library of the DSP board. In this chapter, we will first explain the OpenGL platform where we have implemented the image formation software part of this thesis. We will present the image nonuniformity techniques namely one-point and two-point correction algorithms. We will also discuss the frame averaging and image filtering approaches. These approaches will be evaluated on an image of a resistance infrared heater.

6.2 Introduction to OpenGL

OpenGL is a real time graphics programming developed by Silicon Graphics Company. It is designed as a library of C functions that can be used in any C or C++ program to create 2D and 3D graphics. The programs can be compiled with Visual C/C++ compiler.

Drawing in 2D on the screen can be achieved by GLUT functions of OpenGL Utility Toolkit. GLUT must be initialized by calling the function **glutInit** inside the main function before using any GLUT functions.

After GLUT is initialized, **glutInitDisplayMode** is used to initialize the display mode of the window. The display mode of a window can be double or single buffered, RGB or indexed color table.

It is important not to display the image on the screen whilst the image is still being generated. Double buffering achieves this by allocating two memory spaces, one for the image being generated and the other for the image being displayed. In order to display the final generated image, the image generated should be transferred to the image being displayed memory with **glutSwapBuffers**. The aim is to produce smooth transitions between displaying frames in animations or real time play.

The next step is to create the actual window. Size and position of the window can be adjusted by calling **glutInitWindowSize** and **glutInitWindowPosition** functions. The window is created by the function **glutCreateWindow**. This function accepts a string as its argument. This string is used as the window's name.

After the graphics window has been created, initial attributes can be adjusted by calling an `initialize` function. In initialization, **glClearColor** sets the initial clearing color. The clearing (background) color in this work is set to black (all color values, Red, Green and Blue are set to zero). Color ranges from 0.0 (off) to 1.0 (max) and is represented by a floating-point number. Next the function **glShadeModel** is called in order to set the shading model. The shading model can be either GL_SMOOTH or GL_FLAT. When the shading model is GL_FLAT, only one color

per polygon is used, whereas when the shading model is set to GL_SMOOTH, the color of a polygon is interpolated among the colors of its vertices.

Before anything is drawn on the screen, one must call **glClear** as it clears the background. It accepts one argument that specifies the desired buffer to be cleared. **glClearColor** clears the background to the color set previously in the **init** function. The function **glColor** will set the drawing color; and it accepts Red, Green & Blue float parameters. Smooth animations and consistent motion with computer graphics can be created with a constant frame rate. This frame rate can be achieved using a timer interrupt function to generate regular time intervals, independent of the microprocessor clock speed. **GlutTimerFunc** is the function used for this purpose.

6.3 Image Formation

After a frame data is captured from the output of the front-end electronics board, and it is written to the buffer through EMIF, a mailbox interrupt is sent to the PC in order to transfer the image data from DSP memory to PC memory. When the transfer is completed, the PC sends a mailbox interrupt to the DSP, displays the data in its memory as explained above, and waits for another mailbox interrupt from DSP for transferring the new image data from DSP buffer. The software constructed for image formation is given in Appendix E.

The constructed system, shown in Fig. 6.1, was tested with a highly nonuniform 128x128 InSb detector array grown on Si substrate. The raw thermal image of a resistance heater captured by the system from this detector array is shown in Fig. 6.2.

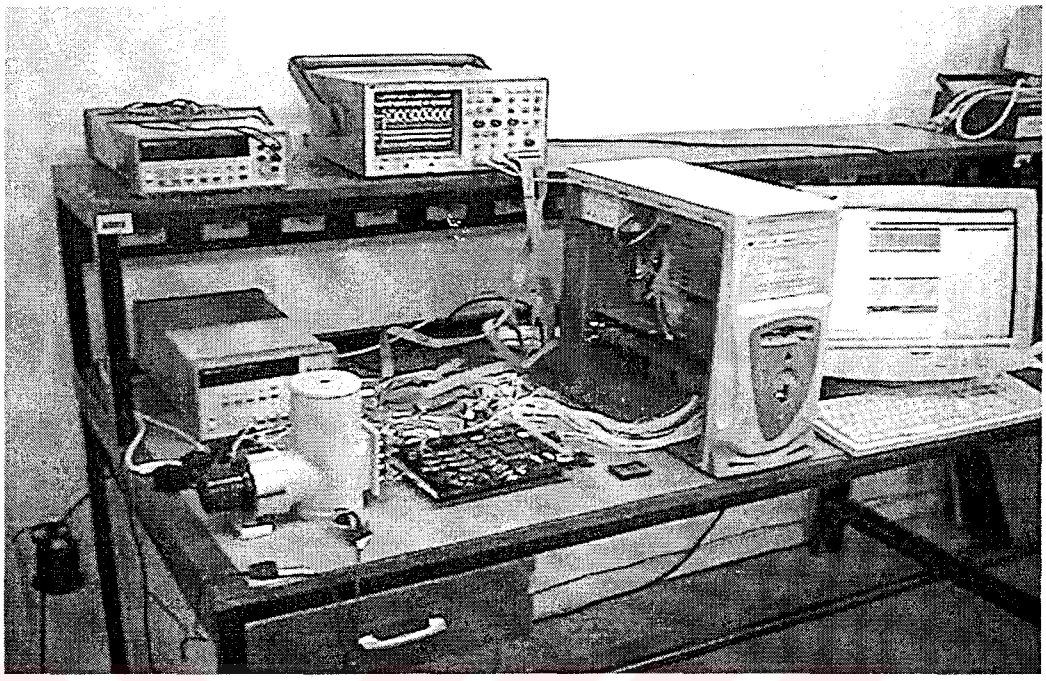


Figure 6.1 The development platform

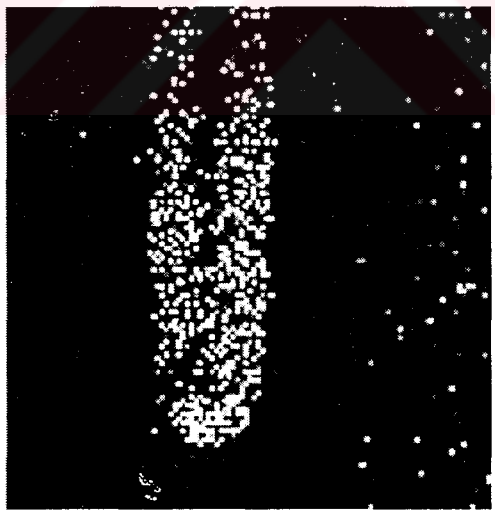


Figure 6.2 Raw thermal image of resistance heater captured by the system from a highly nonuniform 128x128 detector array.

6.4 Non-uniformity Correction

One of the undesired characteristics of an infrared detector array is the non-uniform response of the detectors on the array. In order to produce uniform response, non-uniformity correction algorithms should be used in a thermal imaging system. Many methods have been developed for image correction. These are source based two or multi-point correction algorithms, adaptive non-uniformity correction techniques and scene based techniques. [18, 19] Generally one-point and two point correction algorithms are used. One-point correction technique provides only offset correction, whereas two-point correction technique provides both gain and offset correction.

6.4.1 One-point Correction

One-point offset correction algorithm needs only one uniform flux to calculate the offset coefficients and correct the offset variations. Due to nonuniformities in detectors, bias variations on each detector can occur. If all the pixels are uniformly illuminated, the outputs will be corrected to the same value by adding the individual correction coefficients. Let us assume that Y (i, j) indicates the reference frame. We obtain Y_e (i, j) from Y (i, j) by eliminating the saturated and dead pixels. The average response of the pixels is calculated as

$$F_{av} = \frac{\sum\sum[Y_e(i,j)]}{N} \quad (6.1)$$

where i and j correspond to x and y coordinates of the image array, N is the number of pixels in Y_e (i, j) and F_{av} is the frame average. The offset coefficients, O (i, j), for each pixel in the reference frame Y (i, j) are found as,

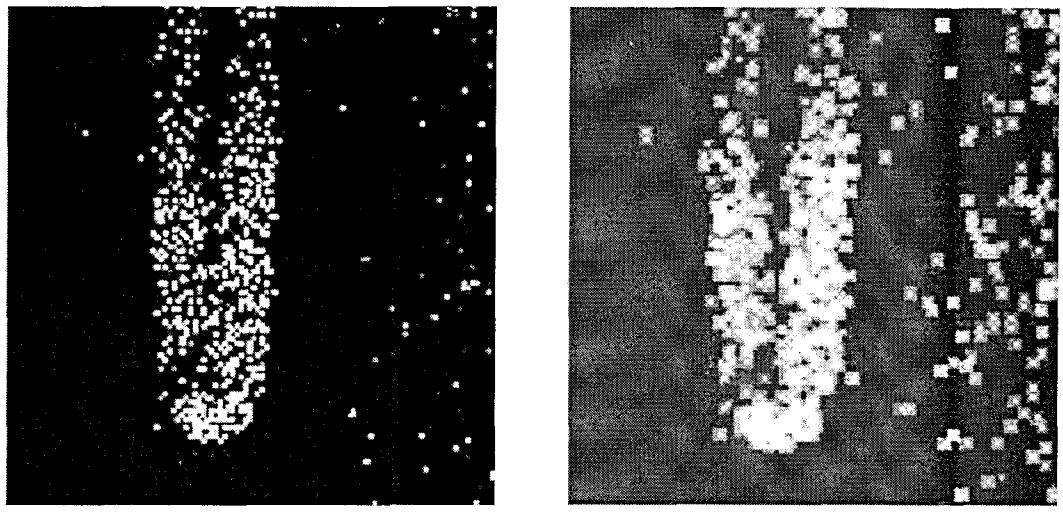
$$O(i, j) = F_{av} - Y(i, j) \quad (6.2)$$

