The Linux DVB API

Marcus and Ralph Metzler

August 15, 2002

The Linux DVB API

The first API for DVB cards that was used for DVB PCI cards in 1999 was an extension of the Video4Linux API which was primarily developed for frame grabber cards. As such it was not really well suited to be used for DVB cards and their new features like recording MPEG streams and filtering several section and PES data streams at the same time.

In early 2000, Convergence was approached by Nokia with a proposal for a new standard Linux DVB API. As a commitment to the development of terminals based on open standards, Nokia and Convergence made it available to all Linux developers and published it on `http://www.linuxtv.org/` in September 2000. Convergence is the maintainer of the Linux DVB API. Together with the LinuxTV community, the Linux DVB API will be constantly reviewed and improved upon. With the Linux driver for the Siemens/Hauppauge DVB PCI card Convergence provides a first implementation of the Linux DVB API.

This paper will cover the latest development in the DVB API for Linux. We will start with a short introduction to the DVB standard concentrating on the technical details that are most relevant for the construction of a useful API. This includes details about the tuning and demuxing process of DVB-S, DVB-T and DVB-C signals, information about the incoming transport streams (TS) and a short introduction to other relevant parts of the MPEG standard.

After this introduction we will know enough to understand what is needed in a kernel driver in order to provide the application programmer with the necessary functions to develop a DVB application that can handle all the tasks that are required by the user to watch television and use the other services provided by the broadcasters. This would include for example such things as EPG, MHP and Internet via satellite. Furthermore, we will go into the details of the kernel driver and the implementation as well as the definition of the API. With the goal that the audience will have an understanding of what is necessary to write a driver for a DVB hardware. We will explain the various parts of a driver implementation. This will include tuner, demuxer, section filters, MPEG decoder and network interface.

Digital Television (DVB- S,T,C)

In this part we are going to introduce some of the terminology of digital video broadcasting (DVB), so that we will understand the various frontend (or tuner) parameters of the API.

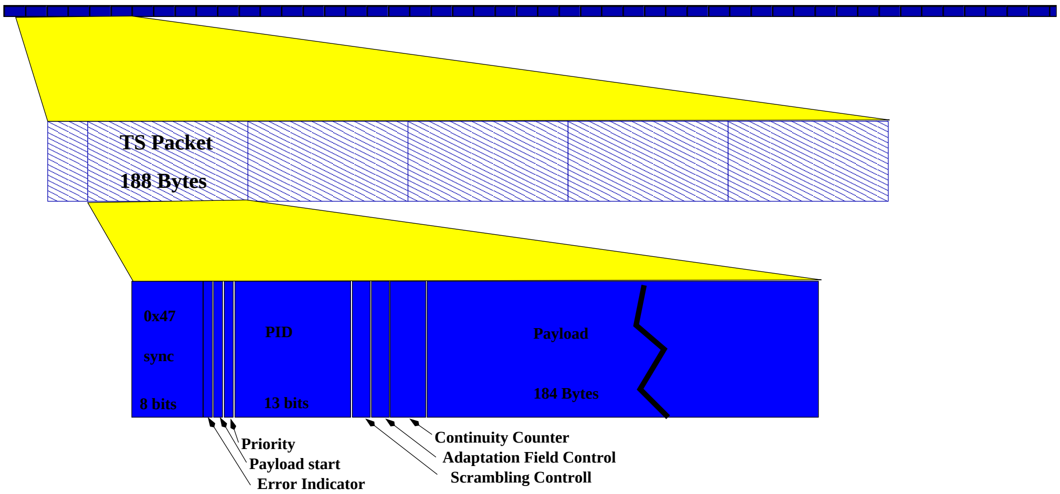

Figure 1: The structure of a Transport Stream

The first step of creating digital television is of course to digitize the analog television signal, or more ideally to use digital recording equipment. This can be done in fairly straight forward ways that we will not discuss in detail. Since these methods usually produce rather large quantities of data, the next step is to compress the digital data. For DVB that means MPEG2 compression of the video and audio data (sometimes AC3 is used for audio). This compressed AV-data is multiplexed along with other data into a so-called transport stream (TS).

