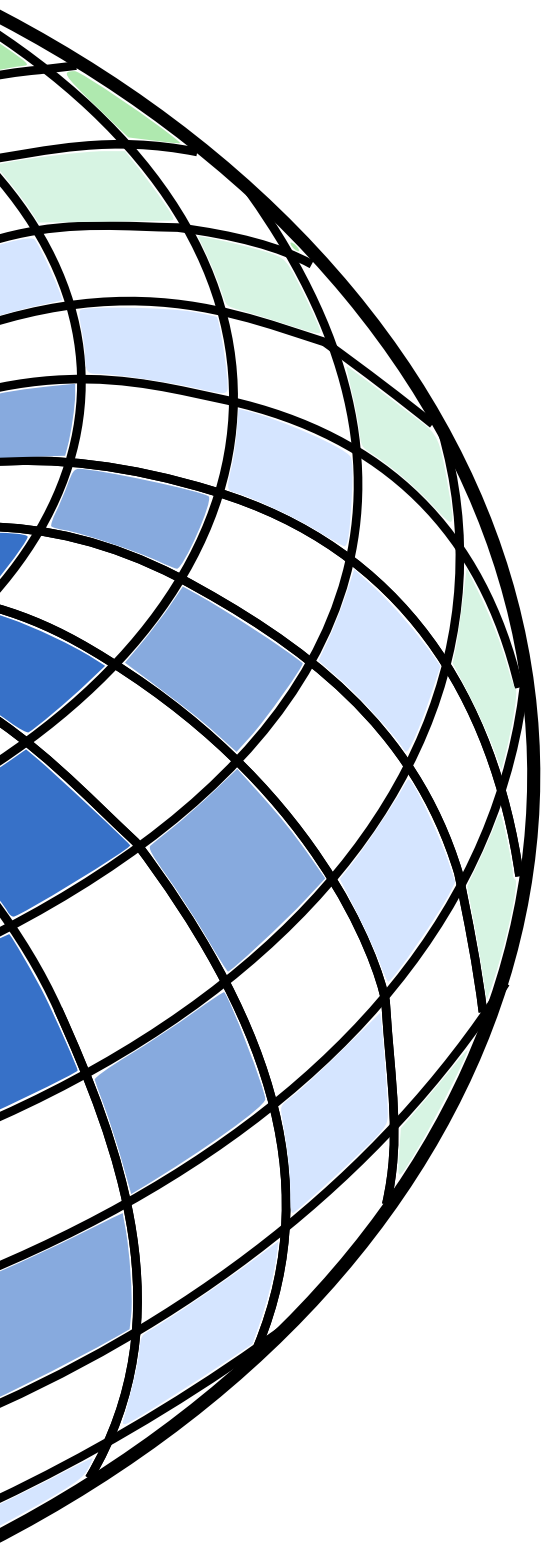


The QPACE Supercomputer

Renormalization of Dynamical CI Fermions

Axial Charges of Excited Nucleons

Dissertation

zur Erlangung des Doktorgrades der
Naturwissenschaften (Dr. rer. nat.) der
Naturwissenschaftlichen Fakultät II (Physik)
der Universität Regensburg

vorgelegt von

Thilo Maurer

aus Regensburg

Mai 2011

Das Promotions-Gesuch wurde am 15.03.2011 eingereicht.
Die Promotions-Kommission tagte am 11.05.2011.
Das Promotions-Kolloquium fand am 29.07.2011 statt.
Die Promotions-Arbeit wurde angeleitet von Prof. Dr. Andreas Schäfer.

Prüfungsausschuss:

Vorsitzender	Prof. Dr. Franz Gießibl
1. Gutachter	Prof. Dr. Andreas Schäfer
2. Gutachter	Prof. Dr. Tilo Wettig
weiterer Prüfer	Prof. Dr. Jaroslav Fabian

Contents

Preface

his PhD Thesis consits of three parts:

- The first part is about the QCD Parallel Computing on the CBE (QPACE) supercomputer. We present a very short introduction to supercomputing in chapter 1 and continue to describe the QPACE special-purpose supercomputer in chapter 2. The main topic of the first part of this thesis is the QPACE Network Processor. We describe its purpose, interfaces, architechture and implementation in chapter 3. The QPACE Ethernet logic, a significant part of the network processor, is described and documented in great detail in chapter 4. Other relevant companion logic within the network processor is documented in chapter 5.

- The second part is about the axial charge of excited nucleons. In chapter 7 we present a short introduction to the strong interaction, as well as its theoretical description model, known as Quantum Chromodynamics (QCD). We present an introduction to Lattice QCD (LQCD), a method to numerical simulations of QCD, in chapter 8 along with detailed description of the methods we employ for extracting masses and matrix elements form such simulations. The main results are provided in chapter 9, where we present our numerical setup, i.e., the available lattices, quark masses as well as other important parameters for our calculation. We further present results for the masses and axial charge of excited nucleons.

- The third part of this thesis is about renormalization of matrix elements obtained from LQCD. In chapter 10 we explain the necessity of renormalization and present a short review of a currently available renormalization method. We finally provide results for the lattices employed in chapter 11.

Acronyms

n this work we make heavy use of the following acronyms:

AdS/CFT	Anti de-Sitter / Conformal Field Theory ... 88
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit ... 11
AUI	Attachment Unit Interface
AWI	Axial Ward Identity ... 121
BRAM	Block RAM ... 11
CBE	Cell Broadband Engine ... 7
CDT-DCR	Clock Domain Transition DCR ... 8
CI	Chirally Improved ... 106
COR	Clear-On-Read ... 25
CPLD	Complex Programmable Logic Device ... 7
CPU	Central Processing Unit ... 8
CRC	Cyclic Redundancy Check ... 17
DCR	Device Control Register ... 8
DDR	Double Data Rate ... 41
DIS	Deep Inelastic Scattering ... 91
DMA	Direct Memory Access ... 6
DSP	Digital Signal Processing ... 18
EIB	Element Interconnect Bus ... 15
EMAC	Ethernet MAC ... 11
EMI	Electro-Magnetic Interference ... 22
FCS	Frame Check Sequence ... 41
FF	Flip Flop ... 18
FIFO	First In First Out ... 19
FLOP	Floating Point Operation
FLOP/s	FLOP per second ... 4
FPGA	Field Programmable Gate Array ... 6
FSM	Finite State Machine ... 22
FWFT	First Word Fall Through ... 35

NWP Network Processor . . . 7
OSI Open Systems Interconnection . . . 21
OWC Outbound-Write Controller . . . 15
PCB Printed Circuit Board . . . 12
PhD Doctor of Philosophy . . . 9
PHY Physical Tranceiver . . . 7
PPE Power Processor Element . . . 15
QCD Quantum Chromodynamics . . . v
QED Quantum Electrodynamics . . . 91
QFT Quantum Field Theory . . . 91
QPACE QCD Parallel Computing on the CBE . . . v
RAM Random Access Memory
RGI Renormalization Group Invariant . . . 159
RGMII Reduced Gigabit MII . . . 12
RI'-MOM Modified RI-MOM . . . 150
RI-MOM Regularization Independent MOM . . . 150
RMS Root-Mean-Square . . . 122
RRI'-MOM Reduced RI'-MOM . . . 155
RX Receive . . . 25
SFB Sonderforschungsbereich . . . 185
SGB Slave GBIF . . . 46
SLOF Slim-Line Open Firmware . . . 8
SPE Synergistic Processing Element . . . 15
SPI Serial Peripheral Interface . . . 12
SP Service Processor . . . 12
TCP Transmission Control Protocol
TP Twisted Pair . . . 21
TX Transmit . . . 25
UART Universal Asynchronous Receiver Transmitter . . . 8
UDP User Datagram Protocol . . . 15
VHDL VHSIC HDL . . . 9
VHSIC Very-High-Speed Integrated Circuits
WAN Wide Area Network . . . 21
WLAN Wireless LAN . . . 21
XAUI 10 Gigabit AUI . . . 12
XGMII 10 Gigabit MII . . . 12

Part I

The QPACE Supercomputer

Chapter 1

Introduction to Supercomputing

The term ”supercomputer” was coined around 1929 by the ”New York World” newspaper [1], referring to tabulating machines [2] manufactured by International Business Machines (IBM).

A supercomputer [3] is a computer that is at the frontline of current processing capacity, particularly speed of calculation. The first commercial supercomputers were introduced in the 1960s and were designed primarily by Seymour Cray. These machines led the market until 1980, where competition started to grow. Today, supercomputers are typically one-of-a-kind custom designs produced by companies such as IBM and Hewlett-Packard (HP), who had purchased many of the 1980s companies to gain their experience.

Today, supercomputers are used for highly calculation-intensive tasks such as problems involving weather forecasting, climate research (including research into global warming), molecular modeling (computing the structures and properties of chemical compounds, biological macromolecules, polymers, and crystals), physical simulations (such as simulation of airplanes in wind tunnels, simulation of the detonation of nuclear weapons, research into nuclear fusion, tests of the current understanding of the strong interaction), cryptanalysis and similar. Major universities, military agencies and scientific research laboratories are heavy users, see fig. 1.1.

Figure 1.1: Photo of a IBM BlueGene/P Supercomputer at Argonne National Laboratory [4].

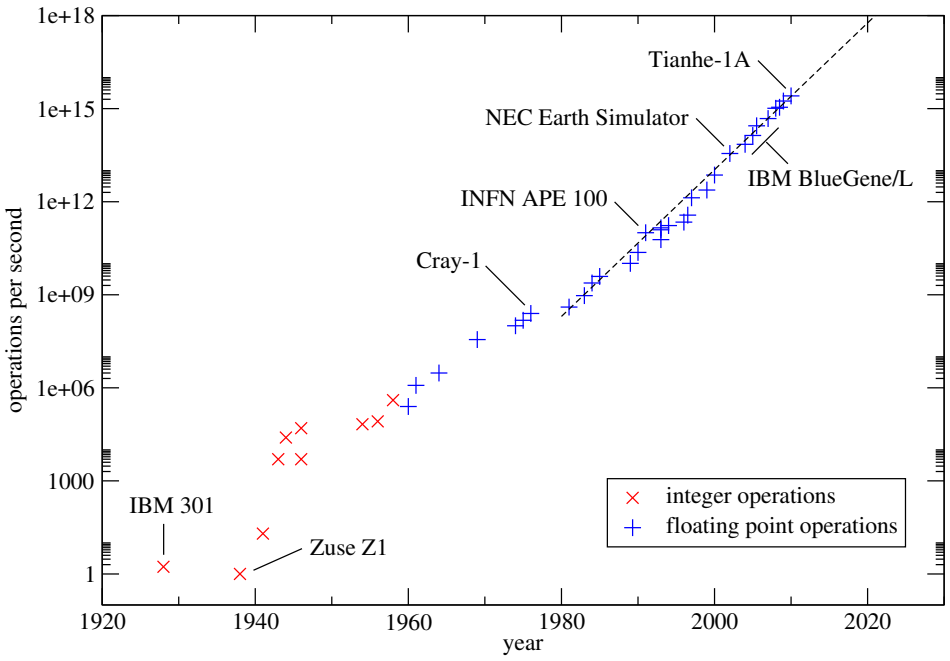


Figure 1.2: Plot of operations per second of the fastest supercomputer installation available each year. Data points taken from [3].

Computing technology has evolved extraordinary, see figure 1.2. In general, the speed of a supercomputer is measured in FLOP per second (FLOP/s). In comparison to current supercomputers, the tabulators only invented 70 years ago can be seen as truly ancient. Since inception, computational speed has risen exponentially by a factor of 10^{15}. As of March 2011 [5] the fastest supercomputer in the world is the "Tianhe-1A" at the National Supercomputing Center in Tianjin, China. It consists of more than 21000 computational units, at a total of 2.5 PFLOP/s. Among the most powerful supercomputer available today, we see an increase by a factor 500 every 10 years. Current predictions purport the first installation of a supercomputer with a computational speed of 10^{18} FLOP/s by the year 2020. The future of supercomputers may be very 'bright'. Current plans involve bioelectrical/neuronal simulations of the human brain at 10^{20} FLOP/s [6, 7] or fine grained global climate simulations at 10^{21} FLOP/s [8].

Supercomputers are one of the driving forces in cutting edge research. Research institutions and universities worldwide have greatly contributed to the development of these machines. Often, the crave to advancements in research drives the development of special-purpose supercomputers, i.e., supercomputers with a hardware architecture dedicated to a single problem. Among the most prominent one can find [3]

- Belle, Deep Blue, and Hydra, for playing chess,
- GRAPE, for astrophysics and molecular dynamics,
- Deep Crack, for breaking the DES cipher,
- MDGRAPE-3, for protein structure computation,
- Anton, for simulating molecular dynamics,
- Reconfigurable computing machines, for example JANUS for simulation of Spin-Glass systems,
- APE series, and QPACE, for simulations of the strong interaction.

Chapter 2

The QPACE Supercomputer

After performance models and hardware benchmarks [9, 10] indicated that the Broadband Engine Processors are a promising option for LQCD, the QPACE project [11] set out to leverage the power of the PowerX-Cell 8i processor by developing, building and deploying a massively parallel computer, optimized for LQCD calculations. We designed a supercomputer with an aggregate performance of $\mathcal{O}(200)$ TFLOP/s peak or $\mathcal{O}(50)$ TFLOP/s sustained (double precision) and an excessively low power consumption at 773 MFLOP/Joule with Linpack.

Custom designed computers for use of simulation of theoretical physics have a long history [12, 13]. In the case of LQCD, an important research field in modern particle physics, interest was (and is) especially high [14], since progress is mainly limited by availability of computing resources [15]. The success of QPACE emerged through a number of key features:

- Node Cards with IBM PowerXCell 8i and custom-designed Network Processor Field Programmable Gate Array (FPGA)
- Tight coupling of processing nodes through a custom network: nearest-neighbor 3D-torus network, 6 GByte/s communication bandwidth per node, remote Local Store to Local Store Direct Memory Access (DMA) communication, low latency, in-operation software-partitionable
- All-to-All switched Gigabit Ethernet network
- Global Signal Tree: evaluation of global conditions and synchronization
- Tightly packed, two fronted rack design
- 256 Node Cards per rack, 4GByte of memory each node
- 25.6 (51.2) TFLOP/s double (single) precision each rack
- Efficient, low-cost water-cooling solution
- Typical power consumption of 21-27 kW per rack
- Capability machine (easily scalable architecture)

The project is based on a broad collaboration of academic and industrial partners. It has made valuable progress during its first year [16] and entered its final deployment phase at the end of the second year [17]. QPACE is now heavily used for simulations of LQCD [18] and a number of other exciting field of research that can make valuable progress at the forefront of computational power, for example [19].

We built on the excellent power efficiency of the PowerX-Cell 8i processor, adding a completely new cooling system. A central element of this new technology is the Cold Plate. The Cold Plate provides a strong heat reservoir to the Node Cards which are thermally coupled to both sides of the plate. It is part of a closed cold water circuit, chilled by an external heat exchanger. We show a picture of a Cold Plate in operation in figure 2.1. Together with the use of carefully selected power supplies, hand-tuned voltage regulation for every single nodecard, all the little details that we optimized cumulated in a power efficiency that has been never been achieved before. As of November 2009 [20], QPACE was the most power efficient super computer in the world and at #1 of the Green 500 list. These advances were so dramatic that QPACE was able to hold it's pole position for a full year [21].

Figure 2.1: Photo of a fully populated QPACE Cold-Plate. The plate is in the very center of the picture. There are 32 Node Cards as well as 2 Root Cards attached to it. On the left one can see the piping that provides cold water to the Cold Plates. On the right there are three power supplies, one for redundancy.

2.1 The QPACE Node Card

The Node Card is the heart of QPACE Supercomputer, see figure 2.2. On each Node Card there is a PowerXCell 8i Processor providing 100 GFlop/s of computational power. It is backed by 4 GByte of dual-channel DDR2-800 memory providing a peak bandwidth of 25.6 GByte/s. The Cell Broadband Engine (CBE) processor interfaces to the Network Processor (NWP), a FPGA (a Xilinx Virtex-5 LX-110T) whose logic was designed to work as a Southbridge to the CBE. The Network Processor interfaces to six Torus Network transceivers as well as an Ethernet transceiver. The physical lines are routed through special connectors onto QPACE Back Planes, which also provide power to Node Cards. The Node Cards contain many more auxiliary devices, like voltage regulators, Ethernet Magnetics, as well as a Service Processor and a companion Complex Programmable Logic Device (CPLD) to manage the infrastructure for the CBE processor. A more thorough description can be found in [17].

2.2 My Role in QPACE

QPACE is built up from a huge number of components. I was responsible for design, development, testing and evaluation of a number of them. Most notably I was responsible for everything connected to Ethernet on the QPACE Node Card. This includes

- Selection of a Gigabit Ethernet Physical Tranceiver (PHY) and matching Magnetics,
- Electric circuitry and connections on the Node Card,

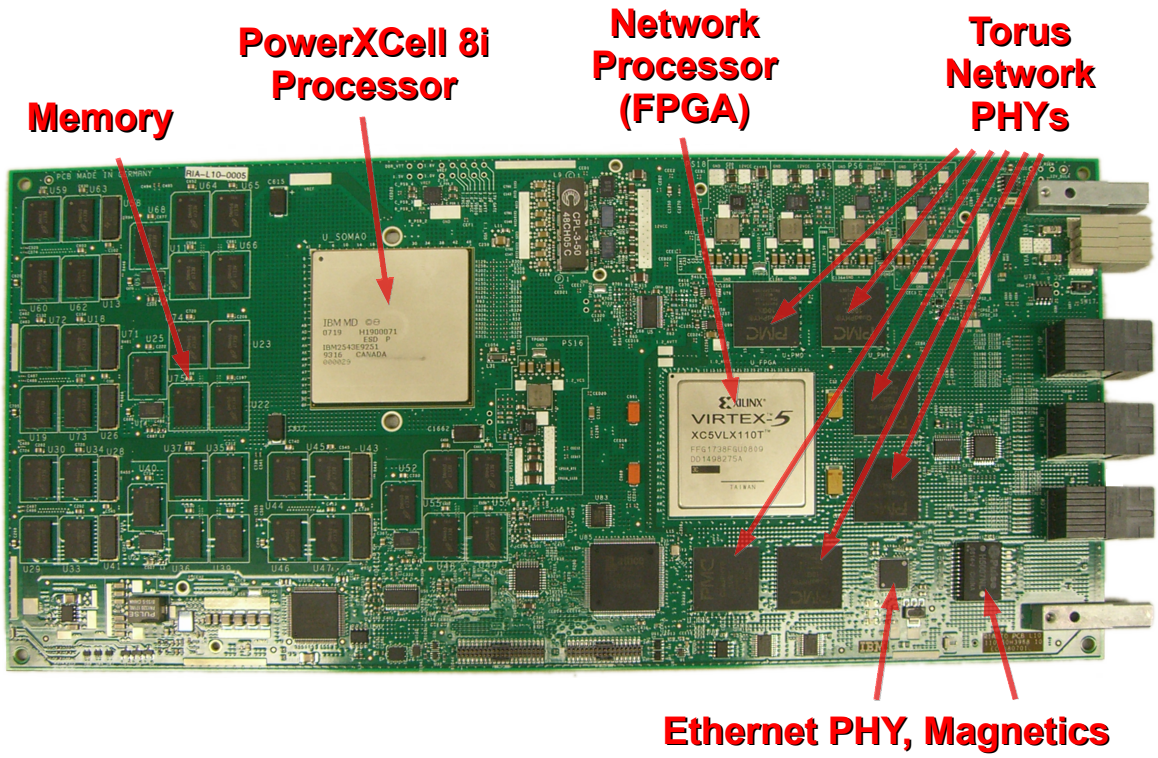


Figure 2.2: Photo of a QPACE Node Card from the top.

- Logic within the FPGA to receive and transmit frames, i.e., management of datagram transfers between System Bus and PHY,
- Slim-Line Open Firmware (SLOF) driver, in order to enable network booting,
- fully featured Linux kernel driver yielding high bandwidth and low latency, kernel patches,
- extensive testing of any component involved.

Apart from these there are many more logic components within the NWP I designed:

- MDIO Clause 45 (MDIO45) core, a common bus used to operate 10 Gigabit Transceivers,
- Global Signals (GS) core,
- an extension to the Inbound-Write Controller (IWC) handling its exceptions and its debugging,
- an extension to the Universal Asynchronous Receiver Transmitters (UARTs) enabling retrieval of UART logs through the Device Control Register (DCR) Backdoor for post Central Processing Unit (CPU)-mortem debugging, and software necessary for operation,
- Data Monitor to enable triggered or selective data stream recording, enabling in-operation-monitoring and debugging, and software necessary for operation,
- Reset Controller,
- Clock Domain Transition DCR (CDT-DCR) Slave: A variant of a traditional DCR Slave which enables access to registers driven asynchronously to the DCR Master clock,

- Most timing constraints within and between logic cores.

Further I designed the following software components

- FPGA Flash dumper for Linux,
- FPGA synthesize, place and route, as well as bitstream generation scripts from Subversion Repository,
- logic simulation testbenches for most of the logic I designed,
- Bidirectional Ethernet Frame Integrity Validation Software (node to node)

I was further involved in the design and execution of:

- RootCard Case,
- SuperRootCard Case,
- Procurement of parts and equipment for PreIntegration, a phase in the assembly of a QPACE rack that fully equips bare-bone racks with all components (less NodeCards),
- Assembly and Packaging of QPACE Racks, during PreIntegration and Acceptance Tests.

Altogether I spent 2 years of full time work on QPACE during my work on my Doctor of Philosophy (PhD) and produced on the order of 5000 lines of C, 16000 lines of VHSIC HDL (VHDL) and 2000 lines of shell code, that would add another 450 pages to this document.

Chapter 3

The QPACE Network Processor

The NWP is designed to work as a Southbridge to the CBE. For the implementation of the NWP we relied on the significant progress in FPGA technologies during recent years.

3.1 Field Programmable Gate Arrays

An FPGA is an electronic device that was developed to provide large user-configurable logic circuits. The devices may be configured after manufacturing, hence 'field-programmable'. The logic is typically specified by a Hardware Description Language (HDL). FPGAs can be used to perform any logic operations that Application-Specific Integrated Circuits (ASICs) can perform. The benefit of using FPGAs is that they allow to develop and test logics within a reasonably short amount of time and that Non-Recurring Engineering (NRE) costs can be kept low. Hardware can be debugged in situ which can help to reduce the time required for logic design and integration. Some of the disadvantages of FPGAs are the typically high prices of the devices and the relatively high power consumption. For the physical layer of our network we decided to rely on commercial standards (here: 10 Gigabit Ethernet) that are well established and for which well-tested and relatively cheap PHYs are available. Using such hardware allows us to move critical logic resources out of the FPGA and to obtain the signal quality and strength required in our design.

Most available FPGAs are built up from basic elements called "slices", interconnected using "switch matrices". They also provide other primitives like Block RAMs (BRAMs), Ethernet MACs (EMACs), processor cores, high-speed transceivers, etc. A slice is made up from a number of Flip-Flops, Look-Up Tables (LUTs) and Multiplexers. A Flip-Flop is a device that

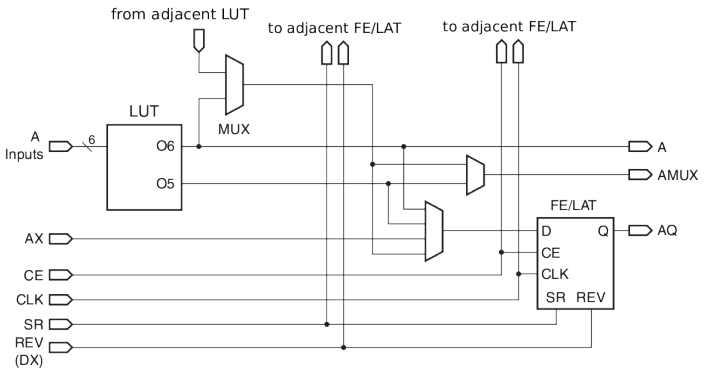


Figure 3.1: Typical basic logic elements within an Xilinx FPGA. From [22].

is able to store exactly 1 bit of information. Each clock period in CLK (notation see figure 3.1), it takes data from its input D and puts the information onto its output Q. Its input D may stem from adjacent logic or a LUT. A LUT is a device that performs some user definable combinatorial logic on its inputs. For our device (see later), we have 6 inputs to a LUT. This is an important number since designing the logic such that combinatorial logic is cleverly spread over adjacent slices may dramatically decrease logic footprint. Of course, there are sets/resets, as well as unregistered outputs A or AMUX, etc. Most inputs of a slice may connect to output of another slice. In effect, data hops from a Flip-Flop to another Flip-Flop each cycle, kind of flowing through the logic and combinatorially processed. It is up to the user to specify how the chip will work. The user needs to set up a number of these primitives and interconnect them in a way that yields the functionality desired. The complexity of the logic, for example the number of interconnects or the footprint, limits the frequencies of the clock that can be used to drive the Flip-Flops because of propagation delays within combinatoric elements or muxers and so on. Like with programming languages, there are more abstract ways to formulate the functionality like the bare-bones approach described. Commonly used HDLs are Verilog and VHDL. For QPACE we went with VHDL, since it is well supported by the tools provided by the vendor of the physical device chosen.

3.2 Architectural Overview

The FPGA thus carries logic designed to provide access to a high bandwidth and low latency nearest neighbor network. Its main interfaces to the system as depicted in figure 3.2 are:

- 2 Rambus FlexIO links running at 2.0 GHz. This provides a bandwidth of 40Gbps in each direction of communication to the CBE.
- 6 10 Gigabit MII (XGMII) running at 250 MHz, each transferring 32 bit each cycle. These standard interfaces are used to saturate the 10 Gigabit AUI (XAUI) PHYs used for the 3D torus network.
- One Reduced Gigabit MII (RGMII) interface running at 250 MHz (transferring 4 bit each cycle) to communicate over 1 Gbps Ethernet using an external Ethernet 1000BASE-T physical transceiver.
- One 2 line custom interface to enable communication through the Global Signal Tree.

There are a number of smaller interfaces that enable monitoring, control and debugging of the FPGA and CBE.

- Management Data I/O (MDIO) interfaces to all of the network transceivers, used to setup the PHYs, for example to switch between primary and redundant links in the 3D torus network.
- UART interface to the CBE.
- UART interface to the Service Processor (SP), used for FlexIO training.
- Serial Peripheral Interface (SPI) to the SP, supporting FlexIO training and providing access to the DCR Bus.
- SPI interface to a Flash Memory. The memory stores the NWP Image itself as well as the CBE Firmware.
- Joint Test Action Group (JTAG) interface to debug headers on the Printed Circuit Board (PCB), used for in-field programming and debugging.

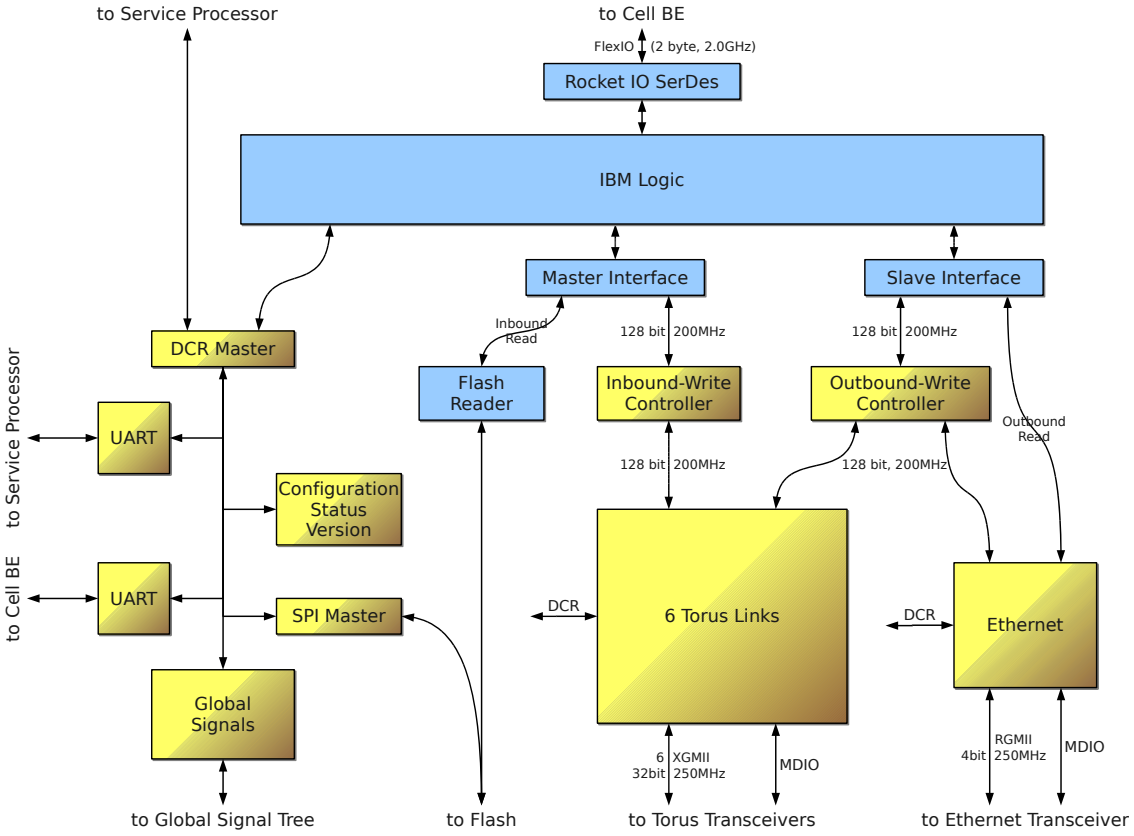


Figure 3.2: Simplified overview of logic and data-paths within FPGA, blue: IBM, yellow: Academic

3.3 FlexIO Interface

The system interface used in CBE is a Rambus design, known as FlexIO. NWP logic was designed to attach to the CBE using this interface. The FlexIO interface is organized into 12 lanes, each lane being an unidirectional 8-bit wide point-to-point path. For the PowerX-Cell 8i there are 5 inbound and 5 outbound point-to-point paths. Among these, 4 inbound and 4 outbound lanes are supporting memory coherency. The FlexIO interface can be clocked independently, typically at 3.2 GHz.

For QPACE we currently use a total of 4 lanes at 2.0 GHz: 2 inbound and 2 outbound. This provides a raw bandwidth of 40Gbps in each direction of communication to the CBE. The efficiency of the torus network links is much higher than that of the FlexIO link. Therefore the torus network can be viewed as a rather ideal adapter to connect CBEs. On the FPGA end of the FlexIO link, Xilinx Virtex-5 Rocket-IO GTP Low-Power Transceivers are used to serialize and deserialize the data using a special 10B/10B mode [23]. Although the Virtex-5 was the only FPGA available to interface to the FlexIO, there is no 100% compatibility between the Rambus FlexIO and the Xilinx GTPs, such that training of the interface has proven difficult.

From there the data is decoded and fed into a logic provided by IBM that exposes a DCR master and two proprietary buffered high-speed interfaces. One of them is a master interface that executes transaction where, effectively, the CBE is master. Therefore its transactions are called inbound to the FPGA. The second one is a slave interface. In this case the entities attached to it are master on the system bus. Its transactions are thus called outbound from

the FPGA. These interfaces have a width of 128 bit and currently operate at 200 MHz, a frequency that has been fixed after various optimizations.

3.4 Device Control Register Bus

The DCR Bus is a bus designed by IBM that is widely used in PowerPC architectures to control and monitor any kind of device. A transaction on a DCR Bus is done by a simple 4-phase handshake protocol, transferring one word at a time. A complete architecture specification can be found in [24]. There are a total of three DCR Masters in our design. The first of these is provided by the IBM logic as described above. A second DCR Master was created to enable access to the DCR Bus from outside of a node in order to configure vital components or when the FPGA's connection to the CBE is down. Commands can be issued by the RootCard-Controller through the SP to the NWP. These two masters are finally arbitrated and fed into a final DCR Master that is the real master on the DCR Bus. The connections of the DCR Master depicted in Figure 3.2 are thus greatly simplified. Since the total number of registers instantiated is rather small, we reduced the default DCR address bus width from 32 to 16. This simplifies routing logic connections of the DCR chain within the FPGA.

3.5 UART to Service Processor

There is a UART connection of the NWP and the SP. This serial interface is mainly used by IBM to control and monitor the training of the FlexIO interface between the CBE and the NWP.

3.6 UART to RootCard

There is a UART connection of the NWP and the RootCard. This data path is used to access a serial console of the CBE and can be used for, beyond a regular Linux shell, accessing the Firmware console during boot. Further, it can often still be used to access the NodeCard when other network connections (e.g. Ethernet) fail.

3.7 Global Signals

The purpose of the GS and the corresponding Root Logic (outside the NWP) is to provide a fast and robust mechanism to communicate within partitions of a QPACE installation [25], in order to

- send an exception from any node to all other nodes within the same partition in order to stop the execution of an application program. This is done using a hardware interrupt to the CBE.
- combine logical conditions evaluated on each node into a single global condition, then return the result to each node, i.e., provide a global OR and a global AND. For example, this can be used to. implement "barriers" in the control flow

The GS core in the NWP is comprised of a few configuration registers that are accessible through the DCR Bus, a transmitter that can send 4 different signals: NOP, KILL, TRUE and FALSE, and a receiver that accepts incoming signals once they are sufficiently stable. During the development of the GS tree, focus was set to achieve a minimum delay through the network.

This was achieved by putting the validation stage into the FPGA receiver and omitting signal registration elsewhere if possible. This combines any intermediate time penalties from save signal registration or signal validation into a single easily optimizable position.

3.8 Inbound-Write Controller

The IWC is responsible for handling incoming write requests for torus communication. It decodes the target addresses and demultiplexes the incoming data into the 6 torus link buffers, supported by a single level address translation, while stopping or halting the transfer as long as buffers of any Torus link are almost full.

3.9 Outbound-Write Controller

The Outbound-Write Controller (OWC) is the master on the slave interface provided by IBM to push data to the CBE. It supports two types of transactions: The first type of transaction is a non-strictly ordered transfer of 128 bytes of data, while the second type is a strictly ordered transfer of 16 bytes of data that can for example be used for notifications. The OWC processes outstanding requests by its slaves (Torus and Ethernet) by round-robin priority arbitration. This type of arbitration selects the next transaction in a priority-less and starvation-free manner. The corresponding data is then fetched from the slave and multiplexed into the buffers of the slave interface and queued for transmission to the designated address.

3.10 Ethernet Core

The Gigabit Ethernet core, and the network it is attached to, has been designed to perform sufficiently well for booting, monitoring and disk Input/Output (I/O). The core pushes(fetches) data to(from) the CBE using outbound DMA transactions only, while it is controlled exclusively using DCR. Its interface to the Gigabit Ethernet PHY is standard: RGMII for data and MDIO for management. The core is build up from a Xilinx Virtex-5 Embedded Tri-Mode EMAC and two separate paths for receiving and transmitting data.

Evaluations performed within a QPACE rack have shown a node-to-node ping round-trip time of 50 µs. We employ “Iperf”, a commonly used network testing tool that can create TCP and User Datagram Protocol (UDP) data streams and measure the throughput of a network that is carrying them, and obtained a sustained TCP bandwidth of 990 Mbps in jumbo frame mode. In this mode the theoretical maximum bandwidth of Gigabit Ethernet is 0.9911 Gbps. We also employed ”NetPIPE”, a Network Protocol Independent Performance Evaluator and obtained the results depicted in figure 3.3. Bandwidth significantly benefits from jumbo frame mode for large message sizes, for the default mode we found it to be CPU (i.e., Power Processor Element (PPE)) bound.

3.11 Torus

The Torus Communication Network [26] is unlike the network implementation of existing CBE-based machines. In QPACE, communication proceeds directly between two Synergistic Processing Elements (SPEs) on adjacent CBEs. Data is passed from the Local Store (LS) of the sending SPE via the Element Interconnect Bus (EIB) to the I/O interface. The NWP sends the data via the appropriate link to the receiving NWP, where the data are directly

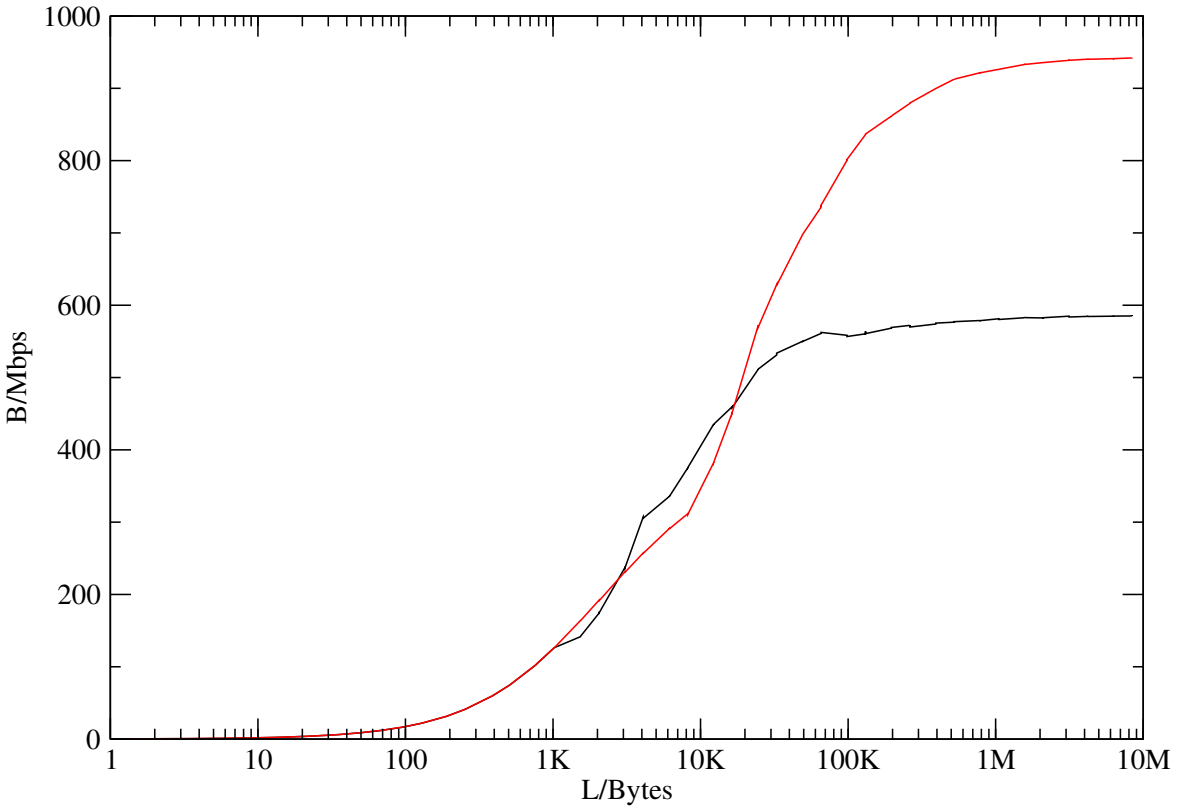


Figure 3.3: The bandwidth of the QPACE Ethernet Core with its Linux Driver on a QPACE Node Card with respect to the TCP message size. We measured the performance in two configurations. For the default Ethernet MTU size of MTU = 1500 Bytes (Maximum Transfer Unit), we obtain the black curve topping out around a bandwidth of 600 Mbps. In Jumbo Frame Mode at MTU = 9000 Bytes we obtained the red curve reaching transfer rates around the full Gigabit (1000 Mbps).

driven to the receiving SPE via the I/O interface and the EIB. For a number of applications, this implementation of networking has several advantages compared to other approaches:

- Datagrams are directly communicated between the processing elements rather than passed through secondary storage (i.e., Main Memory (MM) is not required for data transfer, such that the full memory bandwidth is available to the application).
- The transfer is entirely driven by the SPE DMA engine, no need for PPE intervention.
- The direct data paths support inter-processor communication with very low latency.

However, MM-to-MM or LS-to-MM communication is available as well and heavily used. Since the PowerXCell 8i is a multi-core processor, a communication protocol between two adjacent CBEs may allow for separate paths for each combination of sending and receiving SPE and PPE. For LQCD, and many other applications, only a subset of these data paths is required. To support independent communication amongst the SPEs of different CBEs, logical data paths are provided by the distinction of 8 *virtual channels* per link, and the corresponding number of autonomous buffers on the NWP. The channels allow for at least one sender/receiver pair of SPEs per link for any logical arrangement of $n_1 \times n_2 \times n_3 = 8$ of the SPEs in the 3D sub-volume.

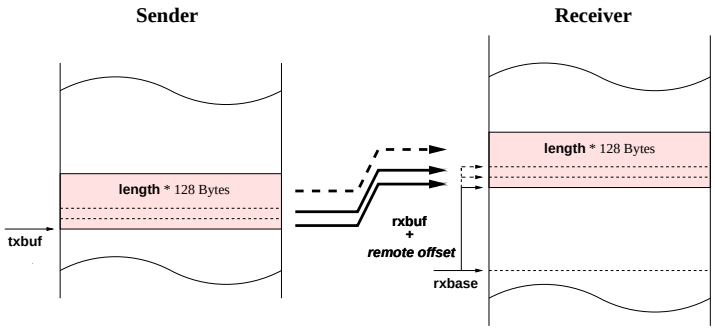


Figure 3.4: Addressing scheme for the torus communication on sending and receiving device for a single link and virtual channel. Packets of fixed size are copied, where the absolute destination address of each datagram is determined by combining base address, local offset and remote offset.

Our communication model is based on matched communication commands. Send commands on the sending node match receive commands on the receiving node. No hardware support is foreseen which would allow sender and receiver to negotiate, for example order and length of communication.

3.11.1 Transmit Path

The sending device addresses the buffer on the destination NWP according to the link and the channel. To relax the constrain of the strict ordering of the packets, a remote offset is assigned to each of the datagrams. The destination base address per link and channel is defined on the receiver side and must be uploaded (once) to the NWP before the data are moved into the device. This allows for any destination, for example local store or main memory. The receiving device defines the first address of the current receive buffer via a local offset. The destination address of a particular datagram, which is received on a particular link, is therefore determined by combining the channel's base address, the remote address and the local offset as indicated in Fig. 3.4.

Transfer size is restricted to multiples of 128 Bytes. This corresponds to the native DMA size on the CBE (on the EIB all data is partitioned into chunks of 128 Bytes). The link protocol is optimized for our needs. Each datagram transmitted on the torus network consists of a small 4-Byte header, 128 Bytes of payload followed by a 32-bit Cyclic Redundancy Check (CRC). The receiving network processor has to generate a 4-Byte feedback for each received datagram. If a CRC mismatch is detected or if the datagram cannot be stored in the receive buffer a not-acknowledge is returned and the sender will re-transmit the datagram.

3.11.2 Receive Path

Transport of received data into the CBE is controlled by credits. Once a datagram has been received, the network processor checks whether for the given link and virtual channel a credit is available. If such a credit is available a request is sent to a multi-level arbiter for granting access to the link towards the CBE. Once the credit has been consumed a notification message is sent to a dedicated address defined by the receiving device, indicating the successful completion of the data transfer. The receiving device, while waiting for the receive operation to complete, polls on this notification address.

3.12 Device and its Utilization

FPGAs are user-programmable hardware chips. They are configured with a logic circuit that specify how the chip will work. They provide the possibility to update the functionality at any time. This offers vast advantages in development and operation over for example ASICs that have high NRE cost and that may be very hard to debug. FPGAs are built up from basic elements called "slices" that are interconnected using "switch matrices". They also provide other primitives like BRAMs, EMACs, processor cores, high-speed transceivers, Digital Signal Processing (DSP) slices and more.

For the QPACE Network Processor we chose to use a Xilinx Virtex-5 LX-110T, a FPGA optimized for high-performance logic and low-power serial connectivity. It provides just enough serial transceivers to connect to the FlexIO interface of the CBE and just enough pins to accommodate the parallel interfaces of the 6 torus links. We chose the highest speed grade available and sufficient capacity to hold our logic.

Generating logic configurations (bitstreams) for an FPGA that meet the timing constraints becomes asymptotically difficult as the utilization ratio increases. Generally, when there is a high number of independent clocks, routing of a global clock tree within an FPGA is hard. Too many clocks lead to restrictions on logic distribution within the FPGA. Further, for each of the clock domains the internal frequencies and thus bandwidths are limited by the complexity of the logic.

As can be seen in table 3.1, for the QPACE NWP this is due to a number of reasons: Due to a high number of clocks used by the IBM logic, the Torus Network logic and the Ethernet logic, routing within a highly utilized FPGA was very time intensive and debugging sometimes tricky. First adding additional debug logic further increases the difficult to build bitstreams that meet timing. Worse, the addition of debug logic may mask certain issues like problems with cross-clock timing constraints or meta-stability, making such cases difficult to solve.

(a) Primitives

Resources	Usage	
Slices	16,029 of 17,280	92%
PINs	656 of 680	86%
LUT-FF Pairs	51,018 of 69,120	73%
Registers	38,212 of 69,120	55%
LUTs	36,939 of 69,120	53%
BRAM/FIFO	53 of 148	35%

(b) Logic

	Flip-Flops	LUTs	Ratio Flip Flop (FF)
Total	38212	36939	100%
IBM Logic	20225	16915	53%
Torus	13672	14252	36%
Ethernet	1537	894	4%
IWC	583	132	1.5%
OWC	446	642	1.2%

Table 3.1: Xilinx Virtex-5 LX-110T Utilization Ratios

3.13 Evaluation

One possibility to remedy the obstacle of overly high utilization ratios is to (re)write the logic for optimal device-specific primitive utilization. This, however, may not always be possible. Common reasons for this are a lack of time or manpower, or the fact that the logic has been originally written for a different device and needs to be reused. A second possibility is to lower the clock frequencies. Although this procedure diminishes bandwidths and increases latency, it may render the whole system usable while optimizations are still in progress, an advantage compared to competing technologies like ASICs. For the QPACE Network Processor we originally set our goal to a frequency of 2.5 GHz for the FlexIO interface, while we are currently operating at a smaller 2.0 GHz, but have verified that the hardware is capable of at least 3.0 GHz. The buffered bus interfaces provided by IBM are currently operating at 200 MHz.

Our current evaluations have shown, that we can sustain and torus communication bandwidth of 0.9 GByte/s per LS-to-LS link at a latency of 3 µs. This sub-optimal latency is mainly caused by a long delay between the start of the data move operation in the CBE and the data exiting the IBM logic towards the torus transmit First In First Outs (FIFOs). The latency between packet injection to network until packet leaving from network is actually ≈ 0.5 µs.

Altogether we profited heavily from the use of an FPGA as Network Processor. The possibility to redesign in late stages of system development enabled gradual improvements and in-operation bug-fixing. This led to an extremely short development time for the whole QPACE system by removing some of the hard dependencies on the Network Processor.

Chapter 4

QPACE Ethernet

Ethernet, standardized as Institute of Electrical and Electronics Engineers (IEEE) 802.3, see [27], is one of many technologies developed to enable computer networking, i.e., communication of computers with each other. It is based on an exchange of frames of data between computers. Originally developed to support Local Area Networks (LANs) over coaxial cables, the standard was later extended to support Wide Area Networks (WANs) using optical signaling, and even later extended to support wireless networking (IEEE 802.11, Wireless LANs (WLANs)). However these new modes mainly differ in the physical layer only, while the heart of the Ethernet standard has been essentially unchanged since inception, which had led to its wide acceptance and success. Today, the combination of the Twisted Pair (TP) versions of Ethernet for connecting computers to the network, along with the fiber optic versions for backbones, is the most widespread wired LAN technology. The standard defines many more wiring and signaling (and wireless) standards for the Physical Layer of the Open Systems Interconnection (OSI) networking model as well as a common addressing format and a variety of Medium Access Control procedures at the lower part of the Data Link Layer.

The OSI model [28] has been created by the International Organization for Standardization (ISO). It sub-divides a communications system into a hierarchy of elements called layers. A layer is a collection of functions that provide services to the layer above it and receives services from the layer below it. In each layer there an instance providing these functions. For example, a layer that provides error-free communications across a network provides the path needed by applications in higher layers, while it calls the next lower layer to send and receive packets. Two instances at one layer are connected by a horizontal connection on that layer, see figure 4.1. We list the layers the OSI model defines in table 4.1.

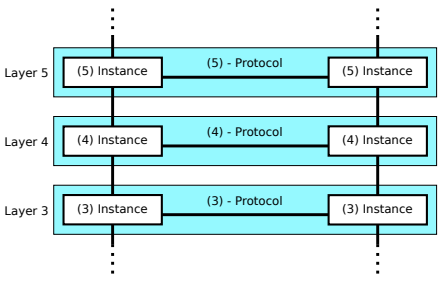


Figure 4.1: OSI Layering

As explained above, Ethernet is involved in the Data Link Layer and the Physical Layer. In figure 4.2 we show any sublayers involved. Within the QPACE project, the Xilinx FPGA provides an Tri-mode Gigabit EMAC hard core [29], the lower part of the link layer. The upper part of the link layer is the Logical Link Control (LLC). It connects the OSI Network Layer with the Media Access Controller (MAC). This part is not provided, since it connects a software layer, (in our case the Linux kernel network interface) with a hardware layer, in this case a Xilinx proprietary one. The LLC therefore bridges from software to hardware. This is generally the responsibility of a device driver. In most cases, there is also some more

Type	Infmtl. Unit	Name	Function	Examples
Host	Data	7. Application	Network process to application	http, ftp
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data	ssl, tls
		5. Session	Inter-host communication	
	Segments	4. Transport	End-to-end connections and reliability, flow control	TCP, UDP
Media	Packet	3. Network	Path determination and logical addressing	Internet Protocol (IP), Internet Control Message Protocol (ICMP)
	Frame	2. Data Link	Physical addressing	EMAC
	Bit	1. Physical	Media, signal and binary transmission	Ethernet, WLAN

Table 4.1: Layers defined within the OSI model.

hardware necessary to transfer data between hardware and software. This includes devices attaching to the system together with FIFOs buffering the data, as well as a number of control interfaces and registers. This (and a part of the reconciliation sublayer) is the job of the QPACE Ethernet Core, which we are going to describe in the next section. The Ethernet core provides a Media Independent Interface (MII) interface to the PHY. For QPACE we took the decision to employ a RGMII to Copper Transceiver. We used the RGMII interface (see also section 4.1.5.1.7) to save on FPGA pins and PCB footprint. For the physical inter-node connection we used 1000BASE-T (with a mixture of PCB and TP cabling) since it is a cheap and robust physical medium for signaling. It includes Ethernet Magnetics on each end of a link to provide electrical isolation and reduction of Electro-Magnetic Interference (EMI). The use of 1000BASE-T also enables the use of consumer network switching equipment.

4.1 The Ethernet Core

4.1.1 Notation and Conventions

Within this logic documentation we use the Most Significant Bit (MSB) 0 bit ordering. So within any binary representation of a value, we number the MSB with 0. The bit number is incremented by one for each subsequent bit position. For example the 32 bit representation of the number `32767=0x7FFF=0b0111111111111111`. The MSB is bit 0 with a value of 0, while the Least Significant Bit (LSB) is bit 31 with a value of 1.

Within each register lisiting we give the registers default value, as well as the registers mask. The default value is applies during reset. For each register bit there is a mask bit. If the mask bit is set, the register bit is writable. All register bits are readable.

Within each diagram of a Finite State Machine (FSM) we represent a state by en ellipse containing the name of the state. Unidirectional arrows connecting ellipses represent possible transitions between connected states. These are labeled with the event name effecting such a transition.

Within each block diagram of logic, we represent registers (i.e., FFs) by boxes containing

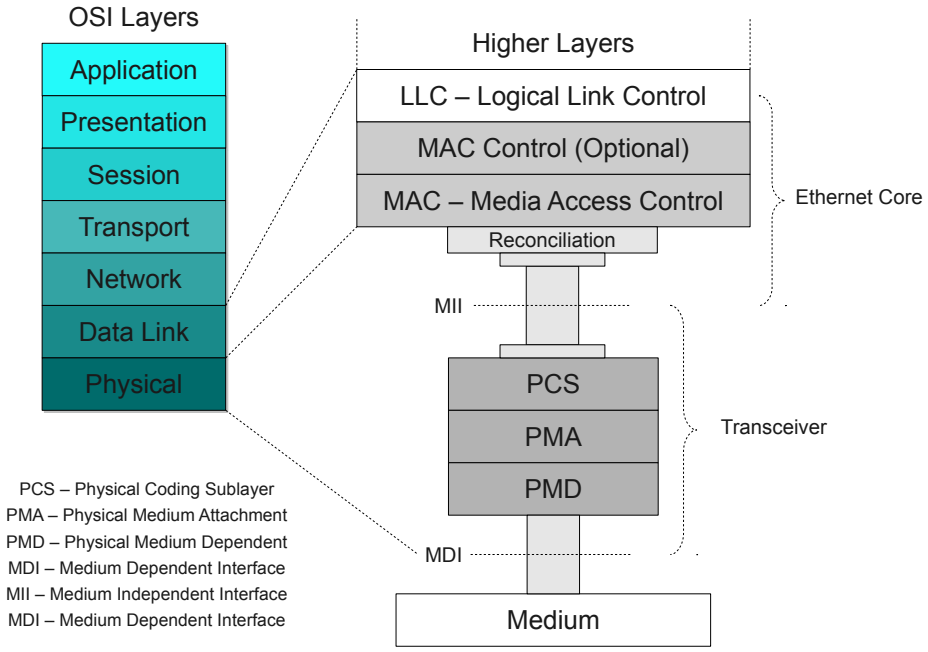


Figure 4.2: Gray layers indicate OSI layers that involve Ethernet Standard IEEE 802.3.

the name of the register. We represent signal names by ellipses containing the name of the signal. For the sake of clarity, we do not show combinatorial logic explicitly. However existence of combinatorial logic is obvious, where multiply arrows converge into a single register or signal. Block input and output signals labelled outside the logic block.

4.1.2 Overview

The Gigabit Ethernet core, and the network it is attached to, has been designed to perform sufficiently well for booting, monitoring and disk I/O. The core pushes (fetches) data to (from) the CBE using outbound DMA transactions only, while it is controlled exclusively using DCR. Its interface to the Gigabit Ethernet PHY is standard: RGMII for data and MDIO for management. The core is build up from a Xilinx Virtex-5 Embedded Tri-Mode EMAC and two separate paths for receiving and transmitting data. For the receive operation, frames are fed through the MAC and shifted into a dual port asynchronous FIFO. From there the data is chopped into chunks of optimal size for being queued into the Slave Interface of the System Bus in order to be pushed directly into main memory. The logic does not wait for the end of a frame to start its operation. This mechanism efficiently reduces latency by roughly the amount of time necessary to transmit the full frame from the Slave Interface into main memory. In a rare case of corrupt data reception, such a frame is marked as broken and will be overwritten in main memory by any subsequent frame. Only then the frame data is committed to the CBE using a notification mechanism similar to the one used by the torus network.

For transmit a similar mechanism can be employed: Once a frame is queued in main memory for transmit, data is fetched using outbound transactions by the core and shifted into a dual port asynchronous FIFO. Since the bandwidth of the system bus is much higher than the bandwidth of Gigabit Ethernet, the core does not strictly need to wait for the end of the frame to reside in the buffer, but can start forwarded the frame data immediately to the Gigabit Ethernet PHY through the EMAC. This reduces the time to transmission-start from the full frame transfer time over the system bus into the FPGA to its latency.

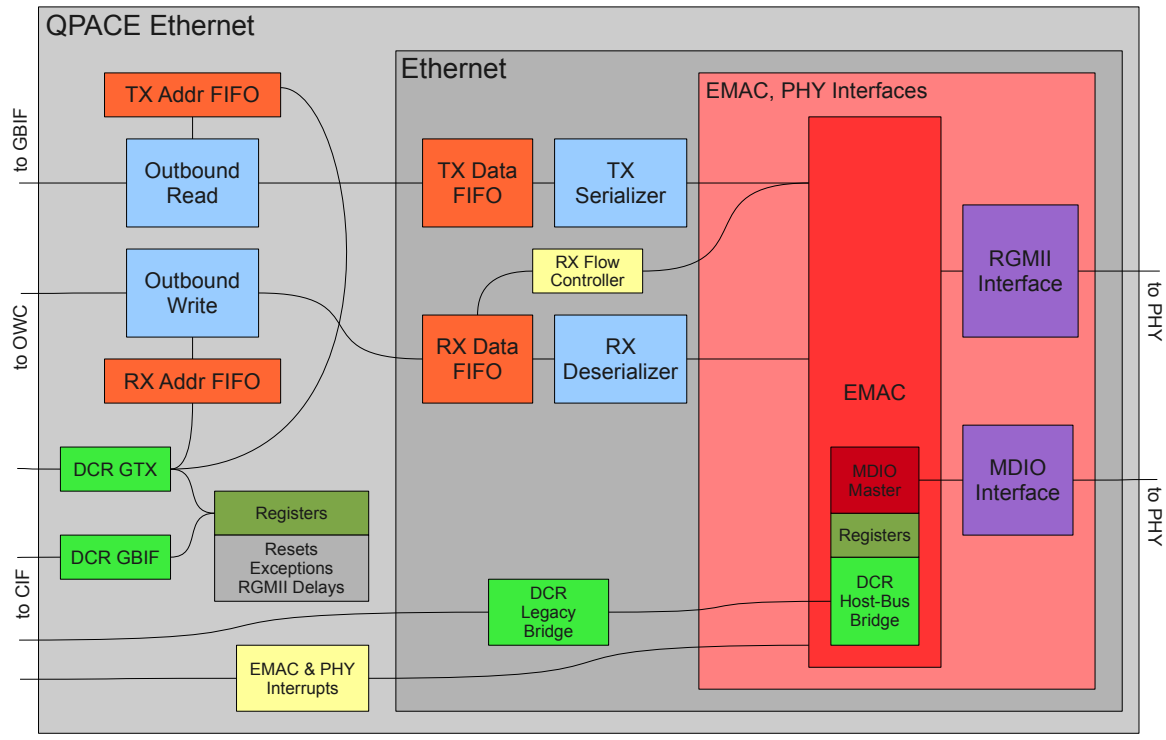


Figure 4.3: Vastly simplified block diagram of the QPACE Ethernet Core that includes most important logic blocks and its main connections with each other.

4.1.3 Device Control Register Interface

The Ethernet Core may be controlled through the DCR Bus. There is a total of 3 DCR slaves in the core, providing access to a number of control registers. These may be used to setup, configure and operate the core, as well as to provide feedback and statistics. Each of the registers obtains its reset value while the reset is active. For each of the registers, bits masked by its mask value can be changed, others are read only.

4.1.3.1 GBIF Slave

These control registers determine the behavior of the logic driven by the Global Bus Interface (GBIF) clock of the Ethernet Core.

Register Name	Address	Register Description
`ETH_Config`	`0x0030`	Default Value: `0xFFFFFFFF` Mask Value: `0x0000076F` This register holds the reset flags. Logic is in reset when the corresponding bit is asserted. Bit 21: Interrupt Logic Bit 22: Receive Address FIFO Bit 23: Transmit Address FIFO Bit 25: Outbound Read Logic Bit 26: Outbound Write Logic Bit 28: Receive Data FIFO Bit 29: Transmit Data FIFO

		Bit 30: Virtex 5 Trimode EMAC Bit 31: External Ethernet PHY
ETH_Exc	0x0031	Default Value: 0x00000000 Mask Value: 0x00000000 This register holds the exception flags. Bits become asserted after an exception event. All exceptions flags are Clear-On-Read (COR). Bit 10: Transmit (TX) address dropped due to Receive (RX) address buffer overflow. Bit 11: read error from GBIF Outbound Interface. Bit 12: read error from TX Data buffer due to buffer underflow. Bit 13: write error to TX Data buffer due to buffer overflow. Bit 14: TX retransmit signal from EMAC. Bit 15: TX underrun signal from EMAC. Bit 26: RX Address dropped due to RX Address buffer overflow. Bit 28: read error from RX data buffer due to buffer underflow. Bit 29: write error to RX Data buffer due to buffer overflow. Bit 31: frame was dropped due to RX buffer overflow.
ETH_RX_Config	0x0032	Default Value: 0x00010000 Mask Value: 0xFFFF0000 This register holds the size of the next control buffer to be N times 16 bytes, where N is the numeric value written to bits 8 to 15. Only pushed forward by a write to ETH_RX_Ctrl_Addr_Low.
ETH_RX_Ctrl_Addr_Low	0x0033	Default Value: 0x00000000 Mask Value: 0xFFFFF000 Contains the low 32 bits of ETH_RX_Ctrl_Addr. The address is rounded down to be 4k aligned. A write to this register pushes the new values of ETH_RX_Ctrl_Addr and ETH_RX_Config forward to define to physical address and the size of the next control buffer to be used after depletion of the current control buffer.
ETH_RX_Ctrl_Addr_High	0x0034	Default Value: 0x00000000 Mask Value: 0x00000000 Contains the low 32 bits of ETH_RX_Ctrl_Addr. Only pushed forward by a write to ETH_RX_Ctrl_Addr_Low. With QPACE we have a main memory of 4GB and therefore fix its value to 0 in order to save some logic.
ETH_TX_Config	0x0035	Default Value: 0x00000000 Mask Value: 0x00000000

		There are no configuration flags for TX.
ETH_RX_Interrupt_Config	0x0036	Default Value: 0x00000000 Mask Value: 0x80000007 Bit 0: Receive Interrupts are enabled when asserted. Bit 29-31: The RX Interrupt line is being asserted when the number of frames received since the last read to ETH_RX_Interrupt_Data becomes greater or equal to 2^N, where N is the numeric value of bits 29 to 31.
ETH_RX_Interrupt_Data	0x0037	Default Value: 0x00000000 Mask Value: 0x00000000 A read to this register unasserts any RX Interrupt and resets the RX frame counter to 0. Bit 0: Asserted when the RX Interrupt line is asserted. Bit 22-31: Hold the number of frames received since the last read to this register.
ETH_TX_Interrupt_Config	0x0038	Default Value: 0x00000000 Mask Value: 0x80000007 Bit 0: Transmit Interrupts are enabled when asserted. Bit 29-31: The TX Interrupt line is being asserted when the number of frames sent since the last read to ETH_TX_Interrupt_Data becomes greater or equal to 2^N, where N is the numeric value of bits 29 to 31.
ETH_TX_Interrupt_Data	0x0039	Default Value: 0x00000000 Mask Value: 0x00000000 A read to this register unasserts any TX Interrupt and resets the TX frame counter to 0. Bit 0: Asserted when the TX Interrupt line is asserted. Bit 22-31: Hold the number of frames sent since the last read to this register.
ETH_OUT_RD_Eval	0x003A	Default Value: 0x00000000 Mask Value: 0x00F00000 One can use this register to generate some statistics on the latency behavior of the GBIF outbound read interface. A write to this register starts counting the number of clock cycles passed between request and availability of a specific transfer tag. The register is updated with the cycles after availability and can be read. Bit 4- 7: The number of the tag. Bit 8-31: The number of cycles passed.
ETH_Exc_En	0x003B	Default Value: 0x003C003C Mask Value: 0x003C003C

		This register holds the exception enable flags. All enable flags can be asserted or unasserted through a write to this register. Bit 10: TX address dropped due to RX address buffer overflow. Bit 11: read error from GBIF Outbound Interface. Bit 12: read error from TX Data buffer due to buffer underflow. Bit 13: write error to TX Data buffer due to buffer overflow. Bit 14: TX retransmit signal from EMAC. Bit 15: TX underrun signal from EMAC. Bit 26: RX Address dropped due to RX Address buffer overflow. Bit 28: read error from RX data buffer due to buffer underflow. Bit 29: write error to RX Data buffer due to buffer overflow. Bit 31: frame was dropped due to RX buffer overflow.
ETH_Reserved_1	0x003C	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_2	0x003D	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_3	0x003E	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_4	0x003F	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.