Once the offset coefficients are found, these are used for the correction of the captured images. Therefore if we assume that $Y_r(i, j)$ is an image captured by the same detector, the corrected image, $Y_c(i, j)$, is obtained as,

$$Y_c(i, j) = Y_r(i, j) + O(i, j) \quad (6.3)$$

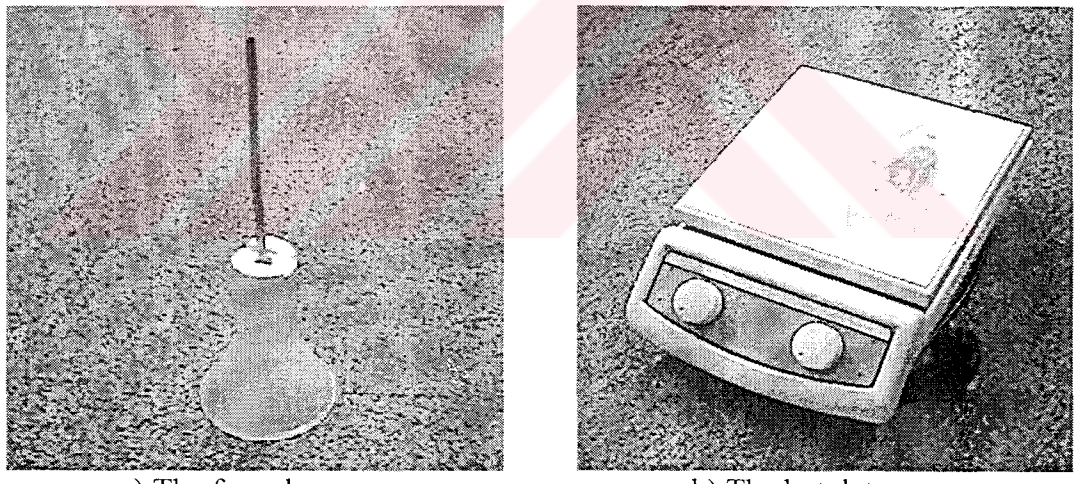
Another image enhancement method is pixel replacement. Dead and saturated pixels are bad pixels and image can be enhanced by replacing these pixels. In this algorithm when a bad pixel is detected, an average pixel value is calculated from the neighbouring pixels. Pixel replacement algorithm is applied after normalization. The saturated and dead pixels after normalization is replaced with the average response calculated from neighbouring pixels. The dead pixels are corrected with the help of the neighbouring regular pixels by using the pixel replacement. The corrected image with one-point offset correction and pixel replacement algorithm is shown in Figure 6.3, which shows a resistance heater. The software for calculation of offset coefficients and the program segment performing one-point offset correction are given in Appendix H and Appendix G, respectively.

Another experiment is done with a 30cm x 30cm hotplate and a funnel. The hotplate is heated and the funnel, shown in Figure 6.4, is put between the hotplate and the imaging system. The radiation coming from hotplate is interfered by the funnel and as a result, a black shape of the funnel is generated on a white background. Figure 6.5 shows the raw and processed images. One-point offset correction and pixel replacement algorithms are applied to the captured image and the result is shown in Figure 6.5.b.



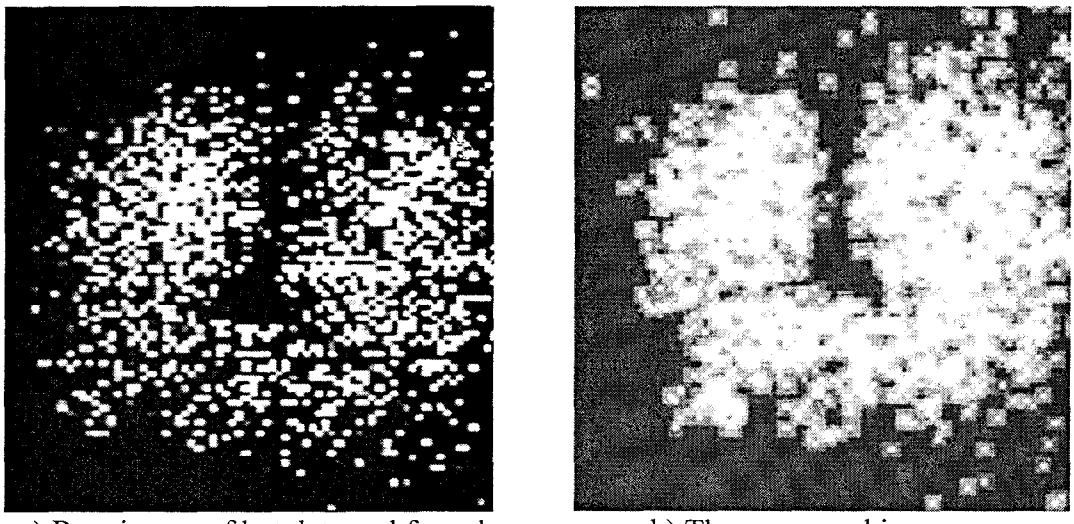
a) Raw image data b) Corrected image data

Figure 6.3 One-point offset correction algorithm with pixel replacement.



a) The funnel b) The hotplate

Figure 6.4 The image of a funnel and hotplate



a) Raw image of hotplate and funnel b) The processed image

Figure 6.5 The raw and one-point corrected image when the funnel is between the hotplate and the constructed imaging system.

6.4.2 Two-point Correction

This approach is called as two-point correction because two uniform fluxes generated by blackbodies at different temperatures are used. Gain and offset coefficients are calculated by using these fluxes. These coefficients are used to reduce the gain and offset variations and produce uniform response from each pixel. General detector response can be modeled as,

$$y_i = a_i x + b_i \qquad (6.4)$$

where y_i is the i^{th} detector output, a_i is the i^{th} detector multiplicative gain, b_i is the i^{th} detector additive offset, and x is the flux level.

The responses of the detectors in the IRFPA (Infrared Focal Plane Array) to the high and low reference fluxes are given by

$$Y_H(i, j) = a_{ij}X_H+b_{ij} \quad (6.5)$$

and

$$Y_L(i, j) = a_{ij}X_L+b_{ij} \quad (6.6)$$

where i, j, H, and L correspond to x and y coordinates in the array and high and low reference fluxes, respectively. X_H , X_L, a_{ij} and b_{ij} are the calibrated flux levels corresponding to the low and high ends of the scene temperature, the detector multiplicative gain and detector additive offset, respectively. [19]

Using the above equations, multiplicative gain factor, a_{ij}, is found as

$$a_{ij} = [\ Y_H(i, j) - Y_L(i, j)]/(X_H-X_L) \quad (6.7)$$

we can define the spatial average value, a_{mean}, as

$$a_{mean} = [Y_{Hmean}-Y_{Lmean}]/(X_H-X_L) \quad (6.8)$$

Then two-point correction factor, $a_{mean}/a_{i\,j}$, is written as,

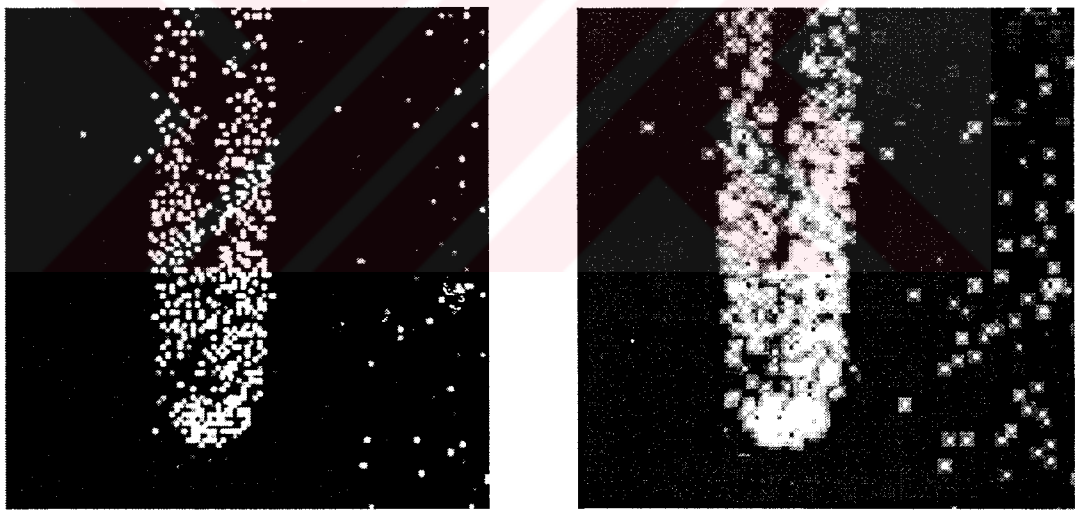
$$a_{mean}/a_{i\,j} = [Y_{Hmean}-Y_{Lmean}]/\ [Y_H(i, j) - Y_L(i, j)] \quad (6.9)$$

Given the above factor, corrected image, Y_c (i,j), can be found as,

$$Y_c(i, j)=\frac{a_{mean}}{a_{ij}}[Y_r(i, j)- Y_L(i, j)] - Y_L(i, j) \quad (6.10)$$

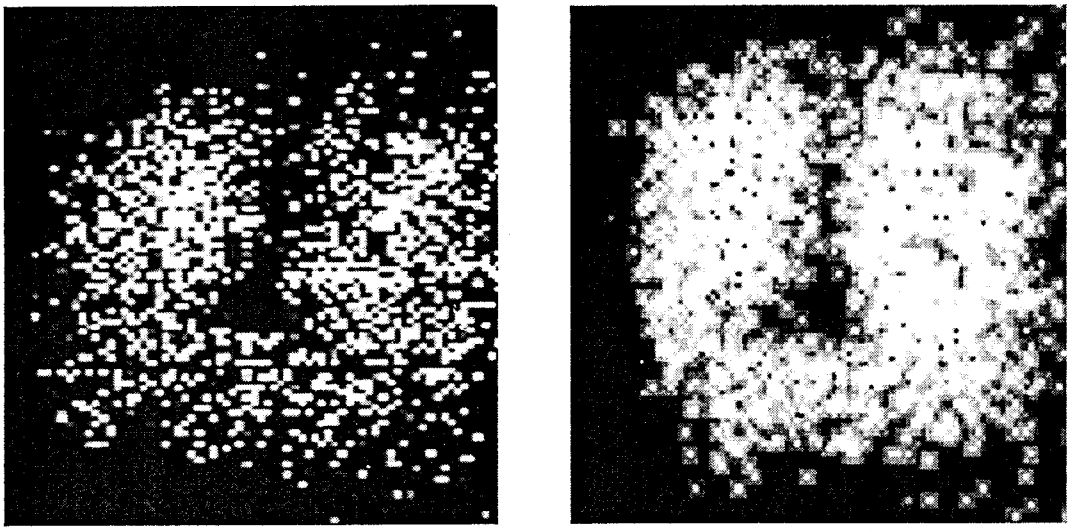
where Y_r (i, j) is the captured image.

