

# Eagle video transmitter software description

Bertold Van den Bergh                      Yuri Murillo Mange  
vandenbergh@bertold.org            yuri.murillo@esat.kuleuven.be

October 24, 2016

## 1 Introduction

In this document we will describe the higher layers of the video transmission system. While the VHDL code you will write implements the physical layer (PHY) more is needed to actually make a practical Digital TV system. Several choices have to be made on how the video bitstream will be framed and transmitted. In Section 2 a non-exhaustive list of different options is presented. To simplify the implementation of the system some software has been developed by the ESAT TELEMIC Networked Systems Group. An overview of the provided tools is given in Section 3. Finally, the signal from the drone needs to be received at the ground station. In Section 4 it is explained how to do this.

## 2 Digital TV implementation choices

While initial Digital TV systems were relatively simple, over the years many methods have been developed for delivering multimedia services. In this project we will look at standardized protocols<sup>1</sup> that can be run over a DVB-X2 system.

### 2.1 BBFRAME mode

The first design choice is what BBFRAME mode to use. DVB supports the following approaches:

---

<sup>1</sup>Some service operators use proprietary transport protocols. In principle you are allowed to do this too, but it means that almost no existing software can be reused. You would have to develop the full chain.

- MPEG Transport Stream: This is the oldest and most-compatible mode. You deliver 188byte packets to the baseband framer, which converts it into a stream of BBFRAMES. This mode is normally used for broadcast television.
- Continuous Generic Stream: You provide the framer opaque data without any information about the structure. The framer does not care about any packetization and simply passes the bitstream as-is.
- Packetized Generic Stream: This is a Generic Stream with a fixed packet length (up to 64Kb). As far as we know this type of service is rarely used in practice. Note that a stream with varying-length packetization is handled like a continuous stream!

In Table 1 and overview is given of the different modes together with some advantages and disadvantages.

|                           | Pro  | Con   |
|---------------------------|--|---|
| MPEG-TS                   | <ul style="list-style-type: none"> <li>• Widely used for video broadcasting.</li> <li>• Very good support by software and hardware.</li> <li>• Easy to use and understand.</li> <li>• BBFRAME decoder for this format is always provided → simplifies software development.</li> </ul> | <ul style="list-style-type: none"> <li>• Slightly less efficient.</li> <li>• Not flexible: fixed format.</li> </ul>   |
| Packetized Generic Stream | <ul style="list-style-type: none"> <li>• Slightly more flexible and efficient than MPEG-TS.</li> </ul>   | <ul style="list-style-type: none"> <li>• Rarely used (if not counting MPEG-TS which is actually a special case of this mode)</li> <li>• Poor software support: BBFRAME decoder (usually) has to be programmed by the user.</li> <li>• Not all demodulators support this.</li> </ul> |

|                                 |   |  |
|---------------------------------|---|--|
| Continuous<br>Generic<br>Stream | <ul style="list-style-type: none"> <li>• Most efficient.</li> <li>• Maximum flexibility.</li> <li>• Supports variable length frames.</li> </ul> | <ul style="list-style-type: none"> <li>• Difficult to use.</li> <li>• Poor software support: BBFRAME decoder (usually) has to be programmed by the user.</li> <li>• Only very few professional demodulators support this.</li> </ul> |
|---------------------------------|---|--|

Table 1: Pros and cons of different DVB-X2 BBFRAME modes

## 2.2 Data encapsulation

After selecting the BBFRAME mode it should be decided how to actually put the data into the chosen framing format. The main decision here is whether the video data will be inserted directly, or an Internet Protocol (IP) layer will be added in between. The main advantage of using an IP based system is that it can easily be extended to carry other types of data. Since the system will then work like an unidirectional network cable existing network applications can easily be used. The main disadvantage is that the IP encapsulation adds a small amount of overhead. Consumer broadcast systems almost always use direct video transmission, while many professional applications use an IP intermediate layer.

### 2.2.1 Direct video

Directly carrying video is the whole point of an MPEG transport stream. We refer you to the MPEG-TS standard, which is available on Toledo, for information on how to handle the multiplexing. You may also want to lookup the format of a H264 PES (Packetized Elementary Stream). Open source tools are available for generating the MPEG-TS multiplex: `gststreamer`<sup>2</sup> and `ffmpeg`<sup>3</sup>. Depending on the receiver you may need to insert DVB-SI signalling information into your transport stream. This standard is available on Toledo.

Directly encapsulating video in Generic Streams is almost never done since it is very inflexible. When it is done the bitstream from the video en-

<sup>2</sup><https://gststreamer.freedesktop.org/>

<sup>3</sup><https://www.ffmpeg.org/documentation.html>

coder is usually directly fed to the BBFRAME framer in Continuous Generic Stream mode.

### 2.2.2 IP based

Two standard protocols are available for creating an IP network over a MPEG Transport Stream:

- Unidirectional Lightweight Encapsulation (ULE): This is a simple protocol that supports both IP and Ethernet encapsulation. It is standardized by IETF <sup>4</sup>. A software to generate ULE frames is available, see Section 3.6.
- Multiprotocol Encapsulation (MPE): This is a more complex protocol that does almost the same thing as ULE. It is standardized by ETSI <sup>5</sup>.

IP packets are also often encapsulated in a Continuous Generic Stream with variable length packets. In this case the IP packet is usually directly inserted into the Generic Stream packets.

## 2.3 'Reference' System

A test version of this system has been developed when preparing the P&D. This system supported two modes. Firstly, direct video in MPEG-TS was tested. Then, an IP based system was built using ULE in MPEG-TS. The video was sent over this IP network as UDP/RTP packets containing H264 data. You are not at all required to follow this structure, but you may not be able to get as much help from your TAs if you don't. When using a different mode, please verify that it is supported by the used receiver. See Section 4 for details.

