On Time, in Style: Nanosecond Accuracy in Network Control Systems

Kreider, M.

PhD August, 2017

Mathias Kreider
Diplomingenieur (FH) Elektrotechnik

On Time, in Style: Nanosecond Accuracy in Network Control Systems

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

DIRECTOR OF STUDIES:	SUPERVISORS:
Prof. Vic Grout	Prof. John Davies
	Dr. Ralph Bär

Research was undertaken under the auspices of
Glyndŵr University and GSI Helmholtzzentrum für
Schwerionenforschung and was submitted in partial fulfilment for
the award of a Degree of the University of Wales

August, 2017

Thesis Declaration

I hereby declare that this work has not been accepted in substance for any degree and is not currently being submitted in candidature for any degree.

Date: 03/08/2017

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by references. A bibliography is appended.

Date: 03/08/2017

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Date: 03/08/2017

The research was completed under the guidance of:
Prof. Vic Grout (Director of Studies), Glyndŵr University.

Date: 03/08/2017

I didn't go to university. Didn't even finish A-levels.
But I have sympathy for those who did.

– Terry Pratchett

GLYNDŴR UNIVERSITY

Abstract

Department of Computing

Doctor of Philosophy

On Time, in Style:
Nanosecond Accuracy
in Network Control Systems

by Mathias Kreider

The aim of this thesis is to prove it is possible to control and coordinate machines at sub-nanosecond accuracy. The demonstration case was the conceptualisation of a control system for large physics experiments, employing high precision timing technology and deterministic communication.

The results were empirically proven by means of a control system for a next generation particle accelerator, the Facility for Antiproton and Ion Research FAIR in Darmstadt, Germany. In the scope of this thesis, concepts and prototypes for both the control system's master unit, the *Data Master*, and the employed low latency network protocol, *Etherbone* were developed. In addition, a formal model of the control system was created. Its purpose is to guarantee deterministic operation by means of an offline analysis ahead of time, assaying latency bounds for a given set of commands.

This research shows that an alarm-based control system with a timing resolution of 1 ns is feasible. A scheme which allows machine schedules to be steered at runtime is presented and has been validated. Timely arrival can be guaranteed for trees of alternative command sequences, known ahead of time, as well as for limited changes during runtime. The proposed approach scales to several thousand controlled machines. It supports fully deterministic, parallel control processes that can utilise the full accuracy of the underlying time distribution system. The rate of control messages is only dependent on available network bandwidth.

The results of this work are already wholly in use at FAIR, controlling a real world particle accelerator. Etherbone components are also already in productive use at the European Centre for Nuclear Research CERN and other institutions, both for particle accelerator control systems and communication applications.

Acknowledgements

Many thanks go to my supervisors, Vic Grout, for being ready to give good advice at all hours and making even the most tricky administrative trouble magically disappear, John Davies, for patiently editing out all my germanisms and always being a great host, and Ralph Bär, for giving me the chance to study beside my normal day job and especially for volunteering to become my supervisor when time was of the essence. My project advisor Dietrich Beck also has my sincere gratitude for his preparedness to read even my most cryptic drafts and discuss topics so far out of the box. I cannot thank you all enough for your continuous support and engagement.

My deepest thanks go to my family, especially my parents, Bärbel and Klaus, for their support and believing in me over the years it took me to write this thesis. The same goes for my future husband Markus, who was, despite being neither an engineer nor researcher, patiently listening to my ramblings about technology and maths he had never heard of before. You have my heartfelt respect for patiently working your way through all that, having nothing to gain from it than understanding my troubles and feeding your curiosity. Deep thanks also go to my brother Michael and my grandmother Ruth for proof reading and always listening to my nerdy ideas. Lastly I'd like to express my gratitude to my late grandfathers, Ludwig and Heinz, for trying to spark my interest in technology when I was but a child. It worked.

Many thanks go to my friends Daniel Fischer, whom I both hated and loved for giving the most hard-hitting feedback of them all, and Sabine Benda for proofreading. I also like to thank my colleagues from GSI for their ongoing support in creating this work. Thanks go to Stefan Rauch, for his readiness to talk ideas and being a universal trouble shooter, Marcus Zweig, for taming the White Rabbit and his dark sarcasm, Alexander Hahn, for his killer instinct and constructive feedback when testing the Data Master, Jiaoni Bai, for the hard numbers her dissertation provided and generally being a much nicer person than I could ever be, and finally Anjan Suresh, for implementing the continuous integration system which saved me such a lot of nerves.

Very special thanks go to my former colleague Wesley Terpstra, for his unconventional ideas, high standards, the enjoyable teamwork on elegant solutions and his both encouraging and brutally honest feedback. I'd also like to thank all the other colleagues from CERN and GSI who contributed to this work and their collaboration over all these years. My thanks also go to Steffen Bondorf and his colleagues at University of Kaiserslautern for the DiscoDNC Simulator and the associated productive exchange.

And finally, special thanks go to Ingo Stengel, the man who started this all by casually asking me at a Christmas party, more than six years ago, if I ever considered doing my PhD…

Contents

List of Figures

List of Tables

List of Abbreviations

ACM	**A**ssociation for **C**omputing **M**achinery
ADC	**A**nalogue-to-**D**igital **C**onverter
ADEV	**A**llan **D**eviation
AES	**A**dvanced **E**ncryption **S**tandard
ARP	**A**ddress **R**esolution **P**rotocol
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
B2B	**B**unch to **B**ucket transfer
BPC	**B**eam **P**roduction **C**hain
BP	**B**eam **P**rocess
CB	**C**ross**B**ar
CERN	European Centre for Nuclear Research
CMD-Q	**C**om**m**an**d** **Q**ueue
CNF	**C**onjunctive **N**ormal **F**orm
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
CPLD	**C**omplex **P**rogrammable **L**ogic **D**evice
CPU	**C**entral **P**rocessing **U**nit
CRC	**C**yclic **R**edundancy **C**heck
CS	**C**ontrol **S**ystem
DAC	**D**igital-to-**A**nalogue **C**onverter
DC	**D**irect **C**urrent
DDS	**D**irect **D**igital **S**ynthesis
DESY	**D**eutsches **E**lektronen **Sy**nchrotron (German Electron Synchrotron)
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DM	**D**ata **M**aster
DNF	**D**isjunctive **N**ormal **F**orm
DSP	**D**igital **S**ignal **P**rocessor
EB	**E**ther**b**one
EBM	**E**ther**b**one **M**aster
EBS	**E**ther**b**one **S**lave
ECA	**E**vent **C**ondition **A**ction unit
EDF	**E**arliest **D**eadline **F**irst
EPICS	**E**xperimental **P**hysics and **I**ndustrial **C**ontrol **S**ystem
Eth	IEEE 802.3 **Eth**ernet
FAIR	**F**acility for **A**ntiproton and **I**on **R**esearch
FEC	**F**orward **E**rror **C**orrection
FH	University of Applied Sciences
FIFO	**F**irst **I**n, **F**irst **O**ut
FOI	**F**low **O**f **I**nterest
FPGA	**F**ield **P**rogrammable **G**ate **A**rray

FSM	**F**inite **S**tate **M**achine
GbE	**G**iga**b**it **E**thernet
GMT	**G**reenwich **M**ean **T**ime
GPS	**G**lobal **P**ositioning **S**ystem
GSI	GSI-Helmholtz Centre for Heavy Ion Research
HDL	**H**ardware **D**escription **L**anguage
HP	**H**igh **P**riority
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IO	**I**nput/**O**utput
IP	**I**nternet **P**rotocol Version 4
IRIG	**I**nter **R**ange **I**nstrumentation **G**roup
ITU	**I**nternational **T**elecommunication **U**nion
JTAG	**J**oint **T**est **A**ction **G**roup
LHC	**L**arge **H**adron **C**ollider
LINAC	**Lin**ear **Ac**celerator
LM32	**L**attice **M**ico 32 Processor
lo	**l**eft **o**ver service
LP	**L**ow **P**riority
LVDS	**L**ow **V**oltage **D**ifferential **S**ignalling
MCU	**M**icro**c**ontroller **U**nit
MSI	**M**essage **S**ignalled **I**nterrupt
MTS	**M**inimal **T**est **S**ystem
MUX	**Mu**ltiple**x**er
NCO	**N**umerically **C**ontrolled **O**scillator
NC	**N**etwork **C**alculus
NIC	**N**etwork **I**ntegrated **C**ontroller
NTP	**N**etwork **T**iming **P**rotocol
OSI	**O**pen **S**ystems **I**nterconnection Model
PAL	**P**rogrammable **A**rray **L**ogic
PBOO	**P**ay **B**urst **O**nly **O**nce
PCIe	**P**eripheral **C**omponent **I**nterconnect **e**xpress
PGS	**P**acketised **G**reedy **S**hapers
PHY	**Phy**sical interface
PLA	**P**rogrammable **L**ogic **A**rray
PLC	**P**rogrammable **L**ogic **C**ontroller
PLL	**P**hase **L**ocked **L**oop
PMOO	**P**ay **M**ultiplexing **O**nly **O**nce (Analysis)
PPS	**P**ulse **P**er **S**econd
PQ	**P**riority **Q**ueue
PSCED	**P**acketised **Sc**heduler, **E**arliest **D**eadline First
PTP	**P**recision **T**ime **P**rotocol
QoS	**Q**uality **o**f **S**ervice
RDMA	**R**emote **D**irect **M**emory **A**ccess
RF	**R**adio **F**requency
RR	**R**ound **R**obin

RTC	**R**eal**t**ime **C**lock
RTOS	**R**eal**t**ime **O**perating **S**ystem
RTS	**R**eal**t**ime **S**ystem
RX	**R**eceiver
SCPU	**S**oft **C**entral **P**rocessing **U**nit
SDB	**S**elf **D**escribing **B**us
SDH	**S**ynchronous **D**igital **H**ierarchy
SERDES	**SER**ialiser / **DES**erialiser
SFA	**S**eparate **F**low **A**nalysis
SI	International System of Units
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
SoC	**S**ystem **o**n a **C**hip
SONET	**S**ynchronous **O**ptical **Net**working
SSL	**S**ecure **S**ocket **L**ayer
ST	**S**tandard **T**ime
SyncE	**Sync**hronous **E**thernet
TAI	**I**nternational **A**tomic **T**ime
TCP	**T**ransmission **C**ontrol **P**rotocol
TDC	**T**ime to **D**igital **C**onverter
TFA	**T**otal **F**low **A**nalysis
TLU	**T**imestamp **L**atch **U**nit
TS	**T**ime**s**tamp
TTF	**T**iming **T**est **F**acility
TX	**T**ransceiver
UDP	**U**ser **D**atagram **P**rotocol
USB	**U**niversal **S**erial **B**us
UTC	**C**oordinated **U**niversal **T**ime
UT	**U**niversal **T**ime
VAT	**V**irtual **A**ddress **T**able
VC(X)O	**V**oltage **C**ontrolled (Crystal) **O**scillator
VME	**V**ersa **M**odule **E**urocard-bus
WB	**W**ish**b**one
WBM	**W**ish**b**one **S**lave
WBS	**W**ish**b**one **M**aster
WR	**W**hite **R**abbit

List of Symbols

R	resistance	Ω (V/A)
C	capacitance	F (V/C)
L	inductance	H (V s/A)
ω	angular frequency	rad

To Markus

Part I

Context and Background

Chapter 1

Introduction

This chapter aims to give a general overview of this thesis. The context and motivation for this work are explained, followed by a detailed breakdown of research methodology. Moreover, the contribution to knowledge and its impact are presented. The structure of the thesis is then outlined, complemented by short summaries of all chapters.

1.1 Motivation

The motivation for this thesis sprang from the need of a new type of control system (CS) for large physics experiments, which can utilise high accuracy timing. It was researched and developed in the context of a case study for the control of particle accelerator facilities.

"White Rabbit" (WR), the underlying time synchronisation technology, was an initiative started in 2008 by the European Centre for Nuclear Research (CERN) and initially aimed at the modernisation of the CS of the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland. At the time, the GSI-Helmholtz Centre for Heavy Ion Research (GSI), in Darmstadt, Germany, took an interest in evaluating suitable technologies for a CS modernisation and a new CS for the upcoming Facility for Antiproton and Ion Research (FAIR), which will become a major extension to the GSI accelerator facilities.

Research and development of WR successively became a close collaboration between CERN and GSI/FAIR. The case study in this thesis is based on a CS design suitable for the FAIR accelerators and GSI facilities. The joint FAIR research complex is planned to go into operation in 2018.

1.2 Research Methodology

This thesis is dedicated to the problem of achieving deterministic control of large realtime systems under the assumption of a common, highly accurate notion of absolute time. The use of WR as the underlying time distribution system is axiomatic, as it is based on a management decision prior to the start of this work. An evaluation in greater depth within the scope of this thesis (see subsection 2.7.1) could confirm that the properties of WR are sufficient for such a concept. The working hypothesis is that a such a CS can be conceived as a deterministic, alarm based system, capable of utilising the full accuracy of the WR time distribution scheme to synchronously generate or distribute signals and assign timestamps to inputs.

Choice of Methodology A hypothesis can either be tested by experiment or by theoretical proof, i.e. trying to, by logical deduction, either confirm all of its aspects or refute one of them. For the research in this thesis, an experimental approach was chosen. The reason for this decision was that the proposed hypothetical system is complex, it consists of a combination of a large number of physical and logical problems to overcome. To obtain a rigorous result in a meaningful timeframe, the analysis must be built onto known theoretical models. This means the problems of the hypothetical system must be separable and transferable to known problems for which solutions have been found in order to prove or refute partial aspects. The advantage of a theoretical approach is that if the system can be modelled, the complete behaviour will be calculable. On the downside, since neither the separability nor the existence of known analogue problems is guaranteed, it is very possible to reach an impasse in research within the given time frame.

An experimental approach, on the other hand, allows black-box tests of complex systems to be conducted quickly, provided they can be built. Separation of the experiment into partial systems is also possible, provided all inputs and outputs can be observed. Since GSI could provide prototype implementations of the underlying WR time distribution infrastructure from the very beginning, it was likely that an adequate experiment setup to either prove or refute the hypothesis could be built. This original prototype hardware and all that were devised within the scope of this thesis feature industry standard interfaces, meaning all of the system's inputs and outputs could be accurately monitored using proven and well documented technology, such as high end oscilloscopes and network analysers.

Objectives The purpose of the primary research is the provision of a concept for a highly accurate CS, scalable to thousands of machines and distances of several kilometres. In conjunction with the case study (see subsection 4.6), three main objectives were defined:

- Scale in number of devices
- Scale in distance
- Accuracy of timing

The first objective requires evidence and/or theoretical justification that the system will be able to support the required number of devices and in consequence provide the required bandwidth, while the second objective requires evidence that the system is able to cover the required geographic distance. The third objective is the most complex, as it requires for the requested synchronicity, peak response time (latency) and time resolution.

Research Objects The primary research covers three distinct objects. The first is the master CS unit, which was named "Data Master" (DM) (chapter 7, 8). It has to generate and deliver control messages to all endpoints in order to control machines. The second object is a concept for deterministic communication between WR platforms, which has been named "Etherbone" network protocol (EB) (chapter 6).

The definition of the third research object emanated from the necessity to verify the others. In order to show that deterministic control can be guaranteed in general and not only for a specific test case, bounds for its application were required. The third object therefore became the conception of a simulation model to provide bounds within which the CS's operation is guaranteed to be deterministic (chapter 9). The model must be usable at run time, although it does not need to satisfy hard realtime constraints.

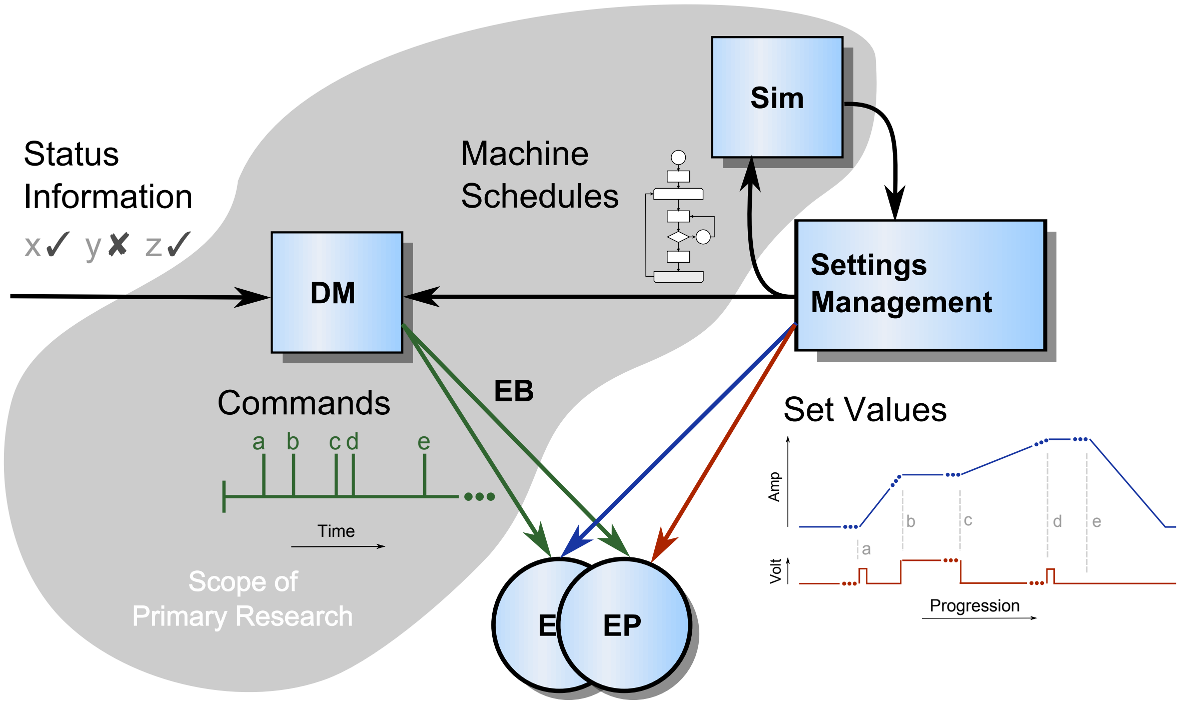


FIGURE 1.1: Overview of realtime CS and Settings Management. Endpoints (EP) receive both commands and set values. Scope of Primary Research is shown in grey.

Scope The term "control system" will be used throughout this thesis in the context of deterministic, hard realtime control. The hypothesis assumes an approach as an alarm based system, which separates commands and set values. Figure 1.1 shows the distinction. The CS instructs machines *what* to do *when* during runtime (green timeline). The corresponding set values, i.e. *how* the machine has to act, are distributed beforehand in a relaxed timeframe by the settings management system. The scope of primary research is indicated by the grey shaded area and the relation of the research objectives, DM, EB and simulation model, to settings management and endpoints (EP) is shown. Set values are data sets that describe the desired behaviour of specific EPs. In the context of a particle accelerator, this could, for example be the output current of a power supply (blue curve) or a voltage applied to an electrode (red curve).

1.2.1 Research

Literature Review The origin of the literature review was centred around publications associated with WR. Most of the related work used in this thesis was obtained from online libraries, such as the IEEE Xplore digital library [1], the ACM digital library [2] and Springer Online [3]. Searches for papers, conference proceedings and journal articles were conducted by search engines such as Google Scholar [4] and CiteSeerX [5]. Most of the employed textbooks were obtained from GSI's on site library or interlibrary loans, the remaining books were perused online by means of the Google Books service [6].

The majority of unpublished sources was GSI/FAIR internal material, such as presentations, technical notes and specification documents for FAIR, which provided information on future machine requirements, compatibility to legacy equipment and management decisions. Yet more background information was acquired from private communications with accelerator physicists and operating engineers at GSI/FAIR. The remaining unpublished sources were comprised of documents in the Open Hardware Repository (OHWR), a website created by CERN to coordinate development and host source code and documentation for WR related projects and other programming or hardware related websites.

Detailed specifications Constraints and use-cases for the approach were obtained from secondary research into existing CSs, available field bus technology, planned experiments at FAIR and the architecture and properties of WR (chapter 4). The related sources stemmed from publications on WR and CSs of particle accelerators, from unpublished FAIR documents and private communication.

Technology The requirements gathered for the detailed specifications were used as the foundation for a survey of suitable technologies for development of a prototype system. The results did not indicate conflicts with the technology present in the underlying WR system. The corresponding research mainly relied on tertiary sources, such as data sheets, application notes and manuals (chapter 5) and secondary accounts from hardware engineers at GSI about their experiences with current technology in hard realtime scenarios.

Existing systems at other sites Existing CSs at accelerators such as GSI, CERN, Diamond and DESY were evaluated in the course of the case study (see chapter 4.2.2). The secondary research into this area indicated that nearly all sites feature standard PCs and servers, which are not realtime capable. They also all use VME and industrial PCs running realtime Kernels, as well as programmable logic controllers (PLC). Both are able to achieve determinism down to microseconds. Finally, all the sites also have an optical clock distribution system, accurate to low picosecond or even femtosecond range. The only exception was CERN, which also provided UTC time accurate to nanoseconds. None of the investigated cases provided a generic control approach accurate to sub-nanosecond level. This very niche is the intended target of the new CS approach.

Future Plans at FAIR In 2018, not all accelerators of the FAIR complex will be completed and in operation. The new CS approach based on the research in this thesis and other research work at GSI is planned to be used for a retro-fitting of GSI's synchrotron. This means a replacement of the old GSI technology and running its main synchrotron, SIS18 with the new CS. This proposition is building on experience already gathered from controlling the smaller CRYRING accelerator. The CS approach will be expanded, tested and adjusted as necessary to control the whole FAIR facility in 2025.

1.2.2 Development

Prototype Design Based on the defined objectives, plans were specified to enable development of prototype systems of the DM and the EB protocol. Development occurred in both cases over several iterations, each intermitted by a phase of tests, presentation to the WR developer community and, if necessary, adjustment of the approach according to feedback and results.

Test Design and Equipment To test the hypothesis, hardware proving grounds for the prototypes were created. Variables were isolated as far as feasible by employing multiple different testbeds at different levels of integration. The lowest level were individual tests of sub-components, whose performance in terms of reliability, determinism, latency, and bandwidth was directly measured. The first step were unit tests by simulation, using randomised test data. These were followed up by integration tests on prototype hardware, verifying the latency, determinism and bandwidth predicted by the simulation. This was achieved by the application of on-chip logic analysers, profiling code, network analysers and oscilloscopes. The same set of tools was used to measure all quantitative targets imposed by the case study, allowing an accuracy between 8 ns for logic analysers and profiling code, 1 ns for timestamping by verified prototypes and below 20 ps for the oscilloscopes.

To conduct tests on higher abstraction levels, complete CS setups were assembled, consisting of a DM, WR network switches and up to several dozen CS endpoints. These scenarios were devised to run both short and long term reliability tests with randomised test data. To ensure the influence of individual platforms did not dominate the results, different constellations were created.

Test The tests cover, among others, scenarios such as 1 on 1 connections, test of time distribution only, high traffic scenarios and regular CS operation. To ensure consistency, these testbeds are persistent. An example can be found in chapter 10, "Timing Test Facility", figure 10.1 and figure 10.9. Ongoing development was verified on a daily basis by a continuous integration system, running standardised tests to ensure new developments did not negatively influence system performance.

Replicability All source code used in the FAIR CS design is open source and can be freely downloaded to replicate test cases described in this thesis. So are most hardware designs for WR infrastructure, easily allowing the replication of these experiments by others. Unfortunately, legal issues prevented public release of the DM hardware specifications and several other GSI form factors to this day. It is yet unknown if and when a public release will occur. However, it is possible for interested parties to develop a suitable platform for the DM based on the freely available open hardware form factor designs [7].

Analysis and Conclusion Because the high level testbeds were run for long times without interruption (several weeks) and for more than a year in total, sending and receiving billions of control messages, a high degree of confidence in system performance could be established. This was further enhanced by the successful control of a production system in the form of a real accelerator since mid 2015 by a FAIR CS prototype, including the DM and EB (see chapter 10).

1.3 Contribution to Knowledge

There are a large number of existing CS designs available today, some of them established and well proven. The work in this thesis aims to improve timing accuracy and resolution in CSs by several orders of magnitude, while simultaneously introducing a new level of determinism, flexibility and scalability. This sets it apart from all other architectures published today.

In this thesis, it is shown, theoretically, and by experimental validation, that an alarm-based CS with a timing resolution of 1 ns is feasible. It is further theoretically shown that the proposed approach does scale to several thousand controlled machines and is validated for more than a hundred endpoints. In addition, it is shown that the proposed CS supports fully deterministic, parallel control processes, synchronised at the full accuracy of the system. Moreover, a control scheme that allows machine schedules to be steered at runtime is presented and has been validated. It is also shown that control loop speed is only limited by external input and control message rate is only limited by network bandwidth. It is further shown theoretically that timely arrival can be guaranteed for trees of alternative command sequences, known ahead of time, as well as for limited changes during runtime.

1.4 Generalisation and Impact

The results of this work are already in use at GSI/FAIR [8], CERN and other institutions. Apart from the field of particle accelerators, wired machine coordination over several kilometres at nanosecond accuracy can provide the greatest benefit to the telecommunication industry, where phase antennas benefit from high accuracy synchronisation. More details and examples of projects already employing WR and EB based timing and CSs can be found in chapter 11.3.3 on page 208. There are more than 20 institutions employing WR (usually with EB) and a complete list can be found at the OHWR Website [9].

1.4.1 Thesis Structure

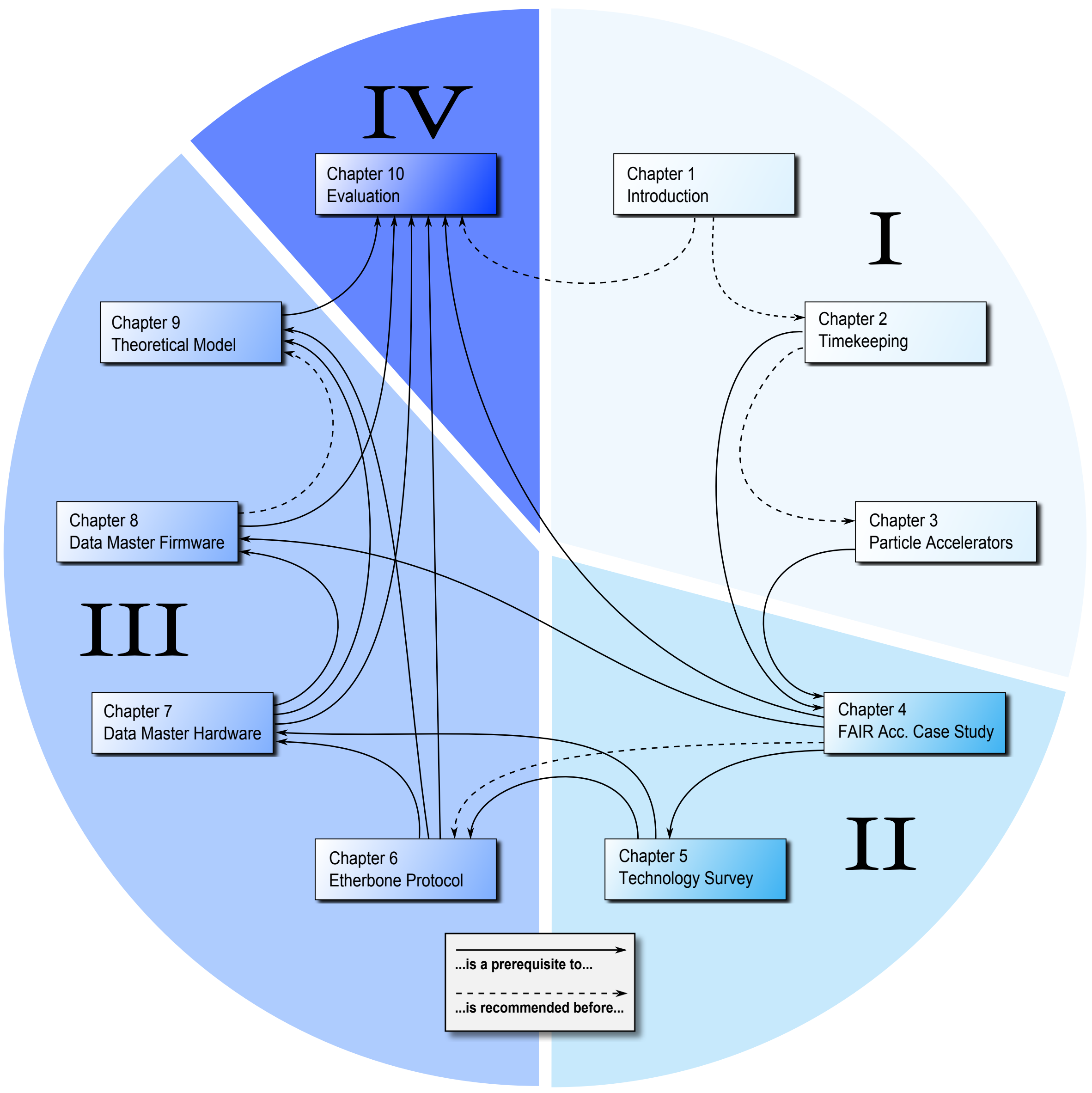

IV
Chapter 10 Evaluation
Chapter 1 Introduction
I
Chapter 2 Timekeeping
Chapter 9 Theoretical Model
Chapter 8 Data Master Firmware
Chapter 3 Particle Accelerators
III
Chapter 7 Data Master Hardware
Chapter 4 FAIR Acc. Case Study
Chapter 6 Etherbone Protocol
Chapter 5 Technology Survey
II
...is a prerequisite to...
...is recommended before...

FIGURE 1.2: Thesis Structure
Coloured Slices indicate Parts,
Arrows mark Dependencies between Chapters

As shown in figure 1.2, the thesis is grouped into four main parts, indicated by the coloured slices. The dependencies between individual chapters are marked with solid lines, starting at the chapter that is the prerequisite and ending with an arrowhead at the chapter depending on the content. Content which is not strictly necessary, but considered helpful to understanding, is connected with dashed lines. The four parts are:

I – Context and Background Apart from this general introduction, the first technical part is an introduction to the fields of timekeeping and particle accelerator physics (chapter 2 and 3). Both are required to understand the problems discussed in the case study. The former provides the necessary background for high precision timing systems, while the latter covers the basic theory of particle accelerators and their CS requirements. They form the introductory material for understanding the problems discussed in the case study.

II – Problem Analysis The FAIR accelerator case study is presented and the problems formulated (chapter 4). This case study analyses existing accelerator CSs, given requirements and planned experiments for FAIR and deduces the imposed requirements for a suitable CS. Building on this case study, a technology survey (chapter 5) follows, which contains a detailed discussion of suitable hardware and software platforms.

III – Approach and Implementation The third part contains different approaches, their discussion, topic specific literature review and solutions to partial problems. Chapter 6 covers deterministic data exchange and the development of the EB protocol. Chapter 7 and 8 contain the discussion on hardware and firmware of the main control unit, the DM. Finally, chapter 9 provides a theoretical model for the calculation of the CS's maximum end-to-end latency. This delivers the boundaries within which deterministic operation of the CS can be guaranteed.

IV – Conclusion Finally, all chapters from part III are evaluated in terms of their fitness as a solution to the problems presented in part II. This single chapter presents the methods used for verification as well as the results from experiments and simulation. It then draws a conclusion and indicates future work (chapter 10).

1.4.2 List of Publications

A complete list of publications containing contributions from my research that was undertaken for this thesis or already existing material that has influenced this thesis can be found in appendix C on page 219.

Chapter 2

Timekeeping

2.1 Overview

A standardised, common system of timekeeping is one of the most crucial agreements necessary for life in a modern civilisation. Over the years, several standards have been created to address this necessity. By now most of the requirements have been successfully met by ever more accurate, stable or more generic systems. There are at least 5 different timekeeping systems in use today (as of 2016), their design criteria spanning from solar-cycle oriented human life, simplicity and stability, astronomic scenarios and even relativistic time. The first part of this section will cover the basics of clocks and their possible accuracy, time systems and their application and finally their digital representations. The second part aims to give a basic understanding of oscillators and their synchronisation and will further introduce time distribution systems.

2.1.1 Terminology

When talking about time, clocks, oscillators and the synchronisation of systems, there are specialised terms which will be introduced here. Some of these terms also have widely used colloquial meanings and require clarification.

Accuracy and Precision While "accuracy" is colloquially synonymous with "precision", scientifically there is a strong distinction between the two. The common scientific definition is "accuracy" being the systematic and "precision" the random contribution to error, with error being the difference between the measured value and the expected value. The expected value is either a theoretical or a reference value. An extended terminology for accuracy and precision is given in the ISO 5725 standard [10]. According to the standard, "trueness" describes the systematic error, often expressed as a bias. "Precision" describes only the random error, this is usually expressed as standard deviation. "Accuracy" is defined as the resulting overall quality, the closeness between the observed value and the true or reference value. In the context of this thesis, the definition will adhere to the ISO 5725 standard .

Clock The term "clock" has two meanings in the present context. The common definition of a clock *"A mechanical or electrical device for measuring time . . . "*, whereas in computing, a clock is *"An electronic device used to initiate and synchronise internal* *operations"* [11]. In order to clarify this, the former will be called a "clock", the latter will be called an "oscillator" when referring to the physical device and a "clock signal" or "sample clock" in the context of device synchronisation in a computer.

Skew Skew is usually defined as the offset between the edge of a clock signal and its corresponding data. This deviation varies for different points in the same synchronous digital system due to propagation delay.

Jitter When talking about oscillators and especially clock signals, the terms "jitter" and "skew" are often used. There are actually multiple definitions of jitter which are commonly used [12]. The most important ones are:

- Period Jitter, the deviation of cycle length to the ideal period
- Cycle to Cycle Jitter, the deviation of cycle length to the adjacent cycle's length
- Long Term Jitter, the deviation of a clock signal from its ideal position over several cycles
- Phase Jitter, the integration of phase noise over a given spectral interval
- Time Interval Error, the deviation of the actual position of edge from the ideal position of said edge

In the context of timekeeping, cycle-to-cycle jitter is the most important property for short term stability. The Time Interval Error can be an important parameter for synchronous digital systems, as it determines the maximum amount of skew before data is sampled at the wrong clock cycle.

Synchronisation This term is ambiguous and can either refer to the more general process of achieving a common notion of time or the phase and frequency alignment of two metronome-like oscillators. In this context, the general process will be called "time synchronisation" and oscillator synchronisation will be called "clock synchronisation" or "syntonisation".

2.1.2 What it takes to measure time

A clock, or chronometer, is a measurement device, counting the occurrence of an event with a known length of time. A more general definition would be a chronometer consisting of:

(a) A time "reckoner", which is a repeatable phenomenon whose motion or change of state is observable and obeys a definite law

(b) A time reference, with respect to which the position or state of the time reckoner can be determined

These elements correspond to the two properties of time measurement: interval and epoch [13]. Or put briefly, a clock consists of an oscillator and a counter.

The accuracy of a clock is a complex interplay of a variety of factors. The resolution is determined by the frequency of the oscillator, because any measured time period cannot be smaller than the time between two events. When talking about clocks, there is a distinction between accuracy and stability [14]. The accuracy of a clock describes the closeness to a known time reference while stability only describes how reproducible its output is. Stability is usually described as a deviation from the mean, accuracy as closeness to the reference. The most common variant for expressing stability is the Allan deviation (ADEV).

2.2 Timekeeping History

The following is a short excursion on the history of timekeeping from the year 1700 onward. It covers the origin, the application, conversion between systems and the impact of legacies on modern timekeeping.

2.2.1 Navigation - the Driving Force for Clocks

A strong demand for accurate timekeeping came up in marine navigation. When navigating by celestial bodies like stars or planets, the observed angle between the horizon and such a body depends on time and the position on earth. The latter is given in polar coordinates, two angles and a distance: latitude (angle on the polar plane), longitude (angle on the rotational plane) and height. Barring accidents or freakish weather, ships are normally found at sea level, so height can safely be ignored in a marine context.

Earth's rotational axis is tilted about 23.5 degrees in relation to its orbital plane. When measuring from the same position on earth, the observed culmination angle of a star changes over the course of a year. Latitude can then be determined from angular tables, the time of year and the observation angle. For finding longitude, the time of day is required. Since earth does a full rotation once per day, the time of day at which the culmination of a celestial body can be observed strongly depends on the longitudinal position. In reverse, longitude can be determined from angular tables and knowing the current time at the location the tabular values were taken.

The Longitude Problem Finding longitude was an impossible navigation problem for centuries, because a very accurate time reference is needed. The following conversions shall illustrate the scale of the time influence for latitude and longitude calculation. For the sake of brevity in the following explanation, a simplified approach is used, ignoring the fact that earth is an ellipsoid.

As rough calculation for latitude based on almanac data [15] can quickly show that even without the time of day, just knowing the date, the maximum latitude error is $\approx 23\,\mathrm{km}$. However, earth rotates $360°$ in a day, making longitude calculation far more sensitive to time than latitude. Dividing the number of arc minutes per day by the number of seconds per day (eq. 2.1), we get an error per second of 0.25

arc minutes or $463\,\mathrm{m}$. As eq. 2.1 shows, this means a clock being off by $1\,\mathrm{min}$ already produces a longitude error greater than the maximum possible latitude time error. Hence, historically, longitudinal navigation errors were more prominent. The probably most famous example of such an error is the Scilly naval disaster of 1707, which had an even higher death toll than the sinking of the Titanic in 1912.

$$\frac{360 \cdot 60}{24 \cdot 3600}\frac{'}{\mathrm{s}} = \frac{1}{4}\frac{'}{\mathrm{s}} = 463\frac{\mathrm{m}}{\mathrm{s}} = 27.8\frac{\mathrm{km}}{\mathrm{min}} \tag{2.1}$$

At the time, it was possible to deduce the time of day from celestial bodies if the longitudinal position was known, or vice versa, but not both. While Galileo Galilei did discover an accurate astronomical method of getting time when observing Jupiter's moons [16], it required a very good telescope and steady ground to put it on, an impossibility on a ship. An accurate time reference suitable for a moving vessel was desperately needed, "accurate" meaning accurate to less than a minute. A similar astronomical method (Lunar-Distance-Method) to find the time in Greenwich, usable on a vessel, would eventually be discovered [17]. However, it was introduced only after the first high quality marine chronometers became available. The advantage lies in the minute cost, the maximum achievable accuracy lies between 1 and 2 minutes.

Marine Chronometers The available clocks, from sand clocks to early clockwork, had an appalling accuracy; useless for navigation. The British Royal navy was showing a strong interest in research of more accurate mechanical clocks. In 1714, the first Longitude Act was announced, promising a series of rewards for a solution to the longitude problem. The top prize of £20000 (approx. £2.6m today) was offered to anyone who could determine longitude with an error less than $0.5°$ (30 nautical miles or $55\,\mathrm{km}$). Today, it is agreed that the British clock maker John Harrison finally solved the problem. In 1775, his fourth marine chronometer prototype (H4) was finally officially tested and acknowledged to being $5\,\mathrm{s}$ adrift over a journey of 81 days [18]. However, the cost of the early chronometers were comparable to the building cost of a flagship.

2.2.2 From Springs to Crystals

The very first quartz clocks were built in 1927 by Marrison and Horton for Bell laboratories. Instead of the oscillation of a spring, a quartz clock uses the period of the electromagnetic resonance in a cut quartz crystal as the event to count (see 2.4.1 for more details). Good quartz watches today offer a stability in the order of $0.5 \times 10^{-7}\,\mathrm{s}$ and are accurate to about $0.1\,\mathrm{s}$ per day [19]. Interestingly, these offer about the same accuracy as Harrison's H4, but with a vast difference in effort and build cost. These are wrist watches though. In a static, controlled setup, much better results are possible with quartz crystals. If the crystal is kept at a constant temperature (crystal oven) and a few other optimisations are used, a stability of $1 \times 10^{-11}\,\mathrm{s}$ is possible [20], making such a clock accurate to about $1\,\mathrm{ms}$ per day.

2.2.3 Clocks and the Quantum Tick

In 1946, the first atomic clocks became available. Atomic clocks use the period of a radio frequency emitted by electron transition in excited isotopes, usually caesium or rubidium, as their oscillator. These quantum ticks are extremely accurate and stable, making atomic clocks the most accurate clocks known today. They have been used ever since as remote time references around the world. In 1967, the International System of Units (SI) redefined the second as the duration of 9192631770 cycles of radiation corresponding to the transition between two energy levels of the caesium-133 atom [21]. This ended the era of a celestially defined unit of time and replaced it with a standard defined by quantum physics.

Types of Atomic Clocks Clocks using an atomic change as the frequency base for their oscillator come in a variety of different technologies and shapes. The most important ones for practical timekeeping today are caesium and rubidium fountain clocks and hydrogen masers, more information about their build can be found in [21]. The most well known example of a caesium clock is the NIST-F2 caesium clock, it being the standard for civilian timekeeping and primary source for radio controlled clocks. It offers a stability of 0.44×10^{-15} s [22] and an accuracy of about 1 second per 300 million years. Considering the much simpler build of a maser in comparison with an atomic fountain, a hydrogen maser offers impressive performance of 1×10^{-15} s [21]. However, continuous running is difficult, as the hydrogen cavity tends to degrade over only a few days. In the last couple of years, rubidium clocks managed to draw even with caesium clocks while being easier to handle and less costly [23]. There exist even more accurate clocks today, using different elements like strontium or ytterbium and higher frequencies. These are still experimental and not acknowledged as official time references yet.

A direct comparison of implementations is somewhat difficult, because the stability hinges on the basic technology, the element used and the improvements available at the time. Table 2.1 gives a broad overview of clock stability as of 2016, usually measuring the stability over a day. The best results possible today come from optical ytterbium or strontium clocks [24], which show a stability of 1×10^{-18} s, with an estimated accuracy of better than a second over a course of several billion years.

Type	Instability
Hydrogen maser	1×10^{-15} s
Rubidium fountain	2×10^{-16} s
Caesium fountain	1×10^{-16} s
Ytterbium optical lattice	1×10^{-18} s

TABLE 2.1: Performance of Atomic Clocks

2.3 Time Systems

Greenwich Mean Time (GMT) The end of the local solar time era in civilian use across the western world came with the railways in Great Britain. While mechanical clocks did keep time, they were mostly only roughly synchronised to noon. A major weakness was the different observation angle depending on the geographical location and the varying accuracy of available equipment. The differences in local times became highly noticeable when travelling by train, chronometers needed adjusting across a trip. In 1884, Greenwich Mean Time, GMT, was introduced to address this issue. It is based on 12:00:00 being defined as the moment of the solar meridian crossing (the upper culmination of the sun) over the city of Greenwich, UK. GMT was then synchronised via telegraph or travelling mechanical chronometers to remote clocks. GMT follows a virtual "mean sun", moving along the equator instead of along the ecliptic, as it does in reality. The variation of about 16 minutes evens out over the course of a year.

The solar second was defined as 1/86400 of a mean solar day, which is defined as the mean time of two meridian crossings of the sun. But because the earth's rotational velocity is not constant, a more reliable reference frame was later sought to define a second. In 1952, the definition of the solar second was replaced by the ephemeris second. It is defined as the fraction 1/31,556,925.9747 of the tropical year on January 1st in 1900, at 12 hours ephemeris time. This became the standard for the SI system.

Universal Time (UT) UT is the modern name for GMT stemming from 1928, but is still based on the GMT system of measuring the mean solar day. The first main difference to GMT was the addition of time zones over the world as proposed in 1879 as Standard Time (ST), with the 0 hour mark on 0 longitude. The second change is the object of measurement. The observation of the sun's upper culmination does not lead to a very accurate result in terms of modern timekeeping. Our sun is too close and bright to get a good reading. Therefore the meridian crossing of other, more distant, celestial bodies like stars and quasars are observed in order to measure the earth's rotation more accurately. The Earth's current rotational velocity and the mean solar day is then computed from those findings. Today, there are several variants of UT in use for civil timekeeping [25]. These are, in ascending accuracy:

- UT0 - derived from measured rotation angle of the earth, corrected by the geographic longitude difference to Greenwich
- UT1 - refinement to UT0, correction of moving earth poles and wobbling of the earth's rotational axis
- UT1R - smoothed version of UT1, all terms with a period less than 35 days have been removed
- UT1D - refinement to UT1, correction for tidal effects
- UT2 - refinement to UT1D, correction for yearly fluctuations of earth's angular velocity

In the 1960s, radio time distribution signals were based on UT2. With the introduction of the Coordinated Universal Time (UTC) in 1972, UT2 started to become less and less important.

International Atomic Time (TAI) Based on the work by Essen et al. in 1955 [26], a relationship between ephemeris time and atomic time was established. The Temps Atomique International (TAI) was defined to start at 1. January 1958, 0:00 so that TAI time approximately equals the same timestamp in UT1. The atomic second is not exactly equal to the ephemeris second, leading to a difference in atomic and celestial time. For this legacy reason, TAI is not usually used in civilian timekeeping. TAI, as we know it today, is maintained by taking into account about 400 atomic clocks worldwide [27].

Coordinated Universal Time (UTC) UTC was first introduced as a close approximation of UT2, while taking advantage of the accuracy of an atomic clock. The earth's angular velocity is not constant due to tidal interaction with the Moon and Earth's rotation is slowing down over time. This loss of momentum is also not constant, as it is partly countered by shifts in mass of the earth core in the direction of the poles and is further influenced by smaller shifts, like changes in vegetation, snow and even strong earthquakes. On average, each day has elongated between 1 and 3 ms over the last 50 years. This makes UTC and TAI time drift apart. At the introduction of UTC in 1972, the difference was 10 seconds, today as of 2016, the difference has increased to slightly over 36 seconds [25].

In order to get both the benefit of stable atomic time and keeping in step with the sidereal day, UTC has a correction offset, the so called "leap seconds". UTC is locked to TAI but a one second offset is added or subtracted whenever the difference between UT and UTC approaches a full second. So far, all leap seconds have been positive. The update interval varies with the slope of the difference. Until 1998, UTC was updated about once a year, currently it is about every three years. While this is somewhat convenient for human life, it is a troublesome arrangement for certain technologies. This is discussed in detail in subsection 2.5.1.

Global Positioning System (GPS) Time GPS consists of a number of satellites in earth orbit. They carry caesium and/or rubidium atomic clocks and continually broadcast their current position and time for civilian use. When GPS time started in January 5th 1980, it was identical to UTC. As mentioned, it is locked to TAI by a fixed offset, but the difference to UTC will increase with every added leap second. GPS time is always 19 seconds behind TAI and, in 2016, 17 seconds ahead of UTC (July 2015). The broadcasted time reference signal will always stay true to atomic GPS system time within 40 ns or less [28].

2.4 Oscillators and Clock Signals

In this section, a general introduction to oscillators will be given and details on the relevance of their properties for timekeeping, communications and synchronous digital systems are provided. As mentioned in the introduction to this chapter, timekeeping requires a measurable periodic event and a counter for the occurrence of the said event. The phrasing “measurable periodic event” describes any system that deterministically traverses a phase-space and provides a measurable indication to the current state. This is called an oscillator.

An oscillator’s frequency is the first derivative of phase over time. The ideal linear oscillator has a constant phase change, any deviation from the constant phase rate is called phase noise. The quality of a real oscillator is therefore determined by the amount of phase noise it introduces, as phase noise governs the spectral purity of the output. The ideal linear oscillator produces a constant fundamental frequency without harmonics or overtones, i.e. a perfect sine wave.

Clock signals are used in all synchronous digital systems, or loosely speaking, computers in general. They keep the state changes in the system in step. For this purpose, a very clear indication of the transition time is desired. Square waveforms serve this requirement better than sine waves and are preferred as clock signals. More details on are provided in subsection 2.4.1 under “Non-linear oscillators”.

2.4.1 Electronic Oscillators

In physical terms, an oscillator is a device able to deterministically store and release energy, like a spring. The simplest example would be a mechanical pendulum, which, once excited, constantly exchanges potential energy for kinetic energy. Since energy is lost to friction, it must be excited again with a 180° phase-shift to the oscillation, i.e. adding kinetic energy at the maximum potential energy. An example would be a swing, where bending and straightening legs at the appropriate time excites the pendulum. In an electronic oscillator, this is substituted for the exchange of energy between a magnetic- and an electric field.

Linear Oscillators There are many different known oscillator circuits. The implementation always consists of two parts. First, a resonator (the “pendulum” from the mechanical analogue), which is providing resistance (R), an inductance (L) and a capacitance (C). Second a supply, able to excite the resonator and replenish the lost energy. The supply requires an active component, like a transistor or amplifier. The oscillator then transforms the constant supply voltage at its input into a sinusoidal voltage at its output. For the frequency ranges typical in communications, linear oscillators usually employ a tuned circuit- or crystal resonators. The first type uses a coil and capacitor, the second a quartz crystal as a resonator.

For convenience, the angular frequency ω will be used in this context. The output waveform of a linear oscillator is a sine wave as described in eq. 2.2, depending on the frequency ω and a constant phase offset ϕ.

$$\begin{aligned} \omega &= 2\pi f \\ f(t) &= \sin(\omega t) + \phi \end{aligned} \tag{2.2}$$

Oscillators have a natural or resonant frequency, at which they produce the highest amplitude in relation to their energy intake. There are more “sweet spots” at the multiples of the resonance frequency. While these are not the only frequencies oscillators can be operated at, it is often practical to stay close to them. Simply speaking, the resonance frequency of an oscillator [29, p. 107] is determined by eq. 2.3:

$$\omega_r = \frac{1}{\sqrt{LC}} \tag{2.3}$$

This relation becomes important in the introduction to Tunable Oscillators, because using a variable capacitance in parallel is a common way of dynamically changing an oscillator's output frequency.

Non-linear Oscillators Non-linear oscillators can produce non-sinusoidal waveforms, such as saw tooth, triangle or square waves. Most interesting in our context are square waves, as they are useful for synchronous digital systems. These systems require a clock signal, a periodic source of events that sharply mark the transition of one system state to the next. A clock signal would therefore ideally be a perfect square wave, its infinitely steep edges marking precisely when a state transition should occur. In practice, this is not the case, the slope is neither infinite nor constant. This makes finding the exact moment of transition more difficult.

This behaviour of a real oscillator is easily explained if one considers the frequency spectrum of the clock signal. As described by Fourier, all waveforms can be described as a sum of a series of sine waves. The Fourier series of a square wave [29, p. 237] of fundamental frequency ω is defined as eq. 2.4:

$$f(x) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin\left((2k-1)\cdot\omega t\right)}{(2k-1)} \tag{2.4}$$

It is easy to see that since the frequency multiplier $(2k-1)$ will be $1, 3, 5 \ldots \infty$, a perfect square wave has an infinite bandwidth. Since electrical impedance is frequency dependent, there must exist a cut-off frequency f_c for all materials after which they show a low-pass characteristic [29, p. 96]. This means the waveform loses its high frequencies, distorting the shape toward a sine wave. The more f approaches f_c, the less steep the edges get as they are smeared out. Since an electronic circuit would find the transition time by comparing the voltage level to a threshold, this comparison is influenced by a known measurement error. The lower the slope of the clock signal, the bigger the time interval the voltage stays within the error range. This translates directly to jitter, because the transition can happen anytime in this interval.

Therefore, fast clock signals are a compromise between sharp edges, a high fundamental frequency and the impedance they are connected to. Fanning out a clock to multiple destinations or trying to send it over a wire drastically increases the load impedance. This explains why long distance communication never features dedicated clock lines and why clock distribution in synchronous digital systems is non-trivial.

Tunable Oscillators The most important type of non-linear oscillators are tunable or variable oscillators. These are devices able to dynamically adjust phase and therefore output frequency. The two classes of interest for communication applications are the voltage controlled crystal oscillator (VCXO) and the numerically controlled oscillators (NCO). Both allow the output phase and frequency to be changed via a control parameter. The relation between this parameter and the output frequency is linear within a certain bandwidth. If the parameter is constant, the output frequency of the oscillator in turn is constant. The mechanism by which this control is achieved differs.

In a VCXO, this is achieved by adding a voltage dependent capacitance C_V, in parallel with the crystal resonator. C_V is usually a varactor diode. If the voltage to the diode is changed, its capacitance changes, and in turn does the resonance frequency of the oscillator (eq. 2.5):

$$f_r = \frac{1}{2\pi\sqrt{L\frac{C_S \cdot C_V}{C_S + C_V}}} \tag{2.5}$$

NCOs are complex devices, but they are extremely flexible. Their frequency range surpasses all other oscillator types. NCOs use a synchronous digital system to generate the numerical values for the amplitude of their output. A common setup uses a phase accumulator as an index to a table, which contains amplitude values for a sine wave. The amplitude values are transformed into a voltage by feeding them through a digital-to-analogue converter (DAC). The clocked transition from one amplitude value to another creates very steep slopes similar to the square wave in eq. 2.4. The DAC's output must therefore be low-pass filtered to suppress the undesired higher harmonic frequencies.

2.4.2 Phase-Locked Loops

Phase-Locked Loops (PLL) are electronic circuits, which can produce frequencies with a given relation to an input frequency. As shown in figure 2.1, a PLL has a phase detector in a control loop with a tunable oscillator [30, p. 955-959]. The phase detector is measuring the phase angle of the input frequency and increasing or decreasing the output phase accordingly. Their purpose is to phase match their output frequency to their input frequency. As the name suggests, a PLL locks the phase angles of both input and output. This means that once locked, the two waveforms do not shift in relation to each other.

Frequency Synthesis PLL also allow manipulation of the output frequency by means of several control parameters. Most PLL feature a frequency divider at their input, as well as a multiplier in the control loop. The divider allows production of integer fractions of the input frequency and decreases jitter at the output, the multiplier allows for integer multiples of the input frequency and increases output jitter [30, p. 974]. Both can be used at the same time in order to generate arbitrary

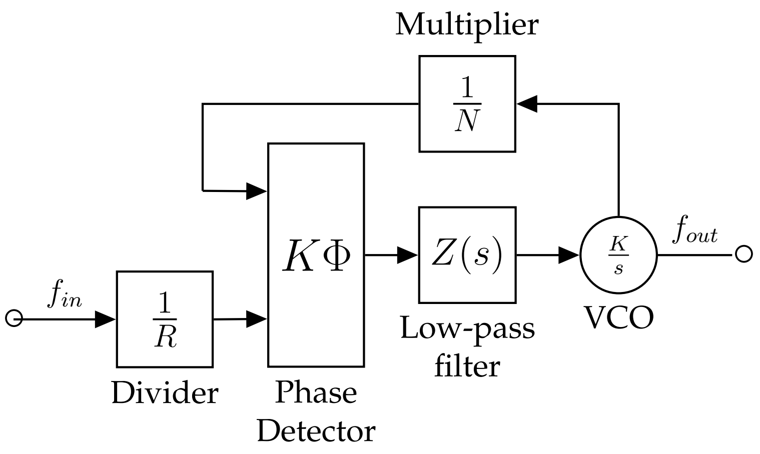


FIGURE 2.1: Block diagram of a Phase-Locked Loop [31]

fractions of the input frequency. Most PLLs also allow a constant phase offset between input and output to be added. PLL designs deriving multiple, individually controllable outputs from the same input, are also quite common.

Application There is a vast variety of uses for PLL, especially in communications. The most important ones that should be mentioned in this context are clock signal generation and clock recovery. Clock generation is important in synchronous digital systems, as there are usually numerous clock signals necessary that all share a fixed relation to their phase angles.

A simple but very common example would be a computer whose Central Processing Unit (CPU) runs at n-times the clock frequency of its bus system. Both clock signals are provided by a PLL circuit, using a crystal oscillator with a lower frequency as its input. Not only is the PLL necessary to generate the two higher clock frequencies, it is also important that the phases of the CPU clock signal and the bus clock signal match. Otherwise one or the other might sample data too early or too late. This would lead to frequent faulty data exchanges, making the whole computer useless.

The second very important application in the context of this thesis is clock recovery. This means a receiver recovering the sender's clock signal from the received data, in order to process it synchronously. A more detailed explanation follows in 2.4.3 as a precursor to the workings of the White Rabbit timing protocol in (see chapter 2.7.1).

2.4.3 Clock Recovery

When two remote, synchronous digital systems exchange data, there is a difference in how this data is handled internally. Inside the system, all data is usually latched, processed and transferred using the same sampling clock or clocks, which are phase locked and multiples of each other. If data needs to be transferred to a remote system, these clock signals are not available. For reasons explained in 2.4.1, it is impractical to have a dedicated clock line in the connection. The receiver system has its own oscillators, which do not run in phase with the sender and can even have a considerable difference in actual frequency, even if they should be nominally equivalent. The latter can stem from a variety of factors, the most significant certainly being the supply voltage and the ambient temperature. This means that a synchronous handling of the transferred data would be erratic if not impossible. In order to remedy this, the receiver's sample clock must be synchronised to the sender's. The receiver has to extract the sender's clock signal from the data stream. This is called clock recovery [32, p. 2].

The recovery is usually done by feeding the data stream into a PLL [32, p. 33-36] or similar architecture. The transitions of the incoming stream drive the phase detector controlling the VCO. If the majority of these transitions coincide with the transitions of the sender's clock signal, it can be recovered. The PLL VCO will then generate a sample clock for the receiver, which will be of the same frequency as, and phase locked to, the sender's clock signal. For this to work, the data must be encoded in a way that contains sufficient sender clock edges to enable the receiver to reconstitute the clock, because raw data does not necessarily have this property.

For example, it's easy to see that the binary sequence 11101010 still makes it possible to guess at the sender's clock. If the data word was 11111111 instead, the receiver would see a continuous 1 state without the possibility to tell how many 1 bits there actually were. To avoid this, encoding schemes are employed that balance the number of 1's and 0's in the data stream and break streaks of the same symbol. The most widely known, simplistic code for this purpose is the so called Manchester encoding. It encodes a logic 1 as the transition from 0 to 1 and a logic 0 as 1 to 0. The data word from the previous example would then be 0101010101100110 and 0101010101010101 respectively. This encoded version offers more than enough transitions for the PLL to stay locked and the data is still easy to decode on the receiving side. There are of course more sophisticated codes available today, such as 16B/18B [33], which are more economical in terms of bandwidth use than Manchester encoding.

2.5 Digital Representations of Timestamps

ISO 8601 Posix Time Format This standard comes from the Unix world, and classically used to be a signed 32 bit integer, with 0 representing January 1st 1970, 00:00. It has been heavily debated whether to make the time format signed or unsigned. In the end, the necessity to represent dates pre-Posix time, i.e. before 1970, outweighed doubling the covered time range from 2038 to 2242. The Posix time format was eventually extended to a 64 bit signed value. The standard use-case is to encode UTC. However, today most Unix/Linux systems also come with the option to encode TAI instead, using a completely leap-second-free notation. This is useful for CSs, because standard timekeeping of the OS can be used without risk of confusion or having to continuously translate timestamps back and forth between UTC and TAI.

These formats have in common that their binary representations use seconds as the base unit. They do not have a fractional part. If timestamp accuracy below a second is needed, a composite data type is available. This means adding another 32 or 64 bit word as sub-second part, which is a decimal value. This makes it easy to convert and retain accuracy over different resolutions. The number of sub-second digits can vary over format and use. The predominant standard, especially in the Unix world, is the use of UTC as the system time. UTC is closely coupled with GPS and TAI time, with one major difference, the leap seconds.

2.5.1 Problematic Nature of Leap Seconds

While conveniently linking the two time standards of UTC and TAI, the non-linear leap second offsets cause problems in time-critical programs, as UTC time is not strictly increasing. Leap seconds are a social, not a technical instrument - human lives revolve around the day night cycles and timekeeping is organised accordingly. This has also influenced the computer world. Leap seconds have been controversially discussed many times now, while being convenient for everyday life, they can be dangerous.

Peculiarities Since leap seconds are introduced whenever necessary, there is no way to convert UTC to TAI arithmetically. Instead, a table must be used. There are further complications, an added leap second is inserted at midnight, and expressed as 23:59:60. It is doubtful that the majority of computer systems feature a correct implementation of handling these time values. Many systems instead stay on 23:59:59 for two seconds, a search in a database for the timestamp 23:59:60 would therefore fail. This is of course not important for everyday life, but for all systems relying on a continuous time, such as CSs, navigation, distributed databases, computer clusters, etc., this can be problematic [13].

Effects on Schedules All schedules ultimately organise the relative times between events, and the time differences cannot be shortened or extended to match a varying time-base without consequences. In electro-mechanical systems, the range of times are in milliseconds and very rarely in microseconds, while in communications and navigation, desired system accuracy goes down to the nanosecond level. Control-systems rely on a strictly increasing and linear flow of time, as they are oriented toward a high degree of determinism. It is easy to imagine how such a time shift could cause disastrous effects. When instructions to machines suddenly lie in the past or overlap, the effects can range from loss of communication, through damage to produce and machines, all the way up to loss of life.

Persistence The main problem is not a technical issue, but a problem of culture. In machine timing, TAI, GPS or TT time would be the obvious safe choice. However, the whole computer world, including the Internet, is using UTC. Sooner or later, all complex systems involve off-the-shelf computers with standard OSs to evaluate data or handle user interaction. The simple solution to the problem would be to just use TAI, but then a carefully planned and maintained system architecture is required. Practically no fast realtime system operates completely stand-alone, at some point, computers using UTC will be encountered. This translation is computationally costly and any software not handling this correctly can lead to a succession of wrong time stamps, making the whole process error prone.

Possible end of leap seconds So far, tradition has taken precedence over utility, although the voices against leap seconds are growing louder, especially from the navigation community. In the GPS timing subcommittee report from 2014, a cessation of the leap second practice was again demanded. The report [34] stated increasing failures of navigation systems like GPS, LORAN and commercial air travel attributed to leap seconds. Further reasons for concern were the observation that approximately 10% of all NTP timing servers worldwide failed to handle the leap second in 2012 correctly, and that there is not a single documented case of all NTP servers worldwide correctly inserting their leap seconds.

Influence on the case study The discussion in regard to the FAIR case study (chapter 4) is still ongoing. The currently favoured solution to the problem is using TAI for machine timing and UTC everywhere else, converting at database-client level. Leap second updates are announced weeks in advance. The plan is therefore to perform a scheduled shut-down of the accelerator shortly before the leap second change, so no running operations can be affected. After the update, all systems shall be restarted, tested and put back into operation.

2.5.2 Timekeeping in Computers

Computers, that is, PCs without special hardware, are notoriously bad timekeepers. This stems from the fact that their primary CPU oscillators can drift as much as several minutes per day. It depends on the power settings (in modern CPUs) and does of course require the computer to be powered. Because of this, computers usually feature Realtime Clock (RTC) modules, a small crystal clock operating at 32 kHz. Using it to readjust the primary oscillator increases short term stability, but the RTC is at best accurate to 1 s per day and usually drifts by as much as 15 s or more per day [35]. The way of dealing with this problem is periodically synchronising the computer's local time to a more stable time reference via a network connection. This ultimately takes the distributed time from an atomic clock, which has an excellent long term stability. There are several network services available today to synchronise PC RTCs to UTC, the most popular being Network Timing Protocol (NTP).

2.6 Time Distribution

2.6.1 Overview

A time distribution system, or chronopher, is an electric or electronic device that makes its value of time available as a reference to remote clocks. Receivers usually have an oscillator of lower quality, which is less stable over time. It is disciplined (periodically readjusted) to the reference. Depending on requirements, the readjustment can be achieved in a single step or adiabatically. There are different concepts for the distribution. An applied method of time synchronisation at the CERN LHC was in fact transporting a caesium clock to various key sites in the facility and synchronising local clocks to it. While being very effective, the efficiency of this method is open to debate.

The most important distinction is the direction of communication either uni- or bidirectional. Most available systems are unidirectional, a client device is listening to the broadcast of the chronopher. A broadcasting station or satellite sends out a coded message with the current time to which clients readjust their clocks. There is a large number of these services available, with varying orders of accuracy. These range from the time announcement on the phone line, terrestrial radio stations and navigation beacons like the LORAN system all the way up to navigation satellites and internet based time services. Table 2.2 shows a brief overview of the possible accuracy with a variety of timing receivers [14], [36]–[40].

Receiver	Accuracy
Home radio-controlled clock	1×10^{-1} s
Industrial Radio-controlled clock	30×10^{-3} s
Network Timing Protocol	$> 1 \times 10^{-3}$ s
LORAN	1×10^{-6} s
GPS	1×10^{-7} s
Precision Time Protocol	1×10^{-7} s
White Rabbit Protocol	1×10^{-9} s

TABLE 2.2: Timing Receivers and possible accuracy

2.6.2 Disciplined Oscillators

Crystal oscillators, and thus the local time driven by them, drift over time. Because it is neither technically nor economically feasible to replace all timekeeping by atomic clocks, it is common practice to use one of the described time distribution services to periodically re-adjust lower quality clocks to more accurate time references. The best known implementations are satellite disciplined oscillators, which are high quality local oscillators disciplined to the atomic time of navigation satellites. This used to be just GPS, but multi-receivers able to combine GPS, GLONASS, Beidou and Galileo satellites are available today. This provides time and frequency normals with excellent short and long term stability at a fraction of the cost and maintenance effort of using an atomic clock each.

2.7 Network Timing Services

Timely availability of services, goods and storage are the most central aspect of today's life. The bigger, global and faster these processes became, the more importance the accuracy of time synchronisation became. Since computers are involved at all levels and most often connected to the Internet, be it directly or via gateways, there is a big demand for network service able to accurately synchronise local times. Strictly speaking, GPS is also a time distribution protocol over packet based networks.

Evolution The choice of protocols available today is surprisingly small. The earliest standard was (IRIG) from 1956, using proprietary connections and systems to distribute one timestamp per second without correction for propagation delay. The implementation with the largest impact is NTP by Mills, which remained the de facto standard for most of the world's computers until the present day. It is a bidirectional protocol, which considers message propagation and was later extended to discipline the client computer's oscillator as well.

This was followed by the Precision Time Protocol (PTP) 1588, developed by the Institute of Electrical and Electronics Engineers (IEEE) and published in 2002. PTP adds support for more accurate hardware timestamping and PLLs and does not stipulate a specific technique for oscillator discipline. It was developed for filling the niche between GPS disciplined systems and NTP, that is, increased accuracy without satellite receivers. There are several other clock synchronisation protocols, which do not distribute absolute time. These are Synchronous Optical Networking (SONET) from 1984, Synchronous Digital Hierarchy (SDH) from 1988, and Synchronous Ethernet (SyncE) from 2010.

In 2008, Moreira et al. published their work on the White Rabbit Protocol [40]. WR is combining methods from several standards and was developed with the explicit aim of further increasing the possible accuracy of PTP.

2.7.1 White Rabbit Timing Protocol

WR is a time synchronisation protocol, similar to PTP. More precisely, WR is an addendum of PTP, using its built in hardware support for hardware timestamping and employing special hardware for phase detection and offset compensation. The aim in WR development was going toward the creation of "*a particularly accurate and precise implementation of the IEEE 1588 standard*" [42]. WR combines several techniques that address time synchronisation issues at different levels. These are:

Time This refers to the exchange of local times between a server and a client. The server is connected to a high quality time source. This is either an atomic clock, or, more often, a satellite receiver linked to an atomic clock. The server's time announcement is compensated for transmission delay which is calculated by timestamping packet exchanges between client and server [41], [43].

Link External influences on measured link delay must be reduced to a minimum. To exclude wait times from cross-traffic from the measurement, PTP packets are timestamped directly at the physical interface (PHY). For highest accuracy, link asymmetry, that is, the difference of delays per direction, also has to be considered in time synchronisation.

Frequency Asynchronous oscillators cause a deviation in local time between synchronisation updates. A control loop at the client side has to adjust its frequency to match the server's, so their timers run at the same rate [42], [44]. The server is connected to a frequency normal (high quality oscillator disciplined by satellite receiver).

Phase To synchronise local time to better than one transmission clock period, it is necessary to get more accurate measurements of the clients phase offset to the server and compensate for the offset.

2.7.2 IEEE 1588 Precision Time Protocol

There are numerous PTP implementations available today, their accuracy largely depends on the level at which the message handshake is timestamped. The closer to the PHY, the higher the accuracy. PTP utilises many different types of clocks, but in this context, only some of them are of interest.

Master and Grandmaster Any PTP clock correcting another is a master clock, a PTP clock receiving no corrections but correcting all others is referred to as a grandmaster clock.

Slave A PTP clock being corrected by another clock is a slave clock; if it does not correct others, it is called a slave-only clock.

Boundary and Transparent The interesting cases are boundary and transparent clocks, as they are both master and slave at the same time. Both require timestamping to happen outside their own control loop, right at the PHYs. A transparent clock forwards the sync messages it received at its slave port on its master port. It compensates its own delay by either updating a correction field on the fly or sending a follow-up message afterwards. A boundary clock acts as sink at its slave port and generates fresh, already corrected, sync messages at its master port. Transparent and boundary clocks both correct for link delay and packet residence time at their own node, and thus the layout of a node's control loop has no influence on time synchronisation [40].

PTP nodes are able to auto negotiate for the most accurate clock as the preferred grandmaster clock. WR does not use this feature though, and uses a fixed preferred grandmaster clock instead. In the intended tree or star topology for a CS, the root clock has no master and must therefore be a grandmaster clock.

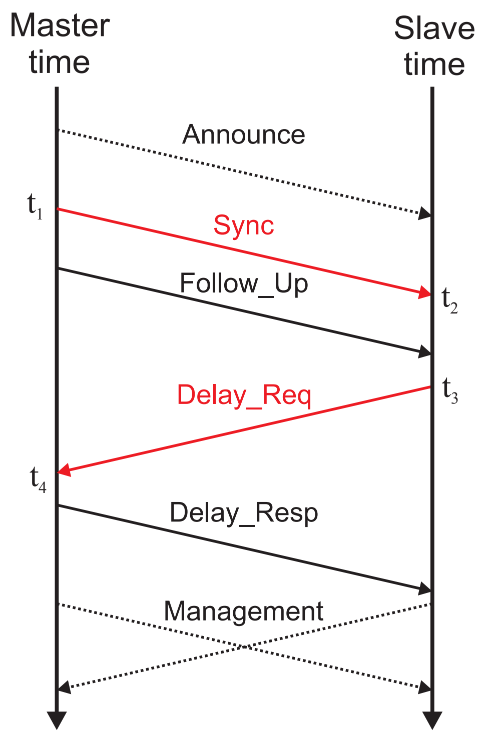


FIGURE 2.2: Simplified PTP Synchronisation Handshake [42]

Figure 2.2 shows a simplified version of the PTP exchange. All message arrivals and departures are timestamped, the link delay between server and client is calculated by latency of packet exchanges.

The calculation of the link delay between master and slave (not considering link asymmetry), following the exchange from figure 2.2, can be written as

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{2.6}$$

Pure hardware PTP solutions can reach a mean difference to the reference of several hundred picoseconds, but have a relative jitter (standard deviation) of several nanoseconds [43]. The reason for the high jitter is the plesiosynchronous relation of server and client clocks, which start to deviate between updates. Contrary to NTP, PTP does not feature a native way of correcting frequency drift in nodes. Thus, between PTP handshake updates, the local oscillators run free. The possible quality of PTP synchronisation therefore depends on the following factors: Oscillator quality, which determines the drift rate, PTP update interval, which determines the maximum accumulated drift, timestamp granularity, which determines the accuracy of drift correction [39], and, as mentioned in the beginning of this subsection, the level at which timestamping takes place.

2.7.3 Synchronous Ethernet

WR makes use of part of the Synchronous Ethernet International Telecommunication Union (ITU) recommendation of phase locking carriers [45], also called Layer-1 syntonisation. The term "layer" refers to the Open Systems Interconnection Model

(OSI), in which layer 1 defines the physical (bit transmission) layer of network communication. Layer-1 syntonisation describes the process of keeping the transmission clocks of different nodes in phase at all times. Their clocks tick with the same frequency and phase, the drift of local time between PTP updates is reduced to a minimum defined by the clock recovery and oscillator accuracy.

Origin Standard IEEE 802.3 Ethernet is by definition asynchronous. In the beginning of Ethernet, a network was a group of computer nodes on a shared physical medium and there is no separation between RX and TX directions. Nodes sending at the same time cause collisions resulting in unintelligible traffic, and therefore have to remain silent between transmissions. During an incoming transmission, a node's sample clock is synchronised by a packet preamble of alternating bits (see clock recovery, 2.4.3) to which the receiving PLL can lock, but its oscillator is running freely otherwise.

