Linux Device Driver for Matrox Meteor-II Framegrabber

Marcus Furlong

B.A. (Mod.) CSLL

Final Year Project May 2003

Supervisor: Fergal Shevlin

Abstract

Linux Device Driver for the Matrox Meteor-II / Multi-Channel Framegrabber

Marcus Furlong

This report describes work carried out to write a device driver for the Linux operating system. The device driver is for the Matrox Meteor-II /Multi-Channel framegrabber. A device driver already existed for this framegrabber, but was still in a development state when the project was initially undertaken. It was also virtually unusable to those without the necessary skills to program a client application for it in c. This report discusses theory of the linux kernel, the work involved in writing a device driver for linux, and how the device driver was written and improved upon.

"I have not failed seven hundred times. I have not failed once. I have succeeded in proving that those seven hundred ways will not work. When I have eliminated the ways that will not work, I will find the way that will work."

Thomas Edison

Preface

The scope of this project was much larger than I had initially thought and a number of different areas had to be researched. I try to provide a chapter by chapter analysis of the various different subject areas, indicating related topics I have learned also along the way. I would hope that this report may also prove useful for anyone who wishes to undertake a similar project. Subject areas that are addressed include the following: image capture devices and their capabilities, the linux kernel and how it enables interaction between the hardware and the user, and how to write a device driver. A good working knowledge of the c language is also needed as well as a basic knowledge in the use of a UNIX-like system and the usual associated development tools.

It should be noted that the author of the initial driver had access to technical documents that I did not have access to. He was also unable to divulge any information contained therein due to an NDA he had signed with Matrox. Unfortunately I did not realise that such documentation would be needed and did not apply to Matrox for the necessary information until it was too late. Anyone reading this report with a view to undertaking a similar project should apply to the appropriate hardware manufacturer as early as possible.

Organisation of the report

The content of this report has been divided up into the following chapters.

Chapter 1 provides an introduction to the subjects of the report, and gives an account of what my goals were in undertaking this project.

Chapter 2 looks at the subject of image capture devices, different image formats that are used, and the underlying hardware principles which must be understood to provide a means of communicating with hardware via software. I also describe the initial steps I took in this project, i.e., assembling the necessary hardware.

Chapter 3 provides an account of what I learnt of linux and of the basics of the linux kernel. It also covers the subject of how to communicate with hardware via the kernel through the use of device drivers.

Chapter 4 looks at the development environment and tools that were used to build the various modules, and discusses some advantages and disadvantages of developing kernel code with these tools. This chapter builds on the concepts introduced in the previous one, and introduces the tools used in both the design and implementation phases.

Chapter 5 takes a closer look at the design options available and the various aspects of kernel modules and linux kernel multimedia API that are relevant to the design of a device driver module.

Chapter 6 describes the actual implementation and investigates how the original design needed to be adapted somewhat, as well as looking at the code, and describing further details of the vl42 API.

Chapter 7 concludes this report by analysing the different areas involved in this project and looking at future development of the linux kernel, in particular the multimedia orientated parts and also looks at other future directions this project may take.

Note on Convention s and Styles

When referring to the framegrabber board used in this project, I will refer to the Meteor-II/MC, whereas when referring to the device driver (or kernel module depending on the context) I will refer to 'meteor2'.

When referring to the different multimedia APIs for linux, v4l refers to video4linux API, old-v4l2 refers to the deprecated video4linux2 API, and new-v4l2 (or simply v4l2) refers to the newer version of video4linux2 that was used to develop the driver this report is documenting.

I have used some different fonts to distinguish between different types of text, provide some clarity and improve the overall readability. The following table summarises the differences:

This font is used for normal text.
`This font is used for code, referring to objects in code or the output of a command.`
`This font is used to indicate a command input by the user.`

Also, anywhere I use four dots (`....`) in the output of a program indicates a portion of the output that could not be included due to practical considerations (e.g. where the output would take up many pages).

All references are contained in footnotes, and if the same reference is used twice on any page, it is referred to by Ibid.

Contents

Chapter 4 - Development Environment

Chapter 5 - Design

Chapter 1

Introduction

1.1 Background

The Matrox Meteor-II framegrabber is currently unsupported by Matrox when using the linux operating system. A device driver exists, but it is still being developed and it does not offer adequate functionality. This report describes my attempts to write a device driver for this framegrabber device, to enable its usage under linux. As an introduction I will present some basic facts about device drivers and linux.

1.1.1 What is linux?

Linux is a clone of the UNIX operating system and was initially designed to run on x86 system. The history is as follows - “Linux is an operating system that was initially created as a hobby by a young student, Linus Torvalds, at the University of Helsinki in Finland. Linus had an interest in Minix, a small UNIX system, and decided to develop a system that exceeded the Minix standards. He began his work in 1991 when he released version 0.02 and worked steadily until 1994 when version 1.0 of the Linux Kernel was released. The current full-featured version is 2.4 (released January 2001) and development continues.”[1]

1.1.2 What is a device driver?

A device driver is a piece of software that enables the operating system to control the functionality of a piece of hardware.

1.1.3 What is a framegrabber?

A framegrabber is an image capture device. I will elaborate on this in the next chapter.

1.2 Motivation

There are many reasons why I undertook this project in particular. I was only introduced to the UNIX system when I came to university and since then it has become a constant source of wonder, as my previous computing experience had primarily been with DOS-based systems. With the widespread attention that linux began receiving, it naturally followed that I would become interested in it. It allowed me to use a UNIX-like system and gave me access to the source code and ability to modify it, whether just to learn or to create something constructive and useful. It also had the gave me benefit of being able run it on my own

1 www.linux.org website

computer, thus giving me even more control over the system, whilst allowing me to reap the benefits of running a UNIX-like system on affordable hardware.

Since my first interest in linux I have become more and more interested in the operating system as a whole and how various parts of the computer interact with each other. Having never studied hardware nor operating systems as part of my degree curriculum, the details of how hardware interacts with software has always remained a somewhat grey area for me and I have never had the chance to investigate fully the relationship between the two. This project allowed me to further my knowledge in this respect, and also allowed me to learn the underlying elements of an operating system that I enjoy using. I also feel that that I owe a certain debt to the millions of developers of both applications and the kernel for the GNU/Linux operating system. I have reaped the benefits of a free operating system without contributing, and in undertaking this project, not only am I furthering the development of a useful device driver so that others may benefit from its functionality, I am also providing my own contribution (tiny in comparison to what many have contributed, but a contribution nonetheless) back into what is referred to as 'the opensource community'.

My own personal motivation for choosing this project is that I spent £100 on a webcam[2] for my computer and consequently found out that it was unsupported under linux. I then discovered the extent to which a lot of hardware is not supported under linux and became aware of the need for more people to learn how to write device drivers so that others may reap the benefits. Although I bought it four years ago, I still have the same webcam and it remains unsupported under linux, However, using the knowledge I have gained from this project, I hope to be able to remedy that situation and add another 'works under linux' device to the list.

1.3 Aims

My aims for this project are simple. I wish to provide as much of the functionality that the framegrabber board is capable of to the user of the linux operating system. Specifically I wish to design a device driver that follows the appropriate standards and will thus work as flawlessly as possible for those who will eventually profit from its existence.

2 This is not the same type of device as the camera/framegrabber combination described in this project, but there are many similarites from the perspective of programming a device driver.

Chapter 2

Image Capture

2.1 Introduction

This chapter provides an analysis of the various different aspects of image capture that I needed to study before proceeding with this project. The relevant image and video formats that are used are presented here, along with a description of how the hardware performs various manipulation operations on data to obtain these different formats.

2.2 Image Capture

Image capture on a computer takes place through the use of a framegrabber device with a camera attached. The framegrabber generally fits into one of the buses in a computer and the camera is connected to the framegrabber. This section explains the process by which an image is transferred from the camera to the computer's memory, via a framegrabber. It includes some basics of video capture including a review of how the Meteor-II/MC operates. An understanding of these fundamentals is prerequisite to writing a functional device driver.

2.3 Video Format Standards

There are many different video formats, and each camera (especially analog cameras[3]) needs to send its image information in one of these predefined formats, in order that it may be understood by the framegrabber. The formats describe, amongst other things, the number of pixels and number of lines of lines that the camera will send, whether the signal is interlaced[4] or not, and the frequency of the line scans. The following are the common North American and European standards.

2.3.1 RS-170 and NTSC

RS-170 is the North American monochrome video format standard and NTSC is the North American colour video format standard. NTSC is a modified version of RS-170 that contains colour information. Both are interlaced analog signals. The video format is 640 pixels x 480 lines.

3 Digital cameras may send a digital signal to the framegrabber. This means that the framegrabber does not need to digitise an incoming analog signal.

4 An interlaced signal is one where the odd numbered lines are scanned first, then even-numbered lines.

2.3.2 CCIR and PAL

CCIR is the European monochrome video format standard and PAL is the European colour video format standard. PAL is a modified version of CCIR that contains colour information. Both are interlaced analog signals. The video format is 768 pixels x 572 lines.

2.4 Camera

The camera used during the project was the Hitachi KP-M1, a standard CCIR camera. This type of camera was chosen as it is a basic camera and would confirm whether the basic functionality offered by the driver actually worked. However, a connection from this camera to the framegrabber alone does not suffice to obtain image from the camera. A framegrabber's camera interface consists of a cable and camera configuration file. The camera configuration file configures the framegrabber to receive a single from a certain type of camera. The file used was a standard CCIR DCF (Matrox Digitizer Configuration File). The DCF also contains other types of configuration information for the framegrabber and some of these will be examined later. The construction of the second part of the camera interface (the cable) is described in the following subsection.

2.5 Cable

The camera can be powered via an external source that connects the power supply directly to the camera or via the framegrabber. Throughout the course of this project, it was powered through the framegrabber; this required one extra wire in the cable between the camera and framegrabber. The cable had a 12-pin connector at the camera end and a 44-pin connector at the framegrabber end. The following diagrams show the different pins on both the camera connector and the framegrabber connector.

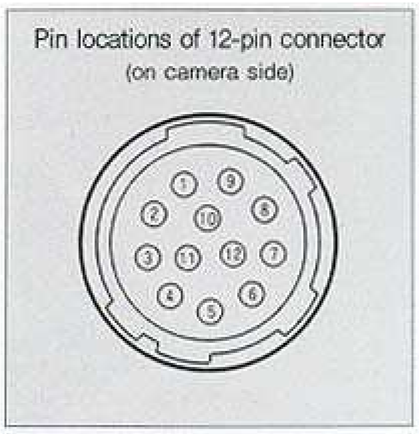


Figure 3.1 Camera Connector

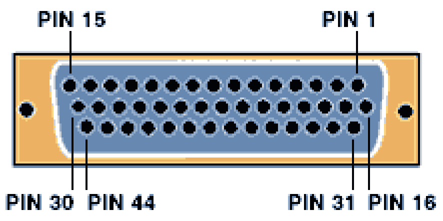


Figure 3.2 Framegrabber Connector

It was necessary to decide what pins on the camera connector were needed to send the signal, and also what pins on the framegrabber were needed to receive the signal. Having chosen the appropriate pins on each connector I then soldered the correct wires to the corresponding pins. The following table shows the pins that I used on each connector. Each row of the table corresponds to a wire connection made between the pins. There is only one video output signal on the camera connector so only one of the six video input channels on the framegrabber was used. Appendix C contain both full tables of the different pins available on each connector.

Camera Connector		***Matrox Meteor-II /MC Connector***		
Pin No	**Signal**	**Pin No**	**Signal**	**Description**
1	GND	25	GND	Ground
2	+12V	1	DC POWER	+12V Power Supply
4	Video Output (Signal)	15	VID1_IN1	RED Analog Video Input (Channel 1)
5	GND	17	GND	Ground

Table 3.1 Connections used between camera and framegrabber[5]

2.6 Framegrabbers

This section describes the functionality of a framegrabber, and using the Meteor-II/MC as an example, looks at some of the various operations which data sent from the camera undergoes.

2.6.1 What is a Framegrabber?

A framegrabber is a piece of hardware designed to capture an image from a camera. It is an interface board which digitises the analog signal sent by a camera. The camera outputs data as an analog video signal as described above (CCIR or RS-170), and the framegrabber captures the video data and sends it to the computer's memory. The framegrabber is also responsible for subsampling the data, which involves changing the incoming datastream into a form which is better for viewing or processing, or performing specific user-requested operations on the datastream. Typical operations a framegrabber can perform include the use of look-up tables to convert the image to a standard image format, gain-control (gain is an amplification of the signal values and can be either negative or positive) and region-of-interest selection. These different operation will be described in more detail later.

5 Tables adapted from WEB-Matrox and WEB-Hitachi

2.6.2 Matrox Meteor-II /Multi-Channel

The framegrabber used was the Matrox Meteor-II /Multi-Channel. The specific model used was the PCI model, with no additional modules. This framegrabber is described as a “monochrome and component RGB analog framegrabber for standard and non-standard video acquisition”[6]. The following diagram provides the schematics of the Meteor-II/MC board, and should serve as a guide for the descriptions of some of the subsampling operations that are carried out on the signal from the camera.

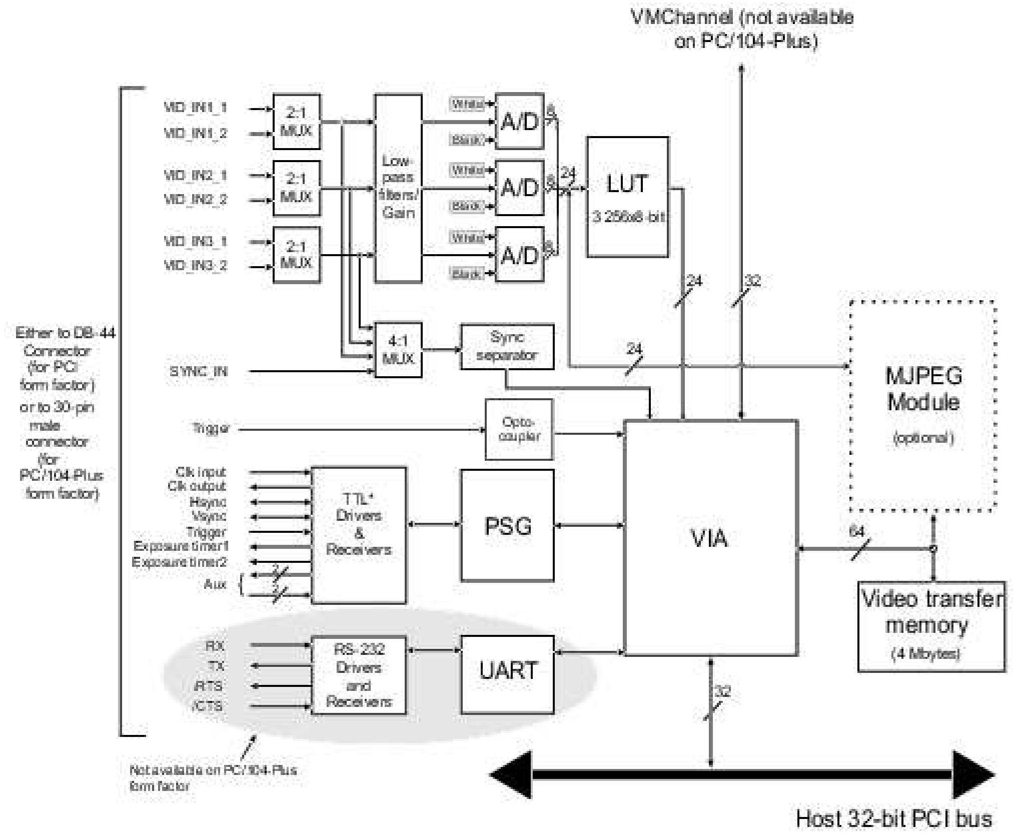


Figure 3.3 – Meteor-II /Multi-Channel Schematic

2.6.3 Signal Processing

This section examines the various different operations that determine how the analog signal is converted to digital data before being copied to PC memory[7] for processing by the software programs, which are usually referred to as client applications.

6 WEB-Matrox

7 I will refer to PC memory throughout this report, as my work was carried out using a PC. However, it is conceivable (though so far untested) that the framegrabber and the linux driver would work on any system that is supported by linux and that has a PCI bus.

2.6.3.1 Subsampling the Video Data

The video data is received from the camera via video input channel VID_IN1_1, and initially passes through the onboard low-pass filter (a 4th order Butterworth filter with a cutoff frequency of 10MHz) which serves to eliminate noise that may occur due to a low-quality or too long cable, or due to electrical interference. Gain-control is then applied to the data and subsequently the software controlled black/white references for the each of the RGB channels is applied. The DCF will specify the default values for gain and black/white references, but it is possible that a client application can change these defaults during operation.

2.6.3.2 Look-Up Tables

The data then uses the look-up tables (referred to here as LUTs), which convert a set of input pixel values to a set of output pixel values by using a mapping function built into the hardware. The Meteor-II/MC has three 8-bit LUTs, which means that for each set of pixels that the LUT operates on, it can output 256 different shades of that colour. The LUT uses the pixel value to determine the address location within the table, and provides the contents of that address as the pixel. This effectively means that the LUTs take the signal and convert it to some standard image format, enabling client applications to receive the digitised images in a format they can understand. The LUTs are capable of outputting values in either RGB 8:8:8, YUV 5:5:5, or YUV 5:6:5 formats. These formats can be also be described by their corresponding 'FOURCC' codes. FOURCC (Four Character Code) formats are “a means to identify a video datastream format unambiguously”[8]. This is particularly useful from a programming perspective as the details of how the format is composed remain relatively transparent. However, as an example I will briefly explain one one of the formats. RGB 8:8:8 means that each pixel is represented by 24 bits and the pixel data in memory would look similar to this:

Red component	Blue component	Green component
10111110	00001010	11010000

RGB 8:8:8 is a packed-pixel format (meaning that all the data for the pixel lies next to each other in memory) and the YUV formats can be “packed formats where Y, U (Cb) and V (Cr) samples are packed together into macropixels which are stored in a single array [or] planar formats where each component is stored as a separate array, the final image being a fusing of the three separate planes”[9].

