CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

High-speed Serial SpaceFibre Link Software Evaluation

Master's thesis in Computer science and engineering

Jesper Mass

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

High-speed Serial SpaceFibre Link Software Evaluation

Jesper Mass

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

High-speed Serial SpaceFibre Link Software Evaluation
Jesper Mass

Supervisor: Muhammad Waqar Azhar, Department of Computer Science and Engineering
Advisor: Daniel Hellström, Cobham Gaisler
Examiner: Jan Jonsson, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

High-speed Serial SpaceFibre Link Software Evaluation
Jesper Mass
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

SpaceFibre is an emerging standard for onboard spacecraft communication. Cobham Gaisler has recently developed an IP core to communicate through a SpaceFibre link. However, no software driver API to use the IP is currently available. By using SpaceFibre the speed of communication can be increased by up to 15 times compared to the previously used SpaceWire. This increased speed could enable more data to be sent from sensors quicker for processing at the onboard computer. Currently, few other drivers for SpaceFibre exist and there is little benchmarking for how well it actually performs. The aim of this thesis was to design a software driver to benchmark the actual performance and validate the SpaceFibre IP developed at Cobham Gaisler. To accomplish this an external tool, developed by the creators of the SpaceFibre standard, was used: the STAR Fire Mk3. With this tool, a test using the driver designed in this thesis was performed where the STAR Fire Mk3 was used to measure the statistics and act as both a recipient and transmitter of messages. As a result, it was found that the IP core does reach speeds up to 1.91 Gbps for reception and 1.5 Gbps for transmission with a link running at effectively 2 Gbps. Using this the user can reach speeds of at least ten times the speed of SpaceWire, and including all the standardised quality of service the protocol provides.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

I would like to thank the company Cobham Gaisler for giving me the opportunity to write this thesis. I would also like to give extra thanks to, Daniel Hellström, for supporting me and taking the time out of his busy day to answer my questions and leading me in the right direction. Joaquin España Nevarra, for answering all my questions about his SpaceFibre IP and how it works. The other thesis workers Jonathan Jonsson and Matteo Toselli who kept me entertained throughout the day with their Borat references.

Jesper Mass, Gothenburg, May 2023

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Problem Statement . . . 2
1.2 Contribution . . . 2
1.3 Thesis Outline . . . 2

2 Background **3**
2.1 SpaceFibre . . . 3
2.1.1 8B/10B encoding . . . 4
2.1.2 Direct Memory Access . . . 4
2.2 SpaceWire . . . 5
2.3 The SpaceFibre IP . . . 5

3 Design **9**
3.1 Low-level API . . . 9
3.1.1 Initialisation . . . 9
3.1.2 Virtual channels . . . 10
3.1.2.1 Transmission . . . 10
3.1.2.2 Reception . . . 11

4 Methods **13**
4.1 Tools . . . 13
4.1.1 FPGA . . . 13
4.1.2 STAR Fire Mk3 . . . 13
4.1.3 Test setup . . . 14
4.2 Latency . . . 15
4.3 Bitrate . . . 16
4.3.1 Reception . . . 17
4.3.2 Transmission . . . 17

5 Results **21**
5.1 Latency . . . 21
5.2 Bitrate . . . 23

6 Discussion **27**
6.1 Latency 27
6.2 Datarate 27
6.3 Summary 28
6.4 Hardware 28
6.4.1 Hardware limitations 28
6.5 Future research 29

7 Conclusion **31**

Bibliography **33**

List of Figures

2.1 Illustration showing 4 SpaceFibre nodes in a SpaceFibre network. . . 4
2.2 The bit representation of each word in a transmission descriptor. . . . 6
2.3 The bit representation of each word in a reception descriptor. 7

3.1 The data structure is used to reference and keep track of packets in the descriptors. n can be up to 1024 for transmission and m is defined in the software. 11

4.1 A block diagram of the hardware architecture of the prototype design. 14
4.2 The test setup used, with the FPGA board, STAR Fire Mk3 and XILINX adapter. 14
4.3 An illustration of the test setup with the software interfaces and physical interfaces. 15
4.4 A flowchart illustrating the reception latency test. 16
4.5 A flowchart illustrating the transmission latency test. 17
4.6 A flowchart illustrating the reception bitrate test. 19
4.7 A flowchart illustrating the transmission bitrate test. 20

5.1 The result from the bombardment reception test. 23
5.2 The result from the bombardment transmission test. 24
5.3 The number of packets sent per second during the bombardment reception test. 25
5.4 The number of packets sent per second during the bombardment transmission test. 26

List of Tables

5.1 The number of clock cycles and microseconds for each of the driver routines on a 100MHz system. 21
5.2 The bitrate when adding delays to the loop during transmission test. 24

1
Introduction

Space is a big place and the amount of information we have about it is continually growing as we explore more and more of it. To gather more information, the sensors and technology keep improving, and with that the need to support an increasing data-rate, increases. Since 2003, a commonly used standard for onboard communication in spacecraft has been SpaceWire. In 2019, SpaceFibre became an accepted standard by the European Cooperation for Space Standardization (ECSS). SpaceFibre is a communications protocol developed by STAR Dundee for the European Space Agency (ESA), it is backwards compatible with the SpaceWire packet format. However, SpaceFibre has a theoretical data rate up to 15 times faster than SpaceWire. It also features up to 32 virtual channels for communication in virtual networks and one broadcast channel for communicating with all of the connected nodes at the same time.

Cobham Gaisler has recently developed a SpaceFibre codec and a direct memory access (DMA) engine which is compatible with the SpaceFibre specification ECSS-E-ST-50 [1]. It is planned to be included in the next-generation ESA LEON5 fault-tolerant microprocessor [2, 3]. It will serve high-speed serial transfers over a SpaceFibre network in modern satellites. The SpaceFibre IP is available as part of the GRLIB SoC IP Library [4]. Since it was recently developed, it has not yet been manufactured into a Silicon component from Gaisler. Instead, the IP is designed and tested using FPGA development boards, which is also a technology targeted by the GRSPFI IP. During the design and testing of the IP, basic software setups have been used, but the IP has now reached a certain maturity level. It would now be suitable to evaluate a Software stack that could utilize the full bandwidth provided and benchmark potential bottlenecks in Software. Also to propose hardware optimization possible to off-load the Software tasks (the processor). Finding optimizations or issues that lead to IP improvements, could have a significant positive effect on next-generation satellite onboard communication. it is important to discover and evaluate the bottlenecks before starting the manufacturing process into a die. The study outcome and software stack prototype may become an important part of more advanced testing, demonstrators, and finally, the user software application.

According to ESA, [5] the resulting software can lead to bit-rates up to 6.25 Gbps, 15 times faster than the previously used standard SpaceWire. The weight of the wiring can also be reduced by 50% when using optical fibres. Reaching this higher data transfer speed in onboard communication can reduce the impact of delays in computer networks. This will enable the use of higher resolution cameras without impacting the transfer speed, more data to be sent between computational units, and more precise control of actuators.

1.1 Problem Statement

SpaceFibre is a standard for a communications protocol developed by STAR-Dundee and published by ECSS. Cobham Gaisler has developed an IP compliant with the SpaceFibre standard intended to be released with their upcoming GR765 board. As The IP has matured, it is now ready for a software driver to be developed. A software driver that can initiate, configure and utilize the hardware registers according to the IP and specification to communicate with other nodes connected. The SpaceFibre protocol is up to 15 times faster than the previously used standard SpaceWire. Enabling this could provide the means to send more data from sensors such as cameras to the central onboard computer. Introduced in the SpaceFibre standard is the Quality of Service (QoS) aspect. The standardization of QoS would simplify the use of other SpaceFibre-compatible equipment to reduce the loss of packets and increase the guarantee that important data will reach the end-point. By implementing a software driver for the SpaceFibre IP and running it on a development FPGA, this thesis will show that the driver implemented will deliver the messages over the SpaceFibre protocol. it will also validate the SpaceFibre IP developed at Cobham Gaisler and the LEON5 core, and evaluate the performance of the new DMA and HW/SW interface. An external tool will be used to receive and send messages as well as measure statistics. Investigating the latency caused by the driver will make it easier to predict the execution time and the overhead.

1.2 Contribution

Contributions of this thesis are:

- A software driver for using the SpaceFibre core developed by Cobham Gaisler,
- An evaluation of the DMA engine developed by Cobham Gaisler,
- An evaluation of the SpaceFibre core developed by Cobham Gaisler.

1.3 Thesis Outline

The structure of this thesis is as follows. In Chap. 2 the necessary technical background for SpaceFibre, SpaceWire and the SpaceFibre IP will be explained. In Chap. 3 the design of the software driver constructed in this thesis will be described. In Chap. 4 for the test setup and the tests to be performed on an FPGA with the SpaceFibre IP to test the hardware and the software driver will be presented. In Chap. 5 the results of the tests from Chap. 4 will be presented. In Chap. 6 the results from Chap. 5 will be discussed and reviewed. In Chap. 7 the thesis will be wrapped up and the results will be related to the thesis problem.

2
Background

2.1 SpaceFibre

The SpaceFibre standard is a very high-speed serial link and network technology created by STAR-Dundee and was released by the ECSS in 2019. It is a communications protocol created for onboard communication in spacecraft and offers data rates of 5 Gbit/s.

According to the SpaceFibre standard [1], the nodes in a SpaceFibre network are connected by copper or fibre cables. SpaceFibre is using a point-to-point connection to connect the ports of two nodes. Each connection is called a link and can consist of multiple lanes that allow the nodes to send data in parallel. Each link is a full-duplex which allows simultaneous transmission and reception of packets. An illustration of a SpaceFibre network can be seen in Fig. 2.1. Every port in a SpaceFibre node is running individual iterations of the IP and will use separate controllers in software.

