Bachelor Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2016

Use of a multiplexer to get multiple streams through a limited interface

Encapsulation of digital video broadcasting streams

Atle Werin

LINKÖPING
UNIVERSITY

Bachelor Thesis in Electrical Engineering

Atle Werin

LiTH-ISY-EX-ET-16/0460--SE

Supervisors:
Patrik Sandström
WiSi

Examiner:
Oscar Gustafsson
ISY, Linköping University

Division of Computer Engineering
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In digital video broadcasting, sometimes many sources are used. When handling this broadcast a problem is a limited interface that has a fixed number to input channels but overcapacity in data transfer rate. To be able to connect more inputs to the interface a protocol that lets the user send more than one channel on a connection is needed. The important part for the protocol is that it keeps the input equal to the output both in timing and in what data is sent. These are done by encapsulating the data and use a header containing information for recreating the input. To solve the timing constraint dynamic buffer are used that makes all data evenly delayed. To validate the functionality of the protocol a test designed is implemented in VHDL and simulated.

Acknowledgments

A special thanks to WiSi that let me do my work there especially to Patrik Sandrstöm for supervising me and to Herman Molinder for feedback on the report.

Contents

List of Figures

List of Tables

Notation

Abbreviation/ Acronym	**Meaning**	**Explanation**	**Context**
BB-frame	Base Band-frame	A format used for sending data in DVB-C2/T2/S2 looks a lot like GSE but bigger size	2.1.1
DVB-C2	Digital Video Broadcasting - Cable - Second Generation	Digital television broadcast standard for cable transmissions	2.1
DVB-S2	Digital Video Broadcasting - Satellite - Second Generation	Digital television broadcast standard for satellite transmissions	2.1
DVB-T2	Digital Video Broadcasting - Terrestrial - Second Generation	Digital television broadcast standard for terrestrial transmissions	2.1
GFPS	Generic Fixed-length Packetized Stream	A format for transporting generic data in packets of a fixed size	2.2.3
GS	Generic Stream	A name for GSE,GSC and GFPS	2.1
GSE	Generic Stream Encapsulation	A format for transporting generic data in packet format	2.2.1

GSC or GCS	Generic Continues Stream	A stream of continues data and not in package	2.2.2
msg	Message	A single unit of data	4.2.4
TS	Transport Stream	A format used for transporting TV-canal package. Can de named MPEG-TS to	2.2.4
	Frame	A unit of data made to be sent	
	Fragment	Part of a packet, the product from fragmenting a packet	
	Packet	A header and following data	
	Pay load	The data to be sent for example the data in a packet	

1 Introduction

1.1 Background

The company WiSi have a product Chameleon that is used for different formats of digital media and transforms them to other formats. For this project, the Chameleon has a removable tuner board with two input ports, tuners and demodulators. The input is connected to a FPGA with two connections. In some use cases, the input data rate is much lower than the connection speed meaning there is a lot of free bandwidth. To use this in a better way a possibility is to change the tuner board to one with more input ports. The connections on the FPGA is locked to two ports meaning the new ports will have to be muxed together. Another interesting aspect is that new formats are coming and making a general solution that can take in not just the formats used today is interesting.

1.2 Motivation

A product has a predefined interface that is not easy to change. However, the interface has higher bandwidth than is needed for its current application. This gives a possibility to have more inputs if only all inputs can be sent over this single interface, which can be one by a protocol. It is important that after the interface the inputs are identical to the recived data. To solve this, an idea is to put a small FPGA before the interface and have it packetize the data with important information to be able to recreate it to its original state. Then the data are sent through the interface and the receiver implements an algorithm that returns the data to the original state. To make a general solution that is easy to test and have a well-defined layout a layer view is used. This means splitting up the process in different layers that are unaware of how the other layers work resulting in that each layer can be implemented and tested separately.

1.3 Purpose

Because the FPGA inside a product already have a fully defined interface but want to be able to take more input streams a solution that do not need any changes in the interface is needed. In the future new formats of input will be interesting meaning that it is desirable that the solution is general purpose and not limited to just the standards used today.

1.4 Problem statements

Questions that are to be answered in this work.

- How is it possible to mux together TS (transport streams) from multiple channels to one single channel send the TS by a serial interface Figure 0 and then demux them back to their original state again?

- Is it possible to take the solution for TS and apply it for other formats like GS (Generic Stream) and if not how to make it possible?

- What design parameters are important and how do they impact the protocol?

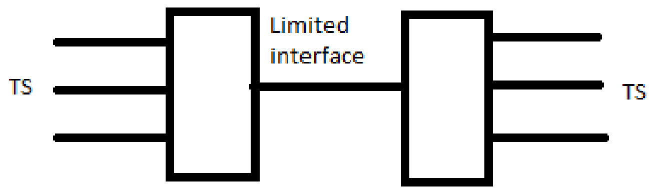


Figure 1 – Limited interface

1.5 Limitations

No implementation of other formats than TS will be done mainly because WiSi is only using TS at the moment. Nevertheless, other formats will be investigated and will be used in the protocol design.

The second generation DVB standards are backward compatible and because of this the first generation is not needed because it does not give extra functionality.

2 Theory

To be able to know the environment of the system and the requirements for it knowledge from different sources are needed and will be discussed in the theory chapter.

2.1 Digital Video Broadcasting

To get an understanding for the environment of the system Digital Video Broadcasting(DVB) is studied. Digital Video Broadcasting is a series of standards developed for digital media broadcasting[1]. The main application of the system will have a source of some form of DVB transmission. It is not the only standard but the one that will be used in this work because it is the European standard.

DVB-T2 is the second generation standard for terrestrial transmissions[1], DVB-S2 is the standard for satellite transmissions[2] and DVB-C2 is the standard for digital television transmissions over cable[3]. All of them send other TS or GS formats of data. The different DVB standards are implemented a little bit differently and have different limitations to work for different signal mediums. To get a TS or GS from any of the transmission standards the signals are needed to be demodulated.

2.1.1 Base Band-frame

A Base Band-frame (BB-frame) is used in DVB for sending and encapsulating data. BB-frames have a 10 byte header and up to 7264 bytes payload[1]. The payload is one or more of TS, GSE, GPFS or GCS[1] like in Figure 1. Continuous streams can be fragmented and fragments of the stream can be sent in different BB-frames but more than one packet can also be sent in one BB-frame. The BB-frame then contains information to put the stream back into one piece.

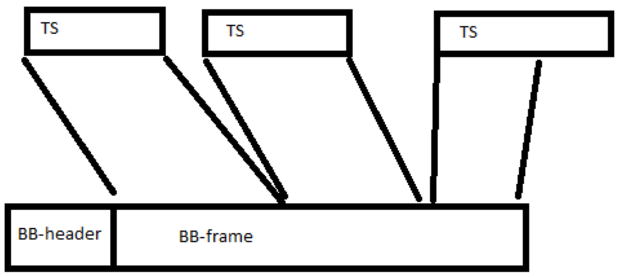


Figure 2 - TS packets in a BB-frame

2.1.2 DVB overview

To send data with satellite DVB-S2 is used. It is a way of taking data in form of TS or GS and encapsulates it to send in a satellite channel. It has three goals, maximise transmission performance, high degree of freedom and not too complex receivers [4]. Performance is gained by two levels of encapsulation. The first layer is a BB-frame and takes the TS or GS and packetizes them. After that the BB-frames are Forward error correction (FEC) encoded symbol mapped by using QPSK, 8PSK, 16PSK or 32PSK and put in a physical layer frame (PL-frame) that is sent. The PL-frame makes the transmission sturdy and makes it tolerant to noise and bit errors. To give freedom the BB-frames can take more or less any kind of data and packetize it and the signals are designed to work with any satellite transponder [4].

In the case of DVB-C2 and T2, procedure similar to DVB-S2 is used but in the last part the data is sent as orthogonal frequency-division multiplexing (OFDM) symbols instead of QPSK, 8PSK, 16PSK or 32PSK[3][1]. The OFDM symbols are sent on multiple orthogonal carrier frequencies. The signal is created from the discrete Fourier transform of the signal and then split it up between frequencies. OFDM lets the receiver compensate for distortion from the transition channel.

2.2 Formats

There are many formats for streaming digital television and they address problems with similarities to the problem statement. The generic formats are more of concepts of how streaming can be done and sets up limitations but have some freedom in how to implement them. The input streams typically come from a demodulator that has received some kind of broadcasting data e.g. DVB-S2, and demodulated it to e.g. TS, to be able to create a protocol that can take different formats an understanding of similarities and differences between t hem is needed.

