MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Technologies industrielles (TIN)

PROTOCOLE SÉRIE POINT À POINT GÉNÉRIQUE ET MODULABLE

Fait par

Cédric Vulliez

Sous la direction de
Prof. Yann, Thoma
Dans l'institut REDS de HEIG-VD

Expert externe, Ralph Hoffman

Genève, HES-SO//Master, 2016/2017

Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de Jonathan Emery.

Prof. Yann Thoma, conseiller de travail de Master
Ralph Hoffman, Expert principal

Lausanne, le …./….. / 2017

Prof. ……..
Conseiller

Prof. …….
Responsable de la filière ….

1. Table of Contents

Acknowledgements

I would first like to thank my thesis advisor Yann Thoma of the Reds institute in HEIG-VD for the total confidence he had in my ability to manage this thesis, as well as my CERN supervisor Jonathan Emery to have proposed this project and followed me through it.

I would also like to thank the experts who were involved in the validation survey for this research project. Without their passionate participation and input, the validation survey could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents, for providing me with unfailing support and continuous encouragement throughout my years of study and to my girlfriend, through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Cédric Vulliez

Nomenclature

ASIC	Application-specific integrated circuit
BI	Beam Instrument Group
BWS	Beam Wire Scanner
CERN	European Organization for Nuclear Research
CRC	Cyclic redundancy check
FIFO	First In First Out
FPGA	Field-programmable gate array
GBTx	CERN Radiation tolerant chip
HES-SO	Haute Ecole spécialisée de Suisse occidentale
R&D	Research and Development
RX	Receiving side
SoC	System on Chip
SPS	The Super Proton Synchrotron
TB	Test bench
TX	Transmitting side
VME	VERSA-Module Euro

Abstract

CERN, the European Organization for Nuclear Research, is a European research organization that operates the largest particle physics laboratory in the world. A large amount of particle accelerators and detectors are developed and used for scientific research purposes. With those, a large amount of information is created and needs to be stored before it can be analyzed.

As a consequence, the communication reliability, integrity, latency and transfer rates between these detectors and the control room are some of the key elements for the success of those experiments. While it is true that existing communication standards, or protocols, are focused on those key elements, it is usually a balanced compromise to stay polyvalent, where the emphasis on a specific aspect reduces the efficiency of the others.

To improve the performance and capabilities of one of the operational accelerator instrument called the beam wire scanner (BWS), the CERN instrumentation Group (BE-BI) has created a complete new electro-mechanical system that will in time replace the current one. This new scanner also has a complete new control and acquisition concept that will require an update of the current communication architecture, mainly because the mechanical and acquisition parts are now split into two distinct elements.

This thesis analyzes the new demands and challenges for this new communication architecture and implement a point-to-point communication protocol on an optical link, linking the two Field-programmable gate arrays (FPGA) present on both elements of the sub-system (at the speed of 4.8 Gb/s). This protocol needs to satisfy all the new technical aspects, be transparent for the different types of information transiting through it as well as being generic and modular, to facilitate its implementation into other instruments or detectors present at CERN.

The protocol designed and developed during this thesis is successful in managing the integrity, latency and transfer rates required, by implementing retransmission mechanisms upon bad data reception and a weighted bandwidth arbiter to deal with latency and transfer rates issues. Furthermore, multiple type of information coming from different services can be transmitted or received in a complete independent and transparent way, thanks to the modularity and generic aspect of the code. A simulation test bench (TB) was also created to verify the reliability of the designed protocol, providing multiple corner case scenarios.

Finally, the first physical tests made between two standalone FPGA development boards were a success, making the protocol ready for further integration and tests with application specific FPGA designs.

2. Introduction

CERN, the European Organization for Nuclear Research, is a European research organization that operates the largest particle physics laboratory in the world. A large amount of particle accelerators and detectors are developed and used for scientific research purposes.

The Beam Instrument Group (BI) from CERN is developing a new generation of instrument for particle beam profile measurements inside the accelerator's vacuum pipe called a Beam Wire Scanner (BWS). Along with a complete new concept for the electro-mechanical system, the control and acquisition has also been updated with a new architecture using latest technical trend. Figure 1 presents the working principle of this instrument which creates local particles beam losses using a mechanical movement which are detected by downstream by a scintillation detector.

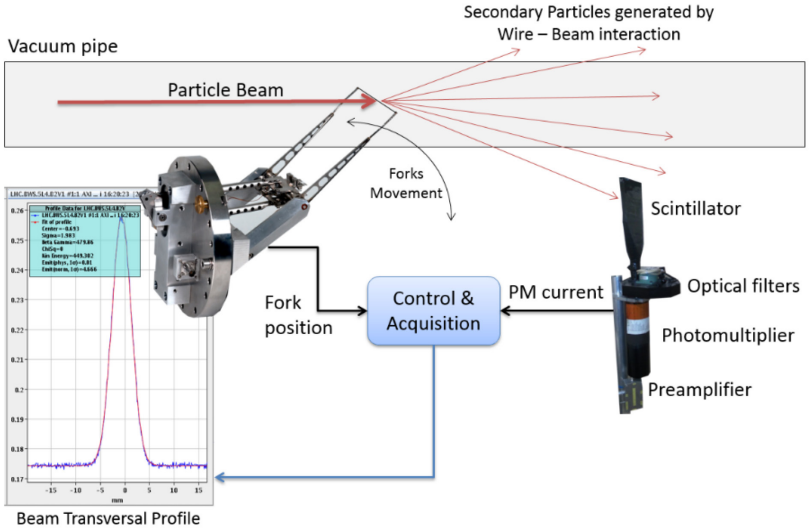


Figure 1: working principle of the Beam Wire Scanner

The system architecture is built with two sub-systems as presented on Figure 2, one for the motor actuation and monitoring called the Intelligent Drive (right side) and the other one for the signal acquisition and supervision (left side). These sub-systems have to be linked together using a digital serial connection (green round at the figure center).

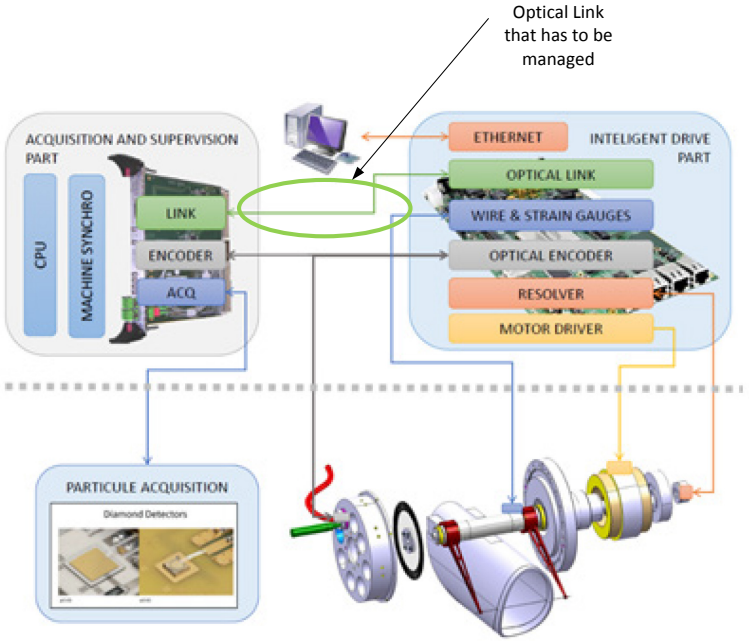


Figure 2: BWS system architecture with two sub-systems

Since the optical link is responsible for the configuration of the movement parameters, the movements triggering and to upload the acquired data recorded during the movement of the scanner, reliability, data integrity, data latency and transfer rates between the two sub-systems are all equally important and must be managed by the protocol in charge of transferring information through that link.

With this thesis, the new technical requirements for the BWS communication have been analyzed, and a protocol was developed to satisfy all its needs.

This thesis will first explain the physical challenges and expectations required by the new BWS system. Secondly, it will then present the implemented protocol created during the six months thesis and finally the physical validation results.

2.1. Aim of the thesis

Since the aim of the thesis is to research and develop a protocol capable of managing the optical link between the two sub-systems of the Beam Wire Scanner, it is important to understand what a protocol is.

In telecommunications, 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. These are the rules or standards that defines the syntax, semantics and synchronization of communication and possible error recovery methods [10].

As multiple entities need to transmit information through this optical link, some rules need to dictate how they will share this unique physical resource to satisfy their own individual needs, without compromising the needs of others. As far as the entities are concerned, their behavior should stay as if they were alone, only dealing with the syntax defined to transmit information.

Many standards and sets of rules already exist and will need to be examined and compare to CERN's criteria. The thesis needs to determine which rules will be required, which standards or ideas can be reused and which should be changed/improved to satisfy the technical requirements of this project.

The final aim is to provide to the user a transparent link, or virtual link between two FPGAs as shown on figure 3.

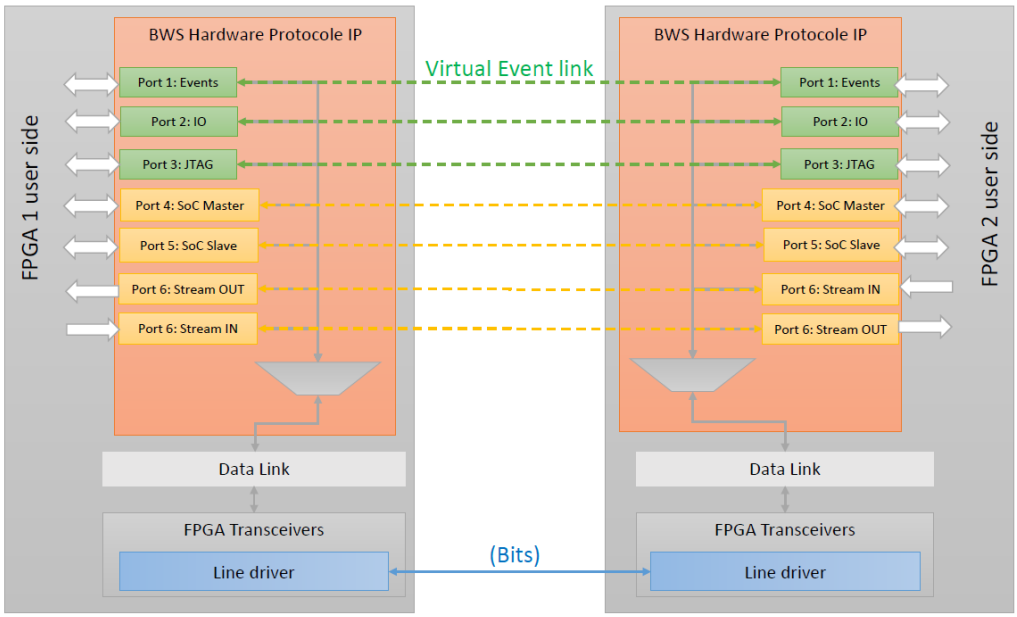


Figure 3 : Virtual link between two FPGAs

2.2. Beam Wire Scanner

The beam wire scanner belongs to the category of interceptive beam transversal profile measurement system. It uses electro-mechanical parts to move a thin wire of about 30µm through a beam. As shown on the figure 4, the losses generated by the interaction of the beam with the wire matter are escaping the beam pipe and are detected by a scintillator and photomultiplier. The generated current is proportional to the beam intensity at the position of the wire, which allows the reconstruction of the transversal beam intensity profile. This profile is processed to obtain the beam sigma and combined with other information to obtain the emittance of the beam [4].

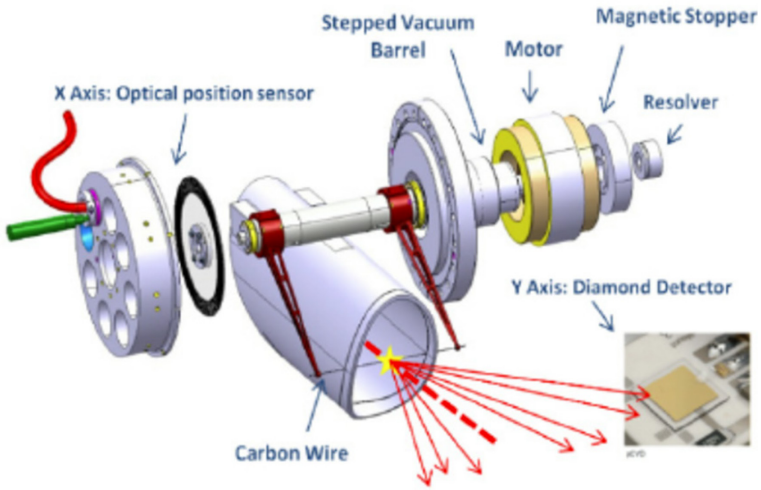


Figure 4 : Interaction of the beam with the wire

The Beam Instrumentation group of the European Organization for Nuclear Research (CERN) has been developing an instrument called the Beam Wire Scanner (BWS) for the past few years. This system is used to measure the size of proton beams in the Large Hadron Collider (LHC) and its injector chain. An electro-mechanical system moves a very thin wire of 30 µm through the particle beam and measures the induced radiation losses generated by this interaction. The actuator, based on a Permanent Magnet Synchronous Motor (PMSM), a solid rotor resolver and an in-house designed high precision optical encoder are located in underground installations and have to cope with large irradiation levels [4].

The wire speed needs to be higher than 20m/s and have a measurement precision of 5 µm.

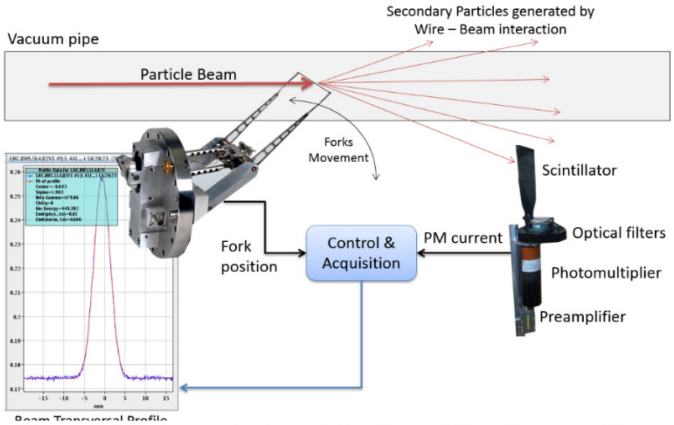


Figure 5: Working principle of the Beam Wire Scanner (2)

This project is part of the development of a new Beam Wire scanner system that can increase this precision and offer new functionalities.

Accelerator specialists are increasingly using the wire scanner to verify the “quality” of beams produced by the injector chain before they reach the Large Hadron Collider (LHC), discarding any poor quality beams. The wire-scanner is also used to optimize the transfer of beams between machines, allowing the injectors to supply the LHC with beams of the smallest possible size. The growing focus on this instrument and the resulting increase in its use during routine accelerator operation has initiated an upgrade of the existing designs. This new, common design should fulfil all the specific requirements of the individual accelerators, at the same time as increasing the availability and the reliability of the system [4].

2.3. BWS data acquisition and needs

In order to fully understand the challenges of this project and protocol requirements, it is important to understand how the Beam Wire Scanners are controlled, their needs, expectations, and the conditions and limitations of the data acquirement.

Firstly, it is important to understand that this system is a point to point communication system. Each Scanner mechanics is connected to its own intelligent drive. The intelligent drives (ID) are then connected separately to the acquisition and supervision cards, the VFCs, concentrated into one VME crate, using the developed protocol. The figure 6 below shows how the scanners are connected.

The implications of having one intelligent drive (ID) per scanner are as followed:

- avoids multiplexing.
- Constant monitoring/control.
- Allows parallel scans.
- But imply more control and acquisition systems

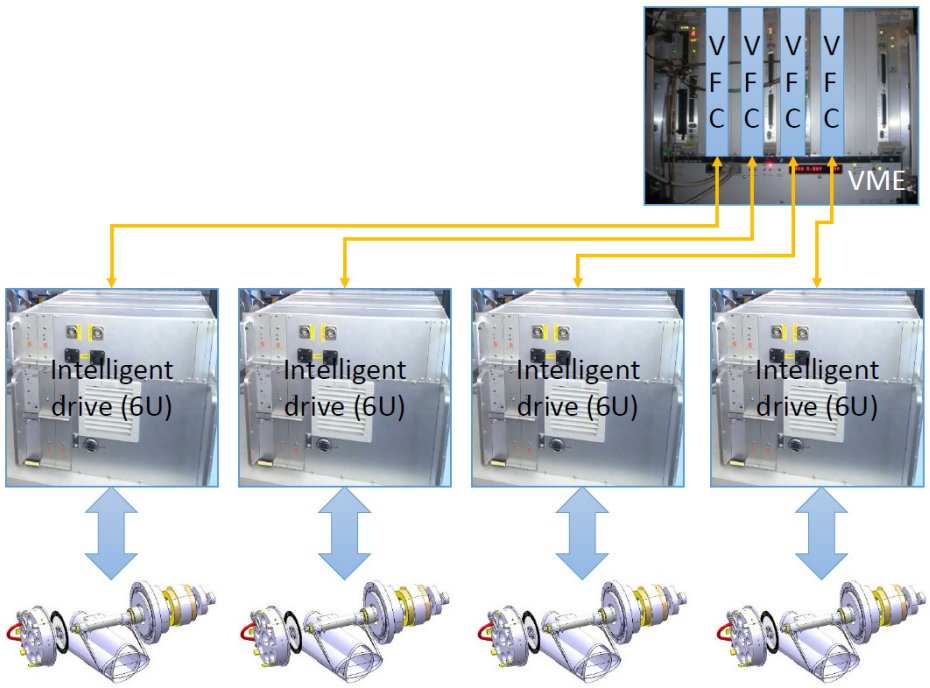


Figure 6: scanners connection

In the crate, the scanners can each be accessed through a single processor (CPU) with the VME bus.

The CPU can thus send commands to the scanners, triggers (Start Scan) and can also read the measurements and store them. Figure 7 shows the different connections to other CERN equipment.

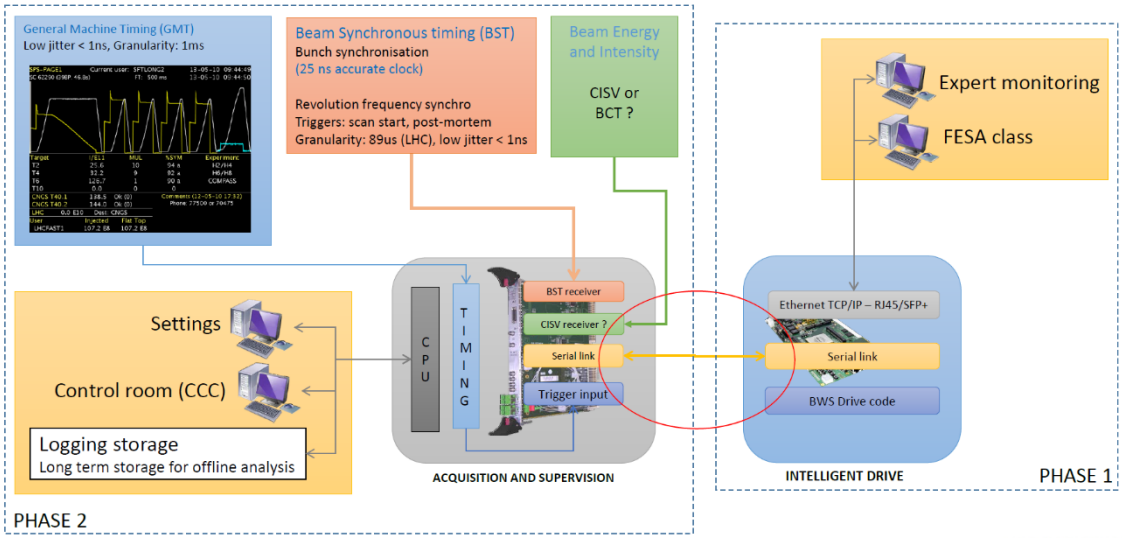


Figure 7 : Different connections to other CERN equipment

The Beam wire scanners are used in almost all CERN accelerators, from the PS Booster, the PS, the SPS and finally on the LHC. The new scanner version will in time replace all of the existing ones.

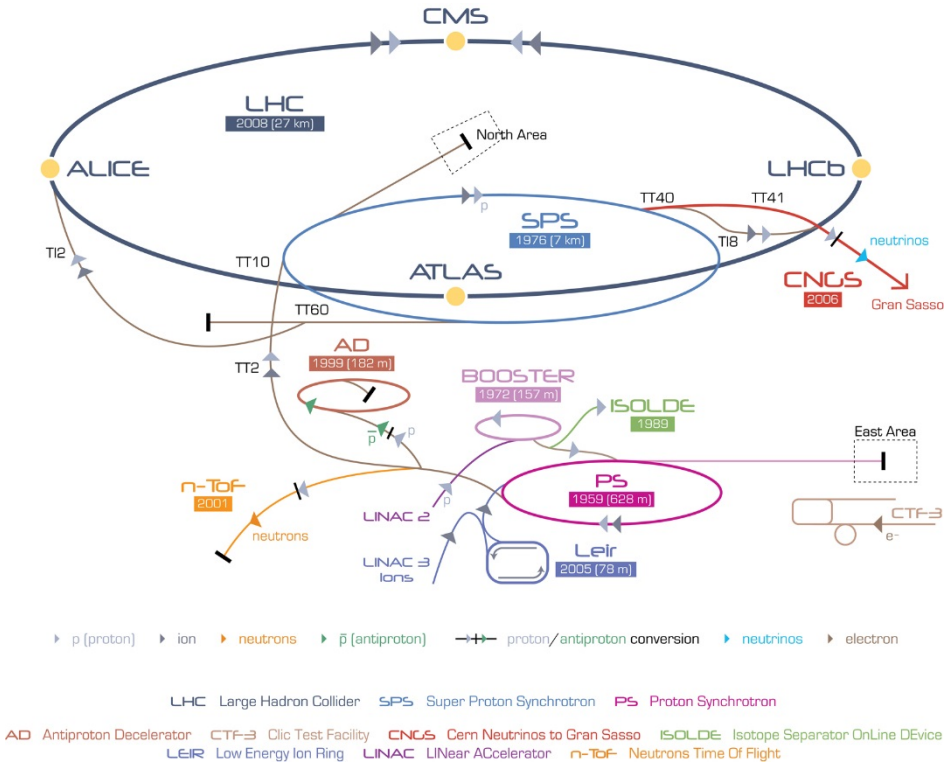


Figure 8 : CERN Accelerators system overview

Understanding the latency importance

Without going into great details, it is important to understand that particles are injected grouped together when sent in the accelerators. This group of particles is called a bunch. These bunches are injected at a certain frequency of around 40Mhz, or every 25 ns.

Figure 9 gives an example of typical display of a so called super-cycle (multiple cycle of the machine put together) of the SPS Accelerator (The Super Proton Synchrotron).

The Super Proton Synchrotron (SPS) is the second-largest machine in CERN's accelerator complex. Measuring nearly 7 kilometers in circumference, it takes particles from the Proton Synchrotron and accelerates them to provide beams for the Large Hadron Collider (LHC) and other experiments [11].

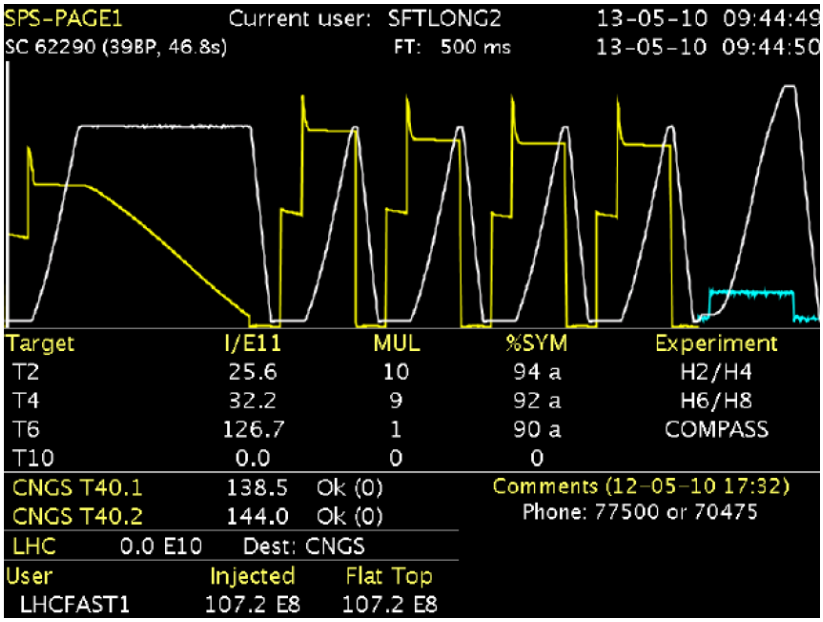


Figure 9: Typical display of a so called super-cycle

Overlaid with the alphanumeric in formation there are two curves. One represents the magnet cycle, the other the instantaneous intensity in the machine.

The white curve indicates the magnetic pulse in which we've defined four stages:

- Flat bottom: Particles are injected in the SPS with low energy;
- Ramp up: Particles are accelerated)
- Flat top: in this stage the particles are extracted)
- Ramp down: remaining particles at the end of the Flat Top are dumped and the SPS magnetic cycle returns to the Flat bottom)

The Yellow curves are indicating the intensity of particles in the SPS machine. [12]

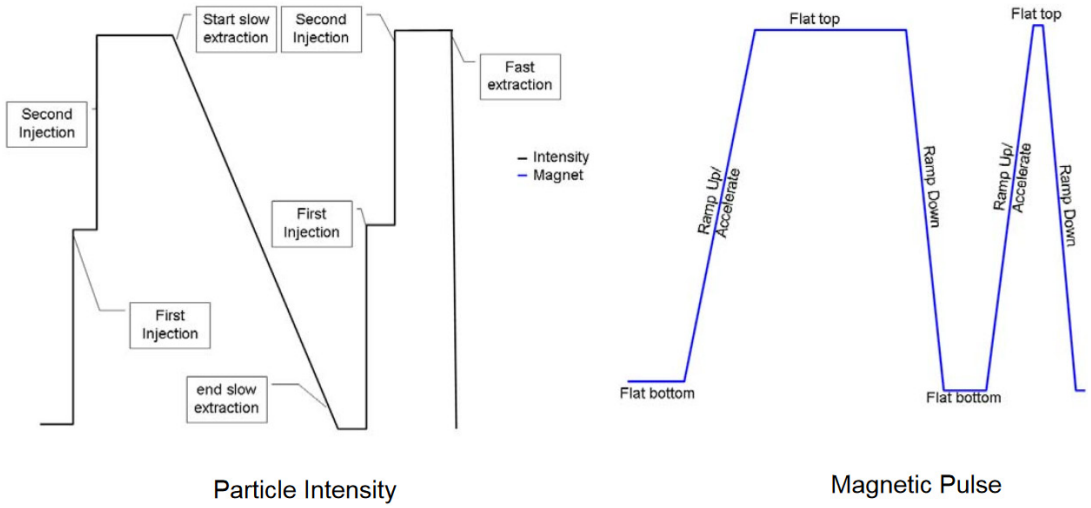


Figure 10 : Magnet cycle and instantaneous intensity

To verify the quality of the beam and its energy, the scanners can sample the beam at the different phases of the cycle, but before the particles are send elsewhere. Thus the trigger that will signal the start of a scan is a time sensitive signal. Two variables will affect this signal: a constant delay and a variable delay or jitter.

The constant delay will be the time the signal takes to travel through the different infrastructures, protocols and the optical link. Having a known constant delay poses relatively low problems as the trigger generated time can be calculated before hands, taking the delay into account.

The variable delay (jitter) however, will affect the scanner's starting time. This jitter is the sum of all variable delays that can happen over the course of the communication, such as jitter due to temperature fluctuations or protocol reactiveness.

In fine, the latency jitter plays a very important role for the scanner's trigger events and the technical requirement for this project demands that this jitter should be kept under 200ns.

Understanding the bandwidth problem

The accelerators have different injection and extraction cycle times based on particles energy level. For the PS Booster (PSB), the basic period is 1.2 second. During that time, the scanner will do a single acquisition. Figure 11 shows a time graph representing this acquisition in regards to the whole PSB period. After 1200ms, the cycle starts again, re-injecting particles to accelerate them.

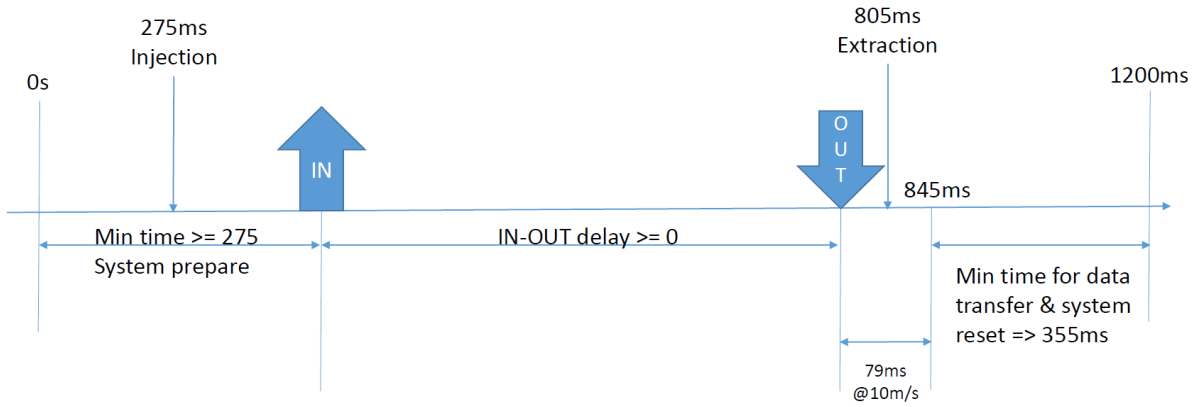


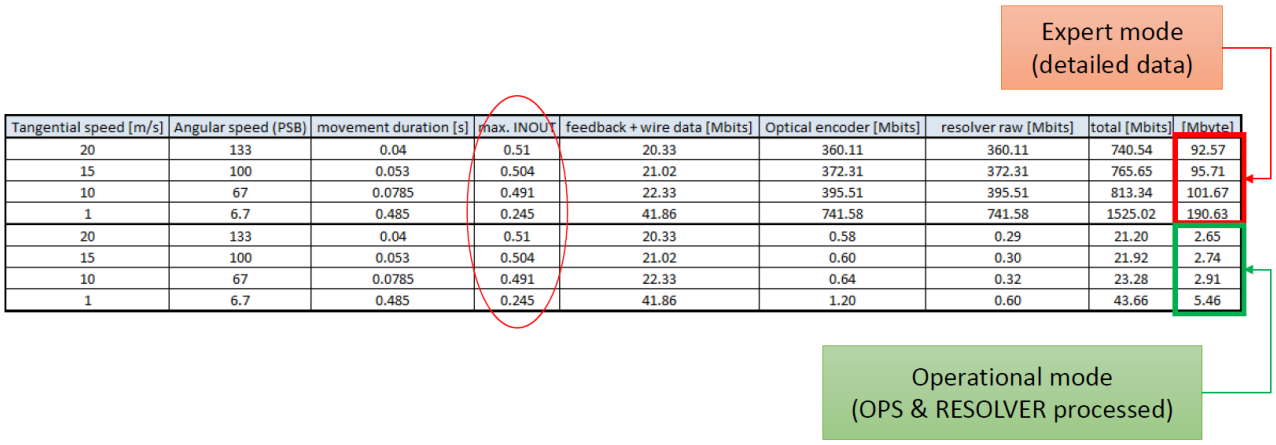
Figure 11: Time graph for the scanner acquisition

The scanner's system needs a minimum of 275ms to prepare itself. The scanner goes IN the beam, waits a defined time and then goes OUT of the beam. The wire then needs time to retract itself from the beam. After the scan is done, the system will need to transfer the recorded data over to the acquisition part before the 1200ms mark, to be ready for a next scan.

This puts a lot of constrains on the burst bandwidth demands, as data can only be sent during a certain period, and needs to be done accessing the memory data before the next cycle writes in it again.

The wire cannot continuously stay static in the beam as it would break due to the energy level passing through. It is constantly moving using a rotational fork. The speed of the fork defines how long the wire can stay in the beam, and thus how much data is generated.

Figure 12 shows a summary of the different speeds that which the BWS can operate, the max IN-OUT time of the scan, and the resulted amount of data produced.



Tangential speed [m/s]	Angular speed (PSB)	movement duration [s]	max. INOUT	feedback + wire data [Mbits]	Optical encoder [Mbits]	resolver raw [Mbits]	total [Mbits]	[Mbyte]
20	133	0.04	0.51	20.33	360.11	360.11	740.54	92.57
15	100	0.053	0.504	21.02	372.31	372.31	765.65	95.71
10	67	0.0785	0.491	22.33	395.51	395.51	813.34	101.67
1	6.7	0.485	0.245	41.86	741.58	741.58	1525.02	190.63
20	133	0.04	0.51	20.33	0.58	0.29	21.20	2.65
15	100	0.053	0.504	21.02	0.60	0.30	21.92	2.74
10	67	0.0785	0.491	22.33	0.64	0.32	23.28	2.91
1	6.7	0.485	0.245	41.86	1.20	0.60	43.66	5.46

Figure 12 : Scanner different speeds operation summary

The MAX IN-OUT delay used for the calculation is the following:

$$t_{maxINOUT} = t_{BeamExtraction} - t_{WireRetraction} - t_{systemPrepare}$$

20 m/s => 805-20-275=510ms

15m/s => 805-53/2-275=504ms

10m/s => 805-79/2-275=491ms

1m/s => 805-570/2-275=245ms

Data is recorded continuously between in and out movement. The expert mode represent the motion and raw encoder's data that will be used until the optical position sensor is digitalized in the VME. Those measures are not meant to be transmitted through the optic link, and will serve as diagnostic data, with a direct connection to the intelligent drive to access it.

The operational mode will be the amount of data that will require transfers when motion and encoders data will be processed. This is the data that is meant to be transmitted through the link and stored.

Information type	Tangential speed [m/s]	Max IN OUT [s]	Total data produced[Mbits]	Time at disposal for data transfer [s]	Bandwidth needed [Mbits/s]
Raw	20	0.51	740.54	0.355	2086
Raw	15	0.504	765.65	0.355	2156
Raw	10	0.391	813.34	0.355	2291
Raw	1	0.245	1525.02	0.355	4295
Processed	20	0.51	21.2	0.355	59
Processed	15	0.504	21.92	0.355	61
Processed	10	0.391	23.28	0.355	65
Processed	1	0.245	43.66	0.355	122

Tableau 1

The bandwidth calculations made here is the worst case scenario, where the max INOUT time is used to record data, leaving only 355ms to transfer the generated data volume before the next cycle.

Processed information that needs to be transferred are requiring a bandwidth of max 122Mbits/s.

When dealing with unprocessed information however, the bandwidth requirement is much bigger, going up to almost 4.3Gbps.

The unprocessed information is not expected to pass through the link, but shouldn't be discarded as it would facilitate the debugging of the signal processing algorithms, removing the need of a direct connection to the intelligent drive.

BWS General needs summary

The beam wire scanner is a complex instrument, requiring different constrains for each type of information data type. Multiple information needs to pass through the optic link in order to operate it, such as movement parameters, movements triggering and acquired data upload. The protocol that manages the link, needs to be able to differentiate those different types of information, and adapt its timing constrains as well as its bandwidth constrains. The protocol needs to stay transparent, providing a virtual link between the two FPGAs for the user. Figure 13 presents one scenario of the movement triggers which will travel through this link.

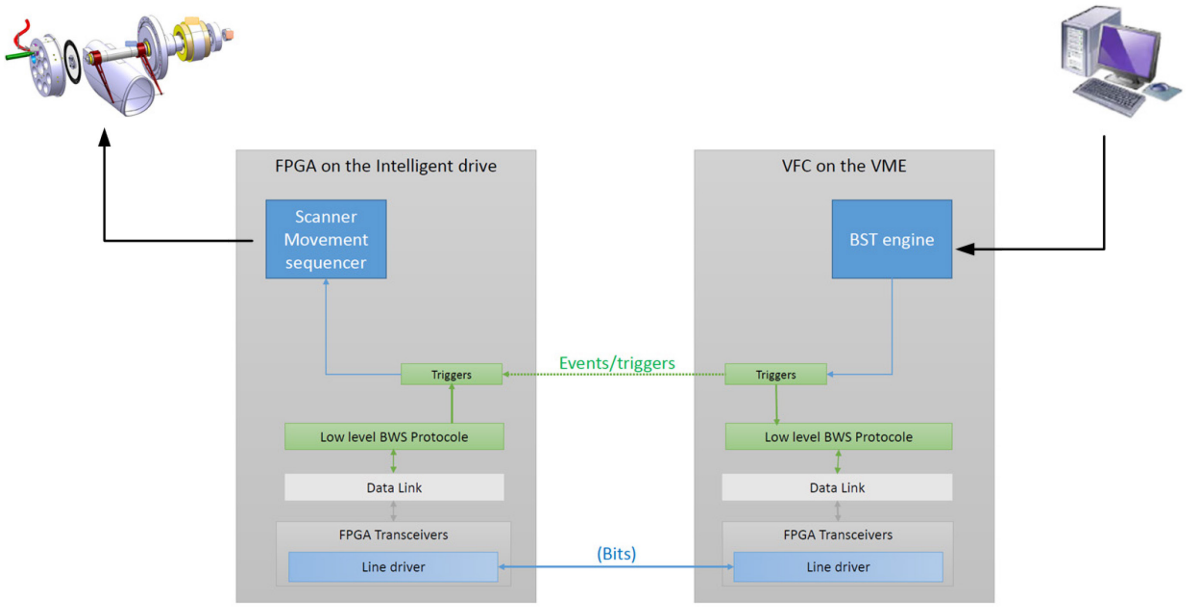


Figure 13: Scenario of movement triggers over the link

3. Technical Requirements

The technical requirements for this project will have to satisfy both the CERN and the HES-SO Master's thesis demands.

This project was given as part of a technical Student program at CERN and thus, does not necessary have the same expectations in terms of requirements or final result than the HES-SO. The technical requirements set at the beginning of this project as well as the planning will have to satisfy both sides.

3.1. HES-SO Technical requirements

The Master thesis is a R&D project with an industrial application that must be supervised by a master research unit teacher. The project should aim to put into practice and to validate the competences seen throughout the master courses, and apply project management skills. The difficulty should be at a sufficient level to validate the end results.

The project management skills are classified into 4 different groups:

- Project Management
- Analysis and product specification
- Development and creation
- Documentation

All of these should be present in the Master's thesis project. Each skill will have a different grate's weight, depending on the nature of the project.

Project Management

The project management skill include the technical advancement of the work, how to deal with costs and delays, communication management as well as risks management.

Analysis and product specification

This skill set should be centered on the multi-disciplinary aspect of the work. This should include how to decompose the workload, how to specify the system and make the conception. The risks of each part have to be analyzed, so that a plan can be specified.

Development and creation

The development and creation skillset evaluates how the detailed specification and analysis was done. It also focuses on the conception, the integration of the product with other components, as well as the system modelling and simulation. The skill also evaluates the testing and measurements methods done.

Documentation

This skillset will evaluate how the project was analyzed and criticized.

It will also judge what enhancements were proposed and the lessons learned from the project.

3.2. CERN Technical requirements

The CERN technical main requirements for this project is the specification and implementation of a protocol capable of transmitting and receiving different information and triggers through the optical link connecting the two sub-systems of the Beam Wire Scanner.

The main points the protocol should satisfy are the following:

1) Data integrity
   Error detection, correction and/or retransmission. Notification and statistics.
2) Event transport
   Trigger and IO port replication between the 2 ends. Link latency jitter < 0.1us. Automatic transmission time evaluation
3) Transparent interconnect SoC bus
   Interconnection of internal FPGA bus transparently (Memory Mapping), data blocks transfer between FPGA (2 directions)
4) Streaming links
   Interconnection of internal FPGA bus transparently (Stream interfaces: 1..N @ >= 1Mbit/s)transparent connections for streaming mechanism

   The optional features are the following:

1) Virtual JTAG over the link (investigation)
   JTAG chain over the link: Use of vendor tool for logic analyzer (SignalTap), memory/register content modification, probing by using Boundary Scan.

2) None-volatile memory management
   External flash management over the link. Read/write flash of program or settings

For more details, see Annexes:

[1] Technical student goals, listofgoals_wire-scanner_link.docx
[2] Beam Wire Scanner (BWS) serial link requirements and architecture, BWS-SerialLink-JEmery.pdf
[3] BWS motion control electronics, J. Emery & P. Andersson, BWS-Intelligent-Drive-Electronics-TB.pdf
[4] Proposition de projet de Master, J. Emery, CERNBeamWireScanner_time_critical_protocol_v2.docx

4. Analysis

4.1. CERN requirements implications

CERN technical requirements needed to be analyzed at the beginning of the project, in order to assess their feasibility and requirements. Certain technical aspects also needed to be better specified, in order to have a good overview of the workload and their implications.

