Connecting Æthereal to the Montium

Master's Thesis
by

T.M. Jongsma
s0066230

University of Twente, Enschede, The Netherlands
Computer Architecture for Embedded Systems
Faculty of EEMCS
October 27, 2010

Abstract

English

A Communication and Configuration Unit (CCU) is developed to make it possible to connect a Montium Tile Processor (TP) to an Æthereal Network-on-Chip (NoC). The CCU is the interface between the Montium TP and the NoC. A system with MicroBlaze processors connected to Æthereal with a Device Transaction Level (DTL) interface is already available. For better performance for Digital Signal Processing (DSP) the system will be extended with Montium TPs. The Montium TP is a coarse-grained reconfigurable processor. In Æthereal can be chosen from 2 types of Network Interfaces: bus or streaming. The only bus protocol used within this project is DTL. The implemented CCU has two interfaces to the NoC: a streaming interface for the data processing and a Memory-Mapped Input-Output (MMIO) interface for configuration and Direct Memory Access (DMA), which can be streaming or DTL. The choice for streaming or DTL is done at design-time, because it is implemented as an optional adapter which converts DTL to streaming. To be able to test the implemented CCU on an Field Programmable Gate Array (FPGA) evaluation board, a system consisting of 2 MicroBlaze Cores and 2 Montium TPs connected to Æthereal is generated. A small application is successfully executed on a Xilinx ML605 evaluation board, which contains a Virtex-6 FPGA. In this setup the Montium can run on 14.82 MHz. To be able to make a comparison with other CCUs, the design of the CCU, DTL adapter and Montium TP is also synthesized for an Application Specific Integrated Circuit (ASIC). The size of the CCU is 0.01478 mm^2 without DTL adapter. The DTL adapter is 0.00149 mm^2. These results were obtained using a 90 nm low power library and a clock frequency contraint of 400 MHz.

Nederlands

Een Communication and Configuration Unit (CCU) is ontwikkeld om het mogelijk te maken een Montium Tile Processor (TP) aan een Æthereal Network-on-Chip (NoC) aan te sluiten. Een CCU is de interface tussen de Montium TP en een NoC. Een systeem bestaande uit MicroBlaze processoren verbonden met Æthereal door middel van een Device Transaction Level (DTL) interface is reeds beschikbaar. Voor betere prestaties bij het uitvoeren van digitale signaalverwerkingsalgoritmen wordt dit systeem uitgebreid met Montium TPs. De Montium TP is een grofkorrelig herconfigureerbare processor.

Binnen Æthereal kan uit 2 soorten Netwerk Interfaces gekozen worden: bus of streaming. Het enige bus protocol dat gebruikt is binnen dit project is DTL.

De CCU die ontwikkeld is in dit project heeft 2 interfaces naar het NoC: een streaming interface voor de data verwerking en een Memory-Mapped Input-Output (MMIO) interface voor configuratie en Direct Memory Access (DMA), welke door middel van streaming of door middel van DTL verbonden kan worden aan het NoC. De keuze voor streaming of DTL moet gedaan worden tijdens het systeemontwerp, omdat het geïmplementeerd is als een optionele adapter, welke DTL converteert naar streaming. Om de ontworpen CCU te kunnen testen op een Field Programmable Gate Array (FPGA) evaluatie bord, is een systeem met 2 MicroBlaze processors en 2 Montium TPs verbonden met Æthereal gegenereerd. Een klein programma is succesvol uitgevoerd op een Xilinx ML605 evaluatie bord, waarop een Virtex-6 FPGA zit. In deze configuratie kan voor de Montium een klok frequentie van 14.82 MHz gebruikt worden. Om een vergelijking met andere CCU's te kunnen maken, is het ontwerp van de CCU, DTL adapter en Montium TP ook gesynthetiseerd voor een Application Specific Integrated Circuit (ASIC). De grootte van de CCU is 0.01478 mm^2 zonder DTL adapter. De DTL adapter heeft een grootte van 0.00149 mm^2. Deze resultaten werden verkregen met gebruikmaking van een 90 nm laag vermogen bibliotheek en een beperking van de klok frequentie op 400 MHz.

Preface

This thesis gives an overview of the design and implementation of a CCU which makes it possible to connect the Montium TP to Æthereal NoC.

This report, the VHDL code I wrote, and the intermediate and final presentations are part of my master assignment of the Electrical Engineering Embedded Systems track I followed at the University of Twente. This assignment was carried out in the scope of the NEST project.

For 10 months I have been working on this CCU. I started with research about the subject and related work. Next, I tried to understand the Montium and became familiar with Æthereal.

Almost every fortnight on Tuesday morning, I had a meeting with (a part of) my committee to point out the features to be implemented, to monitor the progress and to discuss the problems I encountered.

These were valuable moments, because it kept up my discipline, gave me new ideas and made me work even harder on my assignment in the days before the meeting.

Of course, I would like to thank everyone who contributed in some way to the final result. Besides the members of the committee, I would like to thank Marcel van de Burgwal for providing tooling and information about the Montium, as well as his assistance during debugging my CCU, which I greatly appreciate.

Contents

List of Acronyms

ADC Analog to Digital Converter

AGU Address Generation Unit

ALU Arithmetic and Logic Unit

ASIC Application Specific Integrated Circuit

BE Best Effort

BRAM Block Random Access Memory

CCM Central Configuration Manager

CCU Communication and Configuration Unit

DAC Digital to Analog Converter

DMA Direct Memory Access

DSP Digital Signal Processing

DTL Device Transaction Level

FFT Fast Fourier Transform

FIFO First In First Out

FIR Finite Impulse Response

FPGA Field Programmable Gate Array

GPI General Purpose Input

GPO General Purpose Output

GPP General Purpose Processor

GS Guaranteed Service

IP Intellectual Property

JTAG Joint Test Action Group

LUT Lookup Table

MAC Multiply-Accumulate

MMIO Memory-Mapped Input-Output

MP-SoC Multiple Processor System-on-Chip

MSB Most Significant Bit

NI Network Interface

NoC Network-on-Chip

PLB Processor Local Bus

PPA Processing Part Array

RISC Reduced Instruction Set Computing

ROM Read-Only Memory

RTOS Real-Time Operating System

SIO Streaming Input-Output

SoC System-on-Chip

Tcl Tool command language

TP Tile Processor

UART Universal Asynchronous Receiver-Transmitter

VHDL Very High Speed Integrated Circuit Hardware Description Language

XMD Xilinx MicroBlaze Debugger

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Multi-core trend

For years, new generation CPUs which came to market, had their performance gain mainly due to higher clock frequencies. When this became more difficult, other ways to increase the performance were used. One of those ways to maintain delivering increasing performance, a trend to include more cores into a single die started. Today's mainstream computers are equipped with dual- and quadcore CPUs.

This multi-core trend is also visible in other computer architecture markets where energy efficiency is of more importance, for instance in the mobile phone market [9]. General Purpose Processors (GPPs) are very flexible and can perform many different tasks. Due to this flexibility, the power consumption when a computation is performed on a GPP, is often higher than the same computation on an Application Specific Integrated Circuit (ASIC) or Digital Signal Processor specialized for those computations. There is a trade-off between performance and flexibility. A way to keep or extend processing power, using less energy, can be achieved by adding different cores, each with its own specialism, in a single system. When algorithms are mapped in a clever way on the right cores, the same processing can be performed with decreased energy consumption [15].

"Many-core architectures" is an active research subject. A toolchain to generate a Multiple Processor System-on-Chip (MP-SoC) with an arbitrary number of MicroBlazes was available, this toolchain is called 'Starburst'. It can generate an Æthereal Network-on-Chip (NoC) (see Section 1.4) with an arbitrary number of MicroBlaze Soft-Core Processors. The MicroBlaze Soft-Core processor is a processor from Xilinx based on a 32-bits RISC architecture. Also a DDR memory controller and peripherals like LEDs and UARTs are accessible via the NoC.

A MicroBlaze takes multiple clock cycles for a Multiply-Accumulate (MAC) operation. In many Digital Signal Processing (DSP) algorithms the MAC operation is often used. Therefore the MicroBlaze is not well suited to do energy efficient streaming DSP. Specialized DSP cores can perform a MAC operation in a single clock cycle, consuming less energy than the MicroBlaze for the same computation. Streaming is processing of data sample by sample, in contrast to block-based processing, which processes blocks of samples. A useful addition to the Starburst System-on-Chip (SoC) Generator is another processing core which is more suited to do energy efficient streaming signal processing than a MicroBlaze processor.

1.2 Montium Tile Processor

In 2004, a coarse-grained reconfigurable processor, called Montium TP, was developed by Paul Heysters. The Montium is specialized in DSP operations like Finite Impulse Response (FIR)-filtering and Fast Fourier Transforms (FFTs). In most DSP algorithms the MAC operation is frequently used. The Montium can do 5 MAC operations within one clock cycle, which makes the Montium powerful in DSP applications.

Another property of the Montium is that on beforehand is known how long processing steps take and on every clock cycle it is known which instruction is executed on the Montium. The Montium processing structure is straightforward and the Montium is not disturbed by for instance interrupts where GPPs may suffer from. The Æthereal NoC is also capable of giving bandwidth and latency guarantees. This combination of Æthereal and the Montium makes it possible to give latency guarantees, which are required in some applications.

The properties of the Montium mentioned before make the Montium a useful addition to a many-core system currently only consisting of MicroBlazes.

The structure of the Montium is shown in Figure 1.1. The Montium has 10 global busses, which are mainly used for internal communication in the Montium Tile Processor (TP), for example to transfer data between Arithmetic and Logic Units (ALUs). On the right side can be seen that the 10 global busses of the Montium are directly connected to the Communication and Configuration Unit (CCU). The Montium processor has 5 ALUs. Every ALU has two local memories, a left local memory and a right local memory. Those memories are numbered M01...M10 in Figure 1.1. The size of those local memories is parameterizable, because memories are area-hungry and it depends on the application which sizes of local memories are necessary. In the configuration used during this project, the local memories have a depth of 1024 words and a data width of 16 bits. Due to the locality of reference principle, the local memories contribute to the energy-efficiency of the Montium [4]. Every ALU has 4 input register banks, often referred to as register A, B, C or D. A more detailed schematic drawing of an ALU is shown in Figure 1.2. The ALU is split up in 2 levels: level1 and level2. Level1 is for reconfigurable bitwise functions, (saturated) additions, (saturated) subtractions, logic shift left (only function unit 1 and 2) or logic shift right (only function unit 1 and 2) and determine maximum or minimum of two values (only function unit 3 and 4). Level2 is for the MAC operation [7].

1.2.1 Montium interface

The interface of the Montium is not compatible with the Æthereal Network Interfaces (NIs). To make it possible to connect the Montium to a NoC, a CCU is necessary. The CCU takes care of the communication with the NoC: it routes the output of the Montium busses to the right output connection and routes the input to the right Montium bus. The Montium can be paused by the CCU. The

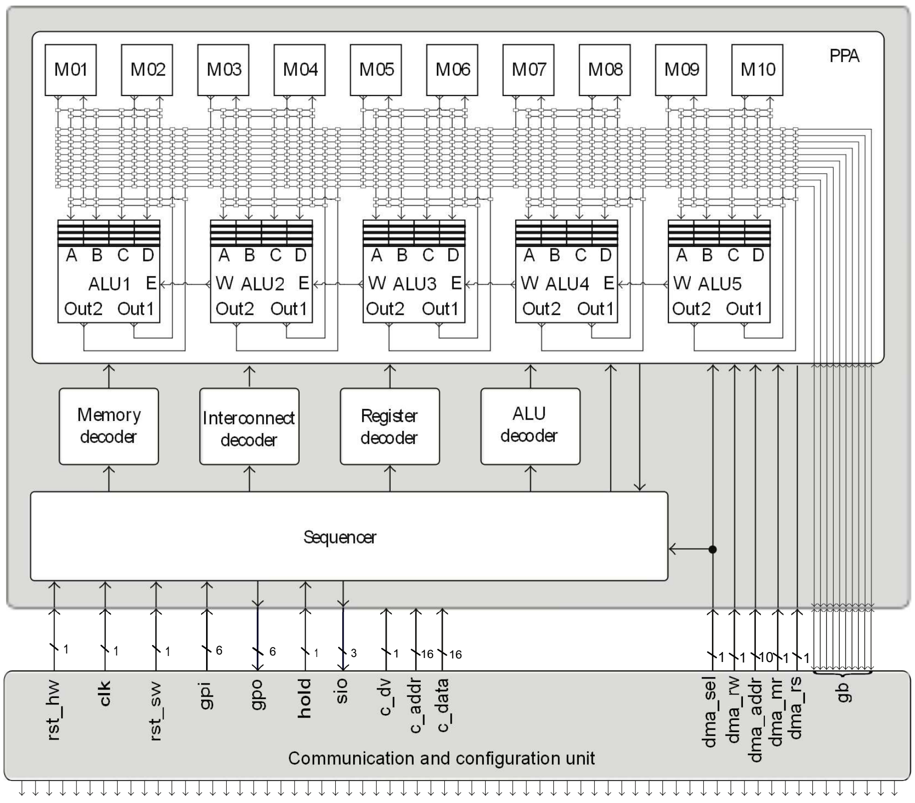

Figure 1.1: Montium structure and interface to CCU

interface of the Montium is shown in Figure 1.1. The interface as shown in the figure is the interface as used in this project. The number of streaming IO pins and the number of synchronization pins (called General Purpose Input (GPI) and General Purpose Output (GPO)) are parameterizable. On the left side the sequencer interface is visible, which controls the program execution. The CLK and RST_HW are connected to the clock network and the system wide reset. Near the Streaming Input-Output (SIO) lines, the configuration interface is shown. When a data and address pair is available on C_ADDR and C_DATA the C_DV line is driven high to clock in the configuration data (shown in Figure 2.2). Using the Direct Memory Access (DMA) interface, data from local memories or register files can be read by a GPP in the NoC. During a DMA transfer, the Montium is paused.