## 3 Provided software

In this section we describe the software packages provided to help you implement the video transmission system. This text only covers traditional software, the VHDL parts are described in a different document. The instructions provided in this document are intended to aid you in quickly solving some design issues that may come up. As always, all roads lead to Rome:

---

<sup>4</sup><https://tools.ietf.org/html/rfc4326>

<sup>5</sup>[http://www.etsi.org/deliver/etsi\\_en/301100\\_301199/301192/01.04.01\\_40/en\\_301192v010401o.pdf](http://www.etsi.org/deliver/etsi_en/301100_301199/301192/01.04.01_40/en_301192v010401o.pdf)

depending on how you design the rest of the system it is likely that some information will not be useful to you. This is normal and does not mean your 'design choices' are inferior! The following software is provided:

- `usb_firmware`: This is the software that runs in the CY7C68013A (FX2LP) microcontroller on the transmitter PCB. It handles the USB connection to the host and maintains the sample FIFO between the Zybo and the transmit DACs. Low level control functions of the clock generators, LO, ... are handled here. This internal workings of this program are described in Section 3.2.
- `eagletx`: This is a module installed in the Linux kernel running on the Zybo ARM core. It connects to the `usb_firmware` and exposes a Video4Linux2 (V4L2) interface to userspace that allows you to transmit a signal. This module also handles the high-level control of the front-end: changing the frequency, setting the transmit power, ... This internal workings of this program are described in Section 3.3.
- `EagleTXController`: This is a userspace program that talks to the front-end through the `eagletx` kernel module. This is the program you will usually interact with when using the transmitter. An explanation of how to use it is given in Section 3.4.
- `ns_nightshade`: This kernel module is used for controlling the DVB-X2 digital baseband in the FPGA. The module provides a Video4Linux2 interface that allows you to upload MPEG2-TS frames to the modulator. V4L2 controls are provided for changing settings. This module needs to be paired with a DMA controller that does the actual data transfer handling. See section 3.5 for more information.
- `ULEFramer`: This userspace program allows you to create a virtual Ethernet or IP interface using the Unidirectional Lightweight Encapsulation protocol. This protocol is often used in satellite internet deployments. It is also used for Digital IPTV over DVB. The signal generated by this tool can be decoded using `dvbnet` on Linux. See Section 3.6 for detailed information.
- `EagleTXCalibrate`: This is a program that you run on a standard PC to calibrate the transmitter. This calibration is required since (affordable) real-life RF components have tolerances. This program will transmit a signal through the front-end and receive it back using a RTL-SDR dongle. The transmitter impairments will then be calibrated out. This

calibration has already been done for you and is stored in the transmitter EEPROM. You only need to rerun the calibration if you want to change the operating band or the stored calibration coefficients have somehow been corrupted. See Section 3.7 for the calibration procedure.

### 3.1 Where to get it?

You can get the software from the redmine GIT repository server. It is in the repository 'software'.

### 3.2 usb\_firmware

The firmware handles the interface between the actual transmitter hardware and the USB host. It runs on a FX2LP microcontroller. Normally you do not interact directly with this firmware, it is handled by the eagletx kernel module which offers a high level interface. This firmware is already installed on your Zybo and front-end board. Usually you do not need to change it, but if you do the instructions are provided.

The USB interface exported to the host has the following options:

#### Device Descriptor:

|                    |                                     |
|--------------------|-------------------------------------|
| bLength            | 18                                  |
| bDescriptorType    | 1                                   |
| bcdUSB             | 2.00                                |
| bDeviceClass       | 255 Vendor Specific Class           |
| bDeviceSubClass    | 0                                   |
| bDeviceProtocol    | 0                                   |
| bMaxPacketSize0    | 64                                  |
| idVendor           | 0xcafe                              |
| idProduct          | 0xbeef                              |
| bcdDevice          | 0.01                                |
| iManufacturer      | 1 ESAT Networked Systems Group      |
| iProduct           | 2 EagleTX (vandenbergh@bertold.org) |
| iSerial            | 0                                   |
| bNumConfigurations | 1                                   |

#### Configuration Descriptor:

|                     |    |
|---------------------|----|
| bLength             | 9  |
| bDescriptorType     | 2  |
| wTotalLength        | 34 |
| bNumInterfaces      | 1  |
| bConfigurationValue | 1  |

```

iConfiguration          0
bmAttributes             0xc0
    Self Powered
MaxPower                 10mA
Interface Descriptor:
    bLength              9
    bDescriptorType       4
    bInterfaceNumber      0
    bAlternateSetting     0
    bNumEndpoints         0
    bInterfaceClass       255 Vendor Specific Class
    bInterfaceSubClass    0
    bInterfaceProtocol    0
    iInterface            0
Interface Descriptor:
    bLength              9
    bDescriptorType       4
    bInterfaceNumber      0
    bAlternateSetting     1
    bNumEndpoints         1
    bInterfaceClass       255 Vendor Specific Class
    bInterfaceSubClass    0
    bInterfaceProtocol    0
    iInterface            0
Endpoint Descriptor:
    bLength              7
    bDescriptorType       5
    bEndpointAddress      0x02 EP 2 OUT
    bmAttributes          2
        Transfer Type      Bulk
        Synch Type         None
        Usage Type         Data
    wMaxPacketSize        0x0200 1x 512 bytes
    bInterval             0

```

We see the device has two interface configurations that can be selected. One without any data endpoints and another one with a bulk data endpoint (EP2). Of course, the EP0 control endpoint is always present. Selecting the interface with the bulk endpoint will select the streaming mode. Data that is written to EP2 will be transferred to the DAC FIFO. The control endpoint is used for controlling the board. The following commands are implemented:

- 0xb1: Perform I2C transfer
- 0xb2: Set pins in non-streaming mode
- 0xb3: Configure streaming IFMODE
- 0xb4: Set state of IO buffers
- 0xb5: Perform SPI transaction
- 0xb6: Read status
- 0xb7: Read board info
- 0xb8: Set frequency
- 0xb9: Set sample rate
- 0xba: Get ADC values
- 0xbb: Set output power
- 0xbc: Disable transmitter

For details on how to use the commands, please read the source code.

