Faculty of Engineering
Computer and Systems
Engineering Department
July 2012

Implementation of Hand Gestures Recognition System using Hardware/Software Co-Design

Graduation Project

Supervisor
Dr Ayman Wahba
Computer and Systems Engineering Deptartment
July 2012

It would be a great pleasure to express our sincere appreciation for those who were helpful to us most when we were facing troubles and although they were busy they donated their time and effort through the year to help us, without them this project may have not seen the light.

Dr. Mohamed Bakr

Mrs. Mahinaz Embaby

Eng. Ahmed Sokar

Mr. Peter Stuge .

Thank you for your efforts,

جعله الله فى ميزان حسناتكم

Team Members

- **Ahmed Abdel Fattah Hassan**
- **Ahmed Hafez Khalil**
- **Asmaa Omar**
- **Amani Mohamed Sedek**
- **Mohamed Ismail Khalil**
- **Mohamed Kamal Ali**
- **Mohamed Maged Abdel Majed**

Contents

Abstract 6

1. INTRODUCTION 7
    - 1.1 Hand gestures recognition systems 7
    - 1.2 Problem Statement 8
    - 1.3 Goals 9
    - 1.4 Tasks 10
        - 1.4.1 Software 10
        - 1.4.2 Hardware 10

2. HARDWARE AND SOFTWARE TOOLS 12
    - 2.1 Hardware tools 12
        - 2.1.1 FPGA 12
        - 2.1.2 Altera DE@-115 Board 13
            - 2.2.2.1 Layout and components 13
            - 2.1.2.2 Block Diagram of Altera DE2-115 board 15
            - 2.1.2.3 DE2-115 Control Panel 17
        - 2.1.3 The TRDB_D5M camera 22
            - 2.1.3.1 Block Diagram of the reference design 22
            - 2.1.3.2 Main features of D5M 23
    - 2.2 Hardware tools 28
        - 2.2.1 Quartus II 28
            - 2.2.1.1 RTL viewer 29
            - 2.2.1.2 Timing analysis and design optimization 29
            - 2.2.1.3 SOPC Builder 31
            - 2.2.1.4 Model-SIM 33
            - 2.2.1.5 Nios II Embedded Design Suite 34
            - 2.2.1.6 BSP and HAL 35
    - 2.3 Software Tools 37
        - 2.3.1 Matlab 37
        - 2.3.1 OpenCV 38

3. SYSTEM ANALYSIS AND DESIGN 40
    - 3.1 System Constrains 40
        - 3.1.1 Time Constrains 40
        - 3.1.2 Memory (Data) Constrains 40
        - 3.1.3 User Constrains 40
        - 3.1.4 Environment Constrains 41
    - 3.2 System Design 41
        - 3.2.1 The TRDB_D5M Camera 42
        - 3.2.2 Camera IP Core 42
        - 3.2.3 Camera SRAM Controller 42
        - 3.2.4 NIOS SRAM Controller 43
        - 3.2.5 SRAM Memory Chip 43
        - 3.2.6 NIOS II A (Image Processing Core) 43
        - 3.2.7 NIOS II B (USB) 43
        - 3.2.8 ISP-1362 chip 43
        - 3.2.9 USB Physical 43

4. HAND GESTURE RECOGNITION SYSTEM 44
4.1 System Description 44
4.1.1 Software Phases 45
4.1.1.1 Offline Phases 46
4.1.1.2 Online Phase 60
4.1.1.3 Gesture Detection Phase 60
4.1.1.3.1 Gesture Recognition Steps 61
4.1.1.3.2 Gesture Recognizer..... 62
4.2 Porting On NIOS II 68
4.2.1 Embedded Systems Programming 68
4.2.2 Implementation 69
4.2.2.1 Content 70
4.2.2.2 Steps For Porting 70
4.2.3 Probable Errors 71
4.2.4 Testing on NIOS II processor 72

5. CHAPTER 5 PROFILING 75
5.1 Instrumenting Profilers 75
5.2 Steps 76
5.3 Custom Instructions Introduction 78
5.4 NIOS II Custom Instructions 78
5.5 Implementation of custom instruction 79
5.6 Custom Instruction Type 80
5.6.1 Combinational Custom Instruction 81
5.6.2 MultiCycle custom Instruction 82
5.7 CvInrangeS Optimization 83
5.7.1 RTL View..... 85
5.8 CvCvtColor Optimization 85
5.8.1 RTL View 86

6. HARDWARE IMPLEMENTATION 87
6.1 Camera Core 87
6.1.1 Introduction 87
6.1.2 Image Capturing 87
6.1.3 Preliminary Work 88
6.1.3.1 Problem in hand 88
6.1.3.2 Proposed Solution 88
6.1.4 Final Design 92
6.1.4.1 System Overview 92
6.1.4.2 System's Block Diagram 92
6.1.4.3 Image Capturing 94
6.1.4.4 SRAM Control 98
6.1.4.5 Timing Specifications 102
6.1.4.6 Debugging the System 105
6.2 Multi-NIOS System 106
6.2.1 Multiprocessor Systems 106
6.2.2. Benefits of multiprocessor Systems 106
6.2.3. NIOS II Multiprocessor System..... 106
6.2.4. Hardware Design Consideration 107
6.2.4.1. Autonomous Multiprocessors 107
6.2.4.2. Multiprocessor that Share Resources 107
6.2.5. Our Design 108
6.2.6. Methods 108

6.2.7. Why did we use Mailbox 110
6.2.8. Why didn't we use FIFO 110
6.2.9 Why didn't we use PIO 110
6.2.10. SOPC System 111
6.2.11. Configuration of Mailbox 111
6.2.12. Software Programming 112

7. USB 114
7.1. Introduction to USB 114
7.2. Why choosing USB? 115
7.3 USB Protocol Stack 116
7.4 USB Data Flow 116
7.5. USB Framework 119
7.5.1. Enumeration Process 119
7.5.2. USB Descriptors 121
7.5.2.1. Introduction 121
7.5.2.2. Device Descriptor 121
7.5.2.3. Configuration Descriptor 122
7.5.2.4. Interface Descriptor 123
7.5.2.5. Endpoint Descriptor 124
7.6. USB Human Interface Device (HID) 124
7.6.1. Introduction 124
7.6.2. HID Descriptor 126
7.6.3. Report Descriptor 126
7.6.4. Report Descriptor Parser 128
7.7. USB On DE2-115 128
7.7.1. Relevant Pins Description 129
7.7.2. ISP 1362 130
7.7.2.1. PIO Access Mode 131
7.7.2.2. Memory Organization for the DC 132
7.7.2.3. USB Device Controller (DC) 133
7.7.2.4. DC Data Transfer Operation 133
7.7.2.5. IN Data Transfer 133
7.8. System Architecture 134
7.9. Firmware 134
7.9.1. Firmware Structure Of the Device Controller 134
7.9.2. Important function in HAL for the Device Controller 136
7.10. USBmon tool 138
7.10.1.Introduction 138
7.10.2. USBmon steps 138
7.10.3. Raw Text Data Format 140
7.10.4. Contents of .mon File 140

8. Conclusions and future work 142
8.1. Summary and Conclusions 142
8.2 Future Work 145
9. References 146

Abstract

Even after more than two decades of input devices development, many people still find the interaction with computers an uncomfortable experience.

Efforts should be made to adapt computers to our natural means of communication: speech and body language.

Hand gestures represent a powerful way of communication between people. Object tracking is an area of research that has attracted a lot of attention lately, for its potential regarding the interaction between man and machine. Hand gesture detection and recognition, in real time, from video stream, plays a significant role in the human-computer interaction and, on the current digital image processing applications, this represent a difficult task.

In general we can't define a natural communication between man and machine without using gestures. These can be associated, to some extent, with temporal and spatial structures; we can indicate directions or actions using gestures.

The aim of our project is the proposal of a real time vision system for its application within visual interaction environments through hand gesture recognition using an FPGA and a camera.

The basis of our approach is a fast segmentation process" using hardware software co-design" to obtain the moving marker from the whole image and a recognition process that identifies the hand posture from the temporal sequence of segmented marker.

Chapter 1

Introduction

This chapter presents the overall context of this project. Firstly, a short background description of Hand gestures recognition systems. Then, the problem statement of the project is formulated as a question. The methodology used for answering the question is subsequently introduced: goals and tasks are defined step-by-step. Finally, the organization of Part I is detailed.

1.1 Hand gestures recognition systems

Body language is an important way of communication among humans, adding emphasis to voice messages or even being a complete message by itself. Then Hand gestures represent a powerful way of communication between people, which are rooted deep in our subconscious, an example in this direction can be considered some people tend to gesticulate even when they are talking on the phone. The social anthropologists Edward T. Hall claims 60% of all our communications are nonverbal.
Gestures are used for everything from pointing at a person or an object to change the focus of attention, to conveying information. From the biological and sociological perspective, gestures are loosely defined, thus, researchers are free to visualize and classify gestures as these fit. Biologists define "gesture" broadly, stating, "the notion of gesture is to embrace all kinds of instances where an individual engages in movements whose communicative intentis paramount, manifest and openly acknowledged"

Thus, automatic posture recognition systems could be used for improving Human-Machine Interaction.

A gesture recognition system could be used in any of the following areas:

- ***Human - Machine Interface:*** using hand gestures to control the computer mouse and/or keyboard functions. An example of this, which has been implemented in this project, controls various keyboard and mouse functions using gestures alone.
- ***3D animation*****:** Rapid and simple conversion of hand movements into 3D computer space for the purposes of computer animation.

- ***Visualization:*** Just as objects can be visually examined by rotating them with the hand, so it would be advantageous if virtual 3D objects (displayed on the computer screen) could be manipulated by rotating the hand in space.
- ***Computer games:*** Using the hand to interact with computer games would be more natural for many applications.

- ***Control of mechanical systems (such as robotics):*** *Using* the hand to remotely control a manipulator.

The main obstacle in achieving a natural interaction between humans and machines based on gestures is the lack of appropriate methods of recognition and interpretation of the gestures by the computer.

This project will design and build a Human-Machine Interface using an Altera FPGA & a camera to control the computer mouse to facilitate computer usage for users.
Despite the increase in the attention of such systems there are still certain limitations in literature. Most applications require different constraints like having distinct lightning conditions, usage of a specific camera, making the user wear a multi-colored glove or need lots of training data. The system mentioned in this project disables all these restrictions and provides an adaptive, effort free environment to the user. System starts with an analysis of different color space performances over marker color extraction. This analysis is independent of the working system and just performed to attain valuable information about the color spaces. Working system is based on two steps, namely marker detection and hand gesture recognition. In the marker detection process, normalized HSV color space color locus is used to threshold the marker pixels in the image. Then an adaptive marker locus, whose varying boundaries are estimated from coarse marker region pixels, segments the distinct skin color in the image for the current conditions. Gesture of the hand is recognized by improved geometrical analysis, which is applied on the markers positions.

1.2 Problem statement

The main aim of this project is to answer this question:

Is it possible to implement an intelligent hand gesture recognition system to control computer mouse on a Field Programmable Gate Array (FPGA) platform to optimize the system?

Project main goal is to implement an image processing code that identify the hand gestures on a FPGA kit and interfacing with a PC using USB interface.

The requirements can be described as follow:

- **First of all,** the architecture has to be based on an FPGA.
- **Secondly,** the Hardware/Software Co-Design methodology should be used to implement the algorithms.

> Hardware/Software Co-Design tries to increase the predictability of embedded system design by providing analysis methods that tell designers if a system meets its performance, power, size goals and synthesis methods that let researchers and designers rapidly evaluate many potential design methodologies hardware software co-design is The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design.

Key concepts of hardware software co-design are:

- **Concurrent**: hardware and software developed at the same time on parallel paths
- **Integrated:** interaction between hardware and software development to produce design meeting performance criteria and functional specs.

The two key concepts involved in co-design are concurrent development of HW and SW, and integrated design. Integrated design allows interaction between the design of HW and SW. Co-design techniques using these two key concepts take advantage of design flexibility to create systems that can meet stringent performance requirements with a shorter design cycle.

The importance of co-design in designing hardware/software systems:

- Improves design quality, design cycle time, and cost
- Reduces integration and test time.
- Supports growing complexity of embedded systems.
- Takes advantage of advances in tools and technologies Processor cores.

1.3 Goals

The main goal is to get the full implementation of the project on an FPGA to answer the question. This means building a system that capture a sequence of frames and processing it to get the markers out of the image & get its position to recognize gesture made by markers. To achieve this, the project is split in several parts described below.

Obtaining output from camera

The camera IP core needs to be tailored for our needs. We need to know where it stores data and in which format and to modify its storage area and quality.

Software part

The goal of this step is to design full software system that can capture a stream of frames and process them to get the markers out of the frame and get its position and then recognizing the gesture that has been made Using image processing algorithms.

Porting open-CV library on a NIOS processor

In our software code we use Open-CV library for image processing parts on the frames that's why we will need to put the vital open-CV functions on our Nios Processor to run our code on it. the goal of that step is to figure out which functions we will need to use in our project & put it on our NIOS project.

USB Interface

We chose USB as an interface for communication between our device and the PC. USB is a flexible standard which is widely adopted around the world.

Optimization part

The biggest problem that could face a real time hand gestures recognition System is the speed of that system as we need it as fast as possible to be a real time & once a gesture has been made the control action based on it is done immediately on the computer. Hardware / Software Co-Design gave as a great opportunity to optimize our software code. So the goal of this step is to figure out which part of the software code that takes the most time to get an output and transfer it into a hardware module to get an optimized output and a good results.

1.4 Tasks

1.4.1 Software

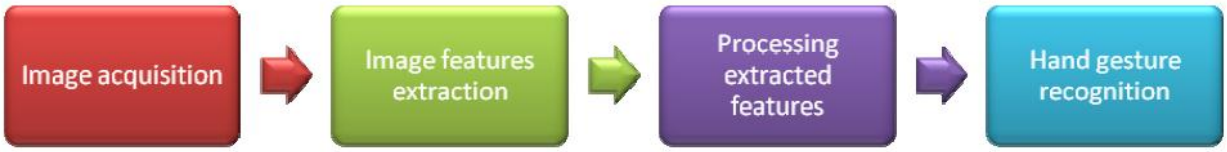


Figure 1.1 abstract block diagram for software system

- Building a complete prototype for the software code using MatLab
- Implementing the prototype algorithm using open CV library in C++

1.4.2 Hardware:

- Capture image with the camera sensor:

This part is essential as it is the first step of the global process. It simply consists of acquiring the image with the camera sensor and made it available to the rest of the system.

The aim of this task is to understand how the camera sensor acquires and sends data through its inputs and outputs.

So we needed to know how to implement the data acquisition from the sensor & verify that the implementation is working according to the requirements to detect potential problems and correct them.

- Saving frames in SRAM memory
- Making NIOS II access the memory and read frames to process the algorithm on them.
- Sending final output to a PC using USB interface.

Chapter 2

Hardware and Software tools

This chapter gives a quick overview of the hardware and software tools used in the project

2.1 Hardware tools

2.1.1 FPGA

Definition

The first constraint is the choice of a technology. In this project, one of the goals is to try to evaluate the feasibility of implementing a motion detection algorithm onto a FPGA platform. FPGA stands for Field-Programmable Gate Array and refers to a semiconductor device containing programmable logic components called "logic blocks" including memories and programmable interconnects. The designer can program the logic blocks and the interconnects like a one-chip programmable breadboard. In the design flow of a new product, the early designs are using FPGAs then migrated into a fixed version that more resembles an ASIC. ASIC means Application- Specific Integrated Circuit and refers to an integrated circuit customized for a particular use. To be configured, the FPGA has to be described by a logic circuit diagram or a source code using a hardware description language. Table 2.1 summarizes the pros and the cons of the FPGAs compared to ASICs.

Pros	Cons
Time-to-market shorter than ASICs	Slower than ASICs
Development cost cheaper than ASICs	More expensive for mass productions than ASICs
Lower non-recurring engineering costs than ASICs	More power consuming than ASICs
Re-programmable in the field	Difficulty to program FPGA Platform
Performance/power relative to standard processors and DSPs	

Table 2.1 Pros and cons of FPGA compared to ASIC: despite their lower performances compared to ASICs, FPGAs are more flexible and easy to develop.

The requirements specified during the early design step of the project are the following:

- Fast processor for processing parallel tasks.
- Support/documentation/examples available and up to date.
- Free of charges or not expensive price.
- Input/output for the camera sensor.
- USB wire for programming/debugging the FPGA.
- Possibility to add a softcore processor.
- Possibility to add an embedded OS.
- SD card or MMC slot to add extra memory.

2.1.2 Altera DE2-115 Board

2.1.2.1 Layout and components

A photograph of the DE2-115 board is shown in Fig 2.1 and Fig 2.2 It depicts the layout of the board and indicates the location of the connectors and key components.

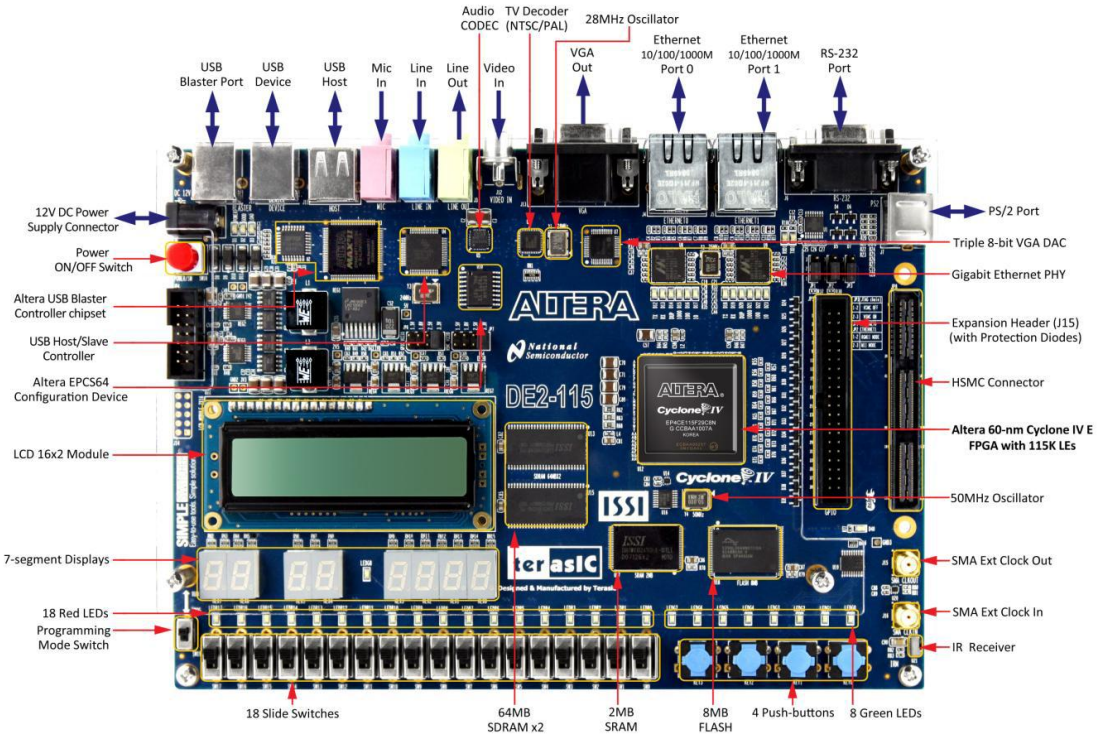


Fig 2.1 DE2-115 board (top view)

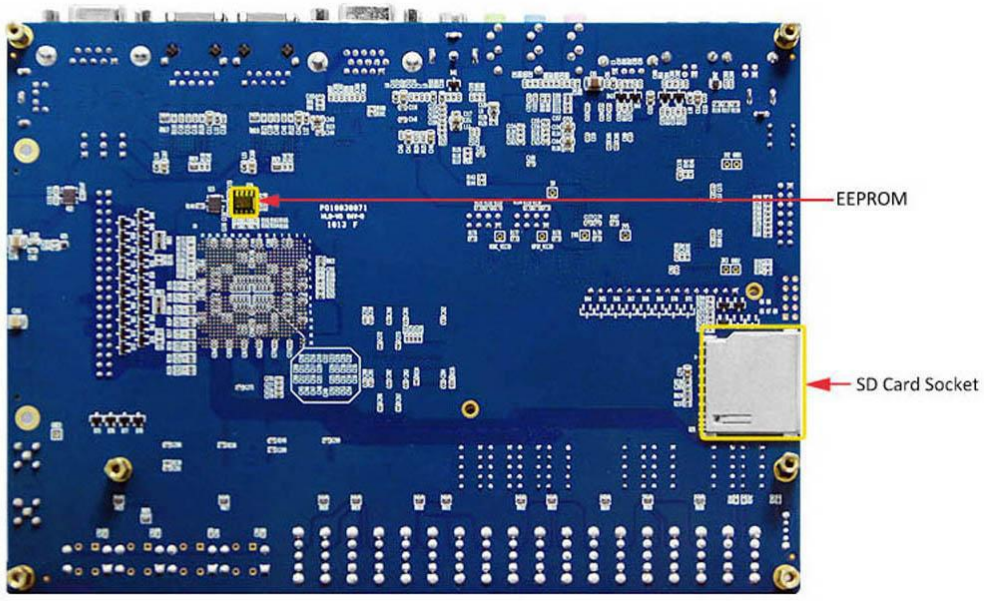


Fig 2.2 De2-115 board (bottom level)

The DE2-115 board has many features that allow users to implement a wide range of designed circuits, from simple circuits to various multimedia projects. The following hardware is provided on the DE2-115 board:

- Altera Cyclone® IV 4CE115 FPGA device
- Altera Serial Configuration device – EPCS64
- USB Blaster (on board) for programming; both JTAG and Active Serial (AS) programming modes are supported
- 2MB SRAM
- Two 64MB SDRAM
- 8MB Flash memory
- SD Card socket
- 4 Push-buttons
- 18 Slide switches
- 18 Red user LEDs
- 9 Green user LEDs
- 50MHz oscillator for clock sources
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (8-bit high-speed triple DACs) with VGA-out connector
- TV Decoder (NTSC/PAL/SECAM) and TV-in connector
- 2 Gigabit Ethernet PHY with RJ45 connectors
- USB Host/Slave Controller with USB type A and type B connectors
- RS-232 transceiver and 9-pin connector
- PS/2 mouse/keyboard connector

- IR Receiver
- 2 SMA connectors for external clock input/output
- One 40-pin Expansion Header with diode protection
- One High Speed Mezzanine Card (HSMC) connector
- 16x2 LCD module

In addition to these hardware features, the DE2-115 board has software support for standard I/O interfaces and a control panel facility for accessing various components. Also, the software is provided for supporting a number of demonstrations that illustrate the advanced capabilities of the DE2-115 board.

2.1.2.2 Block Diagram of Altera DE2-115 board

Fig 2.3 gives the block diagram of the DE2-115 board. All connections are made through the Cyclone IV E FPGA device. Thus, the user can configure the FPGA to implement any system design.

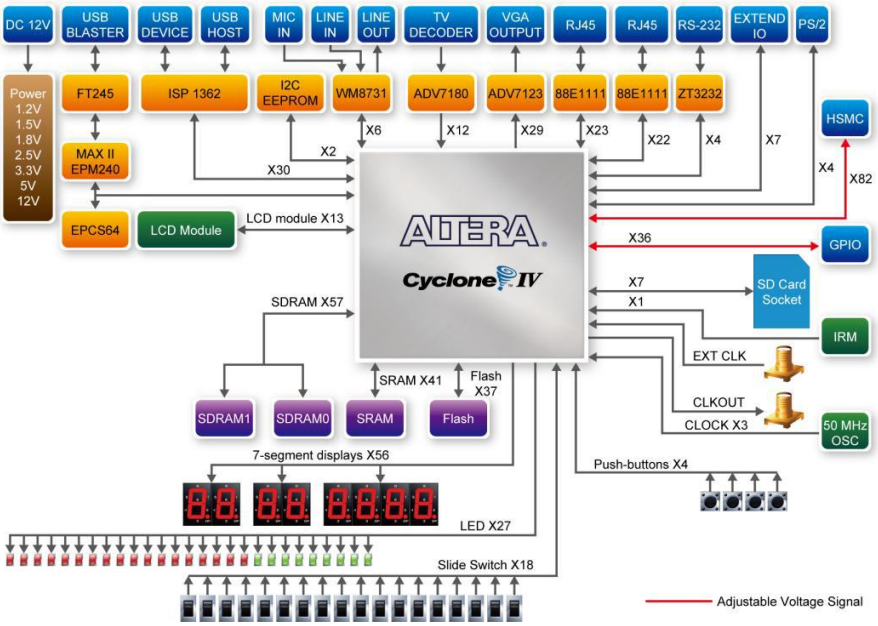


Figure 2.3 DE2-115 block diagram

The following is the details of each block

FPGA Device

- Cyclone IV EP4CE115F29 device
- 114,480 LEs
- 432 M9K memory blocks
- 3,888 Kbits embedded memory
- 4 PLLs

FPGA configuration
- JTAG and AS mode configuration
- EPCS64 serial configuration device
- On-board USB Blaster circuitry

Memory devices
- 128MB (32Mx32bit) SDRAM
- 2MB (1Mx16) SRAM
- 8MB (4Mx16) Flash with 8-bit mode
- 32Kb EEPROM

SD card socket
- Provides SPI and 4-bit SD mode for SD Card access

Connectors
- Two Ethernet 10/100/1000 Mbps ports
- High Speed Mezzanine Card (HSMC)
- Configurable I/O standards (voltage levels:3.3/2.5/1.8/1.5V)
- USB type A and B
- Provide host and device controllers compliant with USB 2.0
- Support data transfer at full-speed and low-speed
- PC driver available
- 40-pin expansion port
- Configurable I/O standards (voltage levels:3.3/2.5/1.8/1.5V)
- VGA-out connector
- VGA DAC (high speed triple DACs)
- DB9 serial connector for RS-232 port with flow control
- PS/2 mouse/keyboard

Clock
- Three 50MHz oscillator clock inputs
- SMA connectors (external clock input/output)

Audio
- 24-bit encoder/decoder (CODEC)
- Line-in, line-out, and microphone-in jacks

Display
- 16x2 LCD module

Switches and indicators
- 18 slide switches and 4 push-buttons switches
- 18 red and 9 green LEDs
- Eight 7-segment displays

Other features

- Infrared remote-control receiver module
- TV decoder (NTSC/PAL/SECAM) and TV-in connector

Power

- Desktop DC input
- Switching and step-down regulators LM3150MH

2.1.2.3 DE2-115 Control Panel

The DE2-115 board comes with a Control Panel facility that allows users to access various components on the board from a host computer. The host computer communicates with the board through a USB connection. The facility can be used to verify the functionality of components on the board or be used as a debug tool while developing RTL code.

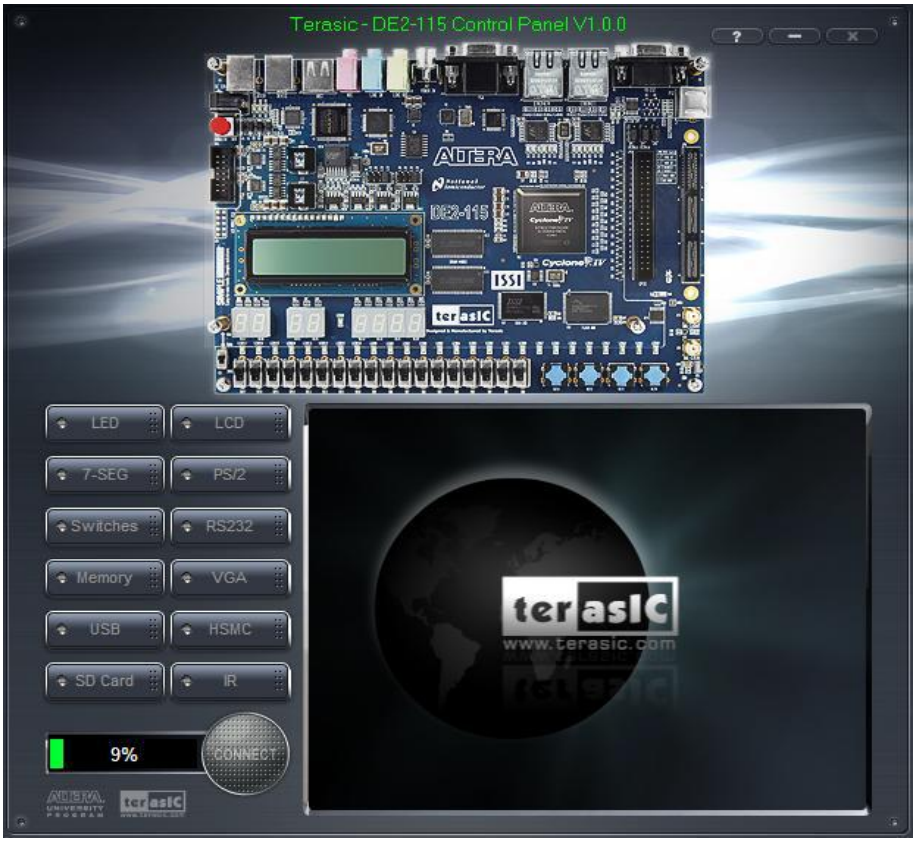


Fig 2.4 DE2-115 control panel

The concept of the DE2-115 Control Panel is illustrated in Figure 2.5. The "Control Circuit" that performs the control functions is implemented in the FPGA board. It communicates with the Control Panel window, which is active on the host computer, via the USB Blaster link. The graphical interface is used to issue commands to the control circuit. It handles all requests and performs data transfers between the computer and the DE2-115 board.

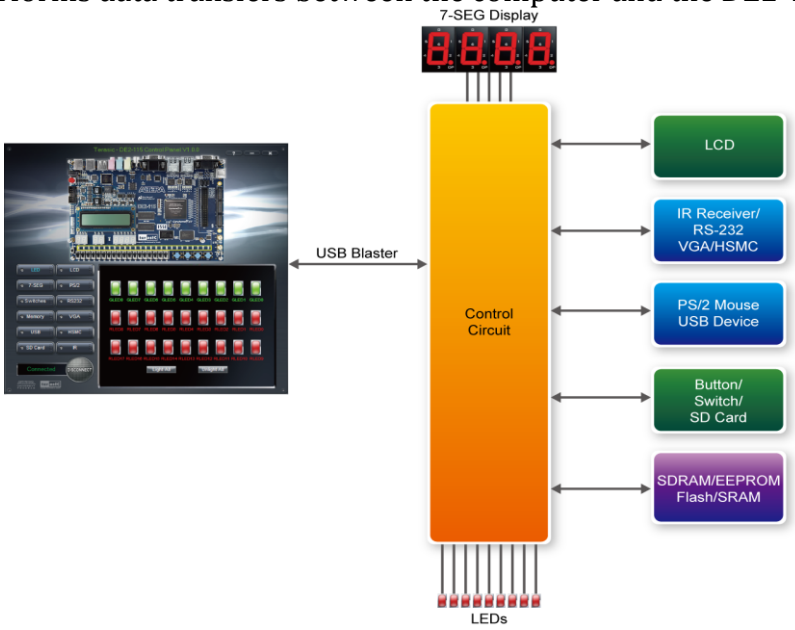


Fig 2.5 The DE2-115 Control panel concepts

The DE2-115 Control Panel can be used to light up LEDs, change the values displayed on 7-segment and LCD displays, monitor buttons/switches status, read/write the SDRAM, SRAM, EEPROM and Flash Memory, monitor the status of an USB device, communicate with the PS/2 mouse, output VGA color pattern to VGA monitor, verify functionality of HSMC connector I/Os, communicate with PC via RS-232 interface and read SD Card specification information. The feature of reading/writing a word or an entire file from/to the Flash Memory allows the user to develop multimedia applications (Flash Audio Player, Flash Picture Viewer) without worrying about how to build a Memory Programmer.

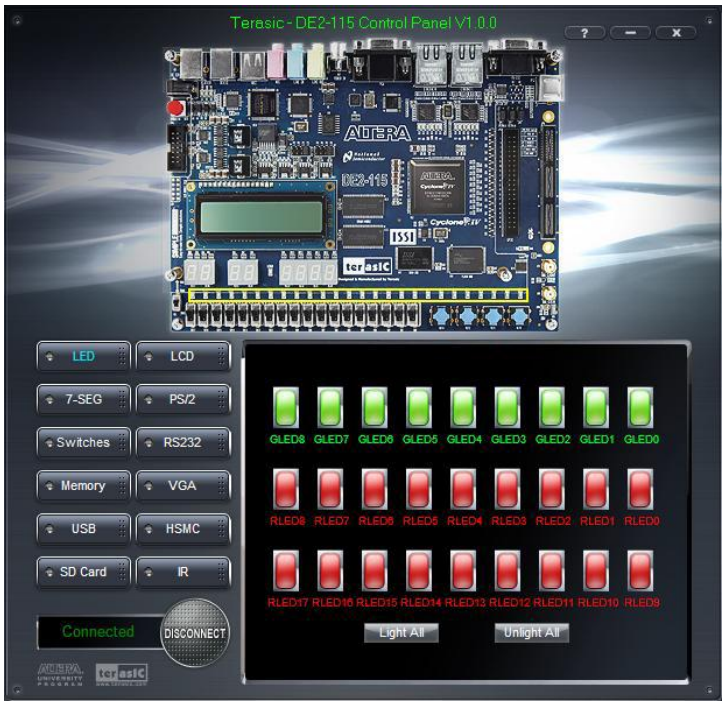


Fig 2.6 Controlling the LEDs

SDRAM/SRAM/EEPROM / Flash controller and programmer

The Control Panel can be used to write/read data to/from the SDRAM, SRAM, EEPROM, and Flash chips on the DE2-115 board. As an example, we will describe how the SDRAM may be accessed; the same approach is used to access the SRAM, EEPROM, and Flash. Click on the Memory tab and select “SDRAM” to reach the window in.

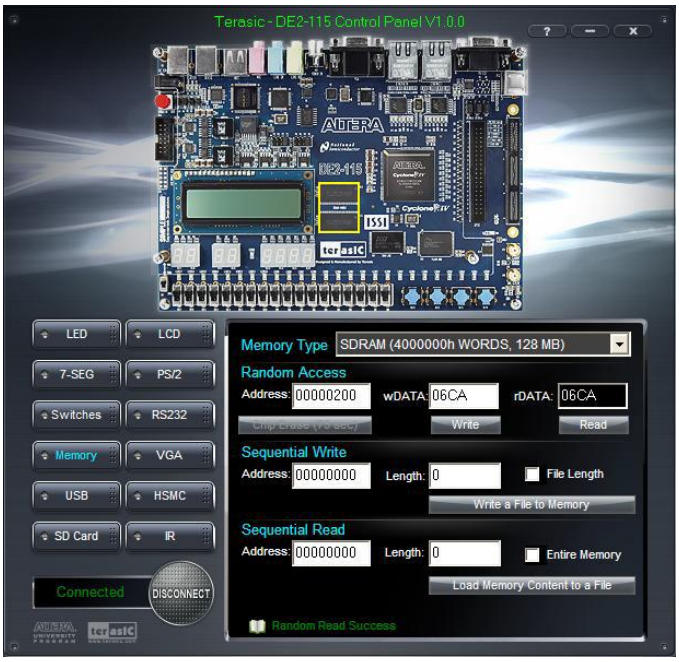


Fig 2.7 Accessing the SDRAM

A 16-bit word can be written into the SDRAM by entering the address of the desired location, specifying the data to be written, and pressing the Write button. Contents of the location can be read by pressing the Read button Figure 2.7 depicts the result of writing the hexadecimal value 06CA into offset address 200, followed by reading the same location.

The Sequential Write function of the Control Panel is used to write the contents of a file into the SDRAM as follows:

1- Specify the starting address in the Address box.

2- Specify the number of bytes to be written in the Length box. If the entire file is to be loaded, then a checkmark may be placed in the File Length box instead of giving the number of bytes.

3- To initiate the writing process, click on the Write a File to Memory button.

4- When the Control Panel responds with the standard Windows dialog box asking for the source file, specify the desired file in the usual manner.

The Control Panel also supports loading files with a .hex extension. Files with a .hex extension are ASCII text files that specify memory values using ASCII characters to represent hexadecimal values. For example, a file containing the line

0123456789ABCDEF

Defines eight 8-bit values: 01, 23, 45, 67, 89, AB, CD, EF. These values will be loaded consecutively into the memory.
The Sequential Read function is used to read the contents of the SDRAM and fill them into a file as follows:

1- Specify the starting address in the Address box.
2- Specify the number of bytes to be copied into the file in the Length box. If the entire contents of the SDRAM are to be copied (which involves all 128 Mbytes), then place a checkmark in the Entire Memory box.
3- Press Load Memory Content to a File button.
4- When the Control Panel responds with the standard Windows dialog box asking for the destination file, specify the desired file in the usual manner.

Users can use the similar way to access the SRAM, EEPROM and Flash.

Overall Structure of the DE2-115 Control Panel

The DE2-115 Control Panel is based on a Nios II SOPC system instantiated in the Cyclone IV E FPGA with software running on the on-chip memory. The software part is implemented in C code; the hardware part is implemented in Verilog HDL code with SOPC builder. The source code is not available on the DE2_115 System CD. To run the Control Panel, users should make the configuration according to Control panel setup Section. Fig 2.8 depicts the structure of the Control Panel. Each input/output device is controlled by the Nios II Processor instantiated in the FPGA chip. The communication with the PC is done via the USB Blaster link. The Nios II interprets the commands sent from the PC and performs the corresponding actions.

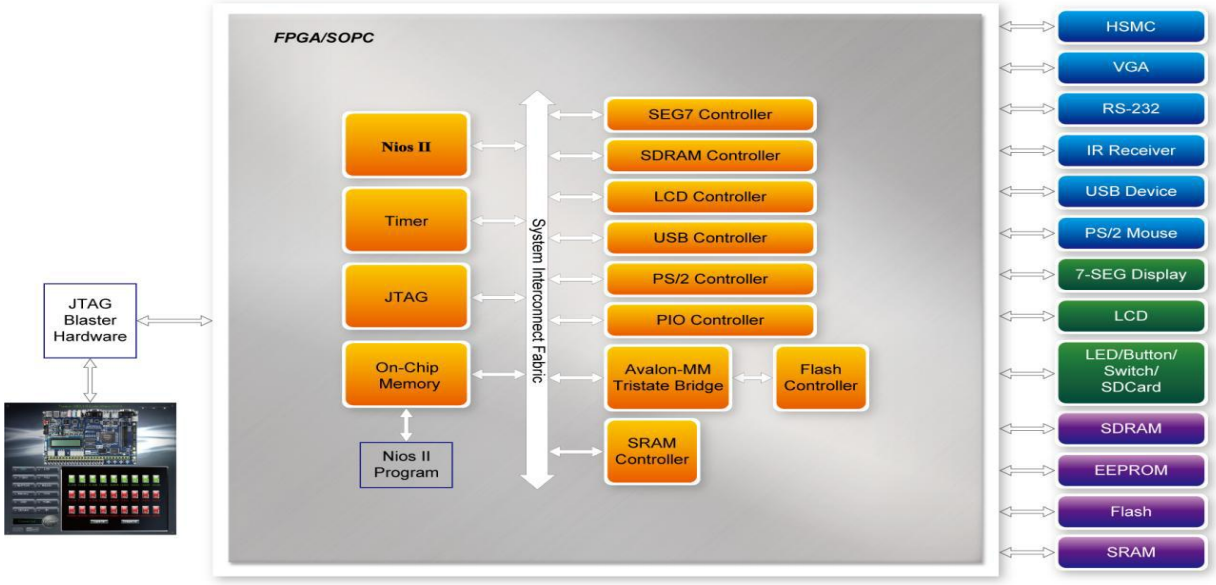


Fig 2.8 The block diagram of the DE2-115 control panel

2.1.3. The TRDB_D5M Camera

The kit contains hardware design in Verilog and software to load the picture taken into a PC and save it as a BMP or JPG file. The image raw data is sent from D5M to the DE2_115 board. The FPGA on the DE2_115 board is handling image processing part and converts the data to RGB format to display on the VGA monitor.