2.2.1 Generic Stream Encapsulation

A Generic Stream Encapsulation (GSE) is a streaming format of packages. The packages have predefined length but different packets can have different length. Every GSE packet has minimum of 2 bytes header containing information about the packet and a GSE payload containing the data that is to be sent. The two first bits of the header tell if the start and /or the end of the data in the GSE payload are present in this packet. Next two bits tell if any extra header data is added. The last part of the mandatory GSE header has a 12bit field for length of GSE packet (=length GSE header + GSE payload). Frag ID is added if not both the start and end bits are set to one and is used to identify all packets that have been fragmented from one source. If the start bit is one but not end bit a Total length field is added that tells the length until the end part of the fragments. The GSE packet is limited to a maximum of 4096 bytes but supports fragmentation of the data that is sent, meaning the data can be spread out between different GSE packets. In Figure 2 two unfragmented data packets are received and then on that needs to be fragmented. With the fragmentation a data block of maximum of 65 536 bytes is supported.[5]

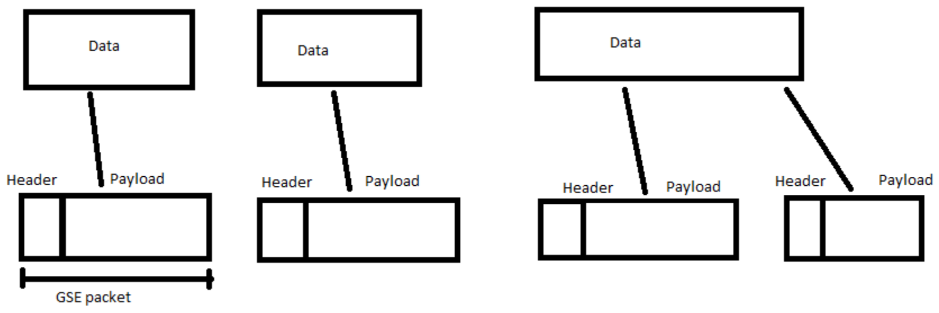


Figure 3 - Encapsulation of data in GSE packets

2.2.2 Generic Continuous Stream

A Generic Continuous Stream (GCS) is use to define a constant stream of data or something that sends more than 64 Kbits in a single packet. Because it is continuous it does not have a need for a beginning or a header. GCS will need to be split into limited sized packets to be sent to work like the other formats. Using any of the DVB standards this will be done by fragmenting the stream to packets that can be sent. [3]

2.2.3 Generic Fixed-length Packetized Stream

Generic Fixed-length Packetized Stream (GFPS) is a concept of streams that have a predefined length that is the same for all packages. But in DVB-T2 and DVB-C2 it is mostly used for compatibility with older standards and GSE is expected to be used instead.[3]

2.2.4 Transport Stream

Transport Stream (TS) is a special case of a GFPS as it has a fixed packet size its own version of header. It has a total length of 188 bytes of which 184 bytes are used for the payload and 4 bytes for the header. Figure 4 shows the layout of the header for TS.

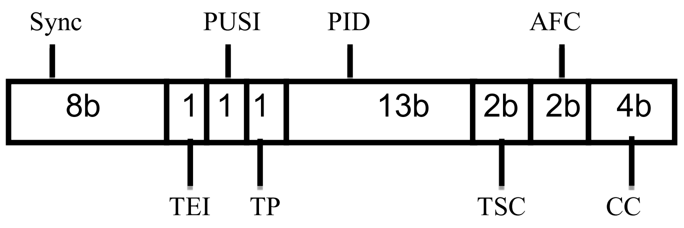


Figure 4 - TS header

The parts mostly important for this work are:

- The Sync bytes are 0x47 and are used for detecting the beginning of a packet.
- Payload unit start indicator (PUSI) is an indicator used when data is fragmented between more then on TS packet and tells if this is the beginning of the fragmented data or not.
- PID (Packet Identifier) tells what kind of data is in the packet up to the user to assign codes.
- Continuity counter (CC) is used for TS packets that have the same PID to see if any packet is missing. It is incremented by one for each packet of same PID.[6]

PUSI, PID and CC are used to get functionality for sending fragmented payloads. PID tells what data is in the payload and all fragments need to have same PID. PUSI tells what the first part of the fragment is. One can use PID to find all packets from a fragment and PUSI to find the first of them then use CC to find the order between the packets. All packets from a fragment have a CC value one more than the previous packet. But in case CC is on its last number the next CC is zero.

2.3 Jitter

Jitter is a phenomena that occurs when different messages gets different delays compared to each other. According to [7] a de-jitter buffer is a good solution to get rid of jitter. The buffer is supposed to detect spikes and increase the buffer size during them and then go back to a normal run state wear it buffer a time dependent on previous buffer time and the delay of the current data. In its normal state the total delay is calculated by Eq. 1$d_i = 0{,}125 \times n_i + 0{,}875 \times d_{i-1}$. To get the time for the data in the de-jitter buffer it is just to take total time in system minus time in system before de-jitter buffer. This formula will give a buffer time that slowly depends on other delays in the system but will be to slow to solve bursts of data.

d_i=total time in system (delay)
n_i=time in system before de-jitter buffer(delay-de-jitter buffer)

$$d_i = 0{,}125 \times n_i + 0{,}875 \times d_{i-1}$$

Eq. 1

2.4 Layer model

A layer model is the OSI layer model contain seven layers that is used in TCP/IP communication. Each layer has a peer that it communicates with. The peer is on the same level in the layer and is expected to receive the same data sequence like the peer sent including header. [8]

3 Method

3.1 Introduction

To be able to make an effective protocol a plan of how to make it is needed. The design of the protocol and the implementation differs a lot resulting in that it is best to use different methods for solving each of them. The protocol is general purpose and need to be easy adaptable but the implementation on the other hand is for a single purpose only and do not need that flexibility.

3.2 Protocol

The problem is analysed by breaking it up to smaller parts and each part is addressed separately. From the parts, requirements and design parameters are found and used to get a better understanding of the problem and a foundation of the solution. To solve the requirements of the parts, other protocols are analysed and key components from them are extracted. The key components are then investigated and evaluated to find the needed components for the protocol. Then the components are modified to work for the protocol and then put in to the protocol designee.

3.3 FPGA Implementation

Implementation will be done by the layer method resulting in layers will be implemented from the top to bottom. This is because a layer is supposed to be independent of the other layers and functionality will be able to be tested without the lower layers. The code will be written in VHDL and synthesized for use in a Xilinx Spartan 6 FPGA. Each layer will have two sides on input and one output side. Using VHDL will result in that the code will be divided into blocks. A top level of implementation will be made that connects the different layers and contains blocks containing one side of a layer in each block.

4 Design

4.1 Introduction

The design process contains of two parts: a protocol design and an FPGA implementation. The protocol design is about making a general protocol for sending the data through a limited interface and recreating the data on the receiver side.

The implementation part aims to make a prototype in VHDL code for a single application. It uses the resulting protocol from the protocol design to define its functionality. To try to make a general system all of the design is done using a layer method containing three layers, an encapsulation layer, a data link layer and a physical layer shown in Figure 5. Each layer is implemented separately and tested without any other layers. Finally all layers are merged and tested together.

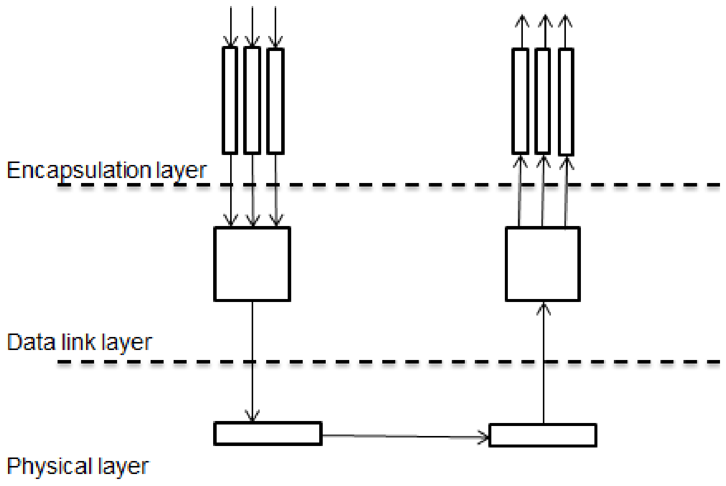


Figure 5 - Layer overview

4.2 Protocol design

The protocol design will contain methods for calculating design parameters and will take discuss process of designing a protocol for how to mux data together and what is needed to send and receive it. The protocol is expected to be general and easy to adapt for different use cases. It is important to be able to adapt the protocol to different design parameters without needing to make a new protocol.

4.2.1 Design parameters

To be able to make a protocol a way of describing the application is needed. This done by using design parameters.

System delay is the time it takes for data to get from the input to the output. It will be calculated as the time from the last bit of a packet is received to the last bit of that packet is sent. A motivation to choose the last bit and not first is in case of fragmentation when it can happen that the first part gets in one early fragment and the last part arrives much later. System delay is used for being able to calculate the systems impact on timing at the output and be able to counter it.

System jitter is a measurement of the difference in delay. It is channel limited and will be calculated in the following way. Maximum delay in the channel subtracted with the minimum delay in the same channel gives the jitter in that channel. A more refined version of the system delay for correcting the timing on the output.

Throughput is the speed of the system. The amounts of data that can pass through in a set time. It is limited to the slowest transfer link in the system but will be define as the amount of bits out from the system during the time it takes to send one packet. Average throughput will need to be equal or greater than the average total of the inputs bit rates for the system to possible work. Thro put is needed for guaranteeing that the system can send all data and that it will not fill up its buffers.

In-Jitter is how much jitter exists in the input and can results in bursts of data. The bit rates of the inputs are constant during long time but in shorter instances the in-jitter can increase them. In-Jitter is used for determine how large buffers are needed and how well the system operates in the worst case.

Channels are how many sources of inputs that exist.