8 WEB-FOURCC
9 Ibid.

The fine details of these various formats are needed by client applications, as they must know what format the image sent from the framegrabber will be in. This format is specified by the user via the DCF, and the LUT will perform the necessary conversion.

To simplify the interface between driver and board, most communication is done through one chip on the board. This chip is called the VIA (Matrox Video Interface ASIC). The next section deals with the VIA and specifically discusses its importance with respect to the overall functionality of the board and also looks at how device drivers need to communicate with the VIA.

2.6.3.3 The Video Interface ASIC

The VIA is central to the operation of the Meteor-II/MC board. An ASIC is an Application Specific Interface Circuit, which is basically a chip that integrates the different circuits on the board and provides a uniform interface to them. The VIA acts mainly as a bridge to the PCI bus and is “capable of high-speed image transfers to Host memory or other PCI devices across the PCI bus. It uses 4 Mbytes of SGRAM[10] (on-board memory) to store data until the PCI bus becomes available. Matrox VIA can manage up to two simultaneous data streams. For example, it can grab into SGRAM, and concurrently transfer data over the PCI bus.”[11] Thus the VIA manages the transfer of the now suitably encoded datastream from the onboard image buffers to the PC memory via the PCI bus.

The next section investigates some other important topics relevant to this project, the FPGA of the Meteor-II/MC card and an operation called scatter-gather.

2.6.4 FPGA

The Meteor-II/MC card contains an FPGA (Field-Programmable Gate Array), which is a technology for making integrated circuits. The key difference from other chip technologies is that it is field programmable, meaning that the chip can be programmed (and reprogrammed) without needing an external manufacturer[12]. Each variant of the Meteor-II family of boards loads specific microcode into its FPGA to allow the board to function correctly. This has two evident side-effects; it allows for easier bug-fixing of the internal programming of the entire board for the manufacturer, but it also limits the number of people who can actually write a working device driver as the board will not function without the appropriate code loaded into the FPGA.

10 SGRAM - Synchronous Graphics RAM, clock-synchronized RAM that is used for video memory.
11 WEB-Matrox
12 Typically the chips programming is stored in microcode that is loaded onto the FPGA chip via software.

2.6.5 Scatter-Gather

To use the Meteor-II/MC boards it is necessary to set aside an area of memory when the operating system is starting. This memory cannot be then used by any other application. It is done by adding an option to the bootloader. In Windows the option is /MAXMEM=xxx and under linux the option is MEM=xxx, where xxx is the difference between the total memory the system has and the amount to be available to the meteor2 device driver. This memory area is not fragmented and is thus said to be a contiguous block of memory.

However, this approach reduces the amount of memory available on the system, and requires a reboot if all the system memory is to be used for normal purposes. To avoid this approach, there exists a technology called scatter-gather. Scatter-gather allows the framegrabber to access memory that is discontiguous (i.e. fragmented) as though it were contiguous. When the framegrabber no longer needs the memory, it can simply free it, to be used by other operations. This results in a more efficient use of system memory.

There are conflicting reports as to whether the Meteor-II/MC supports scatter gather[13], so I initially started this project based on the assumption that the card supported scatter-gather.

2.7 Summary

I have presented here the basic hardware-related information that was needed to understand the internal workings of a device driver for the Meteor-II/MC interface board. An analog signal is sent from the camera to the framegrabber via the cable, and subsampling operations serve to convert the image into a format recognisable by the client application. The VIA oversees the transfer of this image from onboard memory to the PC memory where the client application receives it. The details pertaining to the configuration of the board and the order in which these operations take place are controlled by the device driver, which is essentially a device-specific extension of the operating system. In the next chapter I will look at the different aspects of the linux operating system that I needed to understand before proceeding to write a device driver.

13 There is a PDF with the specifications for the Matrox Meteor-II that states numerous times that the board supports scatter-gather. I have included this on the second disk for reference. The revised PDF specification sheet on the Matrox website makes no mention of scatter-gather.

Chapter 3

Linux Kernel Theory

3.1 Introduction

This chapter examines in greater detail various aspects of linux kernel theory. I look at how the kernel is conceptually built, and how communication within the kernel takes place, whilst attempting to explain exactly how a device driver must integrate itself into this system.

3.2 The Linux Kernel

The linux kernel is a subject central to this project. It is necessary to understand key issues related to the kernel and how it operates. In this section I will take a closer look at the necessary theory and what I have learned about kernel operations as a result.

3.2.1 The kernel

Each computer system includes a basic set of programs called the operating system, the kernel being the most important subset. It is loaded into RAM when the system boots and contains many critical procedures that are necessary for the system to operate. The kernel provides key facilities to everything else on the system and determines many of the characteristics of higher-level software. Hence, the term "operating system" is often used synonymously with "kernel", whereas in reality the kernel is the core of the operating system.

The difference between the concepts of userspace and kernelspace is also important. The difference is clarified thus - “the kernel executes in the highest level (also called supervisor mode), where everything is allowed, whereas applications execute in the lowest level (the so-called user mode), where the processor regulates direct access to hardware and unauthorised access to memory. We usually refer to the execution modes as kernelspace and userspace.”[14]

3.2.2 Hardware control

All the hardware (referred to as devices here) can only be accessed by the user via the kernel. The operating system abstracts the real hardware of the system and presents the system's users and its applications with a means of accessing the hardware. Device drivers

14 RUS01, pp 19

make up the major part of the linux kernel. Like other parts of the operating system, they operate in a highly privileged environment (kernelspace) and control the interaction between the operating system and the hardware device that they are controlling.

3.2.3 Kernel interface abstraction

The linux kernel often abstracts its interfaces. An interface is a collection of functions and data structures that operate in a particular way. Typically an interface will extract features that are common to a specific group of devices. For example, all video device drivers have to provide certain functions in which particular data structures are operated on. For example, a generic function might be called `get_image_from_device()`. This function would perform any generic setup operations and then call a device specific function to do the actual work. This way there can be generic layers of code using the services provided by the lower layers of device-specific code. The video device layer is generic and it is supported by device-specific code that conforms to a standard interface. This allows some level of transparency for the user, who needs only call the generic function. This type of interface is generally referred to as an Application Programming Interface (API).

3.2.4 The Video4Linux API

In this project the interface abstraction that is used is the video4linux2 API. It is a reworking of the original video4linux API, which provides generic routines to access multimedia devices and perform operations on them. This allows client applications to access a device without the need to use a different set of functions for every different device, which simplifies the writing of both client applications and device drivers. Client applications access the device via the v4l2 API and do not need to know anything else about the device, whereas device drivers can be written in a somewhat uniform fashion, following the v4l2 standards. This also provides a framework upon which device driver writers can start building a driver. I will discuss the details of the programming interfaces provided by this API in chapter 5.

3.2.5 Kernel Loadable Modules

The linux kernel is a monolithic kernel, i.e., it is one single large program where all the functional components of the kernel have access to all of its internal data structures and routines - “virtually any procedure can call any other procedure.”[15] However, the traditional UNIX monolithic kernel has some inherent problems - “such lack of structure was unsustainable as operating systems grew to massive proportions.”[16] The alternative which aims to rectify this lack of structure is the micro kernel, where the functional pieces of the kernel are broken out into units with a strict communication mechanism between them - “only absolutely essential core operating system functions should be in the kernel, [...] operating

15 STA01, pp 173
16 Ibid.

system components external to the micro kernel are implemented as server processes [and] interact with each other on a peer basis, typically by means of messages passed through the microkernel."[17] Although some would argue that the microkernel is a better approach, there are some performance issues - "it takes longer to build and send a message via the microkernel, and accept and decode the reply, than to make a single system call."[18] This means that the monolithic kernel remains the viable solution. However, adding a new component into a traditional monolithic kernel necessitated a complete rebuild of the kernel, which is both time consuming and impractical for debugging purposes.

Thus, the alternative of the linux loadable kernel module was born. A loadable kernel module is an extension to the base kernel that can be dynamically loaded and unloaded at any time after the initial loading of the base kernel into memory at boot time[19]. Typically modules are device drivers, or pseudo-device[20] drivers such as filesystem or network drivers. They add extra functionality to the running kernel and are dynamically linked to the running kernel. They can be unlinked from the kernel and unloaded from memory when no longer needed. Modules also have the same rights and responsibilities as the kernel code and therefore extreme care needs to be taken when coding. An unstable or badly written module can effectively render the system unstable. For example, if the module enters a loop from which it cannot exit or accesses a memory region that it should not have, the fact that the module is a part of the kernel means that the kernel has now become unstable.

3.3 Character Devices

Unix systems in general provide access to hardware via special files called device pseudo files, which reside in the `/dev` directory. The kernel redirects operations on devices via these special device files. There are different types of device files, but in general the basic functionality needed by any device can be found by again abstracting devices into two specific categories; block devices and character devices.

A character device is basically a stream of bytes[21] and the kernel performs its operations on this stream of bytes. The operations are performed on one single byte at a time. Block devices are similar to character devices but have extra functionality and are typically devices where faster I/O is needed and blocks of bytes are accessed at a time e.g. hard disks, etc.

A video4linux device is at the most basic level a character device, and the kernel performs operations on stream of bytes to convert them into a meaningful representation for users (in

17 STA01, pp173-174
18 Ibid., pp 176
19 Loadable kernel modules are normally referred to as LKMs or simply modules.
20 Pseudo refers to the fact that these are not physical devices.
21 Traditionally a character was one byte long. Thus a stream of bytes was a stream of characters.

this case an image or a stream of images). This conversion takes place both on the board (which is controlled by the device driver) and at a higher level in the v4l2 API, which provides the userspace application with a stream of bytes that it must convert to representation useful to the user. Thus the Meteor-II/MC framegrabber is treated as a character device, identified as and accessible via `/dev/video0` (if there are no other multimedia devices registered with the kernel). The next section deals with how the kernel decides what device node will represent what device and how device nodes are allocated if there are multiple devices.

3.4 Major / Minor Numbers

The kernel differentiates between the various devices in the `/dev` directory in a different way from that which the visual representation of devices might suggest. A listing of the `/dev` directory might contain entries such as:

```
root@collins$ ls -l /dev

crw-rw----    1 root     audio      14,   4 Mar  2 12:41 audio0
crw-rw----    1 root     audio      14,  20 Mar  2 12:41 audio1
crw-rw----    1 root     audio      14,  36 Mar  2 12:41 audio2
crw-rw----    1 root     audio      14,  52 Mar  2 12:41 audio3
....
brw-rw----    1 root     daemon      3,   0 Oct 21 23:12 hda
brw-rw----    1 root     daemon      3,  64 Oct 21 23:12 hdb
brw-rw----    1 root     daemon     22,   0 Oct 21 23:12 hdc
brw-rw----    1 root     daemon     22,  64 Oct 21 23:12 hdd
....
crw-rw-rw-    1 root     root       81,   0 Mar  7 12:59 video0
crw-rw-rw-    1 root     root       81,   1 Mar  7 12:59 video1
crw-rw-rw-    1 root     root       81,   2 Mar  7 12:59 video2
crw-rw-rw-    1 root     root       81,   3 Mar  7 12:59 video3
```

The names `audio`, `hd`, and `video` are merely names to help the user and userspace programs identify the devices, whereas the kernel identifies the devices via major and minor numbers. In the above example the major and minor numbers are in the fifth and sixth columns respectively.

The `audio0`-`audio4` nodes are character devices (note the c in the very first column of the permissions) which are accessed via major number 14 and minor numbers 4,20,36 and 52 respectively. The `hda` and `hdb` devices have different major numbers from the `hdc` and `hdd`, showing that they are block devices of a different type. The fact that both start with `hd` indicates to the user that they are similar devices, but due to the different major numbers, a different kernel interface is used to communicate with them. In fact these devices represent hard disk drives and compact disc drives. The major number, coupled with whether it is a character or block device, tells the kernel what type of device it must interface with. The minor number typically identifies the device to the generic driver, so it can identify specifically what model or type of device it is[22].

22 The file `Documentation/devices.txt` in the linux kernel sources lists many device types by major number,

In the above example, `/dev/video2` tells the kernel is a v4l or v4l2 device (major 81). For v4l2 devices minor numbers are attributed to device in a arbitrary fashion, and it usually the system administrator who decides what minor numbers to use with what devices. A configuration file then tells the kernel what module needs to be loaded for a given minor number. Both major and minor numbers are no longer being distributed by Linus (the newer devfs filesystem is hoped to eventually replace them) but the maintainers of the various device subsystems of the kernel unofficially use them to simplify device idenentification.

The Meteor-II/MC has had no specific device minor assigned to it so I have been accessing it via the device node /dev/video0 (81,0). However, I could use any minor number and specify it via a configuration file or pass it as an argument to the module. The command to create a new node for a video device (e.g. with major 81 and minor 1) would be as follows:

```
root@collins$ mknod /dev/video0 c 81 1
```

The numbering at the end of the device name (0 in the above example) usually serves to identify either the physical location of the device or the order in which the device was found. It differs for different types of devices and for v4l2 devices it is not very significant. A client application will try to access `/dev/video`[23] first and then try `/dev/video0`-`/dev/video3`.

To summarise, major numbers typically identify what type of device is being accessed, whereas the minor number typically identifies what specific device it is (and thus tells the kernel what device driver to load). How the linux kernel communicates via this pair of numbers and how it affects the API that is used will be discussed in the following section.

3.5 Communication with devices via an API

The kernel loads the appropriate subsystem (either compiled into the kernel or loaded as a module) after querying the devices major number. The subsystem (in this case the `videodev` module) then checks the minor number of the device and loads the appropriate submodule (in this case the `meteor2` module). The `videodev` module provides generic functions for accessing the device which act as wrapper for the functions that do the work in the meteor2 module. The device is treated as a `file` by the kernel and the other submodules. A `file` in this case does not refer to the typical userspace concept of a file, rather it refers to a specific kernelspace data structure that identifies the device and is passed to any functions that

and shows what the relevant minor numbers refer to.

23 Usually a symbolic link to the default device.

operate on the device. Typical operations on a character device include open, close, read, write, poll, and ioctl. I will explain these operations in more detail in the chapter concerning kernel and module programming.

3.6 The proc filesystem

The proc filesystem (`/proc`) provides a means for the kernel to communicate information about the internal state of its structure. From a programmer's point of view, it is a tool that can display any of the values of the variables contained within a running kernel process (i.e. a module). From the documentation with the kernel sources - “The proc filesystem acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime”. The proc filesystem is used by the driver as a means of communicating details about the meteor2 module. It can also be used for debugging purposes as will be explained later.

3.7 Summary

The important aspects of kernel that theory were necessary for this project included an understanding of the kernel and its functions on a computer, especially hardware control. The kernel's abstraction layers help both kernel and userspace programmers by providing standard access points to the kernel. The abstraction layer that was most useful to me in this project was, of course, the video4linux2 API. Also needed was an understanding of character devices and how their major and minor numbers are used by the kernel to communicate with hardware, and exactly how that communication takes place via operations on a kernelspace structure called a `file`.

Having learnt the theory behind what I was trying to achieve, I then realised that even though there was an API to help me, I would still need to come up with a solid design on which I could build and start writing the code, and I would have to learn to use various development tools. In the next two chapters, I identify what aspects of the design of the module were considered before programming, and look at the different tools used during the course of the development.

Chapter 4

Development Environment

4.1 Introduction

This chapter outlines the operating system version and various tools that were used during the course of this project, and gives reasons for the various choices made.

4.2 Distribution and Kernel Version

The distribution was not a standard distribution, but rather a custom linuxfromscratch system. Many of the major distributions' methods of upgrading packages cause problems due to dependencies on other packages. I felt that a self-built system would be a better environment, because most of the packages required for development would already be the latest versions. Also if I needed to upgrade a package or wanted to try a new debugging tool, fewer dependency problems would arise.

Using an up-to-date system of this nature also meant that I could immediately take advantage of any new functionality that had been added to any package.

The kernel used was version 2.4.20, with the v4l2 patch[24] applied, and the v4l2 interface built as a loadable kernel module. This is the latest stable release of the kernel at this time. Also, the new v4l2 interface has been accepted into the 2.5 development kernel, which means it will be part of the next stable kernel release, which will be 2.6. This means that the driver from this project should be loadable under kernel 2.6 with minimum modifications.

4.3 Programming Language

The programming language used was c. The kernel is written in c because it allows many code optimisations not permitted by other programming languages, it also allows low-level access to devices and it is the language UNIX systems have traditionally been written in.

24 WEB-NV4L2

4.4 Development Tools

This section lists the various development tools which proved useful while attempting this project.

4.4.1 Compiler

The compiler used was gcc version 3.2.2 from the GNU Compiler Collection. I opted to use the latest available version of the compiler as it tends to be stricter on various syntactical constructs in c, which will allow the code to compile with more success using future versions of gcc.

4.4.2 Make

Make is a program which automates the compilation and linking of different parts of a project. The version of make was GNU make version 3.80 and the makefile rules used were those of the compiled linux kernel[25].