Fig 2.9 TRDB D5M

2.1.3.1 Block Diagram of the reference design

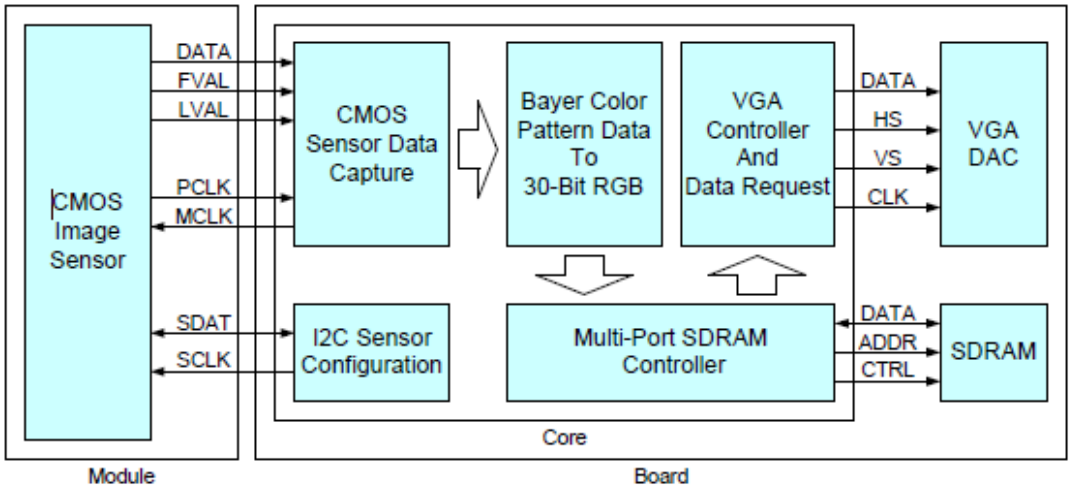


Fig 2.10 the block diagram of the digital camera design

2.1.3.2 Main features of D5M

1- Reset

The TRDB-D5M may be reset by either using RESETn (active LOW) or the Reset register.

a- Hard Reset
Assert (LOW) RESETn pin, it is not necessary to clock the device. All registers return to the factory defaults. When the pin is negated (HIGH), the chip resumes normal operation.

b- Soft Reset

2- Power Up and Power Down

3- PLL

PLL Generated Master Clock

The PLL contains a prescaler to divide the input clock applied on XCLKIN, a VCO to multiply the prescaler output, and a another divider stage to generate the output clock. The clocking structure is shown is Figure 2.11. PLL control registers (in courier font) can be programmed to generate desired master clock frequency. The PLL control registers must be programmed while the sensor is in the software Standby state. The effect of programming the PLL divisors while the sensor is in the streaming state is UNDEFINED.

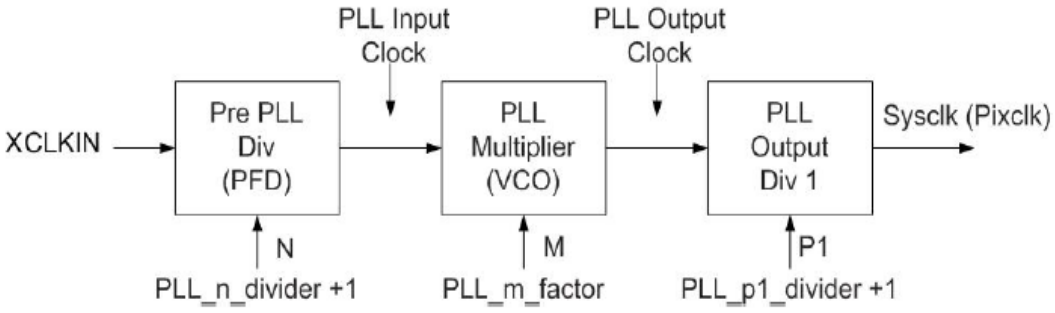


Fig 2.11 PLL Generated Master Clock

4- Standby and Chip Enable

The TRDB-D5M can be put in a low-power Standby state by clearing the Chip_Enable register field (Reg0x07[1] = 0). When the sensor is put in standby, all internal clocks are

gated, and analog circuitry is put in a state that it draws minimal power. The two-wire serial interface remains minimally active so that the Chip_Enable bit can subsequently be cleared. Reads cannot be performed and only the Chip_Enable and Invert_Standby registers are writable. If the sensor was in continuous mode when put in standby, it resumes from where it was when standby is deactivated. Naturally, this frame and the next frame are corrupted, though the sensor itself does not realize this. As this could affect automatic black level calibration, it is recommended that either the chip be paused(by setting Restart_Pause) before being put in standby mode, or it be restarted (setting Restart) upon resumption of operation.

5- Full-Array Readout

The entire array, including dark pixels, can be read out without digital processing or automatic black level adjustments.

6- Window Control

The output image window of the pixel (the field of view or FOV) is defined by four register fields. Column_Start and Row_Start define the X and Y coordinates of the upperleft corner of the FOV. Column_Size defines the width of the FOV, and Row_Size defines the height of the FOV in array pixels.

7- Readout Modes

a- Subsampling
b- Skipping
c- Binning

8- Mirror

a- Column Mirror Image

By setting Reg0x20[14] = 1, the readout order of the columns are reversed as shown in Figure 2.12. The starting color, thus Bayer pattern, is preserved when mirroring the columns.

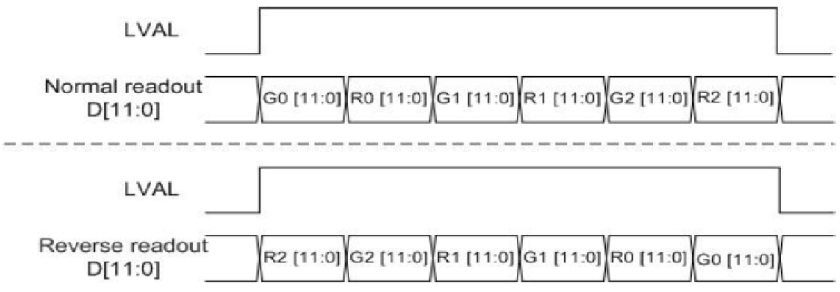


Fig 2.12 Six Pixels in Normal and Column Mirror Readout Modes

b- Row Mirror Image

By setting Reg0x20[15] = 1, the readout order of the rows are reversed as shown in Figure 2.13. The starting color, thus Bayer pattern, is preserved when mirroring the rows.

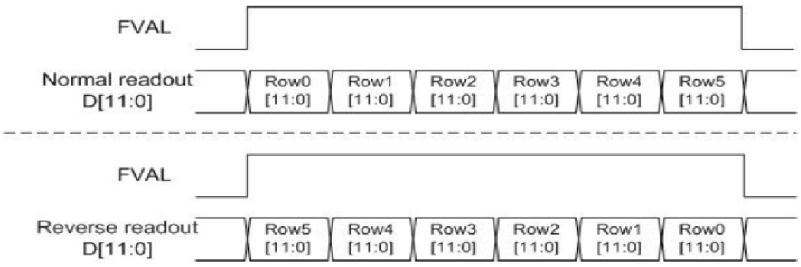


Fig 2.13 Six Rows in Normal and Row Mirror Readout Modes

9- Maintaining a Constant Frame Rate

Maintaining a constant frame rate while continuing to have the ability to adjust certain parameters is the desired scenario. This is not always possible, however, since register updates are synchronized to the read pointer, and the shutter pointer for a frame is usually active during the readout of the previous frame. Therefore, any register changes that could affect the row time or the set of rows sampled causes the shutter pointer to start over at the beginning of the next frame.

10- Synchronizing Register Writes to Frame Boundaries

Changes to most register fields that affect the size or brightness of an image take effect on the frame after the one during which they are written. To ensure that a register update takes effect on the next frame, the write operation must be completed after the leading edge of FRAME_VALID and before the trailing edge of FRAME_VALID.

11- Restart

12- Image Acquisition Modes

13- Exposure

The nominal exposure time, tEXP, is the effective shutter time in ERS modes, and is defined by the shutter width, SW, and the shutter overhead, SO, which includes the effect of Shutter_Delay. Exposure time for other modes is defined relative to this time. Increasing Shutter_Delay (SD) decreases the exposure time.

14- Operating Modes

In the default operating mode, the TRDB-D5M continuously samples and output frames. It can be put in "snapshot" or triggered mode by setting Snapshot, which means that it samples and outputs a frame only when triggered. To leave snapshot mode, it is necessary to first clear Snapshot then issue a Restart.

Mode	Settings	Description
ERS Continuous	Default	Frames are output continuously at the frame rate defined by ^{t}FRAME. ERS is used, and the exposure time is electronically controlled to be ^{t}EXP.
ERS Snapshot	Snapshot = 1	Frames are output one at a time, with each frame initiated by a trigger. ERS is used, and the exposure time is electronically controlled to be ^{t}EXP.
ERS Bulb	Snapshot = 1; Bulb_Exposure = 1	Frames are output one at a time, with each frame's exposure initiated by a trigger. ERS is used. End of exposure and readout are initiated by a second trigger.
GRR Snapshot	Snapshot = 1; Global_Reset = 1	Frames are output one at a time, with each frame initiated by a trigger. GRR is used. Readout is electronically triggered based on SW.
GRR Bulb	Snapshot = 1; Bulb_Exposure = 1; Global_Reset = 1	Frames are output one at a time, with each frame initiated by a trigger. GRR is used. Readout is initiated by a second trigger.

Table 2.2 Operating Modes

15- Signal Chain and Datapath

16- Test Patterns

The TRDB-D5M has the capability of injecting a number of test patterns into the top of the datapath in order to debug the digital logic. With one of the test patterns activated, any of the datapath functions can be enabled in order exercise it in a deterministic fashion. Test patterns are enabled when Enable_Test_Pattern is set. Only one of the test patterns can be enabled at a given point in time by setting the Test_Pattern_Mode register.

D5M Serial Bus description

17- Serial Bus Description

Protocol

The two-wire serial defines several different transmission codes, as follows:

- Start bit
- The slave device 8-bit address
- A(n) (no) acknowledge bit
- An 8-bit message
- A stop bit

18- Two-Wire Serial Register Interface

The electrical characteristics of the two-wire serial register interface (SCLK, SDATA) are shown in Table 2.3 and Figure 2.14.

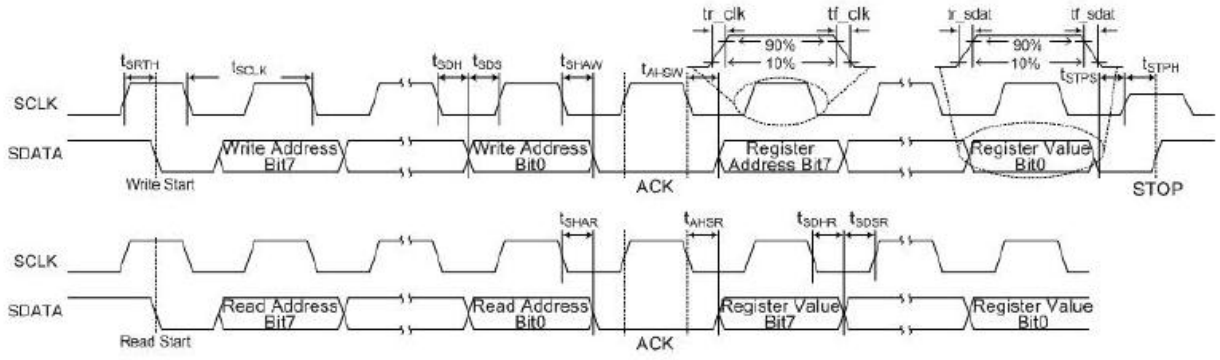


Fig 2.14 Two-Wire Serial Bus Timing Parameters

Symbol	Definition	Condition	Min	Typ	Max	Unit
^{f}SCLK	Serial interface input clock frequency	—	—	—	400	kHz
^{t}SCLK	Serial Input clock period	—	—	—	2.5	µsec
	SCLK Duty Cycle	—	40	50	60	%
tr_sclk	SCLK rise time		—	34	—	ns
tf_sclk	SCLK fall time		—	8	—	ns
tr_sdat	SDATA rise time		—	34	—	ns
tf_sdat	SDATA fall time		—	10	—	ns
^{t}SRTH	Start hold time	WRITE/READ	0	10	28	ns
^{t}SDH	SDATA hold	WRITE	0	0	0	ns
^{t}SDS	SDATA setup	WRITE	0	19.9	59.9	ns
^{t}SHAW	SDATA hold to ACK	WRITE	279	281	300	ns
^{t}AHSW	ACK hold to SDATA	WRITE	279	281	300	ns
^{t}STPS	Stop setup time	WRITE/READ	0	0	0	ns
^{t}STPH	Stop hold time	WRITE/READ	0	0	0	ns
^{t}SHAR	SDATA hold to ACK	READ	279	284	300	ns
^{t}AHSR	ACK hold to SDATA	READ	279	284	300	ns
^{t}SDHR	SDATA hold	READ	0	0	0	ns
^{t}SDSR	SDATA setup	READ	0	19.9	59.9	ns
CIN_SI	Serial interface input pin capacitance	—	—	3.5	—	pF
CLOAD_SD	SDATA max load capacitance	—	—	15	—	pF
RSD	SDATA pull-up resistor	—	—	1.5	—	kΩ

Table 2.3 Two-Wire Serial Bus Characteristics

2.2 Hardware tools

2.2.1 Quartus II

Quartus II is a design software from Altera that provides a complete, multi platform design environment for FPGAs. A typical design flow is: Design Entry, Synthesis, Place and Route, Timing Analysis, Simulation and Programming and Configuration. The Quartus II software includes solutions for all these design phases and a graphical user interface. In the Design Entry process the hardware can either be described completely by using a hardware descriptive language like VHDL, or in combination with a ”drawing” environment, where the designer can place and connect different blocks by using the PC mouse. This tool is very useful and makes interconnection between the modules easy. Synthesis and place and route can be done with either Altera’s own tools or the designer can select a different tool like Synplify from Synopsys. Such dedicated tools can provide additional improvements of the design. For simulation, Altera provides a build-in environment. In addition Altera also support the possibility to use other simulation tools inside Quartus II. ModelSim.

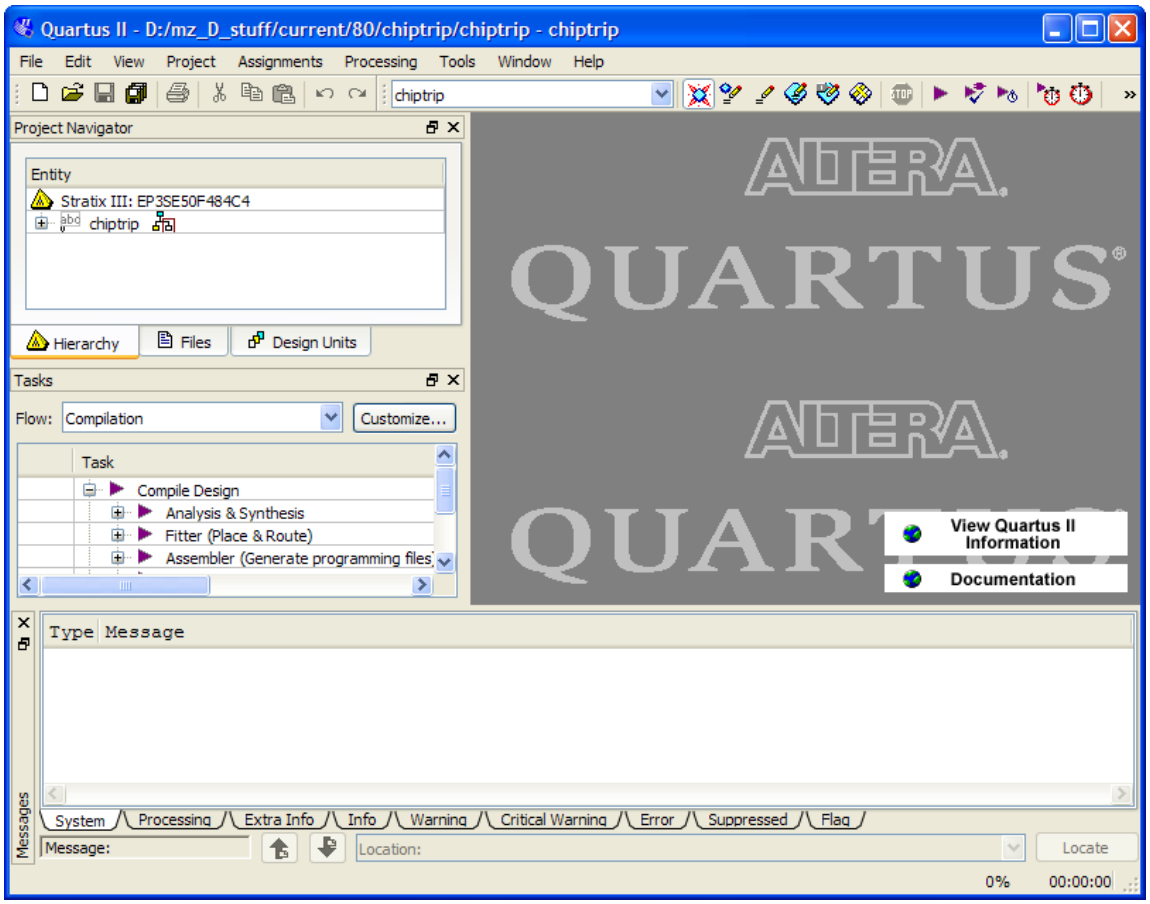


Fig 2.15 Quartus II Graphical User Interface

2.2.1.1. RTL viewer

This tool was very helpful to visualize our designs, The RTL Viewer displays the Analysis & Elaboration results for Verilog HDL or VHDL designs, and AHDL Text Design Files (**.tdf**), Block Design Files (**.bdf**), and Graphic Design Files (**.gdf**). For Verilog Quartus Mapping Files or EDIF netlist files that were generated from other EDA synthesis tools, the RTL Viewer displays the hierarchy for the atom representations of WYSIWYG primitives. the RTL Viewer has a hierarchy list that lists the instances, primitives, pins, and nets for the entire design netlist Figure 2.16.

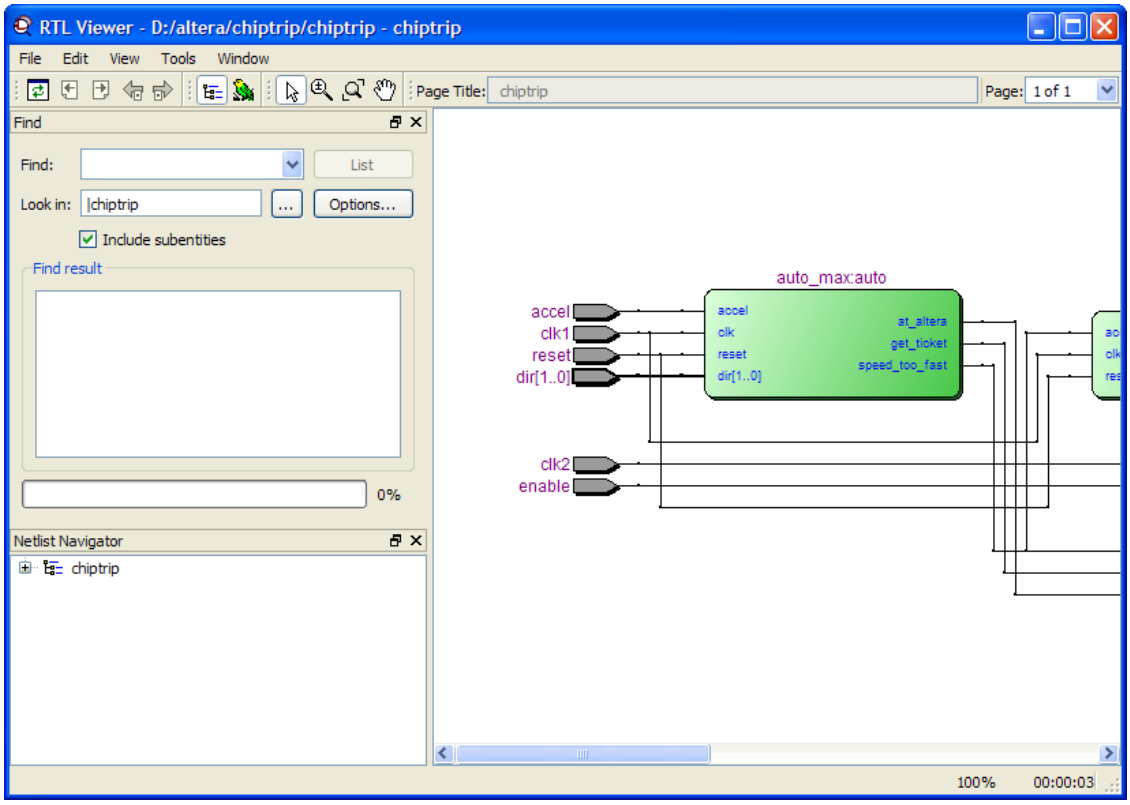


Fig 2.16 RTL Viewer

2.2.1.2. Timing analysis and design optimization

The Quartus II TimeQuest Timing Analyzer allows analyzing the timing characteristics of the design. The TimeQuest analyzer uses industry-standard Synopsys Design Constraint (SDC) methodology for constraining designs and reporting results. The information

generated by the timing analyzer can be used to analyze, debug, and validate the timing performance of the design. The TimeQuest analyzer provides an intuitive and easy-to-use GUI that allows constraining and analyzing designs efficiently. The GUI is divided into the following four panes:

- View pane
- Tasks pane
- Console
- Report pane

Each pane provides features that enhance the productivity of performing static timing analysis in the TimeQuest analyzer Figure 2.17.

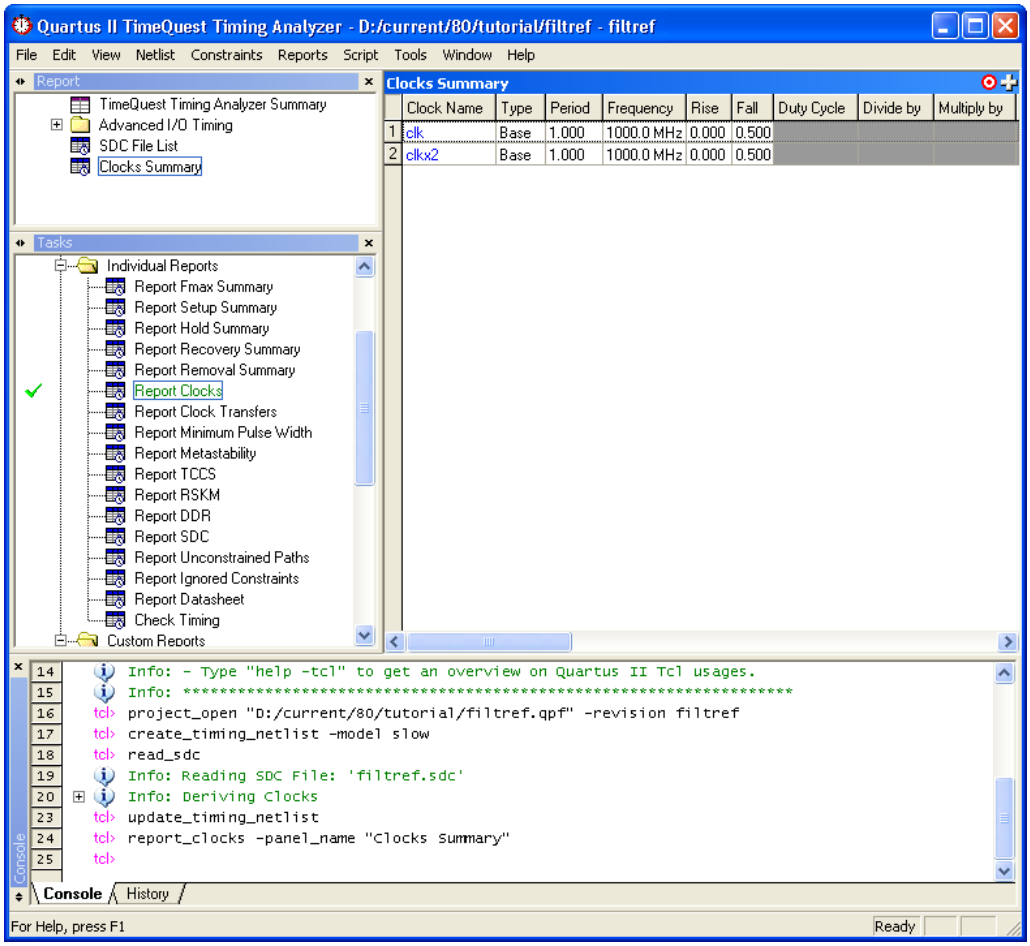


Fig 2.17 TimeQuest Timing Analyzer Window

Clocks Summary

	Clock Name	Type	Period	Frequency	Rise	Fall
1	clk	Base	1.000	1000.0 MHz	0.000	0.500
2	clkx2	Base	1.000	1000.0 MHz	0.000	0.500

Fig 2.18 Output from Report Clocks Shown in the View Pane

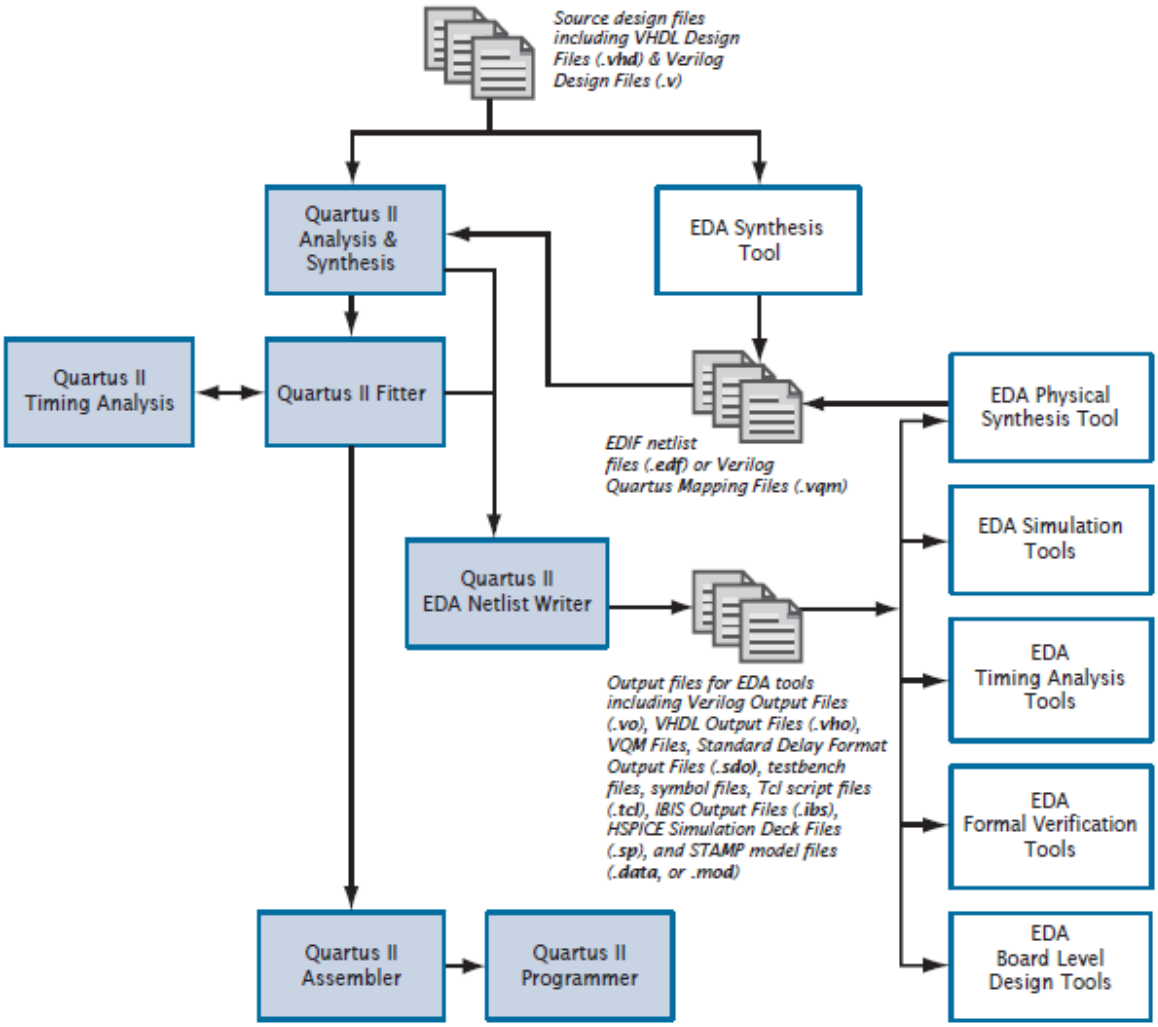


Fig 2.19 EDA Tool Design Flow

2.2.1.3 SOPC builder

The SOPC builder is a part of the Quartus II design environment and it lets the designer define and generate a complete system in much less time than usual by including available IP (Intellectual Property) modules in the design. The tool offers both the possibility to include the IP cores in the design as well as the ability to configure them to fit the

designer's needs. In addition to configuring the cores, the SOPC builder also supports a connection function which allows the designer to simply click on how the cores should be connected to each other. SOPC builder then automatically generates a top-level HDL file that connects the different modules together. Altera Nios II soft processor core is implemented by using this tool. SOPC Builder automates the task of integrating hardware components. Using traditional design methods, you must manually write HDL modules to wire together the pieces of the system. Using SOPC Builder, you specify the system components in a GUI and SOPC Builder generates the interconnect logic automatically. SOPC Builder generates HDL files that define all components of the system, and a top-level HDL file that connects all the components together. SOPC Builder generates either Verilog HDL or VHDL equally. In addition to its role as a system generation tool, SOPC Builder provides features to ease writing software and to accelerate system simulation.

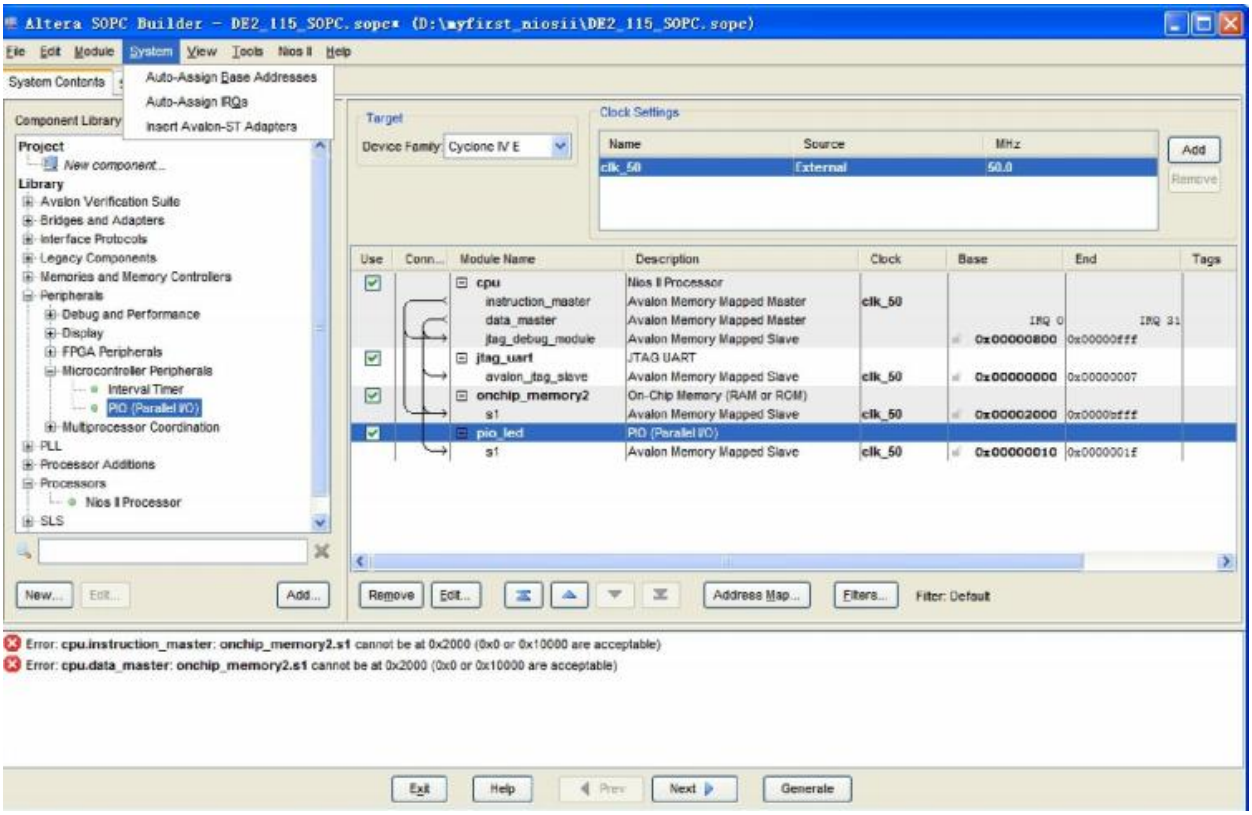


Fig 2.20 Example SOPC System

Available Components

Altera and third-party developers provide ready-to-use SOPC Builder components, including:

■ Microprocessors, such as the Nios II processor

■ Microcontroller peripherals, such as a Scatter-Gather DMA Controller and timer

■ Serial communication interfaces, such as a UART and a serial peripheral interface

■ General purpose I/O

■ Communications peripherals, such as a 10/100/1000 Ethernet MAC

■ Interfaces to off-chip devices

Custom Components

SOPC builder provides importing HDL modules and entities that are written using Verilog HDL or VHDL into SOPC builder as custom components. Once you have created an SOPC Builder component, you can use the component in other SOPC Builder systems, and share the component with other design teams.

2.2.1.4. Model-SIM

Model-SIM is a unified debug environment for hardware descriptive languages (HDL) from Mentor Graphics. The tool uses single kernel simulator (SKS) technology which is said to give it good performance. In the verification process of a design the designer can use Model-SIM to simulate and verify the behavior of a hardware description. This is an important step in order to get rid of potential errors and a great way to ensure that the system behave as intended. Model-SIM also runs any test bench code written in a HDL. The simulation is displayed as a digital timing diagram like Figure 2.21.

Altera delivers a version of Model-SIM called Model-SIM-Altera Edition that comes with a pre-built library many of the mega functions available through Altera's MegaFunctions. The Model-SIM -Altera comes in three different solutions: Web edition, starter edition and Altera edition. It is the two latter versions that currently are under future development. The main differences are simulation performance, design size limitations and price. Two of the main advantages of using Model-SIM over Quartus II simulation tool is its performance, especially in large designs, and its ability to run test bench code.

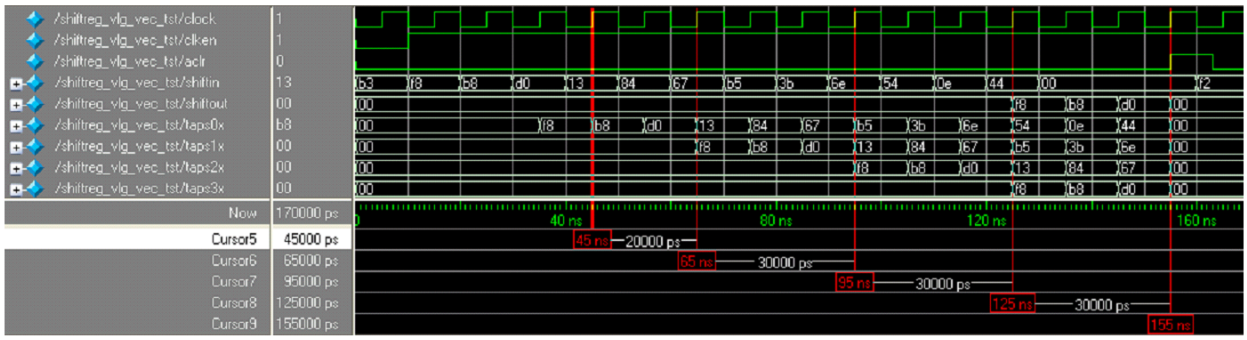


Fig 2.21 Test bench example

2.2.1.5. Nios II Embedded Design Suite

The Nios II EDS is a fully integrated development environment for developing software to Altera's Nios II embedded processor. The environment is based on the industry's Eclipse IDE. The Nios II specific functionality is included as plug-ins. Following is a list of these plug-ins:

- Nios II Software Templates
- Nios II Flash Programmer
- Nios II BSP Editor
- Quartus II Programmer
- Nios II Command Shell

The Nios II EDS provides two distinct development flows and includes many proprietary and open-source tools for creating Nios II programs. Among these tools are the GNU C/C++ tool chain for compiling the C or C++ software language. The Nios II EDS also automates the creation of BSP, and functions as an editor for altering this configuration. There are also provided different software templates for examples on how to use the Nios II. The Nios II command shell is used to display messages and it also functions as a terminal when running code on a Nios II processor. The terminal will display any message printed using C library functions like printf(). Quartus II programmer makes it possible to program and run code on an implemented processor on an Altera FPGA.

