"Will Code For Hardware" – Supporting the Unsupported and Getting Cool Toys

Nathan I. Laredo
500 Northside Cir NW #BT
Atlanta, Georgia, 30309
USA
laredo@gnu.org, http://mpeg.openprojects.net

Abstract

This paper discusses the development of the Stradis MPEG2 hardware decoder driver for Linux. It addresses development process, driver architechture, and problems encountered despite the full technical support of Stradis and full documentation on all of the chips used on the board. It will also cover some techniques used to protect intellectual property of the manufacturer as well as well as covering the optimization of the driver for real use after the initial just-get-it-working hack. The Video4Linux API was used for the driver, and an overview of the extensions to the API will also be presented.

1 Introduction

I was first contacted by Doug Ledford who was swamped with far more projects than the average human is capable of supporting in early 1999 about some new toys he had been given in exchange for writing a Linux driver. It was a good deal for Stradis. The Chief Technical Officer at Stradis, Chris Hodges, mailed Doug a package with two MPEG-2 decoder boards, the Philips SAA7146 PCI Bridge Software API Description (which turned out to be only useful to understand the Windows driver), the Philips SAA7146 Data Sheet (a book actually), a CD-R with some MPEG test sequences, the Windows drivers and software, and the actual source code to the Windows playback application. I thought to myself, "This should be a piece of cake," so I told Doug that I would be happy to take over this project he would never have time for. I received a phone call from Stradis just as I was crawling into bed. They decided they wanted Linux support because they had a customer that was asking for it, and I learned that they have no great love for Microsoft either. The driver was to be written in exchange for two $1000 value 50Mbps MPEG2 decoder boards.

I received a package from Red Hat with the hardware just before I started packing for my move to Atlanta after an unusual four years in the Air Force. I thumbed through the documentation, installed the hardware, and tried the Windows driver. It failed. Since Stradis was in Atlanta, I decided to just wait until I got there to get it working.

I moved into my new apartment March 1st and spent two weeks getting 30 boxes of junk sorted out and new furniture purchased. I met with the CEO and CTO of Stradis two weeks later. After sticking an oscilloscope on the card, we discovered that all of my problems were caused by my PCI bus running at 37MHz, an unfortunate consequence of my I-have-a-faster-machine-than-you-have ego trip. The Windows driver has since been hacked to work with the faster speed. I honestly had zero code written at this point, but I was able to pick up a manual for the MPEG decoder chip on the board.

2 What is MPEG?

These days when most people hear the term MPEG, they immediately think of how large their collection of MP3 files is, but they do not realize that their favorite audio format is really just a small part of a larger audio/video compression algorithm that can take the 31MB/sec of video data and 180K/sec of audio data and compress it into a bitstream anywhere from 50kbps to 1.8Mbps for MPEG1, and 1.5Mbps to 50Mbps and beyond for MPEG2. MPEG1 lends itself to low bitrate material where MPEG2 is designed for high quality material at higher bitrates.

MPEG is not the first effort at video compression. Even as early as the 1950s people were trying to compress video. Then the issue was simple. All the televisions were black and white, and to transmit color would have meant using three full channels of video, or 15MHz. With only twelve channels on the tuner, that meant that for color, there could only be four channels of television. They had to squeeze 15MHz into 5MHz and invented a clever analog technique to do so. They patented it.

Now we are in an age where we want over two hundred channels on television, and we want our remote newsfeeds the instant the news happens. MPEG is a way that people can send many times as much info through the same space the analog signal took up, and maintain decent to near-perfect quality through the whole path

MPEG achieves its savings by transforming all of the component digital data (which itself may be compressed via subsampling the chrominance data) into the frequency domain, and through careful coding of the coefficients so that the most common will compress the most, as well as using inter-frame dependencies which include forward and bidirectional predictive frames with motion compensation.

