DESIGN AND IMPLEMENTATION OF A MICROPROCESSOR BASED DATA COLLECTION AND INTERPRETATION SYSTEM WITH ONBOARD GRAPHICAL INTERFACE

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

BY

GÖKHAN GÖKSÜGÜR

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

DECEMBER 2004

Approval of the 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. İsmet ERKMEN
Head of Department

This is to 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.

Prof. Dr. Hasan Cengiz GÜRAN
Supervisor

Examining Committee Members

Assoc. Prof. Gözde BOZDAĞI AKAR	(METU,EE)	
Prof. Dr. Cengiz Hasan GÜRAN	(METU,EE)	
Asst. Prof. Dr. Cüneyt BAZLAMAÇCI	(METU,EE)	
Dr. Ece GÜRAN	(METU,EE)	
Umur AKINCI (M.S.)	(ASELSAN)	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Gökhan GÖKSÜGÜR

ABSTRACT

DESIGN AND IMPLEMENTATION OF A MICROPROCESSOR BASED DATA COLLECTION AND INTERPRATATION SYSTEM WITH ONBOARD GRAPHICAL INTERFACE

Göksügür, Gökhan

M.S., Department of Electric and Electronics Engineering

Supervisor : Prof. Dr. Hasan Cengiz Güran

December 2004, 103 pages

This thesis reports the design and implementation of a microprocessor based interface unit of a navigation system. The interface unit is composed of a TFT display screen for graphical interface, a Controller Circuit for system control, a keypad interface for external data entrance to the system and a power interface circuit to provide interface between the battery of the navigation system and the Controller Circuit. This thesis reports high speed design of the Controller Circuit and generation of system functions. Main functions of the interface unit are communicating with navigation computer and providing a graphical interface to the driver of the vehicle containing the navigation system. Communication and graphical data preparation functions are implemented through the use of a microprocessor. Driver function of TFT display is implemented through the use of a Field Programmable Gate Array, which is programmed using the Very High Speed IC Description Language (VHDL).

Keywords: Navigation System, Interface Unit, Controller Circuit, Image Generation

ÖZ

VERİ TOPLAYIP YORUMLAYAN GRAFİK ARAYÜZLÜ VE MİKROİŞLEMCİ TABANLI BİR SİSTEMİN TASARIMI VE GERÇEKLEŞTİRİLMESİ

Göksügür, Gökhan
Yüksek Lisans, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Hasan Cengiz Güran

Aralık 2004, 103 sayfa

Bu tez, bir seyrüsefer sistemindeki mikroişlemci tabanlı arayüz ünitesinin tasarımını ve gerçekleştirilmesini anlatmaktadır. Arayüz ünitesi, grafiksel arayüz için bir TFT görüntü ekranı, sistem kontrolü için bir Denetleyici Devresi, sisteme dış verilerin girişi için bir küçük klavye arayüzü ve seyrüsefer sisteminin pili ile Denetleyici Baskı Devre Kartı arasında arayüzü sağlayan bir güç arayüz devresinden oluşmaktadır. Bu tez, Denetleyici Devresinin yüksek hızlı tasarımını ve sistem işlevlerinin oluşturulmasını anlatmaktadır. Arayüz ünitesinin temel işlevi seyrüsefer bilgisayarıyla haberleşmek ve seyrüsefer sisteminin bulunduğu aracın kullanıcısına grafiksel bir arayüz sağlamaktır. İletişim ve grafiksel verilerin hazırlanma işlevleri mikroişlemci kullanılarak gerçekleştirilir.TFT görüntü ekranının sürücü işlevleri Çok Yüksek Hızlı Entegre Devre Tanımlama Dili (VHDL) kullanılarak programlanan Alan Programlanabilir Kapı Dizini kullanılarak gerçekleştirilir.

Anahtar Kelimeler: Seyrüsefer sistem, Arayüz Birimi, Denetleyici Devresi, Görüntü Oluşturma

ABBREVIATIONS

AC	Alternative Current
BUF	Buffer
CPLD	Complex Programmable Logic Device
DC	Direct Current
DMA	Direct Memory Access
DPSRAM	Dual Port Static Random Access Memory
DRAM	Dynamic Random Access Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
EMI	Electro Magnetic Interference
FPGA	Field Programmable Gate Array
GLONASS	The Global Navigation Satellite System
GPS	Global Positioning System
Hsync	Horizontal Synchronization signal
I2C	Inter Integrated Circuit Control
INS	Inertial Navigation System
JTAG	Joint Test Access Group
LCD	Liquid Crystal Display
MAC	Memory Access Control
MIPS	Mega Instructions per second
PCB	Printed Circuit Board
PROM	Programmable Read Only Memory
PWM	Pulse Width Modulation
RGB	Red Green Blue display format
RISC	Reduced Instruction Set Computer
SIM	System Integration Module
SRAM	Static Random Access Memory
TFT	Thin Film Transistor
UART	Universal Asynchronous Receiver Transmitter
VHDL	Very High Speed Integrated Circuit Description Language
Vsync	Vertical Synchronization signal

To My Family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Prof. Dr. Hasan Cengiz Güran for his supervision and guidance.

Special thanks to ASELSAN Inc. Microelectronics, Guidance and Electro-Optics division for providing technical support and laboratory environment in which I could develop my design.

I am thankful to all my family and my friends for their continued patience, understanding, encouragement and support throughout the preparation of this study.

I am also grateful to my fiancé Rafet for her support, patience and belief in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Overview

In automotive and military systems navigation, positioning and situation awareness problems are important problems to be answered. Navigation systems try to give these answers by providing information to moving platforms about their current position and velocity with high navigation accuracy within a common time and reference coordinate system. Although, there are different types of navigation systems most modern navigation systems have a generic structure as shown in Figure 1.1.

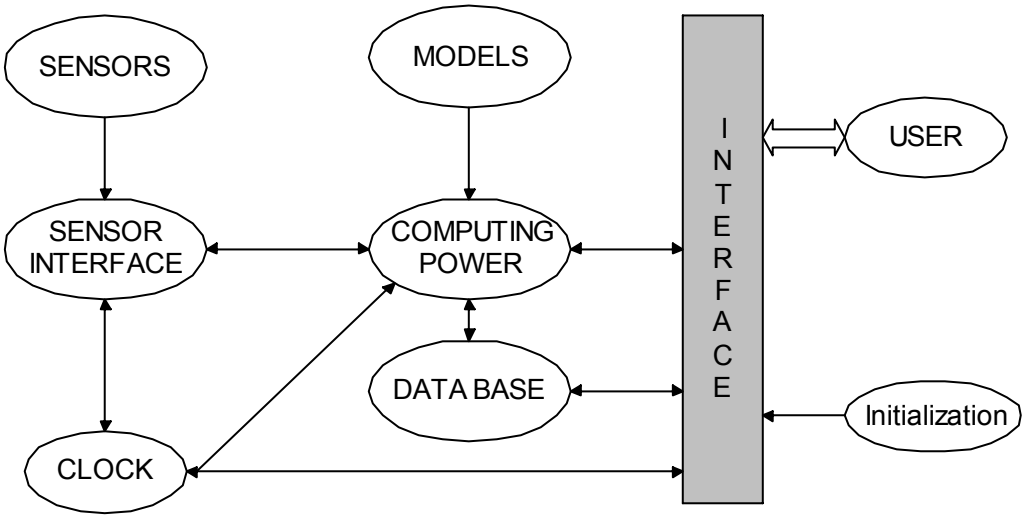

Figure 1.1 Typical Navigation System Structure

This thesis focuses on the design and implementation of an interface unit, Navigation Control and Display Unit, of a navigation system. Main function of Navigation Control and Display Unit is to communicate with navigation computer and provide a graphical interface to the driver of the vehicle containing the navigation system. With the help of Navigation Control and Display Unit the driver can enter some initial data to the navigation system and will get continuous data about the current position of the vehicle. Correct interpretation of present data will lead the driver to correct destination.

Navigation Control and Display Unit is composed of a TFT display screen for graphical interface, Navigation Control and Display Unit Controller Circuit, a keypad interface for external data entrance, communication channels to communicate with the other parts of navigation system and a power inverter and filtering circuit used to supply the interface part from the battery of the vehicle. The scope of this thesis includes the design of Navigation Control and Display Unit Controller Circuit and keypad circuit design and implementation of TFT driver, graphical data generation and communication functions. TFT driver function is implemented by using an FPGA. Graphical data generation and communication functions are implemented by using a microprocessor.

The interface unit is tested with a simulation program that is sending imaginary navigation data to the interface. Simulation program sends the navigation data to the interface unit in the same protocol and format of navigation systems. Figure 1.2 shows the Navigation Control and Display Unit Controller Circuit and Figure 1.3 shows the Navigation Control and Display Unit model.

Figure 1.2 Navigation Control and Display Unit

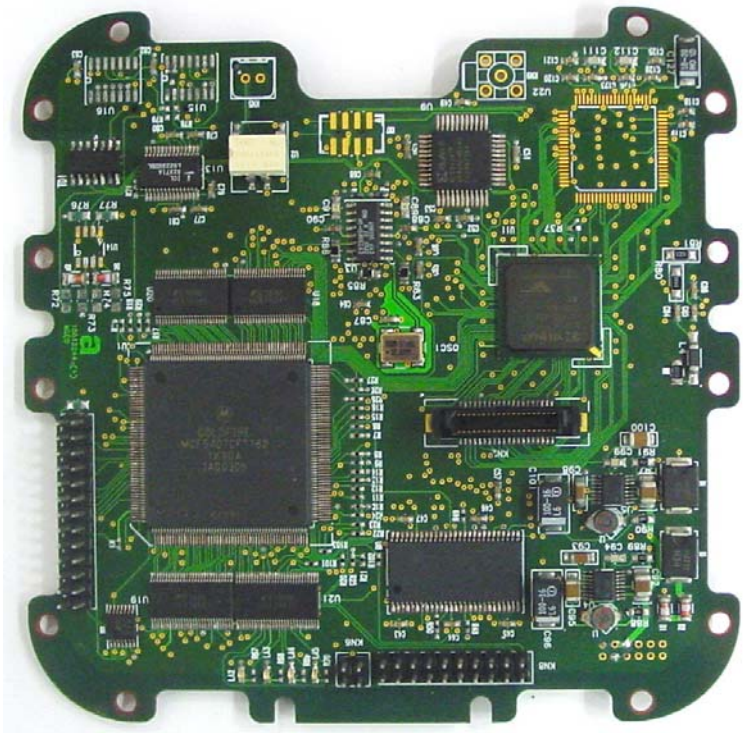

Figure 1.3 Navigation Control and Display Unit Controller Circuit

1.2. Organization of the Thesis

This thesis is composed of six chapters.

Chapter 2 provides an overview of Navigation Systems. It gives information about the current state of navigation systems, navigation system components and types of navigation systems.

Chapter 3 contains detailed information on Navigation Control and Display Unit Controller Circuit design. This section describes hardware requirements of the system, the components used in the Navigation Control and Display Unit Controller Circuit design and High Speed Printed Circuit Board Design considerations used in design and layout processes of the Navigation Control and Display Unit Controller Circuit.

Chapter 4 focuses on the software development of Navigation Control and Display Unit functions. It gives detailed description of implementation of graphical image generation and tools used for image generation. Keypad and communication interface and functions are explained in this section.

Chapter 5 presents operation and simulation of the Navigation Control and Display Unit.

Chapter 6 gives the conclusions.

CHAPTER 2

NAVIGATION SYSTEMS

2.1. Overview

Basically, navigation is traveling and finding way from one place to another. Specifically in science, it is the art of knowing;

- Where you are,
- How fast and in which direction you are moving,
- Positioning yourself in relation to your environment,
- Which way leads you to your destination

Modern weapon systems must have not only the capability to navigate, but also capability to direct a variety of on-board sensors and weapons; hence the need for accurate pointing information. There are three principal needs for positioning and navigation.

- The first is the classical navigation problem: To get from point A to B on a defined route.
- The second is pointing; to guide, launch or direct a weapon or sensor system.
- The third is situation awareness so that not only the platform with the navigation system, but also the team members know where the platform is [1].

2.2. Navigation Systems

Navigation systems can be categorized into three groups depending on how operates to get the knowledge of the dynamic state (position, velocity etc.) of a platform with respect to a common reference frame.

Direct Positioning Systems use discrete measurements from external reference points whose locations are well defined to determine the position and in some cases velocity of the platform. Examples: Radio navigation systems, satellite navigation.

Dead Reckoning Systems use a known initial dynamic state of the platform at a known time and continuously estimate its evolution, based on the information provided by sensors, such as velocity and rotation rate (Inertial Systems).

Correlation Systems sense and recognize some predefined and mapped characteristics of the environment and thus extract information related to dynamic state of the platform [1].

2.2.1. Elements of a Navigation System

Most modern electronic navigation systems have a generic structure as shown in Figure 1.1.

The **sensors** measure position and orientation or their rates of change.

The **clock** provides precise time so the computer can coordinate the data from sensors.

The **sensor interface** provides the measurements from the sensors to the computer.

The **models** compensate for environmental and dynamic effects on the sensors.

The **database** stores information related to the mission such as waypoints, maps, coordinate systems etc.

The **computer** performs the navigation computations based on models of the sensors, clock and environment.

Initialization is any process or data required to define and determine the initial position and alignment of the navigation system to a common reference frame.

Interface is the part showing all obtained and implemented information data to **user** and allowing some initialization process for system. The part implemented in this thesis is the interface part [1].

2.2.2. Satellite Navigation Systems

Three satellite navigation systems have been designed to give three-dimensional position, velocity and time data almost anywhere in the world with an accuracy of a few meters:

- The Global Positioning System – GPS (USA),
- The Global Navigation Satellite System – GLONASS (Russia),
- GALILEO (European Union).

Position is derived by computing the distance of the receiver from each satellite, by measuring the time taken for a radio signal transmitted from the satellite to travel to the receiver.

In order to make precise distance measurements, the accurate time tagging of satellite signal is essential- this is achieved with the aid of atomic clocks on each satellite.

Measurement of range requires at least four satellites to determine four unknowns, three spatial coordinates (latitude, longitude and altitude) and time.

By using the Doppler Shift of the satellite signal, the range rate to each satellite can be computed in the receiver.

Satellite navigation systems have the following advantages

- They are using an absolute Reference Frame,
- They are accurate
- They do not cause error growth in time
- They use low cost receivers

Their disadvantages are:

- They are not self-contained, that is dependent on external signals
- There may be discontinuities (position shifts can occur due to changing satellite geometry).
- Foliage and tall buildings can block the signal
- Received satellite signals are very weak and therefore vulnerable to interference or jamming [1].

2.2.3.Inertial Navigation Systems

Inertial Navigation Systems continuously measure linear motions and rotations, using on board accelerometers and gyroscopes respectively. Starting from a known position provided by the user or another system, the computer then computes the path followed. In this process the velocity and attitude of the vehicle are also provided continuously.

Inertial navigation systems have the following advantages

- They are self-contained, that is no external infrastructure required
- They provide continuous information
- They are robust to jamming

Their disadvantages are:

- Accuracy degrades with time,
- Need for initial position parameters,
- Accurate systems are expensive

2.2.3.1. Gyroscopes

Gyroscopes are sensors used to measure the amount of rotation or the rate of rotation over a certain time interval.

2.2.3.2. Accelerometers

An accelerometer is a sensor which converts accelerations from motion or gravity to electrical signals. If we measure the acceleration of a vehicle and then integrate this value of acceleration we get the vehicles velocity and integrating again will give the distance traveled that is required for inertial navigation.

2.2.4.INS/GPS Navigation

No single navigation system or sensor technology excels across the spectrum of technical, operational and cost requirements. However, by combining several technologies, an integrated system can be formed that has superior performance and cost characteristics. The concept of an integrated navigation system is to combine the outputs in a processor. With a careful design, the integration of one sensor technology with another results in an integrated system with vastly improved accuracy.

Since GPS and INS outputs and characteristics are complimentary to each other, integration of these sensors benefits both sensor systems resulting in a much better total navigation system [1].

CHAPTER 3

NAVIGATION CONTROL AND DISPLAY UNIT CONTROLLER CIRCUIT

3.1. Introduction

Navigation Control and Display Unit Controller Circuit is the main hardware part of the system provides hardware interface to TFT display, navigation computer, keypad circuit and Power Generator Circuit.

TFT display interface generates required signals to drive a 320*240 pixel TFT display providing graphical interface to the driver of the vehicle Navigation Control and Display Unit is placed.

Navigation computer interface is provided through RS232 and RS422 serial communication interfaces.

Keypad Circuit interface, used to initialize Navigation Control and Display Unit and to enter data to system, is provided with Input/Output (I/O) channels of an FPGA.

Power Generator Circuit interface is used for providing 5V to the Controller Circuit.

Block Diagram of Navigation Control and Display Unit Controller Circuit is given in Figure 3.1.

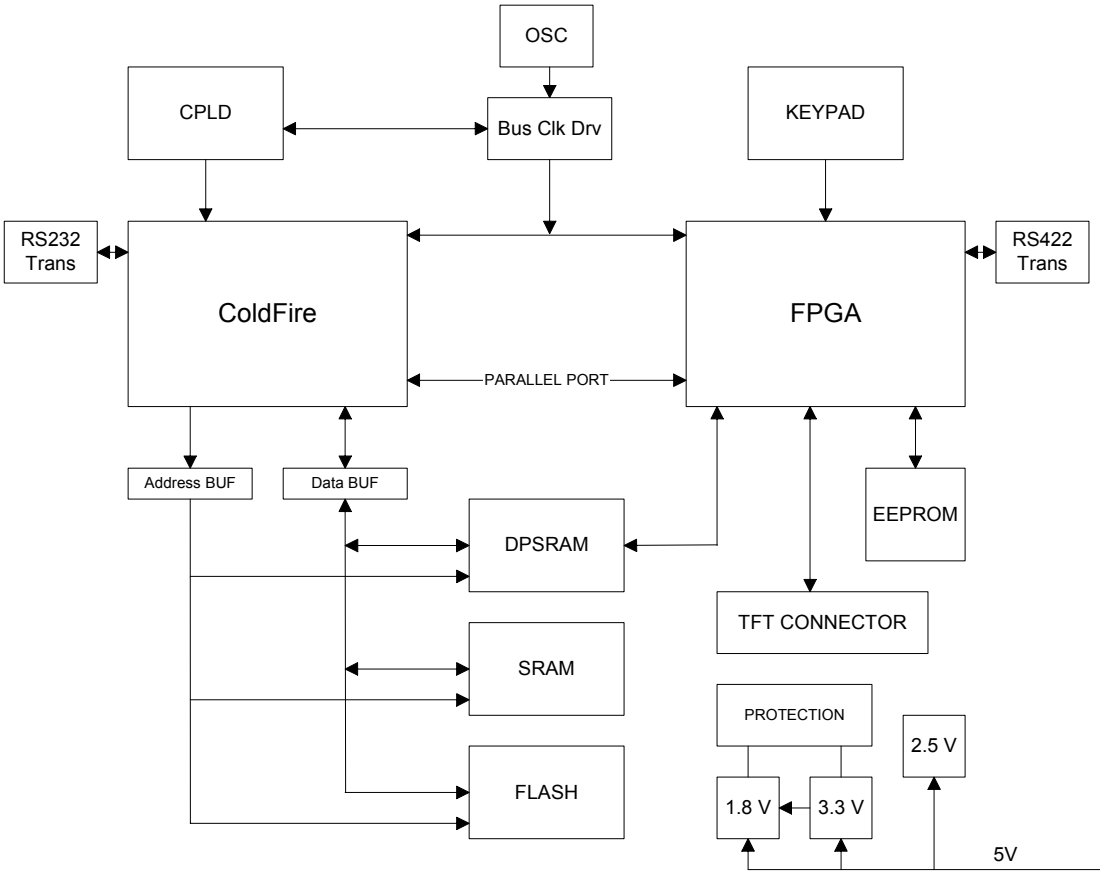

Figure 3.1 Block Diagram of Navigation Control and Display Unit Controller Circuit

3.2. Controller Circuit Components

3.2.1. TFT-LCD Module

Sharp's TFT-LCD [2] module is the display unit of the Navigation Control and Display Unit. This TFT Module is especially designed for use in general equipment designs such as Car Navigation Systems, Automotive Auxiliary Information Systems and Automotive audio visual equipment. This module is a color active matrix LCD module incorporating amorphous silicon TFT (Thin Film Transistor). Graphics and text can be displayed on a 320*RGB*240 dots panel with 262.144 colors. For the TFT work properly, the signals shown in Figure A.1 and the timings shown in Table A.1 must be met.

3.2.2. Power Components

Navigation Control and Display Unit Controller Circuit requires +5 V DC power supply for proper operation and its average power dissipation is 1.5 W. However most of the components on the PCB operates with 3.3 V, FPGA core requires 1.5 V and ColdFire core requires 1.8 V for proper operation. ColdFire microprocessor also requires protection for proper operation. To meet all these requirements a power supply and protection sub-circuit as shown in Figure 3.2 is built in Navigation Control and Display Unit Controller Circuit.

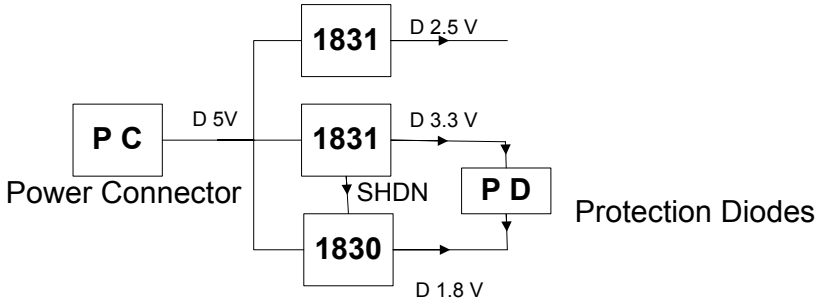

Figure 3.2 Power Supply and Protection sub-circuit

Components used in this sub-circuit are given in section 3.2.2.1 and 3.2.2.2 .

3.2.2.1. Power Regulators

To obtain all necessary voltage values for Navigation Control and Display Unit Controller Circuit power regulators MAX1830 and MAX1831 [3] from Maxim are used.

The MAX1830/MAX1831 constant-off-time, pulse-width modulated (PWM) step-down DC-DC converters are ideal for use in 5V and 3.3V to low-voltage conversion. These devices feature internal synchronous rectification for high efficiency and reduced component count. They require no external Schottky diode. The internal

45mΩ PMOS power switch and 55mΩ NMOS synchronous-rectifier switch easily deliver continuous load currents up to 3A. The MAX1830 produces preset +2.5V, +1.8V, or +1.5V output voltage or an adjustable output from +1.1V to VIN. The MAX1831 produces preset +3.3V, +2.5V, and +1.5V output voltages and an adjustable output from +1.1V to VIN. They achieve efficiencies as high as 94%.

In Navigation Control and Display Unit Controller Circuit MAX1831 is used to obtain 3.3 V and 2.5 V; MAX1830 is used to obtain 1.8 V for the microprocessor core. For the microprocessor, 3.3 V should rise earlier than 1.8 V for proper I/O operation as shown in Figure 3.3. To meet this constraint, output of 3.3V generating regulator is connected to SHDN pin of 1.8 V generating regulator, which cause 1.8 V regulator to open after 3.3 V regulator.