The list below are the main points that needed more explanations or had to be discussed over:

- A distinction from the start had to be made between normal latency (delay) and jitter latency. The normal acceptable maximum latency should be defined as well by CERN.
  - ➔ The important aspect that came out of this discussion was that CERN is mainly concerned about the jitter latency for their triggers used to start scans at a precise moment in time. The total latency, as long as it's reasonable and within what other protocols do, is not important.
- A controlled jitter latency protocol means that all layers should be designed with a "constant latency" mindset in the way the information is handled. When dealing with different time domains, and time domain crossing, this latency is more difficult to handle and out of the developer's hands especially when using IPs. In this regard, the Physical layer, subject to that effect, should be carefully chosen.
  - ➔ CERN is already using different protocols within their experiments and departments. Being about to use an existing protocol, or part of a protocol would be a huge plus.
- Since multiple services will run together, a priority system should be implemented, that will need to handle bandwidth needs and fixed latency needs. A total effective bandwidth estimation will have to be made in order to assess the difficulty of the task.
  - ➔ CERN wants to stay in something simple but effective. There is no need to overcomplicate things for this points.
- Error detection or correction means a bi-directional communication protocol that does not only send information, but also expect an acknowledgement message in return. The implication of such a bi-directional communication will have an impact on the latency and in case of a retransmission, even the jitter latency. This will also have an implication on the bandwidth. CERN will need to specify when to consider a transmission failure (the number of retires. Etc) as opposed to a transmission recovery.
  - ➔ The jitter latency is only critical for triggers. The diminished bandwidth should by CERN calculations not be a problem as not many services will run at the same time, thus the bandwidth needed is relatively low compare to the total bandwidth of the link. Simulations will have to be made in order to confirm this point.
- Replication of signals with a fixed delay as well as I/O replication will need an Rx Buffer. A balance between latency/buffer size and number of packets send per unit of time should be analyzed.
  - ➔ With good priority in the bandwidth management, this should not be an issue.
- Burst Data transfers may need bigger packet size, to increase the effective bandwidth (protocol overhead size << Data size) depending on needs.
  - ➔ CERN is more concerned by having a relatively simple protocol but effective and easy to maintain, rather than more optimized and thus complex. To that regard, unless it is necessary, packet size should be kept constant.
- When dealing with volatile external memory or the FPGA's internal buses interconnection, an Avalon data request might be CPU blocking while the data is being fetched. Since the memory is on the other FPGA, the delay will be higher. CERN needs to specify if such a blocking time can be critical for the CPU speed (longer optical fiber = longer delays). Another solution than making the CPU wait might be needed.
  - ➔ As the logic that will connect itself to the protocol does not exist yet, this is something CERN will have to keep in mind. Instead of a read process initiated by the Master side, constantly asking when the Slave is ready, the Slave should send the information when the Master is ready. This will imply that both FPGAs must have a Master and Slave Avalon Service.

Required Information for the specification

To assert the technical specifications needed and to choose the protocol, several information needed to be sorted and answered first. The main issues were:

- Protocol overhead needed
    - How much bandwidth is still available for the different overheads?
    - How much overhead would each layer need (PHY, Mac, and application)?
    - Will bandwidth be a problem in the future?
- Physical layer and encoding
    - Will symbol encoding be bandwidth critical?
    - Can an existing CERN physical layer be used? What are their respected latencies?
    - Can an existing physical layer (commercial) be used? Is it free? How is jitter latency handle?
    - Is there a need and enough time to create a custom physical layer?
- Effective bandwidth needed
    - What is the total effective bandwidth needed by the services?
    - Will all the services work together? Can they work together (bandwidth and logic wise)?
- Acceptable latency
    - What is the acceptable latency for CERN?
    - What is the implication in terms of buffers (logic needed)?
    - How will the latency affect the different services?
- Jitter
    - What parts of the protocol are subject to jitter latency?
    - How much jitter do the different PHY layers have? Is it acceptable?
    - How will error retransmission affect jitter latency?
- Error correction
    - What method will be used to detect errors?
    - How many errors should the protocol be able to handle and correct?
    - Is the trade-off between more recovery bits and less bandwidth worth it?
- Bi directional Packet acknowledgement latency effect
    - How will the architecture handle this?
    - How will the fiber optic length affect the latency?
    - How much Tx buffering (keeping transmitted packets) is needed? How much logic will it take?
- Packet retransmission effects
    - How will retransmitting packets affect overall bandwidth? Overall jitter latency?
    - How complex is it to implement?
    - How many retransmissions tries should the protocol be able to handle?
    - What to do when the transmission is considered a failure?
- Logic needed
    - How much logic does the protocol has at disposal?
    - What other code will be running on the FPGA that will take space?
    - How critical should the protocol's footprint be?

This thesis is also about creating a generic and modular protocol. There needs to be a discussion in regards of those approaches with CERN to know exactly what parts of the protocol is concerned.

All those aspects need to be defined in order to make a specification that will respect the requirements:

- Bandwidth needs vs Latency needs
- Packet size vs FPGA Logic (Buffering)
- Complexity vs Time at disposal

Each point has been analyzed and evaluated, to be able to answer the questions asked in this section. For each critical point concerning a technical solution, a decision has been made in order to start the implementation.

4.2. Bandwidth assessment

First and foremost, the effective bandwidth needs to be calculated. This includes all the services that should or could be running at the same time on the link. An approximation of the raw data needs to be made, in order to better understand the needs and restriction of the system.

The different services that should be implemented and looked into are the following:

- Trigger and I/Os replication, from 1 FPGA to the other
- Interconnection SoC bus
- Streaming links
- Virtual JTAG
- None volatile memory management

On top of that, an estimation of the bandwidth needed for the error Detections, notifications, and potential retransmission needs to be done.

Trigger and I/Os replication

Since those data's are latency dependent, an RX buffer is needed.

For the I/O replication, a RX buffer needs to be filled by at least the amount of retires * time of 1 transmission for each bit before releasing the data on the RX I/O.

This will ensure that even if Data transmissions errors occurs, the state of the IO can constantly be streamed out.

In case of an I/O, if no transmission errors occur, it will need the bandwidth of the sampling clock. In case of a GBT system, with a 40 MHz clock, 40Mbps will be needed per bit.

If we settle for 10 inputs/outputs transparent streaming, this service will require a bandwidth of around **400 Mbps**.

If 1/10 packets have transmission problems which should never be this high, then the bandwidth needed will be around 440 Mbps.

Triggers could either be considered as I/O and replicated each clock cycle, or in case of a one-time event trigger, could be passed through a single specific frame, which will be latency rather than bandwidth critical.

Interconnection SoC bus

The Interconnection bus shall transfer data transparently (Memory Mapping) between the 2 FPGAs. The transfer shall be bi-directional.

An approximation bandwidth was given for this Service in the 2.3 BWS Section as **118 Mbps** for PSB (The Proton Synchrotron Booster) for processed data. Unprocessed data can go up to **4295 Mbps**, but most tests would only need **2200 Mbps.**

Streaming links

The protocol needs to be able to reproduce streaming links transparently. (1..N @ >= 1Mbit/s)

This is mainly used for low acquisition signals that needs constant monitoring.

An estimation is made to around **100 Mbps.**

Virtual JTAG

Virtual JTAG is a serial link, with a clock that can be set from 1 Mhz to 40 Mhz, depending on devices capabilities and interfaces (Parallel, USB).

Thus, the maximum bandwidth needed from this service will be 40 Mbps per signal at the most.

The JTAG virtual IP needs 4 signals as a minimum to function.

The total bandwidth can be approximated to **160 Mbps** with the minimum signals and max frequency, but could be set lower if needed.

None volatile memory management

None volatile memory is usually done via SPI, or another serial link. On some, 4 bits of data can be used, to accelerate the programming. This can be seen by the protocol as a simple IO replication.

Thus the bandwidth needed for this service is also 40Mbps per signal with a 40 MHz clock. Usually, there is around 5 to 10 pins for serial links or semi serial links, so an approximated bandwidth can be done are around **200 to 400 Mbps.**

Recapitulation table of effective bandwidth needs

Service	**Bandwidth needed (Mbps)**	Comments
Trigger and I/Os replication	400	40 Mbps per IO, based on 10 IO
Interconnection SoC bus	120/2200	For PSB
Streaming links	100	10 links 10Mbps each
Virtual JTAG	160	For 4 pins max frequ.
Non-volatile Memory management	400	Replication of 10 pins@40Mhz

Tableau 2

Total bandwidth:

With processed BWS data: **400+120+100+160+400 = 1180 Mbps**

With unprocessed BWS data: **400+2200+100+160+400 = 3260 Mbps**

When taking all the services together, the data effective bandwidth would be manageable with a 4.8Gbps line rate for the processed data case. When considering higher needs in the SoC interconnect bus, running all services together will begin to be problematic. Adding the application protocol stack overhead (estimation of about 30%), the output needed would be in the range of 4238Mbps, which gives little margin. This case is however not a criteria for this thesis, and only the processed data should be taken into real consideration.

4.3. Physical layer and encoding

The physical layer has to be thought based on but not limited to the following elements:

- CERN specific physical Layers (GBTx, White Rabbit)
- Commercial Physical Layers (Ethernet, USB 3, Altera Interlaken etc)
- Custom physical layer

For this project, the jitter latency needs to be kept low, and the physical layer should be chosen based on its control. With an encrypted IP, there is no control over the jitter, only what is available for the user.

When defining the physical layer, it's also important to take into consideration the bandwidth requirement due to the encoding line overhead (8b/10b, 128b/132b etc), and the protocol overhead (syn bytes, headers, CRC, etc)
Depending on the needed effective data bandwidth, this could have an impact on the decisions.

Additionally, the ease of implementation has to be looked at. A custom physical layer will take more time to be implemented and tested, whereas an existing one, might restrict the possibilities, but will be faster to integrate and test. The cost or licenses needed for specific IP protocols should also be taken into consideration.

CERN specific protocol or physical Layers

White Rabbit

"White Rabbit is a fully deterministic Ethernet-based network for general purpose data transfer and synchronization. It can synchronize over 1000 nodes with sub-ns accuracy over fiber lengths of up to 10 km. commercially available."[1].

White Rabbit allows you to precision time-tag measured data and lets you trigger data taking in large installations while at the same time using the same network to transmit data [1]:

- sub-nanosecond synchronization
- connecting thousands of nodes
- typical distances of 10 km between nodes
- Ethernet-based gigabit rate reliable data transfer
- fully open hardware, firmware and software
- multi-vendor commercially produced hardware

One of the mode can could be used in our case with this protocol is WR Streamers, that provide to the user a FIFO-like interface over Ethernet, as shown by the picture below:

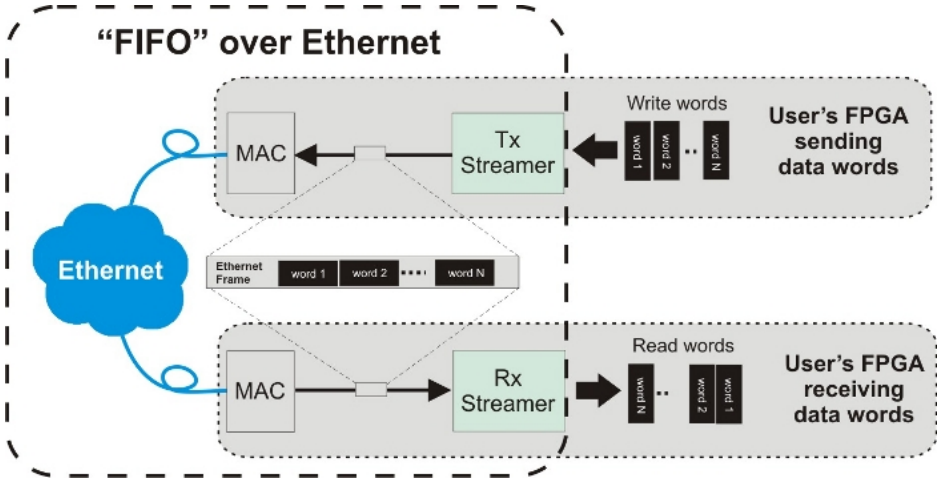


Figure 14 : white Rabbit protocol link concept

This protocol however is based on a gigabit Ethernet protocol (line rate of 1.25Gbps), and is too limited bandwidth wise for this project, based on preliminary estimations (see Effective bandwidth needed section). Since the new 2.5GBASE-T and 5GBASE-T Ethernet standards that could potentially benefit this project will be based on 10GBASE-T [2], it will make the adaptation of this protocol a time consuming task and therefore is not considered as a viable solution.

GBTx

The GBTx is an ASIC radiation tolerant chip developed by CERN that can be used to implement multipurpose high speed (3.2-4.48Gbps user bandwidth) bidirectional optical links for high-energy physics experiments [3].

This chip is using a link that establishes a point-to-point, optical, bidirectional (two fibers), constant latency connection that can function with very high reliability in the harsh radiation environment typical of high energy physics experiments at LHC [3].

The firmware required to allow one or more FPGAs to communicate with the GBTx chipset over the versatile link is handled by the GBT-FPGA project.

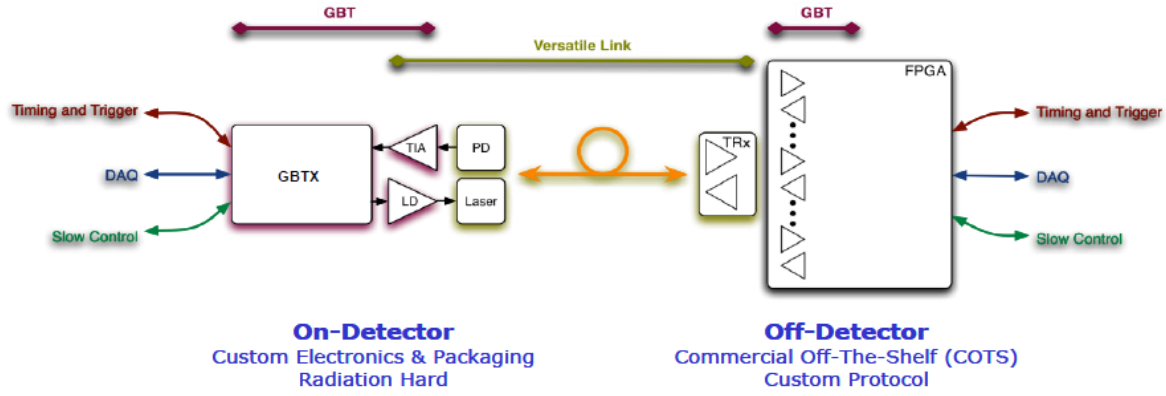


Figure 15: GBT ASIC with GB FPGA Link

The GBT-FPGA core is offered as a full library, compatible with Altera and Xilinx's FPGAs, with 2 types of implementations: Standard or Latency-Optimized. Standard version, is targeted for non-time critical applications and the "Latency-Optimized" version, is ensuring a fixed, low and deterministic latency of the clock and data (at the cost of a more complex implementation).

The encoding mode can also be configured, with GBT-Frames or Wide-Bus, and optionally 8b10b.

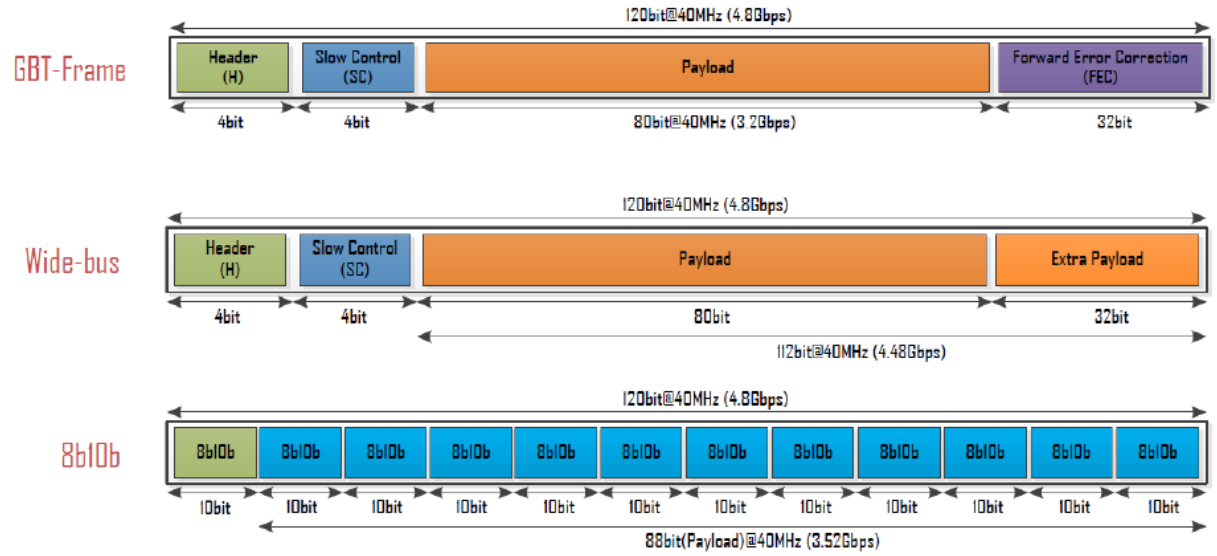


Figure 16 : GBT frame format possibilities

This physical layer could be very interesting for this project, especially in the latency optimized version. Certain restrictions however should be noted, such as the usable bandwidth and the already established frames definitions.

The support for the ArriaV FPGAs is also still in development, and a big risk factor should be considered if this option is chosen, as it could be time critical for the advancement of this project.

The optimized latency mode is also not supported on all FPGAs, as the RX clock is constantly checked and shifted to follow the RX clock Recovery frequency. If not properly done, this could lead to meta stability issues.

The normal latency mode needs to be analyzed, to see whenever the jitter is acceptable for this project (< 100ns jitter).

The GBT frame format offering a forward Error Correction is also very interesting and a good argument to choose this physical layer.

Commercial Physical Layers

Other protocols physical layers could also be implemented, such as Ethernet, USB 3.0, and Interlaken. Those are given as an IP form, and require certain compliances such as:

- Specific line rate
- Specific Packet format and packet interframing that has to be respected

Those PHY layers or MAC layers are also generally not free. Those IPs, either require a special license in the case of Ethernet or Interlaken, or in the case of a USB 3.0 physical layer and MAC, must be commercially bought.

Research shows that most protocols, including USB 3.0, can either guarantee delivery of data,, but not guarantee bandwidth or latency. In a similar way, if bandwidth is guaranteed, errors are not dealt with.

Since the USB 3.0 standard is at 5Gbps, it would seem like a good candidate. However, when looking at the type of transfers supported, it deviate from what this thesis protocol needs to achieve.

Bulk Transfers: (USB3.0 spec p 65)

- Access to the SuperSpeed bus on a bandwidth available basis

- Guaranteed delivery of data, but no guarantee of bandwidth or latency

Isochronous Transfer:

As in USB 2.0, the SuperSpeed isochronous transfer type is intended to support streams that want to perform error tolerant, periodic transfers within a bounded service interval.

- Guaranteed bandwidth for transaction attempts on the SuperSpeed bus with bounded latency
- Guaranteed data rate through the pipe as long as data is provided to the pipe

Errors may prevent the successful exchange of data within the service interval bound, however since the packets in an isochronous transaction are not acknowledged, a host has no way of knowing which packets were not received successfully and hence will not retry packets.

For the Interlaken, Data is transmitted by burst. The minimum size of Data Burst is 32 bytes.

The bandwidth of the Interlaken interface is divided into data bursts from the supported channels. Data packets are transferred across the interface by means of one or more bursts, with the bursts delineated by means of one or more Control Words. [15]

No real information could be found concerning the jitter in the clock domain crossing.

As CERN wants to push to use its own physical layer, the GBTx, the research done on commercial protocol was not as extensive as it could have been. Overall however, the research done showed little information regarding the jitter in transmissions, and usually no access to it. The protocol overhead is usually also very high when considering small frame size, with a lot of information that is not necessary for this thesis protocol.

Since the GBT layer offers latency jitter control, as well as error correction, while keeping the frames small for easy retransmission purposes, it was the natural choice for a physical layer.

4.4. Protocol overhead

Since the effective data bandwidth was determined at 1180 Mbps, if the GBT physical layer is used providing 3.2Gbps of data bandwidth, the protocol overhead is not critical and can be freely chosen.

The question remains, if a specific physical layer like the GBT should be used, how to encapsulate the frames.

With GBT, if we format the data and protocol into the existent 80 bits at our disposal, then the overall layout could be regarded as such:

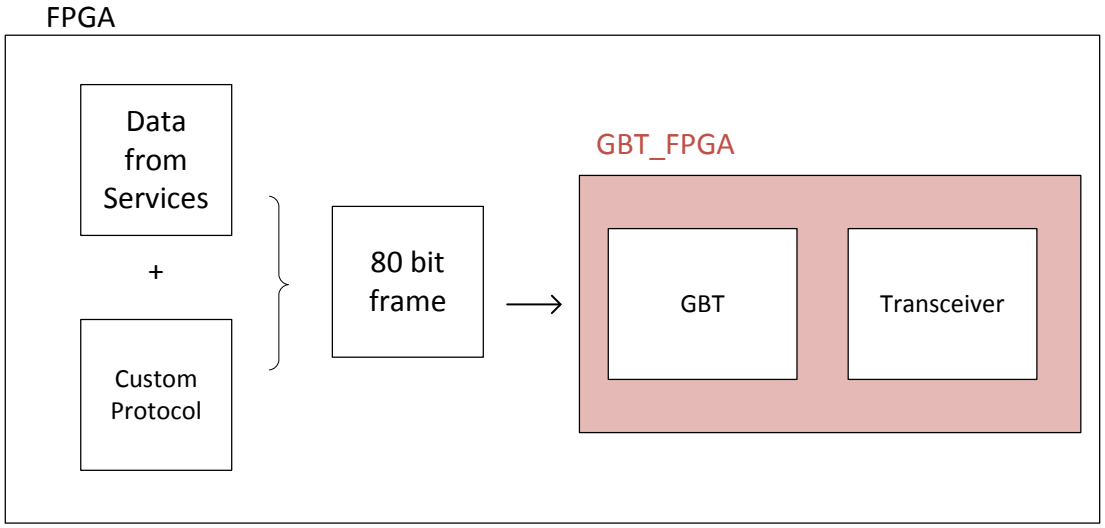


Figure 17: Initial frame encapsulation analysis

The current Physical layer provided by the GBT_FPGA team can be reused without any changes to the PHY layer. Thus, the protocol overhead should be integrated into the GBT-Frame.

An estimation of a simple Protocol overhead would need:

- A Service number field: about 5 bits for the Number of Services that can be used simultaneously: 2**5= 32 Services can be used at the same time
- Type field: about 2 bits for the tram types: 4 types of different trams per services.
- An ID increment field: to track the packets, about 8 bits, for 256 different counter state.

The type's field will be used when 1 GBTx frame isn't enough to transfer all the necessary information. They will code and dictate the splitting of the information on multiple GBTx trams.

This will have a direct effect on the effective data bandwidth as some bits will potentially be unused in certain type of trams, not to overcomplicate the protocol.

For example, in the case of a memory transfer:

- Tram 1 could contain the starting address and the number of words to transfer.
- Tram 2 could contain the data

There are two main options on how to integrate this protocol:

- Use the GBT-Frames as the only frames, and thus limiting the payload size
- Encapsulate bigger frames into the GBT-Frames.

When using the GBT frames the User Data bits available are decreased with the increase need of protocol overhead, as seen by the picture below:

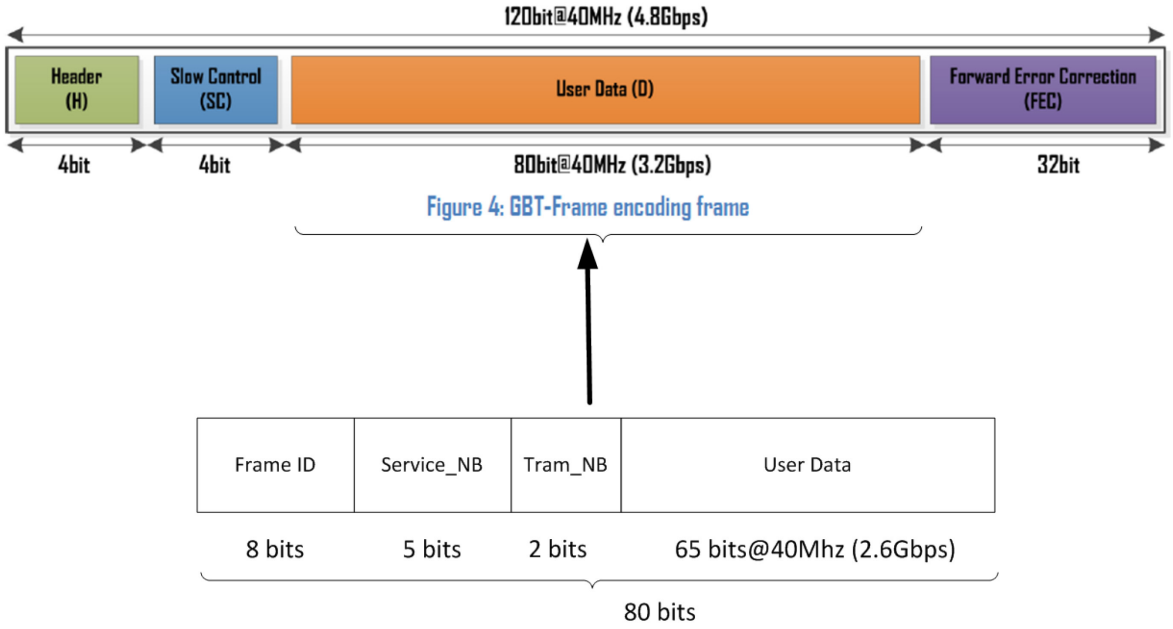


Figure 18: Initial frames format analysis

Instead of trying to fit the protocol into a GBT frame, we can create our own frames and encapsulate them into a GBT frame.

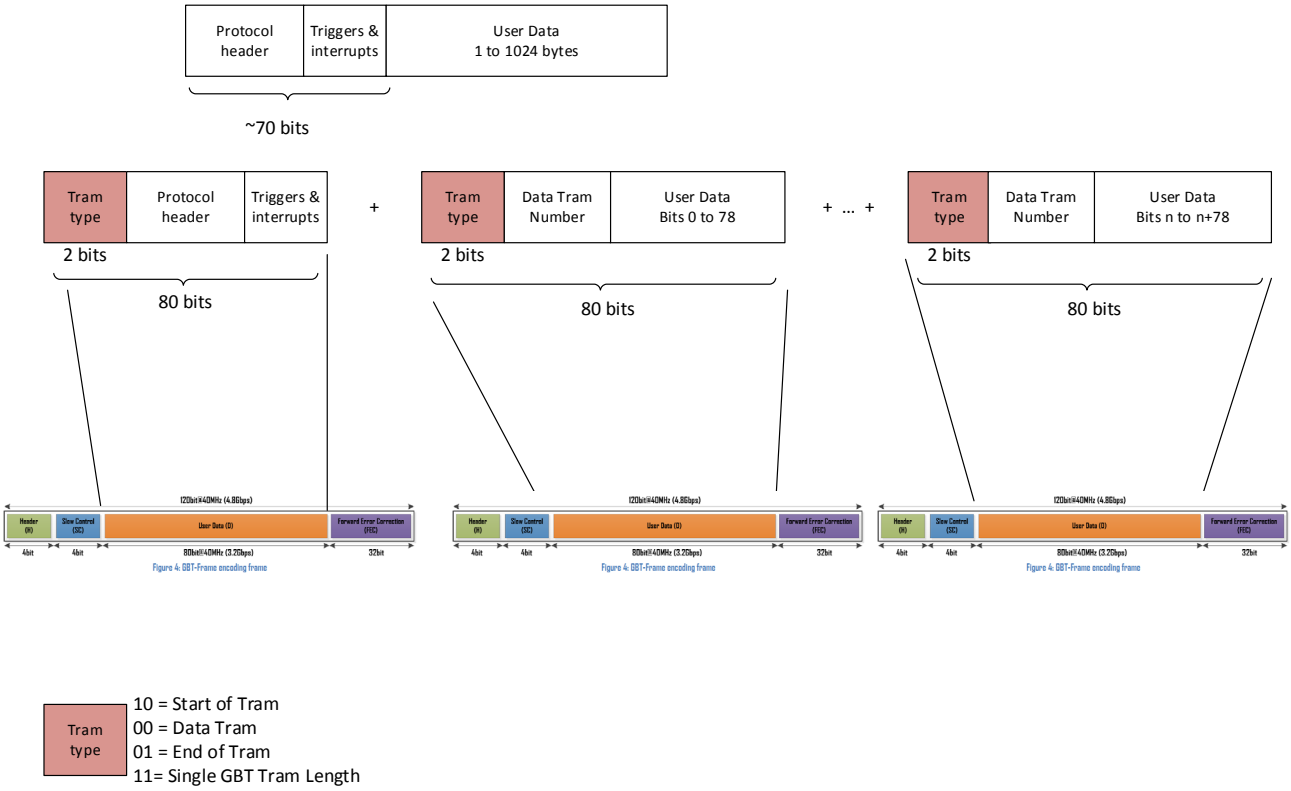


Figure 19: frames encapsulation ideal

The Tram Type is needed to know on a MAC level where the data begins and ends. This is usually done in the PHY layer, and is already done to generate the GBT frames.

Encoding this into the Data ensures that any physical layer will be compliant with the protocol.

Encoding some type of tram Data ID is also needed, to know if a GBT tram was not properly detected and just skipped. The integrity of the frame is done in the GBT PHY layer. There is no real info however if every frame was seen in the Transceivers.

This can be done by Tagging a transaction GBT frame, or recreating a CRC inside the last data frame.

This 2nd option would increase the Bandwidth on long packet size, but would add latency in the error detection. Therefore, if we want to limit the logical elements used, in this case FIFOs, when validating a Tram, the 1st option is better.

To answer both bandwidth and latency problems, protocols usually have variable payloads. This 2nd method ensures that if information needs to be send fast, the payload will be short. This impacts the effective bandwidth of course, depending on the protocol header's length.

Typically, if the protocol header is 20 bytes long, and the data only 10 bytes, then the effective bandwidth is only 1/3 of the line's data rate.

The more the payload length increases, the more effective bandwidth is available.

If we use the GBT frames, as the actual protocol's frames, then we are limited in our payload's length dramatically as the protocol's overhead size is increasing and would be present on every single frame.

With the encapsulation method, Application layer overhead can also be longer without affection the bandwidth as much, but the complexity and data ordering will be harder to keep.

4.5. Error detection

Error detection can either be just detected and not act upon, or detected and dealt with, to insure a successful transmission, even when present.

Most protocols add a cyclic redundancy check (CRC) at the end of a packet to validate the transmission integrity. This method is efficient to check for errors, without taking too much bandwidth. It however can't repair the error, and the higher layers of protocol need to deal with the packet retransmission.

Another method is to include extra bits per packets, like a reed-Solomon or hamming code to be able to recover from a single error. Those methods require usually a relative fix ratio between the number of extra bits needed and the amount of data sent and thus, are more bandwidth demanding than a simple CRC.

Those methods, while being able to detect errors or even recover some of them are still only done on the reception side (RX), and to make sure a transmission was successfully transmitted, there is still a need for a message of acknowledgement to be send from the RX side to the Transmission side (TX).

Sending 2 of the same packets in a row is also an option, but this would double the bandwidth needed, and still doesn't guarantee that the transmission was successful.

The protocol needs to be able to balance between bandwidth needs and data integrity in case of errors. This will require more buffering logic. The protocol should thus implement an acknowledge system, to make sure all the information is passing through. This is vital when streaming datas, or when reconfiguring the FPGA.

This response from the other FPGA will not be instantaneous, as 1km fiber optic cable will induce a 5 us delay on the line, and another 5 us delay to come back. Meaning each frame will need approximately 10 us to be validated from the moment the TX transaction ends to the time the RX transaction comes back.

There are two ways to deal with that:

- Blocking transmissions
- None blocking transmissions

With blocking transmissions, the acknowledge frame for a first transmission needs to be received in order to proceed with the next one. This greatly limits the effective bandwidth, but doesn't require more logic in the FPGA. In the GBT example, instead of 80 bits every 25ns (3.2 Gbps) the bandwidth drops to Only 5 Mbps when sending 80bits every 10us.This method is inappropriate for this project, with short transmissions payloads and long delay in the response time.

With none blocking transmissions, the TX side doesn't wait for a response before sending the next packet. This method has much lower impact on the effective bandwidth, but needs more logic and buffering:

- Transmissions need to be tagged in order to validate/invalidate them properly.
- TX transmissions need to be kept or buffered until they are validated.

In this case, if GBT frames are sent back to back, the data that need to be buffered is 80 bits every 25 ns (1 clock cycle at 40Mhz). For a 10 us delay, this implies 400 clock cycles so 80*400= 32kbits of buffer. This approximated number is easily achieved in the ArriaV that offers around 20'000 Kbit of ram (around 2000 M10k), depending on the FPGA size. If a frame is not well received or even not received at all, another mechanism should be implemented to resend the transmission, based on a transmission tag, or frame number.

This also has implications in the fixed latency calculation. In order to have a fixed jitter, a number of transmission retries has to be set in the protocol. If a data is latency dependent, then the data should be stored for the amount of retires * time of 1 transmission on the RX side, even if no errors were detected to guarantee a fixed latency.

If no errors are detected, the only impact on the bandwidth is the transmission tags that will take some extra bits. To limit the bits used by this mechanism, this could be kept to a minimum. A good approximation could be 8 bits, for 256 different tags. This can ensure the detection of transmissions errors that were not received on the RX side. It needs to be large enough to account for a the transmission latency too.

This would remove **320Mbps** of effective bandwidth for the protocol, in case of a 4.8Gbps line rate such as GBT.

In case of a retransmission, the Bandwidth can drop significantly based on the architecture of the design and how missing packets will be handle by the higher stacks of the protocol.

Such a design could be made to disregard incoming packets until the missing one is retransmitted. This architecture is easier to implement, cost less logic and buffering, but will decrease effective bandwidth significantly. On the other hand, a design that can rearrange packet order, will need a lot bigger frame buffer, will be a lot more complex, but will have a lesser impact on the throughput.

In case of a GBT frame, if looking at a single service using all the line bandwidth, if transmissions errors occur that cannot be corrected for every $5*10^9$ bits, then the bandwidth will drop to 0 for 10us, before starting up again. Depending on the time interval at disposal, this will impact the system differently. For a set of data that needs to be transferred in 1 sec, this has almost no impact, removing $1/10^5$ of the bandwidth.

If the service's data is time sensitive, and buffer sensitive, in case of a I/O reproduction for example, this can potentially be a problem if not enough buffer space was anticipated and something to keep in mind when the implementation will be done.

4.6. Packet acknowledgement and retransmission

Information that is latency sensitive needs special care when dealing with retransmissions.

Going a bit deeper into the architecture can help see the complexity. The example below is done with the GBT PHY layer, but could be done with any other.

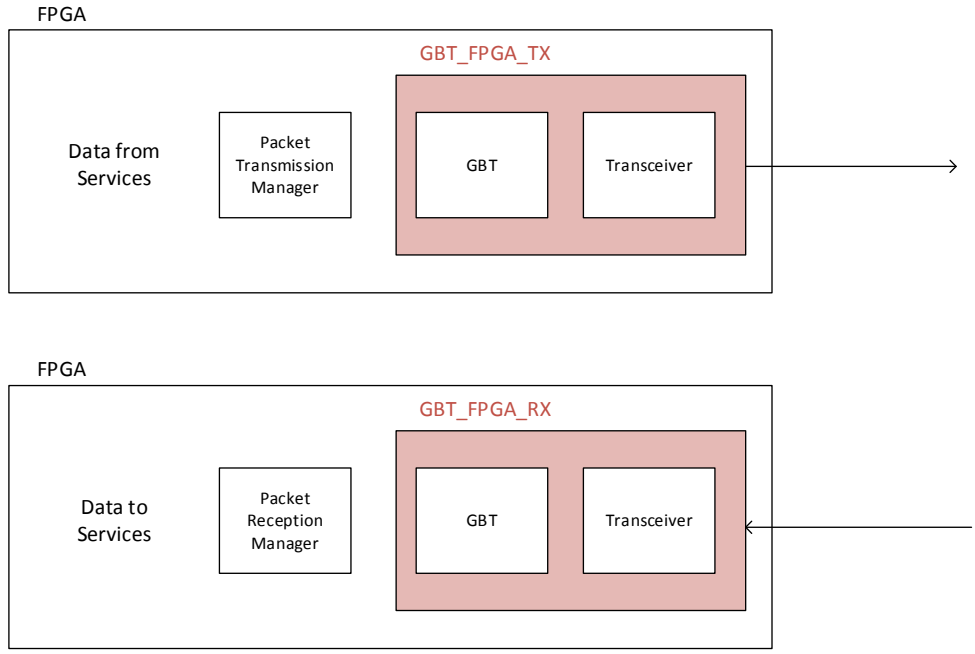


Figure 20: General Packet Transmission view

The next figures shows the initial idea of the retransmission TX side.

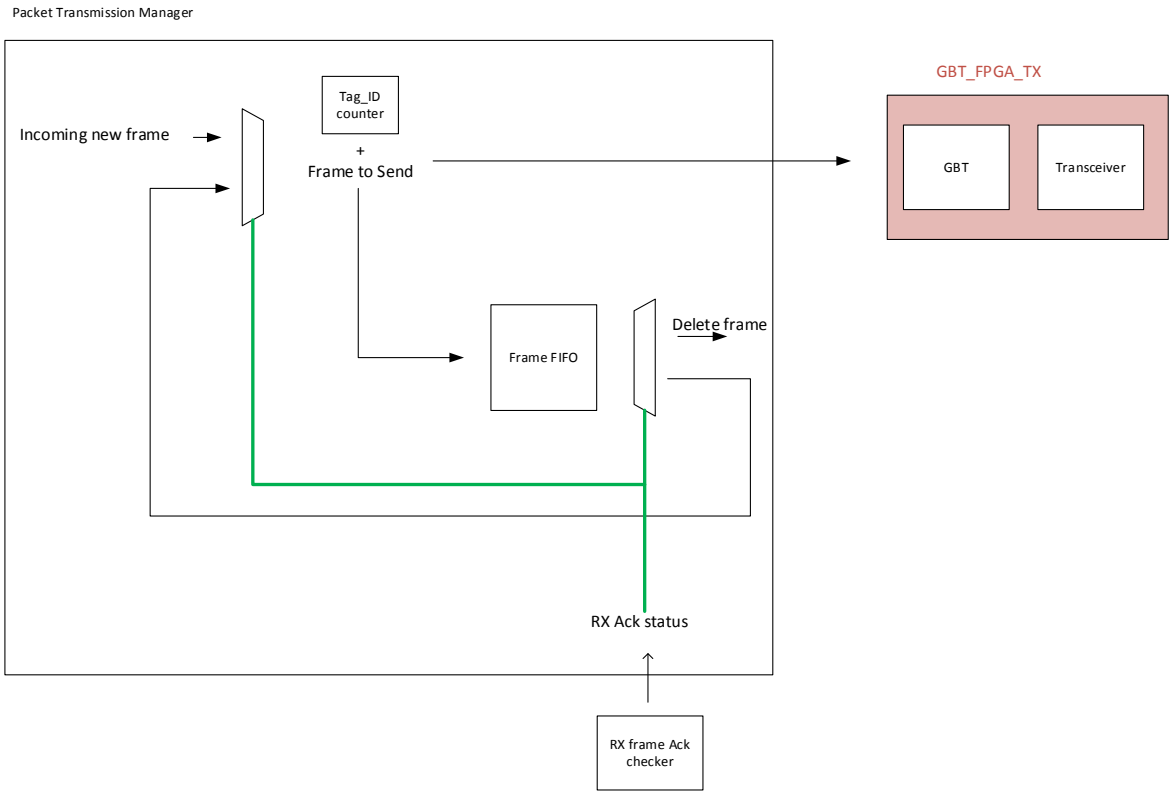


Figure 21 : Retransmission Manager Architecture

If acks are good, then the buffered frame is deleted, and the new incoming frame is read and transmitted to the GBT TX and written to the FIFO.

IF the Ack is not good, or doesn't match the tag ID from the FIFO output, then the frame is resent, and no read is made on the new incoming frame.

The FIFO requirement for the Frames is purely based on the latency of the transmission. The longer it takes for a response, the bigger the FIFO will have to be. If the GBT phy layer isn't used, then packets validation might be of different length, and it will add another layer of complexity into the frames fifo management.

Acks send management, not shown here, should be looked closely upon as they should be concatenated into a single frame, not to eat too much bandwidth. At the same time, if only 1 packet is sent on the TX side, the ack on the RX side should be send at some point, so waiting for a number of frames to be received before sending an Ack frame has limitations.

The RX side, upon the detection a wrong frame or a missing frame seems a lot more complex to implement.

In the effective bandwidth section, it is mentioned that the protocol could disregard incoming valid frames until the missing frame is received. Having a reordering packet management should all the frames be kept will add complexity in the RX management. The RX frame buffering will be more complex, and the RX acknowledge frames will be also a lot more complex to handle.

Since the bandwidth required by the different services isn't critical compare to the total bandwidth available, this project should focus on a simple method of packet acknowledgment.

Regarding the fix latency critical data, since the retries will come at different times it will be hard to guarantee this fixed latency as it's not enough to buffer the data into a FIFO. The time factor should also be looked into, and for each frame inside the buffer, which could be 400 frames or more.

Other methods should be looked into, to limit the complexity, such as sending the packet multiple times, so that the RX latency is easier to do.

In an example for 3 same packets sent back to back, the latency is known, and the trigger can be pipelined 3 clock cycles. This however triple the bandwidth needed for those special latency dependent packets.

Another way to deal with those latency triggers is to send them in every frame.

4.7. Priority Management and Services behaviors

A priority management should be implemented to differentiate between latency critical data, bandwidth critical data, and filling bandwidth data. Not all services will require the same priority and thus, will also require different services behaviors. A couple of behavior models can be created to easily integrate the different services mentioned and to create new ones.

Reproducing Pin/signals from IO or interfaces

This behavior is suited for low Pin counts or signals interface that needs to be reproduced on the other side. This is especially well suited to reproduce a protocol model, where signals can change on each clock cycle. In this case, signals are transmitted in their raw state.

On that particular model, each signal is constantly sampled on each clock cycle and transmitted. It can therefore be very bandwidth demanding if the number of signals increase, such as reproducing a parallel data bus, but bypass the need to make a BFM (Bus functional Model) interface.

In the case of a bi directional latency interface, this method can be used if a wait signal is available. If not, then since the response comes from the other FPGA board with a big delay this method may not work.

Interface communication (BFM)

This mode should be used when large quantities of data that need to be send isn't always active.

Those interface consist of reproducing the BFM (Bus functional Model) on both sides.