A TS consist of packages of 188 bytes length. These packages can contain various forms of data. The majority will carry the audio and video streams of the television programs. Note that one TS can carry many programs depending on the bandwidth of the transmission channel and the bit rate of the individual programs. The AV data is carried in form of packetized elementary streams (PES) that are split into the packets of the TS. Each TS packet has 4 header bytes that, starting with byte one which is alway 0x47 for synching purposes, give information about the data contained in it. Most importantly a PID of 13 bits ($<$ 0x1FFF) indicates the data stream a packet belongs to. So that e.g. all packets with PID 0x60 belong to the audio stream of N-TV on transponder 116 on the Astra satellite. Since the PES also contains packets the header bytes also tell you if a PES packet starts in the current TS packet. The AV packets are multiplexed in such a way that a smooth decoding of the MPEG (or AC3) data is possible.

Apart from the AV data the TS also contains programming information that e.g. tell you which PID belongs to the audio stream of N-TV (a German news channel) on transponder 116 on the Astra satellite. It also possible to send IP data via a TS which allows Internet connections via satellite. Other forms of data include teletext, EPG and MHP information. We will learn more about the packets containing so-called program specific information (PSI) a little later. Now lets have a look at how we get the TS to the receiver.

In order to broadcast the TS we need an error prove method since in such a highly compressed stream with almost no redundancies the slightest errors will ruin the transmission.

The typical bit error rate (BER) should be of the order $10^{-10} - 10^{-12}$. This requires that we use something that is called forward error correction (FEC), which will introduce a calculated redundancy into the signal before broadcasting. The FEC process consist of various steps including randomizing, Reed-Solomon coding (outer coding), Forney interleaving and in the case of terrestial and satellite broadcasting (DVB-T and DVB-S, not cable DVB-C) convolutional coding (inner coding). Without going into further detail, we will just remember that the transmitted signal will have a higher bit rate than the input data. The ratio of input (used) and output (transmitted) bit rate is called code rate R_c or sometimes just FEC (see also figure 2).

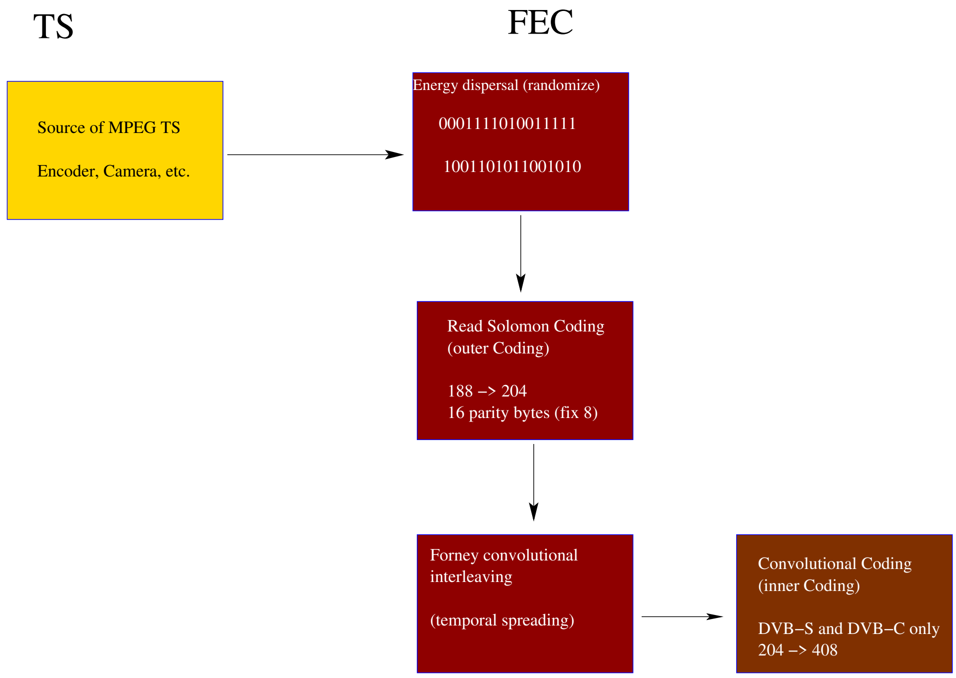

Figure 2: Forward Error Correction

Now that we have taken care of the error correction we have to modulate our signal onto a carrier frequency. Both, the frequency of the signal as well as the circumstances of reception pose different requirements for the type of modulation.