In bit rate specified for every input channel and is the average bit rate over that channel.

Minimum message size is the smallest allowed message to be sent that guarantees that the throw put is not too low. Minimum message size is needed for determine throw put and deciding on fragmentation.

Transmission speed is the internal speed of the system. It is the slowest connection inside the system where the message need to pass.

4.2.2 Priority between input buffers

What buffer that is allowed to send and when might have an impact on the overall performance of the system and will be relevant for calculations of buffer size and system delay. To evaluate what method that gives the most decidable result, one solution is simulating different priority orders. A simulation made in Matlab to evaluate different priority orders is made.

Three different priority orders have been chosen to be simulated. One priority order that always takes the most prioritized channel if it has anything to send if not the second most and going through all channels until it finds anything to send. Round Robin that lets the first channel one send one packet then channel two going through all channels and then restarts at the first again. Lastly Largest buffer first that sends the channel that has the largest input buffer first.

To generate the input for the simulation a randomizing algorithm is used. It creates bursts of data that are supposed to simulate packets and all have an end bit. No input data is generated in the end part of the simulation to give it time to analyse all data and not need to cut away some last data. It is supposed to try to generate more data than the system can handle in a single instance to force the system to select what data to send and have different options. Furthermore, is simulates a busty behaviour of the input sources coming from that BB-frames use bigger packets that can contain more than one packet after the demodulation and then arrive directly after each other forming a short duration over which much data arrive.

During the simulation the random generated data are put in input buffers of arbitrary size that always can store the input. If an end of packet indicator is received it is given a time stamp for calculating the delay. Then the priority algorithm is deciding what buffer is allowed to send and that buffer sends data. When an end of packet indicator is sent, it is counted as being done and the delay is calculated from the time stamp to current time and stored. From the delays from each packet it is possible to get the maximum delay of the system for this input sequence and how much system jitter it has.

4.2.3 Content of the header

The goal is to minimise the size of the header. This can be done in two ways minimising the size of each field in the header and minimise the amount of fields. The header fields size will be pre-determined to be able to solve general requirements on the system. For a specific application a different size might be better but in general the resulting field size will be the best.

Receiver address or identification is needed. This is to be able to send it out on right output port. The size of the channel field limits how many channels are supported and will be chosen accordingly. It needs to be able to contain the design parameter channels.

Size of payload is just needed if different payloads have different size and have to large enough so that it can describe the numbers of bytes in the largest payload.

Time in buffer is useful if a system that compensates for system jitter is to be implemented and needs to be large enough to describe the maximum possible delay to address. The time resolution is based on the maximum possible that will be based on the clock and how much system jitter is allowed. More jitter means lower resolution needed. Time in buffer describes system jitter.

Fragmentation support is done in most input formats. Important parts of its implementation can be found in section 4.2.5 . The need of it is dependent if packets can arrive in wrong order, if jitter exists and if an input packet can have delays inside it. If input throughput is lower than output throughput fragments can have gaps between them. Because the start of packet is used like a flag on the input it is needed on the output so that the start of fragment bit is useful.

Start of packet indicator is a way of solving if a single packet can be split up in two or more different fragments. If more than one packet is to be sent in a single frame a way of distinguish them is needed. This can be done by having a pointer to every start of a packet or by knowing how big the packet is.

4.2.4 Multiple messages in one packet

To send more than one message in a single packet is possible and have a positive effect on throughput. Nevertheless, the header will need more information to be able to distinguish the start of every message it includes and a way to send the delay of every message.

Table 1 - Comparison multiple to single messages

	Multiple message	Single message
Channels	One block	One block
Start of message	For every message a block	One bit
Delay	For every message a block	One block
Size of payload	One block	One block
Number of message	One block	No data
Number of headers	One header	Number of message

Table 1 shows the difference in total header size. Number of message is used for limiting the amounts of fields for delay and start of message when not all are used.

4.2.5 Fragmentation of large input messages

In mostly of the in formats studied fragmentation is supported in different ways. Fragmentation is the concept of dividing a packet into multiple smaller packets. GSE tells if start or end of fragmented data is present in a packet and gives all parts of the fragment a single ID to be able to tell different fragments apart. In TS for comparison there is a start bit but no end bit, an ID and continuity counter to determent the order.

Another aspect is the BB-frames ability to contain more than one frame and is addressed in 4.2.4 .

Pros and cons:

There are two main ways for solving fragmentation. Fragments that have a fixed size similar to TS packets and fragments with dynamic size for example GSE. Fragments with fixed size performs best for known input data where their size can be adapted to the data. If unknown in data is coming it is better to have dynamic size to be able to adapt to small and large inputs. If dynamic frame size is used the header will need to describe its size to let the receiver know how much data to collect. A bigger header will have a negative impact on the systems throughput but because of the dynamic size it is better to have larger packets that gives higher throughput.

The size of the fragments will have an impact on large parts of the system. If a large fragment size is used the delay in the system will be larger and larger buffers will be

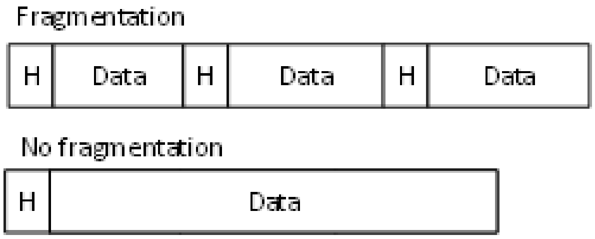


Figure 6 - Comparison between fragmentation or not

needed. This is because it will take more time to send a fragment and the next fragment to send will have to wait longer.

The throughput on the other hand is better the higher fragment size because it is dependent on fragment size compared to header size, see Figure 6, and increasing fragment size have a weary low impact on the header size. Nevertheless, variable fragment size gives the possibility that just small amounts of data will be sent in a packet and giving a bad throughput. A way of avoiding this is to have a minimum fragment size to send if possible. The good thing is it gives a guarantee of a higher minimum throughput. But it can increase the delay because small amounts of data from a silent source never will be sent until more data arrives.

Reasoning about pros and cons:

Deciding to have the fragment size that gives the delay closest to the maximum allowed delay will give the best throughput. The throughput need to be greater than the sum of all average input bitrates if not the buffers will fill up and data will be lost. If limited memory is a constraint an allowed delay from the possible buffer size will need to be compared to the delay constraint and the worst of the two will be used to guarantee that both constraints are met.

If max one message is allowed in a single fragment, fragment size bigger than max message size will never be used. Leading to small messages size giving low throughput and is needed to verify the possibility of getting throughput grater then average input rate of all channels together for the smallest fragment size.

Choosing fragment size can be done by setting up the equations Eq.3 to Eq.7 and compare them. For throughput a minimum fragment size will be decided by looking for the point where throughput is equal to the input using the minimum message size.

In the delay equation the limitation will be the maximum allowed delay giving a max fragment size. The maximum delay can come from a design constraint in delay or from a limitation in buffer size. The maximum fragment size will never need to be larger than maximum message size. If the maximum fragment size is lower than the minimum fragment size this makes the system not possible to implement with the current constraints. If message size is unknown the worst case is to be assumed. This leads to problems implementing the protocol for GSE that have unknown message size where max and min message size is far from each other.

4.2.6 When to send message

Because when a message is received and when it is sent in not synchronized, it is possible to have not received a full message when the channel is allowed to send. The decision the system will need to take is if it is better to send or wait.

If sent there is no risk of a large delay because more than one fragment can be in the buffer next time it is allowed to send and it lowers the requirements on the buffer size for the same reason. The drawback is if just a small amount of data was present or a full message in a size smaller then maximum frame size was to be received a packet extra will be sent. This resulting in more header and lower throughput. If the size of the message is known it can be used to calculate if waiting is a good option or not. Known message size can be used to always use the minimum amount of fragments possible and giving an optimal throughput.

4.2.7 Buffer size

In this application a big buffer size does not affect the system performance in a bad way but as a cost to have a large memory. To guarantee having enough buffer size a buffer size that can store the maximum allowed delay is enough.

Max buffer size= allowed delay × input bit rate

To calculate how much data can be present in the buffer is a more accurate way of determine buffer size. How much data needs to be stored in the buffer is dependent on the in jitter and the message size. In the worst case, throughput for the system is equal to the input rate of the system. If the jitter is zero, a constant input rate can be assumed. The output rate in average for a channel will be equal to its input rate giving a behaviour where a message will be sent before the next arrives. The buffer size will always be less than two messages. Figure 7 has a constant in rate of one and will send out data on even intervals if there is at least five in the buffer. The data out is when the system wants to send out data not when it does. The first data out is missed because not enough data in buffer to send leading to no decrease in buffer.