We then can reproduce the behavior of a known interface such as:

- Avalon Streaming interface
- Avalon Memory Mapped interface

Other interface could also be made available such as:

- Avalon Master Interface
- Avalon Slave interface

BMF behavior can be split into 2 classes:

- Wait or ready signal available
- No ready signal

When having the wait or ready signal, the transmitted transaction on the RX side can begin without the need to have all the information and data in the buffer. This would usually be the case for an Avalon Streaming interface.

When no wait signal is available, then the BFM model needs to wait until all the information for the transaction has arrived on the RX side, and thus needs to buffer a lot more information.

For the priority management, since multiple services will have to coexist, a scheduler will have to be implemented to choose which service frames needs to be send. The next figure shows the initial thought on how this management should look like in the system.

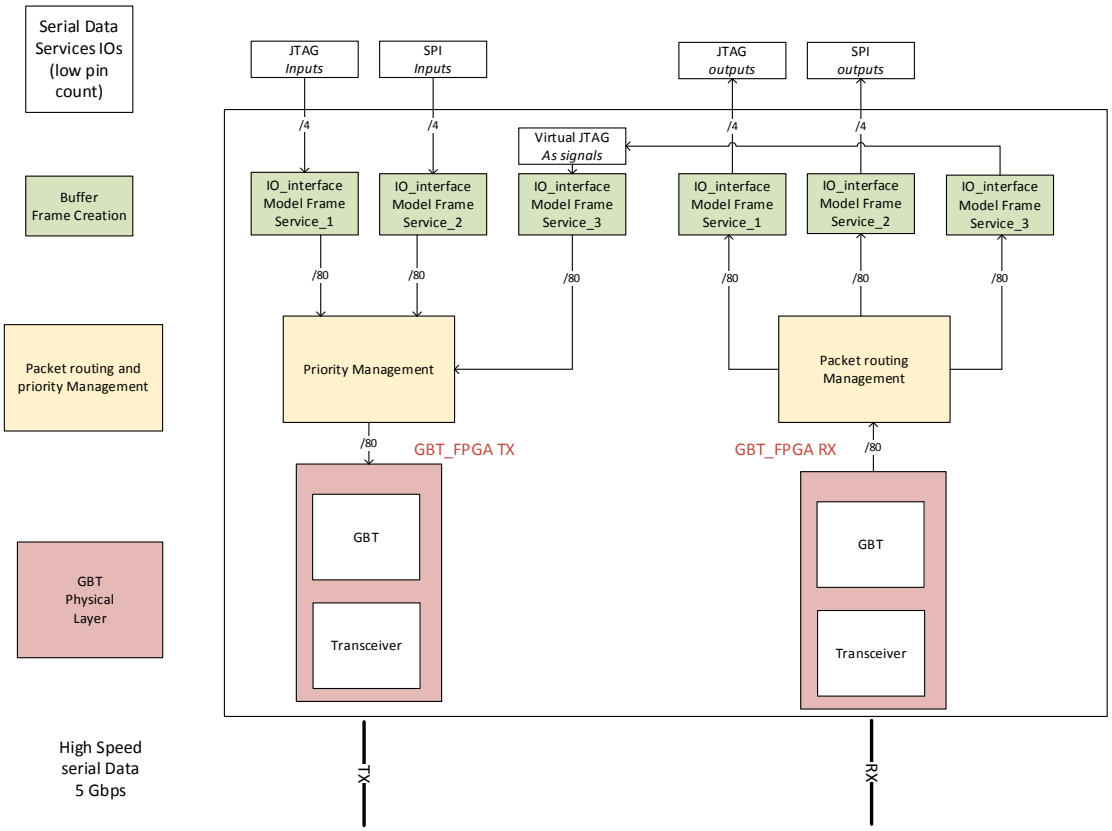


Figure 22 : Priority Initial Architecture

4.8. Selected technical solutions

Based on the above information and analysis, here are the technical decisions that were taken for this thesis:

- Physical Layer:

  Candidates for a Physical layer were down to the GBT or a custom one. A lot of CERN groups are currently using the GBT ASIC for their experiments as well as the GBT_FPGA. Choosing this PHY layer will benefit the CERN and will be easier to integrate in other CERN departments rather than creating a new one. It was thus decided to use this core as the physical layer of this thesis. However, a meeting was done with the team working on the GBT adaptation of the ArriaV in order to determine the availability of the release, and to determine the risk factor linked to such a physical layer. The choice to continue in this direction had to be decided rapidly in order to plan more time for a new PHY layer choice.

- Bandwidth needs:

  In normal conditions, the bandwidth needs is well within the GBT capabilities, even when considering all services running at the same time. Should the unprocessed data from the scanner be stored, then optional services will be switched off for this special need. Bandwidth requirement would be around 2200 Mbps.

- Protocol framing:

  It was decided to keep the framing as simple as possible, using the GBT frames. The protocol overhead will be present on each of the GBT frames, with no variable payload.

- Error detection:

  Since the normal needed effective bandwidth is relatively low compare to the line rate, the focus will be on error detection and recovery, as well as latency consistency, in order to have a transparent link for the application layer.

- Service behavior

  The focus should be on creating different behavioral models for the services, managing bandwidth, priority and latency. The models need to be user friendly, and easily adaptable to different interfaces such as Altera's Avalon interfaces, the wishbone interface or Xilinx's Axis interfaces.

- Complexity

  Overall, a good compromise needs to be done between the logic needed for the protocol, and the complexity of it. Keeping an architecture based on the OSI layer was decided to be the best approach.

5. Implementation

5.1. Implementation Philosophy

Vhdl coding style

- No state machine coding to avoid corner cases.
  - Less extensive simulation needed

- Generic code, less “eye friendly” but:
  - Key features regrouped into packages for users, easy access
  - Easy code maintenance
  -
- Records and functions to pack and unpack.
  - Going from record to bus vectors

- A lot of effort was put into not having to change multiple files when adding or removing services to the protocol. Layers communicate though records or array of records, all define in the protocol package file. This makes the entire protocol easy to manipulate by the user.
- All layers are using specific records, regrouping their demanding field bits. This facilitate but also keep the implementation as generic as possible, only having to change a single record definition in the protocol package.

Programs and version used

For the simulation, coded in vhdl 2008, **ModelSim ALTERA 10.4d** was used to be able to correctly simulate the UVVM code.

For the implementation, **Quartus Prime 16.0.0 Build 211 Standard Edition** was used to create the bitstreams.

Hierarchical File Structure

Under the main folder *Generic_Protocol* here is the main file structure of the different files:

- Quartus_project
- Sim
    - UVVM_cern
        - Cern_gbt_tb
        - Cern_vip_AvalonSlave
        - Cern_vip_gbt
        - Cern_vip_Traffic_Gen
        - Cern_vip_trigger
        - Simulation
            - Scripts
    - UVVM_v1_5_1
- Src
    - IP_Altera
        - FIFO_TX_Network
        - FIFO_RX_PHY
        - FIFO_AckFrames
        - PLL
    - IP_Cern
        - GbtFpga
    - Protocol_interface
        - Services_templates

- The subfolder Quartus_project contains the quartus project files only.
- The *Sim/UVVM v1_5_1* subfolder contains the bitvis source folder, with a few modifications made in the Avalon_mm_simple_access agent.
- The *sim/UVVM_cern* contains the simulation files developed for the test bench, and all the agents useful for simulation. The Simulation folder contains all the Scripts necessary to compile and launch the simulation. More information on this subject can be found under the Simulation section.
- The *Src* subfolder contains all the necessary files used by either the Quartus project or the simulation scripts. Following subfolders are used to differentiate files created for the protocol interface and its external IP used.

5.2. General View in an FPGA implementation

CERN provided in the specification a suggested general overview of the two FPGAs and the protocol to implement, that can be seen in the figure below.

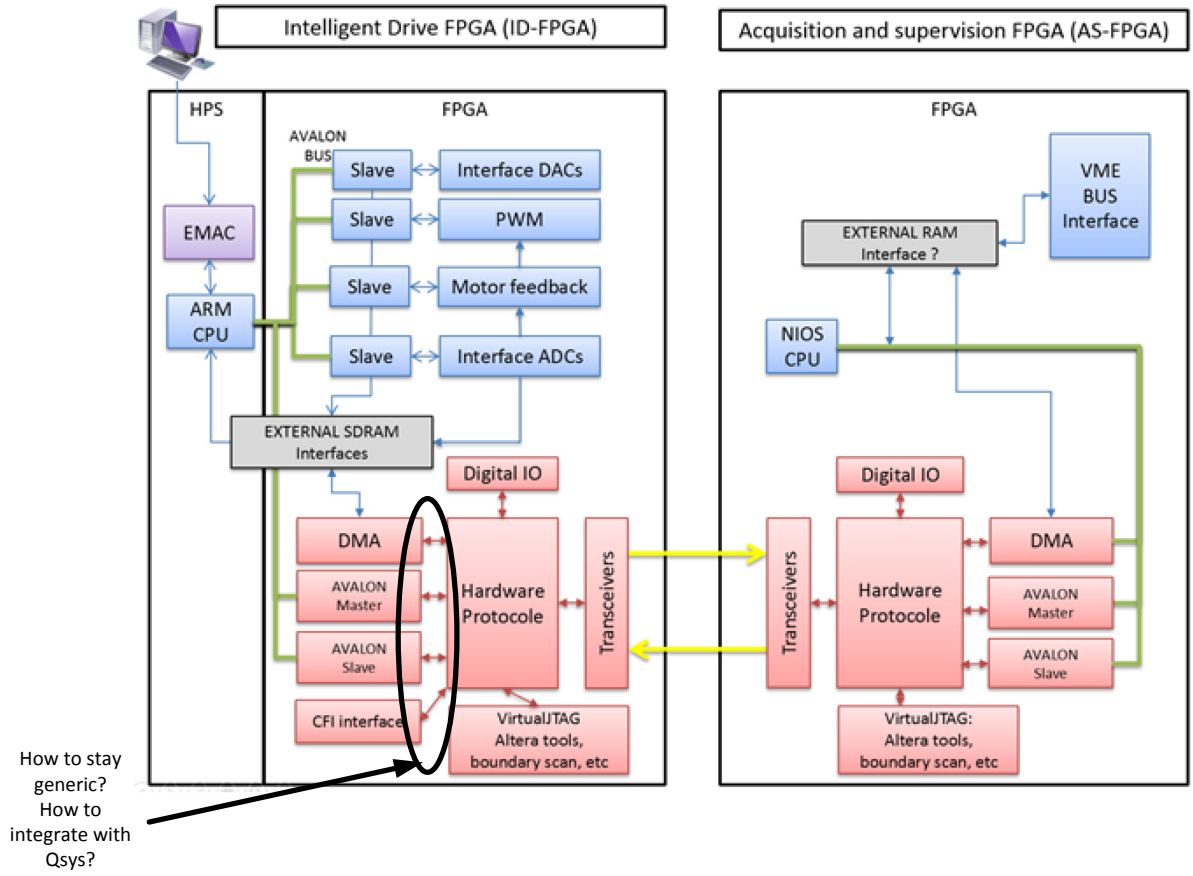


Figure 23 : CERN general suggested FPGA overview with protocol

The red parts are the parts that needed to be done by the protocol created in this thesis. This architecture was slightly modified to keep a generic approach regarding the number of different services supported by the protocol.

Indeed, with this architecture approach, each new added service would change the protocol entity. Similar problems arises when using Qsys for the Avalon bus utilization. Altera Qsys integration tool is not too fond of records for conduits in and conduits out (in and out signals). This approach would thus need more modifications and work for the user when changing services, linking all signals individually.

The architecture was rethought, to include those blocks into the protocol, as show in the below figure.

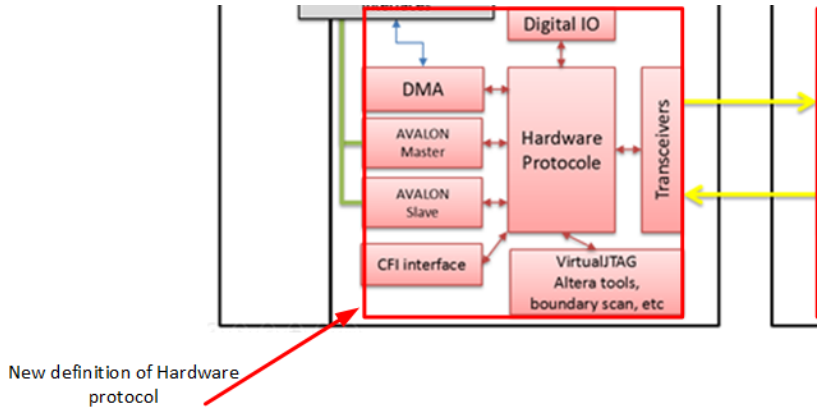


Figure 24: New definition of Hardware protocol limits

Including those services into the protocol facilitate their output interface unification, as well as the entity of the whole “protocol box” itself.

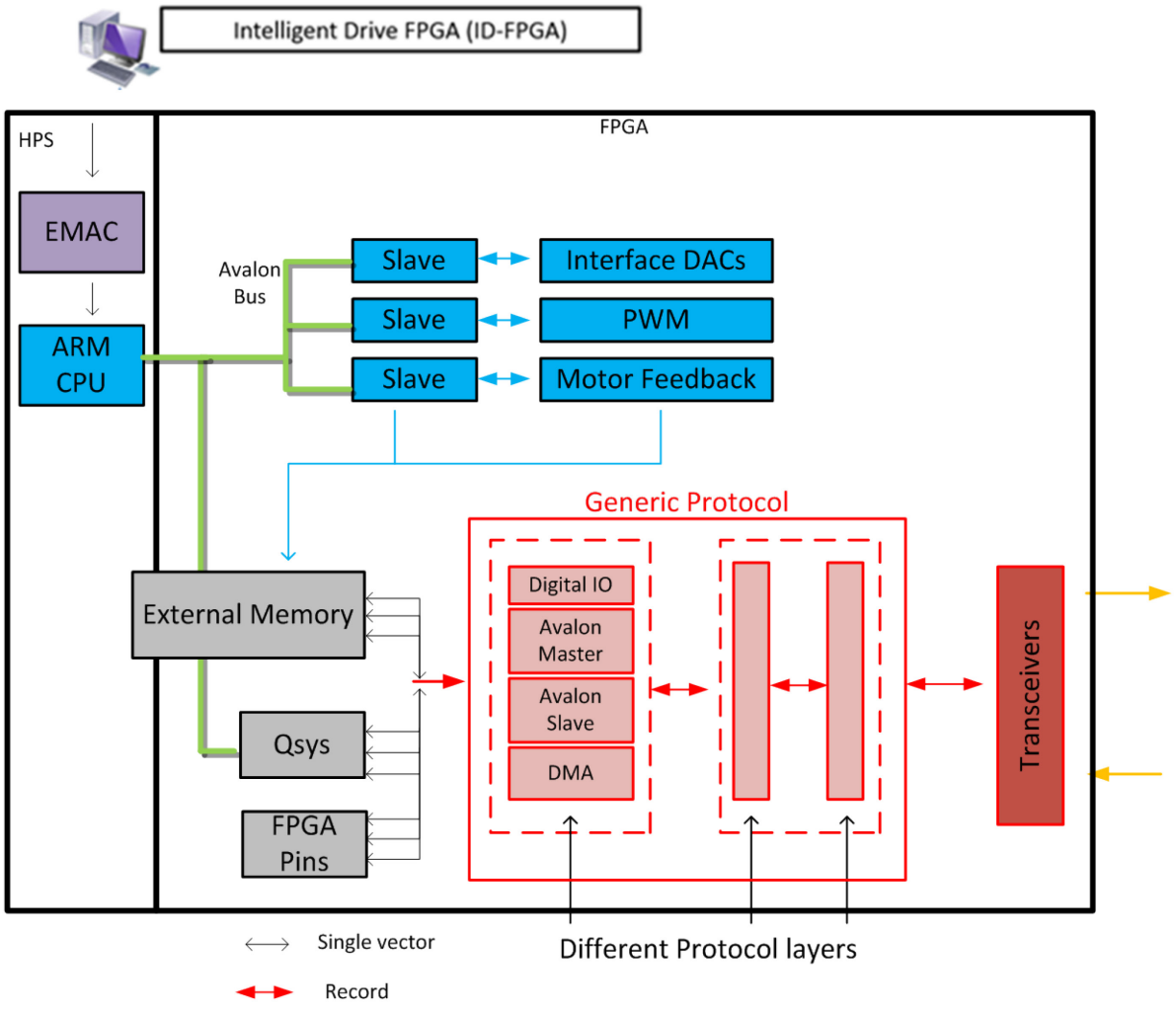


Figure 25: General FPGA Overall Architecture

The figure above shows how the final design was imagined. Only the ID-FPGA was represented, but the AS-FPGA is constructed in a similar mirror way**. It is important to note that the blue parts were not ready during the time of the thesis. The implementation when this paper was written only included the protocol and the FPGA Pins gray box.**

The advantages of such a design is that communication between protocol internal elements and the communication with external components is done through records. Adding, changing or removing a signal from a service connection only requires to change the record definition, and not all the different entities along the way.
The transmitting and reception connections are kept generic inside the protocol.

For the Qsys point of view, it will be like mapping memory on the other FPGA. It only needs an empty element inside to do pass-through connections. The next figures shows how an ARM CPU acting as an Avalon Master can access an Avalon Slave on the other FPGA transparently. The ARM is communicating directly with the generic protocol Avalon Slave service that emulates the Avalon Slave on the FPGA 2.The communication Master to Slave is captured in a way, and reproduced on the other side.

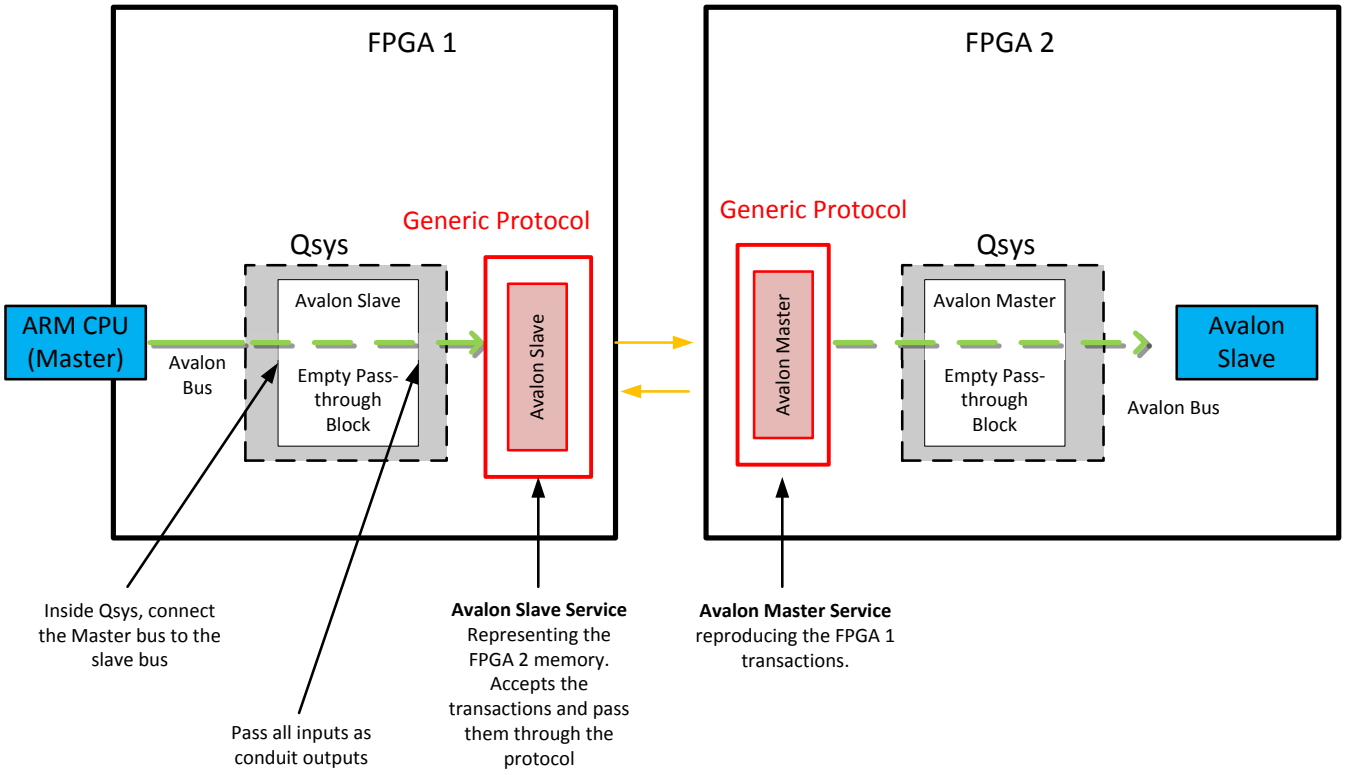


Figure 26: Service communication overview with Qsys

Passing though Qsys is thus an option, and not an obligation. Connections can be made directly outside, without the need of creating conduits. The next figure shows the direct Avalon master to slave connection, without passing through Qsys.

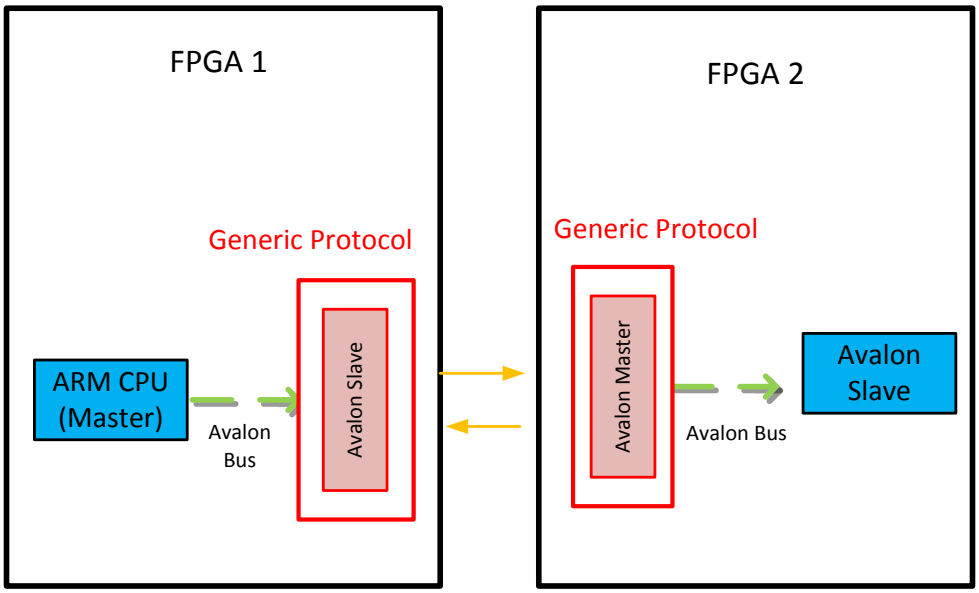


Figure 27: Service communication overview without Qsys

This can greatly simplify the design, when Qsys is not needed.

The following chapters will explain the different protocol elements, or layers, that permits this communication.

5.3. Clocking Architecture

The figure below shows the overall clocking architecture for the FPGA design. The Services can work with the TX clock or their own clocks with a FIFO for the clock domain crossing.

The clocking architecture in most of CERN FPGAs is based to remain synchronized with the accelerators. Since the particles bunches (group of particles) are each separated by 25 ns, 40 MHz clocks are used in most front end systems. The GBT physical layer is one of those system, as it is made to work with the GBT radiation ASIC inside the tunnels.

However, the GBT clocking architecture had to be well analyzed and partially re-thought, to work with an FPGA to FPGA connection, and not ASIC to FPGA.

Physical layer

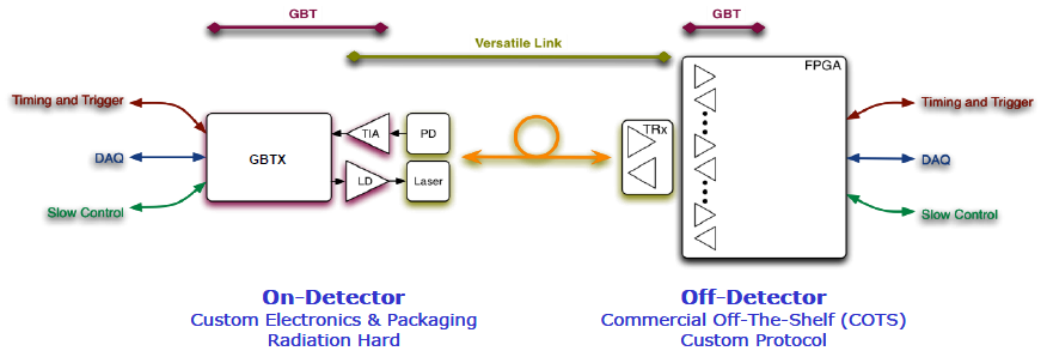


Figure 28: GBTx and GBT-FPGA

The version of the GBT used in this design is the standard version, and not the optimized low latency version, offering low latency and jitter (<1ns) at the cost of complex routing and reset calibrations. For more information about the different GBT version, refer to the GBT User Guide Manuel.

The GBT normal latency clocking architecture can be seen in the figure below.

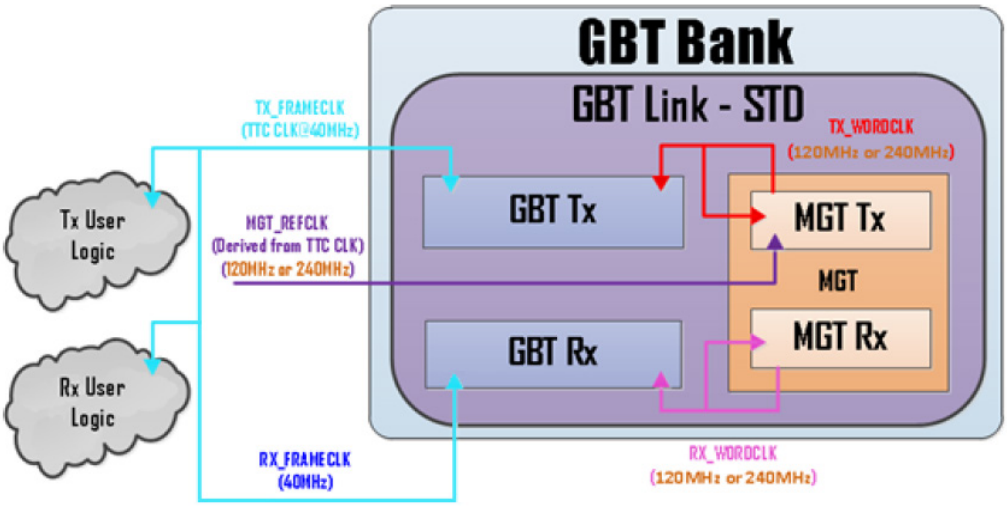


Figure 29: GBT standard version Clocking architecture

An MGT RefClock of 120 or 240 Mhz is required for the transceivers, providing the TX_WorldClk. The GBT link provides a DPRAM buffer used for the time cross domain changes, where data is constantly being written and read.

There are constrains for this architecture to work without errors:

1. The Tx_FrameClk needs to be issued from the same clock as the MGT_Refclk for the DPRAM buffer to work. Otherwise, the write pointer will at some point in time catch up with the read pointer or the other way around.
2. The same can be said for the Rx_FrameClk and the RX_WorldClk.

The first issue can be dealt with a PLL to pass from a 120 Mhz clock to a 40 Mhz clock.

The second issue is dealt by either doing the same thing, or with in the optimized latency version as shown in the figure below, where the RX_FrameClk is constantly re-aligned.

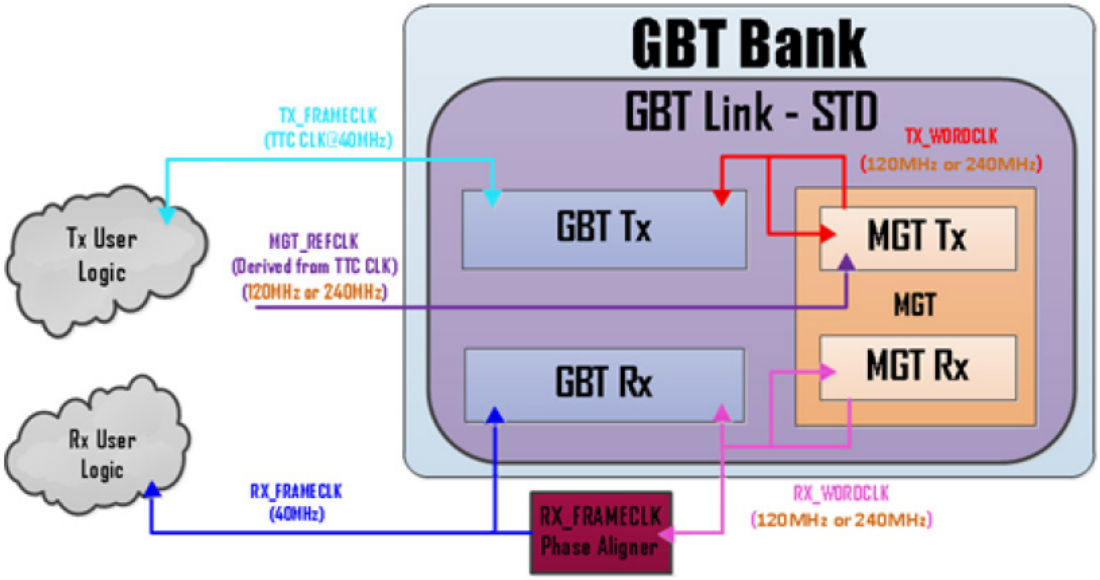


Figure 30 : GBT optimized version Clocking architecture

Doing so, the RX_frameClk is kept in the same time domain as the RX_WorldClk, only at another frequency.

This however creates two clock domains for the user logic (Tx and Rx) which will complicate the protocol connection and not solve the latency issue when signals need to pass from one clock domain to the other.

After investigation, it turns out that the standard GBT version for the RX clocking works, as the GBT ASIC is using its RX recovered clock to send back information. This means that the Rx_WorldClock has the exact same frequency that the TX_WorldClock.

In this thesis case however, since two FPGAs are used, this will not be the case. FPGAs still need to function when the link is not operational, and no Rx Recovery clock can be done. FPGAs will work with their own quartz crystals, ultimately creating some ppm variations between the clocks. Having two different time domains for the Tx and Rx User logic is also not an option and a solution to have a single time domain needed to be found.

A solution was found without modifying the GBT IP core, which would have created an alternate version and would not be compatible with the new core release.

The final GBT clocking architecture can be seen on the figure below, where the PLLs were added externally to the core.

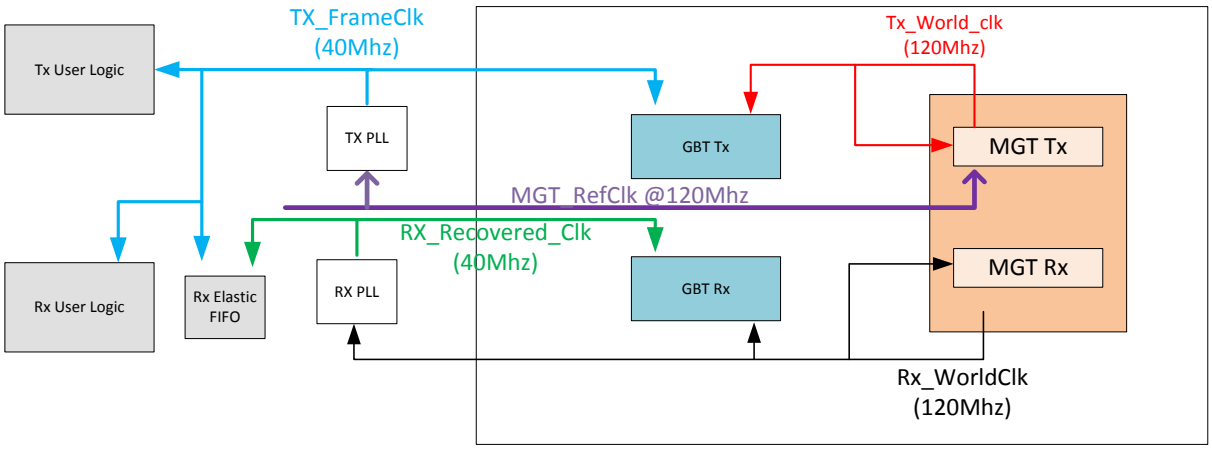


Figure 31 : Implemented GBT clocking Architecture

The time domain crossing is done in an Rx Elastic FIFO controlled by the protocol. Instead of writing every single incoming frames like the GBT DPRAM does, the elastic FIFO only write GBT Data frames and not IDLE frames. If the RX_Recovered Clock is faster than the TX_FrameClock, then sending IDLE frames periodically will prevent a FIFO overflow.

Initially, the TX PLL had the TX_World_Clk as input, but upon generating the Bitstream, Quartus would systematically crash with an unexpected error.

Since the ArriaV SoC dev board doesn't offer multiple clocks on the desired Transceiver Bank, This was the only solution.

MGT RefClock

The Transceiver used is located on the R1 bank. To generate the 120Mhz RefClock needed, the RefClkR3 coming from the Si570 was used.

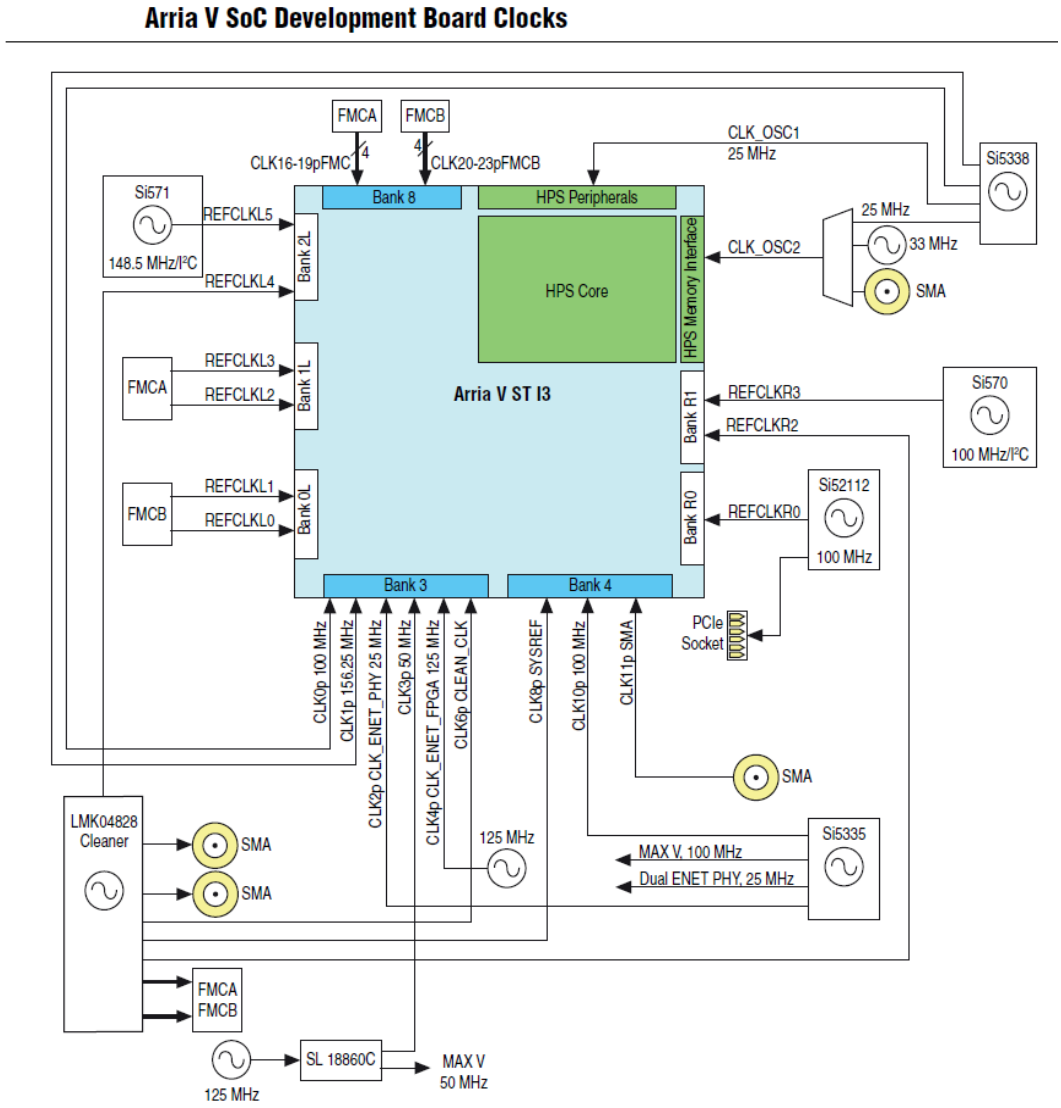


Figure 32 : ArriaV SoC Dev Kit Clocking possibilities Overview

This chip is controlled by the MAXV CPLD, with an initial frequency of 100 Mhz. The MAXV block diagram can be seen on the next figure. Altera provides a clock controller windows application to change the frequency through the USB JTAG.

The frequency changes back to 100 Mhz when the board is switched off. A bitstream of the MAX V needs to redone for this to be permanent. As another board with a similar FPGA will be used in the BWS, this step was not done.

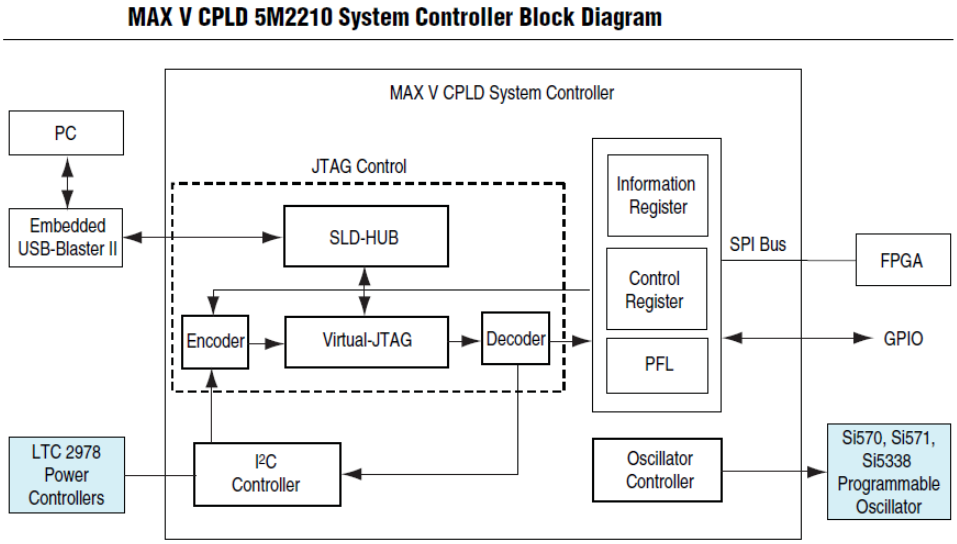


Figure 33: MAXV on dev Kit Architecture

5.4. General Protocol Architecture

The protocol architecture is based on the OSI model. “The Open Systems Interconnection model (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology. Its goal is the interoperability of diverse communication systems with standard protocols. The model partitions a communication system into abstraction layers. The original version of the model defined seven layers” [6].

The following figure shows the 7 different abstraction layers, and their role/application.

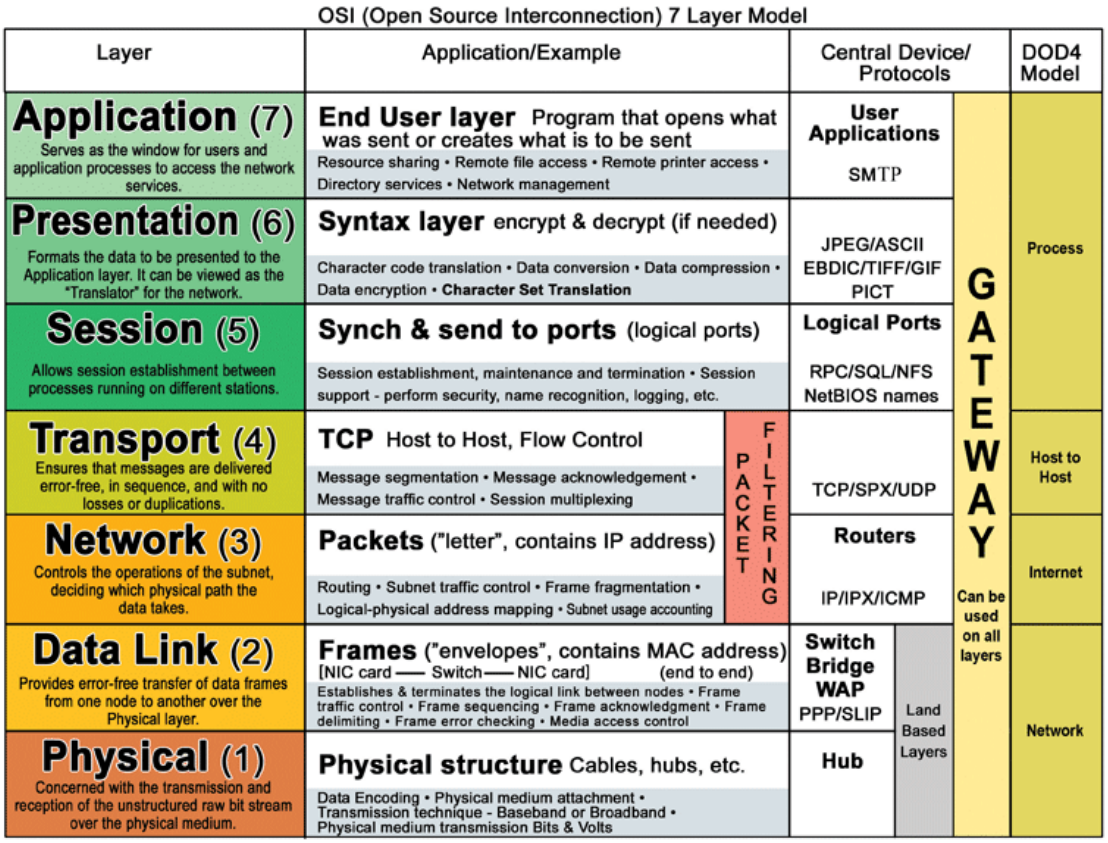


Figure 34 : OSI Model Layer [7]

The decision to base the architecture of this protocol on an OSI model was motivated by different factors:

- To be able to deal with the modularity of the protocol.
- To keep genericity to a high degree
- To facilitate code maintenance
- To deal with different user application integration
- To make it user friendly, keeping details and complexity as low as possible for the user application.