1.3 Beamforming demonstrator

A possible application of the multi-core SoC consisting of MicroBlaze cores and Montium TPs is beamforming. Beamforming is a technique which can make a receiver more sensitive for signals from a certain direction, using multiple

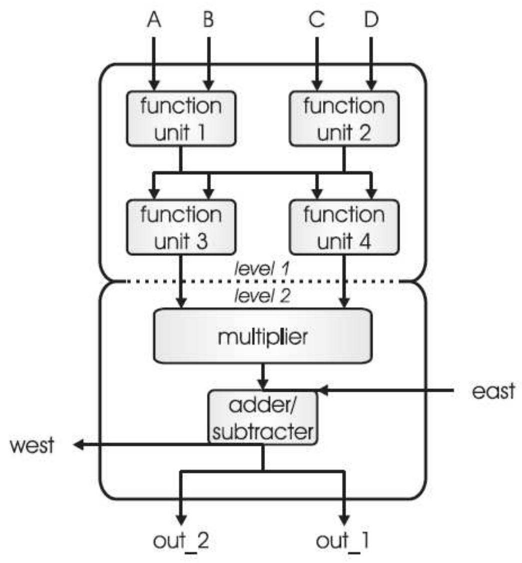

Figure 1.2: Montium ALU structure

antennas. This technique has its origin in radar applications, where it is known as phased array. After the signal from the antenna is digitized, digital signal processing techniques can be used. Using digital signal processing, the signal can be combined such that the array is more sensitive in a certain direction. Due to the multiple antennas involved with beamforming, a lot of DSP is needed. Also the power consumption of this processing is important, because devices, which can be usefully extended with beamforming features, are often mobile devices using wireless communication, like mobile phones, notebooks, netbooks and bluetooth peripherals. Nowadays those devices receive from all directions and transmit to all directions. Making those devices directional, the same Signal-to-Noise Ratio can be achieved, using less power. In these mobile devices, power consumption is an important design aspect, because it influences the battery rundown time.

1.4 Æthereal NoC

The NoC used in the Starburst SoC Generator is Æthereal. Æthereal is a composable and predictable on-chip interconnect developed at NXP [3]. In a composable platform, one application cannot change the behaviour of another application. This allows design and verification of applications in isolation. Æthereal can be used in a real-time environment, because it is able to guarantee minimum throughput and maximum latency [3]. Æthereal offers two types of connections:

- Guaranteed Service (GS) - guaranteed throughput and bounded latency
- Best Effort (BE) - to exploit NoC capacity unused by GS traffic for non-critical communication

Æthereal uses streaming interfaces for its communication. For other protocols, i.e. protocols which need data and address ports, a shell can be used. The protocol shells bridge between a bus protocol and the streaming ports of the network [3]. In Figure 1.3 a schematic drawing of a target protocol shell as used in Æthereal is displayed. On the left of the figure the signals as used in the protocol and on the right side a streaming interface connected to the network is drawn.

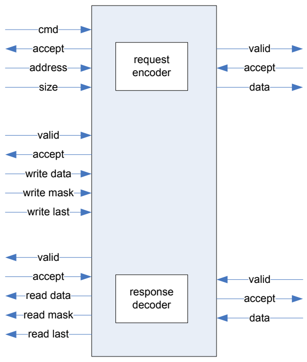

Figure 1.3: Target protocol shell as used in Æthereal

A drawback of these different interfaces is that it is not possible in Æthereal to send data from a streaming NI to a Device Transaction Level (DTL) NI or vice versa.

1.5 Assignment description

The assignment for this Master's thesis is the design and implementation of a CCU which is able to connect a Montium TP to the Æthereal NoC as used in the Starburst SoC Generator. Within this research it is necessary to specify requirements of the CCU, programming the CCU and testbenches in VHDL and connect the CCU to Æthereal and to the Montium TP. After functional simulation of the whole system, the extended Starburst system has to be successfully tested on the Xilinx ML605 evaluation board.

1.6 Related work

In [11], a network interface called Hydra is described. It is mentioned that there is not much related work on network interfaces, because it is often presented as a minor addition to a NoC and network interfaces are assumed to be straightforward. It is also stated that the design decisions in the NoC interface are important for the performance of the overall system. A CCU accommodates the communication between a Montium TP and the NoC. A circuit-switched NoC [13] and a packet-switched NoC [5] are used. The CCU is synthesized in 0.13 μm with a constraint on the clock frequency of 200 MHz. This resulted in about 19000 gates (0.106 mm^2), which is about 5% of the area of the Montium TP [8]. A large part (41.5%) of the total area is needed for input buffering and output buffering. The crossbar connecting the 10 Montium busses to 4 network lanes only uses 9.5% of the CCU area. The NoC uses flits for communication. The flow control and flit formatting are responsible for 20% of the total area.

1.7 Document structure

In this chapter, an introduction to the subject has been given. It also treated the scope of the project. In chapter 2, the requirements of the CCU are explained. Chapter 3 describes the implementation of the CCU. Chapter 4 gives Field Programmable Gate Array (FPGA) and ASIC synthesis results and the data rates that can be achieved when using the CCU. In chapter 5, a communication application is mapped onto the two Montium cores to show a working CCU. In the last chapter the conclusions and recommendations are presented. Three appendices are added to this report: a design specification, a memory map and the source code for a small application which was executed on the FPGA board.

Chapter 2

Requirements

In this chapter the requirements of the CCU are specified. It is divided into three sections: the tasks of the CCU, the interface description to the Montium and the interface description to the NoC NIs. As already mentioned in the introduction, on the NoC side, the type of interface (Memory-Mapped Input-Output (MMIO) or streaming) and the number of interfaces are configurable. On the Montium interface side, the number of streaming IO and the number of GPI and GPO pins are parameterizable.

2.1 View at system level

The Montium TP has to be configured, before any processing can be done by the Montium TP. This means the Montium TP is dependent on the configuration data from the NoC. After startup of a system containing a Montium TP, the first task of the CCU is routing configuration data from the NoC to the Montium. After the Montium is configured, the Montium can select the communication scheme using the SIO lines. With these SIO lines, network lanes are connected to global busses of the Montium.

2.1.1 Tasks of the CCU

The CCU has the following tasks [10]:

- load data to be processed from the NoC
- store (partly) processed data to the NoC
- pause the Montium core (necessary when DMA operations are done, input data is unavailable or saving energy when no work is available)
- restart the Montium TP from pause
- reset the Montium TP
- configure the Montium TP

The network is not configured by the CCU. In Æthereal, there is one processor which has a connection to the configuration port of the NoC. This processor opens and closes connections between cores connected to the NoC.

A difference between the CCU described in reference [10] and this CCU is the location of the clock domain crossing. The Æthereal NoC takes care of the correct exchange of data between different clock domains, in contrast to the

CCU described in reference [10] where the clock domain crossing is inside the CCU by means of dual port asynchronous FIFOs.

2.1.2 Communication with other cores

As mentioned in section 1.4, Æthereal supports a streaming protocol as well as bus protocols. In the Starburst SoC generator, the MicroBlazes are only connected by a DTL interface. Therefore the Montium TP cores have to be configured by DTL. For communication with other Montium TP cores a streaming interface can better be used, because a streaming interface has less overhead than DTL and is faster.

2.2 Area

The area usage of a chip is an important design parameter as there is a strong relation between the area usage of a chip and its price [2] and its power consumption. Therefore it is important to keep the area of the CCU as small as possible. An estimate of an area requirement can be made by taking the Hydra CCU as a reference. The Hydra uses 0.106 mm^2 in 0.13 μm technlogy, which is about 5% of the area of the Montium TP [11]. As in the CCU connecting Æthereal to the Montium, some memory inside the CCU is unnecessary (see section 2.6.1), a requirement is that the CCU must use less than 5% of the area of the Montium TP.

2.3 Clock frequency

The clock frequency of the CCU has to be the same as the Montium, because the clock domain crossing is handled by Æthereal. It is important that the CCU will not be the limiting factor for the clock frequency of the Montium TP. In other words: the longest combinatorial path has to be inside the Montium TP and not in the CCU.

2.3.1 Latency

The latency of the streaming interface is more important than the latency of the MMIO interface, because in applications where latency is important, the streaming interface is the interface to use, because it is better suited to meet low-latency requirements. The influence of the CCU on the latency is in the order of nanoseconds. In typical applications in which the Montium is used, like image processing or beamforming, timing deadlines are in the order of milliseconds. This makes it unnecessary to optimize for latency in the CCU. The Montium TP is able to process a sample every clock cycle. To avoid decreasing the performance of the Montium TP, the CCU must be able to deliver a sample every cycle, which is more important than latency.

The latency of the MMIO interface is less important than the streaming interface, because this interface can be extended with a DTL adapter, which is not the best interface choice when latency is an issue, due to the transaction overhead, which is time-consuming. Besides latency of this interface is not a real issue, the number of cycles to process input data is also not of much importance, because during normal operation (most of the time) the streaming interface of the Montium is used. The MMIO interface is only used during (re)configuration or DMA transfers, which is only the case for a small fraction of time. This interface has no strict requirements for latency or number of clock cycles to process input. This gives the opportunity to optimize this interface for another design parameter as for instance area.

2.4 Verification

It is important that the design performs as expected. Formal verification is considered to be outside the scope of this project, to limit the size of the project. A correct working implementation is important, therefore correct behaviour of the implementation is acquired by functional simulations with coverage. As the coverage will not be 100% this only gives a suspicion of being correct, but is no proof of correct behaviour in all circumstances.

2.5 Debugging

No special debug interface is implemented in the design of the CCU. The GPI and GPO pins can be used for debugging if necessary. They can be hooked up to for example test LEDs to signal certain events. Because the GPI and GPO interface is between the CCU and Montium, the state of the Montium as well as the CCU can be debugged by these interfaces. Also Xilinx MicroBlaze Debugger (XMD) is a useful tool to debug the system. Via XMD commands can be given to the MicroBlaze. This way the CCU can be debugged via the NoC and the MicroBlaze. XMD is considered in Section 3.3.2.

2.6 Montium

Some of the requirements for the CCU have their origin in the Montium design. Those requirements are treated in the next sections.

2.6.1 Memory map

The Montium memory map is divided into zones. The first two bits of the configuration address point out the memory zone.

The first three zones are of minor importance for the design of the CCU (zone 00, 01 and 10). The only task for the CCU for those first three memory zones is routing the configuration data and address for those zones to the configuration interface of the Montium. For zone 11, this is different. The exact location of the data must be known by the CCU designer, because the memories of zone 11 are memories inside the CCU.

The memory inside the CCU is called SIO decoder memory, because the Montium selects with 3 SIO lines 1 of the 8 communication schemes. The SIO decoder memory is filled with data from the compiler such that the Montium can select an input and output connection between the network data lanes and the Montium busses. More details about the meaning of those configuration memories can be found in Chapter 3 and Appendix A, where the implementation details are discussed.

2.6.2 Montium interface

In chapter 5.5 of reference [4], a CCU for the Montium TP is treated. In that chapter the tasks of the CCU as well as the description and meaning of the interfaces are discussed.

After the SoC is powered on, a Central Configuration Manager (CCM) (this can be a GPP inside the SoC) sends a configuration binary of an application to the CCU. This configuration binary is obtained by compiling an application written in MontiumLLL. The CCU uses this binary to configure the Montium and itself. After the configuration is loaded, the CCU receives input data from the NoC. For the input data two choices can be made: block-mode or streaming mode. In block-mode the CCU uses DMA to load data in the local memories of the Montium TP. After the data is stored, the CCU can signal the sequencer of the Montium to start computing the block of data by using the GPI pins. When latency is important, the streaming mode is more attractive, because samples are directly processed. In block-mode, first a whole block of data has to be received. After all data in the block is processed, the data is available at the output. On average a sample is later available at the output when block-mode transfers are compared to streaming-mode transfers. The lanes connected to the streaming NIs of the NoC are directly connected to the busses of the Montium. The handshake signals for the NoC are handled by the CCU. In case of a full or empty buffer, the Montium TP has to be suspended by the CCU until the communication congestion is solved [4]. In [4] the CCU is separated in three parts:

- Sequencer interface
- Configuration interface
- DMA interface

The purpose of those interfaces is treated in the subsequent sections.

Sequencer interface

The sequencer interface is used for general control of the Montium. It is used to synchronize the state of the processor with the state of the CCU. For example by providing a high on the HOLD line the CCU can pause the Montium TP, for example in case the required data is not available yet or no room on the NI is available at the output.The GPI and GPO signals of the sequencer can be used for synchronisation between the CCU and the Montium TP. In Figure 2.1 a waveform of the sequencer is shown. It shows the CCU signaling the Montium TP to start the application. The address counter of the sequencer inside the Montium starts to change. When the CCU holds the Montium TP the address of the sequencer is not changed.

Configuration interface

The configuration interface is used to configure the Montium to execute a specific task. While a new configuration is being loaded, it is recommended to disable the sequencer. When the sequencer is not paused there is a risk the sequencer is reading data which is being changed by the configuration interface at the same time. This can lead to unexpected behaviour. The Central

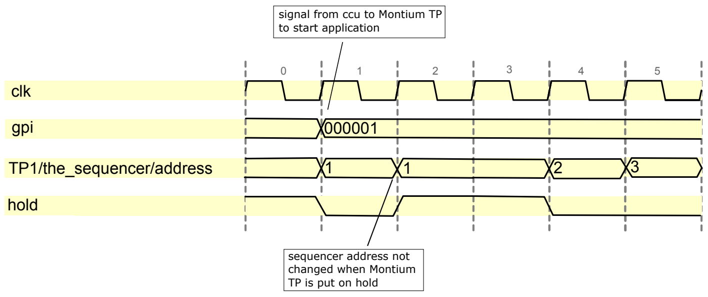
signal from ccu to Montium TP to start application
0
1
2
3
4
5
clk
gpi
000001
TP1/the_sequencer/address
1
1
2
3
hold
sequencer address not changed when Montium TP is put on hold

Figure 2.1: Waveform of the sequencer

Configuration Manager (CCM) is responsible for the configuration of the Montium. The CCM is part of a small Real-Time Operating System (RTOS) that runs on a GPP tile. The configuration interface has three signals: C_ADDR, C_DATA and C_DV. When the correct data and address pair is on the lines C_DATA and C_ADDR, the C_DV line is made high to clock the data in the Montium configuration memory. A waveform showing this behaviour is displayed in Figure 2.2.

