Point-to-point protocol exploration

Nayib Boukadida

July 9, 2018

Supervised by F. Schreuder[1], A. Borga[1], P. Jansweijer[1] and W.E. Dolman[2]
[1]Nikhef, [2]Amsterdam University of Applied Sciences

Nikhef

Summary

In short this document describes the assignment of researching and implementing the best point-to-point protocol matching a set of clear requirements. FPGA's are targeted to implement the protocol. The motivation of this work is primarily the increasing demand in data transfer rates.

This document will start with extensively describing what current point-to-point protocols are used for and what parts they consist of. This explanation will be followed by a survey of currently available protocols will be presented which concludes the Interlaken protocol actually met all of the predefined requirements. These are among others a line rate of 10 Gbps, inclusion of flow control, a determined range distance coverage and CRC. A complete description of the Interlaken protocol will be presented where after an implementation will follow.

The protocol has been realized targeting as much vendor independence as possible. A Xilinx VC707 evaluation board has been provided to realize a proof of concept. Only the transceiver and FIFO's are using IP-cores. The implementation has successfully been tested on a single board using a loop-back fiber and also on two VC707 boards communicating with each other.

The specifications and implementation are published as Free and Open Source on code hosting platforms like OpenCores. Sharing is done with the aim of promoting the dissemination and broad adoption of the designed implementation.

Samenvatting

Kortom zal dit document een opdracht beschrijven waarin onderzoek wordt gedaan naar welke van de bestaande point-to-point protocollen het best voldoet aan vooraf opgestelde eisen. Hieronder vallen onder andere een snelheid van 10 Gbps, de aanwezigheid van flow control, een vooraf gedefinieerde af te leggen afstand en CRC. Ook zal hier een implementatie gericht op FPGA's uit volgen. De aanleiding van dit werk is primair de toenemende vraag naar snellere dataoverdracht.

Dit document zal starten met het uitgebreid beschrijven waar de huidige point-to-point protocollen voor gebruikt worden en uit wat voor onderdelen deze zijn opgebouwd. Een uitgebreid onderzoek naar de huidige protocollen zal hierop volgen waaruit geconcludeerd kon worden dat het Interalaken protocol aan alle eisen voldoet. Een complete beschrijving van het Interlaken protocol en de implementatie hiervan zijn tevens beschreven.

Het protocol is gerealiseerd met in gedachten dit zo leverancier onafhankelijk te houden als mogelijk. Een Xilinx VC707 evaluatiebord is gebruikt om mee te testen en eerste concept als bewijs van realiseerbaarheid op te leveren. Alleen de transceiver en FIFO's maken gebruik van IP-cores. Tevens is de implementatie met succes getest op een enkel VC707 board met teruglussen van de glasvezel en ook is het gelukt twee VC707 borden met elkaar te laten communiceren over glasvezel.

De specificaties en implementatie zijn gepubliceerd als gratis en open source op codehostingplatforms als OpenCores. Het delen word gedaan met achterliggende gedachte om de verspreiding en algehele acceptatie van de ontworpen implementatie te bevorderen.

Version history

Version	Date	Changes
1.0	16 Mar. 2018	Finished first concept
1.1	27 Mar. 2018	Changed lay-out, added clearer explanation of protocols removed code appendix and included the OSI model
1.2	10 Apr. 2018	Added a section dedicated to Interlaken, completely rewritten the hardware implementation and added a list of Figures/Tables
1.3	23 Apr. 2018	Added microsemi LiteFast, improved referencing
1.4	29 May. 2018	Corrected grammar/spelling mistakes, added appendix of obtaining and configuring Core1990
1.5	29 June. 2018	Expanded Core1990 appendix and added test results. Improved the implementation description

Contents

List of Figures

List of Tables

Abbreviations

CERN Conseil Européen pour la Recherche Nucléaire (European Council for Nuclear Research).

CRC Cyclic Redundancy Check (Variants will always be noted with a '-' between CRC and the length).

EMI ElectroMagnetic Interference.

FEC Forward Error Correction.

FIFO First In, First Out.

FPGA Field-Programmable Gate Array.

Gbps Gigabit per second.

GT Gigabit Transceiver.

HT HyperTransport.

ILA Integrated Logic Analizer.

IO Input/Output.

IP core Intellectual Property core.

LFSR Linear-Feedback Shift Register.

MMCM Mixed-Mode Clock Manager.

NVMe Non-Volatile Memory Express.

OSI Open Systems Interconnection.

PCIe Peripheral Component Interconnect Express.

PHY PHYsical layer of the OSI model.

PLL Phase Locked Loop.

SATA Serial Advanced Technology Attachment.

SerDes Serializer/Deserializer.

SFP Small form-factor pluggable.

SPI System Packet Interface.

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

VIO Virtual Input/Output.

XAUI X(Ten) Attachment Unit Interface.

1 Introduction

This brief introduction covers the motivation to this work, providing a general overview on the assignment, and describing the structure of this report.

Virtually any sort of electronic communication system involving two or more peers requires the use of a protocol[1] to achieve proper information transfer. The nature of such protocol depends on the characteristics of the communication channel and the type of communication that one wants to pursue. A general overview of a communication protocol is given in Chapter 2. Since the protocol is targeting low level communication, a key element of it is the so called transmission encoding/decoding. Common techniques for such purpose are researched in Chapter 4.

Before exploring the commonly available and best suitable protocols, one should set a set of requirements that should be met. Such requirements are summarized in Chapter 3. An answer to the question if there is a suitable protocol available for the set requirements, should be clearly addressed. If none of the available protocols meets the requirements then the solution must be custom tailored and implemented. This last option is to be possibly avoided, as custom protocols are implementation time consuming, and harder to share and port to other designs.

This document is intended to summarize the findings of an assignment in which different point-to-point links are to be explored. At the time of writing several options considered to be "actual" have been found and analysed in Chapters 5 and 6.

No stranger to the concept of communication protocols, large particle detector facilities massively deploy electronics systems which always require a specific point-to-point connection at - for example - a certain stage of a Data AcQuisition (DAQ) chain. As a practical study case, this research uses CERN [2] facilities to explore some of its real-life examples. A list of sampled CERN proprietary protocols historically used is provided in Appendix A. CERN innovates in many fields of fundamental research, and technology, thus also on electrical engineering related matters: as the demand of data processing for large experimental facilities increases, and larger dataset are to be transported through existing point-to-point links, technology has to be scaled and adapted over time to meet the future challenges. At the time of writing this document (2018) there is a desire to upgrade certain sub-systems to a different protocol that could transport higher-throughput data and to more efficiently make use of the offered bandwidth.

The complete proposed design is realized in hardware and the process is described in Chapter 8. The implementation targets vendor independence to maximize cross-platform portability; meaning that it should work on every FPGA with mid to high range specifications, and fast transceivers. This should be the only strong requirement in terms of the FPGA target device.

[1] In telecommunication, a communication protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. The protocol defines the rules syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both [1].

To summarize: the purpose of this assignment is to research and to implement the best protocol matching a set of clear requirements, and it should target an implementation using an FPGA. The specifications and implementation are published as Free and Open Source on code hosting platforms like OpenCores. Sharing is done with the aim of promoting the dissemination and broad adoption of the designed implementation.

2 Structure of communication protocols

Before getting into more details about the search and implementation of the surveyed protocols, the structure and functionality of communication protocols is described. This is for the sake of clarity to the readers and the author himself. Without a thorough understanding of a protocol structure and what to expect from it, the requirements could be misinterpreted, resulting in erroneously grounded research. This section is especially beneficial for new or less experienced readers on this subject.

2.1 The OSI model

Communication functions of a computer system can be developed keeping the standardized Open Systems Interconnection (OSI) [3] model in mind. This model is designed to describe how data communication between two peers should take place. According to the OSI model communication systems consist of seven different layers which can be interpreted as groups with their own functions and responsibilities. In short, the model defines a stack of standards describing the way devices communicate and inform each other, when to send data, when to stop transmitting data, and so on. It also ensures devices use the correct data rate, guarantees the arrival of data at the receiver side, and in which way the physical connection is implemented. A general overview of the seven layers can be seen in Figure 1 [3]. The depicted structure goes bottom up with the lowest layer below.

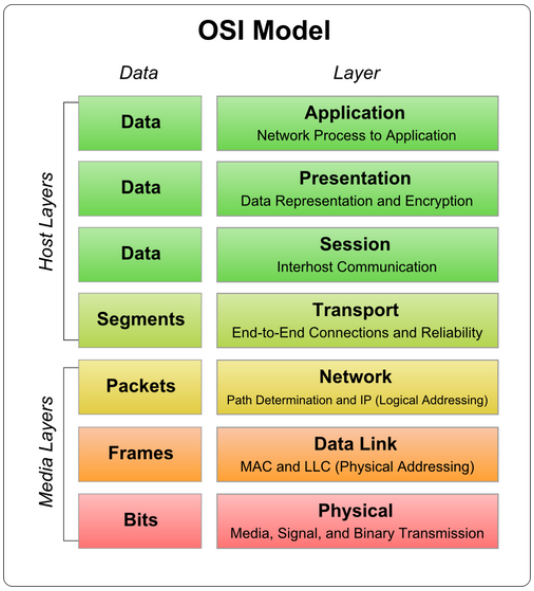


Figure 1: An overview of the OSI model.

The first layer is the *physical layer* which is responsible for the communication signals passing through the connection between two points. Often the term *PHY* is used to refer to this layer. This concerns the physical connectors, cables, pin-outs and also the actual voltage (or the light) on the line, depending on the type of encoding the protocol makes use of. As shown in the figure above, the data from this layer can be seen as multiple "raw" bits. Further specifications the physical layer normally includes are the data rates, maximum transmission distances and low-level error correction. For example in the Ethernet standard

this layer contains multiple sub-layers which are the Physical Coding Sub-layer (PCS) delivering encoded data, Physical Medium Attachment Sub-layer (PMA) responsible for the attachment interface, Physical Medium Dependent Sub-layer (PMD) which defines the electrical transmit/receive characteristics and the Forward Error Correction Sub-layer (FEC) [4].

The second layer is called the *data link layer* and ensures error-free data transmission between two physical layers. This is done by arranging the stream of data in such way the receiver can easily understand. This is done by packing the data into so called frames, an explanation of which will follow in section 2.2. These frames will then be moved to the physical layer. This layer is also responsible for computing and adding data error correction, and including framing bits. The layer can also be divided in two sub-layers, the Logical Link Control (LLC) and the Media Access Control (MAC). The LLC is responsible for error checking, frame synchronization and flow control while the MAC is responsible for controlling how multi-peer network access is granted to the medium, and the permission to transmit data.

The third layer is the *network layer* which is primarily responsible for determining which physical path the data should take. It contains the function to route the data to it's recipient according to circumstances. This is where addresses will be added to the packets which is comparable with the well-known IP addressing from the Internet Protocol Suite [5]. Other functions handled by the data layer are packet switching and sequence control.

The fourth layer is called the *transport layer* and here the focus lies more on the delivery of messages between networked hosts. This is where the messages are fragmented and reassembled. This also contains higher-level error detection and recovery, flow control and guaranteed data delivery. Common protocols that function on this layer are TCP and UDP [5].

Since this document intends to target chip-to-chip communication, not all layers are important. In this case the Physical and Data Link layers are of most interest. Networking is also interesting in case data has to be transmitted over multiple channels. The transport layer can be of interest in case of multi-peer networked topologies, and that is the reason why it was still described. The other layers are of less interest.

2.2 Data structure and framing — OSI Layer 2

Transferring data is not just transmitting a bulk of data bytes to the other side. A certain amount of bytes will be accepted by the protocol which will package this data and prepare this for transmission. Depending on the protocol users can indicate when the data stream starts and when this ends. A whole lot of other indicators could be available to the user like choosing which bytes in the specific package are valid or the channel number to transmit on in the case of multiple data lines. The protocol will register this and pack them into so called frames. [6]

Not only do these frames contain for example the start and end of a package but they could also be used to transmit error correction and other crucial information. These frames are often categorized as control words. However the data itself is packed in a data word. These words always contain the same amount of bits. [7]

Of course the data can be transferred using a serial or parallel connection. This completely depends on what the designer of the protocol has implemented and at which speed the data is required to be transferred. Transmitting data in parallel requires more effort in comparison to a serial connection but delivers more data to the other side in the same time as expected. The data itself can be serially processed inside the electronics but it is also possible to process this in parallel using a specific package width.

2.3 Error detection and correction OSI Layer 2/1

It is never certain whether the data received exactly matches the data that has been transmitted. During the data transmission all kinds of interference can be picked up by the communication line and can cause data bits to flip or be read as the wrong binary value. Without any indication whether these events overcame the data this bit flip will go unnoticed and corrupt data will be processed as if it were identical to the data sent by the transmitter.

To overcome these unnoticed forms of data corruption multiple ways of so called error correction have been researched and are nowadays nearly always implemented. The only downside of error correction is that additional information has to be transmitted between data packets but this sacrifice is in most cases worth the guarantee correct data will be processed by the receiver.

The easiest way of error detection is to add a checksum making use of a parity check. In this case the total number of high bits will represent a value according to their position in a specific byte. These values will be summed and the total will be divided by the amount of bytes again. This method offers easy detection of a single bit flip but if two bits are flipped at the same position in two different words, which represent the same value, the checksum won't change. For small amounts of data which are not that important it should suffice but when the received data is has a more important function, other methods of error detection are preferred.

Another form is the Cyclic Redundancy Check (CRC) which has to be implemented before the encoding of data. In this case a so called polynomial is implemented which will be used as a divisor on the data. This pack of data will be divided by the polynomial starting at the most significant bit. After every division the result will be divided again but this time with the divisor shifted one position to the left. This can be interpreted as constantly using an XOR on the data bits. At the end of constant divisions, the complete data pack itself will be padded with zeros while the remainder should be the length of the divisor minus one bit. While dividing it is important to know that when the leftmost bit of the result data equals zero, the data will be divided by zeros instead of the divisor.

To validate the data this complete division will be repeated again but now using the generated remainder. This should result in is exactly zero for both the resulted data and remainder. [9] When the result is not equal to zero the received data is not identical to the data that was originally transmitted. An example of this check with a correctly arrived and corrupted package is depicted in Figure 2. [10] The most common CRC variants currently in use are CRC-8, CRC-16 and CRC-32.

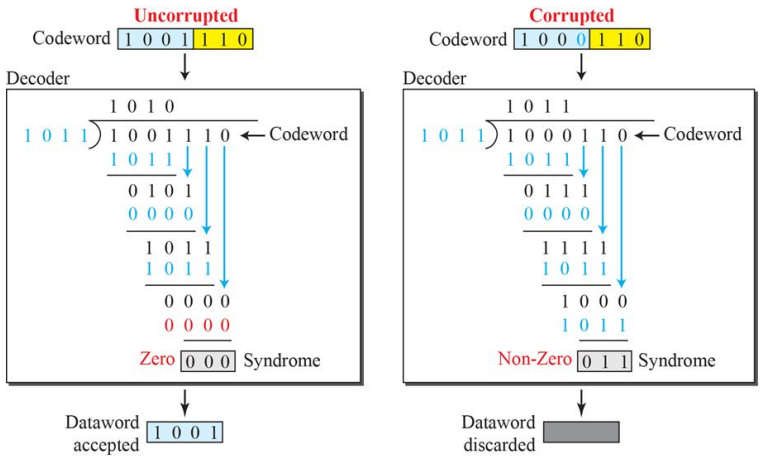


Figure 2: CRC division in case data appears flawless or corrupted.

Forward Error Correction (FEC) is also an option which most of the times is used after encoding the data stream. Another term used for this way of error correction is channel coding. This technique is used to detect errors while communicating over unreliable or noisy channels. The big advantage of this technique is that it contains the ability to correct errors and when strategically placed between the encoder and transmission line, data entering the decoder will not contain errors caused during the physical transmission.

The so called Reed-Solomon error correction is an often implemented variant of FEC which contains the ability to detect and correct symbol errors. Another high-performance variant of FEC is called turbo code. This specific method of error correction makes use of two encoders, an inner and outer encoder with an additional interleaver. Multiple variants of these turbo codes are in use nowadays which are for example popular in deep space communication, 3G, 4G and different satellite networks. Optical forms of communication also make use of turbo code [11].

2.4 Encoding of data — OSI Layer 1

Data has primarily always been transmitted using physical electrical conductive connections. While transmitting a lot of the same signs in a row over such a line, this causes the average line voltage to change in value and thus introduces a DC-component. This can cause the capacitance of line filters to be charged and the average voltage between ones and zeros can shift. This will eventually result in corrupt data and can even damage connected components. While the so called baseline wander is problematic, the short on transitions can additionally cause errors in the delineation of word boundaries, a form of constant EMI will be present and clock recovery[2] won't be possible since there will be no transitions to focus on.

A widely used solution to this problem is the addition of so called line encoding. The purpose of this technique is to encode the data in such way the receiver can still decipher it but long steams of the same binary value will be prevented. Encoders come in a lot of different variants.

[2]The process of regenerating the clock signal at the receiving side with the absence of a separate clock signal. This can be done by using the timing information of the data stream.

The most simple and often used variant is 8b/10b encoding. This makes use of certain algorithms that will translate the bits to a specific pattern that will be transmitted. However as the name indicates for every eight bit transmitted two will be added as a header which results in about 20 percent of wasted bandwidth. The reason why they are still used a lot is because of their excellent DC-balance and they are easy to implement.

Through time more advanced encoders have developed like the 64B/66B encoder. The caused reason to innovate was primarily because of the overhead included with 8b/10b encoding. Since data rates nowadays grow rapidly, as less bandwidth as possible should be wasted. These moderner encoders also don't use a lookup table but make use of a so called scrambler which uses an algorithm to randomize the bit orders. Multiple variants have again tried to improve on the 64b/66b like the 64b/67b which offers better DC-balance. More details on this subject will be explained in section 4 which will be dedicated to the currently existing ways of encoding and decoding.

2.5 Serialization and parallelization of data OSI Layer 1

Serial and parallel data streams have been discussed in short earlier. When high speed transmissions are required it is preferred to gain a speed high as possible over a single transmission line. This doesn't mean all data processing operations inside the chips have to happen in serial order. Most of the times data is processed in parallel using FPGA's or dedicated chips suited for data transfer. Serially processing the data steam inside the chips would be too slow and bottleneck the line speed.

The parallel data bus will be translated into one or multiple serial data steams at the transmitter side and the receiver has to revert this transformation. Of course the speed of these lanes will also be altered according to the speed the parallel bus was transferring data on. The component that is responsible for this task is called a "Gearbox". When for example four 2,5 Gbps lanes are translated into a single 10 Gbps line, this would be mentioned as a 4:1 gearbox. The same can be said when this process is reversed with a 1:4 gearbox. Often this conversion block is also called a SerDes (SerializerDeserializer) and this is the last step before transmitting data. In Figure 3 an example of two gearboxes with a transmission line in between is shown. [12]

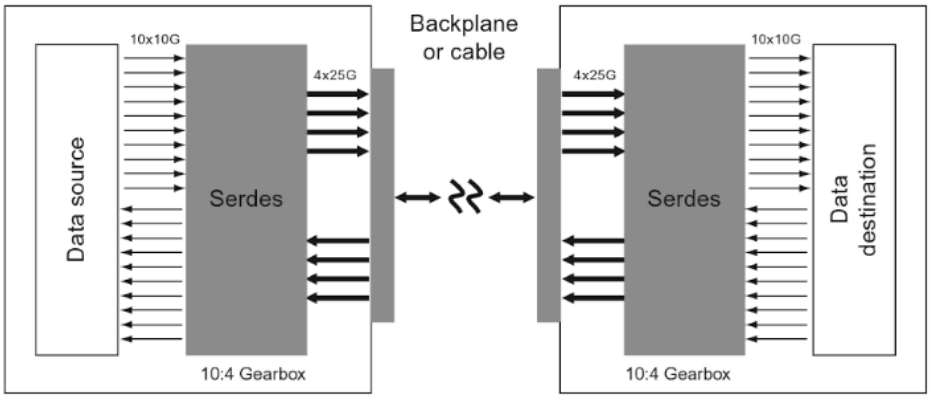


Figure 3: SerDes gearboxes.

2.6 Protocol overview

After this research it should be clearer what to expect of a communication protocol and what fundamental components it consists of. A simple depiction can be made to summarize this section which can be seen in Figure 4. Transmitted and received data will both be stored in separate FIFO's which act as a buffer for adding and removing frames in between the data packets. In case independent clock FIFO's are implemented, the protocol can be assigned different clock speeds than other section on the chip which can be seen as an advantage.

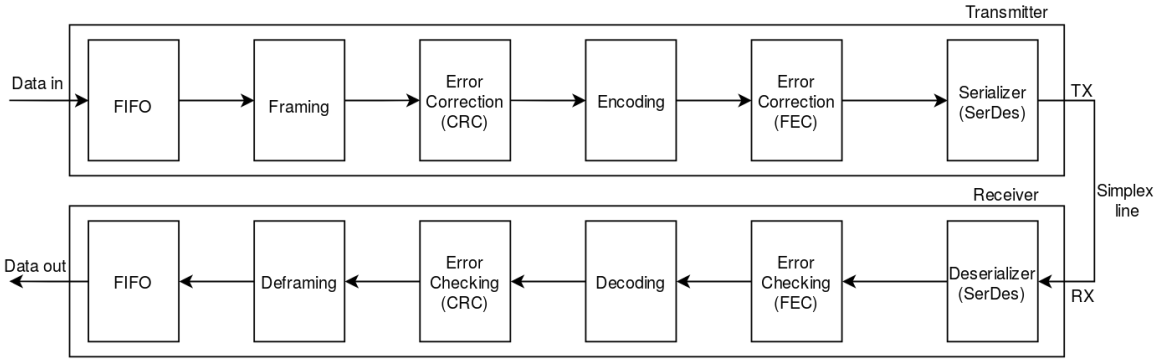


Figure 4: Protocol overview according to earlier subsections.

The framing, encoding and error checking are also present. In this example a simplex line has been displayed. Data is only transmitted in one direction from TX to RX. Both the transmitting and receiving side should have a TX and RX pin. Additional information and status updates on the receiver can then be transmitted in the other direction. This way data transmission could be for example slowed down or completely stopped in case an error occurs at the receiver side.

3 Requirements

Before the search for suitable protocols starts, it is important to know what kind of protocol would suffice and what specification should at least be required. This will drastically increase the process of narrowing the search to a handful available protocols. For example in case the line rate is not sufficient, it is immediately clear the protocol is not suited for the application it has to fulfill.

This section contains these requirements with their description and explanation in case this has not been discussed in the previous section. There is a minimum line rate that should be possible to reach. At that rate a certain line range should be covered and flow control has to be included. Error detections and corrections like the Cyclic Redundancy Check and Forward Error Correction are a very important addition to the protocol and their priority will be discussed.

In addition to these requirements it is the purpose to look for a protocol that transfers data in a serial way and not in parallel. Of course bonding could be implemented but this should only apply in case a single line could at least transmit the minimum amount of data required.

3.1 Line rate target — OSI Layer 1

Line rate is the physical speed at which the device communicates with the equipment directly. For this project it has been specified the line rate has to be about 10 Gbps. This results is a rate of 1,25 GB/s when converted to bytes.

However while the line rate is the physical speed, this doesn't mean the actual data (payload) will also be transported at this rate. There is also overhead that has to be taken into account which adds additional information transmitted. The encoding and framing of data to be transmitted have a big impact on the amount of overhead. Error correction will also play a determining factor because this often adds redundant data. In this case the payload the data contains has to be more than 70%. This means that a maximum of 30% overhead is allowed, which still is quite a large percentage.

3.2 Range distance coverage — OSI Layer 1

This is the effective distance that can be covered by the signal that is carrying the data. In this project it has been specified the data stream should cover short to medium distances that vary between around 10 and 200 m, which corresponds to normal distances covered by cables in data centers.

The range largely depends on the transmission medium, the encoding used by the protocol and how many parallel transmission lines are used. Also the EMI induced on the line by other operating devices or transmission lines can cause the range to decrease. This can be solved by implementing FEC as explained in section 2.3. Each protocol has it's own effective range distance coverage which makes some protocols very suitable to use in this case while others can quickly be skipped.