The implemented architecture however does not implement all of the 7 layers presented above. Many are unnecessary in a points to point protocol as opposed to a multi- point to point one. Furthermore, some of their functions can be regrouped together to reduce the number of logic needed for its physical implementation.

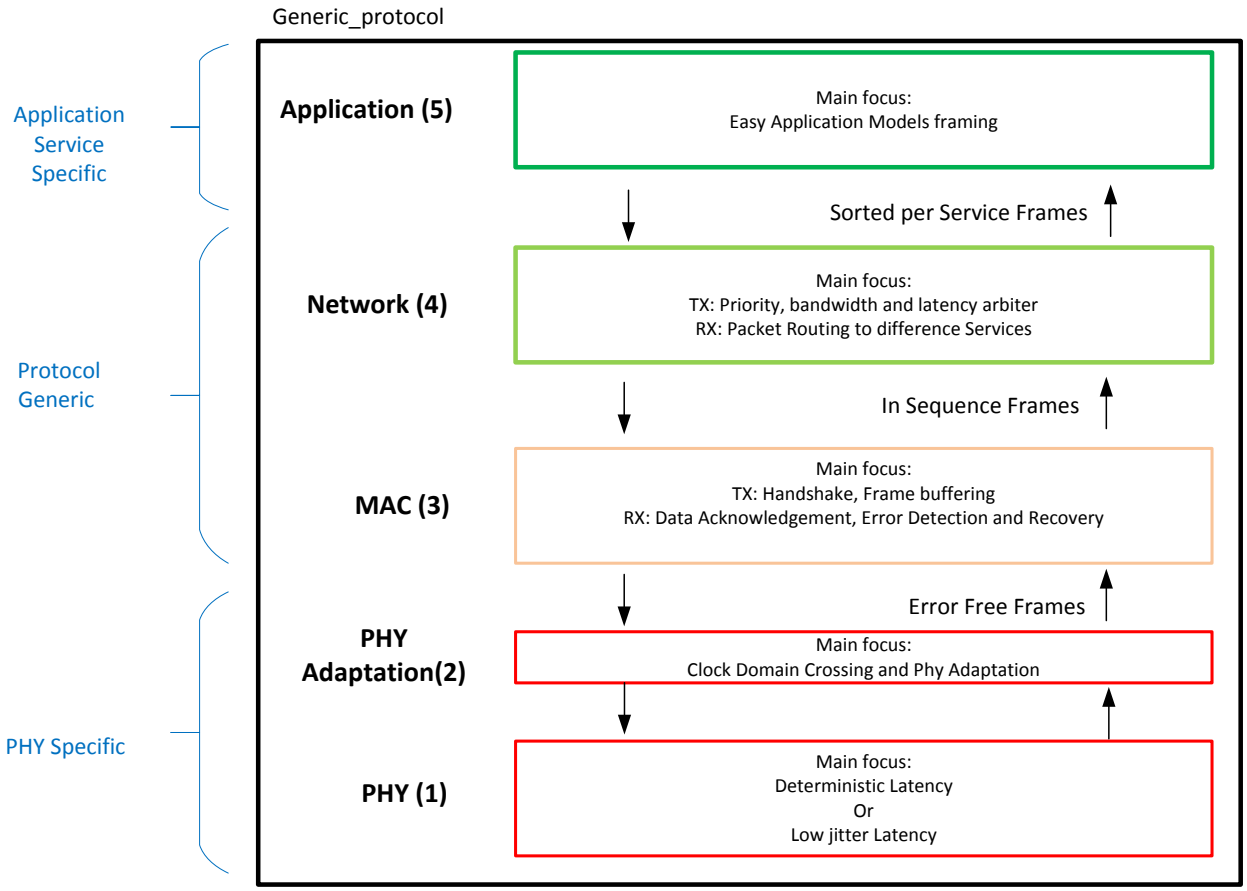


Figure 35 : Selected OSI layers implemented

Only the 4 layers shown in the figure above were used. A 5th one was added to adapt the protocol to any physical layer. Besides the main focuses listed for each different layer, it is important to note how each layer presents information to the next when dealing with an Rx Transmission.

- The physical layer (adaptation) provides error free frames to the Mac layer
- The mac layer provides frames in the right sequence to the Network layer
- The network layer provides each application service their own frames

Initially, the OSI transport layer's job to ensure that data are delivered in sequence was to be done in the Network layer. The sequence order was initially thought to be done per Service, where a transmission error on a frame would only impact the service in question, and not the others. That turned out to be very demanding on logic and a lot more complex to do, especially in the FIFOs buffering demand.

In the end, the sequence integrity was done in the MAC layer (Data Link).

Going further into details, the next figure shows the main building blocks of each layers.

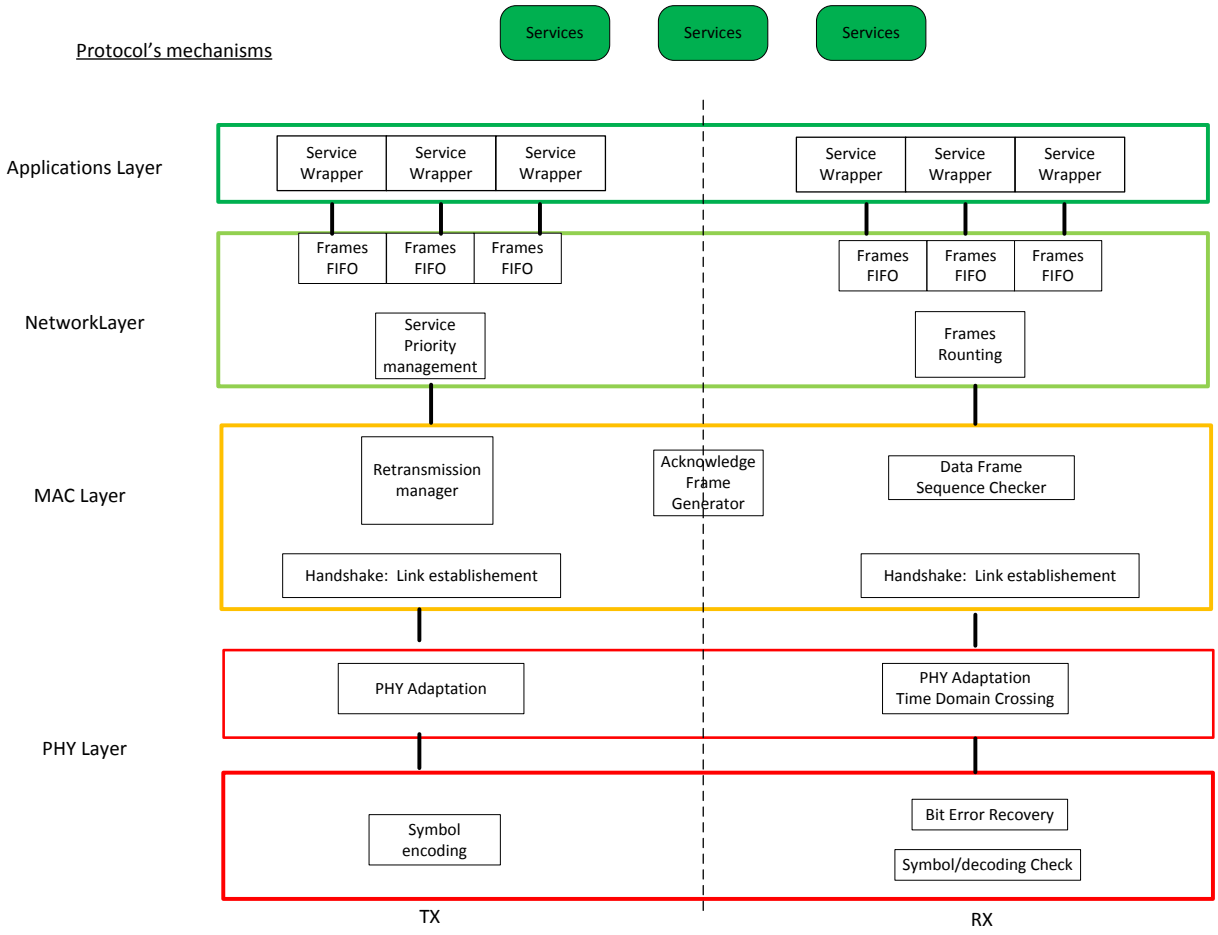


Figure 36: Selected OSI layers implemented in details

Rather than explaining each layer one by one in details, this paper will focus more on listing the main mechanisms inside each layer that truly make up this protocol, and explaining each mechanism individually.

Physical layer

The physical layer has 2 parts. An adaptation layer is used to easily change how the data is presented to the physical layer, in case this layer needs to be changed. This ensures the modularity of the protocol in regards to the physical layer. This is also where the time domain crossing is done, passing from the RX world clock to the TX clock. More information on the time domain crossing in the section *Clocking Architecture*.

The Physical layer used in this protocol is the GBT physical layer. It is a custom physical layer developed and maintained by CERN and was not part of this thesis's creation. The GBT internal structure and mechanism can be read in the Analysis part of this paper as well as in the Clocking Architecture part.

Mac layer

The mac layer manages the link establishment, the frame sequence order and validation. Therefore the two main mechanisms are as followed:

- Protocol handshake, link establishment
- An acknowledge and retransmission manager
    - A buffer that keeps the transmitted frames until they are validated
    - An acknowledge frame generator, that returns the reception frames status.

Network Layer

The network layer's primary role to manage multiple input sources going into a single output for the transmitting side, and a single input going to multiple outputs on the receiving side.

Therefore, the main mechanisms are:

- Buffering management.
- Priority arbiter that can manage a fair distribution of the common resource to all of its requestors.

Application layer

The application layer needs to be able to handle different user services requests for transactions, either wanting to send data or wanting to retrieve (read) some, and convert them into compatible protocol data frames. The main points are:

- Generic coding, with minimum required code change when connecting a different service
- Easy integration or retrieval of information to or from the protocol frames format.

5.5. Communication between layers

Each layer has its own communication signals to communicate with the layer above or below. Those signals are regrouped into records. For the rest of the document, those records will be called layer frames.

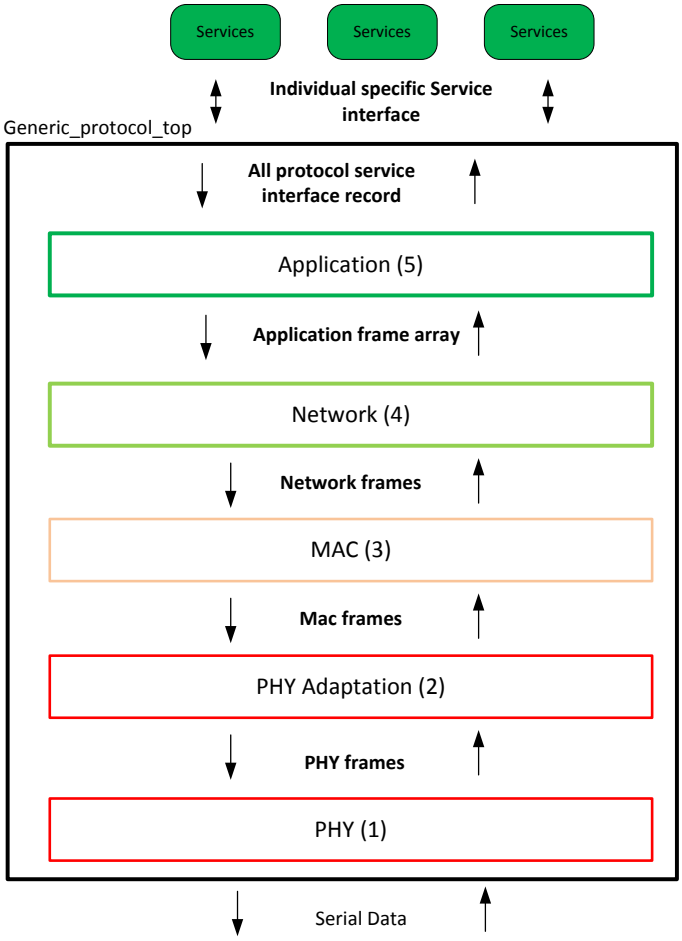


Figure 37: Layers communication Overview

Each Service has its own individual vhdl record interface such as an Avalon Memory Mapped Master or Slave interface, an Avalon streaming interface or a Wishbone interface. It can also simply be signals such as an ADC data bus with the optional data enable bits. Those individual records are put at the top of the protocol in another record that will regroup all services interfaces.

The following figure illustrates an example with 2 services and a potential 3rd one.

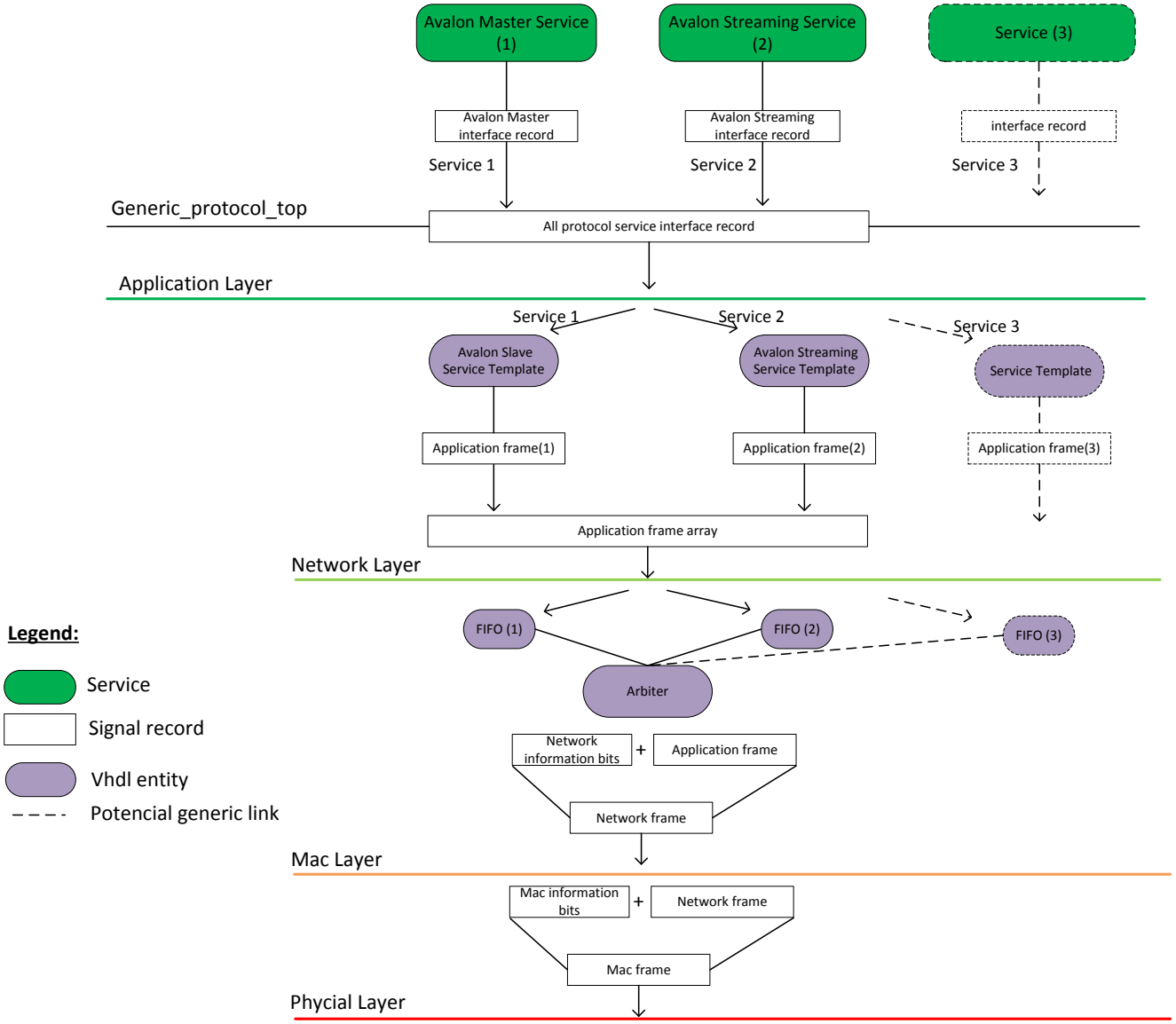


Figure 39: TX Layer communication

Each service template, inside the application layer, will convert the useful information given by the service interfaces into Application frames. The output of the application layer will be an array of those application frames.

The network layer takes the application frames, add its own layer information and creates network frames.

The same way, the mac layer, takes the network frames, add the information related to its layer and creates mac frames.

For the layers dealing with multiple frames derived from the number of services implemented (Top, Application and network layers), it is essential to merge them when entering a new layer and split them again inside to keep a generic code. Thus, the layer's entities don't need to be changed when adding, removing or changing a service.

The logic generated and information pathway are done using the number of implemented services' constant in the protocol package.

The Rx pathway behaves in a similar matter, but instead of adding data for the next layer, each layer remove its own information before passing the rest to the layer above. The next figure represents the basic flow of information when receiving information.

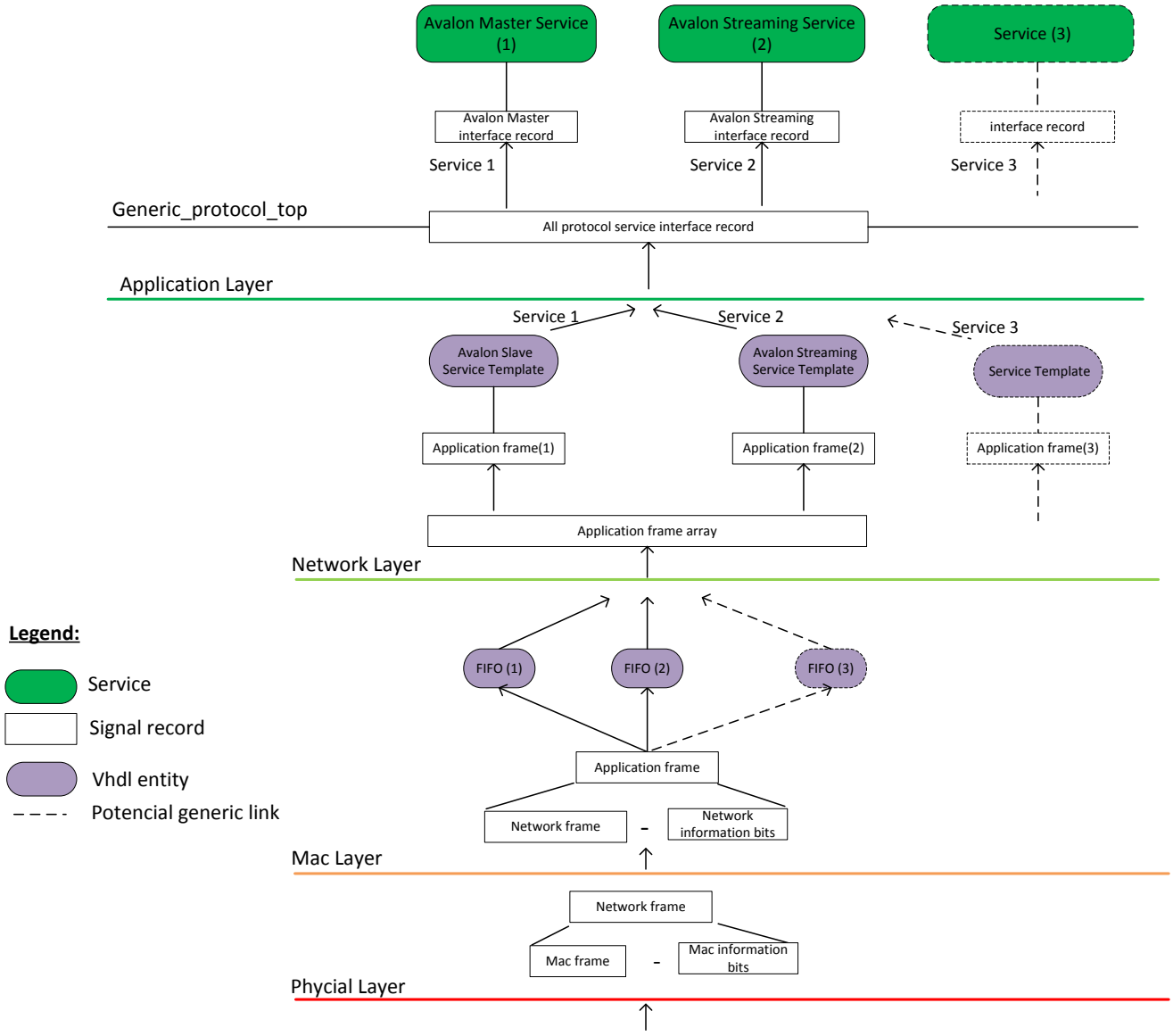


Figure 40: Rx layer communication

This is however a simplified version of the Mac layer, only representing the Services data communication flow. The mac layer, having to deal with the protocol link control, has other frames types that need transferring.

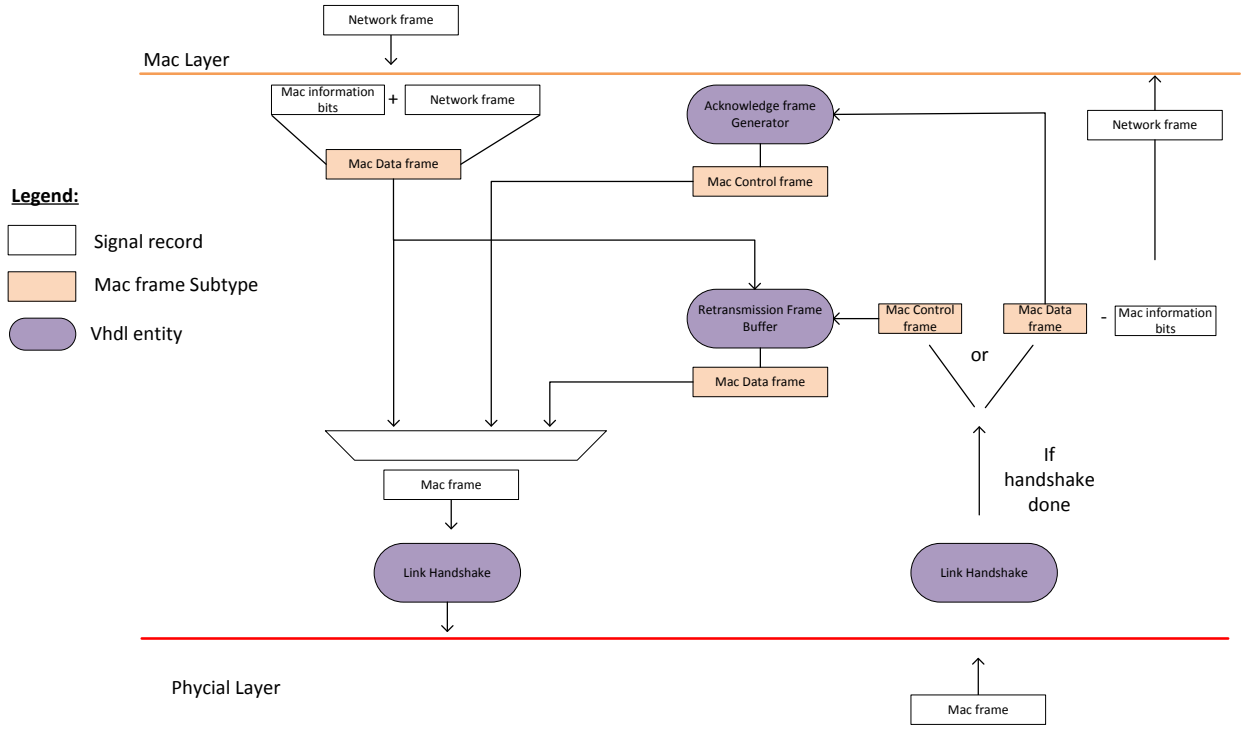


Figure 41: Mac layer internal communication

There are two Mac frame subtype defined by the mac bit *MAC_IsCtlFrame*:

- Mac Data frames, containing the network frame.
- Mac Control frames containing link control information, such as Link handshake or acknowledge control frames.

The Mac control frames are used to establish the link handshake at the start of a communication, and to send back information regarding the status of the received data frames (valid or not valid). Therefore, when receiving those control frames, the retransmission frame buffer either delete the valid frame or resend the data frame in question.

It's important to note that Mac Control Frames do not leave the Mac layer. Only the Mac Data frames, if valid, are transmitted further to the Network layer.

For more information about the retransmission mechanism, refer to the related section of this paper.

5.6. Frames bits ordering

The starting point for the bit ordering is the physical layer frames, the GBT, as each frame contains the Forward Error Correction, validating each 80 bits of user data and the Slow Control 4 bits.

That said, the protocol is made to be generic, and field sizes can be easily changed, added or removed, constants being regrouped in a single package file.

The next figure shows the complete GBT frame breakdown and the bandwidth associated with it.

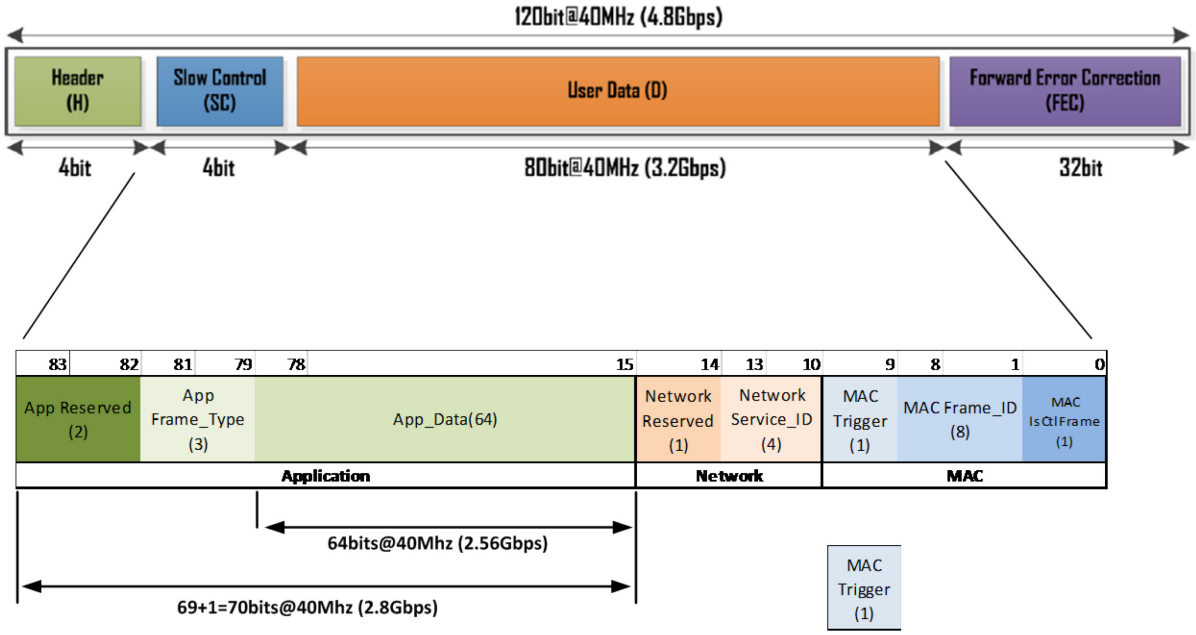


Figure 38: Total frame bit ordering into a GBT frame

Short description of each field:

Field Name	Number of bits	Function
Mac_is_Ctl_Frame	1	Defines the frame as Data frame or Control frame.
Mac_Frame_ID	8	Frame ID counter that validates the frame's sequence order
MAC_Trigger	1	Bit used for trigger
Network_Service_ID	4	Define the Service ID, unique for each running service
Network_reserved	1	Unused reserved bit
App_Data	64	Pure data field that can be used in the application layer
App_Frame_Type	3	Field to differentiate between different application frames within the same service. Useful when more than 64 bits of information is needed "per service transaction". Can also be used as pure data.
App_Reserved	2	Unused bits that can potentially be used for any layer

Tableau 3

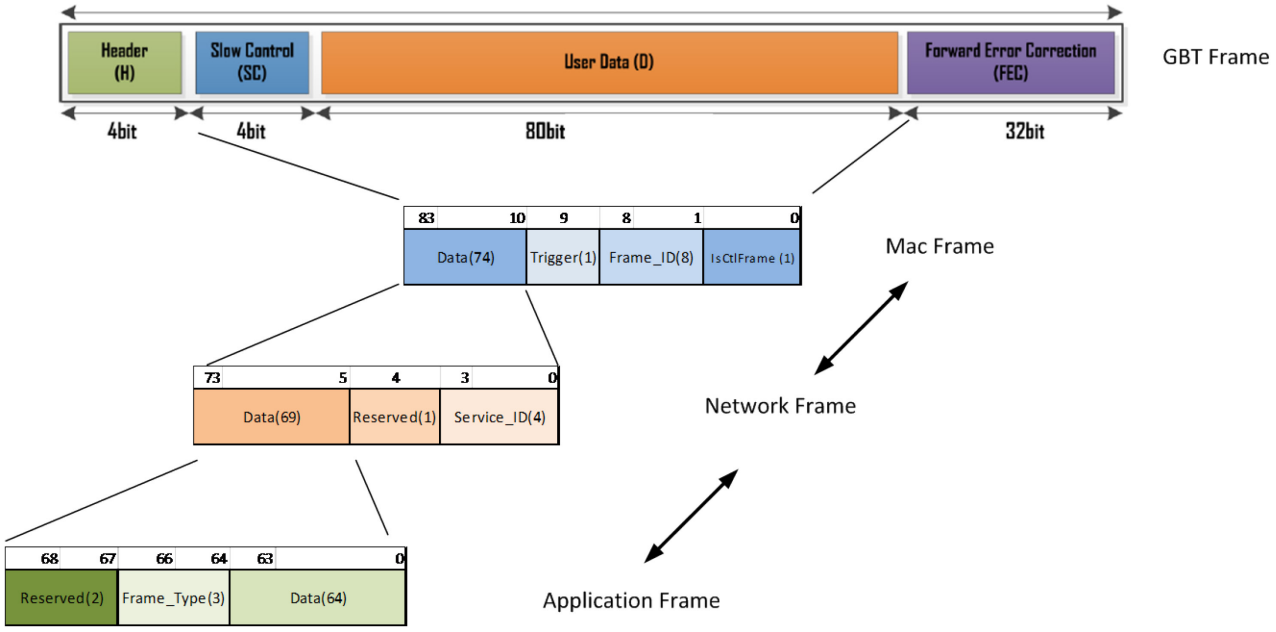


Figure 39: Frame record encapsulation overview

The Mac Frame contains the Network frame that contains the application frame. Even if each frame is a record, the application frame is seen as a bus vector by the Network frame, and the network frame is equally seen as a vector for the Mac frame.

Passing from a format frame to another is possible with the use of functions that pack record fields into a vector and unpack or split a vector back to record fields. Those functions are purely a way to see information bits in a different and more understanding way. It also limits the mistakes of the packing and unpacking by hand.

It's also a way to keep the layers independent, treating the information coming from the layer above as pure data. Layers don't have dependencies between them, keeping the architecture modular.

The figure below shows the function that pack the Application frame record into a vector.

```
function APP_Frame_To_SLV(rec : Protocol_Format_APPLICATION_Frame)   return std_logic_vector is
        variable slv :  std_logic_vector(APPLICATION_FRAME_SIZE+APPLICATION_Reserved_Size-1 downto 0);
        variable idx : integer;
    begin
            slv := (others => 'U');
            idx := 0;
            pack(slv, idx, rec.APPLICATION_Data);
            pack(slv, idx, rec.APPLICATION_FRAME_TYPE);
            pack(slv, idx, rec.APPLICATION_Reserved);

            return slv;
    end function APP_Frame_To_SLV;
```

Figure 40: Record to Std_logic_vector function

The opposite function to unpack can be seen in the figure below:

```
function SLV_TO_APP_FRAME (slv : std_logic_vector(APPLICATION_FRAME_SIZE+APPLICATION_Reserved_Size-1 downto 0))
return  Protocol_Format_APPLICATION_Frame is
        variable rec : Protocol_Format_APPLICATION_Frame;
        variable idx : integer;
    begin
            idx := 0;
            unpack(slv, idx, rec.APPLICATION_Data);
            unpack(slv, idx, rec.APPLICATION_FRAME_TYPE);
            unpack(slv, idx, rec.APPLICATION_Reserved);

            return rec;
    end function SLV_TO_APP_FRAME;
```

Figure 41: Std_logic_vector to Record function

The functions need of course to keep the packing and unpacking of the record fields in the right order. The vector's length is also defined in the package and is adapted when the records are changed.

Acknowledge frames bit ordering

Since this mechanism is located in the MAC Layer, the acknowledge frames are using the Mac_Frame_Data field to transmit their information.

This field normally contains the Network Data frame when the frame is tagged as a data frame (*IsCtlFrame=0*) as shown in the picture below:

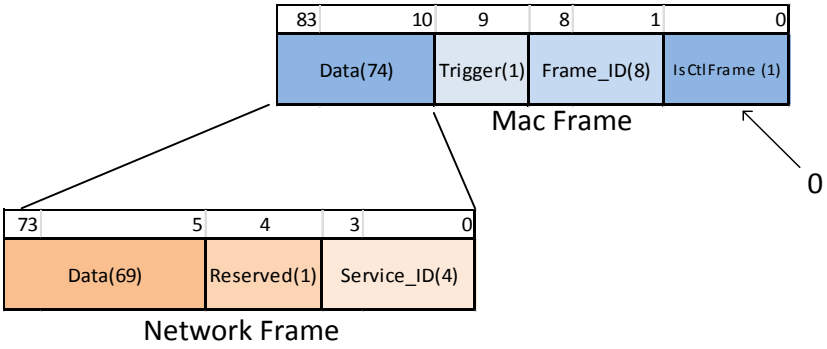


Figure 42: Mac Data frame interpretation

When the Mac bit IsCtlFrame=1 then the data is interpreted has followed:

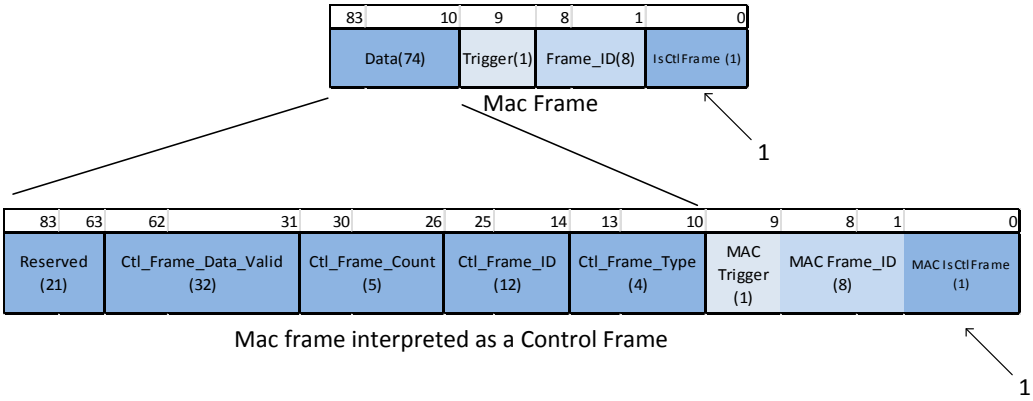


Figure 43: Mac Control frame interpretation

This interpretation, like previously written in the presentation of the protocol, is done by functions that pack and unpack a record type into a std_logic_vector or viscera. Those functions and records can be found in *protocol_pkg.vhd*

```
-- Convert MAC Ack frames
function MAC_Ack_Frame_To_SLV(rec : MAC_Data_Ctl_Ack_Frame)  return std_logic_vector ;
function STV_TO_MAC_Ack_FRAME (slv : std_logic_vector(NETWORK_DATA_SIZE-1 downto 0))  return  MAC_Data_Ctl_Ack_Frame;
```

Figure 44: Mac interpretation functions

```
type MAC_Data_Ctl_Ack_Frame  is record
    Ctl_Frame_Type          : std_logic_vector(MAC_DATA_CTL_TYPE_SIZE-1 downto 0);
    Ctl_Frame_ID            : std_logic_vector(MAC_Frame_ID_Size+NETWORK_Service_ID_Size-1 downto 0);
    Ctl_Frame_Count         : std_logic_vector(MAC_DATA_CTL_ACK_FRAME_COUNT_SIZE -1 downto 0);
    Ctl_Frame_Data_Valid    : std_logic_vector(MAC_DATA_CTL_ACK_FRAME_VALID_SIZE-1 downto 0);
    Ctl_Frame_reserved      : std_logic_vector(MAC_DATA_CTL_ACK_FRAME_RESERVED_SIZE-1 downto 0);
end record;
```

Figure 45: Mac control Ack frame Record

Field Name	Number of bits	Function
Ctl_Frame_Type	4	Refers to a constant, defining the type of Ctl Frame (Handshake, Ack etc)
Ctl_Frame_ID	12	Refers to the first MAC_Data_ID and Network ID that the frame is validating. This ensures the detection of multiple wrong frames missing
Ctl_Frame_Count	5	refers to how many frames are validated (good or bad) in the Frame
Ctl_Frame_Data_Valid	32	Each bit represent the validation of a frame starting from the LSB until *Ctl_Frame_Count.*
Ctl_Frame_Reserved	21	unused bit for this type of frame

Tableau 4

5.7. Protocol Handshake

Each protocol needs a link establishment mechanism, called a Handshake. The mechanism is located in the Mac layer.

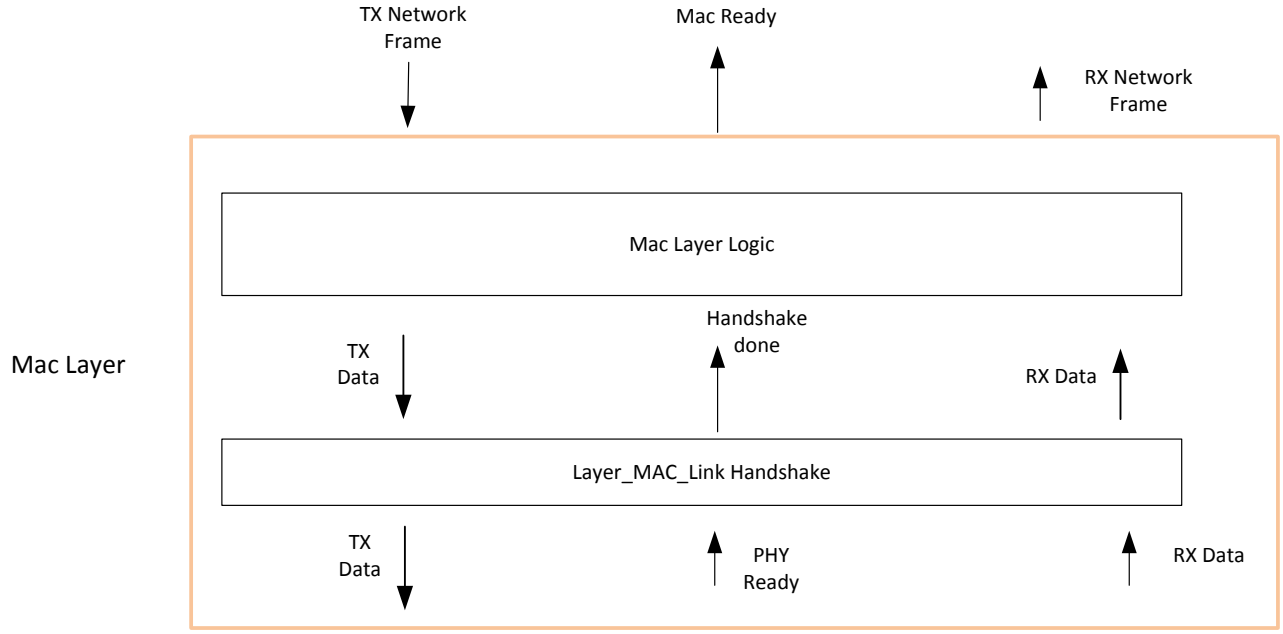


Figure 46: Mac Layer Handshake

Any transmission that goes out or in the mac goes through it. If the handshake isn't done yet, the mechanism prevents any network TX frames to be transmitted, and any RX frame received is not passing through either. This mechanism is implemented using a state machine, the only one used in this protocol.