4.4.3 Debuggers

Debugging kernel code is somewhat more difficult than debugging normal code. Kernel code does not execute in userspace and so needs a specialised debugger. Using the standard GNU debugger gdb is possible by applying a patch to the running kernel, and after recompilation, loading the entire kernel into the debugger, but I was unable to obtain any satisfactory results using this method[26], and so tried other methods. The following are the various different techniques and programs I attempted to use, and the results of each.

4.4.3.1 strace

Strace is a program which provides a trace of system calls, which is useful in that it provides a manner of viewing how and where exactly in code the system calls are made. Its usefulness is limited but it proved useful in certain contexts. For example, it was possible to examine every system call that was made when attempting to load a module onto the kernel, and to review the corresponding return values. Also, this program can be used to examine if a userspace program is making too many system calls that are slowing down execution. This information can in turn be used to determine where to perform optimisation of the code. The version of strace used was 4.4.

25 The file `Rules.make` in the `/usr/src/linux` directory.
26 Specifically, I was unable to find determine how to load a module into a kernel that was being debugged.

4.4.3.2 printk

The kernel provides a system call to write to the kernel log. This can also be redirected to standard output, i.e., the terminal. This is analogous to using cout in c++ or printf in c, and code can be inserted into the program at certain points. There also exists a system of different levels of priority for each of the messages. This can be useful as the level of debugging required can be passed as flag to the module and the output will be changed accordingly. The only problem is that this tends to add extra memory overhead and causes the module to take longer to load. Production versions of drivers tend not to include as much debugging code. However, as I found the information outputted by printk very informative and useful, much of this debugging code has been left in.

4.4.3.3 User-mode Linux

User-mode linux[27] is an impressive project in that it attempts to provide a virtual kernel running in userspace. It is loaded from the command-line and is effectively a virtual kernel running as a userspace process. This means that if there is a problem with kernel (or by proxy a problem with a module), it can be terminated as any normal userspace program would be. I successfully installed and ran user-mode linux but it proved not to be overly useful due to limitations in the virtual PCI device portion of the code. After trying to use this for some weeks I found it has the potential to be an effective kernel debugging tool, but for this project I was unable to profit fully from the functionality that it provides. It did prove useful for determining if a module could load and unload cleanly, i.e., without crashing the system.

4.4.3.4 The proc filesystem

I used the proc filesystem at various stages to keep track of different variables at runtime. Although this was mostly an informational process, it did, however, provide another useful means of debugging, as I could effectively watch the various internal variables changing at runtime.

4.4.5 Video Client Application

In order to view the image sent by board, I initially used a program that was included with the original meteor2 driver package. This client application grabs a single image and saves it to a pgm image file. While this was useful for determining that the driver actually worked, it did not allow for testing the streaming capabilities of the driver, nor was it very practical as it required the user to load an image viewing programs (e.g. the gimp) to view the image after every grab. In addition, the original driver did not correctly implement the v4l2 API, and as such did not work with other v4l2 client applications such as xawtv.

27 WEB-UML

As was stated before, one of the initial goals I set myself was to make a driver which correctly implemented this API, and as such would work flawlessly with applications designed to work with v4l2 drivers. The verbose output option of xawtv, coupled with the printk kernel output, allowed me to monitor the results of any code changes from both the point of view of the kernel and from the point of view of applications with which it would be expected to work. This approach also offered me a unique debugging perspective and provided many insights into how applications in userspace interact with code that is loaded in kernelspace.

4.4.6 Modutils

I also needed a program to load the kernel modules. The modutils package provides a suite of applications tailored to this end. This is a package developed by kernel developers to provide an easy method of loading and unloading kernel modules. The insmod program inserts a module into the running kernel, lsmod lists the currently loaded modules, and rmmod attempts to unload a kernel module from memory. Other useful programs in this suite are as follows; depmod, which checks the dependencies of a module, modprobe, which loads any other modules that the module we are trying to load depends upon, and ksyms, which displays the kernel symbols which have been exported for userspace applications or other modules may use (analogous to public functions in java or c++). Each of these programs provide a wealth of information about the status of loading or currently loaded modules, and as such also provided another means of debugging code, albeit from yet another perspective. I will now briefly examine the usage and output of some of these programs, and describe some of the unfortunate consequences associated with trying to code a linux device driver.

4.4.6.1 insmod and rmmod

insmod attempts to insert the module into the running kernel. The following is some of the output of inserting the meteor2 module:

```
root@collins$ insmod -v meteor2
xftw starting at /lib/modules/2.4.20
xftw_readdir /lib/modules/2.4.20
pruned build
pruned modules.dep
pruned modules.generic_string
pruned modules.ieee1394map
pruned modules.isapnpmap
pruned modules.parportmap
pruned modules.pcimap
pruned modules.pnpbiosmap
pruned modules.usbmap
....
type 2 /lib/modules/2.4.20/kernel/drivers/media/video
```

```
xftw_readdir /lib/modules/2.4.20/kernel/drivers/media/video
user function /lib/modules/2.4.20/kernel/drivers/media/video
user function /lib/modules/2.4.20/kernel/drivers/media/video/meteor2.o
....
Using /lib/modules/2.4.20/kernel/drivers/media/video/meteor2.o
Symbol version prefix ''
```

The output is useful in that it shows the exact order in which the program searches for modules, and shows what configuration files are parsed and the order in which they are parsed. This is useful when there is more than one module of the same name on the same system. As I was experimenting with different meteor2 modules and different v4l modules, this became a source of quite a few problems, which using the verbose output above tended to solve. rmmod is used to remove a module from memory, and if it faces any problems with unloading a module with tell the user the reason why the module cannot be unloaded. Otherwise it quietly unloads the module from memory.

4.4.6.2 lsmod

lsmod lists the currently loaded modules. Here is the typical output:

```
root@collins$ lsmod
Module                  Size  Used by    Not tainted
meteor2               365956   0  (unused)
videodev                6240   1  [meteor2]
v4l2-common             2848   0  [meteor2]
snd-pcm-oss            39236   0  (autoclean)
snd-mixer-oss          13720   0  (autoclean) [snd-pcm-oss]
radeon                108388  10
snd-via82xx            12844   1
snd-pcm                63456   0  [snd-pcm-oss snd-via82xx]
....
snd                    31332   4  [snd-pcm-oss snd-mixer-oss snd-via82xx snd-pcm snd-
timer snd-mpu401-uart snd-rawmidi snd-seq-device snd-ac97-codec]
```

This program basically outputs the contents of the file `/proc/modules`, which contains information about all loaded modules. The size of the meteor2 module is relatively bigger than other modules due to the amount of debugging code, as was discussed earlier. The used by column indicates a variable each module has called the usage count. This is incremented when the module is being used by an application or another module. A bug in the code may leave a module in a certain state where it thinks that it is being used, and its usage count is greater than 0. This is extremely important when writing a device driver module, because it is impossible to unload a module if it believes it is being used. Therefore, when the offending code is corrected, the machine must be rebooted in order to reload the module, to see if the fix worked. This is one reason why the aforementioned user-mode linux would have been an ideal solution. At one stage during development, due to bugs of this nature, I found I was rebooting the machine more than fifty times per day. lsmod indicates if the machine requires rebooting.

4.4.6.3 depmod

depmod is a program that handles the dependencies of modules. For example, the meteor2 module depends on the videodev and v4l2-common modules, and requires those modules to be loaded before it can load. Instead of issuing three separate insmod commands, depmod writes a Makefile-like configuration file indicating the interdependencies of all the modules on a system. If any changes are made to the configuration files the user must run `depmod -ae`, so that changes to global configuration file are propagated to the associated module directory configuration file. The global configuration file is /etc/modules.conf (or /etc/conf.modules on older systems) and the relevant lines in this file for the meteor2 module are as follows:

```
alias char-major-81-0 meteor2        # use minor 0
options meteor2 dmask=-1             # full debugging output
```

These lines tell the kernel that if an application attempts to access the character device file with major 81 and minor 0, it must load not only the videodev module associated with that major/minor pair, but also the meteor2 module. The second line gives the options that are to be passed to the module upon initialisation (analogous to command-line arguments to a program). depmod parses this information and writes another configuration file indicating the interdependencies based on this information in the /lib/modules/KERNEL_VERSION/ directory.

depmod can also be useful for debugging code in that it will not let write the configuration files if there are unresolved symbols. This is useful from a programming perspective as it indicates straight away if symbols in the code should have been exported or not.

4.4.6.4 modprobe

modprobe is the program that uses the configuration file created by depmod, and loads the appropriate modules, thus saving the user from having to use insmod separately for each module that needs to be loaded. For example,

```
root@collins$ modprobe -v meteor2
/sbin/insmod /lib/modules/2.4.20/v4l2/v4l2-common.o
Using /lib/modules/2.4.20/v4l2/v4l2-common.o
/sbin/insmod /lib/modules/2.4.20/v4l2/videodev.o
Using /lib/modules/2.4.20/v4l2/videodev.o
/sbin/insmod /lib/modules/2.4.20/kernel/drivers/media/video/meteor2.o
Using /lib/modules/2.4.20/kernel/drivers/media/video/meteor2.o
```

modprobe can also remove modules, and does so in the correct order. As seen above, the videodev and v4l2-common modules are used by the meteor2 module, so cannot be removed before the meteor2 module. modprobe executes the rmmod commands in the correct order.

4.4.6.5 ksyms

ksyms provides a list of the currently available exported kernel symbols, and similar to the operation of the lsmod command above, it prints out the contents of /proc/ksyms. The kernel symbols that are exported are available to all other modules, and when a module like the meteor2 module uses exported symbols from another module (i.e. videodev and v4l2-common), this is called module stacking. This program can be useful for determining if functions that should be visible outside the driver are indeed visible.

4.4.7 IDE

After initially starting the project using only command-line tools, I began to investigate the possibility of using an IDE. I installed the KDevelop 3.0 Alpha 3 IDE, and despite it still being branded alpha quality software found it to be a quite stable and feature-rich IDE which proved useful[28]. Using this and the tools mentioned above afforded me an environment in which it was relatively easy to work, although to load and unload the module, or use any of the debugging tools, it was easier to switch to and from the command prompt. This is due to the nature of kernel development is being different from normal development, i.e., the compiled program does not execute in a shell or interpreter environment, rather it is loaded into kernel memory.

4.5 Summary

I have presented here a number of the tools that I found most useful whilst undertaking this project, and whilst this list is by no means exhaustive (various other tools were tried), it is representative of those which proved to be most effective. Debugging in particular took a huge part of my time, and I learned to choose the correct tool to help understand and locate exactly where the problem lay.

28 Appendix D contains a screenshot.

Chapter 5

Design

5.1 Introduction

In this chapter I will present the various design options that were available to me and I will also discuss the loadable kernel module and video4linux2 APIs from a programming point of view.

5.2 Design Options

I realised that the design of the device driver would be different from any other code I had written. There are many differences between writing new code and writing code that is essentially maintenance of someone else's code, so I considered the various different design options available. The following is a summary of the three most viable solutions I found.

I gave myself the option of writing a new driver from scratch, following the new v4l2 API specifications, but providing the actual format of the code myself. I believed this would allow me to learn every detail of the API myself and learn problems which others had already faced and overcome.

The second design option I considered was again a rewrite from scratch but instead of following the API as a basis for the driver, I would base my code on the style of the newer v4l2 device drivers which had been rewritten to take advantage of a new PCI videobuffer API. The new API has abstracted many commonly used functions in PCI video devices which use DMA to access the buffers onto which the image is copied and provided a generic module. Using this option I could use the existing code only as a guide where necessary and would only need to copy the direct hardware access calls to the new points, where they should fit in my version of the driver.

The last approach to the design that I considered was just converting the original code so that it would be compatible with the new v4l2 API. No attempts had been made at maintaining backwards compatibility between the two versions of v4l2[29], so this would also be a reasonable approach to take. However, this approach did not take advantage of the

29 There is, however, a backwards-compatibility layer between the new v4l2 and the original v4l APIs

new videobuffer API and it left the format of the driver as it was (which from hints in the original driver code seems to be basically a port of the Windows device driver to fit into the original v4l2 API).

I will discuss the actual implementation which I undertook in the next chapter, but suffice to say that none of the above plans were as clear-cut as I had originally perceived. As is the case with many programming projects, it was necessary to change the design somewhat during the implementation. In fact, all of the three options were pursued in the order presented above and I will explain the reasons why in due course.

5.3 The Loadable Kernel Module API

In order to describe the code, it is necessary to understand the syntax and conventions used when writing a module. There are certain rules to follow when writing a module and in this section I will outline the principal functions required to write a module in order that it will load and unload cleanly. One point to note is that a program that executes in kernelspace has no access to the standard c library and as such any functions used must be declared in the kernel sources.

5.3.1 module_init (init_function)

The kernel provides a module initialisation routine `int init_module(void)` which is analogous to the c function `int main(void)`. However, the modern version allows the programmer to specify the name of the function that the kernel should execute to initialise the module. The sole argument of `module_init` specifies this. This function performs operations such as finding the device, assigning IRQs and other setup functions, such as registering with other modules, or other parts of the kernel. This registration is necessary if a module wishes to have access to the symbols exported and resources of the other module.

5.3.2 module_exit (cleanup_function)

Likewise, the `module_exit` function performs any operations that the device must do before unloading. Kernel modules must 'clean up' after themselves, meaning that any memory they have allocated throughout the course of their execution, must be de-allocated explicitly in the function specified here. Failure to do so leaves unfreed memory and may cause system instability. Devices that have registered with other modules, or any other part of the kernel must also explicitly unregister, thus freeing resources and letting the kernel know exactly what is loaded and what is not.

5.3.3 Usage Count

The usage count was touched on lightly before. The kernel keeps track other modules are using the current module code. This can be explained thus - "The system needs this information because a module can't be unloaded if it is busy: you can't remove a filesystem type while the filesystem is mounted, and you can't drop a char device while a process is using it, or you'll experience some sort of segmentation fault or kernel panic when wild pointers get dereferenced"[30]. The usage count can also be incremented and decremented manually to avoid the kernel trying to unload the module during an important operation. The macros that do so are `MOD_INC_USE_COUNT` and `MOD_DEC_USE_COUNT` respectively.

5.3.4 The `file` structure and `file_operations`

As was noted earlier, the devices use a kernelspace structure referred to as a file, and each `file` data structure is associated with a `file_operations` data structure that contains an array of pointers to the functions that control the device. The following table explains those operations which were used in the meteor2 module.

`ssize_t (*read)`	`(struct file *, char *, size_t, loff_t *)`
	Used to retrieve data from the device. A null pointer in this position causes the read system call to fail with -EINVAL (Invalid argument). A non-negative return value represents the number of bytes successfully read (the return value is a signed size type, usually the native integer type for the target platform).
`unsigned int (*poll)`	`(struct file *, struct poll_table_struct *)`
	The poll method is the back end of two system calls, poll and select, both used to inquire if a device is readable or writable or in some special state. Either system call can block until a device becomes readable or writable. If a driver doesn't define its poll method, the device is assumed to be both readable and writable, and in no special state. The return value is a bit mask describing the status of the device.
`int (*ioctl)`	`(struct inode *, struct file *, unsigned int, unsigned long)`
	The ioctl system call offers a way to issue device-specific commands (like formatting a track of a floppy disk, which is neither reading nor writing). Additionally, a few ioctl commands are recognised by the kernel without referring to the fops table. If the device doesn't offer an ioctl entry point, the system call returns an error for any request that isn't predefined (-ENOTTY, No such ioctl for device). If the device method returns a non-negative value, the same value is passed back to the calling program to indicate successful completion.

30 RUS01, Page 33

`int (*mmap)`	`(struct file *, struct vm_area_struct *)`
	mmap is used to request a mapping of device memory to a process's address space. If the device doesn't implement this method, the mmap system call returns -ENODEV.
`int (*open)`	`(struct inode *, struct file *)`
	Though this is always the first operation performed on the device file, the driver is not required to declare a corresponding method. If this entry is NULL, opening the device always succeeds, but your driver isn't notified.
int (*release)	(struct inode *, struct file *)
	This operation is invoked when the file structure is being released. Like open, release can be missing.

Table 5.1 `file_operations` used by meteor2 module[31]

Most of the explanations above suffice to understand what the operations actually do. However, the ioctl function call is slightly more complicated, as it permits interaction with the loaded module from a userspace application, in this case xawtv. For this reason, a brief explanation of its operation is required. The following section examines its usage.

5.3.4.1 ioctl

As it is the principal method of interacting or communicating with a module when it is loaded, ioctl is very important to a driver, and care needs to be taken, both when writing the driver and when writing a userspace application that will communicate using this method. The corresponding *userspace* function call is

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

The dots represent “not a variable number of arguments but a single optional argument, traditionally identified as `char *argp` [and] are simply there to avoid type checking during compilation.”[32] Although this argument can be used to pass integer values to the driver, the main way in which it is used in this project is as a pointer to data in userspace; in this way the driver has access to userspace data, such as data structures.

`fd` is a file descriptor, and the corresponding arguments of the ioctl *driver* function are the `inode` and `filp` pointers. The `cmd` variable serves as an index to tell the ioctl function what subfunction should be accessed; it identifies what the nature of the specific ioctl command is. Thus ioctl commands are usually switch statements, which compare `cmd` to the available ioctl functions, and then execute the appropriate subfunction upon successful comparison. The ioctl returns -EINVAL to the calling function if the identifier specified by cmd is unknown to the driver, or if there is an error within one of the subfunctions. EINVAL is an standard kernel

31 Adapted from RUS01, pp 63-65
32 Ibid., pp 129

integer errorcode. When the calling application receives this errorcode, it must decide what course of action to take.