4.1.3.2 GTX Slave

These control registers determine the behavior of the logic driven by the Gigabit Transmit (GTX) clock of the Ethernet Core.

Register Name	Address	Register Description
ETH_TX_IFG_Delay	0x0040	Default Value: 0x000000B Mask Value: 0x000000FF This register is fed into the EMAC and controls the Inter Frame Gap (IFG) for TX. The supplied number defines the number of bytes of 'silence' on the link between frames. Its minimum, defined by IEEE 802.3, is 0xB. It can be increased to increase compatibility with older devices. See [29]. Bit 24-31: Length of the IFG.
ETH_TX_Pause_Value	0x0041	Default Value: 0x0000FFFF

		Mask Value: 0x0000FFFF Controls the Pause Value sent by EMAC upon a Pause Value Request by for example a Ethernet RX Flow Controller. See [29]. Bit 16-31: The Pause Value
ETH_RGMII_RXD_Delay	0x0042	Default Value: 0x00000000 Mask Value: 0x00000000 Read Only. Contains the current Delay Values of the RXD lines. Bit 2- 7: RXD_3 Delay Value Bit 10-15: RXD_2 Delay Value Bit 18-23: RXD_1 Delay Value Bit 26-31: RXD_0 Delay Value
ETH_RGMII_RXC_Delay	0x0043	Default Value: 0x00000000 Mask Value: 0x00000000 Read Only. Contains the current Delay Values of the RXC lines. Bit 18-23: RXCTL Delay Value Bit 26-31: RXC Delay Value
ETH_RGMII_RX_Delay_Cmd	0x0044	Default Value: 0x00000000 Mask Value: 0x00000000 A write to this register pushes commands to the RX Delay Unit. Resets supersede increments or decrements. Bit 9: RXCTL Delay Value reset Bit 10: RXCTL Delay Value inc(asserted) or dec Bit 11: RXCTL Delay Value change enable Bit 13: RXC Delay Value reset Bit 14: RXC Delay Value inc(asserted) or dec Bit 15: RXC Delay Value change enable Bit 17: RXD_3 Delay Value reset Bit 18: RXD_3 Delay Value inc(asserted) or dec Bit 19: RXD_3 Delay Value change enable Bit 21: RXD_2 Delay Value reset Bit 22: RXD_2 Delay Value inc(asserted) or dec Bit 23: RXD_2 Delay Value change enable Bit 25: RXD_1 Delay Value reset Bit 26: RXD_1 Delay Value inc(asserted) or dec Bit 27: RXD_1 Delay Value change enable Bit 29: RXD_0 Delay Value reset Bit 30: RXD_0 Delay Value inc(asserted) or dec Bit 31: RXD_0 Delay Value change enable
ETH_RX_SKB_Addr_High	0x0045	Default Value: 0x00000000 Mask Value: 0xFFFFFFFF Contains the high 32 bits of ETH_RX_SKB_Addr.
ETH_RX_SKB_Addr_Low	0x0046	Default Value: 0x00000000 Mask Value: 0xFFFFFFFF

		Contains the low 32 bits of ETH_RX_SKB_Addr. A write to this register pushes the new value of ETH_RX_SKB_Addr into the RX Address FIFO.
ETH_TX_SKB_Addr_High	0x0047	Default Value: 0x00000000 Mask Value: 0xFFFFFFFF Contains the high 32 bits of ETH_TX_SKB_Addr.
ETH_TX_SKB_Addr_Low	0x0048	Default Value: 0x00000000 Mask Value: 0xFFFFFFFF Contains the low 32 bits of ETH_TX_SKB_Addr. A write to this register pushes the new value of ETH_TX_SKB_Addr into the TX Address FIFO.
ETH_Slow_Interrupt_Config	0x0049	Default Value: 0x00000000 Mask Value: 0x00000001 Configures the behavior of interrupts other than TX or RX. Bit 31: PHY Interrupt enable
ETH_Slow_Interrupt_Data	0x004A	Default Value: 0x00000000 Mask Value: 0x00000000 Refects the current interrupt flags other than TX or RX. COR disabled within logic. Bit 30: EMAC Register Access Complete Bit 31: PHY Interrupt
ETH_SKB_Status	0x004B	Default Value: 0x00000000 Mask Value: 0x00000000 Contains status flags on the address FIFOs. Bit 14: TX Address FIFO full Bit 30: RX Address FIFO full
ETH_Reserved_5	0x004C	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_6	0x004D	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_7	0x004E	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.
ETH_Reserved_8	0x004F	Default Value: 0x0000000 Mask Value: 0x00000000 Reserved.

4.1.3.3 Ethernet MAC Slave

These control registers determine the behavior of the logic of the Xilinx Virtex 5 Trimode Gigabit Ethernet Hard Core. They provide an indirect access to its large number of registers. These not only control the EMAC, but also instruct a MDIO clause 22 FSM in the EMAC that talks to the connected Ethernet PHY to access its registers. A complete description can be found in [29].

Register Name	Address	Register Description
ETH_EMAC_dataRegMSW	0x0020	Default Value: Undefined

		Data input from the DCR bus for the EMAC registers or other accessible registers is written into this register. The most significant word of data is read out from the EMAC registers or other registers and deposited into this register.
ETH_EMAC_dataRegLSW	0x0021	Default Value: Undefined Data input from the DCR bus for the EMAC registers or other registers is written into this register. The least significant word of data is read out from the EMAC registers or other registers is deposited into this register.
ETH_EMAC_cntlReg	0x0022	Default Value: 0x0000000 Controls any read or write operations on the internal registers Bit 0-15: Reserved. Bit 16: Write Enable. When this bit is asserted, the data in either dataRegLSW or dataRegMSW is written to the EMAC host interface. When this bit is deasserted, the operation to be performed is read. Bit 17-21: Reserved. Bit 22-31: Address Code. The DCR bus bridge translates the address code for this transaction into the appropriate signals on the EMAC host interface.
ETH_EMAC_RDYstatus	0x0023	Default Value: undefined Status Register Bit 0-14: Reserved. Bit 15: DCR Bridge Ready. High when all of the bits in RDYstatus are High. Bit 16-24: Reserved. Bit 25: Configuration Write-Ready Bit 26: Configuration Read-Ready Bit 27: Address Filter Write-Ready Bit 28: Address Filter Read-Ready Bit 29: MDIO Write-Ready Bit 30: MDIO Read-Ready Bit 31: FPGA Logic Read-Ready

4.1.4 Logic Interface

The following signal types are defined by:

eth_phy_in_type	ET_TXD	std_logic_vector(3 downto 0)
	ET_TX_CTRL	std_logic
	ET_TX_CLK	std_logic
	ET_MDC	std_logic
	ET_RESETN	std_logic
eth_phy_out_type	ET_RXD	std_logic_vector(3 downto 0)
	ET_RX_CTRL	std_logic

	ET_RX_CLK	std_logic
	ET_LED2	std_logic
eth_phy_inout_type	ET_MDIO	std_logic
dcr_slv_in_type	readEn	std_logic
	writeEn	std_logic
	abus	std_logic_vector(0 to 15)
	dbus	std_logic_vector(0 to 31)
dcr_slv_out_type	ack	std_logic
	dbus	std_logic_vector(0 to 31)
iwc_sl_type	data	std_logic_vector(0 to 127)
	addr	std_logic_vector(0 to 41)
	first	std_logic
owc_out_type	done_toggle	std_logic
	addr_line	std_logic_vector(0 to 2)
	read_enable	std_logic
	first	std_logic
owc_in_type	data	std_logic_vector(0 to 127)
	addr_packet	std_logic_vector(0 to 41)
	req_toggle	std_logic
	notify	std_logic
	notify_data	std_logic_vector(0 to 127)
	notify_addr	std_logic_vector(0 to 41)

The Ethernet Core has following of interfaces:

Interface to PHY	eth_phy_out	in	eth_phy_out_type
	eth_phy_inout	inout	eth_phy_inout_type
	eth_phy_in	out	eth_phy_in_type
DCR Interface GTX	dcr_eth_gtx_in	in	dcr_slv_in_type
	dcr_eth_gtx_out	out	dcr_slv_out_type
DCR Interface GBIF	dcr_eth_gbif_in	in	dcr_slv_in_type
	dcr_eth_gbif_out	out	dcr_slv_out_type
DCR Interface EMAC	dcr_eth_emac_in	in	dcr_slv_in_type
	dcr_eth_emac_out	out	dcr_slv_out_type
Transmit Clock	gtx_clk	in	std_logic
	gtx_clk_dcm_locked	in	std_logic
GBIF Clock	gbif_clk	in	std_logic
	gbif_clk_dcm_locked	in	std_logic
DCR Master Clock	dcr_clk	in	std_logic
	dcr_sreset	in	std_logic
IO Delay Controllers ready	iodelayctrl_rdy	in	std_logic
Outbound Read GBIF Signals	al_out_rd	in	al_out_rd_type
	gbif_out_rd	out	gbif_out_rd_type
IWC Interface	iwc_sl	in	iwc_sl_type
	iwc_we	in	std_logic
	iwc_almost_full	out	std_logic
OWC Interface	owc_out	in	owc_out_type
	owc_in	out	owc_in_type
Interrupt lines	interrupt_rx	out	std_logic
	interrupt_tx	out	std_logic

	`interrupt_slow`	out	`std_logic`
	`exception`	out	`std_logic`
Asynchronous Reset	`reset`	in	`std_logic`

4.1.5 QPACE Ethernet Logic

4.1.5.1 Ethernet Logic

4.1.5.1.1 Data FIFOs

The Xilinx Virtex 5 provides its logic devices with embedded BRAMs. These may be configured to operate as FIFOs. In our design we transfer data between a System Bus and an EMAC being operated by different clocks. These are very handy at moving data between clock domains and providing data buffering at the same time. Xilinx provides word widths of $4 \cdot (8 + 1) = 36$, i.e., 4 Bytes and one additional bit per byte, that is commonly used as parity bit. A parity bit is a bit that holds the parity of a sequence of corresponding bits, in this case a byte. A parity bit may be used to check the integrity of the data stored in a memory. If the parity is not as expected, data was corrupted during storage.

The System Bus we use has a width of 128 bits and operates at a rather high frequency for current generation FPGAs, while the EMAC clock frequency is rather low (125 MHz) and the bus width to the EMAC is only 8 bits. For this reason we took the decision to employ as few logic as possible on the System Bus side and move 8-bit serialization or deserialization to the EMAC side. Thus we need to employ FIFOs that have a width matching the System Bus width. Hence we take 4 36 Bit FIFOs in parallel. Parallization of FIFOs to an extra wide FIFO is essentially trivial. In principle, since any write or read to the sub-FIFOs is done at the same clock cycle, all status signals given by the sub-FIFOs are identical. These are for example empty signals, full signals and error signals. This simple view remains correct as long as synchronous FIFOs are employed. For asynchronous FIFOs, information travelling between clock domains can get delayed by one clock cycle or more. The reason is that the physical devices employed today to hold information are Flip-Flops. These are exposed to meta-stability if for example clock signal changes coincide with data signal changes. One or more of the sub-FIFOs may be in such a state and its status signals therefore be delayed. For example, this means that the wide FIFO is already full if one of its sub-FIFOs is full. For this reason a suitable combinatorial logic needs to be taken for each of the sub-FIFO signal types to yield the wide FIFOs signals.

Next, we need to think about the storage capacity necessary to support Ethernet operation. For Transmit, we need to make sure, there is no FIFO underrun to the EMAC, since it needs the frame data to be provided in a continuous data stream. As the bandwidth of the System Bus is significantly higher (a factor on the order of 32), there may only be rare cases of such a underrun. Nevertheless we demand that frames are only started to be sent (TX) once they completely reside in the buffer. While a frame is being sent, we like to store the next one as well, so a buffer space of at least two frames is advisable. In jumbo frame mode an Ethernet frame can be as large as 8kB. We therefore took the decision to provide 32kB of buffer space in the FIFO. This amounts to serialize 4 of the wide FIFOs to obtain a single 4096×144 FIFO. We employ the method explained in [22] to serialize FIFOs. Here we highlight, that in order to have a continuous data atream on the reading side, it is necessary to drive any intermediate FIFOs with the faster clock. This ensures that data is pushed forward to thereading end of the FIFO faster than it can be read out.

For the handling of Ethernet data it is essential to give some additional data to the reading side of the FIFO. In our case all we need are some flags that tag the start and the end of a

frame within the FIFO data stream. We use the following protocol: We abuse the parity bit to convey information on the frame affiliation of the corresponding byte in the following way:

1. A new frame starts with a new FIFO word.
2. All parity bits until the first frame byte are asserted.
3. The first frame byte within each frame is marked with the parity bit being deasserted (i.e., set to 0).
4. The last frame byte within each frame is marked with the parity bit being asserted (i.e., set to 1).
5. The current FIFO word parity bit `0xF` is asserted to mark the end of the frame.
6. Broken or to be disposed frames are marked with parity bits `0xD` to `0xF` asserted in their last FIFO word.

We show examples in figure 4.4. These rules may seem rather abstruse. Instead we could

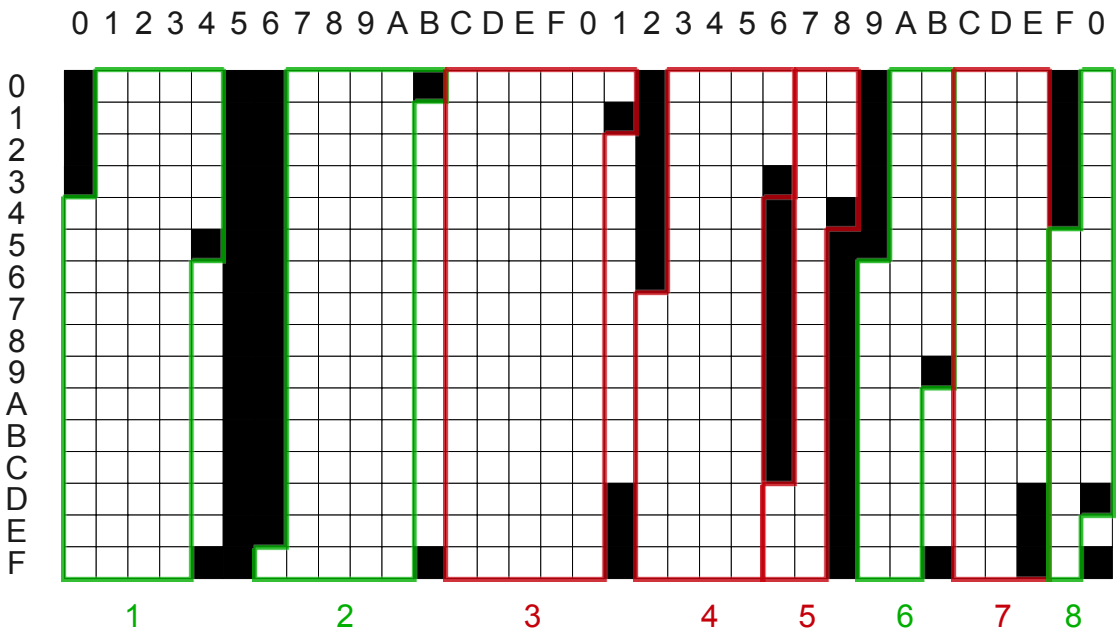


Figure 4.4: A snapshot of an (erroneous) frame stream through our Data FIFOs. We have a FIFO word width of 16 bytes. Each byte is represented by a box. Asserted parity bits are black, unasserted are white. According to our definition the FIFO reader can decode relevant frame data. Frames 1 and 2 are valid frames according to our rules. Frame 3 is also consistent with our rules, however marked as broken by the last three parity bits of the word. Frame 4 will not be recognized since no frame end parity bit is set. Instead it is going to be concatenated with frame 5, that is marked as broken. Frame 6 and 8 are fine, Frame 7 will be disposed.

have easily used the 16 parity bits of each word to put flags of invalid, frame start, frame end, broken and two 4 bit values of first and last byte into the stream and decode everything from that. However, we did not employ this variant in order to keep the logic simple and rather use an unintuitive scheme:

For example, the fifth rule is not necessary in general to properly decode frame boundaries, but we include it in order to decode frame ends more efficiently. We have a frame end when the parity bit of byte 0xF is set. This means we have only one signal to interpret, instead of an or-reduce of 16 bits. For this reason we can save one step in the processing pipeline and reduce the logic footprint by at least 144 Flip-Flops, roughly 1/3 of the outbound write logic for example.

Since we need to know on the read side whether a new frame was written to the FIFO on the write side, we perform a clock domain transition of such an event using the Non-return-to-zero (NRZ)[1] method. Each new frame signal is provided to the reader of the FIFO by a pulse in `rd_new_frame`.

In many cases, ours as well, attached entities might need to know about the fill status of the FIFO as well. Full and Empty, as well as Almost Full and Almost Empty with some fixable threshold are being provide by the Xilinx FIFOs. However some attached write logic might need to know, how many lines it can (at least) write to the FIFO until it may get filled up, while some attached read logic might need to know, how many lines it can (at least) read from the FIFO until it may become empty. For example the Outbound Write Logic makes use of this information to avoid blocking bus transfer tags, when data cannot be forwarded immediately.

We added signals that give the number of free words `WR_FREE_GTE` to the write side. Since the exact determination of the amount of lines is logically intensive for asynchronous FIFOs, we exploit the fact that we work with a depth-stacked FIFO: In this case the write side of the FIFO is faster than the read side, so FIFOs 1 to $N-1$ run at the write clock speed on both ports. (FIFO N is at the read side and provided the clock transition. Within this setup we may decode from the write-enables to FIFO 1 and the read-enables to FIFO $N-1$ whether the fill level of the depth-stacked FIFOs 1 to $N-1$ has increased by one, has decreased by one, or is (effectively) unaffected. After reset of the full FIFO, we have at least $c \cdot (N-1)$ free words available, where c is the depth of a single FIFO. We count up or down each cycle to determine the number of currently at least available free words in the FIFO. A similar technique can be used to generate signals that give the number of occupied words `RD_OCCD_GTE` to the read side, a signal which we also add to the entity.

Altogether we provide a 16 byte wide FIFO of variable depth designed to transport and buffer frames of data between clock domains in an entity called `eth_fifo`:

Signal Type	Signal Name	Description
Generics	`ALMOSTEMPTY`	natural Sets number of FIFO to be stacked
	`WR_FASTER_RD`	boolean Set to true if WRCLK is faster than RDCLK
	`ALMOST_FULL_OFFSET`	`bit_vector(0 to 15)` Sets almost full threshold
	`ALMOST_EMPTY_OFFSET`	`bit_vector(0 to 15)` Sets the almost empty threshold
	`DO_REG`	natural Enable output register, must be true if async
	`EN_ECC_READ`	boolean Enable ECC decoder
	`EN_ECC_WRITE`	boolean Enable ECC decoder

[1] a binary encoding technique that can be used for asynchronous signaling

	EN_SYN	boolean Specifies FIFO as Asynchronous (FALSE) or Synchronous (TRUE)
	FIRST_WORD_FALL_THROUGH	boolean Sets the FIFO First Word Fall Through (FWFT) to TRUE or FALSE
	SIM_MODE	string Simulation: “SAFE” vs “FAST”, see [30] for details
Read Side	DO	out std_logic_vector(0 to 127) Data Output
	DOP	out std_logic_vector(0 to 15) Data Parity Output
	ECCPARITY	out std_logic_vector(0 to 15) ECC Parity Output
	EMPTY	out std_logic Empty Flag
	ALMOSTEMPTY	in std_logic Almost Empty Flag
	RDERR	out std_logic Read Error Flag
	RDCLK	in std_logic Read Clock Input
	RDEN	in std_logic Read Enable
	RD_OCCD_GTE	out std_logic_vector(0 to ETH_FIFO_LINES_LD) Greater-than-or-equal Occupied words
	rd_new_frame	out std_logic Pulses on each new frame written
Write Side	WRERR	out std_logic Write Error Flag
	DI	in std_logic_vector(0 to 127) Data Input
	DIP	in std_logic_vector(0 to 15) Data Parity Input
	WRCLK	in std_logic Write Clock Input
	WREN	in std_logic Write Enable
	FULL	out std_logic Full Flag
	ALMOSTFULL	inout std_logic Almost Full Flag
	WR_FREE_GTE	out std_logic_vector(0 to ETH_FIFO_LINES_LD) Greater-than-or-equal Free words
Other	RST	in std_logic Asynchronous Reset

DBITERR	out std_logic Double Bit ECC Error
SBITERR	out std_logic Single Bit ECC Error

4.1.5.1.2 TX Serializer

In order to pull data from a FIFO and push it to the EMAC some logic essentially need to translate between FIFO and EMAC interface language. For transmission we employ a TX Serializer entity named `eth_packet_to_emac` , that is responsible for forwarding frames from the FIFO to the EMAC. This jobs involves

1. Reading data from the FIFO,
2. Decoding frame boundaries,
3. Serialization,
4. pushing data into the EMAC using the mandatory interface described in [29],
5. Error detection and reporting.

The interface of TX Serializer the was defined as follows:

Signal Type	Signal Name	Description
Common	tx_clk	in std_logic
	tx_reset	in std_logic
EMAC Side	tx_enable	in std_logic
	tx_data	out std_logic_vector(0 to 7)
	tx_data_valid	out std_logic
	tx_ack	in std_logic
	tx_collision	in std_logic
	tx_retransmit	in std_logic
	tx_underrun	out std_logic
FIFO Side	rd_en	out std_logic
	rd_data	in std_logic_vector(0 to 127)
	rd_data_eof	in std_logic_vector(0 to 15)
	rd_empty	in std_logic
	tx_new_frame	in std_logic

The logic contains a FSM that organizes the work to be done. We show a graphical representation of the machine in figure 4.5. After reset, the machine resides in the IDLE state. The `tx_new_frame` signal is used to increment a counter holding the number of available frames. The `frame_available` flag is set once data can be read out of the FIFO, as indicated by `rd_empty`. If the logic is in `ETH_NO_TX_UNDERRUN` mode the number of frames available additionally needs to be larger than 0. In this mode it can be made sure that the FIFO will not underrun during the transmission of a frame, a mode that may be necessary since no automated retransmission of underrun frames is implemented.

Once `frame_available` is set, the machine enters the SEEK state. The SEEK state is used to decode the parity bits `tx_data_eof` of the last word read out of the FIFO to find the first parity bit zero marking the beginning of the frame into a 4 bit counter register i.

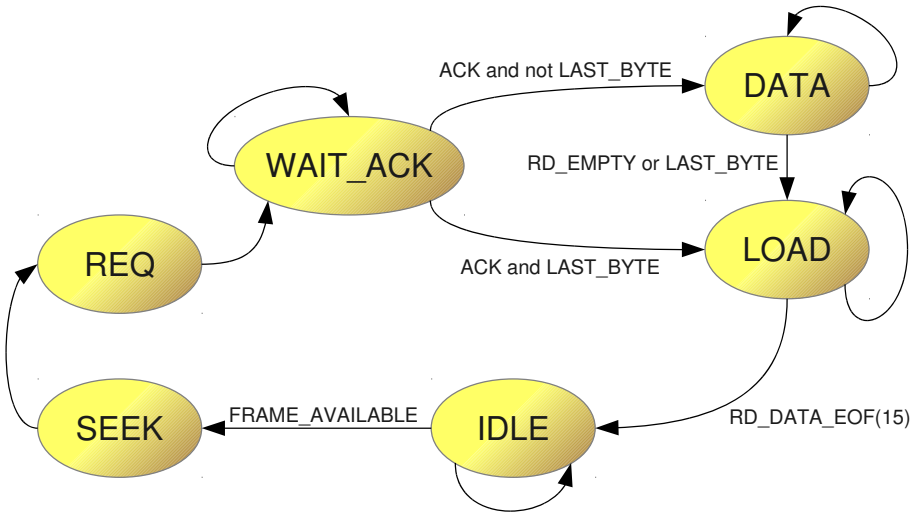


Figure 4.5: FSM within the TX Serializer.

The SEEK state is left immediately for the REQ state. The ith byte of `rx_data` is put on the output `tx_data` and `tx_data_valid` is asserted to mark the beginning of a frame to the EMAC.

The REQ state is left immediately for the WAIT_ACK state. This state loops until an ACK is received from the EMAC. If the byte sent is not last byte of the frame, the machine enters the DATA state. Else the machine jumps over the DATA state into the LOAD state.

The DATA state is the state that provides the frame data to the EMAC. Each (relevant) cycle i is incremented, and the ith byte of `rx_data` provided to the EMAC. The reading of the next word from the FIFO is timed to coincide with i wrapping around. This continues until the end of the frame is detected and the machine switches into the LOAD state. During the DATA state it may be that the FIFO empties unexpectedly. This erroneous behavior of the data stream needs to cancel the transmission of the frame and is indicated to the EMAC using a pulse in `tx_underrun`. Further, we advance to the LOAD state to end the current frame.

The LOAD state posts the last byte to the EMAC and deasserts `tx_data_valid` afterwards. Frame transmission is complete and the machine switches to the IDLE state, thereby decrementing the number of available frames.

There are two more input signals to the TX Serializer: `tx_retransmit` and `tx_collision`. The latter one flags the EMAC having problems with the transmission of the frame due to a collision of RX and TX when in Half-Duplex mode. The EMAC can request a retransmission of a frame several times by `tx_retransmit` until it may give up. For QPACE we took the decision not to be compatible to legacy technology and support the Full-Duplex mode only.

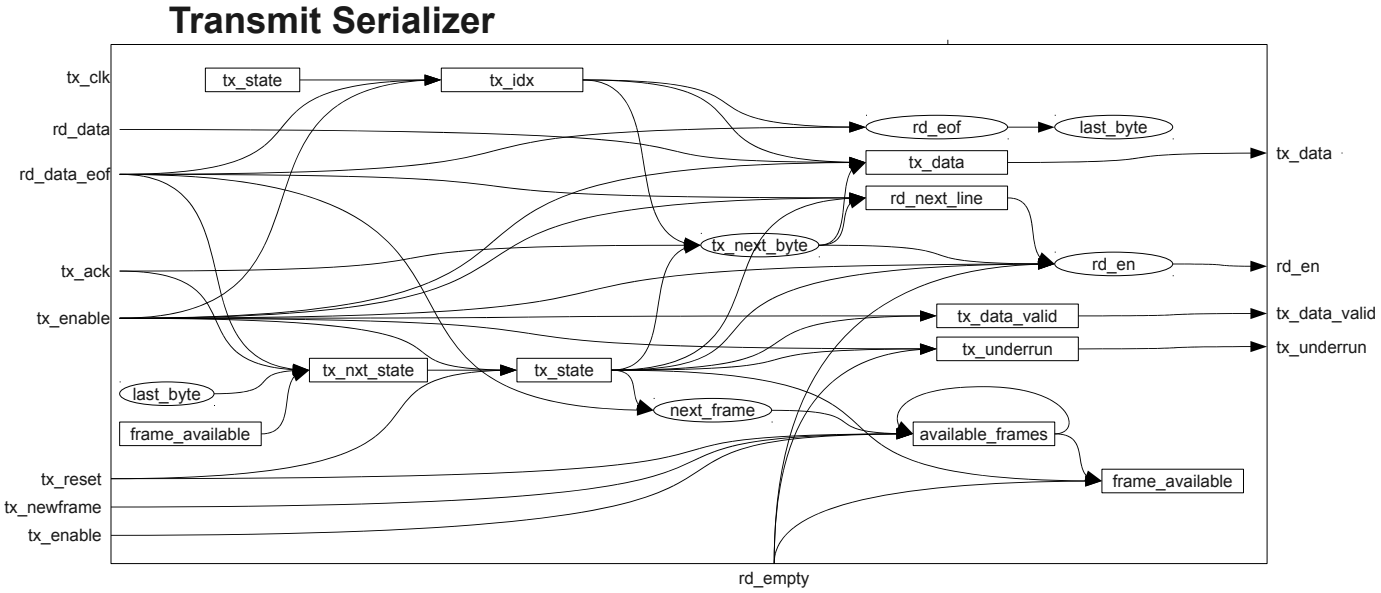


Figure 4.6: Block Diagram of the TX Serializer.

4.1.5.1.3 RX Deserializer

In order to pull data from an EMAC and push it into the FIFO some logic is needed to interface between the FIFO and the EMAC. For reception we employ a RX Deserializer entity named `eth_packet_from_emac`, which is responsible for forwarding frames from the EMAC to the FIFO. These jobs involve

1. Reading the EMAC input stream using the mandatory interface described in [29],
2. Decoding frame boundaries,
3. Deserialization,
4. Pushing data into the FIFO, while providing boundary and validity information using the parity bits
5. Error detection and reporting.

The interface of RX Deserializer the was defined as follows:

Signal Type	Signal Name	Description
Common	`rx_clk`	in `std_logic`
	`rx_reset`	in `std_logic`
EMAC Side	`rx_enable`	in `std_logic`
	`rx_data`	in `std_logic_vector(0 to 7)`
	`rx_data_valid`	in `std_logic`
	`rx_good_frame`	in `std_logic`
	`rx_bad_frame`	in `std_logic`
FIFO Side	`wr_en`	out `std_logic`
	`wr_data`	out `std_logic_vector(0 to 127)`
	`wr_data_eof`	out `std_logic_vector(0 to 15)`
	`wr_full`	in `std_logic`
	`wr_almost_full`	in `std_logic`

Other	`rx_drop`	out `std_logic`
	`rx_accept`	out `std_logic`

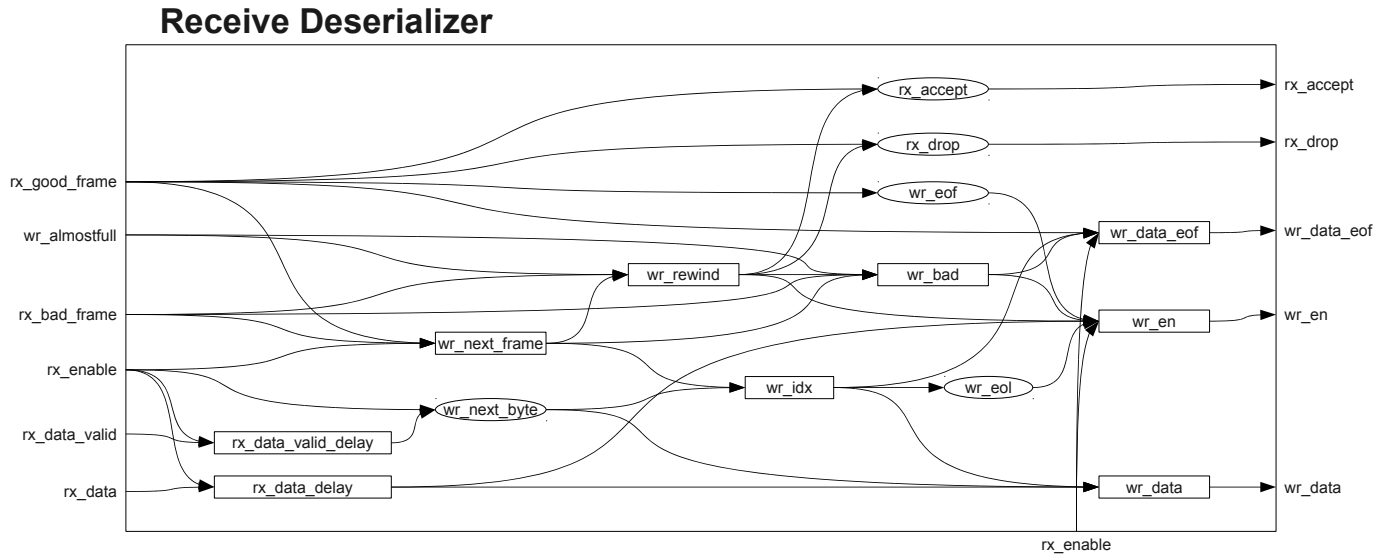


Figure 4.7: Block Diagram of the RX Deserializer.

The RX Deserialization is somewhat simpler that the TX Serializer, so we did not implement a FSM that controls the flow of data. The interface provided by the EMAC is explained in [29]: The EMAC outputs a stream of data in `rx_data` each cycle marked as enabled. The first byte of a frame is marked with the signal `rx_data_valid` being asserted. The first byte after the end of a frame is marked with it being deasserted.

Once we find the first byte of a frame we increment a 4 bit counter register each enabled cycle. Then we use this counter to address a 16 byte wide register to consecutively store the bytes received from the EMAC each enabled cycle. Each cycle for which all 16 bytes are occupied and no frame end is detected, we write the data into the FIFO together with their parity bits deasserted.

Once a frame end is detected, we stop this procedure and wait for the EMAC to signal the validity of the received frame. If the frame is good, as indicated by a pulse in `rx_good_frame`, we encode the parity bits properly, i.e., according to the scheme we defined in section 4.1.5.1.1. If the frame is bad, we assert the last three parity bits of the word to indicate a broken frame.

In case the FIFO got full during the reception of a frame, frame data is lost. In such a case we stop writing to the FIFO, wait for `rx_data_valid` to be deasserted, i.e., the end of the frame and treat the data written to the FIFO as bad, i.e., we assert the last three parity bits of the word to indicate a broken frame.

Finally, data is pushed to the FIFO and we are ready for the next frame start by resetting the counter to zero.

For any good frame we fully wrote into the FIFO, we provide a pulse in `rx_accept`. If the FIFO got full while reception, we provide a pulse in `rx_drop`. Pulses on bad frames are already provided by the EMAC through `rx_bad_frame`.

4.1.5.1.4 DCR Legacy Bridge

The DCR Legacy Bridge is a simple entity that translates the modern variant of the DCR Architecture described in [24] into a legacy variant understood by the Trimode EMAC hard core available in the FPGA we employ. The old variant comes with a smaller bus width of 10 bits instead of 32 bits. Just to match address lines is not enough. We need to validate or invalidate read or write requests to the EMAC DCR Slave depending on whether the full address is within the EMAC's smaller address space as well.

4.1.5.1.5 Flow Controller

As described in [29], a Flow Control Block is commonly employed in Ethernet Communication. It is part of the IEEE 802.3 standard that enables special Ethernet frames to be sent by (A) to (B). These require the link partner (B), i.e., the device on the other end of the physical link (an Ethernet switch or Ethernet hub in most cases), to pause the transmission of frames to (A) for a specific time. Instead frames should be buffered if possible and sent later. The transmission of such pause control frames gives (A) enough time to move data from the RX FIFO into main memory, making space available for new frames. At first sight, if (A) cannot free enough space for new frames, it may just be too slow. However, when both (A) and (B) are clocked independently at the same frequency (a common case), tiny differences in the clock can fill up RX buffers over time leading to hick-ups in data transmission.

For QPACE we prepared a `flow_controller`, an entity that monitors the fill level of the RX Buffers and eventually instructs the Flow Control Block within the EMAC to issue flow control frames appropriately:

Signal Type	Signal Name	Description
RX FIFO Side	`clk`	in `std_logic`
	`reset`	in `std_logic`
	`almostfull`	in `std_logic`
	`almostempty`	in `std_logic`
TX EMAC Side	`tx_clk`	in `std_logic`
	`tx_pause_req`	out `std_logic`
	`tx_pause_val`	out `std_logic_vector(0 to 15)`
	`tx_pause_val_in`	in `std_logic_vector(0 to 15)`

We have two clock inputs for this entity since the RX FIFO write side is driven by the RX Clock, a clock regenerated from the physical link by the PHY which thus resembles the TX clock of second device. The TX Clock is the clock the EMAC uses to operate and it expects the request posed to him to be synchronous to this clock.

Therefore we use the `almostfull` and `almostempty` signals to generate pulses indicating whether the FIFO just got almost full or empty. Then we generate a NRZ clock domain transition for these signals to post a request to transmit a pause control frame in both cases. If the FIFO got almost empty we send a pause value of zero, cancelling all pause requests to the link partner. If it got almost full, we send the pause value from `tx_pause_val_in`. This signal is set by the DCR Register `0x0041`, see section 4.1.3.2.

For QPACE we found that there is no need for a RX Flow Controller, since the System Bus can pull the data vastly faster from the RX FIFO than new data comes in. For this reason, the RX Flow Controller is currently disabled using the VHDL flag `ETH_FLOW_CTRL=false`. The EMAC, of course, still follows the IEEE 803.2 standard and reacts to Pause Control Frames appropriately in order to support any other link partner (which may send Pause Control Frames). QPACE Ethernet does not send them.

4.1.5.1.6 Ethernet Media Access Controller

The FPGA device we employ, the "Xilinx Virtex-5 LT110T", offers its user two Gigabit Ethernet Media Access Controllers, called "Virtex-5 FPGA Embedded Tri-Mode EMAC". All its functionality is described extensively in [29]. For QPACE we are satisfied with a bandwidth of one Gigabit for booting, IO, etc. so we use only one of the two.

The EMAC is defined in the IEEE 802.3 specification in clauses 2, 3, and 4. A MAC is responsible for the Ethernet framing protocols and error detection of these frames. The MAC is independent of and can connect to any type of physical layer device. The MAC Control sublayer is defined in the IEEE 802.3 specification, clause 31. This provides real-time flow control manipulation of the MAC sublayer. Both the MAC CONTROL and MAC sublayers are provided by the EMAC in all modes of operation. We list a number of important functions and defer to [29] for more information:

- Fully integrated 10/100/1000 Mb/s EMAC
- Configurable full-duplex operation in 10/100/1000 Mb/s
- Configurable half-duplex operation in 10/100 Mb/s
- MDIO interface to manage objects in the physical layer
- Configurable IFG adjustment in full-duplex operation
- Configurable in-band Frame Check Sequence (FCS) field passing on both transmit and receive paths
- Auto padding on transmit and stripping on receive paths
- Manageable through DCR bus
- Configurable flow control through EMAC Control PAUSE frames; symmetrically or asymmetrically enabled
- Configurable support for jumbo frames of any length
- Configurable receive address filter for unicast, general, and broadcast addresses
- MII, Gigabit MII (GMII), and RGMII

4.1.5.1.7 RGMII Interface

The RGMII 2.0 Interface is one of the many standardized interfaces available for Ethernet. The MII, the Media Independent Interface, is a legacy variant supporting 100MBit Ethernet. Next there is GMII, designed to fit Gigabit PHYs. A improved variant of GMII is RGMII, a reduced version in that is only needs 12 electric connections instead of the 24 that GMII operates on. Our RGMII Interface entity translates EMAC Transmit and Receive signals into Double Data Rate (DDR) GMII Signals, i.e., RGMII signals. We use the following entity interface:

Signal Type	Signal Name	Description
Common	`RESET`	in `std_logic`
RGMII Transmit	`RGMII_TXC`	out `std_logic_vector(0 to 3)`
	`RGMII_TXD`	out `std_logic`
	`RGMII_TX_CTL`	out `std_logic_vector(0 to 3)`

RGMII Receive	RGMII_RXC	in std_logic
	RGMII_RXD	in std_logic_vector(0 to 3)
	RGMII_RX_CTL	in std_logic
RGMII Delays	RGMII_DELAY_CLOCK	in std_logic
	RGMII_DELAY_RXC_CTRL	in iodelay_ctrl_type
	RGMII_DELAY_RXC_STATUS	out iodelay_status_type
	RGMII_DELAY_RXD_CTRL	in iodelay_ctrl_array(0 to 3)
	RGMII_DELAY_RXD_STATUS	out iodelay_status_array(0 to 3)
	RGMII_DELAY_RXCTL_CTRL	in iodelay_ctrl_type
	RGMII_DELAY_RXCTL_STATUS	out iodelay_status_type
EMAC Transmit	TXD_RISING_FROM_MAC	in std_logic_vector(0 to 3)
	TXD_FALLING_FROM_MAC	in std_logic_vector(0 to 3)
	TX_CTL_RISING_FROM_MAC	in std_logic
	TX_CTL_FALLING_FROM_MAC	in std_logic
	TX_CLK	in std_logic
EMAC Receive	RXD_RISING_TO_MAC	out std_logic_vector(0 to 3)
	RXD_FALLING_TO_MAC	out std_logic_vector(0 to 3)
	RX_CTL_RISING_TO_MAC	out std_logic
	RX_CTL_FALLING_TO_MAC	out std_logic
	RX_CLK	out std_logic

For transmit, this entity instantiates output registers for each of the data and control signals, rising and falling. Their registered outputs are then fed into 5 DDR Output Buffers, combining rising and falling signals into a double data rate signal driven by the transmit clock `TX_CLK`. The transmit clock itself does not need to be buffered and is directly fed into a DDR Output Buffer. Afterwards we delay it 2 ns, as defined by the RGMII standard (the PHY was configured to obey this standard), before it reaches the electric circuit on the board. This definition enables the PHY to simply sample incoming TX lines on incoming (DDR-ed) clock edges. We have a TX clock of 125 MHz, i.e., a cycle period of 8 ns. This amounts to a DDR period of 4 ns. This means that we sample at the middle of the DDR period, i.e., properly if all wires have the same length. Delaying buffered outgoing signals can simply be done by feeding them through an IODELAY instance with our FPGA. IODELAY instances take a reference clock that needs to be supplied to an IDELAYCTRL. In QPACE other entities may use output or input delays as well, so the IDELAYCTRL is hierarchically situated outside the QPACE Ethernet logic. The QPACE Ethernet RGMII Interface assumes a clock at 200 MHz to be provided to the IDELAYCTRL.

For receive, the opposite path needs to be taken. We feed all incoming RX signals (6 DDR signals) to the FPGA through IDELAYs, i.e., input delays. The IDELAY are driven by the clock `RGMII_DELAY_CLOCK` and the delay they apply can be controlled through delay control signals `RGMII_DELAY_RX*_CTRL`. The user can control the delays by posting commands in writing to the relevant DCR registers. We additionally instantiate counter registers `RGMII_DELAY_RX*_STATUS` which hold the current delay tap value. See section 4.1.3.2 on how to access the relevant DCR registers. The delayed receive clock is promoted to a Global Clock, a clock that can drive a large amount of logic within the FPGA, and is provided in `RX_CLK`. This clock is also used to deserialize the remaining properly delayed DDR signals (data and control) which are then provided to the EMAC.

For QPACE, we have found that no user configurable delays need to be employed on RX for a proper operation of QPACE Ethernet. We therefore disabled this feature in order to save some logic using the VHDL parameter `ETH_RGMII_DELAY_ADJUST=false`.

4.1.5.1.8 MDIO Interface

Management Data Input/Output, or MDIO, is a serial bus defined for the Ethernet IEEE 802.3 specification for Media Independent Interface, or MII. The MII connects MAC devices with Physical Transceivers (PHYs). MDIO is similar to the I^2C bus, and provides the MAC access to PHY registers. Since there is already a MDIO Master core in the EMAC, that we can control through DCR, this logic entity is solely comprised of an output buffer for the MDIO Clock and a tri-state buffer for the MDIO Data line. This is the point where signals enter or exit the FPGA onto the logic board.

4.1.5.2 Outbound Read Logic

The Outbound Read Logic is in charge of fetching frame data from memory through Outbound Read, based on addresses and lengths provided by the Address FIFO, into the Transmit FIFO. Depending on the configuration provided at run-time, interrupts shall be issued and status reported. We use the following entity interface:

Signal Type	Signal Name	Description
Common	`clk`	in `std_logic`
	`sreset`	in `std_logic`
Outbound Read	`al_out_rd`	in `al_out_rd_type`
	`gbif_out_rd`	out `gbif_out_rd_type`
Transmit FIFO	`wr_en`	out `std_logic`
	`wr_data`	out `std_logic_vector(0 to 127)`
	`wr_data_eof`	out `std_logic_vector(0 to 15)`
	`wr_full`	in `std_logic`
	`wr_almostfull`	in `std_logic`
	`wr_free_gte`	in `std_logic_vector(0 to ETH_FIFO_LINES_LD)`
Address FIFO	`tx_skb_rd_clk`	out `std_logic`
	`tx_skb_rd_en`	out `std_logic`
	`tx_skb_rd_data`	in `std_logic_vector(0 to 63)`
	`tx_skb_rd_empty`	in `std_logic`
Interrupts	`interrupt`	out `std_logic`
	`interrupt_ack`	in `std_logic`
	`interrupt_enable`	in `std_logic`
	`interrupt_data`	out `std_logic_vector(0 to 31)`
	`interrupt_config`	in `std_logic_vector(0 to 2)`
Status	`gbif_err`	out `std_logic`
	`gbif_latency`	out `std_logic_vector(0 to 23)`
	`gbif_latency_tag`	in `std_logic_vector(0 to 3)`
	`gbif_latency_start`	in `std_logic`
Debug	`Reg_DBusOut`	out `std_logic_vector(0 to 31)`
	`Reg_Read_Valid`	in `std_logic`
	`Reg_Write_Valid`	in `std_logic`
	`Reg_ABus`	in `std_logic_vector(0 to ETH_DCR_ADDR_BUS_WIDTH-ETH_DEVICE_SELECT_WIDTH-1)`
	`Reg_DBusIn`	in `std_logic_vector(0 to 31)`

The entity operates synchronous to the GBIF clock that should be provided in `clk`. All input signals are expected to be synchronous to this clock. At any cycle, the entity can be fully reset by asserting `sreset`. The entity is made up of several blocks of logic that work together:

- a simple FSM,
- a Request Poster, i.e., logic requesting outbound read transfers
- a Data Feeder, i.e., logic that forwards data of completed requests to the FIFO,

- an Interrupt Module,
- an Outbound Read Latency Evaluation Module,
- an Outbound Read Transfer Monitor

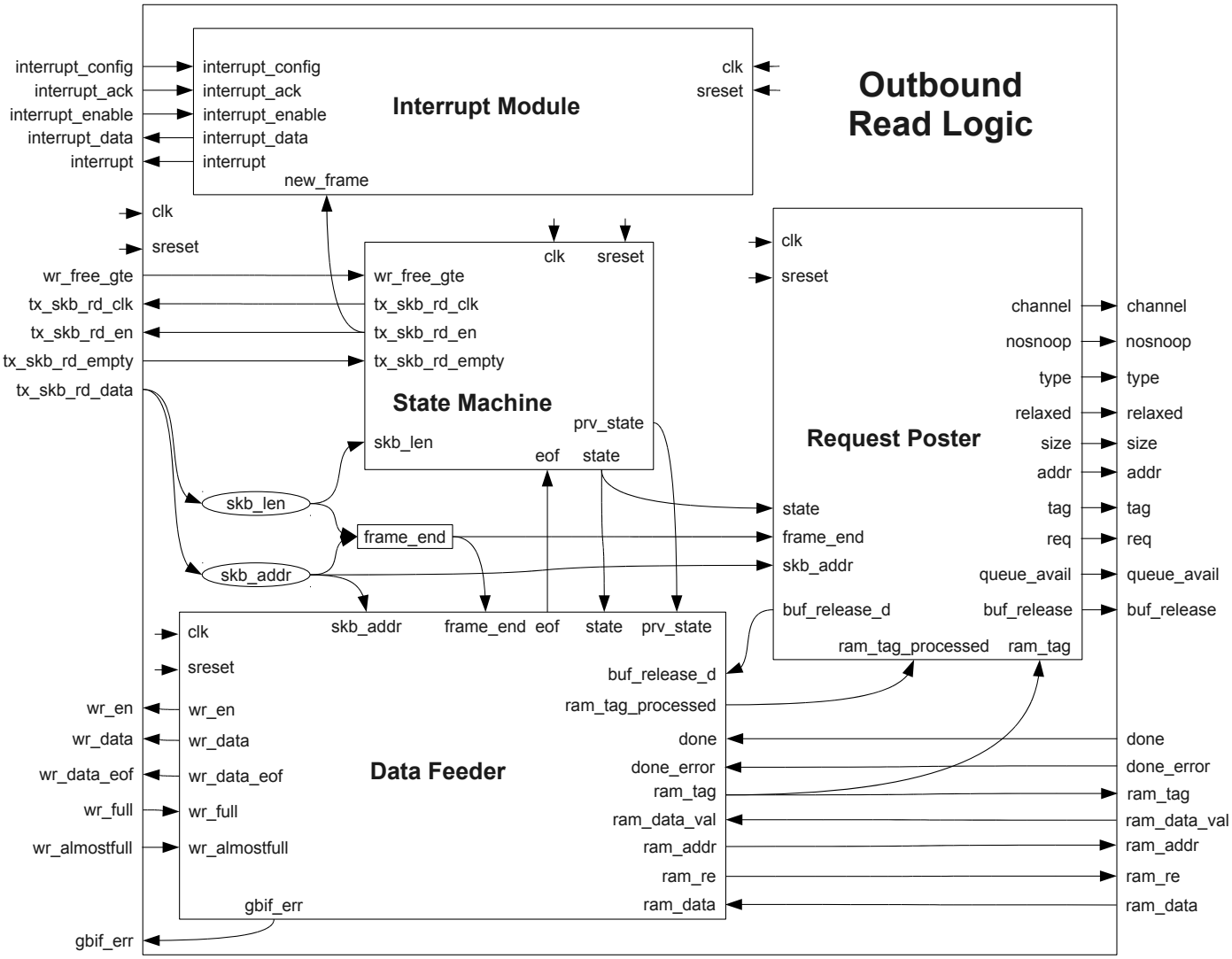


Figure 4.8: Block Diagram of the Ethernet Outbound Read Logic. The Latency Evaluation Module and the Transfer Monitor are not show here for the sake of clarity.

We show a block diagram of the logic in figure 4.8. The FWFT Address FIFO provides the Outbound Read Logic with the first frame instructions in `tx_skb_rd_data` when is deasserts `tx_skb_rd_empty`. The instructions are decoded into frame length and physical frame address. Further the physical address of the last frame byte is computed.

We start with a description of the FSM, see figure 4.9. After reset, the FSM is in the WAIT state. Its input signals `wr_free_gte` and `skb_len` are evaluated in `wr_free_frame` to see whether there is enough free space in the TX FIFO to hold the full frame. This signal can be configured to be ignored, for example if more Ethernet performance is required, by setting the VHDL parameter `ETH_OUT_RD_NO_TAG_BLOCK=false`. When `skb_len` is valid, i.e., `tx_skb_rd_empty` is false, and there is enough space for the frame, the machine switches

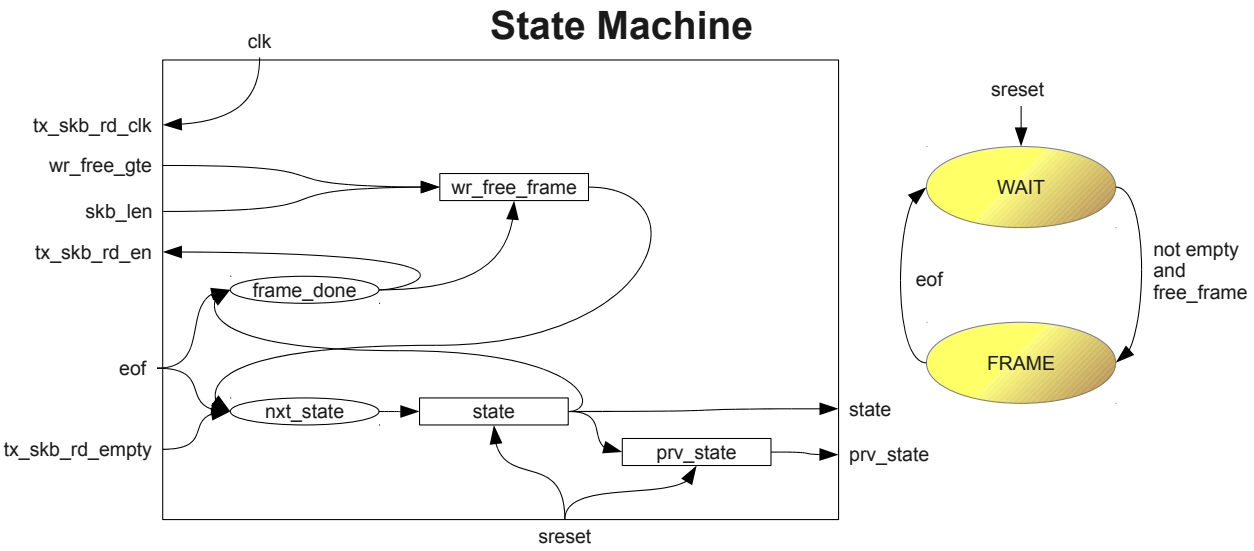


Figure 4.9: Block Diagram and State Diagram of the Ethernet Outbound Read FSM.

into the FRAME state. This state remains active until an end of frame pulse in `eof` is received. A read-enable is sent to the Address FIFO to inquire the next frame instruction and `wr_free_frame` is reset to zero. The machine now returns back to the WAIT state. At any cycle, the current state and the previous cycle state are being provided to adjacent logic.

We now describe the Request Poster, see figure 4.10. The `frame_end` signal provides enough information to decode the number of blocks of `0x80` bytes that need to be transferred into `tags`. When a WAIT state is supplied `tag` and `pre_req` are reset to 0 and `current_tag_avail` and `not_last_block_q` to 1. During the FRAME state, `req_pre` is being asserted when the current tag is available (`current_tag_avail`), there is no current request (`req`) and we have not yet requested all necessary blocks of data (`not_last_block_q`). Altogether, this is when we would like to request a new transfer. Once the queue becomes available (`queue_avail`), we forward (in the same cycle) the request for a transfer with a tag `tag` to the Slave GBIF (SGB) together with a properly calculated transfer address `req_addr`. This address is calculated from the current tag number in `tag` and the start address of the frame (`skb_addr`). At any request posted by `req`, we re-evaluate whether we have requested the final block in `not_last_block_q`. Also, at any request posted or some tag was made available (`ram_tag_processed`), we re-evaluate if the current `tag` is not currently occupied by the Data Feeder (`ram_tag`) into `current_tag_avail`. Additionally, at any cycle, we release any tag `ram_tag` made available by the Data Feeder to the SGB using `buf_release`.

We now describe the Data Feeder, see figure 4.11, a rather challenging logic at small footprint to invent. One of the main goals was to get away without instantiating any frame data registers or pipeline but to still be operable at high clock frequencies. While the WAIT state is supplied, `done`, `ram_tag_ready`, `ram_addr`, `ram_tag`, `wr_en_q`, `eof_q`, `sof`, `ram_tag_processed` are fixed to 0, `wr_en_suppress` is fixed to 1, only the total number of 16 byte words `lines` that need to be transferred on the full frame is updated according `frame_end`. Therefore no FIFO writes are instructed in `wr_en` such that the value of `wr_data_eof` will be irrelevant. During the FRAME state however, the signal `done` is constantly updated by flagging tags whose data has been provided `al_done` by the SGB and unflagging tags whose data has been successfully sent (`buf_release_d` to the TX FIFO). Once the data of current tag `ram_tag` has been made available as indicated by `done` and there is enough space available in TX FIFO as indicated by

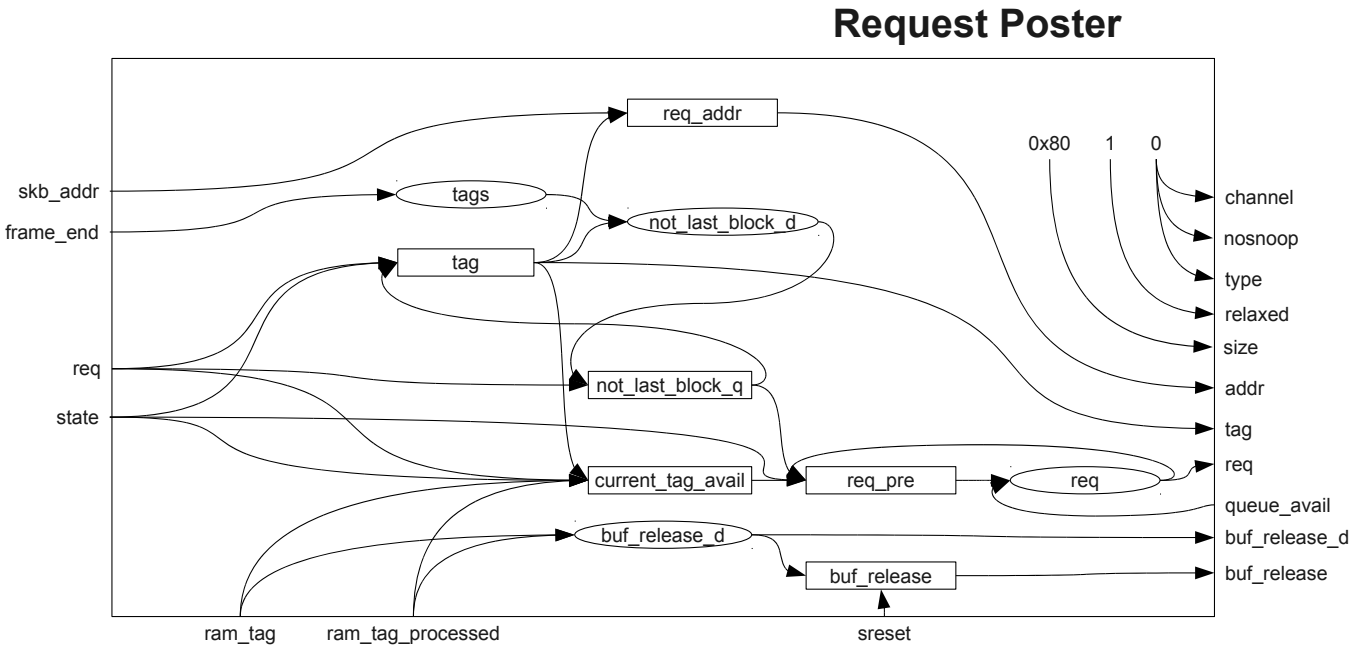


Figure 4.10: Block Diagram of the Outbound Read Request Poster. Transfer requests are posted to the System Bus, transfer tags managed and released.

`not wr_almostfull`, we are ready to transfer the current tag and assert `ram_tag_ready`. Once the tag is ready to be transferred we assert the read-enable for the data provided `ram_re`. Once all necessary data of the tag has been transferred, we create a pulse in `ram_tag_processed`. In turn signals `ram_re` and `ram_tag_ready` are being deasserted and `ram_tag` increased (in order to prepare for the next tag). The tag transfer actually completes, when the current tag buffer subaddress `ram_addr` reaches the last line of the tag or an end of frame (`eof`) is detected. An end of frame is detected when the total number of lines that remain to be transferred `lines` reaches 1 (`eof_q` asserts) and a new line is actually provided (`ram_data_val`) by the SGB in `ram_data`. For each line provided we decrement the number of lines still to be transferred (`lines`) and assert then FIFO write-enable signal `wr_en_q`. This write-enable signal is only forwarded to the FIFO if it is not suppressed by `wr_en_suppress`. The enabled signal needs to be suppressed in cases where the Ethernet frame start is not aligned to a `0x80` Byte boundary in the physical address space. In such a case we need to skip over the irrelevant data until we encounter the start of the frame. During this period, where we assert `first_lines`, we suppress any write-enables. Of course, when `wr_en` is asserted, the data in `ram_data` is written to the TX FIFO. In the same cycle, we need to supply parity bits in `wr_data_eof` as well, defined in section 4.1.5.1.1, to mark frame boundaries. These boundaries are determined from the signals `eof` and start of frame (`sof`). At any time the proper frame boundary parity bits are decoded from `skb_addr` and `frame_end` into `wr_data_sof_d` and `wr_data_eof_d` to be multiplexed to the parity bits `wr_data_eof` when it is time. The start of frame signal `sof` is asserted at the first unsuppressed write-enable signal only.

We describe the Interrupt Module in section 4.1.5.4.

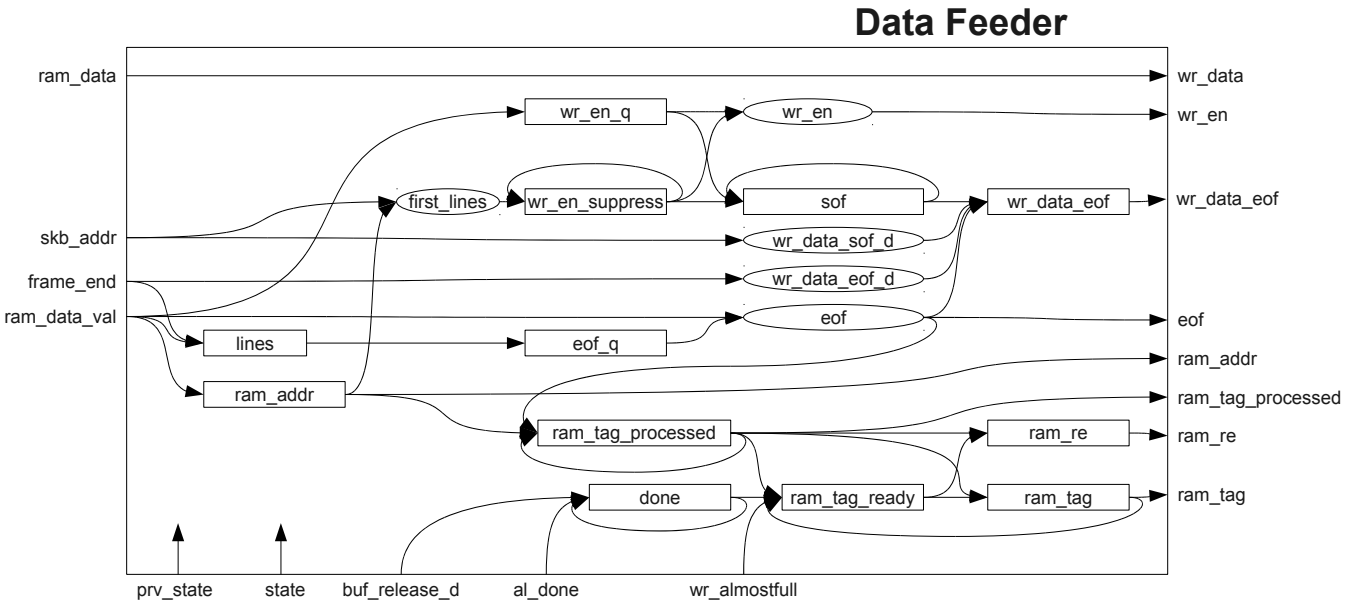


Figure 4.11: Block Diagram of the Outbound Read Data Feeder. Data written by the System Bus is reorganized into our frame stream format and fed into FIFO.

