Institutionen för systemteknik
Department of Electrical Engineering

**Examensarbete**

# Implementation of a VBR MPEG-stream receiver in an FPGA

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan vid Linköpings universitet
av

**Jonathan Liss**

LiTH-ISY-EX--12/4625--SE

Linköping 2012


**Linköpings universitet**
**TEKNISKA HÖGSKOLAN**

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

# Implementation of a VBR MPEG-stream receiver in an FPGA

Examensarbete utfört i Elektroniksystem
vid Tekniska högskolan i Linköping
av

**Jonathan Liss**

LiTH-ISY-EX--12/4625--SE

Handledare: **Carl Ingemarsson**
ISY, Linköpings universitet
**Sebastian Abrahamsson**
A2B Electronics AB

Examinator: **Oscar Gustafsson**
ISY, Linköpings universitet

Linköping, 13 September, 2012

**Avdelning, Institution**
Division, Department

Division of Electronics Systems
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

**Datum**
Date

2012-09-13

**Språk**
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

**Rapporttyp**
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

**ISBN**
__

**ISRN**
LiTH-ISY-EX--12/4625--SE

**Serietitel och serienummer** **ISSN**
Title of series, numbering __

**URL för elektronisk version**
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-89886

**Titel**
Title

Implementering av en mottagare för MPEG-strömmar av variabel bitrate för FPGA

Implementation of a VBR MPEG-stream receiver in an FPGA

**Författare**
Author

Jonathan Liss

**Sammanfattning**
Abstract

Nowdays, the transmission of digital TV-signals tends to move towards more untraditional medias, such as TCP/IP networks.

This thesis focused on the problems involved in receiving MPEG transport streams of variable bitrate from a TCP/IP connection, such as jitter and clock synchronization. A suggestion for recovering the transport stream is presented along with a implementation for an Xilinx FPGA targeted for a head end device. The implementation was written in a mix of VHDL and Verilog.

**Nyckelord**
Keywords

MPEG, DVB, TS, transport stream, VBR, CBR, FPGA, PLL, jitter

# Abstract

Nowdays, the transmission of digital TV-signals tends to move towards more untraditional medias, such as TCP/IP networks.

This thesis focused on the problems involved in receiving MPEG transport streams of variable bitrate from a TCP/IP connection, such as jitter and clock synchronization. A suggestion for recovering the transport stream is presented along with a implementation for an Xilinx FPGA targeted for a head end device. The implementation was written in a mix of VHDL and Verilog.

# Sammanfattning

Sändning av digitala TV-signaler tenderar nuförtiden att göras över icke traditionella medium, så som TCP/IP-nätverk.

Den här rapporten fokuserar på att undersöka problemen som finns i samband med att ta emot MPEG transportströmmar av variabel bitrate som skickas över TCP/IP-nätverk, så som jitter och klocksynkronisering. Ett förslag på åtgärd presenteras samt en implementering för en Xilinx FPGA som används i en huvudcentralsenhet. Implementeringen gjordes i en blandning av VHDL och Verilog.

# Acknowledgments

Thanks to A2B Electronics and Linköping University for making this thesis possible, and special thanks goes to Sebastian Abrahamsson and Markus Råbe as well as all people who helped me during the thesis.

I'd also like to thank examiner Oscar Gustafsson and supervisor Carl Ingemarsson for helping me with the writing of this thesis.

# Contents

# List of Figures



# List of Tables

# Abbreviations

**ASI** Asynchronous Serial Interface

**CBR** Constant Bit Rate

**CPU** Central Processing Unit

**DDR** Double Data Rate

**DVB** Digital Video Broadcasting

**DVD** Digital Video Disc

**FIFO** First In First Out

**FIR** Finite Impulse Response

**FPGA** Field Programmable Gate Array

**IP** Internet Protocol

**MDI** Media Delivery Index

**MPEG** Motion Picture Expert Group

**MPTS** Multiple Program Transport Stream

**MTU** Maximum Transmission Unit

**NCO** Numerical Controlled Oscillator

**PCR** Program Clock Reference

**PID** Packet ID

**PLL** Phase Locked Loop

**PS** Program Stream

**RTP** Realtime Transport Protocol

**SPTS** Single Program Transport Stream

**STC** System Time Clock

**TCP** Tranport Control Protocol

**TS** Transport Stream

**UDP** User Datagram Protocol

**VBR** Variable Bit Rate

**VCO** Voltage Controlled Oscillator

**VHDL** VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

# Chapter 1

# Introduction

This chapter will give an introduction to the A2B Electronics AB company and the goal of this thesis work.

## 1.1 Purpose

The goal of this project is to investigate the difficulties involved in and design a method to receive MPEG transport streams of variable bit rate (VBR) sent over an IP link, as well as investigate whether a CPU or a hardware solution is to prefer.

Summarized, the following details will be investigated

- Investigate how to properly receive a VBR MPEG stream
- Investigate if the RTP protocol may be used to facilitate the receiving
- Investigate if a CPU is more suitable for a regulator

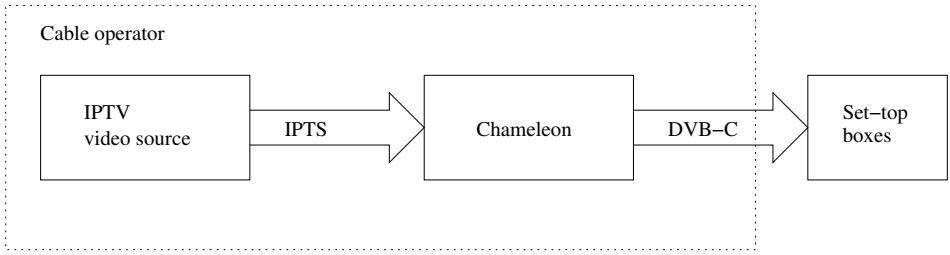
## 1.2 Background

A2B Electronics AB is a company that has specialized on creating systems for television operators, which includes a broad range of equipment (head end systems) for receiving and transmitting television and supports various kinds of interfaces (such as several types of DVB, IPTV and ASI for example).

One of their systems is called Chameleon, and an example application for such a device is to receive an IPTV stream, process it (select a number of channels) and send to an output for conversion to DVB-C television standard for a cable network. (Fig. 1.1) [1].

This device is currently capable of receiving IPTV streams of constant bit rate only, but the advantages of having support for streams with variable bit rate are several. First, it saves bandwidth in the network which may allow more streams without need to invest in new hardware. Second, there may be streams already

**Figure 1.1.** Example of a Chameleon application




available with no possibility to convert to constant bit rate. Third, a well designed solution will handle both variable and constant bit rate streams which allows more flexibility.

## 1.3 Method

This thesis will be performed by studying standard documents and looking for literature on the subject. I will also study the possibility to solve the problem using a CPU, as suggested by A2B. Finally an implementation of the solution will be done with the intention to run on a Chameleon device.

## 1.4 Delimitations

In this thesis, I will study methods of properly receiving VBR streams, create a design and implement it in hardware (VHDL and Verilog) for A2B's Chameleon for testing. The design will then be evaluated and compared against a proposed CPU solution.

## 1.5 Tools and hardware

Following are a list of tools and hardware used for implementation.

#### Chameleon platform

Chameleon is the head end hardware platform used for this thesis project. This hardware contains an FPGA, a CPU, memories and various connectors for different interfaces, such as Ethernet connectors, ASI in/out, DVB in/out etc.

#### Xilinx FPGA

The targeted FPGA for the implementation was the Xilinx Spartan-6 XC6SLX100T. This FPGA has 15,822 slices and 976 kB of block-RAM available (table 1.1), as well as integrated DDR memory controller blocks.

**Table 1.1.** Spartan-6 hardware

| Logic Cells | Slices | Flip-Flops | DistRAM (Kb) | DSP48 | BlockRAM (Kb) |
|---|---|---|---|---|---|
| 101,261 | 15,822 | 126,576 | 976 | 180 | 4,824 |

### Xilinx ISE

The IDE that was used in the design process is Xilinx ISE version 13.3, which is free with an evaluation license. The languages used were a mix of both Verilog and VHDL.

### Xilinx ChipScope

The Xilinx ChipScope software was used to verify the implementation when run on the targeted hardware. It works by attaching a JTAG device to the FPGA and makes it possible to look at internal signals in real time.

### Xilinx ISim

Xilinx ISim was used to simulate the design during the design process. This program resembles the ModelSim software.

### DekTec StreamXpert

A tool and software used to receive and analyze MPEG2 audio/video streams over the ASI interface.

### Simulink

Simulink is a graphical extension to MATLAB used to create and simulate models of systems.

# Chapter 2

# Related theory

This chapter describes some background theory and explanations needed in order to understand the problem.

## 2.1 MPEG

Motion Picture Expert Group (MPEG) is a working group that is responsible for several standards of video and audio coding techniques, the most common being called MPEG-2 and MPEG-4 [2].