- DVB-S,

  for satellite reception the signal to noise ratio (SNR) can be very low ($<$ 10dB), but there are hardly any echos.

- DVB-C,

  here we have good SNR ($>$ 30db), but lots of echos.

- DVB-T,

  for terrestial reception we want to have mobile reception with simple antennas, so we have to take care of multiple echos, interference and signal variations.

Furthermore, there are other constraints on the transmission channels that come from the legacy of the old analog channels. Those have certain frequency ranges that will have to remain the same for the digital channels. Without explanation we will just say that for terrestial transmission orthogonal frequency division multiplexing (OFDM) is used for modulation, for cable and satellite transmission quadrature amplitude modulations (QAM) are used which in the case of DVB-S is the simplest case of QAM4, i.e. quadrature phase shift keying (QPSK). For more information see [1].

Let us go on to the receiving end of the the data stream. Here the signal has to be tuned, demodulated and the FEC treatment has to be reversed. In the next section we will see how a receiver is structured and what that will mean for our API.

The inside of a DVB receiver

A DVB PCI card or DVB set-top-box (STB) usually consists of the following main hardware components:

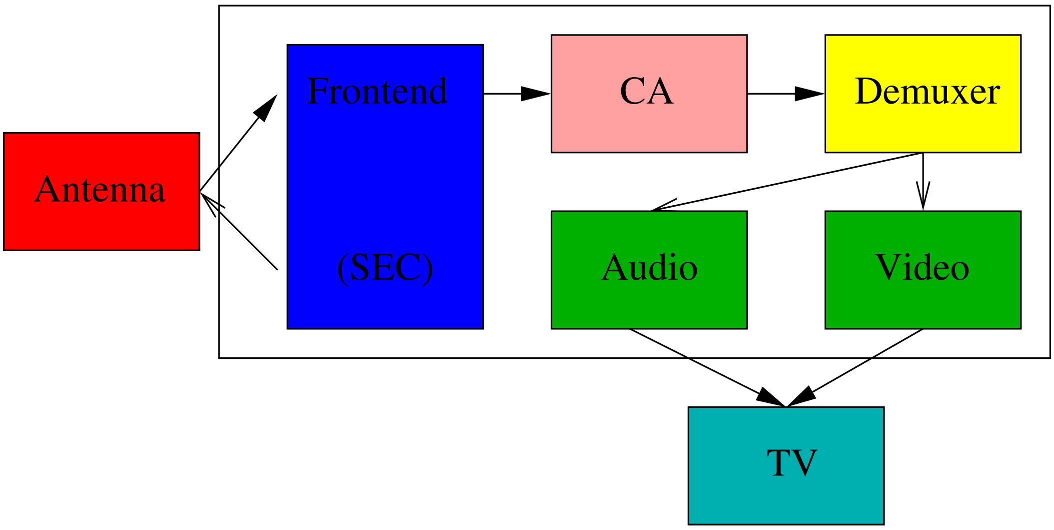

Figure 3: The API structure

- Frontend consisting of tuner and DVB demodulator

  Here the raw signal reaches the DVB hardware from a satellite dish or antenna or directly from cable. The frontend down-converts and demodulates this signal into an MPEG transport stream (TS). For satellite reception the hardware can also send signals back through the satellite cable to control the polarization of the LNB, to switch between different LNBs or even to control the movements of a dish rotor.

- Conditional Access (CA) hardware like CI adapters and smartcard slots

  The complete TS is passed through the CA hardware. Programs to which the user has access (controlled by the smart card) are decoded in real time and re-inserted into the TS.

- Demultiplexer which filters the incoming DVB stream

The demultiplexer splits the TS into its components like audio and video streams. Besides usually several of such audio and video streams it also contains data streams with information about the programs offered in this or other streams of the same provider.

- MPEG2 audio and video decoder

  The main targets of the demultiplexer are the MPEG2 audio and video decoders. After decoding the pass on the uncompressed audio and video to the computer screen or (through a PAL/NTSC encoder) to a TV set.

Figure 3 shows a crude schematic of the control and data flow between those components. On a DVB PCI card not all of these have to be present since some functionality can be provided by the main CPU of the PC (e.g. MPEG picture and sound decoding) or is not needed (e.g. for data-only uses like "Internet over satellite"). Also not every card or STB provides conditional access hardware.