The two-point gain correction algorithm is applied to the image data captured with the DSP board. The image data taken under two different levels of uniform flux is written to DSP' s flash memory. Gain coefficients are calculated from two different fluxes by using Equation (6.9). The calculated coefficients are multiplied with the captured image data excluding the dead and saturated pixels. The average response is calculated and this average value is assigned to the saturated and dead pixels. Figure 6.6 shows the effects of two-point gain correction and pixel replacement on the raw image data. Appendix H and Appendix G present the software for gain coefficient calculation and program segment for two-point calibration. The experiment with the hotplate and a funnel is also done by applying two-point gain correction and pixel replacement. The result is given in Figure 6.7.



a) Raw image data b) Corrected image data

Figure 6.6 Two-point gain correction algorithm with pixel replacement. It is showing a resistance heater

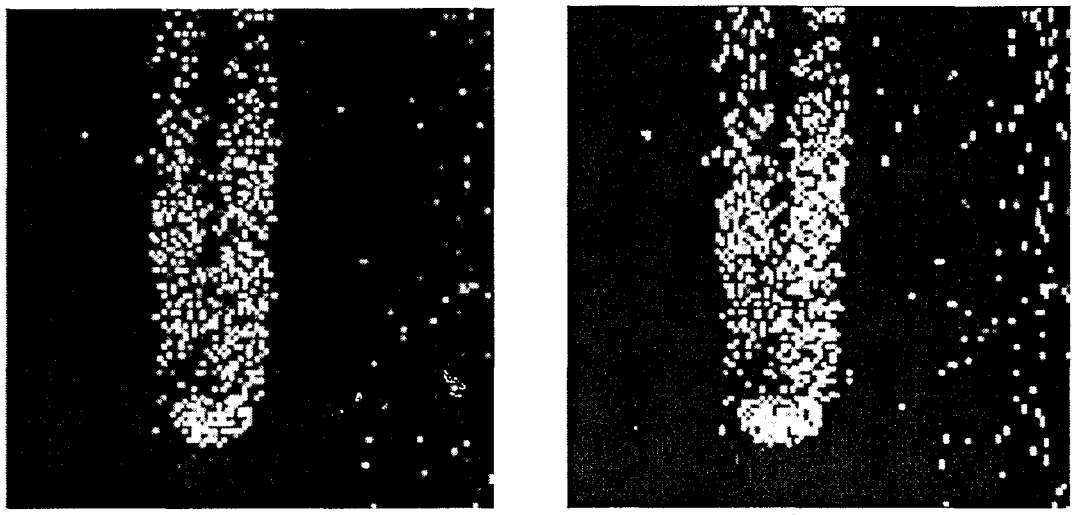


a) Raw image of hotplate and funnel b) The processed image

Figure 6.7 The raw and two-point corrected image when the funnel is between the hotplate and the constructed imaging system.

6.5 Frame Averaging and Image Filtering

Frame averaging is a method to improve the signal-to-noise ratio (SNR) of the video sequence. SNR is a measure of signal strength relative to background noise in analog and digital communications. The noise can be reduced with frame averaging. In this technique, successive frames are accumulated and their average is displayed on the screen. Two-frame averaging is applied to the two-point gain corrected image data with the system while pixel replacement algorithm is off and the result of this algorithm is shown in Figure 6.8.



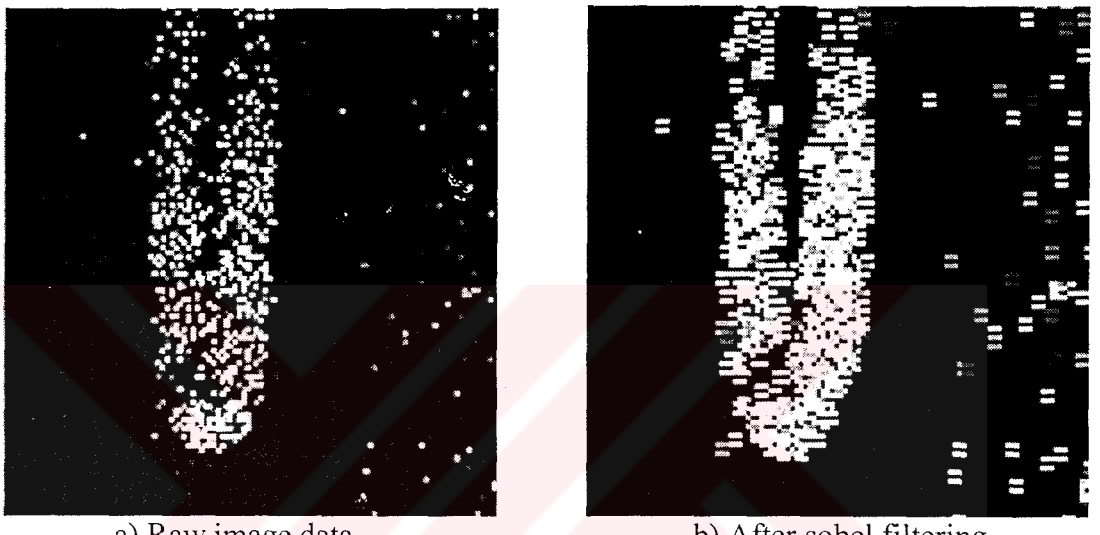
a) Normalized image b) Two-frame averaging

Figure 6.8 The effect of frame averaging algorithm after two-point gain correction. It is showing resistance heater.

Image filtering algorithms are used to enhance low quality images with some distortions. Sobel Filtering is a method for enhancing the edges in an image. It produces a dark image for the areas where the original image has smooth areas, and light areas where the original image has areas of rapidly varying intensity. A program segment has been written for taking 3x3 convolution and a Sobel filter, having a mask of M, given in Equation 6.11, to detect gradients in Y direction, is applied to captured image data after applying two-point gain correction and pixel replacement algorithm.

$$M = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad (6.11)$$

The result of this filtering program is shown in Figure 6.9, which is showing a resistance heater. Since there is no continuous region for the resistance heater image, the effect of using a Sobel edge enhancement filter is not very effective. Nevertheless, we obtain an enlarged region of pixels after Sobel filtering.



a) Raw image data b) After sobel filtering

Figure 6.9 Sobel filtering with normalization and pixel replacement

6.6 Mathematical Analysis of the Algorithms

The Root Mean Square (RMS) value of the difference between images gives a measure of how similar two images are. If the images are exactly identical, RMS value is zero, PSNR (Peak Signal to Noise Ratio) and SNR are infinity. [20] PSNR are calculated in decibels and typical values of PSNR are 20 to 40 decibels for similar

images. [21] The RMS value of difference of two images, g(x,y) and h(x,y) having NxM pixels, can be calculated as

$$\text{RMS} = \frac{1}{NxM}\sum_{x=1}^{N}\sum_{y=1}^{M}\sqrt{[g(x,y)-h(x,y)]^2} \qquad (6.11)$$

where x and y give the coordinates of the pixels in the image array, g(x,y) is the corrected image and h(x,y) is the original image. The PSNR and SNR can be calculated as

$$\text{PSNR} = 20\log_{10}(\text{B/RMS}) \qquad (6.12)$$

$$\text{SNR} = \frac{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1} g(x,y)^2}{\sum_{y=0}^{M-1}\sum_{x=0}^{N-1}(h(x,y)-g(x,y))^2} \qquad (6.13)$$

where B is the peak value for a pixel in image. [20-22]. The RMS value of the difference of these images and PSNR can be calculated in Matlab after converting pixel values to gray level (maximum value is 255) and the similarity of the images can be compared. A reference image is constructed assuming that the temperature is distributed uniformly throughout the heated resistance wire meaning that the wire radiates infrared energy uniformly through its body. By using this image as a reference, the PSNR and SNR values are calculated for each image, which is formed after applying the algorithms.

Table 6.1 Mathematical analysis of algorithms

Algorithms	Compared frame	PSNR (dB)	SNR
Two-point correction	Raw image data	10.48	51.99
	Normalized image data	12.91	89.12
One-point correction	Raw image data	10.48	51.99
	Normalized image data	11.7	77.62
Frame average	Normalized data, pixel repl. is off	10.33	66.20
	Two-frame averaged data	10.53	66.84

If we look at the mathematical results, the processed images have higher SNR and PSNR values than the raw images, which shows us that there is an improvement after applying the algorithms. Frame averaging tends to increase SNR and we have a slight increase in SNR.

6.7 Summary

This chapter illustrated the method for displaying images on computer screen in real time. The OpenGL platform was explained and image formation software was discussed. Two-point and one-point normalization techniques and frame averaging and image filtering algorithms were also described and the effects of these algorithms on raw image data were seen. A mathematical analysis was also done by calculating PSNR and SNR values of the processed images. Conclusion and future work will be given in the next chapter.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The aim of this study was to design and implement a development platform for high performance thermal imagers. The work includes the following main parts: programming TMS320C6701 DSP to perform real time thermal imaging with a DSP board integrated with a front-end electronics board and a detector/dewar assembly, applying some basic signal processing algorithms and testing the system with a 128x128 InSb focal plane array hybridized to Indigo System's ISC9806 ROIC.

The peripheral programs were written and implemented on DSP successfully. After the DSP board was integrated with the front-end electronics board, frame data was captured in real time. A graphic program was written with OpenGL toolkit for image formation. Normalization, pixel replacement, frame averaging and image filtering algorithms were implemented and applied to raw image data with the DSP board.

The system can work up to 25 frame/second with a 320x256 detector array. This limitation is due to delays of analog circuits in the front-end electronics board, which limits the ADC clock to 2.25 MHz. The system can work with a 128x128 detector array up to 100 frame/second. Instruction delay times of the signal processing

algorithms and the speed of displaying program are also effective in the maximum achievable frame rate. The codes should be optimized, and if necessary, both of the DSPs should be used to decrease the signal processing time.

In the extent of future study, the system will be enhanced by constructing high level signal processing algorithms.