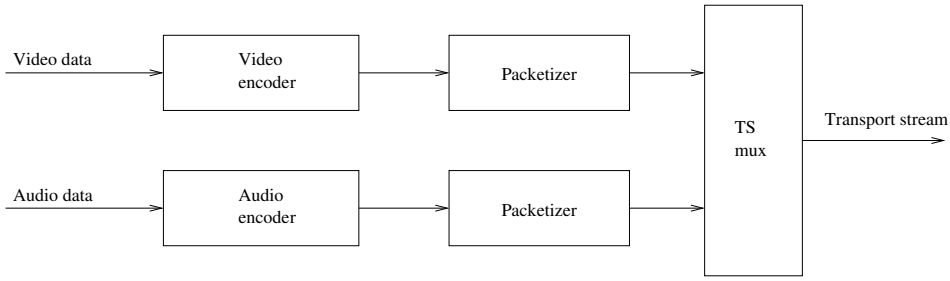
## 2.2 DVB

The Digital Video Broadcasting Project (DVB) is an industry-led consortium of around 250 broadcasters, manufacturers, network operators, software developers and others in over 35 countries committed to designing open technical standards for the global delivery of digital television and data services. Some technology standards defined by DVB are DVB-C (Cable), DVB-S (Satellite) and DVB-T/T2 (Terrestrial). While DVB is the main standard used for video broadcasting in Europe, other standards are also used elsewhere. Some of these are ATSC used in the USA and DTMB used in China [12].

## 2.3 Transport streams

The MPEG transport stream (TS) is a standard designed to encapsulate and send MPEG audio and video (program data) over broadcast systems, such as DVB and ATSC, as illustrated in Fig. 2.1. This standard is also known as TU-T Rec. H.222.0 [6]. There are two variants of transport streams, single program transport stream (SPTS) and multiple program transport stream (MPTS). SPTS means that only one program is sent in a stream and MPTS means that one stream can contain multiple programs, where the programs are identified by special packets that contains tables such as the Program Association Table (PAT).

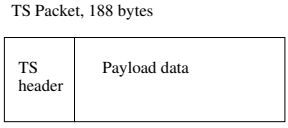
**Figure 2.1.** TS stream




These transport streams may be of constant bit rate (CBR) or variable bit rate (VBR). The actual MPEG data created by the video/audio encoder is inherently always VBR; so empty packets (null packets) are inserted when a CBR stream is needed to maintain a constant bit rate. This is necessary for some types of hardware, such as DVB modulators.

Transport streams consist of packets, each 188 bytes in size (Fig. 2.2). An additional error correction code may sometimes be included which extends the size to 204 bytes. The packet size was originally selected to be compatible with ATM systems, and to be usable with less reliable transmissions. An alternative to transport streams are Program Streams (PS), mainly intended for more reliable media such as the DVD and Blu-ray Disc format. The format of packets is described in Table 2.1

**Figure 2.2.** TS packet structure




The sync byte indicates start of a TS-packet and is always the first byte in a header with the value 0x47. Following are a number of flags and the Packet Identifier (PID). When using CBR streams, empty packets are included as described earlier; such an empty packet has the PID set to 0x1FFF, which is reserved for null packets. This way CBR streams can easily be converted to VBR by removing these packets. The adaption field contains some information that is not available in every TS-packet, and is described in Table 2.2. The payload data contains the actual MPEG video, audio and meta data and makes up for the rest of the TS-packet until all 188 bytes are used.

The adaption field contains some additional flags and values. The most important value in the adaption field is the Program Clock Reference (PCR) timestamp, which contains the timing information of the video stream. Not every packet contains the adaption field and PCR field; it is only guaranteed that it will come

**Table 2.1.** TS packet content

| Field name | Size |
|---|---|
| Sync byte, (0x47) | 8 bits |
| Transport Error Indicator | 1 bit |
| Payload Unit Start Indicator | 1 bit |
| Transport Priority | 1 bit |
| PID | 13 bits |
| Scrambing control | 2 bits |
| Adaption Field Exists | 2 bits |
| Continuity Counter | 4 bits |
| Adaption field | Variable |
| Payload data | Variable |

**Table 2.2.** Adaption field content

| Field name | Size |
|---|---|
| Adaption Field Length | 8 bits |
| Discontinuity indicator | 1 bit |
| Random Access indicator | 1 bit |
| Elementary Stream Priority indicator | 1 bit |
| PCR | 33 + 6 + 9 bits |
| OPCR | 33 + 6 + 9 bits |
| Splice countdown | 8 bits |
| Stuffing bytes | Variable |

in a periodic interval. The largest allowed interval is 100 ms between two PCR values, altough it is recommended in DVB systems that this period is no greater than 40 ms. The format of the PCR value is stored in three parts: first 33 bits is a timestamp based on a 90 kHz clock, then 6 empty bits and the last 9 bits is a timestamp based on a 27 MHz clock. This 27 MHz clock is called the System Time Clock (STC) and is the base clock in a video transmitter. OPCR (Original PCR) may be used when including a TS stream into another TS stream to preserve the original PCR timestamps, e.g. when remultiplexing streams.

The PCR values can be combined to get the full PCR value.

$$PCR = PCR_{long} \cdot 300 + PCR_{short}$$

$PCR_{long}$ is the 33 bit value and $PCR_{short}$ is the 9 bit value.

TS packets with PCR fields will be referred to as PCR packets in this document, although such packets usually also contains MPEG data. Stuffing bytes is used to fill the rest of the space until the size of the adaption field is equal to the Adaption Field Length value [6].

## 2.4 Network communication

Some different methods for transporting TS streams over IP networks are explained here.

### 2.4.1 UDP

User Datagram Protocol (UDP) is used to transmit TS packets over IP networks; it has the clear advantage over Transport Control Protocol (TCP) that it favors timeliness over reliability. There is no need to verify the connection between the transmitting and receiving devices and lost packets does not have to be resent.

The size of datagrams is limited to the Maximum Transmission Unit (MTU) of 1500 bytes [10, 11].

### 2.4.2 RTP

The Real-time Transport Protocol (RTP) is an extension to UDP and TCP, and can also be used to transport TS packets. The RTP packet format is explained in Table 2.3. RTP packets may arrive out of order, to correct this the sequence number is used to identify a single packet in a stream. The timestamp field contains timestamps which depends on the exact RTP profile, in the case of MPEG video streams these comes from a 90 kHz clock (MPEG presentation clock) which is based on the 27 MHz video clock. Immediately following the RTP header is the payload data, which in this case are the TS packets [9].

**Table 2.3.** RTP header content

| Field name | Size |
|---|---|
| Version | 2 bits |
| Padding (P) | 1 bit |
| Extension (E) | 1 bit |
| CSRC Count (CC) | 4 bits |
| Marker (M) | 1 bit |
| Payload Type (PT) | 7 bits |
| Sequence Number | 16 bits |
| Timestamp | 32 bits |
| SSRC | 32 bits |
| CSRC | 32 bits + Variable |
| Extension header | Variable |

### 2.4.3 Unicast

Unicast is the method used when one transmitter sends data to one receiver. This is the common method of communication in most IP network applications.

### 2.4.4 Multicast

Multicast is the method used to send data from one transmitter to multiple receivers. In IPv4, the IP ranges 224.0.0.0 to 239.255.255.255 are reserved for multicast communication. Multicast communication is commonly organized by using IGMP (Internet Group Message Protocol).

This method is commonly used when transmitting transport streams since the network load will be independent of the amount of clients receiving the stream.

## 2.5 ASI

ASI (Asynchronous Serial Interface) is a serial interface used to send video data and is often used to transmit an MPEG TS stream, this interface is commonly available on professional equipment. Tools such as DecTek StreamXpert can be used to receive and analyze ASI streams [3].

# Chapter 3

# Description of the problem

This chapter describes the difficulties involved in receiving and recovering a IPTS stream.

## 3.1 Network jitter

Network jitter is introduced by delays in network equipment, such as switches and routers, but also the fact that multiple TS packets are commonly assembled into one single datagram before they are sent. One Ethernet frame can contain at most 1500 bytes, so up to seven TS packets (1316 bytes) are usually sent toghether in one single packet. Less than seven TS packets may be preferred sometimes when transmitting a low bit rate stream to reduce the jitter, altough the packets/datagram must remain constant.

Fig. 3.1 illustrates a transport stream packed into UDP packets and sent over an IP network, and demonstrates the jitter that is introduced. The TS packets in black indicates packets with a PCR value, and each complete UDP packet contains seven TS packets. Jitter is introduced by the IP transmission itself which causes the arrival time of a datagram (UDP packet) to be slightly random, and as part of the packing process which means that seven packets are always received immediately.

Another mode of operation is to keep a varying number of TS-packets in an Ethernet frame and instead prioritize to keep the time delay between each datagram constant, which will cause the major part of the jitter to originate only from the network equipment. However, this mode is rarely used and not well supported by MPEG and DVB equipment so it will not be considered in this project [8].