4.1.5.3 Outbound Write Logic

The Outbound Write Logic is in charge of transferring frame data from the Ethernet RX FIFOs to main memory using Outbound Writes, based on addresses provided by the RX Address FIFO. Depending on the configuration provided at run-time, interrupts shall be issued and status reported. We use the following entity interface:

Signal Type	Signal Name	Description
Common	`clk`	in `std_logic`
	`sreset`	in `std_logic`
Outbound Write Controller	`owc_out`	in `owc_out_type`
	`owc_in`	out `owc_in_type`
Receive FIFO	`rd_en`	out `std_logic`
	`rd_data`	in `std_logic_vector(0 to 127)`
	`rd_data_eof`	in `std_logic_vector(0 to 15)`
	`rd_empty`	in `std_logic`
	`rd_almostempty`	in `std_logic`
	`rd_drop`	in `std_logic`
	`rd_accept`	in `std_logic`
	`rd_new_frame`	in `std_logic`
Interrupts	`interrupt`	out `std_logic`
	`interrupt_ack`	in `std_logic`
	`interrupt_enable`	in `std_logic`
	`interrupt_data`	out `std_logic_vector(0 to 31)`
	`interrupt_config`	in `std_logic_vector(0 to 2)`
Address FIFO	`skb_rd_clk`	out `std_logic`
	`skb_rd_en`	out `std_logic`
	`skb_rd_data`	in `std_logic_vector(0 to 63)`
	`skb_rd_empty`	in `std_logic`

Control Buffer Configuration	`ctrl_buf_addr`	in `std_logic_vector(0 to 63)`
	`ctrl_buf_lines`	in `std_logic_vector(0 to 7)`
	`ctrl_buf_we`	in `std_logic`

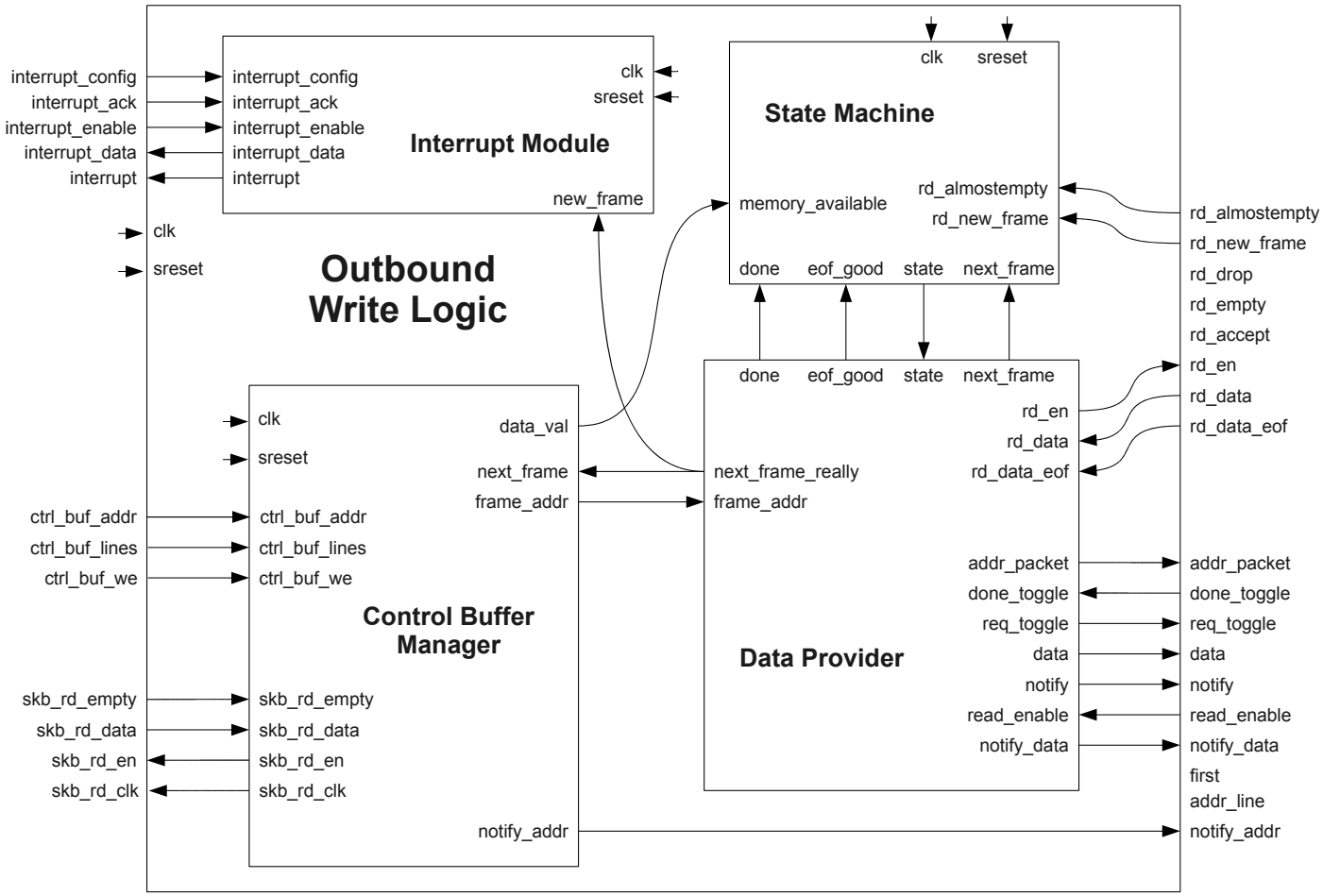


Figure 4.12: Block Diagram of the Ethernet Outbound Write Logic.

We now describe the Outbound Write Module, see figure 4.12. The module is built up from 4 logic blocks. One of them is the Outbound Write FSM, see figure 4.13. After reset the machine enters the IDLE state. We set the number of available frames (`available_frames`) to 1. We decrement its value for each completed transfer (pulse in `next_frame`) and increment its value for every new frame in the Ethernet RX Buffer (pulse in `rd_new_frame`). Since the Outbound Write Data Provider indicates its ability to proceed to the next frame (using `next_frame`) we set the frame counter to 1 to get even (i.e., 0) after reset. During operation, we determine if there is data available (`data_available` high) to be transferred, which is the case when the TX Data Buffer is not almost empty (`rd_almostempty`) or when we are sure there is actually a full frame in the FIFO `available_frames`. Additionally, we need to make sure we have main memory buffer space available (`memory_available`) before the machine enters the REQUEST state. We also enter the REQUEST state when we receive the `eof_good` signal, as signal that indicates that we successfully completed the transfer of the frame data and that the Outbound Read Data Provider wants to send a notification to main memory

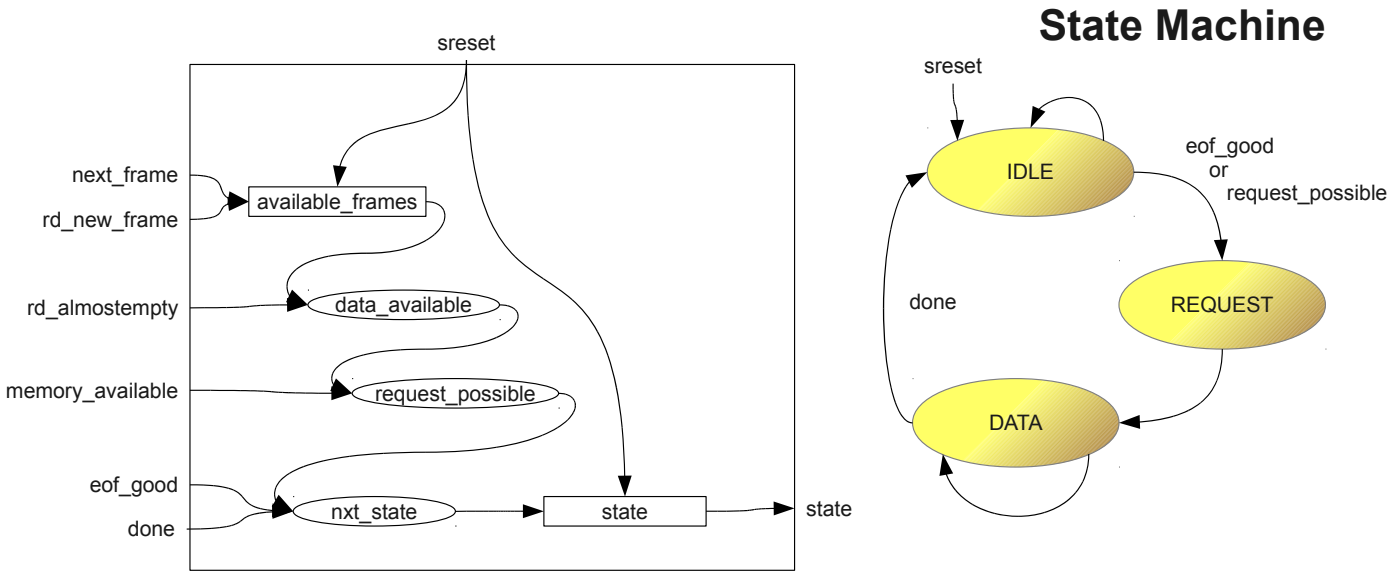


Figure 4.13: Block Diagram of the Ethernet Outbound Write FSM.

about this and no additionally data or main memory credits are necessary. The REQUEST state is left for the DATA state on the next cycle. The REQUEST state is used to place a request to the OWC on the data that needs to be transferred. During the DATA state, the OWC then reads all data (which is provided by the Data Provider Module). Once we receive a completion notification by a pulse in `done`, we switch to the IDLE state.

We now describe the Control Buffer Manager, see figure 4.14. In order for the FSM to kick-off a transfer, it needs to be told whether there is MM buffer space available. The Control Buffer Manager provides this information in `data_val` along with the Control Buffer base address in `notify_addr`, the Frame Data Buffer base address in `frame_addr`. It advances to the next control information on a pulse in `next_frame`. The Control Buffer Manager contains registers that hold the address and length of current Control Buffer being active, as well as address and length of the control buffer to be used once the current control buffer is depleted. During reset, both sets of registers contain invalid data. This is flagged by `ctrl_val`, `next_ctrl_val`, `data_val`, `next_frame_really` and `ctrl_end` being deasserted. During operation control buffer address `ctrl_buf_addr` and `ctrl_buf_lines` and length can be written to the module using a pulse in the control buffer write-enable `ctrl_buf_we`. Any next control buffer information in `next_ctrl_addr` or `next_ctrl_lines` is going to be overwritten and its new information marked as valid by asserting `next_ctrl_val`. At any point in operation the current control buffer information is invalid (e.g. after reset) the next control buffer information is forwarded to `ctrl_addr` or `ctrl_lines` during the cycle, flagged as valid, and `next_ctrl_val` deasserted if no new control buffer is supplied. While the current control buffer is invalid the current word (16 Byte) number (or line number) addressed (`ctrl_line`) is held to 0. We provide the full control buffer address to adjacent logic in `notify_addr`. When it is time to advance for one frame `next_frame_really` pulsed. We use this pulse to pull the next destination physical address from the RX Address Buffer using `skb_rd_en`, to increment the current line counter and to deassert the data valid output (`data_val_q`). At any cycle in operation, we need to evaluate if we depleted the control buffer and assert `ctrl_end` appropriately in the next cycle. If it is depleted and we like to advance to the next frame (`next_frame_really`), we remove the current control buffer by deassert `ctrl_val` for

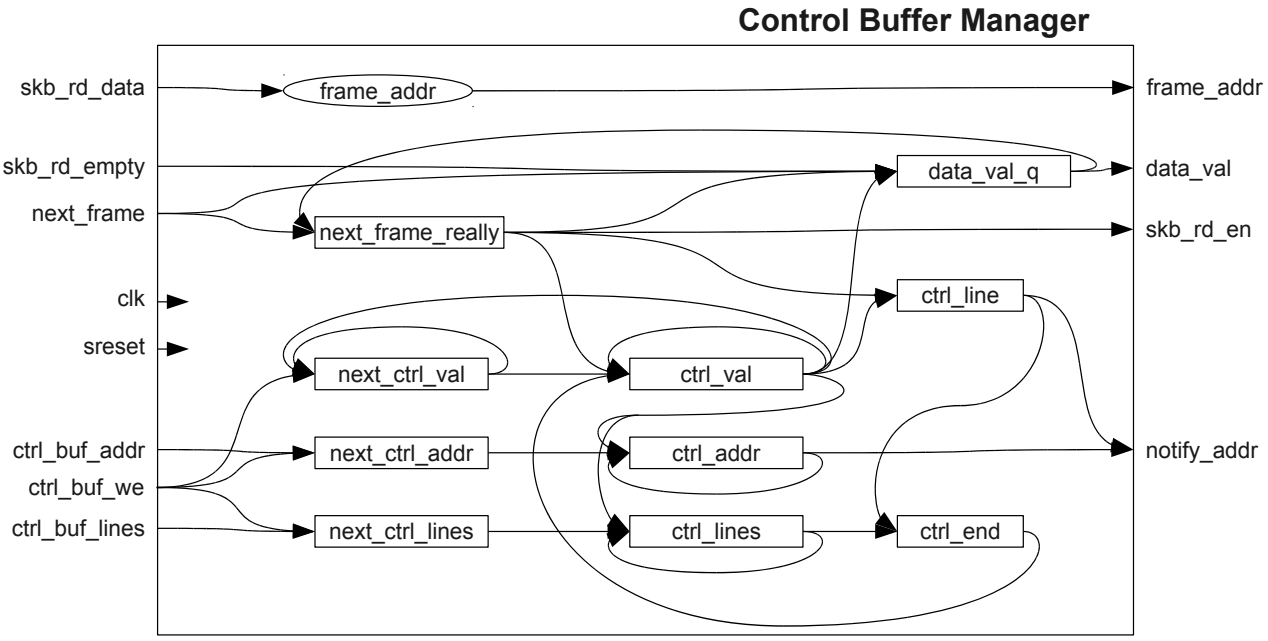


Figure 4.14: Block Diagram of the Ethernet Outbound Write Control Buffer Manager.

the next control buffer information to be forwarded. At the point where the control buffer information is valid and the frame buffer information is valid (not `skb_rd_empty`), we assert `data_val_q` to show the validity of all outputs. Once the adjacent logic demands the next information on the control buffer, it pulses `next_frame`, which can only be righteous if we currently provide valid data at the output (`data_val_q`). In this case we really advance to the frame (`next_frame_really`).

We now describe the Data Provider, see figure 4.15. The Data Provider interfaces to the RX Data Buffer and the OWC. There are multiple clients in the NWP that connect to the OWC. We therefore created a standardized interface that fits our need. The interface was originally defined and documented in [31], but modified later due to a change in requirements. The OWC supports `0x80` Byte transfers and `0x10` Byte transfers.

Signal Name	I/O	Description
`req_toggle`	I	toggle signal for a new request
`done_toggle`	O	toggle signal to indicate a request was dealt with
`first`	O	indicates to a slave that the first line of data has to be provided on the next clock. Note that 0 is driven on `addr_line` one clock before first (that is `read_addr_line` is 1 together with first).
`read_enable`	I	enabled for every cycle `data` is read by the OWC
`addr_line(0 to 2)`	O	line of the packet
`data(0 to 127)`	I	data line corresponding to `addr_line` driven two cycles before
`addr_packet(0 to 41)`	I	address to be used as address on GBIF. Must be valid with `req_toggle` until `done_toggle` (that is from toggling `req_toggle` until `done_toggle` is toggled).

notify	I	asserted to request sending of a notification for current request. Must be high with req_toggle until done toggle. The OWC acts differently on this signal, depending on the slave: For Torus Link Buffers the notification is sent right after the data packet of the current request was sent. For Ethernet no data package for the current request is sent, but only a notification.
notify_addr(0 to 127)	I	address for notifications. Must be valid with req_toggle until done toggle.
notify_data(0 to 41)	I	data for notifications. Must be valid with req_toggle until done toggle. Currently defined to be constant.

We show a timing diagram of the OWC slave in figure 4.16. During reset, we fix the registers read_enable_delay, read_enable_delay2, rd_en_q, done, next_frame_really to 0 and the register next_frame to 1. The signal next_frame has essentially a reset signal to many other registers: Any assertions sets notify, eof, eof_good, eof_bad, page, frame_lines, frame_trail and frame_lines_inc back to 0. For example, this happens in the first cycle after reset.

To initiate a Outbound Write Transfer from the RX Data FIFO into main memory, we flip req_toggle every cycle that we are in the REQUEST state. We provide the physical address of the part of the Ethernet frame to be transferred to main memory in addr_packet, i.e., the frame base address frame_addr (that we demand to be 0x80 Byte aligned) plus the page times 0x80. The FSM switches to the DATA state in the next cycle. Once the OWC starts with reading the data it asserts the read_enable signal. According to timing diagram in figure 4.16, we need to provide first data two cycles later. We therefore delay the read-enable signal in two stages to read_enable_delay and read_enable_delay2. Since the FWFT FIFO already provided the first data word in rd_data, we demand the delivery of the next word from the RX FIFO by asserting rd_en_q for every one-cycle-delayed read-enable signal until we receive an end of frame signal from the parity bits of the FIFO in rd_data_eof(15). Two things can happen now. When the frame extends over the next 0x80 Byte boundary another Outbound Write transfer needs to be initiated and we stop the reading of data from the TX FIFO once there is a falling edge in read_enable_delay, for this we need read_enable_delay2. Additionally we suppress any read-enable when an eof-of-frame was found (eof) in the previous FIFO word. This flag register is updated each cycle a read-enable was issued. We employ two other flag registers: We create a good end of frame signal eof_good that asserts when the end of frame is found to be the end of a valid frame, i.e., when rd_data_eof(13) or (14) are asserted according to out parity bit definitions from section 4.1.5.1.1. We also create a bad end of frame signal eof_bad that asserts when the end of frame is found to be on the end of an invalid frame, i.e., when rd_data_eof(13) and (14).

At this point the module waits for a toggle in the done_toggle signal. Such a transition indicates that the OWC read all data from data and that it is ready to accept new instructions. We decode such a transition into a pulse in done_strobe and register the signal in done. This signal, as well as eof_good, is supplied to the adjacent Outbound Write FSM. The done advances it to IDLE. Once enough memory is available it further advances to the REQUEST state and then the DATA state such that the transfer mechanism describe above loops. After each transfer (done) we increment page to properly concatenate transfers in main memory. The loop is broken once we encounter an end of frame signal eof.

In case a valid frame end was encountered (eof_good) the FSM advances to the REQUEST

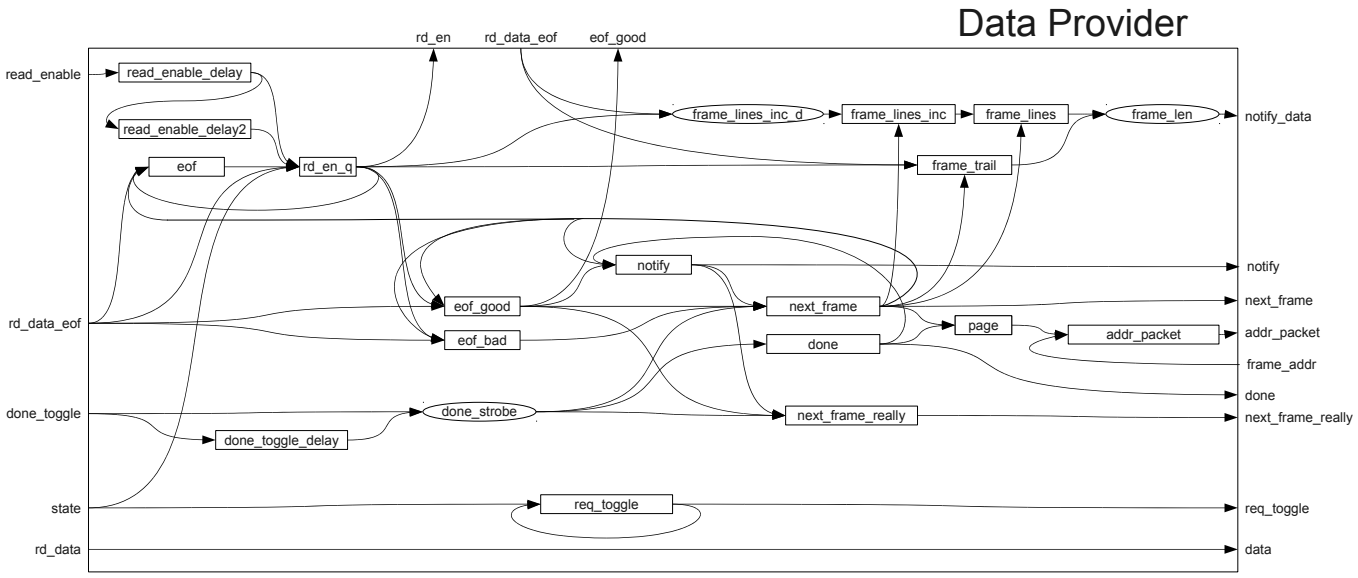


Figure 4.15: Block Diagram of the Ethernet Outbound Write Data Provider.

state. This time, we use the strobe[2] in `done` to instruct the next transfer to be a "notification" to the OWC by asserting `notify` during the toggling in the `req_toggle` signal. A notification is a single word (16 Byte) transfer (and an in-order transfer). The data in `notify_data` is instructed to be written to the physical address `notify_addr` (supplied externally to the OWC). No read-enables are issued by the OWC and thus no further data is read from the FIFO as it should be. At this point the module again waits for a strobe in `done_strobe`. If the strobe coincides with a notification enabled (`notify`), we have completed the transfer of the current frame and indicate this to adjacent logic through `next_frame_really`. We also use this event to assert the internal `next_frame` signal to reset the logic for the next frame to be transferred.

In the other case where an invalid frame end was encountered, we do not indicate this to adjacent logic but instead reset the logic (using `next_frame`) for the next frame to be transferred. This way the bad data in the buffer in main memory is being overwritten with new (most likely) valid frame data.

Finally we need to explain how the data that is sent with a notification is determined. We send a 16 bit word containing the byte size of the frame to main memory. For this reason we count the number of words read from the FIFO for the current frame in `frame_lines`. We increment its value for each cycle `frame_lines` is asserted. We assert this register each cycle we read from the FIFO and the parity bits indicate that all byte belong to the current frame, i.e., `rd_data_eof=0x0001 or rd_data_eof(15)=0`, i.e., we count full lines. At the same time we decode the number of bytes of the current data word that belong to the current frame from `rd_data_eof` and store this value in `frame_trail`. At the end of a frame no new data is read from the FIFO such that `frame_lines` and `frame_trail` keep their value until the next frame (`next_frame`) and we can combine it them to the full frame length and provide it for the a possible notification to be sent.

4.1.5.4 Interrupt Module

[2] a pulse, a single cycle assertion

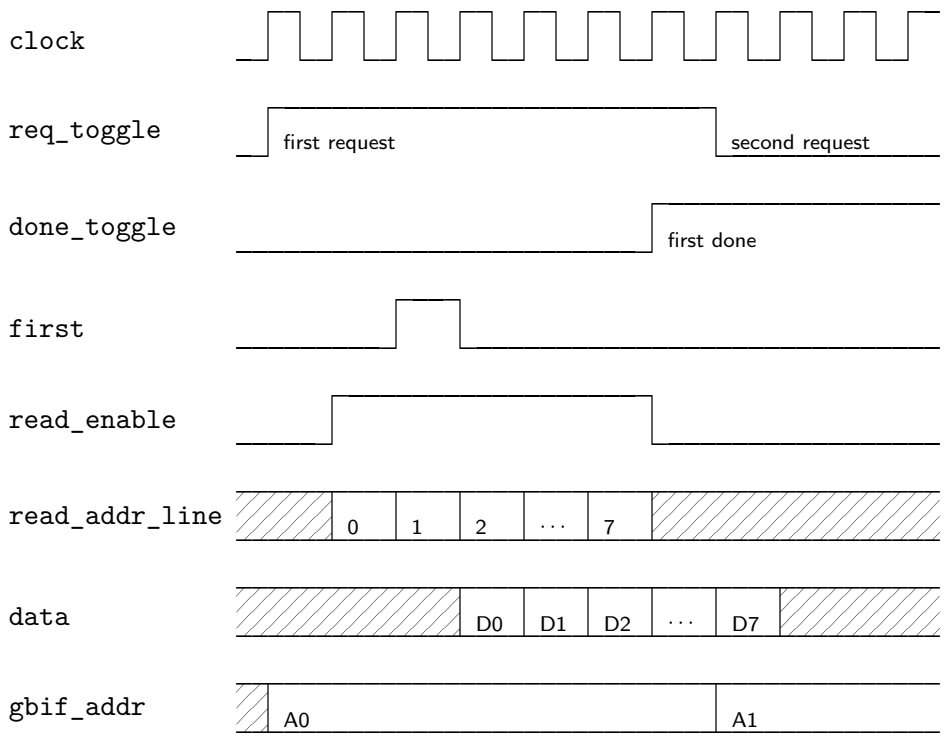


Figure 4.16: Timing diagram of the OWC slave interface

We now describe the Interrupt Module, see figure 4.17. While in reset, `interrupt_q` and `frames` are set to zero. For each frame we transfer (pulse in `new_frame`), we increment `frames`. Once the number of `frames` reaches a certain threshold `limit`, as defined by `interrupt_config`, we assert `interrupt_q` to flag completed frames to other logic. We provide the number of completed frame and the current interrupt state in `interrupt_data`. In the same cycle `interrupt_data` is acknowledged by other logic, `interrupt_ack` needs to be asserted and the `frames` counter will be cleared along with the interrupt line being deasserted.

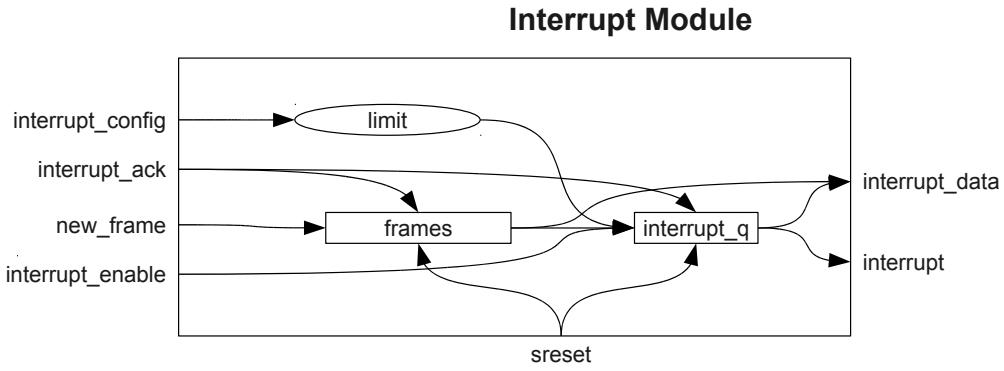
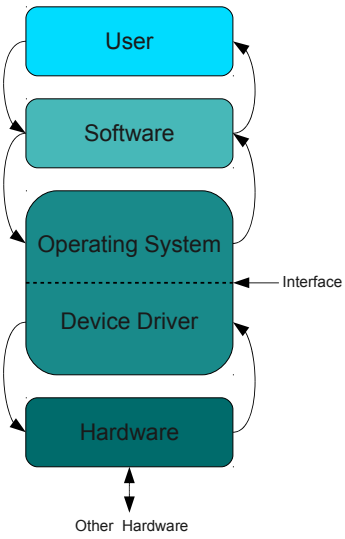


Figure 4.17: Block Diagram of the Ethernet Interrupt module, that is used in the Outbound Read and Outbound Write logic.

4.2 Software Components

Every special hardware component needs some special software that allows its operation. For the sake of flexibility, current computer technology employs the concept of device drivers. There is usually some kind of operating system, (e.g. Linux), that provides standardized interfaces (e.g. the Linux kernel network interface). Depending on the hardware in operation, matching software (e.g. the QPACE Ethernet Driver for Linux) takes commands and data through this interface to operate the device. On the other end of the operating system, any type of higher software (e.g. a web browser), eventually controlled by some user (e.g. you), can now use the hardware on a common basis. Therefore, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device. Also, drivers are obviously hardware-dependent and operating-system-specific.



4.2.1 Linux Device Driver

In this section we describe the operation of the QPACE Ethernet device driver for Linux. It is licensed under GNU General Public License (GPL) v2. We currently support Linux kernels supported up to version 2.6.31. At the time of writing the most recent driver version was v0.5.4. For a general introduction to Linux device drivers, Linux network drivers and their nomenclature we refer to [32]. This book may be seen as a prerequisite to this section. The Ethernet core was designed to be operated easily through a single interface that is DCR. The register interface is explained in section 4.1.3.2, 4.1.3.1 and 4.1.3.3. The driver only needs to set up a few address spaces for the core to push and pull the frame data. In this section we explain how this is done. QP The driver is written in the C programming language consits of a 3 parts:

1. a kernel configuration file (`drivers/net/qpace/Kconfig`), defining the name and type of the driver to the kernel build system,

2. a make-file for the kernel build system (`drivers/net/qpace/Makefile`), specifying which files belong to the driver source code,

3. the driver source code (`drivers/net/qpace/qpace.c`) and its header file (`drivers/net/qpace/qpace.h`).

4.2.1.1 Driver Structures

The C header file `qpace.h` defines a number of parameters, enumerations and structs used by the driver to manage its status and data. We start with an explanation of most parameters

Parameter Name	Description
`MAX_TX_SKBS`	Maximum number of frame addresses to be put in the TX queue of core. Hardware supports up to 512.
`MAX_RX_SKBS`	Maximum number of frame addresses to be put in the RX queue of core. Hardware supports up to 512.
`QPACE_AUTONEG_TIMEOUT`	Configures the driver to wait at most 10000 ms for the Ethernet PHY to complete auto-negotiation after cable detection by default.
`DEFAULT_TX_WATCHDOG_TIMEOUT_MSEC`	Configures the kernel to wait at most 1000 ms for the Ethernet driver to complete frame transmission before requesting the driver to resolve any issue.
`DEFAULT_RX_SKBS`	Number of Socket Buffers (for frames) the driver prepares for data to be put in by the kernel
`DEFAULT_RX_CTRL_LINES`	Number of lines (16 Byte) any RX control buffer will offer.
`DEFAULT_SKB_SIZE`	Size of any Socket Buffer to be allocated. Should be at least 16kB to support jumbo frames. Is 64kB by default.
`DEFAULT_WEIGHT`	Parameter to the kernel for Interrupt Mitigation. Set to 256 by default.
`RX_SKB_ALIGN`	Parameter to the driver to align the data section of any Socket Buffer allocated on address boundaries. Core needs `0x80` minimum.
`MULTICAST_TABLE_SIZE`	Defines the number of multicast addresses the EMAC in the core should handle. Hardware supports up to 4.
`DEFAULT_LD_MAX_RX_PKT`	Configures the core to issue an RX interrupt each $2^{\texttt{DEFAULT\_LD\_MAX\_RX\_PKT}}$ frame received. Values different from 0 are not recommended.
`DEFAULT_LD_MAX_TX_PKT`	Configures the core to issue an TX interrupt each $2^{\texttt{DEFAULT\_LD\_MAX\_TX\_PKT}}$ frame sent. Needs to be tuned for best performance versus memory footprint. Value of 3 was found to be useful.

and continue with most enumerations

Parameter Name	Description
`QPACE_DCR_*`	Addresses of all DCR registers available with respect to the DCR base address of the Ethernet device.

MARVELL_PAG_*, MARVELL_REG_*	Addresses and pages of all operationally relevant registers within the MARVELL Ethernet PHY attached.
MARVELL_INT_*	Naming of events that the MARVELL Ethernet PHY may issue an interrupt to notify the driver.
QPACE_NWP_RESET_*	Addresses of all DCR registers available for resetting devices within the NWP with respect to the DCR base address of the Reset Controller device.
ETH_HOST_ADDR_*	Addresses of all operationally relevant registers within the XILINX Trimode EMAC within the core. See also [29].
ETH_CONFIG_*	Bit numbers of all available resets in the QPACE_ETH_CONFIG configuration register that control relevant parts of the Ethernet Core.
PHYAD_*	Addresses of all available devices on the MDIO Bus. For QPACE there is only one PHY.
ML_*	Proprietary definitions for the message levels (or verbosity) the driver. Relevant for debugging purposes.

The driver stores all of its data in a structure called qpace_device. In Linux a hardware driver is only instantiated once. However it may be called to operate different devices within the same machine (for QPACE there is of course only one Ethernet device), which is commonly specified by a net_device argument to most kernel interface functions available. We employ the following data structure:

Member Type Member Name	Description
net_device * ndev	Pointer to the network device structure provided to the kernel.
struct net_device_stats stats	Instance of a default kernel network device statistics structure to be passed to the kernel upon request.
struct of_device * ofdev	Pointer to the Open Firmware Device structure provided for for device handling by the kernel upon initialization.
struct napi_struct napi	Instance of a Network API (NAPI) interface, supports interrupt mitigation etc...
const struct ethtool_ops * ops	Pointer to an instance of a structure listing all callable ethtool functions to the kernel.
dcr_host_t dcr_host	Handle of the QPACE Ethernet DCR Host
dcr_host_t dcr_host_reset	Handle of the QPACE Reset Controller Host
void __iomem * io_base	Base Address of the QPACE Ethernet Address Space. Fetched from Open Firmware but unused since core fetches and delivers all data using DMA.
int irq_tx	Number of the TX Interrupt. Fetched from Open Firmware.

int irq_rx	Number of the RX Interrupt. Fetched from Open Firmware.
int irq_slow	Number of the "Slow" Interrupts. Contains all interrupts from PHYs. Fetched from Open Firmware.
spinlock_t lock_slow	Spin-Lock handle with respect to Slow Interrupts.
spinlock_t lock_rx	Spin-Lock handle with respect to RX Interrupts.
spinlock_t lock_tx	Spin-Lock handle with respect to TX Interrupts.
struct qpace_rx_buffer rx_buf	Instance of the proprietary struct handling all RX functionality
struct qpace_tx_buffer tx_buf	Instance of the proprietary struct handling all TX functionality
unsigned int ld_maxrxpkt	Variable holding number of received packets before any RX Interrupt is issued by core.
unsigned int ld_maxtxpkt	Variable holding number of transmitted packets before any TX Interrupt is issued by core.
u32 msglvl	Variable holding current Message Level.
u32 slow_int_mask	Variable holding enable mask to apply to the Slow Interrupt Module within the core. Used to enable or disable PHY interrupts.
int mdio_page	Variable holding the current page to registers within the Ethernet PHY.
int autonegretry	Counter cycling through autonegotiation modes until a successful one was encountered.

We employ the following RX Buffer structure `qpace_rx_buffer`:

Member Type Member Name	Description
struct sk_buff * skbs[MAX_RX_SKBS]	Ring-Array of pointers to socket buffers ready to be filled with data by core on reception.
unsigned long skbs_bus[MAX_RX_SKB]	Ring-Array of physical addresses of first data within receive socket buffers.
int skb	Index of current receive socket buffer.
int skb_count	Number of receive socket buffers allocated.
u16 * ctrls[2]	Pointer to two control buffers.
unsigned long ctrls_bus[2]	Physical address of two control buffers.
int ctrl_toggle_bit[2]	Current toggle-bit value for each Control Buffer.
int ctrl	Index of current control buffer.
int entry	Index within current control buffer.
int entries	Allocated number of entries each control buffer.

`int frames_waiting`	Number of received frames waiting in memory to be handed over to the kernel.

We employ the following TX Buffer structure `qpace_tx_buffer`:

Member Type Member Name	Description
`struct sk_buff* skbs[MAX_TX_SKBS]`	Ring-Array of pointers to socket buffers instructed to be sent by the kernel.
`unsigned long skbs_bus[MAX_TX_SKBS]`	Ring-Array of physical addresses of socket buffers instructed to be sent by the kernel.
`unsigned int skbs_bus_len[MAX_TX_SKBS]`	Ring-Array of lengths of payload data of socket buffers instructed to be sent by the kernel.
`int head`	Index within ring-array for next socket buffer instructed to be sent by the kernel.
`int tail`	Index within ring-array of next socket buffer to be freed upon transfer completion interrupt from core.
`int free`	Number of free slots in the ring-array.

4.2.1.2 Driver Functions

The C code file `qpace.c` contains all kernel interface, ethtool interface and auxiliary functions necessary for a Linux network driver to operate properly. For each of the functions we list its definition and describe its effect.

- `static int __init qpace_init(void)`
  Initializes the module by registering the driver to the Open Firmware Platform, supplying the driver name, device match properties `qpace_match` and kernel interface functions `qpace_remove` and `qpace_probe`.

- `static void __exit qpace_exit(void)`
  Deinitializes the module by unregistering the driver to the Open Firmware Platform.

- `static int qpace_probe(struct of_device *ofdev,`
  `const struct of_device_id *match)`
  Called on start of the driver. Allocates network device structures, sets standard startup values, fetches MAC address and IO base address from Open Firmware and maps it, fetches DCR register spaces and maps them, provides network device functions to the kernel, provides ethtool interface functions to the kernel, provides NAPI functions to the kernel, fetches RX, TX and SLOW interrupt numbers from Open Firmware, initializes spin-locks, forces full reset cycle on hardware, sets MAC address to the EMAC, fixes MDIO frequency, enables selected PHY interrupts, disables link (carrier) to the kernel, disables link to the PHY, requests SLOW interrupt handling to the kernel providing an interface function, configures PHY registers for selected mode of operation and finally registers network device to the kernel.

- `static int qpace_remove(struct of_device *ofdev)`
  Called on removal of the driver. Disables SLOW interrupts, puts hardware core into

reset, unregisters the network device to the kernel, unmaps mapped DCR hosts and frees all memory space allocated.

- `static int qpace_open(struct net_device *ndev)`
  Called when the kernel opens the network device to get ready for communication. Request RX and TX interrupt handling to the kernel providing interface functions for each. Allocates the driver's RX buffers using `qpace_alloc_rx_buffers`, takes remaining logic within core out of reset. Initializes TX Buffer's `head`, `tail`, etc..., configures Outbound Write Logic, posts all RX socket buffers to the core, posts first control buffer to the core, instructs the PHY to enable the link, starts the kernel network interface transmit queue, enables the use of NAPI for receive interrupt mitigation, and finally enables RX and TX interrupts to be ready for operation.

- `static int qpace_stop(struct net_device *ndev)`
  Called when the kernel closes (or stops) the network device. Disables and free RX and TX interrupts, disables the NAPI, disables the carrier to the kernel, stops the kernel network interface transmit queue, instructs the PHY to disable the link, puts most logic within core into reset and frees driver's RX buffers.