The savings are impressive, and so is the processing cost. With the reference software MPEG2 decoder from the MPEG group, on my Pentium III 550MHz cpu, I can barely reach 10 frames per second on the average 8Mbps MPEG2 stream. Granted that is not optimal code, but it does give some indication as to the order of magnitude of the work involved to decode an MPEG bitstream.

I had been working on optimizing this software MPEG2 decoder at the same time I got the offer to write a driver for the Stradis decoder in exchange for hardware. I jumped at the chance to get full speed decoding at potentially zero cpu cost. I could advance my DVD decoding project quite a bit without having to worry about the expensive operation of decoding the video stream.

3 The Blank Page

Every writer hates to stare at a blank page, especially when writing a paper for a conference with the deadline the same day. I had zero code, and the project seemed a more daunting task than I thought I could handle. Using bttv.c as a reference, I hacked out a skeleton of a driver that did nothing but printk for everything. I followed the Windows driver initialization code and poked that into the skeleton after I borrowed the pci device detection code from the bttv driver.

I continued hacking in code, replacing every printk I had with code to actually do something. I quickly progressed from no code to something that closely resembled a driver that could very well work, but I had no way to test it fully without finishing more of it. The only thing that I knew that it did was detect all the cards on the pci bus and enable bus master mode for each of them. So, from a feeling that I would never finish, I reached a point where I thought that I was almost finished with the driver, but I soon discovered that nothing that I had written actually did what it was supposed to do.

4 Bugs Are Everywhere, Even in Data Books

After about five days of debugger's block, my brain was about to explode. This was defused by an offer from Chris Hodges to stick the old oscilloscope on the card to see what it was actually doing. The first thing that I learned is that Windows apparently has a concept of breakpoints and watches everywhere, and I only had printk and learned to return -1 on module init to break out, with usual precautions against leaking kernel memory.

In theory, everything in my driver should have been working perfectly. In reality, I produced nothing. The card was nicely designed with a LED that would turn off as soon as the FPGA was properly programmed. Of course in the end I discovered that it was not the FPGA loader code that was wrong, it was the code that wrote to the 16-bit interface of the SAA7146.

Philips, in their infinite wisdom, designed a chip that would raise an IRQ when a transfer was "finished" before the transfer was complete. Not only that, but I was guilty of treating a 32-bit pointer as if it were an 8-bit pointer, so between my crashing the kernel all day and the Philips chip not working as the manual describes, it was a wonder that I made any progress at all.

Since there was a direct path from the IBM MPEG decoder chip through the FPGA to the Philips SAA7121 video encoder chip, and the FPGA was not required to access the I2C bus, I decided the next order of business was to get the SAA7121 programmed properly for output. The code was already written but like the FPGA code, it did not work either. The SAA7146 has a smart I2C interface and a really annoyingly wrong manual. After much oscilloscope poking on the card, we discovered that it also suffered from the same "I'm done now but not really" problem as the other parts. It also suffered from a severe case of poor examples in the manual.

5 It Works! It Really Works!

The way the Stradis card is designed, the FPGA must be programmed properly before anything else can be tested, so once we got the FPGA programmed, I fleshed out the code to program the microcode into the IBM MPEG decoder chip. I had already written it, but never actually called the code since the F-PGA was not working before. I was still on my second visit to Stradis at this point and felt motivated to pull an all-night hack session after seeming to get the IBM chip programmed on the first try. I am not quite sure why the hours between 1AM and 4AM are so productive, but I am quite thankful that they exist.

The IBM chip seemed to get programmed on the first try. I was lucky, but during my all night hack session I got nothing more interesting than colorbars produced directly by the SAA7121 video encoder. The IBM chip would produce output that resembled feeding random data to it. At this point, everything was still being done by polling. No IRQ routines were yet written. This was day two at Stradis. Around lunchtime, I realized that I was walking all over my DMA buffer. I was filling it with new data while the previous block transfer was in progress.