Each link can use up to 32 virtual channels, to send data through. Using these channels, the node can sort the data already at reception by the hardware. Each virtual channel is assigned a bandwidth, a timeslot, and a priority to ensure that important communication channels get the needed space on the link. SpaceFibre uses the same packet format as SpaceWire to simplify bridging to older SpaceWire components. The format is suggested as destination address, a cargo and an end of packet (EOP), or an error end of packet (EEP) marker if an error occurred during transmission. During transmission and reception, the standard makes no difference between the destination address and the cargo, this is up to the receiving application to interpret. The maximum length of the packet is not set by the standard but is implementation-specific.

The broadcast channel is used for communicating with all nodes in the network. When a broadcast message is received by a routing switch, it is propagated to all ports and is further spread across the network. A network can consist of 256 broadcast channels, though it can consist of even more nodes as not every node needs to be able to send broadcast messages even if they can receive them. During configuration, if the node intends to transmit broadcast messages, a broadcast channel number shall be set. This is used to identify what node the broadcast messages were transmitted from. A node should not change from the broadcast channel number it first used in the network. Broadcast messages are more restricted in their packet format than virtual channel messages. A broadcast message consists of 8 bytes of data.

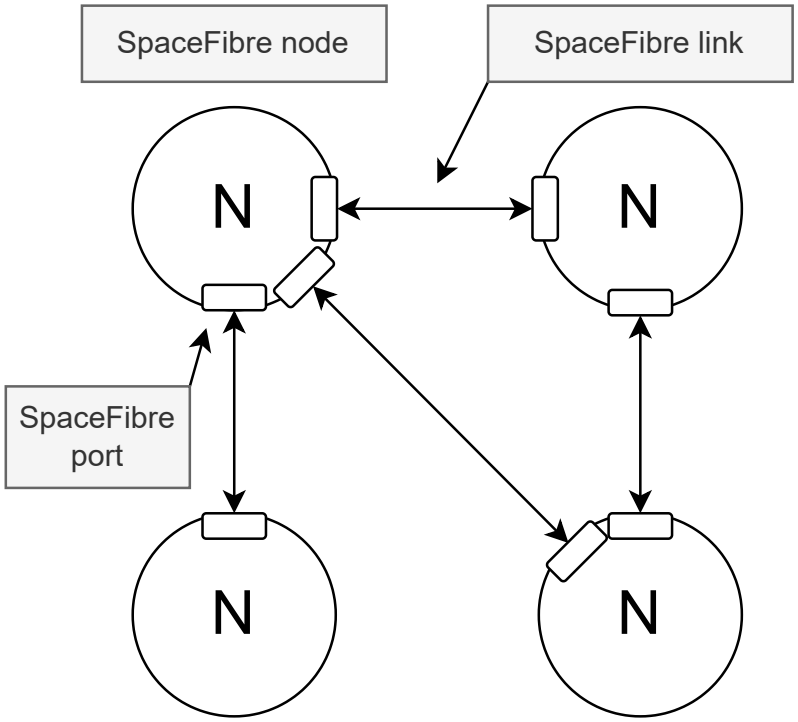


Figure 2.1: Illustration showing 4 SpaceFibre nodes in a SpaceFibre network.

2.1.1 8B/10B encoding

When a signal is sent over a physical wire, the DC bias can drift if the signal contains too many 1s or 0s in sequence, resulting in bit errors. Using 8-bit/10-bit encoding can prevent this from occurring by encoding the 8-bit message as 10-bits where no more than five of the same bit value is sent in sequence. Doing this will in the long run result in the number of ones and zeroes transmitted being 50%. Because of this, it will also offer enough state transitions to base clock recovery on. This is a method used in many communication protocols to avoid errors and is also used in SpaceFibre to make communication systems onboard spacecraft more fault tolerant. When transmitting 8 bytes, first a D/K bit is added, this shows the receiving codec whether this is data or control characters. These 9 bits are then encoded into a 10-bit symbol. This will make the communication more fault tolerant, but it will also slow down the data transfer rate as more bits are required to transfer the data. The data that is to be transferred make up 80% of the symbol. In a link with a bit rate 2.5 Gbps, this will result in a 2 Gbps theoretical data transfer rate.

2.1.2 Direct Memory Access

During a traditional memory operation, the processor core could potentially be busy for a long time handling the memory operation. It would also take longer as all the data had to pass through the processor first before reaching its destination. However using a DMA engine can allow a piece of hardware to access the memory to write or read, without involving the processor. In the case of SpaceFibre, the DMA engine allows the data being received on the link to be immediately written to the memory

or read from the memory in the case of transmission. Using a DMA engine in a time-critical system can be really important as otherwise, any transmission or reception would occupy the processor core and keep it from doing anything else.

2.2 SpaceWire

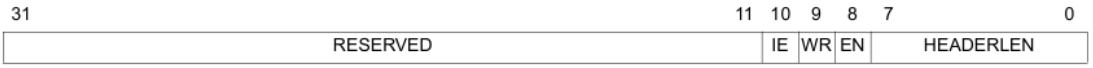
SpaceWire is one of the standards for onboard communication in spacecraft in 2022. It is the predecessor of SpaceFibre and works in similar ways. It offers data rates of 2 Mbps to 200 Mbps on a full-duplex connection, called a link, according to the specification [6]. A major functional difference for the software and user between SpaceWire and SpaceFibre is that SpaceWire does not by default utilize QoS functions such as virtual channels and prioritizing. When SpaceWire-D [7] was introduced they added timeslots to improve the QoS and make it more deterministic. There are also implementations of SpaceWire where virtual channels do exist however since this is not standardized it might not work properly when combining equipment from multiple manufacturers. A major difference with the hardware is the physical cable. SpaceWire does not use optical fibre cables and can therefore not reach the same link speed as SpaceFibre. Similarly to SpaceFibre, there is no set packet format to make it simple to customize for the application. The current implementation of the SpaceWire standard in Cobham Gaisler's products reaches a write rate of 159.81 Mbps and read rate of 159.99 Mbps, according to their benchmarking[8] using a link rate of 200 Mbps. However, they can operate with a link frequency of 400 MHz, leading to an effective measured data transfer rate of 320 Mbps. Their system frequency lies at 250 MHz and uses buses with a width of 32 to 128 bits which leads to a maximum internal rate of up to 8 Gbps. This could indicate that the speed of the communication would be a bottleneck.

2.3 The SpaceFibre IP

LEON5 is an implementation of the Sparc architecture. The hardware used in this report is a prototype of a partial GR765 processor implemented in a Xilinx FPGA and which includes one LEON5 core. As explained in Cobham Gaisler's manual [9] the SpaceFibre IP developed by Cobham Gaisler used in this project functions by writing packets to descriptors. Descriptors are structures in memory informing the SpaceFibre codec about the message to be transmitted or received. During transmission, the descriptor informs the codec of the length and placement of the header and data to be sent. During reception, it informs the codec of where to place the data and gives a space to inform the user of the length of the packet and other flags such as error flags. The SpaceFibre core steps through these descriptors in order as it transmits and receives packets. Therefore it's required that they are placed in a contiguous area in memory. During the initialisation of the SpaceFibre core, before any packets can be transmitted or received, the descriptors' memory space is allocated by the user. The size of this descriptor table is set to fit the number of descriptors set when synthesizing the IP and has to align with the size of the table. The number of transmission descriptors can be 64, 128, 256, or 512. Reception

descriptors are half the size of transmission descriptors, as will be explained later in the thesis, and the number of them can be 128, 256, 512, or 1024.

GRSPFI TX descriptor word 0 (offset 0x0)

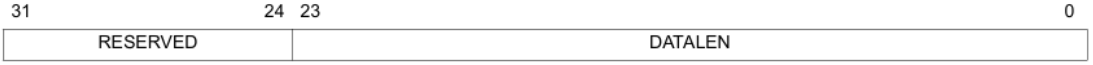


31: 11	RESERVED
10	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
9	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
8	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The core clears this bit when the transmission has finished.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

GRSPFI TX descriptor word 1 (offset 0x4)

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

GRSPFI TX descriptor word 2 (offset 0x8)



31: 24	RESERVED
23: 0	Data length (DATALEN) - Length in bytes of data part of packet. If set to zero, no data will be sent. If both data and header lengths are set to zero no packet will be sent.

GRSPFI TX descriptor word 3 (offset 0xC)

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

Figure 2.2: The bit representation of each word in a transmission descriptor.