- `static int emac_ready(struct qpace_device *qdev)`
  Uses DCR to read the current status of the EMAC Host Bus and returns true when it ready to accept a new command.

- ```
  u32 emac_read(struct qpace_device *qdev,
                unsigned int addr)
  ```
  Uses DCR to read the value of the EMAC register at address `addr`.

- ```
  void emac_write(struct qpace_device *qdev,
                  unsigned int addr,
                  u32 value)
  ```
  Uses DCR to write `value` to the EMAC register at address `addr`.

- ```
  u32 qpace_mdio_read_raw(struct qpace_device *qdev,
                          unsigned int phyad,
                          unsigned int regad)
  ```
  Read the value of MDIO register ad address `regad` (on the current page) in the PHY with MDIO address `phyad`. This command is instructed by the MDIO Master within the EMAC within the core, which is instructed using DCR.

- ```
  void qpace_mdio_write_raw(struct qpace_device *qdev,
                            unsigned int phyad,
                            unsigned int regad,
                            u32 val)
  ```
  Write the value `val` into the MDIO register at address `regad` (on the current page) in the PHY with MDIO address `phyad`. This command is instructed by the MDIO Master within the EMAC within the core, which is instructed using DCR.

- ```
  static void qpace_mdio_set_page(struct qpace_device *qdev,
                                  unsigned int page)
  ```
  Sets the currently active register `page` of the PHY.
  ```

- `u32 qpace_mdio_read(struct qpace_device *qdev,`
  `unsigned int phyad,`
  `unsigned int page,`
  `unsigned int regad)`
  Read the value of MDIO register ad address `regad` on page `page` in the PHY with MDIO address `phyad`.

- `void qpace_mdio_write(struct qpace_device *qdev,`
  `unsigned int phyad,`
  `unsigned int page,`
  `unsigned int regad,`
  `u32 val)`
  Write the value `val` into the MDIO register at address `regad` on page `page` in the PHY with MDIO address `phyad`.

- `static void qpace_enable_rx_irq(struct qpace_device *qdev)`
  Configures the Outbound Write Logic to issue interrupts.

- `static void qpace_disable_rx_irq(struct qpace_device *qdev)`
  Configures the Outbound Write Logic NOT to issue interrupts.

- `static void qpace_enable_tx_irq(struct qpace_device *qdev)`
  Configures the Outbound Read Logic to issue interrupts.

- `static void qpace_disable_tx_irq(struct qpace_device *qdev)`
  Configures the Outbound Read Logic NOT to issue interrupts.

- `static void qpace_enable_phy_irq(struct qpace_device *qdev)`
  Configures the SLOW Interrupt Module within the core to forward any PHY interrupts received.

- `static void qpace_disable_phy_irq(struct qpace_device *qdev)`
  Configures the SLOW Interrupt Module within the core to suppress any PHY interrupts received.

- `static u32 qpace_get_mii_speed(struct qpace_device *qdev)`
  Obtains the current MII link speed from the EMAC within the core.

- `static u32 qpace_get_mii_link(struct qpace_device *qdev)`
  Obtains the current MII link availability from the EMAC within the core.

- `int wait_for_link(struct qpace_device *qdev,`
  `int link,`
  `int ms)`
  Function that loops for at the most `ms` milliseconds until the EMAC within the core reports to have obtained `link` up or down.

- `static void qpace_set_emac_speed(struct qpace_device *qdev,`
  `u32 speed)`
  Instructs the EMAC within the core to interpret PHY signals with respect to the `speed` supplied.

- `static void qpace_set_phy_interrupts(struct qpace_device *qdev,`
  `unsigned int interrupts)`
  Instructs the PHY using MDIO to send selected `interrupts` to QPACE Ethernet.

- `static void qpace_reset_phy_interrupts(struct qpace_device *qdev)`
  Instructs the PHY using MDIO not to send any interrupts to QPACE Ethernet.

- `void link_down(struct qpace_device *qdev)`
  Instructs the PHY using MDIO to take the link offline.

- `void link_up(struct qpace_device *qdev)`
  Instructs the PHY using MDIO to (re)start auto-negotiation or simply enable the specified link mode.

- `static int qpace_set_settings(struct net_device *ndev,`
  `struct ethtool_cmd *cmd)`
  An ethtool interface function used to set the operational mode of the device and its driver.

- `static int qpace_get_settings(struct net_device *ndev,`
  `struct ethtool_cmd *cmd)`
  An ethtool interface function used to obtain the current operational mode from the device and its driver. For example ,this includes auto-negotiation modes, auto-negotiation results, driver capabilities, hardware capabilities, etc...

- `static void qpace_get_drvinfo(struct net_device *ndev,`
  `struct ethtool_drvinfo *info)`
  An ethtool interface function used to obtain the driver name, version and bus information.

- `static u32 qpace_get_msglevel(struct net_device *ndev)`
  An ethtool interface function used the obtain the current message level (i.e.) verbosity of the driver.

- `static void qpace_set_msglevel(struct net_device *ndev,`
  `u32 msglvl)`
  An ethtool interface function used to set the current message level (i.e., verbosity) of the driver.

- `static u32 qpace_get_rx_csum(struct net_device *ndev)`
  An ethtool interface function returning the current RX Checksum. Returns 0.

- `static void packet_dump(const char* txt,`
  `const char* prefix,`
  `char *data,`
  `u16 len)`
  A function using for debugging outputs a hexadecimal frame dump to standard kernel output.

- `static int qpace_tx_post_skb(struct qpace_device *qdev,`
  `struct sk_buff *skb)`
  The physical address of the frame data is posted together with the frame length to the Outbound Read Logic. A memory space containing the socket buffer supplied is mapped for DMA and its physical address determined. The TX spinlock is locked before modifying enlisting the socket buffer and the physical data address into TX ring-array, and adjusting `head` and `free` properly. The address and length is now posted to the Outbound Read Logic logic. The TX spinlock is unlocked.

- `static int qpace_hard_start_xmit(struct sk_buff *skb, struct net_device *ndev)`
  Called by the kernel to send the frame contained in the socket buffer supplied. Functions performs a few sanity checks on the frame length supplied and checks for free space in the driver's TX Buffer ring-array. It the array is full, the frame is returned to the kernel marking the driver as busy. If there is space, the physical address of the frame data is posted together with the frame length to the Outbound Read Logic, a transmission start time-stamp is memorized and statistics counters increased properly before returning a completion signal to the kernel. The socket buffer is now owned by the driver and is going be freed once the logic completes frame transfer into its TX FIFO.

- `static void qpace_pass_skb_up(struct qpace_device *qdev, struct sk_buff *skb, int size)`
  Completes the socket buffer with the frame length specified, increases RX statistics counter properly, shrinks the socket buffer to its minimum value and passes it on to the kernel.

- `static int qpace_renew_skb(struct qpace_device *qdev)`
  Unmaps the current RX socket buffer for DMA since it has already been passed to the kernel. Does not deallocate socket buffer, buf allocated new one fulfilling the alignment requirement of the Outbound Write Logic, maps it for DMA and replaces the old socket buffer entry in the ring-array with the newly created one.

- `static int qpace_ack_rx_irq(struct qpace_device *qdev)`
  Acknowledges the RX interrupt within the Outbound Write Logic, returning the number of frames completed since the last call.

- `static int qpace_ack_tx_irq(struct qpace_device *qdev)`
  Acknowledges the TX interrupt within the Outbound Read Logic, returning the number of frames completed since the last call.

- `static int qpace_ack_irq_phy(struct qpace_device *qdev)`
  Acknowledges the SLOW interrupt within the Ethernet PHY, returning type of PHY Interrupt.

- `static int qpace_slow_int_type(struct qpace_device *qdev)`
  Determines the type of the SLOW Interrupt within the Ethernet Core. Can be PHY or EMAC.

- `static int qpace_is_irq_phy(struct qpace_device *qdev)`
  Determines whether the Ethernet PHY has issued an interrupt. Used from the SLOW interrupt handler.

- `static irqreturn_t qpace_rx_irq(int irq, void *data)`
  Interrupt handler function for RX interrupts. Called by the kernel upon a RX interrupt. Locks the RX spinlock, disables RX interrupts and acknowledges the interrupt in the core returning the number of frame it transferred from the RX FIFO into memory. Finally a NAPI poll is scheduled, the RX spinlock unlocked and the interrupt returned as handled.

- `static irqreturn_t qpace_tx_irq(int irq, void *data)`
  Interrupt handler function for TX interrupts. Called by the kernel upon a TX interrupt.

Locks the TX spinlock, acknowledges the interrupt in the core returning the number of frames it transferred from memory into the TX FIFO, associated socket buffers are unmapped and freed. The TX Buffer (e.g. `head` and `free`) is adjusted appropriately. If for some reason the kernel transmit queue was stopped, it is reenabled since the driver is ready to accept more frames. The TX spinlock is unlocked and the interrupt returned as handled.

- `void qpace_compile_int_list(int code, char* s)`
  Compiles a list of event names from `code` that is supplied by the PHY.

- `static irqreturn_t qpace_int_slow(int irq, void *data)`
  Interrupt handler function for SLOW interrupts. Called by kernel upon a SLOW interrupt. Locks the SLOW spinlock, checks if interrupt was issued by Ethernet Core. If not unlocks spinlock and return interrupt as unhandled. If it was, acknowledge interrupt within Ethernet PHY fetching interrupt reason. On Auto-Negotiation Error cycles to next negotiation mode. On Link Status Change (ready or not), determines the speed of the interface negotiated by the PHY and instructed EMAC to provide signals with respect to the mode negotiated. Informs kernel of link availability (or unavailability). Finally unlocks spinlock and returns interrupt as handled.

- `static void qpace_rx_post_ctrl(struct qpace_device *qdev, int ctrl)`
  Posts the physical address of a new control buffer to the Outbound Write Logic, to be used after depletion of current control buffer.

- `static void qpace_rx_post_skb(struct qpace_device *qdev, int skb)`
  Posts physical address of RX socket buffer with index `skb` to the Outbound Write logic.

- `static void qpace_rx_post_all_skb(struct qpace_device *qdev)`
  Posts physical addresses of all RX socket buffers to the Outbound Write Logic. Used during initialization.

- `static void qpace_poll_controller(struct net_device *dev)`
  The poll controller is an interface provided to the kernel when the device needs to operate in a situation without interrupts available. The functions (formally) disables all interrupts registered, calls interrupt interface functions on itself, then (formally) turns interrupts back on.

- `static int qpace_poll(struct napi_struct *napi, int budget)`
  Called by the kernel for the driver to fetch received or waiting frame data and to hand filled socket buffers over to the kernel. Loops on at most `budget` frames received until no more frame available as indicated by preceding interrupt. Waits for frame data in case interrupt was faster than data (should not happen), decrements number of available frames, fetches frame size from control buffer, hands completed socket buffer over to the kernel, memorizes last time-stamp, renews emptied slot in receive ring buffer with new properly allocated socket buffer, advances to next socket buffer, advances to next control buffer if necessary, posts new control buffer if necessary and finally advances to next control buffer slot. If the frame `budget` did not exhaust, informs the kernel on NAPI completion and turns RX interrupts back on.

- `static void qpace_free_rx_buffers(struct qpace_device *qdev)`
  Frees and unmaps all control and RX socket buffers allocated and mapped.

- `static int qpace_alloc_rx_buffers(struct qpace_device *qdev,`
  `int skbs,`
  `int entries)`
  After a few sanity checks, the driver's RX Buffers are allocates and initialized. This includes an allocation of two memory coherent control buffers and registering them for DMA by the core and filling them with a specific default value necessary for smooth operation. Further the specified amount of socket buffers is allocated (by common kernel network socket buffer allocations), mapping them for DMA by the core and checking for the returned physical addresses to match the buffer alignment constraints by the core.

- `static int qpace_do_ioctl(struct net_device *ndev,`
  `struct ifreq *ifr,`
  `int cmd)`
  Called by the kernel to exercise IO control functions on the driver. Unimplemented.

- `static struct net_device_stats *qpace_get_stats(struct net_device *ndev)`
  Returns a pointer to the network device statistics structure to the kernel.

- `static void qpace_tx_timeout(struct net_device *ndev)`
  Called by the kernel on the detection of a TX Watchdog Timeout. Increases the statistics TX fault counter.

- `void qpace_set_generalmac(struct qpace_device *qdev,`
  `const char *mac,`
  `unsigned int i)`
  Sets the `i`th entry of the address filter table within the core's EMAC to the MAC address supplied. The address filter table may for example be used to store multicast addresses.

- `void qpace_set_promisc(struct qpace_device *qdev,`
  `int on)`
  Sets the promiscuous mode enable to `on` with the core's EMAC.

- `static void qpace_set_multicast_list(struct net_device *ndev)`
  Sets the multicast address list within the core's EMAC to the address list supplied by the kernel network device structure. If there are more addresses than the EMAC can handle (i.e., 4) we set set the EMAC into promiscuous mode, if not specified to be set anyway by the network structure.

- `static int qpace_change_mtu(struct net_device* ndev,`
  `int new_mtu)`
  Sets the MTU within the core's EMAC to the byte length supplied and memorizes the value within the kernel network device structure.

- `static int qpace_set_mac_address(struct net_device *ndev,`
  `void *addr)`
  Sets the MAC address within the core's EMAC to the address supplied and memorizes the address within kernel network device structure.

4.2.2 Firmware Device Driver

Although the Linux driver introduced in the previous section will be the code producing most traffic over the lifetime of the hardware it manages, there may also be the necessity for a second driver. This is the case when there is no locally attached long term storage available that is large enough to store a full Linux kernel. In such a case, the Linux kernel is often fetched over Ethernet from some server. Then this full blown Linux driver for the network device takes the stage. In the case of QPACE there is only a small flash memory available to store a firmware image (along with the FPGA bitstream for the NWP), that is read shortly after power up. The firmware uses a different, usually simpler, network driver, sometimes without the possibility to resort to interrupts to get feedback from the device, but uses polling instead. We designed a driver for QPACE Ethernet working within the SLOF framework used for the CBE processor, that works similar to the Linux driver but differs mainly in its interface to the parent code, but we do not describe its details at this point.

Chapter 5

Companion Network Processor Logic

5.1 Management Data In Out Clause 45

The QPACE Torus Network used a physical link standard of 10 Gigabit Ethernet and PMC Sierra QuadPHY 10GX PHYs. These need to be controlled through MDIO. The MDIO Bus is a part of the IEEE Ethernet 802.3 standard thats allows a host controller to operate a number of PHYs attached to it. It provides access to control and status registers of the PHYs. There are essentially two version of the protocol. The older one "Clause 22" standard was developed also with the Ethernet Standard and provided connection to up to 32 PHYs with up to 32 registers each. With the rise to 10 Gigabit Ethernet, the standard was enhanced in IEEE Ethernet 802.3ae, known as "Clause 45", since a much finer configuration may be necessary for those. It provides access to all 2^16 registers of each PHY attached to the bus.

The MDIO interface is implemented by two electric lines, an Management Data Clock (MDC) clock line and an MDIO data line. The clock line is driven by the master device,

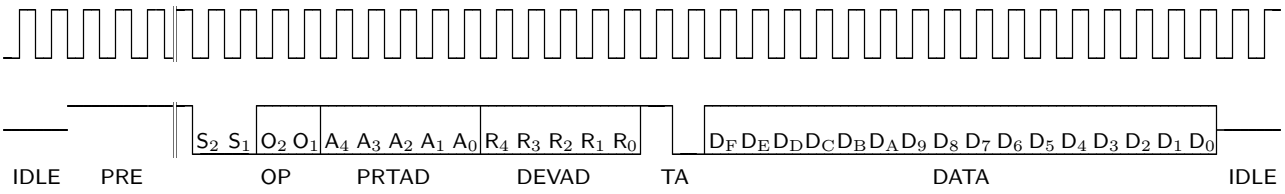


Figure 5.1: Timing diagram of a write cycle. All data is sourced by the master. The upper column symbolizes the signal MDC, the lower one MDIO.

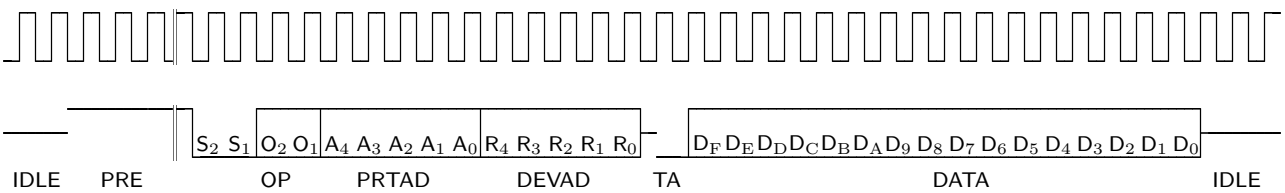


Figure 5.2: Timing diagram of a read cycle. Data up to Turn-Around (TA) is sourced by the master, data is sources by the slave. The upper column symbolizes the signal MDC, the lower one MDIO.

usually at the MAC side of the bus. The data line is bidirectional: the PHY drives it to provide register data at the end of a read operation. There are two types of commands on the bus, a write and a read operation. A write is fully driven by the master, a read driven by the master until the Turn-Around (TA) then driven by the slave. A command is comprised of:

1. 32 preamble bits (PRE), all high
2. 2 start bits (ST), `0b00` for clause 45 (`0b1` for clause 22)
3. 2 opcode bits (OP), `0b00` writes an address, `0b01` writes data, `0b11` reads data, `0b10` incrementally reads data
4. 5 port address bits (PRTAD), (for clause 22 PHY address (PHYAD) instead)
5. 5 device address bits (DEVAD), (for clause 22 register address (REGAD) instead)
6. 2 turnaround bits (TA), `0b10` for a write operation, `0bZ0` for a read operation, where Z just half a cycle.
7. 16 data bits (DATA)

When there is no command the line is electrically in a tristate Z[1]. A full register read or write operation is comprised of two (or more) MDIO45 commands: A write of the register address and a write of the register data.

The bus has a single master, and we implemented such a master for QPACE in a module called "MDIO45". We defined the following logic interface:

Signal Type	Signal Name	Description
Common	`clk`	in `std_logic`
	`reset`	in `std_logic`
Command Interface	`CS`	in `std_logic`
	`RW`	in `std_logic`
	`DataIn`	in `std_logic015`
	`DataOut`	out `std_logic015`
	`Addr`	in `std_logic015`
	`prtad`	in `std_logic_vector(0 to 4)`
	`devad`	in `std_logic_vector(0 to 4)`
	`done`	out `std_logic`
MDIO Interface	`mdc`	out `std_logic`
	`mdio`	inout `std_logic`

The interface was designed to be easily attached to a DCR slave. We show the corresponding block and state diagrams in figure 5.3. A MDIO command may be issued by strobing `CS` together with a specification of the command type (read `RW=1`, write `RW=0`). The port and device address is taken one cycle after `CS` high, the register address `Addr` is taken in the same cycle `CS` is high, while the data to be written `DataIn` needs to stay valid until a completion strobe in `done`. `DataIn` is ignored for reads, instead the data read `DataOut` will be valid with the `done` strobe. A new command may be issued is the cycle after the `done` strobe.

[1]The state 'Z' of an electric line is short for 'Hi-Z', or high impedance. Such a state does not force the electric line to be high or low: It sends nothing.

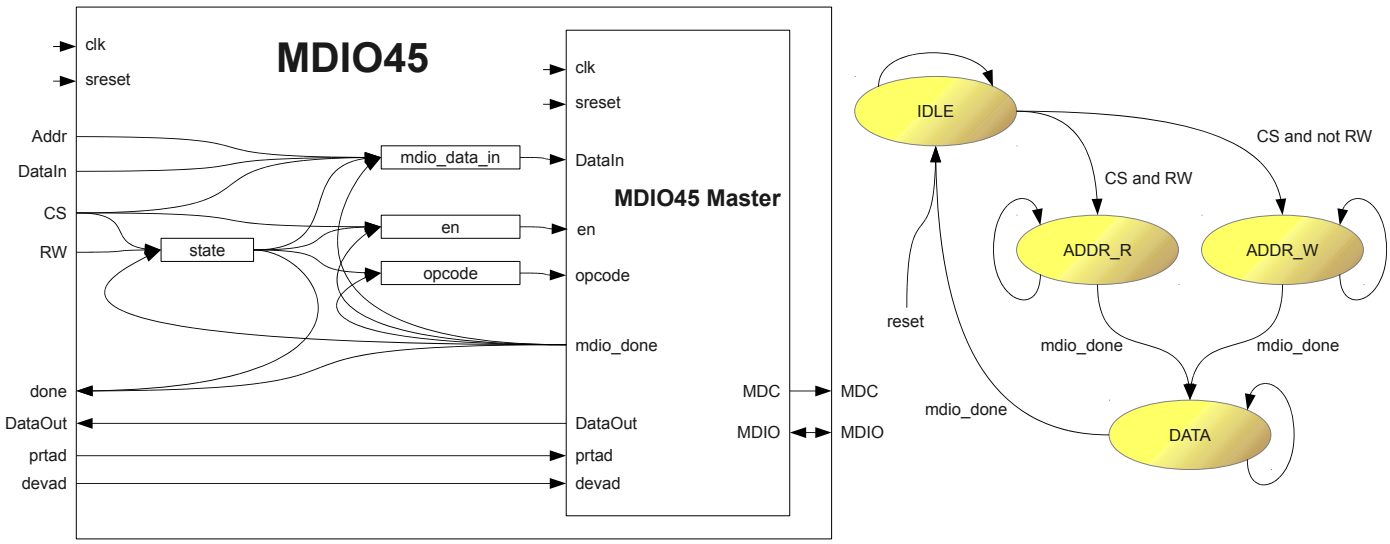


Figure 5.3: A block diagram of the MDIO45 module and a diagram of its FSM.

During a reset the FSM is in its IDLE state and `en` and `opcode` set to 0. During operation, a strobe in `CS` switches the state to either ADDR_R (`RW=1`) or ADDR_W (`RW=0`), writes the address into `mdio_data_in` and strobes `en`. This instructs the MDIO Master module to run a read or write sequence as depicted in figures 5.1 or 5.2. In this case we have an address write (`OP=0b00`) and receive a strobe in `mdio_done` when complete. This event advances the FSM to the DATA state, sets the opcode to write data or read data, and again strobes `en`. In case of a data write `DataIn` will also be provided to the MDIO master through `mdio_data_in`. The MDIO Master runs the sequence as demanded and finishes when `mdio_done` strobes. On a read `DataOut` provides the data read. The machine reverts to the IDLE state.

The MDIO45 Master, depicted in figure 5.4, needs to drive the MDIO clock and data line at a much lower frequency than it is being clocked itself. Therefore it utilizes a clock counter (`clk_counter`) whose width determines the length of a MDIO cycle. Every time the counter wraps around we have an MDIO cycle. We therefore use the most significant bit of the counter as the MDIO clock `mdc` and decode rising `mdc_rising` and falling edges `mdc_falling` of that MDIO clock from the counter. As defined by the MDIO standard all output signals of the Master need to be put on the line with rising edges of the clock, while all input signals need to be sampled with falling edges of the clock (like `mdin`).

After reset, the FSM enters the IDLE state. In this state the byte counter `counter` is reset to 0 and the MDIO Buffer tristate signal `mdio_t` is asserted such that the buffer is in input mode. Once the command enable signal `en` is strobed to the master, the FSM switches to the WAITING state, at the same time registering the signals `prtad`, `devad`, `opcode` and `DataIn` into a 32 bit wide shift register `data` in an order that matches the data to be written out sequentially. The first bit `data(31)` is set to 1 in order to prepare sending the preamble and data through the MDIO tristate buffer to any slaves attached to the bus. Once there is a falling edge in the MDIO clock, the WAITING state is left for the PREAMBLE state and the tristate enable for the MDIO Buffer is deasserted, switching the buffer to output mode. This starts sending data over the MDIO data line (`MDIO`). During this state the byte counter (`counter`) reset is removed and the counter is thus increased for every falling clock edge in `MDC`. Once the counter reaches 31 the preamble has been successfully transmitted and the FSM switches into the SHIFTING state along with setting the first bit `data(31)` to 0, preparing for the MDIO45 start code of `ST=0b00`. The counter wraps around to 0 and continues to count

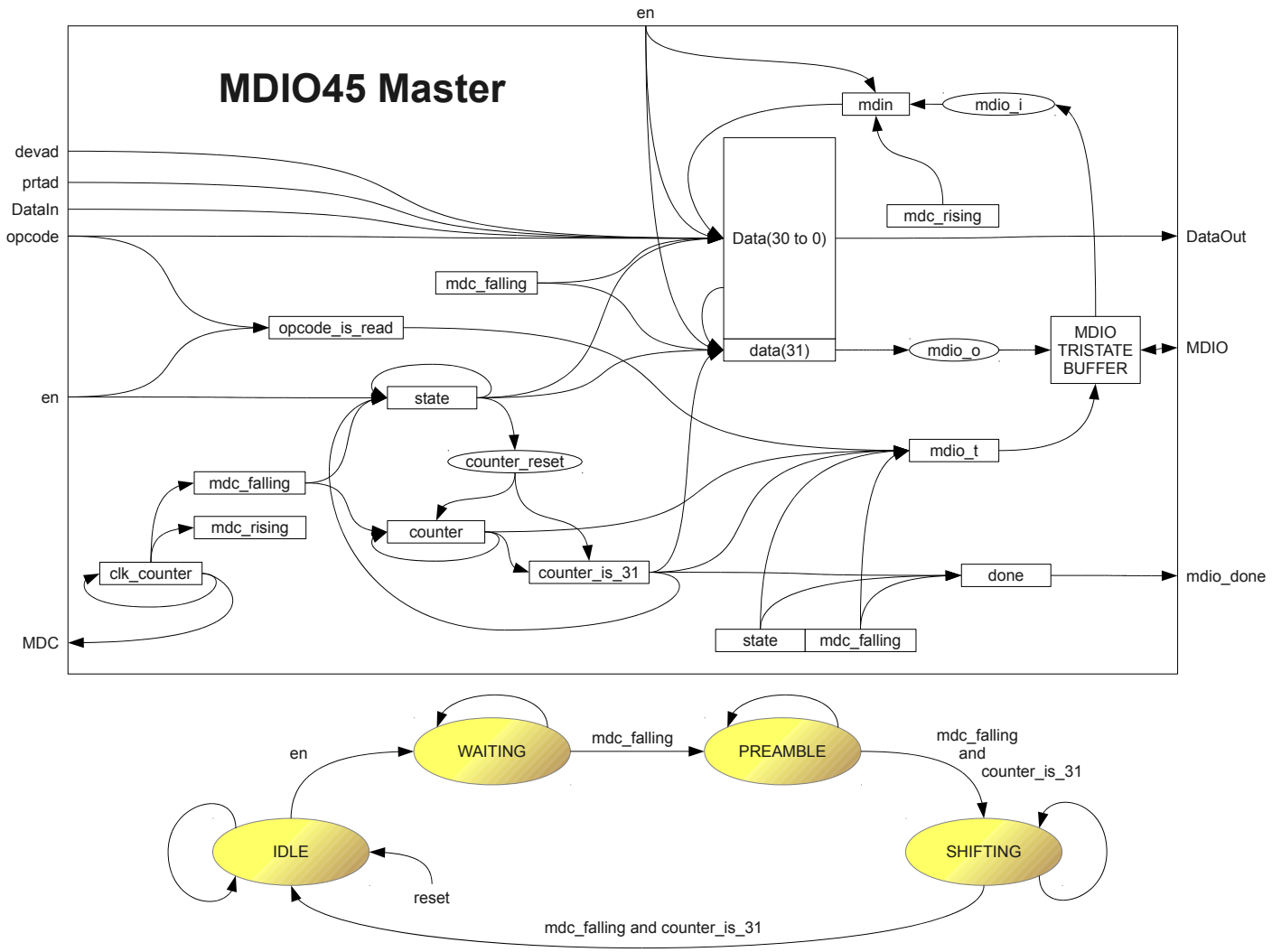


Figure 5.4: A block diagram of the MDIO45 Master module and a diagram of its FSM.

each falling `MDC` edge. During the SHIFTING state the `data` shift register is shifted by one byte each falling clock, filling in bits from `mdin` at the end `data(0)` of the register. The new front is provided to the MDIO Buffer and sent to any slaves attached. Once the byte counter reaches 13 the MDIO Tristate Buffer needs to be switched into the input mode when a read command is in progress. This is decoded from the `opcode`. Once the byte counter reaches 31 all data has been shifted out or in. The MDIO command is complete and eventually read data resides in `data(15 to 0)` and is supplied in `DataOut`. At this point a command completion strobe is generated in `done` and provided to adjacent logic. The FSM switches back to the IDLE state.

5.2 Clock Domain Transition DCR Slave

The DCR Bus is a bus designed by IBM that is widely used in PowerPC architectures to control and monitor any kind of device. A transaction on a DCR Bus is done by a simple 4-phase handshake protocol, transferring one word at a time. A complete architecture specification can be found in [24]. There are many functional blocks in the QPACE Network Processor

that are controlled through the DCR bus. Most slaves are driven by the same clock as the DCR master. There are, for example the UARTs, the GS, the SPI Master and two of the three Ethernet Slaves. Other important logic is however driven by a clock that is synchronous to their physical clocks, for example all Torus slaves and one of the three Ethernet Slaves. Since the original DCR specification defines synchronous slaves only, we designed a variant of a DCR slave that is compatible with the default synchronous one but handles all clock domain transitions of signals internally and using proper timing constraints. We show a block diagram of this CDT-DCR is figure 5.5. When the bus is idle, the DCR Master can issue a

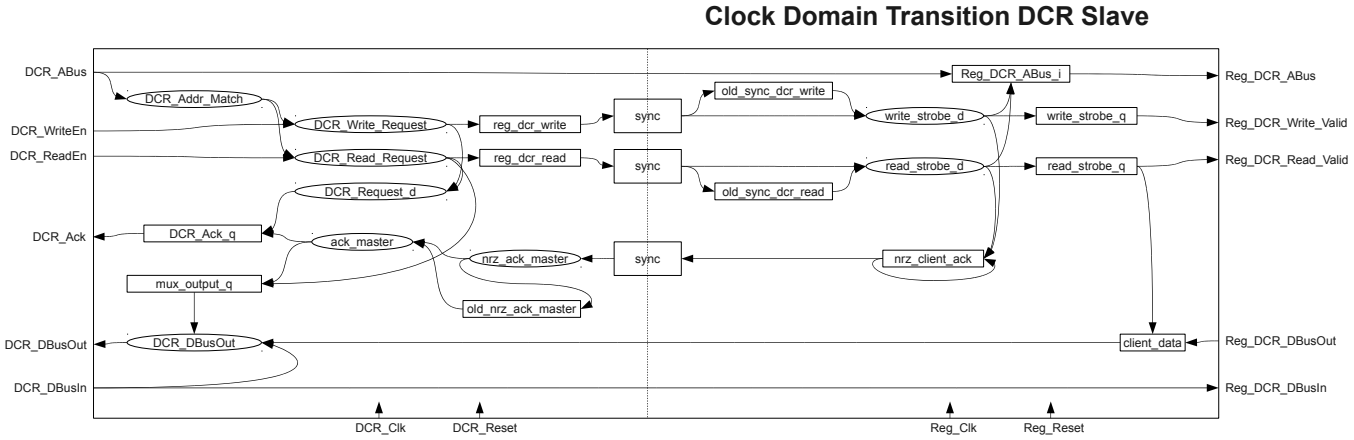


Figure 5.5: A block diagram of the Clock Domain Transition DCR slave module.

new instruction by asserting `DCR_WriteEn` or `DCR_ReadEn`, while supplying the register address in `DCR_ABus` and, for a write operation, the data to written in `DCR_DBusIn`. Depending on the address configured, our DCR slave decoded whether it is addressed in `DCR_Addr_Match` and asserts `DCR_Write_Request` or `DCR_Read_Request` is such a case. In order to convey this information to the slave clock (or "Register" clock), we first buffer the information in Flip-Flops `reg_dcr_write` or `reg_dcr_read` and synchronize the signal to the register clock using a synchronizer module. Our synchronizer modules are made up from two serial registers clocked by the destination clock, leaving enough time for potentially meta-stable Flip-Flops to decay to some definite value. We continue with generating strobes for a request in `write_strobe_d` or `read_strobe_d`. The client side acknowledges any request by toggling `nrz_client_ack`. At the same cycle `write_strobe_q` or `write_strobe_q` pulse the command to the client and provide the register address in `Reg_DCR_ABus` and the register data in `Reg_DCR_ABus`. Up to this point one Master cycle and two slave cycles have passed. For a read strobe the client has to deliver the data in the next cycle in `Reg_DCR_DBusOut`, where we sample the data into `client_data`. The client ack signal `nrz_client_ack` is synced to the master clock domain and converted to a strobe in `ack_master`. If there was a request (`DCR_Request_d`), `DCR_Ack` is strobed for a cycle. A the same time the DCR Data Output (`DCR_DBusOut`) mux selector is set to source the client data (`client_data`) in case a read operation is in progress. If not, the `DCR_DBusIn` data is fed through.

The slave's output signals `DCR_Ack` and `DCR_DBusOut` are not fully in-line with the DCR Specification in [24], since we hold the ack and data signal for one cycle only, instead of as long a the master's read or write command is active plus one cycle. In practice we have found that our implementation works well, resulting in the conclusion that the DCR master employed in the NWP is content with a single cycle ack and samples incoming data in the same cycle.

Finally we need to make sure that signals crossing the clock domain are routed and timed properly within the FPGA such that we obtain expected values on clock edges. The

clock domain boundary is depicted as a vertical dotted line in figure 5.5. First, we have two signals `reg_dcr_read` and `reg_dcr_write` that cross the boundary. Since we put a proper synchronizer we do not need to take care of these signals and therefore lift all timing constraints for these signals to the client clock domain. The address bus `DCR_ABus` however need to be stable at the time it is sampled in the client clock domain. The earliest point the signal could be potentially sampled is one master clock and two slave clock cycles after `DCR_WriteEn` or `DCR_ReadEn` becomes asserted. We therefore constrain the delay of any signal from the master clock through `DCR_ABus` to the client clock domain to be less than

$$t_{\text{delay}}^{\texttt{DCR\_ABus}} < 1 \cdot T_{\text{master}} + 2 \cdot T_{\text{client}}.$$

For the data bus `DCR_DBusIn`, the setup is similar. The signal is potentially being registered one cycle later than the address bus `DCR_ABus`. For this signal we therefore need to specify

$$t_{\text{delay}}^{\texttt{DCR\_DBusIn}} < 1 \cdot T_{\text{master}} + 3 \cdot T_{\text{client}}.$$

On the return side is the ack signal `nrz_client_ack`. We constrain the timing of this signal to be ignored for the same reasons we have above with for example `reg_dcr_read`. The return data signal `client_data` is valid one cycle of the toggle in `nrz_client_ack`, whose ack strobe `ack_master` will be decoded three master cycles later at the earliest. In the fourth cycle the multiplexer is set and data may be sampled by the DCR Master. Altogether the data signal may have a delay of three master cycles maximum

$$t_{\text{delay}}^{\texttt{client\_data}} < 3 \cdot T_{\text{master}}.$$

5.3 IWC Extension Module

The IWC Extension Module (IWC-EM) is designed to be attached to the IWC. It provides registers holding status information of the IWC, exception signals as well as an optional data monitor that can be used for debugging and performance evaluations of the inbound write data traffic. The Data Monitor records any request of the IWC together with request address, request size and the almost full signals from the IWC clients. The registers and the data recorded can be read through DCR within the address space of the IWC-EM. The IWC-EM is attached to the DCR Bus using a CDT-DCR Slave. There are six DCR registers accessible:

Register Name	Address	Register Description
`IWC_FIR`	`0x2100`	Default Value: `0x00000000` This register holds the fault isolation flags and the timeout counter value. Bit 0-16: Timeout Counter Value Bit 26: Command Array Overflow Bit 27: Data Array Overflow Bit 28: Unexpected Request Bit 29: Unexpected Address Bit 30: Unexpected Size Bit 31: Request Timeout
`IWC_FIR_EN`	`0x2101`	Default Value: `0x0000002F` This register holds the fault isolation enables. Bit mapping matches `IWC_FIR`. All enabled by default.
`IWC_ATTR_1`	`0x2102`	Default Value: `0x00000000` In case of an exception, holds data related to it. Bit 0-31: Request Address Bits 10 to 41

IWC_ATTR_2	0x2103	Default Value: 0x00000000 In case of an exception, holds data related to it. Bit 22-31: Request Address Bits 0 to 9 Bit 12-21: Request Size
IWC_MON_DATA	0x2104	Default Value: 0x00000000 Data Register of the Monitor instantiated. See section 5.4.
IWC_MON_CONFIG	0x2105	Default Value: 0x00000000 Configuration register of the Monitor instantiated. See section 5.4.

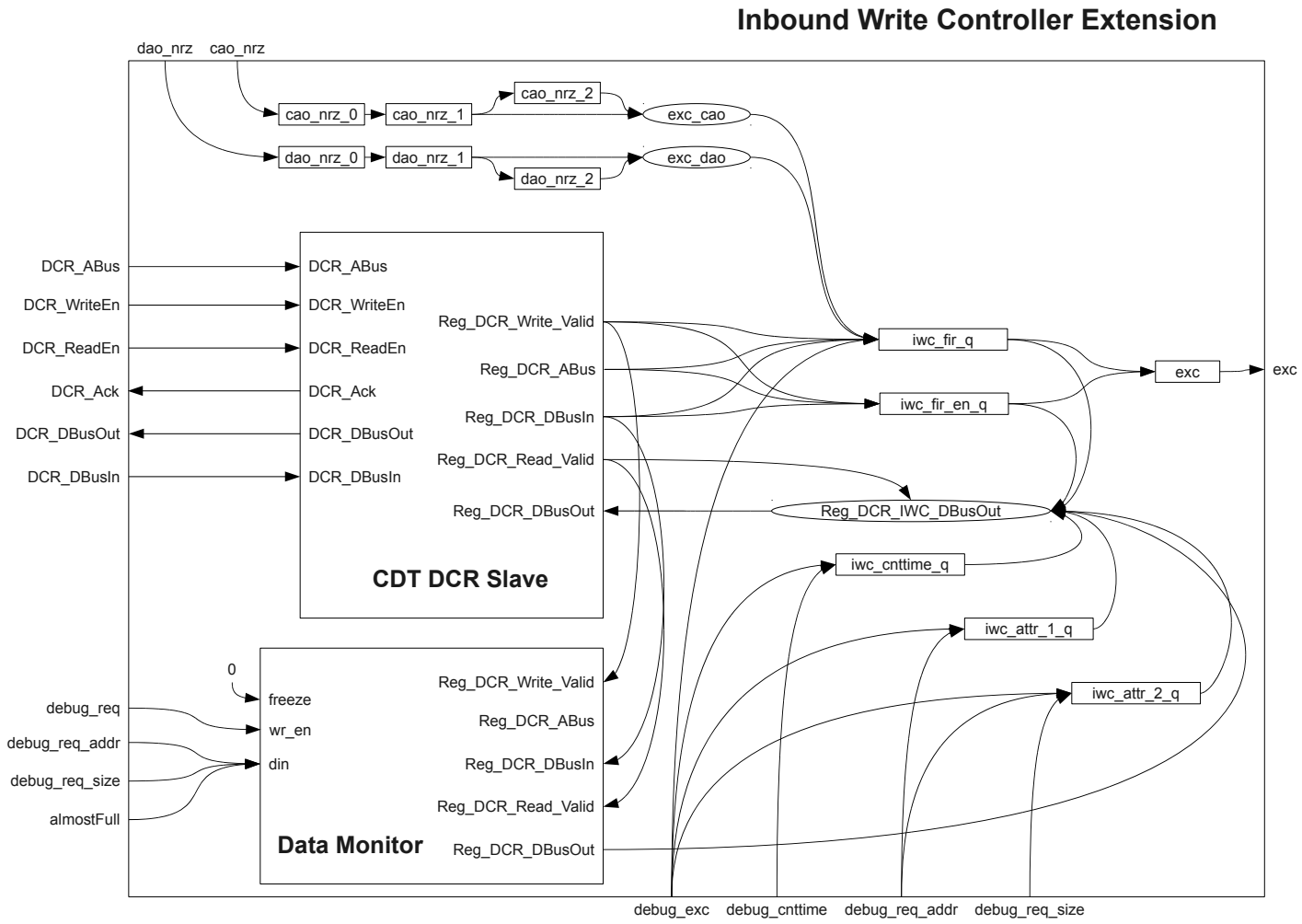


Figure 5.6: A block diagram of the IWC Extension Module.

5.4 Data Monitor

The Data Monitor module is a small but efficient logic-monitoring and logic-debugging tool. It can be used to record any stream of data into a FIFO of (hardware-configurable) width and depth. Data can be read at a later stage using DCR. While the system is in operation any data recorded by the module can be reset using a simple register access using DCR. Once out of reset, the software can release the monitor freeze `freeze_dcr` at any given time in order to start recording data. Data `din` is now written to the buffer each requested `wr_en` cycle together with the number of cycles passed `cycles_passed` since the last write. Additionally there is a (convenient) hardware freeze input that can be used to exclude data from being recorded while asserted. This can be used to stop writing data to the monitor once some hardware exception occurs. This may allow for a reconstruction of events that led to the exception and may lead to more efficient debugging and bug fixing. Once the monitor is frozen, data may be read through DCR. The register layout is described in table 5.3. There is one configuration register that holds a number of readable and writable flags to operator the monitor. Depending on the width of the monitor instantiated there are a multiple data registers that allow access to the data recorded.

Register Name	Address	Register Description
`MON_CONFIG`	configurable	Default Value: `0x00000001` Mask Value: `0x00000003` Configuration Register Bit 28: Hardware Freeze (R) Bit 29: FIFO empty (R) Bit 30: Software Freeze (RW) Bit 31: Software Reset (RW)
`MON_DATA(0 to N-1)`	configurable	Data Registers For a Data Monitor width of N, there are N data registers. Register i reads bits $i \cdot 32$ to $i \cdot 32 + 31$ of current the FIFO output word. Registers should be read sequentially before advancing to the next FIFO word.

The Data Monitor following configuration options:

- `WIDTH` defines the number of parallel 32 bit FIFOs to be instantiated.
- `CYCLES_COUNTER_WIDTH` defines the number of bits at the start of each FIFO word that are dedicated to store the number of cycles passed since the last write. A value of zero is therefore excluded. The counter is protected from wrapping around.
- The last two parameters define the number of data bits that can be recorded each cycle to be $32 \cdot$ `WIDTH` $-$ `CYCLES_COUNTER_WIDTH`.
- `WRITE_PIPE` defines the length of the data register pipe before data is actually written to the FIFO. This is a functionally irrelevant parameter but may have a significant impact on the logic distribution with the FPGA since, the monitor and its FIFO may be placed at a remote place with respect to the logic to be monitored. It may be varied to improve and routing timing within the FPGA.
- `DEVICE_SELECT_WIDTH` together with `DEVICE_ADDR` define the base address (and length)

of the DCR register space for the Data Monitor. The base address will be

$$\texttt{DEVICE\_ADDR} \cdot 2^{\texttt{DEVICE\_SELECT\_WIDTH}}.$$

- `DEVICE_CONFIG_ADDR` defines the offset of the address of the configuration register from the monitor base address.
- `DEVICE_DATA_ADDR` defines the offset of the address of data register 0 from the monitor base address. Any further data registers will be placed at `DEVICE_DATA_ADDR`+1, `DEVICE_DATA_ADDR`+2, etc . . . .

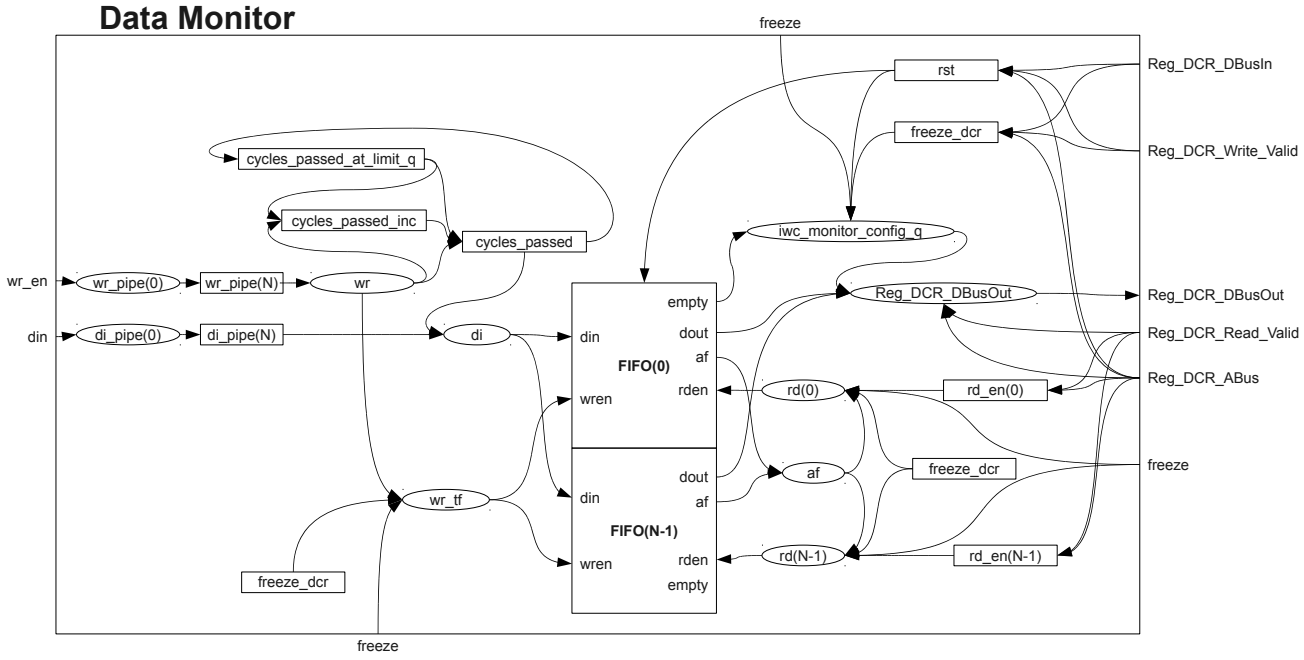


Figure 5.7: A block diagram of the Data Monitor module.

We now describe the logic of the Data Monitor. While in (hardware) reset, the registers `freeze_dcr` and `rd_en` are set to 0, `rst` is set to 1. During operation, data input `din` and write-enables `wr_en` are pipelined with the requested number of steps. If the module is neither frozen through software (`freeze_dcr`) or hardware (`freeze`), the FIFO write-enable signal `wr_rf` is asserted in order to write the FIFO data word `di` into the FIFO. The FIFO data word `di` is a concatenation of the number of cycles passed `cycles_passed` since the last (attempted) write `wr` and the actual payload data from the last stage of the data pipeline. Data is spread to the parallel FIFOs properly. If the FIFO is still in reset (`rst`) any writes will be ignored by the XILINX FIFOs. On the read side of the FIFO we employ a read-enable register `rd_en` for each of the sub-FIFOs. Once any of the sub-FIFOs becomes almost full `af` a read is issued to all of the sub-FIFOs. This drops the oldest data and creates space for most recent data. In effect, the Data Monitor provides the latest part of the data stream recorded if not frozen by software or hardware. All configuration and status bits are gathered in `monitor_config_q` and can be read through the DCR Slave client interface employed. In any freeze mode, incoming read requests to the sub-FIFOs are answered with current FWFT data provided and a read-enable is issued to advance the relevant sub-FIFO to its next word stored.

We note that there is no full DCR Slave instantiated in this module. Instead out DCR Slave client interface is provided such that the Data Monitor is easily attachable to any of the operationally relevant DCR slaves already instantiated.

On the software side, a small Linux tool template of a few hundred lines of C code was provided to configure and operate any Data Monitor instantiated. A few Data Monitors were instantiated in order to monitor and debug bus activity, frame flow, etc... There are monitors for the OWC, the IWC and the Ethernet Outbound Read module. Once the QPACE system was at the edge of operability, the Data Monitors have proven to be extremely helpful in understanding complex behavior of the system.

5.5 Control Box

The Control Box is a simple module that provides an instance of GS and a configurable amount of reset registers. It instantiates a CDT-DCR Slave (see section 5.2) to provide access to the registers. For the node-to-node communication lines used by the GS we add input and output

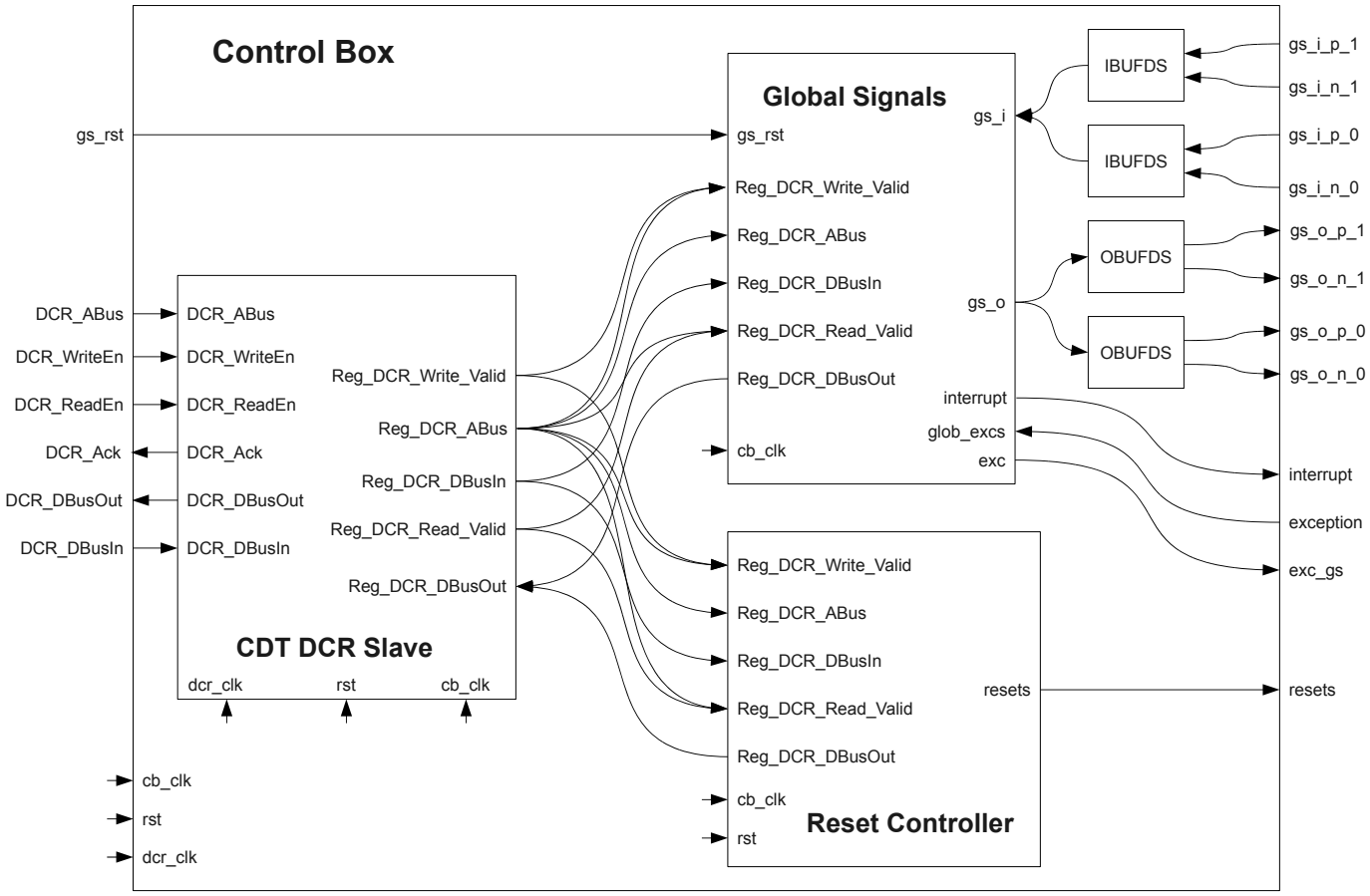


Figure 5.8: A block diagram of the Control Box module.

buffers for differential signaling. We show a block diagram of the logic in figure 5.8. The CDT-DCR Slave interfaces to a DCR Master that is clocked by `dcr_clk` and provides access to registers clocked by `cb_clk`. Depending on the register address (`Reg_DCR_ABus`) write- or read-enables are forwarded to the Reset Controller or the GS.

5.6 Reset Controller

The Reset Controller is a small but important entity within the NWP that provided a number of DCR registers that control the reset states of most functional blocks. The logic is rather simple, so we only list its registers in table 5.4.

Register Name	Address	Register Description
RST_ETH	0x0090	Reset of Ethernet Logic
RST_GS	0x0091	Reset of GS Logic
RST_OWC	0x0092	Reset of OWC Logic
RST_DLYCTRL	0x0093	Reset of I/O Delay Controllers
RST_IWC	0x0094	Reset of IWC Logic
RST_TORUS	0x0095	Reset of Torus Logic
EXC_SOFT	0x0096	Assert to trigger a Software Exception

5.7 Global Signals

We introduced the GS in section 3.7. In this section we present its logic. We defined the following logic interface:

Signal Type	Signal Name	Description
Common	`clk`	in `std_logic`
	`rst`	in `std_logic`
Register Interface	`reg_read`	in `std_logic`
	`reg_write`	in `std_logic`
	`reg_abus`	in `std_logic_vector(0 to 3)`
	`reg_dbusin`	in `std_logic_vector(0 to 31)`
	`reg_dbusout`	out `std_logic_vector(0 to 31)`
Exceptions & Interrupts	`glob_excs`	in `std_logic_vector(0 to 31)`
	`exc`	out `std_logic`
	`interrupt`	out `std_logic`
GS Interface	`gs_i`	in `std_logic_vector(0 to 1)`
	`gs_o`	out `std_logic_vector(0 to 1)`

The GS Module features our standard register interface, such that it can be easily attached to our CDT-DCR Slave, see section 5.2. We list the registers available:

Register Name	Address	Register Description
`GS_STATUS`	`0x0080`	Status Register Reset Value: `0x00000000` Read Only Bit 0: Global Exception Bit 1: Write-Write Exception Bit 2: RX-Idle Exception Bit 3: Timeout Exception Bit 4: No-Ack Exception Bit 7: Interrupt Bit 24: Receive Kill Bit 25: Receive Valid Bit 26: Receive Value Bit 28: Transmit Kill Bit 29: Transmit Valid Bit 30: Transmit Value
`GS_STATUS_RXACK`	`0x0081`	Status Register with RX Acknowledge Same behavior as `GS_STATUS`, but bits 7, 24 and 25 are COR, i.e., received values are acknowledged and a possible interrupt deactivated.
`GS_CMD`	`0x0082`	Command Register Write Only
`GS_CONFIG`	`0x0083`	Configuration Register Reset Value: `0xF9101000`, i.e., all enabled and $N =$ `0x1000`. Bit 0: Global Exception Enable Bit 1: Write-Write Exception Enable Bit 2: RX-Idle Exception Enable