REFERENCES

[1] K.Chrzanovski, "Comparison of shortwave and longwave measuring thermal imaging systems," Applied Optics, Vol. 34, pp. 2888-2897, 1995

[2] Sierra Pacific Infrared, "Understanding and Utilizing Focal Plane Arrays," available at http://x26.com/infrared/images/fpa.htm

[3] The University of Tenesse, Department of Physics and Astronomy, "Electromagnetic Spectrum," available at http://csep10.phys.utk.edu/ astr162/ lect/light/spectrum.html

[4] Sierra Pacific Infrared, "Infrared thermal imaging: theory & application," available at http://infra-red.hypermart.net/infrared_theory.htm

[5] Sierra Pacific Infrared, available at http://www.x20.org/thermal/

[6] C. P. Arthurs, "Long Linear Arrays with Time Delay Integration and Element Deselection," *Proc. of SPIE*, Vol. 3061, pp. 476-483, 1997

[7] Vincent J. Realmuto, "Thermal Anomaly- High Spatial Resolution," EDS IDS Valcolonolgy Team Data Product Document, Product # 3291, Version 3 November 4, 1995

[8] Grenn, M.W., "Performance of portable staring infrared cameras," Proceedings of the 19th Annual International Conference of the IEEE , Vol. 2 , pp. 708 –711, 30 Oct.-2 Nov. 1997

[9] Gerald C. Holst, "Testing and Evaluation of Infrared Imaging Systems," SPIE Press, 1998.

[10] Acreo, "Infrared Detector Arrays for Thermal Imaging Tutorial," available at http://www.acreo.se/acreo-rd/IMAGES/PUBLICATIONS/TUTORIALS/TUTORIALS-INFRARED-2.PDF

[11] Mahmoud Almasri, "Infrared detectors," available at http://engr.smu.edu/~almasri/infrared_detectors.htm

[12] R. B. Emmons, S. R. Hawkins, K. F. Cuff, "Infrared detectors: An overview," *Opt. Eng.,* Vol. 14, pp. 21, 1975

[13] Indigo Systems, "ISC9806 User's Guide," Version 1.3.

[14] Blue Wave Systems, "PCI/C6600 Applications Board Technical Reference Manual," Version 1.01, available at www.powerbridge.de/ download/ manual/ipc/ipc-pci-io/PCI_C6600_TM_V1.01.pdf, September 1999

[15] Fatih Isik, "Design of front-end electronics and user interface board for a compact thermal imager," M.Sc Thesis, Dept. of Electrical & Electronics Engineering, Middle East Technical University, December 2002

[16] Datasheet, Texas Instruments, Spru190c, "TMS320C6000 Peripherals Reference Guide," available at www.ti.com, April 1999

[17] Datasheet, Texas Instruments, Spra488A, "TMS320C6000 McBSP Initialization," available at www.ti.com, September 2001

[18] Maintenance Resources, "Overview of Infrared Terminolgy," available at http://www.maintenanceresources.com/ReferenceLibrary/InfraredThermography/infraredglossary.htm

[19] R. Venkateswarlu, M.H. Era, Y.H. Ganb and Y. C. Fongb, "Nonuniformity compensation for IR focal plane array sensors," *Proc. of SPIE*, Vol. 3061, pp. 915-926, 1997

[20] Applied Information Technology Division, "Peak Signal to Noise Ratio/Mean Squared Error," available at http://appliedit.arc.nasa.gov/videotech/psnr.html

[21] Todd Veldhuiz, "Grid Filters for Local Nonlinear Image Restoration," M. Sc thesis, Dept. of Systems Design Engineering, University of Waterloo, May 1998

[22] Arthur R. Weeks, Jr; "Fundamentals of Electronic Image Processing," SPIE Optical Engineering Press, 1998.

[23] Datasheet, Texas Instruments, Spru301c, "TMS320C6000 Code Composer Studio Tutorial," available at www.ti.com, February 2000

[24] Datasheet, Texas Instruments, Sprs067, "TMS320C6701 floating-point digital signal processor," available at www.ti.com, May 1998

[25] Datasheet, Texas Instruments, Spru187, "Optimizing C Compiler," available at www.ti.com, April 2001

[26] Datasheet, Texas Instruments, Spra542A, "TMS320C6000 EMIF to External Asynchronous SRAM Interface," available at www.ti.com, August 2001

[27] Jackie Neider, Tom Davis, and Mason Woo; "OpenGL Programming Guide," Addison-Wesley Publishing Company, 1993

[28] Blue Wave Systems, "PCI/C6600 DSP Utility Library Function Reference Manual," Version 1.52, September 2000

[29] Blue Wave Systems, "I/O Port Module Specification," Revision 8.0, 1999

[30] Datasheet, D.SignT, "Designing a DSP System," Version 1.0, available at http://www.dsignt.de/download/ansystem.pdf, August 1999

[31] Datasheet, National Semiconductors, "CD4049UBM/CD4049UBC Hex Inverting Buffer, CD4050BM/ CD4050BC Hex Non-Inverting Buffer," available at www.national.com/pf/CD/CD4050BM.html, March 1988

[32] Datasheet, Texas Instruments, Spra433A, "TMS320C6000 EMIF to External SDRAM/SGRAM Interface," available at www.ti.com, June 1999

[33] Datasheet, Texas Instruments, Spra477A, "TMS320C6201/6701 EVM: TMS320C6000 McBSP to Multimedia Audio Codec Interface," available at www.ti.com, August 2001

[34] Datasheet, Texas Instruments, Spra543, "TMS320C6000 EMIF to External FIFO Interface," available at www.ti.com, April 1999

[35] Datasheet, Texas Instruments, Spra568, "TMS320C6000 EMIF to External Flash Memory," available at www.ti.com, February 2002

[36] Datasheet, Texas Instruments, Spra455A, "Using the TMS320C6000 McBSP as a High Speed Communication Port," available at www.ti.com, August 2001

[37] Datasheet, Texas Instruments, Spra477A, "TMS320C6201/6701 EVM: TMS320C6000 McBSP to Multimedia Audio Codec Interface," available at www.ti.com, August 2001

[38] Datasheet, Texas Instruments, Spra489A, "TMS320C6000 McBSP to Voice Band Audio Processor (VBAP) Interface," available at www.ti.com, November 1998

[39] Datasheet, Texas Instruments, Spra491A, "TMS320C6000 McBSP as a TDM Highway," available at www.ti.com, September 2000

[40] Datasheet, Texas Instruments, Spra511A, "TMS320C6000 McBSP Interface to a ST-bus Device," available at www.ti.com, June 2002

[41] Datasheet, Texas Instruments, Spra528A, "TMS320C6000 McBSP: AC'97 Codec Interface," available at www.ti.com, July 2001

[42] Datasheet, Texas Instruments, Spra556B, "A Multichannel/Algorithm Implementation on the TMS320C6000 DSP," available at www.ti.com, February 2000

[43] Datasheet, Texas Instruments, Spra569A, "TMS320C6000 McBSP: IOM-2 Interface," available at www.ti.com, May 1999

APPENDIX A

TMS320C6701 ARCHITECTURE

A.1 TMS320 Family Evolution

Digital Signal Processors are similar to general-purpose microprocessors except that they are more optimized to perform multiplication and addition operations. DSPs also have the advantage of consuming less power and being relatively cheap. The first low cost DSP device, the TMS320C10, was introduced by Texas Instruments in 1982.

The TMS320 family can be divided into three groups of processors such as fixed-point processors, floating-point processors and multiprocessors. The fixed-point processors are 'C1x,'C2x, 'C2xx, 'C5x, 'C54x and 'C62xx. The floating-point DSPs are 'C3x, 'C4x, 'C67xx. 'C8x is the only processor of multiprocessor type. The 'C8x family can have upto five processors running in parallel. In each generation, various processors are available, each sharing the same CPU core but having different peripherals. Among these DSPs 'C6x has the highest performance and lowest power consumption.

A.2 TMS320C6701 Architecture

The internal structure of the DSP can be divided into three main blocks such as the core, peripherals and memory. The core has eight functional units. The TMS320C6701 has two fixed-point and four floating-point ALUs and two fixed/floating-point multipliers. These functional units can perform up to eight 32-bit instructions per clock cycle simultaneously. The on-chip peripherals consist of a Host Port Interface, two bi-directional Multi-channel Buffered Serial Ports, an External Memory Interface, a four channel DMA controller and two timers. These processors provide two 64K byte blocks of internal memory such as a 32-bit wide block of data memory and a 256-bit wide block of program/cache memory. The internal architecture of TMS320C6701 is shown in Figure A.1.

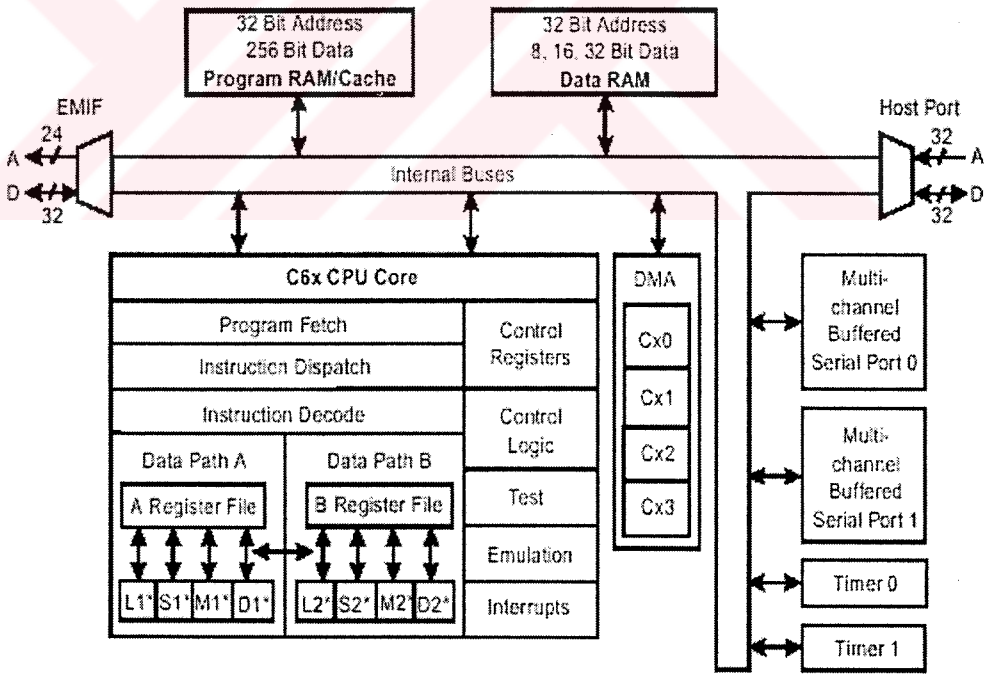


Figure A.1 TMS320C6701 Internal Architecture [23]

There are two sets of functional units in the CPU. Each set has four units and a register file. One set contains functional units .L1, .S1, .M1, and .D1; the other set contains the units .D2, .M2, .S2, and .L2. Each register file, A and B, contains sixteen 32-bit registers. The four functional units on each side of the CPU share the 16 registers belonging to that side. Each side has a single data bus connected to all registers on the other side, so the two sets of functional units can access data from the register files on opposite sides. .D1 and .D2 are the data-addressing units, which are responsible for all data transfers between the register files and the memory. The two M functional units are for multiplication operation. The two .S and .L functional units perform a general set of arithmetic, logical, and branch functions.