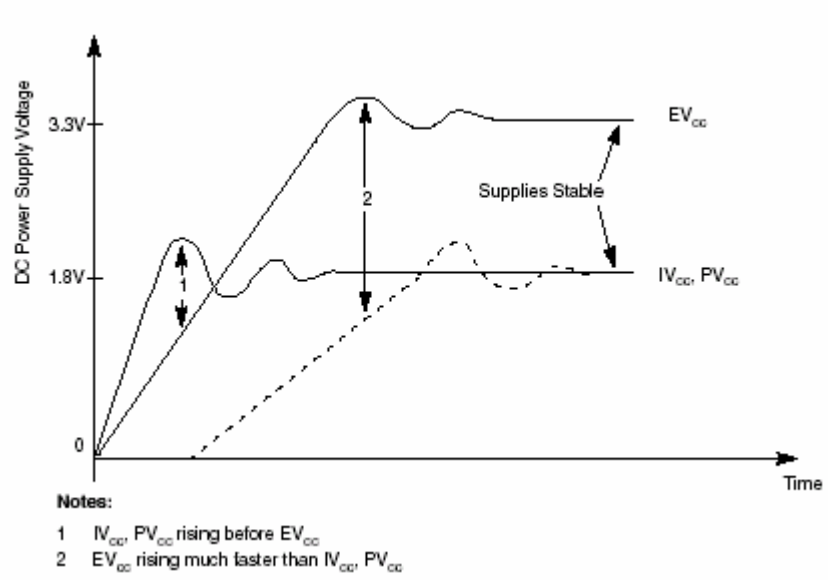

Figure 3.3 Power Up Sequence for ColdFire 5407

3.2.2.2. Protection Circuitry

The diodes are used in Navigation Control and Display Unit Controller Circuit to guarantee that the ColdFire 3.3 V I/O and 1.8 V DC core voltages power up and down properly. It is not desired to power up (or down) with only one supply voltage.

The diodes ensure that the core and I/O pad ring voltages power up/down simultaneously and therefore never exceed the maximum differential.

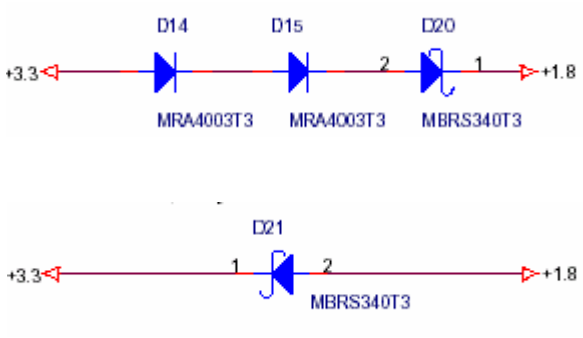

Figure 3.4 Protection Circuitry for ColdFire 5407

3.2.3. Programmable Components

Most of the functions of Navigation Control and Display Unit are implemented with the usage of programmable logic devices; a microprocessor, a Field Programmable Gate Array (FPGA) and a Complex Programmable Logic Device (CPLD). Microprocessor is generally used for long but sequential functions like image generation and remote communication, FPGA is used for fast and frequently used functions like TFT driving and keypad debouncing, and CPLD is used for operations requiring fast power up like system reset generation.

3.2.3.1. Microprocessor

Generation of images to be displayed on the screen and communication with long package sizes are sequential long processes that can be easily implemented with a microprocessor. To deal with such kind of processes a microprocessor MCF5407 [4] from Motorola is used in the Navigation Control and Display Unit Controller Circuit design. Most of the functions of the Controller Circuit; like image generation, menu and keypad control, serial communication and communication with system peripherals; are implemented with this microprocessor. Generations of functions are

explained in Chapter 4 in detail. A technical description of the microprocessor will be given in this part.

The MCF5407 integrated microprocessor combines a V4 ColdFire processor core with the following components, as shown in Figure 3.5:

- Harvard architecture memory system with 16-Kbyte instruction cache and 8-Kbyte data cache
- Two, 2-Kbyte on-chip SRAMs
- Integer/fractional multiply-accumulate (MAC) unit
- Divide unit
- System debug interface
- DRAM controller for synchronous and asynchronous DRAM
- Four-channel DMA controller
- Two general-purpose timers
- Two UARTs, one that supports synchronous operations
- I2C™ interface
- Parallel I/O interface
- System integration module (SIM)

Designed for embedded control applications, the MCF5407 delivers 233 Dhrystone MIPS at 162 MHz while minimizing system costs.

Based on the concept of variable-length RISC technology, the ColdFire family combines the architectural simplicity of conventional 32-bit RISC with a memory-saving, variable-length instruction set. In defining the ColdFire architecture for embedded processing applications, a 68K-code compatible core combines performance advantages of a RISC architecture with the optimum code density of a streamlined, variable-length M68000 instruction set.

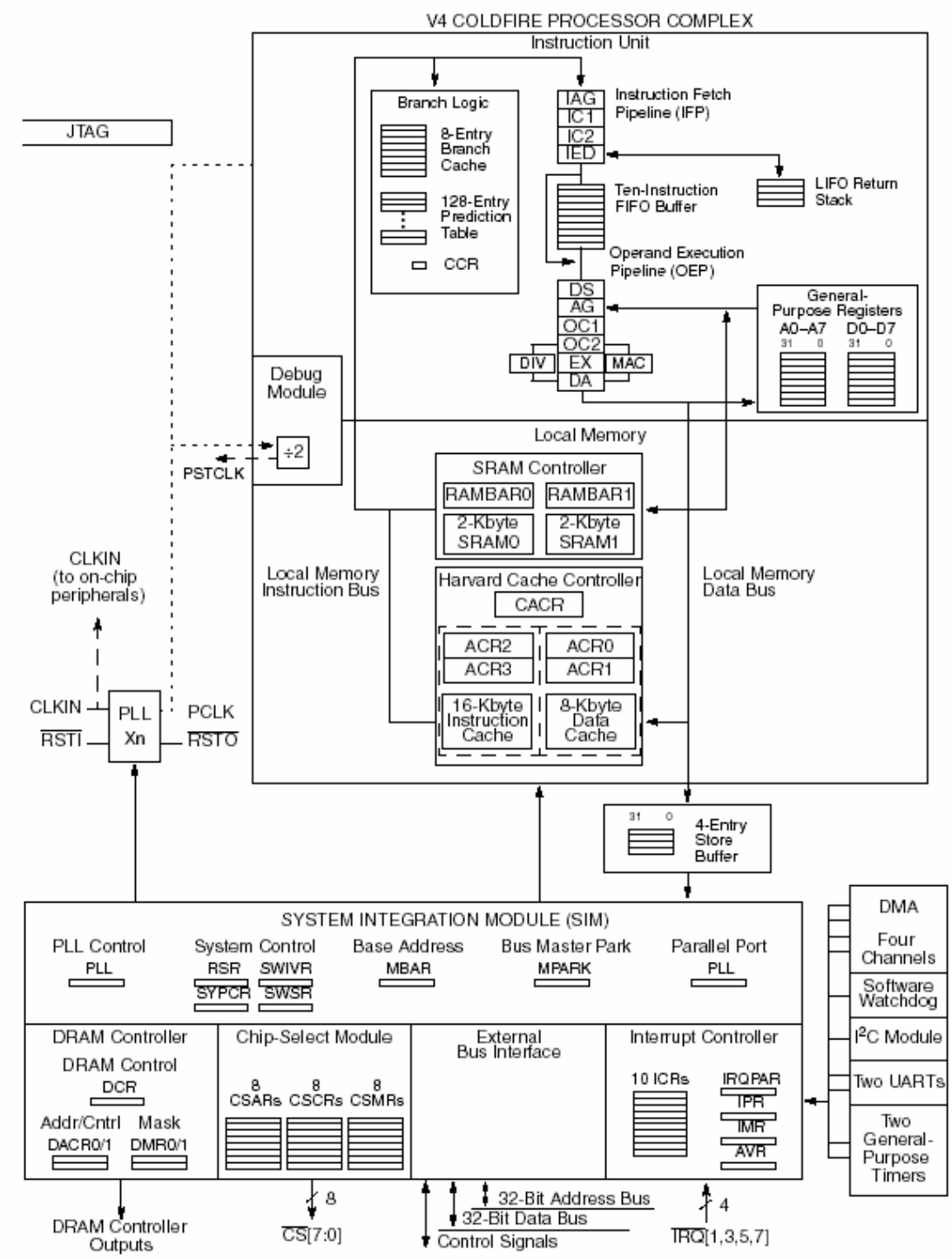

Figure 3.5 MCF5407 Block Diagram

The V4 microarchitecture implements a number of advanced techniques, including a Harvard memory architecture that uses separate buses for instruction and data,

branch cache acceleration logic, and limited superscalar support (dual-instruction issue), which contribute to the 233 Dhrystone MIPS performance level. Increasing the internal speed of the core also allows higher performance while providing the system designer with an easy-to-use lower speed system interface. The processor complex frequency is an integer multiple, 3 to 6 times of the external bus frequency. The core clock can be stopped to support a low-power mode.

Serial communication channels are provided by an I2C interface module and two programmable full-duplex UARTs, one of which provides synchronous communications for soft-modem applications. Four channels of DMA allow for fast data transfer using a programmable burst mode independent of processor execution. The two 16-bit general-purpose multimode timers provide separate input and output signals. For system protection, the processor includes a programmable 16-bit software watchdog timer. In addition, common system functions such as chip selects, interrupt control, bus arbitration, and an IEEE 1149.1 JTAG module are included. A sophisticated debug interface supports background-debug mode plus real-time trace and debug with an expanded set of on-chip breakpoint registers. This interface is present in all ColdFire standard products and allows common emulator support across the entire family of microprocessors.

3.2.3.2. Field Programmable Gate Array

One of the main control components of the Controller Circuit is XC2V250 [5] FPGA from Xilinx. FPGA devices feature a gate-array like architecture, with a matrix of logic cells surrounded by a periphery of Input/Output cells. Segments of metal interconnects can be linked in an arbitrary manner by programmable switches to the desired signal nets between the cells.

FPGAs are customized by loading configuration data into internal memory cells. The FPGA can either actively read its configuration data from an external serial or byte parallel PROM, or the configuration data can be written into the FPGA from an external device.

FPGA designs are supported by powerful and sophisticated software, covering every aspect of design from schematic or behavioral entry, floor planning, simulation, automatic block placement and routing of interconnects, to the creation and downloading of the configuration bit stream. Because FPGAs can be programmed an unlimited number of times, they can be used in innovative designs where hardware is changed dynamically, or where hardware must be adapted to different user applications.

Since TFT in Navigation Control and Display Unit does not have a driver which can refresh the data on the screen, a driver for the TFT display should be implemented. TFT requires continuous refreshment of display data as shown in Figure A. and this process has to be continuously implemented with a frequency of about 6 MHz. Implementation of driver function requires concurrent and continuous processing, which can be easily done by FPGAs. Implementation of the driver with microprocessor is of course possible, but it will drastically reduce the system performance. For a better system performance TFT driver function is decided to be implemented with XC2C250 FPGA.

TFT Driver function implemented in the FPGA produces TFT control signals like Horizontal synchronization (Hsync), Vertical synchronization (Vsync), data enable signal etc. and gets the required image data prepared by the microprocessor from a Dual Port RAM. It communicates with microprocessor to decide which frame is active for the current view and updates the required fields for corresponding frame.

Debouncing function for the keypad is also implemented by FPGA. FPGA continuously polls the keypad and detects the real press for the pads after filtering bounces on the line and produces required interrupt signal for the processor and with the interrupt it also says to the processor which key is pressed.

3.2.3.2.1. FPGA Functions

Processes required concurrent and continuous fast execution times are implemented using the FPGA, since these processes may overload the processor. TFT driver and keypad interface functions of the Controller Circuit are given to the FPGA.

FPGA code design flow starts with the analysis of the system requirements from which the subsections are defined. VHDL (Very High Speed Integrated Circuit Description Language) is used to describe the behavior and structure of digital electronic hardware for sub-functions. The correct operation of VHDL design blocks are verified by functional simulations using test benches. After simulations, gate level netlist is achieved by synthesis of the VHDL design. In the next step, the gate level design is placed and routed in the FPGA. Finally post layout simulation is used to verify the design timings. FPGA code design flow is summarized in Figure 3.6.

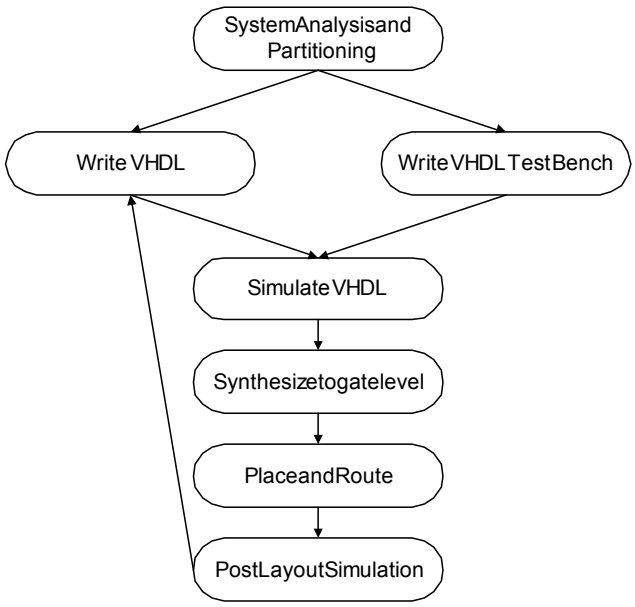

Figure 3.6 FPGA Code Design Flow

3.2.3.2.2. TFT Driver

TFT Driver function implemented in the FPGA produces TFT control signals like Horizontal synchronization signal (Hsync), Vertical synchronization signal (Vsync), data enable Signal etc. and gets the required image data prepared by the microprocessor from Dual Port SRAM. It communicates with the microprocessor to decide which frame is active for the current view and updates the required fields for the corresponding frame.

Generally TFT displays work with the following sequence of events:

For every line on the TFT screen a synchronization signal called Horizontal synchronization signal (Hsync) is generated. After a few clock times, with a data enable signals data corresponding to the pixels on that line are send from data lines. For all data lines this process is repeated. After completing all lines of the TFT display another synchronization signal called Vertical synchronization signal (Vsync) is generated to bring the data pointer to the beginning of the first line of the TFT display. Continuous operation of these processes provides continuous images on the screen.

Figure 3.7 shows the sub blocks of the TFT driver function.

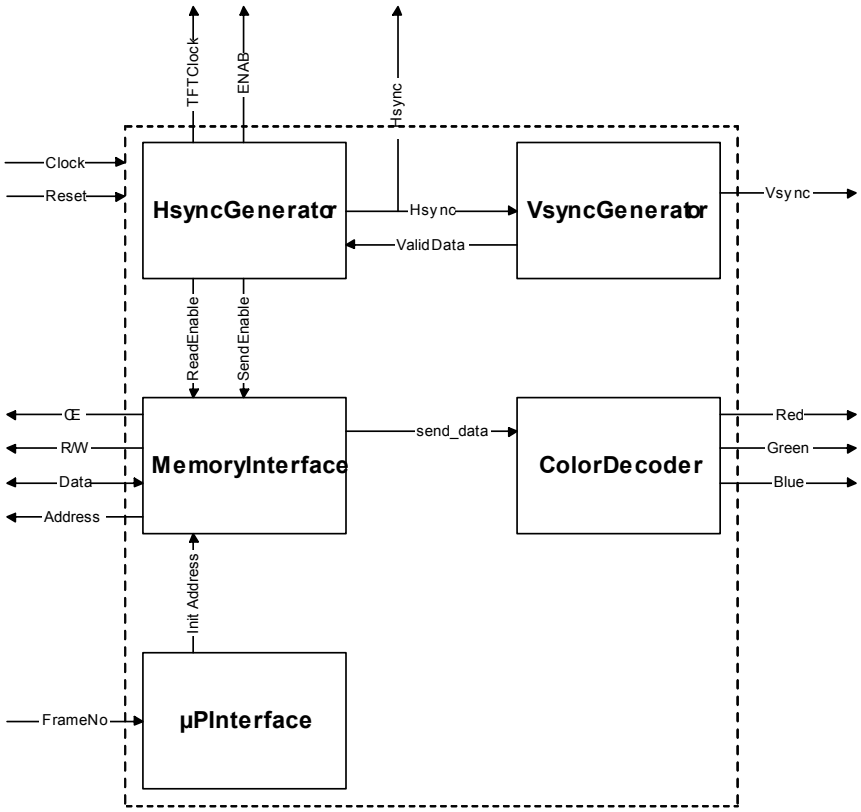

Figure 3.7 Block Diagram of TFT Driver

3.2.3.2.2.1.1. Horizontal Synchronization Generator

Horizontal Synchronization Generator block is written to generate TFT clock signal, data enable signal (ENAB) and horizontal synchronization signal with the timing requirements shown in Figure A.1. It also generates memory interface control signals data_read_enable and data_send_enable according to the value of valid_data signal coming from Vertical Synchronization Generator block.

TFT_clk signal, buffered version of tft_blk_buf, is the clock signal of TFT display. Horizontal synchronization signal Hsync is the signal generated at the beginning of every horizontal line of TFT display. Data_read_enable signal shows when a read from Dual Port SRAM should be done and data_send_enable signal shows when the read data should be sent to Color Decoder Block. As it is easily seen from TFT

display timings, some Hsync signals require data read from DPSRAM. Valid_data signal shows which Hsync signals should read data from DPSRAM. Data enable ENAB signal shows sequential display data is sent to TFT display.

Horizontal Synchronization Generator block is implemented with a state machine. In this state machine at every 8 system clock (Clock) a TFT clock is generated. Since system clock is 50 MHz, TFT clock with 6.25 MHz is obtained. During first 12 TFT clock cycle Horizontal synchronization signal is held at logic level 1.After Horizontal synchronization set to logic level 0, 60 TFT Clock cycle waited and then data enable signal (ENAB) is set to logic level 1. After ENAB set to logic 1, at every TFT clock cycle memory control signals are generated depending on the value of valid data signal coming from Vertical Synchronization Generator block. Memory control signals are generated for 320 TFT clock cycle since a line consist of 320 pixels. After memory control signals, 8 TFT clock cycle is waited and this process is repeated continuously. Block Diagram of Horizontal Synchronization Generator is shown in Figure 3.8 and VHDL code for Horizontal Synchronization Generator block is given in Appendix B.

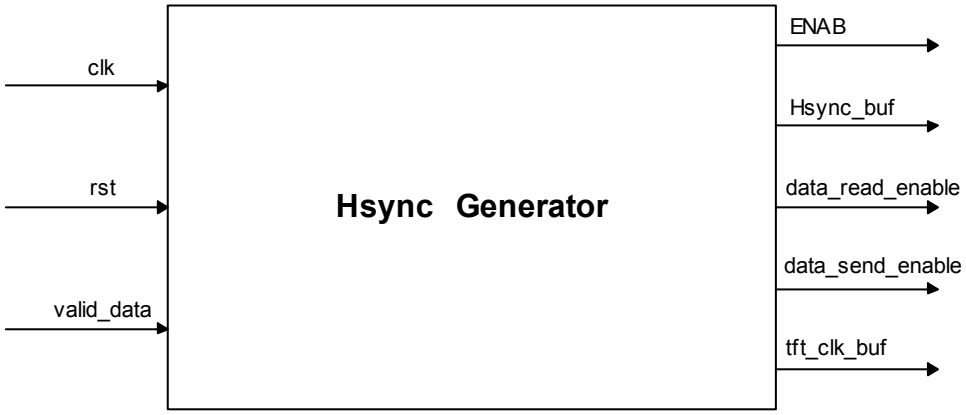

Figure 3.8 Block Diagram of Hsync Generator

3.2.3.2.2.1.2. Vertical Synchronization Generator

Vertical Synchronization generator block is used to generate vertical synchronization signals and to generate valid data signal to Horizontal Synchronization Generator Block. Vertical synchronization signal Vsync is the signal generated when all horizontal lines of TFT display are scanned to bring the data pointer to the beginning of the first line of the TFT display.

If we look at Figure A.1, we see that not at all horizontal synchronization cycle memory access and data send processes are done .Valid data signal tells to Horizontal Synchronization Generator block when to send memory control signals. During vertical invalid data states valid data signal is set to logic 0 means do not make any memory access. Block diagram of Vertical Synchronization Generator is shown in Figure 3.9 and VHDL code for Vertical Synchronization Generator block is given in Appendix B.

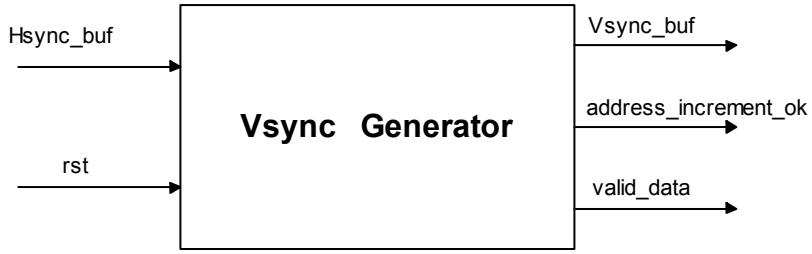

Figure 3.9 Block Diagram of Vsync Generator

3.2.3.2.2.1.3. Memory Interface

Memory Interface Block provides an interface between Dual Port Static RAM (DPSRAM) and FPGA. It waits data_read_enable and data_send_enable control signals from Hsync Generator Block. When data_read_enable signal is active it reads a memory location from DPSRAM which is the color information of a pixel of the TFT display. When data_send_enable is active it sends the read data to Color Decoder block. Initial_address signal is the first address of the DPSRAM to be read.

Memory Interface Block is implemented with a state machine in which when data_read_enable is active logic 0 is put to the Chip Enable CE0 and Output Enable OE and memory address to be read is put to the address port of the DPSRAM. After reading corresponding data Chip Enable and Output Enable is set to logic 1 and memory address is incremented by 1 so that next address will be read at next data_read_enable. Read cycle of DPSRAM is given in Figure 3.10. Block Diagram of Memory Interface is shown in Figure 3.11 and VHDL code for memory interface block is given in Appendix B.