		Bit 3: Timeout Exception Enable Bit 4: No-Ack Exception Enable Bit 7: Interrupt Enable Bit 11: Closed Mode Bit 18-31: Transmit Timeout Limit N, timeout occurs $(2^{24}-1)\,N$ cycles after transmit start if nothing received.
`GS_COUNTERS`	`0x0084`	Counters Register Bit 2-15: Timeout Counter, increased every $(2^{24}-1)$ cycles after transmit start Bit 16-23: Interrupt Counter, increased for every interrupt request, clears on any write Bit 24-31: Latency Counter, increased every cycle with interrupt active, resets on new interrupt request
`GS_GLOB_EXCS`	`0x0085`	Global Exceptions Register Holds incoming Global Exceptions. Up to 32 available. For QPACE we defined: Bit 0: System Bus Exception Bit 1: IWC Exception Bit 2: OWC Exception Bit 3: Ethernet Exception Bit 4: Reserved Bit 5: Reserved Bit 6: Torus TX 0 Exception Bit 7: Torus TX 1 Exception Bit 8: Torus TX 2 Exception Bit 9: Torus TX 3 Exception Bit 10: Torus TX 4 Exception Bit 11: Torus TX 5 Exception Bit 12: Reserved Bit 13: Reserved Bit 14: Torus RX 0 Exception Bit 15: Torus RX 1 Exception Bit 16: Torus RX 2 Exception Bit 17: Torus RX 3 Exception Bit 18: Torus RX 4 Exception Bit 19: Torus RX 5 Exception Bit 20: GS Exception Bit 21: Software Exception Bit 22-31: Reserved
`GS_VERSION`	`0x0086`	Version Register A value of `0x12345678` maps to version v123.4.5678. At the time of writing the latest version is v4.0.5.

There are 4 different signals on the GS Tree,

Signal Name	gs(0)	gs(1)	Kill	Valid	Value
NOP	0	0	0	0	X
KILL	1	1	1	X	X
TRUE	0	1	0	1	1
FALSE	1	0	0	1	0

where a NOP cannot be sent explicitly by the software, but is sent by the logic while nothing is going on. There is also no explicit way to send a KILL from within the GS register space. Instead the driver should raise a Software Exception by a write to the `EXC_SOFT` register, located outside the GS module.

We now describe its logic depicted in figure 5.9. During reset (`rst` asserted), most registers are set to 0: `rx_kill_q`, `rx_valid_q`, `rx_value_q`, `tx_kill_q`, `tx_valid_q`, `tx_value_q`, `exc_wrwr_q`, `exc_rxidle_q`, `exc_to_q`, `exc_noack_q`, `rx_reset_q`, `gs_out`, `interrupt_req`, `interrupt_q`, `int_counter`, `exc`. All exception enables `exc_*_en` are set to 1, as well as `closed`. The transmit timeout limit `to_limit` and counter `to_counter` is also set to its default value.

During operation, the register interface can be used at any time to read any of the registers `gs_config`, `gs_status`, `gs_counter`, `gs_version`, `gs_glob_excs` by strobing `reg_read` and providing the proper register address in `reg_abus`. Data is muxed properly into `reg_dbusout` to be accessed by for example a CDT-DCR Slave. Further, data can be written to all bits of the `gs_config` register by strobing `reg_write` and providing the data in `reg_dbusin` and the destination address in `reg_abus`. Further we prepare strobes in `reg_gs_status_clear` when register `GS_STATUS_RXACK` is read, and in `write_gs_cmd` when register `GS_CMD` is written. Their use is explained shortly.

We start with global exceptions. The global exception signals `glob_excs` are first synchronized to the local clock in `gs_glob_excs` (a register that can be read) and then logically `OR`-ed together into `glob_exc` using an intermediate pipeline step `glob_exc_pipe`. If global exceptions are enabled `glob_exc_pending` is asserted and a strobe in `glob_exc_strobe` is generated. Any such strobe asserts `tx_kill_q`, one of the 3 registers determining the GS Transmit Signal. In case of a transmit command write `write_gs_cmd`, the register `rx_valid_q` is asserted and the supplied value from `reg_dbusin` written to `tx_value_q`. This can only be overridden by a reset of the transmit signals in case `tx_reset_q` is asserted, i.e., when a

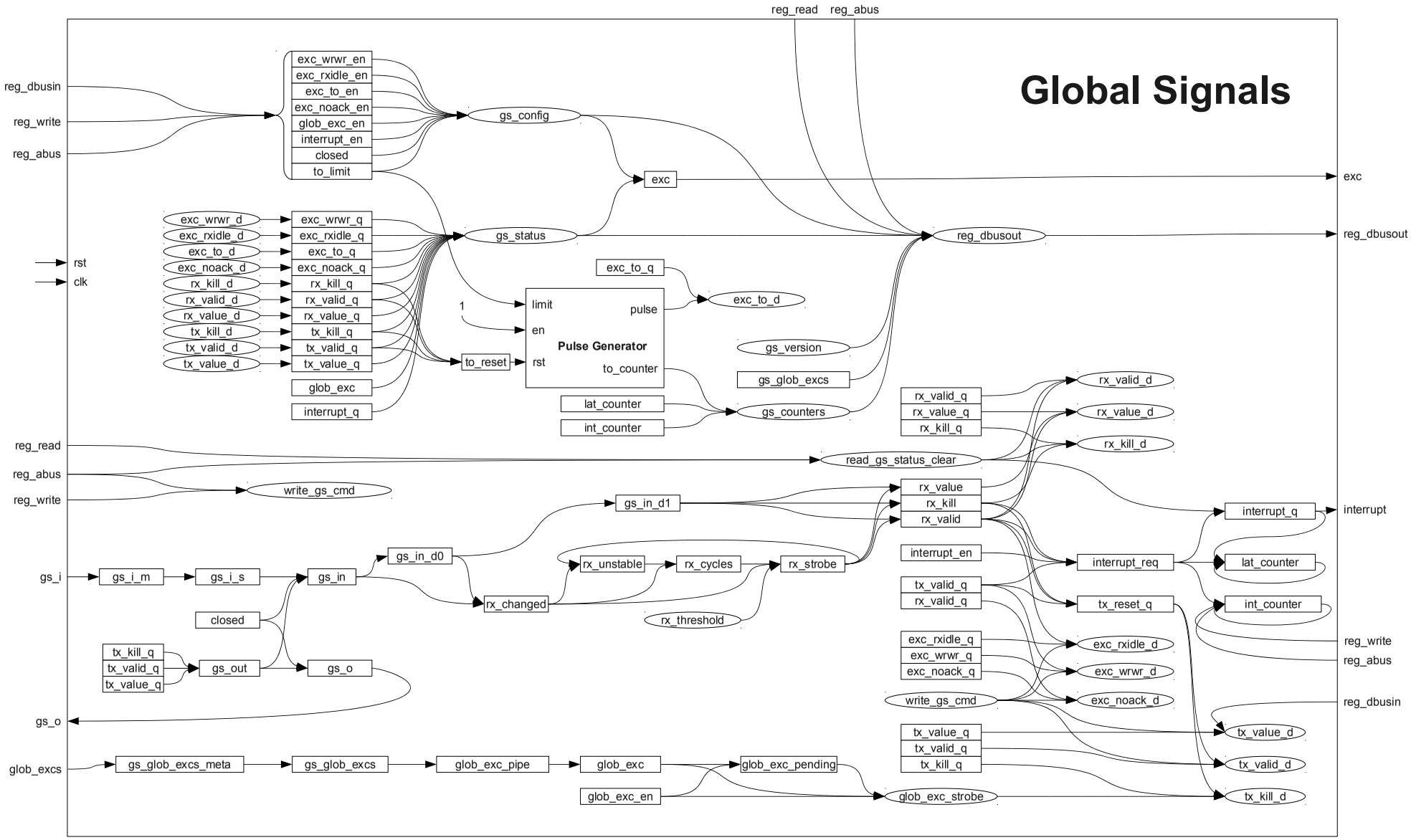


Figure 5.9: A block diagram of the GS Module.

GS communication is completed, see later.

As stated above, `tx_kill_q`, `tx_valid_q` and `tx_value_q` determine the signal sent. The proper two bit signal is encoded in `gs_out` and output to any recipient in `gs_o` if the module is not setup to be `closed`. If closed, a NOP is sent. An incoming signal from other nodes in `gs_i` is first synchronized to the GS clock using a chain of two registers `gs_i_m` and `gs_i_s`. If the module is not `closed`, incoming data is accepted into `gs_in`. If not, the data sent (`gs_out`) is taken into `gs_in`. We delay the effectively received data by two cycles into `gs_in_d0` and `gs_in_d1`. From the first one we decode any change in received signal into a strobe in `rx_changed`. If the signal changed we declare the signal to be unstable (`rx_unstable`) and reset the receive stable counter (`rx_cycles`) to 0. If it did not change and the signal is unstable we increment the receive stable counter. Once the signal did not change for a specific number of cycles (`rx_threshold`) we generate a signal stable strobe `rx_strobe`. This strobe is used to decode the received data into the strobes `rx_value`, `rx_valid` and `rx_kill` containing received values. These strobes trigger a number of things:

1. Any received values are written into the receive data registers `rx_value_q`, `rx_valid_q` and `rx_kill_q`, that can be read using DCR and also cleared (`reg_gs_status_clear`) by a read to the proper address.

2. When a GS communication is complete, i.e., when the logic is currently in transmit mode (`tx_valid_q`) and either a valid signal was received (`rx_valid`), or a kill signal was received (`rx_kill`), the transmit reset is asserted (`tx_reset_q`).

3. When a GS communication is complete and interrupts are enabled (`interrupt_en`), an interrupt request strobe (`interrupt_req`) is generated.

4. The Receive on Idle exception (`exc_rxidle`) is raised if there was data (`tx_valid`) received, but nothing send (`tx_valid_q`).

Three other exceptions can occur:

1. In case data has been received and is ready for any software to be read `rx_valid_q` and acknowledged by reading `GS_STATUS_RXACK`, but instead a new transmit command is issued (`write_gs_cmd`), data may be lost and a No-Acknowledge exception (`exc_noack`) is raised.

2. Similarly, when a transmit is in progress `tx_valid_q` data may not be changed by a new command issued (`write_gs_cmd`) and a Write-Write exception (`exc_wrwr`) is raised.

3. When there is data sent (`tx_valid_q` or `tx_kill_q`) we lower the reset of the timeout counter. Once data is received (`rx_valid_q` or `rx_kill_q`) we set the counter back into reset. If meanwhile, the counter reaches the limit configured by `to_limit`, we raise the Transmit Timeout exception (`exc_to`).

For every interrupt request (`interrupt_req`) generated by the logic, we assert the interrupt output (`interrupt_q`) and increment the interrupt counter (`int_counter`), that can be read together with the timeout counter and the latency counter using DCR. It can be reset by any write. The latency counter (`lat_counter`) however resets to 0 on any interrupt request and increments each cycle an interrupt (`interrupt_q`) is sent. This way it is possible to measure the number of cycles it took for the interrupt to propagate through the system bus into the CPU, for the software to acknowledge the interrupt, and for the DCR read request (`read_gs_status_clear`) to acknowledge the interrupt by lowering `interrupt_q`.

Last but not least, at any cycle any local exception enable (from `gs_config`) is compared with any local exception occurring (from `gs_status`) and the Global Signal Exception output `exc` set appropriately.

The GS logic makes use of a small submodule, the Pulse Generator. This is a simple logic that creates a pulse each configured number of cycles. We show a block diagram in figure 5.10. During reset, all registers are set to 0, only `lfsr` is set to 1. For each cycle enabled (`en`),

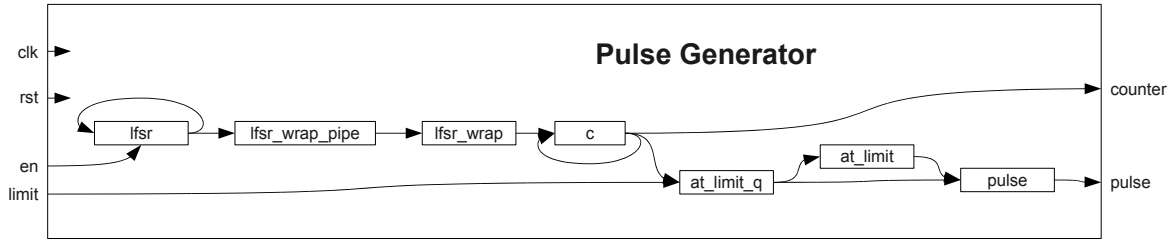


Figure 5.10: A block diagram of the Pulse Generator module.

the 32 bit Fibonacci Linear Feedback Shift Register (LFSR) `lfsr`, configured for a maximum length sequence is advanced to its next state. Each of the `lfsr_wrap_pipe` signals 'AND' a few consecutive bits each together, for the final 'AND' to produce a pulse in `lfsr_wrap` when all bits are 1 and the LFSR is about to cycle. For every LFSR cycle, the binary counter `c` is increased to be easily compared to the binary limit value in `limit`. If they coincide, `limit` is asserted. It they coincide and limit is deasserted, a pulse in `pulse` is generated.

Part II

Axial Charges of Excited Nucleons

Chapter 6

Introduction to Quantum Chromodynamics

Quantum Chromodynamics

In the 1950s the invention of bubble chambers and spark chambers allowed particle physicists to discover an ever-growing number of particles called hadrons. It seemed that such a large number of particles could not all be fundamental. Over the years it became possible to classify them by charge and isospin and later in 1953, according to strangeness. Further insight came with the grouping of hadrons with similar properties and masses using the eightfold way, invented in 1961 by Gell-Mann and Yuval Ne'eman. Two years later, in 1963, Gell-Mann and George Zweig went on to propose that the structure of the grouping could be explained by the existence of three flavors of "smaller" particles inside the hadrons: the quarks. With the discovery of the top quark in 1995 at Fermilab, it is now clear that there are at least 6 quark flavors, grouped in 3 generations

$$\left(\begin{array}{c|c|c} \text{up} & \text{charm} & \text{top} \\ \text{down} & \text{strange} & \text{bottom} \end{array}\right)$$

Back then, one hadron, the Δ^{++} remained mysterious as it is built up from three up quarks with parallel spins. However, the Pauli exclusion principle forbids such a state[1]. A little later, Moo-Young Han with Yoichiro Nambu and Oscar W. Greenberg independently resolved the problem by proposing that quarks possess an additional SU(3) gauge degree of freedom, later named *color charge*. Han and Nambu noted that quarks would interact via an octet of vector gauge bosons: the gluons. This theory of quarks and their interaction by gluons was named *Quantum Chromodynamics* (QCD) and is a special case of a non-abelian gauge theory originally invented by Yang and Mills.

Asymptotic Freedom

One of the most crucial properties of this theory of color interactions is that gluons do carry color charges and therefore interact with each other. This self-interaction makes QCD a highly nonlinear and thus extremely complex theory. Fortunately it turned out that it has a very special property called *asymptotic freedom*. For large momentum transfers, the coupling strength of QCD becomes small, allowing the use of pen and paper to work out perturbation theory predictions for such processes. This property has been discovered by David Gross, David Politzer and Frank Wilczek, who received the Nobel Price in 2004. Particle physics

[1] for a supposedly spatially symmetrical ground state

experiments became more and more precise over the years, culminating in the verification of perturbative QCD at the level of a few percent at the LEP in CERN.

Confinement

At other end of the energy scale there is confinement. At low momentum transfers perturbation theory is not applicable. This non-perturbative sector of QCD has turned out to be troublesome. Since the force between color charges does not decrease with distance, it is believed that quarks and gluons can never be liberated from hadrons. This has not been proven yet, but the Millennium Prize announced by the Clay Mathematics Institute may have to be disbursed in the near future if someone manages to link non-perturbative QCD with some other calculable theory using correspondences like the conjectured Anti de-Sitter / Conformal Field Theory (AdS/CFT) correspondence.

Due to confinement it has never been possible to observe free quarks. This absence of free quarks led to the belief that quarks were merely convenient mathematical constructs, not real particles. Richard Feynman argued that high energy experiments showed quarks to be real: he called them partons. James Bjorken proposed that certain relations should then hold in deep inelastic scattering of electrons and protons. These relations of scaling were spectacularly verified in experiments at SLAC in 1969. We are going to look into this in sections 7.1 and 7.2.

Lattice QCD

Confinement has not been proven but verified within LQCD computations: LQCD is the technique of a numerical evaluation of observables in QCD. To this end one chops the four-dimensional continuous Euclidean spacetime into small chucks, such that a lattice is generated. Using Monte Carlo methods, it is possible to sample the theory, enabling the extraction of masses and matrix elements even in the non-perturbative sector of QCD. One of the biggest drawbacks is the need for massive amounts of computing power due to the vast number of degrees of freedom. Simulating the pure gauge[2] sector has become feasible even on home computers. Quenched[3] simulations require much more computational effort due to the need for quarks propagators. Fully dynamical[4] simulations of the complete theory using chiral fermions have been in the development stage until a few years ago. Today, most research institutions are eager to see results from such 'fully' chiral simulations, but one is still limited by computational power. The chiral symmetry of (massless) fermions is an important feature which we are going to start to explain in the next section.

The Structure of Nucleons

In this PhD Thesis we will perform calculations using dynamical (nearly) chiral LQCD to calculate properties of the internal structure of nucleons. Specifically we are going to look at the lowest moment of structure functions of excited nucleons. This quantity is also known as the axial charge, a coupling strength parameter for weak interactions. The axial charge of the nucleon ground state has been studied widely for a number of different fermion actions [33, 34, 35, 36]. For results on excited nucleons high statistics and improved methods are necessary and few results are available [37, 38, 39].

[2]That is simulations without quarks.
[3]That is simulations without quark loops.
[4]That is simulations with quark loops.

Chapter 7

Continuum Quantum Chromodynamics

Quantum Chromodynamics (QCD) is the theory of quarks and their interaction by the exchange of gluons. One of its most fundamental properties is local gauge invariance under color-SU(3): From the plain fermionic Lagrangian[1]

$$\mathcal{L}_f(x) = \sum_{f=1}^{6} \bar{\psi}_f(x)(i\gamma^\mu \partial_\mu - m_f)\psi_f(x) \tag{7.1}$$

containing free quarks only, one can construct the full QCD Lagrangian by demanding this symmetry. It originates from the fact that physics must not depend on man-made coordinate systems, here for the color degrees of freedom. In general local gauge invariance means that the Lagrangian is invariant under a gauge transformation $g(x)$

$$\psi(x) \to \psi'(x) = g(x)\psi(x) = e^{-i\theta_a(x)T_a}\psi(x), \tag{7.2}$$

where T_a are the generators of the gauge group. This invariance can only be true if we replace the standard derivative ∂ by a covariant derivative D:

$$D_\mu = \partial_\mu + iA_\mu(x) \tag{7.3}$$

The transformation properties of the additional term

$$A_\mu(x) \to A'_\mu(x) = g(x)\left(A_\mu(x) - i\partial_\mu\right) g(x)^{-1} \tag{7.4}$$

are built in a way to render D covariant

$$D_\mu \to D'_\mu = g(x) D_\mu g(x)^{-1} \tag{7.5}$$

and thus the whole Lagrangian invariant. However, postulating a new symmetry does not come for free. We have introduced a new vector field $A^\mu(x)$. Up to now the Lagrangian does not involve any derivatives of $A^\mu(x)$. Deriving the classical equations of motion using the Euler-Lagrange formalism

$$\partial_\mu \left(\frac{\partial \mathcal{L}}{\partial(\partial_\mu \psi)}\right) = \frac{\partial \mathcal{L}}{\partial \psi} \qquad \partial_\mu \left(\frac{\partial \mathcal{L}}{\partial(\partial_\mu A^\nu)}\right) = \frac{\partial \mathcal{L}}{\partial A^\nu} \tag{7.6}$$

would lead to a theory with infinitely many parameters $A^\mu(x)$. Therefore we need to promote $A^\mu(x)$ to physical degrees of freedom by adding a kinetic term to the Lagrangian. It will have

[1] We use units where $\hbar = c = 1$ and the Einstein summation convention throughout this work.

to be invariant under (7.4). The simplest thing we can do is to exploit the covariance of D to build a new covariant field $F^{\mu\nu}$ by

$$F^{\mu\nu} = -i[D^\mu, D^\nu]. \tag{7.7}$$

From Wilson's renormalization group flow we know that at our low energy scales we may only probe relevant operators: The simplest nonzero invariant scalar using the constructed field is $\operatorname{Tr} F^{\mu\nu}F_{\mu\nu}$ such that the total Yang-Mills Lagrangian with some gauge coupling parameter g looks like

$$\mathcal{L}(x) = \sum_{f=1}^{6} \bar{\psi}_f(x)(i\not{D} - m_f)\psi_f(x) - \frac{1}{2g^2}\operatorname{Tr} F^{\mu\nu}(x)F_{\mu\nu}(x). \tag{7.8}$$

Finally the action is defined as

$$S[\bar{\psi}, \psi, A] = \int d^4x\, \mathcal{L}[\bar{\psi}, \psi, A](x). \tag{7.9}$$

There are at least two ways to obtain a quantum theory from the above action. One possibility is canonical quantization which imposes suitable commutation relations. A short introduction to this can be found in reference [40]. Perhaps the most powerful formalism is quantization by functional methods using the Feynman path integral. In this formalism the expectation value of any operator $\mathcal{O}[\psi, \bar{\psi}, A]$ can be obtained via

$$\langle \mathcal{O}[\bar{\psi}, \psi, A] \rangle = \frac{\int \mathcal{D}\bar{\psi}\mathcal{D}\psi\mathcal{D}A\ \mathcal{O}[\bar{\psi}, \psi, A]\ \exp\{iS[\bar{\psi}, \psi, A]\}}{\int \mathcal{D}\bar{\psi}\mathcal{D}\psi\mathcal{D}A\ \exp\{iS[\bar{\psi}, \psi, A]\}}. \tag{7.10}$$

Here the integration measures are formally defined as a product over all possible degrees of freedom: For fermions this is the color c, the flavor f and the spinor index α for every single space-time point $x \in \mathbb{R}^4$, while for vector bosons we have the directions μ for every generator T_a and every single space-time point $x \in \mathbb{R}^4$:

$$\begin{aligned} \mathcal{D}\psi &= \prod_{f,c,\alpha} \prod_{x\in\mathbb{R}^4} d\psi_f^{\alpha,c}(x), \\ \mathcal{D}\bar{\psi} &= \prod_{f,c,\alpha} \prod_{x\in\mathbb{R}^4} d\bar{\psi}_f^{\alpha,c}(x), \\ \mathcal{D}A &= \prod_{a,\mu} \prod_{x\in\mathbb{R}^4} dA_\mu^a(x). \end{aligned} \tag{7.11}$$

Beyond gauge invariance there are a number of other symmetries of our action. The Dirac operator

$$D(m) = i\not{D} - m \tag{7.12}$$

has a very special property in the massless limit. It obeys

$$D(0)\gamma_5 + \gamma_5 D(0) = 0. \tag{7.13}$$

This means the massless action is invariant under chiral transformations

$$\begin{aligned} \psi(x) &\to \psi'(x) = e^{i\epsilon\gamma_5}\psi(x) \\ \bar{\psi}(x) &\to \bar{\psi}'(x) = \bar{\psi}(x)e^{i\epsilon\gamma_5}. \end{aligned} \tag{7.14}$$

This symmetry and its spontaneous breaking are a central property of low-energy QCD. More detailed information can be found in standard textbooks like [40].

7.1 Deep Inelastic Scattering

Most of the structure of the nucleon has been determined from Deep Inelastic Scattering (DIS) experiments. That is a crash of a baryon and a lepton such that the baryon is completely torn apart by the huge momentum transfer from the incoming lepton. Details on this type of reactions can be found in [41, 42] or [40]. For example, scattering an electron off a nucleon may produce a number of possible states X. The corresponding lowest order Feynman graph is shown in figure 7.1. The momentum transfer $q = k - k'$ is space-like so we use $Q^2 = -q^2$

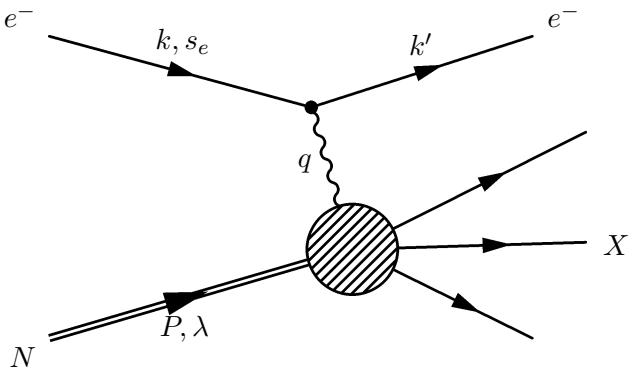


Figure 7.1: Lowest order DIS process.

and this implies an energy loss for the electron in the laboratory frame of $\nu = P \cdot q / m_N$. The matrix element of this process is

$$i\mathcal{M}(eN \to eX) = \bar{u}(k')\,(-ie\gamma^\mu)\,u(k, s_e)\left(\frac{-ig_{\mu\nu}}{q^2}\right)\int d^4x\, e^{iqx}\,\langle X|\, ieJ^\nu(x)\,|P,\lambda\rangle\,, \tag{7.15}$$

where $J^\mu(x)$ is the quark electromagnetic current

$$J^\mu = \sum_f Q_f \bar{q}_f \gamma^\mu q_f. \tag{7.16}$$

Here Q_f is the electromagnetic charge of quark flavor f in units of the elementary charge $|e|$. The hadronic matrix element $\langle X|\, J\,|N\rangle$ cannot be calculated by perturbation theory as the QCD coupling is large at the scale of the nucleon mass. To obtain the cross section from this matrix element, it must however be squared and summed over all possible final states X. Using the optical theorem we can write the result as a product

$$d\sigma \propto L_{\mu\nu} W^{\mu\nu}, \tag{7.17}$$

where the leptonic tensor is fully calculable using Quantum Electrodynamics (QED), the Quantum Field Theory (QFT) of Electrodynamics,

$$L_{\mu\nu} = 2\left(k_\mu k'_\nu + k'_\mu k_\nu - g_{\mu\nu}(k' \cdot k - m_e^2) + i\epsilon_{\mu\nu\rho\sigma} s_e^\rho q^\sigma\right), \tag{7.18}$$

while the hadronic part can be rewritten as

$$W^{\mu\nu}(P,q)_{\lambda'\lambda} = \frac{1}{4\pi}\int d^4x\, e^{iqx}\,\langle P,\lambda'|\,[J^\mu(x), J^\nu(0)]\,|P,\lambda\rangle\,. \tag{7.19}$$

For a spin-$\frac{1}{2}$ target like the nucleon with polarization s, we define a density matrix

$$\rho = \frac{1}{2}\left(1 + \frac{\vec{\sigma}\cdot\vec{s}}{M_N}\right) \tag{7.20}$$

in order to express W as

$$W^{\mu\nu}(P,q,s) = \mathrm{Tr}\,\rho W^{\mu\nu} = W_S^{\mu\nu} + iW_A^{\mu\nu}, \tag{7.21}$$

where in the second step we decompose the tensor into a symmetric and an antisymmetric part. The general Lorentz decomposition can be evaluated to

$$W_S^{\mu\nu} = \left(-g_{\mu\nu} + \frac{q^\mu q^\nu}{q^2}\right) F_1(x,Q^2) + \frac{1}{P\cdot q}\left(P^\mu - \frac{P\cdot q}{q^2}q^\mu\right)\left(P^\nu - \frac{P\cdot q}{q^2}q^\nu\right) F_2(x,Q^2), \tag{7.22}$$

$$W_A^{\mu\nu} = \frac{1}{P\cdot q}\epsilon^{\mu\nu\rho\sigma}q_\rho\left(s_\sigma g_1(x,Q^2) + \left(s_\sigma - \frac{s\cdot q}{P\cdot q}P_\sigma\right) g_2(x,Q^2)\right), \tag{7.23}$$

where the Bjorken-x, a famous scaling variable,

$$x = \frac{-t}{s+u} = \frac{Q^2}{2k\cdot P - 2k'\cdot P} = \frac{Q^2}{2P\cdot q} = \frac{Q^2}{2m_N\nu} \tag{7.24}$$

satisfies $0 \leq x \leq 1$ due to $s = (P+q)^2 \geq m_N^2$. It can be interpreted as a measure for the inelasticity of the scattering process, where $x = 1$ corresponds to the fully elastic limit. The functions F_1, F_2 and g_1, g_2 are called the structure functions of the nucleon. From the Lorentz structure of the leptonic tensor $L_{\mu\nu}$ we can see that F_1 and F_2 can be determined from unpolarized scattering experiments while in order to measure g_1 and g_2 polarized particles are necessary. Using the relative energy transfer

$$y = \frac{\nu}{E_e} = \frac{p\cdot q}{p\cdot k}, \tag{7.25}$$

the final cross section for unpolarized scattering can be written in a compact form:

$$\frac{d^2\sigma}{dx\,dy}(eN \to eX) = \frac{4\pi\alpha^2}{Q^4}s\left[xy^2F_1(x,Q^2) + (1-y)F_2(x,Q^2)\right]. \tag{7.26}$$

7.2 The Parton Model

The essence of the parton model is that at high energies a composite particle is nothing but a distribution of comoving quasi-noninteracting objects called partons. This idea originated from experiments but it became clear later that the running of the coupling constant of the strong force at low energies is very special: It decreases with the distance scale. This has given rise to the idea of asymptotic freedom: Particles are quasi-free at high energies: These are called partons. For a more in depth analysis of the parton model we again refer to [40]. In the language of Feynman diagrams a parton DIS process is depicted in figure 7.2. The comoving partons carry a certain "longitudinal fraction" ξ of the total nucleon momentum. The parton momentum is then $p = \xi P$ and the square of the center of mass energy of the electron-parton scattering neglecting parton and lepton masses is

$$\hat{s} = (p+k)^2 = 2p\cdot k = 2\xi P\cdot k = \xi s. \tag{7.27}$$

Since the scattered parton has a small mass (squared) compared to s and Q^2,

$$0 \approx (p+q)^2 = 2p\cdot q + q^2 = 2\xi P\cdot q - Q^2. \tag{7.28}$$

Therefore ξ coincides with Bjorken-x:

$$\xi = \frac{Q^2}{2P\cdot q} = x. \tag{7.29}$$

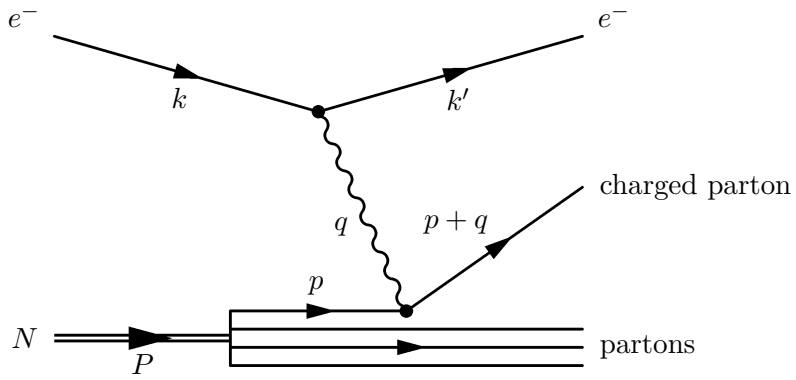


Figure 7.2: Kinematics of a deep inelastic electron scattering in the parton model.

When scattering off a single parton the hadronic tensor $W^{\mu\nu}$ has the same structure as the leptonic tensor $L^{\mu\nu}$, but with the incoming particle having momentum ξP. This means that for the full nucleon, built up from partons of type f with distributions $f_f(\xi)$, we can write the cross section as a sum of incoherent single parton scatterings:

$$\sigma(eN \to eX) = \int_0^1 d\xi \sum_{f \neq g} f_f(\xi)\sigma(ef(\xi P) \to ef(p')). \tag{7.30}$$

These partons can in principle be any of the six known quarks and their anti-quarks. Scattering off gluons at tree level is of course not possible, although they have a distribution $f_g(x)$. The sum of the momenta of all partons must be equal to the momentum of the full nucleon, so

$$\sum_f \int_0^1 dx\, x f_f(x) = 1. \tag{7.31}$$

Additionally, the number of valence quarks N_f leads to constraints

$$\int_0^1 dx \, \left[f_f(x) - f_{\bar{f}}(x) \right] = N_f. \tag{7.32}$$

Carrying out the integral over cross sections of single parton scatterings will result in a very simple form of the cross section for unpolarized scattering:

$$\frac{d^2\sigma}{dx\, dy}(eN \to eX) = \frac{2\pi\alpha^2}{Q^4} s \left[1 + (1-y)^2 \right] \left(\sum_f Q_f^2 x f_f(x) \right), \tag{7.33}$$

where the sum over the flavors may include the gluons as they have zero electric charge. So the parton model expressions for the structure functions can be extracted to be

$$\begin{aligned} F_1(x, Q^2) &= \sum_f \frac{Q_f^2}{2} f_f(x), \\ F_2(x, Q^2) &= \sum_f Q_f^2 x f_f(x), \\ g_1(x, Q^2) &= \sum_f \frac{Q_f^2}{2} \Delta f_f(x), \\ g_2(x, Q^2) &= 0, \end{aligned} \tag{7.34}$$

where we have first formed sum and difference:

$$f_f(x) = f_f^{\uparrow}(x) + f_f^{\downarrow}(x) \quad \text{(unpolarized)}, \tag{7.35}$$

$$\Delta f_f(x) = f_f^{\uparrow}(x) - f_f^{\downarrow}(x) \quad \text{(polarized)}. \tag{7.36}$$

Here $f_f^{\uparrow}(x)$ ($f_f^{\downarrow}(x)$) denotes the distribution function for a parton of flavor f with the same (opposite) helicity as that of the nucleon, respectively. The parton model structure functions satisfy the Callan-Gross relation $2xF_1(x) = F_2(x)$. In general the structure functions vary with x and Q^2. In the parton model, however, they are independent of Q^2. This leads to the famous Bjorken scaling. The tree level process in figure 7.2 will of course have to be corrected by higher order processes. This results in a dependence on Q^2, which is however relatively mild, on the order of $\ln Q^2$. These "scaling violations" were a direct test of QCD.

7.3 Axial and Vector Charges

The axial charge g_A of the ground state nucleon, or more precisely the ratio g_A/g_V is a central property of this baryon. It describes its coupling strength to the weak interaction and has been measured, e.g., from neutron β decay [43]:

$$g_A/g_V = 1.2670 \pm 0.0030$$

As explained in [44], the neutron β decay involves four form factors: vector g_V, tensor g_T, axial g_A and pseudo-scalar g_P:

$$\langle p|V_\mu^+|n\rangle = \bar{u}_p \left[\gamma_\mu g_V(q^2) - q_\lambda \sigma_{\lambda\mu} g_T(q^2)\right] u_n \tag{7.37}$$

$$\langle p|A_\mu^+|n\rangle = \bar{u}_p \left[\gamma_\mu \gamma_5 g_A(q^2) - i q_\mu \gamma_5 g_P(q^2)\right] u_n \tag{7.38}$$

where $V_\mu^+ = \bar{u}\gamma_\mu d$, $A_\mu^+ = \bar{u}\gamma_\mu\gamma_5 d$ and q_μ is the momentum transfer between proton and neutron. In the limit of zero momentum transfer $q^2 \to 0$ the axial and vector form factors dominate. Their values in this limit are called the vector and axial charges of the nucleon: $g_V = g_V(0)$ and $g_A = g_A(0)$. After the nucleon mass, the axial and vector charges are the most simple from the point of view of a LQCD calculation. Other nucleon form factors or moments of structure functions are more difficult to calculate technically. Additionally they obtain more noise and thus larger error bars from a non-zero momentum transfer that comes into play for more difficult operators like those including covariant derivatives.

In this work we neglect the mass differences of up and down quarks (and hence neutron and proton mass difference). In this case the global chiral $SU(2) \times SU(2)$ symmetry (when $m = m_u = m_d = 0$) is broken down to the vector $SU(2)$ subgroup and the associated vector charge $g_V = 1$ is still conserved. The axial symmetry is explicitly broken by $m \neq 0$. Additionally it is spontaneously broken and therefore $g_A \neq 1$ in general. The lattice calculation of axial charges relies on the preservation of vector symmetry. In this case one can write:

$$\langle p|V_\mu^+|n\rangle = 2\langle p|V_\mu^3|p\rangle = \langle p|V_\mu^u|p\rangle - \langle p|V_\mu^d|p\rangle \tag{7.39}$$

$$\langle p|A_\mu^+|n\rangle = 2\langle p|A_\mu^3|p\rangle = \langle p|A_\mu^u|p\rangle - \langle p|A_\mu^d|p\rangle, \tag{7.40}$$

where we abbreviate

$$2V_\mu^3 = V_\mu^u - V_\mu^d \tag{7.41}$$

$$2A_\mu^3 = A_\mu^u - A_\mu^d \tag{7.42}$$

$$V_\mu^{(q)} = \bar{q}\gamma_\mu q \tag{7.43}$$
$$A_\mu^{(q)} = \bar{q}\gamma_\mu\gamma_5 q. \tag{7.44}$$

This links the weak interaction properties to properties determined from the quark content of the state which can be calculated using LQCD: Since the electric charge of the nucleon is zero, it is possible to evaluate:

$$0 = \langle n|j^{\text{em}}|n\rangle = \frac{2}{3}\langle n|V^u|n\rangle - \frac{1}{3}\langle n|V^d|n\rangle \tag{7.45}$$
$$\Rightarrow \quad 2\langle n|V^u|n\rangle = \langle n|V^d|n\rangle \tag{7.46}$$

We exploit isospin symmetry:

$$2\langle p|V^d|p\rangle = \langle p|V^u|p\rangle \tag{7.47}$$

Since the electric charge of the proton is one, we can conclude

$$\langle p|j^{\text{em}}|p\rangle = \frac{2}{3}\langle p|V^u|p\rangle - \frac{1}{3}\langle p|V^d|p\rangle \tag{7.48}$$
$$= \langle p|V^u|p\rangle - \langle p|V^d|p\rangle = \langle p|V^+|n\rangle \tag{7.49}$$

and therefore obtain that the nucleon $g_V = 1$.

Generally, if we look at continuum matrix elements of states k of momentum p and spin s, where $s^2 = -p^2 = -m_k^2$, one defines Lorentz decompositions

$$\langle k|V_\mu^{(q)}|k\rangle = \langle k|\bar{q}\gamma_\mu q|k\rangle = 2v_1^{(k,q)}p_\mu \tag{7.50}$$
$$\langle k|A_\mu^{(q)}|k\rangle = \langle k|\bar{q}\gamma_\mu\gamma_5 q|k\rangle = a_0^{(k,q)}s_\mu \tag{7.51}$$

The operator product expansion relates such matrix elements to moments of structure functions. In the notation of the parton model one can give these values some interpretation:

$$v_n^{(q)} = \int_0^1 dx\, x^{n-1}\left[q(x) + (-1)^n\bar{q}(x)\right] = \langle x^{n-1}\rangle_q \tag{7.52}$$
$$a_n^{(q)} = \int_0^1 dx\, x^n\left[\Delta q(x) + (-1)^n\Delta\bar{q}(x)\right] = \langle x^n\rangle_{\Delta q} \tag{7.53}$$

In this model, $v_1^{(k,q)}$ is the 0th moment of the unpolarized quark distribution function $q(x)$ of a state k and counts its number of quarks q. For nucleons we have

$$g_V = v_1^{(u)} - v_1^{(d)} = 1 = \langle 1\rangle_{u-d}. \tag{7.54}$$

Next, $a_0^{(k,q)}$ is the 0th moment of the polarized quark distribution function $\Delta q(x)$ and determines the average spin fraction carried by all quarks q.

$$2g_A = a_0^{(u)} - a_0^{(d)} = \langle 1\rangle_{\Delta u - \Delta d}. \tag{7.55}$$

If we work with states of zero momentum, we get direct access to these interesting quantities:

$$\langle p|V_0^{(u-d)}|p\rangle = 2g_V p_0 \tag{7.56}$$
$$\langle p,s|A_i^{(u-d)}|p,s\rangle = 2g_A s_i \tag{7.57}$$

Chapter 8

Lattice Quantum Chromodynamics

The basic idea of LQCD is to actually evaluate the Feynman path integral (7.10) in a computer. We want to do this by Monte Carlo methods. A thorough introduction to these methods can be found in [45]. Unfortunately the Boltzmann weight factors e^{iS} in that path integral may be oscillating rapidly, prohibiting the use of statistical sampling. We need to write these factors in a form $e^{-S'}$, where S' is real and bounded from below. We can do this by switching from Minkowskian spacetime to Euclidean spacetime: To this end we let x^0 become purely imaginary

$$x^0 = -ix^4. \tag{8.1}$$

This implies $\partial_0 = i\partial_4$. With this formal substitution the plain fermionic action becomes

$$\begin{aligned} iS_f &= i \int d^4x \, \bar{\psi}(x)(i\gamma^\mu \partial_\mu - m)\psi(x) && (8.2)\\ &= -\int d^4x_E \, \bar{\psi}(x)(\gamma^E_\mu \partial_\mu + m)\psi(x) && (8.3)\\ &= -S^E_f. && (8.4) \end{aligned}$$

The details of this process of Euclideanization can be found in Appendix A.2. From now on we work in Euclidean spacetime and drop the E labels. Still, performing infinitely many integrals in a computer as prescribed in formula (7.11) is not possible. In order to renormalize our theory we need a regulator anyway, so we replace the continuous 4-dimensional spacetime with a hypercubic lattice

$$x = a\,(n_1, n_2, n_3, n_4) \quad \text{with} \quad n_\mu \in \{0, 1, ..., N_\mu - 1\}, \tag{8.5}$$

where the regulator a is the lattice spacing. On the boundaries we continue the lattice periodically in all four dimensions such that we obtain a 4-Torus.

8.1 Fermions on the Lattice

The fermions, being Grassmann degrees of freedom, have to obey antiperiodic boundary conditions in time direction. We need this to be able to use the transfer matrix formalism which we introduce in section 8.9. For the spatial boundary conditions we choose periodic ones, i.e.,

$$\begin{aligned} \psi(x_1 + aN_1, x_2, x_3, x_4) &= +\psi(x_1, x_2, x_3, x_4), && (8.6)\\ \psi(x_1, x_2, x_3, x_4 + aN_4) &= -\psi(x_1, x_2, x_3, x_4). && (8.7) \end{aligned}$$

Due to this discretization our integration measures for the fermions now look like

$$[d\psi] = \prod_{f,c,\alpha} \prod_{n} d\psi_f^{\alpha,c}(an), \tag{8.8}$$

$$[d\bar{\psi}] = \prod_{f,c,\alpha} \prod_{n} d\bar{\psi}_f^{\alpha,c}(an), \tag{8.9}$$

while integrations and derivatives turn (e.g.) into

$$\int d^4x \to a^4 \sum_n \tag{8.10}$$

$$\partial_\mu \psi(x) \to \frac{\psi(x + a\hat{\mu}) - \psi(x - a\hat{\mu})}{2a}. \tag{8.11}$$

Next we factor out all dimensions, such that all variables become dimensionless, i.e.,

$$m \to a^{-1} m, \tag{8.12}$$

$$\psi(x) \to a^{-3/2} \psi(n), \tag{8.13}$$

$$\partial_\mu \psi(x) \to \frac{a^{-3/2}}{2a} \left(\psi(n + \hat{\mu}) - \psi(n - \hat{\mu})\right). \tag{8.14}$$

8.1.1 Naive Discretization and Doublers

Now we can write down the fermionic lattice action:

$$S = \frac{1}{2} \sum_n \bar{\psi}(n) \gamma_\mu \left(\psi(n + \hat{\mu}) - \psi(n - \hat{\mu})\right) + m \sum_n \bar{\psi}(n) \psi(n) \tag{8.15}$$

$$= \sum_{n,m} \bar{\psi}(n) K(n, m) \psi(m). \tag{8.16}$$

We implicitly defined the naive lattice Dirac operator $D^{\text{naive}} = K$:

$$D^{\text{naive}}(n, m) = \sum_\mu \frac{1}{2} \gamma_\mu \left(\delta_{n+\mu,m} - \delta_{n-\mu,m}\right) + \mathbb{1} m \delta_{n,m}. \tag{8.17}$$

There is however a problem with this Dirac operator. It is called the *fermion doubling problem.* The problem becomes apparent when we calculate the propagator of this naive Dirac operator in momentum space. To this end we Fourier transform the operator by

$$\tilde{K}(k, l) = \sum_{n,m} e^{-ikn} K(n, m) e^{+ilm} \tag{8.18}$$

$$= \sum_{n,m} e^{-i(k-l)n} \left(\frac{1}{2} \sum_\mu \gamma_\mu \left(e^{il_\mu} - e^{-il_\mu}\right) + \mathbb{1} m\right) \tag{8.19}$$

$$= N \delta_{kl} \tilde{K}(l), \tag{8.20}$$

where $N = N_1 N_2 N_3 N_4$ is the total number of lattice sites and

$$\tilde{K}(l) = \mathbb{1} m + i \sum_\mu \gamma_\mu \sin(l_\mu). \tag{8.21}$$

When we invert this Fourier transformed Dirac operator in order to obtain the quark propagator

$$\tilde{K}^{-1}(l) = \frac{\mathbb{1} m - i \sum_\mu \gamma_\mu \sin(l_\mu)}{m^2 + \sum_\mu \sin^2(l_\mu)}, \tag{8.22}$$

we find that is does have right continuum limit near $l = 0$,

$$\lim_{a\to 0} a\tilde{K}^{-1}(l \approx 0) = \lim_{a\to 0} a\frac{\mathbb{1}am_{\text{phys}} - i\sum_\mu \gamma_\mu \sin(q_\mu a)}{a^2 m_{\text{phys}}^2 + \sum_\mu \sin^2(q_\mu a)} = \frac{\mathbb{1}m_{\text{phys}} - i\not{q}}{m_{\text{phys}}^2 + q^2} = S(q) \tag{8.23}$$

showing a pole in Minkowski space at $q^2 = m_{\text{phys}}^2$, corresponding to a particle with mass m_{phys}. But, seen most easily in the massless case, within the first Brillouin zone the full continuum limit consists of 15 more poles at q_i, where

$$\begin{aligned} q_i \in \Big\{ &\left(\frac{\pi}{a}, 0, 0, 0\right), \left(0, \frac{\pi}{a}, 0, 0\right), \ldots, \\ &\left(\frac{\pi}{a}, \frac{\pi}{a}, 0, 0\right), \ldots, \\ &\left(\frac{\pi}{a}, \frac{\pi}{a}, \frac{\pi}{a}, 0\right), \ldots, \\ &\left(\frac{\pi}{a}, \frac{\pi}{a}, \frac{\pi}{a}, \frac{\pi}{a}\right) \Big\}. \end{aligned} \tag{8.24}$$

These additional 15 poles/fermions are called the fermion doublers. They are unwanted lattice artifacts which manifest themselves as unphysical fermions and have to be removed in order to obtain sensible physics from LQCD.

8.1.2 Wilson Fermions

Fortunately there are many more lattice actions with the correct continuum limit. Wilson [46, 47] was the first who exploited this ambiguity and modified the naive Dirac operator in a way which removes the doublers from the theory by giving them an infinite mass in the continuum limit. He modified the naive fermionic action by adding another term:

$$S_f^{\text{Wilson}} = S_{\text{naive}} + S_{\text{new}} = \sum_{n,m} \bar{\psi}(n) D_{\text{Wilson}}(n,m)\psi(m). \tag{8.25}$$

In effect we obtain a modification of the Dirac operator:

$$D_{\text{Wilson}}(n,m) = \mathbb{1}m\delta_{n,m} + \sum_\mu \frac{1}{2}\gamma_\mu \left(\delta_{n+\mu,m} - \delta_{n-\mu,m}\right) - \mathbb{1}\sum_\mu \frac{1}{2}\left(\delta_{n+\mu,m} - 2\delta_{n,m} + \delta_{n-\mu,m}\right). \tag{8.26}$$

This additional term does not modify the continuum limit. By writing it in physical units

$$S_{\text{new}} = \frac{a^5}{2}\sum_n \sum_\mu \bar{\psi}(an)\frac{\psi(an + a\mu) - 2\psi(an) + \psi(an - a\mu)}{a^2} \tag{8.27}$$

$$\to \frac{a}{2}\int d^4x\, \bar{\psi}(x)\nabla^2\psi(x) \tag{8.28}$$

we see that it is going to vanish when $a \to 0$. On the lattice, however, the Fourier transformed Wilson Dirac operator looks like

$$\tilde{D}_{\text{Wilson}}(l) = \mathbb{1}m + i\sum_\mu \gamma_\mu \sin(l_\mu) + \mathbb{1}\sum_\mu \left(1 - \cos(l_\mu)\right). \tag{8.29}$$