The device characteristics of C6x such as the capacity of on-chip RAM, the peripherals, the execution time, and the package type with pin count are clarified in Table A.1.

Table A.1 Characteristics of the TMS320C6701 Processors [24]

HARDWARE FEATURES	**TMS320C6701**
EXTERNAL MEMORY INTERFACE (EMIF)	1
DIRECT MEMORY ACCESS (DMA)	4 Channel
HOST PORT INTERFACE (HPI)	1
MULTICHANNEL BUFFERED SERIAL PORT (McBSP)	2
32-BIT TIMERS	2
INTERNAL PROGRAM MEMORY	64KByteS Cache/Mapped Program
INTERNAL DATA MEMORY	64K Bytes/ 2 Blocks/ Eight 16-Bit Banks per Block
FREQUENCY	120/150/167 MHz
CYCLE TIME	6 ns ('6701-167); 6.7 ns ('6701-150); 8.3 ns ('6701-120)
CORE VOLTAGE	1.9 V ('6701-167 only)
I/O VOLTAGE	3.3 V
PROCESS TECHNOLOGY	1.8 μm

A.3 Development Tools for the C6000 DSP Platform

There are software and hardware development tools for the TMS320C6000 DSP platforms, which are used for evaluating the performance of the processors, generating code, developing algorithm implementations, and integrating and debug software and hardware modules. Software development tools consist of Code Composer Studio_Integrated Development Environment (IDE), which includes editor for C/C++/Assembly Code Generation and Debug, Real-Time Foundation Software (DSP BIOS), which provides the basic run-time target software to support any DSP

application. Hardware development tools are Extended Development System Emulator (XDSE) and EVM (Evaluation Module).

A.3.1 Code Composer Studio

Code Composer Studio is an integrated development environment that includes the code generation tools, which are compiler, assembler and linker, Code Composer Studio Integrated Development Environment (IDE), DSP/BIOS plug-ins and RTDX plug-in, host interface, and API. Figure A.2 shows how these components work together.

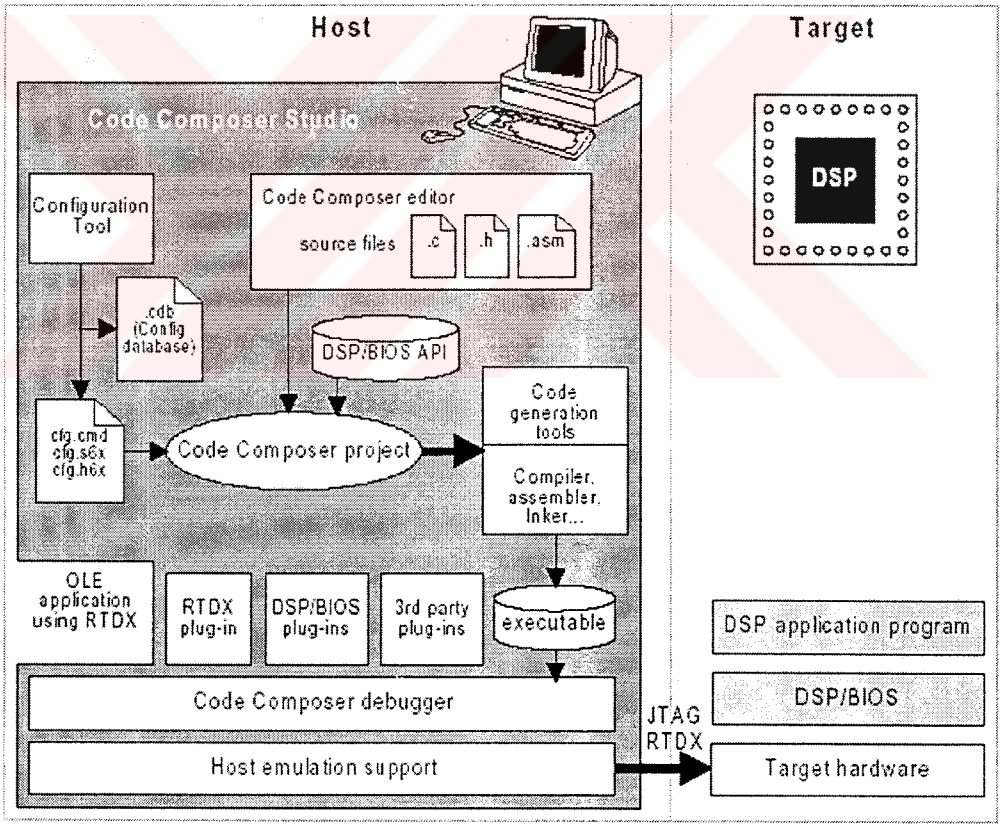


Figure A.2 Working diagram for the parts of Code Composer Studio (CCS) [23]

Code Composer Studio extends the basic code generation tools with a set of debugging and real-time analysis capabilities. The phases of the development cycle are shown in Figure A.3.

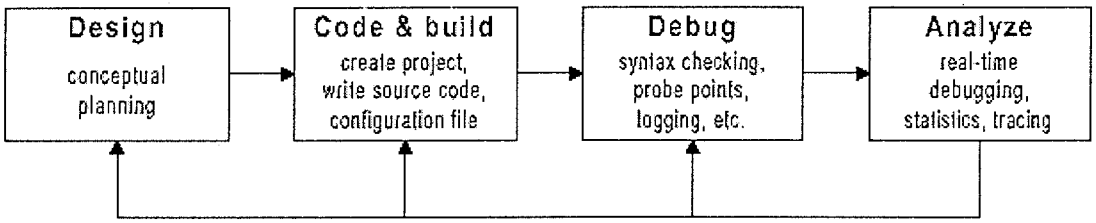


Figure A.3 The phases of the development cycle that CCS supports [23]

The project can be written in the C programming language. This source code is converted to the assembly language code by the C compiler. The assembler converts this code to the machine language object files. The machine language is based on common object file format (COFF). The linker combines object files into a single executable object file. The details of C6x software development flow are described in [23] and [25].

APPENDIX B

CONFIGURATION OF McBSP

```
/* Include files */
#include <c6x.h>
#include <csl.h> /* CSL library */
#include <mcbsp.h> /* MCBSP_SUPPORT */
#include <intr.h>

 /* Declare CSL objects */
MCBSP_HANDLE hMcbsp0; /* Handles for McBSP */
static MCBSP_CONFIG MyConfig = {
MCBSP_MK_SPCR(
MCBSP_SPCR_FRST_NO,  /*frame sync generator reset*/
MCBSP_SPCR_GRST_NO,  /*sample rate generator reset*/
MCBSP_SPCR_DXENA_NA,
MCBSP_SPCR_XINTM_XRDY,/*transmit interrupt mode */
MCBSP_SPCR_XRST_NO,/*transmitter reset*/
MCBSP_SPCR_DLB_OFF, /*digital loopback mode disabled*/
 MCBSP_SPCR_RJUST_RZF, /*right justify and zero-fill MSBs in DRR*/
MCBSP_SPCR_CLKSTP_DISABLE,
MCBSP_SPCR_RINTM_RRDY,
```

```
MCBSP_SPCR_RRST_NO
),
MCBSP_MK_RCR(
MCBSP_RCR_RWDREVRS_NA,
MCBSP_RCR_RPHASE_SINGLE,
MCBSP_RCR_RFRLEN2_OF(0),
MCBSP_RCR_RWDLEN2_32BIT,
MCBSP_RCR_RCOMPAND_MSB,
MCBSP_RCR_RFIG_YES,
MCBSP_RCR_RDATDLY_1BIT,
MCBSP_RCR_RFRLEN1_OF(0),
MCBSP_RCR_RWDLEN1_32BIT,
MCBSP_RCR_RPHASE2_NA),
MCBSP_MK_XCR(
MCBSP_XCR_XWDREVRS_NA, /* no bit reversal */
 MCBSP_XCR_XWDLEN1_32BIT, /* word length 32 bit */
MCBSP_XCR_XFRLEN1_OF(0), /* one word in phase 1 */
MCBSP_XCR_XPHASE2_NA, /* don't care */
MCBSP_XCR_XDATDLY_2BIT, /* transmit data delay 2 */
1, /* MCBSP_XCR_XFIG hard coded due */
/* to a bug in CSL Version 1.2 */
MCBSP_XCR_XCOMPAND_MSB, /* no companding, MSB first */
MCBSP_XCR_XWDLEN2_8BIT, /* don't care */
MCBSP_XCR_XFRLEN2_OF(0), /* don't care */
MCBSP_XCR_XPHASE_SINGLE /* single phase frame */
),
MCBSP_MK_SRGR(
MCBSP_SRGR_CLKGDV_OF(1), /* clock divider - global define */
```

```
MCBSP_SRGR_FWID_OF(0), /* FS width - one cycle low (0+1) */
MCBSP_SRGR_FPER_OF(0), /* FS period - 16 (15+1) cycles */
MCBSP_SRGR_FSGM_FSG, /* FS driven by sample rate gen. */
MCBSP_SRGR_CLKSM_CLKS, /* SRG driven by internal clock */
MCBSP_SRGR_CLKSP_RISING, /* rising edge of CLKS used */
MCBSP_SRGR_GSYNC_FREE /* CLKG running free */
),
MCBSP_MCR_DEFAULT, /* don't care */
MCBSP_RCER_DEFAULT, /* don't care */
MCBSP_XCER_DEFAULT, /* don't care */
MCBSP_MK_PCR(
MCBSP_PCR_CLKRP_FALLING, /* don't care */
MCBSP_PCR_CLKXP_RISING, /* xmit data on falling edge */
MCBSP_PCR_FSRP_ACTIVEHIGH, /* don't care */
MCBSP_PCR_FSXP_ACTIVEHIGH, /* transmit frame sync active low */
MCBSP_PCR_DXSTAT_0, /* don't care */
MCBSP_PCR_CLKSSTAT_0, /* don't care */
MCBSP_PCR_CLKRM_INPUT, /* don't care */
MCBSP_PCR_CLKXM_OUTPUT, /* CLKX used as clock output pin */
MCBSP_PCR_FSRM_EXTERNAL, /* don't care */
MCBSP_PCR_FSXM_EXTERNAL, /* xmit FS generated internally */
MCBSP_PCR_RIOEN_SP, /* don't care */
MCBSP_PCR_XIOEN_SP /* transmitter in serial port mode */
)
};

        void main(void)
{
```

```
CSL_Init();
/* Call MCBSP_Open and open serial port 1. Handle will be returned. */
hMcbsp0 = MCBSP_Open(MCBSP_DEV0, MCBSP_OPEN_RESET);

/* Write above configuration in McBSP1 configuration registers. */
MCBSP_ConfigA(hMcbsp0,&MyConfig);
MCBSP_EnableSrgr(hMcbsp0);
MCBSP_EnableXmt(hMcbsp0);
MCBSP_EnableFsync(hMcbsp0);
while (!MCBSP_Xrdy(hMcbsp0));
MCBSP_Write(hMcbsp0,0x3B040000);

    }
```