The next section will examine in more detail the v4l2 API and should clarify how the ioctl command works by giving solid examples of how a userspace application would communicate with a v4l2 driver.

5.4 The v4l2 API

The v4l2 API is an abstraction layer and thus provides a uniform means to access a device via a driver or a userspace application. v4l2 is both an extension and an improvement of a previous API (video4linux) and attempts to address many shortcomings of the previous one. The functionality provided by v4l and v4l2 is implemented as a loadable kernel module called `videodev`. Both APIs are built into a set of loadable modules, referred to as the Linux Video Capture Interface. It is possible to compile these functions into the base kernel and avoid loading and unloading. However, for the purposes of this project, and taking into account the fact that the v4l2 API exists only as a patch to the current stable kernel and is still somewhat in a state of flux, it was easier to implement the addition of the functions to the kernel as a loadable module. The meteor2 driver is also implemented as a loadable module, so the concept of stacked modules, as described earlier, applies to this set of modules.

The old-v4l2 driver writer's guide is still a useful document and gives a clear introduction to the relationship between the driver modules and the `videodev` module:

"V4L2 is a two-layer driver system. The top layer is the videodev module. When videodev initialises, it registers as char major device 81, and registers its set of char driver method functions. All V4L2 drivers are really clients of videodev, and videodev calls the client drivers through V4L2 driver method functions."[33]

The actual means by which a device driver uses the v4l2 module is also important and the next section describes what the videodev module provides for its clients.

5.4.1 Driver Registration

Upon module initialisation, the driver will typically look for any devices in the system that it will manage and creates a separate `struct video_device` for each one. A pointer to each structure is then passed to the `videodev` module via `video_register_device`. The registration performs some sanity checks (checks the minor number requested is not in use, etc.) and returns 0 if registration was successful. The driver module now has access to all the helper

33 WEB-V4L2, Driver Writers Guide

functions provided by the videodev module, which handles requests from applications to access the driver and redirects requests using the pointer to the struct video_device that it has stored. The following is the format of struct video_device taken from the videodev.h file:

```
struct video_device
{
        struct module *owner;
        char name[32];
        int type;       /* v4l1 */
        int type2;      /* v4l2 */
        int hardware;
        int minor;
#if LINUX_VERSION_CODE < KERNEL_VERSION(2,5,0)
        /* old, obsolete interface -- dropped in 2.5.x, don't use it */
        int (*open)(struct video_device *, int mode);
        void (*close)(struct video_device *);
        long (*read)(struct video_device *, char *, unsigned long, int noblock);
        long (*write)(struct video_device *, const char *, unsigned long, int noblock);
        unsigned int (*poll)(struct video_device *, struct file *, poll_table *);
        int (*ioctl)(struct video_device *, unsigned int , void *);
        int (*mmap)(struct video_device *, const char *, unsigned long);
        int (*initialize)(struct video_device *);
#endif

    /* new interface -- we will use file_operations directly like soundcore does. */
        struct file_operations *fops;
        void *priv;             /* Used to be 'private' but that upsets C++ */

        /* for videodev.c intenal usage -- don't touch */
        int users;
        struct semaphore lock;
        devfs_handle_t devfs_handle;
};
```

The code contained between the #if and #endif is the old interface, that which the original driver used. The newer file_operations method still contains a pointer to the character driver functions. Before registration, the driver must fill in some of the values in this structure, notably the name, type, minor and file_operations fields. Typically the minor number is passed to the driver as a module parameter, but can also be built into the driver. As stated, minor 0 was used throughout this project, and is the default minor in the driver, without needing it to be passed as a parameter.

Similarly, video_unregister_device is called by the module cleanup function to free resources and to unlink the driver module from the videodev module.

5.4.2 Driver function calls

The driver function calls are those shown in table 5.1, and these function calls define the operation of the driver. The ways in which these function calls should be used when programming a v4l2 device are outlined by the official v4l2 API specifications[34]. The basic steps to programming a v4l2 device as put forward by the specifications are as follows:

1. Opening the device
2. Changing device properties (selecting a video and audio input, video standard, picture brightness etc)
3. Negotiating a data format
4. Negotiating an input/output method
5. The actual input/output loop
6. Closing the device

I will now summarise these various different aspects of the API.

5.4.2.1 Opening/Closing a Device

This is done via the the `open()` and `release()` calls respectively. The open call typically increments the usage count of the module, and allocates memory for a pointer to a file structure. This pointer can then be passed to other functions to identify the device that has been opened. Conversely, the release call[35] decrements the usage count and deallocates the file handle memory.

5.4.2.2 Changing Device Properties

The device properties for the meteor2 include aspects discussed earlier, such as the video standard used (PAL, NTSC), gain control, and the black/white references. The driver must have a mechanism by which the values specified in the DCF can initially be loaded onto the board and subsequently changed if necessary. These values are initially set up by the function specified by `module_init`, but can be changed via the ioctl function call.

5.4.2.3 Negotiating a data format

The data format that is sent by the video device (the framegrabber in this case) must be recognised by the v4l2 API. The default data format used by the meteor2 driver is YUV 4:2:2 and although the manual does not specify that this output is supported, it works without any problems. Typically a client application will ask the driver what formats it supports, via the ioctl function, and will select the appropriate one. The specifications explain this clearly - "[...] applications should always negotiate a data format before engaging in data exchange. Negotiation means the application asks for a particular format and the driver selects and reports the best the hardware can do to satisfy the request."[36] The following is a concrete

34 Available from WEB-NV4L2. As of this writing the v4l2 API specifications document is draft version 0.3
35 The release() call is referred to as close() in the specifications, but this has since changed.
36 WEB-NV4L2, Ch 1.8

example of how the ioctl function works and of how it passes data from userspace.

```
int v4l2_ioctl(struct inode *inode, struct file *file, unsigned int cmd, void *arg) {
        [...]
        switch (cmd) {
        case VIDIOC_G_FMT: {
                struct v4l2_format *fmt = (struct v4l2_format *) arg;
                if (fmt->type != V4L2_BUF_TYPE_VIDEO_CAPTURE)
                        return -EINVAL;
                memcpy (&fmt->fmt.pix, &card->pixfmt, sizeof (struct v4l2_pix_format));
                return 0;
        }
        [...]
}
```

The switch command uses the integer that was passed via `cmd`, and compares it to those that are known. The v4l2 API provides a large number of these unique numbers, and they are declared via `#define` statements such as the following:

```
        #define VIDIOC_ENUM_FMT             _IOWR ('V',  2, struct v4l2_fmtdesc)
        #define VIDIOC_G_FMT                _IOWR ('V',  4, struct v4l2_format)
        #define VIDIOC_S_FMT                _IOWR ('V',  5, struct v4l2_format)
```

These ones are standardised ioctls, but the driver can also provide its own 'private' ioctls, and the application can query these and 'remember' them to allow card specific parameters to be changed. The names are designed to be fairly intuitive. These three examples are the video ioctl (VIDIOC) for 'enumerate formats', 'get format' and 'set format'. The naming of the majority of the other ioctl numbers follows the same type of reasoning.

In the above example the argument passed is known to be of the type `struct v4l2_pix_format` so the subfunction above (which attempts to get the format), declares a local pointer to this type and passes the argument to the function as the pointer. If the argument's type is not correct the -EINVAL is returned to the calling application, which will generally use this as an indication that this type does not work, and so tries another type. It then copies the type of format that the currently loaded card supports into the userspace structure, to be examined by the userspace application which will react appropriately.

5.4.2.4 Negotiating an input/output method

The video inputs and outputs generally refer to the physical connections of a device. In the case of the Meteor-II/MC, the inputs are the different channels through which the analog signal is arrives, depending on how the cable has been wired. Thus the input method for the meteor2 driver is dependent on the input channels that are specified in the DCF, and constrained by the choices made when cabling.

5.4.2.5 The actual input/output loop

As the output for the Meteor-II/MC in this case is to the computer screen and not to a device connected directly to the framegrabber, the output loop is unimportant to this project. However, the input loop describes how the framegrabber should handle requests for images. Again these are called via the ioctl function. The application may just request a single grab snapshot which it may convert to a file format if it wishes, or a streaming grab which can also be recorded into any moving image file format (avi, etc.).

The streaming grab requires the driver to implement the image grabbing and consequent copying to onboard buffers. At the same time, the driver must manage a queue of images to be copied from onboard buffers to the PC memory buffers. Management is the appropriate word here. If the PC has not finished the combined operation of reading its memory buffers and giving the result to the application, the framegrabber must wait until this operation is finished before copying the contents of its onboard buffers across. Each driver must implement this control appropriately, as each device is different and will perform the above operations across different bus types and at different speeds.

5.5 Summary

The three viable design options available were presented in this chapter, in addition I took a closer look at the module and v4l2 APIs from a programming point of view. This has hopefully complemented the earlier discussion of the conceptual design of both APIs, and will serve to make the description of the implementation easier to understand. The next chapter discusses the actual implementation, and will refer back to various different topics that have been treated thus far.

Chapter 6

Implementation

6.1 Introduction

I will now provide a description of the final design of the driver in terms of what the actual code does, and the order in which certain portions of code are executed. Rather than explaining the entire working of the driver, I will attempt to explain those parts that are most important, whilst emphasising the changes I found were needed.

6.2 Initial Attempts

My first attempt was a rewrite of the driver from scratch. I used two drivers that were compliant with the new v4l2 API (bttv and saa7134[37]) as an initial reference and added one line of code from the original driver at a time. I used the v4l2 skeleton driver as a basis[38]. This approach meant I could analyse exactly what every line of code actually did, while at the same time incorporating changes that would make the new code compatible with newer compilers[39].

To simplify matters whilst writing this driver, I also disregarded any code contained in the original driver which was not relevant to the Meteor-II/MC. The driver has a lot of code to support other variants of the Meteor-II/MC and also other boards from the Meteor-II family of boards. The exclusion of this code also allowed me to write the module using the PCI API provided by the kernel[40]. I was able to use the module initialisation functions designed specifically for PCI devices, which search for and initialise the PCI device transparently. Again the notion of stacked modules was evident here, with this module using exported symbols from the v4l2, videobuffer, and PCI modules.

Whilst this approach proved to be a valuable learning experience, it was nonetheless based on false assumptions. It transpired that the older Meteor-II/MC specifications sheet had incorrectly posited that the board supported the scatter-gather capabilities. As a consequence it meant that I would not be able to use the videobuffer API or any of the code

37 WEB-xawtv. Both have been included on the disk for reference.
38 Also contained on the disk.
39 As a result of using the latest development tools, the compiler in particular had become stricter on the syntax of the c language and I was thus able to correct certain constructs that future compilers will view as deprecated.
40 Some of the other boards do not use the PCI bus, so this was not an option with the original driver.

provided by that library, as it was written specifically for boards that supported scatter-gather. This means that although this version of driver loads, unloads, initialises the board and correctly registers with the appropriate modules, the actual transfer of the image from the board's memory to the computer's memory does not take place. For reference, I have also included this version of my driver on the disk.

6.3 Final Implementation

Using the knowledge acquired when writing the first driver, I then attempted to re-incorporate changes I had made back into the original version of the driver. The following sections document the various different types of changes that were needed as well as a walk-through of the code.

6.3.1 Cosmetic changes

Many changes proved to be cosmetic, in that the actual code contained in the function did not change hugely, but rather the entry points for the function had to be changed. For example, one of the differences between the old-v4l2 and new-v4l2 APIs was that instead of the `file_operations` containing pointers to the various file operations, it contained a pointer to a `file_operations` structure contained in the actual driver. As an example of these types of changes consider the following portions of code:

```
static int
via_v4l2_open(struct v4l2_device *vdev, int flags, void **idptr)
{
        int retval;
        struct via_card *card = (struct via_card *) vdev;
        struct via_open_list *new;
        struct list_head *head = &card->open_list;

        assert (vdev != NULL);

        MOD_INC_USE_COUNT;

        /* support only multiple no-I/O opens */
        if (card->io_open && !((flags & O_NOIO) == O_NOIO)) {
                via_debug(VIA_DBG_OPEN,
                          "EBUSY: empty: %d  flags: %d\n",
                          list_empty(head), flags);
                retval = -EBUSY;
                goto err_out;
        }

        /* allocate memory for open_list structure */
        new = (struct via_open_list *) kmalloc (sizeof (*new), GFP_KERNEL);
        if (new == NULL) {
                via_error("kmalloc failed\n");
                retval = -ENOMEM;
                goto err_out;
        }
        new->card = card;
        new->flags = flags;
        list_add_tail (&new->open_list, head);
        card->io_open = (flags & O_NOIO) != O_NOIO;

        *idptr = (void *)new;

        via_debug(VIA_DBG_OPEN,
                  "new id: %8X  flags: %d %d\n", (int)new, flags, O_NOIO);
        return 0;
 err_out:

        MOD_DEC_USE_COUNT;
        return retval;
}
```

This is the original code from the driver, which used the older v4l2 API. The corresponding newer code is:

```
static struct file_operations via_fops =
{
        owner:                  THIS_MODULE,
        open:                   via_v4l2_open,
        release:                via_v4l2_release,
        read:                   via_v4l2_read,
        mmap:                   via_v4l2_mmap,
        poll:                   via_v4l2_poll,
        ioctl:                  via_v4l2_ioctl,
        llseek:                 no_llseek,
};

via_v4l2_open(struct inode *inode, struct file *file)
{
        int minor = minor(inode->i_rdev);
        struct via_card *card = NULL;
        struct list_head *head = NULL;
        struct file_data *fh;
        enum v4l2_buf_type type = 0;
        unsigned int i;
        int flags=file->f_flags;
        MOD_INC_USE_COUNT;

        for (i = 0; i < via_card_cnt; i++) {
                if (via_cardlist[i].vdev.minor == minor) {
                        card = &via_cardlist[i];
                        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        break;
                }
        }
        head=&card->open_list;
        /* allocate per filehandle data */
        fh = kmalloc(sizeof(*fh),GFP_KERNEL);
        if (NULL == fh)
```

```
                return -ENOMEM;
        file->private_data = fh;

        memset(fh,0,sizeof(*fh));
        file->private_data = fh;
        fh->card      = card;
        fh->type     = type;
        fh->fmt      = format_by_fourcc(VIA_DEFAULT_PIXFMT);
        fh->width    = VIA_DEFAULT_WIDTH;
        fh->height   = VIA_DEFAULT_HEIGHT;

        /* support only multiple no-I/O opens */
        if (card->io_open && !((flags & O_NOIO) == O_NOIO)) {
                via_debug(VIA_DBG_OPEN, "EBUSY: empty: %d  flags: %d\n",
                                list_empty head), flags);
                MOD_DEC_USE_COUNT;
                return -EBUSY;
        }

        card->io_open = (flags & O_NOIO) != O_NOIO;
        card->users++;
        via_info("open has succeeded. users:%d\n",card->users);
        return 0;
}
```

The newer version retains much of the functionality of the older version, but uses a slightly different style. The entry points to the function have changed and therefore some of the data structures have too. Also, the method of keeping track of the number of times the device has been opened (by different applications, for example) has changed from a linked list to a variable contained within the data structure that describes the board.

Other changes that affected the compilation of the driver included various name changes. For example, the older v4l2 API typically used functions call that started with `v4l2_`, whereas the newer v4l2 reverted back to the v4l1-style `video_` prefix, which allows the two APIs to use the same header file and provides some transparency to those converting v4l1 to v4l2 drivers. Correcting the errors caused by such changes required finding a compilation error, finding the original function call in the old-v4l2 module files, and comparing the old style with the new style. However, in addition to having the function call name modified, many of the functions also contained features that were either considered deprecated or were reimplemented in a more appropriate function. This meant finding the corresponding feature elsewhere in the new-v4l2 module files and attempting to reimplement those features in the meteor2 module.

As an example of the changes that were required, consider the old-v4l2 queue management system. This was a suite of helper functions that controlled video buffer queue management, and essentially managed how the data in onboard buffers was copied to PC memory buffers. This is one of the features that was considered deprecated in the new-v4l2 API, and so it was completely removed. Unfortunately, the original meteor2 driver relied heavily on these functions. Most of the functionality had been transferred to the videobuffer API, which, as was stated before, was not applicable to the Meteor-II/MC due to the lack of scatter-gather capabilities. Therefore the driver would have to provide its own queue management routines. Fortunately, before I had completely rewritten my own functions, I realised that I could possibly reuse the functions that were provided by the old-v4l2 API. I copied the appropriate functions from the old-v4l2 source file into the driver code and with some minor modifications they worked[41].

Other changes proved somewhat more troublesome. Changes to data structures, in particular those that kept the same name but changed the variables contained within the structure, caused some problems and introduced some elusive bugs. As an example, there is a feature that allowed the timestamping of each buffer. This was used to calculate information such as frames per second or as a performance gauge (the number of frames that had to be dropped due to the onboard buffers filling up before the current buffer had been copied to the PC memory). Timestamps originally had a data type of `stamp_t`, a signed 64-bit integer. However, in the new-v4l2 API the data type had changed to `timeval`, a time value data structure provided by the kernel. Whilst this change obviously made sense, a portion of the original driver code performed integer operations on the original timestamp, sometimes using a function specifically provided for by the old-v4l2. The `v4l2_math_div_6432`[42] function, now also deprecated, was an example of a function that was used. Again, I tried copying the required functions from the old-v4l2 API into the driver code, but due the changed data types, incompatibilities still existed. As a workaround I implemented a function to convert a `stamp_t` data type to `timeval`, and vice-versa, which, although it works, should be regarded as a 'hack' that requires extra computation, and as such would quite certainly affect performance.