After lunch, I wrote the code to poll for block transfer complete before loading the new data into the buffer, and as if by magic, I saw beautiful full-frame-rate NTSC output from the elementary MPEG2 streams. No audio yet, but beautiful video. Audio would prove to be a thorn in my side, and in the end a trivial bug, but that is a story for later.

6 Total Time Elapsed: 5 days. No Interruptions

The people at Stradis were impressed. I found myself with a job offer. I was also given an opportunity to go to Las Vegas to the National Association of Broadcasters 1999 convention to demonstrate the card running in Linux. Needless to say, I accepted the free trip. The show was mid-April, and between the design of the new revision of the decoder board and preparing to ship everything to the show and setup the booth, everyone at Stradis was quite busy.

At this point, the driver was working for MPEG video and still using 99% of the CPU because it used no interrupts. I hacked out some code to split a transport stream into audio and video streams since the IBM chip required them to be written to different ports. The driver needed to be moved to an interrupt-based architecture as well to support the audio properly. The IBM chip has a very small audio buffer and a large video buffer. Threshold interrupts could be setup to tell the driver what buffer was starving for data.

Interrupts were a bit more complicated than I first imagined. Thanks to Alan Cox, I learned the virtues of the wait queue. Without `wake_up_interruptible() and interruptible_sleep_on()`, I would have never figured out how to structure the driver. The driver also needed a true ring buffer, but that would have to wait until after NAB99 which was quickly approaching.

With interrupt based code, the driver could go from sitting and polling for a transfer complete flag to letting the kernel do all the work in the background on each interrupt. A big buffer was setup to hold the data from the user, and then the write() code would just return. At this point on the next "I need more data" interrupt from the driver, the interrupt routine would transfer the data to the appropriate dma buffer space and tell the card that another 32k is ready for transfer. This was all great in theory, but a pain to get working. The potential for IRQ loops was high since this wouldn't be the fastest interrupt service routine in the world, given that a bug in the SAA7146 would actually trigger the "transfer complete" interrupt before it was really complete, so I would have to poll and wait for it to really finish on receipt of the interrupt.

Just before the show I got the interrupt-based driver working perfectly on my dual PII400 machine. It was working on less than 0.4% cpu, and I solved that elusive audio bug. Apparently in writing the audio microcode I was trying to do 16-bit writes when the chip could only take an 8-bit write so half of the microcode was being lost to the great bit bucket in the sky.

Now I could take it to Las Vegas and show of a driver that uses next to zero cpu and does both video and sound, and in the meantime I hacked together the extra few lines of code it took to do a fixed-size VGA video overlay to frame-buffer memory. Little did I realize how absolutely horrid the performance of the cheap little show machine hard drive was. On the NAB99 show floor I was hacking the driver back to do no interrupts and no audio only minutes before the show was to open.

7 "One must plan that there will be lots of bugs" - Frederick P. Brooks, Jr.

Overall the major mysterious bugs in the driver were attributable to pointer math being done behind my back. Treating a 32-bit pointer as an 8-bit pointer is not wise, especially if you must wait through fsck on a 5.25" single platter IDE 6.4GB hard drive. The ring buffer code was no exception. While in North Carolina in May for Linux Expo, I was sitting at Red Hat trying to figure out what was causing the random noise every time the buffer wrapped around. I banged my head against the desk, played five levels of Ms. Pac Man, drank about twelve cans of orange juice, only to find another elusive pointer error.

Dr. Michael Fulbright at Red Hat was nice enough to let me demo my driver in the RHAD Labs booth at Linux Expo. This demo was complete with audio and video and Star Wars MPEG trailer. By this point my machine had become a dual Pentium III 550, so I knew that the real question would not be answered until I got home from the conference and tried the driver on the piece of junk 5.25" IDE drive. Indeed I had a good solid driver that even stood up as the slow beast began to swap.

