MASTER THESIS

GUARANTEED-THROUGHPUT IMPROVEMENT TECHNIQUES FOR CONNECTIONLESS RING NETWORKS

Guus Kuiper

DEPARTMENT OF ELECTRICAL ENGINEERING, MATHEMATHICS AND COMPUTER SCIENCE
COMPUTER ARCHITECTURES FOR EMBEDDED SYSTEMS

EXAMINATION COMMITTEE
Prof. dr. ir. M.J.G. Bekooij
Prof. dr. ir. G.J.M. Smit
Dr. ir. J.F. Broenink
B.H.J. Dekens, M.Sc.

UNIVERSITY OF TWENTE.

21-10-2013

Guaranteed-Throughput Improvement Techniques for Connectionless Ring Networks

Master's Thesis
by

Guus Kuiper
Student number: s0142530

Committee:

Prof. dr. ir. M.J.G. Bekooij (CAES)
Prof. dr. ir. G.J.M. Smit (CAES)
Dr. ir. J.F. Broenink (RaM)
B.H.J. Dekens, M.Sc. (CAES)

Research Group Computer Architecture for Embedded Systems, Department of EEMCS
University of Twente, Enschede, The Netherlands
October 24, 2013

Abstract

Increasing the computational power of future System-on-Chips (SoCs) is not possible by increasing the frequency of a processor, because power consumption will become a major issue. Energy efficient systems increase the number of cores to reach higher performance levels to form a multi-core embedded system: a Multi-Processor System-on-Chip (MPSoC). This flexible type of system can be customized for specific applications by changing the software running on it.

At the University of Twente, a prototype MPSoC has been developed specifically for real-time applications, like Software-Defined Radios (SDRs). This class of applications place strict requirements: of a guaranteed minimum bandwidth and a maximum bound on the latency of communication between tasks running on different cores. Communication within the platform is performed over a connectionless Network-on-Chip (NoC), which should allow a small implementation. However, strict guarantees are harder to accomplish in a connectionless network compared to a connection-oriented network.

The original implementation of the ring network can only provide a very low guaranteed bandwidth, because its guarantees are not based on any knowledge of the traffic on the ring. In this work, two types of traffic have been identified on the ring for the software First In, First Out (FIFO) buffers connecting the tasks running on the platform: data and credit traffic. Data traffic requires a high throughput over a short distance and a low latency, whereas credit traffic requires a low throughput over a long distance and is less dependent on latency. A potential issue arises, because the same communication guarantees are offered to both traffic types, which requires an over-allocation of bandwidth, leading to a low Guaranteed Service (GS) bandwidth.

The objective of this research is to define techniques to prevent this over-allocation. Simultaneously, the strengths of a connectionless ring network should be maintained; a small hardware implementation scaling (almost) linear to the number of ring nodes, and a predictable behavior which can be analyzed using data-flow models.

We implemented a separation of these two types of traffic within the connectionless ring network. By doing so, different guarantees can be offered by the ring for data and credit traffic to match their requirements. The separation allows an increase in guaranteed data bandwidth by utilizing knowledge of a mapped task graph and the communication channels between these tasks for a small increase in hardware costs. Furthermore it is shown that an abstraction can be made of the ring network improvement techniques in a compact Synchronous Data Flow (SDF) graph.

An overall increase in GS throughput of the software FIFOs is obtained by using two techniques; one for credit and one for data traffic. The bandwidth of the credit traffic is limited by defining a credit period during which only a single credit may be inserted onto the ring, resembling a Polling Server from (real-time system) literature. This leaves a minimum bound on the available bandwidth for data traffic. Based on

the mapping of tasks onto processor tile, data bandwidth is divided on a slot basis and is set in a “slot mask” in every Network Interface (NI). Ring slots can then be re-used by multiple producers to effectively increase the GS throughput.

When comparing this solution to the previous fairness protocol, a large increase in GS throughput of a best-case of 6 MS/s to 50 MS/s can be reached when a large FIFO token size is used for the communication between tasks. A small token size effectively means that the data and credit bandwidth will be close to each other, resulting in a low overall throughput when the credit bandwidth is limited. The implemented improvement techniques result in an increase of about 70% in hardware cost, which is less than adding a second dedicated ring specifically for credit traffic. A significant increase in GS throughput is obtained by only a small increase in hardware costs.

Acknowledgements

First of all I would like to thank Marco, head of my supervising committee, for all the feedback I received, especially late at night when a notification on my phone showed a new e-mail from Marco. I also like the discussions we had about data-flow models and his critical view on my ideas encouraged me to go a step further.

My daily supervisor, Berend, should not be forgotten for providing a basis for my work by designing a predictable ring, of which a paper recently got accepted at a conference. I had a pleasant time with Berend as a roommate for the largest part of my assignment, where I was 'forced' to enjoy his music taste.

Jochem originally developed the Starburst platform and should be credited for creating a platform on which I could easily integrate a ring network design. I could always ask him to help me debug the system when I introduced errors into it.

I would also like to thank Gerald for helping me getting started with this complicated platform and its tooling. He might also have inspired me to start with this assignment during one of our conversation after a floorball practice.

Last but not least I like to thank Nienke, my girlfriend, who spent a lot of time proofreading and thereby correcting most spelling mistakes. Without her help my thesis would have been unreadable.

Guus Kuiper
Enschede, October 2013

Table of Contents

List of Figures

List of Tables

Acronyms

μB MicroBlaze.

ADC Analog to Digital Converter.
AHB Advanced High-performance Bus.
AMBA Advanced Microcontroller Bus Architecture.
APB Advanced Peripheral Bus.
ASIC Application-Specific Integrated Circuit.
AXI Advanced eXtensible Interface.

BE Best Effort.
BRAM Block RAM, dedicated RAM in Virtex-6.

CPU Central Processing Unit.

DAC Digital to Analog Converter.
DDR3 Double Data Rate type 3 SDRAM.
DMA-C Direct Memory Access-Controller.
DSP Digital Signal Processor.
DSP48E1 Digital Signal Processing element, special Virtex-6 slice.
DVB-T Digital Video Broadcasting — Terrestrial.

FIFO First In, First Out.
FMC FPGA Mezzanine Connectors.
FPGA Field-Programmable Gate-Array.
FSL Fast Simplex Link Bus, uni-directional point-to-point hardware FIFO communication link.

GALS Globally Asynchronous Locally Synchronous.
GPS Global Positioning System.
GS Guaranteed Service.

HD High Definition.
HSDF Homogeneous Synchronous Data Flow.

ITRS International Technology Roadmap for Semiconductors.

KPN Kahn Process Network.

LAN Local Area Network.
LMB Local Memory Bus.
LUT Look-Up Table.
LUTRAM LUT RAM, LUTs used as RAM.

MCM Maximum Cycle Mean.

MMU Memory Management Unit.
MPMC Multi-Port Memory Controller.
MPSoC Multi-Processor System-on-Chip.

NI Network Interface.
NoC Network-on-Chip.

PAL Phase Alternating Line.
PLB Processor Local Bus.

RAM Random Access Memory.
RF Radio Frequency.

SDF Synchronous Data Flow.
SDR Software-Defined Radio.
SoC System-on-Chip.
SPM Scratchpad Memory.

TDM Time Division Multiplexing.

VHDL VHSIC Hardware Description Language.

CHAPTER 1

Introduction

1.1 Context

Traditionally the computational power demands would increase every year, just like you would expect the number of transistors to double approximately every two years as Moore's law dictates. The International Technology Roadmap for Semiconductors (ITRS) expects, for the near future, there will come no end to this growth of processing power [27]. In the year 2026, the system processing performance metric will have grown by a factor of 1000 as compared to 2009.

The need for such an increase becomes obvious when looking at consumer electronics; take for example the color television market. Starting (in Europe) with the Phase Alternating Line (PAL) standard, the image quality keeps increasing and evolved towards High Definition (HD) video (720p or 1080p) to reach a major improvement in quality. Bandwidth requirements double when using 3-dimensional technology to supply a different image to both eyes. Improvements do not end there, as more processing power becomes available; standards that use even more pixels are introduced like 4K Ultra HD (four time more pixels than 1080p HDD). More advanced technologies are on its way, such as 8K Ultra HD, again increase the number of pixels by four times[1]. Processing becomes even more demanding when such videos need to be analyzed for surveillance purposes, preferably in real-time. Thus, an increase in computational power can always be used by applications to improve the user experience.

The increase in processing power could in the past be accomplished by scaling in frequency. This was sufficient until the thermal limits were reached quickly, as power requirements scaled faster than frequency. The only way forward was to increase the number of processing cores, despite of the (even currently unsolved) problem of how to program such multi-core designs. This trend of adding more cores will continue according to the predictions of the ITRS; "The number of cores increases by a factor of 1.4 per year" [27].

Again looking at video context, recent developments in video codecs have picked up this trend towards multi-core systems. The current most commonly used compression standard H.264 (Advanced Video Coding) is hardly supporting parallel

[1] Recently, this technology has been premiered. On the 17th of may Sky-Skan Inc. in association with The Franklin showed "Institute To Space And Back"; an 8K x 8K, 60FPS, 3D film running approximately 25 minutes [3]

processing of the same operation within a single frame. The proposed successor, H.265 (also called "High Efficiency Video Coding"), however, has clever tricks that make it (more) suitable to run on multi-core systems. "The option to partition a picture into rectangular regions called tiles has been specified. (...) Tiles are independently decodable regions of a picture that are encoded with some shared header information" [46]. These tiles can inherently be processed on different cores, allowing decoders to be better mapped to future many-core hardware architectures.

These applications need to be integrated into embedded systems, like mobile phones, to -for example- record and encode HD video. Already, most functionality is integrated into a single chip: a SoC. The next logic step is to move towards multiple processors to fulfill the unique requirements of embedded applications; so the Multi-Processor System-on-Chip (MPSoC) [52] was introduced. These MPSoCs are ideal platforms for semiconductor manufacturers, because they can be specialized into a number of products, mainly by customizing the software that runs on them.

Two types of MPSoCs exist, each with their own (dis)advantages: homogeneous and heterogeneous MPSoCs. Homogeneous ones are easier to program and more flexible, whereas heterogeneous MPSoCs contain specific hardware accelerators which improve power efficiency for certain applications and are also used to meet the performance of more demanding applications, like processing modern wireless standards. The goal of previous work was to integrate accelerators into a homogeneous MPSoC [26] to create a heterogeneous MPSoC, which is capable of accelerating DSP algorithms like PAL video to perform at commercial speeds, while retaining the same system clock speed.

Moving from single to multiple processors in a system imposes some new architectural challenges. It is not possible in such a system to obtain a low main memory latency in combination with many processors using the same shared memory. All kinds of issues relating to caches (coherency) [40], synchronization (locking) [38] and communication (inter-core or core-to memory) [39] arise in MPSoCs. A research platform is developed at the University of Twente to address these kinds of topics.

Communication is crucial for the overall performance of an MPSoC and will be further introduced in Section 1.2. The real-time requirements of embedded applications running on SoC / MPSoCs will be the topic of Section 1.3. A special feature of the research platform, a connectionless interconnect and the (dis)advantages of this connectionless approach are discussed in Section 1.4. After the introductions to (connectionless) embedded communication infrastructures and real-time requirements, the problem statement of this thesis will be described in Section 1.5.

1.2 Embedded Communication Infrastructures

Since communication within a SoC is an important aspect of a chips performance, a short history will be presented in this section. In the past, every system used a bus as interconnect, then crossbars followed, and currently research is focusing on Network-on-Chips (NoCs). The reasons for the shift from buses to more advanced architecture will be described here.

When systems only consisted of a single processor and a very few peripherals buses were used to connect them. A bus is a shared interconnect which originated from

outside chips. Only one entity, a master, can transport data at a time. When multiple masters are present, they need to be granted access to the bus from a central arbiter. An example of such a bus is given in Figure 1.1, where a decoupled view is given, which separates the architecture of the interconnect (the gray box in the middle) from the masters and slaves. A widely used bus architecture in the industry is the ARM Advanced Microcontroller Bus Architecture (AMBA), consisting of a high speed processor bus, the Advanced High-performance Bus (AHB), and a low speed peripheral bus, the Advanced Peripheral Bus (APB), that can be accessed from the AHB through a bridge.

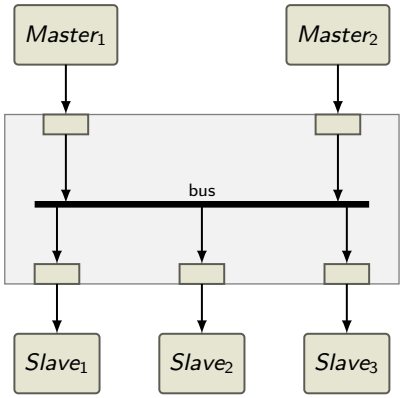

Figure 1.1: Example of a (decoupled) bus interconnect

A bus favors one-to-one communication where one master is active at the time, as the arbitration otherwise stalls bus access requests. Power efficiency is not very high as a master must drive the total length of a bus to be able to reach all peripherals. When moving towards multiprocessor design the bandwidth of a bus is quickly saturated. The advantages of a bus are that it is easy to implement and latencies to peripherals are uniform [8].

Unlike in a bus architecture, where one master broadcasts messages over the interconnect, a crossbar is designed to transmit multiple messages simultaneously. A crossbar, like the ARM Advanced eXtensible Interface (AXI), can connect each of its inputs to any of the outputs. This is implemented as a grid of wires where the inputs cross the outputs and a connection is made by shorting a crosspoint. As can be seen in Figure 1.2, arbitration in a crossbar is only required when multiple masters address the same slave, which has a latency advantage compared to buses. Because multiple inputs are present, this design also scaled better with the number of masters. The scaling, however, comes at a high price penalty, because it results in a large die area.

As the trends are to include more and more processing cores inside a SoC, there is a need for an infrastructure that scales well, is power efficient and supports communication between all these cores. A whole network should be integrated into a chip, an NoC. These NoCs consist of Network Interfaces (NIs), routers and links, as can be seen in Figure 1.3. Links are the physical connections (in the form of wires) between routers. The routers forward data from an input to one of their outputs. They contain a crossbar, and also a buffer that is at least the size of a message that can be received during one clock cycle.

The architecture of a NoC is characterized by three attributes: the topology, routing

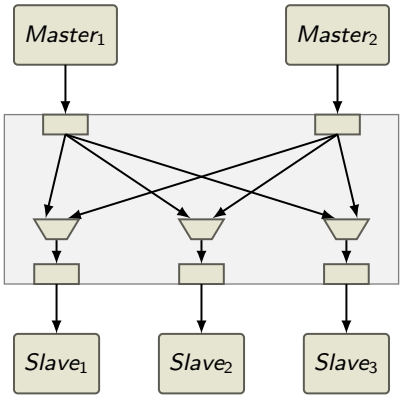

Figure 1.2: Example of a (decoupled) crossbar interconnect

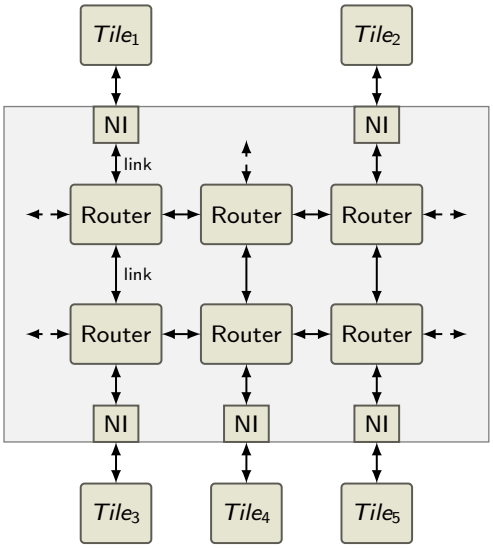

Figure 1.3: Example of a NoC

algorithm and type of flow control [53]. The topology of a NoC describes how routers are connected to each other and it determines how many (in/output) links a router has, which has a direct influence on how messages can be transported across the network; a part of the routing algorithm. The routing algorithm must ensures a path is formed from source to destination and each data producer receives a share of the total network bandwidth.

The last characteristic, the flow control, specifies when messages are transported between routers. Two types of flow control exist: circuit and packet switched. The main difference between them is the size of the buffers that are required in the routers. A circuit switched network first establishes an end-to-end path from source to destination before data is inserted into the network, which requires only small buffers. As a consequence, those links cannot be used in other paths until the path is deallocated. The latency for such an approach is thus relatively high, because the path needs to be set-up and overlapping paths need to be deallocated first. In packet switched networks, routers can store data for multiple cycles. Flow control is then based on the number of free places in the buffer of the next router along the path of a packet. Reducing latency comes at the price of a much larger hardware

implementation caused by the need for large buffers.

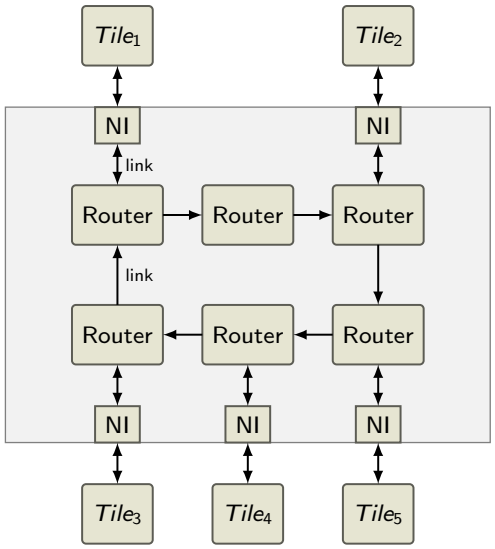

Figure 1.4: Example of a ring interconnect

A special NoC case that deserves additional attention, is a ring topology. A (unidirectional) ring only has a single in- and output, next to the interface with a processing tile. The routing algorithm used in a ring is trivial, since data can only be forwarded in one direction. As a consequence, no large crossbars are required and a minimal amount of buffers is needed (only to store data only for 1 cycle to meet timing constraints of the design). The last NoC characteristic, flow control, is easy to implement at the interface by means of a Time Division Multiplexing (TDM) schedule and using processing tiles that always accept incoming data. It is therefore a very simple and small interconnect.

A lot of research has been performed on all of these three characteristics; topology, routing algorithm and type of flow control. Some implementations, relevant to this thesis, will be discussed in the related work (Section 2.1). Next, the real-time requirements of embedded applications running on processors with such interconnects are covered.

1.3 Real-time requirements

Given the above described context, the type of applications that is targeted is introduced in this section. This type has specific requirements, and services that should be provided by the system it runs on. Then, it is discussed how such a system can be analyzed. It finishes with the consequences and requirements for NoCs that where part of the previous section.

Two classes of processing application can be distinguished: data processing dominated, and control dominated [8]. Many signal processing tasks are part of the data processing class, which is characterized as a sequence of operations that are performed on a stream of data with little to no data reuse. Processing can be performed in parallel, requires high throughput and performance, and favors designs with many processing elements. The other class (control dominated applications)

does have a high data reuse and contains a large amount of states. Examples are (de)compression algorithms, or code with many conditional branches. These tasks are difficult to perform in parallel. A clear separation between the two classes is not always possible. Take for example the H.264 video compression standard. It consists of filtering steps that are clearly data processing tasks, whereas the compression belongs to the control dominated class.

The system which is part of this thesis focuses on signal processing tasks. The input of data originate from an Analog to Digital Converter (ADC) and enters the system at a fixed (sample) rate. At the output of the system a Digital to Analog Converter (DAC) or another digital peripheral is present that requires data to be received at a fixed rate. Constraints are put on the processing chain between the in- and output to ensure a finite number of buffers is sufficient, without the chance of losing data, and to provide a correct output. This type of applications fall into the data processing dominated class and will be referred to as "streaming applications" throughout the thesis.

These constraints on the processing chain fall in the category of real-time constraints. The correct behavior of the real-time systems that have to satisfy these constraints, not only depends on the value of the computation, but also on the time at which the results are produced [12]. In the classical view on single-core real-time systems tasks are characterized by a deadline, which is the maximum time in which a task must be completed. Based on the consequences of a real-time task missing a deadline, three task categories can be distinguished. The first one are hard real-time tasks that may have catastrophic consequences when a deadline is exceeded. Firm tasks produce useless results after their deadline; however, this does not damage the real-time system. Finally, for soft tasks, exceeding a deadline causes a performance degradation, but they can still produce some useful results after their deadline.

In a real-time system, software and hardware need to be co-designed to create a overall predictable system. The new view on multi-core real-time systems is not based on deadlines, but on creating a predictable system which should provide guarantees about a maximum latency and a minimum throughput, even under worst-case conditions. Moreover, problems like deadlock and starvation should never occur.

Deadlock is a situation in which two or more users of a shared resource are waiting independently for access to it, which will never occur [12]. It occurs when multiple users have a requirement to use the same two resources in a nested way, but in a different order. At the moment both acquired their first resource at about the same time, deadlock occurs as they cannot acquire their second resource, and thus never finish, locking those resources permanently. The shared resources can, for example, be a number of hardware accelerators. When they are nested, by chaining them together in a signal processing flow, it could cause deadlock to be introduced. It has been shown that deadlock can also be introduced in a deadlock free NoC, by the interaction between a NoC and connected components [22].

Another situation which should be prevented is starvation. It is quite similar to deadlock in a sense that some users never obtain access to a resource and thus do not progress. The difference lies in the fact that other users can get access to this resource. Inside a NoC, starvation can happen when a user is saturating links in the network, preventing others to send or receive data.

By designing a predictable architecture that satisfies these constraints, the system

can be analyzed in such a way that models can be made of it. It is challenging to model execution times of tasks. It is easy to measure upper bounds on the run-time of an operation, however it remains unknown what the actual worst-case execution time is. With these models, it is possible to calculate lower bounds on the performance of the system. The throughput and latency are important characteristics that can be extracted from a model, and can be used to determine if a certain application can run on the platform, or how parameters need to be set in order to guarantee correct behavior. An example of such a parameter is the size of the buffers. By increasing buffer sizes, latencies can be hidden to increase throughput. In the past, many analysis techniques for real-time systems have been developed; an overview of the most important techniques is given as part of the related work (Section 2.3). Moreover, a choice is made there for which one is going to be used throughout this thesis.

When looking at NoCs, a distinction can be made between two type of traffic based on the guarantees that are required for those types [37]. The first one is GS traffic for which the network has to give real-time guarantees: a guaranteed minimum bandwidth and bounded latency. Best Effort (BE) traffic, on the other hand, only requires an average bandwidth instead of a worst-case bandwidth. It is used for non-real-time applications that are also running on the platform and are used to improve the utilization of the system (of for example hardware accelerators) by exploiting the resources that are left over by the GS traffic. Combining service for both types of traffic in a NoC design however worsens its performance to cost ratio [19]. Therefore this work only focuses on GS traffic for streaming applications.

1.4 Connectionless interconnect

The unique feature of the research platform on which this work is performed, is that it has a connectionless interconnect, which is used within an environment that requires the real-time guarantees mentioned in Section 1.3. Usually these kinds of guarantees are only possible in connection-oriented interconnects.

A connection-oriented interconnect is characterized by a connection that needs to be established before data is transmitted. A separate connection needs to be made for every communication channel between all source-destination pairs. The connection ensures a reliable transfer by using flow control and only allowing in-order communication over a fixed route. It allows applications to be analyzed in isolation, where the influence of other applications running on the same platform is bounded. Each channel requires its own buffer to apply flow control; this is quite expensive for large all-to-all communication networks in a MPSoC [39].

Connectionless interconnects on the other hand do not require a separate buffer and setup per connection, making them much more attractive from a hardware point of view. Data units are individually addressed and routed, rather than relying on predetermined connection information. In general this means that no real-time guarantees are possible without a large overhead at an higher layer in the network stack. A connectionless network cannot guarantee no loss of data, error injection, misdelivery, duplication or out-of-order delivery of data.

A high level protocol must be used to support real-time traffic with a guaranteed minimum bandwidth a bounded latency. An example of a protocol that provides

additional services to a lower connectionless unreliable layer (IP) is TCP. Real-time services can be obtained by using a software protocol like the one defined in Section 3.3.2, in combination with predictable (hardware) arbiters that allow access to the interconnect. In such an interconnect "...experimental evidence is provided, showing that replacing a connection-oriented NoC by a connectionless one in a distributed shared memory system reduces the hardware costs and improves the performance" [39].

To illustrate the consequences of using a connectionless interconnect, an example is given which should also indicate a disadvantage of not having separate communication channels. Take the task graph shown in Figure 1.5. In this case there are 5 tasks; one producing data (task 0), one receiving data (task 4), and three tasks in the middle that forward data coming from producer to consumer. These tasks are running on processors connected via a unidirectional ring interconnect, where the number of a task also indicates the order in which they are mapped on processors connected to the ring.

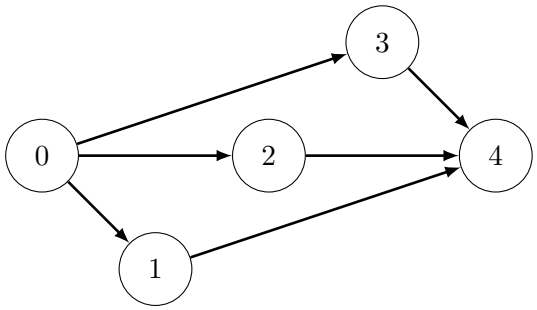

Figure 1.5: Communication pattern between one producer task (0), which send data towards three other task (1-3), which each sent their result to a consuming task (4). The guaranteed bandwidth from producer to each of the three intermediate task must be the same in a connectionless network.

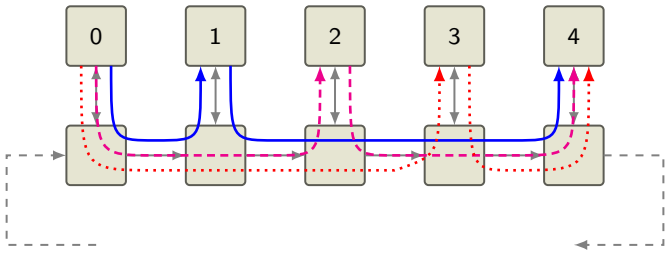

Figure 1.6: Tasks shown in Figure 1.5 mapped onto a ring interconnect to show the communication channels and usage of the ring links

With a connection-oriented interconnect, different characteristics can be specified to different communication channels. It is then possible to split the available bandwidth across the three data paths[2] that are visible in the image, which all use the same physical links. These data paths are also visualized by using different line styles in Figure 1.6 where the tasks have been mapped onto processor tiles. In a connectionless interconnect less bandwidth is available as there is no distinction possible between traffic from for example task 0 to 1 and 0 to 3. Bandwidth allocation must be performed for the total outbound traffic of the producer towards the

[2]The three paths are: $(0 \rightarrow 1 \rightarrow 4)$, $(0 \rightarrow 2 \rightarrow 4)$ and $(0 \rightarrow 3 \rightarrow 4)$

furthest destination (task 3). For example, this means that the bandwidth available at the ring links between task 2 and 3 is not shared by three data streams, but is effectively shared between 5 data streams(0→1, 0→2, 0→3, 1→4, 2→4), the bandwidth requirement for task 2 is extended to the task with the maximum distance from task 0, which is task 3.

This example shows that in a connectionless interconnect, bandwidth needs to be over allocated when there are multiple data streams using the same links in a ring network. A connectionless interconnect however, is much smaller then a connection-oriented interconnect where a buffer is required per connection.

1.5 Problem Description

Now the research context is explored in the previous sections, the main problems of this work will be introduced. It starts by giving a brief description of the research platform that was developed at the University of Twente. Then, the main goals of the platform are stated as an additional requirement next to the real-time requirements mentioned in Section 1.3. Problems and improvements that are observed of the platform are stated next, and are used to formulate research questions. The contributions of this work are summarized in Section 1.6. The chapter finishes by providing an outline of the remainder of this thesis in Section 1.7.

At the University of Twente, a research platform has been developed: Starburst. It is targeted as an experimental next-generation platform for software defined radio purposes and is implemented on a Virtex 6 Field-Programmable Gate-Array (FPGA). An example of an application that is targeted, is a software-based Digital Video Broadcasting — Terrestrial (DVB-T)-decoder; a real-time application with an ADC as periodic source. The goals of the platform are:

- Predictability (firm real-time requirements)
- Composability
- Sufficiently easy programmable
- Low power
- Low hardware costs

Starburst will be described in more details in a chapter that is dedicated to it, Chapter 3. For now, it is sufficient to know that it is a heterogeneous MPSoC, with a distributed shared memory architecture.

Tools are developed on top of the Starburst platform to assist in programming the multiprocessor environment; OIL and Omphale[3]. How these tools are related is illustrated in Figure 1.7.

Applications are written in a sequential programming language, OIL. It is an abstraction layer that hides the issues, mentioned earlier in the introduction, relating to the concurrency model, the memory model and the underlying hardware layer. The language enables concurrency extraction, handling of memory issues, automated model extraction for optimization and mapping of tasks given temporal constraints.