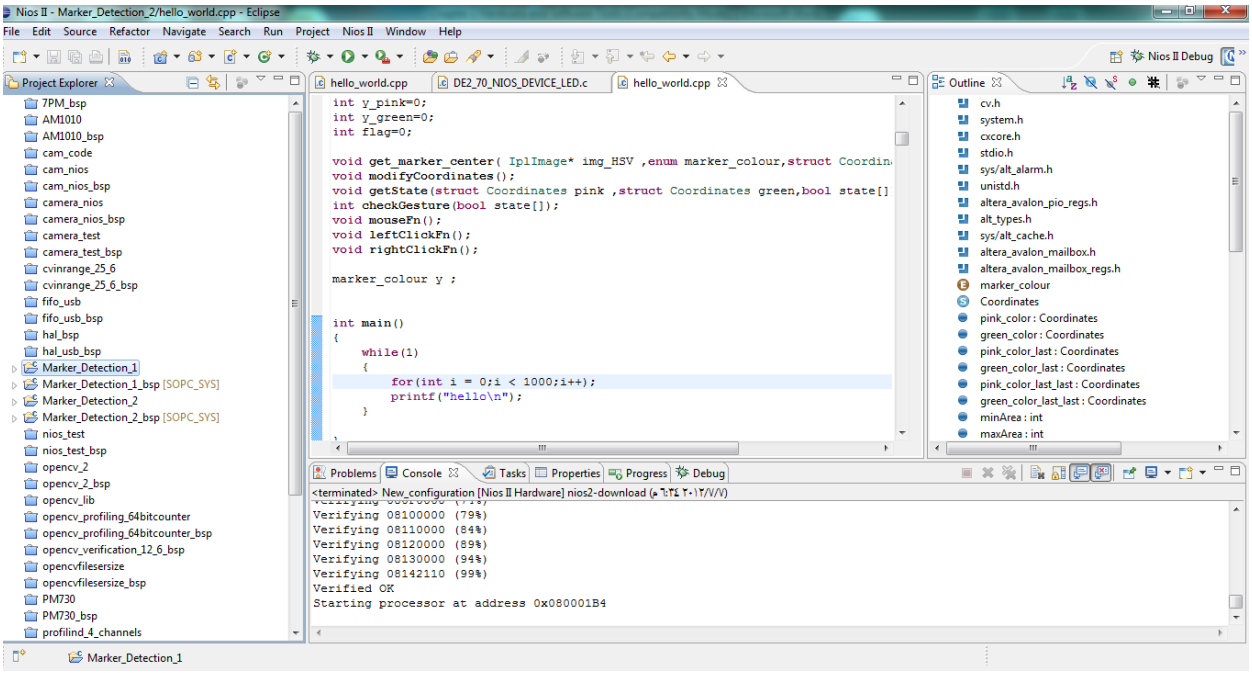


Fig 2.22 NIOS II SBT Main Window

2.2.1.6. BSP and HAL

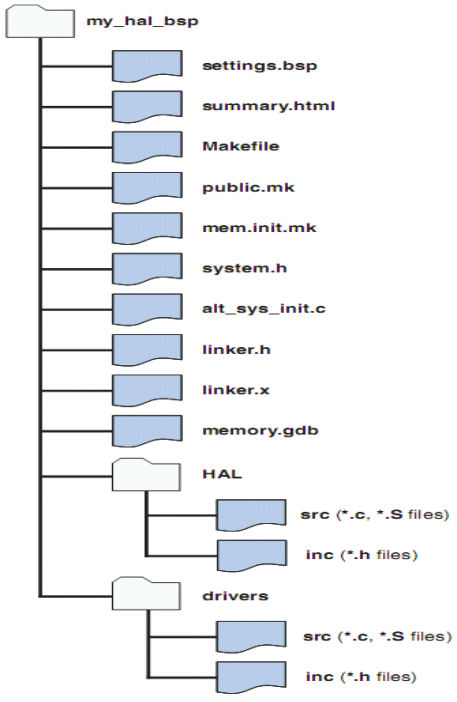


Fig 2.23 HAL BSP after Generating Files

The HAL provides the following services:

■ Integration with the newlib ANSI C standard library—Provides the familiar Cstandard library functions

■ Device drivers—Provides access to each device in the system

■ The HAL API—Provides a consistent, standard interface to HAL services, such as device access, interrupt handling, and alarm facilities

■ System initialization—Performs initialization tasks for the processor and the runtime environment before main()

■ Device initialization—Instantiates and initializes each device in the system before main() runs.

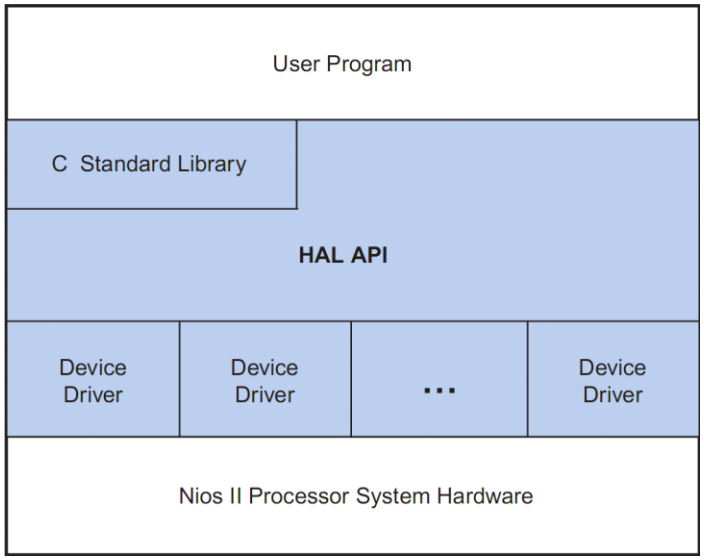


Figure 2.24 The layers of a HAL-Based System

2.3 Software Tools

During initial phase of project multiple algorithms needed to be tested. Much more experimenting over implementing specific algorithm (research phase), that's why we started using Matlab to achieve the goal of detecting the most suitable algorithm to start implementing it using OpenCV and C/C++ language(Implementing phase).

2.3.1 Matlab

For the next reasons we used Matlab:

Faster Coding

1) Computer Vision Researchers need fast prototyping

In research environment, we have often new ideas, and we want to test them really quick to see if it's worth keeping on in that direction. And most often only a tiny sub-part of what we code will be useful. Moreover it's often not possible to guess beforehand if an idea is going to work or not. Matlab is often a bit slower at execution time and opencv is definitely the fastest for running time, but we don't care much. Because we don't know in advance what method is going to be successful, we have to try many things, so our bottle neck is programming time, because our code will most often run a few times to get the results to publish, and that's all.

2) IDE (Integrated Development Environment)

Matlab provides executing code with the command line which makes debugging faster. Matlab underlines errors before execution, to determine problems faster. It proposes some way to make the code faster. IDE includes a Profiler called KCachegrind , while in C++ is much more difficult to do Profiling compared to Mat lab IDE.With OpenCV/C++/Visual Studio, debugger does not allow executing code during debugging, which disables visualizing matrices and so on. So in practice, you have to copy paste some code to dump matrices, to check where the error is. This is much harder.

3) Concise code

Matlab code is more concise, which mean easier to debug, to read, and to understand. During making loops, everything is included in functions, which make the code much easier to read without indices. So that programmer can focus on what to program not how to program it.

4) Plotting tools

Matlab is famous for its plotting tools. They are very helpful. OpenCV has just the basic plotting functions.

2.3.2 Open-CV and C/C++ (Implementing Phase)

Open-CV is a computer vision library created by Intel for use with C/C++. It stands for Open Source Computer Vision Library, it's an open source library. It has many functions for manipulating and displaying images. The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Mat lab, and other languages. Open-CV was designed for computational efficiency and with a strong focus on real time applications. Open-CV is written in optimized C and can take advantage of multi-core processors. The Open-CV library contains over 500 functions that span many areas in vision, including factory medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-in- hand, Open-CV also contains a full, general-purpose Machine Learning Library (MLL). This sub-library is focused on statistical pattern recognition and clustering. Since Open-CV was created by Intel, it takes advantage of the Intel chip designs to dramatically decrease the processing time through parallel processing. Unfortunately, C/C++ programming is not as user friendly as Matlab. Matlab functions were replicated in C/C++. This way, they can be optimized to reduce the amount of computations and take advantage of Open-CV's quick matrix operations. When referencing memory in these functions, debugging and memory errors become much more frequent. Processing an image using C/C++ utilizing Open-CV is 80 times faster than processing the same image using Mat lab. This was achieved through reading and writing directly to memory, processing filter coefficients before the input image stream, reading coefficients from a table, and reducing unnecessary computations. This program takes advantage of C/C++ with Open-CV by implementing the algorithm in a low level language. Matlab processes the replicated images in addition to the original image, which doubles or triples the processing time. The program in C/C++ only replicates the amount of image necessary to convolve the original image.

Open-CV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:
core - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
imgproc - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-

based remapping), color space conversion, histograms, and so on.
video - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
calib3d - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
features2d - salient feature detectors, descriptors, and descriptor matchers.
objdetect - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
highgui - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.
gpu - GPU-accelerated algorithms from different OpenCV modules.

Chapter 3

System Analysis and Design

This chapter presents the system constraints analysis, in terms of FPGA, processing time and data consistency.

3.1 System Constraints

This section gives an overview of the different constraints of the system.

3.1.1 Time constraints

The system run time should be most optimal to be able to run as a real time system with convenient delay, since the system is a virtual mouse application then its run time should be convenient to the user as an interface. In a typical USB mouse the host polls the mouse for data each 8 milliseconds which means that we need to process a frame and extract relevant data from it each 8 milliseconds.

3.1.2 Memory (Data) constraints

We have a various set of on-chip and off-chip memories on the board; On the other hand we need to find out which memory will be used as:

1 – Software code storage (text).

2 – Data area for software during runtime (BSS, Stack, heap, rodata and rwdata).

3 – Frame buffer to store frames

3.1.3 User Constraints

User should take care of these constraints to have the best output from the system.

1- Markers:
User should wear certain colored markers to use them in front of the camera.

2- Distance between user and the camera should range between 0.5 and 1.5 meters to achieve best results.

3.1.4 Environment Constraints

The algorithm is heavily dependent on the surrounding conditions. Lighting and time of the day should be taken into consideration.

3.2 System Design

Figure 3.1 shows clearly th over view of our project and simply the user points infront of the camera and the FPGA translates this to movement of the mouse at the other side of the FPGA, at the laptop or computer side.

Figure 3.2 Shows in more design details each part of our system which will be expalined later on, each block is explained.

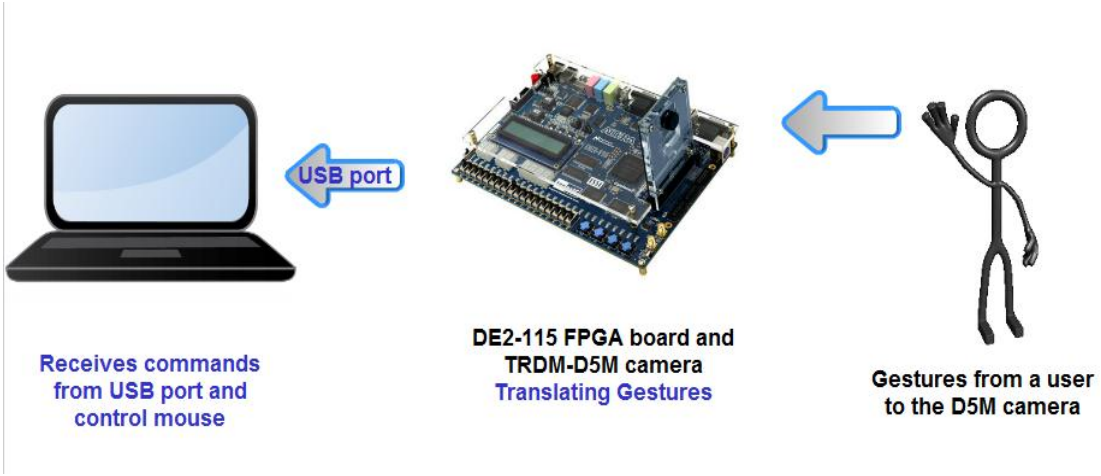


Figure 3.1 Abstract Diagram for the system

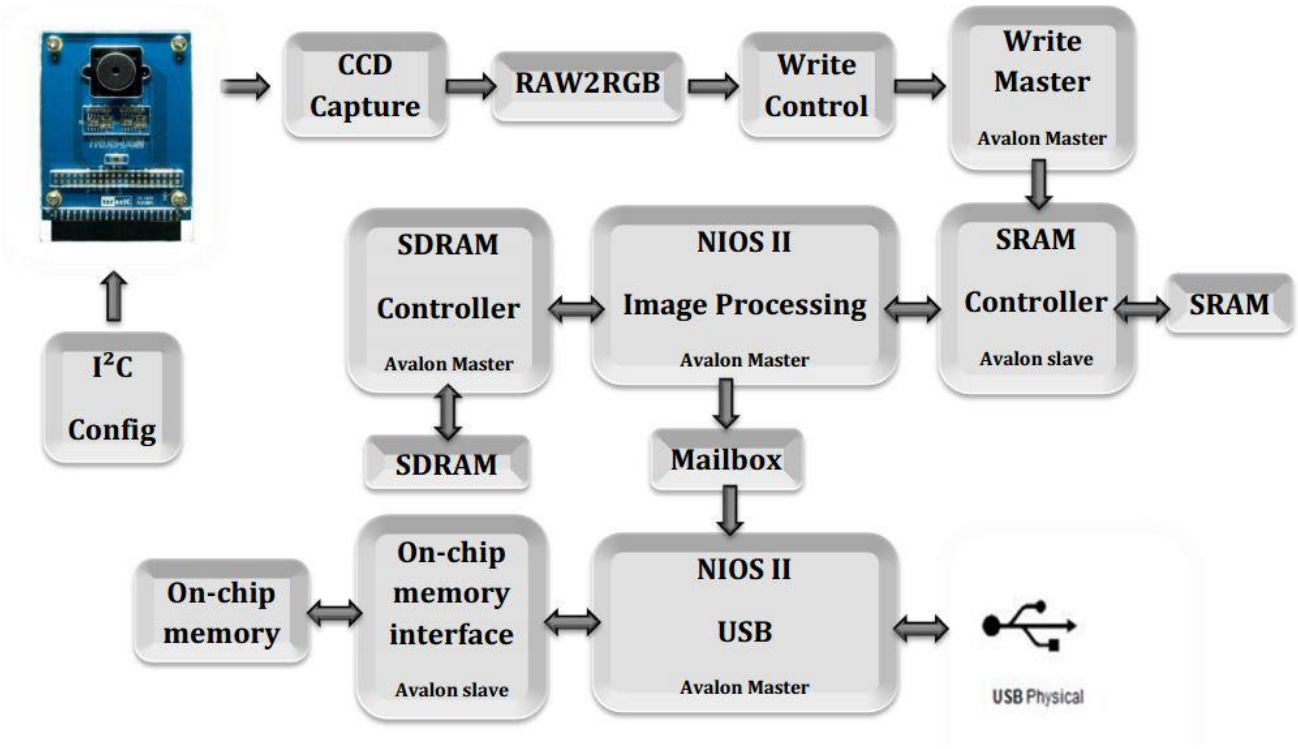


Fig 3.2 detailed block diagram for the system

3.2.1 The TRDB_D5M Camera

The TRDB_D5M Kit provides developing a 5 Mega Pixel Digital Camera on the Altera DE2_115 boards. The kit contains hardware design (in Verilog) and software to load the picture taken into a PC and save it as a BMP or JPG file. The image raw data is sent from D5M to the DE2_115 board. The FPGA on the DE2_115 board is handling image processing part and converts the data to RGB format to display on the DVI / VGA monitor.

3.2.2 Camera IP core

Mainly it is responsible for:

1- Configuring registers of camera for proper operation using I2C.

2- To transform raw data (in Bayer format) to RGB format.

3.2.3 Camera SRAM controller

It is responsible for interfacing between camera and SDRAM memory chip.

3.2.4 NIOS SRAM controller

It is responsible for interfacing between NIOS II Processor and SRAM memory chip.

3.2.5 SRAM memory chip

SRAM memory chip acts as the main frame buffer or the main storage unit, it is considered the used memory for both Frames from camera and program code and data from NIOS II processor.

3.2.6 NIOS II A (Image Processing Core)

It runs the main image processing algorithm implemented with open CV on frames from SRAM memory chip and outputs the position of the marker (hand) and sends this position to NIOS II B.

3.2.7 NIOS II B (USB Core)

NIOS II A outputs the marker position to NIOS II B and it encapsulates the position of marker and sends it to the ISP 1362 chip. Also it runs the firmware needed to use the ISP-1362 chip.

3.2.8 ISP-1362 chip

This chip is integrated with the USB host controller and USB device controller; the controllers are complaint with USB version 2. It is used as a device controller and constantly sends marker coordinates reports to the PC.

3.2.10 USB physical

The physical layer device that transmits commands to the host computer.

Chapter 4

Hand gesture recognition software system

The purpose of this chapter is to provide an overview of the hand gesture recognition system software.

4.1 System Description:

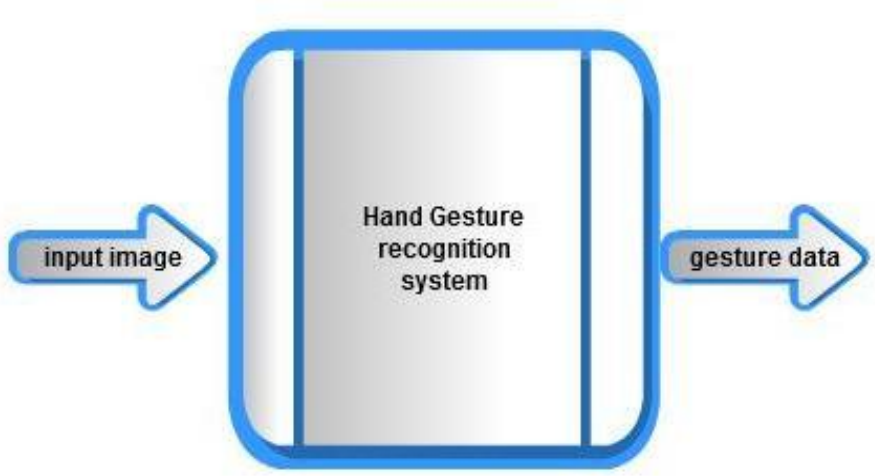


Fig 4.1 an abstract Hand gesture recognition block diagram

Our software is based on Vision-based automatic hand gesture recognition as it has been a very active research topic in recent years with motivating applications such as Human computer interaction (HCI). The general problem is quite challenging due a number of issues including the complicated nature of static and dynamic hand gestures, complex backgrounds, and occlusions. Attacking the problem in its generality requires elaborate algorithms requiring intensive computer resources. What motivates us for this work is to make an easy system that can control computer using hand gestures as the traditional ways for controlling computers nowadays aren't comfortable to users as it's expected.
Due to real-time operational requirements, we are interested in a computationally efficient algorithm. Early approaches to the hand gesture recognition problem in computer control context involved the use of markers on the finger tips.

An associated algorithm is used to detect the presence and color of the markers, through which one can identify which fingers are active in the gesture & identify the gesture based on the position of each marker. The inconvenience of placing markers on the user's hand makes this an infeasible approach in practice. Our project is based primarily on color matching and is performed in several distinct stages. The first stage is to convert the input image from RGB to HSV then we extract the markers using thresholding after color-based segmentation for markers, gestures are recognized using geometric calculations of the center of the markers.

4.1.1 Software Phases

To design an efficient Real-Time hand gesture recognition system we have to pass through three important phases(offline phase-online phase-gesture detection phase) to get out our overall system that could efficiently get a sequence of frame and according to the calculation of markers position a gesture can be identified.

Real-time hand gesture recognition system designing is a very challenging step where you face several problems you have to solve. designing step is the most important step in any system according to it the quality of your system could be identified how fast your system is? What are the conditions that your system works probably in to get the expected output? What are the constraints on your system? How to implement our system to get the best performance we can from it?
As in real-time systems the most important aspect that the system should have is to be a fast system to get the expected output at the expected time.

All this questions were putted in consideration when we were in the step of designing the system offline phase is the most important phase we passed through in the designing as in this phase we decide how our algorithm will look like and what are the steps that should be taken to get expected output as said before with putting on consideration all the above questions that need to be answered.

Fig 4.2 software phases

4.1.1.1 Offline phase(designing step)

In any real-time system to get the expected result you have to pass through offline phase where you have to design your Algorithm and try it on a single frame and modify it until you reach to the expected output you want from your system. Offline phase is considered as the most important step in our system as the design of the system has been specified in this step. In this step we were have to design and built a system that could have an image as an input and then get the position of each marker as an output with the best performance we can get. Offline phase can be divided to three steps.

Fig 4.3 Offline phase steps

Input frame

As in a real-time hand gesture recognition system the algorithm will be run on a sequence of frames and according to the markers position output of each frame the gesture can be recognized. We will design the algorithm that will work only on one frame "one image". As we are using Open CV library so we will use open cv functions to load image.

Convert image from RGB to HSV

- **Color space models:**

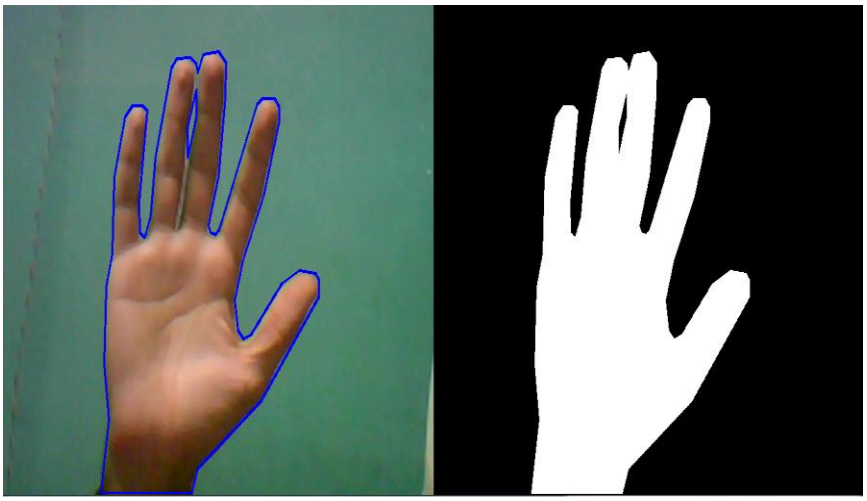
At this point, it is important to consider how colored images are represented and stored. Colored images are represented using a color space, or color model, which is defined as a "specification of a 3D color coordinate system and a visible subset in the coordinate system within which all colors in a particular color gamut lie" Each color space is optimized for a different application, and the same image may be represented using multiple color spaces.

- **Binary**

A binary image is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white though any two colors can be used. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color. In the document scanning industry this is often referred to as bi-tonal.

Binary images are also called *bi-level* or *two-level*. This means that each pixel is stored as a single bit (0 or 1). The names *black-and-white*, *B&W*, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images. In Photoshop parlance, a binary image is the same as an image in "Bitmap" mode.

Binary images often arise in digital image processing as masks or as the result of certain operations such as segmentation, thresholding , and dithering. Some Input/output devices, such as laser printers, fax machines, and bi-level computer displays. The figure below shows the difference between an original (RGB) image and a binary image.



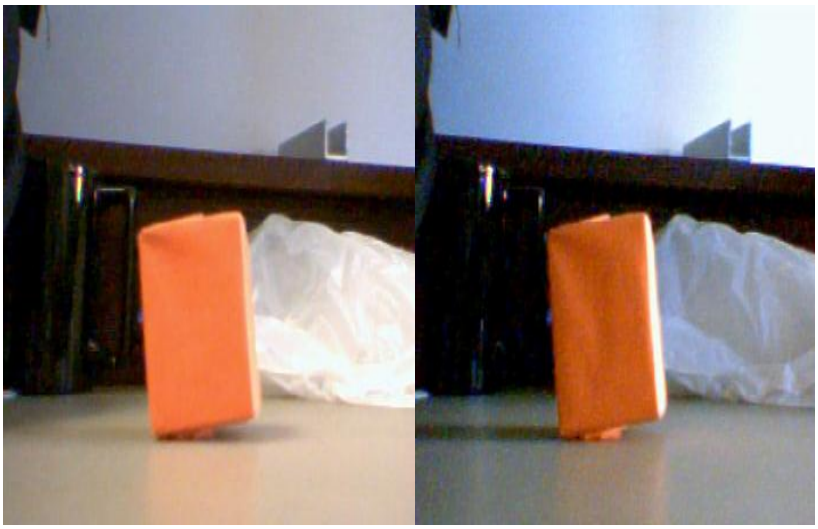
a)Original image b)Binary image

Fig 4.4 difference between colored image and a binary image

- **RGB**

RGB (Red, Green, Blue) is an additive model in which red, green and blue light is combined in various amounts to create other colors .One of the most common applications of RGB is the display of colored images on CRT or LCD screens. In addition, most digital cameras, including the ones used on the EyeBot M6, output data in RGB format. This representation is thus important for the input and output of colored images.

There are, however, several problems with using the RGB color space for image processing. The main problem lies in the color space's close coupling between color and brightness. Changing the values of any one RGB channel not only changes the color of a pixel, but also its brightness. This means that the same colored object will have vastly different RGB channel values in different lighting conditions. This is illustrated in Figure (),where the same orange box is show under various lighting conditions. As can be seen, the RGB channel values of the box differ greatly as the lighting environment changes.



a)original image RGB=(247,121,78)

b) Uneven Lighting RGB=(186,87,28)

c) Dark Lighting RGB= (102,30,5)

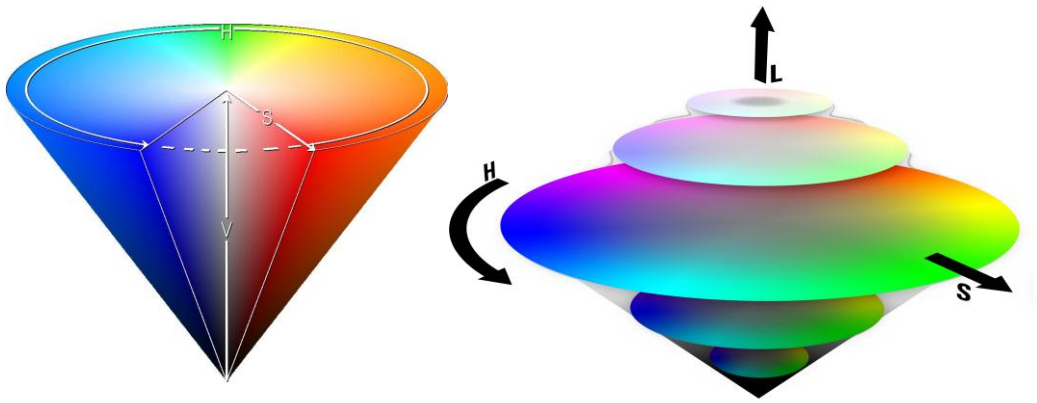
d) Very Dark Lighting RGB=(47,2,3)

Fig 4.5 examples of varying lighting conditions

When using RGB color space for image processing, it is usual to first normalize the channel values in some fashion to decouple the image brightness from the color information. An alternative technique is to convert the RGB data into a different color space, then use the second color space for image processing.

- **HSL/HSV:**

The HSL (Hue, Saturation, Luminance) and HSV or HSI (Hue, Saturation, Value or Intensity) color spaces solve many of the problems associated with RGB. Both models define a colored pixel in terms of three components [27]. Hue is the 'type' of color (green, yellow, etc.), saturation is the 'vibrancy' of the color, and the value or luminance determines how bright a color appears. This is important for two reasons. Firstly, unlike RGB, HSV and HSL separate color from brightness. Thus it is possible to locate, say, an orange box, independent say, and an orange box, independent of local lighting conditions. Secondly the important color information is located in a single channel (Hue). Both these factors make HSV/HSL very good color models for performing image processing. Although the two color spaces are very similar, HSL produces a more 'natural' looking grayscale channel (Luminance) than HSV value. Under HSL, the saturation component ranges from full color to an equivalent grey value, whereas in HSV, it ranges from full color to white. In HSL, the luminance component ranges from black, through the chosen color, to white, whereas in HSV, the value component ranges only halfway, from black to the chosen color. The hue component is the same for both color spaces. This is illustrated in Figure 4.6, which shows the visualization of both color spaces. The conversion from RGB to HSL color space is non-linear, given mathematically by the following equations. Assuming the (Red, Green, Blue) channel values have been normalized to lie between 0.0 and 1.0, let MAX equal the maximum of the (Red, Green, Blue) values and MIN equal the minimum of these values.



(a) HSV Color Space (b) HSL Color Space

Fig 4.6 Visualization of the HSV and HSL color spaces

$$\text{Hue} = \begin{cases} \text{undefined} & \text{if MAX} = \text{MIN} \\ 60^\circ \times \frac{\text{Green} - \text{Blue}}{\text{MAX} - \text{MIN}} + 0^\circ & \text{if MAX} = \text{Red and Green} \geq \text{Blue} \\ 60^\circ \times \frac{\text{Green} - \text{Blue}}{\text{MAX} - \text{MIN}} + 360^\circ & \text{if MAX} = \text{Red and Green} < \text{Blue} \\ 60^\circ \times \frac{\text{Blue} - \text{Red}}{\text{MAX} - \text{MIN}} + 120^\circ & \text{if MAX} = \text{Green} \\ 60^\circ \times \frac{\text{Red} - \text{Green}}{\text{MAX} - \text{MIN}} + 240^\circ & \text{if MAX} = \text{Blue} \end{cases}$$

$$\text{Saturation} = \begin{cases} 0 & \text{if MAX} = \text{MIN} \\ \frac{\text{MAX} - \text{MIN}}{\text{MAX} + \text{MIN}} = \frac{\text{MAX} - \text{MIN}}{2 \times \text{Lumience}} & \text{if } 0 < \text{Lumience} \leq \frac{1}{2} \\ \frac{\text{MAX} - \text{MIN}}{2 - (\text{MAX} + \text{MIN})} = \frac{\text{MAX} - \text{MIN}}{2 - 2\text{Lumience}} & \text{if Lumience} > \frac{1}{2} \end{cases}$$

$$\text{Lumience} = \frac{1}{2}(\text{MAX} + \text{MIN})$$

Fig 4.7 HSL equations

This gives $0^\circ \leq \text{Hue} \leq 360^\circ$, $0 \leq \text{Saturation} \leq 1$, and $0 \leq \text{Lumience} \leq 1$. Note that the condition of MAX = MIN signifies that the pixel corresponds to a grayscale value, and hence the Hue channel has no meaning. This corresponds to points lying along the central axis of the HSL colour cone.

If, as is more often the case, the (Red, Green, Blue) channel values are expressed as 8-bit quantities, these equations become:

$$\text{Hue} = \begin{cases} 255 & \text{if MAX} = \text{MIN} \\ 42 \times \frac{\text{Green - Blue}}{\text{MAX - MIN}} + 42 & \text{if MAX} = \text{Red} \\ 42 \times \frac{\text{Blue - Red}}{\text{MAX - MIN}} + 126 & \text{if MAX} = \text{Green} \\ 42 \times \frac{\text{Red - Green}}{\text{MAX - MIN}} + 210 & \text{if MAX} = \text{Blue} \end{cases}$$

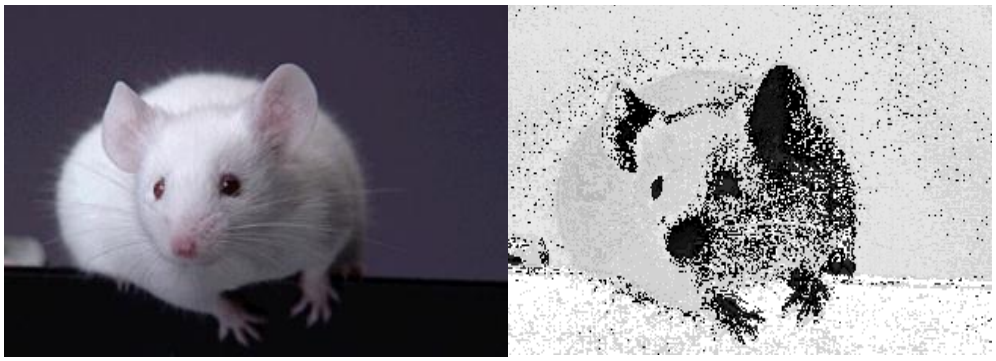
$$\text{Saturation} = \begin{cases} 0 & \text{if MAX} = \text{MIN} \\ 127 \times \frac{\text{MAX - MIN}}{\text{Lumience}} & \text{if } 0 < \text{Lumience} \leq 127 \\ 127 \times \frac{\text{MAX - MIN}}{255 - \text{Lumience}} & \text{if Lumience} > 127 \end{cases}$$

$$\text{Lumience} = \frac{1}{2}(\text{MAX} + \text{MIN})$$

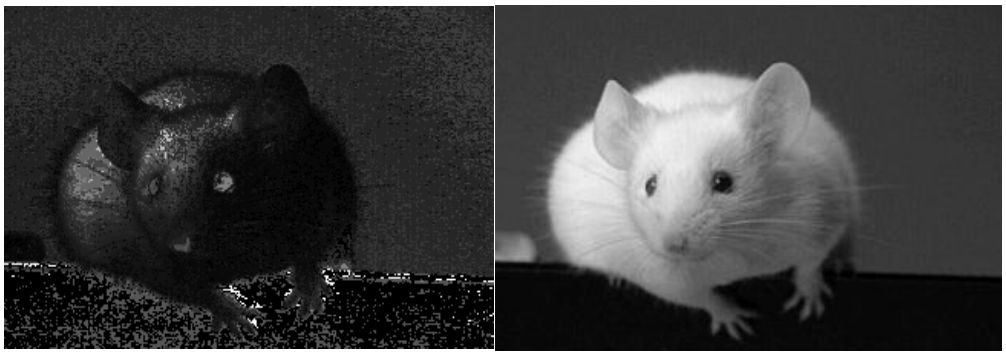
Fig 4.8 HSL values if R,G,B expressed as 8-bit quantities

This gives $0 \leq$ Hue ≤ 252, $0 \leq$ Saturation ≤ 255, and $0 \leq$ Lumience ≤ 255, with Hue $= 255$ signifying a grayscale pixel. In addition to being scaled, the Hue values have been rotated by 60° in order to reduce the number of case statements.

Figure 4.9 illustrates the decomposition of a RGB image into separate Hue, Saturation and Luminance channels. It can be seen that areas of similar color on the original image correspond to similar Hue values, vibrant areas of the image correspond to larger Saturation values, and the Luminance channel is essentially a grayscale version of the original image.



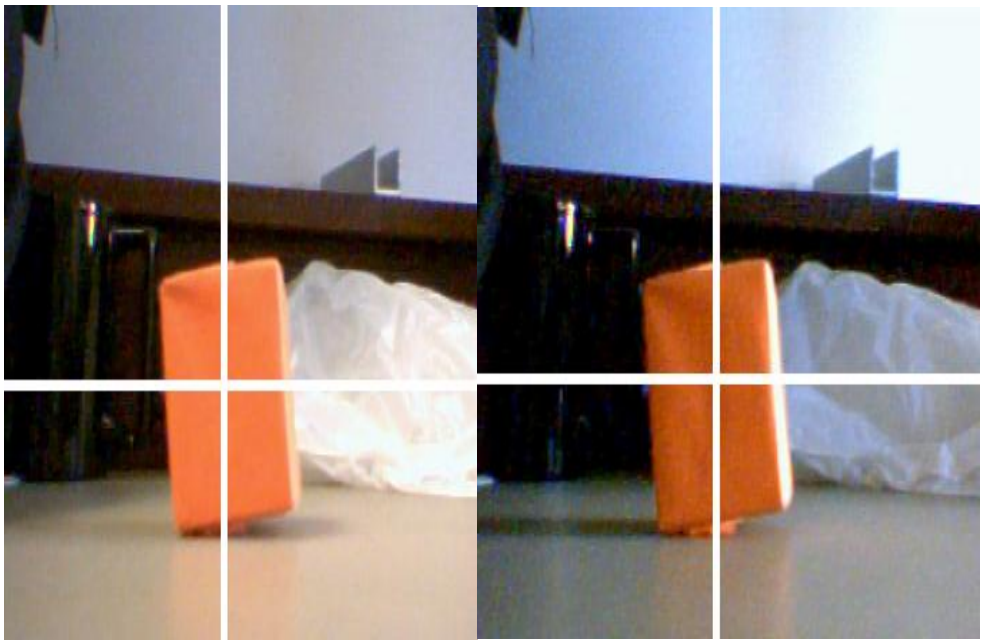
a)original image b)Hue



c)Saturation d)luminance

Fig 4.9 decomposition of a RGB image into separate HSL values

Fig 4.10 shows the same images from Fig 4.11, but the pixels are now analyzed using the HSL color space. As can be seen from the images, the Hue component changes only slightly under varying lighting conditions. The white lines indicate the results of running the object location algorithm over the images. It can be seen that using the Hue channel allows the algorithm to locate the orange box (target Hue value = 54 ± a tolerance of 5) under all but the darkest lighting conditions.



a)original image b)Uneven Lighting

HSL=(52,231,162) HSL=(57,188,107)

Fig 4.10

C) Dark lightening

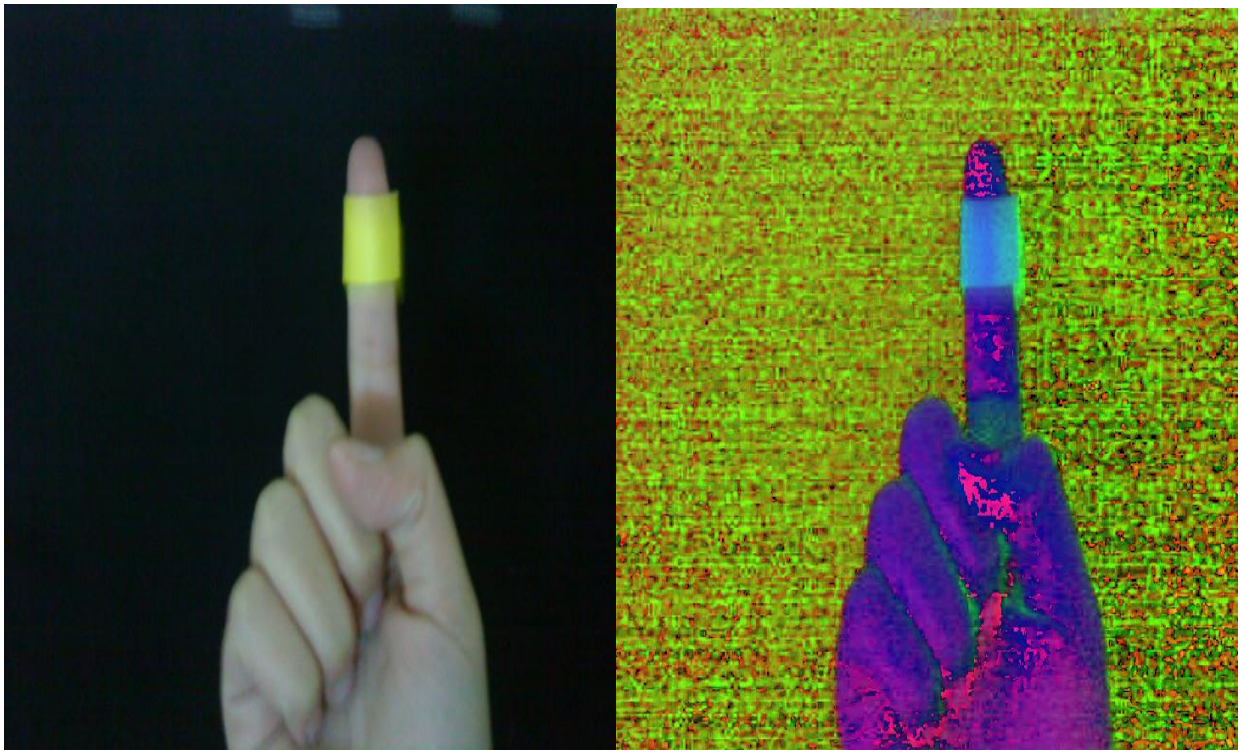
HSL=(52,233,53)

d) Darker lightening

HSL=(41,239,24)

Fig 4.11

According to the previous comparison between RGB & HSV color space models we have found that the Hue component changes only slightly under varying lighting Conditions & on the other hand the main problem in RGB color space lies in the color space's close coupling between color and brightness. And as our project is based primarily on color matching so choosing the color space we are going to deal with is an important phase and as discussed previously HSV color space model is less sensitive to the lightness environment so we decided that the first step we should take after getting a frame is to convert it from RGB to HSV color space.



RGB image HSV image

Fig 4.12 Converting from RGB to HSV

Extract markers from image

After converting the image from RGB to HSV the first problem we thought about is how to extract the markers from the image at the first we have tried to use thresholding to extract the marker from the image by specifying the HSV range of each marker that is used, but there was a problem that is appeared after tresholding process that there was a lot of noise appearing beside the extracted marker due to the existence of things in the same range and the neutral noise that could appear after any thresholding process. After a lot of experimental trials to extract this noise from the output image using several types of filters.(gradient filter, averaging filter) we reached to the best operation we could perform after thresholding and get the best output performance which is marker contour extraction**.**

So the step of extracting markers from the image consists of two vital steps:

i) Thresholding.
ii) Marker contour extraction.

as illustrated in the following block diagram .