APPENDIX C

SOURCE CODE FOR EMIF CONFIGURATION

```
UINT32 gblctl,ce0ctl,ce1ctl,ce2ctl,ce3ctl,sdctl,sdtim,sdext;
/* make global control register value */
 gblctl = EMIF_MK_GBLCTL(
  EMIF_GBLCTL_RBTR8_HPRI,
  EMIF_GBLCTL_SSCRT_CPUOVR2,
  EMIF_GBLCTL_CLK2EN_DISABLE,
  EMIF_GBLCTL_CLK1EN_DISABLE,
  EMIF_GBLCTL_SSCEN_ENABLE,
  EMIF_GBLCTL_SDCEN_ENABLE,
  EMIF_GBLCTL_NOHOLD_0
 );
    /* make CE0 control register value */
 ce0ctl = EMIF_MK_CECTL(
  EMIF_CECTL_RDHLD_OF(0),
  EMIF_CECTL_MTYPE_SBSRAM32,
  EMIF_CECTL_RDSTRB_OF(0),
  EMIF_CECTL_TA_NA,
  EMIF_CECTL_RDSETUP_OF(0),
  EMIF_CECTL_WRHLD_OF(0),
  EMIF_CECTL_WRSTRB_OF(0),
```

```
  EMIF_CECTL_WRSETUP_OF(0)
 );
   /* make CE1 control register value */
 ce1ctl = EMIF_MK_CECTL(
  EMIF_CECTL_RDHLD_OF(3),
          EMIF_CECTL_MTYPE_ASYNC32,
  EMIF_CECTL_RDSTRB_OF(10),
  EMIF_CECTL_TA_NA,
  EMIF_CECTL_RDSETUP_OF(60),
  EMIF_CECTL_WRHLD_OF(3),
  EMIF_CECTL_WRSTRB_OF(30),
  EMIF_CECTL_WRSETUP_OF(4)
 );
   /* make CE2 control register value */
 ce2ctl = EMIF_MK_CECTL(
  EMIF_CECTL_RDHLD_OF(0),
  EMIF_CECTL_MTYPE_SDRAM32,
  EMIF_CECTL_RDSTRB_OF(0),
  EMIF_CECTL_TA_NA,
  EMIF_CECTL_RDSETUP_OF(0),
  EMIF_CECTL_WRHLD_OF(0),
  EMIF_CECTL_WRSTRB_OF(0),
  EMIF_CECTL_WRSETUP_OF(0)
 );
  /* make CE3 control register value */
 ce3ctl = EMIF_MK_CECTL(
EMIF_CECTL_RDHLD_OF(0),
EMIF_CECTL_MTYPE_SDRAM32,
EMIF_CECTL_RDSTRB_OF(0),
```

```
EMIF_CECTL_TA_OF(0),
EMIF_CECTL_RDSETUP_OF(0),
   EMIF_CECTL_WRHLD_OF(0),
   EMIF_CECTL_WRSTRB_OF(0),
   EMIF_CECTL_WRSETUP_OF(0)
 );
   /* make SDRAM control register value */
 sdctl = EMIF_MK_SDCTL(
   EMIF_SDCTL_TRC_OF(6),
   EMIF_SDCTL_TRP_OF(1),
   EMIF_SDCTL_TRCD_OF(1),
   EMIF_SDCTL_INIT_YES,
   EMIF_SDCTL_RFEN_ENABLE,
   EMIF_SDCTL_SDWID_2X16BIT,
   EMIF_SDCTL_SDCSZ_NA,
   EMIF_SDCTL_SDRSZ_NA,
   EMIF_SDCTL_SDBSZ_NA
   );
   /* make SDRAM timing register value */
 sdtim = EMIF_MK_SDTIM(
   EMIF_SDTIM_PERIOD_OF(1040),
   EMIF_SDTIM_XRFR_NA
 );
    /* make SDEXT register value */
 sdext = EMIF_SDEXT_NA;
   /* configure the EMIF */
 EMIF_ConfigB(gblctl,ce0ctl,ce1ctl,ce2ctl,ce3ctl,sdctl,sdtim,sdext);
```

APPENDIX D

THE CODE FOR REAL TIME DATA CAPTURE

```
#define CE1_ADDRS 0x01580000
#define INT_MEM 0x03000000
#define INT_MEM1 0x03100000
#define CE1_CNTRL 0x01800004
#define BOARD_ADDRS CE1_ADDRS
#define SRC_ADDRS INT_MEM
#define SRC_ADDRS1 INT_MEM1

 /* Include files */
#include <c6x.h>
#include <irq.h>
#include <mcbsp.h>
#include <intr.h>
#include <stdio.h>
#include <stdlib.h>
#include <csl.h>
#include <emif.h>

#include <6600misc.h>
#include <6600int.h>
#include <allregs.h>
#include <regs.h>
#include <6600led.h>
#include <6600timr.h>
#include <hpi.h>
#include <error.h>

/* Define constants */
```

```
#define FALSE ((BOOL)(0))
#define TRUE ((BOOL)(1))

unsigned int  * board_ptr =( unsigned int  *)BOARD_ADDRS;
unsigned int  * src_ptr =(unsigned int  *)SRC_ADDRS;
unsigned int  * src_ptr1 =(unsigned int  *)SRC_ADDRS;
unsigned int  * src_ptr2 =(unsigned int  *)SRC_ADDRS;
unsigned int  * src_ptr3 =(unsigned int  *)SRC_ADDRS1;
unsigned int mailboxValue;
volatile int xmit0_done = FALSE;
int recv0_done = FALSE;
int x;
volatile UINT32 y;
/*interrupt void timer0_isr(); */

interrupt void xint1_isr();
interrupt void xint_isr();

void read_board(unsigned int *);
void basadon   (unsigned int * src_ptr1);
void emif_config();


MCBSP_HANDLE hMcbsp0; /* Handles for McBSP */
MCBSP_HANDLE hMcbsp1;

static MCBSP_CONFIG MyConfig = {
.
.
given in Appendix B
.
.

};

void main() {

x=0;
pcic6600InitialiseMisc();
pcic6600InitialiseInterrupts();
pcic6600InitialiseTimers(0);
CSL_Init();
c6xSetISTP((void *)0x02500000);
```

```
pcic6600LockSharedBusB();

if (pcic6600TryToLockSharedBusB())
{
printf("okey \n");
}
else
{
printf("sorry try again\n");
}
emif_config();

c6201SetInterruptSource(C6X_INT_NUM_14,C6201_INT_SOURCE_XINT1);
 if (c6xInstallInterruptHandler(C6X_INT_NUM_14,
            xint1_isr) == FALSE)
{
printf("error\n");
}
c6201SetInterruptSource(C6X_INT_NUM_8,C6201_INT_SOURCE_XINT0);

if (c6xInstallInterruptHandler(C6X_INT_NUM_8,
            xint_isr) == FALSE)
{
printf("error\n");
}

/*if (c6xInstallInterruptHandler(C6201_DEFAULT_C6X_INT_NUM_EXT_INT5,
            timer0_isr) == FALSE)
{
printf("noluyo\n");
} */

/* Enable the interrupt */
/* c6xEnableInterruptNum(C6201_DEFAULT_C6X_INT_NUM_EXT_INT5); */
/* Enable some interrupts */
c6xEnableInterruptNum(C6X_INT_NUM_8);
c6xEnableInterruptNum(C6X_INT_NUM_14);
/* And enable interrupts globally */
c6xEnableGlobalInterrupts();

/* Call MCBSP_Open and open serial port 1. Handle will be returned. */
hMcbsp0 = MCBSP_Open(MCBSP_DEV0, MCBSP_OPEN_RESET);
hMcbsp1 = MCBSP_Open(MCBSP_DEV1, MCBSP_OPEN_RESET);
```

```
/* Write above configuration in McBSP1 configuration registers. */
MCBSP_ConfigA(hMcbsp0,&MyConfig);
MCBSP_ConfigA(hMcbsp1,&MyConfig1);
/* Port is setup, let's enable it in steps. */
MCBSP_EnableSrgr(hMcbsp0);
MCBSP_EnableSrgr(hMcbsp1);
MCBSP_EnableXmt(hMcbsp0);
MCBSP_EnableRcv(hMcbsp0);
MCBSP_EnableXmt(hMcbsp1);
/*MCBSP_EnableRcv(hMcbsp0); */
MCBSP_EnableFsync(hMcbsp0);
MCBSP_EnableFsync(hMcbsp1);


/*MCBSP_EnableRcv(hMcbsp1); */



 for (;;)
{

while (!pcic6600ReadHostMailbox(&mailboxValue));


while (!(xmit0_done == TRUE));

basadon(src_ptr1);
xmit0_done = FALSE;
pcic6600WriteHostMailboxAsync(0);

 }
  }
void emif_config(){
.
.
given in Appendix C
.
.
}

/*function for reading the I/O module*/
void read_board(unsigned int * board_ptr)
{
```

```
 int i;
 unsigned int * ctrl_addr1 = ( unsigned int *) (( unsigned int )board_ptr);

/* reading I/O */

for (i = 0; i < 140; i ++){
ctrl_addr1 =board_ptr ;
* src_ptr++=(* ctrl_addr1);

}

}

void basadon(unsigned int * src_ptr1)  {

 src_ptr =src_ptr1;

}

interrupt void xint_isr()
 {
  MCBSP_Write(hMcbsp0,0x3B040000);
 c6xClearInterruptNum(C6X_INT_NUM_8);
read_board(board_ptr);
 }
  interrupt void xint1_isr()
 {
  int m;
 basadon(src_ptr1);
 MCBSP_Write(hMcbsp1,0x3B040000);
 c6xClearInterruptNum(C6X_INT_NUM_14);
 for (m=0;m<18200;m++){
 *src_ptr3++=(((*src_ptr2))&0x00000FFF)+0x100;
 src_ptr2=src_ptr2+1;
          }
 src_ptr2=src_ptr2-18200;
 src_ptr3=src_ptr3-18200;
 xmit0_done = TRUE;
  }
```