Network jitter is commonly transient in its nature, an illustration is provided in Fig. 3.2 where the packet rate of a captured CBR transport stream is plotted over time. This fact might be of use later when optimizing the design.

One way to measure the jitter is to use MDI (Media Delivery Index), described in Section 3.5.5, or direct PCR measurements for the impact on the TS stream (described in Section 3.5).

**Figure 3.1.** IP transmission of TS packets

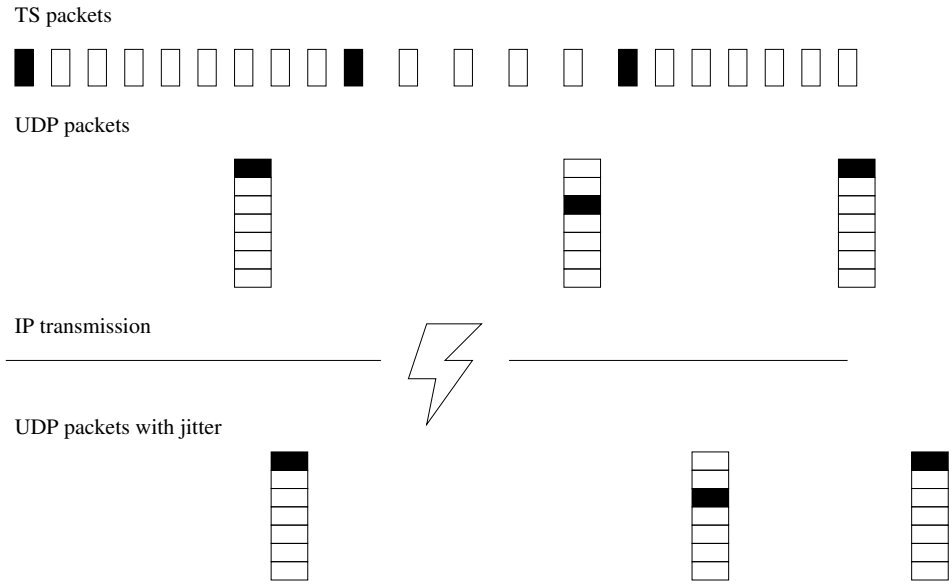
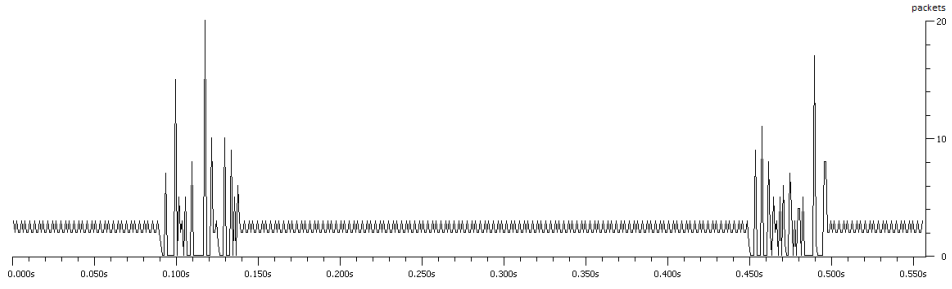



**Figure 3.2.** Example of network jitter in a captured TS-stream




## 3.2 Jitter buffer

The network jitter must be removed when receiving the TS stream in order to properly transmit this stream to interfaces with more narrow jitter tolerances, such as DVB or ASI. One way to do this is to store all TS-packets temporarily in a buffer and then read out the packets at a constant rate without jitter. This buffer must be large enough to handle the jitter without overflowing or underflowing, but a too large buffer might introduce a significant delay in the system. In the case of IPTV-streams, the standard specifies a maximum acceptable jitter in the network of $\pm 200$ ms, which is a starting point when selecting a buffer size. Another possible cause of overflow/underflow is the clock rate deviation between the transmitter and

receiver, this will be discussed in Section 3.3 [8].

Since the bit rate of the stream is varying, some control must be implemented to properly read out the TS stream from the buffer. One problem here is that the bit rate information is not stored in the TS-stream in any way: rather it must be extracted by analyzing the stream.

In the case of handling a CBR-stream, the buffer fill level itself can be utilized to implement the read out control. This is because the buffer fill level should be as near constant as possible, and this is solved in the current Chameleon implementation by first filling the buffer to half and then trying to keep it that way by adjusting the read out speed (increasing the output rate if the buffer fill rate increases or decrease the output rate if the buffer fills rate decreases). Momentary level changes caused by jitter will be filtered out. This is not possible with VBR-streams since the fill level will vary naturally, so another method must be used to read a jitter free stream back from the buffer.

## 3.3 System Time Clock deviation

The transmitter's STC clock will differ slightly from the nominal clock rate due to deviations in electrical components, as it's practically impossible to create perfect components with exact values. The deviation manifests itself in frequency offset, frequency drift and jitter. The clock constraint is defined in the MPEG TS standard document, with limitations on both the frequency offset and clock drift stability.

$$27000000 - 810 \leq \text{System clock frequency} \leq 27000000 + 810$$

$$\text{Rate of change of System clock frequency over time} \leq 75 \cdot 10^{-3} Hz/s$$

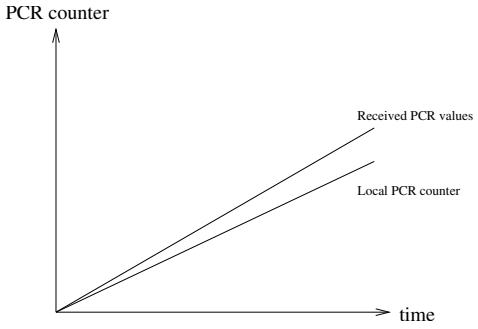
The consequence of the clocks not being synchronized is that the transmitter and receiver has a slightly different view on the data rate, which leads to that the buffer is either not filled or drained fast enough, and eventually the jitter buffer will either overflow or underflow. The clock rate deviation may be low and it may run fine for a long time before any symptoms are apparent, but it will happen eventually. So this problem is essential to address, especially for systems that are designed to be reliable and run for very long periods of time without interrupts (such as TS transmitters). Fig. 3.3 illustrates how the PCR values will gain an offset due to the STC clocks being out of synchronization [6].

## 3.4 Verification

In order to test and verify the design, short-term measurements must be done on the output for jitter, and long-term measurements must be done to verify that the STC clock is properly stabilized.

The following chapters will discuss some measurement methods that may be used to verify the implementation.

**Figure 3.3.** STC deviation




## 3.5 Measurements guidelines for DVB systems

In this chapter is based on some basic measurement methods from the DVB measurement document, which will be useful when measuring the performance of the VBR stream receiver. That document specifies a set of parameters that are recommended for evaluation of DVB TS streams, which are ordered in first, second and third priority errors. Softwares such as StreamXpert is able to perform these measurements automatically [4, 3].

### 3.5.1 First priority errors

First priority errors includes conditions that are necessary for de-codability of a TS stream, these are listed in Table 3.1. All of these would indicate serious problems in the implementation: such as loss of packets in the system, corrupt packets or byte misalignments, and may not appear in a working implementation [4].

### 3.5.2 Second priority errors

Second priority errors are recommended for continous or periodic monitoring, these are listed in Table 3.2 [4].

These are not as fatal as the first priority errors and might not even be introduced by the VBR stream receiver itself, but should still be considered. No. 2.3 - 2.4 might be of particular interest in this case.

### 3.5.3 Third priority errors

There also exists a set of third priority errors, but these are application dependant and not relevant for the actual transportation of the stream, so these are not included in this document. For information regarding these, check Chapter 5.2.3 in [4].

**Table 3.1.** TS stream: First priority errors

| No | Indicator | Precondition |
|---|---|---|
| 1.1 | TS_sync_loss | Loss of synchronization with consideration of hysteresis parameters |
| 1.2 | Sync_byte_error | Sync_byte note equal to 0x47 |
| 1.3 | PAT_error | PID 0x0000 does not occur at least every 0,5 s, a PID 0x0000 does not contain a table_id 0x00, Scrambling_control_field is not 00 for PID 0x0000 |
| 1.4 | Continuity_count_ error | Incorrect packet order, a packet occures more than twice, lost packet |
| 1.5 | PMT_error | Sections with table_id 0x02 do not occur at least every 0,5 s on the PID which is referred to in the PAT, Scrambling_control_field is not 00 for all PIDs containing sections with table_id 0x02 |
| 1.6 | PID_error | Referred PID does not occur for a user specified period |

### 3.5.4 PCR jitter measurements

The PCR jitter tolerance is $\pm 500ns$ and is covered by Test case 2.4 in Table **??** This measurement is usually done by counting the number of bytes between each PCR value and then compare with the actual PCR values. This presumes constant bit rate and is not applicable on VBR streams directly, but can still be used since a failure of this test might indicate problem with the internal VBR to CBR conversion process in Chameleon, which in turn would indicate a problem with properly reading the stream back from the jitter buffer [4].