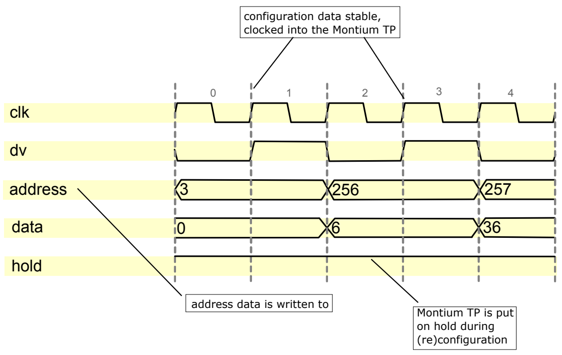
configuration data stable, clocked into the Montium TP
0
1
2
3
4
clk
dv
address
3
256
257
data
0
6
36
hold
address data is written to
Montium TP is put on hold during (re)configuration

Figure 2.2: Waveform of the configuration interface

DMA interface

The DMA interface is used to access data RAM in the Montium. The Montium is paused while the CCU does a DMA transfer. A DMA transfer consists of two phases: DMA initialization and the DMA transfer. A DMA transfer can be performed by selecting a memory or register with the DMA_MR and DMA_RS signals. After the DMA_ADDR is driven with the right address and the data is on the bus, DMA_SEL must be driven with a logical high level to clock the data in the given address and selected register or memory.

2.6.3 NoC interface

Æthereal uses a streaming interface for communication. A streaming interface has three signals: DATA, VALID and ACCEPT. In Æthereal, a streaming port can be extended by a shell which makes a translation between the streaming protocol and a memory-mapped protocol as for instance DTL.

For the data processing, the streaming port is definitely the best choice, because data can be sent in a regular pattern. When DTL would be used, a burst of data can be sent, but this requires that the amount of data has to be known at the start of the transfer, or every byte can be sent separately, which results in a lot of overhead. Therefore DTL is not suitable for this type of data transfer. As switching of signals uses power, the DTL interface is less energy efficient than the streaming interface. Also the throughput of a DTL interface is lower than the throughput of a streaming interface. For the configuration data it is less clear which type is the best choice: Æthereal only provides communication options between the same type of port: a DTL port cannot communicate with a streaming port and vice versa, which makes the choice of the type of interface important, because it determines the possible communication partners.

To keep all options open, the design of this CCU requires a streaming port for communication with the Montium busses during streaming processing. For the configuration, sequencer control and DMA transfers, a CCU with a streaming configuration port or a DTL configuration port is required.

Bandwidth

The bandwidth of the streaming interface and the bandwidth of the DTL interface are both important. The bandwidth of the streaming interface is important, because it is the interface used during normal operation. When the bandwidth of this interface is not sufficient, the processing power of the Montium cannot be fully exploited, because the Montium is stalled when the interface is not ready to accept or deliver data.

The bandwidth of the DTL interface is important, because it determines the communication speed between the Montium TP and the MicroBlazes.

The streaming interface has to be able to transfer a word every clock cycle. Because the Montium uses words of 2 Bytes, 2 Bytes are expected to be transferred per clock cycle per streaming interface.

2.7 List of requirements

From the requirements discussed in the previous sections, a summary of the requirements in the form of a numbered list is given.

1. Able to transfer data via Æthereal streaming interface
2. Able to transfer data via DTL interface for
    a) Sequencer control
    b) Configuration data
    c) DMA transfer
3. No buffering inside CCU on streaming interface for low-latency
4. Capable of transferring data every clock cycle on streaming interface
5. CCU area smaller than 5% of Montium TP area
6. Clock frequency of CCU same as Montium TP
7. Critical path not inside the CCU
8. Compatible with MontiumLLL compiler for SIO memory locations and number of lanes

Chapter 3

Structural design

In this chapter the design choices made during the design of the CCU are explained. The CCU has two types of interfaces, which are very different. Therefore the chapter is split up in two parts:

- a part about the MMIO port for the sequencer, configuration interface and DMA transfers in Section 3.1
- a part about the implementation of the streaming interface in Section 3.2

An overview of the architecture is shown in Figure 3.1.

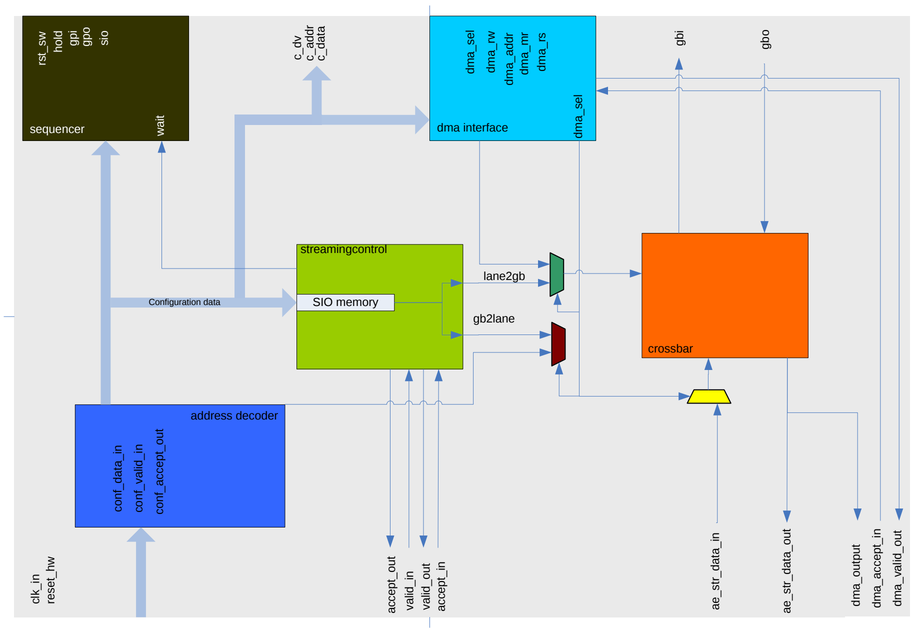

Figure 3.1: Internal structure of the CCU

The signals from the address decoder to the sequencer and streamingcontrol are used for configuration. The DMA interface has a connection to the address decoder for configuration and also for DMA transfers. During normal operation the streamingcontrol controls the crossbar with the LANE2GB and GB2LANE signals. When a DMA transfer is performed, the crossbar is controlled by other signals for correct DMA routing. On the lower left corner the configuration input for the MMIO interface can be seen. On the lower right corner, the streaming signals are connected to a streaming interface of the NoC. In the upper right corner the connections of the global busses between the Montium TP and the CCU are drawn.

3.1 MMIO interface

The MMIO interface is for configuration and control of the Montium TP . Also DMA transfers can be performed using the MMIO interface. The configuration data for the CCU as well as for the Montium TP arrives at this interface. There are three types of data arriving at this interface:

1. Configuration data from the compiler
   a) Configuration data for the Montium TP (memory zone 00, 01 or 10)
   b) Configuration data for the CCU (memory zone 11)
2. Data written to CCU registers which are connected to Montium TP interfaces (generated by the CCU user in addition to data generated by the compiler), to control the Montium TP or to perform DMA transfers

3.1.1 Connection to NoC

The CCU has a streaming interface for its MMIO interface. To comply with requirement 2, an optional DTL adapter is designed for the MMIO interface. This dual-interface requirement comes from a constraint of the NoC (as mentioned in Section 1.4) which only supports communication between interfaces of the same type. To use this DTL adapter or not can be chosen at design-time. In the next section, the implementation of the streaming interface at the MMIO interface is treated. Subsequently the implementation of the DTL adapter is discussed in the next section.

Streaming

The streaming interface signals for input are as described in Table 3.1. There is a 32 bits wide DATA port, because Æthereal uses 32-bits words. When connected to the CCU, the 16 most-significant bits of the received 32 bits word are used as address for the Montium TP configuration registers and CCU registers. The 16 least-significant bits of a received 32 bits word are used as data for the accompanying address. When the DATA signal is stable, the VALID line is driven high. When the destination is able to accept the data, the ACCEPT is made high. After receiving an ACCEPT, the source node is allowed to change the data. In Figure 3.2 the signals of a streaming interface are shown as a waveform.

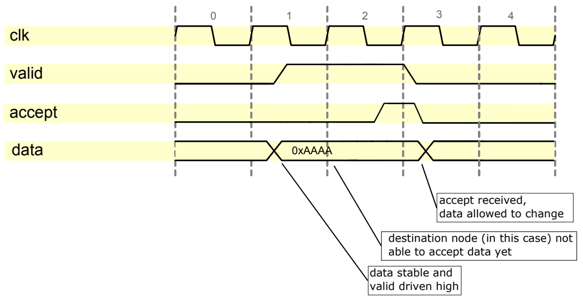
0
1
2
3
4
clk
valid
accept
data
0xAAAA
accept received, data allowed to change
destination node (in this case) not able to accept data yet
data stable and valid driven high

Figure 3.2: Waveform of the streaming interface

Signal	Width	Direction
Input		
Data	32	IN
Valid	1	IN
Accept	1	OUT
Output		
Data	32	OUT
Valid	1	OUT
Accept	1	IN

Table 3.1: Streaming signals of the MMIO interface without DTL adapter

DTL

A DTL interface can be an initiator or a target. The initiator is the only interface which is able to start a transaction. The target can only react on a started transaction by the initiator. A DTL initiator and a DTL target with their interconnections are shown in Figure 3.4. In the figure, the data width is displayed between brackets. When no value between brackets is named, the signal has a width of 1.

A data transfer always starts with a command transfer. In the command transfer the initiator transmits to the target if the initiator would like to write or read data and to or from which address the data has to be written or read. Also the amount of data and a mask can be sent to the target. After the command phase, there can be a write phase or a read phase.

When a write phase follows after the command phase the initiator sends words to the target. As DTL is designed for MMIO targets, the target can put the data on the address received in the command transfer before. When the WR_DATA is stable, WR_VALID is made high by the initiator. The target makes WR_ACCEPT high to signal the data is received and can be changed by the initiator. When the last word is transmitted from the initiator to the target not only WR_VALID, but also WR_LAST is made high to signal the target the last word is transmitted.

If a read phase is announced in the command phase, the target sends data to the initiator. It makes RD_VALID high when correct data is placed at the RD_DATA port and waits for a high RD_ACCEPT. When the last word is transmitted also RD_LAST is made high at the same time as RD_VALID.

The state machine for the DTL interface is shown in Figure 3.3. The state machine consists of 2 separate branches: 1 for performing a DTL write on the left side and 1 for performing a DTL read on the right side. The state machine starts in the NOP state. When it receives a logic high value on the DTL_CMD_VALID line, it starts depending on the value on the DTL_CMD_READ line, a write or read transaction. Data depending on the address given in the command transfer is sent to the CCU to set the right signals at the interface to the Montium TP.

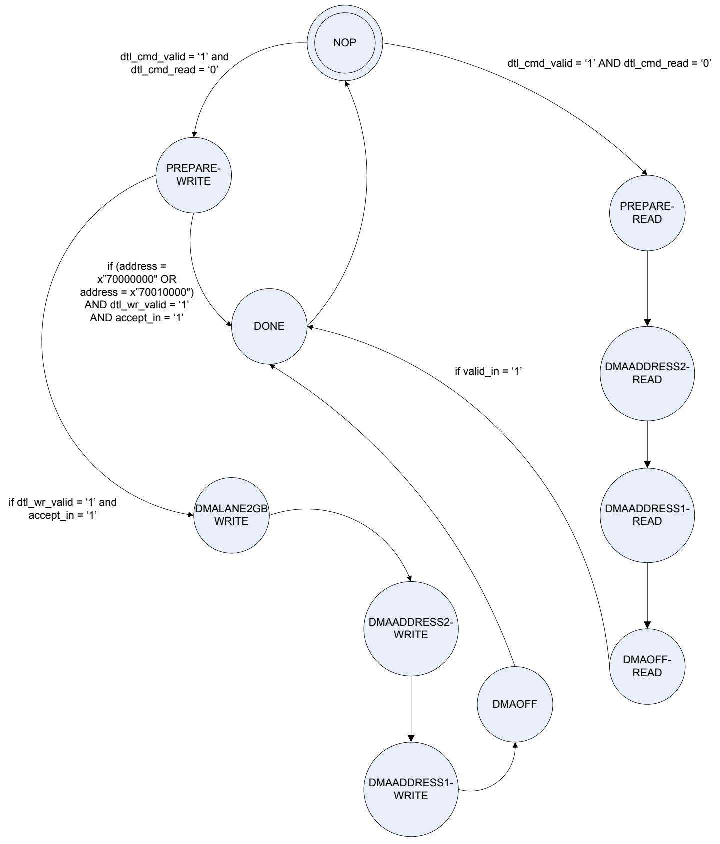
NOP
dtl_cmd_valid = '1' and dtl_cmd_read = '0'
dtl_cmd_valid = '1' AND dtl_cmd_read = '0'
PREPARE-WRITE
PREPARE-READ
if (address = x"70000000" OR address = x"70010000") AND dtl_wr_valid = '1' AND accept_in = '1'
DONE
if valid_in = '1'
DMAADDRESS2-READ
DMAADDRESS1-READ
if dtl_wr_valid = '1' and accept_in = '1'
DMALANE2GB WRITE
DMAADDRESS2-WRITE
DMAOFF-READ
DMAOFF
DMAADDRESS1-WRITE

Figure 3.3: State machine implemented in the DTL adapter

3.1.2 MMIO registers

The registers to control the sequencer interface, configuration interface and DMA interface as well as the streaming memory inside the CCU are accessable via this interface. For the detailed description of the registers, see Appendix A.

The way a DMA transfer is performed depends on whether the DTL adapter is connected or not.

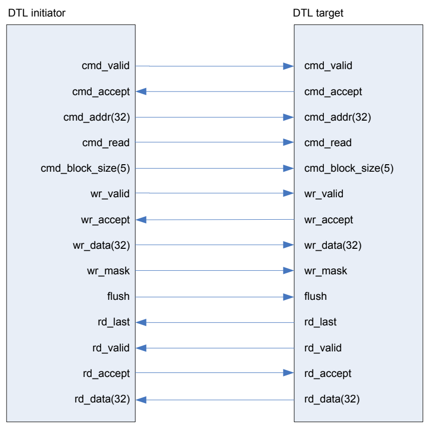

Figure 3.4: DTL initiator and target interface signals

DMA transfer without connected DTL adapter

To do a DMA transfer, some registers have to be set with the right values to select the register and address. To be sure the correct data is written or read, all registers have to be written with the correct value. After all registers have the correct value, the actual DMA access is performed by writing to DMA address 1 with DMA_SEL high, followed by a write to DMA address 1 with DMA_SEL low.