The size of one transmission descriptor is 16 bytes and the contents can be seen in Fig. 2.2. When a packet is to be sent the following steps are to be taken. First, the driver selects a descriptor and enters the address to the header and/or data to be sent and the number of bytes to send for each of them. Most of the time this will be the next descriptor pointed to by the hardware register, as the codec steps through the table in order. This will continue until it reaches the end, at which point it will go back to the beginning of the table. Multiple descriptors can be prepared, but

they still need to be in sequence on the descriptor table. If the length is 0 for either of them the respective header or data will not be sent. After the packet has been added to a descriptor the driver sets the enable bit of the descriptor to indicate that it is ready to be sent. When a descriptor has been enabled a bit in a control register has to be set to show the codec that there are descriptors available for sending. The codec will then read the descriptor the hardware register points to, and send the number of bytes the length indicates from the addresses given. The descriptor enable bit will be cleared by the hardware when the transmission is finished to indicate that the descriptor is ready to be used again. After transmission of that descriptor, the hardware register will increment to point to the next descriptor. The codec will keep sending any enabled descriptors until it reaches a descriptor not enabled, it will then clear the descriptor available bit and set the transmission finished bit instead. The

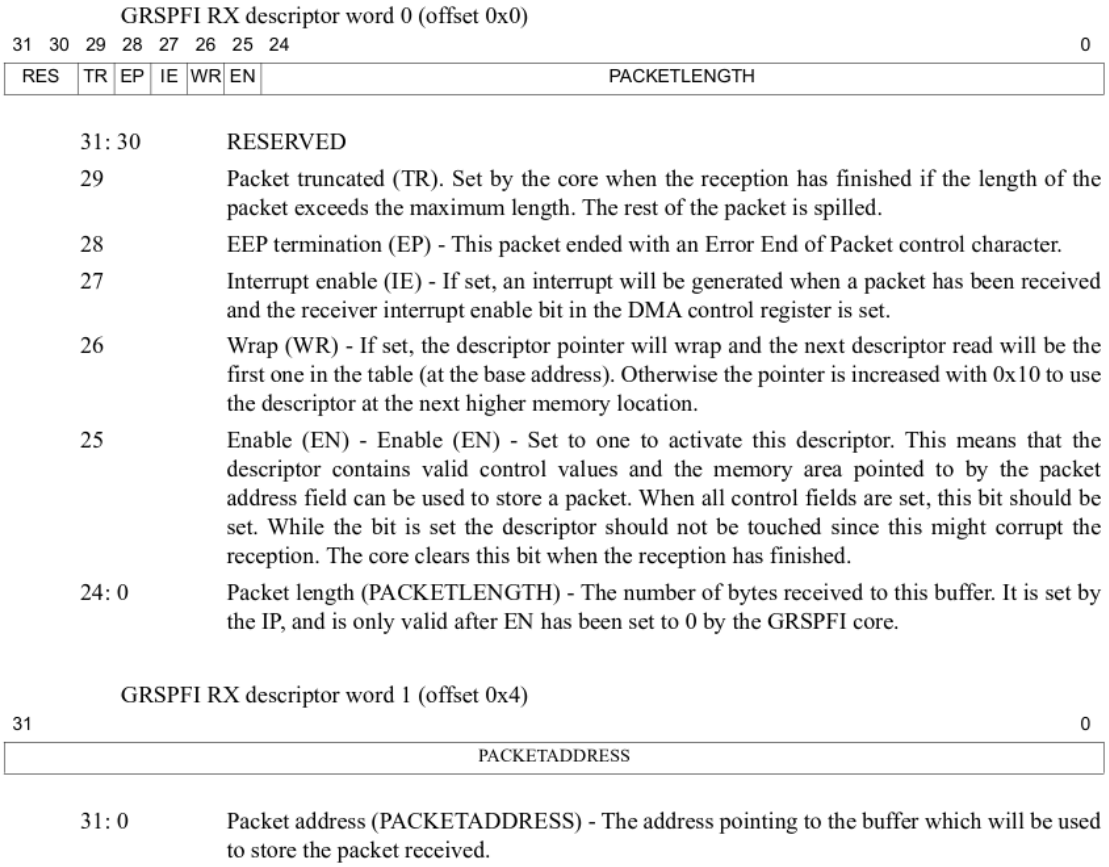


Figure 2.3: The bit representation of each word in a reception descriptor.

size of the reception descriptor is 8 bytes, as can be seen in Fig. 2.3. In contrast to transmission descriptors, reception descriptors are set up in preparation to receive. Empty memory space is created and the address is written to the packet address field and the descriptor enable bit is set. When a packet is received the codec checks the reception descriptor it's currently pointing to see if it's available for reception. If it is, the codec stores the message at that location, set the packet length to the number of bytes the received message was and increment the descriptor pointer to point to the next descriptor.

3
Design

3.1 Low-level API

The first step of developing the driver is to implement all the low-level APIs interacting with the hardware. These are developed for a Bare-C Cross-Compiler 2 (BCC2). The APIs are developed to be scalable and context-independent to allow the user to use them as needed. The routines should produce the same result no matter the circumstances during which they are called, as long as there's room in the descriptor table.

3.1.1 Initialisation

To enable and start using the SpaceFibre core it first has to be set up and initialised. While the order is not vital, each of the steps needs to be performed before the link can be used for communication.
One step is to configure the core by writing the settings to the control registers necessary. Such as the control registers to set up the timeslots and bandwidth of the virtual channels.
Another important step to set up the core is to map the available, up to 32, virtual channels, that are intended to be utilised, to the, up to eight DMA engines. Then to use them the DMA engines need to be enabled for transmission and/or reception. Both of these things are done by configuring hardware registers.
A third step in the initialisation consists of initialising the lane itself. This can be done actively by setting the start lane bit in the relevant hardware register, but also passively by setting the autostart bit in the same register. When using the autostart mode the port will instead wait until the other side of the link tries to initialise the link. By this time it can be a good idea to reset all the status bits as nothing should trigger them to be set immediately again.
The final step before the core is set up to transmit and receive packets is to create a memory area for the descriptors. This needs to be allocated beforehand as the SpaceFibre core will cycle through them automatically without the processors' input. The size of the area needed is known from the capability register stating how many descriptors the core is configured for and the known size of each of them. A smaller memory area is okay, as long as the wrap flag is set in the final descriptor telling the codec to go back to the start again, instead of continuing to the final descriptor.
When all these steps are done, the SpaceFibre core is ready to communicate with another SpaceFibre-enabled node on the other side of the link.

3.1.2 Virtual channels

The virtual channels in a SpaceFibre link act as a way to sort the data using the channels as tags, offering a method to sort the data already by the sending node. To implement QoS each virtual channel can be given a number of timeslots, in which that channel can transmit over the SpaceFibre link. If multiple channels are assigned the same timeslot each channel is also assigned a priority and a bandwidth reservation to make sure there is no bus-hogging. All of these parameters for QoS are configured in hardware registers. The routine should function the same way regardless of which virtual channel is being written to. The virtual channel should be an input to the routines to make it easy for the user to select which virtual channel should be used and to make it independent of the hardware implementation, rather than having individual routines for each of the virtual channels. Using additional routines for each virtual channel would bloat the driver with functions the hardware might not even have access to, since it is configurable how many virtual channels an implementation of the IP core has.

The two main groups of routines needed to utilise the SpaceFibre core is transmission and reception. The reception routines will be two: readying descriptors to be used, and reading the descriptors that have received a message and stored it in memory. The second group is the transmission routines, which are used to transmit messages. The transmission routines are just like the reception routines divided into two: one routine to schedule messages to be sent, and one to reclaim used and sent descriptors to be used again. Both of the mentioned groups of routines will be further described in the following sections.

3.1.2.1 Transmission

Cobham Gaisler's hardware implementation of the SpaceFibre standard uses a memory area with descriptors. The memory address of the first descriptor works as the base address and then the remaining descriptors are offset from that address. As can be seen in Chap. 2.3, each transmission descriptor consists of 4 words. The first word contains flags and the length of the header. The second word contains the address to where the header is stored. The third word contains the length of the data. The fourth word contains the address to the data. In a descriptor, the length of the header or the length of the data can be 0, but not both. If both the length of the header and the length of the data are set to 0, no message will be sent, as there is nothing to send. A message can be sent using only the header or only the data, but also using both. However when using both the user should be aware that the protocol fetches the header separately from the data, which will create extra overhead, rather than just using one of them.

As illustrated in Fig. 3.1, There is a descriptor pointer which contains the ID of the next descriptor to be written to, and also a counter indicating how many more descriptors are available for use. Another structure exists, the packet pointer, which contains the ID of the next packet to write to and a counter for how many more packets are available. The number of packets is not defined in the standard but can be defined in the software.

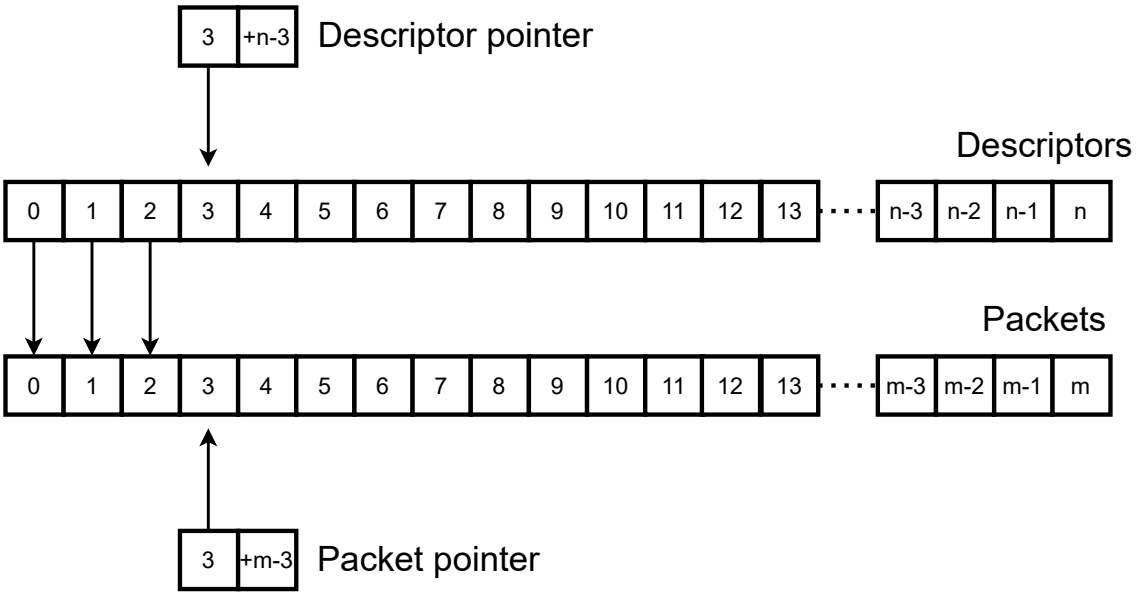


Figure 3.1: The data structure is used to reference and keep track of packets in the descriptors. n can be up to 1024 for transmission and m is defined in the software.