Implementing cosmetic changes as documented above meant that the driver would compile correctly and (sometimes) load as a module, but it also introduced many bugs and incompatibilities into other parts of the code. Tracking down these bugs was very time-consuming, as some of them were not evident at all. However, to date I have found a great many of these bugs, and have either rewritten the appropriate portions of code or implemented workarounds.

As was stated earlier, I also incorporated certain aspects of my driver that I thought would improve the performance of general state of the driver. I will now give details of the operation of the driver, that may serve as a guideline for those who wish to understand the code, whilst building on all the topics discussed so far.

41 Such code is in the file via-oldv4l2.c
42 Divides a 64-bit number by a 32-bit number, returns the quotient and also the remainder.

6.3.2 Data Structures

There are a few important structures in the file which I will now explain. The first is basically a structure that describes the features of the card and is used by the other functions throughout the driver to refer to the card. `struct via_card` contains, amongst other things, a `struct video_device` that contains the corresponding v4l2 device information, a `struct pci_dev` that contains the PCI device information for the board, variables that track the various different states of the board and driver (e.g if the board is idle, doing a snapshot grab or doing a streaming grab), the minor number, the virtual addresses of the boards various registers, the IRQ used by the board and its state, information about the the type of input signal, and the format the image will be sent in[43]. Below is a brief listing of the data structure, the full definition can be found in the file `via.h`[44]. This structure is implemented as a linked list. In the case of a system that has more than one board supported by the same driver, `struct via_card` contains a pointer `next` which contains the address of the card data structure for the next board.

```
struct via_card {

        struct video_device vdev;     /* must come first! */
        struct via_card *next;
        struct pci_dev *pdev;         /* pci bus info for this card */
        int card_num;
        int users;
        struct via_card_info card_info;
        struct via_regs regs;
        int irq;                      /* IRQ (from pdev->irq) */
        int dcf_loaded;
....

};
```

One of the additions that was needed for the old driver was a structure that contained information about the nature of the file structure being passed to the various open, ioctl, read, etc., functions. The generic kernel `file` data structure also contains a field called `private_data`, where a pointer to the appropriate information can be stored. The version I implemented is as follows:

43 It may be useful to note here that although the `via_card` data structure is a representation of the actual board, changes made to it do not propagate to the board directly. The driver must write certain bitmasks to the various registers on the board in order to make it do anything at all. It may then record the fact that this has been done in the data structure representing the device. This is one of the difficulties of writing a driver without the board programmers specifications, as one does not know what register on what chip must contain what value in order to make the board perform a specific task.

44 The full version is not included here due to its length.

```
struct file_data {
        struct via_card   *card;
        unsigned int resources;
        enum v4l2_buf_type       type;
        const struct via_format *fmt;
        unsigned int             width;
        unsigned int             height;
        int                      lines;
        struct v4l2_window         win;
};
```

It contains some very basic information about the file, but most importantly contains a pointer to the `via_card` data structure associated with the file. This is different from the older driver, which explicitly passed a pointer to the `video_device` data structure that could be used to obtain the associated board information.

Two other two data structures I introduced were `via_format` and `via_tvnorm`, used to store information about the analog signal and image format, and both of which reference an array of valid formats and analog signals[45].

Again, introducing new structures caused many existing features to break. The most effective means of finding these bugs was to load the module, examine the kernel output, monitor the appropriate files in proc filesystem, and run my test program xawtv with maximum verbosity, and again examine the output of the application and cross-reference it with the kernel output.

6.3.3 Initialisation Functions

6.3.3.1 via_module_init

The module is loaded by the function named by the `module_init` declaration in `via.c`, `via_module_init`. This function makes a directory under the proc filesystem for the driver (`/proc/driver/meteor2`), uses the PCI kernel API to find the specific devices, calls the function to get high memory access, and outputs information relating to the above to the kernel log.

6.3.3.2 via_register_card

This function is called by `via_module_init` and requests memory for the `via_card` structure and copies default values into the `video_device` structure. It also assigns minor numbers (to be checked by registration with the videodev module) and saves a pointer to the actual PCI

45 This concept was used in the bttv and saa7134 drivers and the appropriate arrays were simply copied from those drivers source files. The arrays contain the relevant information (e.g. pixels per line, lines per scan, etc) The arrays are in the file via-data.h and the structure definitions are in the via.h file.

device where the board was found. This function also registers the card with the videodev module and creates an entry in the meteor2 proc directory for each card found[46]. Another slight change was introduced to the v4l2 registration process. Previously, a driver function named `via_v4l2_initialize` was automatically called during registration. With the change, `via_v4l2_initialize` now needs to be called explicitly from this function.

6.3.3.3 via_v4l2_initialize

This function sets the default values for the numerous variables of the `via_card` data structure, and then adds the structure to the driver's internal array of cards found. It creates a board-specific proc entry in the directory created for the driver. In this way, every card has a unique entry in the directory /proc/drivers/meteor2/. There is also a call to `via_setup_ccir` which claims to set up the board for a ccir camera as default. After my changes, this did not work fully[47] and a DCF must be loaded manually after the loading of the driver. A call to `via_setup_card` initialises those variables that are different to each of the different Meteor-II boards (e.g. number of input channels, etc.). There are also calls to initialise and enable the IRQ for the device, and a call to `via_init_all`.

6.3.3.4 via_init_irq

This function requests the use of an IRQ from the kernel and tells the kernel the local function that will manage IRQs. This is achieved by the following line where, `card->pdev->irq` is the IRQ requested, `via_irq_handler` is the local function, `VIA_MODULE_NAME` is 'meteor2'[48], and `card` provides the kernel with a unique device identifier in the case of an interrupt shared with another device. The bitmasks `SA_INTERRUPT` and `SA_SHIRQ` tell the kernel that the handler is what is known as a fast handler and that a shared IRQ is acceptable.

```
    request_irq(card->pdev->irq, via_irq_handler,
                SA_INTERRUPT | SA_SHIRQ, VIA_MODULE_NAME, card);
```

6.3.3.5 via_init_all

This function performs a huge number of initialisation tasks, many of which are in the `via-init.c`, `via-lut.c`, `via-fpga.c` files. From the filenames, it is possible to guess that these contain the various functions to initialise the LUTs, load the appropriate microcode into the FPGA chip, load the appropriate values into the all onboard registers and set the corresponding variables on the `via_card` structure. Other operations include obtaining PCI memory addresses and initialising any onboard modules that may be present. I had no need (nor the required technical information) to modify any of these files or functions.

46 For example, type `cat /proc/driver/meteor2/0` for information on the first card.
47 I am unsure if it worked before.
48 This is what will appear in the `/proc/interrupts` file to show the owner of the IRQ. Type `cat /proc/interrupts` to review the output.

6.3.3.6 via_get_himem

This function calculates the total high memory and makes it available to the cards. This function is in `via-mem.c` and no changes were made to this file for this version of the driver.

6.3.4 Cleanup Functions

6.3.4.1 via_module_exit

This function tries to undo all the registration with the various kernel APIs, de-allocates memory, and removes the appropriate proc directory and its entries. It does so by calling the following functions.

6.3.4.2 via_unregister_all_cards

This function loops through the drivers array of `via_card` structures and calls `via_unregister_card` for each one.

6.3.4.3 via_unregister_card

This function disables the IRQ, stops the grab if there is one in progress, removes the entries in the /proc/driver/meteor2 directory, unregisters the device with the videodev module, frees the IRQ, unmaps the PCI memory and frees the memory allocated for the `via_card` structure.

6.3.5 file_operations functions

The functionality provided by functions does not differ greatly from the descriptions already given. **via_v4l2_open** is called by an application to open the device for other operations, similarly **via_v4l2_release** is called to release the device. **via_v4l2_read** sets up the memory buffers on the board and in PC memory, takes a snapshot, copies from onboard buffers to PC memory and from there copies it to a userspace structure that the application can access. **via_v4l2_mmap** requests that the framegrabbers memory be mapped to an address space accessible by the driver, and **via_v4l2_poll** polls the device to see if the buffer has been copied to PC memory by the read function, and notifies the driver when it has. The more interesting functions are those that interact with userspace applications and allow these applications access to certain parts of kernelspace mechanisms. These are the functions that handle the ioctl functions.

6.3.5.1 via_v4l2_ioctl

This function is just a wrapper that passes the name of the function that actually handles the ioctl calls to a helper function. The helper function is in the videodev module and provides management for userspace ioctl calls.

6.3.5.2 via_v4l2_do_ioctl

This is the function that really does the work. If the identifier is a private ioctl (i.e. not one of the standard v4l2 ioctl calls,) control is passed to another function that handles those. The function then checks that a DCF has been loaded (checks the `dcf_loaded` variable in the `via_card` structure) and if not, quits and dumps an error in the kernel log to tell the user to load a DCF. Loading a DCF is a private ioctl, so the user can load a DCF before reaching this error. Next, in the switch statement, any ioctls related to streaming are passed to a function that will handle them separately. Finally all the standard ioctls are dealt with. I will not explain each one separately, as it may be more useful to summarise the ioctl calls that appear in the output of the xawtv application:

ioctl: VIDIOC_ENUMINPUT: Enumerate supported input types. (input channels)
ioctl: VIDIOC_ENUMSTD: Enumerate supported video standards. (PAL, NTSC, SECAM)
ioctl: VIDIOC_ENUM_FMT: Enumerate supported video formats (GREY, RGB888, etc)
ioctl: VIDIOC_QUERYCTRL: Ask the driver what controls it provides
ioctl: VIDIOC_G_INPUT: Ask the driver what current input channel is set
ioctl: VIDIOC_G_CTRL: Ask the driver the current values of the controls it supports
ioctl: VIDIOC_S_FMT: Request driver to set a format.
ioctl: VIDIOC_S_INPUT: Set the input channel
ioctl: VIDIOC_S_STD: Set the video standard

It should be noted that if any of the ioctl calls return -EINVAL, this does not usually signify an error, but rather that a certain format, standard, or control is not supported, and the application only keeps track of those ioctl calls that return zero[49]. Also, the controls referred to above are the framegrabber subsampling operations that the user can control interactively, e.g. gain control. I had to change the code for most of the ioctl calls, and as before, incorporated code from my first driver. Many of the ioctl calls had been written but were untested with standard v4l2 client applications and worked solely with the smaller custom written client applications that had been written alongside. This meant that the errors in this code were found and the code was fixed to follow the v4l2 standard correctly. The following two functions were minimally modified and are thus included for completion.

49 In the xawtv log file, these are the ioctl calls ending with `ok`

6.3.5.3 via_priv_ioctl

This function handles the private ioctl calls. As above I will summarise the various different calls, although most are intuitively named.

VIA_G_CARDINFO: returns a data structure with information about the card
VIA_S_LUT: setup the LUTs
VIA_LUT_RESET: reset the LUTs
VIA_START_SNAPSHOT: sets up the memory and registers, and starts a snapshot grab
VIA_S_DCF: setup DCF data

The only one of the above ioctl calls that needed changing was the S_DCF, which for unknown reasons caused repeated crashes with a piece of code that seemed to work correctly in the original driver. Neither the original author nor I were able to find any problems with the original code, but on closer inspection is seems that the standard kernel call `copy_from_user()` was failing. After some research, I have come to the conclusion that this was failing as a result of what is called 'fixup' code in the kernel itself. This code catches many memory errors that occur due to the design of the x86 family of processors. However, a few cases remain and I believe this code to have exposed one. As a workaround, I rewrote the code using another method.

6.3.5.4 via_stream_ioctl

This is the function that handles the streaming ioctls function calls and resides in the `via-stream.c` file. As was stated before, most of the code for streaming was left intact with mainly cosmetic changes, and the required queue management helper functions were copied from the old v4l2 source files. Again the calls are named intuitively.

VIDIOC_STREAMON: Start streaming capture
VIDIOC_STREAMOFF: Stop streaming capture
VIDIOC_REQBUFS: Request memory buffers
VIDIOC_QUERYBUF: Query the state of the buffer (has it finished being copied?)
VIDIOC_QBUF: Queue the buffer
VIDIOC_DQBUF: Dequeue the buffer

The detailed information that can be exchanged from userspace to kernelspace via the ioctl functions can be seen upon inspection of the output log for xawtv[50].

There are of course many other functions in the driver, and the driver has been written following the kernel coding guidelines included with the kernel sources which states that

50 A selection of the output can be seen in Appendix E

“Functions should be short and sweet, and do just one thing. [...] The maximum length of a function is inversely proportional to the complexity and indentation level of that function.” This tries to ensure readability of code as well as simplicity of code. Therefore, describing every function in detail is impractical. The most important functions, along with some that were modified, have been presented here in an effort to show the internal working of the driver. There were also some userspace programs with the original driver sources that allowed me to learn how to write a client application that accessed the kernelspace driver module from userspace.

6.4 Utilities

There were some utilities included with original driver package that were designed to be used as a basis for writing custom userspace programs that interacted with the driver. As with the driver, I also needed to convert these programs to the new-v4l2 format. The utilities include `dcf`, which is a program that loads the values in a DCF to the board, via the ioctl call. When I initially tried to capture an image, I forgot to load a DCF, and in the read function, the driver went to sleep (entered a 'sleep' loop) waiting for an interrupt from the hardware to wake it up. After trying unsuccessfully to track down the bug (including a complete rewrite of the interrupt handling code) I found the true cause of the problem to be that the second part the camera interface, the DCF, had not been loaded, so the board was missing information that it needed to successfully perform the operations. In addition to fixing the dcf utility, I incorporated another change into the driver that disallowed the calling of the ioctl function unless a DCF had been loaded. I also converted the other single grab utilities ppmgrab and pgmgrab to the new-v4l2 format.

6.5 Summary

This chapter has provided a more in-depth look at the actual code of the driver. I have also discussed some of the changes that I incorporated and have hopefully shown the usefulness of understanding the subjects treated by preceding chapters. From the initial implementation attempts to the final implementation proved to be a difficult but informative experience, and the next chapter discusses the results achieved by the final implementation. I also look briefly at other possible solutions for for using Meteor-II family of framegrabbers under linux, and conclude this report.

Chapter 7

Conclusion

7.1 Results

The device driver that is included on the disk should compile and load without problem on any system that meets the configuration requirements. Detailed instructions on installing the driver are included in Appendix A. This device driver module now uses the new-v4l2 API instead of the now-deprecated old-v4l2 API, and is functional. The analog signal sent by the camera to the framegrabber undergoes the necessary subsampling and is outputted in an image format. It is then copied to the onboard buffers, and the driver manages the copying of the image from onboard buffers to the PC memory. The driver then tells the kernel to copy it to userspace where the application can then perform its operations, whether that be displaying, recording or some other operation. Both snapshot and streaming grabs are functional. The following is a screenshot of the streaming capture window of xawtv.

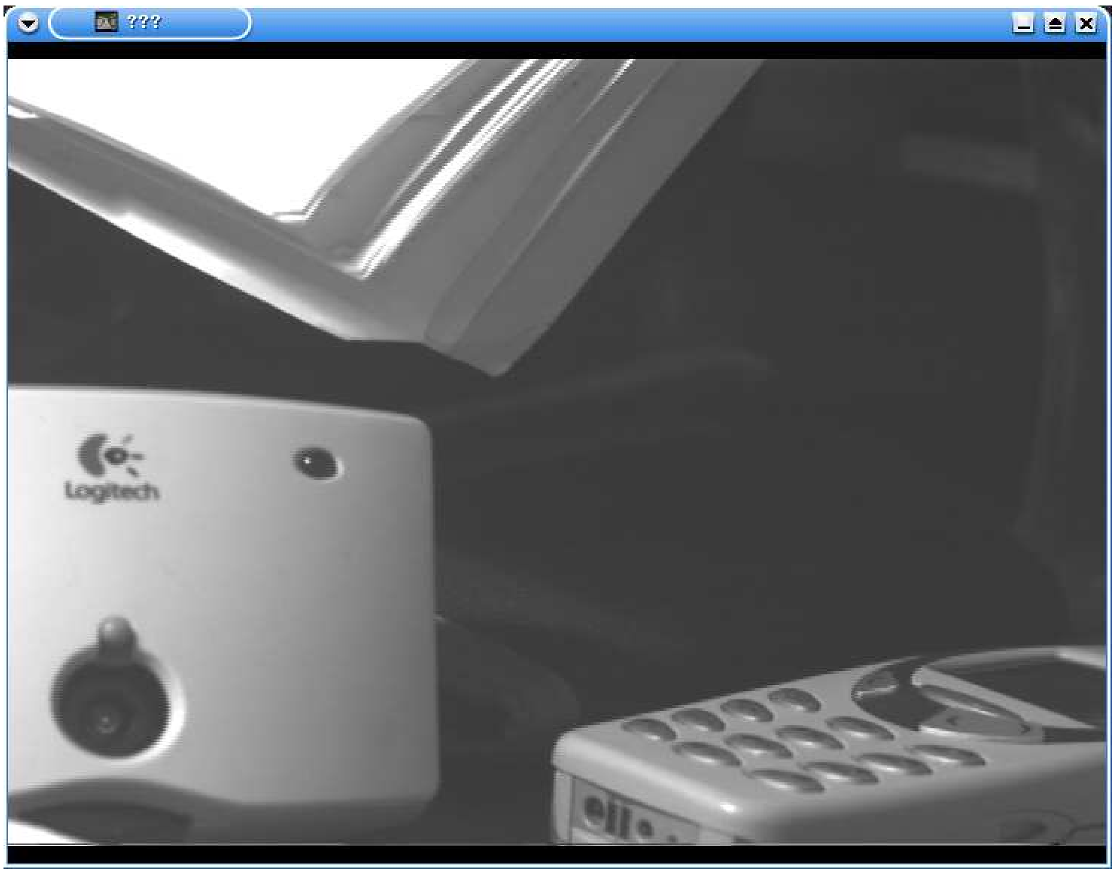


Figure 7.1 xawtv screenshot using the final implementation of the meteor2 driver

Appendix D contains other screenshots of xawtv using the new-v4l2 compliant device driver for the Meteor-II/MC. As of this writing, the new v4l2 API has been accepted into the current development kernel (2.5) and will be a part of the upcoming stable 2.6 kernel. I have submitted my changes to the driver to the original author as a patch and he has said he will incorporate my changes into the next release of the driver. This should facilitate the use of the driver with recent stable kernels and well into the future with the new stable release.