Fig 4.13 extracting marker process

i) Thresholding :

- A brief description about thresholding:

Thresholding is the simplest method of image segmentation . *Segmentation* is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. During the thresholding process, individual pixels in an image are marked as "object" pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as "background" pixels otherwise. This convention is known as *threshold above*. Variants include *threshold below*, which is opposite of threshold above; *threshold inside*, where a pixel is labeled "object" if its value is between two thresholds; and *threshold outside*, which is the opposite of threshold inside. The image is segmented into object and background pixels as described above, creating two sets:

G_1= {f(m,n):f(m,n)>T} (object pixels)
G_2= {f(m,n):f(m,n)$\leq$T} (background pixels)

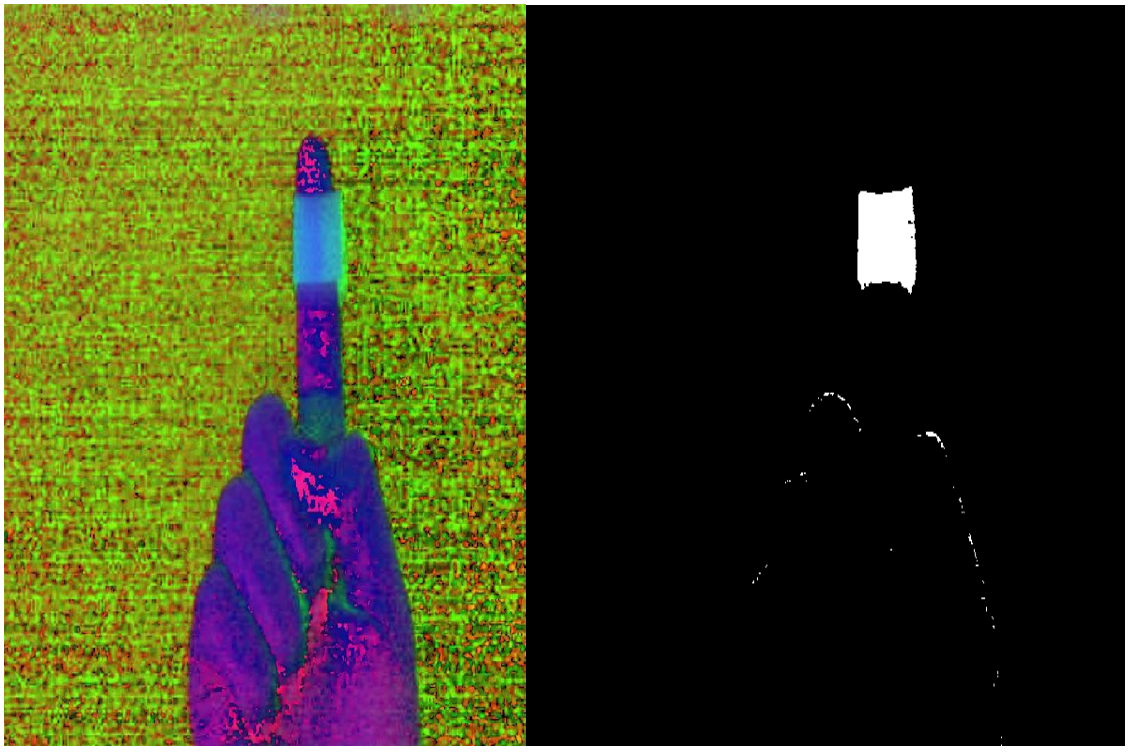
(note : f(m,n) is the value of the pixel located in the m^{th} column, n^{th} row).
Thresholding step is a very important step in extracting the marker from theimage

We have performed our thresholding step over a colored image after converting it to HSV and by using the ranges of the marker color we are using (Red-yellow-green) And we have gotten out a binary image that include the marker at its position put as discussed earlier there might be noise in the output image due to the existence of an object in the same color range or noise happened due to the thresholding step.
We have performed this step using open-CV function as will be illustrated later.

- Open CV marker HSV color ranges:

Color	Range from	To
RED	(0, 100, 100,0)	(10, 255, 255)
Yellow	(35, 40, 50)	(65, 300, 300)
Green	(75, 100, 100)	(120, 255, 255)
Orange	(120, 100, 100)	(150, 260, 290)

The following figure will illustrates the output of the thresholding process to an HSV image which includes a yellow color marker



a)HSV image b)thresholded image

Fig 4.14 displays the output of the thresholding process over an HSV image

as shown in the figure the output contains some kide of noise due to the thresholding process & that what will make us move to the next step to extract markers only from image without noise.(Marker contour extraction step) .

ii) Marker Contour Extraction :

We have two ways to perform this step:

1) Connected component algorithm:

Connected-component labeling (alternatively connected-component analysis, blob extraction, region labeling, blob discovery, or region extraction) is an algorithmic application of graph theory, where subsets of connected components are uniquely *labeled* based on a given heuristic. Connected-component labeling is not to be confused with segmentation.

Connected-component labeling is used in computer vision to detect connected regions in binary digital images, although color images and data with higher-dimensionality can also be processed. When integrated into an image recognition system or human-computer interaction interface, connected component labeling can operate on a variety of information. Blob extraction is generally performed on the resulting binary image from a thresholding step. Blobs may be counted, filtered, and tracked.

A graph, containing vertices and connecting edges, is constructed from relevant input data. The vertices contain information required by the comparison heuristic, while the edges indicate connected 'neighbors'. An algorithm traverses the graph, labeling the vertices based on the connectivity and relative values of their neighbors. Connectivity is determined by the medium; image graphs, for example, can be 4-connected or 8-connected. Following the labeling stage, the graph may be partitioned into subsets, after which the original information can be recovered and processed .

The following figures expresses the experimental results of using connected component algorithm & extracting the biggest connected component area from the image.
But it had some constrains when we performed that experiment .that to get the best result the background should be a black background and that the area of the hand must be the biggest area in the image.to can extract it.

a)original image b)hand extracted image

Fig 4.15 The result of applying connected component algorithm

2) Contour Extraction:

it's another way to extract an object from an image but the difference between this one & connected component one that in contour extraction extract solid areas if there is a hole inside the object the object will be extracted solid & as we extract a marker which is in nature a solid object that's why if we used contour extraction there will not be loss in data .thats why we decided to use contour extraction as it saves time more than connected component labeling as it don't care for the holes in the objects and it is suitable for our task.The following figure will show us the output from marker contour extraction process.

a) Image after thresholding b) image after contour extraction

Fig 4.16 illustrates the output process of marker contour extraction.

As displayed we were successfully able to extract the marker from the image without any noise.

4.1.1.2 Online phase

As discussed earlier to get our overall system we will have to pass through several steps the first step was the offline phase and according to that phase the design of our system was specified .& it was explained in details in the previous pages now we are going to talk about the next step which is the online phase.
It is a very simple step which we want to try our system that was reached in the previous phase but the difference that we will try it over a stream of frames (video) and see how the performance of the output & this will be performed using the laptop camera & the difference of the code will be that instead of operating the system over a single frame it will be operated over a stream of frames & that could be performed by using a while loop that will loop over the offline system while there is an input frame from the cam.

4.1.1.3 Gesture Detection Phase

After extracting the markers from the image the next phase we were concerned with is how to detect a certain gesture made by the markers.
To make the system recognize a gesture we needed to pass through three steps Get the marker center, identify the marker color and track the position of the marker center.

The following diagram shows the process of gesture recognition.

Figure 4.17

4.1.1.3.1 Gesture Recognition Steps:

First of all in our system we have three gestures the system should be able to recognize.

- Mouse gesture.
- Left click gesture.
- Right click gesture.

So to recognize these two gestures we have to pass through the following three steps.

1) **Get the marker center**.

One of the most important steps to recognize hand gesture is to get the marker center to be able to track it's position.

to calculate the center of the marker we have used Green's theorem.

Green's theorem to compute the surface area, volume, center of mass, and moments of inertia of various objects.

We have calculated the moment of inertia of the marker (moment_x , moment_y) and the area of the marker & by dividing them we could be able to get the center of the marker.

center_x = moment_x/area

center_y=moment_y/area

After calculating the marker center the next step we have to pass through is to identify the color of the marker that exists.

2) **identify the marker color**

to make it easy to recognize the gesture we have to make the system to identify the marker color so that a gesture of the above two gestures could be recognized easily as if the system found out that the color of the marker is pink then the system will get indication that the gesture is a mouse gesture.

And if the system found out that the color of the marker is green then the system will get indication that the gesture is left click gesture.

3) **Track the marker center**

The last step in gesture recognition process is to track the center of the marker to know where the mouse points at or where the left click is done.

- a brief description about hand gesture recognition tool

4.1.1.3.2 Gesture Recognizer

Idea is to user will wear 2 markers with certain colors (,pink , green), user move his hand and certain function is proceed .

- Functions proceed depend on the gesture (shape and movement) .

Algorithm

- First we should detect the 2 markers in each frame (tracking).
- Calculate the center of each marker.
- Track each center then imagine the variance of each color (e.g. center of pink changed from (200,200) to (200,250) this mean finger carry pink marker moved UP)
- Combine the four movements and compare it with the template gestures.
- If matching success function related to this gesture is to be called .
- Preparing to next frame by Saving the current coordinates as last coordinates .

Problems

1-In the project we deal with only one camera so by this algorithm so if picture find two yellow color one of them the marker color ,algorithm may take the wrong color

Solution

We can make a constraint that we haven't any other colors like marker colors (pink, green) in the background of the picture.

Function (get_marker_center)

This function is implemented to detect certain color then get the center (x,y) position and area of it .

```
int minArea = 170 ;// 0.1*total area = 0.1*640*480=30720
int maxArea = 30720 ;//  0.0005*total area = 0.0005*640*480=170
int counterCAM = 0 ; /*this counter for the static gesture due to not to be detected more than one timestart 0 end 10 frames .*/
int countpink=0;
int countyellow=0;
int countgreen=0;
int countorange=0;
int countclose=0;
int y_pink=0;
int y_green=0;
int y_yellow=0;
int y_orange=0;
int flag=0;

```

Figure 4.18

Here we decelerate global variable for using in both main and function .

minarea , maxArea :: initially equal the minimum and maximum values of area which we search for i.e. detected color = minarea < element > maxArea .

MAIN FUNCTION

```
// Initialize capturing live feed from the camera
CvCapture* capture = 0;
capture = cvCaptureFromCAM(0);

// Couldn't get a device? Throw an error and quit
if(!capture)
{
    printf("Could not initialize capturing...\n");
    return -1;
}
```

Figure 4.19

Initialization the camera and ensure that it's ready to work

```
int max = 0;
IplImage* img_IN = 0;
img_IN = cvQueryFrame(capture);
for(int prepararionCounter = 0 ; prepararionCounter<20 ; prepararionCounter++ )
{
    cout<<minArea<<"           "<<maxArea<<endl;
    cout<<pink_color.area<<endl;
    img_IN = cvQueryFrame(capture);
    preparation(img_IN);
    prepararionCounter ++ ;
    int temp = pink_color.area;
    if(max<temp)
        max=temp;
}
maxArea=max+500;
minArea=max-800;
```

Figure 4.20

Detecting colors and Fetch Area and Positions of each color

Each time we change y (marker_color enum) and call function get_marker_center then print it on the screen.
This part of code divided into 4 small parts ,all of this parts are equal .So we can utilize this by multithreading (OPENMP) in S.W or parallelization in H.W .

```
getState(yellow_color,pink_color,orange_color,green_color,state);
int status  =   checkGesture(state);
```

Figure 4.21

getState

method that return how many sticker in the frame .

checkGesture

method that return a number is represent gesture using the data returned from getState function .
this number returned is refer that it may be a gesture or not .so it must be a part of ensuring that is a gesture and if yes call the gesture function .
According to the Status Value Function calling is decided .

```
switch ( status )
{
case 1 :
     {
          mouseFn();
              break    ;
      }
case 2 :
     {
          leftClickFn();
          break    ;
     }
case 3 :
     {
          rightClickFn();
          break    ;
     }
case 4 :
     {
          closeFn();
          break    ;
```

Figure 4.22

```
IplImage* get_marker_center( IplImage* img_IN ,enum marker_colour x,struct Coordinates *xCor )
{
    char const * temp;
    IplImage* img_HSV = cvCreateImage(  cvGetSize(img_IN), 8, 3);// points to the HSV image
    IplImage* img_after_thresholding = cvCreateImage( cvGetSize(img_IN), 8, 1);//points to the thresholded image
    IplImage* img_OUT = cvCreateImage(cvGetSize(img_IN) , 8, 1);//points to  the output image

    CvMemStorage* storage = cvCreateMemStorage(0);//for findcontour function
    CvSeq* first_contour = 0;//for findcontour function its a pointer to a linked list

    cvCvtColor(img_IN, img_HSV, CV_BGR2HSV);

    switch(x)
    {

        case yellow : {
                        cvInRangeS(img_HSV, cvScalar(35, 75, 75), cvScalar(53, 250, 250), img_after_thresholding);//this range of yellow color
                        break;
                    }
```

Figure 4.23

Get_marker_center function Implementation

get_marker_center
this method as explained before used to detect color then get area and center of color (x , y) position .

Implementation brief

We must detect yellow color range in HSV mode to search for it in instruction (cvInRanges) cvInranges function return a binary image have all shapes detected have a green colors. So we have to take on of them which have the specification of the sticker (area). So contour will help us to face this problem.

In OpenCv libraries there is a function call (cvFindContour) it's output is linked list of each area found in the binary image and we select the one have the best one .

```
//linked list of each contor the return content//,sizeof(CvContour),CV_RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE,cvPoint(0, 0));
  cvFindContours(img_after_thresholding,storage,&first_contour);
  cvZero( img_OUT);

   //  CV_RETR_CCOMP : changed to --> CV_RETR_EXTERNAL in order to "retriving only the extreme outer contours"
  // because CV_RETR_CCOMP retrieves also the hole which isn't wanted in our case.
 // cvPoint(0, 0) : added because we use the full image not ROI so we don't need any offset or shifting.
   double area  =   0   ;
   int tot_area = img_IN->height*img_IN->width; // 3shan lw area bta3et el sora bete5telef//
   static int Center_X = 0;
   static int Center_Y = 0;
```

Figure 4.24

```
while (cvContourArea(first_contour,CV_WHOLE_SEQ) >(maxArea)|| cvContourArea(first_contour,CV_WHOLE_SEQ)<minArea)
{
        if(first_contour->h_next)
             first_contour = first_contour->h_next;
        else
            break ;
    }
```

Figure 4.25

Searching in the Linked List

```
if(!(cvContourArea(first_contour,CV_WHOLE_SEQ) >(maxArea) || cvContourArea(first_contour,CV_WHOLE_SEQ)<minArea))
{
        CvScalar color = CV_RGB( rand(), rand(), rand() );
        cvDrawContours( img_OUT, first_contour, color, color, -1, CV_FILLED, 8 );//this one will draw the contour i have found
```

Figure 4.26

```
//////////////////////////////////////////////Calculating Center//////////////////////////////////////////////////////////
     CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
    cvMoments(img_OUT, moments, 1);

// The actual moment values
    double moment10 = cvGetSpatialMoment(moments, 1, 0);
    double moment01 = cvGetSpatialMoment(moments, 0, 1);
    area = cvGetCentralMoment(moments, 0, 0);
    Center_X = (int)moment10/area;
    Center_Y = (int)moment01/area;
    cvSet2D(img_OUT,Center_Y,Center_X,cvScalar(0,0,0));
    temp = "Final_Output"+ char(x+48);
    xCor->x     =   Center_X    ;
    xCor->y     =   Center_Y    ;
    xCor->area  =   (int)cvContourArea(first_contour,CV_WHOLE_SEQ)  ;
```

Figure 4.27

Calculating Center of the area using spatial moment theorem

In this part of the code we try to find the center of white area in black image doesn't contain any other white area .

Gesture Detection Algorithm

Mouse

- Mouse is a dynamic gesture .
- There are different cases of mouse .
- This gesture is easy to detect because it depend on finding certain colors and not finding other colors .

Mouse cursor

it work just after pink sticker found a lone in the frame .
Pink sticker work as cursor of the mouse. Function mouse print the (x , y) position as pink sticker move as center changes position changes .

Lift Click

If another color detected with pink we see it if (green) only it mean (Lift Click) function it look like mouse cursor in its function because it print position of pink color .

Right Click

right click gesture detected when green color detected individually and no other color found .
this function return the position of center of green color .

4.2 Porting on Nios II

4.2.1 Embedded Systems Programming:

Embedded systems programming is different from developing applications on a desktop computers. Key characteristics of an embedded system, when compared to PCs, are as follows:

- Embedded devices have resource constraints (limited ROM, limited RAM, limited stack space, less processing power)
- Components used in embedded system and PCs are different; embedded systems typically uses smaller, less power consuming components.
- Embedded systems are more tied to the hardware.

Two salient features of Embedded Programming are code speed and code size. Code speed is governed by the processing power, timing constraints, whereas code size is governed by available program memory and use of programming language. Goal of embedded system programming is to get maximum features in minimum space and minimum time.

Embedded systems are programmed using different type of languages:

- Machine Code.
- Low level language, i.e., assembly.
- High level language like C, C++, Java, Ada, etc.
- Application level language like Visual Basic, scripts, Access, etc.

Assembly language maps mnemonic words with the binary machine codes that the processor uses to code the instructions. Assembly language seems to be an obvious choice for programming embedded devices. However, use of assembly language is restricted to developing efficient codes in terms of size and speed. Also, assembly codes lead to higher software development costs and code portability is not there. Developing small codes are not much of a problem, but large programs/projects become increasingly difficult to manage in assembly language. Finding good assembly programmers has also become difficult nowadays. Hence high level languages are preferred for embedded systems programming.

Use of C in embedded systems is driven by following advantages:

- It is small and reasonably simpler to learn, understand, program and debug.
- C Compilers are available for almost all embedded devices in use today, and there is a large pool of experienced C programmers.
- Unlike assembly, C has advantage of processor-independence and is not specific to any particular microprocessor/ microcontroller or any system. This makes it convenient for a user to develop programs that can run on most of the systems.
- As C combines functionality of assembly language and features of high level languages, C is treated as a 'middle-level computer language' or 'high level assembly language'
- It is fairly efficient
- It supports access to I/O and provides ease of management of large embedded projects.

Difference between C and Embedded C

- C is for desktop computers, embedded C usually is for microcontroller based applications.
- C use the resource of the desktop computers (memory, OS, etc.).
- Embedded C use only limited resources available in chip (limited RAM, ROM, Ports, etc.).
- Embedded C could be a subset of C.

4.2.2 Implementation

- Convert code from a windows platform to Nios II platform or embedded platform which made by extracting all features and instructions in code.
- Output expected from this part is an embedded C++ code which is applicable on NIOS II processor.
- OpenCv code In windows platform contain a GUI instructions to show image, to control image using a slider, windows error messages and so on .
- In embedded platform code must not contain GUI instructions or any files specified to windows.

- OpenCv on Visual studio give GUI capabilities by include "highgui" core.

GUI instructions Example:

- **cvNamedWindow("pic") ;**

- **cvShowImage("pic",img_IN) ;**

So porting task is to extract all windows platform instructions which doesn't work in embedded platform, then generate an embedded C code valid for execution on Nios II's IDE (Eclipse).

4.2.2.1 Content

OpenCV consist of 4 main cores:

A) CV
Component contains the basic image processing and higher-level computer vision algorithms.

B) ML
Is the machine learning library, which includes many statistical classifiers and clustering tools

C) HighGUI
Contains I/O routines and functions for storing and loading video and images.

D) CXCORE
Contains the basic data structures and content.

So we find that

- CV and CXCORE is the used cores .
- ML is doesn't used in our algorithm (can be neglected) .
- HIGHGUI is the core must be extracted .

4.2.2.2 Steps for porting

a. Searching the code and delete any GUI instructions.
b. Create empty project.
c. Ensure that the project's environment is not linked opencv path.
d. Add your main file.
e. Exclude the files that are not used by the algorithm which is done by tracing every function in the algorithm to know which files are needed and which are not.
f. Include all opencv files (.c, .cpp, .h, .hpp) in this project.
g. Build project.

INFACT step 5 is difficult step because opencv hierarchy is very complicated and it's hard to get the complete implementation of any function.
Linking error is a famous error in porting process because you may put non completed files so when the linker search for implementation of a certain function it may not found because the file contain function is not found .

4.2.3 Probable errors

Error LNK2019:
unresolved external symbol _cvReleaseGLCM referenced in function "void __cdecl icvCreateGLCM_LookupTable_8u_C1R(unsigned char const *,int,struct CvSize,struct CvGLCM *,int *,int,int *)" (?icvCreateGLCM_LookupTable_8u_C1R@@YAXPBEHUCvSize@@PAUCvGLCM@@PAHH3@Z)
1>"Project Path"\"filename".exe: **fatal error LNK1120:** 1 unresolved externals

Error LNK2019: unresolved external symbol "class cv::Ptr<class cv::BaseRowFilter> __cdecl cv::getRowSumFilter(int,int,int,int)" (?getRowSumFilter@cv@@YA?AV?$Ptr@VBaseRowFilter@cv@@@@1@HHHH@Z) referenced in function "class cv::Ptr<class cv::FilterEngine> __cdecl cv::createBoxFilter(int,int,class cv::Size_<int>,class cv::Point_<int>,bool,int)" (?createBoxFilter@cv@@YA?AV?$Ptr@VFilterEngine@cv@@@@1@HHV?$Size_@H@1@V?$Point_@H@1@_NH@Z)
1>>"Project Path"\"filename".exe: **fatal error LNK1120:** 1 unresolved externals

Error LNK2019: unresolved external symbol _cvCvtColor referenced in function "void __cdecl cv::cvtColor(class cv::Mat const &,class cv::Mat &,int,int)" (?cvtColor@cv@@YAXABVMat@1@AAV21@HH@Z)
1>newcode19-6.obj : error LNK2001: unresolved external symbol _cvCvtColor
1>>"Project Path"\"filename".exe: fatal error LNK1120: 1 unresolved externals

Error LNK2019: unresolved external symbol _cvSaveMemStoragePos referenced in function _cvStartFindContours

Error LNK2001: unresolved external symbol _cvSaveMemStoragePos
1>>"Project Path"\"filename".exe: **fatal error LNK1120:** 1 unresolved externals

Error C2065: 'icvLuv2BGRx_8u_C3CnR': undeclared identifier

All the above error almost is the same solution, linker search for an certain implementation and failed because it's not found so the solution is to get function implementation or get the file contain the function and include it in the project .

4.2.4 Testing on Nios II processor

- To test the final code we will create a new project with a SOPC building tool as shown in Fig 4.28

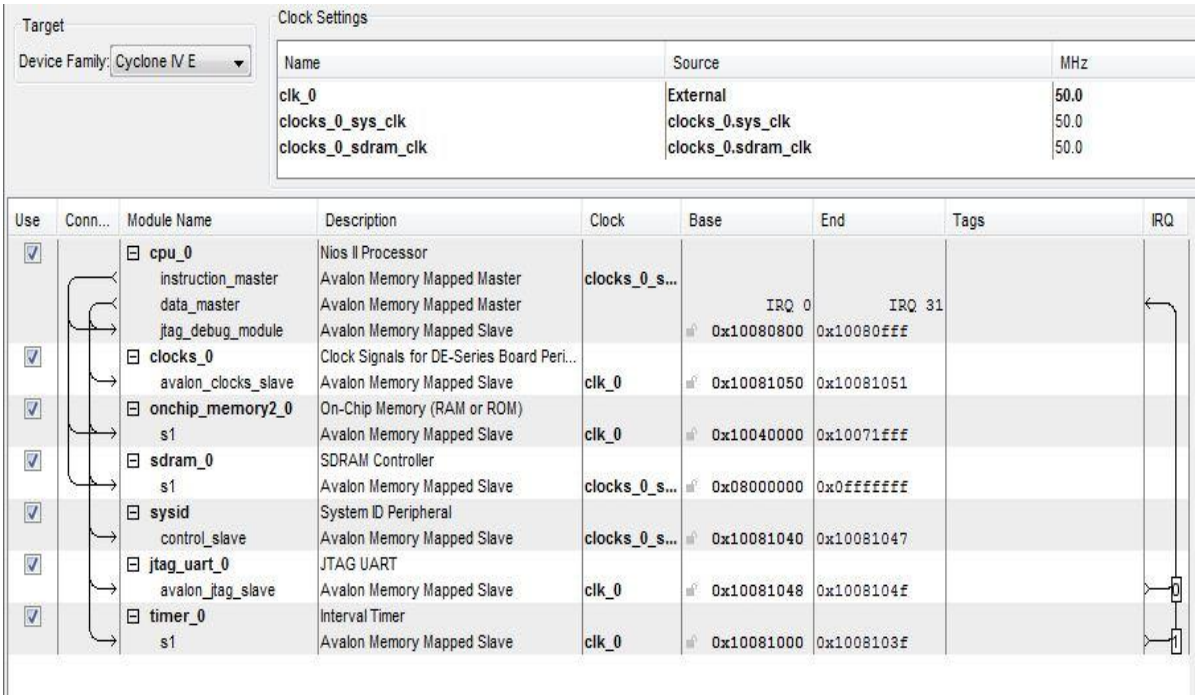


Fig 4.28 Output of SoPC

- Exception and reset vector are on the chip memory as shown in the Fig 4.29

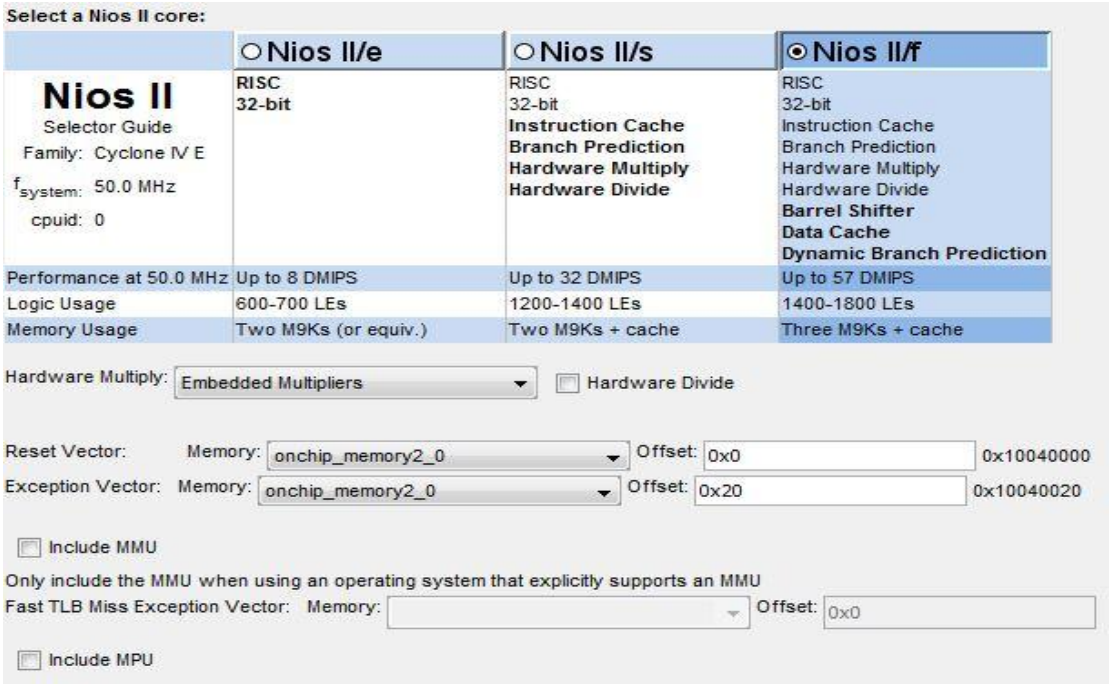


Fig 4.29

- Changing the row and column as shown in Fig 4.30.

Fig 4.30

- Use clocks from university program tab to generate the sdram clock that is different from the system clock in phase shift -3 ns.

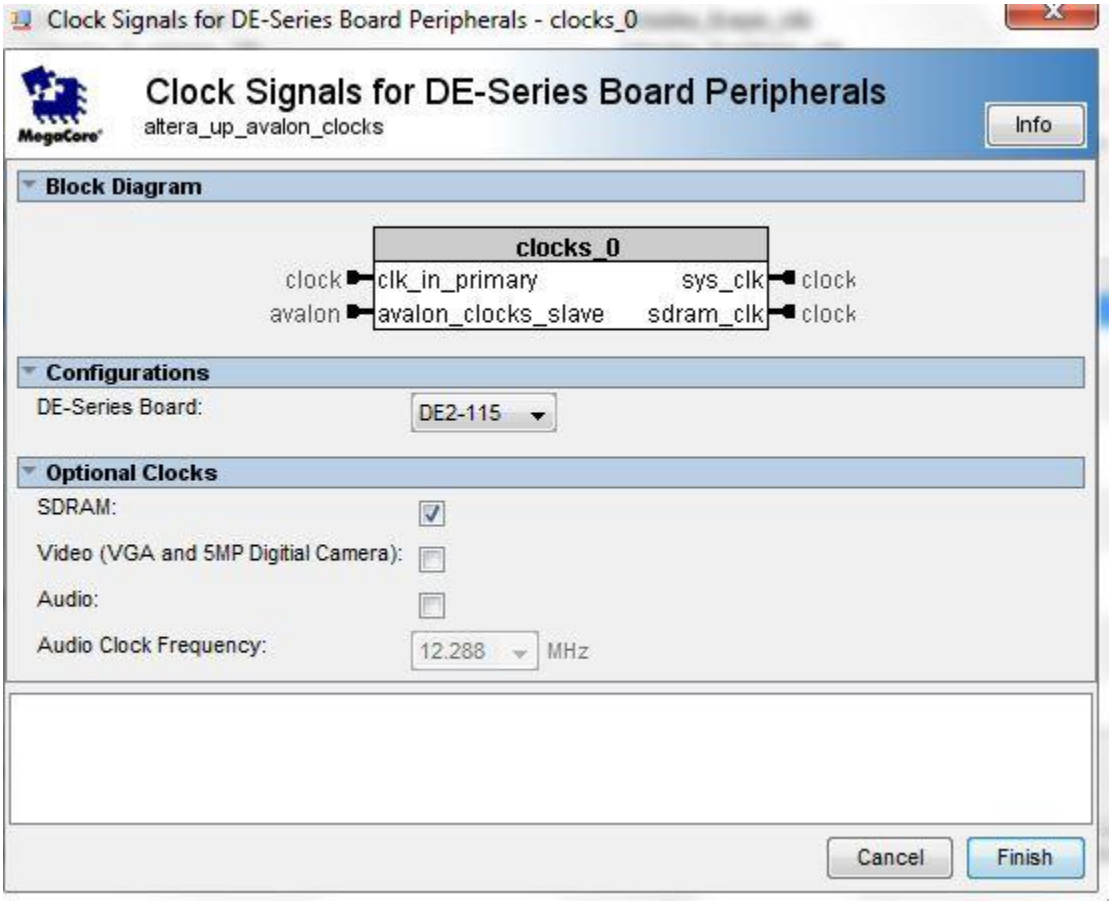


Fig 4.31

- Generate the SOPC builder then compile the project.
- Open NIOS II IDE.
- Create new project add the sopcinfo file to this project.
- Add S/W files then build the project.
- Edit the main by adjust it to get the image as a raw of data which will entered by any means.

Chapter 5

Optimization

In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the usage of memory, the usage of particular instructions, or frequency and duration of function calls. The most common use of profiling information is to aid program optimization.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler (or code profiler).

The methodology of the profiler itself classifies the profiler as event-based, as statistical, as instrumentation, or as simulation.

5.1 Instrumenting profilers:

Some profilers instrument the target program with additional instructions to collect the required information. Instrumenting the program can cause changes in the performance of the program, potentially causing inaccurate results and heisen bugs. Instrumenting will always have some impact on the program execution, typically always slowing it. However, instrumentation can be very specific and be carefully controlled to have a minimal impact. The impact on a particular program depends on the placement of instrumentation points and the mechanism used to capture the trace. Hardware support for trace capture means that on some targets, instrumentation can be on just one machine instruction. The impact of instrumentation can often be deducted (i.e. eliminated by subtraction) from the results. gprof is an example of a profiler that uses both instrumentation and sampling. Instrumentation is used to gather caller information and the actual timing values are obtained by statistical sampling.

Instrumentation

- **Manual**: Performed by the programmer, e.g. by adding instructions to explicitly calculate runtimes, simply count events or calls to measurement APIs such as the Application Response Measurement standard.
- **Automatic source level**: instrumentation added to the source code by an automatic tool according to an instrumentation policy.
- **Compiler assisted**: Example: "gcc -pg ..." for gprof, "quantify g++ ..." for Quantify .
- **Binary translation**: The tool adds instrumentation to a compiled binary. Example: ATOM .

- **Runtime instrumentation**: Directly before execution the code is instrumented. The program run is fully supervised and controlled by the tool. Examples: Pin, Valgrind .
- **Runtime injection**: More lightweight than runtime instrumentation. Code is modified at runtime to have jumps to helper functions. Example: DynInst .
-

5.2 Steps

Manual instrumentation on the code was made by adding a timer to the NIOS II system and monitoring the value of this timer before and after calling functions .
Hardware timer in SOPC Builder is added and time period is set to milliseconds then editing the software code to profile all of its sections.

Initially we got the following results:

Function	Average Time in msec
cvCreateImage()	**1**
cvCvtColor()	**8984**
cvInrange()	**270**
cvFindContour()	**250**
Looping for Marker part	**150**

Total time = 39796 msec.

T1=time of (cvCreateImage()) = 4*1 = 4 msec

T2= time of (cvCvtColor())= 4*8984 = 35396 msec.

T3= time of (cvInrange())=4*270 = 1080 msec

T4= time of (cvFindContour()) = 4*250 = 1000 msec

T5= time of (Looping for Marker part) = 4*150 = 600 msec

Frame processing time on NIOS II equals 39.8 sec which is not meeting our timing constraints.
So the following processes were held to optimize the Software as possible

Steps:

- Using only two of the 4 markers.
  - Processing time is dropped to be 20 seconds only
  - Some of the gestures are now neglected.
- Elimination of duplications (repeated calculations for some blocks)
  - ⇒ Here we called (cvcvtcolor( )) function which is common between the 2 code paths in the main function instead of calling it each time in the main .
  - ⇒ So once we get input image convert to to HSV color mode then working on it all the execution or the running time.
  - ⇒ We can exploit it by release input image (RGB color mode) immediately because it become useless.

Statistics

Function	Average Time in msec
cvCreateImage()	**1**
cvCvtColor()	**3251**
cvInrange()	**300**
cvFindContour()	**350**
Looping for Marker part	**150**

Total time =5400 sec.

T1=time of (cvCreateImage( )) = 4 milliseconds

T2= time of (cvCvtColor( )) = 13004 milliseconds

T3= time of (cvInrange( )) = 1080 milliseconds

T4= time of (cvFindContour( )) =1400 milliseconds

T5= time of (Looping for Marker detection part) =600 milliseconds

Results

- Efficiency is increased because the algorithm is now applied for 2 colors ranges only instead of four.

- Processing time of frames has tremendously decreased.

5.3 Custom Instructions Introduction:

Nios II custom instructions are custom logic blocks adjacent to the ALU in the processor's data path. Custom instructions give you the ability to tailor the Nios II Processor core to meet the needs of a particular application. You can accelerate time Critical software algorithms by converting them to custom hardware logic blocks. Because it is easy to alter the design of the FPGA-based Nios II processor, custom instructions provide an easy way to experiment with hardware-software tradeoffs at any point in the design process.

5.4 Nios II Custom instructions

when designing a system that includes an Altera Nios II embedded processor, you can accelerate time-critical software algorithms by adding custom instructions to the Nios II processor instruction set. Custom instructions allow you to reduce a complex sequence of standard instructions to a single instruction implemented in hardware.You can use this feature for a variety of applications, for example, to optimize software inner loops for digital signal processing (DSP), packet header processing, and computation-intensive applications. In SOPC Builder, the Nios II parameter editor provides a GUI to add custom instructions to the Nios II processor. In Qsys, each custom instruction is a separate component in the Qsys system. You can add as many as 256 custom instructions to your system. In SOPC Builder, the custom instruction logic connects directly to the Nios II arithmetic logic unit (ALU) as shown in Figure 5.1.

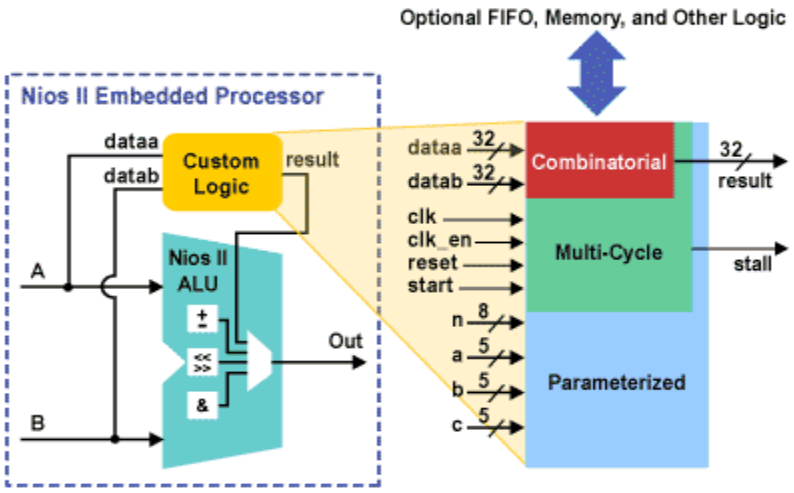


Fig 5.1 custom instruction logic

5.5 Implementing Custom Instructions

Nios II custom instruction logic receives input on its data port, or on its dataa and datab ports, and drives out the result on its result port. The custom instruction logic provides a result based on the inputs provided by the Nios II processor. The Nios II processor supports different types of custom instructions. Figure below lists the additional ports that accommodate different custom instruction types. Only the ports used for the specific custom instruction implementation are required. Figure 5,2 also shows a conduit interface to external logic.
The interface to external logic allows you to include a custom interface to system resources outside of the Nios II processor data path.