When a packet is to be sent, it is written to the packet that is pointed to by the packet pointer and the packet counter is decreased. The packet is then sent to the driver-API. First of all the driver checks if there are any free descriptors by making sure the descriptor counter is not 0. The driver then finds the descriptor through the ID in the descriptor pointer and writes the length of the header and data to the descriptors. It then adds the address to the header and data in the respective descriptor word. When the packet is written to a descriptor, the descriptor enable bit is set, and the descriptor available bit in the assigned DMA control register is also set. this is to indicate to the codec that there are new messages to be sent. The user also needs to call on a driver-API to reclaim used descriptors and clear them from any old data to prevent it from interfering with future transmissions.

3.1.2.2 Reception

The implementation of SpaceFibre uses the same technique for reception as for transmission. The routines for reception are divided into two routines: one for readying the descriptors for receiving data, and one for checking the descriptors if any data has been received. Before any messages can be received the driver has to enable the descriptors for reception. This is done by the ready routine. It is setting the relevant flags, that were introduced in Chap. 2.3. This includes writing the address of an empty packet in the packet address word, enabling the descriptor and setting the descriptor available bit in the assigned DMA control register.
The packets need to be prepared beforehand and allocated memory space. The maximum number of bytes a packet can receive needs to be determined before creating the packets to make sure they can fit the incoming packets. To make sure the incoming packets will not overflow into non-allocated memory if they are larger than expected, the maximum number of bytes can be set in a hardware register. When an incoming message surpasses this limit it sets the error end of package

(EEP) and truncated flag of the descriptor to let the user know the content is not reliable. The number of packets created is independent of the number of descriptors. The user can accept data in packets and read them at a pace determined by the user while readying new descriptors again with other packets. The user can also create fewer packets than there are descriptors. However, when doing this all descriptors cannot be readied as there are not enough packets. The driver does not care about what packets are provided or where they are. When a message has been received the reading routine will check the descriptor if the enable bit has been cleared and if there's a data length written in the descriptor. If that is the case it will return the length of the data received to be used when reading the packet in the list.
The descriptor pointer in this case contains the ID of the next descriptor to read and check if it contains a package and a counter for how many descriptors are available to be readied again. The packet pointer contains the ID of the next packet to be enabled for reception and a counter for how many packets are not enabled and are waiting to be used. Any packets received will be located at the end of this counter.

4
Methods

In this chapter, the tests used to benchmark the system and driver will be explained. It will start by presenting the tools used and then continue by presenting each of the tests.

4.1 Tools

Two tools were used to benchmark the system, the FPGA running a prototype of a partial GR765 with a LEON5 core and a STAR Fire Mk3.

4.1.1 FPGA

A XILINX FPGA with a prototype, partial implementation of the GR765 was used to run the driver prototype. It includes a subset of the GR765 functionality to focus on the performance of the GRSPFI. In Fig. 4.1 a block diagram of the design can be seen. In this prototype design, there's a Xilinx memory controller, in the finished GR765, this will instead be replaced by a DDR3 controller with Error Detection And Correction (EDAC). Additionally, the GR765 will contain a level 2 cache before the memory controller among other units. In the diagram, the SpaceFibre core is located in the GRSPFI. Beyond this lies the codec and the Serialiser and deserialiser (SERDES). Of which the latter is responsible for translating the digital data into a serialised signal to be transmitted on the cable and the other way around for incoming signals. The path the data takes when being received is from the GRSPFI straight to the Xilinx memory controller through the AHB bus, instead of travelling via the processor. This is because the GRSPFI has a DMA, and can access the memory independently from the processor. When the descriptors are modified this is done from the processor cores and thus does not have to pass through the GRSPFI, The codec will fetch the information from the descriptors in memory as they are needed.
The AHB bus in the block diagram is an AMBA AHB bus running on the system frequency 100 MHz and has a 128 bit width.

4.1.2 STAR Fire Mk3

To benchmark the software design, a tool called STAR Fire Mk3[10], developed by STAR-Dundee was used. It can be connected through the SpaceFibre port and controlled through the software on the connected computer. The STAR Fire Mk3

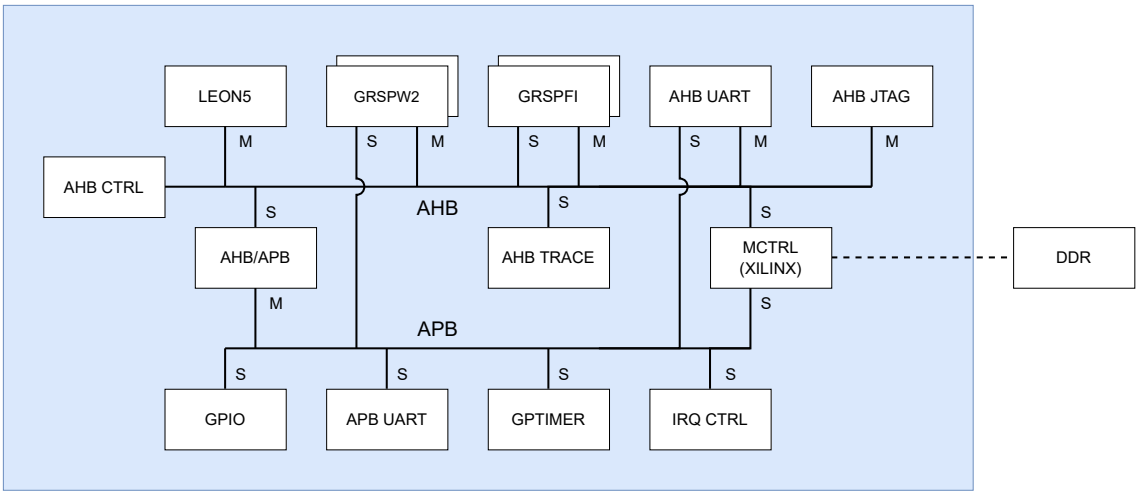


Figure 4.1: A block diagram of the hardware architecture of the prototype design.

features built-in hardware to generate and check data in real-time to prevent computer latency from affecting the execution. Through the software STAR-Fire Mk3 Controller and STAR-Fire Mk3 Statistics, the STAR-Fire Mk3 can be controlled and used to gather statistics. On the other end was the FPGA with a partial prototype GR765 including a LEON5[2] processor and a SpaceFibre core.

4.1.3 Test setup

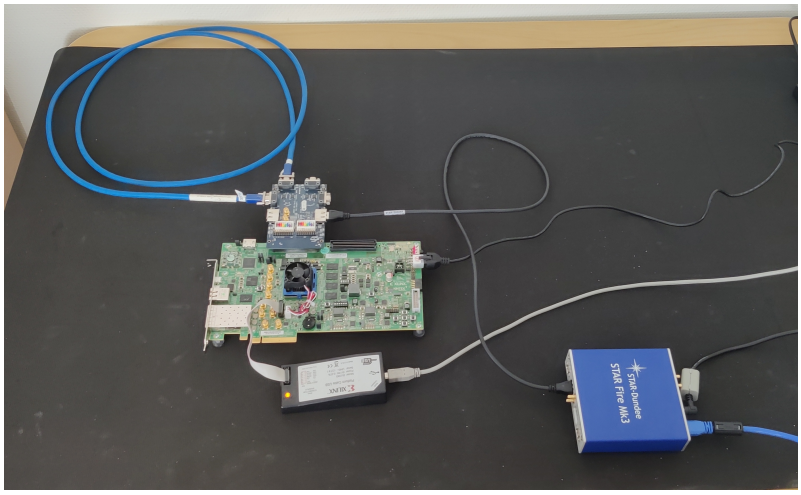


Figure 4.2: The test setup used, with the FPGA board, STAR Fire Mk3 and XILINX adapter.

During the tests the setup in Fig. 4.2 was used, using the XILINX adapter to program the FPGA. The FPGA board was running the driver prototype on a partial GR765 processor prototype. The STAR Fire Mk3 was used to transmit packets to, and receive from the FPGA.
As illustrated in Fig. 4.3 the PC communicated with the Xilinx adapter using the in-house debugging interface GRMON[11]. The Xilinx adapter then in turn commu-

nicates with the FPGA through JTAG, enabling the user to download software, run and debug it by reading and writing to the registers and memory. Using the STAR fire GUI the user can connect to the STAR Fire Mk3 and set it up to communicate with the FPGA over SpaceFibre as if it was another unit in the network.