### **3.2.1 eeprom\_config**

A second very simple firmware 'eeprom\_config' is also provided. This is used to initially program the EEPROM on a new device. If you damage the EEPROM contents you may find that the board is no longer detected correctly by the Zybo. You can reset the EEPROM by running this program with `cycfx2prog` as described in Section 3.2.3. Since the configuration EEPROM is damaged, you will need to program without the `id` parameter. When this is done, you will likely need to recalibrate as well. Be very careful, if you flash an invalid FX2LP configuration descriptor it is possible that the board will no longer connect over USB at all. This makes it impossible to perform further programming attempts. Contact the DVB-X2 technical expert if this happened.



```

bertold@bertold-desktop ~/DVB-T2/EagleTX/usb_firmware $ make
make -C lib
make[1]: Entering directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/lib'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/lib'
make -C projects
make[1]: Entering directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects'
make -C sdr
make[2]: Entering directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects/sdr'
for a in dscr.a51; do \
    cp $a build/; \
    cd build && sdas8051 -logs `basename $a` && cd ..; done
for s in sdr.c; do \
    THISREL=$(basename `echo "$s" | sed -e 's/\.c$/\.rel/'`); \
    sdcc -mmcs51 --code-size 0x3c00 --xram-size 0x0200 --xram-loc 0x3c00 -Wl"-b DSCR_AREA=0x3e00" -Wl"-b INT2JT=0x3f00" -c -I ../../include -I "" $s -o build/$THISREL ; done
sdcc -mmcs51 --code-size 0x3c00 --xram-size 0x0200 --xram-loc 0x3c00 -Wl"-b DSCR_AREA=0x3e00" -Wl"-b INT2JT=0x3f00" -o build/sdr.ihx build/sdr.rel build/dscr.rel fx2.lib -L ../../lib
make[2]: Leaving directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects/sdr'
make -C eeprom_config
make[2]: Entering directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects/eeprom_config'
for a in dscr.a51; do \
    cp $a build/; \
    cd build && sdas8051 -logs `basename $a` && cd ..; done
for s in eeprom.c; do \
    THISREL=$(basename `echo "$s" | sed -e 's/\.c$/\.rel/'`); \
    sdcc -mmcs51 --code-size 0x3c00 --xram-size 0x0200 --xram-loc 0x3c00 -Wl"-b DSCR_AREA=0x3e00" -Wl"-b INT2JT=0x3f00" -c -I ../../include -I "" $s -o build/$THISREL ; done
sdcc -mmcs51 --code-size 0x3c00 --xram-size 0x0200 --xram-loc 0x3c00 -Wl"-b DSCR_AREA=0x3e00" -Wl"-b INT2JT=0x3f00" -o build/eeprom.ihx build/eeprom.rel build/dscr.rel fx2.lib -L ../../lib
make[2]: Leaving directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects/eeprom_config'
make[1]: Leaving directory `/home/bertold/DVB-T2/EagleTX/usb_firmware/projects'
bertold@bertold-desktop ~/DVB-T2/EagleTX/usb_firmware $

```

Figure 1: Compiling the transmitter firmware

### 3.2.2 Compiling the code

To compile the firmware you need to use the sdcc compiler, make sure this is in your path before continuing. Go to the usb\_firmware directory in your checked out students repository. Type make. See Figure 1 for the expected output.

You will find the compiled executables in the following paths:

- usb\_firmware/projects/sdr/build/sdr.ihx
- usb\_firmware/projects/eeprom\_config/build/eeprom.ihx

### 3.2.3 Running the compiled code

There are two ways to run the code. The first method is used when developing and loads the software directly to the memory in the processor. Use the command cycfx2prog:

```
cycfx2prog -id=cafe.beef reset prg:filename.ihx run
```

The specified USB vendor and product ID (0xCAFE, 0xBEEF) are needed if the board is configured. The FX2LP is used in many USB peripherals. It is

common in Digital TV tuners and webcams. The `-id` parameter makes sure you program the Eagle transmitter and not something else. If the FX2LP boots unconfigured you must omit this:

```
cycfx2prog reset prg:filename.ihx run
```

The second way installs the firmware on the Zybo. This will cause it to be automatically loaded when the Zybo boots. The kernel module `eagletx` (Section 3.3) loads a firmware file stored at:

```
/lib/firmware/eagletx-datapump-cafe-beef.bin
```

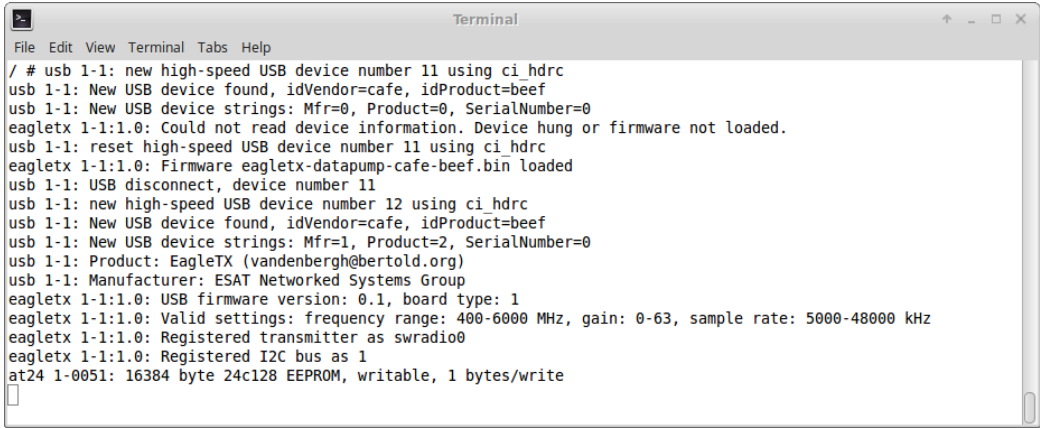
If you put your new firmware here it will automatically be loaded. The output file produced by the compilation is an `ihx`, while a binary file is needed. Thus, convert it using `objcopy`:

```
objcopy -I ihex projects/sdr/build/sdr.ihx -O binary eagletx-datapump-cafe-beef.bin
```

Finally, copy the `eagletx-datapump-cafe-beef.bin` to your SD card root directory. If no firmware is provided in the SD card the default firmware will be used.

### 3.3 eagletx

This is a kernel module that provides a high level interface to the `usb_firmware` and the transmitter module. When you connect the device to the Zybo with the `eagletx` kernel module loaded you should see the firmware being uploaded and afterwards a Video4Linux2 SDR and I2C interface will be created. See Figure 2 for expected output. After loading the red led on the transmitter



```
File Edit View Terminal Tabs Help
/ # usb 1-1: new high-speed USB device number 11 using ci_hsrc
usb 1-1: New USB device found, idVendor=cafe, idProduct=beef
usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
eagletx 1-1:1.0: Could not read device information. Device hung or firmware not loaded.
usb 1-1: reset high-speed USB device number 11 using ci_hsrc
eagletx 1-1:1.0: Firmware eagletx-datapump-cafe-beef.bin loaded
usb 1-1: USB disconnect, device number 11
usb 1-1: new high-speed USB device number 12 using ci_hsrc
usb 1-1: New USB device found, idVendor=cafe, idProduct=beef
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: EagleTX (vandenbergh@bertold.org)
usb 1-1: Manufacturer: ESAT Networked Systems Group
eagletx 1-1:1.0: USB firmware version: 0.1, board type: 1
eagletx 1-1:1.0: Valid settings: frequency range: 400-6000 MHz, gain: 0-63, sample rate: 5000-48000 kHz
eagletx 1-1:1.0: Registered transmitter as swradio0
eagletx 1-1:1.0: Registered I2C bus as 1
at24 1-0051: 16384 byte 24c128 EEPROM, writable, 1 bytes/write
```

Figure 2: EagleTX device being initialized

will be lit.