The additional term vanishes at $l = 0$ producing the right propagator for the actual particle, but lifts the Dirac operator away from zero at the edges of the Brillouin zone by an extra

contribution of 2, which acts like an additional mass term for the doublers. Their mass is now given by

$$m_{\text{eff}} = m_{\text{phys}} + \frac{2l}{a} + \mathcal{O}(a^{-2}), \tag{8.30}$$

where l is the number of momentum components with $p_\mu = \pi/a$. This mass diverges in the continuum limit, such that the doublers decouple and get removed from the theory.

There is, however, one crucial problem with the Wilson action. It explicitly breaks chiral symmetry, even at vanishing quark mass, as

$$\{D^{-1}_{\text{Wilson}}, \gamma_5\} \neq 0 \quad \text{for} \quad m = 0. \tag{8.31}$$

This breaking of chiral symmetry of $\mathcal{O}(a)$ leads to a number of other difficulties: Fluctuations in the eigenvalues of the Dirac operator induced by this breaking increase the computational effort to obtain propagators for low quark masses. In addition, exceptional configurations show up which lead to difficulties in dynamical simulations. So in practical calculations one is restricted to rather large quark masses. Even more problematic is the fact that recovering chiral symmetry in the continuum limit can only be achieved by a fine tuning of the quark mass due to additive mass renormalization.

8.1.3 Clover-Wilson Fermions

In order to reduce discretization effects of the fermionic action, Sheikholeslami and Wohlert [48] proposed to add a special term to the Wilson action:

$$S_f^{\text{Wilson}} \to S_f^{\text{Wilson}} + c_{\text{sw}} \frac{i}{4} a \bar{\psi}(x) \sigma_{\mu\nu} F_{\mu\nu} \psi(x). \tag{8.32}$$

This goes under the name *clover improvement* as the common discretization of $F_{\mu\nu}$ has the form of a clover leaf. It includes the plaquettes that are attached to x in the μ-ν-plane. The coefficient c_{sw} of the clover term has to be determined non-perturbatively. This has been done by the authors of [49, 50] for a couple of lattice spacings. They provide an interpolation formula that enables the prediction of c_{sw} for a large range of β values.

8.1.4 Staggered Fermions

Another possibility of removing doublers is to double the effective lattice extent, taking doublers (to a certain amount) as real particles: Staggered fermions result from naive lattice fermions by a transformation [51, 52, 53, 54]

$$\begin{aligned} \psi(n) &= T(n)\xi(n), \\ \bar{\psi}(n) &= \bar{\xi}(n)T^\dagger(n), \\ T(n) &= \gamma_1^{n_1}\gamma_2^{n_2}\gamma_3^{n_3}\gamma_4^{n_4}, \end{aligned} \tag{8.33}$$

such that the Dirac operator gets *spin diagonal*:

$$T^\dagger(n)\gamma_\mu T(n) = \mathbb{1}\eta_\mu(n), \tag{8.34}$$

$$\eta_\mu(n) = (-1)^{n_1+n_2+\ldots+n_{\mu-1}}, \quad \eta_1(n) = 1, \tag{8.35}$$

$$S = \textstyle\sum_{n,\mu} \eta_\mu(n)\bar{\chi}(n)\partial_\mu\chi(n) + m\sum_n \bar{\chi}(n)\chi(n), \tag{8.36}$$

where the $\chi(n)$ are still 4-component spinors. This transformed Dirac operator does however not depend on that spin at all and the 4 components are therefore mere copies of each other,

i.e., doublers. We can remove these doublers by taking the $\chi(n)$ as 1-component fields. This way we managed to reduce the number of total doublers from 16 to $16/4 = 4$. As a bonus, the massless staggered fermion action has exact $U(1)_e \otimes U(1)_o$ symmetry

$$\left.\begin{array}{l}\chi(n) \to \chi'(n) = e^{i\phi_e}\chi(n) \\ \bar{\chi}(n) \to \bar{\chi}'(n) = \bar{\chi}(n)e^{i\phi_o},\end{array}\right\} \text{ for } \sum_\mu n_\mu \text{ even} \tag{8.37}$$

$$\left.\begin{array}{l}\chi(n) \to \chi'(n) = e^{i\phi_o}\chi(n) \\ \bar{\chi}(n) \to \bar{\chi}'(n) = \bar{\chi}(n)e^{i\phi_e},\end{array}\right\} \text{ for } \sum_\mu n_\mu \text{ odd} \tag{8.38}$$

which is a subgroup of the $SU(4)_L \otimes SU(4)_R \otimes U(1)_B$ chiral symmetry of the corresponding continuum theory, a theory with four mass degenerate fermions. When we recall[1] how fermions with mass m appear in the functional integral, $M = D + m$,

$$\langle \mathcal{O} \rangle = \frac{1}{Z}\int [dU][d\psi][d\bar{\psi}]\mathcal{O}(\psi_i\bar{\psi}_j)\exp\{-\bar{\psi}M\psi - S_G\} \tag{8.39}$$

$$= \frac{1}{Z}\int [dU]\mathcal{O}(M^{-1}_{ij})\det(M)\exp\{-S_G\}, \tag{8.40}$$

it may be possible that for staggered fermions

$$\langle \mathcal{O} \rangle = \frac{1}{Z}\int [dU]\mathcal{O}(M^{-1}_{ij})\det(M)^{1/4}\exp\{-S_G\} \tag{8.41}$$

approaches the right one fermion continuum limit. This is however unclear. More on staggered fermions can be found in review articles like [55].

8.2 Gluons on the Lattice

In chapter 7 we have derived the gauge field and its interaction with fermions by postulating local gauge invariance. Now we repeat that procedure for the lattice. To this end we need to render the fermionic action

$$S = \sum_{n,m}\psi(n)D(n,m)\psi(m) \tag{8.42}$$

invariant under the gauge transformations

$$\begin{array}{l}\psi(n) \to \psi'(n) = g(n)\psi(n), \\ \bar{\psi}(n) \to \bar{\psi}'(n) = \bar{\psi}(n)g^{-1}(n).\end{array} \tag{8.43}$$

Therefore the Dirac operator must transform as

$$D(n,m) \to D'(n,m) = g(n)D(n,m)g^{-1}(m). \tag{8.44}$$

If we remember for example the naive Dirac operator (8.17)

$$D^{\text{naive}}(n,m) = \sum_\mu \frac{1}{2}\gamma_\mu\left(\delta_{n+\mu,m} - \delta_{n-\mu,m}\right) + \mathbb{1}m\delta_{n,m}, \tag{8.45}$$

we see that we need to modify the discretized derivative and turn it into a covariant one. We need to modify each $\delta_{n,m}$ to a $\delta_{n,m}U(n,m)$, where $U(n,m)$ transforms just the way we need, namely

$$U(n,m) \to U'(n,m) = g(n)U(n,m)g^{-1}(m). \tag{8.46}$$

[1] We will see this in section 8.4. S_G is the gluonic part of the action to be introduced in section 8.2.

In the continuum this factor is well known as the Schwinger line integral:

$$U(x,y) = P\left\{\exp\left(i\int_x^y dz_\mu A^\mu(z)\right)\right\}. \tag{8.47}$$

Discretizing it on the lattice by

$$U_\mu(n) = U(n, n+\hat{\mu}) = U(x, x+a\hat{\mu}) \approx \exp\{iaA_\mu(n)\}, \tag{8.48}$$

$$U_{-\mu}(n) = U(n, n-\hat{\mu}) = U(n-\hat{\mu}, n)^\dagger = U_\mu(n+\hat{\mu}), \tag{8.49}$$

we see that the U field does not live on the sites of the lattice like the fermions but rather links from one site to the next, by transporting the gauge. In lattice language these objects are therefore called *link variables*. The Schwinger line integral becomes a product of links along a path $\mu_1, \mu_2, ...\mu_k$ connecting n and m:

$$U(n,m) = U_{\mu_1}(n)U_{\mu_2}(n+\hat{\mu}_1)\cdots U_{\mu_k}(m-\hat{\mu}_k). \tag{8.50}$$

Using these gauge transporters to construct gauge invariant quark bilinears $\bar{\psi}(n)U(n,m)\psi(m)$ we can write down the covariant version of the naive Dirac operator

$$D^{\text{naive}}(n,m) = \sum_\mu \frac{1}{2}\gamma_\mu\left(U_\mu(n)\delta_{n+\mu,m} - U_{-\mu}(n)\delta_{n-\mu,m}\right) + \mathbb{1}m\delta_{n,m} \tag{8.51}$$

or the covariant version of the Wilson Dirac operator:

$$\begin{aligned} D^{\text{Wilson}}(n,m) = &\sum_\mu \frac{1}{2}\gamma_\mu\left(U_\mu(n)\delta_{n+\mu,m} - U_{-\mu}(n)\delta_{n-\mu,m}\right) + \mathbb{1}m\delta_{n,m} \\ &- \mathbb{1}\sum_\mu \frac{1}{2}\left(U_\mu(n)\delta_{n+\mu,m} - 2\delta_{n,m} + U_{-\mu}(n)\delta_{n-\mu,m}\right). \end{aligned} \tag{8.52}$$

8.2.1 Naive Discretization

As in the continuum we had to introduce new degrees of freedom to obtain an action that is invariant under local gauge transformations. We need to give them a kinetic term. As in the continuum we use the most simple nonzero gauge invariant object we can construct. The gauge transporters transform as

$$U(n,m) \to U'(n,m) = g(n)U(n,m)g^{-1}(m), \tag{8.53}$$

so we can take any closed path to build such an invariant object by

$$\operatorname{Tr} U(n,n) \to \operatorname{Tr} U'(n,n) = \operatorname{Tr}\left(g(n)U(n,n)g^{-1}(n)\right) = \operatorname{Tr} U(n,n). \tag{8.54}$$

The most simple paths we can take on the lattice are the plaquettes, squares with edge length a along links μ and ν starting at n. So we define:

$$U_{\mu\nu}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_{-\mu}(n+\hat{\mu}+\hat{\nu})U_{-\nu}(n+\hat{\mu}). \tag{8.55}$$

From this quantity Wilson constructed the Wilson gauge action [46], for SU(N) defined as

$$S_{\text{Wilson,G}} = \beta\sum_n\sum_{\mu<\nu}\left(1 - \frac{1}{N}\operatorname{Re}\operatorname{Tr} U_{\mu\nu}(n)\right), \tag{8.56}$$

where the connection between β and the bare coupling is

$$\beta = \frac{2N}{g^2}. \tag{8.57}$$

Indeed it can be shown by expanding (8.56) to order $\mathcal{O}(a^2)$ that Wilson's gauge action is a discretized version of the continuum gauge action:

$$S_{\mathrm{W,G}} = -\frac{1}{2g^2}\int d^4x \,\mathrm{Tr}\, F_{\mu\nu}(x)F_{\mu\nu}(x) + \mathcal{O}(a^2). \tag{8.58}$$

8.2.2 Lüscher-Weisz Gauge Action

To reduce the discretization effects, Lüscher and Weisz [56, 57] constructed a gauge action for that only some $\mathcal{O}(g^2a^2)$ errors remain for onshell quantities. It was found that the these remaining errors are unusually small, similar to errors of $\mathcal{O}(a^4)$. Once needs to more complex loops for gauge transporters into account: They add 2×1 rectangles R and $1\times 1\times 1$ parallelograms[2] P to the 1×1 squares S Wilson used:

$$\begin{aligned} S_{\mathrm{LW,G}} = \beta_0 \sum_S \left(1 - \frac{1}{N}\mathrm{Re}\,\mathrm{Tr}\, U^S\right) \\ + \beta_1 \sum_R \left(1 - \frac{1}{N}\mathrm{Re}\,\mathrm{Tr}\, U^R\right) \\ + \beta_2 \sum_P \left(1 - \frac{1}{N}\mathrm{Re}\,\mathrm{Tr}\, U^P\right), \end{aligned} \tag{8.59}$$

$$U^S_{\mu\nu}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_{-\mu}(n+\hat{\mu}+\hat{\mu})U_{-\nu}(n+\hat{\mu}), \tag{8.60}$$

$$U^R_{\mu\nu}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_\nu(n+\hat{\mu}+\hat{\nu})U_{-\mu}(n+\hat{\mu}+2\hat{\nu})U_{-\nu}(n+2\hat{\nu})U_{-\nu}(n+\hat{\nu}), \tag{8.61}$$

$$U^P_{\mu\nu\xi}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_\xi(n+\hat{\mu}+\hat{\nu})U_{-\mu}(n+\hat{\mu}+\hat{\nu}+\hat{\xi})U_{-\nu}(n+\hat{\nu}+\hat{\xi})U_{-\xi}(n+\hat{\xi}). \tag{8.62}$$

The lattice gauge coupling $\beta_0 = \beta$ takes the same role as in the standard Wilson discretization. The parameters β_1 and β_2 however need to be set to a specific value in order to cancel $\mathcal{O}(a^2)$ discretization errors. They have been determined within the framework of tadpole improved perturbation theory [58]. They are given in [59] in terms of the expectation value of the plaquette variable u_0^4:

$$u_0^4 = \frac{1}{3}\mathrm{Re}\,\mathrm{Tr}\,\langle U_{\mathrm{plaq}}\rangle, \tag{8.63}$$

$$\alpha = -\frac{\ln u_0^4}{3.06839}, \tag{8.64}$$

$$\beta_1 = -\frac{\beta_0}{20u_0^2}\left(1 + 0.4805\alpha\right), \tag{8.65}$$

$$\beta_2 = -\frac{\beta_0}{u_0^2}0.03325\alpha. \tag{8.66}$$

In all our numerical computations we use the Lüscher-Weisz gauge action exclusively (with these coefficients).

[2] actually "twisted chairs"

8.3 Chiral Improvements

8.3.1 Nielsen-Ninomiya No-Go theorem

Ideally the LQCD community would like to use a Dirac operator that closely resembles the continuum Dirac operator. At zero quark mass the most important features are:

1. Correct continuum limit: $\tilde{D}(p) = i\gamma_\mu p_\mu + \mathcal{O}(p^2)$.
2. There are no doublers, i.e., $\tilde{D}(p)$ is invertible for $p \neq 0$.
3. Exact chiral symmetry, so $\gamma_5 D + D\gamma_5 = 0$.
4. Locality, meaning $|D(x,y)| < Ce^{-\gamma|x-y|}$.

The Nielsen-Ninomiya theorem [60, 61] states that these properties cannot be implemented on the lattice at the same time.

8.3.2 Ginsparg-Wilson Fermions

In 1982 Ginsparg and Wilson [62] proposed a way to evade the No-Go theorem by relaxing chiral symmetry to

$$\{D, \gamma_5\} = 2aDR\gamma_5 D \quad \text{or} \quad \{D^{-1}, \gamma_5\} = 2aR\gamma_5. \tag{8.67}$$

In contrast to the naive Wilson fermion action, which violates chiral symmetry at any lattice spacing, Ginsparg-Wilson fermions violate chiral symmetry in a controlled and in fact minimal way. This lattice version of chiral symmetry reduces to the continuum chiral symmetry upon a continuum extrapolation. Here R can be any local operator which is diagonal in Dirac space. In most applications $R = 1/2$ is used, simplifying the Ginsparg-Wilson relation to

$$\{D^{-1}, \gamma_5\} = a\gamma_5. \tag{8.68}$$

This lattice version of chiral symmetry leads to an invariance of the action under the transformation

$$\begin{aligned} \psi &\to \psi' = e^{i\epsilon\gamma_5\left(1-\frac{1}{2}aD\right)}\psi, \\ \bar{\psi} &\to \psi' = \bar{\psi}e^{i\epsilon\gamma_5\left(1-\frac{1}{2}aD\right)}, \end{aligned} \tag{8.69}$$

which is a slight modification of formula (7.14). This lattice chiral symmetry has amazing consequences for the eigenvalues of the Dirac operator:

Let λ be one of its eigenvalues and $|\lambda\rangle$ the associated eigenvector. Then using γ_5 hermiticity[3]

$$(\lambda + \lambda^*)\gamma_5|\lambda\rangle = (\gamma_5 D + D\gamma_5)|\lambda\rangle = aD\gamma_5 D|\lambda\rangle = a\lambda^*\lambda\gamma_5|\lambda\rangle \tag{8.70}$$

we see that all eigenvalues of D lie on a circle in the complex plane around $z = 1/a$ with radius $1/a$:

$$2\text{Re }\lambda = a|\lambda|^2. \tag{8.71}$$

This is a marvelous property as in the continuum limit the circle gets larger and approaches the imaginary axis and thus the physical spectrum. The doublers at the other end of the

[3] γ_5 hermiticity is a property of the Dirac operator: $(D\gamma_5)^\dagger = D\gamma_5$. The conjugation operates on all (color, spin and spacetime) indices.

circle correspond to larger and larger eigenvalues (effective masses) and decouple from physical quantities.

In order to get a massive Dirac operator we need to shift and rescale a massless Ginsparg-Wilson D operator by

$$D_m = \left(1 - \frac{1}{2}am\right) D + m. \tag{8.72}$$

This gives the actual particle a mass m, whereas the doublers stay around $2/a$. This definition leads to a linear relation between bare and renormalized quark mass up to $\mathcal{O}(a)$ [63].

8.3.3 Overlap Fermions

There are several Dirac operators available, which fulfill the Ginsparg Wilson relation either exactly or approximately. The most popular solution for $R = 1/2$ found by Neuberger [64, 65] is exact and is called the overlap operator [66]. It can be constructed from any γ_5-hermitian lattice Dirac operator D that is free of doublers, like the Wilson Dirac operator $D_{\text{Wilson}}(m)$ with a negative mass parameter $m = -\rho$ by[4]

$$D_{\text{Overlap}} = \mathbb{1} + \gamma_5 \text{ sgn } \left(\gamma_5 D_{\text{Wilson}}(-\rho)\right). \tag{8.73}$$

Is seems that this Dirac operator is from a physical point of view the best one we can use right now. However, the accuracy of the calculation of the sign of a matrix determines the quality of the implementation of the Ginsparg-Wilson relation. At the moment an exact calculation of this sign is almost prohibitively expensive in computer time. More importantly, the action resulting from this Dirac operator is discontinuous for changes in topological charge. This leads to problems with typical Hybrid Monte Carlo (HMC) algorithms used in dynamical simulations.

8.3.4 Domain-Wall Fermions

The domain wall method, invented in the nineties by Kaplan [67], extends Wilson fermions into a fifth unphysical dimension of a lattice. The idea was that zeros in a mass term dependent on this fifth dimension will trap chiral fermions at the domain walls. See for example [68]. This technique has been elaborated on for full QCD in [69, 70]. Ideally one could simulate chiral quarks using this method. But the available computational power limits the extent N_5 of the fifth dimension. The result is an approximate chiral symmetry. The error introduced is on the order of $e^{-\alpha N_5}$.

8.3.5 Fixed Point Fermions

Hasenfratz and Niedermayer have investigated non-perturbative renormalization group blocking transformations on the lattice [71]. The fixed points of such transformations correspond to lattice actions that are completely free of discretization errors. They are called perfect actions. Classically perfect actions have been constructed for pure gauge theories [72] as well as for free Wilson [73] and staggered fermions [74]. Classically perfect actions for full QCD have been elaborated on in [75, 76]. Historically, the Ginsparg Wilson relation [62] got the attention it deserved only after Hasenfratz [77] rediscovered it in the process of his search for perfect actions.

However, due to the complexity of the fixed point equation one is for practical simulations again limited to approximate solutions [78, 79]. Hence one has to work with an approximate lattice chiral symmetry.

[4] The sign of a matrix is evident in its eigenbasis.

8.3.6 Chirally Improved fermions

Any exact solution of the Ginsparg-Wilson equation known so far is extremely expensive in computer time. As for example the sign function of the Overlap operator cannot be evaluated exactly anyway, an approximate solution to the Ginsparg-Wilson relation may be an excellent compromise. Consequently Gattringer, Hip and Lang [80, 81] constructed such an approximate solution, in the following called Chirally Improved (CI) Dirac operator D_{CI}. This is done by first writing down the most general Dirac operator $D(x, y)$ on the lattice. This means that for fixed (x, y) it may be a linear combination of any Clifford algebra generator Γ_α times any gauge transporter leading from x to y:

$$D(x,y) = \sum_{\alpha=1}^{16} \Gamma_\alpha \sum_{p \in \mathcal{P}_{x,y}} c_p^\alpha \prod_{l \in p} U_l. \tag{8.74}$$

For every generator Γ_α and possible path $p \in \mathcal{P}_{x,y}$ we label the coefficients as c_p^α. This general[5] operator is now to be constrained by a number of desired properties. Among those are translation and rotation invariance, invariance under charge conjugation and parity and γ_5-hermiticity. When implementing all these symmetries we find that paths in the original Ansatz become grouped together: Denoting a path from n to m over k consecutive hops along gauge links in directions $\mu_1, \mu_2, \ldots, \mu_k$ as

$$\langle \mu_1, \mu_2, \ldots, \mu_k \rangle = \delta_{m, n+\sum_k \hat{\mu}_k} U_{\mu_1}(n) U_{\mu_2}(n+\hat{\mu}_1) \cdots U_{\mu_k}(m-\hat{\mu}_k) \tag{8.75}$$

we can write the CI Dirac operator as

$$\begin{aligned}
D(n,m) =& \mathbb{1} \left(s_1 + s_2 \sum_{l_1} \langle l_1 \rangle + s_3 \sum_{l_2 \neq l_1} \langle l_1, l_2 \rangle + s_4 \sum_{l_1} \langle l_1, l_1 \rangle + \ldots \right) \\
&+ \sum_\mu \gamma_\mu \sum_{l_1 = \pm\mu} \mathrm{sgn}(l_1) \left(v_1 \langle l_1 \rangle + v_2 \sum_{l_2 \neq \pm\mu} [\langle l_1, l_2 \rangle + \langle l_2, l_1 \rangle] + v_3 \langle l_1, l_1 \rangle + \ldots \right) \\
&+ \sum_{\mu<\nu} \gamma_\mu \gamma_\nu \sum_{\substack{l_1=\pm\mu \\ l_2=\pm\nu}} \mathrm{sgn}(l_1 l_2) \sum_{i,j=1}^{2} \epsilon_{ij} \left(t_1 \langle l_i, l_j \rangle + \ldots \right) \\
&+ \sum_{\mu<\nu<\rho} \gamma_\mu \gamma_\nu \gamma_\rho \sum_{\substack{l_1=\pm\mu \\ l_2=\pm\nu \\ l_3=\pm\rho}} \mathrm{sgn}(l_1 l_2 l_3) \sum_{i,j,k=1}^{3} \epsilon_{ijk} \left(a_1 \langle l_i, l_j, l_k \rangle + \ldots \right) \\
&+ \gamma_5 \sum_{\substack{l_1=\pm1 \\ l_2=\pm2 \\ l_3=\pm3 \\ l_4=\pm4}} \mathrm{sgn}(l_1 l_2 l_3 l_4) \sum_{i,j,k,o=1}^{4} \epsilon_{ijko} \left(p_1 \langle l_i, l_j, l_k, l_o \rangle + \ldots \right).
\end{aligned} \tag{8.76}$$

See table A.2 for a list of paths corresponding to the coefficients s_i, v_i, a_i, t_i and p_i. In the above formula we do not show terms corresponding to longer paths like s_5, s_6, etc. We want a

[5] There is another possible generalization: Any of the mentioned gauge transporters from x to y can be multiplied with any combination of traces of gauge transporter loops (Wilson loops), as they are gauge invariant. These degrees of freedom are, however, not taken into account for the CI Dirac operator.

lattice chiral Dirac operator, so we insert the full expansion into the Ginsparg-Wilson relation and find a set of coupled equations of the remaining coefficients s_i, v_i, a_i, t_i, p_i:

$$\begin{aligned}
0 &= -2s_1 + s_1^2 + 8s_2^2 + 18s_3^2 + 8s_4^2 + 8v_1^2 + 96v_2^2 + 8v_3^2 + 48t_1^2 + 192a_1^2 + 384p_1^2 + \dots \\
0 &= -2s_2 + 2s_1s_2 + 12s_2s_3 + 2s_2s_4 + 12v_1v_2 + 2v_1v_3 + \dots \\
0 &= -2s_3 + 2s_1s_3 + s_2^2 + 4s_3^2 + 2s_3s_4 + 4v_2^2 + 2v_2v_3 + \dots \\
0 &= -2s_4 + 2s_1s_4 + s_2^2 + 6s_3^2 - v_1^2 - 6t_1^2 - 24a_1^2 - 48p_1^2 + \dots \\
0 &= -s_2v_1 - 4s_3v_2 - 2s_4v_2 - s_3v_3 - v_3t_1 - 4v_2t_1 + \dots \\
0 &= -2t_1 + 2s_1t_1 - 2s_4t_1 - v_1^2 - 4v_2^2 - 2v_2v_3 - 4t_1^2 + 8v_1a_1 - 8a_1^2 + 16t_1p_1 + \dots \\
0 &= -s_2a_1 + v_2t_1 - v_3p_1 + \dots \\
0 &= -v_2t_1 + \dots \\
0 &= -s_2a_1 - 2v_2p_1 + \dots \\
0 &= -2s_2a_1 + 2v_2t_1 - 4v_2p_1 + \dots \\
0 &= -2p_1 + 2s_1p_1 - 2s_4p_1 - 2v_1a_1 + t_1^2 + \dots
\end{aligned} \tag{8.77}$$

We again omitted terms corresponding to longer paths. This set of equations is equivalent to the Ginsparg-Wilson relation. Since each of these equations contains infinitely many terms in s_i, v_i, a_i, t_i, p_i we need to cut this series off somewhere in order to obtain a solution. As explained in [80, 81] the Ginsparg-Wilson condition enforces locality on the Dirac operator, which is an exponential decay of the coefficients with their path lengths. Therefore, most naturally we remove paths with large lengths from the above coupled set of equations rendering the final CI Dirac operator ultra-local.

Further, we have to ensure the correct continuum limit:

$$\tilde{D}(p) = i\not{p} + \mathcal{O}(p^2). \tag{8.78}$$

This leads to two more equations for the coefficients, one for the constant term to vanish and the second for the slope to be 1. We add the equations

$$\begin{aligned}
0 &= s_1 + 8s_2 + 48s_3 + 8s_4 + \dots \\
1 &= 2v_1 + 24v_2 + 4v_3 + \dots
\end{aligned} \tag{8.79}$$

to the set. The first of these is implicitly guaranteed by the Ginsparg-Wilson circle touching zero. We however cut off the exact Ginsparg-Wilson relation, so we enforce this necessary property on our new Dirac operator.

From the point of view of a numerical simulation short paths and ultra-locality are a very nice property. It reduces the memory size of the Dirac operator and speeds up the computation of propagators. The choice of terms to be actually included is influenced by two considerations: On the one hand, any additional term certainly improves the quality of the approximation of Ginsparg-Wilson fermions. On the other hand, each new term drives up the cost of a numerical treatment.

Unfortunately it turns out that one needs rather long gauge transporters to obtain a decent continuum limit. The idea [81] was to reinforce the right continuum limit by modifying (8.79) to

$$\begin{aligned}
0 &= s_1 + 2s_2z_s + 48s_3z_s^2 + 8s_4z_s^2 + \dots, \\
1 &= 2v_1z_v + 24v_2z_v^2 + 4v_3z_v^2 + \dots.
\end{aligned} \tag{8.80}$$

The real factors $z_s(\beta)$ for the scalar terms and $z_v(\beta)$ for the vector terms are adjusted from a lattice simulation at β such that locality is improved. This introduces a dependence of the

coefficients of the CI Dirac operator on the gauge coupling. Each coefficient is multiplied by a factor z^n where n is the length of the associated gauge transporter. This gives long paths high weights, which in turn gives them small coefficients. The approximated Ginsparg-Wilson equivalents (8.77) are not adjusted. This way one can optimize the properties of D_{CI} without sacrifying lattice chirality. Instead one shortens paths by selecting more local actions in the huge space of possible lattice actions satisfying the Ginsparg-Wilson condition.

Next we would like to implement $\mathcal{O}(a)$ improvement. Exact solutions of the Ginsparg-Wilson relation do already implement this property. Since we approximated this relation, we put tree level improvement back by

$$s_2 + 12s_3 + 4s_4 + \ldots = 4t_1 + 32t_2 + 16t_3 + \ldots \tag{8.81}$$

In the case of Clover-Wilson fermions this would amount to a tree level value of $c_{\mathrm{sw}} = 1$ for the clover-leaf term.

Finally it has been checked in [81] that the doublers live near the right end of the Ginsparg-Wilson circle and are being shifted to larger and larger eigenvalues in the continuum limit.

Due to the excellent tradeoff between good chiral properties and computational effort we are going to use these CI fermions throughout this work in any numerical simulation. Specifically we use the coefficients given in Appendix A.3.

8.4 The Quenched Approximation

Now that we know which action

$$S[\bar{\psi}, \psi, U] = S_{\mathrm{fermion}}[\bar{\psi}, \psi, U] + S_{\mathrm{gauge}}[U] \tag{8.82}$$

to use we are almost ready to perform a LQCD simulation by calculating some path integral expectation value

$$\langle \mathcal{O}[\psi, \bar{\psi}, U] \rangle = \frac{\int [d\bar{\psi}][d\psi][dU]\ \mathcal{O}[\bar{\psi}, \psi, U]\, e^{-S[\bar{\psi}, \psi, U]}}{\int [d\bar{\psi}][d\psi][dU]\ e^{-S[\bar{\psi}, \psi, U]}}. \tag{8.83}$$

We have to integrate over all possible gauge and fermion fields:

$$\begin{aligned} [d\psi] &= \prod_{f,c,\alpha} \prod_n d\psi_f^{\alpha,c}(n), \\ [d\bar{\psi}] &= \prod_{f,c,\alpha} \prod_n d\bar{\psi}_f^{\alpha,c}(n), \\ [dU] &= \prod_\mu \prod_n dU_\mu(n). \end{aligned} \tag{8.84}$$

As already mentioned above, a full evaluation of this integral is too expensive, so we use Monte Carlo methods. The gauge links can be sampled relatively easily. The fermion fields are a little more difficult to treat as they are Grassmann variables: We integrate out the fermion fields analytically:

$$\begin{aligned} \langle \mathcal{O}[\psi, \bar{\psi}, U] \rangle &= \frac{\int [dU]\ e^{-S_{\mathrm{gauge}}[U]} \left(\int [d\bar{\psi}][d\psi]\ \mathcal{O}[\bar{\psi}, \psi, U]\, e^{-\sum_f \bar{\psi}_f D_f[U] \psi_f} \right)}{\int [dU]\ e^{-S_{\mathrm{gauge}}[U]} \left(\int [d\bar{\psi}][d\psi]\ e^{-\sum_f \bar{\psi}_f D_f[U] \psi_f} \right)} \\ &= \frac{\int [dU]\ e^{-S_{\mathrm{gauge}}[U]} \left(\mathcal{O}[D_f^{-1}[U]], U] \prod_f \det D_f[U] \right)}{\int [dU]\ e^{-S_{\mathrm{gauge}}[U]} \left(\prod_f \det D_f[U] \right)} \\ &= \langle \mathcal{O}[D_f^{-1}[U], U] \rangle_U. \end{aligned} \tag{8.85}$$

In the literature, this integration of the fermion fields is often referred to as fermion contraction. The operator $\mathcal{O}$ is now a functional of gauge links only

$$\mathcal{O} = \mathcal{O}[D_f^{-1}[U], U]. \tag{8.86}$$

This forces us to evaluate propagators $D_f^{-1}[U]$ on each configuration. A tremendous simplification of the numerical effort is obtained by using the quenched approximation. If we put

$$\det D_f[U] = \text{const.} \tag{8.87}$$

we do not need to evaluate the determinant of a $12V \times 12V$ matrix, which is extremely difficult. In this case the path integral simplifies to

$$\langle \mathcal{O}[U] \rangle_U = \frac{\int [dU]\ e^{-S_{\text{gauge}}[U]} \mathcal{O}[D_f^{-1}[U]], U]}{\int [dU]\ e^{-S_{\text{gauge}}[U]}}. \tag{8.88}$$

The quenched approximation is often referred to as $n_f = 0$ approximation, as setting the determinant of D constant effectively gives the quarks an infinite mass such that they are unable to propagate. This is due to the fact that in the determinant of the massive Dirac operator

$$\det M = \det(D + m) \approx m^{12V} = \text{const.} \tag{8.89}$$

the massless Dirac operator gets suppressed relative to the large quark mass.

Unfortunately this vast simplification of full QCD is uncontrolled. From a perturbative viewpoint the quenched approximation neglects all internal quark loops. Nevertheless it seems to work surprisingly well in many cases.

8.5 The Continuum Limit

Any relativistic quantum field theory has to be regularized to define the meaning of the integration measures in the path integral. In the continuum we can do this by using dimensional regularization introducing a regulator ϵ which will have to be sent to zero in the final results.

In a lattice gauge theory, the lattice is the regulator of the theory. It is essentially a UV-cutoff which limits momenta to the first Brillouin zone $|p_\mu| < \pi/a$. After having obtained final results from a lattice calculation we have to send $a \to 0$ to recover continuum physics, i.e., do a continuum extrapolation.

8.6 Setting the Scale

Now that we know how to simulate QCD on a computer there is still one problem to tackle: Computers do only know about numbers. Hence, as constructed, our lattice action does not involve any dimensionful parameter: Besides the dimensionless gauge coupling β, the action involves masses m, however always in a combination am, where a is the lattice constant. The point is that a is the regulator of LQCD. This means that when we perform a simulation on our computer with a fixed value of β we have actually already chosen a specific lattice constant. Like in continuum renormalization, we have to determine its physical value by calculating some dimensionful quantity. We can do this looking at some quantity whose physical value we know about: By a lattice calculation of this quantity we obtain a dimensionless value like am, if we were calculating a mass. Now we have to fix a such that m agrees with its physical value. This procedure is called “setting the scale”.

The method by Sommer

The most prominent method of "setting the scale" has been introduced by Sommer in [82, 83]. It utilizes the static quark potential $V(r)$. A common though not necessary parameterization is

$$V(r) = A + \frac{B}{r} + \sigma r, \tag{8.90}$$

where σ is the so-called "string tension". From QCD phenomenology one expects a value of $\sigma \approx 800\,\mathrm{MeV/fm}$. The parameter B is responsible for the strength of the Coulomb part of the potential and A is some irrelevant normalization of the energy. There is a characteristic length scale r_0 tied to this static potential. In physical units this distance is about 0.5 fm. It is defined by the equation

$$r_0^2 \left.\frac{\partial}{\partial r} V(r)\right|_{r_0} = -B + \sigma r_0^2 = 1.65. \tag{8.91}$$

In lattice units this amounts to

$$\frac{r_0}{a} = \sqrt{\frac{1.65 + B}{a^2\sigma}}. \tag{8.92}$$

On the lattice we are able to compute this static quark potential from Wilson loops and can obtain the numbers B and $a^2\sigma$ by fits of $V(r)$ to the potential calculated. Using the physical value of r_0 we can now extract the renormalized value of the lattice constant a in physical units.

8.7 Propagators and Chiral Extrapolations

One big difficulty in obtaining results from lattice calculations is that simulations at small quark masses are expensive in computer time. In the process of a Monte Carlo simulation one computes observables using the generated configurations. To include fermions one has to invert the Dirac operator $D[U]$ on each of these configurations to obtain the quark propagators $D^{-1}[U]$. The Dirac operator is a huge matrix with edge length $V \cdot N_c \cdot N_d \approx 10^6$. A full inversion of such a matrix is prohibitively expensive. Fortunately, in most lattice simulations one can use properties like γ_5-hermiticity such that only few columns of the full propagator are necessary. This works well for calculations where only connected diagrams appear but fails when disconnected diagrams show up. In this case one is forced to estimate the full propagator using stochastic methods. Nevertheless the computation of a single column of the full matrix $M = D[U] + m$ or equivalently the solution v for some source b by

$$Mv = b \tag{8.93}$$

is still not cheap. In most cases the solution is computed iteratively by conjugate gradient methods [84] which analytically converge in $n = VN_cN_d$ steps. In order to obtain a numerical solution v to a certain relative accuracy

$$\frac{\|Mv - b\|}{\|b\|} < \epsilon \tag{8.94}$$

one will have to spend computer time that increases with the condition number $\kappa(M)$ of the massive Dirac operator M

$$\kappa(M) = \left|\frac{\lambda_{\max}}{\lambda_{\min}}\right|, \tag{8.95}$$

which is the ratio of its largest and its smallest eigenvalue. For small quark masses κ gets large as the lowest eigenvalues get shifted towards zero. Therefore one cannot simulate realistic quark masses of a few MeV at the moment, but is forced to calculate at larger quark masses and extrapolate results down to physical quark masses. This is done by fitting lattice results to predictions of chiral perturbation theory [85, 86, 87]. These depend on the mass of the pions which are the (pseudo) Goldstone bosons of chiral symmetry (breaking).

Since the determination of the quark propagators is numerically quite expensive, one often stores these for each gauge configuration obtained from the Monte Carlo simulation and evaluates them later on to physical results like masses etc.

8.8 Finite Volume Effects

In lattice gauge theory we want to simulate continuum physics. We introduce a finite lattice spacing and a finite volume of the box (torus). On a torus particles may propagate "around the world". Lüscher showed [88] that if the lattice is not too small, then corrections due to the finite extent L of the lattice will be exponentially suppressed by a factor $e^{-\alpha m_\pi L}$. Another consequence of the finite volume is that the minimal spatial momentum (distinct from 0) that can live on the lattice is

$$p = \frac{2\pi}{L}. \tag{8.96}$$

Analyzing dispersion relations for light particles one often needs a better resolution, but increasing L drives up the numerical cost just as using a more fine grained lattice spacing a. In practical simulations one has to find a compromise.

8.9 Extraction of Masses

The most elementary results one can get from a lattice simulation are masses of composite particles. This is done using correlators or two-point functions, which are later evaluated to physical quantites. The transfer matrix formalism (which is nothing but an application of the Euclideanized version of the time-evolution operator $S = e^{-iHt}$) predicts for the two-point functions:

$$\langle B(t_2)\bar{B}(t_1)\rangle = \frac{1}{Z}\mathrm{Tr}\left(S^{N_t - t_2/a}\, B\, S^{(t_2-t_1)/a}\, \bar{B}\, S^{t_1/a}\right), \tag{8.97}$$

where $Z = \mathrm{Tr}\, S^{N_t}$ and $N_t = L_t/a$ is the number of sites in time direction. We use $N_s = L_s/a$ for the number of sites a spatial direction. In the continuum one usually works with the standard normalization convention

$$\langle N(\vec{p})|N(\vec{p}')\rangle = 2E_{\vec{p}}(2\pi)^3\delta^{(3)}(\vec{p}-\vec{p}'). \tag{8.98}$$

On the lattice, however, it is much easier to normalize all states to 1. Using this "lattice normalization" we insert identities $\mathbb{1} = \sum_\nu |\nu\rangle\langle\nu|$ given by the ordered set of energy eigenstates $|\nu\rangle$ of S with energies E_ν:

$$\begin{aligned}\langle B(t_2)\bar{B}(t_1)\rangle &= \frac{1}{Z}\sum_{\nu_1\cdots\nu_5}\langle\nu_5|S^{N_t-t_2/a}|\nu_1\rangle\langle\nu_1|B|\nu_2\rangle\langle\nu_2|S^{(t_2-t_1)/a}|\nu_3\rangle\langle\nu_3|\bar{B}|\nu_4\rangle\langle\nu_4|S^{t_1/a}|\nu_5\rangle \\ &= \frac{1}{Z}\sum_{\nu_1\nu_2} e^{-(L_t-(t_2-t_1))E_{\nu_1}}\langle\nu_1|B|\nu_2\rangle e^{-(t_2-t_1)E_{\nu_2}}\langle\nu_2|\bar{B}|\nu_1\rangle.\end{aligned} \tag{8.99}$$

For $L_t \to \infty$ and the energies chosen such that $E_0 = 0$ only few terms survive:

$$Z = \operatorname{Tr} S^{N_t} = \sum_\nu e^{-L_t E_\nu} \to 1, \tag{8.100}$$

$$\langle B(t_2)\bar{B}(t_1)\rangle = \sum_{\nu>0}\langle 0|B|\nu\rangle\langle\nu|\bar{B}|0\rangle e^{-(t_2-t_1)E_\nu} + \sum_{\nu>0}\langle\nu|B|0\rangle\langle 0|\bar{B}|\nu\rangle e^{-(L_t-(t_2-t_1))E_\nu}. \tag{8.101}$$

The most common procedure is to define the correlator by creating the particle at time zero and annihilating it at time t. When t is large enough, such that higher states contributions become negligible, only the ground state G remains and we obtain for of $0 \ll t \ll L_t/2$:

$$C(t) = \langle B(t)\bar{B}(0)\rangle \approx \langle 0|\, B\,|G\rangle\,\langle G|\,\bar{B}\,|0\rangle\, e^{-tE_G}. \tag{8.102}$$

This essential result allows for the extraction of masses from a lattice simulation, since the strength of the exponential decay in time is controlled by the energy.

8.10 Variational Method

We have seen how to extract ground state masses from LQCD. While ground states are rather straightforward to extract, excited hadrons are difficult to study on the lattice. The reason for this is that contributions of excited states to correlators are exponentially suppressed with increasing lattice time t. Therefore excited states disappear behind the ground state rather soon (with respect to t). Even worse, the statistical noise of states [89] is largely determined by a specific mass difference:

$$\Delta m = m_s - m_{s\bar{s}}/2. \tag{8.103}$$

Here m_s is the mass of state under inspection while $m_{s\bar{s}}$ is the minimal mass of any state that includes all constituent quarks of s and $\bar{s}$. The signal-to-noise ratio may be estimated to be

$$\frac{C(t)}{\sigma(t)} \propto \sqrt{N} e^{-\Delta m t}, \tag{8.104}$$

where N is the number of configurations. Looking at a nucleon correlation function, this is the mass of 3 pions. The signal to noise ratio would therefore behave similar to

$$\frac{C_N(t)}{\sigma_N(t)} \propto \sqrt{N} e^{-(m_N - 3m_q/2)t}. \tag{8.105}$$

Therefore excited state disappear increasingly more quickly in the noise than ground states.

The noise problem is fundamental, but fortunately it is possible to at least separate excited states from ground states and other excited states by adding independent information to the problem: We look at a set operators O_i. We calculate the cross-correlation matrices of 2-point-functions

$$C_{ij}(t) = \langle O_i(t)\bar{O}_j(0)\rangle. \tag{8.106}$$

This matrix has a Hilbert-space decomposition

$$C_{ij}(t) = \sum_n \langle 0|O_i|n\rangle\langle n|\bar{O}_j|0\rangle e^{-tE_n}. \tag{8.107}$$

It can be shown [90], that the eigenvalues of the generalized eigenvalue problem

$$C(t)\psi_k(t) = \lambda_k(t,t_0)C(t_0)\psi_k(t) \tag{8.108}$$

behave as

$$\lambda_k(t,t_0) = c_k e^{-(t-t_0)E_k}\left(1+\mathcal{O}\left(e^{-(t-t_0)\Delta E_k}\right)\right), \tag{8.109}$$

while $\lambda_k(t_0,t_0)=1$. At fixed t_0, ΔE_k is given by

$$\Delta E_k = \min\{E_m - E_n | m \neq n\}, \tag{8.110}$$

while for the special case of $t \leq 2t_0$ and a basis of N correlators [91] ΔE_k is given by

$$\Delta E_k = E_{N+1} - E_n. \tag{8.111}$$

Therefore, at large time separations, each eigenvalue is dominated by a single state, since the difference $E_N + 1 - E_n$ is large. This allows for stable two parameter fits to the eigenvalues. Consequently, the largest eigenvalue decays with the mass of the ground state, the second largest eigenvalue with the mass of the first excited state, and so on.