Properties In several implementations including Gigabit Ethernet (GbE), a network is no longer sharing a physical medium. All connections are point-to-point and RX and TX are separated. This removes the possibility of collisions and thus the necessity for a node to stay silent between transmissions.

The line coding used in $1\,\mathrm{Gbit/s}$ Ethernet (8b/10b) [46] provides a set of alternating codes for data, which allows clock recovery. A subset of the line codes are the so called comma codes, which are employed when the line would otherwise be idle. They are recognised as not being valid data, but also form an alternating pattern containing enough signal changes for the receiving clock recovery circuit to stay locked [42], [45].

Clock recovery alone does not provide information to which clock edge and therefore which bit position in a data word the receiver is locked, though. If it did not lock to the first bit in a word, data is split incorrectly into words and thus wrongly interpreted. This is called bit slip. 8b/10b coding allows the correct synchronisation to word boundaries by the recognition of valid code words within the stream.

2.7.4 Phase Detection Hardware

GbE sends at $1\,\mathrm{Gbit/s}$ ($1.25\,\mathrm{Gbit/s}$ with 8b/10b coding), but the programmable hardware containing the WR core cannot manage the same frequency. Unlike the physical interfaces, it is limited to $\leq 200\,\mathrm{MHz}$; the data rate of the network is achieved by dedicated high speed circuitry in the transceiver (see 7.6.3).

WR thus needs to derive a $125\,\mathrm{MHz}$ frequency (nearest sustainable integer relation to $1.25\,\mathrm{GHz}$) from the recovered $1.25\,\mathrm{GHz}$ RX clock and lock the phase of it's own $125\,\mathrm{MHz}$ core oscillator to it. The receiving node can then derive its own $1.25\,\mathrm{GHz}$ TX clock from the core clock. Because WR's core frequency is $125\,\mathrm{MHz}$, the time resolution is limited to the core period of $8\,\mathrm{ns}$ [39].

The accuracy of WR PTP synchronisation is ultimately determined by the accuracy with which the phase offset between the local and remote oscillator can be measured. This causes a problem, as the phaseshift is smaller than a WR clock period and can therefore not be directly sampled by the hosting programmable hardware.

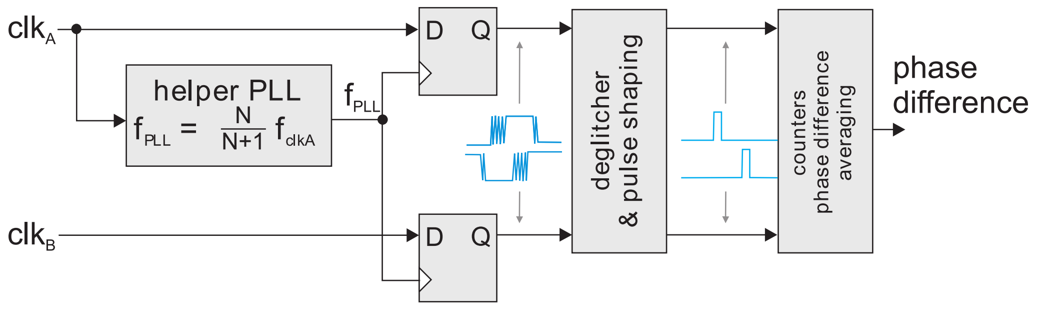

clk_A
helper PLL
$f_{PLL} = \frac{N}{N+1} f_{clkA}$
f_{PLL}
clk_B
D Q
D Q
deglitcher & pulse shaping
counters phase difference averaging
phase difference

FIGURE 2.3: White Rabbit Phase Measurement Hardware [42]

Undersampling Because clock signals are periodic, determining the phase offset with high accuracy does not necessarily require a higher sampling rate. The technique employed in WR is making use of a beat frequency approach. As figure 2.3 shows, the special hardware consists of a numerically controlled oscillator, producing a frequency at a slight offset to the received clock signal. If the clock signal is sampled with it, a slow beat frequency is created. Despite being noisier and having some jitter, it is directly related to the phase offset of the clocks. After cleaning the signal, it can thus be used to find the clock phase offset. Because it is much slower than the clocks themselves, it can be easily sampled by the programmable hardware. The effect is a virtual increase in sampling rate, "magnifying" the view on the phase offset [42].

2.7.5 Link Delay Model

In eq. 2.6, PTP link delay was simplified by the assumption of symmetry. This is of course not the case for a real link, and the yet unknown quantity is therefore the delay distribution between link directions as well as between transceivers and receivers.

WR's current implementation solely employs fibre-optic links. While it is theoretically also possible to synchronise copper links, it is more difficult. The electronics in copper PHYs has to serve multiple twisted pair leads and proved to be not deterministic in the desired time range. In addition, the multiple differential pairs can have slightly different lengths, leading to different propagation delay and adding to the jitter of the PHYs' electronics. For the same reason, WR uses fibre-optic links with wavelength multiplexing, meaning TX and RX direction are going through the same fibre, but on different wavelengths. For separate fibres for both directions, the difference in latency caused by different fibre lengths and temperature expansion is hard to calculate. The link asymmetry resulting from the wavelength dependent propagation through fibre and the different types of PHY can be obtained from a measurement setup. These asymmetry calibration values are then valid for all setups using the same combination of fibre and PHY types.

Matching Core Clocks To make core clocks on both sides match (and thus points at which their time counters increase), the client's 125 MHz core clock is shifted by the measured phase offset, taking into account link asymmetry calibration and bit slip. It then aligns with the word boundary of the sender's core clock. The measured residual phase offset at any node is included into the PTP correction field to the next client downstream. This way, phase offsets do not accumulate between clock layers. After the PTP exchange, local time in both nodes is synchronised to less than half a transmission clock period (≤ 400 ps), and stays this way due to the oscillator lock. Theoretically, a single PTP handshake would be sufficient to ensure synchronous time in the system. It is nevertheless periodically refreshed for reasons of robustness [42].

2.7.6 White Rabbit Performance

In order to show the degree of synchronisation accuracy, the 10 MHz clock outputs of cascaded WR switches were compared in terms of mean skew and standard deviation against the frequency reference connected to the top switch. The results are presented in table 2.3. All switches showed a mean skew to the reference of ± 200 ps and a standard deviation of ≤ 7 ps, experimentally proving WR accuracy to be superior to PTP.

Switch	Integrated jitter [ps]	Offset [ps]	
	10Hz-40MHz	mean	sdev
1	1.6637	161.86	5.45
2	2.4887	24.67	5.30
3	2.3025	-135.25	6.14

TABLE 2.3: WR Performance Evaluation [47]

2.7.7 Summary

After establishing the basic context and presenting examples of modern time synchronisation technology, the properties of the WR concept could be evaluated and discussed. The WR time distribution system was proven to provide absolute time and clock signals at hitherto unknown accuracy while maintaining a scalable hardware effort. Considering the large scale of the FAIR project and other large accelerators like the CERN LHC, both geographically and in the number of required endpoints, this is of particular importance. The good long term stability of an atomic clock as an ultimate reference is an added bonus of the satellite disciplined timing source suggested for WR. The performance evaluation therefore confirmed WR being a suitable choice for the infrastructure of a high accuracy CS for large physics experiments.

Chapter 3

Particle Accelerators

3.1 Overview

This chapter covers two main topics. The first is an introduction to the basic physical principles of particle accelerators, as far as they are relevant for functional understanding. Further details about relativistic effects and particle physics can be found in the relevant literature [48]–[50]. The second topic contains functional explanations for common accelerator designs and their sub-components. All of these aim to give an insight into the reasoning behind the adoption of the kind of control and timing these machines need. More details on the accuracy of the necessary coordination are found in the introduction to timing in chapter 2, a detailed quantitative timing discussion of individual components can be found in the FAIR case study in chapter 4.

3.2 Physics

3.2.1 Charge and charged particles

According to today's scientific view as of 2016, all matter is composed of subatomic particles. For the purpose of accelerator fundamentals, we will limit this view to electrons, protons and neutrons. As measured by Faraday and later demonstrated by the Millikan experiment, the charge of a free standing particle is always an integer multiple of the elementary charge e. By convention, electrons are said to have a charge of $-1\,e$, protons $1\,e$, and neutrons are, as their name suggests, electrically neutral. Atoms having the same number of protons, but differing in neutrons, are called isotopes. Any atom that is not electrically neutral is called an ion. Ions can either have positive or negative charge, determined by the amount of missing or surplus electrons. There are accelerators for electrons, protons and heavy ions, which differ in technology. Free electrons are easier to produce than protons or ions because of their low binding energy. They are also easier to accelerate because of their very low mass.

Besides the common particles, some accelerators can also produce and handle antimatter. These are particles of the same mass as their common counterparts, but with an opposite charge. An anti electron is called a *positron*, the counterpart of a proton is simply called *antiproton*. Whenever a particle and anti particle collide, both are annihilated and their energy is emitted as photons (radiation). Their production is effortful compared to ordinary particles and usually achieved by bombarding special targets with protons or heavy ions.

3.2.2 Forces

Accelerators use a combination of electrical and magnetic forces to accelerate and steer charged particles by means of electrodes and electromagnets.

Lorentz force - Electric Component Any charged particle is attracted to the electrode with a charge of opposite sign as the particle and repelled by charges of the same sign. This electrostatic force is applied whenever a charged particle is exposed to an electric field. Voltage, that is, electric potential difference between electrodes, is defined as the work that has to be applied to a charged particle in order to be moved against a static electric field [51]. This can be understood roughly as equal to a gravitational force: a charged particle inside such a field $\vec{E}$ has a potential energy equal the voltage times its own charge (q). When the electrical force ($\vec{F_{le}}$) is applied, this energy is transformed into kinetic energy, the particle "falls" toward the charge with opposite sign 3.1. Typically, the kinetic energy of a particle is expressed in electron Volts(eV), meaning the energy an electron achieves after being moved over a potential difference of $1\,\mathrm{V}$. This is approximately equal to $1.6 \times 10^{-19}\,\mathrm{J}$.

$$\vec{F_{le}} = q\vec{E} \tag{3.1}$$

Lorentz force - Magnetic Component Whenever a moving, charged particle enters a magnetic field ($\vec{B}$) with a vectorial element perpendicular to its own path at a speed ($\vec{v}$), it is exposed to a force ($\vec{F_{lm}}$) orthogonal to both it's path and the field lines:

$$\vec{F_{lm}} = q\vec{v} \times \vec{B} \tag{3.2}$$

Since the magnetic Lorentz force is equivalent to a centripetal force ($\vec{F_c}$ in eq. 3.3) on the moving particle, it will curve its path. The radius (r) of the resulting curve depends on the mass (m) of the particle, its speed and the orthogonal magnetic flux. Continuous exposure to an orthogonal magnetic flux will force the charged particle into a closed orbit, a technique used in "ion traps" to immobilise ions for observation.

$$F_c = m\frac{v^2}{r} \tag{3.3}$$

In order to simplify the equation, only the orthogonal magnetic flux component is considered. The radius is then given by Eq. 3.4:

$$\begin{aligned} F_{lm} &= F_c \\ vB &= m\frac{v^2}{r} \\ r &= m\frac{v}{B} \end{aligned} \tag{3.4}$$

3.2.3 Why use Magnets at all?

Electromagnets are much more complex to build and costly than electrical deflection plates, not to mention big and heavy. So, why use them at all if electrical force can also change the path of charged particles? The reason comes with particles of high kinetic energy. Electrically forcing these from their path requires very high voltages. Wiedemann explains the necessity for magnets as *"For relativistic particles* ($v \approx c$) *we find that the force from a magnetic field of* $1\,\mathrm{T}$ *is equivalent to that for an electrical field of* $300\,\mathrm{MV/m}$*. Since it is technically straightforward to generate magnetic fields of the order of* $1\,\mathrm{T}$*, but rather difficult to establish the equivalent electric fields ..., ...most beam guidance and focusing elements for relativistic particle beams are based on magnetic fields."* [50]

3.3 Types of Particle Accelerators

3.3.1 Linear accelerators (LINAC)

Function This form of particle accelerator uses an electric field to accelerate charged particles, i.e. electrons or ions, along a linear beam line. Like all accelerators, it must be evacuated to prevent collisions of the accelerated electrons/ions with neutral gas particles. LINACs consist of an *"...array of metallic tubes with an applied potential creating an electric field between tubes, but not inside them. Charged particles launched along the axis of the tubes were to be accelerated while traversing the gaps between adjacent tubes but drifted at a constant velocity when inside a tube."* [49, p. 163-164].

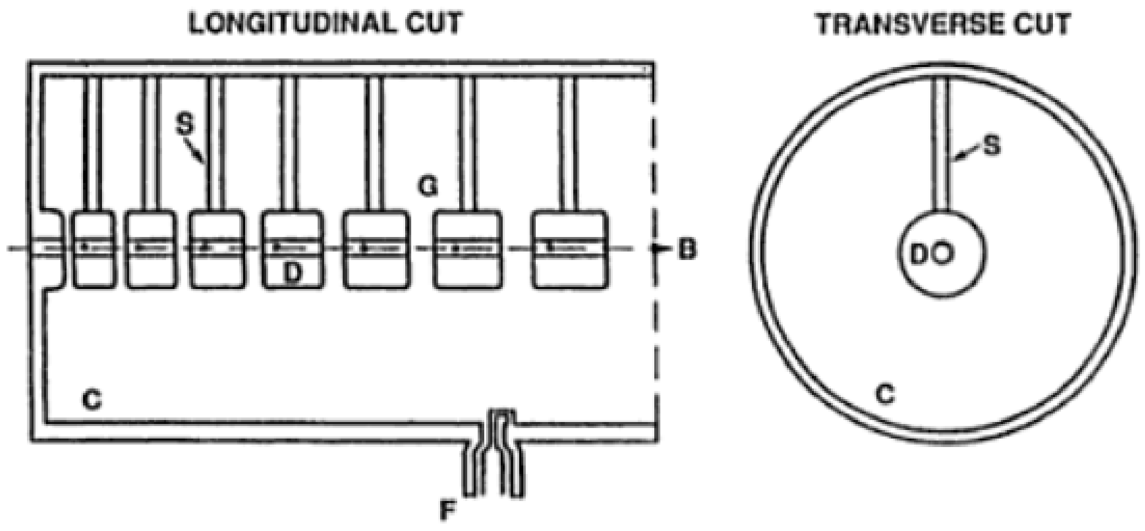


FIGURE 3.1: Schematic of a drift tube linear accelerator based on the Alvarez design [49]
C cavity, *S* struts, *D* tubes, *G* gaps, *F* RF input, *B* axis

Since the charge of the particles is fixed, the polarity of the electric field needs to change with each gap traversed. This can either be achieved by applying accurately synchronised voltage pulses (Ising 1924) or, as Wideroe proposed in 1928, more simply by *"the application of radio frequency (RF) voltages to a succession of drift tubes ... of progressively increased length"* [49, p. 163-164]. As particle speed increases with each gap, the time of flight between gaps decreases. One way to cope would be to offset the time to reverse polarity of the electric field ever increasingly earlier for each tube along the path. This phase shift would then fit the original Ising proposal and a machine built in that way is called an induction LINAC. These machines are relatively new because the required precise timing has only become widely achievable since 1980. The other possibility is to match the length of each drift tube to the particle speed at this acceleration stage in order to even the time of flight. This would match the Wideroe method.

Alvarez Design In 1947, Alvarez designed a linear accelerator for low energy protons, based on the above principles by Wideroe. The Alvarez tank is built in a way that the acceleration is in phase over all gaps. *"The field varies sinusoidally with time, but the drift tube lengths are chosen so that the beam bunches are cumulatively accelerated along the axis ... "*.

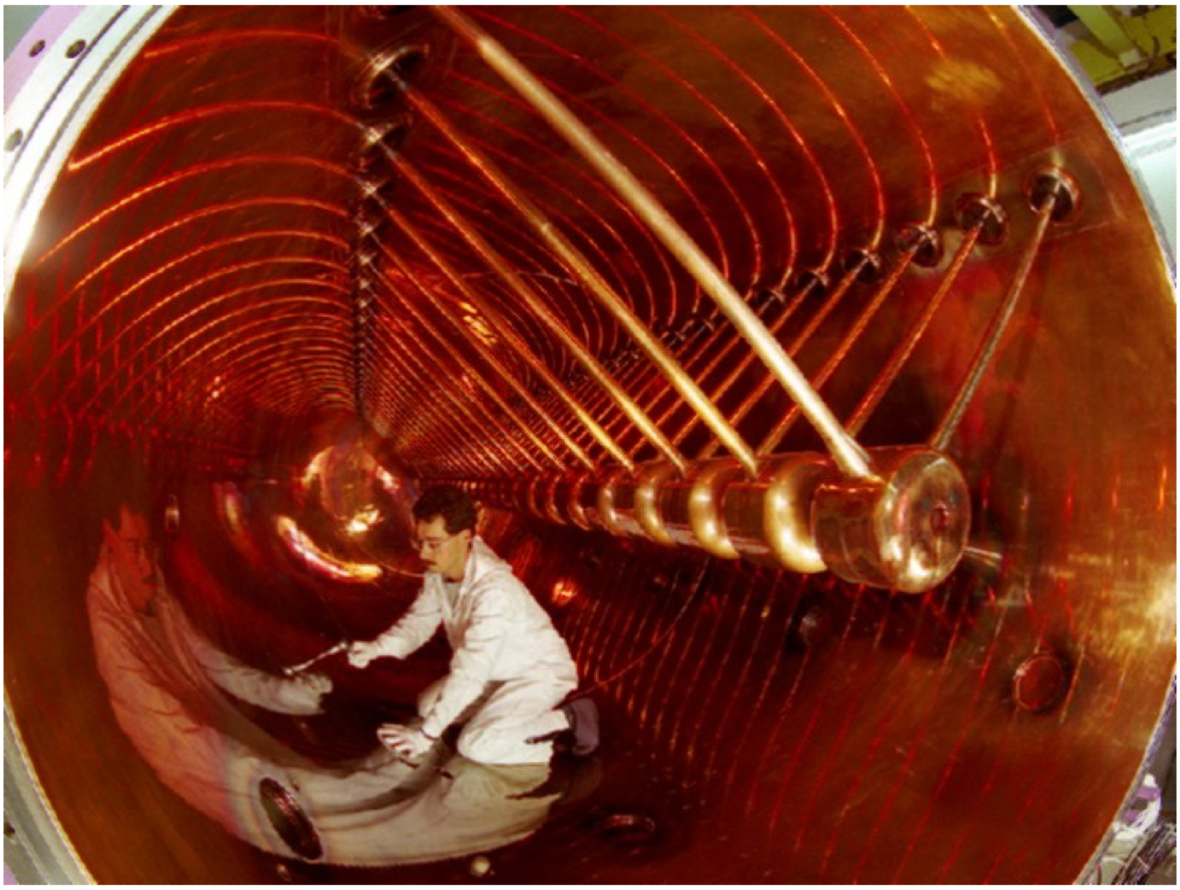


FIGURE 3.2: Inside of an Alvarez tank at GSI. The drift tubes in the centre are supplied with the RF voltage by the diagonal struts. [52]

It also *"allows bigger drift tubes with enough space to put focusing elements"*. This reduces beam dispersion and therefore allows higher energies. Linear accelerators are often used as an injector for synchrotrons, which often need a much higher particle energy at injection than an ion source can deliver. GSI's UNILAC is an RF voltage driven linear accelerator machine which uses four Alvarez cavities (Figure 3.2) to accelerate heavy ions to a final energy of $11.4\,\mathrm{MeV}$. The absolute timing requirements of a LINAC are relatively simple, as it uses an RF voltage of low frequency, e.g. $50\,\mathrm{Hz}$ (mains) for UNILAC. It is important to distribute the RF signal with low jitter in respect to each other both inside the LINAC and following RF cavities.

3.3.2 Cyclotrons

Cyclotrons are the simplest possible type of circular accelerators. Today they are mostly used for electron beams for ionisation and X-ray light sources, but understanding the processes inside a cyclotron is helpful as an introduction to the more complex synchrotron machines.

Function This type of circular accelerator makes use of an oscillating electric field and a constant magnetic field. It consists of a flat cylinder, which has been split along a plane parallel to the cylinder's axis. The two halves are separated by a small gap and are the positive and negative electrodes of an RF voltage source. When charged particles are inserted at the centre of the cyclotron, each time they enter the gap, they are accelerated into the opposing half, the direction depends on the current phase of the alternating voltage. A constant coaxial magnetic field exerts a Lorentz force on these moving particles. This centripetal force drives them on a half circular path perpendicular to the cylinder's axis with a radius depending on the magnetic flux, particle charge/mass ratio and speed [48, p. 7-9]. With each crossing of the acceleration gap, their speed increases in steps and they spiral outward until they reach the tangential extraction tube. A special property of cyclotrons is the constant orbit time for particles, no matter their current speed. This is because the curve radius proportionally increases with each acceleration. This means that the frequency used for acceleration must be the orbit frequency or one of its harmonics in order for all particles to be in phase. GSI, and later FAIR, employ cyclotrons in a certain type of ion sources. In an electron cyclotron resonance ion source, electrons are accelerated in a cyclotron and shot at a heated chemical element to create ions.

3.3.3 Synchrotron Machines

The idea of a ring of magnets and drift tubes to accelerate particles first appeared in 1943 in a proposal by M. Oliphant to the Department of Scientific and Industrial Research: *"Particles should be constrained to move in a circle of constant radius thus enabling the use of an annular ring of magnetic field . . . which would be varied in such a way that the radius of curvature remains constant as the particles gain energy through successive accelerations by an alternating electric field applied between coaxial hollow electrodes."* [53]

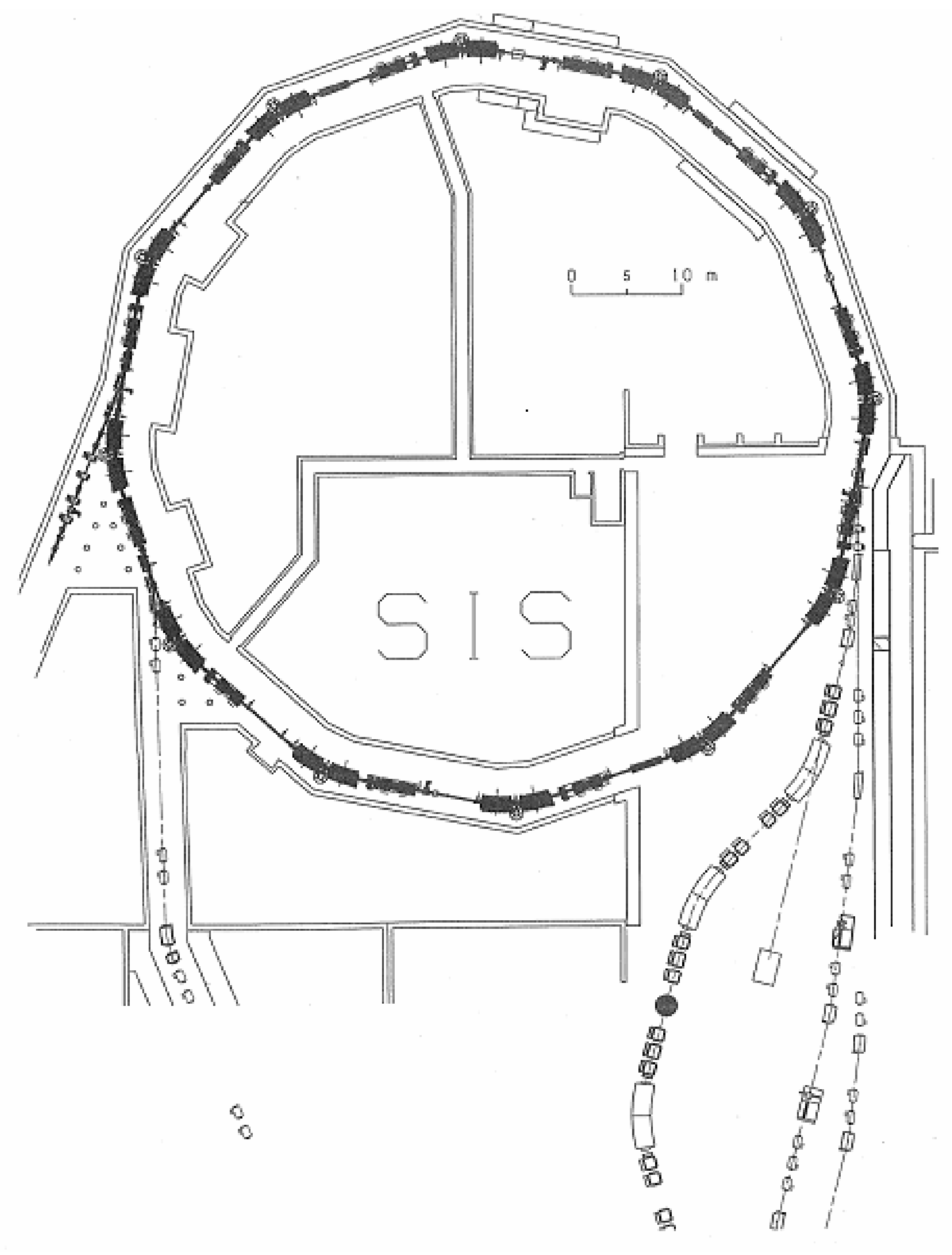


FIGURE 3.3: Schematic of the SIS18 Heavy Ion Synchrotron at GSI. The SIS18 is built as a 12 sided polygon with an injection- (left), a re-injection- (second on left) and an extraction-channel (right) [54]

Function A synchrotron is therefore a ring shaped conjunction of linear acceleration sections, focusing elements and guiding magnets. It is vastly more space efficient than a linear accelerator because the same acceleration stages are reused and particles can also reach much higher energies.

Accelerating, Focusing and Bending While called a ring, a synchrotron is built in the shape of a polygon (see figure 3.3). In its simplest form, it would consist of straight sections of beam guides containing RF elements for acceleration joined by dipole magnets at the corners (see figure 3.4). The dipole magnets emit a field perpendicular to plane of the ring. The ensuing Lorentz force curves the beam inward, off its tangential trajectory and into the next beam guide. Thus, they maintain the horizontal orbit of the beam inside the accelerator [48, p 12-17]. The magnetic flux needed to maintain orbit changes varies with particle speed, only electro magnets are suitable to provide the high flux needed to control relativistic beams.

There are basically two types of magnets: focusing and bending. Focusing magnets come in two shapes, quadru- or sextupol. They effect course corrections on the beam, concentrate it by reducing its diameter and counter beam spreading. The corner magnets on the other hand send the beam around the corners of the polygon. With current accelerators, this flux can be as high as $8\,\mathrm{T}$ in case of the CERN LHC (for comparison: magnetic field of the earth $< 65\,\mu\mathrm{T}$). The top speed of particles achievable inside ring accelerators depends on the mass of the particles but can reach high fractions of the speed of light. Light particles, such as hydrogen, reach as much as 98 percent of light speed in GSI's SIS18 synchrotron.

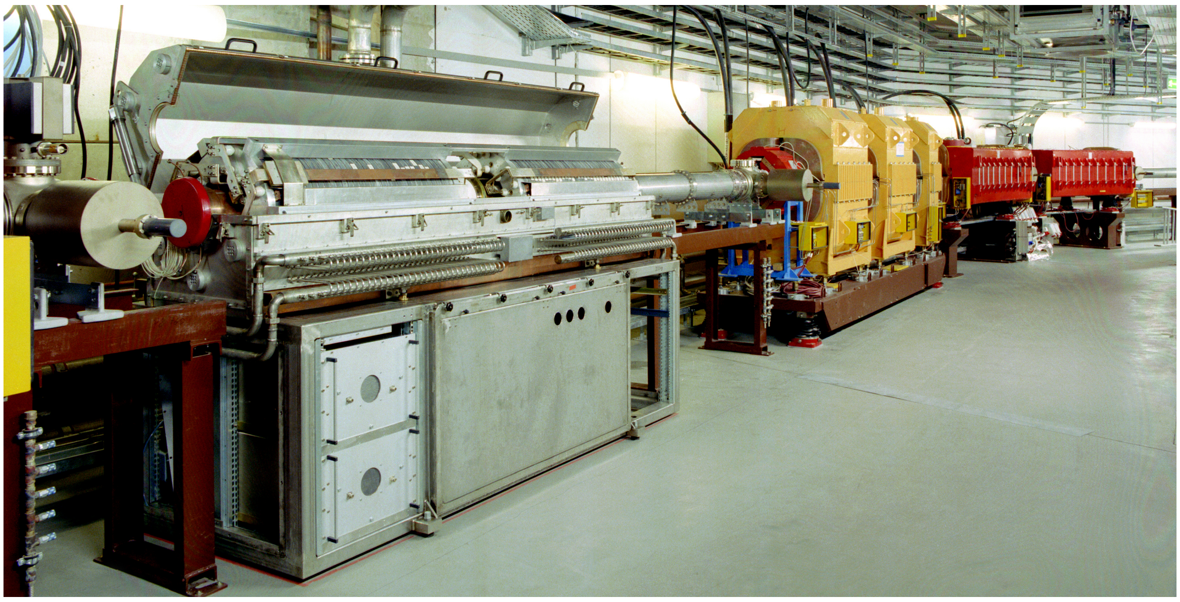


FIGURE 3.4: Synchrotron Components (SIS18)
Front to back: RF acceleration section, yellow quadrupole focusing magnets, red dipole bending (corner) magnets [54]

Requirements of Synchrotron Machines Contrary to a cyclotron, the orbit frequency of synchrotrons changes proportionally to particle energy while the desired orbit stays constant. This has several implications. First and foremost is that RF voltages and magnet current ramps must be synchronised in regard to the current orbit frequency of the particles. This demands accurate time and clock distribution throughout the machine.

A synchrotron is also not capable of accelerating particles below a certain kinetic energy threshold, because the minimal magnetic flux the power supplies can generate is too strong. Slow moving particles are deviated too strongly to hit the following beam-line within its acceptance angle. Instead, an injector stage, usually a linear accelerator, must be used to acquire the minimum kinetic energy before injection.

As already described in the introduction to linear accelerators (3.3.1), the RF voltage is a waveform "moving" around the ring with the accelerating particles. The whole machine is oscillating and this means that at any given time, there are positions on the ring where particles can be accelerated and where they would be decelerated. It is obvious that that there cannot be "correctly" oriented fields at all places along the ring at the same time, but there are travel zones. Particles therefore bunch up into distinct clouds, each one in the stable part of the longitudinal phase space, the so called bucket. If a particle falls behind, it is either accelerated again into the bucket centre, or, if it became too slow, approach the following bucket. If a particle is too fast, it will leave the bucket's optimal acceleration zone, being slowed down again. The possible number of buckets in a synchrotron depends on the ring circumference and the RF harmonic used, the higher, the more buckets.

Particle acceleration is equal to an increase in orbit frequency, which in turn means increasing radio frequency and magnet ramp speed. Acceleration changes the position and size of the buckets, which has to be done adiabatically in order to keep particles near the bucket centres. If the bucket is moved too quickly, particles of the bunch fall outside the bucket's acceptance zone, beam loss occurs as particles are unable to follow the orbit and hit the beam-line surface. This limits the speed of acceleration and is the reason why creation of highest energy particle beams can take as long as several hours.

Beam Injection and Extraction Synchrotrons also need a mechanism to inject or extract beams from the ring. They have (usually) planar, tangential injection- and extraction tubes. Special dipole magnets with a fast reaction time called "kickers" are employed in the flight path before such a junction to get a beam into or out of a synchrotron. The Lorentz force a kicker generates is sufficient to deviate a particle beam by several degrees, the SIS18's for example uses 9°. This is enough to hit the extraction tube or catch an injected beam and force it into the ring orbit. The magnetic flux increases proportionally to the number of windings on the coil, but more windings also increase impedance to current changes and prolong the rise and fall times of the magnet current. In order to ramp up the magnetic field fast enough to influence only a single bunch of particles in the synchrotron, they consist of very few windings. Kickers draw a high current that comes from very large electrical capacity charged in advance. The goal is to get a current pulse with short rise and fall times and an even current in between. Because capacitors show an exponential discharge current characteristic, large drums of thick (several centimetres) coaxial cable are used. The cable's equivalent circuit diagram is series of capacitors, which creates an almost constant discharge current during the pulse. The said current pulse must be very accurately timed. The most demanding scenario is a so called "bunch-to-bucket-transfer" (B2B) between synchrotron rings. As stated before, there are a number of places in the phase space of the synchrotron where clouds of particles can exist in an orbit. A B2B describes the movement of a particle bunch from one accelerator ring into a bucket at the other. As an analogy, one could think of two jugglers each juggling several balls who wish to pass a single ball from one to the other. The ball (bunch) must be thrown so it arrives at the second juggler's hand exactly at the moment it is free (bucket). It is then easy to see that both must be synchronised in order to do such a transfer.

This is achieved by shifting the phase of one ring in relation to the other, meaning acceleration or deceleration for a short time before returning to constant orbit speed. The amount of necessary phase shift is determined by measurements of the bunch/bucket phases. The extraction, injection and the synchronisation between RF cavities are the most demanding applications of a timing system in a particle accelerator.

3.3.4 Colliders

The idea of a collider is to accelerate particles to very high energies and hit other, either stationary particles or another particle beam head on. In the second case, particle beams from two accelerators travel in opposite directions so the combined speed is double that of a beam hitting a stationary target. Synchrotron colliders are used to achieve the highest collision energies between particles possible today. They consists of two synchrotron rings with equal or near equal diameter built in close proximity. The rings accelerate particles in opposite directions and are joined by a beam-guide. When both machines reach extraction energy, the beams are collided and the resulting collision occurs at double the energy achievable with the hardware of a single ring. The most prominent example of a collider ring is the LHC at CERN with a circumference of $27\,\mathrm{km}$ which will be able to achieve centre of mass energies of more than 1.1×10^{12} eV for lead ions.

3.3.5 Storage Rings

Storage or collector rings, are synchrotron machines and are used, as their name suggests, to accumulate particles and hold an existing beam for several hours or even days. They are not meant to increase the energy of the beam they are holding, but to keep it at a constant level, refine it and lose as few particles as possible over time. While increasing particle density by collecting more particles from injector accelerator, they focus and cool the beam. Cooling is in the context of a particle beam understood as dampening all motion of the particles other than their intended, ideal circular orbit. The more concentrated the beam and the less transversal motion, the more effectively it can be used for experiments. Storage rings usually have the same timing requirements as the accelerator rings, maybe slightly more relaxed because the tend to run the same machine sequences for hours on end.

3.3.6 Ion Sources

The common factor of all particle accelerators is that they can only accelerate charged particles, meaning either electrons or ions. Electrons can easily be extracted by heating a cathode and drawing them away from the cathode toward a ring-shaped anode, thus creating an electron beam. Creating an ion beam however is usually more difficult, the necessary effort depends on the chemical element. If the beam element can be easily evaporated, electron bolt ionisation can be employed. This uses an electron beam that is shot at free gas atoms of the beam element. If the particle energy of the electron beam is higher than the binding energy of the electrons at the hit gas atom, the latter electron will be removed and the atom is ionised. The removed electrons hit other atoms, creating an avalanche effect. For more durable materials or rare elements, of which only small quantities are available, sputter ionisation can be used. In this case an electron beam creates heavy ions from easily ionised materials, caesium is often used for this purpose. The caesium ions are then accelerated and used to bombard the beam element, in turn sputtering particles from its surface. After their ionisation, these can be separated from the caesium and directed to the output of the source. GSI and the later FAIR facility feature several types of ion sources and can create various types of heavy ions, such as carbon, copper, lead and uranium for example.

3.4 Miscellaneous Components

3.4.1 Targets

Targets for proton, electron or ion beams literally come in all shapes, materials and sizes. This ranges from solid, liquid or gaseous elemental targets for pure physics experiments through organic tissue as a heavy ion target for cancer treatment research right up to satellite components subject to bombardment with artificial solar wind. Detector arrays vary widely in technology depending on the intended experiment. They all have in common that they can detect the presence of particles and usually their energy and trajectory. In any case, the sensor arrays used are usually large machines with hundreds or even thousands of parallel sensors, producing very high data rates. In order to depict events correctly, these sensors must be carefully synchronised in order to correlate their data correctly. This requires very accurate time stamps and precisely timed signals to trigger sensor acquisition on a facility wide level.

3.4.2 Particle Filters

Particle filters use a combination of an electric field followed by a magnetic field. The direction of the beam, electric and magnetic field are at right angles to each other. The whole arrangement acts as a band-pass for a particular particle energy and charge. Applying Eq. 3.1, the electric field is set to a strength that only particles above a certain energy are fast enough to pass through without being deflected into a hollow grounded electrode. Since the previous electrostatic filter already removed particles of undesired charge and below a certain kinetic energy, it can be assumed that the sorting solely depends on the mass/charge ratio. The following magnetic field curves the particle path, and is set so only particles with the desired mass/charge ratio make it further without being deflected into another catcher electrode. This is important for later acceleration stages, because particles outside the accepted energy window will most likely be lost in beam lines after further acceleration, causing unnecessary radiation. Linear accelerators and ion sources usually have a filter in their low energy stage.

Chopper If the filter is using a pulsed magnet, the whole arrangement is called a chopper. It is used to chop pieces out of a continuous stream of particles or particle bunches. The default behaviour is to deflect the beam into a dump, only when the pulsed magnet is activated, particles leave the chopper through a small slit. This default can be used as a safety measure if beam supply must be stopped immediately. In practice, the beam is often pre-bunched in an RF-cavity before the chopper, using the chopper only to neatly crop the bunch edges, wasting less beam.

3.4.3 Beam Diagnostic Equipment

Many different qualities of a particle beam inside an accelerator are measurable. There are transversal parameters, like the beam position relative to the centre of the tube or the beam profile. For heavy ions, this is usually measured with profile grids. These loom-like devices feature a grid of horizontal and vertical wires with a gas layer. The passing heavy ions ionise the gas around a wire and create a measurable electrical potential. By measuring the voltage on all wires, a two dimensional beam profile can be constructed. Longitudinal parameters are information about particle distribution along the ring orbit. This includes the number of particle bunches, the position of a bunch in the ring relative to the RF phase and longitudinal particle distribution inside a bunch. This is achieved by beam transformer coils placed at regular intervals around the ring. A bunch of charged particles passing through the middle of such a coil induce a measurable current, which signal strength and shape depends on the bunch length, speed and amount of particles. When the signals of all coils on the ring are correlated with the RF phase, an accurate picture of the bunch phase and the particle distribution along the orbit and can be deduced [55].

3.5 Control Systems for Particle Accelerators

For CS design, accelerators are a somewhat special field. They combine big geographical distances with high density of machines in a small space and the accompanying complex connections. They are radiation zones and feature the matching heavy, and often subterraneous architecture. Lastly, many components within a particle accelerator must run synchronised in order to function, the timing requirements grow with geographical distance and complexity of the setup. Large physics experiments like accelerators are, like telecommunication applications, among the few cases that can truly benefit from sub-nanosecond accuracy.

The following subsection describes the timing requirements at each point in an accelerator, starting from the source of a beam and work our way down to its target.

3.5.1 Timing

We shall regard a need for timing as a requirement for synchronisation or measurement accurate to $\leq 1\,\text{ms}$. All slower systems can record data and receive commands with network time distribution over standard computers.

General For an accelerator to work, it needs a high quality vacuum within its tubes. The vacuum system uses mechanical pumps, direct current (DC) ion getters and heating elements, which are too slow for requiring high accuracy timing. The same holds for the cryogenic system, as mechanical valves, pumps and heat exchange are slow.

Source Ion sources can run stand-alone on an RF signal (which is usually mains), they do not require timed commands or accurate timestamps. When the particles leave the ion source towards a LINAC stage, they pass an arrangement of particle filters. These basic filters run on DC and thus do not need timing. If particles are to enter a ring accelerator after the LINAC, the next station is a pre-bunching cavity. These RF cavities bunch up a continuous beam into a series of concentrated particle clouds. To synchronise their RF to others in the system, timed commands can be necessary.

LINAC Next in line is a chopper, a pulsed particle filter. It is used as a safety measure (dumping the beam) and to clip bunches. Because it is pulsed, it needs timed commands. The following LINAC stage interestingly does not need timed commands, as it solely operates on one RF frequency, phase is given by the gaps between electrodes. Conventionally mains is used, because a low frequency, high amplitude RF voltage is needed. However, because other RF is required to stay in phase, timestamps of mains' zero crossing point can be necessary (mains frequency changes over time). Beam diagnostic in this section can benefit from timestamps, but will not require highest accuracy.

Ring Accelerators The entry and exit points of ring accelerators contain fast kicker magnets, which need timed commands. For transfers between rings, they also need accurately timestamped measurements from beam instrumentation and RF. Ring RF must be phase synchronised and therefore requires timing. The corner dipole magnets can run on DC if the energy level of the beam stays constant, but need timing to synchronise their current ramps, the same holds for focusing magnets (quadru- and sextupol).

Detectors Detectors produce very high data volumes at high sample rates. They usually are big and modular machines, which benefit from timed commands synchronising sampling start in their modules as well as from accurate epoch timestamps for data buffers.

Table 3.1 summarises all device classes in accelerators, which includes the corresponding controllers. Their properties are listed in terms of their need for high accuracy timed commands and/or need for timestamps.

Device	Timed Commands	Timestamps
Ion Sources	no	no
LINAC Tanks	no	yes
DC Particle Filter	no	no
Chopper	yes	no
RF Cavities	yes	no
Dipole Magnets	yes	no
Quadrupole Magnets	yes	no
Sextupol Magnet	yes	no
Kicker Magnets	yes	yes
Beam Current Transformer	no	yes
Beam Profile Grid	no	yes
Vacuum System	no	no
Cryo System	no	no
Detectors	yes	yes

TABLE 3.1: Particle Accelerator Devices and Timing Requirements

The corresponding timing constraints for the planned FAIR accelerator are discussed in detail in the case study in chapter 4.

3.5.2 Radiation and Shielding

Sources High energy facilities contain areas with strong levels of radiation. Especially Synchrotron rings can emit strong levels of X-rays at all corner points and most strongly at their exits when charged particles are kicked out. Heavy ion accelerators like GSI and FAIR produce particle (neutron) radiation when beams are misguided. Hit components become radioactive themselves, as part of their nuclei are ripped off by the high energy ions, spreading more particles.

All ionising radiation, that is, radiation powerful enough to remove parts of an atom, has a deleterious effect on electronics. In low dosage, it can flip bits in RAM cells, corrupting memory content. On higher dosage, ICs can be permanently damaged, both from induced eddy currents and direct hits on their silicon structures. Furthermore, continuous exposure to strong radiation increases the attenuation of fibreoptic cables until their signal to noise ratio eventually becomes unsuitable for networks.

Countermeasures Depending on the radiation, the required shielding to counter the effect ranges from thin metal foils all the way to several metres of heavy metal infused concrete and earth or water tanks. Direct exposure of CS equipment to radiation is to be avoided, which is usually achieved by placing most electronics elsewhere. All connections are then routed through so called labyrinths in the shielding, which means a winding path, blocking line of sight and thus radiation. If it cannot be avoided, there are measures to detect/mitigate the effects, like error coding and redundancy.

Implications for Timing and Control Both radiation and shielding prevent physical access, making deployment, connection and maintenance difficult. Furthermore, shielding denies signal reception, using satellite disciplined timing receivers is therefore impossible in most parts of the facility. To still get the benefit of atomic clocks' long term stability, a feasible alternative is to have one or more satellite disciplined receivers in places with good reception, and distribute time from there over wire or fibre. This makes it a good scenario for employing network based time distribution services.

3.5.3 Summary

The underlying principles of particle accelerators presented in this chapter implied, together with the properties of individual sub-components as outlined in section 3.5, that future proof CSs for particle accelerators are a highly demanding application. Apart from the telecommunication industry, large physics experiments can truly benefit from the synchronism of high end time distribution system like WR.

Part II

Problem Analysis

Chapter 4

FAIR Accelerator Case Study

4.1 Overview

This chapter contains the analysis of FAIR's requirements of a CS. The chapter is based in parts on the paper "The FAIR Timing Master: A Discussion of Performance Requirements and Architectures for a High-precision Timing System" [56]. Timing and CS requirements are collected from previous experience with particle accelerators, mainly from GSI and CERN.

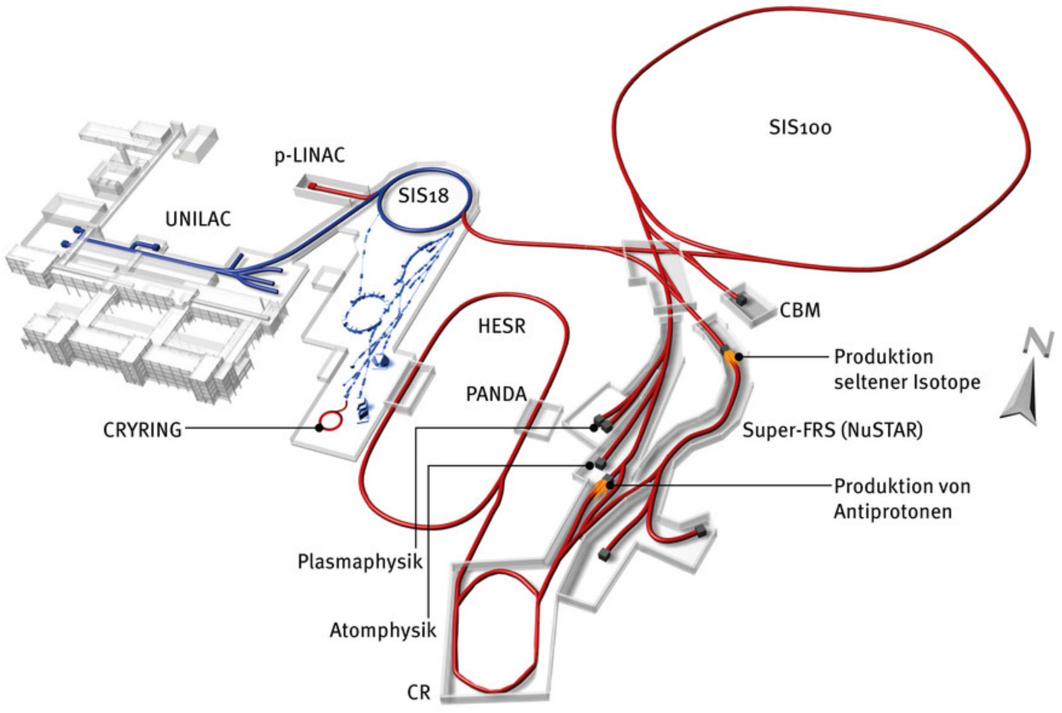


FIGURE 4.1: GSI accelerator and future FAIR extension [57]
Existing GSI facility in blue, planned FAIR machines in red

Founded in 1969, GSI runs a worldwide unique accelerator facility. The focus lies on storage ring experiments with heavy ions and a wide diversity of experiments. GSI distinguishes itself by the large variety of different elements and beam properties it can offer at high energies and the additional capability to create exotic beams of radioactive isotopes. As a counterexample, CERN is focused on generating proton beams at extreme particle energies while experiments with heavy elements are few.

The research program at GSI covers a broad range of topics: nuclear, particle and plasma physics as well as biophysics, material and medical research. GSI is best known for the development of a novel, state-of-the-art type of tumor therapy [58] and the discovery of several new super-heavy elements.

The planned FAIR extension is a new particle accelerator complex, designed for the research of antiprotons and heavy ions (see chapter 3). It has been under construction since 2008 and it is to go into operation in 2018, reaching its final stage of expansion in 2025. It is being built with the cooperation of an international community of countries and scientists on, and next to, the premises of GSI and is to provide a work environment for 3000 international scientists. FAIR will extend the existing accelerators of GSI, using them as its pre-acceleration and injector stages. As figure 4.1 shows, FAIR (red) will expand the existing GSI facility (blue) by adding two linear accelerators (p-LINAC, CRYRING injector) and four ring accelerators (clockwise, starting at 12 o'clock: SIS100, CR, HESR, CRYRING). A fifth ring, SIS300 (not shown), is planned to be built congruently to SIS100, forming a collider.

The following is a short introduction to different CS philosophies and a summary of GSI's legacy CS is given. Afterwards, the concept and requirements for FAIR experiments will be discussed.

4.2 Control Systems

Experiments on particle accelerators are the result of a highly complex interplay between various machines. Each action must be executed at a predefined time, as part of an overall master plan. To conduct experiments with particle beams, magnets, RF cavities, beam diagnostics and data acquisition need a close coordination. The associated production chains for a beam, from ion source to target(s), are an involved scheduling problem, with many interdependences and multiple paths to consider.

The number of planned experiments at FAIR is much higher than at GSI, more beams are produced in parallel and the number of machines needing coordination strongly increases. The larger size of the FAIR facility leads to higher signal propagation times but, more importantly, to a higher divergence between them. The implication of these changes is that a simple reuse of existing GSI CS technology will not suffice for FAIR.

4.2.1 GSI's Control System History

The following paragraph summarises GSI's CS design between 1987 and 2005. GSI's CS is a hybrid of VME based computers, connected via Ethernet, and special hardware equipment controllers, dispatching synchronised commands over MIL bus connections.

The MIL-1553 standard is a field bus system, which was developed by the US military 1973. It is a robust protocol, designed for avionic and spacecraft CS applications. It focuses on determinism and separation of bus and terminal systems, the clock is encoded in the data stream using Manchester encoding. A subset of the standard was implemented for GSI's control system at the specified maximum rating of 1 Mbit/s.

GSI's control system aimed to carry out actions accurate to 1 µs at hardware level and accurate to 10 µs at realtime software level [59], [60]. It nevertheless does not feature a universal time distribution system, all actions were synchronised to accelerator cycles. Only the pulse centres ("Pulszentrale") had an absolute time, receivers, in general, did not. If absolute time was required, pulse centres distributed periodic timestamps over MIL and a static correction for propagation time was used. Physics calculations were carried out on central servers, the resulting machine control data was delivered to the pulse centres as precompiled tables. Requests from experiments and Interlock behaviour made the pulse centres switch between tables.

10BASE2 Ethernet, MIL and analogue lines limited the total possible distance from the control to ≤ 200 m. The dimensions of the system were defined as "...2750 devices are controlled by 256 ECs in 41 VME crates" [61]. In the post 2001 refurbishment, hardware systems were left mostly as is, software systems were upgraded in places.

An extremely stable optical RF clock distribution system, the Bunchphase Timing System (BuTis), was introduced in 2005. Further details are provided under section 4.4.1.

4.2.2 Other Control Systems

Existing CS's at other particle accelerators were of interest to this case study, because their use-cases are similar to GSI. The following account lists the characteristics of three particle accelerator facilities as well as the technology and properties of their CS.

CERN, Switzerland The CERN accelerators provide the most important use-case in the current context because the initiative for the WR time distribution system originated there. Over the past decades, a multitude of collaborative projects between GSI and CERN were conducted, suggesting both common interests and use-cases.

As shown in figure 4.2, CERN features various ring and linear accelerators, the most renowned being the LHC, which is worlds largest and most powerful synchrotron with a circumference of 27 km. The control system consists of a variety of UNIX based servers, industrial PCs and VME computers running real time Linux variants and programmable logic controllers (PLC). Both RTOSs and PLCs can provide determinism in the high microsecond range [62], [63]. These systems are connected over Ethernet, Profibus and WorldFIP field bus. While Profibus was chosen for its simplicity and easy integration with PLCs, WorldFIP is used for deterministic operation down to 10 µs and consisting of an infrastructure of more than 40000 elements and 350 km cabling [64, p. 2]. CERN's central timing system provides UTC time with 25 ns resolution, which by application of a custom chip can be brought down to 1 ns. However, this system uses manually calibrated delay compensation, did not scale well and was an isolated application.

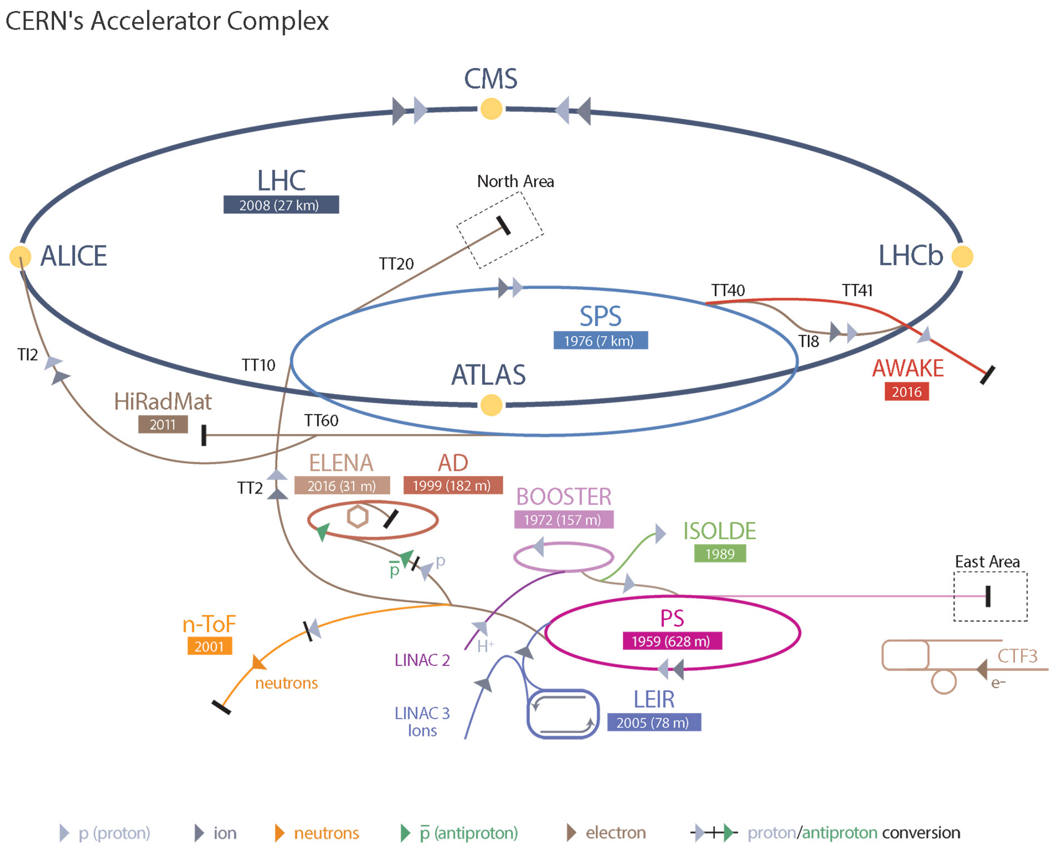


FIGURE 4.2: Overview of CERN Accelerator Complex [65].
Rings are shown by labels with year of
commissioning and circumference

Diamond Light Source, UK Diamond is a modern synchrotron light source near Didcot, UK. A light source generates extremely bright X-ray beams from synchrotron radiation, usually produced with electron beams. The control system employs VME, industrial computers and servers running the Experimental Physics and Industrial Control System (EPICS) framework [66] for soft realtime control and PLCs to implement deterministic control in the low millisecond range [62]. Local FPGA based systems are able to monitor and correct beam parameters deterministically down to $\approx 100\,\mu s$ [67].

DESY, Germany The Deutsches Elektronen Synchrotron (DESY) is located near Hamburg, Germany. DESY is currently best known for its use of electron lasers as light sources. Most CS properties are quite similar to Diamond, especially the use of VME and industrial computers running the EPICS framework and the use of PLCs for safety relevant control tasks. The level of achievable determinism is also matching in the same range. On the lower levels, various field buses such as CAN, Profibus and SEDAC, DESY's proprietary field bus, are employed. Fast local FPGA systems are also employed, a value of $90\,\mu s$ was stated in [67]. Like GSI's

BuTis system, DESY also needs an extremely accurate clock distribution system. The linear collider requires a phase stability of 1 ps [68]. A more stable design was proposed in 2006, offering a stability of 10 fs [69]. Like BuTis however, the system does not provide absolute time and cannot be used for generic control applications, its sole purpose is the synchronisation of RF cavities.

4.2.3 FAIR CS Use-Cases

The FAIR CS's main purpose is to provide trigger signals to machines, which is the same basic functionality the GSI CS supplied. These triggers serve to synchronise magnet ramps and RF frequency generation, the transfer of beams between machines and the correlation of acquired data.

Secondly, the FAIR facility is too large to collect data centrally without the ability to assign epochs to the sampling clocks. For this purpose, timestamps are assigned to recorded data. These timestamps are required for later correlation. The prerequisite is a time distribution scheme that can provide a global, absolute time, synchronised to a high quality reference. This not only enables global timestamps, but also facilitates trigger generation.

The last goal to achieve is the synchronous generation of sine wave signals. If the CS endpoints share both time and clock alignment, it is possible to synchronise local NCOs, generating phase synchronous waves, independent of the endpoint location. This is called a distributed Direct Digital Synthesis (DDS) and can be used for the RF system and for other applications, such as the implementation of a distributed oscilloscope [70].

4.2.4 Philosophy

GSI's former CS and the concept for FAIR are very different in nature. While GSI employed an event based system, the approach for FAIR is an alarm based design. In both concepts, data supply (such as current ramps) can be separated from control.

Events In an event based CS, commands are contained in so called events, which are messages in the system. Upon reception, receivers immediately execute the contained command. This is simple to set up, but the concept is dominated by transmission and processing delay. Connections need to be length matched and/or receivers need programmable delays to achieve synchronous execution of an event in multiple receivers.

Changes to topology require recalibration. In the worst case, adding one new receiver will require all other lines to be recalibrated to match the new receivers latency. This imposes a limit to line length and fan out, the calibration is also only valid within a small temperature range.

Alarms In an alarm based CS, all receivers share the same notion of absolute time. Messages contain a command and the time when it is to be executed. Upon reception, an endpoint locally sets an alarm, which will trigger the corresponding action exactly when due. Contrary to an event-based CS, arrival and execution time are decoupled, thus being conceptually independent of transmission delay. Provided the dispatch time has a lead greater than the sum of transmission and processing delay, commands will always be executed on time. This allows arbitrary connection lengths and almost unlimited fan out, which eliminates manual calibration per receiver.

4.3 Beam Concept

This section aims to provide an overview of the beam production concept proposed for the FAIR complex. FAIR is meant to run multiple experiments in parallel and provide them with different particle beams. The path of a beam from its source to its target is called a Beam Production Chain (BPC). It is a schedule that describes the allocation of resources such as beam guides and accelerator stages to the beam, details are described in its sub-components.

4.3.1 Parallel Operation

The "metro" plan in figure 4.3 gives a first insight into the many ways the FAIR complex can be used to provide the required beams to experiments. The parallel occurrence of several colours marks multiplexed paths, referring to beam lines and accelerators time-shared between multiple experiments. Multiplexing maximises available beam time, but is obviously an additional scheduling problem the CS must solve. Multiple BPCs can be grouped into patterns, which describe the multiplexed set of beams currently under production.

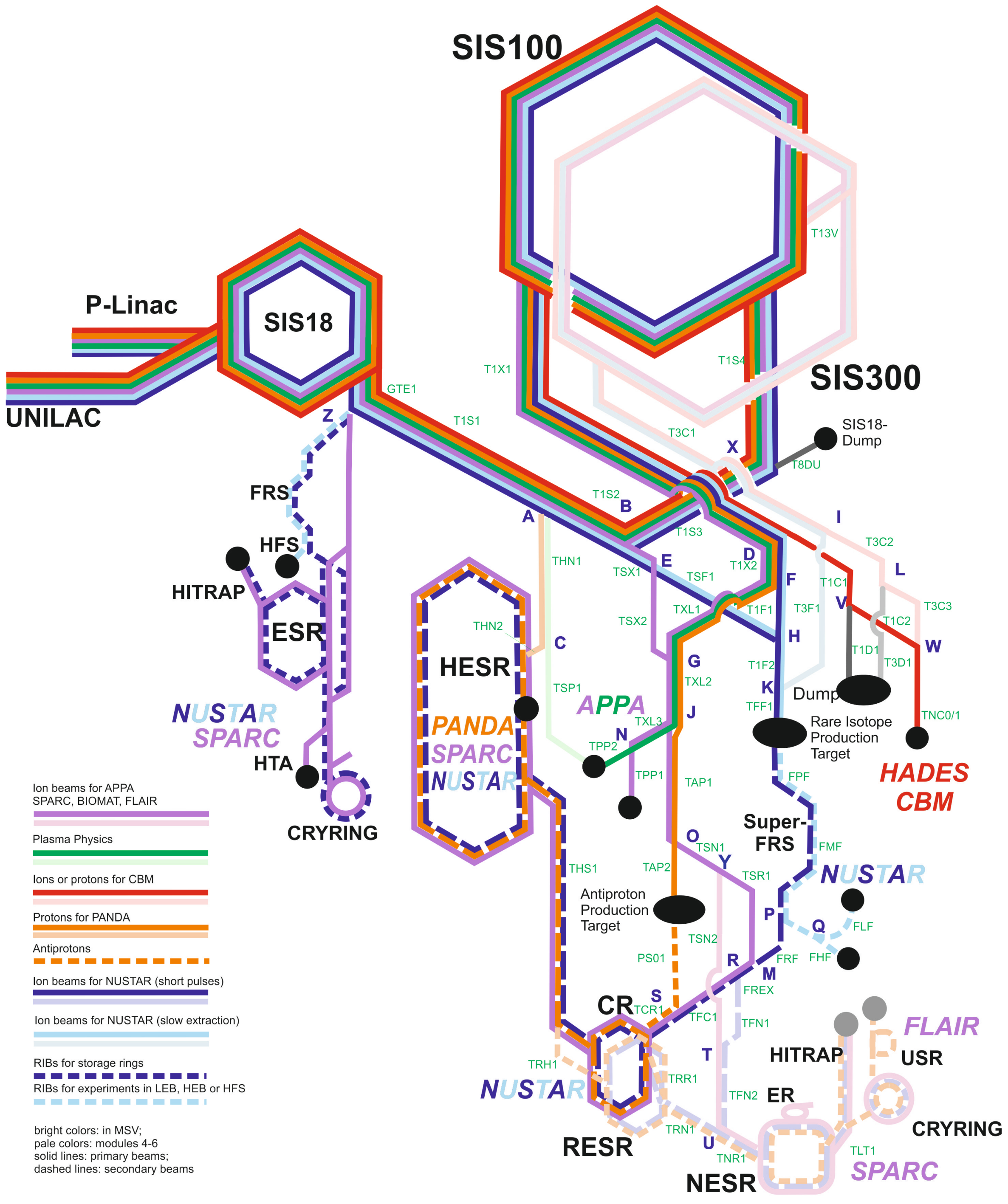


FIGURE 4.3: FAIR beam-line schematic [71]

4.3.2 Beam Production Chain vs Beam Process

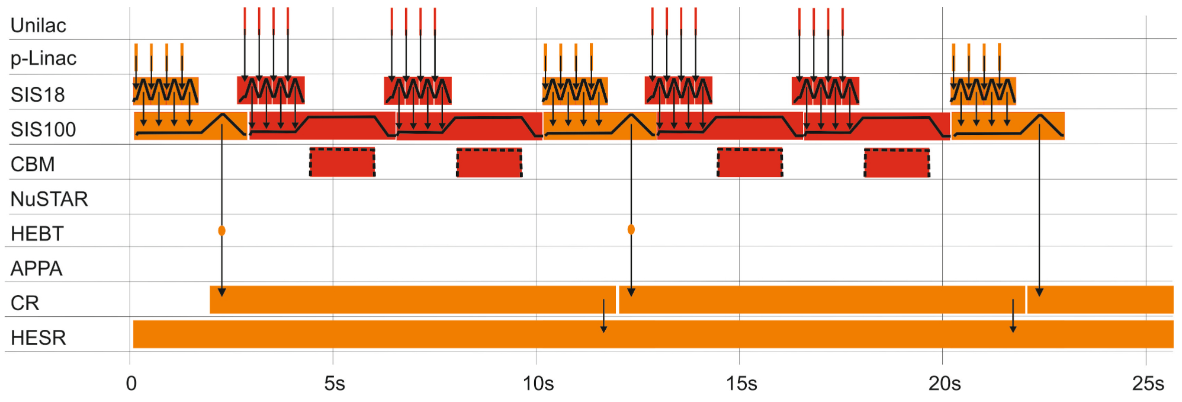


FIGURE 4.4: Abstract view of BPCs, complete Pattern [71]
Vertical Flow shows BPCs, horizontal shows Sequences

The whole set of coloured blocks in figure 4.4 shows a pattern. It consist of BPCs (vertical flow) corresponding to figure 4.3, using the same colour code. The horizontal rows are sequences, which describe a schedule of actions within one resource (linear accelerator, etc.). Sequences in themselves are a simple concept, merely a timeline for control message dispatch. They in turn are composed of beam processes (BP), the smallest indivisible unit (not shown in figure 4.4). They are simple series of control messages that cannot be interrupted. Their succession and possible alternatives can be quite complex however. It depends on external events, like interlocks and beam requests from experiments or other accelerators, as well as internal synchronisation events.

While it seems intuitive that patterns should be completely predictable and all required successions of machine commands can be computed beforehand, this is not the case. The CS has to react to both interlocks, such as components not being ready or failing, and beam requests. Both sources consist of multiple parallel inputs and the resulting combinatorial space is too large to compute static solutions. Instead, the CS must have partial alternatives prepared and dynamically choose between them at runtime. Figure 4.5 shows an example of patterns down to the level of BPs. Arrows show not only their succession, but also possible alternatives and loops. The similarity to a flow chart is no coincidence, as this representation is meant to be implemented as a control program.

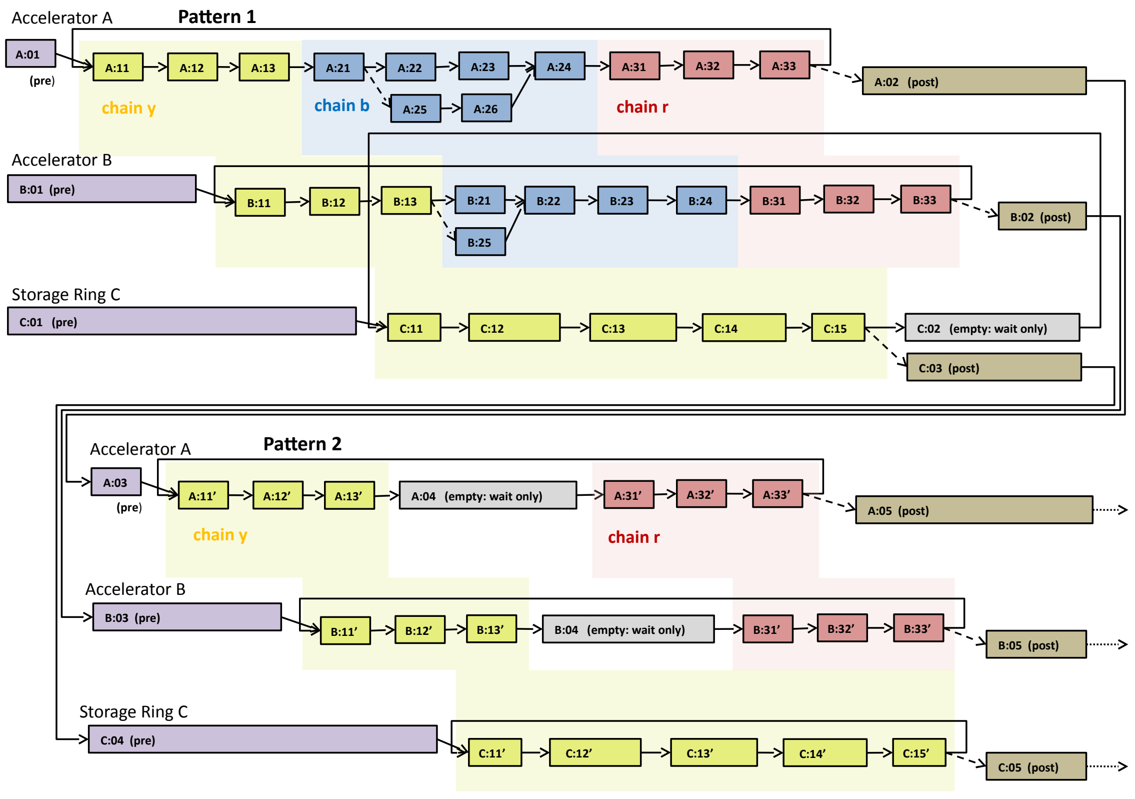


FIGURE 4.5: Machine Schedules for the Accelerator

4.4 Timing Constraints

4.4.1 Common Systems

The facility's timing constraints are to be deduced from machine properties and requirements from physical experiments.

Vacuum, Cryogenics and Mechanics As described in chapter 3 on page 33, the sub-components of these systems, like pumps, heating and motors moving probes or valves, are comparatively slow. They need timing only in the millisecond to second range and thus do not tighten timing constraints, although they are of interest as sources of system interlocks.

Magnets and Beam Diagnosis Most magnets are "slow" components. Except the kickers, which are responsible for beam extraction and injection, magnets need a timing accuracy of $\approx 1\,\mu s$, which is straightforward to achieve. The same holds for beam diagnostic sensors, as they usually need several cycles of the beam to gather data and additionally do not need to be more accurate than the large magnets. While the kicker magnets need to be timed with an accuracy in the lower nanosecond range, control is handled by a dedicated system (see subsection 4.4.2).

RF Systems Local systems synthesising the cavities' sine waves must be carefully phase aligned. BuTis, an optical RF clock distribution system featuring very high quality oscillators already exists at GSI. It provides a clock with 5 ns period and allows a phase shift in the low pico second range [72]. The clock stability lies in the range of 10 fs.

Most of GSI's systems require the maximum clock uncertainty (jitter) to be less than 200 ps [72] in order to maintain proper phase relation between RF cavities, which can be dealt with by the CS. However, the CS must be coordinated with BuTis to make use of its superior clock stability for the remaining systems. In many places within the facility, the BuTis 200 MHz base clock must be available, but most machines do not require the actual accuracy of BuTis. Because of the high cost of BuTis receivers, a distribution of this signal by the CS is desired for these cases. In addition, often a specific period of the 200 MHz clock must be marked for reference. Both requires a 5 ns timing resolution [73].

Detectors Quite often, particle detectors need timestamps to assign epochs to their sample clocks. The detector systems for FAIR come in two varieties, triggered and free running. While a triggered system starts a measurement only at the time its components receive the trigger signal, a free running detector measures every single occurrence if its maximum sample rate is not exceeded. It is necessary for many experiments to correlate data from triggered and free running systems. A sub-nanosecond trueness and 8 ns resolution already allows correlation for measurement rates in the 10 MHz range, but a resolution of 1 ns would be preferable for future use-cases.

While BuTis is extremely stable in short and long term, it can show a measured drift of up to 100 ps over the course of a day, a WR solution is preferred. To eliminate calibration efforts, a maximum standard deviation in the range of 10 ps is required [74], although it could originally not be met by WR. It has subsequently been improved to meet the required precision [75].

4.4.2 Bunch To Bucket Transfers

A B2B is the transfer of particles from one synchrotron ring to another and one of the most involved operations the CS has to coordinate. The theory is already presented in some detail in chapter 3.3.3 on page 37, "Requirements of Synchrotron Machines". Figure 4.6 shows the schematic distribution of buckets and bunches in SIS-18 and SIS-100 synchrotrons, the goal being the transfer of a bunch from one ring to the other. This requires careful coordination between both ring RF systems, beam instrumentation, extraction and injection kicker magnets. The actual transfer is autonomously controlled by the RF system,

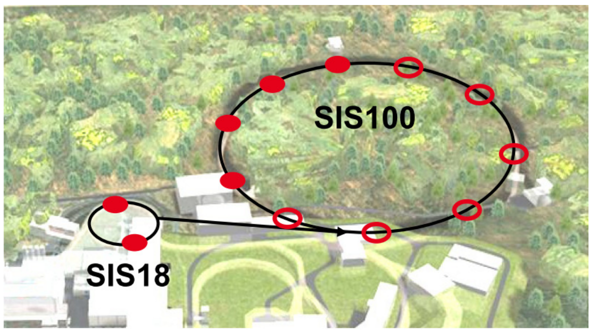


FIGURE 4.6: Bunch-to-Bucket-Transfer [76]. Bunches are filled, Buckets are empty circles

but the CS must control all dependent devices. The required synchronous RF reference clocks are provided by the BuTis system, because the CS cannot provide the needed low picosecond trueness and phase resolution.