The typical structure of a PCI card includes a chip that serves as PCI-bridge, I^2C master and provider of other GPIO ports for additional hardware. Connected to this master chip we have the frontend which is usually accessed via the I^2C-bus and is prone to replacement by better or probably cheaper hardware in later revisions of the card. Additionally, we find connections to CI controllers and MPEG decoders which my through various kind of IO systems. Figure 4 shows the structure of the DVB-S card design by Fujitsu-Siemens.

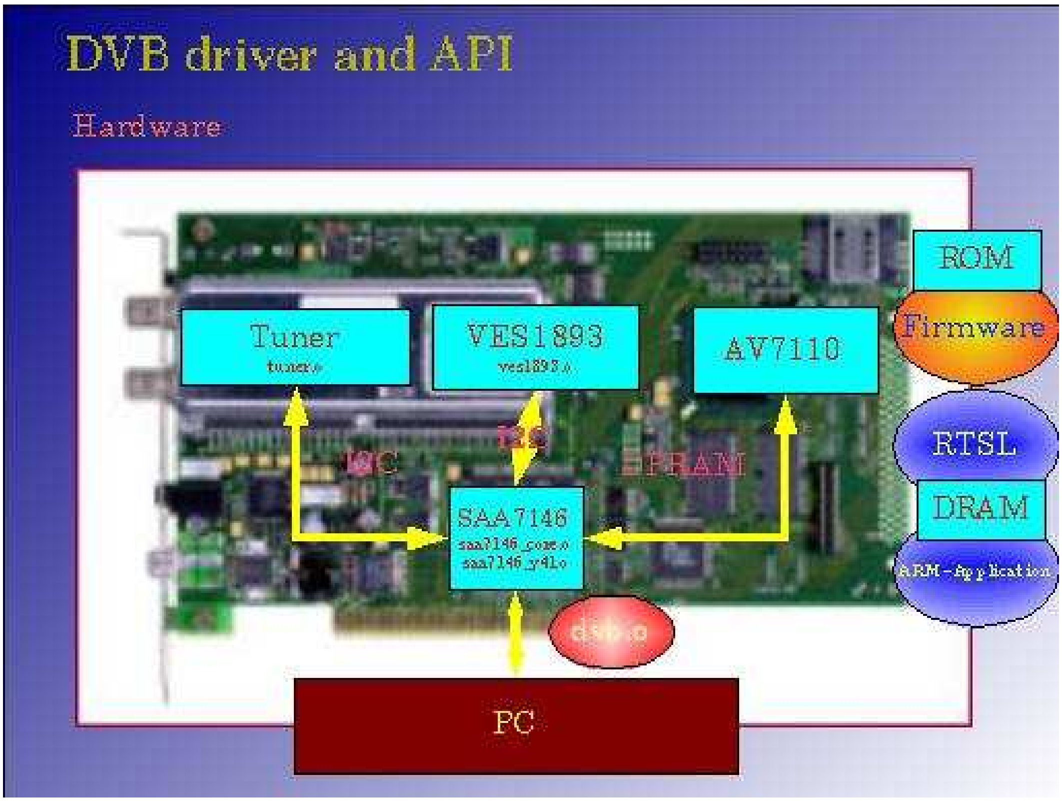

Figure 4: DVB-S card by Fujitsu-Siemens

Here we have the SAA7146 which is the central chip that provides the PCI bridge, the I^2C-Master for the Tuner and demodulator (VES1893) and a DEBI port to the AV711x chip, which is the MPEG decoder and handles the CI and demuxing.

MPEG transport streams (TS)

The previous sections have given us some understanding of the transmission methods and the hardware used for DVB. This helps us to get a grasp of the tuning process and the demultiplexing process controlled by the API. Now we have to see what it is that we are demultiplexing.

As we have already learned the transport stream consist of packets of 188 bytes size, which have 184 bytes of payload. The majority of the packets in a transport stream will carry the PES (see figure 5 and [2]) of audio and video MPEG data. The further packeting of the audio and video elementary streams has among others the purpose to provide time stamps for the synchronization of the audio and video streams. Furthermore, the AV packets carry the program clock reference (PCR) data in so-called adaptation fields that can be added to the TS packets and may also contain stuffing bytes in case a PES packet ends before the 184 bytes are filled.

