Efficient Protocols

Inauguraldissertation

zur Erlangung des akademischen Grades

eines Doktors der Naturwissenschaften

der Universität Mannheim

vorgelegt von

Alexander Giese
(Diplom-Informatiker der Technischen Informatik)

aus Heidelberg

Mannheim, 2015

Dekan:	Professor Dr. Heinz Jürgen Müller, Universität Mannheim
Referent:	Professor Dr. Wolfgang Effelsberg, Universität Mannheim
Korreferent:	Professor Dr. Ulrich Brüning, Universität Heidelberg

Tag der mündlichen Prüfung: 03. März 2016

Für meine Eltern

Abstract

The increasing demand for more and more computing power causes steady advancements of High Performance Computing (HPC) systems. The more powerful these systems will be in the future the further the number of processing units increases. A particularly important point in this context is the latency of the communication among those units, which significantly increases by the distance between two communication partners. One approach to positively influence the latency behavior is optimizing the underlying protocol structures in the overall system. Nowadays, different protocols are used for different communication distances. The latency can be improved by changing the protocol structure with two approaches. On the one hand, the used protocols can be changed to optimize the latency. On the other hand, the protocol structure can be unified. Thus, time-consuming protocol translations can be eliminated. In order to achieve this, a completely new protocol is required which unifies all features of the different protocol levels without compromising an efficient implementation.

This work is dedicated to the design of the new Unified Layer Protocol (ULP) providing a unified communication scheme which allows communication among all processing units at different levels of an HPC system. Initially, the main features of general protocols are analyzed in detail. Further, properties used by modern protocols use are introduced and their function is explained. The two protocols that are deemed most relevant, HyperTransport (HT) and Peripheral Component Interconnect Express (PCIe), are analyzed in detail regarding to the previously specified aspects. The insight gained through this analysis is incorporated into the development of the ULP. During the development process, first the structure of the ULP is defined and various parameters are determined. Special attention is turned on the feasibility in hardware and the scalability for large systems. The following comparison with HT and PCIe shows that the newly developed ULP usually provides superior performance, even when the effective communication distance moves close to the processor.

Further work is dedicated to the hardware development which first gave the inspiration for the development of the ULP. The insights gained during the development of the ULP were integrated into the hardware.

The results show that the ULP fulfills the demands for a protocol used in the field of HPC. This is achieved for both, the processor-near communication, as well as for the communication among different nodes. With the ULP the need for time and energy-consuming protocol conversions is eliminated, while the feasibility in hardware is obtained.

Zusammenfassung

Die steigende Nachfrage nach immer mehr Rechenleistung verursacht eine stetige Weiterentwicklung von High Performance Computing (HPC) Systemen. Je leistungsfähiger diese Systeme werden desto weiter steigt die Anzahl der Recheneinheiten. Ein besonders wichtiger Punkt in diesem Zusammenhang ist die Latenz der Kommunikation dieser Einheiten, welche maßgeblich durch die steigende Distanz zwischen zwei Kommunikationspartnern negativ beeinflusst wird. Ein Ansatzpunkt um das Latenzverhalten positiv zu beeinflussen ist die Optimierung der zugrunde liegenden Protokollstrukturen im Gesamtsystem. Heutzutage werden für verschiedene Kommunikationsdistanzen unterschiedliche Protokolle genutzt. Für die Optimierung bieten sich zwei Ansätze an. Einerseits können die Protokolle selbst optimiert werden um die Latenz zu reduzieren. Andererseits kann die Protokollstruktur vereinheitlicht werden und somit die zeitaufwändige Protokollübersetzung eingespart werden. Um dies zu erreichen wird ein komplett neuartiges Protokoll benötigt welches sowohl die nötigen Eigenschaften der verschiedenen Ebenen vereint und trotz allem umsetzbar bleibt.

Diese Arbeit befasst sich mit dem Entwurf des neuen Unified Layer Protocols (ULP) welches in der Lage ist die Kommunikation zwischen Recheneinheiten in einem HPC System zu vereinheitlichen. Um dies realisieren zu können wurden zunächst die Hauptmerkmale die Protokolle allgemein aufweisen genauer untersucht. Weiter wurden die Eigenschaften die moderne Protokolle besitzen eingeführt und deren Funktionsweisen erläutert. Die zwei für am relevantesten erachteten Protokolle, nämlich HyperTransport (HT) und Peripheral Component Interconnect Express (PCIe), wurden detailliert auf diese Aspekte hin analysiert. Die daraus gewonnenen Erkenntnisse sind in die folgende Entwicklung des ULP eingeflossen. Bei der Entwicklung wurde zuerst die Struktur des ULP festgelegt und verschiedene Parameter bestimmt. Besonderes Augenmerk wurde hierbei auf die Umsetzbarkeit in Hardware und die Skalierbarkeit für große Systeme gelegt. Der nachfolgende Vergleich mit HT und PCIe zeigt, dass das eigens entwickelte ULP schon im Prozessor nahmen Einsatzgebiet meist bessere Performanz bietet als die Vergleichsprotokolle.

Weiter wird auf die Hardwareentwicklung während der Dissertation eingegangen, welche den Anstoß für die Entwicklung von ULP lieferten. Die erarbeiteten Konzepte von ULP flossen mit in die weitere Entwicklung der Hardware ein.

Die Ergebnisse zeigen das ULP die Ansprüche an ein Protokoll, welches im HPC-Bereich eingesetzt wird, erfüllt. Sowohl für die Prozessor nahe Kommunikation als auch für die Kommunikation zwischen verschiedenen Knoten über weite Distanzen. Dabei wird der Einsatz von zeit- und energieintensiven Protokollumsetzungen eliminiert, während die Umsetzbarkeit in Hardware effizient möglich ist.

Acknowledgments

I simply would like to thank all the people who supported me and did not stop believing. ;-)

Table of Content

Chapter 1: Introduction

1.1 Motivation

In nearly every field that involves data processing there is an increasing need for more and efficient compute power. Due to excessive computational expense for objectives as weather forecast or crash test simulation developing High Performance Computing (HPC) systems become more and more relevant. HPC systems are specialized computer systems with a large amount of processing power, memory, and often an interconnection network that is custom built for HPC. The main goal of HPC is to increase compute power and to provide scalable solutions. Thus, it focuses on delivering the required performance to solve these challenging tasks.

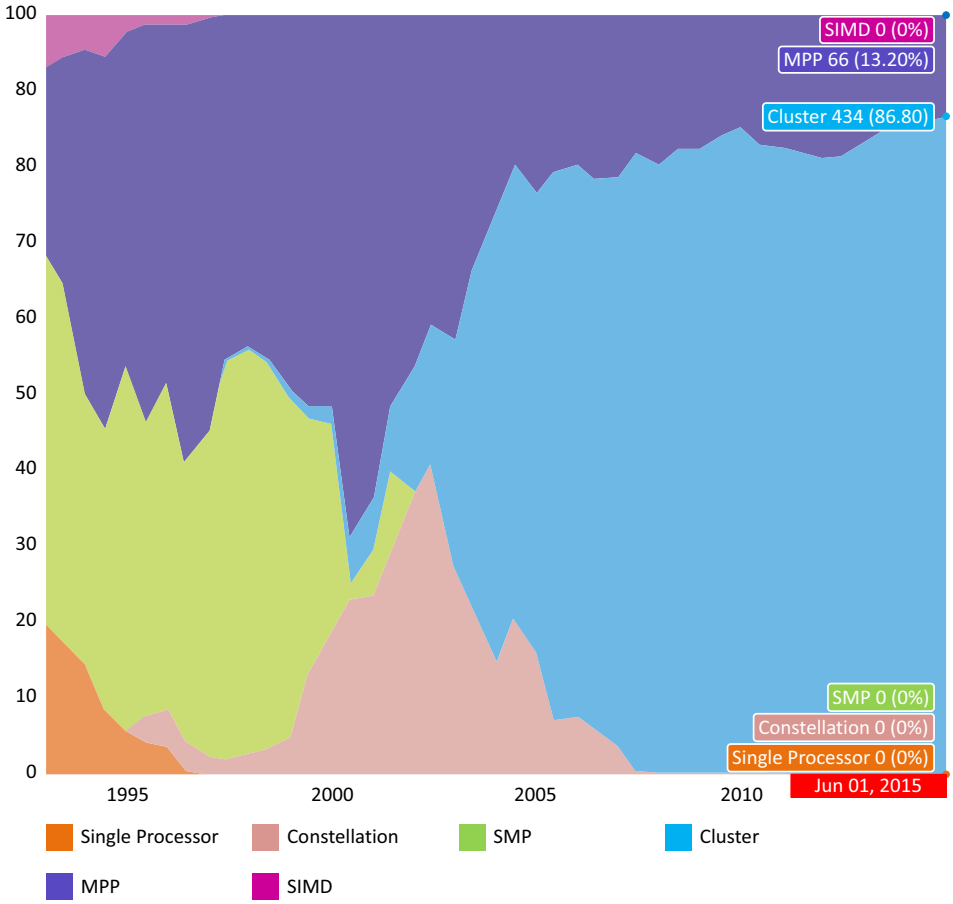


Figure 1-1: Composition of the TOP 500 List [1]

Figure 1-1 depicts the composition of the top 500 HPC systems from 1993 up to now. This shows, that in the TOP500 [1] list, more and more HPC-systems become clusters. Clusters are built in a way that the computing units can be distinguished in different granularities.

One is the processing core. Multiple processing cores can be integrated in the same die or package. Nowadays processors plugged into a system normally consist of multiple cores but are still called central processing unit (CPU). The amount of cores of a CPU can vary from four [2] Xeon E3 from Intel, to sixteen, like in modern Opteron CPUs [3] from AMD, up to over 50 cores in the Xeon Phi [4] from Intel. They are either connected by a socket or can be plugged directly into the system as an accelerator card.

Another granularity would be the node. An HPC-cluster consists of several unified nodes. Each single node contains a motherboard with a certain number of sockets. Every socket is populated with one CPU. In addition a motherboard includes main memory blocks which are dedicated to certain CPU sockets and a Network Interface Controller (NIC) for communication purpose.

With those definitions the granularity of a cluster can be separated into different stages. The smallest granularity is the core, followed by the CPU, the next granularity is the node, and the last one is the cluster. An example is depicted in figure 1-2. The only other current architectures besides cluster systems are Massively Parallel Processing (MPP) systems like the IBM Blue Gene system [5]. All the other architectures like Single Processor, Constellation, Single Instruction Multiple Data (SIMD), and Symmetric Multiprocessing (SMP) are currently outdated.

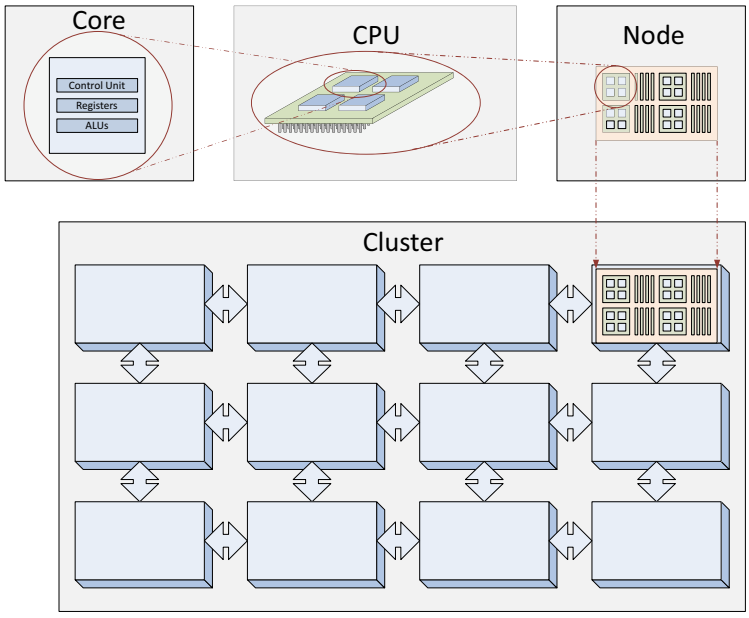


Figure 1-2: Example Cluster System

Two different developing directions can currently be found in HPC. One is to develop more powerful CPUs, like the Intel Xeon [6] with fewer cores but fast and more precise arithmetic units. They can be used to solve universal computable problems with no special restrictions regarding to calculation. In addition, they are best suited if the problem is not divisible into smaller problems that can be parallelized. On the other hand there are many core CPUs like the Intel Xeon Phi or Graphics Processing Units (GPU) like NVIDIA Tesla [7]. Those are used in modern cluster systems like the Dynamical Exascale Entry Platform (DEEP) [8] as an accelerator.

Once the system reaches its performance boundary the need arises to extend its computing power. There are several ways to raise the computing power of a system. One way is to use a new processor generation with more cores, higher frequency, and better arithmetic units. Increasing the core count is limited because of the size of the die and the achievable chip yield. At the moment, the frequency does no longer increase as fast as it did in the past [9]. The technology is still shrinking and higher frequencies should be possible, but the effort to handle the power consumption and heat dissipation is prohibitive [10].

Thus, it is hard to increase the frequency of CPUs in a significant way to meet the needs. The arithmetic units of up to date CPUs are already highly optimized and a tremendous performance boost from this side is not to be expected. Even if there is some potential for improvements to increase the computing power, it is not enough to meet the required performance for the imminent challenging tasks. Another way to increase computing power is to raise the socket count inside one node. However, motherboards are already dense and integrating more sockets than four inside a standard node is hardly feasible and very cost intensive. Alternatively, system speed can be improved by increasing the available memory, but this will not help if the problems that should be solved are not memory bound.

An easy way to extend the system is to add more nodes. Therefore, it is important to have an efficient protocol between the different sockets and the different nodes that allows scaling the system without negatively impacting the overall performance.

"Anyone can build a fast CPU. The trick is to build a fast system." Seymour Cray.

This goal can only be achieved if the key features of a protocol are kept in mind which are:

- Bandwidth
- Latency
- Message rate
- Fault tolerance

Bandwidth needs to be as high as possible to transfer as much data as possible. Latency has to be low to minimize the impact of a hop between nodes. A high message rate is important to maintain high link utilization. Fault tolerance is needed to keep the system running even if it is influenced by the environment.

For the protocols used in HPC systems the topology of the network is of less importance and therefore it is not part of this thesis. The major influence to the protocol is the distance of the communication partners. In case the processing unit is placed on the same chip, the data is transferred directly among different cores via cache coherent protocols or indirectly through main memory. Processing units inside one node but not on the same chip communicate via a CPU specific off chip protocol like the Quick Path Interconnect (QPI) [11] or HyperTransport (HT) [12][13][14]. With those protocols the previously used bus interconnection was replaced and cache coherency is now handled with a special packet based protocol. In case, that processing units are not located in the same node, communication takes place via a NIC. Typically the processing unit then communicates via Peripheral Component Interconnect Express (PCIe) [16] with the NIC, which translates the data stream into its own protocol like 10Gig-Ethernet [17], Infiniband [18], or EXTOLL [19].

Previous systems did already included the NIC inside the processor, like the Transputer [20] with four serial links up to 20 Mbit/s or the nCUBE-2 [21] that included 13 I/O channels that had an operating frequency of 20 Mbit/s. One link was reserved for the typical I/O and 12 created an interconnection network in a form of an order-twelve hypercube. Nevertheless, those systems did not prevail.

Protocols, which are currently used on motherboards, are optimized to exchange data and information among the different components. The most important protocols that fulfill this task are HT, QPI, and PCIe. These protocols are not capable to solve the needs for an

efficient communication among nodes. For those purposes, a specialized protocol is required.

It has to be kept in mind that it is important for a protocol to be efficient. It is necessary to reduce the protocol translation to a minimum because every translation costs time and hardware. To illustrate this, it is depicted in figure 1-3 how data is transferred, from software- and from hardware-perspective. From the software-perspective the programmer creates a 9x3 matrix for every sub problem he wants to be calculated at one node. In order to get a poor data alignment it is assumed that every matrix element is 7 bytes large.

During synchronization the border of the matrix n has to be sent to the matrix n+1. This can be done by a simple Message Passing Interface (MPI) [22] put operation. From the hardware-perspective the sending CPU has to gather the data from the memory. Afterwards, it has to be packed into an HT packet and transferred to the bridge. Then, the bridge translates the packet into a PCIe packet and sends it to the NIC. The NIC from the sender translates the packet to a network packet and transmits it to the receiving NIC. At the receiver all translation has to be repeated backwards until the data can be written to the memory.

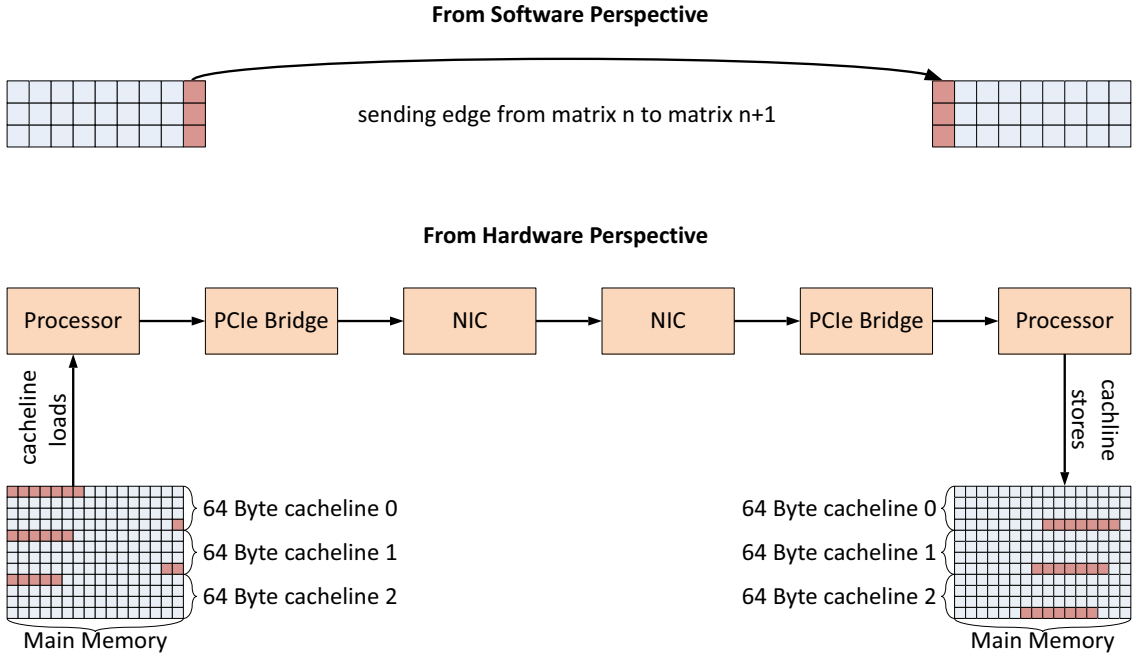


Figure 1-3: Data Transmission from Software- and from Hardware-Perspective

This shows how timing and resource consuming the translations are. Furthermore, the protocol must be easy to decode, as the design effort and the needed hardware resources increase with the complexity of the protocol.

In this thesis current state of the art protocols are analyzed and a new protocol is presented that fulfills the needs for typical communication schemes in HPC systems. It will be usable in different layers of a compute cluster as inter-processor, processor-device-, and inter-node-communication. The translation effort will be reduced to a minimum while all important features, like an ordering scheme and virtual channels, will be supported. Latency will be low and the achievable in-system bandwidth will be competitive.

1.2 Chapter Outline

In chapter 2, Design Space, the general design space of protocols is analyzed with reference to main features. The different main features protocols provide are further classified by their properties. As main features share some properties the influences among the features are explored.

Chapter 3, Off Chip Protocols, focuses on how state of the art off chip protocols are designed. First the difference between on chip and off chip protocols is explained. Then the typical structure and features of state of the art protocols is introduced. The main part of the chapter refers to the detailed analysis of HyperTransport and Peripheral Component Interconnect Express which are the most relevant protocols with regard to the Unified Layer Protocol developed during this thesis.

In chapter 4, Unified Layer Protocol, presents the protocol developed for this thesis. The findings gained in the previous chapters are used to design a protocol which can be used in HPC systems at every point to avoid power consuming and timing critical protocol translations. Special attention has been paid to the feasibility in hardware and the scalability. After the definition of the protocol a detailed comparison against the protocols analyzed in chapter 3 is made.

The contributions to the hardware development of EXTOLL is shown in chapter 5. This work gave the inspiration for the work described in chapter 4. Various methodologies developed for the Unified Layer Protocol have been implemented in this context.

The work concludes in chapter 6, Conclusion and Outlook. In this chapter the outcome of the thesis is summarized and a brief outlook of the future work is given.

Chapter 2: Design Space

Before a protocol can be designed it is crucial to analyze which aspects influence the protocol. In this chapter the main aspects of protocols will be analyzed and how they affect each other.

Modern HPC-systems keep growing in respect to processing units which are distributed over the system. In such systems data has to be exchanged among them. The distance between two communication partners can be in a range from the same die up to different nodes with multiple hops. In the following different connection schemes are presented.

On Chip Connection:

The shortest possible communication distance is between two direct connected functional units (FUs) on the same chip. At this point the number of lanes is not critical as the connection of the FUs is at the same structure size as the logic. The exchange of data can be managed by a simple valid-stop-scheme. Synchronization is no issue as both FUs are in the same clock domain. An example is shown in figure 2-1.

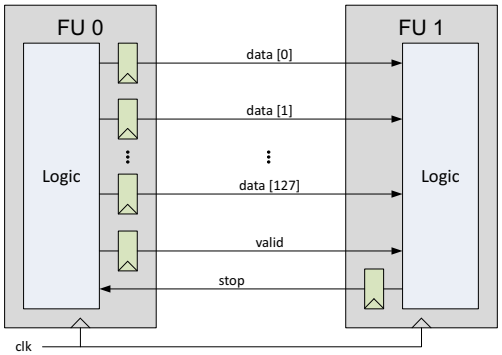


Figure 2-1: On Chip Connection

Off Chip Connection:

Communication between devices which are directly connected, but are not on the same die, needs additional hardware effort. Pin limitation restricts the number of available connections. Thus, serialization has to be used because no large amount of lanes can be used to connect different chips as it is possible on one die. Also logic to synchronize data from the sending clock domain to the receiving clock domain is needed if it is not guaranteed that both domains are synchronous. An example, with a serialization factor of 4, is shown in figure 2-2.

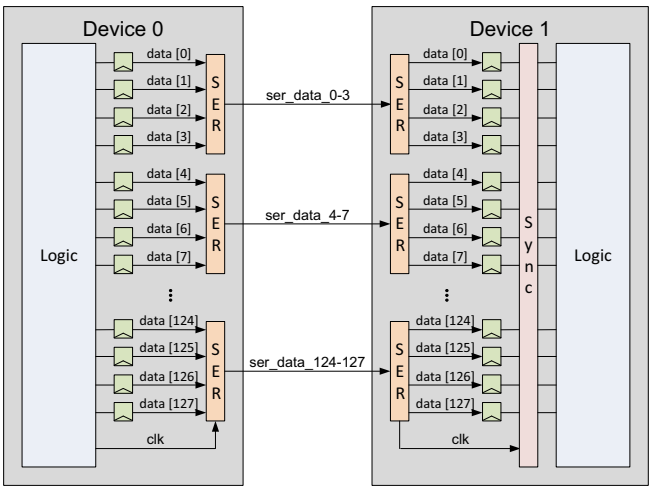


Figure 2-2: Direct Off Chip Connection

Bridged Connection:

A bridged connection is needed if a device has to communicate with one that uses another type of protocol to exchange data. Processors use their own protocols like HT [14] for AMD CPUs and QPI [11] for Intel CPUs. In order to connect other devices protocols like PCIe [16], USB [23], and SATA [24] is common. For accelerators or network devices mainly PCIe is used. An example for a bridged connection can be seen in figure 2-3.

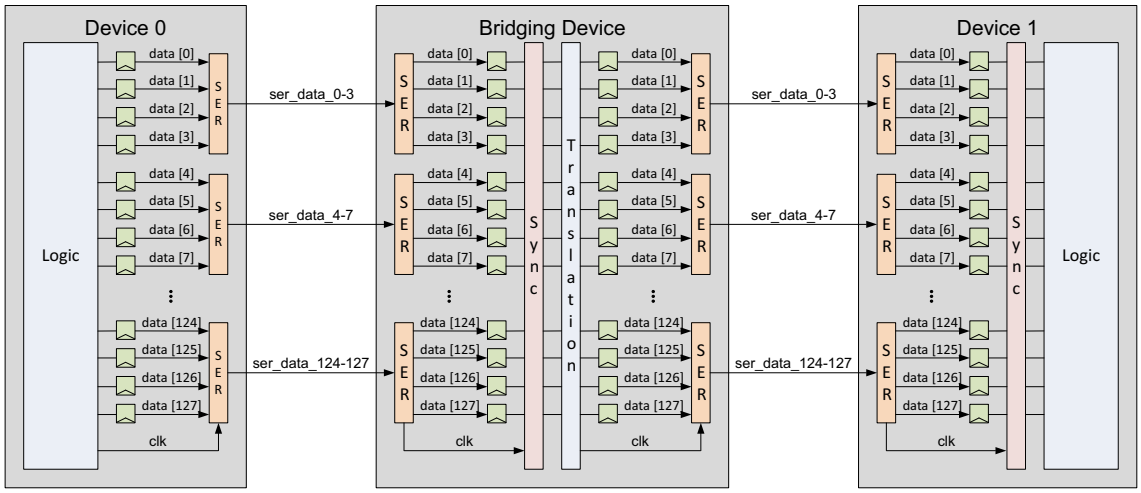


Figure 2-3: Bridged Off Chip Connection

Node to Node Connection:

Nowadays many different protocols are used to transfer data through a computing system. For communication among nodes specialized network protocols like 10 Gigabit Ethernet [17], Infiniband [18], or Extoll [19] are used. So, if communication has to take place among nodes, additional protocol translations have to be made, which can be seen in figure 2-4.

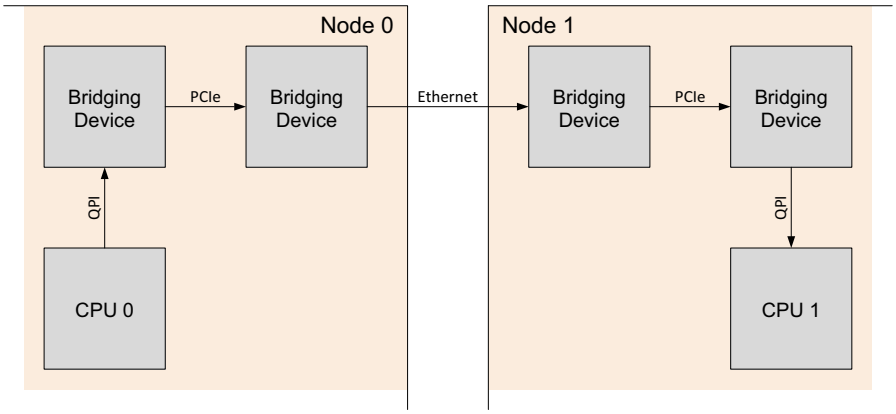


Figure 2-4: Node to Node Connection

Keeping computing time low is very important for an HPC system. With respect to overall computing time it is necessary to minimize the communication effort. This can be achieved with efficient protocols and the reduction of translation effort. Therefore, it is of advantage to unify the communication process and to optimize the protocol structure.

In order to ease understanding how communication is handled inside a computing system it is of advantage to separate different tasks into layers. A well know example how this can be done is the Open Systems Interconnection model (OSI) [25]. The OSI model has 7 layers. An overview of those layers is given in table 2-1.

Layer #	Name	Description
7	Application Layer	Abstract layer to application
6	Presentation Layer	User data is translated into the protocol
5	Session Layer	Organization of the connection as opening and closing
4	Transport Layer	Orders traffic to the right sequence Maps traffic to the corresponding application
3	Network Layer	Logical addressing Packet forwarding Handling routing
2	Data Link Layer	Flow control Error handling Frame synchronization Physical addressing
1	Physical Layer	Physical media (electrical, optical, wireless) Modulation of the signal Line coding Bit synchronization

Table 2-1: OSI Layers Overview

Layers 1 to 3 are the layers which are close to the transport layer where 5-7 are close to the application layer. Layer 4 is a link between those. For this dissertation only layers 1 to 3 are of importance as they are closely coupled to the hardware side of a protocol.

Efficient protocols have primary features. Those features are latency, bandwidth, message rate, and fault tolerance. They have to be well balanced to achieve the required performance. The primary features have attributes such as frequency, data width, data dependencies, overhead, retransmission rate, resource availability, and fault tolerance. An overview is shown in figure 2-5.

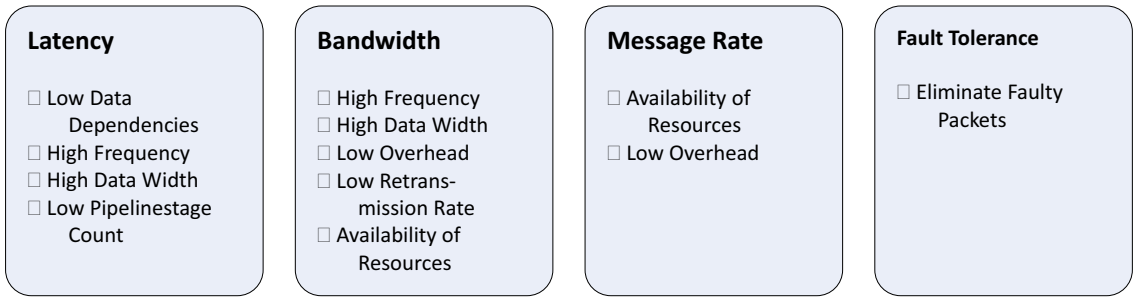


Figure 2-5: Primary Feature Overview

Some of the primary features share the same attributes. Changing an attribute in the favor of one primary feature will influence the others sharing it. Sometimes a change of an attribute will influence other attributes in a positive way, but often a conflict with other at-

tributes occurs. Therefore, it is important to find a suitable balance among all attributes. An overview of the dependencies among the different attributes is shown in figure 2-6. The arrows, which connect the attributes, symbolize the different dependencies between them and that a change will influence the other.

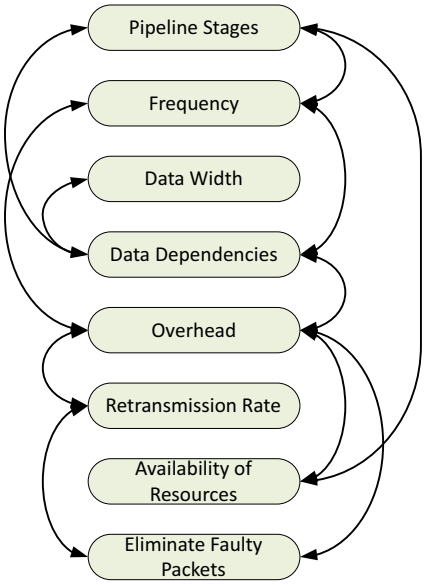


Figure 2-6: Attribute Dependencies

Protocols which are used to achieve high utilization are packet based. One packet consists of different frames. In most cases three different frames are given. First the header frame which contains the information necessary to transfer the data from the source to the destination. Second the data frame which contains the payload. The third frame is the frame which is used for error handling. The error frame contains redundant bits for error detection or error correction. An example of a packet is given in figure 2-7.

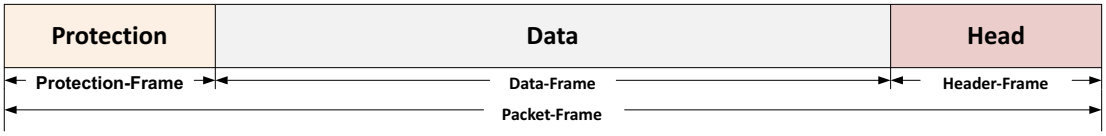


Figure 2-7: Packet Framing

As already mentioned the header frame contains the information of how to handle a packet. Therefore some information is essential. That information is packet type, source of packet, destination of packet, packet size, and address. An example of a header frame is given in figure 2-8.

	7	6	5	4	3	2	1	0
0	Packet Type							
1	Sender ID							
2	Receiver ID							
3	Packet Length							
4	Address							
5	Packet Specific Field							
6	Packet Specific Field							
7	Packet Specific Field							

Figure 2-8: Header Frame Example

The header can also contain additional information in the packet specific fields to support some more advanced features like virtual channel type, credit information or additional identification information. Virtual channels are used to share one physical lane among multiple virtual lanes. Therefore, the resources to manage the physical lane are replicated for every virtual channel. Those resources are buffer spaces and the buffer management. Credits are used to ensure that there is no overflow at the receiver side. Therefore, the receiver sends information to the transmitter about how many packets it is able to receive. That credit information needs to be constantly updated during run-time.

In the following subsections the different primary features are described in detail.

2.1 Bandwidth

Bandwidth is defined as data volume per time unit. In most cases, this means Megabyte per second (MB/s) or Gigabit per second (Gbit/s). There are different types of bandwidth. On the physical layer of a channel the raw bandwidth is defined as how much data can be transferred over a link or inside a chip at a given time period. At every clock cycle the data path is assumed as fully utilized, regardless what kind of data is transmitted, user data or protocol overhead. If the overhead for line coding is subtracted from the raw bandwidth the coding layer bandwidth is given. The remaining overhead from the protocol is the overhead for framing. After removing the framing the amount of user data which can be transferred if the data path is fully utilized and the overhead is minimal. This is the payload bandwidth. An example of how the different bandwidth types utilize a channel is shown in figure 2-9.

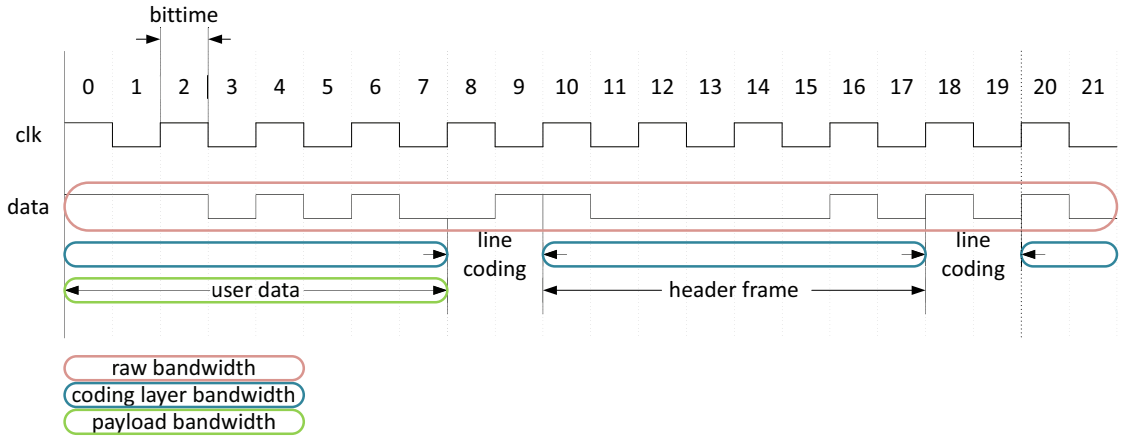


Figure 2-9: Bandwidth Utilization

The last type of bandwidth is the in system bandwidth. In this case, the achieved bandwidth during runtime is measured. For attributes, only the raw bandwidth is of importance. Other types of bandwidth need a complete specified protocol or a defined used case to know what parts of the raw bandwidth are lost to line coding, protocol coding, and system behavior.

Increasing raw bandwidth on a link is easy. There is no influence to the decoding of the protocol but only to the transportation speed. This can be achieved by adding additional lanes or a change to a faster serializer technology. Nevertheless, there are physical restrictions that have to be taken into account such as pin count of a package, connector size, and the frequency of the serializer and their power consumption. If the link bandwidth is increased, the chip internal bandwidth needs to be increased as well.

2.1.1 Data Width

One common way to increase bandwidth is to expand the data width. The theoretical bandwidth changes proportionally to the data width if the frequency is kept constant. Doubling the data width automatically doubles the theoretical bandwidth. If this can be realized without negatively influencing other primary features, it is a convenient way to increase bandwidth. However, some constraints cannot be disregarded.

There are physical restrictions like pin limitation, bracket size of connectors, and multiplexing complexity. Pin limitation and bracket size are dictated by the environment off

chip. Therefore, they can be decoupled from the inner design of the chip. In case the data width is increased, the additional data lanes will require more wiring resources. This results in an increased area requirement inside the chip. Also multiplexing complexity can increase if the data width is changed. Therefore additional logic is needed which has a negative influence on frequency and area.

Inside an ASIC the wiring is flexible but there is a high risk of increasing costs due to larger chip size or additional metal layers. If more space is needed for wiring also the achievable clock frequency is influenced negatively as the distance between two register stages increases.

In case of an FPGA the wiring resources are fixed and limited by the internal FPGA structure. Changing data width will decrease the achievable clock frequency and the mapping into the device will be more difficult. At some point increasing the data width will result in an un-mappable design.

There are also logical constraints which limit the data width. More data is received in one clock cycle. This increases the chance that more data dependencies inside one header frame have to be solved at once. In case, the data width is wide enough not only more dependencies inside one header frame can occur, but also among multiple packets. Solving additional dependencies in one clock cycle will result in more hardware complexity. In addition, the multiplexing complexity has to be increased. This happens because of two reasons. First, if the starting-point of the header is not fixed there are more locations to gather the data from as the different fields of the frame shift through data width. Second multiple packets can be received at the same time and therefore additional wires have to be used. This means increasing the data width can have a significant influence on the attribute frequency, as the frequency cannot be higher as the time that is required for data processing plus wire delay.

2.1.2 Frequency

Increasing the frequency is also a very effective way to increase the bandwidth. Doubling the frequency automatically doubles the theoretical bandwidth. Nevertheless, the frequency of a given technology for a design will be usually fully utilized. Therefore, increasing frequency leads to a redesign. There are two possibilities how this can happen.

One is to change to a faster technology. In case of an ASIC this means scaling to a faster process with smaller feature size. In case of an FPGA, one with better capabilities has to be used.

The second possibility is to modify the HDL code. If the current version of the code is written efficiently, just redesigning the functional units will not result in a huge frequency step. If all or at least the critical path can be pipelined further, a higher frequency can be achieved. However, it is well known that pipelining has a negative effect on latency. At least the additional time for the register stages have to be added to the processing time. In most cases, the current problem which has to be processed cannot be divided into multiple calculations with a similar run-time.

In case the pipeline-stage with the largest timing budget cannot be subdivided into smaller problems, there is the possibility of increasing the frequency by parallelization. Therefore, this pipeline-stage has to be duplicated and special multiplexing logic has to be added. The multiplexing logic switches between the two parallel blocks so the throughput can be doubled. Then the rest of the logic can run at higher frequencies while the duplicated blocks can be clocked by half the new frequency. The disadvantage of this solution is that the additional logic consumes part of the already critical timing budget, more logic resources, and the clocking scheme becomes more complex.

This means that increasing the frequency comes with tremendous costs for a new design and latency might increase. In figure 2-10 part a) shows the normal processing time of a logic block, b) shows an optimal pipelined logic block with the extra timing needed for the additional registers, c) shows a more realistic pipeline approach with not equally divided sub-blocks, and d) shows a pipelining approach with parallelization.

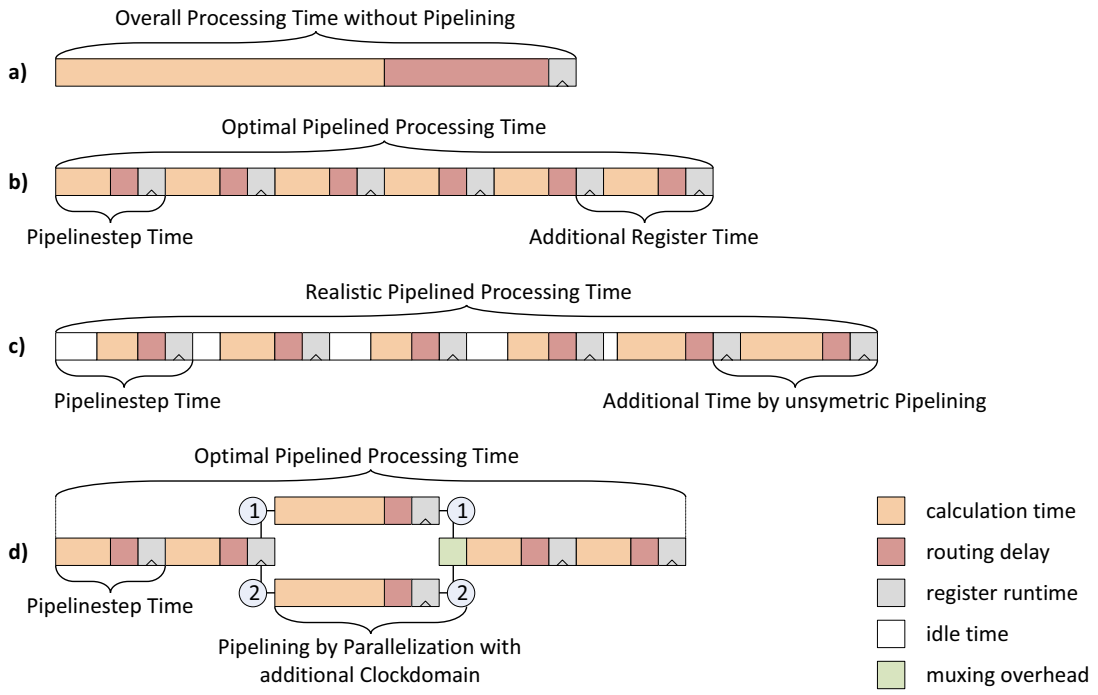


Figure 2-10: Pipelining

There is one additional point that limits the achievable frequency. If there are physical limitations that restrict the link bandwidth, the chip internal bandwidth is also limited to this value. Therefore, there are some fixed ratios between internal data width and frequency that will utilize the full link bandwidth. From a bandwidth perspective it is only useful to increase the frequency to the point until the corresponding ratio between frequency and data width is met. Increasing the frequency further will not increase bandwidth, but only reduce the latency inside the device.

2.1.3 Overhead

The overhead can be subdivided into three major types of overhead:

- Header Overhead: Introduced by the information needed to process a packet.
- Protocol Overhead: Introduced through protocol specification.
- Error Handling Overhead: Introduced through error handling.

Those overhead types will be described in the following sub-chapters.

2.1.3.1 Header Overhead

The challenging part of the header overhead is to find a balanced amount of overhead between features and needed bandwidth. This means header frame size should be as small as possible, but large enough to be able to support all intended features. In addition, it is also important not to shrink header fields in a way that the alignment of the packet or the header will cause problems. It is smart to keep in mind that not only a protocol with low overhead is of importance, but also how efficient it can be realized in hardware. Sometimes it seems to be of advantage to save a single bit by enhanced coding from the protocol perspective, but this can make the decoding hardware more complex. Thus, the small advantage, which was gained by coding, will lead to significant effort inside the hardware.

2.1.3.2 Protocol Overhead

One possibility how a loss of bandwidth through protocol overhead can happen is if there are restrictions inside the protocol regarding frame alignment. If the packet always has to start at a defined position of the data width, but the packet length is not to a multiple of the data width, then there is space at the end of the packet that has to be kept empty. Such an empty space is called a bubble as it occurs due to the defined protocol but cannot be avoided. An example is given in figure 2-11 where a bubble occurs because of doubling the data width on one transmission channel.

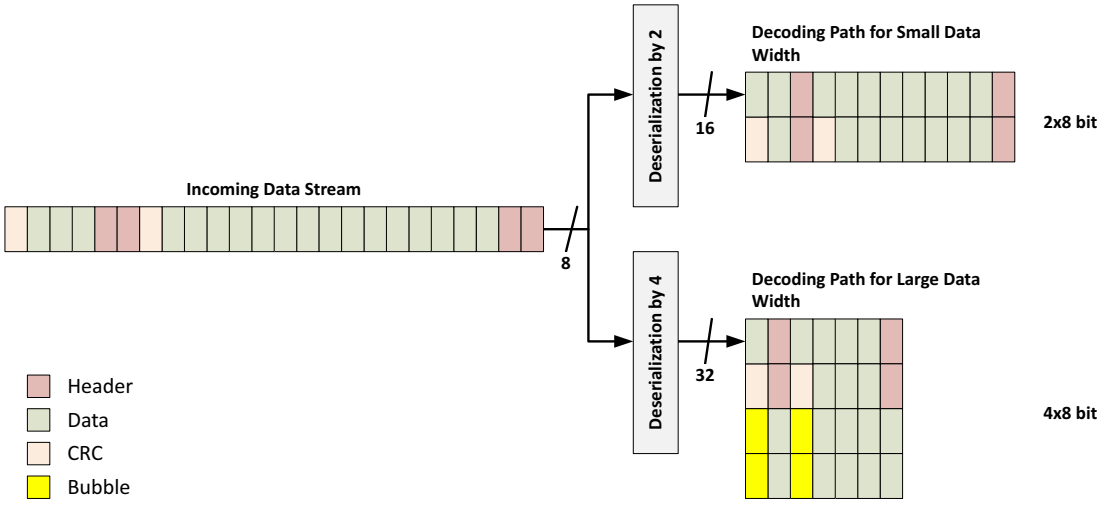


Figure 2-11: Bubble Insertion Through Protocol Constraints

Another very important point, which influences the packet frame overhead, is the data granularity. From a simple perspective, the easiest way of transporting data from sender to receiver is a plain data stream without any additional information. After an initialization phase, there is no wasting of bandwidth because of unnecessary overhead. However, this is only useful if the same amount of data is repeatedly written to a strict defined destination and data failures do not result in a system crash. Furthermore, such a kind of system needs to run completely synchronous, because a loss of sync can never be recovered without re-synchronization. Therefore, there is a very limited field of applications where this kind of granularity can be used, like media streaming and other uncritical and resettable tasks. The problem of a constant data stream can be avoided by splitting the data stream into several packets.

In order to assure the acceptance and usability of a protocol there should be no tremendous restrictions about the length of data or the address range. For example, analyzing HT shows, that packets with every combination of one byte up to 64 doublewords can be sent. Data range can also start and end at every byte-address as long as they do not cross a 4 kB address boundary. Nevertheless, this does not mean that there is always an optimally sized data frame defined to transmit the data without losing any bandwidth.

Bandwidth is always lost when the amount of data transferred does not fit into a given data frame. Thus, different cases are possible. Data that has to be transmitted could fit inside the next larger packet frame. In this case, the unused portion of the data frame leads to a loss of bandwidth. In case the data is too large to fit inside the largest available frame, it has to be split into multiple frames. Thus, additional control information is required. Furthermore, the data alignment could not be suitable. This means the data volume is too large for one frame. Using the next larger frame would result in higher bandwidth loss because most of the frame would be empty. In this case, it is more efficient to use multiple smaller frames. In figure 2-12, the possibilities of how the data can be split are shown. These possibilities are described in the following sub-chapters.

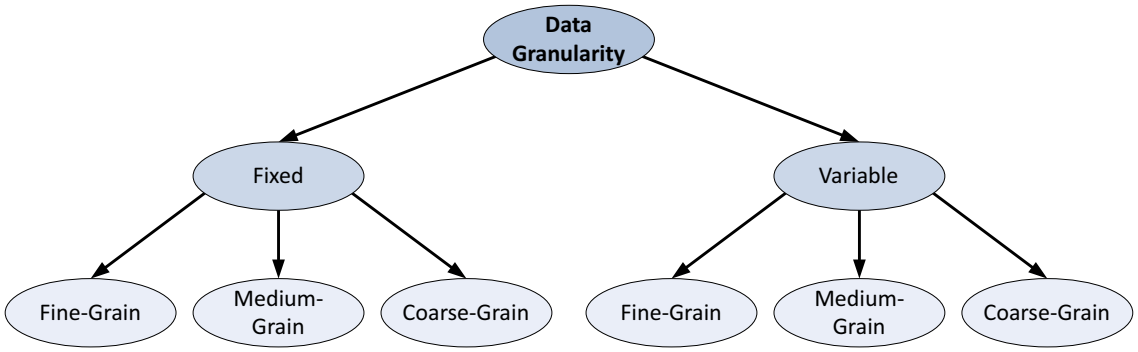


Figure 2-12: Data Granularity

Fixed Data Granularity

With a fixed data length approach, the data has to be split into frames with the same length. A header in front of the data describes the kind of transmitted data. At least, the location where the data has to be written to and the amount of valid data are information the header needs to contain. If an error occurs during transmission this must be detected. Therefore, a protection frame per packet is common in current protocols.

The advantage of a fixed data granularity is that the hardware to decode and process data is simple. The deterministic repetition of frames makes sure that no complex logic is needed to determine which kind of data has been received in one clock cycle and where the data has to be forwarded. In figure 2-13 a) a fixed protocol type with a header, followed by data, and protected with a Cyclic Redundancy Check (CRC) [26] is assumed. As long as the data path width inside a design fits into the protocol granularity the data just has to be gathered and if one packet is complete it can simply be forwarded. If the data path width is larger than the protocol granularity, it is not necessary that the handling of one packet has to become more complicated. As long as the packet length is a multiple of the data path width the wiring gets even more relaxed, as depicted in figure 2-13 b). In the case that the packet length is not a multiple of the data path width the start of packet location will shift through the data path. Thus, additional multiplexing resources have to be added, more wiring resources have to be used, and additional buffer space must be added to gather complete packets. The logic controlling the multiplexer is simple, because the shifting of a packet start through the data path is deterministic. A problem arises if a pack-

et could be finished in the same clock cycle as a new packet starts and the processing unit would still be utilized with the first packet.

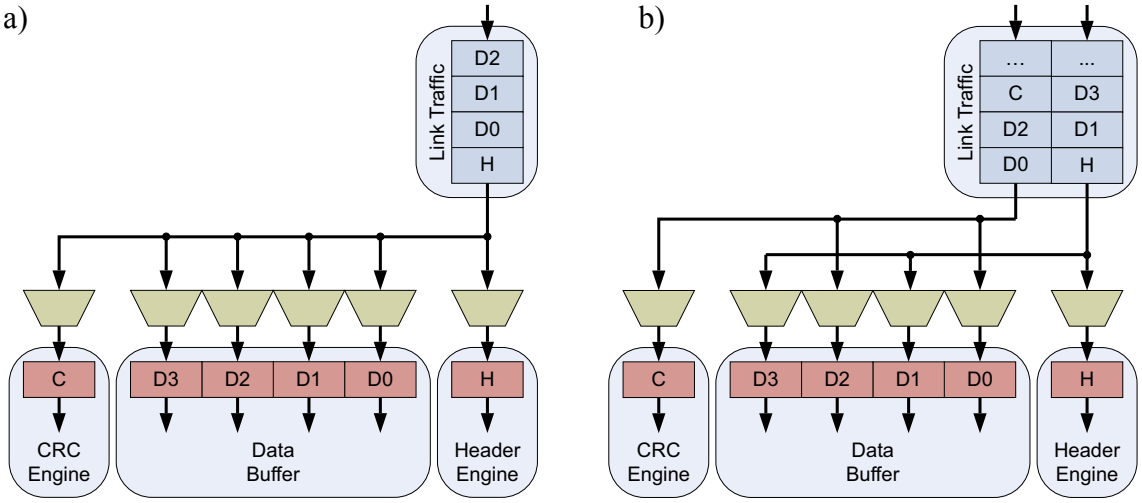


Figure 2-13: Muxing Complexity Fixed Granularity

Finding a suitable packet length for a fixed granularity is not simple. If the data which has to be sent does not have a given granularity, it is hard to determine which granularity fits to avoid a loss of too much bandwidth.

There exist several ways in which choosing a fixed format results in loss of bandwidth. When using a fine grain granularity, there will be a loss of bandwidth if large data packets have to be sent. In this case, the data has to be split into short packets and additional header information has to be sent. However, this is unnecessary and therefore only wasting bandwidth because a large and consecutive amount of data could be transferred with just one header.

If a more coarse grain granularity is chosen there will be less waste of bandwidth in terms of header information. Thus, a large amount of data can be transferred with fewer packets. For short packets on the other hand, the data frame of one packet would be almost empty. This means, that also in the case of a coarse grain granularity, bandwidth can be wasted. An example is given in figure 2-14.

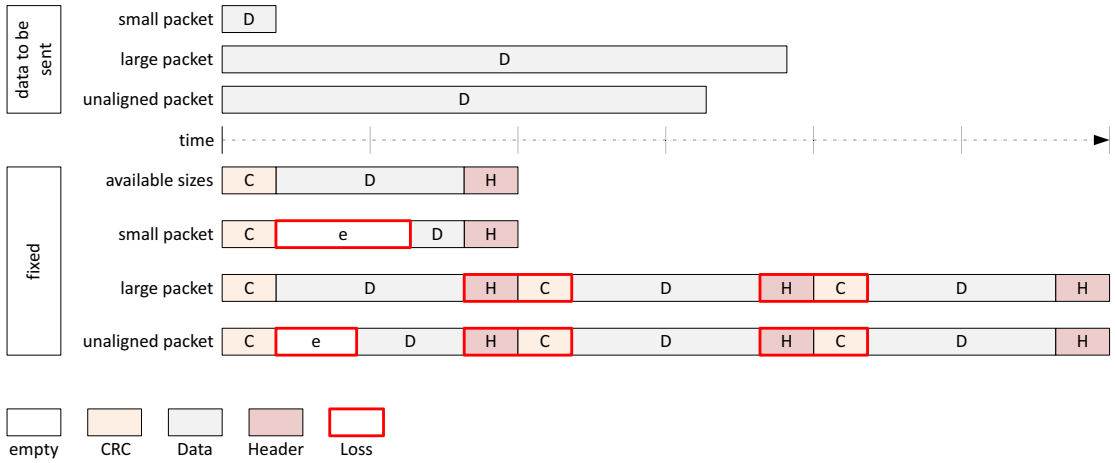


Figure 2-14: Bandwidth Loss with Fixed Granularity

In both cases, small and large granularity, bandwidth will always be wasted if the data that has to be transferred does not match a multiple of the given data frame. Hence, in a design with a fixed granularity where traffic with random data length has to be transferred, it is hard to choose a suitable data frame size. Except of a fixed size data frame generating system, it is impossible to avoid any bandwidth loss so a compromise has to be made among packet sizes to achieve a result with moderate bandwidth loss.

However, in some application areas a fixed data granularity has been chosen. In telecommunication SONET and SDH are widely used [27]. Those protocols have also been embedded into modern FPGAs with the latest version of the Altera Aria series [28]. They use Synchronous Transport Signal (STS) frames to transmit the data. In [29] the structure of the different frames is described. Two frame types are used. One is the 810 bytes large STS-1 frame and the other is 2430 bytes in case of STS-3. STS-1 and STS-3 are structured as a matrix with 9 bytes rows and 90 respective 270 columns. The structure and the differences between the two frame types can be seen in figure 2-15.

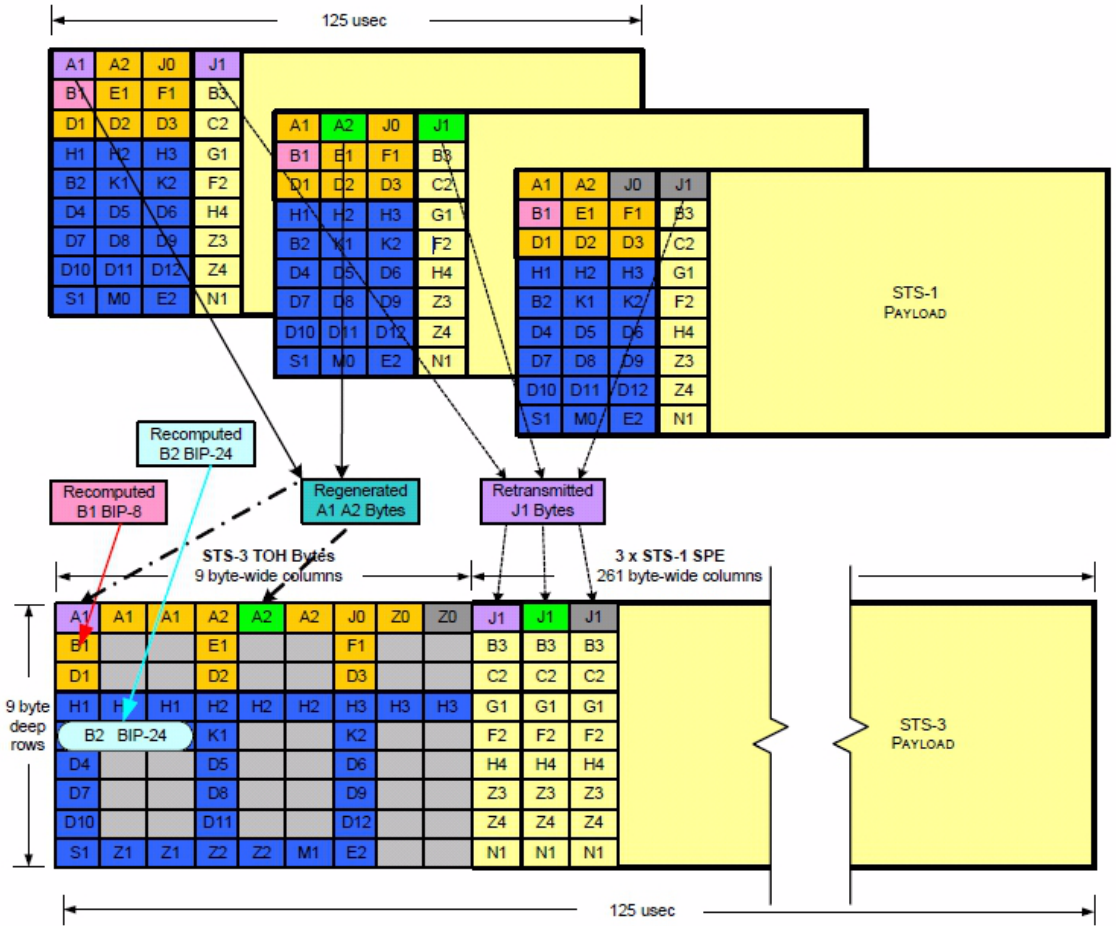


Figure 2-15: SONET/SDH Framing [29]

The Header of the frames consists of two different main sections. First the Transport Overhead (TOH) as a 9 by 3 bytes field. It contains the Section Overhead (SOH) with 9 bytes and the Line Overhead (LOH) with 18 bytes. Second the Synchronous Payload Envelope (SPE), which is a 9 bytes row that contains the Path Overhead (POH) and the STS Payload Envelope. Column 30 and 59 are used as fixed stuff columns and do not contain data. Thus, the protocol overhead is 6.77%. Because of its format SONET/SDH provides a lean protocol which is easy to use and provides a guaranteed bandwidth. Therefore, it is a good solution for data aggregation. Nevertheless, for HPC more flexibility and additional features are of importance, so SONET/SDH does not fulfill the needs for HPC.