3.3 Forward Error Correction OSI Layer 1

Forward Error Correction is an optional feature but as explained the addition could prove very useful. Especially in noisy environments this could prevent a lot of possible errors. Especially since it provides the ability to also correct these. It is certainly a feature to keep in mind and it would be great if a protocol provides at least the must haves and also has implemented FEC.

3.4 Flow control OSI Layer 2

Implementing flow control is a must have so it's absolutely necessary that the chosen or designed protocol includes this feature. Flow control is the ability to manage the data rate of the transmission process between two devices. While the transmitter cant send data at the maximum speed the transceiver and protocol are allowing, this doesn't mean the receiver can always process all this data at the same rate. For example the receiver could have other tasks or the FIFO is read out at a slower rate and has the tendency to overflow, this could cause data loss because the receiver is simply overwhelmed by the amount of data and has no place to store it. The consequences could be disastrous.

By implementing flow control the receiver will be added the ability to make it's processing speed clear to the transmitter. There are multiple variants to implement this but now the transmitter has an indication at which rate it can send data. The most simple way is Stop and Wait which is depicted in Figure 5 [13]. The disadvantage of this way is the waiting time before the ACK arrives at the transmitter and this wastes resources. A different approach is the Sliding Window in which multiple frames are transmitted and while transmitting the ACK's will arrive. This boosts the efficiency considerably.

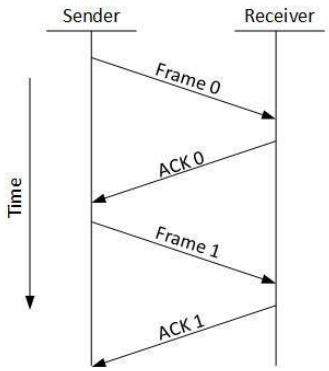


Figure 5: The simple Stop and Wait implementation of flow control.

3.5 Cyclic Redundancy Check OSI Layer 2

The Cyclic Redundancy Check is an optional way of error detection which is used to detect corruption in the received frames. This is not a must have but it would be very useful if a protocol meeting the minimum requirements would also offer this feature. It would cause frame errors to be detected and prevent using corrupt data as if it were identical to the transmitted data.

3.6 Channel bonding OSI Layer 3

This is a technique where two or more links are combined to increase throughput or to add redundancy. Differently formulated this will lead to an increase in the total amount of bandwidth available. For example this technique can often be found in wireless Internet connections like 4G and WiFi. Figure 6 shows a visualization of the bonding technology [14].

Especially in the case of using FPGA's this technique comes in very useful since it offers the possibility to bond multiple transceivers and can handle the high speeds at which data flows. Xilinx and Altera for example both offer their own specific high speed transceivers which individually can reach high line rates but they can be bonded for even higher rates. Bonding is a completely optional feature to add but as explained this could cause a huge increase in the maximum amount of data that could be transported. This would also make the protocol better suitable for more use in the future.

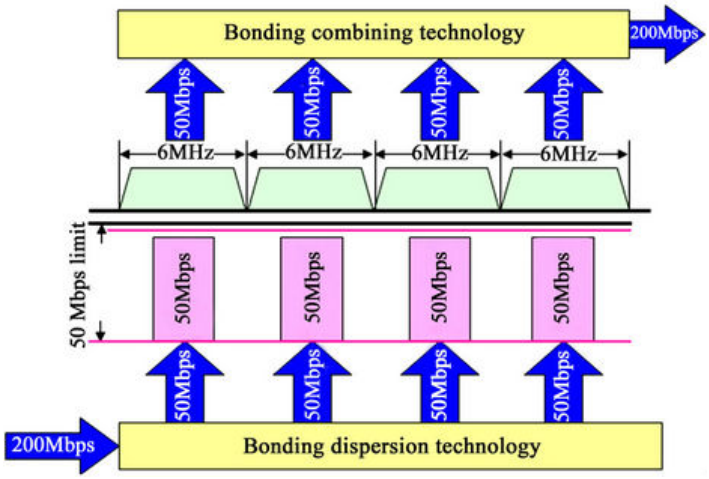


Figure 6: Transmitting data over different channels using bonding.

Not only does bonding provide higher bandwidth, the distance that can be reached while using wires also extends. The data packets will be fragmented and transmitted containing additional headers [15]. This will cause the receiver to be able to reconstruct the complete package using the headers containing crucial information.

Of course implementing this technique requires a lot more effort. All lines will have to be in constant synchronization, otherwise reconstructing packets will be very difficult and the chance is fairly high a lot of data will be lost.

4 Line encoding and decoding

Line encoders and decoders are essential components in the Physical OSI layer, which has also been described in section 2.4. Multiple ways of line encoding have been introduced over time and all have their own characteristics. Since many variants are still widely in use, this separate chapter has been written to provide more information on how important these techniques are. The process of encoding and decoding itself will be described in more detail but also other additions that play a role while encoding will be discussed.

Each line encoding and decoding technique has it's own benefits while the drawbacks should also provide important knowledge. The presence of scramblers in modern ways of encoding will also be discussed. In short they are responsible for the prevention of DC offset by randomizing the data. Modern encoders are responsible for the addition of synchronization headers.

4.1 8b/10b

A fairly simple form of encoding which was developed by IBM in 1983. Each incoming byte will be translated into a constrained 10-bit binary.

The encoder actually consists of two components, a 5b/6b and a 3b/4b encoder. The last five bits of the incoming byte will be picked and placed at the start of the encoded byte. After these bits, an extra bit will be added. The remaining three first bits the incoming byte contained will follow with an extra bit added again.

Those so called extra bits are known as running disparity. These are variable bits to ensure the balance between ones and zeros is maintained. Otherwise DC unbalance can cause malfunctions in the system like explained in section 2.4 [16].

The 5b/6b and 3b/4b encoders make use of separate tables to translate the original data bits to the accompanying encoded bits. This way of coding also provides the ability to transmit control symbols and characters to the receiver over the line.

8b/10b encoding is still widely in use despite multiple attempts have been made to develop better encoders. The reliability that comes with the encoder is why many protocols still use 8b/10b encoding. Also the absence of DC-imbalance is a huge advantage. This is possible because the used algorithm always balances the amount of ones and zeros. However the 20% overhead that comes with it and increasing demands in transfer speeds cause protocols to slowly look for different variants of encoding.

Technologies using this way of encoding are among others DVI, HDMI, Gigabit Ethernet (SGMII), SATA, SAS, USB 3.0 and XAUI.

4.2 64b/66b

At the time of writing 8b/10b encoding is still widely used but the 20% overhead is quickly becoming a bottleneck while transmitting data a high rates. Different and more efficient ways of encoding have been introduced and 64b/66b is one of them. It is immediately clear that this way up to eight bytes can be transmitted in one package while using a two-bit preamble. This way of encoding provides a better approach to prevent the earlier discussed quite large overhead, which in size went from 20% to a decreased percentage of little over 3%. Less bandwidth is wasted using this approach and more data can be transferred in

the same time. Before transmitting, the data will be scrambled using a self synchronous scrambler with a 58-bit polynomial and eventually the receiver will unscramble the data [18].

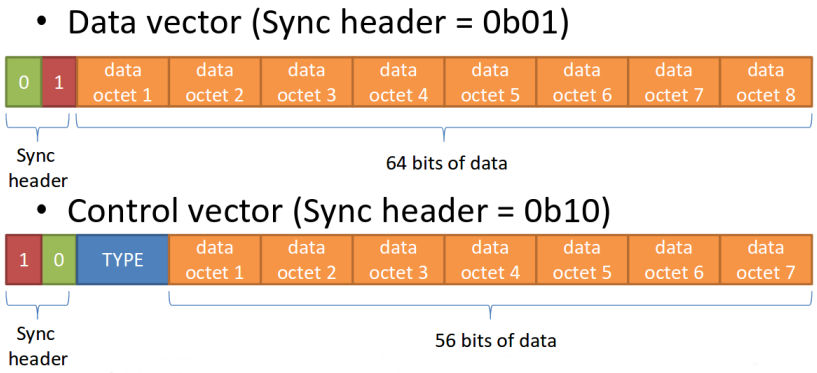


Figure 7: Headers used for 64b/66b encoding.

In Figure 7 two different data blocks are visible [19]. The upper one is just a block containing the eight bytes of data. The lower block contains the control commands. It consists of seven bytes and a two bit type indication. This structure could vary depending on the protocol but it gives a general overview of what the two types of encoded vectors/blocks look like.

Technologies who adopted this way of encoding are for example 10/100 Gigabit Ethernet, InfiniBand and Thunderbolt.

4.3 128b/130b - PCIe 3.0/4.0

This builds further on the 64b/66b encoding but now with twice the payload. It doesn't make use of self-synchronous scrambler and uses a different scrambling polynomial. The two preamble bits still have the same functionality as the case with 64b/66b encoding. The encoding results in less overhead and more bandwidth can be used to transmit data bytes.

This way of encoding is for example used in PCI Express 3.0 and 4.0 which can reach line rates up to respectively 8 and 16 Gbps [20].

4.4 128b/132b - USB 3.1

Very alike with the earlier discussed 128b/130b encoding but with four preamble bits instead of two. This way of encoding is primarily used by the USB 3.1 protocol. The preamble is still used to indicate whether the transmitted information is a block containing data or a block that contains control commands. The first two bits or the preamble are set to indicate a control block and the last two are used to indicate a block containing data.

The two extra bits in the preamble have been added to decrease the error-rate compared to earlier USB versions. The protocol will stay running if a single bit flips and can correct that flip. When two or more bits flip, the protocol will go to into recovery. Same would be the case with a single bit when the two preamble bits were used [21].

4.5 256b/257b - Fibrechannel

This a way of encoding results in a very big packet of data to be transmitted. Four incoming 66 bit blocks are converted into a single 257 bit block. This is done by a of course removing some of the bits but without influencing the actual data and control commands. In Figure 8 a transmission packet containing exclusively data is visualized [22].

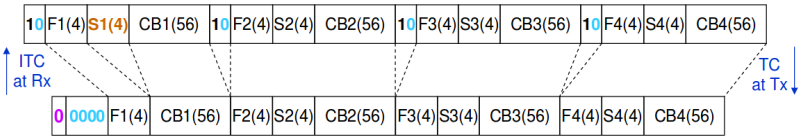


Figure 8: The header of a 256b/257b transmission containing data.

When four eight data byte blocks in a row have to be transmitted, it is fairly simple to put these in a single 257b block by removing all the preambles and putting a single preamble bit in front of the transmitted block. A one indicates that only data blocks are being transmitted and a zero indicated control block or a mix of data and control. Figure 9 shows a transmission packet containing data and control blocks [22].

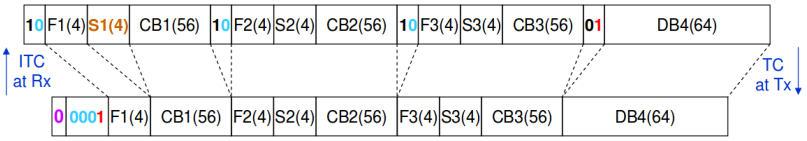


Figure 9: The header of a 256b/257b transmission with mixed blocks.

When only control blocks or both variants are sent in a single block. The first bit will be zero and the following four bits will indicate which bytes are data or control commands. The first nibble of a command block will be removed to make room for the four additional bits. This way the block can be transmitted with less than 2% overhead.

This way of encoding is used by FibreChannel in the 32G and 128G variants [23].

4.6 64b/67b - Interlaken/SerialLite

This is nearly identical to the structure used in 64b/66b encoding. The only difference is the addition of an extra bit to the preamble. While 64b/66b encoding completely depends on the scrambler to prevent DC base wandering, this way of encoding also has the ability to invert the data that will be transmitted. If this data has the tendency to excessively increase or decrease the line voltage, the data pack excluding the preamble can be inverted. This inversion is indicated using the additional bit which is referred to as the inversion bit. This makes the decoder clear whether the data is has been inverted and the data will always leave the decoder with it's original polarity.

Analysis has shown that the scrambler 64b/66b encoding trusts so much on, is not perfect and can still cause DC base wandering over time. This is why the 64b/67b encoding has been developed. The only protocols using this way of encoding at the moment of writing are the Interlaken protocol [7] and Altera/IntelFPGA SerialLite [8].

4.7 Scramblers

It should be taken in account that long streams of ones and zeros create a constant voltage. This could be a harmful phenomenon while synchronizing a communication system because there is a short on transition states. Several ways of encoding were developed to overcome this problem but nonetheless without success since they actually didn't solve the problem but provided workarounds. One of the workarounds was to limit the maximum length of a possible string only containing ones or zeros. Of course this wasn't a direct solution to the existing problem but it was the easiest way to get the transmission suited for long-distance communication.

A more appropriate solution was the so called scrambler. A scrambler randomizes the data input sequence in such way that it is nearly guaranteed there will be no long streams which only contain ones or zeros. After this scrambling the data is still recoverable with a descrambler that will be placed at the receiving side.
Unfortunately the scrambler cannot completely guarantee the prevention of such a long stream of ones and zeros but it will minimize the probability.

Implementing a scrambler provides even more benefits. Beside the prevention of a DC component it provides good error detection, good synchronization capability, eases clock recovery performance and offers the possibility to send data over long distances [17].
A physical scrambler is realized by implementing a parallel register and several XOR gates.

5 FPGA vendor dependent protocols

The first protocols to look at, are these delivered by the FPGA vendors. Unfortunately this introduces some complications because the protocol has to function on every FPGA, whatever vendor these are from. Vendors often offer IP-Cores which are only compatible with their specific FPGA's. This could have several reasons like optimization of the hardware structure for certain models, which only the manufacturer knows of course, and which specific parts are included as a hardcore instead of a softcore. Also lots of effort and man hours went into the development of these IP-cores which makes it understandable they are vendor dependent. Nonetheless we will look at these protocols because it is important and interesting to know what the vendors have to offer.

5.1 Xilinx Aurora

Xilinx Aurora is an existing communication protocol which is as indicated developed by Xilinx and only applicable for Xilinx FPGA's [24]. Aurora is designed to ease the implementation of multi-gigabit transceivers in a project and besides that provides a light-weight user interface on top of which designers have the possibility to build a serial link. It is very useful in situations where serial point-to-point connectivity is required like chip-to-chip links.
The protocol comes in multiple variants, one of them makes use of the older 8b/10b encoding [25] and the other makes use of 64b/66b encoding. This section will be primarily focused on the 64b/66b variant from which a complete IP Product Guide is available [26].

The Aurora protocol has a very high throughput that can vary from 0,5 Gbps to over 400 Gbps depending on the amount of available transceivers and their maximum speed. Data can be transferred in simplex or full-duplex mode. Figure 10 shows a block diagram of the protocol [26].

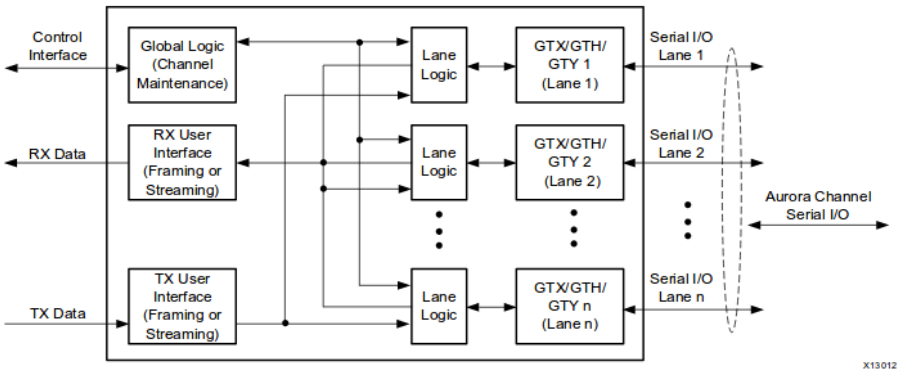


Figure 10: The Aurora block diagram.

Every lane contains a separate logic lane where encoding, decoding and error detection happens. The global logic monitors all lane logic modules and is responsible for channel bonding. Aurora is also capable of CRC encoding/decoding and flow control.

The CRC32 will be calculated for each individual lane on the valid bytes to be transmitted. There are different variants of flow control to choose from. User flow control (UFC) for example allows the applications to send high-priority messages to each other while native flow control (NFC) allows receivers to regulate the speed at which data will be received.

Immediate mode makes it possible to insert idle codes within data frames while completion modes inserts idle codes between complete data frames.

5.2 Altera/IntelFPGA Serial LITE

Altera/IntelFPGA has it's own serial streaming protocol [8] SerialLite is meant for high bandwidth chip-to-chip, board-to-board and backplane communications. The protocol comes in two variants. The older SerialLite II and the newer SerialLite III. This section will focus on the newer variant of the protocol.

SerialLite can be run in two modes. The continuous and burst mode which both are useful for different applications. In the first mode data will be continuously transmitted without gasps. This is useful when a simple high bandwidth interface is required. The second mode will transmit the data is bursts across the interface. Very useful for applications which send a lot of data at certain moments. Like for example raw digital video content where lines of the display raster can be sent in bursts.

The protocol can be used in simplex and duplex mode which is very useful when needed. Speeds up to 28 Gbps can be reached which is of course very dependent on the transceivers connected to the specific FPGA. Channel bonding with up te 24 lanes is supported which results in possible bandwidth to over 300 Gbps. Transmitted data will be encoded by 64b/67b. All data will also contain CRC-32 error correction.

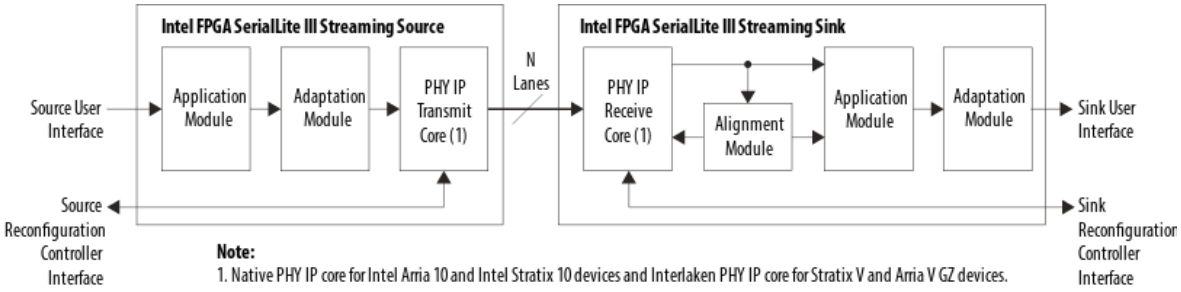


Figure 11: The SerialLite III block diagram (Simplex mode).

In Figure 11 the block diagram of the protocol in simplex mode is shown [27]. The documentation also advertises with low overhead and point-to-point transfer latency. To save soft logic resources it is possible to make use of the hardened Native or Interlaken PHY IP core depending on the model used. The mentioned Interlaken protocol will be discussed in the next section.

5.3 Microsemi LiteFast

Microsemi has also developed it's own serial point-to-point link. It is intended as a low-cost, scalable, light weight and high-speed solution. It also provides built-in flow control and in case no data is transmitted link activity will be maintained.

The protocol includes data framing and in case no data is available idle frames will be transmitted. The data payload will be covered by a CRC-32 packet included at the data frame end. Channel bonding is added as an optional feature and the user can choose between 1, 2 or 4 lanes per SerDes. This supports speeds varying from 4 up to 10 Gbps in case 4 lanes are used. In case multiple lanes are used, the ability to align these is

included. Unfortunately the protocol still uses 8b/10b encoding which offers quite the overhead compared to the newer ways of encoding. A quick overview of the interface can be seen in Figure 12 [28].

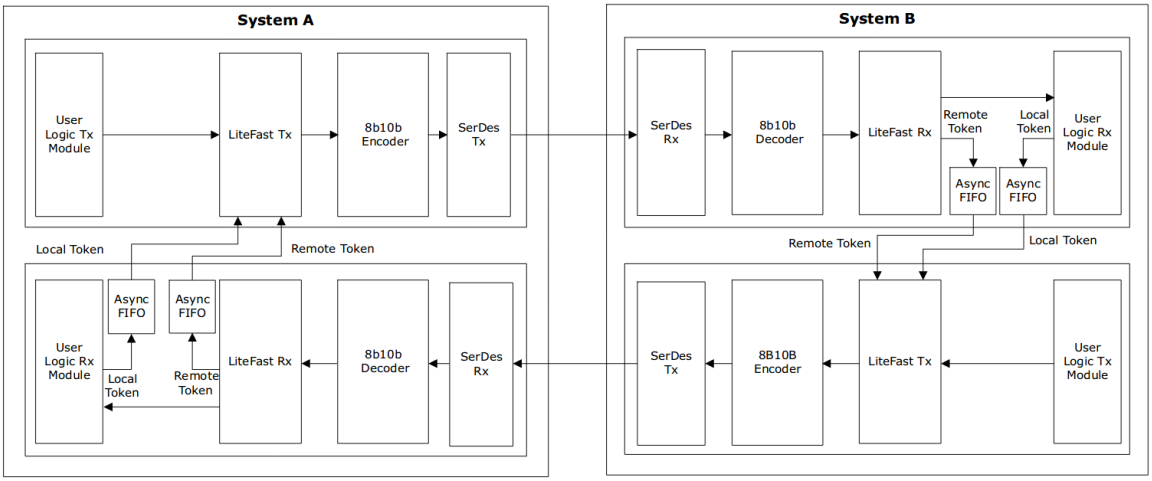


Figure 12: The LiteFast block diagram (Single Lane).

5.4 Conclusion

To conclude this section, a bullet point overview of the protocols is added. This will mention the features of both vendor dependent protocols and compare them with each other.

- Both protocols offer line rates which will push the transceivers to their limits. Aurora mentions about 26 Gbps and Serial Lite III about 28 Gbps per transceiver
- Aurora features flow control while Serial Lite III is not clear on this. (In contrast Serial Lite II does support flow control)
- Both offer CRC-32 error correction, FEC is not mentioned in the documentations
- Aurora uses 64b/66b encoding while Serial Lite III implements 64b/67b line encoding
- Both offer excellent channel bonding up to 16 (Aurora) or 24 (Serial Lite) channels which results in speeds to over 300 Gbps

6 Vendor independent protocols

This section contains a survey on frequently used protocols which offer high bandwidth and reliability. It will be described how the protocols function and how their specifications compare to the requirements mentioned earlier. This section will conclude with a short summary on the discussed protocols and which of them will be the most suitable according to their specifications.

6.1 The Interlaken Protocol

The Interlaken protocol is a narrow, high speed channelized chip-to-chip interface [7]. Interlaken consists of two fundamental structures. These are the data transmission format and the Meta Frame.

The data transmission format relies on the concepts of SPI (System Packet Interface) 4.2 which is a protocol used for data transfer between a link layer and a physical layer [29]. Multiple bursts will contain the data, thus this will not be sent in a single pack. These will be subsets of the original packet data and will be sent sequentially. Before and after each burst, a control word is placed. These words contain important instructions, for example error detection or just an indication of the start or end of a packet. The segmentation of data also allows interleaving of data transmissions from different channels. This can be implemented for low-latency operation.

To transmit the data over a SerDes infrastructure, the Meta Frame is defined. This contains four unique control words to provide lane alignment, clock compensation, scrambler initialization and diagnostic functions. These frames run in-band with the data transmissions, using formatting to distinguish it from the data. In Figure 13 the word type structures are depicted [7].

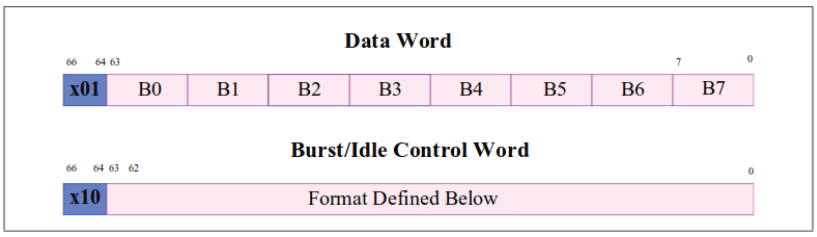


Figure 13: Word formats Interlaken makes use of.