DMA transfer with connected DTL adapter

Using the DTL adapter a DMA transfer can be performed easier than the DMA transfer described in the previous section, because advantage can be taken of the DTL protocol. The DTL adapter takes care of setting the right registers in accordance with the address used. The memory map is shown in Appendix B. In Figure 3.6, a DTL write transfer followed by a DTL read transfer is shown. The transaction starts with a command write transfer in cycle -1. After the command is received, the DTL adapter sends data to the right registers to control the DMA interface of the Montium TP. The current state (see state diagram in Figure 3.3) of the DTL adapter is mentioned in line ‘state’. When all registers are in the right position, DMA_SEL is driven high, for the actual DMA transfer. During this DMA transfer, the LANE2GB signal is controlled by the DMA interface, instead of the streaming control interface (made possible by a multiplexer as shown in Figure 3.1) to route the data to be written to the right global bus of the Montium TP.

After the write transfer, a read transfer is performed. The command transfer of the read transfer is shown in cycle 6. After the command transfer is received, the DTL adapter hops through the states DMAADDRESS2READ, DMAADDRESS1READ, DMAOFFREAD to put the right signals on the DMA interface to the Montium TP. For reading, the DMA_SEL line is high for 2 clock cycles, in contrast to the write cycle where it is only high for one clock cycle. The reason for this difference is a delay of 1 cycle before the requested data is retrieved from the local memories of the Montium TP.

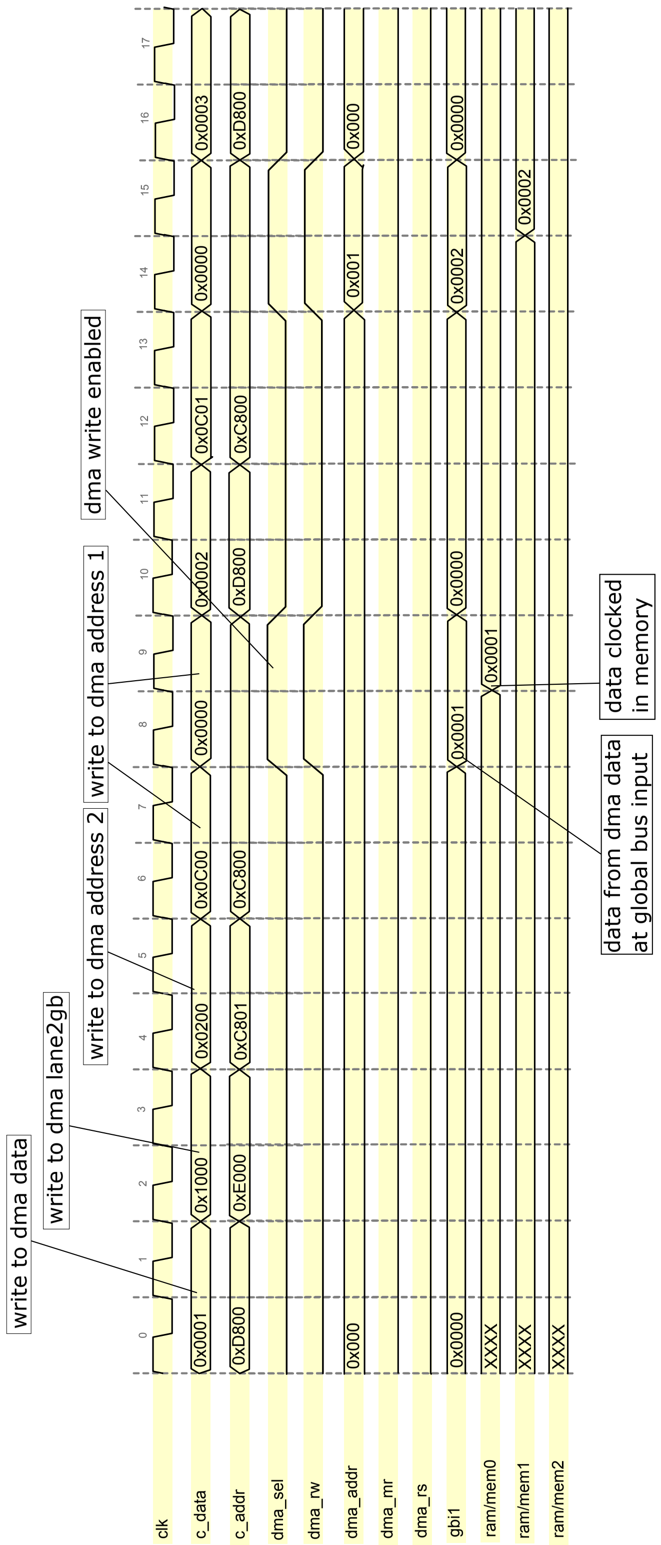

Figure 3.5: DMA write without using DTL adapter

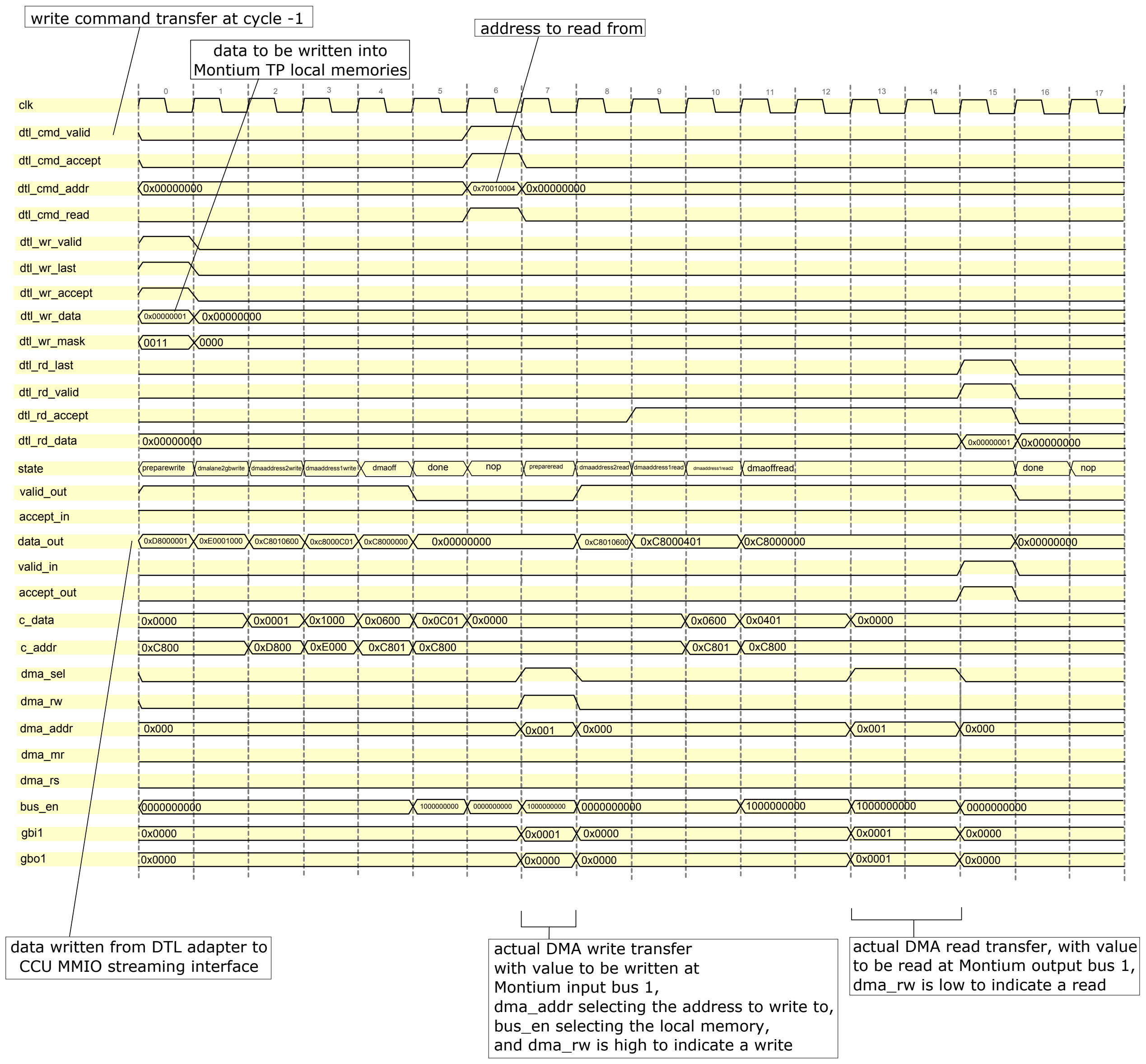

Figure 3.6: DTL write and read waveform

3.2 Streaming interface

In Section 3.2.1 the connection to the NoC is presented. In Section 3.2.2, the way the Montium TP controls the connections is treated. During the connection control of the Montium TP, the CCU pauses the Montium TP on time when a communication problem occurs. Section 3.2.3 treats the results and compares it to the requirements as formulated in Section 2.7.

3.2.1 Connection to NoC

The streaming interface of the CCU has 4 output streaming lanes and 4 input streaming lanes. Every streaming interface has 3 signals as shown in Table 3.1. As the NoC is 32-bits and the Montium TP uses 16-bits words only the least significant bits of the 32-bits word are used.

In the system as implemented and discussed in Chapter 4 (see Figure 4.1), the streaming interface is not connected to Æthereal. The streaming connection between the two Montium TPs are directly connected to each other for simplicity. This way Æthereal is bypassed. This direct connection allowed a long combinatorial path between ALUs of both Montiums. This long combinatorial path is split up by adding FIFOs with a depth of 8 between the streaming lanes of both CCUs.

3.2.2 Implementation details

In this section the actual implementation of the streaming part of the CCU is treated. It starts with the working principle of the Montium TP dictating the communication scheme using the SIO lines (see Figure 1.1). In the subsequent section the mechanism which holds the Montium TP when no input data is available or when there is no room on the NoC to output data is described in detail.

SIO working principle

There is a large crossbar block in the CCU responsible for the connection between the Montium TP busses and the NoC lanes. A schematic diagram of this crossbar is displayed in Figure 3.7.

On the left side the inputs from the streamingcontrol entity (see Figure 3.1) are drawn. The LANE2GB signals control the input for the Montium TP and the GB2LANE signals control the output from the Montium TP to the NoC. The LANE2GB signal is 16 bits wide (4 lanes × 4 bits per multiplexer control signal) and the GB2LANE signal is 20 bits wide (4 lanes × 5 bits per multiplexer control signal). The GB2LANE multiplexer control signal is 1 bit wider, because in the Hydra CCU the most significant GB2LANE bit is used to select the predefined messages from the Command Read-Only Memory (ROM). The MontiumLLL compiler generates 5 bits for every GB2LANE signal. To remain compatible with the MontiumLLL compiler (requirement 8) 5 bits are implemented for the GB2LANE signal. The 16 bits wide multiplexer control signal for LANE2GB and the 20 bits wide multiplexer control signal for GB2LANE, extended with 2 bits to define the flit type from the Montium TP busses to the output lanes [12], makes 38 bits wide registers for the SIO memories. Because the Montium TP

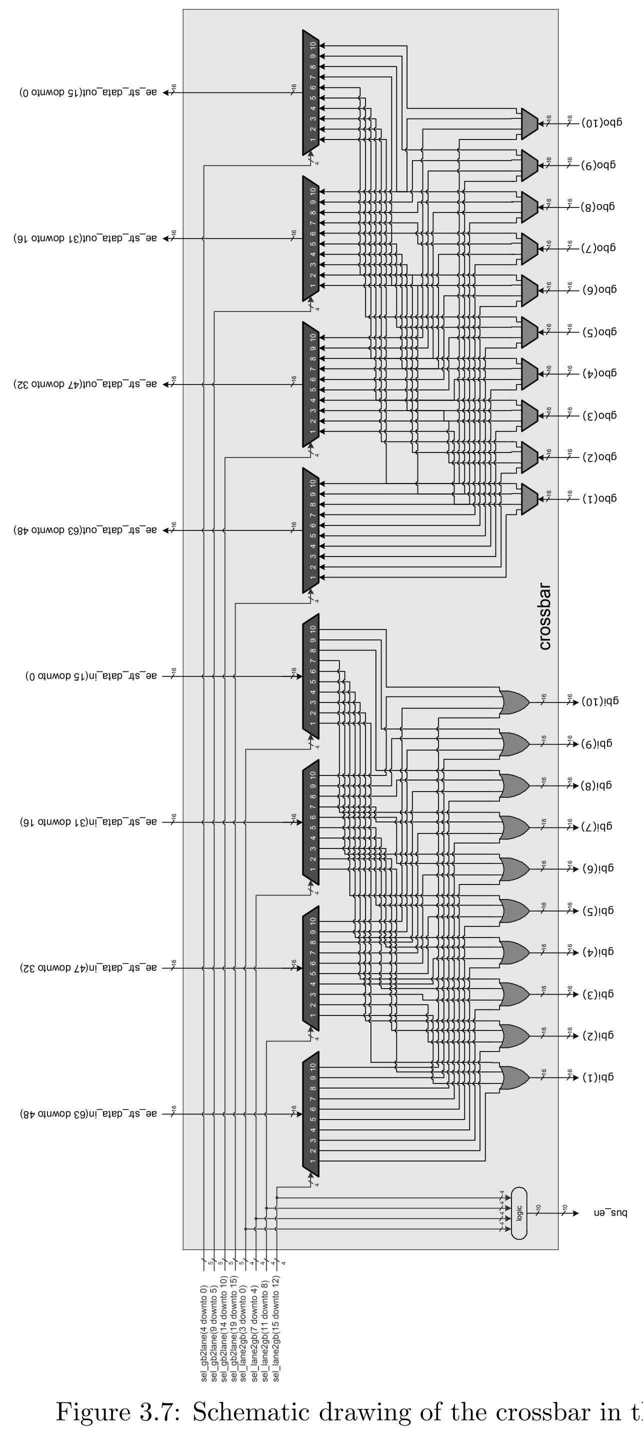
sel_gb2lane(4 downto 0)
sel_gb2lane(9 downto 5)
sel_gb2lane(14 downto 10)
sel_gb2lane(19 downto 15)
sel_lane2gb(3 downto 0)
sel_lane2gb(7 downto 4)
sel_lane2gb(11 downto 8)
sel_lane2gb(15 downto 12)
ae_str_data_in(63 downto 48)
ae_str_data_in(47 downto 32)
ae_str_data_in(31 downto 16)
ae_str_data_in(15 downto 0)
ae_str_data_out(63 downto 48)
ae_str_data_out(47 downto 32)
ae_str_data_out(31 downto 16)
ae_str_data_out(15 downto 0)
logic
bus_en
crossbar
gbi(1)
gbi(2)
gbi(3)
gbi(4)
gbi(5)
gbi(6)
gbi(7)
gbi(8)
gbi(9)
gbi(10)
gbo(1)
gbo(2)
gbo(3)
gbo(4)
gbo(5)
gbo(6)
gbo(7)
gbo(8)
gbo(9)
gbo(10)

Figure 3.7: Schematic drawing of the crossbar in the CCU

uses 16 bits words, there are 2 register views: one view to fill the registers with 16-bits words (called configuration view) and 1 view which maps the registers to the functional width (called normal view). The mapping strategy between those views changed during revisions of the Montium TP. In this project, the new mapping strategy is used, as shown in Figure 3.9 and Figure 3.8 [7].