Variable Data Granularity

Compared to fixed packet granularity the variable granularity changes from coarse grain to fine grain just by the number of data frames with different lengths. The most coarse-grained granularity would be just one allowed packet size, which is in this special case the same as a fixed format. Every additional valid packet length increases the flexibility of sending data without losing bandwidth, but the hardware complexity increases. Granularities which should be chosen depend on what kind of traffic will be mainly transmitted by the protocol. The variety of all possible granularities reaches from a single data-frame length up to all possible data frame sizes. The upper bound for the number of packet sizes is determined by the size of the length field inside the header and the restricted size of the receive buffer. The length field can just contain the plain number of bytes or the number of protocol words that have to be transmitted.

With a variable packet granularity, the complexity of decoding gets more complicated compared to a fixed format. There is no pre-defined position of header-, data-, and CRC-frames inside the stream. These could be signaled by sideband signals, which determine what kind of frames are currently transmitted. Otherwise, the hardware needs to keep track of the data stream so it can be clearly determined at every point in time what kind of frame has to be processed. In this case, additional hardware complexity is introduced for the case that the data payload does not fit into one of the given frame granularities. Thus, the payload has to be split into multiple frames or one larger frame has to be used. Finding the optimal number of frames increases the bandwidth but adds additional hardware for the correct frame selection. However, if the variety of possible frames is large, determining the best fragmentation is complex.

If a coarse grain granularity is chosen, the loss of bandwidth will be relatively high but the hardware can be kept simple. Figure 2-16 shows how the bandwidth loss is reduced compared to the fixed granularity.

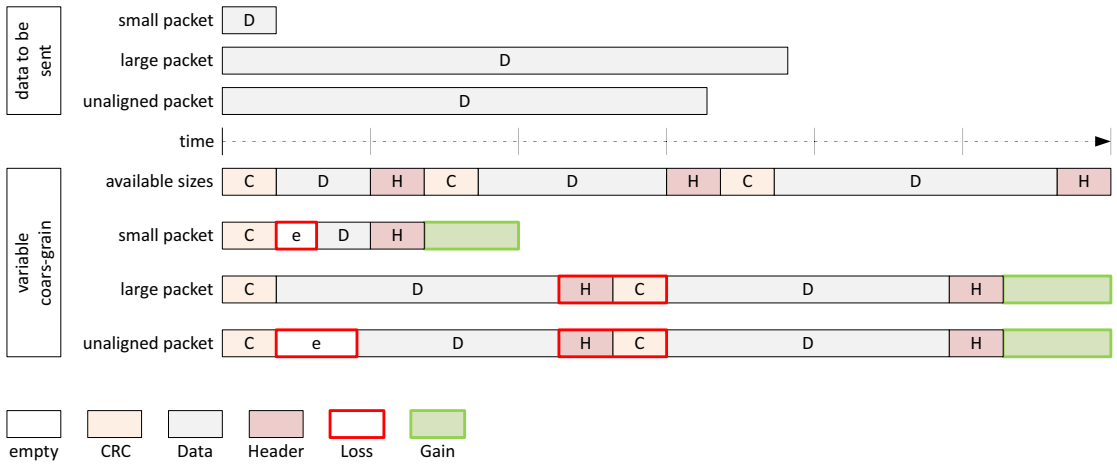


Figure 2-16: Bandwidth Loss with Coarse Grain Granularity

When the granularity levels are increased, the bandwidth loss will further decrease and the hardware effort will increase. An example for a maximum fine grain granularity is given in figure 2-17. This solution is theoretical if the range of possible data frames is too high, because at some point, the gain of bandwidth will not justify the increasing hardware complexity.

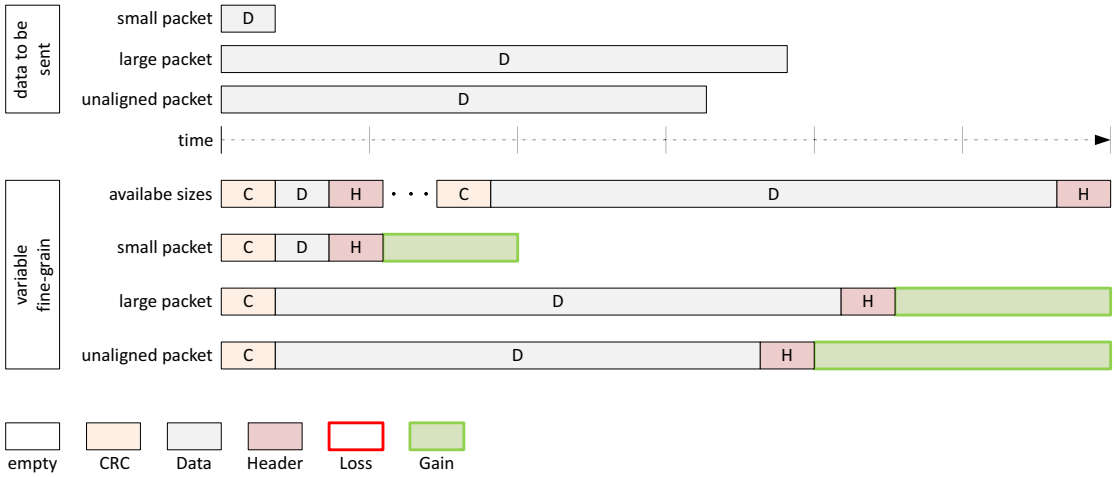


Figure 2-17: Bandwidth with Optimal Granularity

Hierarchical Variable Data Granularity

In order to understand how the hardware complexity is influenced by the types of data transfer it is important to understand how data is transferred in a system. From an abstract view data is just moved from one address space into another address space. In the example of figure 2-18, 128 bytes of data is transferred from address space A to address space B.

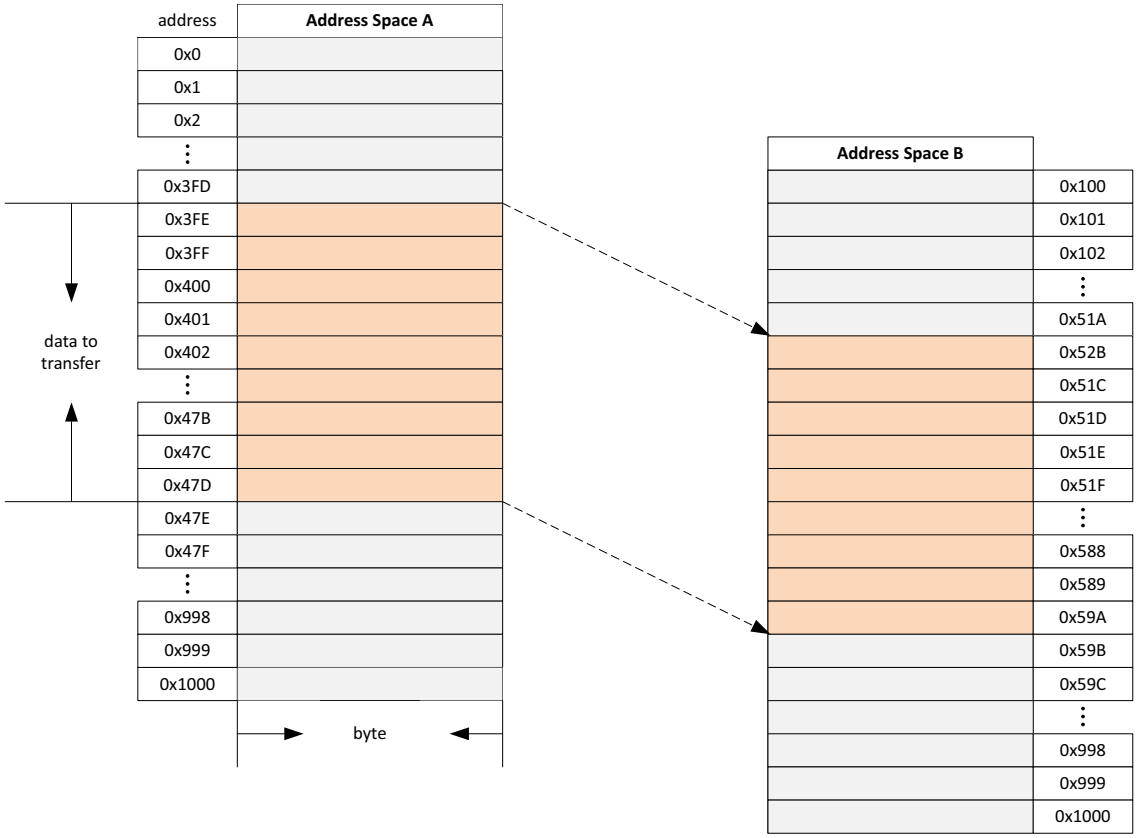


Figure 2-18: Address Space Data Transfer

Reaching a decent amount of flexibility can be achieved by using hierarchical granularities, without increasing the hardware effort to a too complex level. Therefor the transfer granularities are separated into different boundaries. For very small data frames a very fine grain granularity is used. Midsized packets use a more coarse grain granularity, whereas large packets use a coarse grain granularity. Figure 2-19 shows an example where the transmission of data is divided into three packet types with different granularities. Packets of a size smaller than 64 bits are transmitted with the exact amount of bytes they can contain with a range from 1 to 7. If more data than 64 bits has to be transmitted, the

next larger granularity has to be chosen. In this case, this means multiples of 8 bytes are transmitted up to a range of 56 bytes. In addition, a third packet type is available that allows to transmit packets with a data granularity of 64 bytes, up to maximum range of 512 bytes, which is also the maximum packet size in this example. Larger packets have to be split into multiple packets. In this solution no restrictions were made regarding to starting address and address boundaries.

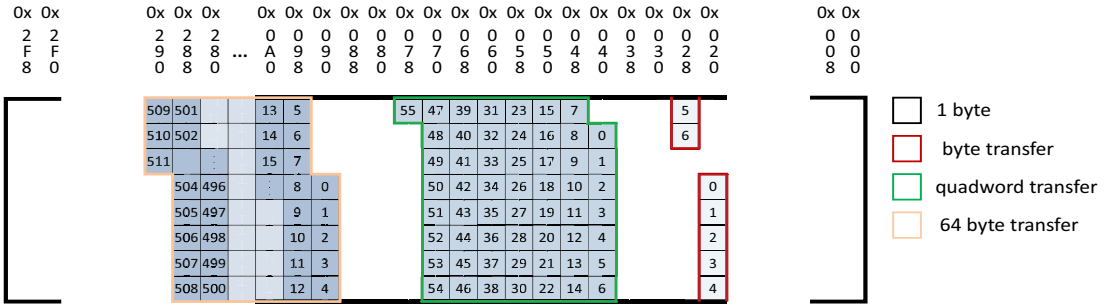


Figure 2-19: Variable Hierarchy Example

If only those types of packets are used, a problem occurs when data has to be transferred that is too large for a smaller frame, but is misaligned to the granularity of the larger packets. This case needs some special handling, as it cannot be managed without additional effort. One is that a larger frame would be used and marked as partly occupied and the additional available space has to be left empty. A better solution to support these unaligned packets is to split one packet into multiple frames. It must be checked what kind of alignment the packet has regarding to start address and length. Then the misalignment has to be solved by sending smaller packet types until the alignment for the next larger packet type is reached. With this mechanism, every length of data can be transferred without wasting data frame bandwidth but at the cost of additional header frames. This solution is shown in figure 2-20. An 86 bytes large data frame has to be sent. Without splitting the packet, a 64 bytes granularity frame with two entries had to be used and 42 bytes would be wasted. As shown, with frame splitting, no data frame space is wasted, but additional headers have to be transmitted.

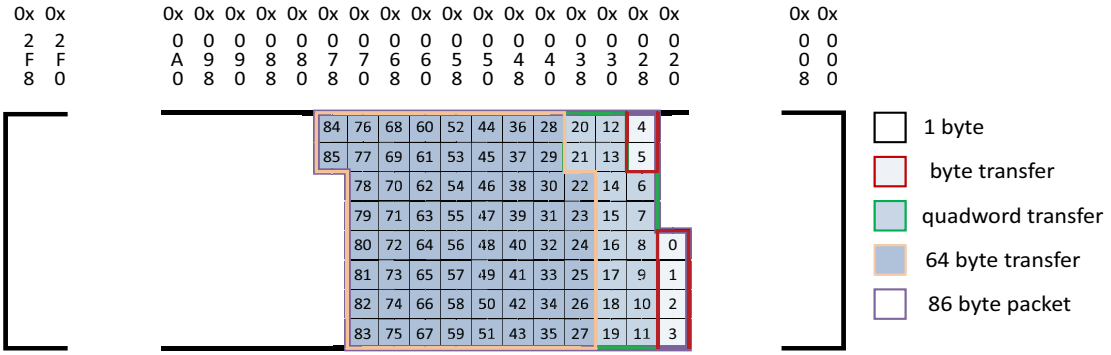


Figure 2-20: Variable Granularity Compact

This solution is efficient from a bandwidth perspective but is still rather complex from a hardware perspective. The calculation what kind of packets have to be used depends on a mixture of the start address and the length field and, after the correct packet type is found, also the length of the packet needs to be determined. Furthermore, the alignment of data is not subject to any restriction. It is possible to structure the data transmission further to relax the hardware effort. The data of a given packet can be transmitted regarding to the addressing and not to the length of the packet. An example is shown in figure 2-21. With this mechanism, additional headers are possible and would result in loss of bandwidth. The advantage is that the frame calculation is less complex and the data is automatically aligned to the address space. For the first misaligned packet the packet type and the packet length only depend on the lower address bits. The last packet is determined by the lower bits of the packet length.

Figure 2-21: Variable Granularity Aligned

2.1.3.3 Error Handling Overhead

Error handling introduces additional overhead. Bandwidth is needed to add redundancy for protection, which reduces the effective bandwidth. Bandwidth will also be reduced if data has to be retransmitted to correct an error. The impact of retransmission will be larger in correlation to how often an error occurs. This aspect will be analyzed in detail in chapter “Fault Tolerance” on page 42.

2.2 Latency

Latency is the time that a packet needs to travel from source to destination. There are two typical examples what kind of latency is of interest in networks. One is the round-trip latency and the other one is the half-round-trip latency. The round-trip latency is measured by sending one packet from the first node to the second node, and then sending the packet back from the second node to the first one. The time from the start of sending to the end of receiving is called the round-trip latency. The half-round-trip latency is just the time a packet needs from start of sending at node number one until full reception at node number two.

2.2.1 Data Dependencies

The most important attribute that influences the latency is data dependencies. In this context data dependencies stand for dependencies among different fields in one header frame. Those dependencies have to be handled by hardware. Besides calculations that have to be made for processing the data there are calculations that occur because of data dependencies. In case of low data dependencies, the required logic to process the data can be kept simple and therefore fast. If the dependencies are high, the logic will become more complex. Most likely, all calculations have to be made before the dependencies can be resolved. Not only will the overall processing time be higher if there are more dependencies, but also pipelining will become more complicated regarding to how efficient and even if resolving those dependencies can be pipelined.

In current packet based protocols, the control frames can be distinguished by their command field. Depending on what kind of control frame it is different fields of the frame can have different meanings. In this case, some fields of one frame have to be decoded before a decision can be made regarding other fields. The longer this chain of decoding gets the higher the negative impact on latency becomes. The impact increases if the dependent field is not received in the same clock cycle. Thus, data has to be buffered and the number of bits that have to be decoded can even be higher than the data width. The example given in figure 2-22 shows a very simple control frame with a command, length, source, and destination field. Every field is 8 bits wide. It is assumed, that there are doubleword (32 bits) and quadword (64 bits) accesses. So first, the command field has to be decoded before the length field can be correctly interpreted.

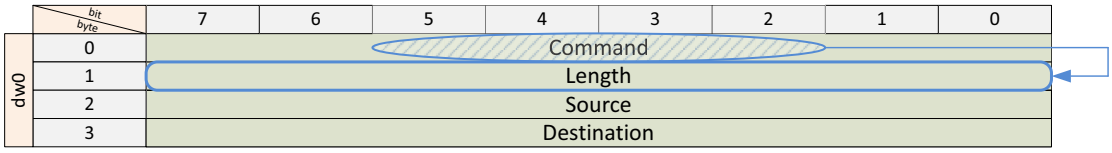


Figure 2-22: Control Frame Dependencies

Not only will the decoding time increase because of such dependencies, but also the wiring complexity. For every bit position inside the control frame that has an additional meaning, a new connection from the same source register to a different decoding unit has to be established. For the simple protocol example, first, the command field has to be decoded before the two length options of doubleword and quadword can be differentiated and forwarded to the corresponding unit. This behavior is shown in figure 2-23.

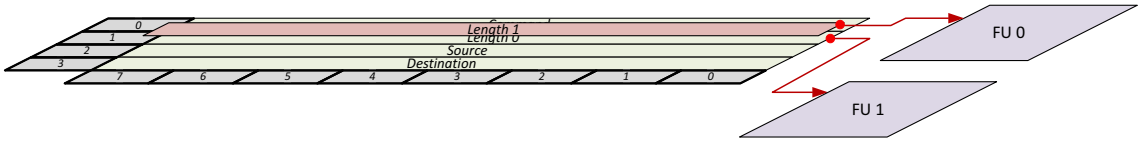


Figure 2-23: Wiring Complexity

In addition, multiplexing complexity increases tremendously if the start of a packet is not well defined inside the data width. The start of a packet can be at many positions and therefore there must be a connection from every possible position of the data width to the decoding logic. In case data width is increased, the packet has more possible starting positions. Every additional possibility results in an additional multiplexing point. The exam-

ple in figure 2-24 depicts how the length field location shifts trough the data width if the protocol granularity would be 8 bits. Multiplexing complexity multiplies for every different meaning the field might have.

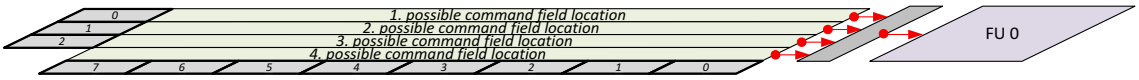


Figure 2-24: Muxing Complexity

As soon as the packet size becomes smaller than the data width, it will be necessary to process multiple packets within one clock cycle. This causes a massive problem as two headers can be received at the same time but cannot be handled simultaneously by the same functional unit. Figure 2-25 shows an example, where the data width can contain up to four control frames and the data dependencies among them.

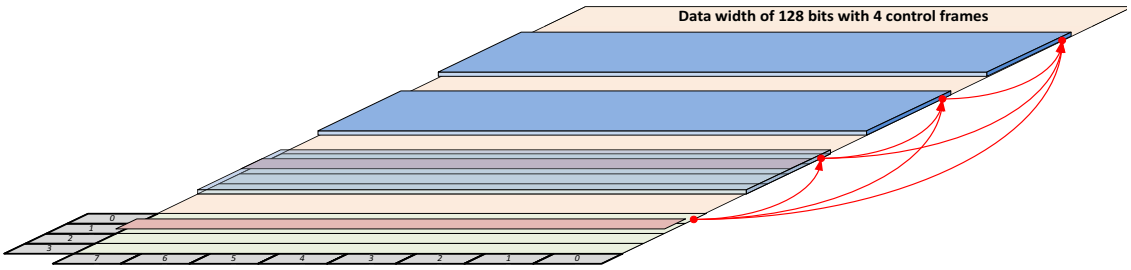


Figure 2-25: Inter Frame Dependencies

In order to avoid back pressure a solution is needed. One would be if one packet is buffered for later processing. This would result in bandwidth loss if it is not guaranteed by the protocol that there will be a time slot to process the stored packet without influencing the rest of the data stream. This will work as long as one packet can be stored for later processing. However, if the packet has to be processed in the same clock cycle to sustain the bandwidth, the functional unit has to be replicated as shown in figure 2-26. Even if the functional unit is replicated, at some point the two simultaneous received packets have to be serialized if they have the same final destination and could cause back pressure at this point.

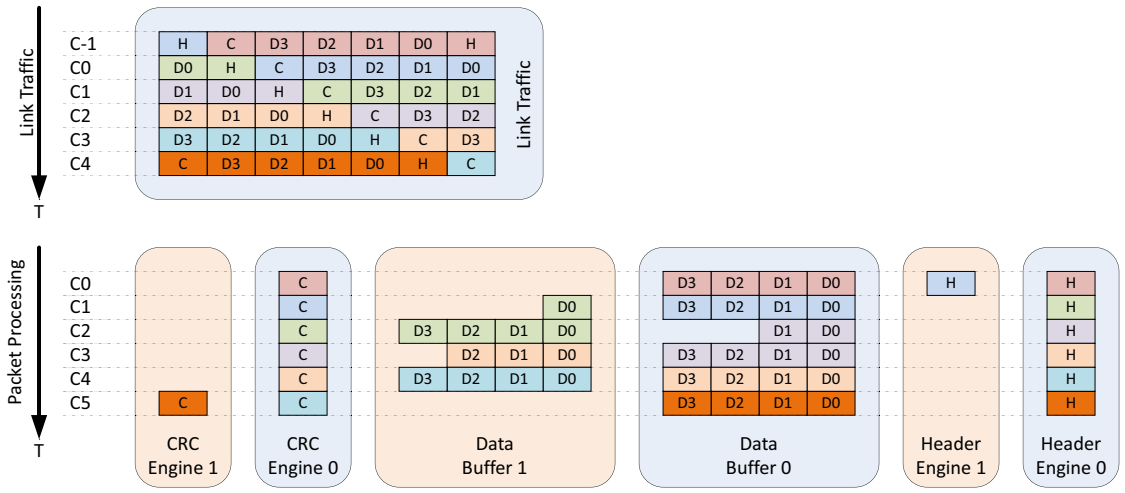


Figure 2-26: Multiple Functional Units

Thus, increasing the data width can have a negative impact on the primary feature latency as well as for the frequency attribute, and also the bandwidth can be influenced in a negative way.

2.2.1.1 Alignment

A point with a tremendous influence on data dependencies is the alignment of data. A protocol defines different types of widths. Those widths are word width, data width, header size, and packet length. In order to reduce the hardware effort it is advantageous that the different widths of a protocol have an adjusted alignment. The solution which provides the least hardware effort is if all larger width are always a binary multiple of all the smaller widths. An example is given in figure 2-27.

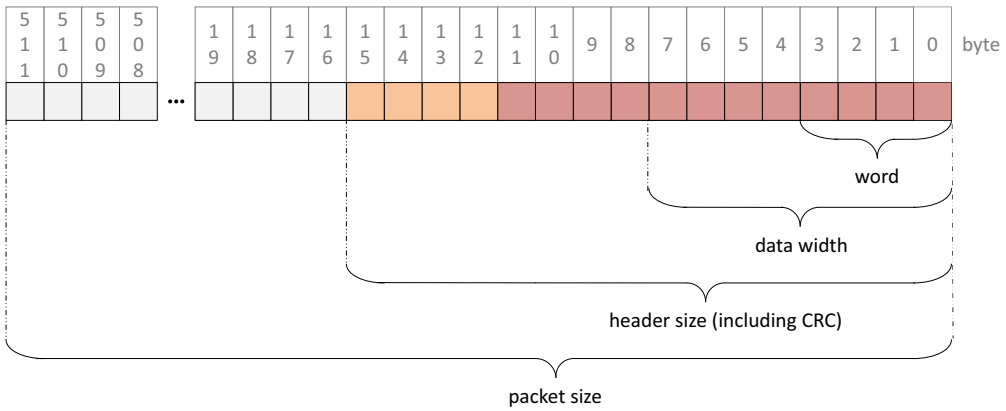


Figure 2-27: Width Alignment

If such an alignment is not met the hardware effort will increase. For example, the start of a packet can move inside of the data width to different locations.

In addition, the packet format should be chosen in a fashion that the different fields are aligned and leveled to ease the decoding. This means for field alignment, that a field with the same meaning for different packet types should have the same position inside of the different header types. In case for leveling this means if packet types have fields with different meanings but reserved space inside the header they do not have to use the same header position. In the following sub-chapters more details will be described.

Packet Alignment and Structure

The different control packets of a protocol should be aligned to each other to reduce the multiplexing effort. This means, that if control packets have fields with the same processing destination, they should have the same position inside the different control frames. A good example for this is the command field which identifies the packet type. If this alignment is disregarded, the multiplexing effort will unnecessarily increase as more possible sources for the same field have to be handled. Furthermore, it is not only the alignment that is of importance, but also how the header is structured. It is advantageous that if fields differ between control frames, that these fields are not arranged in a simply consecutive order. If they were consecutive the beginning of the control frame would be dense with different field types, which results in a high wiring effort, whereas the end of the control frame is barely used. Not all packet types may need all the space inside of the control frame and therefore reserved fields of the different packet types can be used to balance the multiplexing effort across the positions of the data width. This kind of balancing is complicated because it should not influence the processing time. If the whole control frame is received inside one clock cycle, then there is no problem. However, if multiple clock cycles are needed to receive one frame, then the fields have to be arranged in a way that the processing is not delayed regardless of the multiplexing effort. Such a delay can happen if a field, which is needed to start processing the control frame, is shifted to the end of the control frame which will only be received in a following cycle.

Width Alignment

From a hardware perspective there are three important sizes influencing a protocol that have an impact on data dependencies. Those sizes are the data path width, the protocol granularity, and the packet size. If those sizes are not aligned to each other, data dependencies will increase significantly. This happens because of the additional hardware effort already mentioned in “Data Dependencies” on page 31. Furthermore, if the relation among them is wrongly chosen, the application area of the protocol is very limited as it is inflexible in regards to changes of bandwidth or frequency. For example if a given frequency cannot be reached on a device increasing the data width will reduce the required frequency, but the additional complexity will limit the achievable frequency. In order to keep the hardware effort minimal, two rules can be followed. First, all sizes should be a power of two, and second, the data path width should be smaller or equal to the protocol granularity. As long as there are no physical restrictions using a power of two should be no problem. The only problem with those two rules is that bandwidth can be wasted if a too large size for the data path width is used and therefore the resulting protocol granularity causes a framing, which cannot be completely utilized. This can also happen, if a smaller size does not have enough space, and the next step of power of two offers two much space.

2.2.2 Frequency

The frequency of a design influences the latency that a protocol introduces. The achievable frequency is closely coupled with the data dependencies. If there are more data dependencies to solve inside one clock cycle, the clock cycle has to be longer and therefore the frequency has to decrease because of the necessary amount of hardware. To keep the latency low a small data width is of advantage, but this will have a negative influence on bandwidth or an extreme requirement to the pipeline frequency. Most protocols can operate over a range of frequencies. Beside power saving reasons and other limitations from the design environment, the frequency will be chosen as high as possible to reach better throughput and lower latency. As already mentioned in the chapter “Bandwidth” increasing the frequency is only achievable with substantial changes in design and costs. Other

ways to increase the frequency, like pipelining, have negative effects on latency.

2.2.3 Data Width

Reducing the latency can also be achieved by increasing the data width. In this case the latency is only reduced by the number of pipeline-stages a packet occupies traveling through the network. If a packet would need 32 pipeline-stages and then the data width is doubled, it would need 16 pipeline-stages and therefore 16 clock cycles less. This only works as long as increasing the data width does not influence the frequency in a negative way when pipeline-stages are merged together. As soon as the frequency is influenced negatively, increasing the data width can have a negative influence on latency as the gain of occupied pipeline-stages is insignificant compared to the frequency reduction.

2.3 Message Rate

One indicator of protocol efficiency is, is the number of packets that can be transferred in a given period. This indicator is called message rate. While a high message rate is important it is not possible to judge the efficiency of a protocol solely by its message rate. The message rate can be artificially increased by simply defining a smaller packet size, this results in more packets per period, but then the effective bandwidth decreases, which is negative for the overall performance. On the other hand, in some use cases a low message rate may be sufficient or more important. In this case, the effective bandwidth is higher, but the fault tolerance and the availability of resources are negatively influenced. Therefore, the goal of optimizing the message rate is to fully utilize the theoretical maximum bandwidth of the given protocol and not to artificially increase the number of messages.

The message rate between sender and receiver is closely coupled to the bandwidth. Bandwidth is considered the physical possibility to transmit a defined amount of data from a sender to a receiver in a certain period of time. If the packet size and structure are defined, the message rate is the reference point to ensure that the given bandwidth is used efficiently. If the message rate is maxed out, the bandwidth is totally utilized with data and no idle traffic is transferred. In case that the message rate does not utilize a given bandwidth, the

theoretical bandwidth is still identical but the effective bandwidth drops. There is only one attribute which influence the message rate if everything else was done to maximize the bandwidth, and this is the availability of resources.

2.3.1 Availability of Resources

Keeping the utilization of a link high is important for the system performance. One instrument to do so are buffers at the receiver which store the data until it can be processed. Therefore, the sender must be informed about the currently available buffer space which is typically done with credits. In order to ensure that traffic can always be processed it is necessary that there are sufficient resources. For example if the receiver has not enough buffer space to receive further packets, the sender is not able to transmit data and therefore stalls. This can easily happen if a packet is transmitted and the seven steps shown in figure 2-28 take longer from the first transmission to the release of a credit than to fill the buffer space. If the last buffer space is used, the link state becomes idle and the bandwidth during this time is wasted. In step 1-2 the packet is sent and received and in step 3-7 the credit is released.

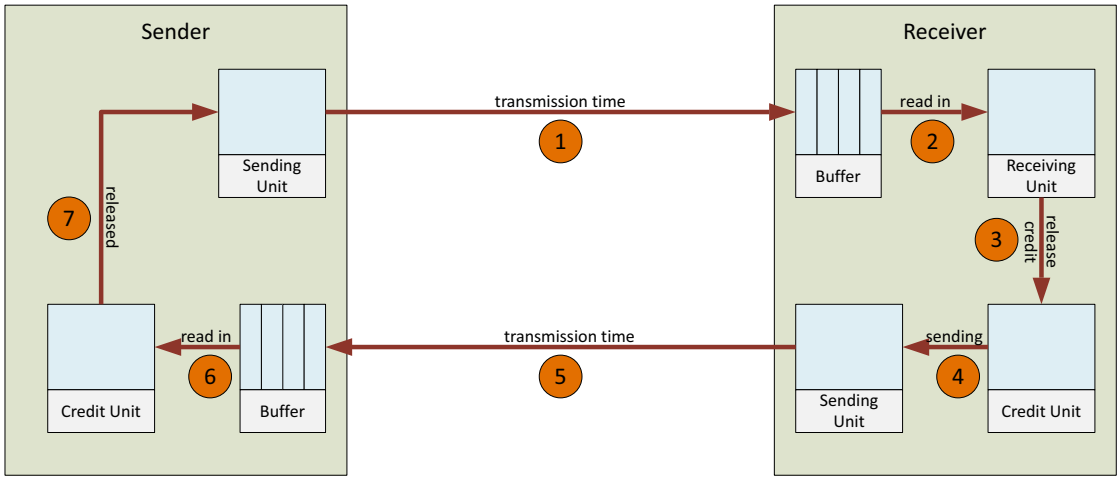


Figure 2-28: Credit Release

In addition, the identifier a packet is tagged with has to be large enough to hide the round trip latency of the furthest destination. This means, that a response of a packet can be received without starvation of tags. Of course, there is a trade-off between header size and scalability of the network. A small identifier may not be sufficient and therefore waste

bandwidth by stalling further transmissions. A large identifier increases the header size, which is negative for the bandwidth. However, the diameter of a network can be larger and still be used efficiently. Nevertheless, if such a large diameter in not needed or the negative influence of the bigger header size would be too large, a smaller identifier should be used.

If a request is transmitted, there is a chance that the response will be delayed. In this case, the identifiers can run out. Enlarging the identifier for this reason makes no sense as the delay cannot be estimated, but at least it should be guaranteed that the round trip latency with no additional delay can be hidden. If the worst case for the round trip is assumed there is additional margin if typical traffic is delayed.

From a bandwidth perspective it is important that a packet which is stalled must not be able to block other unrelated traffic from transmission. In order to ensure this, only data should be transmitted which the receiver can process without blocking. Therefore, it is necessary that data transmission can be interrupted at predefined boundaries. If at any point the transmission is stopped, independent data, which needs the same resource, can be transmitted. This can be handled for example by virtual channels.

The message rate can not only be optimized by ensuring that resources are not blocked, but also by adding additional features like e.g. adaptive routing [30]. If there is a hot spot inside a network, which is used by most of the traffic because of the routing algorithm, but there are additional ways to route a packet safely, it would be of advantage to include a rerouting mechanism inside the protocol. The unused bandwidth of other links can then be used and therefore the overall message rate increases. A simple example is given in figure 2-29.

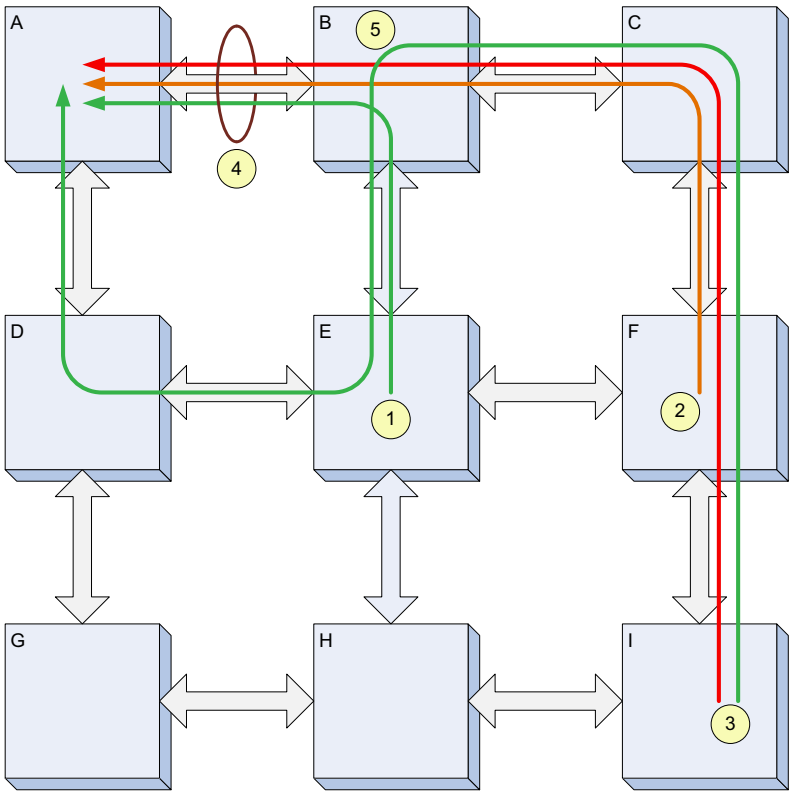


Figure 2-29: Rerouting

1. Data is transmitted from node E to node A
2. Data is transmitted from node F to node A that also uses the link between node A and node B
3. Data is transmitted from node I to node A
4. A highly used link is detected and it is checked if a rerouting is possible
5. The data stream from node I to node A is rerouted over the unused links of the nodes E and D

The implementation of such a mechanism is difficult because of many reasons. One, and maybe the most important, is that the additional hardware effort for adaptive routing is immense. Most of the former and modern protocols for HPC, like Infiniband [18], Myrinet [31], and QsNet [32] use deterministic routing as stated in [33]. With deterministic routing there is no chance for one packet to overtake another while it travels through the network. Thus, no reordering of packets is necessary. In [33] a solution with low hardware complexity for reordering in a network with adaptive routing is proposed. A reordering

buffer is introduced which reorders the packets and if it is full drops the packet with the highest received tag of a data stream. Nevertheless, this solution needs a large amount of buffer space inside a chip, which can have a negative influence on speed and increases the chip size and therefore reduces the yield. It seems to not be suitable for large networks as the reordering is handled with one buffer and if many packets from different nodes are received at one time the reordering and the corresponding pointer logic gets complex. Also, the number of packets in one data stream that can be reordered decreases. In addition, the traffic in the network increases because of the acknowledgement scheme and necessary retransmission of dropped packets.

Additional hardware complexity will be introduced to realize the routing algorithm itself. As deterministic routing can be realized by a simple Finite State Machine (FSM) or simple routing table while adaptive routing needs more hardware resources. If an adaptive approach is used, the needed buffer space increases. One class of those algorithms are the hop algorithms [34]. The simplest version of the hop algorithms just adds a virtual channel for every hop taken. Thus, the buffer space grows tremendously with the size of the network. If D is the dimension of the network, the needed number of buffers to avoid deadlock is D+1. Even if a more efficient version of the hop algorithm is used, like the negative-hop algorithm, the needed buffer space is still immense. In order to reduce the hardware effort further a buffer optimized algorithm can be used. Cypher and Gravano [35] proposed two algorithms with fully adaptive deadlock free routing which have been proven to be optimal in respect to buffer space [36]. The hardware effort is reduced but every time the packet changes the virtual channel the adaptivity is reduced and no adaptivity is given in the last virtual channel.

Furthermore, an adaptive routing algorithm has to be chosen, which is deadlock free. One way to avoid a deadlock was pointed out in [37] by using a routing with no circular dependencies. A solution is to use an algorithm that does not allow any circular dependencies, another one is to introduce a virtual channel every time a circular dependency occurs. Two options are possible. One is a routing algorithm that allows only minimal paths like Planar Adaptive Routing [38] and the other is an algorithm that also allows also non-minimal paths like the Turn Model [39]. If not only the traffic should be balanced by default,

a feedback of the links is needed to decide which path should be chosen. If the path is minimal the information of a congested path becomes more important the closer the packet comes to the receiving node, because the number of possible routes to the destination shrinks if the packet gets closer. The same applies to non-minimal algorithms, but there might be more options depending on the used algorithm.

An optimal hardware mechanism that fulfils all the needs for all means has to be deadlock free, adaptive, non-minimal, ordered, and with feedback. All of those needed features result in additional hardware effort that has to be considered.

2.4 Fault Tolerance

A primary feature that is in contrast to all other primary features in nearly all cases is fault tolerance. It reduces the bandwidth and message rate because of the overhead that is needed to secure data. Furthermore, latency may increase due to the additional processing time that is required for checking. Obviously, there are no systems that always guarantee correctness. Errors, such as bit flips, will occur during run-time. Those errors have to be handled so that the system will still function and not crash. Thus, fault tolerance is essential for a system to guarantee its functionality, correctness, and availability. For large systems the probability of errors increases with the number of nodes.

Errors may occur because of various problems. Single Event Effects (SEE) can be induced by radiation like heavy ions, protons, and neutrons, by crosstalk, and manufacturing defects. Those errors have different results like hard errors and soft errors. Hard errors are not recoverable because some hardware is permanently destroyed. Those errors are called Single Event Burnout (SEB). Soft errors are non-permanent and can be fixed by reset, power cycle, or special recovery mechanisms. They can be distinguished into many different classes like Single Event Transient (SET), Single Event Upset (SEU), Multi Cell Upset (MCU), Multi Bit Upset (MBU), Single Event Functional Interrupt (SEFI) and Single Event Latch-up (SEL). A briefly overview is given in table 2-2, for detailed information please refer to [40].

Error Type	Description
SET	The logical path of the data is influenced by an effect.
SEU	The storage of the data is influenced by an effect.
MCU	Many cells are influenced by an effect.
MBU	Many bits are influenced by an effect.
SEFI	Effect results in a longer period of miss function.
SEL	Effect results in a huge power consumption and possibly in a SEB.

Table 2-2: Single Event Effects

How errors like SEUs and crosstalk influence network on chip switches was analyzed in [41]. Two error models have been used. One influences the value that is stored into a register and the other one delays the signal that has to be stored. In a simulation of 15.2s, 18,909 SEU and 17,108 crosstalk fault were injected. The results are shown in table 2-3. This makes an error handling essential as no faultless data transmission can be assured under all circumstances.

Fault Type	Fault Location	Fault Effects							
		Packet Routing Error		Payload Error		Router Crash		No Effect	
		#	%	#	%	#	%	#	%
SEU	Arbitration Priority Registers	0	0.00	1	0.42	1	0.42	238	99.17
	Arbitration FSM State Register	46	38.33	3	2.50	49	40.83	22	18.33
	FIFO Buffer	80	5.54	168	11.63	24	1.66	1,173	81.18
	FIFO FSM State Register	0	0.00	18	15.00	8	6.67	94	78.33
Crosstalk	NoC Local Link	6	4.84	82	66.13	10	8.06	26	20.97
SEU + Crosstalk	Arbitration Priority Registers + link	48	4.03	313	26.30	64	5.38	765	64.29
	Arbitration FSM State Register + link	330	55.46	38	6.39	144	24.20	83	13.95
	FIFO Buffer + link	1,264	8.66	4,447	30.45	791	5.42	8,102	55.48
	FIFO FSM State Register + link	56	9.41	129	21.68	93	15.63	317	53.28

Table 2-3: Fault Injection Results [41]

As already mentioned in the chapter “Bandwidth” there are two requirements that are needed for fault tolerance. In the first place, an error needs to be detected. After the error has been detected, there must be a mechanism that is responsible for its correction. There are two ways to successfully eliminate an error that has occurred during run-time. One is forward error correction (FEC) and the other one automatic repeat request (ARQ). Figure 2-30 depicts the different possibilities for error handling.

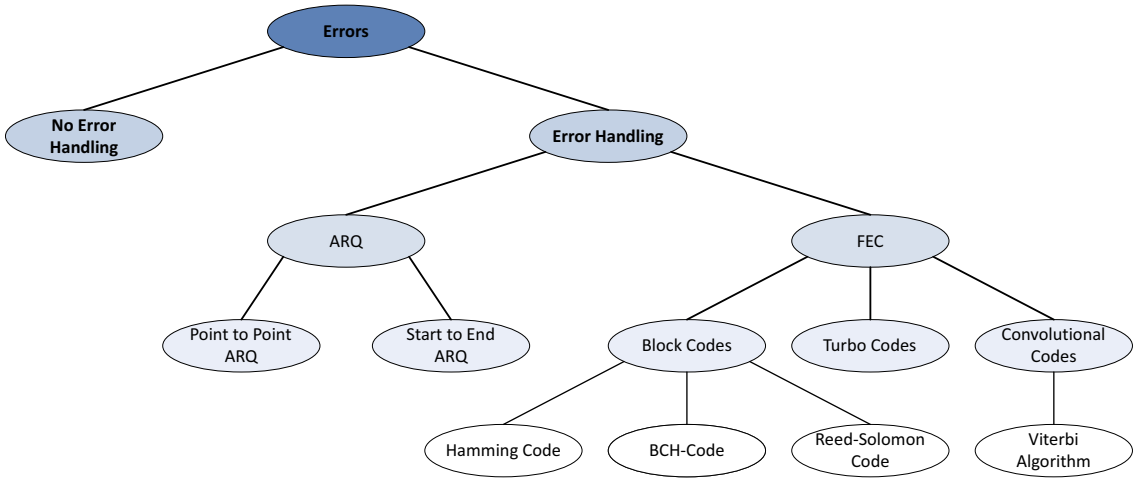


Figure 2-30: Error Handling

FEC is a method in which some redundancy is added to the data so that it can be checked at the receiver and errors can be corrected. Block codes are one solution. Hamming code [42], BCH code [43], and Reed-Solomon [44] code are the most common types of block codes. If it is sufficient to be able to correct only one bit error and to detect 2 bits errors, a Hamming code can be used as it is efficient from a hardware perspective. BCH and Reed-Solomon have to be used if more than one bit error has to be corrected.

If an ARQ mechanism is used it is not necessary to fix an error at the receiver because the packet is retransmitted. Therefore all the additional redundancy added to the packet can be used to detect an error. In this case typically a Cyclic Redundancy Code (CRC) is used. The CRC algorithm can be described as a polynomial division. Therefor a generator polynomial has to be chosen. There are huge differences between the different possible polynomials in respect to the hamming distance they provide. Some evaluations can be found in [45][46][47]. The polynomial can be written in a binary notation were the single bits are the coefficients. For example, the polynomial $X^5+X^2+X^1$ would be written 100110. The data is divided by the generator polynomial and the remainder is transmitted with the data. At the receiver the data and the remainder are also divided by the generator polynomial. The data is correct if the remainder calculated at the receiver is 0. In hardware the CRC calculation can be implemented by a Linear Feedback Shift Register (LFSR. An example is shown in figure 2-31. The switches are closed if the binary notation of the polynomial has a 1 at the same place, otherwise it would be open.

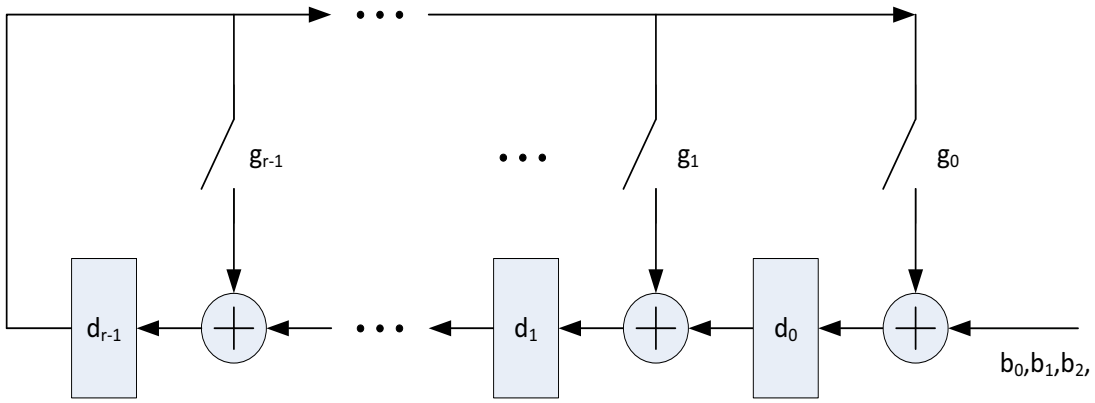


Figure 2-31: CRC as LFSR [48]

The most crucial data that needs protection is the control information. If this data is erroneous, the results can be e.g. misrouted packets, the loss of knowledge where a packet starts, or accessing the wrong address region. Therefore, it must be ensured that the control information is only used if it is already checked or processing it will not lead to an error that cannot be recovered. Checking control information at an early stage is possible if the corresponding security bits are sent right after the control information. As the control information should be a relatively small portion of the complete packet, the difference between using a ARQ or FEC is not extremely large from a bandwidth point of view. For faster processing, it would be of advantage to protect the control information with FEC, so if an error occurs time intensive retransmission can be avoided. Thus, the error can be corrected at the receiver and the data can be processed further.

In common protocols like HT and PCIe, the data of a packet is checked via a CRC, which is transmitted at the end of the packet. This seems to be useful as protecting most of the packet with a mechanism like FEC is too inefficient because of the additional overhead for redundancy. Due to increasing operation frequencies, the error rate does increase as well. At the beginning HT used a CRC every 512 bit times for error detection. As AMD changed the requirements of the HT to adapt higher link speeds for newer generations of CPUs, a retransmission mechanism became mandatory. Keeping retransmission efficient, a per packet CRC had to be used because the buffer space and the retransmission time for the complete 512 bit times would be inefficient. Nevertheless, the restrictions of the protocol cause a loss of bandwidth. The header is so packed that there is no space for addi-

tional protection and the word width is 32 bits, so even if a smaller CRC would be sufficient to protect a maximum sized packet it cannot be used, as the alignment of the data stream would be lost.

Bit Error Rate	1.00E-007	1.00E-008	1.00E-009	1.00E-010	1.00E-011	1.00E-012	1.00E-013	1.00E-014
Bandwidth in GB/s	2							
Packet Size in Bytes	1024							
FEC Packet Overhead in Bit	33							
Retrans. Packet Overhead in Bit	32							
Retrans. Overhead in Packets	8	16	32	64	128	256	512	1024
data width	32							
Error Rate for 32 Lines	3.20E-006	3.20E-007	3.20E-008	3.20E-009	3.20E-010	3.20E-011	3.20E-012	3.20E-013
Clock Cycles per Error	3.13E+005	3.13E+006	3.13E+007	3.13E+008	3.13E+009	3.13E+010	3.13E+011	3.13E+012
Frequency in HZ	536870912							
Seconds per Error	0.00058	0.00582	0.05821	0.58208	5.82077	58.20766	582.07661	5820.76609
Packets per Error	1221	12207	122070	1220703	12207031	122070313	1220703125	12207031250
FEC 33 in KB	4.91	49.17	491.73	4,917.38	49,173.83	491,738.32	4,917,383.19	49,173,831.94
Retrans 8 in KB	12.76	55.68	484.83	4,776.37	47,691.71	476,845.15	4,768,379.58	47,683,723.82
Retrans 16 in KB	20.76	63.68	492.83	4,784.37	47,699.71	476,853.15	4,768,387.58	47,683,731.82
Retrans 32 in KB	36.76	79.68	508.83	4,800.37	47,715.71	476,869.15	4,768,403.58	47,683,747.82
Retrans 64 in KB	68.76	111.68	540.83	4,832.37	47,747.71	476,901.15	4,768,435.58	47,683,779.82
Retrans 128 in KB	132.76	175.68	604.83	4,896.37	47,811.71	476,965.15	4,768,499.58	47,683,843.82
Retrans 256 in KB	260.76	303.68	732.83	5,024.37	47,939.71	477,093.15	4,768,627.58	47,683,971.82
Retrans 512 in KB	516.76	559.68	988.83	5,280.37	48,195.71	477,349.15	4,768,883.58	47,684,227.82
Retrans 1024 in KB	1,028.76	1,071.68	1,500.83	5,792.37	48,707.71	477,861.15	4,769,395.58	47,684,739.82
Diff Retrans 8 FEC 33	7.85	6.51	-6.90	-141.01	-1,482.12	-14,893.16	-149,003.61	-1,490,108.12
Diff Retrans 16 FEC 33	15.85	14.51	1.10	-133.01	-1,474.12	-14,885.16	-148,995.61	-1,490,100.12
Diff Retrans 32 FEC 33	31.85	30.51	17.10	-117.01	-1,458.12	-14,869.16	-148,979.61	-1,490,084.12
Diff Retrans 64 FEC 33	63.85	62.51	49.10	-85.01	-1,426.12	-14,837.16	-148,947.61	-1,490,052.12
Diff Retrans 128 FEC 33	127.85	126.51	113.10	-21.01	-1,362.12	-14,773.16	-148,883.61	-1,489,988.12
Diff Retrans 265 FEC 33	255.85	254.51	241.10	106.99	-1,234.12	-14,645.16	-148,755.61	-1,489,860.12
Diff Retrans 512 FEC 33	511.85	510.51	497.10	362.99	-978.12	-14,389.16	-148,499.61	-1,489,604.12
Diff Retrans 1024 FEC 33	1,023.85	1,022.51	1,009.10	874.99	-466.12	-13,877.16	-147,987.61	-1,489,092.12

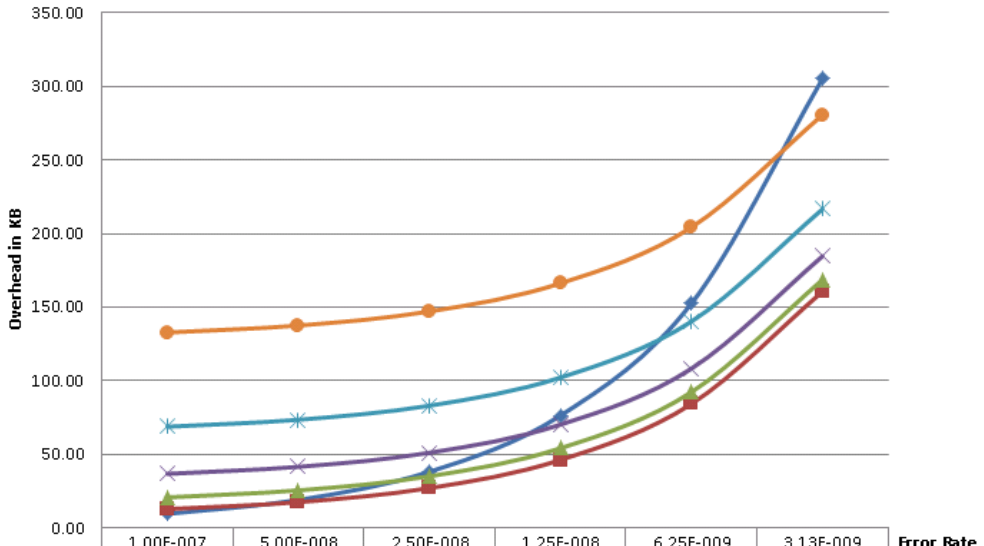
Table 2-4: Fault Tolerance Overhead

Table 2-4 shows the efficiency of FEC versus ARQ. It is divided in three blocks. Block number one contains the parameters, block number two shows the overhead of the different error correction types, and block three is the ARQ overhead subtracted by the FEC overhead. It is shown, that even at a minimal overhead difference by 1 bit per packet, and a high retransmission rate of multiple 1024 bytes long packets, the overhead of retransmission needs less bandwidth at an error rate of 10^{-11}. The green highlighted parts show when FEC is more efficient than ARQ. Orange shows the error rates when ARQ starts to be more efficient than FEC from this point the values are always highlighted red. However, using FEC for parts of the packets can be still of advantage. If there is enough space inside a header to protect it with FEC the processing can be safely started after checking without waiting for the rest of the packet. In this case, the payload could be erroneous but the control information can be used without any harm to the system. Of course, it is required that the data still has to be checked and if it is erroneous writing it to main memory must be prevented.

2.4.1 Retransmission Overhead

Some constraints have to be known to choose the correct way to handle an error. It must be estimated how often an error will occur, and if an error occurs if only one bit is influenced or if multiple bit-errors cloud happen, to choose a useful amount of protection.

If nearly no errors happen during the run-time of the system the influence of error handling is insignificant. This means that the needed per packet bandwidth of the mechanism to detect an error should be as minimal as possible. Whereas the mechanism to solve the error carries nearly no weight and can therefore consume a relatively high amount of bandwidth. On the other hand, if an error occurs relatively often it may be of advantage to invest more bandwidth per packet, if no retransmission is needed instead. In this case, retransmission would consume a lot more bandwidth. The point when to switch from one mechanism to another depends on three things. The most important one is the error rate, because for very high and very low error rates it is clear what kind of mechanism has to be chosen. The second important point that influences what mechanism should be chosen is how much redundancy overhead is needed for FEC compared to retransmission. The last point is the amount of data that has to be retransmitted to correct the error. If only the faulty packet has to be retransmitted at one link, the loss of bandwidth is relatively low. In this case, retransmission has nearly no additional overhead. If the whole data stream transmitted after the faulty packet has to be retransmitted, the impact on bandwidth is relatively high. But retransmission after the faulty packet could be needed to maintain packet ordering. The worst case from a bandwidth perspective would be if a retransmission has to take place between the original starting-point and the final destination, instead of only repeating the erroneous hop. In this case, not only the bandwidth of the hop where the error occurred would be lost, but also the bandwidth of the hops where the packet was transmitted correctly. Figure 2-32 shows that for a high error rates and an additional overhead of 0.34% per packet for FEC a very high amount of data could be retransmitted until the FEC is more preferable.



	1.00E-007	5.00E-008	2.50E-008	1.25E-008	6.25E-009	3.13E-009
FEC 64	9.53	19.07	38.14	76.29	152.58	305.17
Retrans 8	12.76	17.53	27.07	46.14	84.29	160.58
Retrans 16	20.76	25.53	35.07	54.14	92.29	168.58
Retrans 32	36.76	41.53	51.07	70.14	108.29	184.58
Retrans 64	68.76	73.53	83.07	102.14	140.29	216.58
Retrans 128	132.76	137.53	147.07	166.14	204.29	280.58

Error Rate

Figure 2-32: Overhead Error Correction Methods

The bandwidth that is needed for protection is not only constrained by the bits that are needed to realize it in a minimal way, but also by the granularity that is used to structure the protocol. If the protocol word width is restricted to 32 bits, but a CRC of 16 bits would be sufficient to protect the packet, the other 16 bits would be useless for protection but needed to keep the alignment.

The conclusions are that there are many parameters which interfere among the different features. A balance must be found to reach the needed performance of a protocol and the hardware implementation must still be feasible.

Chapter 3: Off Chip Protocols

In HPC-systems, data has to be continuously exchanged among the components. The exchange of data requires an explicitly defined format between communication partners. Communication between modules on a chip normally takes place in a different form than communication between different chips because of the physical wire properties. Therefore, different types of protocols are needed. The types can be distinguished into on-chip and off-chip protocols. Those protocols are mostly differentiated by the following properties:

- Wire delay
- Wire numbers
- Synchronization
- Framing
- Flow control
- Error handling

The differences are described in the following sub-chapters.

3.1 Wire Delay

The exchange and computation of data between two directly connected components on a chip normally has to take place in one clock cycle. In [49] it is stated that in most cases the wire delay is larger than the gate delay. However, the wire delay of an on-chip protocol is small as the distance between components on a chip can be measured in microns. Frequencies, which typically can be reached in modern technologies, range from 1 to 4 GHz. This means, that the time for the sufficiently pipelined logic and the wire delay fits in a time frame of 1 ns or less. To simulate the wire delay a RC-circuit model is sufficient.

Compared to an on-chip protocol, an off-chip protocol has to be able to connect components over much longer distances up to multiple of tens of meters. A RC-circuit model is no longer sufficient at this point. Therefore, the model of a transmission line is used whereas the signal travels as a wave through the channel. These transmission lines are

usually driven by serializer-/deserializer-blocks which transform a parallel data stream high speed serial data stream and vice versa. Using such high speed transceivers results in additional delay and hardware effort as the transmitter the data path has to be transformed from a wider and slower path down to a narrower and faster path, and reverse at the receiver. Those transformations are time consuming.

3.2 Number of Wires

For on-chip connections, small copper connections in the chip can be used. The structure size of a wire is in the same order of magnitude as the structure size of the logic. Adding or deleting a connection from one processing block to another is insignificant in regards to chip area. Thus, data and control signal forwarding is not restricted to a certain granularity like a data link between chips. Only if fully meshed connections among multiple modules are needed, as for switching structures, wiring problems can occur. In figure 3-1, it can be seen how the metal layers get larger for higher metal layers. The big upper layers are mainly used for interlayer connections and power supply.

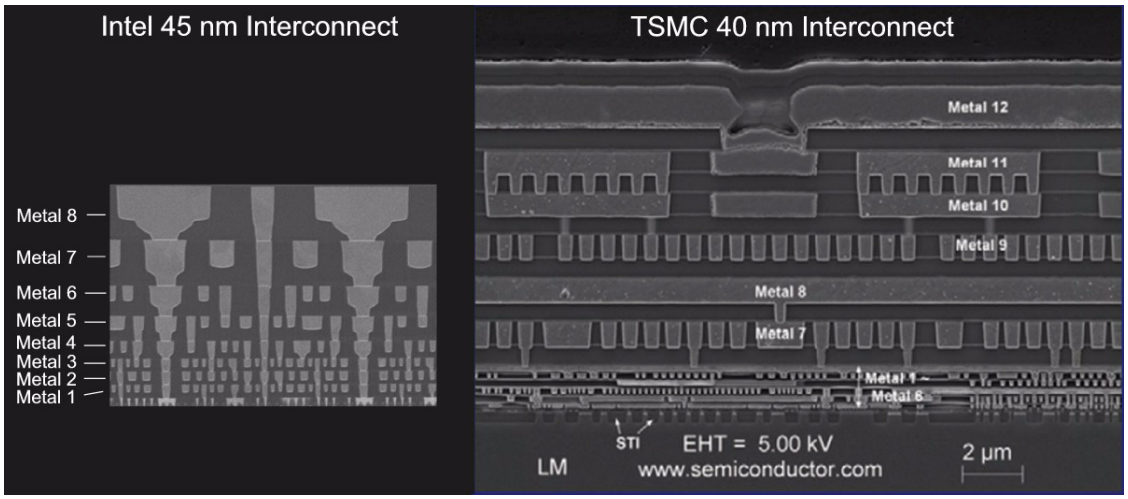


Figure 3-1: Chip Cross Section [50]

For off-chip connections, there exist additional physical restrictions. Compared to the internal connections bigger metal structures for bumps have to be used to leave a chip. Those pads have a size of approximately 60-80um. Compared to the size of the on-chip metal connections this is about a factor of ~1000. In figure 3-2, the size of a bump is shown.