In practice, if statistics is sufficiently high, it is safe to sort the generalized eigenvalues with respect to their magnitude (e.g. descending for $t > t_0$ and ascending for $t < t_0$) in order to obtain a proper relationship of eigenvalues and excited states. The related eigenvectors also bear great value: The time independence of the eigenvector components tells on the purity of the state, while the composition of the eigenvector can serve as a "fingerprint" for the physical state: The name of the "Variational Method" stems from the simplest variant of the eigenvalue problem for correlators:

$$C(t)\psi'_k(t) = \lambda_k(t)\psi'_k(t) \tag{8.112}$$

Here, the diagonalization "searches" for the direction in correlator-space that behaves linearly, thereby extracting the best combination of correlators for a physical state, since linear behavior is obtained once (with respect to the lattice time t) excited states play no role any longer for such a direction. So the eigenvectors ψ'_k can be seen as an optimal (within the given set) combination of operators ("sources") for the states $|k\rangle$.

8.11 Extraction of Matrix Elements

In section 8.9 we have seen how to extract masses of composite particles using the transfer matrix formalism. Now we are interested in matrix elements of certain operators $\mathcal{O}$ in the background of some particle N:

$$\langle N|\mathcal{O}|N\rangle. \tag{8.113}$$

Unfortunately we do not have to full wavefunctions $|N\rangle$ at hand, but by dividing three-point functions by two-point functions we can construct interesting quantities. In section 8.9 we obtained formula (8.102) for the two-point-functions:

$$C(t) \equiv \langle B(t)\bar{B}(0)\rangle \approx \langle 0|B|N\rangle\langle N|\bar{B}|0\rangle e^{-tE_N}. \tag{8.114}$$

Similarly we define three-point-functions and obtain using the transfer matrix formalism:

$$\begin{aligned} C^{\mathcal{O}}(\tau,t) \equiv & \langle B(t)\mathcal{O}(\tau)\bar{B}(0)\rangle = \frac{1}{Z}\mathrm{Tr}\left(S^{N_t-t/a}\, B\, S^{(t-\tau)/a}\, \mathcal{O}(\tau)\, S^{\tau/a}\, \bar{B}\right) \\ &= \sum_{\nu_1\cdots\nu_6} \langle\nu_1|S^{N_t-t/a}|\nu_2\rangle\langle\nu_2|B|\nu_3\rangle\langle\nu_3|S^{(t-\tau)/a}|\nu_4\rangle\langle\nu_4|\mathcal{O}|\nu_5\rangle\langle\nu_5|S^{\tau/a}|\nu_6\rangle\langle\nu_6|\bar{B}|\nu_1\rangle \\ &= \sum_{\nu_1\nu_2\nu_3} e^{-(L_t-t)E_{\nu_1}}\langle\nu_1|B|\nu_2\rangle e^{-(t-\tau)E_{\nu_2}}\langle\nu_2|\mathcal{O}|\nu_3\rangle e^{-\tau E_{\nu_3}}\langle\nu_3|\bar{B}|\nu_1\rangle \\ &\approx \langle 0|B|N\rangle\langle N|\mathcal{O}|N\rangle\langle N|\bar{B}|0\rangle e^{-tE_N}. \end{aligned} \tag{8.115}$$

We can use this formula in order to obtain matrix elements when $t \gg \tau$ and $\tau \gg 0$: By dividing three- by two-point functions we can get access to the quantity of interest:

$$R^{\mathcal{O}}(\tau, t) \equiv \frac{C^{\mathcal{O}}(\tau, t)}{C(t)} = \frac{\langle 0|B|N\rangle\langle N|\mathcal{O}|N\rangle\langle N|\bar{B}|0\rangle e^{-tE_N}}{\langle 0|B|N\rangle\langle N|\bar{B}|0\rangle e^{-tE_N}} = \langle N|\mathcal{O}|N\rangle, \tag{8.116}$$

The independence on τ should thus lead to a plateau-type behavior between time 0 and t.

8.12 Variational Method for Matrix Elements

If we know the optimal composition of correlators it is possible to improve the well known procedure of formula 8.116 to extract matrix elements from correlators by the calculation of ratios. We determine the 3-point correlation matrix

$$C^{\mathcal{O}}_{ij}(\tau, t) = \langle B_i(t)\mathcal{O}(\tau)\bar{B}_j(0)\rangle \tag{8.117}$$

and compute, see the simplest variant from [92],

$$R_k(\tau, t) = \frac{\psi_k(t)^\dagger C^{\mathcal{O}}(\tau, t)\psi_k(t)}{\psi_k(t)^\dagger C(t)\psi_k(t)} = \frac{c_k\langle k|\mathcal{O}|k\rangle e^{-tE_k}}{c_k e^{-tE_k}} = \langle k|\mathcal{O}|k\rangle \tag{8.118}$$

to extract the expectation value of insertions $\mathcal{O}$ within states $|k\rangle$. The independence on τ should thus lead to a plateau-type behavior between time zero and t. In the notation of continuum normalization we obtain

$$\langle k|\mathcal{O}|k\rangle_{\text{continuum}} = 2E_k R^{\mathcal{O}}_k(\tau, t). \tag{8.119}$$

8.13 Lattice Operators for Vector and Axial Charge

In section 7.3 we have introduced continuum operators that may be used to extract the vector and axial charge of zero momentum nucleon states:

$$\langle p|V_0^{(u-d)}|p\rangle = 2g_V p_0 \tag{8.120}$$

$$\langle p, s|A_i^{(u-d)}|p, s\rangle = 2g_A s_i \tag{8.121}$$

For states in lattice normalization $\langle p, s|p, s\rangle = 1$ we obtain:

$$\langle p|V_0^{(u-d)}|p\rangle = g_V \tag{8.122}$$

$$\langle p, s|A_i^{(u-d)}|p, s\rangle = g_A s_i \tag{8.123}$$

where $s^2 = -1$.

8.14 Baryon Operators

There are plenty of baryons. However, the most interesting of these are clearly the nucleons, as the majority of the (visible) matter in the universe is made up just from these. There are two nucleons: the proton and the neutron. Their quantum numbers and physical masses are given by

baryon	quarks	$I(J^P)$	m/MeV
proton	uud	$\frac{1}{2}(\frac{1}{2}^+)$	938.27203(8)
neutron	udd	$\frac{1}{2}(\frac{1}{2}^+)$	939.56536(8)

The values have been taken from [93]. On the lattice we calculate with mass degenerate quarks u and d, such that the nucleons also become degenerate. There are a number of nucleon interpolators that have some overlap with the full nucleon. In this work we focus on

$$\begin{aligned} N_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{-i\vec{p}\vec{x}} \epsilon^{abc} \Gamma_1 u_a(\vec{x},t) \left(u^b(\vec{x},t)^T \Gamma_2 d^c(\vec{x},t) - d^b(\vec{x},t)^T \Gamma_2 u^c(\vec{x},t) \right) \\ \bar{N}_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{+i\vec{p}\vec{x}} \epsilon^{abc} \bar{u}_a(\vec{x},t) \Gamma_1^\dagger \left(\bar{d}^b(\vec{x},t) \Gamma_2^\dagger \bar{u}^c(\vec{x},t)^T - \bar{u}^b(\vec{x},t) \Gamma_2^\dagger \bar{d}^c(\vec{x},t)^T \right) \end{aligned} \tag{8.124}$$

We use them in three different types. Table 8.1 shows the relevant Dirac structures for nucleon-type interpolators. All three types couple only to the spin 1/2 states, but they have different diquark structures: Interpolators χ_1 and χ_3 contain a scalar diquark (which is sometimes also

type	Γ_1	Γ_2
χ_1	$\mathbb{1}$	$C\gamma_5$
χ_2	γ_5	C
χ_3	$i\mathbb{1}$	$C\gamma_t\gamma_5$

Table 8.1: Different variants of nucleon interpolators used. These are valid for N, Σ and Ξ baryons. Here, C denotes charge configuration.

labeled "good" diquark as there is a pronounced attractive interaction for scalar diquarks), while interpolator χ_2 contains a pseudo-scalar diquark. Interpolators χ_1 and χ_3 couple mainly to the nucleon ground state, interpolator χ_2 predominantly to excited states of both positive and negative parity [94, 95]. Further, in order to improve the overlap of our operator with real spatially extended states, we choose Jacobi Smearing [96, 97]. Exchanging light quarks for strange quarks we obtain interpolators for the Σ baryon

$$\begin{aligned} \Sigma_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{-i\vec{p}\vec{x}} \epsilon^{abc} \Gamma_1 u_a(\vec{x},t) \left(u^b(\vec{x},t)^T \Gamma_2 s^c(\vec{x},t) - s^b(\vec{x},t)^T \Gamma_2 u^c(\vec{x},t) \right) \\ \bar{\Sigma}_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{+i\vec{p}\vec{x}} \epsilon^{abc} \bar{u}_a(\vec{x},t) \Gamma_1^\dagger \left(\bar{s}^b(\vec{x},t) \Gamma_2^\dagger \bar{u}^c(\vec{x},t)^T - \bar{u}^b(\vec{x},t) \Gamma_2^\dagger \bar{s}^c(\vec{x},t)^T \right) \end{aligned} \tag{8.125}$$

and the Ξ baryon:

$$\begin{aligned} \Xi_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{-i\vec{p}\vec{x}} \epsilon^{abc} \Gamma_1 s_a(\vec{x},t) \left(s^b(\vec{x},t)^T \Gamma_2 u^c(\vec{x},t) - u^b(\vec{x},t)^T \Gamma_2 s^c(\vec{x},t) \right) \\ \bar{\Xi}_\alpha(t,\vec{p}) &= \sum_{\vec{x}} e^{+i\vec{p}\vec{x}} \epsilon^{abc} \bar{s}_a(\vec{x},t) \Gamma_1^\dagger \left(\bar{u}^b(\vec{x},t) \Gamma_2^\dagger \bar{s}^c(\vec{x},t)^T - \bar{s}^b(\vec{x},t) \Gamma_2^\dagger \bar{u}^c(\vec{x},t)^T \right) \end{aligned} \tag{8.126}$$

8.15 Nucleon Correlator Calculation

We now calculate the two-point-function of a nucleon with momentum $\vec{p}$, parity $\pm$, created at time 0 with quark smearings $\bar{I} = (\bar{I}_1, \bar{I}_2, \bar{I}_3)$ and annihilated at time t with quark smearings $I = (I_1, I_2, I_3)$:

$$C^{I,\bar{I}}_{\alpha\bar{\alpha}}(t,\vec{p}) = \langle N^I_\alpha(t,\vec{p}) \bar{N}^{\bar{I}}_{\bar{\alpha}}(0,\vec{p}) \rangle \tag{8.127}$$

Plugging in formulas 8.124 and appropriate smearing operators, the Wick contractions yield:

$$
\begin{aligned}
&C^{I,\bar{I}}_{\alpha\bar{\alpha}}(t,\vec{p}) = \sum_{\vec{x}} e^{i\vec{p}\cdot\vec{x}} \epsilon_{abc}\epsilon_{\bar{a}\bar{b}\bar{c}} (\Gamma_1^I)_{\alpha\alpha'} (\Gamma_1^{\bar{I}})_{\bar{\alpha}\bar{\alpha}'} (\Gamma_2^I)_{\beta'\gamma'} (\Gamma_2^{\bar{I}})_{\bar{\beta}'\bar{\gamma}'} \\
&\Big[\Big(S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big) S_d^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'} \\
&+\Big(S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}\Big) S_d^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'} \\
&+\Big(S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}\Big) S_d^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'} \\
&+\Big(S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}\Big) S_d^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big].
\end{aligned}
\tag{8.128}
$$

Here, propagators with source smearing $\bar{I}$ and sink smearing I are defined by

$$
S_q^{I,\bar{I}}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha\bar{\alpha}} = \sum_{\vec{y},\vec{\bar{y}}} (W_t^I(\vec{y},\vec{x})^{a'a})^\star S_q(\vec{y},t|\vec{\bar{y}},0)^{a'\bar{a}'}_{\alpha\bar{\alpha}} W_0^{\bar{I}}(\vec{\bar{y}},\vec{0})^{\bar{a}'\bar{a}} \tag{8.129}
$$

8.16 Nucleon Insertion-Correlator Calculation

We now calculate the three-point-function of a nucleon with momentum $\vec{p}$, parity $\pm$, created at time 0 with quark smearings $\bar{I} = (\bar{I}_1, \bar{I}_2, \bar{I}_3)$, measured by an flavor neutral current operator $J_\Gamma^{(q)}$ at time τ

$$
J_\Gamma^{(q)} = \sum_{\vec{x}} \bar{q}^{\bar{d}}_{\bar{\delta}'}(\vec{x},\tau) \Gamma^{\bar{d}d}_{\bar{\delta}'\delta'} q^d_{\delta'}(\vec{x},\tau) \tag{8.130}
$$

and annihilated at time t with quark smearings $I = (I_1, I_2, I_3)$:

$$
C^{I,\bar{I},J_\Gamma^{(q)}}_{\alpha\bar{\alpha}}(\tau,t,\vec{p}) = \langle N^I_\alpha(t,\vec{p}) J^\Gamma_q(\tau) \bar{N}^{\bar{I}}_{\bar{\alpha}}(0,\vec{p})\rangle. \tag{8.131}
$$

The aim of this thesis is to calculate non-singlet matrix elements of current operators. For the nucleon operators it turns out, that any disconnected terms appearing in the three point functions cancel each other upon subtraction of up and down quark three point functions. For this reason we neglect disconnected contributions to simplify the calculations. Working out the Wick contractions, we obtain for a down-quark insertion

$$
\begin{aligned}
&C^{I,\bar{I},J_\Gamma^{(d)}}_{\alpha\bar{\alpha}}(\tau,t,\vec{p}) = \sum_{\vec{x},\vec{y}} e^{i\vec{p}\cdot\vec{x}} \Gamma^{\bar{d}d}_{\bar{\delta}'\delta'} \epsilon_{abc}\epsilon_{\bar{a}\bar{b}\bar{c}} (\Gamma_1^I)_{\alpha\alpha'} (\Gamma_1^{\bar{I}})_{\bar{\alpha}\bar{\alpha}'} (\Gamma_2^I)_{\beta'\gamma'} (\Gamma_2^{\bar{I}})_{\bar{\beta}'\bar{\gamma}'} \\
&\quad\Big[\Big(+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} \\
&\qquad -S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big) S_d^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'} S_d^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&\quad+\Big(+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} \\
&\qquad -S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}\Big) S_d^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'} S_d^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'} \\
&\quad+\Big(+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} \\
&\qquad -S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}\Big) S_d^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'} S_d^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&\quad+\Big(+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} \\
&\qquad -S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'} S_u^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}\Big) S_d^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'} S_d^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'}\Big]
\end{aligned}
\tag{8.132}
$$

and for an up-quark insertion we obtain

$$
\begin{aligned}
C^{I,\bar{I},J_\Gamma^{(u)}}_{\alpha\bar{\alpha}}(\tau,t,\vec{p}) = \sum_{\vec{x},\vec{y}} e^{i\vec{p}\cdot\vec{x}}\Gamma^{\bar{d}d}_{\bar{\delta}'\delta'}\epsilon_{abc}\epsilon_{\bar{a}\bar{b}\bar{c}}(\Gamma_1^I)_{\alpha\alpha'}(\Gamma_1^{\bar{I}})_{\bar{\alpha}\bar{\alpha}'}(\Gamma_2^I)_{\beta'\gamma'}(\Gamma_2^{\bar{I}})_{\bar{\beta}'\bar{\gamma}'} \\
\Big[\Big(&+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_u^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'} \\
&-S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'}S_u^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'} \\
&+S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'} \\
&-S_u^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'}\Big)\,S_d^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'} \\
+\Big(&+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_u^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'} \\
&-S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_2,\bar{0}}(\vec{x},t|\vec{y},\tau)^{b\bar{d}}_{\beta'\bar{\delta}'}S_u^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&+S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&-S_u^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'}\Big)\,S_d^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'} \\
+\Big(&+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_u^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'} \\
&-S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'}S_u^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'} \\
&+S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_2}(\vec{y},\tau|\vec{0},0)^{d\bar{b}}_{\delta'\bar{\beta}'} \\
&-S_u^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'}\Big)\,S_d^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'} \\
+\Big(&+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_u^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'} \\
&-S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_3,\bar{0}}(\vec{x},t|\vec{y},\tau)^{c\bar{d}}_{\gamma'\bar{\delta}'}S_u^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&+S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_3}(\vec{y},\tau|\vec{0},0)^{d\bar{c}}_{\delta'\bar{\gamma}'} \\
&-S_u^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}S_u^{I_1,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{d}}_{\alpha'\bar{\delta}'}S_u^{0,\bar{I}_1}(\vec{y},\tau|\vec{0},0)^{d\bar{a}}_{\delta'\bar{\alpha}'}\Big)\,S_d^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big]
\end{aligned}
\tag{8.133}
$$

8.17 Sequential Propagators

We define single-quark sequential propagators of flavor neutral current operators J_f^Γ by

$$
S^{I,\bar{I}}_{f,\Gamma,\tau}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha\bar{\alpha}} = \sum_{\vec{y}} S_f^{I,\bar{0}}(\vec{x},t|\vec{y},\tau)^{a\bar{b}}_{\alpha\bar{\beta}}\Gamma^{\bar{b}b}_{\bar{\beta}\beta}S_f^{0,\bar{I}}(\vec{y},\tau|\vec{0},0)^{b\bar{a}}_{\beta\bar{\alpha}} \tag{8.134}
$$

$$
= \sum_{\vec{z}}(W_t^I(\vec{z},\vec{x})^{a'a})^\star \sum_{\vec{y},t'} S_f(\vec{z},t|\vec{y},t')^{a'\bar{b}}_{\alpha\bar{\beta}}\left(\Gamma^{\bar{b}b}_{\bar{\beta}\beta}\delta(t'-\tau)S_f^{0,\bar{I}}(\vec{y},t'|\vec{0},0)^{b\bar{a}}_{\beta\bar{\alpha}}\right) \tag{8.135}
$$

In short (i.e., full matrix notation) we have

$$
S^{I,\bar{I}}_{f,\Gamma,\tau} = W_t^I D_f^{-1} V^{\bar{I}}_{f,\Gamma,\tau}. \tag{8.136}
$$

This means, we need to invert the source $V^{\bar{I}}_{f,\Gamma,\tau}$ and perform a sink smearing W_t^I on the result in order to obtain $S^{I,\bar{I}}_{f,\Gamma,\tau}$.

When we rewrite formulas 8.132 and 8.133 with the aid of sequential propagators, we obtain

$$
\begin{aligned}
C^{I,\bar{I},J_d^\Gamma}_{\alpha\bar{\alpha}}(\tau,t,\vec{p}) = \sum_{\vec{x}} e^{i\vec{p}\cdot\vec{x}}\epsilon_{abc}\epsilon_{\bar{a}\bar{b}\bar{c}}(\Gamma_1^I)_{\alpha\alpha'}(\Gamma_1^{\bar{I}})_{\bar{\alpha}\bar{\alpha}'}(\Gamma_2^I)_{\beta'\gamma'}(\Gamma_2^{\bar{I}})_{\bar{\beta}'\bar{\gamma}'} \\
\Big[\Big(S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big)\,S^{I_3,\bar{I}_3}_{d,\Gamma,\tau}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'} \\
+\Big(S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}\Big)\,S^{I_3,\bar{I}_2}_{d,\Gamma,\tau}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}
\end{aligned}
$$

$$+\Big(S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}\Big)\, S_{d,\Gamma,\tau}^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}$$
$$+\Big(S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_u^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}\Big)\, S_{d,\Gamma,\tau}^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big] \tag{8.137}$$

and

$$C^{I,\bar{I},J_u^\Gamma}_{\alpha\bar{\alpha}}(\tau,t,\vec{p}) = \sum_{\vec{x},\vec{y}} e^{i\vec{p}\cdot\vec{x}}\Gamma^{\bar{d}d}_{\bar{\delta}'\delta'}\epsilon_{abc}\epsilon_{\bar{a}\bar{b}\bar{c}}(\Gamma_1^I)_{\alpha\alpha'}(\Gamma_1^{\bar{I}})_{\bar{\alpha}\bar{\alpha}'}(\Gamma_2^I)_{\beta'\gamma'}(\Gamma_2^{\bar{I}})_{\bar{\beta}'\bar{\gamma}'}$$
$$\Big[\Big(+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_{u,\Gamma,\tau}^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}$$
$$+ S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} - S_u^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}\Big)\, S_d^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}$$
$$+\Big(+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_{u,\Gamma,\tau}^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}$$
$$+ S_u^{I_2,\bar{I}_1}(\vec{x},t|\vec{0},0)^{b\bar{a}}_{\beta'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} - S_u^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}\Big)\, S_d^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}$$
$$+\Big(+S_u^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'}S_{u,\Gamma,\tau}^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}$$
$$+ S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_2}(\vec{x},t|\vec{0},0)^{a\bar{b}}_{\alpha'\bar{\beta}'} - S_u^{I_3,\bar{I}_2}(\vec{x},t|\vec{0},0)^{c\bar{b}}_{\gamma'\bar{\beta}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}\Big)\, S_d^{I_2,\bar{I}_3}(\vec{x},t|\vec{0},0)^{b\bar{c}}_{\beta'\bar{\gamma}'}$$
$$+\Big(+S_u^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'}S_{u,\Gamma,\tau}^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'} - S_u^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}$$
$$+ S_u^{I_3,\bar{I}_1}(\vec{x},t|\vec{0},0)^{c\bar{a}}_{\gamma'\bar{\alpha}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_3}(\vec{x},t|\vec{0},0)^{a\bar{c}}_{\alpha'\bar{\gamma}'} - S_u^{I_3,\bar{I}_3}(\vec{x},t|\vec{0},0)^{c\bar{c}}_{\gamma'\bar{\gamma}'}S_{u,\Gamma,\tau}^{I_1,\bar{I}_1}(\vec{x},t|\vec{0},0)^{a\bar{a}}_{\alpha'\bar{\alpha}'}\Big)\, S_d^{I_2,\bar{I}_2}(\vec{x},t|\vec{0},0)^{b\bar{b}}_{\beta'\bar{\beta}'}\Big] \tag{8.138}$$

Comparing these two formulas with the nucleon correlator formula 8.128, we see that we obtained the same contraction structure of quarks in both cases: Each term in the nucleon correlator is a product of several quark propagators. The corresponding three-point function for flavor f contains the same product for each flavor f propagator separately replaced by its sequential version. For example, we obtain twice the structure of the nucleon correlator for the up quark insertion, since there are two up quarks in the nucleon. This is a nice check to see that we did things right and may be visualized easily: Every two point function is a sum of Feynman diagrams. We obtain the corresponding three-point function, we replace each diagram by a sum of all possible diagrams to insert a single cross in a propagator of the desired flavor. At this point, one can get convinced without explicit calculation, that this type of transformation can be used as a general rule for any two-point function under inspection, provided we ignore disconnected contributions.

8.18 Spinor Projections

We have determined explicit formulas for two-point-functions and three-point functions $C_{\alpha\bar{\alpha}}$ which have two open spinor indices and give propagation amplitudes of spinor index $\bar{\alpha}$ to α. In order to evaluate matrix elements like

$$\langle p|V_0^{(u-d)}|p\rangle = g_V \tag{8.139}$$
$$\langle p,s|A_i^{(u-d)}|p,s\rangle = g_A s_i \tag{8.140}$$

we need to use appropriate projection matrices P to extract amplitudes for states of specific quantum numbers of parity and spin by

$$C_P = \frac{1}{4}\mathrm{Tr}\,(PC)\,. \tag{8.141}$$

For example, in Minkowski space, there is the parity projector and the spin polarization projector

$$P_\pm = \frac{1+\gamma_t}{2}, \quad P(s) = \frac{1+\gamma_5 \not{s}}{2} \tag{8.142}$$

If we use the method from section 8.11, we divide out corresponding propagation amplitudes to access matrix elements. Hence we calculate

$$R\left(P, \mathcal{O}, \tau, t, \vec{p}\right) \equiv \frac{C_P^{\mathcal{O}}(\tau, t, \vec{p})}{C_P(t, \vec{p})} = \langle N(\vec{p}) | \mathcal{O} | N(\vec{p}) \rangle, \tag{8.143}$$

to access matrix elements of $\mathcal{O}$ for the state N with the quantum numbers that P projects out.

$$\langle p, s, \pm | V_0^{(u-d)} | p, s, \pm \rangle = g_V^\pm = R\left(P^\pm, V_0^{(u-d)}, \tau, t, \vec{0}\right) \tag{8.144}$$

$$\langle p, s, \pm | A_i^{(u-d)} | p, s, \pm \rangle = g_A^\pm s_i = R\left(P^\pm P(s), A_i^{(u-d)}, \tau, t, \vec{0}\right) \tag{8.145}$$

For the axial charge, we know that the matrix element is proportional to s, so we take the difference of positive and negative polarization to reduce some noise by

$$2g_A^\pm = R(P^\pm P(\hat{e}_i)) - R(P^\pm P(-\hat{e}_i)) \tag{8.146}$$

$$= \frac{\operatorname{Tr} P^\pm P(\hat{e}_i) C^{\mathcal{O}}}{\operatorname{Tr} P^\pm P(\hat{e}_i) C} - \frac{\operatorname{Tr} P^\pm P(-\hat{e}_i) C^{\mathcal{O}}}{\operatorname{Tr} P^\pm P(-\hat{e}_i) C} \tag{8.147}$$

$$= \frac{\operatorname{Tr} P^\pm (P(\hat{e}_i) - P(-\hat{e}_i)) C^{\mathcal{O}}}{\frac{1}{2} \operatorname{Tr} P^\pm C} \tag{8.148}$$

$$\tag{8.149}$$

Altogether we calculate

$$g_{A,V}^{\pm,\text{ren}} = Z_{A,V} g_{A,V}^\pm = \frac{\operatorname{Tr} P_\pm \Gamma^{A,V} \langle N(t) J_\mu^{A,V}(\tau) \overline{N}(0) \rangle}{\operatorname{Tr} P_\pm \langle N(t) \overline{N}(0) \rangle} \tag{8.150}$$

with $\Gamma_V = \gamma_\mu$ and $\Gamma_A = \gamma_\mu \gamma_5$.

Chapter 9

Results for Nucleons

There are 7 dynamical $16^3 \times 32$ lattices of $n_f = 2$ available. We use the Lüscher-Weisz gauge action and the CI Dirac operator. These are at various lattice spacings and quark masses.

lattice	β	am_0	a/fm	a/GeV^{-1}	am_{AWI}	m_{AWI}/MeV	N_{cfg}
A50	4.70	-0.050	0.15032(112)	0.7619(57)	0.03027(8)	39.73(10)	200
A66	4.70	-0.066	0.13487(95)	0.6835(48)	0.00589(40)	8.62(57)	200
B60	4.65	-0.060	0.14980(89)	0.7591(45)	0.02356(13)	31.04(16)	325
B70	4.65	-0.070	0.14055(104)	0.7123(53)	0.00836(23)	11.74(32)	200
C64	4.58	-0.064	0.15831(127)	0.8023(64)	0.02995(20)	37.33(25)	200
C72	4.58	-0.072	0.15051(115)	0.7627(58)	0.01728(16)	22.65(21)	200
C77	4.58	-0.077	0.14487(95)	0.7342(48)	0.01054(19)	14.35(26)	300

Table 9.1: Parameters of our dynamical CI lattices

lattice	β	am_0	a/fm	a/GeV^{-1}
A	4.70	-0.06987	0.1311(15)	0.6644(76)
B	4.65	-0.07576	0.1352(21)	0.6851(106)
C	4.58	-0.08377	0.1380(17)	0.6993(86)

Table 9.2: Parameters of our chirally extrapolated dynamical CI lattices

For their parameters, see table 9.1 and for the data from chirally extrapolated lattices, table 9.2. We use the CI coefficients listed in Appendix A.3. Lattice spacings have been calculated in [98] using the method proposed by Sommer [82, 83] where the Sommer scale was chosen to $r_0 = 0.48\,\mathrm{fm}$. Axial Ward Identity (AWI)-masses have been determined in [98].

We chose to look at observables on lattices C77, C72 and C64 in order to do that necessary chiral extrapolation. Additionally we look at observables on lattice A50 in order to have some cross-check at a different β value.

9.1 Our Parameters

In this work we are going to look into a couple of observables in LQCD. Our goal is to shed light on the axial charge of the nucleon and its excited states. To this end we need to calculate the masses of the nucleon states first.

In order to reduce some of the high frequency content of the gauge configurations, we employ one step of stout smearing [99] and three steps of hypercubic-blocking [100], then calculate two- and three-point functions on them.

We use Gaussian smearing in order to improve the overlap of our interpolators with ground states. Table 9.3 shows the values used, along with the resulting Root-Mean-Square (RMS) radii.

lattice	identifier	κ	N	r_{RMS}/a
C77	1	0.223	15	2.30(1)
C77	2	0.184	70	4.67(7)
C77	3	0.15	10	0.665(2)
C77	4	1	0	local
C72	1	0.28	7	1.604(3)
C72	2	0.1925	37	3.493(33)
C72	3	0.4	2	0.8052(6)
C64	1	0.28	7	1.603(4)
C64	2	0.1918	37	3.463(28)
C64	3	0.4	2	0.8052(8)
A50	1	0.223	15	2.30(1)
A50	2	0.184	70	4.66(8)
A50	3	0.172	400	6.98(27)

Table 9.3: Variety of Gaussian smearings employed. Errors denote standard deviations.

In order to reduce contamination from back-propagating states, we use a Dirichlet boundary condition at t_D, i.e., we modify the gauge configuration by setting all gauge links from t_D to $t_D + 1$ to zero by employing

$$U_t(t_D, x, y, z) = 0 \tag{9.1}$$

before inverting any sources. When we calculate the lattice pion correlator we can see deviations from a perfect exponential decay behavior at times $|t - t_D| < 6$. For this reason we choose $t_D = 21$ such that there is no influence from the boundary condition in the interval $t \in (0..15)$. We use a EigCG Inverter to speed up to calculation of propagators for different sources on the same configuration. For each configuration we calculate standard propagators and sequential propagators with all source smearings listed in table 9.3. We calculate sequential propagators for each $t \in \{2, 3, 4\}$ and $\Gamma \in \{\gamma_4, \gamma_3\gamma_5\}$ to allow a determination of matrix elements of J^A and J^V and hence g_V and g_A. Results for g_V are only available for lattices C64 and C72. We work with a valence quark mass equal to the sea quark mass and defer calculations for matrix elements for Ξ and Σ states to a later stage. For the matrix elements under inspection, it is sufficient to work with states of momentum $\vec{p} = \vec{0}$.

9.2 Baryon Masses

In this section we look at positive and negative parity nucleon masses. We start with an inspection of diagonal elements of the correlation matrix C_{ij} in figures 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7 and 9.8. For all plots we use the source type naming convention "$\chi - I_1 - I_2 - I_3$" for a nucleon of quark structure χ and quark smearings $I = (I_1, I_2, I_3)$. For the positive parity nucleon, interpolators with χ_1 and χ_3 heavily couple to the ground state for all available smearings. Interpolators with χ_2 couple to excited states predominantly. For the negative parity nucleon, interpolators with χ_1 and χ_2 couple to the ground state for all available smearings. Interpolators with χ_3 have a stronger coupling to excited states. For the negative parity states we observe are weaker signal than for the positive parity states.

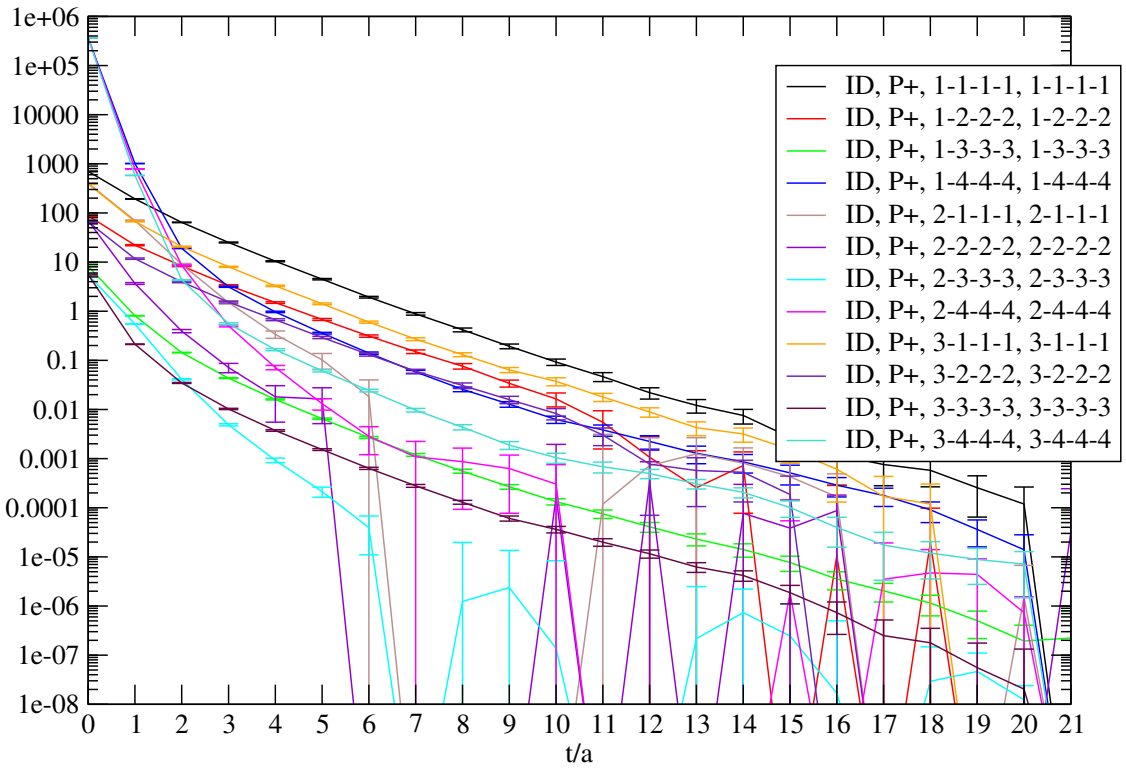


Figure 9.1: Diagonal elements of the positive parity nucleon correlation matrix for lattice C77.

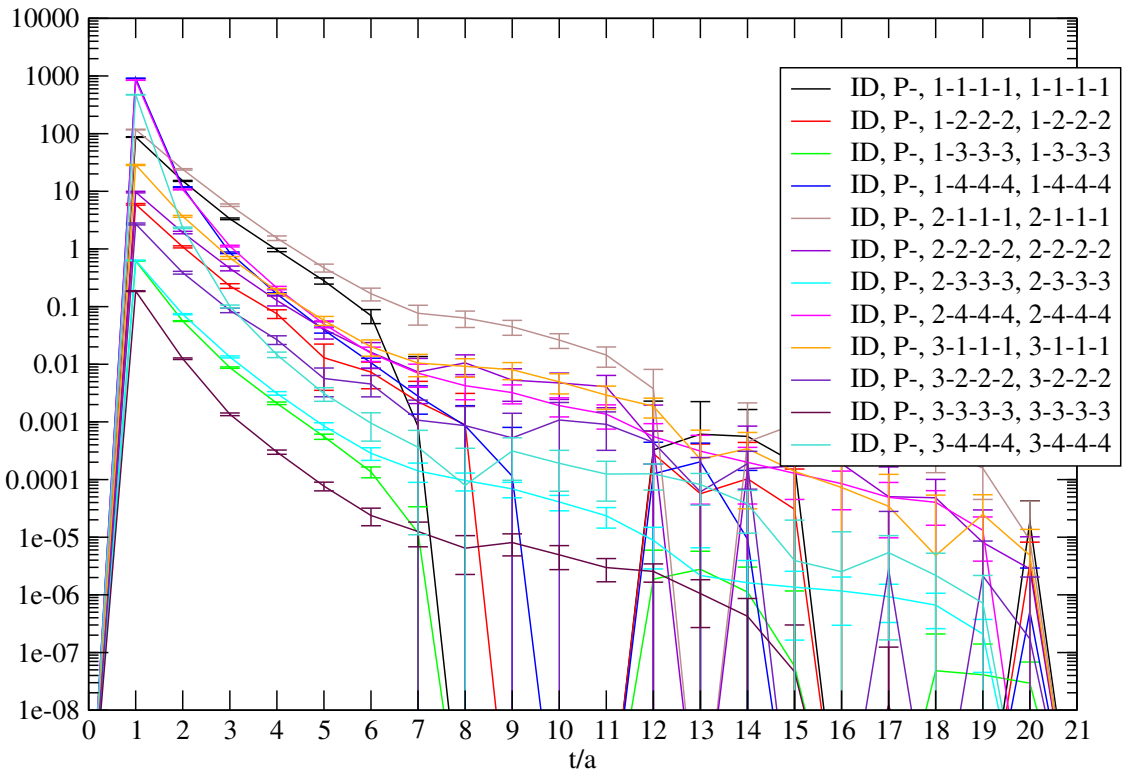


Figure 9.2: Diagonal elements of the negative parity nucleon correlation matrix for lattice C77.

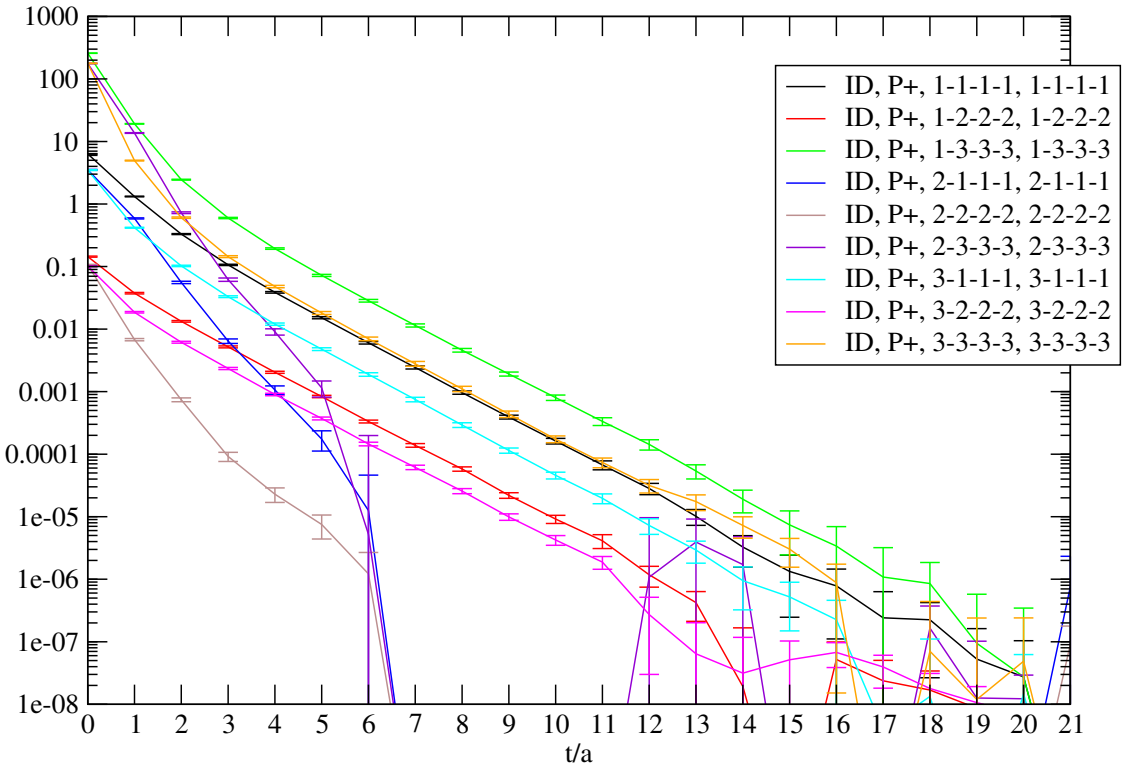


Figure 9.3: Diagonal elements of the positive parity nucleon correlation matrix for lattice C72.

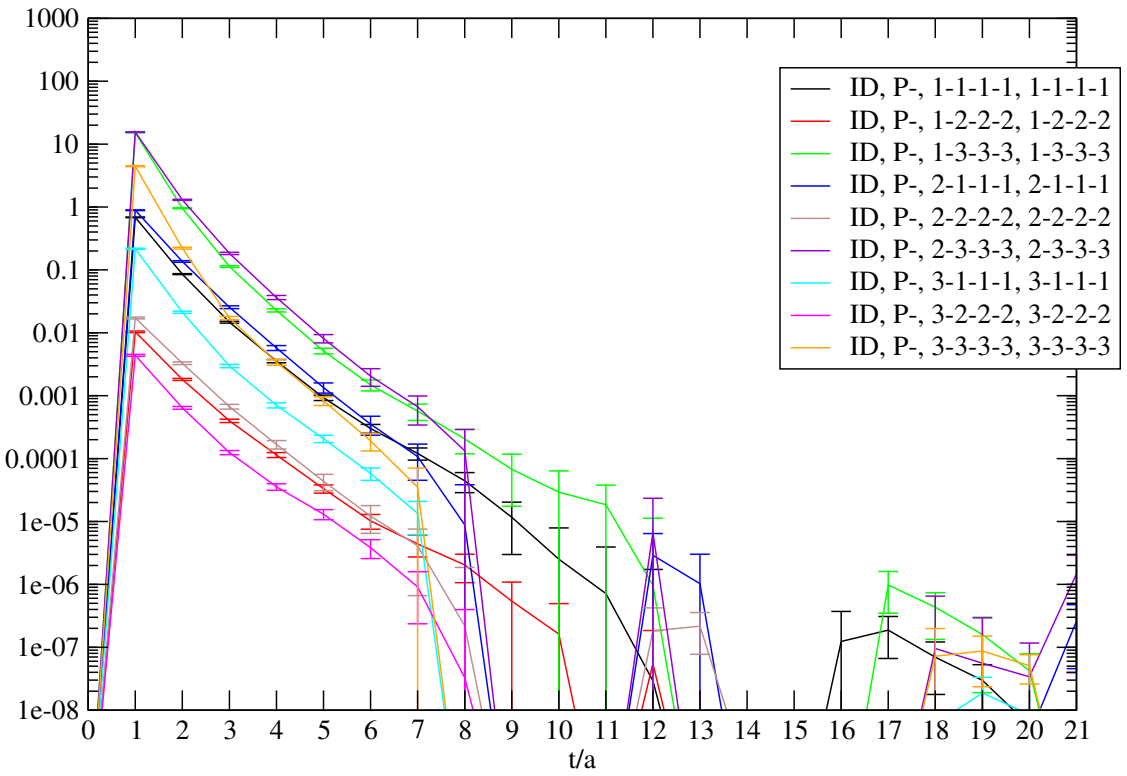


Figure 9.4: Diagonal elements of the negative parity nucleon correlation matrix for lattice C72.

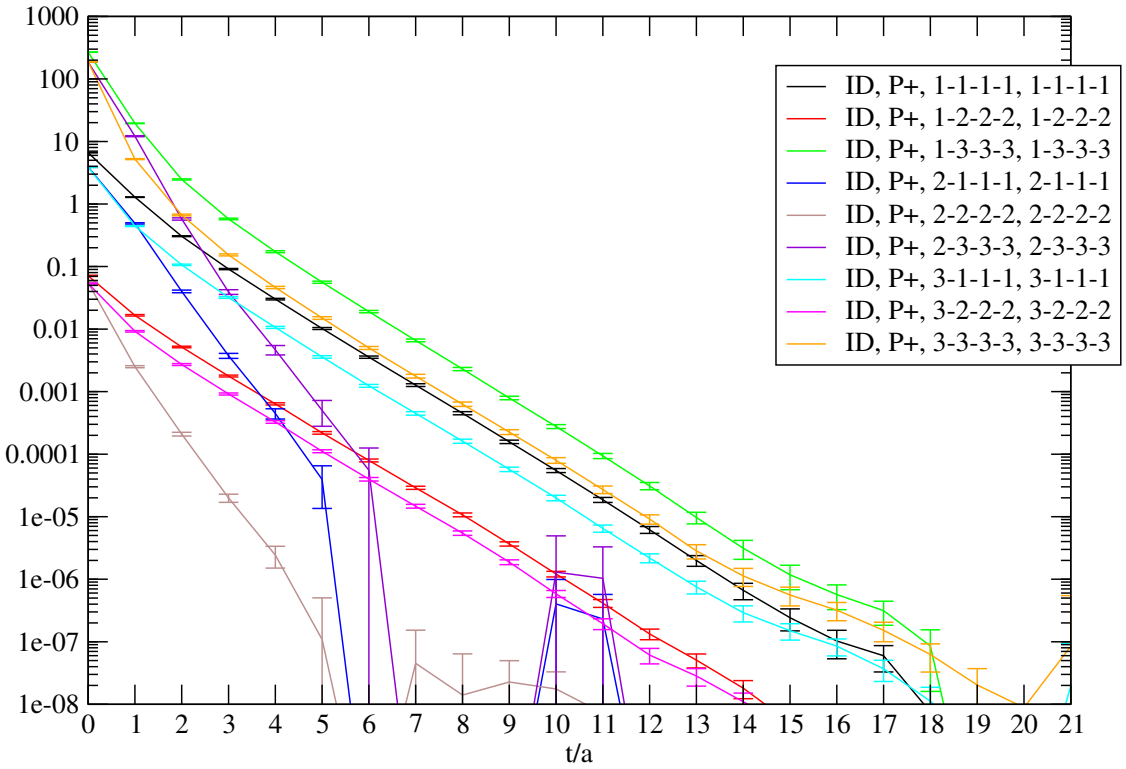


Figure 9.5: Diagonal elements of the positive parity nucleon correlation matrix for lattice C64.

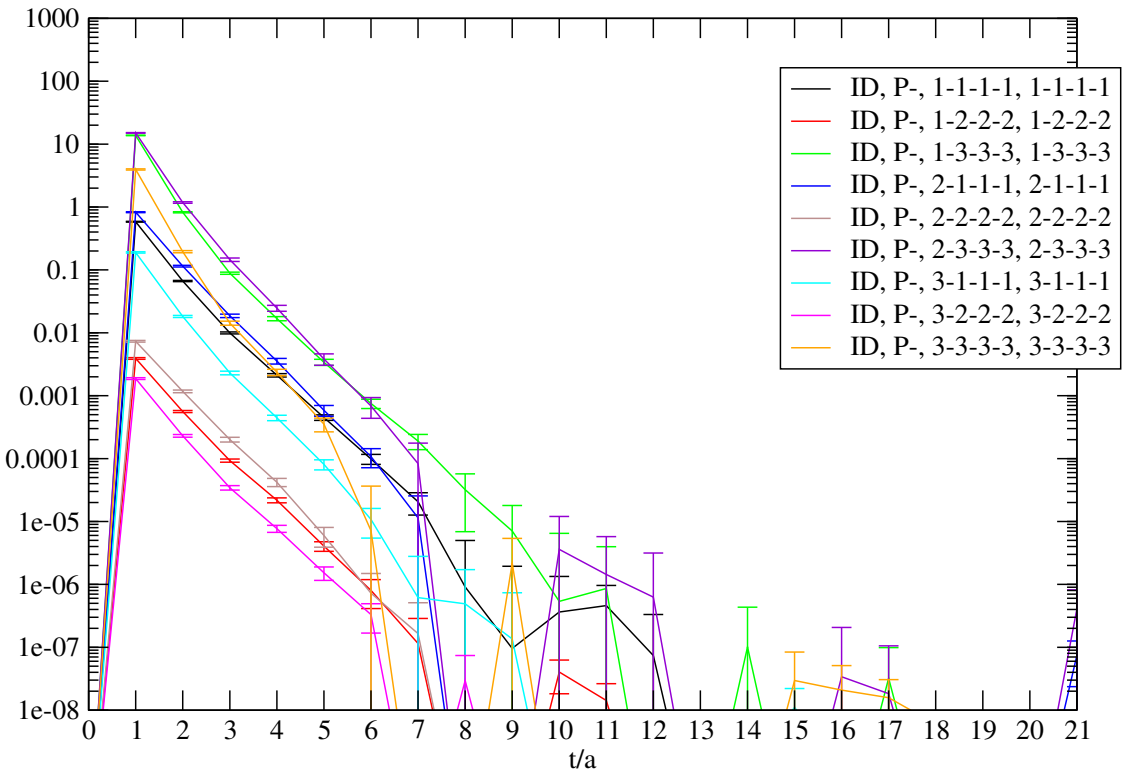


Figure 9.6: Diagonal elements of the negative parity nucleon correlation matrix for lattice C64.

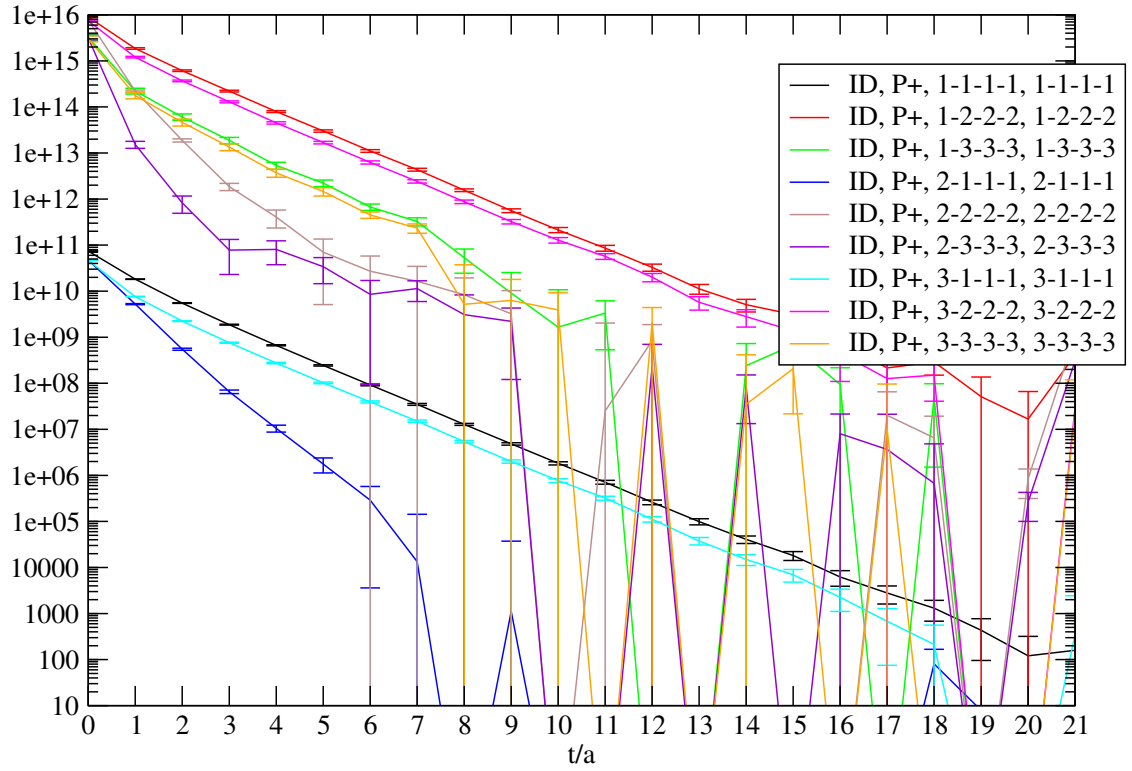


Figure 9.7: Diagonal elements of the positive parity nucleon correlation matrix for lattice A50.

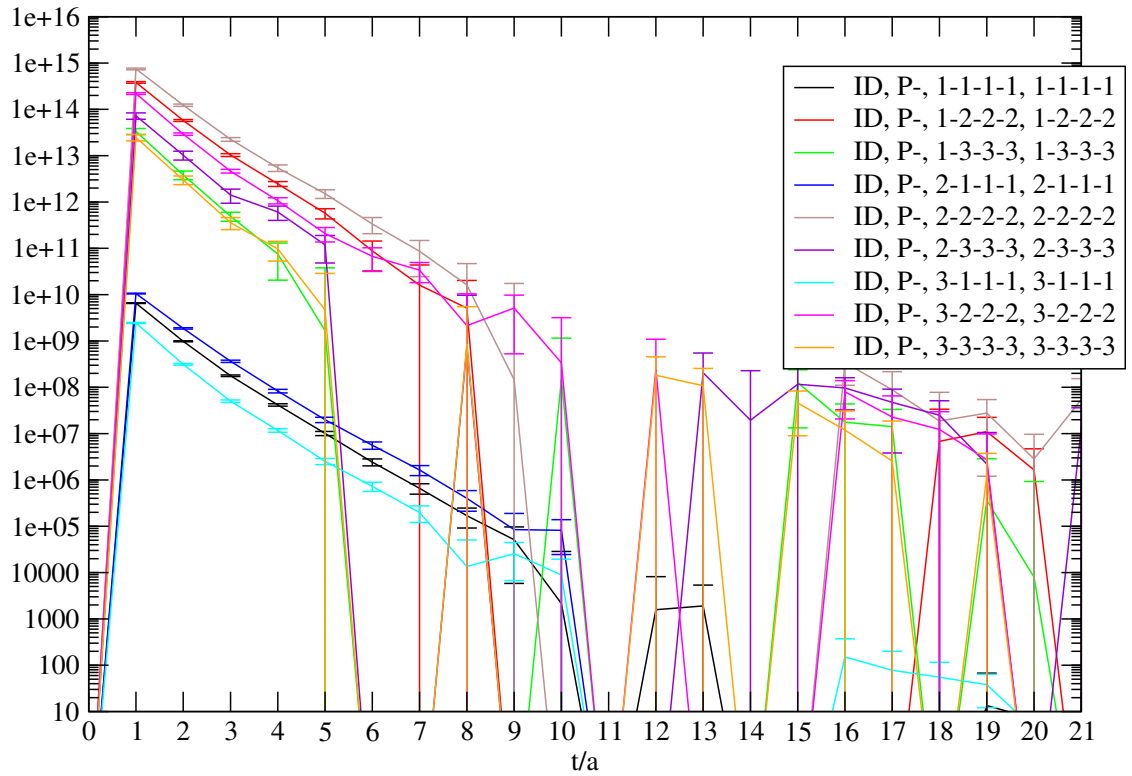
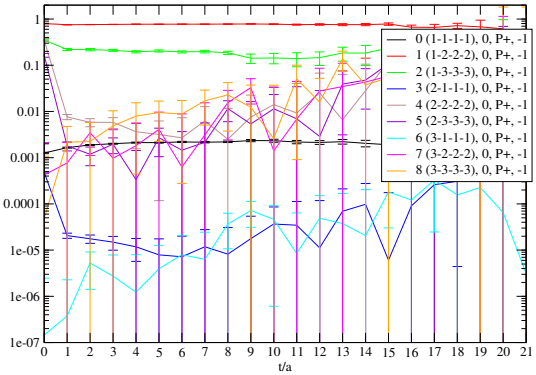
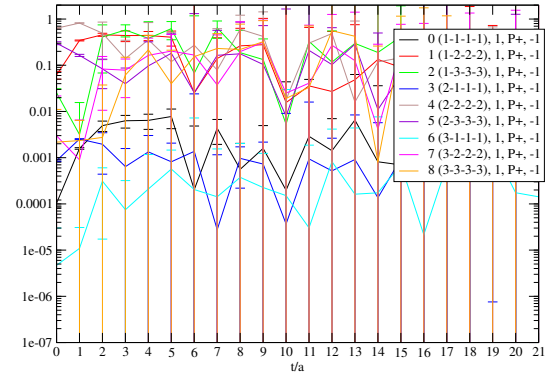


Figure 9.8: Diagonal elements of the negative parity nucleon correlation matrix for lattice A50.

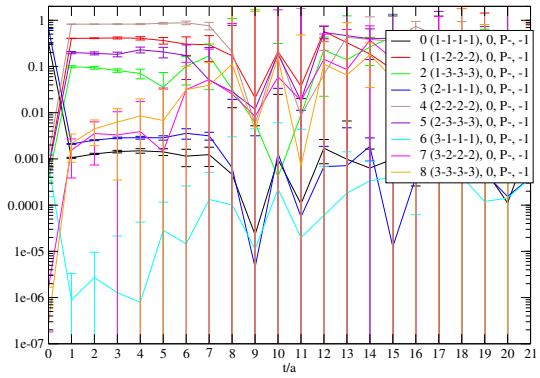
We now switch over to the full correlation matrix C_{ij} and employ the variational method in order to improve the signal and the reliability of our results. The use of a reference matrix C_0 in the variational method degrades the eigenvalue signals at the benefit of improved signal separation for excited states. We use $C_0 = C(t = 1)$ for the positive parity states since the signal is stable enough for such a tradeoff. For the negative parity states we use the variational method as well, but employ its trivial variant where $C_0 = \mathbb{1}$. This means, we diagonalize $C(t)$ for every t and plot resulting eigenvectors, eigenvalues as well as corresponding effective masses. For each of the lattices we have full matrices and results available. However from the eigenvector composition of ground and first excited states, shown exemplary for lattice A50 in figures 9.9 (a-d), we see that we can make some additional profit (with respect to the size of the statistical errors) here: The positive parity ground states couples to χ_1 only. Neither



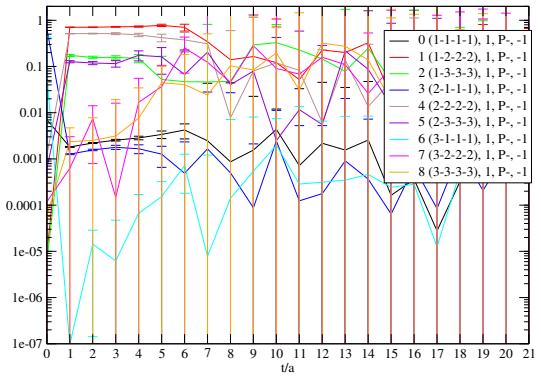
(a) the positive parity nucleon ground state for lattice A50.



(b) the positive parity nucleon first excited state for lattice A50.



(c) the negative parity nucleon ground state for lattice A50.



(d) the negative parity nucleon first excited state for lattice A50.

Figure 9.9: Eigenvector Compositions of ...

positive or negative parity states of ground or first-excited states couple to interpolators of type χ_3. Although not shown here, we saw the same behavior on all our lattices. We therefore remove this type of interpolator from all our future analysis, since we can improve the results of the variational method by removing the noise we would introduce by including χ_3.

On the lattice A50 we choose to work with 3 different amounts of source smearing: We employ the same smearing parameters as the authors of [101], but add a third even wider smearing. As evident from figure 9.7 and 9.8, the third type has large noise when compared to the other ones. We therefore remove this type of smearing from the future analysis of A50.

We continue with lattice C77. Here we try to use a more narrow smearing instead of a wider one like on lattice A50: We employ the same smearing parameters as the authors of

[101], but add a third smearing whose RMS radius is at a quarter of the size of the previous ones, as well as a fourth one that is local. We find that the relative noise of interpolators built from these smearings have the same order as the original ones and that the results from the variational method neither improve nor degrade. For the sake of clarity of our results we exclude these smearings from the analysis of C77.

This is different with lattices C72 and C64. Here we found no effects on the state masses, but plateaus of three-point over two-point functions become more stable when we include this very narrow smearing. So we do.

We plot results for this subset in figures 9.10, 9.11, 9.12, 9.13, 9.14, 9.15, 9.16 and 9.17.

For the positive parity we obtain a solid ground state signal. Its eigenvector composition indicates a contraction structure of χ_1 with a mixture of smearings radii 1 and 2. Excited states for the positive parity are extremely noisy, since we purposely used interpolators with good overlap with the ground state.

For the negative parity we obtain two almost degenerate states. Their eigenvector composition is stable up to time-slice 6, then explodes into noise. In order to see longer plateaus the number of configurations would need to be increased strongly.

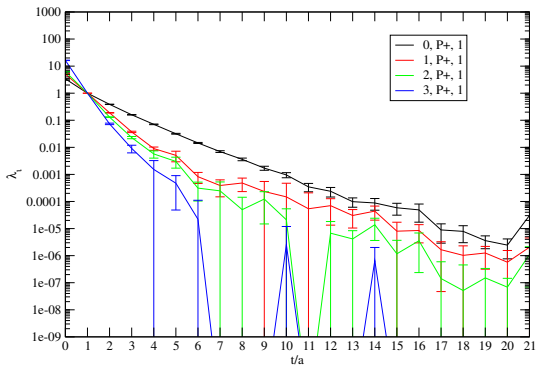
In order to extract the masses of the states we fit double exponentials

$$\lambda(t) = A_1 e^{-m_1 t} + A_2 e^{-m_2 t} \tag{9.2}$$

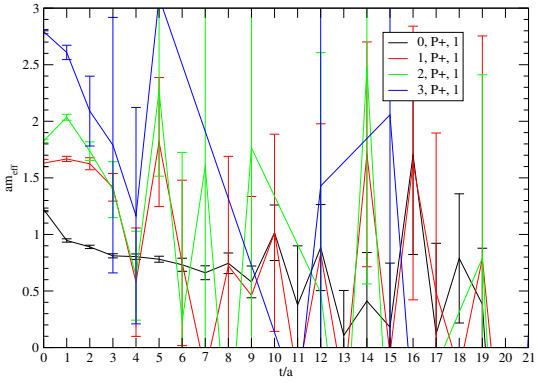
to the eigenvalues and list them, together with the fit intervals used, in table 9.4.

lattice	parity	state	fit interval	am
C77	+	1	2-11	0.691(47)
C77	+	2	2-6	1.564(48)
C77	-	1	1-5	1.07(19)
C77	-	2	1-5	1.08(14)
C72	+	1	2-11	0.881(21)
C72	+	2	2-6	1.772(20)
C72	-	1	1-5	1.451(79)
C72	-	2	1-6	1.350(63)
C64	+	1	2-11	1.033(6)
C64	+	2	2-6	1.806(18)
C64	-	1	1-5	1.65(2)
C64	-	2	1-5	1.59(2)
A50	+	1	2-11	0.974(18)
A50	+	2	2-6	1.585(35)
A50	-	1	1-7	1.19(16)
A50	-	2	1-7	1.340(85)

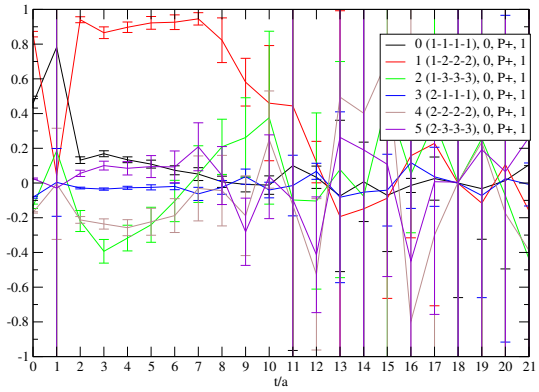
Table 9.4: Nucleon masses measured. Fit intervals include data points at their boundaries.



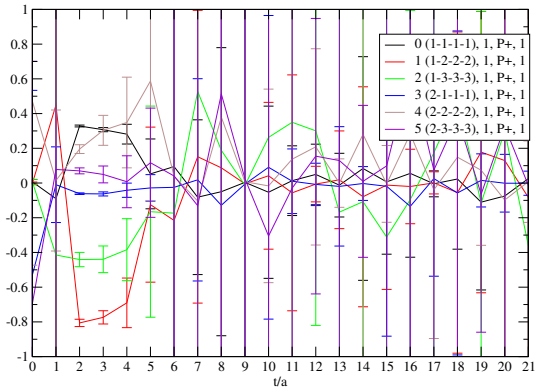
(a) Eigenvalues



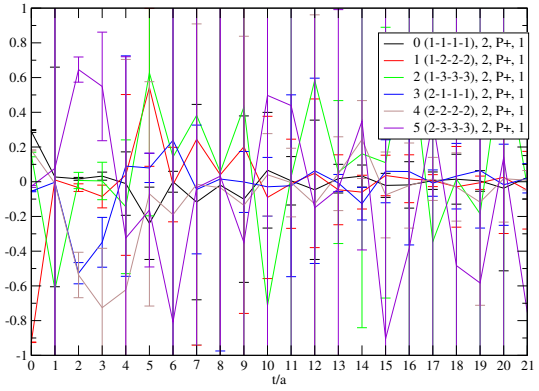
(b) Effective masses



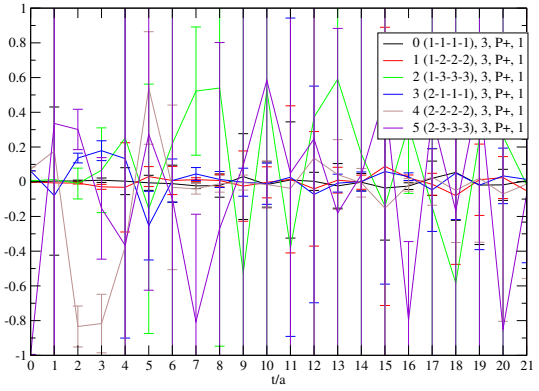
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

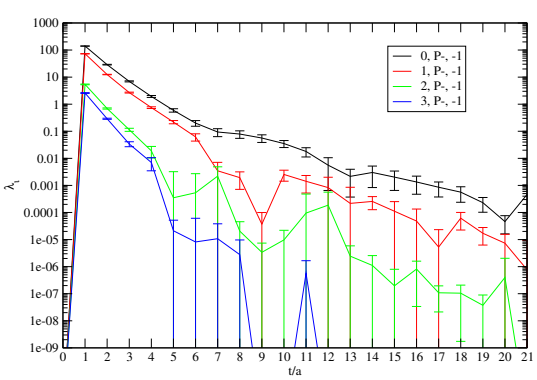


(e) Composition of eigenvector 3

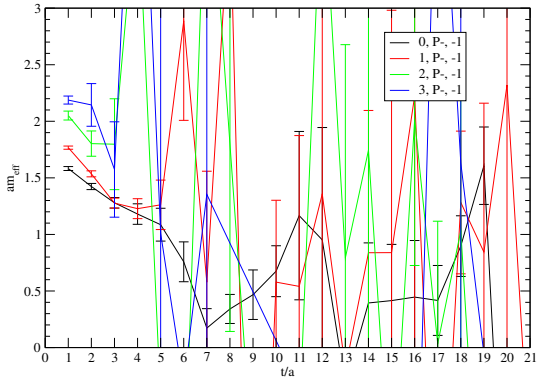


(f) Composition of eigenvector 4

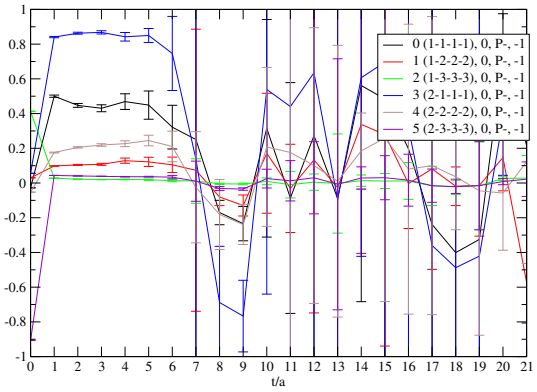
Figure 9.10: Results from the variational method for the positive parity nucleon on lattice C77.



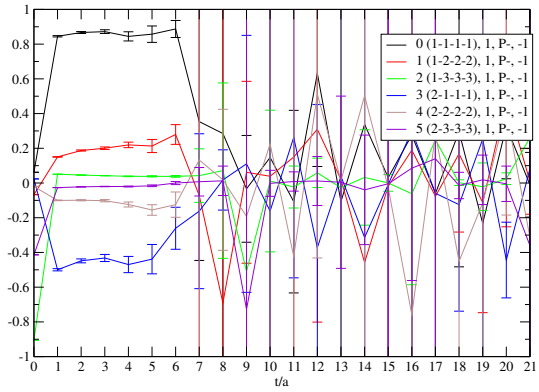
(a) Eigenvalues



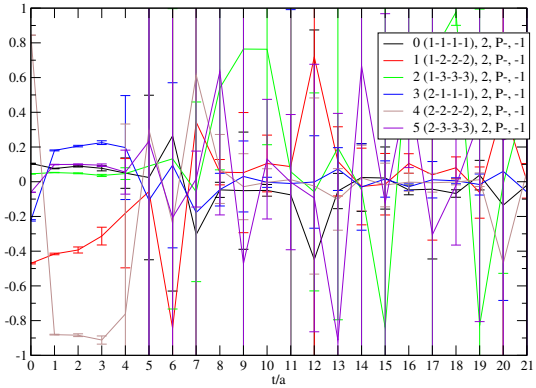
(b) Effective masses



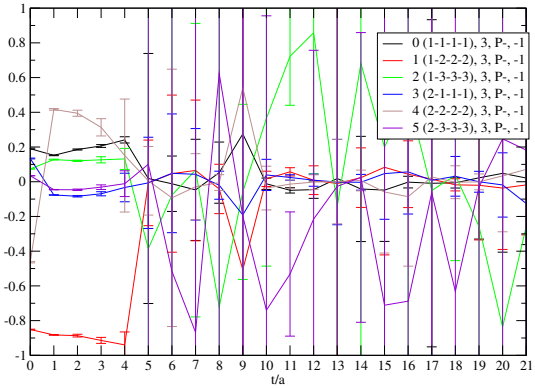
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

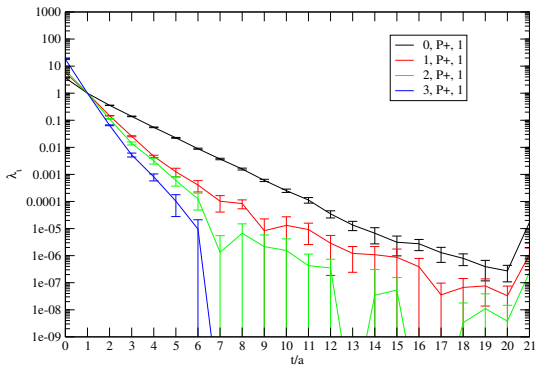


(e) Composition of eigenvector 3

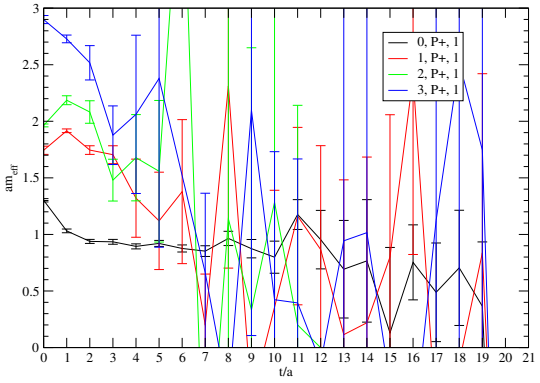


(f) Composition of eigenvector 4

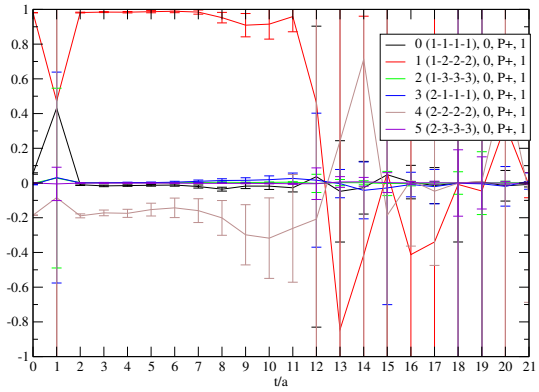
Figure 9.11: Results from the variational method for the negative parity nucleon on lattice C77.



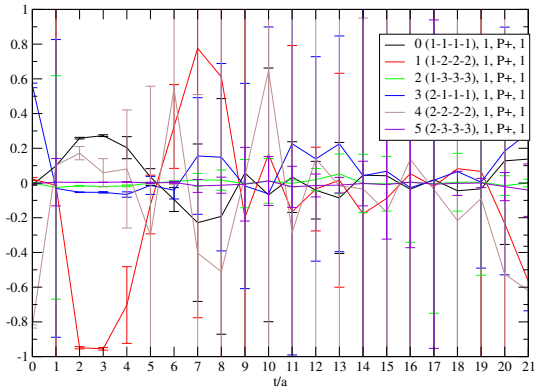
(a) Eigenvalues



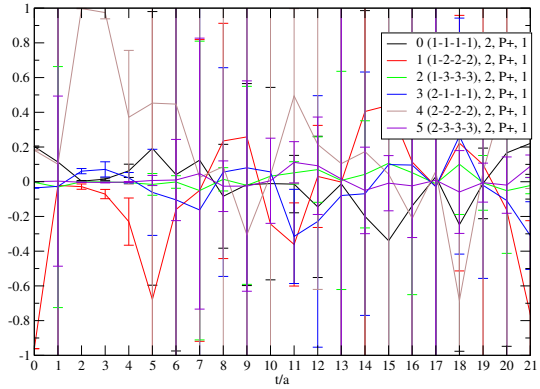
(b) Effective masses



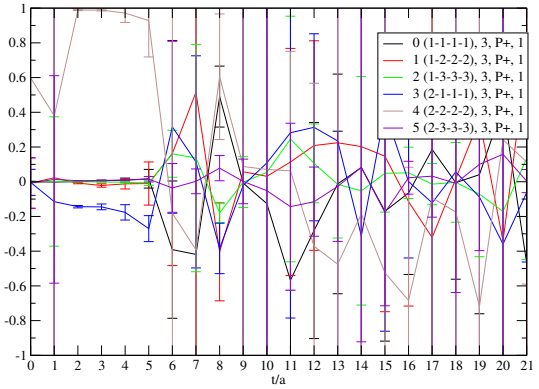
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

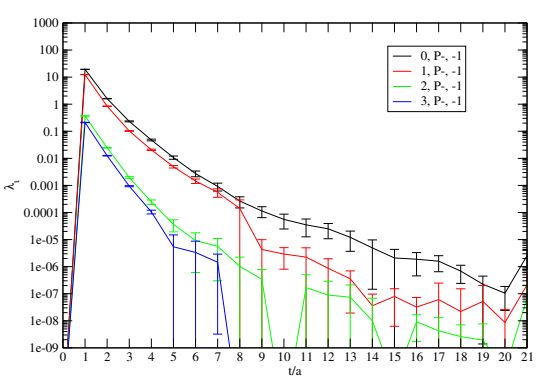


(e) Composition of eigenvector 3

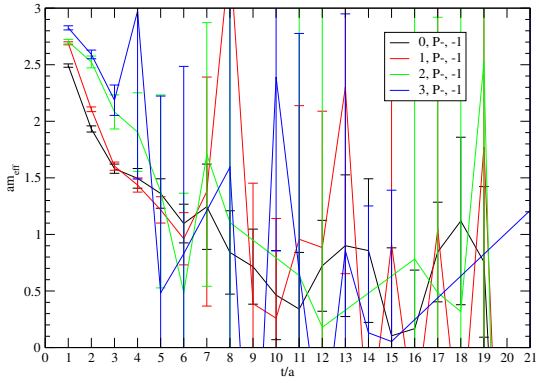


(f) Composition of eigenvector 4

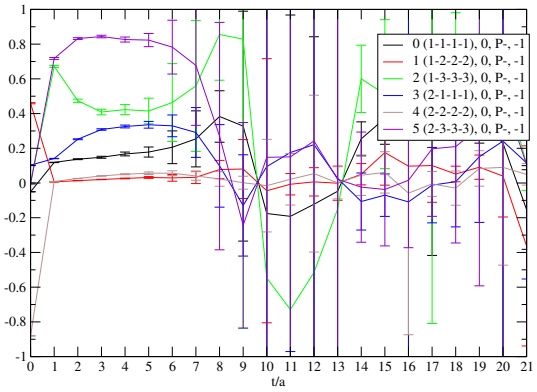
Figure 9.12: Results from the variational method for the positive parity nucleon on lattice C72.



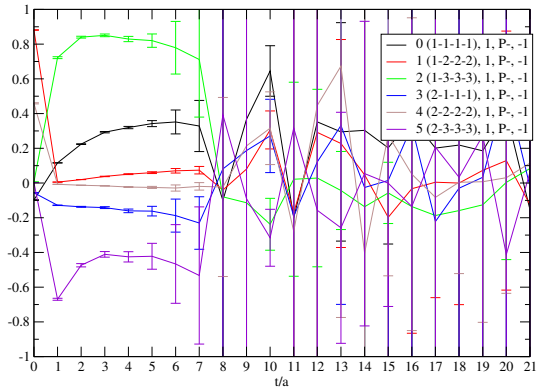
(a) Eigenvalues



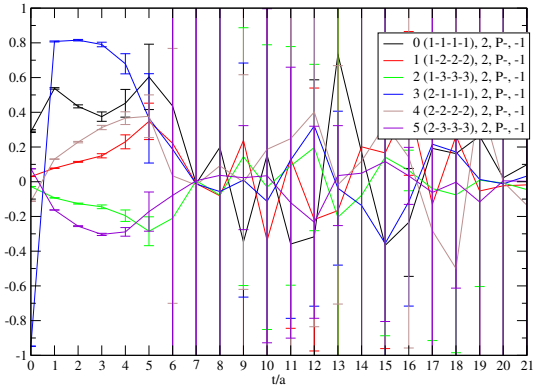
(b) Effective masses



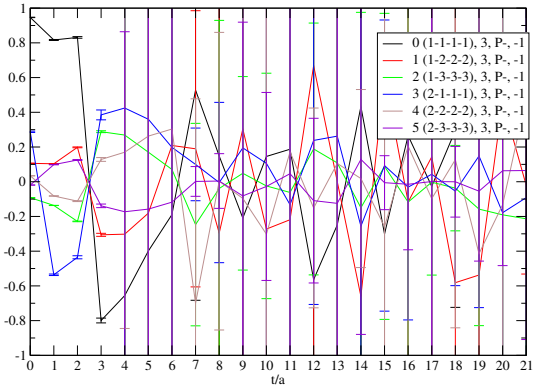
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

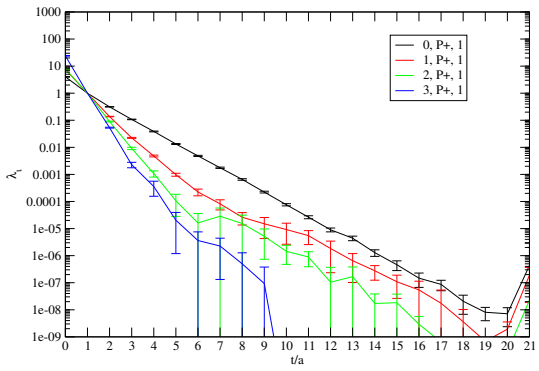


(e) Composition of eigenvector 3

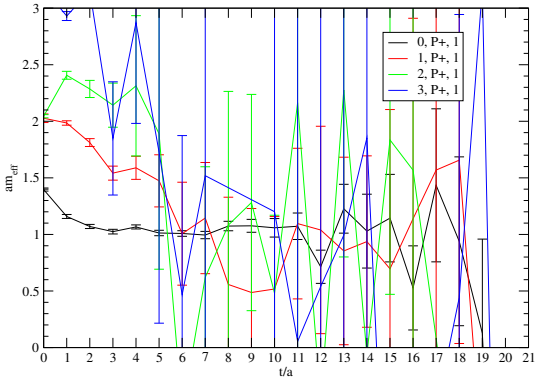


(f) Composition of eigenvector 4

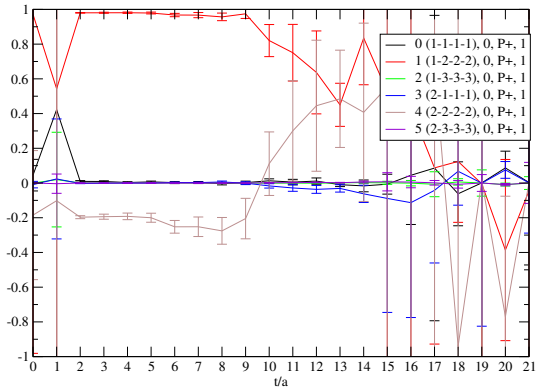
Figure 9.13: Results from the variational method for the negative parity nucleon on lattice C72.



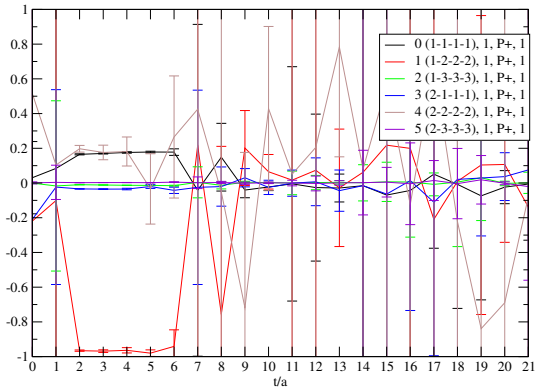
(a) Eigenvalues



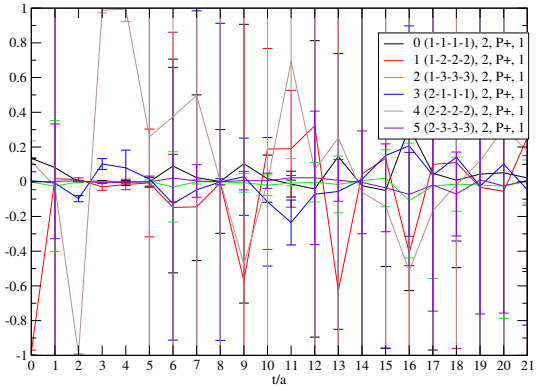
(b) Effective masses



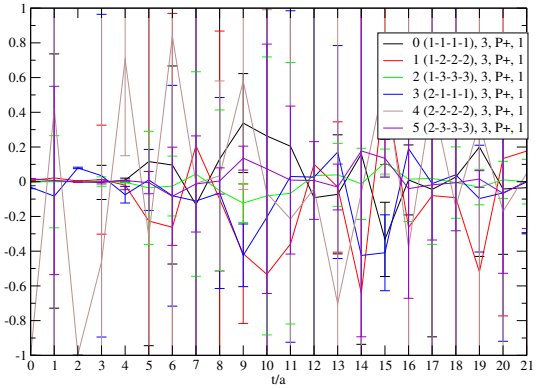
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

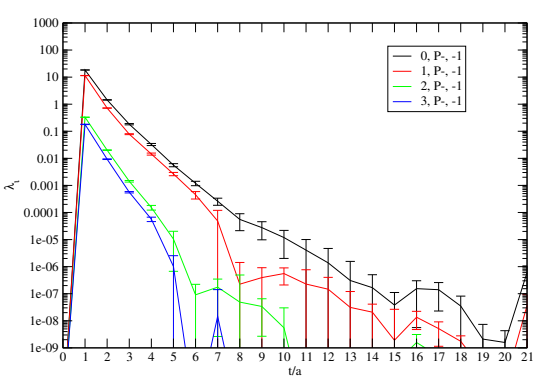


(e) Composition of eigenvector 3

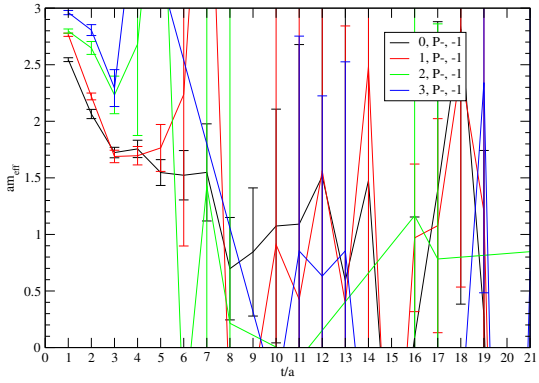


(f) Composition of eigenvector 4

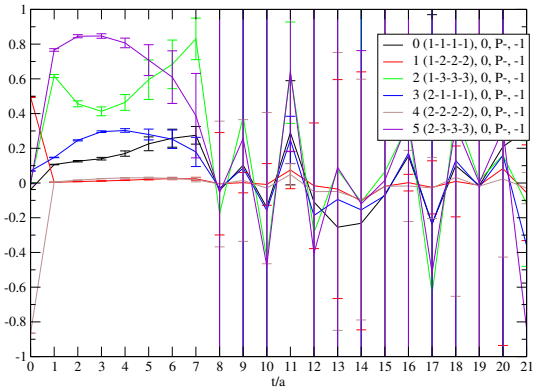
Figure 9.14: Results from the variational method for the positive parity nucleon on lattice C64.



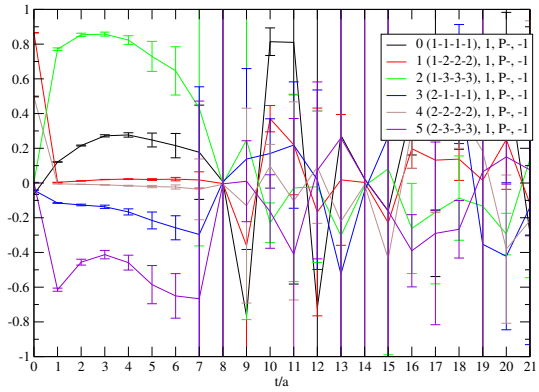
(a) Eigenvalues



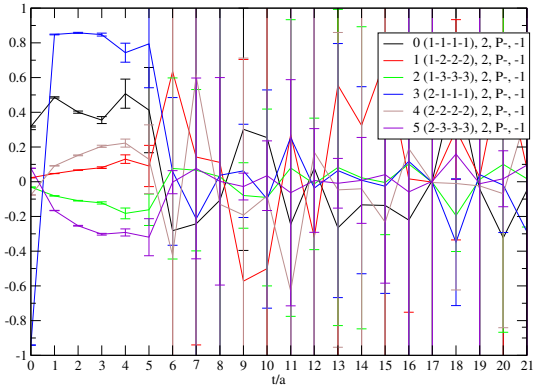
(b) Effective masses



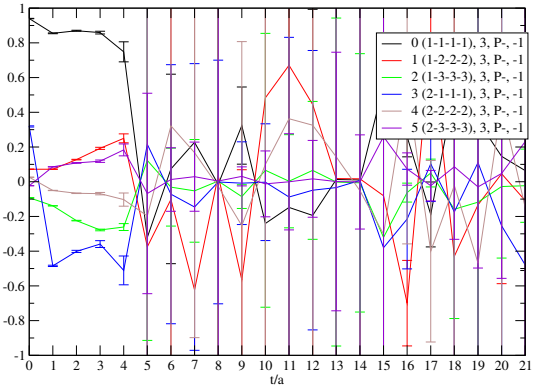
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

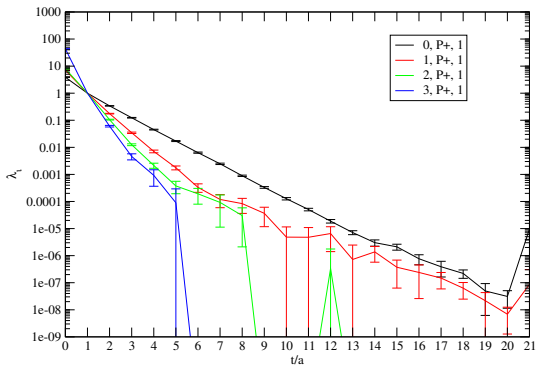


(e) Composition of eigenvector 3

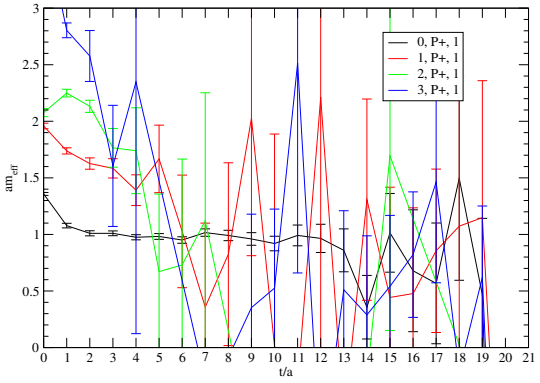


(f) Composition of eigenvector 4

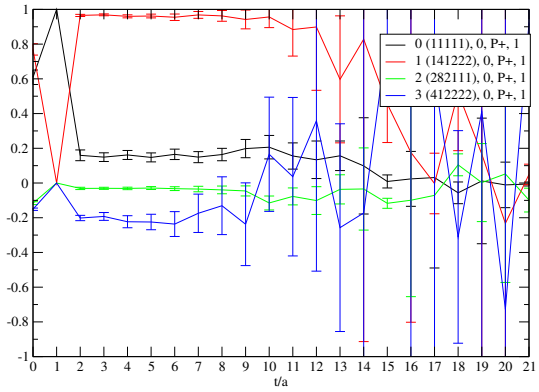
Figure 9.15: Results from the variational method for the negative parity nucleon on lattice C64.



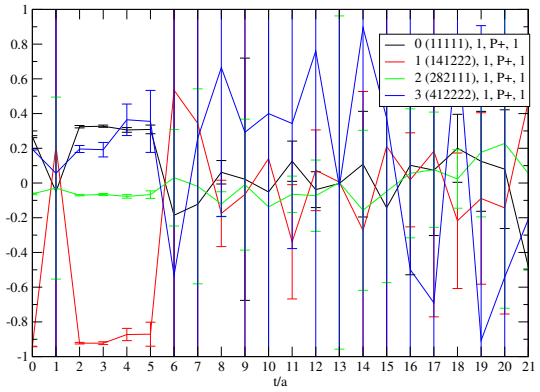
(a) Eigenvalues



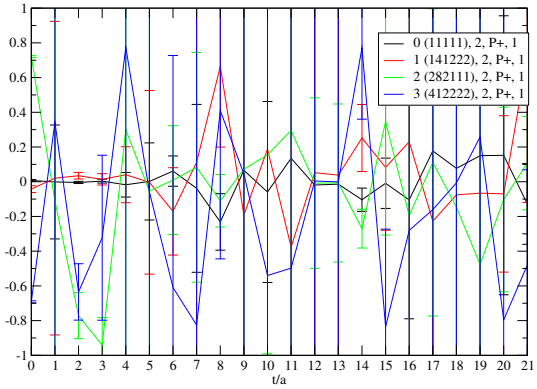
(b) Effective masses



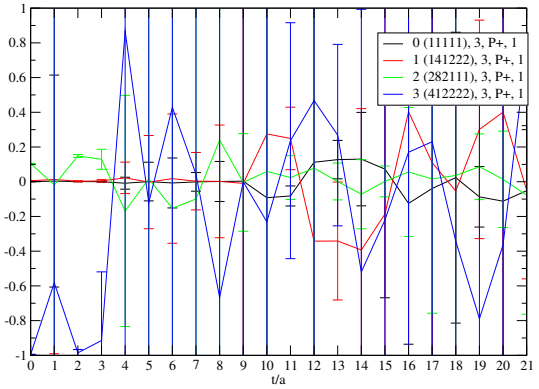
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2

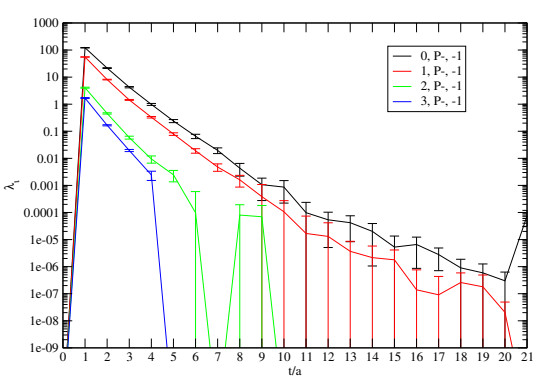


(e) Composition of eigenvector 3

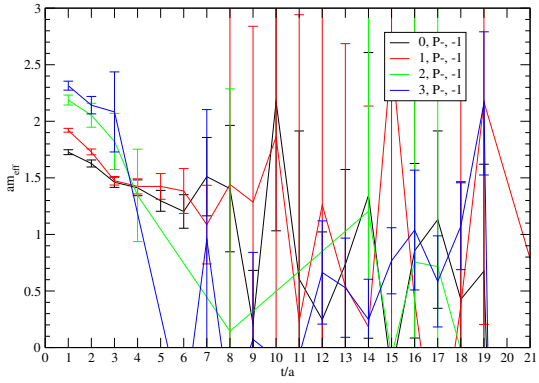


(f) Composition of eigenvector 4

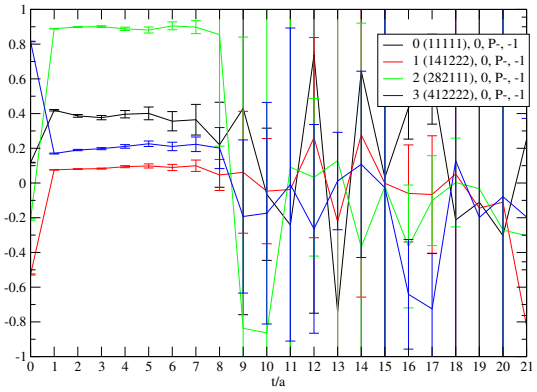
Figure 9.16: Results from the variational method for the positive parity nucleon on lattice A50.



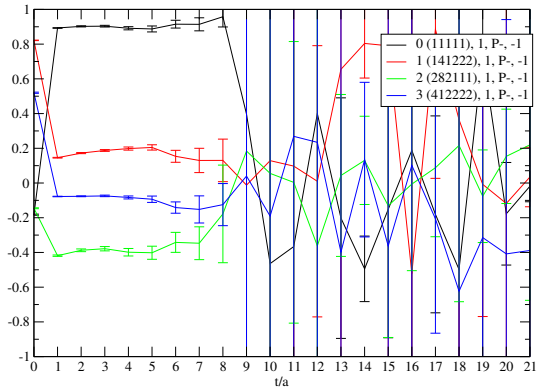
(a) Eigenvalues



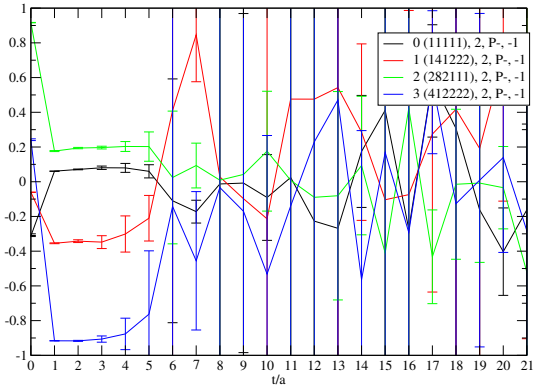
(b) Effective masses



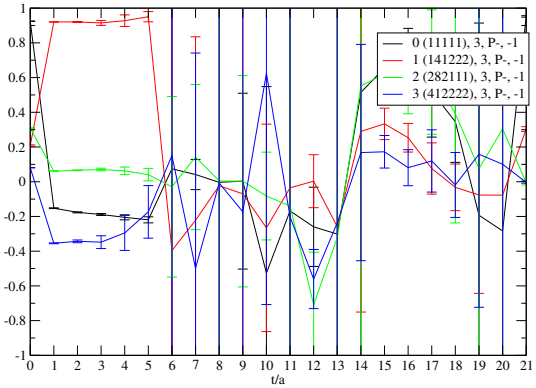
(c) Composition of eigenvector 1



(d) Composition of eigenvector 2



(e) Composition of eigenvector 3



(f) Composition of eigenvector 4

Figure 9.17: Results from the variational method for the negative parity nucleon on lattice A50.

9.2.1 Chiral Extrapolation

Because of the large errors obtained, we refrain from any fit to the masses obtained but only plot them with respect to the AWI quark mass.

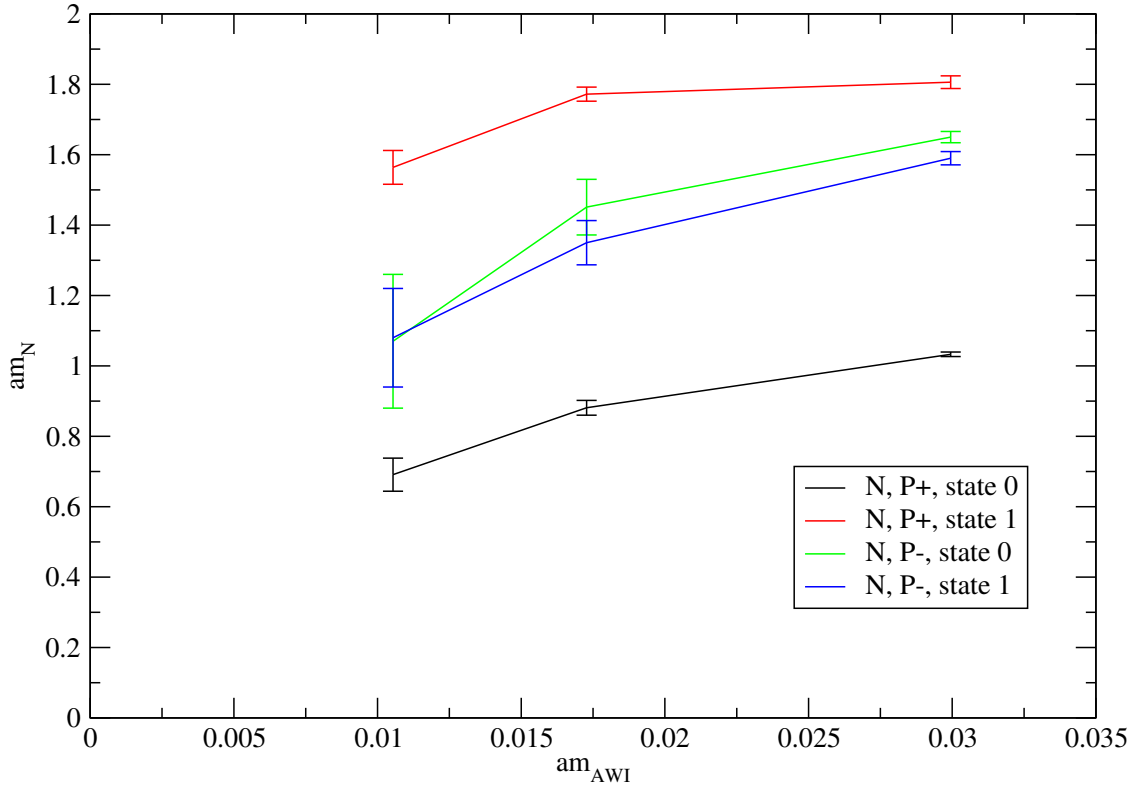


Figure 9.18: Dependence of the nucleon masses on the AWI quark mass (lattices C77, C72 and C64). Shown are the lowest two states of positive and negative parity.

9.3 Baryon Charges

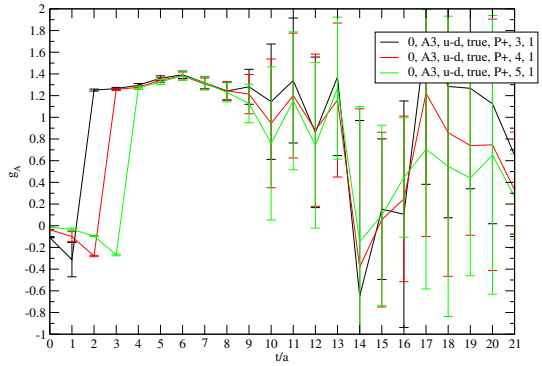
At this point we are ready to use the knowledge of the optimal operator combination for the evaluation of three-point correlation matrices $C^{\mathcal{O}}_{ij}$, see formula 8.117. In this section we show ratios of three-point over two-point functions by the use of the variational method, see formula 8.118:

$$R_k(\tau,t) = \frac{\psi_k(t)^\dagger C^{\mathcal{O}}(\tau,t)\psi_k(t)}{\psi_k(t)^\dagger C(t)\psi_k(t)}. \tag{9.3}$$

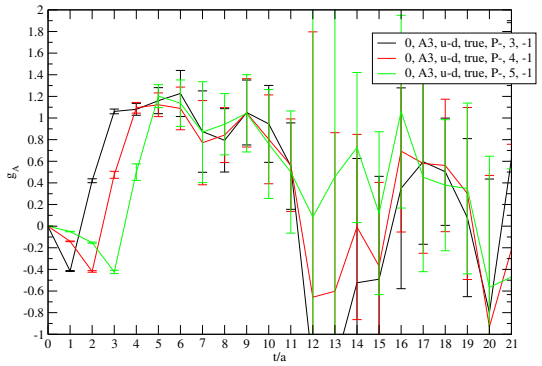
for vector and axial-vector operators, see formula 8.150

$$g^{\pm,\mathrm{ren}}_{A,V} = Z_{A,V} g^{\pm}_{A,V}(0) = \frac{\mathrm{Tr}\, P_\pm \Gamma^{A,V} \langle N(t) J^{A,V}_\mu(\tau) \overline{N}(0)\rangle}{\mathrm{Tr}\, P_\pm \langle N(t)\overline{N}(0)\rangle}. \tag{9.4}$$

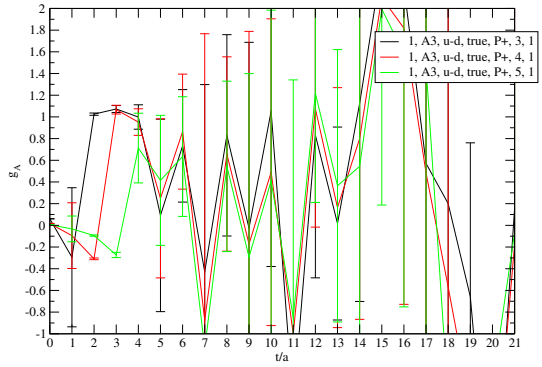
Here, we use the values for E_k determined in the previous section and plot the ratios R_k for each of our lattices for the two lowest states. We show results for $\tau \in \{3,4,5\}$. We show results for the (unrenormalized) axial charge g_A in figures 9.19, 9.20, 9.21 and 9.22 and results for the (unrenormalized) vector charge g_V in figures 9.23 and 9.24.