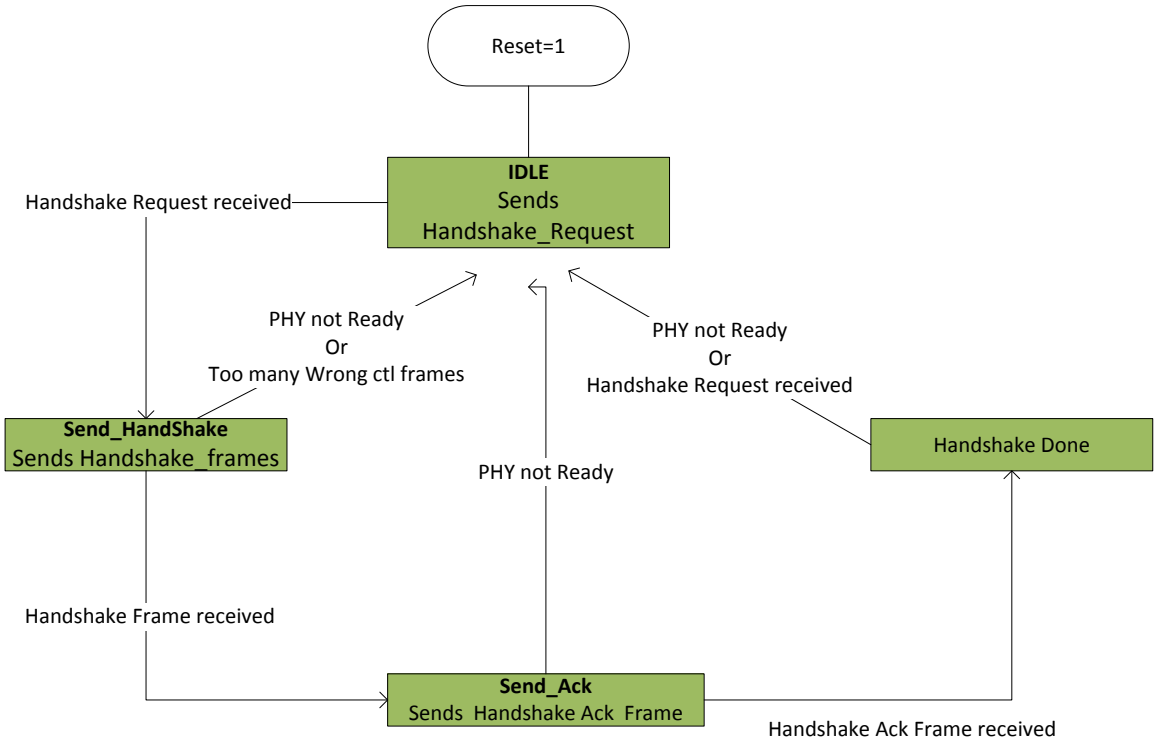


Figure 47: Mac Layer Handshake Statemachine

The handshake waits for the PHY to be ready, then initiate the link communication. Once established, it will signal the mac with a handshake done flag. The mac will then inform the Network Layer that it is ready and will permit Transmissions to go through.

A handshake is usually between a Master and a Slave, and implements two different behaviors. In this protocol, it was decided not to go with this option, as it complicates testing, going from loopback mode to two boards testing without reconfiguration.

Both sides have the same state machine, and expect the same frames to be received. Physical tests showed that the physical layer GBT sometimes unlocks itself and relocks itself on power up, resetting the state machine in the handshake. This is however not seen on the other side, which led to blocking states. To counter that, a watch dog counter reinitiate the state machine to IDLE if such event is detected.

5.8. Bandwidth & Priority Management

Understanding the implications

Bandwidth and priority management are important aspects with any protocol dealing with multiple hosts. In this thesis, 1 up to 16 different services have the possibility to send information through the protocol.

A first issue regarding bandwidth, as the number of services increases, is that two or more services will want to send information at the same time. However, only one frame can be send at the time. Since the idea is to have complete independent services, one cannot block the other on the application level. This is addressed using FIFOs to buffer incoming frames that cannot be sent immediately.

This enable the protocol to remove the bandwidth spike demands that briefly exceed 100% of the bandwidth at disposal, and smooth it over time. The spike length absorption time depends on the depth of the FIFOs and how many services are involved.

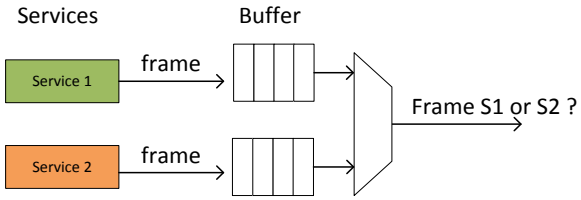


Figure 48: Priority multiplexor issue

The second issue is the priority management between different services, which relates to the bandwidth allocation.

After frames have been buffered, they need to be transmitted one by one.

An easy approach would be to code a fixed priority system, with multiplexor, where services are classified in priority order.

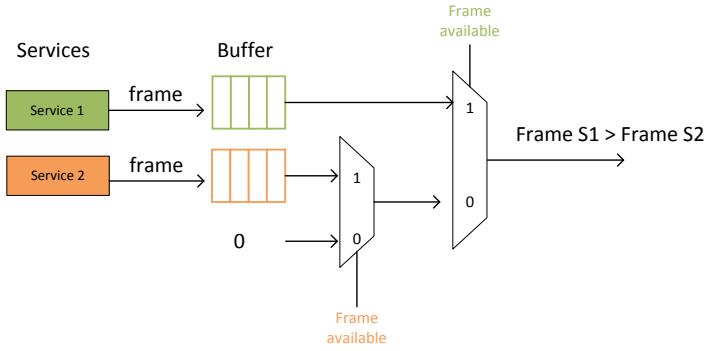


Figure 49: Fixed Priority multiplexor

That priority system is easy to code, however it is blocking, meaning that as long as services with high priority have data to send, the others have to wait and may never be able to send data. A minimum bandwidth allocation for a service is therefore unpredictable.

A third and last issue is the generic aspect of the code. Adding or removing a service should not have any implications in the priority management, the code itself, or the complexity of the logic created.

Coding a state machine with different cases is therefore not adapted for such a priority management.

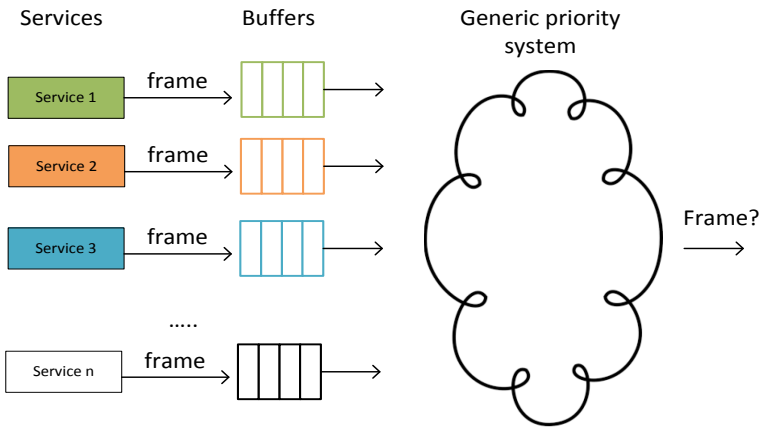


Figure 50: Priority system to conceive

Implemented architecture

The implemented priority and bandwidth management architecture of this thesis was made to deal with all those issues, while keeping the code as simple as possible.

The architecture is done with a generic weighted Round-robin scheduling arbiter.

Round robin arbitration is a scheduling scheme which gives to each requestor its share of using a common resource for a limited time or data elements. The basic algorithm indicates that once a requestor has been serves he would “go around” to the end of the line and be the last to be served again.

The simplest form of round robin arbiter is based on assignment of a fixed time slot per requestor; this can be implemented using a cyclic counter. Alternatively, the arbiter can be defined to allow a specific number X of data elements per each requestor, in which case X data elements from each requestor would be processed before moving to the next one. This type of round robin arbiter is referred to as a weighted arbiter where each requestor is assigned a predefined weight [8].

It is important to start with a clear definition of our needs for this arbiter. The design should incorporate the elements below:

- Number of requestors has to be generic. This is needed to connect as many requestors, aka services, as we want in the application layer, and not touch the protocol’s code.

- One cycle calculation, so the arbiter can grant a different requestor in each cycle. This is essential as we deal with a different frame every clock cycle.
- Wraparound functionality, meaning that the arbiter does not loose cycles at the end of each round when moving from a grant to the last active requestor, back to the first one. This is an important aspect not to lose bandwidth.
- Work conserving functionality, so no cycles are lost on services that are inactive. This is another important requirement for the bandwidth, as services will not output data all the time.
- Grants for requests based on weighted distribution. This is to ensure that bandwidth allocation will be correctly done when dealing with high burst demanding services.

A weighted Round-robin system gives “strong” fairness, as all services will get served (none blocking) and the grants given to each service will be defined by its predefine weight.

With this weight, the user can allocate a minimum bandwidth utilization specific for each service. This system ensures services with high burst bandwidth needs to transfer their data rapidly in time (in order not to have FIFOs overflow), without blocking the other services completely, by decreasing their bandwidth to their respective minimum values.

The starting point for the arbiter was a generic round-robin arbiter without the weight management. The file is accessible on the git [9] and allowed to be changed.

This was a very good starting point, as almost all of the elements listed above were present. The missing part of the incorporation of the weight distribution.

In this classic architecture, each service has a weight priority of 1 clock cycle.

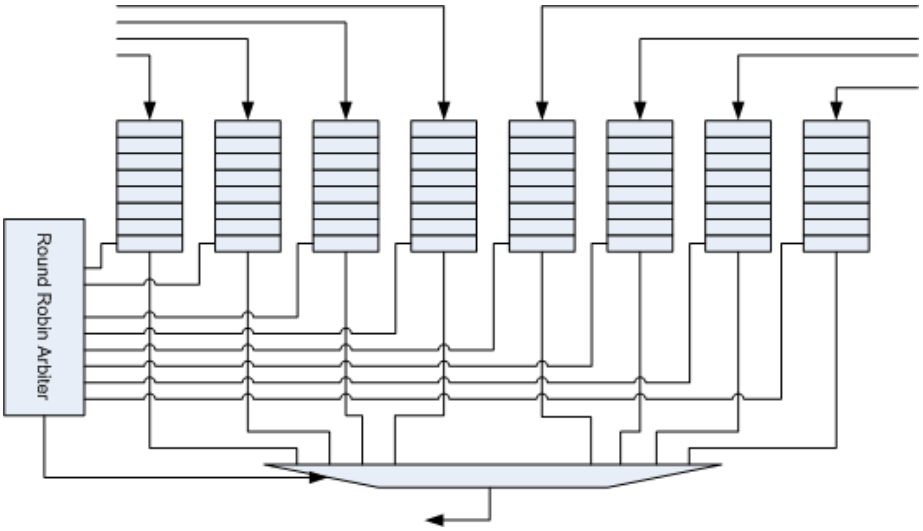


Figure 51: Round Robin Arbiter

The arbiter will cycle through each service one by one, checking for available frames to send. If one or multiple services have no data, without losing clock cycles, the arbiter will skip them until it can found the next data.

For example, with a five services system, if only one service has data, then that service will have 100% bandwidth at its disposal. If two of those five services had data in their FIFOs, then they would each have 50% of the bandwidth.

In the case where all services have data ready, then the minimum guarantee bandwidth for each service would be:

$\frac{1}{\text{NB\_service}} = \frac{1}{5}$ or 20%.

.

So to summarized, with such a system, each service has a minimum bandwidth of $\frac{1}{\text{NB\_service}}$ and a maximum bandwidth of 100%.

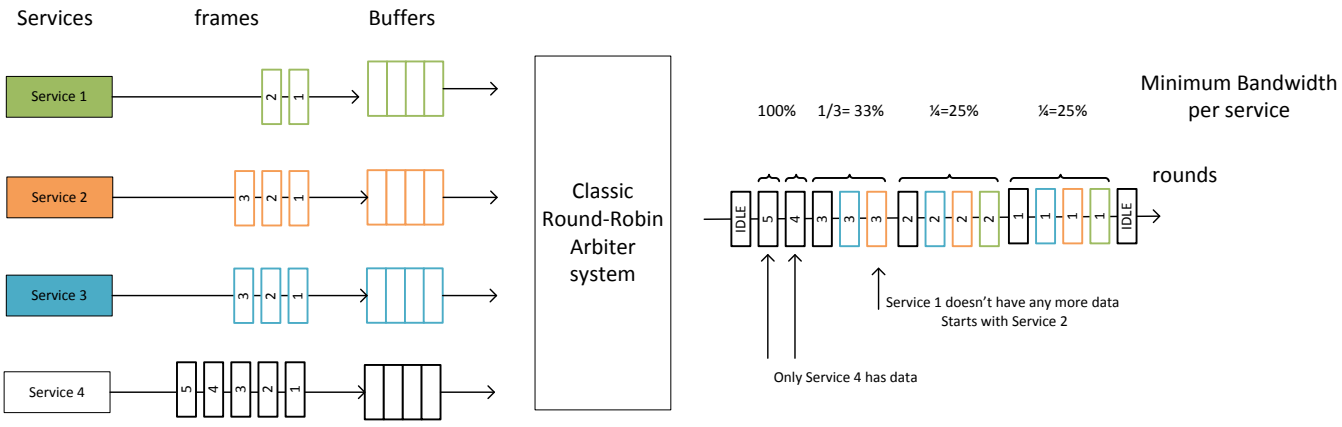


Figure 52: Round Robin frame priority management

This arbiter was then modified to be able to specify priority weights to allow minimum bandwidth allocation for each service. The weights are the number of timeslots per round cycle (the cycle through all services) that a service may send data.

An important points to mention is that some weighted Round-robin arbiters periodically activate each service's grant for their allocated bandwidth slots disregarding if data is being send or not. This results in an overall bandwidth loss if the served service has no data to send (and others still do) like previously mentioned.

The implemented arbiter of this thesis ensures that bandwidth is never lost, and priority is established by the weight constants of the services currently requesting a grant (thus active).

The weights are entered in the service package vhdl file.

```
Constant Services_priority_weight : Services_priority_weight_array:=(
            --0 => 0, -- NOT USED but needed for compilation
            1   => 1,
            3   => 2,
            4   => 1,
        others  => 1
);
```

Figure 53: Service priority weights constant

In the above weight example, service 3 has 2 timeslots at disposal while the others have 1.

If we retake the five services system example, now the total weight changed from 5 to 6. So services 1, 2, 4 and 5 will have a minimum bandwidth of $\frac{1}{6} = 16.6\%$ while service 3 will have a minimum bandwidth of $\frac{2}{6} = 33.3\%$. This is of course if all services have data in their FIFOs ready to be send.

If only one service has data ready, then the situation hasn't changed, and that service still has 100% bandwidth.

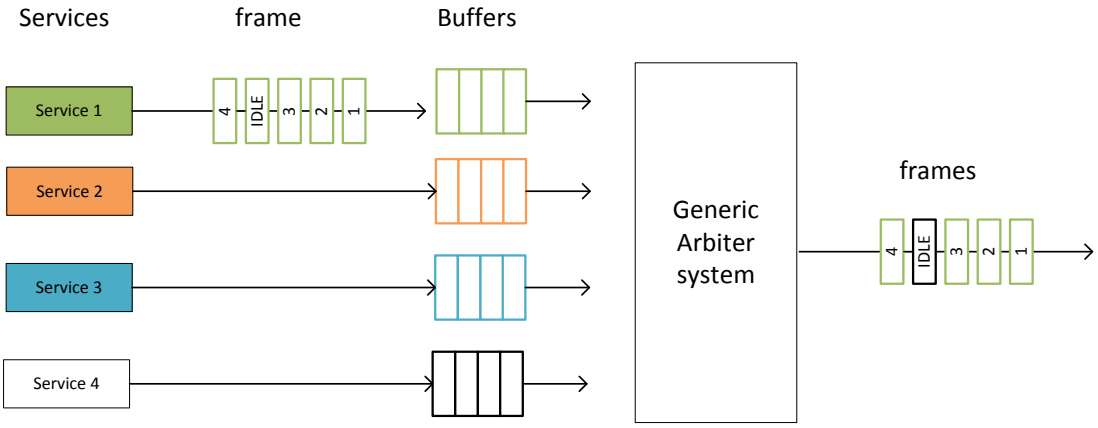


Figure 54: Arbiter with a single service

If only service 1 and 3 have data to send however, then instead of the 50/50% bandwidth allocation like before, now service 1 will have a minimum bandwidth of $\frac{1}{1+2} = 33.3\%$ and service 3 will have a minimum bandwidth of $\frac{2}{1+2} = 66.6\%$.

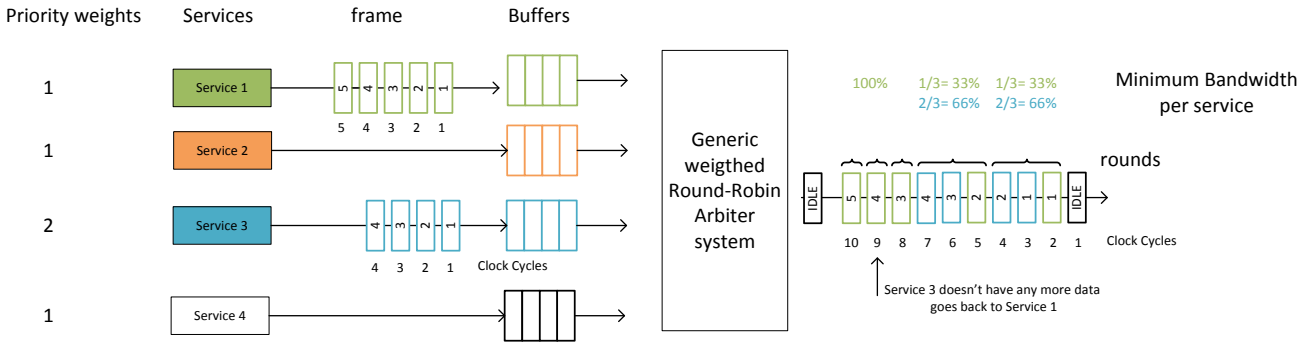


Figure 55: weighted Arbiter with 2 services example

Those values are true as long as service 3 has data ready. If service 3 doesn't have 2 frames ready to be send in its FIFO, then the arbiter will automatically switch to the service that has data ready.

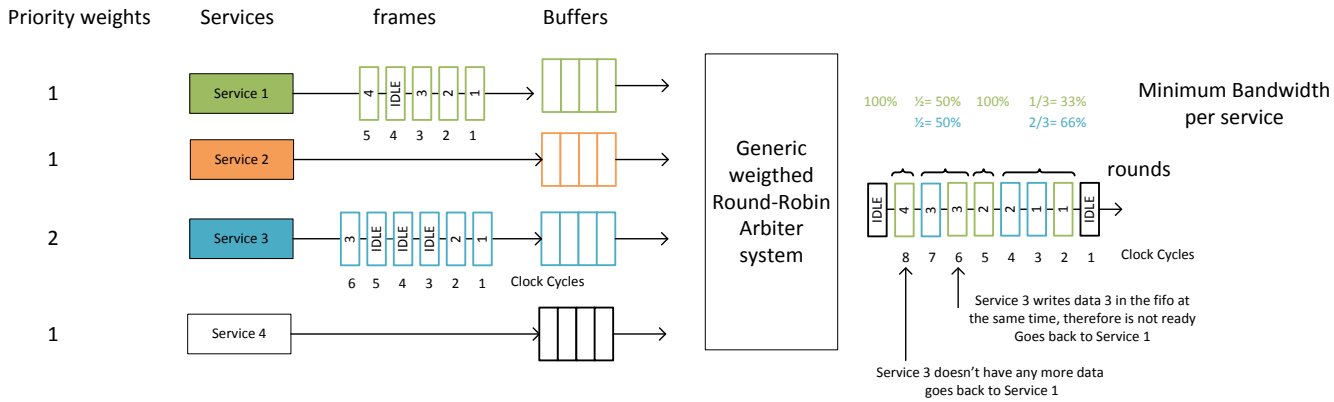


Figure 56: weighted Arbiter with 2 services example (2)

This architecture is completely generic in regard to the number of services supported, and the weight constant in the vhdl package. The end user can add or remove services, without any modifications needed in the arbiter.

Difficulty encountered

The major difficulty that had to be resolved in this arbiter was the corner case where a service had multiple slots at its disposal but less data available. With the signal *fifo_empty* from its respective FIFO coming 1 clock cycle after the actual read, the arbiter kept serving that service for one more clock cycle, even if it was empty. This created empty timeslots where no data was send, greatly reducing the total bandwidth and not meeting the requirements.

The use of the *Rd_fifo_words,* (to know the number of data in the FIFO) on its own wasn't enough as data could be written at the same time that it was read, potentially providing a constant flow of incoming data. An *almost_empty* signal had to be created, taking into account various cases.

The end result assures that no timeslots are wasted between services switching in the arbiter.

Examples of bandwidth management can be found in the Simulation chapter.

5.9. Error Management

The physical layer (GBT) already has an error recovery element, by its Forward Error Correction (FEC), a 32bits Reed–Solomon code error correction for every GBT frame.

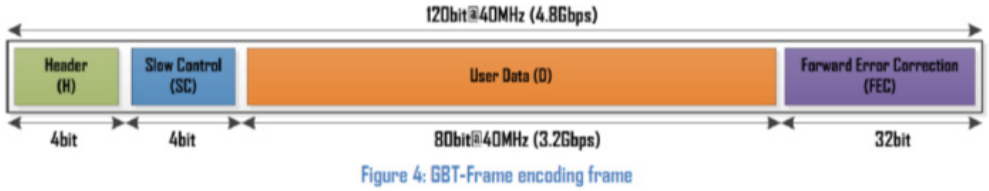


Figure 4: GBT-Frame encoding frame

Figure 57 : GBT frame encoding frame

Because of this, the protocol does not incorporate any other frames integrity check like cyclic redundancy check (CRC).

However, a lot of effort was put into a protocol integrity check, in other words, a data flow integrity check with a retransmission mechanism which is described in the next section.

5.10. Retransmission

When sending streaming data over any protocol or medium, data ordering may or may not be critical. When streaming music or video for example, a missing frame can have a visual impact, but that effect is usually only temporary and doesn't affect the rest of the streaming data. An image was missing or part of an image was missing, but the video goes on.

However, only checking for frames integrity isn't enough when data ordering needs to be verified and ensured. When a memory content needs to be transferred somewhere else, a missing piece of the transmission will have consequences on the result and even byte ordering problems depending on the transfer method. This will be unacceptable in many cases, and will most likely corrupt the memory.

The most noticeable architecture difference in those two transfers is in its nature. The first can be unidirectional whereas the second needs to be bidirectional. For the data transfers to be ensured, there has to be a response, an acknowledgement of the state of the transfer. That response can be positive or negative, but without a response, the transmitting side cannot know if a transmission was successful or not.

Most of the time, when doing transfers without a response feedback, after a write data integrity can be checked by reading the same data and check if it matches back. This is time consuming, and usually involves an action on the user side. In other words, it is not transparent for the user.

A big part of the thesis was passed to implement a mechanism that verifies the transfers and correct them if needed, in a complete transparent way for the end user. Each frame is acknowledged upon its reception and an acknowledge frame is send back.

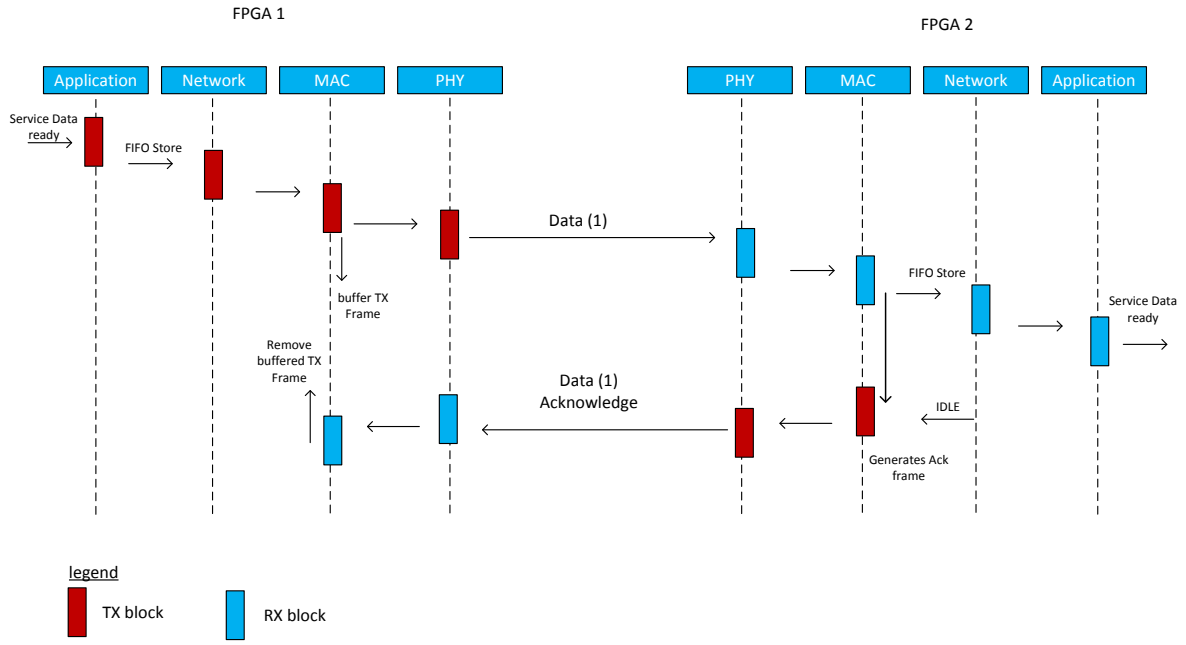


Figure 58: Single Transmission with retransmission

The mechanism is made up of two big parts:

- A TX buffer that keeps sent frames
- An acknowledge frame generator, based on the RX data reception.

The TX buffer keeps the sent frames until they are acknowledged as received. If the transmission failed and the frame is marked as not valid, then the frame is resend and rebuffed until validated.

The Mac Layer contains both the TX buffer in the retransmission manager and the RX acknowledge frame generator.

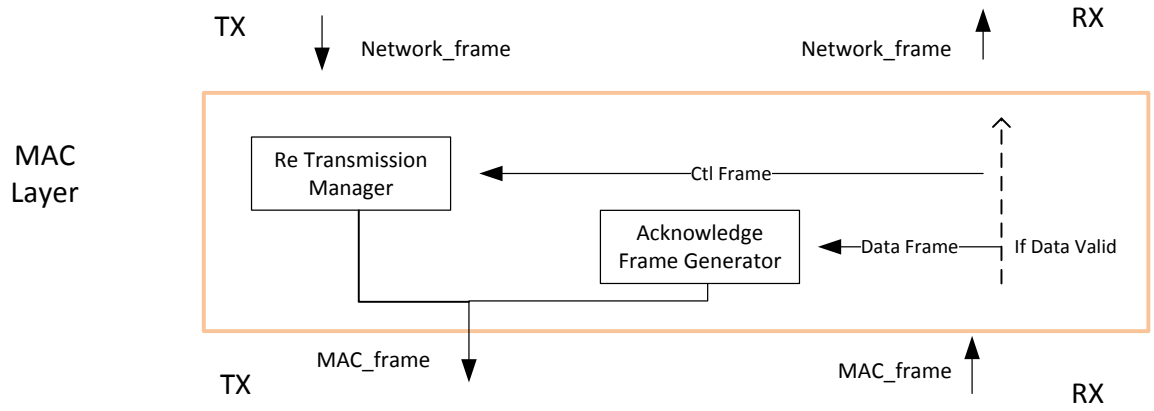


Figure 59: Retransmission Mac architecture overview

AckFrame Generation

One of the biggest issue when creating this acknowledge mechanism was its bandwidth need.

If for each incoming frame, there needs to be an outgoing acknowledge frame, then the RX bandwidth will equal the TX one, and leaves very little place for bi directional data traffic, essentially cutting the total bandwidth in half.

Sending the response for a single frame reception in a GBT frame on its own is also not bandwidth efficient.

To deal with this issue, and reduce the bandwidth utilization by this mechanism, acknowledgement of incoming frames have the possibility to be grouped together. The status of reception of up to 32 frames can be grouped in a single GBT frame.

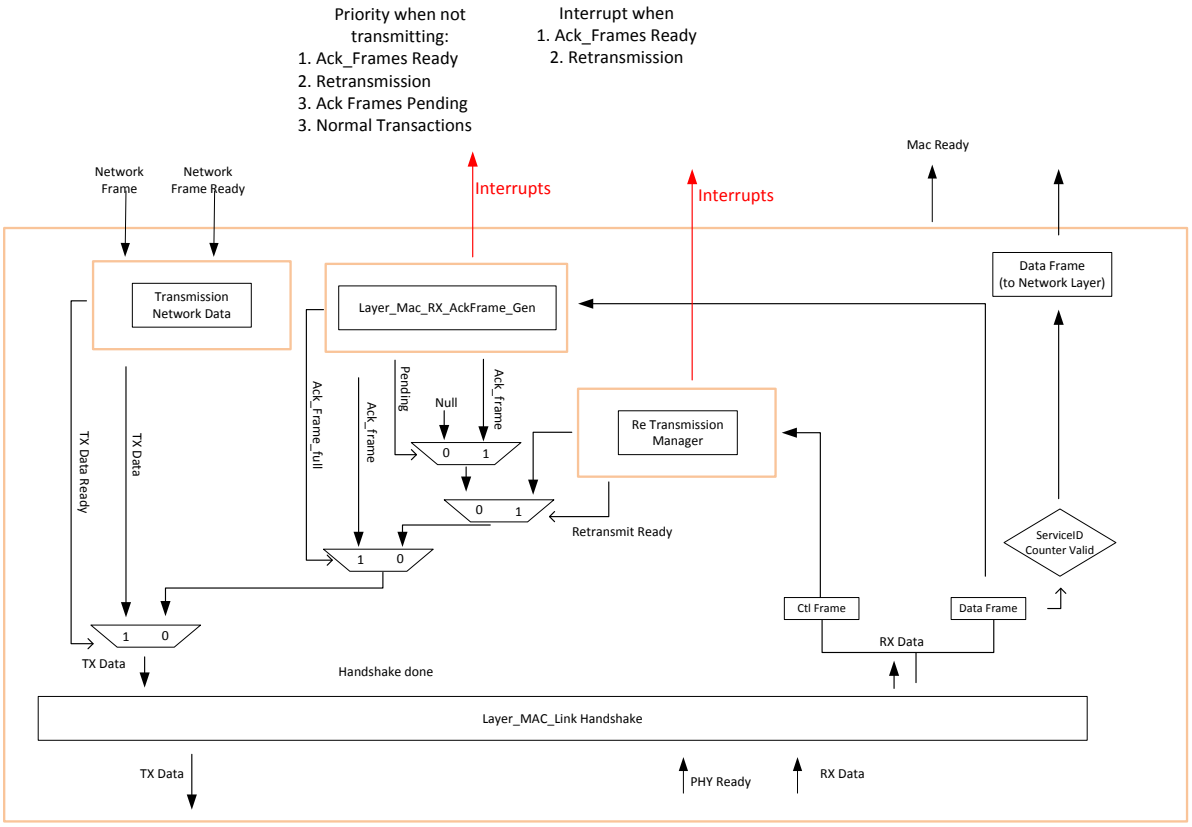


Figure 60 : Complete Mac Architecture

The above figure shows a detail view of the Mac architecture. The sending priority of that acknowledge frame is done as followed:

- When a RX frame is received and no TX frame is currently being send, then the acknowledge frame is sent with 1 frame validation.
- When a RX frame is received and a TX transaction is happening, then the validation is buffered and the number of validation frames incremented. When no TX transaction is seen on a clock cycle, the acknowledge frame is sent.
- If more than 31 frames are received while no IDLE time for a TX transaction, then an interrupt is made from the MAC layer to the Network Layer, creating an IDLE Data transmission clock cycle, allowing the full acknowledge frame to be sent.

The implications of this mechanism are the following:

- When not continuously sending Data frames, the acknowledge frames are send on IDLE data frames, and thus don't take any needed bandwidth.

  In the worst case scenario, when both sides are transmitting at 100% bandwidth, $\frac{1}{32} = 3.125\%$ of the bandwidth will be taken by this mechanism.

The following picture is an example of a simplified data transaction between FPGA 1 and FPGA 2.

Scenario1: The FPGA1 needs to send 3 Data Frames to FPGA2.

The 3 Data Frames are sent one by one in a row. The FPGA2 receives them, and since no Data has to be send to FPGA1, acknowledge frames are created and sent for each Data Frame.

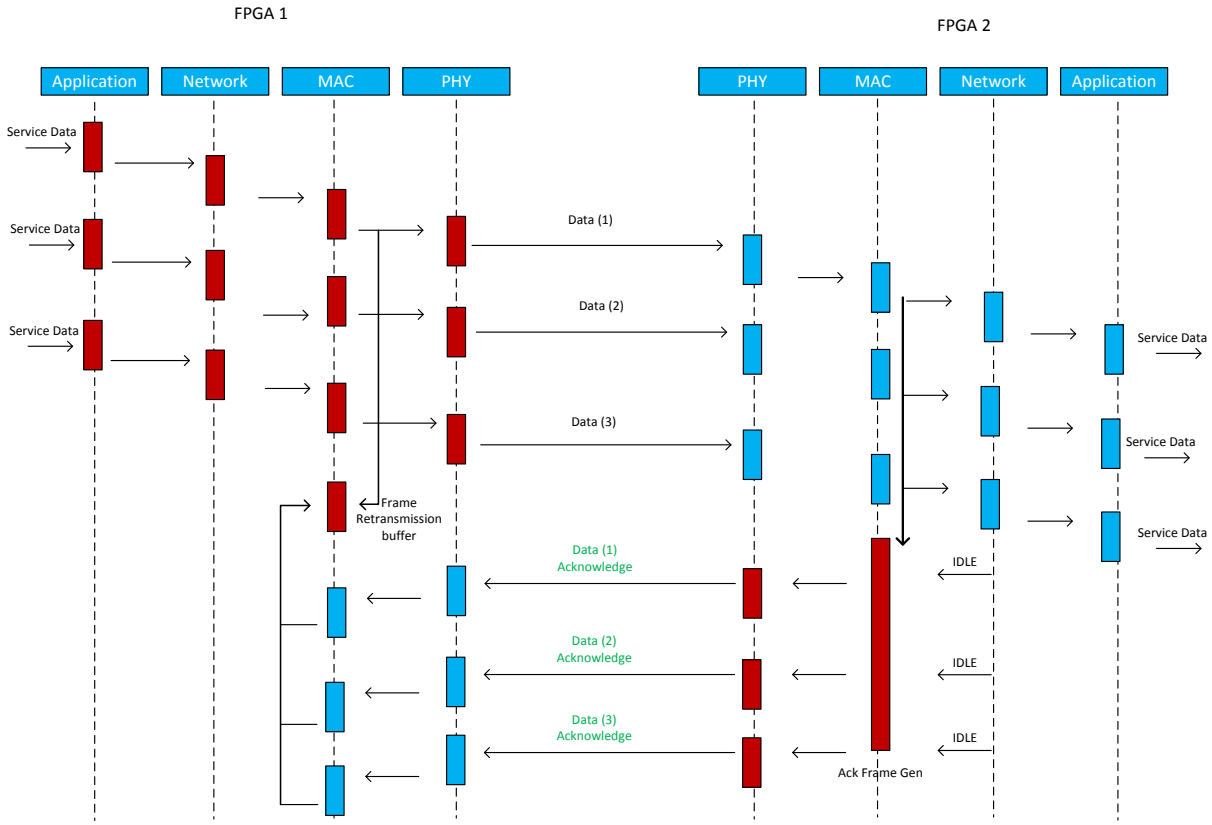


Figure 61 : transaction model for 3 frames + Acks

In case of a bi directional Data transmission, acknowledge frames cannot be send instantly if the network layer has information to send. In that case, the status of the received RX data frame is buffered until the Mac layer sees an idle frame coming from the network layer.

Scenario2: FPGA1 has 10 Data frames to send to FPGA2, while FPGA2 has 3 Data Frames to send.

FPGA1 will send its 10 data frames first, then send a single Ack Frame with the received status of FPGA2 Data frames 1, 2 and 3.

FPGA2 will send its 3 Data frames first, then send an Ack frame with the current buffered RX data frames (1, 2 and 3). Since no Data needs to be send anymore, the following frames will be Ack frames only containing the status of 1 Data frame.

The next diagram illustrates this example.

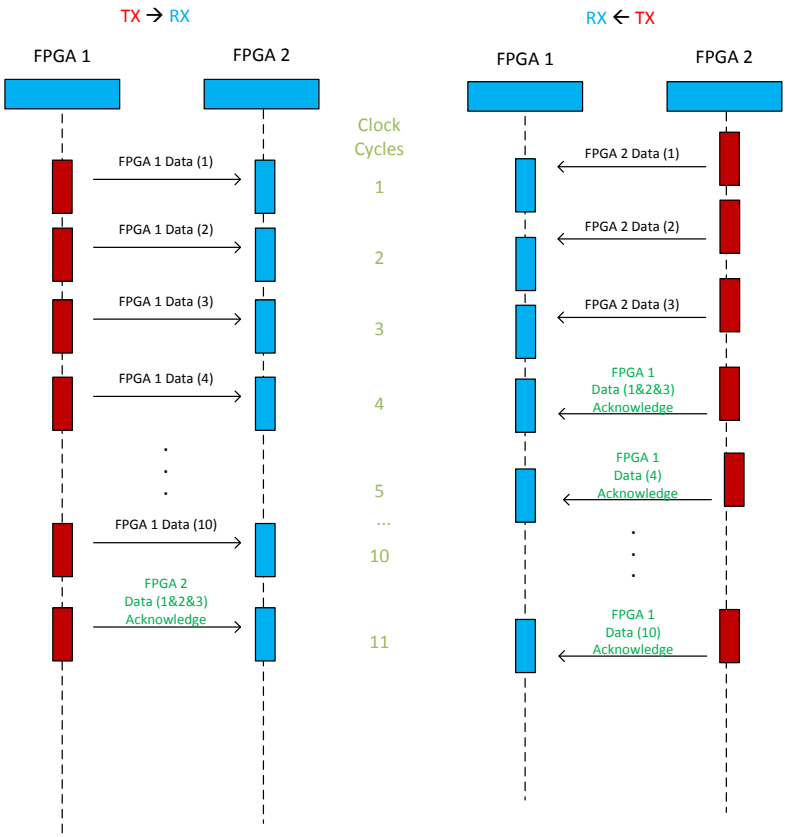


Figure 62 : bi-directional Data transmission with acks

In the case where more than 32 Rx data frames are received without any TX Idle frames at disposal, an acknowledge frame containing the status of those 32 frames will be forced out, interrupting the network layer current Data streaming for one frame (1 clock cycle).

Transmission errors

When the physical line integrity is compromised, bits errors will occur changing the data. The physical layer will either be able to recover from those with its forward error correction or not.

If the integrity cannot be regenerated, the frame in question will be marked as idle frame, as opposed to a Data Frame.

With such mechanism in the physical layer, a transmission error for the Mac layer is thus define as a missing frame. This ensures that any frame arriving in the Mac Layer is valid. The GBT layer had to be slightly modified to that regard, as it was still outputting error frames, potentially corrupting the protocol mechanism depending on the data received.

To detect this missing frame, each frame is tagged with an ID counter when sent, incrementing itself upon each new Tx frame. On the Rx side, this ID counter is then compared with a local Rx ID counter. When matching, the Rx counter is incremented. When the counters don't match, it means at least one or multiple frames were discarded by the physical layer.

To ensure service data transmission integrity, the order of the frames presented to the service needs to be preserved. To avoid complex buffering, it was decided that any incoming frame received not matching the ID counter should be discarded in the Mac layer and not transmitted further to the Network Layer. Further information regarding this decision can be found in the following section *"issues when conceiving the design*".

With such decision, a frame missing will interpret each of the following frames as invalid until the right ID is received.

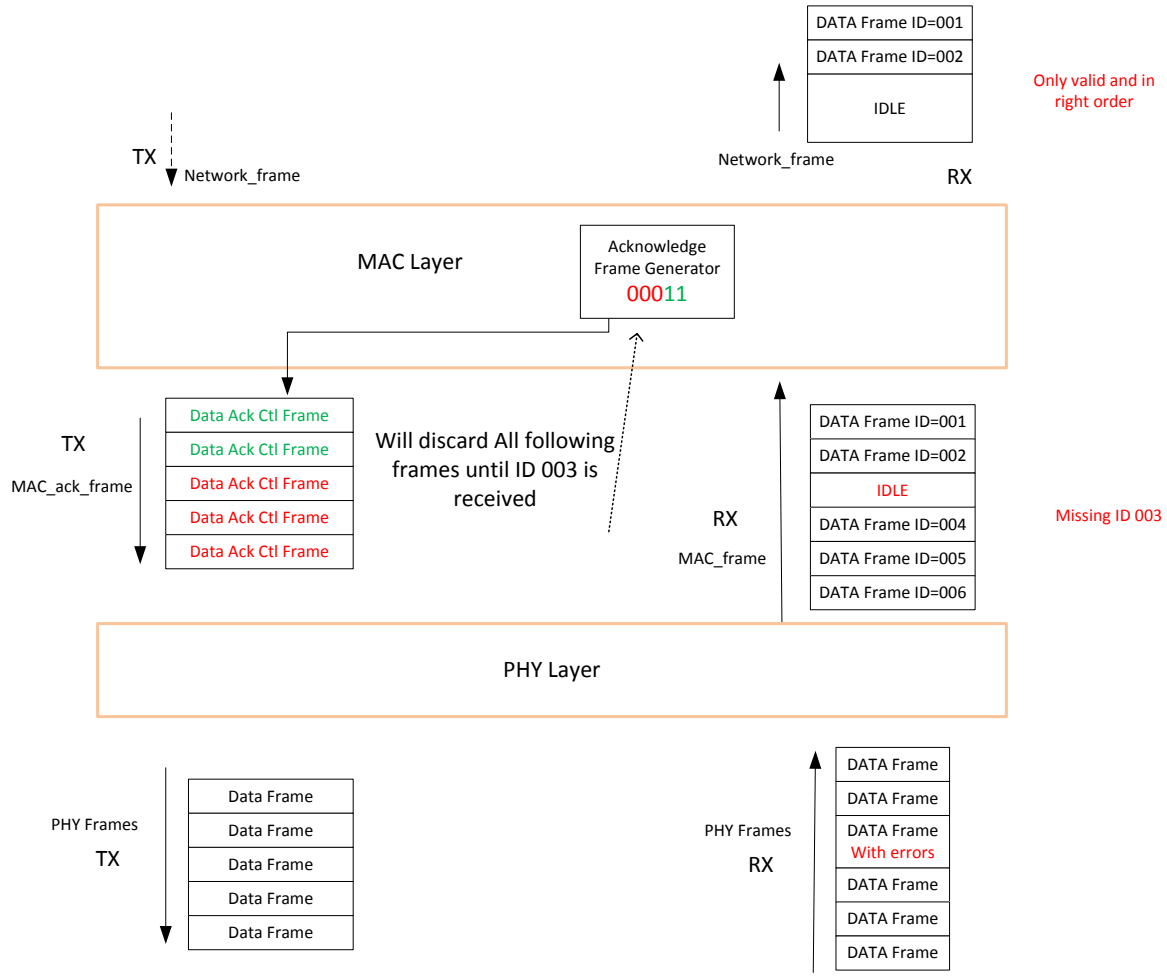


Figure 63 : Bad frame or missing frame Handle

When transmission errors occurs, the return status will be negative (in red), asking for a retransmission or positive (in green), validating the frame.