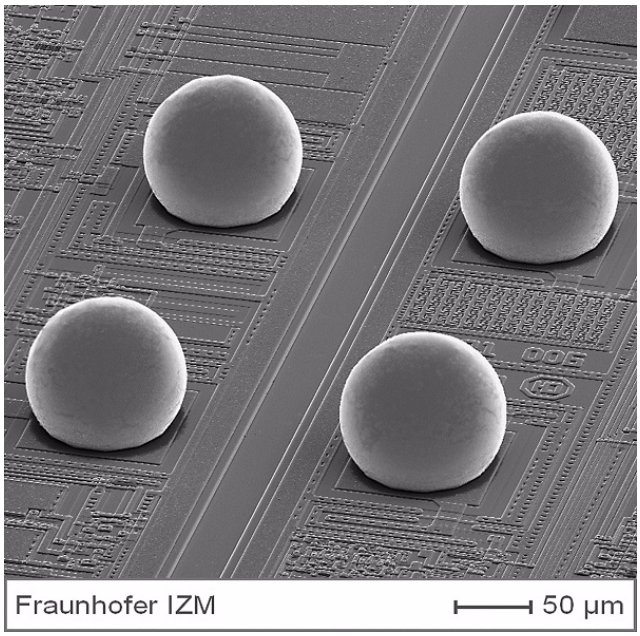


Figure 3-2: Solder Bump Size [51]

With those bumps a chip can be flip-chipped to the package. In most cases, the package provides fewer balls than there are bumps on a chip, which also is a restricting factor. The package, including the chip, can be soldered to the PCB. The PCB is connected to the system via an HT or PCIe connector and to other systems via connectors like Small Form-factor Pluggable (SFP). The connection into the system is additionally limited through the specification of the used protocol. Connections to other systems are limited by the number of connections which can physically leave a node. For example, the limitation of a standard case for external connections is the available size of all brackets. Because all of those above mentioned reasons, the off-chip connections cannot be as wide as the on-chip connections. Therefore, the internal connections of many bits have to be serialized. As the internal frequency of a chip is already relatively high, transceivers with 10 GB/s or more have to be used. Those transceivers need large area of silicon which further restricts the number of possible off-chip connections.

3.3 Synchronization

Between different domains synchronization is needed if the two domains do not use the same clock source. Therefore special logic is needed to ensure that no data is lost or doubled between the two domains.

For on-chip connections, in most cases, no synchronization is needed, as most of the logic will run in the same clock domain. Otherwise, a simple synchronization technique such as two-flip-flop-synchronizers can be used. This is possible because the clock for both logic domains will be derived from the same clock source.

For off-chip connections, the clocks are most likely not derived from the same clock source. Therefore, more advanced techniques such as synchronization FIFOs with one-bit changing codes have to be used. One-bit changing codes ensure that a register value can only be changed by bit position between two clock cycles. An example for an one-bit changing code is the Gray Code [52]. Even if the logic of both communication partners runs theoretically at the same frequency, the clocks of the sender and the receiver can differ by several ppm. Therefore, the clock difference must be accounted for. If the sender runs at a faster clock frequency than the receiver, additional data that has not to be processed by the receiver has to be inserted into the stream so it can be dropped by the receiver to catch up. In the other case, when the receiver is faster than the sender, it must be ensured either that the data stream can be interrupted at any time or all data needed for one processing step will be ready at the processing start. In most cases, the data will be synchronized at the receiver. The clock from the sending domain will be provided by an extra clock lane or derived from the received data signal. Using an extra lane will consume a valuable resource which otherwise could be used for additional bandwidth. If the clock is derived from the data, line coding mechanisms like 8B10B [53] have to be used to guarantee enough signal changes to be able to recover the clock frequency from the signal.

3.4 Framing

If different types of data, like header-, payload-, and data-frames, have to be distinguished for on-chip connections, additional wires can be used for differentiation. For easy decoding purpose, it is of advantage to use a lane for every single type of coding. If many sideband signals have to be used it is possible to switch to a coding scheme to save wires but increase the decoding effort. For example if four wires for a one hot coding is used it can be changed to two wires with a binary coding.

In terms of off-chip connections, additional wires can also be used for differentiation, but at this point they are more expensive, please refer to chapter 3.2. The bandwidth is limited by number of available wires. Therefore, special framing tokens can be used as a start of packet (SOP) and an end of packet (EOP) token. It is of huge advantage if those tokens have a special protection so that they can always be identified inside the data stream even if the framing alignment was lost [54].

3.5 Flow Control

For on-chip connections, flow control can be easily handled by a valid-stop mechanism. The transmitter sends a valid signal as soon as data is available whereas the receiver sends a stop signal if the current data cannot be processed at this time. An example connection is shown in figure 3-3. The different connections can be grouped in three different types, data path, control path, and flow control.

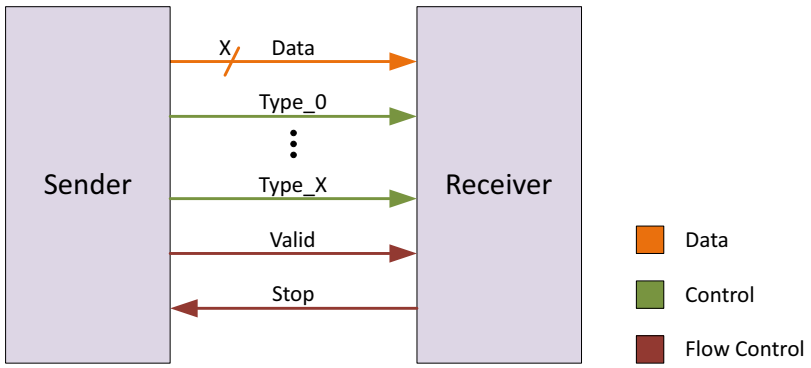


Figure 3-3: Valid Stop Scheme

Table 3-1 shows how the valid stop mechanism works in detail.

valid	stop	description	reaction
0	0	data can be processed but no data available	nothing to do
0	1	data can not be processed and no data available	nothing to do
1	0	data can be processed and data available	data will be processed by receiver, sender needs to change data or valid signal
1	1	data cannot be processed and data available	data connot be processed by receiver, sender needs to keep data and valid signal

Table 3-1: Valid Stop Description

Long distances have to be overcome in case of off-chip protocols. Therefore, some mechanism is needed to decouple the transmission of data from its reception. In this case, a packet based transmission with a credit based flow control can be used. The receiver signals the sender how much data can be received without losing packets. If packets have been sent from the transmitter, the credit counter has to be reduced. As soon as the receiver has processed one packet, it has to notify the transmitter that an additional packet can be transmitted without the danger of data loss.

3.6 Error Handling

Typically no error handling is needed for on-chip connections, as it is very unusual that an error occurs inside the logic. Therefore, an extra error handling is not justified. However, the hardware has to be built in a way that an error does not lead to a system failure and can be always recovered. For example, state-machines have to be built in a way that no invalid states can be reached. Inside a chip, only memories have to be protected.

For off-chip connections, an error handling scheme is mandatory. If data is transmitted over a long cable, the signal integrity degrades over the distance. Therefore, an induced failure is much more likely than inside the chip. There are two ways to protect a connection between two nodes. One is an end-to-end protection and the other one is a link-level protection. This results in additional hardware effort, delay, overhead, and potential bandwidth loss, but it is essential.

3.7 State of the Art Protocol Features

State of the art protocol features are not mandatory but are supported by the most modern protocols. They increase the usability of the protocol and the performance.

3.7.1 Virtual Channels

Virtual channels are widely used in modern protocols. As already mentioned in chapter 2 on page 41 they are used to avoid deadlocks and head of line blocking [55]. Virtual channels only replicate resources which are needed to manage the link, but not the physical connection itself.

This physical connection is shared by all virtual channels. Thus, if one channel is stalled another one can utilize the link without changing the context.

In order to be able to share a physical link it is necessary that a data stream can be interrupted by another one if it does not make any progress. Thus, the protocol must provide defined points in the data stream where the transmission can be stopped without losing data or causing an error. It must be guaranteed that the receiver is able to accept data until a point is reached where the transmitter can interrupt the data stream.

If a credit based flow control is used, the receiver must be able to process received data immediately or to store it until it can be processed. In order to provide a good performance at least the round-trip latency should be hidden. Regarding to the bandwidth of the link and the distance between the sender and the receiver the required buffer size can be very high. Adding a virtual channel to a link makes it necessary to replicate all those structures, which will need relevant chip size. In addition, the hardware effort will increase with additional virtual channels, as the arbitration between the different channels has to be managed.

3.7.2 Ordering

In most cases, it is important in which order data is received to guarantee correct system behavior. Therefore, it is of importance how data is transmitted through the system. There is no problem if data is always transmitted strictly in order. But in this case no rerouting can be used and streams traveling through different virtual channels must be independent.
Packet Information

Various information is needed to transmit and process data in a system:

- What is the destination of the data?
- What type of data is transmitted?
- What is the size of the packet?
- Who was the sender?
- Is ordering required?
- Is there enough buffer space?
- How is an error handled?

The most important information is the destination of the data. The destination can be defined by an address, a hop count, or a unique identifier like a node- or unit-identifier.

Next the type of data must be defined. Two types of data can be distinguished, control data and user data. Control data contains information to manage the link connection such as flow control, error handling, or initialization. From the protocol perspective user data contains only raw data which is not interpreted.

It must also be known how much data has to be gathered to process the packet. This can be handled by fixed data size or length information. The fixed length can be given for all data transmitted in the system or defined by the type of packet. For example, all control information could have a fixed length and therefore no additional length information must be given. Otherwise, if the length is not fixed the length of the data must be defined some-

how. This can be done by framing tokens or a length field. With framing tokens, the start end the end of the data is signaled by a unique identifier. The length field can be used to define a number of bytes transmitted in a data stream, but also different granularities can be supported by changing the meaning of the length field to doublewords (DW = 32 bits) or quadwords (QW = 64 bits).

In case a data stream is received by a device the sender must be identified. Otherwise, the receiver cannot differentiate among data streams of different senders. In those cases the data must contain information about the sender. This can be handled by an address or an unique identifier. For non-posted requests like a read request the sender of the request needs a response to complete the transaction. In this case the sender must also be identifiable.

In cases were the ordering of a data stream is not guaranteed ordering information has to be added. That information is needed to clarify which data belongs to which data stream and if it is the next data in the corresponding stream which has to be processed. Thus, a stream number has to be given. It is also of advantage to add information if an ordering to other streams is needed to simplify processing. Additionally a reorder buffer is needed which stores out of order received data until it can be processed. For large systems the size of the buffer is hard to determine as it is unclear how many different streams have to be reordered and how many data can be out of order in one stream.

The data stream must contain flow control information if it is not guaranteed that data transmission is allowed all the time. The receiver must signal the transmitter if a packet was released from the receiving buffer. If different virtual channels are used the credit release must be assigned to the corresponding channel. Thus, additional information is necessary to identify the virtual channel.

Error handling is also an important task, which must be provided. Therefore, additional information must be given to detect and correct an error. A data stream can contain information if an error has occurred while processing a packet but the transmission could not be stopped. This is typically the case if the error occurred while processing the data part

of a packet and the packet header had already been sent. Also a sender can be informed if an erroneous packet was received and has to be re-transmitted.

3.7.3 Buffer Space

For large systems, it is very important that the features provided by a protocol specification can be realized in hardware. From a protocol perspective, it is of advantage to give the user the possibility to distinguish the traffic into many different streams with as many outstanding packets as possible. Three main points have to be taken into account:

- How much buffer space is needed from link output to link input?
- How much buffer space is needed from end to end?
- How many virtual channels are needed?

3.7.3.1 Link Level Buffer Space

At link level it is important to fully utilize the link so that no bandwidth will be wasted. In order to guarantee this, at least the round-trip latency of two directly connected communication partners must be hidden. Therefore, the buffer of the receiver must be deep enough to store all the traffic until the first credit release can be received by the transmitter. In order to calculate the depth some information is needed. The width and the frequency of the link, as well as the largest distance between two communication partners and the physical medium must be known. Also, the time to completely receive one packet and to process the released credit has to be taken into account.

In case that re-transmission is supported by a protocol, data has to be stored at the sender until it is acknowledged by the receiver. This results in an amount of buffer space equal to the VC buffers to hide the round trip latency.

3.7.3.2 End-to-End Buffer Space

End-to-end buffer space is the buffer space needed to match responses to requests. If a sender sends a request to the receiver, the response must be assignable to the request. Therefore, all the matching information corresponding to one packet must be stored at the

receiver. In addition, all information to process the response has to be stored as well. The buffer space for this purpose must be large enough to hide the round-trip latency between the source and the destination of a communication. As the communication can take place not only between two directly connected partners, the number of outstanding requests can be very high. Thus, a trade-off has to be made between the possible distance in a network and the available buffer space in hardware. In this case not only the bandwidth and distance is of importance, but also the size of a packet, as not the whole packet has to be stored but only the needed matching information. If packets are larger, less information has to be stored as fewer packets are needed to utilize the whole connection. The end to end buffer space can be avoided if all needed information to process a response is always transmitted with the request. This would increase the header size and therefore consume additional bandwidth. Thus, it is of advantage to store the data at the requester to gain a better network performance. Many protocols do not realize this in hardware but in software as it is simpler to realize.

3.7.4 Link Initialization

Before an exchange of data can take place between two communication partners, a connection has to be established. First, it must be detected if there is a physical connection. One way to do this is by using a connector detect signal. If a physical connection is available, the detection of the communication partner has to start. This can be done by using a special electrical circuit, using sideband signals, or always checking received data for the start of an initialization sequence. Using a special electrical circuit, like the AC coupled link in HT, enables the communication partners to deactivate the link for power-saving reasons until an initialization is possible. Afterwards, the connection must set up until data can be safely transmitted and received.

3.8 HyperTransport

HyperTransport (HT) [14] is a state of the art protocol which is used for communication closely coupled to the processor. For its communication scheme no protocol translations have to be made. Therefore it provides an efficient way for data exchange and is interesting for a detailed analysis.

HT, introduced on April 2. 2001 by the HyperTransport Consortium, is a protocol which is used to interconnect different components inside one node. In most cases processors are interconnected. In [56] it was shown that connections also between nodes can be accomplished. HT is bidirectional and packet based. The packet based scheme is used to decouple the request from the response. HT is mainly used for communication among AMD processor like Opteron and with bridges.

There are three stages of expansion from HT. They are HT1, HT2, and HT3. HT1 and HT2 only differ by the amount of possible link frequencies, whereas HT3 includes some protocol changes. Therefore, HT1 and HT2 are grouped into a Gen1 specification and Gen3 is the Gen3 specification.

3.8.1 HT Topologies

Three different types of devices are defined in HT. There are bridge, tunnel, and cave devices. Bridge devices are used to spawn a secondary chain which can be connected to other HT devices or different protocols. Tunnels have two active HT connections. Requests are received from the upstream devices. Requests for the tunnel device are extracted whereas other requests are forwarded to lower devices. Caves have only one active HT connection. Packets which are received and do not belong to the cave device are dropped. The different devices are shown in figure 3-4, where "P" stands for primary interface block and "S" for secondary interface block.

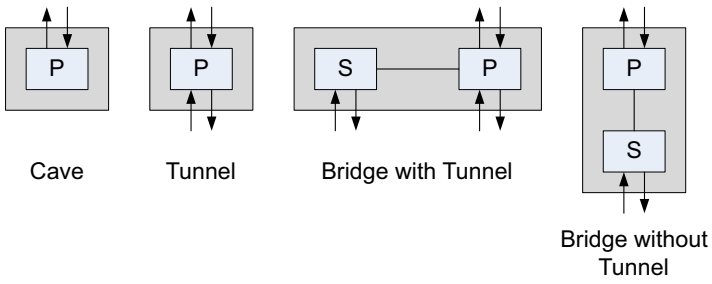


Figure 3-4: HT Topology Elements

HT topologies are organized in chains. The top device hereby initializes the chain. Figure 3-5 shows some examples of different HT topologies.

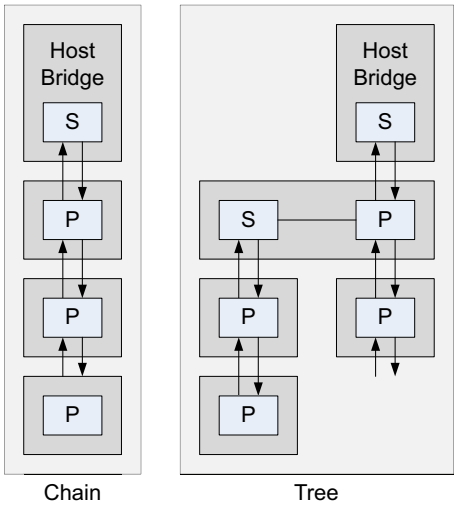


Figure 3-5: HT Example Topologies

3.8.2 Physical Layer

The physical layer of HT consists of two different groups of signals. One is the group with slow clocked sideband signals: PWROK, RESET#, LDTSTOP, and LDTREQ and the other group contains the high speed lanes clock (CLK), CTL, and a combined Control, Address, and Data (CAD) bus. The sideband signals are used for reset, initialization, and power management whereas the high-speed lanes are used to transmit clocks, control information, and data. Those high-speed signals work in a frequency range from 200 MHz double data rate (DDR) up to 3.2 GHz DDR. All possible HT frequencies are shown in table 3-2.

Transmitter Clock Frequencies			
MHz DDR	HT-Speed	MT/s per lane	
200	HT200	400	
300	HT300	600	
400	HT400	800	
500	HT500	1000	
600	HT600	1200	
800	HT800	1600	
1000	HT1000	2000	Gen1/Gen3 Border
1200	HT1200	2400	
1400	HT1400	2800	
1600	HT1600	3200	
1800	HT1800	3600	
2000	HT2000	4000	
2200	HT2200	4400	
2400	HT2400	4800	
2600	HT2600	5200	
2800	HT2800	5600	
3000	HT3000	6000	
3200	HT3200	6400	

Table 3-2: HT Frequencies

In order to establish a link between two HT capable devices a certain number of connections are needed. The minimal required connection is PWROK, RESET#, two CAD lanes, one CTL lane, and one CLK lane. A valid set of CAD, CTL, and CLK in both directions are needed to establish a link. The sideband signals are only transmitted from the upper device in the chain. Several configurations of connections are allowed. CAD can be 2, 4, 8, 16, and 32 lanes wide. A bundle of eight CAD lanes is called a byte-lane. Every byte-lane needs its own CLK lane. At Gen1 speeds, only the first byte-lane needs a CTL. To fulfill the Gen3 specification every byte-lane also needs an associated CTL lane. Figure 3-6 shows a detailed view of the theoretical combinations of lanes for 2, 4, 8, 16 and 32 bits wide links. PWROK and RESET# is always required, LDTSTOP is mandatory for x86 systems, LDTREQ is optional.

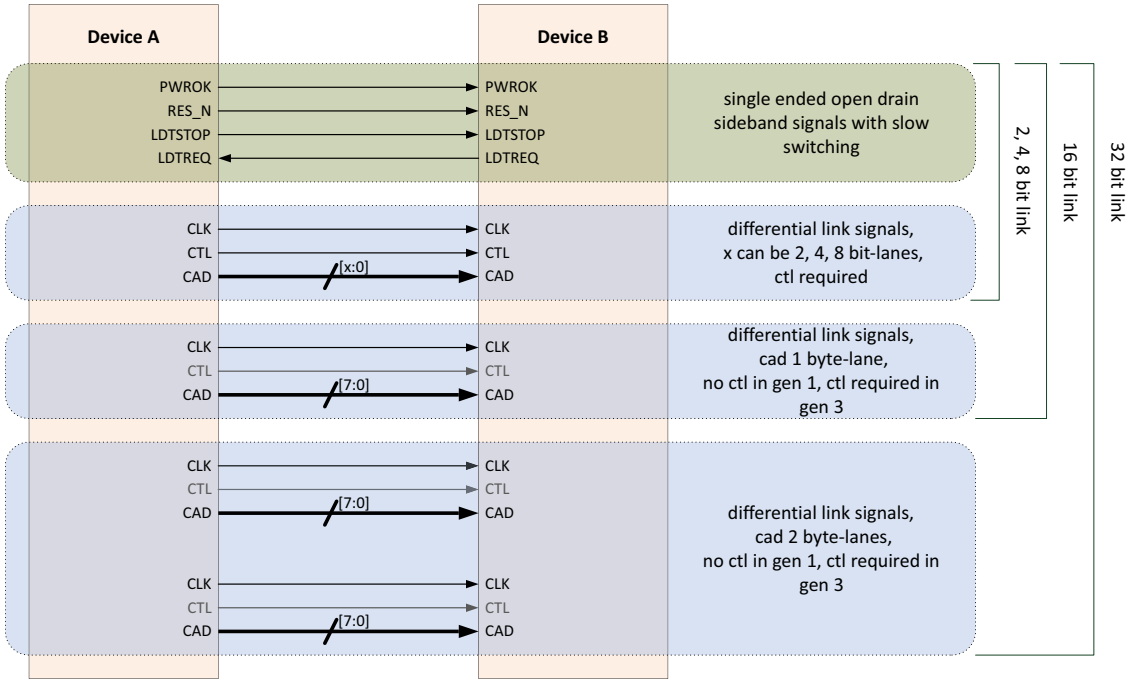


Figure 3-6: HT Link Configurations

Compared to Gen1, CTL has a different meaning in Gen3. In Gen1 CTL determines the framing of HT and distinguishes if control information or data is transmitted at the CAD lanes. Thereby, 32 bits of CAD information is always defined together by CTL. This means that only if the full HT link width of 32 lanes is used, every bit of CTL has a unique meaning. If the link width is smaller than 32 lanes, multiple bits of CTL must have the same meaning as they belong to the same 32 bits of CAD information. If CTL is asserted, the CAD lanes contain control information such as special packets or packet headers. Otherwise, if CTL is deasserted CAD contains data.

In Gen3 CTL has more encodings. First, more CTL information is given. Every doubleword of CAD is defined by 4 bits of CTL. This can be realized by giving every byte-lane of CAD a corresponding CTL lane. Thus, at a full link width of 32 lanes, 4 bits of CTL are received. If the link width is less than 8 lanes, multiple bits of CTL have the same meaning. The used encodings of the CTL signal are shown in table 3-3.

	CTL[3:0]	Description
Gen1	XXX0b	part of data frame
	XXX1b	part of command frame
Gen3	1111b	part of command frame
	0111b	part of inserted command frame
	1100b	CRC for command with data
	0011b	CRC for command without data
	0000b	part of data frame

Table 3-3: CTL Coding

At Gen1 CTL is useful for error detection, as the data length of a packet has to match with the behavior of the CTL signal. Data frames, which are too long, can be easily found, whereas finding data frames that are too short is complex because of packet insertion. Packet insertion allows embedding a control packet inside the data payload of another packet. The CTL signal can be also used for framing reasons, but a start of a new control frame can only be found if a data frame was transmitted before. Therefore, for the relatively low benefit, a relatively high amount of bandwidth of at least 2.7% is used, which is the bandwidth of one lane at Gen1 used for CTL.

The information delivered by CTL at Gen3 is more useful. Besides the abilities already given in Gen1, it can be used to gather full packets without too much decoding effort. However, the used bandwidth for CTL at Gen3 is higher by using at least 10%, which are the 4 of the 40 lanes that are used for the CTL lanes.

In order to establish a working link between two HT devices, connections into both directions are needed. It is not necessary that the links are symmetrical, which means the number of the corresponding lanes in the opposite direction can differ, as well as the link speeds. At the beginning, a low-level initialization has to take place. During this sequence it is determined if the link is connected and the size it uses. Gen1 initialization always starts with a link frequency of 200 MHz and the link width can be two up to eight CAD lanes. A switch to a higher lane count of multiple byte-lanes will be performed after the low-level initialization and the exchange of the capabilities of the devices. The low-level initialization is shown in figure 3-7. A simple sequence of asserted and deasserted lanes is used to detect a link and the start point of the valid packets.

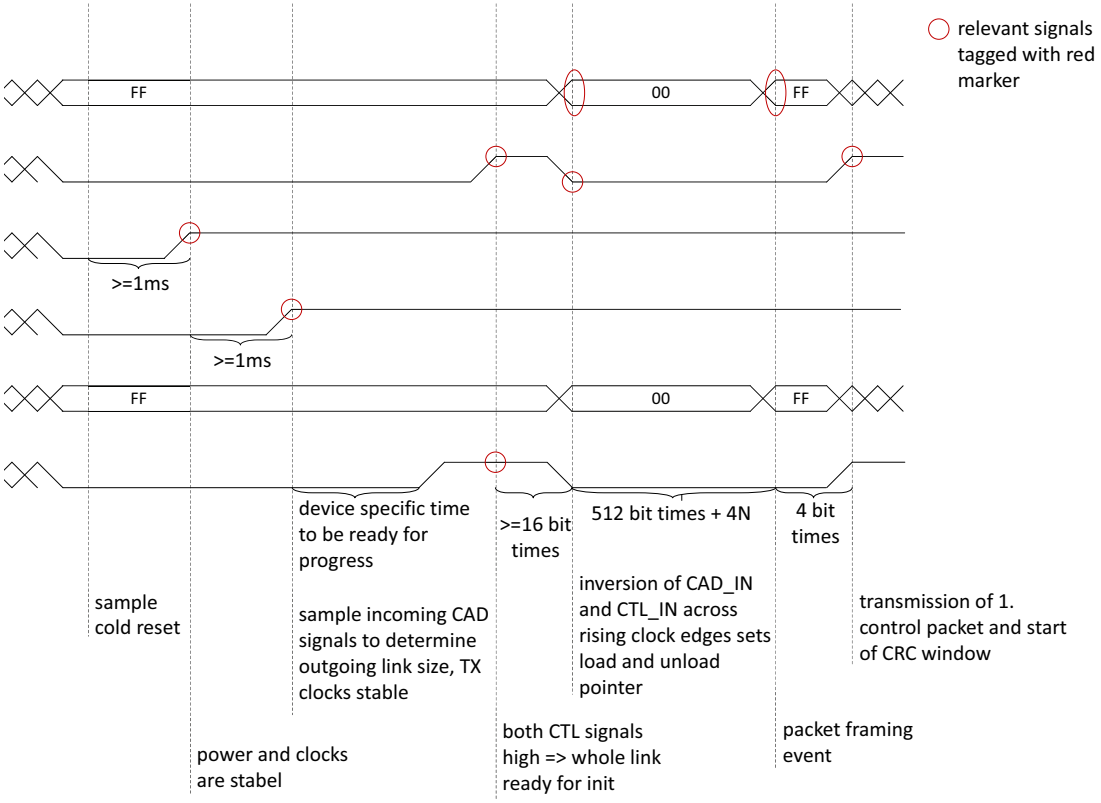


Figure 3-7: Gen1 Low Level Initialization

At Gen3 it must first be determined if the link is DC- or AC-coupled. If the link is DC-coupled, the first initialization phase takes place like in Gen1. Afterwards the link can be reinitialized for Gen3 configuration. Scrambling is used to counteract direct current voltage.

For an AC-coupled link, a special mechanism is used. Gen1 initialization is bypassed. If an AC-coupled link is detected at startup the initialization begins with a frequency of 1200 MHz and the training sequences that are described at [14] chapter 12 page 222 and following pages. To achieve DC-balance 8b10b coding is used. A diagram of the Gen3 startup sequence is shown in figure 3-8. After the possible low-level initialization and a link restart different training patterns are sent to align the data of the different lanes to each other and to determine the start of valid packets.

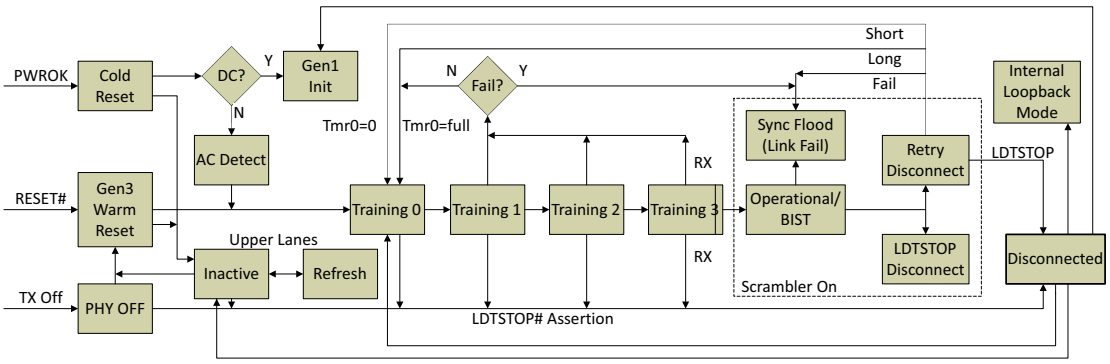


Figure 3-8: HT3 Startup Sequence [14]

3.8.3 I/O Stream

In HT packets travel in I/O streams. An I/O streams contains data which can be treated independently from other streams. The streams are separated into two flow directions, downstream from and upstream to the host bridge. Upstream streams can be distinguished by the UnitID as requests and responses always contain the UnitID of the node which generated the packet. Downstream traffic always travels in the same stream as requests contain the UnitID of the host bridge and responses contain the UnitID of the receiving node. Thus, no distinction of streams can be made.

3.8.4 Virtual Channels

HT supports different virtual channel (VC) sets for non-info traffic. Every VC has its own buffer set to prevent the different VCs from starvation. Each VC has its dedicated flow control.

There are five different VC sets defined by HT, they can be distinguished by the Isoc bit and the SeqID field (for requests) and the RspVCSet fields (for responses). The different types are Base, Isoc, AltSet, Non-FC, and Stream VC. As the defined VCs do not utilize the all encoding of the SeqID and the RspVCSet fields some of them are user specific while others are reserved for future standardization.

The Base VC set consists of three sub-virtual channels: posted-, non-posted-, and the response-VC. In the posted virtual channel requests travel which do not get a response, the

non-posted channel is for requests which need a response, and the response channel is for responses. Between the different sub-VCs an ordering scheme exist which guarantees the access to the actual data of an address region and prevents from deadlocks. The ordering scheme is depicted in table 3-4. A no indicates that subsequent packets are not allowed to overtake the current one, if the entry is a yes the subsequent packet must be able to overtake and if the entry is a yes/no overtaking is allowed but not necessary.

row pass column	posted request		non-posted request	response	
	ppw = 0	ppw = 1		ppw = 0	ppw = 1
posted request, ppw = 0	no	no	yes	yes	yes
posted request, ppw = 1	yes/no	yes/no	yes	yes	yes
nonposted request, ppw = 0	no	no	yes/no	yes/no	yes/no
nonposted request, ppw = 1	yes/no	yes/no	yes/no	yes/no	yes/no
response, ppw = 0	no	no	yes	no	no
response, ppw = 1	yes/no	yes/no	yes	yes/no	no

Table 3-4: HyperTransport Ordering

Besides the Base VC set all other VCs are optional. If optional VCs are used both sides of the link must support the corresponding VC. The only commonly used optional VC is the isochronous VC. It provides an additional set of VCs similar to the Base VC, but the isochronous channels are always prioritized. This is used to support timing critical traffic.

3.8.5 Packet Format

For data exchange between sender and receiver HT uses packets to decouple the outgoing request from the incoming response. HT has a word width of 32 bits and therefore the packet format is structured in this granularity. One packet can be distinguished into three frames: control frame, data frame, and protection frame. There are four different control frames types: request frame, response frame, info frame, and extension frame. A control frame can consist of different fields; those fields are shown and briefly explained in table 3-5.

Code	VChan	Command	Comments/Options	Packet Type
000000	-	NOP	Null packet. Contains flow control information	Info
000001		Reserved-HOST		
000010	NPC	Flush	Flush posted writes	Request
000011 0001xx		Reserved-HOST		
001xxx 101xxx	NPC PC	Wr (sized)	Write Request [5] Defines whether request is posted: 0: Nonposted 1: Posted [2] Defines the data length: 0: Byte 1: Doubleword [1] Defines bandwidth/latency requirements 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (reserved and set if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Addr/Data
01xxxx	NPC	Rd (sized)	Read Request [3] RespPassPW Defines ordering requirements for response: 0: Response may not pass posted requests 1: Response may pass posted requests [2] Defines the data length: 0: Byte 1: Doubleword [1]: Defines bandwidth/latency requirements: 0: Normal 1: Isochronous [0] Indicates whether access requires host cache coherence (reserved and set if access is not to host memory): 0: Noncoherent 1: Coherent	Req/Address
100xxx		Reserved-I/O		
110000	R	RdResponse	Read Response	Resp/Data
110001 110010		Reserved-HOST		
110011	R	TgtDone	Tell source of request that target is done.	Response
11010x		Reserved-HOST		
110110		Reserved-I/O		
110111	-	Extended FC	Contains Flow Control information for VCSets 0-7	Info
11100x		Reserved-HOST		Req/Address
111010	PC	Broadcast	Broadcast Message	
111011		Reserved-HOST		Request
111100	PC	Fence	Fence for posted requests	
111101	NPC	Atomic-RMW	Atomic Read-Modify-Write	Req/Addr/Data
111110	-	AddrExt/SourceID	Address Extension or Source Identifier Extension	Address
111111	-	Sync/Error	Link Synchronization and Error Packet	Info

Table 3-5: HT Control Fields [14]

The command field will always start at bit 0 of a doubleword and defines how the rest of the packet is structured. Different commands are more or less composed of fields that have different meanings and different locations inside the control frame.

3.8.5.1 Request Packets

Request packets are used to read or write data and to set the system into special conditions, and they consist of either 4 or 8 bytes. The supported request types are Write, Read, Broadcast, Flush, Fence, and Atomic Read-Modify-Write (RMW). Reads and Writes are used to exchange data. Writes can be posted or non-posted. For posted requests no responses are generated from the receiver, whereas for non-posted requests a response from the receiver is mandatory. Broadcasts are only issued by the host and travel downstream. All devices in a chain must accept and forward Broadcast packets. Flushes ensure that all previous transmitted posted packets in one I/O stream have been received. Fence packets are used to push all packets out of the posted channel independent of the I/O stream. The Atomic RMW is an optional request packet. Two versions of the Atomic RMW operation exist. One is Fetch and Add and the other one is Compare and Swap. All operations defined for Atomic RMW must be executed atomically. During Fetch and Add, data is read from the memory and written back after the transmitted value is added to it. A Compare and Swap request contains two values. The first value is compared to the data read from the memory and if they are equal, the second value is written back. The fields of the first two bytes of all request packets have always the same meaning. They contain the Cmd (command type), SeqID (ID of a data stream), UnitID (identification number of the device), and PassPW (used for relaxed ordering) fields shown in figure 3-9. The rest of the packet contains different fields depending on the type of the packet defined by the command field.

	bit / byte	7	6	5	4	3	2	1	0
dw0	0	SeqID[3:2]		Cmd[5:0]					
	1	PassPW	SeqID[1:0]		UnitID				
	2	*Command-Specific*							
	3	*Command-Specific*							
dw1	4	Addr[15:8]							
	5	Addr[23:16]							
	6	Addr[31:24]							
	7	Addr[39:32]							

Figure 3-9: HT Request Packet Format

3.8.5.2 Response Packets

There are two types of response packets, read responses and target dones. All response packets consist of 4 bytes. They are used to transfer data from the receiver to the sender of a read request or to acknowledge that a write operation has been completed. The fields of all response packets have the same meaning. How response packets are structured is shown in figure 3-10.

	bit / byte	7	6	5	4	3	2	1	0
dw0	0	*Command-Specific*		Cmd[5:0]					
	1	PassPW	Bridge	Rsv	UnitID				
	2	*Command-Specific*		Error0	*Command-Specific*				
	3	Rsv/RqUID		Error1	RespVCSet[2:0]			*Command-Specific*	

Figure 3-10: HT Response Packet Format

3.8.5.3 Info Packets

Three different types of info packets exist: Sync/Error, NOP, and Extended Flow Control packets for virtual channel sets (VCSets). Info packets can consist of 4 or 8 bytes. The Sync/Error packet is used during low-level link initialization or after an error occurred in Gen1 and consists of CAD and CTL all asserted for 32 bits. NOPs and the Extended Flow Control packets for other VCSets contain the released credit information of the different virtual channels. If retransmission is enabled, the acknowledgment counter of the correctly received packets is also included inside these packets. The NOP also contains the Discon bit, which signals that a reinitialization of the link is needed. There are two different versions of the Extended Flow Control packet, a short one with 4 bytes and a long one with 8 bytes. The 8 bytes version contains flow control information for different streams.

3.8.5.4 Extension Packet

There are two types of packet extensions. One is the Extended Address and the other one is the Source Identifier Packet Extension. They always consist of 4 bytes. Extended Address is used to extend the address range of an unextended packet from 40 bits addresses to 64 bits. If an Address Extension is used, the extension is transmitted first followed by a normal packet. Source Identifier Packet Extensions are used for high node counts. High node count enables HT to connect more than 32 devices, which would not be possible with the normally available Unit IDs. As long as the whole system can be mapped into an

address range of 40 bits and no large topology is used, extensions are unnecessary. Compared to other packets, extension packets do not consume an additional credit but are part of the packet they extend.

3.8.6 Dependencies

Besides the command field, the different packet types do not share many fields that have to be treated similarly. Also in packets of the same type, fields can have different meanings. Commands like reads and writes can be distinguished into several sub-commands. It is then possible that, the fields of the different sub-commands can have different meanings. This results in dependencies that increase the complexity of the hardware.

Even if fields are reserved, which means that they do not hold any information, this does not necessarily mean that it will not negatively influence the logic complexity. If there are for example two processing units A and B, one bit inside the control frame is needed to distinguish if A or B is needed to process the packet. However, for packets that will not need these two processing units, this bit is reserved. Then, there are not two but three possibilities how this field has to be handled: forward to A, forward to B, or ignore. This will not add wiring complexity, but logic has to be added for the ignore path.

In order to show how the dependencies sum, up they are separated into three categories, control frame dependencies, doubleword dependencies, and word width dependencies.

3.8.6.1 Control Frame Dependencies

Request packets cannot only be separated into read-, write-, broadcast-, fence-, flush-, and atomic-accesses. Reads and writes can also differ because of further control bits inside the header. All of those different types of control frames have dependencies between their fields that have to be solved before the packet can be processed. For example, it is not sufficient to determine that a packet is a read to know how the packet is structured. It is further necessary to known in which virtual channel it travels to know how the Sequence ID has to be interpreted. If it is a byte or doubleword read determines if it has a count field or a mask field. What kind of dependencies have to be handled is shown in figure 3-11.

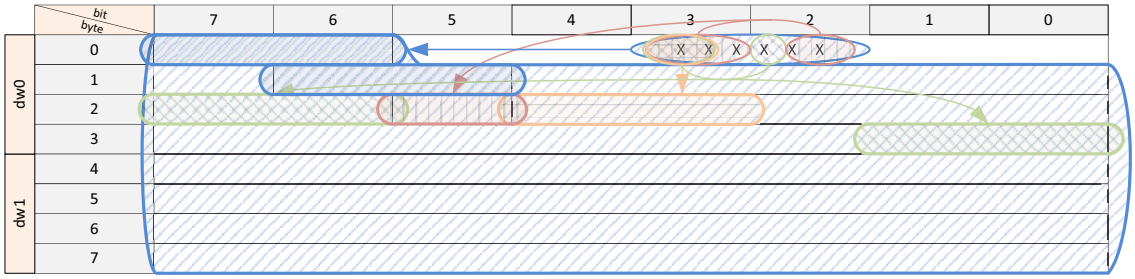


Figure 3-11: Request Header Dependencies

For response packets there exist fewer dependencies, because there are only two possibilities: read response and target done. The field structure of those two types is completely identical but nevertheless there are dependencies between the fields. Those dependencies occur because some fields can be either reserved or used and are shown in figure 3-12.

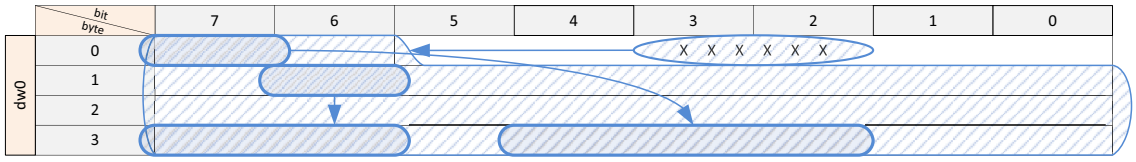


Figure 3-12: Response Header Dependencies

Info packets have completely different fields. For Sync/Error packets, the whole packet consists of all bits set to 1. NOPs and Extended Flow Control packets only have two fields in common, which are the command field and the RxNextPktToAck field. Only for small Extended Flow control fields, the RxNextPktToAck has the same location as in NOPs. In order to distinguish between the two types of Extended Flow Control packets bit 6 of bit time 0 is used. This saves command encoding but 7 bits instead of the 6 bit have to be decoded. In figure 3-13, the dependencies among info packets are shown.

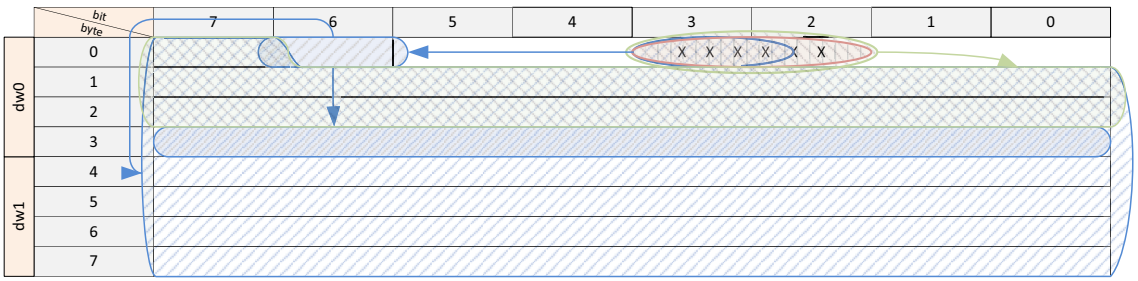


Figure 3-13: Info Header Dependencies

Packet extensions share no fields besides the command field. To be able to distinguish between the two different types Address Extension and Source Identifier Extension bits [7:6] of bit time 0 are used and therefore extend the command field. In figure 3-14 the dependencies of the extension packets are shown.

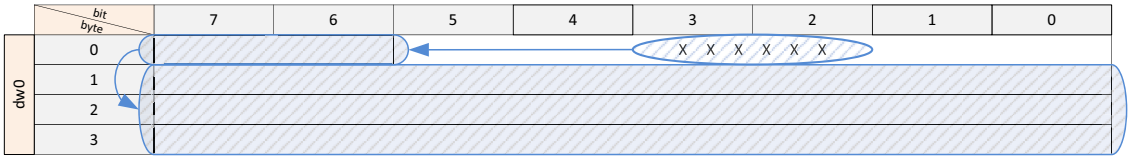


Figure 3-14: Extension Header Dependencies

The above mentioned dependencies result in an additional hardware effort for differentiating the meaning of the fields inside a control frame. However, not only the decision has to be made what kind of field has to be processed, but the information must also be forwarded to the corresponding unit. In order to get a comprehensive impression about is the complexity it is possible to layer the different packets above each other. If the fields are layered onto one another it results in a stack of seven mostly filled layers shown in figure 3-15. This means nearly every bit inside the first doubleword of the control frame needs a 1to7-mux and every bit of the second doubleword of the control frame will need a 1to2-mux.

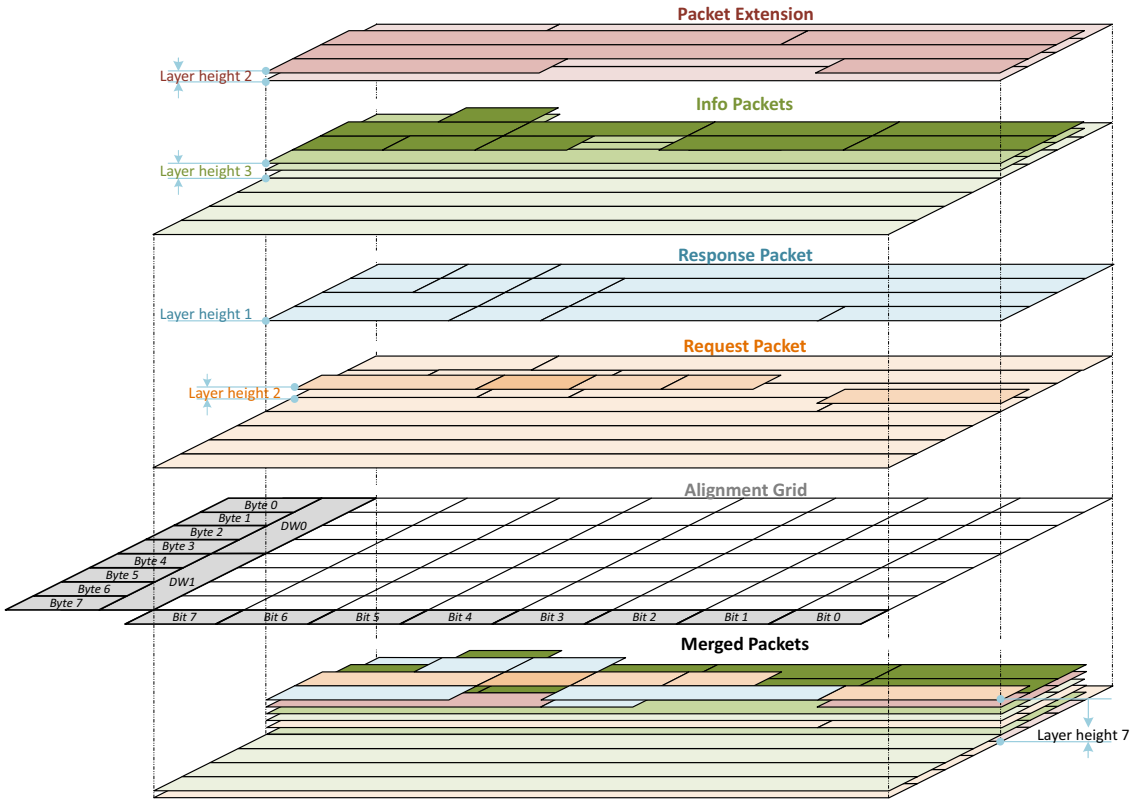


Figure 3-15: Dependency Layers

3.8.6.2 Doubleword Dependencies

Depending on how wide the data path is and how many restrictions are made where a control frame can start not only the dependencies inside of a control frame are important, but also the dependencies between the words of the protocol. In HT, every word of the control frame can occur at every word position of the data width. Therefore, the control frames have to be sliced into the doublewords they consist of and those slices have to be stacked above each other. In addition to the words of the control frames, all other frames have to be taken into account at this point regardless of the type (control, data, and protection). Figure 3-16 shows that the doubleword dependencies consist of 13 layers. The six additional layers compared to the packet dependencies result from the two packets that are longer than one doubleword, the byte-write-mask, data, CRC, and the reserved fields. Those layers cause an additional amount of hardware effort. More fields have to be identified at the same position of the data width and the majority of the bits need a 1to13 mux to be connected to the corresponding processing unit.

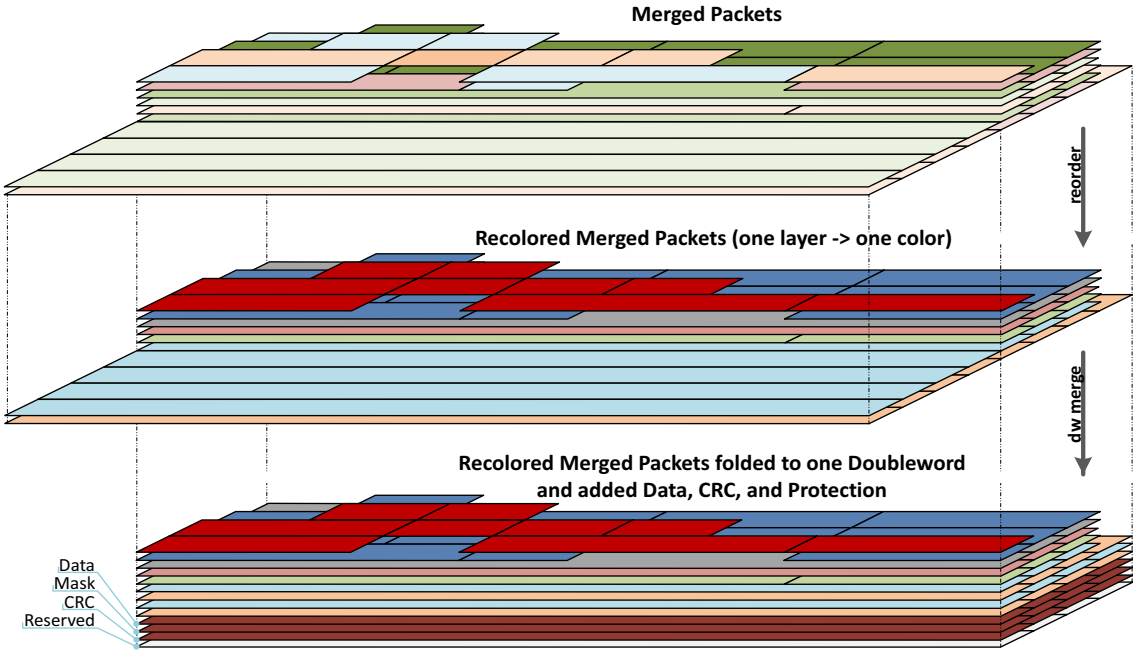


Figure 3-16: Doubleword Layers

3.8.6.3 Word Width Dependencies

If the data width chosen to realize a protocol is larger than the word width and no restrictions are made where a control frame is allowed to start than there are additional dependencies between the different word positions. This means for HT if the data width is for example 64 bits two doublewords will be received in one clock cycles. As there is no restriction where a packet starts or ends, the effort to solve the doubleword dependencies doubles. In addition, new dependencies between the doublewords will occur. In Gen1, for example it must be checked if a doubleword is part of control or a data frame. This can be determined by the CTL signal. However, it cannot be checked if the control frame and the data frame belong together or if the control frame is an inserted packet. When both doublewords are marked as control frames, it is not clear if those doublewords belong to the same control frame or not. First, it must be determined if the first doubleword is the end of a control frame or not. Therefore, it must be checked if the start of a control frame was received in the previous cycle that needs more doublewords to complete in this clock cycle, or if it is the start and end of a control frame at the same time. If this is not the case, the two doublewords belong to the same control frame. Due to the additional encoding of CTL and the restrictions of inserted packets, it is easier to determine what doubleword be-

longs to which control frame. If the packets are in the first step only processed by the CTL coding, control frames without data can be only handled if the corresponding CRC has been received or three doublewords of the same control frame have been received. In Gen1 as well as in Gen3 it must first be determined what kind of doublewords have been received before they can be stored, analyzed, and/or forwarded. If there are multiple control frame doublewords received in one clock cycle all the dependencies add up in processing time. First, the dependencies between the doublewords received in one clock cycle, then the dependencies between the different layers inside one doubleword, and then the dependencies between the fields of a packet have to be resolved. Only after this is completely done the processing of a packet can start. This shows that it is very important to keep these dependencies low so that the hardware effort is manageable.

As the only restriction in HT is the doubleword granularity, every doubleword slot inside the data width must be able to process every possible doubleword. If the data width is large enough to receive more than one control frame of the same type at the same clock cycle, not only the effort to analyze one doubleword after the other, and the effort to analyze the doubleword to create the correct control frames has to be added, but also the ability to process the information in the next stage must be doubled. Otherwise, not all data can be processed in the next clock cycle and bandwidth would be lost. In this case, Gen1 is more complicated than Gen3, as a packet is always followed by a CRC in Gen3 so it can be guaranteed that a packet is at least 64 bits long and no multiple packets can be received for a 64 bits data width. Of course, if it is necessary to increase the data width further this will result in even more hardware effort.

3.8.7 HT Bandwidth

It is necessary to accumulate all bandwidth which can be physically used to compare bandwidth between protocols. Therefore all lanes which could transmit data have to be summed up, regardless of their purpose in the protocol. For maximum bandwidth the largest link with the fastest transmission rate has to be chosen. In case of HT a Gen3 link has to be chosen. A link consists of 32 CAD lanes, 4 CTL lanes, and 4 clock lanes. The maximum transmission rate is at a speed of 3.2 GHz with DDR. As at DDR the data is sampled

at the positive and negative clock edge 6.4 giga-transfers per second are provided. This results in a physical bandwidth of 32 GB/s.

In order to get the user data bandwidth, the four clock lanes have to be removed from the calculation as they transmit only clocks and no information. Afterwards, only 28.8 GB/s could be used for user data transmission, which is a reduction of 10% (3.2 GB/s). As the four CTL lanes are only used to transmit control information, the achievable user data bandwidth is reduced by additional 10%, which results in a bandwidth of 25.6 GB/s. This means that the bandwidth which could be used on the CAD lanes for data transmission is 80% of the physically available bandwidth.

The user data bandwidth is further reduced as not all transmitted data on the CAD lanes is user data. For the calculation it is assumed, that the link can be fully utilized and no management traffic has to be transmitted. Further, the minimum overhead for a packet is chosen with the maximum user data payload. The overhead for the user data is the header, the protection CRC, and the periodic CRC slot, which is not used for protection in Gen3 but still has to be transmitted to handle clocking differences.

First, the achievable bandwidth including the periodic CRC is calculated. Every 512 bytes a 4 bytes CRC is transmitted:

$$\text{Bandwidth with periodic CRC: } 100 \cdot \frac{512}{(512 + 4)} = 99.22\ \%$$

Second, the achievable bandwidth per packet is calculated. The minimal header size is 8 bytes, per packet CRC is 4 bytes, and the maximum payload is 64 bytes:

$$\text{Header Overhead: } 100 \cdot \frac{64}{(64 + 8 + 4)} = 84.21\ \%$$

So from the 80% of physical bandwidth only 99.22% can be used for packet transmission:

$$\text{Achievable Packet Bandwidth: } 80\ \% \cdot 99.22\ \% = 79.38\ \%$$

Those 79.38% bandwidth can be used to transmit packets with 84.21% of user data:

$$\text{Achievable Bandwidth: } 79.38\ \% \cdot 84.21\ \% = 66.85\ \%$$

This means, in the best case 66.85% of the physical available bandwidth can be used to transmit user data in HT.

3.8.8 Fault Tolerance

Depending on the HT version that is used there are two different protection modes. In Gen1 CRC windows are used. A 32 bits CRC is transmitted after 572 bytes of CAD and CTL. If the link width is up to 8 bit lanes, a single CRC is generated and transmitted. For 16 and 32 wide links one CRC is generated for every byte-lane. The first byte-lane also includes the CTL lane, for the other byte-lanes CTL is assumed as 0. CAD and CTL signals are stored until 8 bits CAD and 1 bit CTL have been received. For links smaller than a byte-lane the additional received CTL bits are ignored because CTL must be the same for one byte of CAD. If an error is detected the standard behavior for Gen1 is to use sync packet flooding. This causes the link to transmit 1 at all lanes until a warm reset is performed. In Gen1 the retransmission protocol is optional.

In Gen3 higher frequencies are used and therefore it is more likely that errors occur. Thus, the retransmission protocol became mandatory and every packet has to be protected. The bandwidth is reduced by the per packet CRC. Also the CRC slot for the CRC window has to be maintained to allow clock adjustment. For the retransmission protocol the receiver must acknowledge the last correctly received packet, not till then the sender can delete the packet from the retransmission buffer the polynomial of the CRC is shown below.

HT CRC Polynomial:

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$$

HT is mainly used for inter-processor communication. Thus, it was optimized for small message sizes and small communication distances. This resulted in an extremely compacted header format and a relative high communication overhead. Besides the usage in AMD processors HT is also used in projects like NumaConnect [15] and EXTOLL [19].

3.9 Peripheral Component Interface Express

Peripheral Component Interface (PCIe) is the state of the art protocol used to connect performance critical devices of a node. Nearly every current accelerator or network card used in HPC systems uses PCIe for inter node communication. This makes PCIe very interesting for a detailed analysis.

PCIe [16] is an enhancement from the former PCI [16] and PCI-X [16] bus, which were real bus based interconnects where only one participant of the interconnect was allowed to communicate at a time. The first version of PCI was developed in 1990 by Intel and it was issued 1992, for the later versions the PCI-SIG (Special Interest Group) was formed and PCI became an industry standard. Because of the increasing amount of communication inside a system among different participants it was necessary to use bandwidth more efficiently. Thus, a change was made with PCIe to a point to point packet based communication in 2004.

3.9.1 PCIe Topologies

PCIe topologies are built out of four different devices: root complex, switch, bridge, and endpoint. From a software perspective the only sure thing known is that the processor is connected to a bridge device which belongs to the PCIe root complex. The root complex contains the connection to the CPU and the main memory. It is always the head of the topology from where the whole system is initialized. Connections from the root port are called downstream connections and connections to it are called upstream connections. Switches are used to connect multiple PCIe devices, bridges are used to connect legacy PCI or PCI-X devices to PCIe, and endpoints have only one PCIe connection and do not connect further devices. An overview of the different devices is shown in figure 3-17.

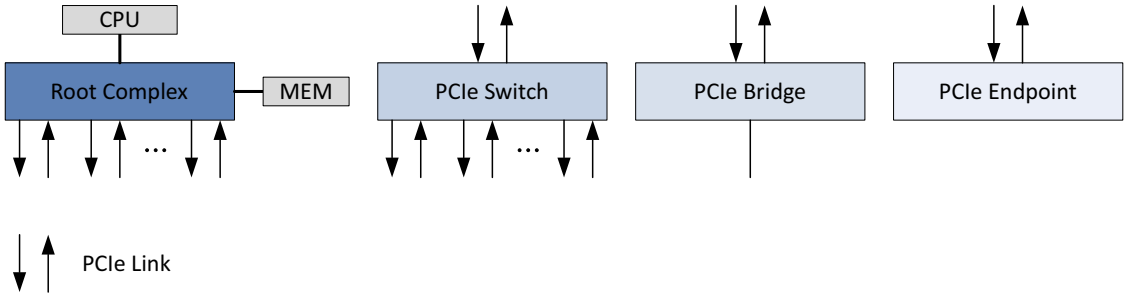


Figure 3-17: PCIe Device Overview

A tree like topology can be built with the PCIe devices. In order to enumerate PCIe devices a bus, device, and function scheme is used. Every connection between at least two PCI devices is called a bus. Up to 256 buses are allowed in one PCIe fabric. Every bus is capable of connecting 32 devices and each device can have up to 8 functions. An example topology is shown in figure 3-18.