To start a transfer, RF regulation loops are deactivated and the system will stay sufficiently stable in this mode for more than $10\,\mathrm{ms}$ [77, p. 100]. During this time, bunch and bucket positions must be measured, data exchanged, the transfer window calculated, ring phase shifted and finally the kickers must be commanded to extract and inject the beam. This leaves $\approx 1\,\mathrm{ms}$ total time for command distribution. Because this is a round-trip time, the maximum allowable message generation and transmission time is $500\,\mu\mathrm{s}$.

4.4.3 External Requests

Interlocks Their handling is divided into two distinct layers. The first is the machine and personnel protection layer, which is placed close to the machines where the error might occur. They are dedicated hardware, able to react in the low microsecond range, handling critical actions like beam dumps and emergency shutdowns of power supplies. This is too fast for the CS to react, but it needs to handle the consequential errors, e.g. a linear accelerator is not supposed to deliver still more beam after a dump in the following synchrotron has occurred. The reaction time for interlocks is in the range of $\leq 100\,\mathrm{ms}$, but this involves all necessary processing. Because many of these signals are safety relevant, their acquisition is handled by Programmable Logic Controllers (PLC). While being highly reliable and deterministic, these systems are relatively slow and can take several tens of milliseconds to process and forward an interlock signal [62, p. 28]. Afterwards, the signals need to be aggregated and checked against dependencies provided by the physics framework, e.g. which beam transfer is influenced by error signal X. This is processed in software on computers running a realtime OS. According to current GSI planning, all other processing times leave a window of $10\,\mathrm{ms}$ for the CS to react.

Beam Requests Target setups and downstream accelerators can request particle beams from the CS. However, requests are not regarded as time critical. A best effort service with a reaction time in the same range as the interlocks, i.e. $10\,\mathrm{ms}$ is aimed for.

4.4.4 Determinism and Transmission time

Control systems are realtime systems, of which there are two basic types. Soft realtime means the value of results degrade if they are late. Hard realtime means that late results are worthless, with possible dire consequences. Because FAIR's CS is hard realtime, the primary goal is to ensure commands (results) are never late. The central attribute to achieve is therefore deterministic behaviour, providing upper bounds on how long any CS action can take. The delivery time of messages to endpoints is one of the most important factors.

Table 4.1 contains preliminary estimates for message delay over a WR based network. The data is based on the figures provided by Lipiński et al. [78, p. 15] and is calculated specifically for the FAIR scenario, assuming five layers of switches, 2 km total link length and a maximum control message size of 1500 B. The example shows the necessary time for the transport of command messages over a WR network, including the time for FEC [78]–[80].

Name	**Value** Min	**Value** Max
Eth Frame TX Delay	0	$(13 + 3 \cdot 9)$µs
Switch Routing Delay	0 µs	13 µs
Link Delay	2 km · 5 µs/km	2 km · 5 µs/km
Eth Frame RX delay	3 µs	9 µs
FEC Encoding	2 µs	2 µs
FEC Decoding	2 µs	2 µs
Sum	17 µs	128 µs

TABLE 4.1: Delay estimation for control messages
Case for 5 Switches, 2 km link length, Message Size 500 and 1500 B

4.5 Reliability

4.5.1 Availability and Robustness

Availability At present, the CS for FAIR has been specified to have a "high availability", though a specific figure for uptime was never postulated. A reverse approach, from an analysis of CS module prototypes for their mean time to failure (MTTF) or availability figures was not feasible either. Figures are available for the WR network, but the MTTF figures are not calculated for WR switches, but were in fact taken from switches manufactured by Cisco [80]. While there is an extensive analysis available for the hardware of the main FAIR timing receivers, none exists for their firmware and software. This analysis would be a difficult prospect in any case, because many components are still under development and subject to frequent changes. No updated values for any of the above systems were available until the submission of this thesis in the beginning of 2017.

Robustness The most important factor for flawless operation of the CS is the occurrence of transmission errors. The estimated bit error rates for fibre links showed a probable occurrence of frame loss of 3.94×10^{-7}, which would, assuming FAIR's expected traffic, be roughly equal to 12.4×10^{4} lost frames per year [79]. When gathering requirements for the FAIR CS's robustness, there was only a semi-official statement available, quoting a management decision for "less than one lost timing message per year" [81]. Despite being elevated in subsequent publications to an absolute requirement for a frame loss rate of 3.17×10^{-12} at GSI [80], the value in the quote was chosen arbitrarily should therefore not be treated as a hard requirement.

Best Effort Because of the insufficient data, no numerical values for robustness and availability were included in this analysis of the CS requirements. Further analyses to obtain more accurate requirements have been postponed.

The CS is *not* responsible for preventing harm or damage caused by errors during the operation of the facility. Machine and personnel protection is handled by dedicated hardware systems at the controlled machines. In case of personnel protection, these systems also have to undergo an official certification process. For this reason, the lack of specification for availability and robustness of the FAIR CS was considered tolerable by the responsible executives. All measures taken for fault tolerance listed under 4.5.2 serve to increase beam time and ease maintenance (see chapter 3.5.2), but the CS only has to provide a best effort service.

4.5.2 Fault Tolerance

Due to the stringent timing constraints of the CS, there is no time available for retransmission of corrupted or lost frames, yet the CS requires deterministic message delivery for operation. This implies that packet loss must be avoided in the first place.

Preventing Packet Loss Protecting an Ethernet based system against packet loss is not trivial, because while the frame checksum will detect most errors, no corrections are made. The standard behaviour according to IEEE 802.3 Ethernet [82] is to discard a faulty packet, which is called a Packet Erasure Channel (PEC). To prevent packet loss, a combination of two layered Forward Error Correction (FEC) algorithms has been proposed [80], [78, p. 23].

Between Switches The concept preserves the compatibility to IEEE 802.3. Between WR switches, faulty frames will be dropped, forming a PEC. Codes suitable for PEC encode k frames into n frames, with $k < n$. The reception of any subset of k frames at a timing receiver allow the reconstruction of the original frames. Details on concrete algorithms are presented in [78, p. 24]. Furthermore, redundant links were suggested between WR switches to allow continuation of operation in case of link loss and to mitigate packet loss.

Between Switch and Receiver However, between a WR switch and a timing receiver, there is no need to drop packets containing errors. The reason is that neither compatibility with IEEE 802.3 has to be maintained, nor is the header information required any more, as this is the last link in the chain. Only faulty bits pose a problem. This forms a Binary Erasure Channel (BEC). A variety of different encoding schemes is suggested, the final choice depends on the number of expected bit errors per frame [78, p. 25].

In the end of 2016, no implementation was available for the testbeds used in this thesis. The FEC will therefore be treated as a black box, the bandwidth increase caused by the FEC scheme is assumed to be at a factor of 3 to 4 [83], [78, p. 68].

4.6 Summary of FAIR CS Requirements

The obtained requirements are not complete and in some cases quite loose. It has to be taken into account though that the FAIR accelerator was to be designed for a lifespan of 25 to 30 years. The CS requirements hinge on those of the experiments, which are only loosely set for the whole lifespan. This is because each generation of experiment setups changes with the obtained results of previous experiments and progress in the field as a whole. Experimenters therefore tended to be very noncommital when asked about their CS needs. Since research and development for the FAIR CS had to proceed regardless, this meant a necessity for pessimistic extrapolation from known usecases and massive overengineering to make the resulting system as future-proof as possible.

The gathered information was used as the design criteria for this thesis to outline and investigate approaches for sub-modules of the FAIR CS. The sub-modules consist of the CS master unit (Data Master) and a protocol for deterministic data transmission between SoC systems. Choosing WR was a management decision made before the start of this work, so was the choice to base the CS on GbE over optical fibre. However, the case study's results do not contradict the validity of this decision, as all requirements of the FAIR accelerator can be fulfilled by a WR based CS.

The goals of the new CS design are:

- FAIR Concept
  - Independence of the CS from geographic distance, fanout and temperature drift
  - High parallelism i.e. supplying multiple experiments with different beams at the same time
  - Highly optimised utilisation of the facility
- CS Concept
  - Separation of control and data supply
  - Absolute Time
  - Alarm based CS
  - Provide Triggers, Clock Signals and Timestamps
  - All Commands available to all Endpoints
- Scale

– Endpoints	≥ 2000
– High Robustness	*no value available*
– High Availability	*no value available*
– CS Reaction time	$\leq 10\,\text{ms}$
– Node Distance	1 to 2000 m
– Net Command Bandwidth	$\geq 100\,\text{Mbit/s}$

- WR Time Distribution
  - Time Resolution 1 to 5 ns
  - Trueness to Ref. (Mean) < 1 ns
  - Precision (Std. Dev.) ≤ 10 ps
  - Satellite Disciplined Time Reference,
    24h stability 1×10^{-10} to 1×10^{-12} ADEV
- Communication
  - Standardised Field Bus
  - Broadcast Command Stream
  - Deterministic
  - Protection against Packet Loss and Bit Errors (FEC)
  - Link Length 1 to 2000 m
  - Maximum Latency ≤ 500 µs
- Master Unit
  - Synchronised to Time Reference
  - Generate Command Stream from Beam Production Schedule
  - Serve Interlocks and Requests at Runtime
  - Guarantee timely Command Delivery
- Receiver Unit
  - Synchronised to Time Reference
  - Filter Command Stream
  - Generate Triggers
  - Issue Timestamps
  - Distributed Frequency Synthesis

Chapter 5

Technology Survey

5.1 Overview

The evaluation of the case study in order to choose a hardware platform for the DM produced mixed results. While it did become clear that some of the timing constraints are very tight, pointing in the direction of a hardware solution, the analysed use-cases also require a high degree of flexibility from the system, which indicates a software solution. In this chapter, the limits of hardware and software solutions are discussed and evaluated in context of the case study's result.

5.2 CPUs

With increasing complexity and number of processes, keeping code execution and synchronisation deterministic becomes ever more challenging. Fast modern CPUs, especially multi-core architectures, offer vast processing power. Their drawback is that they require a managing OS to unlock most of their advanced features. These have worst case service latencies in the low millisecond range and do not really guarantee service at all, making them unusable for the DM architecture.

This issue can be mitigated by using a specialised RTOS. These are deterministic and can offer reaction times in the low microseconds range. Most are made for Micro-Controllers (MCU). The reason why very few RTOSs target modern, multi-core, out-of-order-CPUs is that a delay bound is extremely difficult to obtain due to all the optimisations they feature to speed up average processing time. While there are general purpose RTOSs, they are dubious candidates for a CS like the FAIR case study with a time resolution 1×10^{-9} s. Deterministic just means having an upper latency bound, it does not provide any information about the tightness of the bound. General purpose RTOSs work by deactivating some hardware acceleration, assigning quite loose bounds to the rest and implement strict scheduling policies in order to give guarantees. Naturally, this does not lead to the best of reaction times. We will therefore focus on RTOSs targeting MCUs.

Table 5.1 shows the latency measurements for a PowerPC 604 CPU (300MHz) on a MVME2306 board [84]. The discussed timing constraints could possibly be achieved with an MCU running an RTOS system, but as shown in table 5.1, jitter is in the range of several microseconds. There are also strong peaks under load (which would be common, since the RTOS has to handle all machines and IO sources in the system) which go as high as 200μs. We must also bear in mind that a network transmission time between 80 and 200μs must be added to the total reaction time, restricting the use even further. This leads to the assumption that an MCU with RTOS would probably be capable of running the required number of machine agents, but as figures show, could hardly cope with timing requirements for IO service requests.

	Interrupt Latency		**Context Switching**	
	max	**avg ±**	**max**	**avg ±**
Idle System				
RTL	13.5	(1.7 ± 0.2)	33.1	(8.7 ± 0.5)
RTEMS1	14.9	(1.3 ± 0.1)	16.9	(2.3 ± 0.1)
RTEMS	15.1	(1.3 ± 0.1)	16.4	(2.2 ± 0.1)
vxWorks	13.1	(2.0 ± 0.2)	19.0	(3.1 ± 0.3)
Loaded System				
RTL	196.8	(2.1 ± 3.3)	193.9	(11.2 ± 4.5)
RTEMS1	19.2	(2.4 ± 1.7)	213.0	(10.4 ± 12.7)
RTEMS	20.5	(2.9 ± 1.8)	51.3	(3.7 ± 2.0)
vxWorks	25.2	(2.9 ± 1.5)	38.8	(9.5 ± 3.2)

TABLE 5.1: RTOS latency measurement results [84]
Times in μs

The figures used in the discussion in [56] are from 2001 [84], so their absolute values are not a criterion for exclusion today. However, they attest to a basic trend which has not changed over the years, as newer studies from 2009 and 2011 show: RTOS systems tend to behave very differently under high load, the bounds they provide, are loose [63], [85, p. 308]. The IO bottleneck, resource arbitration and scheduling overhead pose a severe hindrance to satisfying the performance requirements of the FAIR case study. In view of the time available to react, there is a strong indication that a pure software solution on an RTOS will not be able to satisfy all requirements. The situation would be worse if a high end multi-core processor is involved. Standard OSs are completely out of the question, as their schedulers are not meant to provide sub-millisecond reaction times or any kind of determinism.

5.3 Hardware

Since the CS is about timely generation and distribution of messages, a digital signal processor (DSP) in combination with a network integrated controller (NIC) might do the job. Pure chip solutions are very fast and deterministic. But as inquiries with silicon manufacturers like Texas Instruments, Intel and several others confirmed, there is no off-the-shelf that available which offers the majority, if not all of the requested features. While this was to be expected, there are a lot of companies who offer application specific integrated circuits (ASIC). These chips are bespoke, customised for the job they were designed for and do it perfectly, with no workarounds necessary. A pure custom hardware solution such as this would satisfy all of the given timing constraints and provide all the requested functionality.

There is one major problem though, which has little to do with hardware capabilities and much with development flow of the FAIR project. Requirements are coming from a large number of different people from different specialist fields with no coordination, but are drip fed into the specifications. This is not overly surprising for a system that is meant to operate for the next 25 years. People are careful what they wish for and feature requests tend to be iterative. This is especially the case in large physics experiments, where the boundaries of knowledge and therefore expectations from a new facility, are expanded by the continuous work of ones own and other institutions in the meantime.

So while such ASICs would be perfect for the job, the design requires complete feature lists and the final product offers no flexibility. Even within the first ten years, it is inconceivable that one chip would be capable of meeting the evolving requirements. Instead several generations would be necessary, separated by long development cycles and not inconsiderable amounts of money. The task of implementing future changes and expansions would be hard at best and therefore discourages such a solution if there is an alternative.

When thinking about the design process of ASICs, one element is prominent. ASICs are made by describing the chips logic in a hardware description language (HDL). These are in turn optimised and synthesised into gate logic plans, which can be constructed in silicon. There is another technology which has a similar design process, which would allow the length of development cycles to be reduced and cut production cost to a bare minimum: Programmable Hardware.

5.4 Programmable Hardware

5.4.1 Introduction

Programmable hardware has existed for several generations now. They emerged from standard, digital circuits with a fixed logic when people tried to generalise circuit design in order to reduce fabrication effort. The basic concept was simple: using Boolean canonical logic like disjunctive (DNF) or conjunctive (CNF) normal forms [86][56-61]. Any logic table can be described in such a normal form. Table 5.2 shows an example, expressed as a DNF. Column elements are connected by logic *and*, whole rows by logic *or*. The implemented logic function solely depends on whether individual inputs are in positive or negative (inverted) logic. This means that a large number of different logic applications could be implemented using the same basic logic gate constructs.

A	**B**
0	1
1	0

TABLE 5.2: Simple Example of a Logic table
In DNF: *not*(**A**) *and* **B** *or* **A** *and not*(**B**)

This was first postulated in the late 1970s as Programmable Array Logic (PAL) or Programmable Logic Arrays (PLA) [86][161-165]. Both are implementing sums (*or*) of product terms (*and*) in a configurable matrix [30][764-768]. The first versions used one-time programmable input choices in the form of fuses. Using a higher voltage for programming, such a fuse could be tripped, permanently severing the connection to either the positive or negative signal input. In the next generations of chips, the configuration could be set more than once. Chips could be erased, some even by such exotic mechanisms like shining UV light directly on the chip's silicon structures.

Current implementations of programmable logic use a look-up table variant of a DNF, with 8 or more inputs. Multiplexers are used to choose the inputs, RAM cells store the bits to control the multiplexer selection. Apart from implementation details like these, the basic principle of configurable logic stayed the same for more than 35 years.

5.4.2 Advanced Capabilities

The two commonly used chip types today are Complex Programmable Logic Devices (CPLD) and Field Programmable Gate Arrays (FPGA). The main difference lies in the complexity of employed silicon structures. CPLDs are made for fast, simple interconnection of many IOs and keep their programming internally. FPGAs are slightly slower and additionally require an extra flash chip to contain their program. In exchange, FPGAs feature far more complex silicon blocks, like full adders in each macro cell, fast communication blocks (as used for Gbit Ethernet and above) and even DSPs to speed up multiplication and division. In addition, modern FPGAs contain not only logic cells, but RAM cells as well.

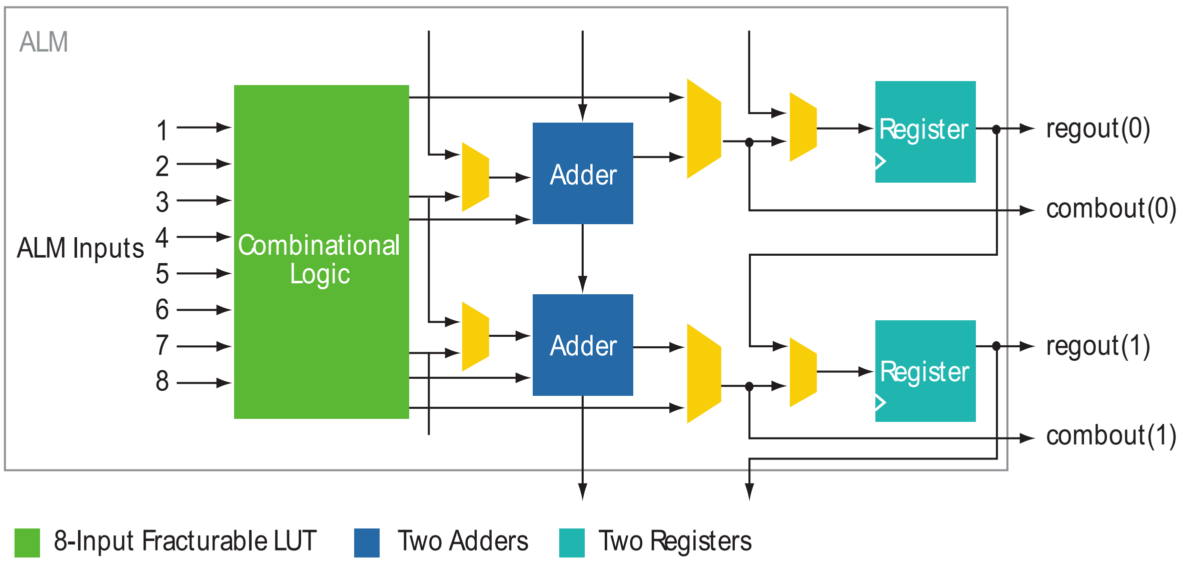


FIGURE 5.1: Block Diagram of an FPGA Macro Cell [87]

The lowest logic groups in FPGAs implement some form of product terms. To enhance performance, they are grouped with more specialised silicon. Figure 5.1 illustrates how full adders are added to the LUTs, followed by optional registers. Registers are necessary to form synchronous designs, as only the signal time between register stages needs to be bound, not between source and destination. The top and bottom of figure 5.1 also shows the carry lines leading in out of the logic block. These can optionally be connected to neighbouring logic cells, forming wider ripple-carry adders.

The omnipresence of registers in standard logic cells makes the development of large synchronous designs possible, ,and, in the newest generations of FPGAs, the trend goes to inserting even more optional registers directly into the interconnect logic. Current FPGAs also tend to offer a large variety of specialised silicon controllers, as illustrated in figure 5.2. The depicted FPGA contains, among other peripherals, a pure hardware implementation of a PCI Express bus controller ("PCIe Hard IP Blocks"). Most importantly among their specialised cores, FPGAs also feature analogue circuits, digitally controlled, fractional PLLs (see Phase Locked Loops, 2.4.2). These are used to generate almost any arbitrary fraction or multiple of input clocks, they can add programmable phase offset and distribute the clock signals inside the FPGA or to output pins.

The clock frequencies, at which designs are run, are scalable and largely depend on the length of combinatorial logic paths between registers; the minimum clock period is determined by the maximum signal propagation delay between registers. If the system is to be synchronous, all data has to arrive at the next register stage, poetically speaking, before the clock strikes. The shorter and balanced logic paths are, the faster the circuit can be clocked.

Performance Today FPGAs are able to mimic almost every conceivable digital circuit, ranging from a simple counter to a full blown embedded CPU. Their major advantage is their high degree of flexibility. State of the art FPGAs (2016) manage clock frequencies of up to 1 GHz core performance. This is the maximum frequency at which logic cells can be clocked and is not to be confused with design performance, which is limited by parameters such as logic depth and always lower than core performance. The number of logic cells varies for different models and are not totally comparable, as their computational performance depends on the number of inputs and whether they can be split to contain independent functions. For cutting edge FPGAs, the number logic cells ranges around 5-6 million, RAM cells range up to about 30 MB [88]. These figures might seem quite low from the PC perspective, where systems with more than 16 GB memory are common. But the charm of FPGA RAM cells lies in parallel control: it can be split to be assigned to dozens of different logic cores and natively supports full parallel dual access with single cycle latency. This provides excellent determinism and eliminates the need for caches on CPUs. Current FPGAs also feature dedicated high speed transceivers, up to 30 Gbit/s per connection. They are therefore well suited for network applications.

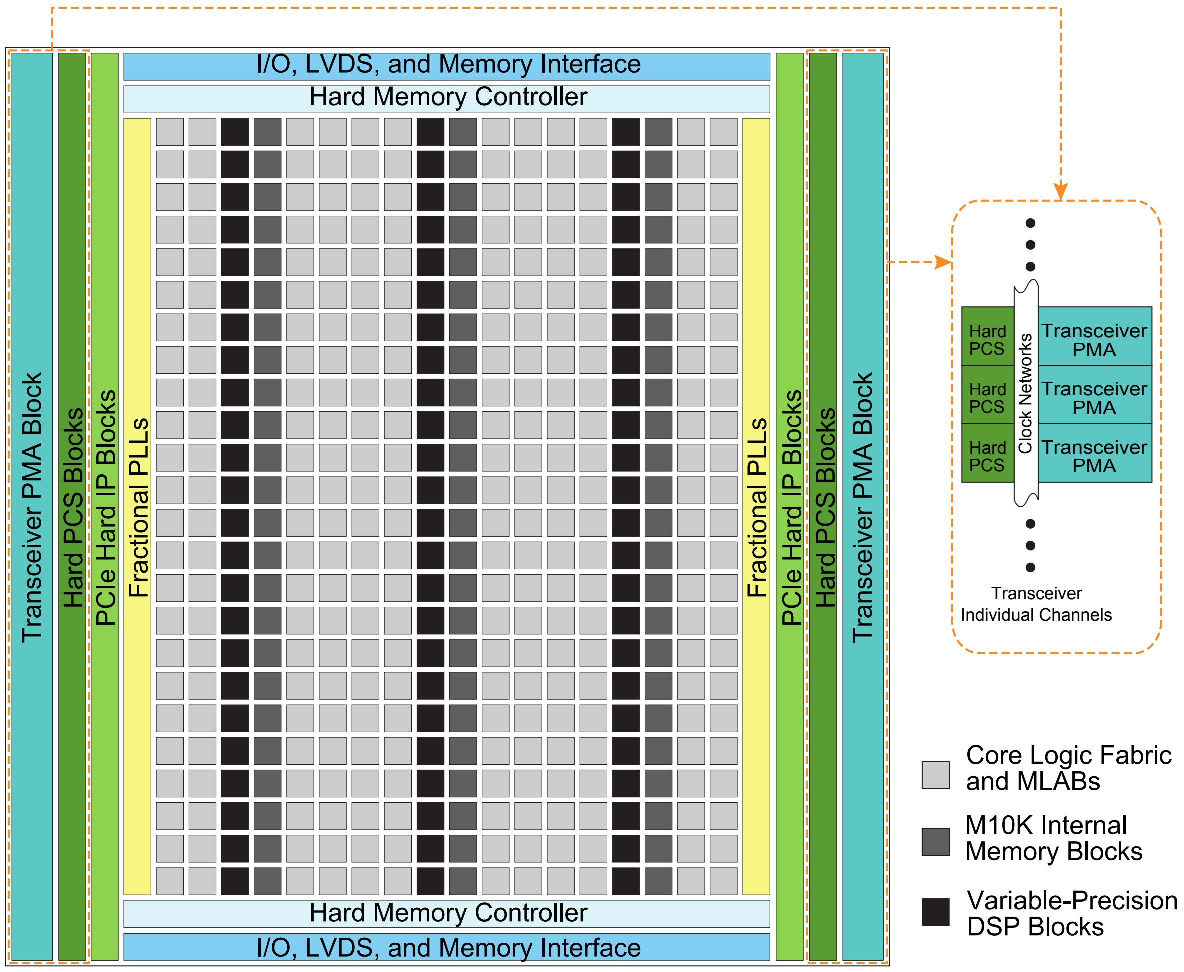

I/O, LVDS, and Memory Interface
Hard Memory Controller
Transceiver PMA Block
Hard PCS Blocks
PCIe Hard IP Blocks
Fractional PLLs
Hard Memory Controller
I/O, LVDS, and Memory Interface
Hard PCS
Transceiver PMA
Clock Networks
Transceiver Individual Channels
Core Logic Fabric and MLABs
M10K Internal Memory Blocks
Variable-Precision DSP Blocks

FIGURE 5.2: Functional Layout of an FPGA (Altera Arria V GX) [89]

As a comparison to the actual platform chosen for the DM during the work of this thesis in 2012, the employed Arria V GX chip has a core performance of 500 MHz. It features roughly 500k logic cells, and about 4 MB worth of RAM cells. The high speed transceivers can manage 3.5 Gbit/s per connection [89]. For the DM implementation described in chapter 7, the calculated maximum design performance on an Arria V GX is 138 MHz, the implemented (and verified) performance lies at 125 MHz.

Inside Informant One of the biggest benefits of current FPGAs, which has not been addressed yet, is In-System-Debugging. All large hardware manufacturers provide tools for their FPGAs, which allows logic to be configured as hardware logic analysers. These can tap the design currently developed, employing neighbouring, idle logic- and memory cells to do so. This allows complete examination of the design under investigation and makes the results available at the developer's computer. In-System-Debugging is an invaluable asset for programmable hardware development, considerably shortening development cycles.

5.4.3 The Limits of State Machines

Specialised logic cores are perfect when determinism and high throughput are called for, since they work in parallel, and do not need to share resources, there are no bottlenecks by design either – parallel tasks is where they excel.

But this also gives a first hint to the drawback – long sequential tasks are hard to implement. When using HDLs the assignment of hardware has a much higher impact on design performance than the way software compilers choose processor instructions. HDLs are therefore talkative languages, especially when it comes to the description of complex finite state machines (FSM). The development process is tedious and error prone when providing FSMs with a large degree of flexibility.

If there is a need for specialised logic cores, yet some tasks are highly complex sequences, there comes a point when the capabilities of a CPU are definitely called for. A processor that can synthesised for an FPGA and run a program with the desired task would provide all the desired benefits without adding a physical CPU. It costs less hardware resources than a comparable FSM, is easily maintainable and increases flexibility to a new level.

5.4.4 Best of both Worlds

Building from Scratch Due to the given flexibility in an FPGA, building a complete custom Soft-CPUs (SCPU) tailor-made for the given scenario is possible. Yet the major drawback of a custom processing unit, wielding its own instruction set, is the development in a total void: there are no existing tool chains. Virtually everything in a tool chain that makes a productive work-flow possible must be created from scratch, including assembler, linker, compiler, debugger etc.. There will most likely be little community support, because the application is very narrow. Since the scenario requires standard functionality like basic arithmetic, counters, loops and realtime clock support, there will be a lot of redundant work. A better choice would be the evaluation and possible adaptation of existing SCPUs.

Existing CPU Designs There are several open source SCPU as well as a couple of commercial ones available today. Soft-CPUs in FPGAs are a fast, lightweight alternative to real hardware CPUs. An extensive investigation of existing open source SCPUs has been conducted in the scope of the FAIR case study [90]. The processors have been evaluated in terms of processing capabilities, logic footprint, memory usage, periphery and documentation. There are several promising architectures available, which could be instantiated multiple times in a high-end FPGA.

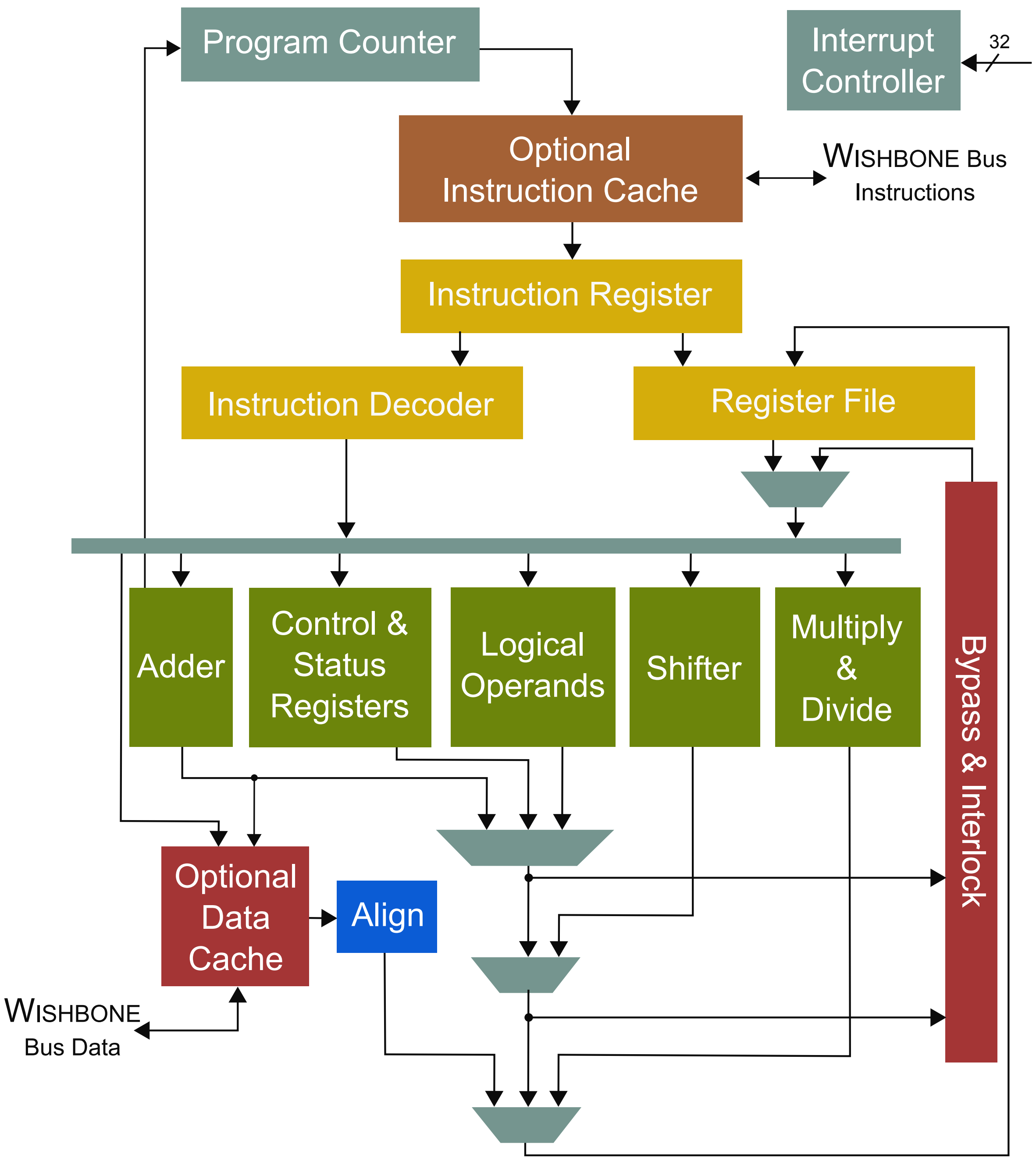


FIGURE 5.3: Block Diagram of Lattice Mico 32 Soft CPU [91]

A good approach would therefore be using tried and tested open source SCPUs and enhance them with custom periphery, or, if need be, modify the processors. This can range from adding new specialised registers, changing the way memory is accessed or extending their instruction set. The best choice from the evaluation in all categories was the Lattice Mico 32 (LM32). It is a powerful 32 bit RISC processor (yet at around 2000 logic cells still lightweight enough to be instanced multiple times), with excellent documentation and a large community. It is also fully configurable, having individual build switches for caches, hardware multiplier, barrel-shifter and JTAG debugging interface (see figure 5.3).

5.4.5 The missing Link

The last open problem is then interfacing such an SCPU to the specialised hardware cores needed to satisfy the tight timing requirements of the case study. The LM32 processor natively provides an interface for the open source WB Bus standard [92]. A bus system allows the creation of a System-on-Chip (SoC), a small scale computer system inside the FPGA. If all specialised cores were to feature a bus interface, full integration with the LM32 firmware is achievable, providing hardware acceleration whenever problems could not be handled solely by the SCPU. It further allows easy interfacing with external computers providing set values and operator control, if there were a protocol able to extend the bus interface outside the FPGA (see chapter 6, “Etherbone Protocol”).

5.4.6 Wishbone Bus Protocol

Wishbone (WB) is an open source bus standard created by the OpenCores Organization. The latest version as of 2016 is B4, which was released in 2010. WB is aimed at programmable hardware such as FPGAs and ASICs. Its focus lies on providing a robust and portable bus standard, thus enforcing compatibility between logic cores. However, while WB is a cycle based bus with strict master/slave roles, it was deliberately designed not to unduly constrain bus configuration, such as address and data widths, handshaking, topology and routing [92].

Choice In 2011, the decision was therefore made to employ the LM32 processor on FPGAs as part of a WB based SoC system. These systems are the infrastructure allowing timing endpoints to run the White Rabbit Timing protocol and the DM to form an RTS running machine schedules and dispatching timing messages.

The chosen WB configuration uses 32 bit addresses with 32 bit wide data and sub-word access, but is compatible to 8, 16 and 64 bit setups. It further supports flow control and pipelined operation.

Part III

Approach and Implementation

Chapter 6

Etherbone Protocol

6.1 Overview

The choice was thus made to build the FAIR CS using SoC architecture based on the WB Bus protocol (as described in 5.4.6) and using WR as the timing protocol. This opened the question of the interface over which SoCs should exchange data with each other and CS computers. A uniform interface for different hardware platforms was necessary, ideally supporting common network technology, as it is well tested and scalable. This led to an investigation into the possibility of transparently extending the WB SoC bus to another device. A decision was made in favour of a reusable approach, which should be able to model generic WB bus transactions, not just information specific to the FAIR or CERN CSs.

After a discussion of existing low level network protocols, the need for a specialised protocol for CSs became apparent. The following steps are the design considerations for the EB protocol. This chapter is based in large parts on the journal article "Open borders for system-on-a-chip buses: A wire format for connecting large physics controls" published in 2012 by the American Physics Society [93].

6.2 Purpose and Environment

There are several SoC bus systems available today. Typically, these buses are confined on-chip and rely on higher level components to communicate with the outside world. Taking these systems a step further, it is possible to extend the reach of the SoC bus to remote Field-programmable gate arrays or processors. This lead to the idea of the EB core, which connects a WB Ver. 4 Bus to remote devices. EB was named after the underlying technologies of its first implementation, Ethernet and WB, but is not bound to a specific hardware or transport protocol. It is today, as of 2016, available over Peripheral Component Interconnect Express (PCIe), Universal Serial Bus (USB), User Datagram Protocol (UDP) and Versa Module Eurocard-bus (VME).

EB was designed as a fast, low-level network protocol, intended for either software to hardware or hardware to hardware communication. It was developed in the scope of the WR Timing Project at CERN and GSI/FAIR and, due to the origin in accelerator CSs, it focuses strongly on determinism.

EB acts as a transparent interconnect module towards attached WB Bus devices. Address information and data from one or more WB bus cycles is preceded with a descriptive header and encapsulated in a network packet. Because of this standard compliance, EB is also able to traverse Wide Area Networks and is therefore not strictly bound to a geographic location, if the higher latency is acceptable for the application. Due to the low level nature of the WB bus, EB provides a sound basis for remote hardware tools like a Joint Test Action Group (JTAG) debugger, In-System-Programmer, boundary scan interface or logic analyser modules.

The following considerations apply to the original Ethernet based version.

6.3 Requirements

Timing Challenge Control of the accelerator's machines is time critical, but limits vary from a few nanoseconds, over several microseconds up to hundreds of milliseconds. When compared with a delivery time of $\leq 500\,\mu s$ for an EB packet, this leads to several conclusions:

- All actions need to be known in advance
- All actions must be precisely timed
- No time for acknowledgement or re-transmission

→ A deterministic, low latency command distribution system is necessary

For the efficient running of the accelerator it is clearly necessary that the CS must produce signals that are deterministic, i.e., the time at which an action is carried out must be precisely known and reproducible. To facilitate this, it is necessary that there is minimum delay in the path from the generation of the signal to its use.

Deterministic command distribution EB was designed to have very low latency and high determinism, giving secondary consideration to throughput. Because of this specialisation, EB application focuses on commands rather than the transport of raw data. Separation between the data and the commands working with this data is possible in most cases. This means machines are fed with data via a standard network infrastructure and also receive EB packets over the timing network, containing a command to carry out a preset action and a time of execution. EB is also able to address all specialised control blocks inside the WB HDL designs directly, a fact that is very useful for this implementation. For example, consider remotely programming a function generator with a parameter set for output level, gate length, and a sequence of trigger pulses plus a time stamp to execute all this. The function generator would be a WB memory mapped device, so EB will provide direct and easy access to all its registers and functions.

Compatibility and expandability Particle accelerator facilities like GSI and CERN host large heterogeneous pools of equipment, which have evolved for more than 40, or in CERN's case, almost 60 years. Most equipment at the LHC and on the FAIR project is quite new, but there are still many legacy systems to cover. FPGA based technology enables all sorts of adaptors and converter logic with little or no extra hardware effort, while Ethernet based infrastructure easily covers the distances in growing facilities. This shall also be reflected by the next generation CSs.

6.4 Further Applications

There are many places on site where well-known hardware tools like debug modules, in-system programming adaptors, logic analysers, and similar are needed to deploy, maintain, and upgrade hardware. Because of the distances between nodes, which are about 2 km at FAIR and 10 km at CERN, and the quantity of nodes involved, routing was another requirement of EB. It will not only reduce the time it takes engineers to travel on site, but also makes automated testing easier. Personnel can also collaborate more easily, since access to the hardware tools can be shared over the network. Last but not least, it is possible that the electronics are not accessible during beam time and for some time afterwards due to radiation concerns. This feature will reduce time requirements for maintenance and deployment.

6.5 Related Work

There are many different examples of available protocols for direct data exchange. Among the most commonly used low-level were Myrinet in the super-computing sector (almost completely replaced now by Ethernet based equipment) and different Remote Direct Memory Access (RDMA) [94], [95] implementations. While there are pure software implementations of RDMA, their latencies cannot compete with hardware implementations like Infiniband [96] or iWARP [97], which can achieve latencies below 7 µs. However, these are mostly optimised for maximising throughput, while short message latency is a secondary factor. PCI Express falls into the same category and will be addressed later in more detail.

There are also high level protocols available like the Common Object Request Broker Architecture (CORBA) [98] and the Simple Object Access Protocol (SOAP) [99], which aim for abstract software to software communication in heterogeneous environments. While being very versatile, due to their higher logistics overhead and generic nature, they are not well suited for fast communication between software to hardware or hardware to hardware implementations. The common factor of the above is that they are not tied to a specific underlying bus protocol at their end points. While they keep data content, they will not preserve syntax during transport.

PCI Express A comparison of widely used field buses, such as USB, PCI, and PCIe [100] showed PCIe as the best match to the niche EB is aiming for. As will be shown in detail in section 10.3, PCIe does not quite achieve the required low overhead though, mainly because of difficulties associated with routing and interfacing to other systems. PCIe has nevertheless many features that are desirable for EB. Like the GbE Interface, EB uses a 125 MHz clock rate. While the network end point uses an 8 bit interface, the WB interface connected to EB is 32 bit wide, giving it 4 times the bandwidth. Regardless of delays for processing the packet structure, the difference in bandwidth ensures EB is fully capable of streaming. EB has several design traits in common with PCIe. Both are serial field bus protocols, they feature error detection in the form of a cyclic redundancy check (CRC) to ensure packet integrity, carry routing information, and provide quality of service (QoS). Also, both protocols go all the way down to the physical layer. PCIe features auto-discovery of bus devices, which is also present in WB and (and therefore EB) since March 2012 under the name self-describing bus (SDB) [101].

However, there are also differences. While PCIe is packet-based from the bottom up, EB's underlying WB bus is cycle based. PCIe was also designed for higher throughput than WB, by bundling several lanes into one connection, and is meant to run at higher frequencies. This said, there are also differences to EB on the upper layer. The first lies in the routing capabilities. While PCIe can be switched much like the MAC layer of Ethernet, it cannot do complex routing and most importantly is not native to wide area networks. This makes long distance connections over common network architectures impossible. PCIe could of course be encapsulated in IP packets to do just that, but given the similarity between PCIe and Ethernet/IP packet headers, this would be almost completely redundant and therefore double the overhead. PCIe is more powerful than WB, but it also has its downsides. The controllers are vastly more complex, most of this is due to backward compatibility for PCI. It is much harder to implement in an FPGA than WB and therefore takes up many times more resources. Also, while PCIe controller chips are readily available, high quality HDL cores providing similar functionality are all commercial and closed source. Superior routing capabilities, easy connection to HDL blocks, the possibilities for expansion, and the focus on latency over throughput make EB a worthwhile project to investigate. To conclude the evaluation, all differences are based on the trade-off between flexibility and extensibility on the one hand versus latency and overhead requirements.

6.6 Architecture

6.6.1 General Consideration

Since bus protocols can differ greatly in their operation and packet layout, conversion between them can severely reduce fidelity. For EB, WB B4 therefore has been chosen as the bus implementation, while leaving the underlying transport protocol open. There are two categories of EB devices under development: buffered, non-deterministic software modules and low-latency, deterministic streaming hardware cores (see figure 6.1). Software nodes are used for all applications where determinism and latency are not the main issue, but

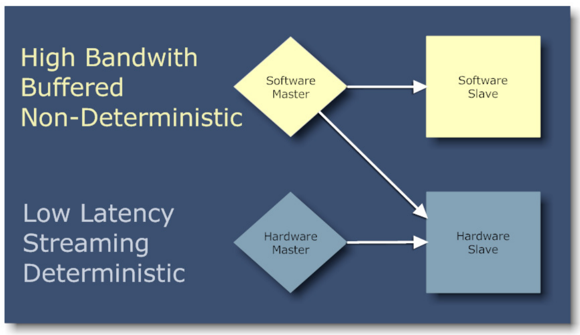


FIGURE 6.1: Compatibility between EB node types

interoperability and fidelity of bus signals are. Examples would be a developer's remote computer, running a serial console on one of the timing end points or debugging software via JTAG module on an embedded system elsewhere on the site. Figure 6.3 shows an example block diagram of such a setup. The top box contains the EB library running on a PC, while the lower is an FPGA board hosting an EB slave. It is attached to multiple HDL blocks via a WB interconnect. The EB software library provides a generic interface for the driver, and the method of connection to the EB device is not visible to the application. Most packet-based protocol could be applied to carry EB information, so an EB device locally connected via USB would behave exactly the same as if it were connected remotely over Ethernet. Hardware nodes operate in full streaming mode, they are fully deterministic and designed to minimise latency. An application example for a hardware node would be an end point of a timing system, receiving commands to generate a pulse at a specific execution time. The deterministic characteristics of EB ensure that the available time frame for delivery does not vary. Streaming provides low latencies, reducing the reaction time the CS has to an event occurring elsewhere on the timing network. Hardware implementations are of course not as flexible as software. The streaming hardware slave implementation (Figure 6.5) uses an HDL block as a deterministic EB node. On the network interface side, it features streaming WB channels to GbE block, on the SoC side, it has a WB master that will usually be connected to an interconnect. The RX and TX cores are directly linked in order to already create the TX reply header while the incoming header is processed.

6.7 Etherbone Design Choices

6.7.1 Underlying Transport Protocols

The EB protocol has been designed to be deterministic with a focus on minimal latency. It also needed to be able to use standard transport layer architectures. So a widely supported protocol with very low overhead was needed and the decision to use IPv4 at the network layer and UDP at the transport layer was made [102].

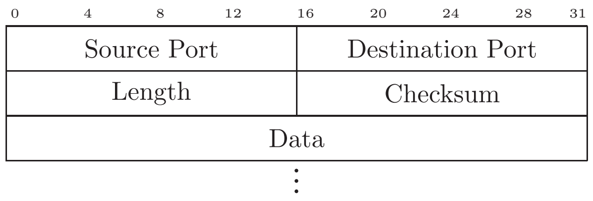


FIGURE 6.2: Simplistic structure of the UDP Header

Ethernet Since WR utilises IEEE 802.3 Gigabit Ethernet technology, making EB inter-operable with Gigabit Ethernet standard was an obvious choice in the development. Raw Ethernet frames, however, were not an option, because they cannot pass routers and firewalls without special configuration. Delays are reduced by the use of QoS functionality for command messages to ensure that they are not held up behind other frames. For lowest latencies, WR switches support cut-through mode. This means a packet is not stored and forwarded but directly passed on, the first outgoing bit leaves the switch before the last incoming bit was received.

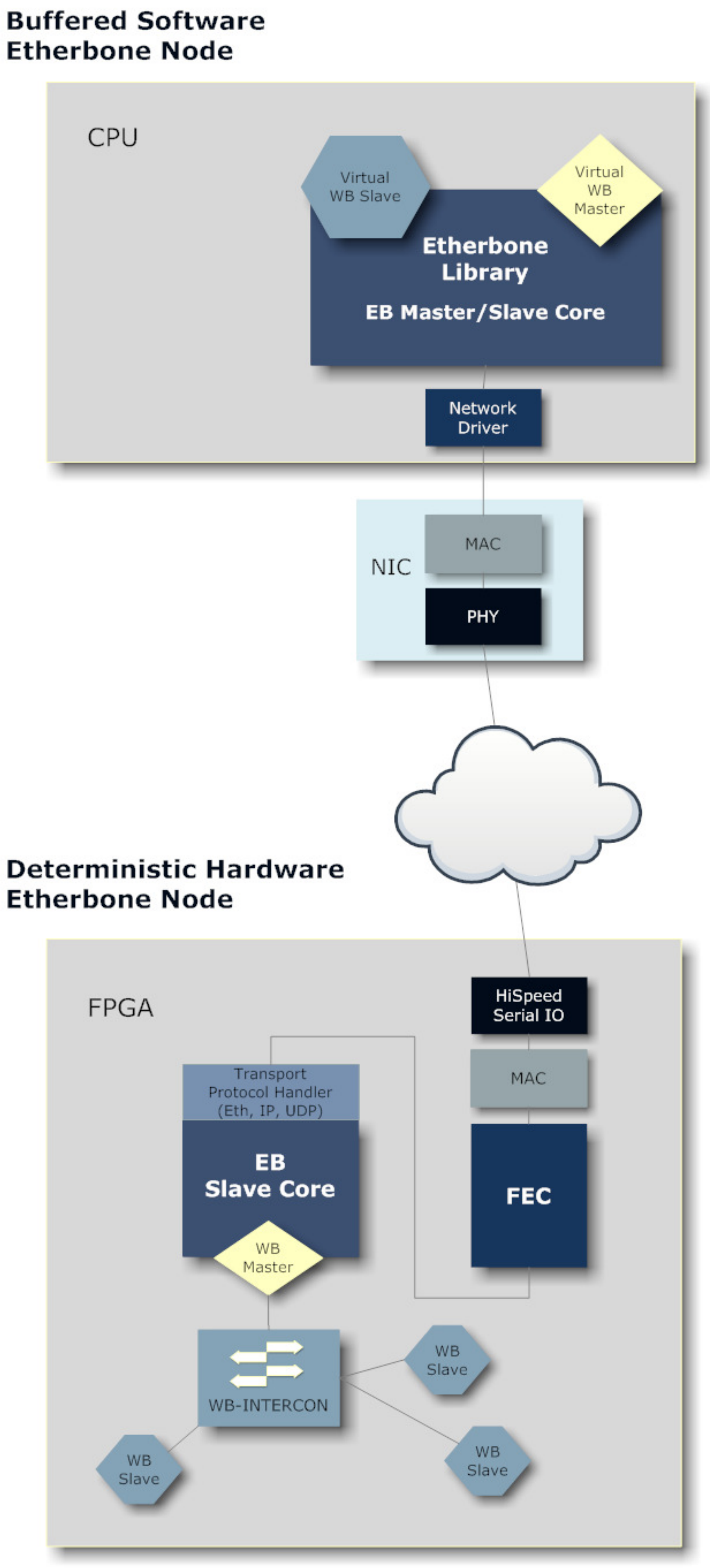


FIGURE 6.3: PC EB master and FPGA based EB slave

IP The choice for IP was based on the simple fact that it is the most widespread general purpose protocol. In order to send EB datagrams over WAN, there is no alternative supported by an off-the-shelf network equipment.

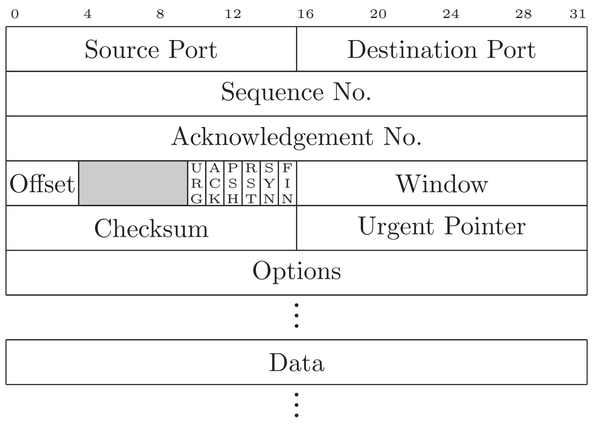


FIGURE 6.4: TCP Header. More powerful, but more complicated

UDP When comparing the structure of UDP (Figure 6.2) to the more powerful, the main difference lies in UDP being a stateless protocol. It dispenses with a handshake, sequence numbers, and acknowledgements in favour of simplicity and low overhead. Contrary to UDP, TCP is capable of retransmitting of lost packets [103]. Unfortunately, this is not an option for the CS scenario, since there is not enough time for re-transmission. These properties make UDP a better fit for the goals of EB than the TCP protocol.

6.7.2 EB Protocol

EB itself is designed for low overhead and latency as well. The packet header consists of a flag block, containing information on master and slave WB bus type, protocol version and a mechanism for negotiating a common bus mode. It is followed by records containing bus operations. Each record will add a header and one or two base address fields to the overhead. In our implementation, this adds 8 or 12 Bytes overhead per record. Packet header and record format have a smaller footprint than those of RDMA, the records are similar to PCIe. The implementation also uses a 125 MHz bus clock for WB. At 32 bits per cycle, WB has 4 times the data rate of the GbE interface. This makes packet processing without additional delay possible, the only limit stems from the reply time of targeted WB devices.

6.8 Methods and Test Implementation

The current hardware/software test implementation utilises UDP/IP protocol. As a requirement, EB needs duplicate free transmission, which UDP alone cannot guarantee. It will therefore be assisted by a forward error correction scheme on OSI layer II. In order to be fully deterministic and to achieve lowest latency possible, EB needed to be fully streaming capable, ideally introducing no additional delay to the passing data. Waiting for replies would block the EB master and its network interface for an unknown time, which is unacceptable. To ensure determinism, a hard timeout is enforced at this point. If an EB slave has to wait too long for a connected WB device to answer, a buffer underrun will occur, corrupting the streaming reply. The main problem is that the IP and UDP packet headers contain length information and checksums on the payload. However, the Ethernet CRC field is trailing, so

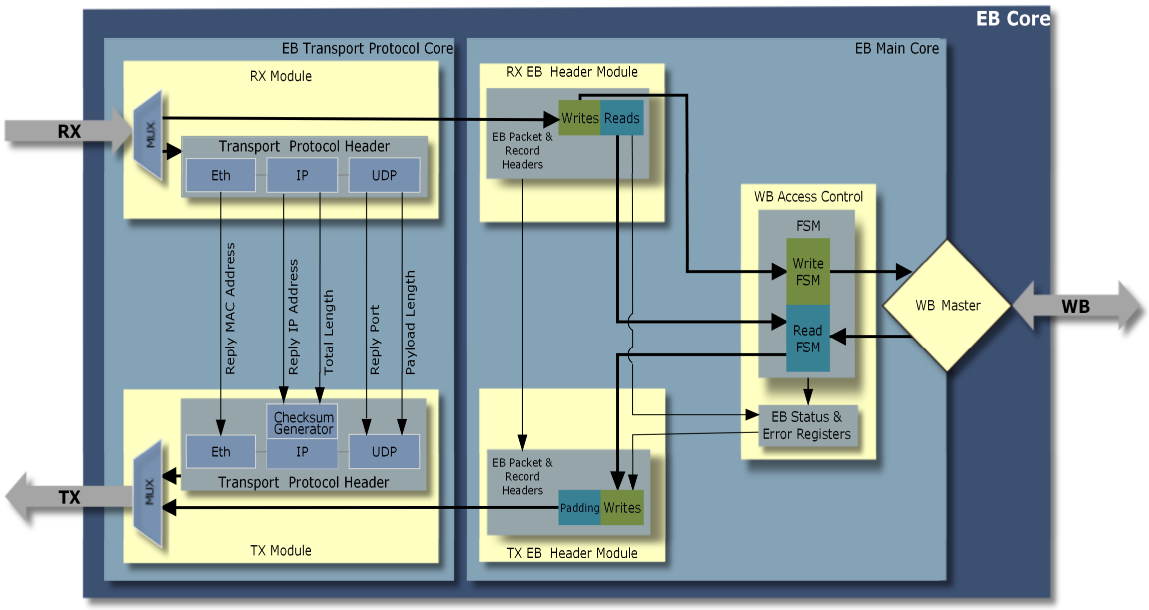


FIGURE 6.5: EB streaming hardware slave

it is possible to insert information and calculate a new CRC on the fly, solving the problem. This is not the case with either the UDP or IP fields, where prior knowledge about the payload is necessary. For low-latency streaming, it is necessary to resolve the dependencies between packet header and payload. Like the underlying WB bus [92], EB has master and slave nodes, which form complementary pairs. This leads to a bridge architecture, where an EB master accepts bus operations from local WB masters for transfer to a remote node. A sample hardware implementation is shown in Figure 6.5. EB slaves therefore have a WB master interface and form the remote representation of the local WB master.

6.8.1 Packet Length

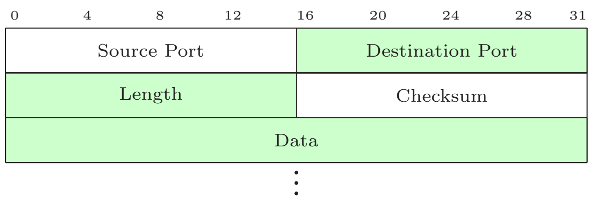


FIGURE 6.6: UDP Header, fields causing dependencies for the reply are marked in green

The UDP/IP header requires the packet length fields before the payload [102], [104]. To avoid waiting for the complete packet to be received, streaming EB replies are the same length as the corresponding request, allowing the full header to be determined in advance. EB responds to every incoming read operation with an outgoing write operation, while incoming writes are answered with zero padding. There are no exceptions to the rules since these are treated as empty EB records. The result is similar to a no-operation instruction in a CPU. Figures 6.6 and 6.8 show the structure of an outgoing UDP/IP header. The outgoing header does not need prior knowledge of the contents of the payload in the incoming packet. All header fields can either be taken from the header of the incoming request (reply address, port, length) or deduced from it (IP checksum) or use data already locally available (source address and port). All processed fields are coloured in light green.

6.8.2 Checksums

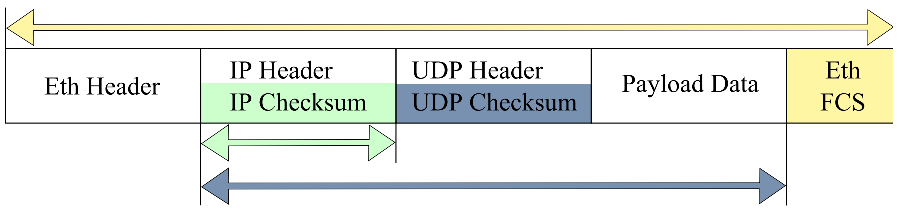


FIGURE 6.7: Checksum coverage in a UDP/IP packet

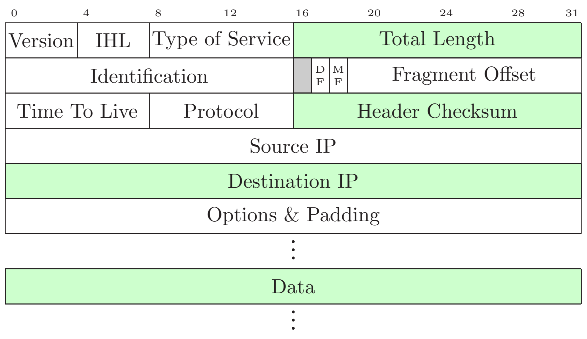


FIGURE 6.8: IP Header, fields causing dependencies for the reply are marked in green

Coverage The Ethernet frame checksum field follows the frame's payload, so it can always be calculated on the fly. This is not the case with either the UDP or IP fields, where prior knowledge about the payload is necessary. Figure 6.7 gives a brief overview of the areas protected by different checksums in a UDP/IP packet. When replying to a request, almost all information required can be taken from the incoming packet header, except source IP address, IP checksum, and source UDP port. The IP checksum is only dependent on fields of the IP header (see figure 6.8) This includes source and destination address, IP packet options, and the length field. Source address and port are already known to the node, which leaves the payload length, on which the checksum depends, and the checksum itself [105]. Because the IP checksum algorithm is basically a sum, all known fields can already be added in advance. This leads to a prefabricated checksum that can be kept for future reference. Since only the length and new destination information needs to be included, this process is very fast (see figure 6.5, TX block).

Solving Dependencies Because of the introduced symmetry, the length is also known in advance and so the IP checksum of the reply can be calculated in advance directly after header reception. The UDP checksum is calculated from a pseudo-header, which is not transmitted. It contains the source IP, destination IP, protocol (see figure 6.8), the UDP length field, and the following data (see figure 6.2). The UDP checksum therefore depends on the payload, but UDP protocol specifications allow the checksum to be set to zero. This signals the recipient "not used" [102]. Since the more powerful forward error correction is used in addition to the CRC, the UDP checksum can be omitted without risking data integrity. With this, all reply header information is available at the beginning of the incoming payload. This eliminates all wait times for payload reception, trading bandwidth for latency. In effect, the slave will already start sending the reply at the moment that the request header is processed.

6.9 Etherbone Data Format

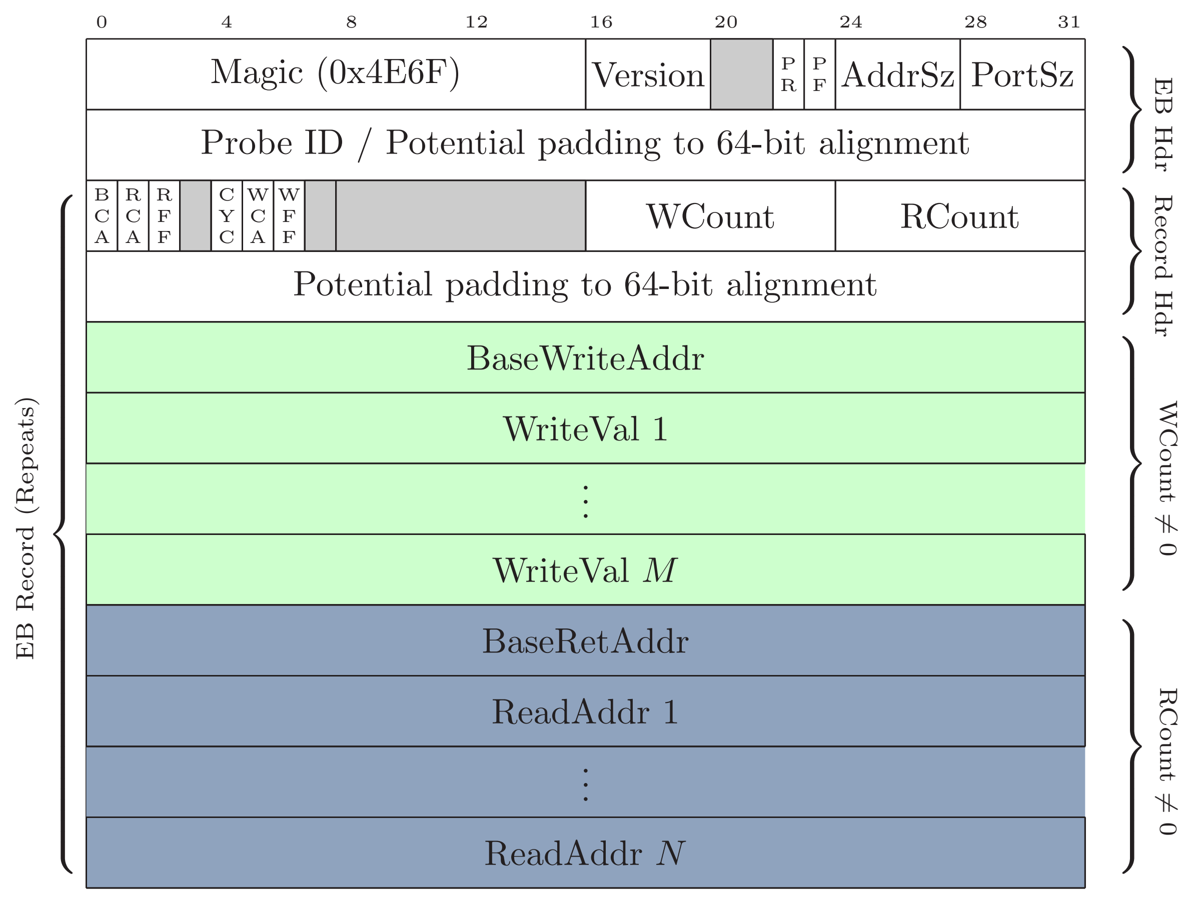


FIGURE 6.9: EB packet structure. The 8B header is followed by EB records containing WB bus operations.

Based on the previous discussion, a design for the protocol's data structure was proposed, the full EB datagram is shown in figure 6.9. The EB layer consists of a header block and one or more record headers with matching write or read operations. There is also the option to set a probe flag, which is used for negotiation of usable bus and address widths between two devices. A probe packet is always padded to the maximum alignment, 64 bits in this application to ensure compatibility. A record header contains a set of flag bits stating optional information about source and destination like the use of FIFO mode. The flags are followed by the number of write and read operations in the record, this can be a number between zero and 255 if no feedback is necessary (assuming sufficient free space in the packet). If the bus is wider than the record header, it will be padded. After the record header follow bus operations, writes first, then reads. Bus operations are not mandatory, an EB record can therefore be empty, contain writes, reads, or both. Each of these blocks is preceded by an address field. For a write, this field signifies the target start address on the slave; for a read it is the address to which the read values should be written to on the master.

6.9.1 Communication

Negotiation The developer has a free choice in both bus width and address width of a WB bus to best suit the requirements and can choose to also support smaller configurations. This leads to the question of which mode is supported by both a master and the targeted slave device. Initially, the master sends a probe packet to the slave, consisting solely of an EB header without a payload. This header has a set probe flag and shows all modes supported by the master, followed by the probe ID. The ID is unique to the master and can either be derived from configuration or random. This ID is necessary in case the destination IP of the sender does not match the source IP of the recipient. The slave then sets the probe response flag and answers with all modes it supports, followed by the received probe ID code. The EB master then knows about all possible bus and address widths it can choose from for communication with this particular slave device, completing the negotiation. The next step is to send a normal EB packet containing one or more EB records. The slave will reply with the bus width and address size chosen by the master in the request header. In the FAIR and CERN CSs, the most common bus width will be 32 bits.

Atomics EB supports atomic WB bus operations. By holding the cycle line on the WB interconnect, the connection to the target slave is kept, allowing the next record to be transferred without another slave being able to use the bus. Each EB record comes with the option of ending the current bus cycle on completion or keeping it for the next record. With this mechanism, bus ownership can be held over as many EB records as required, avoiding interference with the operation by other bus devices.

Status Information To determine the success of any EB operation, the status register in the EB configuration space can be read at the end of a stream of bus operations. It shows the flags for all previous operations, either ACK or ERROR. This is a log of all acknowledgements received at the ACK / ERR lines of the WB bus interface [92].

Symmetry Command messages mostly go from a master to a slave, but there are also cases where the master urgently needs to read information from a slave device. Equal packet length of incoming and outgoing traffic is essential for EB streaming mode, because it significantly reduces round-trip time. In order to keep equal length between request and reply, results from reads are converted to write operations while writes are turned into empty records, i.e., padding. This makes the length field of UDP/IP headers known in advance and by that removes all wait times stemming from header/payload dependencies.

Addressing EB write operations are values to be written, while reads are addresses to be read. Write addresses therefore need to be generated by incrementing the start address, reads can be random access. The increment can either be zero, in which case the target is treated as a FIFO, or sequential. An EB master must keep track of read addresses it sent in order to correctly interpret the answering write. Addresses that are out of bounds cause either an error, if the address is within the mask, but not mapped, or wraps around if it is greater than the mask.

Management EB also supports the use of a configuration space, an address space that is not associated with the local WB bus and only concerns the EB node itself. It can both be accessed via EB and a local WB slave interface used for configuration of the EB node. This configuration space has several purposes. For one, the EB node's own MAC and IP address are set via one of its interfaces and kept here. The configuration space is also used to map incoming EB packets to the originating queries. Last but not least, it contains a feedback register of operations on the bus. If feedback for success of operations is required, this shift register can be read to supply the ACK or ERR bit for the last 64 operations on the WB bus interface.

Security In the development of the EB protocol, the decision about the protocol for OSI transport, Internet and link-layer has been intentionally left open to provide maximum flexibility. Likewise, EB does not contain any specific authentication or encryption mechanism. If access control and cryptography are required, it can be wrapped in any proven secure protocol of choice (e.g. L2TP, IPsec, TLS/SSL). However, doing so would add several hundreds of micro seconds processing time, depending on the performance of the FPGA based embedded processors, which would be unsuitable in a high speed control system scenario. Dedicated cryptography cores, on the other hand, could provide encryption at a very low latency cost and full throughput. For symmetric ciphers like Advanced Encryption Standard (AES), there are fast FPGA based open source implementations available today. The work of Yadav and Rajawat [106] for example describes an 128 bit key AES core, which can easily handle a bandwidth beyond 5 Gbit/s and provide a latency of as little as 11 clock cycles (88 ns). Secure key exchange, as it is not time critical, could then be handled in advance by a standard software protocol, such as a variant of the Internet Key Exchange (IKE) protocol.

It needs to pointed out that control systems typically are not overly concerned with eavesdropping attempts, but about unauthorised or manipulated commands. In the FAIR case study in particular, all endpoints are allowed to listen, but very few have the right to issue commands. This means that there is no actual need for encrypted links, but for authentication, i.e. certificates for commands. A complete implementation would require asymmetric encryption in all nodes, which is very expensive in hardware and interferes with high precision timing.

One other alternative should therefore be considered for investigation. Because the network is point-to-point, almost all endpoints are passive and eavesdropping is not an attack scenario, the last mile, the connection between switches and endpoints, could be in plain text. The switches should then use a fast symmetric encryption for communication among themselves and to endpoints authorised to issue commands.

The distribution of the symmetric session keys is not time critical and can be handled by a proven public key infrastructure. By dropping all content which does not form valid packets after decryption, unauthorised sending can be prevented. Man-in-the-middle-attacks can then only target the last mile, requiring physical access to an endpoint's fibre link.

Access to current trusted network Since deterministic communication in the microsecond range is not possible over WANs, all necessary connections from the outside world into the timing network can be assumed to have loose timing constraints. They are therefore handled by gateway computers connected both to the campus and the timing network. These computers will host a Secure Socket Layer (SSL) connection to the campus network and will tunnel all the client's EB traffic.

Chapter 7

Data Master Hardware

7.1 Overview

In this chapter, a design approach for an SoC system able to fulfil the requirements of the case study (chapter 4) on the FPGA platform favoured in the technology survey (chapter 5) is presented. It covers the architecture for the DM and provides solutions for further improvements by the addition of bespoke hardware acceleration modules.

7.2 System on Chip Architecture

7.2.1 Processing Unit

In order to generate timing messages from machine schedules, a deterministic processing scheme needed to be devised. A pure hardware state machine approach was considered and dropped, because as shown in figure 4.5, machine schedules are graphs with multiple possible paths. Even for a simple BPC with only one alternative sequence, the necessary flow control features loops and branches. This would require more than one state machine, and for more complex patterns, and a stack as well. The resulting custom component would still lack the flexibility of a real microprocessor, but consume almost as many resources in the FPGA. A more sensible approach therefore was to chose a processor that can be implemented in an FPGA and is capable of deterministic operation. The choice fell in favour of the LM32 soft CPU [90], [91]. An LM32 has around 2000 logic cells, but is still lightweight. Today's mid range to high end FPGA have more than 250000 logic cells and can therefore easily create the functionality of multiple CPUs. According to early tests, the limiting factor was more likely to be the amount of RAM present in an FPGA than the number of logic units.

7.2.2 WB Bus and Crossbar Switches

The connections between the LM32s and its peripherals are provided by a collection of Cross Bar switches (CB). Such a switch supports parallel connections of m masters to n slaves, provided only one master tries to access the same slave at the same time. In addition, the CB contains a parallel WB bus with the Message Signalled

Interrupt (MSI) subsystem, which is used to trigger interrupts or exchange short synchronisation messages between modules (see chapter 8.5.9, 126).

7.2.3 Memory Access

Loading new schedules into a CPU's memory without influencing current execution was the first problem to solve in such a design. Fortunately, programmable hardware has a native solution to provide non-blocking access, which are dual port memories. By placing one port under the control of the CPU and assigning the other to the host bridge, both sides can access memory without causing delay. However, this does not prevent race conditions in the case of simultaneous read/write or write/write accesses. A synchronisation scheme using semaphores is necessary to avoid collisions. Since the host system is not a hard realtime system, it can be made to wait until the embedded CPU marks a memory region safe to update. More on this topic can be found in chapter 8.4 on page 117 onward.

7.2.4 Global Time

To execute machine schedules, it is important for the CPU to know the current global time. While it is available at a register of the WR core, the latency for access to this information must be deterministic. This is not possible if the resource is shared between multiple CPUs. The approach taken was to duplicate the WR time register and directly include each duplicate as a distinct WB device in the personal crossbar of each CPU core. This means that the resource is not shared and access times are guaranteed.

External Event Sources Remote hard realtime devices, such as endpoints involved in a B2B, sometimes need to communicate their readiness to the DM. The endpoints send EB packets to the DM, targeted at areas in the memories of individual CPUs reserved as in-boxes. In order to know which DM bus address is their corresponding in-box, endpoints must receive this information from the DM beforehand.

All other event sources, like beam requests or interlocks, communicate with the host platform, which in turn sends messages to the DM. A detailed explanation of the proposed messaging scheme is given in chapter 8.5 on page 122 onward.

7.3 Prototype Synthesis Analysis

A decision needed to be made on how many CPU instances shall be included into the prototype. It was therefore required to find criteria for the number of possible and the number of useful instances, i.e., the maximum number of CPUs able to be gainfully employed in the CS. The following paragraphs contain the discussion on theoretical and practical limits.