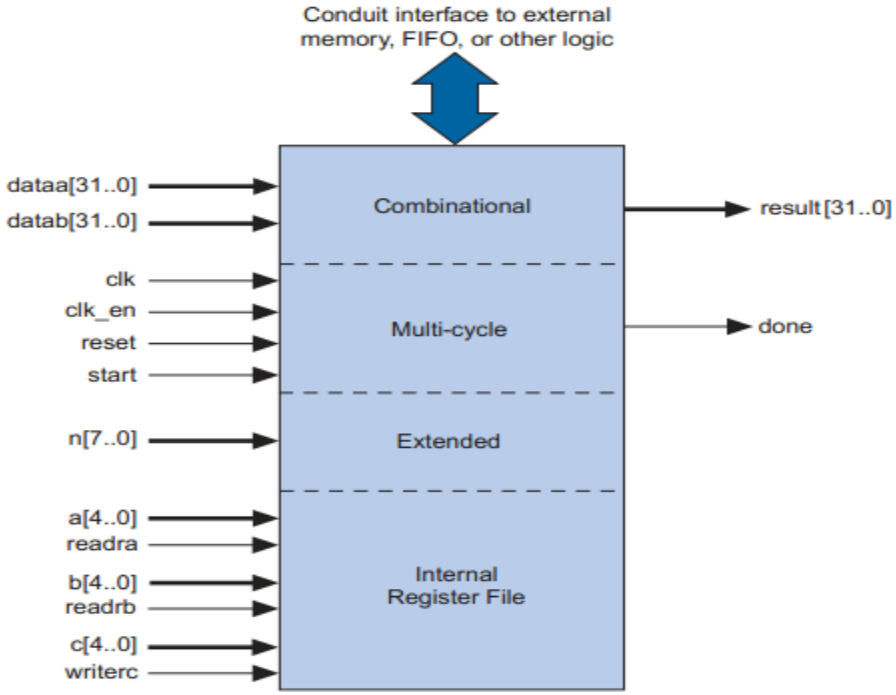


Fig 5.2 conduit interface to external logic

5.6 Custom instruction types

Type	Application	Hardware Ports
Internal register file	Custom logic blocks that access internal register files for input or output or both.	■ dataa[31:0] ■ datab[31:0] ■ result[31:0] ■ clk ■ clk_en ■ start ■ reset ■ done ■ n[7:0] ■ a[4:0] ■ readra ■ b[4:0] ■ readrb ■ c[4:0] ■ writerc
External interface	Custom logic blocks that interface to logic outside of the Nios II processor's datapath	Standard custom instruction ports, plus user-defined interface to external logic.

Fig 5.3

Type	Application	Hardware Ports
Combinational	Single clock cycle custom logic blocks.	■ dataa[31:0] ■ datab[31:0] ■ result[31:0]
Multicycle	Multi-clock cycle custom logic blocks of fixed or variable durations.	■ dataa[31:0] ■ datab[31:0] ■ result[31:0] ■ clk ■ clk_en *(1)* ■ start ■ reset ■ done
Extended	Custom logic blocks that are capable of performing multiple operations	■ dataa[31:0] ■ datab[31:0] ■ result[31:0] ■ clk ■ clk_en *(1)* ■ start ■ reset ■ done ■ n[7:0]

FIG 5.4

5.6.1 COMBINATIONAL CUSTOM INSTRUCTION :

A combinational custom instruction is a logic block that completes its logic function in a single clock cycle. Figure 5.5 shows a block diagram of a combinational custom instruction.

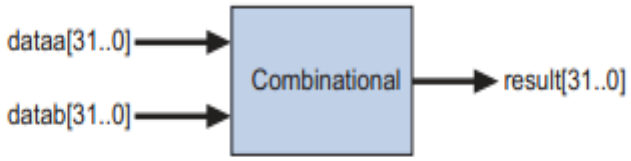


Fig 5.5 combinational custom instruction.

In the above Figure the dataa and datab ports are inputs to the logic block, which drives the results on the result port. Because the logic function completes in a single clock cycle, a combinational custom instruction does not require control ports.

Port Name	Direction	Required	Description
dataa[31:0]	Input	No	Input operand to custom instruction
datab[31:0]	Input	No	Input operand to custom instruction
result[31:0]	Output	Yes	Result of custom instruction

The only required port for combinational custom instructions is the result port. The dataa and datab ports are optional. Include them only if the custom instruction functionality requires input operands. If the custom instruction requires only a single input port, use dataa. Figure 5.6 shows the combinational custom instruction hardware port timing diagram. In Figure below, the processor presents the input data on the dataa and datab ports on the rising edge of the processor clock. The processor reads the result port on the rising edge of the following processor clock cycle.

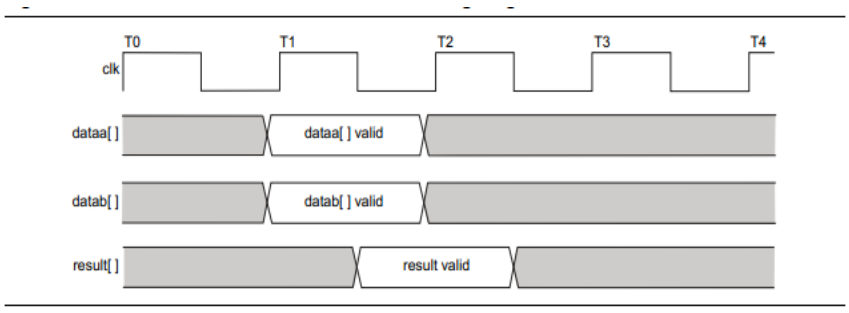


Fig 5.6

The Nios II processor issues a combinational custom instruction speculatively; that is, it optimizes execution by issuing the instruction before knowing whether it is necessary, and ignores the result if it is not required. Therefore, a combinational custom instruction must not have side effects. In particular, a combinational custom instruction cannot have an external interface.

5.*6.2.* Multicycle custom instruction

Multicycle or sequential, custom instructions consist of a logic block that requires two or more clock cycles to complete an operation. Additional control ports are required for multicycle custom instructions, as shown in following Table.

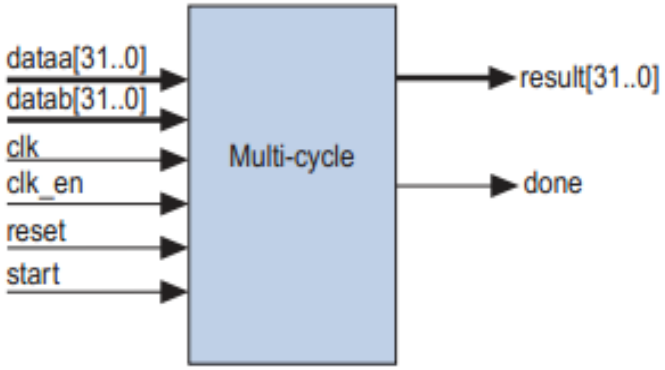


Fig 5.7 Multi cycle custom instruction block diagram

Multicycle custom instructions complete in either a fixed or variable number of clock cycles. For a custom instruction that completes in a fixed number of clock cycles, you specify the required number of clock cycles at system generation. For a custom instruction that requires a variable number of clock cycles, you instantiate the start and done ports. These ports participate in a handshaking scheme to determine when the custom instruction execution is complete.

Multi cycle custom instruction ports

Port Name	Direction	Required	Description
clk	Input	Yes	System clock
clk_en	Input	Yes	Clock enable
reset	Input	Yes	Synchronous reset
start	Input	No	Commands custom instruction logic to start execution
done	Output	No	Custom instruction logic indicates to the processor that execution is complete
dataa[31:0]	Input	No	Input operand to custom instruction
datab[31:0]	Input	No	Input operand to custom instruction
result[31:0]	Output	No	Result of custom instruction

As indicated in the above table , the clk, clk_en, and reset ports are required for multicycle custom instructions. However, the start, done, dataa, datab, and result ports are optional. Implement them only if the custom instruction functionality requires them.

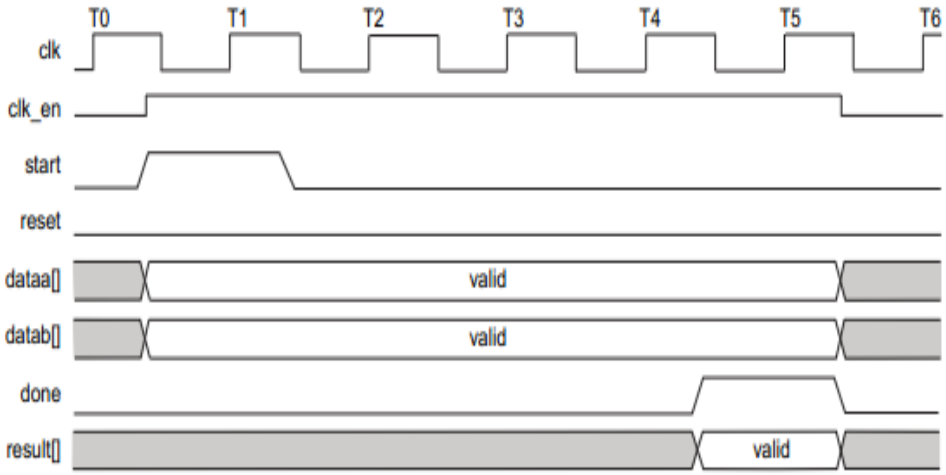


Fig 5.8 Multi cycle custom instruction timing diagram

The processor asserts the active high start port on the first clock cycle of the custom instruction execution. At this time, the dataa and datab ports have valid values and remain valid throughout the duration of the custom instruction execution. The start signal is asserted for a single clock cycle. For a fixed length multicycle custom instruction, after the instruction starts, the processor waits the specified number of clock cycles, and then reads the value on the result signal. For an n-cycle operation, the custom logic block must present valid data on the nth rising edge after the custom instruction begins execution.For a variable length multicycle custom instruction, the processor waits until the active high done signal is asserted. The processor reads the result port on the same clock edge on which done is asserted. The custom logic block must present data on the result port on the same clock cycle on which it asserts the done signal.The Nios II system clock feeds the custom logic block's clk port, and the Nios II system's master reset feeds the active high reset port. The reset port is asserted only when the whole Nios II system is reset.The custom logic block must treat the active high clk_en port as a conventional clock qualifier signal, ignoring clk while clk_en is deasserted.

5.7 CvInrangeS custom instruction

cvInrangeS( )function consumes a lot of time , so it is lbliged to optimize it using the custom instruction Concept .cvInrangeS( ) is a thresholding function that search for pixels in the same color range. The algorithm is to decide the output pixel of certain position equal wither to be set to (255 or 0) depending on input pixel value. In fact, this function is simple and doesn't need a clock so it was preferred to be combinational custom instruction.

cvInRangeS(img_HSV, cvScalar(55,60,60,0), cvScalar(73,255,255,0), image_out);

Hardware module flow chart code

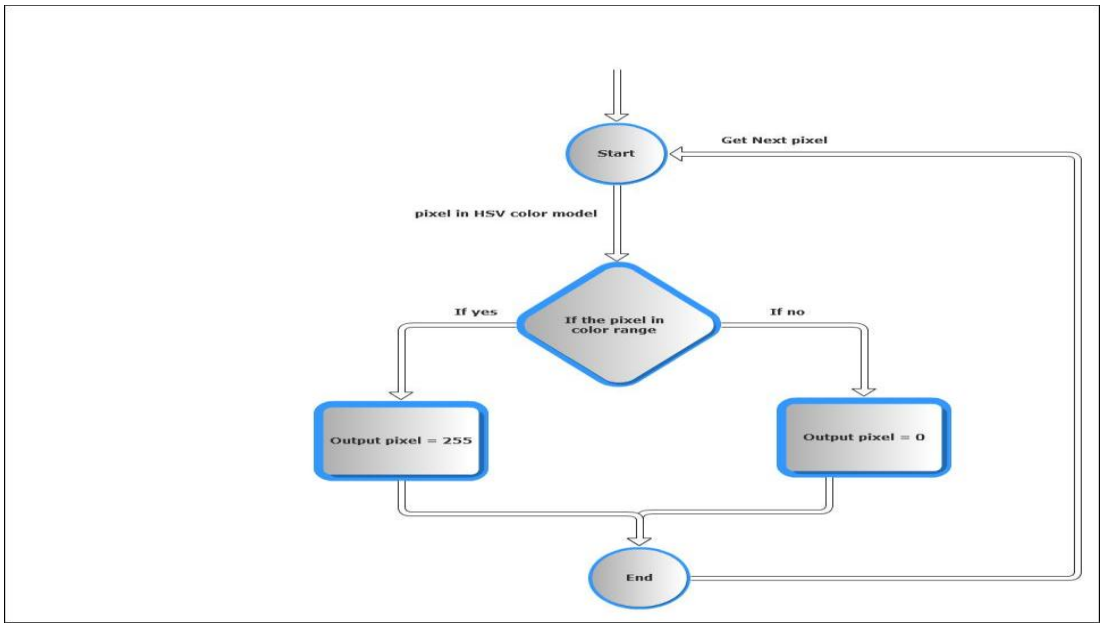


Fig 5.9

the last figure is the flow chart of the hardware module that describe the cvInrangeS()function , in this code parallelism is allowed because 2 pixels are executed at each time , which is a positive for this code.After replacing the built in code by the custom instruction, actual time of this function is decreased by a factor of two .

5.7.1 RTL View :

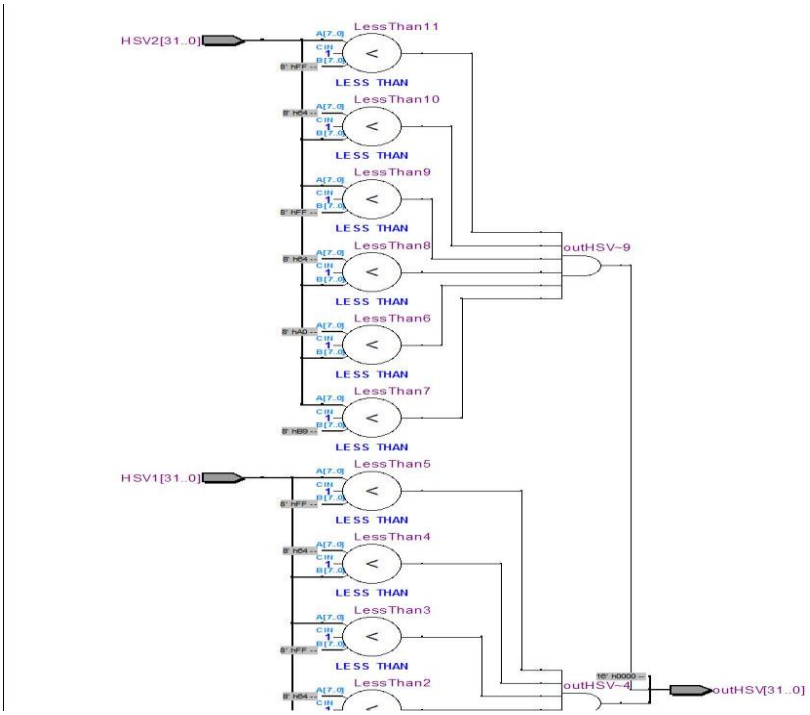


Fig 5.10

5.8 CvCvtColor() custom instruction

-CvCvtColor( ) is Time critical function which consumes about 90% of the frame processing time.

-This function converts from one color model to another (e.g. Converting from **RGB** to **HSV**).

-Function executes conversion on every pixel of the image.

- **multi cycle** custom instruction shall be used.

cvCvtColor(img_IN, img_HSV, CV_BGR2HSV)

5.8.1 RTL View

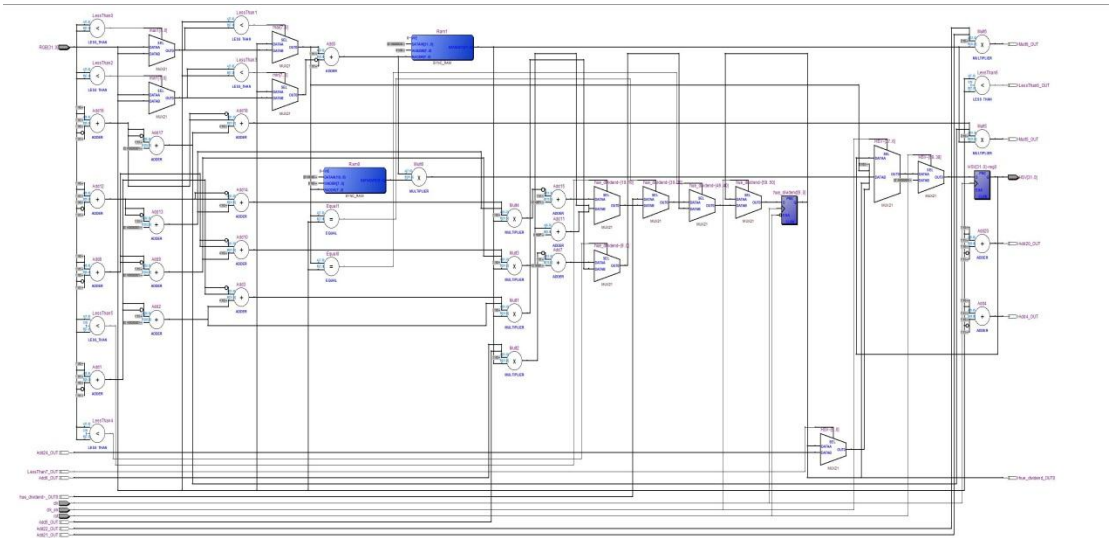


Fig 5.11

Chapter 6

Hardware Implementation

6.1. Camera Core

6.1.1. Introduction:

The main goal of this part is to fully implement an IP camera system on a FPGA. This means, in terms of hardware, to implement a system with a soft-core processor, memories, image acquisition, and image processing blocks.
Following to this intro will be the details of the design, explaining all the tasks to build the final system. It makes reference to the image sensor and its technical specifications, the implementation of the processor and its peripherals, testing and troubleshooting of the system, memory utilization and overall system performance.

6.1.2. Image Capturing:

Image acquisition process includes frames capturing with the camera CCD sensor and saving them on SDRAM memory. As a mean of monitoring, testing the process, and checking frames consistency, a VGA display is used to display residing frames in the SDRAM memory through a VGA Controller.

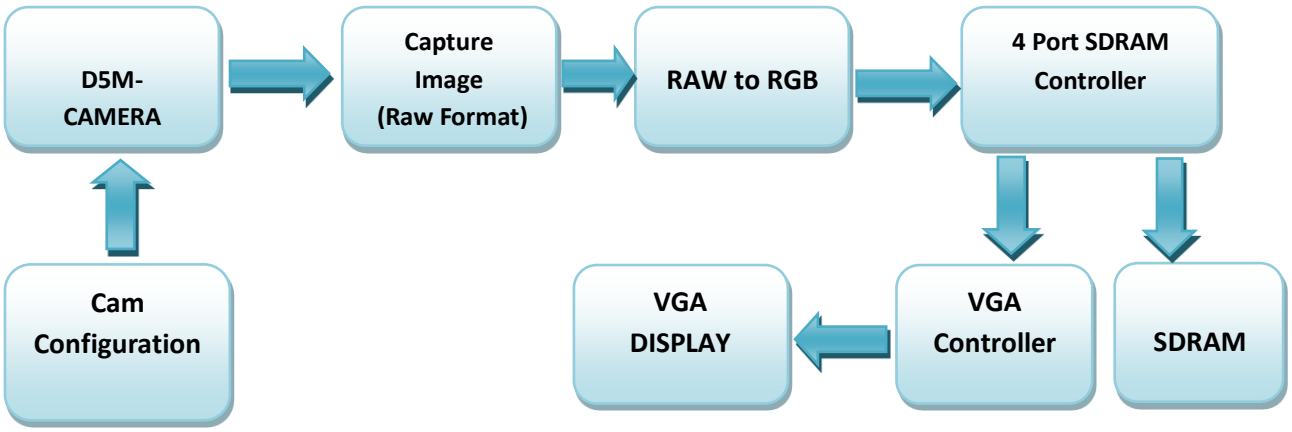


Figure 6.1 Image Capturing block diagram.

6.1.3. Preliminary Work:

6.1.3.1. Problem in hand

The Dilemma here was to schedule the allocation of the Control of the SDRAM (as a shared resource) between the Camera IP core (for writing frames in the memory) and the NIOS II processor responsible for applying the Gesture recognition algorithm on the buffered frame in the same memory.

6.1.3.2. Proposed solution

The most basic and efficient solution was to design a H/W module ”arbiter” for memory arbitration allowing both the Camera core and the NIOS II processor to take advantage of using the memory where The NIOS II processor Decides which Side controls the memory using control Signals to the arbiter through GPIO Ports.

Controlling the SDRAM

- **NIOS II SDRAM Controller**

  The SDRAM controller manages all writing, reading, sorting and synchronization. These are the signals of the SDRAM controller generated from the SOPC Builder tool provided by ALTERA. It was used by the NIOS II processor for accessing the Frame written in the memory chip.

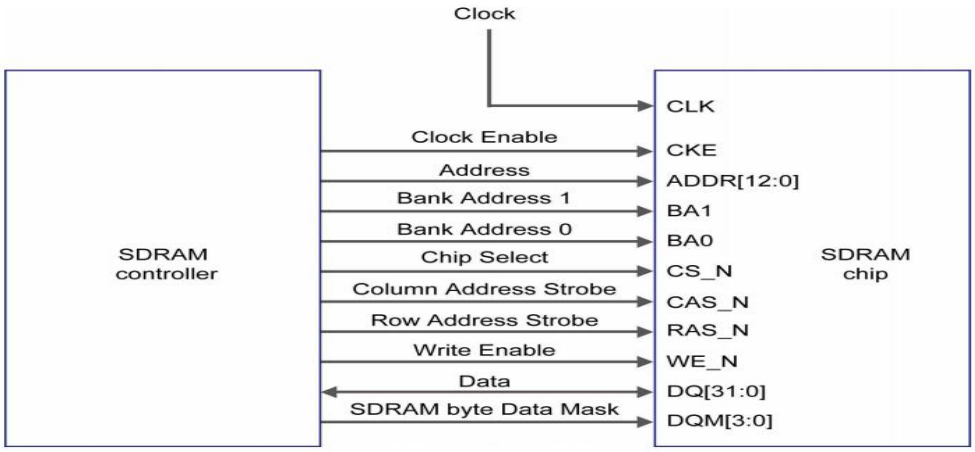


Fig 6.2 The SDRAM signals.

- **Multi-Port SDRAM Controller**

  This SDRAM chip 4 port controller is the one used by the Camera Core to deal with the SDRAM chip.
  Changes were made in terms of read and write addresses, frames are temporarily stored in the SDRAM.

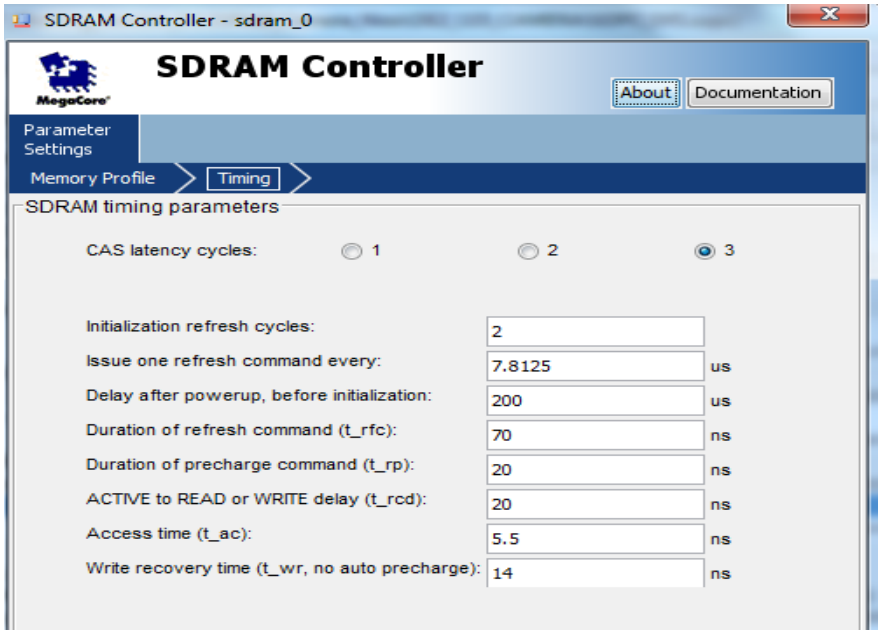


Fig 6.3.A Sdram Controller parameters

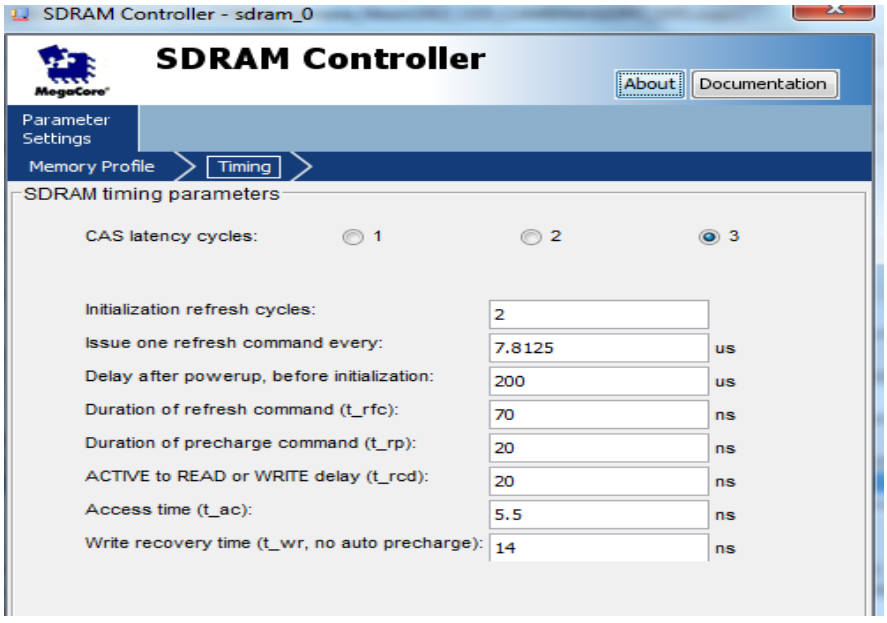
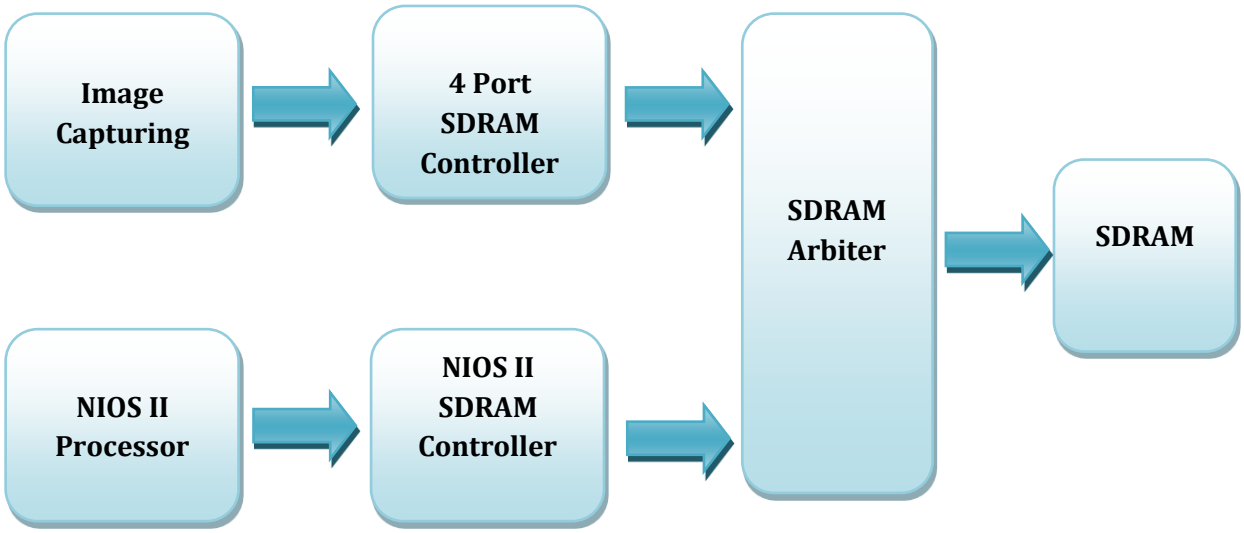


Fig 6.3.B Sdram Controller parameters

- **Initial Design**

  The System consists of the Image Capturing modules, NIOS II Processor and the SDRAM Arbiter.



- **Problems:**

  - The main problem associated with this design was the delay added by including SDRAM arbiter in the data path between camera and memory resulting in adding some noise to the captured image, which lead the algorithm to fail in detection of the hand gestures.
  - Another problem was that the software requires a large memory to run, so the SDRAM (largest memory chip on board) was needed for this task.

  The latter system was replaced due to the previously discussed problems and the final design will be described in details in the next section.

Fig 6.4 Correct Captured Image

Fig 6.5 Noisy Captured Image

6.1.4. Final design

6.1.4.1. System overview:

The final gesture recognition system was achieved by assembling of different hardware and software modules implemented in C language and Verilog HDL, Where the hardware part contains the image capturing modules that provide configuration for the D5M camera while other modules are responsible for getting image information from camera then pass it to memory controller to buffer it in memory. Nios II processor and its peripherals is the platform where the software algorithm runs. Some custom hardware modules were added to the system for acceleration. The software was implemented using C language and openCV library after porting it to be able to run on Nios II processor. In the following sections the function of each module is described.

6.1.4.2. System's Block Diagram:

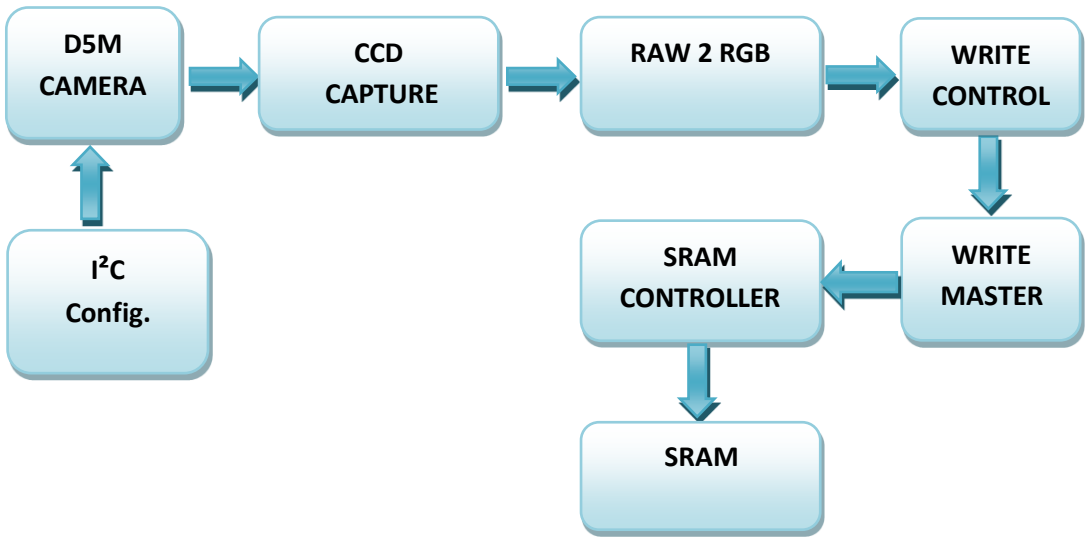


Fig 6.7 block diagram of modified camera core

- The modules that accomplish the image capturing task are:

Module name	Description
"I2C CCD Config.v"	This module is used to configure the operation of the D5M camera. It consists of two modules: The configuration module and a sub-module for transmitting data following the I2C standard. This module was modified to support the new system specifications.
"CCD Capture.v"	A module for handling the process of capturing image's data from the camera. The pixels data arrive at the rate of the pixel clock, generated from the camera internal PLL. In addition, this module counts the current number of captured frames and displays it on the seven segment displays on DE2-115.
"RAW2RGB.v"	This module converts the RAW image data from Bayer Format to RGB.
"Write Master.v"	This module is provided by Altera Avalon MM master templates, and it acts as an Avalon MM master for the SRAM memory controller. It writes the image data in the memory according to pixel data valid signal provided by "CCD Capture" module.
"Write Control.v"	This module is implemented over the write master module and it's used by the NIOS II Processor to Control the Write Master (i.e. gives the write master the appropriate addresses and signals).
"SEG7 LUT.v"	This is a module for translating binary values into Seven segment control signals. It was used for Displaying number of frames captured during Blocks of Time.
"Reset Delay.v"	This module is used to reset the system. When Key [0] is pushed, a reset will be triggered in this module. The module will then reset all hardware modules and wake them up, one by one with a little delay in between. This is done to make sure that the essential modules are ready before others that rely on them.

Table 6.1 Image Capturing Modules

6.1.4.3. Image capturing:

The image capture operation is divided into other tasks. First it is required to configure the camera sensor. In operation a raw image from the camera sensor is captured, translated to RGB (Red, Green, and Blue) format and temporarily stored in SRAM, frame by frame. Some of These blocks were originally provided by Altera and changes were made to adapt them to the project requirements, in terms of image capture resolution and SRAM's configuration to meet frame data size. The memory control module has been replaced to use SRAM memory as the frame buffer instead of using SDRAM.

- **I2C Sensor Configuration:**

The Verilog HDL module that configures programming registers of the camera is called "I2C_CCD_Config.v". This HDL file was originally written by Altera. Only configuration values were changed. The purpose of this module is to control and configure: exposure time, resolution, and frame rate. This block uses the two-wire serial interface bus to communicate with TRDB-D5M registers.

"I2C is a multi-master serial single-ended computer bus developed by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cell phone. Figure __ shows the I^2C sensor configuration block: inputs and outputs.

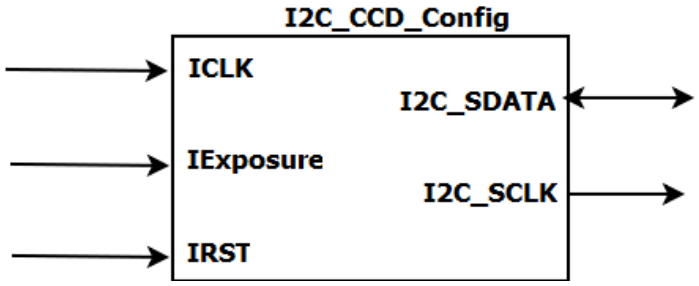


Fig 6.8 I^2C Camera Configuration module

ICLK is connected to the board main clock and IRST is connected to Reset delay module. Exposure time can be changed using toggle switches that are connected to IExposure. TRDB-D5M is a serial interface slave and is controlled by the serial clock I2C_SCLK and, data is transferred into and out of the TRDB-D5M through the serial data I2C_SDAT.

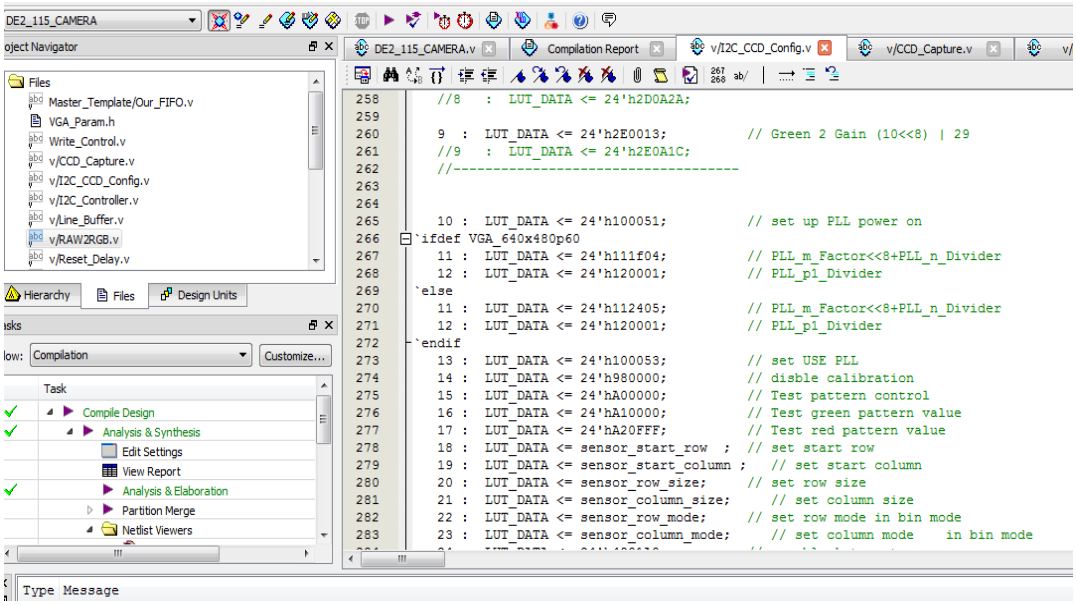


Fig 6.9 I^{2}C_Config module

➢ CMOS Sensor Data Capture:

The Verilog HDL file implementing the module that handles data capture from the camera is called "CCD_Capture.v". It was provided by Altera and no change was needed for this design. This module gets raw data from the camera sensor and sends raw data with X and Y coordinates to next module. Figure __ shows "CCD_Capture" block with inputs and outputs.

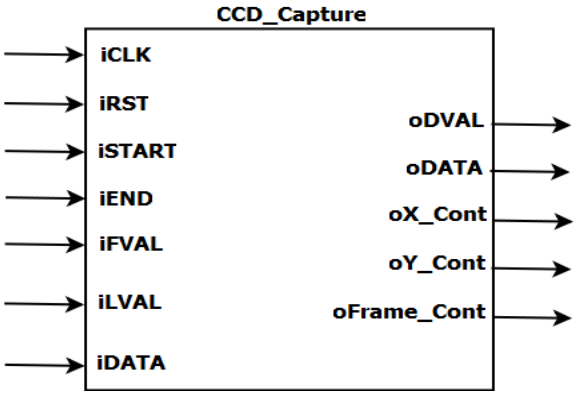


Fig 6.10 CMOS Data Capture module

iCLK is connected to camera Pixel clock and reset is connected to Reset delay module. iSTART and iEND are connected to push buttons. iSTART is the signal to

start reading raw data from sensor and calculates X and Y coordinates. iFVAL, iLVAL, and iDATA are sent by the camera sensor. Each time a valid line is set, iLVAL value is 1, and when camera sensor has a valid frame, iFVAL value is set to1. iDATA represents raw data from camera sensor.

This module has five outputs. When a pixel is ready to be sent, oDVAL value is 1. Raw data is sent by oDATA signals. This module calculates X and Y coordinates and sends these values on oX_Cont and oY_Cont. Frame count value is sent with each pixel

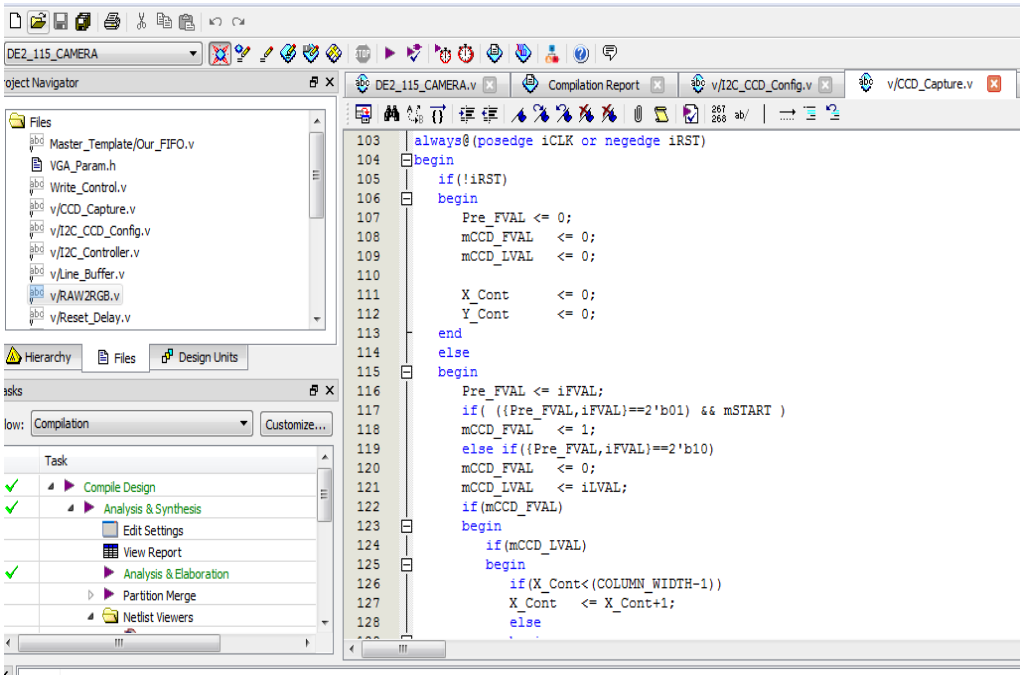


Fig 6.11 CCD_Capture module

➢ RAW to RGB

The raw data from camera is divided into Red, Green and Blue colors by the module RAW2RGB.v.This module uses the X and Y coordinates to determinate RGB values. Pixels are output in a Bayer pattern format consisting of four colors: green1, green2, red, and blue. Redundancy with green pixels produces an image which appears less noisy and has finer detail than could be accomplished if each color were treated equally. The first raw outputs data alternates between green1 and red pixels, and the second raw output alternates between blue and green2 pixels. The green1 and green2 have the same color filter and are outputted as green.