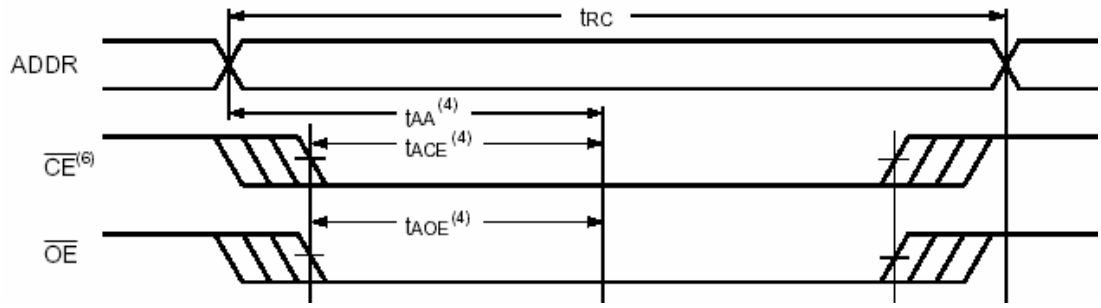

Figure 3.10 Read Cycle of DPSRAM

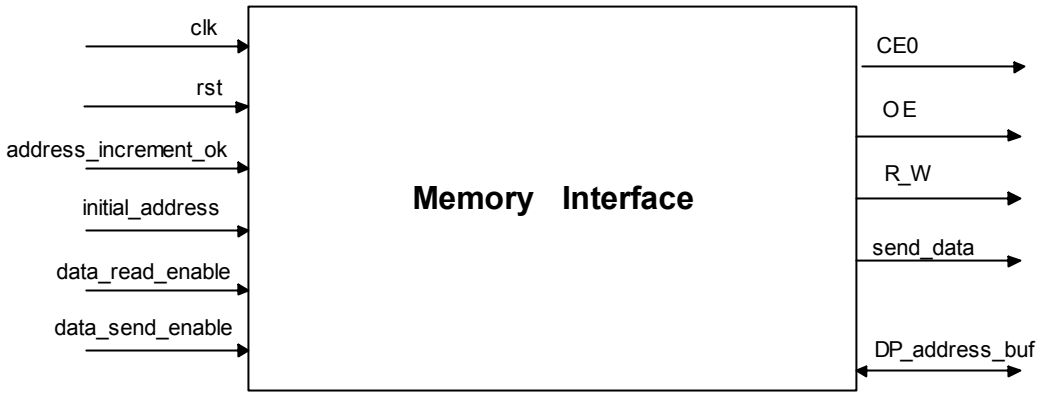

Figure 3.11 Block Diagram of Memory Interface

3.2.3.2.2.1.4. Color Decoder

Color decoder is a truth table that converts the color codes to RGB format to be displayed. Memory interface block sends read data to color decoder and decoded colors are sent to TFT display in RGB format. Figure 3.12 sows the color decoder truth table. Block Diagram of Color Decoder is shown in Figure 3.13 and VHDL code for color decoder block is given in Appendix B.

	A	B	C	D
1	send_data	TFT_RED	TFT_GREEN	TFT_BLUE
2	BLACK	"000000"	"000000"	"000000"
3	BLUE	"000000"	"000000"	"111111"
4	GREEN	"000000"	"111111"	"000000"
5	CYAN	"000000"	"111111"	"111111"
6	RED	"111111"	"000000"	"000000"
7	MAGENTA	"111111"	"000000"	"111111"
8	YELLOW	"111111"	"111111"	"000000"
9	WHITE	"111111"	"111111"	"111111"

Figure 3.12 Color Decoder Truth Table

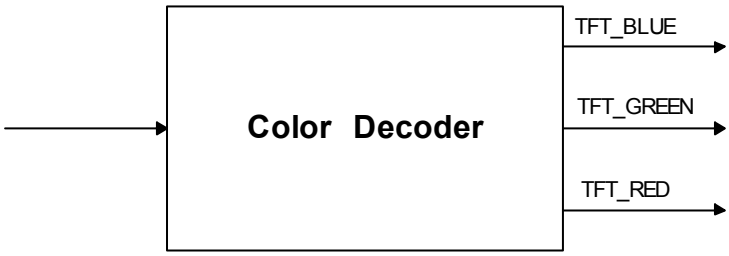

Figure 3.13 Block Diagram of Color Decoder

3.2.3.2.2.1.5. Microprocessor Interface

Microprocessor interface is a look up table that calculates initial address to be read according to the frame number. As presented at section 4.2 and shown in Figure 4.4 images are stored in DPSRAM as frames. For each frame there is a corresponding initial address.

3.2.3.2.2.2. Keypad Interface

Keypad Interface Block is an interface between keypad PCB and microprocessor. Keypad buttons are mechanical equipments so when they are pressed common debouncing problem occurs. Debouncer block implemented in keypad interface is designed to eliminate debouncing effect. Debouncer checks the buttons and if they are pressed it continues to check the line for 50 ms. If line changes state during the check time then button is assumed not pressed. During the all check time if it is continuously seem to be pressed it is assumed to be pressed. If a button is decided to have been pressed then interrupt level 7 pin (IRQ7) of the microprocessor is driven to logic level 0 which generates an interrupt for the microprocessor. Meanwhile, pressed button is decoded and sent to the microprocessor through the parallel part of the microprocessor. When microprocessor gets the interrupt, it checks the decoded keypad signal and decides what to do according to the pressed button. Block diagram of keypad interface is given in Figure 3.14 and VHDL code for debouncer is given in Appendix B.

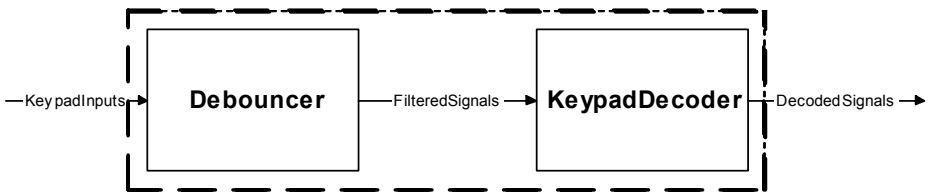

Figure 3.14 Block Diagram of Keypad Interface

3.2.3.3. Complex Programmable Logic Device (CPLD)

Another digital programmable control component present on the Controller Circuit is XC2C256 [6] CPLD from Xilinx. CPLD devices are programmable logic devices like FPGAs but they are slightly different from FPGAs in configuration view. Although FPGAs are configured by configuration memories and configuration takes some time CPLDs are preconfigured devices that are immediately ready to function with power on.

For the processor to be configured initially and to run properly reset timing specifications shown in Figure 3.15 must be meet. Since FPGA is not ready after power on immediately, it can not be used for reset timing and configuration of the processor. XC2C256 CPLD decided to be used for initial configuration and reset timings of the processor.

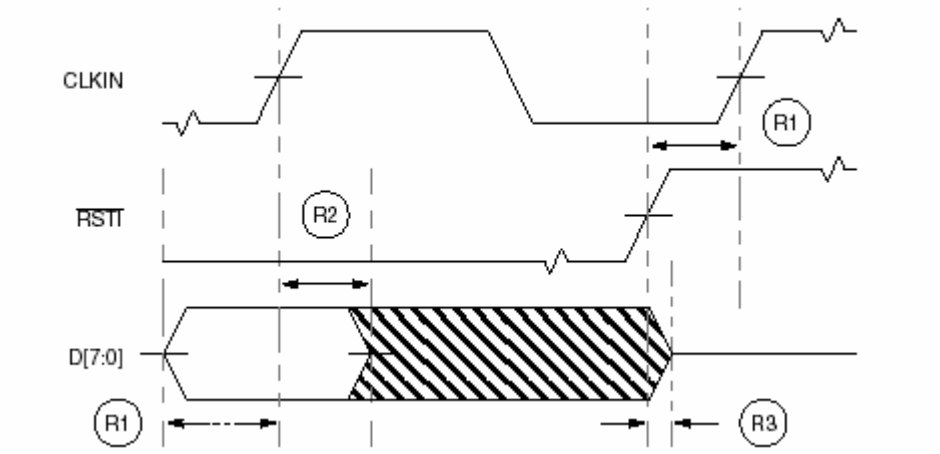

Figure 3.15 Reset Timing for ColdFire 5407 Microprocessor

3.2.4. Timing and Buffer Components

To provide the timing requirements; rise and fall time of clock line; and driving capabilities of data, address and control buses some precautions on the Navigation

Control and Display Unit Controller Circuit must be taken. Figure 3.16 shows the components used for timing and buffering.

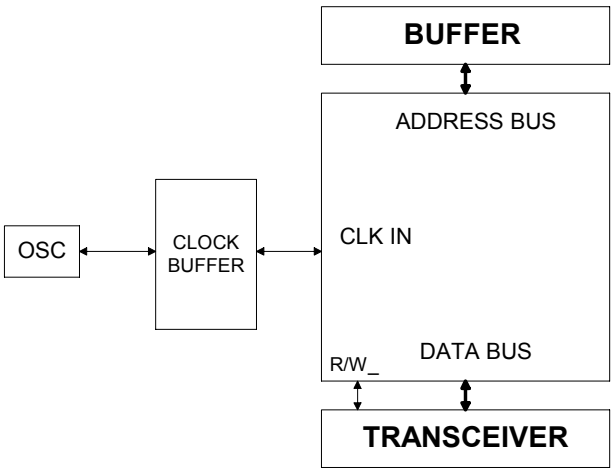

Figure 3.16 Timing and buffering precautions

3.2.4.1. Clock Driver

Since the microprocessor requires clock rise-time less than 2 ns and the oscillator rise-time is 5 ns, some improvement have to be made on the clock. For this purpose a clock driver have to be used on the board.

The CY2308 [7] is a 3.3V Zero Delay Buffer designed to distribute high-speed clocks in PC, workstation, datacom, telecom, and other high-performance applications. The CY2308 has two banks of four outputs each, which can be controlled by the select inputs. Since our systems requires 3 clock inputs(for microprocessor, FPGA and CPLD) Cypress's buffer is a suitable clock driver for our system.

3.2.4.2. Buffers

3.2.4.2.1. Address Buffer

Address bus of the microprocessor is spread out and connected to all memory components; SRAM, DPSRAM and Flash; and FPGA. To increase the driving capacity of address bus and also to decrease the rise time of the address bus a buffer is required. Since address bus is an output port, one directional buffer should be used as address buffer. 74VCX16827 [8] from Fairchild is used as address buffer in Controller PCB.

The 74VCX16827 contains twenty non-inverting buffers with 3-state outputs. It is a byte controlled device with each byte functioning identically, but independent of each other. The control pins may be shorted together to obtain full 16-bit operation. The 3-state outputs are controlled by Output Enable (OEn) inputs.

The 74VCX16827 is designed for low voltage (1.65V to 3.6V) VCC applications with I/O capability up to 3.6V. The 74VCX16827 is fabricated with an advanced CMOS technology to achieve high speed operation while maintaining low CMOS power dissipation.

3.2.4.2.2. Data Buffer

Data bus of the microprocessor is connected to all memory components; SRAM, DPSRAM and Flash; FPGA and CPLD in the Controller PCB. Both to increase the driving capacity and to improve the rise time of the memory components a buffer is required. Data bus is a bi-directional bus, since it is requires both input and output connections. 74VCX16245 [9], from Fairchild, is used as bi-directional data buffer for the Controller PCB. Direction of the buffer is controlled by Read/Write of the microprocessor. When Read is active, direction is into the microprocessor and when Write is active, direction is out of the microprocessor.

The 74VCX16245 contains sixteen non-inverting bi-directional buffers with 3-state outputs and is intended for bus oriented applications. The device is byte controlled. Each byte has separate 3-state control inputs which can be shorted together for full 16-bit operation. The T/R inputs determine the direction of data flow through the device. The OE inputs disable both the A and B ports by placing them in a high impedance state. The 74VCX16245 is designed for low voltage (1.65V to 3.6V) VCC applications with I/O compatibility up to 3.6V. The 74VCX16245 is fabricated with an advanced CMOS technology to achieve high speed operation while maintaining low CMOS power dissipation.

3.2.4.2.3. JTAG Buffer

There are four JTAG compatible components in the Controller PCB; FPGA, microprocessor, CPLD and Configuration EEPROM. Driving capacity of JTAG port is increased with a buffer 74LCX125 [10], from National Semiconductor.

The LCX125 contains four independent non-inverting buffers with 3-state outputs. The inputs tolerate voltages up to 7V allowing the interface of 5V systems to 3V systems.

3.2.5.Memory Components

As all microprocessor systems require nonvolatile memory components for code storage and volatile memory components for variables, Navigation Control and Display Unit Controller Circuit also requires such kind of memory components. For graphical data storage a Dual Port SRAM and for configuration data of Field Programmable Gate Array a configuration memory are also used in Navigation Control and Display Unit Controller Circuit.

3.2.5.1. Flash Memory

There is a need for nonvolatile memory in microprocessor systems to store executable code of the system. In Navigation Control and Display Unit Controller Circuit we use Am29BL162C [11] flash memory from AMD. This flash memory is a 1M*16 bit memory used for not only to keep microprocessor executable code but also to keep system constants and bitmaps.

The Am29BL162C is a 16 Mbit, 3.0 Volt-only burst mode flash memory devices organized as 1,048,576 words. These devices are designed to be programmed in-system with the standard system 3.0-volt VCC supply. The device offers access times of 65 ns, allowing high speed microprocessors to operate with a few or without wait states.

Reading timings are fully compatible with read cycle of ColdFire microprocessor that makes this flash memory suitable in my design.

3.2.5.2. SRAM

A volatile and fast memory is needed in my design for fast memory access. This memory is not only used for variables but also for copying main program to a fast memory. The CY7C1061AV33 [12] high-performance CMOS Static RAM organized as 1,048,576 words by 16 bits, operating at 3.3 V. 10 ns access time makes CY7C1061AV33 a suitable volatile memory for our application.

The other reason of using this SRAM is the compatibility of its write and read memory cycles to the memory cycles of the microprocessor.

3.2.5.3. Dual Port SRAM

Generally a video RAM is required for display interfaces to keep data to be displayed on the screen. Using a Dual Port RAM is generally more meaningful as a video RAM, since it allows both image generation and image refresh to the display at the same time. In my design the microprocessor generates data to be displayed and write them to the Dual Port SRAM (DPSRAM) and FPGA reads the data from the other port of the DPSRAM and prints them to the screen.

IDT70V631S [13], used as dual port SRAM in my design, has true Dual Port memory cells which allow simultaneous access of the same memory location. IDT70V631S has a high speed access time of 12 ns. IDT70V631S supports JTAG features compliant to IEEE 1149.1 which allows easy test of DPSRAM.

The IDT70V631 is a high-speed 256K x 18 Asynchronous Dual-Port Static RAM. The IDT70V631 is designed to be used as a stand-alone 4608K-bit Dual-Port RAM or as a combination MASTER/SLAVE Dual-Port RAM for 36-bit-or-more word system.

This device provides two independent ports with separate control, address, and I/O pins that permit independent, asynchronous access for reads or writes to any location in memory. The 70V631 can support an operating voltage of either 3.3V or 2.5V on one or both ports, controlled by the OPT pins. The power supply for the core of the device (VDD) remains at 3.3V.

Write and read memory cycles are compatible to memory cycles of the microprocessor that make this DPSRAM suitable for my design.

3.2.5.4. FPGA Configuration Memory

Configuration data for the FPGA must be resided in a nonvolatile memory and loaded at configuration time. For this purpose a nonvolatile memory XC18V04 [14],

from Xilinx is used. It is an Electrically Erasable Programmable 4,194,304-bit Serial Memory operating at 3.3V designed to store configuration programs for FPGAs.

M0, M1 and M2 pins of the FPGA select FPGA configuration mode. Applying logical low to these pins sets the configuration to master serial mode. In this mode, CCLK output of the FPGA drives CLK input of the EEPROM. DIN input of the FPGA is connected to the DATA output of the EEPROM. The FPGA accepts the data on each rising edge of the CCLK. After loading of all the configuration data, DONE pin of the FPGA goes to logical high level indicating that configuration of the FPGA is successful.

3.2.6. Communication Components

Navigation Control and Display Unit has to communicate with navigation computer to get navigation data that should be displayed on Navigation Display Unit. Initialization data and control commands entered with the help of keypad also has to be sent to navigation computer. To communicate Navigation Control and Display Unit and navigation computer RS422 and RS232 serial data communication protocols are used. To interface these formats with the microprocessor's UARTs following components are used.

3.2.6.1. RS422 Transmitter

Communication of Navigation Control and Display Unit with navigation computer is provided with RS422 serial communication protocol. Since it uses differential lines for each communication line it is immune to noises and it is suitable for long distance communication. MAX3491E [15] is used as RS422 Transmitter in the system.

MAX3491E is a ±15kV ESD-protected, +3.3V, low-power transceiver for RS-485 and RS-422 communications. Each device contains one driver and one receiver.

3.2.6.2. RS232 Transceiver

A spare RS232 communication channel is put to the system that can be used for short distance serial communication. A RS232 transceiver is required to convert RS232 signal levels to ColdFire UART level. ICL3237 [16] is used as RS232 Transceiver in the system.

The Intersil ICL3237 contains 3.0V to 5.5V powered RS232 transmitters/receivers which meet ElA/TIA-232 and V.28/V.24 specifications, even at VCC = 3.0V. The transmitters are proprietary, low dropout, inverting drivers that translate TTL/CMOS inputs to EIA/TIA-232 output levels.

3.3. High Speed Circuit Design Considerations

3.3.1. Power Distribution

The most important consideration in high-speed board design is the power distribution network. For a noise-free board, it is necessary to have a noise-free power distribution network. Note that it is just as important to develop a clean VCC as it is to get a clean ground. The power distribution network also must provide a return path for all signals generated or received on the board.

3.3.1.1. Power Distribution Network as a Power Source

In PCB designs, the goal is to deliver exactly the same voltage to the power pins of every device on the board, regardless of its position relative to the power source. Furthermore, the voltage at the pins should be free of line noise. A power source with these characteristics would be schematically represented as an ideal voltage source (Figure 3.17a), which has zero impedance. Zero impedance would ensure that the load and source voltages would be the same. It also would mean that noise signals would be absorbed because the noise generators have finite source impedance.

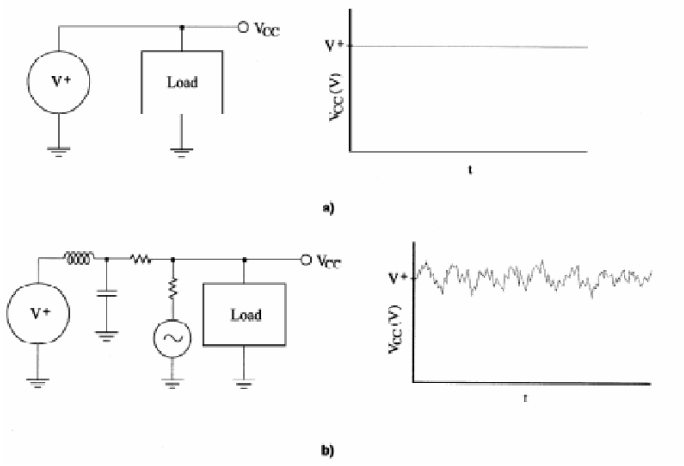

Figure 3.17 The Power Source. a) Ideal b) Realistic

Figure 3.17b illustrates a real power source with associated impedances in the form of resistance, inductance, and capacitance. These are distributed over the power distribution network. Because of the network's impedance, noise signals can add to the voltage.

The design goal is to reduce the power distribution network impedances as much as possible. There are two approaches: power buses and power planes.

Two power-distribution schemes are shown in Figure 3.18. A bus system (Figure 3.18a) is composed of a group of traces with the various voltage levels required by the system devices. For logic, these are typically +5 V and ground. The number of traces required for each voltage level varies from system to system. A power-plane system (Figure 3.18b) is composed of entire layers (or sections of layers) covered with metal. Each voltage level requires a separate layer. The only gaps in the metal are those needed for placing pins and signal feed-through.

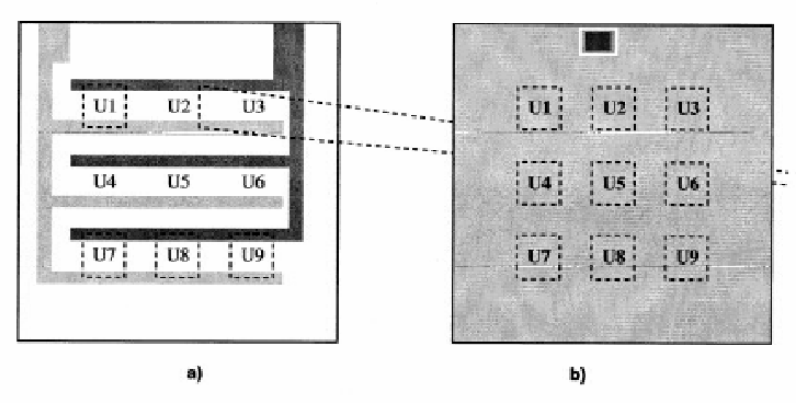

Figure 3.18 Power Distributions System. a) Power Buses; b) Power Planes

Because the power plane fills an entire layer, the only area constraints are the dimensions of the board. The resistance of a power plane is a small fraction of that of a power bus supplying the same number of devices. Thus a power plane is more likely than a bus to supply full power to all the devices.

On a bus, currents are restricted to paths defined by the bus. Any line noise generated by a high-speed device is introduced to other devices on that power bus. On the board in Figure 3.18a, noise generated by U9 is sent to U7 by the bus.

On the power plane, the noise currents are distributed because the current path is not restricted. This, along with lower impedance, makes power planes quieter than power buses [17].

3.3.1.2. Power Distribution Network as Signal Return Path

One of the more surprising functions of the power network is the provision of a return path for all signals in the system, whether generated on or off the board. Designs that accommodate this aspect of the power distribution system eliminate many high-speed noise problems.

Of greatest concern in high-speed design is the energy generated at the signal switching edges. Each time a signal switches, AC current is generated. Current requires a closed loop. As illustrated schematically in Figure 3.19a and 3.19b, the return path needed to complete the loop can be supplied by the ground or VCC. The loop can be represented by Figure 3.19c.

Current loops have inductance and can be thought of as single-turn coils. They can aggravate ringing, crosstalk, and radiation. The current-loop inductance and associated problems increase with loop size. Minimizing the size of the loop minimizes these problems. AC return signals have an entire plane in which to choose a path, but they take the path of least impedance (not necessarily least resistance) to the current.

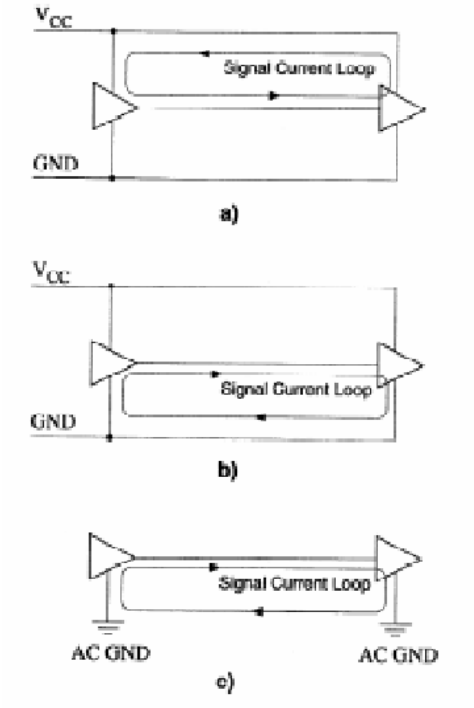

Figure 3.19 Current Loop of a Signal on the Board
a) Through V_{CC}; b) Through Ground; c) The Equivalent AC Path

Figure 3.18a shows that a power bus has a fixed path. The return signal must follow this path, whether optimal or not. Unless the signal lines are purposely laid out near the power buses and oriented to minimize loop size, there will probably be large loops. If the layout of a board using buses for power distribution is not thought out carefully, it can result in a configuration that generates much noise [17].

The power plane imposes no natural restrictions on current flow. Thus the return signal can follow the path of least impedance, which is the path closest to the signal line. This results in the smallest possible current loops, which makes it the preferred solution for high-speed systems.

3.3.1.3. Need for Power Planes

3.3.1.3.1. Impedance Control

If we want to control trace impedance as a strategy for the control of reflections (using proper trace termination techniques), then good, solid, continuous planes are almost always required. It is very difficult to control trace impedance without the use of planes.