7.3.1 Necessary number of CPU Cores

A good criterion for finding the minimum number of CPUs is their combined output bandwidth, since their sum should be sufficient to saturate the network interface. Jumbo packets are not supported in the WR network design, all calculations therefore assume a maximum packet size of $1500\,\mathrm{B}$. The size of a single timing message, l_m, is $32\,\mathrm{B}$ or $256\,\mathrm{bit}$. The protocol overhead from Ethernet, IPV4, UDP and EB will lie in the range 20 to $44\,\%$ of total GbE bandwidth, depending on the overhead to packet ratio and therefore mean packet size. A detailed analysis can be found in chapter 10.3.1. With the Forward Error Correction encoder from current development considerations, the overhead rises to 80.72 to $88\,\%$ [78]. The number of CPUs must therefore be sufficient to at least generate $1\,\mathrm{Gbit/s} \cdot (100 - 80.72)\% = 192.8\,\mathrm{Mbit/s}$ of net command traffic.

The maximum message traffic a single CPU can generate depends on the processor clock rate, the number of cycles an instruction takes to execute, bus behaviour and access delay, and finally the firmware employed. An attempt at formally determining the execution time of a program does not necessarily yield a solution (see also (chapter 8.6.3). However, the base cycle of the scheduler is quite easy to measure in the present case by loading one CPU with a current prototype firmware and assigning it a machine schedulethat generates a timing message every clock cycle. Some extra code inserted into the firmware is sufficient to perform the measurement, adding the current WR time to a cumulative sum and recording the historic maximum.

The result for the mid 2016 implementation was $7.75\,\mu\mathrm{s}$, with a marginal deviation between maximum and mean of 28 clock cycles. This leads to the following requirement for the number of CPUs:

$$\begin{aligned} n &\geq \left\lceil b_{max} \cdot \frac{T}{l_m} \right\rceil \\ &\geq \left\lceil 192.8\,\mathrm{Mbit/s} \cdot \frac{7.75\,\mu\mathrm{s}}{256\,\mathrm{bit}} \right\rceil \\ &\geq \lceil 5.84 \rceil \\ &\geq 6 \end{aligned} \tag{7.1}$$

The minimal number of CPUs in the prototype should therefore be 6. This is a not yet optimised version of hard and software, however. If optimisations were added, fewer CPUs would be necessary, but the added processing power would still be useful as a safety margin and for the implementation of new features into the firmware.

7.3.2 Test Setup

Several experiments with the Quartus II 15.1 and Quartus Prime 16.0.2 hardware design software [107] were conducted to test the upper limit of CPU instances on the employed ARRIA V GX (5AGXMA3D4F27I3) chip [89]. The limiting factors could be available memory, available logic cells, timing closure, routing cost and distance between memory and logic.

Routing Whenever a value is processed in more than one logic term, it needs to be physically copied, a "fan-out" must be created and routed to the destination. Logic cores must be distributed over a two dimensional area, so routing between logic costs more logic. Quite often, routing can reach similar costs as designs.

Timing Closure The other problem in FPGA designs is timing closure - the signal propagation delay along dependent logic cells. It is a complex topic and shall only briefly be addressed at this point. Timing closure deals with synchronously routing signals (data and clock signals) in the chip. Clock signals must arrive at dependent locations within the same period, data must arrive before the clock edge. In addition, all logic paths in a design must kept so short, that a signal can traverse it from start to finish within one clock period. This means cutting long logic paths by inserting of registers, the complete operation taking as many clock cycles as register stages.

7.3.3 Result

Visualisation Figure 7.1 shows the output of the "Chip Planner" tool, a visualiser for resource utilisation on the FPGA chip, which is part of the synthesiser suite. The memory cell areas can be recognised as vertical columns of olive colouring. In all other areas, different grey shades indicate logic utilisation, from black (unused) to white (full), logic designs of interest are highlighted in colour. Pure hardware cores are shown in grey (PCIe communication and analogue PLLs, see 2.4.2). The programmable logic cores are, starting from the top right:

• Event-Condition-Action unit (ECA)	red, top right, see 7.6
• Timestamp Latch Unit (TLU)	yellow, centre, see 7.6
• Priority Queue (PrioQ)	orange, right, see 7.4
• Processors (CPU*n*)	blue to green, lower half, see 5.3, 7.2.1
• PCIe host bridge cores (PCIe)	yellow, lower left
• Etherbone Master core (EBM)	dark green, left, see 7.5
• White Rabbit Core (WR)	purple, top left, see 2.7.1

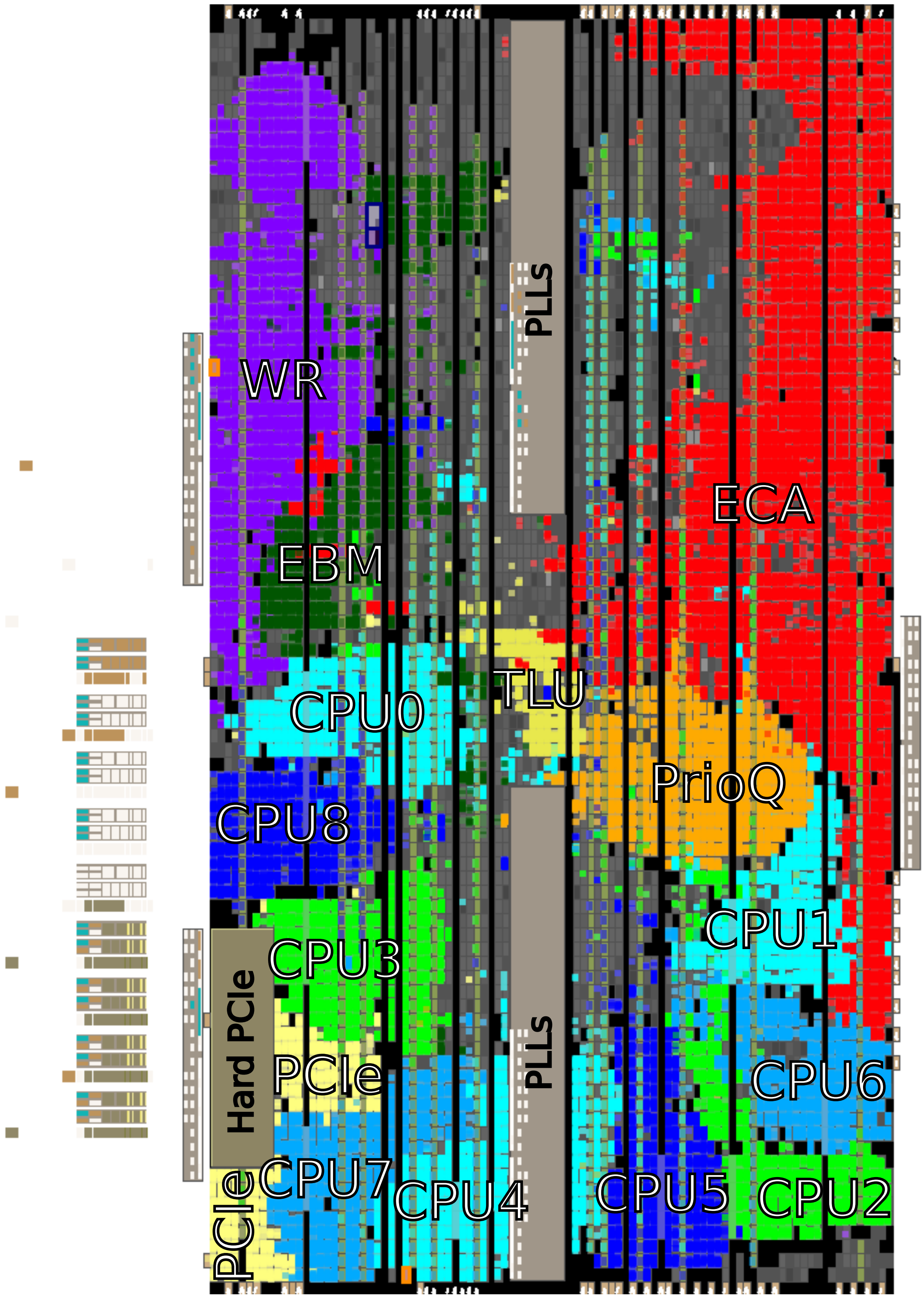


FIGURE 7.1: Visualisation of Chip Usage with 9 CPU instances. More details are provided in the text.

Implications The logic utilisation of CPUs is marked in colours from green to blue. It is obvious from the coloured area that 9 CPUs is very close to the limit of available logic. Not far behind as a limiting factor turned out to be memory availability, the aim being to assign each CPU at least 64 kB of memory. While there is more memory in the system, the organisation in 10 kbit blocks lead to small components in other parts of the system wasting much of the memory they were assigned.

The intuitive assumption of routing distance to memory having an impact on design performance was proven to be wrong. Observed routing paths between processor and RAM cells quite often spanned more than half of the chip, the performance of the interconnect fabric is far better than logic routing delays. Instead, most effort is put into placing the CPUs into close vicinity to each other, seemingly in order to minimise the WB bus routing effort. Timing closure turned out not be a limiting factor in this prototype.

A maximum of 9 cores was successfully tested with 64 kB memory each at 125 MHz design frequency.

7.4 Message Priority Queue

Timing messages must ultimately be dispatched via the timing network interface. Since there are multiple CPUs producing timing messages at certain times, the network interface is a shared resource. Figure 7.2 shows how individual CPUs act as sequencers. When going through their schedules (entries), they are outputting streams of due timing messages (green arrows) to the network interface (grey arrow). It is necessary that the CPU's network access is deterministic, which requires a fair arbitration.

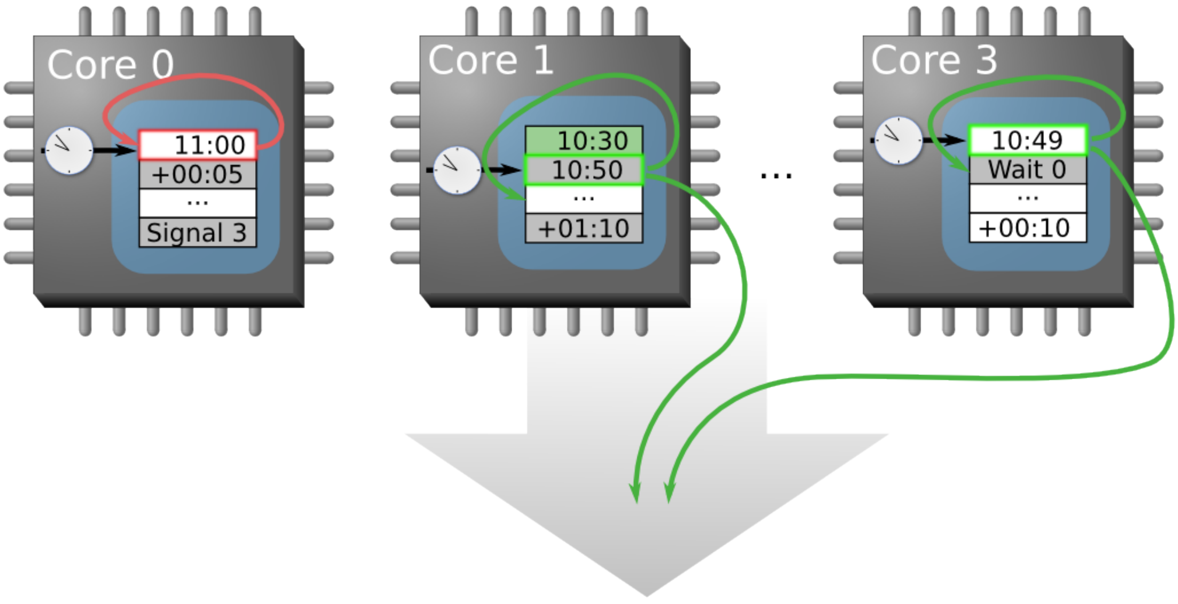


FIGURE 7.2: Dispatch of Timing Messages in Soft-CPU Cluster [108]

7.4.1 Basic Considerations

A simple round-robin scheduler, granting access in turn to each requester appears to be a sufficiently fair arbitration scheme at first glance. The very first DM implementation used such a simplistic scheme for testing. However, it is insufficient. If the different message streams were not reordered according to their deadline, urgent messages could be unduly delayed. This is especially the case if the message lead would be used for traffic shaping and is thus not the same for all messages. Furthermore, waiting for the network resource could block individual CPUs, making their programs non-deterministic.

The problem can be solved by introducing an earliest-deadline-first scheduler (EDF). The policy is to always grant access to the most urgent message. To avoid starvation in case of large bursts of simultaneous deadlines, a round robin policy shall be applied in case of a tie. Using this principle in a system with n CPUs, the maximum latency for any message is bounded to the processing time for $n - 1$ messages while simultaneously making the arbitration impervious to bursts of early messages (messages with high remaining delay budget).

7.4.2 RAM-based Heap

There are several ways to implement priority queues, depending on the focus. In the present case, the goal is minimal execution time of the get-Min and extract-Min operations, which returns the minimum key element. In order to get better performance than sequential reordering in software, parallel accesses to memory are necessary. For the second test implementation, hardware assisted sorting algorithms were investigated.

For a heap based solution, at least two independent read ports are required on the RAM holding the sorting keys, so both left and right child node can be read at the same time. With the moving element in a register, it is possible to compare parent and both children in a single cycle. For reordering, an independent write port is required, so the minimum is a 3 port RAM (2 read, 1 write). Many manufacturers provide macro cores which emulate a multi-port RAM with 2 read and 2 write ports. However, the synthesizer can only achieve this by replication. So instead, it is more efficient to exploit the fact that the two read ports never access the same child node and keep left and right children in two separate dual port memories (DPRAM)s.

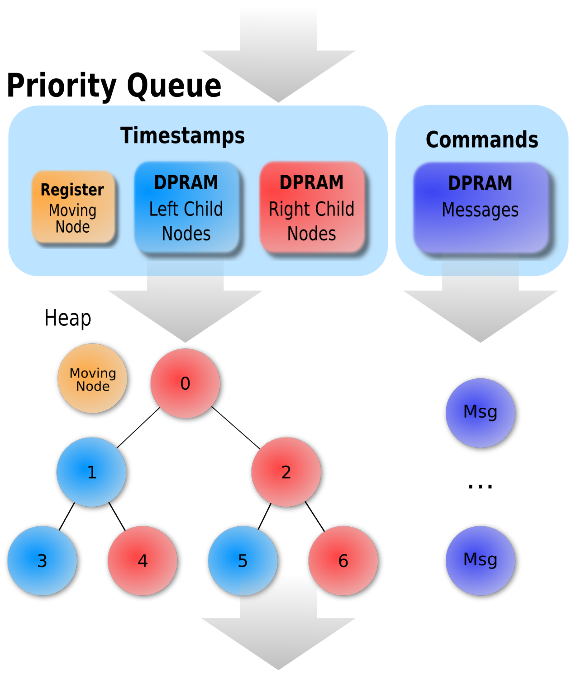


FIGURE 7.3: Heap-based priority queue with Dual Port RAMs

As figure 7.3 shows, no redundant copy of the keys is necessary, and the arbitration logic is mostly Multiplexers and Write enable signals controlled by the Least Significant Bit (LSB) of the node index. The path taken through the heap is a sequence of indices. Placed in shift register, it can be used as addresses to order the message payload with constant delay to the key sorting. This simple design can implement *INSERT* and *REMOVE* Operations in $O(log(n))$ time, with one comparison/move per clock cycle per heap level plus two cycles delay and minimal memory cost. However, there is still room for improvement.

Optimising the Solution The second implementation was based on the approach of simultaneous access to all heap levels Because *REMOVE* and *INSERT* normally work in different directions, pipelining of mixed operations is prevented. *REMOVE* operations can, however, be pipelined if all heap levels are accessible at the same time [109]. At the cost of fan-out and additional arbitration logic, one could reformulate the *INSERT* operation to work from top to bottom as well. In addition, two DPRAMs, one for left children, one for right, are necessary per level so each level is a pipeline stage that can move one element and all operations must work from top to bottom. This is likely to be the implementation with the best performance of this design, managing one *INSERT* or *REMOVE* per two or three clock cycles, depending on desired design frequency. It would, however, be overpowered for the present scenario and still suffers from an input bottleneck due to the shared interface inserting into the heap. In most situations this will not matter, if the lead for message dispatch is sufficient, but it prevents determinism at high load. A viable alternative, which would enable complete load calculation so no CPU is unforseeably stalled, would require individual queues and an arbitration scheme between them. If the policy was round robin, this would mitigate the problem, but not remove it, as early messages could delay more urgent ones. If the policy was EDF, the problem is resolved, but it would also make the heap obsolete. A heap is therefore not the best solution to the problem. A different approach supporting a scenario with multiple input ports was necessary and could possibly even provide a more efficient solution than a heap implementation.

7.4.3 Sorting Networks & Sorting Trees

The third approach investigated the properties of sorting networks and trees. Sorting networks, like a bitonic sorter, are proven to be an optimal solution for sorting a list in hardware. However, there are arguments against their use in the present case. They would come at a high hardware cost, because the 64 bit wide WR timestamps are very wide sorting keys in an FPGA context, making the necessary multiplexers highly expensive. Furthermore, timing messages do not solely consist of timestamps, but also payload. A link to this payload would also need to be moved through the network, making the multiplexers even wider, and yet another logic core would be needed to regroup the sorted keys with their payload before sending them onward.

But while sorting a list is a viable way to find the minimum, it is not mandatory. Only the next due item is of interest, which reduces the sorting problem to a minimum computation. A sorting tree can find the minimum as well as a sorting network. The comparison effort is lower, yet it leaves room for optimisation, especially for pipelined hardware implementations. The only important criteria are determinism and meeting line speed. Comparisons can be carried out completely in parallel if dedicated comparators are instantiated.

There are two factors limiting performance in an FPGA. One is fan-out, which is the number of places the content of register has to be distributed to. This has an impact on the possible maximum frequency the design can be clocked with. The other is the depth of logic networks, meaning the number of sequential logic gates, which directly transfers to signal propagation delay. In order to curb both, a divide and conquer strategy is in order. For fanout, this can be achieved by parallel instantiation of the same logic core and/or register copies, while logic depth can be controlled by pipelining operations. The number of function cores and pipeline stages can then be scaled to fit more inputs.

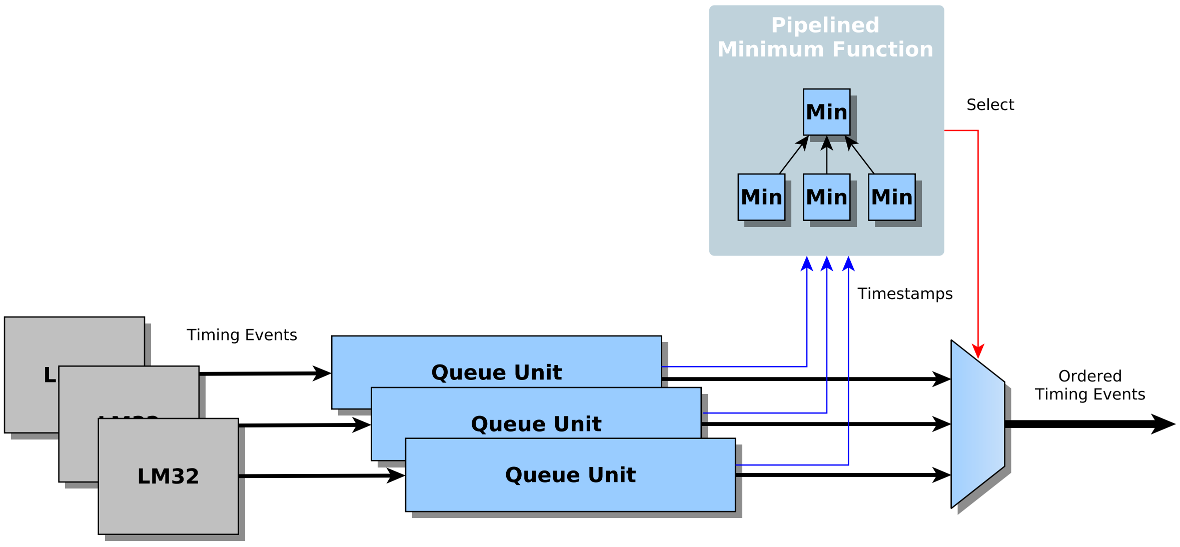


FIGURE 7.4: Overview of Priority Queue v3 Hardware

The sorting tree approach shown in figure 7.4 turned out to be better suited for a small number of elements than a hardware heap because of the minimum size of RAM blocks. It provides multiple inputs and scales better than a sorting network, as it does not try to deliver a sorted list. The lower bound for the number of comparisons necessary to sort a list shall be used as a guideline for the fitness of the algorithm. Because of the pipelining, the number of necessary comparisons is lower than the literature values [110]. Figure 7.5 shows the internal layout of a 3-1 minimum solution.

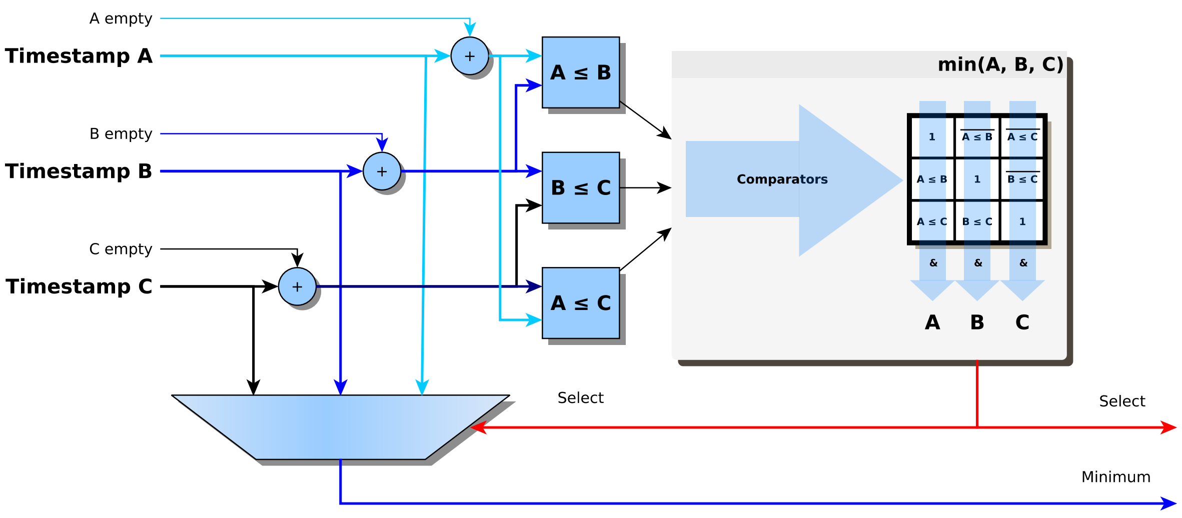


FIGURE 7.5: 3-1 Minimum Function Hardware

As shown in figure 7.4, each LM32 CPU features a local CB with a dedicated bus connection to a queue unit . The PQ arbitrates its output interface between the queue units, the selection is determined by a hardware minimum function, working on the timestamps provided by the queue units (see figure 7.6). While the multiplexed output stream is ordered, a strictly increasing sequence of timestamps cannot be guaranteed.

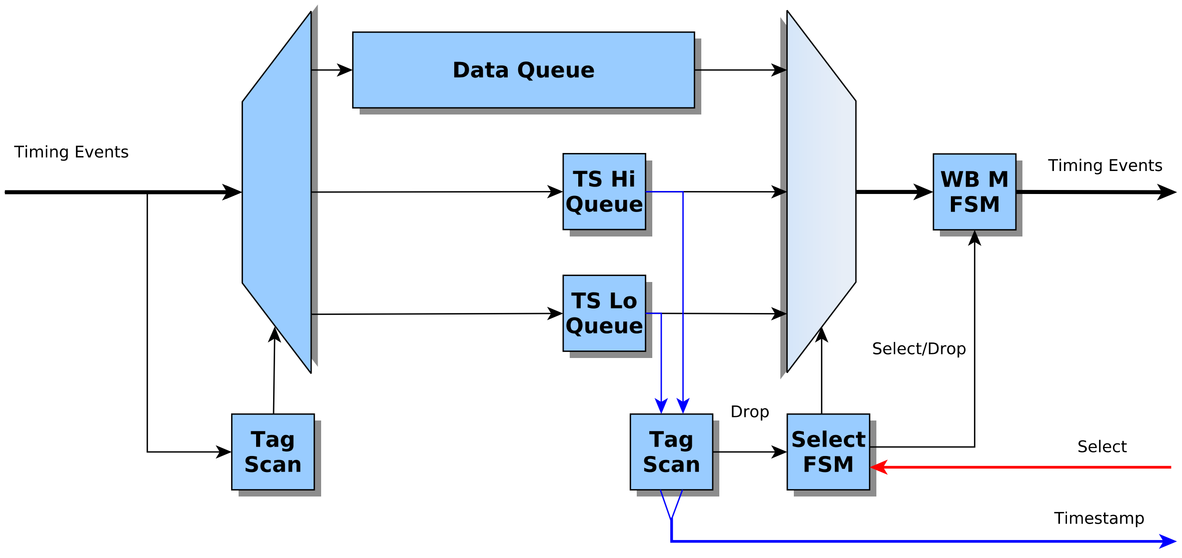


FIGURE 7.6: Hardware Queue Unit with tag-based sorting

Consider a new message arriving between selection and end of the current transfer. If its timestamp is lower than the current one, the new arrival will precede all other messages, yet the timestamp sequence decreases at this point.

The proposed solution resembles a merge sort and is designed to find the minimum of 9 64 bit timestamps within 2 clock cycles (16 ns). The memory consumption is less than half the memory resources necessary for a heap implementation of similar performance. The logic utilisation is at 1800 LUTs in an ArriaV chip significantly higher than the heap with 750 though, but one has to consider that the heap implementation could also not provide a real solution to the input bottleneck problem the sorting tree solves.

7.5 Etherbone Master

The last challenge to solve was a hardware protocol core to wrap timing messages for transport. While EB provides all the necessary functionality, so far the solution was only workable for software masters communicating with hardware slaves. The very first implementation of an Etherbone Master (EBM) used in this project was in fact a slimmed port of the EB software libraries to the LM32 CPU and first employed in 2011. While it was functional, it was slow and still required a large amount of memory (from the perspective of a 64 kB RAM embedded system). Deterministic execution was also out of the question, making the implementation in firmware more a proof of concept than a workable solution.

7.5.1 Challenges in Hardware Implementation

The need for more performance and determinism drove the investigation of hardware EBM implementations. The main obstacle was again a basic property of WB, which is being a cycle based bus. All operations have to be completed/acknowledged before a cycle can be finished. Over the network, this leads to lag stalling a local bus device, with the added risk of packet loss and therefore unacknowledged operations holding onto the bus indefinitely. For write-operations, this can be worked around, but a read operation expects the acknowledgement at the same time as the read data is returned. The round-trip creates back pressure over the whole chain. Such a behaviour can be handled easily in a software master, because all that it needs is a callback function that will deliver the data once it arrives and then the system is free to do other things. Hardware does not provide this kind of freedom, a WB master cannot create new connections while the previous connection is still open. So the problem the hardware implementation had to solve was making a clean break at the interface, achieving independence from the remote devices' response.

7.5.2 Design Decisions

The proposed solution was to design a core which presents a simple interface to the user and fully conforms to the wishbone standard [92] while at the same time having a pure feed-forward behaviour. While being aware that the DM requires only a fixed message format, the design is aimed toward a generic solution. It was not clear at the design stage if the DM might need to communicate more than timing messages, especially in the context of data exchange for calculations during the Bunch-To-Bucket-Transfer (see chapter 4).

Configuration As figure 7.7 shows, the local EBM interface was split into two areas. The first is used for configuration, such as source address for the network layer and additional meta information needed by EB [93]. Once this is set up, a WB master establishing a connection to a remote slave only needs to provide the destination IP, if it has changed.

Data The second area of the EBM interface acts as a FIFO Buffer for WB operations. WB write operations can be acknowledged directly the moment they are received by a WB slave (WBS). Our approach uses this property by requiring WB masters to rephrase read-operations to a remote WBS as writes to the local EBM. This enables the EBM to acknowledge all incoming requests immediately without waiting for the remote device, breaking the back pressure loop at the WBM's side. Internally, they are converted back into read-operations once translated into the EB protocol. The remote EB Slave (EBS) executes the read-request, gathering the replies. Once the cycle is complete, it returns an EB packet over the network, containing the replies as writes to the provided read back address. This creates no further dependencies and the back pressure loop is cut.

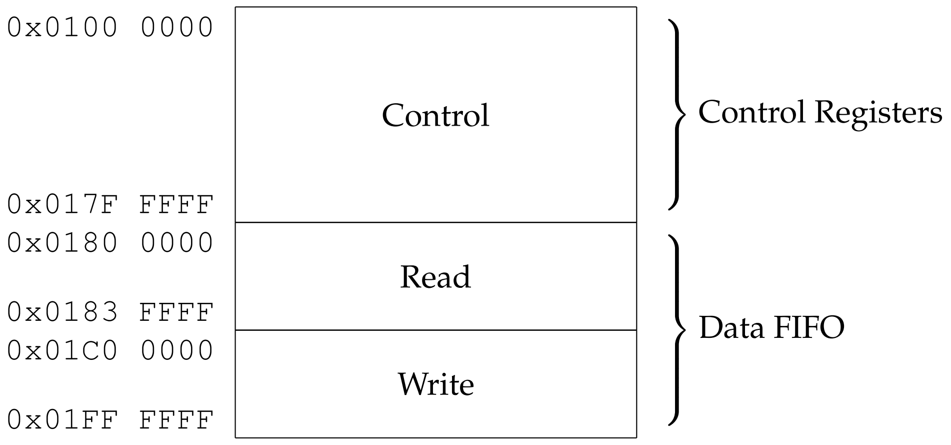


FIGURE 7.7: Example Memory Map for EBM at 0x01000000

7.5.3 Implementation

While the basic concept is sound, it leads to the problem of implementing it without breaking compatibility to the WB standard. There are three problems to solve: atomicity, the address of incoming operations and how to mark a read-operation.

Atomicity While it would be a clean interface design to separate the configuration registers from the data FIFO, there is a practical problem. Creating two distinct WBSs for the purpose effectively prevents atomic access to the EBM. This is because locking a WBS for a master requires the cycle line to be kept high during access [92]. Since it is not possible for a master to open cycles to two distinct WBSs at the same time, there would be the risk of another using the configuration interface while a

WBM is using the data interface and vice versa. This would most likely lead to a corrupt transmission. Configuration registers and the data FIFO were therefore merged into one WBS, so the EBM can stay wholly locked during access. The EBM's most significant address bit is used to distinguish between control and data area. The memory map in figure 7.7 illustrates this being *very* wasteful in terms of address space, but the design decision fell in favour of safety. The responsibility could in future be moved to the WBMs. The lock mechanism would be replaced by a semaphore register, used to show assignment of the EBM to a certain WBM by a unique ID.

Marking Read-Operations As stated before, all operations to be sent must be writes. To mark read-operations without using the write-enable bit of the WB bus, the bit is encoded into the second most significant address bit of the EBM. In the case of a WB write operation, address and value have the normal meaning except the high address bits. As already mentioned in Subsection 7.5.2, a read operation is treated differently. While the address passed to the EBM for remote reading is used as intended, the value is used as the read-back address.

When arriving at the destination's EBS, the request makes it read information from the targeted WBS. The results of remote reads are gathered and sent back as EB write operations to the initiator, targeted at the read-back address provided in the original read-request. Any WBM using the EBM thus has to provide a memory area to which the requested data can be written. Since all implemented WBMs using EBM are so far either host bridges or Soft-CPUs, this does not present a problem, as they come with their own memory.

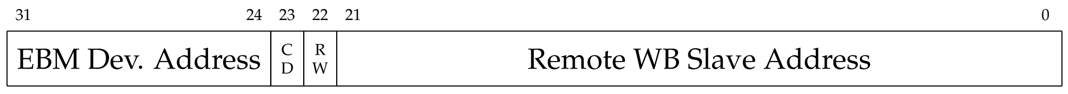


FIGURE 7.8: Example Addr. Fields for EBM at 0x01000000
(10 high bits: 8b Dev. Addr., 1b **C**ontrol/**D**ata, 1b **R**ead/**W**rite)

High Address Offset Using the address of the targeted remote slave in WB operations sent over EB is not feasible, at least not directly. The EBM's interface is itself a WB slave, it occupies a certain address. Writing any operation to the EBM requires using its bus address. The high address bits depend on EBM's own position in the bus hierarchy and is likely to be different to those of the remote slave. In addition, the two bits already claimed for control/data and read/write selection are also not available (see figure 7.8).

To give the EBM its full address range back, the control area contains a register providing the unavailable high bits of the address. They are replaced in all incoming operations by the content of this register, which the sending WB master has to provide before starting to load operations. In practice, any address inside the region mapped to the target remote WB slave will suffice. The instantiation width of the high address bit register is flexible, but always fits address space mapped to the EBM. The wider the mapped range, the more efficiently remote slaves can be addressed, because the high address register seldom needs to be updated. On the downside, the mapped range is of course not available to other devices any more.

Background Activity and Initiating Transfer While operations are loaded, the EBM is parsing them and assembles the EB content. As shown in [93], this means inserting the EB packet header, EB record headers, target and read back addresses as needed. The EBM follows this scheme for all incoming WB operations, turning them into EB records. Once the packet is completely loaded, a write to the EBM's flush control register starts transmission over the network interface.

7.5.4 Fitness for the Data Master

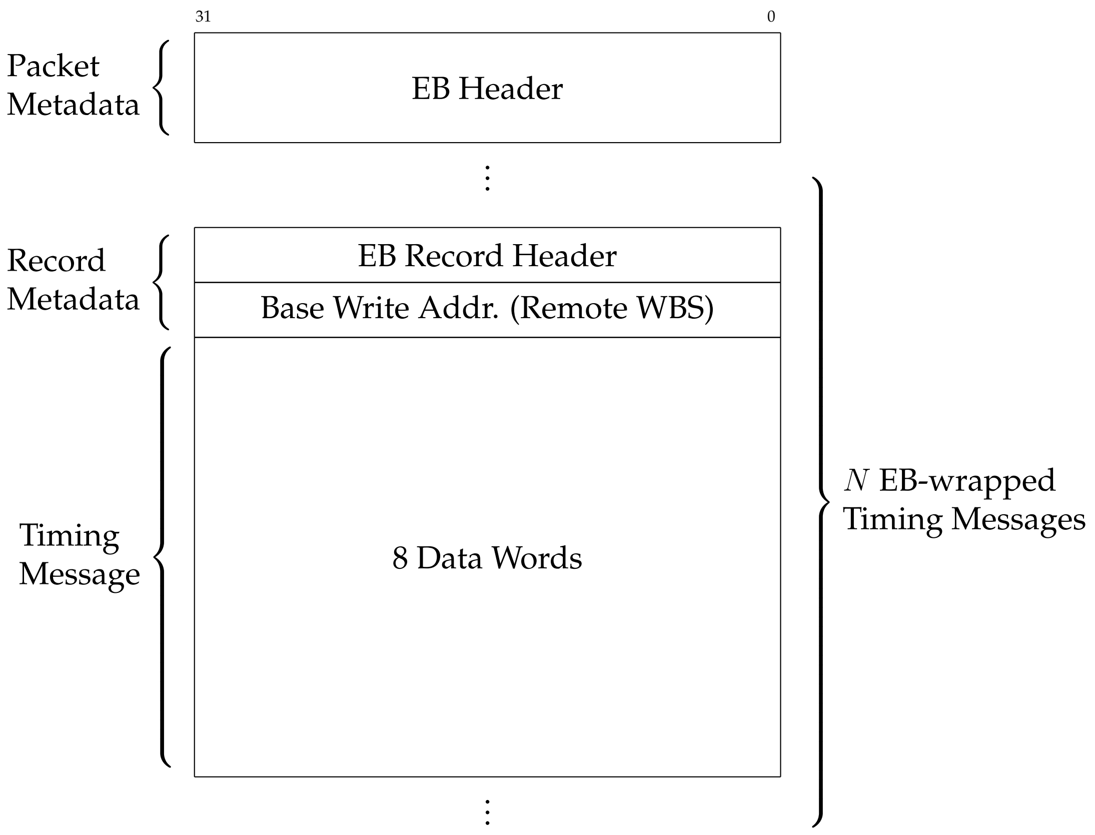


FIGURE 7.9: EBM Output for Timing Messages

In the context of the DM, determinism is of paramount importance, directly followed by speed and flexibility. The EBM's parser inserts fitting headers into the data stream depending on order, target, type and optional parameters of the WB operations. This dependency does not look deterministic at first glance, yet this impression is misleading. Figure 7.9 illustrates the context: Because incoming operations will always follow the same 8-word-write scheme for timing messages, the EBM always generates the same overhead structure. The packet generation is replicable and therefore deterministic (compare to variable EB overhead in chapter 6.9 "Etherbone Protocol", figure 6.9).

The fixed format makes the presented implementation deterministic. It is true that if meta data is known in advance, an even faster and much simpler EBM core design would have sufficed for the DM. However, there is greater value in its flexibility to also control a remote core to program settings or for diagnosis than in pure speed.

A detailed analysis of the delay introduced by the EBM and its impact on maximum bandwidth can be found in chapter 9, "Theoretical Delay Model".

7.6 Fast Input/Output Module

Control systems are meant to control machines outside their own hardware. It usually provides a generic interface that can be connected to the form factor specific IOs. In the case of the FAIR CS, these IOs are intended to either output or sample digital signals with the full WR timing accuracy.

7.6.1 Application

Input - Timestamp Latch Unit In beam diagnostics and particle detectors, it is necessary to measure values at multiple locations simultaneously. These measurements must be exactly correlated, though the amount of data is usually too big to timestamp individually. The Analogue-To-Digital converters (ADC) have their own sample clocks, continuously gathering and saving values to memory. Timestamps are used to add an epoch to each block of memory data. The epochs in conjunction with the sample clocks' intervals are sufficient to correlate all acquired values. The Timestamp Latch Unit (TLU) was developed in the context of this work to take the time when an external or internal digital signal changes and make the timestamp buffers available over EB. For this purpose, it features multiple parallel input channels with their own FIFO buffers. For use with external sources, the TLU's trigger level can be programmed to match different signal slopes and supports filtering of input glitches.

An endpoint's TLU is also employed for the verification of the CS. It samples the Pulse per Second (PPS) outputs of timing receivers at different levels in the switch hierarchy in order to trace timing accuracy.

Output - Event-Condition-Action Unit The main requirement of all timing receivers is the ability to produce digital signals on their output ports at precisely defined times. For this purpose, the Event-Condition-Action Unit (ECA) was introduced with the explicit requirement to be able to output signals at a rate equal to one timing message per clock cycle. Contrary to TLU, ECA was not developed in the scope of this thesis. However, it plays a vital role in verification of the CS implementation, as it can detect and log late message arrivals.

The basic concept of the ECA is a calendar memory. Consider a hypothetical dual port RAM with an address as wide as a WR timestamp. When a message arrives, the deadline is used as the write address to insert a command. The current WR time is constantly used as the read address of the second port. Such a calendar RAM will sort incoming messages and all commands will be executed in order of their deadline.

Sorting incoming messages might seem redundant at first, since the DM is supposed to generate the stream in chronological order. There are circumstances however, when the order of messages cannot be guaranteed. One is packet reordering in the network, the other the DM reacting to an external event.

Bridging the gap between the simple theoretical construct and a real hardware implementation made ECA one of the largest and most complex hardware modules in the CS (see red tinted area in figure 7.1, page 95). The memory must be sparse (the theoretical RAM would otherwise have a size of 2.3×10^{18} B) and mechanisms for handling concurrency, early (arrival $\leq$ deadline $- 4s$) and late messages were required, as well as multiple bespoke output channels, working fully parallel, to mention just a few of the challenges. In addition, ECA provides logging capabilities.

ECA and TLU are both of paramount importance to the CS verification setup described in chapter 10, "Evaluation".

7.6.2 High Speed IO

The WR system in its current implementation is clocked with 125 MHz which equals 8 ns per clock cycle. All form factors would therefore natively count time with an 8 ns resolution, which would also be the IO resolution for ECA and TLU. However, the requirements dictated a resolution better than 5 ns and WR provides a synchronisation accuracy of $\pm$ 200 ps. The goal was therefore to make the IO resolution match the next order of magnitude, i.e. 1 ns.

7.6.3 Increasing Sampling Rates

Increasing an FPGAs internal clocks to 1 GHz is not possible, the limit, even for very fast designs on cutting edge FPGAs in 2016 is usually reached at 300 MHz to 400 MHz. The following are several approaches on how to increase IO sampling rates without using faster chips.

Analogue Hardware - PLL In the input direction, there several are ways to increase the timing resolution without the use of dedicated hardware. One way would be to sample the same input signal multiple times in parallel, using multiple phase-shifted sampling clocks. This would return one sample value per phase offset, in effect multiplying the sampling rate.

Asynchronous Logic - Time-To-Digital Another way could be to create a Time-To-Digital Converter (TDC) circuit in the FPGA. This makes use of signal propagation delay to turn a time to be measured into a path length. Inside FPGAs, there are ripple-carry-adder chains. Consider a very wide adder having both operands filled with ones. The signal input is connected to the carry in of the least significant adder block. If flip-flops are placed as taps at the carry outputs of the individual adders, the bit pattern in the taps at a sampling edge shows how far the signal (overflow) travelled. The result is equivalent to a phase offset to the sampling clock.

Digital Hardware - SerDes While the previous approaches are certainly interesting techniques, they are susceptible to ambient influences, scale very badly, and are not usable for outputting signals. Fortunately, modern FPGAs come with dedicated hardware for communication purposes. This can be split into two individual parts, the electrical, and the Serialiser/Deserialiser (SERDES) part. The electrical part of signalling is left to Low-Voltage-Differential-Signalling (LVDS) hardware. It outputs a signal not as a voltage difference to ground, but as a voltage difference between a pair of directly adjacent terminals. This approach is much less prone to cross talk and noise, allowing higher frequencies. To make use of the high frequencies possible with an LVDS driver, an interface compatible to the slower logic matrix of the FPGA is supplied. The output at the LVDS is a fast, serial stream of single bits, the FPGA matrix can deliver many bits in parallel, but is comparatively slow. To use the full bandwidth, a shift-register like circuit is introduced, which is loaded in parallel and emptied serially for an output or vice versa for an input. SerDes circuits are used for all high-speed IO applications network connections.

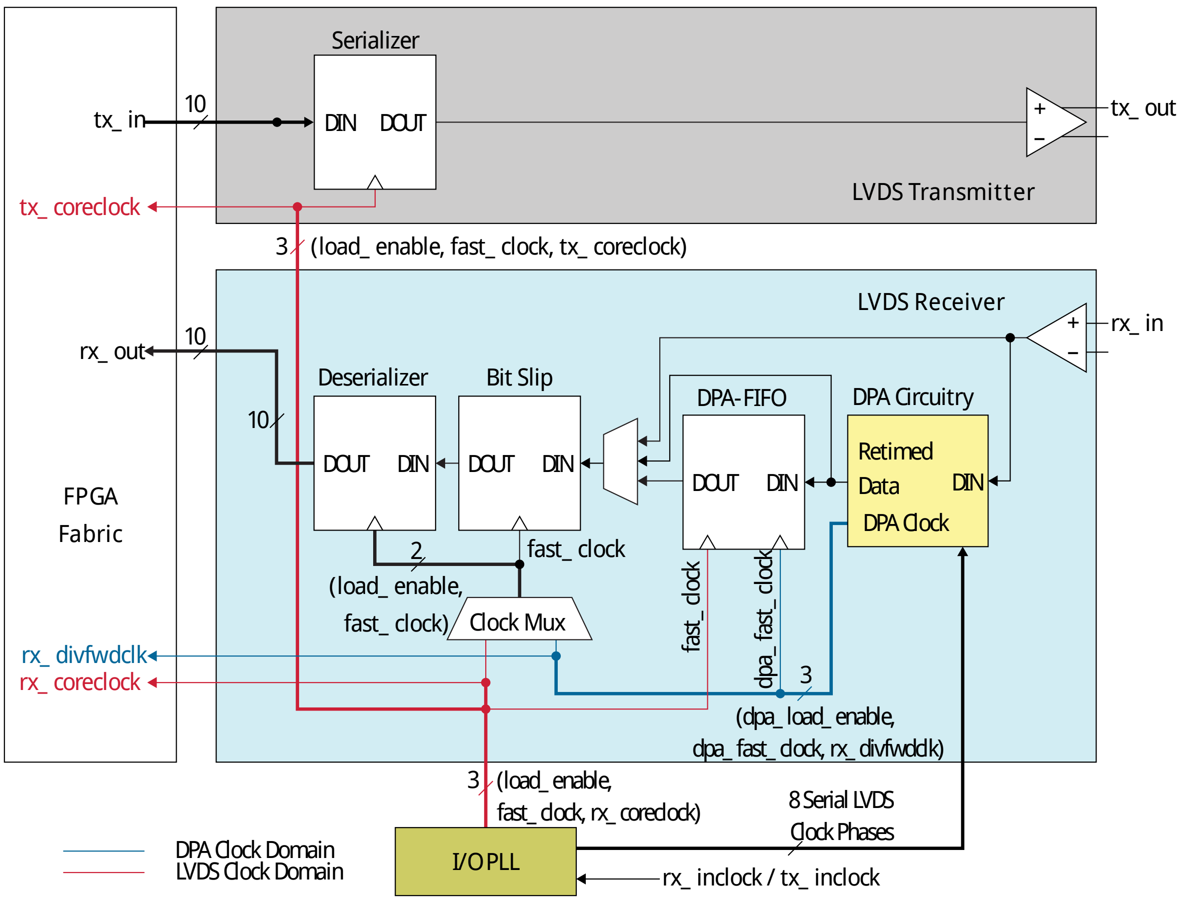


FIGURE 7.10: Altera LVDS SERDES Channel [111]

Figure 7.10 shows an implementation of such a circuit in a modern FPGA. As can easily be seen in the diagram, deserialisation is the more complex process, because it includes clock recovery (see chapter 2.4.3) and bitslip correction. This is not necessary in TLU channels, because the sampling clock is provided by WR. SerDes circuits have many advantages in implementation over trying to use logic cells. They are bespoke silicon and have short, length matched routes to the LVDS IO drivers. Within the same block of transceivers, they are also connected to a fast compensated clock distribution network. They are also shielded against crosstalk and are guaranteed not to suffer from temperature drifts within the operating range of the FPGA.

As an example, consider the following: A 2 ns wide pulse is desired at time ... 12 ns. A 2 ns wide pulse becomes two adjacent bits being set to high. The time offset 12 is not divisible by eight, the modulo of 4 becomes the bit shift. This yields the following bit pattern, put into the serialiser for use at time offset 8: $0^7 0^6 1^5 1^4 0^3 0^2 0^1 0^0$ SerDes transceivers in the ARRIA V GX FPGA can easily manage Gigabit Ethernet over fibre, which runs at 1.25 GHz (10 bit/8 ns). For TLU and ECA, they are set to 1 GHz, 1 bit/1 ns). Every 8 bit word now describes the output behaviour within an 8 ns clock period. Using the network transceivers for digital signal IO is a scalable solution, which provides the desired IO resolution of 1 ns.

Chapter 8

Data Master Firmware

8.1 Overview

This chapter presents an approach for the utilisation of LM32 CPUs as schedulers and sequencers, generating timing messages from machine schedules. In addition, schemes for the synchronisation between CPUs as well as between the host system and CPUs are introduced, a lightweight memory management scheme is presented and various optimisation techniques are discussed.

8.2 Scheduler

Figure 8.1 shows the concept for the DM's scheduler. An EDF based system is employed to find the most urgent path trough machine schedule graphs, modified by external events.

Threads The distribution of timing messages in accelerator schedules varies over different beam configurations. Typically, control patterns are repeated, and the time needed to process and dispatch messages is minute compared to the base period. This means a lot of computing time would go unused if there is only one schedule per CPU. Since the release time of a message is flexible in relation to the execution time (deadline), multiple schedules could be processed side by side. This called for the introduction of threads into the DM's architecture. A thread in computing is a series of tasks, running pseudo-parallel with other such threads. Execution only seems parallel, but is in fact sequential. It is switching between threads, granting each the use of the processor for a limited time-slice. For infinitely small time-slices, their progress is parallel.

Prioritisation Since the DM is a hard RTS, results must never be late. The problem is already solved in hardware with the priority queue (see 7.4), which assigns dynamic priorities to messages depending on their deadline. Preference is given to the most urgent task, this is called Earliest-Deadline-First or EDF scheduling. To get the maximum throughput from the system while being independent of processing order, the EDF policy shall now be applied to thread selection. It is therefore necessary for each task within a schedule to have a due time or deadline. The time when a task's processing starts is called its release time.

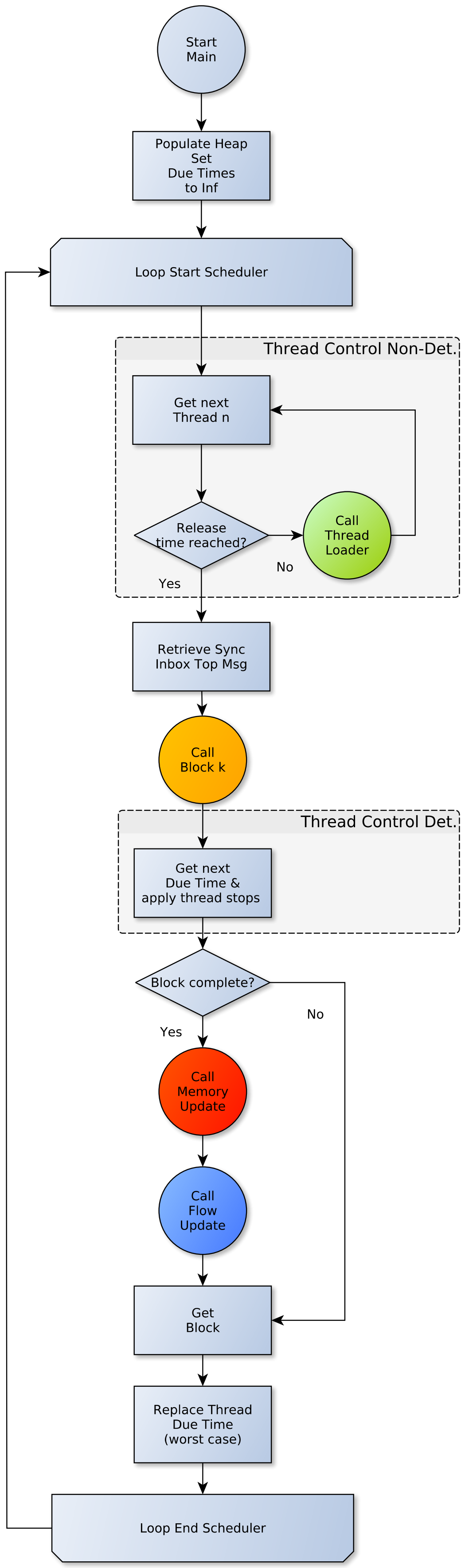


FIGURE 8.1: The Data Master's EDF Scheduler

EDF is work conserving, if there is at least one task, the scheduler cannot be idle. The environment of the DM calls for a change to policy there. All dispatched messages are irrevocable and the DM is required to provide a fast reaction time to external requests so the earliest possible release time of a task is bounded by the deadline minus the maximum delay ($500\,\mu s$). This means that the DM has to check if a task is ready for release (Figure 8.1, “Thread Control Non Det.”).

8.2.1 Sorting Algorithms

Possible Candidates The DM is a hard RTS with limited resources. It needs a space-efficient sorting algorithm with a fast performance. Since the DM is all about guaranteeing timely message delivery, we do not care about average performance, only worst case. It can be proven that comparison sorting algorithms have a lower bound of $\mathcal{O}(n \log n)$ [112, p. 191]. There are implementations among them requiring only $\mathcal{O}(1)$ space, so the data structure is the original data array without additional overhead. Non-comparison sorting algorithms like Radix Sort perform in $\mathcal{O}(n\frac{k}{d})$, where k denotes the number of bits and d the number of digits. While this is fast, the space requirement is exponential for the number of digits, $\mathcal{O}(2^d)$. Sorting networks in hardware, like a bitonic sorter, perform better at $\mathcal{O}(\log^2 n)$, with a space requirement of $\mathcal{O}(n \log^2 n)$. Newer findings by Anderson, Thorup, and most recently Han [113], show that it is possible to sort deterministically in $\mathcal{O}(n \log \log n)$ time while using only linear space $\mathcal{O}(n)$.

Requirements The DM is a non-standard case, because the number of integers to sort is quite small, yet at 64 bit, they are wide ($n = 8, k = 64, d = 64$). The set of integers is also often changed and only the minimum is of interest. As a consequence, an algorithm providing a fast *extract-min* and *replace*-operation is sought for, a fast *insert* would be a bonus. Another hardware support module like the priority queue that is aggregating message flows of multiple CPUs would be an option, since it is definitely fast and deterministic. The gain is only theoretical, though: in the DM's case, bus device access at three cycles per operation would cost almost as much as the whole sorting for a small number of integers inside the CPU. The increased physical hardware cost per CPU would be an another drawback. A solution in firmware is therefore to be preferred.

Fitness Non-comparison algorithms tend to have a higher execution time than heap sort for small integers in addition to a generally higher space requirement. Han's algorithm uses exponential trees of integers, hashing them before insertion to reduce the number of bits. While being the fastest, there are problems when using it for small numbers of integers when the set is often changed. The hash operation is costly at $\mathcal{O}(n^3)$ worst case. But even if we only consider the *replace*-operation, the individual cost of the hash function $h_a(x) = (ax \mod 2^b) \div 2^{b-s}$ is high in our scenario, since it requires multiplication > 32 b. This places the hashing, without any sorting, close to the total cost of a *replace* in a min-heap. In conjunction with the higher space requirement of $\mathcal{O}(n)$, we can conclude that the Han's algorithm is not the best match for our purpose.

Choice Binary min-heaps, being optimal among comparison sorts at $\mathcal{O}(n \log n)$, also provide excellent space efficiency at $\mathcal{O}(1)$. They provide the desired *extract-in* in $\mathcal{O}(1)$ and also grant a fast *replace* at $\mathcal{O}(\log n)$. These properties lead to the choice of a balanced binary heap for the DM's Scheduler.

8.2.2 Application in the DM

Using deadlines as keys, the top element of the min-heap is then always the thread holding the task with the earliest deadline. Whenever a task is selected and processed, its old deadline is replaced with a new value afterwards and the heap is re-ordered. This is a standard case called a *replace*-operation. It means removing the top element, inserting a new element in its place and pushing it downwards to restore the min-heap property, which has a worst case of $\mathcal{O}(\log n)$. In the DM's case, this will be implemented to always take worst case in order to avoid execution jitter. As will be explained in more detail in the following subsection 8.2.3, starting new threads requires the same effort as initially turning an unsorted array into a heap. A heap sort provides a guaranteed $\mathcal{O}(n \log n)$ worst case performance, which is optimal.

8.2.3 Additional Requirements to the Scheduler

The working of the control interface to the RTS system is not a standard way to control a scheduler, as access is fully parallel. A synchronisation scheme had to be developed for any command that interferes with the scheduler, especially the addition or removal of a thread.

Starting and stopping threads Starting a new thread is not time critical, stopping a running thread is. In the upper right box of figure 8.2, the thread loader module is introduced. As the group in figure 8.1 "Thread Control Non-Det." shows, it is only called when the scheduler would otherwise be idle. This is because adding elements does not happen the same way as replacing; new elements are pushed upwards, not downwards. As a consequence, it cannot be done in tandem with replacing. Also if more than one thread can be started simultaneously, there is no single element to push any more. Instead, a complete heap sort is necessary.

Since adding/starting threads also does not happen often, re-sorting the heap in every iteration of the scheduler would be a huge waste of processing time. The sorting algorithm must be called on demand, but without delaying the scheduler. If there is time available for a new thread to run (which can be calculated beforehand), there must also be some window in the immediate future where there is time to run the *add*-routine. On the contrary, a system running at $P = 1$ would not have time for any additional operations. Wanting to remove a thread then leads to a chicken-and-egg situation, because the routine to remove a thread needs resources, which can only be freed by removing a thread.

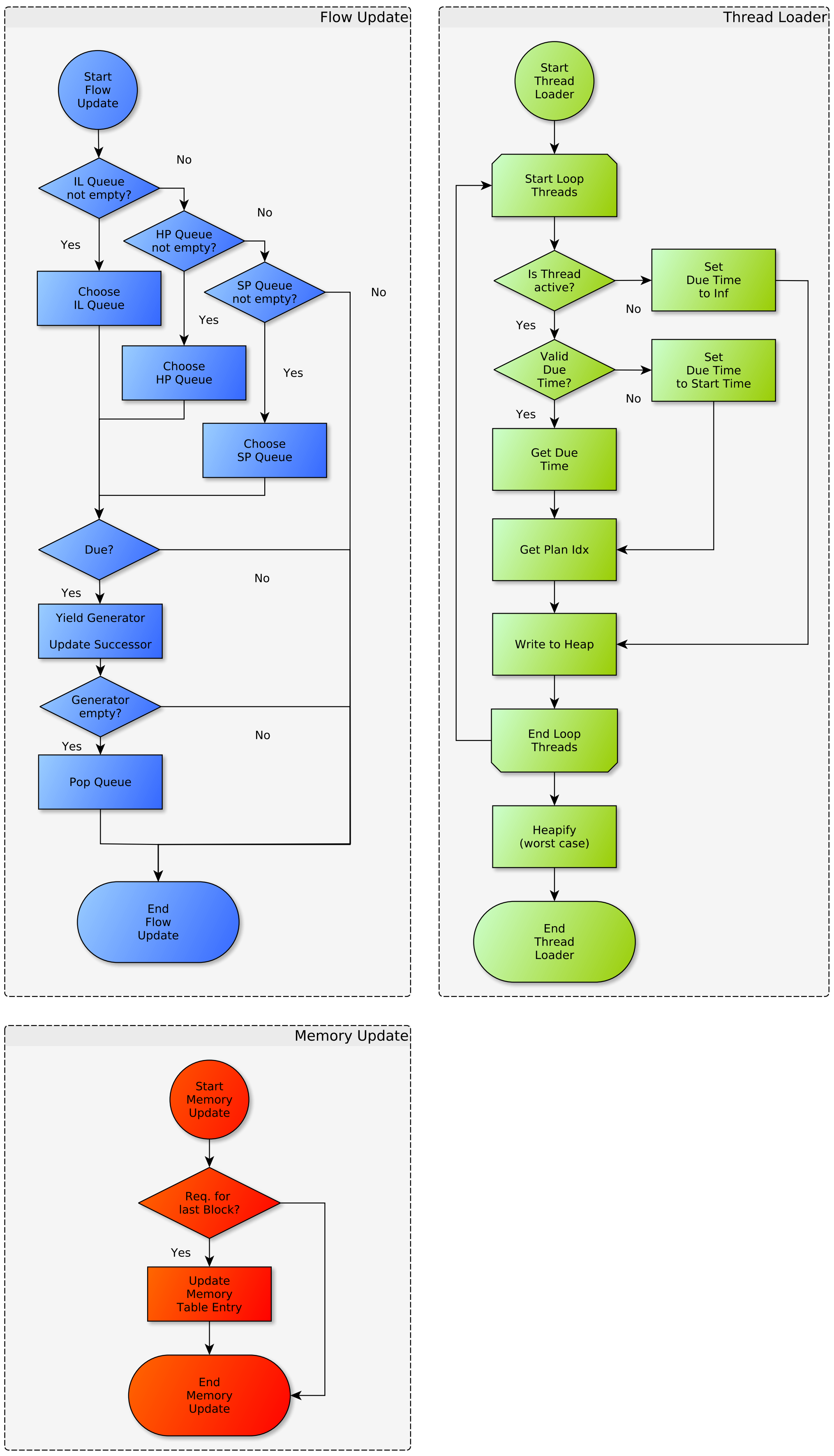


FIGURE 8.2: Sub-Components of the Data Master's EDF Scheduler

The proposed solution is to use the mandatory deadline update to remove a thread. If a tasks deadline is set to infinity (in actual implementation, this is the maximum of a 64 bit timestamp, *uint64_MAX*, year 2554), its earliest release time is never reached, the scheduler never selects it. A simple, efficient and deterministic implementation is the scheduler masking each timestamp bit with the thread's *stopped* bit when receiving the new deadline. If the thread is running, the key for the *replace*-operation is the original deadline, if the thread was stopped, it becomes *uint64_MAX* instead.

Task Length In order to judge priorities correctly, task length is considered in most schedulers. The DM is not allowed to run at $P > 1$. This policy can only be kept when the maximum workload, i.e., task length and task list, is known in advance. This is called offline-scheduling and will be discussed in detail in chapter 9. If the workload never supersedes a hundred percent, all sets can be scheduled. This implies, that at identical deadlines, the order of release times assigned to tasks is not relevant, provided a thread is prevented from monopolising the interface (compare 7.4). The execution time of possible tasks is very similar within the DM and can even be made identical for all cases.

External Events All interaction influencing the DM's path through the schedule graph will lead to a problem. Since this changes the sequence calculated in the offline schedule, $P \leq 1$ cannot be guaranteed. We will consider two possible approaches to this problem, each with its own merits and limits.

If the change is not time critical, as would an experiments request for beam be, the offline scheduling can be recalculated following the intended new path. If a change at the time of the request would create a load higher than one hundred percent, there are two options. The request can be postponed and therefore shifted in relation to all other schedules until a time where a dip in workload can encompass the new tasks. If such a state is not foreseeable within the acceptable time-frame, the request must be rejected.

If the change is time critical, like an interlock state, there is no option of postponing. This just means, however, that all time critical changes have to be part of the offline-scheduling in the first place. It must always choose the worst case (in terms of load and deadlines) of all alternative scenarios. Since the points of decision at which the path can be changed and all possible alternatives are available to the calculation to begin with, the phase space of a schedule is bounded and the worst case load over time is calculable. An approach to this problem is presented in chapter 9.3.2.

8.2.4 Summary

- Scheduler
  - All schedule elements must be tasks with a deadline
  - Deadline = block start time + task offset - handling constant
  - Tasks return their succeeding tasks' deadline to scheduler
  - Scheduler chooses thread with most urgent task by EDF
  - Binary heap implementation with worst case execution
- Blocks are opaque
  - Deadline of a succeeding task in a different block is block start time
  - Deadline is adjusted to first working element's after entering succeeding block
- All tasks lengths are known in advance
- Offline-Scheduling with $P \leq 1$ is mandatory
  - Points of decision are static parts of schedule
  - All alternatives are known at time of analysis
  - Support for limited changes during runtime
    - Uncritical changes need recalculation and possible delay
    - worst case of possible time critical changes is part of offline-schedule

8.3 Payload Programs

Machine schedules have already been introduced in the case study in chapter 4. We shall now describe the approach for a prototype implementation in the DM.

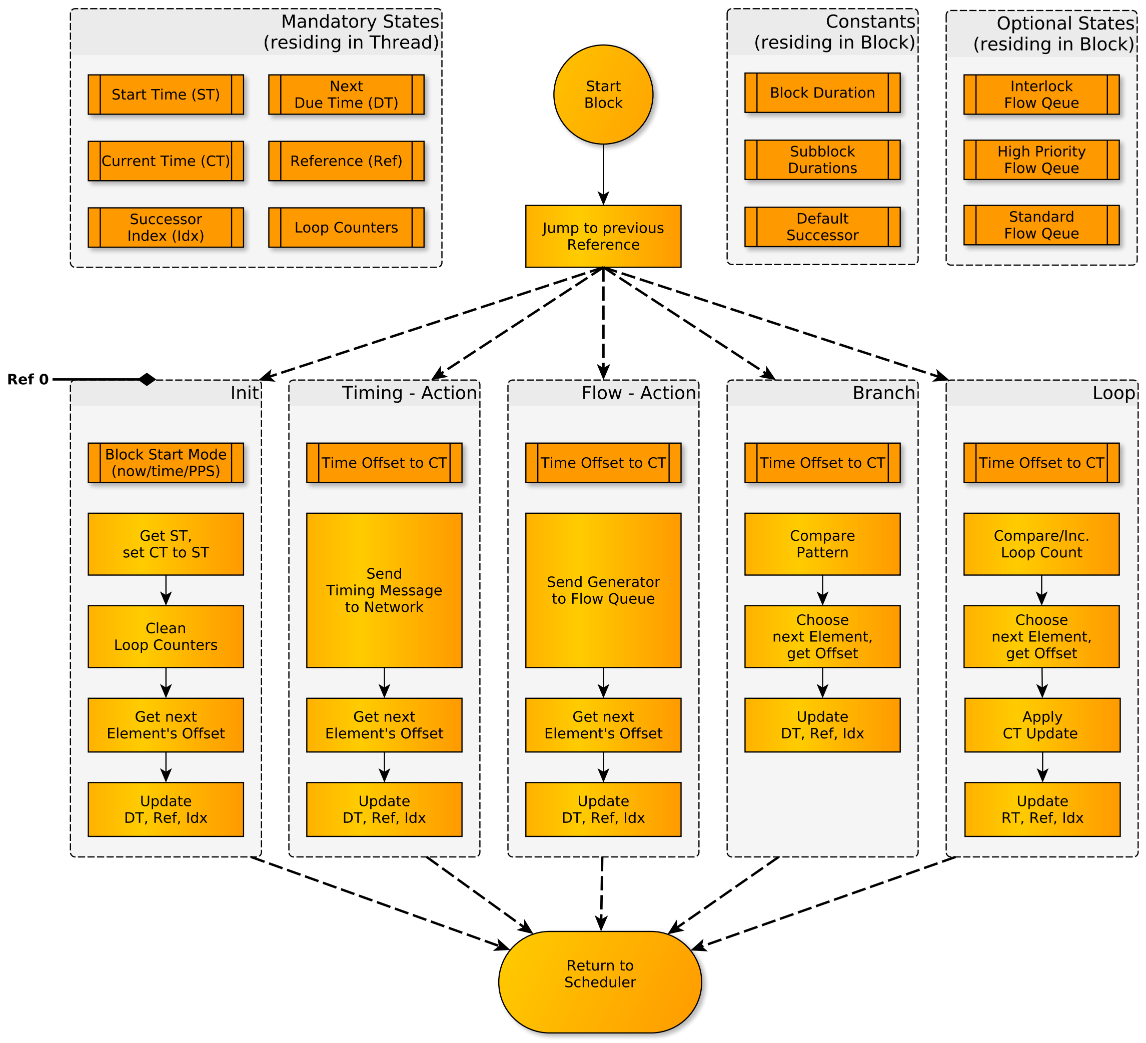


FIGURE 8.3: DM Payload Program Structure

Payload Internals The structure of a payload program 8.3 follows simple rules, it can be divided into several sub-components. The meta information on the timing block the program describes are used to determine the base time for all actions inside the program. The payload programs are meant to act as state-machines: Whenever they are entered, they carry out the next task, save a reference where they left of, and return control, and their next due time, to the scheduler. When they are called again, the reference is used for an unconditional jump to the next task. It is important that this jump stays within the scope of the payload program, otherwise it would corrupt stack and registers. As soon as the payload program is completed, the reference i.e. its internal state, is reset to its beginning. This always leaves a clean slate, which is easy to manage.

Absolute Addresses vs Position Independent Code Payload programs are live machine code that is compiled on the fly by the host system. To successfully execute them once placed in memory, there are two possible strategies. The first would be to use the knowledge of the allocator *where* they will be placed as their section offset in the linker. This creates an executable version with correct absolute addresses, yet it is not possible or at least very difficult, to move them around in memory once they are compiled.

The other alternative is to compile them as position independent code, where all jumps are implemented as the addition of a relative offset to the program counter. As long as the scope of the payload program is not left, this works well and is more flexible, yet can be slightly less efficient.

Actions Actions within payload programs are actual function calls, to eliminate redundant code. They take care of sending timing messages or synchronising with other threads or processors. The execution time is known beforehand and a constant in the DM's delay model.

Optimising Actions Function calls are not very efficient, because all information going in and out has to be placed on the stack, as have all registers which are going to be reused inside the function. RISC processors like the LM32 have a large number of General Purpose Registers, which can be a huge performance burden if a lot of registers are reused (in case of an interrupt, all 32 registers need to be saved to stack and read back later).

In the context of the payload programs, this can be turned into an advantage though. The GCC compiler allows registers to be specified per source file which shall not be touched or which ought to be used [114]. Using this in conjunction with the LM32's high register count, function calls from the scheduler can be made cheap by using a carefully specified register layout which does not overlap with the payload programs.

8.4 Memory Management

Memory management is an important field of computer science and comes in various forms. The DM will use its memory for several different purposes and it will change large blocks of memory contents frequently. This section will cover the basics of the field and discuss several memory allocation schemes [115] with regard of their fitness for the DM's hardware platform and software concept.

8.4.1 Overview

The trivial case of having larger fixed areas of memory into which one consecutive block of data can be loaded does not require discussion. While it has almost no overhead, it is very limited and wasteful. The process of dynamically assigning memory to a requester is called allocation. The allocator is the program responsible to find an appropriate area of memory for each request. There are different types of management schemes, their main attributes are outlined in the following list:

- Physical Addressing
  - Arbitrary Size
  - Pages
  - Size Trees
- Virtual Addressing
  - Arbitrary Size
  - Pages
    - Pure Software
    - Hardware assisted

Memory Fragmentation Allocation of dynamic block sizes comes at the penalty of memory fragmentation. Fragmentation arises when memory content changes from the use as small blocks to the use as large blocks. When small blocks are freed, they are not necessarily adjacent. If they are, their space can be coalesced into large blocks. When a request for a large block comes in, but only small blocks are available, it does not matter if the sum of their space is sufficient. The memory is fragmented, and as the large request cannot be split, the space is inaccessible. There are several techniques to combat fragmentation described in the following paragraphs.

Physical VS Virtual Memory The difference between both approaches is that in virtual memory, there is no case where the continuity of the allocated memory is important. All used addresses are virtual and remapped to physical addresses, the virtual addresses creating the illusion of a continuous block of memory, regardless of its physical location. Virtual memory reduces fragmentation to a minimum, but the address redirection requires extra time and also space for overhead data. It is therefore often hardware accelerated. Almost all modern processors feature a hardware memory management unit (MMU).

8.4.2 Schemes in DM context

The DM is a hybrid system of host (fast Intel CPU with several GB of RAM) and embedded System ("slow" Soft CPU cluster on FPGA (125 MHz, 64 kB RAM), dedicated hardware priority queue and network core. The detailed feature list is as follows:

- Realtime System
  - Hard Realtime / deterministic runtime
  - Very small memory per CPU (64 kB)
  - No free hardware MMU

  - User data is position independent code (no easy software paging)
  - Limited computation power

- Host System
  - No realtime
  - Computation is virtually free
  - Memory is virtually endless
  - Fully parallel access to RTS
  - Access to the embedded system has high latency

It is clear that memory is precious on the RTS and should be managed as efficiently as possible. On the other hand, it is also clear that the RTS is already very busy, and complex management schemes cost time and memory themselves. But does the RTS need to be involved in the actual management or can this be outsourced?

The following approach outlines a management scheme for the host system with minimal impact on the RTS.

Fixed Block-size The simplest scheme possible would be the assignment of several pages (continuous blocks) of memory with fixed sizes. Each page can store a machine schedule and be activated and deactivated as needed. While being the solution with the lowest overhead, the drawback is the obvious lack of flexibility. Differing schedule sizes will lead to internal fragmentation, the total number of schedules is limited and there is a hard limit to schedule size.

Dynamic Block-size Dynamic allocation of memory to different schedules is therefore a better use of resources. In its simplest form, overhead consists of a pointer to the block and its size. If one assigns blocks of arbitrary sizes, lists keeping track of all free and used blocks must be kept. A scheme to find fitting free space for all new allocation requests is necessary. To regain fragmented memory, adjacent free blocks must be coalesced again.

Allocation Algorithm Wilson et al. [115] provide a comprehensive overview of schemes for dynamic block sizes. An approach using the "Best-Fit" algorithm can deal with an arbitrary number of schedules of arbitrary sizes and produce relatively low waste of space at reasonably low additional space requirement. The algorithm showed slightly better results in simulation than "First-Fit". The "Buddy"-Algorithm has a better fragmentation behaviour than both previous algorithms, but the overhead is higher. For this implementation, the actual research of allocation algorithms was cut short in favour of easy implementation and performance over fragmentation avoidance.