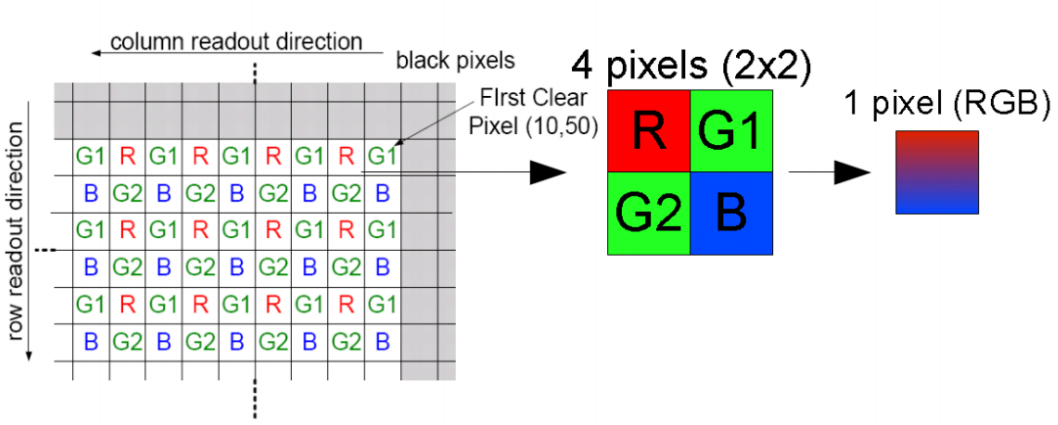


Figure 6.12 using neighboring pixels to determine one full range RGB pixel.

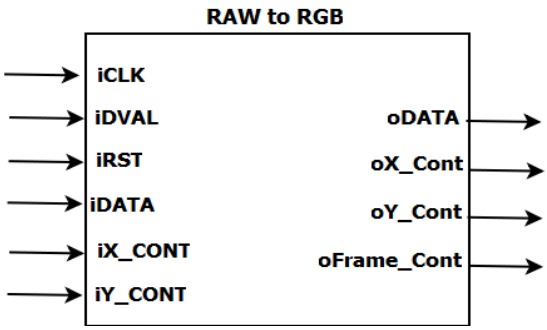


Fig 6.13 RAW to RGB Block

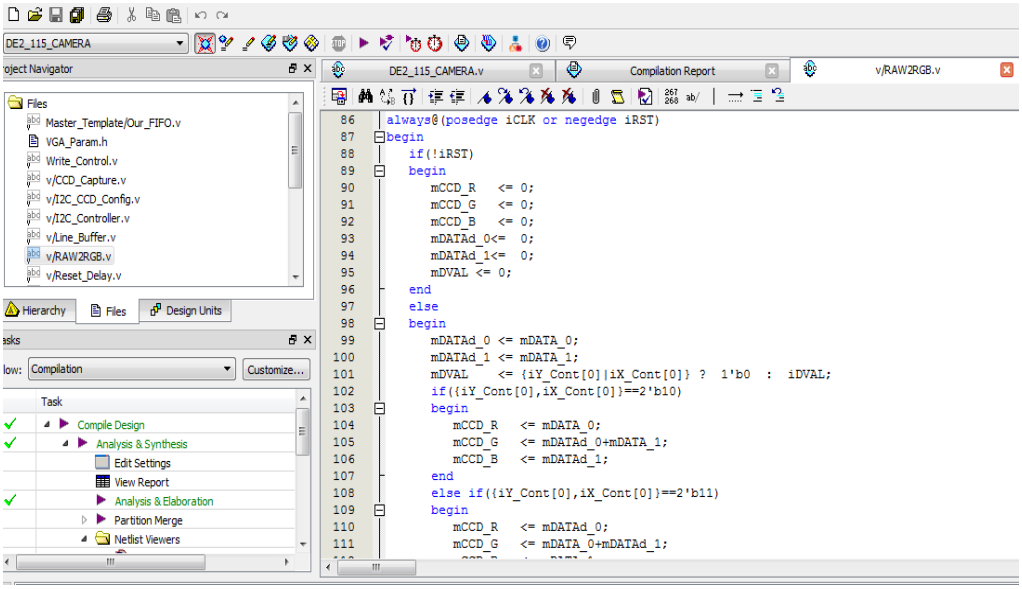


Fig 6.14 RAW to RGB Format module

6.1.4.4. SRAM Control

- In order to apply the gesture recognition algorithm on the captured image, the stream of pixels is stored in SRAM memory to construct a complete frame in order to be ready for processing. The modules that control the writing process are SRAM Controller, Write master, and Write Control. The information about the frame's start, pixel rate, and which pixels are valid are fed to these modules from CCD Capture, RAW2RGB module and the D5M camera.

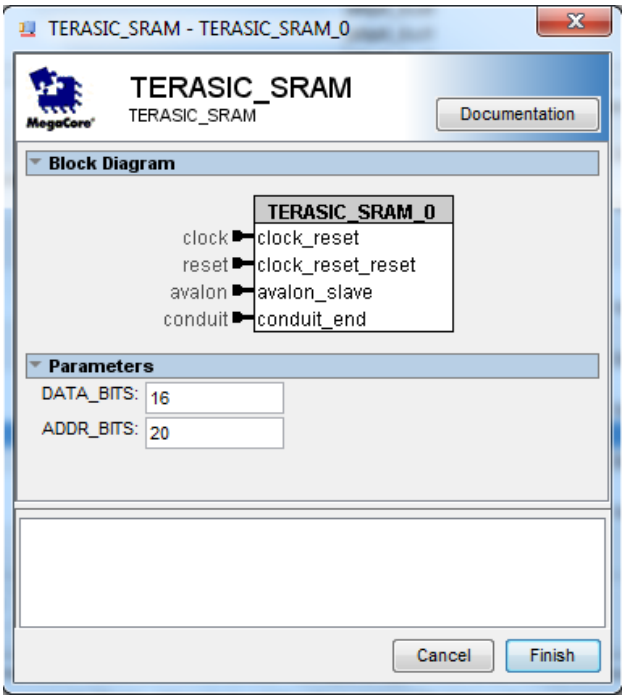


Fig 6.15 SRAM Memory Controller

- **Write Master**

  - This module is provided by Altera in the Avalon-MM package of templates which is used for adding mastering capabilities for the SRAM controller as Avalon slave component. The state machine that used to control the SRAM memory is implemented in this module. Control actions are sent to the SRAM controller through the interface of Write master with Avalon bus, and the other side of write master is used to connect the module with data and control signals from the camera system, as illustrated in Fig__.

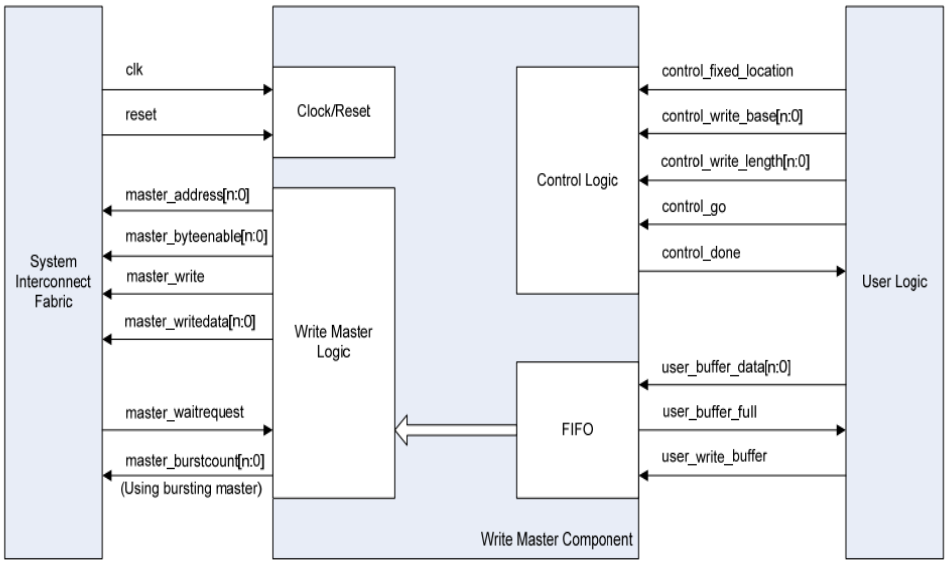


Fig 6.16 Write Master

- There are two main interfaces exposed to image writing process logic, the control interface and the data interface. Before any data is transmitted control signals must be sent first to the write master. The following tables contain details about each control signal and its usage.

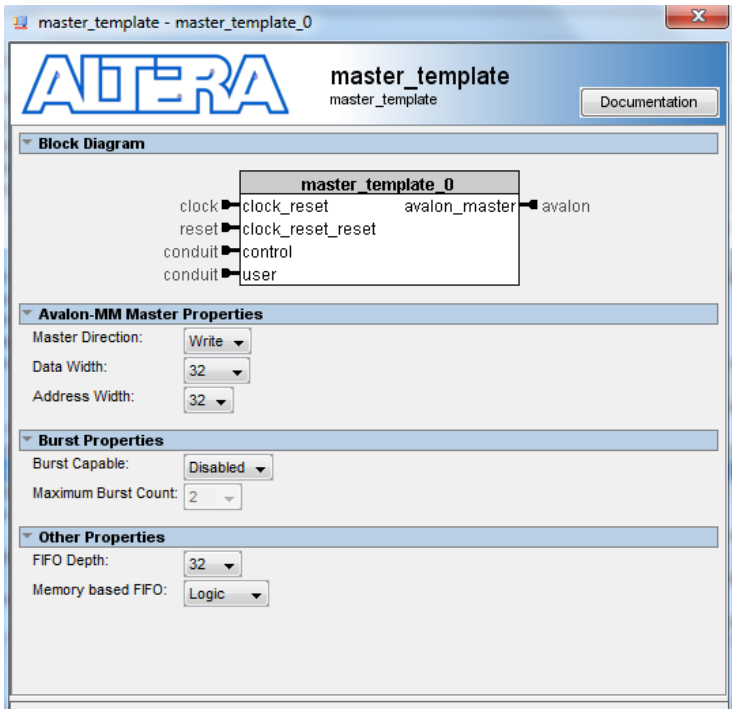


Fig 6.17 Master Template

- Write Master Control interface:

Signal Name	Direction	Width	Usage
control_fixed_location	Input	1	When set the master address will be constant and will not be incremented
control_write_base	Input	32	Word aligned byte address where the master begins transferring data.
control_write_length	Input	32	Number of bytes to transfer. This number must be a multiple of the data width in bytes.
control_go	Input	1	One clock cycle strobe that instructs the master to begin transferring. The fixed_location, base, and length values are registered on this clock cycle.
control_done	Output	1	Asserted and held when the master finished transferring the last word of data. This occurs when the last write transfer is completed or the last pending read returns. New data transfer request can start the master again on the next cycle after control_done signal is asserted.
control_early_done	Output	1	This signal is only applicable to the read masters. It is asserted when the final read is posted and not when the final word returns from the system interconnect fabric. As a result this signal is always asserted before 'control_done'.

Table 6.2 Write Master Control Interface

- Write Master User Interface:

Signal Name	Direction	Width	Usage
user_buffer_data	Input	32	Valid data word that the camera core writes into the user buffer.
user_buffer_full	Output	1	When asserted the user buffer is full. Asserting 'user_write_buffer' signal while this signal is asserted may lead data being lost and the write master failing to complete the entire transfer.
user_write_buffer	Input	1	Acts as a write qualifier. This signal is asserted to write valid data into the user buffer. It must not be asserted if 'user_buffer_full' is asserted otherwise data overflow will occur.

Table 6.3 Write Master User Interface

- **Write Control**
  - This module is implemented to provide the control interface signals to write master module. It's controlled by Nios II that requests a new frame by asserting Cam_mask signal. Cam_mask signal triggers the Control_go signal for one clock cycle to inform write master to start the data transfer operation. Nios II asserts Cam_mask each time it needs a new frame in the memory. Figure __ shows "Write_Control" block with inputs and outputs.

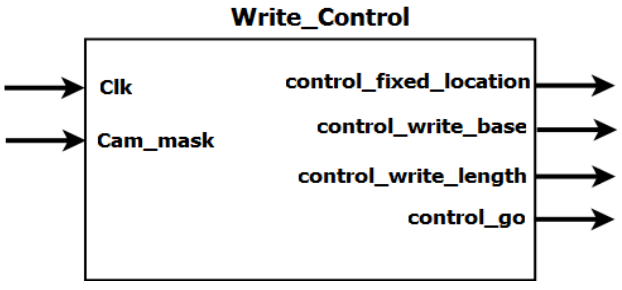


Fig 6.18 Write_Control Block

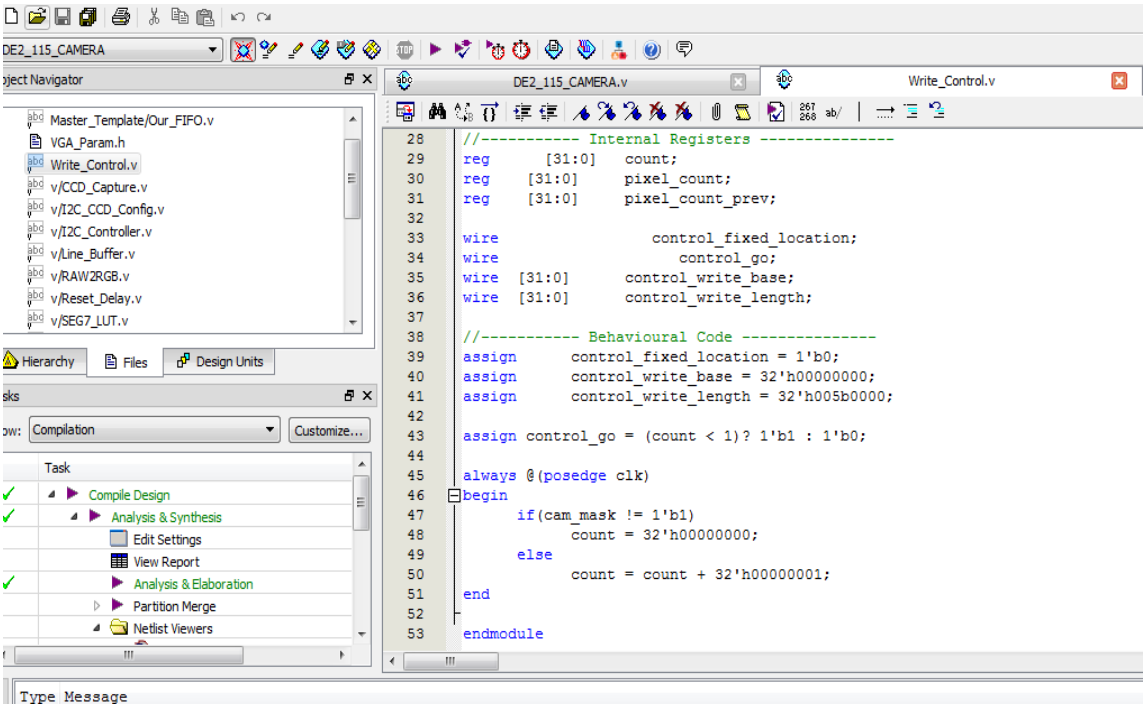


Fig 6.19 Write_Control module

6.1.4.5. Timing Specifications

The gesture recognizer assumes that there is a frame already buffered in the memory then it starts applying the recognition algorithm on that frame. The captured frame is supplied to the system pixel by pixel. So the entire system must have the same timing specifications of the D5M camera. The camera clocks and internal PLL parameter is set by the"I^2C_CCD_Config.v" module. The following section contains the specifications of D5M clocks and data timing waveforms.

- **D5M Clocks**

  The TRDB-D5M requires one clock (XCLKIN), which is nominally 96 MHz By default, or using the built-in PLL which is able to generate the pixel clock internally. The PLL is used in this design with to generate a PIXCLK equal to 77.5 MHz. The pixel data, Frame valid, and Line valid outputs are launched on the rising edge of PIXCLK, and should be captured on the falling edge of PIXCLK.

- **Output Data Timing**

  The pixel clock (PIXCLK) can be used as a clock to latch the data. For each PIXCLK cycle, one 12-bit pixel datum outputs on the DOUT pins. When both FRAME_VALID and LINE_VALID are asserted, the pixel is valid. PIXCLK cycles that occur when FRAME_VALID is negated are called vertical blanking. PIXCLK cycles that occur when only LINE_VALID is negated are called horizontal blanking. PIXCLK Represents the time needed to sample 1 pixel from the array, and is typically equal to1 EXTCLK period. The sensor outputs data at the maximum rate of one pixel per PIXCLK.

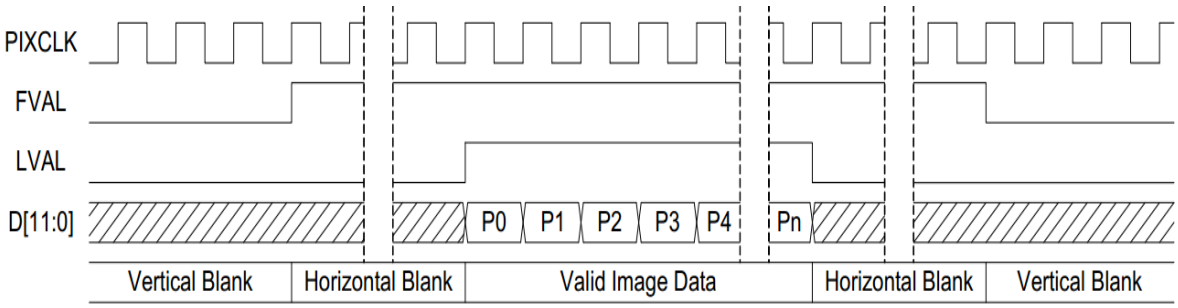


Figure 6.20 Default Pixel Output Timing

- **LINE_VALID and FRAME_VALID**

The timing of the FRAME_VALID and LINE_VALID Signals is closely related to the row time and the frame time. A valid Pixel is recognized when FRAME_VALID and LINE_VALID are both high.

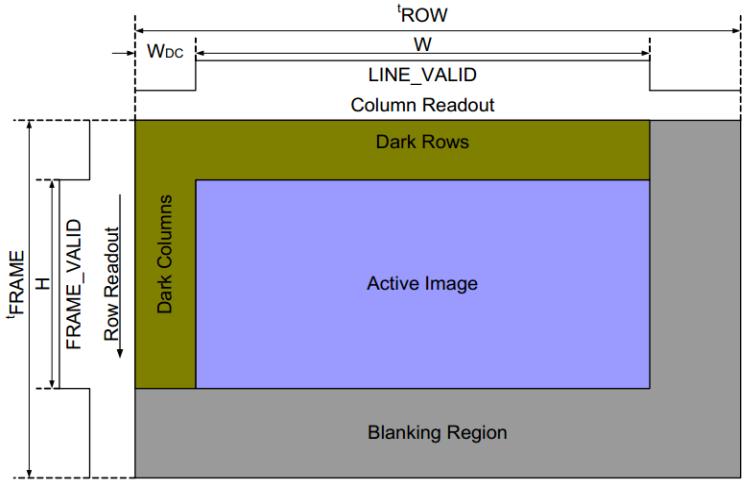


Fig 6.21 Frame Timing with FRAME_VALID and LINE_VALID signals

- **System clocks**

For the sake of synchronization between system components a PLL is used to generate different clocks. Avalon Altera PLL is the one used in this design. The following table includes inputs and outputs clocks of the PLL.

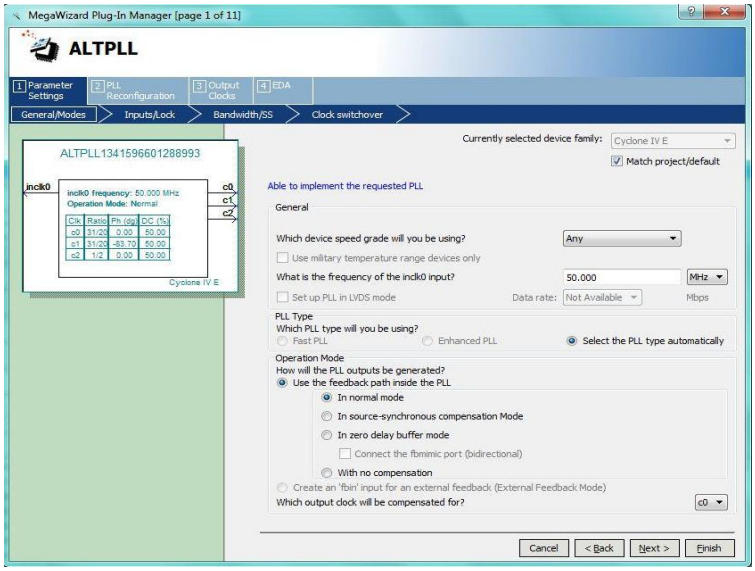


Fig 6.22 Altera PLL from Quartus II MegaWizard

Clock name	Direction	Frequency	Description
Main clock	Input	50 MHz	This clock is connected to the main clock from the DE2-115 board. All system clocks are derived from this clock.
System clock	Output	77.5 MHz	This clock is the system main clock and it's adjusted to match the incoming pixel rate from the camera which has the frequency of PXLCLK. This clock is the input clock for all system components except SDRAM chip and the D5M camera.
SDRAM clock	Output	77.5 MHz	This clock has the same frequency of system main clock but with delay -3 ns. That delay is added for proper dealing with SDRAM off-chip memory
Camera clock	Output	25 MHZ	This clock is the camera input clock and it's connected to XCLKIN of D5M camera

Table 6.4 PLL Input and Output clocks

6.1.4.6. Debugging the system

The debugging tool that used in this project was SignalTap II logic analyzer which allowed us to debug our design in real-time and at high-speed. Each time SignalTap II runs it captures data and save it to the SignalTap II File. The logic analyzer editor provides the control to select specific signals and choose when and how much data to capture from those signals. Also at least one trigger condition must occur to notify signaltap to start acquire data. The following figures contain screen shots of some signaltap waveforms that captured during debugging the system.

- Control Go signal generated by "Write_Control.v" module, which asserts it for one clock cycle for each new frame to be written. The behavior of Control go signal is illustrated in Fig__

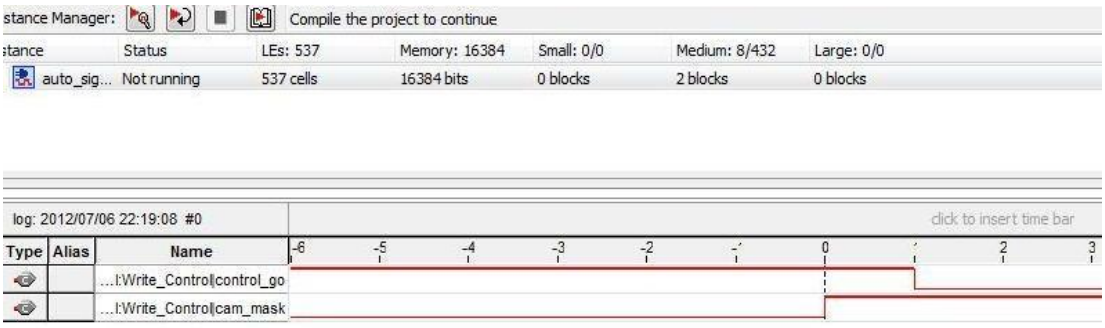


Fig 6.23 Control go signal

- Control signals of the "Write_Master.v" module that control frame writing in the memory. Fig__ shows some of those control signals provided by "Write_Control.v" and "RAW2RGB.v" modules.

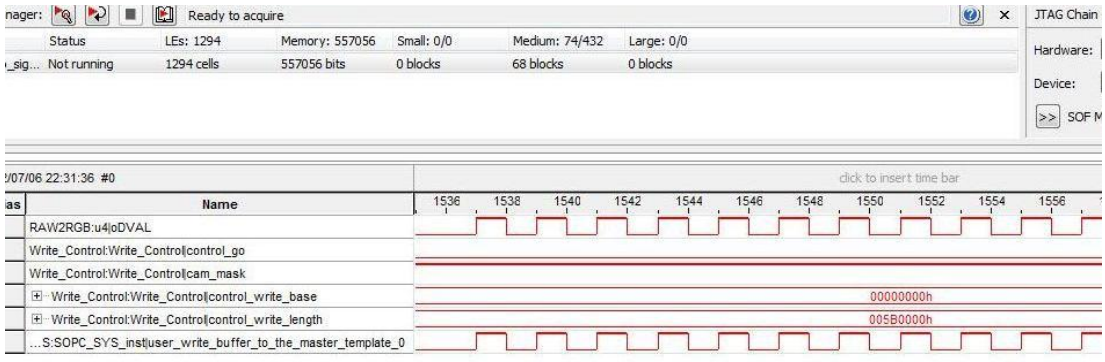


Fig 6.24 Write Master Control Signals

6.2. Multi-Nios System

6.2.1. Multiprocessor Systems

- Any system which incorporates two or more microprocessors working together to perform tasks commonly referred to as a multiprocessor system. Developers using Altera's Nios II processor and SOPC Builder tool can design and build multiprocessor systems that share resources quickly. SOPC Builder is a system development tool for creating SOPC design systems based on processors, peripherals, and memories. A Nios II processor system refers to a system with a processor core, a set of on-chip peripherals, on-chip memory and interfaces to off-chip memory all implemented on a single Altera device.

6.2.2. Benefits of Multiprocessor Systems

- Multiprocessor systems possess the benefit of increased performance, but nearly always at the price of significantly increased system complexity. For this reason, the use of multiprocessor systems has historically been limited to workstation and high-end PC computing using a complex method of load-sharing often referred to as symmetric multi-processing (SMP). While the overhead of SMP is typically too high for most embedded systems, the idea of using multiple processors to perform different tasks and functions on different processors in embedded applications (asymmetrical) is showing increased interest. Altera FPGAs provide an ideal platform for developing asymmetric embedded multiprocessor systems since the hardware can easily be modified and tuned using the SOPC Builder tool to provide optimal system performance. Furthermore, with a powerful integration tool like SOPC Builder, different system configurations can be designed, built, and evaluated very quickly.

6.2.3. Nios II multiprocessor system

- Multiple Nios II processors are able to efficiently share system resources thanks to the Multi master friendly slave-side arbitration capabilities of the Avalon bus fabric. Since the capabilities of SOPC Builder now allow users to almost effortlessly add as many processors to a system as desired, the design challenge of building multiprocessor systems no longer lies in the arranging and connecting of hardware components. The design challenge in building multiprocessor systems now lies in writing the software for those processors so they operate efficiently together, and do not conflict with one another.
- To aid in the prevention of multipleprocessors interfering with each other, a hardware cores are included in the Nios II Development Kits. E.g. (Mutex core, Mailbox core).

6.2.4. Hardware Design Considerations :

Nios II multiprocessor systems are split into two main categories, those that share resources, and those in which each processor is autonomous and does not share resources with other processors.

6.2.4.1. Autonomous Multiprocessors :

- While autonomous multiprocessor systems contain multiple processors, these processors are completely autonomous and do not communicate with the others, much as if they were completely separate systems. Systems of this type are typically less complicated and pose fewer challenges since by design, the system's processors are incapable of interfering with each other's operation. Figure 1 shows a block diagram of two autonomous processors in a multiprocessor system.

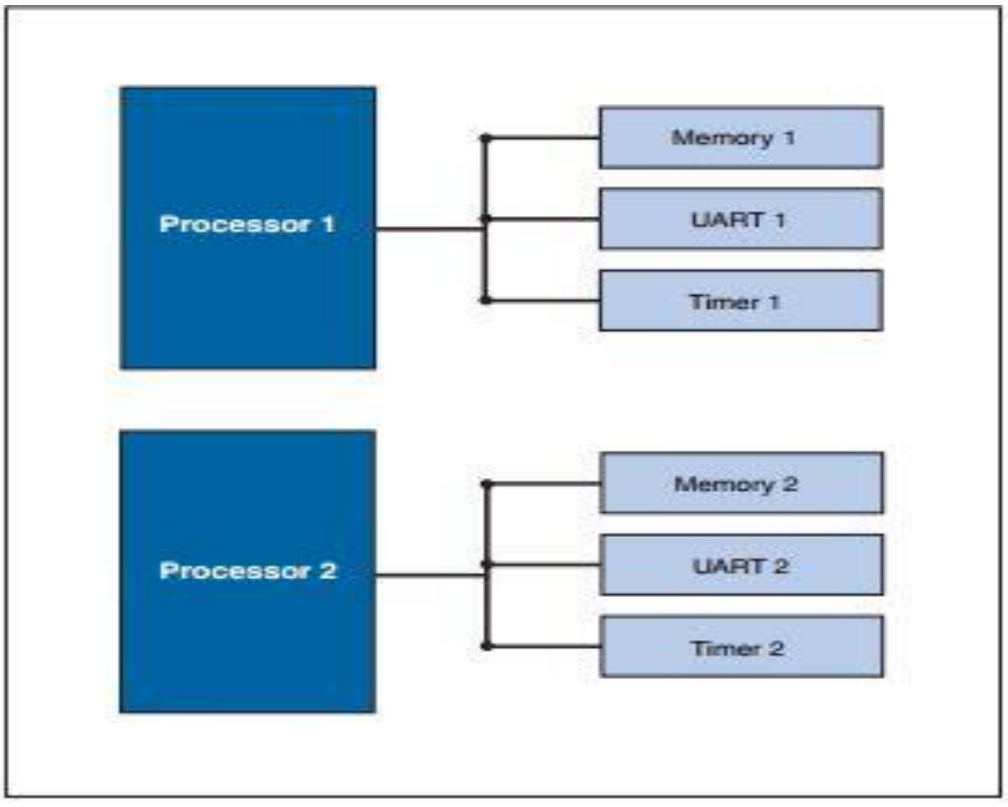


Fig 2.25 Autonomous Multiprocessor System

6.2.4.2. Multiprocessors that Share Resources

- Multiprocessor systems that share resources can pose many more challenges.

- Resources are considered shared when they are available to be accessed by more than one processor. Shared resources can be a very powerful aspect of multiprocessor systems, but care must be taken when deciding which system resources are shared, and how the different processors will cooperate regarding the use of resources. Figure 2 shows a block diagram of a multiprocessor which shares resources.

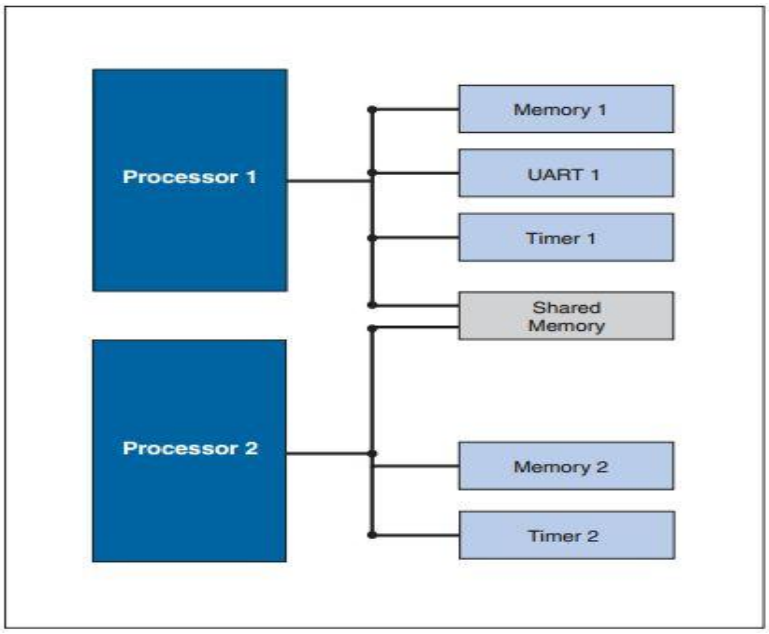


Fig 6.26 Shared Memory Multiprocessor System

6.2.5. Our Design:

We want to design and implement a multiprocessor system that consists of:

- Nios that runs the Image processing code and send the report to USB Nios.
- Nios that runs the USB Interface code and send the HID report to the PC.
- Method of interface between the Camera nios & USB nios that is responsible for communicating between them to send data.

6.2.6. Methods:

1. FIFO Buffer.
2. Directly connected using PIO.
3. Using mutex or mailbox with Shared memory.

- **Mailbox**:
  - Multiprocessor environments can use the mailbox core with Avalon interface to send messages between processors.
  - The mailbox core contains mutexes to ensure that only one processor modifies the mailbox contents at a time. The mailbox core must be used in conjunction with a separate shared memory that is used for storing the actual messages.
  - Reference:

Embedded Peripherals IP User Guide, Altera PDF Section 30.

- **FIFO:**

Fig 6.27 FIFO buffer

Inputs:

Data [31..0]: 32-bit Input data .
Wrreq: write request if set '1',the Data will be written in the fifo.
rdreq: read request if set '1',the data in fifo will be available on q[31..0].
Clock: input clock.
Sclr: Synchronous clear to flush the fifo.

Outputs:

q[31..0]: 32-bit output data.
Full: output signal if value='1' then the buffer is full.
Empty: output signal if value='1' then the buffer is empty.

- **Directly connected using PIO:**
  - Require handshake protocol to transfer the data successfully.

- **Mutex:**
  - The mutex core provides a protocol to ensure mutually exclusive ownership of a shared resource.

- The mutex core provides a hardware-based atomic test-and-set operation, allowing software in a multiprocessor environment to determine which processor owns the mutex. The mutex core can be used in conjunction with shared memory to implement additional interprocessor coordination features, such as mailboxes and software mutexes.

6.2.7. Why did we use Mailbox

- Uses Shared memory to store the USB Reports.
- The mailbox component contains two mutexes: One to ensure unique write access to shared memory and one to ensure unique read access from shared memory, ensuring message integrity.
- The mailbox software implements a message FIFO between processors.
- It is a hardware core that is easily added using SOPC builder tools.
- Availability of mailbox's API that are easily used in Nios II processor.

6.2.8. Why didn't we use FIFO

- We didn't choose FIFO as the speed of FIFO hardware is faster than the Nios processor so there is duplicate data.

6.2.9. Why didn't we use PIO

- We need Nios to work in parallel, so we didn't use directly connected using PIO.

6.2.10. SOPC system:

Use	Connections	Module Name	Description	Clock	Base	End	Tags	IRQ
☑		⊟ cpu1	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave		0x01040800	0x01040fff		
☑		⊟ cpu1_timer	Interval Timer					
		s1	Avalon Memory Mapped Slave	clk_0	0x01041400	0x0104141f		1
☑		⊟ cpu1_jtag_uart	JTAG UART					
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x01041448	0x0104144f		2
☑		⊟ cpu2	Nios II Processor					
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master		IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave		0x00040800	0x00040fff		
☑		⊟ cpu2_timer	Interval Timer					
		s1	Avalon Memory Mapped Slave	clk_0	0x00041000	0x0004101f		3
☑		⊟ cpu2_jtag_uart	JTAG UART					
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00041020	0x00041027		4
☑		⊟ mailbox1	Mailbox					
		s1	Avalon Memory Mapped Slave	clk_0	0x01041420	0x0104142f		
☑		⊟ mailbox2	Mailbox					
		s1	Avalon Memory Mapped Slave	clk_0	0x01041430	0x0104143f		
☑		⊟ mailbox_memory	On-Chip Memory (RAM or ROM)					
		s1	Avalon Memory Mapped Slave	clk_0	0x01041000	0x010413ff		
☑		⊟ mutex	Mutex					
		s1	Avalon Memory Mapped Slave	clk_0	0x01041450	0x01041457		

Fig 6.28 SOPC System

6.2.11. Configuration of Mailbox:

You can instantiate and configure the mailbox core in an SOPC Builder system using the following process:

1. Decide which processors share the mailbox.
2. On the SOPC Builder System Contents tab, instantiate a memory component to serve as the mailbox buffer. Any RAM can be used as the mailbox buffer. The mailbox buffer can share space in an existing memory, such as program memory; it does not require a dedicated memory.
3. On the SOPC Builder System Contents tab, instantiate the mailbox component.

The mailbox MegaWizard™ Interface presents the following options:

- Memory module—specifies which memory to use for the mailbox buffer. If the Memory module list does not contain the desired shared memory, the memory is not connected in the system correctly. Refer to Step 4.
- CPUs available with this memory—Shows all the processors that can share the mailbox. This field is always read-only. Use it to verify that the processor connections

are correct. If a processor that needs to share the mailbox is missing from the list, refer to Step 4 .

- Shared mailbox memory offset—Specifies an offset into the memory. The mailbox message buffer starts at this offset.
- Mailbox size (bytes)—Specifies the number of bytes to use for the mailbox message buffer. The Nios II driver software provided by Altera uses eight bytes of overhead to implement the mailbox functionality. For a mailbox capable of passing only one message at a time, Mailbox size (bytes)must be at least 12 bytes.
- Maximum available bytes—Specifies the number of bytes in the selected memory available for use as the mailbox message buffer. This field is always read-only.

4. If not already connected, make component connections on the SOPC Builder System Contents tab.

a. Connect each processor's data bus master port to the mailbox slave port.
b. Connect each processor's data bus master port to the shared mailbox memory

6.2.12. Software Programming:

- The following sections describe the software programming model for the mailbox core. For Nios II processor users, Altera provides routines to access the mailbox core hardware. These functions are specific to the mailbox core and directly manipulate low-level hardware.

- The mailbox software programming model has the following characteristics and assumes there are multiple processors accessing a single mailbox core and a shared memory:

- Each mailbox message is one 32-bit word.
- There is a predefined address range in shared memory dedicated to storing messages. The size of this address range imposes a maximum limit on the number of messages pending.
- The mailbox software implements a message FIFO between processors. Only one processor can write to the mailbox at a time, and only one processor can read from the mailbox at a time, ensuring message integrity.
- The software on both the sending and receiving processors must agree on a protocol for interpreting mailbox messages. Typically, processors treat the message as a pointer to a structure in shared memory.
- The sending processor can post messages in succession, up to the limit imposed by the size of the message address range.

- When messages exist in the mailbox, the receiving processor can read messages. The receiving processor can block until a message appears, or it can poll the mailbox for new messages.
- Reading the message removes the message from the mailbox.

- Software Files :

Altera provides the following software files accompanying the mailbox core hardware:

- altera_avalon_mailbox_regs.h—defines the core's register map, providing symbolic constants to access the low-level hardware.
- altera_avalon_mailbox.h—defines data structures and functions to access the mailbox core hardware.
- altera_avalon_mailbox.c—contains the implementations of the functions to access the mailbox core.