The device provides a Video4Linux2 interface for changing settings on the fly. Three settings can be changed:

- Frequency: Central frequency, change with the command `v4l2-ctl -f freq -d /dev/swradio0 -tuner-index=1` where freq is the transmit frequency in MHz. The effect of changing the frequency is shown in Figure 3a.
- Sample rate: The DAC clock frequency. Cannot be changed when the stream is running. Change using `v4l2-ctl -f freq -d /dev/swradio0 -tuner-index=0` where freq is the sample frequency in MHz.
- RF Gain: This setting controls the output power. Set via `v4l2-ctl -c rf_gain=gain -d /dev/swradio0` where gain is a value between 0 and 63. The effect of changing the output power is shown in Figure 3b.

We recommend that you use EagleTXController to set these values for you, unless you want to adapt them while the system is running. The source code of this module is included in the software package in case you want to change it and to aid debugging.

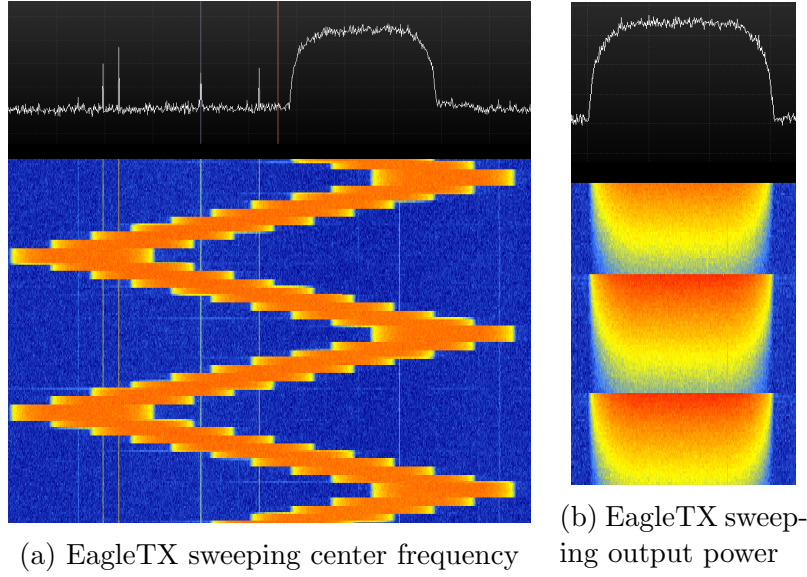
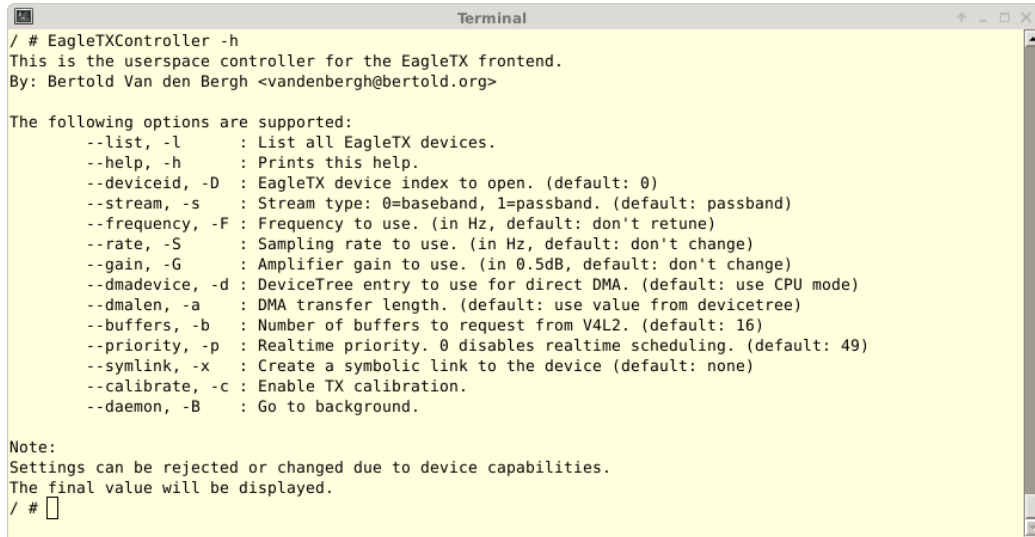


Figure 3: EagleTX dynamic configuration

### 3.4 EagleTXController

This program does the actual transmission. It will either generate the signal itself (on the CPU) or it will use a DMA technique to copy samples directly from the FPGA to the transmitter. The second option is recommended since it saves CPU resources. If you only use DMA mode you do not need to edit

the program. If you want to use the CPU mode you will need to put your modulation code in the file `CPURadioHandler.cpp` in the `EagleTXController` eclipse project. To help you understand how to implement it, a simple `CPU-RadioHandler.cpp` code is provided that transmits a 1KHz AM modulated sine wave.



```

Terminal
/ # EagleTXController -h
This is the userspace controller for the EagleTX frontend.
By: Bertold Van den Bergh <vandenbergh@bertold.org>

The following options are supported:
--list, -l      : List all EagleTX devices.
--help, -h      : Prints this help.
--deviceid, -D  : EagleTX device index to open. (default: 0)
--stream, -s    : Stream type: 0=baseband, 1=passband. (default: passband)
--frequency, -F : Frequency to use. (in Hz, default: don't retune)
--rate, -S      : Sampling rate to use. (in Hz, default: don't change)
--gain, -G      : Amplifier gain to use. (in 0.5dB, default: don't change)
--dmadevice, -d : DeviceTree entry to use for direct DMA. (default: use CPU mode)
--dmlen, -a     : DMA transfer length. (default: use value from devicetree)
--buffers, -b   : Number of buffers to request from V4L2. (default: 16)
--priority, -p  : Realtime priority. 0 disables realtime scheduling. (default: 49)
--symlink, -x   : Create a symbolic link to the device (default: none)
--calibrate, -c : Enable TX calibration.
--daemon, -B    : Go to background.

Note:
Settings can be rejected or changed due to device capabilities.
The final value will be displayed.
/ #

```

Figure 4: Supported configuration options of EagleTXController

The configuration options of the `EagleTXController` commands are shown in Figure 4:

- `list`: Shows all connected Eagle Transmitter Modules. Usually there is just one.
- `deviceid`: Selects which device to open. The IDs are obtained using the `list` command. If you have only one, do not specify this.
- `stream`: Whether to use a baseband (scalar) or passband (complex) stream.
- `frequency`: Transmission frequency.
- `rate`: Sample rate. For DVB-S2: 13000000, for DVB-T2: 11657143.
- `gain`: Transmit power. From 0 to 63, unit is 0.5dB.
- `dmadevice`: Which devicetree block to use as DMA source. If unspecified, uses CPU mode. Using the provided devicetree, the name is `dvb_output_buffer`. To use this, your devicetree needs an entry like this:

```

dvb_output_buffer {
    compatible = "networkedsystems,eagletx-dma";
    reg = <0x43C00000 0x10000>;
};

```

Replace 0x43C00000 with the address of the DVBOutputBuffer Vivado block.