When receiving these acknowledge frames, the TX buffer FIFO will either erase the frame if the status was positive, or resend that frame while re-buffering it if the status was negative.

For more transmission problems scenarios, refer to the simulation section.

Retransmission Manager

Now that the reception of the frame is explained and how the status of this reception is sent back, it's easier to understand the first part of the mechanism that deals with frames buffering.

This part may look simpler than the frame acknowledge generation at first glance, only having to deal with buffering and removing frames.

The reality is however a bit more complex:

- The rate of frames removal needs to match the rate of buffering, not to have a FIFO overflow.

- The ID of the acknowledge frame needs to match the current FIFO output frame, to check for missing frames.
- A retransmission frame cannot be send if the network layer is also sending data.
- The acknowledge frames can contain 1 up to 32 frames to check. Rate of frame matching needs to be without delay not to have a FIFO overflow.

It was also necessary to stop the network from sending more TX frames and wait until the FIFO was empty before continuing with the normal transmission.

As a result, bandwidth is taking a temporary hit and go to 0 until all buffered frames are well received on the other side. The time needed for this task will depend on the total transmission delay between the two FPGAs. This delay is composed of a fix part, coming from the protocol and PHY time to send frames. The second part is the delay coming from the optical fiber length used. The FIFO's depth in the frame buffering mechanism needs to be big enough to contain all sent frames.

The fixed delay can either be seen in simulation or with signal tap in a loopback mode.

Determining the FIFO's depth

In order to know the minimum FIFO's depth needed the latency needs to be evaluated. The diagram below shows a single frame transaction and acknowledge, Points A to be B represent the data latency between the 2 boards, while point A to C represents the minimum time a frame needs to be kept in the buffer.

Below are the results with the single transaction in loopback mode. The same FPGA is sending and receiving the data. The fiber optic cable is around 30cm long.

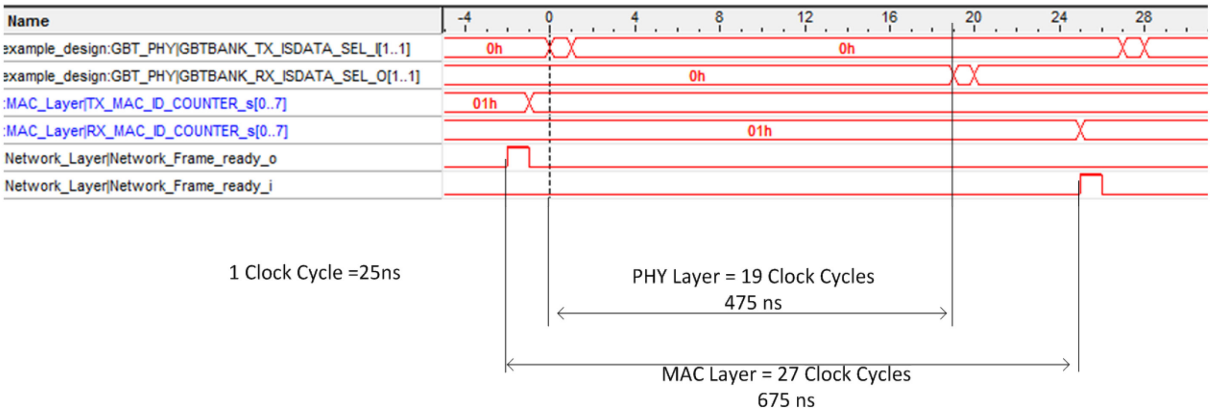


Figure 64 : Loopback mode Single Transaction Delay

It takes 19 clock cycles or 475 ns for a frame to pass though the GBT physical layer, and 27 clock cycles or 675ns to pass between MAC + PHY.

At this point, the data is received and validated (or not). The acknowledge frame for this data is then send. Only upon its reception will the buffered frame be removed.

If we look at the total transaction minimum transaction time:

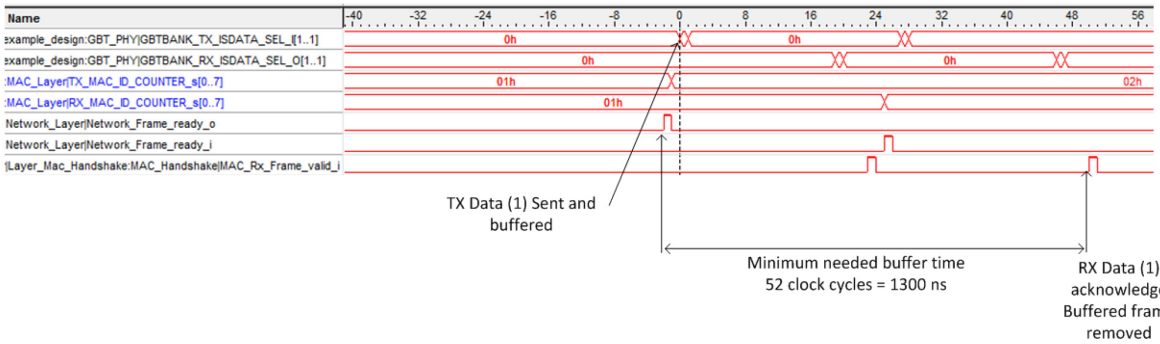


Figure 65: Loopback mode Single Transaction +Ack Delay

This means that if we continually send data, the FIFO needs to be able to hold at least those 52 frames and potentially 32 more if the other side is also sending information, and cannot send the acknowledge frame instantly.

The current FIFO used for the design can contain up to 256 frames, leaving enough space to account for higher fiber optic length:

$$(256 - 52 - 32) * 25ns = 172 * 25ns = 4300 \text{ ns}$$ Equation 1

Since the data need to travel back and forth, each line can have a delay of:

$$\frac{4300\text{ns}}{2} = 2150ns$$ Equation 2

"Optical cables transfer data at the speed of light in glass. This is the speed of light in vacuum divided by the refractive index of the glass used, typically around 180,000 to 200,000 km/s, resulting in 5.0 to 5.5 microseconds of latency per km." [5]

As such, the maximum length permitted by this fifo's depth, under the worst conditions (100% bandwidth utilisation) can be calculated as below:

$$\frac{2150\text{ns}}{5500ns} * 1km = 0.390km$$ Equation 3

If more fiber optic length is needed between the two FPGAs then the FIFO's depth will need to be re-evaluated. For the standard design, the 256 element depth is a good compromise between the possibilities and the logic space taken.

5.11. Triggers and IOs Reproduction

Triggers and IO pins reproduction were a big part of the technical requirements for CERN, as the need for very low latency jitter is essential. This requirement disqualified many existing protocols, unable to reach a 100 ns jitter.

The GBT physical layer was chosen in the end for its ability to implement a normal latency mode and a special latency mode, should the first not be enough.

The Trigger and IO pins are however handled differently in the protocol despite their almost identical nature.

Even if both have a critical latency characteristic, Triggers will be kept at 1 or 2 in their numbers, while IO Pins can be much higher, potentially 16 or 20. An IO pin reproduction example can be seen as an ADC input bus, or buttons/Leds reproduction.

Since the protocol frames are encapsulated into GBT frames to benefit from the forward Error correction on every frame, the number of bits available to continuously send the state of an IO with each frame is limited.

Furthermore, sending frames every clock cycle for IO reproduction would mean sending data frames every clock cycle, much like using the GBT frames without the protocol. Issues could also happen when frames need to be retransmitted.

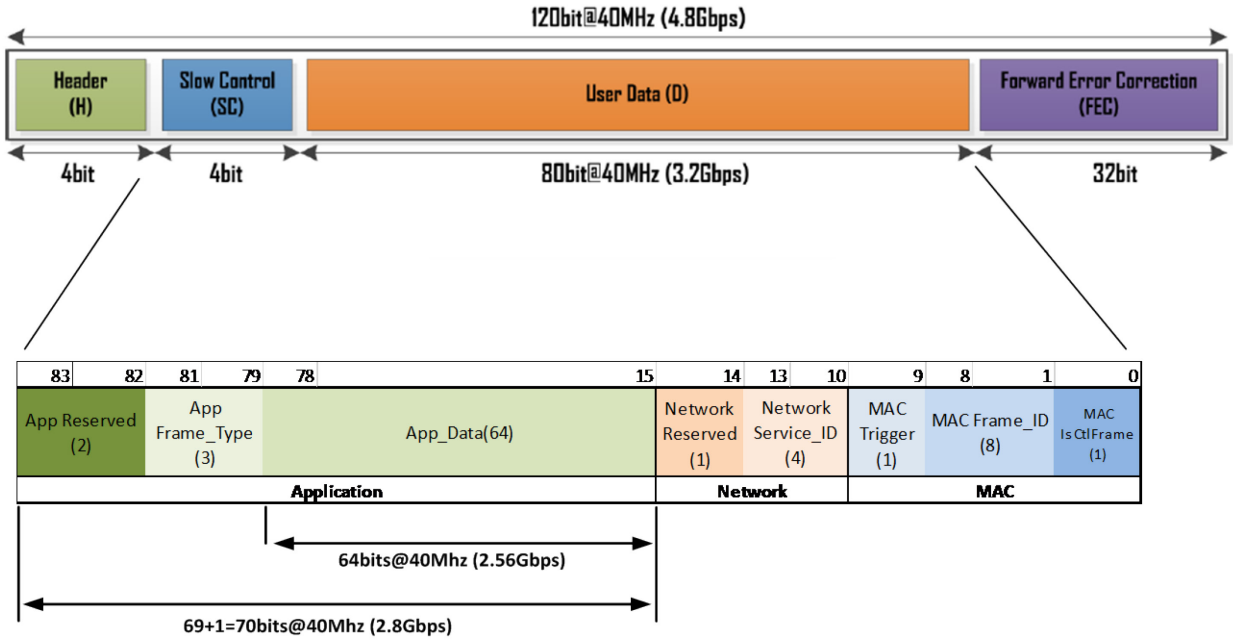


Figure 66: Usable Protocol Data Frame bandwidth

To deal with this issue, only the trigger is sent with every frames, while IO pins reproduction is consider a service with its specific data frame.

On the figure above, the trigger is located in the MAC data field. For the time being, only 1 trigger bit is set, but can be easily increased using reserved bits if needed. Only a constant needs to be changed in the protocol package file.

Trigger integration

The trigger integration needs to be reliable in latency, even if a frame is missing due to error transmission.

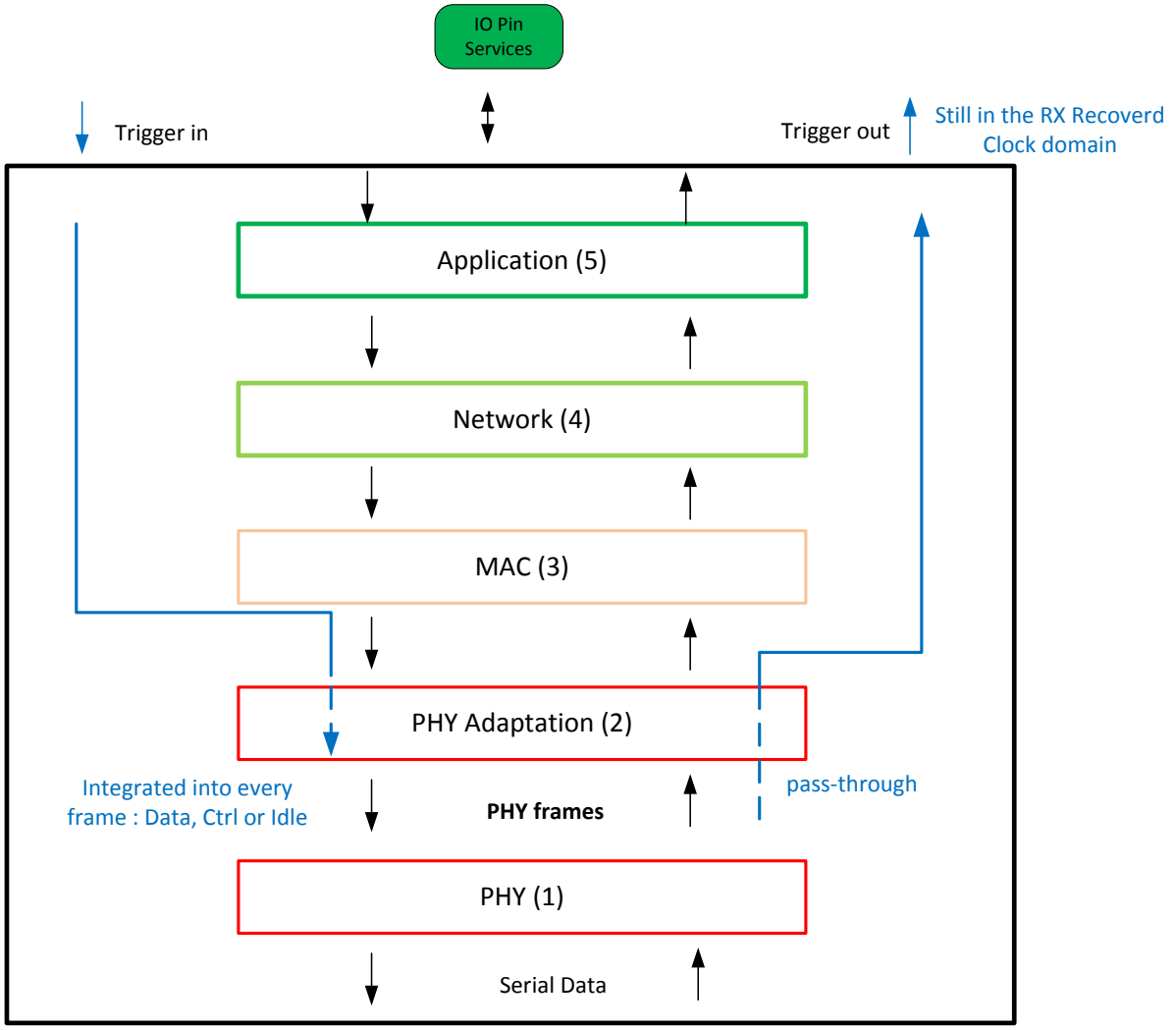


Figure 67: Trigger Integration

Because of that, the Trigger is integrated to every TX physical frames. All Data frames, Control frames and most importantly idle frames will contain the trigger field.

The physical adaptation layer, as seen in the Clocking architecture, contains an RX elastic buffer to deal with the time crossing domain from the RX recovered clock to the TX clock. Physical Idle frames are not buffered to avoid a FIFO overrun. The trigger however needs to pass through in every condition.

To overcome this problem, the trigger received is taken out of every physical frame and directly goes out of the protocol. The signal is thus still in the RX recovered clock domain. This also guaranties that no jitter latency is added. Indeed, since the GBT RX side is entirely in the RX recovered clock domain, no jitter is added in the RX physical layer. It is then up to the user, to either use this signal as it is and use it as an output to a pin, or change its clock domain. Changing the clock domain will add a jitter latency equal to the clock's period. Thus, sampling this signal with the TX clock at 40Mhz will create up to 25 ns jitter.

If the jitter's source is coming from a Pin, then a sampling jitter will be added upon its level detection equal to the TX clock period, 25ns.

Overall, if the Rx trigger goes through the time domain crossing, the maximum jitter will be:

$$t_{totalJitter} = t_{RxSamplingJitter} + t_{TxSamplingJitter}$$

If the same clock is used both times, then the equation becomes:

$$t_{totalJitter} = 2 * t_{SamplingJitter} = 2 * t_{clockPeriod}$$

In this case, since the used clock has a period of 25ns, the total jitter will be 50 ns, which fits the requirements of the 100 ns fixed by CERN.

IO Pin integration Application Service

IO Pin reproduction behaves in a different way than triggers. Due to their higher number, they need to be treated as a standard service.
The service works as a streaming link, continuously streaming the state of signals. Such service behavior has a few drawbacks:

- Pin states need to be buffered and send in an application frame
- The initial latency of the stream isn't fixed.

Because sending a data frame every clock cycle with only the current state of the pins would take all available bandwidth, the pin states need to be buffered and fill the application data frame. The App_Data bus is a 64 bit vector. Therefore, 64 states can be buffered and sent into a single Data application frame.
This induce a bigger latency than the trigger mechanism, depending on the number of pins to buffer.
The buffering time inside the Service Application Data can be calculated as such:

$$t_{App\_latency} = round(\frac{\mathrm{AppData}_{Size}[bits]}{NbPin})$$

t measured in clock cycles

With $\mathrm{AppData}_{Size} = 64$ and $NbPin$ Representing the number of pins to reproduce through the service.

If 1 pin needs to be reproduced then the latency inside the Application service will be 64 clock cycles. If 2 pins needs to be reproduced, then they will be each buffer 32 times before the frame will be sent.

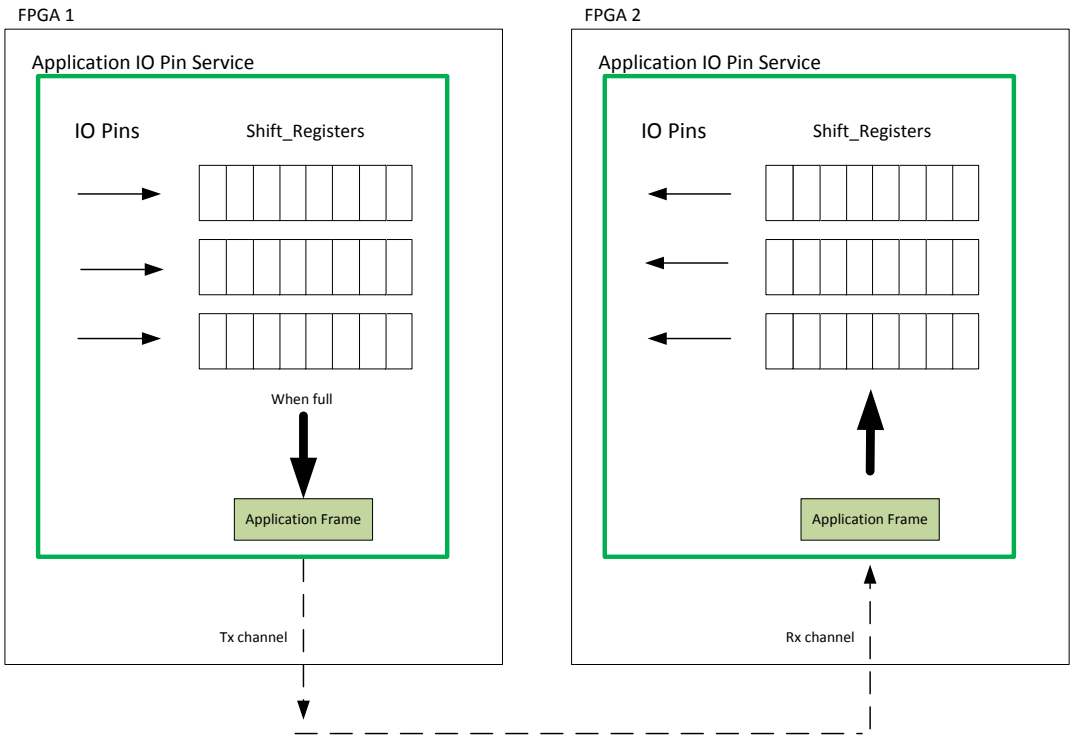


Figure 68: IO Pin Application Service principle

The second issue, once the application frame has been sent to the Network Layer, is determining when the frame will be transmitted to the Mac Layer. Since the FIFO associate with this service will be empty, the worst case scenario will be when the frame is ready just after the arbiter was checking its request. In this case, the maximum delay for the frame transmission will be:

$$t_{NetworkLatency}(y) = \sum_{i=1}^{NBservice} [Services\_Priority\_Weight(i)] - Service\_Priority\_Weight(y)$$

t measured in clock cycles

In other words, the max delay will be when all other services will have been served and the grant can come back to the Pin IO service again.

If the service priority weight is properly setup, the frame will be taken out of the FIFO before the next one is ready.

For example, if a total priority weight of 10 (not counting the IO Pin service) is entered, then the maximum network jitter will be 10*cycles, or 250ns.

On the RX side, a similar issue will arise if the application service is reading the first frame as soon as it is ready. A certain buffer size needs to be present before the start of the streaming process can begin, avoiding an Rx fifo empty issue.

When the streaming link has begun, the latency will stay fixed assuming bandwidth allocation was correctly set. The network layer will only induce this latency at the start, and will stay fixed afterwards. The latency is however unknown at the start, unless the user makes sure all other services are blocked until the first IO pin frame is sent.

If transmission errors occurs, it would be wise to let the RX buffer increase sufficiently before starting the stream, to overcome a momentary lack of incoming frames.

The Pin_IO_Service.vhd service provides a working IO Pin streaming service with a generic approach regarding the IO pins to reproduce and the initial RX buffer. The same service can thus be used to stream any number of pins from 1 to 64.

The complete maximum latency for the IO pins can be calculated as the following:

$$t_{TotalPinLatency} = t_{App\_latency} + t_{NetworkLatency}(y) + t_{Network2NetworkLatency} + RxFrameDelay * t_{App\_latency}$$

With:

t measured in clock cycles

$t_{App\_latency} = buffered\ time\ inside\ the\ application\ Service$

$t_{NetworkLatency}(y) = maximum\ latency\ from\ Scheduling$

$t_{Network2NetworkLatency} = total\ protocol\ latency\ between\ TX\ network\ out\ and\ RX\ network\ in$

$RxFrameDelay = Number\ of\ RX\ application\ frames\ buffered\ before\ the\ start\ of\ the\ stream$

To summarize, the trigger mechanism has a short fixed transmission delay ensured, and a maximum jitter delay known coming from the sampling clock. The IO pin reproduction has a higher delay, and a higher jitter coming from the protocol priority queue. Both mechanism satisfy different needs, and it is up to the user to decide if the network jitter latency from

5.12. Communication Latency

The latency for each layer is either fixed or deterministic based on knows factors.

The summary of the TX and RX latency is shown in the next figure.

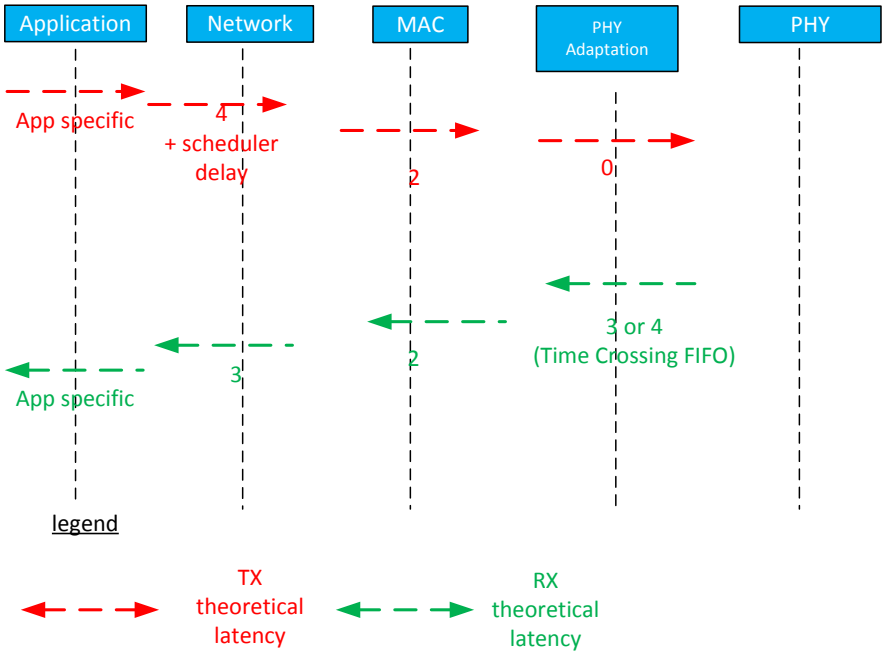


Figure 69: Overall protocol Layers latency

TX transmission

- Each Application service as its own latency. For most cases, this latency will be 1 as the transfers are accepted from the Service side, and directly send to the network layer. The IO pin Application service is an exception, as data needs to be buffered first.

- The network needs a minimum of 2 clock cycles to pass the TX FIFO, 1 for the arbiter, and 1 to add information to the frame. On top of that comes the delay that can be induced by the arbiter scheduling. The maximum latency was calculated as:

$$t_{NetworkLatency}(y) = \sum_{i=1}^{NBservice} [Services\_Priority\_Weight(i)] - Service\_Priority\_Weight(y)$$

t measured in clock cycles

- The Mac layers needs to add its information and information needs to pass through the handshake entity, making the latency 2 clock cycles.

- Since the Tx side has the same clock as the PHY , the PHY adaptation adds no latency.

RX transmission

- Depending on clock phase shift, the PHY adaptation layer will need between 3 to 4 clock cycles to pass the information to the mac.
- In the mac, the information travels through the handshake entity for 1 clock cycle, get the Mac information removed before it's passed to the network
- The network needs 1 clock cycle to remove its information, and 2 to pass the RX FIFO.
- The RX application services also have their own latency, depending on their type.

6. Simulation

6.1. Simulation framework (UVVM)

To test and validate the implemented design, a vhdl simulation test bench has been created, capable of creating corner cases scenarios.

The simulation framework used to test the implemented design is using the UVVM open source framework, developed by the company bitvis, which not only speed up the simulation creation process but also greatly facilitates it.

The UVVM framework is a VHDL 2008 verification environment based on the UVM methodology that makes use of transaction level modeling (TLM), by increasing the level of abstraction.

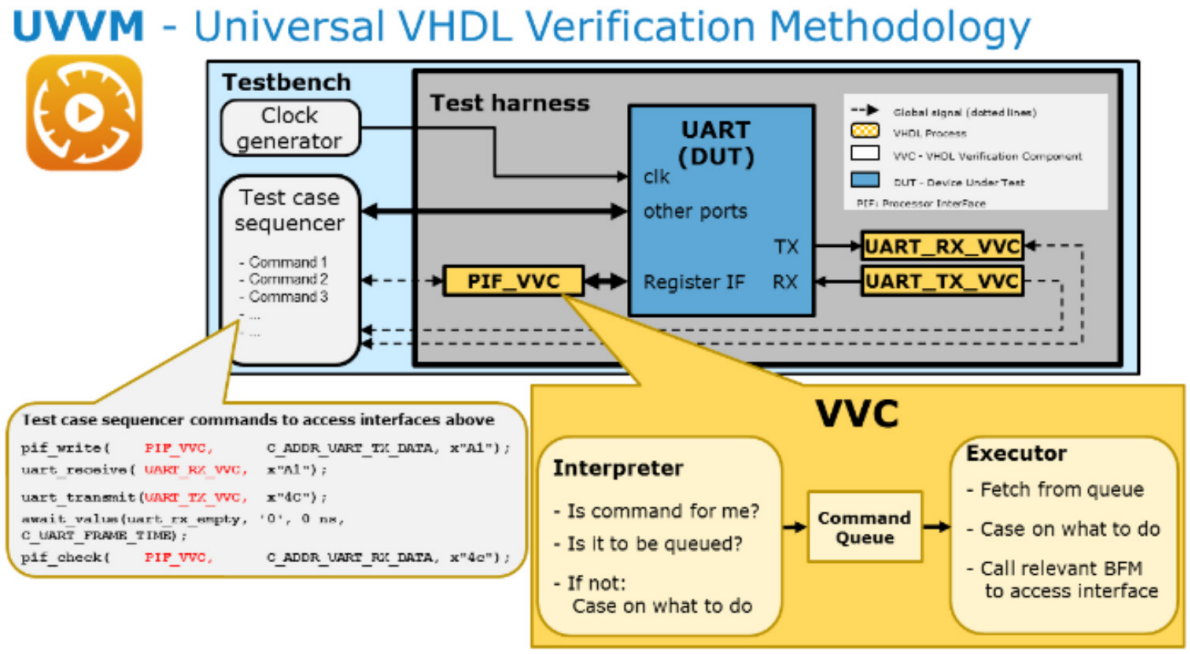


Figure 70: UVVM Framework Presentation

A detailed explanation of this framework can be found on bitvis website [13], as well as in Annex 5.

Simulation structure overview

The simulation has the general structure shown in the figure below.

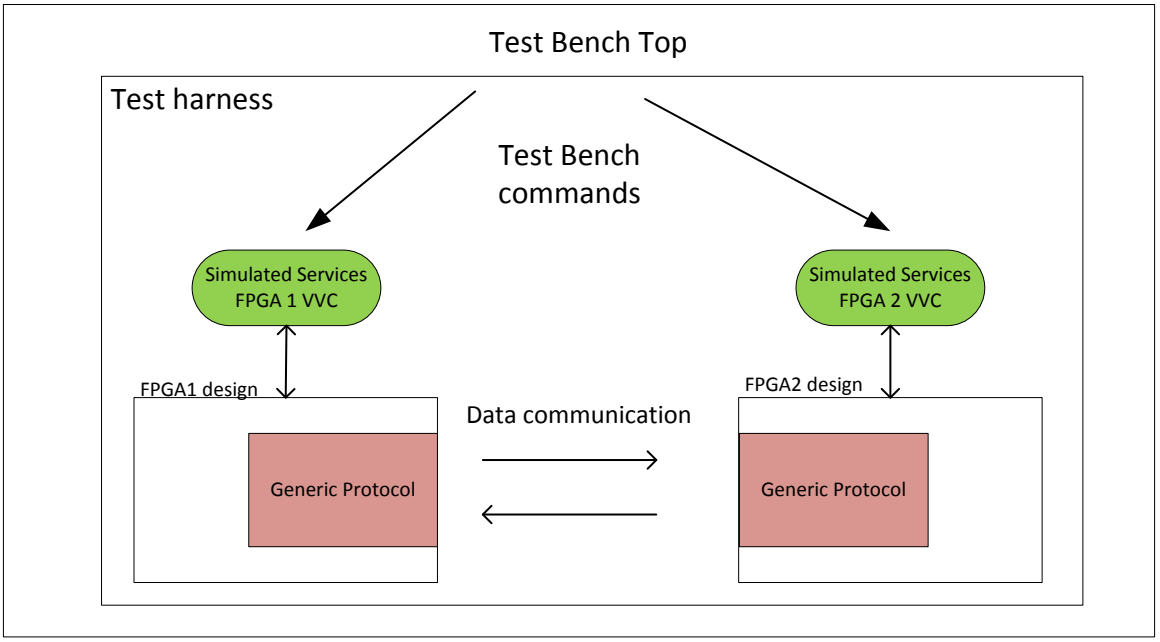


Figure 71: Simulation Test bench architecture

To speed up the simulation, the generic protocol can be implemented in two ways:

- Simulating the GBT physical layer including the Altera Transceivers IPs. The Data communication between the 2 designs is then a serial communication, done by the transceivers.
- Bypassing the physical layer and transceivers. The data communication is then a parallel communication, linking the respective Physical frames of each design, the TX frame of one to the RX frame of the other.

The second method is making use of a VHDL 2008 feature, where internal signals can be forced to certain values directly from the Test Bench without code modification. This is a greatly appreciate feature, as those internal buses are not part of the top entity, and would have require special entities for this implemented way. Forcing the GBT and transceiver clocks to 0, and connecting those internal buses, speed up the simulation by a big factor (60x or more), passing from around 2min to 2sec simulation time.

A generic parameter in the Test_Bench_Top determines if the GBT is bypassed or not for the simulation.

Many Tests can be made with the simulation test bench, and not all will be shown here. Some simple simulations to validate the design will be explained here. A complete explanation of the test bench, its use and the corner cases that can be done will be presented in another document for the CERN users.

GBT PHY latency simulation

Using the complete GBT physical layer including the transceivers, it's possible to have a good estimate of a transaction delay.

A single transaction such as seen below can be simulated for timing purposes.

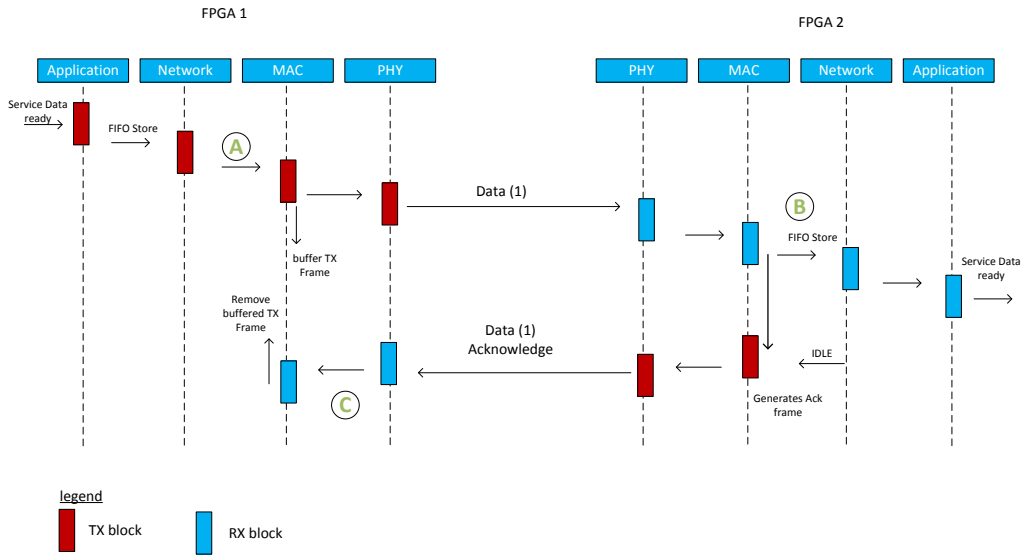


Figure 72: Single transaction with Ack timing

From points (A) to point (B) in the picture above, The results show a 793ns delay between the network layer of FPGA1 and FPGA2.

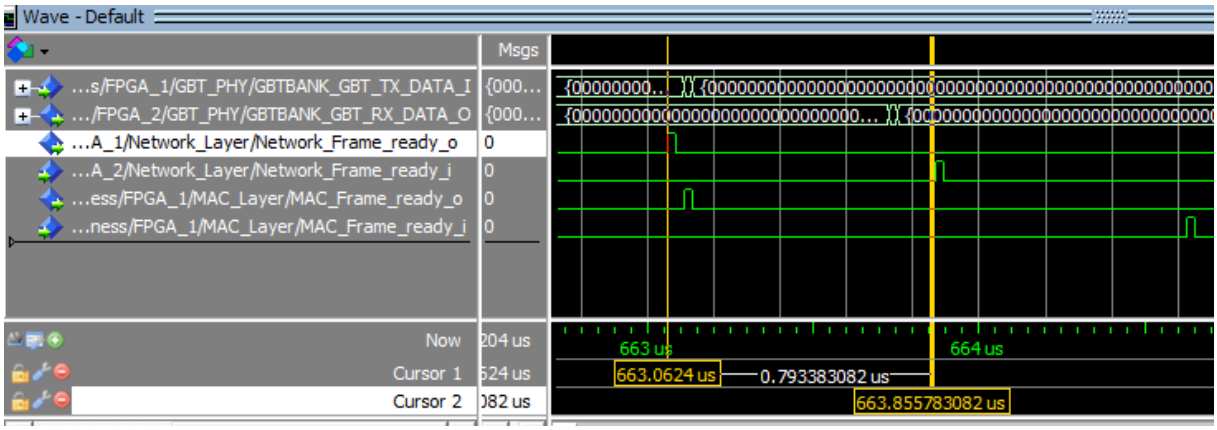


Figure 73: Simulation Timing Network to network delay

For a total time transaction, from points Ⓐ to point Ⓒ, the simulation results show a 1562 ns delay.

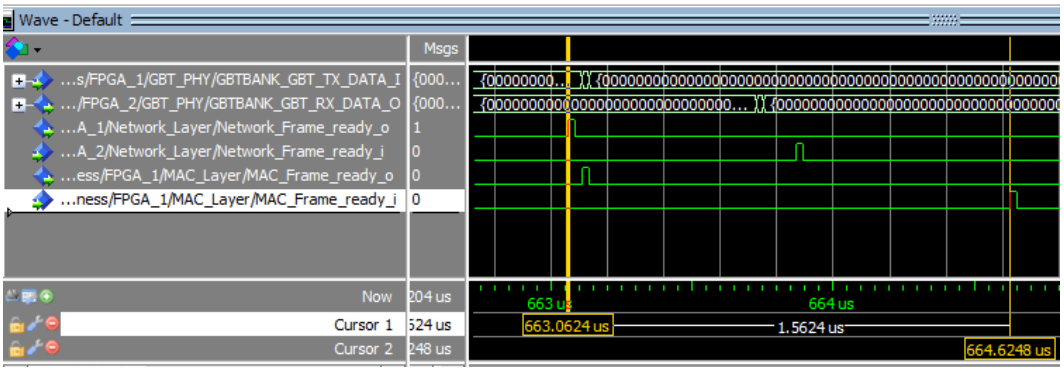


Figure 74: Simulation Timing single transaction

This is a good estimation of transaction delays. As seen with signal tap results afterwards, the real delays were slightly different, showing lower numbers, 675ns and 1300 ns respectively.

Agents types available

The Agents types, or VVC for vhdl verification components, are available for the Avalon Master and Slave transactions.

The Avalon master provided by bitvis had to be modified in order to do read transactions.

An Avalon Slave VVC had to be created in order to validate bi-directional traffic simulation tests.

6.2. Bandwidth allocation Simulations

Connecting different Avalon Services to the protocol enables to simulate different bandwidth allocation

A first a simple test to understand the bandwidth allocation can be shown, with
Scenario 1:

- 4 Avalon Master Services doing 10 Write transactions to Avalon Slave Services.
- The Avalon master 4 does 20 transactions.
- All send their data once every 4 clock cycles, 25% bandwidth each
- Priority weight is set to 1 each

The simulation results are focused on the scheduling and the TX FIFO occupation.

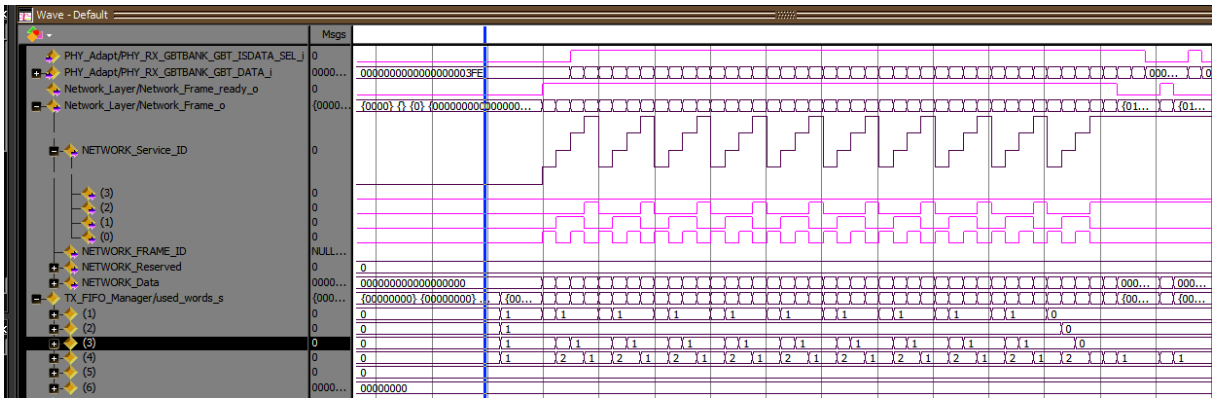


Figure 75 : bandwidth Simulation of 4 Avalon Master (1)

As seen with the graph, the services are served equally. The bandwidth is used at 100%, and their FIFO are not going over 2 words in.

Scenario 2:

- 4 Avalon Master Services doing 10 Write transactions to Avalon Slave Services.
- The Avalon master 4 does **20 transactions at full speed (100%)**
- All others send their data once every 4 clock cycles, 25% bandwidth each
- Priority weight is set to 1 each

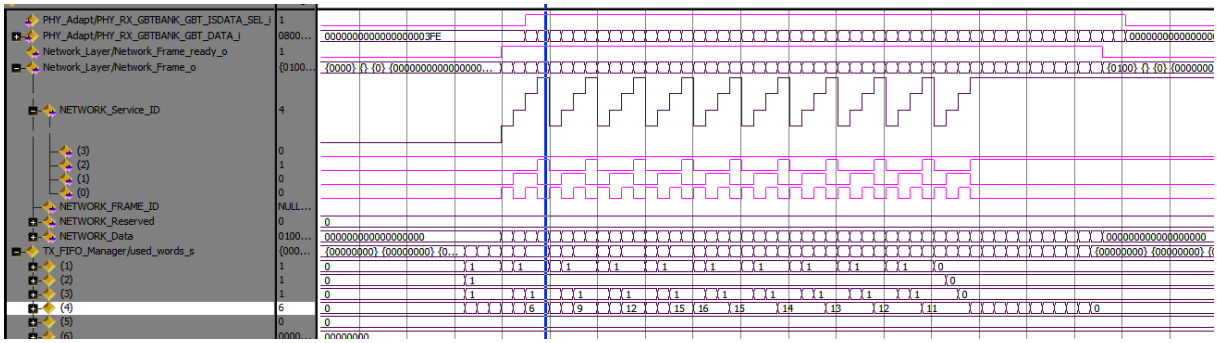


Figure 76: bandwidth Simulation of 4 Avalon Master (2)

Here, the allocation hasn't changed is still 1 transaction each. However, not the FIFO 4 is getting filled up and goes up to 16 frames, because of the high burst writes, with the same allocation than the others.