Now the question came up from Stradis, "This is nice that it all works, but you're still using the console to do everything. We think it should have some graphical user interface like the Windows version with VCR-like buttons." But, luckily I'd just listened to Miguel and Federico and decided that GNOME was the way to go. So far, this part has not materialized, but some day it will. I like to keep my hands in the kernel if possible, and the original agreement was to write the driver in exchange for two cards, and it mentioned nothing of a user interface, but all the same, eventually for completeness, I'll get around to it.

The driver also needed statistical collection and frame and timecode tracking, as well as other functions not specifically required for the actual playback of the MPEG files, but instead for the applications that the board was actually intended for when installed in a professional environment.

These features would wait until I got my hands on the actual production card, which was sufficiently different that major portions of the driver had to be rewritten to deal with both the new cards and the old cards at the same time.

Not to mention there were at least 4 different versions of the production cards, but at least they put a eeprom on the new cards with a list of installed features, what FPGA files to load and so on.

8 Can I Buy You a Thought?

I knew it would happen from the start. Stradis came back to me and said, "You know, it would probably be better if we didn't provide our competition with everything that they needed to reverse engineer our board and produce a cheap knockoff by taking advantage of all the time and effort that we've invested in the board." Ok. This is fair enough. "There's a lot of effort put into the FPGA code for example, and withholding that would make it next to impossible to do anything with the driver without obtaining a proper FPGA licence." Enter the next two weeks I spent moving all the intitialization code around so that the FPGA and IBM microcode could all be loaded runtime instead of included in the kernel itself.

Yet another chance to crash my machine. In retrospect, it may have been wise for me to setup a dedicated NFS server and use NFS root for kernel development so that long delays from crashes would be non-existant, but I have limited means at home. The new microcode loader also allows a user to change to new FPGA code without ever recompiling the driver again. This could prove useful in a world of people who would not want to hack kernels. The board is soft enough to allow one pixel wide timing changes just by updating the FPGA code.

So what do we gain by allowing the firmware to be uploaded instead of compiling in the kernel? It means that some Asian knock-off company will have a harder time reverse engineering the board without the FPGA bitfile which not only lets the Stradis board be more flexible than any other board, but it also was a considerable investment in hardware development time. Giving anyone this firmware allows them to potentially copy the Stradis card and mass produce it in such a way that Stradis could not compete as a small startup company and would fail. This eases the fears of Stradis relating to the idea of an open source driver.

Of course the final production boared required a rewrite of all this code too.

9 The Alan Cox Seal of Approval

After all is said and done, I still had to get the elusive Alan Cox Seal of Approval. I sent Alan a copy of the API changes, and naturally he sent me back a one line cryptic reply as if I were some coding machine. Sometime around Linux Expo, I lost all of Alan's comments on the proposed changes to the API.

Yes, there were other comments outside of email longer than one line.

Once again, I sent an email with the proposed API (which just happened to exactly match the one I had already implemented), and he obligingly sent me back yet another suggestion. After a short chat, everyone seemed to be happy and Alan hinted that we could see the new stuff in the 2.2.11 kernel. I had earlier met Linus when he did a keynote at a conference in Atlanta and found out that there would be no problems getting the driver itself into the 2.2 tree.

Of course, at the same time this happened, Stradis was developing the new production cards. Between some traveling to other countries and to visit friends in other states, I never got around to finishing the changes for the production card until the day before the ALS paper deadline, so now it's looking more like 2.2.13 or maybe 2.2.14 could see the driver according to Alan.

10 The New Board

Just after I returned from Ottawa, I got the final production board from Stradis. The diff file from the old windows source tree to the new windows source tree was in excess of 200K. I had the board for two days before my trip to New York and discovered that some of the functions that I never used in the original driver did not really work. While in New York I attemted to write as much of the driver as I could on my laptop without access to the actual hardware. This was supposed to be a relaxing vacation for me and writing a driver without the hardware can really give a person a headache.

