Examensarbete

An FPGA Based Software/Hardware Codesign for Real Time Video Processing

A Video Interface Software and Contrast Enhancement Hardware Codesign Implementation using Xilinx Virtex II Pro FPGA

Master thesis performed in Institute for Integrated System, Technical University Munich, Germany

by

Jian Wang

Report number: LITH-ISY-EX--06/3756--SE
Linköping March 2006
An FPGA Based Software/Hardware Codesign for Real Time Video Processing

A Video Interface Software and Contrast Enhancement Hardware Codesign Implementation using Xilinx Virtex II Pro FPGA

Master thesis in Computer Engineering, Institutionen för systemteknik at Linköping Institute of Technology

by

Jian Wang
LiTH-ISY-EX--06/3756--SE

Supervisor: Dr.-Ing. Walter Stechele
Institute for Integrated System,
Technical University Munich, Germany

Examiner: Prof. Dake Liu
Division of Computer Engineering,
Institutionen för systemteknik,
Linköping University, Sweden

Linköping March 2006
An FPGA Based Software/Hardware Codesign for Real Time Video Processing

Jian Wang

Xilinx Virtex II Pro FPGA with integrated PowerPC core offers an opportunity to implementing a software and hardware codesign. The software application executes on the PowerPC processor while the FPGA implementation of hardware cores coprocess with PowerPC to achieve the goals of acceleration. Another benefit of coprocessing with the hardware acceleration core is the release of processor load. This thesis demonstrates such an FPGA based software and hardware codesign by implementing a real time video processing project on Xilinx ML310 development platform which is featured with a Xilinx Virtex II Pro FPGA. The software part in this project performs video and memory interface task which includes image capture from camera, the store of image into on-board memory, and the display of image on a screen. The hardware coprocessing core does a contrast enhancement function on the input image. To ease the software development and make this project flexible for future extension, an Embedded Operating System MontaVista Linux is installed on the ML310 platform. Thus the software video interface application is developed using Linux programming method, for example the use of Video4Linux API. The last but not the least implementation topic is the software and hardware interface, which is the Linux device driver for the hardware core. This thesis report presents all the above topics of Operating System installation, video interface software development, contrast enhancement hardware implementation, and hardware core’s Linux device driver programming. After this, a measurement result is presented to show the performance of hardware acceleration and processor load reduction, by comparing to the results from a software implementation of the same contrast enhancement function. This is followed by a discussion chapter, including the performance analysis, current design’s limitations and proposals for improvements. This report is ended with an outlook from this master thesis.

Keywords
Hardware and software codesign, embedded Linux, video4linux, custom IPcore, noise reduction, contrast enhancement
Abstract

Xilinx Virtex II Pro FPGA with integrated PowerPC core offers an opportunity to implementing a software and hardware codesign. The software application executes on the PowerPC processor while the FPGA implementation of hardware cores coprocess with PowerPC to achieve the goals of acceleration. Another benefit of coprocessing with the hardware acceleration core is the release of processor load. This thesis demonstrates such an FPGA based software and hardware codesign by implementing a real time video processing project on Xilinx ML310 development platform which is featured with a Xilinx Virtex II Pro FPGA. The software part in this project performs video and memory interface task which includes image capture from camera, the store of image into on-board memory, and the display of image on a screen. The hardware coprocessing core does a contrast enhancement function on the input image. To ease the software development and make this project flexible for future extension, an Embedded Operating System MontaVista Linux is installed on the ML310 platform. Thus the software video interface application is developed using Linux programming method, for example the use of Video4Linux API. The last but not the least implementation topic is the software and hardware interface, which is the Linux device driver for the hardware core. This thesis report presents all the above topics of Operating System installation, video interface software development, contrast enhancement hardware implementation, and hardware core’s Linux device driver programming. After this, a measurement result is presented to show the performance of hardware acceleration and processor load reduction, by comparing to the results from a software implementation of the same contrast enhancement function. This is followed by a discussion chapter, including the performance analysis, current design’s limitations and proposals for improvements. This report is ended with an outlook from this master thesis.

Keywords
Hardware and software codesign, embedded Linux, video4linux, custom IPcore, noise reduction, contrast enhancement
Acknowledgements

Reaching the point of my master thesis’ publishing is never my work alone. There are many people that deserve my thankfulness, for all they have done for me and my work.

- My supervisor Dr.-Ing. Walter Stechele. Thanks for offering me this opportunity to do my master thesis in your research group, and also for your academic directions and support on my work.

- My examiner Prof. Dake Liu. Thank you for paying attention on my thesis work and exam it for my graduation.

- Dr.-Ing. Thomas Wild. Thank you for guiding me into Linux world.

- Univ.-Prof. Dr. sc.techn. Andreas Herkersdorf, the head of Institute for Integrated Systems(LIS), TUM. Thank you for organizing a very nice research environment in LIS where I really enjoy my work.

- My group mates Florian Müller, Johannes Zeppenfeld, Mayank Gupta, Ingmar Cramm, Traboulsi Shadi. Thank you all for the excellent team cooperation.

- All other staffs and students at LIS that I have chance to know. Thank you all for giving me a good time in TU Munich.

- The last but definitely not the least. Thank my parents, Wang Qingyue and Yang Zhongxia, for bring me up, my sister Wang Wenjing for a constant support on me. And thank my girl friend Zhang Yanan’s love. You all make me never feel lonely even I’m living so far from you.

Jian Wang
Linköping, Sweden. March 2006
Contents

1 INTRODUCTION ............................................................................................................. 1

2 OVERVIEW ..................................................................................................................... 3

2.1 TARGET PLATFORM .................................................................................................... 3
2.1.1 Xilinx ML310 Board ............................................................................................ 3
2.2 SYSTEM OVERVIEW .................................................................................................. 5
2.2.1 Architecture Platform .......................................................................................... 5
2.2.2 Software .............................................................................................................. 5
2.2.3 Hardware ............................................................................................................ 7
2.3 DATA FLOW ............................................................................................................. 8
2.3.1 Data Format ........................................................................................................ 8
2.3.2 Data Flow .......................................................................................................... 8

3 EMBEDDED OS: MONTAVISTA LINUX ......................................................................... 11

3.1 HARDWARE SUPPORT ............................................................................................. 11
3.1.1 Xilinx ML310 PCI Reference Design .................................................................... 12
3.2 BOARD SUPPORT PACKAGE .................................................................................. 13
3.3 LINUX CONFIGURATION ....................................................................................... 14

4 SOFTWARE: VIDEO & MEMORY INTERFACES .......................................................... 15

4.1 VIDEO4LINUX API ................................................................................................... 15
4.1.1 What is Video4Linux? ........................................................................................... 15
4.1.2 Why we use Video4Linux? .................................................................................. 16
4.2 IMAGE CAPTURE ..................................................................................................... 16
4.2.1 Capture Mechanism ............................................................................................ 16
4.2.2 Linux ioctl Command ........................................................................................ 17
4.2.3 Camera Initialization ......................................................................................... 18
4.2.4 Image Capture .................................................................................................... 21
4.3 DISPLAY .................................................................................................................. 21
4.3.1 Gray to RGB conversion ..................................................................................... 22
4.3.2 Framebuffer display ........................................................................................... 22
4.4 KEYBOARD HIT DETECTION ................................................................................... 22

5 HARDWARE: CONTRASTENGINE .......................................................................... 25

5.1 FUNCTIONAL DESCRIPTION ................................................................................... 25
5.1.1 Noise Reduction .................................................................................................. 25
5.1.2 Contrast Enhancement ....................................................................................... 26
5.2 HARDWARE IMPLEMENTATION .............................................................................. 28
5.2.1 IPIF to PLB ......................................................................................................... 29
5.2.2 Memory Arbiter ................................................................................................ 29
5.2.3 Config Register .................................................................................................. 31
5.2.4 Master Unite ...................................................................................................... 33

6 SW/HW INTERFACE: LINUX DEVICE DRIVER ........................................................... 37

6.1 DEVICE FILE AND MAJOR-MINOR NUMBERS OF DEVICE ..................................... 37
6.2 DRIVER MODULE LOAD AND UNLOAD ............................................................... 37
6.3 DEVICE OPEN AND RELEASE ............................................................................... 38
6.4 MAPPING PHYSICALLY CONTINUOUS MEMORY .................................................. 39
6.5 WRITE CONFIG REGISTER ................................................................................... 39
6.6 SEND START SIGNAL ............................................................................................ 40
6.7 WAIT AND RESPONSE TO INTERRUPT FROM HW ............................................ 40
7 RESULTS .......................................................................................................................................... 41
  7.1 FRAME RATE .............................................................................................................................. 41
  7.2 FPGA USAGE ............................................................................................................................. 41
  7.3 PERFORMANCE MEASUREMENTS ........................................................................................... 41
8 DISCUSSION .................................................................................................................................... 45
  8.1 MEMORY ACCESS BOTTLENECK ............................................................................................... 45
  8.2 USB1.1 INTERFACE LIMITATION ........................................................................................... 45
9 OUTLOOK .......................................................................................................................................... 47
APPENDIXES ......................................................................................................................................... 49
  APPENDIX A: LINUX COMMANDS AND PROGRAMMING SKILL ........................................................... 49
  APPENDIX B: DESIGN PATCH AND TRICK ........................................................................................ 51
  APPENDIX C: VIDEO 4 LINUX API .................................................................................................... 53
FIGURE INDEX ....................................................................................................................................... 63
TABLE INDEX ......................................................................................................................................... 65
BIBLIOGRAPHY ...................................................................................................................................... 67
1 Introduction

This thesis is part of the AutoVision project in Institute of Integrated System (LIS) at Technical University Munich. The AutoVision project is towards a future video based driver assistant which can recognize different objects on the road in real time and thus to improve security. This target reorganization algorithm is done by using MPEG-7 standard for video analysis. In contrast to the previous MPEG standards like MPEG-1/2/4, which are widely used in digital TV, MPEG-7 is a standard for describing multimedia contents. In AutoVision project, the overall algorithm is separated in two levels; a high level application code with much complexity and a low pixel level operations which iteratively apply a same calculation to plenty of pixels from the video image sequence. This separation best suits a platform architecture of software and hardware coprocessing; the high level application code will be executed on a general CPU, and the low level operations on pixels will be targeted on hardware accelerators.

Xilinx has released a Virtex II Pro FPGA family which offers a possibility to design a FPGA based hardware and software coprocessing project. This FPGA has up to four PowerPC processors integrated for running software application, while the surrounding FPGA fabric can be configured as hardware accelerators. This codesign architecture forms the first concept of AutoVision project.