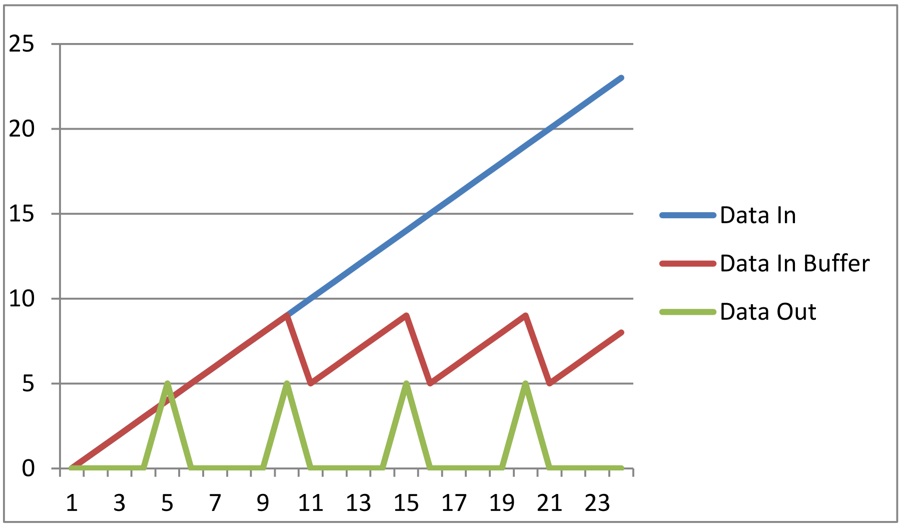


Figure 7 - Used buffer during constant in bitrate

If jitter is present it will result in bursts of data. The worst jitter is a period of maximum silence on the input follow by a burst and then average in rate. If the average in rate is equal to the out rate only the average rate will be handled and the burst will be stored in the buffer. This gives a needed buffer size less than max burst plus two messages. Figure 8 has burst in data at time 1 then constant in rate. In reality, the burst will last a short time and not be momentarily but approximating it in only one-time instance gives a possible worse scenario guaranteeing the system will not be under designed.

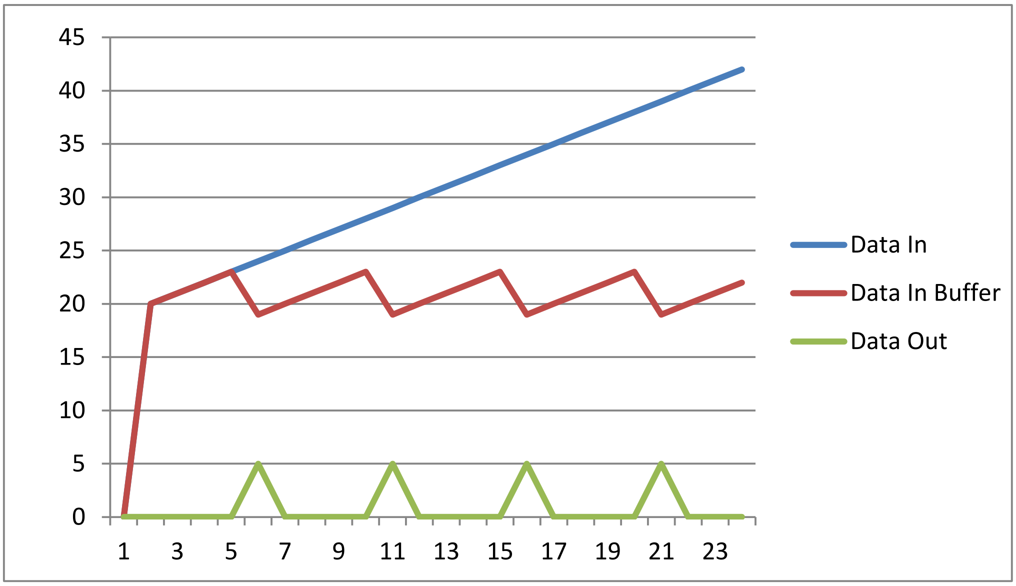


Figure 8 - Used buffer during after burst

4.2.8 System jitter

Solving system jitter by using a de-jitter buffer has a drawback on the systems delay. The de-jitter buffer will need to make all packets evenly delayed. The packet with the lowest delay will have same delay like the one with the highest. If this is achieved the system jitter of the system is zero. The maximum delay can be two things, a design limitation that is not allowed to be overstepped or the worst-case scenario possible of inputs compared to the throw put. In any of the cases a maximum delay is given and the de-jitter buffer can be sized accordingly. One limitation is how good the system is on calculating the delay and in what precision it sends it to the de-jitter buffer. It will not be able to be better than the clock rate and because of that, the minimum jitter is the clock rate and will be seen like zero for computations.

The main reason for not have a dynamic delay is that the max delay is a fixed value for the system. Using it leads to less system jitter than adapting the time in the de-jitter buffer and makes the system more robust to bursts of in-jitter. This because it is always ready for the maximum jitter it can solve.

4.2.9 In-Jitter

The in-jitter will be the limitations in buffer size and when it bursts, the buffers will be filed up and delays will be high. In-Jitter is the maximum allowed relative delay from a stream of packets evenly delayed. The worst scenario is data is coming in the normal bit rate and then maximum jitter. This will give a burst of data.

$$Burst = in\ jitter \times in\ bit\ rate$$

Eq. 2

The burst can be seen like extra data in the buffer compared to a system running on average input. Table 2 is an example with blue data arriving and burst calculated using Eq. 2.

Table 2 - Amount of burst from jitter

normal	1		2		3		4		5		6		7		8		9		10		11		12
Jittered signal	1		2		3	4	5	6	7	8	9	10	11		12								
Burst	0		0		0	1	1	2	2	3	3	4	4		4								

4.3 *FPGA Implementation*

The first step of the implementation is to take the protocol from the protocol design and apply a requirements specification on it. This to make sure the system is possible to implement and to get the design parameters.

4.3.1 Design parameters

From the design specification throughput and fragment size will be decided. The goal will be to get the maximum number of input channels and still have enough throughput and keep the buffer size within the size of the memories. If memory is not a limited recourse, the buffer sizes will be dimensioned by input jitter.

4.3.2 Encapsulation layer

Two parts like in Figure 9 need to be implemented, an in part that takes the input data and store it in a buffer and sends it to the system when the system is ready. The second part is the out part that solves the jitter from the input and reconstructs all signals to make an identical output like the input.

The input needs a buffer that can store the input data and compensate for bursts. For large jitter it will be large and because of that block RAMs will be used for it. It will be a circular buffer that pushes in data and pops it out operating like a FIFO.

Time stamps for when a message is received will be stored in a separate register that will be able to hold one time stamp for every possible message in the buffer. Time stamps will be stored when an end of packet signal is received that can be generated from start of packet and knowing the size of the packet.

A control block will be deciding on when to send and if delay or data is to be sent. Delay will be calculated from the time when it is to be sent minus the time stamp plus the time it takes to send one message.

On the output side a block will analyse the in stream and extract the header to be able to get the delay and message length. It will also remove the header from the data stream and make a new valid that is only when data is sent.

A buffer in same size like in the input will be needed. It will work in much same way like the in buffer but its goal is to compensate for system jitter made by the system.

The delay of a packet will be used to calculate a time stamp when to send out the data from the packet. The time stamp will be set according to the de-jitter buffer design to make all data equally delayed.

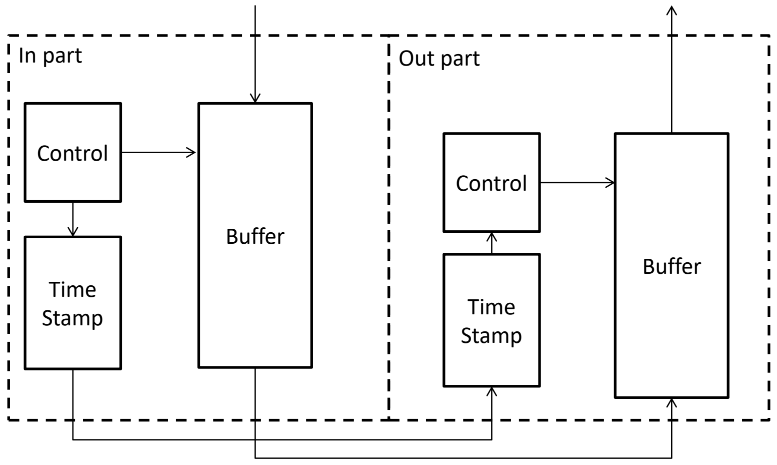


Figure 9 – Encapsulation layer layout

4.3.3 Data link layer

Like in the Encapsulation layer the data link layer have an input side and an output side. The data link layer will multiplex the streams from the encapsulation layer together to a single connection and on the outside back up to multiple connections. It will allow have logic for deciding what buffer to receive and send data to.

The main part of the data link layer is a multiplexer. It will be in the size of the amount of supported channels and will need a control block to decide what channel to send.

The control block in the input side will be a priority protocol decided from 4.2.2 . It will give the different in channels priority and they will respond with a done when they have set data or if they have no data to send.

On the output side, the control block needs to read the header to be able to get to which channel to send the data packet it receives.

4.3.4 Physical layer

The physical layer is the interface between the input side and the output side. It will be specified in the design specification and is a hard ware based part of the system.

4.3.5 Testing the implementation

To be able to validate the system, tests need to be designed. Because VHDL is based on blocks each block can be tested before they are combined. Blocks are to be designed resulting in the need to do small fast tests. This can be done using a simulator and writing a script that makes inputs that gives known outputs.