- Mailbox API:

This section describes the application programming interface (API) for the mailbox core.

- **altera_avalon_mailbox_close():** closes the mailbox.
- **altera_avalon_mailbox_get():** returns a message if one is present, but does not block waiting for a message
- **altera_avalon_mailbox_open():**opens a mailbox
- **altera_avalon_mailbox_pend():** is a blocking routine that waits for a message to appear in the mailbox and then reads it.
- **altera_avalon_mailbox_post() :** posts a message to the mailbox

Chapter 7

USB

In the following sections a basic introduction to USB Protocol Stack, USB Data flow and USB Framework is given, in addition to key concepts that must be understood well to write firmware for the USB protocol followed by a brief description of the HID class .

Then a description of the ISP1362 chip used on the DE2-115 board is introduced.

After that a block diagram of the system and a flow chart and a review of important functions of the firmware are presented.

Finally the tools used for debugging and testing are explained.

7.1. Introduction to USB

USB is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication and power supply between computers and electronic devices. USB was designed to standardize the connection of computer peripherals, such as keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smartphones, PDAs and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

USB version 1.1 supported two speeds, a full speed mode of 12Mbits/s and a low speed mode of 1.5Mbits/s. The 1.5Mbits/s mode is slower and less susceptible to EMI, thus reducing the cost of ferrite beads and quality components. For example, crystals can be replaced by cheaper resonators. USB 2.0 which is still yet to see day light on mainstream desktop computers has upped the stakes to 480Mbits/s. The 480Mbits/s is known as High Speed mode and was a tack on to compete with the Firewire Serial Bus.
USB Speeds .

- High Speed - 480Mbits/s
- Full Speed - 12Mbits/s
- Low Speed - 1.5Mbits/s

The Universal Serial Bus is host controlled. There can only be one host per bus. The specification in itself, does not support any form of multi master arrangement. However the On-The-Go specification which is a tack on standard to USB 2.0 has introduced a Host Negotiation Protocol which allows two devices negotiate for the role of host. This is aimed at and limited to single point to point connections such as a mobile phone and personal organizer and not multiple hub, multiple device desktop configurations. The USB host is responsible for undertaking all transactions and scheduling bandwidth. Data can be sent by various transaction methods using a token-based protocol.

USB 3.0 was released in November 2008. The standard specifies a maximum transmission speed of up to 5 Gbit/s (625 MB/s), which is more than 10 times as fast as USB 2.0 (480 Mbit/s, or 60 MB/s), although this speed is typically only achieved using powerful professional grade or developmental equipment. USB 3.0 reduces the time required for data transmission, reduces power consumption, and is backward compatible with USB 2.0. The USB 3.0 Promoter Group announced on 17 November 2008 that the specification of version 3.0 had been completed and had made the transition to the USB Implementers Forum (USB-IF), the managing body of USB specifications.

7.2. Why choosing USB?

There were multiple choices for implementing the communication interface between the DE2-115 kit and the PC like:

- Ethernet
- I2C
- UART
- USB

We chose USB according to the following factors:

1- Unlike other protocols If we identify our device as HID device, no special software or driver is needed on the PC to respond to the device commands as most operating systems support HID devices (e.g. USB optical mouse).
2 - HID reports are very flexible and are designed to suit a various set of commands.
3- Although USB is more complex, it will provide us with much more capabilities compared to other protocols like higher data rate, plug and play support.

7.3. USB Protocol Stack

Unlike RS-232 or similar serial interfaces where the format of data being sent is not defined, USB is made up of several layers of protocols. In fact most USB controller I.C.s will take care of the lower layer, thus making it almost invisible to the end designer.

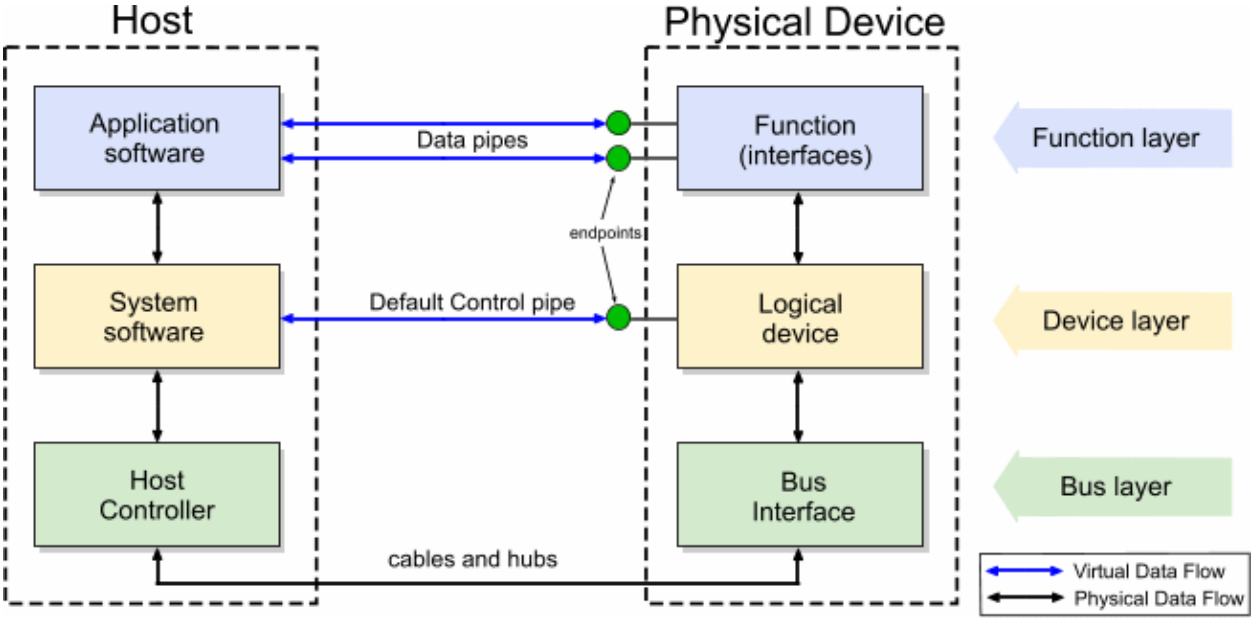


Figure 7.1

7.4. USB Data flow

USB Transactions

Each USB transaction consists of a

- **Token Packet** (Header defining what it expects to follow), an
- **Optional Data Packet**, (Containing the payload) and a
- **Status Packet** (Used to acknowledge transactions and to provide a means of error correction)

USB is a host centric bus. The host initiates all transactions. The first packet, also called a token is generated by the host to describe what is to follow and whether the data transaction will be a read or write and what the device's address and designated endpoint is. The next packet is generally a data packet carrying the payload and is followed by an handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data .

Endpoints

Endpoints can be described as sources or sinks of data. They are used to send (and receive) data to (and from) the host. Before assigning address to the device, the host communicates with the device through EP0 to perform the enumeration process. All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

Endpoint Types

The Universal Serial Bus specification defines four transfer/endpoint types,

- Control Transfers
- Interrupt Transfers
- Isochronous Transfers
- Bulk Transfers

In this document we are concerned with Control and Interrupt Transfers.

Control Transfer

Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed using control transfers. They are typically burst transfers, random packets which are initiated by the host and use best effort delivery. The packet length of control transfers in low speed devices must be 8 bytes, high speed devices allow a packet size of 8, 16, 32 or 64 bytes and full speed devices must have a packet size of 64 bytes.

A control transfer can have up to three stages.

- **The Setup Stage** is where the request is sent.

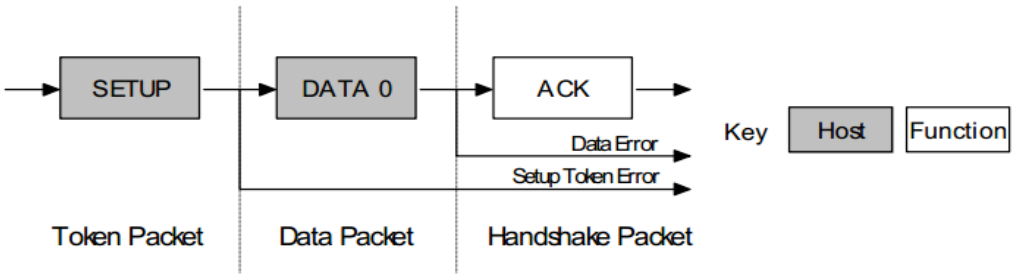


Figure 7.2

- **The optional Data Stage** consists of one or multiple IN or OUT transfers. The data stage has two different scenarios depending upon the direction of data transfer.

1- IN :

When the host is ready to receive control data it issues an IN Token.

2- OUT:

When the host needs to send the device a control data packet, it issues an OUT token followed by a data packet containing the control data as the payload

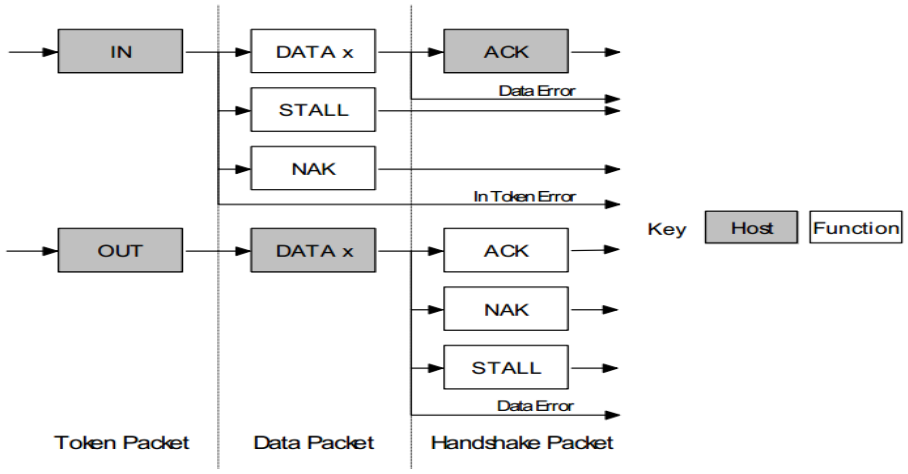


Figure 7.3

- **Status Stage** reports the status of the overall request

Interrupt Transfers

- Guaranteed Latency
- Stream Pipe – Unidirectional
- Error detection and next period retry.

Interrupt transfers are typically non-periodic, small device "initiated" communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data.

- The maximum data payload size for low-speed devices is 8 bytes.
- Maximum data payload size for full-speed devices is 64 bytes.
- Maximum data payload size for high-speed devices is 1024 bytes.

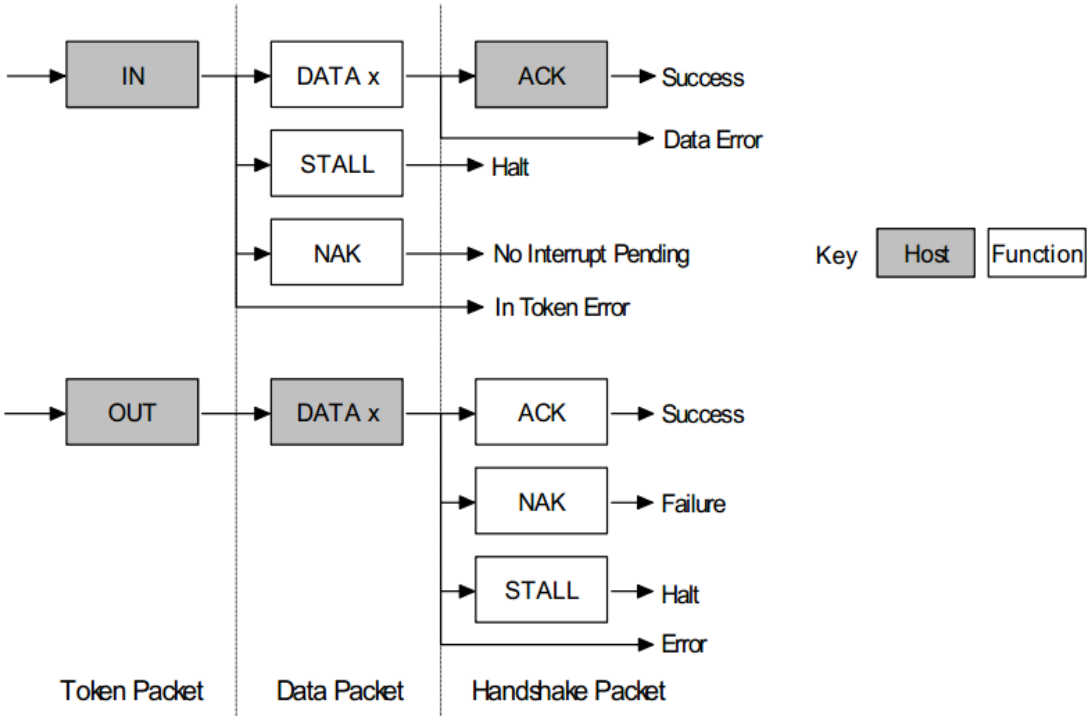


Figure 7.5

The above diagram shows the format of an Interrupt IN and Interrupt OUT transaction.

- **IN** : The host will periodically poll the interrupt endpoint. This rate of polling is specified in the endpoint descriptor which is covered later. Each poll will involve the host sending an IN Token.
In the case of a Standard USB mouse it has a polling rate of 125 Hz meaning that it sends information once every 8 miliseconds. So a timing constraint arises in our project. A frame needs to be processed each 8 ms to update the pointer coordinates in the PC.

OUT : When the host wants to send the device interrupt data, it issues an OUT token followed by a data packet containing the interrupt data .

7.5. USB Framework

7.5.1 Enumeration process

Enumeration is the stage when the host tries to identify the device through several steps:
1. The host or hub detects the connection of a new device via the device's pull up resistors on the data pair. The host waits for at least 100ms allowing for the plug to be inserted fully and for power to stabilize on the device.
2. Host issues a reset placing the device is the default state. The device may now respond to the default address zero.

3. The OS host asks for the first 64 bytes of the Device Descriptor.
4. After receiving the first 8 bytes of the Device Descriptor to know Endpoint 0 max packet size and then it immediately issues another bus reset.
5. The host now issues a Set Address command, placing the device in the addressed state.
6. The host asks for the entire 18 bytes of the Device Descriptor.
7. It then asks for 9 bytes of the Configuration Descriptor to determine the overall size.
8. The host asks for 255 bytes of the Configuration Descriptor.
9. Host asks for any String Descriptors if they were specified.
10. At this point the device is addressed but not configured so it can respond only to standard requests.
11. Once the host has all information it loads a suitable device driver.
12. Device driver will select a suitable configuration for the device using set configuration command.

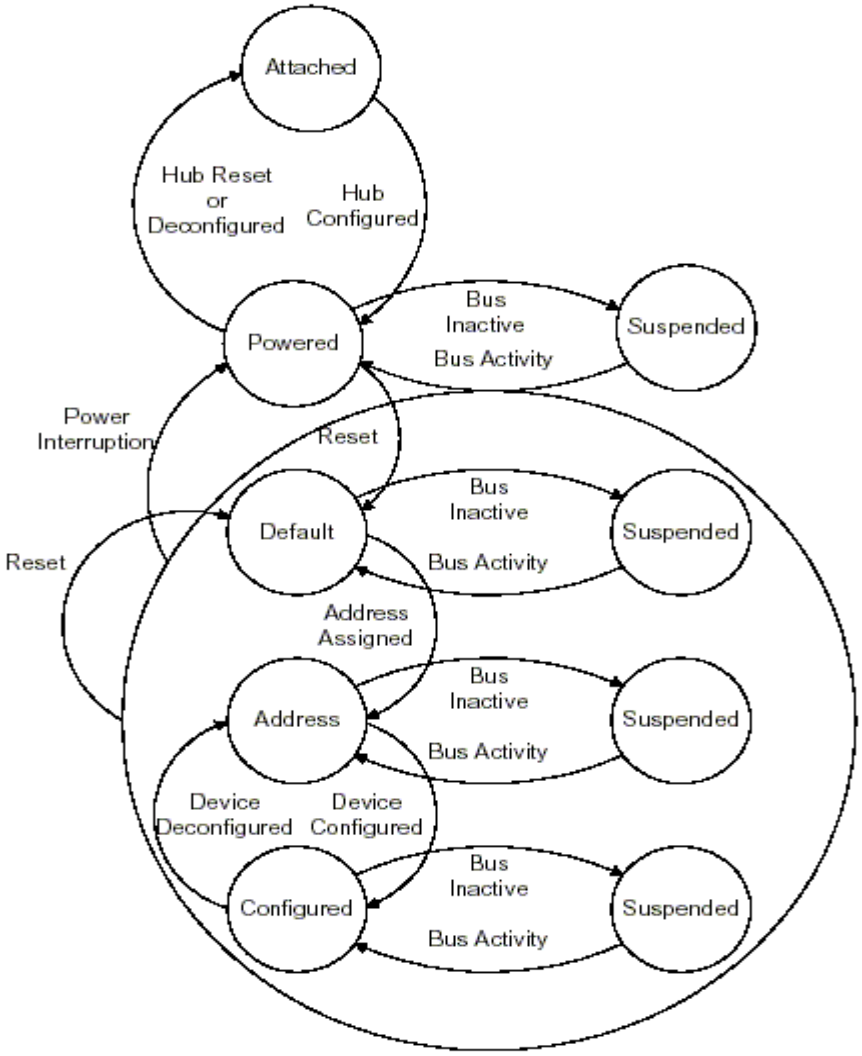


Figure 7.6

7.5.2 USB Descriptors

7.5.2.1. Introduction

All USB devices have a hierarchy of descriptors which describe to the host information such as what the device is, who makes it, what version of USB it supports, how many ways it can be configured, the number of endpoints and their types etc

The more common USB descriptors are:

- Device Descriptors
- Configuration Descriptors
- Interface Descriptors
- Endpoint Descriptors
- String Descriptors

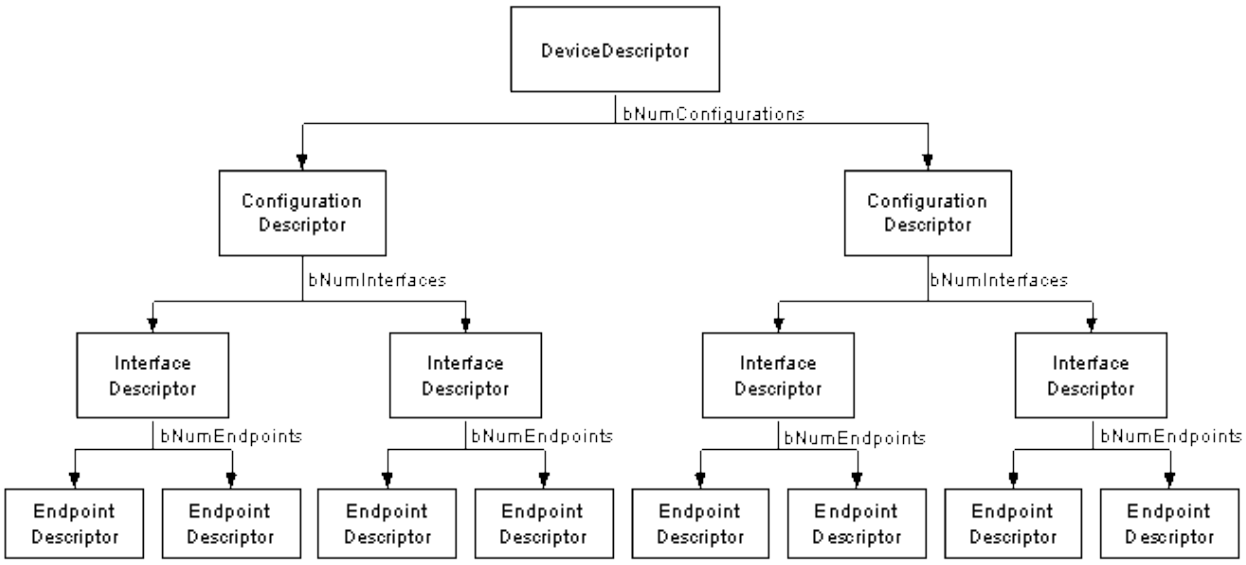


Figure 7.7

7.5.2.2. Device Descriptor

The device descriptor of a USB device represents the entire device. As a result a USB device can only have one device descriptor. It specifies some basic, yet important information about the device such as the supported USB version, maximum packet size, vendor and product IDs and the number of possible configurations the device can have.

The device descriptor was modified to adapt with our project's specifications.

- Size of the USB Device Descriptor is 18 Bytes.
- Type of descriptor is set to 1 which refers to device descriptor
- Max Packet size of Endpoint 0 is 64 Bytes.
- Number of configurations is one.

```
// Device Descriptor

0x12, //sizeof(USB_DEVICE_DESCRIPTOR),
0x01, //device descriptor type
0x00, //LSB for USB specs
0x01, //MSB for USB specs Release 0100=1.0 ,0200=2.0
0x00, //bDeviceClass,
0x00, //bDeviceSubClass,
0x00, //bDeviceProtocol,
0x40, //max EP0_PACKET_SIZE,
0xFF, // LSB Vendor ID
0xFF, // MSB Vendor ID
0x01, // LSB Product ID
0x00, // MSB Product ID
0x00, // LSB Device Release Number 1.0
0x01, // MSB Device Release Number
0, //STR_INDEX_MANUFACTURER,
0, //STR_INDEX_PRODUCT,
0, //STR_INDEX_SERIALNUMBER,
0x01 //number of configuration
```

Figure 7.8

7.5.2.3. Configuration Descriptor

A USB device can have several different configurations although the majority of devices are simple and only have one. The configuration descriptor specifies how the device is powered, what the maximum power consumption is, the number of interfaces it has. Therefore it is possible to have two configurations, one for when the device is bus powered and another when it is mains powered. As this is a "header" to the Interface descriptors, it's also feasible to have one configuration using a different transfer mode to that of another configuration.

The Configuration descriptor was modified to adapt with our project's specifications.

- Size of the USB Configuration Descriptor is 9 Bytes.
- Type of descriptor is set to 2 which refers to configuration descriptor
- Total size of upcoming descriptors (configuration, interface, HID& Endpoint) is 34 Bytes
- Number of interfaces is one.

```
// Configuratrion Descriptor

0x09,
0x02, //BYTE bDescriptorType //Assigned by USB
0x22, //BYTE wTotalLength of data returned (include config,interface,HId & endpoint descriptors)
0x00, //BYTE wZero, always 00
0x01, //BYTE bNumInterfaces
0x01, //BYTE bConfigurationValue
0x00, //BYTE iConfiguration
0xA0, //BYTE bmAttributes, Bus powered and remote wakeup
0x32, //BYTE MaxPower
```

Figure 7.9

7.5.2.4. Interface descriptor

The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device. The interface descriptor conforms to the following format.

The interface descriptor was modified to adapt with our project's specifications.

- Size of the USB interface Descriptor is 9 Bytes.
- Type of descriptor is set to 4 which refers to interface descriptor
- Number of Endpoints is one beside Endpoint 0.
- Interface class is set to HID.
- Interface protocol is set to 2 which is mouse protocol.

```
// Interface Descriptor

9,    //sizeof(struct INTERFACE), //BYTE bLength
0x04, //BYTE bDescriptionType, assigned by USB
0x00, //BYTE bInterfaceNumber
0x00, //BYTE bAlternateSetting
0x01, //BYTE bNumEndpoints, uses 2 endpoints
0x03, //BYTE bInterfaceClass, HID Class - 0x03
0x00, //BYTE bInterfaceSubClass //report protocol
0x02, //BYTE bInterfaceProtocol
0x00, //BYTE iInterface
```

Figure 7.10

7.5.2.5. Endpoint Descriptor

Endpoint descriptors are used to describe endpoints other than endpoint zero. Endpoint zero is always assumed to be a control endpoint and is configured before any descriptors are even requested. The host will use the information returned from these descriptors to determine the bandwidth requirements of the bus.
The Endpoint descriptor was modified to adapt with our project's specifications.

- Size of the USB Endpoint Descriptor is 7 Bytes.
- Define Endpoint 1 as an output Endpoint.
- Type of descriptor is set to 5 which refer to Endpoint descriptor.
- Endpoint Type is interrupt.
- Maximum Packet size of Endpoint 1 is 4 Bytes which is the size of Mouse report.

```
// Endpoint Descriptor

0x07,//size of ep descr.in bytes
0x05,//USB_ENDPOINT_DESCRIPTOR_TYPE,
0x81,//EP01, OUT
0x03,// type: 3 -> interrupt ,2 -> bulk ........
0x04,// LSB of maximum packet size //4 bytes
0x00,// MSB of maximum packet size
0x0A//interval for  polling EP in milliseconds
```

Figure 7.11

7.6. USB Human Interface Device (HID)

7.6.1. Introduction

It is a part of the USB specification for computer peripherals, it specifies a device class (a type of computer hardware) for human interface devices such as keyboards, mice, game controllers and alphanumeric display devices.

Why choosing USB HID Class?

The USB HID class describes devices used with nearly every modern computer. Many predefined functions exist in the USB HID class. These functions allow hardware

manufacturers to design a product to USB HID class specifications and expect it to work with any software that also meets these specifications.

Keyboards

Keyboards are some of the most popular USB HID class devices. The USB HID class keyboard is normally designed with an IN endpoint that communicates keystrokes to the computer and an OUT endpoint that communicates the status of the keyboard's LEDs from the computer to the keyboard. The PC 97 standard requires that a computer's BIOS must detect and work with USB HID class keyboards that are designed to be used during the boot process.

Mice

Computer mice are equally popular USB HID class devices. USB HID mice can range from single-button simple devices to multi-button compound devices. Most modern operating systems ship with drivers for standard HID mice designs (the most common modern mouse design has two dedicated buttons and a mouse wheel that doubles as the third button); mice with extended functionality require custom drivers from the manufacturer.

Game controllers

Modern game controllers and joysticks are often USB HID class devices. Unlike legacy game port devices, USB HID class game devices do not normally require proprietary drivers to function. Nearly all game devices will function using onboard drivers as long as the device is designed around the drivers and the USB HID class specifications.

Other devices

The USB HID class specifications allow for myriad other devices under the USB HID class. Some examples are automobile simulation controllers, exercise machines, telephony devices, thermometers, audio controls and medical instrumentation. Even uninterruptible power supplies declare themselves under this class, despite the fact they often have no human interface at all. Any device can be a USB HID class device as long as a designer meets the USB HID class logical specifications. This is not to say that there is no need to ship drivers for these devices, nor that an operating system will immediately recognize the device. This only means that the device can declare itself under the human interface device class.

Drivers

One of the benefits of a well-defined specification like the USB HID class is the abundance of device drivers available in most modern operating systems. The USB HID class devices and their basic functions are defined in USB-IF documentation without any specific software in mind. Because of these generic descriptions, it is easy for operating system designers to include functioning drivers for devices such as keyboards, mice, and other generic human interface devices. The inclusion of these generic drivers allows for faster deployment of devices and easier installation by end-users.

7.6.2. HID Descriptor

The Endpoint descriptor was modified to adapt with our project's specifications.

- Size of the HID Descriptor is 9 Bytes.
- Type of descriptor is set to 33 which refers to HID descriptor
- Total size of upcoming descriptors (configuration, interface, HID& Endpoint) is 34 Bytes
- Number of HID report Descriptor is one.
- The size of Report Descriptor is 64 Byte

```
// HID Descriptor

9,//sizeof(struct HID descriptor), //BYTE bLength
0x21, //BYTE bDescriptionType, assigned by USB HID discriptor
0x10, //LSB HID specs
0x01, //MSB HID specs 1.0
0x00, //country code
0x01, //number of hid class descriptor to follow (report)
0x22, //report descriptor type
0x40, //LSB of report descriptor size
0x00, //MSB of report descriptor size
```

Figure 7.12

7.6.3. Report Descriptor

Our HID mouse consists of:

- 1 Byte that represent 3 Buttons (Left, right &middle click) and 5 bits padding.
- 2 Bytes for Motion in X, Y direction.
- 1 Byte for Wheel.

```
//Report Descriptor Part(1)

0x05,0x01, // Usage Page (Generic Desktop)
0x09,0x02, // Usage (Mouse)
0xA1,0x01, // Collection (Application)

0x09,0x01, // Usage (Pointer)
0xA1,0x00, // Collection (Physical)
0x05,0x09, // Usage Page (Buttons)
0x19,0x01, // Usage Minimum (01)
0x29,0x03, // Usage Maximun (03)
0x15,0x00, // Logical Minimum (0)
0x25,0x01, // Logical Maximum (1)
0x95,0x03, // Report Count (3)
0x75,0x01, // Report Size (1)
0x81,0x02, // Input (Data, Variable, Absolute),        ;3 button bits
0x95,0x01, // Report Count (1)
0x75,0x05, // Report Size (5)
0x81,0x03, // Input (Constant)                         ;5 bit padding

0x05,0x01, // Usage Page (Generic Desktop)
0x09,0x30, // Usage (X)
0x09,0x31, // Usage (Y)
0x15,0x81, // Logical Minimum (-127)
0x25,0x7F, // Logical Maximum (127)
0x75,0x08, // Report Size (8)
0x95,0x02, // Report Count (2)
0x81,0x06, // Input (Data, Variable, Relative)
```

Figure 7.13

```
//Report Descriptor Part(2)

0x05, 0x01, // Usage Page (Generic Desktop)
0x09, 0x38, // Usage (Wheel)
0x15, 0x81, // Logical Minimum (-127)
0x25, 0x7F, // Logical Maximum (127)
0x95, 0x01, // Report Count (1)
0x75, 0x08, // Report Size (8)
0x81, 0x06, // Input (Data, Variable, Relative)

0xC0,       // End Collection
0xC0        // End Collection
```

Figure 7.14

7.6.4. Report Descriptor Parser

The HID class driver contains a parser used to analyze items found in the Report descriptor. The parser extracts information from the descriptor in a linear fashion. The parser collects the state of each known item as it walks through the descriptor, and stores them in an item state table. The item state table contains the state of individual items.

From the parser's point of view, a HID class device looks like the following figure:

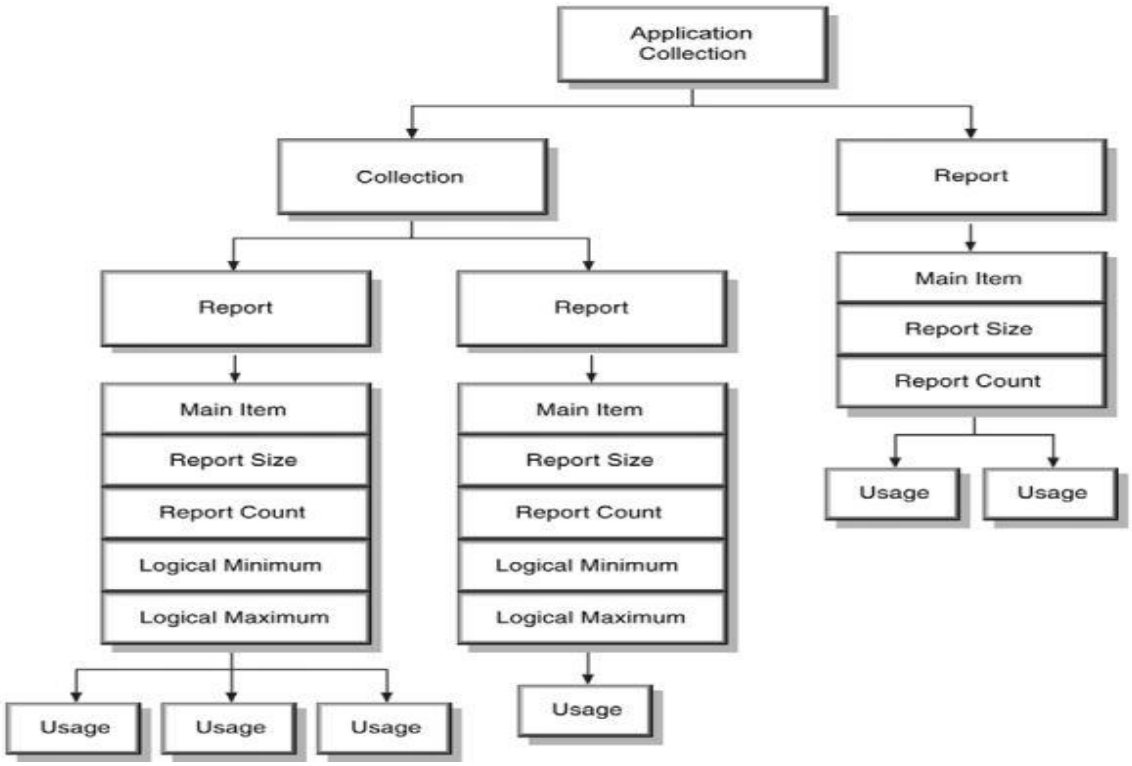


Figure 7.15

7.7. USB on DE2-115

The DE2-115 board provides both USB host and device interfaces using the Philips ISP1362 single-chip USB controller. The host and device controllers are compliant with the Universal Serial Bus Specification Rev. 2.0, supporting data transfer at full-speed (12 Mbit/s) and low-speed (1.5 Mbit/s). Figure 7.6 shows the schematic diagram of the USB

circuitry on the DE2-115 board.

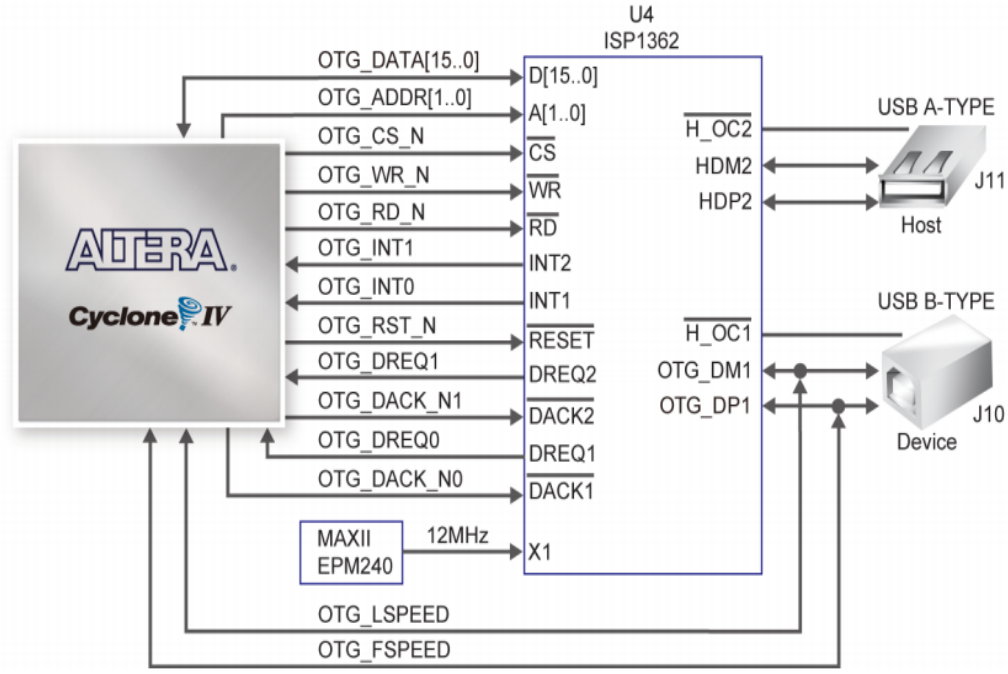


Figure 7.16

7.7.1. Relevant Pins description:

Pin	Description
$\overline{\text{RESET}}$	reset input
D0..15	bidirectional data bus that connects to the internal registers and buffer memory of the ISP1362
A0	command or data phase
A1	LOW — PIO bus of the HC is selected HIGH — PIO bus of the DC is selected
$\overline{\text{RD}}$	read strobe input; when asserted LOW, it indicates that the HC/DC driver is requesting a read to the buffer memory or the internal registers of the HC/DC
$\overline{\text{WR}}$	write strobe input; when asserted LOW, it indicates that the HC/DC driver is requesting a write to the

	buffer memory or the internal registers of the HC/DC
INT	interrupt request from the HC; provides a mechanism for the HC to interrupt the microprocessor;
INT2	interrupt request from the DC; provides a mechanism for the DC to interrupt the microprocessor;
OTG_DM1	D− signal of the OTG port, the downstream host port 1 or the upstream device port; when not in use, leave this pin open and set bit ConnectPullDown_DS1 of the HcHardwareConfiguration register
	D+ signal of the OTG port, the downstream host port 1 or the upstream device port; when not in use, leave this pin open and set bit ConnectPullDown_DS1 of the HcHardwareConfiguration register[

Table 1

7.7.2 ISP 1362

The ISP1362 is a single-chip Universal Serial Bus (USB) On-The-Go (OTG) controller integrated with the advanced Philips Slave Host Controller (PSHC) and the Philips ISP1181B Device Controller (DC). The USB OTG controller is compliant with On-The-Go Supplement to the USB 2.0 Specification Rev. 1.0a. The host and device controllers are compliant with Universal Serial Bus Specification Rev. 2.0, supporting data transfer at full-speed (12 Mbit/s) and low-speed (1.5 Mbit/s).

The ISP1362 has two USB ports: port 1 and port 2. Port 1 can be hardware configured to function as a downstream port, an upstream port or an OTG port whereas port 2 can only be used as a downstream port. The OTG port can switch
roles from host to peripheral, or from peripheral to host. The OTG port can become a host through the Host Negotiation Protocol (HNP) as specified in the OTG supplement.

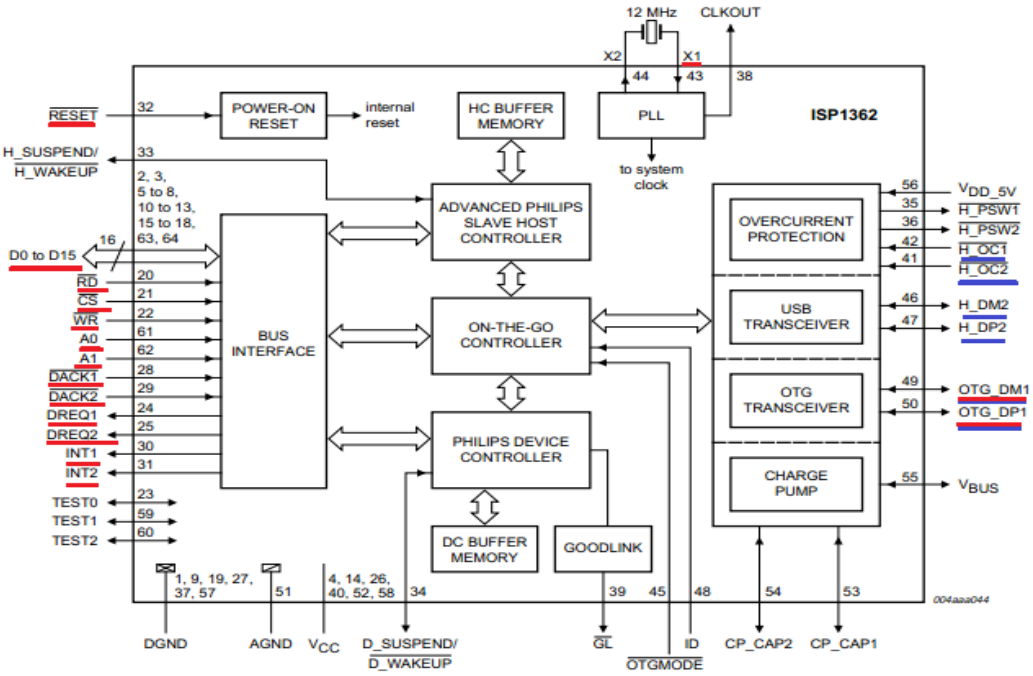


Figure 7.17 shows the block diagram of ISP 1362 chip:

The pins underlined in red are connected to the board while the pins underlined with blue are connected to the host and device PHYs.