Another concept behind the AutoVision project is the FPGA’s dynamically and partially self-reconfiguration. From AutoVision project’s specification, the pixel level operations still require algorithm flexibility due to different road environments like the scenarios listed in Figure 1-1. For example, a Shape Engine is required when driving in normal highway; at the tunnel entrance, the algorithm is changed to a contrast enhancement on the dark entrance area plus edge detection; when inside the tunnel, we should use another algorithm of luminance segmentation because of the low luminance environment. To make the hardware accelerators flexible to execute these pixel level operations, the second idea of reconfigurable computing comes up. The goal of the FPGA’s reconfiguration in AutoVision project is to achieve a dynamically and partially self-reconfiguration, which means part of the FPGA can be reconfigured at run-time by the platform itself without any host PC.

As part of the AutoVision project, this thesis demonstrates the software and hardware codesign concept on a Xilinx Virtex II Pro FPGA. The software part executing on a
PowerPC processor manages the video and memory interfaces. In the software side, we also choose to use an Embedded Linux to ease our software development. The hardware part is an FPGA acceleration core doing the contrast enhancement. This hardware acceleration engine will be configured to the FPGA in the second scenario of tunnel entrance in Figure 1-1. It does a contrast enhancement on the sub-area of tunnel entrance.

This report is organized as start from an overview of our prototyping platform, system architecture and video data format and flow in Chapter2.

Chapter3 discusses about the Embedded Operating System, MontaVista Linux, including its installation on our hardware platform and the feature configuration.

In Chapter4 the software application is documented. Topics in this chapter include an introduction of the Linux API Video4Linux used in the project, capturing images to memory, and framebuffer display. A Linux programming trick of keyboard hit detection to jump out our image sequence processing loop is put at the end of this chapter.

After the software part, Chapter5 focuses on the contrast enhancement core ContrastEngine’s hardware implementation. It starts from a functional description and then followed by the FPGA hardware implementation, both structure building and each sub-modules’ implementation is documented.

In Chapter6, the Linux device driver for ContrastEngine is presented. The operations supported by the device driver to control ContrastEngine are discussed in this chapter’s sub-sections.

Chapter7 and 8 include the test result from this thesis project and the discussion on its limitations and improvement proposals.

Some useful references as the Linux command, the tricks and patches, and Video4Linux API document are put in the appendixes.
2 Overview

This thesis project is based on the concepts of a software and hardware codesign on modern FPGA platform. The reason for using a software/hardware coprocessing architecture is that it takes advantage of both sides. Software solution can offer more flexibility and a low cost implementation, while the hardware implementation is faster in processing. Since the cost and speed issues are the most critical topics in embedded system, the combination of software and hardware to improve system performance attracts more and more engineers.

2.1 Target Platform

The target platform for this thesis project consists of a Xilinx ML310 FPGA board [1] and several video peripheral devices for video input and output including a Logitech 4000 Pro USB webcam, a Matrox PCI graphic card and a PC monitor. The interconnection between ML310 board and the video peripherals is through the very common PC interface supported by ML310; USB port for connecting webcam and PCI slot for the graphic card. The top-level hardware connection setup is showed in Figure 2-1.

![Figure 2-1 Develope Platform Setup](image)

2.1.1 Xilinx ML310 Board

The Xilinx ML310 board belongs to Xilinx Virtex II-Pro FPGA development platform series. The “heart” of ML310 board, as other boards of the same series, is a Virtex II-Pro FPGA chip, which has two built-in PowerPC processors surrounded by FPGA fabric. as showed in Figure 2-2. This architecture offers a big flexibility for improving embedded system’s performance by a method of software and hardware codesign. The software tasks can be executed on the PowerPC processor, while the FPGA is used for hardware acceleration as coprocessors.
The Xilinx ML310 board makes use of the Virtex II-Pro chip with other on-board components such as DDR memory, Ethernet, PCI slots, Audio, and multiple common PC interfaces like USB and PS/2. Detail of these on-board components and their connections can be seen in Figure 2-3, a block diagram of ML310 board from Xilinx [1]. In AutoVision Project, we use the USB (standard 1.1) port for video input from a webcam, the on-board DDR memory for caching the grabbed video stream, and the PCI slot (the 5.0V PCI slot) for video display through a graphic card.
2.2 System Overview

For this software and hardware codesign project, our software part performs the video interface task, including capturing image sequence from camera, storing images into the DDR memory, and display. The hardware part, the FPGA coprocessor core, operates on the image stored in the memory to perform the image processing task.

2.2.1 Architecture Platform

The FPGA fabric for coprocessor implementation and the PowerPC for running software communicate through a Processor Local Bus (PLB), a bus connection fashion used by Virtex II-Pro FPGA. Other peripherals such as DDR memory, camera and graphic card are also connected to PLB bus through different kind of bridges or controllers. Since those bridges are not interesting points here, we can simplify the system architecture diagram by ignoring them, as illustrated in Figure 2-4.

![Figure 2-4 SW/HW Codesign System Architecture](image)

2.2.2 Software

The software runs on one of the two PowerPCs on ML310 to perform the video interface task. To ease the implementation of video peripherals, the camera and the graphic card, a Real Time Operating System (OS), MontaVista Linux is chosen and installed on the target Xilinx ML310 platform. As the drivers for all video peripherals are supported by the Linux OS, we do not need to write them ourselves. Moreover, the installation of Linux also benefits the software programming; the software can use all libraries from Linux and be developed no different than a usual Linux programme on other computing
platform, take a very common example, Linux on Desktop PC. The last but not the least advantage of using Linux comes from the worldwide contribution to Linux development for its open source property. We can get help and even the code from other developers for free under General Public Licence (GPL).

The software video interface task is performed in three steps; image capture, store, and display. Meanwhile, it should leave an intermediate step between image store and display for image processing. This executing flow is demonstrated in Figure 2-5. In this software/hardware codesign project, the image processing step is realized by a hardware acceleration core, the ContrastEngine.

![Figure 2-5 Video Interface Software Flow Chart](image)

Although the Linux OS saves our development time on video device driver and software programming, it introduces another crucial topic at software side, which is the Linux device driver for the hardware coprocessor core, ContrastEngine. We need such a driver to control ContrastEngine from Linux user space. As any other Linux device driver, this ContrastEngine driver can be viewed as the software and hardware interface; it stands in-between Linux software and ContrastEngine, as showed in Figure 2-6. More details about the ContrastEngine driver is discussed in chapter1.
2.2.3 Hardware
ContrastEngine, a coprocessor IP core to perform contrast enhancement operation on an input image, composes the hardware part in this codesign project. The initial intention of using this hardware coprocessor core is not only doing hardware acceleration but also releasing the CPU’s computing load.

How does the ContrastEngine work with other parts in the system? To illustrate this, let’s generally view it as a black box which communicates with outside world through a start signal, an interrupt signal and a data transfer bus, as showed in Figure 2.7. When the software runs into the step of image processing, it sends the start signal to ContrastEngine to start it, and then the software part goes to sleep waiting for the completion of ContrastEngine processing. During this time interval, the ContrastEngine read image data from memory, do computations, and then write it to the memory. Above is a simplified process description for a general view. In the real HW implementation of ContrastEngine, more aspects should be taken into consideration. This can be found in Chapter5, Hardware: ContrastEngine.
2.3 Data Flow

After the introduction of the overall system architecture, this part discusses the image data format and the data flow inside the system.

2.3.1 Data Format

As one of the specifications from the AutoVision project, the image to be processed should be greyscale image with 8 bits per pixel. Greyscale means only luminance value (Y-value) of the pixels from an image is of interest in this project. The way how the pixel value is stored in the memory can be seen in Figure 2-8.

![Image Data Format and Storage](image)

Nowadays most of the non-professional cameras in the market do not support a greyscale capture, they use a data palette of YUV. By simply copying the Y-value and discarding the rest colour information from the YUV-format image, we can get the greyscale input image. This is the way we use in this project with the Logitech 4000 Pro webcam.

2.3.2 Data Flow

The image data flow in our codesign system is illustrated by Figure 2-9. Under the flow control from PowerPC software, image data is captured by camera and delivered to the on-board DDR memory in the Input Image Buffer area. Then the ContrastEngine starts to work, it read the input image, processes it, and then writes the result image back into the memory in a Result Image Buffer area. In the real hardware implementation, image data transfer through buses, bridges, and device controllers.
Figure 2-9 Image Data Flow
3 Embedded OS: MontaVista Linux

In the software overview, we talked about the reason why we choose to use the OS Linux, that is, to ease the installation of video peripherals and the software development. Furthermore, this also inherits a lot of other merits of an embedded real time operating system, such as multi-processing with a high level concurrency between applications and reliability for the time critical tasks. We will get more benefit from the OS in the future extension of this project, when the software side becomes more complex than the current one.

MontaVista is one of the third-party companies that cooperate with Xilinx to support OS on Xilinx FPGAs. Now the MontaVista Linux and its associated development tools support the PowerPC processor inside the Virtex-II Pro FPGA.

3.1 Hardware Support

Although the Linux runs on the PowerPC, it needs hardware support for communication with the outside world, for example, the access of the DDR memory and the Ethernet. This hardware support for Linux consumes the FPGA fabric. See Figure 3-1 below.

![Figure 3-1 Linux Hardware Support](image)

Figure 2-1 is a demonstration of the fact that the Linux desired hardware support consumes the FPGA fabric around the PowerPC processor on Virtex-II Pro chip. In the real installation of MontaVista Linux on Xilinx ML310 platform, the FPGA hardware support is a more complex design. Xilinx provides its customers with a ML310 targeted
reference design [2], which includes the hardware support for a Linux installation. And more, this reference design supports PCI components on the board. This is very important to our project, because, as discussed in the system overview, the graphic card we use for image display is connected to the PCI slot, and the webcam for image capture is connect via USB port through a Southbridge to the PCI bus; our video peripheral devices are all directly or indirectly connected to the PCI bus. The following section is an overview of this Xilinx ML310 PCI reference design.

3.1.1 Xilinx ML310 PCI Reference Design

Xilinx ML310 PCI reference design is a complete EDK project which includes the hardware support for a Linux installation in Xilinx ML310 platform. Table 3-1 lists all the IP cores in this reference design. One note-worthy thing here is that this reference design is built for a Xilinx software environment of EDK6.2. If you were using Xilinx EDK software of version newer than 6.2, for example, 6.3 or 7.1i, you need to apply a patch on this design project. Find the patch information in Appendix B: Tricks and Patches.