- `dmalen`: Overrides the FPGA buffer length. You do not need this option, the correct value is obtained from the devicetree.
- `buffers`: Number of Video4Linux2 buffers. Default value is reasonable.
- `priority`: Enable realtime scheduling. Default value is reasonable.
- `symlink`: If you specify this option the program will create a symbolic link to the opened device. This useful if you want to access it later using `v4l2-ctl` or other tools. Typically `/dev/eagletx` is used.
- `calibrate`: Use calibration coefficients in EEPROM. The signal quality will be higher if you use this option.
- `background`: Go to background. This allows you to start the program while the Zybo continues processing other programs. Omit this when testing, set it when running for real.

See Figure 5 for a screenshot when the transmitter is running. You should see an almost constant number of buffers per second and on the left side a number that increases once every second. The green led on the transmitter board should be solidly lit. If the green and red leds are blinking this means that your Zybo is delivering data too slowly, you should reduce the sample rate or the CPU load. If you cannot achieve 13MHz sample rate without the red led flashing, the system cannot work properly.

**Be careful when specifying the frequency. If an antenna is connected you can cause significant interference to other services if you transmit on their frequency.** On the drone you should use a transmission frequency between 5735 and 5865MHz. Setting a gain of 63 will transmit just below the legal power limit, so you can safely do this. Of course, two groups cannot test on the same frequency. Discuss with the other groups who will use what frequency. You should leave at least 10MHz between each transmitter to avoid interference.

```

Terminal
File Edit View Terminal Tabs Help
/ # EagleTXController -F 868000000 -G 63 -S 13000000 -c
This is the userspace controller for the EagleTX frontend.
By: Bertold Van den Bergh <vandenbergh@bertold.org>

Trying to open device with settings:
  Device ID:      0
  Frequency:      868000000Hz
  Sample rate:    13000000Hz
  Gain:           31.5dB
  Buffers:        16
  Priority:        49
  Calibrate:      1
  Stream:         Passband

Requested format: SDR_CU08
Obtained format: SDR_CU08, buffersize: 16384, buffers: 16

Requested frequency: 868000000Hz
Obtained frequency: 868000000Hz

Requested sampling rate: 13000000Hz
Obtained sampling rate: 13000000Hz

Requested gain: 31.5dB
Obtained gain: 31.5dB

Calibrating: nullingI=0.425519, nullingQ=0.605617

Starting transmitter in CPU mode!

Enabling realtime scheduling.

4769 -> TX buffers per second: 1588

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB1

```

Figure 5: EagleTXController transmitting in CPU mode

### 3.5 ns\_nightshade

When your VHDL code is ready you can embed it in the FPGA on the Zybo. Blocks for this will be made available in due time. This kernel module is used for configuring these blocks and uploading the data from the CPU to the FPGA core.

It creates a Video4Linux2 interface that can be used to upload MPEG-TS packets. If you use one of the Generic Stream Modes this module will need to be significantly altered. Several Video4Linux2 controls are provided:

- `dvb_s2_modcod` (int): DVB-S2 modcod, between 1 and 28.
- `dvb_s2_rrc_rolloff` (int): Rolloff factor, 0=0.35, 1=0.25, 2=0.2
- `dvb_s2_baseband_interpolation` (int): Baseband interpolation ( $N_i$ ). This controls the signal bandwidth. The Symbol Rate is:

$$S_R = \frac{f_{DAC}}{3 \cdot N_i}$$

Figure 6 shows what happens if you change the interpolation factor on the fly.

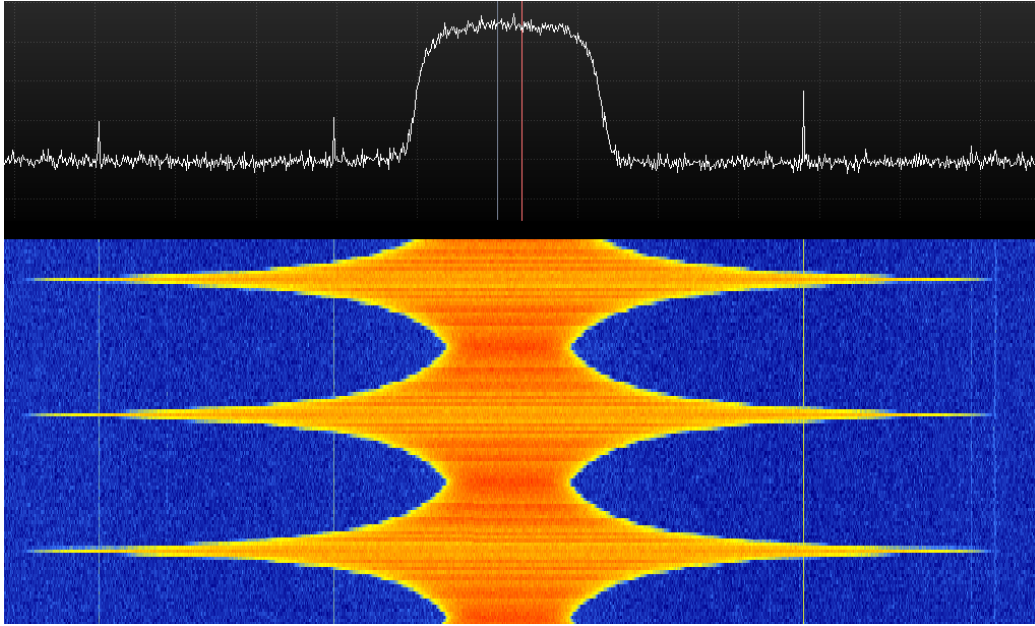


Figure 6: Changing the bandwidth dynamically

- `dvb_s2_baseband_gain` (int): Baseband gain. This is a soft control on the baseband power. Increase until the spectral mask is just not violated. Putting a very high value will increase the transmission power quite a bit, but the signal will be so distorted (clipped) that the receiver will not lock. Furthermore, you may cause adjacent channel interference to other groups. Always use a spectrum analyzer when increasing this.
- `add_mpeg_null_packets` (bool): Insert NULL (PID 0x1FFF) packets.
- `dvb_s2_dummy_pl_frames` (bool): Allow encoding dummy PLFRAMES.
- `dvb_s2_pilot_tones` (bool): Insert pilot tones. Turn this on when the source is moving or the signal is very weak.
- `dvb_s2_blocksize` (bool): 0=64800, 1=16200
- `lsb_of_null_pid` (int): Least significant byte of null PID. This is to work around a bug in a certain receiver (that you are not using). Leave as default.