For every simulation, the transactions details can be viewed with the created log.

```
UVVM: ID_LOG_HDR                        0.0 ns  TB seq.              Starting simulation of TB for Protocol using VVCs
UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000000") completed.   [21]
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000A") completed.   [51]
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000014") completed.   [81]
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [82]
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,2      Running: insert_delay(AVALON_MM_VVC,2,fpga1, 75600000 fs)  [52]
UVVM: ID_BFM                         2799.7 ns  AVALON_MM_VVC,1      Running: insert_delay(AVALON_MM_VVC,1,fpga1, 75600000 fs)  [22]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000001") completed.   [24]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000B") completed.   [54]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000015") completed.   [84]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [85]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,2      Running: insert_delay(AVALON_MM_VVC,2,fpga1, 75600000 fs)  [55]
UVVM: ID_BFM                         2900.5 ns  AVALON_MM_VVC,1      Running: insert_delay(AVALON_MM_VVC,1,fpga1, 75600000 fs)  [25]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000002") completed.   [27]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000C") completed.   [57]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000016") completed.   [87]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [88]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,2      Running: insert_delay(AVALON_MM_VVC,2,fpga1, 75600000 fs)  [58]
UVVM: ID_BFM                         3001.3 ns  AVALON_MM_VVC,1      Running: insert_delay(AVALON_MM_VVC,1,fpga1, 75600000 fs)  [28]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000003") completed.   [30]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000D") completed.   [60]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000017") completed.   [90]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [91]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,2      Running: insert_delay(AVALON_MM_VVC,2,fpga1, 75600000 fs)  [61]
UVVM: ID_BFM                         3102.1 ns  AVALON_MM_VVC,1      Running: insert_delay(AVALON_MM_VVC,1,fpga1, 75600000 fs)  [31]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000004") completed.   [33]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000E") completed.   [63]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000018") completed.   [93]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [94]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,2      Running: insert_delay(AVALON_MM_VVC,2,fpga1, 75600000 fs)  [64]
UVVM: ID_BFM                         3202.9 ns  AVALON_MM_VVC,1      Running: insert_delay(AVALON_MM_VVC,1,fpga1, 75600000 fs)  [34]
UVVM: ID_BFM                         3238.2 ns  AVALONMM_SLAVE_VVC,1 avalonSlave_Write_expected(A:x"00000001", x"00000000")=> OK, received data = x"00000000".   [23]
UVVM: ID_BFM                         3263.4 ns  AVALONMM_SLAVE_VVC,2 avalonSlave_Write_expected(A:x"00000001", x"0000000A")=> OK, received data = x"A".   [53]
UVVM: ID_BFM                         3288.6 ns  AVALONMM_SLAVE_VVC,3 avalonSlave_Write_expected(A:x"00000001", x"00000014")=> OK, received data = x"14".   [83]
UVVM: ID_BFM                         3303.7 ns  AVALON_MM_VVC,1      avalon_mm_write(A:x"00000001", x"00000005") completed.   [36]
UVVM: ID_BFM                         3303.7 ns  AVALON_MM_VVC,2      avalon_mm_write(A:x"00000001", x"0000000F") completed.   [66]
UVVM: ID_BFM                         3303.7 ns  AVALON_MM_VVC,3      avalon_mm_write(A:x"00000001", x"00000019") completed.   [96]
UVVM: ID_BFM                         3303.7 ns  AVALON_MM_VVC,3      Running: insert_delay(AVALON_MM_VVC,3,fpga1, 75600000 fs)  [97]
```

For more information about the UVVM simulations, refer to the UVVM website, or the Annex 5.

Scenario 3:

- 4 Avalon Master Services doing 10 Write transactions to Avalon Slave Services.
- The Avalon master 4 does 20 transactions at full speed (100%)
- All others send their data once every 4 clock cycles, 25% bandwidth each
- **Priority weight is set to 1 each except service 4 at a weight of 2.**

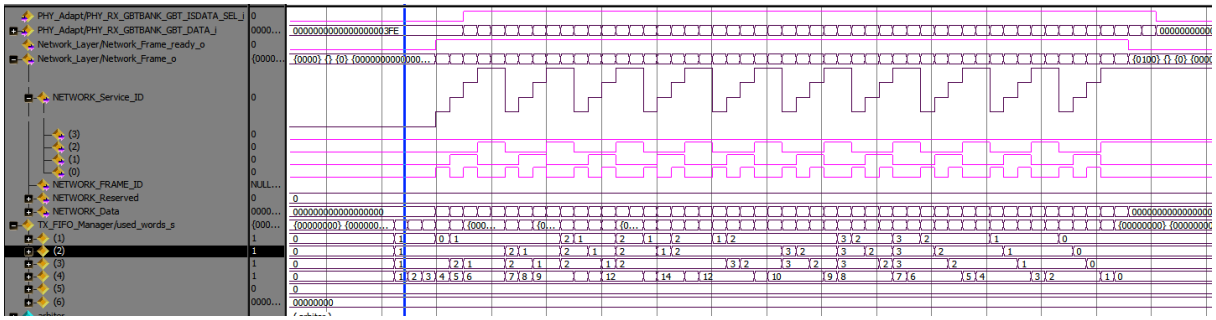


Figure 77: bandwidth Simulation of 4 Avalon Master (3)

Service 4 now has double the bandwidth compare to the others, and double the amount of data to send. The TX buffer still absorb the frames coming at 100%. We can see now that all services are finished at the same time (0 data remaining).

Scenario 4:

- 4 Avalon Master Services doing 10 Write transactions to Avalon Slave Services.
- The Avalon master 4 does 20 transactions at full speed (100%)
- All others send their data once every 4 clock cycles, 25% bandwidth each
- **Priority weight is set to 1 each except service 4 at a weight of 3.**

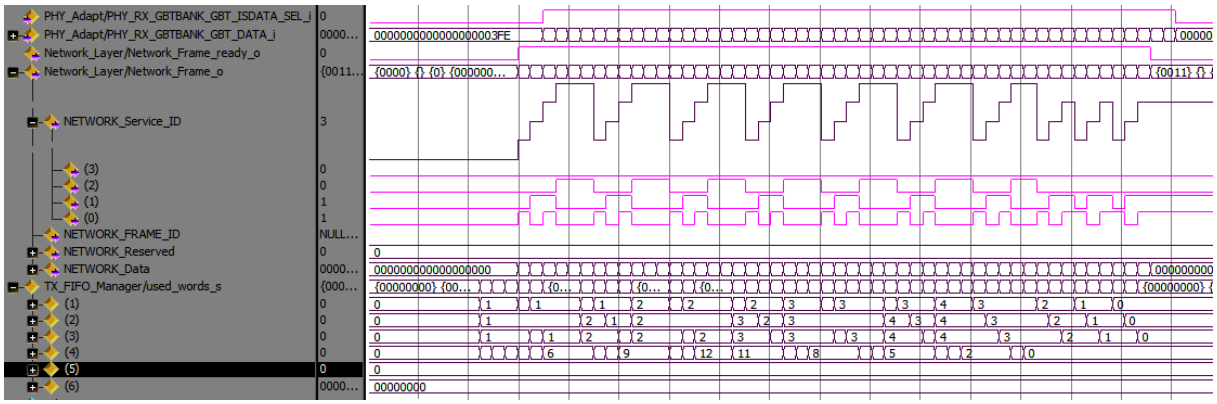


Figure 78: bandwidth Simulation of 4 Avalon Master (4)

Since Service 4 has 3 times more priority than the others, and only 2 times more data to send, the service has finished transferring its data before the others.

The ramp shows servers 4 no longer transmitting data, while the Arbiter still goes 2 times around the 3 others.

Avalon Master Read Transaction

It is also possible to do read transactions, but the Slave VVC doesn't have any memory attached or mapped for the moment.

Upon reception of a read, the Avalon Slave will always return "CAFECAFE".

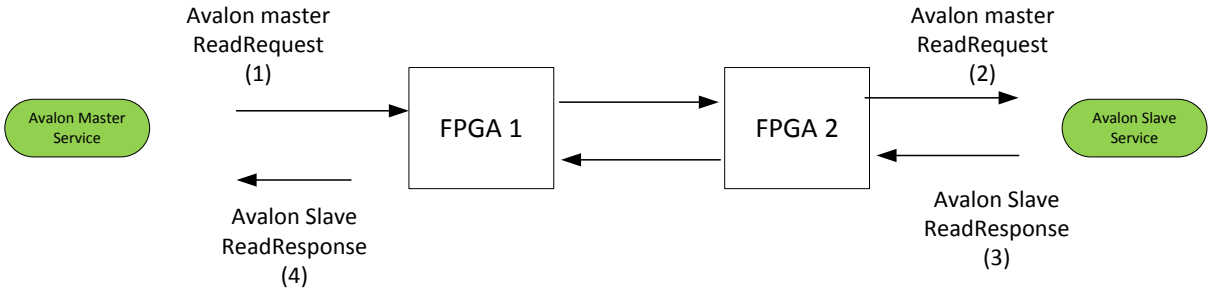


Figure 79 : Avalon Master Simulation transactions Diagram

```
# UVVM:
# UVVM: ID_LOG_HDR                    0.0 ns  TB seq.            Starting simulation of TB for Protocol using VVCs
# UVVM: ------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_BFM                     7257.6 ns  AVALONMM_SLAVE_VVC,1 avalonSlave_Read_expected(A:x"00000001")=> OK, Read at Address = x"1".   [22]
# UVVM: ID_BFM                     7688.5 ns  AVALON_MM_VVC,1      avalon_mm_read(A:x"00000001")=> x"CAFECAFE". reading test  [21]
```

From the Master Point of view, here are the signals seen:

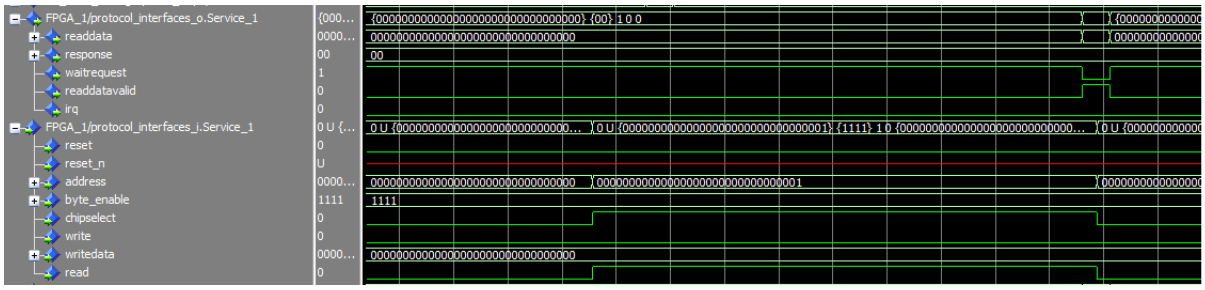


Figure 80: Simulation of an Avalon Read transaction Master POV

The master is asserting chipselect and the read signal while the waitrequest drops and ReaddataValid is asserted. This shows the transactions (1) and (4) from the diagram.

The following figures shows the Avalon bus at the Slave point of view.

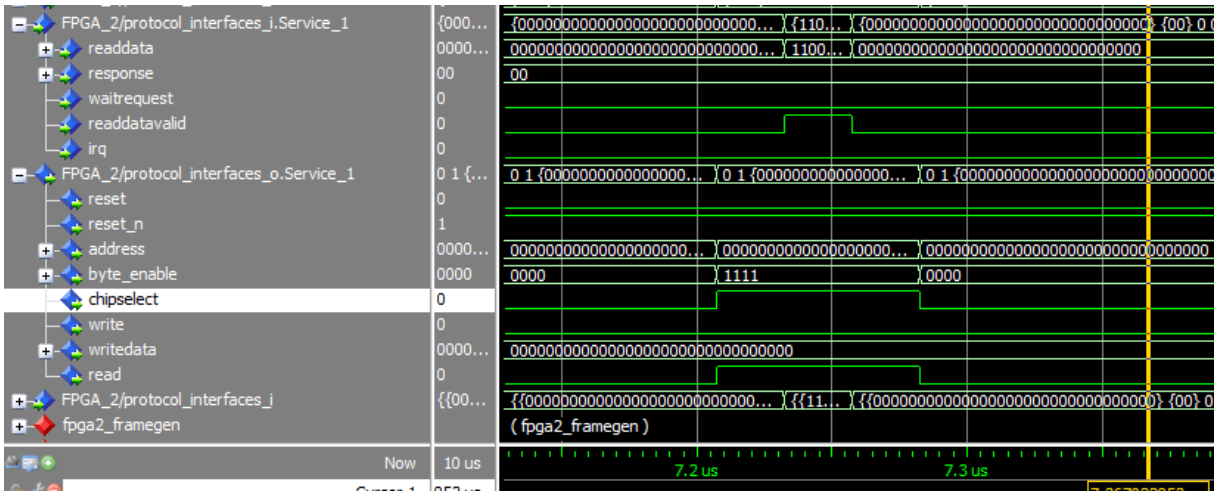


Figure 81: Simulation of an Avalon Read transaction Slave POV

The transaction from this side is a simple Read, a lot shorter, as it's only transactions (2) and (3).

6.3. Transmission Error Simulation

A big mechanism that could only be validated in simulation was the transmission error.
While no component was create to provoke an error, Modelsim force signal can have the same effect.
If the signal GBT_IS_DATA_SEL_i is set to 0 when it was a data frame, it would be like a lost frame.

Retaking scenario 4 from the Avalon transfers:

If during the transmission, 1 frame went missing, all of the network frames in the TX buffer would have to be resent. Only then can the transaction to the Services continue.

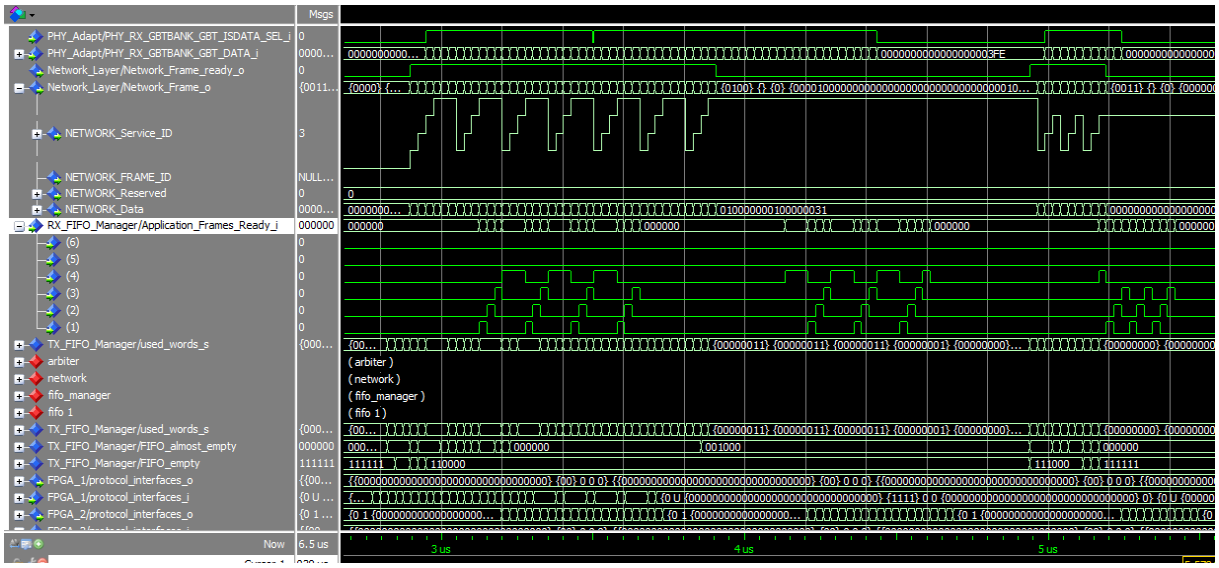


Figure 82: Simulation Error transmission

In the above figure, we can see the network frames being sent, just like scenario 4 without errors.

At around 3.5 us, ISDATA_SEL_I is force to 0 to generate a missing frame.

The result can be seen on the RX FIFO Manager, on the 2nd FPGA. Transmission to the application stop, and moments after is continuing.

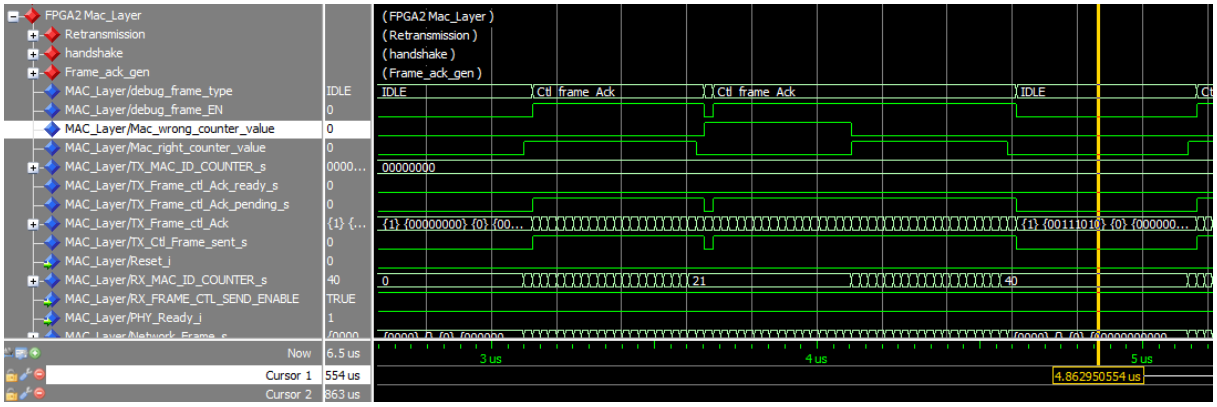


Figure 83: Simulation Error transmission RX Mac signals

When looking into the Mac layer from the FPGA2, we can see that the signal Wrong_counter_value is going up for a while. This is the amount of time for the ack to come back to the FPGA1, asking for a retransmission, and finally seeing the right Frame ID. The TX buffer, is resending his whole buffer, blocking any of his network frames, waiting to see if all the frames where this time received. Once the acknowledges are back a 2nd time, this time validating the frames, the FPGA1 resumes the transfer of its services.

This is why on the figure 82, we can see 3 distinct waves of arrival in the RX fifo.

- The first wave is the valid frames before the error.
- The second wave were the frame in the TX buffer on FPGA1 that were all discarded due to the error, and were retransmitted.
- The last wave of frames is due to the delay the last acknowledge frame from the 2nd wave needs to reach the FPGA1 and the time it too for the next valid frames to be transmitted.

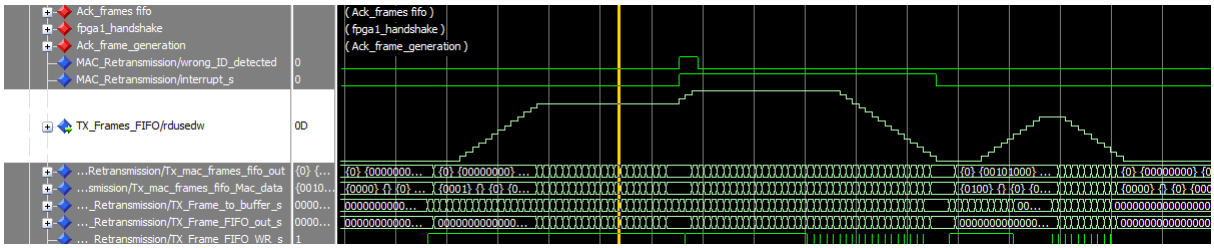


Figure 84: Simulation Error transmission TX Mac signals

The above figure shows the Wrong ID detection from the TX buffer frame, and the interrupt, blocking any frames from the network layer to come out until the fifo is empty.

We can see the FIFO ramping up, while the frames are being transmitted, then as acknowledge frames are received, the fifo stays constantly in its used. Then frames are transmitted back, and only all were successfully received, does the removes the interrupt, and starts normally again.

This simulation validates the retransmission mechanism and physical tests now needs to be made to confirm it.

7. Physical Tests

The physical tests were made on ArriaV SOC boards. As the FPGA design inside the current Beam Wire Scanner is not yet adapted to work on this new platform, the tests made to validate the protocol mechanisms did not have any type of existing services.

The Quartus FPGA implementation had to be made from scratch, and simple services had to be created in order to test the board. Since the aim of the thesis wasn't to create the services themselves, but rather the Application services, Avalon Memory mapped tests remained in simulation, and other simple application services were created:

- A counter application service to pass different increment to a RX counter
- A data traffic generator, to generate 0%, 25%, 50%, 75% or 100% Data traffic.
- An IO Pin service application, to test the IO pin latency.

Furthermore, tests in the Beam Wire Scanner could not yet be done at the time of this writing as the complete design is not yet adapted for the new system.

7.1. ArriaV SOC board Presentation

The ArriaV SoC development board does not have easy pin access, as the next section will show. Due to a lack of time, test validation stayed in SignalTap at the time of this writing, with the intension of using an FMC adaptor to tests many features. The SMA connectors are for differential clocks.

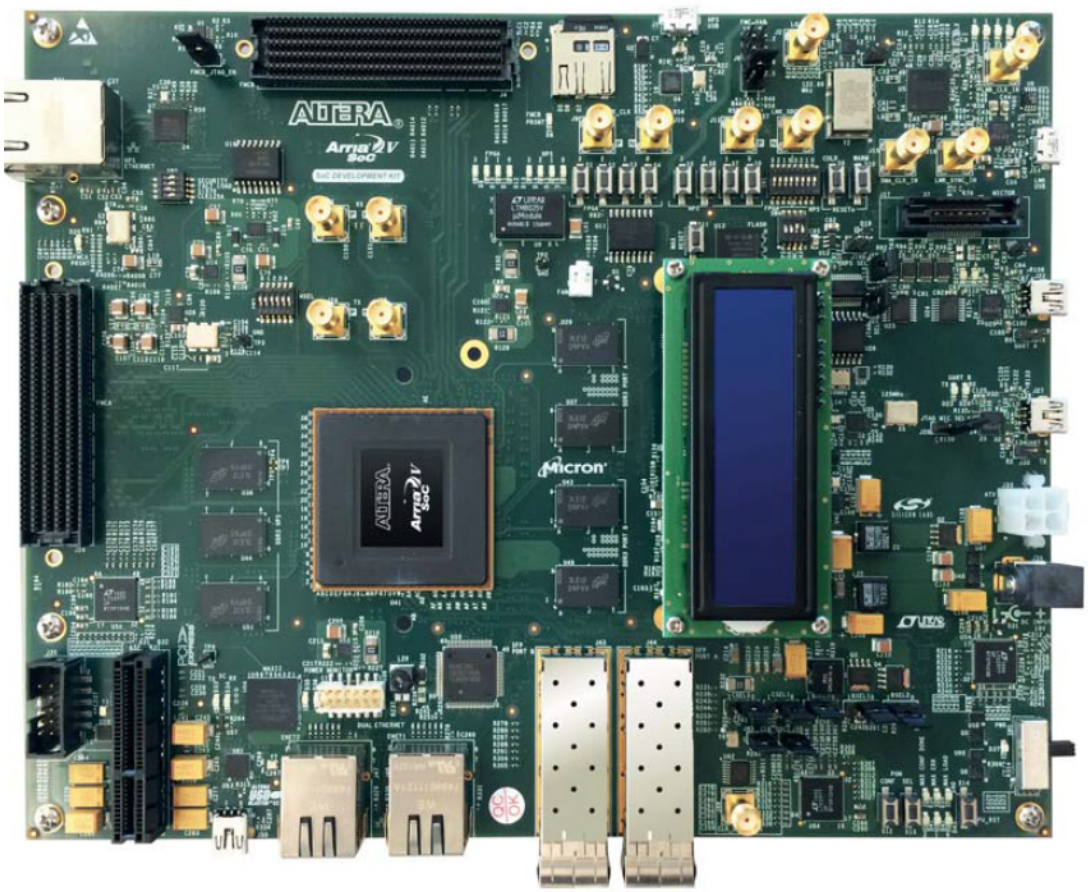


Figure 85: ArriaV Dev kit Board

Only 4 switches and 4 Leds, and 4 deep switches are easy to access inputs outputs.

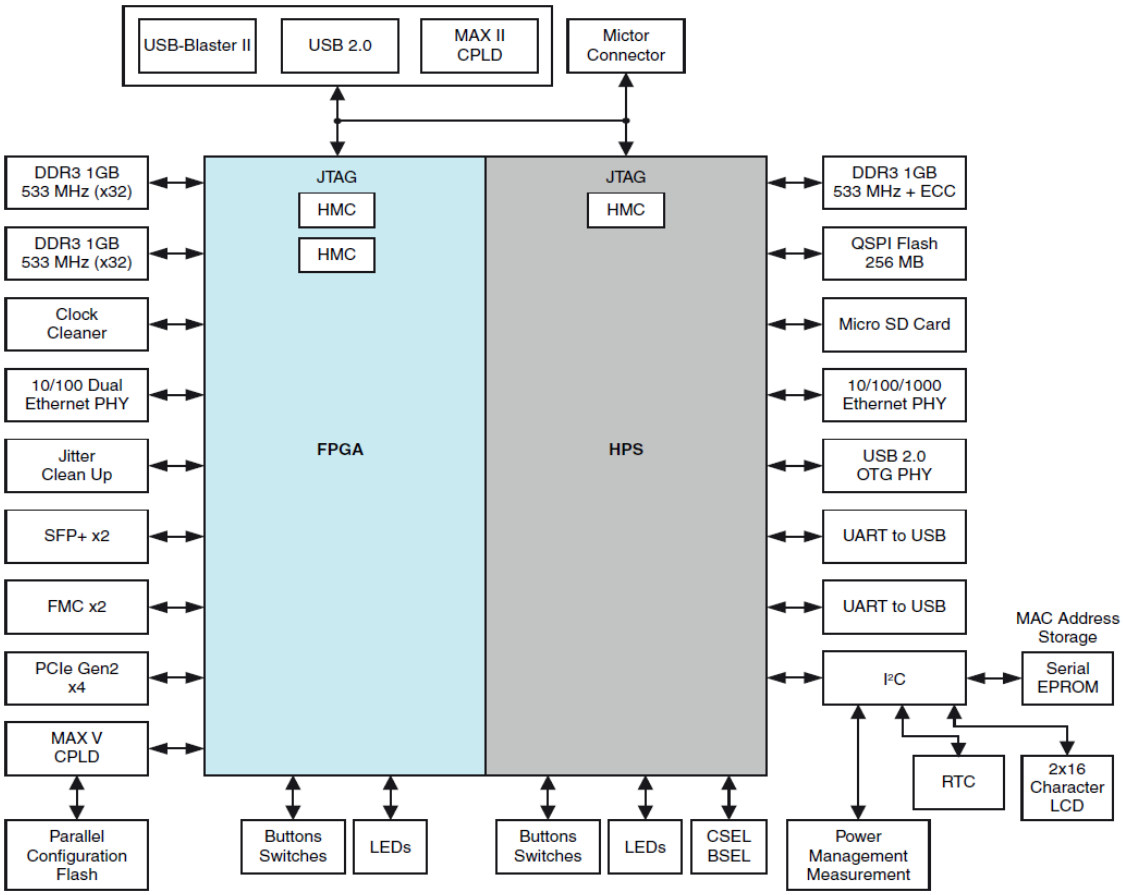


Figure 86: ArriaV development board diagram

7.2. Single Board Tests

Single board transmission can be physically tested in loopback mode. In this mode the fiber optic TX cable is connected back to the RX input.

This mode validates the protocol transactions mechanisms, but its primary drawback is staying in the same clock domain between the TX and the RX side. Therefore, the elastic buffer cannot be fully tested in this mode.

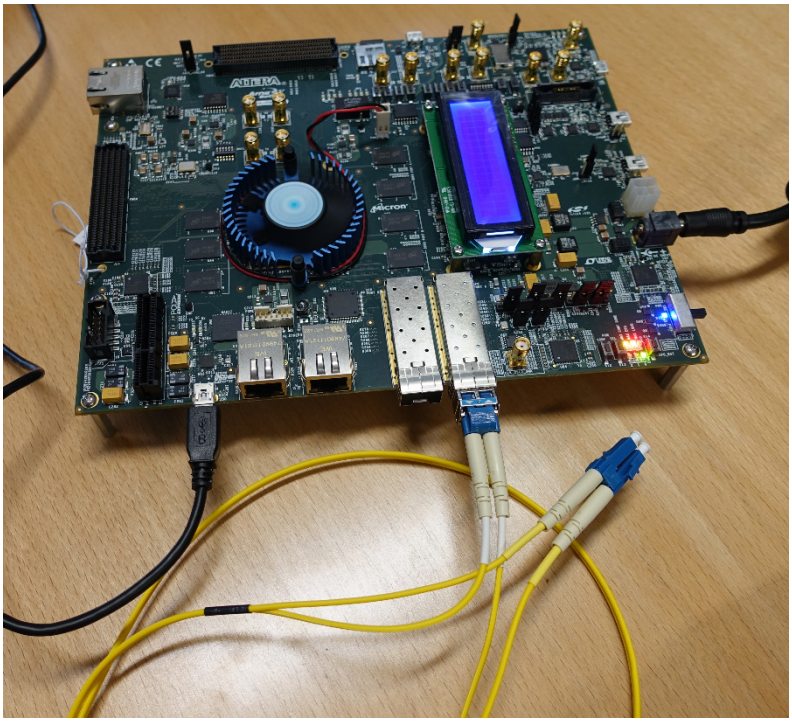


Figure 87: ArriaV in loopback mode

On the picture above, the fiber optic cable was switched to be in loopback mode (linking the 2 white ends together).

First initial test:

The first test used for the single board design was first done with a counter increment each time a button was pressed, showing up on the 4 useable LEDs. The state of the 4 deep switch was sent over the protocol on the TX side. On the RX side, upon reception, that state determined the increment amount of the RX counter. Therefore, when “0001” was present on the deep switch, the RX counter was increasing by 1, when “0010” was present when pressing a button, the RX counter would increment by 2.

This simple transaction test was done before the retransmission acknowledge mechanism was put into place. Refer to the Simple_Counter_Service.vhd service template for more details on the service.

Another test was made to validate the complete acknowledge mechanism when the design was finished. A data generator service was created and used to generate controllable traffic bandwidth. This service generator can generate 25% bandwidth traffic, 50%, 75% and 100% bandwidth utilization.

To control the bandwidth utilization, the 4 deep switches were used. The results can then be seen though signal tap.

An important point to mention in loopback mode with the acknowledge mechanism, since the FPGA will receive its own data, it will have to produce the acknowledge frames as well. It is therefore the same as generating bi-directional traffic between two boards.

The signal tap is sampled with the TX clock, running at 40 Mhz. This means that every clock cycle represents a Data frame.

Loading the line to 25% bandwidth

This test sends a data frame every four frames.

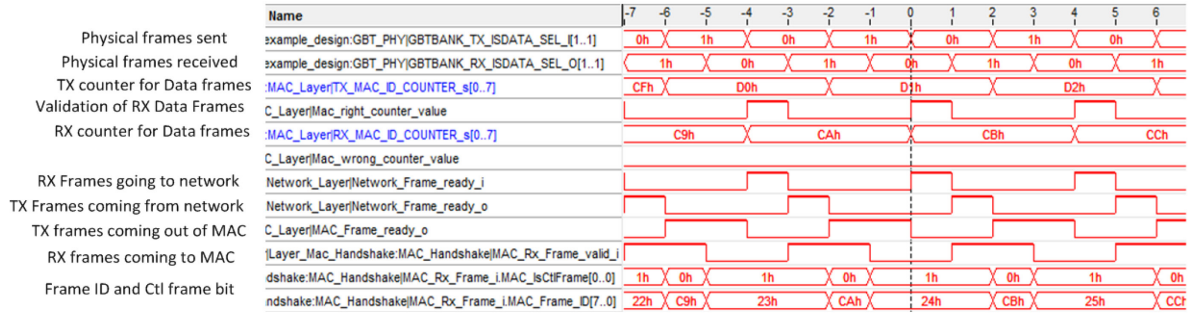


Figure 88: Signal tap Loopback 25% traffic (1)

Looking at the first 2 signals, we can see that 2 frames are sent every 4 frames. The same goes for the reception.

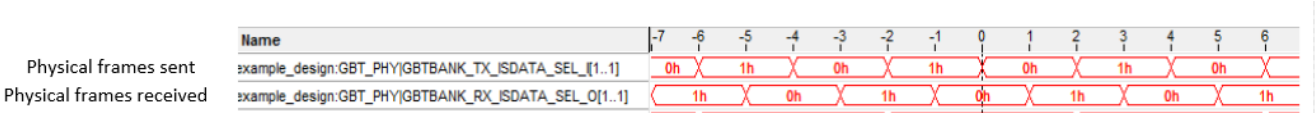


Figure 89: Signal tap Loopback 25% traffic (2)

This is due to the acknowledge frame that also needs to be send.

Next we can see that the frames are validated. The TX counter is incrementing itself once every 4 frames, as it should since the Data traffic is at 25%. The validation of the Rx Data frames can also be seen, with the Rx counter also incremented once every 4 frames.

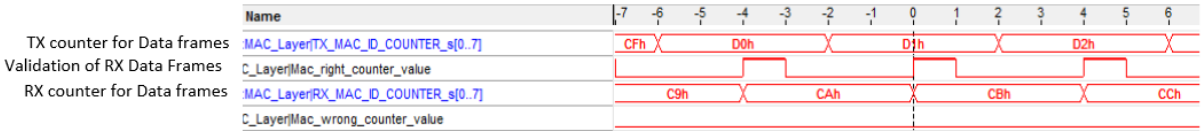


Figure 90: Signal tap Loopback 25% traffic (3)

The last part of the signals shows us where frames are generated from and where they go to.

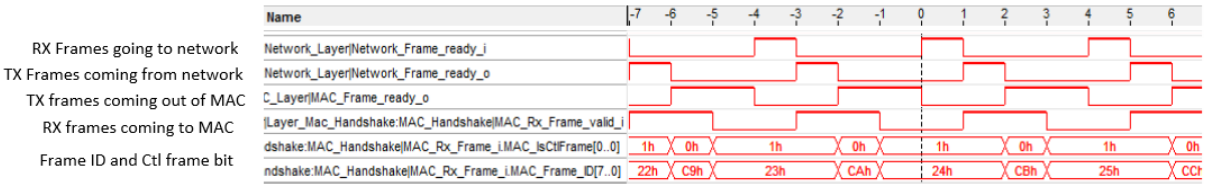


Figure 91: Signal tap Loopback 25% traffic (4)

There is only 1 Rx frame going to the network every 4 frames, as well as coming out. This is the Data frames, leaving and coming back to the application layers before going to the service itself.

The Mac layer however is sending and receiving 2 frames every 4 cycles. Looking into the data, the 2 last signals show the data frame and its ID, as well as the Ctl acknowledge frame with its own ID. The Ctl frame value is hold until the next change.

Loading the line to 50% bandwidth

Looking at a 50% data bandwidth utilization in a loopback mode will seem as a 100%. In fact it is. Since now 2 frames are sent, there is a need for 2 frames acknowledge to be generated. Since the 2 other clock cycles are free of data, the mac layer has no need to concatenate those 2 acknowledges into a single frame.

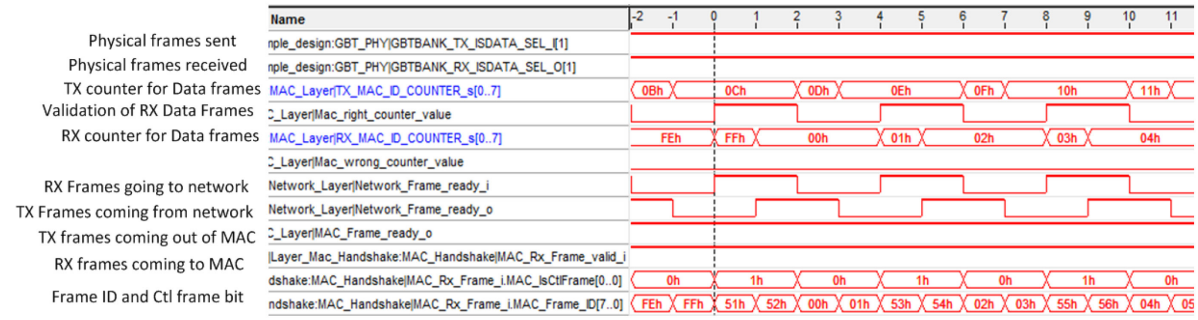


Figure 92: Signal tap Loopback 50% traffic (1)

As expected frames are send and received every clock cycle.

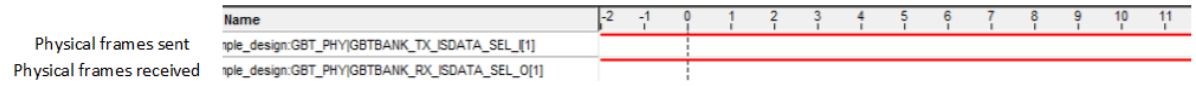


Figure 93: Signal tap Loopback 50% traffic (2)

The two counters for data are now incremented every 2 times per 4 clock cycles as expected. The Mac validation of Data frames is also active 50% of the time.

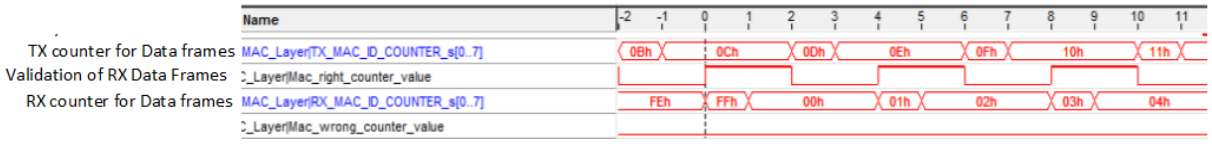


Figure 94: Signal tap Loopback 50% traffic (3)

Last, we can see network frames coming in and out 50% of the time as well, while frames are coming out and in the mac layer 100% of the time. The type of frames can be seen in the last 2 signals: 2 data frames and 2 control frames.

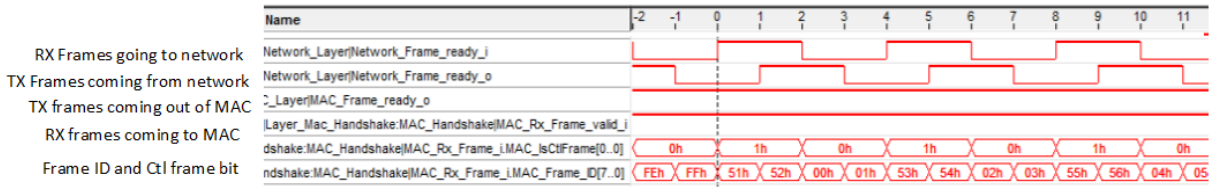


Figure 95: Signal tap Loopback 50% traffic (4)

The link management, even if using 50% of the physical bandwidth, does so without removing any data bandwidth for the user.

Loading the line to 75% bandwidth

This is where things starts to get interesting, as it will validate the concatenation of the acknowledge frames. Now that there is less free frames to send a single acknowledge frame for every frame received, the protocol needs to buffer them and send them on free available slots, when the network layer isn't sending data.

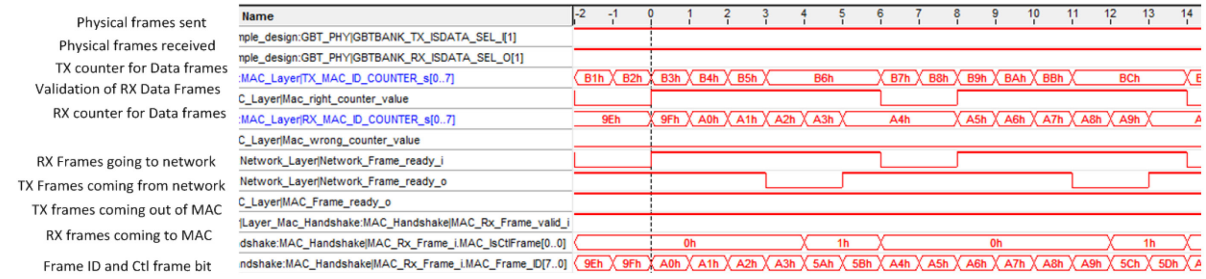


Figure 96: Signal tap Loopback 75% traffic (1)

There is physically a frame sent every clock cycle, like the test before.

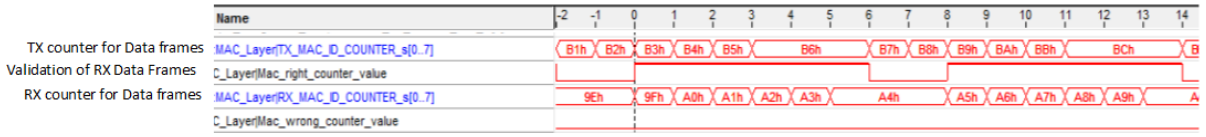


Figure 97: Signal tap Loopback 75% traffic (2)

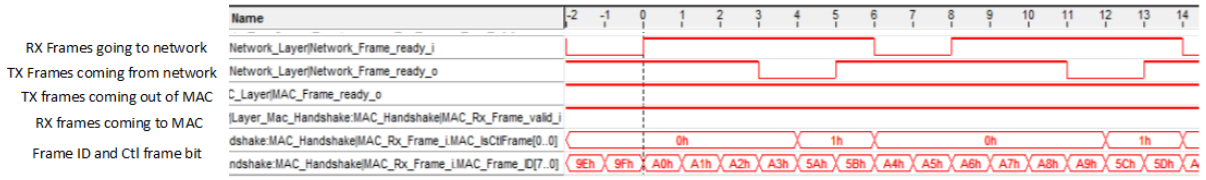


Figure 98: Signal tap Loopback 75% traffic (3)

As shown in the figures above, network frames are now being sent and received 6 out of 8 frames, or 75% as expected by the test.

The mac layer is outputting a frame every cycle, which consist of 6 Data frames and 2 control frames that share the 6 acknowledge frames feedback.

In this test, even if the link management takes 25% of the frames, it is sent on unused frames, thus not decreasing the usable bandwidth in any way.