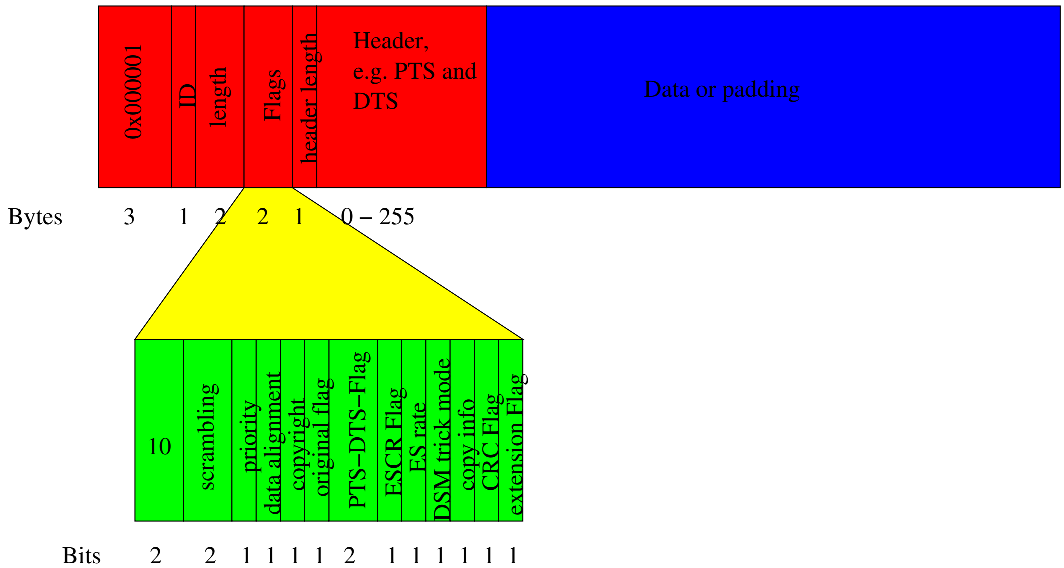

Figure 5: PES header

Another information that comes in the shape of a PES is the teletext data that is reinserted into the TV picture so that the teletext decoder of your TV set can pick it up (see [4]).

The additional programming information in the TS comes in the shape of so-called sections which have a maximal length of 4096 bytes, but except for private sections are only 1024 bytes long. They have a common header consisting of 3 bytes mainly containing ID and length of the section.

Sections most importantly carry data containing PSI. This information is carried in three standard tables which can be split into several sections. In figure 6 we can see the description of those three tables which are:

- Program Association Table (PAT)

  Associates the Program Number and the PMT.

- Program Map Table (PMT)

  Specifies the PID values for components of one or more programs.

- Conditional Access Table (CAT)

  Associates one or more EMM streams with a unique PID.