The Interlaken protocol provides a handful of important features: [7]

- Support for 256 communications channels, or up to 64K with channel extension
- A simple control word structure to delineate packets, similar in function to SPI4.2 [29]
- A continuous Meta Frame of programmable frequency to guarantee lane alignment, synchronize the scrambler, perform clock compensation, and indicate lane health
- Protocol independence from the number of SerDes lanes and SerDes rates
- Both out-of-band and in-band per-channel flow control options, with a simple Xon/X-off semantic
- 64b/67b data encoding/decoding and scrambling/unscrambling
- Performance that scales with the number of lanes

Interlaken also features error correction in the form of a 24-bit CRC. Instead of the 64b/66b encoding 64b/67b has been chosen to prevent DC balance or baseline wandering. This adds a little overhead but prevents possible bit-errors and complications in the circuitry the receiver contains.
Forward Error Correction has later been added as an extension and offers Reed-Solomon (544,514) encoding. The protocol definition also mentioned the additional overhead. Line coding itself adds about 4,7% overhead and the RS FEC extension will add an additional about 2,7% to this which results in 7,5% overhead [30].

Both Xilinx [31] and Altera/IntelFPGA [32] developed IP-Cores based on the Interlaken protocol which are capable of achieving huge speeds, 150 Gbps and 300 Gbps respectively. This applies when channel bonding is used. Both vendors claim that a single lane can provide 12,5 or even around 25 Gbps. These speeds would be sufficient according to the requirements and even offer more speed when needed. According to the documentation these cores have recently been updated so they are still maintained which is a good indication.

6.2 SATA protocol

Serial-ATA is a communication protocol that evolved from Parallel-SATA [33]. Nowadays the technology is often used to for the connection of hard drives [34].

Unfortunately the speed of SATA 2 is limited to 3 Gbps and SATA 3 can reach speeds up to 6 Gbps. The maximum speed of SATA 3 is around 0.6 GB/s, the speed current SATA SSD's also reach their maximum read/write speeds. SATA-Express is a newer variant of the SATA standard but didn't change much. It is part of the SATA 3.2 standard and actually is just a connector to combine SATA and PCIe [35]. An overview of the SATA Express architecture is depicted in Figure 14 [36].

Thanks to the PCIe support, SATA Express has the ability to use the newer NVMe driver [36]. Unfortunately this is a whole different protocol and actually has nothing to do with the SATA protocol itself, which doesn't reach the required speed and is unsuited for this application.

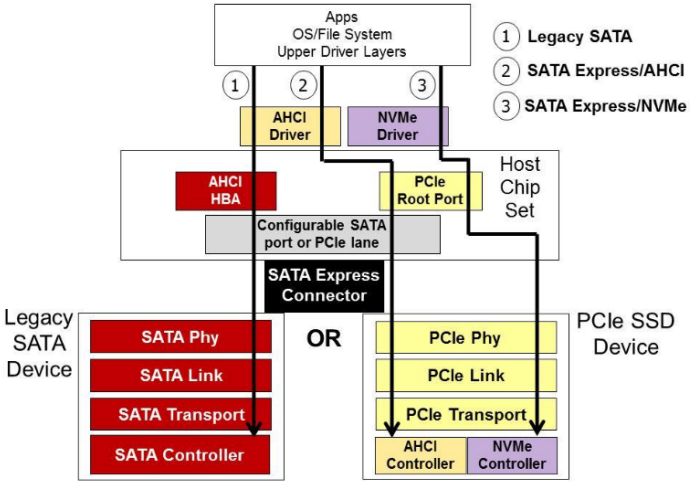


Figure 14: Overview of the SATA Express architecture.

6.3 CPRI

Common Public Radio Interface is an initiative protocol which was meant to define a publicly available specification [37]. This would standardize the protocol interface between the radio equipment control (REC) and radio equipment (RE) in wireless base-stations. It is designed with an optical or copper transmission line in mind.

CPRI offers high lane rates up to 24,330 Gbps while using a serial connection. The two most common ways of encoding are used, 8b/10b and 64b/66b are supported. 8b/10b maxes out at 9,8 Gbps and from there on 64b/66b starts to increase the line rate up to te earlier mentioned 24 Gpbs. Speeds of 10,1 and 12,2 Gbps are also possible.
According to the documentation the physical layer is designed in such way that bit errors are very uncommon. This is why error detection is not directly included in the framing or encoding but is more an optional feature. Detection of sync header violation is used to detect link failures in case this occurs.
Unlike CRC which is not mentioned, Forward Error Correction is an optional feature in this protocol.

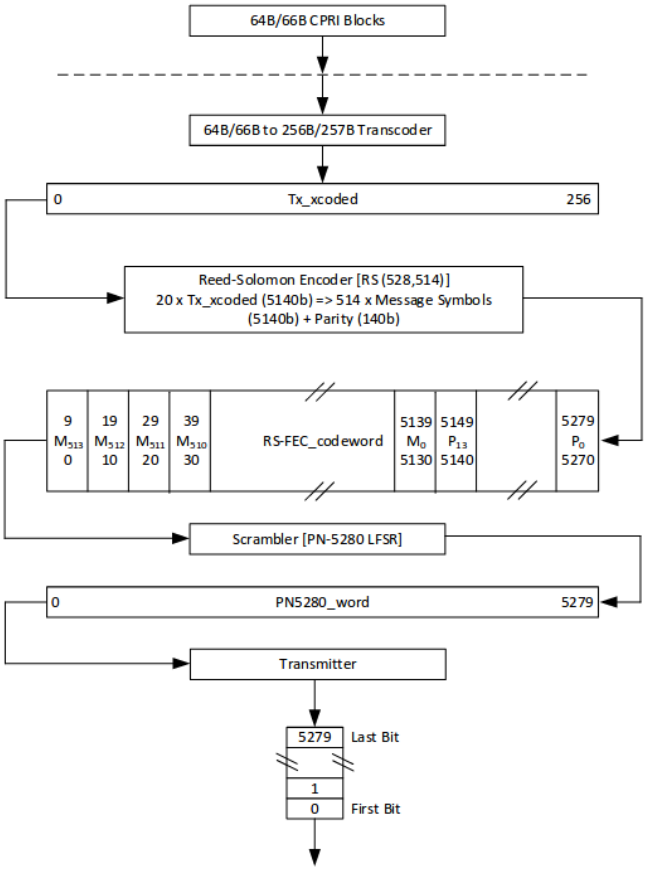


Figure 15: Overview of the CPRI architecture using FEC.

In Figure 15 the architecture of CPRI uses is visualized [37]. It is clearly visible that the data is packed into 66b blocks thanks to the 64b/66b encoder. But these CPRI packets are put into a block of 257b. This is like the earlier explained 256b/257b encoder. These are encoded again by the RS(528,514) FEC, scrambled and transmitted.

There are two different protocols available for the Control and Management (C&M) information exchange. The slow High level Data Link Control (HDLC) and the faster Ethernet variant. Unfortunately flow control is only available to use for the slow C&M channel. Selecting HDLC or Ethernet is purely optional. The specifications recommend to support at least one non-zero C&M channel bit rate on a link [37].

Altera and Xilinx both developed their IP Cores for the CPRI protocol.
So in short CPRI offers the required speed and delivers line rates up to 24,330 Gbps. It doesn't mention CRC but offers the addition of FEC. Flow control is support but only when using HDLC. Channel bonding is not mentioned and the range depends on the cabling.

6.4 HyperTransport

This is a frequently used packet-based, high-bandwidth, scalable, low-latency point-to-point interconnect technology [38]. The purpose of this technology was to increase the communication speed between integrated circuits in for example computers, servers and embedded systems. Hereby the number of buses is a system can be kept at a minimum, which can reduce the occurrence of possible system bottlenecks. HyperTransport is an open standard which is royalty-free managed. Before implementing HyperTransport a license is required which can be provided by the developers. Figure 16 [39] shows the current versions of HyperTransport and their specifications.

Special Features	HT 1.x	HT 2.0	HT 3.0	HT 3.1
Max Clock Speed	800 MHz	1.4 GHz	2.6 GHz	3.2 GHz
Max Aggregate Bandwidth (32-bit links)	12.8 GB/s	22.4 GB/s	41.6 GB/s	51.2 GB/s
AC Operation – Capacitive Coupling (optional) with AC/DC Autosensing, Autoconfiguration	No	No	Yes	Yes
Link Splitting (un-ganging) Each HT Link Split into 2x Half-Width Links	No	No	Yes	Yes
Hot-Plugging	No	No	Yes	Yes
Dynamic Link Clock/Width Adjustment	No	No	Yes	Yes
DirectPackets™ Data Streaming - +16 Virtual Channels (22 total), Peer-to-Peer Support, Native Packet Handling	HT 1.1 only	Yes	Yes	Yes
PCI Express Mapping	No	Yes	Yes	Yes
Common Features				
2 x Unidirectional, Low-Voltage-Differential-Signaling Links per HT Bus				
Scalable Link Width (2-bit to 32-bit)				
Asymetric Link Support (different link widths)				
Asynchronous Link Operation				
Clock Forwarding (no SerDes latency penalty)				
DC Operation (direct signal coupling)				
PCI, PCI-X Mapping				
Priority Request Interleaving™ (lowest I/O latency)				
Virtual Channels Support (6)				
Error Retry				

Figure 16: HyperTransport versions and their specifications.

The bandwidth of this protocol is very high but it should be taken in account that this is the case for a 32-bit link. It is possible to make use of the HT point-to-point link using a PCI bus. Unfortunately HT has it's own connectors. An open-source HyperTransport IP-core has been developed which offers a bandwidth of 1,4GB/s which results in 11,2Gbps [40]. Xilinx used to have it's own IP-Core for HT but unfortunately this has been discontinued. Altera

also abandoned it's HT Core and does not recommend use of this IP in new designs. Another paper has been written on a HT3 Physical Layer Interface for FPGA's but unfortunately this reaches proven speeds up to 1600Mbps. The authors claim possible link speeds up to 12,8GB/s but not proven [41].

6.5 Fibre channel

Fibre channel as the name indicates is a communication protocol with optical data transmission kept in mind. The Fibre Channel Industry Association claim it to be a reliable, cost-effective and capable of high speed data transmission [42]. The recent release variant named 64GFC should reach a throughput of 12,8 GB/s which is sufficient according to the requirements. 128GFC offers a throughput of 25,6 GB/s but is just four parallel 32GFC lanes.
One of the notable things about Fibre Channel is that it makes use of 256b/257b encoding. It packs four control/data words in a single package and adds a header that can be the size of one or five bits. After that a Reed-Solomon Encoder, RS(544,514), is added so it contains FEC. It distributes the symbols and eventually transmits the data using PAM4 technology [43].

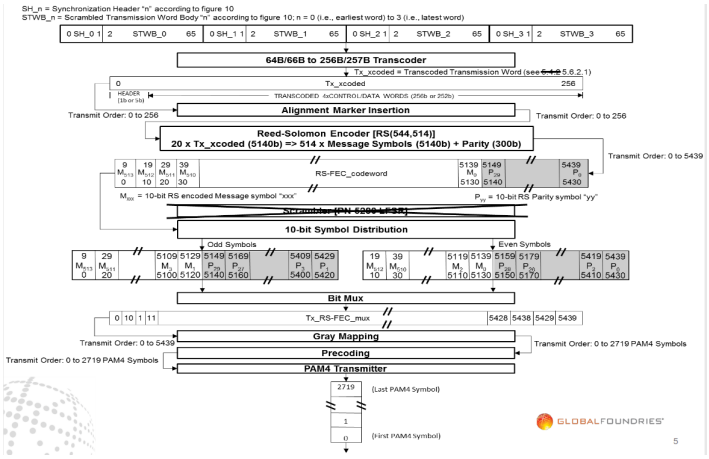


Figure 17: Proposed architecture of 64GFC.

Figure 17 [43] shows a proposed architecture but it's not 100% certain that this is exactly the same as how it functioned since the official release. However the chance is fairly high it is and slides have been updated through time. Of course this architecture has been well thought through since the source is Global Foundries.

There is not much documentation to be found on the protocol so unfortunately a clear description has not been stumbled upon. Xilinx had an IP-Core developed in 2010 for Fibre Channel but unfortunately this IP has been discontinued [44]. A newer variant has been released in 2016 but is only available on request and contains an encrypted RTL [45].

6.6 XAUI

XAUI is a modern chip interface that came along with the innovations of the 10 Gigabit Ethernet Task Force. The name came into existence by merging AUI from Ethernet At-

tachment Unit Interface and X (10 in roman numbers) which represents 10 Gigabit. It is designed to extend the XGMII (10 Gigabit Media Independent Interface).

The serial bus used for this interface has a low pin count and is self-clocked. It provides 2,5 times the speed of usual Gigabit Ethernet. The purpose of this protocol is to combine four of these lanes to a single 10 Gbps line [46].

This unfortunately concludes that it is not possible to use a single lane to transfer around 10 Gbps of data. In addition there is still a 8b/10b encoder which can be substituted by a 64b/66b encoder. This could prevent the rather big overhead that comes with 8b/10b encoding. One of the biggest downsides is that the protocol lacks flow control which is an absolute must have [47].

Altera/IntelFPGA, Xilinx and Lattice have developed IP cores to implement XAUI in a simple way on your FPGA. Broadcom released HiGig which is used to enhance the XAUI PHY. It makes use of different headers and increases the speed to 6,375 Gbps per lane [48].

6.7 Conclusion

This section will conclude which of the earlier described protocols is more suitable and whether or not it meets the requirements. Table 1 provides a quick overview of their specifications. In case a yes is noted, this means support is available but the exact specifications have not been described clear enough. When there is a '-' noted, there is no support or documentation has not been clear enough to provide the required information. All mentioned protocols and links to their documentation can be found in Appendix B.

	Interlaken	SATA	CPRI	Fibre channel
Lane rate	25,3 Gbps	6 Gbps	24,33 Gbps	12,8 Gbps
Encoding	64b/67b	8b/10b	64b/66b	256b/257b
Flow control	Yes	Yes	-	-
Range distance	Cable dependent	Short	Cable dependent	Cable dependent
CRC	CRC-24/32	CRC-32	-	Yes
FEC	RS(544,514)-Ext	-	RS(528,514)	RS(544,514)
Channel bonding	Upto 400 Gbps	-	-	-

Table 1: Overview of the most suited protocols.

SATA is an interesting protocol but the line rate is insufficient. HyperTransport is great for huge bandwidths but implements a parallel bus while serial transmission is required in this case. Fibre channel looks like an interesting alternative. Unfortunately the lack of documentation and not being open will bring a lot of risks with it.

CPRI is another very interesting option offering a high line rate and a good way of encoding. Unfortunately the unclear documentation on CRC and flow control plus the lack of channel bonding cause this option to be a less suited option. Nevertheless a protocol to keep in mind.

The Interlaken Protocol comes out best. While having excellent documentation, the protocol also meets all the requirements. Even the optional/ nice to have specifications. Interlaken is open to use and is even promoted to use by Cortina Systems and Cisco Systems.

7 The Interlaken Protocol

After the survey of available protocols and their specifications, the Interlaken protocol appeared as the one meeting nearly all requirements in contrast to the other surveyed protocols [49]. Not only the specifications looked propitious for the current situation but the documentation also offered lots of information which made it a lot clearer how the protocol functions and how to possibly realize this in hardware.

Until now the Interlaken Protocol has been described in short but this section will be dedicated to a more extended description of the Interlaken protocol. As described earlier the protocol can be divided into different components which belong in their specific OSI layer. A complete Interlaken frame is formed in the first two OSI layers. The frame itself containing the data and control words is formed in the second layer while the first layer adds the three bit preamble that comes with the encoding.

7.1 Overview

This section will present a complete overview to clear up how the protocol is structured and in what order the functions will be discussed. The earlier presented Figure 4 can be used as reference in this case. The Interlaken also functions is this order but the blocks get somewhat complexer.

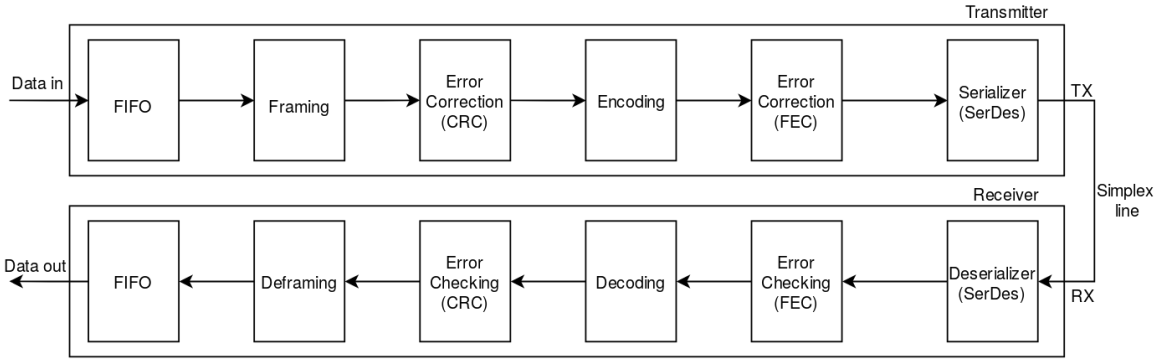


Figure 4: Protocol overview. (repeated from page 17)

Interlaken uses two ways of framing which both add control words. First bursts will be formed by adding framing words and these will be covered by the first CRC and after this the meta frame will be formed which will be covered by another CRC. This can be interpreted as the framing and error correction blocks repeating once. Because of this the control word itself will be explained first and after this the two ways of framing will be discussed in more detail. Flow control and it's supported variants will also be described. This will be followed by the CRC variants Interlaken makes use of since all components containing CRC have been described.

The encoding will consist of a scrambler and the 64b/67b encoder itself which will be described respectively. FEC is available as an extension and will not be described yet since it is not included in the standard protocol description.

7.2 Control Word Format

It is essential to know how the control word is structured to understand how Interlaken handles data transmission. There are two different types of control words used depending on the control bit status. In case this is a one it concerns an idle/burst control word and in case this is a zero it concerns a framing layer control word. Figure 19 depicts the word formats [7].

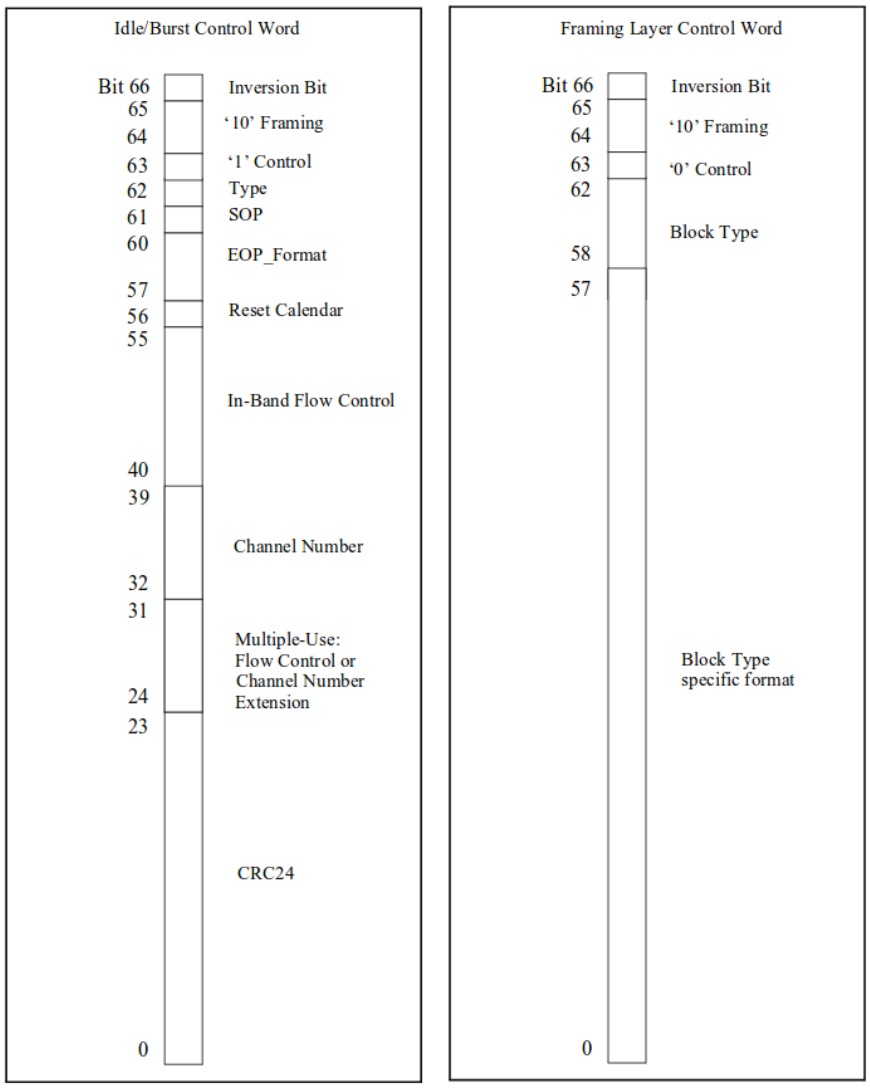


Figure 19: Interlaken control word formats.

As seen the two idle/burst control words contain a type bit which indicates whether the word is an idle or burst word. It includes Start and End Of Packet (SOP and EOP) indicators which will be explained later. Flow control and CRC are also present in this word and there is space for the specific channel number in the case of bonding.

The framing word has a less complex structure and contains a block type field which functions as an identification. After this the specific format according to the block type itself will follow. The structure of this last data block differs for each type and this will be explained further in section 7.4.

For the sake of clarity Figure 20 contains a diagram which depicts all word types and shows how they are distinct from each other. This makes it easier to distinguish the words in one overview compared to repeatedly looking up the images and tables depicted in the Interlaken documentation.

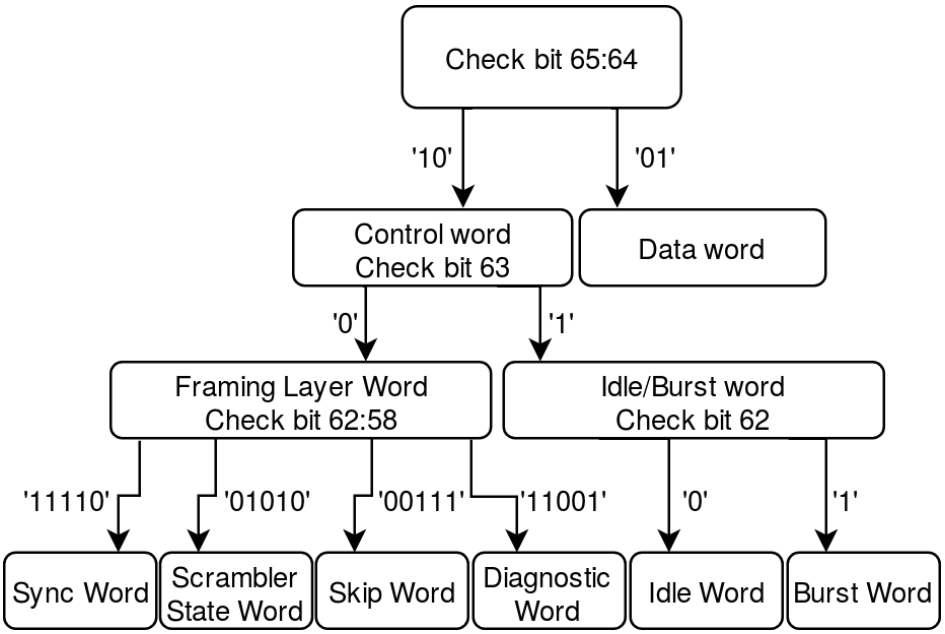


Figure 20: A complete overview of word types and their structure.

7.3 Bursts OSI Layer 2

The Interlaken interface is designed to transmit data in packets. Incoming data will firstly be packed in so called bursts which are of a specific length. The burst will always start with a control word containing the Start Of Packet (SOP) bit set. After this the data will follow and the burst will end with a control word containing the End Of Packet (EOP) bit set. This is only the case when data can be packed in a single burst. In case the amount of data to be transmitted is greater than the data that fits in a single burst of maximum length, multiple bursts will be transmitted with control words in between to still keep the data in separate bursts. These words will have neither the SOP or EOP bits set.

In case the to be transmitted data consists of less bytes than the minimum amount required for a burst, this won't cause problems. The first part of the burst will be filled up with the data. After it a control word with the EOP bit set will follow and the other bytes will be filled with idle words to reach the minimum burst length. This situation is depicted in Figure 21 [7]. A total of 72 bytes have to be transmitted and the maximum burst length is 64 bytes so the first burst is completely filled. The burst after it will contain the last 8 bytes but this is not enough to fill the minimum length of a burst. So the first eight bytes will be put in a burst pack and the other required bytes are filled with idle words. In this case the first idle word will contain the EOP bit set.