[3]Although Omphale is currently only compatible with an old build of Starburst

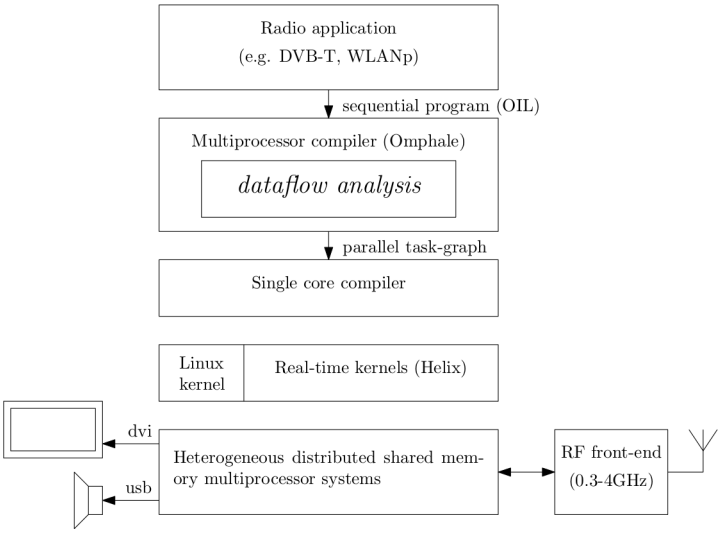

Figure 1.7: Experimental real-time multiprocessor compiler

The sequential program can be compiled with the multiprocessor compiler Omphale [5]. It maps the application to the Starburst platform and analyses if the real-time requirements are met or can be met. This results in a parallel task-graph, which can be used by a regular single core compiler. The tasks can run on one of the processor tiles running a real-time kernel (Helix, see Section 3.3.1) that are described in Section 3.2.1. The last component in the setup of Figure 1.7 is a Linux kernel that is mainly used as an interface to all peripherals.

It is observed that a connectionless interconnect allows a low hardware cost implementation of the Starburst platform. Using the a connection-oriented network Æthereal [19], only 8 soft core MicroBlaze (μB) processors fit into the Virtex-6 FPGA (which not even uses all-to-all communication, but each core is connected to a master core, its two neighbors and the Double Data Rate type 3 SDRAM (DDR3)). A connectionless ring interconnect allows up-to 32 μBs to fit into the same FPGA [39]. When improving the platform it should therefore remain low cost, to have an advantage over connection-oriented interconnects.

The first improvement to the platform is related to the traffic types that are present on the ring NoC. Two distinct types of traffic can be identified on the ring NoC for streaming applications; one type of high throughput short-distance communication, and one low speed over large distances. This property is only true for streaming applications, where consecutive tasks are mapped on processing tiles placed after each other on the unidirectional ring interconnect.

The current communication protocol makes no difference between the two types, such that the same bandwidth and latency guarantees are applicable for both. There, however, is a work-conversing (for a sort of BE traffic) feature build in, which allows bandwidth to be hijacked when certain conditions are satisfied that make sure no negative influence occurs on other data producers. No increase in bandwidth (for GS traffic) is guaranteed and the potential bandwidth increase, decreases with longer communication distances.

Moreover, the communication pattern of streaming data is very regular. Once all processing tasks are mapped onto the platform, the platform is not adjusted to it in

any way. By making use of additional task mapping information or by changing the way tasks are mapped onto the platform, the performance of the whole system can potentially increase. The room for improvements is caused by the ring interconnect which provided the same guarantees for all NIs, regardless of the communication requirements of the task(s) running on the processor that is connected to it. It provides a basis to distribute the bandwidth by lowering it for tasks that have little communication to other cores and allocating it to tasks that do require a lot of bandwidth.

Next to these improvement to the performance of the interconnect, another metric is becoming more and more important. Currently, devices containing multiple processors are becoming small, to a scale where they can be worn, like smart watches. This form factor leaves little space for a battery, leading to strict constrains on the power consumption of the SoC inside the device. Identifying major power consuming parts of the Starburst platform and improving them without loss of the predictability of the system is another interesting research topic.

The final observation of the platform is related to the accelerator integration. As is mentioned in the future work of [26], currently hardware accelerators can only predictably be used by one producer. It however, is desirable to share them among multiple producers to improve the utilization of accelerators, reduce hardware costs and allow more computation intensive applications to be run on the heterogeneous MPSoC. There already are some ideas about how this sharing can be implemented, but before that is implemented, it first need to be carefully analyzed and evaluated to see if it doesn't compromise the real-time guarantees.

From these problems the following research questions are formulated:

- What are the benefits of separating different types of ring traffic on an application level (to remain connectionless and low cost) by making use of the mapping of a task graph?
- How can hardware accelerators be shared by multiple producers in a predictable way?
- How can the power of the hardware architecture of Starburst be reduced without sacrificing the predictably of the platform?

1.6 Contributions

The main contributions of this work are:

- Implementing a (low cost) separation of different types of traffic on the connectionless ring network, to provide different guarantees which match the requirements of the specific types of ring traffic. The separation is made at an application level in contrast to separation in, for example, the destination in a connection-oriented network (Chapter 5).
- Improving the bandwidth of the data traffic type on the ring by making use of the knowledge of a mapped task graph (Chapter 6).
- Identifying issues related to the estimation of the power consumption of specific parts of the MPSoC.

- A compact data-flow analysis model is obtained as an abstraction of the implemented communication ring.

1.7 Thesis Outline

The remainder of this thesis starts by presenting related work in Chapter 2, to place the claimed contribution in the right context. It includes research about NoCs, sharing of accelerators and real-time analysis models. After that the research platform that is used during this work will described in Chapter 3 as it allows a relatively easy prototyping environment for proposed architecture changes.

Next, the focus will be on the problems related to inter-core communication. A more detailed view on the context of the problem is presented in Chapter 4. Then solutions are proposed to these problems and a choice, including details about the implementation, for one of them is made in Chapter 5. It also contains an analysis model for the FIFO communication, which is used in Chapter 6 to evaluate the design. The implemented design is also compared to the previous architecture and tested in a case study, a PAL decoder.

After that a brief look is given into how accelerators can be shared within the Starburst platform. It is therefore part of the future work (Section 7.2) which is included in the final chapter, the conclusion (Chapter 7).

CHAPTER 2

Related work

Now the main topics of this thesis have been introduced in the previous chapter, an overview is given of the research that already has been performed in those research areas. This will show the strengths and weaknesses of the NoC topology, which is used in the Starburst platform and also reveals the current shortcomings, which leaves room for improvements.

We start by addressing the different NoC topologies in Section 2.1, that have gained the attention of many researchers around the world. The differences between these topologies will be explained there. After the introduction to NoCs, the focus will be on different types of ring networks, as a ring is used in the Starburst platform (details about the platform are presented in Chapter 3) and has certain advantages compared to other topologies. An important aspect of a NoC is the real-time guarantees that it provides. NoCs, which provide real-time guarantees are therefore discussed next. The relevance of ring networks is shown in Section 2.1.4 where an overview is given of state-of-the-art chips from the industry, where single or multiple ring networks are used as interconnect between its processing cores.

Within those NoCs, hardware accelerators can be placed to create a heterogeneous MPSoC. Research concerning the integration of accelerators in a NoC is mentioned in Section 2.2.

The last research area that is related to this thesis, is that of real-time analysis models. Section 2.3 gives an overview of different models known from literature. Those models are required to evaluate the performance of a MPSoC consisting of a NoC, and possibly a number of accelerators, and show the limiting factor when running applications on the platform.

Finally, a summary is given in Section 2.4, which indicates the strength of the ring network and from this, it is deduced what the focus of this research should be to make the current architecture even more attractive for integration on certain chips designed for real-time streaming applications.

2.1 Network-on-Chips

A lot of research has been performed at NoCs in recent times. These NoCs can be classified using the three characteristics that are mentioned in Section 1.2 (topology, routing algorithm and type of flow control). All will be treated in this section,

however, the topology will be given the most attention as it is the notable part of the NoC that is used within the research platform.

Many different NoC topologies have been proposed in literature, since the one of the ideas of a NoC is to be free in how processing elements are connected. A non exhaustive list of these topologies consists of: mesh, torus, ring, (fat-)tree, butterfly, octagon and irregular interconnections [2]. This is a logical trend, because no optimal topology has been found. "It is worth noting that no single fixed topology categorically outperforms all others" [42].

The most used topology is a mesh and several researchers have suggested that it is the most efficient in terms of latency and power consumption. That suggestion however is application dependent, and targets random traffic. This work on the other hand, looks at a different traffic type by using streaming applications as workload.

Streaming data, which originates from tasks in a task graph without cycles can favor a different topology. When consecutive tasks are mapped on successive processing elements, it forms a sort of chain of tasks. This should map very well to a ring shaped network. A ring can offer a very low latency, in this case, as there are dedicated (short) links from the previous and towards the next task.

A few survey papers have been published that compare many NoCs based on their characteristics [2, 7, 42]. They observed that about 80% of the investigated NoCs uses packet-switched flow control. The most popular routing algorithm is wormhole routing. Most of the routing algorithms are deterministic, while adaptive algorithms have the advantage of being able to recover from or prevent deadlock situation by changing routes. About 60% of all the examined NoCs apply a mesh or torus topology, while only a very few are ring-based topologies.

As the research platform consists of a ring with guaranteed service, focus of the related work will be on NoCs with a ring topology and / or support for GS.

2.1.1 Ring NoCs

In the past, ring networks have been very popular and were applied in Local Area Networks (LANs). In these kinds of local networks tokens were passed among users that grant permission to transmit onto the ring. In the current era of NoCs, a few researchers are reintroducing the ring as a viable NoC topology. A ring topology gives good performance for specific traffic types [30, 31]. Research focused on the three NoC characteristics; flow control, routing, topology.

Topology

A ring already defines a NoC topology, however, it can be subdivided into more advanced designs: uni/bidirectional and hierarchical designs. Bidirectional designs offer a higher bandwidth and lower latency compared to a unidirectional ring as traffic can be scheduled in the direction of the lowest distance to its destination. In hierarchical rings, multiple local rings are connected by mean of global rings to offer better performance for large networks.

It is for example shown that a hierarchical ring network with a single global ring can suffer from congestion at the global ring due to non-ideal traffic loads [10]. A two-dimensional "hyper ring", consisting of multiple global rings, is shown to perform

better under those loads during simulations. The resource requirements of such a design is still low compared to mesh architectures.

Flow control

Flow control used to be implemented using a token, or by allocation slots. In such a slotted ring, less bandwidth is wasted [4] as multiple messages can be present on the ring in contrast to a single message in a token ring. Users need to wait until a free slot passes with a least the size of the message they want to send.

Nowadays, flow control is usually implemented by controlling the buffers between the sending and receiving NI. For a ring network, flow control can be implemented by only inserting data when is assured that the target NI can accept the data. Data on the ring can always progress in this way; once data is allowed to enter it, the path and delay to its destination are known. Back-pressure is then applied at higher layer protocol based on the state of the consumer, which is fed back to the data producer. The access policy to the ring manages how the bandwidth of the ring is divided across all its user.

When moving towards more advanced ring topologies, like hierarchical rings, flow control become more important. A single ring can never deadlock as a message continues to move closer to its destination. A situation of livelock, where data continues to move but never reaches is destination, is not possible as the destination can always accept data.

A hierarchical ring, however, requires bridges between the global and local rings. These bridges are very prone to introduce deadlock problems when not carefully designed. The bridge buffer can be full, blocking new messages originating from a source ring, a ejection deadlock. On the other hand, a injection deadlock occurs if the destination ring is already full when data needs to be injected. Flow control algorithms to prevent these two forms of deadlock have been proposed, providing a end-to-end delivery guarantee [16, 17].

Routing

Routing in a unidirectional non-hierarchical ring NoC is trivial, as messages can only follow a single path to their destination. It gets more interesting when bidirectional routers are used, as then a choice must be made between two paths.

In the bidirectional cast, routing can be based on the shortest path to a destination. This method allows two messages to arrive simultaneously at an NI, one from each direction. A solution is to for example only accept data from the clockwise ring on the even cycles and from the counter-clockwise ring on odd cycles. The producing NI can calculate the appropriate time at which a message needs to be injected into the ring by making use of the known distance to its destination [29].

Routing mechanisms, which are applied to mesh type NoCs are also applied to more advanced hierarchical rings [10]. The routing of packets is determined by: store-and-forward, virtual cut-through or wormhole routing. The methods have been compared on performance (latency and throughput) and hardware costs [33].

2.1.2 Guaranteed service NoCs

In this section, NoCs will be discussed that are targeted to offer the real-time requirements mentioned in Section 1.3. As there are hardly any GS ring NoCs, other topologies are also mentioned. The following NoCs are treated: Æthereal, MANGO, NOSTRUM and SoCBUS.

Æthereal

Æthereal [37] is designed by Philips to separate computation from communication. To accomplish this, GS should be supported by the NoC where resources are reserved for the worst-case behavior. Because this leads to a low utilization for the average traffic, BE support is added to Æthereal.

BE services are provided by packet switching, where the routing information is present in the header of a (self-contained) packet. Real-time traffic is sent across TDM connections over pipelined circuits, a kinds of circuit switching. The routing is performed by specifying the usage of slots inside a table present in all routers. Inside a router, GS traffic is assigned a high priority, whereas BE uses a low priority to maintain guarantees for real-time traffic.

The design of Æthereal is extensively evaluated after 10 years [19]. It was concluded that BE support is expensive. Utilization of a NoC is not relevant, but the performance to cost ratio is; which is worse when BE traffic is included. Later versions of the NoC changed to use virtual-circuits, where packets contain a routing header, to remove the need for large slot tables.

MANGO

MANGO is a clockless NoC, which is targeted for Globally Asynchronous Locally Synchronous (GALS) SoCs [6]. It supports connectionless BE traffic as well as connection-oriented GS by allocating a sequence of virtual channels across the network. Latency and rate guarantees are not inversely coupled like in TDM-based scheduling schemes, but are managed by a scheme called *Asynchronous Latency Guarantees*.

NOSTRUM

The NOSTRUM NoC introduces two concepts to provide guaranteed services; *Looped Containers* (LC) and *Temporally Disjoint Networks* (TDN) [32]. TDN are used to separate traffic and making sure packets never collide, and are similar to the principle of slots in Æthereal.

The mesh topology of the NoC already splits the network in two, the *Topology Factor*. A packet can never collide with those of neighboring routers as they can never address the same router. It is only possible with packets that are at a router, which is an even distance away. A second way to separate network is by applying buffers in the router. Each *Buffer Stage* also increases the number of TDN according to the following formula:

$$\text{TDN} = \text{Topology Factor} \times \text{Buffer Stages}$$

These disjoint networks (Virtual Channels) can be used to provide services to the network.

The idea of LCs is to loop container packets between the source and destination. They can only insert data into an empty container. The LCs thus provide guaranteed access to the network and guarantee bandwidth and latency for all the VCs, which are set up.

SoCBUS

SoCBUS is a circuit switched NoC designed for hard real-time embedded systems [51], providing hard, short-lived GS connections. Circuit switching in a two-dimensional mesh network allows a low complexity design that avoids deadlock. The concept of Packet Connected Circuit (PCC) is introduced; a packet is switched through the network, locking the circuit as it goes to its final destination. It has shown acceptable performance for local traffic. However, it suffers from random traffic patterns, because the high probability of routes being blocked. As the paths are set up using BE routed packets, SoCBUS can be categorized as soft GS [2].

2.1.3 NoC evaluation

A major problem in NoC research, which is indicated in survey papers is that it is hard to make a fair comparison between different NoCs. There are many performance indicators, which cannot be compared. When for example looking at the bandwidth of a NoC, a close relation should be specified at the latency that is experienced. Jitter has been introduced for that purpose [7]. It is defined as the maximum time window wherein the specified bandwidth would always be reached. The same problem is true for the comparison of the cost functions of NoCs. It is hard to make a comparison of the area and power of designs implemented in different technologies (FPGA and ASIC with a range of feature sizes), and layout specific choices concerning the length of / distance between links.

The type of traffic that is used to evaluate the performance of a NoC is also under discussion. Two types of test cases can be distinguished; computation kernels and full applications. The first consists of tasks performing small operations like image processing, filtering or matrix calculations. Unfortunately, this is not a realistic workload for a MPSoC as those tasks represent too fine grained parallelism [42]. Full applications are a better traffic source and include for example video en-/decoding. The best approach would be to run multiple applications on a MPSoC to evaluate if the system is composable and also satisfies the expected usage of NoC-based systems.

Related to this is the issue that there is no common benchmark tool currently, in which NoCs can be evaluated [20]. Although numerous publications target such a tool, a common benchmark tool is not yet available [43].

2.1.4 Industry

The industry is also adapting their interconnects to the current trend of adding more and more cores to a SoC. Especially ring networks are becoming popular, which is in contrast to research where meshes are the standard. This might be

caused by different requirements, as the industry is for example more focused on a very high raw bandwidth instead of providing guarantees about maximum latency and minimum throughput.

The semiconductor chip maker with the highest revenue, Intel, is actively involved in interconnect developments. A few of its designs will be treated in this section; the Larrabee GPU and the teraFLOPS research processor, which both led to the release of the many-core Xeon Phi. Its current generation Central Processing Unit (CPU), Haswell, is also covered. The consortium of Sony, IBM and Toshiba developed an interesting interconnect for the Cell microprocessor.

Larrebee is a many-core visual computing architecture that uses multiple x86 CPU cores, supported by a wide vector processor unit. Inter-processor communication is performed over a bi-directional ring network. "When scaling to more than 16 cores, we use multiple short linked rings" [44]. It uses a very wide data-path of 512 bits per direction. Routing decisions are made before a message is injected into the ring. Processor tiles can only accept messages from one direction on even clocks and from the other direction on odd clocks cycles, allowing an efficient router implementation. This results in a high bandwidth interconnect with minimal contention at a very low hardware cost.

The ring is also used for the cache coherency. Each core has access to a subset of the L2-cache, the ring network ensures coherency for shared data. Fixed function logic is also placed around the ring around the ring network and is spread to reduce congestion. Lastly, access from the L2 caches to the memory is provided by the ring. The extra ring latency is typically very small compared to the latency of DRAM access [44].

The more research oriented Intel teraFLOPS makes uses of a different NoC topology to connect its 80 tiles (not consisting of CPUs but floating-point multiply accumulators) [47]. It features a packet switched 2-D mesh NoC designed to run at 4GHz, resulting in a bisection bandwidth of 2 Terabits/s. The NoC uses wormhole routing and flow control that is debit-based, where almost full bit signals are used. The mesh network is not as promising for the industry as researchers believe, since that network is not used in the next many-core architecture Xeon Phi, which makes use of a different network topology.

The Xeon Phi brings many x86 CPU cores to process highly parallel workloads on a single chip [13]. A bidirectional ring is implemented to allow communication to and from all those cores. It consists of a 512-bit wide data ring, a much smaller address ring and the least expensive flow control ring (all for each direction). The last two rings are actually implemented twice as they are small compared to the data ring, but increases its throughput.

Intel introduced a ring network to mainstream CPUs in their Sandy Bridge architecture. Each core has two entry points on the ring to minimize the latency that is experienced when communicating to the System Agent and GPU, which are placed on opposite sites on the chip [21]. Ring links are short enough to run at CPU speed, which covers the latency of traversing multiple hops to a destination on the ring. In Haswell though, the clock is decoupled from the CPU from a power efficiency point of view.

The Cell architecture of Sony, IBM and Toshiba consists of four rings as NoC [30]. Two of these turn in clockwise direction, while the others are rotating counter-

clockwise. Each data ring is 128-bits wide and is scheduled by a bus arbiter which handles requests. It schedules requests across the two rings with the shortest path to the destination. Each ring supports three concurrent data transfer that don't overlap, providing a total peak bandwidth of 204.8 Gigabytes/s (128 bytes at 1.6 GHz).

2.2 Accelerator sharing

Currently, most research concerning hardware accelerators is focused on tightly coupled accelerators that are attached to a processor. These kinds of co-processor(s) cannot be share among different processors, making system consisting of these accelerators very inflexible. One of the possible applications of a heterogeneous MPSoC is a SDR. As an SDR has many uses where multiple streams, potentially of different wireless standards, might need to be processed simultaneously, common Digital Signal Processor (DSP)-blocks (like a Viterbi/Turbo decoder or an FFT) can be shared to reduce SoC area.

Usually accelerator designs are prototyped on, for example, a Xilinx FPGA where a number of MicroBlaze (μB) processors are integrated within the system. These processors contain a Fast Simplex Link Bus, uni-directional point-to-point hardware FIFO communication link (FSL) interface that allows simple addition of custom hardware accelerators to a core.

An example of such a system is presented in [9], however, a few different configurations, regarding the placement of accelerators, are also investigated. Next to the tightly coupled scenario, accelerators are placed between μBs where one core can write data to it and the other can only read the resulting data (sort of an accelerator ring). The last scenario consists of an accelerator placed on a interconnect, which is shared by four μBs.

An interesting result is obtained in that work. It is found that better results are obtained when sharing a single accelerator among all processors, rather than connecting a dedicated accelerator to each processor. This can be explained by the fact that the utilization of the accelerator increases when sharing it, without a proportional increase in the load on the share interconnect. Instead processors are waiting until a Synchronization Engine, also connected using an FSL, grants access to the accelerator. As long as the accelerator's utilization does not approaches 100%, processors will become interleaved, increasing the accelerator utilization.

This architecture is not suitable for real-time application. The Synchronization Engine uses locks and barriers, which cannot guarantee a bound on the waiting time of the connected processors. Moreover, interrupts are used to signal the status of the shared accelerator; a troublesome method for real-time systems.

The only (known) accelerator sharing implementation in research that make use of hardware block is presented in [14]. A Global Accelerator Manager (GAM) is added to a heterogeneous MPSoC onto a mesh based NoC, which manages all accelerators. The paper shows that a performance and energy consumption improved compared to OS-bases management.

When a processor wants to make use of an hardware accelerator, the following actions take place:

1. The core requests an accelerator type X from the GAM, which acknowledges with a message if there is one available now or later, or reject the request.
2. Based on the response, the core takes one of the following actions:
   - (now): The Direct Memory Access-Controller (DMA-C) of the Accelerator is set up to transfer input data to the accelerator's local Scratchpad Memory (SPM).
   - (later): The core waits and reserves it, or continues on software path
   - (reject): The core continues on software path
3. Upon completion of the accelerated operation, interrupts are used to indicate that the operation is completed and when the output data is copied to main memory. The core then frees the accelerator by sending a message to the GAM.

Multiple accelerators of the same type can be managed by the GAM. The GAM tracks the type of accelerators that are present in the MPSoC, the number of each type, the jobs running and a waiting list combined with an estimate run time (to predict when an accelerator will become free). When no accelerator of the requested type is immediately available and the waiting time is too long for potential gains, a core switches to use a software implementation of the operation.

The GAM, however, provides no real-time guarantees about latency, as it only indicated the waiting time for a resource to cores, but does put a strict bound on it. Switching to software implementations and the usage of interrupts is also not desirable for a real-time system. For BE traffic, significant performance and energy improvement can be reached by utilizing such a accelerator manager.

"Operating Systems Should Manage Accelerators" is the title of a paper that promotes the use of the OS for accelerator management [36]. It states that current operating systems provide little support for accelerators, whereas future (heterogeneous) processor will contain more and more accelerators. The context of the paper is not in the area of real-time embedded systems and target a coarse grain sharing that should be implemented in OSs. Such an approach is less suitable for real-time systems, as it introduces high latencies, which result the need for large buffers between tasks.

2.3 Real-time analysis models

In order to guarantee correct real-time behavior of a system, it need to be analyzed using a real-time analysis models. End-to-end guarantees need to be provided to verify if design constraints can be met. In this section, the three main branches of analysis models are covered: Event Models (EM), Real-Time Calculus (RTC) and Synchronous Data Flow (SDF).

Event Models is the underlying analysis technique used in the SymTA/S approach [25]. It is based on scheduling techniques for tasks on single processor systems. The model describes the period of traffic and the jitter, the maximum deviation in arrival time of an event and the start of the corresponding period. This method, however, is not suitable for arbitrary cyclic graphs [23], where there is a cyclic dependency between a high and low priority task running on the same core for

example. Moreover, it does not capture correlation between data streams, resulting in a low accuracy.

Another analysis approach is Real-Time Calculus, which originates from traffic characteristics described in Network Calculus. It also has problems with cyclic (resource) dependencies and introduces inaccuracies compared to SDF [23]. Since the approach is at a higher abstraction level compared to SDF, usually a less accurate result is obtained.

SDF is a very powerful model for analyzing (streaming) applications. Especially Homogeneous Synchronous Data Flow (HSDF) graphs, which can be derived from more general SDF graphs, have strong analytical properties. The graphs can be used to analyze, at design-time, the temporal behavior and resource requirements (buffer sizes) of applications [24].

All these analysis techniques require the computation of a fixed point, since the analysis is a global problem where all component can affect the behavior of other components. Using SDF, this fixed point is described in linear equations (such that a linear program solver can be applied) whereas the other two techniques have a higher exponential time-complexity.

2.4 Summary

In this chapter, we first looked at NoCs and also specifically to a few research examples that provide guarantees for GS traffic in a mesh type NoC. All of those GS NoCs have at least one disadvantage, their implementation is quite large, because the connection-oriented approach requires much buffering. The industry, however, is targeting a ring topology for their state-of-the-art multi-core chips. Consequently, there is room for more research at small (connectionless) ring architectures that provide GS.

Efficient sharing of a hardware accelerator within a multi-core system is another topic that deserved more attention. Only a few sharing solutions are found in literature, which do not provide real-time guarantees.

A number of real-time analysis techniques have been discussed: Event Models, Real-Time Calculus and SDF. The technique that is best suitable for a broad range of applications while providing accurate results is SDF. Therefore SDF will be use throughout this thesis for analysis purposes.

CHAPTER 3

Starburst platform

The previous chapter gave a look at what other researchers have reached regarding important aspects of MPSoCs and the analysis models that can be applied to these. In the next chapter (Chapter 4), an analysis of the inter-core communication will be presented, but first, this chapter presents the underlying MPSoC in detail. This MPSoC was developed at the University of Twente, especially regarding the aspects that are relevant to the research questions of this thesis. A top-down approach is used to provide an insight in the physical layer of the system, the reconfigurable hardware layer and the software layer on top of it.

At the bottom of the platform lies an FPGA development board that is used to test and evaluate designs of the reconfigurable hardware and software. Section 3.1 contains all relevant parts of that physical system required for the upper layers.

Next is the tool-chain, called Starburst, which automatically generates the reconfigurable hardware. A high level overview of this hardware (including the processor tiles, the interconnect to the main memory, accelerator integration and the core-to-core communication network) is given in Section 3.2, which is further described in detail in its subsections.

At the end of this chapter, Section 3.3, the final layer is covered, the software layer. It focuses on the the communication between the processor tiles.

3.1 Hardware platform

The basis of the whole MPSoC platform is the physical layer which consists of the Xilinx Virtex-6 FPGA ML605 evaluation board. Important features, for example I/O capabilities, of the board, used for the Starburst platform, are treated in this section.

The ML605 board is build around a Xilinx Virtex-6 XC6VLX240T FPGA, see Figure 3.1 for an overview of the board. This FPGA contains 241,152 Logic Cells, each consisting of four Look-Up Tables (LUTs) and eight flip-flops. Next to these cells, 769 DSP-slices are present that can be used for high-speed multiply-accumulate operations, which are very often used in DSP-applications. With these numbers, the FPGA is a well balanced within the Virtex-6 series, containing an average number of Logic Cells, DSP-slices and I/O pins.

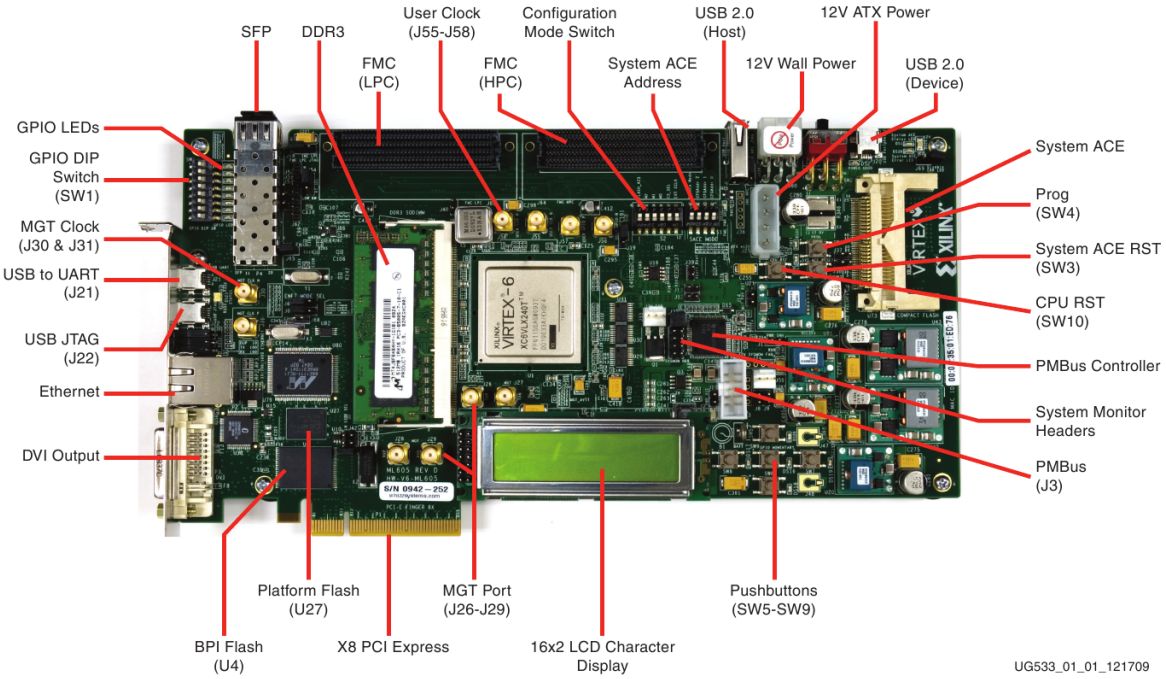

Figure 3.1: Overview of the Xilinx Virtex-6 FPGA ML605 development board [57].

Standard Xilinx tooling can be used to program the FPGA. It includes support for the MicroBlaze (μB) soft processor of Xilinx [54]. It is shown that 32 μB cores fit into this FPGA [39].