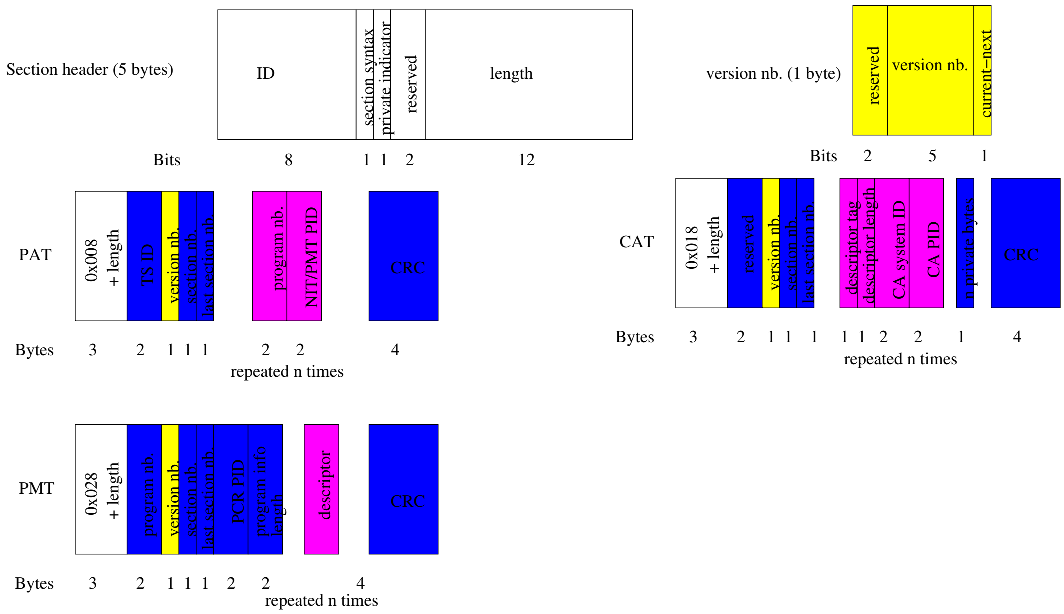

Figure 6: PSI data sections

In addition to these tables, which are not DVB specific and are already described in the MPEG2 documentation [2], the DVB standard adds tables containing service information (DVB-SI) among other things. This information allows a receiver to configure itself and provides the user with an electronic program guide. The four basic tables of DVB-SI are [3]:

- Network Information Table (NIT)

  Carries the physical network parameters, such as frequencies, transponder numbers, etc.

- Service Description Table (SDT)

  lists the names and other parameters of the services in the TS.

- Event Information Table (EIT)

  informs about current events contained in the TS, e.g. name and description of current program.

- Time and date Table (TDT)

  is used to update the internal RTC of the receiver.

Those tables all conform to the private data section specification in the MPEG2 documents. Also defined by the DVB standard, we have the section format for the transmission of IP and other data [5]. For modern receivers there are other services transmitted via those sections like MHP data.

It is obvious that the possibility to multiplex AV streams with additional programming information as well as other forms of data is very convenient and provides a versatile way to enhance the television program as well as the opportunity for new multimedia applications which may lead away from the traditional TV usage. In the following sections we will see how the Linux DVB API can be used to access all this data providing an interface for the application programmer.

The DVB API

The DVB API maps each of the various functional blocks of a set-top-box as they were discussed above onto its own device. The most minimal cards (so-called low-budget card) will a least have a *frontend* device and a *demux* device. Cards with hardware decoders and video capabilities also have an *audio* and *video* and an *osd* device for on screen graphics.

Furthermore, there are the *dvr* device for recording and playing back sub-streams of the TS and network devices (with the name *dvbX_Y* for the Yth network device on card X) to support networking over satellite. These devices and the *demux* device are actually only frontends to an internal kernel demux API which has to be implemented by each driver. They internally just set filters in the kernel demux device and use them for their special purposes. The user can open the *demux* as many times as there are hardware or software filters provided by the driver. Each file handle then controls one of the filters. In the same way, the user can assign network devices to each PID he wants to until all filters in the kernel demux device are exhausted.

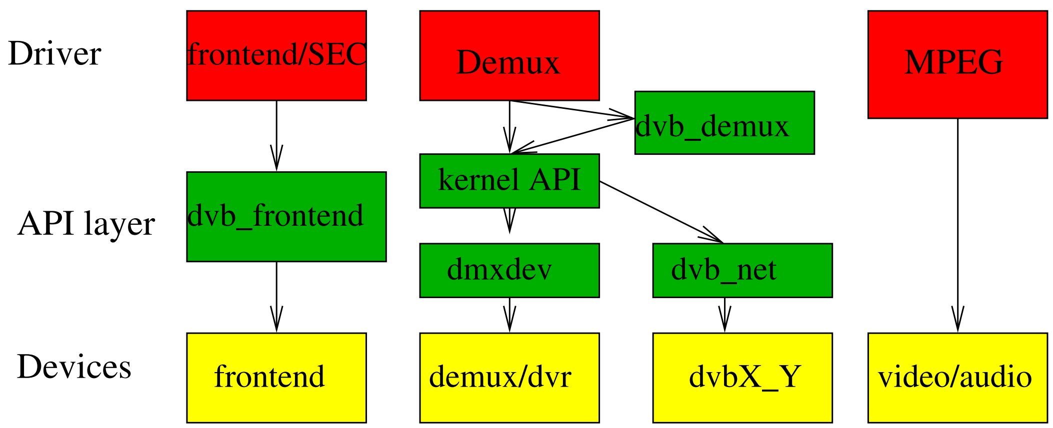

Figure 7: DVB API structure

The original Nokia API put the DVB devices in the directory `/dev/ost/` (OST = Open Standards Terminal) and did not number them. It was only designed to support one DVB card. We later added support for more than one card in the same system so that e.g. `/dev/ost/demux1` would correspond to the demultiplexer of the second card. The Nokia API also had a separate *sec* device for controlling special equipment like DiSEqC switches or rotors. This is now integrated into the *frontend* device because this hardware is usually controlled via the frontend.