3.3.1.3.2. Loop Areas

Loop area can be visualized as the area defined by the path of the signal (traveling down a trace) and its return current. When the return signal is on the plane immediately under the trace, loop area is minimized. Since EMI is directly related to loop area, EMI is minimized when good, solid, continuous planes exist under traces.

3.3.1.3.3. Crosstalk

The two most practical ways to control crosstalk are

(a) Separation between traces and

(b) Closeness of the traces to their reference planes.

Crosstalk is inversely proportional to the square of the distance between the traces and their reference planes.

3.3.1.3.4. Planar Capacitance

The capacitance formed by the proximity of two planes placed close together can be very important and beneficial in circuit decoupling at very high frequencies, where bypass capacitors and their associated mounting and lead inductance begin to have problems. And planar capacitance can be effective in controlling EMI radiations caused by both differential mode and common mode noise signals.

3.3.1.3.5. Ground Bounce

Some circuits result in large signal currents switching rapidly at the same time. Any inductance in the path of these current swings will result in noise voltages being generated across the inductance. The concept of "ground bounce" relates to this issue. Planes generally provide the lowest inductance paths for these currents. Therefore, planes are used to reduce noise when there are high current flows in power distribution systems.

For all reasons explained above Power planes are used in Navigation Control and Display Unit Controller Printed Circuit Board for better system performance. Power plane layers of Navigation Control and Display Unit Controller Printed Circuit Board are shown in Appendix D.

3.3.2. Decoupling Capacitors

The power plane alone does not eliminate line noise. Since all systems generate enough noise to cause problems, regardless of the power distribution scheme, extra filtering is required.

The purpose is not to keep noise on the power planes from getting into your circuit; the purpose is to keep circuit noise from getting onto the power planes.

In high speed, high performance circuits, the noise of interest comes from current transients caused by rapid (fast rise time) switching. These current transients flow on traces and planes. The traces and planes have some finite inductance. Thus, a current pulse with a fast rise time flowing through an inductance creates a voltage transient. The voltage transient affects noise margins for logic devices and radiates from antennas to create electromagnetic interference (EMI) problems.

Since the problem is an inability of the power and ground system to supply enough charge (source of electrons) to meet the initial current requirement during the first few ns of switching, the natural answer is to provide a supply of charge close to where it is needed. That's what a Decoupling capacitor does.

Consider the traces illustrated in Figure 3.20. Trace (a) illustrates the transient current requirement for the chip as logic levels change state. Trace (b) illustrates conceptually the fastest response that the power and ground planes can provide. Trace (c) illustrates what a standard Decoupling capacitor can provide. It can only supply a "burst" of charge, not a long term supply, and it has its own response time. Trace (d) illustrates that a smaller, faster Decoupling capacitor can supply charge faster (because its lead inductance is typically smaller), but it stores much less charge and can't meet the short term requirements [18].

Therefore it seems clear that not only are Decoupling capacitors absolutely necessary in high speed switching circuits for supplying "local" charge, but more than one size

and type of Decoupling capacitor may be necessary to meet all the switching requirements. Navigation Control and Display Unit Controller Printed Circuit Board is designed with these facts in mind.

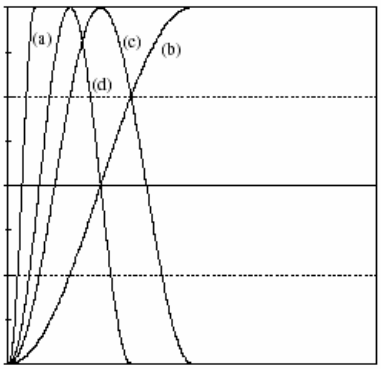

Figure 3.20 **(a) Desired Response**
(b)Response possible from power planes
(c) Additional charge supplied from larger Decoupling cap
(d) Charge available from smaller, faster decoupling cap

CHAPTER 4

SOFTWARE DEVELOPMENT

4.1. Overview

In this chapter, functions of the Navigation Control and Display Unit Controller Circuit and the implementation of these functions are explained. Main functions of the Controller Circuit are generation of visual data to be displayed on TFT display, driving the TFT Display, communication with Navigation Computer and providing keypad interface for initialization and menu interface. Block diagram showing the digital functions of Controller Circuit are shown in Figure 4.1.

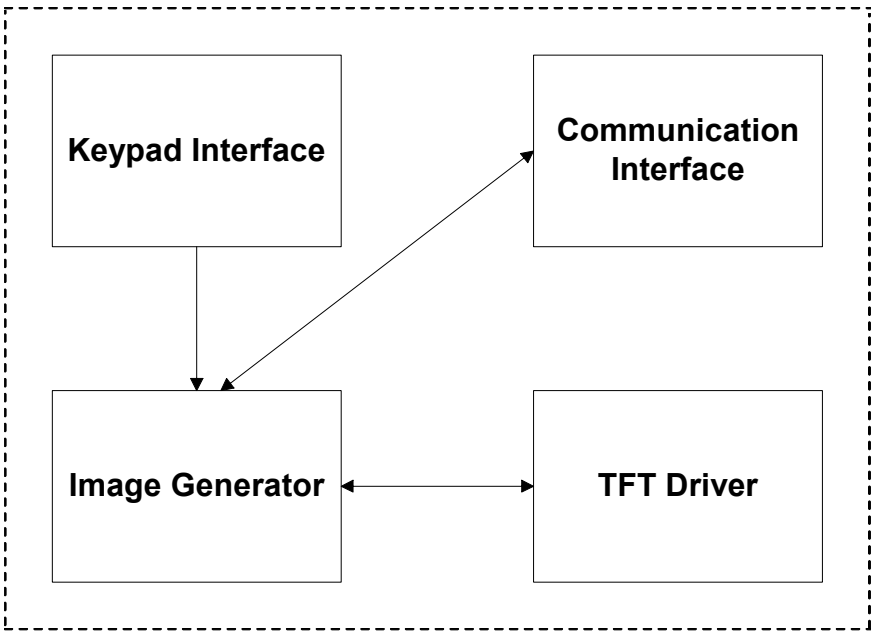

Figure 4.1 Functional Block Diagram of Navigation Control and Display Unit Controller Circuit

Digital functions of the Controller Circuit are mainly implemented with FPGA and ColdFire microprocessor. Concurrent and continuous high speed processes are implemented with FPGA and sequential, long but not time critical functions are implemented with the microprocessor. Implementation of FPGA functions are given in Chapter 3. In this section software development of the microprocessor will be given.

Microprocessor software design starts with the analysis of the systems requirements. Boot code describing the hardware configuration of the system is written. System functions are divided into sub functions. Sub functions are implemented using assembler or C++ description languages. Compiler generates object files from the source files. Linker combines and generates executable file to be downloaded to a nonvolatile memory. Executable file is downloaded to the nonvolatile memory with a programmer. Code functionality is observed and tested with the help of a debugger and verification is provided. Design flow for microprocessor software is given in Figure 4.2.

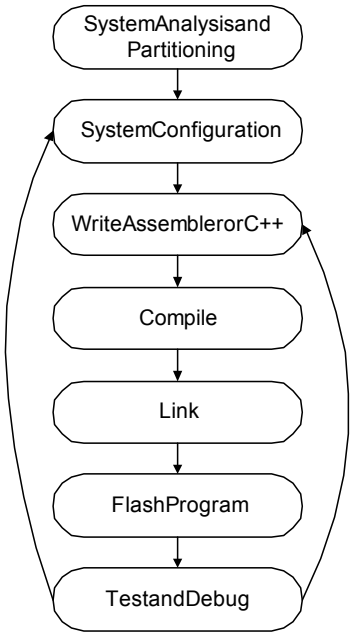

Figure 4.2 Microprocessor Software Design Flow

4.2. Image Generation

Images displayed on the screen of the TFT display are generated with the cooperation of FPGA, microprocessor and a Dual Port SRAM (DPSRAM) that is used as a video memory in the system. All display data; like characters, background images, graphics and bitmaps are generated by the microprocessor. Generation details for these digital image data is described in section 4.3. Microprocessor generates a screen shoot as a sequence of digital Red-Green-Blue (RGB) image data. This generated screen shoot called as frame is written to DPSRAM. Since display is a 320*240 pixel display a frame contains 76800 sequential digital image data. Each data corresponds to a pixel of the TFT. Each pixel is represented as a 15 bit image data showing the color of the pixel in RGB format. Microprocessor generates frames and sends frame number to be displayed to the FPGA. With the frame number information FPGA reads the image sequence from the other port of the DPSRAM and drives the TFT with the current image. According to the communication channel and keypad information microprocessor decides what kind of data to be displayed on the screen. When the image displayed on the display needs to be changed microprocessor generates the frame and after finishing the frame generation tells FPGA to read and send new screen shoot corresponding to the current frame number. Graphical description of image generation process is presented in Figure 4.3.

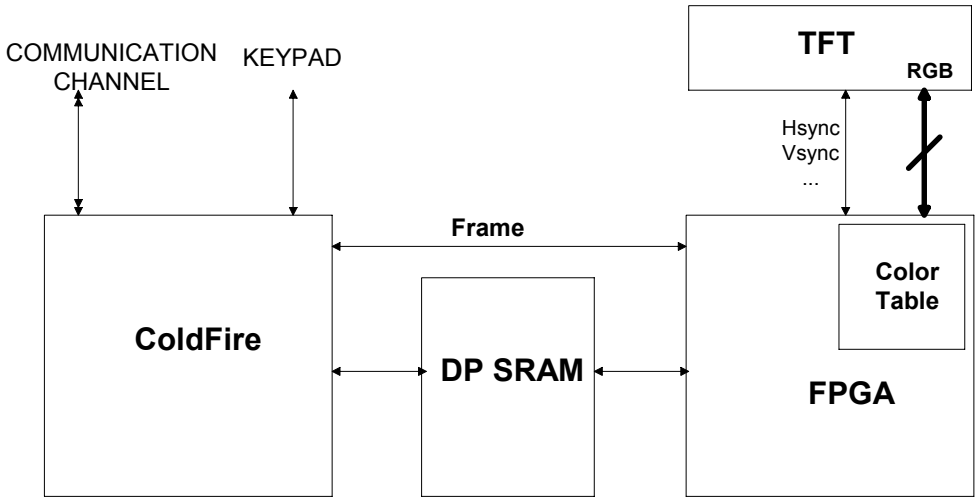

Figure 4.3 Image Generator Interface

There are three main frames in DPSRAM. Active frame represents the frame shown on the screen, passive frame shows the frame that is currently prepared by the processor and will be displayed soon and Background or Image frame is generally bitmap frames that are slowly generated and rarely changed. Different frames are located at different positions of the DPSRAM. Figure 4.4 shows frames located at DPSRAM.

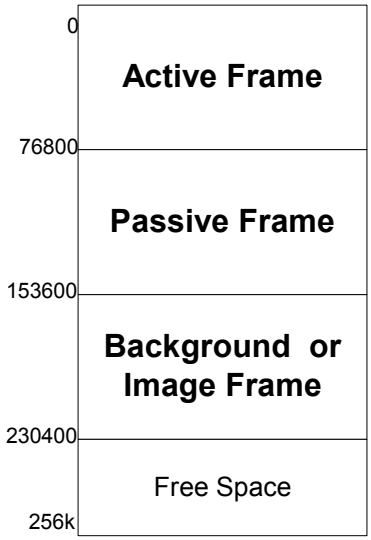

Figure 4.4 DPSRAM Frames

4.3. Microprocessor Functions

ColdFire microprocessor used in the system is responsible for communicating with navigation computer, getting keypad commands and after interpreting these messages and commands generation of graphical image for the TFT display. Interfacing with FPGA, CPLD and system memories are also achieved by the processor. All required interfaces for the microprocessor are shown in Figure 4.5. For all interfaces some codes are written. Following subsections describe the functionality of these codes.

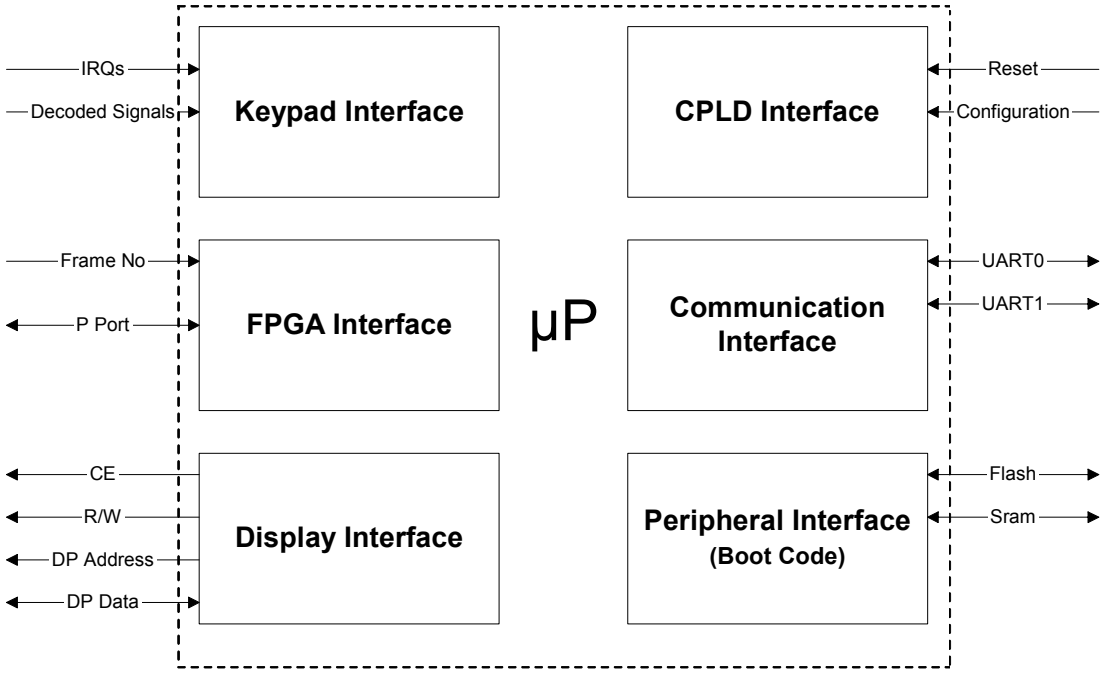

Figure 4.5 Block Diagram of Microprocessor Functions

4.3.1.Peripheral Interface

For every microprocessor a boot code describing the peripherals and memory map of the system have to be written. The boot code for our system do the following works so that our system is ready for required functionality.

4.3.1.1. System Integration Module Initialization

The System Integration Module (SIM), shown in Figure 4.6, provides overall control of the bus and serves as the interface between the ColdFire core processor complex and the internal peripheral devices.

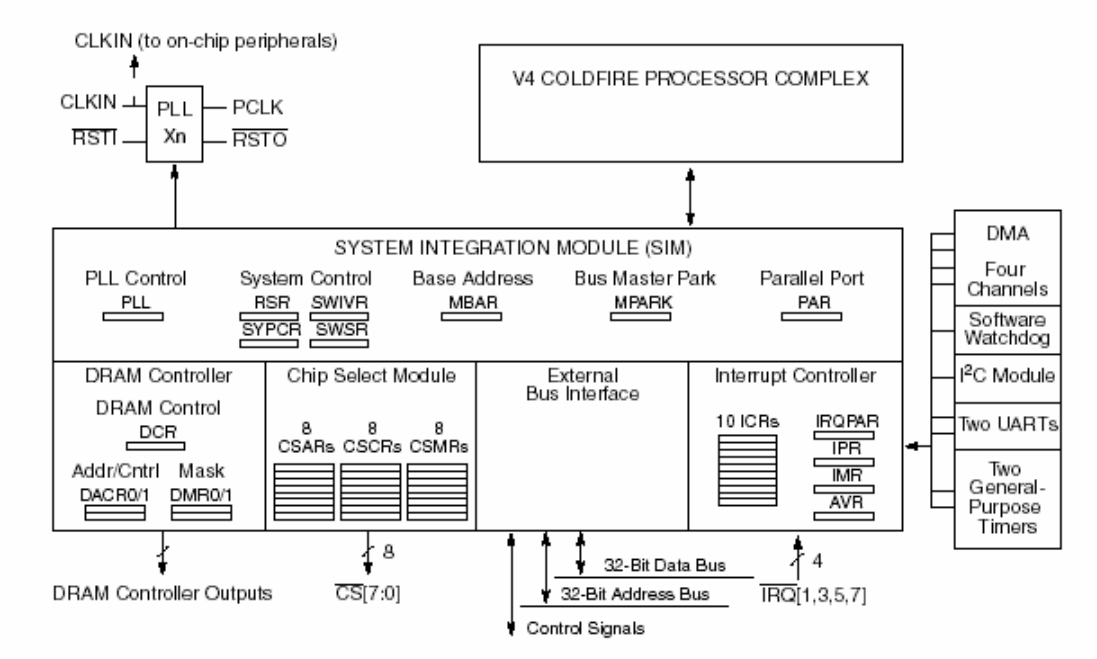

Figure 4.6 System Integration Module

With SIM initialization code internal peripherals are placed to a location in the memory map to be called later and initial configuration of these peripherals are provided. For the microprocessor system, Module Base Address Register (MBAR) is set to 0x10000000 address and all internal peripherals are located by some offset from these MBAR value. Software watchdog timer is disabled so that it does not reset the system when it overflows. All parallel port pins are defined as general purpose input/output pins since they will be used later as communication channel with FPGA. All interrupt vectors are masked and autovector values for the interrupt sources like UARTs, Timers and DMAs are set. Chip selects for external memories are also programmed with this initialization process.

Memory map of the system after SIM initialization is given in Figure 4.8. Boot code is given in Appendix C for details of these initializations. Software flow for System Integration Module initialization is given in Figure 4.7.

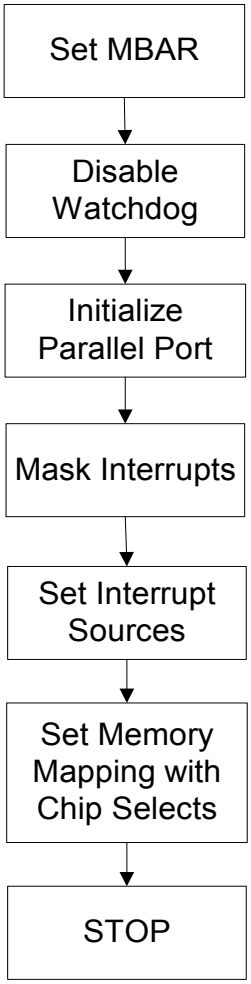

Figure 4.7 Software Flow for SIM Initialization

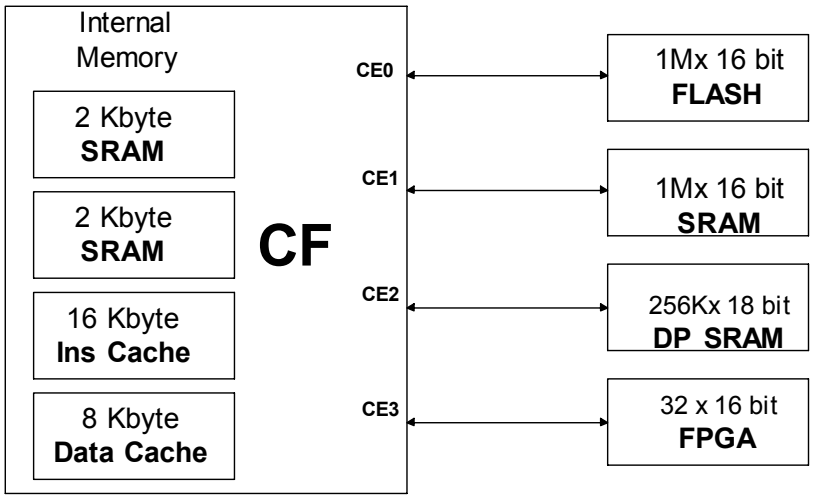

Figure 4.8 Memory Map for SIM

4.3.1.2. Peripheral Module Initialization

Peripheral Module initialization process initializes parallel ports, DMA Controllers, Timers and UARTs of the system. Parallel Ports of the system are set as inputs initially not to drive any lines connected to parallel port. UART is configured as both receiver and transmitter for 8 bit data transmission with no parity and one stop bit at 38400 baud rate. All interrupts related to UART are disabled. Software flow for Peripheral Module initialization is given in Figure 4.9.

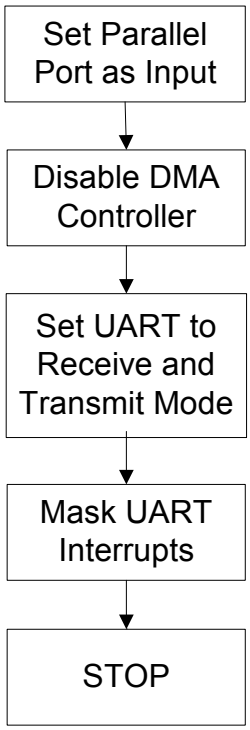

Figure 4.9 Software Flow for Peripheral Module Initialization

4.3.2. CPLD interface

CPLD interface of the microprocessor provides system reset and initial configuration of the microprocessor. Boot code of the system resides at Flash memory but microprocessor does not know anything from software about memories until boot code is executed. Till initialization process some knowledge about the flash of the

system are given as hard signals externally by CPLD. Configuration signals provide information about port size of flash, wait count for read of flash, clock frequency to be used internally.

4.3.3.FPGA Interface

FPGA function of microprocessor is used to send frame number to be displayed on the screen. Details of framing were introduced at section 4.2. When a frame is prepared by the microprocessor and decided to be printed on the screen microprocessor puts the corresponding frame number to the corresponding bits of the parallel port of the microprocessor. FPGA reads the value of frame number and refresh the screen with corresponding frame view.

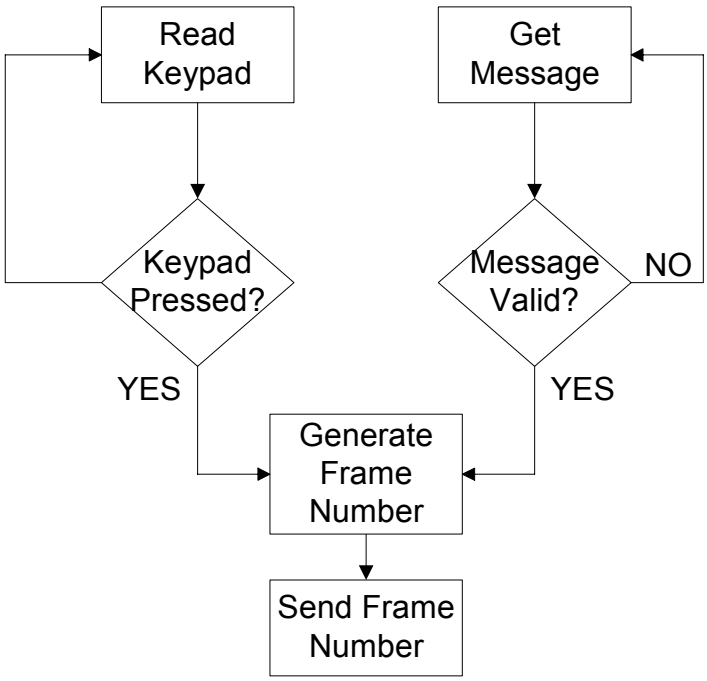

Figure 4.10 Software Flow for FPGA Interface

4.3.4. Keypad Interface

Keypad interface of microprocessor consist of an interrupt service routine and a keypad interface function that updates the display according to meaning of decoded keypad signal.