The newer kernel will also provide a better means for those who wish to write multimedia device drivers with the inclusion of the v4l2 API. The addition of the videobuffer API is another welcome improvement to the kernel interface, and will hopefully provide another easier path for those who aspire to write device drivers for such devices. With respect to the Meteor-II/MC I also believe future development is possible and should continue.

7.2 Future Development

There are of course many improvements and cleanups that could be made to the device driver at the moment. Unfortunately, some access is required to technical documents that Matrox may or may not be willing to share, and I do not currently have them. However, in its current state, there exist a number of different avenues along which development for this particular framegrabber device may continue. I will briefly outline what I believe to be some alternative means of obtaining more functionality from this framegrabber under linux.

7.2.1 MIL

Contained in the code for the driver is the initial support for other boards of the Meteor-II family, and if this driver continues to be developed the driver will support all members of this family. The Matrox Imaging Library provides a high-level interface for those who wish to program the Matrox boards.

One very viable option as this driver develops (and even in its current state) would be the development of an imaging library interface for linux. This could take the form of a port of the MIL library that Matrox provides for Windows users, or could be a custom-made library. This would require an understanding of the interface with the driver (as described in this report) and would essentially provide a library that would hide the details of the driver from the user. Ultimately, this would provide greater enduser ease of use for this driver, and permit more people to benefit from its existence.

7.2.2 WINE

WINE stands for Wine Is Not an Emulator. Nevertheless, WINE is a type of emulator that permits programs designed to run under Windows, to run under linux. It provides a compatibility layer for the Windows API and emulates some of the system calls, whilst invoking its own calls for other ones. The reason I have included it in this section is because I investigated the possibility of running Matrox's suite of programs using WINE. The screenshot S-4 in Appendix D shows Matrox Intellicam running under linux[51]. This proves enormously useful in itself, as it provides a way of designing DCFs without rebooting to Windows or hand-editing.

However, the Meteor-II Windows device driver fails to load, as can be seen from the program's output in terminal window in the background. Most of the functionality (save capture) of the program is still there. However, I believe it would be possible to write the appropriate virtual device driver for the WINE project. This would involve inspecting the output of the Windows driver in a Windows debugger (reverse engineering) coupled with an investigation of the calls required to allow a device driver to interact with the Windows multimedia API, and then redirecting the appropriate system calls to the v4l2 device driver. This would permit the Intellicam program to become somewhat more portable[52], and in doing so improve the capabilities of this framegrabber under linux.

If the previous suggestion were implemented, one could also reap the benefits of the sample programs that Matrox supplies, as Microsoft Visual Studio is also reported to work correctly using WINE.

I believe that if both the preceding projects were undertaken, coupled with the continued development of this driver, it would forward progress for using another family of hardware devices under linux. This project has taught me that an operating systems is basically just an interface to available hardware, and therefore, I must conclude that making more hardware accessible to users in any operating system can only be a good thing.

7.3 Valuable Research

I would also like to note the various other subjects that I was exposed to as a result of doing this project. The subjects presented in this report are obvious candidates and include the basics of machine vision, operating system theory, linux kernel development and

51 WINE version 20030408 was used for this shot, some earlier versions did not work fully.
52 Users still require a copy of Windows on a local disk or network file system for this to work.

programming in c. Apart from these, there were many other subjects that I learnt as a direct result of doing this project. A non-exhaustive list would include setting up a CVS repository and using it for development, linux userspace programming, learning how to solder, and also the concept of *netiquette* and how one should proceed when contributing to an opensource project. I have also learned how to navigate through the linux kernel source, find and modify exactly what I need, a feat that at one time would have been a daunting prospect for me. This project also introduced me to a wide range of software that I may not have otherwise encountered, and helped me prove to myself that kernel development *can* be done by normal people.

7.4 Conclusion

The various fields of interest entailed in this project were far greater than I had imagined in the beginning. However, I have no regrets in having chosen it, as the wide range of subjects it permitted (and sometimes forced) me to study is a reward in itself. I hope I have provided a succinct account of the work that went into this project, and have successfully imparted the knowledge that I have gleaned from it. I will continue to help with the development efforts of this driver where possible and I look forward to starting development of a device driver for a certain webcam that is not (yet) functional under linux.

Bibliography

The lack of references to printed books in this report is due mainly to the fact that most information has been taken from the linux kernel source code. Comments in the code and the Documentation directory provide provide perfectly concise explanations of necessary concepts. This view is reiterated by the authors of [RUS01, pp527] - “Most of the information in this book has been extracted from the kernel sources which are the best documentation of the linux kernel”. Also, many books that cover the subject of linux become outdated very quickly due to the volatile nature of kernel development.

References

[RUS01] *Alessandro Rubini & Jonathan Corbet*
Linux Device Drivers, 2nd Edition, O' Reilly (2001)

[VER91] *David Vernon*
Machine Vision, Prentice Hall (1991)

[STA01] *William Stallings*
Operating Systems Internals, 4th Edition, Prentice Hall (2001)

Kernel Source Documentation

The following are those documents contained in the Documentation directory of the linux kernel sources that proved very useful throughout the course of this project

Documentation/BUG-HUNTING
Documentation/CodingStyle
Documentation/DMA-mapping.txt
Documentation/devices.txt
Documentation/pci.txt
Documentation/spinlocks.txt
Documentation/video4linux/API.html

Internet References

[WEB-Meteor2]

The original driver source – http://www.emlix.com/en/opensource/meteor2

Cord Seele's Meteor2 Driver Page. Cord Seele is the author of the original driver.

[WEB-LFS]

LFS website – http://www.linuxfromscratch.org

The linuxfromscratch website.

[WEB-LKW]

Linux Kernel Website – http://www.kernel.org

Main website for the source code of the linux kernel

[WEB-V4L2]

Video4Linux2 Website – http://www.thedirks.org/v4l2

The original v4l2 website. Contains the original v4l2 specifications, and the Driver Writers Guide.

[WEB-NV4L2]

'New' Video4Linux2 Website – http://www.bytesex.org/v4l

Contains the specifications of the new v4l2 API

[WEB-xawtv]

The xawtv homepage – http://www.bytesex.org/xawtv

xawtv is a suite of video capture applications that supports v4l, old-v4l2 and new-v4l2 drivers

[WEB-Matrox]

Matrox Meteor-II/MC website – http://matrox.com/imaging/products/meteor2_mc/home.cfm

Product description and literature for the Matrox Meteor-II/MC Multi-Channel

[WEB-WINE]

WINE HeadQuarters – http://www.winehq.org

Principal website containing information about WINE, and the appropriate downloads

[WEB_FOURCC]

The [Almost Definitive] FOURCC Definition List – http://www.fourcc.org

Contains descriptions and formal definitions of the various fourcc formats

[WEB-UML]

User-mode Linux – http://www.usermodelinux.org

Documentation and downloads relating to usermode linux

[WEB-Hitachi]

Hitachi KP-M1 – http://www.hitachi-denshi-uk.com/industrial_video/monochrome_cameras/pages/kpm1.htm

Product description and literature for the Hitachi KP-M1 Camera

Appendix A

Installation and Usage

NB Make sure that you are running kernel 2.4.20 or later. The v4l2 module on the disk is only for kernel 2.4.20 but a patch for a later kernel will probably be available from [WEB-NV4L2] if it has not already been incorporated into the kernel proper.

1. Untar the v4l2.new.tar.gz file using the command **`tar xfz v4l2.new.tar.gz`**
2. Enter the directory and type **`make`**
3. **`su`** to root and type **`make install`**, this will install the module in the /lib/modules/2.4.X/kernel/ hierarchy.
4. Enter the utils subdirectory and type **`make`**
5. Untar the file meteor2-1.1.9.tar.gz using the command **`tar xfz meteor2-1.1.9.tar.gz`**
6. Enter the directory and type **`make`**
7. **`su`** to root and type **`make install`**, this will install the module in the /lib/modules/2.4.X/kernel/ hierarchy
8. Both modules are now installed, however the kernel itself may require rebuilding as the video4linux[53] module must be built if it is not already. Rebuilding the entire kernel is beyond the scope of this installation guide.
9. Add the line `alias char-major-81 meteor2` to the file /etc/modules.conf and type **`depmod -ae`**
10. The bootloader must also be modified, I have no experience with grub but if you use lilo the entry append="MEM=xxx" as described in the report must be added to the relevant section of the /etc/lilo.conf file. When you have edited the file, type **`lilo`** to write the new configuration to the boot sector. Here is a sample entry on a system with 128Mbytes memory (i.e. 28Mbytes for meteor2).

```
image=/boot/vmlinuz
        label="linux-fb"
        read-write
        root=/dev/hda3
        append="video=radeon:1024x768-8@60 mem=100M"
```

11. After rebooting, type **`modprobe meteor2`** (as root) to load the module and then use the dcf utility in the utils directory to load a DCF (a sample ccir/rgb DCF is provided)
12. Load a v4l2 client application. I used xawtv, the command used was **`xawtv -c /dev/video0`**
13. Various controls are presented to you on the menu.

53 This is the original v4l module contained in the kernel sources

Appendix B

Contents of Accompanying Disks

DISK 1

/code	**-**	**contains the source code**
/code/meteor2mc-0.4.tar.gz	-	my 'scatter-gather' PCI driver
/code/meteor2-1.1.9.tar.gz	-	my final version of the driver
/code/v4l2.new.tar.gz	-	the new-v4l2 module
/code/orig/meteor2-1.1.tar.gz	-	original version of the driver
/code/patch/meteor2-newv4l2.diff.gz	-	patch file containing all differences between original and new driver

/docs	**-**	**contains various documentation**
/docs/kern.log.gz	-	kernel output when loading driver and running xawtv
/docs/xawtv.log.gz	-	corresponding xawtv output

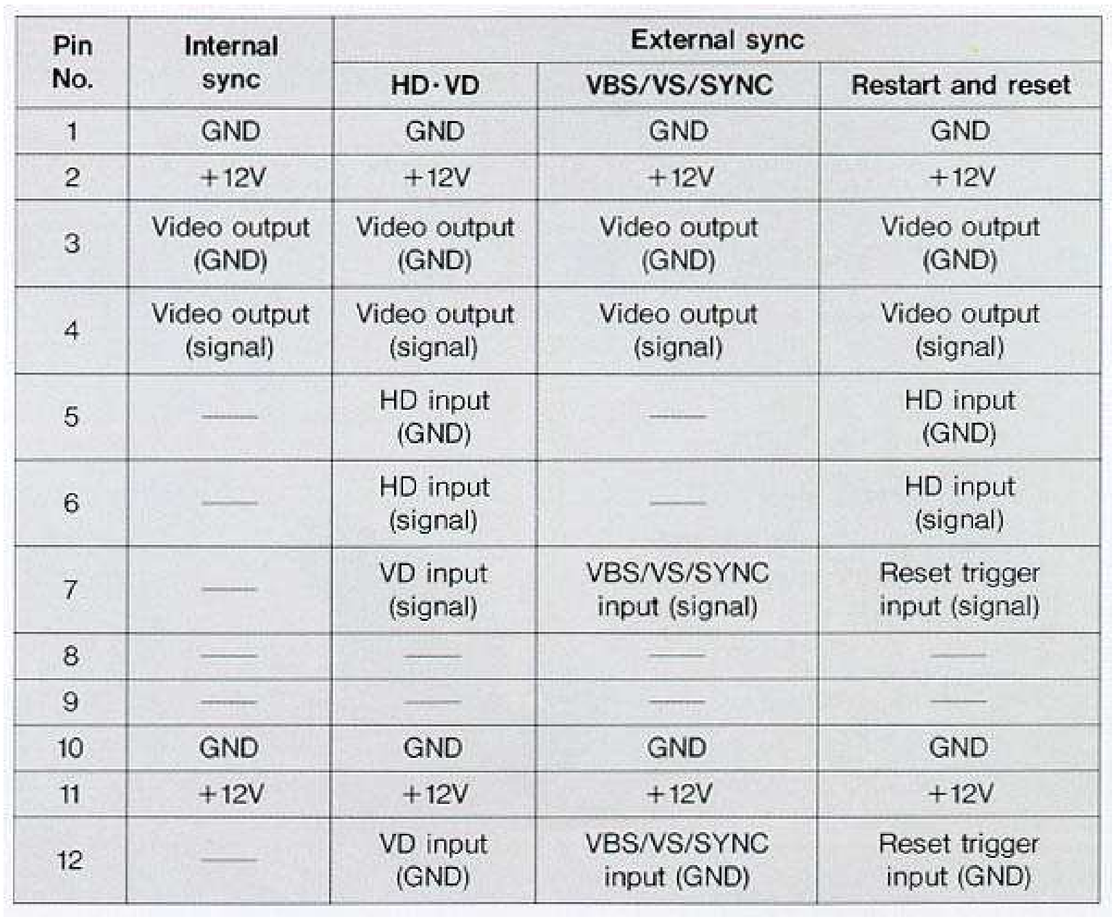
DISK 2

/b_meteor2.pdf	-	original Matrox PDF stating that the board supported scatter-gather
/bttv-0.9.8.tar.gz	-	the bttv driver that was used for reference
/saa7134-0.2.6.tar.gz	-	the saa7134 driver that was used for reference
/skeleton-20020409.tar.gz	-	v4l2 skeleton driver

<u>Appendix C</u>

Pinout connections on Meteor-II/MC and KP-M1 camera

Hitachi KP-M1



Pin No.	Internal sync	External sync		
		HD·VD	VBS/VS/SYNC	Restart and reset
1	GND	GND	GND	GND
2	+12V	+12V	+12V	+12V
3	Video output (GND)	Video output (GND)	Video output (GND)	Video output (GND)
4	Video output (signal)	Video output (signal)	Video output (signal)	Video output (signal)
5	——	HD input (GND)	——	HD input (GND)
6	——	HD input (signal)	——	HD input (signal)
7	——	VD input (signal)	VBS/VS/SYNC input (signal)	Reset trigger input (signal)
8	——	——	——	——
9	——	——	——	——
10	GND	GND	GND	GND
11	+12V	+12V	+12V	+12V
12	——	VD input (GND)	VBS/VS/SYNC input (GND)	Reset trigger input (GND)

From [WEB-Hitachi]