Table 3-1 Xilinx ML310 PCI Reference Design IP List [1]

<table>
<thead>
<tr>
<th>ML310 XPS Design</th>
<th>XPS PCI Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLB_BRAM</td>
<td>v1_00_a</td>
</tr>
<tr>
<td>OPB_INTC</td>
<td>v1_00_c</td>
</tr>
<tr>
<td>OPB_IIC</td>
<td>v1_01_c</td>
</tr>
<tr>
<td>OPB_SYSACE</td>
<td>v1_00_b</td>
</tr>
<tr>
<td>OPB_UART16550</td>
<td>v1_00_c</td>
</tr>
<tr>
<td>PLB2OPB_BRIDGE</td>
<td>v1_01_a</td>
</tr>
<tr>
<td>OPB2PLB_BRIDGE</td>
<td>v1_00_c</td>
</tr>
<tr>
<td>PLB_BRAM_IF_CNTL</td>
<td>v1_00_b</td>
</tr>
<tr>
<td>PLB_DDR</td>
<td>v1_11_a</td>
</tr>
<tr>
<td>PPC405</td>
<td>v2_00_c</td>
</tr>
<tr>
<td>OPB_V20</td>
<td>v1_10_b</td>
</tr>
<tr>
<td>PLB_V34</td>
<td>v1_02_a</td>
</tr>
<tr>
<td>OPB_SPI</td>
<td>v1_00_b</td>
</tr>
<tr>
<td>OPB_GPIO</td>
<td>v3_01_a</td>
</tr>
<tr>
<td>OPB_PCI</td>
<td>v1_00_c</td>
</tr>
<tr>
<td>OPB_PCI_ARBITER</td>
<td>v1_00_a</td>
</tr>
<tr>
<td>DSOCM_V10</td>
<td>v2_00_a</td>
</tr>
<tr>
<td>ISOCM_V10</td>
<td>v2_00_a</td>
</tr>
<tr>
<td>DSBRAM_IF_CNTL</td>
<td>v3_00_a</td>
</tr>
<tr>
<td>ISBGRAM_IF_CNTL</td>
<td>v3_00_a</td>
</tr>
<tr>
<td>DSOCM_BRAM</td>
<td>v1_00_a</td>
</tr>
</tbody>
</table>
In order to inform the Linux kernel part about the hardware configurations which we have in the platform, such as the physical address of different IP cores and the interrupt register numbers, we use the Board Support Package (BSP). The BSP can be automatically generated from Xilinx EDK tool according to the current hardware specifications. The Linux kernel only needs to read from this package to get hardware information. This is done by putting this BSP into the Linux kernel source before the configuration and compilation of Linux. After this work, we will have a special Linux porting to our hardware platform. The design flow of installing such a Linux is illustrated in Figure 3-2 below.

### 3.2 Board Support Package

Figure 3-2 Linux Installation Flow
3.3 Linux Configuration

One significant characteristic of embedded Linux is its configurable ability. The reason why we need to configure an embedded Linux is that this kind of OS is usually targeted on an application specific embedded system, the task of the system could be predicted, thus we only need to keep the features required by the system in order to minimize the Linux kernel size. This is done by the Linux configuration. In this thesis project, the features other than a minimum configuration setting of the Linux are listed as below.

- Console
  * Matrox accelerator
  * Millennium I/II
- Multimedia
  * Video4Linux
- USB
  * Philips camera driver

The Matrox option opens the support of the graphic card. The Video4Linux option in Multimedia is required for software to control video devices. The Video4Linux is discussed in Chapter 4 Software: Video & Memory Interfaces. The last custom configure setting, the Philips camera driver is included to support our Logitech 4000 Pro webcam. If you were using the same webcam and MontaVista Linux Preview kit version as we are, you need to do a patch on the camera driver here, since the Preview kit uses Linux kernel 2.4 and the driver in 2.4 is an old version which does not support Logitech 4000 Pro camera. Find the patch instruction in Appendix B: Tricks and Patches.

After the Linux configuration, the last step is to compile the kernel for the target PowerPC processor. The output from kernel compilation is a *.elf file which can boot from our platform.
4 Software: Video & Memory Interfaces

The video and memory interfaces software is developed as a regular Linux programme after the OS Linux is setup on the platform. The overview of the software flow is presented in Figure 2-5 Video Interface Software Flow Chart. In this chapter, the video interface software will be discussed in detail within three sub-sections as Video4Linux API introduction, image capture using Video4Linux, and a description of the image display routine.

4.1 Video4Linux API

In Linux world, it is very popular to use an Application Programming Interface (API) called Video4Linux [3] to play with video related peripherals. In this thesis project, I choose to follow this regulation.

4.1.1 What is Video4Linux?

From Linux kernel source documentation we can get the definition of Video4Linux as this: *Video for Linux is a set of APIs and standards for handling video devices on Linux. It is a suite of related driver specifications for different types of video devices and video-related data.* [4] This Video4Linux API can be viewed as an interface mechanism stands in-between Linux software and video device drivers, as illustrated in Figure 4-1 below. From the software side, Video4Linux provides a unique set of device control interfaces. In the other side, it supports a communication with different video device drivers, which include the drivers for most of the current video related products.

![Figure 4-1 Video4Linux API](image-url)
4.1.2 Why we use Video4Linux?

There are two reasons which encourage us to use the Video4Linux API in our video interface software. First, it simplifies the integration of camera. Designers do not need to look into the detail about any special device driver. Instead, they only need to know the interfaces provided by Video4Linux. Moreover, a lot of Linux “open sources” can be found on internet about how to use the Video4Linux. This accelerates our software development very much. Second, the use of Video4Linux API also leaves us with the flexibility for future interchange of the camera in our project. If we change the current webcam to a new camera with a different driver to improve system performance, we do not need to spend more time on modifying the software, since its device control is only through the interface to Video4Linux, not any special device driver.

4.2 Image Capture

4.2.1 Capture Mechanism

The image capture is a process of synchronizing a memory area, located in the on-board memory space, with the corresponding image buffer inside the camera. This on-board buffer can be viewed as a mirror of the image buffer built in camera.

The image buffer built in camera is device-dependent. For example, some devices support buffer for only one image and some others support double image buffers. Our Logitech 4000 Pro belongs to the later. Here let’s name the whole image buffer as “BIGBUF” and buffer for each image “BUF”. Thus the Logitech webcam has a built-in BIGBUF which includes two BIFS. Each BUF in camera has two states; open and close. The BUF is set close when synchronization is required on it, and it can be opened by an open command. When the camera starts to run, it keeps delivering image sequence into the BIFS circularly at a pre-set framerate. If the BUF is close, the camera will drop that frame and go to the next one.

Based on the information above, the image capture process of synchronizing the mirror buffer on board with the camera’s BIFS can be drawn as Figure 4-2.
4.2.2 Linux ioctl Command

The camera control through Video4Linux is achieved by using several Linux ioctl commands provided by Video4Linux. This section is an introduction to this Linux ioctl function call and it is followed by a detailed discussion on how to use the Video4Linux ioctl commands. If you are familiar with Linux ioctl, you can jump over the ioctl introduction and go to section directly.

The grammar of this ioctl function call is like this,

```
int ioctl(int fd, int cmd, ...);
```

The first parameter “fd” is the file descriptor that indicates which device this ioctl command is passed to. The second parameter “cmd” is the device control command defined by device driver to tell the device what kind of operation the software is requiring. In this project, the ioctl commands to control camera are defined by Video4Linux. The “dots” at the end represent variable values or pointers to variables (depend on the cmd) required by the ioctl command. An example of this ioctl function call used in our software is taken as below.

```
ioctl(cam, VIDIOCGCAP, &v_cap);
```

Where the “cam” is the file descriptor of the camera, VIDIOCGCAP is a Video4Linux ioctl command for enquiry of camera capabilities, and the pointer to a pre-defined structure “v_cap” is used to gather and store the received capability data from camera.

The Video4Linux ioctl commands we use in our project can be categorized into two types; camera initialization and image capture, which are the main focus of the rest part of this section.
4.2.3 Camera Initialization

Before reading image from camera, we need to pass some capture configuration parameters such as image size, colour palette, white balance value, etc. These capture settings are the main topic in this section.

- **Query camera capability**

Video4Linux provides an ioctl command VIDIOCGCAP for querying video device capability. The returned device capability information includes:

* Canonical name for this interface
* Type of interface
* Number of radio/tv channels if appropriate
* Number of audio devices if appropriate
* Maximum capture width in pixels
* Maximum capture height in pixels
* Minimum capture width in pixels
* Minimum capture height in pixels

In this thesis project, I use this VIDIOCGCAP ioctl to get the Logitech 4000 Pro webcam’s capability, by calling the following ioctl function.

```c
struct video_capability v_cap;
...
ioctl(cam, VIDIOCGCAP, &v_cap);
```

It returns the capability information of the webcam as what is printed in the window below.

```
---------------VIDEO CAPABILITIES------------------
Name : Logitech QuickCam 4000 Pro
Type : 1
Channels : 1
Max width : 640
Max height : 480
Min width : 160
Min height : 160
```

- **Capture window**

The ioctl command for setting capture window parameters is like

```c
ioctl(cam, VIDIOCSWIN, &v_win);
```

Where the `v_win` is a structure defined by Video4Linux. It includes the parameters of capture window settings like offset, size, and framerate. The value of `v_win` should be set before the ioctl function call, as the code showed below.

```c
struct video_window v_win;
...
v_win.x=0;
v_win.y=0;
v_win.width=WIDTH;
v_win.height=HEIGHT;
v_win.clipcount=0;
v_win.chromakey=1;
```
One evident restriction here is that the capture parameters should not be set beyond the
device capability. Moreover, even within the capability, the settings may not be accepted
by the device. For example, if we set the capture resolution as 300*300, the device may
only set itself to a resolution of 320*240, the closest one to the user configuration under
which the device can work. Another point need consideration is the balance between
capture window resolution and framerate. In this thesis project playing with a Logitech
4000 Pro camera, to get a frame size greater than 176x144 you need to set a proper frame
rate (we have to set 5 fps to get a size of 320x240). Since the settings may not be
accepted by the device under control, we need to check the capture configuration after
setting, by the ioctl function call below.

\[
\text{ioctl(cam, VIDIOCSWIN, &v\_win)};
\]

An example output for the capture window setting can be seen in the following window.