In DVB-T2 mode you get different options. I recommend starting in S2 mode first since less can go wrong. You should add the module to your devicetree as follows:

```

nightshade_dma: axidma@40400000 {
    compatible = "xlnx,axi-dma-1.00.a";
    reg = <0x40400000 0x10000>;
    #dma-cells = <0x1>;
    xlnx,include-sg;
    dma_channel_0@40400000 {
        dma-channels = <1>;
        compatible = "xlnx,axi-dma-mm2s-channel";
        interrupts = <1 45 4>;
        interrupt-parent = <&intc>;
        xlnx,datawidth = <0x8>;
        xlnx,device-id = <0x0>;
    };
};

dvb_nightshade: ns_nigthshade@44000000 {
    compatible = "networkedsystems,nightshade-1.0";
    reg = <0x44000000 0x10000>;
    use-driver-queue;
    mpeg-buffer-size = <1880>;
    dmas = <&nightshade_dma 0>;
    dma-names = "mpeg_data";
};

```

Of course, you need to change the settings to match your Vivado design.

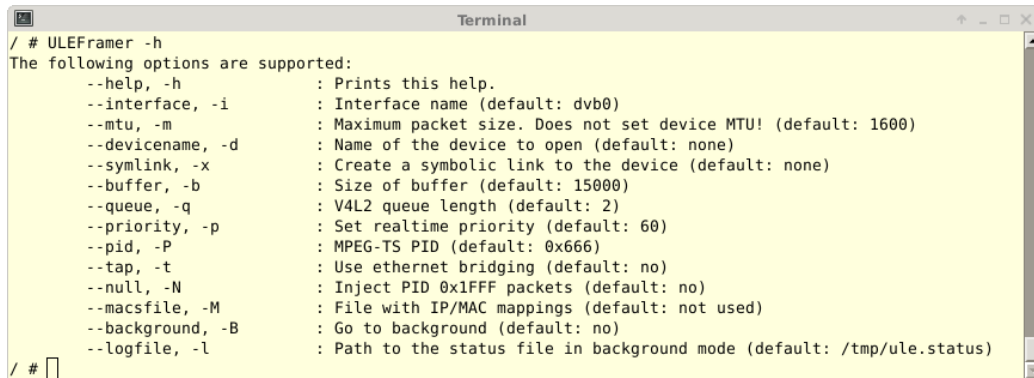
### 3.6 ULEFramer

This program allows you to create a virtual network card on the Zybo. Packets received via this interface will be encapsulated using the ULE protocol and delivered to the modulator for transmission. There are several settings you can configure. Run the ULEFramer utility with the `-h` parameter to get information on how to use it. A screenshot of the output is shown in Figure 7.

Here we give an overview of the settings:

- **interface:** This is the name of the virtual network card. By default it will be `dvb0`, but you can specify anything you want. You will need to use this name in all commands that have to do with configuring the network.
- **mtu:** This sets the maximum packet size the program will transmit. It should always be equal to or larger than the device MTU. The default value is suitable and normally should not be changed.



A terminal window titled "Terminal" showing the help output of the ULEFramer program. The output lists various command-line options and their default values. The prompt is "/ #".

```
/ # ULEFramer -h
The following options are supported:
--help, -h           : Prints this help.
--interface, -i       : Interface name (default: dvb0)
--mtu, -m             : Maximum packet size. Does not set device MTU! (default: 1600)
--devicename, -d       : Name of the device to open (default: none)
--symlink, -x         : Create a symbolic link to the device (default: none)
--buffer, -b          : Size of buffer (default: 15000)
--queue, -q           : V4L2 queue length (default: 2)
--priority, -p         : Set realtime priority (default: 60)
--pid, -P             : MPEG-TS PID (default: 0x666)
--tap, -t             : Use ethernet bridging (default: no)
--null, -N            : Inject PID 0x1FFF packets (default: no)
--macsfile, -M        : File with IP/MAC mappings (default: not used)
--background, -B      : Go to background (default: no)
--logfile, -l         : Path to the status file in background mode (default: /tmp/ule.status)

/ #
```

Figure 7: Usage description of the ULEFramer program

- **devicename:** This is the name of the modulator device to open. You are required to specify something, otherwise the program will not start. Suppose your `ns_nightshade` peripheral is configured in your device tree at the address `0x44000000`. This parameter should then be:  
*44000000.ns\_nigthshade*
- **symlink:** If you specify this option the program will create a symbolic link to the opened device. This useful if you want to access it later using `v4l2-ctl` or other tools. Typically */dev/nightshade* is used.
- **buffer:** The framer has an internal buffer that is used to queue incoming packets before transmission. With this parameter you set the size. The larger the buffer the higher the tolerance to rate variations, but the higher the potential latency. You can try increasing this if you see sporadic packet drops in the log. If packets are dropped continuously you should reduce the source rate or increase the transmission bitrate since a larger buffer will just delay the onset of the problem.
- **queue:** The number of Video4Linux2 buffers to use. More buffers improve the robustness under high CPU load but increase latency. Two to four are reasonable values. If you see transmitter underruns you can try increasing this. A few underruns are harmless, as long as some hardware padding insertion is enabled (null packets or dummy frames, see Section 3.5).
- **priority:** This parameter sets the realtime priority for the framer. It is recommended to leave it at the default value of *60*. To turn off realtime mode, specify *0*.

- **pid:** The MPEG-TS stream is a multiplex. Different logical streams are identified by their 13-bit Program ID (PID) value. You could for example have a video stream at PID 0x100, the accompanying audio at 0x101, the PAT at 0x0 and the PMT at 0x1. The value you choose doesn't matter, as long as it is unique in the multiplex and you specify the same value in the receiver. The value 0x1FFF is reserved for null packets and may not be used.
- **tap:** If you set this parameter you enable Layer 2 mode ('TAP' interface). If you don't specify it the interface will operate in Layer 3 mode ('TUN' interface). What you need depends on the system configuration. See the section on network configuration (3.6.1) for more information.
- **null:** Inject NULL packets if the framer is idle. Enable this if you are using single MPEG-TS mode and not using hardware NULL packet insertion.
- **macsfile:** Specifies a file containing MAC/IP mappings. This is only used in TUN mode. It is optional unless your configuration involves Layer 3 routing. See the section on network configuration (3.6.1) for more information.
- **background:** Go to background. This allows you to start the program while the Zybo continues processing other programs. Omit this when testing, set it when running for real.
- **logfile:** The framer outputs status information that is useful for debugging. You cannot see this information when you enabled background mode. Using this option you can specify a file where this information will be stored. The default value is usually fine.