However, the actual management still has to be done, and in the present case, the RTS is not the best place to do it. Memory can be accessed fully parallel from the outside, which suggests that the host is in a position to manage the RTS's memory.

If the management is outsourced to the host, the chosen management scheme can be computationally costly without any side effects on the RTS. Memory overhead on the user side (the RTS) must still be low for obvious performance reasons.

Another limit to complexity on the host was introduced for safety reasons. The host, a high end computer with an OS, is more complex and therefore more prone to error than the RTS. Because the RTS can run independently once it is set up, a reboot of the host should not require a reboot of the RTS. Instead, the host should to be able to determine the complete memory allocation from the RTS, instead of its own persistent records. The scheme is therefore limited to overhead data which can *completely* be deducted from the minimal overhead data on the RTS. In addition, a commit mechanism is necessary so the hosts actions are synchronised to the RTS.

Countering Fragmentation The mandatory first step is combining adjacent free blocks. The DM cannot conduct defragmentation, i.e., moving live blocks so more free blocks can be combined. In a non-realtime system, this could be done if the system is idle, but a hard RTS with external input cannot enter this state without being at risk of needing to handle an urgent request from an external event. Half finished cleaning work is not viable.

Defragmentation could be achieved by the host system though, under three conditions:

- Inactive live blocks are moved first to create swap space
- Active live blocks are only moved as a whole, i.e., create copy, redirect RTS, free original
- Swap space must suffice to create a copy of the largest active live block

Implementing and integrating a defragmentation algorithm for the host was outside the time-frame of this thesis. The first implementation stage will deal with the problem by writing a compacted version of the fragmented live blocks to an idle CPU's memory. Operation is then synchronously passed from the fragmented to the fresh core.

Virtualisation and Pages The last option to consider is virtualisation. Virtual Pages combine the advantage of fixed page sizes, no fragmentation, with the advantage of low memory space wasted. While it is not necessary to implement the controlling logic in the RTS, the Virtual Address Table (VAT) would require extra hardware. Address translation units are by definition content addressable memories. While the generic implementation of such a module is not trivial in programmable hardware, in the special case of address look-up, it is quite straightforward.

The contents are page addresses, all of which could be mapped, therefore the VAT must provide space for all of them. The content set is then continuous and bounded to the VAT's own range. This means all content addressable memory logic can be omitted and a normal RAM be used instead, using the virtual page addresses as the RAM's address lines. The VAT is directly inserted into the normal memory's address lines, from the address' MSB down to the page alignment boundary. If only a certain region of the RAM is to be virtualised, an additional multiplexer is required. If the VAT is programmed correctly by the host, all virtual page addresses are translated into physical addresses, the RTS' CPU never knowing about the redirection. Figure 8.4 shows an example for a 32 bit aligned 64 kB RAM with 1024 pages of 64 B each. The maximum wasted space at 63B per block would be marginal.

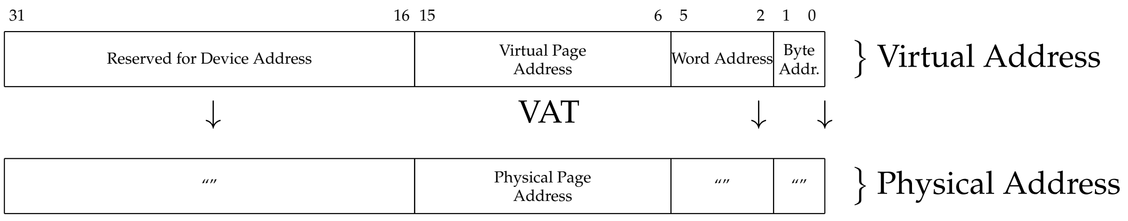


FIGURE 8.4: Virtual Address Translation

While technically elegant and interesting, this solution also has its drawbacks. The implementation introduces at least one more cycle delay to memory access, more if the memory is connected via WB instead of directly to the CPU. The look-up tables require extra memory resources per CPU. The extra development effort for building and evaluating an MMU prototype was outside the time-frame of this thesis, yet a future assessment should be considered.

8.4.3 Summary

A simple scheme of dynamic memory management was chosen for the prototype implementation. The advantage mainly comes from the ability to cope with the higher numbers of machines schedules, which all require their own block in memory. A *Best-Fit*-Algorithm for arbitrary sized blocks is employed, free blocks are coalesced if possible. To keep the RTS deterministic, scan, allocation and maintenance of free lists have been moved to the host system, making use of its parallel access to the RTS' memory. To preserve the information for the host and give a synchronised access, the overhead data on the RTS side consists of a single bitmap showing live entries in a table, populated with pointers to blocks and their size. Changes to existing blocks are to be communicated over a synchronised queue, so the RTS can update the address to a block once it has finished processing it.

For standard operation, the RTS only ever needs to access the table of pointers to live blocks assigned to threads. The chosen strategy is fast, flexible and has minimal overhead. It is however susceptible to memory fragmentation. The host currently does not feature a defragmentation scheme, yet this problem could be remedied in future work by a suitable algorithm or switching to virtual paged memory. For now, the problem can be worked around without stopping the RTS by copying a compacted version to a different core and synchronously transferring operations to the new core.

8.5 Considerations on RT Flow Control

As already outlined in chapter 4, the graph representing machine schedules is not a simple list of timing messages. The individual blocks contain messages and are not meant to change at runtime. Yet some part of the schedule must be able to be changed as needed to react to new requests and interlocks. This was achieved by treating the succession of blocks as a graph, in which the flow from one block to another can be changed at runtime. The encountered problems in RT flow control and detailed solutions are discussed in this section.

8.5.1 Points of Decision

The links between blocks are the core control mechanism of the system. The succession of blocks depends on requests i.e. "Give me a shot of uranium beam now!") and interlocks i.e. "Synchrotron not ready for extraction!". If there is a pending request and no interlocks are prohibiting it, the corresponding branch in the graph will be followed, its blocks executed. The only information needed by the RTS is the chosen path whenever the flow graph is branching out into two or more destinations. The problem is to achieve non-blocking, concurrent, yet if necessary, synchronised behaviour.

It is evident that a block's successor must only be called after the complete block is processed, no matter when the link to the successor is modified. Otherwise an inconsistent mix of two blocks would be played out. The same holds true for an updated version of a block. This is easy to arrange if the change is applied not by the host, but the RTS. It is further evident that due to the parallel nature of the access, the host system cannot have accurate knowledge about the progress in the graph. It can tell which blocks lie in the "far" past, but cannot get reliable information about the most recent, current, and future blocks. A command has to refer to the right block with the right instruction at the right time (the latter being important in repeated executions).

8.5.2 Queues

Analysis A link is defined by predecessor and successor. A command must therefore only be applied to a certain block and has to carry an identifier to ensure this is the case, and the command must also carry an identifier naming the desired successor. The right time is a more complex issue. The RTS can identify the end of a certain block, but what if this block is processed more than once? This deliberation, in conjunction with the problem of the host only having a vague notion of the progress through the graph, implies that the host should be able to deliver a stack of commands at once, the RTS applying them when due. This forms a consumer queue at the RTS, which will consume a command each time it comes by the appropriate recipient.

Queue Size The number of required slots in a queue depends on the number of changes necessary. We'll consider several use-cases to narrow down the possible choices, which are:

• Single Change	$A \to C$
• Infinite Loop	$A \to A$
• Repeat n times	$A \to A, A \to A, \dots A \to B$

8.5.3 Order

A thread's consumer queue consumes the top element only under the condition of finding the right recipient. Consider a loop of n blocks. The task is to re-program a block's link, so the loop is broken as soon as possible and the thread moves onward to a different successor. So we can chose a block J and change its successor from K to the desired alternative, X. The command is entered into the queue, and after a time $0 \leq t \leq nT$, the RTS will come around to process J again. Once it passes J, the command is consumed and applied, the loop is left towards X. This worst case time cannot be shortened by addressing multiple blocks. Because only the top element will be matched, the queue will stay blocked until the element is consumed.

Now consider instead an infinitely long string of blocks. The task is again to re-program a block's link, so the graph leads to X. Which block should be chosen? The host cannot know which blocks are not yet processed. Choosing an already processed block would block the queue forever and have no effect on the path, so the only truly safe choice would take infinite time to reach. Both examples show that the possibilities offered by a single queue pre thread are not sufficient for the present case.

There are at least two possible solutions to the problem. The first would be to include a conditional action in each block along the graph from the very beginning. If all conditions monitor the same signal (memory location) A and change their block's successor to X once they see A, only the currently processed block would be reprogrammed, and the destination reached in minimal time. The second is to allow for further parallelism. If there was not one queue per thread, but one per block, the problem can be solved by writing the command to multiple blocks simultaneously, the next time the RTS passes, the path towards X will be followed.

Another, tough minor drawback also comes into play: it is hard to allocate an appropriate size to the queues without wasting memory.

8.5.4 Pre-emption

However, this approach so far only covers *pre-planned* changes of behaviour. What isn't covered yet is an expected, yet spontaneously occurring behaviour i.e. interlocks. If an interlock condition arises, the RTS has to react as fast as possible. This would, however, mean that every block would need at least one conditional action or must feature a queue. But even if it does feature a queue, what if there are already other queued commands? Interlock commands must be executed first; they need to bypass, pre-empt [85], all other commands. This could be achieved by flushing the queue once an interlock command arrives, but this approach in turn causes multiple problems: Firstly, the host is not synchronous, it cannot safely manipulate the reading side of the queue to flush it. Secondly, the RTS will only inspect the queue every so often, so the interlock command has to wait somewhere for processing. What happens if there is more than one incoming interlock command? The answer "They go into the queue" does not suffice, because it is unclear on how the queue flush should operate if there is already another interlock command present. This leads to the conclusion that there should be at least two queues of different priorities per block. An arbiter must choose the highest priority queue that contains elements.

8.5.5 Flushing

Having implemented a pre-emption mechanism, the question remains if there is another case that requires flushing a queue. Continuing the previous behaviour directly after emergency handling might lead to unexpected results, so the answer is probably yes.

The flush command must be of a higher priority than the commands to flush, as waiting in line would defy the purpose. Flushing the whole queue, on the other hand, could also be problematic, because the flush must be synchronous to the RTS. If the host has lower priority commands to enter after the flush command, it is impossible to know if the flush command is already carried out or if the flush will also remove the newly inserted elements. It is therefore necessary to control the scope of the flush.

When taking a closer look at the low-level implementation, a concept presents itself. A queue is likely to be implemented as some sort of ring buffer. It has a read and a write pointer, which usually have the possibility to overflow once. If the pointers are equal, the overflow shows if this means the buffer is full (overflow differs) or empty (overflow equal). The read pointer, wielded by the RTS, can only ever increase to the write pointer, wielded by the host. If the host memorises the last writing position before the flush and includes it in the flush command, the RTS can flush up to exactly that point by setting the read pointer to match. All following commands will be treated normally.

8.5.6 Repetitions and Redundancy

As in the example given earlier, the case of repeating a command n times is quite common. The problem is that this requires large queues, because there is no absolute limit to n provided by use-cases of machine schedules. It is likely to be below 100, but even this number could be problematic, as the total number of blocks requiring queues is also not limited. Since the memory of the embedded system on the other hand is severely limited, it is necessary to search for a more sophisticated solution.

A possible solution are generators [116], which are pre-configured functions with a changing result. For example, consider a generator for the numbers from one to ten. Each time it is called, it outputs a single number, in increasing sequence. Once it hits ten, it returns nothing when called.

If a command would not be an element, but a generator of elements, it would output the same command until it reached its limit, and can be popped from the queue once it is empty. This would greatly simplify the loop scenario $10 \cdot A \rightarrow A, A \rightarrow B$, especially because this allows to severely limit queue sizes to a range between 2 and 5 elements.

8.5.7 Synchronisation

Simultaneous Commit Sending a command means accessing a CPU's memory and writing this command to the queue of a specific block. When writing to multiple destinations, atomicity cannot be guaranteed. While one could implement ways to truly broadcast the operations in parallel to multiple bus devices (CPU memories), there is no way to atomically write to multiple areas inside the same CPU's memory.

Since the host does not have accurate information about the RTS's current state, there would be no way to anticipate the execution order of commands sent. What is needed instead is a handshake or commit event, after which a command becomes valid for execution. Unfortunately, distribution of such an event would suffer from the same penalties as command distribution does.

The approach taken to avoid this introduces a commit date, attached to all sent commands. WR Time is globally available to all CPU's and their threads via dedicated hardware. If a queue's arbiter would check if the commit date is valid before executing a command, a synchronised execution of multiple commands can be achieved. When sending commands as a host, an educated guess is enough to calculate a safe lead for a commit date.

Sequential Commit The proposed commit date as a synchronisation mechanism also has a second important application. When controlling the accelerator facility, a beam traverses multiple stages (Source, LINAC, Synchrontron1, Synchrontron2, ...), which are managed in parallel. While it would be possible to merge all timing messages into a single chain of blocks, this would make pipelined operation of the involved resources very rigid.

The use-cases are flow changes, exceptions to or cancellation of queued changes, and emergency changes (interlocks), in increasing priority. This requires three individual queues per command queue instance, always returning the element with the highest priority available. Higher priority queues shall be able to receive commands to clear a queue of lower or equal priority up to a certain point. On equal priority, clearing cannot happen out of order, the command waits in line.

8.5.8 Summary and Command Queue Design

Drawing the conclusion from the previous deliberation leads to a design for a system of software queues. Their purpose is to allow the host synchronous online control of the RTS's path through the schedule graph. Since dual port RAMs are used, non-blocking parallel memory access by host and RTS is possible. The command queue system must also work for processors each with multiple threads. The mechanism shall be the same for host to CPU and CPU-to-CPU communication. The execution order of commands at each point of decision in the graph is local only, it is independent from all others. Therefore, each block containing a point of decision (branching sequences in figure 4.5) must have a dedicated instance of a command queue (CMD-Q).

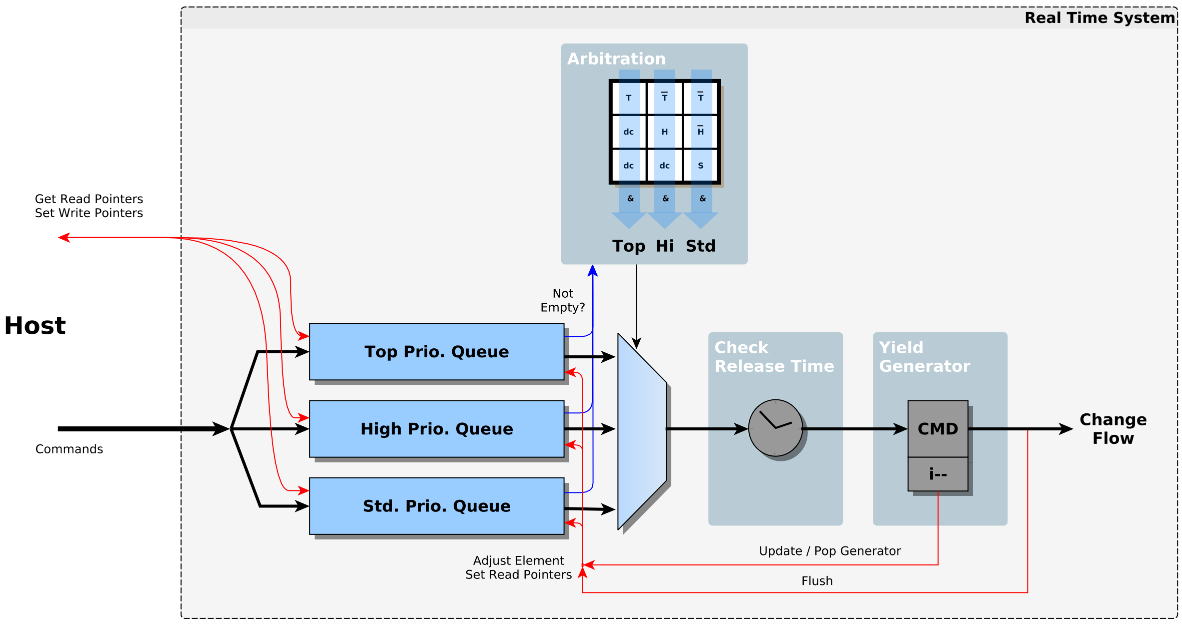


FIGURE 8.5: CMD-Q Layout: Three prioritised generator queues, time aware, partial/complete flush

The common denominator to coordinate more than one flow change is global WR time. Because dedicated hardware is involved, time is a shared notion, but access is not a shared resource. All commands therefore have a commit date indicating when they become valid. In order to save on redundant commands, commands are in fact command generators, issuing a command a given number of times before leaving the queue.

- Command Content
  - Priority – Priority level of this command (Top, High, Std.)
  - Commit – The time when this command becomes valid
  - Action – Action to be carried out:
    - Flow – New Successor and Generator Quantity
    - No-Op – Placeholder and Generator Quantity
    - Flush – Target Queue(s) and Range(s) to be flushed

8.5.9 Detailed DM Command Specification

Commands will be distributed over the WB bus, which is in turn extended via EB to the host platform. The communication must be completely non-blocking, so the external access to another CPUs memory may not be shared between host and CPU. Instead, CPUs shall use the MSI bus instead of WB. Both buses use the same protocol, yet MSI is used solely for interrupts to master devices [117]. Both the address and the data lines of WB are used. Figure 8.6 shows how the address is used to chose CPU, CMD-Q, priority and write pointer of a command. Each factor adds a distinct offset, leading to a unique memory address.

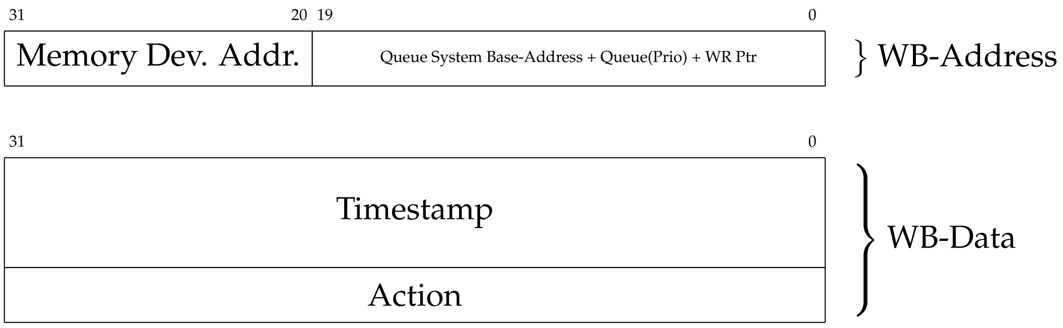


FIGURE 8.6: DM Command Message

The current MSI subsystem, which is a dedicated WB bus, is yet only meant to send a single 32 bit word per message. This is a convention though, not a limitation. The situation could either be remedied by changing this limit to 3 words per message, as shown in figure 8.6 and 8.7 or use a more compact format (figure 8.8).

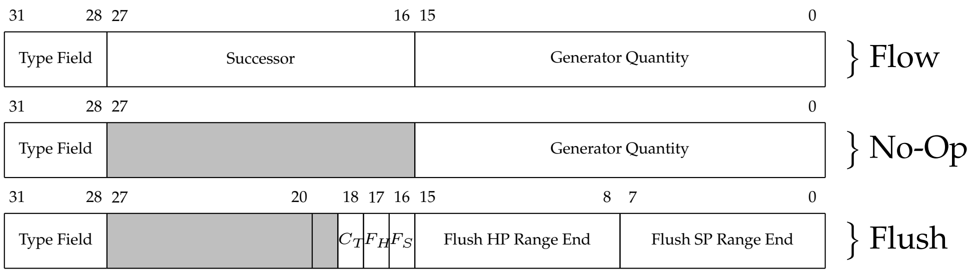


FIGURE 8.7: Possible Actions for DM Command Messages

Since the commit dates signify a "not-before" boundary and focus on equality or being inside a larger safe zone, using a dynamic representation with mantissa and exponent is possible. Sacrificing accuracy for dynamic range, it is possible to fit all necessary information into a single MSI message. Figure 8.8 shows the shorthand form usable over the MSI system.

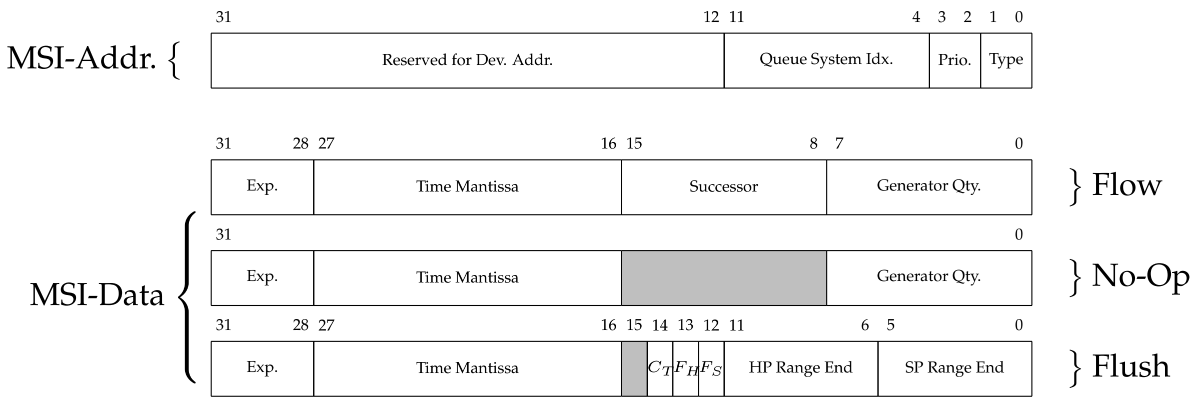


FIGURE 8.8: Alternative Short-Hand for current MSI System

8.6 Deterministic Programming

Producing deterministic programs, that is, deterministic in terms of their execution time, is not a trivial task. It can easily be observed using suitable measurement tools like profilers or logic analysers, that most programs will jitter heavily. This is usually a desired effect: the average case is faster than the worst case and far more likely, therefore saving processing time.

In RTSs however, this makes it all the harder to give guarantees for execution. Because algorithms are usually chosen to give optimal average performance, their worst case performance is often so bad that it is useless as a guarantee in the time frame available for the control loop. The DM's hardware architecture showed how shared resource bottlenecks were avoided in the design process, section 8.2 gave an impression on how to chose algorithms by worst case performance. This section deals with removing jitter from the firmware implementation and discusses various techniques to achieve this.

8.6.1 External Sources of Non-Determinism

Shared Resources One of the most influential factors are external resources the CPU tries to access over the bus. Any slow or busy resource will cause a delay. In some cases, like RAM, this problem can be solved by outsourcing both the work and the associated wait times to dedicated hardware. DMA controller and caches are the answer to this problem in modern CPUs, the processor is free to do other work. However, for a CS, this merely slightly down scales the problem of non-determinism. What about hard deadlines? If data has to be retrieved at a certain time and evaluated inside a close time-frame, access to external resources still is a bottleneck.

This is exasperated if the resource is shared between several masters. If the resource is controlled by an arbiter or all accessors adhere to a strict policy, shared use is possible in a realtime context. But whatever scheduling algorithm is used in the arbitration, one problem still remains. If a CPU is to avoid jitter in its program execution, then it always has to assume the worst case access time and wait until it is over. This comes at a high cost in computing cycles.

Atomicity All synchronisation with external processes, in the DM's case the host system, come with some sort of semaphore or handshake. Albeit a conclusion from the preceding paragraph, it is important to note that the RTS's behaviour on synchronisation can never be blocking. Semaphores must be checked in a strictly periodic manner, always returning control to the scheduler. Read and write access to a semaphore must of course also always be separate structures in order to be independent.

Caches All modern CPUs have to fight one common problem. While system memory is vast, it is also slow, from the CPUs perspective. This is countered by several layers of faster cache memory, holding content ready for faster access, prioritised by frequency of use. When a CPU mispredicts a branch taken in its program flow, this leads to a refill of the pipeline and is usually associated with the wrong content being in the cache. This takes time to refill, and leads to non-deterministic behaviour. Since the DM's CPUs reside within an FPGA, memory connection can be direct with 1 cycle access latency, which is equivalent to L1 cache. In the current implementation however, memory is implemented as WB bus devices. Bus access is slower, so LM32's do have an instruction cache enabled in order to speed up performance.

If the DM loads and unloads blocks of live code in its most recent incarnation, this means a cache flush is necessary each time new code is loaded, because the CPU would otherwise be blind to the change in memory. This would contribute a big portion of non-deterministic behaviour and is not tolerable. The long-term solution is to change the hardware implementation in the FPGA so that only one port of the dual port memory is connected to the WB bus via a standard WB interface, the other is connected directly to the CPU. Because this exceeds the time frame of this thesis, the workaround is to disable the CPU's instruction cache and accept the lowered performance in exchange for deterministic behaviour.

8.6.2 Non-Determinism through Program Flow

The main source of non-determinism inside a CPU may sound trivial, but is an important field of research in computer science. Whenever a branch is taken, the following instructions are different from the ones if the branch had not been taken. Since all modern CPUs pipeline instructions to maximise the utilisation of their ALU and specialised math cores, sudden changes of plans are a problem. The CPUs instruction pipeline has to be flushed (at least partially), and refilling the pipeline takes time before execution can proceed. For this reason, branches taken consume more time than branches not taken. Since Branches usually evaluate the processed data, it is impossible to tell at compile time how often a branch will be taken. This means online branch prediction is an important aspect in all modern CPUs and usually implemented in hardware. However, even good predictions are useless to an RTS which needs a tightly bounded guarantee.

Loops Quite often, user data, e.g. the number of student records in a table, determines how often a loop is executed. All that can be said about the execution time of the loop is that it will take n times the number of cycles it would take if it was called only once, n being the number of students. Any deterministic implementation would need to assume the worst case, i.e. the maximum number of students permitted in a year.

Branches Just as loops, branches introduce uncertainty. There are two possible cases: equality and different length, there being a corner case of one part having a length of zero. If both parts of the branch are equal in execution length, execution time would not matter, but the chance of this occurrence, if not by design, are remote. If both parts are of different length, the execution time of the chosen part will be added. If the branch only has one part (*if* ... *then* ..., but no ... *else* ...), then the branch is either taken, adding its execution time, or not.

Cases Cascaded branches, also called *Case*-statements, are implemented as a cascade of *If*-statements. While it is possible to make simple branching take equal time no matter if the branch is taken or not, this does not work if there is more than two alternatives. Not only does it matter which of the scores of alternatives are taken, the execution time also depends on the order of *If*s. When a branch on the first *If* inside a *Case* is taken, it adds the time for it's own execution. But if the same branch condition would instead be the last to be tested in a series of *If*s, this would add the execution time of *all* previous condition tests. It is only possible to evaluate multiple conditions deterministically if the execution time is worst case for all possible paths, which is a complex, and wasteful, construct.

8.6.3 Proposed Approach

When optimising code for deterministic execution, the abstract view on code is left behind and the optimisation takes place in the machine language used by the CPU. It is therefore platform specific, because the processor architecture (instruction set, register layout, pipeline design, cache) determines the operations and the time per operation. Optimisations on this level make use of arithmetic or logical properties of the involved data structures.

Calculation of Program Execution Time The execution time of a program can in theory be calculated, if its machine code and the time it takes the machine to carry out each individual instruction is known. This only holds for trivial programs though. Above a certain complexity, the question if and when a program terminates can no longer be solved by a generic formal approach, as it can be reduced to a variant of Turing's Halting-problem [118]. In the DM's case, the RTS firmware is designed highly modular and has a simple flow with a short period. This allows exact measurements of execution times. The results can be kept up to date with the current version much more efficiently than a formal proof.

Profiling The term "Profiling" in computer science means measuring the execution time of programs. Profiling is useful for finding bottlenecks and the comparison of different approaches in terms of performance. It is usually only applied to a single algorithm or section in a program at a time in order to isolate the possible sources for performance loss or gain. Most processors feature an internal cycle counter, a special purpose register, which is increased with each clock cycle. The LM32 has a 32 bit wide cycle counter and runs at 125 MHz.

The problematic aspect of profiling is that the act of observation changes the result. If profiling is done internally, then the code which times a program's execution adds its own execution time to the final measurement and must be subtracted later. This implies that the execution time of the critical section of the profiler has to be constant and known. Profilers in embedded systems have to conserve memory, so they usually only offer minimum and maximum values in addition to mean values created from a rolling buffer. The profiler code for the DM prototype follows this simple design.

8.6.4 Applied Techniques

In this section, several of the techniques used are explained and their validity shown. An excellent, if not the best available, source for clever bit manipulations proved to be S. E. Anderson's website on "Bit twiddling hacks" at Stanford University [119]. When using Boolean equations, the following symbols and notations will be used:

$$\begin{aligned}
\neg A &\quad \text{Negation} \\
A \wedge B &\quad \text{Logical } \mathit{AND} \\
A \vee B &\quad \text{Logical } \mathit{OR} \\
A \oplus B &\quad \text{Logical } \mathit{XOR} \\
\neg A = \overline{A} &\quad \text{Shorthand Negation} \\
A \wedge B = AB &\quad \text{Shorthand } \mathit{AND} \\
A \oplus B = (A\overline{B}) \vee (\overline{A}B) &\quad \mathit{XOR} \text{ expansion} \qquad (8.1)
\end{aligned}$$

Accessing Packed Bits When using smaller data types like bytes, the compiler will usually pack them in memory. Most memory types have byte enable signals, being able to manipulate a single byte in a data word. But if byte enables are not available or data groups smaller than 8 bits need to be modified, there are sequences of logic operations that can be used to access the data. Using a single data word to store multiple flag bits can save a significant amount of memory. The access, however is more complicated. In order to change a single bit in a vector, the following techniques are used.

$$\begin{aligned}
A \ll i = A \cdot 2^i &\quad \text{Left Shift by } i \\
A \gg_a i = \lfloor A/2^i \rfloor &\quad \text{Arithmetic Right Shift by } i \qquad (8.2) \\
A \gg_l i &\quad \text{Logical Right Shift by } i
\end{aligned}$$

Isolating a single bit at position i in arbitrary vector F containing multiple flags in the form of Y is a simple task, as shown in Eq. 8.3. Logical right shifting (see 8.2) moves the bit to the LSB, a logical AND with the literal 1 turns all other bits to zero.

$$Y = (F \gg i) \wedge 1 \quad \rightarrow Y_0 = F_i, Y_{n-1} \ldots Y_1 = 0 \qquad \text{Get bit } i \tag{8.3}$$

$$F = F \vee (1 \ll i) \quad \rightarrow F_i' = 1 \qquad \text{Set bit } i \tag{8.4}$$

$$F = F \wedge \overline{(1 \ll i)} \quad \rightarrow F_i' = 0 \qquad \text{Clear bit } i \tag{8.5}$$

$$F = F \oplus (1 \ll i) \quad \rightarrow F_i' = \overline{F_i} \qquad \text{Toggle bit } i \tag{8.6}$$

Efficient Mask Creation Conditionally changing content usually requires masking, that is, optionally clear a value by bitwise *AND* of the mask and content. Masking makes use of the 2nd Boolean law; a value's logical *AND* conjunction with 0 is 0. To make it depend on the flag bit, all bits of the mask show the same value as the flag bit. The most efficient way to achieve this is to exploit the binary representation of negative numbers, in this case the Two's Complement [86, pp 246].

$$\begin{aligned} -Y &= 2^n - Y \\ &= \overline{Y} + 1 \end{aligned} \qquad \text{Two's complement} \tag{8.7}$$

$$-1 = 0b\ldots1111$$

Negative numbers are represented by subtracting their absolute value from 2^n These are in the Two's Complement, which is obtained by inverting all bits in a vector and adding 1. Let the flag Y and the mask M both be bit vectors n bits wide. Y's least significant bit is the flag bit, Y_0. All other bits, $Y_{n-1} \ldots Y_1$ are zero. Using the Two's Complement representation from 8.7, lemma 8.8 is valid; each bit of M takes the value of Y_0.

$$\forall Y \in \{0, +1\} \ : \ M = -Y \ \Rightarrow \ M_i = Y_0 \tag{8.8}$$

Branch Avoidance I: Conditional Assignment If equal execution times are required regardless whether a branch is taken or not, a condition branch can also be expressed in pointer arithmetic followed by a jump. For this case, when a flag is true, an alternative target address is assigned to the pointer. To achieve the same execution length whether a branch is taken or not, the pointer must always be updated. Only the written content depends on the flag bit.

As shown in Eq. 8.9, this is can be done by using the mask and the negated mask in the same assignment. It is already a minimal function, which can easily be proven by applying a logic minimisation algorithm like Karnaugh Veitch or Quine-McCluskey. The function is a trivial case, the claim that it assigns either A or B easy to follow through. The inner AND functions mask A and B, using the mask and the inverted, one of the terms in the OR function must be 0. By applying 1st Boolean law, a value's logical OR conjunction with 0 is the value itself, the OR term resolves to either the value of A or the value of B. In conclusion, A is either assigned the value of A and therefore unchanged or assigned the value of B.

$$f(A) = A\overline{M} \vee BM \tag{8.9}$$

Conditional assignments are a very common and important operation, and it therefore makes sense to optimise this step. The trivial case requires 4 operations (NOT, AND, OR, AND). Jeffery claims [119] that the assignment from eq. 8.9 can also be expressed as eq. 8.10

$$f(A) = A \oplus ((A \oplus B) \wedge M) \tag{8.10}$$

Depending on the processor architecture, this reduces the operation cost from 4 (NOT, AND, OR, AND) to 3 (XOR, XOR, AND). To verify Jeffery's suggestion, applying the Boolean laws and theorems [86, p. 51] delivers the proof by algebraic transformation that both equations are indeed equivalent:

Proof: $A\overline{M} \vee BM = A \oplus ((A \oplus B) \wedge M)$

$$
\begin{aligned}
& A \oplus ((A \oplus B) \wedge M) && \text{Subst. 8.1} \\
&= A \wedge \overline{((A\overline{B} \vee \overline{A}B) \wedge M)} \vee \overline{A} \wedge ((A\overline{B} \vee \overline{A}B) \wedge M) \\
&= A \wedge (\overline{A\overline{B}M} \wedge \overline{\overline{A}BM}) \vee \overline{A}BM && \text{De Morgan} \\
&= A \wedge ((\overline{A} \vee B \vee \overline{M}) \wedge (A \vee \overline{B} \vee \overline{M})) \vee \overline{A}BM \\
&= A \wedge (\overline{A}A \vee \overline{AB} \vee \overline{AM} \vee BA \vee B\overline{B} \\
&\qquad \vee B\overline{M} \vee \overline{M}A \vee \overline{MB} \vee \overline{M}M) \vee \overline{A}BM && \text{Complement} \\
&= A \wedge (\overline{AB} \vee AB \vee \overline{M}) \vee \overline{A}BM \\
&= A\overline{AB} \vee A\overline{M} \vee AAB \vee \overline{A}BM \\
&= AB \vee A\overline{M} \vee \overline{A}BM && \text{Simplification} \\
&= AB \vee A\overline{M} \vee BM && \text{Consensus}
\end{aligned}
$$

$$= A\overline{M} \vee BM$$

Both algorithms are equivalent. To show that here is in fact an improvement, the total execution time, which is platform dependent, has to be evaluated.

Instruction	Issue	Result
and	1	1
not	1	1
or	1	1
xor	1	1
add	1	1
branch	4	–
branch gr	1/4	–

TABLE 8.1: Execution Time per LM32-Instruction

An excerpt of the corresponding values from the processor documentation [91, pp. 60-96], is shown in table 8.1, which can be used to calculate execution times for all approaches. The operations will be pipelined, the delay between start of issue and end result in the present example is the time it takes to obtain the result plus the sum of all issue times $R_N + \sum T_i$.

Approach	Algorithm	Issue + Result	Cycles
Branching	IF M $X \rightarrow A$ ELSE $X \rightarrow B$	$(1/4 + 1)I + 1R$	2/6
Standard Arith.	$A\overline{M} \vee BM$	$(1 + 1 + 1 + 1)I + 1R$	5
Optimised Arith.	$A \oplus ((A \oplus B) \wedge M)$	$(1 + 1 + 1)I + 1R$	4

TABLE 8.2: Execution Time by Type of Conditional Assignment

Table 8.2 shows the resulting cost in CPU cycles for each approach. A slash signifies alternatives, not division. It clearly shows that the optimised arithmetic conditional assignment is at 4 cycles faster than the standard version with 5 cycles. It is also the mean of best and worst case of the branch instruction, which is a satisfying outcome.

Calculating Absolute Calculating the absolute of a number is another common case which is usually optimised by compilers. If the CPU does not have a native ABS instruction, the compiler does not use it or if there are no standard libraries, it can be sensible to write the operation explicitly. Eq. 8.11 shows a solution which operates in constant time. Since A is in two's complement, an arithmetic bit-shift by integer size n will give two possible results for mask M. If A is positive, $M = 0$, if it is negative, $M = -1$. It can then easily be seen in the following line that adding 0 and *XOR*ing with 0 does not change A. *XOR*ing with -1, however, logically inverts

A. In combination with adding -1, this is A's twos complement. Since it is only applied if A is negative, it follows $f(A) = |A|$.

$$\begin{aligned} M &= A \gg n \\ f(A) &= (A + M) \oplus M \end{aligned} \tag{8.11}$$

Calculating Minimum/Maximum Getting the minimum or maximum of a set of numbers is a common problem, even more common when the set consists of only of two numbers. Eq. 8.12 shows a variation of the conditional assignment from eq. 8.10 and 8.8, the mask is obtained by expanding the logical result of $X < Y$.

$$\begin{aligned} f_{min}(X, Y) &= Y \oplus ((X \oplus Y) \wedge -(X < Y)) \\ f_{max}(X, Y) &= X \oplus ((X \oplus Y) \wedge -(X < Y)) \end{aligned} \tag{8.12}$$

Branch Avoidance II: Conditional "unconditional" Maybe even more important than branchless conditional assignment variables (as described earlier in this subsection), is conditional branching with a fixed execution time. This can be achieved by the aforementioned assignment, using the register either as a direct target or an immediate offset for an unconditional jump. The execution time for assignment followed by an unconditional branch is high in comparison to a vanilla conditional branch instruction ($3 + 4$ vs $1/4$), but the purpose of the operation, deterministic branch behaviour, is achieved.

8.6.5 Summary

Taking all of the discussed approaches into account, as well as the scheduler design from section 8.2, it is possible to implement the DM's firmware (see figure 8.1) in a completely jitter free, deterministic fashion. This is a necessary prerequisite to design an accurate delay model of the DM, to be used in the offline scheduling analysis (see chapter 9).

Example The source code 8.1 in C99 shows an application of all the techniques presented in this chapter as an unrolled, branchless heap sort algorithm which is used in the DM scheduler

```c
#define HEAP_DEPTH   4 // Heap size and therefore depth is constant
#define NUM_ELEMENTS 9 // Example, value actually only needs to be constant over one execution

// heap element
struct he
{
  void* pThr;  // thread
  uint64_t dl; // deadline
} he;

//the heap
struct he heap[HEAP_SIZE]; // array of ptrs to threads and their corresponding deadlines
struct he mov;             // moving element

uint8_t src, dst; // Swap Operation Destination and Source, first destination is top of heap
uint8_t lLEr;     // left child is less or equal right child
uint8_t mGr, mGl; // moving element is greater right child / left child
uint8_t cl, cr;   // right child / left child was chosen for swap
uint8_t mask;     // mask to prevent further swapping if the moving element is sorted in correctly
uint8_t l, r;     // indices of child nodes
uint8_t lOK, rOK; // left child / right child exists. Only exists for better readability
uint8_t steps;    // step counter for swapping operations

dst   = 0;
src   = 0;
steps = 0;

// Branchless Heapsort Replace operation (constant execution time, worst case)
// replace single element, sort always takes log n
while (steps < HEAP_DEPTH)  {
  l      = (dst<<1)+1; // left child
  r      = l + 1;      // right child
  lOK    = (l  < NUM_ELEMENTS);
  rOK    = (r  < NUM_ELEMENTS);
  l     &= ~(lOK-1);  // mask to zero if out of bounds
  r     &= ~(rOK-1);
  lLEr   = (heap[l].dl <= heap[r].dl) | ~rOK; // (left child less or equal r c) or r c non existent
  mGr    = (mov.dl > heap[r].dl)      &  rOK; // mover greater right child and r c exists
  mGl    = (mov.dl > heap[l].dl)      &  lOK; // mover greater left child and l c exists
  cl     = ( mGl &  lLEr);                    // choose left child
  cr     = ( mGr & ~lLEr);                    // choose right child
  mask   = ~((cl | cr) -1);                   // all 0 when no chosen child, all 1 otherwise
  src    = dst + ((dst + 1) & mask) + cr;     // parent  = dst, left = dst+(dst+1), rightC  = dst+(dst+1)+1
  heap[dst] = heap[src];                      // swap
  dst       = src;                            // new destination is old source
  steps++;
}
heap[dst] = mov; // copy moving element into final position
```

LISTING 8.1: Heap Sort with constant execution time

Chapter 9

Theoretical Model

9.1 Overview

9.1.1 Motivation

This chapter investigates one of the most important aspects of this CS – the conditions under which it can be guaranteed to work deterministically. The DM is the head of an alarm based CS. As pointed out during the case study in chapter 4, being alarm based provides a certain degree of freedom when choosing a "safe" lead for transmission. There are limiting factors though, as discussed in chapter 7.3:

- Target control loop speed
- Available computing power
- Available network bandwidth

It is therefore necessary to create a model of the CS, which verifies if a chosen set of machine schedules can be scheduled within CPU utilisation $\leq 100\%$, does not generate traffic $> 1\,\text{Gbit/s}$ (including overhead) and does not exceed the target delay. This goal is made more difficult by the possible need to change machine schedules during runtime. This will happen whenever interlocks and beam requests from experiments need to be serviced.

The purpose of the model is to provide a guarantee that a given set of messages can be delivered on time.

9.1.2 Choice of Implementation

After a period of research in classical queuing theory [120], [121] the conclusion was reached that queuing theory is not well suited for the task at hand. Queuing theory is very generic, requiring a lot of the mathematical elements to model this specific problem to be developed first. Queuing theory is also more focused on throughput and probabilities, rather than determinism and providing delay bounds. Attempts in modelling the DM system using queuing theory turned out to lead into a lot of dead ends, the model never quite matched the prototype. Without an expert level knowledge in the field, there seemed to be little chance of accurately model a system as complex as the DM in queuing theory.

The author's specialisation in electrical engineering is communications, which is probably the main reason why *Network Calculus* (NC) had to be encountered at some point in the search for a suitable tool to solve this particular problem. Parts of the problem in modelling the DM were exhibiting a striking resemblance to problems quite common in system theory. Modelling machine schedules, in a way that would make their superposition and shifting in time manageable, showed a lot of the hallmarks of signal processing. Expressing the burstiness of a schedule over time as a function of frequency in a spectrogram seemed natural, just as the idea to smooth bursty flows by a filter element did.

After deeper investigations of the work of [122] (1991), and later [123] (2001), NC presented itself as an ideal tool for the problem at hand. NC is a mathematical framework to model concepts from system theory in a networking context, with focus on deterministic behaviour and bounds.

9.2 Introduction to Network Calculus

9.2.1 Overview

System Theory for Signal Processing Signal processing is a very important field in electrical engineering and computer science. In most electronic devices signals must be generated, shaped, filtered or distorted. Sheath current filters suppress low frequency humming in a sound system, a blur effect removes the high frequencies from an image, to just name a few examples.

Complex behaviour in system theory is modelled by the concatenation of basic elements. These are small two-port networks, for which the signal transfer functions can be calculated. System theory provides the necessary mathematical tools to put these elements in series or parallel and so enables the calculation of complex signal transformations.

Computer Networks Another application of system theory would be modelling the traffic in computer networks. Similar to signals, network traffic can be shaped, filtered, split or joined, but the application of classical system theory is rather tedious. Specialisation within queuing theory has evolved to deal with the flow in computer networks, focusing on optimising throughput.

Network calculus NC is an approach that applies system theory to deterministic queuing systems found in communications, such as computer networks. Contrary to traditional system theory used for electronic circuits, NC employs a different set of algebra, the Min-Plus Dioid (addition becomes computation of the minimum, multiplication becomes addition). The approach is aimed at understanding and modelling fundamental properties of networks, such as delay or buffer requirements, scheduling or window flow control. The focus of NC lies on worst case analysis in order to provide guarantees for a communication system.

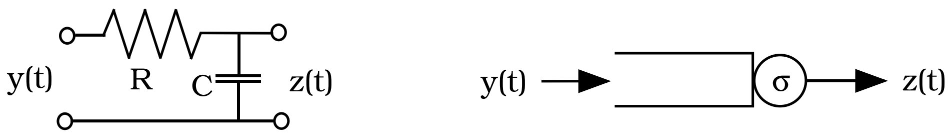


FIGURE 9.1: Equivalency: System Theory Low-Pass and NC Shaper [123]

As an example, the similarity between an RC low pass filter in system theory and a rate-limiting shaper node (server) in NC is shown in figure 9.1. The two-port filter network on the left transforms an incoming analogue signal by applying the convolution with the circuit's impulse response. In electrical engineering, a two-port network has a transfer function, which defines the output voltage in relation to the input voltage. In NC, input- and output-"signals" are cumulative flows. This means the cumulative sum of units of data (bits, words, packets, etc.) over time.

The similarity between a signal filter and a shaping node becomes more apparent when considering a constant rate of arriving packets is equivalent to a *frequency* of packet arrivals. It therefore follows, that a shaper imposing a maximum rate (frequency) is low pass filtering the packet flow (signal).

Example NC focuses on guarantees, it shows the bounds for maximum delay and backlog that a flow can experience. A convenient property of NC is the minimal effort necessary to obtain backlog and delay values. As figure 9.2 shows, any input flow $y(t)$ passing a node produces an output flow $z(t)$ for which $z(t) \leq y(t)$ holds true. At any point in time, the current backlog can be determined by the vertical deviation of the flows. The current delay can be determined by calculating the horizontal deviation. Finding the respective maximum values by applying the supremum is then trivial.

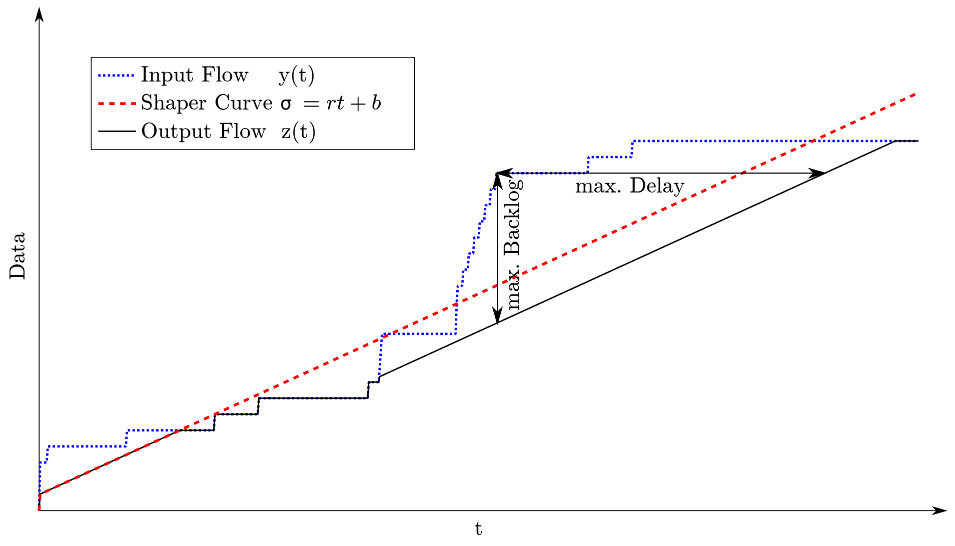


FIGURE 9.2: Flow passing through a Shaper

9.2.2 Network Calculus Core Concepts

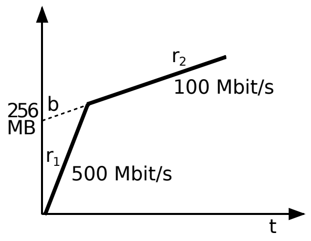
While flows are defined as the cumulative sum of data over time, NC defines systems in terms of arrival curves and service curves. Figure 9.3 shows two related examples.

Arrival Curves Describe sets of constraints that govern the input flow's behaviour over time. These curves are usually piece-wise linear, concave functions. Their slope describes the maximum allowed rates, their vertical offset the burst tolerance.

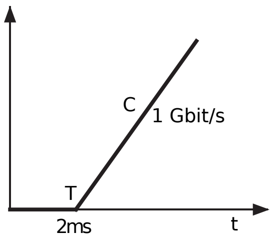
An arrival curve could state that an incoming flow is allowed a peak rate of no more than $500\,\text{Mbit/s}$ up to an input buffer size of $256\,\text{MB}$, afterwards it must fall back to a sustainable rate of $100\,\text{Mbit/s}$ until the buffer's fill level is lower (figure 9.3a).

Service Curves These are the counterpart to arrival curves, they describe the service a system offers to an input flow. Their horizontal offset is describing the amount of time lag packets experience, their slope describes the minimum rates. To again provide an example, a server's minimum service curve could state that it will delay packets for at least $2\,\text{ms}$ and can handle traffic up to a rate of $1\,\text{Gbit/s}$. These curves are usually piece-wise linear, convex functions (figure 9.3b).

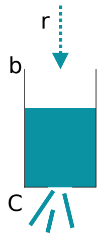
One of the core constructs of NC is a node behaviour called the "leaky bucket controller". The analogy is simple: Consider a water bucket, able to hold an amount of water b, leaking with a constant rate of C. It can handle one or more of gushes of water of arbitrary volume, up to the capacity of the bucket. Once the bucket is full, it is easy to see that there is a maximum rate at which one can add more water without overflowing, which is $r \leq C$ – a system with buffer size b, input data rate r and output data rate C. This is equivalent to featuring an arrival curve $\alpha = \upsilon_{b,r} = rt + b$ and a minimum/maximum service curve $\beta = \lambda_C = Ct$ (figure 9.3c).



(A) Arrival Curve



(B) Service Curve



(C) "Leaky Bucket"

FIGURE 9.3: NC Examples

Output Arrival Curves Arrival and service curves put constrains on input and system behaviour, but to give a guaranteed flow behaviour, the output of a system also needs to be constrained. The matching instrument is called an output arrival curve. It is similar to an arrival curve, but describes the flow *after* it has traversed a node, having experiencing its service. It is used to define the input constraints for the next downstream node. As an example, consider a periodic input flow, sending with a rate of $1\,\text{Gbit/s}$ for 20% of the time and idle otherwise. If the node provides a service of 250 Mbit/s, the output arrival curve would also be a periodic flow, sending at 250 Mbit/s for 80% of the time.

Shaping Curves are the application of service curves to form a flow. If a node forces a flow to conform to a specific target arrival curve, it is called a shaper. More details on shapers can be found in subsection 9.2.4.

Bounds The main goal of NC is to provide bounds on delay, backlog and output, in order to give guarantees. Bounding the end-to-end delay a flow experience is for example necessary when calculating if the lag for an individual VoIP connection will always stay low enough not to impair a conversation. Bounding backlog, for example, is important for calculating the size of memory a router will need. And lastly, the necessity for bounding an output flow, which is explained under "output arrival curves".

9.2.3 Mathematical Background

Most of the theory on Network Calculus in this chapter stems from Thiran and Le Boudec's text book on the subject [123]. This introduction will focus on the application of the presented theorems and limit the formal proof and mathematical background to the necessary minimum. After the work of Thiran and Le Boudec, NC has forked into the direction of stochastic applications [124]. This is of no further interest to this case study, as the CS must be deterministic. Some of the most interesting advance in the field of deterministic NC, as well intriguing problems and tricks of the trade, were gathered from the post-2006 publications of Schmitt, Zdarsky, and Thiele, as well as Fidler and most recently, Bondorf [125]–[129].

Min-Plus Algebra NC uses a different algebraic Dioid (similar to e.g. Boolean algebra, which replaces arithmetic operations by logic), which replaces addition with computation of the minimum and multiplication with addition. The two most important equations describe the convolution operations, similar to standard system theory. They are defined as

$$\begin{aligned}
&\text{Min-Plus Convolution} && f(t) \otimes g(t) = \inf_{0 \le s \le t} \{f(t-s) + g(s)\} \\
&\text{Min-Plus Deconvolution} && f(t) \oslash g(t) = \sup_{s \ge 0} \{f(t+s) - g(s)\}
\end{aligned} \tag{9.1}$$

Max-Plus Algebra The corresponding max-plus operations are also listed, not only for the sake of completeness, but because max-plus deconvolution will become useful when finding a curve providing a lower bound to a function. The concrete application will be shown in conjunction with scaling operators in section 9.2.4.

$$\begin{aligned} &\text{Max-Plus Convolution} && f(t)\,\overline{\otimes}\,g(t) = \sup_{0\le s\le t}\{f(t-s)+g(s)\} \\ &\text{Max-Plus Deconvolution} && f(t)\,\overline{\oslash}\,g(t) = \inf_{s\ge 0}\{f(t+s)-g(s)\} \end{aligned} \tag{9.2}$$

Curves We will follow the convention of marking output related curves and flows by appending an asterisk. Arrival curves (if not otherwise stated, an upper bound) are denoted by the letter α (and therefore, α^* denotes an output arrival curve). For service, there exists a maximum service denoted as γ, which is useful to calculate buffer sizes, and minimum service denoted β, which is used to calculate delay. Shapers are always denoted as σ. The definitions of all introduced curve types are as follows:

$$\begin{aligned} &\text{Input-Output Relation} && R(t) \ge R^*(t) \\ &\text{Max. Arrival Curve} && R(t)-R(s) \le \alpha(t-s) \\ &\text{Max. Service Curve} && R^* \le R\otimes\gamma \\ &\text{Min. Service Curve} && R^* \ge R\otimes\beta \\ &\text{Shaping Curve} && R^* = R\otimes\sigma \le (R\otimes\alpha)\otimes\sigma \end{aligned} \tag{9.3}$$

Bounds The following equations concern the “three bounds”, as Thiran and Le Boudec called them: backlog, delay and output flow. Backlog and delay can be directly calculated from the difference between input and output flows (see figure 9.2), as well as from arrival and service curves. It is easy to see here, that a node with an arrival curve, of a higher continuous rate than the service curve, cannot have bounds.

$$\begin{aligned} &\text{Flow Backlog} && b(t) = R(t) - R^*(t) \\ &\text{Flow Delay} && d(t) = \inf\{\tau \ge 0 : R(t) \le R^*(t+\tau)\} \\ &\text{Curve Backlog} && b(t) = \alpha(t) - \beta(t) \\ &\text{Curve Delay} && d(t) = \inf\{\tau \ge 0 : \alpha(t) \le \beta(t+\tau)\} \end{aligned} \tag{9.4}$$

For all systems, an output arrival curve (that is, the arrival curve for the following node) can be calculated by deconvolution of the input arrival curve with the node's service curve. If no arrival curve is known for a node, a minimal arrival curve can always be calculated by deconvolution of the input flow with itself. This leads to the following expressions:

$$\begin{aligned} &\text{Output Arrival Curve} && \alpha^* = \alpha\oslash\beta \\ &\text{Minimal Arrival Curve} && \alpha_{min} = R\oslash R \end{aligned} \tag{9.5}$$

Concatenation Whenever a flow passes through multiple nodes in sequence, it is possible to concatenate service curves into a single equivalent node. This is similar to concatenation of transfer functions in system theory. The service curve of the equivalent node is the convolution of passed service curves:

$$\text{Service Concatenation} \quad R^* \geq r_1 \otimes \beta_2 \geq (r_1 \otimes \beta_1) \otimes \beta_2 = R \otimes (\beta_1 \otimes \beta_2) \tag{9.6}$$

In NC, there is selection of basic curve functions that are frequently encountered when modelling networks. More complex curves can be constructed from basic functions by adopting a piecewise-linear approach (figure 9.4). A compilation of common basic functions in the context of NC is found in figure 9.5.

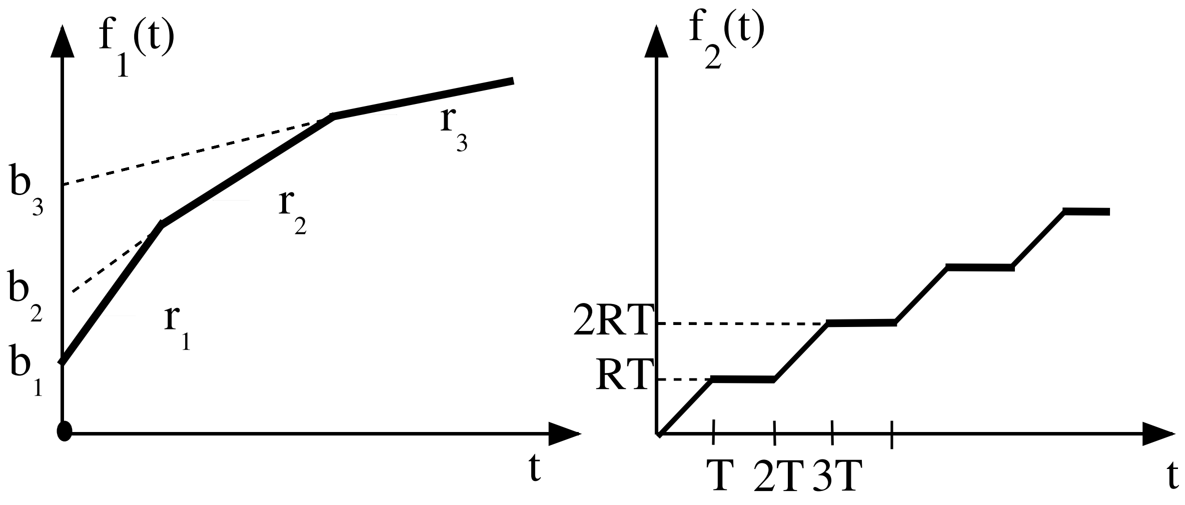


FIGURE 9.4: Examples of piecewise-linear Functions [123]

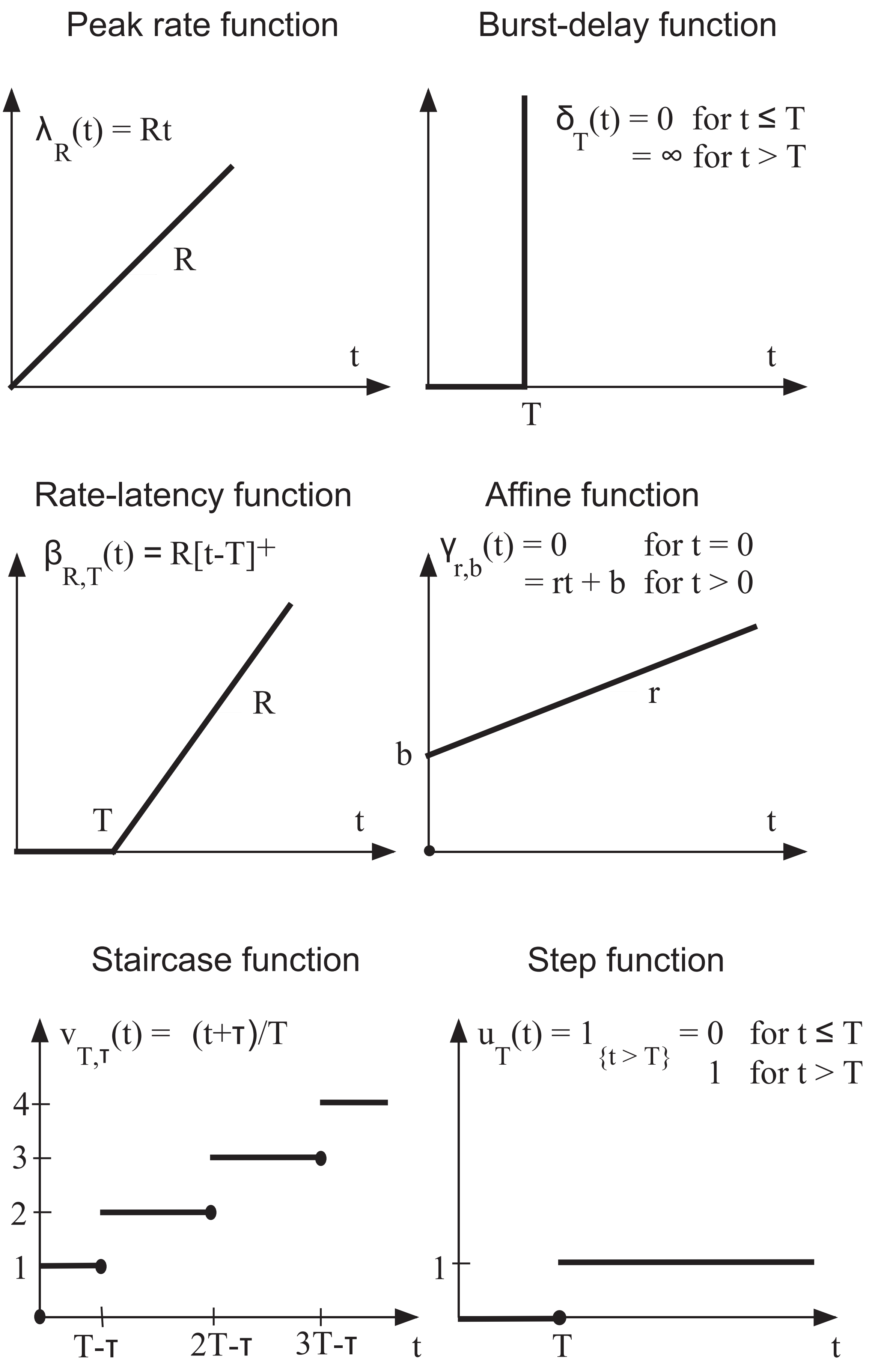


FIGURE 9.5: Catalogue of commonly used Curve Functions in NC [123]

9.2.4 Elementary Building Blocks

NC uses a number of basic elements to construct network models, similar to system theory having different types of filter, mixer and splitter nodes as basic elements. The behaviour of a complex, composite system is then derived from the behaviour of these basic elements.

Shaper A shaper is an element offering its shaping curve as both a minimum and maximum service. As figure 9.6 shows, the convolution applies the shaping curve simultaneously at *every* point of the input flow, thus forcing the flow to have σ as its arrival curve. Whenever it would exceed the curve, data is delayed in time (moved to the right). There are two types of shapers defined in [123]. The first complies to the above definition, although it is left unclear how the process is actually implemented. Focusing on the second definition, elements called *greedy shapers*. A greedy shaper "*delays the input bits in a buffer, whenever sending a bit would violate the constraint σ, but outputs them as soon as possible*" [123, p. 42].

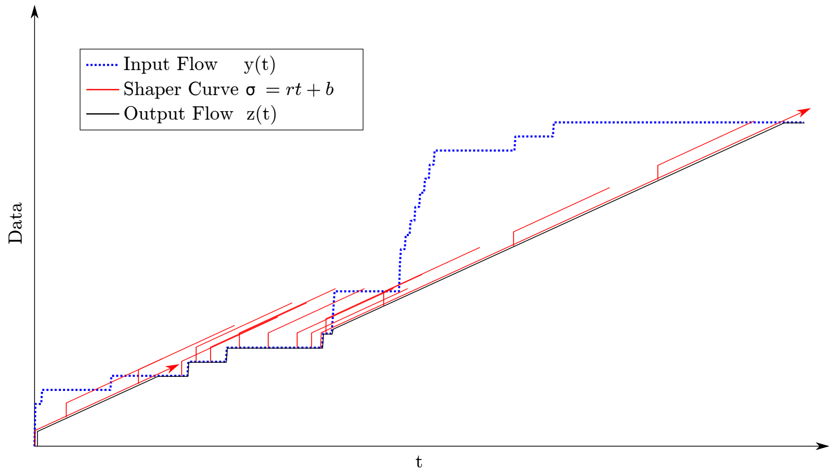


FIGURE 9.6: Visualisation of the Effect of min-plus Convolution: Shaping curve σ is enforced at every point of input flow $y(t)$

Packetiser A packetiser is a variable delay element with a service curve roughly resembling a staircase (figure 9.7). It delays the output so it stays at step level $L(n-1)$ until the input flow has reached $L(n)$. Packet data is thus only forwarded once the whole packet was received [123, pp. 218]. An element behaving in the way described is the so called "L-Packetiser". These are theoretical constructs, assuming instantaneous packet arrival and departure. It employs the indicator function $1_{\{<expr.>\}}$, which is defined as

$$1_{\{<\text{expr.}>\}} = \begin{cases} 1 & \text{when } \textit{expression} \text{ is true} \\ 0 & \text{when } \textit{expression} \text{ is false} \end{cases} \tag{9.7}$$

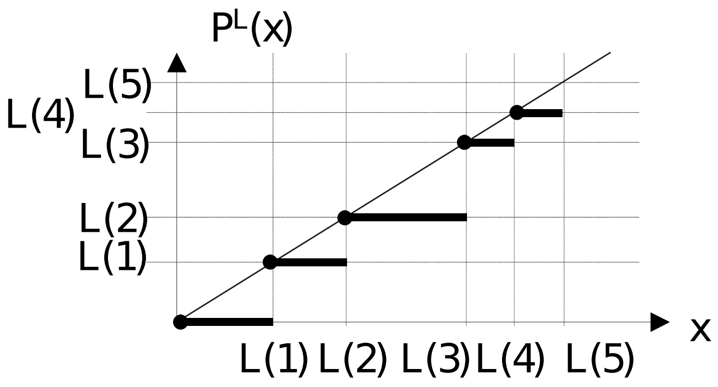


FIGURE 9.7: Definition of Function P^L [123]

The function for an L-Packetiser can be then written as eq. 9.8, x being the current flow level ($R(t)$).

$$\mathrm{P^L(x)} = \sup_{n \in \mathbb{N}} \left\{ L(n) 1_{\{L(n) \leq x\}} \right\} \tag{9.8}$$

We will now take a closer look at the step function $L(n)$ which the packetiser employs to delay data until a packet is complete. $L(n)$ is the cumulative function of packet lengths, $l(n)$ being the length of the n-th packet. $L(n)$ is defined as

$$\begin{aligned} \mathrm{L(0)} &= 0 \\ \mathrm{l(n)} &= L(n) - L(n-1) \\ \mathrm{l_{max}} &= \sup\{l(n)\} \end{aligned} \tag{9.9}$$

For variable length packets, step values for $L(n)$ are either delivered a-priori or calculated iteratively from $L(n-1)$. If packet length is constant, cumulative packet length can simply be written as

$$\mathrm{L(n)} = n \cdot l \tag{9.10}$$

L-packetisers are a virtual construct, because no components can provide instantaneous arrival *and* departure (except wires if relative times are considered). A more realistic application of the theory are Packetised Greedy Shapers (PGS), which, as the name suggests, model buffer delay by prefixing the L-packetiser with a greedy shaper:

$$\mathrm{R}^* = P^L(R \otimes \sigma) \tag{9.11}$$

An L-Packetiser buffers a packet, the first bit of any incoming packet is delayed until the arrival of its last bit. Assuming a constant rate shaper λ_C as the bit-by-bit system, the maximum delay experienced by a packet is the time it takes the maximum size packet, $\frac{l_{max}}{C}$. The equivalent minimum service of a packetiser can therefore be written as the concatenation of a constant rate node and the buffering delay, a rate-latency service of the form $\beta_{C,\frac{l_{max}}{C}}$.

Multiplexer Multiplexers are the most complex building elements in NC, because their impact can vary widely depending on their inputs and policy. Aggregation of flows is a common scenario for routers, switches or even endpoints, if they run multiple services in parallel. The multiplexer is a node offering a total service β, which is usually a constant rate of λ_C, to all incoming flows $\sum R_i(t)$. Service allocation to the individual flows is defined by a scheduling policy. The main distinction is made between arbitrary multiplexing, which assumes no knowledge about policy, and several other cases. The assumption of an arbitrary policy always provides correct, but usually most pessimistic bounds. The most important other case is FIFO multiplexing, which assumes messages are served in order of their arrival. FIFO and fixed priority policies (one input flow always preferred over another) are also handling well in NC.

We shall give an example of a simple fixed priority setup with two flows. The service available to the low priority (LP) flow is the residual service, after the preferred flow has been served. The high priority (HP) flow's arrival curve is subtracted from the available service, which amounts to the residual service for the LP flow. Because service curves are always wide-sense increasing [123, p. 19], so the supremum of the difference must be used [125].

Whenever the slope of the arrival curve is greater than the slope of the service curve, the difference would be a decreasing function. This is in effect service backlog, but service curves cannot represent this directly. The delay this backlog causes is added instead. This is achieved through the supremum, as it keeps the residual service at constant level when it would be decreasing. Since no arrivals are serviced during that period, it introduces an equivalent delay. The shorthand notation for this residual service is defined as

$$\beta_2^{l.o.} = \sup_{0 \le s \le t} \{\beta(s) - \alpha_1(s)\} = \beta \ominus \alpha_1 \tag{9.12}$$

More explicitly, consider a multiplexer offering a constant service rate C. The high priority flow is constrained by an affine function of maximum rate r and a buffer size b, with $r < C$. It is easy to see that the rate experienced by the LP flow will be the difference between the HP rate and the available rate in the system. Furthermore, if the HP flow has backlogged traffic, it can completely saturate the system. This makes the LP flow wait until all HP backlog is serviced. Thus the residual service is a rate latency curve

$$\beta^{l.o.} = \beta_C \ominus \alpha_{r,b} = \beta_{C-r, \frac{b}{C-r}} \tag{9.13}$$

If there is more than one flow to be multiplexed, the residual service experienced by the flow of interest is calculated by summing up the arrival curves of all interfering flows, such that

$$\beta_{foi}^{l.o.} = \beta \ominus \sum_i \alpha_i \tag{9.14}$$

Depending on the system's arbitration policy, a peculiarity can occur for HP service. If the system operates under a FIFO policy, HP itself has a waiting condition, due to an ongoing LP transmission (because it cannot pre-empt the LP flow). In line with FIFO, there must exist an upper bound for the size of such units of data, l_{max}. Because the length is arbitrary, this holds for bits and whole packets alike. HP is waiting for any ongoing LP transmissions to complete, so its minimum service is defined as

$$\beta_1^{hi} = \beta \ominus l_{max} \tag{9.15}$$

Scaler In many networks, there are nodes that compress or decompress data (video-encoder, etc.). This is a problem in NC, because the fundamental criterion, that $R(t) \leq R^*(t)$, is violated in the case of compression. While it is kept in the case of decompression, the relation between input and output flow is still distorted. In both cases, horizontal and vertical deviation no longer correspond to delay and backlog a flow is experiencing.

Data Scaling, first introduced by Fidler and Schmitt, is a concept for NC which handles this problem by application of scaling curves [126]. The Scaler will assign each bit of data $a = R(t)$ a scaled image of $S(a)$. S is a wide-sense increasing, bijective function curve, thus an inverse function S^{-1} exists. From the perspective of the system's ingress, delay and backlog can then be calculated since perspective is important because R_S is scaled in relation to the ingress, but not in relation to a downstream node.

$$\begin{aligned} \mathrm{R}_\mathrm{S}^*(\mathrm{t}) &= S(R(t) \otimes \beta(t)) \\ \\ \mathrm{b}(\mathrm{t}) &= R(t) - S^{-1}(R_S^*(t)) \\ \mathrm{d}(\mathrm{t}) &= \inf\left\{\tau \geq 0 : R(t) \leq S^{-1}(R_S^*(t+\tau))\right\} \end{aligned} \tag{9.16}$$

Scaling Functions and Curves Scaling functions can be directly applied to flows. However, application to arrival and service curves is not directly possible. Scaling curves must be derived from the function first. $\underline{S}$ is a minimum scaling curve of S if it is less or equal to its max-plus deconvolution, likewise, $\overline{S}$ is a maximum scaling curve of S if it is less or equal to its min-plus deconvolution:

$$\begin{aligned} \underline{\mathrm{S}}(\mathrm{b}) \leq \inf_{a\in[0,\infty)} \{S(b+a) - S(a)\} \quad &= S(b)\, \overline{\oslash}\, S(b) \\ \overline{\mathrm{S}}(\mathrm{b}) \leq \sup_{a\in[0,\infty)} \{S(b+a) - S(a)\} \quad &= S(b) \oslash S(b) \end{aligned} \tag{9.17}$$

Inverse Scaling Curves Obtaining inverse scaling curves follows directly from eq. 9.17 by applying the process to the inverse scaling function S^{-1}. It further holds that the maximum scaling curve of the inverse scaling function, $\overline{S^{-1}}$, is equal to the inverse of the minimum scaling curve of the scaling function, $\underline{S}^{-1}$. and vice versa [126, p. 290 (4)].