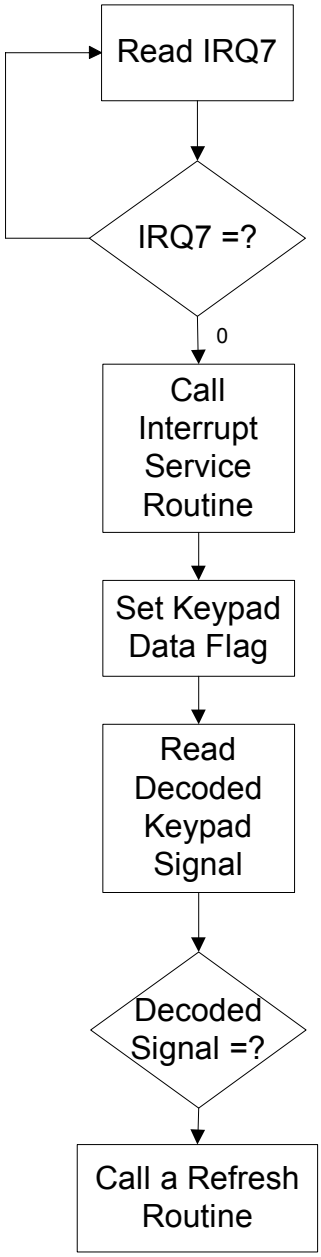

Figure 4.11 Software Flow for Keypad Interface

Since polling the keypad signals continuously may result in reduction of microprocessor performance, interrupt port of microprocessor is used for keypad action detection. When a keypad button is pressed a level 7 interrupt is generated by FPGA to warn the microprocessor. With a keypad press IRQ7 pin of microprocessor is set to logic level 0 which causes a call to level 7 interrupt service routine. Interrupt service routine sets a flag showing a keypad action has taken. Then keypad interface function checks the value of decoded keypad signals from the parallel port of microprocessor. According to current state of menu and button pressed, keypad action gets a meaning. Then corresponding update function is called as shown in Figure 4.11.

4.3.5. Communication Interface

Communication of Controller Unit with navigation computer is provided with UARTs of the microprocessor. UART interface of microprocessor is provided with communication interface part of microprocessor software.

Communication interface communicates with navigation computer asynchronously to get navigation information of the moving platform. Communication is provided with communication packages of about 100 bytes and with a baudrate of 38200. Communication packages are reaches to microprocessor with a frequency of about 1 Hz. Communication is provided with RS422 serial communication protocol.

A communication package consists of some sections. A communication package starts with header bytes which are used to detect beginning of messages. Then command bytes tell the microprocessor what kinds of actions to be taken. Information bytes provide system information like altitude, latitude of the system, control point number etc. Finally check sum bytes are provided which is used to control if the full set of message is reached without error. Sum of all bytes including check sum values must be zero for a communication package to be meaningful. If communication package is validated, then commands in the command bytes are executed. Software flow for communication interface is given in Figure 4.12.

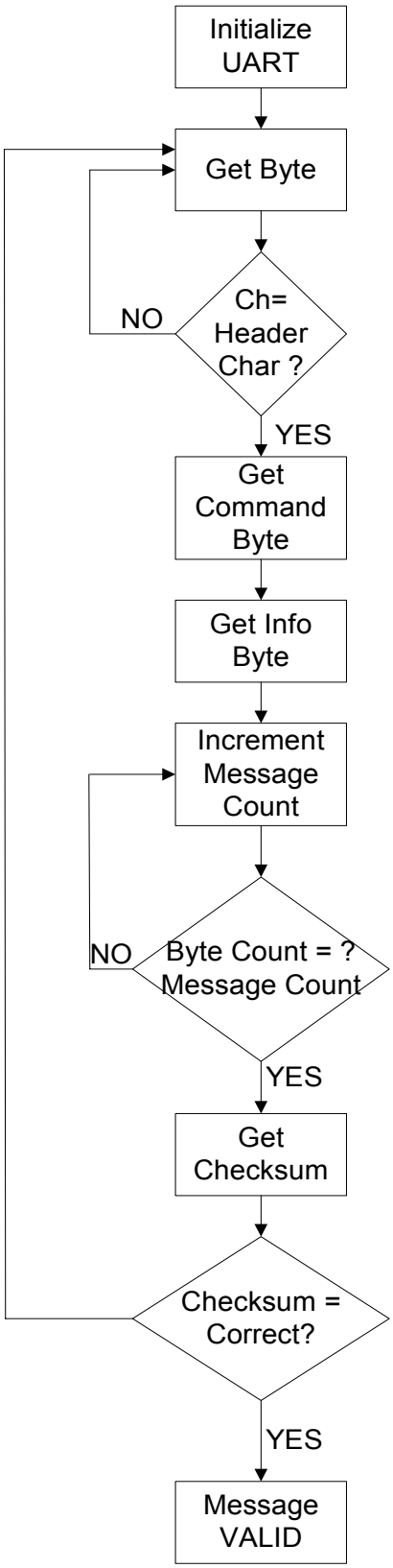

Figure 4.12 Software Flow for Communication Interface

4.3.6. Display Interface

Display interface is the part of microprocessor that is responsible for image generation to be displayed on the TFT display. Lots of functions are implemented and a few software tools are created in computer for image generation.

Images are generated as a sequence of pixel color information at DPSRAM. To get a still image on the display, every pixel should be driven with a color data. For character or a visual image generation, color information of each pixel should be properly set. Since a 320*240 TFT display is used in this Controller System, pixel color information of 240 lines are sequentially put to DPSRAM as an array of 320 elements. Full set of these 320*240 data is called a frame.

There is a mapping as shown in Figure 4.13 between a pixel of TFT and the address of the frame.

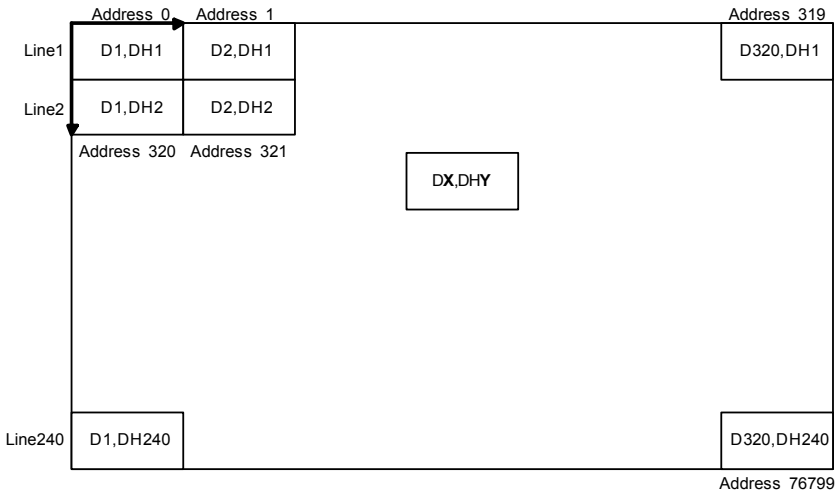

Figure 4.13 TFT Display- Memory Address mapping

For image generation a pointer is generated in software that points to a pixel of the display. While generating images, the only thing for a pixel to highlight to a color is

moving the pointer to desired position in the screen using this mapping and putting the required color data to corresponding memory address of the frame.

4.3.6.1. Display Functions

4.3.6.1.1. Move Pointer

Move pointer is a function that takes position and frame number information and calculates corresponding DPSRAM memory address and points to that location. Before the write function , move pointer function must be called.

4.3.6.1.2. Display Character

Display character function is used to print an alphanumeric character to display. Move pointer function is called to bring the pointer to desired position and display character function is called. Different sizes of characters may be displayed on the screen. Assume a 12*16 character will be displayed on the screen. Characters are displayed on the screen with their background and foreground colors as shown in Figure 4.14. In Figure 4.14 white color shows the foreground color of the character ‘G’ and gray color shows the background color of the character.

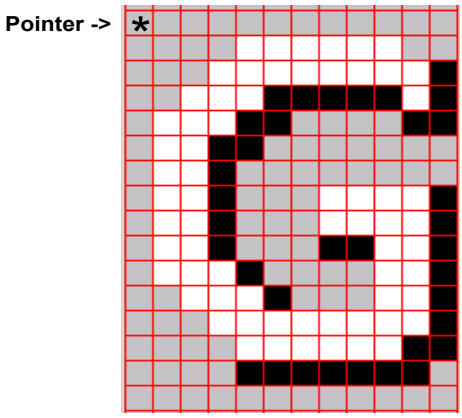

Figure 4.14 A Character shown on the screen

Characters are stored in flash memory as array of binary data. A zero means a background color should be displayed for that pixel and one means a foreground color should be displayed for the corresponding pixel. Display character function reads the first line of array constant and prints foreground or background color to the TFT display according to binary value and then every print pointer is moved to next address. At the end of the line, pointer is moved to next line. All lines of characters are printed in the same manner.

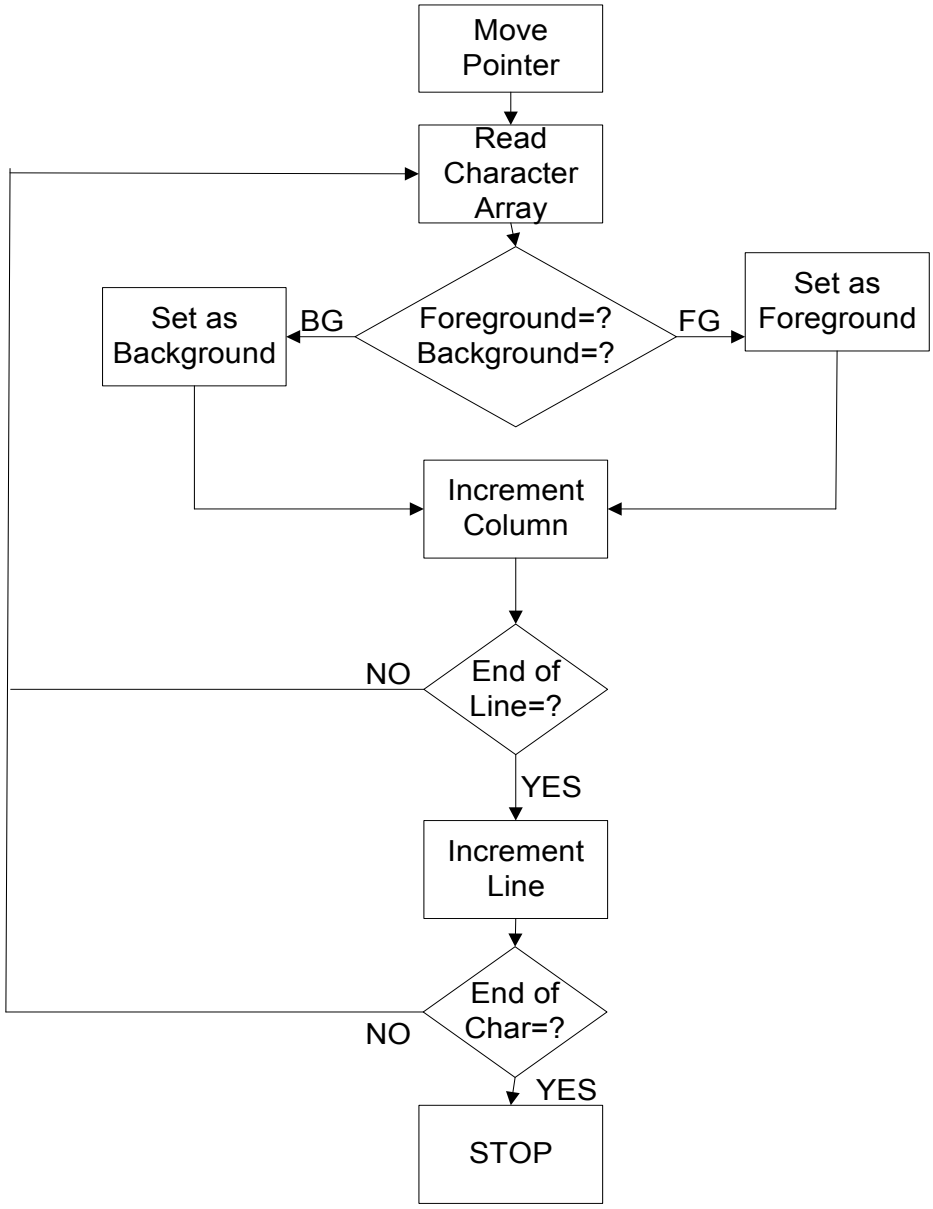

Figure 4.15 Software Flow for Display Character

4.3.6.1.3. Character Generator

At the last section, we assumed to print an already defined character to screen. Definition of character set is the most difficult part of displaying a character actually. Character generation function is implemented with a character generator tool that is prepared in C Builder software.

Character Generator tool is a graphical interface as shown in Figure 4.16 that runs on a PC and let user to define a character or a set of characters.

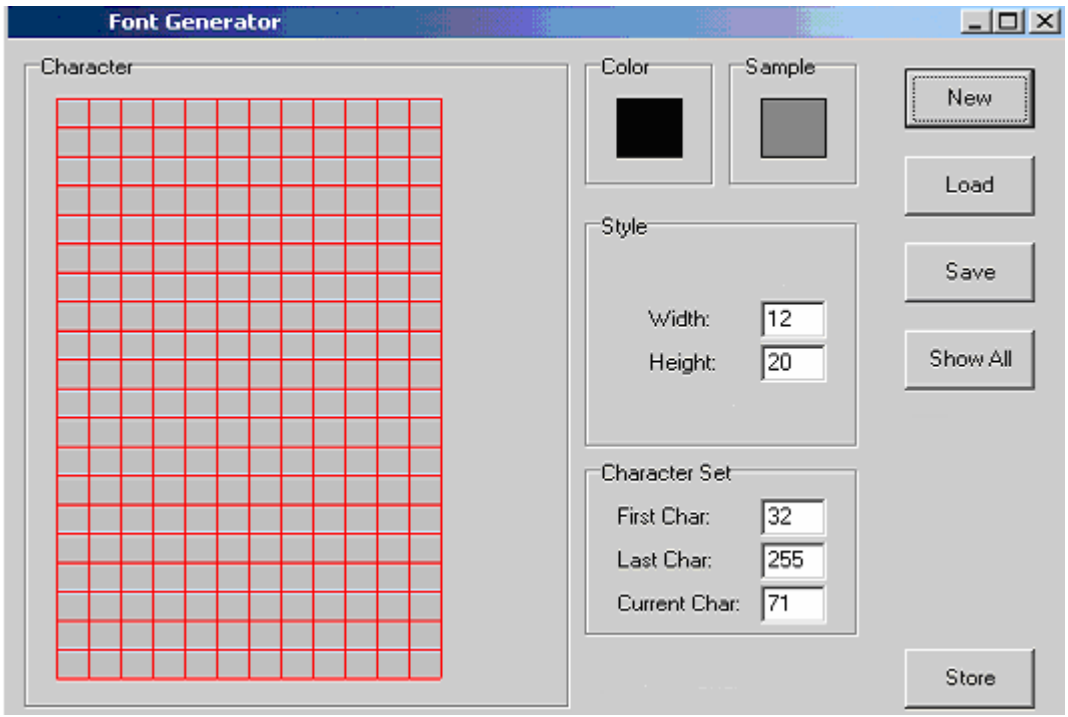

Figure 4.16 Character Generator Graphical Interface

Font size is entered from "Style" section and character number is entered from "Character Set" section. Pixels intended as foreground are painted with mouse to white color. Character is added to character set using "Store" button. Generating all characters and saving this character set generates an assembler file that stores these characters as an array of binary data as shown in Figure 4.18. Figure 4.17 shows full set of characters defined in 12*20 font size.

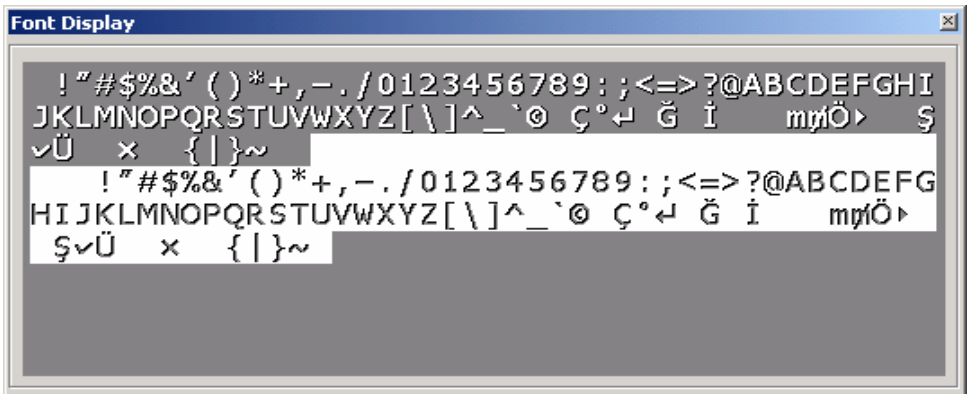

Figure 4.17 Character Set

```
; Character 67 (C)
    .word  0000000000000000000000000b
    .word  0000000000000000000000000b
    .word  0000000000000000000000000b
    .word  0000000001010101010000000b
    .word  0000000101010101010011000b
    .word  0000010101101010100110000b
    .word  0001010110100000001010000b
    .word  0001011010000000000000000b
    .word  0001011000000000000000000b
    .word  0001011000000000000000000b
    .word  0001011000000000000000000b
    .word  0001011000000000000000000b
    .word  0001010110000000000000000b
    .word  0000010101100000000011000b
    .word  0000000101010101010011000b
    .word  0000000001010101011010000b
    .word  0000000010101010101000000b
    .word  0000000000000000000000000b
    .word  0000000000000000000000000b
    .word  0000000000000000000000000b
```

Figure 4.18 A character array

4.3.6.1.4. Display String

Display string is a function that takes a string and prints it as a sequence of characters. Before calling this function, a move pointer function call is required to print the string to desired position of the screen.

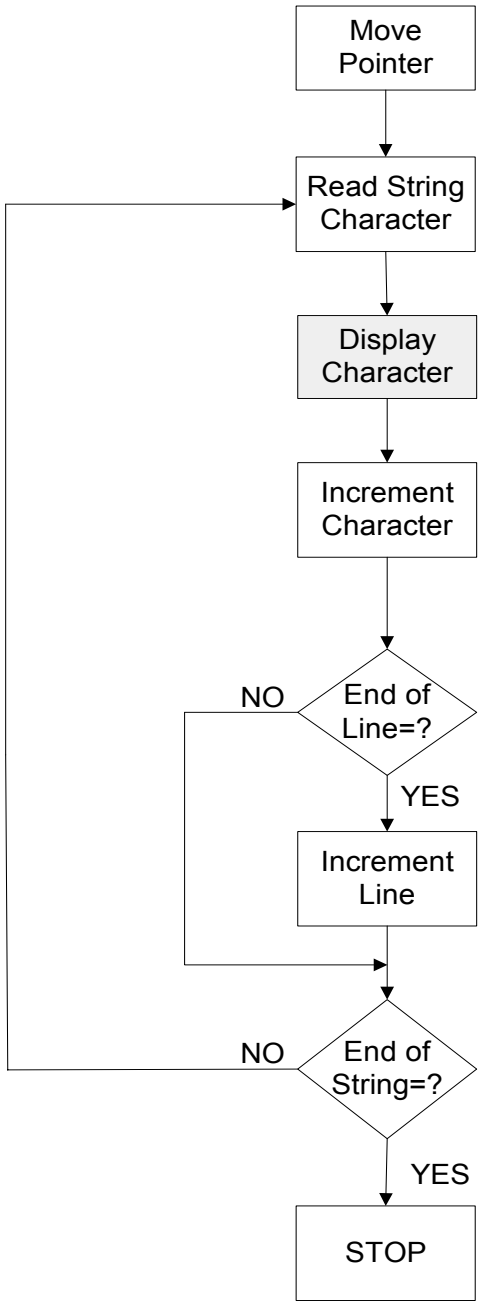

Figure 4.19 Software Flow for Display String

4.3.6.1.5. Display Bitmap

Display bitmap function displays an early stored bitmap to screen. Size of the bitmap, address of the bitmap and the position information on the screen must be

entered to call function. Figure 4.20 shows software flow for display bitmap function.

Bitmaps are saved on the flash memory as a sequence of pixel color information and depending on the size of the bitmap these prestored data are read and printed to the screen. Storing bitmaps to flash requires a tool described in the following section that generates bitmap arrays in assembler format.

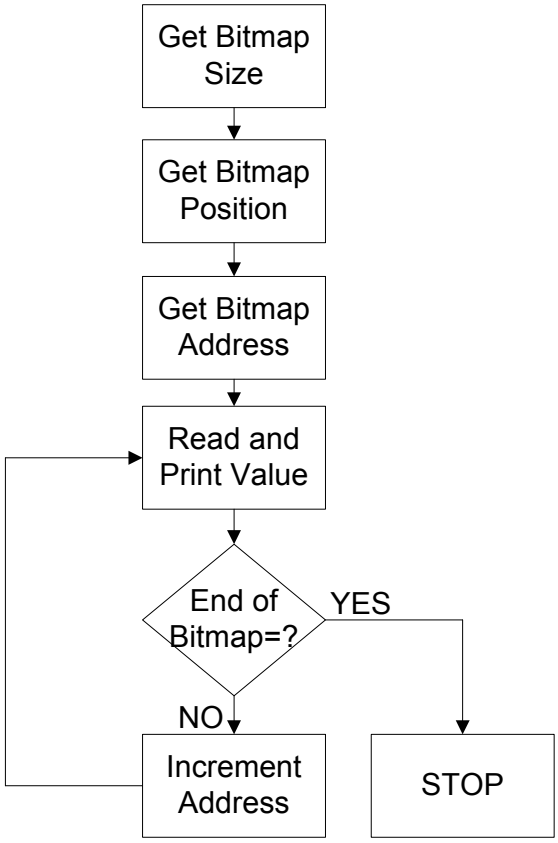

Figure 4.20 Software Flow for Display Bitmap

4.3.6.1.6. Bitmap Generator

Bitmap Generator is a graphical interface tool is generated at Borland C Builder. With “Open Bitmap” button it opens standard bitmaps images and shows them on computer screen. You do not have to work with opened bitmap. You can take a

sample part of it and work with it. “Shows Grids” button shows grids on the screen that helps you to decide which sections of the image should be seen. Entering X start and Y start coordinates and pressing “Sample Between” button will sample a 320*240 image starting from entered coordinates. “Save as txt” button prepares image to be displayed on TFT screen and stores the image as an assembler file.

Graphical interface of Bitmap Generator is shown in Figure 4.21.