--- CAPTURE WINDOW ---
Position of window: X-coord: 0 Y-coord: 0
Windowwidth : 320 Windowheight : 240
Clipcount : 0
Chromokey : 0
Flags: 0
framerate : 5

- **Image property**
We can get image property by calling this ioctl function

\[
\text{struct video\_picture v\_pic;}
\]

\[
\text{ioctl(cam, VIDIOCGPICT, &v\_pic)};
\]

The returned image property includes the image brightness, colour, contrast, whiteness,
deepth, and palette. See the example showed in the window below.

--- IMAGE PROPERTY ---
Brightness : 126
Colour : 48
Contrast : 200
Whiteness (gamma) : 0
Depth : 24
Palette : 15

- **PWC white balance**
This is a special ioctl command for the camera using PWC (Philips Webcam) driver,
including our Logitech 4000 Pro camera. This is the part of code we need to change in
software side after changing the camera and driver. The same problem is with the PWC shutter setting.

The white balance is the setting used to change the amount of blue and red in the images. This may be useless for this grey image based processing project. But I would put it here for the other project based on colour image processing. By this ioctl command, the white balance can be set to modes as automatic, manual, indoor, etc [5]. The pass of these settings to the video device also uses the set-and-get method as the capture window setting. The code can be like this

```c
struct pwc_whitebalance pwc_white;
pwc_white.mode = PWC_WB_FL;
// or PWC_WB_MANUAL;
//   pwc_white.manual_red = 5000;
//   pwc_white.manual_blue = 5000;
if (ioctl(cam, VIDIOCPWCSAWB, &pwc_white) < 0){
    perror("VIDIOCPWCSAWB");
    close(cam);
    exit(1);
}
if (ioctl(cam, VIDIOCPWCGAWB, &pwc_white) < 0){
    perror("VIDIOCPWCGAWB");
    close(cam);
    exit(1);
}
```

The result printed out by the software can be seen in the following window, where the value 2 indicates a white balance mode of PWC_WB_FL [5].

```
---------------PWC WHITE BALANCE-------------------
PWC white balance : 2
```

**PWC shutter**

Another Philips WebCam specific capture setting is the PWC white balance. The programming code is as below.

```c
unsigned int pwc_shutter = 45000;
...
if (ioctl(cam, VIDIOCPWCSSHUTTER, &pwc_shutter) < 0){
    perror("VIDIOCPWCSSHUTTER");
    close (cam);
    exit(1);
}
```

The shutter value we use on the Logitech webcam is displayed in the window below.

```
---------------PWC SHUTTER-------------------------
Camera shutter : 45000
```

**Map buffer**

This initialization step maps the “mirror” image buffer in the on-board memory, which is the same size as the camera-built-in image buffer. In order to map such a buffer, we need first get the device-dependent parameters from the camera by using a VIDIOCGMBUF
ioctl command. Those parameters include the buffer size and the frame number, which will be used in memory map Linux command. See the code below as example.

```c
struct video_mbuf v_mbuf; // Video4Linux structure
unsigned char *bigbuf; // image buffer
...
if (ioctl(cam, VIDIOCGMBUF, &v_mbuf) == -1) {
    perror("VIDIOCGMBUF");
    close(cam);
    exit(1);
}
// to get buffer mapping parameter
bigbuf = (unsigned char *) mmap(0, v_mbuf.size, PROT_READ | PROT_WRITE, MAP_SHARED, cam, 0); // map buffer
-------MAP BUFFER----------------------
video buffer size: 921600
video buffer frame number: 2
```

- **Set buffer**

The last thing need to be done in camera initialization is to open all the BUFs in the camera for capture (review section 4.2.1 Capture mechanism for reason). The ioctl command for open is VIDIOCMCAPTURE.

```c
struct video_mmap v_mmap; // video4linux structure
...
v_mmap.width = v_win.width;
v_mmap.height = v_win.height;
v_mmap.format = v_pic.palette;
for (frame = 0; frame < v_mbuf.frames; frame++) {
    v_mmap.frame = frame;
    if (ioctl(cam, VIDIOCMCAPTURE, &v_mmap) < 0) {
        perror("VIDIOCMCAPTURE");
        close(cam);
        exit(1);
    }
}
```

Until now, we have finished the camera initialization process.

### 4.2.4 Image Capture

As discussed in capture mechanism, two camera control commands are needed in the image capture loop; VIDIOCSYNC for buffer synchronization and VIDIOCMCAPTURE to open the synchronized buffer.

### 4.3 Display

The task of the display routine is to read a greyscale image and show it on the screen through a PCI graphic card. To programme this task in Linux, we borrow an open source routine named fb_display from Smoku, Stellar Technologies. Since the fb_display takes input image in RGB format, we need to convert our gray image to RGB and then display
it. The gray to RGB conversion and framebuffer display routine compose the main topics of this section.

4.3.1 Gray to RGB conversion
The basic principle of transforming a grayscale image to a RGB format one is to assign the grey value to all the R, G, and B, as presented in equation below.
\[ R = G = B = Y \]

To ease the understanding of this value assignment, the code of gray to RGB conversion is listed here.

```c
//Convert a grayscale image to a grayscale rgb image
int gray2rgb (unsigned char *rgb_out, unsigned char *gray_in, int width, int height){
    int x,y;
    int position = 0;
    unsigned char Y;
    unsigned char *pOut = rgb_out;
    for (y = 0; y < height; y++){
        for (x = 0; x < width; x++){
            Y = gray_in[position++];
            *pOut++ = Y;  //R
            *pOut++ = Y;  //G
            *pOut++ = Y;  //B
        }
    }
    return 0;
}
```

4.3.2 Framebuffer display
The framebuffer display routine reads a RGB format image from memory and display it on the screen connected to the graphic card. The parameters that should be passed to this function include:
- A pointer to the RGB image buffer;
- The size(width and height) of the image to be displayed;
- The position of the image on the screen.

4.4 Keyboard Hit Detection
The last step in one frame processing is to decide whether to go to the next frame or jump out the loop of image sequence processing. This is done by a function called kbhit, which detects the keyboard hit of ENTER. If an ENTER is pressed, this function returns 1, otherwise 0. The software decides whether to go out of the loop according to this return value.

The kbhit function is realised by calling an ioctl as below:
```c
int kbhit(){
    int n;
    ioctl(0, FIONREAD, &n);
    return n;
}
```
And the use of kbhit in the main software can be seen in the code here:

```csharp
//check keypress to quit
if (kbhit() <= 0 )
    run = 1;
else
    run = 0;
```
5 Hardware: ContrastEngine

The previous chapters focus on the video interface software. From this chapter on, the second half of this thesis, the video processing and its hardware acceleration will be described. This chapter starts from a functional description of the ContrastEngine, associated by a software implementation demonstration. Then the major part, the hardware implementation is discussed in detailed.

5.1 Functional Description

The ContrastEngine is designed on the purpose of increasing the contrast in a pre-defined sub-window of an input image. This sub-window, named ROI (Region of Interest) in AutoVision Project, is usually the detected tunnel entrance, which is dark and low contrast compared to the region outside. In order to get information from this low quality part of image, we need first apply a contrast enhancement to this window. Another problem needs to be concerned here is that the noise coming from the camera lens and electrical devices will also be enlarged when we doing contrast enhancement. So we need to reduce noise before enhance the contrast. In conclusion, the ContrastEngine performs two operations on the ROI widow; Noise Reduction and Contrast Enhancement.

5.1.1 Noise Reduction

The noise from camera lens and electrical devices are usually pepper-salt like noise which randomly disturbs single pixel in single frame. In other words, if one pixel is a noisy one, the possibility of its surroundings and its corresponding pixels in the neighbour frames being noisy is very low. Based on this property, two algorithms can be considered to reduce the noise. One is to use a matrix based filter to determine the pixel value by itself and its surrounding pixels. Another method is to determine the luminance values at each pixel position by taking a median from a set of previous ROIs. Our AutoVision project chooses the average filter algorithm to reduce the noise. This algorithm can be easily understood with the association of Figure 5-1 and Eq.5-1.

\[ y_{\text{out}}(n) = \frac{y(n) + y(n-1) + y(n-2) + y(n-3)}{4} \]  

Eq. 5-1
The number of images being used by average filter has to be carefully chosen. It should be neither too small nor too big. If very few images are taken from the image sequence, the filter will not have enough effect on the noise. On the other hand, averaging too many images will not only increase the computing load, but also add more memory access operations to ContrastEngine since more images or pixels need to be read from the memory. Moreover, too many input images may not lead to a better performance, since the far before image is much different from the current captured image. The last but not the least consideration should be taken in this filter design is on the FPGA implementation. Because of FPGA’s weakness in doing division calculation, this average filter should better be power of 2. That is, an average filter which calculates the medium of two input images can be implemented by an adder and a one-bit shifter. If the number of input images is four, we need to shift out the lower 2 bits from the adder’s result. This thesis’ finally decision is to use four input images to do the average.

5.1.2 Contrast Enhancement
The second functionality performed by ContrastEngine is to enhance the ROI window’s contrast, without disturbing the pixels outside this window, as illustrated in Figure 5-2. The rectangular area where the tunnel is shows the ROI window.
This contrast enhancement function is realized by two scan loops. The first scan is performed in order to estimate the luminance range, that is, to get the maximum and minimum luminance value. This scan is applied to a sub-sampling window inside the ROI. The definition of this window is shown in Figure 5-3.

![Figure 5-3 ROI and Sub-sampling Window](image)

The second scan loop scales the pixels’ value in the ROI window to a full range of luminance value from 0 to 255, as in Figure 5-4 Contrast Enhancement from Small Luminance Range to Full Range. This scale calculation is based on Eq.5-2 by using the maximum luminance value $Y_{\text{max}}$ and the minimum luminance value $Y_{\text{min}}$, which are estimated in the first scan loop.

![Figure 5-4 Contrast Enhancement from Small Luminance Range to Full Range](image)

$$Y_{\text{out}} = \frac{Y_{\text{in}} - Y_{\text{min}}}{Y_{\text{max}} - Y_{\text{min}}} \times 255$$  \hspace{1cm} \text{Eq. 5-2}

Again, the FPGA hardware implementation of equation Eq. 5-1 requires a consideration on the division by $Y_{\text{max}} - Y_{\text{min}}$, which is a variable and hard to be implemented in FPGA. The following section describes the FPGA implementation of the ContrastEngine including this algorithm.
5.2 Hardware Implementation

The ContrastEngine works as a hardware acceleration core hanging on the Processor Local Bus (PLB) to coprocess with the PowerPC on the same bus. Other onboard components or video devices such as DDR memory, camera and graphic card are also connected to this PLB bus through bridges and controllers. The system architecture can be viewed in Figure 5-5 Codesign System Architecture.