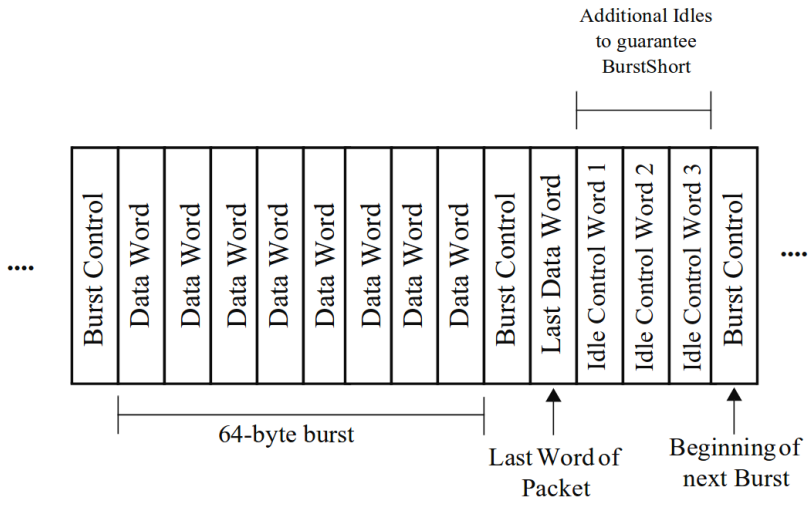


Figure 21: An example of a short burst.

The maximum recommended size of the bursts which is mentioned as BurstMax contains 256 bytes. The minimum recommended size of a burst is 32 bytes and is known as BurstShort [50]. Every byte amount in between can also be transmitted, this has to be an eight byte increment of course since the data packs are all 64-bit. BurstMax and BurstMin can be configured by the designer who implements the burst controller.

Unfortunately a lot of bandwidth is wasted in case idle words are added. A solution for this complication is mentioned in the Interlaken documentation. An additional variable BurstMin is introduced which in size is half that of BurstMax and is bigger or equal to BurstShort. When the payload to be sent is bigger than BurstMax but smaller than BurstMax plus BurstShort this means that too much idle words will be used again. So in this

case a payload of BurstMax minus BurstMin will be sent. This way it can be guaranteed that the last data to be transmitted is enough to fill up BurstShort.

In the same case of the 72 byte transmission it is now possible to prevent the presence of idle words. The total to be transmitted data is smaller than BurstMax and BurstMin summed and bigger that BurstMax alone so two transmission bursts are required. Burstmax and BurstMin are assigned a value of 64 and 32 bytes respectively. The solution is simple by deciding the first burst will contain BurstMax-BurstMin bytes, so 32 bytes will be packed. After this 40 bytes are still remaining and will be transmitted in a burst without the necessity of idle words. The bursts are visualized in Figure 22.

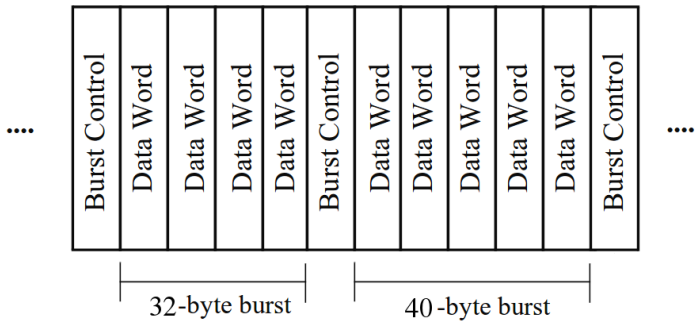


Figure 22: An example of a burst without idles.

As seen this results in a significant reduction of wasted bandwidth. In the first example fifteen packets were transmitted including the idle words. With the optimization added this occupied size has been reduced to 12 packets which saves 24-bytes or 20% of bandwidth.

Another important aspect is that data and control integrity is ensured by the generation of a 24-bit CRC. The control word which follows the data will contain the CRC24. All data words the burst contains and the control word itself will be covered by the CRC24.

7.4 Meta Framing OSI Layer 2

The Interlaken Protocol introduces a way of framing using the term Meta Frame. This introduces four control words which are used in combination with the payload to form a complete frame containing essential information for the receiver. The payload in this case is the data which was packed in bursts while Synchronization, Scrambler State, Skip and Diagnostic words are added. The structure of a Meta Frame is visualized in Figure 23.

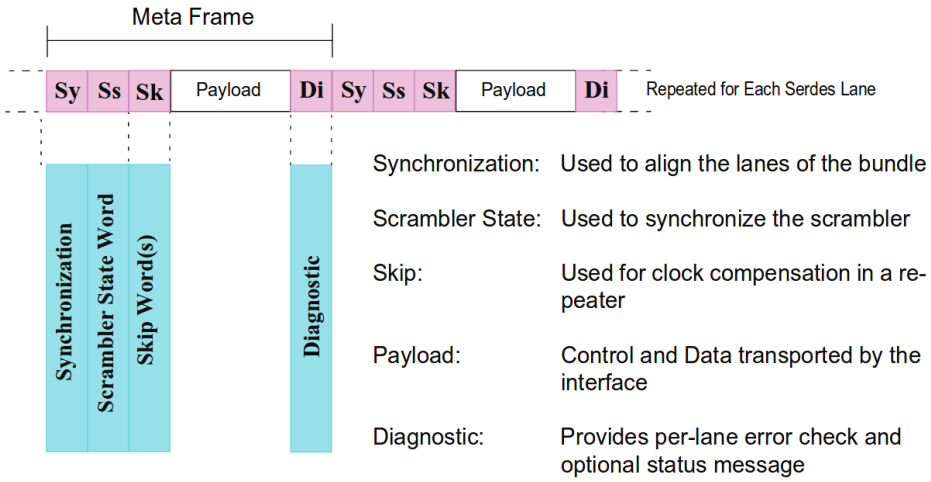


Figure 23: Interlaken Meta Frame structure.

The total amount of bytes the Meta Frame contains is fully configurable and using a variable called MetaFrameLength. However an amount of 2048 words is recommended by the Interlaken Alliance [50]. This also includes the Synchronization, Scrambler State, Skip and Diagnostic words. It is also mentioned that this frame length is a real limit, so the meta framer doensn't wait for a burst to finish. The framing words can appear at any moment during a burst. This also indicates the components responsible for generating the bursts and meta frames don't have to communicate about this. The control words and their block types are shown in Figure 24 and will be explained in their own dedicated sections.

Meta Frame Control Word	Block Type (positive disparity)	Block Type (negative disparity)
Synchronization	011110	100001
Scrambler State	001010	110101
Skip	000111	111000
Diagnostic	011001	100110

Figure 24: Interlaken framing layer block types.

While the Meta Frame control words take up a total bandwidth of 32-bytes every frame, this doesn't really increase the amount of overhead since they appear infrequently. When for example the length of a Meta Frame is chosen to be the recommend size of 2048 words, 2044 of these words will effectively carry the payload. Leaving generated overhead by the bursts out of account, this results is an overhead of about 0,2%.

7.4.1 Synchronization and Scrambler State

The synchronization and scrambler state words are unique in the aspect they are the only words that must be transmitted unscrambled. The synchronization word is a static word both known to the transmitter and receiver. One of it's purposes is to lock the scrambler and literally synchronize the transmitter and receiver. After receiving the 4th consecutive synchronization word, the scrambler is locked and can descramble data.

Another purpose of the synchronization word is to align multiple lanes in case channel bonding is used. The sync word will be transmitted simultaneously on all lanes and the receiver recognizes these words and will measure the skew between all lanes. Interlaken also features additional logic which can be adjusted to compensate for skew across lanes so all data lanes will be aligned nearly perfectly. Unfortunately Interlaken doesn't describe this logic and the implementation is left to the designer. A visual representation can be seen in Figure 25.

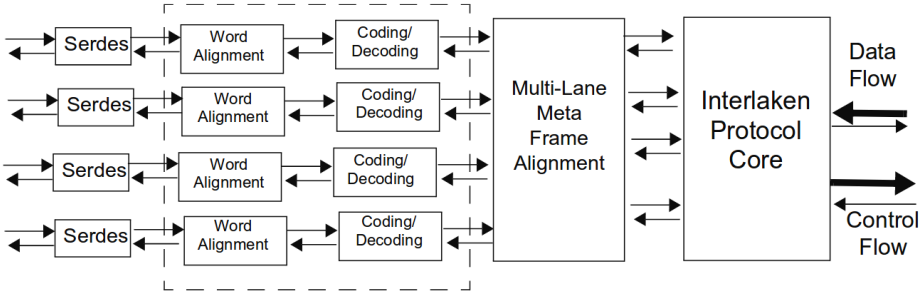


Figure 25: Interlaken lane alignment.

The scrambler state word is used to compare the current state of the receiving side scrambler to the state it has to be according to the transmitter. When these words match, all data has been descrambled correctly. If this is not the case something went wrong and this will result in an error after three consecutive mismatches. This will also cause the scrambler to lose it's lock and to reset.

The synchronization word containing it's valid pattern and scrambler state word are shown in Figure 26.

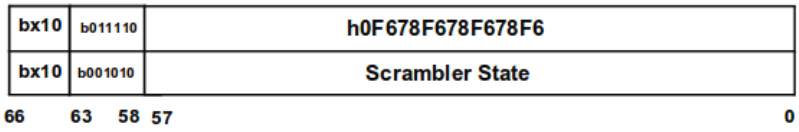


Figure 26: Interlaken synchronization and scrambler state words.

7.4.2 Skip Word

The skip word is used enable to the ability of clock compensation in case a repeater stands in between the transmitter and receiver. The clock rate can slightly differ on each side of the repeater which results in corrupt data on the receiving side. Adding skip words is very useful in this situation because these can later be removed by the repeater or more skip words can be added. This way the differing clock rate can be compensated.

The structure of a skip word is depicted in Figure 27 and as seen this contains just a static package of bits. Skip words can be placed nearly anywhere is the complete Meta Frame. Except in between the diagnostic, synchronization and scrambler state words. It is also possible to add multiple skip words at different positions in the Meta Frame.

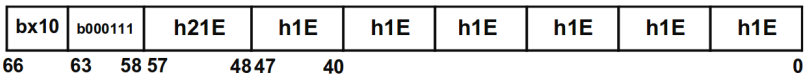


Figure 27: Interlaken skip word.

7.4.3 Diagnostic Word

This word type is meant to indicate the current lane status and offers error correction for the lane this message is received on. The diagnostic word structure is depicted in Figure 28.

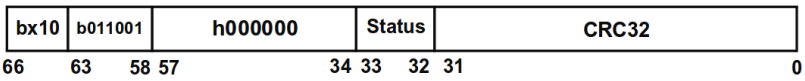


Figure 28: Interlaken diagnostic word.

The 2-bit Status field contains two different messages. One bit indicates the health of this lane and the other bit represents the health of the entire interface. When the bits are high this indicates a healthy interface while a low bit gives indication a problem occurred.

Error-correction is added in the form of a 32-bit CRC. So every diagnostic word contains a CRC-32 field which covers all previous data and the diagnostic word itself. More detailed information will be present in section 7.6.

7.5 Flow Control OSI Layer 2

Interlaken provides documentation on multiple ways of flow control that can be implemented. Communication will be through per-channel backpressure. This indicates that the receiver will hold off the transmitting device on sending packages in case no more data can be processed. When the receiver has solved the problem transmission can continue where it left off. This can occur in case the receiving buffer is nearly completely filled or the receiver lacks processing power. Out-of-Band, In-Band and Full-Packet Flow Control are featured by Interlaken.

A so called calendar can be used in the case of flow control. This is simply a structure to which channels may be mapped. It can be used to map the flow control to any set of calendar entries or to provide link-level flow control. In the last case a binary one would mean permission to transmit data (XON) and a zero would indicate transmission has to cease immediately (XOFF).

7.5.1 Out-of-Band Flow Control

An Out-of-Band Flow Control option is defined to support systems that require simplex operation. So in case data flows only in one direction. To overcome this and still feature flow control, a separate channel is required. This can be made possible by an extra physical connection and the advantage of this technique is that the full bandwidth is available on the main data transmission channel.

The interface makes use of three signals. A clock, the flow control data and a synchronization indicator. Figure 29 [7] shows a visual representation of this. The data is synchronized to the clock signal while the separate synchronization signal indicates the start of a new calendar.

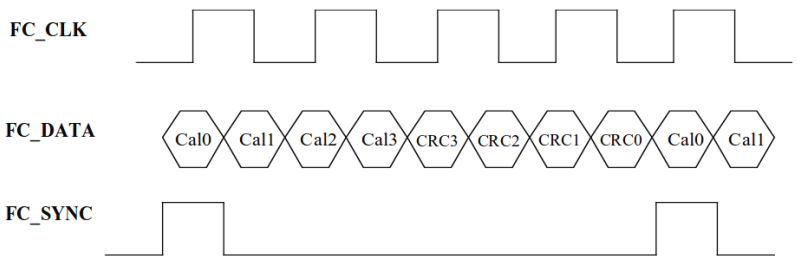


Figure 29: Interlaken Out-of-Band Flow Control timing diagram.

In this case a 4-bit calendar is used but this of course depends on the number of channels implemented. To ensure flow control data integrity a 4-bit CRC is used which section 7.6.1 will discuss in more detail.

7.5.2 In-Band Flow Control

This makes use of the data channel to transmit the flow control status. This is an option provided for systems that support full-duplex operations and saves physical connections. The flow control calendar will be moved to the space in the idle/burst control words which is intended for this use.

There are 16-bits available but the 8-bit multi-use field can also be used for flow control which brings the total amount of usable bits to 24. The reset calendar bit is available

to synchronize the moment at which the calendar starts. In case this bit is not set, the calendar will not be reset and continue where it left off in the previous control word.

Since the idle/burst control words are covered by the CRC-24 this will render the earlier mentioned CRC-4 unnecessary to implement.

7.5.3 Full-Packet Flow Control

This way of flow control is optimized for usage while complete package transmissions, without any interleaving, are required. Two interpretations of the full-packet mode flow control are given.

The first method is to stop the transmission immediately after receiving the XOFF message. This reduces the required size of the receiving buffer but causes head-of-line blocking of other channels.

The second interpretation is to finish the current packet before stopping the transmission. This increases the required buffer size and prevents head-of-line blocking of channels.

It is possible to choose one of the two interpretation but is also possible to combine them in a way if required. This depends on what kind of behavior the application requires.

7.6 CRC generation OSI Layer 2

Interlaken covers different parts of the to be transmitted data with separate CRC polynomials. This subsection will be dedicated to how the CRC's have to be generated according to the Interlaken documentation and which part of the transmitted data they will cover.

The generation of an n-bit CRC will start with the polynomial being reset to all ones. After this the data stream will enter the component and thus generate the CRC. This will be sent to the CRC function with the MSB of the bytes always entering first. When this is done, the polynomial will be inverted and moved to the reserved space in the right bit order. To keep things consistent the CRC will be moved using the same format as the data itself.

Three different CRC polynomials are documented by the Interlaken protocol. This concerns a 4-bit, 24-bit and 32-bit polynomial. The next sections will provide more information on what data they will cover, how they will be calculated and what polynomial will be using.

7.6.1 CRC-4

Out-of-band flow control data integrity is ensured by a 4-bit CRC generation. This covers up to 64 bits of data used for flow control. The polynomial chosen for this CRC variant is 0x0D in hexadecimal form. The complete polynomial in equation form is also written down.

$$X^4 + X + 1$$

In case In-Band Flow Control is used this 4-bit CRC won't be necessary since the Flow Control information will then be included in the Idle/Burst words.

7.6.2 CRC-24

The data bursts will be covered by a 24-bit CRC. The data packets and the control word itself will be covered. The Idle/Burst control words contain a reserved space where the generated CRC-24 can be moved to. Of course the CRC field will be padded with zeros while the CRC is being generated.

The polynomial of CRC-24 can be written out as 0x328B63 in hexadecimal form. The equation below represents the complete polynomial.

$$X^{24} + X^{21} + X^{20} + X^{17} + X^{15} + X^{11} + X^9 + X^8 + X^6 + X^5 + X + 1$$

7.6.3 CRC-32

The Meta Frame will be covered by a 32-bit CRC. This will cover the complete payload in a frame and the Synchronization, Scrambler State, Skip and Diagnostic words. Since the CRC-32 will be included in a Diagnostic word at the same frame that has been covered, the bits of where the CRC-32 will be placed have been padded with zeros. The Scrambler State word will also be filled with zeros since generating the CRC has to happen before scrambling the data. Framing bits will of course also be excluded since these have not been added yet and this is really meant to check the data itself.

The CRC-32 is generated for every individual lane which offers the advantage that errors can be traced to a specific lane in case channel bonding is implemented. This could prove a very useful feature in case one of the lanes will cause error's since it is immediately clear which lane is the cause.

The polynomial Interlaken uses for the implementation of CRC-32 is 0x1EDC6F41 in hexadecimal format. The complete polynomial in equation form is also written out.

$$X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$$

A visual representation of the CRC-32 calculation can be seen in Figure 30. The Scrambler State words and placeholder for the CRC-32 will be padded with zeros while the CRC is generated.

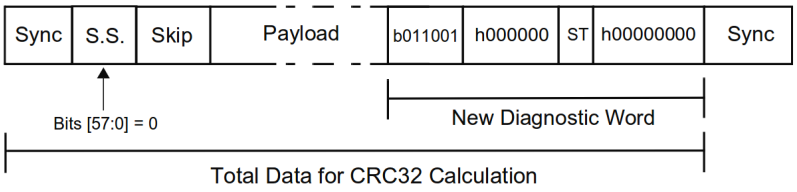


Figure 30: Interlaken CRC-32 calculation.

7.7 Scrambler OSI Layer 1

While other protocols may choose to implement a self-synchronous scrambler, the Interlaken protocol makes use of an independent synchronous scrambler. A self-synchronizing scrambler offers the great advantage it doesn't require constant synchronization but this comes with a disadvantage. When an error occurs the scrambler will replicate this error multiple times because it makes use of two feedback taps. This way even correct data will arrive corrupted at the receiving side which is absolutely undesirable. This is the primarily reason Interlaken has implemented another form of scrambling. The independent synchronous scrambler merely uses multiple XOR gates to generate output data.

The scrambler make use of a 58-bit polynomial which is visible beneath and accepts a 64-bit input. The polynomial can be interpreted in hexadecimal value as 0x400008000000000.

$$X^{58} + X^{38} + 1$$

The polynomial is activated after resetting the device and won't have to be reset anymore. This also makes clear why the scrambler state word has to be transmitted at the start of each Meta Frame. The current state of the polynomial will be compared to the state it should be according to the transmitter. This way it can be ensured the descrambled data is identical to the data before scrambling at the transmitter side.

All to be transmitted words will be scrambled except for the synchronization and scrambler state words. In this case the scrambler will be put on hold and the polynomial state won't change since these words have to be transmitted unscrambled. When the descrambler starts after a reset, the first scrambler state word should be used to descramble the incoming data. This is also one of the reasons these two words have to be transmitted unscrambled, otherwise they can't be read because the descrambler is not yet synchronized for the first time or there is a chance they will be descrambled to an incorrect word and there is no way to detect mistakes in the descrambler state.

It is recommended to reset all scrambler states to a different value on each lane. This minimizes the cross-talk between lanes but this choice is up to the designer. While this is important for the transmitting side, it is not necessary to transmit the reset state to the receiver since short after this reset the scrambler state will be send to synchronize the scrambler and descrambler.

7.8 Encoder OSI Layer 1

Interlaken makes use of the 64b/67b encoding as already explained in subsection 4.6. Two of these bits will indicate the presence of a data or control word like used in the 64b/66b encoding. The third bit or actually last by of the encoded packet will cause an inversion of the complete word when set. It is clearly documented in the Interlaken Protocol Definition that 64b/66b encoding completely relies on the scrambler in case of DC-balancing. This comes which the risk of unwanted increases in the bit-error rate after certain time periods. An excellent solution to prevent the occurrence of these disadvantages is to use 64b/67b encoding. One bit additional overhead is added but excellent DC balance will be provided. It is important to know that in high speeds communications timings often won't allow a full voltage swing before the next bit is transmitted so causing DC unbalance is done fairly quick.

The encoder keeps track of the running disparity in data packets. When the incoming binary data contains a logic one the disparity will be incremented by one and when the data contains a logic zero the disparity will be decremented by one. This will also be done for the new data following up this packet. In case both data packets contain averaged more ones or zeros then the new data will be inverted to balance the average amount of ones and zeros thus canceling possible DC unbalance. The encoder will always try to keep the running disparity within a +/- 96-bit boundary. The complete preamble used by the 64b/67b encoder is depicted in Figure 31 [7]. Every lane keeps track of it's own running disparity.

Bits [66:64]	Interpretation
001	Data Word, no inversion
010	Control Word, no inversion
101	Data Word, bits [63:0] are inverted
110	Control Word, bits [63:0] are inverted
All others	Illegal states

Figure 31: Preamble of the 64b/67b encoding used in the Interlaken protocol.

The encoder gains lock after 64 consecutive legal sync headers appear at the same position in the incoming data. Since the preamble is 3-bit which offers eight possibilities but only four of them are legal, the occurrence of incorrect sync will be very low. Of course the occurrence of illegal states are possible. If these conditions appear multiple time the encoder will lose it's lock or won't lock.

8 Hardware implementation

The purpose of this assignment was to search and implement the best protocol matching a clear set of requirements. In chapter 7 this best protocol has been found and described in details. This section will focus on the implementation of Interlaken on an FPGA and the hardware provided to test it.

The author has been provided a Xilinx VC707 Evaluation Board [51] by Nikhef to eventually test the implemented design on. The provided board is depicted in Figure 32.

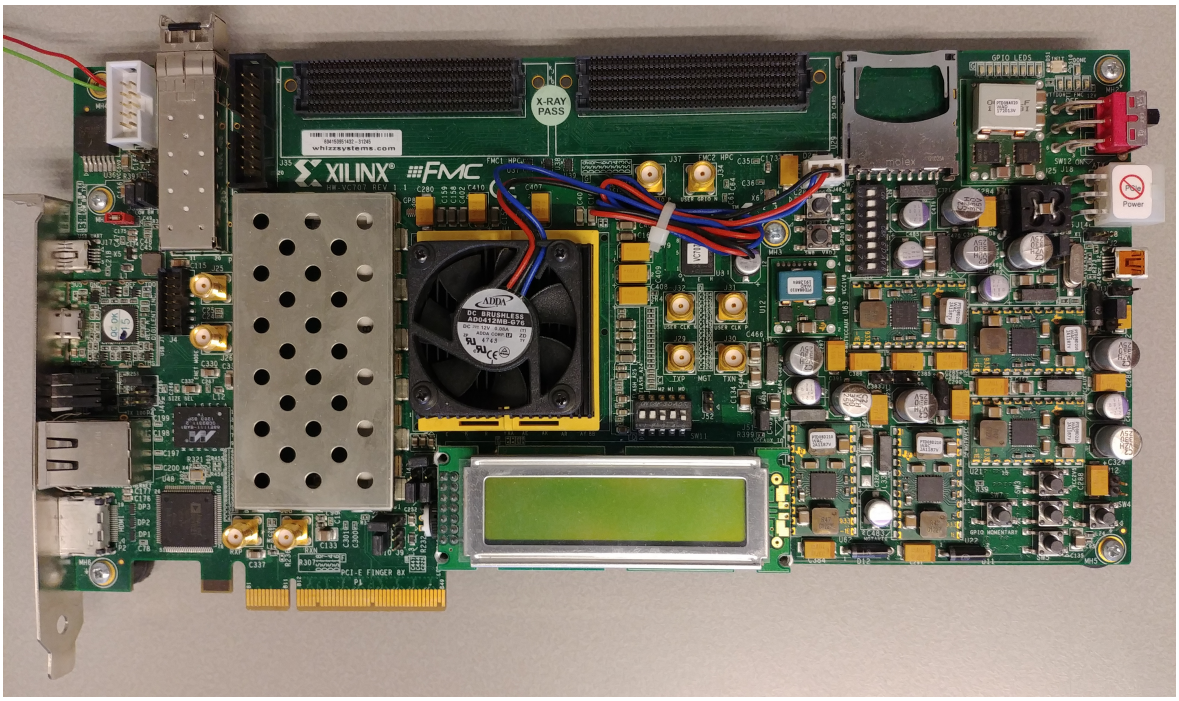


Figure 32: The Virtex-7 VC707 Board provided by Nikhef.