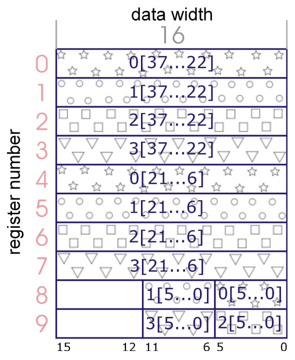

Figure 3.8: Configuration view of the configuration registers

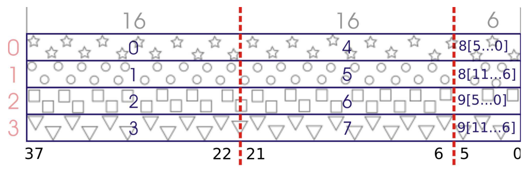

Figure 3.9: Normal view of the configuration registers

During (re)configuration of the Montium TP, the SIO registers inside the CCU are also configured. Using the SIO lines, the Montium TP selects one of the 8 configured interconnect configurations. According to the selection of the SIO lines, the LANE2GB and GB2LANE signals control the multiplexer. Because the SIO configuration is stored in flipflop registers, a memory access is required to obtain the selected SIO configuration. This memory access of the SIO memories takes one clock cycle. After the Montium TP made a change to the SIO lines, the interconnect is configured after 1 cycle. During this SIO memory access the Montium TP has to be paused. More about pausing the Montium TP and reaction time can be found in the next section and Section 3.2.3.

Pausing the Montium TP

The Montium TP has to be put on hold by the CCU in three situations:

- When the SIO lines are changed
- When the NoC is not ready to accept output from the Montium TP
- When an input signal is not ready for input to the Montium TP

Holding the Montium TP as a result of changing SIO lines The values on the SIO lines are delayed 1 cycle to SIO_OLD. The value on SIO_OLD is also delayed 1 cycle to SIO_OLDER. When the values on the (delayed) SIO lines are not identical, the Montium TP is paused, as shown in Figure 3.10.

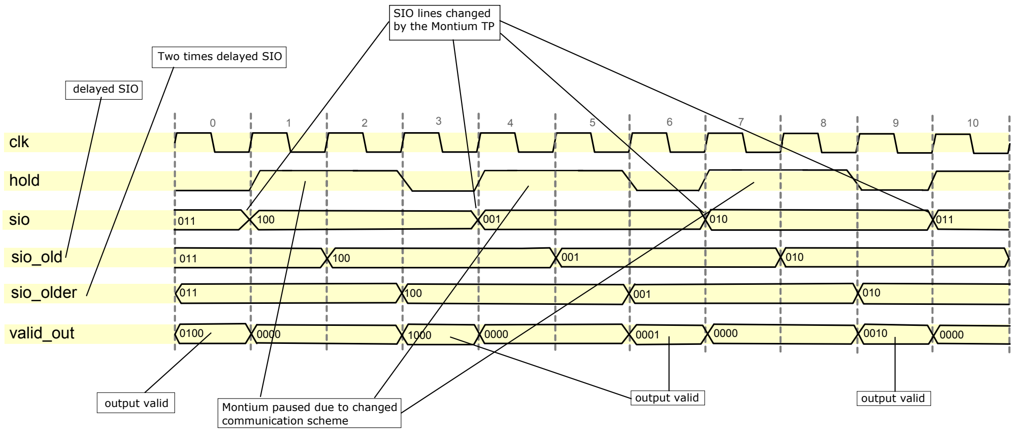

Figure 3.10: Waveform with changing communication scheme

Holding the Montium TP as a result of NoC not ready to accept output from the Montium TP By the SIO signals the Montium TP selects a communication path. Bits 19 downto 15 are for busses to lane 0, bits 14 downto 10 are for busses to lane 1 etcetera (see Appendix A for more details). When a group of bits is 00000 the lane is disabled (unused), just OR-ing the individual bits gives a signal which is only high when the Montium TP tries to output data to the corresponding lane. This signal is connected via an AND port to the VALID line of the corresponding lane. When the VALID line is high and the ACCEPT is low, the NoC does not accept data, which has to result in a stalled Montium TP. This hold signal is made by inverting the ACCEPT signal and AND-ing this with the VALID signal. In Figure 3.11 a waveform is shown with congestion at the streaming lane. CCU1 is transmitting on a single lane, while CCU2 has not started receiving samples yet. The FIFO between the streaming interfaces of both CCUs has a depth of 8 words. After 8 words the FIFO is full and CCU1 sets Montium1 on hold until the congestion is solved.

Holding the Montium TP as a result of input signal not ready for input to the Montium TP Almost the same method as explained in the previous paragraph can be performed here: Bits 35 downto 32 controls the connection to the busses of lane0, bits 31 downto 28 control the connection of lane1 etcetera (see Appendix A for more details). All zeros in a group of bits is a disabled connection. When the bitswise OR of a 'lane2gb-group' results in a logic high signal, the connection is used. This signal can be used to drive the ACCEPT signal, when the Montium TP is not stalled. The bitswise OR of a 'lane2gb-group' has to be AND-ed with the inverted other signals which can cause the Montium TP to hold. This structure to hold the Montium TP

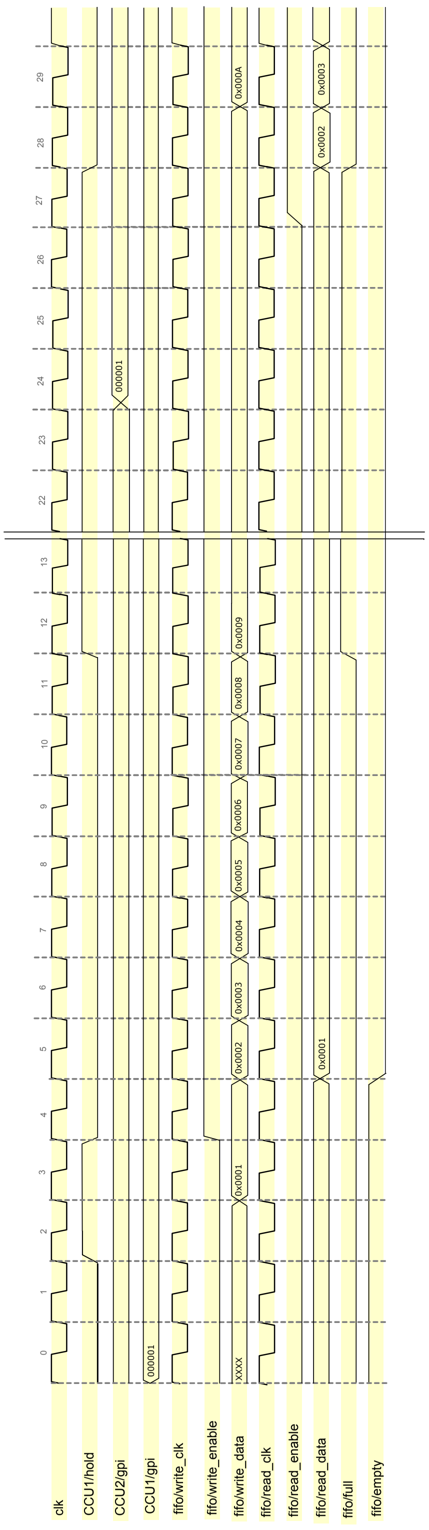

Figure 3.11: Montium on hold due to congestion on streaming lane

and controlling the VALID and ACCEPT lines of the network is visualized in Figure 3.12.

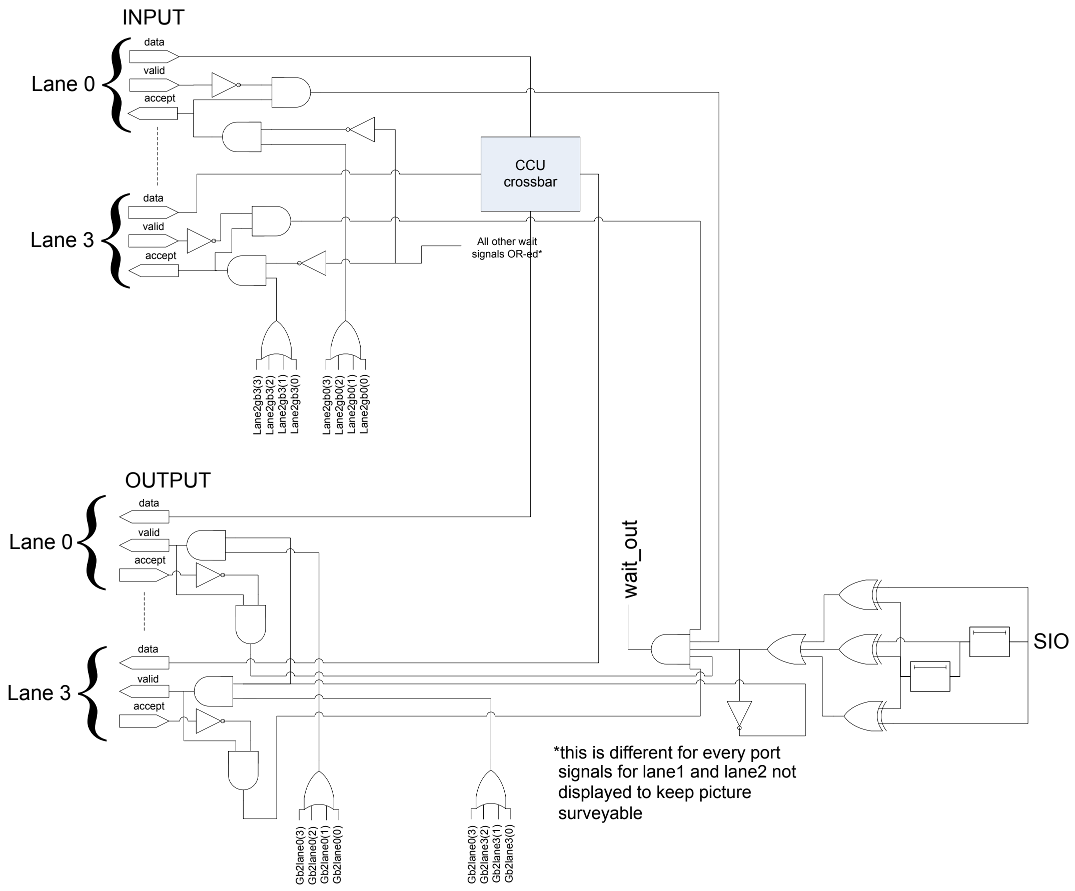

Figure 3.12: Logic responsible for pausing the Montium TP

3.2.3 Latency of streaming interface

As mentioned before the reaction time of the streaming interface is important, because it is the fastest connection to the Montium TP. Low-latency applications will use this interface. Because no buffering of the streaming signals is performed in the CCU, the streaming interface of the CCU is fast. There are only 2 cycles delay when the communication scheme changes. While the communication scheme remains unchanged, the data from the network is directly sent to the Montium TP busses without adding extra delay.

3.3 FPGA tests

This section explains how an application can be loaded into the Montium TP when the design is running on an FPGA and ways to test for correct functionality.

3.3.1 ML605 Evaluation Board

For the FPGA tests an ML605 Evaluation board from Xilinx is used. Its key specifications are:

- Virtex-6 XC6VLX240T-1FFG1156 FPGA
- 512 MB DDR3 Memory
- 16 MB Platform Flash XL
- 32 MB Linear BPI Flash
- USB JTAG
- 10/100/1000 Tri-Speed Ethernet PHY

3.3.2 Xilinx MicroBlaze Debugger

XMD can connect to the MicroBlaze using a Serial Interface or a Joint Test Action Group (JTAG) interface [14]. Via XMD, words can be written to addresses or read from addresses. Also the contents of the MicroBlaze registers and other internal values of the MicroBlaze like the value of the Program Counter can be retrieved by XMD. When many values have to be written, for instance to load an application, a Tool command language (Tcl) script can be used to control XMD commands.

3.3.3 Starburst S-Record Loader

A file format, called SREC is an ASCII text encoding for binary data. Because of the ASCII encoding the files can be edited with a text editor. Also a checksum is added to each record to be able to be aware of data being corrupted during transmission. An S-Record loader is available on the Starburst SoC. Using this S-Record loader, data can be transferred between a PC and the evaluation board using a Universal Asynchronous Receiver-Transmitter (UART) connection. This way data can be stored in for example the 512MB DDR3 memory the ML605 evaluation board is equipped with. Within this project an application is designed to convert the output from MontiumLLL to input usable for SREC.

Chapter 4

Realization

In this chapter the results are discussed. This chapter covers the hardware resource usage and speed of the CCU embedded in the Starburst SoC. In the last section of this chapter, a comparison with the Hydra is made.

4.1 Hardware design

The hardware consists of four processing cores: 2 MicroBlaze cores and 2 Montium TP cores which are interconnected by the Æthereal NoC. The MicroBlazes are connected to the NoC by a DTL interface. This connection is implemented in Starburst SoC Generator by adapters connected to the Processor Local Bus (PLB). The Montium TP configuration interfaces are connected by a DTL interface and the Montium TP streaming lanes are connected using the streaming interface of the NoC. The hardware discussed above is visualized in a block diagram in Figure 4.1.