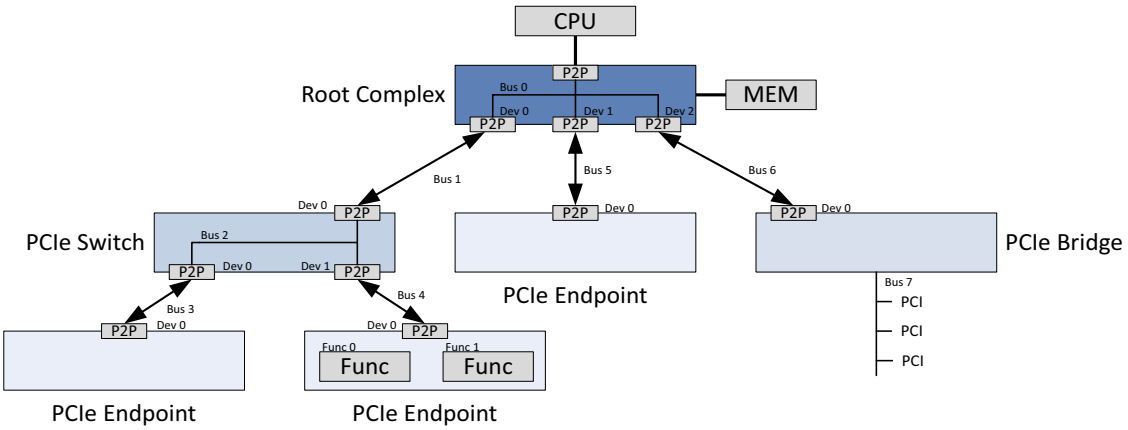


Figure 3-18: PCIe Topology Example

In this chapter the focus is mainly placed on the Gen3 version of the PCIe protocol as it is the version with the highest performance with respect to bandwidth and latency. Gen1 and Gen2 are partly described to get a picture of the whole protocol and it is necessary for the initialization of PCIe and the backwards compatibility.

3.9.2 Physical Layer Structure

Compared to HT PCIe uses no sideband signals. Thus, no sideband signals are available for control or management reasons. Therefore, all which could be transmitted separately must be somehow embedded inside the normal transmission lanes. For example, the clock

signal is embedded and must be recovered with a CDR mechanism. Also management signals as for interrupts, power management, and error handling are mapped to special message packets. A PCIe link consists of a symmetrical and bidirectional connection of at least one lane. The width of the link can be anything of 1, 2, 4, 8, 12, 16, or 32 lanes. The different connections are shown in figure 3-19.

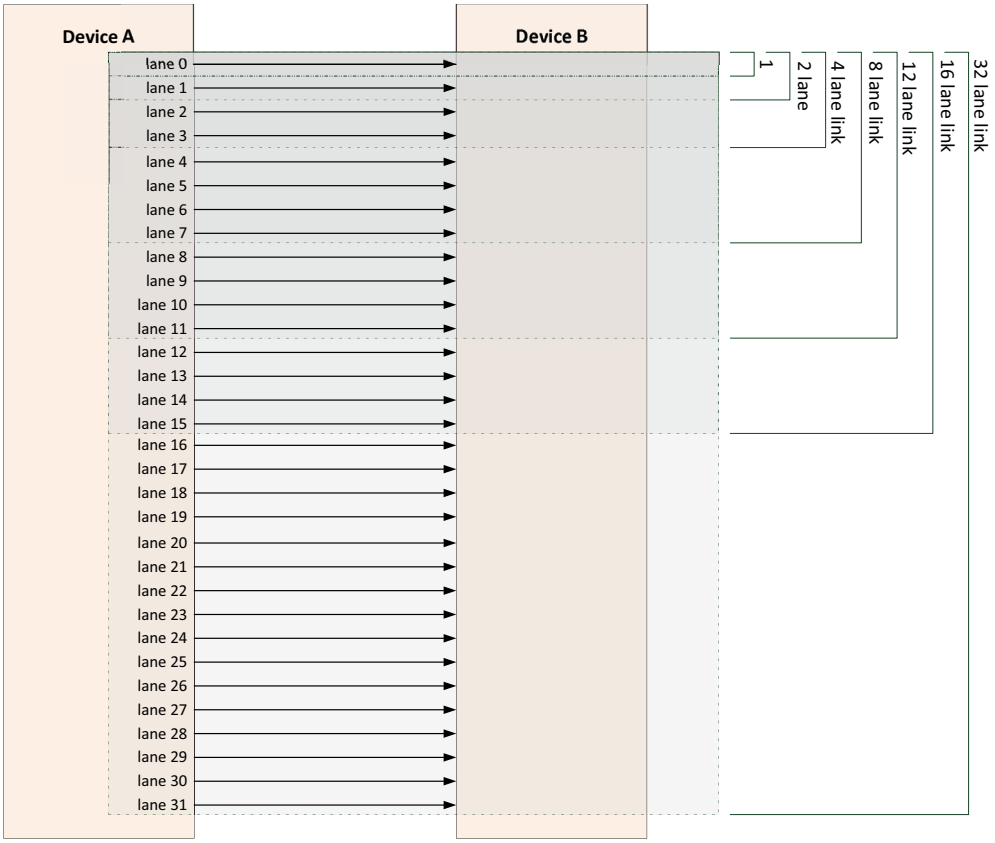


Figure 3-19: PCIe Link Configurations

PCIe has three different speed grades. A new speed grade was introduced with every new major update of the PCIe generation. The different speed grades are shown in table 3-6. As the Gen1 speed grade is used for initialization of every PCIe device, the 2.5 GT/s are mandatory. Thus, also the 8b/10b line coding for Gen1 and the corresponding framing must be implemented. In case of a Gen3 capable device it is assumed that the device is capable of the Gen2 link speeds. With every generation of PCIe the bandwidth has been doubled. From Gen1 to Gen2 simply the frequency was doubled. With Gen3 the decision was made to not double the bandwidth because the high frequencies were not easy to meet. Thus, the line coding was changed from 8b/10b to 128b/130b. With this decision

nearly 20% of bandwidth was saved and a change from 5GT/s to 8 GT/s was sufficient to double the bandwidth.

	Transmitter Clock Frequency (MHz DDR)	
Generation	MHz DDR	Mbit/lane
Gen1	1250	2500
Gen2	2500	5000
Gen3	4000	8000

Table 3-6: PCIe Link Frequencies

3.9.3 PCIe Logical Network Layers

PCIe uses different logical layers for initialization, management, and data transmission. Three different layers are used: physical layer, data link layer, and transmission layer. The structure of the different layers is shown in figure 3-20.

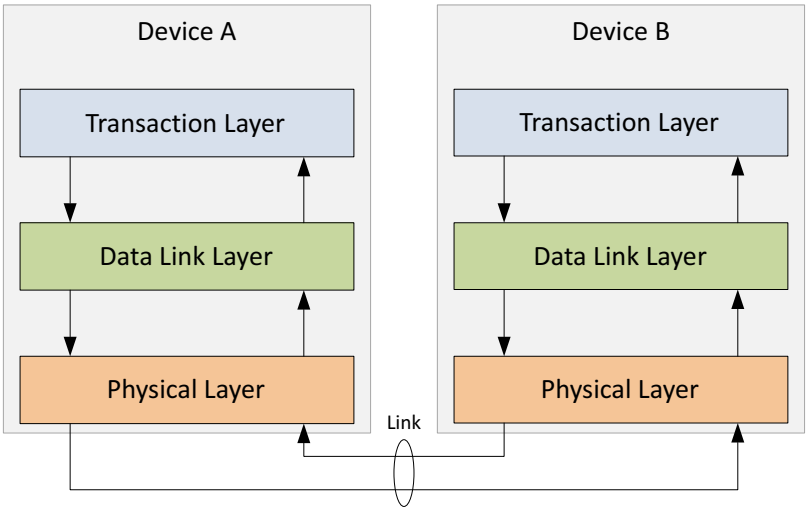


Figure 3-20: PCIe Network Layers

In the transaction layer the actual user data is transferred among devices. The user data gets a header which describes the type of transaction. An optional prefix to the header can be added as well as an optional CRC which gives an end to end protection. An overview of a transaction with the different frames is given in figure 3-21.

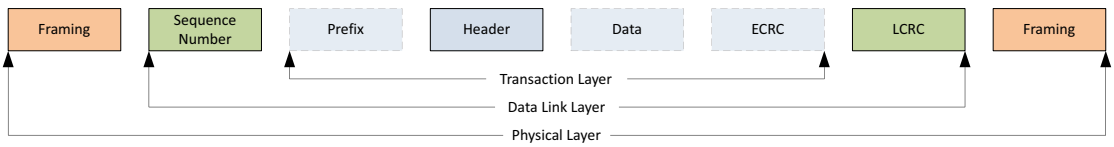


Figure 3-21: Framed Packet Transmission

Packets transmitted in the data link layer are called data link layer packets (DLLP). In this layer the retransmission of data is handled with an ACK/NACK scheme. Correct transmitted transaction layer packets are acknowledged whereas erroneous packets are negative acknowledged and a retransmission is therefore requested. DLLPs are not included in the retransmission scheme. Further, the flow control is managed in the data link layer. The number of available buffer space is exchanged for every virtual channel. Power management is also part of this layer, but as this aspect of a protocol is not in the scope of this thesis it is not further described.

The physical layer is used for link initialization and framing. During initialization, defined patterns, which belong to the so called ordered sets (OS), are transmitted to achieve clock lock, symbol lock for lanes, and lane deskew among different lanes. Other OS are used for compensation of slightly different frequencies of two devices, for power saving, or if the stream starts. Frames are used to add protection to the data stream and to transmit additional information.

3.9.3.1 Transaction Layer (TL)

In PCIe packets are transferred doubleword aligned. Every doubleword is separated into symbols. In one symbol-time one symbol is transmitted per lane. One symbol transports one byte of data without line coding. On a lane the symbol is serialized for transmission. If multiple lanes are used line 0 contains the first symbol of one symbol-time. The next symbol is transmitted in lane 1 and so on. An overview of the structure is given in figure 3-22.

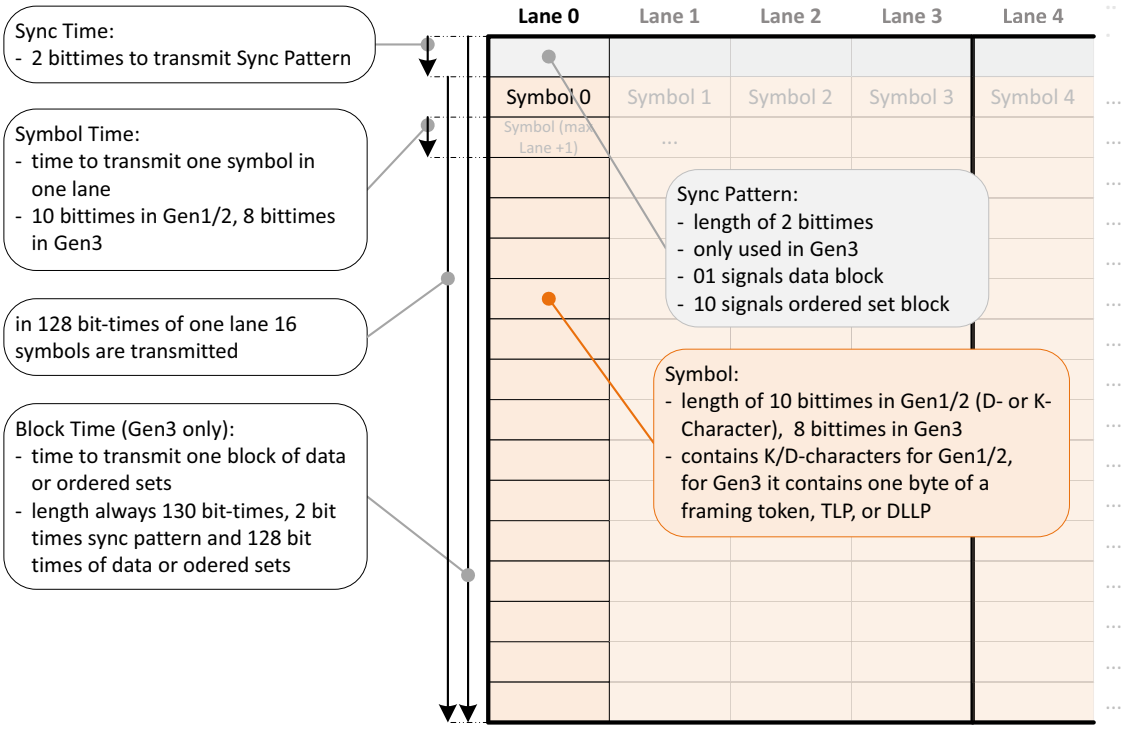


Figure 3-22: PCIe Traffic Transmission Procedure

Virtual Channels

Different virtual channels are available in PCIe. There are 8 different virtual channels available in PCIe, VC0-VC7. A higher number for the VC indicates a higher priority. Pretty similar to HT every VC has its own buffers and flow control. Used virtual channels must be supported by both sides of the link. If one side supports more VCs than the other the link is configured to the maximum number both sides support. Every of the 8 VCs has 3 sub virtual channels which are: non-posted, posted, and completion. Non-posted means that the actual packet is a request which needs a response, posted request do not get a response, and completions are responses to a request.

Several schemes are available to handle the prioritization of the different virtual channels. One is the strict priority arbitration. In this chase, if a channel with a higher priority wants to transmit traffic it is allowed to do so until no more traffic is available for this channel. This implies the danger of starvation of lower prioritized channel. However, a more fair arbitration scheme can be used which allows higher prioritized channels to transmit data more often than the lower channels. Two ways are possible to handle this, one is a fixed

arbitration scheme by hardware, and the other one is a weighted round robin arbitration. In order to handle the weighted round robin arbitration a table has to be written to state which virtual channel is allowed to transmit at a certain time slot. High priority channels simply get more time slots compared to other channels. In addition, the fixed and the arbitrary schemes can be mixed. Therefore, high priority VCs have a strict priority scheme and VCs with lower priority use an arbitrary scheme.

In the packet header the VC is represented by the traffic class (TC) field. As this field is 3 bits wide it supports exactly 8 alternatives from 0-7. However, the TC is not exactly the same as the virtual channel which the packet is traveling in, because not every link supports every traffic class. If the corresponding VC is not available the traffic class must be mapped to another virtual channel. The TC field is unchanged even though the VC is changed.

Among the 8 VCs no ordering exists. If the VC with the highest priority is blocked a VC with a lower priority should be arbitrated. In one channel the sub virtual channel follow the ordering rules in figure 3-23.

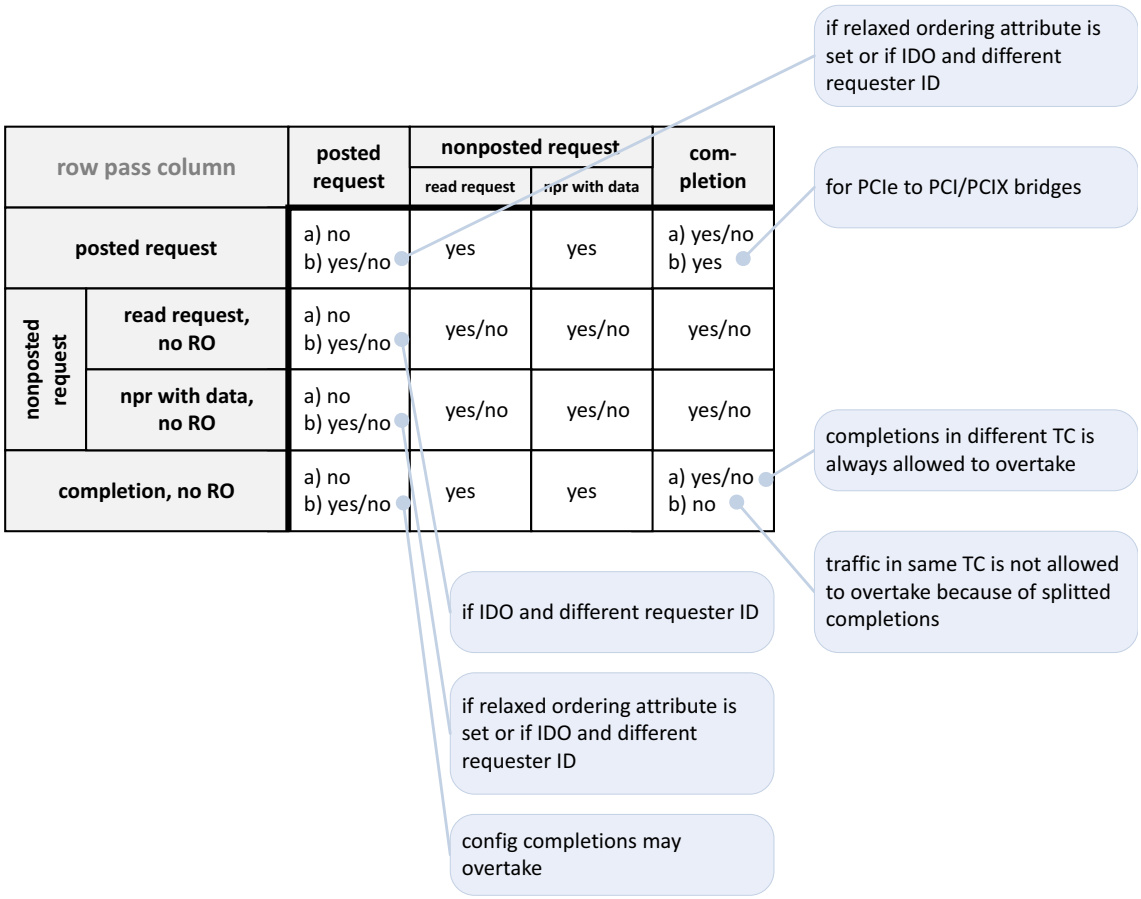


Figure 3-23: PCIe Ordering Rules

A no means that the packet with the corresponding VC type of the row is not allowed to overtake a packet with the VC type of the column. A yes means that the packet must be able to overtake the other one and a yes/no means that the packet is allowed to overtake but it doesn't have to be able to overtake for the correct function of the link. With these ordering rules the producer consumer model of PCIe is guaranteed. Therefore, data is always received in the right order and no deadlocks might occur.

Several fields of the ordering scheme shown in figure 3-24 contain an entry a) and b). In some of the cases the ordering is relaxed to increase the performance of the link. Otherwise the link could be stalled if a preceding packet is blocked but the following packet is independent. There are five exceptions but the two main ones are relaxed ordering and ID based ordering (IDO). If the hardware determines during generation of a packet that it is independent, then the relaxed ordering can be used and the packet might overtake other-

wise forbidden types. IDO is used if two packets are received from different IDs (Bus/Device/Function). As long as it is sure that the traffic from both devices is completely independent those packet can be reordered.

Header Format

As data is transferred doubleword aligned the start of a new packet can only occur on lane 0 and on lanes which are doubly even. How the packet is transmitted through the different layers in Gen3 is shown in figure 3-24, the pale parts are optional and must not be contained in an actual packet.

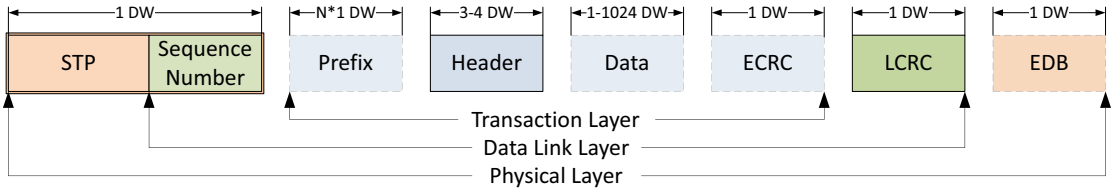


Figure 3-24: Gen 3 TLP Transfer

The header of a TLP is almost equal for all packet types. They consist of three or four doublewords corresponding to the type. Several different header types exist in PCIe. Those are IO requests, memory requests, configuration requests, completions, and messages. The first doubleword is the same for all requests and completions, only for messages the length field is reserved in most cases. It contains all needed information how the header is structured. What kind of packet is transferred is defined in the first byte which contains the Format (Fmt) and the Type field, the coding of both is shown in table 3-7. For a detailed description of all fields please refer to [16].

FMT	Type	VChan	Command	Comments/Options
000/001	00000	NP	MRd	Memory Read Request
000/001	00001	NP	MRdLk	Memory Read Request – Locked Access
010/011	00000	P	MWr	Memory Write Request
000	00010	NP	IORd	IO Read
010	00010	NP	IOWr	IO Write
000	00100	NP	CfgRd0	Configuration Read (Type 0)
000	00101	NP	CfgRd1	Configuration Read (Type 1)
010	00100	NP	CfgWr0	Configuration Write (Type 0)
010	00101	NP	CfgWr1	Configuration Write (Type 1)
001	10xxx	P	Msg	Message Request without Data
011	10xxx	P	MsgD	Message Request with Data
000	01010	C	Cpl	Completion without Data
010	01010	C	CplD	Completion with Data
000	01011	C	CplLk	Completion without Data – associated with Locked Memory Read
010	01011	C	CplDLk	Completion with Data – associated with Locked Memory Read
010/011	01100	NP	-	Fetch and Add AtomicOp Request
010/011	01101	NP	-	Unconditional Swap AtomicOp Request
010/011	01110	NP	-	Compare and Swap AtomicOp Request
100	0LLL	-	-	Local TLP Prefix
100	1EEE	-	-	End-to-End TLP Prefix

Table 3-7: TLP Format and Type Field Coding

In the following the different header types of TLPs are described. In the figures the highlighted fields show how the packet is structured. In most cases it is defined only by the first byte. Only by memory accesses there is also a difference if the steering tag is used. In some headers different fields have restrictions as they must have a value of 0. In this case the 0 value is added to the field with the notation “field name”=0.

The specification encourages not using IO requests because, they are only for legacy devices which rely on IO map and not on memory map. However, they are still available for backward compatibility reasons. Both, a read and a write request are available. The header size is always three doublewords. Performance of IO requests is weak because the amount of data of one request is restricted to one doubleword. The structure of the header is shown in figure 3-25.

TLP IO Request									
	bit / byte	7	6	5	4	3	2	1	0
dw0	0	Fmt[2:0]			Type[4:0]				
	1	*Reserved*	TC[2:0]==000			*Reserved*			
	2	TD	EP	Attr[1:0]==0		AT[1:0]==00		Length[9:8]	
	3	Length[7:0]							
dw1	4	Requester ID[15:8]							
	5	Requester ID[7:0]							
	6	Tag[7:0]							
	7	Last DW BE[3:0]==0000				First DW BE[3:0]			
dw2	8	Addr[31:24]							
	9	Addr[23:16]							
	10	Addr[15:8]							
	11	Addr[7:0]						*Reserved*	

Figure 3-25: IO Request Header Format

Memory requests are used to transport the user data. The header size can be either three or four doublewords. Three different accesses can be performed with those requests: memory reads, memory writes, and memory locked reads. The address which is accessed defines the header size. If an address space has to be accessed beyond 32 bits the larger header has to be used, otherwise the smaller header must be used. Regarding to the address field, one thing to mention is that the byte which contains the MSB of the address is transferred in byte eight of the header. This leads to the point that sometimes the bits [31:2] of the address are transferred in doubleword three and sometimes in doubleword four. Whether it is a read or a write access, the TLP processing hints (TH) field determines if byte six or seven of the header contain the tag or byte enable, or the steering tag. This is an optional and very specialized feature of the specification and is out of the scope of this thesis. As memory locked reads are only available in the spec for legacy reasons and modern devices are prohibited to use them they are not described in detail. The structure of the memory request header is shown in figure 3-26.

TLP Memory Request 64									
	byte \ bit	7	6	5	4	3	2	1	0
dw0	0	Fmt[2:0]				Type[4:0]			
	1	*Reserved*	TC[2:0]			*Reserved*	Attr[2]	*Reserved*	TH
	2	TD	EP	Attr[1:0]		AT[1:0]		Length[9:8]	
	3	Length[7:0]							
dw1	4	Requester ID[15:8]							
	5	Requester ID[7:0]							
	6	Tag[7:0] / ST[7:0]							
	7	Last DW BE[3:0] / ST[7:4]				First DW BE[3:0] / ST[3:0]			
dw2	8	Addr[63:56]							
	9	Addr[55:48]							
	10	Addr[47:40]							
	11	Addr[39:32]							
dw3	12	Addr[31:24]							
	13	Addr[23:16]							
	14	Addr[15:8]							
	15	Addr[7:0]						PH[1:0]	

Figure 3-26: Memory Request Header Format

Configuration requests are used to configure a PCIe device and are sent from the root complex. The header is always three doublewords long and the packet contains one doubleword of data. Inside the type field the least significant bit (LSB) determines if the packet has reached the correct bus. As long as the packet is traveling downstream and the secondary bus of the device is not the target bus the field is 1, otherwise the field is set to 0. Compared to the former request it is not routed by the address but by the ID of the receiving device. As the ID is only two bytes, the other two bytes of doubleword three indicate which configuration register is addressed. The structure of the header is shown in figure 3-27.

TLP Configuration Request									
	byte \ bit	7	6	5	4	3	2	1	0
dw0	0	Fmt[2:0]				Type[4:0]			
	1	*Reserved*	TC[2:0]==000			*Reserved==0000*			
	2	TD	EP	Attr[1:0]==00		*Reserved==00*		Length[9:8]	
	3	Length[7:0]							
dw1	4	Requester ID[15:8]							
	5	Requester ID[7:0]							
	6	Tag[7:0]							
	7	Last DW BE[3:0]==0000				First DW BE[3:0]			
dw2	8	Bus Number[7:0]							
	9	Device[5:0] / Function Number ARI[7:3]					Function[2:0] / Function Number ARI[2:0]		
	10	*Reserved*				Ext Reg Number[3:0]			
	11	Ext Reg Number[5:0]						*Reserved*	

Figure 3-27: Configuration Request Header Format

Completions are the responses to requests. They are always three doublewords long and routed by the ID of the requester. The tag must be the same as in the request so the requester can merge the completion to the corresponding request. Completions for a single request can be partitioned into multiple packets and therefore the requester must be able to handle multiple completions. Multiple completions can be avoided if the requested data is within one read completion boundary which are junks of 128 bytes in most cases. Completion status gives information if everything worked on the completer side or if an error occurred. In case an error occurred, the type of error is encoded. The byte count modified (BCM) field is only used by PCI-X completers and therefore out of the scope of this thesis. Together with the byte count and the lower address the start- and the end-address can be determined. The structure of the header is shown in figure 3-28.

TLP Completion									
	bit / byte	7	6	5	4	3	2	1	0
dw0	0	Fmt[2:0]			Type[4:0]				
	1	*Reserved*	TC[2:0]			*Reserved*	Attr[2]	*Reserved*	
	2	TD	EP	Attr[1:0]		AT[1:0]		Length[9:8]	
	3	Length[7:0]							
dw1	4	Completer ID[15:8]							
	5	Completer ID[7:0]							
	6	Completion Status[2:0]			BCM	Byte Count[11:8]			
	7	Byte Count[7:0]							
dw2	8	Requester ID[15:8]							
	9	Requester ID[7:0]							
	10	Tag[7:0]							
	11	*Reserved*	Lower Address[6:0]						

Figure 3-28: Completion Header Format

The last type of packets traveling in the transaction layer are messages. Messages are used to get rid of additional sideband signals. A message is always four doublewords long. In most cases the message only contains information in the first two doublewords. The lower 3 bits of the type field define the structure of bytes 8-15 and the routing of the packet, which can be address based, ID based, or implicit. Byte 7 contains the actual information of the message which is called message code. The different meanings of the code are shown in table 3-8.

Message Code	Explanation
0000 0000	Unlock Message
0001 0000	Lat. Tolerance Reporting
0001 0010	Optimized Buffer Flush/Fill
0001 XXXX	Power Management 0100 - Active State Nak 1000 - PME 1001 - Turn Off 1011 - PME to Ack
0010 0XXX	INTx Message 000 - Assert INTA 001 - Assert INTB 010 - Assert INTC 011 - Assert INTD 100 - Deassert INTA 101 - Deassert INTB 110 - Deassert INTC 111 - Deassert INTD used if Message Signaled Interrupt (MSI) is not supported
0011 00XX	Error Message 00 - Error Correctable 01 - Error Uncorrectable, Non-Fatal 11 - Error Uncorrectable, Fatal
0100 XXXX	Ignored Messages Formerly used for hot Plug. With Spec. 1.1 not longer supported.
0101 0000	Set Slot Power Message
0111 111X	Vendor-Defined Messages 0 - Message 0 1 - Message 1

Table 3-8: Message Codes

If no Message Signaled Interrupt (MSI) is supported the INTx interrupt message travels upstream to inform higher level devices. Power management messages are used to support the PCI power management and add hardware based link management. Error messages travel upstream and are implicitly routed to the root complex for error detection. In order to support the locked transaction protocol for PCI the unlock message is provided. It ends the lock so other devices are able to use VC0. With the set slot power limit message a downstream port signals an endpoint plugged into a slot the power limit it has. There are also vendor specific, latency tolerance reporting, and optimized buffer flush/fill messages. As they are optional they are not described in detail. An overview for the described packets is shown in figure 3-29.

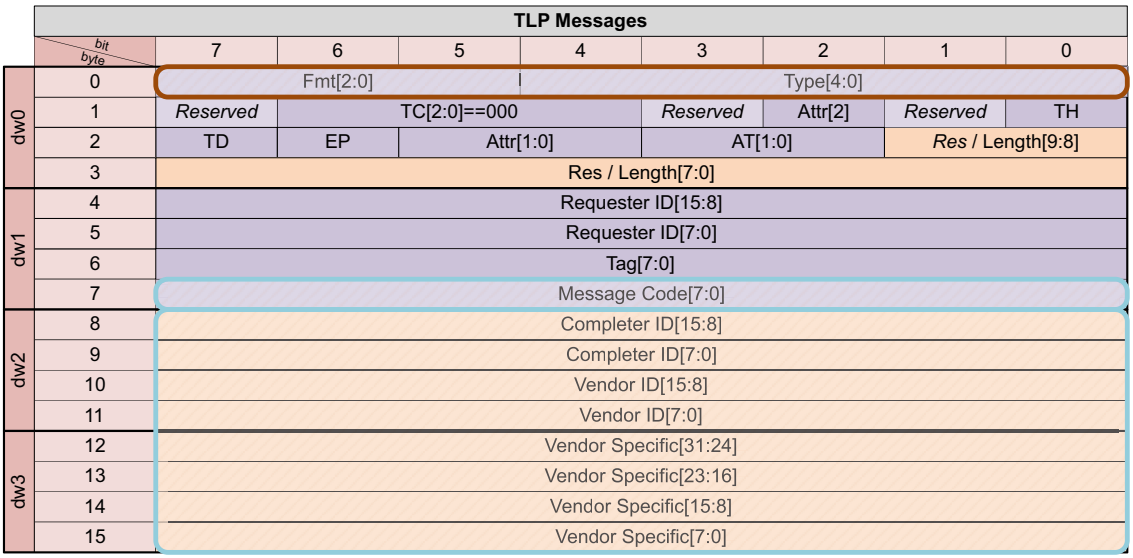


Figure 3-29: Message Header Format

Transaction Layer Extensions

One extension is the prefix. A single extension consists of one doubleword, but multiple prefixes are allowed for a single TLP. Two different types of prefixes exist: local and end to end. The local prefix is only used between the two sides of one link and no restrictions are made to the number of local prefixes whereas the end to end version travels from the source to the destination and the number of prefixes is limited to four. Local prefixes always precede the end to end prefixes. For every type of prefix one is completely defined but for both two vendor specific prefixes are available. For end to end the steering tag can be transported inside the prefix if it is not inside the header. Multi-root-I/O virtualization is the available prefix. As this feature is optional it is not described in more detail. Of course all participants of the transmission must be capable to handle the corresponding prefix.

The second extension, which is sent at the end of the TLP, is the end-to-end CRC (ECRC). It is also one doubleword long and protects the TLP from the source of the packet up to the destination. For more details of the ECRC please refer to chapter 3.9.7.

3.9.3.2 Data Link Layer (DLL)

The DLL manages the link between two devices. Therefore packets are exchanged on the link which are called data link layer packets (DLLP). This traffic is always sent between two devices and never forwarded to another device and is therefore called local. The tasks of the DLL are flow control, power management, link initialization, and error handling.

The core of the DLLP is always 4 bytes. It contains the type field and all the information of the packet. In Gen3 a DLLP always starts with a two bytes start of DLLP (SDP) token and ends with a 16 bits CRC. The different types of DLLPs are shown in table 3-9. All headers shown later will include the SDP token to keep the doubleword alignment, even though the token belongs to the physical layer.

Encoding	Type
0000 0000	Ack
0001 0000	Nak
0010 0000	PM_Enter_L1
0010 0001	PM_Enter_L23
0010 0011	PM_Active_State_Request_L1
0010 0100	PM_Request_Ack
0011 0000	Vendor Specific – Not used in normal operation
0100 $0v_2v_1v_0$	InitFC1-P (v[2:0] specifies Virtual Channel)
0101 $0v_2v_1v_0$	InitFC1-NP
0110 $0v_2v_1v_0$	InitFC1-Cpl
1100 $0v_2v_1v_0$	InitFC2-P
1101 $0v_2v_1v_0$	InitFC2-NP
1110 $0v_2v_1v_0$	InitFC2-Cpl
1000 $0v_2v_1v_0$	UpdateFC-P
1001 $0v_2v_1v_0$	UpdateFC-NP
1010 $0v_2v_1v_0$	UpdateFC-Cpl

Table 3-9: DLL Type Encoding

Together with a positive acknowledgment (ACK) or an negative acknowledgment (NACK) the sequence number of the corresponding packet is transmitted. With the sequence number it can be determined which packets have been received correctly. If an ACK is received all packets, including the packet corresponding to the received sequence number, are acknowledged to be received correctly. On the other hand, if a NACK is received all traffic has to be retransmitted from the corresponding sequence number on. The structure of the header is shown in figure 3-30.

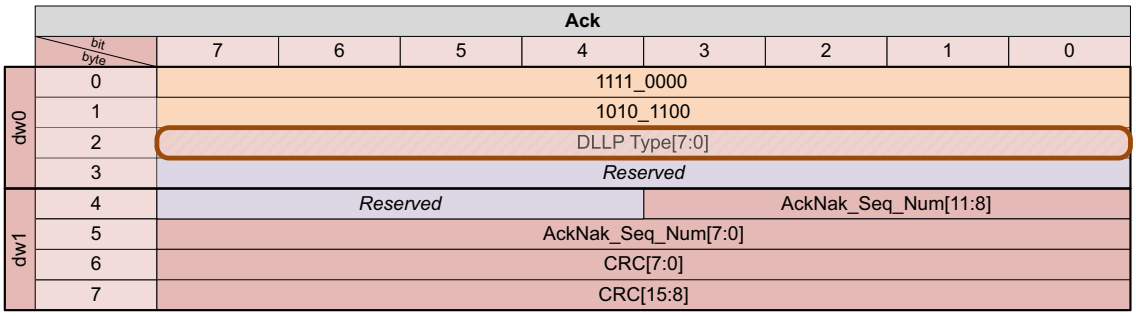


Figure 3-30: ACK/NACK Header Format

The DLL is also responsible for the flow control. Every sub VC must support a minimum number of credits. The size of one credit differs for the VCs. Posted and non-posted have a size of 5 doublewords per header were a completion has 4 doublewords. At least one header must be available per VC. Data credits are grouped in blocks of 4 doublewords. The maximum possible payload of a packet is important to calculate the correct minimum number for the posted and completion VC. There must be enough credits available for at least one maximum payload size data block. Thus, for a 512 bytes maximum access at least 32 credits must be available. The non-posted VC needs only one respectively two credits. One credit is needed for IO write requests as always only one doubleword is transmitted. Two credits are needed if the completer supports atomic requests. The maximum number of credits for headers are 128 units and 2048 for data credits. In case of non-posted data credits the number can be safely reduced to 128 or 256 because of the maximum possible payload for IO requests and atomic operations.

The type field encodes if it is a normal flow control packet or one used during initialization. In addition, it contains the information which kind of sub VC is updated and of which priority. Inside the packet one byte is used for the amount of available header credits and 12 bits are used for the data credits. The fields carry the actual value of the available credits and therefore a lost flow control DLLP does not lead to a loss of credits. The structure of the header is shown in figure 3-31.

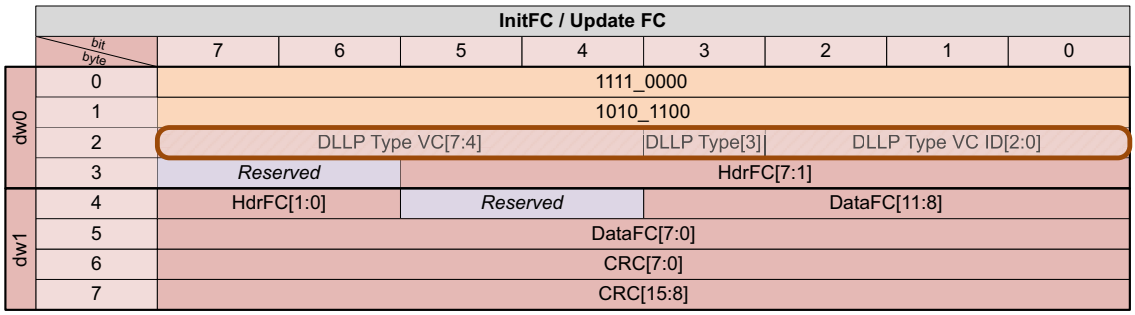


Figure 3-31: Flow Control Header Format

For power management the DLL provides dedicated power management DLLPs. They are used to change between different power management states. The lower 3 bits of the type field encode to which power state a transition should be made. Besides the type the rest of the packet is empty. The structure of the header is shown in figure 3-32. As the power management is not of interest for this thesis it won't be described further.

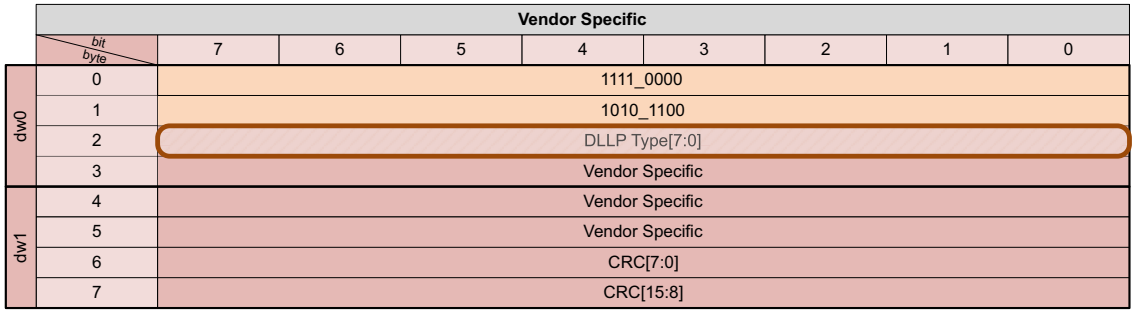


Figure 3-32: Power Management Header Format

Also vendor specific DLLP can be used in PCIe, but as they are completely optional and device specific they are left open.

From the framing side the DLL always adds the sequence number in front of the corresponding packet, regardless if it is a TLP or a DLLP. With the sequence number the receiver of a packet can monitor if one packet is missing and therefor start the error handling. As already mentioned the sequence number is also needed to release packets from the retransmission buffer of the sender or to start a retransmission.

The DLL also adds the Link CRC (LCRC) to the data stream after every packet. With this LCRC the data is protected on the link. For more details please refer to chapter 3.9.7.

3.9.3.3 Physical Layer

The physical layer of PCIe has three main tasks. One is to add frame tokens to the data stream, the second one is to communicate with the other physical layer connected to the link, and the third one is that it adds line coding to the single lanes.

Line Coding and Frame Tokens

In Gen1 and Gen2 8b/10b-coding is used for line coding. Therefore every 8 bits symbol is changed into a 10 bits line coded symbol character. An example is shown in figure 3-33. The advantage of 8b/10b-coding is for 8 bits only 256 characters are needed but 1024 are available, which allows for them to be smartly chosen. In case of 8b/10b the coding is chosen in a fashion that the data transmission is DC balanced. For every possible symbol two encodings are possible and chosen based on the current running disparity, one is used if the disparity is positive and one if it is negative; if it is neutral both can be chosen. Those characters are called D-characters.

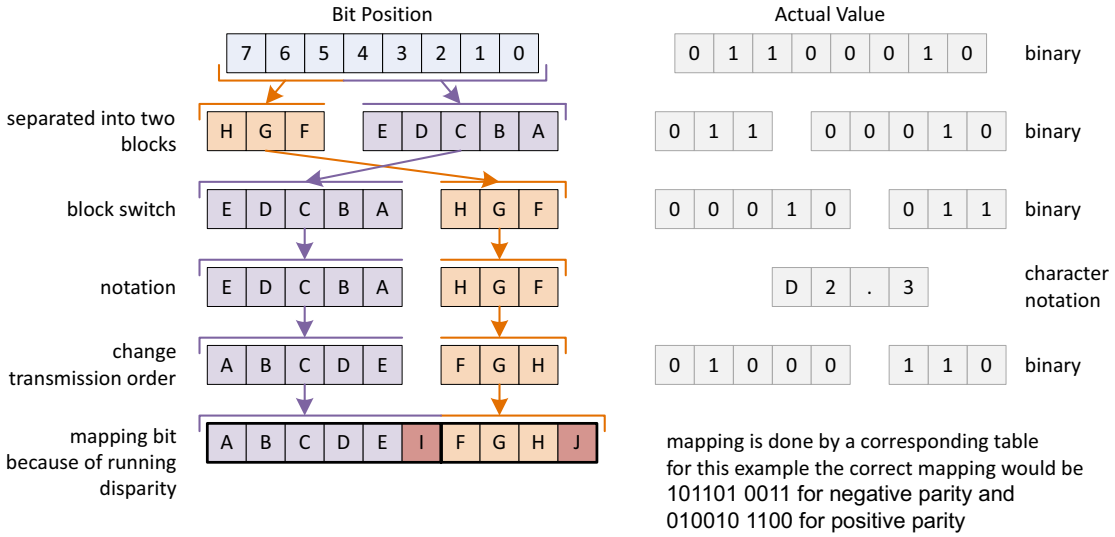


Figure 3-33: 8b/10b Line-Coding Example

With the left open coding space also special characters are available which cannot occur in the normal encoded data stream, those are called K-characters. In PCIe K-characters are used to transfer the frame tokens. Thus, as long as the symbol lock is not lost the framing of packets can always be regained. The frame tokens are shown in table 3-10.

Character	8b/10b	Coding	Description
SKP	K28.0	- 001111 0100 + 110000 1011	SKiP: used for clock tolerance compensation
FTS	K28.1	- 001111 1001 + 110000 0110	Fast Training Sequence: exits from L0s low power state to L0
SDP	K28.2	- 001111 0101 + 110000 1010	Start DllP: signals start of DLLP
IDL	K28.3	- 001111 0011 + 110000 1100	IDLe: switch link into electrical idle state
COM	K28.5	- 001111 1010 + 110000 0101	COMma: sent in front of every OS, used for symbol lock during training
EIE	K28.7	- 001111 1000 + 110000 0111	Electrical Idle Exit: ramp speed to higher frequencies than 2.5 GT/s
PAD	K23.7	- 111010 1000 + 000101 0111	PAD: used to pad unused lanes in one symbol time
STP	K27.7	- 110110 1000 + 001001 0111	Start TIP: signals start of TLP
END	K29.7	- 101110 1000 + 010001 0111	END: signals end of TLP or DLLP without an error
EDB	K30.7	- 011110 1000 +100001 0111	EnD Bad: signals end of TLP or DLLP with an error

Table 3-10: Gen1/Gen2 K-Character Token

An additional advantage of the 8b/10b coding is that signal changes are guaranteed so that not more than 5 consecutive bits with the same polarity can be transmitted in a row. This eases to keep the CDR locked.

The huge disadvantage of 8b/10b is that 20% of the bandwidth is lost with the coding. Therefore, line coding was changed to 128b/130b-coding in Gen3. This is also the needed change to double the bandwidth from Gen2 to Gen3 without doubling the line rate. The doubled frequency would be 10GT/s, if it is reduced by 20% it is only 8GT/s as it is in Gen3. Of course this calculation is not completely correct as the 2 bits overhead for the new line coding was added, but this is only ~1.5%. Compared to 8b/10b, 128b/130b is simpler. Every 128 bit times 2 bits are added to guarantee a polarity change. The type is not determined by the previous traffic but by the next data block. If the next block is a data block than in the first bit time of the 2 bits sync symbol a 0 is transmitted and in the second bit time a 1 (01). In case the next block is an ordered set block the first bit time is a 1 followed by a 0 (10). All other bit combinations are forbidden.

Without the K-character the frame token had to change as the bit-pattern is no longer unique in the data stream. Therefore insignificant tokens have been removed and other ones have been changed. The most drastic change was to remove the end token. Thus, the start token needed a length field to encode how long the overall packet is to know when the current packet ends and the next token is received. Additional protection was added to the token by adding a CRC and a parity bit. This ensures that a 3 bits error can always

be safely detected. The sequence number is now embedded into the start of TLP (STP) and is no longer transmitted separately. The Gen3 frame tokens are shown in figure 3-34.

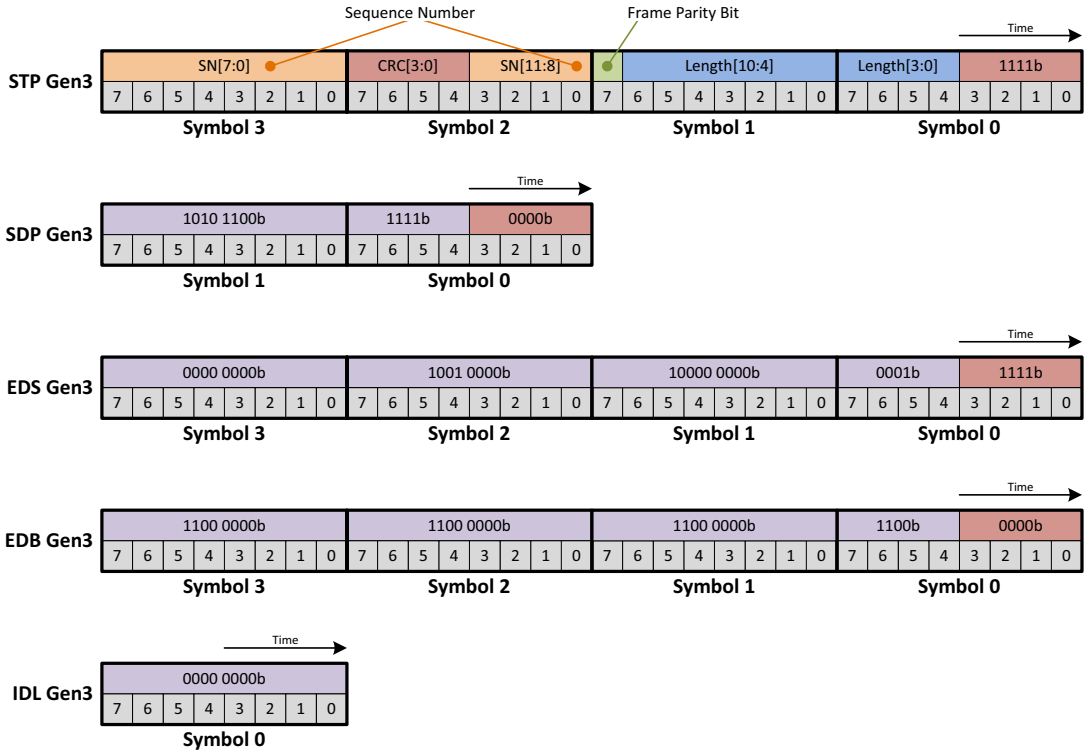


Figure 3-34: Gen3 Frame Token Overview

The end of data stream token (EDS) token is used to end the transmission of data and to start an ordered set block. Therefore, it is only sent at the end of a data block on the last doubleword before an sync character.

When the sender of a TLP detects an error during transmission an end bad (EDB) token can be sent after the packet. If this is done, the LCRC must be inverted. With those two indicators the receivers can ignore this packet completely. If the physical layer receives an EDB it must inform the DLLP so it can check the inverted CRC and drop the packet.

The last token is the logical idle (IDL) token. It is sent if no valid TLP or DLLP is available for transmission. If this happens on a link with more than 8 lanes and the current packet ends on lane 3, the rest of the lanes must transmit IDL. In case IDL is sent on lane 0, all other lanes must also contain IDL.

Together with the frame tokens the traffic on the link, without the ordered sets is now complete, example traffic is shown in figure 3-35.

	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
Sync 0	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
Symbol Time 0 Block 0	SDP 0		DLLP 0		DLLP 0			
Symbol Time 1 Block 0	STP 0				TLP Header 0			
Symbol Time 2 Block 0	TLP Header 0				TLP Header 0			
Symbol Time 3 Block 0	TLP Data 0				TLP Data 0			
Symbol Time 4 Block 0	LCRC 0				SDP 1		DLLP 1	
Symbol Time 5 Block 0	DLLP 1				IDL	IDL	IDL	IDL
Symbol Time 6 Block 0	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL
Symbol Time 7 Block 0	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL
Symbol Time 8 Block 0	STP 1				TLP Header 1			
Symbol Time 9 Block 0	TLP Header 1				TLP Header 1			
Symbol Time 10 Block 0	TLP Data 1				TLP Data 1			
Symbol Time 11 Block 0	TLP Data 1				TLP Data 1			
Symbol Time 12 Block 0	LCRC 1				STP 2			
Symbol Time 13 Block 0	TLP Header 2				TLP Header 2			
Symbol Time 14 Block 0	TLP Header 2				TLP Header 2			
Symbol Time 15 Block 0	LCRC 2				SDP 2		DLLP 2	
Sync 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
Symbol Time 0 Block 1	DLLP 2				IDL	IDL	IDL	IDL
Symbol Time 1 Block 1	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 3-35: PCIe Link Traffic Example

Ordered Sets

In order to communicate with the other physical layer OSs are used. There are seven different order sets: electrical idle exit (EIEOS), electrical idle (EIOS), fast training sequence (FTS), start data stream (SDS), training sequence 1 (TS1), training sequence 2 (TS2), and skip (SKP). OSs are not sent as packets but as patterns which must be transmitted on all lanes simultaneously. The first symbol always identifies the corresponding OS. In Gen3 every OS is 16 symbols long and therefore exactly one block size. There is one exception which is SKP, which can be 8, 12, 16, 20, or 24 symbols long.

The EIEOS is used to exit from the electrical idle state in Gen3. As in high frequencies the change of the electrical signal is so quick it might not reach a high voltage. Therefore, it is harder to distinguish between the electrical idle state and operation mode. Thus, EIEOS provides a sequence which is a lower frequency signal by sending one byte of 0 followed by one byte of 1. This makes the operational mode easier to detect.

EIOS is used for power management to inform the link partner that the link will be set into tristate. Pattern is 0110_0110 for all symbol times. If the receiver gets this OS it knows that the transmitters of the sender will go into tristate and the missing signals will not lead to an error.

The FTS is used to set the link from a low power state to the fully operational state. The minimum number of needed sets is given by the link neighbor during training. For Gen3 FTS consists of 16 different symbols, they are shown in table 3-11.

Symboltime	FTS
0	55h
1	47h
2	4Eh
3	C7h
4	CCh
5	C6h
6	C9h
7	25h
8	6Eh
9	ECh
10	88h
11	7Fh
12	80h
13	8Dh
14	8Bh
15	8Eh

Table 3-11: Fast Training Sequence Ordered Set

Before starting a data stream an SDS must be transmitted. It consists of one symbol E1h followed by 15 symbols of 55h.

Two devices with separate clock sources never run with the exact same frequency. Therefore a mechanism is needed to compensate this frequency difference. In PCIe SKP is used for compensation. It must be inserted at least every 370 – 375 blocks and is simply dropped by the receiver to avoid an overflow. The SKP consists of blocks of 4 AAh symbols followed by one symbol-time E1h which signals the end of the OS. The last three bytes consist of AAh, data parity, the entry of the LSFR, or error status, for detailed information please refer to [16].

TS1 and TS2 are used for link initialization and training. With those symbols the link is able to achieve bit lock and symbol lock and is able to configure the link. The structure of TS1 and TS2 for Gen3 is shown in table 3-12.

Symboltime	Symbol	Description
0	OS Identifier	TS1=1Eh, TS2=2Dh
1	Link Number	PAD (F7h) if in polling state, otherwise link number
2	Lane Number	PAD (F7h) if in polling state, otherwise lane number
3	Number of FTS	Required number of FTS order sets for L0s recovery
4	Rate ID	Indicates the supported link frequencies *
5	Train Ctl	Communicates special conditions *
6 9	EQ Info	Contains values for the Equilization process
10 13	TS ID	TS1=4Ah, TS2=45h
14 15	TS ID	Contains symbols regarding to the running disparity*

*for more details please refer to [16]

Table 3-12: Training Sequence Ordered Sets

Scrambling

Signal transitions are needed to be able to recover the clock from the signal. Therefor PCIe provides scrambling. For Gen1 and Gen2 the traffic is scrambled before it is 8b/10b coded. The traffic which is scrambled are data blocks and the main parts of TS1 and TS2. Symbol 0 of the training sets always bypasses the LFSR, Symbol 14 and Symbol 15 only bypass to enhance DC balance, otherwise they are also scrambled. The rest of the traffic, the main part of the ordered sets bypass the scrambling logic as they are DC balanced and provide enough signal changes. All traffic sent over the link also advances the LFSR even if it bypasses, the only exception is the SKP pattern. The polynomial used for the LFSR is shown below.

PCIe Scrambling Polynomial: $X^{16}+X^5+X^4+X^3+1$

3.9.4 Dependencies

Dependencies in and among packets types must be analyzed slightly differently in PCIe compared to HT, as the traffic is already logically separated into different layers. This is already a tradeoff between latency and hardware complexity. Also the bandwidth can be influenced as it is doubtful that the same frequencies could be achieved if layers would be merged together and therefore the internal bandwidth would decrease. The amount of decoding is reasonable for the single layer as only a well-defined portion of the stream has to be decoded, tokens in the physical layer, in the next layer DLLP and the LCRC, than TLP. The header types of the different layers also show that they have not many dependencies inside the corresponding header and most of them can be clearly determined with the first byte of the header, but not more than the first two bytes of the header are needed.

The difference of the packet structure among packets of the same layer is also relatively low. The most interesting layer for this consideration is the transaction layer as there are the most differences and the most complex packet structure. The first two doublewords are always identical in case of field position and meaning, only some packets have more reserved fields. In doubleword three and four the routing information is stored and how the transmitted data has to be stored. Only the messages have one byte in there which triggers the corresponding action.

Figure 3-36 is separated into two parts by the alignment grid. The part above the grid shows the header type for each layer of PCIe. Within one header type the different meanings of the corresponding header fields is illustrated with additional layers. The part under the alignment grid shows the merged doublewords of all header types of all layers. This has been done to depict the overall complexity of PCIe to make it comparable to other protocols. It can be seen that PCIe contains a large complexity if all layers are included as the layer height is 20. This would represent a high muxing effort for the different doublewords if everything had to be handled in one layer.

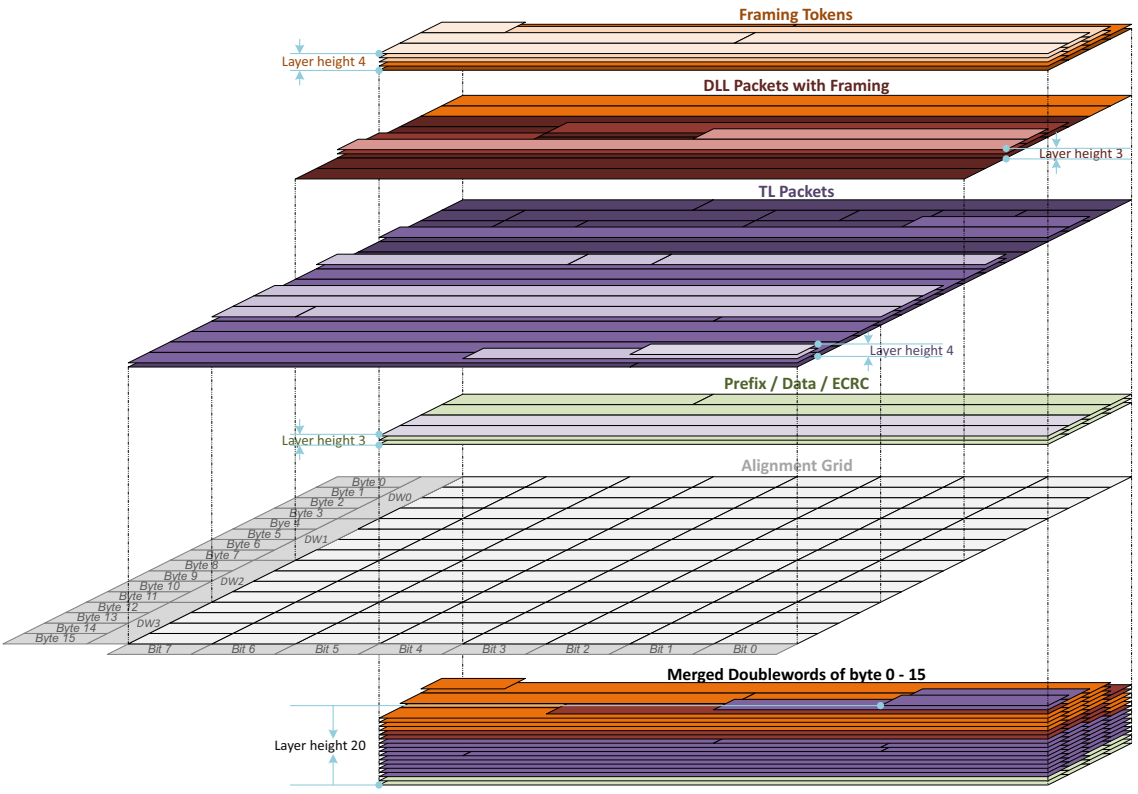
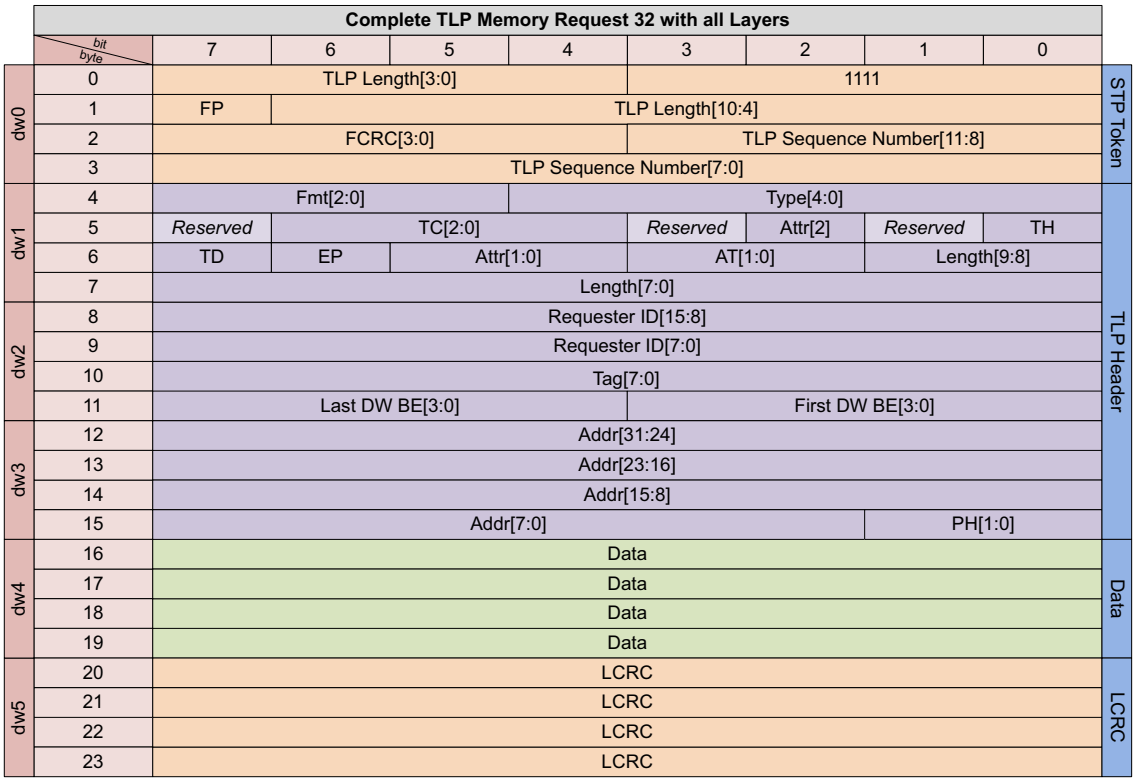


Figure 3-36: PCIe Doubleword Dependencies of Traffic

The influence of the data width is relatively low compared to HT. That is because the overhead for one packet is relative large. At least 5 doublewords overhead are needed for any packet which transfers data. An example is shown in figure 3-37. This results in a data width of at least 160 bits without data and 192 bits with data before a second packet could start in the same clock cycle.