APPENDIX E

SOURCE CODE FOR IMAGE FORMATION

```
#include <stdio.h>
#include <string.h>
#include "genrhl.h"
#include <windows.h>
#include <gl/glut.h>
#define X 10
#define Y 10
#define MEMORY_OFFSET       0x03000000   /* C6x SDRAM Memory */
#define SDRAM_MEMORY_OFFSET 0x00010000   /* MPC860 SDRAM */
#define BUFFER_LENGTH       0x4100       /* Length of Data Transfer */
void timer(int t);
void Draw_Image(void);
void DisplayErrorMessage(Gen_Handle_t handle, Gen_Error_t result, const
char * context)
{
 /* Simple function to output error messages to console */
char errmsg[100];
 GEN_GetErrorString(handle, result, errmsg, sizeof(errmsg));
 printf("Failed: [%s]\n'%s'\n", context, errmsg);
```

```
}
void DisplaySystemErrorMessage(Gen_LibHandle_t handle, Gen_Error_t result,
const char * context)
{
 /* Simple function to output error messages to console */
 char errmsg[100];
 GEN_GetSystemErrorString(handle, result, errmsg, sizeof(errmsg));
 printf("Failed: [%s]\n'%s'\n", context, errmsg);
}
void DisplayBuffer(uint32_t * buffer, uint32_t address, uint32_t length)
{
 /* Simple function to output buffer contents to console */
int32_t i, j;
for (i = 0; i < length; i += 4)
{
printf("0x%08x: ", address + (i * 4));
 for (j = i; (j < length) && (j < i + 4); j++)
{
printf(" 0x%08x", buffer[j]);
}

printf("\n");
}
}
// Global card variables
Gen_LibHandle_t  library;
Gen_Handle_t     board, node;
Gen_Transfer_t   transfer;
Gen_Error_t      error;
```

```
uint32_t        buffer[BUFFER_LENGTH], offset;
Gen_Property_t  property;
char            boardType[50];
uint32_t m,n,k,l,a[128][128];
int Initialize_Board()
{
 /* Initialise the system */
if ((error = GEN_Initialise(&library)) != GEN_ERR_SUCCESS)
{
 DisplaySystemErrorMessage(library, error, "GEN_Initialise");
  return 1;
  }
  /* We are going to use the first board available in the system. */
  /* Attempt to open the board */
  if ((error = GEN_BoardOpenByEnum(library, 0, FALSE, &board)) !=
GEN_ERR_SUCCESS)
  {
 DisplaySystemErrorMessage(library, error, "GEN_BoardOpenByEnum");
 GEN_Shutdown(library);
  return 1;
  }
  /* Find out what type of board we are using */
  property.p_type      = GEN_PROP_STRING;
  property.p_string    = boardType;
  property.p_string_len = sizeof(boardType);
  if(error=GEN_GetDetails(board,"BoardType",&property))!=GEN_ERR_SUCCESS)
{
 DisplayErrorMessage(board, error, "GEN_GetDetails");
 GEN_Close(board);
```

```
 GEN_Shutdown(library);
 return 1;
 }
 /* If the card is not a CPCI/C5421 we are going to use the DSP_0 node  */
 /* In all other supported hardware this will resolve down to a C6x DSP. */
 if (strcmp(boardType,"CPCI/C5421") != 0)
 {
 /* Attempt to open the node */
  if ((error=GEN_NodeOpenByName(board, "DSP_1", TRUE, &node)) !=
GEN_ERR_SUCCESS)
 {
 DisplayErrorMessage(board, error, "GEN_NodeOpenByName");
 GEN_Close(board);
 GEN_Shutdown(library);
 return 1;
  }
  offset = MEMORY_OFFSET; /* Set memory offset */
 }
 else /* CPCI/C5421 */
 {
 /* Attempt to open the node */
 if ((error=GEN_NodeOpenByName(board, "MPC860_A", TRUE, &node)) !=
GEN_ERR_SUCCESS)
 {
 DisplayErrorMessage(board, error, "GEN_NodeOpenByName");
 GEN_Close(board);
 GEN_Shutdown(library);
 return 1;
 }
```

```
offset = SDRAM_MEMORY_OFFSET; /* Set memory offset */
  }
  }
void GetData()
{
 /* Read data from memory */
 transfer.xf_mode   = GEN_XFER_MODE_ANY;
 transfer.xf_region = 0;
 transfer.xf_offset = offset;
 transfer.xf_length = BUFFER_LENGTH;
 transfer.xf_buffer = buffer;
 if ((error = GEN_Read(node, &transfer)) != GEN_ERR_SUCCESS)
 {
 DisplayErrorMessage(node, error, "GEN_Read");
 GEN_Close(node);
 GEN_Close(board);
 GEN_Shutdown(library);
 };
for (l=0;l<BUFFER_LENGTH;l+=1)
{
buffer[l]=buffer[l]&0x00000FFF;
};
 /*DisplayBuffer(buffer, offset, BUFFER_LENGTH);*/
}
void Close_Conn()
{
 GEN_Close(node);
 GEN_Close(board);
 GEN_Shutdown(library);
```

```
}
 void init(void)
{
glClearColor(0.0, 0.0, 0.0, 0.0) ;/*clear values for the color buffers*/
glShadeModel(GL_SMOOTH) ;/*selects flat or smooth shading*/
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT) ;/*The buffers currently enabled for color
writing*/
glPushMatrix() ;/*push and pop the current matrix stack.*/
Draw_Image() ;
glPopMatrix() ;
glutSwapBuffers() ;
}
void Draw_Image(void) // Function to draw a simple wire frame cube
{
int x,y;
GetData();// data collection
for (x=0;x<128;x++)
{
for (y=0; y<128; y++)
{
glColor3f((float)*(buffer+(y+2)*128+x+10)/2048,
(float)*(buffer+(y+2)*128+x+10)/2048,
(float)*(buffer+(y+2)*128+x+10)/2048);
glRectf(x*X,y*Y,(x+1)*X, (y+1)*Y);/*draw a rectangle*/
}
}
```

```
 }
void reshape(int w, int h)
{
 if (w >= h)
 glViewport(0, 0, (GLsizei)h, (GLsizei)h) ;
 else
 glViewport(0, 0, (GLsizei)w, (GLsizei)w) ;
 glMatrixMode(GL_PROJECTION);/*specifies which matrix is the current matrix.*/
 glLoadIdentity() ;/*replaces the current matrix with the identity matrix*/
 gluOrtho2D(0.0,128*X,0.0,128*Y) ;/*defines a 2-D orthographic projection matrix.*/
 glMatrixMode(GL_MODELVIEW) ;
 glLoadIdentity() ;
}
void keyboard (unsigned char key, int x, int y)
{
switch (key)
{
case 's' :
case 'S' :
glShadeModel(GL_SMOOTH) ;
 break ;
case 'f' :
case 'F' :
glShadeModel(GL_FLAT) ;
break ;
default :
break ;
}
glutPostRedisplay() ;
```

```
}
void timer(int t)
{
glutTimerFunc(200, timer,  0);
glutPostRedisplay(); //tell GLUT to call display again
}
int main(int argc, char** argv)
{
glutInit(&argc, argv) ;/*initialize the GLUT library.*/
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB) ;/*sets the initial display
mode-use double buffering to draw to screen*/
glutInitWindowSize(256, 256) ;/*set the initial window position*/
glutInitWindowPosition(1, 1) ;
glutCreateWindow("Real time image viewer - v1.0") ;/*creates a top-level window.*/
init() ;
glutDisplayFunc(display) ;/*sets the display callback for the current window*/
glutReshapeFunc(reshape) ;
glutKeyboardFunc(keyboard) ;
Initialize_Board();
glutTimerFunc(200, timer,  0);
glutMainLoop() ;
Close_Conn();
return 0 ;   }
```

APPENDIX F

THE CODES FOR NORMALIZATION COEFFICIENTS

F.1 The Code for Calculating One-point Offset Coefficients

```
/*This code is used for calculating of offset coefficients*/

#define INT_MEM      0x03800000 /*gain coefficient*/
#define FLASH_ADDRS  0x01440000 /*address to image*/
#define FLASH FLASH_ADDRS
#define BUFFER INT_MEM

#include <stdio.h>
#include <stdlib.h>
#include <c6x.h>

unsigned int  *flash_ptr  = (unsigned int *)FLASH;
unsigned int  *buffer_ptr = (unsigned int *)BUFFER;

void main()
 {
unsigned int x,sum,avg;
int i,sayi;
sayi=0;
sum=0;
printf("start\n");

for (i=0;i<16384;i++)
 {
x=(*flash_ptr++)&0x00000FFF;
```

```
c6201MicrosecondSleep(500);

if ((x<0x550)&&(x>0x300))
{
 sum=sum+x;
 sayi=sayi+1;
}
}
avg=sum/sayi;
printf("%x avg\n",avg);
 printf("%x sum\n",sum);
  printf("%x sayi\n",sayi);
flash_ptr=flash_ptr-16384;
for (i=0;i<16384;i++)
{
*buffer_ptr++=(avg-((*flash_ptr++)&0x00000FFF));



}



printf("end\n");
}
```