Besides the FPGA, many other peripherals are present on the evaluation board. A number of memories are present: DDR3, Platform Flash and a CompactFlash card. The 512 MB of DDR3 can be used as the main memory for processors implemented on the FPGA. It is interfaced by a memory controller that is part of the Xilinx tooling. The other memories are mainly used to store FPGA configuration files, so called bit files. When a fast setup of the FPGA is required (for example when it needs to comply to the PCI Express standard), the 128 Mb of Platform Flash is adequate and can be use to program the device in less than 100 ms.

An alternative location for the bit files is a CompactFlash card. The SystemACE CompactFlash configuration controller allows a selection of one of eight images by means of a hardware switch on the board. This selected image from the Compact-Flash is then used to program the FPGA (although a very specific structure on the memory card is required).

Communication with the outside world is mainly performed through Ethernet. It can be used to connect a host PC to the board. Other communication is enable by USB-ports. One is utilized as an USB-to-UART bridge. A second allows the FPGA to be program by a USB-to-JTAG bridge. The last USB port can for example be used to connect Human Interface Devices (HID), like a keyboard or mouse, to the board.

Video can be output via a DVI port on the board. An useful feature of the current platform is visualizing the content of the DDR3 to see if code has been uploaded to it.

When the board is used for specific applications, daughter boards can be connected to extend the functionality of the evaluation platform. The ML605 board contain two FPGA Mezzanine Connectors (FMC) for these purposes. Two boards have been connected to the Starburst to form and SDR platform; a Bitshark and a multichannel ADC board. These two boards will be briefly discussed in the following sections.

3.1.1 Bitshark

The Bitshark FMC-1RX daughter board, which can be see in Figure 3.2, is a broadband configurable Radio Frequency (RF) receiver made by Epiq Solutions [15]. It contains a single RF tuner that support a tuning range between 300 MHz and 4 GHz. Channels with a bandwidth up to 50 MHz can be processed and sent over its FMC to the ML605 board. It also contains a Global Positioning System (GPS) receiver, which can be used to obtain accurate timing information (next to its location).

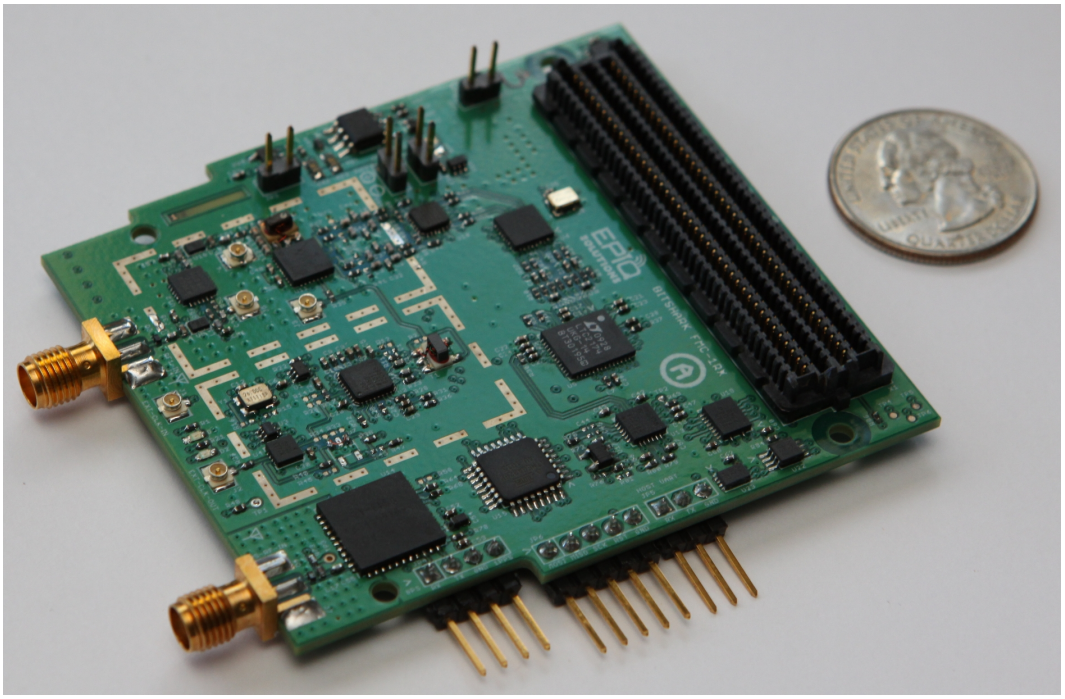

Figure 3.2: The Bitshark FMC-1RX daughter board

The board has a number of advantage compared to other RF receivers. Its precision is high, as two 14 bit ADCs are use to sample the I and Q-signals. Moreover, the ADCs have a high output rate of 105MS/s.

The real advantage is in the demodulation technique that is used. Normally, an RF signal is first down-converted to an intermediate frequency (IF). After that, further processing needs to be performed, in for example an FPGA, to convert the signal to baseband. Superheterodyne receivers use this technique.

The Bitshark is a direct-conversion receiver, where the signal is immediately converted to baseband without the need for an intermediate stage. Consequently, no IF-stage is required in the FPGA saving area. A disadvantage of a direct-conversion receiver is that it has difficulties with signals that are part of analogue standards.

3.1.2 Multichannel ADC

The second board that can be connected through an FMC is a multichannel ADC designed by 4DSP, the FMC125 [1]. The ADC can sample four channels simultaneously at 1.25 GS/s (two at 2.5GS/s) or it combines the channels into a single channel, then samples at 5.0 GS/s.

This multichannel output can for example be used for beamforming purposes, where the signals are combined after being shifted in time to 'look' into a certain direction. It is also possible to use it for wireless communication standards that use multiple radio channels (SIMO/MIMO) to improve communication performance.

The sampling frequency of this board is very high, however, it cannot directly be used in the same way as the Bitshark. Before the output of this multichannel ADC can be used, it first needs to be converted to baseband. It results in additional hardware or FPGA resources, largely dependent on the desired filter characteristics, to obtain a baseband signal.

3.2 Starburst

Starburst was originally developed at the University of Twente within the NEST (NEtherlands STreaming) research project [34]. The goal of the NEST consortium was to support future high-performance streaming applications that require real-time guarantees, such that the Dutch companies within NEST can compete with the international competition. By limiting the targeted applications to streaming ones, more specialized processor architectures can be used, shifting the focus of research from General Purpose Processors to MPSoCs.

At the University of Twente, Rutgers created a MPSoC platform, including a tool chain to automatically generate it from a specification in a XML-file. This platform is called Starburst, based on the term in astronomy: "In astronomy, starburst is a generic term to describe a region of space with an abnormally high rate of star formation. It is reserved for truly unusual objects" [50].

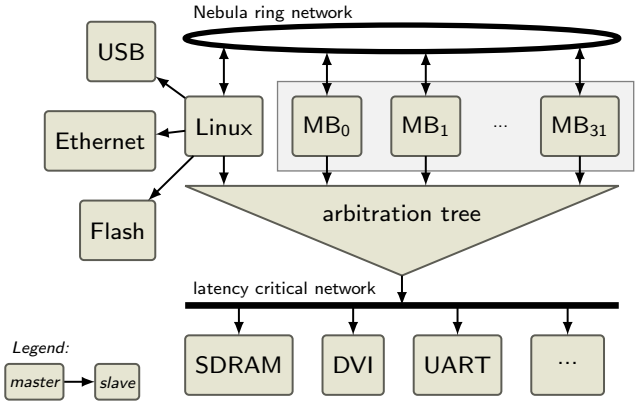

Figure 3.3: High level hardware architecture of the Starburst platform

A high-level overview of the hardware platform is given in Figure 3.3. As can be seen, the platform consists of a power-of-two number of μB processors and a Linux

tile for I/O capabilities. The inter-core communication is performed by means of a ring network. An arbitration tree is (mainly) used for access to the shared DDR3 memory.

In the following sections all these components will be treated. First, the processor tiles will be discussed in detail in Section 3.2.1 since they are the basis of the hardware platform. A special case of the tile is the Linux one. Differences between the two tiles are explained in Section 3.2.2.

Communication of these tiles is provided by the Warpfield interconnect. It consists of an arbitration tree (Section 3.2.3) and the Nebula ring. Details about this Nebula ring are important for the research problem and its implementation will be discussed in Section 3.2.4. Once hardware accelerators are integrated in the MPSoC, flow control is required, which is implemented in the extended version of the ring, introduced in Section 3.2.5.

3.2.1 Processor tile

The processor tiles in Starburst are based around the Xilinx MicroBlaze (μB) processor. It can easily be implemented in the reconfigurable FPGA using the default Xilinx tooling, which also contains a (C) compiler, allowing software to be created for it. The μB can support hardware features like a multiplier, barrel shifter and floating-point unit that are all enabled in Starburst. An overview of the additional hardware present on the processor tile is shown in Figure 3.4

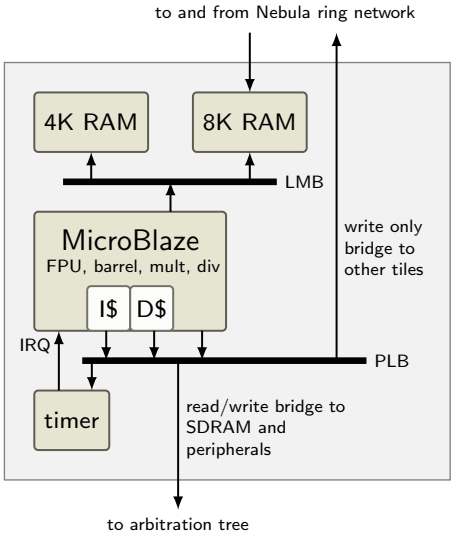

Figure 3.4: Processor tile containing a MicroBlaze processor, caches, memories and a timer.

Memory latency is minimized in the tile by making use of a few different types of memories. First, a data and instruction cache (by default for a total of 32 kB) is present. The instructions of small streaming applications should fit into the cache, reducing the number of requests to the shared memory. Local data is stored in a dedicated memory (4 kB) that is connect via the single cycle latency Local Memory Bus (LMB). Finally, a 8 kB large Scratchpad Memory (SPM) is added to bypass the shared memory for inter-core communication.

The only non-memory peripheral in a tile is a timer. It acts as the exclusive interrupt source for the μB and indicates when the processor needs to switch to a different thread. A fixed interrupt interval can be set in the timer which results in a predictable time-division multiplexed schedule running on the μB.

The connection to the main shared memory is enabled by the arbitration tree. It provides a maximum bound on the memory latency as is required by real-time applications. The μB can access it via the Processor Local Bus (PLB).

Communication to other cores is performed over the Nebula ring. Data can be written to other cores through the PLB interface. A small buffer is present in the ring interface to make sure that the PLB does not immediately stall when multiple words are transmitted.

The other part of the ring interface is the one where data enters the processor tiles. One port of the dual-ported SPM is used to directly store incoming data. This dedicated port is crucial for the ring, as it allows it to always keep turning without stalls. The second port of the SPM can be connected to either the PLB or LMB (for a lower latency).

3.2.2 Linux tile

Basically, the Linux tile is an extended version of the normal processor tiles. One of these tiles is added to the Starburst platform for I/O purposes. A stripped down version of Linux is running on the μB that is present on the tile. The advantage of running Linux is that a lot of peripherals can be added to the platform that are already supported by the operating system.

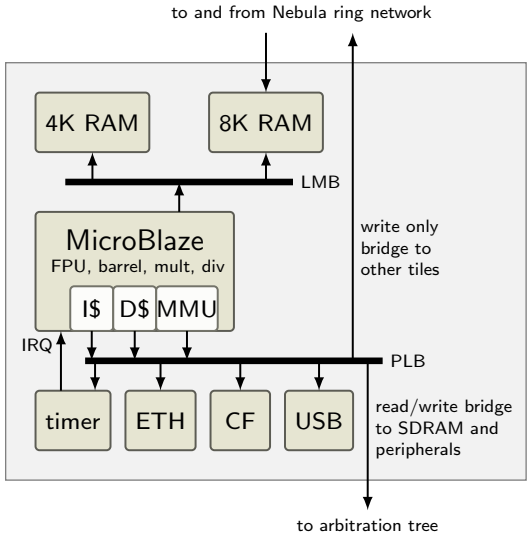

Figure 3.5: Linux tile

The only big difference between the regular tile and the Linux one is the addition of a Memory Management Unit (MMU) to the μB, as can be seen in Figure 3.5. It translates virtual to physical addresses and provided memory protection.

Most peripherals of the ML605 board are managed by the operating system running on this tile: the Ethernet connection to the host PC, CompactFlash storage, USB HID, LCD text display and switches.

3.2.3 Arbitration tree

For the access of the shared DDR3 memory a interconnect has been developed to overcome two traditional problems; larger than linear scaling in hardware to the number of processors, and high latency for memory reads [39]. Based on these shortcomings, an interconnect has been implemented that has the following list of features:

1. Starvation-free scheduling, such that every core eventually is granted access.
2. Work-conserving to optimize for latency, and thus trying to prevent unused memory cycles when requests of cores are pending.
3. Scale linearly in hardware costs to the number of cores.
4. Pipelined and decentralized arbitration to avoid long wires for high performance.

To reach these requirements, memory access requests are packetized and a timestamp and source ID are added. These packets enter a binary tree where each step performs local arbitration based on the timestamp, allowing the packet with the lowest timestamp to proceed. Linear scaling in hardware is reached as the number of arbitration steps in the binary tree are equal to the number of connected cores minus one, where the hardware costs of a step are independent on the number of cores. A small buffer can be placed between steps to shorten wires, however, increasing the latency.

Data coming back from the memory after a read command is multiplexed to the source based on the source ID, without the need for arbitration.

3.2.4 Nebula ring

The Nebula ring interconnect offers lossless communication between different processor tiles. Its simple unidirectional ring design results in a small implementation within the development platform. An overview of the ring is presented in Figure 3.6. The processor tiles are connected to the ring by a network interface which buffers incoming packets (plb2ring). When data reaches its destination after a few hops on the ring, it is written to the SPM of the destination tile.

Originally, the ring was designed for BE communication [39]. However, small extensions provide fairness and work-conserving behavior to the ring, making it suitable for predictable communication (as defined in Section 1.3). These extensions are implemented as additional rules in the inserter block that can be seen in Figure 3.7, which shows a schematic overview of a single ring link.

The principle behind the ring is that packets on it can always progress, no stalls are possible. Data packets thus need to be halted before they are inserted on the ring. As a consequence, it is possible that no packet is inserted at a link although it has

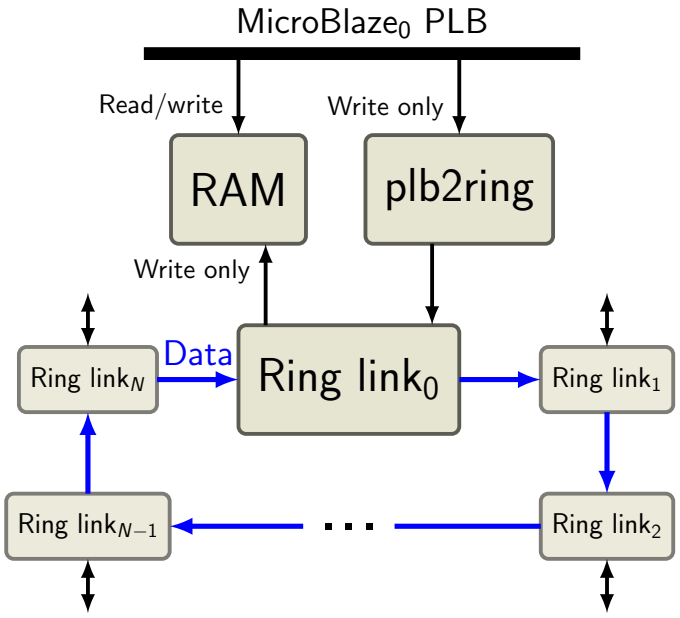

Figure 3.6: Block diagram of the Nebula ring connected to the PLB of a MicroBlaze.

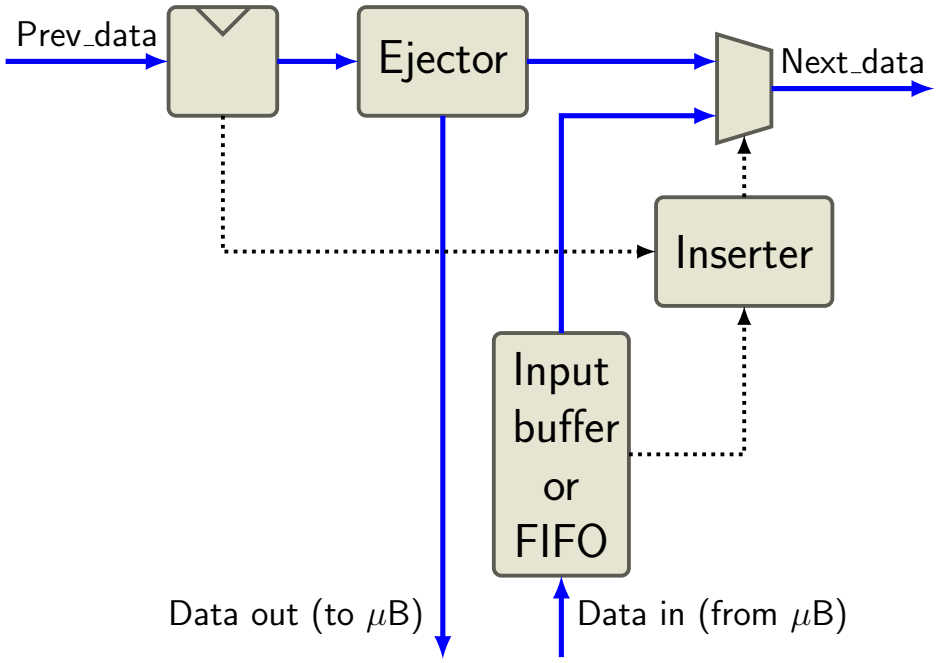

Figure 3.7: Schematic overview of a single ring link.

place for a packet. This has to do with the services that are provided by the ring. Starvation is easily possible in case a processor tile fills all passing slots, preventing access to the next link(s) for neighboring tiles.

Once a data packet enters the ring at a link, it is always forwarded to the next link until it reaches its destination where it is ejected. Packets wait in an input FIFO until they can be inserted onto the ring. Flip-flops between the links act a kind of shift register where data is constantly being moved.

Guarantees are provided to the system by labeling all the ring slots. Each slot receives a unique ID upon reset, which corresponds to the ID of the current local processor tile. Processors obtain a share of the total bandwidth of the ring, by only being allowed to use the slot with an ID corresponding to its own ID.

A lot of bandwidth is lost in this way as empty slots cannot be used when IDs are not matching. A work-conserving principle is introduced to allow those slots to be hijacked under specific conditions. When the destination of data of an empty slot is reached before the owner of the slot is reached, the slot may be filled. In this way the available bandwidth, especially for communication over a low number of links, is increased a lot, however, no guarantees can be given over this bandwidth. An overview of these rules is given in Table 3.1.

Table 3.1: Ruleset of a *ring link* in the Nebula ring

1. Incoming packets addressed to the local node are ejected to the local node.
2. All other incoming packets are passed to the neighbor router if the current slot does not belong to the local node and the local node has a packet to send.
3. Insert a packet on the ring if the local node has a packet to send and the slot is *available*.
A slot is available if at least one of the following rules is true:
1. The slot ID is equal to the local node ID.
2. The target address of the local packet is reached before the current slot passes its owner and the slot is free.
3. The target address is equal to owner of the slot and the slot is free.

A major disadvantage of this simple ring architecture is that it is impossible to attach address unaware hardware accelerators to it. The next section describes the solution to this limitation by adding flow control to the Nebula ring.

3.2.5 Flow controlled Nebula ring

Once hardware accelerators are placed on the ring, a form of flow control is required. For processors this flow control is implemented in a software layer (more about this in Section 3.3). Accelerators however, do not have a SPM and software running that manages communication, and instead use limited sized hardware First In, First Outs (FIFOs) to buffer incoming data. Overflow of these hardware FIFOs is prevented as an addition to the Nebula ring.

Flow control is implemented on a credit based way. When a node wants to send streaming data to an accelerator it needs a conservative estimation of the number of places that are free in the output buffer of the accelerator. The estimation is stored in a (hardware) counter that is decremented when a word is inserted onto the ring. Back-pressure is applied and stalls this stream when the counter reaches zero. At the moment that the accelerator frees a place in its output buffer, a credit is sent back to the source of the data stream incrementing the credit counter.

A separate ring is designed to transport credits from accelerators to stream sources, which can be seen in Figure 3.8. The separate ring allows a higher performance as

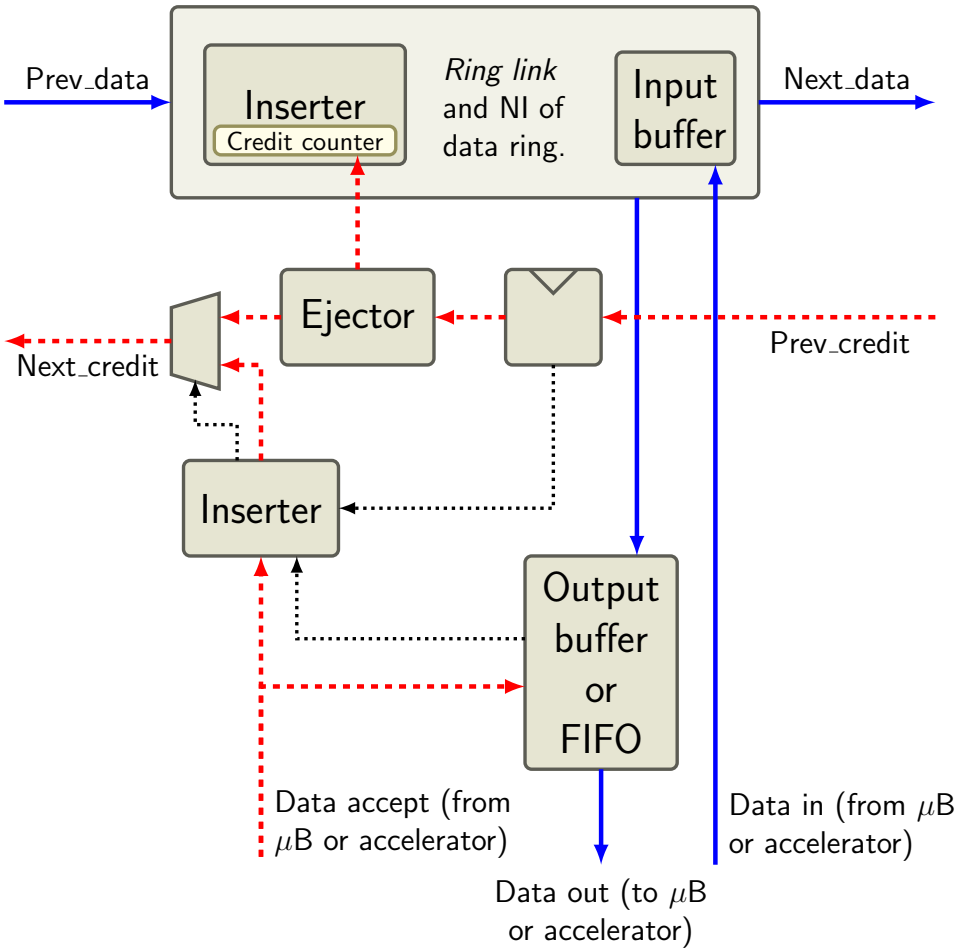

Figure 3.8: Schematic overview of a single ring link of the flow controlled version of the Nebula ring.

compared to reusing the main ring, without increasing the load on the main ring. In addition, hardware cost are only increased by a small amount as the credit ring is less wide then the other ring.

Another advantage is that the credit ring can turn in the opposite direction. A different direction results in a much better best-case performance over short distances, whereas the same direction causes a better worst-case behavior.

The address unaware accelerators obtain knowledge of the currently relevant part of the memory map of the system by means of a number of configuration registers. Two registers need to be set before an accelerator can be used; one contains the return address of credits and the second the address where the output of an accelerator needs to be sent (which can also be the address of another accelerator, creating a chain of accelerators). The registers resemble a doubly-linked list structure, since they point to the previous and next accelerator in a processing chain.

3.3 Software layer

On top of the hardware, generated by the Starburst tool-chain, a software layer resides to provide certain services. Each processor tile is running the custom micro-kernel, Helix, whose features are described in Section 3.3.1.

The most important aspect for a multi-core real-time system is the inter-process communication. The communication method is based upon the C-HEAP protocol, described in Section 3.3.2, whereas the actual implementation is discussed in Section 3.3.3.

3.3.1 Helix kernel

Every core in the MPSoC runs the Helix micro-kernel. It is a custom POSIX-like kernel, which supports the newlib C library. Multiple threads can run on the same core, as the Pthread standard is implemented in Helix. A TDM scheduler switches between threads timed by the interrupts coming from the hardware timer.

A number of daemon modules are started by the kernel that provide services like: communication and synchronization, statistics and profiling, memory management, timers and scheduling.

Regarding the main DDR3 memory of the MPSoC, it is important to note that it is split into two halves; a cached and uncached part. Local and read-only data should be stored in the cached region. Global variables, on the other hand, are placed in the uncached half of the memory, that is accessible by all cores, but is very slow. The Malloc Daemon manages all these memory requests.

3.3.2 C-HEAP

Communication quickly becomes a problem in heterogeneous multi-core systems. The in [35] presented approach can be used to mitigate this issue. This approach is as follows:

1. a top-down design methodology with various abstraction levels, which allows the designer to focus on the right problems at each level,
2. a design template to limit the design and modeling efforts through the reuse of predefined modules, and
3. a protocol for cooperation and communication between autonomous tasks, which should be transparent to the designer to ease the design process.

A template and protocol that incorporates these important points has been defined which is called C-HEAP, CPU-controlled HEterogeneous Architectures for signal Processing [35]. It uses a distributed communication implementation, where each processing device is able to initiate communication on its own, no central master is required. The distributed implementation allows a fine grain synchronization size, resulting in small buffer sizes.

Furthermore, communication is divided into synchronization and data transportation. Synchronization is most important and makes use of the following four primitives: `claim_space`, `release_data`, `claim_data`, `release_space`. Claim primitives

are blocking as a task requires filled token buffer as input and a empty token buffer at its output, before the task can run. A release call only takes places after a task is finished and is non-blocking.

FIFO behavior can be reached using some additional administration information. No consistency problems arise when a read and write counter are maintained that count modulo `maxtokens`. The producer increments it when it has written a new token, whereas the consumer increments it when a token is freed. An additional wrap flag is required to solve the ambiguity problem (is the FIFO empty or full?) that exists when both counter values are equal. Another approach to prevent this problem is to always leave one location in the FIFO empty such that the read and write pointers can never be the same.

3.3.3 CFIFO

Within Starburst, an implementation of C-HEAP is made as a library within Helix. It can be used to create a FIFO between a producer-consumer pair that is stored in a SPM. All communication is performed across the Nebula ring to reduce the load on the shared memory.

A read and write pointer (rp, wp) administration is stored in the SPM of the producer, as well as the consumer of a FIFO, which can also be seen in Figure 3.9. This double administration contains copies of variables (rp', wp') that represent a delayed version of the original variable, giving a conservative estimation of the free places in the FIFO. It is a sort of software implementation of credit based flow control.

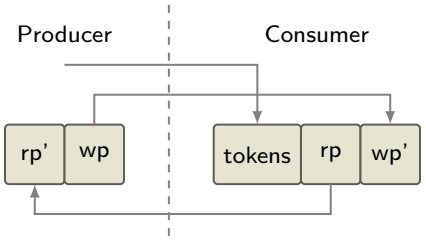

Figure 3.9: CFIFO memory and administration structure

An advantage of the double information is that the producer can poll for empty token in its local SPM and the same holds true for filled tokens at the consumer. By using these local memories less main memory bandwidth is consumed and a low latency is obtained. Tasks running on other tiles are therefore not influences by a tile polling its locally stored FIFO administration.

CHAPTER 4

Inter-core communication

The background of the Starburst platform has been explained in the previous chapter. In this chapter, a more detailed look at the inter-core communication is provided. Problems and possible improvements are identified, and are translated into a problem statement at the end of this chapter.

First, some details about the mapping of tasks onto such an MPSoC as Starburst are given in Section 4.1. The mapping has consequences on the performance of the inter-core communication, which is further investigated and analyzed in Section 4.3. In this section, an SDF analysis is performed on different task graphs that are mapped on Starburst processor tiles. Before the analysis is performed, an introduction to data flow analysis is provided in Section 4.2, as a basis for the subsequent sections.

A fundamental problem about the generation and transmission of software based FIFO credits in the current architecture is sketched in Section 4.4.

From these sections a problem statement is formulated in Section 4.5. This provides the basis for the implementation of the inter-core communication improvements that are described in the next chapter (Chapter 5).

4.1 Task mapping

Streaming applications exhibit a regular communication pattern. Individual tasks within such an application form a chain of tasks connected by streams of data. How these tasks are mapped on one or more cores on an MPSoC platform, can have a large influence on the performance of the application or the efficiency of the NoC, over which all tasks communicate.

The types of applications that are supposed to be mapped on Starburst are those that can be modeled with a Kahn Process Network (KPN) [28]. A KPN is a generalization of SDF graphs, without firing rules and timed behavior. A desirable property of these types of networks is that two KPNs can be combined to a new KPN, without changing the overall properties of the model. Using KPNs to model and map applications results in a deterministic behavior of a system.