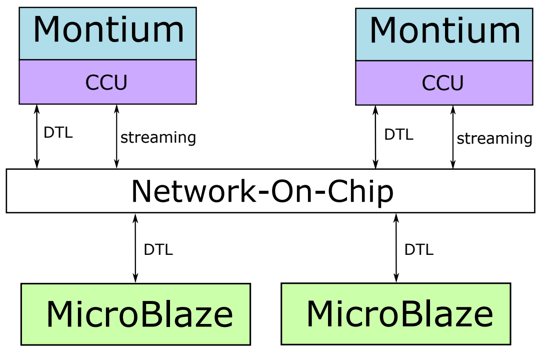

Figure 4.1: Cores connected by Æthereal for a test application

4.2 Clock frequency

The Intellectual Property (IP) containing the Montium TP, CCU and (optional) DTL adapter are synthesized and placed and routed with a clock speed constraint of 12.5 MHz on the Virtex-6 FPGA as on the Xilinx ML605 evaluation board. The place and route report gives a maximum period of 67.465 ns which corresponds to a frequency of 14.82 MHz. The reason why a clock frequency of 12.5 MHz (80 ns period) is chosen is the better alignment with other clocks.

4.3 Resource usage

The resources used by the CCU, the Montium and the DTL adapter at 12.5 MHz (after place and route) on a Virtex-6 XC6VLX240T-1FFG1156 FPGA are shown in table 4.1.

Cell	Montium usage	CCU usage	DTL adapter usage
Slice registers	3066	449	35
LUTs	15986	468	112
BRAMs	13	0	0
DSP48E1	5	0	0

Table 4.1: Resource usage Montium, CCU and DTL adapter

According to requirement 5 the CCU area must be smaller than 5% of the Montium TP area. This area constraint is more important for an ASIC than for an FPGA, because area on an ASIC is expensive. Therefore an ASIC synthesis is performed. For an FPGA the resource usage is less important as long as the design fits in the FPGA. With the results from the ASIC synthesis, a comparison between the area of the CCU and the area of the Montium TP is made. Using the synthesis results also a comparison with the Hydra is made.

4.3.1 ASIC

The CCU and DTL adapter are synthesized and placed and routed using the TSMC 90 nm low power high performance ultra high Vt library, with a clock frequency constraint of 400 MHz. The process used is TSMC 90 nm Logic low-power(1P9M,1.2V) Design Rule. More of the settings used for obtaining these results are mentioned in Table 4.2.

Parameter	Setting
Clock gating	Off
Core utilization	0.7
Power ring width	10
Number of metal layers	7
Place & route effort	high

Table 4.2: Synthesis settings

The results after synthesis and place and route are shown in Table 4.3.

A piechart of Table 4.3 is shown in Figure 4.2. The largest part inside the CCU is the streaming control entity. This large area is the consequence of the SIO memories which are inside this entity. Those memories are implemented

Component	# gates	Area (mm^2)	Area (%)
Address decoder	406	0.00086	5.3
Crossbar	2072	0.00439	27.0
DMA inputdata multiplexer	199	0.00042	2.6
DMA interface	467	0.00099	6.1
Sequencer	185	0.00039	2.4
Streamingcontrol	3651	0.00773	47.5
DTL adapter	702	0.00149	9.1
Total	7682	0.01627	100

Table 4.3: Synthesis results of the CCU and DTL adapter using 90 nm libraries

using flip-flops, which are area-hungry. The SIO memories hold 8×38 bits = 308 bits = 38 Bytes. The crossbar is also a large component with 27.0 % of the total CCU area.

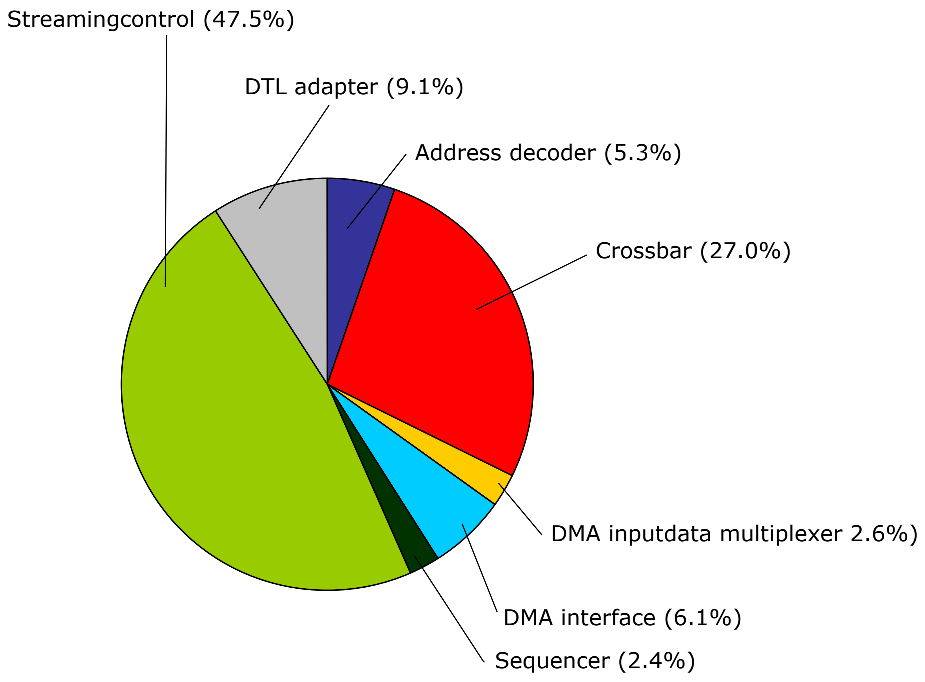

Figure 4.2: Pie chart of the area usage of components in the CCU

4.4 Comparison with the Hydra

In [11], the results as in Table 4.4 are presented:
It is obtained by synthesizing the VHDL model in 0.13 μm Atmel technology with a clock frequency constraint of 200 MHz. A few differences and similarities are important to mention. The crossbar as used in the Hydra and the CCU designed in this project are the same: they both multiplex 10 Montium General

Component	# gates	Area (mm^2)	Area(%)
Crossbar	1810	0.010	9.5
Flit formatting	3695	0.021	19.3
Flow control	3893	0.022	20.4
Buffering	7920	0.044	41.5
Message execution	1788	0.010	9.4
Total	19106	0.106	

Table 4.4: Synthesis results of the Hydra

Busses to 4 Network lanes. The CCU designed within this project is a lot smaller than the Hydra, mainly due to the absence of Command ROMs and the absence of Address Generation Units (AGUs) in the CCU. Because of the different features, a fair comparison is hard to make. It is more important the CCU is small compared to the Montium, as formulated in requirement 5. This requirement is evaluated in the next section.

4.5 CCU area compared to the Montium TP

An estimate is made of the area usage of the Montium TP in 90 nm technology. The Montium has an area usage of 1.8 mm^2, when synthesized and placed and routed in 130 nm technology [8]. The crossbar as used in the Hydra is the same as the crossbar used in this project, although the number of gates are different. This is likely due to the CCU is synthesized as being on the outside of the chip. This results in extra transistors added for sufficient output drive capability, which accounts for the difference in gates. In the Hydra the crossbar needs an area of 0.01 mm^2. In 90 nm the crossbar needs an area of about 0.0044 mm^2. This is an area reduction of 66%. Using the same area reduction for the Montium, the Montium has an estimated area usage of 0.612 mm^2. The CCU including DTL adapter, with a total area of 0.01627 mm^2, uses about 2.7 % of the estimated Montium TP area in 90 nm technology.

4.6 Data rate

In this section the data rates of the DTL interface and streaming interface are treated. The measurement is done by simulation in Questasim.

4.6.1 DTL interface

For testing the data rate of the DTL interface 1000 words are written to the DTL interface and 1000 words are read from the DTL interface. Reading is slower than writing, because for reading two transfers over the NoC are necessary (a command transfer from MicroBlaze to Montium TP and a transfer of data to be read back to the MicroBlaze). For writing only transfers (a command transfer and a transfer with the actual data to be written) from MicroBlaze to Montium TP are necessary. There is no transfer from Montium to MicroBlaze, as is the case by a DTL read. Therefore writing is faster.

Reading

Reading 1000 words of 16 bits from the local memories or register files of the Montium by a MicroBlaze processor took 1920480 ns, which corresponds to 192048 clock cycles at 100 MHz and about 24006 clock cycles at 12.5 MHz. This corresponds to a data rate of 0.99 MB/s. The influence and possibilities of this data rate are treated in Chapter 5.

For a one word read from a local memory in the Montium TP by the MicroBlaze, the time is measured. By measuring the time the commands arrive at the various busses involved by the data transfer, an indication can be obtained concerning which transfer takes the most time. The results of a read transaction are displayed in Table 4.5. A large delay occurs in the CCU and DTL adapter, this is due to a low clock frequency of 12.5 MHz for the Montium processing tile, in combination with the number of words that have to be written for a DTL transaction, due to the compatibility with the streaming interface. For a DTL read transaction 3 words have to be written: to DMA address 2 for enabling the correct bus and GB2LANE setting, to DMA address 1 for selection of the right address and memory and enabling DMA, and to DMA address 1 again, to turn off DMA (see Appendix A).

Component	Delay (ns)	Clockcycles	Fraction of total delay
Microblaze adapters	70	7	4%
NoC	530	53	29%
CCU+DTL adapter	800	10	44%
NoC (retour)	430	43	23%

Table 4.5: Measured delay introduced by different components during DTL read transaction

The delay of the CCU+DTL adapter during a DMA read can be seen in the waveform in Figure 3.6. It takes 10 cycles for a DTL read from the command transfer at the DTL adapter, until the data to be read is available at the NoC NI. In cycle 0, the command is transferred. In cycle 1, the DTL adapter is in the PREPAREREAD-state (see state diagram of the DTL adapter in Figure 3.3). In cycles 2, 3, 4, 5 and 6 the correct values are placed in the registers of the CCU. In cycle 7 and 8 the actual DMA transfer is performed and in cycle 9 the retrieved data is available at the DTL_RD_DATA line. This is 10 clockcycles, which takes 800 ns when a clock frequency of 12.5 MHz is used.

Writing

Writing 1000 words of 16 bits to the local memories or register files of the Montium by a MicroBlaze processor took 719040 ns. This corresponds to 71904 clock cycles at 100 MHz and 8988 clock cycles at 12.5 MHz. This corresponds to a data rate of 2.65 MB/s. The usefulness of this data rate in an application is described in Chapter 5. The arriving and leaving of signals on various busses, belonging to the write transaction, are measured. The results of this measurement are written in Table 4.6. Again, a large part of the total delay (59%) occurs in the CCU and DTL adapter. This is a consequence of the implementation of the requirement to be compatible with DTL and streaming for the configuration and DMA interface. For a write command, 5 transfers are necessary: the data to be written to DMA address 3, the correct setting of the LANE2GB signals in DMA address 4, a transfer to DMA address 2 for the correct bus enable and two writes to DMA address 1. One to select the right address and memory and perform the actual transfer, and one to turn off DMA. When these low data rates for DTL are a problem for a certain application, it would be better to integrate the DTL adapter into the CCU, instead of implementing it as a separate adapter. The drawback of support for DTL integrated into the CCU is that it conflicts with requirement 1 (see Section 2.7) for the MMIO interface. An in the CCU integrated DTL implementation makes it possible to make a direct connection between the DMA signals and reduces the amount of different words that have to be written each DTL transaction. Another possibility is to add more configuration streaming interfaces to the CCU and the DTL adapter to make it possible to write more words per clock cycle.

When the NoC and MicroBlazes are assumed to be ideal, this means communication over the NoC and on the MicroBlazes adapters take no time, it takes 9 clock cycles for a word to be written to the local memories of the Montium. This corresponds to a data rate of 2.65 MB/s.

Component	Delay (ns)	Clockcycles	Fraction of total delay
Microblaze adapters	70	7	6%
NoC	420	42	35%
CCU+DTL adapter	720	9	59%

Table 4.6: Measured delay during DTL write transaction

4.6.2 Streaming interface

For this streaming performance measurement only 1 lane at a time is used, but between every transfer is switched between lanes. This switching between communication schemes introduces 2 cycles extra delay per switch operation (see Figure 3.10). Due to this delay, it is important to measure the performance when switching between communication schemes occurs very often. For this performance measurement 100 words are transferred between Montium1 and Montium2. The time between the first VALID and the last ACCEPT is measured. This took 23920 ns. In this time 200 Bytes are transferred (100 words of 16 bits). This results in a datarate of 7.97 MB/s for the streaming interface using a single lane.

When only a single lane is used, the data transfers are faster, because no delay of communication scheme switching is involved. A measurement of transferring 200 Bytes, using the same lane, takes 8080 ns. This corresponds to a datarate of 23.61 MB/s, which is about 3 times more than the 7.97 MB/s when continuous switching between communication schemes occurs.

A measurement, using 4 lanes in parallel and a constant communication scheme transferring 800 Bytes (200 Bytes per lane), took in simulation with the Montium core running at 12.5 MHz 8080 ns. This corresponds to a datarate of 94.44 MB/s, which is exactly 4 times the throughput of a single lane.

Chapter 5

Application

5.1 Introduction

To be able to test the functionality of the CCU and to be able to show that the CCU functions, a small application is mapped on a few cores which are interconnected by a NoC. The code coverage of this application is presented in this chapter, as well as data rate tests on an evaluation board.

5.1.1 Practical application information

In the system as realized on the Xilinx ML605 evaluation board the Montium TP cores are configured by the MicroBlazes. It is not necessary to have a MicroBlaze processor for each Montium TP core, because one MicroBlaze core can configure two Montium TP cores successively, but by adding two MicroBlazes the Montium TP cores can be configured simultaneously, which reduces (re)configuration time. This can be necessary in timing-critical applications. Streaming communication can be done between the two Montium TP processors. Data to be processed as well as twiddle factors or filter coefficients can be loaded in the Montium TP by DMA via the DTL interface. This DTL interface is much slower than the streaming interface (see Section 4.6). In addition to this, the Montium TP is stalled during DMA transfers. Therefore, this interface can best be used at the beginning or end of an application and is not well suited to use during streaming data processing. During normal execution of an application, the CCU connects lanes on the NoC side to global busses of the Montium TP according to an SIO instruction. Which lane is connected to which global bus is stored in the SIO memory during the configuration phase (see Section 3.2.2).