Complete TLP Memory Request 32 with all Layers										
	byte \ bit	7	6	5	4	3	2	1	0	
dw0	0	TLP Length[3:0]				1111				STP Token
	1	FP	TLP Length[10:4]							
	2	FCRC[3:0]				TLP Sequence Number[11:8]				
	3	TLP Sequence Number[7:0]								
dw1	4	Fmt[2:0]			Type[4:0]					TLP Header
	5	*Reserved*	TC[2:0]			*Reserved*	Attr[2]	*Reserved*	TH	
	6	TD	EP	Attr[1:0]		AT[1:0]		Length[9:8]		
	7	Length[7:0]								
dw2	8	Requester ID[15:8]								
	9	Requester ID[7:0]								
	10	Tag[7:0]								
	11	Last DW BE[3:0]				First DW BE[3:0]				
dw3	12	Addr[31:24]								
	13	Addr[23:16]								
	14	Addr[15:8]								
	15	Addr[7:0]						PH[1:0]		
dw4	16	Data								Data
	17	Data								
	18	Data								
	19	Data								
dw5	20	LCRC								LCRC
	21	LCRC								
	22	LCRC								
	23	LCRC								

Figure 3-37: Minimum PCIe Packet with Data

In order to reduce decoding effort some restrictions exist in the PCIe specification. Only one STP token at one symbol-time. Same as the STP is for the SDP token as also only one SDP token is allowed in one symbol-time. If the link was idle the next STP or SDP is only allowed to start in lane 0. But both start tokens are allowed to be in the same symbol-time. This is only relevant for 32 bits wide links as one packet utilizes at least 20 lanes. Thus, 12 lanes could be idle if only TLPs should be sent and no DLLPs. But even if a DLLP is sent in the same symbol time it always fits into 8 lanes because its size is fixed. Therefore on 4 lanes IDL has to be sent and one doubleword of bandwidth would be lost. DLLPs without TLPs would be more critical as they utilize only 8 lanes, but if only DLLPs have to be sent the link is practically idle.

No relevant restrictions are made regarding to where a packets starts on the link. As PCIe is doubleword aligned and symbols are transmitted on each lane it only can start every doubly even lane. But on every quad bundle of lanes every type of packet can be transmitted. Only after IDL or OS have been transmitted the next STP or SDP must start at lane 0.

3.9.5 Initialization

During initialization several conditions have to be met. The link must be detected, the correct parameters of the link have to be found, and the link must be configured correctly. Therefor PCIe provides the link training and status state machine (LTSSM). An overview of the state machine is given in figure 3-38. This is a brief overview of the initialization sequence of PCIe and not complete in all details. Every state in the LTSSM figure represents a small sub-state-machine. For more detailed information please refer to [16].

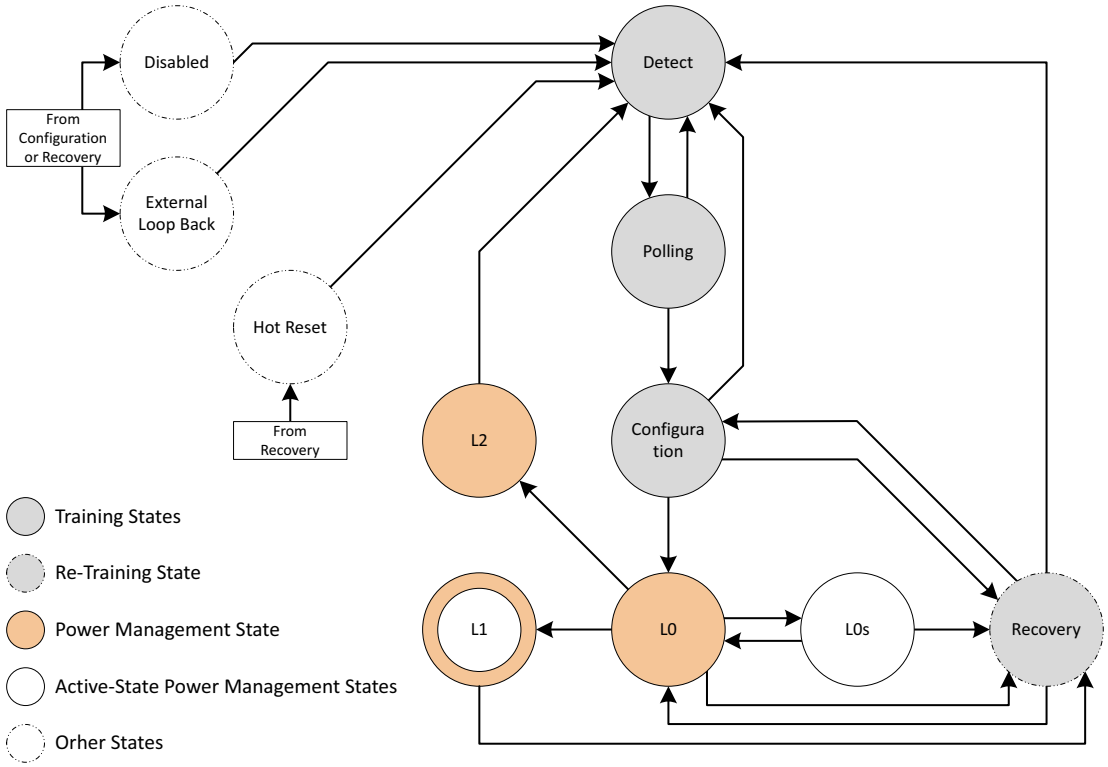


Figure 3-38: LTSSM State Machine

The detect state is the initial state after reset. In this state it is detected if a receiver is connected to the link. In the polling state training sequence order sets are exchanged. They enable the link to find bit lock, symbol lock, exchange the data rates, and invert the lanes if needed. During the configuration state still training sets are exchanged to merge the working lanes into one link. The link width is set up, the lanes are enumerated, lane reversal may be performed, and the skew among the lanes is removed. At this point data can be exchanged between the two participants of the link and the LTSSM can move to L0 which is the working state for PCIe.

The recovery state is used to retrain the link. This is necessary if the link frequency has to be changed, an error occurred, or if a power saving state has to be left without using FTS. L0s, L1, and L2 are power saving. L0s does save only little power but therefore it can quickly recover from power saving by sending FTS, if this does not work it can be also tried to recover by using the recovery state. L1 saves more power but always has to go through recovery and configuration to get back to L0 and a working link. In L2 the power of the device is shut down and therefore the complete initialization process has to be performed for a working link afterwards. If the recovery does not work correctly also a complete initialization has to be made.

The loopback state is used to test the device. During this state the master device sends TS1 with the loopback bit set and if two of those ordered sets are received at the slave, the slave simply mirrors the traffic back to the master. The disable state is reached if the link is not working anymore, for example if a device is removed during operation. In this state the senders are disabled and the receiver is set to low impedance. The hot reset state is entered if software sets the corresponding register. Then TS1 are sent with the hot reset bit set. If a receiver gets two consecutive order set of this type the device has to be reset also.

3.9.6 Bandwidth

The theoretical maximum bandwidth of a 32 lane link operating at 8 GT/s is 32 GB/s. As PCIe does not use any sideband signals as HT they do not have to be taken into account. In order to determine the real bandwidth the protocol overhead must be removed from the theoretical bandwidth.

One overhead which has to be removed is the per packet overhead. In order to calculate the highest possible bandwidth the lowest packet overhead with the largest amount of data is assumed. As already mentioned the lowest overhead of one TLP are 5 doublewords consisting of 1 doubleword STP, 3 doublewords TLP, and 1 doubleword LCRC. The largest amount of data is 1024 doublewords. In this calculation it is assumed that flow control and needed buffer size do not influence the data transmission.

$$\text{Bandwidth without Header Overhead: } 100 \cdot \frac{1024}{(1024 + 6)} = 99.41\ \%$$

The other relevant overhead is the periodical overhead during link transmission as for line coding. In Gen3 the line coding consumes 2 bits every 128 bit times.

$$\text{Bandwidth without Line Coding Overhead: } 100 \cdot \frac{128}{(128 + 2)} = 98.46\ \%$$

The frequency compensation is handled with SKP which is removed from the data stream at the receiver. A SKP must be transmitted every 370 to 375 blocks and has a minimum length of at least 8 symbols. As a block consists of 16 symbols it must be sent every 6000 symbols.

$$\text{Bandwidth without SKP Overhead: } 100 \cdot \frac{6000}{(6000 + 8)} = 99.86\ \%$$

This results in a reachable maximum bandwidth of 97.74% of the theoretical maximum bandwidth.

$$\text{Accumulated Bandwidth: } 99.41\ \% \cdot 98.46\ \% \cdot 99.86\ \% = 97.74\ \%$$

3.9.7 Fault Tolerance

In PCIe three types of errors are defined: correctable, non-fatal uncorrectable, and fatal uncorrectable errors. Correctable errors are handled autonomously by hardware. They only influence the performance of the system but no information is lost. A non-fatal uncorrectable error comes along with loss of data but the link is still working. This can happen when a packet is somehow lost in the system but the link is working without any problems. An example would be a completion timeout. That kind of error is not handled by hardware but software could be able to handle the situation. A fatal uncorrectable error occurs if nothing but a reset of at least the link can fix the error.

PCIe supports various mechanisms for error detection. The most common one is that most of the traffic is protected by CRC. For three traffic types a CRC is required: TLPs, DLLPs, and STP. TLPs are protected with the 32 bits wide LCRC. It protects the whole TLP in-

cluding the sequence number. DLLPs are protected with a 16 bits CRC. The length field of STPs is protected with a 4 bits frame CRC (FCRC). In addition the length field and the CRC are protected. The corresponding polynomials are shown below.

LCRC/ECRC: $X^{26}+X^{23}+X^{22}+X^{13}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$

DLLP CRC: $X^{12}+X^{3}+X+1$

FCRC: $X^{4}+X+1$

The LCRC protects TLPs only on the link between two DLLs. Thus, a TLP is not protected after it has been checked in the DLL. If an error occurs afterwards the packet could be corrupted without noticing it. Therefor PCIe provides the optional 32 bits ECRC which is also called digest. As 2 bits in the header are allowed to change during transmission they are assumed to be 1 for the CRC calculation. Those bits are bit 0 of the type field an EP. A flip cannot be covered by the ECRC. The last bit of the type field can change if a configuration access has reached its corresponding bus. EP changes if poisoned data is signaled. As can be seen, the ECRC uses the same polynomial as the LCRC. If prefixes are used the ECRC does not protect local prefixes as they are not part of the end to end traffic.

In case that a TLP or a STP is malformed the error results in a request for retransmission. A NACK with the corresponding sequence number has to trigger the retransmission of the data stream from this point. A corrupted DLLP is simply dropped. This can be done as the next DLLP will compensate the missed one or further error handling will start. ACKs or NACKs have to be inserted into the data stream in a certain time period after a TLP has been received. If the sender of the TLP does not an ACK or NACK in time it retransmits its retransmission buffer so a lost ACK or NACK can be compensated.

Additional to the CRC checks the different layers also check for other errors. In Gen1 and Gen2 the physical layer checks if the coding of the characters fits the 8b/10b coding. In Gen3 the framing is checked as the physical layer always knows at which point the next frame token should be.

The DLL also checks the sequence number to detect errors. The sequence number must always be ascending. Thus, if a wrong sequence number is detected it signals a missing TLP or an error which was undetected from the CRC. In this case a retransmission has to take place. In addition, the counters which check if an ACK or NACK for a packet was received are monitored here and initiate a retransmission from the retransmission buffers.

On the TL the header is checked if it is malformed. For example if fields that are required to carry 0 do not contain any other values. It is also checked if an unsupported packet has been received.

Errors are always reported even if they are correctable. With the information software can decide if it has to take action or not. Some checks of PCIe are optional, but if those errors are checked they also have to be reported. The number of allowed retries in a row is limited to four. If then still no new ACK is received it is assumed that the link has a permanent failure and a recovery of the link is performed.

In summary, it can be said that the headers of the PCIe protocol are arranged efficiently as there is little overlapping. The layered structure simplifies the implementation in hardware but increases the latency due to additional pipeline stages. Additionally PCIe provides a very low overhead as 97.85 can be used for payload transmission. A detailed comparison of PCIe and HT is provided in chapter 4.13.

Chapter 4: Unified Layer Protocol

In this chapter the Unified Layer Protocol (ULP) will be introduced, which was designed to fulfill three required main goals:

- Replace all the performance critical protocols of a system with a single unique protocol type to reduce the translation overhead
- Improved usability of the protocol in large scalable networks
- Advanced protocol structure for efficient implementation in hardware

First the different design decisions will be presented which result in a specific protocol structure. At the end the performance of ULP will be compared to HT and PCIe.

4.1 Device Types

Target device types are grouped into two subtypes, network devices and node devices. Node devices also have two subtypes which are switches and caves. An overview of the devices is given in figure 4-1. At the beginning the network device which is the root must initialize all devices by distributing the IDs. This root device has the node ID 0.

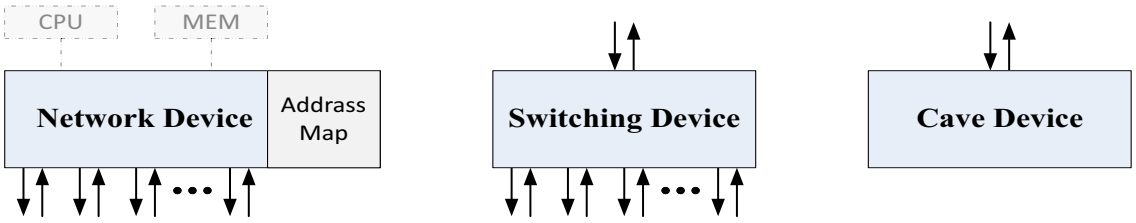


Figure 4-1: ULP Device Types

Network devices have an address map so they are capable of sending packets directly to a corresponding node based on a target address. The address is translated into a node ID and a network ID, afterwards it is determined which link has to be used to transmit the packet. Node devices have no address map and route their packets implicitly upstream to the root device which has initialized the node IDs. The link with the fewest hops to the root is marked as the upstream link. At the first network device found upstream the target address is mapped to the corresponding target ID and the packet is forwarded. A response could be directly returned to the requester as the request contains its ID. However, to en-

sure the ordering of the packets the responses have to take the same route as the requests. In order to be able of ID based routing node devices can contain a node address map if it is sure that it only communicates with other devices inside one node. Therefor every device stores which link has to be used for a corresponding ID window. The packets are forwarded upstream as soon as one link provides the ID window. Then the ID window path is used until the corresponding device is reached. An example topology is shown in figure 4-2.

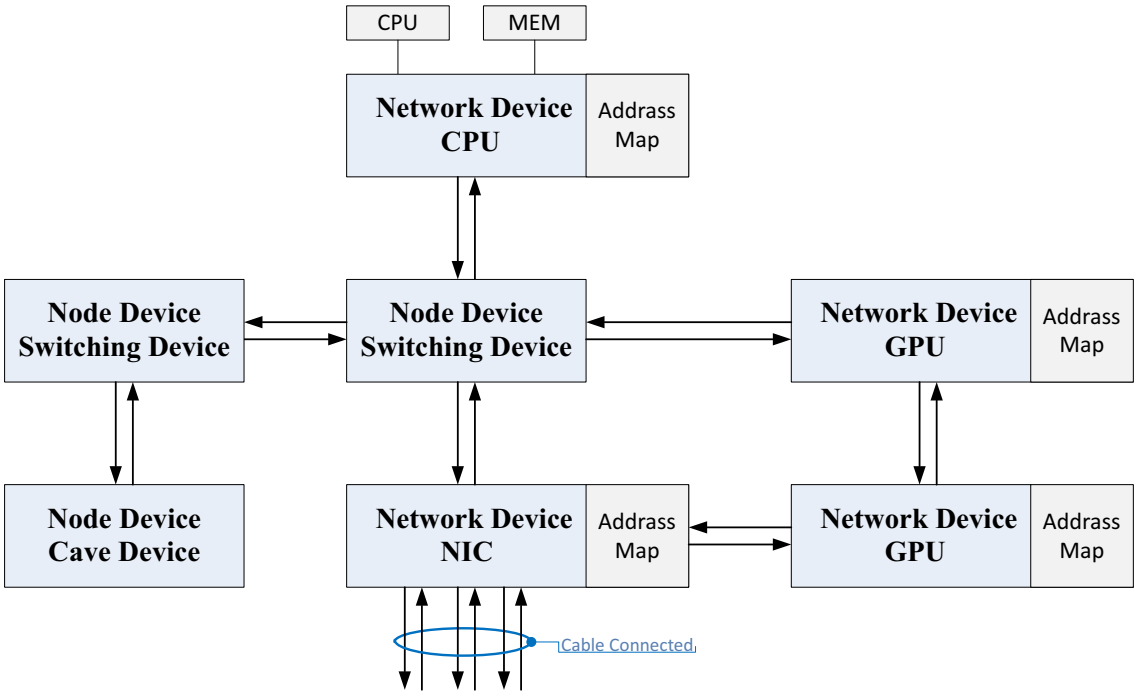


Figure 4-2: ULP Example Topology

In figure 4-3 the initialization of the example topology is shown. During initialization process there is only a single active configuration request allowed. The response must be returned before the next one can be sent by the root. Every device has to enumerate its links. The first initialization phase is sent to link 0 of the root and sets the node ID of the device and reads the number of active links from the device, which must be at least one. If more than one link is available the next link is used to forward the configuration packets. This is repeated until a cave device is found or a node which is already initialized. If a cave is found the link is marked as finished and upper devices are checked for uninitialized links. In case all links are initialized they are also marked as finished. Otherwise the uninitialized branch will be configured. It can happen that during initialization an already

configured node is found. In this case the distance to the root must be checked. If it is shorter the link to root must be changed to this link and previously found links which are not marked as finished must also be checked if they use the correct link for implicit routing. If all links are marked as finished the node is ready.

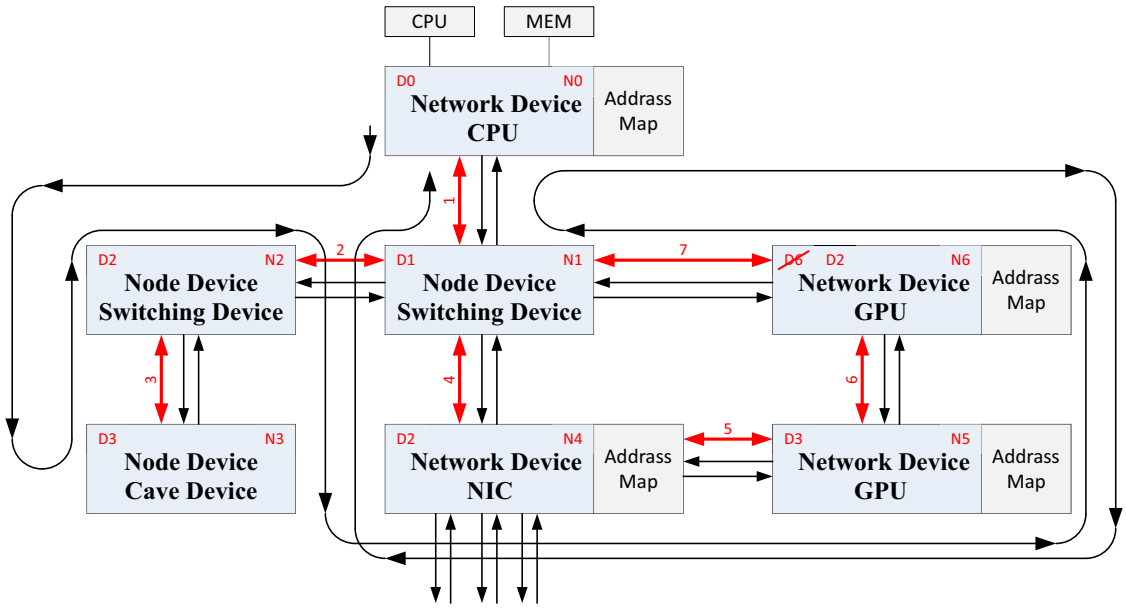


Figure 4-3: ULP Node ID Initialization

4.2 Physical Layer

On the physical layer the actual data bits of the stream are transmitted. Its composition is defined by the bandwidth it has to deliver. A defined minimum bandwidth is useful to ensure that different devices can communicate with each other regardless of their typical performance. It is determined by the minimum number of lanes with the lowest allowed link frequency. A maximum link bandwidth is important to get a upper bound for the internal bandwidth. Theoretically the link bandwidth can always be increased by adding lanes or increasing the frequency. However, the internal bandwidth is not so easy gradable. Nevertheless, those numbers are only needed for compatibility reasons and are not tightly coupled with the protocol itself.

A link must always provide lane 0 as this is the minimum width configuration for a link and it must be contained by all link widths. Transmit lane 0 must always be connected to

receive lane 0 of the link partner, a misalignment is not allowed. The link has to be symmetrical. In case unsymmetrical link widths are detected the link is initialized to the highest configuration both link partners support.

4.2.1 Lanes

For communication ULP provides a specific number of lanes which can be 1, 2, 4, 8, 16, or 32. From the connector side it would also be interesting to support 12 bits wide lanes as if the connection has to leave the node via a PCIe-like extender card common high frequency connectors like [57], [58], or [59] often use quad based connections. The problem with this is that if the protocol is based on the power of two a 12x connection automatically leads to a misalignment of the data stream. Thus, a rate converter is needed which handles the misalignment.

4.2.2 Sideband Signaling

In order to reduce the pin count of a chip and the lane count for connectors no sideband signals are used for things like link management or clock transmission. Thus, the clock has to be embedded into the signal. Therefor a line coding has to be used. Many different line codings are available. One common line coding is 8b/10b coding like in PCIe Gen1 and Gen2 or EXTOLL. The advantages would be the additional security, build in DC balance, and easy clock recovery as only 5 bit times without a signal change are allowed [53]. However, it has two disadvantages. One is it comes with a 20% overhead as to transmit 8 bits 10 bits are used. The other one is that packets have to be transmitted in symbols. Thus, data is not received in bit times but in symbol times which increases the received data at a time period.

An alternative would be 64b/66b line coding. Every 64 bit times a 2 bits sync pattern is inserted into the data stream. In order to guarantee a signal transition the only allowed patterns are 10_2 and 01_2. It does not have the advantages of 8b/10b like additional security and DC balance but is capable of clock recovery. A scrambler should be used to enhance the probability of signal transitions. The functionality of this coding is proven in different

protocols like 10G Ethernet [17] and Infiniband [18].

An additional choice is 128b/130b coding like it is used in PCIe. It is similar to 64b/66b but only the number of bit times without a guaranteed signal transition is increased to 129. As for 64b/66b a scrambler should be used to enhance the chance of signal transitions.

All three versions have advantages and disadvantages, but as the developed protocol has to provide the maximum performance 128b/130b is chosen as it is the coding with the least overhead. Only ~1.5% of the bandwidth is used for the sync pattern and the rest can be used for packet transmission.

4.2.3 Frequency

Common frequencies like 2.5 GT/s up to 10GT/s are supported by ULP. More important as the plain frequency of the link is the achievable bandwidth and the resulting internal frequency of the device. For example if a maximum link width of 32 bits and a frequency of 10 GT/s are assumed the resulting bandwidth is 40 GB/s. Thus, for an internal data width of 128 bits an internal device frequency of 2.5 GHz is needed to handle the link bandwidth. This seems feasible for CPUs and GPUs. If for some reason it is not feasible for a device smaller link widths or lower link frequencies have to be used.

4.3 Layers

ULP provides four different layers. The first layer is the physical layer. It is responsible for scrambling, line coding, and the hardware initialization sequence.

The second layer is the info layer. This layer is used only for communication between link partners. Information for flow control and error handling is exchanged in this layer. Info packets must always be accepted and are not allowed to be buffered as they are essential for the correct functionality of the link.

For coherent devices a third layer is assumed which handles the traffic needed for coherency. The corresponding traffic is extracted from the data stream and the rest is forwarded

to the non-coherency layer. This is done to avoid timing critical large headers and data transfers to achieve the needed performance for efficient coherency timing. This layer is not part of this thesis.

The fourth layer is the data layer. In this layer the actual request and response packets are transmitted and data is transferred. A layer overview is shown in figure 4-4.

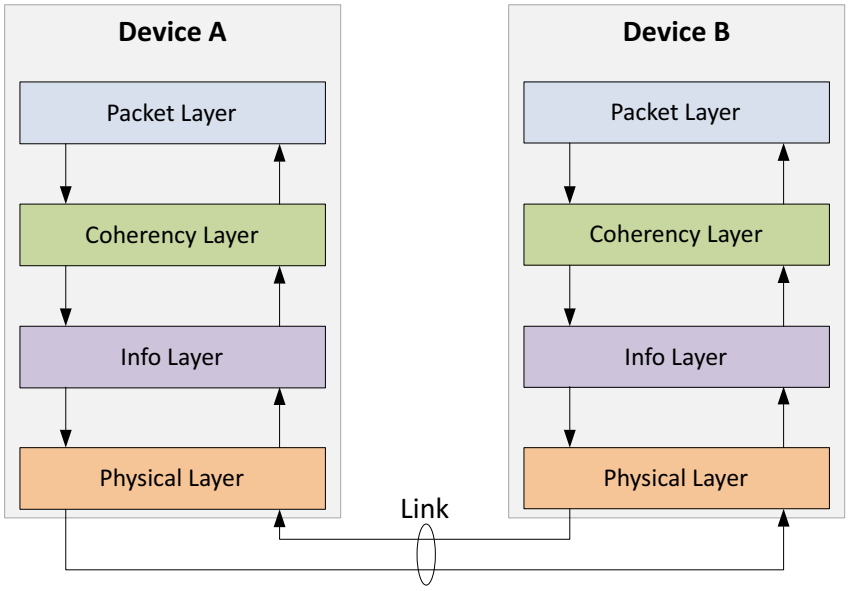


Figure 4-4: ULP Layer Overview

4.4 Flow Control

In case a small packet is transmitted it is unlikely that it is able to utilize the whole link. Therefore, more packets have to be transmitted to utilize the bandwidth. At the receiver side a packet first must be checked and then forwarded to its destination to ensure it is correctly received. Thus, a mechanism is needed to ensure that a packet is not overwritten by further traffic. A credit based flow control is used in ULP to handle the correct functionality.

For small packet sizes one credit per packet is suitable. It is easier to handle as no calculation has to take place to determine if the receiver is capable of storing an additional packet. The additional needed buffer size is insignificant especially if no long distances have to be covered which would result in more outstanding packets. However, ULP pro-

vides a rather large packet size with 4 KB of data payload and long distances for a link. Thus, a packet based credit handling is not appropriate. ULP provides two types of credits, header credits and data credits. One header credit always represents one complete header regardless of its size, while a data credit has the granularity of one octaword (OW).

For easy handling it is suggested that one header as well as 16 bytes of data represent one entry in the VC buffer. This eases the coordination with the retransmission buffer and always one credit is released if an entry is consumed. Otherwise it has to be monitored how many credits are released corresponding to one entry.

For the correct behavior of the link a credit packet must be inserted to the data stream at least every time 4 KB are consumed from one VC buffer. It does not have to be inserted immediately but right after the current packet in transmission. In case multiple buffers have finished such a sequence the credits must be released back to back.

4.5 Buffer Size

The buffer size is mainly dependent on the number of available credits and the number of outstanding requests. In order to be more flexible it is current to split larger packets into multiple flow control digits (FLITs). Thus, the buffers can be more efficiently used as a small packet does not require the same buffer space as a large one. However, there are also restrictions from the hardware size. It is important that the buffer sizes are not allowed to be unrealistic large if the specification is fully utilized. For example, if a credit count of 20 bits would be allowed with a FLIT size of 32 bits the corresponding buffer would have a size of at least 4MB. The size of the count field would be possible; the size for one VC buffer is unrealistic. Therefore, a reasonable value for buffer size and credits has to be found.

4.5.1 VC Buffer

The size of a VC buffer depends on two factors. One is the amount of traffic which is needed to fully utilize the link until a credit is returned and the second one is the packet

size. For the estimation the maximum link width is used in combination with long cables and large packets as this is the worst case for the buffer size.

Four requirements have to be fulfilled to hide the round trip latency. First the pipeline stages on the cable and in the devices to the receive VC buffer have to be utilized and stay utilized until the credit information has returned. The packet has to be completely checked before it can be processed. It also has to be extracted from the receive buffer before a credit can be released. In the worst case the link back to the sender can be occupied with a maximum sized packet which delays the credit. Figure 4-5 shows all the stages for the overall credit timing.

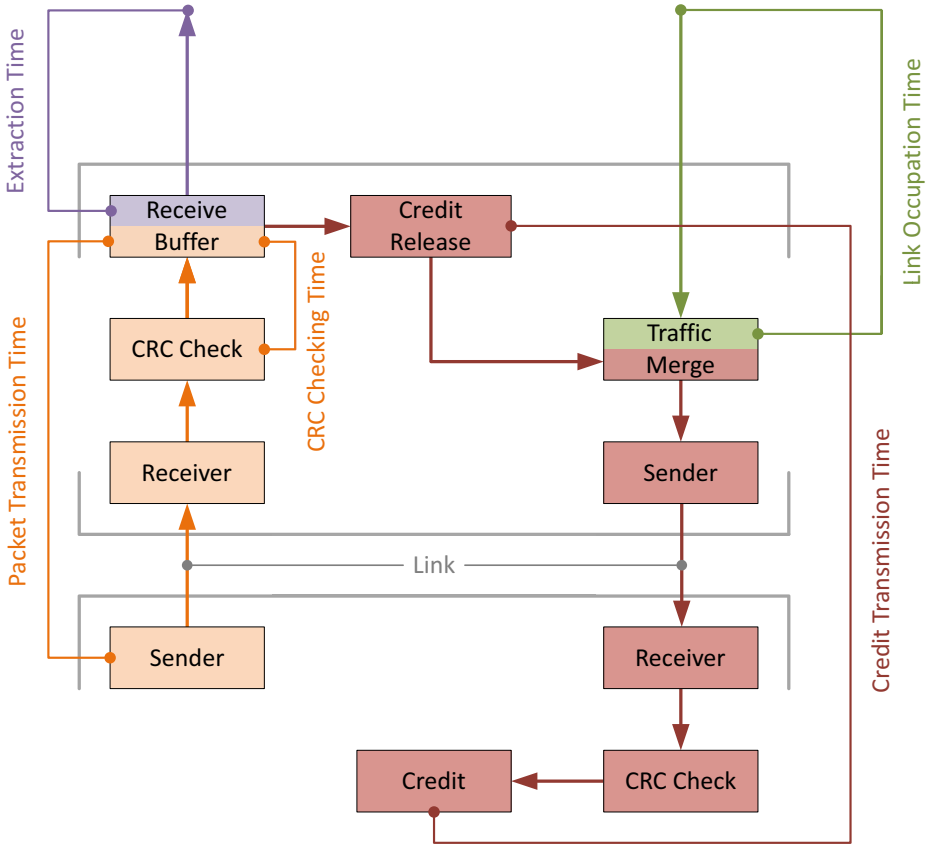


Figure 4-5: Stages Influencing the Credit Timing

The packet size is important especially if large packets are allowed to be transmitted. Thus, the misalignment from packet size to the data in transmission causes overhead as well as the checking of the packet, the packet extraction from the buffer, and the time needed to insert credit information into the data stream. If the data in transmission is not

a multiple of the packet size one additional packet must be partly transmitted to utilize the link. The impact increases with the size of the packet. At least four packets are needed to occupy the transmission path the whole time. One is needed to hide the transmission time on the link. Another one is needed for the extraction time at the receive buffer. A third one is needed to cover the transmission time of a current packet. The fourth packet is needed to hide the time the credit is sent back. It is possible that a fifth packet is needed to cover the path from the sender to the CRC checking unit. Therefore, buffer space for them has to be available. In table 4-1 an example calculation for the data path utilization is given.

Parameters	
Packet Size in Byte	4096
Propagation Delay Optical Fiber in km/s	185000
Link Frequency in GT/s	10
Cable Length in Meter	25
Link Width	32
Internal Data Width in Bit	128
Pipeline Stages until Transmission	5
Pipeline Stages until Reception	5
Pipeline Stages until Credit Transmission	5
Pipeline Stages until Credit Release	5

Transmission Time	
Seconds per Meter	5.4054E-09
Bits per Meter	54.05
Packets per Meter	0.002
Packets per Cable	0.04
Packets per Link	1.32

Round Trip Latency	
Packets per Pipeline Stage	0.004
All Pipelinestages needed	20
Packets in Pipeline Stages	0.078
All Packets (Pipeline Stages + Link)	1.40
Pipeline Stages plus Transmission Time	2.80

Buffer Size Needed in Packets	6
Buffer Size Needed in KB	**24**

Table 4-1: Virtual Channel Buffer Example Calculation

In this example the accumulated needed buffer space is 24 KB. 2.8 maximum sized packets are needed for the data path utilization. Three maximum sized packets are needed to hide the round trip latency. This offers a buffer to avoid a hiccup in the worst case and to hide the headers. Additionally, three maximum sized packets have to be added for checking, packet extraction, and the possibly occupied outgoing link.

In case two separate buffers are used to handle headers and data the needed buffer size increases as it cannot be qualified if only read requests or large write requests are transmitted. Additionally, the buffer size would increase for headers as they can be 192 bits wide but still 1536 entries are needed. This results in a buffer size of 36 KB. The buffer size can be minimized in two ways. One way is to reduce the VC buffer for headers as they are small and the time for checking and extraction is nearly irrelevant. This would reduce the buffer space to 1024 entries with a header of 192 bits the VC buffer size would be 24 KB. The other way to reduce buffer space is to merge the header and data buffer into one. However, this also increases the buffer size as headers can be 192 bits wide and to be able to access the whole header in one clock cycle the buffer must be wider than 128 bits. It must be increased by 64 bits which is 50% of the normal VC buffer size. This increases the overall needed buffer space for one virtual channel to 36 KB, but it still saves 12 KB compared to the separate model.

In order to show the impact of the cable length and the packet size figure 4-6 has been inserted. The left diagram shows the impact if the cable length is changed from 1 to 20 meters in meter steps. The right diagram shows how the packet size influences the buffer size. It can be seen that the misalignment causes an overhead until the packet size is a multiple of the data in transmission and how it gets worse for larger packet sizes.

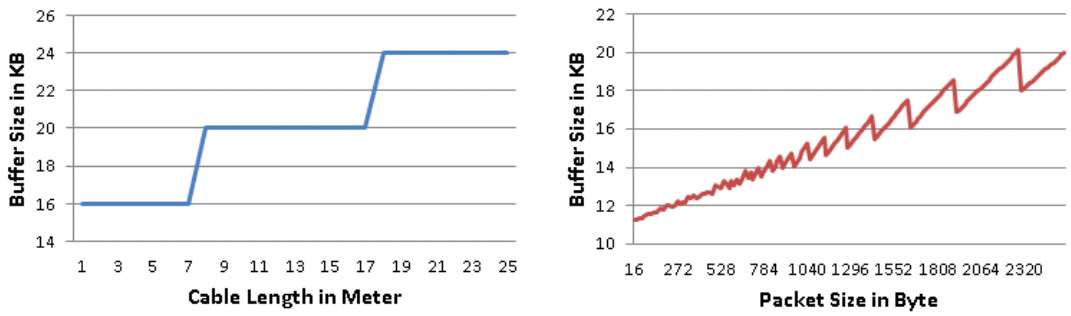


Figure 4-6: Influence of Cable Length and Packet Size to Buffer Space

4.5.2 Retransmission Buffer

The retransmission buffer must be able to store traffic that was sent to be able to retransmit it if an error was detected at the receiver. A simple and correct estimation would be that

all traffic which can be stored at the link input can be stored in the retransmission buffer. For this case the retransmission buffer would be rather large as it would include the size of all VC buffers.

A more elegant and resource saving solution is to set the size of the retransmission buffer to the same size as for one VC. This can be done as the error response for one packet can be inserted earlier as the fastest credit release. The buffer size can further be reduced as the packet does not need the time to be extracted, but it is good to keep this buffer for security reasons and it has advantages if the retransmission buffer size matches the VC buffer size. For more information please refer to chapter 4.5.1.

4.5.3 Tag Matching Buffers

A tag matching buffer is needed at the sender to merge responses to the corresponding request by a unique tag of every transmitted request. The size of the tag matching buffer is dependent on the time a packet needs to travel through the network and the response to get back. For the calculation a mixture of best case and worst case assumptions are made. For the best case it is expected that the request and the response are not delayed by any other traffic and therefore the round-trip latency can be used for the calculation. Otherwise the credits of every link have to be taken into account. For the worst case assumption it is expected that only the shortest tag consuming requests are generated and every link provides the maximum bandwidth. Thus, the maximum number of outstanding tags is used. If not enough tags are available the link is idle and bandwidth is wasted. In order to get a realistic number of outstanding packets it is not valid to assume that per hop only one packet is traveling and forwarding a packet is done in zero time. One realistic example is shown in table 4-2.

System Specification	
Request Packet Size in Byte	24
Propagation Delay Optical Fiber in km/s	185000
Link Frequency in GT/s	10
Average Cable Length in Meter	2
Link Width in Bit	32
Internal Data Width in Bit	192
Network Diameter in Hops	61
Packets per Cable	
Seconds per Meter	0.000000005
Bits per Meter	54.05
Packet per Meter	0.28
Packet per Cable	0.56
Packet per Link	18.02
Pipeline Stages per Node	
Pipeline Stages per Request Transmission	5
Pipeline Stages per Hop	10
Half Read Time in Stages	50
Pipeline Stages per Response Reception	5
Packets in all Nodes	
Packets per Pipeline Stage	1
Needed Pipeline Stages	670
Packets in all Nodes	670
Overall Packets one Direction	
Packets in all Cables	1099.10
Packets in Cables plus Nodes	1769.10
Complete Round-Trip	**3538.20**
Overall Packets	**3539**
Number of Entries in Buffer	3539
Lower Power of Two Bound	2048
Upper Power of Two Bound	4096
Lower Tag Size	11
Upper Tag Size	12
Address Size	64
Lower Buffer Size in KB with 16 Bit for Tag	**20**
Upper Buffer Size in KB with 16 Bit for Tag	**40**

Table 4-2: Tag Matching Buffer Example Calculation

In the first section of table 4-2 the assumed system is described with the physical conditions. With that information the number of packets which fit on one cable can be calculated. This can be seen in section two. Afterwards, the number of hops which are relevant

for one node are given in section three. Those are accumulated to the number of packets in all nodes in section four. Then all packets of node and cable are accumulated in section five. They are afterwards doubled for the round trip in section six.

In table 4-2 the required tag buffer space for the packets for one round-trip is calculated. The estimation of outstanding packets results in a number of 3092 tags. If a lower boundary would be chosen buffer space for 2048 entries would be sufficient. For an upper bound 4096 are necessary. This results in a tag length of 11 or 12 bits for the tag field. As additional information is needed to write the data back the tag matching buffer needs to carry more data to process a response. At least the size for an address must be stored. For further calculations an entry size of 64 bits address plus 16 bits of configuration information is assumed. In combination this results in a required buffer space of 20 KB for 2048 entries or 40 KB for 4096 entries. Those sizes seem to be feasible from a hardware perspective.

The numbers calculated in this example are relatively pessimistic regardless the assumption that requests and responses are not blocked. It is unlikely that one device continuously sends reads to the farthest distant device without any other tag consuming traffic from the requesting node. Also the responses will need more time as they occupy more space in the channel back than the requests in channel to the consumer. In case that the diameter is not the maximum of the allowed node count or the link bandwidth is reduced the tag size can also be reduced.

4.6 Virtual Channels

ULP provides four different types of virtual channels. The basic VC is mandatory to enable packet transmission. The quality of service (QOS) VC is optional and is used to give packets a priority. The register file contains a 16 bits priority map which has a hardware specific reset value but can be overwritten by software. A zero in the register stands for the basic VC whereas a one stands for the QOS VC. Thus, in worst case each VC is only enabled every 15th cycle if both channels are utilized. The ISOC VC is also optional and has the highest priority of all VCs. If it contains a packet it has to be processed regardless

of the other VCs. This can cause starvation at the other VCs which must be taken into account if this VC is used. The last VC is the adaptive VC. If this VC is used a device should provide multiple paths from one destination to a source. Packets traveling in this VC do not provide any guarantees for order relations among each other, which must be taken into account for this VC. Which route is chosen and how ordering is handled at the receiver is outside of the scope of the specification.

Every VC, besides the adaptive VC, has three sub VCs which are used for ordering among different packet types within the VC. Those VCs are called posted, non-posted, and response. Inside the posted channel packets travel which do not require a response. The non-posted channel is used for packets which need a response to complete a request. Inside the response channel, responses for previously sent non-posted requests return. To ensure the correctness the same ordering scheme as in HT is used.

4.7 Packet Format

4.7.1 Request and Response Headers

For packet based data exchange a header is needed which contains information about the packet. In ULP all data of the packet which does not belong to the payload is called header. Some fields are essential for the header format. Those are type, destination, source, tag, and address. The type field is needed to identify what kind of packet has been received. The destination field defines where the destination of the packet is. In order to be able to reply to a packet it is necessary that a header contains the source of a packet. For error handling reasons the source can also be interesting for packets which normally do not get a response but it can be reported which device caused the erroneous packet. The tag is essential as otherwise a response to one packet can't be merged to the request as the responses are not guaranteed to return in a specific order. The address field is needed to map a request to the corresponding node and to access the correct address region at the receiver.

In order to make the protocol more flexible and more efficient a length field should be used to describe the packet. The size of the header is fixed but the length of the data is otherwise unknown and therefore a fixed data volume for a header type had to be assumed.

From the software side it is desired that every type of packet size is allowed to be transmitted but from the protocol side it is more efficient to have a fixed granularity to keep the hardware effort low as described in chapter 2.1.3.2. If a fixed word width is chosen a mechanism is needed to perform smaller or misaligned accesses. For ULP a data granularity of 128 bits respectively 16 bytes is chosen. A byte mask is a common approach to handle those types of accesses but the information needed to handle them also has to be transmitted in the header.

Packets might be corrupted during transmission. In order to be able to identify those packets protection bits have to be added. A common way to handle this is by adding a CRC which is also done in ULP. For more information about the CRC please refer to chapter 2.4. All this information can be combined to an abstract header design which is shown in figure 4-7.

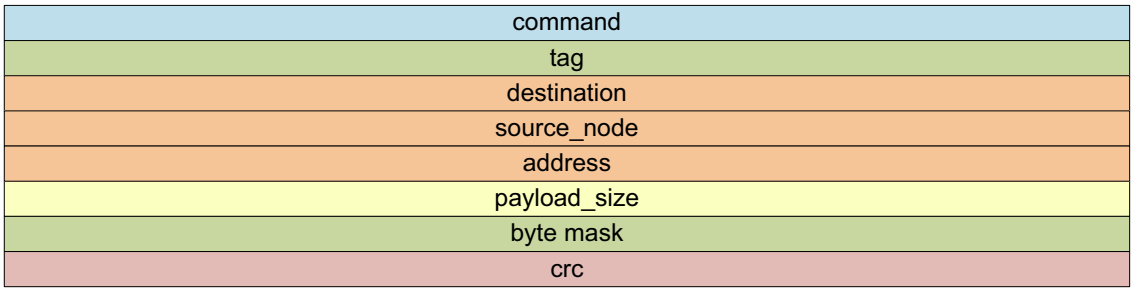


Figure 4-7: General Header Format

In order to get a more specific header format the width of the different header entries has to be defined. The type field must at least contain the information what kind of packet is transmitted. Thus, it must be as large as the number of available packet types. In ULP the commands listed in table 4-3 are provided.

Code[3:0]	VChan	Command	Comments/Options	Packet Type
0000	-	IDLE	Link IDLE, must be dropped at receiver	Info
0001	-	Credit	Carries credit information for the different virtual channels	Info
0010	NP	Write	Write request with response	Req/Addr/Data
0011	NP	Config Write	Configuration write request with response	Req/Addr/Data
0100	NP	Read	Read request with response	Req/Addr
0101	NP	Config Read	Configuration read request with response	Req/Addr
0110	R	Write Response	Response for write request	Resp
0111	R	Config Write Response	Response for configuration write request	Resp
1000	R	Read Response	Response for read request	Resp/Data
1001	R	Config Read Response	Response for configuration read request	Resp/Data
1010	P	Write	Write request without response	Req/Addr/Data
1011	P	Broadcast	Broadcast Message	Req/Data
1100	P	Fence	Fence, ensures posted request sent before are received	Req
1101	-	Error	Error Handling Packet	Info
1110	NP	Atomic	Atomic Request Compare and Swap	Req/Addr/Data
1111	NP	Flush	Flush, flushes posted request and returns response	Req

Table 4-3: ULP Type Field Encoding

For those 16 packet types a coding of 4 bits would be sufficient. In order to be able to add types the field is extended by 2 bits, as long as they are 0 the standard encoding is used. Bit 4 it is reserved for further encoding and bit 5 is used for vendor specific packets which can be used only among devices which provide the corresponding capability. If the protocol provides special functionality, like for example different virtual channels, they also belong logically to the type field of the header. ULP provides four different virtual channels and therefore 2 bits are used in the header. Additionally 3 bits are used to mark if the packet is allowed to pass posted packets (PP), the address has to be translated from virtual to physical (AT), and if an error occurred during transmission of the packet. The corresponding fields are shown in figure 4-8.

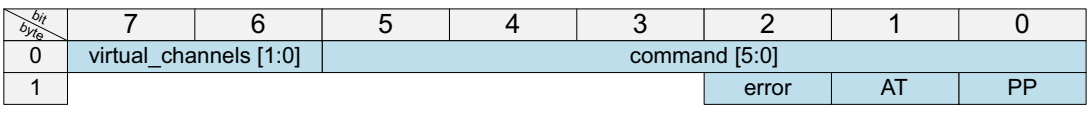


Figure 4-8: ULP Header Type Definition

For ULP a fixed maximum tag size is used as it has to be transferred inside the header. A tag has to be used for non-posted requests to be able to match the responses. The tag size is dependent on the diameter of the network. The example of chapter 4.5.3 shows that a tag number of 12 bits seems to be a suitable choice for a network size of 64 k nodes. However, for further extension a 13 bits tag field has been chosen. The tag field location is shown in figure 4-9.

bit / byte	7	6	5	4	3	2	1	0
1	tag [4:0]							
2	tag [12:5]							

Figure 4-9: ULP Header Tag Field

For packet routing source and destination IDs are needed. One feature ULP has to provide is to be able to establish large networks. Thus, large ID numbers are needed. Inside one node 4 bits wide node IDs are used. This enables up to 16 devices inside a node. If more devices are needed a connection which is marked as cable connected is needed to establish a second node. Among nodes network IDs are used. As 64 k are supported a 16 bits wide network ID is used. The combined network and node ID fields are shown in figure 4-10.

bit / byte	7	6	5	4	3	2	1	0
3	source_node_id [3:0]				destination_node_id [3:0]			
4	destination_network_id [7:0]							
5	destination_network_id [15:8]							
6	source_network_id [7:0]							
7	source_network_id [15:8]							

Figure 4-10: ULP Header ID Fields

In order to handle data transfers which are misaligned to the data granularity mask fields have to be used. In ULP it is not allowed to transmit data which is not continuous. Non-continuous data has to be transferred with multiple packets. The mask field for one OW consists of 4 bits. Every bit represents one sub granularity of 16 bytes. Bit 3 stands for 8 bytes, bit 2 for 4 bytes, bit 1 for 2 bytes, and bit 0 for 1 byte. The coding is shown in figure 4-11. With this coding every combination of valid bytes can be indicated. All bits set to 0 stands for a complete valid packet which means the data is aligned. Two double-words are needed as the data may start misaligned and end misaligned. Thus, ULP provides a start byte mask and an end byte mask. If the data is separated in only one misaligned data word only the start mask is used and bits [3:0] of the address show where the data starts.

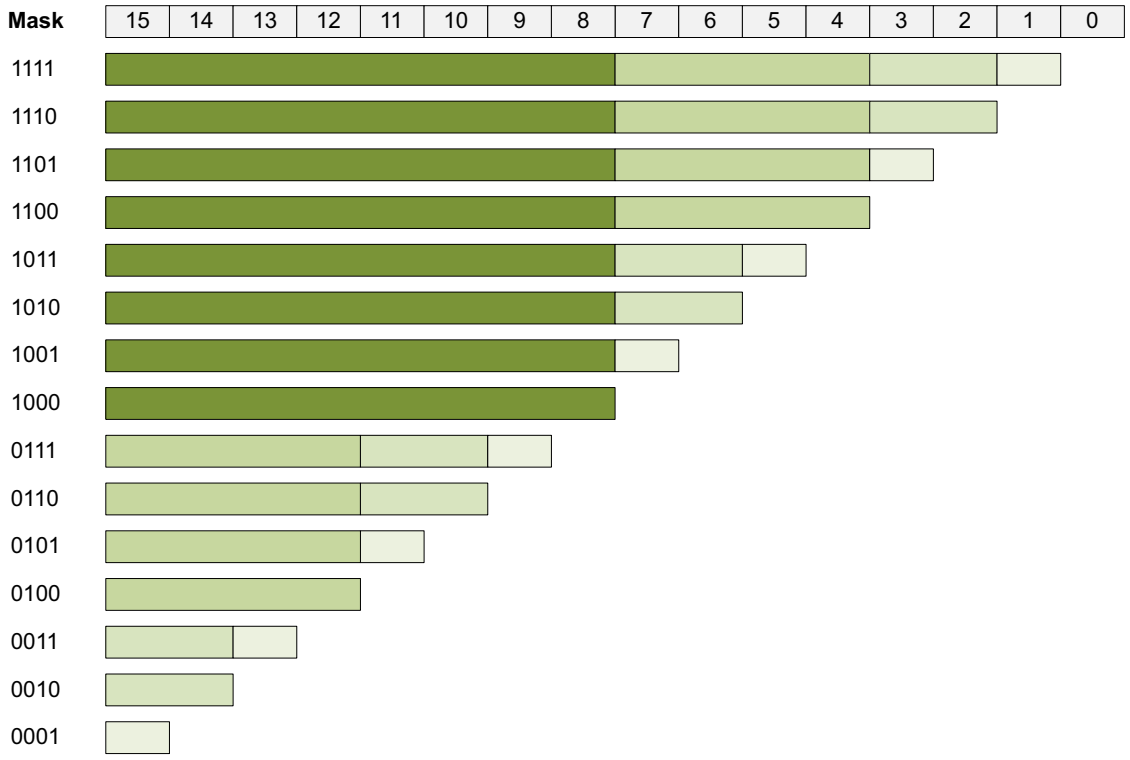


Figure 4-11: ULP Mask Field Coding

A more flexible masking scheme would provide more bandwidth for non-continuous data. However, for efficient data exchange transmitting such small packets is always suboptimal. The header overhead for one packet is by at least 50%. From a bandwidth perspective a suitable data transfer granularity would be cache entries, as at this granularity processing units are able to exchange data very efficiently without masking or multiple reads or writes. The mask fields are shown in figure 4-12.

bit / byte	7	6	5	4	3	2	1	0
8	end_byte_mask [3:0]				start_byte_mask [3:0]			

Figure 4-12: ULP Header Mask Fields

The payload size field is needed to be able to transmit different payload sizes with one type of packet. In ULP the granularity of the payload is always one OW. A maximum transfer unit of 4 KB can be realized in ULP. 4 KB in 16 bytes granularity result in a payload size field of 8 bits. The number of doublewords is the length field + 1, so all 0 represents one OW of data. Empty writes or read responses are not allowed. If a packet is transferred which type indicates that it contains no data the payload the size field must be ignored. The payload size field is shown in figure 4-13.

bit / byte	7	6	5	4	3	2	1	0
9	payload_size [7:0]							

Figure 4-13: ULP Header Payload Size Field

State of the art systems use address spaces of up to 64 bits. In order to be able to use this address space request packets use a 64 bits address field. The address is given in byte granularity even if the data granularity is OW aligned. Response packets do not needed addresses as the corresponding address is stored in the receiver's tag map. Thus, responses simply do not provide an address field. The address field is shown in figure 4-14.

bit / byte	7	6	5	4	3	2	1	0
12	address [7:0]							
13	address [15:8]							
14	address [23:16]							
15	address [31:24]							
16	address [39:32]							
17	address [47:40]							
18	address [55:48]							
19	address [63:56]							

Figure 4-14: ULP Header Address Field

At the end every header contains a CRC which protects the packet and the data of the previous packet, if it contained data. It must be able to protect the size of an MTU and a header. For this reason a 32 bits CRC was chosen. The CRC field is shown in figure 4-15. For more information about the CRC please refer to chapter 4.12.

bit / byte	7	6	5	4	3	2	1	0
20	crc [7:0]							
21	crc [15:8]							
22	crc [23:16]							
23	crc [31:24]							

Figure 4-15: ULP Header CRC Field

In combination the above introduced fields can be merged to form the corresponding headers. One type is used for request packets and one is used for response packets. The structure of those packets is shown in figure 4-16 and figure 4-17. Request packets are always 24 and response packets 16 bytes long. The size of 24 bytes for requests was chosen as a tradeoff between loss of bandwidth and granularity. As the main granularity for ULP is 16 bytes the misalignment causes a shift of the header through the data width. Thus, there are two positions where the header can start. If this would have not been done an-

other header size would have to be chosen. With 16 bytes the header would be undersized to realize all needed features for ULP. In combination with a header size of 32 bytes the overhead of the header would be 25% larger for every packet and must be filled with useful information. As the 24 bytes header already contains 2 vendor specific bytes this had to be avoided.

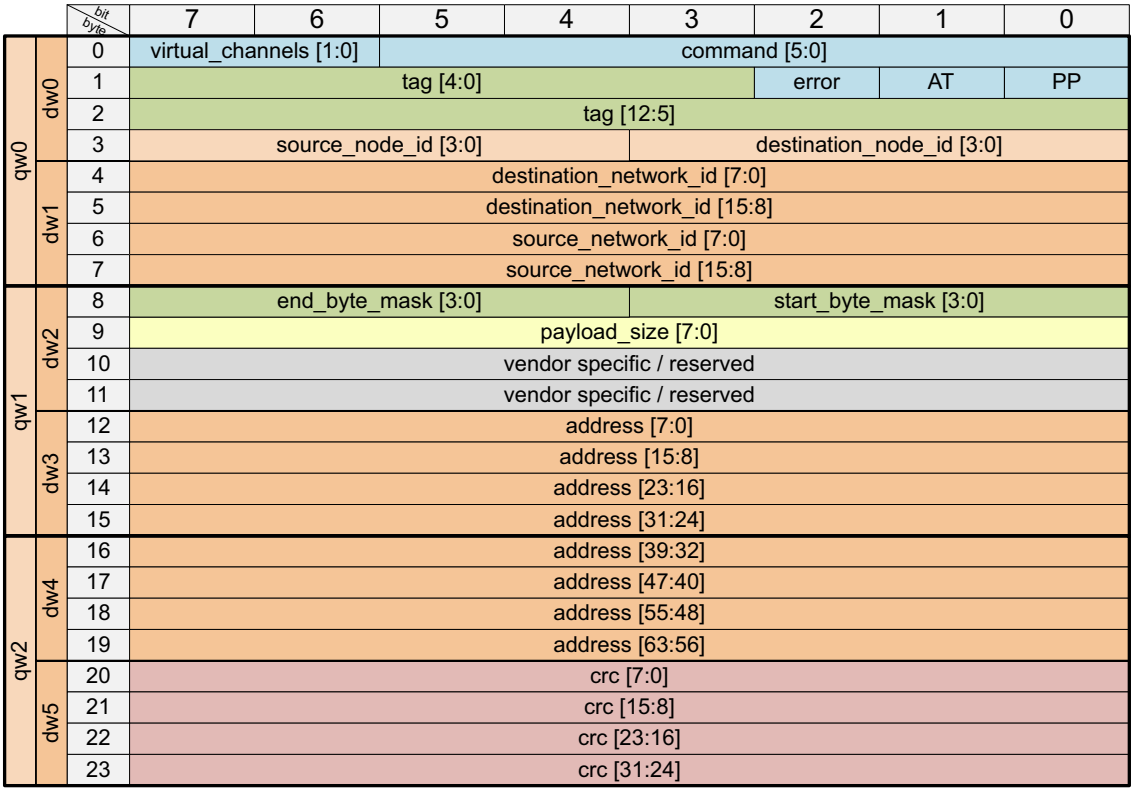


Figure 4-16: ULP Request Header Format

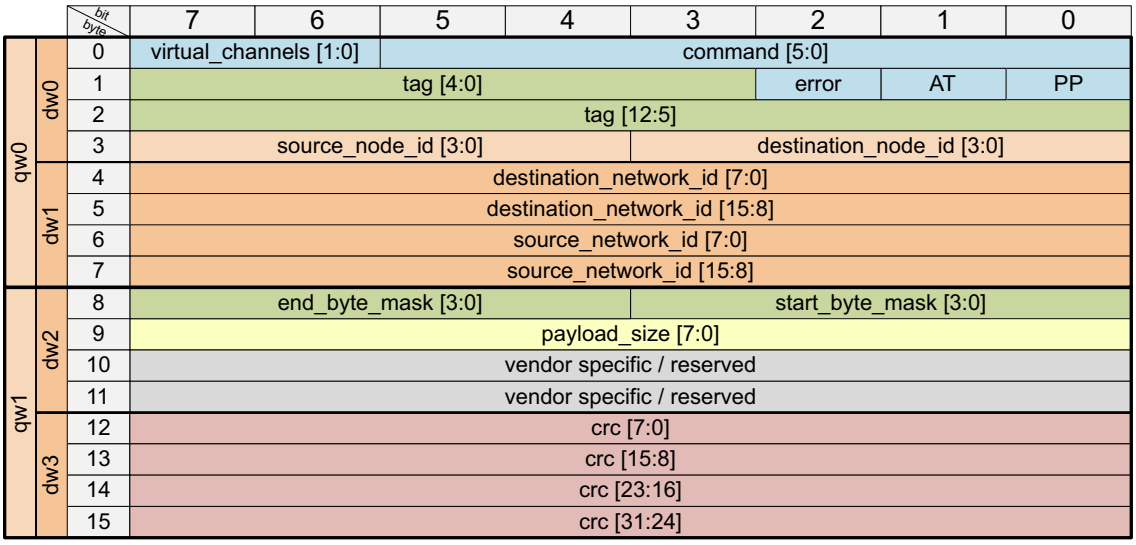


Figure 4-17: ULP Response Header Format

The vendor specific bytes can be used to add features to the devices which are not required by ULP. It must be assured that those features are only used by devices which know that the link partner is also capable of those features. In case modified packets are sent to a receiver which does not provide the feature the best case would be that it is ignored, in the worst case this causes an uncorrectable error and the link needs a reinitialization.