To test the systems overall performance a test bench is needed. To make input signals a file containing a TS stream can be used and read by the test bench. If the test bench writes down the output of the system in file, the two files can be compared and if same the system worked correctly. Some more aspects to simulate are bursts and calculating the delay between different packages. Bursts can be generated by reading the in file in bursts. The delay is easiest found by comparing Start of Packet (SoP) from the input to the output in a simulator. The important part of the SoP is that the difference between input SoP and output SoP for the same data then comparing this difference between different packets.

5 Results

5.1 Introduction

A protocol for sending data is produced that encapsulates the data with header containing metadata. With the help of the header, the receiving side is able to recreate the signals from input side. The FPGA implementation validates the functionality of the protocol by showing during simulations that the input is equal to the output.

5.2 Protocol design

The protocol design is using the components from the method and evaluate them to decide if needed and how to best make use of them. Following is the results from the evaluation of the components.

Using a layer model encapsulation and jitter is solved in the encapsulation layer and handling that there is more the one channel is done in the data link layer.

Data and a start of packet is received and stored until it is ready to be sent. There are multiple input channels and a Round Robin algorithm decides what one that is allowed to send. A maximum fragment size is needed to be decided by the on implementing the protocol and the fragment size decides if fragmentation will be used or not and if enough data is received to be ready to send. When the system sends a message it is not allowed to contain data from more than on packet and if it contains the first part of the packet it will be indicated in the header. The header will contain how much data it has and an address to what output it is supposed to be sent on. To be able to eliminate relative delay the protocol will make a time stamp every time end of a packet is received. From the time stamp it will calculate the delay time from when the packet is received until it is sent. This information is also put in the header. When a channel is allowed to send it will send if it is ready to send. If it sends it sends a message containing a header followed by data. On the receiving side, the header is decoded and the ID decides to what channel the message will go to. Then the receiver takes the delay from the message and calculates a new timestamp based on the buffer size divided by the transmission speed and the received delay. This timestamp is stored and when the time is equal to the timestamp the data related to the timestamp is sent.

5.2.1 Priority between input buffers

In Appendix B, simulation data from the simulations of different priority orders are presented. It shows that round robin has the lowest maximum delay. Priority is worst mainly because it does not prioritize the worst scenario. Largest buffer first tries to keep all buffer sizes the same size but not trying to send the last part of a packet. Lowest maximum delay and a balanced behaviour make round robin the best priority order and it will be used in the system.

5.2.2 Multiple messages in one packet

To get an approximation of how the overhead changes depending on number if messages all addresses are assumed to be one byte long. The size is calculated of Table 3 using Table 1.

Table 3 - Size comparison between single and multiple messages

Number of messages	Multiple message header size	Single message header size
1	5 bytes	3bytes + 1bits
2	7 bytes	6 bytes + 2bits
3	9 bytes	9 bytes + 3bits
4	11 bytes	12 bytes + 4bits
5	13 bytes	15 bytes + 5bits

This shows that multiple message is better to use if many message fit in a frame and there are many of them ready to send. But if the messages are bigger or there are not that many messages ready to be sent single message will perform better. To get a low delay the best would be to just send one message every time and do it that often that newer are any messages waiting. This idea will be the goal in the protocol.

5.2.3 When to send message

Message size known gives the opportunity to try to optimise throughput. All messages will be sent in the minimum possible fragments determined by the max and min fragment size. If a message is smaller than max fragment size it will only be sent when all of it is received. If it is larger it can be sent when at least message size minus an integer number of max fragments are received. The integer is the largest value that still returns a positive number.

5.2.4 Content of the header

From the decision to send only single messages the complexity of the header is reduced and for different fields is needed in the header. The size of them needs to be working for general solutions because the header in not intended to be changed.

Channels are how many channels are possible to have on the input depending on applications this can vary from some few to hundreds. Setting the field to 1 byte gives the possibility to have 256 channels.

Start of msg is just one bit that tells if the fist bit in the data is the first in a message. Used for be able to send start of packet on the output.

Size of payload is the size of the header + the payload that is sent. 13 bits gives the possibility to have fragment size bigger than the maximum size of a GSE packet.

Delay is the time in the input buffer and need to be able to give the maximum delay from it.

18 bits gives a possibility for delay of 262144 time units. If the clock rate is 100 MHz and one time unit is one clock pulse, it is a maximum delay of approximately 0.26s.

This sums up in a header of 5 bytes in total that looks like Figure 10.

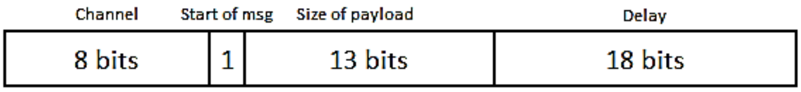


Figure 10 - Header field layout

Revaluating if multiple or single message is to be used using the decided header values is given in Table 4. The decided header is put into Table 1.

The number of messages supported is assumed 256 for multiple messages.

Table 4 - Size of multiple and single messages using header

Number of messages	Multiple message header	Single message header
1	60 bits	40 bits
2	91 bits	80 bits
3	122 bits	120 bits
4	153 bits	160 bits
5	184 bits	200 bits

The advantage of single message is improved.

5.2.5 Fragmentation of large input messages

There are two problems that have impact on fragmentation, throughput and delay.

Throughput needs to be good enough for the system to work. That means the throughput need to be equal or greater than the average input rate. Three parameters are needed to decide the throughput and they are header size, transition speed and fragment size. Header size is already determined. Transmission speed is a hardware parameter from the environment of the system and it is the lowest speed inside the system. Fragment size is not predetermined. To be able to guarantee that the throughput is enough minimum fragment size will be used to get the lowest throughput. The minimum fragment size is the smallest message size because only one message is sent in single packet.

$$Minimum \text{ throughput} = \frac{min\ fragment\ size}{min\ fragment\ size + header\ size} transmisstion\ speed$$

Eq. 3

If fragmentation is used it can lower the throughput by sending a message in two or more fragments to not get these fragments smaller than min fragment size the max fragment size need to be two times the min fragment size. It is also possible to achieve by having max fragment size greater or equal to the max message size because no fragments will be needed.

The second constraint on fragment size is delay or time in buffer. Delay gives a limitation in how big a fragment can be. Bigger fragments lead to greater delay. If all channels want to send a max fragment size packet it cannot take more than the maximum delay time.

If fixed message size is used fragmentation will be done either always or never. It can be used to overcome restrictions in delay and buffer size but at significant cost in throughput because it will multiply the amount of header by amount of fragments a message is needed to be divided into. Optimal for throughput if the delay or buffers are not limited the fragment size is equal to the message size.

5.2.6 Buffer size

Buffer size can be used for two things, with known input jitter a minimum needed buffer can be determined or a limited buffer can give the systems toleration of in-jitter.

$$Buffer\ size = 2 \times fragment\ size + burst$$

Eq. 4

Fragment size in this case is the fragment size from fixed message size input and is the only fragment size possible for the system. If not one single fragment size is possible to obtain this formula might not work. Burst is equal to the in-jitter times the channels input rate.

To get the jitter toleration of the system in-jitter is broken out of Eq. 4 giving Eq. 5.

$$In\text{–}jitter = \frac{Buffer\ size - 2 \times fragment\ size}{input\ rate}$$

Eq. 5

5.2.7 System jitter

System jitter or out jitter is the jitter the system makes and it is not the same as in-jitter. To address jitter a de-jitter buffer is placed in the end of the system delaying messages to a constant system delay. The size of the de-jitter buffer needs to be the same as the in buffer because it will compensate for the in buffers maximum delay. The time in the de-jitter buffer is calculated from max delay in the input buffer.

$$Max\ input\ delay = \frac{Buffer\ size}{\frac{\text{throughput}}{channels}}$$

Eq. 6

To make the delay always being max input delay the time in de-jitter buffer plus time in in-buffer needs to be equal to max input delay. Meaning max input delay is same like system delay.

$$Time\ in\ de\text{–}jitter\ buffer = Max\ input\ delay - time\ in\ in\text{–}\ buffer$$

Eq. 7

5.3 FPGA Implementation

The implementation did never got put in a FPGA only simulated using a test bench that just had one in channel. The functionality of the simulation did receive some error during buffer wraparounds making the system only stable for short amounts of time before data lost started toappear. Nevertheless, during the time it worked approximately

12 ms of simulation the results out of the system were exactly as expected. In both aspect of delay, data was delayed right time for the system to only make a constant delay and the data that got out was put in a file and was validated that every bit was the same like in the input file.

The implementation was made using the full protocol and the implementation of it is described below.

5.3.1 Design parameters

To be able to determine design parameters the system requirements from appendix A is used.

The first design parameter to calculate is throughput using Eq. 3. Deciding on not using fragmentation to maximise the throughput.

$$\frac{188\ B}{188\ B + 5\ B} \times 200\ Mbps = 194{,}81\ Mbps$$

Eq. 8

From the throughput gained in Eq. 8 a limitation in possible number of input channels are gained by dividing by the input rate on one channel

$$\frac{194{,}81\ Mbps}{32{,}28\ Mbps} = 6{,}0327 \approx 6\ channels$$

Eq. 9

Using 6 channels given from Eq. 9 a need of 6 input buffers and 6 de-jitter buffers are needed. This gives a total of 12 buffers. There is a bit more than 250 memories able to use according to the design specification. To not divide memories between different buffers, 252 memories will be for the buffers.

$$\frac{252}{12} = 21\ memories$$

Eq. 10

Twenty-one 1kB memories give 21kB buffer size. Using Eq. 5 to get the tolerance of in-jitter for the system gives the following:

$$In\text{–}jitter = \frac{21kB - 2 \times 188B}{32{,}28Mbps} = 5111\ \mu s \approx 5ms$$

Eq. 11

Eq. 11 gives a maximum in-jitter on 5ms using the current buffer size. Not meeting the requirements in Appendix A

5.3.2 Top level

The top layer is the combination of all layers and describes the connections between the layers. Figure 5 hows a system with three in channels. It has multiple buffers where each of them works the same but have different channel number. Implemented in VHDL the top level looks the same except more well-defined connections.