### 3.5.5 Media Delivery Index

One method of verifying an IPTS stream quality is called MDI (Media Delivery Index), and consists of two different measurement methods: the Delay Factor (DF) and Media Loss Rate (MLR). Results of MDI measurements are written as numeric values on the form *DF : MLR*.

The DF is computed as following:

$$X = |BytesReceived - BytesDrained|$$

$$DF = \frac{[max(X) - min(X)]}{MediaRate}$$

$MediaRate$ is the received number of bytes / second.

**Table 3.2.** TS stream: Second priority errors

| No | Indicator | Precondition |
|---|---|---|
| 2.1 | Transport_error | Transport_error_indicator in the TS-header is set to "1" |
| 2.2 | CRC_error | CRC error occurred in CAT, PAT, PMT, NIT, EIT, BAT, SDT or TOT table |
| 2.3 | PCR_error | PCR discontinuity of more than 100 ms occurring without specific indication. Time interval between two consecutive PCR values more than 40 ms |
| 2.3a | PCR_repetition_error | Time interval between two consecutive PCR values more than 40 ms |
| 2.3b | PCR_discontinuity_indicator_error | The difference between two consecutive PCR values ($PCR_{i+1} - PCR_i$) is outside the range of 0 ... 100 ms without the discontinuity_indicator set |
| 2.4 | PCR_accuracy_error | PCR accuracy of selected programme is not within ±500 ns |
| 2.5 | PTS_error | PTS repetition period more than 700 ms |
| 2.6 | CAT_error | Packets with transport_scrambling_control not 00 present, but no section with table_id = 0x01 present Section with table_id other than 0x01 found on PID 0x0001 |

The DF value indicates the latency contribution from the network to the total latency in delivering the video from source to destination. If the DF value continuously grows, there is a mismatch between the buffer fill rate and the buffer drain rate, meaning that the buffer will eventually underflow or overflow.

The MLR is computed as following:

$$MLR = \frac{(PacketsExpected - PacketsReceived)}{IntervalTime}$$

This value indicates the number of media packets lost per second.

The relevance when constructing a receiver is that the DF could be used to measure the output jitter; this must be within the IPTS specifications (±200 ms). If the DF is slowly increasing or decreasing, this would indicate some problem with the STC clock synchronization. MLR determines the packet loss; clearly there should be no loss of packets when measuring directly on the output from the device, so such a problem would indicate an underflow or overflow of the jitter buffer or another major problem [5].

# Chapter 4

# Proposed solution

This chapter will describe the proposed solution to the problems involved with receiving a VBR stream.

## 4.1 Reconstruction of STC

The local clock must be synchronized with the transmitter clock to ensure that the jitter buffer will never underflow or overflow. The only timing information available for the TS-stream is the PCR values that are sent once in a while, as well as the time the packets were received by the device. The local time can be constructed from a local PCR counter driven by the local system time clock.

**Evaluating the clock error**

The following steps can be used to get the error between the transmitter and the receiver clock:

1. Keep a local PCR counter that is driven by the local STC clock.
2. Each time a PCR packet is received, save the received PCR value and save a timestamp of the local PCR counter.
3. Calculate the difference between the timestamp and the received PCR value.

In theory, when no jitter is present and both clocks are equal, the difference should be equal if the local counter is synchronized to the TS stream's PCR values. In practice, there is jitter that randomly will affect the difference each time the value is calculated, and there is a clock offset that will remain constant and slowly contribute to the deviation of both counters.

### 4.1.1 Local PCR counter

A local PCR counter must be available to keep track of the local time of the receiver, and must be synchronized with the transmitter's PCR counter. That

can be done by loading a received PCR value into the counter, for example, if the difference to received PCR values are too large or if it is known that no TS packets has been received for a while (meaning that a new stream is available). The PCR counter can be implemented as a 42 bit counter driven by the STC. One important detail that must be kept in mind is that this counter must wrap at $(2^{33} - 1) \cdot 300 + 299$ rather than $2^{42} - 1$ to stay consistent with incoming PCR values. It's also important to consider the wrapping to zero every time both PCR values are compared against each other.

### 4.1.2 Timestamps

The local PCR counter can then be used to obtain timestamps for received PCR packets. The problem is that several TS-packets are stuffed into the same datagram, described in Section 3.1. This means that the exact arrival time of PCR packets might be unknown, as the PCR packet may be found on any position in a datagram and the arrival time is only known for the entire datagram. One option would be to ignore this and treat it as jitter, assuming that the postition will be random each time. The size of the jitter introduced is illustrated in Example 4.1, and this is not small enough to be neglected in a network with low jitter.

**Example 4.1: Datagram jitter in a 1MBit stream**

This example illustrates the jitter in a 1 MBit stream when ignoring the exact TS packet position in a datagram. The jitter will be less at higher bitrates.

$$1 \text{ MBit} \approx 665 \text{ packets/second}$$

$$665 \text{ packets/second} \approx 1.5 \text{ ms/packet}$$

$$\text{Max deviation of 7 packets } \approx 10\text{ms in jitter (at most)}$$

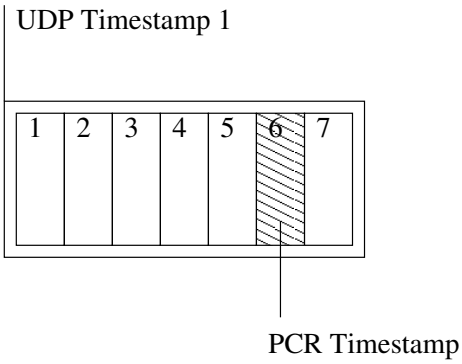
However, it is possible to achieve better results by using the fact that datagrams are sent once the last TS packet has been assembled. Fig. 4.1 illustrates a datagram with seven TS packets, and the actual timestamp available for the datagram is valid only for the last TS-packet.

### 4.1.3 Bitrate calculation

If also the bitrate is known, then it will be possible to interpolate a more accurate timestamp for the PCR packet. The bitrate can be obtained by counting the number of packets received between two PCR packets, using the following formula:

$$BPS = \frac{27 \cdot 10^6}{PCR_n - PCR_{n-1}} \cdot 188 \cdot 8 \cdot Packets$$

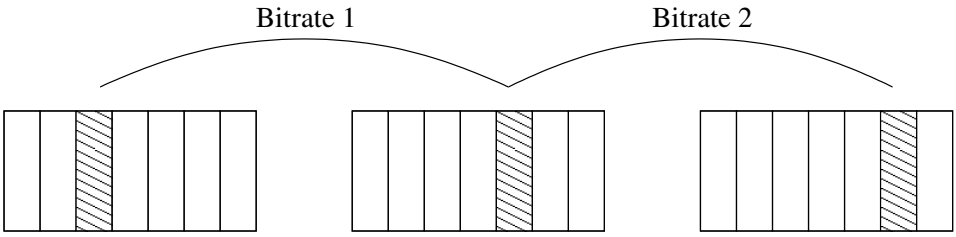
**Figure 4.1.** Visualization of a datagram




*Packets* is the total number of TS packets between two PCR values, and $PCR_n$ is the current PCR value and $PCR_{n-1}$ is the previous PCR value. This must be caluclated for every PCR packet received since the bitrate is variable and may change at any time.

To properly interpolate the timestamp, it is necessary to have access to three PCR packets at once. This is because the bitrate both before and after the actual packet is needed, as the PCR packet itself indicates a possible change in bitrate. Fig. 4.2 illustrates how the bitrate changes between PCR packets.

**Figure 4.2.** Visualization of several datagrams




The interpolated timestamp can then be calculated for a single datagram using the following formula:

$$DG_t - PCR_t = \frac{1}{BPS} \cdot NumPkts \cdot 188 \cdot 8$$

$DG_t$ is the time a datagram is received and $PCR_t$ is the estimated time when the PCR TS packet should have been received. $NumPkts$ is the number of TS packets following the PCR TS packet in that datagram. $BPS$ is the bit rate of the stream.

Because of this, we need at least two PCR packets in the buffer to accurately play back all packets in between these PCR packets. This is an important fact to keep in mind when calculating the size necessary for the jitter buffer.

Once this has been performed, the estimated TS packet timestamp ($PCR_t$) is known, it can finally be subtracted with the received PCR value ($PCR_r$) to get the error value containing both jitter and the clock offset.

$$PCR_{error} = PCR_t - PCR_r$$

This error value will be the input to the STC controller.

### 4.1.4 RTP timing

The RTP protocol adds additional timestamps based on a 90 kHz clock (the MPEG presentation frequency) which may be derived from the STC [9]. These timestamps are associated with the datagrams and provides additional timing information that may be used to control the clock. It is important to note however that this clock may not be based on the STC, especially in the case of receiving a MPTS which can contain several clock sources.