Appendix C

Pinout connections on Meteor-II/MC and KP-M1 camera

Meteor-II/Multi-Channel

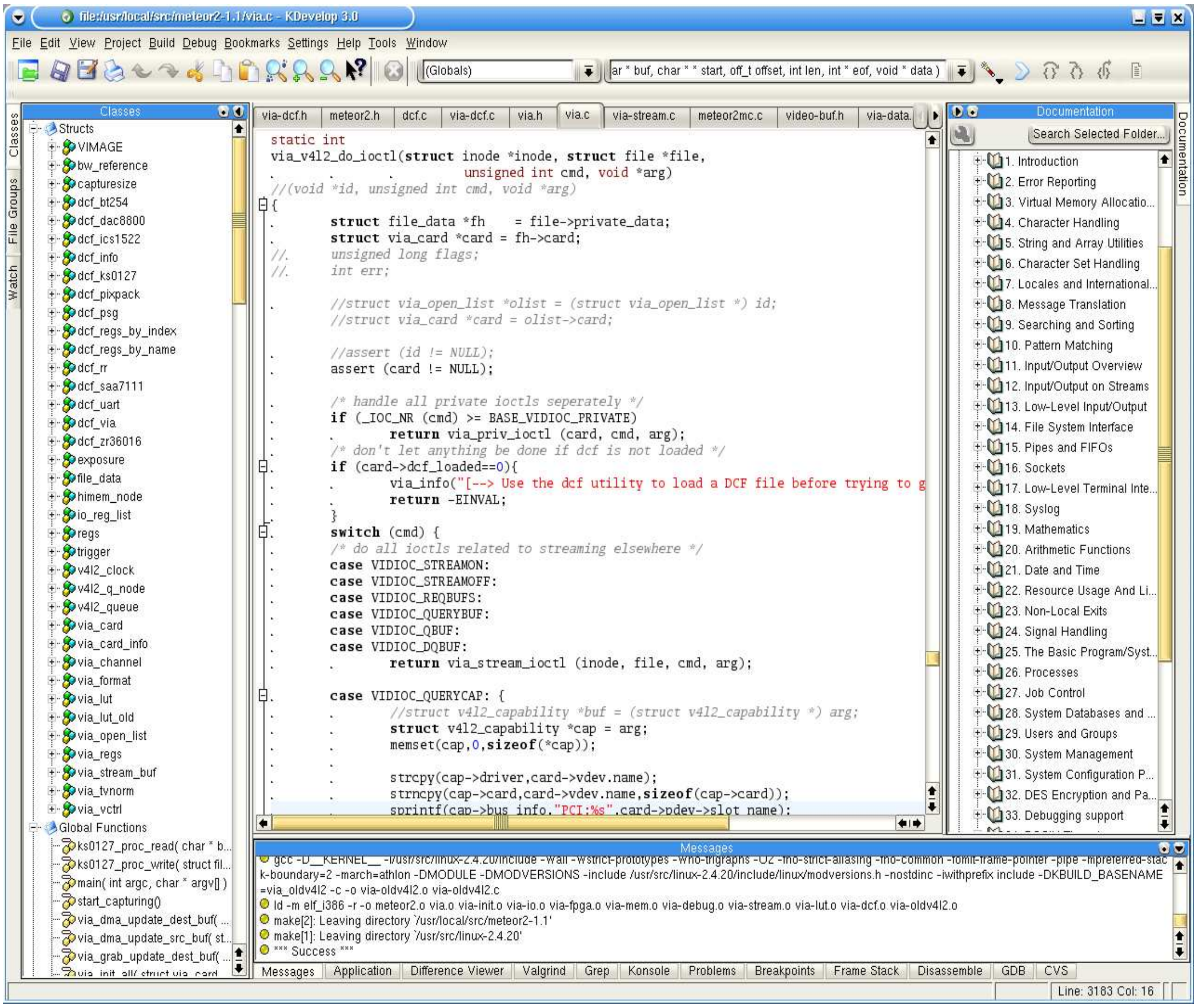
PIN	Signal	Description
15	VID1_IN1	RED Analog Video Input (channel 1)
44	VID1_IN2	GREEN Analog Video Input (channel 1)
13	VID1_IN3	BLUE Analog Video Input (channel 1)
43	SYNC_IN	Analog Video Input (SYNC)
11	VID2_IN1	RED Analog Video Input (channel 2)
41	VID2_IN2	GREEN Analog Video Input (channel 2)
40	VID2_IN3	BLUE Analog Video Input (channel 2)
35	OPTOTRIG+	Opto-Isolated trigger positive input
34	OPTOTRIG-	Opto-Isolated trigger negative input
20	TRIGGER	Non-Protected TTL trigger input
19	CLK_IN_TTL	Clock input (TTL)
33	CLK_OUT_TTL	Clock output (TTL)
32	VSYNC_TTL	Vsync input or output (TTL)
2	HSYNC_TTL	Hsync input or output (TTL)
38	EXP(1)	Exposure #1 output (TTL)
23	EXP(2)	Exposure #2 output (TTL)
36	TX	Transmit (RS-232)
22	RX	Receive (RS-232)
6	CTS	CTS (RS-232)
21	RTS	RTS (RS-232)
39	USER1IN+	Auxiliary User Input #1 (positive)
12	USER1IN-	Auxiliary User Input #1 (negative)
9	USER2IN+	Auxiliary User Input #2 (positive)
10	USER2IN-	Auxiliary User Input #2 (negative)
24	USER1OUT	Auxiliary User Output #1 (TTL)
8	USER2OUT	Auxiliary User Output #2 (TTL)
1, 6	DC POWER	+12V OR +5V Power Supply
7, 37	NC	Not connected on METEOR-II/MC
3-5, 14, 17-18, 25-31, 42	GND	Ground

From [WEB-Matrox]

Appendix D

Screenshots

Screenshot S-1

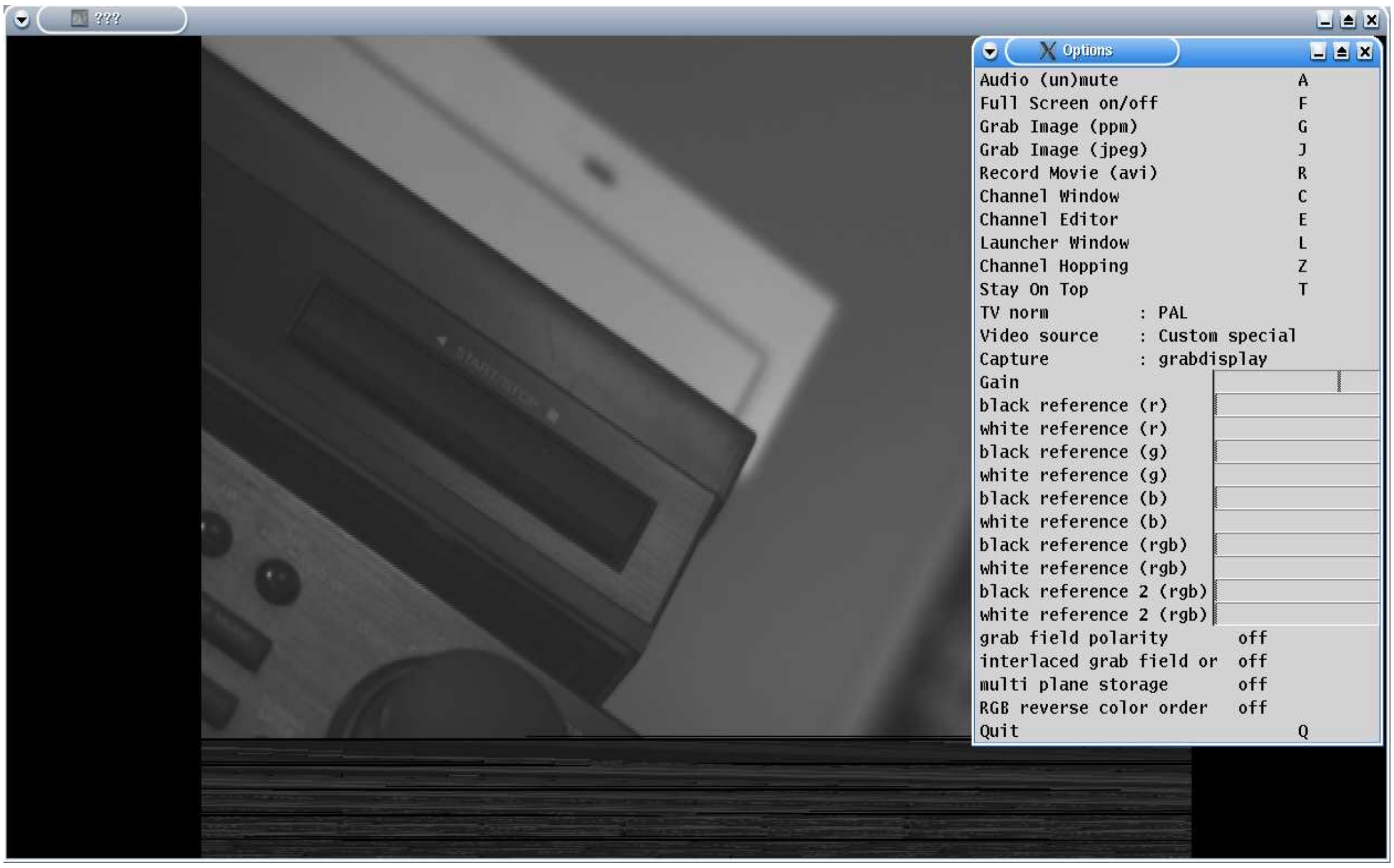


```
static int
via_v4l2_do_ioctl(struct inode *inode, struct file *file,
                        unsigned int cmd, void *arg)
//(void *id, unsigned int cmd, void *arg)
{
        struct file_data *fh    = file->private_data;
        struct via_card *card = fh->card;
//.     unsigned long flags;
//.     int err;

        //struct via_open_list *olist = (struct via_open_list *) id;
        //struct via_card *card = olist->card;

        //assert (id != NULL);
        assert (card != NULL);

        /* handle all private ioctls seperately */
        if (_IOC_NR (cmd) >= BASE_VIDIOC_PRIVATE)
                return via_priv_ioctl (card, cmd, arg);
        /* don't let anything be done if dcf is not loaded */
        if (card->dcf_loaded==0){
                via_info("[--> Use the dcf utility to load a DCF file before trying to g
                return -EINVAL;
        }
        switch (cmd) {
        /* do all ioctls related to streaming elsewhere */
        case VIDIOC_STREAMON:
        case VIDIOC_STREAMOFF:
        case VIDIOC_REQBUFS:
        case VIDIOC_QUERYBUF:
        case VIDIOC_QBUF:
        case VIDIOC_DQBUF:
                return via_stream_ioctl (inode, file, cmd, arg);

        case VIDIOC_QUERYCAP: {
                //struct v4l2_capability *buf = (struct v4l2_capability *) arg;
                struct v4l2_capability *cap = arg;
                memset(cap,0,sizeof(*cap));

                strcpy(cap->driver,card->vdev.name);
                strncpy(cap->card,card->vdev.name,sizeof(cap->card));
                sprintf(cap->bus_info,"PCI:%s",card->pdev->slot_name);
```

```
gcc -D__KERNEL__ -I/usr/src/linux-2.4.20/include -Wall -Wstrict-prototypes -Wno-trigraphs -O2 -fno-strict-aliasing -fno-common -fomit-frame-pointer -pipe -mpreferred-stac
k-boundary=2 -march=athlon -DMODULE -DMODVERSIONS -include /usr/src/linux-2.4.20/include/linux/modversions.h -nostdinc -iwithprefix include -DKBUILD_BASENAME
=via_oldv4l2 -c -o via-oldv4l2.o via-oldv4l2.c
ld -m elf_i386 -r -o meteor2.o via.o via-init.o via-io.o via-fpga.o via-mem.o via-debug.o via-stream.o via-lut.o via-dcf.o via-oldv4l2.o
make[2]: Leaving directory `/usr/local/src/meteor2-1.1'
make[1]: Leaving directory `/usr/src/linux-2.4.20'
*** Success ***
```

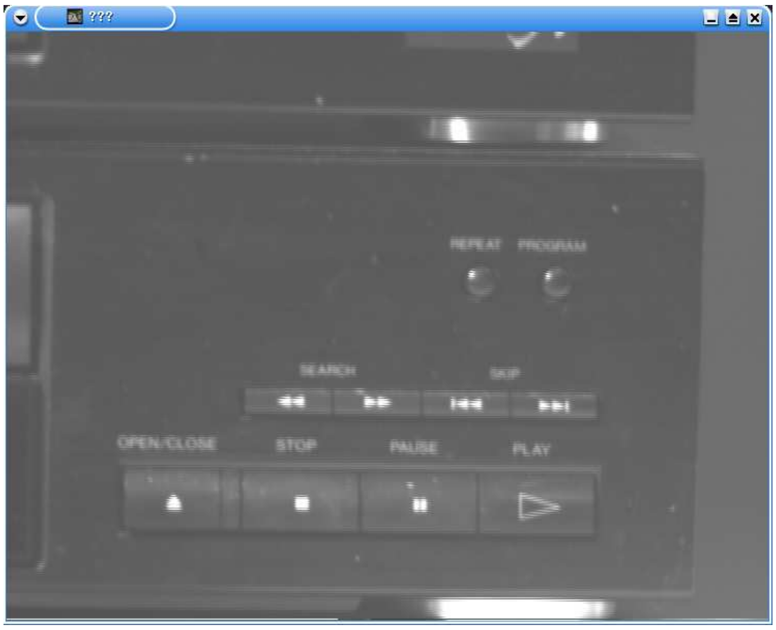
This is the KDevelop IDE, version 3 alpha, where most of the project development took place.

Screenshot S-2 and S-3



```
Audio (un)mute                 A
Full Screen on/off             F
Grab Image (ppm)               G
Grab Image (jpeg)              J
Record Movie (avi)             R
Channel Window                 C
Channel Editor                 E
Launcher Window                L
Channel Hopping                Z
Stay On Top                    T
TV norm          : PAL
Video source     : Custom special
Capture          : grabdisplay
Gain
black reference (r)
white reference (r)
black reference (g)
white reference (g)
black reference (b)
white reference (b)
black reference (rgb)
white reference (rgb)
black reference 2 (rgb)
white reference 2 (rgb)
grab field polarity        off
interlaced grab field or   off
multi plane storage        off
RGB reverse color order    off
Quit                           Q
```

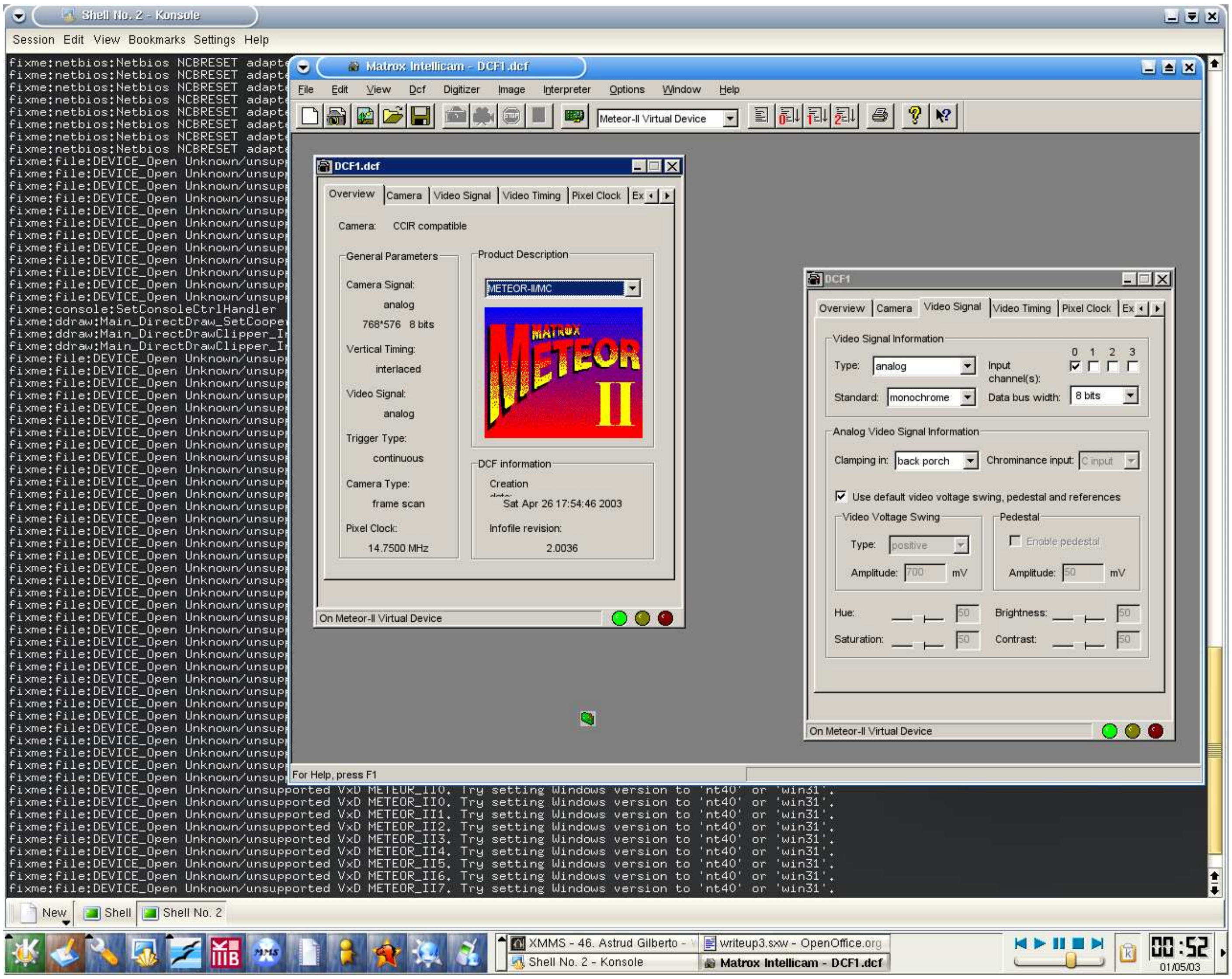
This is xawtv capturing video using the meteor2 driver. The control menu above allows the configurable onboard controls to be changed. The window below shows an image where the onboard black/white references that have been changed since the first shot.



Appendix D

Screenshots

Screenshot S-4



```
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:netbios:Netbios NCBRESET adapte
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:console:SetConsoleCtrlHandler
fixme:ddraw:Main_DirectDraw_SetCoope
fixme:ddraw:Main_DirectDrawClipper_I
fixme:ddraw:Main_DirectDrawClipper_I
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsup
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II0. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II0. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II1. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II2. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II3. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II4. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II5. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II6. Try setting Windows version to 'nt40' or 'win31'.
fixme:file:DEVICE_Open Unknown/unsupported VxD METEOR_II7. Try setting Windows version to 'nt40' or 'win31'.
```

DCF1.dcf — Overview

Camera: CCIR compatible

General Parameters

- Camera Signal: analog
- 768*576 8 bits
- Vertical Timing: interlaced
- Video Signal: analog
- Trigger Type: continuous
- Camera Type: frame scan
- Pixel Clock: 14.7500 MHz

Product Description: METEOR-II/MC

DCF information

- Creation date: Sat Apr 26 17:54:46 2003
- Infofile revision: 2.0036

On Meteor-II Virtual Device

DCF1 — Video Signal

Video Signal Information

- Type: analog
- Input channel(s): 0 [✓] 1 [ ] 2 [ ] 3 [ ]
- Standard: monochrome
- Data bus width: 8 bits

Analog Video Signal Information

- Clamping in: back porch
- Chrominance input: C input
- [✓] Use default video voltage swing, pedestal and references
- Video Voltage Swing: Type: positive; Amplitude: 700 mV
- Pedestal: [ ] Enable pedestal; Amplitude: 50 mV
- Hue: 50
- Saturation: 50
- Brightness: 50
- Contrast: 50

On Meteor-II Virtual Device

For Help, press F1

Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Matrox Intellicam - DCF1.dcf
File Edit View Dcf Digitizer Image Interpreter Options Window Help
Meteor-II Virtual Device
Overview Camera Video Signal Video Timing Pixel Clock Ex
MATROX METEOR II
New Shell Shell No. 2
XMMS - 46. Astrud Gilberto -
writeup3.sxw - OpenOffice.org
Shell No. 2 - Konsole
Matrox Intellicam - DCF1.dcf
00:52
01/05/03

This is Matrox Intellicam running under linux. The required Windows device driver is not yet supported by WINE but the program can still be used to edit and create DCFs.