Scaled Servers To apply the above statements to finding an end-to-end delay bound, it is necessary to scale servers. This means applying scaling curves to service curves to obtain a scaled service. This may seem trivial, but is important as a way to allow concatenation of systems in the presence of scalers.

The core concept is the equivalency of the following systems: The minimum and maximum service, β and γ, of a server with scaled output (a) and a server with scaled input (b) lead to equivalent bounds, if S is bijective and:

$$\begin{aligned}\beta^{\mathrm{S}}(\mathrm{t}) &= \underline{S}(\beta(t))\\ \gamma^{\mathrm{S}}(\mathrm{t}) &= \overline{S}(\gamma(t))\\ \\ \beta(\mathrm{t}) &= \underline{S^{-1}}(\beta^{S}(t))\\ \gamma(\mathrm{t}) &= \overline{S^{-1}}(\gamma^{S}(t))\end{aligned} \tag{9.18}$$

9.2.5 Delay Analysis Methodology

There are many different approaches to obtain end-to-end delay bounds for a system. The first three, *Total Flow Analysis (TFA)*, *Separate Flow Analysis (SFA)* and *Pay Multiplexing Only Once Analysis (PMOO)*, are the "classical" approaches, solely relying on network calculus. The different methods are demonstrated on a minimal example of two nodes in tandem [125], through which two flows are multiplexed as a FIFO aggregate.

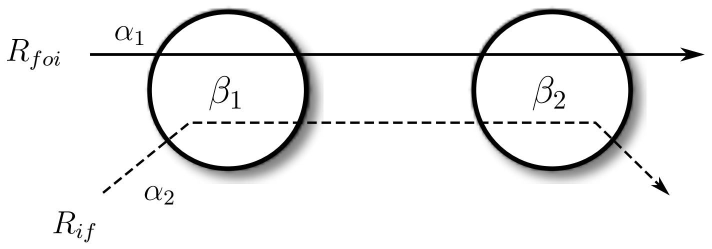


FIGURE 9.8: Minimal Network Example: 2 Nodes, 2 Flows

Total Flow Analysis This form of an end-to-end analysis adds the delays encountered by the total flow, that is, the *sum* of all flows, along the path. This is calculated per node, using the arrival curves transformed by the node. This means using the original arrival curves α_1 and α_2 at the first node, their output arrival curves α_1' and α_2' (see eq. 9.5, p. 142) at the second node and so on. Delay is defined as horizontal deviation between input and output flow, so the delay for TFA is calculated as

$$d_{TFA} = h(\alpha_1 + \alpha_2, \beta_1) + h\big((\alpha_1 + \alpha_2) \oslash \beta_1, \beta_2\big) \tag{9.19}$$

The obtained bound is valid for both the Flow Of Interest (FOI) and the interfering flow(s), but does not provide information about which flow it belongs to. It is thus overly pessimistic for all but one flow. TFA tends to produce the least tight bounds.

Separate Flow Analysis SFA aims to obtain a tight bound for the FOI by removing it from the system and inspecting the residual service available to it after servicing the interfering flow(s). This is achieved by summing up all flows except the FOI and subtracting the aggregate flow from the total service of the system (see eq. 9.12, p. 147). The residual service is then computed over all nodes by convolution, which equals the end-to-end service encountered by the FOI and thus provides its delay. This can be written as

$$d_{SFA} = h\Big(\alpha_1, \big(\beta_1 \ominus \alpha_2\big) \otimes \big(\beta_2 \ominus (\alpha_2 \oslash \beta_1)\big)\Big) \tag{9.20}$$

Because SFA includes topology information when calculating residual service, it is proven to deliver tight bounds for all multiplexing policies. Contrary to TFA, SFA pays bursts only once (PBOO criterion), because residual services are concatenated before calculating end-to-end delay.

Pay Multiplexing Only Once Analysis The downside of SFA is an overly pessimistic accumulation of multiplexing delay at every node, even if it is not occurring there. Consider two FIFO multiplexed flows, which can both send at the same rate, passing through several nodes all capable of this rate. While it is true that only one flow can send at the same time, this only determines delay at the first node. Once the order of sending is determined, additional nodes should not introduce more multiplexing delay.

PMOO is a special case of SFA which tries to, as the name suggests, avoid paying multiplexing multiple times by concatenation all encountered nodes into one equivalent node before the residual service calculation. In the example from figure 9.8, this means the convolution of β_1 and β_2 before subtracting α_2:

$$d_{PMOO} = h\Big(\alpha_1, \big((\beta_1 \otimes \beta_2) \ominus \alpha_2\big)\Big) \tag{9.21}$$

In most cases, PMOO delivers the tightest bounds. There are some special cases when SFA can perform better than PMOO, when the service rates is higher at downstream nodes than at the ingress [127]. Contrary to SFA, PMOO analysis has only been proven for arbitrary multiplexing policy.

Building on NC There are known problems with the described analysis methods when treating aggregated flows, because it can be proven [127] that even with PMOO analysis, multiplexing over multiple hops does not always produce the tight bounds. There were more recent advance in providing tight end-to-end delay bounds from Lencini et al. [130] and Schmitt et al. [127](2008), employing optimisation algorithms on top of network calculus. In order to obtain the minimum delay bound, these approaches define the service curve of all traversed nodes and then solve a linear optimisation problem for the distribution of backlog between these.

The most recent research by Bondorf, Nikolaus, and Schmitt from 2016 shows a very interesting development back to pure algebraic solutions. In their publication, they prove the existence of a completely algebraic technique, which requires much less computation than linear optimisation, but still closely matches experimental results to within 1.16% [131].

9.3 Approach for modelling the Data Master

9.3.1 Overview

The following is the overview of the intended NC model to be used for an end-to-end delay analysis of the DM and its environment. The first part of the model will be the DM itself and its sub-components. The second part will be a black box model of the WR switches, the third a model of timing receiver. While it would be feasible to model the switches more accurately, a white box approach is outside the scope of this work.

The purpose of this model (and indeed of NC as a whole) is to provide worst case bounds on delay, backlog and output flows, it does not provide exact input-output transformations. While it is technically possible to create accurate transfer functions, the benefit of using "low-level" models would be limited to verifying the formally proven abstract modelling techniques provided by Thiran and Le Boudec and Schmitt.

Naming Conventions Within the timing system, server nodes are processing data at four different (three distinct) rates r_x. These are, in descending order:

$$\begin{aligned}
r_3 &= 4\,\mathrm{B} \cdot 8/\mathrm{ns} &&= 4\,\mathrm{Gbit/s} \\
r_{2a} &= 4\,\mathrm{B} \cdot 16/\mathrm{ns} &&= 2\,\mathrm{Gbit/s} \\
r_{2b} &= 2\,\mathrm{B} \cdot 8/\mathrm{ns} &&= 2\,\mathrm{Gbit/s} \\
r_1 &= 1\,\mathrm{B} \cdot 8/\mathrm{ns} &&= 1\,\mathrm{Gbit/s}
\end{aligned}$$

9.3.2 Machine Schedules as Flows

Before constructing a detailed service model, we shall have a closer look at the input flows, i.e. how they originate from a collection of machine schedules. All flows entering from the CPU side are of potential interest for analysis, while the injected headers from the EBM and WR traffic will always be treated as interfering flows. Machine Schedules provide content for timing messages as well as information about points of decision, i.e., which schedules can be played next and which is the default selection. Each message within a schedule requires a dispatch time in relation to its time offset. With these offsets and their arrival time, an EDF scheduler can create the corresponding message flow, the cumulative sum of messages over time.

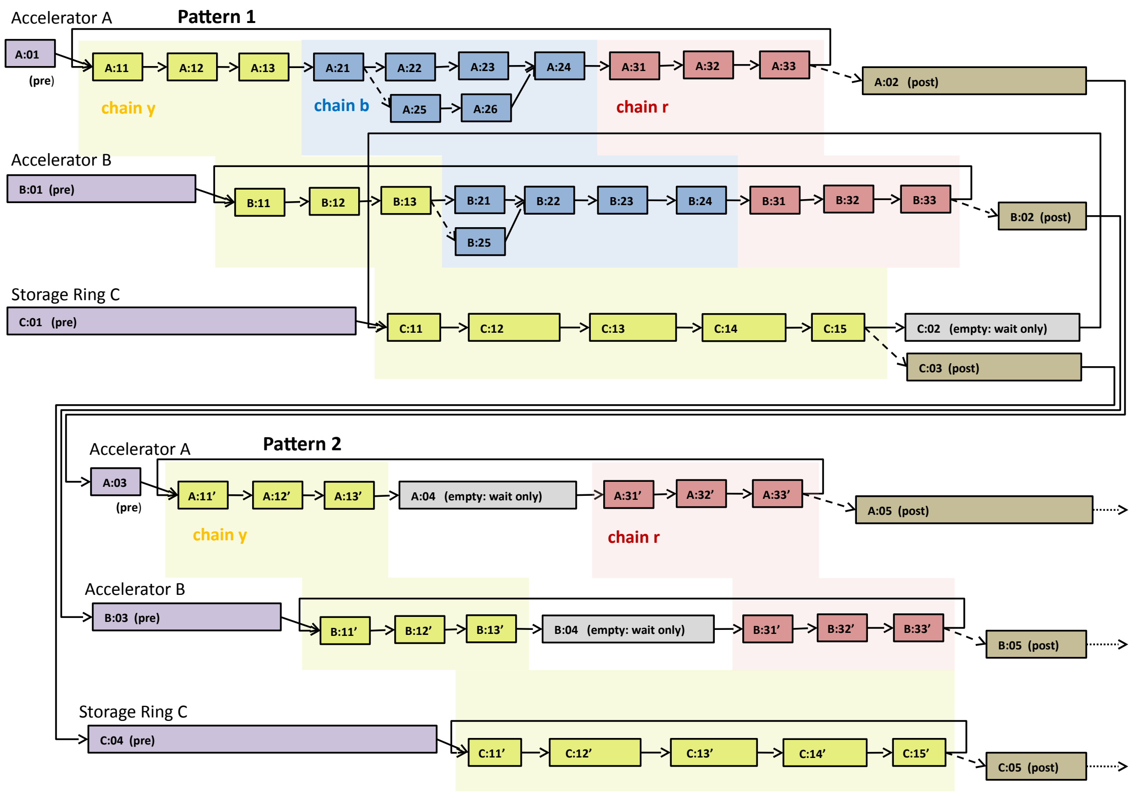


FIGURE 4.5: Machine Schedules for the Accelerator (p. 57)

Assignment of Arrival Curves It is always possible to find a minimal arrival curve for a flow. This is obtained by the min-plus deconvolution of the flow with itself, thus $\alpha = R \oslash R$. These arrival curves tend to be *very* form-fitting to the actual flow and are therefore tedious to describe formally. They are also often not concave, but star-shaped. For convenience (and especially for the following worst case of several schedules), it is advantages to find an approximated arrival curve from within a certain family of functions. A piece-wise affine function as shown in figure 9.4 tends to provide a good approximation [123, p.]. We will only briefly touch the subject of a suitable approach for approximation of arrival curves, as a full investigation is out of the scope of this work.

The solution to the problem is finding the minimax solution, that is, minimisation of the maximum error when choosing affine segments. Apart from least square approximation, which does not necessarily converge [132], there are splitting algorithms that search for segments within a certain error criterion [133] and also recursive approaches, which try to find the location of the tangent pivots directly [132]. While [132] will find an optimal solution, the result is only proven to be optimal for concave or convex functions. This is a problem for minimal arrival curves: While concave functions are always sub-additive, sub-additivity does not imply concavity. It would therefore be necessary to either construct the concave hull of the minimal arrival curve before applying [132], evaluate the quality of fits to sub-additive functions or choose for example the algorithm of Vandewalle [133], which is applicable to arbitrary functions.

It is important to note that assignment of arrival curves needs to consider the *whole* flow, i.e. the concatenation of machine schedules. All successions of machine schedules in the DM are either finite or periodic for the validity period of the analysis. Once a fitting arrival curve has been defined, it is possible to assign service to this flow at every node it passes through. This will allow to obtain an end-to-end delay bound for the corresponding flow.

As a proof of concept for this approach, an example for the generation of a piece-wise affine arrival curve describing a periodic message flow is given in the following paragraph.

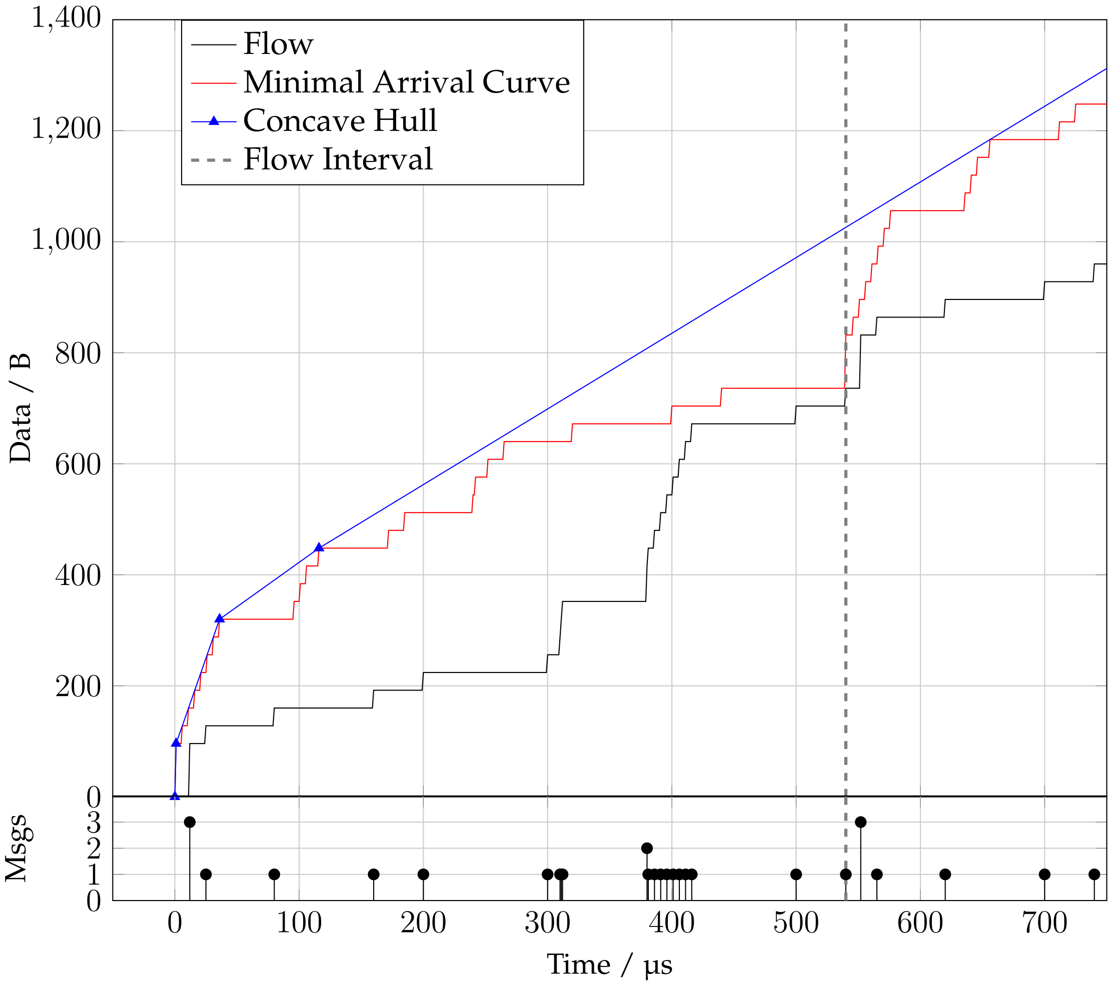


FIGURE 9.10: Generation of a piece-wise affine Arrival Curve from Flow.
The corresponding Messages are shown in the stem plot below.

The example given in figure 9.10 constructs the affine arrival curve for a periodic flow. The corresponding message flow is visualised as a stem plot at the bottom of the figure. Each circle signifies as a timing message of 32 B, stacked circles show concurrent execution times. As a first step, the flow's vector is cloned and concatenated to the original (black curve). Secondly, the minimal arrival curve is calculated by min-plus deconvolution of the flow with itself (red curve). Because the flow was cloned before, the minimal arrival curve (within the flow's interval) matches a periodic repetition. As the third step, the concave hull of the minimal arrival curve is constructed (blue dashed curve). All nodes not contributing to the outer shape are removed. As the fourth step, the affine function is reduced to the length of the flow's period, keeping the slope it had at the end of the interval. The resulting function is piece-wise described by affine functions of the form $m \cdot x + b$. It has a peak rate of ≈ 5 Mbit/s and a sustainable rate of ≈ 1 Mbit/s, with an initial burst of 96 B. Eq. 9.22 shows the description of this arrival curve (burst values in bytes, rates bytes per second, time in micro seconds):

$$\alpha_{r_i,b_i} = \left\{ \begin{array}{lrl} \gamma_{96,\ 6.5\cdot 10^5} & 0 & < x < \ 360 \\ \gamma_{320,1.28\cdot 10^5} & 36 & \leq x < 116 \\ \gamma_{448,1.25\cdot 10^5} & 116 & \leq x < \infty^+ \end{array} \right\} \tag{9.22}$$

9.3.3 Outside Interference

Worst Case for Online Flow Control Outside intervention through interlocks will change the path through the machine schedule graph (see figure 4.5) in realtime, yet the delay analysis will be done offline. The reason is that even if the analysis is carried out in realtime, detecting an imminent violation of the system's delay bound will not help containing the situation. Instead, the worst case combinatorial scenario of all possible flows at this point must be considered by taking the supremum of all minimal arrival curves. The supremum of sub-additive curves is also sub-additive, preserving their property.

- Arrival curves of all involved Schedules
- Time of Points of decision
- Sets of alternative arrival curves for each Point of Decision

If this information is available, it is possible to create compound worst case arrival curves from several individual ones by supremum of all alternatives. In the case of the DM, the combinatorial arrangement can only be conducted *after* the first EDF scheduler in the LM32's firmware. The reason is finding the worst case combination is only possibly with arrival curves of flows in which messages already occupy their scheduled release times.

Relaxed case for optional Online Flow Control All requests from experiments are not regarded as time-critical, they can thus be delayed without penalty. This creates a degree of freedom in online flow control, as each requested change to the schedule configuration, can instead be included by re-computation of the delay analysis. If the delay bound is violated, the change will not be executed and there are several possibilities to solve the problem. These range from telling the operator that this change is not allowed to automatically shifting the desired schedule change in time until the system can provide a suitable service. Arrival curves thus do not need to cover all possible combinations of requests from experiments, only the ones currently selected.

9.3.4 Recurring Analyses

At the time the very first analysis is undertaken, the model is representing a system at time $t = 0$ and thus without history. When the analysis is repeated during runtime on change of machine schedules, it is obvious that the system is *not* in this state. There are three possible approaches to treating case study, the first being trivial:

Clean Slate The first possibility is to halt the system completely. Because all messages were scheduled to spend a maximum time Δ_t in the system, ceasing the input flows and waiting for Δ_t will guarantee a system with empty buffers, hence the system is equivalent to the state at $t = 0$. This will initially be the preferred mode of operation for the case study.

Prepare for Everything The second possibility is accepting more loose arrival curves for the input flows and cover *all* possible combinations of machine schedules per input flow, thus never needing a second analysis. This would work for smaller sets of schedules and can be complemented by rare resets as described in the first approach.

Time Stop The third option involves halting time at the point of change, calculating backlog at every node, update input flows according to the requested changes, apply Thiran and Le Boudec's theorem [123, p. 225] for shapers with non-empty buffers and update service curves.

9.4 Scheduler Models

Type The DM requires two layers of schedulers to sort timing messages into chronological order by their deadlines. All schedulers are implemented as packetised earliest deadline first based on delay values.

Lower Layer The lower level scheduler is implemented in hardware and aggregates the flows from all instantiated processors. This module has been dubbed Hardware Priority Queue (PQ) and has been described in detail in chapter 7.4.

Upper Layer The upper level of schedulers is implemented in firmware inside the processors, as presented in chapter 8.2. The scheduler would not strictly be necessary at this point, but does allow a better utilisation of available processing time. Machine schedules must be distributed to individual CPUs depending on their current utilisation, as it must always be $\leq 100\%$. While it is possible to aggregate all these schedules into one big schedule per processor before running, this would be very inflexible. Every update would require a complete stop and exchange of the whole aggregate. Instead, software EDFs can easily aggregate individual machine schedules, each assigned to a worker thread, into a chronological flow. Such a scheduler is thus a prerequisite to enable online update of machine schedules and online flow control as described in 8.5 and 9.3.3.

9.4.1 Scheduling under Network Calculus

Both schedulers are treated here within the DM as delay based schedulers, which means the decision for the next packet to service is done by the remaining delay budget per packet. A delay budget is spanning the time from a packe's arrival to its latest possible departure. The general schedulability criterion is derived from the maximum horizontal deviation between the sum of all arrival curves and the available service $\beta = \lambda_C$. If it is finite and not greater than the maximum allowed delay budget, the set of arrival curves is schedulable.

$$\sum_i \alpha_i(t - d_i) \leq \beta \tag{9.23}$$

In the case of the DM, this presents a problem, as each packet's delay budget is referring to its execution at the endpoint, not the departure time at that particular DM node. Each local deadline is a part of the total delay budget, but it is unknown. A rough estimate can be given once all static delays are known and subtracted. If all delays from cross traffic are bounded as well, exact calculation is possible, but this does not provide any additional information at this point. The scheduler equation can be used, however, as a simple and fast instrument to detect overload by a set of flows before attempting a full delay analysis.

9.4.2 Soft-CPU Scheduler

The input flows in CPUs are derived from timing messages in machine schedules. It is the purpose of the CPU scheduler to chronologically sort and aggregate messages from all threads and send them as early as possible within the time window of $D_j - \Delta_t$.

Joining Two Worlds The very first point to address is the existence of several distinct domains within the DM: CPU, WB bus and Network. The latter two relate and are thus trivial to describe in NC, as they only differ in bandwidth and in the network always being packetised while WB is a cycle based bus. The relation between programs in a DM CPU and bus/network activity is not trivial though and we shall start modelling the DM with an approach for a CPU/traffic relation.

CPU Activity vs Generated Traffic An LM32 CPU can be described as an NC node, offering a constant rate service (operations executed over time), and a program as a flow (cumulative operations over time). The output flow (of interest) is all bus activity downstream towards the network interface. This means that a program is already an aggregate of flows. They are flows that generate traffic downstream and flows that do not, i.e. message and overhead flows. The composition of overhead and its impact on message service are discussed here.

Overhead Concept Consider the following: The processor arbitrates its computing power between a number of N threads and the scheduler itself. According to chapter 8.6, it is assumed that all tasks have a deterministic execution time which is previously known. The scheduler itself also has a deterministic execution time. So there is not only a CPU rate, but also a message rate, the maximum rate the firmware can send messages at. While formal investigation of program execution time can be a highly complex and computationally intensive task, measuring the maximum message rate using the CPU cycle counter or a logic analyser is trivial.

The scheduler further has a dispatch function f, which transforms a skeleton message from RAM into a timing message on the bus. Let there be another function g which sends synchronisation messages to CPUs. The function g does not produce messages on the timing network, and since it uses the MSI WB bus, it has no impact on the normal WB traffic neither. Because the effort of preparing and sending synchronisation and timing messages is very similar, f and g can be assumed to have equal execution times.

Sync messages are part of machine schedules, all message flows therefore have an associated sync overhead flow. This produces the interfering flows in the CPU node. Sync flows are of no further interest to the analysis, as they are extracted again directly after the CPU node. Only their effect on the CPU's residual service curve to the messages is considered.

Overhead Flows Figure 9.11 shows the block diagram of the CPU service node, a constant rate server used by the message flows and several interfering overhead flows.

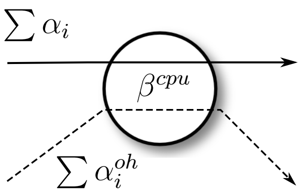


FIGURE 9.11: CPU Scheduler node

The leftover service available to all timing messages is the total service of the CPU after per flow sync overhead α_{oh_i} has been subtracted:

$$\beta^{\text{cpu}} = \lambda_{r_3} \ominus \sum_i \alpha_i^{oh} \tag{9.24}$$

Actual Implementation We know there exists a limit Δ_t dictated by the control loop speed, which is the end-to-end delay budget of a message. In the present case, it signifies the minimum time before its deadline D_j a message shall be dispatched.

The actual implementation determines the task with the minimum deadline at the very moment its predecessor was serviced. Afterwards a separate check is run periodically if this message is eligible for dispatch, that is, if $t \geq D_j - \Delta_t$. If it is positive, the message is sent. The rate of this check is the same as the service rate offered to messages by the CPU.

Simplification Representation in NC can be simplified by reordering these steps and crafting slightly different input flows. Instead of letting new messages arrive immediately after service, messages can be placed in accordance with their arrival times $D_j - \Delta_t$. This already takes care of the eligibility window Δ_t. Feeding such a flow through the CPU's service window will then introduce the same delay as in the prior case.

CPU Schedulability It is first necessary to obtain information about the possible size of the delay budget d for the scheduler input flows. However, it is not possible to determine the budget in the present case. The delay budget for EDF schedulers is defined as the maximum time between arrival and departure *at the server containing the scheduler*. In the present case, this partial budget is unknown - only the end-to-end budget is. A loose approximation can be obtained by deducting the sum of all static delays $\sum \delta$ (which will be deduced in this chapter) from the end-to-end budget Δ_t. Note that being schedulable is no guarantee for timely arrival with regard to the endpoint, but unschedulable flows are guaranteed to be late. A general schedulability criterion for each processor node is:

$$\begin{gathered}\sum_i \alpha_i(t-d) \leq \beta^{cpu}(t) \\ \text{with } d = \Delta_t - \sum \delta\end{gathered} \tag{9.25}$$

9.4.3 Processor Output

Since β^{cpu} is known, the output flow of a CPU can be derived. For TFA, the output flow and arrival curve can be calculated with the aid of the sum of all inputs:

$$\begin{aligned} R^* &= \sum_i R_i \quad \otimes \beta^{cpu} \\ \alpha^* &= \sum_i \alpha_i \quad \oslash \beta^{cpu} \end{aligned} \tag{9.26}$$

The leftover service curve required for both SFA and TFA on the other hand can be obtained from a slight variant of eq. 9.24. The overhead caused by the scheduler, all sync flows and all interfering message flows can be subtracted from the CPU's service, resulting in the residual service for the message flow of interest:

$$\beta_{\text{foi}}^{\text{l.o.}} = \lambda_{r_3} \ominus \left(\sum_i \alpha_i^{oh} + \sum_{i \neq foi} \alpha_i \right) \tag{9.27}$$

9.4.4 Priority Queue Scheduler

We now have the processors' outputs, which are packet flows with wide-sense increasing deadlines. The next node on the path is the hardware PQ, the second layer of EDF schedulers in the DM. Its purpose is the chronological aggregation of all input flows into one output flow, ordered by deadlines.

PQ Schedulability The constraint is that there must be no back-pressure to the CPU, so overflow of the input queues is not permitted. This is necessary to keep program execution deterministic and hence maintain the schedulability criterion. We will first derive an upper limit to the delay budget d_i from the buffer capacity of the input queue. The input queue can take in data at a rate of r_3, so it offers a maximum service γ_{r_3}. The delay d_i must therefore be the time it takes an incoming flow constrained by α_i (which must be sub-additive) to fill an input queue of size b. Since γ_{r_3} poses an upper limit on the input rate, we will convolute it with α_i, the min-plus convolution of two sub-additive functions being the minimum [123, p. 113] of both. This will result in the following schedulability criterion:

$$\sum_i \alpha_i(t - d_i) \leq \beta(t)$$
$$\text{with } d_i = min\{s > 0 : min(\alpha_i, \gamma_{r_3})(s) = b\} \tag{9.28}$$

Abstract Model - Service Curves In the analysis of the abstract model of an EDF, the maximum delay depends solely on the total flow passing through the scheduler's constant rate node. In this case, a service curve containing a fixed delay T_a for evaluation of each timestamp and the impact of the packetiser in each queue ($\beta_{r3,\frac{l_p}{r3}}$) must be included, as all inputs experience this. This can be written as

$$\beta^{ch} = \delta_{T_a} \otimes \beta_{r3,\frac{l_p}{r3}} \tag{9.29}$$

The complete EDF scheduler in the PQ module features M channels, each connected to the constant rate node of the PQ.

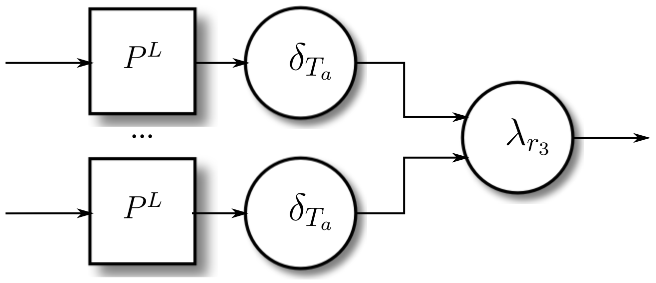


FIGURE 9.12: PQ Scheduler node

And with equation 9.29, we can finally calculate the residual service a single flow (at the PQ, which already can be aggregates) would experience:

$$\beta^{l.o.}_{foi} = \lambda_{r_3} \ominus \Big(\sum_{j \neq foi} (\alpha_j \oslash \beta^{ch}) \Big) \tag{9.30}$$

9.5 Etherbone Master – Framer

The EBM is responsible for wrapping WB accesses to other systems in the EB protocol, it creates a network packet and hands it over to the network interface.

9.5.1 EBM Functional Recap

The EBM gathers Wishbone Bus Operations and a framer sub-module analyses type, order and destination. It then generates appropriate EB record headers and inserts them as required. Once dispatch of the opened packet is requested, the EBM finalises and inserts the network header information, and starts transmission. More details can be found in chapter 7.5. Because the EBM is controlled by the PQ in the present case, requests for dispatch can be caused by reaching the size limit or by hitting a timeout.

Following the path from the PQ downstream, we will begin by modelling the framer sub-module of the EBM.

9.5.2 Input Parser

The overhead produced by the framer depends on the WB operations. In the case of DM traffic though, only timing messages will arrive. These follow a fixed format of 8 consecutive write operations to the same destination, which makes the introduced record overhead constant (see chapter 7.5).

Timing messages must not be split, and therefore the payload of EB records will be of a constant length l_p. This is packetisation at message level, and so the framer needs to contain a fixed length packetiser (see section 9.2.4) in the payload flow. Because l_p is constant, the length function $L(n)_p$ of the n-th payload packet is

$L(n)_p = n \cdot l_p$. This results in the following service curve for the message payload L-packetiser:

$$\mathrm{P}^{\mathrm{L}}_{\mathrm{p}}(\mathrm{x}) = \sup_{n \in \mathbb{N}} \left\{ l_p n \cdot 1_{\{l_p n \leq x\}} \right\} \tag{9.31}$$

The framer now needs a shaper prefixing the L-packetiser, which would be a guaranteed rate node of some arbitrary rate λ_r. The time T_a the framer requires to analyse the input needs to be accounted for. Because the analyser is pipelined and the input format fixed, this delay can be expressed as a simple guaranteed delay node δ_{T_a}. The two nodes can be concatenated to form a rate-latency node as the shaping curve σ:

$$\sigma_{r,T_a} = rt + T_a \tag{9.32}$$

Impact It is known from [123, p. 43] that a packetiser offers a minimum service described by the rate of the bit-by-bit system and the maximum delay from packet buffering, $\beta_{r,\frac{l_{max}}{r}}$. The impact of payload's packetised greedy shaper on the system's service is:

$$\beta_p = \beta_{r,\frac{l_p}{r}} \otimes \delta_{T_a} \tag{9.33}$$

9.5.3 Header Generation

The framer must now prefix each timing message with an appropriate EB record header of constant length l_h, which will create EB records of length $l_h + l_p$. While the header themselves are of no particular interest to an analysis, they are necessary in terms of the system service they consume. There are two different strategies to discuss through which injection of overhead can be modelled.

Scaled Flow Approach Consider the knowledge about the expected input and output flows. The output is obtained by injection of data of size l_h every l_p in the input flow. Assuming header insertion would happen instantaneously, gives

$$R^*(t) = R(t) \cdot \frac{l_p + l_h}{l_p} \tag{9.34}$$

Eq. 9.34 shows header injection to be in fact data scaling, if a bijective relation between header and payload size exists. The overhead can be modelled by application of a scaling function (see "Scaler", p. 145) in the EBM and the inverse function in the timing receiver. The scaling curve and its inverse are:

$$\begin{aligned} S_R(a) &= a \cdot \frac{l_p + l_h}{l_p} \\ S_R^{-1}(a) &= a \cdot \frac{l_p}{l_p + l_h} \end{aligned} \tag{9.35}$$

9.5.4 Output Flow and Service Curves

Scaling is the simple and accurate representation of constant header insertion into packets of constant size.

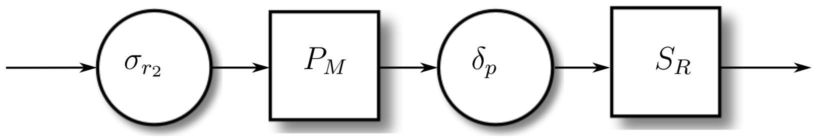


FIGURE 9.13: Block Diagram of EBM Framer Module

Figure 9.13 shows the resulting block diagram with packetiser, parser and scaler. With eq. 9.31, 9.32, 9.33, and 9.35, the output flow and service of the EBM framer can be modelled using the following equations:

$$R^*_f = S_R\big(P^L_p(\sigma \otimes R) \otimes \delta_{T_p}\big) \qquad 9.31, 9.32, 9.35 \qquad (9.36)$$

$$\beta_f = \beta_{r_2, \frac{l_p}{r_2}} \otimes \delta_{T_p} \qquad 9.33, 9.35 \qquad (9.37)$$

Scaling It is noteworthy that the service does *not* include the scaling function S_R, although the output flow R^*_f does. Fidler and Schmitt show [126] that the order of scaling and service elements is interchangeable within a certain rule-set. In order to obtain a suitable equivalent system, it is necessary to consider the complete network, not individual modules.

All scaling effects spilling over to downstream modules are therefore ignored until reaching the analysis section 9.10. Instead only the scaling functions and unscaled service equations are provided.

9.6 Etherbone Master – TX

The purpose of the TX sub-module is to take in a variable number of EB records and, by prefixing it with a packet header, turn them into network packets. The maximum packet length and how long TX should gather EB records before producing a packet is configurable.

Header and Payload Size Let l_{nh} be the aggregated size of all headers for the Ethernet IEEE802.3, IPv4, UDP and EtherBone protocol. Let l^S_p be the length of a timing message with an EB record header. The number of messages going into the same packet has a constant upper limit given by the maximum payload size, $l_{max} - l_{nh}$, and a constant lower limit of one message, l^S_p.

9.6.1 Variable Length Function

It is clear that waiting for a full packet is not an option, since the delay would be inversely proportional to the arriving flow. This is an undesired effect, as incoming flows would need to be padded to reduce delay and an equilibrium between transmission delay from the rest of the system and waiting time in the EBM TX would have to be found.

Timeout A timeout was introduced to bound the delay, which requires limiting the maximum waiting time for the first message to enter a packet. This results in a variable payload length $l_to(t)$. The payload packetiser therefore employs a length function $L(n)$, which provides cumulative packet lengths with a step-size being the minimum of l_{max} and the level at the timeout, l_{to}. Since $l(n) = L(n) - L(n-1)$, it is possible to calculate $L(n)$ from $L(n-1)$.

The timeout starts at the first message entering a packet, which is the case when $R'(t)$ (which is $R(t)$ after the packetisers bit-by-bit system) crosses the packet boundary (now $L(n-1)$). With $F(t)$ being the flow level at time t, we get

$$\mathrm{g(x)} = \inf_{s\in\mathbb{R}_+^*} \{s : F(s) > x\} \tag{9.38}$$

The timeout occurs after a timespan T. Unfortunately, this will introduce a problem: While R is packetised to be always multiple of l_p^S, it has to pass a bit-by-bit shaper, becoming $F = (R \otimes \sigma)(t) = R'$. Therefore, $R'(g(x) + T)$ can return a level right in the middle of a message (eq. 9.39).

$$\begin{aligned} &M = \{k \cdot l_p^S\}, \quad k \in \mathbb{N} \\ &x \in M && \rightarrow && F\big(g(x)\big) \in M \\ &T \in \mathbb{R} && \rightarrow && \exists T : F\big(g(x) + T\big) \notin M \end{aligned} \tag{9.39}$$

To guarantee the delay bound, it is not possible to wait for a commenced message to fully arrive. To only put complete messages into a packet, it is necessary to round $R'(t)$ down to the nearest multiple of l_p^S. This means applying a floor function, which is equivalent to run $R'(t)$ again through the L-packetiser of the EBM framer:

$$\mathrm{R''(t)} = \left\lfloor \frac{R'(t)}{l_p^S} \right\rfloor \cdot l_p^S = P_f^L\big((R \otimes \sigma)(t)\big) \tag{9.40}$$

And with eq. 9.40 and $F = R''$ so L-function for the network payload is:

$$\mathrm{L_{np}(n)} = \min\big\{(L(n-1) + l_{max}), R''\big(g(L(n-1)) + T\big)\big\}, \qquad n \in \mathbb{N} \tag{9.41}$$

9.6.2 Header

The present case requires packets of variable payload length $l(n)$, yet with a header of fixed length l_{nh}. As mentioned in [126, p. 290 (4)], scaling functions can be applied to the length function $L(n)$ of a packetiser. $L(n)$ is only point-wise defined for $n \in \mathbb{N}$ though, while scaling functions must be continuous.

$$\mathrm{P^L(x)} = \sup_{n\in\mathbb{N}} \big\{L(n)1_{\{L(n)\leq x\}}\big\} \tag{9.42}$$

However, the L-packetiser function from eq. 9.42 is defined for all real numbers [123, p. 42], thus allowing the extension of $L(n)$ into the $\mathbb{R}^*_+$ domain by assigning each $a = R(t)$ a value from $L(n)$. So having obtained a continuous scaling function, scaling curves can be derived.

Scaling Function The L-packetiser equation is modified by a scaling function for constant header insertion. This means that each packet length $l(n)$ must be scaled by the addition of l_h, so that $S^p_P(l(n)) = l(n) + l_{nh}$. With the definition of the length function given by $L(n) = L(n-1) + l(n) \to L(n) = \sum_i^n l(i)$, the point-wise defined scaling function S^p_P now becomes:

$$S^p_P(L(n)) = \sum_i^n (l(i) + l_h) = L(n) + n \cdot l_{nh}$$

Combination with the L-packetiser equation 9.42 provides the scaling function S_P defined for $\mathbb{R}$, swapping the pre-factor for the indicator function with its condition provides the inverse:

$$\mathrm{S_P(a)} = \sup_{n\in\mathbb{N}} \left\{ (L(n) + n \cdot l_{nh}) 1_{\{L(n)\le a\}} \right\} \tag{9.43}$$

$$\mathrm{S_P^{-1}(a)} = \sup_{n\in\mathbb{N}} \left\{ L(n) 1_{\{L(n)+n\cdot l_{nh}\le a\}} \right\} \tag{9.44}$$

The appropriate minimum and maximum scaling curves can once again be derived from max-plus and respectively min-plus deconvolution of the scaling function with itself.

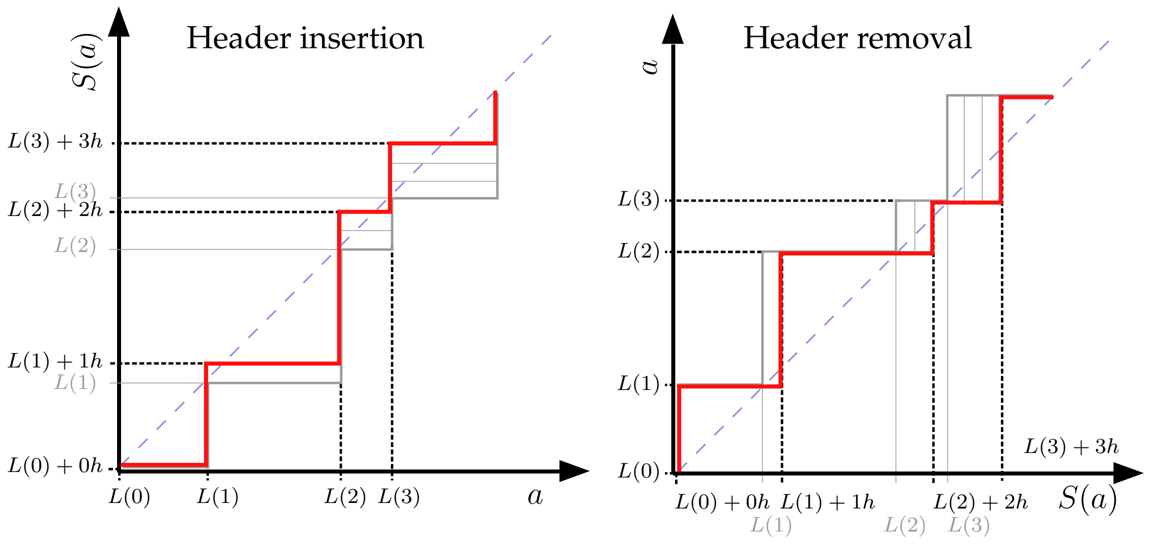


FIGURE 9.14: Packet Header Scaling Functions

Figure 9.14 illustrates the scaling functions (thick red line) for packet header handling. The left plot shows header insertion with eq. 9.43, the right the removal via the inverse function, eq. 9.44. The diagonal mirror axis is sketched in to demonstrate function inversion.

9.6.3 Finding Limits for Payload Length and Timeout

With the presented scaling curve, it is possible to calculate the exact introduced overhead at any given point in time for a specific flow. However, by careful choice of the payload limit and timeout value of the packetiser, it is possible to constrain the setting in a way that allows simplification of the scaling curve to a constant factor.

Impact of Payload Length The behaviour of the TX module is governed by two parameters, the allowed payload length Φ and the timeout T. Bandwidth utilisation depends on the overhead to payload ratio, the larger the payload, the better. As Φ approaches $l_{max} - l_{nh}$, that is, maximum possible packet length ($1500\,\mathrm{B}$) without the header, bandwidth utilisation is at its optimum with $r_2\frac{\Phi}{l_{max}}$. Lowering Φ splits the payload into more packets, which generates more overhead and therefore directly consumes extra bandwidth. Because latency in a packetiser is determined by the maximum packet length over rate, lowering Φ also lowers latency.

Impact of Timeout The necessity of $T \geq \frac{\Phi}{r_2}$ immediately becomes obvious. If T was less, the packetiser could never reach Φ before hitting the timeout and so a lower limit for T has been found. We shall now establish a sensible upper boundary for T.

A packet size l processed over a timespan T is an expression of bandwidth. Φ being set, we can now employ T to choose the bandwidth. Because the DM is the only source of high priority traffic, unused bandwidth is equivalent to bandwidth lost to overhead. The proposition is therefore that it is allowable to introduce additional overhead without changing maximum throughput, as long as the combined traffic does not exceed the maximum bandwidth at the system's bottleneck.

The bottleneck is encountered at the WR network, the switches having the lowest rate at r_1. This is in turn down scaled by a factor S_F^{-1} because, as discussed in section 9.7, forward error correction will introduce further redundancy. The resulting bandwidth is denoted as the system's sustainable rate r_s. This can be used to propose the limits for T:

$$\frac{\Phi}{r_2} \leq T \leq \frac{\Phi + l_{nh}}{r_s} \tag{9.45}$$

The reason is that if a maximum sized payload plus its header can be processed at the systems sustainable rate within the timeout, any reduction in payload flow will create more overhead. However, if the sum stays within the sustainable rate, backlog from overhead cannot accumulate. So $T = \frac{\Phi + l_{nh}}{r_s}$ would ensue an optimal bandwidth utilisation. In the absence of any other high priority source, eq. 9.43 and 9.44 can be simplified to:

$$\mathrm{S_{Ps}}(\mathrm{a}) = \frac{\Phi + l_{nh}}{\Phi} \tag{9.46}$$

$$\mathrm{S_{Ps}^{-1}}(\mathrm{a}) = \frac{\Phi}{\Phi + l_{nh}} \tag{9.47}$$

If any lower latency is required, it can be obtained at the loss of bandwidth to overhead by decreasing T_{tx}.

Absolute Figures The total delay budget for the system was given as $\Delta_t = 500\,\mu s$, so it is interesting whether to test the obtained boundaries chosen for T actually fit within this frame.

$$T_{min} = \frac{\Phi}{r_2} = \frac{1440\,\mathrm{B}}{2\,\mathrm{Gbit/s}} = 5.76\,\mu s \tag{9.48}$$

$$T_{max} = \frac{\Phi + l_{nh}}{S_F^{-1}(r_1)} = \frac{1496\,\mathrm{B}}{0.25 \cdot 1\,\mathrm{Gbit/s}} = 47.78\,\mu s \tag{9.49}$$

At 1.2% of the total delay budget, we can assume T_{min} to be a safe choice. T_{max} however, at 9.5%, should be examined again in the final analysis.

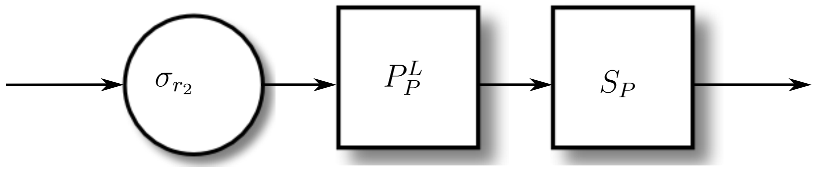


FIGURE 9.15: Block Diagram of EBM TX Module

Service The minimum (and because of the fixed timeout also maximum) service curve for the TX module is that of a standard packetiser, but the delay is solely determined by T. Note that scaling is applied last and therefore not part of the service curve of this module. The resulting curve is thus very straightforward:

$$\beta_{tx} = \beta_{r_2,T_{tx}} \tag{9.50}$$

9.7 Forward Error Correction

Context and Necessity The CS ultimately needs a central instance (separate, but synchronised instances are just an equivalent model) which communicates that servers providing the physical calculations to control the accelerator. This again means that there must be a fan-out from the CS's DM to numerous endpoints, placing it as the root node of one or more networks of a tree topology.

The concept of the current CS relies on treating the whole system and timing network as lossless. If it was not, commands would need to be re-sent if they did not arrive at their destination, which adds the need for feedback from the endpoints. Due to the tree topology, this is already a problem because the bottleneck on the way up to the root node is getting ever tighter and a reason to avoid re-transmission. The second reason is that an upper bound on control loop delay is only possible if re-transmission does not need to be considered.

Application The chances of packet loss has to be reduced to a level at which, for all practical purposes, the system can be treated as lossless. The way to achieve this involves both increasing the system's mean-time-between-failure and employing forward error correction algorithms. The purpose of the latter is to protect both network packets *and* their meta information against bit errors, more details can be found in the work of Prados Boda and Fleck, [79].

Impact on the Model In the scope of this work, the FEC will be treated as a black box system. The observable effect is the creation of k interleaved packets from one incoming packet after a packetisation and encoding delay.

9.7.1 FEC Encoding

The FEC re-packetises to the length set in the EBM TX modules, then starts the encoding process, buffers the resulting encoded packets and finally sends the encoded (scaled) data.

Packetiser Because the packets are of variable size, the first packetiser does introduce a delay equal to $\frac{l_{max}}{r_2}$. After encoding, the second packetiser has to deal with the scaled version of the packets, thus adding a delay of $k \cdot \frac{l_{max}}{r_2}$.

Encoding Time We will assume a constant encoding time in the FEC, introducing a delay of T_e, which already includes the introduced $k-1$ inter-frame gaps between the generated packets. Additionally it is assumed that a time $T_d \geq T_e$ is required to decode the information again in the timing receiver.

Scaling Data will be scaled up to mimic the FECs introduction of redundant data. Similar to the EBM framer in 9.5, this can be described by the application of a scaling curve for the FEC, S_F, which multiplies packet size by k, the number of packets generated.

The resulting sub-system, packetiser, delay and scaler and output packetiser, is shown in figure 9.16.

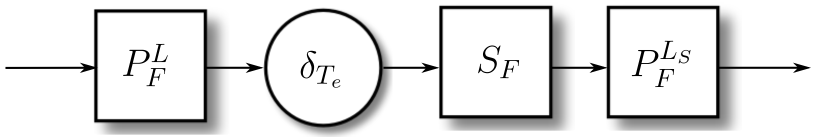


FIGURE 9.16: Block Diagram of FEC Module

The resulting scaling functions are:

$$\mathrm{S_F}(\mathrm{a}) = k \cdot a$$
$$\mathrm{S_F^{-1}}(\mathrm{a}) = \frac{1}{k} \cdot a \tag{9.51}$$

Service The rightmost packetiser in figure 9.16 is expecting the scaled data. We can therefore apply the scaling function to the maximum packet size and obtain the following service curve for the FEC:

$$\beta_{\mathrm{F1}} = \beta_{r_2, \frac{l_{max}}{r_2}} \otimes \delta_{T_e} \otimes \beta_{r_2, \frac{S_F(l_{max})}{r_2}} \tag{9.52}$$

9.7.2 Timing Receiver

FEC Decoding Interest lies in determining an *end-to-end* delay bound for the timing system. In the present case, the decoding happens in the timing endpoint, which is why it is necessary to also model this part of the timing receiver in some detail.

Gathering Packets and Decoding Time For forward error correction, only a part of the packets belonging to one transmission need to arrive. It is necessary to assume the worst case though, which means waiting for the last packet to arrive. The first step is therefore re-packetising all arriving packets from one transmission into one big packet. Afterwards, the packets are decoded, which makes the whole DFEC use the same equation as the FEC. The service curve is similar to eq. 9.52.

Symmetric Scaling As already presented in section 9.2.4, a symmetric scaling variant is applied, which requires the corresponding decoder to apply the inverse of the original scaling function.

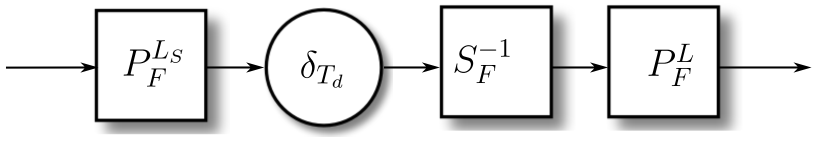


FIGURE 9.17: Block Diagram of the DFEC Module

$$\beta_{\mathrm{F2}} = \beta_{r_2, \frac{S_F(l_{max})}{r_2}} \otimes \delta_{T_d} \otimes \beta_{r_2, \frac{l_{max}}{r_2}} \tag{9.53}$$

9.8 Etherbone Slave and Event-Condition-Action Unit

Demultiplexing Removal of the packet header happens instantaneous at the inverse scaler. So the EBS RX block diagram looks like a mirrored version of the EBM TX (see figure 9.15 and 9.18). The EBS de-framer does not work the same way as the EBM framer, it removes the EB record header and adds a delay, but does not re-packetise. This can therefore be described as an inverse scaler followed by a rate-latency system (see lower right of figure 9.18, “EBS Deframer”).

This finally leaves the ECA unit (see 7.6). Being one of the most complex logic cores in the system, it can nevertheless be described as a simple black box model. ECA is responsible to schedule actions originating from arriving messages to be executed at the timestamp they carry. To sort arrivals, ECA adds a bounded delay of $4\,\mu s$.

This would be followed by an EDF scheduler, which is not modelled. The reason is that all messages are dispatched $\Delta_t = 500\,\mu s$ before they are due. For all messages that arrive on time, the EDF would hold them back until their execution time. This would hide the leftover delay budget from the analysis.

9.9 White Rabbit Network Model

9.9.1 Interference at NIC

Origin At the network interface, the DM's flow is multiplexed with flows from the WR timing core. WR uses PTP packets to synchronise the UTC time between timing receivers/switches. In addition, there are other services spuriously sending packets, like the Address Resolution Protocol (ARP), Dynamic Host Configuration Protocol (DHCP) or Simple Network Management Protocol (SNMP).

Approach for DM HP Service DM traffic has the highest priority of all services. However, pre-emption is not allowed, so the minimum service to the DM has to consider waiting for transmission of the longest possible low priority packet. Packet lengths of lower priority services can be described as $l_{\text{<name>}} \sup_{n \in \mathbb{N}}\{l_{\text{<name>}}(n)\}$. The longest possible low priority packet is thus $l^{lo}_{max} = \max\{l_{ptp}, l_{arp}, l_{dhcp}, l_{snmp}\}$, resulting in the DM a minimum service of

$$\beta_{\text{N}} = \beta_{r_1} \ominus l^{lo}_{max} \tag{9.54}$$

Improving WR PTP performance WR PTP periodically has to send packets to synchronise UTC time to counter long term drift against the time reference (GPS receiver). It is questionable if a system with only two priorities is a good design choice because a continued starvation of WR PTP service would lead long periods of uncompensated clock drift in all downstream timing switches and receivers. It would therefore be sensible to introduce another priority level between DM and background for WR PTP, and allocating a minimum service rate for clock synchronisation.

PTP needs periodic adjustment, so it is assumed that all WR PTP flows are periodic. The corresponding staircase functions are sufficient to model the arrival curves [123, p. 8]. The presented approach models the mutual influence on service by application of one shaper per flow. This limits the influence of the interfering flow to its maximum allocated rate. Assuming a fraction of the total rate k is assigned, with $k \in \mathbb{N}^*$, to WR traffic. It would then be forced to obey $\sigma_{ptp} = \lambda_{\frac{r_1}{k}}$, making sure α_{ptp} (as an interfering flow) does not exceed this rate in the presence of DM traffic. It follows that the shaper for the DM traffic must guarantee the agreed minimum rate to WR, which leads to $\sigma_{dm} = \lambda_{r_1 \cdot \frac{k-1}{k}}$. The shaping curves are the guaranteed service for each flow, and $\lambda_{r_1} \geq \sigma_{ptp} + \sigma_{dm}$. The maximum length packet to be considered in the delay equation differs for WR and DM flows, as DM is of highest priority, WR of second highest and all others of lowest priority. The leftover service for each of the higher priority flows can be calculated as:

$$\alpha_{\text{ptp}} = l_{ptp} \cdot \upsilon_{T_{ptp},0} = l_{ptp} \cdot \left\lceil \frac{t+0}{T_{ptp}} \right\rceil = \gamma_{\frac{l_{ptp}}{T_{ptp}}, l_{ptp}} \tag{9.55}$$

$$\beta_{\text{nic}} = \lambda_{r_1} \ominus (\alpha_{ptp} \oslash \sigma_{ptp}) \otimes \delta_{\frac{l_{max}}{r_1}} \tag{9.56}$$

9.9.2 WR Switches

The DM is connected via a tree topology to 2000+ timing receivers. WR switches feature 18 ports each, which means a fanout of 1-17. This indicates a minimum of $k = \lceil \log_{17} 2000 \rceil = 3$ layers of switches. The topology is adjusted for geographical reasons though, so the actual size is likely to be 5 layers. WR switches are treated as black boxes. The delay they introduce has been removed from traffic measurements and is represented in a simplified model.

Switch Properties All switches treat traffic from the DM as high priority. The switches feature cut-through for low latency. This means HP packets are passed on as soon as possible, sending the first bits before their last bits have arrived. The switches are non-preemptive, meaning they must buffer (introduce a delay) if a lower priority packet is currently being transfered.

The switches are therefore modelled as a small constant delay T_s representing the time it takes to inspect the packet header and apply the switching matrix. Following this is the multiplexer, a constant rate node operating at r_1. Because the switch can be busy with a low priority packet, there is another delay of $\frac{l^{lo}_{max}}{r_1}$ in its minimum service. Because LP traffic is partly point-to-point and originates at all switches, it cannot be assumed multiplexing is applied only at the first switch. The service of a WR switch is therefore defined as:

$$\beta_{\text{sw}} = \delta_{T_s} \otimes (\beta_{r_1} \ominus l^{lo}_{max}) \tag{9.57}$$

9.10 End-to-End Delay Analysis

All sub-components of the CS have been modelled and an end-to-end delay analysis of the complete system can now be conducted. We will perform the necessary preparation for a PMOO analysis, as it (in most cases) leads to the tightest delay bounds. In the scope of this thesis, PMOO also has the benefit that the required reduction of the system into a single equivalent node allows a clearer visualisation.

For ease of representation, the system is split into four parts. The first is the sink tree posed by CPUs, aggregating flows from their threads, and the PQ, aggregating flows from CPUs. The second is the single-path section of the DM without the NIC, the third are the NICs of DM and timing receiver as well as all WR switches. The fourth and last is the timing receiver.

Packetisers in the Big Picture To reduce the size of the figures, all packetiser blocks in the following diagrams show not just L-packetisers, but already PGS, a combination of a bit-by-bit system and an L-packetiser (see figure 9.13, 9.15). Furthermore, a substitution of L-packetiser functions was applied. Packetiser P_{M_1} has a maximum packet size of $l_p = 32\,\text{B}$. Afterwards, EB record headers are added, the packets are scaled. P_{P_1} collects EB records, which is a timing message scaled with S_R. It has maximum packet (payload) size of $l_{np} = k \cdot S_R(l_p) = S_R(kl_p)$. FEC input collects payloads plus network header, which means scaling them by S_P so maximum packet size is $l_n = S_P(S_R(kl_p))$ and FEC output and DFEC collect encoded packets, means scaled by S_F, which equals a maximum packet size of $l_f = S_F(S_P(S_R(kl_p)))$. All three L-packetisers can therefore be expressed by the same L-packetiser and a scaling function. In the block diagram, all L-packetiser scaling is noted *above* the PGS' name and any scaling applying to the bit-by-bit system *below* the PGS' name.

Network Tunnel Because flows of timing messages are aggregated into network packets and separated at the endpoints EBS, they intermittently become a single flow in the model. This is called a trunk or tunnelled connection and is marked in grey in the following overview figure 9.18. More details can be found under subsection 9.10.5.

9.10.1 EDF Sink Tree

Depending on whether the flow of interest for PMOO analysis is defined as all timing message input flows or a single one, different service and arrival curves for CPU and PQ have to be used. We shall designate a flow of interest with ingress at the CPU level as R_{xy}, being the x-th flow at CPU y.

Sum of all Timing Flows If the intention is finding the delay bound for any and all of the timing flows, the arrival curves of all flows must be aggregated and the service of the PQ's constant rate note is concatenated with the single path section of the system, denoted as β_{sp}. The incoming flow at the PQ node is then defined by

$$\alpha_{\text{pq}} = \sum_{j}^{N-1} \sum_{i}^{M-1} \left(\alpha_{ij} \oslash \left(\beta^{CPU} \otimes \beta_q \otimes \delta_a\right)\right) \tag{9.58}$$

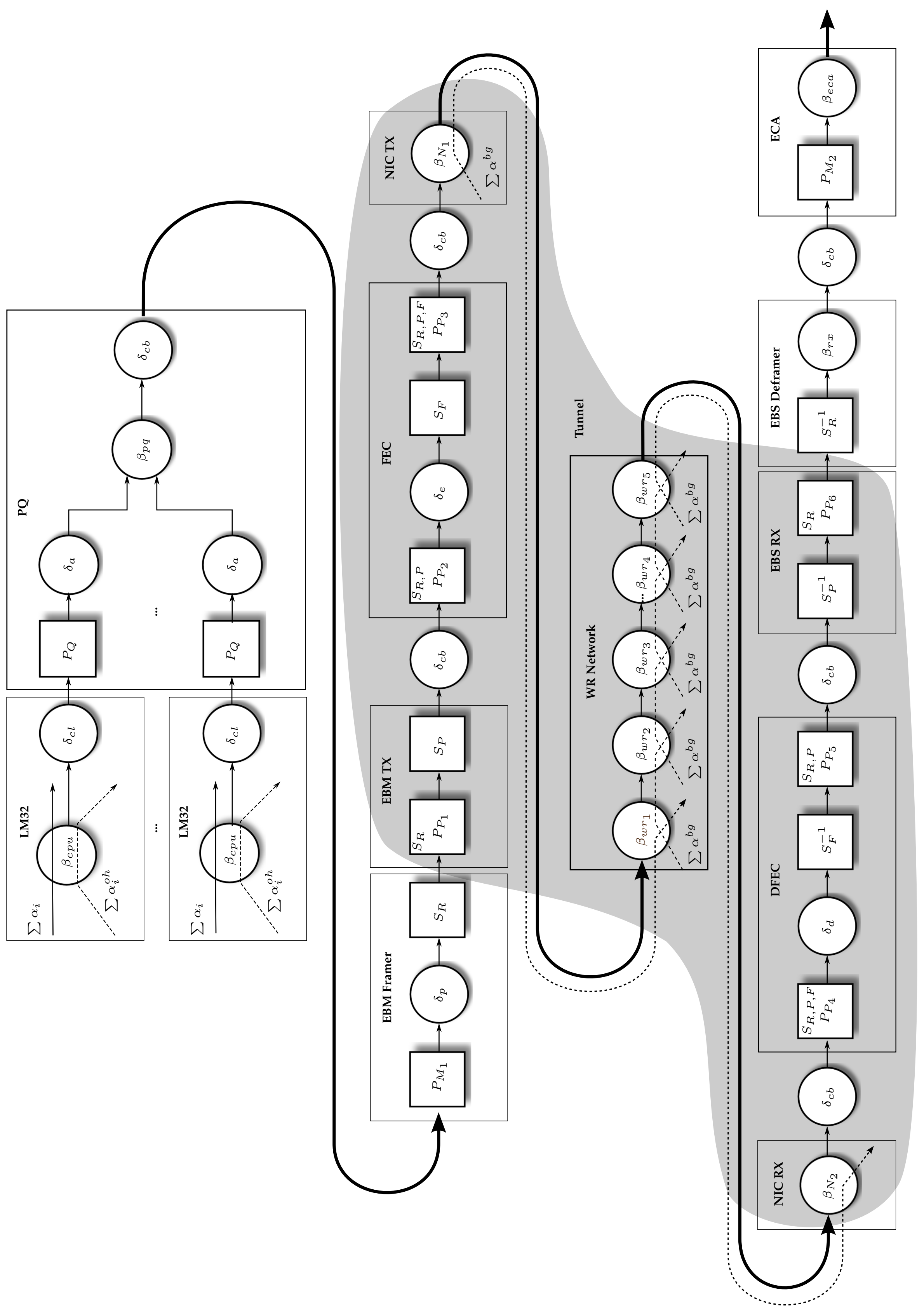


FIGURE 9.18: Block Diagram of NC Control System Model: Data Master (1st and 2nd row), White Rabbit Network (3rd row), Timing Receiver (4th row). Tunnel coverage is shown in grey

Single Flow of Interest The delay bound for a single timing message flow can be obtained by calculating the leftover service at both the CPU at which the FOI originates and the PQ. We shall start by modifying eq. 9.58 to include only the flows originating at *other* CPUs by replacing the limits of the first sum by $j \in [0, N) - \{y\}$, with y being the index of the origin CPU of the FOI.

$$\alpha^*_{-\mathrm{y}} = \sum_{j \neq y}^{N-1} \sum_{i}^{M-1} \left(\alpha_{ij} \oslash \left(\beta^{CPU} \otimes \beta_q \otimes \delta_a \right) \right) \tag{9.59}$$

We then need to add all flows originating at CPU y, except for the FOI x, and calculate the matching output arrival curve by feeding the aggregate flow through the CPUs leftover service, after serving the FOI:

$$\beta^{\mathrm{l.o.}}_{-\mathrm{xy}} = \left(\beta^{CPU} \ominus \left(\sum_{i}^{M-1} \alpha^{oh}_i + \alpha_{xy} \right) \right) \otimes \beta_q \otimes \delta_a \tag{9.60}$$

$$\alpha^*_{-\mathrm{xy}} = \sum_{i \neq x} \alpha_{iy} \oslash \beta^{l.o.}_{-xy} \tag{9.61}$$

This leaves concatenating the leftover service the FOI experienced at both CPU and PQ level. The leftover service for the FOI at CPU level is given by eq. 9.27 on p. 159. Combined with eq. 9.59 and 9.61, the leftover service in the sink tree is

$$\beta^{\mathrm{l.o.}}_{\mathrm{st}} = \lambda_{r3} \ominus \left(\alpha^*_{-y} + \alpha^*_{-xy} \right) \tag{9.62}$$

9.10.2 Equivalent Circuit for WR Network

The WR network consists of several layers of WR switches for which an equivalent node needs to be created. The NIC nodes of DM and TR are also to be combined into the equivalent system, because they also carry the interfering WR background flows (PTP, BOOTP, ARP, SNMP). The middle row in figure 9.18 shows the structure of the WR system. An equivalent node is therefore the leftover service in the concatenation of all involved nodes. Since DM traffic is high priority and non-preemptive, all nodes must allow for a maximum length low priority packet to complete. For the WR switches 9.57 and the NICs 9.54, this is already included.

It is noteworthy that the calculation of leftover service in the present case does pay for multiplexing several times. This is done on purpose, because WR background traffic is generated at all switch levels and the NICs. Therefore, multiplexing *does* happen at each node again. The equivalent service for all WR network nodes can be written as

$$\beta_{\mathrm{wr}} = \beta_{N1} \otimes \bigotimes_{1 \leq i \leq k} \beta_{sw} \otimes \beta_{N2} \tag{9.63}$$

9.10.3 Data Master to Timing Endpoint

Concatenation and Scalers The presence of scalers in the system (all S_x blocks in figure 9.18) prevents direct analysis. Nodes separated by a scalers cannot be concatenated by standard NC, so the scalers need to be removed. This can be achieved by one of three ways:

- Move all scalers to ingress
- Move all scalers to egress
- Move symmetric scalers to their inverse

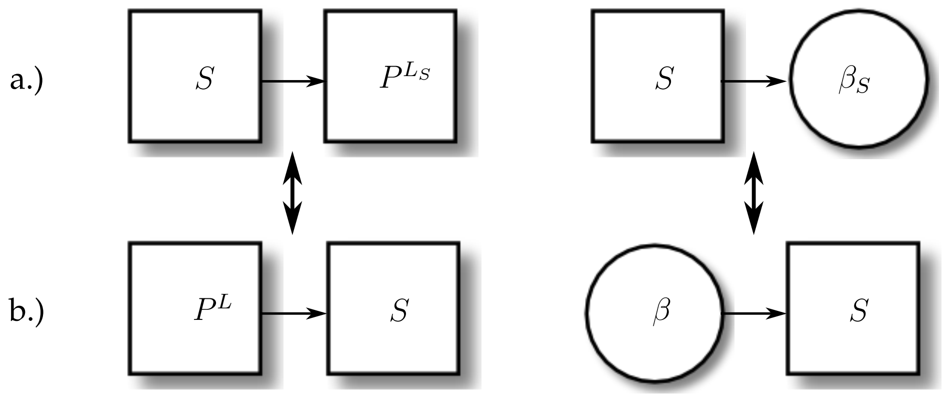
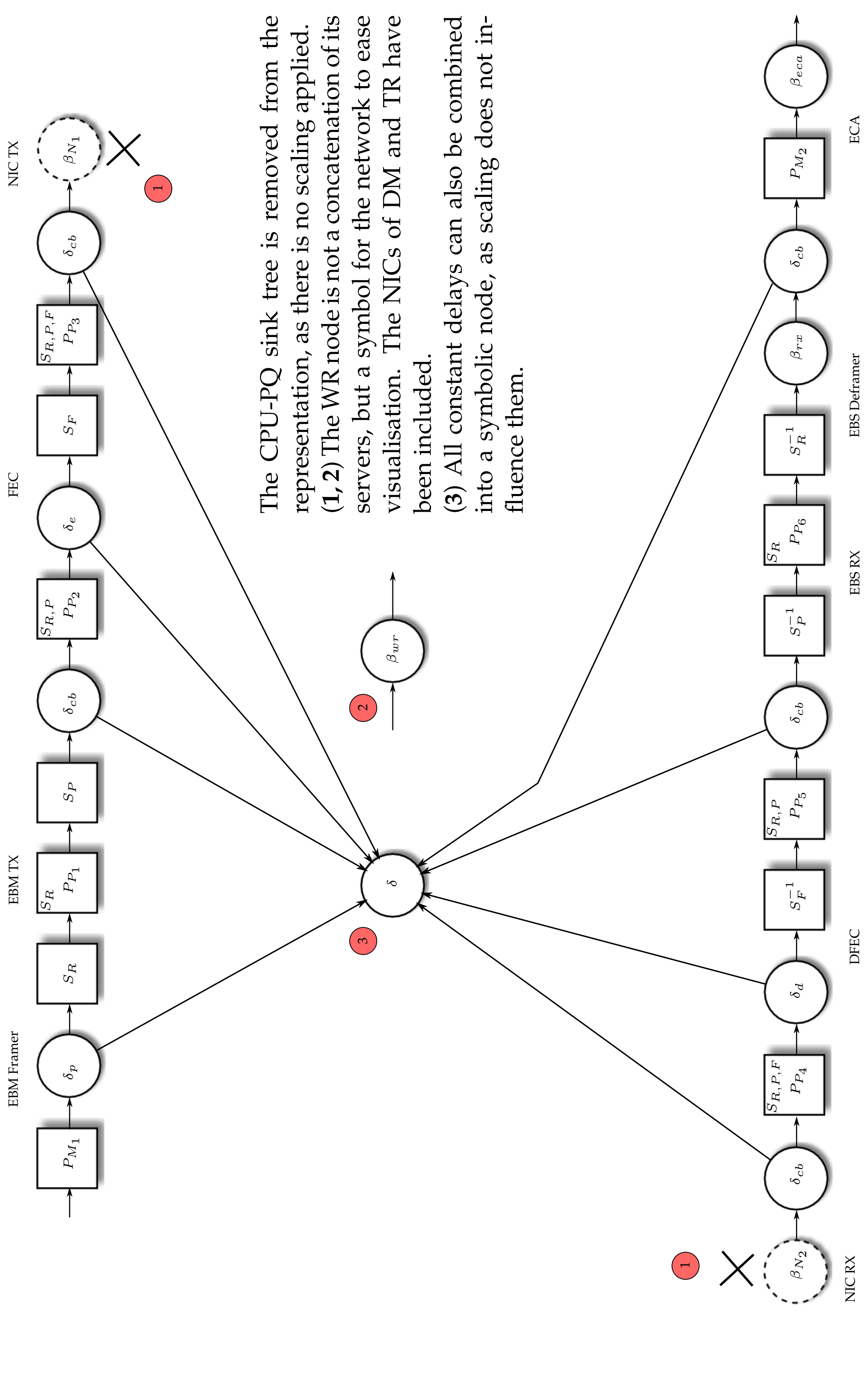


FIGURE 9.19: Equivalent Circuits for Scalers

Moving Scalers The present case has symmetric scalers, so it is possible to move them towards their respective inverse functions (S_x adjacent to S_x^{-1}) so they cancel each other out. Moving a scaler is achieved by replacing it with its equivalent circuit from figure 9.19. The replacement follows the rules for scaled service on p. 148 in section 9.2.4, the same principle holds for L-packetisers and their scaling functions.