The decision was to not use the RTP protocol in this design, for the reason that the PCR values are still needed and must be examined in order to calculate the exact bit rate information of each moment in the stream (since the bit rate may change in the middle of a datagram and RTP timestamps are only available for the entire datagram), so it would not save any complexity. Another reason is the fact that RTP is not mandatory for IPTV streams, so basing the regulator on that would imply a restriction on streams.

An option would be to add support for RTP streams as an additional layer to boost performance of the STC controller in severe network conditions. This won't be considered in this project, but is a proposition for future work.

## 4.2 Designing the STC controller

An analysis of the error value was performed in order to decide which kind of controller to choose. As apparent by now, the error signal is affected by two sources. One is the network jitter which can be seen as a high-frequency component. The second is the clock offset which slowly accumulates an offset between the two counters, and this can be seen as a low-frequency component.

### 4.2.1 Analysis of the jitter error component

The jitter is random in nature and can be seen as noise. This is not entirely true for the individual case, but rather as a generalization. Given the jitter limitation $\pm 200$ ms, the lower frequency limit of the derived noise is approximately 5 Hz.

### 4.2.2 Analysis of the clock error component

The difference in clocks will slowly accumulate in the PCR counters, the local and the remote available as PCR timestamps. This difference will slowly get bigger with time, which basically behaves like a ramp.

The ramp has the transfer function:

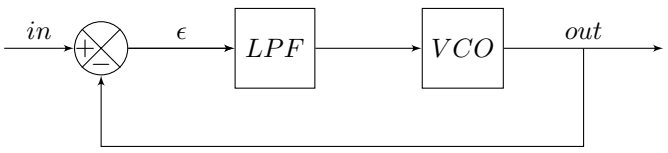
$$G(s) = \frac{1}{s^2}$$

This consists of mostly low frequency content.

### 4.2.3 The control loop

As suggested in [8], a PLL (Phase Locked Loop) may be used to create the local clock signal. The schematic of a typical PLL is described in Fig. 4.3, and is a control system that consists of a low-pass filter, an integrator and a feedback loop. The idea is that the output of the PLL should be identical to the input signal without the noise. The low-pass filter is used to filter noise, and the VCO is used to create the new clock.

**Figure 4.3.** PLL schematic




In order to implement a PLL in our system, it was necessary to identify the different components in the already existing design. The feedback signal is already constructed from the PCR values, earlier referred to as $PCR_{error}$.

The transfer function of the VCO is essentially an integrator with a scale factor:

$$G_{VCO}(s) = \frac{k_{VCO}}{s}$$

Where the scale factor is $k_{VCO}$. This is true because the output phase of the VCO is continously increasing, with the speed defined by its input. In our system, this would be identical to the local PCR counter which will be the reconstructed reference.

This leads to the conclusion that only a low-pass filter is necessary to implement the PLL in our design. The task of the low-pass filter is to remove jitter from the input signal, and the output from the filter will proportionally control the STC frequency.

### 4.2.4 Lowpass filter

The lowpass filter must have a very long time constant in order to filter out the jitter. Because of this, a FIR filter is usually unsuitable as the required number of taps directly depends on the desired cutoff frequency.

One way to implement this filter is to create an IIR circuit, and I choosed one variant of an exponential averager which is a simple variant of a low-pass filter. The equation of such an exponential averager is:

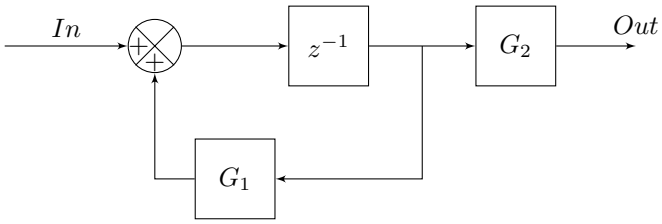
$$y[n] = x[n-1] + G \cdot y[n]$$

The gain $G$ should be close to 1 to improve performance, but not exactly 1 as that results in instability. The result $y[n]$ should be scaled down enough to suppress the jitter. Both these values were selected through a simulation where the effects could be more closely evaluated, the details are explained in Section 4.2.5.

### 4.2.5 Simulation of the lowpass filter

Simulink was used to perform a simulation of the filter circuit in Fig. 4.4 to obtain the filter constants.

**Figure 4.4.** The exponential averager circuit




The following parameters were used to simulate input data: Sampling rate is 25 Hz which corresponds to a PCR interval of 40 ms. The input consists of the error value in 27 MHz cycles between the local PCR counter and received PCR values, and is made from the combination of a ramp and noise. The ramp is used to simulate the clock offset and was set to increase by 30 each interval, which is equal to a clock offset of 750 Hz (close to the 810 Hz limit according to the specification). Noise is used to model network jitter and has the amplitude ±5400000 STC cycles, which corresponds to the maximum allowed jitter of ±200ms.

After some testing, the following parameters was selected:

$$\text{Gain: } G_1 = 0.99$$

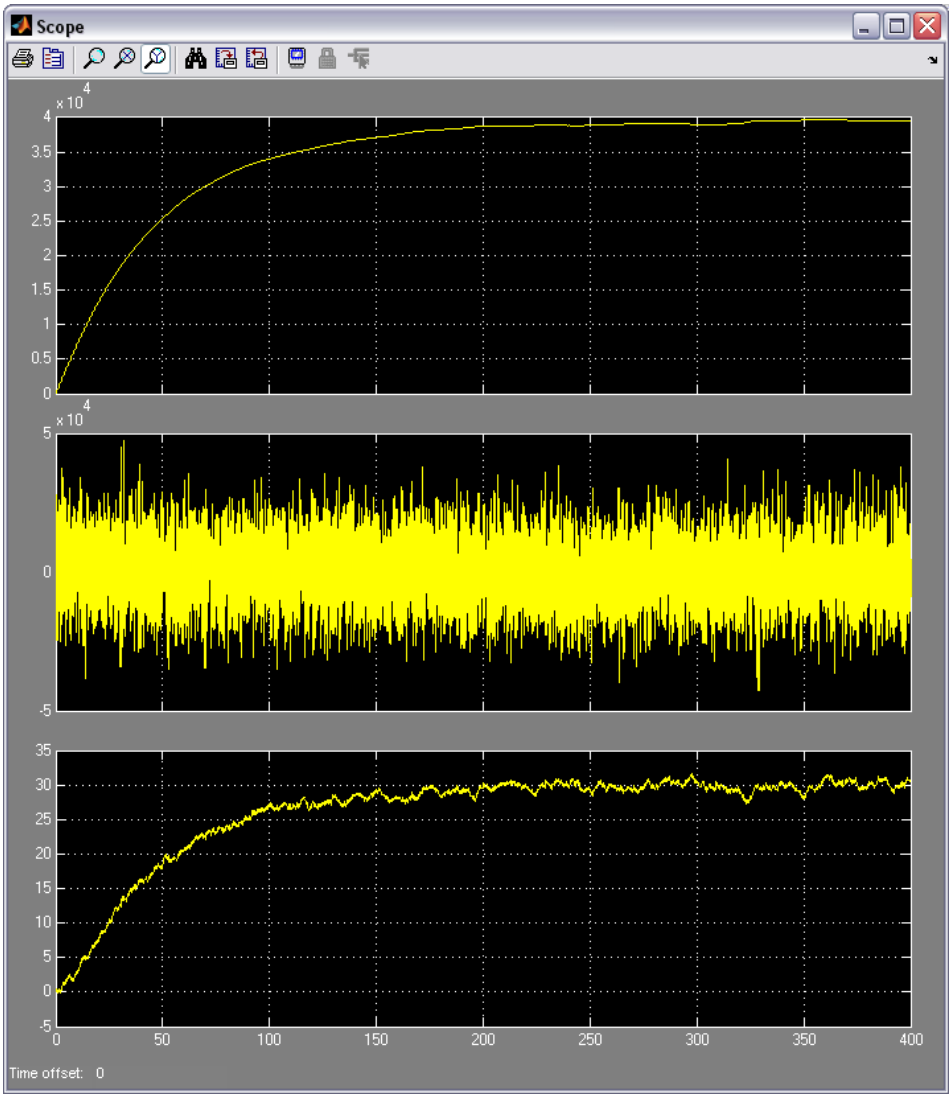
$$\text{Scaling: } G_2 = 2^{-17}$$

The gain chosen close to 1, but not too close as a value too large might result in rounding errors and cause instabillity. The scaling was also selected to ease the implementation in hardware, it's equal to a right shift operation of 17 bits. The results are shown in Fig. 4.5, where the top signal is the clock offset, middle signal is the total error signal (jitter + offset) and bottom signal is the lowpass filtered signal used to control the STC clock. The output stabilizes after about 3 minutes, and the jitter is suppressed to approximately ±60 Hz. The top signal reveals that there will be a static offset by approximately 1.48 ms between the

local PCR counter and received PCR values once the output has stabilized, the significance of this offset is explained in Section 4.2.6 about PLL errors.