![Figure 5-5 Codesign System Architecture](image)

The operation mode of ContrastEngine is Master/Slave mode, which means that the ContrastEngine is capable of accessing other resources on the bus actively without going through the CPU. A typical example is its memory access. ContrastEngine gets data from memory directly, and so does its write operation to memory.

The following sub-sections focus on the hardware implementation of ContrastEngine including the following sub-modules as its interface block IPIF, memory arbiter, config register, and the master processing unite. A block diagram is given below as an overview of these components and their interconnections.
5.2.1 IPIF to PLB

The PLB IPIF, standing for Processor Local Bus IP Interface, is a Xilinx CoreConnect compatible LogiCORE™ that provides a bi-directional interface between a User IP core, for example this ContrastEngine, and the PLB 64-bit bus standard. The PLB is the local bus for the PowerPC processor integrated in the Xilinx Virtex™ II Pro FPGA. Xilinx provides this interface core to simplify the bus connection protocols and ease the integration of the user logic. IPcore designers can use an IP create/import wizard, which is packaged with Xilinx EDK tool, to config this IPIF in order to get the features needed by their user logic. For example, the select of Master/Slave mode or Slave only mode, the DMA enable or disable, the size of register, and etc.

The IPIF block in ContrastEngine is configured to include the feature of User Logic Master Support. And an additional signal Bus2IP_Addr should be open manually during IP interconnect configuration.

5.2.2 Memory Arbiter

The Memory Arbiter is the unique module connected to the IPIF in the user logic side. It works as a MUX to distinguish data from PLB bus between register configuration data and master transfer data. The register configuration data is the data sent or required from processor by software. These data includes the ContrastEngine configuration data under which the ContrastEngine works and a status report of ContrastEngine showing its current status like busy or idle. Another category of data is the master transfer data, the data for master transfer operation. If a memory read is requested by the master unite of
the user logic, the acknowledged data from memory will be feed to the master unite after the Memory Arbiter. This master mode data transfer is frequently used when the ContrastEngine is started and it processes the image data stored in the memory actively. A simplified diagram of this MUX-like Memory Arbiter is shown in Figure 5-7 Memory Arbiter’s Function View. The key signal to differentiate data between config register data and master unite data is the “Bus2IP_IPMstTrans” signal indicated by IPIF. This signal is high when the current read or write operation on the user IP is initiated by the user IP’s master transaction request interface.

![Figure 5-7 Memory Arbiter’s Function View](image)

Figure 5-7 lists the interfaces of Memory Arbiter. Readers can refer to the following subsections of Config Register and Master Unite to find which ports are connected to which blocks. Here is a general description of these interconnects. The ports with a prefix of “reg_” are the signals connected to config register, which includes address bus “reg_addr”, data bus “reg_din”, and a write enable signal “reg_we”. Another bunch of signals with a prefix of “trin_” or “trout_” or “tran_” are connected to Master Unite for master transfer. The rest signals as “Bus2IP_” or “IP2Bus_” are for the IPIF interface.
5.2.3 Config Register

The config register is the space storing ContrastEngine’s configuration data such as the images’ physical address in the memory, the processing window position and size, as illustrated in Figure 5-9. A register mapping is listed in Table 5-1 Contrast Engine Config Register Mapping.

![Figure 5-8 Memory Arbiter Block View](image-url)
Figure 5-9 ContrastEngine Config Parameters

Table 5-1 Contrast Engine Config Register Mapping

<table>
<thead>
<tr>
<th>reg_addr</th>
<th>32bit</th>
<th>16bit</th>
<th>16bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Resaddr</td>
<td>respos.a</td>
<td>respos.b</td>
</tr>
<tr>
<td>001</td>
<td>In1addr</td>
<td>in1pos.a</td>
<td>in1pos.b</td>
</tr>
<tr>
<td>010</td>
<td>In2addr</td>
<td>in2pos.a</td>
<td>in2pos.b</td>
</tr>
<tr>
<td>011</td>
<td>In3addr</td>
<td>in3pos.a</td>
<td>in3pos.b</td>
</tr>
<tr>
<td>100</td>
<td>In4addr</td>
<td>in4pos.a</td>
<td>in4pos.b</td>
</tr>
<tr>
<td>101</td>
<td>winsize.a</td>
<td>winsize.b</td>
<td>-</td>
</tr>
<tr>
<td>110</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>111</td>
<td>-</td>
<td>-</td>
<td>“GO” signal</td>
</tr>
</tbody>
</table>

* Blanks are not used currently. It will be used for further potential extension like engine status report.

The port-list of config register is shown in the block view of Figure 5-10 ContrastEngine Config Register Block View. The “regs” to Master Unite is a package of configuration data includes the register’s contents as in Table 5-1. The Config Register also takes in charge of the start trigger which is implemented by writing to a certain register address with a special value named GO_DATA_KEY, as the “GO signal” shown in Table 5-1. As the Master Unite finishes its pixel processing, a done signal is returned to Config Register and then it sends the interrupt to the interrupt controller.
5.2.4 Master Unite

The Master Unite does the major job of ContrastEngine; as soon as it is started, it reads
the pixel value from image and writes the processing result back to the memory. This
process is running in master mode and without the PowerPC’s control. The PowerPC is
just waiting for the interrupt. The implementation of this Master Unite includes several
sub-modules for example a top-level state machine, an address counter, the transfer
input/output block, and the processing blocks as average filter and compare/scale
function block. The data width inside the master unite is 64 bits, since the current version
of ContrastEngine uses Single Beat data transfer, which means each time it read 64 bits
or 8 bytes from the memory. It processes 8 pixels at the same time.
5.2.4.1 Master State Machine

This top level state machine functions as the central control unite inside the Master Unite. It sends out all the control signals to other blocks to schedule their cooperation. The state diagram assisted with a processing route inside the ROI is shown in Figure 5-12. Since an average filter is implemented to reduce noise, we need to read from four input images to do the average; this is done by the self-loop in read state. The image id range from 0 to 3 indicates which input image is currently read by the ContrastEngine. In the first scan of the sub-sampling window, no write operation is needed, instead of which the comparison to find the $Y_{max}$ and $Y_{min}$ is required. This comparison is pipelined with the data reading so the sub-sampling window’s scan is done by repeatedly read. After this first scan achieves point B, which is the end of sub-sampling window as shown in Figure 5-12, the next pixel will be pointed to the top-left one of the ROI window. From this time on the second scan loop which performs the contrast enhancement starts. The processing of this scan is scheduled in this way: first read from four input images and then do the average, after that a scaling operation is applied, the final step is write the result value back to the memory of result image buffer. The ContrastEngine is started by a start signal, which is sent from the software running on PowerPC, by writing a special GO_DATA_KEY to a certain register address. When the Master Unite finishes its processing of one frame by arriving at point C, the state machine jumps to the state of “done” and sends out a done signal to trigger the interrupt. At the next clock cycle it goes to idle and keeps waiting for the next start signal.
5.2.4.2 Address Counter

The address counter calculates the addresses for both Tran.In and Tran.Out blocks. Its outputs have both reading address and writing address. These two addresses change when a request of next address next_addr_req arrives. An additional duty of this address counter block is to indicate both the end of sub-sampling window and the end of ROI window, which are the point B and C in the Figure 5-12 above. A block view of this address counter is included below.

![Address Counter Diagram](image)

**Figure 5-13 ContrastEngine Address Counter Block View**

5.2.4.3 Tran.In and Tran.Out

The Tran.In and Tran.Out are the blocks for master data transfer, Tran.In for read and Tran.Out for write. The addresses for both read and write are provided by the Address
Counter. The Tran.In sends read-in data to average filter while the Tran.Out writes the result data back to the memory.

### 5.2.4.4 Average Filter
The average filter for noise reduction is implemented by using an accumulator and a two-bit shifter. The accumulator does the accumulation of four input images. It is reset when the data from the first image arrives, and keeps adding when the second, third and fourth image’s data arrive one by one. The two-bit shifter does the division by four. Here we can see the select of average image numbers’ influence in hardware implementation.

![Figure 5-14 Average Filter](image)

### 5.2.4.5 Compare and Scale
The comparator is used to find $Y_{\text{max}}$ and $Y_{\text{min}}$. Its function is enabled by a signal called “comp_en” sent from the top level state machine. At the start of the ContrastEngine, this comparator is reset to a max output of 0 and a min output of 255. During the scan in the subsampling window the input data is compared and the max and min value changes according to the input pixel value. After the first scan the comp_en will not active and the max and min keep as constant indicating the $Y_{\text{max}}$ and $Y_{\text{min}}$, which will be used as input by the scale function block doing contrast enhancement.

![Figure 5-15 Max/Min Comparator](image)

The scale function block is enabled by a “scale_en” signal sent from the master state machine. It uses the max and min inputs to scale the current input data to a larger range of luminance value. This is done by first checking the distance between max and min, that is $Y_{\text{max}} - Y_{\text{min}}$, and then searching this value in a look-up-table to get the scaling parameter, the multiplier. The result data "dout" is the input “din” multiply by this scaling parameter.

![Figure 5-16 Luminance Scaler](image)

Until now the ContrastEngine’s hardware implementation is introduced. For further and detailed understanding or following work on this project the readers can refer to the ISE and EDK project design file and VHDL code which are included in the CD of this thesis.
To control the ContrastEngine hardware core from Linux user space, we need to write a driver. This driver is developed in the same way as most computer devices. The ContrastEngine is viewed as a character device. A very good and popular reference for writing Linux device driver is written by Alessandro Rubini and Jonathan Corbet, whose book is now available online at the URL http://www.xml.com/ldd/chapter/book/. Moreover, this ContrastEngine driver is developed after Mayank and Florian’s excellent work on the AddressEngine driver, as mentioned in the acknowledgement section of this thesis report. The ContrastEngine driver supports operations including device open and close, device register writing, start of the device and interrupt handling. These operations are separately discussed in the following sub-sections.

### 6.1 Device File and Major-Minor Numbers of Device

Linux manages hardware devices as files. The file describing a hardware device is composed of a major number and a minor number. Software program requests hardware driver by looking into its device file. Linux puts all those device files in the directory /dev of its file system. Before a device driver is loaded, its corresponding device file should be created in the folder /dev. This is done by a command:

```bash
mknod /dev/cteng c 254 0
```

where the device file “cteng” (for ContrastEngine) is created in Linux filesystem under the directory of /dev. The c means cteng is a character device, and 254 is its major number, 0 is its minor number.