5.3.3 Encapsulation layer

In the encapsulation layer, there are two kinds of components in buffers and out buffers. Figure 11 shows an in buffer and Figure 12 shows an out buffer.

In the in buffer, both the buffer and time stamps blocks are buffers working as FIFO registers. The first data to get in will be the first to get out. Buffer stores the input data and sends it out when the control block tell it. Time stamps stores the time when end of packet (EoP) is received. The time stamp is then sent to make header that calculates the time from the current time to when the data was received. When make header is asked to send a heeder it combines all needed information to make a header and then sends it. The control is responsible for first sending header then directly followed by the corresponding data and when that is done tell the layer below that it is sent. Header and data out is sent on the same channel.

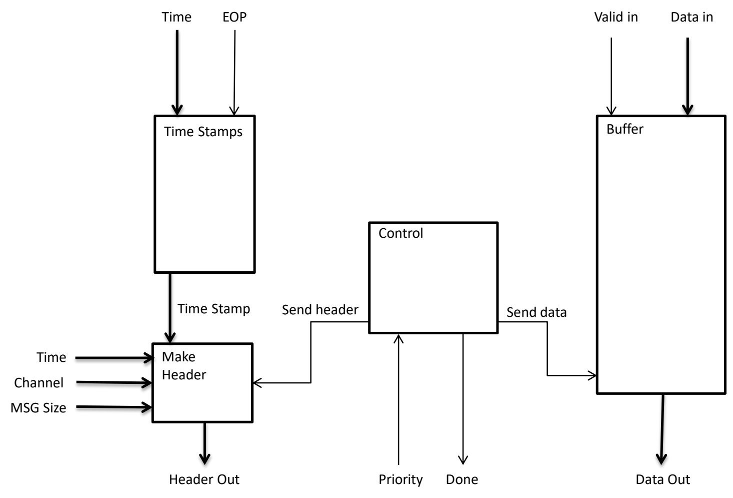


Figure 11 - Input side of encapsulation layer

Table 5 – Signals in Figure 11

Signal	Properties
Time	Gives the current time of the system
EoP	One when last bit of a packet is received on data in
Time Stamp	A stored time from the time stamp buffer
Channel	A ID for what channel that is to receive the data
MSG Size	How much data a packet contains
Valid in	Tells if data on data in is valid to read or not
Data in	Data that is to be sent by the system
Send header	Control signal that the system is supposed to send a header
Send data	Control signal that the system is supposed to send data
Priority	Tells when the system is allowed to send
Done	System tells lower layer when it have sent or if it do not have anything to send.
Header out	Sends its header on this channel
Data out	Sends out its data on this channel

The out buffer gets a header followed by data. “Get header” is a block that sorts out all the components in the header and data. The data is stored in buffer that is a FIFO register like in the in buffer. The delay that is received is used to calculate a time stamp when the message is supposed to be sent and placed in the SoP buffer. The timestamp is the current time plus max delay minus the delay from the header. When this time is reach, SoP is sent and the control will send the corresponding data from the buffer.

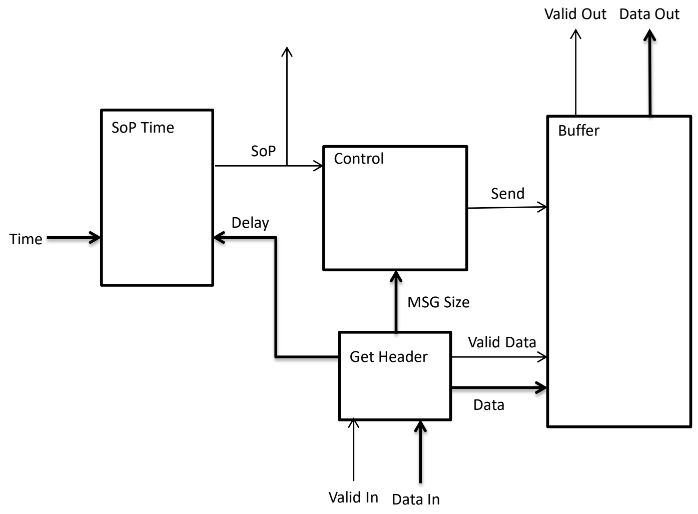


Figure 12 - Output side of encapsulation layer

Table 6 - Signals in Figure 12

Signal	Properties
Time	Gives the current time of the system
SoP	Tells when first bite is sent out
Delay	The delay taken from the header field delay
MSG Size	How big the message is taken from header
Valid in	When to read data in
Data in	The data from lower layer encapsulated
Valid data	When data and not header is received
Data	Same like Data in but delayed to sync with Valid data
Send	Control signal to when to send data
Valid Out	When o read out data
Data Out	The data out of the system same like data in on the receiver side

5.3.4 Data link layer

The in and out multiplexers will work the same except opposite directions and different control blocks. It will multiplex data and valid from same source at a time.

The inside multiplexer will have a control block that have a round robin in it and gives priority to the different buffers then waiting on the buffers to be done. While a buffer has priority, it will be the selected one in the multiplexer.

On the outside, the control block will read the header and get what channel the data is to be sent to. Then select that channel until next header arrives and the channel in that header is selected.

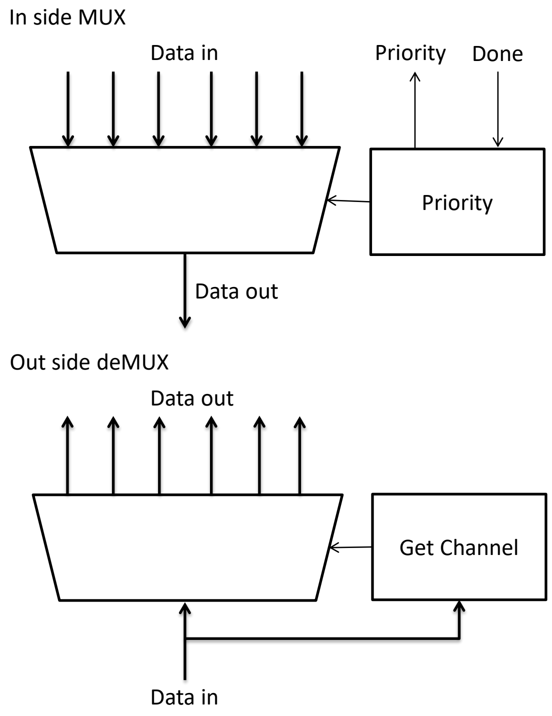


Figure 13 – Data link layer on both in and out side

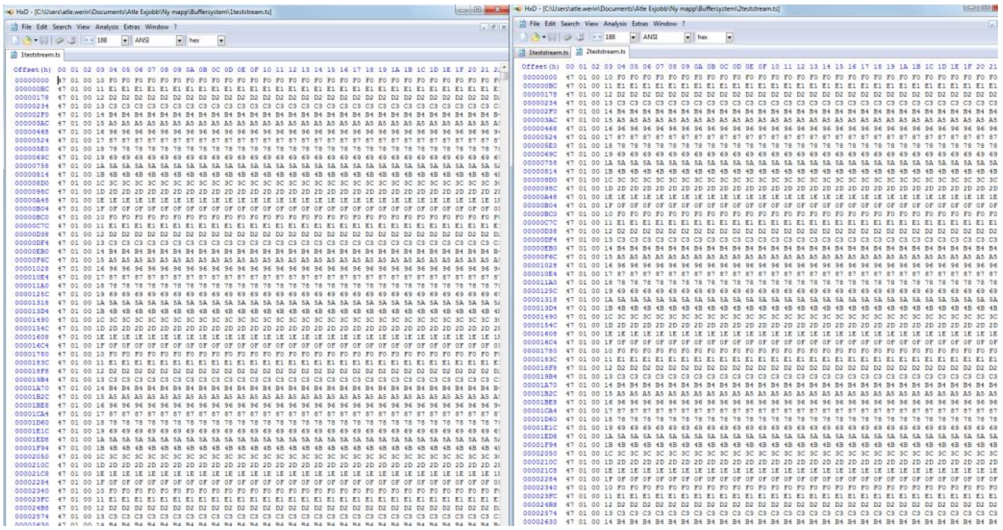
5.3.5 Physical layer

Because the expected hardware does not exist, another chip did become the platform to implement the code on. That hardware has only one FPGA and because of that, no physical connection is needed resulting in the physical layer is just a connection between the multiplexers.

5.3.6 Simulation results

Figure 14 shows to the left the beginning of the input file, note that a TS packet is 188 bytes and one line has 188 bytes represented on it but the later part is cut away to be able to show mote then on TS packet. All data in every packet is the same except the header in this data. To the left in Figure 14 is the outputs file from after running the simulation and it is identical to the input showing that received data is equal to send data.