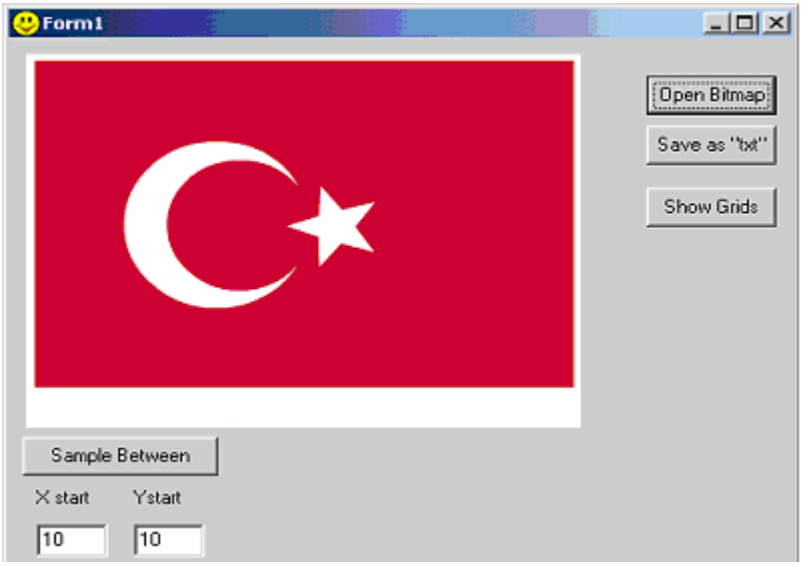

Figure 4.21 Bitmap Generator Graphical Interface

CHAPTER 5

OPERATION OF NAVIGATION CONTROL AND DISPLAY UNIT

This section describes the operation of Navigation Control and Display Unit and how to manipulate on menu items.

With initial power on a welcome screen appears for a few seconds on the screen as shown in Figure 5.1. Meanwhile, processor makes initial adjustments of the system.

Figure 5.1 Welcome Screen

After the system is settled Main Menu screen appears that lets you make a selection on present menus as shown in Figure 5.2. Using Up and Down keypad buttons, user can navigate on Main Menu screen and select one of the active menu item with Enter keypad button. While navigating on the menu, active menu item highlights.

Figure 5.2 Main Menu

Some menu items have submenu items, which appear when these menu items are selected as shown in Figure 5.3.

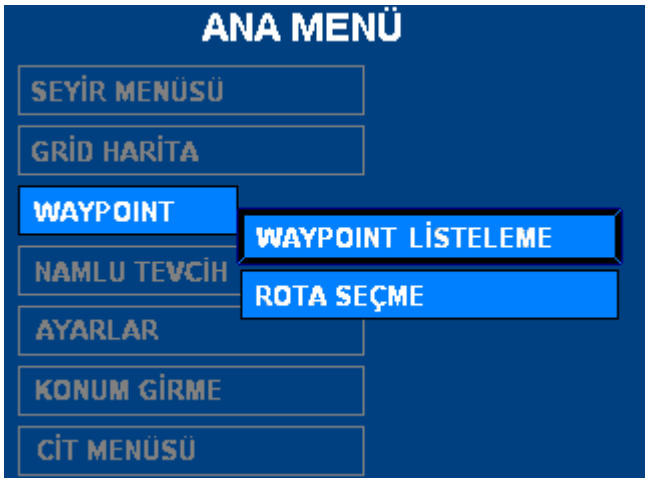

Figure 5.3 Submenu Items

Some of the menu items, for example position initialization, require data entrance to the system. Since keypad of Driver Display Unit has just four buttons, entrance of alphanumeric characters is achieved with an imaginary keypad as shown in Figure 5.4. User can navigate on imaginary keypad using Up and Down keypad buttons and select active character with Enter keypad button.

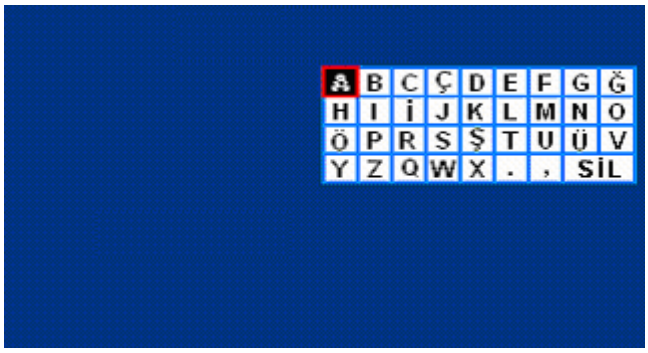

Figure 5.4 Imaginary Keypad

Most frequently used menu item of Driver Display Unit is Navigation Menu, as shown in Figure 5.5, which prints navigation data obtained through the communication with the navigation computer. Navigation Menu continuously updates values on the screen and also prepares graphical data shown on the right side of Navigation Menu.

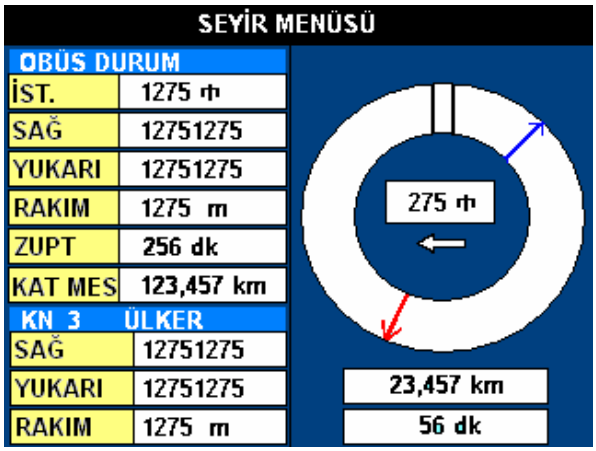

Figure 5.5 Navigation Menu

Simulation of the system communication with navigation computer is provided with a simulation program that sends imaginary navigation data to the Driver Display Unit. User interface of simulation program are shown in Figure 5.6. The simulation program sends the navigation data to the interface unit in the same protocol and format of navigation systems. With simulation program few waypoints are defined

for the interface unit and an imaginary vehicle start to moving on the map. During the motion, navigation data of the vehicle is sent to interface unit. Status flags of accelerometers, gyroscopes, INS, GPS and temperature are also sent to show the current status of the navigation system.

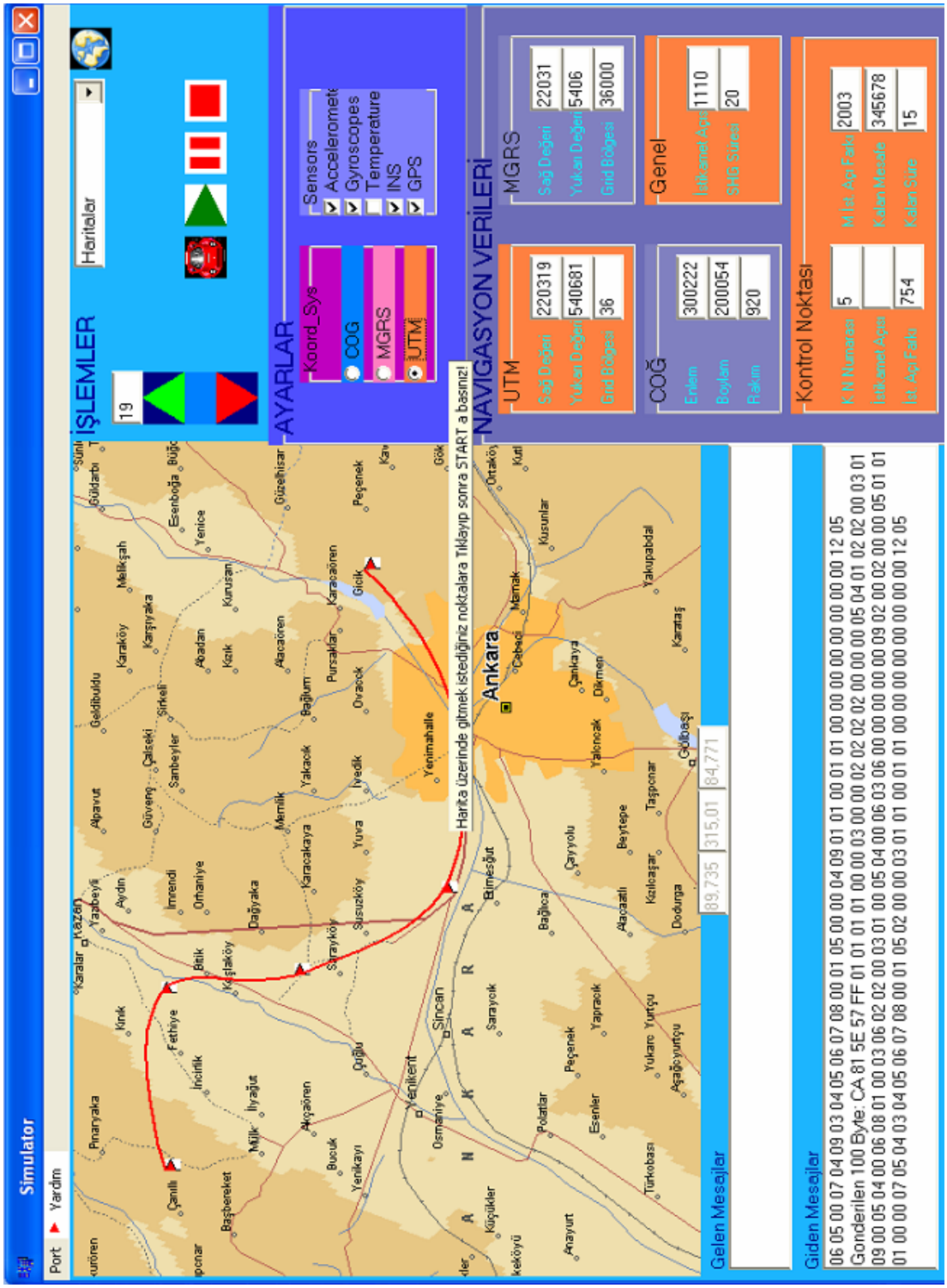

Figure 5.6 Simulation Software Interface

CHAPTER 6

CONCLUSIONS

The aim of this thesis was the design and implementation of a microprocessor based interface unit of a navigation system, providing graphical interface to the driver of the vehicle containing the navigation system. Interface unit communicates with a navigation computer, gets navigation information and shows them on a graphical TFT display. Implementation of the thesis required the design of the Navigation Control and Display Unit Controller Circuit and implementation of system functions.

Design of the Navigation Control and Display Unit Controller Circuit has required detailed work on system requirements. With detailed examination of system requirements and functions, architectural design of the Navigation Control and Display Unit Controller Circuit was achieved. Selection of components, that would work properly together and meeting system requirements, has required detailed examination of available candidate products. A microprocessor and a programmable logic device, Field Programmable Gate Array (FPGA), were mainly needed in this design. Motorola ColdFire MCF5407 microprocessor is used in design. Fast rise and fall times of the processor has required high speed Printed Circuit Board design techniques for the Navigation Control and Display Unit Controller PCB design. After prototype PCB Design, simulation of the design was made with a PCB simulation tool. By examining simulation results, required changes on the Navigation Control and Display Unit Controller Printed Circuit Board had been made and the design was finalized.

Digital functions of the Navigation Control and Display Unit Controller Circuit are mainly implemented with FPGA and ColdFire microprocessor. Concurrent and continuous high speed processes are implemented with FPGA and sequential, long

but not time critical functions are implemented with the microprocessor. Communication, image generation and peripheral interface functions were implemented in the microprocessor using assembler and C programming languages. TFT driver and keypad interface functions were implemented in FPGA using Very High Speed Integrated Circuit Description Language (VHDL).

Navigation Control and Display Unit is successfully tested with a simulation program that is sending imaginary navigation data to the interface. Simulation program sends the navigation data to the interface unit in the same protocol and format of navigation systems.

REFERENCES

[1] NATO Research and Technology Organization, RTO-SET-054/RTG-30 Basic Guide to Advanced Navigation, 2004

[2] SHARP, Device Specification for TFT-LCD module LQ038Q5DR01 available at www.sharp.co.jp/products/device/lineup/data/pdf/datasheet/ LQ038Q5DR01_LCY01004B.pdf, 2003

[3] MAXIM, MAX1830/MAX1831 3A, 1MHz, Low-Voltage, Step-Down Regulators with Synchronous Rectification and Internal Switches Data Sheet available at pdfserv.maxim-ic.com/en/ds/MAX1830-MAX1831.pdf , 2001

[4] Motorola, MCF5407UM MCF5407 ColdFire® Integrated Microprocessor User's Manual available at www.freescale .com, 2001

[5] XILINX, Virtex-II FPGAs Complete Data Sheet available at http://direct.xilinx.com/bvdocs/publications/ds031.pdf, 2004

[6] XILINX, CoolRunner-II CPLD Family Data Sheet available at http://direct.xilinx.com/bvdocs/publications/ds090.pdf, 2004

[7] CYPRESS, CY2308 3.3 V Zero Delay Buffer Data Sheet available at www.cypress.com/cfuploads/img/products/cy2308.pdf, 2004

[8] FAIRCHILD, 74VCX16827 Low Voltage 20-Bit Buffer/Line Driver with 3.6V Tolerant Inputs and Outputs Data Sheet available at www.fairchildsemi.com/ds/74/74VCX16827.pdf, 2002

[9] FAIRCHILD, VCX16245 Low Voltage 16-Bit Bidirectional Transceiver with 3.6V Tolerant Inputs and Outputs Data Sheet available at www.fairchildsemi.com/ds/74/74VCX16245.pdf, 2002

[10] FAIRCHILD, LCX125 Low Voltage Quad Buffer with 5V Tolerant Inputs and Outputs Data Sheet available at www.fairchildsemi.com/ds/74/74LCX125.pdf , 2004

[11] AMD, Am29BL162C 16 Megabit (1 M x 16-Bit) CMOS 3.0 Volt-only Burst Mode Flash Memory Data Sheet available at www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22142f5.pdf, 2002

[12] CYPRESS, CY7C1061AV33 1M * 16 Static RAM Data Sheet available at www.cypress.com/cfuploads/img/products/CY7C1061AV33.pdf, 2003

[13] IDT, IDT70V631S High-Speed 3.3V 256K * 18 Asynchronous Dual-Port Static RAM Data Sheet available at www.idt.com/docs/70V631_DS_87343.pdf, 2003

[14] XILINX, XC18V00 Series In-System Programmable Configuration PROMs Data Sheet available at www.xilinx.com/isp/compfpgatables/prom_virtex.htm, 2004

[15] MAXIM, MAX3491E, 3.3V-Powered, ±15kV ESD-Protected, 12Mbps and Slew-Rate-Limited True RS-485/RS-422 Transceivers Data Sheet available at pdfserv.maxim-ic.com/en/ds/MAX3483E-MAX3491E.pdf, 1999

[16] Intersil, ICL3237, 1 Microamp Supply-Current, +3V to +5.5V, 1Mbps, RS-232 Transmitters/Receivers Data Sheet available at www.intersil.com/data/fn/fn6003.pdf, 2004

[17] ASELSAN, High Speed PCB Design Guide: Power Planes, 2002

[18] ASELSAN, High Speed PCB Design Guide: Decoupling Capacitor, 2002

APPENDIX A : TFT SIGNAL WAVEFORM and TIMINGS

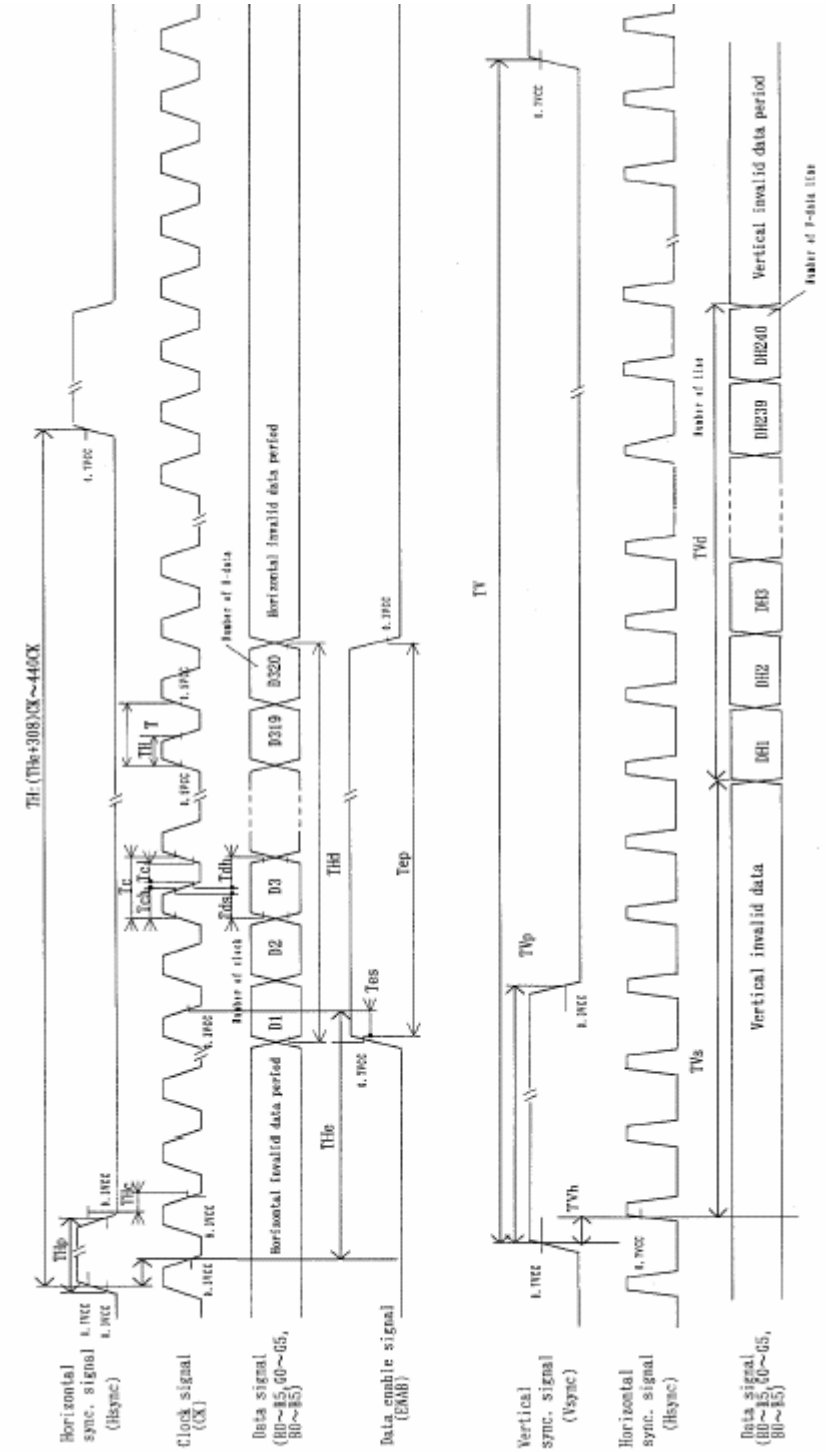

Figure A.1 TFT Signals waveform

Table A.1 TFT Display timing constants

Parameter		Symbol	MIN	MAX	TYPICAL	Unit
Clock	frequency	1/Tc	4,5	6,3	6,8	MHz
	High Time	Tch	50	-	-	ns
	Low Time	Tcl	50	-	-	ns
Data	Setup Time	Tds	50	-	-	ns
	Hold Time	Tdh	50	-	-	ns
Hsync-Clock phase difference		THc	50	-	120	ns
Hsync-Vsync phase difference		TVh	0	-	TH-10	us
Horizontal Sync. Signal	cycle	TH	50	63,5	80	us
			TH e+308	400	440	clock
	pulse width	THp	4	12	30	clock
Enable Signal	Setup Time	Tes	50	-	Tc-10	ns
	pulse width	Tep		320		clock
Hsync-Enable phase difference		TH e	14	-	72	clock
Horizontal Display period		THd	320	320	320	clock
Vertical Sync. Signal	cycle	TV	246	263	330	line
	pulse width	TVp	1	-	-	line
Vertical Display start position		TVs	6	6	6	line
Vertical Display period		TVd	240	240	240	line

APPENDIX B : VHDL CODES FOR FPGA FUNCTIONS

Horizontal Synchronization Generator Block VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Hsync_Generator IS
  PORT(
    clk             : IN     std_logic;
    rst             : IN     std_logic;
    valid_data      : IN     std_logic;
    ENAB            : OUT    std_logic;
    Hsync_buf       : OUT    std_logic;
    data_read_enable : OUT    std_logic;
    data_send_enable : OUT    std_logic;
    tft_clk_buf     : OUT    std_logic);
END Hsync_Generator ;
ARCHITECTURE fsm OF Hsync_Generator IS
  SIGNAL tft_clk_count    : std_logic_vector(8 downto 0);
  SIGNAL clk_count        : std_logic_vector(4 downto 0);
  SIGNAL ENAB_ok          : std_logic;
  TYPE STATE_TYPE IS (initialisation, s0, s1, s2, s3, s4, s5, s6, s7, s8,s9, s10, s11, s12, s13, s14   );
  ATTRIBUTE state_vector : string;
  ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state" ;
  SIGNAL current_state : STATE_TYPE ;
  SIGNAL next_state : STATE_TYPE ;
  SIGNAL ENAB_cld : std_logic  ;
  SIGNAL Hsync_buf_cld : std_logic  ;
  SIGNAL data_read_enable_cld : std_logic  ;
  SIGNAL data_send_enable_cld : std_logic  ;
  SIGNAL tft_clk_buf_cld : std_logic  ;
BEGIN
  --------------------------------------------------------------------------
  clocked : PROCESS( clk, rst )
  --------------------------------------------------------------------------
  BEGIN
    IF (rst = '0') THEN
```

```
    current_state <= initialisation;
     ENAB_cld <= '0';
    Hsync_buf_cld <= '0';
    data_read_enable_cld <= '0';
    data_send_enable_cld <= '0';
    tft_clk_buf_cld <= '0';
    ENAB_ok <= '0';
    clk_count <= "00000";
  ELSIF (clk'EVENT AND clk = '1') THEN
    current_state <= next_state;
     data_read_enable_cld <= '0';
    data_send_enable_cld <= '0';
    CASE next_state IS
    WHEN initialisation =>
      tft_clk_buf_cld <= '0';
      Hsync_buf_cld <= '0';
      ENAB_cld <= '0' ;
      data_read_enable_cld <= '0';
      data_send_enable_cld <= '0';
      clk_count <= (others => '0');
      tft_clk_count <= (others => '0') ;
      ENAB_ok<='0';
    WHEN s0 =>
      clk_count <= clk_count +1;
    WHEN s1 =>
      Hsync_buf_cld <= '1' ;
      tft_clk_buf_cld <= '0';
      clk_count <= clk_count +1;
      ENAB_ok<='0';
    WHEN s2 =>
      tft_clk_buf_cld <= '1';
      clk_count <= clk_count +1;
    WHEN s3 =>
      tft_clk_buf_cld <= '0' ;
      clk_count <= clk_count +1;
    WHEN s4 =>
      Hsync_buf_cld <= '0' ;
      tft_clk_buf_cld <= '1';
```

```
  clk_count <= clk_count +1;
WHEN s5 =>
  tft_clk_buf_cld <= '0' ;
  clk_count <= clk_count +1;
WHEN s7 =>
  tft_clk_buf_cld <= '0';
  if ((tft_clk_count /= (THd)) AND (valid_data='1'))then
  data_read_enable_cld <= '1';
  end if ;
  clk_count <= clk_count +1;
  data_send_enable_cld <= '0' ;
WHEN s8 =>
  data_read_enable_cld <= '0' ;
  clk_count <= clk_count +1;
WHEN s9 =>
  tft_clk_buf_cld <= '0' ;
  clk_count <= clk_count +1;
WHEN s10 =>
  tft_clk_buf_cld <= '1';
  if (ENAB_ok='1') then
  ENAB_cld <= '0' ;
  end if;
  clk_count <= clk_count +1;
WHEN s11 =>
  ENAB_ok <= '1';
  tft_clk_buf_cld <= '0';
   clk_count <= clk_count +1;
WHEN s6 =>
  clk_count <= clk_count +1;
  tft_clk_buf_cld <= '0' ;
WHEN s13 =>
  clk_count <= clk_count +1;
WHEN s14 =>
  clk_count <= clk_count +1;
WHEN OTHERS =>
  NULL;
END CASE;
CASE current_state IS
```