The VC707 Evaluation Board contains 27 accessible GTX transceivers according to the documentation. Eight are wired to the PCI Express x8 connector and sixteen are connected to the FMC connectors. This results in three left from which one is wired to the SMA connectors, another one is connected to the SFP/SFP+ connector and the last one is used in combination with the Ethernet PHY for SGMII connection.

This makes clear that only two of these GTX transceivers are immediately accessible for communication with other boards or products. These are the transceivers wired to the SMA and SFP+ connectors.

The included GTX transceivers support transfer speeds up to 12,5 Gbps in case the QPLL is used instead of the CPLL, which is excellent since 10 Gbps is the target line rate [52]. The difference between PLL types and which clocks they generate will be explained in section 8.4. It is even specifically mentioned that in case of Interlaken a line rate of 10,3125 Gbps would be supported. The QPLL frequency would of course be 10,3125 GHz since all data will be serialized and transmitted over the line.

This Chapter will contain separate sections describing the transmitter, receiver and transceiver parts. In case IP cores are used this will be noted with the accompanying version and vendor.

8.1 Transmitter side

The transmitter side will be described and designed first. This will deliver more insight in the framing and encoding of the data which will make it easier to remove the framing and decode the transmitted data at the receiving side.

Figure 33 depicts the complete transmitter side of the interface. Only useful data will be stored and the FIFO can also indicate it is full to other logic. Framing components have to communicate with each other because of the extra space required in between the data flowing.

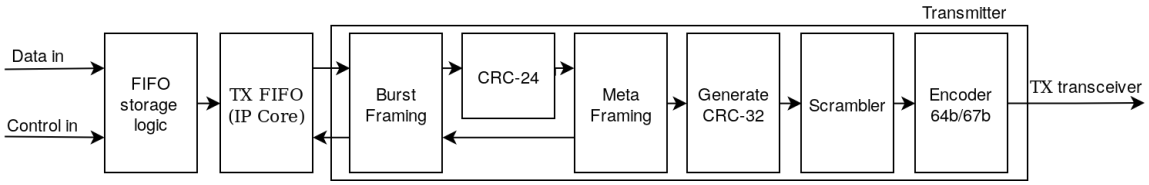


Figure 33: Overview of the TX block diagram.

8.1.1 TX FIFO OSI Layer 2

The FIFO will act as a buffer temporary storing data when frames are added or the interface can process no more data. It is also especially useful in case the interface needs some extra time before the next frame can be processed. This will often occur during the addition of burst and meta frames.

Another very useful feature of the FIFO is the ability to cross clock domains. Data will be written in the FIFO at a by the user determined rate while to FIFO will be read at the maximum speed the transceiver allows to always keep the line and logic alive/busy.

A separate input port is available to the user which enables the write ability of the FIFO. So only data the user really want to be stored will enter the FIFO. Besides this other control signals like the SOP, EOP and EOP valid will be put together with the data in one signal. The single 68-bit variable will then enter the FIFO.

In case the bursts will be generated according to the optional scheduling enhancement, an extra feature will be required to read the amount of data already placed in the FIFO.

The most recent version implements the Xilinx FIFO generator 13.1 IP core.

8.1.2 Bursts OSI Layer 2

Data leaving the FIFO will be converted to complete bursts. The control signal provided by the user which will leave the FIFO with the data, will be read and according to this the bursts will be formed. Then for example an SOP or EOP burst control word can be generated.

In the Burst component a state machine can be found which remains in idle state unless the burst is enabled and a start of packet is detected. This will trigger the state machine and the arriving data from the FIFO will be read. The data will be saved in several pipelined registers to make packing the data in burst words possible. As explained in Section 7.3 a burst control word has to be added first. After this the data will follow and as long as the state machine doesn't detect an EOP this continues. Every word of data processed will also cause an increment by one in the word counter because of the maximum burst length,

BurstMax, that is allowed. When this value is reached the state machine will switch to another state for one cycle and will return to processing the data again. This has been done so a burst control word can be transmitted between bursts of maximum lengths.

When an EOP signal is detected the state machine will switch to another state. Reading the FIFO will be stopped and a control word which contains the EOP and CRC-24 will follow at the output. After this it is important to check the word counter value. When the data length transmitted in this situation is shorter than the predefined BurstShort, the control word will be followed by one of multiple idle words until the transmitted data length is equal to BurstShort and the state machine will again wait for an SOP signal. In case the transmission contained an amount of words in between the values of BurstShort and BurstMax, no idle words will be necessary to include.

There is also a situation possible where the user sends data at a slower rate than the complete interface can process. In this case the FIFO will transmit multiple empty flags during transmission. But the transceiver is still expecting data so the empty spaces are filled up with idle words. This way data is still transmitted and the link is kept alive. In case these idle words won't be used, the FIFO will still output the last value in it's memory. The interface will then just processes this as useful data which results in duplicated data at the receiving side. Plus an RX FIFO overflow since the RX FIFO will be read at the same rate the TX FIFO is filled.

An implementation of the optional scheduling enhancement is recommended but not yet developed.

8.1.3 Meta Framing **OSI Layer 2**

This component will add the meta frames to the data transmission, as discussed in section 7.4 this will be four words. It contains a state machine that counts the passed data words. When the transmission starts, these four control words will appear at the output. During these four cycles data will be read and pipelined so when the control words have passed, data will immediately follow. When the word length of the passed data reaches a value of MetaFrameLength, including the several data words, the state will change.

Firstly the pipelined data will be output so it takes several cycles before all data left the component. After this the framing words will be output again to complete the meta frame and during this process input data will already enter the pipeline registers again. This way the cycle repeats and a complete meta frame will always appear at the output. One last thing to consider is the FIFO will also stop being read for four clock cycles. Otherwise there won't be place for the framing words.

8.1.4 Generating CRC **OSI Layer 2**

Bursts and Meta Frames both contain variants of the CRC and generating this will be explained in this subsection. Both contain the same method since they will check a certain length of words and also the specific word that will contain the CRC itself later.

The component responsible for generating the CRC needs two clock cycles to let this check appear at the output. In this case a method is used where data enters the CRC component but is also saved in original state parallel to the register because this has yet to be transmitted. Since the control words containing the reserved space for the CRC also have to be checked, the data has actually to be held in parallel registers for two clock

cycles. This method makes it fairly simple to put the generated CRC in the control word since this word and the CRC now are available on the same clock cycle. Now it's just a matter of moving the bits to the right position in the control word. Figure 34 shows a visual representation of this.

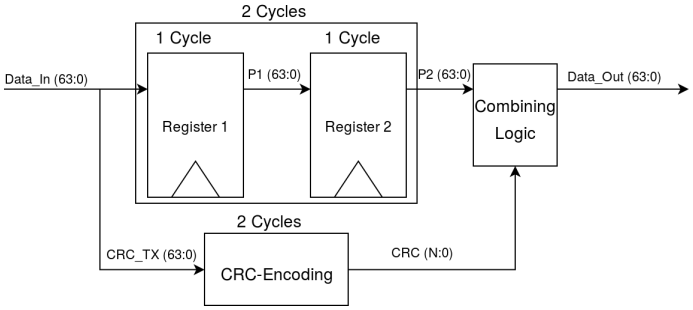


Figure 34: Used method generating CRC.

8.1.5 Scrambler **OSI Layer 1**

The scrambler will receive a 64-bit data input from the meta framing component. Since the synchronization and scrambler state words won't have to be scrambled these have to be detected first. When the control input signal is low this means data is entering the scrambler and this will always be scrambled. In case the control input is high this first six bits will be looked at. This way the scrambler can determine whether the word is synchronization, scrambler state or other control word. The first will leave the scrambler untouched while the second will appear at the output after the current scrambler state has been added to the word. All other control words will be scrambled. The scrambled data will be outputted alongside the control word indicator and a data valid signal.

The scrambler polynomial has been defined before. The Interlaken Protocol Definition already includes a piece of code in Appendix B showing how to constantly generate the output and new state of the polynomial. This code was unfortunately written in Verilog so only the part generating the polynomial has been used and was translated to VHDL.

8.1.6 Encoder **OSI Layer 1**

The encoder will accept the scrambled data output. When enabled all data packets will be added a 3-bit preamble header. The control word input signal will indicate what the first two preamble bits accompanying the word should be. When the word is data '01' will be added and when a control word appears this will be '10'.

For the determination of the inversion bit a separate variable will be reset every clock cycle. This will count the running disparity of the incoming data using a for-loop. The value will be saved and compared to the running disparity value of the data just being transmitted. This data is located in a separate variable.

In case both words contain a majority of the same symbol, the bits of the incoming data will be inverted and the inversion bit will be added to the preamble as a logic high. After this the data will be moved to another variable and will this time be compared to the newly appearing input concerning running disparity.

After this process a 67-bit word will leave the encoder and should be ready for transmission. The transceiver will accept this data and is responsible for the real transmission.

8.2 Receiver side

The receiving side will be responsible for restoring the data to its original form before this entered the interface. During development of the transmitter side many knowledge has been gained which makes it easier to develop the receiver side. While the transmitting side added framing words and encoded data, the receiving side has to decode this again and remove the frames. Figure 35 depicts an overview of the receiver side.

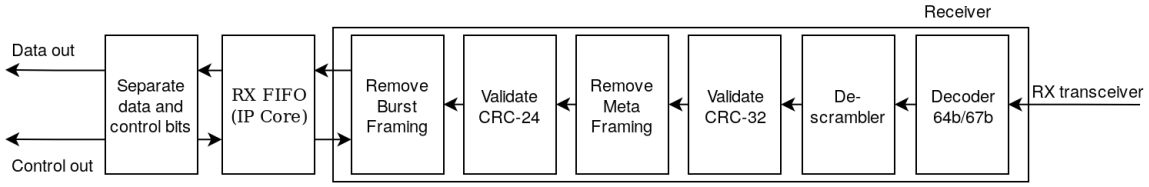


Figure 35: Overview of the RX block diagram.

8.2.1 RX FIFO — OSI Layer 2

The original data identical to the data before entering the interface will be written in this FIFO. Again it is also used to cross clock domains since the interface itself will be running at a standard frequency but the user clock may vary or configure at a different frequency.

Only when the component removing the burst frames output a valid signal, the data appearing at the FIFO input will be stored. Otherwise the data will be ignored since it then concerns a control word or duplicated data which is not useful. This means the valid signal is connected to the FIFO write enable pin.

The user has to set the FIFO read signal high before any data will appear at the output which prevents data loss. In case the user logic is busy with other tasks and can't process the data, this will remain stored in the FIFO. However the chance on the FIFO overflowing is fairly high in such situating. This will be prevented by flow control which will be discussed in 8.3. For this the FIFO programmable full signal can be used which offers a set value and threshold based on how far the FIFO is filled with data.

The most recent version implements the Xilinx FIFO generator 13.1 IP core.

8.2.2 Deframing Burst — OSI Layer 2

The added burst control words have to be removed. However these words contain critical information which have to be read and processed before simply deleting words. For example in case an SOP or EOP is detected, the user interface will output a high SOP or EOP pin.

The deframing can easily be done by inspecting the valid and control indicator which accompany each word. When the valid signal is low the data will simply be ignored. In case valid is high and control is also high, this indicates a valid burst control word. Since different burst control words each contain other viable information, it will first be determined which word this is. According to this analysis the useful information will be extracted. This data will be saved and appear at the output accompanying the next data word. For example an SOP be added to the next word. One exception is the EOP since this actually has to accompany the most recent passed data word.

In case valid is high and control low, the entering word will be considered as valid data. However this data will not yet appear at the output but is saved. This is done because an

EOP can follow anytime and in case the data word is released to early the EOP will miss. When another valid data or control word enters the component, the saved word will be released. This means the possible appeared EOP or earlier SOP will appear at the output of the component together with the data.

Another valid signal at the output will switch to high when the data leaves the component to indicate the FIFO that this data is useful and has to be stored.

8.2.3 Deframing Meta — OSI Layer 2

The meta frames are useful during transmission and deframing but they have to be invalidated in the process of outputting only the original user data. The descrambler uses these frames to synchronize itself and before deframing the CRC-32 has also to be checked. After this the meta frames will simply be ignored by making their accompanying valid signal low at the output. This way other components will just ignore them. Of course when data enters this component with a logical low accompanying valid signal, the data will be ignored.

8.2.4 CRC checking — OSI Layer 2

Error checking will be done generating the CRC again like at the transmitter side. This generated value will be compared to the received CRC value in the control words. In case these match the data arrived flawless. When these don't match data corruption occurred and the data is not identical to that at the transmitting side.

8.2.5 Descrambler — OSI Layer 1

Data leaving the decoder is still in scrambled format and not yes usable. When starting the descrambler it won't process any input data but instead look for the unscrambled synchronization words. The control words indicator from the decoder will be read. In case this indicates a control word and the block type is identical to the synchronization one, the data will be compared to the predefined sync data. In case these are identical, the state machine will move into another state and two counters start. In case the input data valid signal is a logical low, this word will be ignored so the scrambler state will also not change.

The amount of passing words will be counted. After a certain MetaFrameLength amount of words the synchronization word has to appear again. In case this happens the sync word counter will be incremented by one. In case this reaches the value of four, the state machine moves to the next state indicating a lock.

When in lock all words at the input will be descrambled, except the sync and scrambler state words of course. There will still be checked on correct synchronization words and in case this is not identical to the word expected, the sync word error counter will be incremented by one. After this the scrambler state word is expected to arrive. In case this matches the current polynomial the status should remain locked. Otherwise the scrambler state mismatch counter will increment by one.

When the sync word error or the scrambler state mismatch counters reaches the value of respectively four or three, a reset will follow. The descrambler loses its lock and has to look for synchronization words again to get in lock.

8.2.6 Decoder OSI Layer 1

The decoder will immediately receive data from the transceiver. Thus data entering the decoder will be 67-bit wide. This part will be responsible for removing the preamble and reducing the data width to 64-bits. The scrambled data will remain to be descrambled and the preamble will be converted to a separate control word indicator. The inversion bit in the preamble will be read and according to this data will be inverted or not, then the inversion bit will be discarded.

One of the most important functions of the decoder is to alight the data correctly. Since the data is scrambled and cannot be read, the preamble has to be used for alignment. The decoder can lock on the bit transition in the preamble which assures the 64-bits leaving the decoder are really identical to the packet that has been transmitted.

In combination with the transceiver from Xilinx, the decoder can use a separate pin for bit slipping. This will cause the transceiver to change the alignment of data during data processing and changes the preamble position. After every bit slip operation the current location of the preamble will be checked and when this is not located at the last three bits, the slip operation will repeat until the preamble is at the right location. This will also cause the decoder to go into lock after 64 consecutive words contain the right preamble location.

When in lock the amount of processed words will be tracked by using a counter. A single word that is not correctly synchronized will not immediately cause the decoder to lose lock. For this an error counter is used which will keep track of the incorrectly synced words. After sixteen errors the decoder will lose lock and reset. This is determined over all words processed but every 64 words. After these 64 words the word counter and error status will be reset. Fortunately the three preamble bits only effectively use 50% of the valid conditions so errors should not be common and will be easily detected.

The implemented decoder contains three outputs. Two are for the control and valid signals while another one is of course for the data/control word itself. This makes it easy for the components after this to know which type of data this 64-bit word is.

8.3 Flow control OSI Layer 2

It is of great importance to constantly check the RX FIFO status. It can be that the user logic reading the FIFO is busy and cannot keep up with the link speed. In case overflows normally start to occur, data will be lost permanently and this may not be allowed to happen. The solution to this problem is to let the receiver constantly check the RX FIFO status. When the FIFO start filling up more than usual and begins to have a tendency to overflow, there will be a signal generated that will cause the control words to transmit an XOFF for the specific channel. When the side responsible for transmitting the side receives this message, action will be taken by stopping to read the TX FIFO. This way the TX FIFO will fill up somewhat more and the FIFO will update the user logic on this which will stop sending data.

8.4 Transceiver

Separate hardware is required to set up the 10 Gbps link itself. For such link a 10 GHz clock signal is required and of course the parallel data should be serialized to ready it for serial transmission. A great thing is that the transceiver already takes care of this. The only requirements are to configure it correctly and to provide stable clocks.

Because a Xilinx FPGA is used for testing, the Xilinx transceiver will also be used. This will of course have consequences for the whole core being FPGA vendor independent but in this case there is no other solution.

The GTX transceiver in the Virtex-7 contains two types of PLL's (Phase Locked Loops) which generates the clock frequency required for the transmission line. In this case the QPLL will be used since the CPLL is limited at a lower maximum frequency and is less stable. The transceiver connected to the SFP+ connection will be used to transmit the data over fiber. The accompanying GTX transceiver is located at Quad0 and is noted as GTX_X1Y2. The clock the Interlaken Interface itself works on is expected to be $10,0Gbit/64bits = 156,25MHz$.

During startup the transceiver needs several microseconds to configure itself and to lock the QPLL. After this data can be applied to get the other part of the interface locked.

The transceiver also requires a 40 MHz DRP clock which will be generated with the Xilinx Clocking Wizard 5.3 IP core. The most recent version implements the Xilinx Transceiver Wizard 3.6 IP core.

8.5 Complete interface

When all components are combined to one complete interface the overview should look like depicted in 36. This includes the transmitter and receiver logic connected to the transceiver. While the interface only requires the user to input data and control signals with it, certain clocks should of course also be provided. The complete core that has been developed will be accompanied by it's own documentation which will be described in Appendix C.

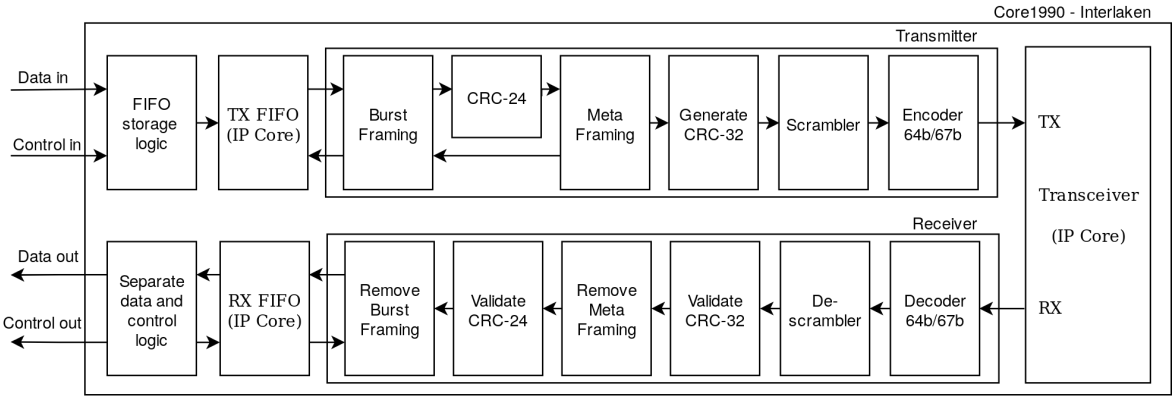


Figure 36: Complete Core1990 architecture.

9 Test runs

Testing is an essential aspect while developing a product. During development of the protocol several tests were run and the successful ones reaching certain milestones will be described. This also gives more insight in the complete development cycle and what the current state of the protocol is. Xilinx Vivado 2016.4 has been used during development.

9.1 Early testing

At the 12th of June 2018 the first design had been implemented and tested on hardware. The core worked, but several bugs appeared and the interface didn't completely behave as intended. Some packets were lost in the transmission and their placeholder were filled with packets that appeared at the position before them. Many duplicates could be seen. Further investigation quickly resulted in the conclusion the framing and deframing of bursts didn't operate flawlessly. However the positive news was that the other components did behave as expected.

After exhaustive debugging and testing of the framing and deframing of bursts, another test run followed at June the 18th. This time the core behaved as expected and this was the first successful test of Core1990!

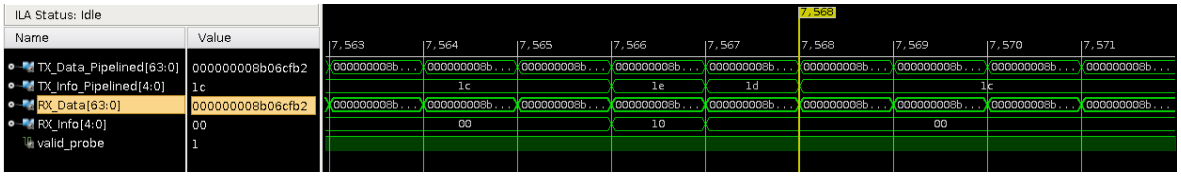


Figure 37: The ILA during test.

In Figure 37 a small fragment of the analyzed data is visualized. The TX data is pipelined to compensate the duration it takes to get data from the TX input to the RX output. To be exact this was 25 clock cycles. The TX and RX data are exact the same, which indicates the data has been transmitted over fiber and was received without any corruption. The valid probing signal also confirms this. Additionally the info signals stand for the EOP, SOP and valid bytes. Here some improvements are still to be made but the most important part is the data itself arriving flawlessly.

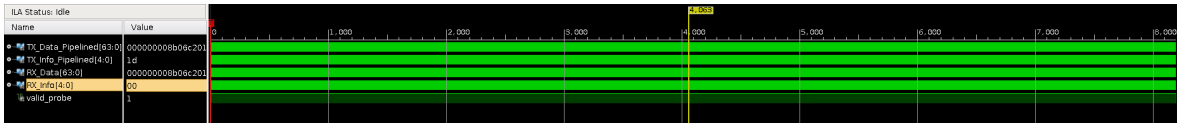


Figure 38: The full sample range of the ILA during test.

Figure 38 shows another run showing the full 8192 samples the ILA could take. During this time the valid probing signal never contains a falling edge which indicates no errors in the data integrity did appear. The test has been performed with the data itself being generated at 40 MHz so this means the data rate itself was 2,560 Gbps but the core was still functioning at 156,25 MHz which means the lane rate still performed at its nominal 10 Gbps bandwidth. The core waits for data and now about every four cycles one new packet was ready. Further testing can be done at higher rates.

In short this test contained burst framing/deframing, meta framing/deframing, scrambling/descrambling and encoding/decoding which all performed flawlessly under test. Generating CRC and checking was also included but not analyzed using the ILA so this still has to be proven. At this point flow control is still in early stage so this doesn't count in this test.

9.2 Clock troubleshooting

Earlier test introduced a strange problem while communicating between two boards. Both links were in lock for a while but suddenly after a undefined period of time lost lock while this didn't happen with the loop-back tests. This clearly indicated clocking signals between the boards weren't correctly synchronized and started to mismatch.

The problem was fixed by changing the transceiver clock (GTREFCLK) from the 125 MHz SGMIICLK to a 156,25 MHz reference clock. This was generated by the on-board Si570 clock generator IC. On of the downsides was that the Si570 output was not directly connected to one of the clock inputs of the transceiver but a different IO on the FPGA. However one of the reference clock inputs of the transceiver is physically connected to on-board SMA connectors, this can be used to provide the 156,25 MHz input.

To solve the clock problem, the Si570 differential signal entered the FPGA and was converted to single-ended with an IBUFDS. After this the signal was converted to a differential output with an OBUFDS. This way the clock signal has been routed to IO that has been connected to SMA connectors. An external connection can now be made from the output 156,25 MHz signal to the transceiver inputs. Figure 39 shows a picture of the VC707 board using the Si570 generated clock for the transceiver.