Newer hardware even offers several frontends, demuxers and decoders on a single chip. To accommodate such hardware the devices are now accessible via directories structured like this

```
/dev/dvb/adapter0/frontend0
                  demux0
                  demux1
                  video0
                  audio0
                  ...

/dev/dvb/adapter1/frontend0
                  frontend1
                  demux0
                  demux1
                  ...
```

where each `/dev/dvb/adapter*` directory contains all the devices belonging to the same DVB card. If you are using *devfs* these directories will automatically only be filled with those devices which actually present on each card.

The available function calls for each device are documented in the Linux DVB API document [6] which can be found at `www.linuxtv.org`. We will not go into any details here but tuning to a specific satellite channel would involve the following steps:

- open the frontend device
  - send the necessary DiSEqC commands for your satellite setup
  - set the frequency, symbol rate, etc. of the transponder on which the channel is transmitted
  - wait for an event returned from the frontend device which indicates if the tuning process was successful or not
- open the demux device twice
  - set one filter to type PES_AUDIO, the other to type PES_VIDEO
  - select the right video and audio PIDs and for both filters select the decoder as output
- open the audio and video devices and start decoding

As you can see, the tuning process is a little more involved than in the analog case where you only have to select the frequency. A complete set-top-box application also would have to do much more. It would have to handle channel lists, update channel entries and their PIDs according to the data obtained in PAT/PMT scans on each transponder, extract and process teletext data, program guides and MHP applets, etc.

If only a low-budget card is present there will of course be no decoder hardware and thus no *video* and *audio* devices. In this case, the application will have to read the MPEG2 data from the demux and send it to a software decoder (e.g. xine or mplayer) or another MPEG2 decoder card.

Some DVB hardware also offers DMA or overlay of the decoded picture into/onto the normal display of the graphics card. Such features of the hardware are not covered by the DVB API and should be programmed and registered as a Video4Linux or Video4Linux2 device.

Building a new driver

Apart from the DVB API definition there also is complete framework of hardware independent support modules in place which makes writing a new DVB device driver easier. The sources of these modules can be found in the DVB distribution in the directory `DVB/driver/`. If you prefer patching the kernel tree (shell scripts for doing this from inside the DVB package are provided) you will find them in `linux/driver/media/dvb/`. The actual drivers for DVB cards and frontends are located in the sub-directories of this `driver` directory.

The core of the DVB architecture is the `dvbdev` module. It handles the registration of the DVB hardware much like the videodev module does in the case of Video4Linux. Through this module each DVB card has to register itself as a new adapter. This automatically creates a new `/dev/dvb/adapterX` entry in *devfs* (or just supports the corresponding major/minor device if you do not use *devfs*). The driver then can register new devices as children of this adapter according to its capabilities.

Frontends are usually handled through I^2C interfaces. There already are functional drivers for several popular frontend types. They can be found in the `frontends` subdirectory of the `driver` directory. They all register themselves to the `dvb_i2c` module. When a DVB card registers a driver for its I^2C bus to the same module, all frontend drivers scan this I^2C bus and try to detect if the supported hardware is present. This is similar to the mechanism already present in the I^2C kernel driver. Because some frontend hardware needs special treatment which is not supported by the kernel driver we are currently using our own module for this.

Each frontend driver has to support a special set of commands which is needed for DVB tuning. The `dvb_frontend` module controls the complete tuning process through this command interface. It also automatically takes care of frequency drifts, re-tuning after loss of sync and other special events.

Similarly, each DVB driver only has to implement the kernel demux API, which we talked about in the last section. The modules `dmxdev` and `dvb_net` implement the *demux*, *dvr* and DVB network devices on top of this.

Low-budget cards usually do not have any hardware filtering. They only can provide the raw transport stream. For such cards the `dvb_demux` module is provided. It contains a complete DVB demultiplexer with PID, PES and section filters implemented in software. It also contains an implementation of the kernel demux API. The driver only has to register itself with the `dvb_demux` module and deliver the transport stream or, if it can, a suitable sub-stream with only the relevant PIDs to it.