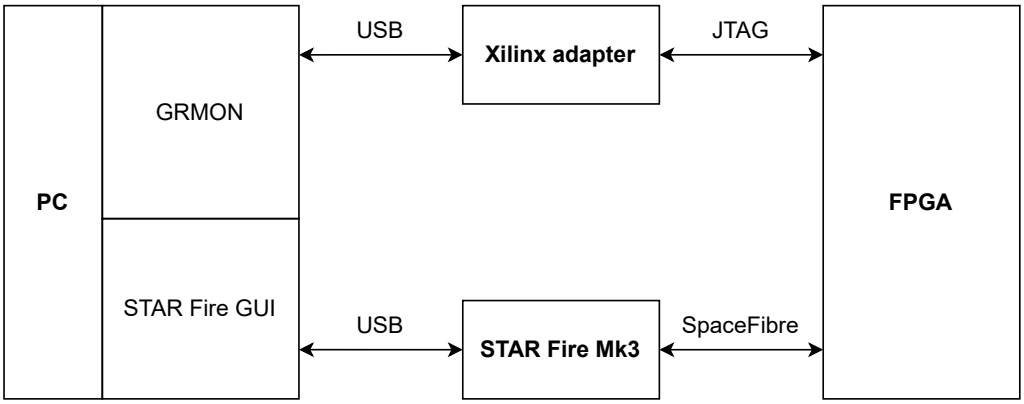


Figure 4.3: An illustration of the test setup with the software interfaces and physical interfaces.

4.2 Latency

To determine the latency of the driver a test was set up where the driver routines were repeatedly called and the number of clock cycles was recorded before and after calling the routines. To get a span of how long the driver actually takes, multiple tests were done where the number of times in a loop the routine was called was varied. The number of clock cycles was then divided by the number of times the routine was called and an average number of clock cycles was found. The STAR Fire Mk3 was used to fill the available reception descriptors and receive the transmission packet during the tests.
To get the time for running only one of the routines some limitations had to be set. For reception there were only 512 descriptors, so the time could only be measured for readying the 512 descriptors. For reading them they had to have received a packet first, and then the time was measured for reading the 512 descriptors. During the calling of the readying and reading routines, the reception of packets was turned off in the codec. This was done to reduce the delay caused by DMA operations. The flowchart illustrating this test can be seen in Fig. 4.4.
The test was similarly constructed for transmission, however, for transmission, only 256 descriptors were available. First, the time was measured to prepare 256 descriptors for sending. The packets were then sent and the time was measured to reclaim them again after all of them had been sent. Again the transmission of the packets was turned off during the time the sending and reclaiming routines were being called, to stop the DMA from interfering. The flowchart illustrating this test can be seen in Fig. 4.5.
One important point in these tests was to not allow the codec operations to interrupt the routines on the bus by transmitting and receiving packets. This was done by

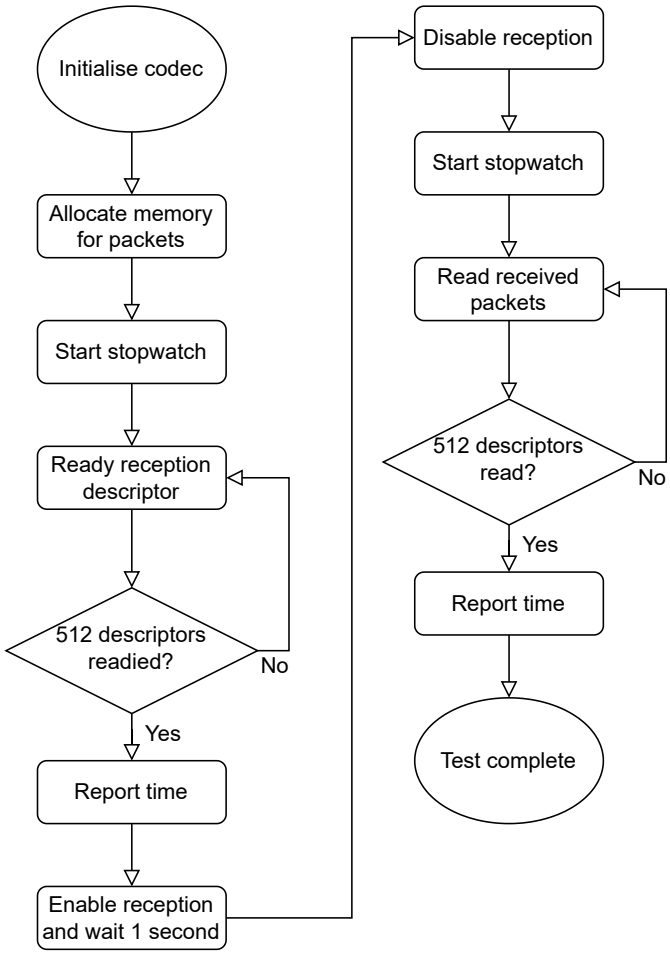


Figure 4.4: A flowchart illustrating the reception latency test.

disabling the codec during the tests and enabling it again in between to allow it to transmit or receive messages on the descriptors.

4.3 Bitrate

To determine the bitrate The STAR-Fire Mk3 was again used to transmit and receive messages to and from the FPGA board running the SpaceFibre driver.

For this test, the STAR Fire Mk3 and the FPGA running the partial GR765 prototype will bombard each other with messages to read out the data-transfer frequency. Because of the flow control properties of SpaceFibre messages will never be sent faster than they can be received. This test will be performed in two steps: one where the STAR Fire Mk3 transmits packets to the FPGA, and one where the FPGA transmits packets to the STAR Fire Mk3. This is to avoid the writing and reading of packets competing over the bus, even though the link is full-duplex.

Important to have in mind is that the signalling rate of the link is 2.5 Gbps but with the 8b/10b encoding, the theoretically highest speed is 80% of that, i.e. 2 Gbps.

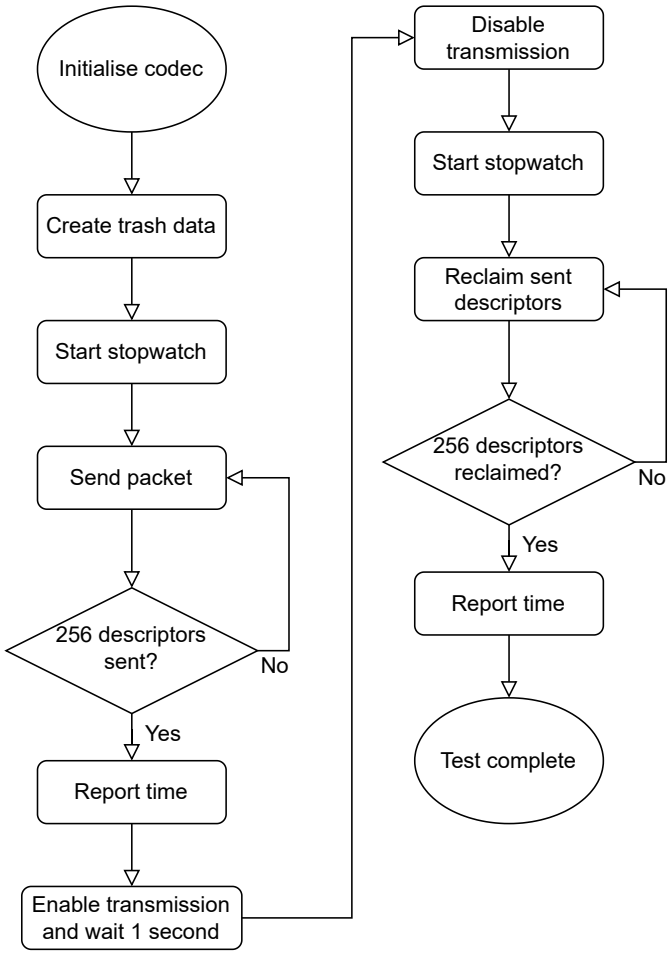


Figure 4.5: A flowchart illustrating the transmission latency test.

4.3.1 Reception

To measure the bitrate of reception a testing software as illustrated in Fig. 4.6 was set up to initiate the SpaceFibre core and then run in a loop. During each cycle of the loop checking for incoming messages and readying new descriptors to receive messages through. To reduce the impact of the memory speed an area in memory was allocated the size of a packet and then this area was reused for all the packets. In each run of the loop, the driver routines to read incoming packets were called, and the information was trashed as that is irrelevant as long as no error flags were set. A packet was then initiated with the memory area and used to call the ready routine. The STAR-Fire Mk3 was set up to transmit packets with random data with a data signalling rate of 2.5 Gbit/s. The built-in random data generator is used to avoid any latencies caused by the slow USB3 connection to the PC. A counter is also set up to record how many messages are being received.

4.3.2 Transmission

To measure the bitrate of transmission, just as for reception, the testing software as illustrated in Fig. 4.7 was set up to initiate the SpaceFibre core, and then run in a

loop. Sending messages and reclaiming messages that have been sent each cycle of the loop. As in reception, the STAR Fire Mk3 was set up, but this time to receive messages instead. Because the memory latency should have minimal impact the same data was sent in every packet. As the content of the packets did not matter, trash data was used. To create the trash data memory was allocated with the size of the packet. This memory was then without being cleared sent as is. This way the data to be sent could be created before entering the loop. In the loop the reclaim routine was first called, to reclaim any successfully sent packets. then the data was assigned to a packet and sent using the send routine. In this test, only the data section of the packet was used, To not create additional overhead. As headers didn't exist in SpaceWire, using only the data will give a more comparable result.