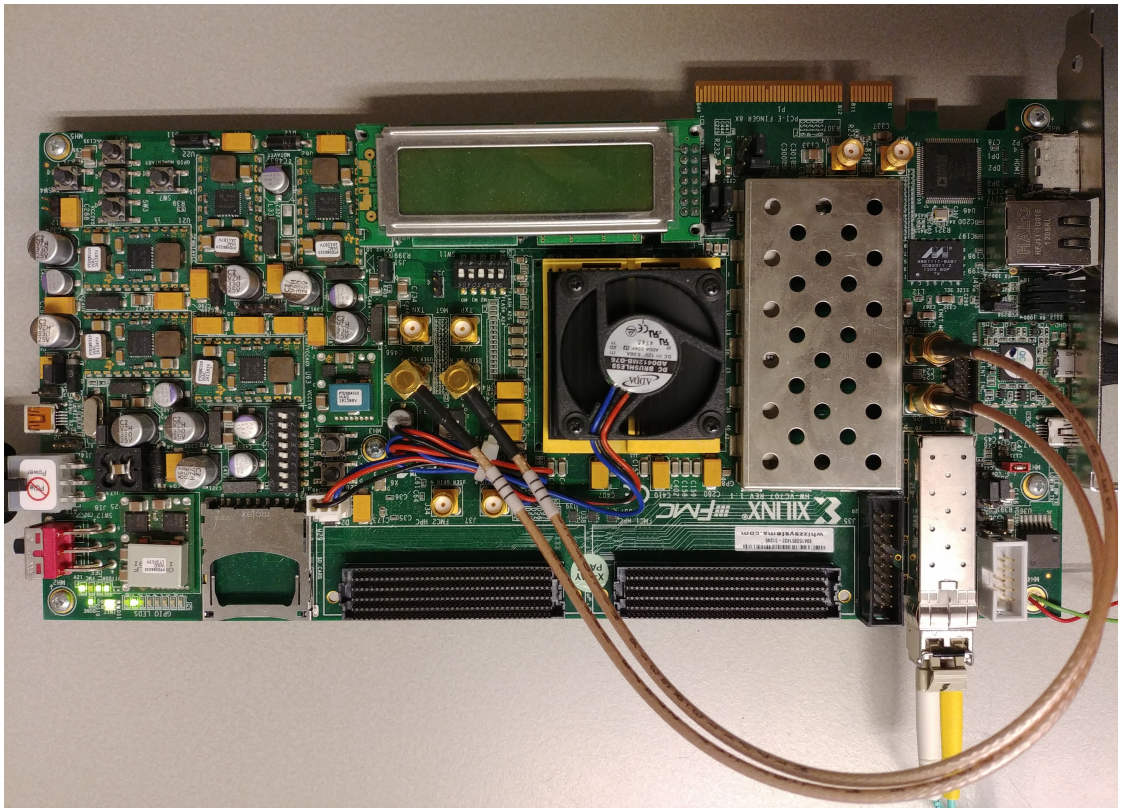


Figure 39: Using the Si570 clock on the VC707.

Schematically this looks like depicted in Figure 40 which shows the connection between the Si570 and external transceiver clock input. All signals have been named according to the original VC707 schematic by Xilinx [53].

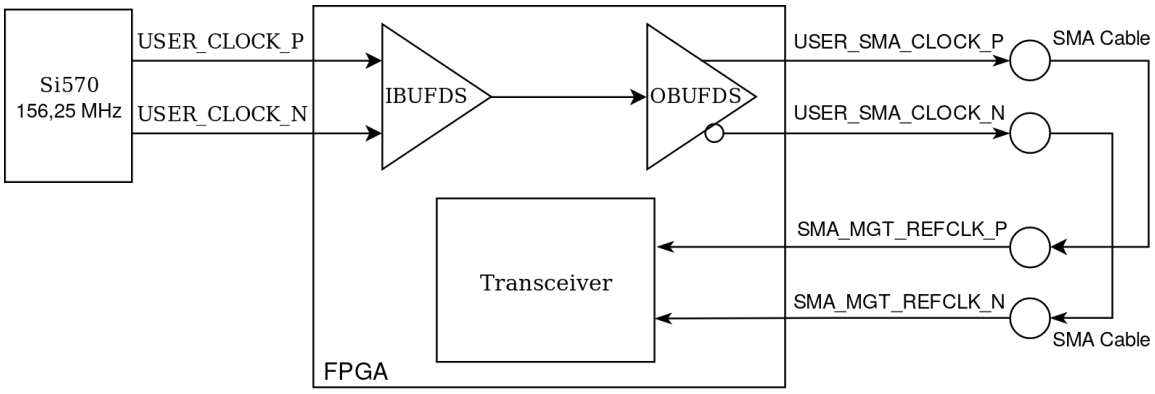


Figure 40: Schematically viewed configuration of the Si570 clock on the VC707.

9.3 Communication between boards

Another run at the 27th of June proved better results. Instead of a single board with a loop-back fiber, now two VC707 board could be used. Because of the newly connected 156,25 MHz reference clock, the link was stable and didn't lose lock. Figure 41 shows the two connected boards. Even unplugging the fiber cable and reconnecting it caused an immediate lock which indicates the link had a self recovering capability.

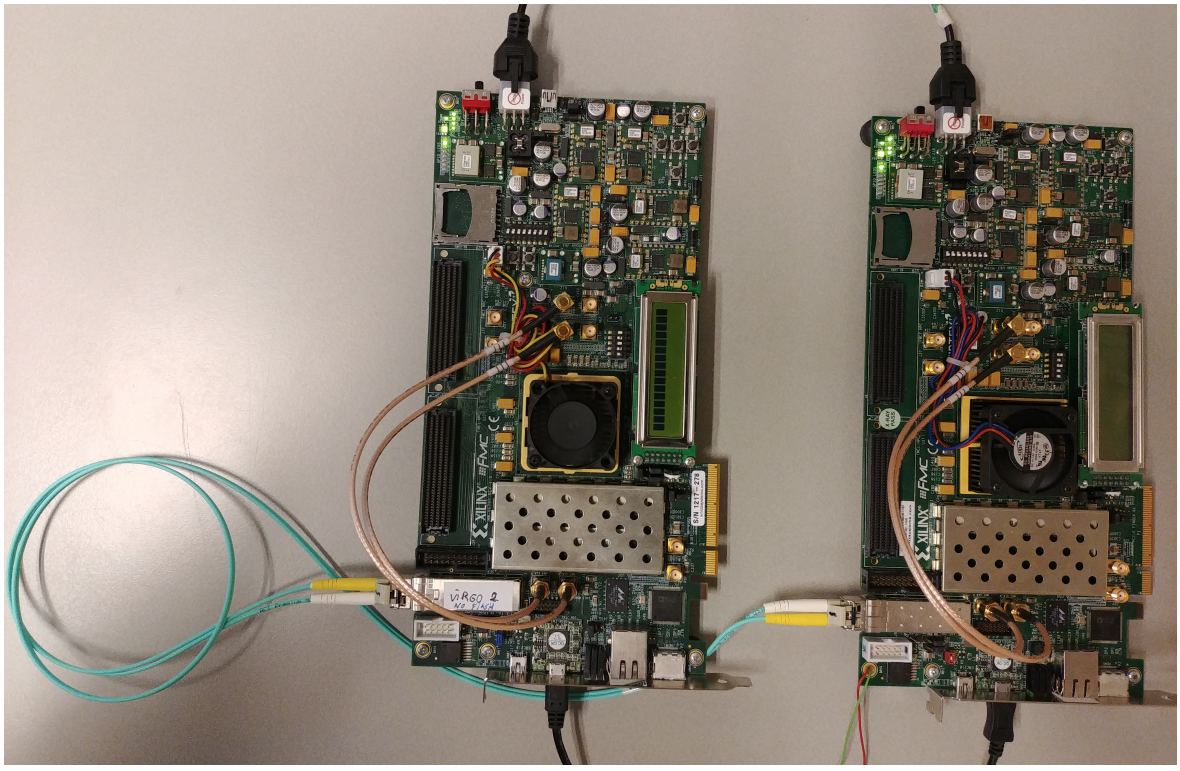


Figure 41: Testing communication between two VC707 boards.

Both boards have a status led that indicated lock of the decoder and descrambler which was always on. A good indication but to be completely sure the lock was very stable, the ILA was used to get a better view.

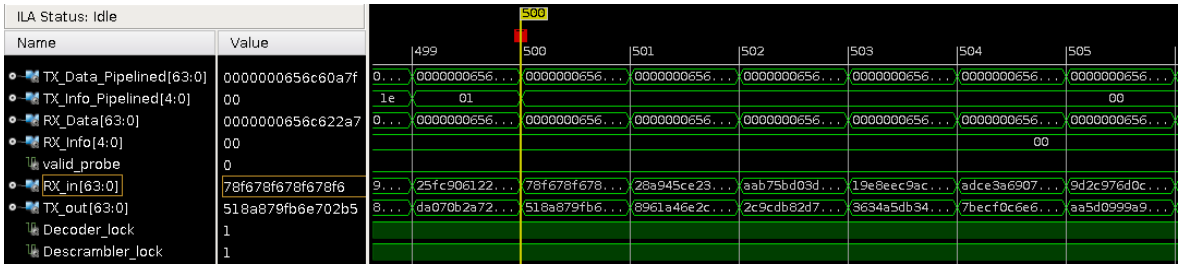


Figure 42: Wave forms captured during the communication between VC707 boards.

A small piece of data viewed with the ILA can been seen in Figure 42. At the yellow marker line a synchronization word being received and followed by the scrambler state can be seen. The data differs a bit but that is because the boards differ some cycles from each other and the longer fiber wire used causes some extra clock cycles delay.

In this case the data generator was connected to a clock of 150 MHz which means the line transferred an amount of 9,6 Gbps on user data. However sometimes the TX FIFO became a bit full and the data generator had to wait. It should also be taken into account the overhead plays a role and 9,6 Gbps plus this overhead could easily require more than the 10 Gbps transfer speed of the transceiver.

While it was not easy to verify, the inspected data by hand was correct and didn't show the data missing. However not everything could be checked by far and this process should be automated for the next run. In case error's appear these will be easier to detect.

For testing this example design with the two boards connected, several thing mentioned underneath were required.

- Two VC707 boards (Including power supply and USB cable)
- Four SMA-SMA (female-female) cables
- Two fiber wires
- Two SFP+ optical - electrical modules

10 Conclusion

To conclude this document it took a while to really take form, it describes a trajectory on understanding how a point-to-point protocol works and then start a survey to find one suited best for specific applications. While the author himself had no experience on any of these protocols or even how point-to-point protocols function, many research proved there are some great features and ideas developed into these protocols.

After the explanation of point-to-point protocols and the survey of the available variants, this document has proven that the Interlaken protocol is the best suited point-to-point protocol available according to the requirements it had to meet. It is royalty-free and provides excellent bandwidth accompanied by many important features.

The Interlaken protocol has been described extensively to completely understand how it works and how a possible implementation could be developed. After this many days/weeks have gone into developing the VHDL code to implement the correct hardware that behaved as expected according to the Interlaken Protocol Definition.

In the end it can be said this was a success. Not all features of the Interlaken protocol are included but basic communication including framing, scrambling, encoding and generating/verifying CRC has proven to be a success. The 10 Gbps target has been reached and far higher lane rates are possible with faster transceivers. While flow control still has to be implemented, the basics are already there. Unfortunately the protocol has only been developed on a Xilinx FPGA meaning it lacks the specific parts to work with an Altera/Intel FPGA product because their transceiver are different. However most parts have been kept as much vendor independent as possible which results in many of the hardware should also be implementable on FPGA's of other vendors.

During this project an Interlaken variant Core1990 was born. The result can be found on OpenCores and hopefully this way of sharing the design will promote it's dissemination and broad adoption. It has been developed with the terms free and open source in mind. This makes it easier for others to see how the protocol works and where improvements are possible.

Still a lot of improvements could be developed like the inclusion of correctly functioning flow control. There is also room for some more testing and especially to ensure it's robustness. The core still has to be tested communicating with a real certified Interlaken machine guaranteeing this really matches the Interlaken protocol as intended. Since the protocol is meant to be vendor independent, it still has to be tested on other devices like Altera/Intel FPGA, Microsemi and Lattice.

This hopefully is the first step of many, introducing the Interlaken protocol open source for anyone requiring high speed data transfer without high costs or constantly paying royalties.

References

[1] Wikipedia, *"Communication protocol"* [On-line] Available: https://en.wikipedia.org/wiki/Communication_protocol [Apr. 03, 2018]

[2] CERN official website, *"Acceleration science"* [On-line] Available: http://cern.ch [Apr. 03, 2018]

[3] Tech-faq, *"The OSI Model - What It Is; Why It Matters; Why It Doesn't Matter."* [On-line] Available: http://www.tech-faq.com/osi-model.html [Mar. 27, 2018]

[4] Truechip, *"Exploring Forward Error Correction Trends in Ethernet"* [On-line] Available: http://www.truechip.net/articles-details/exploring-forward-error-correction-trends-in-ethernet/1909580257 [Mar. 28, 2018]

[5] Amar Shekar, *"OSI Model And 7 Layers Of OSI Model Explained"* [On-line] Available: https://fossbytes.com/osi-model-7-layers-osi-model-explained/7 [Apr. 19, 2018]

[6] Eli Bendersky, *"Framing in serial communications"* [on-line]. Available: https://eli.thegreenplace.net/2009/08/12/framing-in-serial-communications/ [Feb. 14, 2018]

[7] Cortina Systems Inc. and Cisco Systems Inc. *"Interlaken Protocol Definition"* [On-line] Available: http://www.interlakenalliance.com/Interlaken_Protocol_Definition_v1.2.pdf [Feb. 09, 2018]

[8] Altera, *"Intel® FPGA SerialLite III Streaming IP"* [on-line] Available: https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-seriallite3.html [Mar. 29, 2018]

[9] Lammert Bies, *"Introduction to CRC calculation"* [On-line] Available: https://www.lammertbies.nl/comm/info/crc-calculation.html [Feb. 08, 2018]

[10] Joleen Charles, *"Cyclic Redundancy Check CRC Chapter 4"* [On-line] Available: http://slideplayer.com/slide/8190698/ [Feb. 14, 2018]

[11] Dr. Sylvie Kerouédan, Dr. Claude Berrou, *"Turbo code"* [On-line] Available: http://www.scholarpedia.org/article/Turbo_code [Feb. 14, 2018]

[12] Louis E. Frenzel, *"Gearbox operations"* [On-line] Available: https://books.google.nl/books?id=wnGDBAAAQBAJ P.26 [Feb. 12, 2018]

[13] TutorialsPoint, *"Flow Control"* [On-line] Available: https://www.tutorialspoint.com/data_communication_computer_network/data_link_control_and_protocols.htm [Mar. 29, 2018]

[14] Sunsik Roh, *"Design of Out-of-Band Protocols to Transmit UHDTV Contents in the CATV Network"* [On-line] Available: http://file.scirp.org/Html/3-9701557_19481.htm [Feb. 14, 2018]

[15] Altera, *"Using FPGA-Based Channel Bonding for HDTV Over DSL"* [On-line] Available: https://www.altera.co.jp/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01053-using-fpga-based-channel-bonding-for-hdtv-over-dsl.pdf [Mar. 21, 2018]

[16] Knowledge Transfer, *"8b/10b encoding"* [On-line] Available: www.knowledgetransfer.net/dictionary/Storage/en/8b10b_encoding.htm [Feb. 15, 2018]

[17] M. Moussavi. (5 dec. 2011) *"Data Communication and Networking: A Practical Approach"* [on-line]. Available: https://books.google.nl/books?id=gX8KAAAAQBAJ. [Feb. 07, 2018] Cengage Learning, 5 dec. 2011

[18] National Instruments. *"High-Speed Serial Explained"* [On-line] Available: ftp://ftp.ni.com/evaluation/HighSpeedSerial_WP_Final.pdf [Feb. 07, 2018]

[19] Marek Hajduczenia, *"64b/66b line code"* [On-line] Available: http://www.ieee802.org/3/bn/public/mar13/hajduczenia_3bn_04_0313.pdf [Feb. 16, 2018]

[20] Intel. *PHY Interface for the PCI Express, SATA, USB 3.1, DisplayPort and Converged IO Architectures* [On-line] Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/phy-interface-pci-express-sata-usb30-architectures-3.1.pdf P.124 [Feb. 07, 2018]

[21] Synopsys. *"USB 3.1: Physical, Link, and Protocol Layer Changes"* [On-line] Available: https://www.synopsys.com/designware-ip/technical-bulletin/protocol-layer-changes.html [Feb. 07, 2018]

[22] Roy Cideciyan (IBM), *"256b/257b Transcoding for 100 Gb/s Backplane and Copper Cable"* [On-line] Available: http://www.ieee802.org/3/100GNGOPTX/public/mar12/interim/cideciyan_01_0312_NG100GOPTX.pdf [Feb. 16, 2018]

[23] Craig W. Carlson, QLogic Corporation, *"Gen 6 FibreChannel What You Need to Know "* [On-line] Available: http://www.snia.org/sites/default/orig/DSI2014/presentations/StorPlumb/CraigCarlson_Gen_6_Fibre_Channel_v02.pdf (Slide 14) [Mar. 29, 2018]

[24] Xilinx. *"Aurora 64B/66B"* [on-line] Available: https://www.xilinx.com/products/intellectual-property/aurora64b66b.html [Feb. 13, 2018]

[25] Xilinx. *"Aurora 8B/10B"* [on-line] Available: https://www.xilinx.com/products/intellectual-property/aurora8b10b.html [Mar. 29, 2018]

[26] Xilinx, *"Aurora 64B/66B v11.2 LogiCORE IP Product Guide"* [on-line] Available: https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b/v11_2/pg074-aurora-64b66b.pdf [Feb. 13, 2018]

[27] Altera, *"Intel FPGA SerialLite III Streaming IP Core User Guide"* [on-line] Available: https://www.altera.com/documentation/jbz1470383208039.html [Feb. 16, 2018]

[28] MicroSemi, *"UG0701 User Guide LiteFast IP"* [on-line] Available: https://www.microsemi.com/document-portal/doc_view/135971-ug0701-litefast-ip-user-guide [Apr. 23, 2018]

[29] eInfochips Ltd. *"System Packet Interface (SPI) 4.2 IP Core"* [On-line] Available: https://www.design-reuse.com/articles/18135/system-packet-interface-spi-4-2-ip-core.html [Feb. 21, 2018]

[30] Cortina Systems Inc. and Cisco Systems Inc. *"Interlaken Reed-Solomon Forward Error Correction Extension Protocol Definition"* [On-line] Available: http://www.interlakenalliance.com/extension_v1.pdf, Dec. 2016, [Feb. 09, 2018]

[31] Xilinx. *"Integrated Interlaken 150G v2.0 LogiCORE IP Product Guide"* [On-line] Available: https://www.xilinx.com/support/documentation/ip_documentation/interlaken/v2_0/pg169-interlaken.pdf [Feb. 09, 2018]

[32] Altera. *"Interlaken IP Core (2nd Generation) User Guide"* [On-line] Available: https://www.altera.com/documentation/dsu1465510510715.html [Feb. 09, 2018]

[33] Serial ATA International Organization, *"Serial ATA International Organization: Serial ATA Revision 3.0 "* [On-line] Available: http://www.lttconn.com/res/lttconn/pdres/201005/20100521170123066.pdf [Apr. 09, 2018]

[34] Donovan (Don) Anderson *"SATA Storage Technology"* [On-line] Available: https://www.mindshare.com/files/ebooks/SATA%20Storage%20Technology.pdf [Feb. 09, 2018]

[35] Serial ATA International Organization. *SATA Express Specification from SATA-IO in Ratification* [On-line] Available: https://sata-io.org/sites/default/files/documents/SATA%20Express%20In%20Ratification_Final_Website.pdf [Feb. 12, 2018]

[36] Dave Landsman, Sandisk *AHCI and NVMe as Interfaces for SATA Express™ Devices - Overview* [On-line] Available: https://sata-io.org/sites/default/files/images/NVMe_and_AHCI_as_SATA_Express_Interface_Options_Overview_final.pdf [Feb. 12, 2018]

[37] Ericsson AB, Huawei Technologies Co. Ltd, NEC Corporation, Alcatel Lucent, and Nokia Networks. *"CPRI Specification V7.0"* [On-line] Available: http://www.cpri.info/downloads/CPRI_v_7_0_2015-10-09.pdf [Feb. 21, 2018]

[38] HyperTransport Technology Consortium. *"HyperTransport™ I/O Link Specification Revision 3.10c"* [On-line] Available: https://docs.wixstatic.com/ugd/071cb6_53b2dc066f2d4408b5c9368dc447e2f5.pdf [Feb. 21, 2018]

[39] HT consortium. *"HyperTransport Link Specifications"* [On-line] Available: https://www.hypertransport.org/ht-link-specifications [Feb. 08, 2018]

[40] David Slogsnat, Alexander Giese, Mondrian Nüssle, Ulrich Brüning. *"An open-source HyperTransport IP-Core"* [On-line] Available: https://docs.wixstatic.com/ugd/071cb6_1d4e8365b49f4d9f8eab9b1e611ea60e.pdf [Feb. 08, 2018]

[41] Heiner Litz, Holger Froening, Ulrich Bruening, *"A HyperTransport 3 Physical Layer Interface for FPGAs"* [On-line] Available: https://people.ucsc.edu/~hlitz/papers/ht3phy.pdf [Feb. 08, 2018]

[42] Fibre Channel Industry Association (FCIA), *"State of the Fibre Channel Industry"* [On-line] Available: http://fibrechannel.org/wp-content/uploads/2015/10/FCIA_Sol_Guide_2010_Final_v2.pdf [Feb. 12, 2018]

[43] Adrian Butter, *"64GFC PCS/FEC Architecture Proposal for FC-FS-5"* [On-line] Available: https://standards.incits.org/apps/group_public/download.php/82006/T11-2016-314v5.pdf [Feb. 12, 2018]

[44] Xilinx, *"LogiCORE™ IP Fibre Channel User Guide v3.5"* [On-line] Available: https://www.xilinx.com/support/documentation/ip_documentation/fibre_channel_ug136.pdf [Feb. 12, 2018]

[45] Xilinx, *"32G Fibre Channel (32GFC) RS-FEC v1.0"* [On-line] Available: https://www.xilinx.com/support/documentation/ip_documentation/fc32_rs_fec/v1_0/pb048-fibre-channel-32gfc-rs-fec.pdf [Feb. 12, 2018]

[46] 10 Gigabit Ethernet Alliance (10gea), *"XAUI interface"* [On-line] Available: https://www.10gea.org/whitepapers/xaui-interface/ [Feb. 12, 2018]

[47] Agilent Technologies, *"10 Gigabit Ethernet and the XAUI interface"* [On-line] Available: http://literature.cdn.keysight.com/litweb/pdf/5988-5509EN.pdf [Feb. 12, 2018]

[48] Altera/IntelFPGA, *"HiGig / HiGig+ / HiGig 2"* [On-line] Available: https://www.altera.com/solutions/technology/transceiver/protocols/pro-higig.html [Feb. 16, 2018]

[49] Interlaken Alliance, *"Interlaken Alliance"* [On-line] Available: http://interlakenalliance.com/ [Mar. 29, 2018]

[50] Interlaken Alliance, *"Interlaken Interoperability Recommendations"* [On-line] Available: http://www.interlakenalliance.com/interlaken-interoperability-recommendations-v1.10.pdf [Apr. 03, 2018]

[51] Xilinx. *"VC707 Evaluation Board for the Virtex-7 FPGA - User Guide"* [On-line] Available: https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf [Feb. 19, 2018]

[52] Xilinx. *"7 Series FPGAs GTX/GTH Transceivers - User Guide"* [On-line] Available: https://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf [Feb. 19, 2018]

[53] Xilinx. *"VC707 EVALUATION PLATFORM HW-V7-VC707 (XC7VX485T-FF1761)"* [On-Line] Available: https://www.xilinx.com/support/documentation/boards_and_kits/vc707_Schematic_xtp135_rev1_0.pdf [Feb. 19, 2018]

[54] Erik van der Bij, *"S-Link Overview"* [On-line] Available: http://hsi.web.cern.ch/HSI/s-link/introduc/overview.htm [Feb. 08, 2018]

[55] S. Baron, J.P. Cachemiche, F. Marin, P. Moreira, C. Soos, *"Implementing the GBT data transmission protocol in FPGAs"* [On-line] Available: https://cds.cern.ch/record/1236361/files/p631.pdf [July. 09, 2018]

[56] A. Marchioro, P. Moreira, *"Low Power GBT"* [On-line] Available: https://indico.cern.ch/event/153564/contributions/1397870/attachments/161703/228199/Marchioro_LP_GBT__FNAL_Nov_2011.pptx [July. 09, 2018]

[57] J. Anderson, K. Bauer, A. Borga, H. Boterenbrood, H. Chen, K. Chen,G. Drake, M.Dönszelmann, D. Francis, D. Guest, B. Gorini, M. Joos, F. Lanni, G. Lehmann Miotto, L. Levinson, J. Narevicius, W. Panduro Vazquez, A. Roich, S. Ryu, F. Schreuder, J. Schumacher, W. Vandelli, J. Vermeulen, D. Whiteson, W. Wu and J. Zhang, *"FELIX: a PCIe based high-throughput approach for interfacing front-end and trigger electronics in the ATLAS Upgrade framework"* [On-line] Available: https://cds.cern.ch/record/2229597/files/ATL-DAQ-PROC-2016-022.pdf [July. 09, 2018]

A Traditional CERN protocols

CERN of course already had some protocols developed to transfer data between devices. It is very interesting and important for the purpose of this assignment to also look at the existing protocols CERN already has implemented. This way it would also be possible to inspect what their specific pros and cons are and why they are or were adopted. The protocols analyzed are S-Link, Full mode, GBT and the low power variant of GBT.