Appendix E

xawtv output log

```
This is xawtv-3.85, running on Linux/i686 (2.4.20)
visual: id=0x23 class=4 (TrueColor), depth=24
visual: id=0x24 class=4 (TrueColor), depth=24
visual: id=0x25 class=4 (TrueColor), depth=24
visual: id=0x26 class=4 (TrueColor), depth=24
visual: id=0x27 class=4 (TrueColor), depth=24
visual: id=0x28 class=4 (TrueColor), depth=24
visual: id=0x29 class=4 (TrueColor), depth=24
visual: id=0x2a class=4 (TrueColor), depth=24
visual: id=0x2b class=5 (DirectColor), depth=24
visual: id=0x2c class=5 (DirectColor), depth=24
visual: id=0x2d class=5 (DirectColor), depth=24
visual: id=0x2e class=5 (DirectColor), depth=24
visual: id=0x2f class=5 (DirectColor), depth=24
visual: id=0x30 class=5 (DirectColor), depth=24
visual: id=0x31 class=5 (DirectColor), depth=24
visual: id=0x32 class=5 (DirectColor), depth=24
x11: color depth: 24 bits, 3 bytes - pixmap: 4 bytes
x11: color masks: red=0x00ff0000 green=0x0000ff00 blue=0x000000ff
x11: server byte order: little endian
x11: client byte order: little endian
check if the X-Server is local ... **** ok
x11 socket: me=localhost, server=localhost
main: dga extention...
DGA version 2.0
main: xinerama extention...
main: xvideo extention [video]...
main: xvideo extention [image]...
  image format list for port 69
    0x32595559 (YUY2) packed [ok: 16 bit YUV 4:2:2 (packed)]
    0x59565955 (UYVY) packed
    0x32315659 (YV12) planar
    0x30323449 (I420) planar [ok: 12 bit YUV 4:2:0 (planar)]
main: init main window...
main: install signal handlers...
main thread [pid=1485]
main: open grabber device...
x11: 1280x1024, 32 bit/pixel, 5120 byte/scanline, DGA
v4l-conf: using X11 display :0
dga: version 2.0
mode: 1280x1024, depth=24, bpp=32, bpl=5120, base=0xd0000000
/dev/video0 [v4l2]: no overlay support
got sigchild
```

```
waitpid: No child processes

v4l-conf had some trouble, trying to continue anyway

vid-open: trying: v4l2-old...

vid-open: failed: v4l2-old

vid-open: trying: v4l2...

ioctl: VIDIOC_QUERYCAP(driver="meteor2 (#0)";card="meteor2 (#0)";
bus_info="PCI:00:06.0";version=1.1.9;capabilities=0x1 [VIDEO_CAPTURE]): ok

v4l2: open

v4l2: device info:

  meteor2 (#0) 1.1.9 / meteor2 (#0) @ PCI:00:06.0

ioctl: VIDIOC_ENUMINPUT(index=0;name="Custom special";type=CAMERA;audioset=0;tuner=0;
std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=1;name="Mono Channel 0 (R)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=2;name="Mono Channel 1 (G)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=3;name="Mono Channel 2 (B)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=4;name="Mono Channel 3 (R)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=5;name="Mono Channel 4 (G)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=6;name="Mono Channel 5 (B)";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=7;name="Mono Channel 6";type=CAMERA;audioset=0;tuner=0;
std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=8;name="Mono Channel 7";type=CAMERA;audioset=0;tuner=0;
std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=9;name="Component-RGB Channel 0";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=10;name="Component-RGB Channel 1";type=CAMERA;audioset=0;
tuner=0;std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K];status=0x0 []): ok

ioctl: VIDIOC_ENUMINPUT(index=11;name="";type=unknown;audioset=0;tuner=0;std=0x0 [];
status=0x0 []): Invalid argument

ioctl: VIDIOC_ENUMSTD(index=0;id=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,
PAL_K];name="PAL";frameperiod.numerator=1;frameperiod.denominator=25;framelines=625):
ok

ioctl: VIDIOC_ENUMSTD(index=1;id=0x3000 [NTSC_M,NTSC_M_JP];name="NTSC";frameperiod.
numerator=1001;frameperiod.denominator=30000;framelines=525): ok

ioctl: VIDIOC_ENUMSTD(index=2;id=0x7f0000 [SECAM_B,SECAM_D,SECAM_G,SECAM_H,SECAM_K,
SECAM_K1,SECAM_L];name="SECAM";frameperiod.numerator=1;frameperiod.denominator=25;
framelines=625): ok

ioctl: VIDIOC_ENUMSTD(index=3;id=0x0 [];name="";frameperiod.numerator=0;frameperiod.
denominator=0;framelines=0): Invalid argument

ioctl: VIDIOC_ENUM_FMT(index=0;type=VIDEO_CAPTURE;flags=0;description="8 bpp gray";
pixelformat=0x59455247 [GREY]): ok

ioctl: VIDIOC_ENUM_FMT(index=1;type=VIDEO_CAPTURE;flags=0;description="15 bpp RGB, le";
pixelformat=0x4f424752 [RGBO]): ok

ioctl: VIDIOC_ENUM_FMT(index=2;type=VIDEO_CAPTURE;flags=0;description="15 bpp RGB, be";
pixelformat=0x51424752 [RGBQ]): ok

ioctl: VIDIOC_ENUM_FMT(index=3;type=VIDEO_CAPTURE;flags=0;description="16 bpp RGB, le";
pixelformat=0x50424752 [RGBP]): ok

ioctl: VIDIOC_ENUM_FMT(index=4;type=VIDEO_CAPTURE;flags=0;description="16 bpp RGB, be";
pixelformat=0x52424752 [RGBR]): ok

ioctl: VIDIOC_ENUM_FMT(index=5;type=VIDEO_CAPTURE;flags=0;description="24 bpp RGB, le";
pixelformat=0x33524742 [BGR3]): ok

ioctl: VIDIOC_ENUM_FMT(index=6;type=VIDEO_CAPTURE;flags=0;description="32 bpp RGB, le";
pixelformat=0x34524742 [BGR4]): ok

ioctl: VIDIOC_ENUM_FMT(index=7;type=VIDEO_CAPTURE;flags=0;description="32 bpp RGB, be";
pixelformat=0x34424752 [RGB4]): ok
```

```
ioctl: VIDIOC_ENUM_FMT(index=8;type=VIDEO_CAPTURE;flags=0;description="4:2:2 packed,
YUYV";pixelformat=0x56595559 [YUYV]): ok

ioctl: VIDIOC_ENUM_FMT(index=9;type=VIDEO_CAPTURE;flags=0;description="4:2:2 packed,
UYVY";pixelformat=0x59565955 [UYVY]): ok

ioctl: VIDIOC_ENUM_FMT(index=10;type=VIDEO_CAPTURE;flags=0;description="4:2:2 planar,
Y-Cb-Cr";pixelformat=0x50323234 [422P]): ok

ioctl: VIDIOC_ENUM_FMT(index=11;type=VIDEO_CAPTURE;flags=0;description="4:2:0 planar,
Y-Cb-Cr";pixelformat=0x32315559 [YU12]): ok

ioctl: VIDIOC_ENUM_FMT(index=12;type=VIDEO_CAPTURE;flags=0;description="";
pixelformat=0x00000000 [....]): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=9963795;type=INTEGER;name="Gain";minimum=0;maximum=4;step=1;
default_value=3;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=0;type=unknown;name="Unsupported control";minimum=0;
maximum=0;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=9963800;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963801;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument
```

```
ioctl: VIDIOC_QUERYCTRL(id=9963802;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963803;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963804;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963805;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963806;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=9963807;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217728;type=INTEGER;name="black reference (r)";minimum=0;
maximum=255;step=1;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217729;type=INTEGER;name="white reference (r)";minimum=0;
maximum=255;step=1;default_value=245;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217730;type=INTEGER;name="black reference (g)";minimum=0;
maximum=255;step=1;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217731;type=INTEGER;name="white reference (g)";minimum=0;
maximum=255;step=1;default_value=245;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217732;type=INTEGER;name="black reference (b)";minimum=0;
maximum=255;step=1;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217733;type=INTEGER;name="white reference (b)";minimum=0;
maximum=255;step=1;default_value=245;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217734;type=INTEGER;name="black reference (rgb)";
minimum=0;maximum=255;step=1;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217735;type=INTEGER;name="white reference (rgb)";
minimum=0;maximum=255;step=1;default_value=245;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217736;type=INTEGER;name="black reference 2 (rgb)";
minimum=0;maximum=255;step=1;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217737;type=INTEGER;name="white reference 2 (rgb)";
minimum=0;maximum=255;step=1;default_value=245;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217738;type=BOOLEAN;name="grab field polarity";minimum=0;
maximum=1;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217739;type=BOOLEAN;name="interlaced grab field order";
minimum=0;maximum=1;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217740;type=BOOLEAN;name="multi plane storage";minimum=0;
maximum=1;step=0;default_value=1;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217741;type=BOOLEAN;name="RGB reverse color order";
minimum=0;maximum=1;step=0;default_value=0;flags=0): ok

ioctl: VIDIOC_QUERYCTRL(id=134217742;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217743;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217744;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217745;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217746;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217747;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217748;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217749;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217750;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217751;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217752;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument
```

```
ioctl: VIDIOC_QUERYCTRL(id=134217753;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217754;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217755;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217756;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217757;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217758;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

ioctl: VIDIOC_QUERYCTRL(id=134217759;type=unknown;name="";minimum=0;maximum=0;step=0;
default_value=0;flags=0): Invalid argument

vid-open: ok: v4l2

[...]

main: creating windows ...

main: init frequency tables ...

freq: reading /usr/share/xawtv/Index.map

main: read config file ...

freq: newtab 5

main: checking for vidmode extention ...

main: checking for lirc ...

lirc: not enabled at compile time

main: checking for midi ...

main: adding kbd hooks ...

main: mapping main window ...

xt: pointer show

main: initialize hardware ...

ioctl: VIDIOC_G_INPUT(int=0): ok

ioctl: VIDIOC_G_CTRL(id=9963795;value=3): ok

ioctl: VIDIOC_G_CTRL(id=134217728;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217729;value=255): ok

ioctl: VIDIOC_G_CTRL(id=134217730;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217731;value=255): ok

ioctl: VIDIOC_G_CTRL(id=134217732;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217733;value=255): ok

ioctl: VIDIOC_G_CTRL(id=134217734;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217735;value=255): ok

ioctl: VIDIOC_G_CTRL(id=134217736;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217737;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217738;value=1): ok

ioctl: VIDIOC_G_CTRL(id=134217739;value=1): ok

ioctl: VIDIOC_G_CTRL(id=134217740;value=0): ok

ioctl: VIDIOC_G_CTRL(id=134217741;value=0): ok

main: parse channels from config file ...

xt: handle_pending:  start ...

video: tv(+root): DestroyNotify

video: tv(+root): DestroyNotify

video: shell: size 384x288+0+0

gd: init

blit: init
```

```
blit: gl: DRI=Yes

blit: gl: texture max size: 2048

blit: resize 384x288

gd: config 384x288 win=2a00189

blit: gl: extention GL_EXT_bgra is available

blit: gl: extention GL_EXT_bgra is available

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x56595559 [YUYV];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 16 bit YUV 4:2:2 (packed) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x32315559 [YU12];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 12 bit YUV 4:2:0 (planar) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x33524742 [BGR3];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 24 bit TrueColor (LE: bgr) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x34524742 [BGR4];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 32 bit TrueColor (LE: bgr-) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x33424752 [RGB3];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 24 bit TrueColor (BE: rgb) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x34524742 [BGR4];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 32 bit TrueColor (LE: bgr-) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x33524742 [BGR3];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 24 bit TrueColor (LE: bgr) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x33424752 [RGB3];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 24 bit TrueColor (BE: rgb) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x33524742 [BGR3];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 24 bit TrueColor (LE: bgr) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x00000000 [....];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 32 bit TrueColor (BE: -rgb) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x34524742 [BGR4];fmt.pix.field=ANY;fmt.pix.bytesperline=0;fmt.pix.
sizeimage=0;fmt.pix.colorspace=unknown;fmt.pix.priv=0): Invalid argument

setformat: 32 bit TrueColor (LE: bgr-) (384x288): failed

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x59455247 [GREY];fmt.pix.field=ANY;fmt.pix.bytesperline=384;fmt.pix.
sizeimage=110592;fmt.pix.colorspace=unknown;fmt.pix.priv=0): ok

v4l2: new capture params (384x288, GREY, 110592 byte)

setformat: 8 bit StaticGray (384x288): ok

grabdisplay: using "8 bit StaticGray"

video: root: ConfigureNotify

cmd: "capture" "overlay"

gd: start [2]

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=384;fmt.pix.height=288;fmt.pix.
pixelformat=0x59455247 [GREY];fmt.pix.field=ANY;fmt.pix.bytesperline=384;fmt.pix.
sizeimage=110592;fmt.pix.colorspace=unknown;fmt.pix.priv=0): ok

v4l2: new capture params (384x288, GREY, 110592 byte)

setformat: 8 bit StaticGray (384x288): ok

main: setting defaults

ioctl: VIDIOC_S_INPUT(int=0): ok

ioctl: VIDIOC_S_STD(std=0xff [PAL_B,PAL_B1,PAL_G,PAL_H,PAL_I,PAL_D,PAL_D1,PAL_K]): ok

main: enter main event loop...

PropertyNotify _XAWTV_STATION

PropertyNotify WM_NAME

PropertyNotify _NET_WM_VISIBLE_NAME

video: shell: size 384x288+3+26

v4l2: start ts=1051670730769324000

blit test: 24 bit TrueColor (LE: bgr)

blit test: 24 bit TrueColor (BE: rgb)

blit test: 24 bit TrueColor (LE: bgr)

blit test: 32 bit TrueColor (BE: -rgb)

blit test: 32 bit TrueColor (LE: bgr-)

blit test: 8 bit StaticGray

convert-in : 384x288 8 bit StaticGray (size=110592)

convert-out: 384x288 32 bit TrueColor (LE: bgr-) (size=442368)

blit: 384x288/[8 bit StaticGray] => X11 via [32 bit TrueColor (LE: bgr-)]

blit: putframe

    12 fps blit: putframe

[....]

video: shell: ClientMessage

CloseMainAction: received WM_DELETE_WINDOW message

cmd: "capture" "off"

gd: stop

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=768;fmt.pix.height=576;fmt.pix.
pixelformat=0x59455247 [GREY];fmt.pix.field=ANY;fmt.pix.bytesperline=768;fmt.pix.
sizeimage=442368;fmt.pix.colorspace=unknown;fmt.pix.priv=0): ok

v4l2: new capture params (768x576, GREY, 442368 byte)

setformat: 8 bit StaticGray (768x576): ok

ioctl: VIDIOC_S_FMT(type=VIDEO_CAPTURE;fmt.pix.width=768;fmt.pix.height=576;fmt.pix.
pixelformat=0x59455247 [GREY];fmt.pix.field=ANY;fmt.pix.bytesperline=768;fmt.pix.
sizeimage=442368;fmt.pix.colorspace=unknown;fmt.pix.priv=0): ok

v4l2: new capture params (768x576, GREY, 442368 byte)

ioctl: VIDIOC_REQBUFS(count=1;type=VIDEO_CAPTURE;memory=MMAP): ok

ioctl: VIDIOC_QUERYBUF(index=0;type=VIDEO_CAPTURE;bytesused=0;flags=0x0 [];
field=unknown;;timecode.type=0;timecode.flags=0;timecode.frames=0;timecode.seconds=0;
timecode.minutes=0;timecode.hours=0;timecode.userbits="";sequence=0;memory=unknown): ok

v4l2: buf 0: video-cap 0x0+442368, used 0

ioctl: VIDIOC_QBUF(index=0;type=VIDEO_CAPTURE;bytesused=0;flags=0x0 [];field=unknown;;
timecode.type=0;timecode.flags=0;timecode.frames=0;timecode.seconds=0;timecode.
minutes=0;timecode.hours=0;timecode.userbits="";sequence=0;memory=unknown): ok

ioctl: VIDIOC_STREAMON(int=1): ok

ioctl: VIDIOC_DQBUF(index=0;type=VIDEO_CAPTURE;bytesused=0;flags=0x105 [MAPPED,DONE,
TIMECODE];field=unknown;;timecode.type=2;timecode.flags=0;timecode.frames=20;timecode.
seconds=3;timecode.minutes=0;timecode.hours=0;timecode.userbits="";sequence=0;
memory=unknown): ok

convert-in : 768x576 8 bit StaticGray (size=442368)

convert-out: 768x576 32 bit TrueColor (LE: bgr-) (size=1769472)

v4l2: close
```