F.2 The Code for Calculating Two-point Gain Coefficients

```
/*This code is used for calculating gain coefficient*/

#define INT_MEM      0x03800000 /*gain coefficient*/
#define INT_MEM1     0x03900000 /*address to buffer2*/
#define FLASH_ADDRS  0x01440000  /*address to image from cold*/
#define FLASH1_ADDRS 0x01480000  /*address to image from hot*/
#define INT_MEM3     0x03000000
#define SRC_ADDRS INT_MEM1
#define FLASH FLASH_ADDRS
#define FLASH1 FLASH1_ADDRS
#define BUFFER INT_MEM
#define BUFFER3 INT_MEM3
```

```
#include <stdio.h>
#include <stdlib.h>
#include <c6x.h>

unsigned int  *flash_ptr  = (unsigned int *)FLASH;   /*cold*/
unsigned int  *flash1_ptr = (unsigned int *)FLASH1;  /*hot*/
unsigned int  *buffer_ptr = (unsigned int *)BUFFER;  /*gain*/
unsigned int  *src_ptr    = (unsigned int *)SRC_ADDRS;
unsigned int  *buffer3_ptr = (unsigned int *)BUFFER3;
float aveg;

void main()
 {

unsigned int x,y,z,sum,avg,gain,image;
int i,j,m,u,total;
sum=0;
printf("start\n");

for (i=0;i<16384;i++)
 {
x=(*flash1_ptr++)&0x00000FFF;
c6201MicrosecondSleep(500);
y=(*flash_ptr++)&0x00000FFF;
c6201MicrosecondSleep(500);
 z=x-y;
 if(z&0x10000000){
z=(~z)+1;
sum=sum-z;/*total of difference*/
*src_ptr++=z; /*difference 0x0390*/
 }
 else
 {
  sum=sum+z;
*src_ptr++=z;
 }
 };
 src_ptr=src_ptr-16384;
 flash_ptr=flash_ptr-16384;
 flash1_ptr=flash1_ptr-16384;
 avg=1000*sum/0x4000;

  for (j=0;j<16384;j++)
```

```
 {
 gain=avg/((*src_ptr++)+1);
  *buffer_ptr++=gain;
   }
 buffer_ptr=buffer_ptr-16384;
 src_ptr=src_ptr-16384;

for (m=0;m<16384;m++)

            {

  image = (*flash_ptr++)&0x00000FFF;
  if (((image)<0x500)&&((image)>0x100))
  {
  total=total+image;
  u=u+0x1;
  }

 }
   printf("end \n");
   if (total<0)
   {
   total=(~total)+1;
   }

  aveg=total/(u);
  *buffer3_ptr=aveg;


}
```

APPENDIX G

THE PROGRAM SEGMENTS FOR SIGNAL PROCESSING APPLICATIONS

G-1 One-point Offset Correction

```
    interrupt void xint1_isr()
{

  int m,n;

  unsigned int z;

MCBSP_Write(hMcbsp1,0x3B040000);

c6xClearInterruptNum(C6X_INT_NUM_14);

  x=x+1;
  if (x==2){
  x=0;
  basadon(src_ptr1);
 for (m=0;m<128;m++)
{
for (n=0;n<128;n++)
{
 z=*buffer1_ptr++;
 if ((z)&0x10000000){
 z=~(z)+1 ;
```

```
*src_ptr2++=((*(src_ptr11+(2+m)*140+n+10))&0x00000FFF)-z;

      }
      else
      {
*src_ptr2++=((*(src_ptr11+(2+m)*140+n+10))&0x00000FFF)+z;
      }
      }
      }
 src_ptr2=src_ptr2-16384;
 buffer1_ptr=buffer1_ptr-16384;
  for (m=0;m<128;m++)
{
  for (n=0;n<128;n++){
 if ((*src_ptr2<0x400))
 {
 *pixel_ptr++=((*(src_ptr2+1)&0x00000FFF)+ (*(src_ptr2-1)&0x00000FFF)
 + (*(src_ptr2-127)&0x00000FFF) +(*(src_ptr2+127)&0x00000FFF)+
  (*(src_ptr2-128)&0x00000FFF)+ (*(src_ptr2+128)&0x00000FFF)+
  (*(src_ptr2-129)&0x00000FFF) +(*(src_ptr2+129)&0x00000FFF))/8;
 src_ptr2=src_ptr2+1;
 }
else
{
*pixel_ptr++=*src_ptr2++;

}
 }
 }
 pixel_ptr=pixel_ptr-16384;
src_ptr2=src_ptr2-16384;
xmit0_done = TRUE;
   }
    }
```

G.2 Two-point Gain Correction

```
interrupt void xint1_isr()
 {
  int m,n;
 unsigned int z,p;
 MCBSP_Write(hMcbsp1,0x3B040000);
 c6xClearInterruptNum(C6X_INT_NUM_14);
 basadon(src_ptr1);  /*duruma gore degisecek**/
 x=x+1;
 if(x==3)
 {
 x=0;
 for (m=0;m<18200;m++){
 *src_ptr2++=((*src_ptr11)&0x00000FFF);
   src_ptr11=src_ptr11+1;
            }
 src_ptr2=src_ptr2-18200;
 src_ptr11=src_ptr11-18200;
    for (m=0;m<128;m++)
              {
           for (n=0; n<128; n++)
                     {
           z= (*(buff2_ptr+(m+2)*140+10+n));
              p= (* buffer1_ptr++);
     if (((z)<0x700)&&((z)>0xBB))
      {
      *src_ptr++= z*p/0x200+0x200;
      }
      else
      {
      *src_ptr++=(z);
      }

       }
        }
   buffer1_ptr=buffer1_ptr-16384;
   src_ptr=src_ptr-16384;

  for (m=0;m<128;m++)
 {
```

```
 for (n=0;n<128;n++)
 {
 if ((*src_ptr<0x400))
 {
 *pixel_ptr++=((*(src_ptr+1)&0x00000FFF)+
(*(src_ptr-1)&0x00000FFF)+(*(src_ptr+129)&0x00000FFF)+
 (*(src_ptr-129)&0x00000FFF)+
(*(src_ptr-128)&0x00000FFF)+(*(src_ptr+128)&0x00000FFF)+
 (*(src_ptr+127)&0x00000FFF)+(*(src_ptr-127)&0x00000FFF))/8;
  src_ptr=src_ptr+1;
 }
 else
 {
 *pixel_ptr++=*src_ptr++;

 }
 }
 }
 pixel_ptr=pixel_ptr-16384;
src_ptr=src_ptr-16384;
 xmit0_done = TRUE;

  }
}
```

G.3 Frame Averaging

```
interrupt void xint1_isr()
 {
  int m,n,k;
 unsigned int z,p;
  MCBSP_Write(hMcbsp1,0x3B040000);
   c6xClearInterruptNum(C6X_INT_NUM_14);
  x=x+1;
if(x==2)
{
x=0;
basadon(src_ptr1);


    for (k=0;k<2;k++)
```

```
    {
     for (m=0;m<128;m++)
          {
          for (n=0; n<128; n++)
          {

    z= (*(src_ptr11+(k*18200)+(m+2)*140+10+n))&0x00000FFF;
          p= (* buffer1_ptr++);

       if (((z)<0x700)&&((z)>0xBB))
       {
       *src_ptr++= z*p/0x150+0x200;
       }
       else
       {
       *src_ptr++=z;
       }
       }
       }
        buffer1_ptr=buffer1_ptr-16384 ;
                  }
        src_ptr=src_ptr-32768;

      for (m=0;m<128;m++)
  {
  for (n=0;n<128;n++)
  {
  *pixel_ptr1=((*(src_ptr)+(*(src_ptr+16384)))&0x00001FFE)/2;
   pixel_ptr1=pixel_ptr1+1;
   src_ptr=src_ptr+1;
   }
   }
   pixel_ptr1=pixel_ptr1-16384;
   src_ptr=src_ptr-16384;
    xmit0_done = TRUE;
          }
}
```

G.4 Sobel Filter Application

```
interrupt void xint1_isr()
{
  int m,n;
unsigned int z,p;
MCBSP_Write(hMcbsp1,0x3B040000);
c6xClearInterruptNum(C6X_INT_NUM_14);
 basadon(src_ptr1);
 x=x+1;
if(x==4)
{
x=0;

  for (m=0;m<128;m++)
          {
        for (n=0; n<128; n++)
                {

     z= (*(src_ptr11+(m+2)*140+10+n));
          p= (* buffer1_ptr++);

      if (((z)<0x800)&&((z)>0x50))
     {
     *src_ptr++= z*p/0x50+0x500;
     }
     else
     {
     *src_ptr++=(z);
     }

      }
      }
  buffer1_ptr=buffer1_ptr-16384;
  src_ptr=src_ptr-16384;

 for (m=0;m<128;m++)
{
 for (n=0;n<128;n++)
{
if ((*src_ptr<0x400))
{
```

```
 *pixel_ptr++=(((*(src_ptr+1))+
  (*(src_ptr-1))+(*(src_ptr+129))+
 (*(src_ptr-129))+(*(src_ptr-128))+(*(src_ptr+128))+
 (*(src_ptr+127))+(*(src_ptr-127)))&0x00007FF8)/8;
  src_ptr=src_ptr+1;
 }
 else
 {
 *pixel_ptr++=*src_ptr++;

 }
 }
 }
  pixel_ptr=pixel_ptr-16384;
src_ptr=src_ptr-16384;

for(m=0;m<126;m++){
for(n=1;n<127;n++){
 * pixel_ptr1=
                    ((*(pixel_ptr+ (m+3)*128+n-127)))*1
        +((*(pixel_ptr+ (m+3)*128+n-128)))*2
        +((*(pixel_ptr+ (m+3)*128+n-129)))*1
        +((*(pixel_ptr+ (m+3)*128+n+129)))*-1
        +((*(pixel_ptr+ (m+3)*128+n+128)))*-2
        +((*(pixel_ptr+ (m+3)*128+n+127)))*-1;

  if (* pixel_ptr1 &0x10000000){
  *pixel_ptr1=~(*pixel_ptr1)+1;
  pixel_ptr1=pixel_ptr1+1;
  }
  else
  {
  *pixel_ptr1=*pixel_ptr1;
   pixel_ptr1=pixel_ptr1+1;
 }
            }
       }
pixel_ptr1=pixel_ptr1-15876;
xmit0_done = TRUE;

 }

}
```