4.7.2 Info Packets

Info packets are only used between direct link partners to exchange information and are never forwarded indirectly connected nodes. They are always 16 bytes long. Three different types of info packets exit in ULP: idle, credit, and error handling packets. Idle packets are transmitted on a link if no other packets are available. Besides the type field and the CRC they contain a bit pattern which ensures transitions on each lane if transmitted, but they are also scrambled. The idle packet is shown in figure 4-18.

		bit / byte	7	6	5	4	3	2	1	0
qw0	dw0	0	10		IDLE==000000					
		1	1100_1001							
		2	1111_1100							
		3	1111_0000							
	dw1	4	0000_1111							
		5	1111_0000							
		6	0011_0000							
		7	1001_1100							
qw1	dw2	8	1111_1100							
		9	1111_0000							
		10	0000_1111							
		11	1100_0011							
	dw3	12	crc [7:0]							
		13	crc [15:8]							
		14	crc [23:16]							
		15	crc [31:24]							

Figure 4-18: ULP Idle Header Format

Credit packets are used to handle the flow control of a link. They contain the number of released credits for a virtual channel. The VC field determines the corresponding virtual channel which the credit belongs to. The three value groups (VG) describe the sub-type of VC buffer the credits belong to. VG0 stands for posted, VG1 for non-posted, and VG2 for response credits. Credits are accumulated until the credits are released and one credit stands for one entry in the buffer. A credit field is 11 bits wide and can therefore release

the complete buffer at once. For additional protection the credit header provides a 16 bits sequence number. The last correct received sequence number is stored at the receiver and transmitted with an error packet if a retransmission is requested. With the sequence number an additional error can be checked. In case a credit packet is lost undetected from the CRC check one or more sequence numbers are missing. As they are stored in the retransmission buffer it would be retransmitted in the retry sequence. The structure of the packet is shown in figure 4-19.

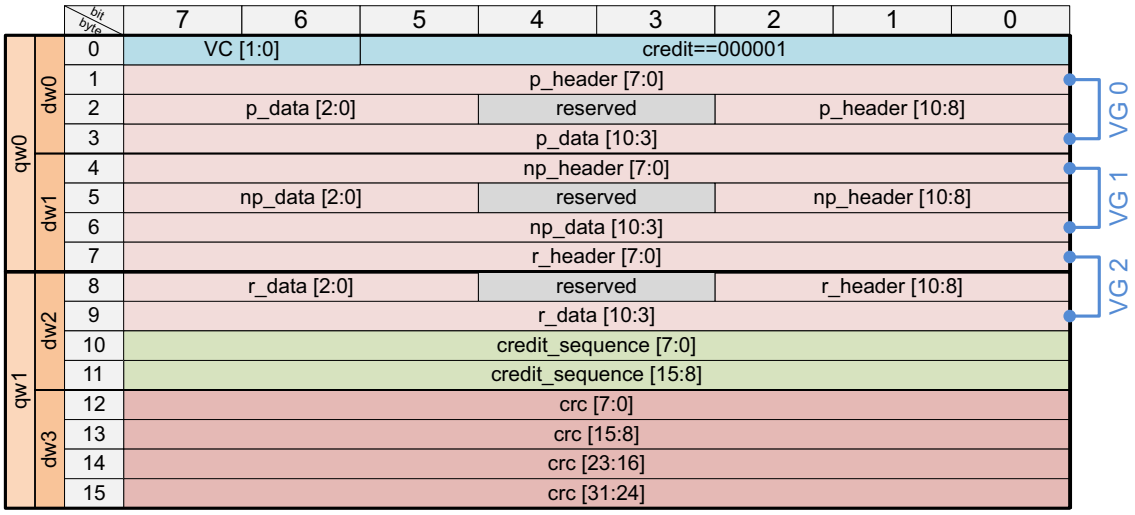


Figure 4-19: ULP Credit Header Format

Error handling packets are used to handle the retry sequence. Two different types of error handling packets are available. One is the error start packet and the other one is the error end packet. The two different packets are distinguished by the error type field. A 01_2 stands for an error start packet and a 10_2 for an error end packet. An error start packet contains the amount of checked data which has not been released by a credit jet. The number cannot be larger than the amount of all virtual channels combined and therefore 16 bits are sufficient. The error header also provides a sequence number which enables the receiver to detect missing error packets but for error handling packets the sequence number is optional. The error handling header is shown in figure 4-20.

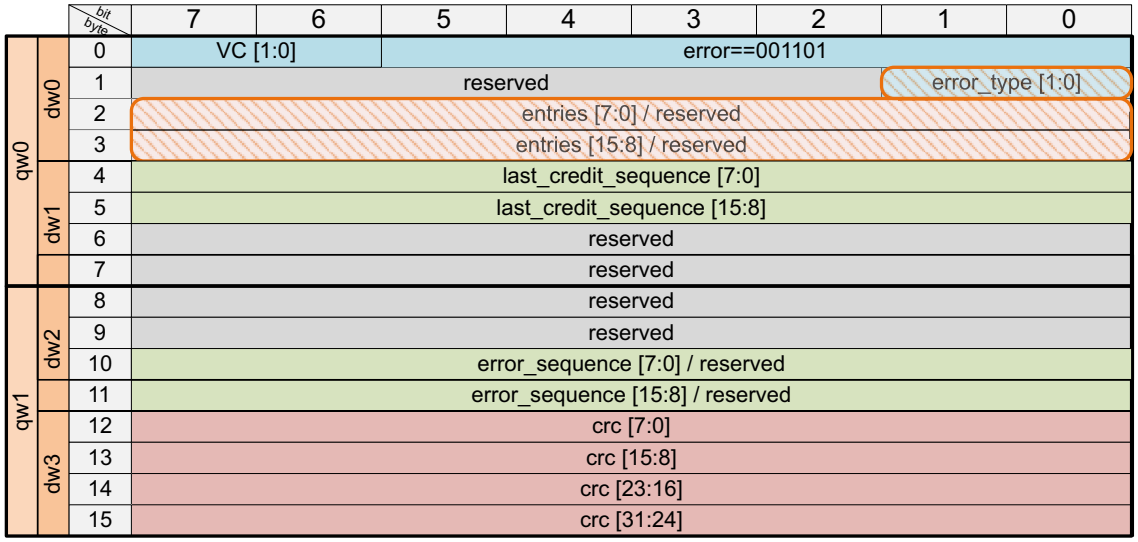


Figure 4-20: ULP Error Handling Header Format

4.8 Decoding Dependencies

ULP has a structure which can be efficiently mapped to hardware. One big part to achieve this was to reduce the different types of dependencies. Control frame dependencies have been minimized as almost the complete packet is always completely defined by the type field. Only error headers have a 2 bits sub-type field which distinguishes the packet further. However, the structure of the error header is so simple that this difference has almost no impact to the complexity of the hardware.

An additional step to minimize complexity was to have a more coarse grain granularity of the protocol. Two granularity levels have been chosen. The granularity of the protocol was set to 128 bits. This means that there are no parts of the protocol, like header and data, which are allowed to be smaller than 128 bits. Therefore, only one packet can be finished at a time if the internal data width is 128 bits or below. Additional to the protocol granularity a separate header granularity of 64 bits is used in ULP. Header always consists of chunks of 64 bits but are at least 128 bits long. Thus, there are no doubleword dependencies like in PCIe and HT but only quadword dependencies. Those two granularities have been chosen to have the needed flexibility for the header size and the coarse grain granularity for the relaxed hardware effort.

In figure 4-21 the quadword dependencies of ULP and the different header types are depicted. It can be seen that the clean structure of ULP provides almost no hardware overhead. The layer height for quadwords is 9 which is significantly lower than for HT and PCIe.

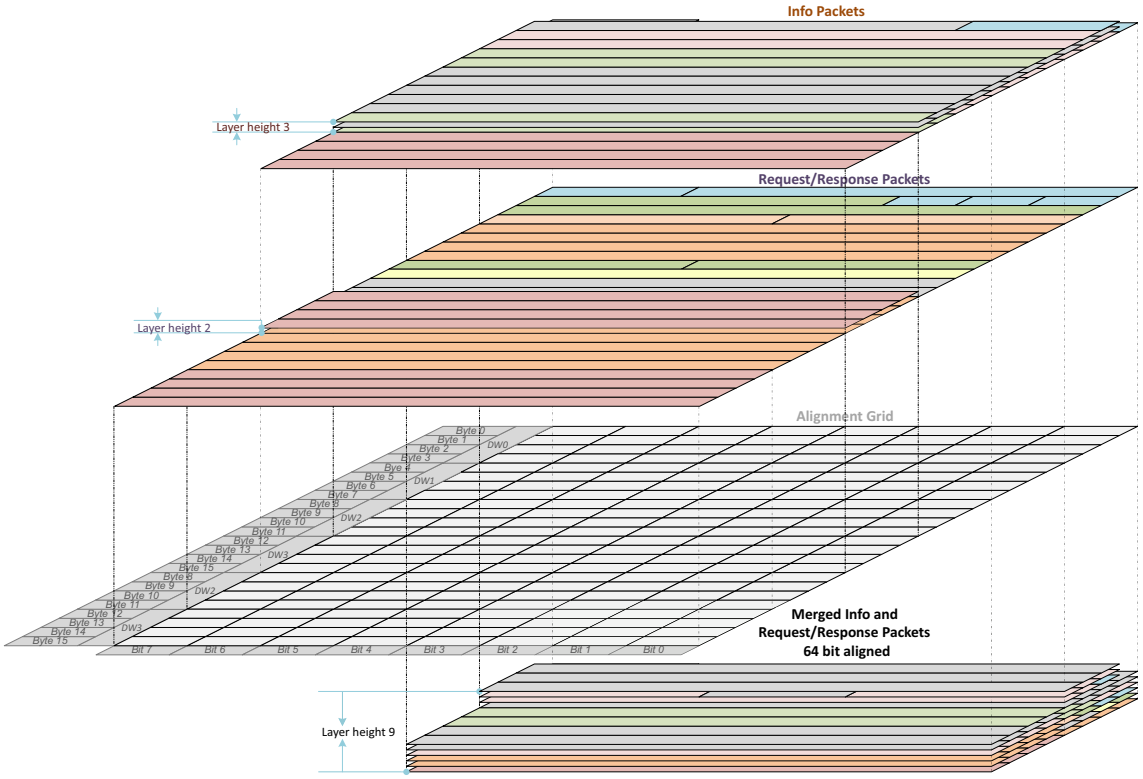


Figure 4-21: ULP Quadword Protocol Dependencies

4.9 Initialization Sequence

Before a link can be used it has to be initialized. Therefor a sequence has to be performed which allows the usage of the link afterwards. This sequence is depicted in figure 4-22. Different stages have to be completed until the link can be used to transfer packets. First the corresponding device needs to be powered and has to wait until its clocks are ramped up and everything is ready. This state is represented by the reset state which is left to lane detect afterwards. During lane detect special bit pattern are transmitted and tried to be received. More details to the corresponding sequences are given in chapter 4.9.1. If the link is not marked as connected and no cable is connected to the link the FSM goes to uncon-

nected. In this state the receivers and transmitters are turned off. The state can be left to lane detect if a cable is connected and detected. If the link is marked as connected or a cable is connected in the lane detect state, but no initialization sequence was received, the link goes to no link found. In this state the transmitters are turned off but the receivers are still active to be able to receive a training sequence. If a correct pattern is received the state returns to lane detect. Receiving correct patterns in lane detect the state is changed to lane training. In this state training sequences are exchanged to find the correct byte, block, and partly lane alignment. In case alignment cannot be found the state is switched to parameter change. Otherwise, it is changed to exchange. In exchange a sequence is used to exchange link information and block numbers. If all patterns were received correctly and the link runs at maximum bandwidth the state is changed to operational and the link is fully initialized and ready to exchange packets.

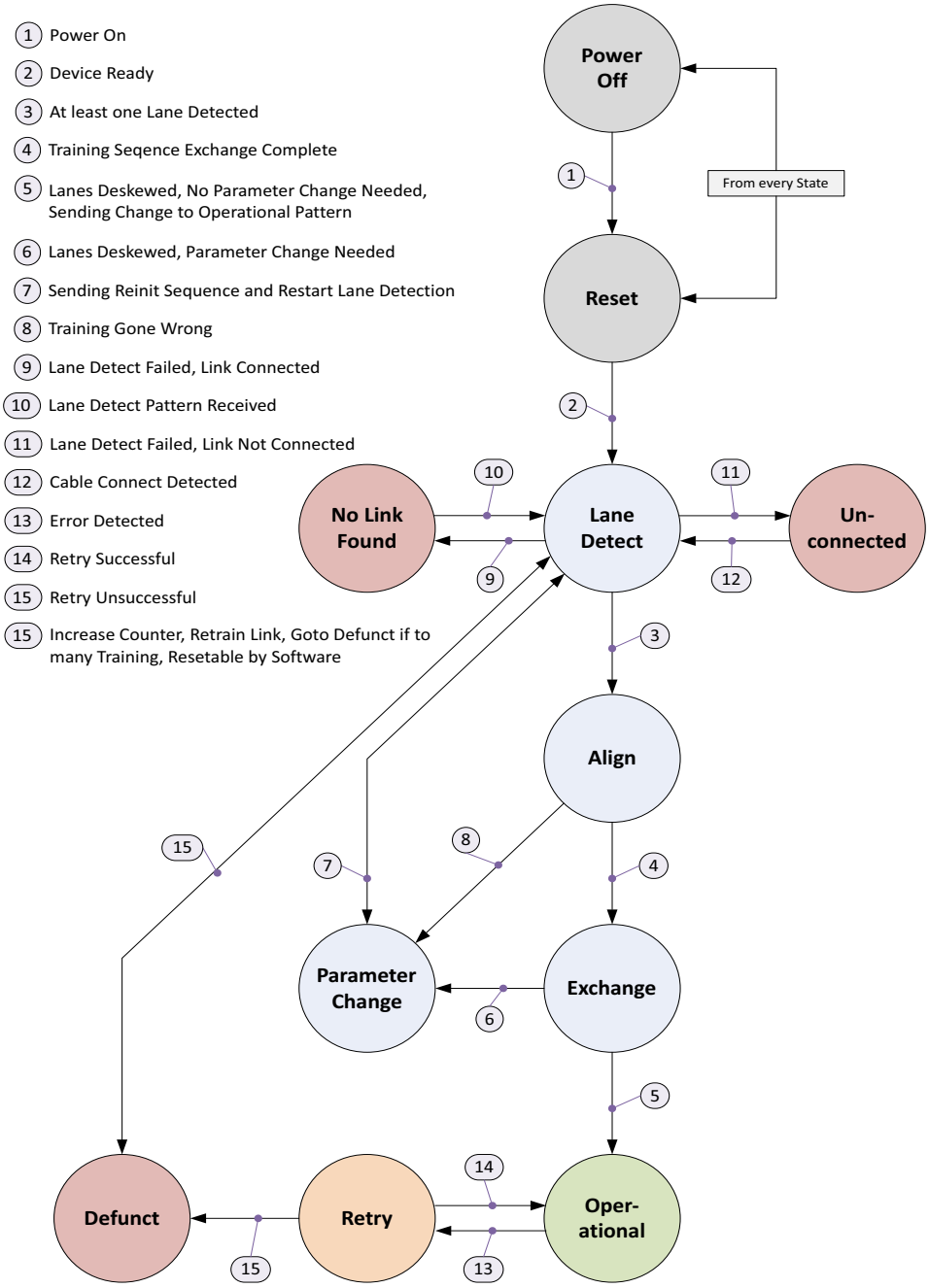


Figure 4-22: ULP Initialization Sequence State Machine

4.9.1 Training Sequences

For link initialization different training sequences (TS) are used. They are not scrambled and not 128b/130b coded as the rest of the traffic. TSs are not spread over the links but transmitted on every lane separately. All TS are built in the same way. The first byte contains an identifier which is unique for every TS. The odd bytes are used to DC balance the

traffic so every 16 bits chunk has a neutral DC balance. The structure of the five TSs is shown in figure 4-23. If at least 16 consecutive blocks of a sequence are received the TS is marked as successful for the corresponding lane and it is attempted to find the next TS.

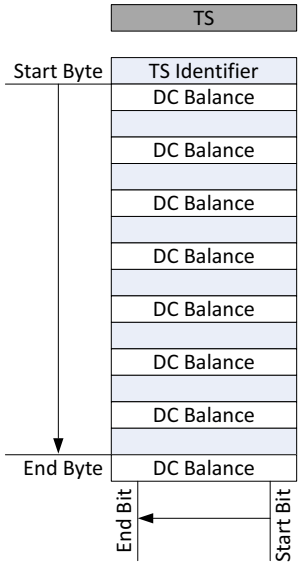


Figure 4-23: ULP Training Sequence Structure

At the first try TS have a maximum length of 10 ms. In case the correct pattern of the corresponding TS is received from the link neighbor the TS has to persist at least 1 ms afterwards. Then it can be changed to the next TS. In case the link setup was not successful and has to be restarted, the timing for minimum and maximum will be doubled until the maximum number of initialization retries have been reached. A timing example of a TS change is given in figure 4-24.

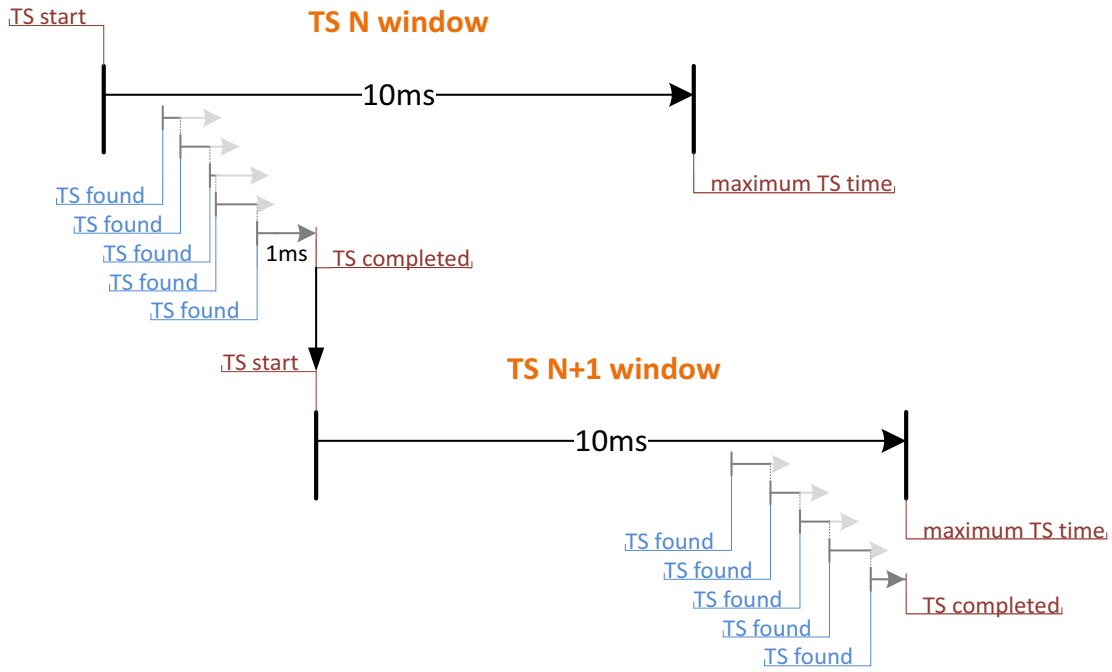


Figure 4-24: ULP Training Sequence Timing Example

Every TS is used to manage a specific task. The different TSs are shown in figure 4-25. TS0 is transmitted in the lane detect state and is used for CDR. Thus, simply a clock pattern is transmitted. If at least one lane receives the correct pattern for one block a timer starts which is responsible for the 1 ms window. During this window other lanes can receive the correct pattern and reset the timer. If no more lanes are found for 1ms or the time limit is met the state is changed to lane training.

TS0	
Binary	HEX
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA
1010_1010	hAA

CDR

TS1	
Binary	HEX
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00
1111_1111	hFF
0000_0000	h00

Byte Align

TS2	
Binary	HEX
1111_1110	hFE
1111_1111	hFF
0000_0001	h01
0000_0000	h00
0101_0101	h55
1010_1010	hAA
1100_1100	hCC
0011_0011	h33
1111_0000	hF0
0000_1111	h0F
1110_1110	hEE
1000_0010	h82
1100_1111	hCF
0001_1000	h18
1111_0000	hF0
1111_0000	hF0

Block Align

TS3	
Binary	HEX
0000_1111	h0F
1111_0000	hF0
Type / BN	h--
DC Balance	h--
Link Width	h--
DC Balance	h--
Frequency	h--
DC Balance	h--
Parameters	h--
DC Balance	h--
Parameters	h--
DC Balance	h--
Parameters	h--
DC Balance	h--
1111_0000	hF0
DC Balance	h0F

Parameter Exchange

TS4a	
Binary	HEX
0101_0101	h55
0101_0101	h55
0011_0011	h33
0011_0011	h33
0000_1111	h0F
0000_1111	h0F
0000_0000	h00
1111_1111	hFF
1111_1111	hFF
0000_0000	h00
0000_1111	h0F
0000_1111	h0F
0011_0011	h33
0011_0011	h33
0101_0101	h55
0101_0101	h55

End of Training

TS4b	
Binary	HEX
0101_0101	h55
0101_0101	h55
0011_0011	h33
0011_0011	h33
0000_1111	h0F
0000_1111	h0F
1111_1111	hFF
0000_0000	h00
0000_0000	h00
1111_1111	hFF
0000_1111	h0F
0000_1111	h0F
0011_0011	h33
0011_0011	h33
0101_0101	h55
0101_0101	h55

Parameter Change

Figure 4-25: ULP Training Sequences

In the lane training state the different lanes first transmit TS1. This is done to ease the alignment process. Blocks of bytes with all bits set to 0 or 1 are alternating sent. With this sequence the receiver can search for the change from 0 to 1, or vice versa, and find the byte alignment. The change to TS2 takes place in the same way as from TS0 to TS1 with the 1ms timer and the maximum timing window. Detecting additional lanes at this point is allowed. The initialization process is always continued for all lanes. In TS2 more complex patterns are exchanged. With those patterns the block can be detected and the reliability of the lanes can be checked as the pattern is prescribed. After block alignment the lanes can be block aligned to each other, but it is not sure if at every lane the same block time is received.

After TS2 the state is changed to exchange and the sequence is changed to TS3. It contains fields which are not fixed by the protocol. Thus, the corresponding odd bytes must be determined after the even bytes are known. The inverted value of the even bytes has to be used for the odd bytes to enable a check for the correct transmission. In TS3 information is exchanged between the link partners. The first non-identifier byte contains the type of device and the block number (BN). With the type field the link partner is informed of special capabilities like for example support of coherent operation. The BN is used to deskew whole blocks as it might be that for lane N block time X contains block number M and for lane N+1 block time X contains block number M-1. The lane deskew is not allowed to be more than one block size. The calculation is shown below.

$$\text{Bit per cm Transmission Lane: } \frac{10\,\text{Gb/s}}{200000\,\text{km/s}} = 0.05\,\text{b/cm}$$

$$\text{Deskew Distance: } 0.05\,\text{bit/cm} \cdot 128\,\text{bit} = 6.4\,\text{cm}$$

If a link speed of 10 GB/s is assumed and a propagation speed of 200000 km/s one bit occupies 0.05 cm of the transmission lane. For 128 bits this results in a distance of 6.4 cm. If one block is shifted in positive direction and one is shifted in negative direction a distance of two blocks could result. Thus, the allowed distance is simply cut in half which is 3.2 cm.

Further information is the supported link width and link frequencies. For the link width every combination of valid link widths is allowed. For the frequencies at least the minimum frequency has to be supported as this is the initialization frequency. All other frequencies are allowed to be enabled in any combination. Additional parameters can be exchanged to give the transmitter hints which electrical settings are expected. The evaluation of those parameters is optional and it could reduce the hardware effort. Using the additional parameters could improve the initialization speed. The ULP field descriptions of TS3 are shown in figure 4-26.

type	byte	description
identifier:	byte 0	b1111_0000
type and block number:	byte 2	type[5:0]: b00_0000 = coherent CPU b00_0010 = coherent GPU b00_0001 = non-coherent CPU b00_0011 = non-coherent GPU b00_0101 = non-coherent node b00_1001 = non-coherent network BN[7:6]: b00 = block time 0 b00 = block time 1 b00 = block time 2 b00 = block time 3
link width:	byte 4	b????_???1 = 1 lane b????_??1? = 2 lanes b????_?1?? = 4 lanes b????_1??? = 8 lanes b???1_???? = 16 lanes b??1?_???? = 32 lanes
frequency:	byte 6	b????_???1 = 2.5 GT/s b????_??1? = 5 GT/s b????_?1?? = 10 GT/s
electrical parameters:	byte 8-12	depending of the electrical specification and therefore out of the scope of this thesis
end token:	byte 14	b1111_0000

Figure 4-26: ULP TS3 Field Description

At the end TS4 is transmitted. It is used to signal that the training sequence has finished. There are two different TS4s, TS4a and TS4b. They are used to distinguish between the next states. In case of TS4a it signals that the maximum possible link bandwidth has been reached. The following state is the operational state and the normal packet transfer starts, as depicted in figure 4-27. Otherwise, if the information exchanged in the TS3 states that a higher link bandwidth can be achieved. In this case, TS4b is used and the state is changed to parameter change. In this state the link frequency and electrical parameters of the transceivers are changed and the link starts a new initialization sequence with the new parameters.

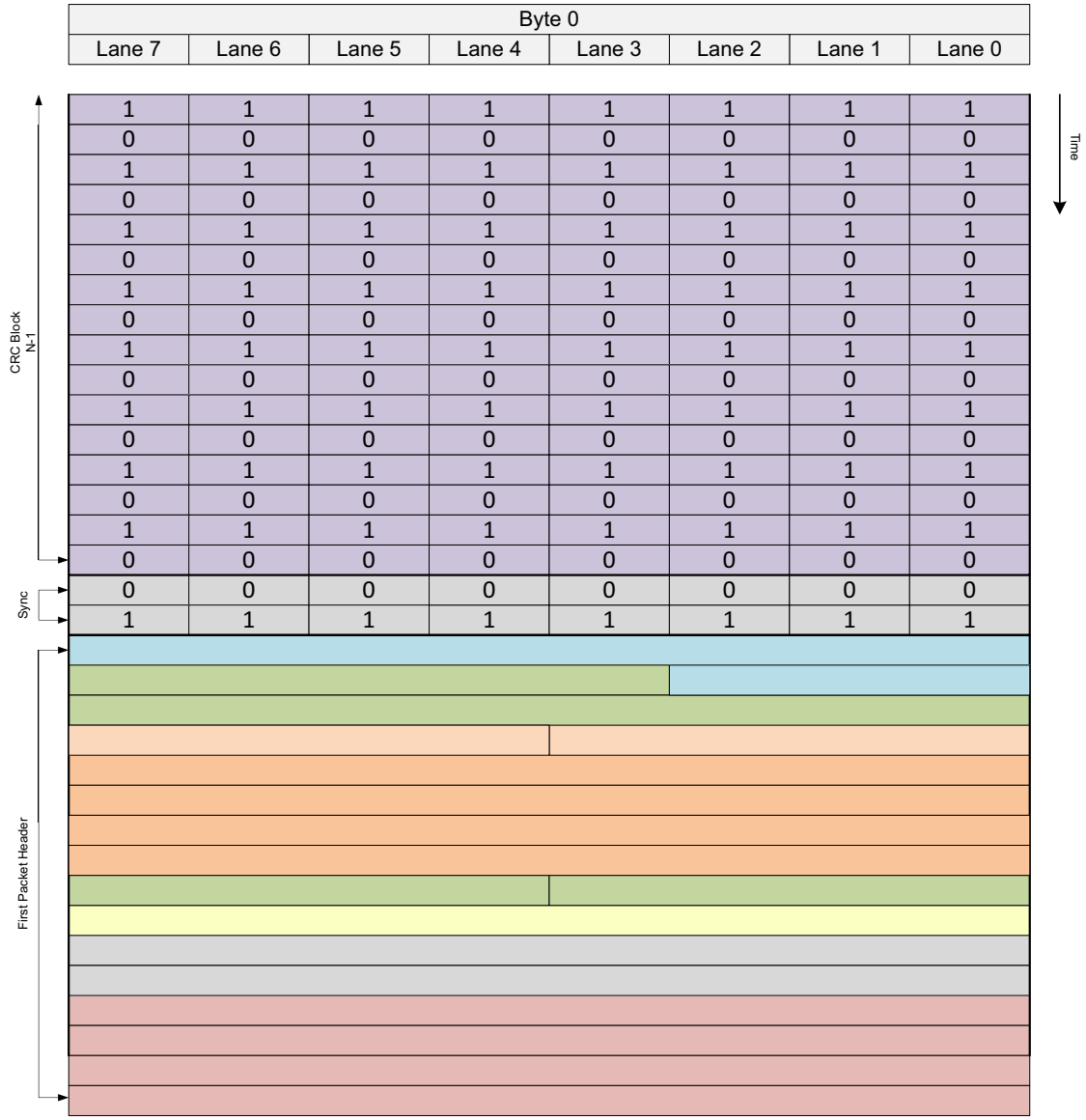


Figure 4-27: Switch from TS4a to Packet Transfer Example

If a working link is available and the maximum allowed reinitializations are exploited the link is set to operational regardless if the exchanged parameters promise a higher link bandwidth. A bit in the register file is set to signal that not the theoretical maximum bandwidth is achieved. This bit can be read by software and the further proceeding is outside of the scope of this thesis.

4.10 Routing Scheme

Network devices must provide an addressing scheme to be able to route packets. There exist many different approaches which solve this problem like in [30] and [65]. The 4 bits node IDs are used if the current ID of the node is the same as the destination node ID. Otherwise, the 16 bits network IDs are used to route the packets among nodes.

4.11 Bandwidth Calculation

The achievable bandwidth of ULP can be calculated by subtracting the bandwidth required for line coding, clock compensation, and header overhead. For the header overhead a maximum size payload is assumed. ULP provides 4096 bytes of data and a header has the size of 24 bytes.

$$\text{Bandwidth without Header Overhead: } 100 \cdot \frac{4096}{(4096 + 24)} = 99.41\ \%$$

For line coding every 130 bits 2 bits are used.

$$\text{Bandwidth without Line Coding Overhead: } 100 \cdot \frac{128}{(128 + 2)} = 98.46\ \%$$

For clock compensation the same scheme as for PCIe is chosen which specifies an insertion every 370 to 375 blocks of 8 bytes per lane.

$$\text{Bandwidth without Clock Compensation Overhead: } 100 \cdot \frac{6000}{(6000 + 8)} = 99.86\ \%$$

Resulting in an achievable maximum bandwidth of 97.74 % of the theoretical maximum.

$$\text{Accumulated Bandwith: } 99.51\ \% \cdot 98.46\ \% \cdot 99.86\ \% = 97.74\ \%$$

4.12 Error Handling

For the error handling different checks can be performed in ULP. One mandatory check is the check of the CRC transmitted with every header. This CRC enables the receiver of

the data to evaluate if the eventually previously received data and the actual header is correct. An example of the CRC window is shown in figure 4-28.

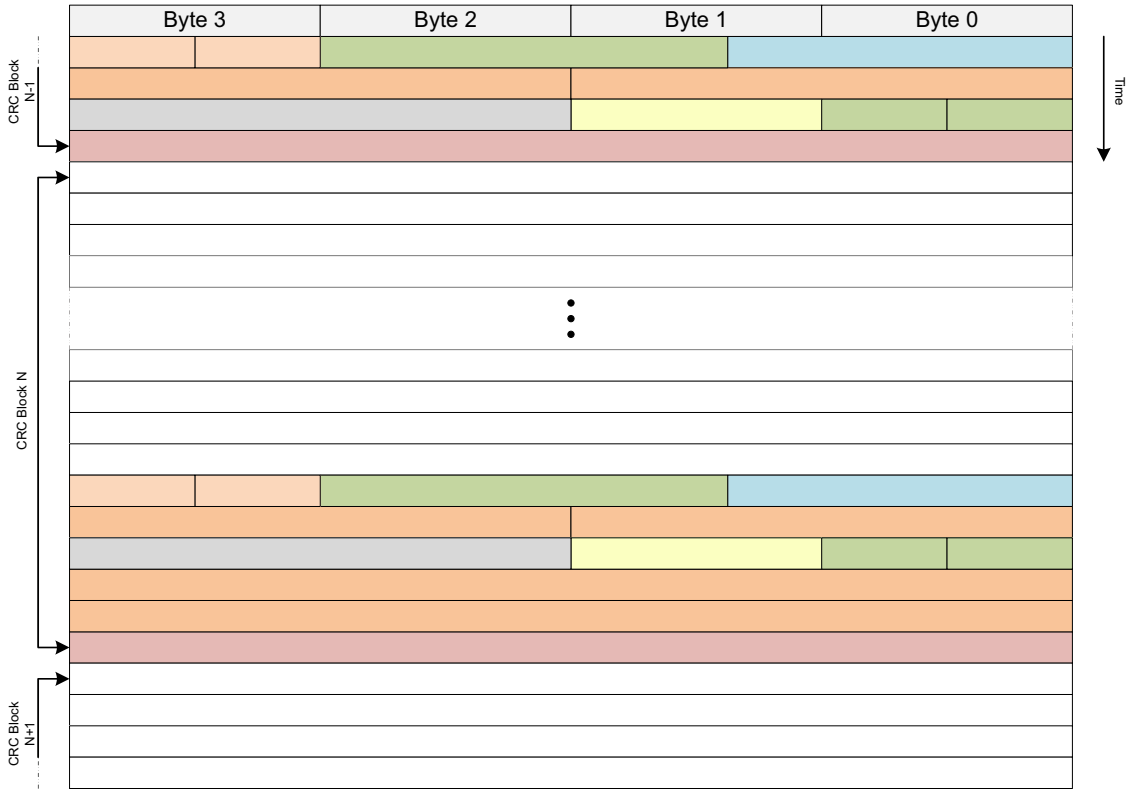


Figure 4-28: ULP CRC Window Example

In order to get a relative high protection the Koopman polynomial is used. As stated in [46] this polynomial provides a hamming distance of 5 for message sizes from 8 - 65505 bits which is suitable for ULP. With a hamming distance of five all arbitrary errors of up to 4 bits can be found. The polynomial representation is shown below:

$$\text{Koopman Polynomial: } X^{31} + X^{20} + X^{15} + X^{10} + 1$$

It is assumed that the CRC check is sufficient to cover the possible errors during runtime. If more protection is favored additional checks can be made which should also result in a retransmission. Some of those checks are listed below.

- Type field invalid
- Header format not allowed

- Packet too short
- Unknown response
- Credit sequence missing

Additional checks which result in an uncorrectable error can be performed like a wrong sync pattern.

4.12.1 Retransmission Handling

As soon as one of the above mentioned correctable errors is detected it is assumed that one or multiple bit errors occurred during transmission. In order to be able to recover automatically from such an error a retransmission protocol is included in ULP. Therefor a retransmission buffer is needed. The buffer is built as a ring buffer. It must be ensured that only invalid values will be overwritten. If the size of a VC buffer is chosen as suggested in chapter 4.5.1, the handling is more demanding compared to a full blown buffer. In case of the VC buffer size it is taken advantage of the fact that traffic in the VC buffers is already checked and must not be retransmitted. The problem is that there is traffic which is checked but not released by credits from the buffer. In order to find the correct starting point the amount of correct data stored between the point where the packets are checked and the VC buffer must be monitored. In case an error occurs the number of buffer entries and the sequence number of the last correctly received credit packet must be transferred with the error message that triggers the retransmission. In combination with the pointers of the retransmission buffer and the number of the buffer entries, the corresponding read pointer can be set to the correct value. The VC buffers cannot be managed as plain FIFOs. An entry of the buffer is only valid if header and data are checked. Furthermore, there must be separate pointers for written and checked data. In case the last written data is not valid the write pointer must be set to the next entry after the checked data pointer. This has also to be taken into account for the calculation of the valid data value for the pointer setting of the retransmission buffer.

For the correct behavior of the retransmission buffer it is important to find the correct starting point. Therefore, the retransmission buffer must be able to set the read pointer in-

dependently and it can be overtaken by the write pointer. At the point when the error packet is received the read pointer is adjusted to the correct value in combination with the checked data value. This can always be done if additional virtual pointers are used to manage the overall possible VC buffer space. Thus, the distance between the actual read pointer and the write pointer can be calculated and transferred to the physical retransmission buffer. The buffer handling is shown in figure 4-29. In case all VC buffers are of the same size the calculation can be reduced to the bits corresponding to one VC buffer and not over all VC buffers and the virtual pointers can be avoided.

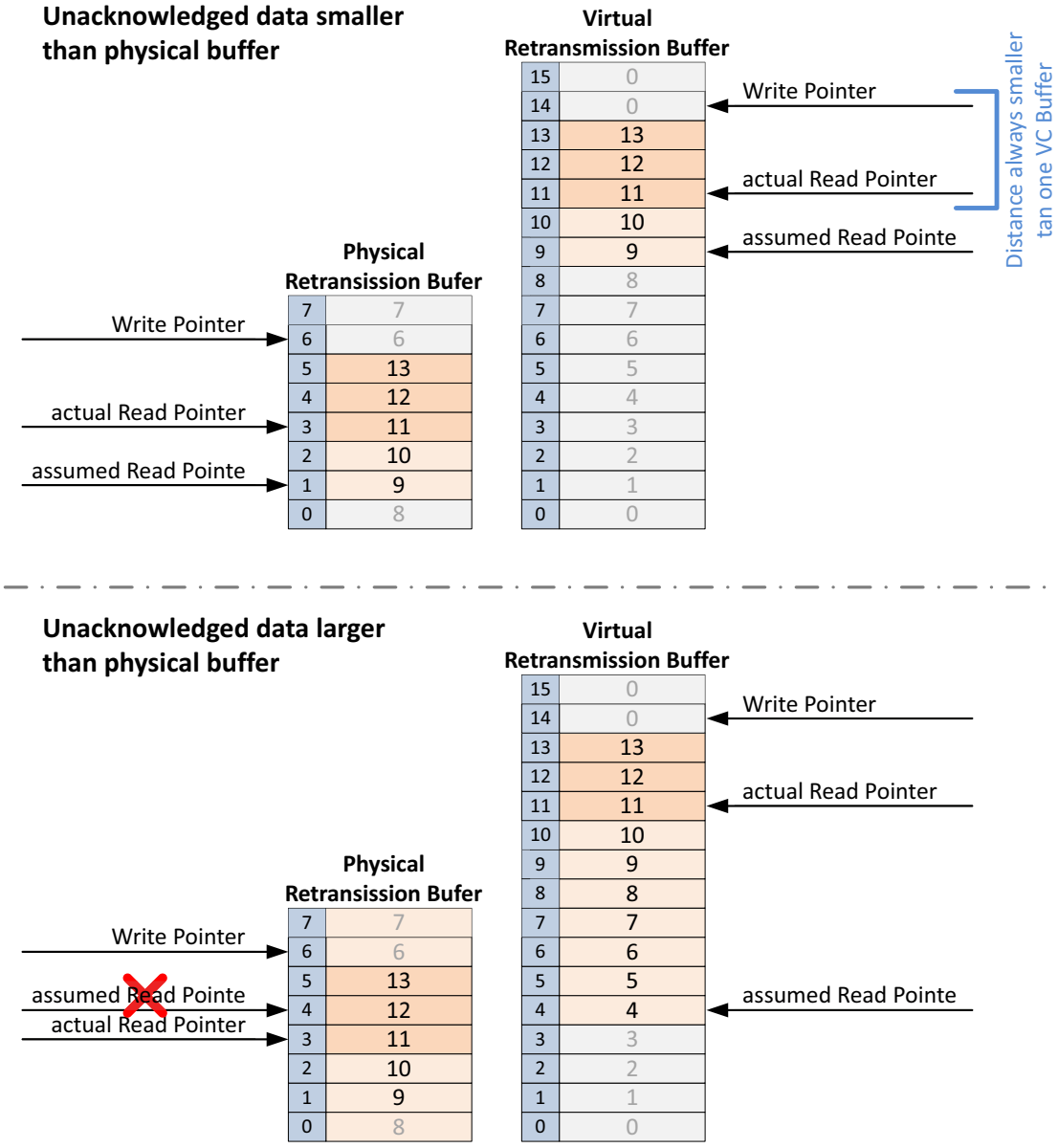


Figure 4-29: Retransmission Buffer Read Pointer Setting

It must be assured that the checked data value is correct even if the error occurs right at the end of a packet streamed out of the VC buffer. Thus, the checked data value is not changed if the buffer space is released but when the credit packet is transmitted before the error packet. Error packets have to be inserted into the data stream as soon as possible to reduce bandwidth loss.

For the reliability it is of importance that neither credits nor error packets are lost during transmission as a loss does not only lead to a reduced performance of the link but it would be impossible to recover an error as the retransmission starts from the wrong pointer value. Thus, credit packets have to be inserted into a separate retransmission buffer and must be retransmitted if an error occurred. The correct starting point is set with the last credit sequence number. In order to reduce the buffer size for credits credit insertion is restricted. A normal maximum sized buffer would have 24 KB for credits which are 1536 entries. For every time the allowed credit insertion is halved the buffer size is also halved. As it makes no sense to release every small portion of credit with the calculated buffer sizes it is no problem. Thus, only one credit every 16th slot is allowed. This results in a buffer with 96 entries to hide the round trip latency. As the sequence number is 16 bits it can be safely determined which credit packets have to be released. For the retransmission the outstanding credits are released first and then the retransmission buffer is replayed.

Error packets need a watchdog timer which ensures that they are repeated if they are lost. A suitable timer would be larger as the time needed to fill one VC buffer. If an error packet is received the current transmission is stopped immediately and the read pointers of the buffers are set to the correct value. The first packet transmitted is a retry packet which is an error packet with the sub type field set to 10_2. Afterwards the credit buffer is replayed the retransmission buffer follows.

In case the watchdog timer runs out eight times the link is marked as defunct as it is unlikely that the same error packet is erroneous for that many times without a permanent defect, like a lost bit time of the link or a broken/unplugged cable. The start of the read pointer at retransmission is stored for the case that the retransmission failed and has to be repeated. In case the retransmission fails four times successively a permanent defect of

the link is assumed and the link is set to defunct.

The receiver detecting an error drops all traffic received afterwards and also stops transmitting normal traffic. This has to be done as the retransmission buffer is significant smaller as the available VC-buffer space and could be overwritten otherwise. Only error packets or retransmission traffic are allowed to be sent until the end of the error sequence is received. This is done to avoid a retransmission buffer overflow in case both sides of a link have an error at the same time and the error packet was not received.

4.12.2 Link Retraining

With the retransmission capability correctable errors of the link can be fixed. If for some reason the retransmission fails a permanent defect of the link is assumed. In this case the buffers and registers are reset and the link initialization sequence is repeated to train the link. It is attempted to set up a new link to keep the system functional. If this does not work the link is set to the defunct state and it is waited until the software resets the error counters and starts a new initialization sequence. In case the device cannot be reached by software anymore a hardware reset has to be performed.

4.13 Protocol Comparison

ULP provides all needed functionality a protocol needs. In order to prove the performance it has to be compared to the corresponding protocols. In the following sub-chapters ULP will be compared to HT and PCIe in terms of bandwidth, frequency, number of devices, network diameter, link distance, and decoding effort.

4.13.1 Bandwidth

To be able to compare the bandwidth of the different protocols it is necessary to consider different types of bandwidth. The pure accumulated link bandwidth can lead to a wrong impression. One bandwidth type is the physical lane bandwidth which is the bandwidth when all data lanes are accumulated. The next bandwidth is the physically available band-

width which includes all high speed lanes regardless if they are used for data or other purpose. In order to make the different protocols more comparable the bandwidth differences due to physical characteristics like link frequency and link width must be eliminated as it must be assumed those are more or less only definitions and the different link frequencies and link widths could also be used by other protocols. If this is done it is the neutral bandwidth. Nevertheless, the most interesting bandwidth is the achievable bandwidth. This is the bandwidth if all overhead is subtracted. In this comparison it is calculated as the neutral bandwidth without the overhead for better comparability. In table 4-4 those bandwidths of the different protocols are opposed.

	HT	PCIe	ULP
Physical Lane Bandwidth	25.6 GB/s	32 GB/s	40 GB/s
Physical Available Bandwidth	38.4 GB/s	32 GB/s	40 GB/s
Neutral Bandwidth with 10GT/s	60 GB/s	40 GB/s	40 GB/s
Achievable Neutral Bandwidth	33.42 GB/s	39.11 GB/s	39.11 GB/s

Table 4-4: Bandwidth Comparison HT/PCIe/ULP

An additional bandwidth is of interest for comparison which is the bandwidth for different data payloads. In figure 4-30 it is shown how the bandwidth is utilized while the payload size is doubled. In this case only the per packet overhead is calculated for packets which use 64 bits addressing.

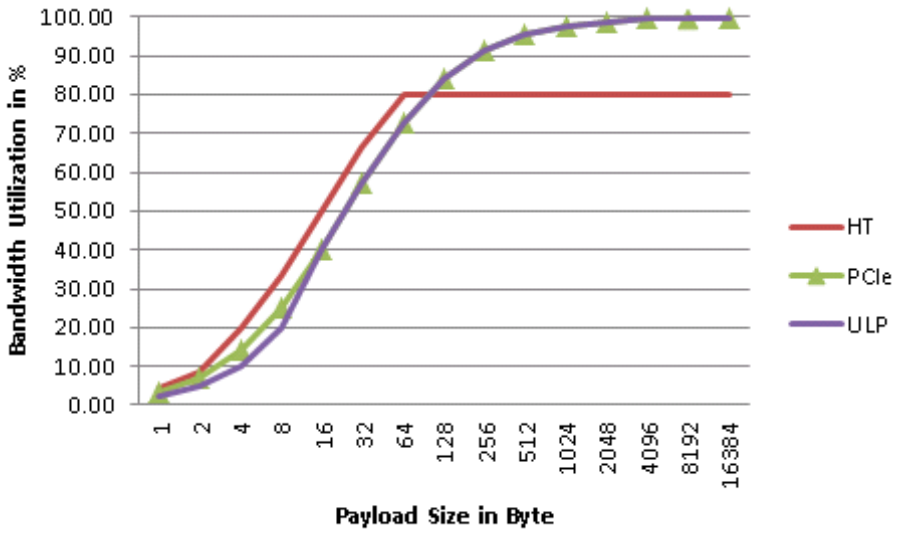


Figure 4-30: Packet Bandwidth Comparison for increasing Packet Sizes

It can be seen that for small payloads HT has minimal advantages because of the lower

header overhead and the utilization of the payload. In this case also PCIe is slightly better than ULP as the data granularity of 16 bytes results in more loss of bandwidth for the payload. If payload sizes are 16 bytes or larger ULP has the same performance as PCIe for payloads which fit in the 16 bytes data granularity. If the maximum payload size of HT is reached its packet bandwidth saturates at 80% and is then overtaken by PCIe and ULP which both saturate at 99.42% at a payload size of 4 KB.

In the bandwidth comparison ULP follows best industry praxis. However, it is not the goal of ULP to be better performing in every point but to be at least competitive. It is assumed that for efficient data transfer the payload should be at least one cache line as those are data chunks which are moved by processing units. Typical cache line sizes are 32, 64 or 128 bytes.

Also the theoretical maximum values provided by the protocols have been used to calculate those numbers. For example, to the best of my knowledge HT devices only use 16 bits wide links and therefore devices do not provide the full theoretically defined bandwidth of the protocol.

4.13.2 Frequency

One important factor for the comparison of in terms of frequency is the maximum link bandwidth which can be achieved. For the comparison the link bandwidth is calculated and the resulting internal frequency is determined in table 4-5 with regards to the internal data width. The important values are bold and the assumed internal link width is highlighted orange.

	HT	PCIe	ULP
Link Width in Lanes	32	32	32
Link Frequency in Gbit	6.4	8	10
Link Bandwidth in Gbyte	25.6	32	40
Internal Width in Bit	**Internal Frequency in GHz**		
32	**6.4**	**8**	10
64	**3.2**	4	5
128	1.6	2	**2.5**
256	0.8	**1**	1.25

Table 4-5: Internal Link Frequencies for Different Data Widths

The granularity of the protocol is important at this point. As HT has 32 bits granularity the only internal link width with no additional effort with regard to fan-out, dependencies, and multiple packet reception in one clock cycle is 32 bits. However, the internal frequency at this width is unrealistic high so that larger link width should be taken into account to reduce the internal frequency. At Gen3 frequencies HT needs a header and a CRC which results in at least two doublewords, so a 64 bits link width can be used without the disadvantage of receiving multiple packets in one clock cycle, but the doublewords will start to shift through the internal link width and therefore increase the fan-out for every doubleword. Also the dependencies in and among packets will rise. The frequency at 64 bits is still ambitious, but increasing the data width further will lead to a large hardware complexity.

In PCIe symbols are used at every lane so those 8 bits have to be gathered from each lane before a packet can be assembled. Thus it is assumed that a suitable data width is 256 bits for 32 bits wide links. However, at this granularity it is possible that read and write request are not able to utilize the link bandwidth as it is only allowed to start a header in one symbol time. For example a read request needs 5-6 doublewords but the internal width is 8 doublewords. Same applies for writes with headers of 5-6 doublewords and small payload sizes of 1-2 doublewords. Additionally a link width of 256 bits has a high dependency among the doublewords and high multiplexing effort.

ULP does not use a 32 bits granularity but a 128 bits granularity. With this feature it is not reasonable to use an internal link width smaller than 128 bits. As ULP allows header formats of 192 bits the start of a header can shift but only with two fixed positions. An internal frequency of 2.5 GHz is feasible, but doubling the link width to 256 is still possible. An additional header start point would have to be managed and two packets can be received in one clock cycle. But the impact is significantly less than for a 32 bits granularity with four more starting points and multiple more packets which can be received in one clock cycle.

4.13.3 Number of Devices

The number of devices can be determined by checking how many devices can be addressed in the system. All protocols provide some identification fields for this task.

HT provides the unit ID to address a device in the fabric. The field is 5 bits wide and therefore only 32 devices addressable. It is obvious that this is insufficient for large systems. In [14] high node count is mentioned which seems to be an attempt to enlarge the node count. On the HT-Consortium homepage two papers can be found [60][61] which provide little details about it. However, there is no detailed specification available and therefore it can't be compared.

PCIe uses a bus-device-function (BDF) model to address unique entities. Eight bits are used to address a bus so 256 buses are available. A device on a bus is determined by a 5 bits field. Thus, every bus can be connected by up to 32 devices. Each device can contain functions which are distinguished by a 3 bits field which results in 8 functions. Therefore, the BDF field does not exactly address devices but functions. However, the bit-vector is large enough to theoretically address 64 K devices.

ULP provides two different IDs, one is the 4 bits wide node ID and the other one is 16 bits wide the network ID. Thus, 64 K nodes can be connected and every node can contain 16 devices. In combination 20 bits can be used to address a device which results in 1 M devices. The distance for node devices is assumed to be very small so the influence to tags should be insignificant.

4.13.4 Network Configuration

4.13.4.1 Network Diameter

For the network diameter three parameters of a protocol are of importance. The most important is the number of devices which can be addressed in the system. However, it is negligent to use only this number for the calculation and to ignore the others. Of course it is possible to build a large network with only the addressable devices, but it is also important if it can be utilized. The other two parameters, which are available tags and credits, deter-

mine if the network can be efficiently used by utilizing the links. For an end-to-end view the tags are important as they hide the end to end latency. In order to hide the round trip latency on a link the number of available credits are important.

The number of tags is relevant as if none is available no new non-posted requests can be transmitted and therefore the link will be idle. Thus, the calculation has to be made for minimum sized packets with responses, which is the case for read requests. It is assumed that an address range of 64 bits is used.

For HT the source tag is used to uniquely identify a request. It is 5 bits wide and therefore 32 outstanding minimum sized packets can exist. For 64 bits addresses an 8 bytes header is used in combination with a 4 bytes CRC. Thus an amount of 32 times 12 bytes can be transmitted with the available tags which are 384 bytes. With this amount of data 96 pipeline stages of 32 bits width can be filled.

In PCIe 8 bits tags are available resulting in 256 outstanding minimum sized packets. The header for a read is at least 20 bytes long and is followed by a 4 bytes CRC. Thus 256 packets with a size of 24 bytes can be transmitted with the available tags which is exactly 6 KB resulting in 192 pipeline stage entries of a 256 bits width.

For ULP a 12 bits of the tag field are used. This results in 4096 outstanding minimum sized packets. A header of 24 bytes is used which already includes the CRC. Thus, distance of 96 KB respectively 4 K pipeline stages with a width of 192 bits can be bridged with this.

For the same parameters as assumed in table 4-6 on page 156, table 4-6 shows the achievable network diameters of the different protocols without nonrecurring overhead, like memory access at the receiver.

	HT	PCIe	ULP
Network Diameter	~1	~3	~72

Table 4-6: Network Diameter Comparison

4.13.4.2 Link Distance

The number of credits per link is needed to calculate the maximum distance between two directly connected devices. For the calculation simply the maximum number of outstanding packets is taken. This number is multiplied with the minimal size of a request packet to get the number of outstanding bits. Afterwards the bits per lane for a link width of 32 lanes are calculated. Together with the corresponding link speed and the propagation delay of 185000 km/s the bridgeable link distance is calculated and cut in half for the round trip. For the calculation the time needed for processing is not taken into account as it is irrelevant for calculation.

$$\text{Available Bits per Lane} = \frac{(\text{Number of Packets} \times \text{Packet Size in Byte} \times 8)}{\text{Number of Lanes}}$$

$$\text{Bits per Meter} = \frac{\text{Lane Speed}}{\text{Propagation Speed}}$$

$$\text{Link Distance} = \frac{\text{Available Bits per Lane}}{\text{Bits per Meter}}$$

The HT specification makes no restrictions with respect to the number of available credits. However, in the BKDGs [62][63] there exist numbers for header credits for all virtual channels of about 32 - 48. At this point the most interesting request type are reads, as they are the smallest type an upper bound for one virtual channel can be assumed which is the number of available tags with 32. With 32 packets a distance of 1.39 meters can be bridged with a link width of 32 bits and a link frequency of 6.4 GT/s without losing performance.

For PCIe there are defined numbers for the maximum available credits. Credits are measured in doublewords and a maximum buffer space of 2560 bytes for headers which are 128 entries and 32768 bytes for data which are 2048 entries. Of course only the outstanding read requests are important as well, so PCIe is limited to 128 packets. A distance of 8.88 meters can be bridged with a link width of 32 bits and a frequency of 8 GT/s.

ULP credit information is exchanged with 11 bits. The resulting number of credits is 2048. As every header consumes one credit a maximum distance of 2 K read request packets of 24 bytes can be bridged. For the calculation it is assumed that the link width is 32 bits and

the frequency is 10 GT/s resulting in an achievable maximum distance of 133.66 meters which is very comfortable. The number of all protocols is brought together in table 4-7.

	HT	PCIe	ULP
Link Distance	1.39	8.88	133.66

Table 4-7: Link Distance Comparison

4.13.5 Decoding Effort

The comparison of the decoding effort is not simple as every protocol has its special features. However, there are some higher level aspects which are comparable. Those aspects are dependencies inside one header, the number of headers which can be received in one clock cycle with the dependencies among them, the granularities with the corresponding header shift and multiplexing effort, and the fan-out for a corresponding granularity.

4.13.5.1 Header Dependencies

Header dependencies originate from header fields which have to be analyzed before the structure of the header can be determined. HT has many dependencies as it has the a smallest header size and the dependencies are used to realize the functionality in such small space. However, this has to be compensated with additional hardware effort. PCIe has a more straight forward header format with more space for the functionality of the different headers. Thus, the dependencies inside one header type are low, but additionally to the large header size PCIe has reached this with different header types (TLP, DLLP, and token) and multiple layers. This also results in some additional hardware effort and increased latency. ULP has a very straight header format. The header is defined by the 6 bits of type. Only the error packets have two additional bits to distinguish among them. It can be seen that from the header complexity point of view ULP has a similar hardware effort as PCIe but without the disadvantages of multiple layers and different header types.

4.13.5.2 Multiple Header Reception

For multiple header reception the internal link width is the crucial factor. As soon as the link width is larger than the protocol granularity it can happen that more than one header is received at the same clock cycle. The problem with this is that the bandwidth has to be

sustained and therefore the decoding engines have to be replicated for the additional headers or it must be guaranteed that the decoding engine has time to catch up without losing bandwidth. In table 4-8 is shown how the link width influences the number of headers for the different protocols. Relevant are not only complete headers but also headers started in a previous clock cycle that finish later with other ones.

	Number of different possible Packets per Clock Cycle			
Link Width	32	64	128	256
HT	1	2	3	5
PCIe	1	2	2	2(3)
ULP	1	1	2	3

Table 4-8: Comparison Multiple Header Reception

It can be seen that for the protocols with a 32 bits granularity the number of multiple headers start at as soon as the data width contains multiple chunks of the granularity. From this point it is relevant what the minimum header size is. For HT it is 64 bits and therefore additional packets can be in one clock cycle for every additional 64 bits. As ULP has a header granularity of 64 bits two headers can be received at the earliest with a data width of 128 bits. PCIe and ULP have the same header size of 192 bits for one packet, thus additional full packets can only occur from a data width of 256. PCIe avoids this by restricting the number of header starts for one clock cycle which results in a loss of bandwidth if otherwise multiple packets would have been received. However two packets can be received in PCIe if TLPs and DLLPs are considered as in the same layer.