Currently, the Nebula ring offers the same guarantees at each NI, independent of requirements and requested services. The fairness protocol ensures every processor receives the same amount of bandwidth and maximum bound on its latency. The

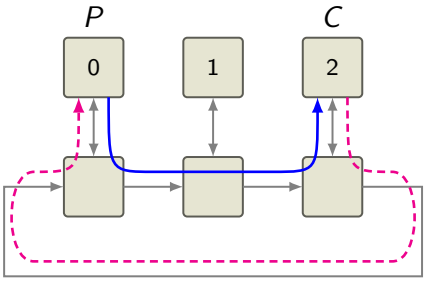

Figure 4.1: FIFO ring traffic where the consumer is mapped on processor tile 2

mapping of tasks only influences the work-conserving bandwidth, meant for BE traffic.

Figure 4.1 shows an example of a FIFO producer-consumer pair mapped onto processor tiles connected via a ring network. In this case, the consumer is placed on tile 2, whereas the producer is located at tile 0. Both the path from the producer to consumer as the other way around are highlighted, to show the communication taking place over the NoC. When this figure is compared to that of Figure 4.2, where the consumer is mapped to tile 1, it can be noticed that also in this case all ring links are used, with the only difference that the one between tile 1 and 2 is now used for traffic originating from the consumer instead of the producer. It can therefore be concluded that when both communication paths have the same performance requirements, it does not matter at which tile the consumer and producer are mapped.

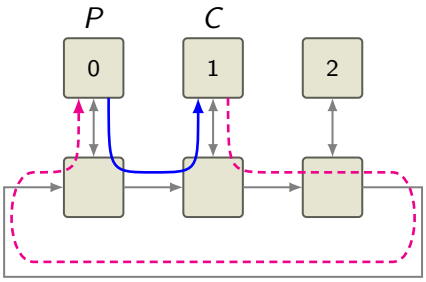

Figure 4.2: FIFO ring traffic where the consumer mapped on processor tile 1

All inter-core communication within Starburst happens through the CFIFO library discussed in Section 3.3.3. It does require greatly differing bandwidths for the NI of the producer and consumer task. The producer of a FIFO transmits a data packet of a certain length together with a read pointer, whereas the consumer only writes a write pointer on the ring in return. The scenario where both requirements to the NoC are closest, happens when the data packets are of size one. This minimal bandwidth difference is a factor: $\frac{BW_P}{BW_C} = 2$. For more realistic packet sizes, the factor scales almost linearly to the size. There is a large difference in bandwidth requirements for a producer and consumer of a FIFO, whereas they receive the same amount of guaranteed bandwidth.

With such a difference in requirements, it is advantageous to map a consumer close after the producer on the ring, as is the case in Figure 4.2. The high bandwidth stream is then only transported over a short distance (a low number of ring links),

limiting the utilization of the ring. Only the low bandwidth write pointer is sent over a large distance. The overall network usage, defined as:

$$\sum_{links} \text{link utilization} = \sum_{links} \frac{\text{used bandwidth}}{\text{link bandwidth}}$$

which can be reduced by mapping tasks according to their bandwidth requirements.

A lot of bandwidth is now wasted as it is not possible to adjust the Nebula ring to the application(s) that run on it. The fairness protocol implemented in the ring hardware and the software CFIFO communication library do not match very well. The software library must use all ring links to support the double FIFO administration, where the bandwidth requirement is high for one part and low for the remaining links. The ring is not customizable through software and provides a fixed bandwidth for every processor tile connected.

It is a research topic on itself how tasks should be mapped, especially on a heterogeneous MPSoC. Therefore, it is out of scope of this thesis. A recent paper in this field targets real-time tasks communicating over a ring bus [11]. Also a survey paper is available regarding application mapping strategies for NoC designs [41].

4.2 Data flow analysis

The previous section discussed the influence of the task mapping on the inter-core communication performance. What follows now is an introduction to data flow analysis using (H)SDF graphs (which were chosen in Section 2.3 as most suitable for the targeted real-time system) as this will be the basis for the next sections that use this technique to analyze the performance of the CFIFO library on top of the Nebula ring.

We start by analyzing the least expressive SDF model, which however, has the best analytical properties, the HSDF model. It can be used to determine the minimal guaranteed throughput that can be reached in a model, or the minimal buffer size required to reach a certain target throughput by hiding internal latencies can be calculated. After that, it is shown how a more general SDF graph can be converted to an HSDF graph, such that the previously mentioned properties can also be derived from it.

As an example, an HSDF graph will be created from a task graph of a producer connected to a consumer via a FIFO-buffer, which is shown between explicitly in Figure 4.3. When this task graph is modeled with an HSDF graph, it should only consist of actors, queues and tokens. The actors in a graph are connected by unbounded FIFOs in which tokens can be stored and are represented by arrows. Dots serve as tokens that are initial stored in these queues. Actors have a firing rule, which for an HSDF graph, states that there must be one token present on every incoming queue. When the actor fires, it instantly produces one token on every outgoing queue after its firing duration, $\rho(actor)$, has elapsed.

The FIFO in the task graph can be modeled by two queues and initial tokens on the queue returning from the consumer to the producer, as is visualized in Figure 4.4. A self-edge with one initial token can be added to the producer and/or consumer to indicate that the actor cannot fire before the previous firing has finished, preventing

Figure 4.3: Task graph of a producer that is connected to a consumer via a FIFO-buffer which is explicitly shown between them.

parallel executions of a task. When the consumer uses one token from the FIFO, it sends a message back to the producer to indicate a new token can be placed in the FIFO. These initial tokens thus represent the bounded size of the FIFO and prevent overflows in it.

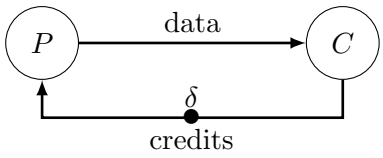

Figure 4.4: Data flow model of the task graph in Figure 4.3

An interesting property of such an HSDF graph is that it can show whether deadlock can occur. Deadlock can only take place when there are cycles in a graph that do not contain initial tokens and preventing the graph from making progress.

The maximum throughput that can be reached in an HSDF graph directly follows from a property of the graph, the Maximum Cycle Mean (MCM). The MCM of a graph is defined as:

$$MCM = \mu(G) = \max_{\text{all loops } o \in O(G)} \left(\frac{\sum_{v \in V(o)} \rho(v)}{\sum_{e \in E(o)} \delta(e)} \right) \tag{4.1}$$

where v are actors, e edges, o simple cycles in the graph (where each node is only passed once) and $\delta(e)$ initial tokens on those edges. The throughput is equal to the inverse of this expression and represents the average number of tokens produced per time unit by every actor in the graph.

As an example, the MCM of the graph in Figure 4.4 is now calculated. However, self-edges on both actors are included to show to effect of multiple simple cycles. The following list of cycles can be identified in the graph:

$$\begin{aligned} o_1 &= (P, C), (C, P) \\ o_2 &= (P, P) \\ o_3 &= (C, C) \end{aligned}$$

For each of these cycles, the firing duration of all actors is added and divided by the number of initial tokens on all edges. This results in the expression for the MCM:

$$MCM = \max \left(\frac{\rho(P) + \rho(C)}{\delta}, \frac{\rho(P)}{1}, \frac{\rho(C)}{1} \right)$$

What can be concluded from this expression is that it is advantageous to have a FIFO with a size of at least two tokens, as the FIFO is limiting the throughput of the system otherwise.

Realistic streaming applications often cannot be modeled with an HSDF graph, as its actors consume and/or produce more than one token. SDF graphs offer this freedom at the cost of losing a lot of properties compared to HSDF graphs, like the MCM. Nonetheless, it is possible to convert a SDF to an HSDF graph [45].

The conversion is performed by increasing the number of actors and queues by duplicating them. As an example, a part of a SDF graph is shown in Figure 4.5, where a producer and consumer actor work with packets of S tokens whereas the data is transported over a (ring) network, which delivers only single tokens. In the graph, the L-actor corresponds to a latency in the network and the B-actor limits the throughput between the producer and consumer.

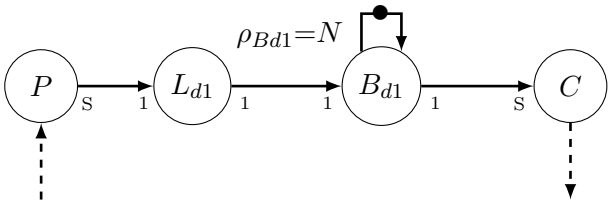

Figure 4.5: SDF graph

An HSDF graph is obtained by duplicating the L and B-actor S times. Moreover, the self-edge on the D-actor is changed to a cycle, to prevent those B-actors from firing simultaneously or without a fixed order. The result of the conversion is the much larger HSDF graph shown in Figure 4.6.

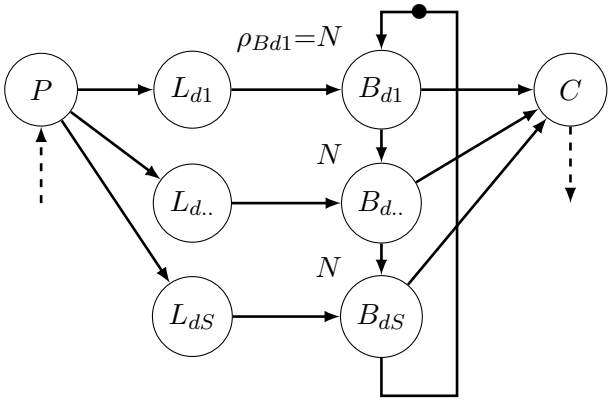

Figure 4.6: Converted to HDSF graph of Figure 4.5

As this case consisted of a very simple graph, an additional minimization can be performed to reduce the number of actors. By increasing the token size by a factor S, all B-actors are combined to a single actor with an S times larger firing duration, as is proven in Appendix A. All tokens experience the same latency, allowing the L-actors to be replaced by a single actor. This reduced graph is shown in Figure 4.7

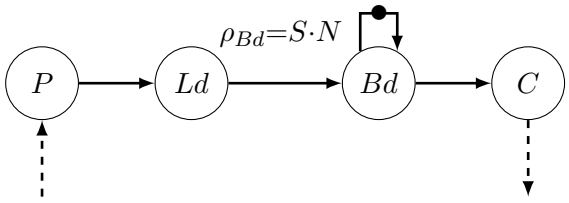

Figure 4.7: Reduced HDSF graph of Figure 4.6

4.3 FIFO data flow

The data flow analysis presented in the previous section will now be used to analyze the FIFO communication as implemented within the Starburst platform. First, the basic FIFO SDF model (Figure 4.4) is extended by including the fairness protocol of the Nebula ring. Then, the focus will be on the influence of chaining multiple producer-consumer pairs, or tasks, together, as is the case in streaming applications that tend to use more than two of the many cores present in a MPSoC. A distinction is made between acyclic task graphs (Section 4.3.1) and cyclic tasks graphs (Section 4.3.2).

Extending the FIFO model with the ring protocol requires the relevant properties of the fairness protocol to be taken into account. The Nebula ring places a limit on the bandwidth a tile can use, by only allowing it to insert data on the ring in its own slot. A latency is also experienced as it takes some time before the right slot is returned to its owner, and data needs to be moved over a certain distance to its destination.

These effects can be modeled by a latency-rate server [48]. It consists of two actors, one with a self-edge determining the maximum rate, and the other adding a latency. The ring bandwidth depends on total number of slots on the ring, which is equal to the number of ring links, N, as only one of those slots is allowed to be used. It is modeled in the data and credit bandwidth limiting actors B_d and B_c shown in Figure 4.8. Both have the same firing duration of N clock cycles.

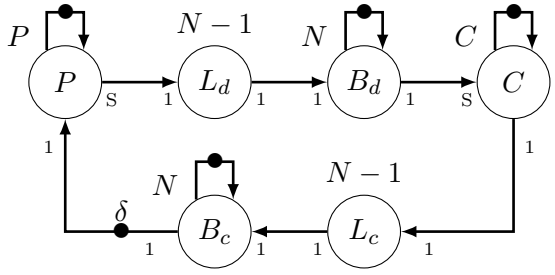

Figure 4.8: SDF model of the FIFO

The worst case insertion latency is experienced when the 'own' slots just moved away. The delay that is than experienced is $N-1$ clock cycles. The latency actors L_d and L_c add this delay to the model. The execution time of L_d must be increase by a number equal to the distance D from producer to consumer, to cover the time it takes to transport the data. Credits travel over a distance $N-D$, the remainder of the ring length minus the producer to consumer distance, which should be added to the L_c-actor.

Finally, the difference in required bandwidth for data and credits is modeled in the S factor. For each credit the producer receives it transmits one FIFO token, consisting of $S-1$ data packets and one read pointer. The SDF is no longer homogeneous and a conversion step is required to determine the MCM of the graph. This process is already shown in Section 4.2 and results in a firing duration of B_d of $S * N$. The coupling between the new model and the fairness protocol is less obvious now as the tokens on the graph from producer to consumer represent $S-1$ data packets, whereas those heading from consumer to the producer are a single data packet containing the write pointer. The actual data rate of the top part of the graph is a factor S (or $S-1$ to exclude the write pointer) higher than the inverse of the MCM, ($\frac{S}{MCM}$).

Equation 4.1 can be used to calculate the Cycle Mean of the one large simple cycle, next to the cycles made by self-edges.

$$\begin{aligned} MCM &= \max\left(\tfrac{\rho(P)+L_d+B_d+\rho(C)+L_c+B_d}{\delta}, \tfrac{\rho(P)}{1}, \tfrac{\rho(C)}{1}, \tfrac{\rho(B_d)}{1}, \tfrac{\rho(B_l)}{1}\right) \\ &= \max\left(\tfrac{\rho(P)+N-1+D+S*N+\rho(C)+N-1+N-D+N}{\delta}, \tfrac{\rho(P)}{1}, \tfrac{\rho(C)}{1}, \tfrac{S*N}{1}, \tfrac{N}{1}\right) \\ &= \max\left(\tfrac{\rho(P)+\rho(C)+(S+4)N-2}{\delta}, \tfrac{\rho(P)}{1}, \tfrac{\rho(C)}{1}, \tfrac{S*N}{1}, \tfrac{N}{1}\right) \end{aligned} \tag{4.2}$$

When it is assumed that the firing duration of the producer and consumer are not the limiting factor, as the actual duration is application dependent, it is quickly seen that the throughput is limited by the data bandwidth of the ring, the B_d-actor. A buffer size (δ) of two is then enough to hide the latency of the large cycle, when the packet size is not very small.

4.3.1 Acyclic data flow

A task graph is acyclic when a stream of raw data enters the system and is passed from task to task without feedback loops. Tasks are not adjusted based on information obtained at a later stage of the processing chain. As a consequence, also the SDF model of such an application can be constructed by composing it of producer-consumer pair graphs.

An example of such a graph, consisting of two pairs, can be seen in Figure 4.9. Task T_2 requires data from the first FIFO before it fires and forwards its output to the second FIFO.

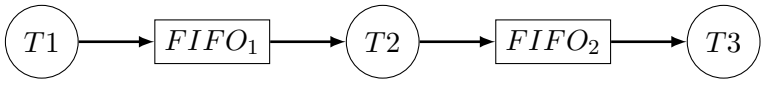

Figure 4.9: Acyclic task graph

For simplicity reasons, all non-relevant edges are removed from the SDF graph of Figure 4.8, such as self-edges and rate-actors. The resulting graph only has two simple cycles, one per FIFO, as shown in Figure 4.10.

The calculation of the MCM is straightforward as there is no additional feedback introduced by combining multiple SDF graphs.

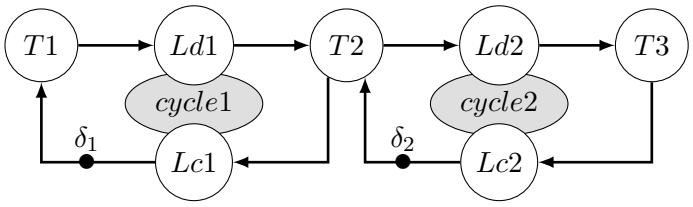

Figure 4.10: SDF model of task graph of Figure 4.9

$$\begin{aligned} o_1 &= (T_1, L_d1), (L_{d1}, T_2), (T_2, T_{c1}), (T_{c1}, T_1) \\ o_2 &= (T_2, L_d2), (L_{d2}, T_3), (T_3, T_{c2}), (T_{c2}, T_2) \end{aligned}$$

$$MCM = \max\left(\frac{\rho(T_1) + L_{d1} + \rho(T_2) + L_{c1}}{\delta_1}, \frac{\rho(T_2) + L_{d2} + \rho(T_3) + L_{c2}}{\delta_2}\right)$$

Task T_2 acts as both a consumer and producer for the MCM. Communication latency can be hidden by increasing the δ_1 and/or δ_2 buffer.

Composing a SDF model of a chain of acyclic task can however reduce the maximum throughput that can be reached, compared to that of the individual producer-consumer pairs. Even in an acyclic graph, cycles can be introduced by combining these pairs. When for example looking at the task graph and its SDF model in Figure 4.11 (a-b), two additional simple cycles are introduced; (T1 → T3 → T2 → T1) and (T1 → T2 → T3 → T1). The first contains 3 tasks and 2 buffer, a better MCM than the isolated producer-consumer pair case where 2 tasks share a single buffer. The second cycles only contains one buffer and 3 tasks, leading to a worst MCM. This negative effect of the additional SDF cycles in a acyclic task graph can be compensated by increasing the buffer size in each path to an equal number ($\delta_3 = \delta_1 + \delta_2$).

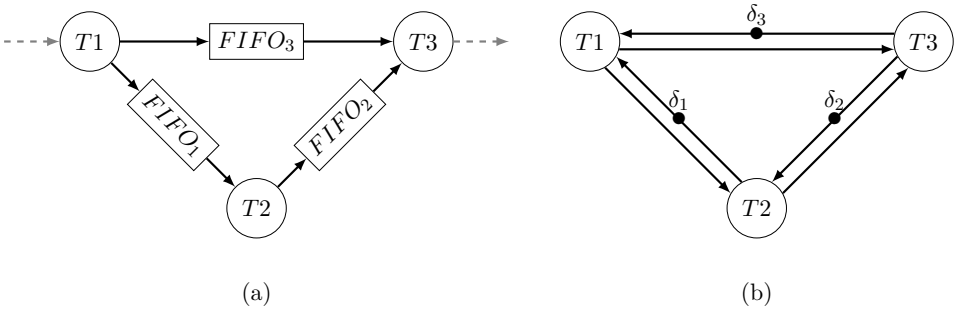

Figure 4.11: Acyclic task graph containing 2 different paths (a), and the SDF model of it in (b)

In the case where parallel acyclic paths in a task graph are equal, they containing the same number of tokens an we assume an equal firing duration of all tasks, the MCM is not negatively influences by composing a large SDF out of smaller producer-consumer pair graphs. Figure 4.12 show this case, where the two parallel paths to task T4 contain the same number of buffers. When $\delta_1 + \delta_2 = \delta_3 + \delta_4$, the MCM of the SDF graph is not worsened by the additional clockwise and counter-clockwise cycles; (T1 → T3 → T4 → T2 → T1) and (T1 → T2 → T4 → T3 → T1). Both

cycles contain 4 tasks and 2 buffers, thereby this shows the same MCM is obtained as for a single producer-consumer pair.

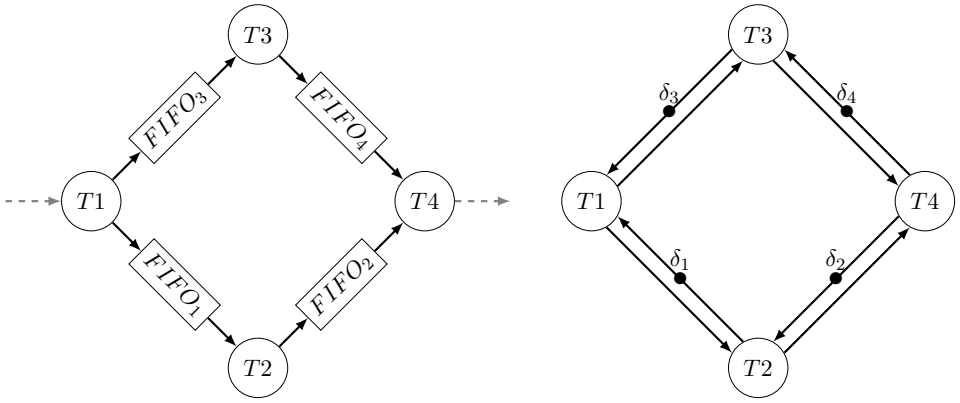

Figure 4.12: Acyclic task graph containing 2 equal paths (a), and the SDF model of it in (b)

4.3.2 Cyclic data flow

When looking at cyclic task graphs, like the one in Figure 4.13 that contains a feedback path, the situation becomes more interesting. When the same analysis is performed as for the acyclic case, it follows that the cycle in the task graph results in a different type of loop in the SDF, as can be seen in Figure 4.14.

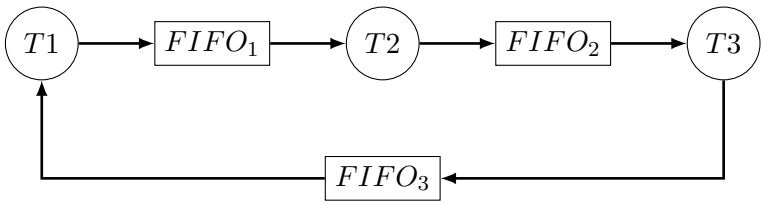

Figure 4.13: Cyclic task graph

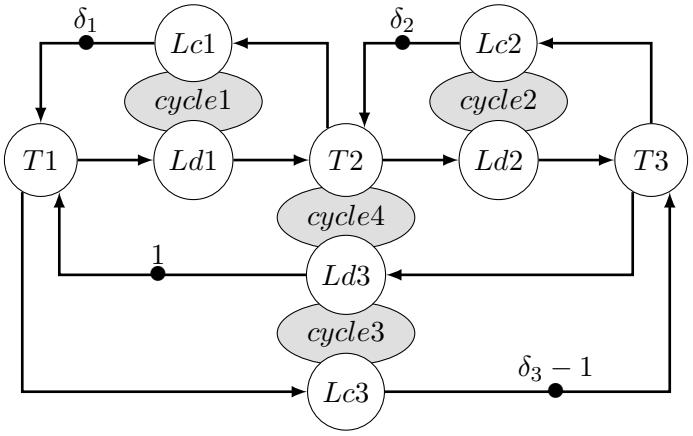

Figure 4.14: SDF model of task graph of Figure 4.13

Also for cyclic task graphs, the MCM is determined by using Equation 4.1, resulting in the following MCM expression:

$$
\begin{aligned}
o_1 &= (T_1, L_{d1}), (L_{d1}, T_2), (T_2, L_{c1}), (L_{c1}, T_1) \\
o_2 &= (T_2, L_{d2}), (L_{d2}, T_3), (T_3, L_{c2}), (L_{c2}, T_2) \\
o_3 &= (T_3, L_{d3}), (L_{d3}, T_1), (T_1, L_{c3}), (L_{c3}, T_3) \\
o_4 &= (T_1, L_{d1}), (L_{d1}, T_2), (T_2, L_{d2}), (L_{d2}, T_3), (T_3, L_{d3}), (L_{d3}, T_1)
\end{aligned}
$$

$$
MCM = \max \left(\begin{array}{l}
\frac{\rho(T_1)+L_{d1}+\rho(T_2)+L_{c1}}{\delta_1}, \\
\frac{\rho(T_2)+L_{d2}+\rho(T_3)+L_{c2}}{\delta_2}, \\
\frac{\rho(T_3)+L_{d3}+\rho(T_1)+L_{c3}}{\delta_3-1}, \\
\frac{\rho(T_1)+L_{d1}+\rho(T_2)+L_{d2}+\rho(T_3)+L_{d3}}{1}
\end{array} \right)
$$

The fourth cycle, as shown in Figure 4.14, cannot be influenced by increasing a buffer size. It only contains latencies related to the data paths of the FIFO, ignoring the credit paths. Optimization for this type of task graph therefore requires an increased data packet bandwidth in order to increase its performance.

Task mapping can also affect the performance of this cycle as it is, amongst others, determine the distance over which the data is transported. The minimum latency is however bounded by the size of the ring. Cyclic tasks will always experience a latency in the data path of at least the round-trip time of the ring, which cannot be reduced by mapping tasks differently.

4.4 Credit burst

Within the Starburst platform, guarantees about the worst-case performance of real-time applications are required. A situation has been identified however, which has a unwanted negative influence on the performance of the platform. The situation occurs when multiple threads are running on a processor tile, with at least one producer and consumer thread, filling and emptying software FIFOs.

The communication of such an application should satisfy the monotonic behavior. This means that an earlier production of a token cannot result in a later firing of actors during self-timed execution. A monotonic abstraction of the application should therefore exist. If the abstraction an application is possible using an HSDF graph, the monotonic behavior is automatically guaranteed [18, 49].

When a consumer produces tokens earlier, it should have no negative consequence on the worst-case performance of the rest of the system. An example of an earlier production occurs when a consumer of a FIFO has a best-case firing duration (much) lower than its worst-case. At its extreme, the execution time of the consumer can reach 0 cycles, shown Figure 4.15. It then instantly consumer all tokens present in its FIFO by generating a number of credit tokens, the read pointers. These credits must all be send across the Nebula ring towards the producer of the tokens.

This is no problem for other processor tiles as the ring prevents tiles from using more than its own allocated bandwidth, but it does influence the performance of other tasks running on the processor. A number of credits can be produced equal

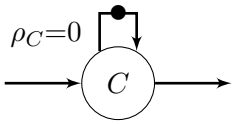

Figure 4.15: Best-case execution time of the consuming task 'C' of 0 cycles.

to the maximum number of FIFO tokens present in its local SPM. By producing all these credits, the input buffer of the NI is quickly filled, preventing data to be inserted by the producing task running on the same processor.

At the moment the TDM scheduler switches from the consumer to producer task the producer cannot immediately send data over the ring as it stalls because of the full NI buffer. Its data is only inserted onto the ring when the buffer, filled with credits, is emptied first.

This influence from two independent tasks can be compensated for in the SDF graph. It will, however, introduce a large latency corresponding to the time it takes to transmit the content of a full NI buffer across the ring. Moreover, it is not useful to receive credits before certain points in a precomputed periodic task schedule if the requirements are to satisfy a throughput constrain. Receiving multiple credits in quick succession, credit burst, therefore does not offer any advantage.

Based on this analysis, it would be advantageous to provide different network services to consumers and producers. Doing so can prevent the influence of the two types of traffic on each other and increases the worst-case performance of an application running on the Starburst platform.

4.5 Problem statement

The different analyses provided in this chapter all conclude that the services provided by the Nebula ring do not match the requirements of the software FIFO library. When looking at the mapping of tasks, the MCM of the HSDF graph of a FIFO, the cyclic task graph or the burstiness of credit tokens, all suggest a better performance can be reached by customizing the ring to the characteristics of the FIFO traffic. The Nebula ring is a connectionless NoC, offering the same service to each NI, and hence, currently does not support such a customization.

It is found that the traffic can be split in two types; data and credit traffic. Data traffic is usually mapped to short distance communication and requires a high (bursty) bandwidth and a low latency. Credit traffic, on the other hand, travels over a longer distance, but requires a much lower throughput and is less dependent on its latency as is illustrated in Section 4.3.2.

In this chapter, the following problems related to the design of the current Nebula ring have been identified:

- There are two types of traffic on the ring with greatly differing requirements, data and credit traffic, whereas the same "fairness" guarantees are offered to both. Moreover, both can influence each other in a negative way preventing a composable behavior of the Starburst system.

    - Data traffic takes place over short distance of a few ring hops and requires a high throughput. The way tasks are mapped onto the platform is shown to provide an opportunity to increase its bandwidth for streaming applications.
    - Credit traffic on the other hand has the opposite requirements, it demands a low throughput, however, over a large distance of about the complete round-trip of the ring. A burst of credit can cause additional latency for data traffic originating from other tasks running on the same processor tile.
- The strength of the connectionless ring interconnect is its small hardware size scaling linear to the number of cores and the predictable behavior. Improvements to the ring design related to the two traffic types should also satisfy these properties.

A solution to these problems is presented in the next chapter (Chapter 5), which also contains the details of its implementation.

CHAPTER 5

Implementation

The previous chapter explained the problems regarding inter-core communication inside the Starburst platform. A solution to these problems will be presented in this chapter, together with details about how it is implemented in reconfigurable hardware.

All identified problems are related to the differences in requirements between the two types of traffic that are present on the ring; data packets and credits. Those two types require a different technique to improve and are therefore discussed in separated sections.

First, the credits are treated in Section 5.1, where its traffic requirements are determined, different solutions are presented, a final solution is chosen and the implementation of this solution is explained and modeled in an HSDF graph.

After that, a simple solution is presented in Section 5.2 to improve the guaranteed data bandwidth of tasks, assuming information about the task graph and the mapping of it are known.

At the end of this chapter, the implemented changes are combined into an analysis model of a FIFO; an SDF graph. This model shows the predictability of the total design that is implemented and can be used in the Evaluation (Chapter 6) to compare it to the previous hardware architecture.

5.1 Credit

The previous chapters showed there is a mismatch between the service offered by the Nebula ring and the requirements of the software FIFO communication library. The so called credits receive the same amount of throughput and maximum bound on latency as data traffic, although less bandwidth is required. This is possibly limiting the overall communication performance, as more bandwidth could be allocated to data traffic.

A logical step is to separate credit from data traffic. By sending both types of traffic through a different NI buffer the influence on each other is removed. This second buffer can be small, as much less data is passing through it, compared to the other data buffer. When a producer wants to send data across the ring it cannot be stalled any more as a consequence of a full credit buffer and vice versa for a consumer and the data buffer.

The Nebula ring can than be adjusted to offer a different kind of service to the two buffers inside each NI. A traffic shaper could be designed which ensures a predictable behavior by multiplexing the credit and data tokens to a ring link offering different guarantees for both types of traffic.