Figure 15 is a picture from the simulator showing the outputs and inputs of the system. Valid in is in short burst because the in bit rate in is 400 Mbps and to get the average bit rate right it cannot send that often. In data, in and out, numbers are written but the actual data is sent in short bursts between them that can be seen in Figure 16. It shows the header and data that can be seen in the file at line five with data “b4”.



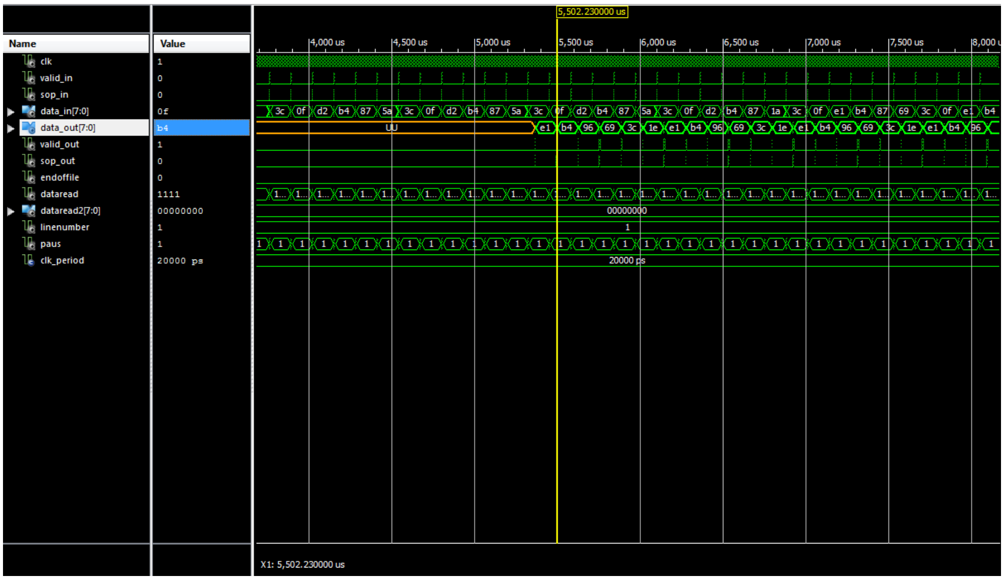
HxD - [C:\Users\atle.werin\Documents\Atle Exjobb\Ny mapp\Buffersystem\1teststream.ts]

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
00000000  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
000000BC  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
00000178  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00000234  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
000002F0  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
000003AC  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00000468  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
00000524  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
000005E0  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
0000069C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00000758  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
00000814  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
000008D0  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000098C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
00000A48  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
00000B04  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00000BC0  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
00000C7C  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
00000D38  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00000DF4  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00000EB0  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
00000F6C  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00001028  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
000010E4  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
000011A0  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
0000125C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00001318  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
000013D4  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
00001490  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000154C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
00001608  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
000016C4  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00001780  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
0000183C  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
000018F8  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
000019B4  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00001A70  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
00001B2C  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00001BE8  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
00001CA4  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
00001D60  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
00001E1C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00001ED8  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
00001F94  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
00002050  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000210C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
000021C8  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
00002284  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00002340  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
000023FC  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
000024B8  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00002574  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00002630  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
```

HxD - [C:\Users\atle.werin\Documents\Atle Exjobb\Ny mapp\Buffersystem\2teststream.ts]

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21
00000000  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
000000BC  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
00000178  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00000234  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
000002F0  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
000003AC  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00000468  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
00000524  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
000005E0  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
0000069C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00000758  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
00000814  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
000008D0  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000098C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
00000A48  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
00000B04  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00000BC0  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
00000C7C  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
00000D38  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00000DF4  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00000EB0  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
00000F6C  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00001028  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
000010E4  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
000011A0  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
0000125C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00001318  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
000013D4  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
00001490  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000154C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
00001608  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
000016C4  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00001780  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
0000183C  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
000018F8  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
000019B4  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00001A70  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
00001B2C  47 01 00 15 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5
00001BE8  47 01 00 16 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
00001CA4  47 01 00 17 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87 87
00001D60  47 01 00 18 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78
00001E1C  47 01 00 19 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69
00001ED8  47 01 00 1A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A 5A
00001F94  47 01 00 1B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B
00002050  47 01 00 1C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
0000210C  47 01 00 1D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
000021C8  47 01 00 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E 1E
00002284  47 01 00 1F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F
00002340  47 01 00 10 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0
000023FC  47 01 00 11 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1 E1
000024B8  47 01 00 12 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2 D2
00002574  47 01 00 13 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3
00002630  47 01 00 14 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4 B4
```

Figure 14 - Input and output data files



```
Name            Value
clk             1
valid_in        0
sop_in          0
data_in[7:0]    0f
data_out[7:0]   b4
valid_out       1
sop_out         0
endoffile       0
dataread        1111
dataread2[7:0]  00000000
linenumber      1
paus            1
clk_period      20000 ps
```

Figure 15 - Simulation serial interface

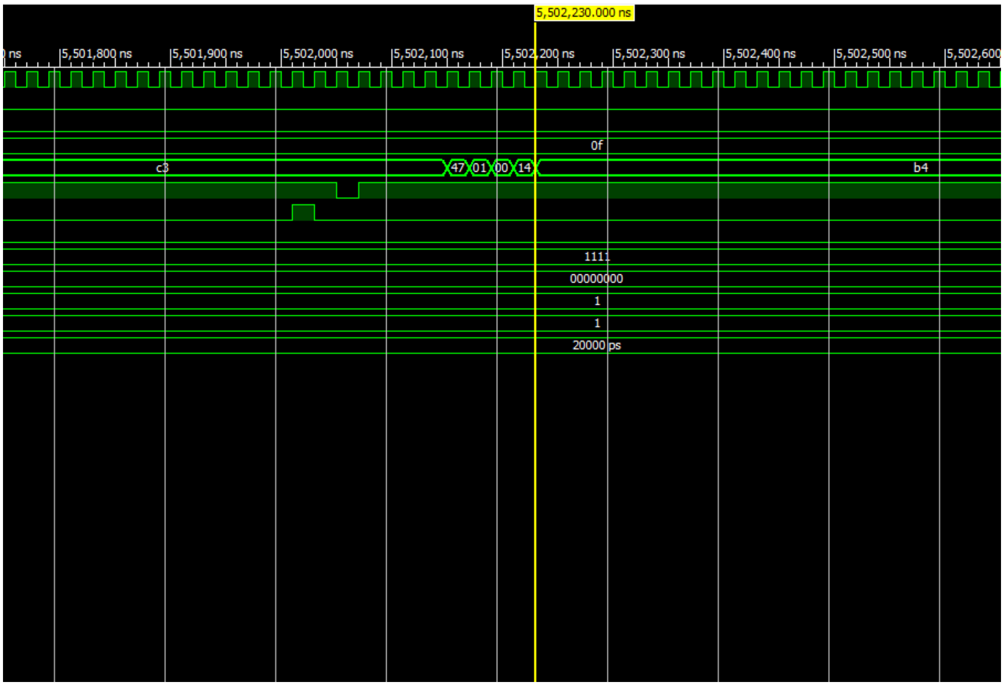


Figure 16 - Zoomed in simulation serial interface

6 Discussion

6.1 Method

Discussing design flaws in the method and how to improve the designee for future studies.

6.1.1 Protocol

The decision of what priority procedure to use is a weak point. More procedures that are different had been good to examine and verifying which one was the best. However, what priority procedure to use was not the main part of the design of the protocol. The impact of the priority procedure did affect the design of later parts and only changing it might have bad consequences for the reliability of the system. If the priority procedure was to be changes buffer size and system jitter will be affected and needed to be re-designed.

One other weakness is the verification simulation of what is the best priority procedure. It only takes one sort of in arguments that looks like TS data if other formats was to be used a simulation that included different packer size would be desirable.

Multiple messages only take in account if a part in the header is needed and how many times the parts are needed, not the size of the parts. It might be possible to go deeper in optimising the size of the fields in the header for multiple messages.

To be able to do better assumptions about if to use fragmentation or not knowledge of how the input data actually looks had been useful. However, limitation in only knowing of applications using TS a general approach had to be taken. The advantage of this is that the protocol will be able to handle mostly formats but might not be optimized for how it will be used in reality.

The way of using a de-jitter buffer in the same size like the in buffer is something that would be interesting to investigate if it ever is used to hundred percentage in a real application or if in reality a smaller buffer could be used. In theory, it is a good and easy assumption that the worst possible case exists but performance wise it might be better to look on a real system instead.

From a replicability perspective will this method generate a protocol that solves the problem of a limited interface but will there be any differences? The procedure of declining the header does not fully define how big the different fields will be and thus give a possibility for variations. If more knowledge of a specific application is known, it might change some designee decisions like if it is known that many small messages are to be sent it might be a good idea to have multiple message support.

6.1.2 Implementation

One thing that the design of the implementation is lacking is the problem of clocking, the whole idea derives from the idea that all clocks are the same. However, in reality the in clock might not be the same like the system clock.