Data rate calculations for beamforming example

As mentioned in the introduction, an application for the Starburst SoC extended with Montium TPs is beamforming. Therefore data rate calculations with a beamforming application are discussed in this section. It gives insight in the performance of the CCU in a practical application. When a beamforming application is mapped onto the Montium TP cores, during normal operation the Montium TP will be busy with streaming data processing, like FIR-filtering. To be able to perform calculations, first assumptions need to be made in order to simplify the problem. In a beamforming application a signal is received on multiple antennas. In the case of smallband beamforming, a phase shift must be added to every signal, which can be implemented by a complex multiplication on every incoming sample.When for example an 8x8 antenna array is used, for every antenna element a complex number must be stored [6]. For 64 antenna elements, 128 words have to be stored in the local memories of the Montium TP for the delays. When the transmitter or receiver is moving, it may become necessary to change the direction of the beam. When the direction of the beam has to be changed, new coefficients have to be loaded into local memories of the Montium TPs, to steer the beam in the new direction. With a data rate of 2.65 MB/s this takes about 92 μs. The Montium TP is not paused for this whole 92 μs, but only for about 10.24 μs, because the Montium TP is only paused for 1 clock cycle for every word that is written and 2 clock cycles for every word that is read, as shown in the waveform of Figure 3.6. When for instance 1 local memory is filled with the result of an FFT which is performed on incoming samples and this FFT-data is read by the MicroBlaze, the MicroBlaze has to retrieve 2kB of data (1024 words of 2 Bytes), which takes about 2 ms. During this 2 ms the Montium TP is paused for only 163.84 μs.

5.2 Communicating test algorithm on the Montium

To show the CCU is working like described, a small application is mapped onto the Montium TP cores, making use of the following features the CCU offers :

- a DTL write transaction to a local memory of the Montium TP
- a DTL read transaction from a local memory of the Montium TP
- a streaming transaction which includes switching between lanes

The MontiumLLL source code is included in Appendix C.

5.2.1 Code coverage

The code coverage feature of Questasim measures how often certain aspects of the source code are exercised while running a test. It gives an indication what percentage of the code is tested with the application. There are several types of coverage, the coverage types measured with the coverage test within this project are discussed first:

Statement coverage Counts the execution of each statement on a line individually, even if there are multiple statements in a line. Statement coverage is the most basic form of coverage supported by Questasim.

Branch coverage measures branches constructed by “if” and “case” statements. Both true and false branches are measured. For 100% branch coverage, each branching statement in the source code must have taken both the true and false paths.

Condition coverage analyzes the decisions made in “if” statements and can be considered as an extension to branch coverage. The conditions determining whether the body of the if statement is executed or not are analyzed with the condition coverage option.

Expression coverage analyzes the expressions on the right hand side of assignment statements. This can be logic expressions, for example for an 2-

input OR there are 4 different input vectors possible [1].

The coverage results of the test application as shown in Appendix C are shown in Table 5.1. Not all entities contain the statements analyzed by a certain coverage type, therefore the coverage report gives - -, for those entities.

Component	Statement	Branch	Expression	Condition	Total
Address decoder	100%	100%	- -	100%	100%
Crossbar	100%	56%	- -	31%	60%
DMA inputdata multiplexer	100%	100%	- -	- -	100%
DMA lane2gb multiplexer	100%	100%	- -	- -	100%
DMA gb2lane multiplexer	100%	50%	- -	67%	67%
DMA interface	83%	81%	- -	67%	72%
Sequencer	91%	71%	100%	33%	70%
Streamingcontrol	81%	90%	47%	100%	75%
Weighted Average	89%	71%	49%	54%	61%

Table 5.1: Code coverage results of test application

The coverage report also gives numbers, from which a weighted average is calculated. Those numbers are shown in Table 5.2.

Coverage type	Bins	Hits	Misses	Coverage
Statement	332	295	37	89%
Branch	208	148	60	71%
Expression	146	72	74	49%
Condition	109	59	50	54%

Table 5.2: Weighted averages for code coverage test

The coverage of the expression in the streamingcontrol is with 47% one of the lowest coverages. This includes the logic to hold the Montium TP on time and select the right signal for the crossbar. Also the coverage of the crossbar itself is with 56% branch coverage and 31% condition coverage not high.

Although the test application is rather simple, the code coverage results are quite high. This is because of the application mainly consists of communication instructions, this is the only task the CCU is involved in. The behaviour inside the ALUs is tested within the design of the Montium TP, therefore it is unnecessary to test it in a CCU test application.

5.3 Data rate tests on evaluation board

For this data rate tests a large number of words are written and read to and from the Montium TP local memories. The start and end signals of the transactions are signaled using flashing LEDs.

For the writing data rate test 200.000.000 words are written by the MicroBlazes to the Montium TP local memories on the evaluation board. This took 144 seconds. This corresponds to a write speed of about 2.65 MB/s.

Reading 100.000.000 words from the Montium TP local memories took 192 s on the evaluation board. This corresponds to a data rate of about 0.99 MB/s.

The on the evaluation board measured data rates are the same as the data rates measured in simulation. Due to the large number of words used for these tests, the reaction time does not cause a visible influence.

Chapter 6

Conclusions and recommendations

In this chapter the results achieved within this project are evaluated. A working demonstration of a CCU connecting the Montium TP to Æthereal is achieved within this Master's assignment. The demonstration consists of a small application executed on the ML605 evaluation board containing a Virtex-6 FPGA. However, there is still room for improvement on some features of the CCU, which are left as a recommendation.

6.1 Conclusion

A working prototype with 2 Montium TPs connected with a DTL interface to the NoC and interconnected to each other by 8 streaming lanes (4 input and 4 output) is realized on the Xilinx ML605 evaluation board, with the Montium TP and CCU running on 12.5 MHz. A small communication application is succesfully executed on the platform. For the configuration data, sequencer control and DMA transfers, a streaming interface or DTL interface can be chosen at design time. This is made possible by an optional DTL adapter which converts DTL-commands into streaming commands for the CCU. The datarate of the streaming interface is 7.97 MB/s per lane when the Montium TP runs at 12.5 MHz and every transfer is switched between communication schemes. This is a worst-case scenario, because when a constant communication scheme is used, the datarate increases to 23.6 MB/s. For an application, the performance lies somewhere in between these numbers, dependent on the number of switches between communication schemes. For the DTL interface the data rate is 0.99 MB/s for reading and 2.65 MB/s for writing when the Montium TP runs at 12.5 MHz. This rather low datarate is caused by the number of words to be written via the streaming interface by the DTL adapter for a DMA transfer at a low clock frequency.

6.2 Requirement evaluation

Nr.	Requirement description	Result	Comment
1.	Able to transfer data via streaming interface NoC	$\surd$	
2a.	DTL interface for sequencer	$\surd$	
2b.	DTL interface for config data	$\surd$	
2c.	DTL interface for DMA transfer	$\oslash$	Only for transfers of $>$ 1 Byte
3.	No buffering on streaming interface inside CCU	$\surd$	
4.	Capable of transferring data every clock cycle on streaming interface	$\pm$	Only when communication scheme is constant
5.	CCU area smaller than 5% of Montium TP	$\surd$	Only 2.7 % of Montium TP
6.	Clock frequency CCU same as Montium TP	$\surd$	12.5 MHz on ML605 board
7.	Critical path not inside the CCU	$\surd$	Critical path in Montium ALU
8.	Compatible with MontiumLLL	$\surd$	

$\surd$	Requirement met
$\pm$	Requirement not met
$\oslash$	Requirement not met, room for improvement

Table 6.1: Conclusions according to requirements formulated in chapter 2

The requirements as established in Chapter 2 are summarized and evaluated in Table 6.1.

Some requirements require more explanation, which is given in this section.

Because a direct connection, without using Æthereal, is used for streaming interconnection between the two Montium TPs, FIFOs are inserted, otherwise timing violations occur. This is caused by a long combinatorial path between an ALU of Montium 1, via the streaming interface to an ALU of Montium 2, via the streaming interface back to Montium 1 etcetera. Buffering by connecting a First In First Out (FIFO) between the streaming interfaces of both CCUs reduced the length of the combinatorial path.

Requirement 2a and 2b cannot be completely compatible with the DTL specification, because reading from configuration addresses is not possible in the Montium TP. The configuration interface also does not support single byte transfers.

For requirement 2c it is possible to implement single byte transfers, because reading from memories is possible via DMA, but is not implemented.

For requirement 4 only data can be transferred every cycle if the communication scheme is constant, because the Montium TP is paused for 2 cycles when a switching between communication schemes occurs.

Requirements 5, 6, 7 and 8 are satisfied.

The dual-interface requirements for the sequencer, configuration data and DMA transfer (requirement 1 and 2) are the main cause for low datarates for DMA transfers.

6.3 Recommendations

In this section some modifications for more efficient use of the Montium TP in an Æthereal NoC are suggested. Also some pitfalls when the CCU is further integrated within the tool flow are mentioned.

6.3.1 Streaming

The Montium TP is the first IP equipped with streaming interfaces in the Starburst SoC. All data to be processed by the Montium TP must be transferred via DMA. During a DMA transfer the Montium TP is paused. Every word that is read from the Montium TP memories, the Montium TP is paused for 2 clock cycles. For every word written to the Montium TP, the Montium TP is paused for only 1 clock cycle. Also as mentioned in Section 4.6, the bandwidth of the DTL interface is smaller than the bandwidth of the streaming interface. Therefore, pausing the Montium during normal operation must be reduced to a minimum, because it reduces the performance. To be able to deliver data to the Montium TPs, without pausing the Montium TP, more different cores with streaming interfaces have to be added to the SoC. Possible useful cores to extend the SoC with are discussed in the subsequent sections.

Analog to Digital Converter

With an Analog to Digital Converter (ADC) an analog signal can be converted to a digital signal which can be processed using DSP techniques. In case of a beamforming demonstrator, the signal received at the antenna is digitized (in many cases after analog preprocessing). As an ADC delivers a continuous stream of data, a streaming interface is a suitable interface to connect an ADC to the NoC. The ADC can be the source of data to be processed by Montium TPs.

Digital to Analog Converter

When not only receiving signals, but also transmission of signals is necessary, a Digital to Analog Converter (DAC) can be a useful addition of the SoC. A DAC converts a digital sample into an analog value. When a DAC is interfaced to the NoC using a streaming interface, the Montium can provide a continuous stream of data to the DAC.

Streaming memory

A streaming memory can be a useful method to interface between GPPs and Montium TPs without using DMA. By extending the SoC with a streaming memory a streaming data source for the streaming interface of the Montium TP is available. This streaming memory can be useful during testing of applications mapped to Montium TPs, because (ideal, noise-free) samples can be generated and stored in this streaming memory. After the algorithm is working correctly, signals from an ADC can be used to verify correct functionality.

6.3.2 DTL adapter

In this section recommendations for improvements of the DTL adapter are considered.

Single Byte transfers

Within the DTL specification it is possible to read or write only a single Byte. This is not implemented into the DTL adapter, because the Montium only supports writes of complete words for the configuration interface. Therefore only support of memory accesses of at least 2 Bytes are supported. For DMA transfers only the 16 least significant bits are really used. For configuration, data transfers of 4 Bytes are necessary, because the 16 most significant bits are used as address and the 16 least significant bits as data. For the DMA transfers it is possible to support single Byte transfers, because in contrast to the configuration interface, reading from the Montium is supported at the DMA interface. This 16 bits word read from local memories can be stored in the same memory location, after one of the two Bytes is changed. When the Montium DTL adapter is used in a system in which it is unsure whether single Byte transfers are used, the DTL adapter needs to be extended with single Byte transfer capability.

Data rate

The data rate of DMA transfers using the DTL adapter is with 0.99 MB/s for reading and 2.65 MB/s for writing not high. This is mainly due to the requirement DMA transfers must be supported for a streaming interface as well as a DTL interface, which resulted in an implementation using an optional adapter. This adapter had a large negative influence on the achievable bandwidth. When more bandwidth is required, it is recommended to implement the DTL support inside the CCU in addition to the streaming interface, instead of an adapter for the streaming interface.

6.3.3 Parameterizability

Within the VHDL source code of the DTL adapter, the base addresses of the Montium TP cores are hardcoded instead of using addresses as specified in the XML file from which the system is built. When more Montium tiles are added to the MP-SoC or when they are mapped on other addresses, it is necessary that these addresses are not only updated in the XML file describing the system, but also in the DTL adapter.

Appendix A

CCU design specification

This appendix provides detailed information about the signals and registers in the CCU. The information in this appendix added with some introductory text is also available as as a standalone document called "CCU design specification" which gives all necessary information to use the CCU.

A.1 Æthereal Network Interfaces

The Æthereal NoC provides two types of NIs: A streaming NI with only three lines: DATA, VALID and ACCEPT and a bus protocol. The only bus protocol implemented in Starburst is DTL. The DTL interface is perfectly suited for MMIO.

The sequencer interface and configuration interface require only one-way communication (only from the network to the tile). The sequencer interface and the configuration interface are both MMIO. Via this interface also the CCU can be configured (e.g. the memory in the SIO decoder). The CCU is native equipped with a streaming interface target which can be connected directly to the NoC. When communication via DTL is desired, the CCU can be connected to a DTL to streaming adapter. This adapter has two sides: a DTL side and a streaming side. The streaming side is connected to the CCU and the DTL side can be connected to the NoC.

A.1.1 Number of network lanes

For the streaming data streaming NIs are used to save the overhead of a DTL interface. Although the Montium processor has 10 buses, it is unlikely the Montium outputs data every cycle on all 10 buses. Therefore the number of NIs can be smaller. This greatly reduces the size of the multiplexers in the CCU.

In the Hydra CCU there are 4 input lanes and 4 output lanes, each 16 bits wide. This choice is made due to the network topology used at the NoC the Hydra is connected to. With 4 input lanes it is possible to load two complex values each clockcycle, which will be enough for most filtering and FFT applications.

Compiler

Generating configuration data for the Montium is done using the same compiler as used for the Hydra. The Hydra makes use of 4 input and 4 output lanes. Therefore, when using 4 input and 4 output lanes or less for the Æthereal CCU, the same compiler can be used. When more lanes are necessary, modifications to the compiler have to be done. This will probably not be an easy task, because only an executable without configuration file is available. When one tries to compile code using for example lane5 the compiler gives the following error:

```
Instr:  mov "data" "p1o1" -> "ext5"
Flit types can not be used here.
(Only when moving to an external communication identifier.)
Abort. (error code: 1)
```

A.2 TP interface

The Montium TP contains 5 ALUs.

For the connections to the Montium, the interface used in the Hydra is used as a starting point.