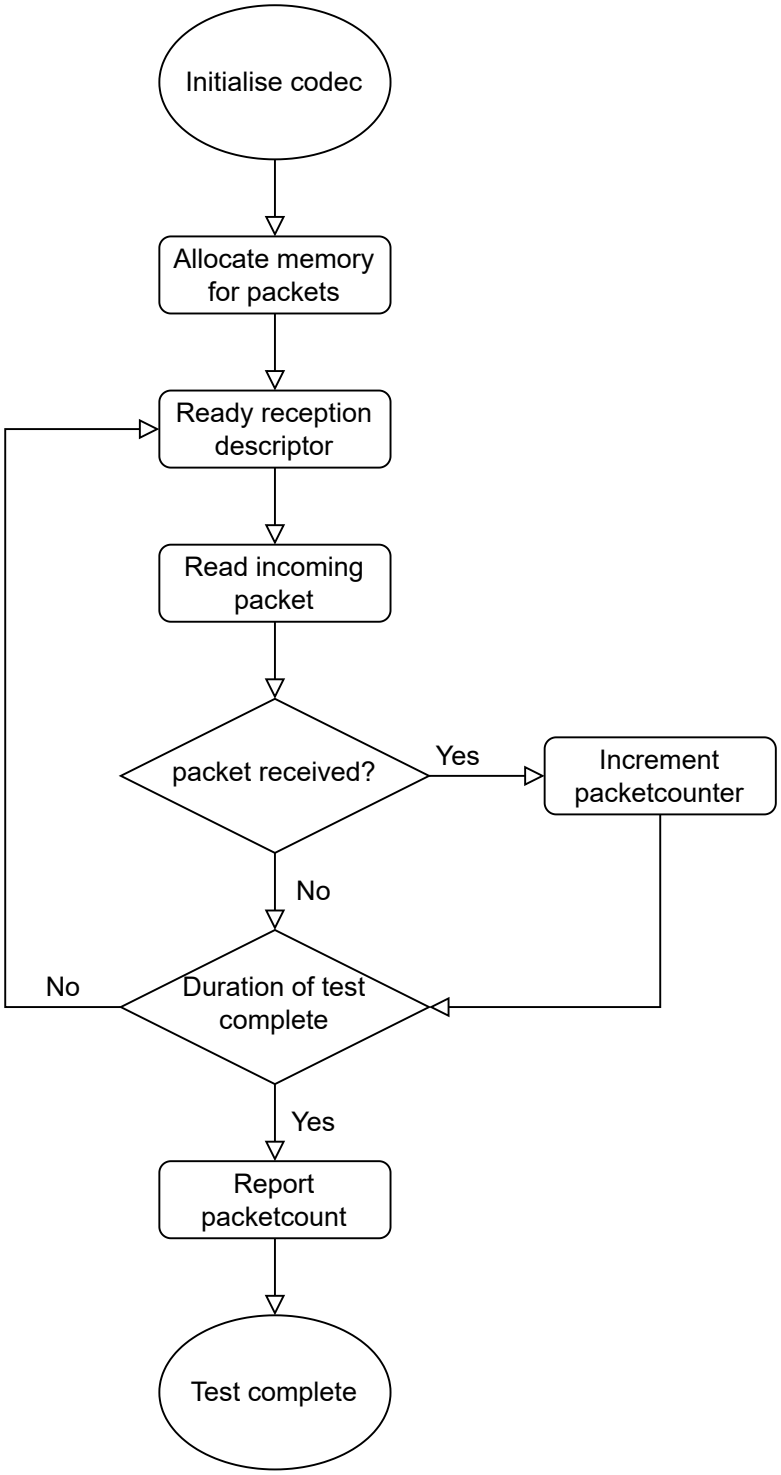


Figure 4.6: A flowchart illustrating the reception bitrate test.

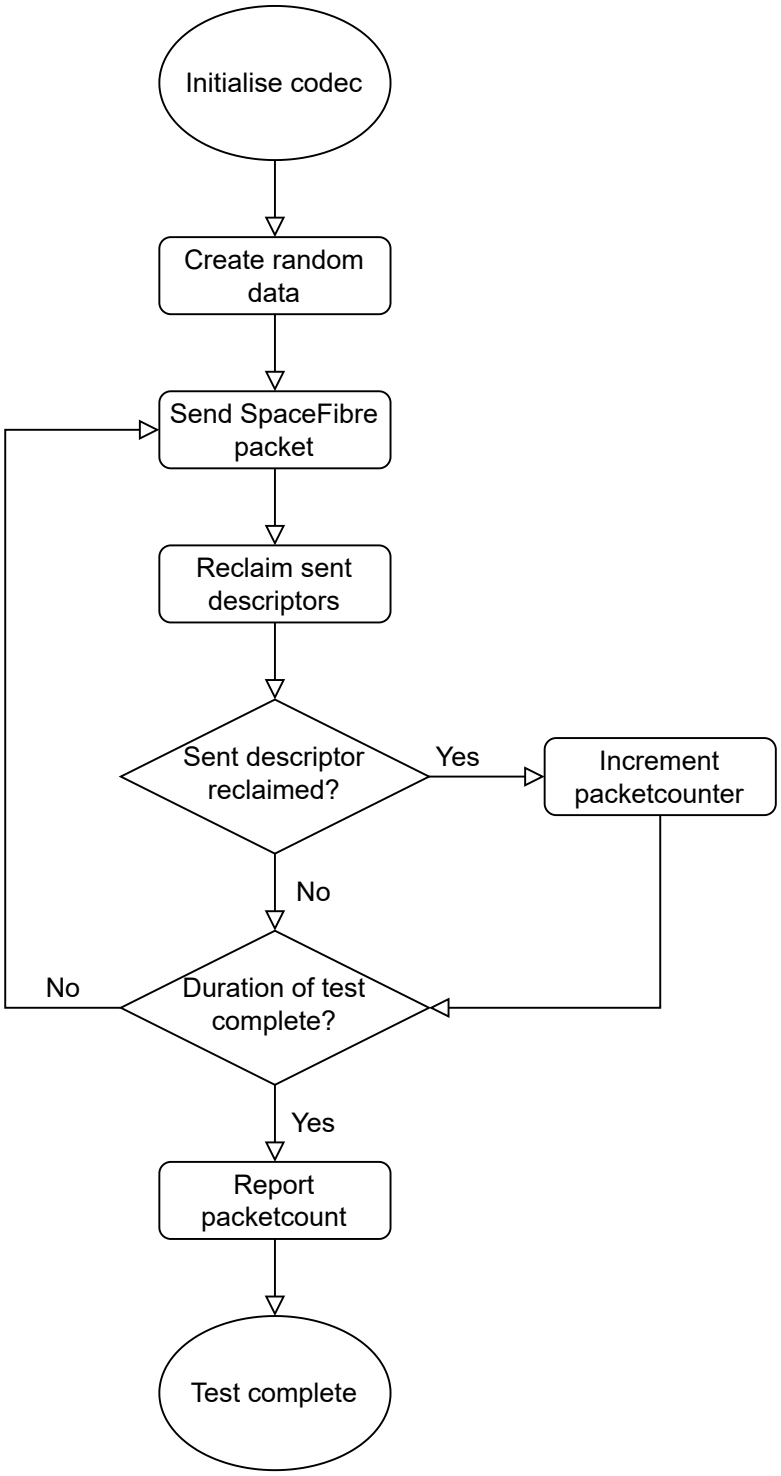


Figure 4.7: A flowchart illustrating the transmission bitrate test.

5
Results

In this chapter the results from the tests described in Chap. 4 are presented.

5.1 Latency

After performing the tests described in Sec. 4.2 the results for each of the driver routines are stated in Tab. 5.1. As can be seen, the transmission routines take an increased amount of clock cycles compared to the respective reception routines. This makes sense because the transmission descriptors consist of 16 bytes, while the reception descriptors consist of 8 bytes. The AMBA protocol allows bus widths of 8, 16, 32, 64, 128, 256, 512, and 1024 bits. The implementation in the FPGA has a bus width of 128 bits. After a quick read test by trying to read from the memory and looking at the bus traffic, the number of clock cycles required for reading 32 bits is about 47 clock cycles. Reading up to 128 bits in the prototype using the AMBA protocol should take about as long, but as one word in the descriptors is read at a time and not all at once this is not fully taken advantage of.

Reception		Transmission	
Ready	Read	Send	Reclaim
105 cc	84 cc	139 cc	102 cc
1.05 us	0.84 us	1.39 us	1.02 us

Table 5.1: The number of clock cycles and microseconds for each of the driver routines on a 100MHz system.

When readying the descriptor it reads the first word to check if it's already enabled. This can be done by looking at the descriptor enable bit, and the data length bits, neither of the bits should be set. Reading a word from memory takes about 47 clock cycles. Writing to memory however is not as easily predictable, it depends on the cache coherency protocol and if the data is requested immediately after. When only writing to memory the processor does not care about when it is done, it will continue executing the next instructions as soon as it is able to, as it does not need to wait for the data being stored into memory. When readying a descriptor, the routine will write the relevant flags to the first word, the enable bit, and then the address to store the data in the second word of the descriptor.
When reading a packet the routine will load the first word of the descriptor to check if it is no longer enabled and if there is a data length stored, this again takes 47 clock cycles. If it is disabled and a data length is found, the routine will return the

length of the data from the routine. To prepare for being readied again and avoid any potential errors, the routine will also clear all bits of the descriptor.
The sending routine takes significantly longer than both of the reception routines. When running the sending routine it will read both the first and the third word in the descriptor. Each word requiring 47 clock cycles leads to at least 94 clock cycles, and then additional cycles are spent on the logic of the driver and interruptions on the shared bus. The first descriptor word contains the descriptor enable bit and the length of the header, and the third descriptor word contains the length of the data. If a descriptor is to be used for sending it should be empty in all bit places. If the descriptor is free, the routine will write to all four words in the descriptor. The driver is zero-copy so the execution time of the routines is independent of the size of the packet being transmitted. The routine does not know whether the user wants to transmit a header, data, or both. It will copy the address to both the header and the data into the descriptor and write the length of both the header and the data. If the length of either is zero it will not transmit that part of the packet.
The reclaim routine is similar to the read routine, it will check that the descriptor is disabled and that there is a length written to it. In the case of the reclaim routine, it will check both the header length and the data length. Reading the two relevant words, the first and the third word requires just like the send routine two reads that require about 47 clock cycles each. If the packet in the descriptor has been sent the routine will clear all the bits in the descriptor to make it clear that it is ready for new packets.