4.13.5.3 Granularity Effects

The internal data width with regard to the granularity effects the complexity of the hardware. The smallest type of granularity is of importance for the header shift. Especially two factors are influenced by the granularity, the header shift and the multiplexing complexity. In figure 4-31 examples for HT, PCIe, and ULP are given which show the dependency of the data width, header shift, and the header granularity. A potential header start is marked red but not the complete header is displayed. Small granularity results in many possible combinations for a header start whereas larger granularities relax it.

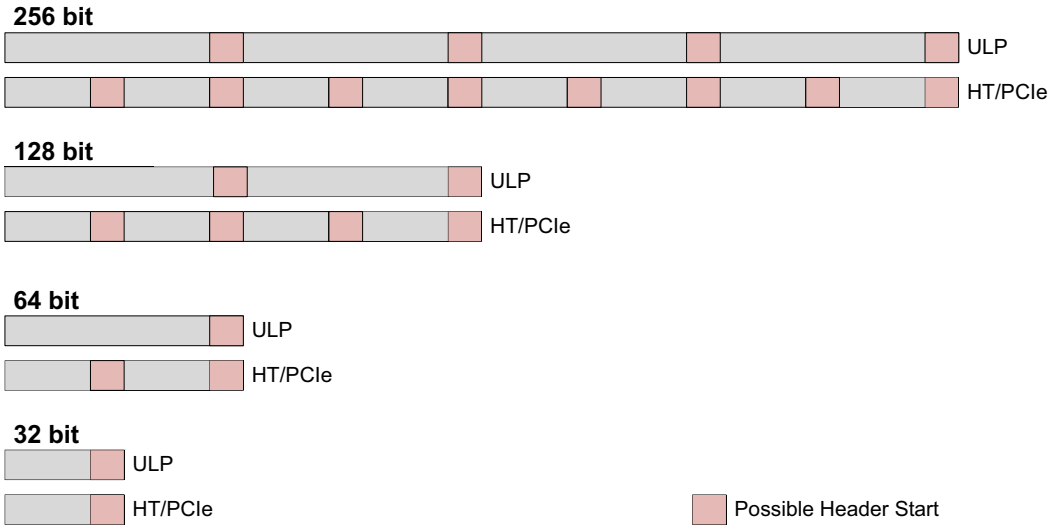


Figure 4-31: Header Shift for 32 and 64 Bits Granularity

Additionally to the header shift the multiplexing effort is influenced by the granularity of the protocol with regard to data width. In order to achieve a certain bandwidth the data stream has to be parallelized to the corresponding internal data width. The more the protocol granularity is smaller than the data width the larger gets the multiplexing complexity. In figure 4-32 ULP is compared to HT with a header data width of 192 bits and a data width of 128 bits for ULP, and a 128 bits header width and a 128 bits data width for HT. The effect of the smaller data granularity of HT can be clearly seen.

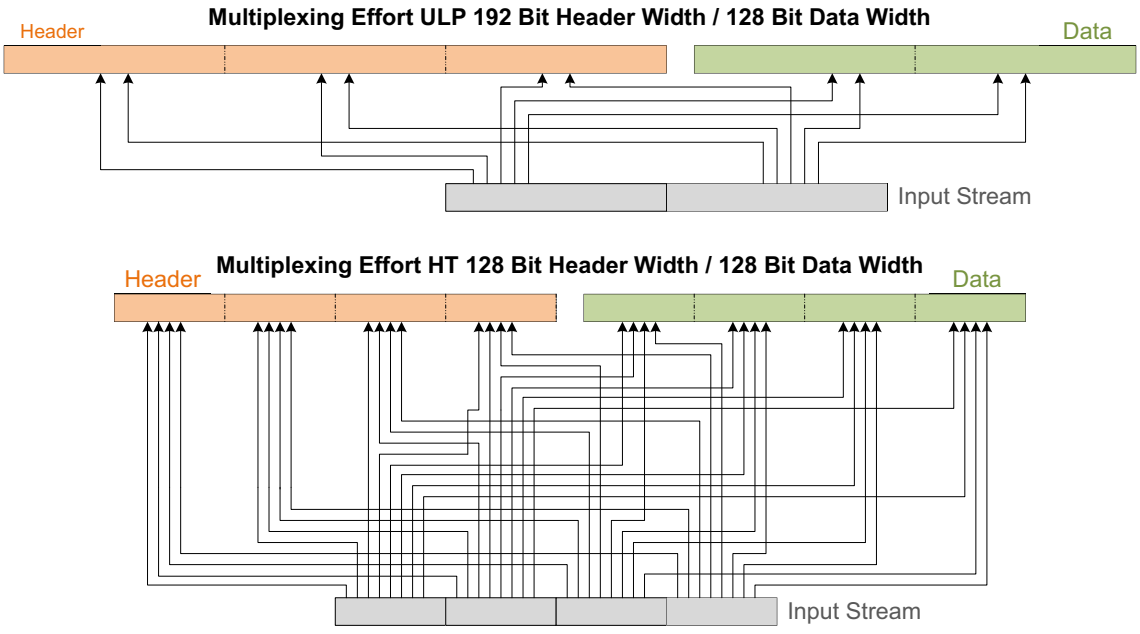


Figure 4-32: Comparison Multiplexing Complexity Doubleword - Quadword

4.13.5.4 Fan-Out

The fan-out of one doubleword influences also the hardware complexity. Before, the multiplexing complexity with regard to the granularity was shown. This is one part of the fan-out, but additionally the content of the doubleword has to be taken into account. In figure 4-32 above the multiplexing only distinguishes between header and data, but it is also important what the destination of the doubleword, or parts of the doubleword, is as it can belong to different decoding engines. Thus, the complete fan-out has to be taken into account as calculated in chapter 3.8.6, chapter 3.9.4, and chapter 4.8. Table 4-9 shows the different fan-outs from HT, PCIe, and ULP.

	HT	PCIe	ULP
Fan-Out	13	21	10

Table 4-9: Comparison Fan-Out

In table 4-10 an overview about the comparison is given. It shows that ULP is at least competitive or superior than the other protocols with regard to the performance.

	HT	PCIe	ULP
Achievable Neutral Bandwidth in GB/s	33.42	39.11	39.11
Achievable Bandwidth in %	55.7	97.8	97.8
Device Count	32	64 K	1M
Efficient Network Diameter	~3	~4	~72
Link Distance in Meter	0.62	7.77	83.97
Decoding Effort	high	medium	low

Table 4-10: Comparison Overview

In conclusion, it can be said that with its features ULP fulfills all needs for a protocol which can be used among all different hierarchies of a system and avoids time consuming protocol translations and complex hardware effort.

Chapter 5: Hardware Development

5.1 EXTOLL

5.1.1 Introduction

The roots of the EXTOLL project started in 2005 at the Computer Architecture Group (CAG) which is located at the University of Heidelberg. EXTOLL is a NIC that was especially build for HPC to realize an interconnection network with a large node count. It is optimized for latency, scalability, bandwidth, and message rates. This has been achieved by optimizing every single layer, from software to hardware.

One approach to optimize latency is to reduce time consuming protocol translations. Therefore one supported host interface of EXTOLL is HT, which enables the device to connect directly to a processor without any intermediate bridging devices. This also guarantees also high bandwidth as the link to the processor does not have to be shared with other devices. In order to avoid complex protocol translations the internally used HyperTransport On-Chip (HTOC) [64] protocol is closely coupled to the HT protocol.

EXTOLL is a switchless design, which means the switch is integrated inside the EXTOLL hardware and no additional hardware like external switches are needed. Every EXTOLL device has six links that can be connected in every suitable configuration, for example a 3D torus [65].

In figure 5-1 an overview over the EXTOLL architecture is given. The Extoll architecture can be distinguished into three main blocks: the host interface, the network interface, and the network. Two different host interfaces are available, HT or PCIe. The network interface contains six different modules: the HTAX (HypterTransport™ Advanced X-bar), the Address Translation Unit (ATU) [66][67], the Virtualized Engine for Low Overhead (VELO) [68][69], the Remote Memory Access Unit (RMA) [70][71], the Shared Memory Functional Unit (SMFU) [72][73], and the register file. In order to provide different communication schemes EXTOLL includes three functional units, RMA, VELO and SMFU. The Network consists of the Network Ports (NP), Link Ports (LP), and the actual switch. During the development of EXTOLL the principles were always tested in hardware [74].

The modules which were worked on during this thesis are the host interface and the RMA, which are highlighted red. For a better understanding of the complete functionality of the host interface and the network interface of EXTOLL will be described, the network itself will be left open. The highlighted blocks will be described in more detail.

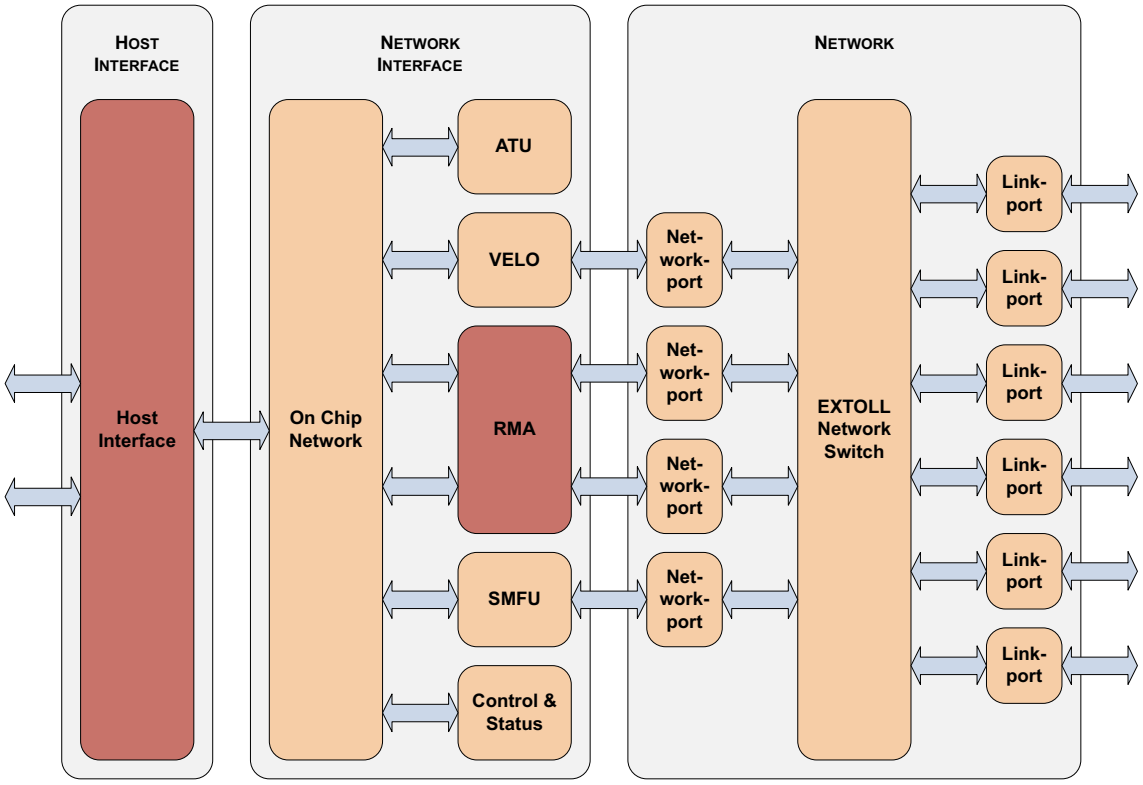


Figure 5-1: Changed Modules

5.2 Host Interface

The host interface is responsible for the connection between EXTOLL and the host node. Current interfaces which can be used to connect EXTOLL to the host system are PCIe and HT. As PCIe is the most common standard to connect extender cards to the system it seems to be a logical choice to be included into EXTOLL. Nevertheless, it needs a bridge chip, which translates the protocol used from the CPU into valid PCI packets, and in typical systems, this link between CPU and bridge needs also to be shared with all other devices connected to PCI that communicates with the CPU.

In contrast, HT has an immediate connection to the CPU as it is the protocol used by AMD

CPUs. HT also provides a connector specification [75], which enables developers of motherboards to integrate an HT connector to a system.

For rapid prototyping several extender cards with different FPGAs for HT and PCIe have been developed [76][77][78]. In those designs, only the corresponding host interface that matches the card type has been included to save space and to ease the testing. For the ASIC version of EXTOLL both interfaces are included in the chip and it can be chosen before runtime which kind of interface is used to connect EXTOLL to the system. A block diagram of the ASIC host interface is shown in figure 5-2.

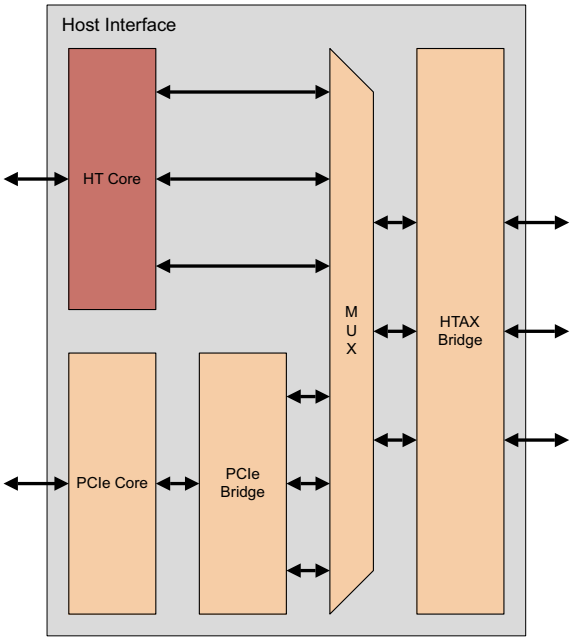


Figure 5-2: Host Interface EXTOLL ASIC

5.2.1 HT Interface

For the first designs of EXTOLL a host interface with low latency was desired because low latency is one of the main features of EXTOLL. An interface directly connected to the CPU was a very promising starting point to achieve this task. At the beginning only the HT Gen1 interface was available and provided by Opteron processors. Thus, the decision was made to build a Gen1 core with the minimal feature set which was tested with a corresponding FPGA board [76]. This core was later extended [79] to achieve better performance and additional features which is part of this thesis. After the development of the Gen1 core was finished and showed its potential [80] it was released as open source.

As the Gen1 core fulfilled the needed requirements in the first test phase the development was extended to a HT Gen3 core by the CAG. This was done because of bandwidth reasons and to be able to integrate EXTOLL into Opteron systems by not influencing the overall system with the low link speed.

5.2.1.1 Gen1 Interface

The Gen1 interface was built for rapid prototyping. Gen1 capabilities of HT provide low frequencies and several link widths which made it possible to integrate a functional interface into an FPGA. The first version of the Gen1-Core only fulfilled the absolute minimal criteria for a connection with an Opteron processor. In [81] the first version of an HT-core was developed. It was used to evaluate if it was possible to connect an external accelerator card to an Opteron processor via an HTX-Connector. The parameters which were chosen were an 8 bits link width and a link frequency of 200 MHz DDR. An internal link width of 32 bits was defined as it is the HT protocol granularity, this resulted in a core frequency of 100 MHz. As target device a Virtex 4 FX 60 was chosen. A block diagram of the first version of the Gen1-Core is shown in figure 5-3. Only data paths are illustrated and the control signals between the different modules have been left out for clarity reasons.

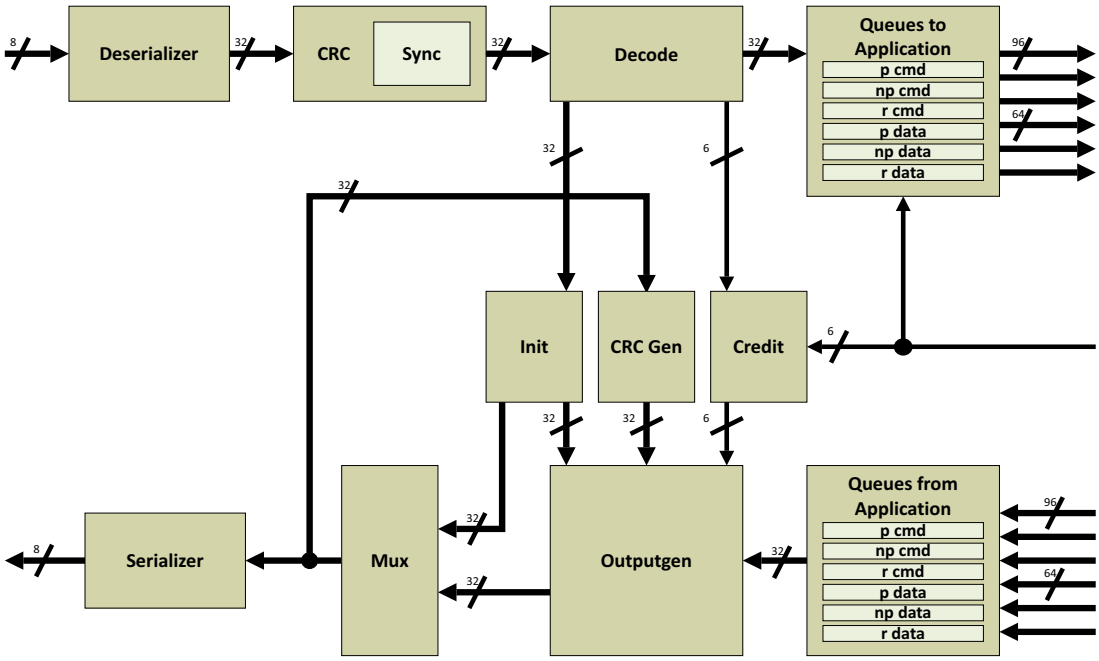


Figure 5-3: 8 Bits / HT200 Gen1-Core

The first version was a relatively quick to develop because of the limitations that had been made regarding to performance. As HT is doubleword aligned the decoding can be handled with a decent hardware effort as the internal data width was also 32 bits. Thus, the first version of the Gen1 core only needed ~10% [82] of logic of the FX60 FPGA. This left enough space to include the first prototype version of EXTOLL to test the functionality of the processing units.

During the thesis, the Gen1 core was extended to increase performance. The link width was doubled to the maximum capable link with of an Opteron to 16 bits and the link speed was raised from HT200 to HT400. In order to provide the needed bandwidth a compromise between internal frequency and data path width was chosen. The internal frequency was raised to 200 MHz and the internal link width was doubled to 64 bits. In addition, the handling of the LDTSTOP signal was included in the core. LDTSTOP is optional in HT but for x86 platforms it is required. The modules which were changed for the new version of the Gen1 core are marked in figure 5-4.

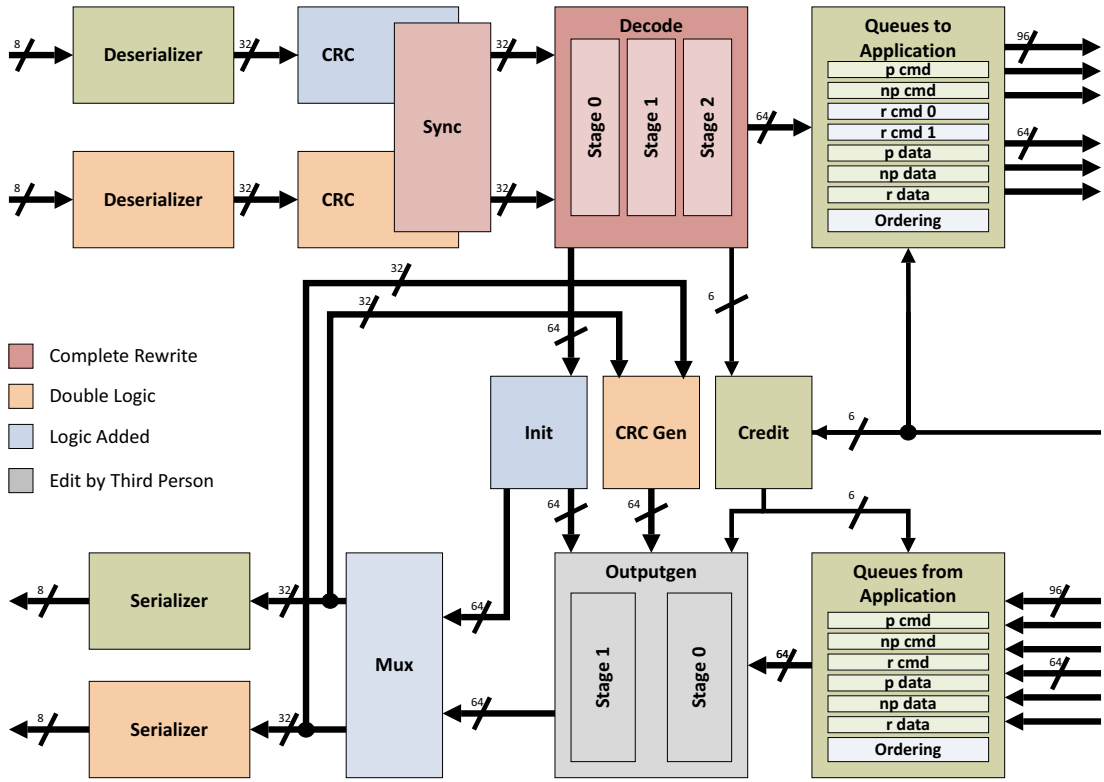


Figure 5-4: HT-Core Changes

In order to realize the doubled link width for the receive path additional deserializer mod-

ules had to be used for the extra lanes. The first version of the core only needed to synchronize the data from the link clock domain into the core clock domain. However, byte-lanes are allowed to be delayed to each other. Therefore, a mechanism is needed to ensure that all data belonging to one clock cycle is valid at the same time. In addition, the clock difference between the two sending clocks and the receive clock are handled here. Those clocks are allowed to differ in a range of +/-1000ppm for the sending clocks and +/-2000ppm for the receiving clocks. The delay was handled by using two synchronizing FIFOs which both have to be valid to consume their data. This also handles the case when the receive clock is faster than the send clock. In the other case, if the receive clock is slower that the send clock, the CRC is extracted and evaluated before it would be inserted into the FIFO, which allows the receiver to catch up with the data. This is possible as the CRC is calculated for each byte-lane separately.

Changes for the transmit path were very moderate. Additional serializers were needed for the extra lanes. As the data that has to be serialized can be transmitted with the same clock, only the additional serializer module was needed. In addition, changes had to be made to the muxing structure to lead the correct bytes to the corresponding lanes.

During the initialization sequence of HT several capability- and configuration registers are read and written by the host. In order to enable the changes made by the CPU there exist two possibilities in HT. One is the Warm Reset and the other one is the LDTSTOP sequence. Warm Reset can be simply seen as a reset signal which resets the whole logic but the value of some configuration registers is maintained which is for example used to change to larger link widths and higher frequencies. The first version of the core only supported this mechanism as the target system was capable of handling this behavior. However, HT demands the LDTSTOP sequence for x86 systems. Therefore, the LDTSTOP sequence had to be added to the HT core. Changes had to be made to the synchronization module, the initialization module, and the outputgen module. After receiving the start of the LDTSTOP sequence the sequence must be performed as shown in figure 5-5.

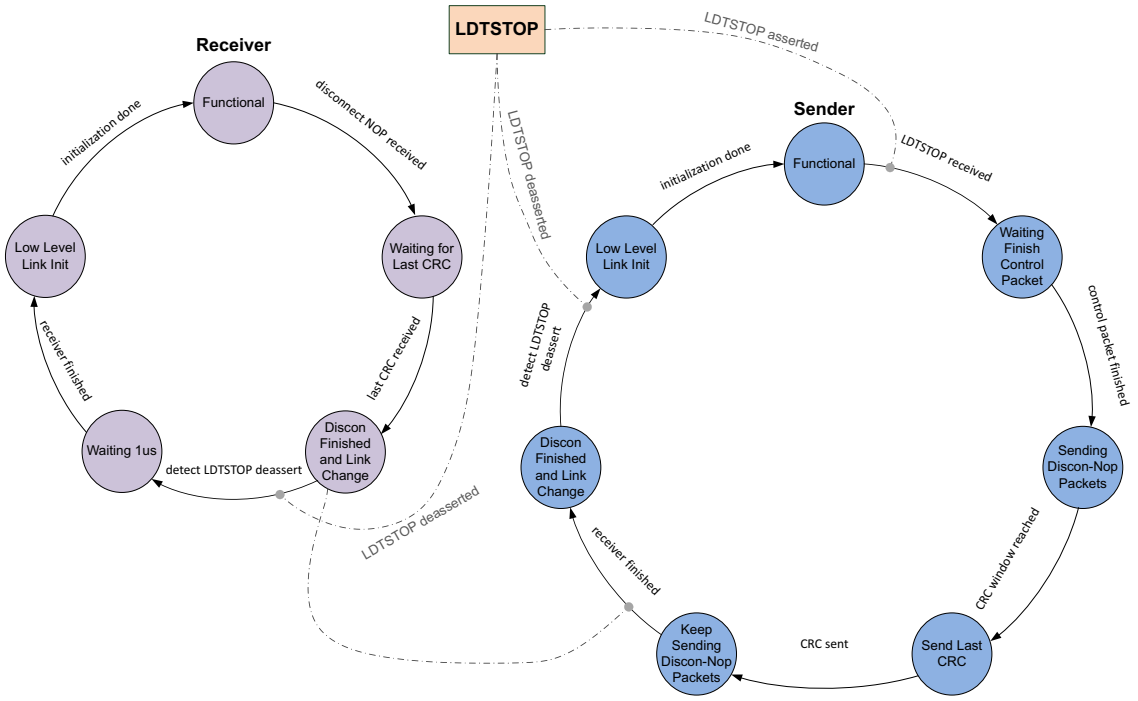


Figure 5-5: LDTSTOP Sequence

First the LDTSTOP must be asserted. The sender now finishes the current control packet and starts to send disconnect NOP packets afterwards. Disconnect NOP packets are sent at least 64 bit times after the CRC of the current CRC window was transmitted. If the receiver is finished with the LDTSTOP sequence the sender can be deactivated at this point, otherwise it keeps transmitting disconnect NOP packets until the sender is finished.

For the sender the disconnect NOP sequence starts with the first reception of a disconnect NOP packet. The sender keeps receiving packets until the CRC of the last CRC window with non-disconnect NOP packets is received.

At this point changes to the register file will be performed like taking over link width and frequency changes. Afterwards the device is sensitive for the deassertion of the LDTSOP signal. If it is deasserted the sender ramps up the clocks to the programmed frequency and then asserts CTL to start the low level link initialization sequence. The receiver inputs remain deactivated for at least 1us after the deassertion of LDTSTOP, then they are activated and waiting for the assertion of CTL.

In the first version of the HT core the decode module was a complex case construct which decoded the data and forwarded it in one clock cycle. This was possible because only one doubleword was received in a clock cycle. Therefore, a large case was sufficient but hard to comprehend when looking at the code. To be able to process the new link bandwidth the internal bandwidth needed to change from 400MB/s to 1600MB/s. As already mentioned this was achieved by doubling both, the internal data path width from 32 to 64 bits as well as increasing the core frequency from 100 to 200 MHz. Both changes made the reuse of the old decode module impossible because more than one doubleword was now received in one clock cycle and the timing budget was too small to process the whole data at once. The decoding mechanism was totally changed and the logic was pipelined into three stages. Figure 5-6 shows that doubling the data width creates multiple possibilities what combinations of doublewords may be received. This is in close relation to how the complexity of the hardware increases. On the left side it is shown what type of data can be received in one clock cycle while on the right side the decoding effort is visualized as well. The decoding effort not only includes the position of the doublewords but also takes into account what data can be received before and after a doubleword.

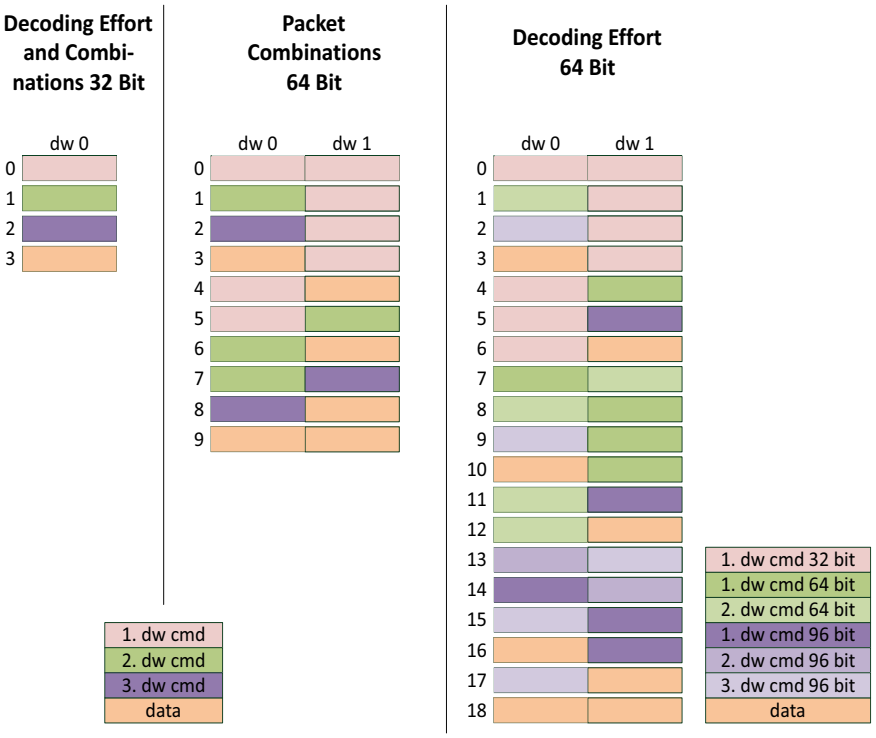


Figure 5-6: Complexity Increase through Data Width Increase

The first stage consists of four parallel modules. Those modules analyze the data stream for the type of commands which may be received, the positions of the commands in a quadword, the number of data which may be attached to the command, and if the packet is a configuration packet. All doublewords are analyzed but some information is not valid. The later stages are built in a way that they only use the information if it is valid.

The second and the third stage belong logically together and consist of three modules each. In the first approach it was analyzed if only two stages were sufficient to reach the target frequency for the complete decoding logic. It showed that two additional pipeline stages were necessary. So the logic was separated into two portions. In the first portion the information from the first stage was evaluated which was compacted to small bit vectors for the corresponding task and in the second portion the bit vector was used to head the data stream to the corresponding buffers and to create the shift-in signals. One module collects the data which is forwarded to the command buffers, one module is responsible for the corresponding payload data, and the last module generates the shift in signals for the buffers.

A simple analysis was performed to determine the increased complexity of the decode module. As an example of how this was done a fragment of the Verilog code is shown in figure 5-7. The signal bit16_en (highlighted red) was used to distinguish if the core was used with an 8 bits link width or a 16 bits link width. It can be seen that there are only seven cases for the 8 bits link, whereas the 16 bits link covers 43 cases. This means that the complexity is over 6 times higher as for the 16 bits version. The analysis of the other modules showed a similar difference of 5.9 to 7.5 times.

```
if ( valid )
begin
  casex ( { bit_16_en, ctrl_nxt, ctl_2_fi, cmd_hi, cmd_lo } )
    { NO , FIRST_TO_COME, BOTH, NPW  , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { NO , FIRST_TO_COME, BOTH, PW   , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { NO , FIRST_TO_COME, BOTH, NPR  , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { NO , FIRST_TO_COME, BOTH, BC   , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { NO , FIRST_TO_COME, BOTH, ARMW , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { NO , FIRST_TO_COME, BOTH, ADREX, X6    }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b10; end
    { NO , TWO_TO_COME  , BOTH, X6   , X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, LOW , X6   , ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, NOP  , NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, NOP  , PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, NOP  , NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, NOP  , BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, NOP  , ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, NOP  , ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FLUSH, ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, FENCE, NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FENCE, PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FENCE, NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FENCE, BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FENCE, ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, FENCE, ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, RDR  , NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, RDR  , PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, RDR  , NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, RDR  , BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, RDR  , ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, RDR  , ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, TGTD , NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, TGTD , PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, TGTD , NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, TGTD , BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, TGTD , ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES, FIRST_TO_COME, BOTH, TGTD , ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    { YES, FIRST_TO_COME, BOTH, ADREX, X6    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b10; end
    { YES,   ONE_TO_COME, BOTH, X6   , NPW   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES,   ONE_TO_COME, BOTH, X6   , PW    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES,   ONE_TO_COME, BOTH, X6   , NPR   }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES,   ONE_TO_COME, BOTH, X6   , BC    }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES,   ONE_TO_COME, BOTH, X6   , ARMW  }: begin ctrl_nxt <=   ONE_TO_COME; was_adrex <= 2'b00; end
    { YES,   ONE_TO_COME, BOTH, X6   , ADREX }: begin ctrl_nxt <=   TWO_TO_COME; was_adrex <= 2'b01; end
    default                                   : begin ctrl_nxt <= FIRST_TO_COME; was_adrex <= 2'b00; end
  endcase
end
```

Figure 5-7: Gen1-Core Complexity Code Example

An additional problem of increasing the data width from 32 to 64 bits was that two complete HT packets could now be received in one clock cycle. This is only possible for two types of packets, NOPs and target dones as other packets are always larger due to an address or attached data. A second NOP packet in one clock cycle is no problem as only the credit information has to be summed up and forwarded to the decoding logic. An additional target done on the other hand causes problems as two complete packets have to be forwarded to the receive queues. One possibility to solve this problem was to delay the second target done until the first one was forwarded. This would work if it could be guaranteed that before receiving two further target dones in the same clock cycle, there would

be one clock cycle with no target done to be able to forward the buffered target done. As this is not guaranteed by the protocol another solution has been chosen. The queue for the responses was split into two queues of half depth. The received response packets were now alternately stored in the queues and if two target dones are received at the same clock cycle they were both shifted into the corresponding queue. Logic was added to the output of the queues so that the change was transparent to the applications and it still looked like one queue.

5.2.1.2 Ordering

In the first version of core all applications connected to the outgoing queues had to handle the ordering rules by themselves. This was no problem as all of these modules fulfilled the strict ordering rules naturally. Later during the extension of EXTOLL an additional unit was development which is called the SMFU. System tests for this unit [83] showed that without the ordering scheme inside the core, deadlocks can easily occur. So it became mandatory to include the functionality inside the core.

The ordering rules of HT are defined by the specification and are shown in table 5-1.

row pass column	posted request		non-posted request	response	
	ppw = 0	ppw = 1		ppw = 0	ppw = 1
posted request, ppw = 0	no	no	yes	yes	yes
posted request, ppw = 1	yes/no	yes/no	yes	yes	yes
nonposted request, ppw = 0	no	no	yes/no	yes/no	yes/no
nonposted request, ppw = 1	yes/no	yes/no	yes/no	yes/no	yes/no
response, ppw = 0	no	no	yes	no	no
response, ppw = 1	yes/no	yes/no	yes	yes/no	no

Table 5-1: Ordering Rules HT-Spec

It can be seen that there are many fields that show "yes/no". At these fields a decision can be made if it is allowed for one packet to overtake another one. In this implementation of the core packets which travel inside one virtual channel are not able to overtake each oth-

er, so these fields are filled with "no". In all other cases of "yes/no" the field is filled with "yes". This was done to ease the hardware but still fulfill all of the conditions where it is allowed to overtake, but it does not have to. The resulting table can be seen at table 5-2. The ordering of the core can be simply described as posted packets are not allowed to be overtaken by non-posted and response packets without the PPW-bit set.

row pass column	posted request		non-posted request	response	
	ppw = 0	ppw = 1		ppw = 0	ppw = 1
posted request, ppw = 0	no	no	yes	yes	yes
posted request, ppw = 1	yes	yes	yes	yes	yes
nonposted request, ppw = 0	no	no	yes	yes	yes
nonposted request, ppw = 1	yes	yes	yes	yes	yes
response, ppw = 0	no	no	yes	no	no
response, ppw = 1	yes	yes	yes	yes	no

Table 5-2: Ordering Rules Core

Different ordering schemes were possible. One idea was to simply tag the packets with a sequence number depending on the time slot it was received. Theoretically an elegant solution as it could be easily checked which packet was received first and if it could be processed without violating the ordering rules. The problem hereby was how to handle the overflow of the tags if a packet is delayed for a long time. In addition the logic needed to check if a tag is larger than another is complex and therefore timing and resource consuming.

The mechanism used was much simpler and is more efficient. Until they reach the queues all packets which travel through the core have a strict order. At this point packets which travel in different virtual channels could overtake each other. To handle this, an additional ordering FIFO is used. This FIFO is just 3 bits wide and deep enough to maintain one entry for each entry in every control FIFO. One bit position inside an entry represents one virtual channel. Every time a control packet is shifted into the corresponding FIFO it is checked if the PPW bit of the command header is set. If it is set then no value is inserted

into the ordering FIFO. Otherwise, one entry is inserted into the ordering FIFO with the corresponding virtual channels set. As multiple packets can be received in one clock cycle multiple bits can be set in one entry. With valid entries in the ordering FIFO the output is checked for the posted virtual channel bit. If it is set the entry is held until the corresponding packet from the posted virtual channel FIFO is consumed. Otherwise, if the posted bit is not set the value is extracted from the FIFO. When a packet of the same virtual channel is consumed at the same time no action has to take place. If no packet is consumed a counter for the corresponding virtual channel type is increased. With this information a decision can be made if a packet can be consumed. Valid combinations are shown in table 5-3.

virtual channel	posted fifo valid	non-posted fifo valid	response fifo valid	posted ordering entry valid	non-posted ordering entry valid	response ordering entry valid	non-posted counter != 0	response counter != 0
posted	1	x	x	x	x	x	x	x
non-posted	x	1	x	0	1	x	x	x
	x	1	x	x	x	x	1	x
response	x	x	1	0	x	1	x	x
	x	x	1	x	x	x	x	1

Table 5-3: Criteria for Ordering Conform Packet Consummation

The generation of the new valid signal for the application made a new pipeline stage mandatory, otherwise every application module would need to get all information and the logic to evaluate if a packet can be consumed. With the extra pipeline stage the valid signal would need two clock cycles to show the correct next value. Every application which receives data from the buffers would need to take care of this delay. In order to avoid this, a special build FIFO has been added. Instead of a normal FIFO, a FIFO with two registered output stages has been used. If no packet was consumed the valid was generated from the first entries of the FIFOs, else from the second entry. With this feature the ordering logic has been kept transparent.

Two implementations of FIFOs with two outputs have been made. In the first version two FIFOs of half the size were used in parallel. Shifting in or out was alternated every time. In simulation this solution worked, but on the FPGA the usage of RAM-blocks increased which became a problem of the resource usage. There also started to be a timing problem to reach the target frequency because of the additional muxing in front and after the buffers. Therefore, a second type of FIFO was created which is built of a RAM based FIFO followed by a smaller register based FIFO. The RAM based FIFO and can be bypassed if

it is empty to optimize latency. The first two entries of the register based FIFO are visible for the next pipeline stage. An overview of the ordering logic can be seen in figure 5-8.

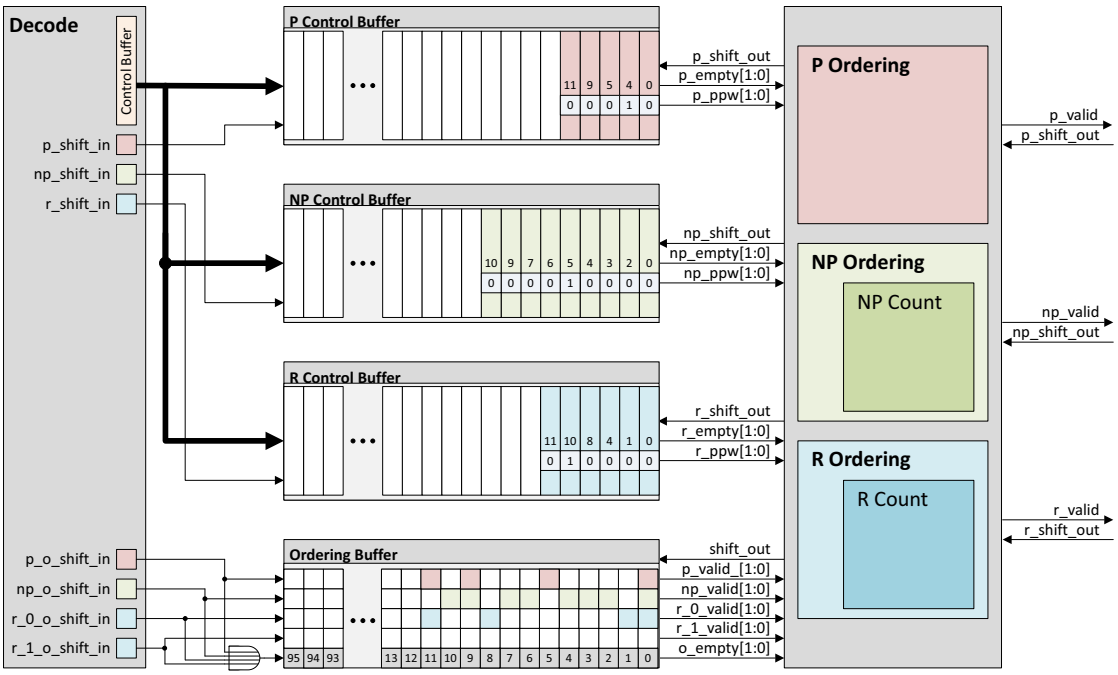


Figure 5-8: Ordering Logic

The outputgen module also needed to be rewritten and pipelined. This was performed by the CAG and is therefore not described in detail.

5.2.2 Gen3 Interface

The ASIC version of EXTOLL needed an interface which provides a higher bandwidth than the Gen1 core. A Gen3 compliant version was built by the CAG [84] to provide the higher link frequency and therefore more bandwidth [85]. As the minimal link frequency for a Gen3 device is 1.2 GHz the resulting minimal link bandwidth is 4.8 GB/s for a 16 bits wide link. Therefore the internal bandwidth has to be adapted by raising the data width to 128 bits and the core frequency to 300 MHz.

The internal bandwidth change of the Gen1 core already showed the massive impact to the hardware complexity. This increased complexity and additional mandatory features, as for example re-transmission, made a complete rewrite of the core structure mandatory.

Although next generation FPGA technology promises higher achievable frequencies more pipeline stages are needed to achieve the needed frequency for Gen3. The additional pipeline stages increased the latency of the core but also resulted in the problem of the available credits of the Opteron. Referring to [62][63] the Opteron does not provide a large number of credits and therefore the bandwidth drops for smaller packets as the round trip latency cannot be hidden. As long as the maximum bandwidth can be achieved with larger packets this is no major problem, but if more pipeline stages are needed for increasing complexity the problem will aggravate. The functionality of the Gen3 core was tested in [77][78].

In order to compare the complexity of the two cores the number of look up tables (LUTs) needed for an FPGA implementation can be used from [86] and [87]. The Gen1 core needed 6,371 LUTs with a Virtex 4 FPGA whereas the Gen3 core needed 37,094 LUTs with a Virtex 5 FPGA. For a realistic comparison it must be taken into account that the Virtex 4 FPGA uses 4 input LUTs and the Virtex 5 FPGA uses 6 input LUTs which means that it can handle 1.5 times the complexity. To be fair it must be mentioned that a used LUT may not be fully utilized but this still means that the complexity of the Gen3 core is ~8 times as high as the Gen1 core.

5.2.3 PCIe Interface

The third interface available for EXTOLL is PCIe. As it is the most common interface for extension cards EXTOLL also provides a PCIe interface. There are two boards which provide a PCIe solution. One is a FPGA based board called [88] and the other one is for the ASIC based version called [89]. For the FPGA version the integrated PCIe core is used whereas for the ASIC version a third party IP is integrated into the ASIC. In order to connect the PCIe core to EXTOLL a bridge module was built by the CAG which provides the conversion between the PCIe protocol and the internally used HTOC protocol. All information needed to generate valid PCIe responses is also stored there. As the MTU of PCIe can be larger than the MTU of HT the application modules have to be able to handle larger sizes.

5.3 Network Interface

5.3.1 HTAX

The HTAX is a crossbar which switches the traffic among the different functional units and the host interface. Therefore the transmitting unit requests the port of one or multiple target units. If a target unit is capable of receiving traffic it generates a grant to the HTAX. The arbitration logic of the HTAX matches the requests to the grants and forwards the grants to the sending unit. The grant is given until the Release Grant is set by the transmitting unit. This triggers a new arbitration round for the utilized port. For the receiving unit, the data transmission is framed by a Start of Transaction (SOT) signal and will proceed until an End of Transaction (EOT) signal is received. A timing diagram of an example sequence is shown in figure 5-9.

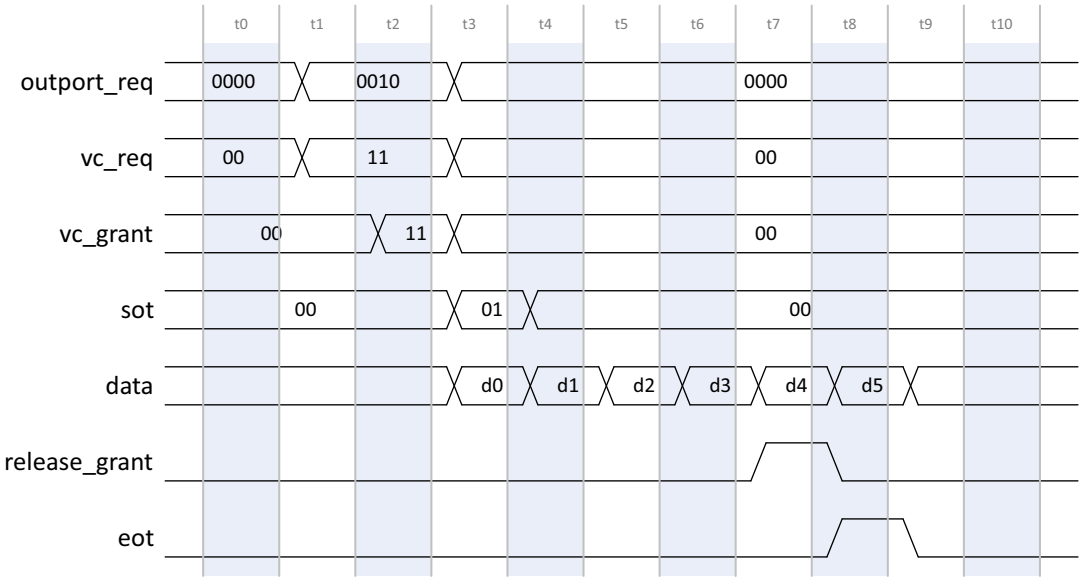


Figure 5-9: HTAX Timing Diagram

The HTAX is protocol agnostic. It provides easy manageable and latency optimized communication between different partners. The participating units are responsible for flow control and must be aware of the restrictions of the receiving unit. In order to achieve high bandwidth, two transmissions can be transferred back to back if the preceding transmission is at least 2 clock cycles long without adding any additional overhead. For detailed information please refer to [56] and [90].

5.3.2 ATU

For security reasons, the user should not be allowed to directly use physical addresses. Otherwise other processes or the operating system could be influenced accidentally or on purpose. Therefore the user has virtual addresses in EXTOLL. Typically the Operating System (OS) is needed to translate a virtual address into a physical address. This handling causes timing overhead because of scheduling of threads which increases the latency of the hardware. The ATU provides an efficient hardware driven address translations for virtual addresses from the user space to the physical address space. Not only the addresses are translated by hardware, which reduces the overall latency of a transaction, but the ATU also provides a Translation Lookaside Buffer (TLB). Thus, the latency is further reduced if the address translation is already stored. For more information please refer to [67].

5.3.3 VELO

This unit provides a communication scheme with minimized latency in terms of hardware and software. The access to the hardware is by user level and therefore avoids a context switch which would be necessary if the access would be made by the operating system kernel. At the sender side data is written to the VELO with a PIO access by a single write operation. If the data is received it is written back with a DMA operation into a ring buffer in the main memory where the CPU can poll for new packets. This might be inefficient in terms of CPU utilization but is the most efficient way from a latency perspective. The principals of VELO are described here [68].

Compared to the RMA, VELO provides a better performance for smaller packages because of the saved DMA access from the device to the main memory. This advantage shrinks if the size of the data increases. The point when RMA becomes more efficient depends on the technology were EXTOLL is used. For more information of the current implementation please refer to [91].

5.3.4 RMA

The first revision of EXTOLL already provided a fully functional RMA unit [72]. In order to enhance the performance and usability some substantial changes were made.

List of changes:

- Added byte access for RMA transactions
- Support for access over 4k boundaries
- Pipelining to hide ATU latency
- Support for lock operations
- Additional data path width of 128 bits
- Support for larger network MTU
- Support for lager host MTU

The RMA communication engine is meant to transfer medium to large amounts of data among nodes. To reduce the load of the CPU descriptors are used to manage a transaction. Those descriptors are sent to the RMA and the rest of the data transfer is completely handled by hardware.

The RMA mainly consists of three modules which are Requester, Responder, and Completer. The Requester and the Responder read data and forward them to the corresponding Completer which writes the data to its destination. Other small modules are used to share resources between the different main modules. Those are the VPID Reader, ATU Handler, and Buffer Queue Handler. A block diagram of the RMA is shown in figure 5-10.

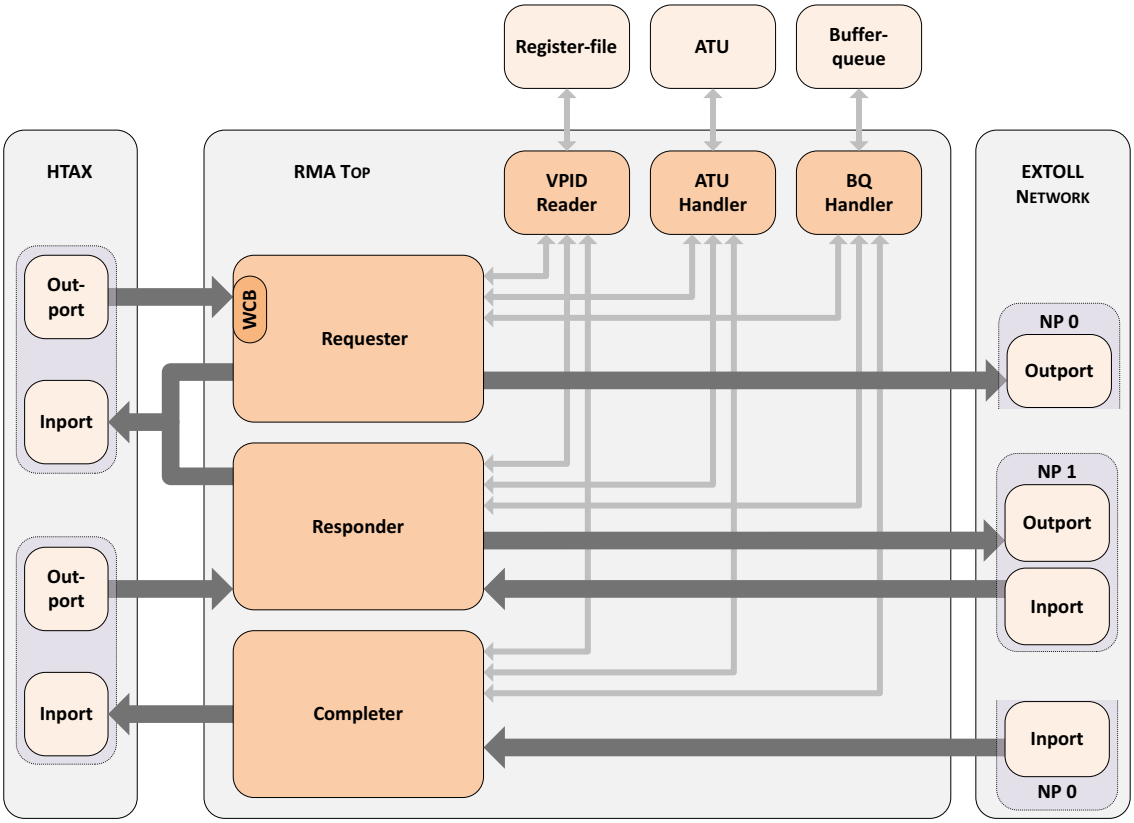


Figure 5-10: RMA Block Diagram

Four communication types are provided: put, get, lock, and special put. The two mainly used communication types are the put and the get operation. If a put operation is performed a descriptor is written to the Requester (1) and the data is read from main memory (2). After the data is received from main memory (3) it is transmitted to the Completer (4) of the receive node and writes the data to memory (5). This sequence is shown in figure 5-11.

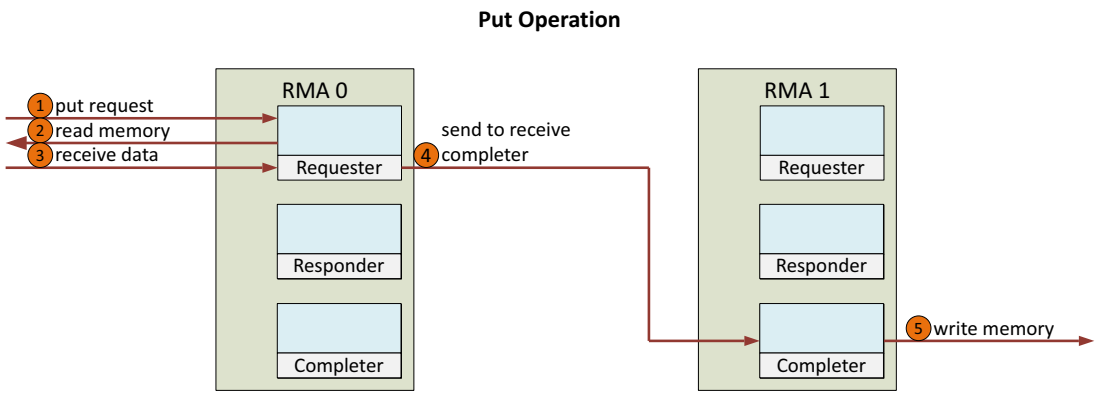


Figure 5-11: RMA Put Operation

For a get request the CPU writes also a descriptor to the Requester module (1). The descriptor is sent to the Responder of the communication partner (2). The Responder reads the data from main memory (3). As soon as the data is received from main memory (4) it is transmitted to the Completer of the RMA which initiated the request (5) and the Completer writes the data to main memory (6). The get sequence is shown in figure 5-12.

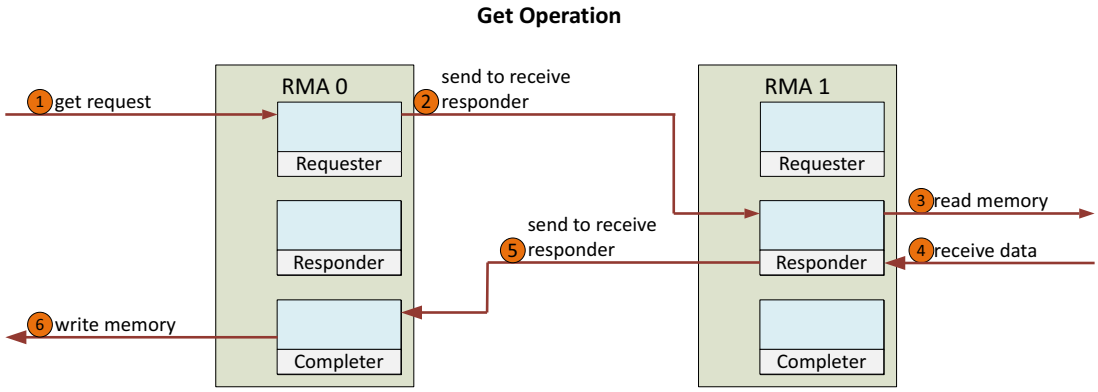


Figure 5-12: RMA Get Operation

In the following sub-chapters the functionality of the RMA modules is described.

5.3.4.1 VPID Reader

The VPID Reader receives the request signals from the different main modules. If more than one request is set simultaneous arbitration has to take place. After reading the values of the corresponding register file entry the values are given back to the requesting module.

5.3.4.2 ATU Handler

Due to the fact that the RMA not only uses physical addresses but also virtual addresses a mechanism is needed to translate a virtual address into a physical address. In order to realize this efficiently the ATU was developed by the CAG. As it is unlikely that every main RMA module requests an address translation all the time but only every couple of clock cycles it is of advantage to share one requesting port between the modules. Therefore the ATU Handler was developed to arbitrate between the different modules. The ATU handler is able to store multiple address translation requests and forwards them to the ATU. If a response to an address translation is given to the ATU Handler it is merged which address translation it belongs to and then it is given back to the requesting unit.

5.3.4.3 Buffer Queue Handler

This unit handles the address requests for notifications of the RMA. Similar to the ATU Handler it is not possible that the different main modules request the buffer queue all the time and therefore the buffer Queue is shared by the Buffer Queue handler. The notifications of different modules are written into the same ring buffer inside the main memory. Only the next possible entry of the ring buffer is visible to the application. Thus, the address may only be requested if the notification is ready to be sent. Otherwise an entry of the ring buffer can be empty while the next one is written and therefore the reception of the invisible notification is delayed. As soon as an address is given by the Buffer Queue Handler to the module the notification is ready to be sent.

5.3.4.4 Requester

The transaction starts at the time when a Software Descriptor is written to the Requester. A software descriptor consists of three doublewords. The configuration of a descriptor is shown in figure 5-13.

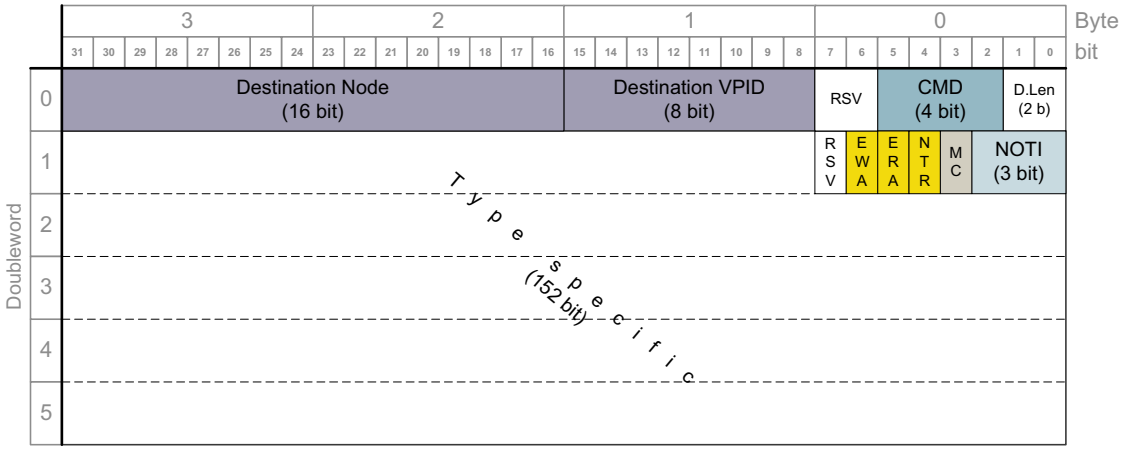


Figure 5-13: RMA Software Descriptor

As it is possible that the CPU splits data belonging to one transaction into multiple host interface packets at a doubleword boundary, it can be necessary to combine the different chunks back to a complete descriptor before forwarding it to the Requester. This is done by the Write Combining Buffer (WCB) which is included in the Requester.