```
WHEN s0 =>
  IF (clk_count = Tstable) THEN
    clk_count <= (others => '0') ;
  END IF;
WHEN s1 =>
  IF (clk_count = THc) THEN
    clk_count <= (others => '0') ;
  END IF;
WHEN s2 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0');
  END IF;
WHEN s3 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0') ;
    tft_clk_count <= tft_clk_count +1;
  END IF;
WHEN s4 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0') ;
  END IF;
WHEN s5 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0') ;
    tft_clk_count <= tft_clk_count +1;
  END IF;
WHEN s8 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0') ;
  END IF;
WHEN s9 =>
  IF (clk_count = Tdivision) THEN
    clk_count <= (others => '0') ;
    tft_clk_count <= tft_clk_count +1;
    if (valid_data='1') then
    data_send_enable_cld <= '1' ;
    end if;
    ENAB_cld <= '1' ;
```

```
      END IF;
    WHEN s10 =>
      IF (clk_count = Tdivision) THEN
        clk_count <= (others => '0') ;
      END IF;
    WHEN s11 =>
      IF (clk_count = Tdivision) THEN
        clk_count <= (others => '0') ;
        tft_clk_count <= tft_clk_count +1;
      END IF;
    WHEN s6 =>
      IF (clk_count = Tdivision) THEN
        clk_count <= (others => '0') ;
        tft_clk_count <= (others => '0') ;
      END IF;
    WHEN s13 =>
      IF (clk_count = Tdivision) THEN
        clk_count <= (others => '0') ;
      END IF;
    WHEN s14 =>
      IF (clk_count = Tdivision) THEN
        tft_clk_count <= (others => '0');
        clk_count <= (others => '0') ;
      END IF;
    WHEN OTHERS =>
      NULL;
    END CASE;
  END IF;
END PROCESS clocked;
nextstate : PROCESS (clk_count, current_state, tft_clk_count)
BEGIN
  CASE current_state IS
  WHEN initialisation =>
      next_state <= s0;
  WHEN s0 =>
    IF (clk_count = Tstable) THEN
      next_state <= s1;
    ELSE
```

```
      next_state <= s0;
    END IF;
  WHEN s1 =>
    IF (clk_count = THc) THEN
      next_state <= s2;
    ELSE
      next_state <= s1;
    END IF;
  WHEN s2 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s3;
    ELSIF (tft_clk_count = THp) THEN
      next_state <= s4;
    ELSE
      next_state <= s2;
    END IF;
  WHEN s3 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s2;
    ELSE
      next_state <= s3;
    END IF;
  WHEN s4 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s4;
    ELSIF (tft_clk_count = (THe-2)) THEN
      next_state <= s12;
    ELSE
      next_state <= s4;
    END IF;
  WHEN s5 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s4;
    ELSE
      next_state <= s5;
    END IF;
  WHEN s7 =>
    IF (tft_clk_count /= THd) THEN
```

```
      next_state <= s8;
    ELSIF (tft_clk_count = THd) THEN
      next_state <= s10;
    ELSE
      next_state <= s7;
    END IF;
  WHEN s8 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s9;
    ELSE
      next_state <= s8;
    END IF;
  WHEN s9 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s7;
    ELSE
      next_state <= s9;
    END IF;
  WHEN s10 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s10;
    ELSIF (tft_clk_count = TH) THEN
      next_state <= s14;
    ELSE
      next_state <= s10;
    END IF;
  WHEN s11 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s10;
    ELSE
      next_state <= s11;
    END IF;
  WHEN s6 =>
    IF (clk_count = Tdivision) THEN
      next_state <= s7;
    ELSE
      next_state <= s6;
    END IF;
```

```
      WHEN s13 =>
        IF (clk_count = Tdivision) THEN
          next_state <= s6;
        ELSE
          next_state <= s13;
        END IF;
      WHEN s14 =>
        IF (clk_count = Tdivision) THEN
          next_state <= s1;
        ELSE
          next_state <= s14;
        END IF;
      WHEN OTHERS =>
        next_state <= initialisation;
      END CASE;
    END PROCESS nextstate;
    ENAB <= ENAB_cld;
    Hsync_buf <= Hsync_buf_cld;
    data_read_enable <= data_read_enable_cld;
    data_send_enable <= data_send_enable_cld;
    tft_clk_buf <= tft_clk_buf_cld;
END fsm;
```

Vertical Synchronization Generator Block VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Vsync_Generator IS
  PORT(
    Hsync_buf          : IN     std_logic;
    rst                : IN     std_logic;
    Vsync_buf          : OUT    std_logic;
    address_increment_ok : OUT    std_logic;
    valid_data         : OUT    std_logic
  );
END Vsync_Generator ;
library UNISIM;
use UNISIM.all;
ARCHITECTURE fsm OF Vsync_Generator IS
  -- Architecture Declarations
  SIGNAL Hsync_count    : std_logic_vector(8 downto 0);
  TYPE STATE_TYPE IS (s0, s1, s2, s3);
  -- State vector declaration
  ATTRIBUTE state_vector : string;
  ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state" ;
  -- Declare current and next state signals
  SIGNAL current_state : STATE_TYPE ;
  SIGNAL next_state : STATE_TYPE ;
  -- Declare any pre-registered internal signals
  SIGNAL Vsync_buf_cld : std_logic  ;
  SIGNAL address_increment_ok_cld : std_logic  ;
  SIGNAL valid_data_cld : std_logic  ;
BEGIN
  ----------------------------------------------------------------------------
  clocked : PROCESS(Hsync_buf, rst )
  ----------------------------------------------------------------------------
  BEGIN
    IF (rst = '0') THEN
      current_state <= s0;
      -- Reset Values
      Vsync_buf_cld <= '0';
```

```
    address_increment_ok_cld <= '0';
    valid_data_cld <= '0';
  ELSIF (Hsync_buf'EVENT AND Hsync_buf = '1') THEN
    current_state <= next_state;
    -- Default Assignment To Internals
    address_increment_ok_cld <= '0';
    -- Combined Actions for internal signals only
    CASE current_state IS
    WHEN s0 =>
      Hsync_count <= Hsync_count+1;
      Vsync_buf_cld <= '1' ;
      valid_data_cld <= '0' ;
      address_increment_ok_cld <= '0' ;
    WHEN s1 =>
      Vsync_buf_cld <= '0' ;
      Hsync_count <= Hsync_count +1;
      IF (Hsync_count = TVs) THEN
        Hsync_count <= (others => '0') ;
      END IF;
    WHEN s2 =>
      valid_data_cld <= '1' ;
      Hsync_count <= Hsync_count +1;
    WHEN s3 =>
      valid_data_cld <= '0' ;
      Hsync_count <= Hsync_count +1;
      address_increment_ok_cld <= '1' ;
      IF (Hsync_count = TV) THEN
        Hsync_count <= (others => '0') ;
      END IF;
    WHEN OTHERS =>
      NULL;
    END CASE;
  END IF;
END PROCESS clocked;
--------------------------------------------------------------------------------
nextstate : PROCESS (Hsync_count, current_state )
--------------------------------------------------------------------------------
BEGIN
```

```
    CASE current_state IS
    WHEN s0 =>
      IF (Hsync_count = TVp) THEN
        next_state <= s1;
      ELSE
        next_state <= s0;
      END IF;
    WHEN s1 =>
      IF (Hsync_count = TVs) THEN
        next_state <= s2;
      ELSE
        next_state <= s3;
      END IF;
    WHEN s2 =>
      IF (Hsync_count = TVd) THEN
        next_state <= s3;
      ELSE
        next_state <= s2;
      END IF;
    WHEN s3 =>
      IF (Hsync_count = TV) THEN
        next_state <= s1;
      ELSE
        next_state <= s3;
      END IF;
    WHEN OTHERS =>
      next_state <= s0;
    END CASE;
  END PROCESS nextstate;
  -- Clocked output assignments
  Vsync_buf <= Vsync_buf_cld;
  address_increment_ok <= address_increment_ok_cld;
  valid_data <= valid_data_cld;
END fsm;
```

Memory Interface Block VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Memory_Interface IS
  PORT(
    address_increment_ok : IN     std_logic;
    clk                  : IN     std_logic;
    data_read_enable     : IN     std_logic;
    data_send_enable     : IN     std_logic;
    initial_address      : IN     std_logic_vector (17 DOWNTO 0);
    rst                  : IN     std_logic;
    CE0                  : OUT    std_logic;
    OE                   : OUT    std_logic;
    R_W                  : OUT    std_logic;
    send_data            : OUT    std_logic_vector (15 DOWNTO 0);
    DP_address_buf       : INOUT  std_logic_vector (17 DOWNTO 0) );
END Memory_Interface ;
ARCHITECTURE fsm OF Memory_Interface IS
  signal  data_read : std_logic_vector (15 downto 0);
  signal  wait_count : std_logic_vector (1 downto 0);
  TYPE STATE_TYPE IS ( s0, s1, s2, s3, s4, s5);
  ATTRIBUTE state_vector : string;
  ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state" ;
  SIGNAL current_state : STATE_TYPE ;
  SIGNAL next_state : STATE_TYPE ;
  SIGNAL CE0_cld : std_logic  ;
  SIGNAL DP_address_buf_cld : std_logic_vector (17 DOWNTO 0) ;
  SIGNAL OE_cld : std_logic  ;
  SIGNAL R_W_cld : std_logic  ;
  SIGNAL send_data_cld : std_logic_vector (15 DOWNTO 0) ;
BEGIN
  --------------------------------------------------------------------------
  clocked : PROCESS(clk ,rst )
  --------------------------------------------------------------------------
  BEGIN
    IF (rst = '0') THEN
      current_state <= s0;
```

```
    CE0_cld <= '1';
    DP_address_buf_cld <= "000000000000000000";
    OE_cld <= '1';
    R_W_cld <= '1';
    send_data_cld <= "0000000000000000";
    data_read <= "0000000000000000";
  ELSIF (clk'EVENT AND clk = '1') THEN
    current_state <= next_state;
    R_W_cld <= '1';
    -- Combined Actions for internal signals only
    CASE current_state IS
    WHEN s0 =>
      R_W_cld <= '1' ;
      OE_cld <= '1' ;
      CE0_cld <= '1' ;
      DP_address_buf_cld <= initial_address ;
      wait_count <= (others => '0') ;
    WHEN s2 =>
      CE0_cld <= '0' ;
      OE_cld <= '0' ;
    WHEN s4 =>
      data_read <= DP_data ;
    WHEN s5 =>
      CE0_cld <= '1' ;
      OE_cld <= '1' ;
      wait_count <= wait_count +1;
      IF (wait_count = Rc THEN
        DP_address_buf_cld <= DP_address_buf_cld +1;
        send_data_cld <= data_read ;
        wait_count <= (others => '0') ;
      END IF;
    WHEN OTHERS =>
      NULL;
    END CASE;
  END IF;
END PROCESS clocked;
--------------------------------------------------------------------------------
nextstate : PROCESS (address_increment_ok, current_state, data_read_enable, wait_count )
```

```
  ---------------------------------------------------------------------------
  BEGIN
    CASE current_state IS
    WHEN s0 =>
        next_state <= s1;
    WHEN s1 =>
      IF (data_read_enable = '1') THEN
        next_state <= s2;
      ELSIF (address_increment_ok = '1') THEN
        next_state <= s0;
      ELSE
        next_state <= s1;
      END IF;
    WHEN s2 =>
        next_state <= s4;
    WHEN s4 =>
        next_state <= s5;
    WHEN s5 =>
      IF (wait_count = Rc) THEN
        next_state <= s1;
      ELSE
        next_state <= s5;
      END IF;
    WHEN OTHERS =>
      next_state <= s0;
    END CASE;
  END PROCESS nextstate;
  DP_address_buf <= DP_address_buf_cld;
  OE <= OE_cld;
  R_W <= R_W_cld;
  send_data <= send_data_cld;
END fsm;
```

TFT Controller Structural Block Connector VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY tft_controller IS
  PORT(
    DP_data    : IN     std_logic_vector (15 DOWNTO 0);
    clk        : IN     std_logic;
    frame_no   : IN     std_logic_vector (1 DOWNTO 0);
    rst        : IN     std_logic;
    CE0        : OUT    std_logic;
    DP_address : OUT    std_logic_vector (17 DOWNTO 0);
    ENAB       : OUT    std_logic;
    HVR        : OUT    std_logic;
    Hsync      : OUT    std_logic;
    OE         : OUT    std_logic;
    R_W        : OUT    std_logic;
    TFT_BLUE   : OUT    std_logic_vector (5 DOWNTO 0);
    TFT_GREEN  : OUT    std_logic_vector (5 DOWNTO 0);
    TFT_RED    : OUT    std_logic_vector (5 DOWNTO 0);
    Vsync      : OUT    std_logic;
    tft_clk    : OUT    std_logic
  );
END tft_controller ;
ARCHITECTURE struct OF tft_controller IS
  SIGNAL count : std_logic;
  SIGNAL DP_address_buf      : std_logic_vector(17 DOWNTO 0);
  SIGNAL Hsync_buf           : std_logic;
  SIGNAL Vsync_buf           : std_logic;
  SIGNAL address_increment_ok : std_logic;
  SIGNAL data_read_enable    : std_logic;
  SIGNAL data_send_enable    : std_logic;
  SIGNAL initial_address     : std_logic_vector(17 DOWNTO 0);
  SIGNAL send_data           : std_logic_vector(15 DOWNTO 0);
  SIGNAL tft_clk_buf         : std_logic;
  SIGNAL valid_data          : std_logic;
  COMPONENT Color_Decoder
  PORT (
```

```
  send_data : IN     std_logic_vector (15 DOWNTO 0);
  TFT_BLUE  : OUT    std_logic_vector (5 DOWNTO 0);
  TFT_GREEN : OUT    std_logic_vector (5 DOWNTO 0);
  TFT_RED   : OUT    std_logic_vector (5 DOWNTO 0)
);
END COMPONENT;
COMPONENT Hsync_Generator
PORT (
  clk             : IN     std_logic ;
  rst             : IN     std_logic ;
  valid_data      : IN     std_logic ;
  ENAB            : OUT    std_logic ;
  Hsync_buf       : OUT    std_logic ;
  data_read_enable : OUT    std_logic ;
  data_send_enable : OUT    std_logic ;
  tft_clk_buf     : OUT    std_logic
);
END COMPONENT;
COMPONENT Memory_Interface
PORT (
  DP_data             : IN     std_logic_vector (15 DOWNTO 0);
  Vsync_buf           : IN     std_logic ;
  address_increment_ok : IN     std_logic ;
  data_read_enable    : IN     std_logic ;
  data_send_enable    : IN     std_logic ;
  initial_address     : IN     std_logic_vector (17 DOWNTO 0);
  rst                 : IN     std_logic ;
  CE0                 : OUT    std_logic ;
  OE                  : OUT    std_logic ;
  R_W                 : OUT    std_logic ;
  send_data           : OUT    std_logic_vector (15 DOWNTO 0);
  DP_address_buf      : INOUT  std_logic_vector (17 DOWNTO 0)
);
END COMPONENT;
COMPONENT Vsync_Generator
PORT (
  Hsync_buf           : IN     std_logic ;
  rst                 : IN     std_logic ;
```

```
    Vsync_buf           : OUT    std_logic ;
    address_increment_ok : OUT    std_logic ;
    valid_data          : OUT    std_logic
  );
  END COMPONENT;
  FOR ALL : Color_Decoder USE ENTITY tft_control.Color_Decoder;
  FOR ALL : Hsync_Generator USE ENTITY tft_control.Hsync_Generator;
  FOR ALL : Memory_Interface USE ENTITY tft_control.Memory_Interface;
  FOR ALL : Vsync_Generator USE ENTITY tft_control.Vsync_Generator;
BEGIN
  -- eb1 1
  tft_clk   <= tft_clk_buf;
  Hsync   <= Hsync_buf;
  Vsync   <= Vsync_buf;
  HVR   <='0';
  DP_address<= DP_address_buf;
  process(frame_no)
  begin
   case frame_no is
    when "00" =>
      initial_address <=(others =>'0');
    when "01" =>
     initial_address <= FRAME1_AD;
    when "10" =>
     initial_address <= FRAME2_AD ;
    when others =>
      initial_address <=(others =>'0');
   end case;
  end process;
  I0 : Color_Decoder
   PORT MAP (
     send_data => send_data,
     TFT_BLUE  => TFT_BLUE,
     TFT_GREEN => TFT_GREEN,
     TFT_RED   => TFT_RED);
  I1 : Hsync_Generator
   PORT MAP (
     clk             => clk,
```

```
      rst             => rst,
      valid_data      => valid_data,
      ENAB            => ENAB,
      Hsync_buf       => Hsync_buf,
      data_read_enable => data_read_enable,
      data_send_enable => data_send_enable,
      tft_clk_buf     => tft_clk_buf
    );
  I3 : Memory_Interface
    PORT MAP (
      DP_data             => DP_data,
      Vsync_buf           => Vsync_buf,
      address_increment_ok => address_increment_ok,
      clk                 => clk,
      data_read_enable    => data_read_enable,
      data_send_enable    => data_send_enable,
      initial_address     => initial_address,
      CE0                 => CE0,
      OE                  => OE,
      R_W                 => R_W,
      send_data           => send_data,
      DP_address_buf      => DP_address_buf
    );
  I2 : Vsync_Generator
    PORT MAP (
      Hsync_buf           => Hsync_buf,
      rst                 => rst,
      Vsync_buf           => Vsync_buf,
      address_increment_ok => address_increment_ok,
      valid_data          => valid_data
    );
END struct;
```

Color Decoder VHDL Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Color_Decoder IS
  PORT(
    send_data : IN     std_logic_vector (15 DOWNTO 0);
    TFT_BLUE  : OUT    std_logic_vector (5 DOWNTO 0);
    TFT_GREEN : OUT    std_logic_vector (5 DOWNTO 0);
    TFT_RED   : OUT    std_logic_vector (5 DOWNTO 0)   );
END Color_Decoder ;
ARCHITECTURE tbl OF Color_Decoder IS
 BEGIN
  truth_process: PROCESS(send_data)
  BEGIN
    CASE send_data IS
    WHEN BLACK =>
      TFT_RED <= "000000";
      TFT_GREEN <= "000000";
      TFT_BLUE <= "000000";
    WHEN BLUE =>
      TFT_RED <= "000000";
      TFT_GREEN <= "000000";
      TFT_BLUE <= "111111";
    WHEN GREEN =>
      TFT_RED <= "000000";
      TFT_GREEN <= "111111";
      TFT_BLUE <= "000000";
    WHEN CYAN =>
      TFT_RED <= "000000";
      TFT_GREEN <= "111111";
      TFT_BLUE <= "111111";
    WHEN RED =>
      TFT_RED <= "111111";
      TFT_GREEN <= "000000";
      TFT_BLUE <= "000000";
    WHEN MAGENTA =>
      TFT_RED <= "111111";
```

```
      TFT_GREEN <= "000000";
      TFT_BLUE <= "111111";
    WHEN YELLOW =>
      TFT_RED <= "111111";
      TFT_GREEN <= "111111";
      TFT_BLUE <= "000000";
    WHEN WHITE =>
      TFT_RED <= "111111";
      TFT_GREEN <= "111111";
      TFT_BLUE <= "111111";
    WHEN OTHERS =>
      NULL;
    END CASE;
  END PROCESS truth_process;
END tbl;
```

APPENDIX C : BOOT CODE FOR COLDFIRE MICROPROCESSOR

```
asm_startmeup:
_asm_startmeup:
        move.w          #0x2000,SR
        move.l    #VECTOR_TABLE,d0
        movec      d0,VBR
        /* Invalidate the cache and disable it */
        move.l  #0x01000000,d0
        dc.l      0x4e7b0002              /* movec d0,cacr */
        /* Disable ACRs */
        moveq.l #0,d0
        dc.l      0x4e7b0004              /* movec d0,ACR0 */
        dc.l      0x4e7b0005              /* movec d0,ACR1 */
        /* Initialize RAMBAR0 */
        move.l  #0xE0000001,d0
        dc.l      0x4e7b0C04
        nop
        // Point SP into SRAM
        move.l  #0xE0000000+SRAMsize,SP
        /* Obtain pointer to where MBAR is to be mapped */
        jsr             mcf5407_mbar
        move.l  d0,d6
        /* Obtain pointer to where RAMBAR0 is to be mapped */
        jsr             mcf5407_rambar0
        /* Adjust SP to SRAM */
        move.l  d0,a0
        lea             SRAMsize(a0),SP
        /* Map RAMBAR0 and MBAR */
        addq.l  #1,d0
        dc.l      0x4e7b0C04
        move.l  d6,d0
        addq.l  #1,d0
        /* Obtain pointer to where RAMBAR1 is to be mapped */
        jsr             mcf5407_rambar1
        addq.l  #1,d0
```

```
        dc.l    0x4e7b0C05              /* movec d0,RAMBAR1 */
        /* Initialize mcf5407 periphs, etc */
        move.l  d6,-(sp)        /* pointer to internal resources */
        jsr             mcf5407_init
        lea             4(sp),sp
        move.l  #___SP_INIT,sp
        nop
        nop
        jmp             __start
_cpu_cache_flush:
cpu_cache_flush:
        nop                     /* synchronize - flush store buffer */
        moveq.l #0,d0           /* init line counter */
        moveq.l #0,d1           /* init set counter */
        move.l  d0,a0           /* init An */
_cpu_cache_disable:
cpu_cache_disable:
        nop
        move.w  #0x2700,SR              /* mask off IRQ's */
        jsr             _cpu_cache_flush        /* flush the cache completely */
        clr.l   d0
        movec   d0,ACR0                         /* ACR0 off */
        movec   d0,ACR1                         /* ACR1 off */
        movec   d0,ACR2                         /* ACR2 off */
        movec   d0,ACR3                         /* ACR3 off */
        move.l  #0x01000000,d0 /* Invalidate and disable cache */
        movec   d0,CACR
        rts
mcf5407_wr_vbr:
_mcf5407_wr_vbr:
        move.l  4(sp),d0
        andi.l  #0xFFF00000,d0 /* align to 1M boundary    */
        movec   d0,VBR
        nop
        rts

mcf5407_wr_cacr:
_mcf5407_wr_cacr:
```

```
    move.l  4(sp),d0
    dc.l    0x4e7b0002      /* movec d0,cacr    */
    nop
    rts
mcf5407_wr_acr0:
_mcf5407_wr_acr0:
    move.l  4(sp),d0
    dc.l    0x4e7b0004      /* movec d0,ACR0*/
    nop
    rts
mcf5407_wr_acr1:
_mcf5407_wr_acr1:
    move.l  4(sp),d0
    dc.l    0x4e7b0005      /* movec d0,ACR1*/
    rts

mcf5407_wr_acr2:
_mcf5407_wr_acr2:
    move.l  4(sp),d0
    dc.l    0x4e7b0006      /* movec d0,ACR2*/
    rts

mcf5407_wr_acr3:
_mcf5407_wr_acr3:
    move.l  4(sp),d0
    dc.l    0x4e7b0007      /* movec d0,ACR3*/
    rts

mcf5407_wr_rambar0:
_mcf5407_wr_rambar0:
    move.l  4(sp),d0
    dc.l    0x4e7b0C04      /* movec d0,RAMBAR0    */
    rts

mcf5407_wr_rambar1:
_mcf5407_wr_rambar1:
    move.l  4(sp),d0
    dc.l    0x4e7b0C05      /* movec d0,RAMBAR1    */
```