When the program is running without errors the virtual network interface will be created. You may also start seeing a valid MPEG-TS stream at the receiver side. To transmit actual data, the interface must be configured correctly. This is explained in the next section.

### 3.6.1 Network configuration

This section assumes some familiarity with networking protocols (IPv4, IPv6, Ethernet). The expected end result is a wireless network between the video encoder board and a computer. The first design choice is whether to use a

| Layer # | Layer Name          | Protocol           | Protocol Data Unit     | Addressing  |
|---------|---------------------|--------------------|------------------------|-------------|
| 5       | Application         | HTTP, SMTP, etc... | Messages               | n/a         |
| 4       | Transport           | TCP/UDP            | Segments/<br>Datagrams | Port #s     |
| 3       | Network or Internet | IP                 | Packets                | IP Address  |
| 2       | Data Link           | Ethernet, Wi-Fi    | Frames                 | MAC Address |
| 1       | Physical            | 10 Base T, 802.11  | Bits                   | n/a         |

Figure 8: Overview of different layers

Layer 2 or Layer 3 encapsulation. The different layers are shown on Figure 8. A comparison of both modes is given in Table 2.

In L2 mode you can bridge different network segments together. This makes a transparent connection that needs no further configuration. The whole system will look like one big LAN. In L3 mode it is not possible to create bridges, packets are routed instead. This requires that you assign different address ranges (subnets) to different parts of the system. Routing tables should then be filled in to make the packets arrive at their destination.

First, we will explain how to configure the interface in Layer 2 (TAP, L2) mode. Then, the a Layer 3 setup example is given. Unless you are going to run a pure L2 protocol or pure bridging you should do (parts of) the L3 protocol configuration as well. The L3 configuration is independent of the interface TUN/TAP mode.

### 3.6.1.1 Layer 2 setup

| Word Offset | Byte 0                  | Byte 1 | Byte 2             | Byte 3 |
|-------------|-------------------------|--------|--------------------|--------|
| 0x0000      | Destination MAC Address |        |                    |        |
| 0x0010      |                         |        | Source MAC Address |        |
| 0x0020      |                         |        |                    |        |
| 0x0030      | Type (Level 2)          |        |                    |        |

Figure 9: Layer 2 header

|              | Pro  | Con  |
|--------------|--|--|
| Layer 2 (L2) | <ul style="list-style-type: none"> <li>• Transparent bridging → simple configuration.</li> <li>• Supports almost all protocols (L2 and L3).</li> </ul> | <ul style="list-style-type: none"> <li>• Slightly more overhead.</li> <li>• Addressing (ARP and ND) issues when using unidirectional links.</li> </ul>   |
| Layer 3 (L3) | <ul style="list-style-type: none"> <li>• One layer higher so one layer less where things can go wrong.</li> <li>• Less overhead.</li> </ul>            | <ul style="list-style-type: none"> <li>• No support for L2 protocols (obviously).</li> <li>• L3 to L2 address translation table needs to be specified in the framer (macsfile option)</li> </ul> |

Table 2: Layer 2 vs Layer 3 mode

In Figure 9 the Layer 2 packet header is shown. The destination MAC address specifies the client which should receive the packet. The source address identifies the sender. The type field explains which protocol is being used (IPv4, IPv6, ...). The examples given assume you called the ULE device 'dvb0'. You can see your MAC address with the command: *ip link show dev dvb0*. The output will be something like:

```
44: dvb0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop
        state DOWN group default qlen 500
    link/ether 8e:34:ba:2c:85:92 brd ff:ff:ff:ff:ff:ff
```

The address *8e:34:ba:2c:85:92* is our MAC. The address is randomly generated every time you start the ULEFramer application. If you want to configure your own, do it as follows: *ip link set dev dvb0 address 00:11:22:33:44:55*. Note that the L2 address cannot be changed after the interface has been activated (state changed from DOWN to UP).

Before you can use the interface you have to bring it up: *ip link set dev dvb0 up*. You can rerun the 'link show' command to verify that the interface is indeed up. Technically speaking your Layer 2 setup is complete. You can send L2 frames to any device of which you know the MAC address (or you can broadcast/multicast). Usually you will want to run a Layer 3 protocol over this Layer 2 link. This is explained in the next section. If you want to create Layer 2 bridges, use the *brctl* command. Example:

```
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 dvb0
brctl setfd br0 0
ip link set dev br0 up
```

This will create a virtual switch with 3 ports. One goes to the ethernet port on the Zybo, another one to the DVB transmitter and the remaining one to the Zybo Linux system. A bridge on linux by default waits 30 seconds before it starts forwarding packets. During this time it learns about the network topology. In this case this is not needed so we set the forwarding delay to 0 using setfd. If you were to create this type of bridge on both the transmitter and receiver you would have made something that behaves like a normal wired network cable, but without the physical wire. Note that it will only send packets in one direction since at this time there is no return link yet. If the dvb0 interface is added to a bridge you should use the bridge device (br0) in the Layer 3 setup instead of dvb0.

### 3.6.1.2 Layer 3 setup

In this section we will setup an IPv4 link. If you configured the interface as TUN, you start immediately here. If you configured the interface as TAP, make sure that you did the instructions in the previous section first. The ULEFramer program supports IPv6, which is a newer protocol with many advantages. You can also use this protocol, but you need to search on the internet how to set it up. The advantages offered by IPv6 are not very useful in this project.

First you should choose an addressing plan: all your network segments need to be assigned an address and netmask (subnet, watch the video <sup>6</sup> if you don't know what this means). Warning: interfaces that you bridged together work like single interface when it comes to L3 configuration. Indeed, all forwarding magic happens on L2. Thus, if you bridged everything together you only need to configure one interface, the bridge (br0 in the previous section). The instructions given here for routing packets don't apply since there is only a single subnet and thus nothing to route.

As an alternative this type of routing it is also possible to use Network Address Translation (NAT). Instead of adding routes to the routing tables, you rewrite the source and destination addresses using connection tracking to create a valid path. This is technique sometimes used in complicated networks, look on the internet if you are interested.

---

<sup>6</sup>[https://www.youtube.com/watch?v=aA-8owNNy\\_c](https://www.youtube.com/watch?v=aA-8owNNy_c)

While theoretically you can use any IP address, it is recommended to use one in a range reserved for private deployments. The most well known are ranges 192.168.x.x and 10.x.x.x. The 10.x.x.x range is used for the KULeuven network so it is not recommended to use this one due to the risk of address conflicts.