For a received put request the data is read from main memory. As the whole transaction can be much larger than the host MTU multiple accesses to main memory are likely to read the data. Also the network MTU can be smaller than a transaction and the transaction must be split into multiple network packets. Thus, the received data from main memory is gathered until the data for one network packet is received. Then the network packets are sent to the Completer. Data of byte accesses are pre-formatted for the Completer which means that up to the first 3 bytes of the data might be empty regarding to the lower two bits of the destination address. Get requests are framed into a network packet and forwarded to the Completer.

5.3.4.5 Responder

The Responder gets the descriptors forwarded from the Requester. As soon as the descriptor is received from the network the sequence is similar to a put descriptor of the Requester. The data is read from main memory and forwarded to the Completer. The preparation of the data for the Completer is the same as from the Requester.

5.3.4.6 Completer

The Completer consists of six modules: stream gen, VPID check, stream buffer, control buffer, lock order, and write engine. Those modules are separated into three pipeline stages. The two main modules are stream gen and write engine. The stream gen consumes the packets from the EXTOLL network, extracts the control information from it, generates the corresponding requests for other modules, and partitions the incoming data to valid host interface portions. The write engine checks if all information is available to forward a valid host interface packet and to handle the requests to the HTAX. In the second pipeline stage different checks are handled. The VPID check module gets the needed information to match the VPID requests to the corresponding response and stores it until the corresponding network packet is forwarded. The lock order module signals the Responder that a lock packet has been received and that all network packets received before the lock has been forwarded to the host interface. Stream buffer and control buffer just store control information and data until the needed responses from the register file or the ATU have been received to forward the data to the host interface. The buffers are deep enough to

hide the typical latency of the register file and ATU request to prevent the pipeline from stalling. A detailed block diagram of the Completer is shown in figure 5-14.

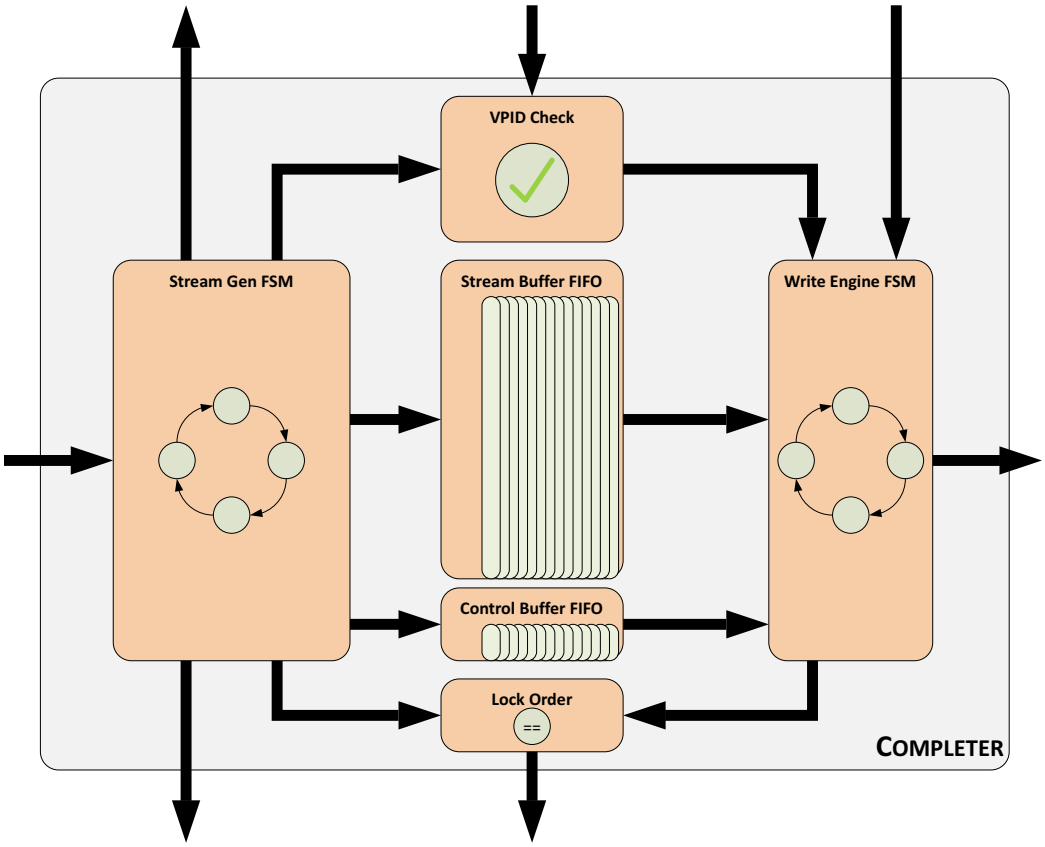


Figure 5-14: Completer Block Diagram

Stream Gen

Network packets received from the network port are handled by the stream gen. At this point the module receives network packets from the network port. A Network packet consists of a SOP cell, a network descriptor, and the data. The structure of a packet received from the network port is shown below.

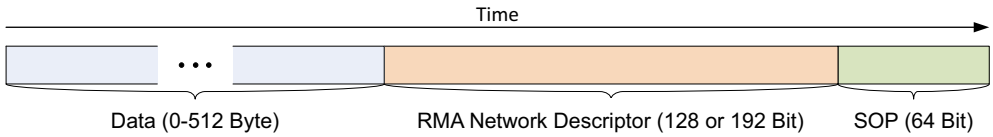


Figure 5-15: Completer Network Packet

Information transported inside the SOP cell is mainly important for the module before the Completer. The only needed information is the destination VPID which is needed to

match the network packet to the corresponding process. The simplified structure of the SOP cell is shown in figure 5-16.

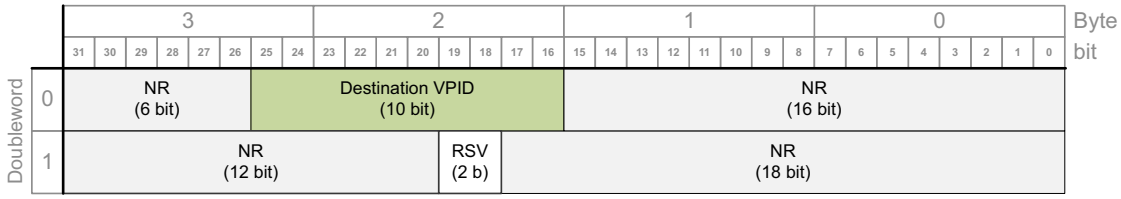


Figure 5-16: Network Packet SOP Header

The network descriptor contains all the information needed to process the network packet besides the destination VPID. Two different sizes are possible for a network descriptor, 128 or 192 bits. At the Completer side only one packet type is 192 bits large which is the lock request. All other types are 128 bits. In z the generic design of a network descriptor is shown. In most cases the type specific 9 bits in the second doubleword describe the payload size, only in lock requests the target is described.

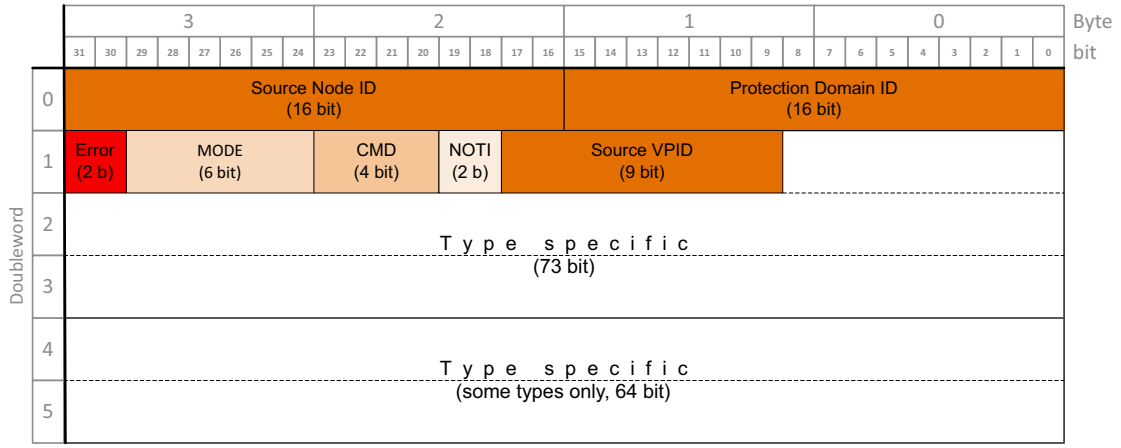


Figure 5-17: Network Descriptor

Field Description (first 64 bits):

- Protection Domain ID (PDID, 16 bits [15:0]):
  Used to ensure that the network packet is allowed to access the memory space of the process.
- Source Node ID (16 bits [31:16]):
  Origin node id where the network packet was generated.
- Payload Size (9 bits [40:32]):
  Number of bytes transferred, where all 0 = 1 byte.

- Target (TGT, 1 bit [32:32]):
  Defines target of a lock packet, 0 = Completer and 1 = Responder, only used for lock packets
- Source VPID (9 bits [49:41]):
  VPID of the origin node process.
- Notification (NOTI, 2 bits [51:50]):
  Corresponding unit must write notification after network packet is done, bit position 0 is for Responder whereas position 1 is for Completer
- Command (CMD, 4 bits [55:52]):
  Describes which type of command is transmitted
- Modifiers (MODE, 6 bits [61:56]):
  Enables different features of the RMA unit
- Error (2 bits [63:62]):
  Signals that errors have occurred in previous modules, bit position 0 is for Requester whereas position 1 is for Responder. Completer needs this information if notifications have to be generated.

Command Encoding

Ten different commands are given in the RMA unit. To binary encode those commands at least 4 bits are needed. A special command encoding was chosen to ease the hardware implementation of the Completer. Bit position one signals if the packet is of a type which might need an address translation. If bit position two is not set the packet belongs to a group which can be handled all the same way corresponding to their structure. Bit position three and two signal if the packet is a short or a lock packet. Therefore not the whole command has to be decoded to determine how the packets have to be handled. A table of the command encoding is shown below.

	Bit-Position			
	3	2	1	0
Byte Put	0	0	1	0
Put	0	0	1	1
Byte Get Response	1	0	1	0
Get Response	1	0	1	1
Byte Get	0	0	0	0
Get	0	0	0	1
Immediate Put	0	1	1	0
Notification Put	0	1	0	1
Lock Request	1	1	0	0
Lock Response	1	1	0	1

normal packet with data attached
short packet
lock requests
packets which might need an address translation

Table 5-4: RMA Command Encoding

Modifiers

The modifiers field contains 6 different bits. Every bit handles one feature of the RMA. An Overview of the features is given at figure 5-18.

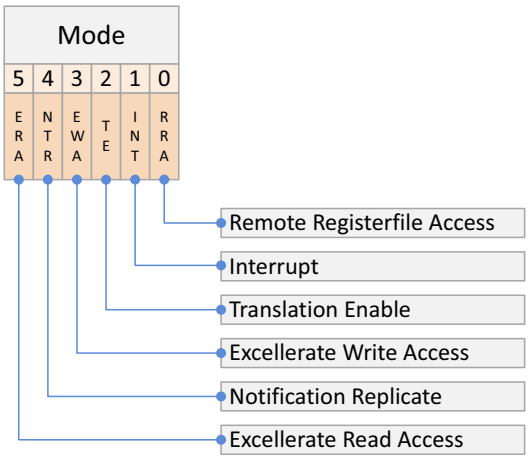


Figure 5-18: RMA Mode Encoding

Field Description:

- Remote Registerfile Access:
  If set, the packet accesses the register file. This allows an access to the register file of the node via the RMA. Therefore, the register file of an EXTOLL card can be accessed via the network interface even if the host interface is not connected.

- Interrupt:
  If set, an interrupt is needed at the end of the packet
- Translation Enable:
  If set, the packet uses virtual addresses so an address translation is needed to handle it.
- Excellerate Write Access:
  If set, data is written to the excellerate interface and not to the host interface.
- Notification Replicate:
  If set, a notification/interrupt is generated for every network packet belonging to one software descriptor and not only to the last one.
- Excellerate Read Access:
  If set, data is read from the excellerate interface and not from the host interface.

Stream Gen Functionality

The functionality of the stream gen can be described with a finite state machine (FSM) with 14 states. During every stage parts of a network descriptor are consumed and information and data are forwarded to other modules. The states of the FSM can be separated into three different groups, data forwarding, special operation, and pause. A picture of the FSM is shown in figure 5-19.

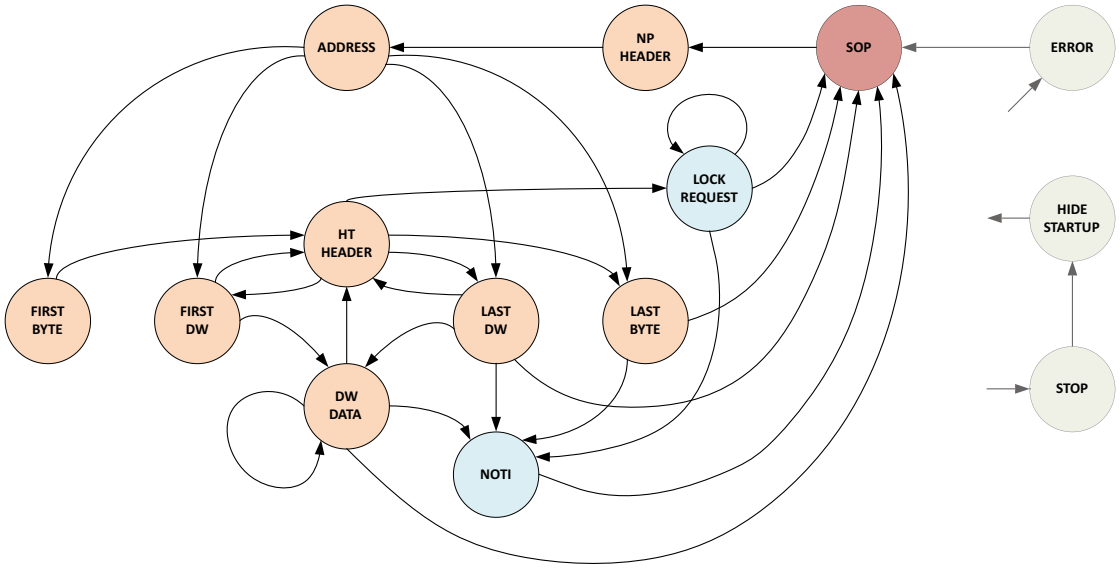


Figure 5-19: Completer Stream Gen FSM

During the data forwarding states network packets, which are normally larger than packets for the host interface, are partitioned into parts that can be transmitted with one host

packet. Therefore, first the needed control information from the network packet, needed to create the host packet header, is stored into the stream buffer. At the same time the needed control information to process the host packet at the write engine is stored in the control buffer. In the following clock cycles the corresponding data of the host packet is also stored in the stream buffer.

There are two special states, one is the lock request state and the other one is the noti state. During the lock state a lock request is consumed from the network port and is forwarded to the lock order module. At the noti state all needed information to generate a notification is stored into the stream buffer and the control buffer. This happens at the end of a network packet which has the notification bit for the Completer set or if a notification put request is received which only triggers a single notification.

During the pause states the stream gen makes no progress even if valid data is available from the network port. There are two possibilities how this can happen. One is that the write engine does not make any progress and back pressure makes it necessary for the stream gen to halt. Therefore the stop state is introduced, it ensures that it is possible to stop in every state without loss of data and to continue processing the network packet without any re-initialization of the FSM. The other one is that an error occurs during the operation of the stream gen. For example, this could occur if the packet length of the network packet does not match the payload size field, which can be determined with the start of packet and end of packet bit which is provided by the network port. If this happens, the stop state ensures that the missing data is forwarded with dummy data and if no new start of a network packet is signaled by the network port the data is drained until a start is signaled. This ensures that even if an error occurs the hardware is still functional and not stuck.

Data Forwarding

Other than the previous version of the RMA it is allowed to transmit every kind of packet size from 1 up to 512 bytes. Therefore the different granularities and transaction types of the protocols have to be taken into account.

For memory accesses which are not aligned to byte boundaries it must be determined how the packet is misaligned. Three types of misalignment are possible, only the start of the data is misaligned, the end of the data is misaligned, and both, start and end, are misaligned.

Data received from the Requester and the Responder is already doubleword aligned to the starting address of the destination, which means that a byte which is written to a byte address with the lower 2 bits zero is also the start of a doubleword. For example, if the destination address has the lower bits set to 10_2 the first two bytes of the data packet are empty and the valid data starts at byte three. This also means that the maximum payload size of the packet is reduced by up to three bytes if the packet starts misaligned.

It is also of importance to not only determine the start- and endpoint of the packet if it is misaligned, but also the length of the packet. This ensures that very small sized packets, which only need a single byte access at the host side, are also handled correctly. The types of possible byte fragmentation of a network packet can be seen in figure 5-20.

Figure 5-20: Completer Byte Access Data Fragmentation

For the host interface a byte mask is needed to perform a byte access. Depending on the host interface the allowed byte masks can look different. Depending on how large the byte mask, which has to be calculated, is the more complex the hardware will be. Therefore the minimum size of a byte access has been chosen to realize.

As the network Maximum Transfer Unit (MTU) is larger than the host MTU large network packets have to be split into multiple host packets. The Completer is able to support

host MTUs from 64 up to 512 bytes. If the first data has to be written to a doubleword misaligned address a byte access has to be performed. The Completer only performs byte accesses at the beginning of a network packet to reach doubleword alignment. Therefore a byte access never exceeds a size of 3 bytes, even if it would be possible to use larger byte accesses of the host interface. This design decision was chosen to reduce the hardware effort compared to the relative low gain of performance. If the performance is important at this point the user should chose host MTU aligned accesses to avoid fragmentation overhead.

After doubleword alignment is achieved the next packet is used to reach the host MTU alignment. Therefore, it is determined how many doublewords are needed to reach the next host MTU address alignment. This is done to fulfill some host protocol requirements. For example, in the HT protocol a packet is not allowed to cross a 64 bytes boundary. With this mechanism it is also ensured that a packet does not cross 4k page boundary. At this point full MTU sized host packets are prepared until insufficient data from the network packet is provided. If still enough data is available for at least one doubleword a further doubleword access is performed. Following this the remainder of the data is forwarded with a byte access. An example how the data of a network packet is fragmented into multiple host packet conform data blocks is shown in figure 5-21.

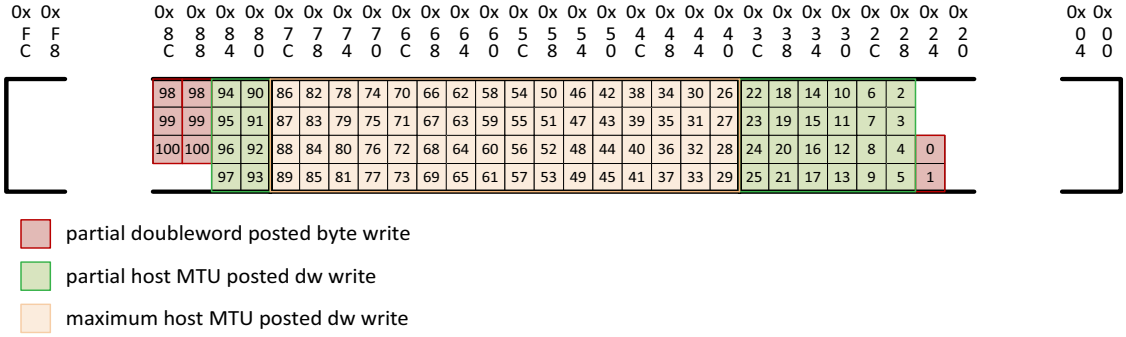


Figure 5-21: Completer Network Packet fragmentation Example

VPID Check

The VPID check module checks all the security bits regarding the VPID of every network packet except for remote register file accesses. Therefore, the information given from the

stream gen is stored in an input FIFO until the data read from the register file by the stream gen is returned. After the check of the PDID and the excellerate bit the gathered data is shifted into an output FIFO where it is stored until it is consumed by the write engine.

Stream Buffer

In order to hide the latency of the responses from the ATU and the register file the stream buffer is introduced. It is a FIFO which stores the data that travels from the stream gen to the write engine. There are two different types of data that are stored. One is the information which is needed to create a header for the host interface and the other one is the raw data which has to be transferred. A header information entry is always followed by at least one doubleword of data even if an error has occurred and the rest had to be filled with dummy data. The encoding of the header information is shown in figure 5-22.

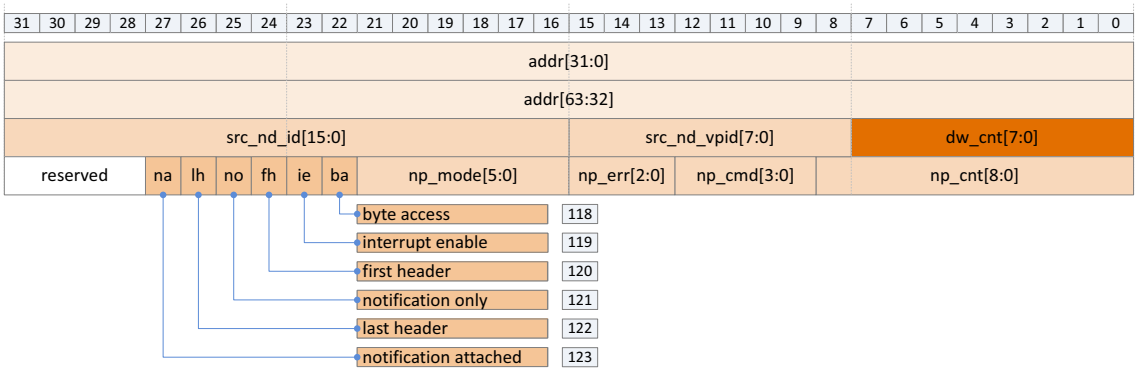


Figure 5-22: Stream Buffer Header Format

Field explanation:

- addr 64 bits [63:0]:
  address of a packet (if it is a physical address all bits are valid, if it is a virtual address only the lower 12 bits are valid)
- dw_cnt 8 bits [71:64]:
  number of raw data in doublewords which have to be sent
- src_nd_vpid 8 bits [79:72]:
  source node VPID needed for error processing Source Node ID
- src_nd_id 16 bits [95:80]:
  source node ID needed for error processing

- np_cnt 9 bits [104:96]:
  network packet count needed for error processing, containing overall size of network packet
- np_cmd 4 bits [108:105]:
  network packet command needed for error processing, containing command type of network packet
- np_error 3 bits [111:109]:
  network packet error needed for error processing, which kind of error occurred
  001: Requester error
  010: Responder error
  100: Completer error
- np_mode 6 bits [117:112]:
  network packet operation modifier for error processing, to determine in which kind of operation the Completer was when an error occurred
- ba 1 bit [118]:
  determines if the packet is a doubleword (0) or byte access (1)
- ie 1 bit [119]:
  interrupt will be generated after the notification has been sent
- fh 1 bit [120]:
  this command is the first header for an entire network packet
- no 1 bit [121]:
  this command is only a notification with no data from a network packet
- lh 1 bit [122]:
  this command contains the last header of a network packet
- na 1bit [123]:
  notification attached, the end of this network packet sequence will be followed by a notification

Control Buffer

The control buffer stores the information which is needed to request the HTAX or to request a notification address from the buffer queue. Every host packet in the stream buffer has its corresponding entry. The format of a control buffer entry is shown in figure 5-23.

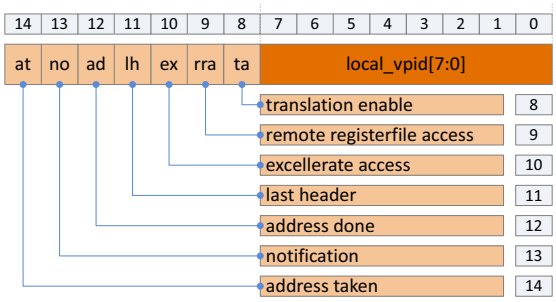


Figure 5-23: Control Buffer Entry Format

Field explanation:

- local_vpid 8 bits [7:0]:
  local vpid is needed to generate a request to the buffer queue or for error handling
- ta 1 bit [8]:
  translation enable determines if a physical or a logical address is used for the HTAX packet
- rra 1 bit [9]:
  remote registerfile access is set if the packet is routed to the register file
- ex 1 bit [10]:
  excellerate access is set if the packet is routed to the excellerate port
- lh 1 bit [11]:
  last header is set if the current HTAX packet is the last header corresponding to one network packet
- ad 1 bit [12]:
  address done is set if the current HTAX packet is the last header corresponding to an address translation and is no longer needed
- no 1 bit [13]:
  notification is set to show that this header is part of an notification packet
- at 1 bit [14]:
  address taken is set if the current HTAX packet is the last header corresponding to an address translation and is in use

Lock Order Module

EXTOLL provides a lock operation which is an atomic fetch-compare-and-add operation. As this feature is mainly handled by the Responder it is not described in detail but only the part of the Completer is described.

In order to guarantee a correct behavior of the lock operation all data previously received has to be written back to the memory before the lock operation takes place. Therefore, the lock order module signals to the Responder that all data received before the lock has been completely processed. Afterwards the lock operation has no chance to overtake any previously received data and can be executed by the Responder without any problems.

The functionality of the lock order module is based on two resettable counters. One of the counters is the in flight count. It shows how many potential host packets have been forwarded from the stream gen to the write engine but not been completed. If a lock request has been received by the stream gen it is forwarded to the lock order module. At this point the current in flight count value is written into a FIFO and the in flight count is reset. Simultaneously the second counter, which is the processed count, starts to count the finished packets of the write engine. Finished packets can be both, packets successfully forwarded to the HTAX or packets which have to be dropped because of an error. If the processed count value is equal to the output value of the FIFO the lock order module signals the Responder that the forwarded lock request from the stream gen is now safe to use. A block diagram of the lock order module is shown in figure 5-24.

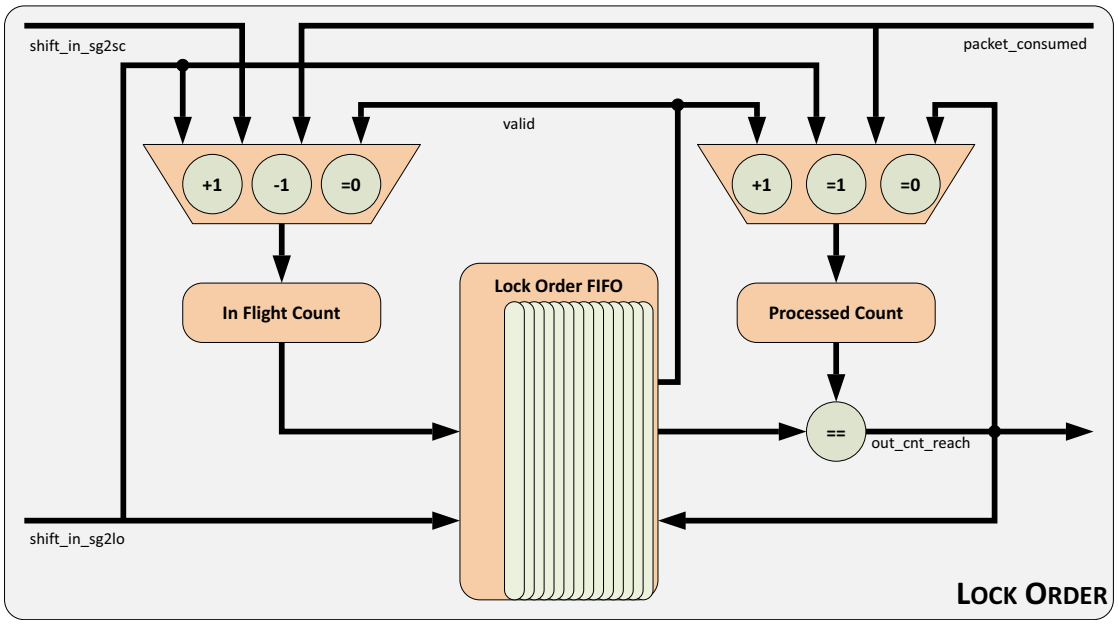


Figure 5-24: Completer Lock Order Module

Write Engine

The write engine consists of two small FSMs which work almost independently as long as no error occurs. They are synchronized with the grant signal of the HTAX. One FSM is responsible for requesting the HTAX and the other one is responsible for forwarding the data to the HTAX. This design decision was made as the HTAX is a critical resource in performance perspective. Neither an unnecessary request has to be set nor should a valid request wait. Both FSMs are shown in figure 5-25.

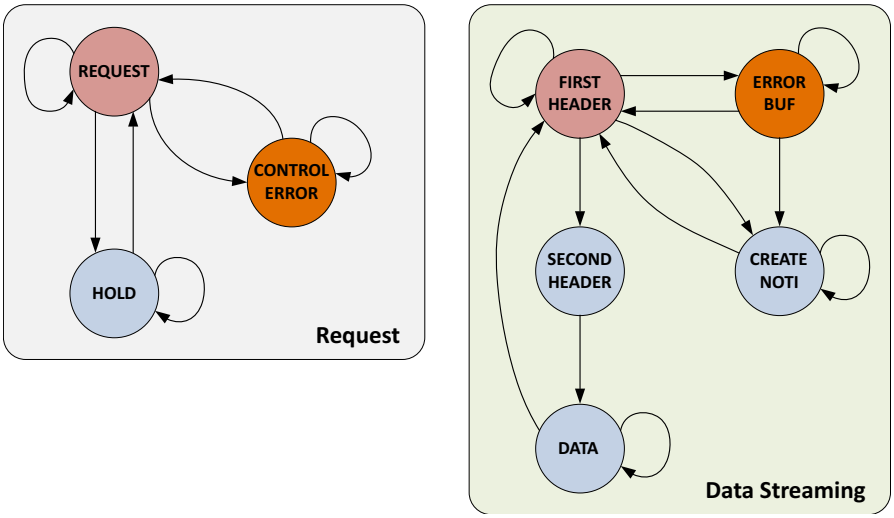


Figure 5-25: Completer Write Engine FSMs

Before requesting an HTAX port the request FSM must ensure that all needed requirements of the entry of the stream buffer are met. A valid physical address must be available and it must be allowed to access the addressed region. If the requirements are given a request to the HTAX is set, the entry of the control buffer is shifted out, and the request FSM goes to the hold state. The request is held until a grant is given from the HTAX. Right after the grant, the state is switched to request again. If the next packet is ready for transmission the request to the HTAX is set again and the state is switched to hold regardless if the previous packet has already finished. This can be done because the next grant cannot be set before a release grant was set from the write engine.

As already mentioned the data streaming FSM is responsible for the data forwarding from the stream buffer to the HTAX. It waits in the first header state with SOP set until a grant

of the HTAX is received and the first part of the header is transmitted. Than it goes to the second header state and transmits the second part of the header. As a host packet from the Completer may contain only one doubleword of data it must be checked if the release grant is already set in this state. In the next clock cycle the data streaming FSM goes in the data state and only streams the data from the stream buffer to the HTAX. If the second to last data portion is reached the release grant is set which enables the HTAX to give a new grant. With the last portion of data the EOP is set to signal the end of the host packet. Afterwards the FSM goes to the first header state. The sequence is shown in figure 5-26.

Notifications need a special handling from the write engine. The sequence is almost similar to the normal data forwarding of the data streaming FSM but only a single header is stored without data. In order to be able to build a correct host packet for the notification the needed information of the first host packet of a network packet is stored. Therefore, a special state was introduced to ease the different handling.

The error handling of the write engine is complex because of the almost independent work of the two FSMs. For starting and ending an error sequence the request FSM is responsible. Therefore, it has to check if all requirements to start a transaction are met. If an error occurs because of an invalid address or trying to access a forbidden destination the states of both FSMs have to be taken into account. An error can only be determined during the req state of the request FSM and it will jump into the control error state as soon as an error is detected. At this point it is of importance in which state the data streaming FSM is. During the first header state the current packet has to be dismissed and the state will be changed to error buf in the same clock cycle. Otherwise, the previous packet has to be completed before a change to the error buf state can be made. The packet will be finished regardless of the error. After the previous packet has been finished the state will be changed to error buf. In the error buf state the current packet will be dismissed by streaming it out of the stream buffer, the error bits will be set, and an interrupt or a notification will be sent if it is indicated.

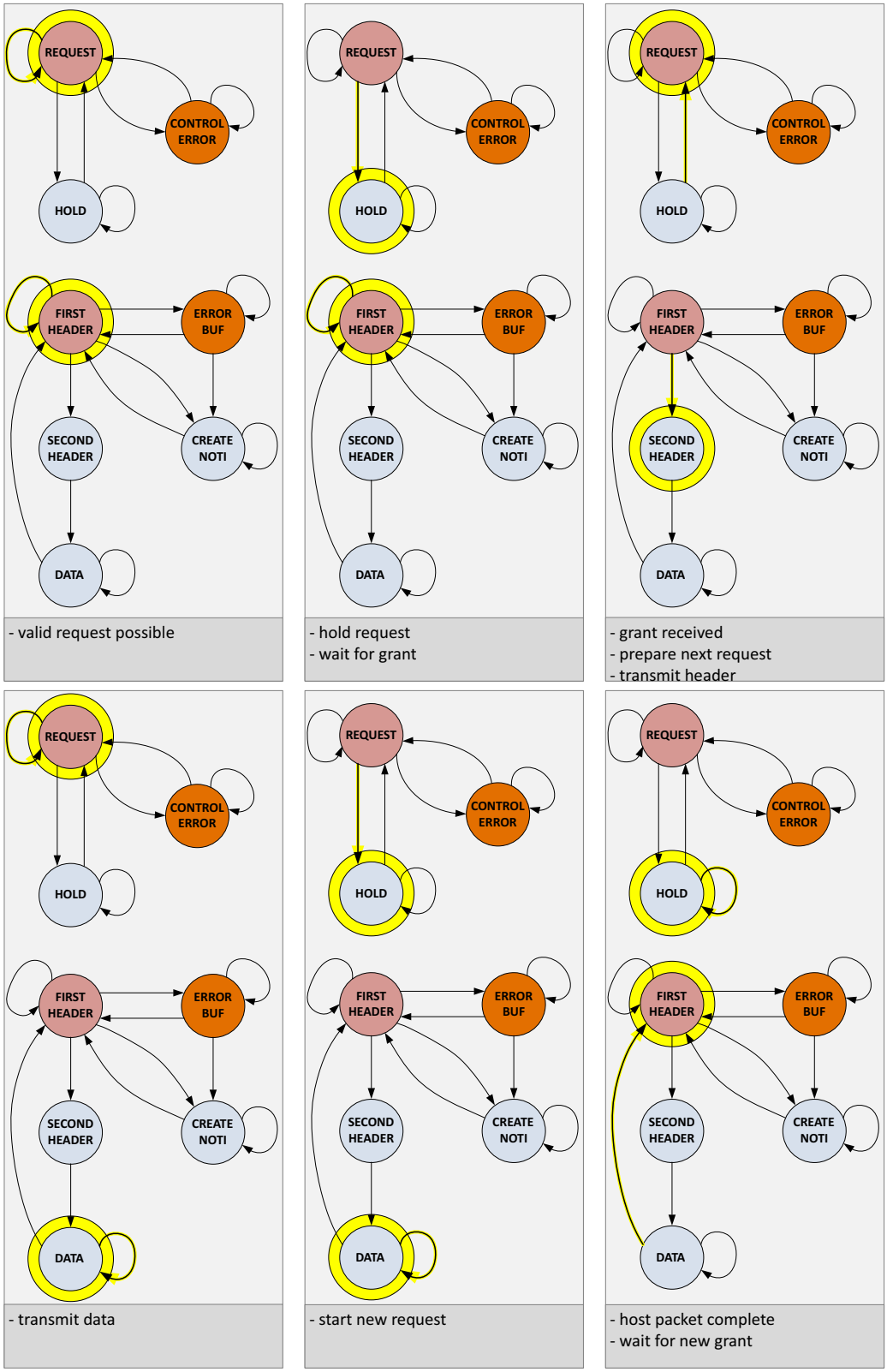


Figure 5-26: Completer Write Engine Sequence

Chapter 6: Conclusion and Outlook

6.1. Conclusion

The main goal of this thesis was the optimization of communication schemes in an HPC system. First a general evaluation has been made about the necessary features of a protocol. The analysis showed the existence of complex dependencies among the different features. Those dependencies make it difficult to optimize the properties as the change of one nearly always has an influence on another. Thus, the main task of a protocol is finding a balance without a negative impact on performance.

In chapter 2 the relevant state of the art protocols HT and PCIe have been analyzed with reference to the previously defined features and their attributes. For the analysis only protocols closely coupled to the processor were relevant because the processor is the start of the communication process.

The individual contribution of this thesis is a network protocol which is innovative and applicable at every point inside an HPC system regardless of the distance of the communication partner. Particular attention has been paid to the balance between feasibility in hardware, protocol granularity, and communication distance. Some characteristics are shared with the state of the art protocols but substantial enhancements have been made. With the optimized alignment among the different layers of the protocol a better scalability can be reached. Thus, the communication scheme is scalable for large systems with up to 2^{20} participants. In addition, dependencies have been minimized in every respect and therefore the hardware implementation effort is low. The results of ULP have been compared with the state of the art protocols showing that ULP is superior in most cases and in all others at least competitive.

Beyond the development of the ULP protocol, contributions have been made to the EXTOLL design. The work on both projects has cross-fertilized each other. During the hardware development two main tasks have been accomplished. One was the data width and frequency extension of the HT core which has been used in various academic and industry projects. The other one was the extension of the RMA unit. In both cases a protocol decoding and translation was the essential part of the task. The derived findings of ULP were

integrated in the corresponding hardware modules to optimize the performance. Especially the hardware efficient methodology of payload transfers with any byte granularity and address alignment has been proven.

In combination the two projects accomplished the objective of a protocol which is feasible to replace all other protocols of an HPC system. It efficiently provides communication among processors, inter node communication, and node to node communication. It allows a minimized communication overhead, optimized latency due to reduced hardware effort, and a competitive bandwidth. In addition, it is capable of scaling for large systems without restrictions to the performance. While doing this, all time and power consuming bridging devices and the corresponding protocol translations are eliminated.

6.2. Outlook

For the future there are three interesting elements which have to be analyzed and included into the development. The first point is how cache coherent traffic can be included into the protocol. A simple solution would be to extend the current header format with the necessary cache coherent packets to handle the corresponding traffic. This would only result in additional decoding effort for the coherent packets, which is easy to handle. The disadvantage of this solution is the relatively coarse packet granularity of ULP as coherent traffic consists mainly of small packets which might not need the full granularity and therefore waste bandwidth. Thus, as already noted in chapter 4.3, an additional layer is intended for ULP which allows smaller packet sizes and only handles the coherent traffic. It must be determined how this layer can be integrated into the data stream without wasting bandwidth, but also with minimal influence to the upper layers. The two alternatives for the coherent communication need to be analyzed to achieve a reasonable decision.

The second point which is interesting for the future is eliminating the view of hierarchical boundaries from the protocol point of view. If the communication scheme among devices is unified the location of the device is not of importance, but the distance to the communication partners. Thus, a mechanism must be included to measure the distance among them. As an example, for a coherency domain it is not important if the communication

partner is in the same node, another node, or a different rack, it is only important if it can be reached in a defined period of time. In order to realize this, the link distance can be measured by defined timing packets and their responses. With this information, coherent devices with a specific time-distance can be grouped to one coherency domain. Thus, they can be used as if they would be in one node. For the future it must be analyzed what the advantages of this approach can be, how the system can be initialized, and for which use case it is relevant.

The third point is analyzing the influence of ULP in terms of energy efficiency. It is assumed that with ULP the hardware complexity of a single device is significantly lowered compared to other devices as special attention was put on it. This should result in an overall lowered energy consumption. However, a fair comparison is difficult to perform as devices do not only consist of a network interface and therefore the power consumption of a single network interface is complex to measure. Information on the hardware complexity and energy consumption of other interfaces is hard to get. Even if they are available the efficiency is always dependent of how they were implemented. For example, two independently developed PCIe cores with the same features can have substantial power consumption differences. In addition, the energy consumption of the supported protocol features would be needed for a fair comparison. Otherwise, optional and complex features could not be considered. Nevertheless, with ULP fewer devices are needed in the system, because the unnecessary protocol translation engines are eliminated.

Abbreviations:

AC	Alternating Current
ACK	Positive Acknowledgment
AMD	Advanced Micro Devices
ARQ	Automatic Repeat Request
ATU	Address Translation Unit
ATOLL	Atomic Low Latency
ASIC	Application-Specific Integrated Circuit
BCM	Byte Count Modified
BDF	Bus Device Function
BER	Bit Error Rate
BKDG	BIOS and Kernel Developer's Guide
CAG	Computer Architecture Group
CAS	Chinese Academy of Sciences
CDR	Clock Data Recovery
CLK	Clock
CMD	Command
COM	Comma
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CTL	(not defined at HT specification)
DC	Direct Current
DDR	Double Data Rate
DEEP	Dynamical Exascale Entry Platform
DLL	Data Link Layer (PCI)
DLLP	Data Link Layer Packet
DW	Double Word
ECC	Error Correcting Code
ECRC	End-to-End CRC

EDB	End Bad (TLP/DLLP)
EDS	End of Data Stream
EIE	Electrical Idle Exit
EIOS	Electrical Idle Order Set
END	End (TLP/DLLP)
EOT	End of Transfer
EXTOLL	Extended ATOLL
FCRC	Frame CRC
FEC	Forward Error Correction
FIFO	First In First Out
Fmt	Format
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FTS	Fast Training Sequence
FU	Functional Unit
Gb	Giga Bit
GB	Giga Byte
GHz	Gigahertz
GPU	Graphics Processing Unit
GT	Giga Transfers
HARQ	Hybrid Automatic Repeat Request
HDL	Hardware Description Language
HPC	High Performance Computing
HT	HyperTransport
HT1	HT Generation 1
HT2	HT Generation 2
HT3	HT Generation 3
HTAX	HyperTransport Advanced X-Bar
HTOC	HyperTransport On-Chip

ICT	Institute of Computing Technology
ID	Identification
IDEL	Logical Idle
IDO	ID based Ordering
ISOC	Isochronous
LCRC	Link CRC
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LTSSM	Link Training and Status State Machine
LUT	Lock Up Table
MBU	Multi Bit Upset
MCU	Multi Cell Upset
MHz	Megahertz
MIC	Many Integrated Cores
MSB	Most Significant Bit
MSI	Message Signaled Interrupt
MTU	Maximum Transfer Unit
NAC	Negative Acknowledgment
NIC	Network Interface Controller
NOP	Null Packet
NOTI	Notification
NP	Network Port
OS	Operating System
OS	Ordered Set (PCIe)
OSI	Open System Interconnect
OW	Octaword
PAD	Pad
PCB	Polychlorinated Biphenyl
PCIe	Peripheral Component Interconnect Express

PDID	Protection Domain ID
PPW	Pass Posted Write
QOS	Quality Of Service
QPI	Quick Path Interface
QW	Quad Word
RMA	Remote Memory Access
RMW	Read Modify Write
RRA	Remote Registerfile Access
SATA	Serial Advanced Technology Attachment
SDP	Start DLLP
SEB	Single Event Burnout
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEL	Single Event Latchup
SET	Single Event Transient
SEU	Single Event Upset
SFP	Small Form-factor Pluggable
SKP	Skip
SMFU	Shared Memory Functional Unit
SOP	Start of Packet
SOT	Start of Transfer
STP	Start TLP
TGT	Target
TH	TLP Processing Hints
TC	Traffic Class
TL	Transaction Layer
TLB	Translation Lookaside Buffer
TLP	Transaction Layer Packet
TS	Training Set (PCIe)

TS	Training Sequence (ULP)
ULP	Unified Layer Protocol
USB	Universal Serial Bus
VG	Value Group
VPID	Virtual Process ID
WCB	Write Combining Buffer

List of Figures

List of Tables

Bibliography

[1] TOP500 Supercomputer website, http://www.top500.org [Online July 2015].

[2] Intel website, *Intel Xeon Processor E3-1200 v3 Product Family,* http://www.intel.com [Online July 2015].

[3] AMD website, *AMD Opteron 6300 Series Processor Quick Reference Guide,* https://www.amd.com [Online July 2015].

[4] Intel website, *Intel Xeon Phi Coprocessor x100 Product Family,* http://www.intel.com [Online July 2015].

[5] IBM Blue Gene team, *Overview of the IBM Blue Gene/P project,* IBM Journal of Research and Development, vol. 52, no. 1.2, pp. 199-220, Jan. 2008.

[6] Intel website, *Intel® Xeon® Processor E7-8800/4800 v3 Product Families: Brief,* https://www.intel.com [Online July 2015].

[7] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, *NVIDIA Tesla: A Unified Graphics and Computing Architecture,* Micro IEEE, vol. 28, no 2, pp. 39-55, March 2008.

[8] N. Eicker and T. Lippert, *DEEP – An Accelerated Cluster Architecture for Exascale Computing,* Intel European Exascale Labs Report 2011, pp. 12-17, http://www.exascalecomputing.eu [Online July 2015].

[9] Herb Sutter, *The Free Lunch Is Over - A Fundamental Turn Toward Concurrency in Software,* Dr. Dobb's Journal, vol. 30(3), March 2005.

[10] S. Borkar and A. Chien, *The Future of Microprocessors*, Communications of the ACM, vol. 54, no. 5, pp. 67-77, May 2011.

[11] R. Maddox, G Singh, and R. Safranek, *Weaving High Performance Multiprocessor Fabric,* ISBN-10: 193405318X, ISBN-13: 978-1934053188, Intel Press, Oct. 2009.

[12] D. Anderson, and J. Trodden, *HyperTransport System Architecture,* ISBN-10: 0321168453, ISBN-13: 978-0321168450, Addison-Wesley, Feb. 2003.

[13] B. Holden, J. Trodden, and D. Anderson, *HyperTransport 3.1 Interconnect Technology,* ISBN-13:978-0-9770878-2-2, Mindshare, Inc., Sep. 2008.

[14] HyperTransport Consortium, *HyperTransport I/O Link Specification Rev. 3.1,* http://www.hypertransport.org [Online July 2015]

[15] Numascale website, *NumaConnect™ A high level technical overview of the NumaConnect technology and products*, https://www.numascale.com [Online July 2015]

[16] M. Jackson and R. Budruk, *PCI Express Technology,* ISBN-13: 978-0-9770878-6-0, Mindshare, Inc., Sep. 2012.

[17] IEEE Computer Society, *IEEE Std 802.3ae™-2002, IEEE Inc.,* IEEE Standards, Aug. 2002.

[18] InfiniBand℠ website, *InfiniBandSMArchitecture Specification Volume 1 Rel. 1.3,* http://www.infinibandta.org [Online July 2015].

[19] EXTOLL website, http://extoll.de/index.php/technology/resourcesapapers.

[20] C. Whitby-Strevens, *The Transputer,* 12th Annual International Symposium on Computer Architecture, Proceedings, pp. 292-300, June 1985.

[21] B. Duzzet and R. Buck, *An Overview of the nCUBE 3 Supercomputer,* Fourth Symposium on the Frontiers of Massively Parallel Computation, pp. 458-464, Oct. 1992.

[22] MPI website, *MPI: A Message-Passing Interface Standard - Version 3.1,* http://www.mpi-forum.org [Online July 2015].

[23] USB website, *Universal Serial Bus 3.1 Specification,* http://www.usb.org [Online July 2015].

[24] SATA website, *Serial ATA Revision 3.2 Specification,* https://www.sata-io.org [Online July 2015].

[25] H. Zimmermann, *OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection,* IEEE Transactions on Communications, vol. 28, no. 4, pp. 425-432, April 1980.

[26] W. Peterson and D. Brown, *Cyclic Codes for Error Detection*, Proceedings of the IRE, vol. 49, no. 1, pp. 228-235, Jan. 1961.

[27] F. Lemke, *Unified Synchronized Data Acquisition Networks,* Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2012.

[28] Altera website, *Arria 10 Transceiver PHY User Guide,* https://www.altera.com [Online July 2015].

[29] M. Yan, *SONET/SDH Essentials,* White Paper, SONET Aggregation and T/E Carrier Applications Group, Exar Corporation, 2008.

[30] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks*, ISBN-10: 1-55860-852-4, ISBN-13: 978-1-55860-852-8, Morgan Kaufmann, July 2002.

[31] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, *Myrinet: A Gigabit-per-Second Local Area Network,* IEEEMicro,vol. 15, no. 1, pp. 29-36, Feb 1995.

[32] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg,The Quadrics Network (QsNet): High-Performance Clustering Technology, Hot Interconnects 9, pp. 125-130, Aug. 2001.

[33] J. Martinez, M. Koibuchi, J. Flich, A. Robles, P. Lopez, and J. Duato, *In-Order Packet Delivery in Interconnection Networks using Adaptive Routing,* Proceedings 19thIEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Denver, April 2005.

[34] I. Gopal, *Prevention of Store-and-Forward Deadlock in Computer Networks,* IEEE Transactions on Communications, vol. 33, no. 12, Dec. 1985.

[35] R. Cypher and L. Gravano,Storage-Efficient, *Deadlock-Free Packet Routing Algorithms for Torus Networks,* IEEE Transactions onComputers, vol. 43, no. 12, Dec 1994.

[36] R. Cypher and L. Gravano, *Requirements for Deadlock-Free,* Adaptive Packet Routing,11thAnnual ACM Symposium on Principles of Distributed Computing, Proceedings, pp.25-32, 1992.

[37] W. Dally and C. Seitz, *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,* IEEE Transactions on Computers, vol. C-36, no. 5, pp. 547–553, May 1987.

[38] A. Chien and J. Kim, *Planar Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors,* The 19th Annual International Symposium on Computer Architecture, Proceedings, pp. 268-277, 1992.

[39] C. Glass and L. Ni, *The Turn Model for Adaptive Routing,* The 19th Annual International Symposium on Computer Architecture, Proceedings, pp. 278-287, 1992.

[40] M. Nicolaidis et al., *Soft Errors in Modern Electronic Systems,* ISBN 978-1-4419-6992-7, Frontiers in Electronic Testing, Vol. 41, Springer, 2011.

[41] A. Frantz, F. Kastensmidt, L. Carro, and E. Cota, *Evaluation of SEU and Crosstalk Effects in Network-on-Chip Switches,* Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design, pp. 202-207, 2006.

[42] R. Hamming, *Error Detecting and Error Correcting Codes,* Bell System Technical Journal, vol. 29, no. 2, pp. 147–160, 1950.

[43] R. Bose and D. Ray-Chaudhuri, *On A Class of Error Correcting Binary Group Codes,* Information and Control, vol. 3, no. 1, pp. 68–79, Mar. 1960.

[44] I. Reed and G. Solomon, *Polynomial Codes Over Certain Finite Fields,* Journal of the Society for Industrial and Applied Mathematics, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[45] J. Ray and P. Koopman, *Efficient High Hamming Distance CRCs for Embedded Networks,* International Conference on Dependable Systems and Networks, pp. 3-12, June 2006.

[46] P. Koopman, *32-bit Cyclic Redundancy Codes for Internet Applications,* International Conference on Dependable Systems and Networks, Proceedings, pp. 459–468, 2002.

[47] P. Koopman and T. Chakravarty, *Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks,* International Conference on Dependable Systems and Networks, pp. 145–154, June 2004.

[48] Mathworks website, http://www.mathworks.de/de/help/comm/ug/error-detection-andcorrection.html [Online July 2015].

[49] L. Lev and P. Chao, *Down to the Wire - Requirements for Nanometer Design Implementation, Eetimes,* http://www.eetimes.com/document.asp?doc_id=1205898 [Online July 2015], Aug. 2002.

[50] Chip-Architect website, http://www.chip-architect.com/news/2010_09_04_AMDs_Bobcat_versus_Intels_Atom.html [Online July 2015].

[51] Fraunhofer IZM website, http://www.izm.fraunhofer.de/de/abteilungen/high_density_interconnectwaferlevelpackaging/arbeitsgebiete/mikrogalvanischesbumpingaufchipwafern.html [Online July 2015].

[52] F. Gray, *Pulse Code Communications,* U.S. Patent 2632058 A, Mar. 1953.

[53] A. Widmer and P. Franaszek, *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code,* IBM Journal of Research and Development, vol. 27, no. 5, pp. 440-451, Sept. 1983.

[54] N. Burkhardt, *A Hardware Veri_cation Methodology for an Interconnection Network with fast Process Synchronization,* Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2012.

[55] W. Dally, *Virtual-Channel Flow Control*, 17th Annual International Symposium on Computer Architecture, pp. 60-68, May 1990.

[56] H.Litz, *Improving the Scalability of High Performance Computer Systems,* Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2010.

[57] Molex website, *Technology in Development - I/O Infiniband,* http://www.molex.com [Online July 2015].

[58] Samtec website, *QSFP+ Active Optical Cable Assembly User Manual,* http://www.samtec.com [Online July 2015].

[59] Samtec website, *High Density/High Speed Cable System,* http://www.samtec.com [Online July 2015].

[60] J. Duato, F. Silla, B. Holden, P. Miranda, J. Underhill, M. Cavalli, S. Yalamanchili, U. Brüning, and H. Fröning, *Scalable Computing: Why and How,* HyperTransport Consortium, White Paper, March 2010.

[61] HyperTransport Consortium, *HyperTransport High Node Count - System-Wide Resource-Sharing,* http://http://www.hypertransport.org [Online July 2015].

[62] AMD website, *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors,* https://www.amd.com [Online July 2015].

[63] AMD website, *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 30h-3Fh Processors,* https://www.amd.com [Online July 2015].

[64] H.Litz and A.Giese, *HyperTransport On-Chip (HTOC) Protocol Specification,* Internal Documentation, Computer Architecture Group at the Department of Computer Engineering, University of Mannheim.

[65] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks,* ISBN-10: 0122007514, ISBN-13: 978-0122007514, Elsevier Science & Technology, Jan. 2004.

[66] M. Nüssle, *Acceleration of the Hardware-Software Interface of a Communication Device for Parallel Systems,* Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2008.

[67] C. Leber, *Efficient Hardware for Low Latency Applications*, Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2012.

[68] H. Litz, H. Froening, M. Nüssle, and U. Brüning, *VELO: A Novel Communication Engine for Ultra-low Latency Message Transfers,* 37th International Conference on Parallel Processing ICPP'08, pp 238–245, Sept. 2008.

[69] H. Litz, H. Froening, M. Nüssle, and U. Brüning, *A HyperTransport Interface Controller for Ultra-low Latency Message Transfers*, HyperTransport Consortium, White Paper, Feb. 2008.

[70] M. Nüssle, M. Scherer, and U. Brüning, *A resource optimized remote-memory-access architecture for low-latency communication,* International Conference on Parallel Processing ICPP '09, pp. 220-227, Sept. 2009.

[71] M. Scherer, I*mplementation, Synthesis and Verification of a Remote Shared Memory Access Functional Unit*, Diploma Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2008.

[72] H. Froning and H. Litz, *Efficient Hardware Support for the Partitioned Global Address Space*, 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd. Forum (IPDPSW), pp. 1-6, April 2010.

[73] F. Silla, H. Froning, J. Duato, and H. Montaner, *MEMSCALETM: a Scalable Environment for Databases*, 2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC), pp. 339-346, Sept. 2011.

[74] M. Nüssle, B. Geib, H. Fröning, and U. Brüning, *An FPGA-Based Custom High Performance Interconnection Network*, International Conference on Reconfigurable Computing and FPGAs (ReConFig '09), pp. 113-118, Dec. 2009.

[75] HyperTransport Consortium, *HTX3™ Specification for HyperTransport™ 3.0 Daughtercards and ATX/EATX Motherboards*, http://http://www.hypertransport.org [Online July 2015].

[76] H. Fröning, M. Nüssle, D. Slogsnat, H. Litz, and U. Brüning, *The HTX-Board: A Rapid Prototyping Station*, 3rd annual FPGAWorld Conference, Nov. 2006.

[77] H. Litz, H. Froning, M. Thürmer, U. Brüning, *An FPGA based Verification Platform for HyperTransport 3.x*, International Conference on Field Programmable Logic and Applications, pp. 631-634, Aug. 2009.

[78] F. Lemke, S. Kapferer, A. Giese, H. Fröning, and U. Brüning, *A HT3 Platform for Rapid Prototyping and High Performance Reconfigurable Computing*, Proceedings of the Second International Workshop on HyperTransport Research and Applications (WHTRA2011), Feb 2011.

[79] M. Nüssle, H. Fröning, A. Giese, H. Litz, D. Slogsnat, and U. Brüning, *A HyperTransport Based Low-Latency Reconfigurable Testbed for Message-Passing Developments*, Proceeddings of the Workshop Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC), 2007.

[80] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, *An Open-Source HyperTransport™ Core*, ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 1, no. 3, pp. 1-21, Sep. 2008.

[81] A. Giese, *Development and Verification of a HyperTransport-Interface with Optimizations for FPGA Environments*, Diploma Thesis presented to the Department of Computer Engineering, University of Mannheim, March 2007.

[82] D. Slogsnat, A. Giese, and U. Brüning, *A Versatile, Low Latency HyperTransport Core*, Proceedings of the 15th International Symposium on Field Programmable Gate Arrays, pp. 45-52, 2007.

[83] H. Fröning, A. Giese, H. Montaner, F. Silla, and J. Duato, *Highly Scalable Barriers for Future High-Performance Computing Clusters*, 18th International Conference on High Performance Computing (HiPC), pp. 1-10, Dec. 2011.

[84] B. Kalisch, *Design and Implementation of a HyperTransport I/O-Link Controller complying with Specification 3.0*, Diploma Thesis presented to the Department of Computer Engineering, University of Mannheim, March 2008.

[85] B. Kalisch, A. Giese, H. Litz, and U. Brüning, *HyperTransport 3 Core: A Next Generation Host Interface with Extremely High Bandwidth*, Proceeding of the First International Workshop on Hyper-Transport Research and Applications (WHTRA'09), Feb. 2009.

[86] D. Slogsnat, A. Giese, and U. Brüning, *Leveraging HyperTransport on Xilinx FPGAs*, Xcell journal, issue 61, third quarter 2007.

[87] B. Kalisch, A. Giese, H. Litz, and U. Brüning, *HyperTransport 3 Core: A Next Generation Host Interface with Extremely High Bandwidth*, Proceedings of the First International Workshop on HyperTransport Research and Applications (WHTRA2009), Feb. 2009.

[88] EXTOLL website, *Galibier Overview*, http://extoll.de/index.php/productsoverview/ galibier, [Online July 2015].

[89] EXTOLL website, *Tourmalet Overview*, http://extoll.de/index.php/productsoverview/ tourmalet, [Online July 2015].

[90] H. Litz, H. Fröning, U. Brüning, *HTAX : A Novel Framework for Flexible and High Performance Networks-on-Chip*, Fourth Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC) in conjunction with Hipeac, Jan. 2010.

[91] B. Geib, *Hardware Support for Efficient Packet Processing*, Doctoral Thesis presented to the Department of Computer Engineering, University of Mannheim, Germany, 2012.