```
   rts

mcf5407_wr_mbar:
_mcf5407_wr_mbar:
   move.l  4(sp),d0
   dc.l    0x4e7b0C0F      /* movec d0,MBAR          */
   rts

 /*********************************************************************
* NAME                          :       SYSINIT.C
* DATE                          :       26 June 2004
* STATUS                        :       Progress
* DESCRIPTION                   :       Power-on Reset configuration of the MCF5407.
 *--------------------------------------------------------------------
 * REVISION HISTORY :          REV     DATE            AUTHOR          DESCRIPTION
 *                             N/A     26.06.04        G.GOKSUGUR      Original Code.
* COMPILER:       CodeWarrior IDE version 4.2.6.832
 *********************************************************************/
void
mcf5407_init (MCF5407_IMM *imm)
{
        /*******************************************************
        *       Out of reset, the low-level assembly code calls this
        *       routine to initialize the MCF5407
        *******************************************************/
        extern char __SDRAM_INIT[];
        extern char __DATA_ROM[];
        extern char __DATA_RAM[];
        extern char __DATA_END[];
        extern char __BSS_START[];
        extern char __BSS_END[];
        extern uint32 VECTOR_TABLE[];
        extern uint32 __VECTOR_RAM[];
        extern int __end_of_text, _text, __ltext, __letext;
        uint8 *dp;
        uint8 *sp;
        uint32 n;
        mcf5407_sim_init(imm);
```

```
        mcf5407_timer_init(imm);
        mcf5407_pp_init(imm);
        mcf5407_uart_init(imm);
        mcf5407_mbus_init(imm);
        mcf5407_dma_init(imm);
        mcf5407_cs_init(imm);
        MCF5407_WR_SIM_IMR(imm, MCF5407_RD_SIM_IMR(imm) &
~MCF5407_SIM_IMR_EINT7);
        MCF5407_WR_SIM_IMR(imm, MCF5407_RD_SIM_IMR(imm) &
~MCF5407_SIM_IMR_UART0);
#ifdef NOTUSINGRUNTIME
        // Copy the vector table to RAM
        if (__VECTOR_RAM != VECTOR_TABLE)
        {
                for (n = 0; n < 256; n++)
                        __VECTOR_RAM[n] = VECTOR_TABLE[n];
        }
   mcf5407_wr_vbr((uint32)__VECTOR_RAM);
   int *src = &__ltext;  // pointer src gets the value(address) of _ltext
   int *dst = &_text;    // pointer dst gest the value(address) of _text

   while (dst < &__end_of_text) {  // copy the text from ROM to RAM
      *dst++ = *src++;
   }

        // Move initialized data from ROM to RAM.
        if (__DATA_ROM != __DATA_RAM)
        {
                dp = (uint8 *)__DATA_RAM;
                sp = (uint8 *)__DATA_ROM;
                n = __DATA_END - __DATA_RAM;
                while (n--)
                        *dp++ = *sp++;
        }
        // Zero uninitialized data
        if (__BSS_START != __BSS_END)
        {
                sp = (uint8 *)__BSS_START;
```

```
                n = __BSS_END - __BSS_START;
                while (n--)
                        *sp++ = 0;
        }
#endif // NOTUSINGRUNTIME
   mcf5407_wr_acr0( 0
      | MCF5407_ACR_BASE(SRAM_ADDRESS)
      | MCF5407_ACR_MASK(0x00000000)
      | MCF5407_ACR_E
                | MCF5407_ACR_CM_00
      | MCF5407_ACR_S_IGNORE
                );
   mcf5407_wr_acr1( 0
      | MCF5407_ACR_BASE(FLASH_ADDRESS)
      | MCF5407_ACR_MASK(0x00000000)
      | MCF5407_ACR_E
                | MCF5407_ACR_CM_00
      | MCF5407_ACR_S_IGNORE
                );
}
void
mcf5407_sim_init (MCF5407_IMM *imm)
{
        MCF5407_WR_SIM_SYPCR(imm,0);            /*  Disable the Software Watchdog */
        MCF5407_WR_SIM_PAR(imm,0);              /*  Set all Par Ports to General I/O */
        MCF5407_WR_SIM_IRQPAR(imm,0);   /*  IRQ1,3,5 are Internal Interrupts 1,3,5 */
        MCF5407_WR_SIM_PLLCR(imm,0);            /*  Disable CPU STOP Instruction */
        MCF5407_WR_SIM_MPARK(imm,0);    /*  Disable external visibility of internal bus */
        MCF5407_WR_SIM_IMR(imm,0xFFFFFFFF);     /*  Mask all interrupt sources */
        MCF5407_WR_SIM_AVCR(imm,0xFF);          /*  Autovector all external interrupts */
        //Setup Interrupt Source Vectors
        MCF5407_WR_SIM_ICR0(imm, 0          /* Software Watchdog */
                        | MCF5407_SIM_ICR_AVEC
                        | MCF5407_SIM_ICR_IL(7)
                        | MCF5407_SIM_ICR_IP_EXT ) ;
                        | MCF5407_SIM_ICR_AVEC      /* UART 1 */
                        | MCF5407_SIM_ICR_IL(3)
                        | MCF5407_SIM_ICR_IP_EXT ) ;
```

```
        MCF5407_WR_SIM_ICR5(imm, 0
                        | MCF5407_SIM_ICR_AVEC        /* UART 2 */
                        | MCF5407_SIM_ICR_IL(3)
                        | MCF5407_SIM_ICR_IP_INT ) ;
/* Enable interrupts to ColdFire core */
        MCF5407_WR_SIM_IMR(imm,              MCF5407_RD_SIM_IMR(imm)              &
~MCF5407_SIM_IMR_UART0);
}
/********************************************************************/
void
mcf5407_pp_init (MCF5407_IMM *imm)
{
        MCF5407_WR_PP_PADDR(imm,0);                /* Init PADDR to all inputs. */
}
/********************************************************************/
void
mcf5407_mbus_init (MCF5407_IMM *imm)
{
        MCF5407_WR_MBUS_MBCR(imm,0);    /* Disable MBUS */
}
/********************************************************************/
void
mcf5407_dma_init (MCF5407_IMM *imm)
{
        MCF5407_WR_DMA0_DCR(imm,0);                /* Disable DMA 0 */
        MCF5407_WR_DMA1_DCR(imm,0);                /* Disable DMA 1 */
        MCF5407_WR_DMA2_DCR(imm,0);                /* Disable DMA 2 */
        MCF5407_WR_DMA3_DCR(imm,0);                /* Disable DMA 3 */
}
/********************************************************************/
void
mcf5407_uart_init(MCF5407_IMM *imm)
{
        /*****************************************************************/
        /*  UART 0                                       */
        /*****************************************************************/
        MCF5407_WR_UART0_UCR(imm,MCF5407_UART_UCR_RESET_TX);
        MCF5407_WR_UART0_UCR(imm,MCF5407_UART_UCR_RESET_RX);
```

```
        MCF5407_WR_UART0_UCR(imm,MCF5407_UART_UCR_TX_ENABLED);
        MCF5407_WR_UART0_UCR(imm,MCF5407_UART_UCR_RX_ENABLED);
        MCF5407_WR_UART0_UCR(imm,MCF5407_UART_UCR_RESET_MR);
        MCF5407_WR_UART0_UMR(imm, 0
                | MCF5407_UART_UMR1_PM_NONE                     /* No parity*/
                | MCF5407_UART_UMR1_BC_8                /* 8 bits per character*/
                ) ;
        MCF5407_WR_UART0_UMR(imm, 0
                | MCF5407_UART_UMR2_CM_NORMAL       /* No echo or loopback*/
                | MCF5407_UART_UMR2_STOP_BITS_1     /* 1 stop bit*/
                ) ;
        MCF5407_WR_UART0_UCSR(imm, 0
                | MCF5407_UART_UCSR_RCS0
                | MCF5407_UART_UCSR_RCS2
                | MCF5407_UART_UCSR_RCS3
                | MCF5407_UART_UCSR_TCS0
                | MCF5407_UART_UCSR_TCS3
                ) ;
        MCF5407_WR_UART0_UIMR(imm,0);                   /* Disable all interrupts    */
        MCF5407_WR_UART0_UBG1(imm,0);
        MCF5407_WR_UART0_UBG2(imm,0x51);/* Set baud to 19200, 50MHZ              */
  /*********************************************************************/
  /*  UART 2                                          */
  /*********************************************************************/
   MCF5407_WR_UART1_UCR(imm, 0         /* Disable UART 1 */
       | MCF5407_UART_UCR_TX_DISABLED
       | MCF5407_UART_UCR_RX_DISABLED
       ) ;
}
/*****************************************************************/
void
mcf5407_cs_init (MCF5407_IMM *imm)
{
        /*      ChipSelect 1 - SRAM */
        MCF5407_WR_CS_CSAR1(imm,MCF5407_CS_CSAR(SRAM_ADDRESS));//SRAM_AD
DRESS
        MCF5407_WR_CS_CSCR1(imm,0x0180);
        MCF5407_WR_CS_CSMR1(imm,(MCF5407_CS_CSMR_MASK_2M | 0x01));
```

```
	/*	ChipSelect 2 - DPSRAM */
	MCF5407_WR_CS_CSAR2(imm,MCF5407_CS_CSAR(DPSRAM_ADDRESS));
	MCF5407_WR_CS_CSCR2(imm,0x0180);
	MCF5407_WR_CS_CSMR2(imm,(MCF5407_CS_CSMR_MASK_512K | 0x01));
	/*	ChipSelect 3 - FPGA */
	MCF5407_WR_CS_CSAR3(imm,MCF5407_CS_CSAR(FPGA_ADDRESS));
	MCF5407_WR_CS_CSCR3(imm,0x0180);
	MCF5407_WR_CS_CSMR3(imm,(MCF5407_CS_CSMR_MASK_64K | 0x01));

	/*	ChipSelect 4,5,6 and 7 - Invalid */
	MCF5407_WR_CS_CSMR4(imm,0);
	MCF5407_WR_CS_CSMR5(imm,0);
	MCF5407_WR_CS_CSMR6(imm,0);
	MCF5407_WR_CS_CSMR7(imm,0);
	MCF5407_WR_CS_CSAR0(imm, MCF5407_CS_CSAR(FLASH_ADDRESS));
	MCF5407_WR_CS_CSCR0(imm,0x0D80);
	MCF5407_WR_CS_CSMR0(imm,0x001F0001);
}
/********************************************************************/
void *
mcf5407_mbar (void)
{
	return (void *)IMM_ADDRESS;
}
/********************************************************************/
void *
mcf5407_rambar0 (void)
{
	return (void *)INT_SRAM0_ADDRESS;
}
/********************************************************************/
void *
mcf5407_rambar1 (void)
{
	return (void *)INT_SRAM1_ADDRESS;
}
```

APPENDIX D : PCB LAYOUT OF CONTROLLER CIRCUIT

The twelve layers Printed Circuit Board is designed using Mentor Graphics' layout tool. Six layers are used for signal routing; two layers for power layers and four layers are used for ground. The following figures show some of the layers of the Controller Printed Circuit Board including component layers, power layers and ground layers.

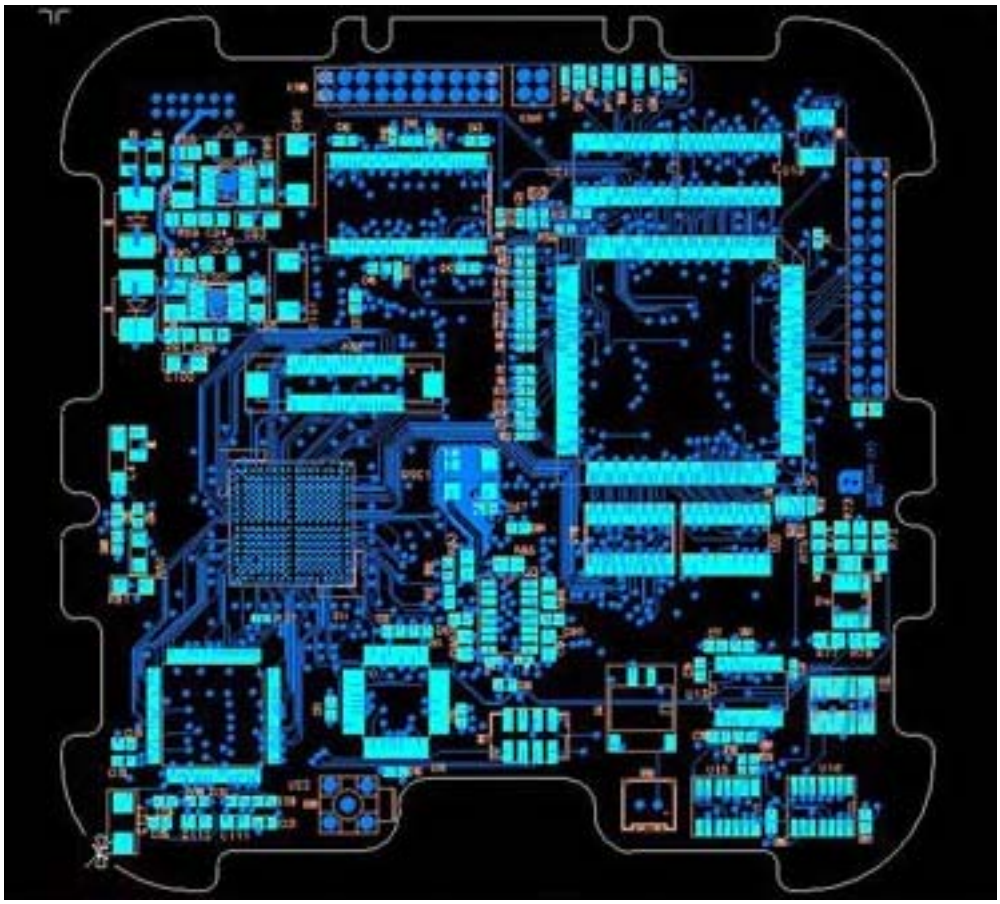

Figure D.1 Component Layer 1

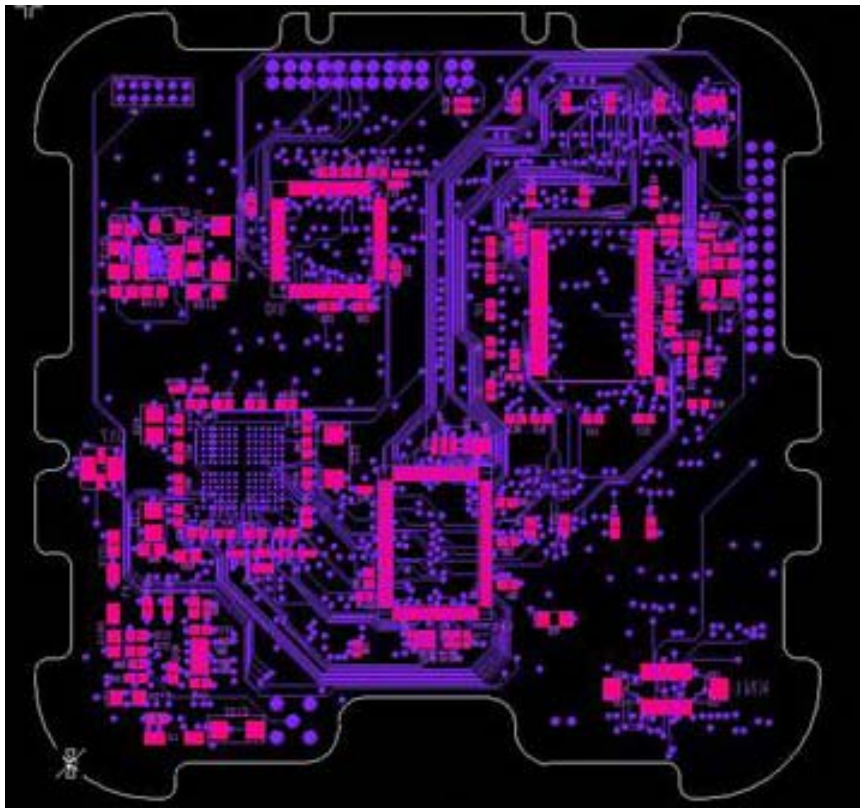

Figure D.2 Component Layer 2

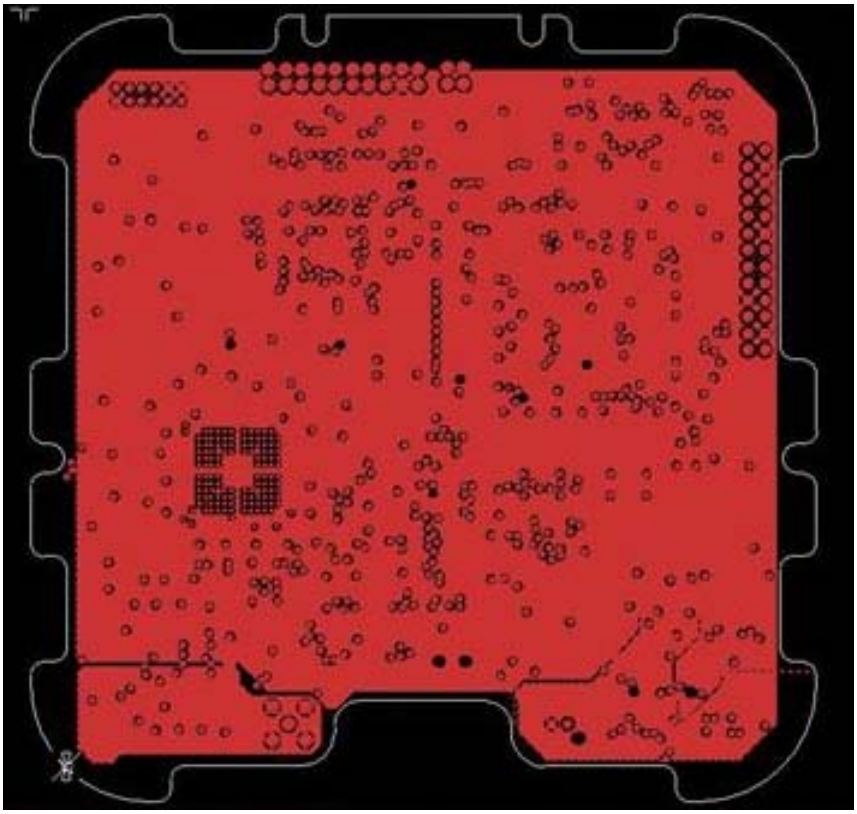

Figure D.3 Ground Layer

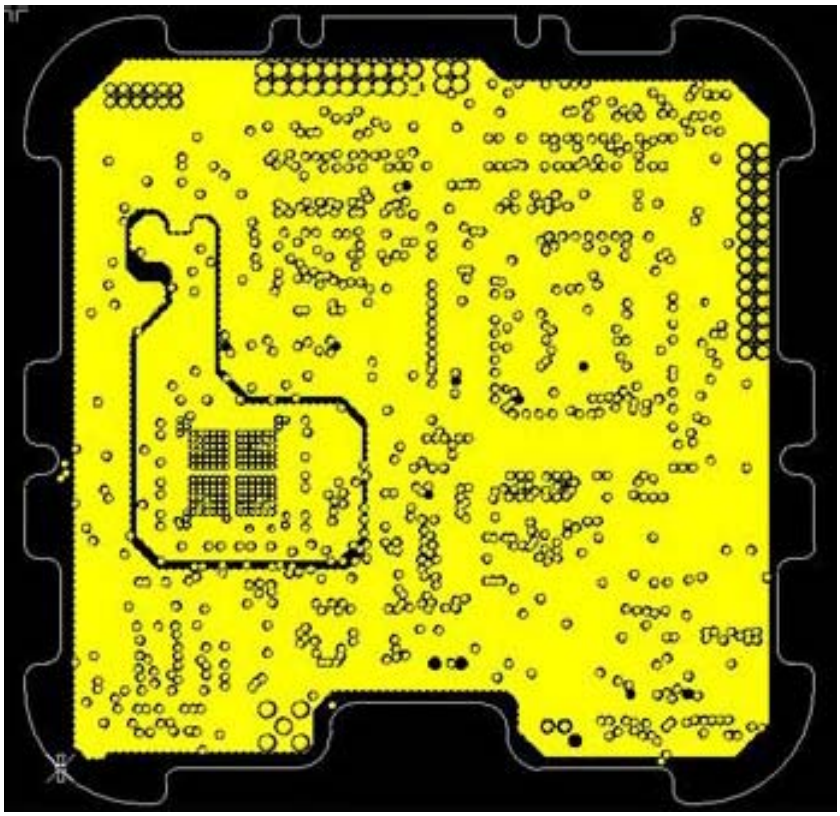

Figure D.4 Power Layer

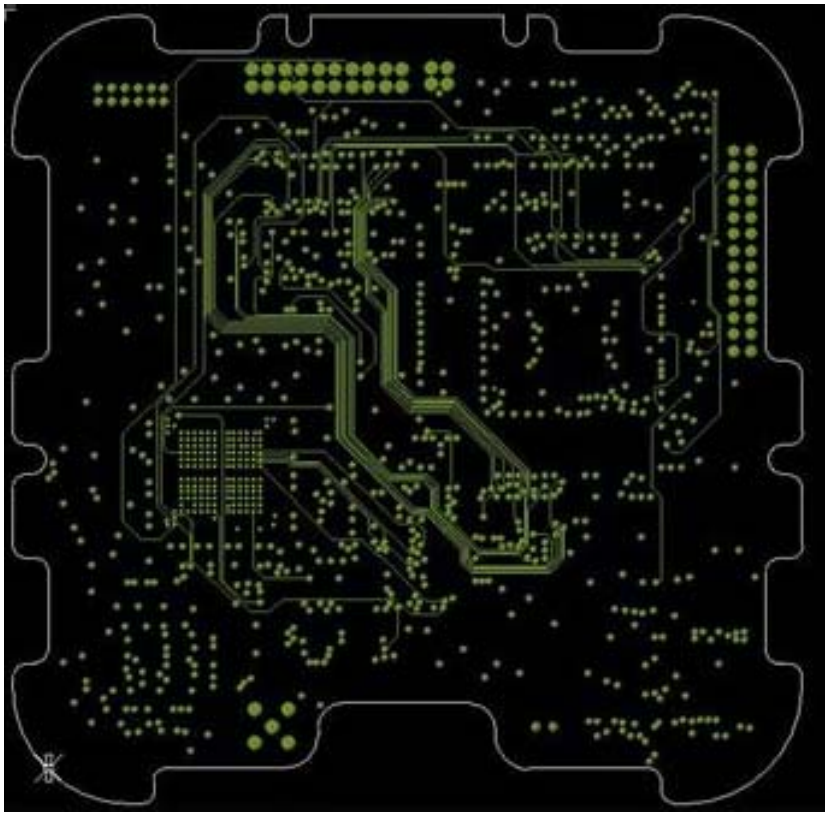

Figure D.5 Signal Layer