A.1 S-Link

S-Link is a protocol developed in 1995 at CERN and stands for Simple Link Interface. It was developed to connect any layer of front-end electronics to the next layer of read-out electronics. There are multiple implementations of the S-Link available which are also sold as cards. HOLA (High-speed Optical Link for Atlas) is the most recent variant which offers data rates up to 2.0 Gbps. There is a some information on the S-Link64 which could achieve a throughput of 6,4Gbps. [54] In addition to the data transfer, S-Link also offers error detection, a return channel for flow control and for return line signal and even offers a function for self-testing.

A.2 GBT

The GBT (GigaBit Transceiver) protocol developed by CERN provides a radiation-hard optical link which can transmit data at speeds of 4,8 Gbps. Generating the GBT frames will be done with an radiation tolerant ASIC. A single GBT frame consists of 120 bits from which 116 are data/payload. The remaining 4 bits are used as a header which is applied for aligning and to distinguish data and control words. FEC is also included in the GBT protocol. [55]

The so called low power variant improves on the energy consumption by reducing this to 25% in comparison with the original GBT protocol. Data transfer speed stay the same and off course the protocol itself structurally stays the same. However several features are removed to reduce the energy consumption. [56]

A.3 Full mode

Full mode is a lightweight protocol with a throughput of 9,6 Gbps. Since it uses 8b/10b encoding the maximum user payload is limited at 7,68 Gbps. It is currently applied in the FELIX (Front-End LInk eXchange) system which is part of the Atlas experiment. [57]

B Specifications of discussed protocols

Several protocols have been discussed and described in this document. The purpose of this appendix is to easily provide links to documents containing the specifications on these discussed protocols. The protocol name will be mentioned followed by the version and date the document has been updated for the last time. In case the protocol is not located in this list, no clear documentation has been found.

Xilinx Aurora 64b/66b v11.2 October 4, 2017 :
https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b/v11_2/pg074-aurora-64b66b.pdf

Xilinx Aurora 8b/10b v11.1 April 4, 2018 :
https://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b/v11_1/pg046-aurora-8b10b.pdf

Intel FPGA SerialLite III December 29, 2017 :
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_slite3_streaming.pdf

Microsemi LiteFast March, 2018 :
https://www.microsemi.com/document-portal/doc_view/135971-ug0701-litefast-ip-user-guide

Interlaken Protocol Definition Revision 1.2 October 7, 2008 :
http://www.interlakenalliance.com/Interlaken_Protocol_Definition_v1.2.pdf

Serial ATA Revision 3.0 June 2, 2009 :
http://www.lttconn.com/res/lttconn/pdres/201005/20100521170123066.pdf

CPRI Specification V7.0 October 9, 2015 :
http://www.cpri.info/downloads/CPRI_v_7_0_2015-10-09.pdf

HyperTransport™ I/O Link Specification Revision 3.10c June 5, 2010 :
https://docs.wixstatic.com/ugd/071cb6_53b2dc066f2d4408b5c9368dc447e2f5.pdf

C Core1990

Core1990 is a point-to-point communication protocol using the royalty-free Interlaken protocol as it's foundation. It is designed by engineers and students of the Electronics Department of Nikhef (Amsterdam, The Netherlands) with large experiments at CERN (e.g. ATLAS) in mind.
The development of Core1990 was intended to publish an open source protocol providing high throughput with a small percentage of overhead. Certain features like flow control and error detection are included.

This document will describe setting up the core including configurations of IP-cores included in the design. During development and writing this document a Xilinx VC707 evaluation board is used and sometimes certain IO's will be referred to. This is of course board dependent and these are mentioned as examples to clear things up.

Figure 43: Core1990 logo

C.1 Features

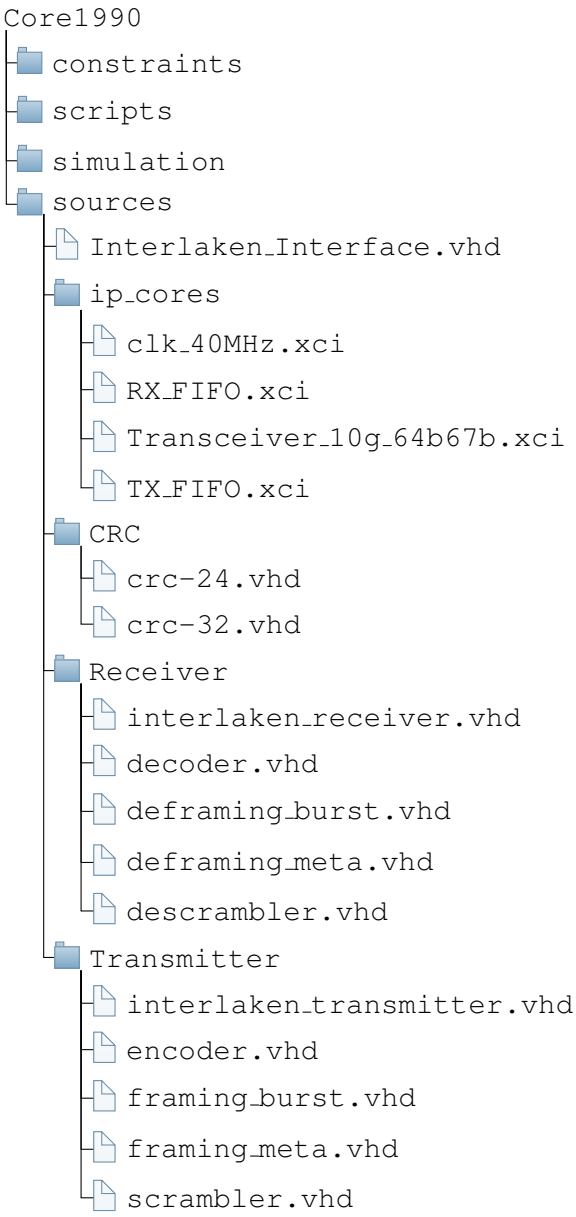
Core1990 is packed with a lot of features providing among others data integrity and detection of errors while transmitting. These features are designed to be compliant with those featured in the Interlaken protocol definition.

- Lane rate transceiver dependent
- Support framing consistent with the Interlaken Protocol Definition
- Generates CRC-24 and CRC-32 for error checking
- 58-bit independent synchronous scrambler
- 64b/67b encoding
- About 90% bandwidth efficiency possible (depends on user configuration)
- Self-synchronizing links
- Flow control

C.2 Obtaining and building Core 1990

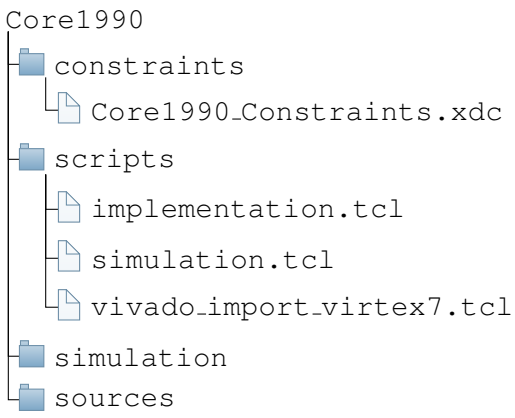
Implementing the core in a design can be done easily by using the files provided with core1990. The complete process of obtaining and building Core1990 will be described including which files are required and should be included. The project has been designed in Xilinx Vivado 16.4 but should be compatible with other versions. The correct configuration of the ip cores will also be mentioned to ensure behavior as expected.

The complete project can be downloaded through the download link on the OpenCores site itself or through SVN. For OpenCores the link is https://opencores.org/project/core1990_interlaken and the SVN links is https://opencores.org/ocsvn/core1990_interlaken/core1990_interlaken/trunk in case this is preferred.



The directory tree depicts the file structure in the sources folder. This should contain several files to configure the ip cores, a folder containing two crc error detection modules, a receiver and transmitter folder containing the module files. A main file is included in the folder that is meant for the top level connecting all modules correctly to each other.

Besides these files there are also three different folders. One contains the constraints which are responsible for connecting the physical pins to the signals in the package and providing clock information to the fitter of the design. Another folder contains several scripts to build the project by just running a single script. Another script will be able to generate testbenches on request of the user and a script generates the implementation of the design. The simulation folder contains a lot of testbenches used to simulate all the components included in the design.



For building the project Vivado has to be opened and the vivado_import_virtex7.tcl has to be executed. This can be done by changing the directory in the tcl console to the scripts folder and then giving the command 'source vivado_import_virtex7.tcl'. This will add the project folder to the directory tree and contains the just generated project.

When the project has been imported in vivado the structure should match the one depicted in Figure 44.

Figure 44: Structure of the project in Vivado

C.3 Transceiver IP Core

Configuring the transceiver is an essential step in setting up the core. This will describe the correct settings to use for the functional behavior of the protocol. In case the transceiver is configured in a wrong way, no data or corrupted data will arrive at the receiving side. This section will guide the user to set up the transceiver in an easy way without adjusting too many clocks, targeting the VC707 board.

The transceiver core can be configured by browsing through a separate window that will pop up. The fist tab named GT Selection should already have the GTX as GT Type selected and the shared logic should be included in the core, not the example design.

After this the second tab will provide more important options. The line rate should be set to 10 Gbps while the reference clock can be set at 125 MHz. This clock is available on the board at IO pins AH7 and AH8, REFCLK0_Q0. Using the QPLL GTX X1Y2 can be used. Figure 45 shows the correct configuration.

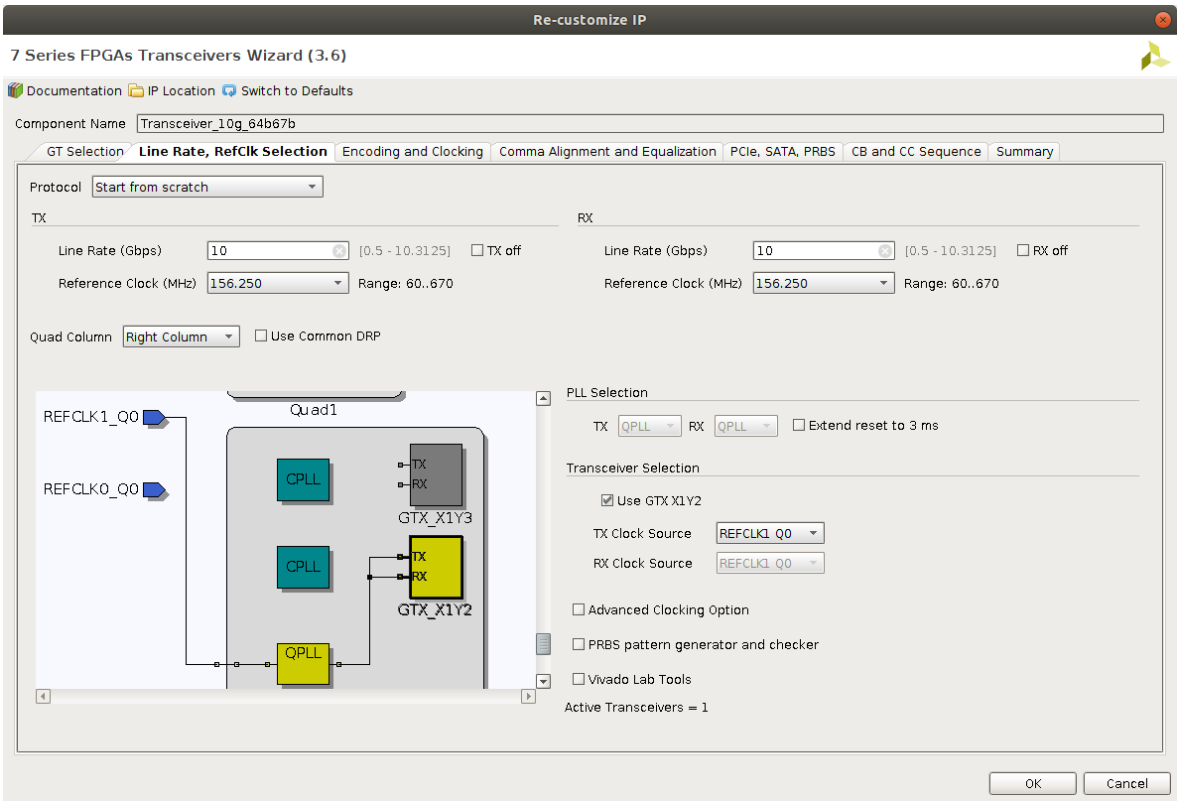


Figure 45: Transceiver lane rate and reference clock selection

The encoding and clocking tab shows other important settings. For both the TX and RX, the external data width should be set at 64-bits while the internal data width is 32 bits. Encoding has to be set at 64B/67B with Internal Sequence Counter and decoding has to be set at 64B/67B. The DRP/System Clock Frequency has to be set at 40 MHz. Figure 46 shows the right settings.

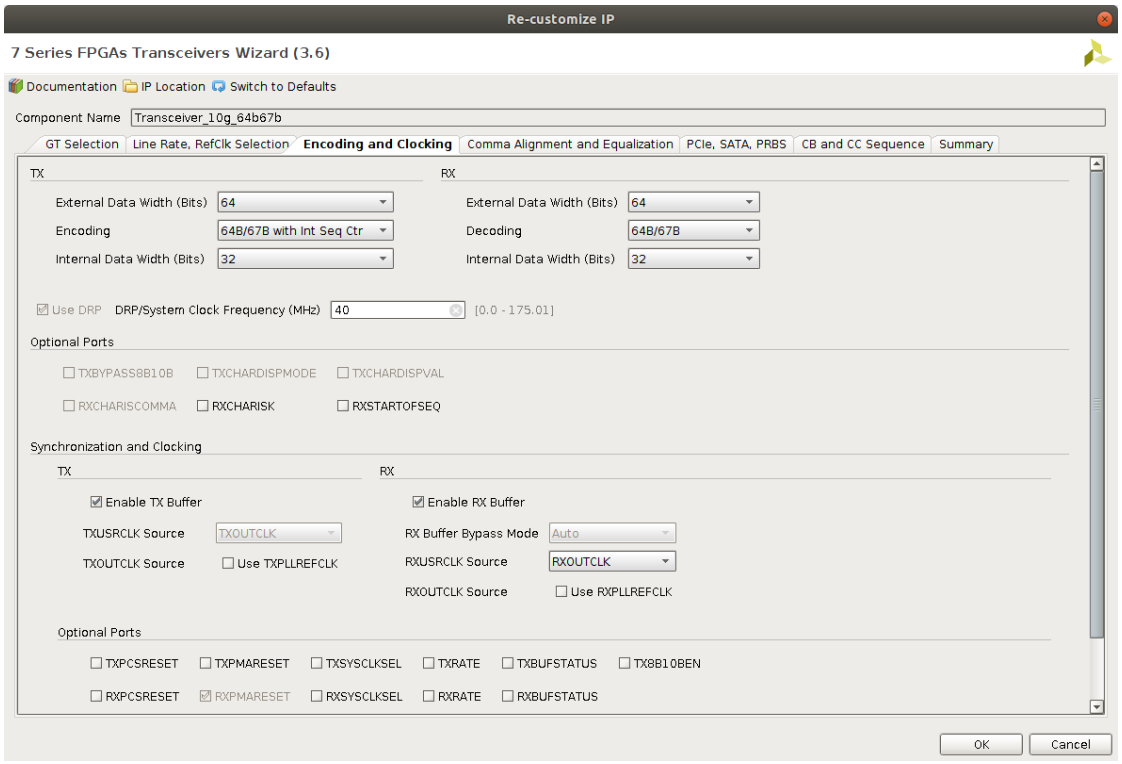


Figure 46: Transceiver encoding and system clock selection

The other tabs are not important and no settings should be changed in these tabs. Figure 47 shows a complete summary of the features included with the transceiver when this core will be generate. The user should have the same settings on screen.

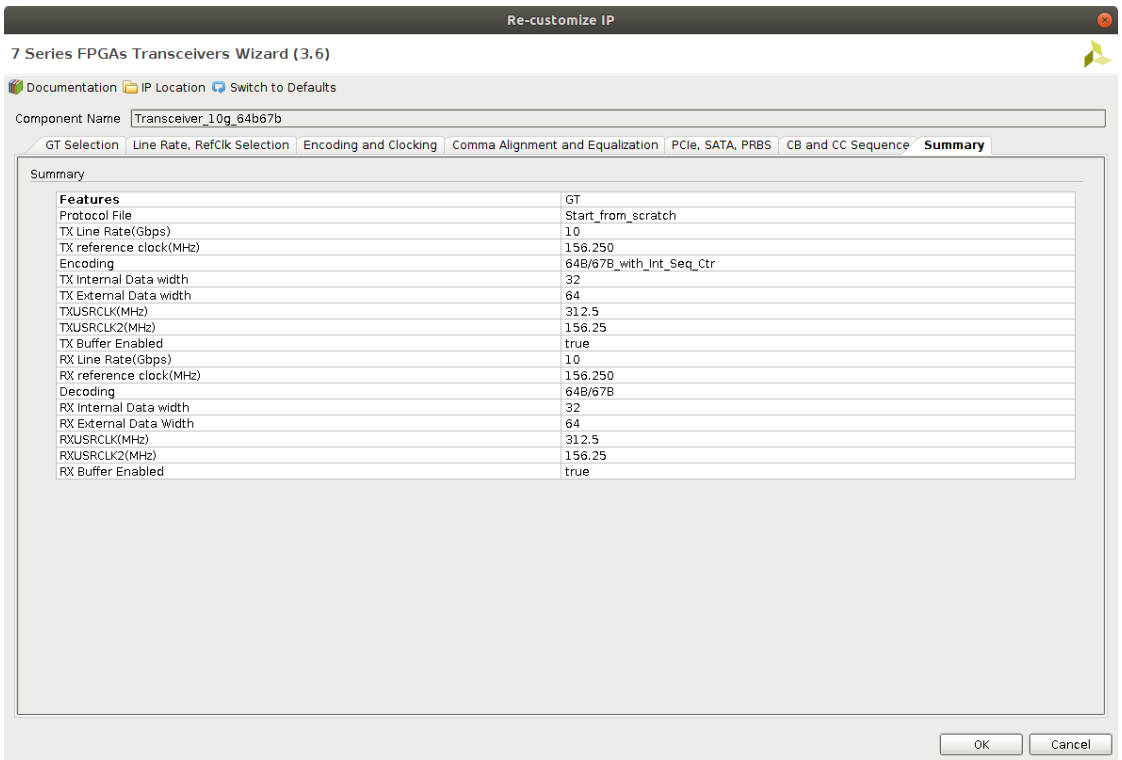


Figure 47: Transceiver summary of configuration

C.4 Clocking Wizard IP Core

The system clock will be generated using a clock input provided to the chip. In this case this is a 200 MHz input at IO pins E18 and E19. The frequency will be scaled down using the Xilinx Clocking Wizard which generates an IP core. Two clocks will be provided in this case. One will end up being the DRP clock of the transceiver and another clock will be the clock selected by the user. This clock can be connected to external user logic, for example a data generator, or the FIFO write and FIFO read on respectively the TX and RX side.

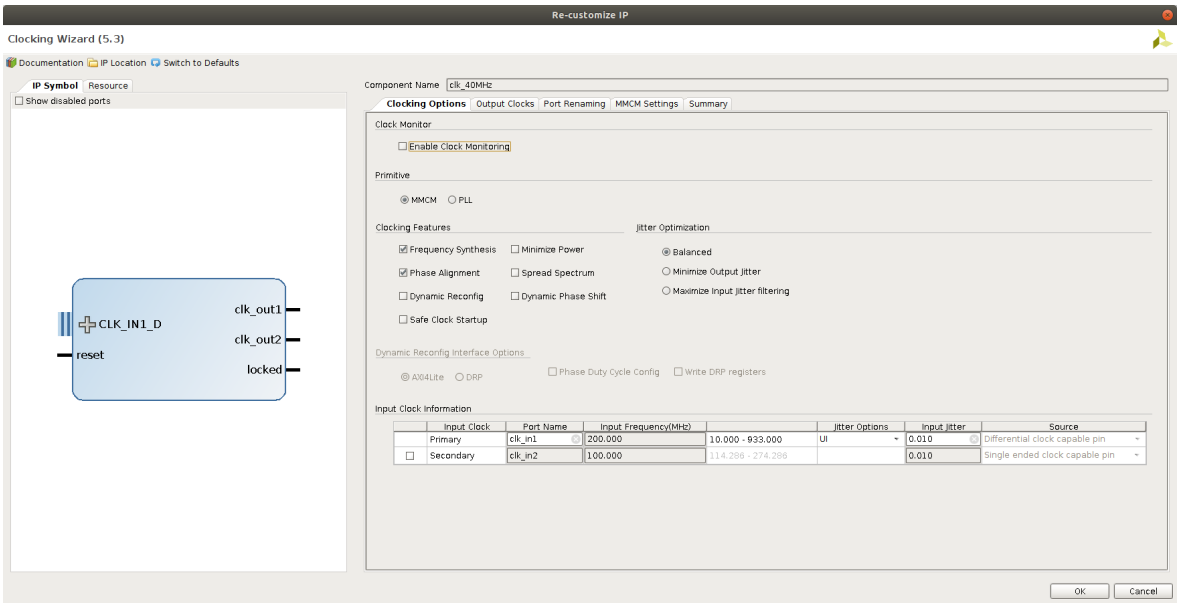


Figure 48: Clocking wizard input clock(s) and features

The first tab of the clocking wizard provides some basic information on the input clock which in this case will be a 200 MHz differential clock. It is important to select 'differential clock capable pin' under source here. This core makes use of a MMCM (Mixed-Mode Clock Manager). From clocking features only Frequency synthesis and phase alignment are selected. The jitter optimization is balanced and the input frequency of the primary clock is 200MHz. There is no secondary clock to use thus this is not selected and relevant. Figure 48 shows the configuration on the first tab.

The second tab will provide more configuration on the output clocks. Like depicted in Figure 49 there will be one 40 MHz clock and this has to stay at 40 MHz. Another clock is set at 150 MHz but this is the clock available for user logic like mentioned earlier. This can be configured as reference frequency to send data packets at. It is up to the user to determine whether this clock will be used and at which frequency it will run. However the maximum frequency of this clock also determines on the lane rate of the transceiver and at which speed the Interlaken interface itself runs. The example design in C.6 will also use this clock and describe more on it's usage. All clocks use a duty cycle of 50% and from the optional outputs only the reset and locked pins are required.