5.2 Bitrate

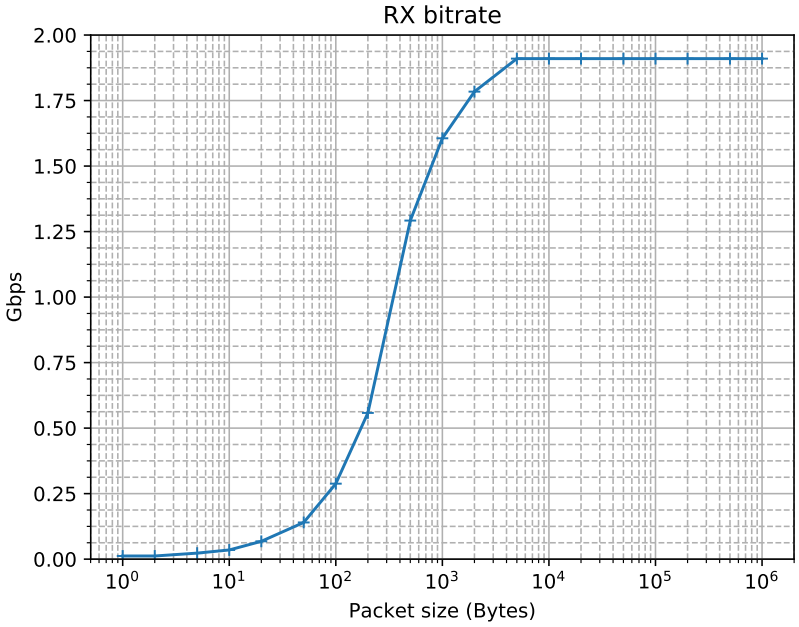


Figure 5.1: The result from the bombardment reception test.

The results from tests described in Sec. 4.3 are plotted on a graph with the package size on the x-axis and the resulting bitrate on the y-axis. The resulting plots can be seen in Fig. 5.1 and Fig. 5.2. As can be seen in both figures the bit-rate starts off low, around 0.01 Gbps. At this point the Packets are so small the DMA is quickly transmitting or receiving the packets to the descriptors as they are made available. As the packet size increases, at 20 Bytes, the transfer rate starts to increase. It plateaus when it reaches a packet size where it spends the majority of the time sending the data rather than handling descriptors, which is by 5000 Bytes. The result that the transmission bitrate falls behind the reception bitrate is expected, as transmission descriptors are twice the size of reception descriptors fetching them takes twice as long, and with the AMBA protocol reading takes longer than writing, as first, a read request has to be transmitted before the memory starts sending data to be read. As opposed to when writing, where a write request is followed by the data to be written and sent on the bus and immediately written to the memory.
As can be seen in Fig. 5.2 the graph with the transmission bitrate starts to plateau at 1.5 Gbps when the package size reaches 5000 Byte. As opposed to the reception bitrate which reaches the speed of 1.9 Gbps before it reaches its peak. This is likely caused by the hardware and not by the processor core hogging the bus during the loop. This is confirmed by adding a delay in the loop to stop the processor from handling descriptors when the queue is full, to prevent the processor core from accessing the memory as often and leave the bus to the IP core. This however only leads to slightly faster bitrates but still about 85% of the reception bitrate. The delay of 100 clock cycles was chosen with regard to the results in Tab. 5.1, as that is

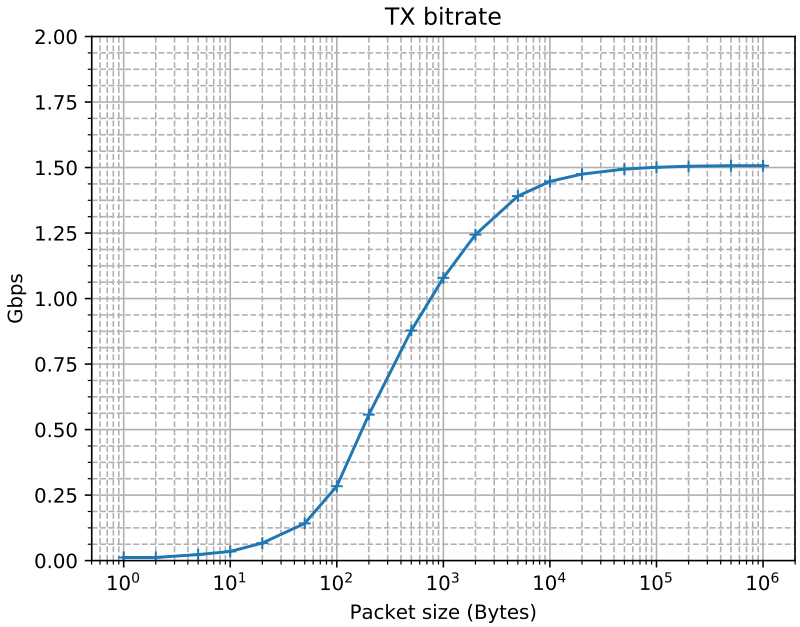


Figure 5.2: The result from the bombardment transmission test.

roughly the length of one run of the routine. This delay was then multiplied to see if additional time would increase the performance. To ensure that the delay itself causes minimal impact on the execution of the test, an assembly routine, counting the number of clock cycles in the processor core was used. With the total number of clock cycles taken by the send routine, the reclaim routine and a few more for the loop logic, 100 clock cycles were selected as the delay in the loop. The packet size 100 000 Bytes was selected as it is when the core has reached peak performance.
As can be seen in Tab. 5.2 the bitrate does increase a bit without the interference of the processor, as would be expected as the processor and the SpaceFibre core share the same bus. However, it still does not reach the theoretical maximum transfer speed.

Delay: 100×	0	1	2	3	4
Bitrate:	1.501	1.594	1.594	1.594	1.594
Delay: 100×	5	6	7	8	9
Bitrate:	1.595	1.595	1.595	1.595	1.596
Delay: 100×	10	100	1000	10000	100000
Bitrate:	1.596	1.596	1.596	1.595	1.595

Table 5.2: The bitrate when adding delays to the loop during transmission test.

As can be seen in Fig. 5.3 and Fig. 5.4. The number of packets sent per second is relatively unchanged in the beginning, it only drops by 13%. This is because the packets are so small the DMA quickly writes or reads the data before the next descriptor is made available. The driver does not copy the data to be transmitted or was received. no matter the size of the packets, the driver always handles 16 bytes

for transmission, and 8 bytes for reception. After 100 Bytes, the transmission and reception start to take longer than an iteration of the loop and the packetrate starts to drop. As can be seen, when comparing Fig. 5.4 and Fig. 5.3 the packetrate for transmission is slower than for reception, similar to how the bitrate is also slower for transmission. This is possibly caused by the fact that transmission descriptors are twice as big as reception descriptors and as can be seen in Chap. 5.1 the routine takes nearly twice as many clock cycles to perform. This also makes the processor core utilize the busses twice as much.

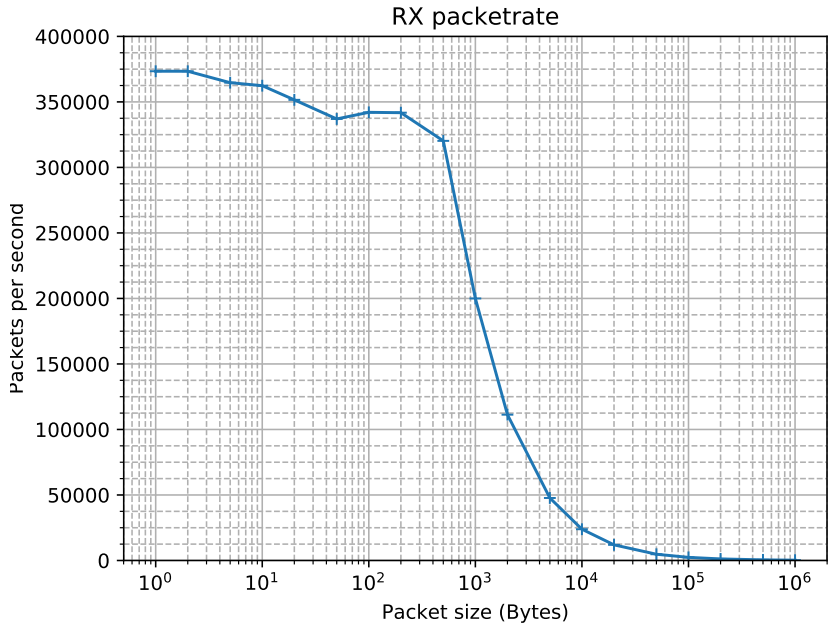


Figure 5.3: The number of packets sent per second during the bombardment reception test.