A.2.1 System signals

Signal	Usage	Description
clk	-	Clock
rst_hw	low	Active low synchronous reset
	high	Normal operation

A.2.2 Sequencer interface

Signal	Usage	Description
Resetn	low	Active low synchronous reset
	high	Normal operation
Wait_TP	low	Enable the Montium TP
	high	Freeze the Montium TP
DV	low	Data not valid
	high	Data valid
SIO	3-bit output	Control lines for streaming
GPI	6-bit input	General purpose input
GPO	6-bit output	General purpose output

A.2.3 Configuration interface

There are four zones in the total configuration map:

- Sequencer
- Decoders
- PPA
- CCU

The two Most Significant Bits (MSBs) select the right configuration memory. Not all configurable entities have a width of 16 bits. To avoid complicated situations, the configuration memory can be viewed in two ways: the normal view and the configuration view. In the normal view the entity has the dimensions as implemented in the Montium TP and in the configuration view an entity has a width of at most 16 bits. For more information about the mapping of those register views, look at page 97 of [4]. [1]

Signal	Usage	Description
c_dv	low	Do not configure
	high	Configuration data valid
c_data	16 bit data	Configuration data
c_addr	16 bit address	Configuration address

A.2.4 DMA interface

The signals of the DMA interface are described in Table A.1.

Signal	Usage	Description
dma_sel	low	Do not use DMA: the TP is enabled
	high	use DMA: the TP is frozen (similar to the hold signal of the sequencer interface
dma_rw	low	DMA read action (when dma_sel is active)
	high	DMA write action (when dma_sel is active)
dma_addr	10-bit address	address within local memory (10-bit) or register file (2-bit)
dma_mr	low	Select memories (when dma_sel is active)
	high	select registers (when dma_sel is active)
dma_rs	low	Select register files A and C (when dma_sel is active)
	high	Select registers B and D (when dma_sel is active)
bus_en	10-bit mask	Vector of 10 flags, of which bus_en(1) is the MSB and bus_en(10) the LSB. bus_en(1) enables PP1.MemLeft, PP1.RFA or PP1.RFB, depending on dma_sel, dma_mr and dma_rs.

Table A.1: DMA interface signals

A.3 Sequencer

The address for the sequencer interface is 0xD000 (53248 unsigned). They are connected to the sequencer outputs as described in Table A.3.

The Montium device is resetted when the RESETN line is driven low. After a software reset, the most occuring situation is that the Montium will start running an application. The RESETN must be driven high to run. To give the Montium a software reset, data must be sent to address 53248(unsigned). The data in this MMIO is shown in Table A.3. When the counter bits are set to 0 the RESETN is driven high again after 1 clock cycle. Due to synchronization issues it may be necessary to hold the RESETN low for more than 1 clock cycle.

[1]In the latest version of the Montium the register view mapping is changed: the normal view is mapped vertically instead of horizontally

	dma_sel=1		
	dma_mr=0	dma_mr=1	
	dma_rs=-	dma_rs=0	dma_rs=1
bus_en(1)	PP1.MemLeft	PP1.RFA	PP1.RFB
bus_en(2)	PP1.MemRight	PP1.RFC	PP1.RFD
bus_en(3)	PP2.MemLeft	PP2.RFA	PP2.RFB
bus_en(4)	PP2.MemRight	PP2.RFC	PP2.RFD
bus_en(5)	PP3.MemLeft	PP3.RFA	PP3.RFB
bus_en(6)	PP3.MemRight	PP3.RFC	PP3.RFD
bus_en(7)	PP4.MemLeft	PP4.RFA	PP4.RFB
bus_en(8)	PP4.MemRight	PP4.RFC	PP4.RFD
bus_en(9)	PP5.MemLeft	PP5.RFA	PP5.RFB
bus_en(10)	PP5.MemRight	PP5.RFC	PP5.RFD

Table A.2: Addressing signals for selection of memories and register files

Bit	Outside connection	Description
0	Wait_TP	Stall the Montium processor
1	Resetn	!Reset
2	GPI(0)	General purpose input
3	GPI(1)	General purpose input
4	GPI(2)	General purpose input
5	GPI(3)	General purpose input
6	GPI(4)	General purpose input
7	GPI(5)	General purpose input
8	-	Count bit 0
9	-	Count bit 1
10	-	Count bit 2
11	-	Count bit 3
12	-	Count bit 4
13	-	Count bit 5
14	-	Count bit 6
15	DV	Data Valid

Table A.3: Sequencer data bits

Bit	Outside connection	Description
0	DMA_addr(0)	DMA address bit 0
1	DMA_addr(1)	DMA address bit 1
2	DMA_addr(2)	DMA address bit 2
3	DMA_addr(3)	DMA address bit 3
4	DMA_addr(4)	DMA address bit 4
5	DMA_addr(5)	DMA address bit 5
6	DMA_addr(6)	DMA address bit 6
7	DMA_addr(7)	DMA address bit 7
8	DMA_addr(8)	DMA address bit 8
9	DMA_addr(9)	DMA address bit 9
10	DMA_sel	DMA enable
11	DMA_rw	DMA read/write
12	DMA_mr	DMA memory/register
13	DMA_rs	DMA register select
14	-	Reserved
15	-	Reserved

Table A.4: DMA address 1 on 0xC800 interface connections

The RESETN can be held low by an additional number of cycles as written in bits 8 to 14. The extra delay is programmable in the range 0-126 cycles delay. When all count bits are driven high, the RESETN will not be driven high until an extra MMIO access is performed to drive the RESETN high.

A.4 Direct Memory Access

The DMA interfaces have four configuration addresses: 0xC800 (51200 unsigned), 0xC801 (51201 unsigned), 0xD800 (55296 unsigned) and 0xE000 (57344 unsigned). The meaning of the bits at those addresses are shown in Table A.4, Table A.5, Table A.6 and Table A.7.

Bit	Outside connection	Description
0	bus_en(10)	Enable bus 10
1	bus_en(9)	Enable bus 9
2	bus_en(8)	Enable bus 8
3	bus_en(7)	Enable bus 7
4	bus_en(6)	Enable bus 6
5	bus_en(5)	Enable bus 5
6	bus_en(4)	Enable bus 4
7	bus_en(3)	Enable bus 3
8	bus_en(2)	Enable bus 2
9	bus_en(1)	Enable bus 1
10	gb2lane(0)	gb2lane bit 0
11	gb2lane(1)	gb2lane bit 1
12	gb2lane(2)	gb2lane bit 2
13	gb2lane(3)	gb2lane bit 3
14	gb2lane(4)	gb2lane bit 4
15	-	Reserved

Table A.5: DMA address 2 on 0xC801 interface connections

Bit	Outside connection	Description
0	DMA_data(0)	DMA data bit 0
1	DMA_data(1)	DMA data bit 1
2	DMA_data(2)	DMA data bit 2
3	DMA_data(3)	DMA data bit 3
4	DMA_data(4)	DMA data bit 4
5	DMA_data(5)	DMA data bit 5
6	DMA_data(6)	DMA data bit 6
7	DMA_data(7)	DMA data bit 7
8	DMA_data(8)	DMA data bit 8
9	DMA_data(9)	DMA data bit 9
10	DMA_data(10)	DMA data bit 10
11	DMA_data(11)	DMA data bit 11
12	DMA_data(12)	DMA data bit 12
13	DMA_data(13)	DMA data bit 13
14	DMA_data(14)	DMA data bit 14
15	DMA_data(15)	DMA data bit 15

Table A.6: DMA address 3 on 0xD800 interface connections

Bit	Outside connection	Description
0	-	Reserved
1	-	Reserved
2	-	Reserved
3	-	Reserved
4	-	Reserved
5	-	Reserved
6	-	Reserved
7	-	Reserved
8	-	Reserved
9	-	Reserved
10	-	Reserved
11	-	Reserved
12	DMA_lane2gb(0)	lane2gb bit 0
13	DMA_lane2gb(1)	lane2gb bit 1
14	DMA_lane2gb(2)	lane2gb bit 2
15	DMA_lane2gb(3)	lane2gb bit 3

Table A.7: DMA address 4 on 0xE000 interface connections

Appendix B

Memory map

In the Starburst SoC generator the address range 0x60000000 to 0x7FFEFFFF is already mapped to the NoC. Within this project Montium 1 was used with base address 0x70000000 and Montium 2 had 0x70010000 as base address. It is not advisable to change these addresses, because these addresses are hardcoded in the DTL-adapter and in some configuration scripts which convert the output from the Montium compiler to data usable on this platform.

Configuration data for the Montium, as well as configuration data for the CCU, must be sent to the base address of the corresponding Montium with the 16 most significant bits representing the configuration address within the Montium and the 16 least significant bits representing the data.

Address bits 15 downto 12 selects the local memory within a Montium TP as described in Table B.1.

For every entry in the local memory of the Montium TP, 4 Bytes are

Address 0x700 - A- - -	Montium local memory
0	PP1.MemLeft
1	PP1.MemRight
2	PP2.MemLeft
3	PP2.MemRight
4	PP3.MemLeft
5	PP3.MemRight
6	PP4.MemLeft
7	PP4.MemRight
8	PP5.MemLeft
9	PP5.MemRight
A	Register Files
B	-
C	-
D	-
E	-
F	-

Table B.1: Montium memories mapped into MicroBlaze memory map

Address 0x700 - - AAA	Montium local memory
000	Memory entry 0(except for PP1.MemLeft)
001	Memory entry 0
002	Memory entry 0
003	Memory entry 0
004	Memory entry 1
005	Memory entry 1
006	Memory entry 1
007	Memory entry 1
008	Memory entry 2
009	Memory entry 2
00A	Memory entry 2
00B	Memory entry 2
00C	Memory entry 3
00D	Memory entry 3
00E	Memory entry 3
00F	Memory entry 3
etc.	etc.
FFC	Memory entry 1023

Table B.2: Local memory mapping strategy

mapped to this memory, except for the first memory. To the first entry in PP1.MemLeft only 3 Bytes are mapped, because this address is the configuration address of the Montium TP and the CCU. This mapping strategy is displayed in Table B.2.

Within a register file are 4 memory locations. Those memory locations are selected using the 2 least significant bits of an address.

Address bits 11 downto 7	Montium register file
00000	PP1.RFA
00001	PP1.RFB
00010	PP1.RFC
00011	PP1.RFD
00100	PP2.RFA
00101	PP2.RFB
00110	PP2.RFC
00111	PP2.RFD
01000	PP3.RFA
01001	PP3.RFB
01010	PP3.RFC
01011	PP3.RFD
01100	PP4.RFA
01101	PP4.RFB
01110	PP4.RFC
01111	PP4.RFD
10000	PP5.RFA
10001	PP5.RFB
10010	PP5.RFC
10011	PP5.RFD

Table B.3: Register file memory map

Appendix C

Source code test application

Source code Montium 1

```
start: clock
agu p1m1 p1m2 p2m1 p2m2 p3m1 p3m2 p4m1 p4m2 p5m1 p5m2=0 |=0
llc lc4 99
clock
jnc gpi1 start //wait until CCU gives ready signal
communicate: clock
mov data p1m1 -> ext1
agu p1m1 ++
clock
agu p1m1 ++
mov data p1m1 -> ext2
clock
agu p1m1 ++
mov data p1m1 -> ext3
clock
agu p1m1 ++
mov data p1m1 -> ext4
clock
agu p1m1 ++
mov data p1m1 -> ext1
mov data p1m1 -> ext2
clock
agu p1m1 ++
mov data p1m1 -> ext1
mov data p1m1 -> ext3
clock
agu p1m1 ++
mov data p1m1 -> ext1
mov data p1m1 -> ext4
loop lc4 communicate
clock
nop
clock
frz
```

Source code Montium 2

```
start: clock
agu p1m1 p1m2 p2m1 p2m2 p3m1 p3m2 p4m1 p4m2 p5m1 p5m2=0 |=0
jnc gpi1 start //wait until CCU gives ready signal
clock
mov ext1 -> p1m2
communicate: clock
agu p1m2 ++
mov ext2 -> p1m2
clock
agu p1m2 ++
mov ext3 -> p1m2
clock
agu p1m2 ++
mov ext4 -> p1m2
clock
agu p1m2 ++
mov ext1 -> p1m2
clock
agu p1m2 ++
mov ext1 -> p1m2
mov ext2 -> p2m1
clock
agu p1m2 ++
agu p2m1 ++
mov ext1 -> p1m2
mov ext3 -> p2m1
clock
agu p1m2 ++
agu p2m1 ++
mov ext1 -> p1m2
mov ext4 -> p2m1
jmp communicate
```

Bibliography

[1] Mentor graphics corporation, questa sv/afv user's manual, software version 6.5c.

[2] International technology roadmap for semiconductors 2007 edition. `http://www.itrs.net/links/2007itrs/execsum2007.pdf`, 2007. page 21.

[3] Andreas Hansson. *A composable and Predictable On-Chip Interconnect.* PhD thesis, Technische Universiteit Eindhoven, 2009.

[4] P. M. Heysters. *Coarse-Grained Reconfigurable Processors - Flexibility meets Efficiency.* PhD thesis, Univ. of Twente, Enschede, September 2004.

[5] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen. A virtual channel router for on-chip networks. In *Proc. IEEE International SOC Conference*, pages 289–293, September 12–15, 2004.

[6] R. Mucci. A comparison of efficient beamforming algorithms. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 32(3):548 – 558, jun. 1984.

[7] P.M.Heysters. Montium tile processor design specification. Strictly confidential, June 2005.

[8] G. J. M. Smit, A. B. J. Kokkeler, P. T. Wolkotte, P. K. F. Hölzenspies, M. D. van de Burgwal, and P. M. Heysters. The chameleon architecture for streaming dsp applications. *EURASIP Journal on Embedded Systems*, 2007:78082, 2007.

[9] C.H. van Berkel. Multi-core for mobile phones. In *invited paper DATE conference*, 2009.

[10] M. D. van de Burgwal. Serving the montium: Design of an energy-efficient processor-network interface. Master's thesis, University of Twente, April 2005.

[11] M. D. van de Burgwal, G. J. M. Smit, G. K. Rauwerda, and P. M. Heysters. Hydra: an energy-efficient and reconfigurable network interface. In *Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms, Las Vegas, USA*, pages 171–177, Las Vegas, USA, June 2006. CSREA Press.

[12] M.D. van de Burgwal. Hydra design specification - multiplexer version. Technical report, 2005.

[13] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, page 155a, April 04–08, 2005.

[14] Xilinx. Embedded system tools reference guide, September 2009. EDK 11.3.1.

[15] Tei-Wei Kuo Yi-Hung Wei, Chuan-Yue Yang and Shih-Hao Hung. Energy-efficient real-time scheduling of multimedia tasks on multi-core processors. *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 258–262, 2010. ISBN:978-1-60558-639-7.