Following my trip to New York, I had another trip planned to Durham, North Carolina to visit a little software company based there. I still did not have the hardware with me and continued coding blind. At that company, someone introduced me to color vim syntax highlighting. Suddenly I felt ultra productive. Hey, it highlighted the tags in this paper as I was working on it as well, so I was hooked.

Sometime during this period Stradis started to get impatient claiming that I made promises that I could not keep and they had customers that were abandoning their Linux projects for NT. Having finished so much work on the road without the real hardware, I knew that once I got back, I could work with Stradis and have the new card working before the following Monday after I returned.

I returned from North Carolina late Thursday and went to work at the offices of Stradis the next day. Luckily, I had the new card working by 3PM the same day, and Stradis was happy, but no audio.

The new card had some really nice audio features, the nicest of which is AES3 digital audio output (which I could test by plugging into the S/PDIF connector on my home amplifier), so I went home to continue my work. On listening to the balanced audio output, I noticed some noise that sounded more like digital audio, so I plugged it in and found that the left balanced audio connector was mislabeled and was really the digital audio output. I told myself to never trust the labels on hand-made cables.

Happy with the digital audio success, I decided to sleep and attack the analog audio in the morning. As it turned out, a fresh look at the 200K diff file for the Windows driver revealed the missing register that needed to be programmed to turn the volume up on one of the other audio chips on the board.

Success.

11 The Extended Video4Linux API

11.1 VIDIOCSPLAYMODE:

This set the output video mode (PAL/NTSC/SECAM) As well as special features such as GENLOCK, Pause, Fast Forward, Single Frame Step, Slow Motion, Normal Play, and Non-Interlace Mode, Black Frame, and Colorbar output.

11.2 VIDIOCSWRITEMODE:

This tells the driver what type of data to expect on write(). The user-mode code could write compressed MPEG audio, MPEG video, an on-screen-display overlay bitmap (alpha blending done in hardware by the IBM MPEG decoder, written in native format), Closed Captioning or Teletext bytes to overlay using the SAA7121 video output, or MJPEG (which was for other devices, not the Stradis decoder)

11.3 VIDIOCGPLAYINFO:

This returns information about the current stream being played. Info returned includes frames output since decode began, the horizontal and vertical unscaled sizes, current SMPTE timecode the current picture type, current temporal reference, and the last user data found in the compressed stream.

11.4 VIDIOCSFIRMWARE:

This was added to load new FPGA bitfiles and IBM Decoder microcode once the device driver is loaded. A driver can use the "loadwhat" field to address the given code to a specific area of a card.

12 Usage

12.1 The Simple Case

```
cat susi_080.m2v >/dev/video0
```

for sending a video elementary stream to the decoder card for playback without video framebuffer overlay.

12.2 The Most Complicated Case

```
parsempeg french.mpeg 224 192
```

for sending a transport stream with video on PID 224 and audio on PID 192 for playback without the video framebuffer overlay.

MPEG playback at its finest. Multiple opens were hacked in around an upper level Video4Linux restriction in order to allow a user to use separate processes for sending the MPEG file to the card for decoding and doing other controls with the card. This way a Video4Linux-compliant program such as tvset could be used for the VGA overlay while cat or a more complicated MPEG parser is used to actually send the MPEG sequence to the card for decoding.

Still remaining to implement is the MMAP capture interface of Video4Linux and a poll routine to allow the user to use select to synchronize events on the pending output decoded video frame.

13 Conclusion

At the start, I began to doubt that I would ever be able to get the hardware to do anything and that Stradis would ask for the cards back in frustration. The moral of the story is always follow pointers and never forget to check your references, and if you ever offer to write a device driver in exchange for free hardware, even the coolest hardware in the world, you are grossly underestimating your own worth. The mythical man month really was right after all.

14 Acknowledgments

Special thanks go to Alan Cox for his sage advice, and to the countless people who helped edit this paper.

15 Availability

The Stradis 4:2:2 Hardware MPEG-2 Decoder driver and the development diary is at

`http://mpeg.openprojects.net/`