The actual drivers for DVB cards and frontends can be found in sub-directories of the `driver` directory.

Next Generation DVB Platforms

The next generation Linux DVB platforms come in various sizes and shapes. Some manufacturers are moving to general purpose multimedia PCs with selectable add-on features. The basis is usually a standard low end PC system, preferably those which do not need any fans, with all-included (IDE, video, TV-out, USB, firewire, etc.) motherboards. To look less like a PC they come in nice black desktop PC size case. PCI cards and DVD players/burners are added according to the wishes of the customer.

The advantages of such a system are that you practically get a complete PC and can do everything as on any other PC on them. But they usually are slow compared to state of the art PCs and e.g. not well suited for the latest 3D games. Sometimes the manufacturer also offers high-end models with state-of-the-art hardware built in. But often it turns that it is better to just build your own multimedia PC, especially if free Open-Source solutions (e.g. Linux + VDR [7]) exist.

In between there are some systems which use special low power i86 CPUs (Geode, VIA) which already have some features like PCI bridges, IDE ports and video included. Frontends and MPEG2 decoders are usually part of the motherboard and no PCI slots are provided. These systems are much smaller and more attractive to end customers than standard PCs but often still too expensive to really make a breakthrough in the consumer market.

The traditional STB companies mainly keep relying on embedded systems with special purpose hardware and CPUs integrated on the same chip. Older systems used very low performance CPUs (e.g. an ARM7TDMI at 40MHz) which were only good enough to power so-called "zapping-boxes" which could let you zap through a channel list but not much more. The new chips have to support much hungrier applications like OpenTV, MHP and Internet browsers. They now have processor cores like the ARM9, MIPS cores and PPCs. The clock speeds range from about 100 to 300 MHz.

Although these CPU cores are quite slow compared to high-end CPUs, it is enough to let a Linux system run on these boxes which can provide an Internet browser and mail services, serve as an ADSL or ISDN router and run anything else you would want on a Linux system (besides Quake). At the same time, without using much CPU time (thanks to the special hardware) they also display digital television. Just remember that your PC was not much faster just a few years ago. Some people still use much slower PCs for the same purposes. Thanks to mass production, these new set-top-boxes also will be very cheap.

The DVB API is already prepared for some of the capabilities the new platforms offer. But to fully accommodate the new features and also some shortcomings of these platforms, the API will have to become even more flexible.

Most new platforms now support between 2 and 4 satellite frontends plus various other feeds from firewire, hard-disk and system memory. The data from these frontends can be processed by up to 3 demuxers, be descrambled, decoded and displayed and sent back to hard-disk, firewire or back to memory at the same time. Some hardware even contains 2 complete MPEG decoders on one chip. Just for selecting all the data paths, we will have to come up with new API calls.

The greater challenge is to change the demultiplexer API to allow all this with minimal copying by the CPU. Right now, once data reaches the PCs memory, it is copied through the kernel API layer to buffers in the device layer and then again to user space. By efficiently allocating and re-using DMA buffers and only passing along pointers to these buffers, this copying, at least inside the kernel, can be reduced to a minimum. To reduce the copying into user space we will also offer memory mapping of the kernel buffers.

If you have DVB hardware with all the special devices on the same chip it will also often offer special dedicated DMA channels to move data between the devices. In this case, it is easy to further reduce the CPU load. For systems which e.g. use a low-budget DVB card together with an MPEG decoder card we will have to come up with other solutions which would let different PCI card drivers share DMA buffer memory.

References

[1] H. Benoit 1997, *Digital Television*, London: Arnold.

[2] ISO 13818-1 (ISO/IEC), *Coding of moving pictures and associated audio.*

[3] ETSI EN 300 468, *Digital Video Broadcasting; Specification for Service Information (SI) in DVB systems.*

[4] ETSI EN 300 472, *Digital Video Broadcasting; Specification for conveying ITU-R System B Teletext in DVB bitstreams.*

[5] ETSI EN 300 192, *Digital Video Broadcasting; Specification for data broadcasting.*

[6] M. and R. Metzler, *LINUX DVB API*, convergence GmbH, http://www.linuxtv.org.

[7] Klaus Schmidinger, *Video Disk Recorder*, http://www.cadsoft.de/people/kls/vdr/index.htm