In this example we will assume two different networks are in use:

- The (ethernet) network between the encoder and the Zybo
- The (dvb) network between the Zybo and the ground station PC.

Note: there is also the ESAT network between the ground station PC and the computer where the video is watched. This configuration is mostly done for you and will be explained later.

For the network with the encoder we use the subnet 192.168.123.x/24. This subnet has 254 addresses, not counting the broadcast and network address. Although only 15 are needed, /24 is a standard netmask. Since these are private ranges, we are not wasting address space by over-provisioning.

Configure the ethernet port as follows:

```
ip addr add 192.168.123.1/24 dev eth0
ip link set dev eth0 up
```

When using L2 mode together with bridging, please use 192.168.123.2 since 192.168.123.1 is the address of the receiver bridge.

The encoder IP will be visible on the receiver PC. Therefore it has to be unique. Your encoder will be delivered with an IP address in the 192.168.123.x range. The value of  $x$  is written on a sticker attached to the board. You may change it if needed, but please change it back to what it was before handing it in. Verify that the network connection works by pinging the encoder: *ping 192.168.123.x*. You should see replies if you did everything correctly.

*For L2 networks only: Since we only have an unidirectional link L2 address autodiscovery does not work (the replies will never make it back to the sender). You need to configure static ARP (IPv4) or ND (IPv6) for the MAC addresses you want to use. For example: `arp -s 192.168.111.111 00:22:44:66:88:aa`. Do this static configuration anywhere where an ARP or ND request would otherwise be sent.*

If you use L2 bridging, replace eth0 with br0. In this case, your L3 configuration is complete, skip the rest of this section.

Now, we need to choose the subnet between the Zybo and receiver. In this document I will use 192.168.10.x/24, but since multiple groups may use

the receiver at the same time you should use something unique to avoid routing conflicts. Discuss the address and netmask you will use with the other groups to make sure it is unique. Do not use 192.168.123.x since this is already used for the encoder. Also do not use 192.168.124.x since this is used in the receiver PC. Set your chosen address and netmask on dvb0 in the same way we did it for eth0. In my case:

```
ip addr add 192.168.10.1/24 dev dvb0
```

Tip: You can add multiple addresses to one interface. You can verify the address configuration with the command: *ip addr show dev dvb0*. IPv4 addresses are called inet. You may see some inet6 addresses, these are for IPv6.

Now, log into the receiver PC and configure another address in this subnet there. I used 192.168.10.2. See the receiver section (4) for instructions. Be careful not to change the configuration of the wrong interface, as this will not work and can cause problems for other groups.

Now we can check the routing table on the Zybo. Run the command: *ip route*. A typical output would be:

```
192.168.123.0/24 dev eth0  proto kernel  scope link      \
                               src 192.168.123.1
192.168.10.0/24 dev dvb0  proto kernel  scope link      \
                               src 192.168.10.1
```

Your output may be slightly different, but the gist should be the same. We see that packets going to an address in the range 192.168.123.0/24 are sent via eth0 using 192.168.123.1 as source address. On the other hand packets to 192.168.10.0/24 are sent through dvb0, using 192.168.10.1 as source address. So far so good.

But how do the RTP packets from the encoder reach the receiver? Enter something called a default route. This is a route of last resort. Any packet that not match any other routing table entries is handled by this. The routing table on the encoder is this:

```
default via 192.168.123.1 dev eth0  proto static
192.168.123.0/24 dev eth0  proto kernel  scope link      \
                               src 192.168.123.15
```

By default, the encoder sends the RTP packets to 192.168.124.1, which is the IP of the receiver. Since there is no routing table entry for the subnetwork 192.168.124.0/24 the default route is used. The packets are thus sent to the

gateway 192.168.123.1, which is the IP of the Zybo. In L2 mode this is the IP of the big dvbnet bridge on the receiver PC.

Nothing described here will work: linux by default does not allow forwarding packets for security reasons. Enable IPv4 forwarding with the command: *echo 1 >/proc/sys/net/ipv4/ip\_forward*. Do this on the Zybo. It has already been done for you on the receiver.

Your Zybo will receive packets from the encoder with a destination IP of 192.168.124.1. It does not know what to do with them since it has no routing table entry for 192.168.124.1. The precious video packets are being dropped on the floor... Add a routing table entry that will forward the packets to the receiver to fix this.

Packets are still not arriving at the receiver. This is because the ULE standard does not allow off-link routing without knowing the receiver L2 address<sup>7</sup>. This is where the macsfile comes in. Look up the MAC address of the receiver channel you are using (in the example 96:e5:ed:3c:d1:08) and make a file on the Zybo with the following contents:

```
4 192.168.10.2 96:e5:ed:3c:d1:08
```

Restart ULEFramer with this file specified as macsfile. The first number is 4 or 6, for IPv4 or IPv6. The second part is the IP address and the third part the L2 MAC.

There is a final pitfall. Many systems use something called Reverse Path Filtering (RPF). If a reply packet would not go out over the interface where the request was received (a so called martian) it is discarded. This is to make it more difficult to spoof source addresses. You need to add a valid reverse route to make RPF happy. RPF is disabled on the receiver PC, so you only need to worry about this if you use your own system as receiver.

If you got here and packets are arriving at the receiver (check using tcpdump) your configuration is done.

**Do not create any extra default routes on the receiver PC. This will make the system inaccessible since it will use your route when trying to send data to the ESAT network. When adding routes, use specific ones!** Reboot the receiver PC if it can no longer be accessed. The root filesystem is a RAMDisk so you may just switch it off without shutting it down.

---

<sup>7</sup>To know why, imagine what would happen if there are two routers connected to the same DVB network.



### 3.6.2 Receiving

How to receive this ULE stream? Use the linux application dvbnet: *dvbnet -a adapter -n demux -p PID -U*. Replace adapter and demux with the number of the receiver you plan to use. Replace the PID with the PID you chose when starting ULEFramer. -U configures dvbnet for ULE. Leave this out for MPE.

A new network interface named dvbX.X will be created that will receive the packets your transmitter sends. Do not forget to bring it up or it won't do anything at all. Make sure the demodulator you are using is tuned and locked to your transmitter before wasting time on debugging network issues.

### 3.7 EagleTXCalibrate

The calibration procedure has been changed because a new version of the frontend has been made. This will be updated once the procedure is known.

## 4 Receiver PC

This section will be added in a later version of the document (when the PC is ready). The used DVB-S2 receiver is a TBS6908 and the DVB-T2 receiver is the TBS6205.