### 6.2 Driver Module Load and Unload

The ContrastEngine driver is compiled as a loadable module which can be loaded at run-time of the system. To use the driver, we need first load this module by the command:

```bash
insmod ContrastEngineDriver.o
```

The .o file is the compiled driver object file. To remove this module, we can use the command

```bash
rmmod ContrastEngineDriver.o
```

The instantiate and remove of driver module will cause the execution of driver functions of module load and unload. This part of code in the driver is taken to here as example.

```c
/***********************
Module load and unload
***********************
/*
 * Initialize the module - Register the Character device
 */
int init_module(void) {
    int result = register_chrdev(254, AUTOVISIONNAME, &autovision_fops); //Register the character device (atleast try)
```
if(result < 0){                                                //Negative value signify an error
    printk(KERN_ERR "cteng: can't get major %d\n", 254);
    return result;
}
printk("<1>AutoVision:Hello, world, register_chrdev returned %d\n", result);
return 0;
}

/* Cleanup - unregister the appropriate file from /proc */
void cleanup_module(void) {
    unregister_chrdev(254, AUTOVISIONNAME); //Unregister the device
    printk("<1>AutoVision:Goodbye cruel world\n");
}

6.3 Device Open and Release
Now we come to the driver programming part. Let us start from two basic device file operations; open and release. All devices should be opened before use and be closed afterwards. The open function performs the task of checking if the devices I/O-region is free to use, and if yes, register our device. What else should be included during device open in our ContrastEngine driver is to request interrupt and install interrupt handler. In the contrary part of open function, the release function should disable the interrupt and un-map the devices I/O-region. The part of open and release in ContrastEngine driver is taken as example. See the codes below.

static int cteng_open(struct inode *inode, struct file *file) {
    int result;
    /*Check if we can access the Devices I/O-Region and then register us*/
    if (check_mem_region(CTENG_PHYS, CTENG_SIZE)) {
        printk(KERN_ERR "cteng: memory already in use\n");
        return -EBUSY;
    }
    request_mem_region(CTENG_PHYS, CTENG_SIZE, CTENG_NAME);

    cteng_addr = ioremap_nocache(CTENG_PHYS, 64);
    printk("<1> Went throught cteng_open, remapped ContrastEngine from %x to %x\n", CTENG_PHYS, ((unsigned int)(cteng_addr)));

    /*Request interrupt and install handler*/
    result = request_irq(CTENG_IRQ, cteng_interrupt, 0, CTENG_NAME, NULL);
    if (result) {
        printk(KERN_ERR "adeng: can't get assigned irq %i\n", CTENG_IRQ);
    }
}
return -ERESTARTNOINTR;
}
else
{
    enable_irq(CTENG_IRQ);
    #ifdef DEBUG
    printk(KERN_DEBUG "cteng: got interrupt %i\n", CTENG_IRQ);
    #endif
    return SUCCESS;
}

static int cteng_release(struct inode *inode, struct file *file){
    disable_irq(CTENG_IRQ);
    free_irq(CTENG_IRQ, NULL);
    iounmap(cteng_addr);
    release_mem_region(CTENG_PHYS, CTENG_SIZE);
    MOD_DEC_USE_COUNT;
    return SUCCESS;
}

6.4 Mapping Physically Continuous Memory

There is a gap between Linux’s memory management and our ContrastEngine’s memory access. Linux uses virtual memory as memory management method which is not physically continuous, but the ContrastEngine can only access memory by using data’s physical address. So we need to map a special memory area in Linux and put image data in this area. This is done by using the high memory area in Linux and the mapping of this memory area is performed in the ContrastEngine driver, in the way as showed in the code below.

/*
 * This Function enables the User Space Process to get the Device continuous physical memory mapped to it's own virtual address space
 */
int cteng_mmap(struct file *file, struct vm_area_struct *vma)
{
    unsigned long physical = vma->vm_pgoff << PAGE_SHIFT;
    vma->vm_flags |= VM_RESERVED | VM_IO ;
    if ( remap_page_range( vma->vm_start, physical, vma->vm_end - vma->vm_start, vma->vm_page_prot ) )
        return -EAGAIN;
    return SUCCESS;
}

6.5 Write Config Register

To pass the configuration data to ContrastEngine we write this file operation in the driver by using a kernel function memcpy_toio(), as in the codes example below.

ssize_t cteng_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos){
if ( count != 64 ) return -EINVAL;
memcpy_toio(cteng_addr, buf, 64);
wmb();
return count;

6.6 Send START signal

The start of ContrastEngine is done by writing to a special register address with a certain value called GO_DATA_KEY. This is implemented as a ioctl command CTENG_IOSTART.

switch( cmd )
{
    case CTENG_IOSTART:
        working = 1;
        // * GO_DATA_KEY = 0x90;
        writeb(0x90, cteng_addr+63);//LOW_ADDR:111111
        wmb();
        break;
... 
}

6.7 Wait and Response to Interrupt from HW

The ContrastEngine finishes its every frame processing by sending an interrupt signal to the PowerPC through an interrupt handler IP core. In the device driver side, we also need a function to wait and response to the interrupt from the HW. This is implemented as an ioctl command CTENG_IOWAIT.

switch( cmd )
{
    case CTENG_IOWAIT:
        if(working)
            interruptible_sleep_on(&cteng_queue);
        break;
    default:
        return -ENOTTY;
        break;
... 
}

The above is an outline of the ContrastEngine driver and its operations. For detail driver programming the readers can refer to Alessandro Rubini and Jonathan Corbet’s online book Linux Device Driver (http://www.xml.com/ldd/chapter/book/) and for the complete codes of ContrastEngine driver please look into this thesis’ CD.
7 Results

7.1 Frame Rate
The current design achieves a frame rate of about 3.5 frames/second. The camera is running at 5 frames/second.

7.2 FPGA Usage
By prototyping this project in Xilinx ML310 board where a Xilinx Virtex II Pro FPGA is integrated, the Linux base system’s hardware support costs 47% of the FPGA. And the ContrastEngine spends 17% FPGA resource. In total, this thesis project costs 64% of the Virtex II Pro FPGA.

7.3 Performance Measurements
The performance measurement result of this thesis project is displayed in two bar-charts as below. Figure 7-1 shows the comparison of the execution time between SW and HW implementation. From this figure we can see the HW costs the minimum time. Our acceleration aim is achieved. The difference between SW-nocache and SW-cached is as what is indicated by their names. The nocache implementation doesn’t use the cache for the image data because of our imperfect solution of memory mapping. The physically continuous memory is mapped with very basic memory management method, no data cache is introduced. In contrast, the SW-cached solution uses the Linux’ own virtual memory management mechanism. That is, the image data are not stored in a special mapped memory as the nocache implementation; instead the data are put into a data array which is virtually continuous. Linux uses its own memory management mechanism to manage these image data, so the cache is available to use.
Another comparison is made on the CPU load between the SW and HW implementation of ContrastEngine. From the bar-chart of Figure 7-2 we can see that our second goal of using HW ContrastEngine is achieved; the CPU load reduced from 50% to 40%. An additional phenomenon need a notice here is that when using the physically continuous memory as image buffer (both the SW and HW example in the bar-chart use this method), the CPU load increases very much compared to using a data array to store captured image, from about 17% to 50%. This again shows the imperfect of our memory map mechanism, on which there is a potential for improvement. However, except for this drawback, when considering only the ContrastEngine function, the HW implementation can reduce 10% of the CPU load.
Figure 7-2 Performance Measurement of CPU Load
8 Discussion

The last part of this report is a discussion on this thesis project, including the performance analysis, current design’s limitations and proposals for improvements.

8.1 Memory Access Bottleneck

From Figure 7-1 we can find that the HW implementation doesn’t give significant performance enhancement; only around half of the time is reduced from the SW-cached solution. This indicates that for the task of ContrastEngine, the bottleneck exists on the memory access. The current HW ContrastEngine design uses Single Beat data transfer, a very basic memory access method by which each read or write operation is performed on 64bits (8bytes). To improve the performance from this point, we can use a faster data transfer mechanism supported by the manufacture Xilinx; either the Burst data transfer or DMA. As the memory access speed increases, the computational load of the HW ContrastEngine will also grow. Then more data need to be processed in the hardware pipeline. This will increase the FPGA resource occupation.

8.2 USB1.1 Interface Limitation

Another limitation is on the USB interface of camera. This ML310 board is featured with USB1.1 port, of which the bandwidth is limited to 12Mbits/sec. What’s even worse is that for single port of the two available ones, the speed can only achieve 6Mbits/sec. Let’s do a calculation for capturing 320*240 YUV420P images. The size of each image is

\[
320 \times 240 \text{(pixels)} \times 12 \text{(bits / pixel)} = 921600 \text{(bits)}
\]  
Eq. 8-1

If here we only consider the USB data transfer, the frame rate can be estimated as in the equation below.

\[
\frac{6 \times 10^6 \text{(bits / sec)}}{921600 \text{(bits / frame)}} = 6.5 \text{(frame / sec)}
\]  
Eq. 8-2

Thus the camera is configured as running at 5frames/sec. Plus the image processing and display time, the final frame rate can only achieves about 3.5frames/sec.

This framerate is not our final goal as a real-time application, so an improvement proposal is made to overcome this USB1.1 limitation. We plan to use a framegrabber card with a PCI interface which is also supported by ML310 board. The PCI bus is much faster than the USB1.1 so the image capture and data transfer bottleneck could be removed.
9 Outlook

This thesis demonstrates a basic platform for video processing with a software and hardware codesign concept. Plus with an Embedded Linux OS, it provides much flexibility to be used in prototyping other video processing algorithms. Other users can either test their algorithm as software application code or implement it in FPGA hardware core. Both the video interface and the SW/HW interface can be borrowed from this thesis work.

By taking the advantage of FPGA’s configurable property, such an FPGA based HW/SW codesign platform will gain even more benefit as a currently very hot research area, the reconfigurable computing. The Embedded Linux and FPGA’s self-configuration support will enable the hardware’s function to be reconfigured at run-time easily. In contrast to the current widely use of FPGA in telecommunication Base-Station, which used to be reconfigured by engineers from long-distance to modify the function or solve problems, this dynamic partial self-reconfiguration bring a new idea which is going to widen the use of FPGA in electronic applications. Such a platform neither needs to config its hardware by a host PC, nor to stop the running system to do reconfiguration. The FPGA hardware core is reconfigured at runtime and by the system itself to adapt to different required function. One try of this concept is happening in automotive area; the AutoVision group in Institute for Integrated System is prototyping a video based driver assistant on such a FPGA platform. The hardware acceleration core will be reconfigured when the car drives into different environments. This reconfigurable computing technique will bring us more in the future.
Appendixes

Appendix A: Linux Commands and Programming Skill

1. Use USB memory stick

# mkdir /mnt/usb1
# mount /dev/sda1 /mnt/usb1
# umount /mnt/usb1

…to create a directory
…to mount the USB stick after plug-in
…un-mount the USB stick after use
and before remove

2. Load and unload of the device driver module

# insmod <driver_dir>/ContrastEngineDriver.o
# rmmod ContrastEngineDriver.o

…to instantiate the device driver module
…to remove the device driver module

3. CPU load measurement

# top -d 1 -n 50 -b > top.out&

4. Execution time measurement

Insert the codes as below before and after the function whose execution time is need to be measured.