7.7.2.1 PIO access mode

There should be an external microprocessor that is responsible for operating and controlling the chip.

The ISP1362 provides the PIO mode for external microprocessors to access its internal control registers and buffer memory. It occupies only four I/O ports or four memory locations of a microprocessor. An external microprocessor can read or write to the internal control registers and buffer memory of the ISP1362 through the PIO operating mode. Figure 7.18 shows the PIO interface between a microprocessor and the ISP1362.

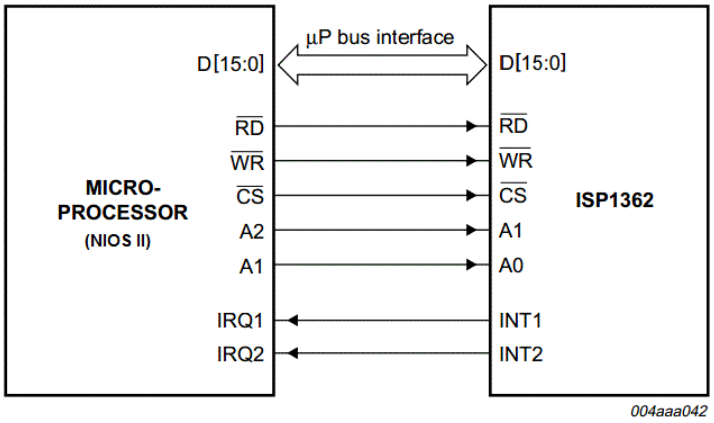


Figure 7.18

7.7.2.2 Memory organization for the DC

The ISP1362 DC has a total of 2462 bytes of built-in buffer memory. This buffer memory is multiconfigurable to support the requirements of different applications. The DC buffer memory is divided into 16 areas to be used by control OUT, control IN and14 programmable endpoints.
Figure 7 provides a snapshot of the DC buffer memory.

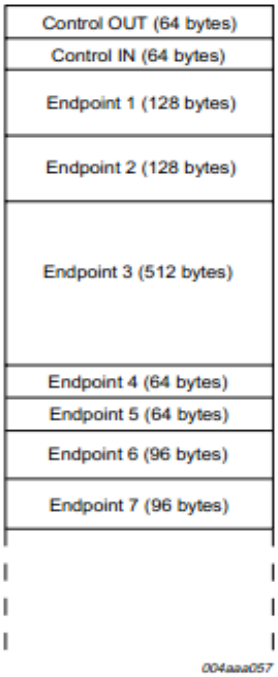


Figure 7.19

7.7.2.3 USB Device Controller (DC)

In our project the ISP1362 is configured to work in Device Mode while the laptop or PC will act as a host. The DC in the ISP1362 provides 16 endpoints for the USB device implementation. Each endpoint can be allocated RAM space in the on-chip ping pong buffer RAM.

Remark: The ping pong buffer RAM for the DC is independent of the buffer RAM for the Host Controller (HC). When the buffer RAM is full, the DC transfers the data in the buffer RAM to the USB bus. When the buffer RAM is empty, an interrupt is generated to notify the microprocessor to feed in data. The transfer of data between a microprocessor and the DC can be done in either the Programmed I/O (PIO) mode or in the direct memory access (DMA) mode.

7.7.2.4 DC data transfer operation

We configured the ISP1362 to accept IN data transfers. An IN data transfer means transfer from the ISP1362 to an external USB host (through the upstream port). In the device mode, the ISP1362 acts as a USB device.

7.7.2.5 IN data transfer

The arrival of the IN token is detected by the Serial Interface Engine (SIE) by decoding the Packet IDentifier (PID).

- The SIE also checks the device no-umber and the endpoint number to verify whether they are okay.
- If the endpoint is enabled, the SIE checks the contents of the DcEndpointStatus register (ESR). If the endpoint is full, the contents of the buffer memory are sent during the data phase else an NAK handshake is sent.
- After the data phase, the SIE expects a handshake (ACK) from the host (except for ISO endpoints).
- On receiving the handshake (ACK), the SIE updates the contents of the DcEndpointStatus and DcInterrupt registers, which in turn generates an interrupt to the microprocessor. For ISO endpoints, the DcInterrupt register is updated as soon as data is sent because there is no handshake phase.
- On receiving an interrupt, the microprocessor reads the DcInterrupt register. It will know which endpoint has generated the interrupt and reads the contents of the corresponding ESR. If the buffer is empty, it fills up the buffer so that data can be sent bythe SIE at the next IN token phase

- ## 7.8 System Architecture

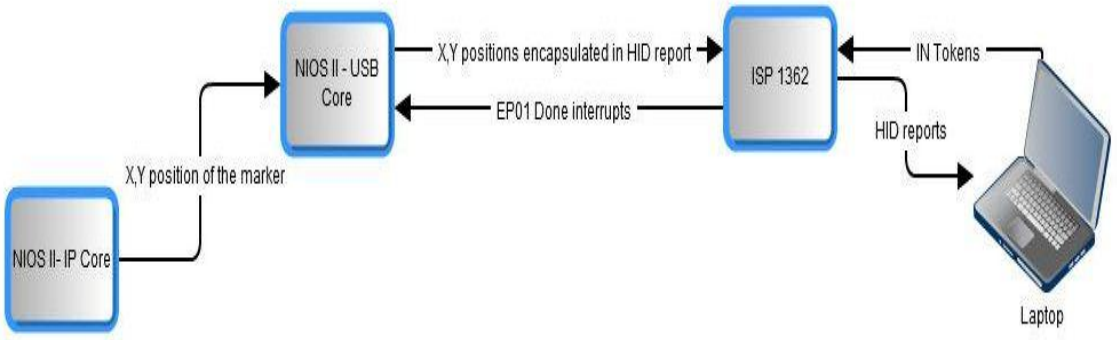


Figure 7.20

The NIOSII-Image processing core constantly sends the X and Y position of the marker to the NIOSII-USB core. The data is encapsulated in a HID report and written to EP01(Any Endpoint can be used but EP01 was used in our project) buffer of the chip. The PC frequently sends IN tokens to the ISP1362 and reads the data from EP01 to update the mouse position.

7.9 Firmware

7.9.1. Firmware Structure of the Device Controller

The firmware consists of two major portions: the processing of information and the interrupt service routine. The Hardware Abstraction layer just moves data from hardware to memory space to be processed by the Main Loop as shown in Figure 7.21

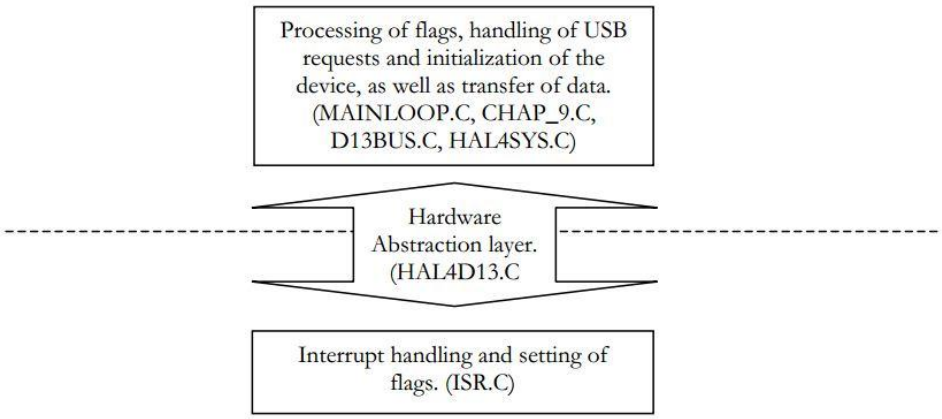


Figure 7.21: Firmware Structure of the Device Controller of the ISP1362

As can be seen in Figure 7.21, the firmware structure can be divided into the following six building blocks:

- Hardware Abstraction Layer—HAL4SYS.C
- Hardware Abstraction Layer—HAL4D13.C
- Interrupt Service Routine—ISR.C
- Protocol Layer—CHAP_9.C
- Protocol Layer—D13BUS.C
- Main Loop—MAINLOOP.C.

Hardware Abstraction Layer—HAL4SYS.C
This is the lowest-layer code in the firmware that performs hardware-dependent I/O access of the Device Controller of the ISP1362, as well as the evaluation board hardware. When porting the firmware to other CPU platforms, this part of the code always needs modifications or additions.

Hardware Abstraction Layer—HAL4D13.C
To further simplify programming with the Device Controller of the ISP1362,the firmware defines a set of command interfaces that encapsulate all the functions used to access the Device Controller of the ISP1362. When porting the firmware to other operation systems, this portion of the code must be modified.

Interrupt Service Routine—ISR.C
This part of the code handles interrupt generated by the Device Controller of the ISP1362. It retrieves data from the ISP1362 Device Controller's internal FIFO to CPU memory and sets up proper event flags to inform the Main Loop program to process.

Protocol Layer—CHAP_9.C
This Protocol layer handles standard USB device request, which is defined in the Chapter 9 of USB Specification Rev. 2.0.

Protocol Layer—D13BUS.C
This Protocol layer handles specific vendor requests. Examples are the bulktransfer and the isochronous (ISO) transfer.

Main Loop—MAINLOOP.C
The Main Loop checks event flags and passes to appropriate subroutine for further processing. It also contains the code for human interface, suchas the keyboard scan.

7.9.2 Important functions in HAL for the device controller

Hal4D13_SetEndpointConfig(UCHAR bEPConfig, UCHAR bEPIndex);
//Configure the endpoint, define the FIFO size and the EP direction.
//Modifies the content of DcEndpointConfigurationRegister.
Hal4D13_GetEndpointConfig(UCHAR bEPIndex);
//Get the endpoint configuration.

Hal4D13_SetAddressEnable(UCHAR bAddress, UCHAR bEnable);
//Enable and Set the device address.
//Modifies the content of DcAddressRegister.
Hal4D13_GetAddress(void);
//Get the address of the device.

Hal4D13_SetIntEnable(ULONG dIntEn);
//Enable different interrupts according to the interrupt mask dIntEn.
//Modifies the content of DcAddressRegister.
Hal4D13_GetIntEnable(void);
//Get the address of the device.

Hal4D13_ResetDevice(void);
//Reset the device.

Hal4D13_WriteEndpoint(UCHAR bEPIndex, UCHAR * buf, USHORT len);
//Write data of length len to the endpoint bEPIndex.

Hal4D13_ReadEndpoint(UCHAR bEPIndex, UCHAR * buf, USHORT len);
//Read data of length len from the endpoint bEPIndex..

Hal4D13_SetEndpointStatus(UCHAR bEPIndex, UCHAR bStalled);
//Modifies the DcEndpointStatusRegister

Hal4D13_GetEndpointStatusWInteruptClear(UCHAR bEPIndex);
//Read the DcEndpointStatusRegister to return important information like if the EP is stalled or full.

The Device Controller of the ISP1362 firmware is fully interrupt-driven. The flowchart of Interrupt Service Routine (ISR) is given in Figure 7.22

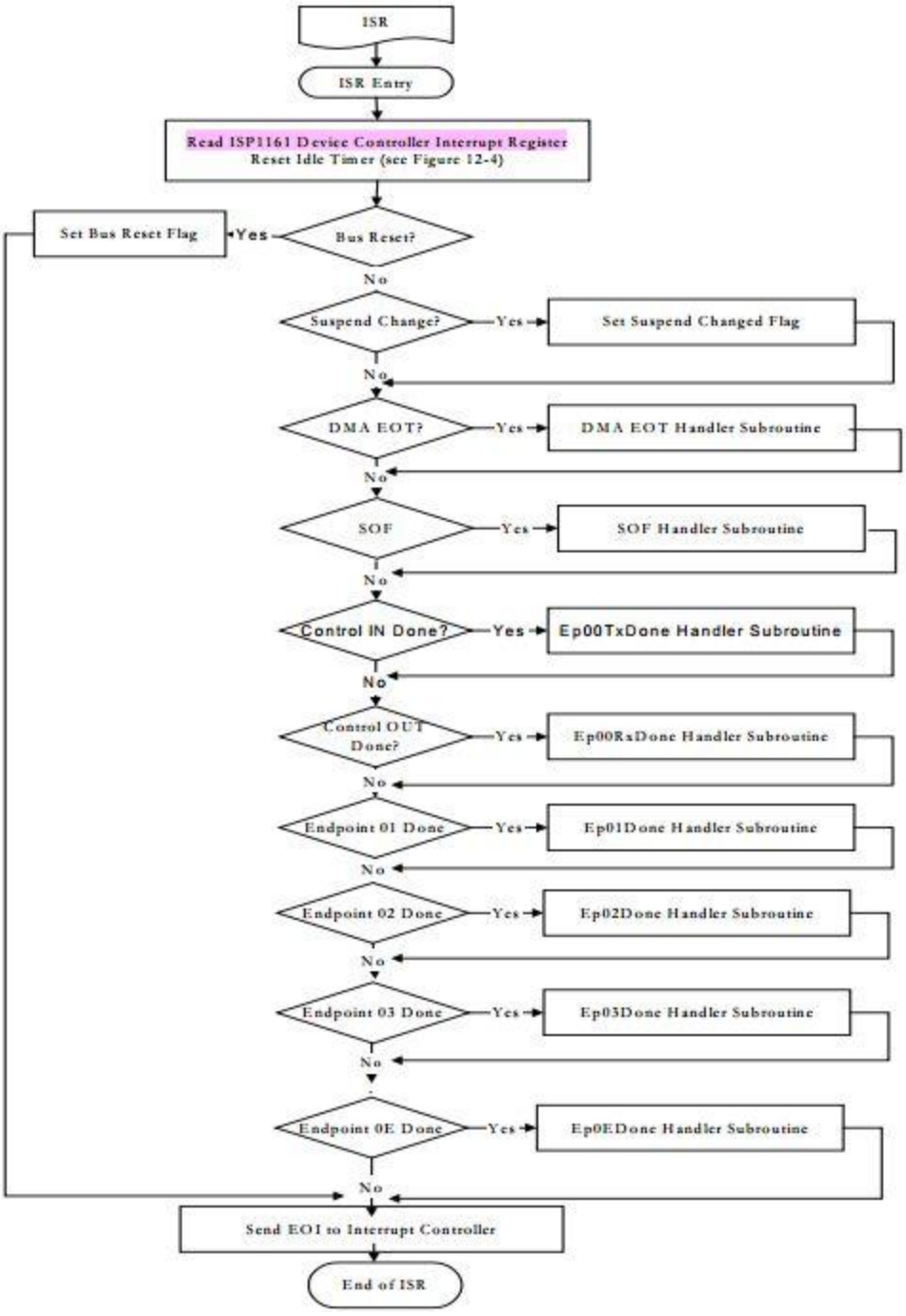

ISR
ISR Entry
Read ISP1161 Device Controller Interrupt Register
Reset Idle Timer (see Figure 12-4)
Bus Reset?
Yes
Set Bus Reset Flag
No
Suspend Change?
Yes
Set Suspend Changed Flag
No
DMA EOT?
Yes
DMA EOT Handler Subroutine
No
SOF
Yes
SOF Handler Subroutine
No
Control IN Done?
Yes
Ep00TxDone Handler Subroutine
No
Control OUT Done?
Yes
Ep00RxDone Handler Subroutine
No
Endpoint 01 Done
Yes
Ep01Done Handler Subroutine
No
Endpoint 02 Done
Yes
Ep02Done Handler Subroutine
No
Endpoint 03 Done
Yes
Ep03Done Handler Subroutine
No
Endpoint 0E Done
Yes
Ep0EDone Handler Subroutine
No
Send EOI to Interrupt Controller
End of ISR

Figure 2.22

7.10 USBmon Tool:

In this project the USBmon tool was used to debug the USB code by sniffing the packet transferred between the PC and the FPGA.
USBmon outputs a file that shows the log of USB packets and Errors.

7.10.1 Introduction

The name "usbmon" in lowercase refers to a facility in kernel which is used to collect traces of I/O on the USB bus.
This function is analogous to a packet socket used by network monitoring tools.

The usbmon reports requests made by peripheral-specific drivers to Host Controller Drivers (HCD). So, if HCD is buggy, the traces reported by usbmon may not correspond to bus transactions precisely.

7.10.2 How to use usbmon to collect raw text traces

Unlike the packet socket, usbmon has an interface which provides traces in a text format. This is used for two purposes. First, it serves as a common trace exchange format for tools while more sophisticated formats are finalized. Second, humans can read it in case tools are not available.

To collect a raw text trace, execute following steps.

A) Prepare

Mount debugfs (it has to be enabled in your kernel configuration), and load the usbmon module. The second step is skipped if usbmon is built into the kernel.

```
# mount -t debugfs none_debugs /sys/kernel/debug
# modprobe usbmon
#
```

Verify that bus sockets are present.

```
# ls /sys/kernel/debug/usb/usbmon
0s  0u  1s  1t  1u  2s  2t  2u  3s  3t  3u  4s  4t  4u
#
```

Now you can choose to either use the socket '0u' (to capture packets on all buses), and skip to step #3, or find the bus used by your device with step #2.

This allows filtering away annoying devices that talk continuously.

B) Find which bus connects to the desired device

Run "cat /proc/bus/usb/devices", and find the T-line which corresponds to the device. Usually you do it by looking for the vendor string. If you have many similar devices, unplug one and compare two /proc/bus/usb/devices outputs.

The T-line will have a bus number. Example:

```
T:  Bus=03 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0557 ProdID=2004 Rev= 1.00
S:  Manufacturer=ATEN
S:  Product=UC100KM V2.00
```

Bus=03 means it's bus 3.

C) Start 'cat'

```
# cat /sys/kernel/debug/usb/usbmon/3u > /tmp/1.mon.out
```
to listen on a single bus, otherwise, to listen on all buses, type:

```
# cat /sys/kernel/debug/usb/usbmon/0u > /tmp/1.mon.out
```

This process will be reading until killed. Naturally, the output can be redirected to a desirable location. This is preferred, because it is going to be quite long.

D) Perform the desired operation on the USB bus

This is where you do something that creates the traffic: plug in a flash key,
copy files, control a webcam, etc.

E) Kill cat

Usually it's done with a keyboard interrupt (Control-C).

At this point the output file (/tmp/1.mon.out in this example) can be saved, sent by e-mail, or inspected with a text editor.

7.10.3 Raw text data format

Two formats are supported currently: the original, or '1t' format, and
the '1u' format. The '1t' format is deprecated in kernel 2.6.21. The '1u' format adds a few fields, such as ISO frame descriptors, interval, etc.
It produces slightly longer lines, but otherwise is a perfect superset of '1t' format.

If it is desired to recognize one from the other in a program, look at the "address" word (see below), where '1u' format adds a bus number. If 2 colons are present, it's the '1t' format, otherwise '1u'.

Any text format data consists of a stream of events, such as URB submission, URB callback, submission error. Every event is a text line, which consists of whitespace separated words. The number or position of words may depend on the event type, but there is a set of words, common for all types.

7.10.4 Contents of the .mon file:

Here is the list of words, from left to right:

a. URB Tag. This is used to identify URBs, and is normally an in-kernel address of the URB structure in hexadecimal, but can be a sequence number or any other unique string, within reason.

b. Timestamp in microseconds, a decimal number. The timestamp's resolution depends on available clock, and so it can be much worse than a microsecond (if the implementation uses jiffies, for example).

c. Event Type. This type refers to the format of the event, not URB type.
Available types are: S - submission, C - callback, E - submission error.

d. "Address" word (formerly a "pipe"). It consists of four fields, separated by colons: URB type and direction, Bus number, Device address, Endpoint number.
Type and direction are encoded with two bytes in the following manner:
Ci Co Control input and output
Zi Zo Isochronous input and output
Ii Io Interrupt input and output
Bi Bo Bulk input and output
Bus number, Device address, and Endpoint are decimal numbers, but they may have leading zeros, for the sake of human readers.

e. URB Status word. This is either a letter, or several numbers separated by colons: URB status, interval, start frame, and error count. Unlike the "address" word, all fields save the status are optional. Interval is printed only for interrupt and isochronous URBs.

Start frame is printed only for isochronous URBs. Error count is printed only for isochronous callback events.

The status field is a decimal number, sometimes negative, which represents a "status" field of the URB. This field makes no sense for submissions, but is present anyway to help scripts with parsing. When an error occurs, the field contains the error code.

f. In case of a submission of a Control packet, this field contains a Setup Tag instead of a group of numbers. It is easy to tell whether the Setup Tag is present because it is never a number. Thus if scripts find a set of numbers in this word, they proceed to read Data Length (except for isochronous URBs). If they find something else, like a letter, they read the setup packet before reading the Data Length or isochronous descriptors.

g. Setup packet, if present, consists of 5 words: one of each for bmRequestType, bRequest, wValue, wIndex, wLength, as specified by the USB Specification 2.0. These words are safe to decode if Setup Tag was's'. Otherwise, the setup packet was present, but not captured, and the fields contain filler.

h. Number of isochronous frame descriptors and descriptors themselves.
If an Isochronous transfer event has a set of descriptors, a total number of them in an URB is printed first, then a word per descriptor, up to a total of 5. The word consists of 3 colon-separated decimal numbers for status, offset, and length respectively. For submissions, initial length is reported.
For callbacks, actual length is reported.

i. Data Length. For submissions, this is the requested length. For callbacks, this is the actual length.

j. Data tag. The usbmon may not always capture data, even if length is nonzero.
The data words are present only if this tag is '='.

k. Data words follow, in big endian hexadecimal format. Notice that they are not machine words, but really just a byte stream split into words to make it easier to read. Thus, the last word may contain from one to four bytes.
The length of collected data is limited and can be less than the data length reported in the Data Length word. In the case of an isochronous input (Zi) completion where the received data is sparse in the buffer, the length of the collected data can be greater than the Data Length value (because Data Length counts only the bytes that were received whereas the Data words contain the entire transfer buffer).

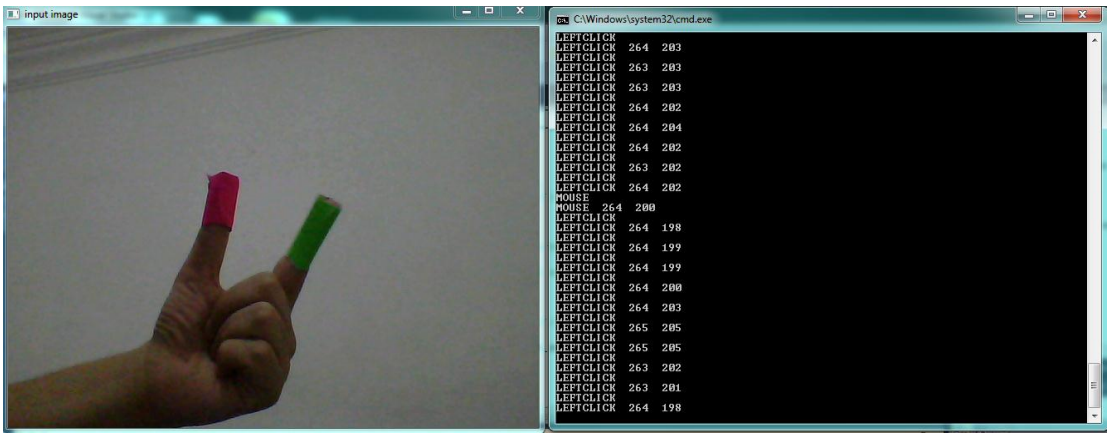
Chapter 8

Conclusions and Future Work

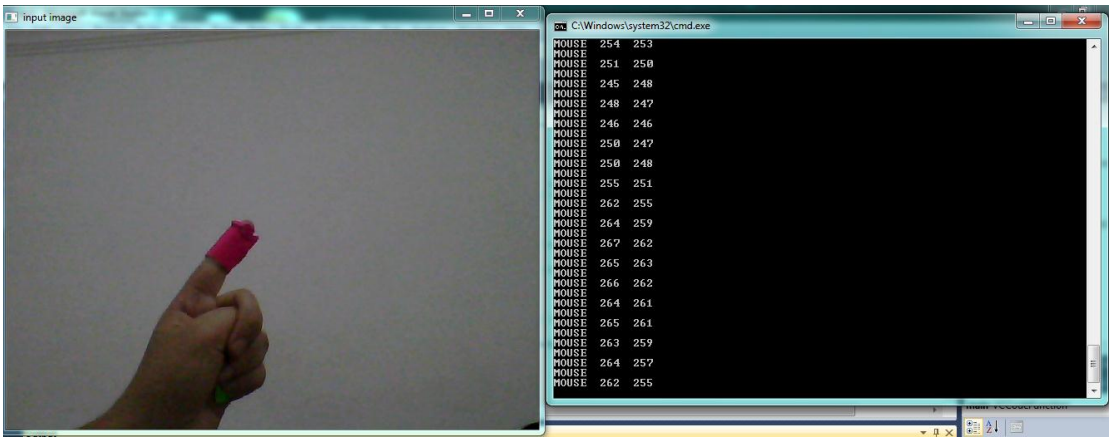
8.1 Summary and Conclusions

In this graduation project we proposed a design for a gesture recognition system based on ALTERA FPGA and tools. The idea was chosen for its motivational and educational properties.

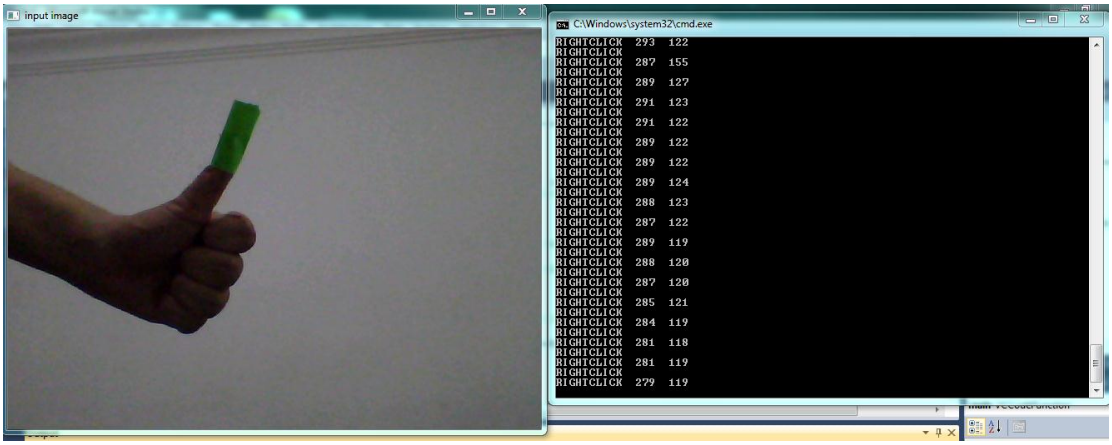
At first we produced a software only version of the project as a proof of concept then we built an offline version by porting the software code to NIOS II processor and loading the SDRAM memory with a frame.



(a) Left click gesture



(b) Mouse gesture



(C) Right click gesture

Fig 8.1 Different screenshots from the software version of the system

The next phase was integrating the offline version with the USB core and loading a set of frames to the memory, producing a fully working offline version.

The final and most difficult phase was combining this subsystem with the camera subsystem to produce the final system. We faced many problems in the final integration phase. Among the problems was synchronization between the system, we had to drop down the clock of the whole system to the PIXLCLK else the system would be unsynchronized and the frame will be distorted in memory.

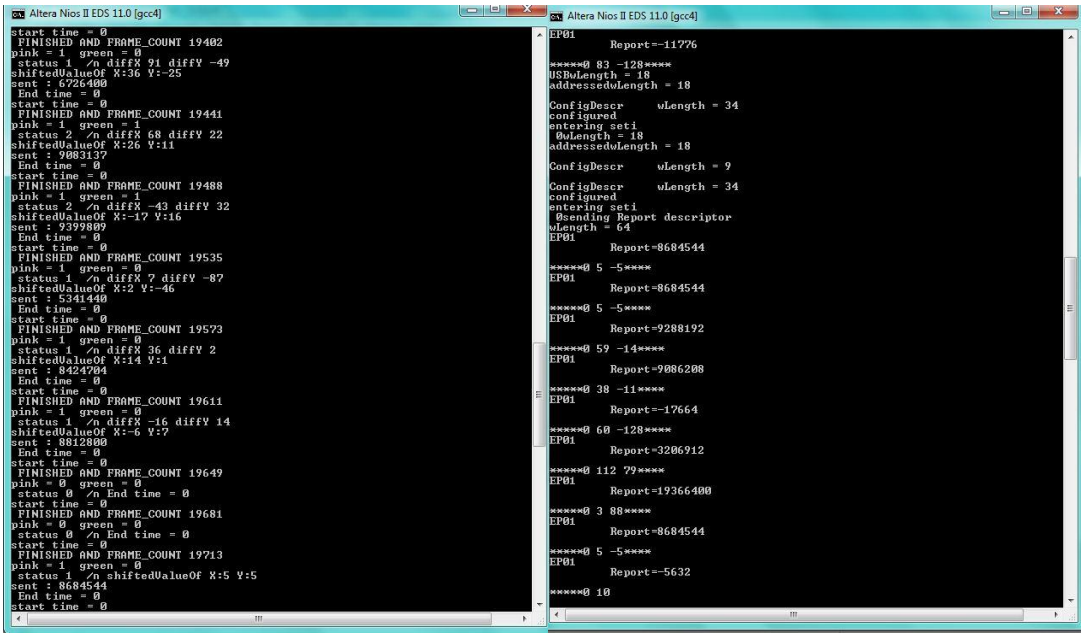


Fig 8.2 The final system running

Another complicated issue was memory system design, the SDRAM space was sufficient for our project (128 MBs) but forwarding all the memory access requests to a single chip would result in a system delay because the Avalon bus would schedule the time of using the chip among all the subsystems in our design so we took the decision of using all memory chips on our kit to reduce the access time latency as much as possible.

One has to be very careful when choosing the IP core for the memory controller of any offchip memory. One of two considerations has to be taken, either making sure that this IP core is tailored for the development kit or ensure that the timing is correct, usually the memory chips need the clock phased with a negative amount. Timing being incorrect in the memory chips leads to the famous memory verification failed problem when downloading the code to the memory.

It's worthy to be noted too that ALTERA tools are glitchy and crash often especially the NIOS II SBT, we have suffered along the year from continuous hang-ups and crashes. During the integration phase we had to run NIOS II from terminal as the NIOS II IDE crashes and behave randomly when more than one console is open. Sometimes we noticed that NIOS II core itself behaves randomly in some cases it needs a reset to act normally.

The results from the system are decent; still it needs a lot of enhancement and optimization to reach a fully real time system.

8.2 Future Work

As stated earlier the system requires optimization and many aspects of optimization can be thought of:

- Improving the worst case delay of system, slack of different clocks and performing advanced timing analysis would increase the system stability as a whole allowing running the system on a higher frequency.
- Depending more on the custom hardware to offload processing from hardware. DMA core can be used to allow custom hardware modules to interact with memory directly.
- Redesigning the software code tree, currently we rely on the OpenCV tree structure and we noted that there are too much unneeded function calls which requires pushing to stack and popping from it which affects the system response significantly.
- The only gesture supported now is the mouse gesture, in the future many features and more gestures can be supported.

9. References

[1] http://www.beyondlogic.org/usbnutshell/usb5.shtml .
[2] http://libusb.sourceforge.net/api-1.0/structlibusb_device_descriptor.html#a9c3a91102d3d53d9414d0dda0191c5ab .
[3] http://www.usb.org/developers/defined_class .
[4] http://www.libusb.org/browser/examples/lsusb.c .
[5] http://www.linux-usb.org/ .
[6]http://www.cs.indiana.edu/~bpisupat/work/oc_usb.c .
[7]http://www.cs.indiana.edu/~bpisupat/work/usb.html .
[8]http://lwn.net/Kernel/LDD3/ .
[9]http://www.algorithm-forge.com/techblog/2009/12/using-libusb-to-write-a-linux-usb-driver-or-the-arexx-tl-500-part-i/ .
[10]http://www.freesoftwaremagazine.com/articles/drivers_linux .
[11]http://www.linuxjournal.com/article/7353?page=0,2 .
[12]http://alterauserforums.net/forum/showthread.php?p=114241 .
[13]http://lists.alioth.debian.org/pipermail/sane-devel/2010-February/026178.html .
[14]http://stackoverflow.com/questions/6554863/migrating-c-code-that-talks-to-a-usb-glucometer-device-from-linux-to-android .
[15]http://rowsandcolumns.blogspot.com/2011/02/read-from-magtek-card-swipe-reader-in.html
[16]http://www.linuxjournal.com/article/8145 .
[17]http://libhid.alioth.debian.org/ .
[18]http://www.silabs.com/Support%20Documents/TechnicalDocs/AN249.pdf .
[19]http://www.tracesystemsinc.com/USB_Tutorials_web/USB/B1_USB_Classes/Books/A3_A_Closer_Look_at_HID_Class/sbook2.htm .
[20]http://www.microchip.com/forums/tm.aspx?m=340898 .
[21]http://www.usb.org/developers/devclass_docs/Hut1_11.pdf .
[22]http://www.freebsddiary.org/APC/usb_hid_usages.php .
[23]http://msdn.microsoft.com/en-us/library/windows/hardware/ff539946(v=vs.85).aspx .
[24]http://www.ladyada.net/learn/diykinect/ .
[25]http://www.jespersaur.com/drupal/book/export/html/21 .
[26]http://lists.alioth.debian.org/pipermail/sane-devel/2010-February/026183.html .
[27]http://www.cs.indiana.edu/~bpisupat/work/usb.html .
[28]http://www.algorithm-forge.com/techblog/2009/12/using-libusb-to-write-a-linux-usb-driver-for-the-arexx-tl-500-part-i/ .
[29]http://www.frank-zhao.com/cache/hid_tutorial_1.php .
[30]http://www.usbmadesimple.co.uk/ums_5.htm .
[31]http://www.usblyzer.com/usb-human-interface-device-hid-class-decoder.htm .
[32]http://www.cygnal.org/ubb/Forum9/HTML/001381.html .
[33]http://msdn.microsoft.com/en-us/library/windows/hardware/ff537057(v=vs.85).aspx .
[34]http://code.google.com/p/nios2-usb/ .
[35]http://www.linux-usb.org/USB-guide/x75.html#FIGDESCRIPTOR .

[36]http://blogs.msdn.com/b/usbcoreblog/archive/2009/10/31/how-does-usb-stack-enumerate-a-device.aspx .
[37]http://libusb.sourceforge.net/doc/function.usbreset.html .
[38]http://www.lvr.com/usbcenum.htm .
[39]http://www.clearchain.com/blog/posts/resetting-the-usb-bus-under-linux .
[40]http://libusb.sourceforge.net/api-1.0/group__dev.html#_details .
[50]http://code.google.com/p/nios2-usb/source/browse/.
[51]http://www.circuitsathome.com/products-page/arduino-shields/ .
[52]http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1288239935 .
[53]http://www.practicalarduino.com/projects/virtual-usb-keyboard.
[54]http://www.obdev.at/products/vusb/index.html.
[55]http://kerneltrap.org/mailarchive/linux-usb-devel/2007/7/4/339219 .
[56]http://www.mail-archive.com/linux-usb
[57]http://kerneltrap.org/mailarchive/linux-usb-devel/2007/7/4/339219 .
[58]http://kerneltrap.org/mailarchive/linux-usb-devel/2007/7/4/339216 .
[59]http://www.mjmwired.net/kernel/Documentation/usb/usbmon.txt .
[60] PHILIPS. *"ISP1362 Embedded Programming Guide Rev: 0.9"*
[61] PHILIPS. *"ISP1362 Single-chip Universal Serial Bus On-The-Go controller"*
[62] XAVIER CALBET. *"Writing device drivers in Linux: A brief tutorial"*
[63] SEBASTIEN JAN, MICHAEL OPDENACKER, THOMAS PETAZZONI.
"Linux kernel and driver development training"
[64]MULI BEN YEHUDA. *"Introduction to Linux Device Drivers Recreating Life One Driver At a Time"*
[65]ABBOTT. *"Kernel Modules and Device Drivers*"
[66]STUART ALLMAN. *"Using the HID class eases the job of writing USB device drivers*"
[67]SILICON LABS. *"Human Interface Device Tutorial"*
[68]USB.ORG. *"Device Class Definition for Human Interface Devices (HID)*"
[69]CRAIG PEACOCK . *"USB in a Nutshell"*
[70]USB.ORG. *"Universal Serial Bus 2.0 Specification*"
[71]http://www.pages.drexel.edu/~nk752/tutorials.html
[72] http://opencv.willowgarage.com/documentation/ .
[73]http://www.catenary.com/howto/vicdemo.html .
[74]http://opencv-users.1802565.n2.nabble.com/demo-of-handGestures-and-detection-of-fingertips-td3918028.html .
[75]http://stackoverflow.com .
[76] http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html
[77] http://opencv-users.1802565.n2.nabble.com
[78] http://opencv.wordpress.com/
[79]http://cplus.about.com/od/codelibraryforc/A_Library_of_Software_written_in_C_with_full_source_code.htm
[80] http://www.pages.drexel.edu/~nk752/tutorials.html
[81] http://dasl.mem.drexel.edu/~noahKuntz/openCVTut7.html
[82] http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/

[83] http://www.yafla.com/yaflaColor/ColorRGBHSL.aspx
[84] https://github.com/liquidmetal/AI-Shack--Tracking-with-OpenCV
[85] ALTERA COROPORATION. *"Specifications Avalon Interface"*
[86] ALTERA COROPORATION. *"Nios II Command-Line Tools*"
[87] ALTERA COROPORATION. *"Introduction to Quartus II*"
[88] ALTERA COROPORATION. *"Creating Multiprocessor Nios II Systems Tutorial* "
[89] ALTERA COROPORATION. *"Nios II Software Developer's Handbook*"
[90] ALTERA COROPORATION. *"Getting Started from the Command Line"*
[91] ALTERA COROPORATION. *"Quartus II Programmer"*
[92] ALTERA COROPORATION. *"Release Notes and Errata MegaCore IP Library"*
[93] ALTERA COROPORATION. *"Nios II Embedded Design Suite Release Notes and Errata"*
[94] ALTERA COROPORATION. *"Tutorial Nios II Hardware Development"*
[95] ALTERA COROPORATION. "User Guide Embedded Peripherals IP www.atlera.com .
[96] ALTERA COROPORATION. *"Getting Started with Quartus II Simulation Using the ModelSim-Altera Software"*
[97] ALTERA COROPORATION. *"Nios II C2H Compiler*"
[98] LUCAS BENDA. *"Hardware acceleration for image processing"*
[99] PINAR DUYGULU. *"Tutorial on Image Processing"*
[100] MATTIAS KARLSSON. *"Implementation of algorithms on FPGAs"*
[101] X. ZABULISY , H. BALTZAKISY , A. ARGYROSZY. "*Vision-based Hand Gesture Recognition for Human-Computer Interaction"*
[102]ALTERA COROPORATION. *"Nios II C2H Compiler*"
[103] ALTERA COROPORATION. *"Hardware Acceleration and Coprocessing*"
[104] ALTERA COROPORATION. *"SignalTap II with Verilog Designs"*
[105] ALTERA COROPORATION. *"Design Debugging Using the SignalTap II Logic Analyzer"*
[106] ALTERA COROPORATION *"Nios II Flash Programmer*"