Currently, a predictable communication ring is obtained by only allowing an NI to insert data onto the ring when its own slot passes. As there are N nodes on the ring, an NI is guaranteed to be able to insert data once every N clock cycles. Improvement techniques for the overall communication throughput can make use of the mapping of tasks, which will be the topic in Section 5.2. A predictable data throughput can only be reach when a technique is applied to limit the bandwidth of credit traffic. Section 5.1.1 explains why and how credit bandwidth is limited.

5.1.1 Limit credit bandwidth

In Chapter 4, the traffic pattern for credits has been identified. It has shown that credits travel over a long distance; almost the complete length of ring, but require a low bandwidth up to half of that of data traffic, but in practice much less then it.

When every NI is sending credits in their own slot over a distance equal to the total length of the ring, the ring will have a network usage of 100%. It is therefore not possible to maintain the current guarantees (an NI can always use its own ring slot) for the long distance credit communication, as it leaves no room for the higher throughput demanding data traffic. Given the large distance over which the communication takes place, and low requirements on bandwidth, a technique should be implemented to limit the bandwidth for credit traffic below the present guarantee of 1/N cycles.

There are multiple implementations possible to limit the credit bandwidth where the communication remains predictable. Three possibilities are discussed in the following sections. The possibility with the best performance will be chosen and its implementation will be treated in Section 5.1.2.

Multiple credit stream per slot

One option is to reserve a number of slots for credits. The total available bandwidth can then be distributed between data and credit traffic, by changing the number of slots that are available for both types of traffic.

Each of these credit slots can be further divided in a number of sub-slots. A NI can only insert its credit tokens in one sub-slot, providing a fair share of credit bandwidth, and a fixed (maximum) latency. The latency and bandwidth are coupled and both depend on the number of nodes that can use a credit slot. The maximum latency occurs when all sub-slots are consecutively filled with credits traveling over their maximum distance N. When there are S sub-slots, the maximum possible latency before a node can insert a credit token onto the ring is then $N \cdot S$.

As there are not enough data slots left for all nodes on the ring, they need to be shared. A data slot need to be shared between at most two nodes when the credit bandwidth is chosen to be equal or less then the data bandwidth. The maximum latency for data traffic then is $2N - 1$.

One credit stream in multiple slots

Another option is to spread the credits of an NI across multiple slots. The amount of credit bandwidth that is required by an NI is split in a number of smaller credit streams. Because many slots can be used to transmit credit in, the average credit latency shall be relatively low.

In order to for the system to remain predictable, credits cannot be inserted randomly in one of the slots available for credits. It must be prevented that many NIs are all inserting credits in the same slot, causing a large latency. Credits therefore need to be spread over the possible slots. This can for example be accomplished by only allowing an NI to insert a second credit in a slot once all others have been used once.

A major disadvantage of this approach is that although the latency for credits can be low, it will be at the cost of the latency for data traffic. Data packet can be blocked by credits originating from multiple NIs, resulting in the high latency. This is a less favorable option, since the previous chapter concluded that a low latency is required for data traffic, whereas the credit latency is less of an issue.

One credit stream per slot

The final option is to limit an NI to be only allowed to transmit credits in its own slot. Data bandwidth is guaranteed by preventing credits to be inserted every time the slot passes. This idea resembles the present Nebula ring mostly, allowing an easy integration of it into the ring.

Predictability can be obtained by setting the time limit of a period during which an NI can insert one credit. The length of the period is a multiple of the ring round-trip time, and depends on the number of nodes on the ring. The available bandwidth and maximum latency for credits can be derived directly from that period: the credit period (P_c). Every other time a slot passed its owner, the NI can insert data onto the ring, making the remaining bandwidth available for data communication.

The maximum latency for data traffic is increased by the ring round-trip time N caused by a credit inserted onto the ring, compared to the current $N-1$ cycles. The comparison to the previous architecture is not fair, as the $N-1$ latency made no distinction between data and credit traffic. When not looking at the buffer source, the credit or data buffer, in the NI, every N cycles a packet can be inserted onto the ring.

A work-conserving principle can easily be applied to this option. When the credit buffer is empty, it is always possible to insert a data packet onto the ring, giving all the bandwidth of a slot to data communication, in case no credits are present. The opposite, credits packets using data bandwidth, is not possible because of the large distance credits travel over the ring.

5.1.2 Implementation

From these three options, one bandwidth limiting technique must be chosen. Spreading a credit stream over multiple slots is the worst option, since it introduces high worst-case latencies and favors credits instead of the higher demanding data traffic.

The maximum latency for data traffic is the same when using credit sub-slots or when using the own slot for credits. However, the sub-slot option has a high credit latency. The sum of both data and credit latency is important as it determines the minimal buffer size required to reach a certain throughput. Therefore, the single stream per slot option has been chosen to be implemented. It also has the advantage that it is the easiest to implement within the present platform, as it resembles the current fairness protocol the most.

The simple ring architecture, as shown in Figure 3.7, will be the basis for the implementation of a ring where data and credit traffic is separated from each other. Features which need to be added to incorporate guarantees for credit traffic are: a second input buffer, for credits only, and a traffic shaper (TS), determining when to insert data or a credit onto the ring. These additions are shown in the shaded area in Figure 5.1.

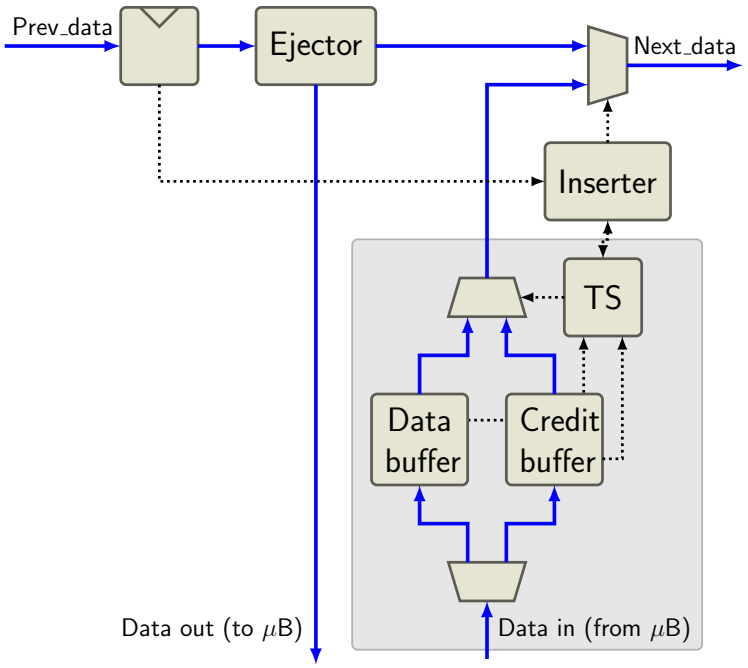

Figure 5.1: Overview of the ring architecture where the additional components are shaded.

The double buffer is implemented such that a full credit buffer has no influence on the data buffer making both types of traffic independent of each other. The PLB will not stall for data traffic destined for a data buffer which is not full. When credit traffic is present on the PLB, it will stall however, because of the full credit buffer. The distinction between data and credit is made by looking at the addresses on the PLB. As can be seen in Table 5.1, there are a number of unused bits in the address map of the ring. Bit 19 to 13 are unused, but they wrap around to the same location in a remote SPM. A dedicated bit, number 19, is implemented to distinguish the two types of traffic. Setting the bit indicates credit traffic is present on the PLB, which should be stored in the credit buffer. A demultiplexer uses this bit and places the traffic in the right ring buffer.

A multiplexer is connected to the output of both buffers, which allows a selection to be made of which one will be connected to the ring. Fairness can be guaranteed by means of a traffic shaper, selecting one of the two buffers.

Table 5.1: Nebula ring address map.

Bits	31 - 29	28	27 - 20	19 - 13	12 - 0
Value	PLB memory map	Config	Target μB or accelerator ID	Unused	Scratch-pad memory address

The largest contribution to the hardware cost of the bridge between the PLB and the ring (*plb2ring*) is the NI buffer with a default depth of four places. Hardware costs will approximately double with the addition of the second buffer. However, it is shown in Chapter 4 that the throughput of credit traffic is at least a factor two lower than of data traffic. The credit buffer can be two times less deep, such that the size of the *plb2ring* will only increase by about 50%.

In general, out-of-order delivery of packets is a negative property of a NoC, as additional processing is required to order the packets. Separating data from credit traffic, is also a kind of out-of-order behavior, as data can take over credits, or the other way around. However, this is not a problem in the Nebula ring and is overcome by the relaxed memory consistency model in combination with the CFIFO library [40]. The data and credit in the buffer in one NI are part of entirely different software FIFOs. There is no negative influence when these two types of traffic overtake each other. Credits that are delayed by overtaking data packets, have no negative influence as they already represent a pessimistic and delayed copy of the FIFO administration. It would be a problem if read pointers and data are reordered, as the consumer can then work with old or invalid data. This is prevented, because read pointers are also seen as data traffic.

These hardware additions are supported in software (the CFIFO library) by changing the constructor of a write end-point of a FIFO. The credit bit is set for the return address of credits. A macro is used to perform this action; `MAKE_NOC_CREDIT_PTR(data_ptr)`. For backwards compatibility in software, it is only executed when it is defined that data and credit traffic need to be split (`#define CFIFO_SPLIT_CREDIT`).

As stated before, the traffic shaper is meant to select one of the two buffers as a source of packets for the ring. It implements the credit bandwidth limiter that has been chosen; only inserting credits in the owned slot. The previous work-conserving feature, meant for BE traffic, is discarded as it is replaced by a new GS feature, which will be explained in the next section (Section 5.2).

Rules have been defined to guarantee a certain latency and throughput for the two traffic types, as stated in Table 5.2. Compared to the previous version, a counter has been introduced to each NI to track the credit period, as an NI is only allowed to insert one credit onto the ring during each period. This counter is reset each time a credit is inserted, and is incremented every time the owned slot passes. Only when it reaches its maximum value, which corresponds to the maximum duration of the credit period, a credit may be inserted onto the ring to limit the credit bandwidth.

A check for availability for the own slot of an NI is not required. Other NIs are not allowed to block the owner, as work-conserving property of the fairness protocol is not used. A new work-conserving feature has been introduced, which allows data to be inserted into an owned ring slot when no credits are available in the credit buffer. New credits experience a low latency as the credit buffer is empty, and the credit

Table 5.2: Ruleset of a *ring link* in the data and credit separated Nebula ring

1. Incoming packets addressed to the local node are ejected to the local node.
2. Incoming packets with another destination are passed to the neighbor router.
3. When the current slot belongs to the local node, because of a matching ID, it can always be used to inject a packet onto the ring.
Data and credit period counter rules:
1. The counter is incremented when the slot ID is equal to the local node ID, until its value is equal to its maximum.
2. When the counter is at it maximum and that slot passes again, a waiting credit is inserted onto the ring, and the counter is reset.
3. When the credit buffer is empty, a data packet can be inserted onto the ring instead.

period counter will likely be at its maximum value, allowing credits to be inserted into the next owned slot that is passing.

In hardware, these rules only add a small counter to the NI. Other credit bandwidth limiting options will probably cost more FPGA resources. Appendix B contains the VHSIC Hardware Description Language (VHDL) source code of the hardware implementation.

The principle of using a credit period resembles a Polling Server [12, Section 5.4] scheduling aperiodic tasks amongst period tasks. The server is characterized by its period P_s, equal to the credit period, and the computation time C_s, equal to the one credit that can be inserted every period. Credits act as aperiodic tasks because of their lower bandwidth compared to data traffic.

A Polling Server has a low complexity compared to more advanced servers like Deferrable Server, Priority Exchange, Sporadic Server and Slack Stealer servers. This is important for a small FPGA implementation of the traffic shaper, which acts like a Polling Server.

An improvement in average response time can be accomplished by preserving the server capacity when no credits are present in the buffer. A Sporadic Server for example, preserves its capacity and replenishes it exactly a period after the budget has been consumed. This replenish mechanism still ensures that only one credit can be transmitted onto the ring during every credit period.

5.1.3 Data flow model

The implemented credit bandwidth limiter can easily be modeled with an HSDF graph. The credit period, P_c, in which only one credit may be inserted in the owned ring slot, acts as a bandwidth limiting actor. The maximum latency is experienced when a credit is inserted and a new credit arrives immediately during the next clock cycle. That credit then experiences a delay of the duration of the credit period minus one, as one clock cycle is already elapsed. The resulting HSDF graph is shown in Figure 5.2.

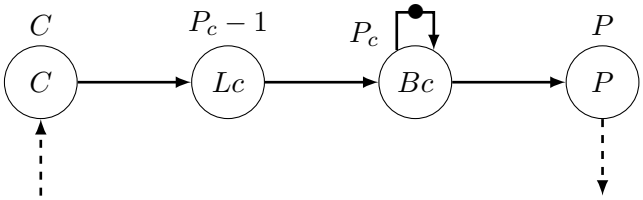

Figure 5.2: HSDF graph of credit mechanism

5.2 Data

Improving the available GS bandwidth cannot easily be accomplished in the connectionless Nebula ring without large hardware costs, when for example doubling the size of all ring links. Random traffic can travel over each distance, such that guarantees are only possible when the maximum communication distance is assumed. Only if more information about the communication pattern of nodes on the ring is known, a higher GS performance can be guaranteed.

The communication pattern follows from the requirements of the tasks that run on the Starburst platform. Besides that, the placement of these tasks strongly determines the usage of certain ring links. When subsequent tasks in a streaming application are also mapped subsequently on the ring, the average communication distance is low, therefore limiting the network usage. In the optimal case, all communication is performed over a distance of a single ring link, allowing all ring slots to be used by all ring nodes, increasing the guaranteed bandwidth by a factor N compared to the present fairness protocol. Mapping thus strongly affects the available bandwidth and should be used to improve the guaranteed bandwidth available for data traffic on the ring.

Techniques that make used to the mapping of task to improve the data traffic bandwidth will be the topic of Section 5.2.1, where a number of implementations are discussed. One of these implementations is chosen and further detailed in Section 5.2.2, after which a data flow model follows in Section 5.2.3.

5.2.1 Distributing data bandwidth

How the mapping information can be used to implement a way to distribute the ring data bandwidth is explained in this section. The implementation will hold on to the current idea of ring slots, as a basis for a fair and predictable system.

Mapping information can be used to determine data streams that do not overlap. In other words, communication channels using different ring links. Examples of such non-overlapping data streams are a stream from ring node 0 to 2 and one from node 3 to 4, as can be seen in Figure 5.3. Node 0 can always use its own slots, whereas it is now also guaranteed to be available for node 3 to use the slot owned by node 0, as node 0 will never send a data packet across the link between node 3 and 4. This guaranteed availability can then also be expressed in a SDF model of the ring.

The development of a task mapping algorithm is out of the scope of this thesis. However, when tasks have already been mapped to processor tiles on the platform, the communication channels between the tasks can be identified. The ring can then

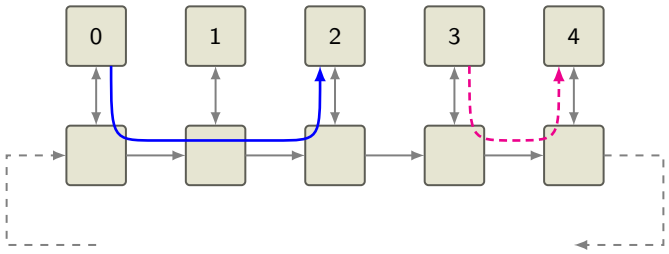

Figure 5.3: A ring network onto which two non-overlapping data streams are mapped, one between node 0 and 2, and one between node 3 and 4.

be configured accordingly to make use of the non-overlapping data streams. Predictability can be ensured by predefining which slots can be used as a communication channel.

In a connectionless interconnect (Section 1.4) like the Nebula ring, no separate NI buffer is present for every outgoing communication channel. All these channels are going through the same NI buffer, to which the network provides a fixed throughput and latency bound. It is therefore not possible to fully utilize the existence of non-overlapping data streams. All outbound traffic of an NI should be aggregated, and the stream with the maximum communication distance should be used to determine if there are other NIs with non-overlapping streams.

In the worst-case, every NI is transferring a data stream over the complete ring distance, preventing the existence of non-overlapping data communication channels. Every NI can then still use its own slots to provide the same network guarantees as the present fairness protocol. This large distance communication should however no usually be the case for streaming applications.

The use-case of the ring would be that the calculation of the slot distribution for a certain application is performed offline in software by the task mapping tooling. Currently, this step has to be performed manually, because the task mapping tool (Omphale) is only compatible with an old version of the Starburst platform. What results from this step is a list of slots that may be used per NI. During the boot process of an application on the platform, all NIs should be configured according to the pre-calculated lists.

The configuration itself should set a register in an NI; the slot-mask register. This slot-mask contains one bit per ring slot and states whether a slot may be used by the NI for data traffic. The register should be programmable via software, however, only when all NIs are in their reset state, where NIs only use their own slot. It is also possible to return to a reset state by programming the masks to a state where an NI can only use its own slot. Otherwise errors can be introduced at the moment of switching to another applications, as a few NIs will already be updated while others are working with an old slot-mask.

Multiple implementations of the slot-mask in combination with the slot IDs on the ring are possible to reach the desired predictable behavior. In the following three sections, implementations will be described and next a trade-off will be made based on their power consumption, required FPGA resources (registers and LUTs) and maximum clock frequency.

Slot ID on the ring

The idea that resembles the current situation the most is to continue using slot IDs on the ring links and add a slot-mask register in each NI. The slots must be numbered consecutively, such that a multiplexer can be fed with the current slot ID to select a single mask bit. The output signal of the multiplexer states if the current slot can be used for data traffic.

The hardware costs per ring node of this option will be dominated mainly by the registers in an NI, the slot-mask (N-bits), and ring slot ID ($log_2 N$-bits) and to a lesser extend, the N-to-1 multiplexer. Especially up to a ring size of 16 nodes, this implementation can be quite efficient as the multiplexer can be implemented in the FPGA using one LUT per 4 bits [59]. For larger ring sizes, no direct connection is available between the LUTs to form a larger multiplexer, potentially limiting the operating frequency of the ring.

The power of the design can be predicted by calculating the average number of bit-flips in the NI and ring registers. The slot-mask register remains constant, but the slot ID traversing the ring constantly switches. For consecutive slot numbers, it can be shown that the number of bit flips approaches two per clock cycle per ring link.

An improvement in power efficiency can be obtained by applying Gray coding. The code results in bit patterns of two successive numbers that only differ in one bit. The number of bit flips is reduced by a factor two (1 bit flip per cycle). For a power-of-two number of ring node, this coding reorders the ring slots, because decoding it in each NI will be too expensive. Reordering however, will have no negative consequence as the ring guarantees do not depend on the order of the ring slots and a mapping tool could also take this into account.

Gray coding is applicable on different ring sizes, although the power efficiency is worst in that case. It should namely be prevented that the encoded bits exceed the range of addresses of the multiplexer. For a non-power-of-two size, it is possible to encode the number until the highest power-of-two, including at least half of all numbers. The remaining numbers are not encoded, and therefore increase the number of flips. Using this strategy, the number for flips increases from one, for a power-of-two, to $\leqslant 1.5$ for the next power-of-two minus one.

One-hot encoding on the ring

Another option is to encode the slots on the ring using one-hot encoding. Every slot is uniquely defined by a single bit in each ring link. One-hot encoding ensures only one of these slot bits will be high. Inside the NI only very simple logic is required to compare the one-hot encoded slots to the slot-mask. The credit slot can be identified by tapping of one of the one-hot encoded signals, whereas a tree of AND-gates or something similar can be used to compare the slot-mask to the current slot to see if a data slot is present.

Hardware wise, this type of encoding results in an increase in the number of registers. Instead of $log_2 N$-bits, a N-bits wide register is required in every ring link. The compare logic, an AND-tree, can be smaller compared to the large multiplexer required for the previous design option. When implemented on an FPGA however, this might not be the case. For a ring of 16 nodes for example, a 16:1 multiplexer

would occupy one FPGA Slice, whereas a 2*16 input AND-tree (composed of 6-input LUTs) would require seven LUTs, a little bit less than two Slices [59].

When taking this inefficiency into account, the operating frequency can be lower than the previous option, although one-hot encoding typically results in a faster clock rate. This is the result of the additional delay caused by the wiring between two Slices. An Application-Specific Integrated Circuit (ASIC) implementation can be faster, and moreover, it is easy to add a pipeline stage in the AND-tree.

The power consumption of the one-hot encoded slots is easily predictable. Because during every clock cycle one flip-flop is set and one reset, 2 bit flips occur per cycle in every ring link.

Circular shift register in the NI

The last option is to use a local circular shift register in every NI. The shift register switches its output every clock cycles to the next slot-mask bit. A counter can be used to check if the own credit slot is currently available.

A shift register can very efficiently be implemented in a FPGA. A single LUT can contain a 32-bit shift register, and cascading them is possible, to create a 128-bit shift register inside a Slice [59]. However, there is a problem as it is not possible to load a new slot-mask into the shift register in parallel, because the individual shift register bit cannot be set or reset.

Since this implementation would be useless without a way to load a slot-mask, a few additions are necessary. A slot-mask register is added, meant to temporarily hold new masks. Only when the shift register is in its credit slot position, the content of the mask is copied to the shift register, preventing the need for an expensive barrel shifter. With these additions, this option uses the most registers; N-bits for the temporary slot mask, N-bits for the shift register itself and several control registers, such as the ones to implement the counter.

A high operating frequency is possible for this option as the slot system is implemented locally on each node on the ring. Therefore it will degrade less than the other option for increasing system sizes. The locality of the slots also has a disadvantage, as it is very important that all shift register in all NIs remain synchronized, otherwise multiple NI can incorrectly claim the *same* slot.

The number of bit flips cannot directly be derived from the hardware architecture, since it depends on the mask value in the shift register. Unless zero or all slots are set in the slot-mask, the number of flips will be higher than that of the other designs.

5.2.2 Implementation

Before one of the options is implemented within the Nebula ring, all options are first synthesized without other ring components. This should give a idea about the correctness of the predicted area, power and operating frequency, which gives the foundation for the final choice.

Table 5.3 shows the number of Slice registers and LUTs obtained by synthesizing the designs for a ring consisting of 32 nodes. Moreover, the number of bit flips

is stated such that it is comparable to the power estimation resulting from the Xilinx XPower Analyzer software. Finally, the maximum operating frequency of each design is presented, which was obtained after Place & Route.

Table 5.3: Synthesis results for a ring consisting of 32 nodes

Implementation	Registers	LUTs	Bit flips/clk	Power (mW)	Max. post-par (MHz)
One-hot	2048	449	2	2.60	838
Shift register	2240	1313	#slots	5.08	589
ID	1184	384	2	2.79	1011
ID (Gray)	1184	384	1	1.42	1011

The results show the number of registers matches the expectation, also for a larger ring of 64 nodes, although these numbers are not shown here. The shift register design uses much more LUTs than expected, and is overall the worst option. A clear relation exists between the number of bit flips and the power consumption of the circuits. The tooling used to obtain these power estimates internally also makes use of toggle rates of registers, originating from simulation data, which explains the relation between the two.

Overall, the preferred design is using a Gray coded slot ID on the ring. It uses the least amount of FPGA resources, consumes the least power and can operate at a high clock frequency.

This slot mechanism is integrated into the Nebula ring. The slot-mask register in an NI can be programmed by every node by sending a special packet across the ring to the NI. Like the credit implementation, a dedicated bit is used to distinguish this special configuration packet: bit number 18 (see Table 5.1 for the complete ring address map). Only a maximum of 32 ring nodes is currently supported, as this corresponds to the word size of 32 bits. The configuration itself can be performed in software by writing the new slot-mask to the address calculated by a macro (`MAKE_NOC_RINGLINK_CONFIG_PTR(core)`).

5.2.3 Data flow model

The changes to the way data traffic is transported can also be expressed in the SDF graph shown in Figure 4.6. Once multiple slots are available for an NI, this should lead to a higher throughput, a lower firing duration of the bandwidth limiting actor in Figure 4.7.

The usage of multiple ring slots by an NI can be modeled by changing the cycles of bandwidth limiting actors (in Figure 4.6). When multiple slots are available, data packets can be transferred in parallel over multiple slots. Every slot can then be used once every N clock cycles. It would be logical to model every slot separately with bandwidth limiting actors, a cycle to connect them and a single token to only allow a single data packet to be inserted into a slot. The large cycle of bandwidth limiting actors in Figure 4.6 is split into several smaller ones, equal to the number of slots that is available, which is shown in Figure 5.4. The bandwidth within one slot is independent on the available bandwidth in another slot, therefor the model

uses independent cycles of bandwidth limiting actor to model the limitation in each slot.

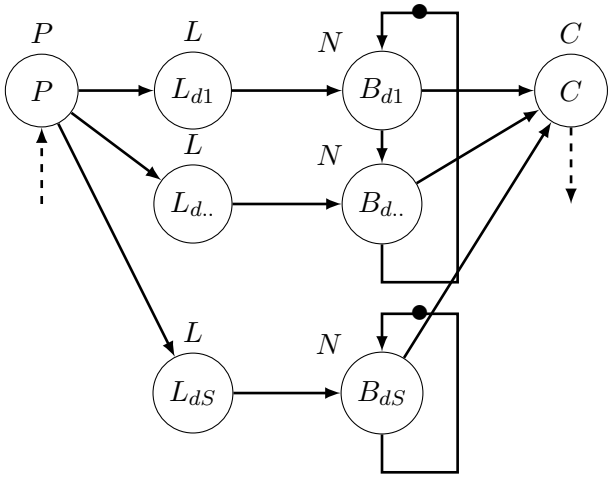

Figure 5.4: HSDF graph of a distributed data bandwidth mechanism

The S data packets the producer sends across the ring for every CFIFO entry are distributed among the available slots. The exact order of how these packets are inserted into a slot is of no importance and is unknown, since the current state of the ring is not assumed to be known. What does influence the model, is the number of data packets transmitted over each ring slot, as each slot is limited in bandwidth.

Since all S data packets are produced at the same time, the NI can and will use all available slots according to its slot-mask. Data packets cannot be reordered between the producer and consumer, because all packets pass through the same FIFO and once a packet leaves the FIFO and enters the ring, the latency to the consumer is fixed preventing other data packets from overtaking it. It can then be guaranteed[4] that a maximum number of packets is transmitted from the FIFO over each ring slot equal to $\lceil \frac{S}{slots} \rceil$. Since the bandwidth and latency guarantees for every slot are the same, it has no negative influence to assume this maximum number of data packets is transmitted over every slot, as the consumer must always wait for the last token to arrive at its input. For every slot, a parallel path is created in the HSDF graph from the producer to the consumer, which are all equivalent. The consumer takes the maximum arrival time of equal maximum arrival times for every slot, so effectively it is sufficient to only take the arrival time of one of the (equivalent) producer-consumer slot paths into account. Every slot will therefore result in the same worst-case execution time, although in practice different numbers of packets will be transmitted over the available ring slots.

Like the original HSDF graph, a conversion can be made to reduce the number of actors, to make the model easier to analyze. It is only required to include the worst-case performance of a single slot in the reduced graph, as all offer the same performance, and therefore result in the same maximum arrival time of input tokens at the consumer. This reduced graph is shown in Figure 5.5. As expected, the number of available ring slots directly increases the maximum throughput of the bandwidth limiting actor B_d.

[4] although it requires a more thorough justification to prove the correctness of this maximum number of packet with respect to the HSDF model

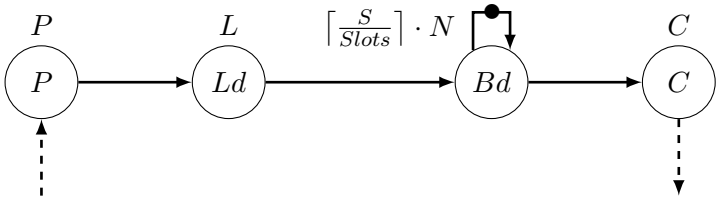

Figure 5.5: Reduced HDSF graph of Figure 5.4

The worst-case latency, modeled in the firing duration of the L_d-actor, improves a little when using multiple ring slots. An NI no longer needs to wait for one specific slot to pass, but can choose between a number of slots. It would be beneficial to for example place two slots on the opposite side of the ring to reduce the latency by a factor two. However, this will make the task mapping algorithm more complex. The original latency of $N - 1$ for one slot is reduced to $N - slots$ in case the slots' positions on the ring are unknown.

5.3 Data and credit combined

A final step is to combine the system to limit the credit bandwidth and the system to distribute data bandwidth. Until now it has been assumed that all the ring bandwidth could be used for data traffic. However, a limited amount of bandwidth should be reserved for credit traffic. This section explains how the total system can be constructed from the two chosen ideas, presented previously in this chapter.

The credit bandwidth is already limited and remains the same after extending the ring with multiple usable data slots. The analysis model for credits, shown in the HSDF graph in Figure 5.2, remains valid.

Data traffic on the other hand, experiences a reduction in bandwidth and a increase in latency caused by the addition of the guaranteed credit bandwidth. This bandwidth reduction is modeled by an additional actor, B_{dc}, in each slot cycle. The firing duration ($\rho(B_{dc})$) of this actor is a multiple of N and is based on the number of packets per slot and the credit period, such that it represents the maximum credit bandwidth used in that slot.