//to measure execution time
struct timeval tpstart, tpend;
float timeuse;
gettimeofday(&tpstart, NULL);

//here is the code going to be measured

gmtimeofday(&tpend, NULL);
timeuse=1000000*(tpend.tv_sec-tpstart.tv_sec)+tpend.tv_usec-
tpstart.tv_usec;
timeuse/=1000000;
printf("ContrastEngine SW IMP Used Time:%f\n", timeuse);
Appendix B: Design Patch and Trick

1. Xilinx PCI Linux Reference Design Patch

The Xilinx Linux PCI reference design is first released for EDK6.2i. To work this project on later version of EDK6.3 or EDK7.1, we need to apply a update patch which can be downloaded from http://www.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=22129

2. Bus2IPMstTrans signal

This signal is not used after running the IP Create/Import Wizard, designers should open it manually by remove the previous “open” connection and connect this signal to a local iBus2IP_IPMstTrans signal and use it in the design.

```
--USER
--Bus2IP_IPMstTrans => open,
Bus2IP_IPMstTrans => iBus2IP_IPMstTrans,
```

3. Bus Conflict

The ContrastEngine may have conflict with other data transfer through the same bus. This bus conflict will cause a stall in processing. Here we manually set its priority to always the lowest by making changes on ContrastEngine’s hardware implementation:

- Block the IPIF’s M_priority signal
  
  ```
  --User
  --M_priority => M_priority,
  ```

- Assign the M_priority with a constant value which makes it always the lowest priority
  
  ```
  --User
  M_priority <= b"00";
  ```
Appendix C: Video 4 Linux API

Devices

Video4Linux provides the following sets of device files. These live on the character device formerly known as "/dev/bttv". /dev/bttv should be a symlink to /dev/video0 for most people.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Minor Range</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/video</td>
<td>0-63</td>
<td>Video Capture Interface</td>
</tr>
<tr>
<td>/dev/radio</td>
<td>64-127</td>
<td>AM/FM Radio Devices</td>
</tr>
<tr>
<td>/dev/vtx</td>
<td>192-223</td>
<td>Teletext Interface Chips</td>
</tr>
<tr>
<td>/dev/vbi</td>
<td>224-239</td>
<td>Raw VBI Data (Intercast/teletext)</td>
</tr>
</tbody>
</table>

Video4Linux programs open and scan the devices to find what they are looking for. Capability queries define what each interface supports. The described API is only defined for video capture cards. The relevant subset applies to radio cards. Teletext interfaces talk the existing VTX API.

Capability Query Ioctl

The VIDIOCGCAP ioctl call is used to obtain the capability information for a video device. The struct video_capability object passed to the ioctl is completed and returned. It contains the following information:

- name[32] Canonical name for this interface
- type Type of interface
- channels Number of radio/tv channels if appropriate
- audios Number of audio devices if appropriate
- maxwidth Maximum capture width in pixels
- maxheight Maximum capture height in pixels
- minwidth Minimum capture width in pixels
- minheight Minimum capture height in pixels

The type field lists the capability flags for the device. These are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VID_TYPE_CAPTURE</td>
<td>Can capture to memory</td>
</tr>
<tr>
<td>VID_TYPE_TUNER</td>
<td>Has a tuner of some form</td>
</tr>
<tr>
<td>VID_TYPE_TELETEXT</td>
<td>Has teletext capability</td>
</tr>
<tr>
<td>VID_TYPE_OVERLAY</td>
<td>Can overlay its image onto the frame buffer</td>
</tr>
<tr>
<td>VID_TYPE_CHROMAKEY</td>
<td>Overlay is Chromakeyed</td>
</tr>
</tbody>
</table>
VID_TYPE_CLIPPING  Overlay clipping is supported
VID_TYPE_FRAMERAM  Overlay overwrites frame buffer memory
VID_TYPE_SCALES    The hardware supports image scaling
VID_TYPE_MONOCHROME Image capture is grey scale only
VID_TYPE_SUBCAPTURE Capture can be of only part of the image

The minimum and maximum sizes listed for a capture device do not imply all that all height/width ratios or sizes within the range are possible. A request to set a size will be honoured by the largest available capture size whose capture is no large than the requested rectangle in either direction. For example the quickcam has 3 fixed settings.

Frame Buffer

Capture cards that drop data directly onto the frame buffer must be told the base address of the frame buffer, its size and organisation. This is a privileged ioctl and one that eventually X itself should set.

The VIDIOCSFBUF ioctl sets the frame buffer parameters for a capture card. If the card does not do direct writes to the frame buffer then this ioctl will be unsupported. The VIDIOCGFBUF ioctl returns the currently used parameters. The structure used in both cases is a struct video_buffer.

    void *base    Base physical address of the buffer
    int height    Height of the frame buffer
    int width     Width of the frame buffer
    int depth     Depth of the frame buffer
    int bytesperline  Number of bytes of memory between the start of two adjacent lines

Note that these values reflect the physical layout of the frame buffer. The visible area may be smaller. In fact under XFree86 this is commonly the case. XFree86 DGA can provide the parameters required to set up this ioctl. Setting the base address to NULL indicates there is no physical frame buffer access.

Capture Windows

The capture area is described by a struct video_window. This defines a capture area and the clipping information if relevant. The VIDIOCGWIN ioctl recovers the current settings and the VIDIOCOWIN sets new values. A successful call to VIDIOCOWIN indicates that a suitable set of parameters have been chosen. They do not indicate that exactly what was requested was granted. The program should call VIDIOCGWIN to check if the nearest match was suitable. The struct video_window contains the following fields.

    x          The X co-ordinate specified in X windows format.
y The Y co-ordinate specified in X windows format.
width The width of the image capture.
height The height of the image capture.
chromakey A host order RGB32 value for the chroma key.
flags Additional capture flags.
clips A list of clipping rectangles. (Set only)
clipcount The number of clipping rectangles. (Set only)

Clipping rectangles are passed as an array. Each clip consists of the following fields available to the user.

x X co-ordinate of rectangle to skip
y Y co-ordinate of rectangle to skip
width Width of rectangle to skip
height Height of rectangle to skip

Merely setting the window does not enable capturing. Overlay capturing (i.e. PCI-PCI transfer to the frame buffer of the video card) is activated by passing the VIDIOCCAPTURE ioctl a value of 1, and disabled by passing it a value of 0.

Some capture devices can capture a subfield of the image they actually see. This is indicated when VIDEO_TYPE_SUBCAPTURE is defined. The video_capture describes the time and special subfields to capture. The video_capture structure contains the following fields.

x X co-ordinate of source rectangle to grab
y Y co-ordinate of source rectangle to grab
width Width of source rectangle to grab
height Height of source rectangle to grab
decimation Decimation to apply
flags Flag settings for grabbing

The available flags are

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIDEO_CAPTURE_ODD</td>
<td>Capture only odd frames</td>
</tr>
<tr>
<td>VIDEO_CAPTURE_EVEN</td>
<td>Capture only even frames</td>
</tr>
</tbody>
</table>

Video Sources
Each video4linux video or audio device captures from one or more source channels. Each channel can be queried with theVIDIOCGCHAN ioctl call. Before invoking this function the caller must set the channel field to the channel that is being queried. On return the struct video_channel is filled in with information about the nature of the channel itself.

The VIDIOCSCHAN ioctl takes an integer argument and switches the capture to this input. It is not defined whether parameters such as colour settings or tuning are maintained across a channel switch. The caller should maintain settings as desired for each channel. (This is reasonable as different video inputs may have different properties).

The struct video_channel consists of the following

channel The channel number
name The input name - preferably reflecting the label on the card input itself
tuners Number of tuners for this input
flags Properties the tuner has
type Input type (if known)
norm The norm for this channel

The flags defined are

VIDEO_VC_TUNER Channel has tuners.
VIDEO_VC_AUDIO Channel has audio.
VIDEO_VC_NORM Channel has norm setting.

The types defined are

VIDEO_TYPE_TV The input is a TV input.
VIDEO_TYPE_CAMERA The input is a camera.

Image Properties

The image properties of the picture can be queried with theVIDIOCGPICT ioctl which fills in a struct video_picture. The VIDIOCSPICT ioctl allows values to be changed. All values except for the palette type are scaled between 0-65535.

The struct video_picture consists of the following fields

brightness Picture brightness
hue Picture hue (colour only)
colour Picture colour (colour only)
contrast Picture contrast
**whiteness**  The whiteness (greyscale only)
**depth**  The capture depth (may need to match the frame buffer depth)
**palette**  Reports the palette that should be used for this image

The following palettes are defined

- **VIDEO_PALETTE_GREY**  Linear intensity grey scale (255 is brightest).
- **VIDEO_PALETTE_HI240**  The BT848 8bit colour cube.
- **VIDEO_PALETTE_RGB565**  RGB565 packed into 16 bit words.
- **VIDEO_PALETTE_RGB555**  RGV555 packed into 16 bit words, top bit undefined.
- **VIDEO_PALETTE_RGB24**  RGB888 packed into 24bit words.
- **VIDEO_PALETTE_RGB32**  RGB888 packed into the low 3 bytes of 32bit words. The top 8bits are undefined.
- **VIDEO_PALETTE_YUV422**  Video style YUV422 - 8bits packed 4bits Y 2bits U 2bits V
- **VIDEO_PALETTE_YUV420**  YUV420 capture
- **VIDEO_PALETTE_YUV411**  YUV411 capture
- **VIDEO_PALETTE_RAW**  RAW capture (BT848)
- **VIDEO_PALETTE_YUV422P**  YUV 4:2:2 Planar
- **VIDEO_PALETTE_YUV411P**  YUV 4:1:1 Planar

### Tuning

Each video input channel can have one or more tuners associated with it. Many devices will not have tuners. TV cards and radio cards will have one or more tuners attached.