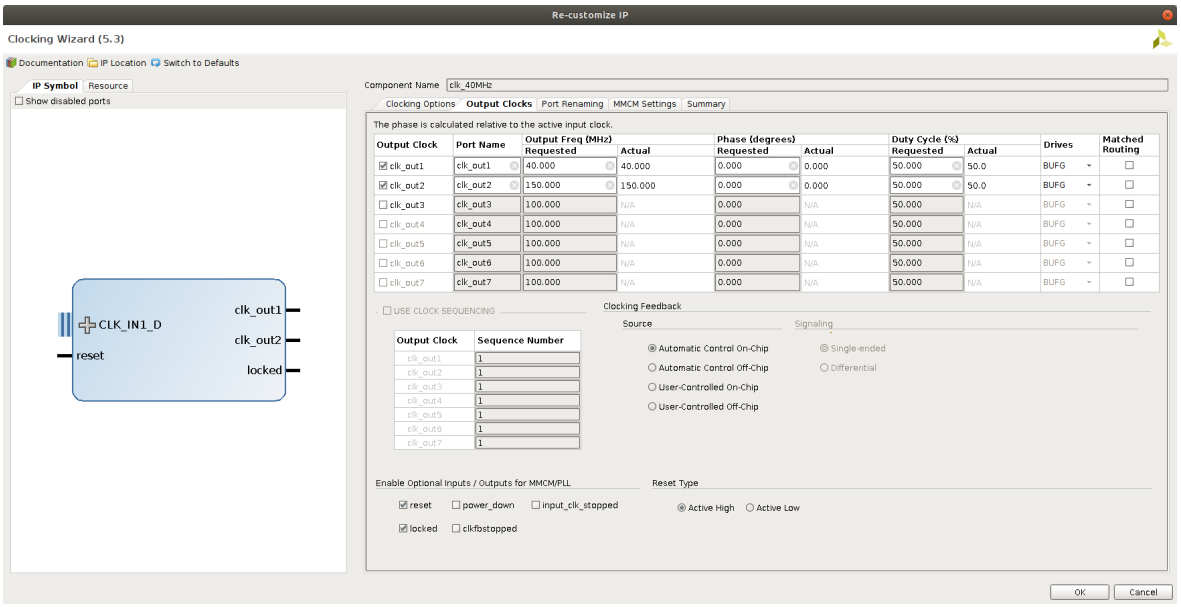


Figure 49: Clocking wizard output clocks

The last important tab is the summary tab. This should give an overview that contains the same information and values as depicted in Figure 50. However if the second output clock is set at another frequency this will of course give a different value is the overview.

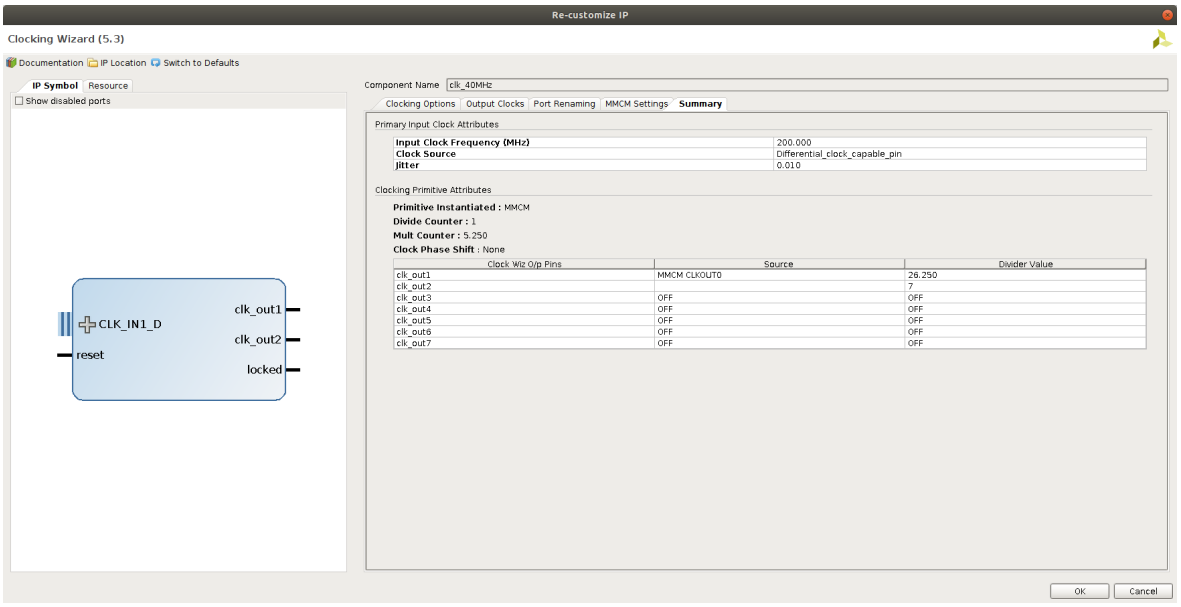


Figure 50: Clocking wizard summary

C.5 FIFO IP Cores

The Core1990 design contains two FIFO's which are used for the to be transmitted and received data. These two FIFO's have a different purpose but their configurations are nearly identical. This is why they are both described in one section.

The TX side FIFO stores all user data transmitted to the interface. Configuring the FIFO starts at the basic tab which allows the user to define the interface type and implementation of the FIFO. Figure 51 depicts the first tab. In this case the native interface is used in combination with an independent clocks distributed RAM and two synchronization stages. This way the FIFO can also be used for data crossing clock domains.

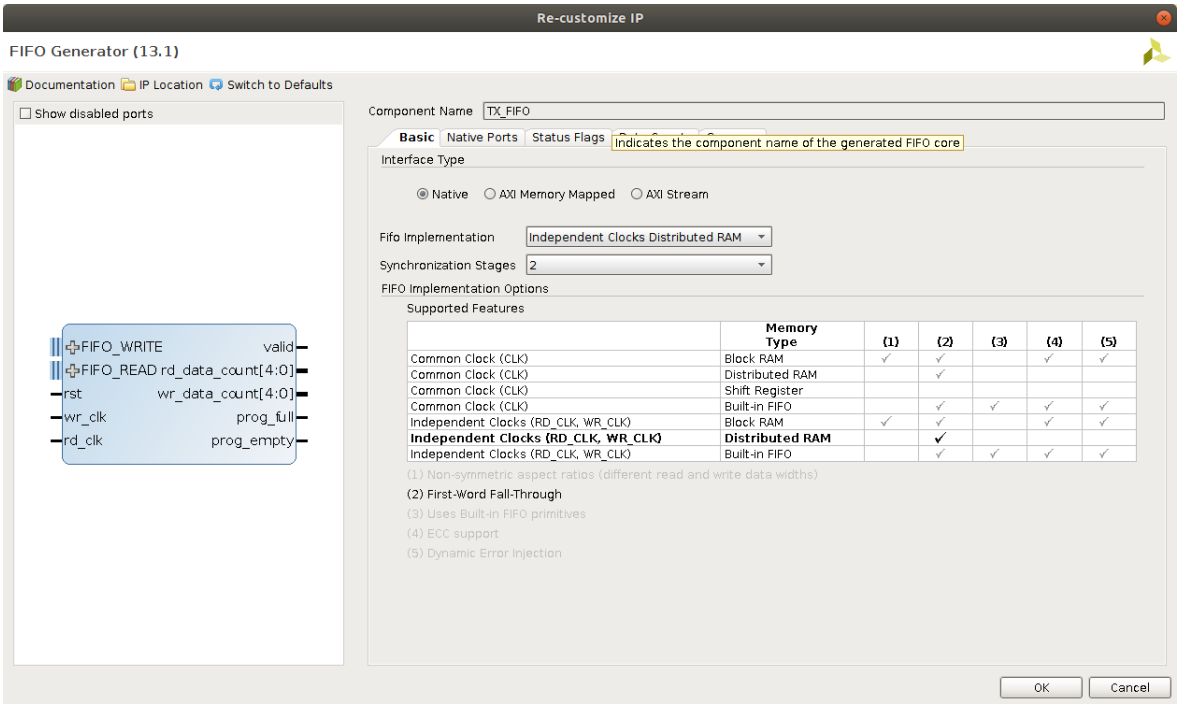


Figure 51: FIFO generator basic settings

The second configuration tab, as seen in Figure 52, offers more settings on the data width, read mode and initialization. This is meant as a standard FIFO so this option is selected, the only difference with the other mode is interface has to wait one clock cycle after read enable contains a logic high input.

The FIFO has a data width of 69 bits. This value has been chosen because all essential information now fits in a single packet entering the FIFO. Such packet fits the user data, Start Of Packet, End Of Packet and End Of Packet validbytes.

The write depth of the FIFO can be selected according to user preference. In the example 32 packets have been chosen to be sure enough space is available but this could possibly be lowered. Besides this the FIFO contains a reset pin and the output resets to zero. The reset is also synchronized in the core.

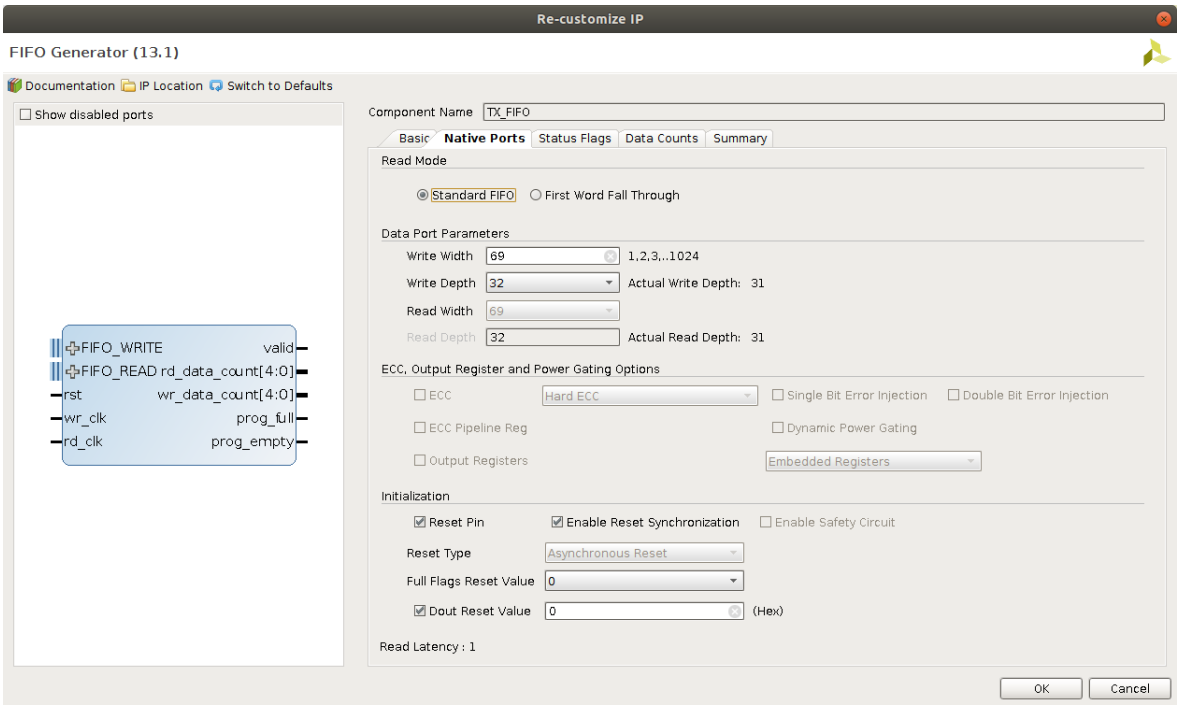


Figure 52: FIFO generator data port and initialization configuration

In the third tab, depicted in Figure 53, several status flags can be enabled which are very useful. The read port for example uses a valid flag in case the data is really valid. This prevents the interface to transmit duplicated while there is no new data. When valid is a logic low then data will not be read. There are also a few valid programmable flags. In this case Multiple Programmable Full Threshold Constants are used which are useful in combination with flow control.

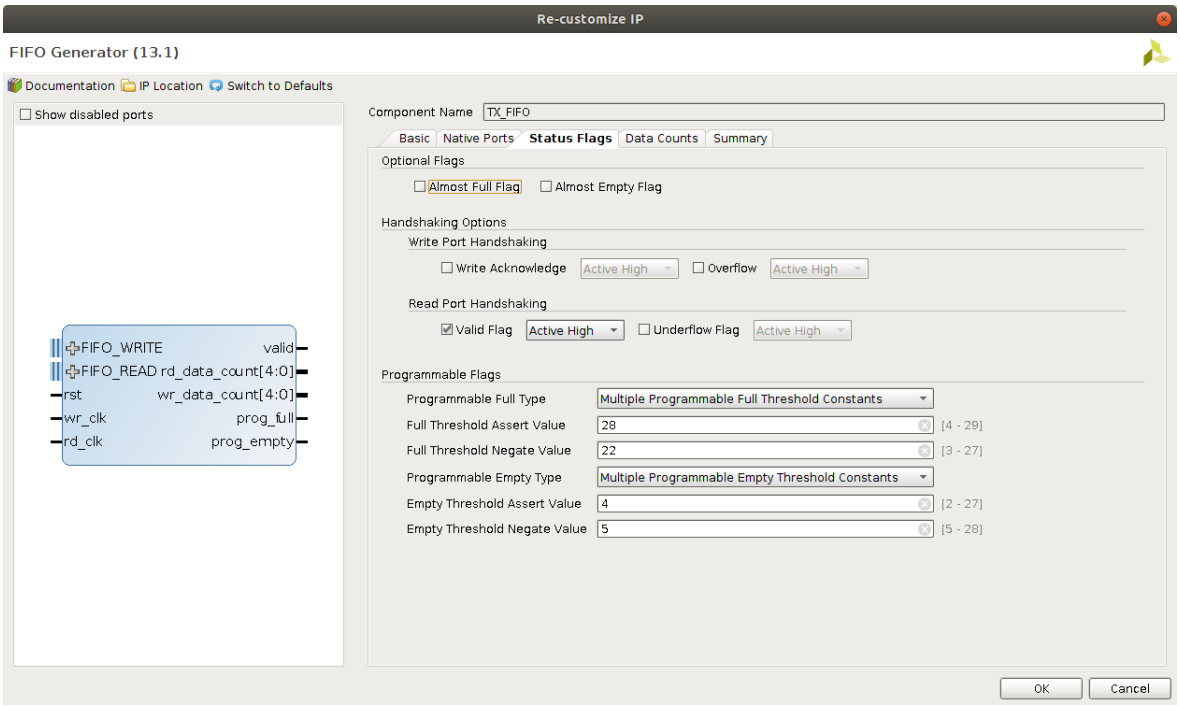


Figure 53: FIFO generator status flags

The fourth tab contains setting on reading the data count in the FIFO. As usual a summary like the one depicted in Figure 54 will always be visible in the last tab. Here you can check whether the necessary settings are configured correctly.

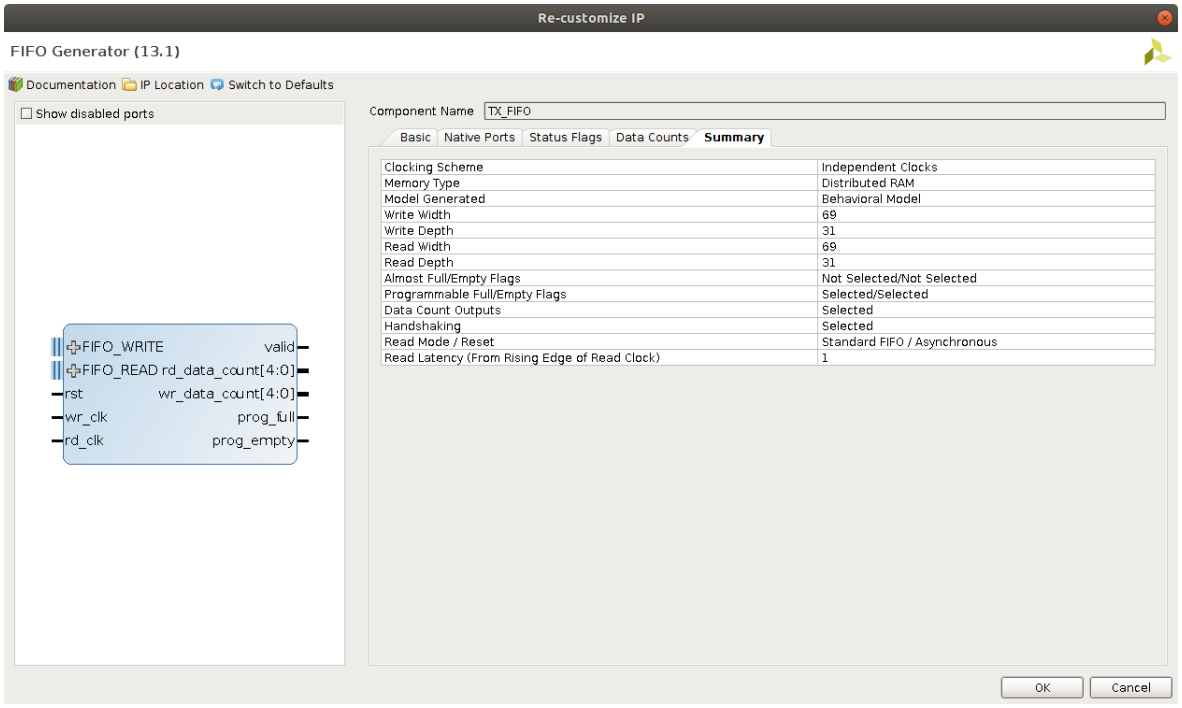
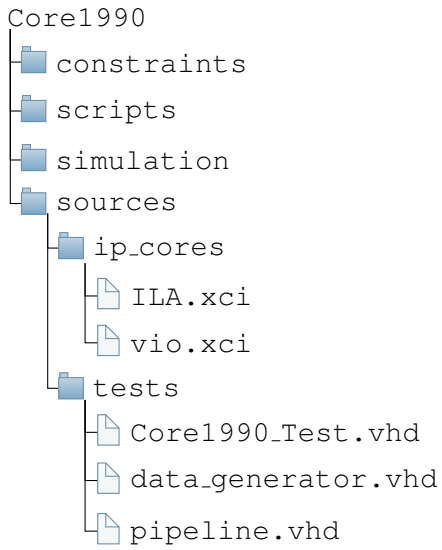


Figure 54: FIFO generator summary

C.6 Example design

During the design stage of Core1990 multiple tests have been run to inspect how the link behaves, whether the correct data arrives or how robust the link itself is. These tests have been performed by implementing a data generator connected to the Core1990 inputs. A VIO (Virtual Input/Output) is used to control the length of bursts and an ILA (Integrated Logic Analyzer) is used to sample the input and output data (ChipScope). The input data will be pipelined for alignment between the two. This will make it easier to detect errors and ensure data integrity.

The example design can be generated by running the 'source ./vivado_import_virtex7_exampledesign.tcl' command from the tcl console.



The additional IP cores and VHDL components are all included in the Core1990_Test.vhd file. This top level only requires two clocks to run, two differential signals for RX and TX are defined and two output signals indicating locked status and valid compared data are used and connected to an external led. However during testing another clock appeared to be better suited for this purpose (156,25 MHz from the Si570). This requires two additional in- and outputs which are added in the Core1990 file.

The example design realized in Vivado for the VC707 uses 6861 LUT (LookUp Tables), 15599 FF (FlipFlops) and 89 BRAM (Block RAM). Power consumption of the complete design is about 1,3 W.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM
synth_1 (active)	**constrs_1**	**synth_design Complete!**								**5001**	**11651**	**0**
impl_1	constrs_1	route_design Complete!	0.376	0.000	0.044	0.000	0.000	1.296	0	6861	15599	89
Out-of-Context Module Runs												
RX_FIFO_synth_1	RX_FIFO	synth_design Complete!								165	194	0
vio_0_synth_1	vio_0	synth_design Complete!								56	136	0
Transceiver_10g_64b67b_synth_1	Transceiver_10g_64b67b	synth_design Complete!								249	268	0
clk_40MHz_synth_1	clk_40MHz	synth_design Complete!								0	0	0
TX_FIFO_synth_1	TX_FIFO	synth_design Complete!								105	176	0
ILA_Data_synth_1	ILA_Data	synth_design Complete!								1885	2926	89

Figure 55: Resource usage by the example design

When the example design is opened in Vivado the structure of the project should look similar to what is depicted in Figure 56.

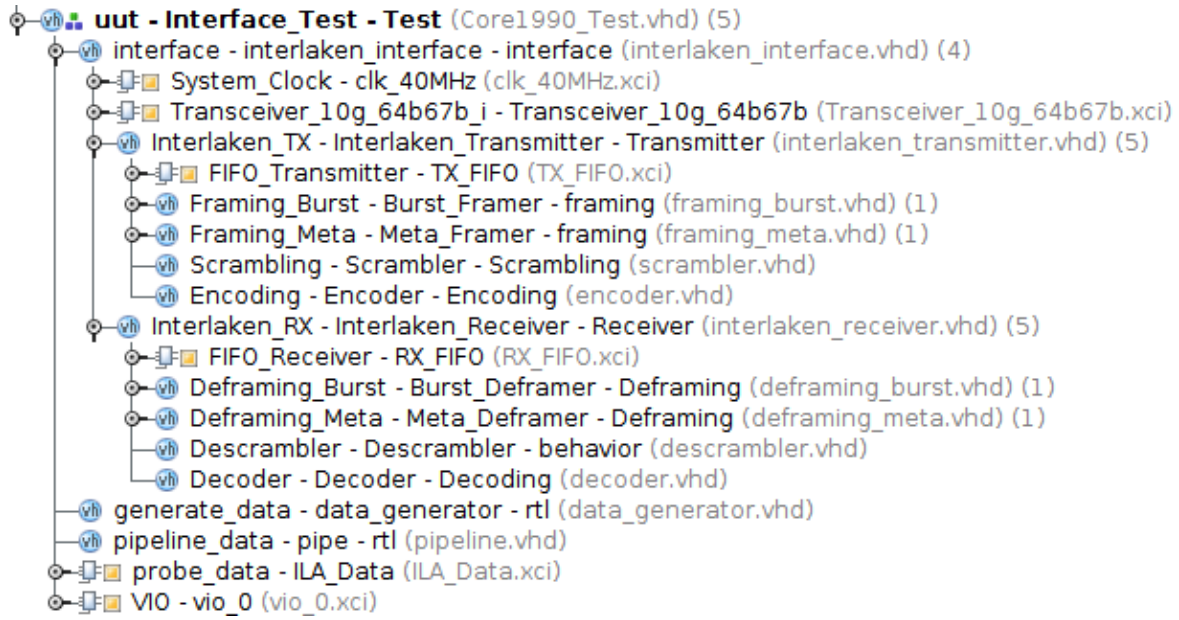


Figure 56: Structure of the example design in Vivado

C.7 Simulating the core

If the link doesn't behave as expected or is malfunctioning, the situation can be analyzed in simulation. This simplifies the process of locating errors. Simulation of the core1990 protocol can easily be configured by running the simulation script. This can be done by browsing to the scripts folder again and running the tcl command 'source simulation.tcl'. After this a short explanation of the command this script accepts should appear. For example if the user would like to simulate the decoder, this can be done by giving the command 'simulate decoder' in the tcl console. Simulation of the interface itself van be done by 'simulate interface' and to simulate the example design it is 'simulate core1990'.

When running the simulation.tcl script all testbenches will be added to the project. Besides running a specific command to simulate a part it is of course also possible to just start a simulation by selecting the top level in the Vivado GUI in the simulation sources. All added testbench files and the waveform configuration file can be seen in Figure 57.

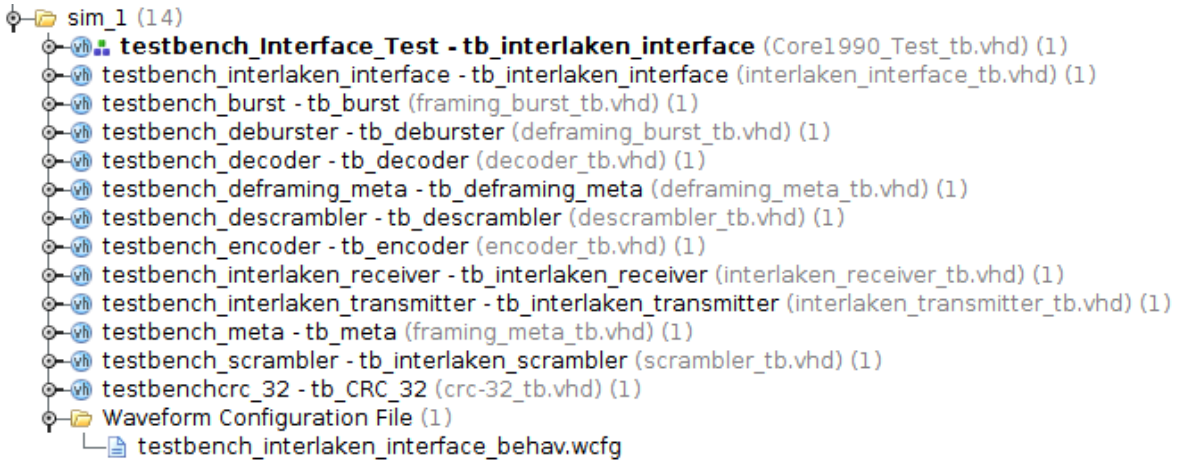


Figure 57: Structure of the simulation files in Vivado