A full system test including in-jitter from a real system was missing the data was true but how the in-jitter behaved was just a guess. The part that lacked a bit of testing because of this was the delay of the output buffer. Meaning the functionality of sending data and receiving it would not change but the delays in the buffers and how the de-jitter buffer would have to compensate for it. The tests did include in-jitter but constant in-jitter meaning that only a few cases wear tested.

The only measurement that was done was in simulations. Because the simulator gives an exact value the error is weary small but it is a closed and controlled system not a real system.

6.2 Results

The results are divided up into two parts the protocol and the implementation.

6.2.1 Protocol

Main part of the protocol is to get the same output like the input. All the functionality of the protocol does achieve this both in that the relative delay is minimized and that is sends the data to the right output. Because this is achieved, the protocol is working and the designee is a success. Secondary is that good performance and support for all possible protocols. This is harder to validate but it has support for different protocols but might not give optimal performance.

The goal was to make a general protocol that would work for any kind of input. But because the implementation was for TS it is hard to say how good it will work for other formats. The decision of using not multiple messages is an example of this, in GSE small packets are supported and if only packets of the smallest size was to be obtained the throughput would be low. Even using multiple messages would not make a good throughput but better meaning an alternative solution might be required.

If comparing the protocol to formats discussed in the theory chapter the biggest difference is the system jitter support. Having a mandatory field in the header for delay is unique. Other formats that not using a delay field might be to be able to have a smaller header or even implement multiple messages without much extra header for each packet. Another reason can be that the use other methods of removing jitter or ignoring it. If the jitter made by the protocol is used it might even help nullifying jitter from the transmission formats. This because DVB formats sends BB-frames containing multiple packets and making jitter by delaying the first more and the last least. In this protocol during bursts the first is delayed the least and the last the most. How wheel this work need to be evaluated if it is to be used but it is a possibility.

6.2.2 FPGA implementation

Even when the FPGA implementation is not stable for more than a short time it proves the functionality of the protocol designee by receiving same data on the output like the input and the relative delay is the same for all data.

In Eq. 11 the fragment size is only a small part and is not that relevant in how much it affects the possible jitter. This meaning that the question of fragmentation or not is less important when the jitter is large. The delay of the system will be big when the jitter is large and evaluating if this is a good solution might be a good idea.

6.3 Source criticism

Because the project is done in an industrial environment it is mostly based of existing standards and not academic papers. The gain of this is that it is well suited to putting in a context of existing technology but it might limit the new value.

6.4 The work in a wider perspective

Ethical affects are not present in a greater degree but environmental positive effect on that it can potentially reduce amount of products needed by increasing amount of inputs. The environmental effect is achieved by resulting in both reducing the need of resources for making products and reducing the power consumption by having fewer running at a time.

With the application today no major impact on society is expected but it might be used in different location to reduce the amount of needed hardware that can make it more easily accessible.

7 Conclusions

7.1 Problem statements

- How is it possible to mux together TS (transport streams) from multiple channels to one single channel send the TS by a serial interface and then demux theme back to their original state again?

It is possible by having a protocol that have knowledge about important details and relay them to the receiver by using a header. The receiver then uses the information and recreates the signals.

- Is it possible to take the solution for TS and apply it for other formats like GS and if not how to make it possible?

Yes it is possible but the performance might not be optimal. Limited knowledge in how GS is used makes it hard to know what to expect, excepting extremes and using a protocol made for TS then is a limitation. The protocol is not only designed for TS, it has some extra functionality to be able to support different GS formats. A purely TS designed protocol would not work for GS but a general attitude was taken during the protocol designee.

- What design parameters are important and how do they impact the protocol?

Se 4.2.1 , worth noting is that some are calculated like throw put and some are needed from a designee specification describing the physical application.

7.2 Results

The results of the protocol designee are what was expected a protocol that solves the specified problem. The implementation did not reach its intended state of being implemented in a physical FPGA. One reason for this was limited knowledge in how to implement a system in VHDL. Implementing a VHDL system differs from normal programing and to be able to use features of VHDL one need to think more in hard ware and clocking then normal programing. It took some time to get used to it and that resulting in problems in the first parts of the code. Limited time lead to not enough time to go back and fix the problems this caused. Resulting in a not stable implementation and because of this the decision not to put it in a FPGA was taken.

Appendix A

Requirements specification

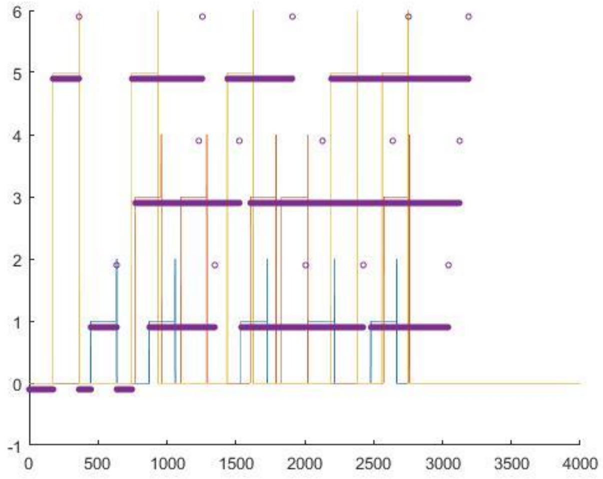
- Each input channel has an average bit rate on 32.28 Mbps
- In-jitter 20 ms
- Internal transmission speed in the physical layer 200 Mbps
- System have demodulators and serial to parallel interfaces for the inputs
    - 8 bits parallel data channel
    - Valid bit
    - Clk 50MHz
    - SoP
- Buffer is limited to approximately 250 8kbits block RAMs
- In data format will be TS

Appendix B

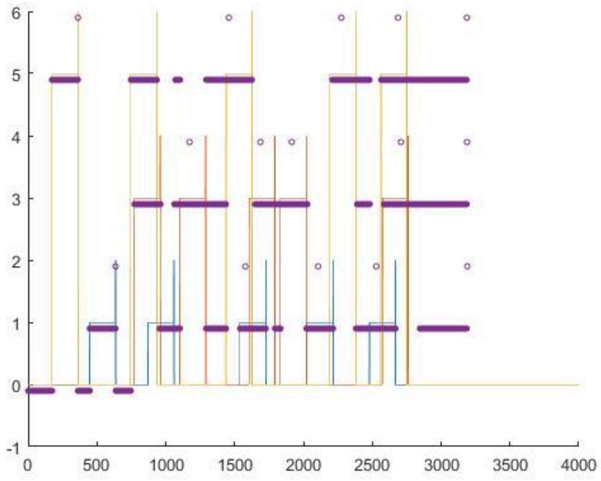
Priority simulation results

Plots from the mat lab simulation of priority order. Three channels in different colour and data have different values in them. In channel 1 data have the value one and the end of packet bit is two. Channel 2 has three and four and channel 3 has five and six. The purple circles indicate when a specific data value is sent out. The small offset in the purple circles are only graphical. The Y-axis is what data is sent or received the X-axis is time units.

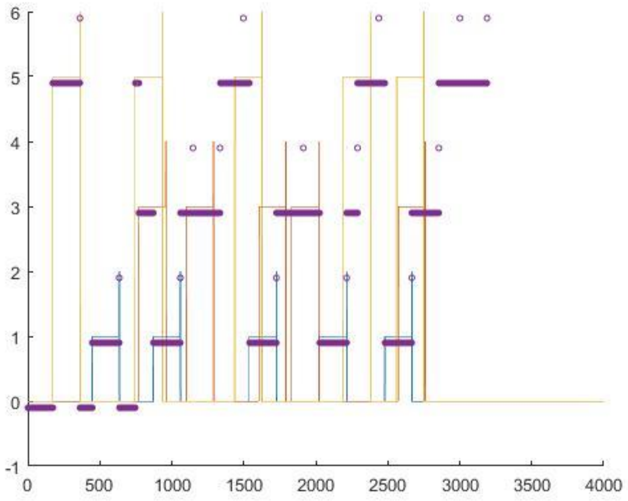
Round robin



Largest buffer first



Priority order



	Max delay
Round robin	619
Largest buffer first	688
Priority order	812

Bibliography

[1] European Telecommunications Standards Institute (ETSI), “Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2),” EN 302 755 (V1.2.1), 2010.

[2] European Telecommunications Standards Institute (ETSI), “Digital Video Broadcasting (DVB); modulation systems for Broadcasting , Interactive Services ,” EN 302 307-1 (V1.4.1), 2014.

[3] European Telecommunications Standards Institute (ETSI), “Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2),” EN 302 769 (V1.3.1), 2015.

[4] A. Morello and V. Mignone, “DVB-S2: The second generation standard for satellite broad-band services,” *Proc. IEEE*, vol. 94, no. 1, pp. 210–226, 2006.

[5] European Telecommunications and Standards Institute (ETSI), “Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE) Protocol,” TS 102 606 (V1.1.1), 2007.

[6] International Organization for Standardization (ISO), “Information technology -- Generic coding of moving pictures and associated audio information -- Part 1: Systems” ISO_IEC_13818-1, 2007.

[7] R. Ramjee and J. Kurose, “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks,” *IEEE Infocom*, pp. 680–688, 1994.

[8] Mohammed M. Alani, *Guide to OSI and TCP/IP Models*. 2014.