The firing duration is to be be calculated based on the number of data packets sent over a slot:

$$\left\lceil \frac{S}{Slots} \right\rceil$$

The maximum number of credits that could be claiming the slots follows from the credit period, P_c:

$$\left\lceil \frac{S \cdot N}{Slots \cdot P_c} \right\rceil$$

which is multiplied by N, as the credit period is selected to be a multiple of N. Converting this number of credits only involves a multiplication with the ring round-trip time, N:

$$\rho(B_{dc}) = \left\lceil \frac{S \cdot N}{Slots \cdot P_c} \right\rceil \cdot N \tag{5.1}$$

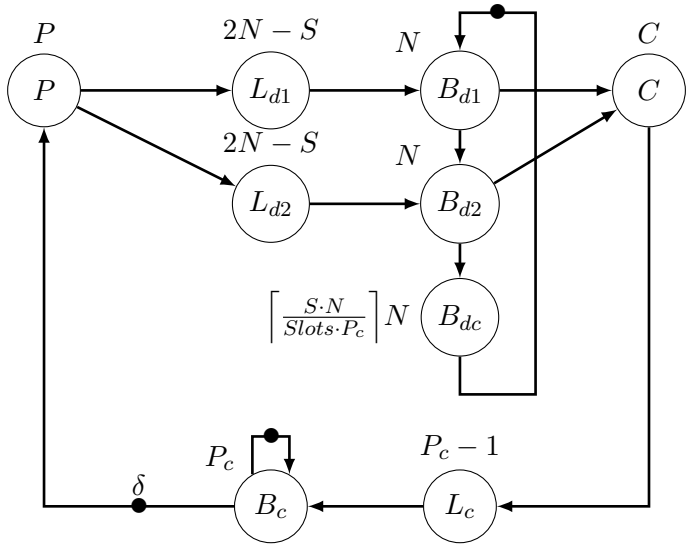

Figure 5.6: HSDF graph of the complete CFIFO communication chain over the Nebula ring

Data packets also experience an increase in their latency, $\rho(Ld)$. This is caused by the fact that a credit can be inserted in a ring slot, preventing it to be used for data traffic for a maximum of one round trip of the ring. After P_c clock cycles it is available for credits again. In between, data packets may use the ring slot. Figure 5.6 shows the HSDF graph of the combined data and credit graph where these changes are included.

The same reduction can be performed on the graph as used in Section 4.2 and Section 5.2.3. The only addition that needs to be included is the credit bandwidth actor B_{dc}, reducing the available data bandwidth. This will translate in an increase of the firing duration of the B_d actor. The new duration is composed of the throughput of data packets and a latency due to credits using the same ring slot. Therefore, the firing duration of B_d, as explained in Section 5.2.3, is increased with the credit latency, which is the result of Equation 5.1. The resulting firing duration $\rho(B_d)$ is:

$$\rho(B_d) = \left(\left\lceil \frac{S}{Slots} \right\rceil + \left\lceil \frac{S \cdot N}{Slots \cdot P_c} \right\rceil \right) \cdot N = \left\lceil \left\lceil \frac{S}{Slots} \right\rceil \left(1 + \frac{N}{P_c} \right) \right\rceil \cdot N$$

The graph resulting from this reduction is presented in Figure 5.7. The expression for the MCM of the credit and data split can be derived from the graph by using Equation 4.1. The following expression for the MCM is obtained:

$$\begin{aligned} MCM &= \max \left(\tfrac{\rho(P)+L_d+B_d+\rho(C)+L_c+B_d}{\delta}, \tfrac{\rho(P)}{1}, \tfrac{\rho(C)}{1}, \tfrac{\rho(B_d)}{1}, \tfrac{\rho(B_l)}{1} \right) \\ &= \max \left(\begin{array}{l} \tfrac{\rho(P)+2N-S+D+\left\lceil \left\lceil \frac{S}{Slots} \right\rceil \left(1+\frac{N}{P_c}\right) \right\rceil \cdot N+\rho(C)+P_c-1+P_c}{\delta}, \\ \tfrac{\rho(P)}{1}, \tfrac{\rho(C)}{1}, \tfrac{\left\lceil \left\lceil \frac{S}{Slots} \right\rceil \left(1+\frac{N}{P_c}\right) \right\rceil \cdot N}{1}, \tfrac{P_c}{1} \end{array} \right) \end{aligned} \tag{5.2}$$

The MCM expression will be used in the next chapter to evaluate the performance of the Nebula ring, with different guarantees for the data and credit traffic.

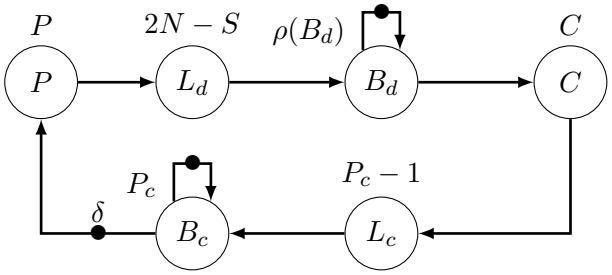

Figure 5.7: Reduced HDSF graph of Figure 5.6

CHAPTER 6

Evaluation

The implementation of the improvements to the inter-core communication method are shown in the previous chapter. This implementation is to be evaluated in this chapter. It starts by looking at the hardware cost metric in Section 6.1, to see if the increase in hardware is reasonable compared to the additional features that are implemented.

The (dis)advantages of using task mapping information are analyzed in Section 6.2. A comparison is made between the performances of the inter-core communication versus that of the previous architecture (which supported fairness, but made no distinction between data and credit traffic on the ring network).

To also verify the performance in a real-life context, a case study is performed. A demo application was available, developed by Dekens, which decodes a PAL signal and displays it on a screen. This application is mapped to the improved communication method and the results are discussed in Section 6.3.

6.1 Hardware costs

One of the goals of the additions to the ring was that is should provide a low cost solution to communication problems. In order to see if the credit and data split implementation satisfies this constraint, the hardware costs of it will be evaluated in this section.

To place the hardware cost into context, a comparison is made to the following ring designs: the original ring, fairness and fairness plus flow control. Especially the cost of the fairness ring will be interesting, as it offers about the same feature, however it can only guarantee a relatively low data throughput. Flow control is added to the comparison, since it shows the effect of adding a second ring (the ACK-ring) to the design, although that one is very small, because it does not transport wide data packets.

A fair comparison is obtained by configuring the MPSoC with exactly the same parameters. All systems include 16 μB cores (and a Linux core), the number that is required to run the PAL-demo (Section 6.3). These cores are present on tiles containing 4 kB of local memory, 8 kB SPM on the LMB and 32 kB for the caches. Synthesis is performed for a frequency of 100 MHz for the tiles and the interconnect, the arbitration tree and the ring.

The Xilinx ML605 evaluation board, discussed in Section 3.2, is used as hardware platform for the MPSoC. The most important hardware resources it contains in a limited number are: slice registers, LUTs, LUT RAM, LUTs used as RAMs (LUTRAMs), Block RAM, dedicated RAM in Virtex-6s (BRAMs) and Digital Signal Processing element, special Virtex-6 slices (DSP48E1s). Each slice in the FPGA contains eight registers and four LUTs. These LUTs contain six independent inputs and two outputs in which a function generator can implement any (six-input) boolean function. Within specific slices, LUTs can also be configured as a synchronous Random Access Memory (RAM) resource. These distributed RAM elements can for example implement a single-port 256 x 1-bit RAM. BRAMs are another memory resource that can implement up-to 36 Kb of dedicated RAM. It is an efficient data storage resource for, for example, FIFO buffers [58]. The last resources are the DSP48E1 slices, meant to efficiently implement high throughput multiply and add functionality.

Hardware costs of individual ring components, as shown in Figure 3.6, will now be evaluated. The ring components of the credit and data split are a *plb2ring* and a *ring link*. As a reference, the hardware costs of the large components in the Starburst MPSoC are presented in Table 6.1: a μB processor, a Linux μB which also included a MMU and a Multi-Port Memory Controller (MPMC).

Table 6.1: Hardware usage of reference components: Starburst MicroBlaze (i.e. all non-Linux MicroBlazes), Linux MicroBlaze and Multi-Port Memory Controller (MPMC).

	Starburst μB	Linux μB	MPMC
Slice reg	3089	3418	4759
LUTs	3949	4507	3315
LUTRAM	313	236	188
BRAM or FIFO	18	19	17
DSP48E1	6	6	0

The first ring component to be analyzed is the *plb2ring*, as a large resource increase is expected due to the addition of a second (credit) buffer. By default the first buffer has a depth of four places; the credit buffer is configured at the same depth. Next to the buffer, a small increase in hardware compared to that of the fairness ring is expected because a number of multiplexers is added.

Table 6.2: Hardware usage of the averaged *plb2ring* with a FIFO of four places deep. The percentages show the relative size with respect to a Starburst MicroBlaze.

	Original		Fairness		Fairness and flow-control		Split credit and data	
Slice reg	6	(0.2%)	6	(0.2%)	6	(0.2%)	10	(0.3%)
LUTs	74	(1.9%)	74	(1.9%)	85	(2.2%)	138	(3.6%)
LUTRAM	56	(17.9%)	56	(17.9%)	68	(21.7%)	114	(36.4%)

The synthesis results for the *plb2ring* are shown in Table 6.2, where the percentages indicate the relative size of the component with respect to the reference Starburst μB. The last column in the table shows that the hardware costs are as expected; doubling the number of buffers roughly doubles the number of LUTs and LUTRAMs.

An interesting observation is obtained when the size of the credit buffer is changed. Because the throughput of the credit buffer will at least be a factor two lower than that of the data buffer, its size has been decreased to two during a synthesis run. Its results showed that the size of the buffers do not influence the total hardware costs of the *plb2ring*. This can be explained by the fact that both buffers are implemented as shift registers using a dedicated FPGA primitive (the SRLC16E [59]). The primitive support shift register with a depth of 1 up to 32 using only a single LUT. Hardware costs will barely increase when the depth of both buffers is increased to 32.

The *ring link* is the other ring component where changes have been made to update the insertion protocol. A traffic shaper is implemented within this block, as a counter in combination with multiplexers, and the slot mask register is added.

Table 6.3: Hardware usage of the averaged *ring link*. The percentages show the relative size with respect to a Starburst MicroBlaze.

	Original		Fairness		Fairness and flow-control		Split credit and data	
Slice reg	57	(1.8%)	62	(2.0%)	90	(2.9%)	83	(2.7%)
LUTs	58	(1.5%)	63	(1.6%)	80	(2.0%)	85	(2.2%)
LUTRAM	0	NA	0	NA	0	NA	0	NA

Table 6.3 shows that the credit / data implementation is larger than that of the fairness design, as it contains approximately the same functionality, but has more features. The 17 bits slot-mask and 4 bits counter account for the increase in Slice registers. With these additions, the ring is approximately the same size as the one that offers fairness and flow control.

Flow-control, however, mainly adds hardware in the form of a *ring shell* component, which is not required for the other ring designs. The size of a *ring shell* is large in comparison to a *ring link*, and depends on the component connected to it. The exact hardware costs of the *ring shell*, as presented in [26], are shown in Table 6.4.

Table 6.4: Hardware usage the *ring shell* for different ring entities. The percentages show the size relative to a Starburst MicroBlaze.

	μB tile		Accelerator		FSL	
Slice reg	203	(6.6%)	234	(7.6%)	233	(7.5%)
LUTs	133	(3.4%)	199	(5.0%)	155	(3.9%)

The result presented in the previous tables are now summarized in Table 6.5 to show the hardware costs of the complete ring per ring node. It shows that the credit and data split is more hardware efficient than doubling the fairness ring, except for LUTRAM, because there already is a need to double the number of buffers in the *plb2ring* of the credit and data split design. Doubling the ring, however, offers a higher throughput as each ring is not shared between credit and data in that case.

The design is smaller than the one with flow control. This is mainly caused by the large number of configuration registers in the *ring shell* ($5 \cdot 32$ bit), requiring many slice registers.

The total hardware resources used in a 16 core Starburst MPSoC are presented in Table 6.6. It shows that a high amount of BRAMs and LUTs are used within

Table 6.5: Total hardware usage of the average ring node. The percentages show the relative size with respect to the original ring.

	Original		Fairness		Fairness and flow-control		Split credit and data	
Slice reg	63	+0.0%	68	+7.9%	299	+374.6%	93	+47.6%
LUTs	132	+0.0%	137	+3.8%	298	+125.8%	223	+68.9%
LUTRAM	56	+0.0%	56	+0.0%	68	+21.4%	114	+103.6%

the platform, whereas there are plenty of slice registers and LUTRAMs left. These numbers also indicate that the number of cores of the platform can probably be doubled at least (the maximum number of cores is currently 32 when using smaller caches) when switching to a newer FPGA evaluation board like the Xilinx Virtex-7 FPGA VC709, which contains about three times as many resources.

Table 6.6: Total hardware usage of a 16 core Starburst MPSoC, with different ring configurations, compared to the hardware resources available in the Xilinx ML605 FPGA evaluation board.

	Available resources		Fairness		Fairness and flow-control		Split credit and data	
Slice reg	301,440	100.0%	73,354	24%	78,422	26%	73,761	24%
LUTs	150,720	100.0%	91,255	60%	95,505	63%	93,020	61%
LUTRAM	58,400	100.0%	8,989	15%	9,243	15%	9,974	17%
BRAM or FIFO	416	100.0%	346	83%	346	83%	346	83%
DSP48E1	768	100.0%	102	13%	102	13%	102	13%

6.2 Credit and data split vs Fairness

To show the (dis)advantages of using task mapping information and splitting the data and credits, an evaluation is performed in this section. The expression of the MCM is used to evaluate several scenarios, and its results are compared to that of previous versions of the ring providing fairness.

The expression for the MCM of the credit and data split has been derived in Equation 5.2 for the graph presented in Figure 5.7. The following expression for the MCM is obtained:

$$MCM = \max\left(\begin{array}{c} \frac{\rho(P)+2N-S+D+\left\lceil\left\lceil\frac{S}{Slots}\right\rceil\left(1+\frac{N}{P_c}\right)\right\rceil\cdot N+\rho(C)+P_c-1+P_c}{\delta}, \\ \frac{\rho(P)}{1}, \frac{\rho(C)}{1}, \frac{\left\lceil\left\lceil\frac{S}{Slots}\right\rceil\left(1+\frac{N}{P_c}\right)\right\rceil\cdot N}{1}, \frac{P_c}{1} \end{array}\right) \tag{6.1}$$

For the fairness protocol, this expression has already been determined in Equation 4.2:

$$MCM = \max\left(\frac{\rho(P)+\rho(C)+(S+4)N-2}{\delta}, \frac{\rho(P)}{1}, \frac{\rho(C)}{1}, \frac{S*N}{1}, \frac{N}{1}\right) \tag{6.2}$$

The MCM does not relate directly to the worst-case data throughput that can be reached over the ring. It states how many how many cycles it on average takes to

transport a FIFO token across the ring. The size of a FIFO token will therefore also affect the throughput. The packet size (S) determines how many data words can be sent for every single credit returning to the producer. Moreover, one of the words in a packet of size S is a value used to update the write pointer in the FIFO administration of the consumer and will therefore be ignored for throughput results, since we are only interested in data throughput. The throughput can therefore be determined from the MCM by using the following formula: $throughput = \frac{S-1}{MCM} \cdot F$ (samples/s), where F is the clock frequency of the ring (100 MHz) and one sample corresponds to 4 bytes.

First, the influence of the credit period, which determines the ratio between the reserved data and credit bandwidth, on the best-case throughput is shown by evaluating the MCM expression. The credit period will limit the overall throughput in case not enough bandwidth is available for credit traffic and the producer side of a FIFO needs to wait for credits to return before it can sent data words across the ring.

The packet size (S) also influence the choice for the best suitable credit period, because it determines the ratio between required data and credit bandwidth. When multiple ring slots are available to an NI, the credit period will need to decrease, since only the data bandwidth scales with the number of slot, whereas the credit bandwidth remains constant, leading to a mismatch between required and available credit bandwidth which can be compensated by decreasing the credit period. Reducing the credit period also has a negative effect of decreasing the bandwidth left for data traffic, as more bandwidth is reserved for credit traffic.

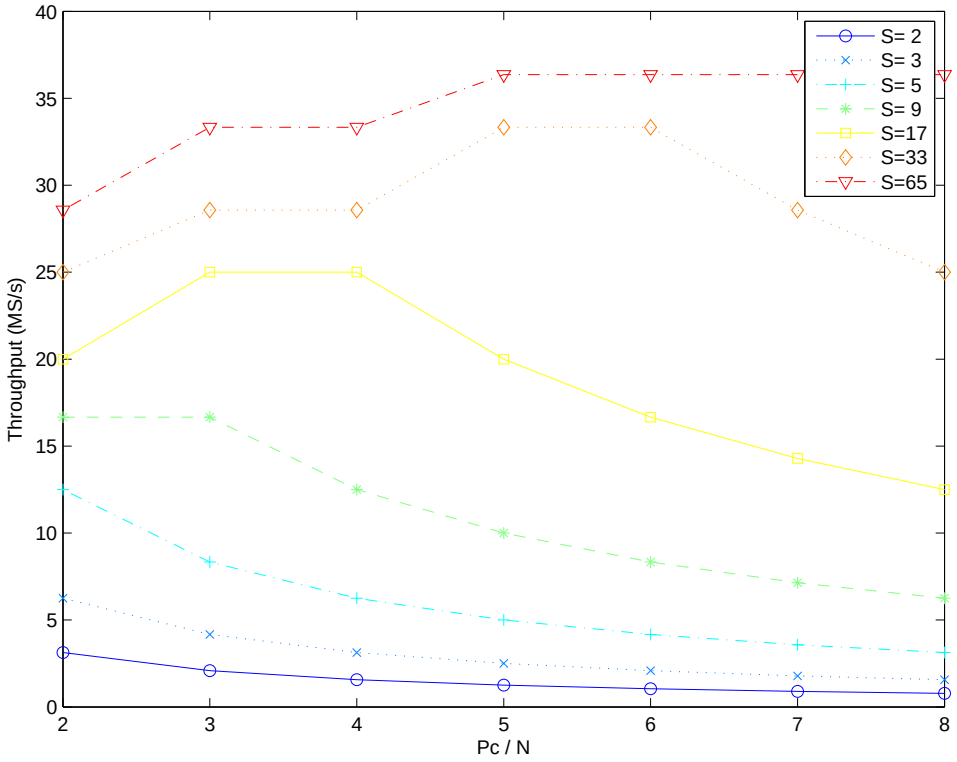

Figure 6.1: Throughput dependency on the credit period (P_c), $\rho(P) = 1$, $\rho(C) = 1$, $N = 16$, $slots = 8$, $\delta = 3$

Figure 6.1 shows the throughput that can be reached when changing the credit period

(P_c) for different packet sizes (S). As can be conducted from the figure, a larger packet size always increases the throughput, as less credits need to be transmitted compared to data words. A possible downside of a large packet size is it increases the end-to-end latency. Furthermore, we can conclude from the figure that a increase in credit period only leads to a higher throughput when a large packet size is used. For a low packet size the throughput decreases when the credit period is increased. The minimum P_c/N is 2, because otherwise ($P_c/N = 1$) all available ring bandwidth is reserved for credits, and no bandwidth is left for data communication.

The effect of the packet size in combination with the number of slots ('Slots' in the equation) will now be evaluated, as Equation 6.1 already suggests both are coupled. The relation between these parameters is plotted in Figure 6.2. The figure shows that the throughput only scales well with the number of slots for large packet sizes. In that case the throughput is limited by the reserved data bandwidth instead of the credit bandwidth. The data bandwidth obviously scales with the number of available slots. It can also be concluded from the figure that it is not useful to allocate many slots when using a small packet size since that does not improve the limiting factor in that case; the credit bandwidth. The guaranteed throughput can therefore only be increased significantly when a large packet size is used and when the application allows many slots to be (re)used.

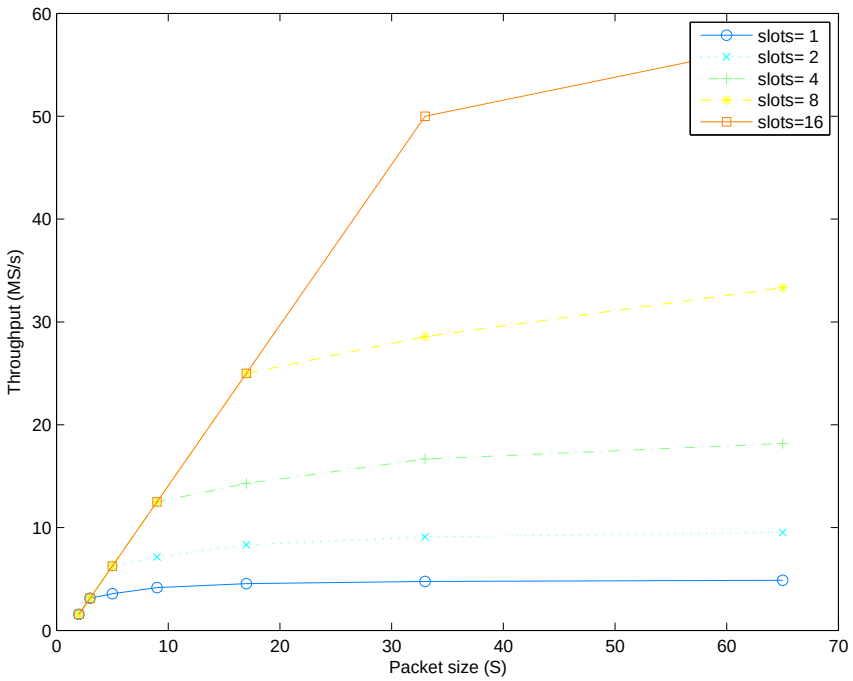

Figure 6.2: Throughput effected by the packet size and number of slots; $\rho(P) = 1$, $\rho(C) = 1$, $N = 16$, $slots = 16$, $\delta = 3$, $P_c = 4 \cdot N$

The best-case throughput of the credit and data ring will now be determined using the analysis model of the implemented ring. The firing duration of the producer and consumer are set to one ($\rho(P)$ and $\rho(C)$) and the producer uses all of the N slots to reach this best-case. The obtained throughput then depends on the packet size and number of initial tokens (δ), which is equal to the FIFO depth.

The throughput keeps increasing almost linearly with the packet size, as can be seen in Figure 6.3, until a certain point ($S = 33$, throughput ≈ 50 MS/s), which depends

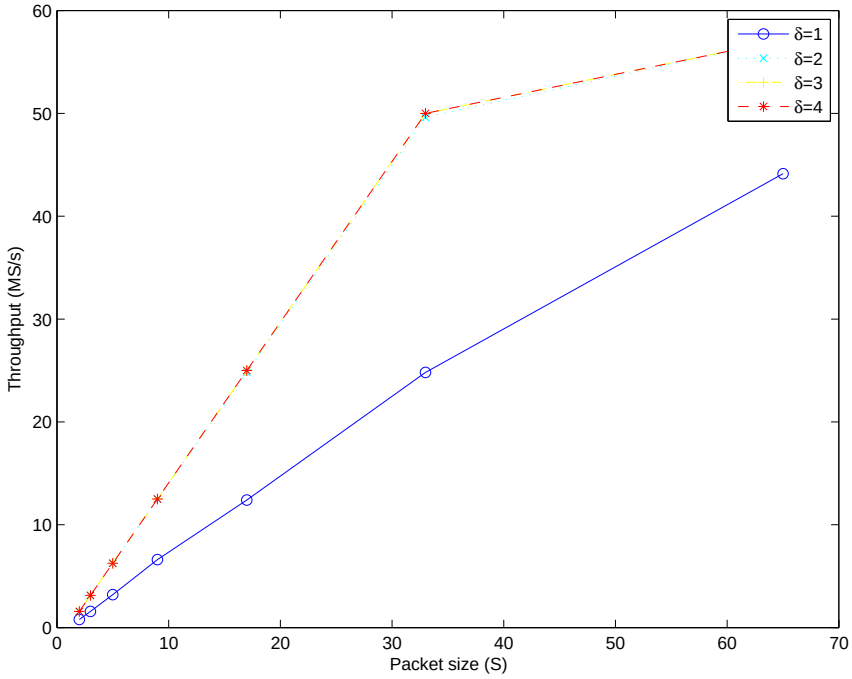

Figure 6.3: Best-case throughput with a credit and data split; $\rho(P) = 1$, $\rho(C) = 1$, $N = 16$, $\delta = 3$, $slots = 16$, $P_c = 4 \cdot N$

on P_c. A minimum of three initial tokens (δ) are required to hide the latencies in the graph, although the different between 2 and 3 tokens is minimal. Increasing the number of tokens further does not affect the performance any more.

The same type of best-case throughput plot is also constructed for the ring with the fairness protocol. Using the same parameters and the MCM as defined in Equation 6.2, the results shown in Figure 6.4 were obtained. The same number of minimal initial tokens follows from the MCM evaluation. However, the throughput for GS traffic on the ring remains low, even when the packet size is increased. For a large packet size, the throughput will be limited by the guaranteed data bandwidth, which cannot be increase in a way like increasing the number of reserved slots for the other implementation.

Finally, a comparison is made between fairness and the credit and data split ring design. The same best-case parameters are used as in the previous figures. Figure 6.5 shows that only when using a very small packet size, the throughput of the fairness protocol is (two times) higher than that of the credit and data split design. From a size of four, the fairness ring is outperformed and at a size of 65, a huge advantage of a 9 time higher throughput is achieved by splitting credit from data traffic and using multiple ring slots.

This comparison shows that large throughput benefits can be obtained when using task mapping information to increase the number of available data slots. Doubling the number of slots (at large packet sizes) leads to almost a double throughput.

These results might also be beneficial for the flow controlled communication with hardware accelerators. Currently, every data word sent by a producing core returns a credit, corresponding to a packet size of one (no administrative FIFO pointer is sent to the accelerator). Although a dedicated ring is used for flow control credits,

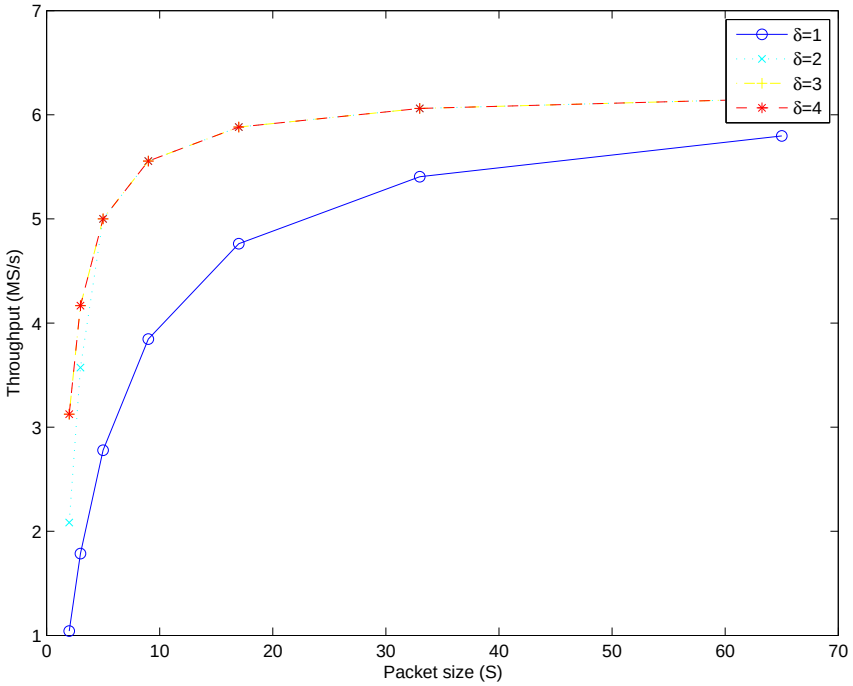

Figure 6.4: Best case throughput with fairness; $\rho(P) = 1$, $\rho(C) = 1$, $N = 16$

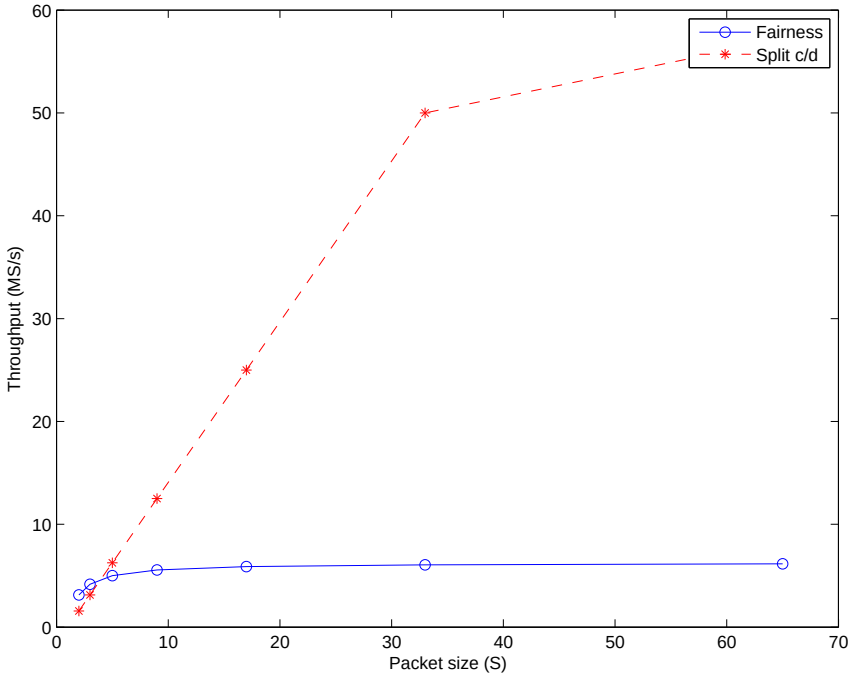

Figure 6.5: Throughput comparison between fairness and the credit and data split; $\rho(P) = 1$, $\rho(C) = 1$, $N = 16$, $slots = 16$, $\delta = 3$, $P_c = 4 \cdot N$

it might still be beneficial to use multiple ring slots for data traffic and also decrease the number of credits to increase the overall throughput. Credits are now limiting the throughput that can be reached when communicating to an accelerator.