Loading the line to 100% bandwidth

In terms of bandwidth utilization by the protocol's mechanisms, this is the worst case scenario where 100% of the frames should be data frames, and 100% of the received frames should also be data.

When the service(s) continuously output(s) data frames to transmit, 100% of the bandwidth is used.

In loopback mode however, since frame status also needs to be transmitted from the same physical entity (the same FPGA), data traffic cannot use all the bandwidth. Since the maximum received data buffering in a single control frame is 31 data frames, at least 1 control frame has to be sent every 32 frames. As a result, the real frame bandwidth utilization for data is 96.875% while the link management takes 3.125% of the bandwidth.

This can be seen in the figure above, either with the number of frames coming and going to the network layer, the validated data frames seen by the mac layer or by looking at the data frames vs control frames received.

The same scenario can be achieved with two boards when both boards will try to utilize 100% of the bandwidth for data frames.

The result of this test demonstrates that the link management for retransmissions only impacts the data bandwidth when both the TX and RX streams reach 96.875% data utilization.

7.3. Latency Tests

Latency tests could for the moment only be done with signal tap in a loopback configuration. Further tests are required to correctly measure the latency jitter, especially with the Rx Elastic buffer when using 2 different boards.

For Triggers, Pin to Pin tests with a single board, and then two boards still have to be done to fully characterize the latency and jitter.

The following latency initial tests validate the expectations digitally, with a jitter step of 1 clock cycle by the sampling clock, or 25ns.

Internal layer to layer latency

The major point of the protocol is to deal with latency jitter issues. Layers were not codes in state machines for this regard, to remove corner cases and keep the same latency in the information flow process.

The following internal tests were done with only a single service running, removing the Arbiter priority delay.

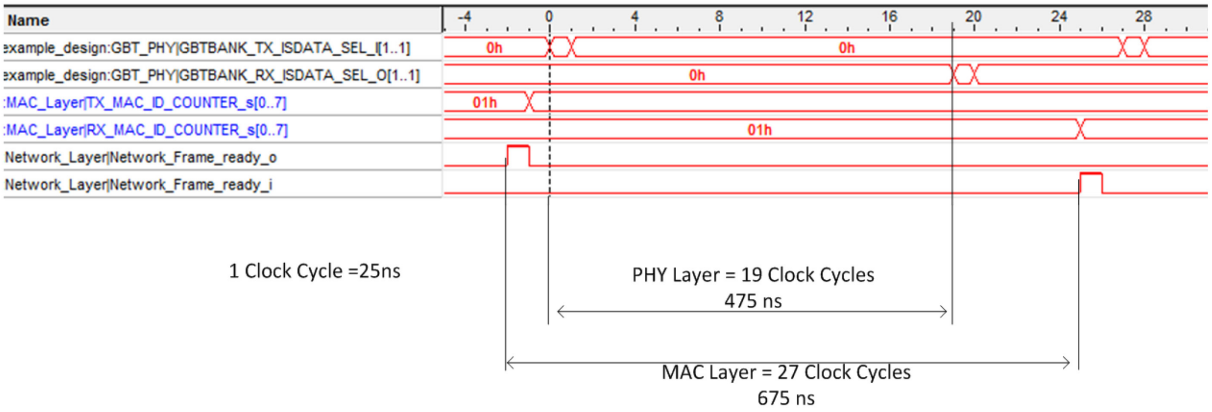


Figure 99: Signal tap Loopback Internal latency (1)

The figure above is showing the latency of the Physical layer (GBT) + propagation medium (fiber optic cable) as well as the latency from the Mac TX to the Mac RX including the Phy adaptation layer.

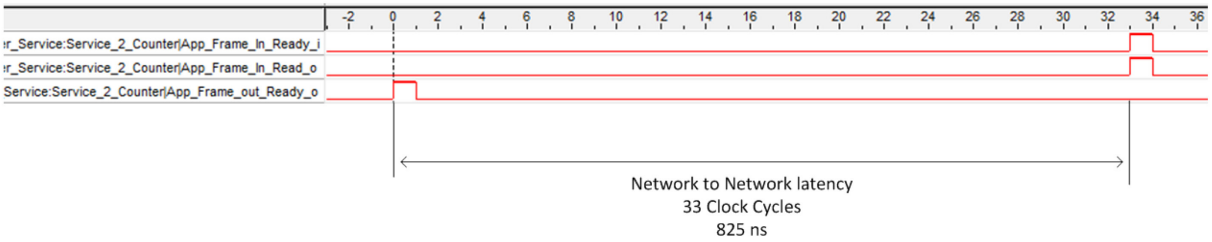


Figure 100: Signal tap Loopback Internal latency (2)

The figure above shows the total latency from a frame going out of the application layer TX side (into Network) to the RX side of the application layer (out of Network).

Those tests were repeated multiple times (50 to 100 times) with signal tap and showed the same values over and over. When the board was reset, the phy2phy values were the same, and Mac2Mac were sometimes 26 or 27 clock cycles. This is due to the Physical Adaptation Rx buffer that contains a time domain crossing FIFO. In loopback mode, since it's the same clock in and clock out, there is no drift happening, but the phases will vary, which can cause the clock cycle difference. Further extensive automated tests have to be done to confirm this latency and the rx buffer drift jitter, but those first results are very stable.

The overall latency results can be seen in the figure below.

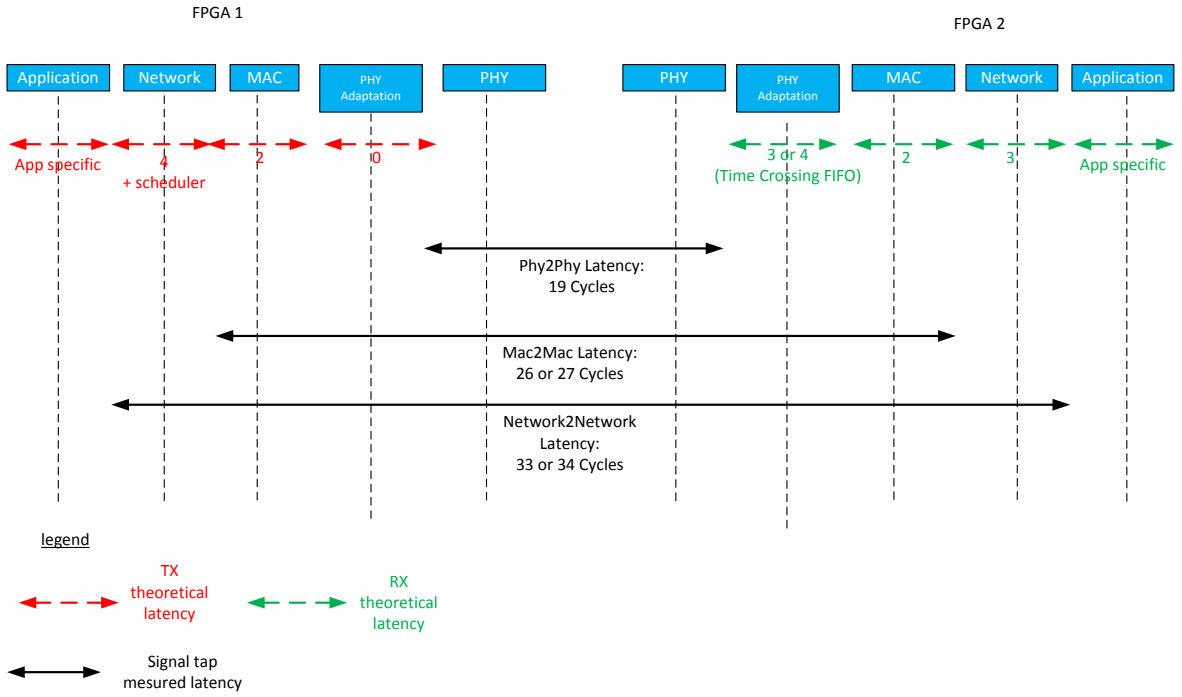


Figure 101: Signal tap Loopback Internal latency summary

The physical results match the theoretical values perfectly. The Phy2Phy latency, dependent on the optic cable length and jitter (due to temperature). Other latencies can be calculated from this point within 1 clock cycle in the loopback mode.

When more services are running at the same time, the scheduler delay will increase and each Y Service will have a maximum network delay of:

$$t_{NetworkLatency}(y) = \sum_{i=1}^{NBservice} [Services\_Priority\_Weight(i)] - Service\_Priority\_Weight(y)$$

With t measured in clock cycles.

IO pin latency

The other type of latency that could be measured is the IO pin latency. This is a special Service 2 Service latency value, as the RX application frames coming out of the network layer are not read as soon as they are ready.

This is to deal with the scheduler traffic latency issue stated in the above section.

The physical test for the IO pin could not be done on actual pins and measured with an oscilloscope in remanence mode at the time this paper was written. Tests remains in Signal Tap. Two buttons were used as the input pins, and two LEDs were used as the output pins. Incoming and outgoing signal states coming from the application service were analyzed in signal tap.

Since 2 pins were used for this test, the Service latency can be calculated as followed:

$$t_{TotalPinLatency} = t_{App\_latency} + t_{NetworkLatency}(y) + t_{Network2NetworkLatency} + RxFrameDelay * t_{App\_latency}$$

With:

$$t_{App_{latency}}[clk\ cycles] = round\left(\frac{\text{AppData}_{Size}[bits]}{NbPin}\right) = \left(\frac{64}{2}\right) = 32$$

$t_{NetworkLatency}(y) = 0\ (single\ service\ test)$

$t_{Network2NetworkLatency} = 33$, calculated above

$RxFrameDelay$= 4, constant set in the application service

Thus, theoretically, the latency seen between a Pin state change should be:

$$t_{TotalPinLatency[Cycles]} = t_{App\_latency} + t_{NetworkLatency}(y) + t_{Network2NetworkLatency} + RxFrameDelay * t_{App_{latency}} = 32 + 0 + 33 + 4 * 32 = 193$$

The following figures shows the signal tap results of IO pin delays in loopback mode with the above conditions.

The delay from the interface_i to interface_o was constant, with either 193 clock cycles or 194 depending on board reset tries (no variation if the system wasn't reset).

This result matches the theoretical number expected. The variation is due the clock alignment in the Rx elastic buffer.

Frames are sent from the network layer once every 32 frames, containing the 2*32 pin states shift registers. The figure also shows 2 frames sent and received from the Mac layer, due to the acknowledge frame.

The first latency tests using signal tap were a success, achieving the expected theoretical results.

At the writing time of this paper, the real pin to pin tests still need to be done. Using an oscilloscope signal generator and measuring the latency of the signal coming from the second board in remanence mode is the next step to see the real latency and jitter, coming from the pin sampling time.

7.4. Dual Board Tests

Dual board tests could be made briefly at the end of the project. Multiple initial tests were done that validated the link between the FPGAs. Those initial tests were also done only with SignalTap at the time of this writing, and need to be further validated with real IOs.

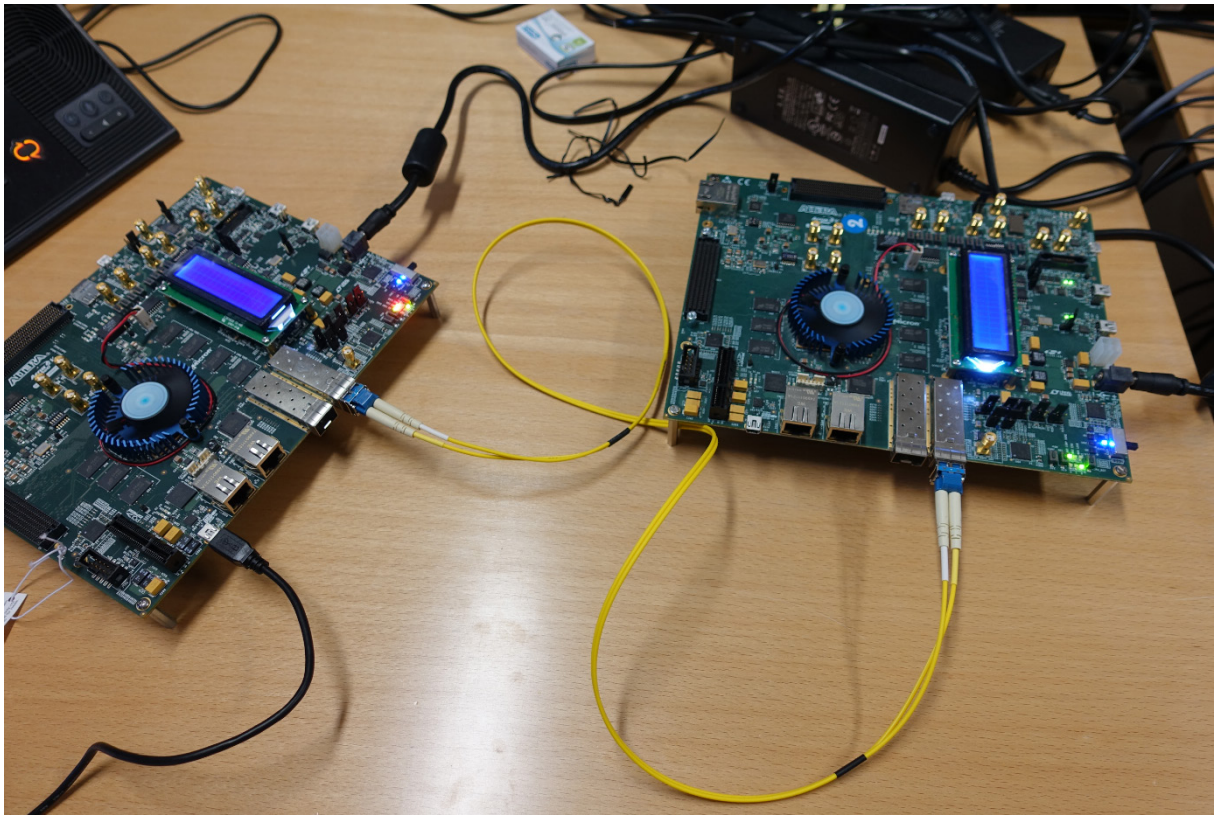


Figure 102: Dual Board test physical setup

IO reproduction with two boards

The IO reproduction test was made using with only a single board transmitting 2 pin status, while the 2nd one was idle and only replying with acknowledge frames.

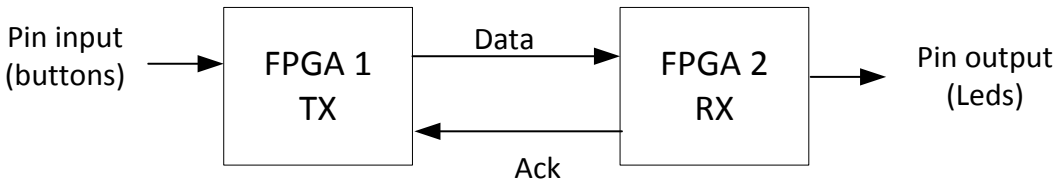


Figure 103: Two board IO pin Test Setup

The TX board is sending the 2 pins states every 32 clock cycles, like in loopback mode.

The difference here is that acknowledge frames are created and send on the other FPGA. As a result, we can see the TX_MAC_ID_COUNTER incrementing itself every 32 frames, while the RX counter stays at the same value.

An RX Control frame is received for every frame sent.

The below figure highlights that Frames come out of the Network layer but none come in.

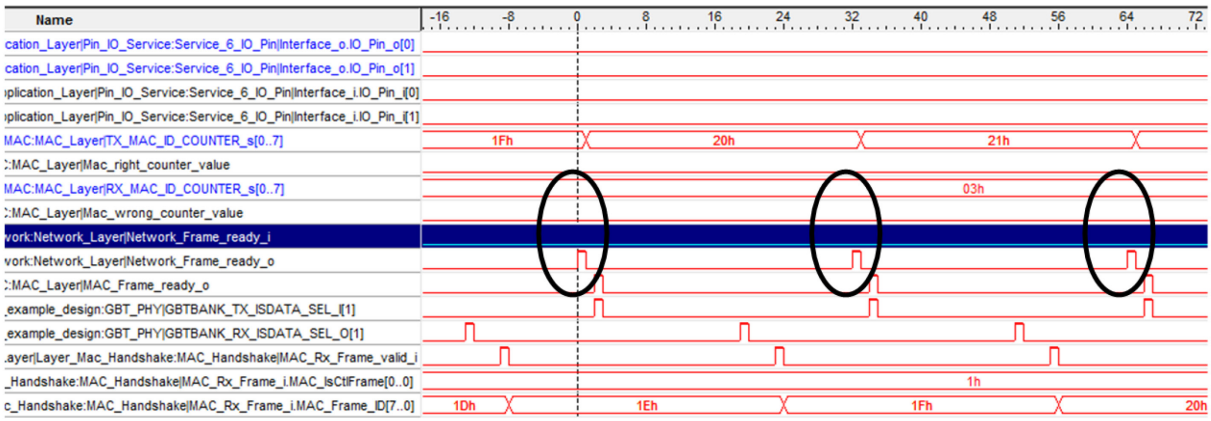


Figure 104: Two board IO pin result (1)

On the RX side, Data frames are received and passed to the network Layer, but none are sent. The below figure shows the frames received, when no frames were sent.

We can also see the RX ID counter increasing, while the TX counter is constant.

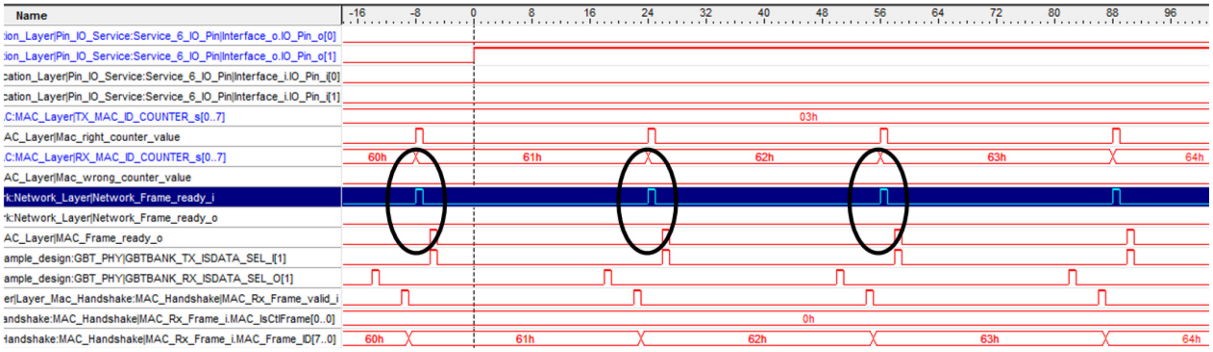


Figure 105: Two board IO pin result (2)

From the available data of this test, This IO pin reproduction was fully functional. Pressing the button on the 1st board could be seen on the LED of the other. The SignalTap analysis was also as what could be expected. Further tests will be done to measure the exact response time of the delay, and the reproduction of those tests over a long period of time with an oscilloscope in persistence mode.

Traffic generation with IO Pins reproduction

This test shows a scenario where the FPGA 1 will send the state of the IO pins and traffic generation for the FPGA 2.

Two services transmitting information, while the FPGA 2 only acknowledge the frames, in order to keep the signaltap readable.

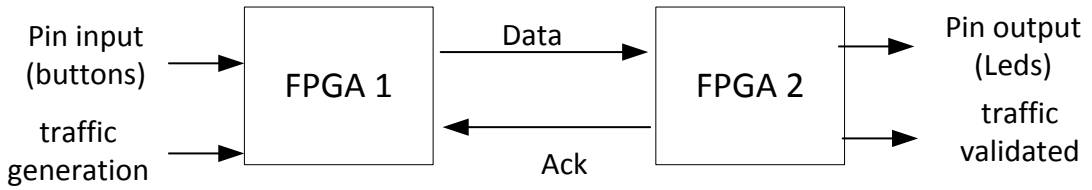


Figure 106: Two board IO pin + traffic Test Setup

On the transmission side, the arbiter's job can be seen, with requests coming from 2 different services. A frame every 4 clock cycles is been transmitted, and 1 more every 32, representing the 25% traffic generator and the IO Pin service respectively.

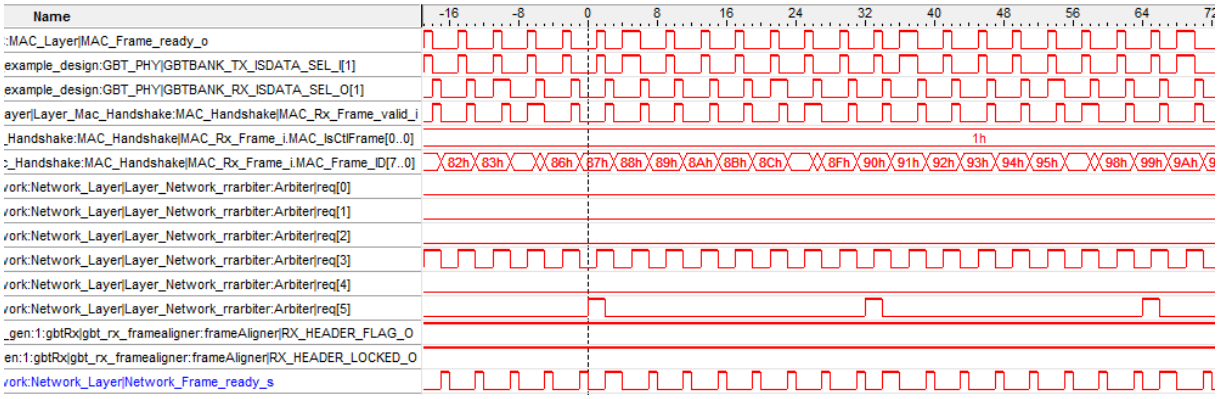


Figure 107: Two board IO pin + traffic result (1)

The RX side is receiving the information, and passing it to the correct Service.

The next figure shows tests when traffic is being increased to 50% with the IO PIN

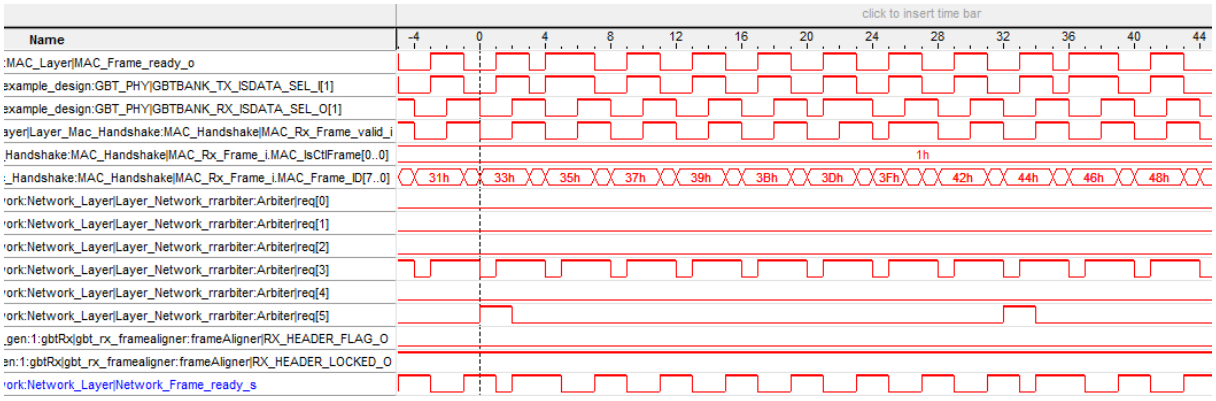


Figure 108: Two board IO pin + traffic result (1)

The request always stays 1 more clock cycle before it is being served, hence the 3 clock cycles active every 4.

Looking at the total frames being sent, 2 frames are sent every 4 frames now, with still the IO pin frame being transfer once every 32 frames (can be seen on the first signal, MAC_Frame_ready_o).

8. Planning

The planning has to take into account the 4 major steps: specification, development, testing and documenting. Holidays break should also be considered, as CERN will be closed during 2 weeks on Christmas.

The delivery of the ArriaV dev kit also has to be considered, and if the GBT should be chosen as PHY layer, its release for ArriaV devices.

8.1. Projected Planning

The planning can be seen in the annex table, with specifications in green, development in blue, testing in red and documenting in purple.

The following elements should be dealt and answered:

- Analyzing the different services, and their critical aspects related to the protocol.
- Have the CERN answers for the questions and limitations
- Define a physical layer approach
- Make a general schematic plan of the protocol's implementation + services
- Get the PHY layer working
- Get the MAC layer working
- Have a simple service and protocol implementation working
- Get all services working

The projected planning made at the start of this thesis can be found in annex 6.

8.2. Followed Planning

The followed planning diverged slightly from the projected planning. The implementation of the retransmission transactions took longer than initially expected. More time was passed on this feature, as it wasn't working properly in simulation. The physical testing of this retransmission had thus to be delayed. The testing of the acknowledge mechanism was tested on the board successfully, but without creating transmission errors. Those transmission errors scenarios were only tested successfully in simulation. Hence, only 50% of this mechanism was able to be fully tested physically on the board.

The following planning of this thesis can be found in annex 7.

9. Protocol performances summary

Here is a summary list of the protocol performances and advantages:

- ☑ Bi-directional communication protocol with independent TX and RX channels.
- ☑ Based on CERN GBT physical layer that provides 3.36Gbps useable data bandwidth.
- ☑ Provides a maximum of 2.84Gbps usable Data bandwidth, with a low protocol overhead of 15.5%.
- ☑ Error detection and transparent, automatic retransmission capabilities.
- ☑ Protocol handshake for reliable communication establishment.
- ☑ Link management with frames transmission status feedback for reliable communication.
- ☑ Very low link management bandwidth needs, taking less than 4% bandwidth when dealing with a worst case scenario of 100% bi-directional traffic communication.
- ☑ Possibility of adapting bandwidth allocation per Service.
- ☑ Can send or receive Low jitter latency triggers.
- ☑ Modular: Built on independent layers structure, adaptive to other physical layers.
- ☑ Generic: Generic code structure, easy to change and maintain.

Services advantages:

- ☑ Easy to use and implement in an existing system.
- ☑ Transparent communication. Existing Service Interfaces do not need modifications to function with the protocol.
- ☑ Support up to 16 different Application services, communicating simultaneously.
- ☑ Transparent bandwidth management. Services are independents and can function at their own speed rates.
- ☑ Minimum bandwidth allocation assurance. Minimum Bandwidth allocation can be setup individually for each service, allowing services with high bandwidth needs more priority.
- ☑ Transparent reliable transmissions. Error detections or retransmissions are transparent for the services. The protocol handles frames retransmissions while locally maintaining service's interfaces bandwidth needs.
- ☑ Support of existing communication interfaces, such as *Avalon MM Master* and *Slave*.
- ☑ Support heterogeneous communication, such as Avalon to Wishbone.
- ☑ Support bi-directional communication accesses, such as an *Avalon Master Read Request*.
- ☑ Support Streaming interfaces.
- ☑ Support IO Pins reproduction with fixed delays by streaming signals' states.
- ☑ Changing, adding or removing a service can be done fast and with ease, thanks to the generic code structure.

10. Archived protocol specification

Here is the compared list of the achieved protocol performances regarding the specification.

Required specifications:

- Data integrity
  - ☑ Error detection, correction and/or retransmission.
  - ☑ Notification

Data integrity is assured by Error detection, and frames retransmission when correction cannot be achieved. Notifications can be made to the user when a none-valid frame is received.

- Event transport
  - ☑ Trigger and IO port replication between the 2 ends
  - ☑ Link latency jitter <0.1us

Event transport have been properly implemented, keeping the link latency jitter below the required 0.1us.

- Transparent interconnect of SoC bus
  - ☑ Interconnection of internal FPGA bus transparently (Memory Mapping)
  - ☑ Data blocks transfer between FPGA (2 directions)

The transparent interconnection of the 2 FPGAs was fully simulated, with different services templates at disposal, including Avalon MM Master and Slave. Both Write and Read access are supported to transfer information on both sides. Once the rest of the design can be added to the protocol, those interface will be able to be physically tested.

- Streaming links
  - ☑ Interconnection of internal FPGA bus transparently (Stream interfaces: 1..N @ >= 1Mbit/s)
  - ☑ Transparent connections for streaming mechanism

Transparent Streaming links, was also fully integrated. A service template is at disposal.

All of the required specifications were met with most of them already physically validated. Once the current working services are updated to work on the new Wire Scanner, more physical tests will be made to validate the simulation results. However, since Application services are already working, they validate the protocol working mechanisms. What remains is only to test the specific application services themselves, rather than the protocol as a whole.

Optional specifications:

- Virtual JTAG (investigation)

  JTAG chain over the link: Use of vendor tool for logic analyser (SignalTap), memory/register content modification, probing by using Boundary Scan (tbc)
- Data and error statistics
- Automatic transmission time evaluation
- None-volatile memory management

  External flash management over the link. Read/write flash of program or settings.

After the main features were implemented and tested, there was no remaining time to implement those optional specifications. Since the retransmission mechanism took longer than initially expect to be fully operational, the efforts and time spend was chosen to be spend on optimizing this mechanism, rather than those optional features.

The transmission time evaluation is however currently done manually. It is depended on the fiber optic cable length used, and the protocol latency. It can for the moment be seen with signal tap. More information is also needed for an automatic task, especially the starting and ending points, and the method of access.

The none-volatile memory management needs more details regarding the external flash that will be used in the final VFC board of the Wire Scanner. But since the read and write mechanisms are already working for the Avalon Memory Mapped Master, creating an application service that will handle the flash memory management is half way there. Either a specific service can be created to handle this mechanism, or this process can be done through the Avalon Bus, with a higher software layer.

11. Difficulty encountered

Physical layer Implementation

The physical layer choice had a big risk factor in its decision making. CERN strongly wanted to use the GBT, and a lot of time was initially spend to understand its mechanisms, to evaluate the feasibility in its use for this project. Since the ArriaV is not officially supported by the GBT official release, a none official release was provided to be able to test and simulate it.

For the implementation, one of the changes that had to be done in the code provided was adding a PLL to create a 40Mhz recovered Clock, based on the 120Mhz Recovered Clock.

That bypasses the Matching Rate FIFO problem in the GBT (no need to change any files) so that an extra FIFO can be put after the GBT. (PHY Adaptation LAYER), passing from 40Mhz Recovered Clock to the 40Mhz TX generated clock.

The final PHY layer simulation takes 2 GBT design example and make them communicate though the serial data lines. A GBT_VVT was created to send out specific frames (compare to a counter process) inside the UVVM framework.

Wanted to simulate 2 different time domains for the 2 boards to test the RX elastic buffer, 2 different clock period were given to the simulation boards. Errors were then seen in the GBT itself, where the frame aligner was delocking itself from the GBT header that marks a frame start.

It turned out that it wasn't the GBT's fault, as Altera IP simulation doesn't simulate CDR clock implementation.

On real devices, the write side of the rate match FIFO is clocked by the RX parallel recovered clock, and the read side of the rate match FIFO is clocked by the TX parallel clock. But in the simulation model, both write and read clocks are synchronous to the TX parallel clock due to the limitation of the CDR modeling. [14]

The clock Parallel Clock Recovery frequency is the same as the TX parallel clock frequency, even though the serial Data line has is at a different frequency, which caused the GBT synchronization problems.

The following figures shows a simulation model Hack, but doing so breaks the clock topology of the PHY IP, and therefore was not attempted.

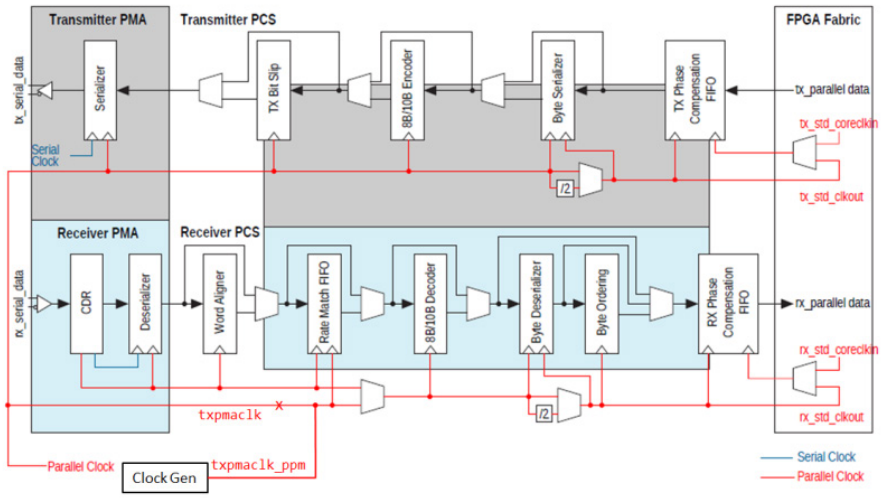


Figure 109 : Altera CDR simulation limitation

This limitation means that it's not possible to simulate 2 different Data rates to look how the elastic fifo would react.

The simulation framework is thus using the same frequency on both boards.

Retransmission mechanism

This mechanism took longer to validate that initially expected. The initial analysis was too simple.

- It was using too much RX bandwidth

Having a GBT frame Acknowledge for every frame received was taking 50% of the bandwidth for nothing. This was not acceptable for a bi-directional protocol. Transmitting a single frame with fixed amount of frame status was not an option either, as it meant no status would be returned as long as that amount was not reached. As implementing timeout counters to prevent it would complexity the protocol, it was not a good solution either. It would also not be able to use the free bandwidth at disposal to its full advantage.

- Compacting the acknowledge frames also meant having special ID headers to verify that the right frames were validated on the TX side.

Since the protocol can send any amount of status per frame (1 to 32 actually), the validation process had to make sure that the first status in the frame was matching the frame in the TX buffer FIFO, and what to do when they don't match. How many frames were missing, 1, 2 , 100?

- At first, the design was thought to deal with those missing frames per service, meaning that if an error occurred on a frame, only that frame's service would be affected while the rest of the services would continue functioning normally.

This was great in theory, but much harder and complex to implement, especially in the TX Buffer side. How to validate different frames from different services and pass this information in a single GBT frame without taking too much bandwidth?

Furthermore, since a frame can only be seen as missing when the RX side receives the next one and IDs don't match, the situation can also be the following:

- A tx transmission of Service 1, 2, 2, 1, 2, 2
- The 1st frame is not received.
- The RX acknowledge will return valid, valid and not valid for the 3rd
- The tx side will remove the 1st and 2nd frame and resend the 3rd.
- The 1st frame not received is lost.

This was thus implying that the TX buffer should contain multiple FIFOs for each service. Since the FIFO depth needs to be large enough to contain a continuous streaming of packets until the first acknowledge is received, and since all packets physically on the optical link for that amount of time could be from the same service, the memory requirements are exploding.

Overall, a step backward had to be taken in this mechanism, to limit the complexity behind it. The implemented design has a single ID counter and disregards any frames not matching the wanted ID. A missing frame will thus impact all services bandwidth for the duration of the retransmission needed to resend those frames.

Physical tests

Not many connections were present on the ArriaV dev kit board that could be used to quickly validate service integration. The only real pin connectors are the FMC, but not all FMC pins are available for the FPGA on the ArriaV dev kit board.

The initial idea was to buy a HDMI or DVI Input/Output FMC Module to transport a high speed video transmission.

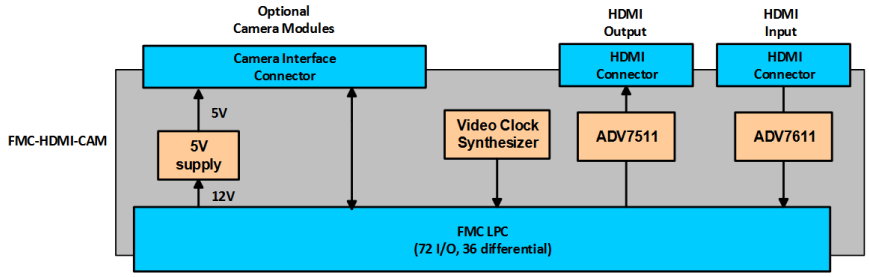


Figure 110: FMC-HDMI board bloc diagram

However, the pins not physically connected to the FPGA were needed by those modules, rendering those interfaces incompatible.

Due to time restrictions, doing a custom electronic board with ADCs was not attempted either.

Simple services had to be designed for the physical tests, and the deep switch, buttons and LEDs were used.

12. Improvements

12.1. Unresolved known issues

- The RX buffer implementation will still overflow with 100% traffic generation, as no IDLE frames are forced by the protocol.
    - A watchdog timer could be implemented.
    - Could put RX data frames FIFO and RX control frames FIFO

- If an ack control frame is corrupted and not seen by the TX buffer, frames will be send again, no matter the status. (Implication is rather big with unidirectional communications, implying a lot of Ack frames.)
    - implement a retransmission mechanism max retry, before deleting the frames from the buffer
    - Validate retries when the Mac_ID received is below the current RX counter
- The Avalon Slave service (VVC) for simulation doesn't have any memory mapped to return when a read request is initiated. It always returns the same value to validate the protocol transaction.

12.2. New Improvements

Here is a none-exhaustive list of the work that still need to be done:

- Increase the number of Application services type.
- Optional requirements that could not be implemented yet
    - Link statistics
    - Virtual JTAG, external memory management
- Simulation
    - More simulation regarding Trigger and IO Pins simulations and verifications
    - Random Error injector module for automatic retransmission tests
- More physical testing, using the FMC adaptors
    - Oscilloscope remanence for pin jitter tracking
    - Trigger latency
- Transmission error physical testing
    - With attenuators
    - BER profiling
- Implementing the design with the BWS code
    - Fully characterize the protocol

Most of those improvements or tests will be done in the following months, after the written of this paper at CERN, with a contract extension.

13. Conclusion

When judging communication protocols, we tend to focus on their performances or their features rather than their usability. But a good protocol is not necessary the best one in terms of performance. It is usually the easiest to implement and integrate with existing elements. Keeping that in mind, even if the specifications required for this thesis were almost all purely technical, a lot of effort was put into creating a protocol that would not only satisfy those technical needs, but also one that users would value in terms of integration and transparency.

The protocol developed during this thesis has achieved both successfully. Its modular and generic structure makes its use transparent and integration effortless, while meeting all of CERN technical requirements for the Beam Wire Scanner. As such, the protocol acts as a true virtual link between two FPGAs, assuring error-free transparent communication between any existing user elements, without modifying them.

With bi-directional capabilities, it manages data transmission integrity, latency and bandwidth needs for multiple services simultaneously, by implementing data reception acknowledge and retransmission mechanisms as well as a weighted Round-Robin scheduler. It offers the possibility to send and receive triggers with less than 50ns jitter while at the same time assuring a total data throughput of more than 2600Mbits/s. Data streaming services and services with communication bursts, requiring high temporary bandwidth, can coexist thanks to the weighted priority scheduler.

The VHDL simulation test bench developed to test this protocol and its integration with existing elements, offers a complete environment based on a UVVM framework to quickly and easily create corner case scenarios. The simulation process duration for a complete system validation is thus greatly reduced, providing better reliable designs.

The protocol was physically implemented in the ArriaV SoC development boards, and initial tests successfully validated the expected results. More stressful extensive test validations still need to be done, in order to fully characterize the link before its integration with the existing code.

In conclusion, the protocol developed during this thesis pushes CERN GBT standards even further, not only providing users with a physical layer, but extending it to a full, high performance transparent communication protocol. Its simple integration will, I'm sure, motivate other CERN's departments to integrate it into their future or even actual systems, as will the Beam instrumental department do for the Beam Wire Scanner.

Geneva, February 9th, 2017.

Cédric Vulliez

14. Bibliography

[1] White Rabbit Cern project, http://www.ohwr.org/projects/white-rabbit

[2] 2.5 and 5GBASE-T, https://en.wikipedia.org/wiki/2.5GBASE-T_and_5GBASE-T

[3] The GBT-FPGA team, (2016) : GBT FPGA User Guide, version 1.4,

[4] J. Emery et al. (2014), A fast and accurate wire scanner instrument for the CERN accelerators to cope with high environmental constraints and increasing availability demand, in 2014 IEEE Multi-conference on Systems and Control, Nice France October 8–10 2014.

[5] Wikipedia, Optical fiber, Propagation speed and delay https://en.wikipedia.org/wiki/Optical_fiber_cable#Propagation_speed_and_delay

[6] Wikipedia, OSI Model, https://en.wikipedia.org/wiki/OSI_model

[7] OSI Model Image, https://www.pinterest.com/explore/osi-model/

[8] RTLery, (2016), Weighted round robin multi fifo queuing, http://rtlery.com/components/weighted-round-robin-multi-fifo-queuing

[9] GIT Round Robin Arbiter, https://git.krll.de/public/snippets/src/master/vhdl/rrarbiter.vhd

[10] Wikipedia, Communication protocol, https://en.wikipedia.org/wiki/Communications_protocol

[11] CERN, Super proton synchrotron, https://home.cern/about/accelerators/super-proton-synchrotron

[12] CERN, Super proton synchrotron Manuel, https://op-webtools.web.cern.ch/vistar/Doc/SPS1.pdf

[13] Bitvis, UVVM Framework, http://www.bitvis.no/products/uvvm-vvc-framework/

[14] alterawiki How to simulate ArriaV native PHY IP with Rate Match FIFO http://www.alterawiki.com/wiki/Simulate_Arria_V_Native_PHY_IP_Rate_Match_FIFO_under_clock_compensation_condition

[15] Cortina Systems and Cisco Systems, (2008), Interlaken Protocol Definition v1.2

15. Figures

16. Annexes

[1] CERN List of goals for the Technical Student program, listofgoals_wire-scanner_link.docx

[2] Beam Wire Scanner (BWS) serial link requirements and architecture, J. Emery, BWS-SerialLink-JEmery.pdf

[3] BWS motion control electronics, J. Emery & P. Andersson, BWS-Intelligent-Drive-Electronics-TB.pdf

[4] Proposition de projet de Master, J. Emery, CERNBeamWireScanner_time_critical_protocol_v2.docx

[5] Transaction Level Modelling en VHDL-2008 grâce à UVVM, Cédric Vulliez, 2016

[6] Projected Thesis Planning, Cédric Vulliez

[7] Followed Thesis Planning, Cédric Vulliez