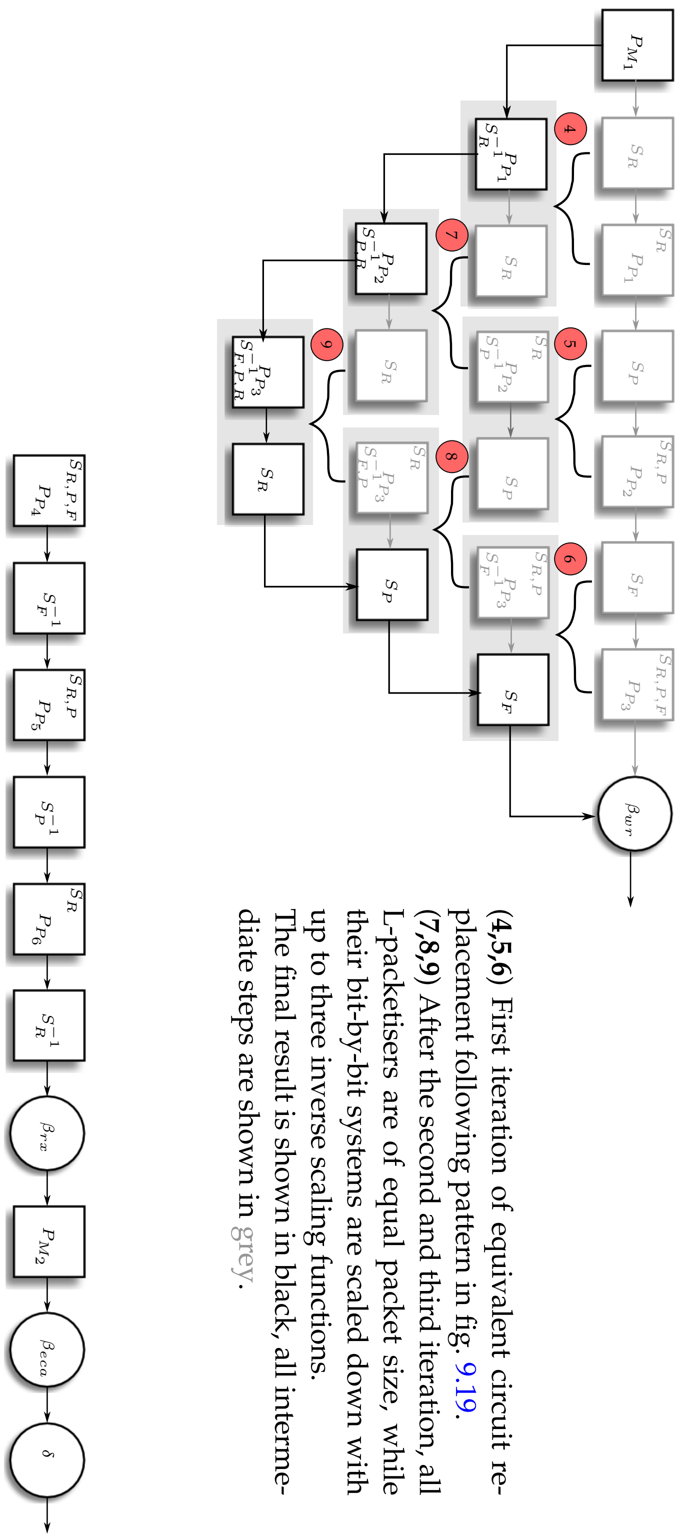
The tightness of the achieved bounds depends on the chosen equivalent circuits. The best choice differs for backlog, output and delay bounds. [126, p. 294 (8)] states that for delay analysis, a system in the a.) row of figure 9.19 should stay unchanged, and a system from the b.) row can be changed to a.). This means the three scalers S_R, S_P and S_F in the DM are to be moved downstream until they reach S_R^{-1}, S_P^{-1} and S_F^{-1} and cancel each other out.

Step-by-Step Removal While the presented method removes the scaler blocks, it is clear that their influence on other components remains. The removal process is described using the notation for packetisers and their bit-by-bit systems from p. 170.



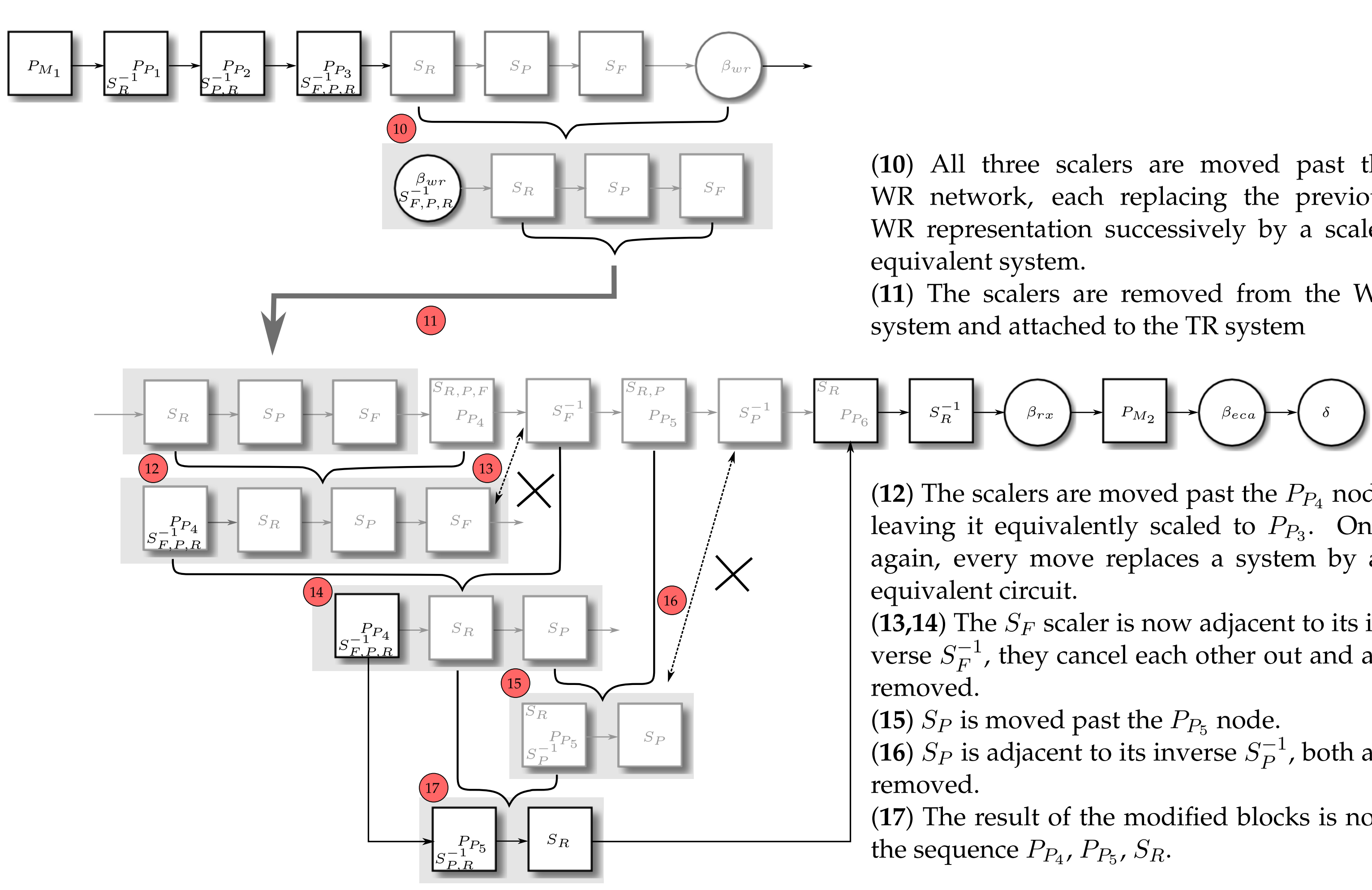
The CPU-PQ sink tree is removed from the representation, as there is no scaling applied.
(1, 2) The WR node is not a concatenation of its servers, but a symbol for the network to ease visualisation. The NICs of DM and TR have been included.
(3) All constant delays can also be combined into a symbolic node, as scaling does not influence them.

FIGURE 9.20: Introduction of Symbols for static Delays and WR Network



(4,5,6) First iteration of equivalent circuit replacement following pattern in fig. 9.19.
(7,8,9) After the second and third iteration, all L-packetisers are of equal packet size, while their bit-by-bit systems are scaled down with up to three inverse scaling functions.
The final result is shown in black, all intermediate steps are shown in grey.

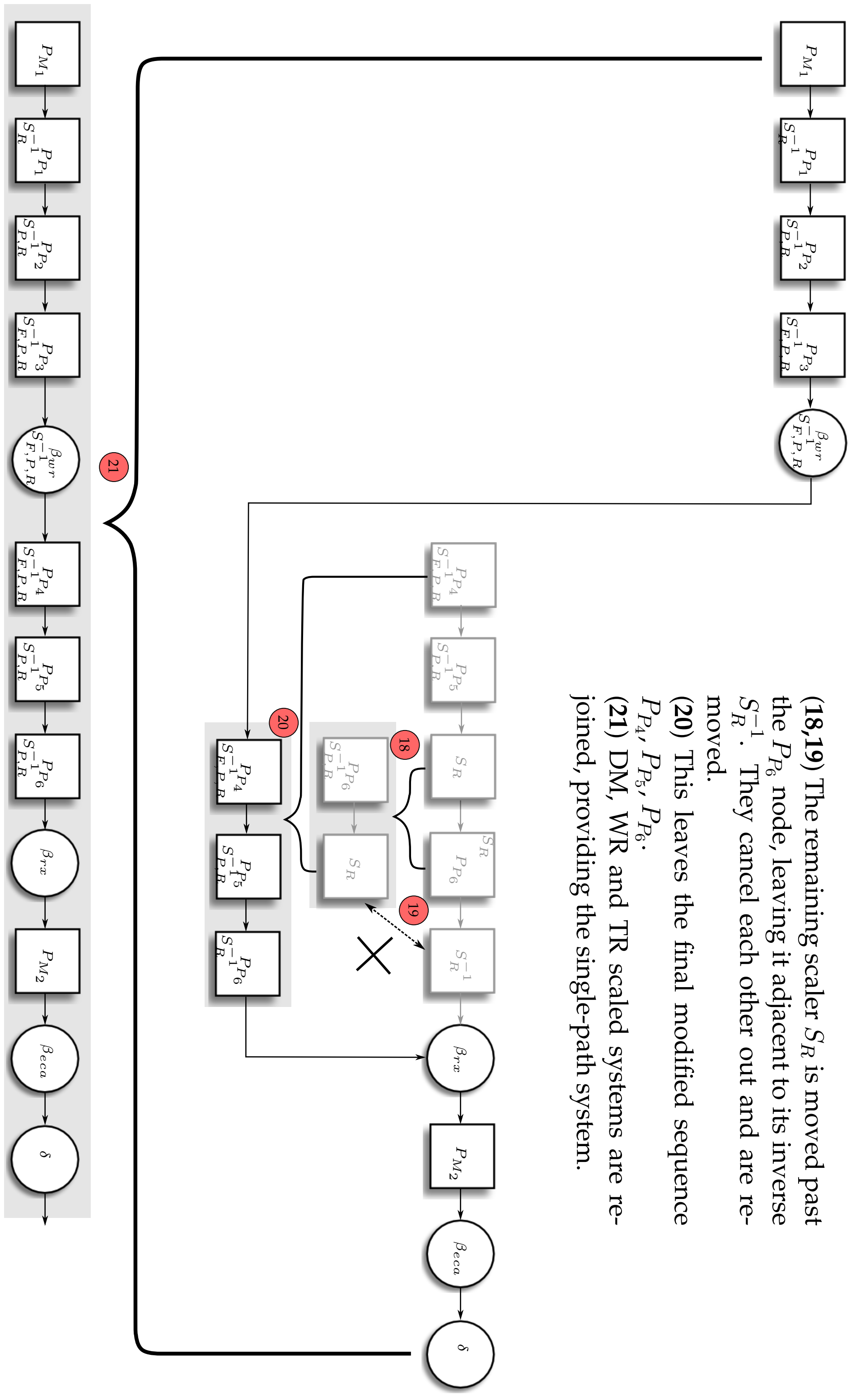
FIGURE 9.21: Replacement of DM Scalers



(**10**) All three scalers are moved past the WR network, each replacing the previous WR representation successively by a scaled equivalent system.
(**11**) The scalers are removed from the WR system and attached to the TR system

(**12**) The scalers are moved past the P_{P_4} node, leaving it equivalently scaled to P_{P_3}. Once again, every move replaces a system by an equivalent circuit.
(**13,14**) The S_F scaler is now adjacent to its inverse S_F^{-1}, they cancel each other out and are removed.
(**15**) S_P is moved past the P_{P_5} node.
(**16**) S_P is adjacent to its inverse S_P^{-1}, both are removed.
(**17**) The result of the modified blocks is now the sequence P_{P_4}, P_{P_5}, S_R.

FIGURE 9.22: Scaling WR NW and Replacement of TR Scalers



(18,19) The remaining scaler S_R is moved past the P_{P_6} node, leaving it adjacent to its inverse S_R^{-1}. They cancel each other out and are removed.

(20) This leaves the final modified sequence P_{P_4}, P_{P_5}, P_{P_6}.

(21) DM, WR and TR scaled systems are re-joined, providing the single-path system.

FIGURE 9.23: Final TR Scaler Replacement and Join with DM Blocks

9.10.4 Equivalent Service of Packetised Greedy Shapers

Moving a scaler downstream past a packetised greedy shaper will influence both the L-packetiser and the bit-by-bit system. Thus, it will transform a system of the form σ, P^{L_S} into an equivalent circuit of $\underline{S^{-1}}(\sigma), P^{L}$.

Iterative Scaling Because the minimal scaling curve (max-plus deconvolution) of scaling functions are defined to represent less or equal service, it follows that iterative use drives the curves towards pessimistic service representations, so

$$(S_X \overline{\oslash} S_X)((S_Y \overline{\oslash} S_Y)(a)) \;\leq\; S_X(S_Y(A)) \overline{\oslash} S_X(S_Y(a))$$

Therefore the process followed is to firstly apply all scaling functions, i.e. $S_X(S_Y(S_Z(a))$ …, then apply the deconvolution operator. Because the scaling functions are bijective, the order is of no consequence. To provide a more readable representation, the following notation is used:

$$\begin{aligned}
\mathrm{S}_1(\mathrm{a}) &= S_R(a)\\
\mathrm{S}_2(\mathrm{a}) &= S_R(S_P(a))\\
\mathrm{S}_3(\mathrm{a}) &= S_R(S_P(S_F(a)))
\end{aligned}$$

Using these shortforms, the equivalent service of all packetisers in block diagram 9.23 is given by the following equations:

$$\mathrm{P}_{\mathrm{M}_1} = \mathrm{P}_{\mathrm{M}_2} \qquad \rightarrow \beta_{r_2, \frac{l_p}{r_2}} \tag{9.64}$$

$$\mathrm{P}_{\mathrm{P}_1} = \mathrm{P}_{\mathrm{P}_6} \qquad \rightarrow \underline{S_1^{-1}}(\beta_P) \qquad \leq \beta_{S_1^{-1}(r_2), \frac{kl_p}{S_1^{-1}(r_2)}} \tag{9.65}$$

$$\mathrm{P}_{\mathrm{P}_2} = \mathrm{P}_{\mathrm{P}_5} \qquad \rightarrow \underline{S_2^{-1}}(\beta_P) \qquad \leq \beta_{S_2^{-1}(r_2), \frac{kl_p}{S_2^{-1}(r_2)}} \tag{9.66}$$

$$\mathrm{P}_{\mathrm{P}_3} = \mathrm{P}_{\mathrm{P}_4} \qquad \rightarrow \underline{S_3^{-1}}(\beta_P) \qquad \leq \beta_{S_3^{-1}(r_2), \frac{kl_p}{S_3^{-1}(r_2)}} \tag{9.67}$$

For the sake of completeness, a shorthand scaled version of the WR service reads

$$\beta_{\mathrm{wr_s}} = \underline{S_3^{-1}}(\beta_{wr}) \tag{9.68}$$

9.10.5 Aggregate Scheduling

The system has been modelled in terms of its service, but there is a hitherto unconsidered constraint on flows traversing the system. When flows of timing messages are multiplexed on the DM's WB bus, this follows a standard NC model. Between the DM and the endpoint however, this becomes an Ethernet based network connection.

Aggregation From the EB TX module to the EB RX module, messages are bundled into network packets (grey area in figure 9.18). This wrapping is called aggregate scheduling, or in more general networking terms, a tunnelled connection. There is a significant difference in the multiplexing behaviour, because the multiple timing message flows become one single flow of network packages while they are in the tunnel. Message flows are no longer running as cross traffic to each other and thus cannot delay each other. The resulting single flow only has the WR services as its cross traffic and because DM traffic is treated as HP without pre-emption within the WR network, only the processing times for maximum length LP traffic accumulate as latency. As a result, the latency for traversing the tunnel, and therefore the end-to-end delay, strongly decreases (compare chapter 10.4, figure 10.5 and 10.6).

For TFA, this is of no consequence, as it operates on the assumption that all incoming flows are aggregated (added) before analysis. For SFA and PMOO however, the ingress, the tunnel and the egress must be analysed as separate cases. The transition between the ingress and the tunnel is trivial, because the arrival curve entering the tunnel is the aggregate, i.e. sum of all the ingress's output arrival curves:

$$\alpha_{\mathrm{t}} = \sum \alpha^*_{in_i} \tag{9.69}$$

Regaining Individual Flows Once the tunnel ends at the EB RX module though, a problem presents itself. The output aggregate flow constrained by α_t must now be split up again into the original number of flows, which is not as trivial as it might seem. For the demonstration cases in chapter 10, the problem can be circumvented because all incoming flows were chosen to be equal. Thus, the output arrival curve of the last node in the tunnel can be divided by the number of original flows. Finding a generic solution for the corresponding residual service curves is not trivial and still work in progress in the beginning of 2017 (see chapter 11.5.1). However, there exists usable workaround for the present case. Bondorf and Schmitt proved in [129] that the maximum backlog encountered in a TFA at a given node is also the maximum backlog any other form of analysis can encounter, thus capping the backlog. Furthermore, the sustainable rates of the individual flows cannot have increased inside the tunnel. If one consequently assigns the rates of the ingress's output arrival curves $\alpha^*_{in_i}$ to α_{out_i} and sets their initial burst to the TFA bounded burst value of the aggregate output arrival curve α^*_t, all α_{out_i} are defined by valid arrival curves. From the rate and burst limits, it follows that the resulting arrival curves must be greater or equal to the tight arrival bound, which means they are still valid constraints to their flows.

This allows to obtain an end-to-end delay by adding the individual delay for ingress, tunnel and egress. The calculated difference in latency between assigning the best case (zero burstiness) and the worst case (aggregate burstiness) to any or all output arrival curves causes latency differences in the single digit microsecond range in the DM simulation. The approach was therefore considered an acceptable intermediate solution for the present case.

9.10.6 Summary

In this chapter, it has been shown that NC is applicable to the case study and how its peculiarities can be handled. Additionally, it has been shown that machine schedules, which control accelerator components in the FAIR case study, can be modelled as network flows. Changing flows can be expressed by using suprema of alternative arrival curves or recurring analyses with non-empty buffers.

The model has been further enhanced to show how the trinity of Program, Cycle based Bus and packet based Network of the SoC System can be modelled in NC. All sub-modules have been discussed in detail and service representations have been deduced. The findings were then combined to produce a single equivalent service, which can be used to calculate the maximum delay for a particular flow of interest or the sum of all flows.

In the evaluation in chapter 10, the results obtained by simulating the model in the Disco DNC v2 simulator [128] will be represented and the results, as far as feasibly possible, compared with tests of the prototype system.

Part IV

Conclusion

Chapter 10

Evaluation

10.1 Overview

This chapter aims to evaluate all solutions from chapters 6-9 against the problems presented in the FAIR case study in chapter 4. The research methodology is outlined and the experimental setup described in detail. The results from simulations and experiments are presented as are current real world applications of the research. Finally, a conclusion from this work is drawn and future work is outlined.

10.2 Test and Verification

When undertaking the research and prototype development for this thesis, a suitable verification scheme was devised alongside it. The approach was split into several layers of tests, ranging from individual unit tests through simulation to tests on real hardware, all the way to the full CS stack necessary for control of a real accelerator.

10.2.1 Hardware Unit Tests

All hardware modules developed during the course of this work have been individually verified by simulations in the QuestaSim hardware simulator [134]. This was achieved via test benches that probed the module with semi-random stimuli and verified the desired behaviour. For simpler cases, test benches were constructed as pure VHDL. For more complex bus devices, the Coroutine Co-simulation Test Bench (Cocotb) framework was employed [135]. In the course of this work, a WB driver and monitor class for Cocotb was created to facilitate complex device regression tests, the resulting source code was contributed to the project [136].

10.2.2 Firmware Unit Tests

Whenever firmware performance needed to be verified, it was tested between two LM32 CPUs in the same SoC. One was running the firmware module to be tested, the other ran a monitor program, timestamping certain bus activity of the device under test. Other aspects of the firmware were tested in the hosting CPU itself by use of minor extra code, timestamping execution by means of either the CPU's own program counter register or by accessing WR time.

10.2.3 Full Test System

To test the interaction of all components in a real world scenario, the timing test facility (TTF) was constructed. While the author contributed test cases, the implementation is not part of this thesis. The TTF's purpose is to put the CS under development constantly through a series of tests to verify its functionality and accuracy. It was used to verify the DM's performance in the scope of this work. The DM is run on a PEXARIA V [137] platform, connected to an industrial PC host via PCIe.

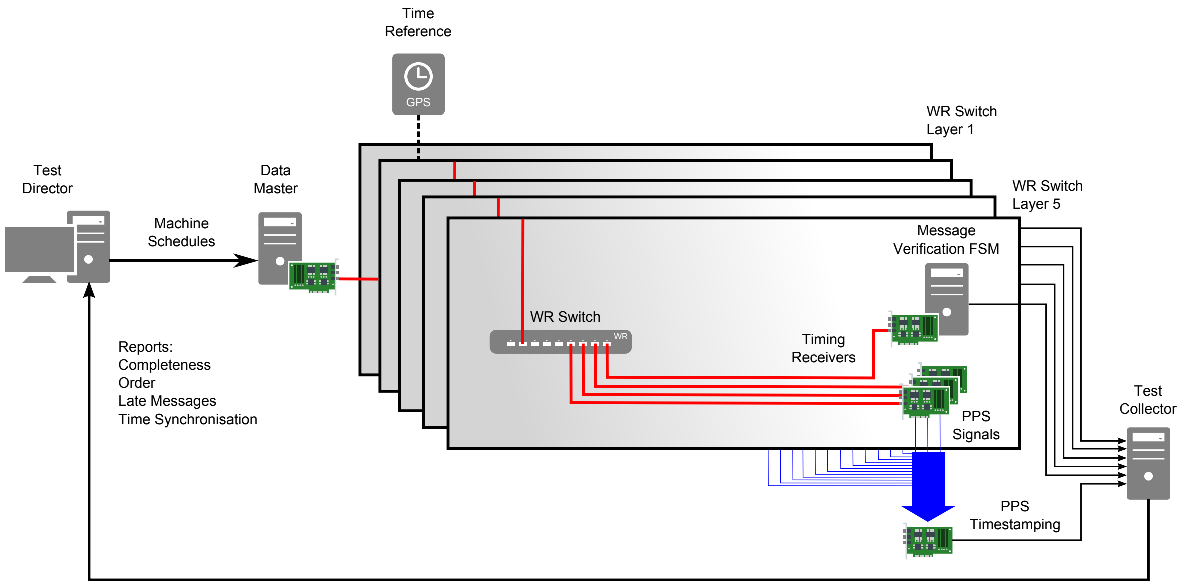


FIGURE 10.1: GSI Timing Test Facility,
Schematic of Verification System

Figure 10.1 shows the employed TTF setup. The system monitors the following cases simultaneously:

Interface Test The TTF accesses at least one endpoint per switch level via EB and retrieves various information. As the first step, it traverses the SDB records to list all available WB devices at this endpoint. Success confirms a working EB connection over the timing network.

Image Verification The TTF tries to access the build information ROM next, which contains information such as the date, the target platform, hashes of the most recent commits of the repository which went into this build, etc. The verification makes sure all endpoints run images of the expected version and intended for their platform.

Time Synchronisation Test Next, the TTF checks the status of the endpoint's WR core. If the core confirms that it is properly phase tracking the upstream switch and its local time is close to the expected current time, the result is valid. This method of synchronisation test is coarse and complemented by "Pulse Offset Monitoring" described later.

Message Sequence The DM is sending several message sequences generated from machine schedules and one for synchronisation testing. Endpoints are then programmed by the TTF to react to timing messages. If the endpoint has a host system, it is programmed to deliver message content to the host, which runs a verification FSM. These are basically a downgraded software DM, constructing expected traffic from machine schedules offline. Reordering at the DM or during transmission is of no consequence, as the ECA sorts arrivals by their deadline. Timing messages are then checked by the FSM against the expected content. If a message is missing, or any of its fields (such as ID, execution time, etc.) do not match the expected value, an error is logged.

Message Timeliness In the CS, messages must not be delayed, otherwise unforeseeable consequences may arise. The ECA therefore verifies message timeliness. If a message arrival time is later than its execution time minus ECA's processing time, it is flagged "late" and an error is logged.

If messages arrive much too early, ECA's input buffers run the risk of overflowing. To prevent this, a limit has been introduced to the time lead a message may have. If any message arrives more than four seconds early, it is removed from the buffer and an error is logged.

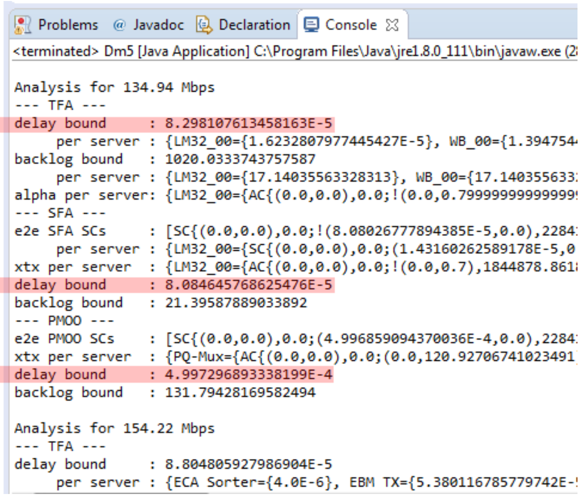
The ECA's design presents one major problem though. While ECA does have logging capabilities for late messages, it does not log the difference between message arrival and its execution time. The ECA is the single largest component in the whole SoC system (see area in figure 7.1) and at one action per clock cycle and 1 ns output granularity, extensively optimised for high performance. For this reason, including logging capabilities for mean, minimum and maximum difference have been postponed. It is, however, definitely a sensible feature to have in future.

Pulse Offset Monitoring As mentioned under "Message Sequence", the DM also generates a message stream for synchronisation testing, the PPS. It generates one message every second, its execution time always lying at the very moment the WR sub-second counter resets to zero.

Each endpoint is programmed to emit a pulse at one of its IO ports upon reception of this message. The outputs are connected to endpoints of the ExploderV [138] form factor, which feature 16 digital inputs each. They are used as a TDC and tag all incoming pulses with a timestamp. The Exploders run "golden" images, which are thoroughly tested for several months and can be viewed as reliable. TTF collects these timestamps via EB and uses them to monitor time synchronisation accuracy over all switch levels and endpoints.

10.2.4 Network Delay Simulations

All delay calculations have been conducted with the DiscoDNC v2.2.6 simulator framework [128]. DiscoDNC is a Java framework for deterministic NC, which accepts network and flow descriptions in form of piecewise linear service and arrival curves. It then performs TFA, SFA and PMOO analyses to obtain delay, backlog and arrival bounds.


Problems @ Javadoc Declaration Console

```
<terminated> Dm5 [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (2

Analysis for 134.94 Mbps
--- TFA ---
delay bound       : 8.298107613458163E-5
      per server : {LM32_00={1.6232807977445427E-5}, WB_00={1.394754
backlog bound     : 1020.0333743757587
      per server : {LM32_00={17.14035563328313}, WB_00={17.140355633
alpha per server: {LM32_00={AC{(0.0,0.0),0.0;!(0.0,0.79999999999999
--- SFA ---
e2e SFA SCs       : [SC{(0.0,0.0),0.0;!(8.08026777894385E-5,0.0),2284
      per server : {LM32_00={SC{(0.0,0.0),0.0;(1.43160262589178E-5,0
xtx per server    : {LM32_00={AC{(0.0,0.0),0.0;!(0.0,0.7),1844878.861
delay bound       : 8.084645768625476E-5
backlog bound     : 21.39587889033892
--- PMOO ---
e2e PMOO SCs      : [SC{(0.0,0.0),0.0;(4.996859094370036E-4,0.0),2284
xtx per server    : {PQ-Mux={AC{(0.0,0.0),0.0;(0.0,120.9270674102349
delay bound       : 4.997296893338199E-4
backlog bound     : 131.79428169582494

Analysis for 154.22 Mbps
--- TFA ---
delay bound       : 8.804805927986904E-5
      per server : {ECA Sorter={4.0E-6}, EBM TX={5.380116785779742E-
```

FIGURE 10.2: Screenshot of Delay Analysis with DiscoDNC

Multiplexing Disciplines An important factor when constructing NC simulation models is the multiplexing discipline of each server. DiscoDNC only supports *FIFO* and *Arbitrary* policy, i.e. serving packets in the order of their arrival or an unknown order. PMOO is only proven for arbitrary multiplexing, i.e., no knowledge about the used policy. This will always lead to correct results. However, the results are always more pessimistic than using explicit knowledge about policy.

The difference between arbitrary and FIFO policy lies in the calculation of the leftover service curve. For the purpose of this explanation, a simple rate-latency curve is assumed (see 9.5, page 144). When the multiplexing policy is FIFO, the leftover service curve has two summands contributing to its latency: the constant latency of the server's minimum service curve and the additional latency caused by processing the backlog of the cross traffic. In arbitrary multiplexing, however, there is an additional summand. Because there is no order, additional data from cross traffic can overtake the flow of interest, increasing the server's backlog in the meantime. The latency therefore consists of the server's constant latency and the processing time for the initial and runtime backlog.

EDF Most of the timing system follows a FIFO discipline, however. To obtain the tightest bound possible, the network was configured differently for each analysis strategy. The EDF schedulers in the LM32s and the PQ are always set to an arbitrary policy, the rest of the network is set to FIFO for SFA and TFA.

Fixed Priority The WR network uses non-preemptive, fixed priority policy with WR traffic being low priority, DM traffic being HP. This is a special case. The LP traffic is technically arbitrary, but is of no interest to the analysis. HP traffic, however, does not have to share bandwidth with LP. This allows a simplification when modelling fixed high priority flows in DiscoDNC: The LP traffic flows are removed and represented by their worst case effect on HP. This is the wait time (non-preemptive) for completion of a maximum length LP packet, which is applied as fixed latency to the HP flow aggregate.

Arrival Curve Generation DiscoDNC does not work directly with flows, only with the arrival curves describing them. When analysing the DM's behaviour, this was yet another reason for developing a scheme to obtain affine arrival curves directly from flows (machine schedules). The basic concept is described in chapter 9.3.2 on page 151.

10.3 Etherbone Analysis

This section relies in large part on the evaluation from the 2012 journal article on the EB protocol [93]. Its performance in terms of overhead and latency is analysed and compared against other low level bus protocols.

10.3.1 Overhead Details

EB Overhead in EB contains the network header, meaning all protocol headers below EB (802.3 GbE, IP, UDP). Inside EB, overhead consists of EB record headers, base write addresses, read back addresses, and read addresses. The amount of overhead in an EB packet depends on the type, the order of WB bus operations, and their addressing. EB supports block and random access operations in a single network packet. Minimum packet overhead in the current implementation with 802.3 Ethernet, IPV4, UDP, and EB is $18 + 20 + 8 + 8 + 8 = 62$ Bytes. Efficiency is calculated as payload over packet data (payload + overhead). Assuming a maximum packet size of 1500 Bytes, the best case efficiency for EB is in block write operations $1432/(66 + 1432) \cdot 100 = 95.6\%$, the worst case is random access writes $(480/(1010 + 480) \cdot 100 = 32.2\%$. Block reads do not differ in format from random access reads, all read addresses must be provided. This takes up a great deal more bandwidth than block writes. Because of streaming, however, block read operations have no impact on latency.

RDMA In the case of RDMA, packet overhead is somewhat bigger in general. Let us have a closer look at sample read request and write operations of captured iWarp RDMA packets. With 802.3 Ethernet, IP, TCP, iWarp, DDP, and RDMA, it has more protocol layers than EB. The overhead of all the packet headers add up to $18 + 20 + 32 + 6 + 14 + 1 = 91$ Bytes [95].

PCIe PCI Express is the most lightweight of the three protocols in terms of overhead. Physical layer, DLLP, and TLP add up to a minimum overhead of $2 + 6 + 16 = 24$ Bytes [100]. The small footprint is mostly due to the lack of routing capabilities, though PCIe still carries many legacy features meant to keep compatibility with PCI.

10.3.2 Overhead Comparison

It becomes clear that EB fills a specialist role for control applications when it comes to bandwidth efficiency. For block writes of 256 B EB can compete well with the other protocols. When it comes to block reads, EB is the least efficient of the three, because all addresses are contained inside the packet. However, due to the streaming functionality, this does not have an impact on latency. The more random access operations are present, the better EB fares in comparison. When handling many small bus operations, EB beats RDMA as well as PCIe, because it can transport all of them in the same network frame. Infiniband RDMA is of course vastly superior to EB in terms of bandwidth, as is PCIe, if more lanes were added. The outcome fits well with EB's intended role as a slim protocol for control applications. For FAIR's planned CS, the high efficiency for block writes is also useful for the occasional distribution of set values.

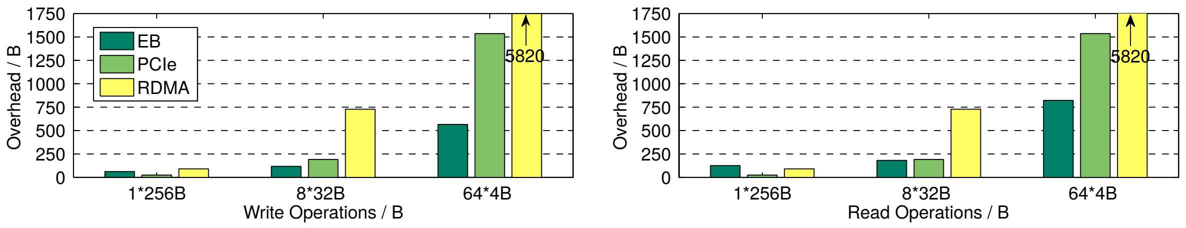


FIGURE 10.3: Overhead of EB, PCIe, and RDMA when transmitting 256 Bytes read and write operations, 32 Bit word width. EtherBone distinctly excels in random access operations and block writes [93]

10.3.3 Latency

EB was designed to introduce very low latency. In this implementation, the WB data rate is 4 times that of the GbE network, therefore all processing can be handled while gathering the next data word. Ideally, EB therefore does not cause any additional latency. The delay introduced when processing or answering an EB packet equals the time it takes to receive and process the packet headers, determining if a valid EB packet was received and if a reply was requested. However, there is a threshold to the latency caused by targeted WB slaves above which delays will accumulate. While pipelined operations are supported, a bus cycle is not complete until all acknowledgements and data blocks have been received by the WB master. The EB core has to wait for all answers at the end of a WB cycle. If the maximum latency of the ACKs on the WB bus is never greater than 3 (4 cycles per word on IF side, 1 cycle on EB/WB side) cycles, EB is guaranteed not to need pre-buffering. Since the TX header is complete in the buffer by the time the first WB operations take place, there is still a reserve of 42 cycles if individual operations should take longer to complete. The measurements for EB were taken using the WR time stamping unit, normally used for a special version of precision time protocol. Values for all other systems were taken from literature. We define the round-trip latency as first byte sent to the first byte of the reply arriving at the sender, connected end to end, in our case over a 10 m fibre cable. This shows EB to work well for short message latency with a

1.2 µs round-trip time. The prototype hardware implementation still encompassed packet buffers that could not be removed in this version due to their importance for the WR timing framework. Without this buffering, simulations show EB latency to be around 550 ns for our setup and will also be independent of message size. This result fits in well with the 1 µs estimates made by Rumble et al. in 2011 [139].

Direct quantitative comparison with other systems is difficult when the latency measurements have to be taken from third parties. In most papers, the test beds are not completely described and the understanding of latency is not exactly defined. For example, when talking about round-trip latency, it is very important at which OSI layer the turnaround takes place. On top of this, the underlying hardware differs widely. Bearing this in mind, we will stick to a qualitative view on latency. The lowest figures found for RDMA over Infiniband come from a performance benchmark found online [96] and is stated as < 1 µs. Since it is given as an end to end value, it would have to be doubled to 2 µs. The results for PCIe [100] were taken from a comparison to the hypertransport protocol. The read latency of an 8 B packet came down to 232 ns from first byte sent to the first byte of the reply packet. Considering the PCIe bandwidth of 2.5 Gbit/s and the bus controller running at 333 MHz, EB did not fare badly in comparison. Furthermore, there is still room for improvement in EB down to 550 ns when the WR store-and-forward buffers are removed. In the context of these figures, EB results can be said as being on the edge of the technically feasible and being a good fit for a timing/CS application.

10.4 Network Calculus Simulation Models

The DiscoDNC models to be used were constructed from the model in chapter 9. There were a total of four different models built. All numerical values used in their simulation, such as the rates and latencies for servers, can be found in the tables C.2, C.3, C.3 in appendix C from page 221 onward.

FAIR CS Model The first FAIR model was built to show the delay bound for the final FAIR implementation over net command bandwidth. Figure 10.5 shows the delay bound in milliseconds for different analysis types over the bandwidth of timing messages (without any overhead) in megabits per second. The different series of analyses – TFA, SFA and PMOO – are calculated for different sets of flows with identical arrival curves. The identifiers in the legend show the number of flows in the first digit and the number of CPUs each generating this number of flows in the second digit after the x. The simulation is made to show the maximum delay value occurring within a series. Because the resulting delay values for identical arrival curves are also identical, this is of no consequence. However, when one considers the implication for different flows, it becomes clear that the basic situation is the same. Only the maximum delay is of interest when determining if a set of commands can be delivered on time.

FAIR CS Tunnel Influence The second model, represented by the graph in figure 10.6, shows the same setup, but without the network tunnel between the EB TX module in the DM and the EB RX module in the endpoint. This effectively allows multiplexing/reordering between individual timing messages. While this simulation is not an accurate model of the real world system, removing the influence of the tunneling property and observing the effects on latency was helpful in understanding the behaviour of the FAIR CS model.

GSI TTF Model The third model for the current TTF test system at GSI in order to compare values with a real world test system. Because to this day (November 2016) there is no FEC encoder or decoder prototype available for the test system, these are not included in this simulation and therefore do not increase traffic, loosening the GbE bottleneck. Furthermore, the WR switches do not yet show the desired behaviour of a fixed policy of high priority command traffic neither do the NIC modules in the DM and endpoint [140]. In addition, packet losses were observed when WR switches were subjected to an input bandwidth much greater than $\approx 100\,\text{Mbit/s}$. In order to produce a model compatible to the TTF behaviour, WR switches were modelled as FIFO nodes receiving IEEE 1588 WR PTP, ARP, DHCP and SNMP cross traffic of equal priority as the DM. The bandwidth of the cross traffic by type was estimated from packet dumps in the WR network, see table C.6. No packet loss was simulated.

The last difference to the FAIR CS model lies in the DM firmware. The current version cannot yet reach the $5\,\mu\text{s}$ interval planned for the FAIR implementation, as it is yet not fully optimised. It lacks a hardware optimisation reducing RAM access from 4 to 1 cycles, the removal of several function calls and a manual assignment of general purpose registers to variables of the scheduler routine, to avoid stack use. The measured value for the current implementation is $7.75\,\mu\text{s}$. All values changed versus the FAIR model can be found in table C.4.

GSI Minimal Test System The fourth model, as shown in figure 10.8, depicts a minimal test system (MTS) for testing close to the maximum total bandwidth. The purpose of the simulation is a comparison between measured and simulated values. Contrary to the GSI TTF model, the MTS consists solely of two PEXARIA V boards [137], one configured as a DM, the other as an endpoint. They are directly connected, without a network of WR switches in between.

10.5 Observed Simulation Results

10.5.1 FAIR CS Model

The bandwidth limit at 192 Mbit/s is imposed by the bottleneck at the WR network, where the lowest available bandwidth in the system coincides with the highest overhead. When crossing this threshold, all delay values jump to positive infinity (∞^+ values are not drawn in the graph). This latency is to be expected, any input flow bigger than the system's maximum throughput must eventually lead to infinite wait times. The same holds true for the second bottleneck in the system, the maximum message rate per CPU. If the threshold is crossed in a CPU, all flows originating at that CPU will experience infinite wait times.

SFA vs PMOO Performance A remarkable aspect of figure 10.5 is the difference in performance between SFA and PMOO. The common notion used to be PMOO delivering the tightest bounds, because it saves on multiplexing cost. This has been refuted in recent years; there are circumstances when SFA can indeed outperform PMOO [127].

The FAIR timing network is very similar to the example given by Schmitt et al. [127]. There is a tight bottleneck right in the beginning (CPU message rate), followed by servers with higher service rates. We shall use the minimal case of two nodes with two flows from figure 9.8 again as an example.

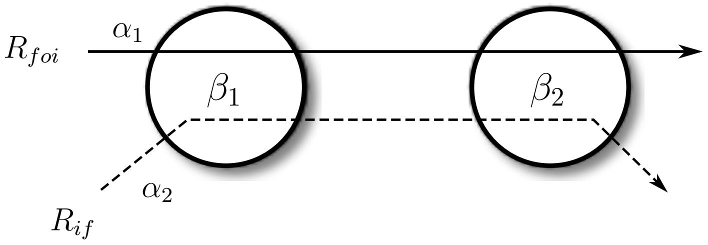


FIGURE 9.8: Minimal Network Example: 2 Nodes, 2 Flows (p. 149)

The input flows α_i are given as leaky-bucket arrival curves of the form γ_{r_i,b_i}, the servers β_i as rate latency nodes of the form β_{R_i,T_i}. Simply speaking, when concatenating servers by convolution of their service curves, the minimum of both rates is used and the latencies are added. Arbitrary multiplexing is assumed, and for the purpose of the explanation, only the case of $b_2 = 0, T_1 = 0$ is inspected. The delay bounds for the minimal system resolve to:

$$d^{SFA} = \frac{T_2 + b_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{R2 - r2} \tag{10.1}$$

$$d^{PMOO} = \frac{T_2 + b_1}{(R_1 \wedge R_2) - r_2} + \frac{r_2 T_2}{(R_1 \wedge R_2) - r_2} \tag{10.2}$$

By comparison of the right fractions in both equations, it is easy to see that the advantage of SFA over PMOO can be made arbitrarily large by increasing R_2, the rate of the downstream server.

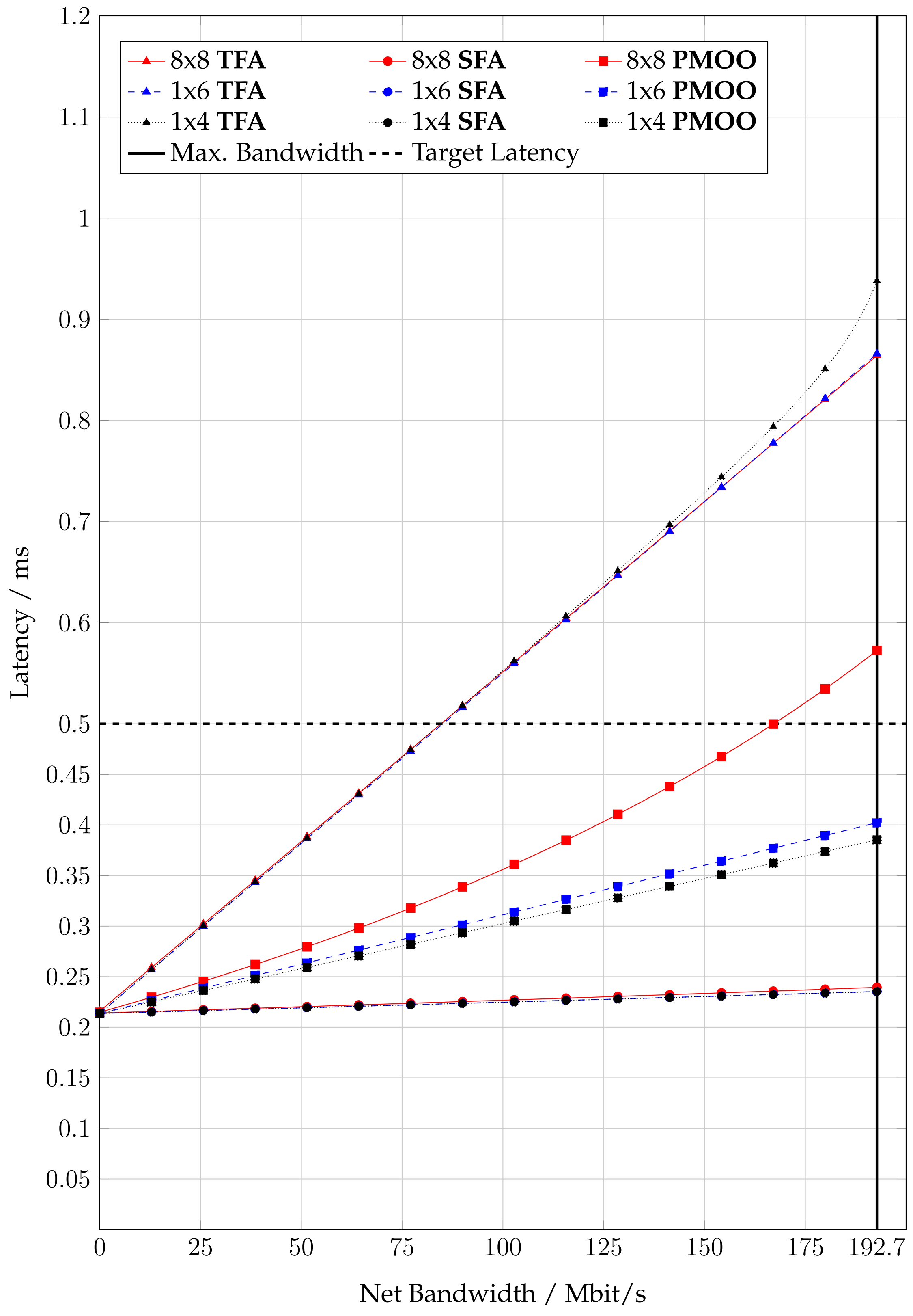


FIGURE 10.5: FAIR CS Simulation,
command latency bound over bandwidth. CPU x Threads equals the number of flows per type of analysis (marker shape). Infinum of curves of equal colours shows the corresponding latency bound

The reason for this effect lies in the loss of information about topographical service distribution when forming a single equivalent node before calculating leftover service. The bottleneck becomes dominant, which leads to an overly pessimistic representation of all downstream servers in the present case.

Message Delay The SFA analyses in the bottom of figure 10.5 (circular markers) show a maximum delay bound of $\approx 240\,\mu s$. With PMOO lying above $380\,\mu s$, it is clear that SFA outperformed PMOO in this scenario. Figure 10.5 also clearly confirms that it is theoretically possible to deliver timing messages at the maximum bandwidth of $192\,\text{Mbit/s}$ within the desired target latency of $500\,\mu s$. Considering table C.3, the infinum of all series must lie above $218\,\mu s$, as this is the sum of all static latency of the servers along the path. This means it occurs independently of traffic rate, burst or multiplexing cost.

10.5.2 FAIR CS Tunnel Influence

The model shown in figure 10.6 on page 196 was built solely for the purpose of demonstrating the effect of the added multiplexing cost. The strongly increased cross traffic created by the other timing message flows would have a heavy impact on the total latency compared to the tunnelled model, reducing the usable bandwidth within the desired delay budget of $500\,\mu s$ to $\approx 50\,\text{Mbit/s}$. It is noteworthy that in this case PMOO analysis does outperform SFA again, as SFA pays the extra delay caused by the many multiplexed flows at every node that used to be tunnelled.

10.5.3 GSI TTF Model

Message Delay It is interesting to see PMOO excelling in this scenario. The recurring multiplexing with WR cross traffic is obviously costly, putting TFA and SFA at more than $300\,\mu s$ off compared to to PMOO with a start at $\approx 140\,\mu s$. The 8x8 PMOO analysis is also visibly deviating from a linear course, as the greater CPU message interval decreased rate R_1 in the minimal example given on page 193. The 1x4 analyses all fall short and stop at $\approx 125\,\text{Mbit/s}$. This is to be expected, as the message rate of $32\,\text{B}/7.75\,\mu s = 33\,\text{Mbit/s}$ per CPU does not suffice to saturate the interface with only four CPUs. However, the 1x4 series has been included for comparison with the FAIR model and the yet to be presented minimal model.

The results in figure 10.7 on page 197 show PMOO delay bounds at $100\,\text{Mbit/s}$ are at $250\,\mu s$. This matches the observations on the real TTF system described in 10.6.2 where test runs with a lead less than $300\,\mu s$ indeed occasionally returned late messages.

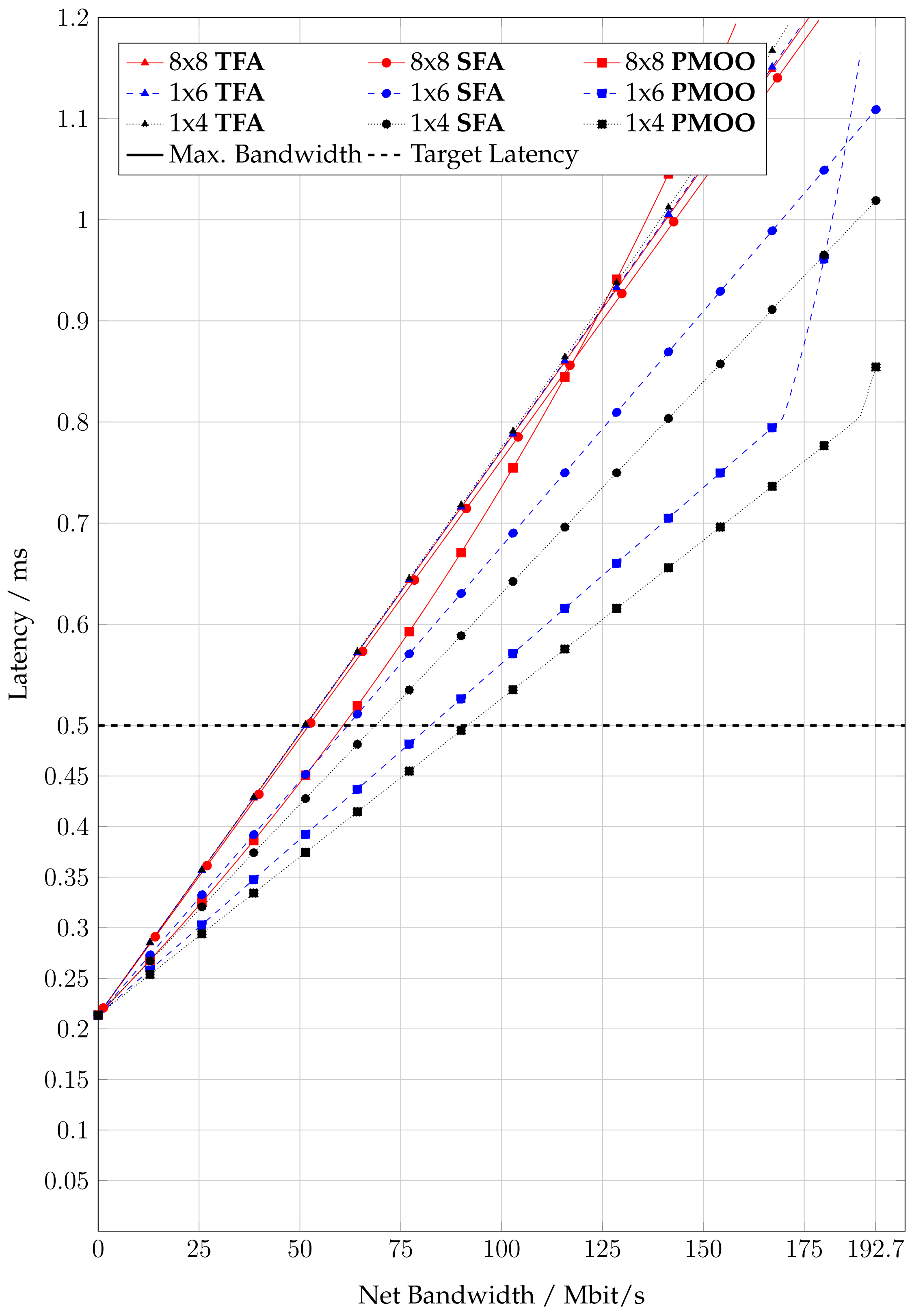


FIGURE 10.6: FAIR CS Tunnel Influence Simulation, command latency bound over bandwidth. Removal of the tunneling effect on messages leads to a strong latency increase compared to 10.5

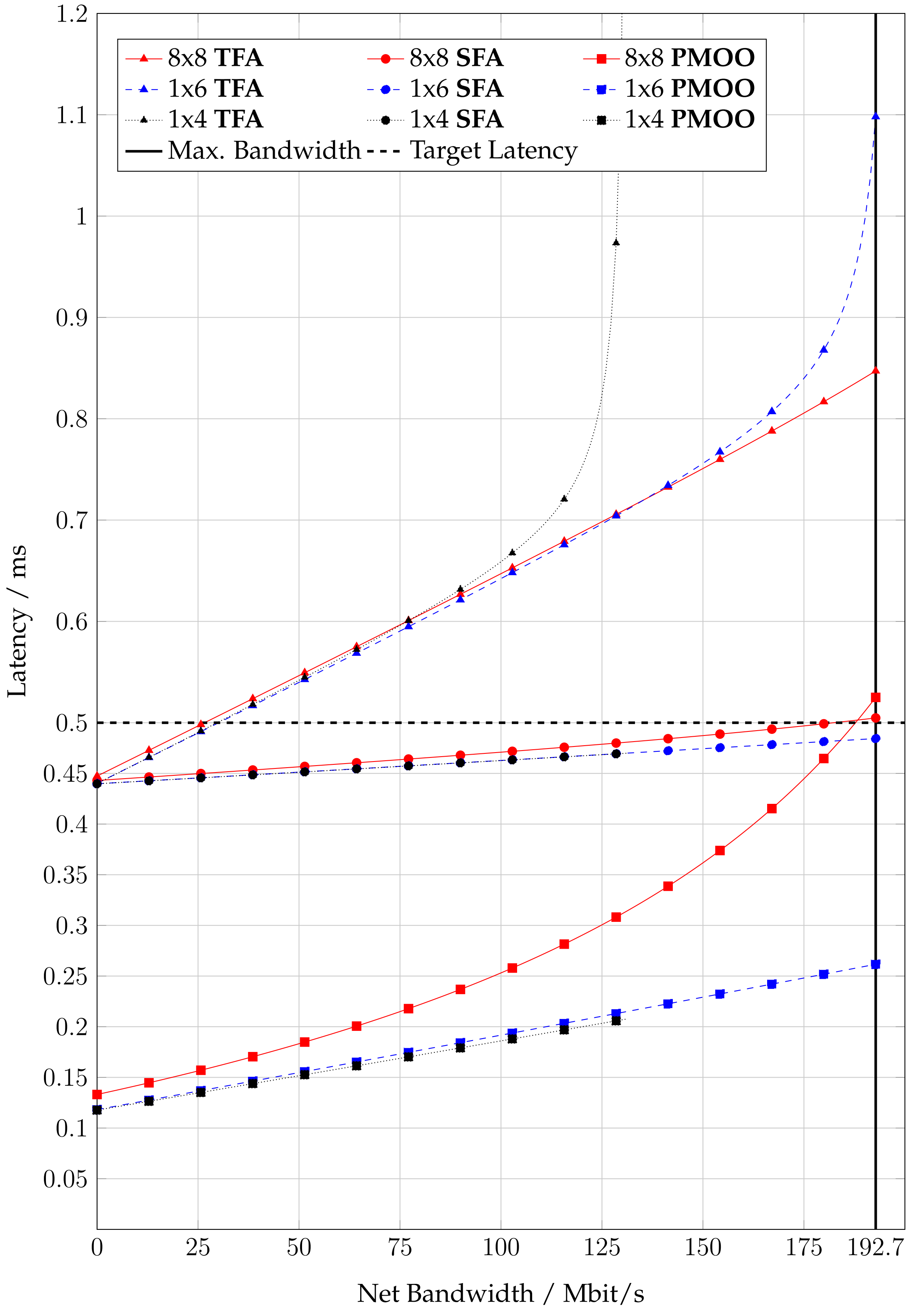


FIGURE 10.7: TTF Simulation, command latency bound over bandwidth. 64 Flow (8 x 8) PMOO and SFA analyses (red squares and circles) show the goal of a 500 µs latency bound can be guaranteed up to 187 Mbit/s. This more than satisfies the goal of 100 Mbit/s net bandwidth (see 4.6)

10.5.4 GSI MTS Model

Message Delay This case has at first glance the most unusual results of the four models, as the resulting curves for different analyses actually cross each other. Still, this is not an unexpected outcome considering the SFA-PMOO situation dependent performance. A delay bound can be obtained from the infinum of all analysis types for a given flow configuration. This results in an upper bound of $\approx 270\,\mu s$. The main purpose of this model was a comparison to real world hardware, which is undertaken in 10.6.3. The actual values used are shown in table C.5.

The infinum of delay bound curves for 64 flows at $100\,\text{Mbit/s}$ were compared with the latency budgets used for the MTS experiment documented in table C.5, shown in the graph as cyan markers. The values at the markers show the ratio of late messages to total messages measured at the corresponding latency budget (marker's position on the y-axis). All budgets above the latency bound infinum produce a rate of 0, which indicates that the area above indeed represents a "safe" zone inside which all messages arrive on time. This confirms by both NC simulation and experiment that the goal of a $500\,\mu s$ latency was achieved. The ratio approaching 1 at $150\,\mu s$ means that virtually all messages will be late if the budget is chosen lower than the static delay all packets experience regardless of traffic when traversing the system.

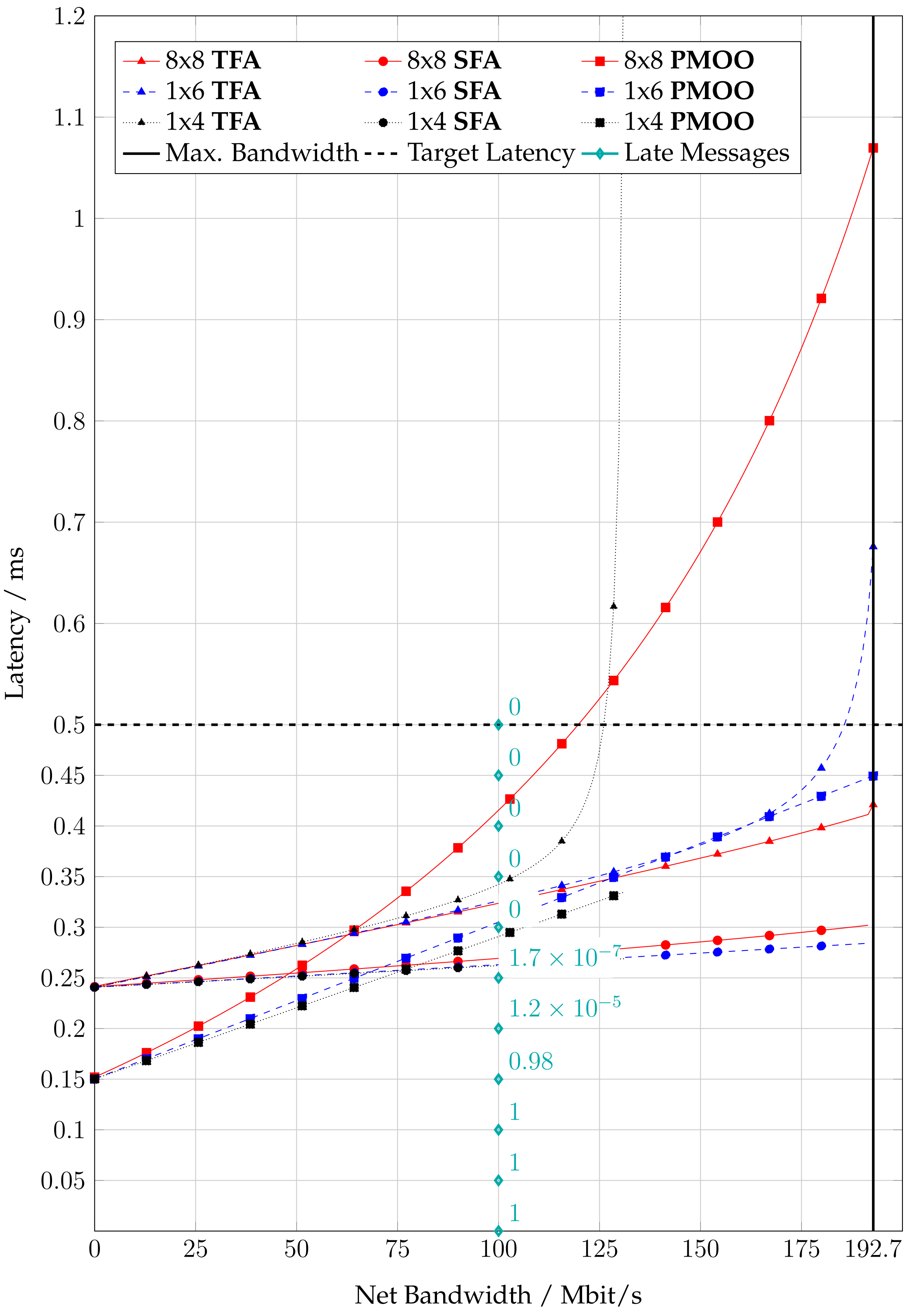


FIGURE 10.8: MTS Simulation, command latency over bandwidth. Real MTS's ratio of late messages to messages at 100 Mbit/s (table 10.3) is shown in cyan. Position on the y-axis corresponds to latency budget, the number to the observed ratio. Ratios of 0 show that budgets above the curves' infinum ensured timely delivery

10.6 Full Test System Results

10.6.1 Evolution of Data Master Modules

Version	**1**	**2**	**3**	**4**	**5**
Timing Granularity	8 ns	8 ns	1 ns	1 ns	1 ns
CPU Cores	1	4	8	8	9
Threads per Core	1	1	1	2	8
Thread Scheduler	–	–	RR	EDF	EDF
Schedule Sync.	–	SM	MSI	MSI	MSI/Ext
Priority Queue	SW	Heap	ST	ST	ST
Etherbone Master	SW	HW	HW	HW	HW
Memory	–	2 pages	2 pages	dynamic	dynamic
Fixed Exec. Time	–	–	–	–	yes
Native Schedules	–	–	–	–	yes
CPU Load Analysis	–	–	–	yes	yes
NW Load Analysis	–	–	–	yes	yes

TABLE 10.1: Data Master Features

Both EB and the DM itself have undergone various changes over the last five years. Table 10.1 lists the most important features and architectural changes. The scheduler has been changed from Round Robin (RR) to EDF, the synchronisation concept changed from a shared memory for all CPUs to MSI mailboxes and recently the CMD-Q system described in chapter 8. The first PQ implementation was software based, this design changed from a dual port memory based heap to a sorting tree in the current implementation. Memory management changed from direct access over a shadow page approach to a fully dynamic concept managed in the DM's host system.

Version	**1**	**2**	**3**	**4**	**5**
Unit Tests	yes	yes	yes	yes	yes
Minimal Test System	yes	yes	yes	yes	–
Int. Test System	yes	yes	yes	–	–
Delay Simulation	–	–	yes	yes	yes

TABLE 10.2: Data Master Testbeds

As table 10.2 shows, the interplay of system components could not fully be tested yet. Because of the timeframe of this thesis, certain features such as dynamic memory management and multi-thread operation were not fully implemented at the time and could only be tested in a minimal test setup yet. While this test is not significant on an absolute scale, it provides a relatively solid qualitative assessment of the concepts. However, version 3 of the DM and CS has undergone and passed the full test stack.

10.6.2 GSI Timing Test Facility

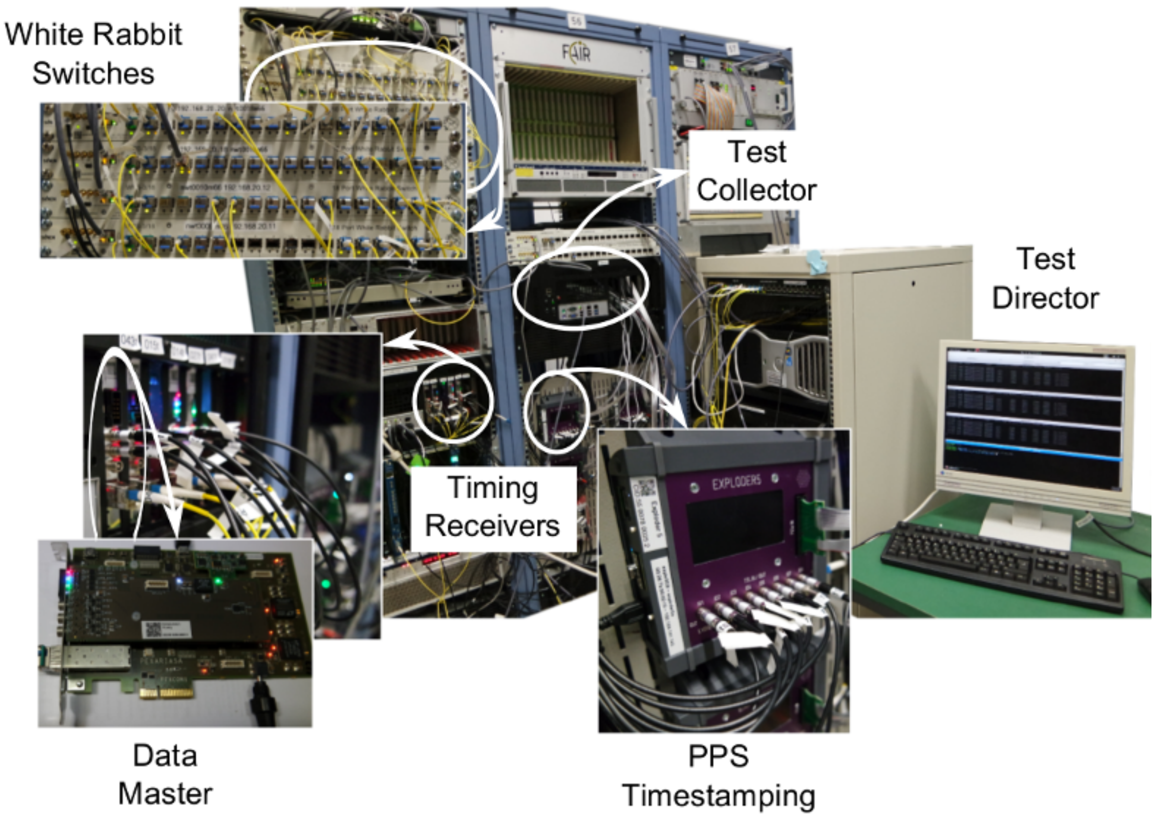


FIGURE 10.9: GSI Timing Test Facility,
Photos of actual Verification System

The TTF has been run for months in continuous operation. Figure 10.9 shows the actual hardware setup used, a more detailed description can be gained from section 10.2.3 and schematic 10.1 on page 186.

The TTF was running four cores at up to half capacity of randomly generated schedules, which were verified in the endpoints. This entailed several one week long tests, each producing a message volume between 1×10^7 to 5×10^{10} messages. The observed results all complied with the simulation results presented in figure 10.7: All tests set up with a message lead of $\geq 400\,\mu s$ never showed any messages flagged as late. Values below $300\,\mu s$ occasionally showed test results containing spurious late messages. Values below $250\,\mu s$ produced late messages in an apparently random pattern, any values $\leq 150\,\mu s$ lead to a continuous stream of late messages in all test cases and occasional message loss. Cutting the message lead short in such an extreme manner leads to accumulation at the DM. All other monitored conditions, such as functional EB connections, completeness of the received command schedules and time synchronisation accuracy were behaving according to specifications in all cases. The measured accuracy of the WR system was verified manually via an oscilloscope. The results were consistently satisfying the requirements of the case study with a jitter $\leq 15\,ps$ and a mean difference to the GPS time reference of $\leq$

180 ps. The measured timestamps were in almost all most cases within the desired $\pm\ 0.5$ ns of the mean. Deviating endpoints could in almost all cases be traced to reproducible bugs that have been fixed. The important aspect of this measurement is not guaranteeing perfect synchronisation though, but detection of the condition to rule out wrongful attribution of late/lost message errors to the DM prototype.

A different test run with CPU host systems channels as a sink to the ECA [141]. While the ECA software output channel is slow and cannot keep up with more ≈ 1 Mbit/s, the test showed that a net rate of 50 Mbit/s produced neither late nor lost messages.

10.6.3 Minimal Test System

Message Lead / µs	500	450	400	350	300	250	200	150	100	50
Late Msg Ratio	0	0	0	0	0	1.7e-7	1.2e-5	0.98	1	1

TABLE 10.3: Late Messages Ratio at 100 Mbit/s Net Bandwidth in the Minimal Test System

A small scale test system of two directly connected boards was employed to test a setup at a net bandwidth of 100 Mbit/s (≈ 500 Mbit/s total), which is not yet reliable using WR switches. The minimal test system consists solely of two PEXARIA V boards [137], one configured as a DM, the other as an endpoint. They are directly connected via fibre, without a network of WR switches in between. The purpose is to enable high rate tests by removing the influence of yet unreliable switch policy and lossy transmission [140]. The effect of the unavailable FEC core is mimicked by quadrupling each message when sending.

When testing the setup at a total bandwidth close to full GbE line speed, WR did lose time synchronisation on several occasions. Since WR, like PTP, does not depend on transmission time, this behaviour can most probably be attributed to a race condition in the WR firmware. The system was therefore configured to run 6 CPUs with quadrupled output at maximum of 64 Mbit/s each, achieving the required net bandwidth rate of the final FAIR system of 100 Mbit/s. Each test run was set up to generate a total of $5 \cdot 10^{10}$ messages over the course of one day. No messages were lost in the experiments. The number of late messages detected by the ECA, as shown in table 10.3 follows the simulation results from figure 10.8. The measured values are marked and labelled in cyan, a white overlay was added on top the other plots to retain legibility. The plot clearly shows that no late messages occurred above the infinum of all delay plots at ≈ 250 µs.

Chapter 11

Conclusion

11.1 Overview

This chapter aims to evaluate all solutions from chapters 6-9 against the problems presented in the FAIR case study in chapter 4. The research methodology is outlined and the experimental setup described in detail. The results from simulations and experiments are presented as are current real world applications of the research. Finally, a conclusion from this work is drawn and future work is outlined.

11.2 Experimental Results

11.2.1 Etherbone

Performance The desired small protocol footprint has been achieved. The measured round-trip latency at 1.2 µs was above the calculated values, but the difference stemmed from packet buffering in this prototype, which is indispensable for the operation of the WR clock synchronisation cores in the present version. Without the buffering time, the latency will be even lower. This performance proves EB to be a worthwhile protocol for use in future timing and CSs and hopefully further applications in the large physics community will be seen in the near future.

Integration The interplay between EB HDL macro-cores and software API has been successfully shown at the 6th WR Timing Workshop in 2012 [142]. Faithful transmission of bus operations and timely message delivery were demonstrated, as well as scanning the device chain on a remote WB bus. Between 2013 and 2016, a complete EB software framework was developed. There now exists a complete API for EB discovery and communication, providing functions for finding and listing SDB devices as well as bi-directional access to the remote WB bus. There also now exist Linux Kernel drivers for PCIe, USB and UDP network connections. TCP is also available as a convenience feature for PCIe targets, although not natively. Instead it is sent to the target's host system (Linux computer), which deals with the TCP wrapper and talks EB over PCIe with the target. Pure software slaves, capable of running in Linux environments were also designed, as well as a pure hardware master for high speed EB communication.

The most common needs for interaction with an EBS system were identified and a whole suite of tools has been developed in order to address those needs. This covers finding EBS on networks, listing their internal WB bus structure, finding SDB devices on the bus and direct read and write access to bus locations. A flash memory programmer and CPU firmware loader for EB were added along with some diagnostic tools. The latter consists of a packet dissector for the Wireshark Network Analyser and a snooping tool for EB connections on Linux computers. The EB tool suite is completed by a reader for gate- and firmware build information, a serial console and a tunnel application for redirecting EB connections. See appendix B.3.1 for details.