6.3 Case study: PAL-demo

In this final section of the evaluation of the implemented ring, a realistic application will be used to verify the ring's performance. A PAL-demo is used to compare the performance of the new ring to that of the previous designed fairness ring. This should show if additional guaranteed bandwidth also increases the performance of the demo, or that fairness with the (BE traffic) work-conserving feature is better suited for it. The case study will also demonstrate how a mapped task graph is used to configure the slot mask of all NIs in the ring.

In the demo, developed specifically for the Starburst platform, a PAL signal is decoded and displayed on a VGA compatible screen. The analogue PAL signal is fully processed in the digital domain. The spectrum of the PAL signal is displayed in Figure 6.6. Only the luminance carrier is used, such that a black and white image is obtained. The carrier over which the color signal is modulated is ignored as well as the audio signal, mainly because of the limited processing power on the Starburst platform.

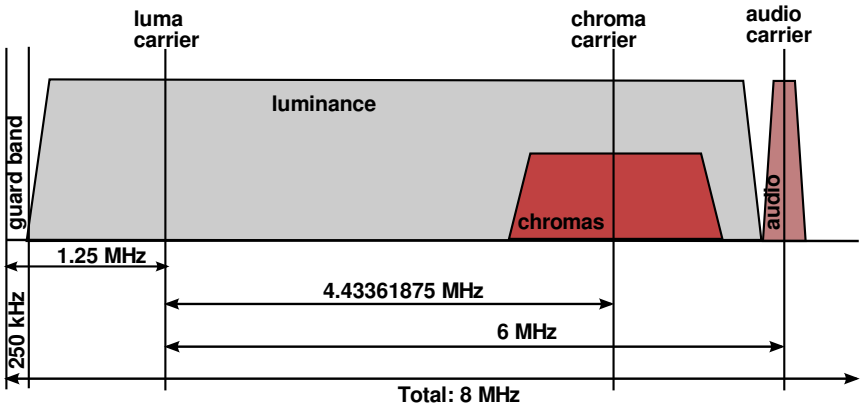

Figure 6.6: The spectrum of a PAL signal

The software based PAL-decoder consists of 8 different tasks, mapped onto a 16-core Starburst system. A single μB is not very powerful and therefore some tasks are running on multiple cores. Another option, to run demanding tasks on a hardware accelerator, has already been studied. An accelerator can for example replace 4 cores on which a I/Q magnitude task is running, for a total (including the additional ring) hardware cost of a bit more than a single μB core [26]. How these 8 tasks are mapped on the 16 different processor tiles, is shown in Figure 6.7.

Data enters the software based system as raw I and Q samples from the Bitshark extension board (see Section 3.1.1 for more information about the board) that is connected to a camera with a PAL output. Another option is to load test samples directly from a CF-card. Between all tasks, communication is performed by using the CFIFO buffers and the Nebula ring. At the end of the processing chain, the resulting image is displayed on a VGA monitor.

In order to run the demo on the new ring implementation, a couple of parameters need to be set: the credit period (P_c) and the slot mask of all NIs. Most tasks communicate to the next one by means of data packets with a size of 64 to 68 words, except for the Automatic Gain Control (AGC) with a size of 18 words, and

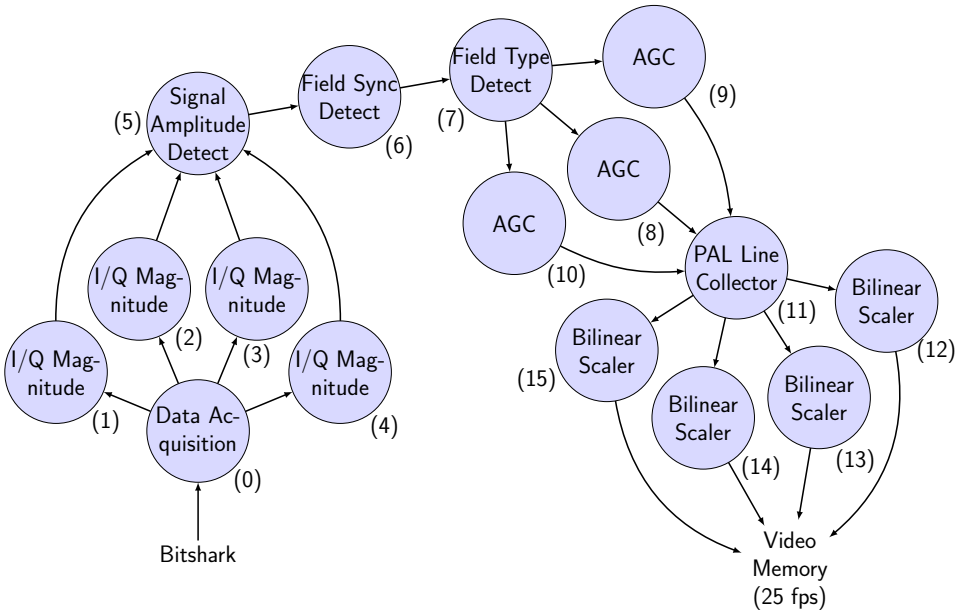

Figure 6.7: PAL-demo task graph

the Line Collector (Liner) with a size of 49 words. As can be seen in Table 6.7, the data rate of the AGC is relatively low, so choosing a high credit period of $5 \cdot N$ will leave enough data bandwidth for the AGC tasks and will result in a high bandwidth for the more bandwidth demanding tasks, which is also shown in Figure 6.1 and Figure 6.2.

The slot mask set in every NI is the other parameter that will be discussed. It sets which slots are available for an NI to insert data words onto the ring. The mapped task graph, shown in Figure 6.7, shows that there are only overlapping data streams where tasks run on multiple cores, and samples are distributed between them. The tasks that are running on a single processor and only send data to the next processor tile, can utilize all ring slots, because no other data stream is using the connecting ring link. For the other tasks, slots are divided based on the data rate of a task as stated in Table 6.7. The capture task, for example, has a data rate of 4 times that of a convert task and therefore is assigned 4 times more slots. The maximum communication distance, also presented in the table, indicates how many ring links are occupied by the task for the assigned number of slots. The number of slots assigned to all the NIs is finally shown in the last column of Table 6.7.

Each slot can provide a throughput of 5.9 MS/s at maximum (100 MHz divided over 17 ring nodes, 16 μB cores and a Linux core). A minimum of two slots should already be sufficient to run the system with an input of 10 MS/s. More slots are assigned however, to reduce the latency in the system and improve the overall throughput.

The PAL-demo will be used to benchmark the performance of the different ring architectures; the one with fairness and the work-conserving property, a ring without separate credit and data buffer using the default ring slot, one with a split between data and credit traffic using the default slots and one with the slot assignment as stated in Table 6.7. The video output is disabled for the benchmark to remove a potential bottleneck in the system when the scalers all write pixels to the shared main memory. The data will be loaded from a pre-recorded source file instead of

Table 6.7: Ring traffic specification for the tasks part of the PAL-demo, for a 10 MS/s input.

Task	μB tile	Data rate (MS/s)	Max. Distance	Assigned slots
Capture	0	10.00	4	8
Convert	1	2.58	4	2
Convert	2	2.58	3	2
Convert	3	2.58	2	2
Convert	4	2.58	1	8
Levels	5	10.23	1	16
Sync	6	10.39	1	16
Type	7	10.54	3	9
AGC	8	0.93	3	3
AGC	9	0.93	2	3
AGC	10	0.93	1	3
Liner	11	8.28	4	12
Scale	12	0.00	0	1
Scale	13	0.00	0	1
Scale	14	0.00	0	1
Scale	15	0.00	0	1

the Bitshark, such that every run is processing the same data. The 3 MS/s of raw data samples are looped 100 times, many orders more than the capacity of all FIFOs in the demo. The rate at which data can be inserted from the source file into the buffers of the I/Q magnitude tasks will then be an indication of the total system throughput, since back-pressure is automatically applied by upstream tasks liming the insertion rate of the I/Q samples.

Table 6.8: Benchmark results of the PAL-demo for different configurations of the Nebula ring.

Configuration	Data rate (MS/s)
Fairness and work-conserving	1.01 - 1.49
Split and default slots	1.49
Split and slots as in Table 6.7	1.50
No split and default slot	1.47

The results of the benchmark are shown in Table 6.8. As can be seen, the fairness and work-conserving protocol has the lowest worst-case performance of 1.01 MS/s. The fairness and work-conserving protocol shows a large different of about 50% in throughput over multiple runs of the benchmark, whereas the other rings provide a very constant throughput. The work-conserving feature might be a cause of this spread in data rate results. Burst of data can block each other introducing additional latency in the system. It is however strange that the case where the same number of guaranteed slot is available (default slots), and no BE feature is enable can lead to a higher throughput. It can be concluded that the demo is not suitable to make a distinction between use usage of BE or GS bandwidth.

The other ring designs mainly provide GS bandwidth and can show the effect on the overall throughput when for example increasing the number of available ring slots. The effect of adding more slots on the average data rate is not as high as anticipated

(1.49 to 1.50 MS/s as shown in Table 6.8). In this application, the μB processors are limiting the rate at which data words are sent to the NI of the ring. The additional bandwidth created by allocating more ring slots is not used. The small decrease in latency, cause by the earlier arrival of reserved slots, can already be the reason for the slightly better performance.

Splitting data and credit traffic over 2 NI buffers leads to a small increase in performance. The additional buffer leads to less stalled cycles of the processor, which would occur when an NI is full and new data is prevented to leave the PLB bus. The increase in latency for credits in the credit buffer does not decrease the overall system throughput, because of the large buffers between all tasks (5-6 places deep, except for duplicated tasks with buffers of 3-4 places deep). For these large buffers, the throughput of data traffic seems to be more important.

Dropping the work-conserving feature of the fairness ring seems to cause the biggest improvement in throughput, predictability of the system and repeatability of the results. Small improvements for the average case throughput are shown in this demo by splitting credit and data traffic, and using more ring slots. Mainly, a better worst-case guarantee is obtained, resulting in a smaller spread in the results obtained in different runs of the benchmark.

A synthetic benchmark has been developed to offer more predictable throughput results for different ring architecture. It consists of 16 tasks connected by 15 FIFOs, each having a buffer depth of one ($\delta = 1$) to shift the limiting factor in the benchmark towards the communication on the ring network. The execution time of the tasks are minimized by letting the tasks only copy data from input FIFO to an output FIFO located on a consecutive processor tile. This creates a best-case scenario where each NI can use all ring slots and where the influence of increasing the number of slots can be seen in the measured throughput.

Table 6.9: Results of a synthetic benchmark for different configurations of the Nebula ring.

Configuration	Data rate (MS/s) $S = 2$	Data rate (MS/s) $S = 65$
Fairness and work-conserving	0.74	8.36
Split and default slots	0.56	5.24
Split and maximum number of slots	0.58	8.04

The results of the benchmark, as presented in Table 6.9, do show a large increase in throughput when multiple ring slots are reserved and large packets are transferred ($S = 65$). These results also agree upon the results of evaluation of the MCM, as shown in Figure 6.2, that increasing the number of slots has almost no influence on the throughput for small packet sizes.

There still is a limitation by the μB processors, because no difference is experienced in throughput when more than 2 slots are reserved. Increasing the buffer size to 3 tokens should hide internal (processor and communication) latencies and can reveal the limitation of the available ring bandwidth. The measured throughput of the fairness and work-conserving ring is in that case the same as for the ring where all slots are reserved (8.38 MS/s). This shows that the new ring performs as well as the previous design for high locality processing.

From these results, it can be concluded that the synthetic benchmark shows results as expected by the MCM evaluation, except that the best-case cannot be reached since execution times cannot reach their best-case of a single cycle. Furthermore, the same overall throughput can be reached in an application where the total of BE and GS bandwidth of the fairness and work-conserving ring is replaced by same amount of GS bandwidth. A more predictable ring however, offers the advantage that it is possible to reason about throughput constrains being reached before an application runs on a hard- and software based prototyping platform.

CHAPTER 7

Conclusion and Future Work

After the evaluation of the GS throughput improvement techniques for connectionless ring networks, the results of this thesis, including answers to the research questions, are discussed in Section 7.1. In the final section, Section 7.2, ideas about future work will be presented.

7.1 Conclusion

Computational power demanding application continue to use more processing cores. This trend led to the development of MPSoCs where new research problems related to multi-core designs arise. For the stream processing application (like SDRs) that are the target of this research, predicable communication between cores is a concern, where a GS throughput needs to be reached between critical tasks. Using a connectionless interconnect, a small hardware implementation can be realized, however, guaranteeing a certain throughput becomes even harder to realize. A predictable ring interconnect is developed offering a high GS throughput by utilizing the mapping of tasks onto the platform.

Before continuing by presenting the conclusion about the research questions, first, a quick look is given at the related work of this thesis, to show the conclusion in the right context. After that, the answers to the research questions (as formulated in Section 1.5) are presented.

The related work in Chapter 2 has shown a ring network is a promising architecture. It is a very small interconnect, relatively easy to implement, predicable and it can be modeled with real-time analysis models. This is in contrast to traditional mesh-based NoCs which are large and / or have an unknown worst-case behavior. Moreover, the industry is using rings in state-of the art chips like the Intel Xeon Phi [13].

When moving towards heterogeneous MPSoCs, new challenges arise on how to integrate the hardware accelerators within the NoC. Not many researchers have focused on this problem, where accelerator can be shared among multiple data producers. It justifies the research question about accelerator sharing that is part of this thesis.

SDF is chosen as the best suitable analysis model for the platform. A general SDF graph can be converted into an HSDF graph, which has strong analytical properties. Also approximation algorithms for SDF graphs exist that have a polynomial time-complexity for finding the fixed point in the global analysis optimization problems.

Such approximation algorithms do not exist for Real-Time Calculus models and Event Models which require the use of algorithms with an exponential complexity. Another property that we exploit is that a closed form expression can be derived for the throughput of SDF graphs and that this expression makes all trade-offs explicit.

The answers to the three research questions will now be presented, starting with the question about the benefits of separating different types of traffic on the ring.

1. What are the benefits of separating different types of traffic on the ring

In Chapter 4, problems and improvements were found for the inter-core communication within the research platform Starburst (Chapter 3) that have to do with the different types of communication on the ring. The FIFO communication can be separated in credits and data words traveling over the ring network. One of the problems with the current FIFO and ring implementation is that the latency for data depends on the maximum number FIFO tokens that are allocated in the local memory. There also is a greatly difference in the requirements for both types of traffic whereas the same latency and bandwidth guarantees are offered to both, leading to a requirement to over-allocation system resources. Moreover, the bandwidth for the (short distance) data traffic was not based on any information, and could be improved by making use of the mapping of the task graph of an application.

As described in Chapter 5, two techniques are implemented to control the bandwidth of data and of credit traffic, after a split is made between both types when their traffic arrives at an NI. The separation of traffic is based on the assumption that tasks are mapped on consecutive tiles on the ring, close to each other and in the same direction as the data stream is flowing. From this, it follows that data is transferred over a short distance, whereas credits are transferred over the remaining distances of almost a full ring round-trip. Moreover, data is always transferred at a higher rate as credits, at least two time the rate rate, as a single data word is accompanied with a FIFO administration word (sent towards the same destination) and only returns a single credit word. When the token size increases, the difference between the rates becomes much larger, leading to a large difference between required and offered bandwidth.

To split the data from the credit traffic, an additional ring input buffer has been added, together with a traffic shaper. The traffic shaper ensures the real-time guarantees for both traffic types are satisfied. Credits are scheduled onto the ring at a rate less than once every ring round trip, to be specific, only once every X times the ring round trip time (only when the ring slot matches the ID of the ring link). This allows data to be transmitted in the other available round-trips. It is made work-conserving, such that when no credits are present in the credit input buffer, data can be transmitted from the data buffer instead.

Fairness is still guaranteed as a ring link can only insert words onto the ring if the current ring slot matches the fixed ID of the ring link. Based on the task mapping, data can also be sent in other slot to increase their guaranteed bandwidth by the extension of each ring link with a slot mask. According to the mapping of tasks and their communication behavior (communication channels / FIFOs) slots can be used by multiple data streams as long as they don't overlap, i.e. are not passing the

same ring link. The addition of the slot mask ensures the provided service can be adjusted to match the traffic requirements.

When looking at the mapping of tasks, another factor is shown to be important, the cyclicity of a task graph (Section 4.3). In an cyclic task graph, next to being limited by cycles that are introduced by the credits that are returned to the producers of data, the system can also be limited by task graph cycle(s). These cycles do not contain credit dependent latencies; however, they cannot be mapped efficiently, as there is a fundamental limit: a cycle can never be smaller than one time the ring round-trip. The solution to this is to move to a bidirectional ring where shorter communication paths can be reached, therefore reducing the work load on the ring, and thus also leaving more bandwidth for other streams. This, however, deserves a more thorough look as part of future work (Section 7.2) as it will also increase the size of the interconnect a lot.

Another important observation is that even in acyclic task graphs, additional cycles can be introduced limiting the performance of the task communication. It is possible to compensate for this effect by increasing buffer sizes until all parallel paths between two actors contain the same number of initial tokens in its SDF model. The MCM will in that case not be limited by the additional cycles created by composing a SDF graph of the graphs of multiple producer-consumer (FIFO) pairs.

The predictability of the design is show by being able to model the improvement techniques with an SDF graph. First each technique is modeled individually, and later both are combined to model all FIFO communication over the ring. A compact HSDF model is obtained by reducing the large model to an equivalent HSDF model consisting of only 4 actor, as is proven in Appendix A. In Section 6.2 this model is used to compared the throughput of the newly created ring network to the previous version.

The comparison shows a GS throughput increase compared to the fairness protocol from a best-case of about 6 MS/s to 50 MS/s. Especially when large data packets are sent from producer to consumer and many ring slots can be utilized by the producer, a significant increase in throughput is obtained. Doubling the number of available slot will in that case lead to almost a doubling in GS throughput.

Also a case study has been performed where a PAL-demo is adjusted to make use of the newly added features of the ring. It did not show that the increase in GS throughput also increase the performance of the demo. This is mainly because the PAL-demo is limited by the processing power of the μB cores instead of the inter-core communication over the ring.

The objective to keep hardware costs at a minimum have been verified in Section 6.1. The NI buffer is doubles to enable a separation between the different traffic types, however, the ring links are shared in a predicable way. It is shown that the hardware costs of the implementation are less than when a second ring would be added specially for credit traffic.

It can be concluded that the GS throughput is significantly improvement against a low increase in hardware. Separating credit from data traffic enables ring slots to be re-used by different data producers based on the mapping of tasks on the platform, to increase the GS throughput experienced at the application level, using the software FIFOs.

2. How can hardware accelerators be shared by multiple producers in a predictable way?

Hardware accelerators are needed in a MPSoC to meet performance requirements in a power efficient way. The μBs in the Starburst platform (Chapter 3) are relatively slow processors. Moreover, CPUs are in general not very (power) efficient for DSP tasks. This in contrary to FPGA implementations where FPGA specific DSP slices [55] can be used for, for example, high speed multiply-accumulate operations (filters).

For this reason hardware accelerators were integrated into the, at that moment, homogeneous MPSoC. It is shown that a single CORDIC accelerator [56] has the same processing power as a number of μBs, for far less hardware costs [26]. The implementation however had the disadvantage that only a single producer and consumer task could predictably be 'connected' to an accelerator. The single producer task resulted in a low utilization of the hardware (a μB cannot keep up with pipelined accelerator that can accept data every clock cycle), and applications that need the same accelerator multiple times, require a duplication of that accelerator, resulting in more hardware. There are many applications in which more or less the same computation is performed multiple times in the processing chain, think for example of filters whose coefficients can be adjusted. Sharing an accelerator across multiple producing tasks is thus relevant.

An answer to the question how hardware accelerator can be shared remains future work, since not enough time was available to solve this issue. A preliminary study has however been carried out by implementing a software based sharing mechanism to identify the requirements for a hardware implementation. The analysis of the software implementation showed there is a very long path from the producers of data to the accelerator, on to the next producer(s) before coming back to the first producer and notifying it that it can use the accelerator again. A lot of buffering is required to resolve the large latency of this cycle, and worst, it is also dependent on the execution times of the other producers. When one of them exceeds its 'worst-case', the entire system becomes unpredictable. It is identified that the cycles in the data flow graph of such an implementation need to be shortened by only passing data to an accelerator when a producer has made a whole block of data that can then be processed. The hardware implementation of a hardware sharing mechanism remains future work (Section 7.2).

3. How can the power of the hardware architecture of Starburst be decreased without sacrificing the predictability of the platform?

The last research question regarding the power efficiency of the system is not answered in this thesis. It remains an interesting topic especially when looking at the current trends, to increase the processing power of mobile devices, and extending their capabilities.

There are a lot of opportunities where power efficiency can be improved and during this work only a quick look at it was performed. An example of a small power improvement is for example the reordering of ring slots. The current ring architecture does not depend on the slot order, in contrast to the previous one, which uses it for the work-conversing behavior. A sort of Gray coded schema could be used to number the ring slots which is in particular efficient when the number of ring links

is a power of 2. In that case the static power of the ring will reduce as all the slot registers will flip only one bit per link per clock cycle instead of about two flips.

More substantial improvements can be made by using techniques like power/clock-gating, since it was concluded earlier that the utilization of accelerators is low. This shows that there is room to power them down for some period. The consequences of this, however, have to be carefully evaluated in order to guarantee the real-time behavior when accelerators are power/clock-gated. How long does it for example take to power them on when new data is ready to be sent to the accelerator?

Issue have been identified to obtain a reliable estimate of the power consumption of specific parts of the MPSoC. A small experiment was carried out to show the difference when Gray coding the slot number, but it proved to be hard to get reliable power reading from the Xilinx tooling (in Xilinx XPower Analyzer). Without a predictable power estimation of the design, it is complicated to make improvement to the power efficiency of the platform. Future work should thus also include a way to reliably verify the power usage of the entire design.

7.2 Future Work

During this research a number of questions have been answered related to the communication over a connectionless ring network. Other potential interesting questions however, have also arisen that are still unanswered. A list of ideas will now be presented to cover a few possible directions of future research.

- Within this work, the focus was on a low-cost communication implementation. Therefore, the existing ring links are re-used for the two types of ring traffic and only the NI buffers are doubled. Another option would, however, be to add an additional ring. A trade-off can then be made between additional hardware costs versus an increase in GS throughput. Different configurations are possible for such a double ring design:

  - One option is to use this additional ring as a dedicated ring for credit traffic, whereas the original ring is only used for data traffic. A small increase in GS throughput and a decrease in latency can be expected, because credit traffic can no longer delay data packets and it is no longer required to reserve credit bandwidth at the expense of data bandwidth.

    Additional advantages occur when this credit ring turns in the opposite direction of the data ring. The communication distance (number of ring link) from a FIFO producer to its consumer and will then always be equal to the distance from the consumer to the producer. Both will also be small (as small a single cycle latency) when an application can be mapped efficiently on such a platform. This reduction in latency can result in a lower requirement on the depth of the FIFO buffer between two tasks.

    The complexity of the NoC will also decrease in this case as parts like the Traffic Shaper are not needed any more. This also has the advantage that it is easier to model the FIFO communication with an HSDF graph.

  - Another idea is to use the additional ring in the same way as the other one, providing services for both data and credit traffic, only differing in the direction in which the rings turn. In the best-case, this will double the available data bandwidth. It, however, depends on the application if this bandwidth will be available. Streaming applications with a acyclic task graph will not benefit much from the additional ring as the communication between tasks already takes place over short distances, mainly to consecutive tiles.

    Applications described by cyclic task graphs can benefit from a second ring. It can decrease the maximum communication distance to half of the current maximum by making a minimal distance routing decision to sent data over the normal or opposite rotating ring, whichever has the shorted path to the destination. Decreasing this maximum distance also allows a better usage of the ring slots for data traffic, because the slots can then be used by at least 2 NIs instead of 1.

    This increase in application flexibility will however come at a higher hardware costs, because more NI buffers are required than for a dedicated credit and data ring. Moreover, a solution is required to solve the problem occurring when 2 data words arrive at the same time at a processor tile, one from each ring.

- Previous work targeted the integration of hardware accelerators by implementing a flow-controlled ring. This ring is currently limited in throughput by the rate at which flow-control credits return to an NI. The principles applied for the normal ring; sending large packets of data and making use of the mapping of tasks, can also be used for the flow-controlled ring to increase the throughput of streams towards and from hardware accelerators.

- One of the application of a MPSoC like Starburst is the usage as a SDR platform. Future ultra low-power RF front ends must also be supported by a low-power processing chain. Power efficiency will therefore also become an issue for this type of applications.

  Only a minor improvement in the power consumption of the ring has been obtained by reordering the ring slots to a sort of Gray coding such that only 1 bit of the slot ID changes every clock cycle in each ring link. More substantial improvements can be made by being able to halt the ring when it is not in use, although this caused the system to become less predicable.

  The power efficiency of the processor tile and hardware accelerator can furthermore be improve by creating power islands. The voltage levels and frequencies of these islands can be set independent based on the tasks running on such an island to reduce their power consumption.

- Currently, the tasks running on the Starburst MPSoC are mapped manually onto the available processor tiles. Since manual placement is very time consuming, the automated mapping of tasks should be looked into. Tools have been developed to guide this process, as shown in Figure 1.7, but these are

not compatible with the most recent version of the Starburst platform. Future work can target mapping algorithms that are optimized to map streaming applications onto a platform consisting of processor tiles connected by a ring NoC.

- The open research question about how hardware accelerators can be share in a predicable way is still relevant for future work. Turning the Starburst MPSoC into a flexible heterogeneous platform can have major advantages in power consumption and hardware costs. Ideas about how an accelerator can be shared among different data producers are already presented in [26].

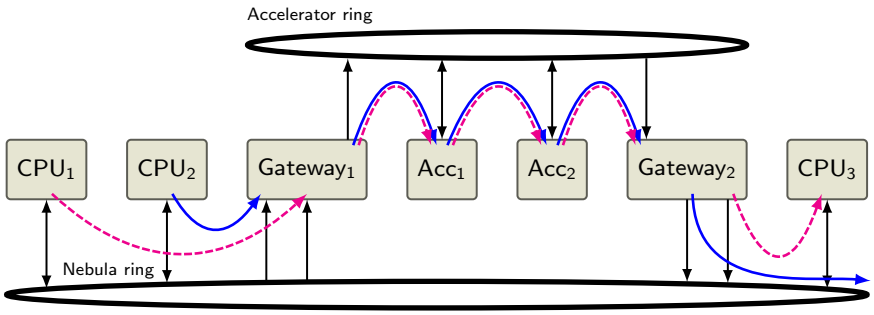

Figure 7.1: Accelerator ring idea

  - One of these ideas is to implement a "Gateway". This idea can be extended further by also looking at the communication pattern of the accelerators. As already mentioned, sharing an accelerator is most efficient when processing blocks of data instead of single data words. These blocks are stored in the memory of a input gateway which is slowly filled by processors. Once a block is complete, a burst takes place to send the data to the accelerator. Due to heavy pipelining in an accelerator, a very high throughput is reached, such that the output gateway is quickly filled with a processed block of data. The high data rates, bursts, associated with the transmission of blocks of data are not supported by the flow-controlled ring, which will therefore limit the accelerator throughput.

    A potential solution is to add a dedicated accelerator ring between the input gateway, connected accelerators and towards the output gateway, as is shown in Figure 7.1. The gateways act as bridges between the Nebula ring (meant for processor tiles) and the accelerator ring. Moreover, it is then easy to run accelerators, including their ring, at a higher clock frequency, which is not limited my the μBs. The gateways (that already contains FIFOs) can be used to cross the different clock domains.

APPENDIX A

HSDF graph reduction

In this appendix chapter, a prove will be given that the HSDF graph of a producer-consumer pair with a FIFO buffer and ring network in between can be reduced to a equivalent HSDF graph with significantly less actors. The original SDF model of the pair is already shown in Figure 4.5 together with its equivalent, but much larger, HSDF graph Figure 4.6. The equivalence of that HSDF graph and the reduced HSDF graph presented in Figure 4.7 will now be proven. The edges of both HSDF graphs are therefore labeled, as shown in Figure A.2 and Figure A.3

Figure A.1: Task graph of a producer-consumer pair with a communication channel in between.

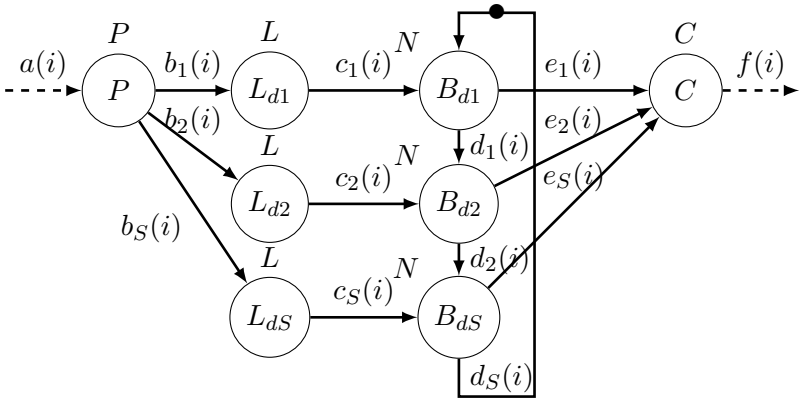

Figure A.2: HSDF model of data traffic on the ring as presented in Figure 4.6, but with labeled edges.