Tuners are described by a **struct video_tuner** which can be obtained by the **VIDIOCGTUNER** ioctl. Fill in the tuner number in the structure then pass the structure to the ioctl to have the data filled in. The tuner can be switched using **VIDIOCSTUNER** which takes an integer argument giving the tuner to use. A struct tuner has the following fields

- **tuner**  Number of the tuner
- **name**  Canonical name for this tuner (eg FM/AM/TV)
- **rangelow**  Lowest tunable frequency
- **rangehigh**  Highest tunable frequency
- **flags**  Flags describing the tuner
- **mode**  The video signal mode if relevant
- **signal**  Signal strength if known - between 0-65535
The following flags exist

- **VIDEO_TUNER_PAL**  PAL tuning is supported
- **VIDEO_TUNER_NTSC**  NTSC tuning is supported
- **VIDEO_TUNER_SECAM**  SECAM tuning is supported
- **VIDEO_TUNER_LOW**  Frequency is in a lower range
- **VIDEO_TUNER_NORM**  The norm for this tuner is settable
- **VIDEO_TUNER_STEREO_ON**  The tuner is seeing stereo audio
- **VIDEO_TUNER_RDS_ON**  The tuner is seeing a RDS datastream
- **VIDEO_TUNER_MBS_ON**  The tuner is seeing a MBS datastream

The following modes are defined

- **VIDEO_MODE_PAL**  The tuner is in PAL mode
- **VIDEO_MODE_NTSC**  The tuner is in NTSC mode
- **VIDEO_MODE_SECAM**  The tuner is in SECAM mode
- **VIDEO_MODE_AUTO**  The tuner auto switches, or mode does not apply

Tuning frequencies are an unsigned 32bit value in 1/16th MHz or if the **VIDEO_TUNER_LOW** flag is set they are in 1/16th KHz. The current frequency is obtained as an unsigned long via the **VIDIOCGFREQ** ioctl and set by the **VIDIOCSFREQ** ioctl.

**Audio**

TV and Radio devices have one or more audio inputs that may be selected. The audio properties are queried by passing a **struct video_audio** to **VIDIOCGAUDIO** ioctl. The **VIDIOCSAUDIO** ioctl sets audio properties.

The structure contains the following fields

- **audio**  The channel number
- **volume**  The volume level
- **bass**  The bass level
- **treble**  The treble level
- **flags**  Flags describing the audio channel
- **name**  Canonical name for the audio input
- **mode**  The mode the audio input is in
- **balance**  The left/right balance
- **step**  Actual step used by the hardware
The following flags are defined

- **VIDEO_AUDIO_MUTE** The audio is muted
- **VIDEO_AUDIO_MUTABLE** Audio muting is supported
- **VIDEO_AUDIO_VOLUME** The volume is controllable
- **VIDEO_AUDIO_BASS** The bass is controllable
- **VIDEO_AUDIO_TREBLE** The treble is controllable
- **VIDEO_AUDIO_BALANCE** The balance is controllable

The following decoding modes are defined

- **VIDEO_SOUND_MONO** Mono signal
- **VIDEO_SOUND_STEREO** Stereo signal (NICAM for TV)
- **VIDEO_SOUND_LANG1** European TV alternate language 1
- **VIDEO_SOUND_LANG2** European TV alternate language 2

### Reading Images

Each call to the `read` syscall returns the next available image from the device. It is up to the caller to set format and size (using the VIDIOCSPICLT and VIDIOCSWIN ioctls) and then to pass a suitable size buffer and length to the function. Not all devices will support read operations.

A second way to handle image capture is via the mmap interface if supported. To use the mmap interface a user first sets the desired image size and depth properties. Next the VIDIOCGBUF ioctl is issued. This reports the size of buffer to mmap and the offset within the buffer for each frame. The number of frames supported is device dependent and may only be one.

The video_mbuf structure contains the following fields

- **size** The number of bytes to map
- **frames** The number of frames
- **offsets** The offset of each frame

Once the mmap has been made the VIDIOCMCAPTURE ioctl starts the capture to a frame using the format and image size specified in the video_mmap (which should match or be below the initial query size). When the VIDIOCMCAPTURE ioctl returns the frame is *not* captured yet, the driver just instructed the hardware to start the capture. The application has to use the VIDIOCSYNC ioctl to wait until the capture of a frame is finished. VIDIOCSYNC takes the frame number you want to wait for as argument.
It is allowed to call `VIDIOCMCAPTURE` multiple times (with different frame numbers in `video_mmap->frame` of course) and thus have multiple outstanding capture requests. A simple way do to double-buffering using this feature looks like this:

```c
/* setup everything */
VIDIOCMCAPTURE(0)
while (whatever) {
    VIDIOCMCAPTURE(1)
    /* process frame 0 while the hardware captures frame 1 */
    VIDIOCMCAPTURE(0)
    VIDIOCSYNC(0)
    /* process frame 1 while the hardware captures frame 0 */
    VIDIOCSYNC(1)
}
```

Note that you are *not* limited to only two frames. The API allows up to 32 frames, the `VIDIOCGMGBUF` ioctl returns the number of frames the driver granted. Thus it is possible to build deeper queues to avoid loosing frames on load peaks.

While capturing to memory the driver will make a "best effort" attempt to capture to screen as well if requested. This normally means all frames that "miss" memory mapped capture will go to the display.

A final ioctl exists to allow a device to obtain related devices if a driver has multiple components (for example `video0` may not be associated with `vbi0` which would cause an intercast display program to make a bad mistake). The `VIDIOCGUNIT` ioctl reports the unit numbers of the associated devices if any exist. The `video_unit` structure has the following fields.

- **video**: Video capture device
- **vbi**: VBI capture device
- **radio**: Radio device
- **audio**: Audio mixer
- **teletext**: Teletext device

### RDS Datastreams

For radio devices that support it, it is possible to receive Radio Data System (RDS) data by means of a `read()` on the device. The data is packed in groups of three, as follows:

<table>
<thead>
<tr>
<th>First Octet</th>
<th>Second Octet</th>
<th>Third Octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least Significant Byte of RDS Block</td>
<td>Most Significant Byte of RDS Block</td>
<td>Bit 7: Error bit. Indicates that an uncorrectable error occurred during reception of this block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit 6: Corrected bit. Indicates that an error was corrected for this data block.</td>
</tr>
</tbody>
</table>

60
Bits 5-3: Received Offset. Indicates the offset received by the sync system.
Bits 2-0: Offset Name. Indicates the offset applied to this data.
Figure Index

FIGURE 1-1 DIFFERENT ROAD ENVIRONMENTS................................................................. 1
FIGURE 2-1 DEVELOPE PLATFORM SETUP........................................................................ 3
FIGURE 2-2 XILINX VIRTEX II-PRO CHIP................................................................. 4
FIGURE 2-3 XILINX ML310 BOARD BLOCK DIAGRAM [1]........................................ 4
FIGURE 2-4 SW/HW CODESIGN SYSTEM ARCHITECTURE........................................ 5
FIGURE 2-5 VIDEO INTERFACE SOFTWARE FLOW CHART....................................... 6
FIGURE 2-6 SW/HW INTERFACE: LINUX DEVICE DRIVER........................................... 7
FIGURE 2-7 CONTRASTENGINE GENERAL VIEW....................................................... 7
FIGURE 2-8 IMAGE DATA FORMAT AND STORAGE.................................................... 8
FIGURE 2-9 IMAGE DATA FLOW.................................................................................. 9
FIGURE 3-1 LINUX HARDWARE SUPPORT.................................................................... 11
FIGURE 3-2 LINUX INSTALLATION FLOW..................................................................... 13
FIGURE 4-1 VIDEO4LINUX API................................................................................... 15
FIGURE 4-2 IMAGE CAPTURE LOOP............................................................................ 17
FIGURE 5-1 NOISE REDUCTION BY AVERAGE FILTER.............................................. 26
FIGURE 5-2 TUNNEL ENTRANCE AND ROI............................................................... 26
FIGURE 5-3 ROI AND SUB-SAMPLING WINDOW...................................................... 27
FIGURE 5-4 CONTRAST ENHANCEMENT FROM SMALL LUMINANCE RANGE TO FULL RANGE 27
FIGURE 5-5 CODESIGN SYSTEM ARCHITECTURE.................................................. 28
FIGURE 5-6 CONTRASTENGINE HARDWARE IMPLEMENTATION BLOCK DIAGRAM... 29
FIGURE 5-7 MEMORY ARBITER’S FUNCTION VIEW................................................... 30
FIGURE 5-8 MEMORY ARBITER BLOCK VIEW........................................................... 31
FIGURE 5-9 CONTRASTENGINE CONFIG PARAMETERS.......................................... 32
FIGURE 5-10 CONTRASTENGINE CONFIG REGISTER BLOCK VIEW...................... 33
FIGURE 5-11 CONTRASTENGINE MASTER UNITE BLOCK VIEW............................ 34
FIGURE 5-12 CONTRASTENGINE MASTER STATE MACHINE................................. 35
FIGURE 5-13 CONTRASTENGINE ADDRESS COUNTER BLOCK VIEW.................... 35
FIGURE 7-1 PERFORMANCE MEASUREMENT OF EXECUTION TIME...................... 42
FIGURE 7-2 PERFORMANCE MEASUREMENT OF CPU LOAD................................... 43
# Table Index

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 3-1 XILINX ML310 PCI REFERENCE DESIGN IP LIST [1]</td>
<td>12</td>
</tr>
<tr>
<td>TABLE 5-1 CONTRAST ENGINE CONFIG REGISTER MAPPING</td>
<td>32</td>
</tr>
</tbody>
</table>
Bibliography

[1] Xilinx ML310 Embedded Development Platform, at URL:

[2] Xilinx ML310 PCI Reference Design, at URL:
http://www.xilinx.com/products/boards/ml310/current/index.html#pci

[3] Video 4 Linux API, at URL:
http://www.exploits.org/v4l/

[4] Linux Kernel Source Documents, available with every Linux installation

[5] Philips USB Webcam Driver for Linux, at URL:
http://www.saillard.org/linux/pwc/

http://www.xml.com/ldd/chapter/book/

[7] Simón, J.: ”Hardware implementation and verification of the AddressLib video
Library”, Universidad Politécnica de Valencia, 2005

[8] Stechele, W.; Herrmann, S.: ”Reconfigurable Hardware Acceleration for Video-
based Driver Assistance”, Technische Universität München, 2005

[9] Xilinx Core Connect Architecture – Processor Local Bus

[10] Xilinx Virtex II Pro FPGA, Datasheet DS083, at URL:

[11] Xilinx Datasheet DS448 on the PLB IPIF, at URL:
Copyright

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/