Network jitter is modelled as noise in this simulation, but this is not the case in real environment (Section 3.1) so it might be necessary to adjust these values for optimal performance in a real device.

**Figure 4.5.** Result of lowpass-filter simulation



### 4.2.6 PLL errors

The constant average offset between the input and output signal in a PLL is called static phase offset, which would be equal to the stationary error in a P controller. In the case of our system, this will be a constant time offset between the local PCR counter and the received PCR values. According to the simulation example in Section 4.2.5, this error is in the order of a few milliseconds (1.48 ms in that particular case) and should easily be covered by the jitter buffer (which is in the order of hundreds of milliseconds as will be described in Section 4.3.1) [7].

## 4.3 Buffer handling

The current way of handling CBR streams in Chameleon is to make the buffer fill level stay constant and near the middle of the buffer; if the buffer is slowly increases or decreases in fill level then the read speed is adjusted. This way will not work anymore because when VBR is used, the buffer fill rate will vary with the bit rate.

To solve this, the output rate from the buffer must depend on the timing information of the stream rather than the buffer fill level. This timing information is available as PCR timestamps and the bit rate, which concludes that the buffer delay should be constant in time, rather than in size (which was the same thing in the case of CBR streams).

### 4.3.1 Buffer size

First, the max buffer size should be determined. According to the method proposed by the SMPTE document [8] (page 14), the buffer size should be:

$$2 \cdot \text{PCR interval} + \text{Packet Delay Variation} = \text{Minimum Buffer Size}$$

The exact PCR interval will vary depending on the stream, since no exact interval is specified by the standards. The maximum allowed period between each PCR value is 100 ms according to the TS standard, altough the DVB standard recommends a period of no longer than 40 ms. It is explained in Section 4.1.3 that at least two PCR packets are needed in the buffer. The maximum allowed jitter on an IPTV stream according to the TS standard is ±200 ms, although typical jitter values will usually be lower than that in most networks. Taking this in account, a suggested buffer size for universal streams is:

$$2 \cdot 100\text{ms} + 200\text{ms} = 400\text{ms}$$

This will cause a delay of 400 ms from input to output of the dejitter unit. Alternatively, if only DVB compatible streams will be used:

$$2 \cdot 40\text{ms} + 200\text{ms} = 280\text{ms}$$

It would also be possible to implement some dynamic control to adjust the delay according a measured PCR interval in order to minimize the jitter buffer delay, altough that is out of scope for this thesis.

### 4.3.2 Playback control

The process of playing back packets from the buffer is illustrated in the flow chart in Fig. 4.6. To keep the packet stream delayed in the jitter buffer, a delayed version of the local PCR counter is calculated and used to control playback. This is done by subtracting the desired delay from the local PCR counter: for a delay of 400 ms, $0.4 \cdot 27,000,000 = 10,800,000$ STC cycles are subtracted from the local PCR counter.

Each packet is then sequentially read from the jitter buffer and scheduled for playback according to the following scheme:

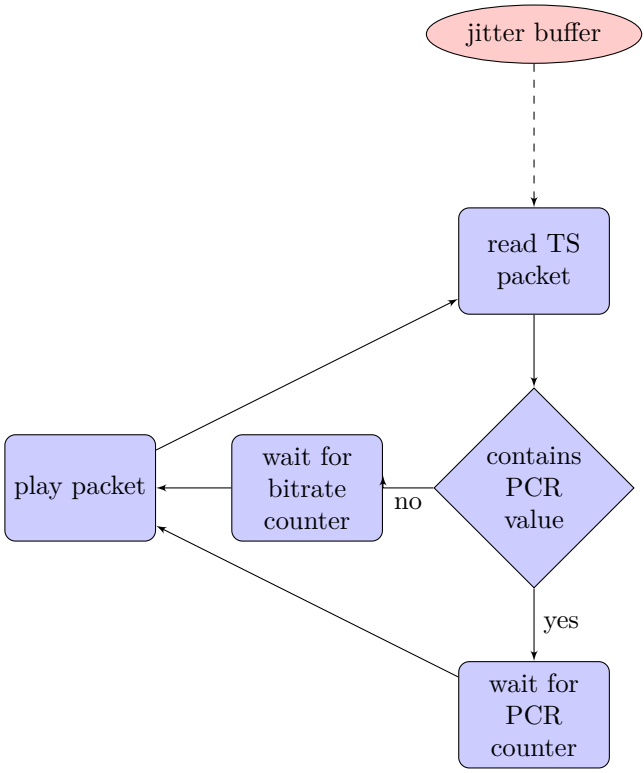
- If the packet contains a PCR value, hold it until the delayed local PCR counter has reached the packet's PCR value.
- If the packet does not contain a PCR value, a bit rate counter is used to trigger the playback of each packet.

The bit rate values are calculated before the packets are stored in the jitter buffer, according to Section 4.1.3. These bit rate values should be stored in a buffer so that each PCR value in the jitter buffer will have an associated bit rate value available. The jitter buffer cannot be utilized for the bit rate information due to the PCR interval delay involved in the calculation, a separate buffer must be used.

## 4.4 Using a CPU

One suggestion by A2B was to include a soft CPU on the FPGA and run the controller as software instead of hardware. The proposed CPU is called PicoBlaze, which is a free 8-bit RISC processor by Xilinx designed for use with their FPGA and CPLD products. The size of that CPU is small, only 96 slices in the Spartan-3 FPGA + one or several block RAMs to store the software. The plan is not to do an implementation in software too, but to compare with the hardware implementation and analyze possible gains. Another option that will be considered is to use the onboard CPU available on the Chameleon board. This will be done in Chapter 5.

**Figure 4.6.** Playing back data from jitter buffer



# Chapter 5

# Implementation and testing

This chapter describes how the implementation of the design was integrated into A2B's system.

## 5.1 Existing design

The TS stream receiver part in Chameleon is implemented in hardware and written in a mix of VHDL and Verilog.

VBR stream capabilities was added to their current design, which initially could receive CBR streams. The method of receiving CBR was implemented as described earlier: incoming TS stream was stored in a buffer (1MB in size), up to half before playback started. Then the buffer fill level was used to adjust playback speed (Section 3.2).

The basic layout is illustrated in Fig. 5.1. The device is capable of receiving UDP or RTP streams (without utilizing the RTP information), these are disassembled to TS packets and stored into a FIFO buffer where it will later be written down to the jitter buffer. Another module is responsible for reading the packets back to a jitter free TS stream. The jitter buffer includes control for automatically generating addresses and behaves basically like a large FIFO memory.

**Figure 5.1.** Overview of existing design

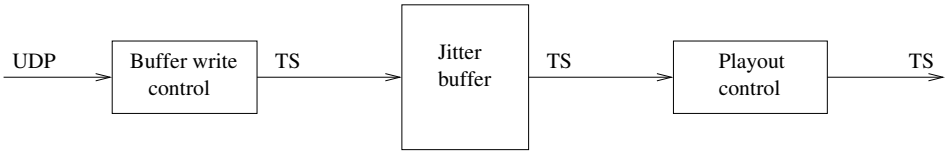
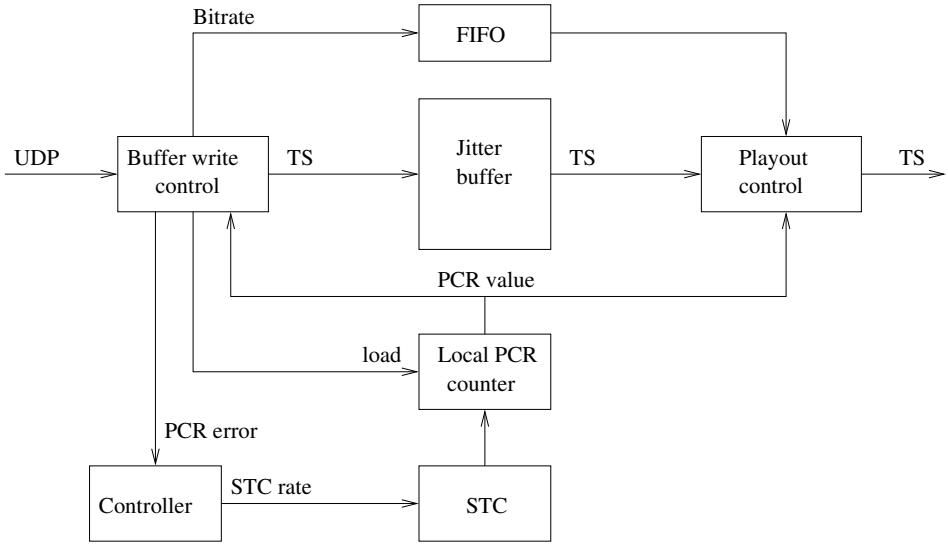


**Figure 5.2.** Overview of new design




## 5.2 Design overview

The implementation will extend the original design with the necessary changes to allow handling of VBR streams. Large parts of the old design was reused: such as the part receiving UDP/RTP streams, low level jitter buffer handling and the control which transports the recovered TS stream. Fig. 5.2 shows an overview of the extended design. The new blocks are explained more in detail following in this chapter.