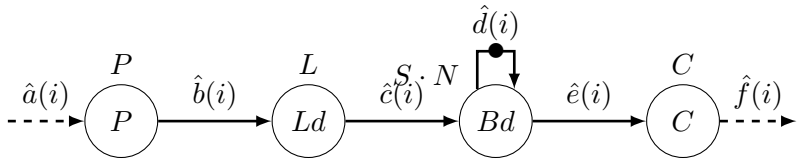

Figure A.3: Reduced HSDF graph of the one presented in Figure A.2 with labeled edges.

To prove the equivalence the following equations need to be satisfied:

$$f(i) = \hat{f}(i), \text{when } a(i) = \hat{a}(i)$$
$$\forall i \geq 0, (\forall j, \hat{a}(j) = a(j) \wedge \hat{b}(j) = b(j), ...) \Rightarrow f(i) = \hat{f}(i)$$

This can be proven as follows:

$$\begin{aligned}
&f(i) = \max\left(e_1(i), e_2(i), ..., e_S(i)\right) + \rho(C)\\
&\left.\begin{aligned} b_1(i) &= a(i) + \rho(P)\\ b_2(i) &= a(i) + \rho(P)\end{aligned}\right\} b_1(i) = b_2(i) = b_S(i)\\
&\left.\begin{aligned} c_1(i) &= b_1(i) + \rho(L)\\ c_2(i) &= b_2(i) + \rho(L)\end{aligned}\right\} c_1(i) = c_2(i) = c_S(i)\\
&d_1(i) = \max\left(c_1(i), d_S(i-1)\right) + \rho(B_{d1})\\
&d_2(i) = \max\left(c_2(i), d_1(i)\right) + \rho(B_{d2})\\
&d_S(i) = \max\left(c_S(i), d_{S-1}(i)\right) + \rho(B_{dS})\\
&e_1(i) = d_1(i)\\
&e_2(i) = d_2(i)\\
&e_S(i) = d_S(i)
\end{aligned}$$

$$\begin{aligned}
d_S(i) &= \max(c_S(i), d_{S-1}(i)) + \rho(B_{dS})\\
&= \max(c_S(i), c_{S-1}(i) + \rho(B_{d(S-1)}), d_{S-2}(i)) + \rho(B_{dS})\\
&= \max\left(\begin{array}{l} c_S(i),\\ c_{S-1}(i) + \rho(B_{d(S-1)}),\\ c_{S-2}(i) + \rho(B_{d(S-1)}) + \rho(B_{d(S-2)}),\\ ...,\\ c_1(i) + \rho(B_{dS-1}) + \rho(B_{d(S-2)}) + ... + \rho(B_{d1}),\\ d_S(i-1) + \rho(B_{dS-1}) + \rho(B_{d(S-2)}) + ... + \rho(B_{d1}) \end{array}\right) + \rho(B_{dS})
\end{aligned}$$

As $c_1(i) = c_2(i) = ... = c_S(i)$ and $\rho(B_d) \geq 0$:

$$\begin{aligned}
d_S(i) &= \max\left(\begin{array}{l} c_1(i) + \rho(B_{d(S-1)}) + \rho(B_{d(S-2)}) + ... + \rho(B_{d1}),\\ d_S(i-1) + \rho(B_{d(S-1)}) + \rho(B_{d(S-2)}) + ... + \rho(B_{d1}) \end{array}\right) + \rho(B_{dS})\\
&= \max(c_1(i), d_S(i-1)) + \rho(B_{dS}) + \rho(B_{d(S-1)}) + \rho(B_{d(S-2)}) + ... + \rho(B_{d1})
\end{aligned}$$

The firing duration of all bandwidth limiting actors has to be equal to N. Therefore, the expression can be reduces to:

$$d_S(i) = \max(c_1(i), d_S(i-1)) + S \cdot N$$

The location of the initial token on the cycle of bandwidth limiting actors already defines that $d_{S-1}(i) \leq d_S(i)$. For every new data packet produced by P, B_{d1} will be the first bandwidth limiting actor to fire, since the initial token is present on one of its input queues. Eventually, B_{dS} will fire as last actor of the cycle of actors to complete the transfer of the last part of the data packet to C. Therefore:

$$d_{S-1}(i) \leq d_S(i)$$

These calculation can also be performed for the 'potentially' equivalent graph:

$$
\begin{aligned}
\hat{b}(i) &= \hat{a}(i) + \rho(P) \\
\hat{c}(i) &= \hat{n}(i) + \rho(L) \\
\hat{d}(i) &= \max(\hat{c}(i), \hat{d}(i-1)) + \rho(B_d) \\
\hat{e}(i) &= \hat{d}(i) \\
\hat{f}(i) &= \hat{e}(i) + \rho(P)
\end{aligned}
$$

When $\hat{a}(i) = a(i)$:

$$
\begin{aligned}
\hat{b}(i) &= b_1(i) = b_2(i) = .. = b_S(i) \\
\hat{c}(i) &= c_1(i) = c_2(i) = .. = c_S(i) \\
\hat{d}(i) &= \max(d_1(i), ..d_S(i)) \\
&= d_S(i) \Rightarrow \rho(B_d) = S \cdot N \\
\hat{e}(i) &= \max(e_1(i), ..e_S(i)) \\
&= e_S(i) \\
\hat{f}(i) &= f(i)
\end{aligned}
$$

This concluded the prove that both graphs shown in Figure A.2 and Figure A.3 are equivalent for $\rho(B_d) = S \cdot N$.

APPENDIX B

VHDL Source Code

This appendix contains the source code of the most important files for the ring interconnect.

B.1 Ring link

Listing B.1: `simplering_cd.vhd`. *Ring link* VHDL source code.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity simplering_cd is
6    generic (
7      ADDR_WIDTH : positive := 32;
8      DATA_WIDTH : positive := 32;
9      MATCH_HIGH : positive := 28;
10     MATCH_LOW : positive := 20;
11     ID : natural := 0;
12     SLOTWIDTH: positive := 9;
13     RINGSIZE: positive := 32;
14     ADDR : natural := 0;
15     CREDIT_PERIOD: positive := 4
16   );
17   port (
18     clk : in std_logic;
19     rstn : in std_logic;
20
21     -- Data input from our neighbour
22     prev_slotid : in std_logic_vector(SLOTWIDTH-1 downto 0);
23     prev_addr : in std_logic_vector(ADDR_WIDTH-1 downto 0);
24     prev_data : in std_logic_vector(DATA_WIDTH-1 downto 0);
25     prev_mask : in std_logic_vector(DATA_WIDTH/8-1 downto 0);
26     prev_valid : in std_logic;
27
28     -- Output port to the local peripheral
29     addr_out : out std_logic_vector(ADDR_WIDTH-1 downto 0);
30     data_out : out std_logic_vector(DATA_WIDTH-1 downto 0);
31     mask_out : out std_logic_vector(DATA_WIDTH/8-1 downto 0);
32     valid_out : out std_logic;
33
34     -- Data input port from the local peripheral
35     data_addr_in : in std_logic_vector(ADDR_WIDTH-1 downto 0);
```

```
36      data_data_in : in std_logic_vector(DATA_WIDTH-1 downto 0);
37      data_mask_in : in std_logic_vector(DATA_WIDTH/8-1 downto 0);
38      data_valid_in : in std_logic;
39      data_accept_in : out std_logic;
40
41      -- Credit input port from the local peripheral
42      credit_addr_in : in std_logic_vector(ADDR_WIDTH-1 downto 0);
43      credit_data_in : in std_logic_vector(DATA_WIDTH-1 downto 0);
44      credit_mask_in : in std_logic_vector(DATA_WIDTH/8-1 downto 0);
45      credit_valid_in : in std_logic;
46      credit_accept_in : out std_logic;
47
48      -- Data input to our other neighbour
49      next_slotid : out std_logic_vector(SLOTWIDTH-1 downto 0);
50      next_addr : out std_logic_vector(ADDR_WIDTH-1 downto 0);
51      next_data : out std_logic_vector(DATA_WIDTH-1 downto 0);
52      next_mask : out std_logic_vector(DATA_WIDTH/8-1 downto 0);
53      next_valid : out std_logic
54    );
55  end simplering_cd;
56
57  architecture rtl of simplering_cd is
58    -- The data is a configuration value for the ring link
59    signal is_link_configuration : std_logic;
60    constant LINK_CONF_BIT : natural := 18;
61    -- The current address is local
62    signal is_local : std_logic;
63    -- The current slot id is whatever we just received from the "
          prev_slotid" port
64    signal current_slotid : std_logic_vector(SLOTWIDTH-1 downto 0);
65    -- High if the current slot is available
66    signal is_available : std_logic;
67    -- Integer version of the current slot id, used for comparison
68    signal current_slotid_i : natural;
69    -- High is slot can be used to send a credit
70    signal is_credit_slot : std_logic;
71    -- High if there is a credit slot and a valid credit input
72    signal is_valid_credit_slot : std_logic;
73    -- High is slot can be used to send data, allowed by the slotmask
74    signal is_data_slot : std_logic;
75    -- High if a credit should be inserted onto the ring, data otherwise.
76    signal select_credit : std_logic;
77    -- Counter of is_credit_slot, only insert credits once every credit
          period.
78    signal credit_count_i : natural range 0 to (CREDIT_PERIOD-1);
79    -- A mask where is stored which slots may be used for data
80    signal slot_mask : std_logic_vector(RINGSIZE-1 downto 0);
81    -- Address of the selected input
82    signal addr_in : std_logic_vector(ADDR_WIDTH-1 downto 0);
83    -- Data of the selected input
84    signal data_in : std_logic_vector(DATA_WIDTH-1 downto 0);
85    -- Mask of the selected input
86    signal mask_in : std_logic_vector(DATA_WIDTH/8-1 downto 0);
87    -- Valid of the selected input; data or credit valid_in
88    signal valid_in : std_logic;
89  begin
90    process(clk, rstn)
91    begin
92      if rising_edge(clk) then
93        if rstn/='1' then
94          current_slotid <= std_logic_vector(to_unsigned(ID, SLOTWIDTH));
```

```
95          -- Initialize the slot mask to only use the own slot for data
96          slot_mask <= (ID => '1', others => '0');
97        else
98          current_slotid <= prev_slotid;
99          -- Increment the credit counter
100         if is_credit_slot='1' then
101           credit_count_i <= (credit_count_i + 1) mod (CREDIT_COUNT + 1)
                ;
102         end if;
103         -- Set the slot-mask
104         if is_local='1' and is_link_configuration='1' then
105           -- Max of 32 ringlinks
106           for i in slot_mask'range loop
107             slot_mask(i) <= prev_data(i);
108           end loop;
109         end if;
110       end if;
111     end if;
112   end process;
113   is_link_configuration <= prev_addr(LINK_CONF_BIT);
114   next_slotid <= current_slotid;
115   current_slotid_i <= to_integer(unsigned(current_slotid));
116   -- Determine if the slot can be used and what its type is
117   -- When the slot ID matches the link ID the slot is a credit slot
118   is_credit_slot <= '1' when current_slotid_i = ID else '0';
119   -- When the selected bit in the mask is set the slot is a data slot
120   is_data_slot <= slot_mask(current_slotid_i);
121   -- Check if the slot is available for the selected traffic type
122   is_available <= is_credit_slot when (select_credit = '1') else
        is_data_slot;
123
124   -- Determine if the credit input is valid and a credit slot is
        available
125   -- Work-conversing: when there is no credit input, use the slot for
        data
126   is_valid_credit_slot <= is_credit_slot and credit_valid_in;
127
128   -- The Traffic Shaper will select the credit input once every
129   -- credit period
130   select_credit <= is_valid_credit_slot when credit_count_i = (
        CREDIT_PERIOD-1) else '0';
131   -- The average credit latency can be reduced by implementing a
132   -- Sporadic Server, which use a replendishment interval instead of a
        period
133
134   addr_in <= credit_addr_in when (select_credit = '1') else
        data_addr_in;
135   data_in <= credit_data_in when (select_credit = '1') else
        data_data_in;
136   mask_in <= credit_mask_in when (select_credit = '1') else
        data_mask_in;
137   valid_in <= credit_valid_in when (select_credit = '1') else
        data_valid_in;
138
139   process(clk)
140   begin
141     if rising_edge(clk) then
142       if rstn/='1' then
143         next_addr <= (others=>'-');
144         next_data <= (others=>'-');
145         next_mask <= (others=>'-');
```

```
146         next_valid <= '0';
147       -- Forward the data if this is not our slot and its in use (and
              not meant for us)
148       elsif prev_valid='1' and is_local='0' then
149         next_addr <= prev_addr;
150         next_data <= prev_data;
151         next_mask <= prev_mask;
152         next_valid <= prev_valid;
153       -- Push local data onto the ring if its available and we have
              data
154       elsif valid_in='1' and is_available='1' then
155         next_addr <= addr_in;
156         next_data <= data_in;
157         next_mask <= mask_in;
158         next_valid <= valid_in;
159       else
160         next_addr <= (others=>'0');
161         next_data <= (others=>'0');
162         next_mask <= (others=>'1');
163         next_valid <= '0';
164       end if;
165     end if;
166   end process;
167
168   is_local <=
169 --pragma translate_off
170     '0' when is_x(prev_addr) else
171 --pragma translate_on
172     prev_valid when prev_addr(MATCH_HIGH downto MATCH_LOW) =
          std_logic_vector(to_unsigned(ADDR,MATCH_HIGH-MATCH_LOW+1)) else
173     '0';
174
175   addr_out <= prev_addr;
176   data_out <= prev_data;
177   mask_out <= prev_mask;
178     valid_out <= is_local and not is_link_configuration;
179
180   -- Accept data when the previous slot is valid (not used by a
181   -- credit of another link) and the data input is selected
182   data_accept_in <= (not prev_valid or is_local) and is_data_slot and
        not select_credit;
183
184   -- Credits should always be accepted when the own credit slot is
185   -- available, because that slot is not shared.
186   credit_accept_in <= is_credit_slot and select_credit;
187 end rtl;
```

B.2 NI buffer

Listing B.2: `plb2ring.vhd`. *NI buffer* VHDL source code.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  library ip_srlfifo_lib;
6  use ip_srlfifo_lib.srlfifo;
7
8  entity plb2ring_cd is
9    generic
10   (
11     C_SPLB_BASEADDR : std_logic_vector := x"FFFFFFFF";
12     C_SPLB_HIGHADDR : std_logic_vector := x"00000000";
13     C_SPLB_MID_WIDTH : natural := 1;
14     C_SPLB_NUM_MASTERS : natural := 1;
15     C_AWIDTH : integer := 32;
16     C_DWIDTH : integer := 32;
17     C_CFIFO_DEPTH : natural := 1; Credit FIFO depth
18     C_FIFO_DEPTH : natural := 1   Data FIFO depth
19   );
20   port (
21   ...
22   --PLB signals: plb_* are inputs, and sl_* are outputs to the bus
23   ...
24
25   data_addr_out : out std_logic_vector(C_AWIDTH-1 downto 0);
26   data_data_out : out std_logic_vector(C_DWIDTH-1 downto 0);
27   data_mask_out : out std_logic_vector(C_DWIDTH/8-1 downto 0);
28   data_valid_out : out std_logic;
29   data_accept_out : in std_logic;
30
31   credit_addr_out : out std_logic_vector(C_AWIDTH-1 downto 0);
32   credit_data_out : out std_logic_vector(C_DWIDTH-1 downto 0);
33   credit_mask_out : out std_logic_vector(C_DWIDTH/8-1 downto 0);
34   credit_valid_out : out std_logic;
35   credit_accept_out : in std_logic
36   );
37 end plb2ring_cd;
38
39 architecture rtl of plb2ring_cd is
40   function swapix(v:std_logic_vector) return std_logic_vector is
41     variable s : std_logic_vector(v'reverse_range);
42   begin
43     for i in 0 to v'length-1 loop s(v'high-i) := v(v'low+i); end loop;
44     return s;
45   end swapix;
46
47   constant CREDIT_BIT : natural := 19;
48   -- compensate for reversed bit in plb
49   alias is_credit is plb_abus(C_AWIDTH-1 - CREDIT_BIT);
50   type state_t is (REQUEST,DATA,RERR,WERR);
51   type reg_t is record
52     state : state_t;
53     mid : natural range 0 to 2**plb_masterid'length-1;
54     weight_counter : std_logic_vector(C_DWIDTH-1 downto 0);
55   end record;
56   signal r_in, r : reg_t;
57
```

```
58    signal rstn : std_logic;
59    signal clk : std_logic;
60    signal plb_pavalid_i : std_logic;
61    signal fifo_in : std_logic_vector(C_DWIDTH/8+C_AWIDTH+C_DWIDTH-1
          downto 0);
62    signal data_fifo_in, data_fifo_out : std_logic_vector(C_DWIDTH/8+
          C_AWIDTH+C_DWIDTH-1 downto 0);
63    signal credit_fifo_in, credit_fifo_out : std_logic_vector(C_DWIDTH/8+
          C_AWIDTH+C_DWIDTH-1 downto 0);
64    signal data_rd_accept, data_valid_out_i : std_logic;
65    signal credit_rd_accept, credit_valid_out_i : std_logic;
66    signal fifo_wr_valid, fifo_wr_accept : std_logic;
67    signal data_fifo_wr_valid, data_fifo_wr_accept : std_logic;
68    signal credit_fifo_wr_valid, credit_fifo_wr_accept : std_logic;
69  begin
70
71    rstn <= not splb_rst;
72    clk <= splb_clk;
73    fifo_in <= swapix(not plb_be)&swapix(plb_abus)&swapix(plb_wrdbus);
74
75    data_mask_out <= data_fifo_out(C_DWIDTH/8+C_AWIDTH+C_DWIDTH-1 downto
          C_AWIDTH+C_DWIDTH);
76    data_addr_out <= data_fifo_out(C_AWIDTH+C_DWIDTH-1 downto C_DWIDTH);
77    data_data_out <= data_fifo_out(C_DWIDTH-1 downto 0);
78
79    credit_mask_out <= credit_fifo_out(C_DWIDTH/8+C_AWIDTH+C_DWIDTH-1
          downto C_AWIDTH+C_DWIDTH);
80    credit_addr_out <= credit_fifo_out(C_AWIDTH+C_DWIDTH-1 downto
          C_DWIDTH);
81    credit_data_out <= credit_fifo_out(C_DWIDTH-1 downto 0);
82
83    data_fifo_i : entity ip_srlfifo_lib.srlfifo
84      GENERIC MAP (
85        WIDTH => C_DWIDTH/8+C_DWIDTH+C_AWIDTH,
86        DEPTH => C_FIFO_DEPTH)
87      PORT MAP (
88        clk => clk,
89        rstn => rstn,
90        wr_data => data_fifo_in,
91        wr_valid => data_fifo_wr_valid,
92        rd_accept => data_rd_accept,
93        rd_data => data_fifo_out,
94        wr_accept => data_fifo_wr_accept,
95        rd_valid => data_valid_out_i
96      );
97
98    credit_fifo_i : entity ip_srlfifo_lib.srlfifo
99      GENERIC MAP (
100       WIDTH => C_DWIDTH/8+C_DWIDTH+C_AWIDTH,
101       DEPTH => C_CFIFO_DEPTH)
102     PORT MAP (
103       clk => clk,
104       rstn => rstn,
105       wr_data => credit_fifo_in,
106       wr_valid => credit_fifo_wr_valid,
107       rd_accept => credit_rd_accept,
108       rd_data => credit_fifo_out,
109       wr_accept => credit_fifo_wr_accept,
110       rd_valid => credit_valid_out_i
111     );
112
```

```
113   data_fifo_wr_valid <= fifo_wr_valid when (is_credit = '0') else '0';
114   credit_fifo_wr_valid <= fifo_wr_valid when (is_credit = '1') else
          '0';
115   fifo_wr_accept <= data_fifo_wr_accept when (is_credit = '0') else
116       credit_fifo_wr_accept when (is_credit = '1') else
117       '0';
118   data_fifo_in <= fifo_in;
119   credit_fifo_in <= fifo_in;
120
121   data_rd_accept <= data_valid_out_i and data_accept_out;
122   credit_rd_accept <= credit_valid_out_i and credit_accept_out;
123
124   data_valid_out <= data_valid_out_i;
125   credit_valid_out <= credit_valid_out_i;
126
127   ...
128   --Processes controling the PLB-signals
129   ...
130
131 end rtl;
```

Bibliography

[1] 4DSP. FMC125. Online, `http://www.4dsp.com/FMC125.php`, Date accessed: 2013-07-15.

[2] Ankur Agarwal, Boca Raton, and Cyril Iskander. Survey of Network on Chip (NoC) Architectures & Contributions. *Journal of Engineering, Computing and Architecture*, 3(1), 2009.

[3] Angela Haskell. To Space & Back latest Planetarium feature. Online, `http://www.phillytrib.com/westsouthwestmetroarticles/item/8256-%E2%80%98to-space-back%E2%80%99-latest-planetarium-feature.html`, Date accessed: 2013-06-24.

[4] LA Barroso and Michel Dubois. Performance evaluation of the slotted ring multiprocessor. *Computers, IEEE Transactions on*, 44(7), 1995.

[5] T Bijlsma, M J G Bekooij, G J M Smit, and P G Jansen. Omphale: Streamlining the Communication for Jobs in a Multi Processor System on Chip. Technical Report TR-CTIT-07-44, Centre for Telematics and Information Technology University of Twente, Enschede, July 2007.

[6] T Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, {DTU}, Richard Petersens Plads, Building 321, {DK-}2800 Kgs. Lyngby, 2005.

[7] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of Network-on-chip. *ACM Computing Surveys*, 38(1):1–es, June 2006.

[8] Geoffrey Blake, RG Dreslinski, and Trevor Mudge. A survey of multicore processors. *IEEE Processing Magazine*, (November):26–37, 2009.

[9] Simone Borgio, Davide Bosisio, Fabrizio Ferrandi, Matteo Monchiero, Marco D Santambrogio, and Politecnico Milano. Hardware DWT accelerator for Multi-Processor System-on-Chip on FPGA. pages 107–114, 2006.

[10] S. Bourduas and Z. Zilic. A comparison of two multistage ring architectures for NoC using high-level simulation models. *2008 1st Microsystems and Nanoelectronics Research Conference*, pages 37–40, October 2008.

[11] BD Bui, R Pellizzoni, and M Caccamo. Real-time scheduling of concurrent transactions in multidomain ring buses. *. . . , IEEE Transactions on*, 61(9):1311–1324, 2012.

[12] GC Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. 2011.

[13] Chrysos, George. Intel Xeon Phi Coprocessor - the Architecture. September 2012. Online, `http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner`. Date accessed: 2013-06-11.

[14] Jason Cong, MA Ghodrat, and Michael Gill. AXR-CMP: Architecture support in accelerator-rich CMPs. *... on SoC Architecture, ...*, 2011.

[15] Epiq Solutions. Bitshark FMC-1RX. Online, `http://epiqsolutions.com/fmc-1rx.php`, Date accessed: 2013-07-15.

[16] Chris Fallin, X Yu, and Kevin Chang. HiRD: A Low-Complexity, Energy-Efficient Hierarchical Ring Interconnect. 004:1–20, 2012.

[17] Chris Fallin, Xiangyao Yu, Gregory Nazario, and Onur Mutlu. A High-Performance Hierarchical Ring On-Chip Interconnect with. Technical report, Computer Architecture Lab, Carnegie Mellon University, 2011.

[18] Marc Geilen, Stavros Tripakis, and Maarten Wiggers. The earlier the better: a theory of timed actor interfaces. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, HSCC '11, pages 23–32, New York, NY, USA, 2011. ACM.

[19] Kees Goossens and Andreas Hansson. The aethereal network on chip after ten years. In *Proceedings of the 47th Design Automation Conference on - DAC '10*, volume 4, page 306, New York, New York, USA, 2010. ACM Press.

[20] Cristian Grecu, Andrè Ivanov, and R Pande. Towards open network-on-chip benchmarks. *Networks-on-Chip, ...*, 2007.

[21] L Gwennap. Sandy Bridge spans generations. *Microprocessor Report*, (September 2010), 2010.

[22] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip. *VLSI Design*, 2007:1–10, 2007.

[23] Joost P. H. M. Hausmans, Maarten H. Wiggers, Stefan J. Geuns, and Marco J. G. Bekooij. Dataflow analysis for multiprocessor systems with non-starvation-free schedulers. *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems - M-SCOPES '13*, page 13, 2013.

[24] Yifan He, Dongrui She, Sander Stuijk, and Henk Corporaal. Efficient Communication Support in Predictable Heterogeneous MPSoC Designs for Streaming Applications. *Journal of Systems Architecture*, May 2013.

[25] Rafik Henia, Arne Hamann, Marek Jersak, and Razvan Racu. System level performance analysis–the SymTA/S approach. *... -Computers and Digital ...*, 2005.

[26] G. J. Hoekstra. Hardware Accelerator Integration in a Connectionless Network-on-Chip. Technical report, University of Twente, 2013.

[27] ITRS. International technology roadmap for semiconductors System drivers, 2011.

[28] G Kahn. The semantics of a simple language for parallel programming. *Information Processing*, 74, 1974.

[29] John Kim and Hanjoon Kim. Router microarchitecture and scalability of ring topology in on-chip networks. In *Proceedings of the 2nd International Workshop on Network on Chip Architectures - NoCArc '09*, page 5, New York, New York, USA, December 2009. ACM Press.

[30] M. Kistler, M. Perrone, and F. Petrini. Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro*, 26(3):10–23, May 2006.

[31] Kanishka Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, number ii, pages 29–35. IEEE Comput. Soc, 2001.

[32] M. Millberg, E. Nilsson, R. Thid, and a. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pages 890–895, 2004.

[33] Akiko Narita, Kenji Ichijo, and Yoshio Yoshioka. Comprehensive Evaluation of Packet Flow Control Methods for a Ring Nework of Processors on Chip. *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, pages 75–80, August 2010.

[34] Netherlands Streaming (NEST) Consortium. Research Summaries. Online, `http://www.nest-consortium.nl/projectSummary.php`. Date accessed: 2012-11-26.

[35] A Nieuwland, J Kang, O P Gangwal, R Sethuraman, N Busa, K Goossens, R P Llopis, and P Lippens. C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems. *Design Automation for Embedded Systems*, 7(3):233–270, 2002.

[36] Sankaralingam Panneerselvam and MM Swift. Operating systems should manage accelerators. *... of the 4th USENIX conference on Hot ...*, pages 1–7, 2012.

[37] E Rijpkema, K Goossens, and A Rădulescu. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *... - Computers and Digital ...*, 150(5), 2003.

[38] JH Rutgers. An efficient asymmetric distributed lock for embedded multiprocessor systems. *Embedded Computer ...*, 2012.

[39] JH Rutgers, MJG Bekooij, and GJM Smit. Evaluation of a Connectionless NoC for a Real-Time Distributed Shared Memory Many-Core System. pages 13–16, 2012.

[40] JH Rutgers, MJG Bekooij, and GJM Smit. Portable memory consistency for software managed distributed memory in many-xore SoC. 2013.

[41] Pradip Kumar Sahu and Santanu Chattopadhyay. A survey on application mapping strategies for Network-on-Chip design. *Journal of Systems Architecture*, 59(1):60–76, January 2013.

[42] Erno Salminen, Ari Kulmala, and Timo D. Hämäläinen. Survey of network-on-chip proposals. White paper. Online, `www.ocpip.org/socket/whitepapers`, March, 2008.

[43] Erno Salminen, K Srinivasan, and Z Lu. OCP-IP Network-on-chip benchmarking workgroup. *OCP-IP,[online]*, page 5, 2010.

[44] Larry Seiler, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, Pat Hanrahan, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, and Jeremy Sugerman. Larrabee. *ACM Transactions on Graphics*, 27(3):1, August 2008.

[45] Sundararajan Sriram and Shuvra S. Bhattacharyya. Embedded Multiprocessors: Scheduling and Synchronization. February 2009.

[46] Hevc Standard, Gary J Sullivan, Jens-rainer Ohm, Woo-jin Han, Thomas Wiegand, Abstract High, Efficiency Video, and Coding Hevc. Overview of the High Efficiency Video Coding. 22(12):1649–1668, 2012.

[47] Sriram R. Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, Vasantha Erraguntla, Clark Roberts, Yatin Hoskote, Nitin Borkar, and Shekhar Borkar. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, January 2008.

[48] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *Proceedingsof the 10th international workshop on Software & compilers for embedded systems - SCOPES '07*, page 11, New York, New York, USA, April 2007. ACM Press.

[49] Maarten H. Wiggers, Marco J.G. Bekooij, and Gerard J.M. Smit. Monotonicity and run-time scheduling. *Proceedings of the seventh ACM international conference on Embedded software - EMSOFT '09*, page 177, 2009.

[50] Wikipedia. Starburst region. Online, `http://en.wikipedia.org/wiki/Starburst_region`. Date accessed: 2013-06-12.

[51] Daniel Wiklund and Dake Liu. SoCBUS: switched network on chip for hard real time embedded systems. *Parallel and Distributed Processing ...*, 00(C), 2003.

[52] Wayne Wolf, A.A. Jerraya, and Grant Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, October 2008.

[53] P.T. Wolkotte. *Exploration within the Network-on-Chip Paradigm*. PhD thesis, University of Twente, 2009.

[54] Xilinx. MicroBlaze Processor Reference Guide (UG081, v13.4). 081.

[55] Xilinx. DSP48E1 Slice (UG369 v1.3). 369:1–52, 2011.

[56] Xilinx. LogiCORE IP CORDIC (DS249 v4.0). pages 1–30, 2011.

[57] Xilinx. ML605 Hardware User Guide (UG534 v1.6). 534, 2011.

[58] Xilinx. Virtex-6 FPGA Memory (UG363 v1.6). 363:1–84, 2011.

[59] Xilinx. Configurable Logic Block (UG364 v1.2). pages 1–50, 2012.