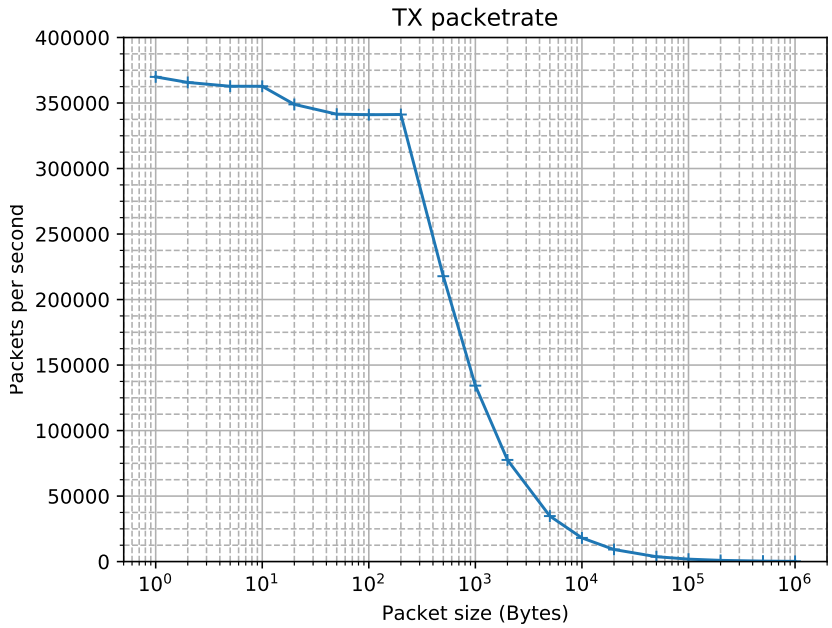


Figure 5.4: The number of packets sent per second during the bombardment transmission test.

6
Discussion

In this chapter, the results from the previous chapter and their implications will be discussed. Limitations and potential improvements will be presented.

6.1 Latency

As can be read in Chap. 5.1 the latency of the routines makes sense considering the amount of data being read from memory.
Having a fast reception routine can be advantageous. If the user is periodically polling the SpaceFibre device for incoming messages, having a fast routine to check for incoming messages will not hog the processor. In the same way, if interrupts are used, having a fast routine will keep it from causing too much overhead whenever a packet is received. The handling of the data is not included in the latency of the routines. However, as this is a zero-copy driver and there can be more packets than descriptors, receiving a packet in one descriptor does not mean the driver stops until that data is handled. It can be readied with another packet and that packet that has been received can be handled at the appropriate for the application timeframe.
Having a slower transmission routine than reception will cause the driver to perform worse during transmission. This is difficult to avoid as the transmission descriptor is twice the size of the reception descriptor, and reading data on an AMBA AHB bus is slower than writing. However, the user has more control over when to call for the transmission routines. As previously mentioned the driver is zero-copy and the packet to be transmitted has to be prepared beforehand. Transmitting large packets will cause an equal amount of latency as transmitting small packets.

6.2 Datarate

As can be seen in Fig. 5.1 and Fig. 5.2 the transmit speed depends a lot on the size of the packets being sent. Small packets will take a longer time to transfer, with larger packets with a packet size of between 5000 Bytes and 1 000 000 Bytes the datarate will plateau and remain constant. Comparing this to the results seen in Fig. 5.3 and Fig. 5.4, using smaller packets allows the driver to transmit multiple packets faster. Larger packets will require more time to transmit by the codec. The increased time to transmit will prevent the driver from enabling more descriptors as the descriptor table will be filled with enabled descriptors.
As the increased packet size causes the packetrate to drop significantly, if there is vital data to be transmitted periodically, that should be assigned a different

virtual channel. this would allow the packets of that channel, depending on the QoS parameters, to send their packets interleaved with the virtual channel busy with large packets.

6.3 Summary

With a fixed latency independent of the size of the packets, both for reception and transmission the driver offers predictable results. Whereas, for small packets, a majority of the time will be spent preparing descriptors. For large packets, the latency of the routine will be only a fraction of the time of reading and writing packets to and from memory by the codec.

6.4 Hardware

When during the development of the driver the hardware was still in the prototype stage. There was no other implementation of the Space Fibre core available than a soft-core on a prototyping FPGA. The implementation of the IP in the FPGA was delayed and there was no hardware to test the driver on until halfway through the project when there were a few hundred lines of code to debug. In the end, the data structures were rewritten for easier debugging and were made into a more straightforward functional driver.
Another obstacle that halted the development for a period was a bug in the IP that caused the core not to clear its buffers. If the descriptor pointer was changed manually during runtime the core wouldn't update the data address even after a reset. Because of this the codec save or fetch the data from the previous descriptor pointed to. This was updated after finding so that the buffers would be cleared during a restart of the core.
The strange behaviour during the transmission where the bitrate suddenly plateaus earlier than expected was a great cause for confusion, adding delays in the routine did not significantly improve the behaviour. The cause for this behaviour is at the moment of writing not found and investigating it fully could possibly be enough work for another thesis.

6.4.1 Hardware limitations

The hardware was operating at the frequency of 100 MHz and had a bus width of 128 bits. This frequency is significantly lower than the goal for future use in the GR765 board, which will be running at a frequency of 1 GHz. In a different hardware configuration with a higher clock frequency, the maximum speed would be higher. The rise in transfer speed would also probably occur earlier as the processor would be faster at preparing many new packets. This is because the bottleneck during reception and transmission of small packets is the speed at which the CPU can supply the codec with newly enabled descriptors.
Another limitation of the prototype design was that it was single-core. The upcoming GR765 will be octa-core allowing the preparation of many more packets at a

time. The processor cores would be able to prepare new packets for transmission in parallel and enable them. It would be able to handle the data in the incoming packets at the same time as it prepares new descriptors to receive data on.
As the SpaceFibre link can only transmit and receive one packet at a time, the use of multicore will not raise the maximum bitrate. However, it would allow the descriptor tables to fill up faster and saturate the link even with smaller packets.

6.5 Future research

For future research, it would be interesting to implement an algorithm to let packets of certain sizes bypass the level 2 cache to avoid the bottleneck of cache misses, which will inevitably happen when the size of the packet is larger than what fits in the cache.
Another suggestion is to investigate how to use the width of the AMBA AHB bus to the driver's advantage when doing memory operations. In the prototype FPGA and the GR765, the width of the AMBA AHB bus is 128 bits, which is equal to the length of the larger transmission descriptor. If used optimally the bus would allow the driver to read the entire descriptor in one read. The same thing can be applied when writing to the descriptors. Even though writing to memory using the AMBA AHB bus is significantly faster than reading, doing multiple write operations, instead of one will take longer. This would lower the latency of the driver and allow for more descriptors to be prepared in the same amount of time. However, this would mainly have an impact on small packets as that's where the preparation of the descriptors is the bottleneck.
Making sure that the driver will still function in a multicore system and not suffer from race conditions as the processor cores are writing to the same packet or descriptor will also be work for future improvements.

7
Conclusion

After designing the driver and testing it with the SpaceFibre core developed by Cobham Gaisler the driver has been proven to work.
The data structure chosen for the driver has turned out to be reliable and keeps track of how many free packets and descriptors are available. It has been functioning adequately both during reception and transmission, during a saturated link and during an underutilized link.
A bug was found in the core that was fixed during the course of this thesis, where the IP core prefetched the descriptors and didn't reset when the address of the descriptor table was modified. this caused the core to write and fetch data to and from the wrong address. Had it not been found it could have caused errors and lost packets for customers. The core throttles the bitrate during transmission. However, a cause for this has not yet been found.
Even though the results did not reach the full theoretical maximum of 2 GHz it is significantly faster than the previously used standard SpaceWire which reached a maximum of 159.81 Mbps when writing and 159.99 Mbps when reading. In comparison, the driver developed in this thesis reached a maximum of 1596 Mbps when transmitting and 1910 Mbps when receiving, even in this demonstrator prototype.
Unlocking this increased bitrate for onboard communication will further increase the amount of data to be collected and might further increase the knowledge of space.

Bibliography

[1] SpaceFibre - Very high-speed serial link, ECSS-E-ST-50-11C, May 2019. [Online]. Available: https://ecss.nl/standard/ecss-e-st-50-11c-spacefibre-very-high-speed-serial-link/.

[2] Cobham Gaisler AB, "GR765 Octa-Core Processor," [Online]. Available: https://www.gaisler.com/index.php/products/components/gr765, (accessed: 25.01.2022).

[3] Cobham Gaisler AB, "Processors," [Online]. Available: https://gaisler.com/index.php/products/processors, (accessed: 25.01.2022).

[4] Cobham Gaisler AB, "GRLIB IP Library," [Online]. Available: https://www.gaisler.com/index.php/products/ipcores/soclibrary, (accessed: 25.01.2022).

[5] ESA, "Spacefibre," [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Data_Processing/SpaceFibre, (accessed: 14.06.2022).

[6] SpaceWire - Links, nodes, routers and networks, ECSS-E-ST-50-12C, May 2019. [Online]. Available: https://ecss.nl/standard/ecss-e-st-50-12c-rev-1-spacewire-links-nodes-routers-and-networks-15-may-2019/.

[7] STAR-Dundee, "SpaceWire-D: Deterministic Data Delivery over SpaceWire," [Online]. Available: https://www.star-dundee.com/wp-content/star_uploads/conference_papers/spacewire/2014_DASIA_SpaceWire-D_Deterministic_Data_Delivery_SpaceWire.pdf, (accessed: 12.09.2022).

[8] Cobham Gaisler AB, "GR740 Technical Note on Benchmarking and Validation," [Online]. Available: https://www.gaisler.com/doc/gr740/GR740-VALT-0010.pdf, (accessed: 20.06.2022).

[9] Cobham Gaisler AB, *Grlib ip core user's manual*, (2022). [Online]. Available: https://www.gaisler.com/products/grlib/grip.pdf.

[10] STAR-Dundee, "STAR Fire Mk3," [Online]. Available: https://www.star-dundee.com/products/star-fire-mk3/#product_features, (accessed: 12.08.2022).

[11] Cobham Gaisler AB, "GRMON3," [Online]. Available: https://www.gaisler.com/index.php/products/debug-tools/grmon3, (accessed: 18.10.2022).