### 5.2.1 Design of clock synchronization

The following will be done to add clock synchronization:

- Create a Numerical Controlled Oscillator (NCO) that will act as the local STC clock.
- Add a local PCR counter, driven by that STC.
- For each received datagram, save a local PCR timestamp.
- If the datagram contains a TS-packet with a PCR timestamp, use the previous PCR value to interpolate a timestamp for the PCR TS-packet.
- Take the difference between the received TS PCR and the local PCR counter. The difference is used as input to a controller which is used to control the local clock NCO.

### 5.2.2 Design of buffer playback

The following will be done to control buffer playback:

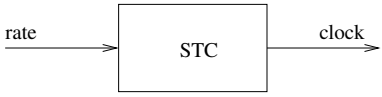
- Read a TS packet from the buffer into a temporary buffer.
- If the TS packet contains a PCR value, keep it until it the delay has passed since it was inserted into the buffer.
- When the PCR TS packet is played, fetch a bit rate from the FIFO.
- If the TS packet does not contain a PCR value, play packet according to the bit rate.

## 5.3 Design details

The details of the design are explained in the following chapter.

### 5.3.1 STC

**Figure 5.3.** STC block




A Numerical Controlled Oscillator (NCO) is used to create the adjustable STC. The NCO operates on the ratio between the input and output values and creates an output signal which can be used as a new clock (as an enable signal with the input clock). A 55 MHz clock is available in this design and is used as the input clock rate. The output clock rate is adjustable around 27 MHz and comes from the STC controller. Fig. 5.3 illustrates the block diagram.

### 5.3.2 STC controller

**Figure 5.4.** STC controller block

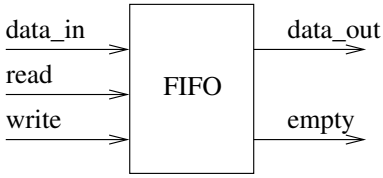

The STC controller consists of a low-pass filter and a P-controller, which essentially forms a PLL together with the input. The low-pass filter is implemented according to the details in Section 4.2.4. While it might be necessary to provide some extra marigin in order to be sure that all network conditions can be handled,

this will be good enough for testing purposes in A2B's network where the jitter is typically within 10–20 ms.

The input to this module is the difference between the local PCR counter and the PCR values extracted from TS packets, the output is the adjusted 27 MHz clock rate which is connected to the STC clock NCO.

### 5.3.3 Bit rate FIFO

**Figure 5.5.** Bit rate FIFO block

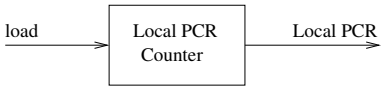



The bit rate FIFO is used to buffer the bit rate information associated with each PCR packet. For each PCR packet in the jitter buffer, there will be an associated bitrate value in the bitrate FIFO.

The input/output to this module is the data signals and a few control signals.

### 5.3.4 Local PCR counter

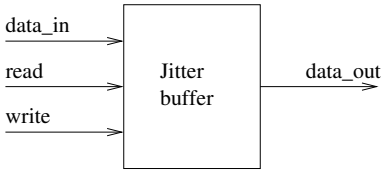
**Figure 5.6.** PCR counter block




The local PCR counter is a 42 bit counter that wraps around at $(2^{33}-1)\cdot 300+299$. It has an option to load the counter with a value from the buffer write control module, used to synchronize the counter with the transmitter's PCR counter.

### 5.3.5 Jitter buffer

**Figure 5.7.** Jitter buffer block

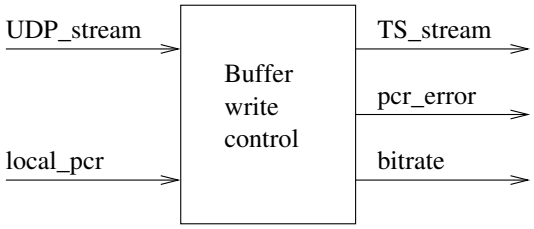


The jitter buffer consists of a DDR memory, interfaced by a memory controller. A single memory is shared among multiple instances of the buffer reader/writer and contains a scheduler. The already existing module for this memory automatically generates addresses so this behaves pretty much like a FIFO.

The most important signals are used to read/write data.

### 5.3.6 Buffer write control

**Figure 5.8.** Buffer write control block




The buffer control module receives the UDP-packets from the Ethernet device, extracts the TS-packets and writes these into the jitter buffer. The following additions was made to this module:

- Assign timestamps to incoming UDP-packets
- Extract PCR values from TS-packets
- Calculate estimated TS-packet arrival time
- Calculate bit rate between PCR-packets

The packets are inspected on the fly as they are written down to the jitter buffer. In order to properly determine and extract the PCR value from a TS-packet, the following conditions must be met in the TS packet header (Section 2.3):

- The adaption field bit must be set
- The adaption field size must be larger than zero
- The PCR value field must be set

When these conditions is met, the PCR value in that packet is extracted and an arrival time for that packet is calculated. Then the difference between these values are calculated and assigned to the signal called pcr_error.

#### Bitrate calculation

One practical way of calculating the bitrate is to calculate the number of cycles between each packet rather than number of bits/second, as this will both make the calculation simpler and aid when playing the TS packets back from the jitter buffer. The following forumla can be used for that:

$$Cycles = \frac{PCR_n - PCR_{n-1}}{Packets}$$

*Packets* is the number of packets between both PCR packets. Xilinx has an IP library called Xilinx Core Generator which provides some common function blocks that can be used to speed up the design process, a division module from this library was used to perform the division.

Once the bitrate is known, it is sent to the bitrate FIFO. The bitrate is also used to interpolate the estimated PCR packet arrival time in 27 MHz cycles:

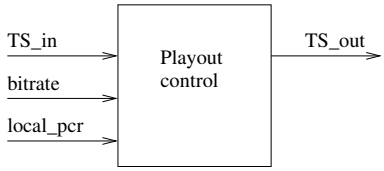
$$PCR_t = DG_t - (Cycles \cdot NumPkts)$$

$DG_t$ is the datagram arrival time according to the local PCR counter, $NumPkts$ is the number of TS packets following the PCR packet in the received datagram and $Cycles$ is the bitrate.

The $PCR_t$ timestamp is then subtracted from the packet PCR to form the error value, and that value is sent to the STC controller.

### 5.3.7 Playout control

**Figure 5.9.** Playout control block



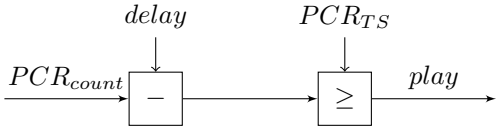

This module reads the TS stream back from the jitter buffer and maintains the rate at which packets should be played back, this module implements the functionality described in Fig. 4.6 (in Section 4.3.2).

To do this properly, it contains a packet buffer that fits one packet (188 bytes in size) and this buffer is always filled with one packet. This is done to minimize the delay between when a packet should be played and when the packet is actually read from the jitter buffer, this is because the jitter buffer is shared and might have an unknown delay. Another reason is that PCR values are used to control playback; so every packet must be inspected for PCR values, and is done at the buffer stage.

When a PCR packet is found, the PCR value is compared against the local PCR counter minus the delay of the buffer. At the same time, a new bitrate value

is also fetched from the bitrate FIFO. This bitrate value is used as input to a 27 MHz cycle counter which controls the playback rate of each packet between the PCR packets.

**Figure 5.10.** Play control signal generation




## 5.4 Implementation results

Because the VBR controller is only a small part of a larger design, the exact FPGA utiliziation used by this implementation is not known, and the utilization displayed in Table 5.1 includes additional hardware used in these modules.

**Table 5.1.** FPGA utilization, new VBR receiver

| Module | Slice | FF | LUT | DistRAM | SRL | DSP48 | RAMB |
|---|---|---|---|---|---|---|---|
| Buffer | 550 | 1630 | 1750 | 0 | 99 | 13 | 18 |
| Playout | 299 | 546 | 452 | 288 | 0 | 8 | 0 |

The division unit is the largest block used by the control loop and is displayed separately to make it easier to compare with a software implementation, Table 5.2. This block is included in the Buffer module in Table 5.1. For comparison, the original CBR receiver design is also presented in Table 5.3.

**Table 5.2.** FPGA utilization, division block

| Slice | FF | LUT | DistRAM | SRL | DSP48 | RAMB |
|---|---|---|---|---|---|---|
| 143 | 709 | 363 | 0 | 99 | 9 | 1 |

One detail that was evaluated were if a CPU could be used to optimize the control loop. My conclusion of the thesis is that the process of receiving and recovering a VBR stream is a straight forward operation that can be done with little effort in hardware.