EB for Accelerator Timing The main purpose of the planned CS is to provide two basic functions. The first is time tagging events, the second is sending commands that will trigger pre-programmed actions at a given time. EB can be used broadcast or unicast mode, the former will be used for commands. Because all WB devices in the target SoC systems are memory mapped, EB can be employed for all requirements, reading timestamps as well as delivering commands and pre-programming the necessary command filter rules into receivers. The difference between a command and any other EB content lies only in the targeted WB slave and data content, not in format. Commands go to a FIFO input register in the ECA unit, which controls the end point's IO cores. Programming filter rules is achieved by writing to corresponding control registers of the ECA, Timestamps can be read from the TLU.

11.2.2 DataMaster Platform

11.2.3 Network Calculus Simulations

It has been shown that NC is applicable to the case study and how its peculiarities can be handled. Additionally, it has been shown that machine schedules, which control accelerator components in the FAIR case study, can be modelled as flows. Changing flows can be expressed by using suprema of alternative arrival curves or recurring analyses with non-empty buffers.

The model has been further enhanced to show how the trinity of program, cycle based bus and packet based network of the SoC system can be modelled in NC. All sub-modules have been discussed in detail and service representations have been deduced. The findings were then used to create a numerical simulation for DiscoDNC, which can be used to calculate the maximum delay for a particular flow of interest or the sum of all flows.

11.3 Real World Applications

There are already a number of real world applications employing WR based sub-nanosecond timing and CSs, proving the concept by controlling machines in a production environment. The most important examples for this thesis are the applications at FAIR, which both employ the DM and EB on top of WR to form a complete

CS. Further applications employing only WR and EB are listed under 11.3.3, “Application in other Projects”.

Etherbone in Control Systems Basic functionality for the GSI/FAIR’s and CERN’s timing system was shown in 2012 in the form of timestamping pulses from a signal generator and producing a pulse on the end points IO based on the read timestamp plus a delay.

In the beginning of 2013, design on the first version DM, the top unit of the FAIR accelerator, was started. It uses the hardware EB master implementation to create control messages. It was tested and found suitable for deterministically sending machine control messages to several timing receivers, triggering IO actions. This was first employed in March 2014 in the testing of the source of the FAIR proton linear accelerator, developed in Saclay, France.

In April 2015, the DM started to control the linear accelerator stage of the CRYRING accelerator, a small, independent synchrotron ring at FAIR. In this scenario, EB is now used to control approximately 50 different timing receivers with several different form factors. As of 2016, including lab and test setups, EB is now used over UDP, TCP, USB, PCIe, PCI, VME and SCU-Bus. EB has finally proven to be both suitable for deterministic CS use and as a flexible interface between the computer and the SoC world.

11.3.1 Proton Injector

In the course of 2014, the second DM implementation was assigned as a prototype controller to the ion source of the proton linear accelerator for FAIR currently under construction in Saclay, France. In 2015, the WR based CS with the DM at its top successfully controlled the microwave cavity and chopper [143].

11.3.2 CRYRING Accelerator

The CRYRING accelerator is a small synchrotron machine with its own linear injector stage (figure 11.1). It was used as a heavy ion storage ring at the Manne Siegbahn Laboratory at Stockholm University and was contributed to the FAIR project in 2014. CRYRING serves as a testbed for the FAIR CS. Tests operations with a WR based CS, receiving commands from the DM over the EB protocol, were commenced in the end of 2014 [8].

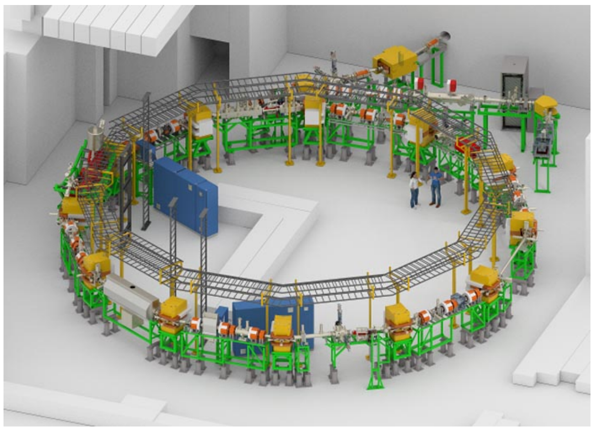


FIGURE 11.1: 3D Model of the CRYRING Synchrotron and Injector [144]

First Beam In June 2015, the first ion beam was produced in CRYRING's injection stage. This was the second successful application of the DM as a control unit for real machinery. This event was documented on the WR development mailing list:
"*...for the first time we have beam in the injector of CRYRING at GSI involving a White Rabbit based timing system ... We control the beam chopper of the CRYRING LINAC section ... and trigger ... beam instrumentation to detect the signal of the pulsed beam after the chopper ...*" [145].

First Turn The most recent successful test of the prototypes developed for this thesis was CRYRING's first turn, that is, the beam achieving more than one orbit in the synchrotron ring. This success is documented in a FAIR press release from August 2016, where it was noted: "*Successful beam transport from ESR to CRYRING ... Furthermore, new beam diagnostics and FAIR-like control hardware and software could be tested with real beam.*" [144]

11.3.3 Application in other Areas

Existing Applications Apart from the DM, which is currently only employed in the FAIR project, WR and EB implementations in soft- and hardware are already employed elsewhere. Timing and CSs with an accuracy in the sub-nanosecond domain are drawing an increased interest from various institutions. An application subject to a large amount of publicity was the deployment of WR between CERN and Grand Sasso Neutrino Detectors [146] to help answer the question if particles faster than the speed of light exist. This was the first field application of WR. The technology is also already employed in other large physics experiments, such as the Culham Centre for Fusion Energy, Oxfordshire, UK [147].

Apart from the field of particle accelerators, wired machine coordination at nanosecond level over several kilometres provides the greatest benefit to the telecommunication industry. Increased synchronisation accuracy of a multitude of receivers drastically increases the spatial resolution of distributed tracking systems, for example. It also enables higher bandwidths by exact synchronisation of sample clock phases in standard communication applications.

In particular the accurate synchronisation of phase antenna arrays has already provided several promising applications for such a WR based CS. There are several examples in the field of radio astronomy, such as the Large High Altitude Air Shower Observatory in the Sichuan province, China [148]. There are more than 20 institutions employing WR (usually with EB) and a complete list can be found at the OHWR Website [9].

Further possible Applications This research could be directly employed in other systems that will benefit from high precision, deterministic control over several kilometres. Examples would be the aforementioned phase antenna arrays in telecommunications, timing and synchronous control of mobile network base stations, distributed airspace monitoring and control, timestamping for stock-market applications or correlated data acquisition in industrial factory control systems. While the provided implementation does require specialised hardware, integration with standard computers was a major consideration in prototype development.

The PEXARRIA V boards used in the experimental validation of this research provide a simple way of integrating the high precision control system features of this approach with standard industrial PCs or servers. By installing one of the boards in a PCIe slot, the computer is fully equipped to act as timing receiver. Connecting these boards with WR switches over fibre then provides a fast, deterministic channel for control traffic. An interface to the OS (currently Unix/Linux based systems only) is provided in form of PCIe drivers for EB and a complete API framework called Simple API For Timing library (SAFTlib). SAFTlib comes with an XML based code generator, which supplies the necessary C++ code to neatly integrate timing devices into user-space programs.

11.4 Conclusion

The research undertaken and presented in this thesis has shown, both theoretically and by experimental validation, that sub-nanosecond accuracy time synchronisation can be utilised to create a deterministic, alarm-based CS. The underlying WR time synchronisation technology was analysed and the results showed both the need and the potential for a protocol to enable deterministic data exchange between FPGA based SoC systems. This need was satisfied with the creation of EB, the analysis of its properties in this chapter showed that it rightfully found a niche next to established low level hardware protocols such as RDMA or an attempt at routing PCIe over local area networks. Both the EB and WR protocols are now thoroughly tested and in productive use for more than four years on more than thousand devices around the world. The whole CS hardware-software stack is therefore, within known limits concerning WR switch hardware, validated by experiment.

The goals (scale in number of supported endpoints, scale in distance and timing accuracy) as stated in chapter 1.2, have been fulfilled. This has been shown either through direct validation by experiment or could be deduced from experimental results.

11.4.1 Number of Endpoints

While experimental setups were limited to 50 endpoints due to the availability of prototype timing receivers, the approach's scalability can be deduced from the small scale experiments and several system properties.

The network topology resembles a tree, with time critical control traffic being injected at the root. Due to the tree topology, the required bandwidth for downward traffic at each link port can either stay constant or decline, but cannot increase. The link at the root node therefore dictates available bandwidth . With command traffic being broadcasted to allow easy tapping control traffic and deployment of new control equipment, it follows that the available bandwidth B (100 Mbit/s) at each link is independent of m and n .

WR switches feature 18 ports. To meet the goal of 2000 endpoints 4.6, the minimum number of required levels is $m = 3$. The number of links would be sufficient to connect $(18-1)^3 = 4913$ devices. Considering the proposed redundant links except the last mile to the endpoints 4.5.2, $m = 4$ levels of switches are required to satisfy the goal, providing support for $(\lfloor(18-1) \div 2\rfloor^3 \cdot (18-1) = 8704$ devices. To be able to adapt the topology to geographical necessities (placement of switches only possible in radiation-free zones, maximum capacity of cable conduits, etc.), a 5th level was introduced. The system's theoretical capacity then increases to $\lfloor(18-1) \div 2\rfloor^4 \cdot (18-1) = 69632$ devices.

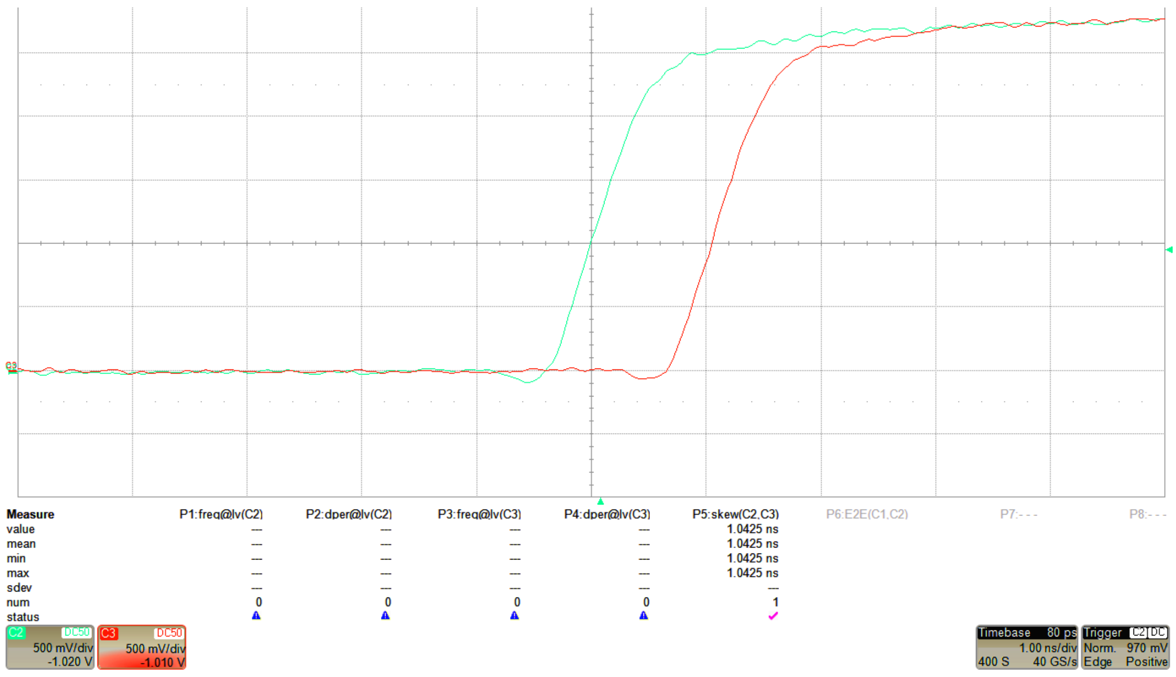
A conjecture has then been deduced from two experiments. First, a single WR switch was proven by experiment to correctly function as an interconnect between 1 uplink and 17 downlinks($m = 1, n = N = 17$) and to propagate timing and control traffic within the expected bandwidth and latency bounds 4.6. Second, 5 cascaded WR switches were proven by experiment to correctly function as an interconnect between 1 master uplink and 25 downlinks, 5 at each level in the hierarchy ($m = M = 5, n = 5$). Functionality was verified by once again measuring the WR timing accuracy (see 2.7.6) in order to verify time distribution and comparing the measured offset of pulses generated by the control system against their set values to verify control traffic. In both scenarios, each endpoint received the total available bandwidth B. This shows that both the maximum fanout N and necessary depth M can be achieved. With B being independent of m and n and experiments by Bai and Prados [140] having shown latency for broadcast to be independent of n and proportional to m, it is safe to assume that increasing n to capacity will also result in a functional system. It therefore follows that the CS approach proposed in this thesis will scale to several thousand controlled machines by broadcasting control messages over a network tree on GbE over fibre.

11.4.2 Distance

Experimental proof that the approach can bridge the required distance of 2 km is a trivial matter. The experimental setup consisted of a WR switch and two endpoints, each connected over a cable spool containing 2 km of fibre. This experiment was already conducted by Lipiński, Włostowski, Serrano, *et al.* [47] in 2011, including WR correcting the thermal drift caused by the application of a hot air gun to the spools. The experiment was replicated at GSI up to 10 km and the findings confirmed that the WR system does correctly compensate the ensuing transmission delays.

11.4.3 Timing Accuracy

Resolution and Accuracy The requested timing resolution for command execution of 1 ns could be proven by experiment. The experiment setup consisted of two separate timing receivers connected by a cascade of 5 WR switches, which were in turn connected to the DM. The digital signal outputs of both timing receivers were hooked up to an oscilloscope, and the DM issued commands to generate two pulses with 1 ns offset (figure 11.2).



Measure	P1:freq@lv(C2)	P2:dper@lv(C2)	P3:freq@lv(C3)	P4:dper@lv(C3)	P5:skew(C2,C3)	P6:E2E(C1,C2)	P7:- - -	P8:- - -
value	---	---	---	---	1.0425 ns			
mean	---	---	---	---	1.0425 ns			
min	---	---	---	---	1.0425 ns			
max	---	---	---	---	1.0425 ns			
sdev	---	---	---	---	---			
num	0	0	0	0	1			
status	⚠	⚠	⚠	⚠	✓			

C2 DC50 500 mV/div -1.020 V
C3 DC50 500 mV/div -1.010 V
Timebase 80 ps 1.00 ns/div 400 S 40 GS/s
Trigger C2 DC Norm. 970 mV Edge Positive

FIGURE 11.2: Measurement on outputs of two timing receivers (red and green curves). Time difference between curves (P5 skew measurement) shows the achieved timing resolution and accuracy when executing DM commands for two pulses offset by 1 ns

Each timing receiver was programmed to execute only one of the commands. The outcome can be seen in figure 11.2, showing clearly that a resolution of 1 ns could indeed be achieved. The remaining 42.5 ps difference is comprised of the synchronisation uncertainty of the WR system and the measurement error of the oscilloscope.

Latency Bound It was shown by both delay model simulations and experiment, that the desired control loop speed of 500 µs (one way) could be achieved (see figure 10.8 and section 10.6.3), control loop speed only being limited by the reaction times of external inputs (1 ms for B2B, up to 10 ms for interlocks and experiment requests). Both the formal model and the numerical simulations confirmed that control message rate is, in the final FAIR system employing FEC, only limited by the available net bandwidth in the WR network of 192 Mbit/s. Combining these findings, it can be shown theoretically that timely arrival can be guaranteed for specified traffic both offline and for limited changes during runtime. It was validated by experiment that these findings hold true for all sequences known ahead of time.

In addition, it is shown that the proposed CS supports fully deterministic, parallel control processes, synchronised to the full accuracy of the system. Currently, a human readable intermediate format has been chosen in the form of XML strings, each containing a machine schedule. With the addition of hardware modules for sorting and sending command messages, most sources for non-determinism, such as message aggregation, network protocol coding and synchronisation handling could already be removed from the DM design. Approaches for dealing with remaining sources of jitter are touched upon under 11.5.1.

Months of tests in the GSI TTF, the successful control of the Saclay ion source and control of the CRYRING accelerator have shown that these properties also hold true under real conditions, in a complete CS setup. Moreover, a control scheme that allows machine schedules to be steered at runtime is presented and has been validated in the GSI TTF, running control sequences for a linear injector and a synchrotron ring in parallel.

11.5 Outlook & Future Work

Despite all the the effort which has gone into this project and all successes so far, much future work still remains.

11.5.1 Optimisations

DM Optimisations There is still much potential for optimisation in the DM SoC, especially hardware optimisations. Memory is currently attached to CPUs via WB and therefore needs four cycles of access time. Binding one port of its dual port memory without a WB interface directly to the CPU would result in a constant access time of one cycle, speeding up message processing considerably. At the same time it would remove any need for caches, thus freeing memory and removing a possible source of jitter.

There is plenty more memory optimisation to be undertaken, as about 30% of the RAM blocks in other parts the DM design are badly utilised. The standard memory blocks are organised as units of 10 kbit (M10K). Components wrongfully acquiring an M10K when trying to use only a small amount of memory are wasting the leftover bits. This memory could be freed and added to the pool of RAM available to the LM32 CPUs, enabling more CPUs or pre-loading more machine schedules.

Simulation Current simulation material was obtained by processing machine schedules manually in Octave/Matlab [149], calculating their minimal arrival curves and concave hulls. An automated process for the deduction of arrival curves from machine schedules has yet to be implemented.

An collaboration has been proposed by one of the creators of the DiscoDNC simulation environment at the Distributed Computer (Disco) Systems Lab at the University of Kaiserslautern, Germany, and was started in March 2017. This involves the research and development of a solution which calculates NC arrival curves from machine schedules. The algorithm would be used for delay calculation in FAIR's CS and is currently being developed in the scope of a bachelor thesis.

In addition, DiscoDNC does not natively support network tunnelling. The solution utilising separate calculations and adding the resulting bounds described in chapter 9.10.5 is a workable solution, but does not necessarily provide tight bounds. An improved version is currently under development with the consultancy of NC experts from Disco.

11.5.2 New Features

Etherbone As of mid 2016, there are very few extensions to EB planned. One consists of changing the EB network core to be able to be shared between multiple hosts, others are minor diagnostic features for hardware cores such as the addition of packet counters.

DM Features The firmware approach presented in chapter 8 has not been fully integrated yet (see 10.6.1) and still needs to be verified in a full test system. Furthermore, to better test the complete FAIR CS stack, logging capability for mean, minimum and maximum difference between message arrival and execution time shall be added to the ECA unit. Another possible implementation could be a separate debug module between the ECA and the top CB switch, looping ECA traffic through. While such an approach offers an easier implementation, the impact on ECA performance would still need to be carefully evaluated beforehand.

11.5.3 Integration with FAIR

The DM has yet to be fully integrated with the FAIR CS and further prove its ability to control real world accelerators of increasing complexity. As figure 11.3 shows, a concept for the software interface and control of the DM by operators, beam requests and machine interlocks already exists. Of the depicted software modules, the Generator is part of the DM project. The Generator is converting all abstract commands into hardware view, i.e. memory addresses and pointers. It is also responsible for automatic conversion of physical parameters into machine schedules.

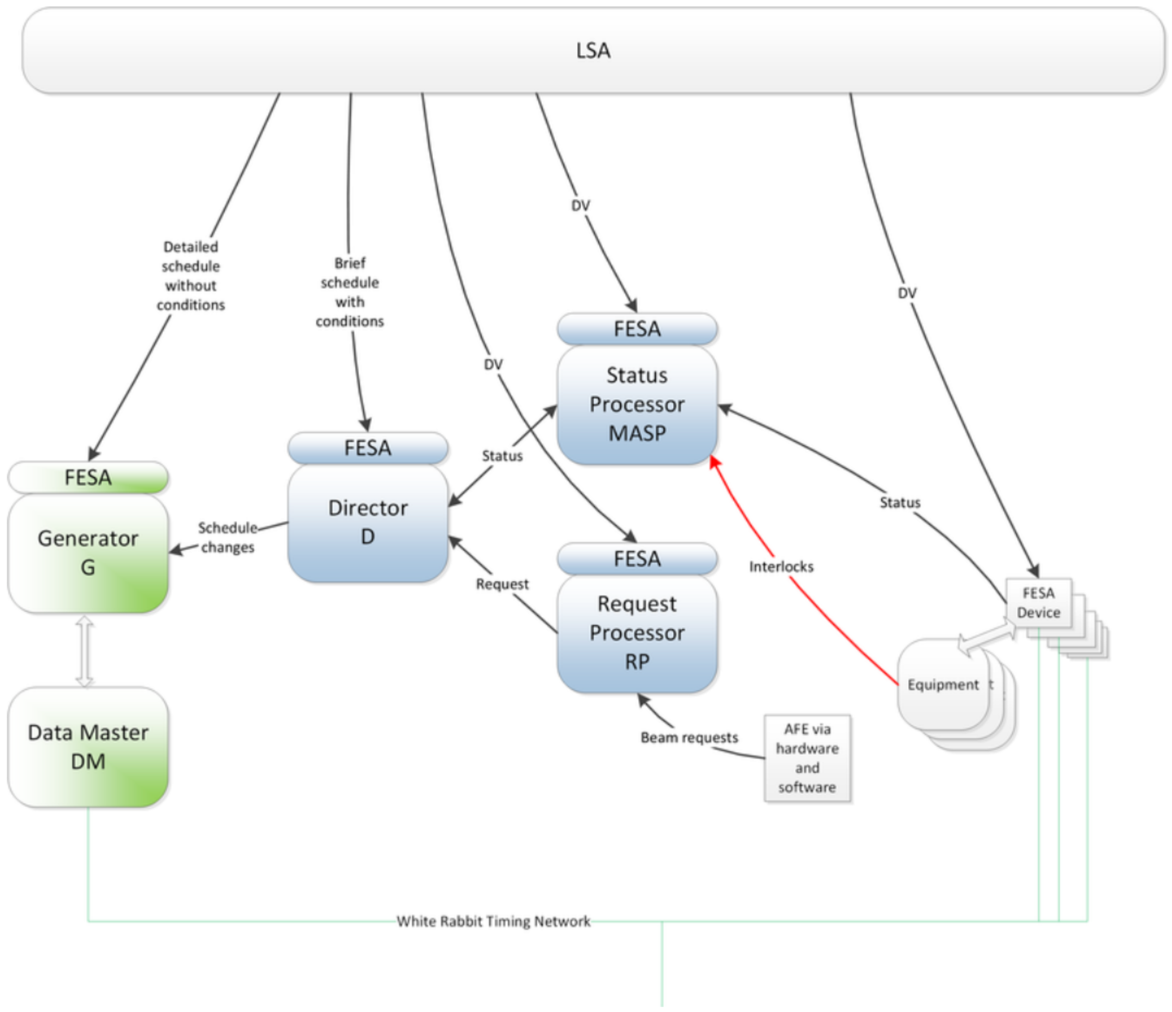


FIGURE 11.3: Excerpt from the Control System Interface Design Process

The Generator modules contain large contributions of the research and designs in chapter 8. Currently only part of the functionality is implemented in the form of several libraries, a full prototype is planned for fall 2017. The Director module's task is to combine input from the physics framework, interlock controller and beam requests and order changes from the Generator module regarding its path through machine schedules. The CS presented in this thesis will face its next big challenge, taking control of GSI's SIS18 synchrotron ring, in early 2018. However, the real trial is yet to come in 2025 when the complete FAIR facility is in operation.

Appendix A

Source Code

A.1 Data Master Gateware & Firmware

All sources regarding the DM as well as the complete FAIR timing system are public domain under the GNU Lesser General Public License Version 3. Sources are available at GitHub at the repository of the GSI control system division under the name "bel_projects" The current (01/12/2016) release branch "balloon" can be found and browsed under https://github.com/GSI-CS-CO/bel_projects/tree/balloon. All DM development/experimental branches bear the prefix "dm", the most recent development for the full CS is always to be found under the "proposed_master" branch.

All DM specific sources can found under the directory *modules/ftm*, all EB code is a sub-module to bel_projects imported from the Open Hardware Repository and can be found under the directory *ip_cores/modules/etherbone-core*. The sub-module branch used for EB is called "proposed_master".

A.1.1 Checkout of Project Sources

```
git clone --recursive https://github.com/GSI-CS-CO/bel_projects.git
cd MYPATH/bel_projects
git checkout <branchname>
git submodule update --init --recursive
```

The build workflow is fully make / automake based and is currently supported for several Linux distributions such as Debian, Ubuntu and Archlinux. A windows build workflow is not provided and it has proven difficult to build the projects on Windows computers in the past.

An important prerequisite for building FPGA images is to have a compatible locale setting. The Quartus design software only supports locale settings using a point as decimal separator, a comma (as normally used in German) will inevitably lead to wrong PLL frequencies and following build errors.

A complete guide on how to checkout, build and work with the project sources of GSI/FAIR's future control system can be found on the GSI Timing Wiki at https://www-acc.gsi.de/wiki/Timing/TimingSystemHowConfigureEnvironment.

Appendix B

Employed Software

B.1 Programming Languages

- VHDL 2003
- Verilog
- ANSI C99
- C++
- Python 2.7
- Matlab / Octave
- Bash script
- Make

B.2 3rd Party Tools

B.2.1 Development & Verification

- Synthesis & Compilation
  - *Altera Quartus 16.0.2* – VHDL/Verilog Synthesis
  - *Xilinx ISE* – VHDL/Verilog Synthesis
  - *GCC 4.3.3 for x86* – C/C++ Compilation
  - *GCC 4.3.3 for LM32* – C/C++ Compilation
- Simulation
  - *MentorG. QuestaSim 10.1* – VHDL/Verilog Simulation
  - *CocoTb Verification* – VHDL/Verilog Verification
- Monitoring
  - *Wireshark Network Analyser* – Packet capture and analysis

B.3 Custom Inhouse Tools

B.3.1 Etherbone

- Discovery
  - *eb-discover* – List EB slaves (EBS) on the network
  - *eb-find* – Find an SDB device on the EBS's bus by ID
  - *eb-ls* – List SDB devices on the bus of an EBS
- Programming
  - *eb-flash* – Program the Flash memory of an EBS
  - *eb-fwload* – Load firmware to CPUs of an EBS
- Direct bus access
  - *eb-get* – Download binary file from RAM in an EBS
  - *eb-put* – Upload binary file to a RAM in an EBS
  - *eb-read* – Read from a WB address within an EBS
  - *eb-write* – Write a value to a WB address within an EBS
- Introspection
  - *eb-snoop* – Snoops on ports of open EB connections
  - *EB dissector* – Packet dissector for Wireshark
- Misc
  - *eb-console* – Serial console over EB
  - *eb-info* – Get build information from an EBS
  - *eb-mon* – Get WR and EB status information from an EBS
  - *eb-tunnel* – Redirect EB connection between PC's interfaces

B.3.2 Wishbone

- Code Generator
  - *WBGenPlus* – VHDL from XML for WB Slaves Interfaces

B.3.3 DataMaster

- Code Generator
  - *ScheduleMaster* – C from Accelerator Schedule for machine control

Appendix C

Tables

C.1 List of Publications

Authors	Title	Published in or presented at	Type	Pages	Year
Serrano, J., Alvarez, P., Cattin, M., Garcia Cota, E., Lewis, J., Moreira, P., Wlostowski, T., Gaderer, G. T., Loschmidt, P., Dedic, J., Bär, R., Fleck, T., Kreider, M., Prados, C., Rauch, S.	The White Rabbit Project	Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS	Conf. Paper	93 - 95	2009
Bär, R.,Fleck, T., Kreider, M., Mauro, S.	The Timing Master for the FAIR Accelerator Facility	Proc. of the Int. Conf. on Accelerator and Large Experimental Physics Control Systems ICALEPCS	Conf. Paper	642-645	2011
Kreider, M., Terpstra, W., Lewis, J., Serrano, J., Wlostowski, T.	EtherBone - At network layer for the Wishbone SoC Bus	Proc. of the Int. Conf. on Accelerator and Large Experimental Physics Control Systems ICALEPCS	Conf. Paper		2011

Kreider, M., Bär, R, Beck, D., Terpstra, W., Davies, J., Grout, V., Lewis, J. Serrano, J., Wlostowski, T.	Open borders for system-on-a-chip buses: A wire format for connecting large physics controls	Physical Review Special Topics - Accelerators and Beams	Journal		2012
Kreider, M., Bär, R., Beck, D., Davies, J., Grout, V.	The FAIR Timing Master: A Discussion of Performance Requirements and Architectures for a High-precision Timing System	Proceedings of the Fifth Int. Conf. on Internet Technologies and Applications (ITA 13)	Peer Rev. Conf. Paper		2013
Kurz, N., Adamczewski-Musch, J., Frühauf, J., Hoffmann, J., Beck, D., Kreider, M., Prados, C., Rauch, S., Terpstra, W., Zweig, M.	White Rabbit Applications for FAIR Experiments	GSI Scientific Report	Journal	380	2013
Kreider, M., Bär, R., Beck, D., Terpstra, W., Davies, J., Grout, V.	New Developments on the FAIR Data Master	Proceedings of the 9th Int. Conf. on Personal Computers and Particle Accelerator Controls PCaPAC	Conf. Paper	204-206	2014
Kreider, M., Bai, J., Beck, D., Hahn, A., Prados, C., Rauch, S., Terpstra, W., Zweig, M.	Launching the FAIR Timing System with CRYRING	Proceedings of the 10th Int. Conf. on Personal Computers and Particle Accelerator Controls PCaPAC	Conf. Paper	152-154	2015

Terpstra, W., Kreider, M.	Message Signalled Interrupts in Mixed-Master Control	Proc. of the 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems ICALEPCS	Conf. Paper	152-154	2015
Kreider, M.	To the very nanosecond: A tutorial on open source timing and control systems	Sixth Int. Conf. on Internet Technologies and Applications (ITA 13)	Invited Talk		2015

TABLE C.1: Publications with Contributions from this Research Work

C.2 Tables of Simulation Values

Scaler	**Symbolic**	**Numeric**	**Scaling Factor**
$S_R^{-1}(a)$	$\frac{l_p}{l_p+l_h} \cdot a$	$\frac{32+8}{32} \cdot a$	0.8
$S_P^{-1}(a)$	$\frac{\Phi}{\Phi+l_{nh}} \cdot a$	$\frac{36(32+8)+46}{36(32+8)} \cdot a$	0.9626
$S_F^{-1}(a)$	$\frac{\Phi+l_{nh}}{4\Phi+l_{nh}} \cdot a$	$\frac{1}{4} \cdot a$	0.25
$S_1^{-1}(a)$	$S_R^{-1}(a)$	$\frac{4}{5} \cdot a$	0.8
$S_2^{-1}(a)$	$S_P^{-1}(S_R^{-1}(a))$	$\frac{4}{5} \cdot \frac{180}{187} \cdot a$	0.7701
$S_3^{-1}(a)$	$S_F^{-1}(S_P^{-1}(S_R^{-1}(a)))$	$\frac{4}{5} \cdot \frac{180}{187} \cdot \frac{1}{4} \cdot a$	0.1925

TABLE C.2: Rate Scaling Factors for all Models

Node	Rate			Latency		
	Symb.	Num.	Mbit/s	Symb.	Num.	µs
CB 3	r_3	4 Gbit/s	4000.0	T_{cb3}	$4 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.032
PQ	r_3	4 Gbit/s	4000.0	T_p	$6 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.048
PQ In	r_3	4 Gbit/s	4000.0	T_p	$6 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.064
CB 2	r_2	2 Gbit/s	2000.0	T_{cb2}	$4 \cdot \frac{32\,\text{bit}}{2\,\text{Gbit/s}}$	0.064
EBS DF	r_2	2 Gbit/s	2000.0	T_{ebs}		0.064
EBM F	r_2	2 Gbit/s	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\text{B}}{2\,\text{Gbit/s}}$	0.128
ECA In	r_2	2 Gbit/s	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\text{B}}{2\,\text{Gbit/s}}$	0.128
FEC1	$S_2^{-1}(r_2)$	$0.7701 \cdot 2\,\text{Gbit/s}$	1540.2	$\frac{k \cdot l_p}{S_2^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.7701 \cdot 2\,\text{Gbit/s}}$	0.748
DFEC3	$S_2^{-1}(r_2)$	$0.7701 \cdot 2\,\text{Gbit/s}$	1540.2	$\frac{k \cdot l_p}{S_2^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.7701 \cdot 2\,\text{Gbit/s}}$	0.748
FEC2	∞^+			T_e		2.000
DFEC2	∞^+			T_d		2.000
PQ Sort	∞^+			T_e		2.000
FEC3	$S_3^{-1}(r_2)$	$0.1925 \cdot 2\,\text{Gbit/s}$	385.0	$\frac{k \cdot l_p}{S_3^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.1925 \cdot 2\,\text{Gbit/s}}$	2.992
DFEC1	$S_3^{-1}(r_2)$	$0.1925 \cdot 2\,\text{Gbit/s}$	385.0	$\frac{k \cdot l_p}{S_3^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.1925 \cdot 2\,\text{Gbit/s}}$	2.992
ECA Sort	∞^+			T_{eca}		4.000
CPU	$\frac{l_p}{T_{msg}}$	$\frac{32\,\text{B}}{5\,\mu\text{s}}$	51.2	T_{msg}		5.000
EBS RX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\text{Gbit/s}$	1600.0	$\frac{k \cdot l_p}{S_1^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.8 \cdot 2\,\text{Gbit/s}}$	5.760
NIC	$S_3^{-1}(r_1)$	$0.1925 \cdot 1\,\text{Gbit/s}$	192.5	$\frac{l_{max}}{r_1}$	$\frac{1500\,\text{B}}{1\,\text{Gbit/s}}$	12.000
WR SW	$S_3^{-1}(r_1)$	$0.1925 \cdot 1\,\text{Gbit/s}$	192.5	$\frac{l_{max}}{r_1} + T_{sw}$	$\frac{1500\,\text{B}}{1\,\text{Gbit/s}} + 0.5\,\mu\text{s}$	12.500
EBM TX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\text{Gbit/s}$	1600.0	$\frac{l_{nh} + k \cdot l_p}{S_1^{-1}(r_1)}$	$\frac{56 + 36 \cdot 32\,\text{B}}{0.1925 \cdot 1\,\text{Gbit/s}}$	47.780

TABLE C.3: NC Node Values, final FAIR implementation

Node	**Rate**			**Latency**		
	Symb.	Num.	Mbit/s	Symb.	Num.	µs
CB 3	r_3	4 Gbit/s	4000.0	T_{cb3}	$4 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.032
PQ	r_3	4 Gbit/s	4000.0	T_p	$6 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.048
PQ In	r_3	4 Gbit/s	4000.0	T_p	$6 \cdot \frac{32\,\text{bit}}{4\,\text{Gbit/s}}$	0.064
CB 2	r_2	2 Gbit/s	2000.0	T_{cb2}	$4 \cdot \frac{32\,\text{bit}}{2\,\text{Gbit/s}}$	0.064
EBS DF	r_2	2 Gbit/s	2000.0	T_{ebs}		0.064
EBM F	r_2	2 Gbit/s	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\text{B}}{2\,\text{Gbit/s}}$	0.128
ECA In	r_2	2 Gbit/s	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\text{B}}{2\,\text{Gbit/s}}$	0.128
FEC1	-	-	-	-	-	-
DFEC3	-	-	-	-	-	-
FEC2	-	-	-	-	-	-
DFEC2	-	-	-	-	-	-
PQ Sort	∞^+			T_e		2.000
FEC3	-	-	-	-	-	-
DFEC1	-	-	-	-	-	-
ECA Sort	∞^+			T_{eca}		4.000
CPU	$\frac{l_p}{T_{msg}}$	$\frac{32\,\text{B}}{5\,\mu\text{s}}$	33.0	T_{msg}		7.750
EBS RX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\text{Gbit/s}$	1600.0	$\frac{k \cdot l_p}{S_1^{-1}(r_2)}$	$\frac{36 \cdot 32\,\text{B}}{0.8 \cdot 2\,\text{Gbit/s}}$	5.760
NIC	$S_2^{-1}(r_1)$	$0.7701 \cdot 1\,\text{Gbit/s}$	770.1	$\frac{l_{max}}{r_1}$	$\frac{1500\,\text{B}}{1\,\text{Gbit/s}}$	0.500
WR SW	$S_2^{-1}(r_1)$	$0.7701 \cdot 1\,\text{Gbit/s}$	770.1	T_{sw}	0.5 µs	0.500
EBM TX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\text{Gbit/s}$	1600.0	$\frac{l_{nh}+k \cdot l_p}{S_1^{-1}(r_1)}$	$\frac{56+36 \cdot 32\,\text{B}}{0.1925 \cdot 1\,\text{Gbit/s}}$	47.780

TABLE C.4: NC Node Values, 2016 GSI TTF Implementation

Node	**Rate**			**Latency**		
	Symb.	Num.	Mbit/s	Symb.	Num.	µs
CB 3	r_3	$4\,\mathrm{Gbit/s}$	4000.0	T_{cb3}	$4 \cdot \frac{32\,\mathrm{bit}}{4\,\mathrm{Gbit/s}}$	0.032
PQ	r_3	$4\,\mathrm{Gbit/s}$	4000.0	T_p	$6 \cdot \frac{32\,\mathrm{bit}}{4\,\mathrm{Gbit/s}}$	0.048
PQ In	r_3	$4\,\mathrm{Gbit/s}$	4000.0	T_p	$6 \cdot \frac{32\,\mathrm{bit}}{4\,\mathrm{Gbit/s}}$	0.064
CB 2	r_2	$2\,\mathrm{Gbit/s}$	2000.0	T_{cb2}	$4 \cdot \frac{32\,\mathrm{bit}}{2\,\mathrm{Gbit/s}}$	0.064
EBS DF	r_2	$2\,\mathrm{Gbit/s}$	2000.0	T_{ebs}		0.064
EBM F	r_2	$2\,\mathrm{Gbit/s}$	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\mathrm{B}}{2\,\mathrm{Gbit/s}}$	0.128
ECA In	r_2	$2\,\mathrm{Gbit/s}$	2000.0	$\frac{l_p}{r_2}$	$\frac{32\,\mathrm{B}}{2\,\mathrm{Gbit/s}}$	0.128
FEC1	$S_2^{-1}(r_2)$	$0.7701 \cdot 2\,\mathrm{Gbit/s}$	1540.2	$\frac{k \cdot l_p}{S_2^{-1}(r_2)}$	$\frac{36 \cdot 32\,\mathrm{B}}{0.7701 \cdot 2\,\mathrm{Gbit/s}}$	0.748
DFEC3	$S_2^{-1}(r_2)$	$0.7701 \cdot 2\,\mathrm{Gbit/s}$	1540.2	$\frac{k \cdot l_p}{S_2^{-1}(r_2)}$	$\frac{36 \cdot 32\,\mathrm{B}}{0.7701 \cdot 2\,\mathrm{Gbit/s}}$	0.748
FEC2	∞^+			T_e		2.000
DFEC2	∞^+			T_d		2.000
PQ Sort	∞^+			T_e		2.000
FEC3	$S_3^{-1}(r_2)$	$0.1925 \cdot 2\,\mathrm{Gbit/s}$	385.0	$\frac{k \cdot l_p}{S_3^{-1}(r_2)}$	$\frac{36 \cdot 32\,\mathrm{B}}{0.1925 \cdot 2\,\mathrm{Gbit/s}}$	2.992
DFEC1	$S_3^{-1}(r_2)$	$0.1925 \cdot 2\,\mathrm{Gbit/s}$	385.0	$\frac{k \cdot l_p}{S_3^{-1}(r_2)}$	$\frac{36 \cdot 32\,\mathrm{B}}{0.1925 \cdot 2\,\mathrm{Gbit/s}}$	2.992
ECA Sort	∞^+			T_{eca}		4.000
CPU	$\frac{l_p}{T_{msg}}$	$\frac{32\,\mathrm{B}}{5\,\mu\mathrm{s}}$	33.0	T_{msg}		7.750
EBS RX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\mathrm{Gbit/s}$	1600.0	$\frac{k \cdot l_p}{S_1^{-1}(r_2)}$	$\frac{36 \cdot 32\,\mathrm{B}}{0.8 \cdot 2\,\mathrm{Gbit/s}}$	5.760
NIC	$S_2^{-1}(r_1)$	$0.7701 \cdot 1\,\mathrm{Gbit/s}$	770.1	$\frac{l_{max}}{r_1}$	$\frac{1500\,\mathrm{B}}{1\,\mathrm{Gbit/s}}$	0.500
WR SW	-	-	-	-	-	-
EBM TX	$S_1^{-1}(r_2)$	$0.8 \cdot 2\,\mathrm{Gbit/s}$	1600.0	$\frac{l_{nh} + k \cdot l_p}{S_1^{-1}(r_1)}$	$\frac{56 + 36 \cdot 32\,\mathrm{B}}{0.1925 \cdot 1\,\mathrm{Gbit/s}}$	47.780

TABLE C.5: NC Node Values, Minimal Test Implementation

Flow	**Rate**			**Burst**		
	Symb.	Num.	kbit/s	Symb.	Num.	B
WR	α_{ptp}	$4 \cdot 490\,\text{B/s}$	16	l_{ptp}	$490\,\text{B}$	490
ARP	α_{arp}	$64\,\text{B/s}$	0.5	l_{arp}	$64\,\text{B}$	64
DHCP	α_{dhcp}	$576\,\text{B/s}$	4.6	l_{dhcp}	$576\,\text{B}$	576
SNMP	α_{snmp}	$1500\,\text{B/s}$	6	l_{snmp}	$1500\,\text{B}$	490

TABLE C.6: Estimated Background Flow Values for WR Systems without QoS (TTF, Mini Test)

References

[1] Institute of Electrical and Electronics Engineers (IEEE), *Ieee xplore digital library*. [Online]. Available: http://ieeexplore.ieee.org (visited on 05/29/2016).

[2] Association for Computing Machinery (ACM), *ACM digital library*. [Online]. Available: http://dl.acm.org/ (visited on 05/29/2016).

[3] Springer, *Springer online library*. [Online]. Available: http://www.springer.com/gb/computer-science (visited on 05/29/2016).

[4] Google Inc., *Google scholar*. [Online]. Available: https://scholar.google.com/ (visited on 05/29/2016).

[5] Pennssylvania State University, *Citeseerx*. [Online]. Available: http://citeseerx.ist.psu.edu/ (visited on 05/29/2016).

[6] Google Inc., *Google books*. [Online]. Available: https://books.google.com/ (visited on 05/29/2016).

[7] Open Hardware Repository (OHWR), *Form factors for White Rabbit based Control Systems*, Nov. 2016. [Online]. Available: http://www.ohwr.org/projects (visited on 11/30/2016).

[8] M. Kreider, J. Bai, D. Beck, A. Hahn, C. Prados, S. Rauch, W. W. Terpstra, and M. Zweig, "Launching the FAIR Timing System with CRYRING", English, in *Proceedings of the 10th International workshop on personal computers and particle accelerator controls*, OCLC: 945432373, Eggenstein-Leopoldshafen: Synchrotron radiation source ANKA, 2015, pp. 152–154, ISBN: 978-3-95450-146-5. [Online]. Available: http://accelconf.web.cern.ch/1E2790D3-C34E-4E94-806A-EB670A9C6B4A/FinalDownload/DownloadId-0A05DAA3CCE5C084AC64AC1B31F1D956/1E2790D3-C34E-4E94-806A-EB670A9C6B4A/AccelConf/PCaPAC2014/papers/proceed.pdf (visited on 11/28/2016).

[9] Open Hardware Repository (OHWR), *Users of White Rabbit Technology*, Nov. 2016. [Online]. Available: http://www.ohwr.org/projects/white-rabbit/wiki/WRUsers (visited on 11/30/2016).

[10] International Organization for Standardization (ISO), "ISO5725-6: Accuracy (trueness and precision) of measurement methods and results-Part 6: Use in practice of accuracy values", 1994.

[11] *Oxford dictionary*. [Online]. Available: http://www.oxforddictionaries.com/definition/english/clock (visited on 05/29/2016).

[12] Tektronix Inc., *Understanding and Characterizing Timing Jitter*, Aug. 2003. [Online]. Available: info.tek.com/rs/tektronix/images/55W_16146_5_MR_Letter.pdf (visited on 11/25/2016).

[13] R. A. Nelson, D. D. McCarthy, S. Malys, J. Levine, B. Guinot, H. F. Fliegel, R. L. Beard, and T. R. Bartholomew, "The leap second: its history and possible future", *Metrologia*, vol. 38, no. 6, p. 509, 2001.

[14] M. A. Lombardi, L. M. Nelson, A. N. Novick, and V. S. Zhang, "Time and frequency measurements using the global positioning system", *Cal Lab: International Journal of Metrology*, vol. 8, no. 3, pp. 26–33, 2001. [Online]. Available: http://www.glb.nist.gov/calibrations/upload/1424.pdf (visited on 06/11/2016).

[15] *Nautical Almanac of the stars*, Jan. 2016. [Online]. Available: www.nauticalalmanac.it (visited on 05/28/2016).

[16] M. Denny, *The science of navigation: from dead reckoning to GPS*. JHU Press, 2012.

[17] J. W. Norie, *A new and complete epitome of practical navigation*. Rarebooksclub.com, 1852, ISBN: 1-236-63584-1.

[18] R. T. Gould, *The marine chronometer, its history and development*. Acc Art Books, 2013, ISBN: 1-85149-365-4.

[19] M. A. Lombardi, "The Accuracy & Stability of Quartz Watches", *Horological Journal*, vol. 150, no. 2, p. 57, 2008. [Online]. Available: http://tf.nist.gov/general/pdf/2276.pdf?iframe=true&width=100%&height=100% (visited on 06/13/2016).

[20] M. E. Frerking, "Fifty years of progress in quartz crystal frequency standards", in *Frequency Control Symposium, 1996. 50th., Proceedings of the 1996 IEEE International.*, IEEE, 1996, pp. 33–46.

[21] W. M. Itano and N. F. Ramsey, "Accurate measurement of time", *Scientific American*, vol. 269, no. 1, pp. 46–53, 1993.

[22] T. P. Heavner, E. A. Donley, F. Levi, G. Costanzo, T. E. Parker, J. H. Shirley, N. Ashby, S. Barlow, and S. R. Jefferts, "First accuracy evaluation of NIST-F2", *Metrologia*, vol. 51, no. 3, p. 174, 2014.

[23] Y. B. Ovchinnikov, "Development of NPL Rb fountain frequency standard", in *European Frequency and Time Forum (EFTF), 2012*, IEEE, 2012, pp. 96–100.

[24] N. Hinkley, J. A. Sherman, N. B. Phillips, M. Schioppo, N. D. Lemke, K. Beloy, M. Pizzocaro, C. W. Oates, and A. D. Ludlow, "An atomic clock with 10–18 instability", *Science*, vol. 341, no. 6151, pp. 1215–1218, 2013.

[25] E. F. Arias, B. Guinot, and T. J. Quinn, "Rotation of the Earth and Time scales", in *ITU-R SRG Colloquium on the UTC Time Scale*, 2003, pp. 28–29.

[26] L Essen and J. Parry, "An atomic standard of frequency and time interval: a caesium resonator", *Nature*, vol. 176, no. 4476, pp. 280–282, 1955.

[27] Physikalisch Technische Bundesanstalt, *Die Zeitskalen TAI und EAL*. [Online]. Available: http://www.ptb.de/cms/ptb/fachabteilungen/abt4/fb-44/ag-441/darstellung-der-gesetzlichen-zeit/die-zeitskalen-tai-und-eal.html (visited on 06/13/2016).

[28] US Department of Defense, "GPS SPS performance standard", Tech. Rep., 2008. (visited on 06/10/2016).

[29] H. Clausert, G. Wiesemann, V. Hinrichsen, and J. Stenzel, *Grundgebiete der Elektrotechnik 2*. Oldenbourg Verlag, 2007, vol. 2, ISBN: 3-486-25428.

[30] P. Horowitz, W. Hill, and T. C. Hayes, *The art of electronics, 3rd editon*, 3rd ed. Cambridge university press Cambridge, 2015, ISBN: 978-0-521-80926-9.

[31] D. Banerjee, *PLL performance, simulation and design*. Dog Ear Publishing, 2006, ISBN: 1-59858-131-1.

[32] B. Razavi, *Monolithic phase-locked loops and clock recovery circuits: theory and design*. John Wiley & Sons, 1996, ISBN: 0-7803-1149-3.

[33] A. Widmer, *Partitioned dc balanced (0,6) 16b/18b transmission code with error correction*, US Patent 6,198,413, Mar. 2001. [Online]. Available: https://www.google.com/patents/US6198413.

[34] GPS Timing Subcommittee, "Timing Subcommittee Report", US Government, Tech. Rep., 2014. (visited on 06/10/2016).

[35] M. A. Lombardi, "Computer time synchronization", *NIST, Time and Frequency Division*, 2000. [Online]. Available: http://elektron.pol.lublin.pl/users/ELEKP/ap_notes/NIST_comp_time_synchro.pdf (visited on 06/13/2016).

[36] ——, "How accurate is a radio controlled clock", *The Horological Journal*, vol. 152, no. 3, pp. 108–111, 2010. [Online]. Available: http://tf.boulder.nist.gov/general/pdf/2429.pdf (visited on 06/11/2016).

[37] D. Deeths, G. Brunette, and S. BluePrints, "Using NTP to control and synchronize system clocks-part i: Introduction to NTP", *Sun BluePrints OnLine-July*, 2001.

[38] M. A. Lombardi, "Comparing loran timing capability to industrial requirements", in *Proceedings of the 2006 International Loran Association (ILA) Meeting*, 2006. [Online]. Available: http://tf.nist.gov/general/pdf/2193.pdf (visited on 06/11/2016).

[39] P. Loschmidt, R. Exel, A. Nagy, and G. Gaderer, "Limits of synchronization accuracy using hardware support in IEEE 1588", in *Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008. IEEE International Symposium on*, IEEE, 2008, pp. 12–16.

[40] P. Moreira, J. Serrano, T. Włostowski, P. Loschmidt, and G. Gaderer, “White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet”, in *Proc. of the International Symposium on Precision Clock Synchronization for Measurement, Control and Communication ISPCS*, Oct. 2009, pp. 1–5.

[41] D. L. Mills, “Internet time synchronization: the network time protocol”, *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=103043 (visited on 05/09/2016).

[42] J. Serrano, M. Lipiński, T. Włostowski, E. Gousiou, E. van der Bij, M. Cattin, and G. Daniluk, “The White Rabbit project”, in *Proceedings of the 2nd International Beam Instrumentation Conference (IBIC2013)*, EDA Consortium, 2013. [Online]. Available: http://cds.cern.ch/record/1743073/files/thbl2.pdf (visited on 05/09/2016).

[43] K. Lee, J. C. Eidson, H. Weibel, and D. Mohl, “IEEE 1588-standard for a precision clock synchronization protocol for networked measurement and control systems”, in *Conference on IEEE*, vol. 1588, 2005, p. 2.

[44] D. L. Mills, “NTP performance analysis”, 2004. [Online]. Available: http://www.cis.udel.edu/~mills/database/brief/perf/perf.pdf (visited on 11/10/2016).

[45] ITU, Telecommunication Standardization Sector, “Distribution of timing information through packet networks”, International Telecommunication Union, Tech. Rep. G.8264, 2014.

[46] A. X. Widmer and P. A. Franaszek, “A DC-balanced, partitioned-block, 8b/10b transmission code”, *IBM Journal of research and development*, vol. 27, no. 5, pp. 440–451, 1983.

[47] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez, “White rabbit: a PTP application for robust sub-nanosecond synchronization”, in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, Sep. 2011, pp. 25–30. DOI: 10.1109/ISPCS.2011.6070148.

[48] E. Wilson and E. J. Wilson, *An introduction to particle accelerators*. Clarendon Press, 2001, ISBN: 0-19-850829-8.

[49] H. Schopper, *Advances of Accelerator Physics and Technologies*. World Scientific, 1993, ISBN: 981-02-0957-6.

[50] H. Wiedemann, *Particle accelerator physics*. Springer, 2015, ISBN: 978-3-319-18316-9.

[51] R. F. Harrington, *Introduction to electromagnetic engineering*. Courier Corporation, 2003, ISBN: 0-486-43241-6.

[52] GSI, *GSI Public Relations Material*, 2008.

[53] M. Oliphant, "The acceleration of particles to very high energies", *Classified memo submitted to DSIR University of Birmingham Archive*, 1943.

[54] J. Ross, *SIS18 Technische Zeichnungen und Fotografien*, 2008.

[55] P. Strehl, *Beam instrumentation and diagnostics*. Springer, 2006, vol. 120, ISBN: 3-540-26401-9.

[56] M. Kreider, R. Bär, D. Beck, J. Davies, and V. Grout, "The FAIR Timing Master: A Discussion of Performance Requirements and Architectures for a High-precision Timing System", in *Proceedings of the Fifth International Conference on Internet Technologies and Applications (ITA 13)*, Wrexham, UK: North East Wales Institute, Oct. 2013, ISBN: 978-0-946881-81-9.

[57] Facility for Anti-Proton and Ion Research (FAIR), *Overview of FAIR accelerator*, Aug. 2016. [Online]. Available: https://static1.bmbfcluster.de/5/4/8/4_cb57bb50968c33b/5484abg_62da0208476191f.jpg.

[58] G. Kraft, "Tumor therapy with heavy charged particles", *Progress in Particle and Nuclear Physics*, vol. 45, S473–S544, 2000.

[59] R. Steiner, G. Englert, W. Rösch, H. Brandis, H. Hübner, and E. Schaffner, "The GSI control system", 1990.

[60] U. Krause, V. Schaa, and R. Steiner, "The GSI Control System", in *Proceedings of ICALEPCS*, vol. 91, 1991.

[61] U. Krause and V. Schaa, "Re-engineering of the GSI control system", *arXiv preprint physics/0111060*, 2001. [Online]. Available: http://arxiv.org/abs/physics/0111060 (visited on 11/28/2016).

[62] Siemens, *Simatic Cycle and Response Times*, Jan. 2016. [Online]. Available: https://cache.industry.siemens.com/dl/files/558/59193558/att_895996/v1/s71500_cycle_and_reaction_times_function_manual_en-US_en-US.pdf (visited on 01/11/2016).

[63] R. V. Aroca and G. Caurin, "A real time operating systems (RTOS) comparison", 2009. [Online]. Available: http://home.iitj.ac.in/~saurabh.heda/Papers/Survey/RTOS%20Performance%20Comparison%20-2008.pdf (visited on 10/03/2016).

[64] B. Frammery, "The LHC Control System", in *Proceedings of ICALEPCS*, 2005, pp. 10–14. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ica05/proceedings/pdf/I1_001.pdf (visited on 11/28/2016).

[65] European Centre for Nuclear Research (CERN), *Overview of CERN accelerator*, Aug. 2016. [Online]. Available: http://www.stfc.ac.uk/stfc/includes/themes/MuraSTFC/assets/legacy/LHCinteractive/LHC_default.jpg.

[66] Argonne National Laboratory, *Experimental Physics and Industrial Control System (EPICS)*, 1995 – 2016. [Online]. Available: http://www.aps.anl.gov/epics/ (visited on 05/29/2016).

[67] M. G. Abbott, J. A. Dobbing, M. T. Heron, G. Rehm, J. Rowland, I. S. Uzun, and S. Duncan, "Performance and future development of the diamond fast orbit feedback system", *Proceedings of EPAC08*, p. 3257, 2008.

[68] A. Gamp, M. Liepe, T. Plawski, K. Rehlich, and S. N. Simrock, "Design of the RF phase reference system and timing control for the TESLA linear collider", in *XIX International Linear Accelerator Conference*, vol. 98, 1998.

[69] A. Winter, P. Schmüser, F. Ludwig, H. Schlarb, J. Chen, F. X. Kärtner, and F. Ö. Ilday, "High-precision laser master oscillators for optical timing distribution systems in future light sources", in *Proceedings of EPAC*, 2006.

[70] T. Włostowski, G. Daniluk, M. Lipiński, F. Vaga, and J. Serrano, "Trigger and RF Distribution Using White Rabbit", 2015. [Online]. Available: http://cds.cern.ch/record/2213491 (visited on 12/20/2016).

[71] P. Schütt, *Presentation Beamlines and Production Chains*, from GSI internal presentation material, 2010.

[72] P. Moritz, "BuTiS Development of a Bunchphase Timing System", GSI, Tech. Rep., 2006. [Online]. Available: http://gsi.helmholtz.de/informationen/wti/library/scientificreport2006/PAPERS/FAIR-ACCELERATORS-20.pdf.

[73] Facility for Anti-Proton and Ion Research (FAIR), *Detailed specification of the FAIR accelerator control system component „General Machine Timing System"*, Aug. 2012.

[74] N. Kurz, J. Adamczewski-Musch, J. Frühauf, J. Hoffmann, D. Beck, M. Kreider, C. Prados, S. Rauch, W. Terpstra, and M. Zweig, "White rabbit applications for fair experiments", *GSI Scientific Report*, 2013. DOI: 10.15120/GR-2014-1.

[75] N. Kurz, *Detector Requirements*, private communication, Jan. 2017.

[76] H. Klingbeil, *Presentation on bunch to bucket transfer*, from GSI internal presentation material, 2010.

[77] J. Bai, "Development of the timing system for the Bunch-to-Bucket transfer between the FAIR accelerators", Doctoral Thesis, Goethe Universität Frankfurt am Main, Nov. 2017.

[78] M. Lipiński and C. Prados, "White Rabbit Robustness", GSI/CERN, Tech. Rep., 2011. [Online]. Available: http://www.ohwr.org/attachments/742/WhiteRabbitAndRobustness.pdf.

[79] C. Prados Boda and T. Fleck, "FEC in deterministic control systems over Gigabit Ethernet", in *Proc. of the 8th International Workshop on Personal Computers and Particle Accelerators, PCaPAC*, 2010, pp. 172–174.

[80] M. Lipiński, C. Prados, J. Serrano, and T. Włostowski, "Reliability in a White Rabbit network", in *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, 2011, pp. 698–701.

[81] D. Beck, “Robustness”, Tech. Rep., Nov. 2013. [Online]. Available: https://www-acc.gsi.de/wiki/Timing/TimingSystemNetworkBasics (visited on 11/30/2016).

[82] IEEE Standards Association, *IEEE 802.3 Ethernet Standard*, 2015. [Online]. Available: http://standards.ieee.org/getieee802/download/802.3-2015.zip (visited on 05/29/2016).

[83] D. Beck, *Forward Error Correction Values*, Sep. 2016. [Online]. Available: https://www-acc.gsi.de/wiki/Timing/TimingSystemEvent (visited on 11/30/2016).

[84] T. Straumann, “Open Source real-time operating systems overview”, in *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, 2001, pp. 235–237.

[85] B. B. Brandenburg, “Scheduling and locking in multiprocessor real-time operating systems”, PhD thesis, 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.359.6393&rep=rep1&type=pdf (visited on 10/03/2016).

[86] R. H. Katz, *Contemporary Logic Design*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994, ISBN: 0-8053-2703-7.

[87] Altera Corp., “FPGA Architecture”, Tech. Rep., 2006. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01003.pdf (visited on 10/03/2016).

[88] ——, *Stratix X documentation*, 2015. [Online]. Available: https://www.altera.com/literature/hb/stx/stratix_handbook.pdf (visited on 09/12/2016).

[89] ——, *Arria V documentation*, 2015. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/arria-v/av_51001.pdf (visited on 09/10/2016).

[90] W. Terpstra, “The Case for Soft-CPUs in Accelerator Control Systems”, in *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, Oct. 2011, pp. 1252–1255.

[91] Lattice Semiconductor, *LM32-Processor Reference Manual*, Jul. 2016. [Online]. Available: http://www.latticesemi.com/view_document?document_id=50900 (visited on 09/26/2016).

[92] Opencores, “Wishbone B4 WISHBONE System-on-Chip (SoC)Interconnection Architecture for Portable IP Cores”, OpenCores, Tech. Rep., 2010. [Online]. Available: http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[93] M. Kreider, R. Bär, D. Beck, W. Terpstra, J. Davies, V. Grout, J. Lewis, J. Serrano, and T. Włostowski, "Open borders for system-on-a-chip buses: A wire format for connecting large physics controls", en, *Physical Review Special Topics - Accelerators and Beams*, vol. 15, no. 8, Aug. 2012, ISSN: 1098-4402. DOI: 10.1103/PhysRevSTAB.15.082801. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevSTAB.15.082801 (visited on 05/09/2016).

[94] A. Romanow and S. Bailey, "An Overview of RDMA over IP", in *Proc. of the First International Workshop on Protocols for Fast Long-Distance Networks PFLDnet*, Feb. 2003.

[95] R. Recio, P. Culley, D. Garcia, J. Hilland, and B. Metzler, "An RDMA protocol specification", Internet Engineering Task Force, Tech. Rep., 2005.

[96] J. Liu, J. Wu, and D. K. Panda, "High Performance RDMA Based MPI Implementation over Infiniband", *International Journal on parallel programming*, vol. 32, no. 3, pp. 167–198, 2004.

[97] M. J. Rashti and A. Afsahi, "10-Gigabit iWarp Ethernet: Comparative Performance Analysis with Infiniband and Myrinet-10g", in *Proc. of the International Parallel and Distributed Processing Symposium*, Mar. 2007.

[98] A. Gokhale and D. Schmidt, "Measuring and optimizing CORBA latency and scalability over high-speed networks", *IEEE Transactions on Computers*, vol. 47, no. 4, pp. 391–413, Apr. 1998.

[99] Y. Ying, Y. Huang, and D. W. Walker, "A Performance Evaluation of Using SOAP with Attachments for e-Science", in *In Proc. of the UK e-Science All Hands Meeting*, 2005, pp. 796–803.

[100] "PCI Express Base Specification Revision 3.0", PCI-SIG, Tech. Rep., 2010. [Online]. Available: http://www.pcisig.com/specifications/pciexpress/base3/.

[101] A. Rubini, W. Terpstra, and M. Vanga, "Self Describing Bus Specification", Open Hardware Repository, Tech. Rep., Apr. 2013. [Online]. Available: http://www.ohwr.org/attachments/4021/sdb-1.1.pdf (visited on 05/06/2016).

[102] J. B. Postel, "User Datagram Protocol", Internet Engineering Task Force, RFC 768, Aug. 1980. [Online]. Available: http://www.rfc-editor.org/rfc/rfc768.txt.

[103] ——, "Transmission Control Protocol", Internet Engineering Task Force, RFC 793, Sep. 1981, p. 85. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt.

[104] ——, "Internet Protocol", Internet Engineering Task Force, RFC 791, Sep. 1981, p. 45. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt.

[105] N. Alachiotis, S. A. Berger, and A. Stamatakis, “Efficient PC-FPGA Communication over Gigabit Ethernet”, in *Proc. of the 2010 10th IEEE International Conference on Computer and Information Technology*, ser. CIT ’10, 2010, pp. 1727–1734, ISBN: 978-0-7695-4108-2.

[106] D. Yadav and A. Rajawat, “Area and throughput analysis of different aes architectures for fpga implementations”, in *IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, Dec. 2016, pp. 67–71. DOI: 10.1109/iNIS.2016.026.

[107] Altera Corp., *Quartus prime design software v16.0.2*, 1991 – 2016. [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html (visited on 05/29/2016).

[108] M. Kreider, R. Bär, D. Beck, W. Terpstra, J. Davies, and V. Grout, “New Developments on the FAIR Data Master”, in *Proceedings of PCaPAC2014*, Karlsruhe, Germany, 2014, pp. 204–206, ISBN: 978-3-95450-146-5. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/PCaPAC2014/papers/fpo022.pdf (visited on 10/02/2016).

[109] W. M. Zabolotny, “Dual port memory based Heapsort implementation for FPGA”, in *Proc. SPIE*, vol. 8008, 2011, 80080E–80080E–9. DOI: 10.1117/12.905281. [Online]. Available: http://dx.doi.org/10.1117/12.905281.

[110] M. Peczarski, “New results in minimum-comparison sorting”, *Algorithmica*, vol. 40, no. 2, pp. 133–145, 2004.

[111] Altera Corp., *LVDS SERDES IP Core User Guide*, May 2016. [Online]. Available: https://www.altera.com/literature/ug/ug_altera_lvds.pdf (visited on 09/15/2016).

[112] T. H. Cormen, *Introduction to algorithms*. Massachusetts Institute of Technology, 2009, ISBN: 978-0-262-03384-8.

[113] Y. Han, “Deterministic sorting in O (n log log n) time and linear space”, in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ACM, 2002, pp. 602–608.

[114] GNU Project, *GNU Compiler Collection Manual*, 2007. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc-4.2.0/gcc/Explicit-Reg-Vars.html (visited on 05/29/2016).

[115] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, “Dynamic storage allocation: A survey and critical review”, in *Memory Management*, Springer, 1995, pp. 1–116.

[116] A. Ralston, E. D. Reilly, and D. Hemmendinger, *Encyclopedia of computer science*. Petrocelli/Charter New York, 2000, ISBN: 0-470-86412-5.

[117] W. Terpstra and M. Kreider, “Message Signalled Interrupts in Mixed-Master Control”, in *15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, 17-23 October 2015*, JACOW, Geneva, Switzerland, 2015, pp. 1083–1086. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2015/papers/thha2o03.pdf (visited on 01/01/2017).

[118] A. M. Turing, “On computable numbers, with an application to the entscheidungsproblem”, *Proceedings of the London mathematical society*, vol. 2, no. 1, pp. 230–265, 1937.

[119] S. E. Anderson, *Bit Twiddling Hacks*, 2005. [Online]. Available: https://graphics.stanford.edu/~seander/bithacks.html (visited on 05/29/2016).

[120] W. Yue, H. Takagi, and Y. Takahashi, *Advances in queueing theory and network applications*. Springer, 2009, ISBN: 978-0-387-09702-2.

[121] J. N. Daigle, *Queueing theory with applications to packet telecommunication*. Springer Science & Business Media, 2005.

[122] R. L. Cruz, “A calculus for network delay. I. Network elements in isolation”, *IEEE Transactions on information theory*, vol. 37, no. 1, pp. 114–131, 1991. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=61109 (visited on 09/25/2016).

[123] P. Thiran and J. Y. Le Boudec, *Network Calculus*. Springer, 2001, ISBN: 3-540-42184-X.

[124] Y. Jiang, “A basic stochastic network calculus”, *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 123–134, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1159929 (visited on 10/22/2016).

[125] J. B. Schmitt, F. A. Zdarsky, and L. Thiele, “A Comprehensive Worst-Case Calculus for Wireless Sensor Networks with In-Network Processing”, IEEE, Dec. 2007, pp. 193–202, ISBN: 978-0-7695-3062-8. DOI: 10.1109/RTSS.2007.17. [Online]. Available: http://ieeexplore.ieee.org/document/4408304/ (visited on 10/09/2016).

[126] M. Fidler and J. B. Schmitt, “On the way to a distributed systems calculus: An end-to-end network calculus with data scaling”, in *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, ACM, 2006, pp. 287–298. [Online]. Available: http://dl.acm.org/citation.cfm?id=1140310 (visited on 10/09/2016).

[127] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, “Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch...”, in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, IEEE, 2008, pp. 1669–1677. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4509823 (visited on 09/25/2016).

[128] S. Bondorf and J. B. Schmitt, “The DiscoDNC v2: a comprehensive tool for deterministic network calculus”, in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 44–49. [Online]. Available: http://dl.acm.org/citation.cfm?id=2747659 (visited on 10/16/2016).

[129] S. Bondorf and J. Schmitt, “Improving cross-traffic bounds in feed-forward networks–there is a job for everyone”, in *International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, Springer, 2016, pp. 9–24.

[130] L. Lenzini, E. Mingozzi, and G. Stea, “End-to-end delay bounds in FIFO-multiplexing tandems”, in *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, p. 61. [Online]. Available: http://dl.acm.org/citation.cfm?id=1345341 (visited on 10/16/2016).

[131] S. Bondorf, P. Nikolaus, and J. B. Schmitt, “Delay Bounds in Feed-Forward Networks-A Fast and Accurate Network Calculus Solution”, *arXiv preprint arXiv:1603.02094*, 2016. [Online]. Available: http://arxiv.org/abs/1603.02094 (visited on 10/16/2016).

[132] A. Imamoto and B. Tang, “A Recursive Descent Algorithm for Finding the Optimal Minimax Piecewise Linear Approximation of Convex Functions”, in *World Congress on Engineering and Computer Science 2008, WCECS ’08. Advances in Electrical and Electronics Engineering - IAENG Special Edition of the*, Oct. 2008, pp. 287–293. DOI: 10.1109/WCECS.2008.42.

[133] J. Vandewalle, “On the calculation of the piecewise linear approximation to a discrete function”, *IEEE Transactions on computers*, vol. 24, no. 8, pp. 843–846, 1975.

[134] Mentor Graphics, *Questa advanced simulator v10.2*, 1991 – 2013. [Online]. Available: https://www.mentor.com/products/fv/questa/ (visited on 05/29/2016).

[135] Potential Ventures, *Coroutine Co-simulation Test Bench*, Nov. 2016. [Online]. Available: http://potential.ventures/cocotb/.

[136] M. Kreider, *Wishbone for cocotb*, Nov. 2015. [Online]. Available: https://github.com/potentialventures/cocotb/tree/wishbone.

[137] J. Hoffmann, “PEXARIA5 General Description”, Tech. Rep., Oct. 2013. [Online]. Available: https://www.gsi.de/fileadmin/EE/Module/EXPLODER/pexaria5_14.pdf (visited on 01/11/2016).

[138] ——, “EXPLODER3 preliminary specification”, Tech. Rep., Aug. 2013. [Online]. Available: https://www.gsi.de/fileadmin/EE/Module/EXPLODER/exploder3_v4.pdf (visited on 01/11/2016).

[139] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, “It’s Time for Low Latency.”, in *HotOS*, vol. 13, 2011, pp. 11–11.

[140] J. Bai and C. Prados, “WR Switch under RFC 2889 Test”, Tech. Rep., Jan. 2016. [Online]. Available: https://www-acc.gsi.de/wiki/pub/Timing/TimingSystemDocumentsReportsAndMeasurements/WR_Switch_RFC_2889.pdf (visited on 01/11/2016).

[141] D. Beck, “’Torture’ Report about GMT with Debian on PC and SL6/CentOS 7 on SCU3”, Tech. Rep., Jun. 2016. [Online]. Available: https://www-acc.gsi.de/wiki/Timing/TimingSystemDocumentsRep201607222 (visited on 03/11/2016).

[142] Open Hardware Repository (OHWR), *6th White Rabbit Workshop*, 2002012. [Online]. Available: http://www.ohwr.org/projects/white-rabbit/wiki/Mar2012Meeting (visited on 05/29/2016).

[143] International Particle Accelerator Conference, “Status of the FAIR proton LINAC”, English, OCLC: 958149341, Geneva: JACoW, 2015, ISBN: 978-3-95450-168-7. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/IPAC2015/ (visited on 01/01/2017).

[144] FAIR, *First ring for FAIR*, Aug. 2016. [Online]. Available: http://www.fair-center.eu/en/news-events/news-view/article/first-ring-for-fair.html (visited on 11/15/2016).

[145] D. Beck, *First beam with WR based timing system at GSI*, WR development mailing list, Dec. 2015. [Online]. Available: http://lists.ohwr.org/sympa/arc/white-rabbit-dev/2015-06/msg00018.html (visited on 02/11/2016).

[146] M. Lipiński, T. Włostowski, J. Serrano, P. Alvarez, J. D. G. Cobas, A. Rubini, and P. Moreira, “Performance results of the first White Rabbit installation for CNGS time transfer”, in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, IEEE, 2012, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6336610 (visited on 12/31/2016).

[147] J. Milnes, N. B. Ayed, F. Dhalla, G. Fishpool, J. Hill, I. Katramados, R. Martin, G. Naylor, T. O’Gorman, R. Scannell, and others, “MAST Upgrade–Construction Status”, *Fusion Engineering and Design*, vol. 96, pp. 42–47, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092037961500160X (visited on 12/30/2016).

[148] H. Li and G. Gong, “Prototype of White Rabbit network in LHAASO”, in *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, 2015. [Online]. Available: http://icalepcs.synchrotron.org.au/papers/wepgf126.pdf (visited on 12/30/2016).

[149] GNU Project, *Octave Scientific Programming Language*, 1992 – 2017. [Online]. Available: https://www.gnu.org/software/octave/ (visited on 05/29/2016).