When compared to the proposed PicoBlaze CPU, the division unit itself is currently a little larger than the CPU. But it is possible to use alternative methods for the division unit as the speed is not critical, the only requirement is that a division result is provided within a PCR interval. The current Xilinx division unit provides a result within a few cycles, so the size of a division unit can probably be reduced quite a bit by choosing a slower algorithm.

**Table 5.3.** FPGA utilization, original CBR receiver

| Module | Slice | FF | LUT | DistRAM | SRL | DSP48 | RAMB |
|---|---|---|---|---|---|---|---|
| Buffer | 652 | 1503 | 1457 | 0 | 15 | 0 | 13 |
| Playout | 172 | 128 | 432 | 0 | 0 | 0 | 1 |

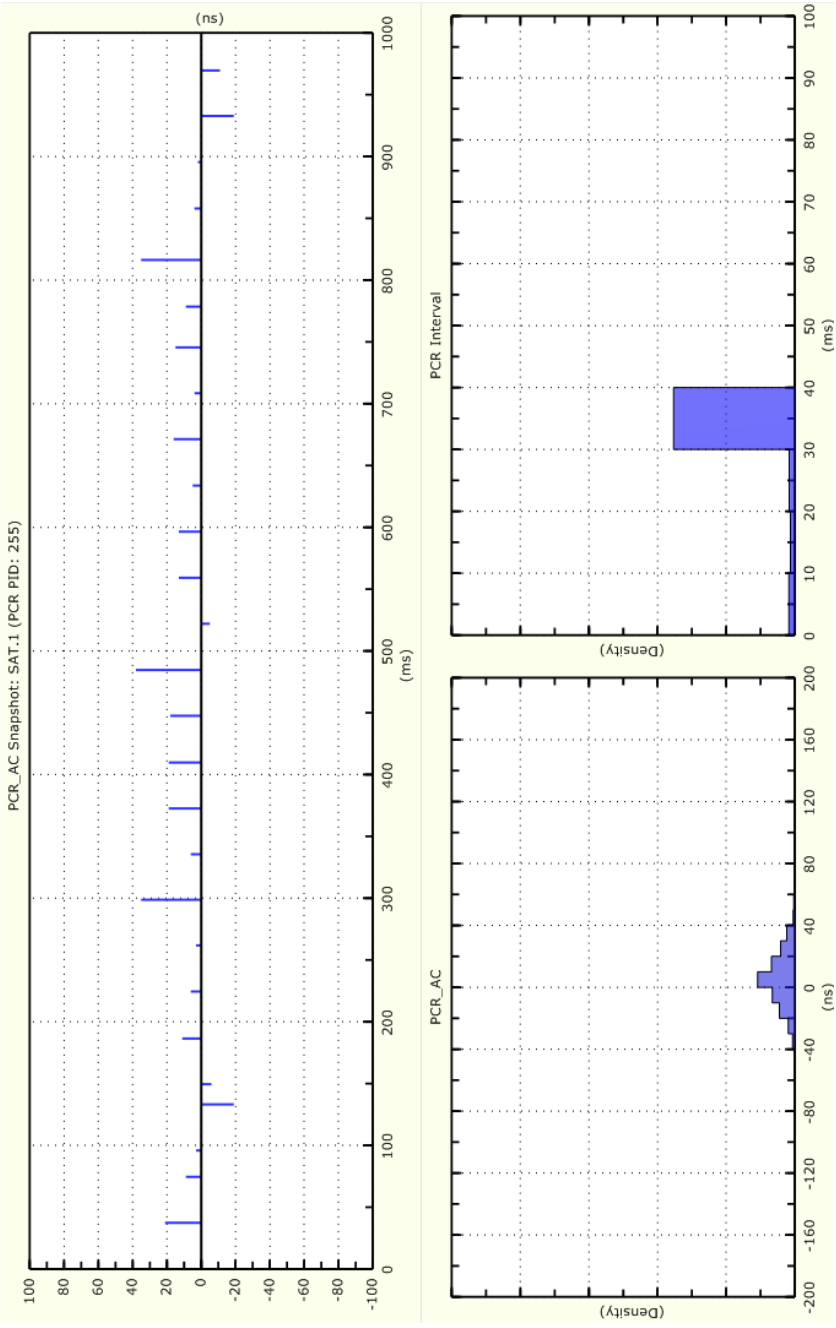
However there are other possible gains by using a CPU solution, such as the option to use more advanced control algorithms or beign able to tune the control parameters depending on the environment. Such possibilities was not considered when designing the hardware implementation.

## 5.5 Measurements

Measurements was made with StreamXpert with the following setup: VBR IP stream in to the device and a CBR ASI stream out from the device.

A screenshot with the results from the program is shown in Fig. 5.11. The top part displays the interval between the PCR packets, as well as the PCR accuracy level (which is ±500 ns). The bottom left part displays the PCR jitter distribution, the results should be mostly around the middle bar, some slight deviation is acceptable. The bottom right part displays the PCR interval distribution, the largest bar should be just below 40 ms which is the recommended DVB interval. Some packets below this interval may occur and is no indication of problem.

**Figure 5.11.** Measurement from StreamXpert



# Chapter 6

# Summary, conclusions and future work

A design for receiving VBR streams was created and implementated for one of A2B's hardware platforms. To properly do that, a controlling algorithm was designed to restore the transmitter's clock in the receiver as well as removing the network jitter from the incoming IPTS stream. This controller consists of a PLL with properties tuned to the TS limitations according to the DVB specifications.

The conclusion is that the clock synchronization can be made without too much resource demands on the FPGA.

### Using a CPU

One part of the task was to investigate if there was anything to gain by using a CPU, and my conclusion is that the controlling part can be made very small in hardware by utilizing an IIR lowpass filter, and with a better division algorithm, there would not be much to gain by embedding a CPU. Utilizing the existing CPU on the same board however would probably save some resources, as the communication between the FPGA and CPU would require only low bandwidth (i.e. the PCR timestamps to the CPU and the clock rate back to the FPGA). This might also help when scaling the design to handle several streams.

### Using RTP

The RTP protocol was also considered but not used, as while it provides an extra layer of timestamps that could improve performance in the STC controller, it would not save any resources in the FPGA. This is because all the existing logic would still be necessary to handle the PCR timestamps as the PCR values indicates changes in the bit rate, information that is not available in the RTP header itself.

## 6.1 Future work

More testing should be done to verify the jitter suppression and possibly adjust the lowpass filter constants if needed, especially on very high jitter networks close to the tolerance limit. Beside that, a few details were discovered during the project which needs some attention:

- When using a very low bit rate stream, there is a possibility that two PCR packets may be sent in the same datagram. This was not considered in the design and will not work properly because the receiver expects at most one PCR packet in a single datagram, so modification is needed for this to be handled correctly.
- MPTS streams does not work well with the current design. The reason is that in MPTS, multiple PIDs with separate STC clocks will be available in the same stream. A possible work-around for that case would be to keep and use the old implementation for the cases where MPTS is received, and this would work because MPTS is practically never VBR. Another option would be to let the regulator lock on one specific PID in the stream and let the bitrate of the entire MPTS stream be included in that specific PID, which also would work because MPTS normally is CBR.

# Bibliography


[1] A2B electronics ab, 2012.

[2] Official MPEG home page, 2012.

[3] DekTec, http://www.bjpace.com.cn/DEKTEC/manuals/DTC-320M.pdf. *Manual for DTC-320 StreamXpert Software*, february 2005.

[4] DVB. Digital video broadcasting (DVB); Measurement guidelines for dvb systems. Technical report, ETSI, 2001.

[5] Inc IneoQuest Technologies. Media delivery index, 2006.

[6] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Infrastructure of audiovisual services, transmission multiplexing and synchronization. Technical report, SMPTE, http://mumudvb.braice.net/mumudrupal/sites/default/files/T-REC-H.222.0-200605-I!!PDF-E.pdf, 2006.

[7] Prof S. Long. Phase locked loop circuits, April 2005.

[8] The Society of Motion Picture and Television Engineers. Unidirectional transport of variable bit rate MPEG-2 transport streams on IP networks. Technical report, SMPTE, 2010.

[9] Colin Perkins. *Rtp: Audio and Video for the Internet*. Addison-Wesley, 1 edition, 2003. ISBN 0-201-52983-1.

[10] J. Postel. RFC 768: User datagram protocol. Technical report, Internet Engineering Task Force, http://tools.ietf.org/html/rfc768, August 1980.

[11] J. Postel. RFC 791: Internet protocol. Technical report, Internet Engineering Task Force, http://tools.ietf.org/html/rfc791, September 1981.

[12] DVB Project. DVB - digital video broadcasting, April 2012.

## Upphovsrätt



## Copyright