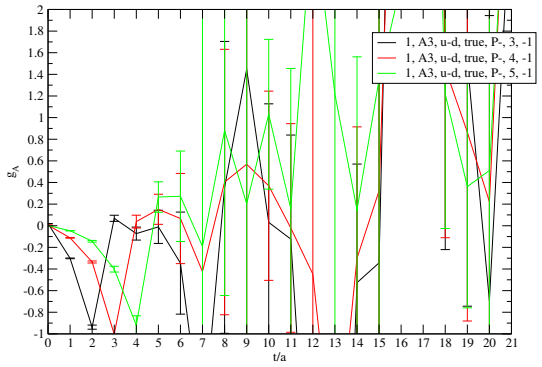
(a) positive parity nucleon state 0.



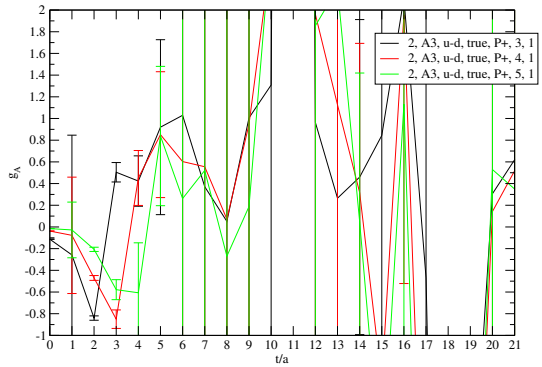
(b) negative parity nucleon state 0.



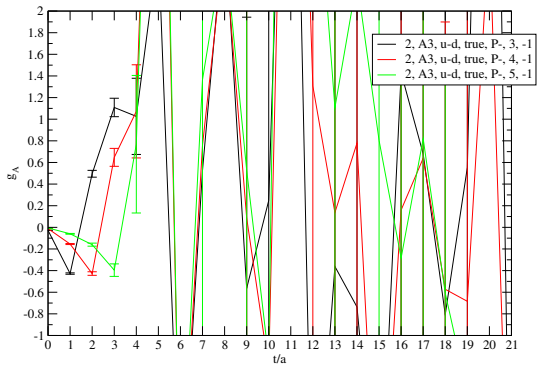
(c) positive parity nucleon state 1.



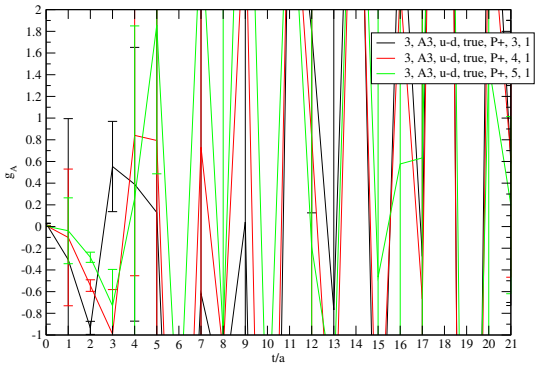
(d) negative parity nucleon state 1.



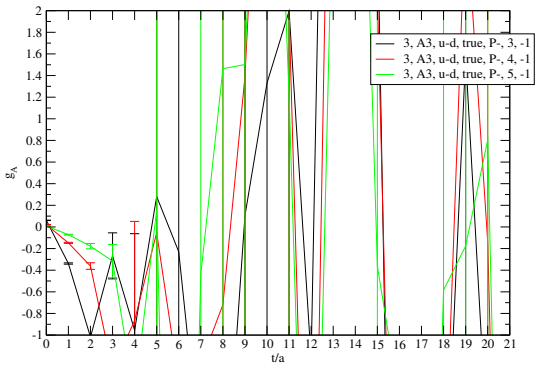
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.

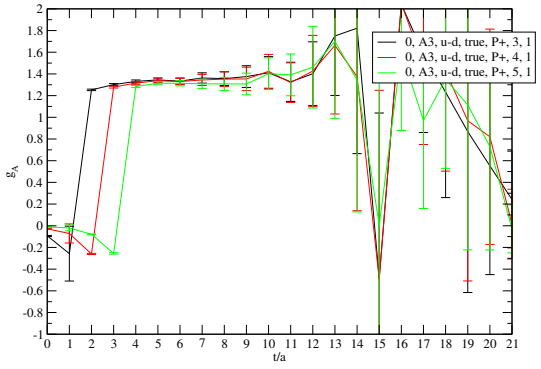


(g) positive parity nucleon state 3.

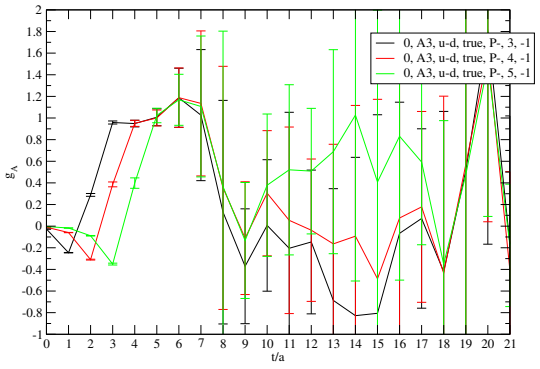


(h) negative parity nucleon state 3.

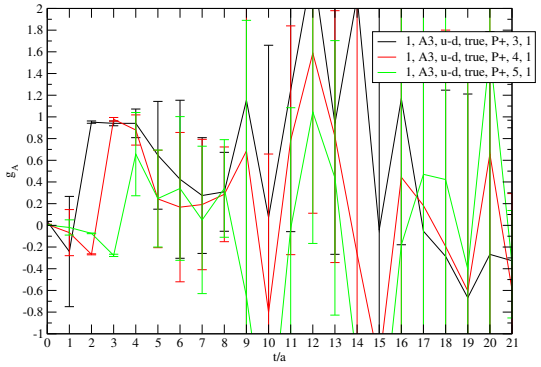
Figure 9.19: Axial charge for lattice C77 of the ...



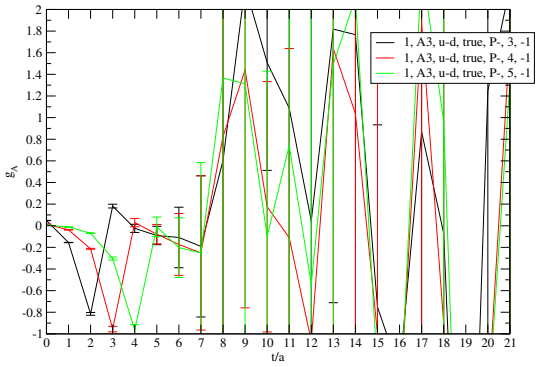
(a) positive parity nucleon state 0.



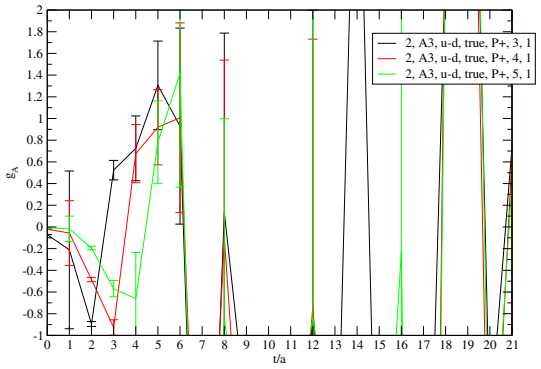
(b) negative parity nucleon state 0.



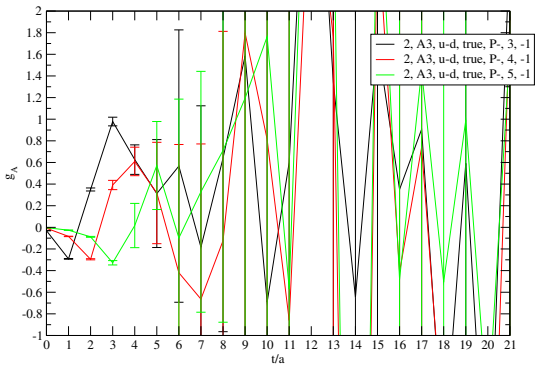
(c) positive parity nucleon state 1.



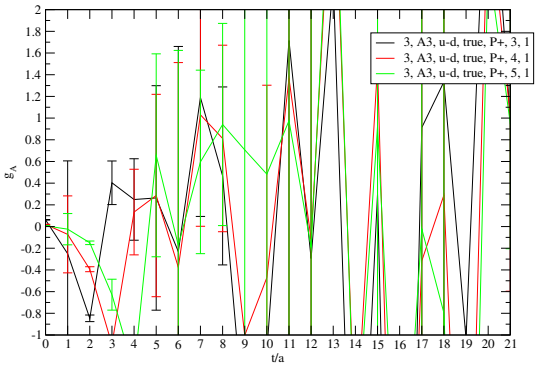
(d) negative parity nucleon state 1.



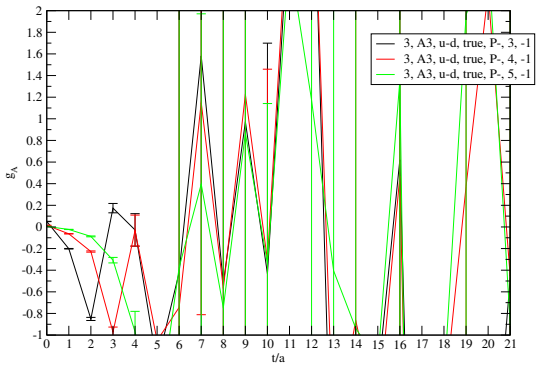
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.

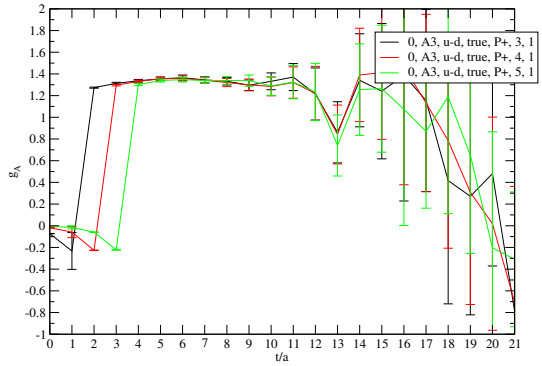


(g) positive parity nucleon state 3.

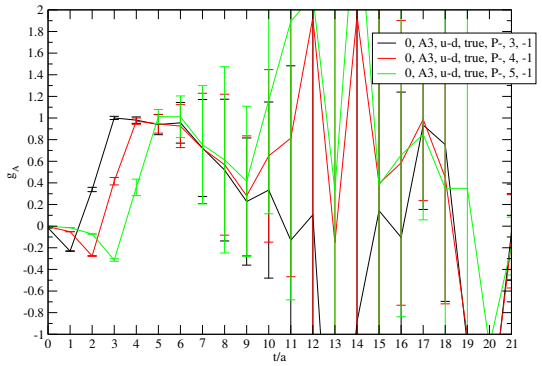


(h) negative parity nucleon state 3.

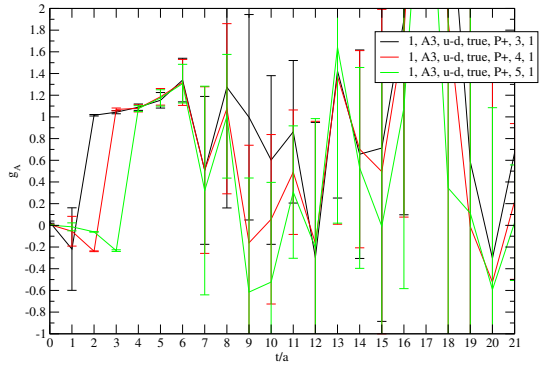
Figure 9.20: Axial charge for lattice C72 of the ...



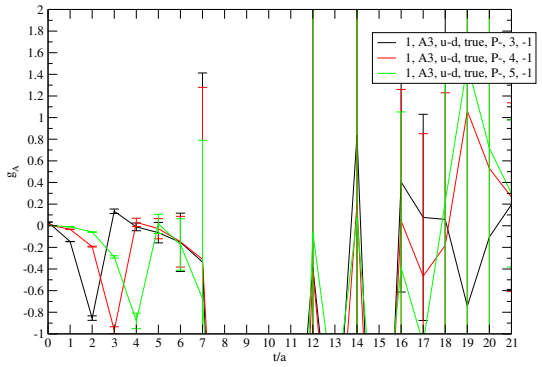
(a) positive parity nucleon state 0.



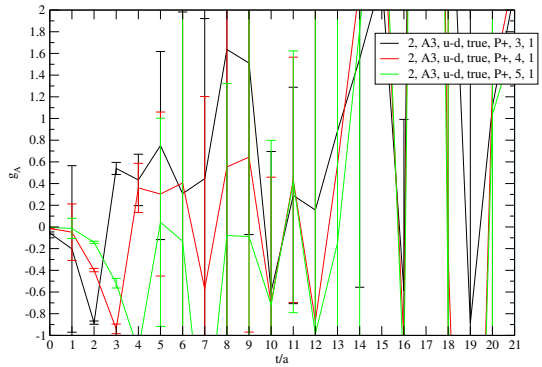
(b) negative parity nucleon state 0.



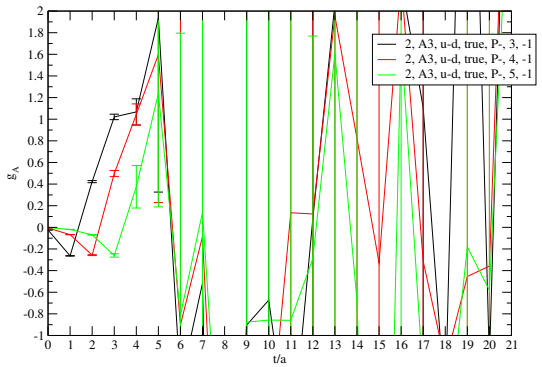
(c) positive parity nucleon state 1.



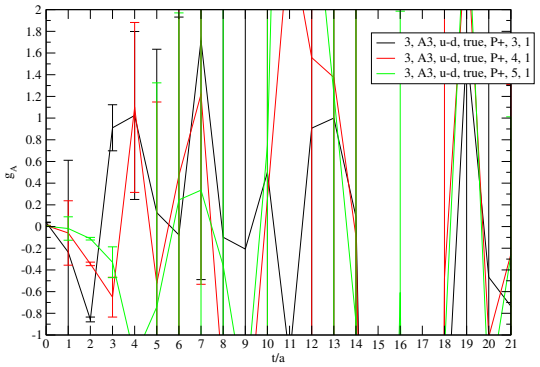
(d) negative parity nucleon state 1.



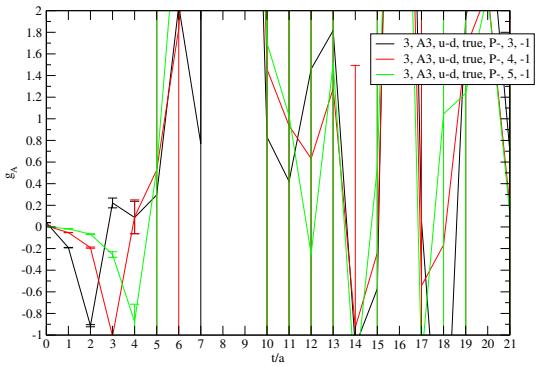
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.

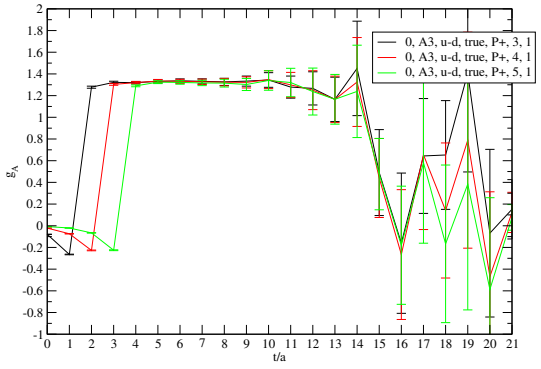


(g) positive parity nucleon state 3.

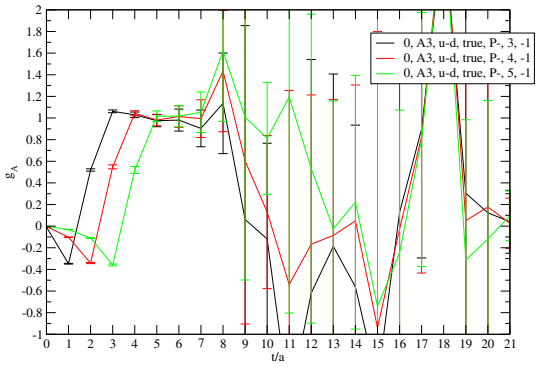


(h) negative parity nucleon state 3.

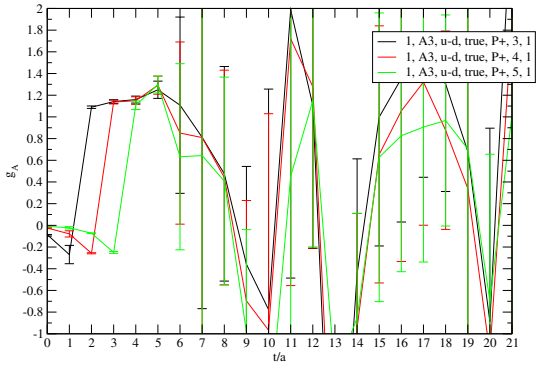
Figure 9.21: Axial charge for lattice C64 of the ...



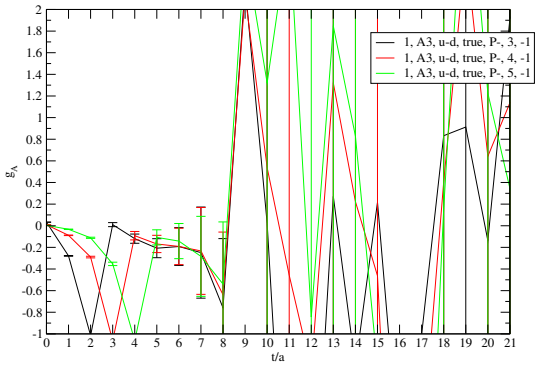
(a) positive parity nucleon state 0.



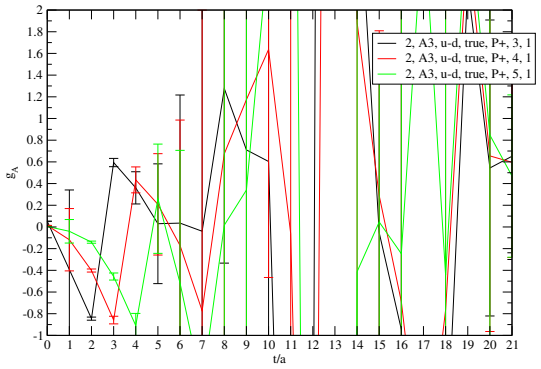
(b) negative parity nucleon state 0.



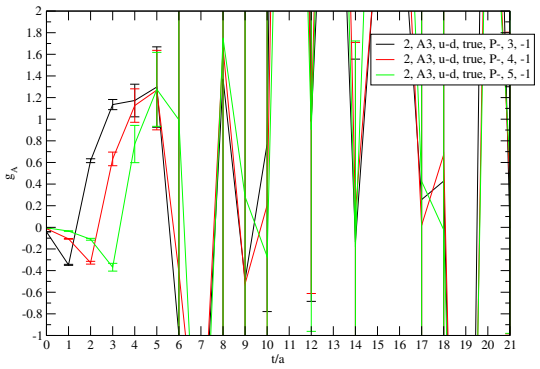
(c) positive parity nucleon state 1.



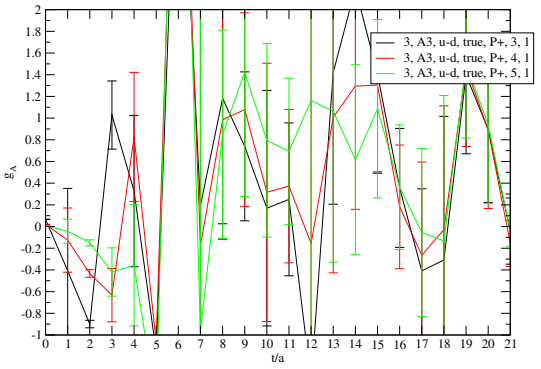
(d) negative parity nucleon state 1.



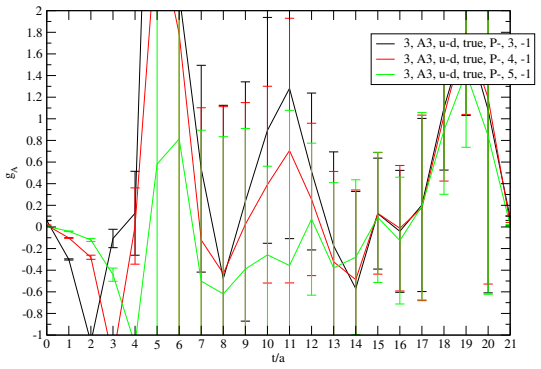
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.

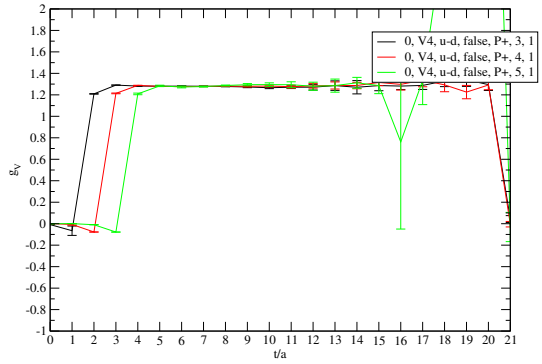


(g) positive parity nucleon state 3.

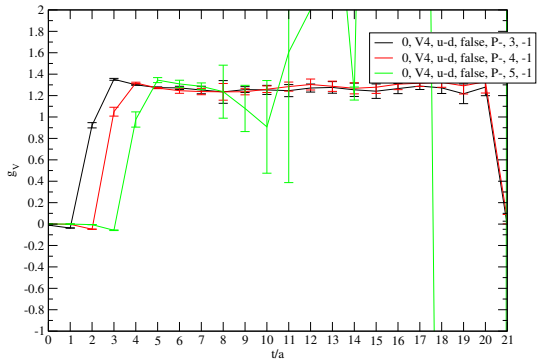


(h) negative parity nucleon state 3.

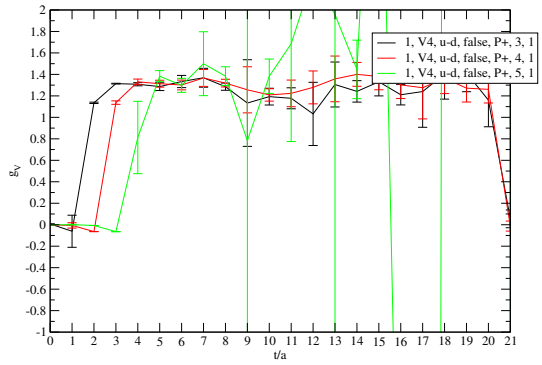
Figure 9.22: Axial charge for lattice A50 of the ...



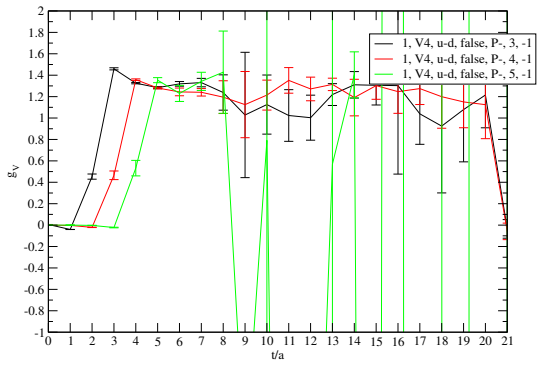
(a) positive parity nucleon state 0.



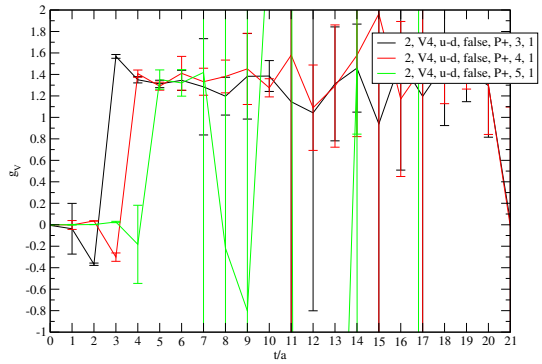
(b) negative parity nucleon state 0.



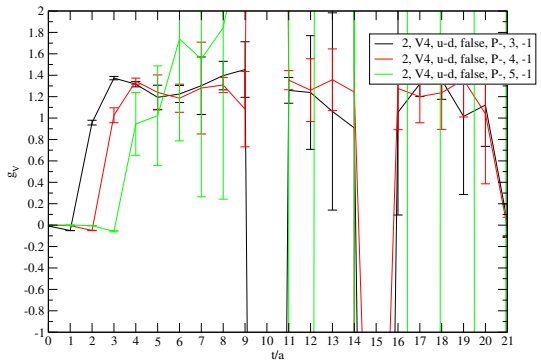
(c) positive parity nucleon state 1.



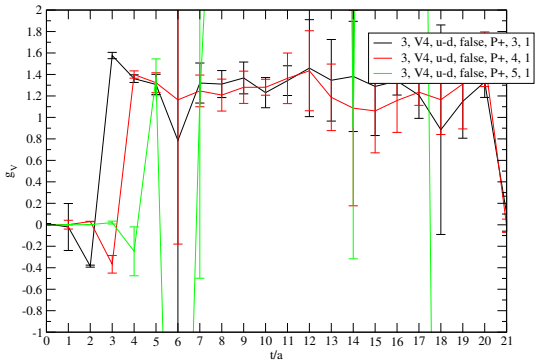
(d) negative parity nucleon state 1.



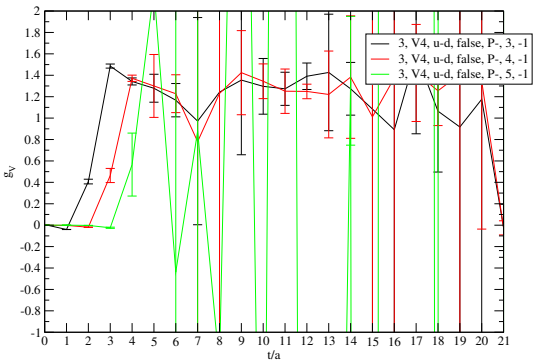
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.

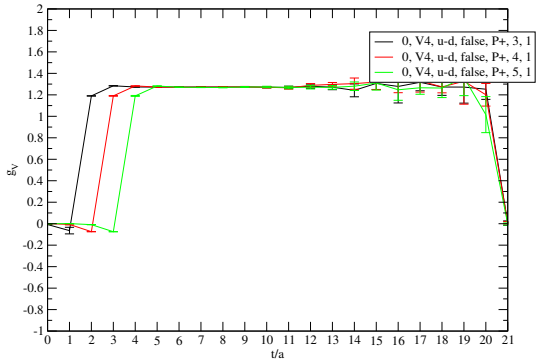


(g) positive parity nucleon state 3.

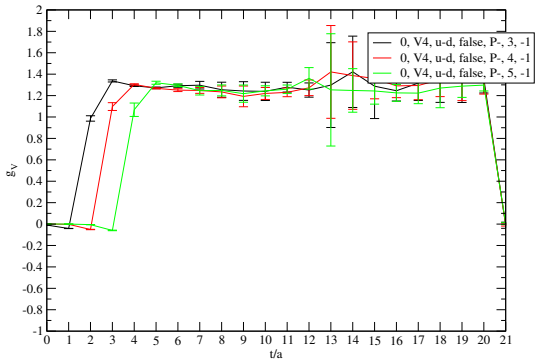


(h) negative parity nucleon state 3.

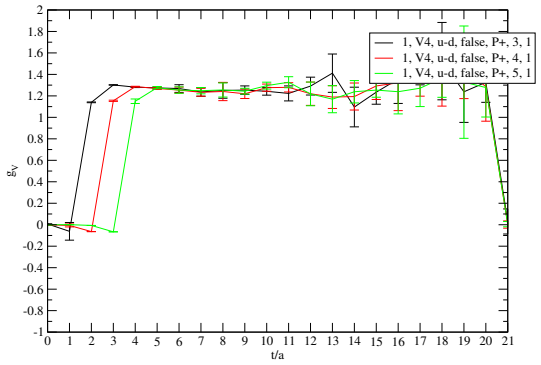
Figure 9.23: Vector charge for lattice C72 of the



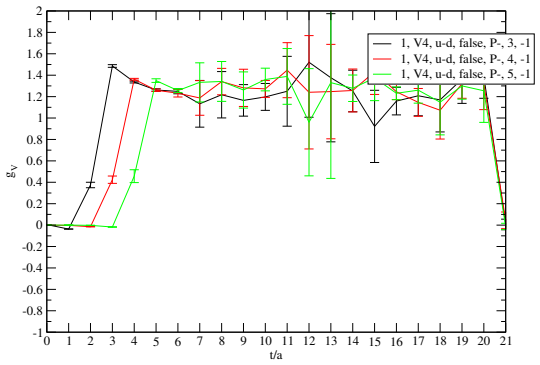
(a) positive parity nucleon state 0.



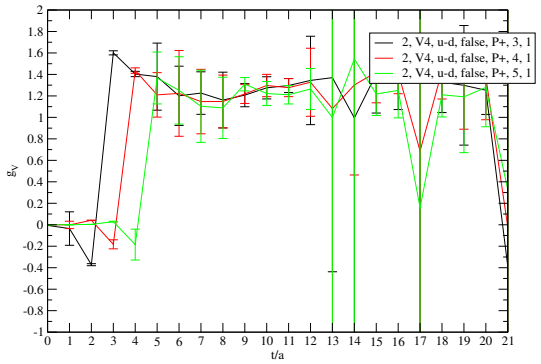
(b) negative parity nucleon state 0.



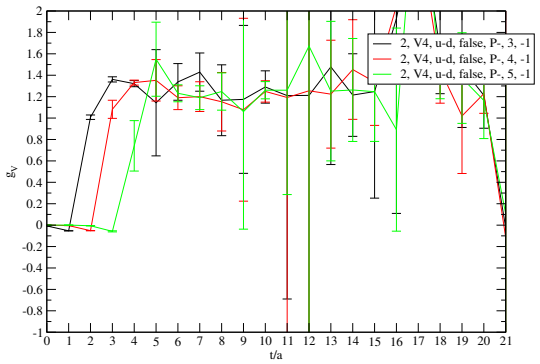
(c) positive parity nucleon state 1.



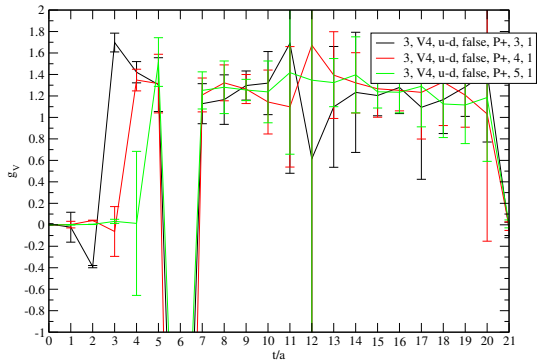
(d) negative parity nucleon state 1.



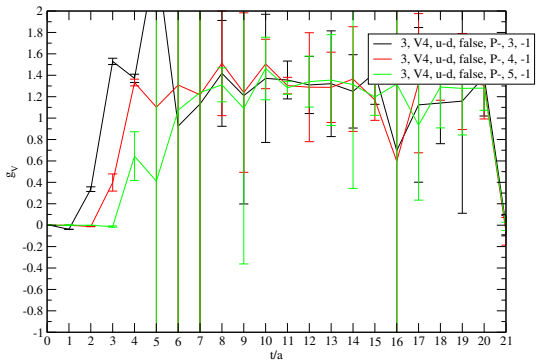
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.



(g) positive parity nucleon state 3.



(h) negative parity nucleon state 3.

Figure 9.24: Vector charge for lattice C64 of the ...

Ideally, we seek to extract plateau values using a combined fit in (t, τ) using the fit function

$$R(\tau, t) = R_0 + c_1 e^{-c_2 t} + c_3 e^{-c_4 \tau} + c_5 e^{-c_6 (t-\tau)} \tag{9.5}$$

in a region $t \in [\tau, t_{\max}]$, $\tau \in \{2, 3, 4\}$, where we use the same upper fit boundary $t_{\max}$ that we use to extract masses, see table 9.4.

Unfortunately fits turned out to be too unstable. Instead we extract plateau values where plateaus are obvious to the human eye. A list of results can be found in table 9.5.

The renormalized vector charges are expected to be consistent with $g_V \equiv 1$ for any state listed. For states where this is not the case, axial charges may show similar deviations from a correct value.

lattice	parity	state	plateau time t	R_A	R_V	$g_A = Z_A R_A$	$g_V = Z_V R_V$
C77	+	0	4	1.281(15)	-	1.137(13)	-
C77	+	1	4	0.888(186)	-	0.788(165)	-
C77	+	2	4	0.089(315)	-	0.079(280)	-
C77	+	3	4	0.497(1382)	-	0.441(1227)	-
C77	-	0	5	1.162(112)	-	1.032(99)	-
C77	-	1	5	0.136(144)	-	0.121(128)	-
C77	-	2	5	5.649(15605)	-	5.016(13855)	-
C77	-	3	5	0.128(2526)	-	0.114(2242)	-
C72	+	0	4	1.310(12)	1.260(3)	1.164(10)	1.036(3)
C72	+	1	4	0.826(218)	1.150(126)	0.733(194)	0.946(104)
C72	+	2	4	0.246(331)	0.859(142)	0.219(294)	0.706(117)
C72	+	3	4	-0.315(400)	0.838(99)	-0.280(355)	0.689(81)
C72	-	0	5	1.010(73)	1.296(12)	0.897(65)	1.066(10)
C72	-	1	5	-0.059(88)	1.306(14)	-0.053(78)	1.074(11)
C72	-	2	5	0.401(458)	1.152(247)	0.356(407)	0.947(204)
C72	-	3	5	-1.380(2473)	1.594(571)	-1.225(2196)	1.311(470)
C64	+	0	4	1.323(12)	1.249(2)	1.175(11)	1.027(2)
C64	+	1	4	1.082(30)	1.239(10)	0.960(26)	1.019(8)
C64	+	2	4	-0.114(228)	0.885(65)	-0.101(203)	0.728(53)
C64	+	3	4	0.305(736)	0.927(290)	0.271(654)	0.763(239)
C64	-	0	5	0.966(82)	1.287(9)	0.858(73)	1.058(7)
C64	-	1	5	-0.025(92)	1.290(13)	-0.022(82)	1.061(11)
C64	-	2	5	1.590(1342)	1.348(345)	1.412(1192)	1.109(284)
C64	-	3	5	0.452(2747)	1.344(35508)	0.401(2439)	1.105(29201)
A50	+	0	4	1.310(10)	-	1.178(9)	-
A50	+	1	4	1.139(36)	-	1.024(32)	-
A50	+	2	4	-0.038(127)	-	-0.034(114)	-
A50	+	3	4	0.265(617)	-	0.238(555)	-
A50	-	0	5	0.991(53)	-	0.891(48)	-
A50	-	1	5	-0.161(79)	-	-0.145(71)	-
A50	-	2	5	1.281(360)	-	1.152(324)	-
A50	-	3	5	3.215(937168)	-	2.891(842607)	-

Table 9.5: Charges measured. Renormalization factors were taken from chapter 10.

9.3.1 Chiral Extrapolation

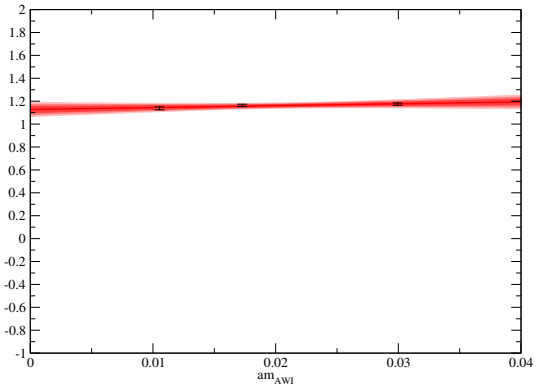
We perform a chiral extrapolation for the axial charge. On lattice C we do have 3 data points available and chose to use a linear extrapolation in the AWI quark mass of the renormalized charges

$$g_A^{\text{ren}} = Z_A g_A, \tag{9.6}$$
$$g_V^{\text{ren}} = Z_V g_V. \tag{9.7}$$

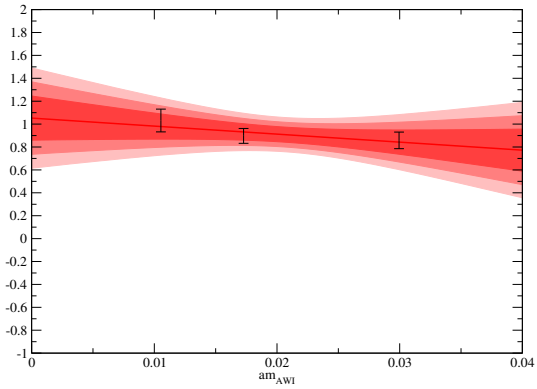
We plot results in figure 9.26 and quote our fully chirally extrapolated results:

lattice	parity	state	$g_A^{\text{ren,ext}}$
C	+	0	1.128(18)
C	+	1	0.63(22)
C	+	2	0.29(40)
C	+	3	-0.63(109)
C	-	0	1.05(13)
C	-	1	0.062(16)
C	-	2	-1.04(188)
C	-	3	-0.78(341)

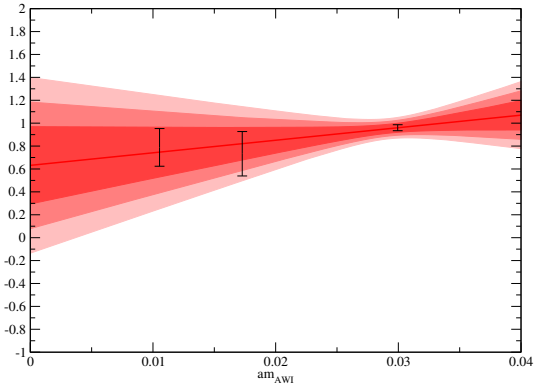
Figure 9.25: Axial charges of excited nucleons on lattice C77.



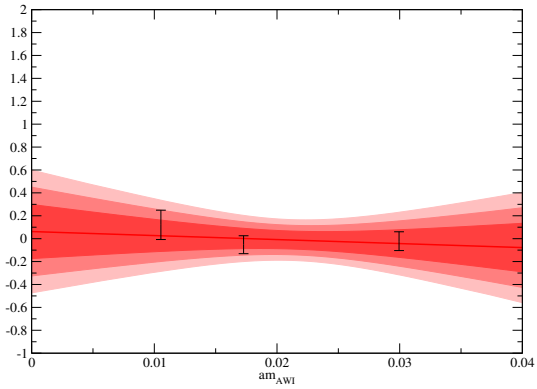
(a) positive parity nucleon state 0.



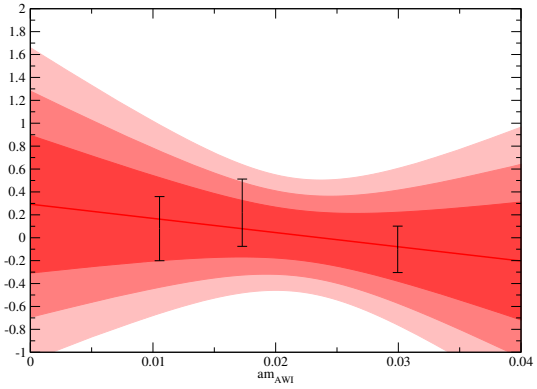
(b) negative parity nucleon state 0.



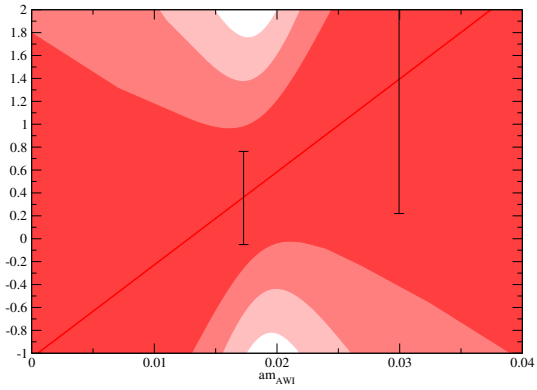
(c) positive parity nucleon state 1.



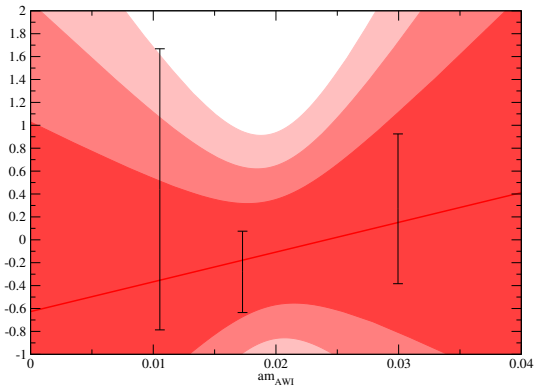
(d) negative parity nucleon state 1.



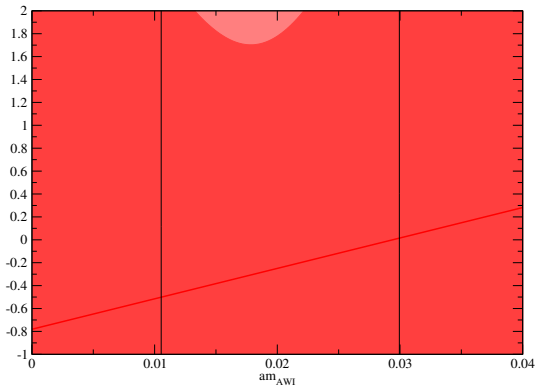
(e) positive parity nucleon state 2.



(f) negative parity nucleon state 2.



(g) positive parity nucleon state 3.



(h) negative parity nucleon state 3.

Figure 9.26: Extrapolation of the renormalized axial charge for lattice C. We show data points as well as 1, 2 and 3σ error bands.

Part III

Renormalization of Dynamical CI Fermions

Chapter 10

Introduction to Lattice Renormalization

10.1 Continuum Renormalization

Regularization and renormalization are essential steps in calculating physical predictions from relativistic quantum field theories.

In continuum perturbation theory one encounters the problem of divergences when integrating over internal loop momenta. This is not a mathematical but a physical problem which has been understood and solved by Wilson. At very high energies all available quantum field theories break down as the fourth interaction, gravity, has to be taken into account. It is known that this may happen on the order of the Planck scale. Even below that scale other presently unknown effects could interfere. But Wilson's "Renormalization Group" flow explains a change in magnitude of different parts of the interaction, or more technically speaking the relevance of operators in the Lagrangian, towards fixed points, the running of the coupling with the scale and why at low scales we are essentially independent of very high energy effects. This is connected to the fact that the naive Lagrangian contains "bare" couplings only. At first hand, these "bare" couplings are unphysical mathematical objects necessary to parameterize the theory. The process of renormalization combines the bare quantities with the divergences[1], or actually with our ignorance of physics at high scales, into physical parameters which we do measure. For a short introduction we refer to [40].

The most prominent renormalization scheme in perturbation theory is Modified Minimal Subtraction ($\overline{\text{MS}}$), a modification of the Minimal Subtraction (MS) scheme: As a first step, one regularizes the divergent integrals using dimensional regularization, i.e., one replaces the 4 dimensional spacetime by a $4-\epsilon$ dimensional one, such that all integrals become finite but dependent on the introduced regularization parameter ϵ (and a scale μ which appears in a necessary redefinition of the coupling constant). Now the MS scheme demands to subtract all power divergences in ϵ and the more convenient $\overline{\text{MS}}$ scheme also demands to subtract certain constant terms proportional to $\ln 4\pi$ or the Euler-Mascheroni constant γ, as all these would cancel in physical results anyway.

10.2 Lattice Regularization

As in the continuum we have to renormalize matrix elements to make them comparable with experiments. However, in contrast to continuum calculations all results obtained from a lattice calculation are already finite as we have already chosen a special regularization. This lattice

[1] In this context the term "divergence" must be carefully interpreted as the behavior of certain terms contributing to Feynman diagram which *would* diverge if the "cutoff" were taken to infinity. In reality the "cutoff" is most likely not infinity but may be somewhere near the Planck scale.

regularization scheme is more or less some kind of momentum cutoff, introduced by the finite lattice spacing. Generally operators have to be renormalized by

$$\mathcal{O}^{\mathcal{S}}(M) = Z^{\mathcal{S}}_{\mathcal{O};\mathrm{bare}}(M)\mathcal{O}_{\mathrm{bare}} \tag{10.1}$$

using a scheme $\mathcal{S}$ at some scale M. Ideally we would like our final results to be given in the $\overline{\mathrm{MS}}$ scheme at a scale of $\mu = 2\,\mathrm{GeV}$. For example, if we wish to calculate hadronic structure functions, then the renormalized continuum matrix element has to be multiplied by the corresponding Wilson coefficient to give the final result. As a physical result must be independent of the regularization procedure employed, the regularization parameters, for example some scale μ, must drop out in the product of Wilson coefficient and renormalization constant. Since Wilson coefficients are usually computed using the $\overline{\mathrm{MS}}$ scheme, we will have to convert our lattice regularized results to the same $\overline{\mathrm{MS}}$ scheme. We could of course determine the renormalization factors using lattice perturbation theory [102] and then convert them to $\overline{\mathrm{MS}}$. Due to its poor convergence properties the authors or reference [58] have proposed a "tadpole improved" lattice perturbation theory. Still the ignorance of higher order contributions is not a minor obstacle in the extraction of physical quantities. Additionally non-perturbative parts of the renormalization may be missing.

One way out is to use a non-perturbative renormalization procedure based on Monte Carlo simulations [103]. We want to use the Modified RI-MOM (RI'-MOM) scheme which is a close relative to the Regularization Independent MOM (RI-MOM) scheme, both proposed by Martinelli et al. [104]. These Momentum Subtraction (MOM) renormalization schemes are by construction independent of the regularization employed, which is the lattice regularization in our case. The RI scheme is difficult to handle on the lattice as its definition includes a derivative with respect to a discrete variable. Therefore we concentrate on the RI'-MOM scheme in this work.

10.3 The Regularization Independent MOM Schemes

The RI schemes mimic what is usually done in continuum perturbation theory in defining the renormalization conditions for a certain operator $\mathcal{O}$ by imposing that the corresponding Green functions coincide with their tree level value (often called Born term) at the scale μ of their external momenta.

$$Z_{\mathcal{O}}\,\langle p|\mathcal{O}|p\rangle|_{p^2=-\mu^2} = \langle p|\mathcal{O}|p\rangle_0 \tag{10.2}$$

This procedure has the advantage of defining the same renormalized operators in all regularization schemes (as long as the same renormalized coupling is used). However we have to fix the gauge completely such that the matrix elements in formula (10.2) do not vanish. As most other groups, we use Landau gauge here. We fix this gauge using over-relaxation. For a short introduction to gauge fixing on the lattice we refer to [105].

This basic definition can be translated to the lattice. The resulting procedure is expected to work in a window

$$\Lambda^2_{\mathrm{QCD}} \ll \mu^2 \ll 1/a^2, \tag{10.3}$$

where discretization effects can be neglected because the renormalization scale μ is small compared to the lattice cutoff $1/a$. On the other end of the energy scale we need to use continuum perturbation theory to connect different schemes. This works if the energy scale μ is much larger than the QCD scale parameter Λ_{QCD}. If this window exists we can obtain

the renormalization factor $Z_{\mathcal{O}}$ for, for example the generic two-quark operator $\mathcal{O} = \bar{\psi}\Gamma\psi$ by requiring

$$Z_{\mathcal{O}} Z_{\psi}^{-1} \Gamma_{\mathcal{O}}|_{p^2=\mu^2} = 1. \tag{10.4}$$

Here the Z_{ψ}^{-1} accounts for the renormalization of the two quark fields in the operator $\mathcal{O}$ and $\Gamma_{\mathcal{O}}$ is defined by

$$\Gamma_{\mathcal{O}} = \frac{1}{12}\mathrm{Tr}\left(\Lambda_{\mathcal{O}}\hat{P}_{\mathcal{O}}\right) \tag{10.5}$$

in terms of the amputated Green function $\Lambda_{\mathcal{O}}$,

$$\Lambda_{\mathcal{O}} = S^{-1} C_{\mathcal{O}} S^{-1}, \tag{10.6}$$

and a suitable projector $\hat{P}_{\mathcal{O}}$ which in many cases can be taken as the inverse Born term of $\Lambda_{\mathcal{O}}$. Here S is the quark propagator. Altogether we can write the RI'-MOM renormalization factor as

$$Z_{\mathcal{O}}^{\mathrm{RI'}}(\mu) = \left.\frac{12 Z_{\psi}^{\mathrm{RI'}}(\mu)}{\mathrm{Tr}\left(\Lambda_{\mathcal{O}}(p)\left(\Lambda_{\mathcal{O}}^{\mathrm{Born}}(p)\right)^{-1}\right)}\right|_{p^2=\mu^2}. \tag{10.7}$$

There is quite a number of different RI-MOM schemes based on the different lattice definitions of the quark propagator renormalization factor. However, all of them will lead to the same results in the continuum (i.e., $a \to 0$) limit. The original RI-MOM scheme is difficult to handle on the lattice as its definition includes a derivative with respect to a discrete variable:

$$Z_{\psi}^{\mathrm{RI}}(\mu) = -i\frac{1}{12}\mathrm{Tr}\left.\frac{\partial S(p)^{-1}}{\partial \not{p}}\right|_{p^2=\mu^2} \tag{10.8}$$

10.4 The Modified RI-MOM Schemes

In the RI'-MOM schemes one exploits a corner-stone of renormalization: In the continuum limit, the full propagator S, a Dirac matrix, turns 'parallel' to the free propagator S_{Born} for every scale $\mu^2 = p^2$. In this limit the following equation holds:

$$Z_{\psi}^{\mathrm{RI'}}(\mu) S(p) = S_{\mathrm{Born}}(p). \tag{10.9}$$

Therefore one defines the quark field renormalization factor by the following formula:

$$Z_{\psi}^{\mathrm{RI'}}(\mu) = \frac{1}{12}\left.\mathrm{Tr}\left(S^{-1}(p) S_{\mathrm{Born}}(p)\right)\right|_{p^2=\mu^2}. \tag{10.10}$$

It extracts the factor by taking the average off all diagonal elements of the propagator. Equally well, one can also use a different definition:

$$\frac{1}{Z_{\psi}^{\mathrm{RI'}}(\mu)} = \frac{1}{12}\left.\mathrm{Tr}\left(S(p) S_{\mathrm{Born}}^{-1}(p)\right)\right|_{p^2=\mu^2}. \tag{10.11}$$

Here, the free propagator S_{Born} needs to converge to the free continuum propagator in the continuum limit. Therefore a possible choice is the continuum propagator itself:

$$S_{\mathrm{Born}}^{\mathrm{continuum}} = \frac{-i\gamma_\mu p_\mu}{p^2}. \tag{10.12}$$

A huge number of free propagators exist, but in order to obtain the mildest continuum extrapolation one should choose a free propagator that best matches the properties of the full propagator. For this reason another choice is the naive lattice propagator, which is the traditional RI'-scheme choice and may work best for Wilson-type fermions:

$$S_{\text{Born}}^{\text{lattice}} = \frac{-i\gamma_\mu \sin p_\mu}{\sum_\mu \sin^2 p_\mu}. \tag{10.13}$$

For CI fermions there is a number of other choices. This is because when we determine a free quark propagator we have to set the gauge links to $U = \mathbb{1}$. If we look back at the expansion of the CI Dirac operator in formula (8.76) we see that all the tensor, axial and pseudo-scalar terms vanish because of the totally antisymmetric tensors multiplying the paths $\langle l_i, l_j, \ldots \rangle$ which are now totally symmetric. Only the scalars and vector contributions survive. In equation (10.14) R and a_μ denote the scalar and vector terms appearing in the free CI Dirac operator, which in momentum space reads

$$D_{\text{CI}}^{\text{free}}(p) = \mathbb{1}R(p) + i\gamma_\mu a_\mu(p). \tag{10.14}$$

Using Fourier transformation one can compute R and a_μ from the definition and the parameters of the CI Dirac operator. In the continuum limit they are normalized such that one finds

$$a_\mu(p) = p_\mu + \mathcal{O}(ap)^2 \qquad \text{and} \qquad R(p) = \mathcal{O}(ap)^2. \tag{10.15}$$

To this end we need the general form of a free CI quark propagator:

$$S_{\text{Born}}^{\text{CI, } U(p)=\mathbb{1}}(p) = \frac{\mathbb{1}R(p) - i\gamma_\mu a_\mu(p)}{R(p)^2 + \sum_\mu a_\mu(p)^2}. \tag{10.16}$$

At this stage we again stress that any free propagator with the right continuum limit can be chosen. The CI-propagator with the coefficients for the full simulation has been optimized to approximate Ginsparg-Wilson best for the interaction case. So its continuum extrapolation behavior for the free $U = 1$ case will be suboptimal, however it is still correct. Therefore these coefficients form a possible choice for a free CI quark propagator

$$S_{\text{Born}}^{\text{CI-full, } U(p)=\mathbb{1}}(p). \tag{10.17}$$

We can do better if we choose a CI-propagator whose coefficients fulfil best the Ginsparg-Wilson equation for the free $U = 1$ case, i.e., we match our as-chiral-as-possible-on-the-lattice interacting CI operator to the as-chiral-as-possible-on-the-lattice non-interacting CI operator

$$S_{\text{Born}}^{\text{CI-free, } U(p)=\mathbb{1}}(p). \tag{10.18}$$

Both sets of CI coefficients can be found in Appendix A.3.

10.5 Computational Steps

To actually perform a numerical computation of these RI'-MOM renormalization factors a number of steps have to be taken:

10.5.1 Quark Propagator

From formula (10.7) and then formula (10.10) we see that we need the quark field renormalization factor and thus the quark propagator in momentum space $S(p)$. It is defined by

$$S(p) = \frac{1}{V} \sum_{x,y} e^{-ip(x-y)} \langle q(x)\bar{q}(y)\rangle_{q,U} \tag{10.19}$$

Further, we cannot be sure about the scale window (10.3). Therefore we should calculate renormalization factors for as many as possible, ideally diagonal, lattice momenta

$$p = 2\pi \left(\frac{k_1}{L_1}, \frac{k_2}{L_2}, \frac{k_3}{L_3}, \frac{k_4 + 1/2}{L_4} \right), \tag{10.20}$$

where k_μ is an integer in the range

$$k \in \left[-\frac{L_1}{2}, \frac{L_1}{2} - 1 \right] \times \left[-\frac{L_2}{2}, \frac{L_2}{2} - 1 \right] \times \left[-\frac{L_3}{2}, \frac{L_3}{2} - 1 \right] \times \left[-\frac{L_4}{2}, \frac{L_4}{2} - 1 \right]. \tag{10.21}$$

We can calculate $S(p)$ using the usual definition of the lattice propagator:

$$\sum_y D(x,y)G(y,z) = \mathbb{1}\delta_{xz}. \tag{10.22}$$

We use "momentum sources" $\mathbb{1}e^{ipz}$ to calculate "momentum propagators" $G(y|p)$ by

$$\sum_y D(x,y) \underbrace{\sum_z G(y,z)e^{ipz}}_{G(y|p)} = \mathbb{1}e^{ipx} \tag{10.23}$$

on every single configuration U. We can use these momentum propagators to evaluate the full propagator to

$$\begin{aligned} S(p) =& \frac{1}{V} \sum_{x,y} e^{-ip(x-y)} \langle q(x)\bar{q}(y)\rangle_{q,U} \\ =& \frac{1}{V} \sum_x e^{-ipx} \langle G(x|p)\rangle_U. \end{aligned} \tag{10.24}$$

10.5.2 Quark Renormalization Factor

As a second step, we use this propagator to calculate the quark renormalization factor (10.10) for CI fermions by [106]

$$\begin{aligned} Z_\psi^{\mathrm{RI'}}(\mu) =& \frac{1}{12} \operatorname{Tr} \left(S^{-1}(p) S_{\mathrm{Born}}(p) \right) \Big|_{p^2=\mu^2} \\ =& \frac{1}{12} \operatorname{Tr} \left(S^{-1}(p) \frac{\mathbb{1}R(p) - i\gamma_\mu a_\mu(p)}{R(p)^2 + \sum_\mu a_\mu(p)^2} \right) \Bigg|_{p^2=\mu^2}. \end{aligned} \tag{10.25}$$

On an unit gauge configuration, where $S^{-1}(p) = D_{\mathrm{CI}}^{\mathrm{free}}(p) + m$, we can simplify (10.25) to

$$Z_{\psi;U=\mathbb{1}}^{\mathrm{RI'}}(\mu) = 1 + m \left. \frac{R(p)}{R(p)^2 + \sum_\mu a_\mu(p)^2} \right|_{p^2=\mu^2} \tag{10.26}$$

Crosschecking this analytic result with the actual lattice calculation is certainly a good idea.

10.5.3 Operator Green Function

As a third step, we calculate the momentum space Green function of the operator $\mathcal{O}(z)$ which we wish to renormalize:

$$C_{\mathcal{O}}(p) = \sum_{x,y} e^{-ip(x-y)} \langle q(x) \left(\frac{1}{V} \sum_z \mathcal{O}(z) \right) \bar{q}(y) \rangle_{q,U} \tag{10.27}$$

In general one may rewrite operators like

$$\mathcal{O}^{(q)}_{\mu_1 \cdots \mu_n}(x) = \bar{q}(x) \gamma_{\mu_1} D_{\mu_2} \cdots D_{\mu_n} q(x) \tag{10.28}$$

as

$$\mathcal{O}_q(\tau) = \sum_{\vec{y}} \mathcal{O}_q(\vec{y}, \tau) = \sum_{\vec{y},v,w} \bar{q}^a_\alpha(v) O^{ab}_{\alpha\beta}(v, w; \vec{y}, \tau) q^b_\beta(w). \tag{10.29}$$

This way we may evaluate

$$\begin{aligned}
C_{\mathcal{O}}(p)^{ab}_{\alpha\beta} =& \frac{1}{V} \sum_{x,y,z,v,w} \langle q^a_\alpha(x) \bar{q}^c_\gamma(v) O^{cd}_{\gamma\delta}(v,w,z) q^d_\delta(w) \bar{q}^b_\beta(y) \rangle e^{-ip(x-y)} \\
=& \frac{1}{V} \sum_{x,y,z,v,w} O^{cd}_{\gamma\delta}(v,w,z) \left\{ S^{ac}_{\alpha\gamma}(x-v) S^{db}_{\delta\beta}(w-y) - S^{dc}_{\delta\gamma}(w-v) S^{ab}_{\alpha\beta}(x-y) \right\} e^{-ip(x-y)} \\
=& \frac{1}{V} \sum_{x,y,z,v,w,p_1,p_2} O^{cd}_{\gamma\delta}(v,w,z) \Big\{ e^{ip_1(x-v)+ip_2(w-y)} S^{ac}_{\alpha\gamma}(p_1) S^{db}_{\delta\beta}(p_2) \\
& \qquad - e^{ip_1(w-v)+ip_2(x-y)} S^{dc}_{\delta\gamma}(p_1) S^{ab}_{\alpha\beta}(p_2) \Big\} e^{-ip(x-y)} \\
=& \frac{1}{V} \sum_{v,w,z} O^{cd}_{\gamma\delta}(v,w,z) \left\{ S^{ac}_{\alpha\gamma}(p) S^{db}_{\delta\beta}(p) e^{-ip(v-w)} - S^{ab}_{\alpha\beta}(p) V \sum_{p'} S^{dc}_{\delta\gamma}(p') e^{-ip'(v-w)} \right\} \\
=& (S(p) O(p) S(p))^{ab}_{\alpha\beta} - S(p)^{ab}_{\alpha\beta} V \sum_{p'} \mathrm{Tr}_{\mathrm{DC}}(S(p') O(p)).
\end{aligned} \tag{10.30}$$

where we have defined $O(p) = \frac{1}{V} \sum_z O(p,z) = \frac{1}{V} \sum_{v,w,z} e^{-ip(v-w)} O(v,w,z)$. From the above expression we see that we have a connected and a disconnected part. This is illustrated in figure 10.1. We are interested in the renormalization of non-singlet quantities $\langle \mathcal{O} \rangle_{u-d}$ only. In such a case we obtain

$$\begin{aligned}
C_{\mathcal{O}} &\propto \langle u \left(\bar{u} \mathcal{O} u - d \mathcal{O} d \right) \bar{u} \rangle \\
&= \overbrace{u \bar{u}}\, O \, \overbrace{u \bar{u}} + \overbrace{u \overbrace{\bar{u} O u} \bar{u}} - \overbrace{u \overbrace{\bar{d} O d} \bar{u}} \\
&= \overbrace{u \bar{u}}\, O \, \overbrace{u \bar{u}}
\end{aligned} \tag{10.31}$$

and find that the disconnected part cancels like for the non-singlet three-point functions we calculated in section 8.16. For the connected part we can also write

$$\begin{aligned}
C_{\mathcal{O}}(p) &= \sum_{x,y,v,w} e^{-ip(x-y)} \langle G(x,v) O(v,w) G(w,y) \rangle_U \\
&= \sum_{v,w} \left\langle \gamma_5 \left(\sum_x G(v,x) e^{ipx} \right)^{\dagger_{\mathrm{DC}}} \gamma_5 O(v,w) \left(\sum_y G(w,y) e^{ipy} \right) \right\rangle_U \\
&= \sum_{v,w} \left\langle \gamma_5 G(v|p)^{\dagger_{\mathrm{DC}}} \gamma_5 O(v,w) G(w|p) \right\rangle_U
\end{aligned} \tag{10.32}$$

by reshuffling phase factors and exploiting γ_5-hermiticity.

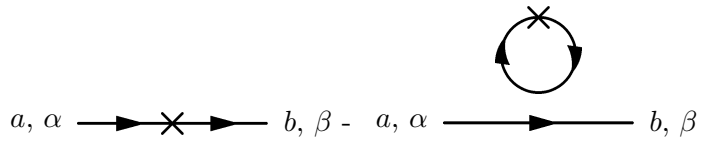


Figure 10.1: Diagrams contributing to renormalization.

10.5.4 Operator Renormalization Factor

As a fourth step we calculate the Dirac-color-inverse of the momentum space propagator to obtain the vertex function $\Lambda_{\mathcal{O}}$ of the operator Green function C

$$\Lambda_{\mathcal{O}}(p) = S^{-1}(p) C_{\mathcal{O}}(p) S^{-1}(p). \tag{10.33}$$

Finally we fix the operator's renormalization constant using the RI'-MOM scheme definition

$$Z_{\mathcal{O}}^{\mathrm{RI'}}(\mu) = \left. \frac{12 Z_{\psi}^{\mathrm{RI'}}}{\mathrm{Tr}\left(\Lambda_{\mathcal{O}}(p)\left(\Lambda_{\mathcal{O}}^{\mathrm{Born}}(p)\right)^{-1}\right)} \right|_{p^2=\mu^2} \tag{10.34}$$

at the scale of the external momenta. For the Born term we obtain

$$\Lambda_{\mathcal{O}}^{\mathrm{Born}}(p) = S_{\mathrm{free}}^{-1}(p) C_{\mathcal{O}}^{\mathrm{Born}}(p) S_{\mathrm{free}}^{-1}(p) = O_{\mathrm{Born}}(p) = \sum_{v,w} O_{U=\mathbb{1}}(v,w) e^{-ip(v-w)} \tag{10.35}$$

In the most simple case of $\mathcal{O}_{\Gamma}^{(q)}(x) = \bar{q}(x)\Gamma q(x)$ we obtain for the correlator:

$$C_{\mathcal{O}}(p) = \frac{1}{V} \sum_{y} \left\langle \gamma_5 G(y|p)^{\dagger_{\mathrm{DC}}} \gamma_5 \Gamma G(y|p) \right\rangle_U . \tag{10.36}$$

For its corresponding Born term we obtain

$$\begin{aligned} \Lambda_{\mathcal{O}}^{\mathrm{Born}}(p) &= \sum_{v,w} O_{U=\mathbb{1}}(v,w) e^{-ip(v-w)} \\ &= \sum_{v,w} e^{-ip(v-w)} \frac{1}{V} \Gamma \delta_{v,w} \mathbb{1}_{\mathrm{C}} = \Gamma \mathbb{1}_{\mathrm{C}} \end{aligned} \tag{10.37}$$

Now we are finally in a position to actually evaluate first $Z_{\psi}^{\mathrm{RI'}}(\mu)$ and then $Z_{\mathcal{O}}^{\mathrm{RI'}}(\mu)$ for every momentum and quark mass available.

10.6 The Reduced RI'-MOM Schemes

There is a modification of the RI'-schemes that tries to reduce discretization artifacts and thus to improve the continuum extrapolation of its renormalization factors. It has been introduced in [107] and is going to be called Reduced RI'-MOM (RRI'-MOM) scheme in this work. This scheme defines a subtraction of the diagonal part of a matrix M by

$$\overline{M} = M - \mathbb{1}_D \frac{1}{4} \mathrm{Tr}_D M \tag{10.38}$$

Written more explicitly using index-notation:

$$\overline{M}_{\alpha\beta}^{ab} = M_{\alpha\beta}^{ab} - \delta_{\alpha\beta} \frac{1}{4} \sum_{\gamma} M_{\gamma\gamma}^{ab} \tag{10.39}$$

The reduced schemes RRI'-MOM prescribe to calculate the renormalization factors using reduced propagators $\overline{S}$ throughout:

$$Z_\psi^{\text{RRI'}}(\mu) = \frac{1}{12}\,\text{Tr}\left(\overline{S}^{-1}(p)\overline{S}_{\text{Born}}(p)\right)\Big|_{p^2=\mu^2}. \tag{10.40}$$

$$\Lambda_{\mathcal{O}}^{\text{RRI'}} = \overline{S}^{-1} C_{\mathcal{O}}^{\text{RRI'}} \overline{S}^{-1}, \tag{10.41}$$

where the operator Green function $C_{\mathcal{O}}^{\text{RRI'}}$ is also determined from reduced propagators:

$$\begin{aligned} C_{\mathcal{O}}^{\text{RRI'}}(p) &= \sum_{x,y,v,w} e^{-ip(x-y)} \left\langle \overline{G}(x,v) O(v,w) \overline{G}(w,y) \right\rangle_U \\ &= \sum_{v,w} \left\langle \gamma_5 \left(\sum_x \overline{G}(v,x) e^{ipx} \right)^{\dagger_{\text{DC}}} \gamma_5 O(v,w) \left(\sum_y \overline{G}(w,y) e^{ipy} \right) \right\rangle_U \\ &= \sum_{v,w} \left\langle \gamma_5 \overline{G}(v|p)^{\dagger_{\text{DC}}} \gamma_5 O(v,w) \overline{G}(w|p) \right\rangle_U \end{aligned} \tag{10.42}$$

For clarity we add that the reduction needs to be done for any color combination G^{ab} separately.

10.7 Conversion to the Modified Minimal Subtraction Scheme

Generally operators have to be renormalized by

$$\mathcal{O}^{\mathcal{S}}(M) = Z^{\mathcal{S}}_{\mathcal{O};\text{bare}}(M) \mathcal{O}_{\text{bare}} \tag{10.43}$$

using a scheme $\mathcal{S}$ at some scale M, where the running of the renormalization factor with the scale is controlled by its anomalous dimension

$$\gamma_{\mathcal{O}}^{\mathcal{S}}(g^{\mathcal{S}}(M)) = \left.\frac{\partial \ln Z_{\mathcal{O}}^{\mathcal{S}}(M)}{\partial \ln M}\right|_{\text{bare}} = -\gamma_{\mathcal{O};0}(g^{\mathcal{S}}(M))^2 - \gamma^{\mathcal{S}}_{\mathcal{O};1}(g^{\mathcal{S}}(M))^4 - \gamma^{\mathcal{S}}_{\mathcal{O};2}(g^{\mathcal{S}}(M))^6 - \dots \tag{10.44}$$

This function is given perturbatively as a power series in the coupling constant, where the first coefficient $\gamma_{\mathcal{O};0}$ is scheme independent. The β function controls the running of the coupling constant at the scale M by

$$\beta^{\mathcal{S}}(g^{\mathcal{S}}(M))) = \left.\frac{\partial g^{\mathcal{S}}(M)}{\partial \ln M}\right|_{\text{bare}} = -\beta_0 (g^{\mathcal{S}}(M))^3 - \beta_1 (g^{\mathcal{S}}(M)^5) - \beta_2^{\mathcal{S}} (g^{\mathcal{S}}(M))^7 - \dots \tag{10.45}$$

Again a perturbative expansion gives a power series in the coupling constant, where the first two coefficients β_0 and β_1 are independent of the renormalization scheme employed. We are going to use only a single continuum scheme in this work. It is the most prominent one, which we introduced earlier: $\overline{\text{MS}}$. Comparing different intermediate schemes can deliver important information about possible systematic errors within the lattice renormalization procedure. We postpone this to later studies.

In the $\overline{\text{MS}}$ scheme where $(\mathcal{S}, M) = (\overline{\text{MS}}, \mu)$ the coefficients are given by [108]

$$\begin{aligned}
\beta_0 &= \frac{1}{(4\pi)^2}\left(11 - \frac{2}{3}n_f\right), \\
\beta_1 &= \frac{1}{(4\pi)^4}\left(102 - \frac{38}{3}n_f\right), \\
\beta_2 &= \frac{1}{(4\pi)^6}\left(\frac{2857}{2} - \frac{5033}{18}n_f - \frac{325}{54}n_f^2\right), \\
\beta_3 &= \frac{1}{(4\pi)^8}\left\{\left(\frac{149753}{6} + 3564\zeta_3\right) - \left(\frac{1078361}{162} + \frac{6508}{27}\zeta_3\right)n_f \right. \\
&\qquad \left. + \left(\frac{50065}{162} + \frac{6472}{81}\zeta_3\right)n_f^2 + \frac{1093}{729}n_f^3\right\}.
\end{aligned} \tag{10.46}$$

We may integrate the series expansion (10.45) to obtain the perturbative part of the coupling $g^{\mathcal{S}}(M)$:

$$\frac{M}{\Lambda^{\mathcal{S}}} = \exp\left(\frac{1}{2\beta_0 g^{\mathcal{S}}(M)^2}\right)\left(\beta_0 g^{\mathcal{S}}(M)^2\right)^{\frac{\beta_1}{2\beta_0^2}} \exp\left\{\int_0^{g^{\mathcal{S}}(M)} d\xi \left(\frac{1}{\beta^{\mathcal{S}}(\xi)} + \frac{1}{\beta_0 \xi^3} - \frac{\beta_1}{\beta_0^2 \xi}\right)\right\}. \tag{10.47}$$

Here $\Lambda^{\mathcal{S}}$ is the scheme specific scale, an integration constant. This implicit formula for $g^{\mathcal{S}}(M)$ can be solved numerically. We plot the couplings $\alpha = g^2/4\pi$ in the scheme $\overline{\text{MS}}$ for the known orders in perturbation theory in figure 10.2 and their dependence on the number of flavors in figure 10.3.

If we multiply the results of the bare matrix elements from section 8.11 with the RI' renormalization factors from the last section we get renormalized matrix elements that are valid in the RI' scheme. We need to convert them to the $\overline{\text{MS}}$-scheme to render them easily comparable with available results and experiments. We use the notation

type	operator	ren. factor
quark	-	Z_q
scalar	$\mathcal{O}_S = \bar{q}\mathbb{1}q$	Z_A
vector	$\mathcal{O}_V = \bar{q}\gamma_\mu q$	Z_V
tensor	$\mathcal{O}_T = \bar{q}\sigma_{\mu\nu}q$	Z_T
axial-vector	$\mathcal{O}_A = \bar{q}\gamma_\mu\gamma_5 q$	Z_A
pseudo-scalar	$\mathcal{O}_P = \bar{q}\gamma_5 q$	Z_P

It was found from perturbative calculations in [109], [110] and [111] that in Landau gauge, using the $\overline{\text{MS}}$ coupling $g = g^{\overline{\text{MS}}}$, the following relations of conversion factors hold:

$$\begin{aligned}
\frac{Z_P^{\overline{\text{MS}}}}{Z_P^{\text{RI'}}} = \frac{Z_S^{\overline{\text{MS}}}}{Z_S^{\text{RI'}}} &= 1 + \frac{16}{3}\frac{g^2}{16\pi^2} - \left(\frac{4291}{18} - \frac{152}{3}\zeta_3 - \frac{83}{9}n_f\right)\left(\frac{g^2}{16\pi^2}\right)^2 \\
&\quad + \left\{\frac{3696847}{324} - \frac{224993}{54}\zeta_3 + \frac{2960}{9}\zeta_5 + \left(-\frac{241294}{243} + \frac{4720}{27}\zeta_3 - \frac{80}{3}\zeta_4\right)n_f \right. \\
&\quad \left. + \left(\frac{7514}{729} + \frac{32}{27}\zeta_3\right)n_f^2\right\}\left(\frac{g^2}{16\pi^2}\right)^3 + O(g^8)
\end{aligned} \tag{10.48}$$

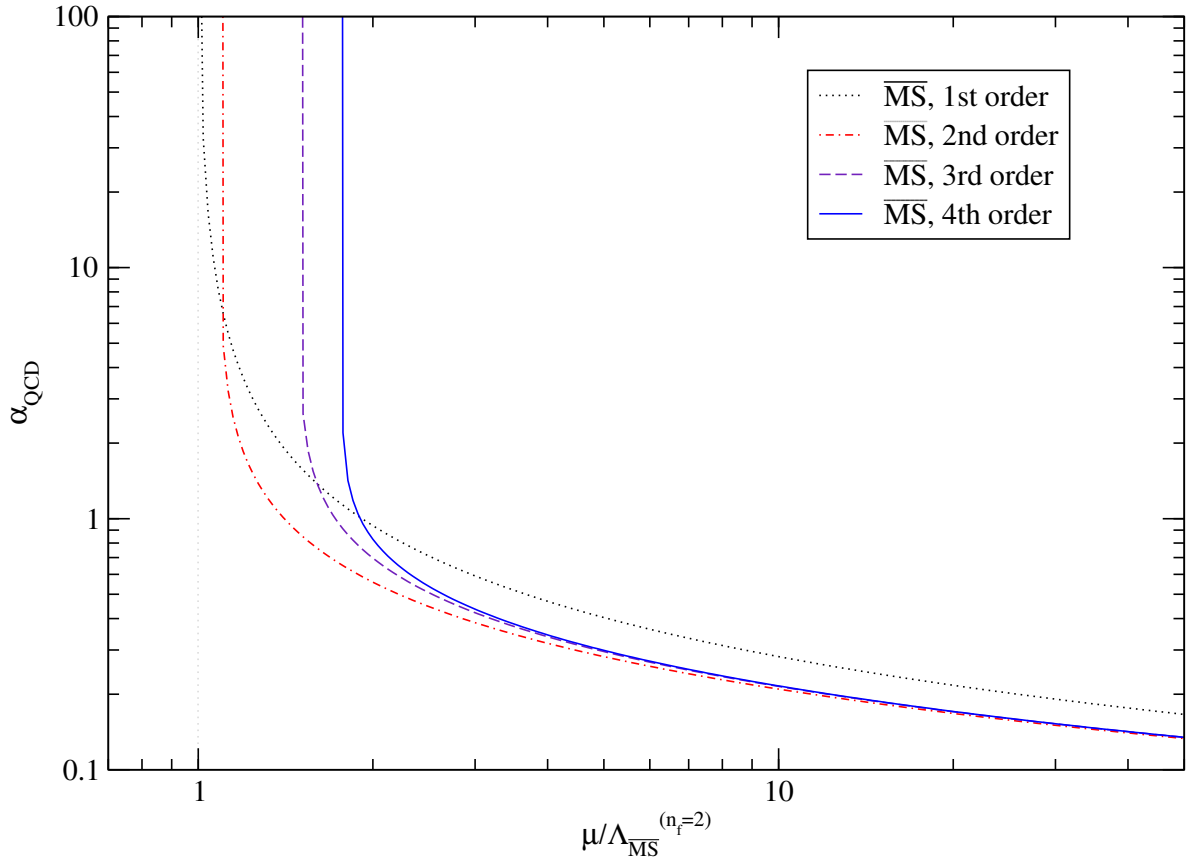


Figure 10.2: Coupling 'constants' $\alpha_{\rm QCD}$ at various orders in the $\overline{\rm MS}$ (using $g^{\overline{\rm MS}}$) scheme at $n_f = 2$.

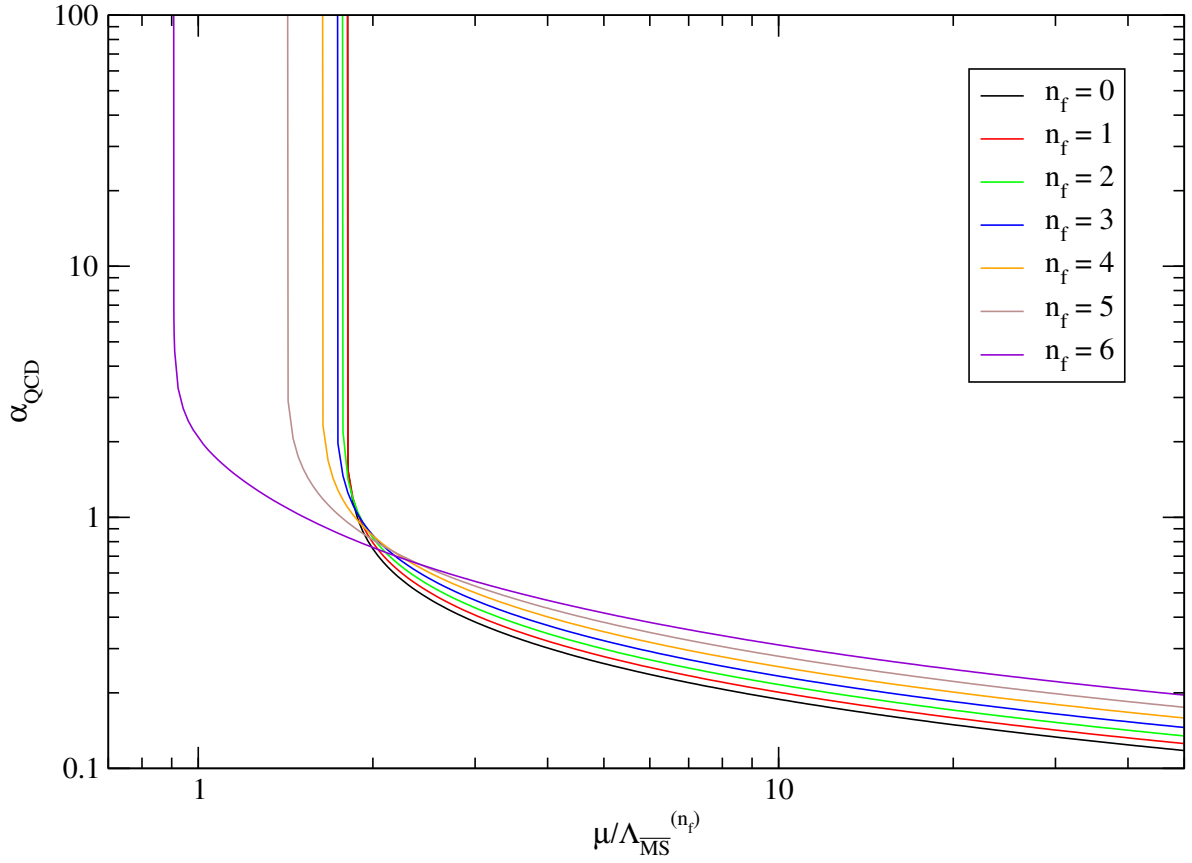


Figure 10.3: Coupling 'constants' $\alpha_{\rm QCD}$ in the $\overline{\rm MS}$ scheme at various numbers of flavors (4th order).

$$\begin{aligned}\frac{Z_T^{\overline{\text{MS}}}}{Z_T^{\text{RI'}}} = 1 &+ \left(-\frac{3847}{54} + \frac{184}{9}\zeta_3 + \frac{313}{81}n_f\right)\left(\frac{g^2}{16\pi^2}\right)^2 \\ &+ \left\{-\frac{9858659}{2916} + \frac{678473}{486}\zeta_3 + \frac{1072}{81}\zeta_4 - \frac{10040}{27}\zeta_5\right. \\ &\quad + \left(-\frac{2096}{27}\zeta_3 + \frac{80}{9}\zeta_4 + \frac{286262}{729}\right)n_f \\ &\quad \left. + \left(-\frac{32}{81}\zeta_3 - \frac{13754}{2187}\right)n_f^2\right\}\left(\frac{g^2}{16\pi^2}\right)^3 + \mathcal{O}(g^8)\end{aligned} \tag{10.49}$$

$$\begin{aligned}\frac{Z_V^{\overline{\text{MS}}}}{Z_V^{\text{RI'}}} = \frac{Z_A^{\overline{\text{MS}}}}{Z_A^{\text{RI'}}} = 1 &+ \left(-\frac{67}{6} + \frac{2}{3}n_f\right)\left(\frac{g^2}{16\pi^2}\right)^2 \\ &+ \left\{\frac{-52321}{72} + \frac{607}{4}\zeta_3 + \left(\frac{2236}{27} - 8\zeta_3\right)n_f - \frac{52}{27}n_f^2\right\}\left(\frac{g^2}{16\pi^2}\right)^3 + \mathcal{O}(g^8)\end{aligned} \tag{10.50}$$

$$\begin{aligned}\frac{Z_q^{\overline{\text{MS}}}}{Z_q^{\text{RI'}}} = 1 &+ \left(-\frac{359}{9} + 12\zeta_3 + \frac{7}{3}n_f\right)\left(\frac{g^2}{16\pi^2}\right)^2 + \left\{-\frac{439543}{162} + \frac{8009}{6}\zeta_3 + \frac{79}{4}\zeta_4\right. \\ &\left. - \frac{1165}{3}\zeta_5 + \left(\frac{24722}{81} - \frac{440}{9}\zeta_3\right)n_f - \frac{1570}{243}n_f^2\right\}\left(\frac{g^2}{16\pi^2}\right)^3 + \mathcal{O}(g^8)\end{aligned} \tag{10.51}$$

10.8 Renormalization Group Invariant Operators

We now know the dependence of the renormalization factors on the scale in the $\overline{\text{MS}}$ and MOM schemes. Due to the inherent problems of the lattice discretization causing finite volume effects and discretization errors, the common procedure is to compensate the dependence on the scale in order to see if the renormalization factors evolve as expected [112]. This is done by defining an Renormalization Group Invariant (RGI) operator for each operator under inspection. They are designed such that any dependence on the scale that is known from perturbation theory is canceled. Therefore the RGI curves should be flat at perturbative scales. Looking for a plateau there gives the correct RGI value. Converting this factor back to $\overline{\text{MS}}$ at the desired scale, usually 2 GeV, gives a improved value for the renormalization factor in the desired scheme (see equation (10.56)):

$$\mathcal{O}^{\overline{\text{MS}}}(2\,\text{GeV}) = \left(\Delta Z_{\mathcal{O}}^{\overline{\text{MS}}}(2\,\text{GeV})\right)^{-1}\mathcal{O}^{\text{RGI}} \tag{10.52}$$

This method removes some systematical uncertainties in the extraction of a plateau region. To be able to estimate the errors introduced by these perturbative matchings we use the procedure to calculate RGI operators using a second intermediate scheme. We use the MOM scheme with the $\widetilde{\text{MOM}}$gg coupling [113].

Defining the RGI operators is done by looking at the conversion factor between two schemes $(\mathcal{S}, M) \to (\mathcal{S}', M')$:

$$\mathcal{O}^{\mathcal{S}'}(M') = Z_{\mathcal{O}}^{\mathcal{S}\to\mathcal{S}'}(M', M)\mathcal{O}^{\mathcal{S}}(M) \equiv \frac{Z_{\mathcal{O};\text{bare}}^{\mathcal{S}'}(M')}{Z_{\mathcal{O};\text{bare}}^{\mathcal{S}}(M)}\mathcal{O}^{\mathcal{S}}(M). \tag{10.53}$$

One can set up the two anomalous dimension equations for this conversion factor $Z^{\mathcal{S}\to\mathcal{S}'}$, obtained by either differentiating with respect to $\ln M'$ or $\ln M$. Reintegrating these it is possible to factor (see for example [112])

$$Z_{\mathcal{O}}^{\mathcal{S}\to\mathcal{S}'}(M',M) = \frac{\Delta Z_{\mathcal{O}}^{\mathcal{S}}(M)}{\Delta Z_{\mathcal{O}}^{\mathcal{S}'}(M')}, \tag{10.54}$$

where we have defined

$$\Delta Z_{\mathcal{O}}^{\mathcal{S}}(M) = \left(2\beta_0 g^{\mathcal{S}}(M)^2\right)^{-\gamma_{\mathcal{O};0}/2\beta_0} \exp\left\{-\int_0^{g^{\mathcal{S}}(M)} d\xi \left(\frac{\gamma_{\mathcal{O}}^{\mathcal{S}}(\xi)}{\beta^{\mathcal{S}}(\xi)} - \frac{\gamma_{\mathcal{O};0}}{\beta_0\xi}\right)\right\}. \tag{10.55}$$

Plugging (10.54) into (10.53), we see that we can create an scheme and scale independent operator from an operator in the scheme $\mathcal{S}$ at the scale M by

$$\mathcal{O}^{\mathrm{RGI}} = \Delta Z_{\mathcal{O}}^{\mathcal{S}}(M)\mathcal{O}^{\mathcal{S}}(M) \equiv Z_{\mathcal{O}}^{\mathrm{RGI}}\mathcal{O}_{\mathrm{bare}}. \tag{10.56}$$

The only input necessary are the coefficients of the β function and the coefficients of the γ function of the operator under inspection (each in the right scheme $\mathcal{S}$). For the $\overline{\mathrm{MS}}$ scheme the coefficients are listed below. For the scalar and pseudo-scalar operators S and P we have [114]:

$$\begin{aligned}
\gamma_0 &= \frac{1}{(4\pi)^2}\left(-8\right),\\
\gamma_1 &= \frac{1}{(4\pi)^4}\left(-\frac{404}{3}+\frac{40}{9}n_f\right),\\
\gamma_2 &= \frac{1}{(4\pi)^6}\left(\frac{344}{9}-\frac{22826}{9}+\left(-\frac{92}{9}+\frac{4708}{27}+\frac{320}{3}\zeta_3\right)n_f+\frac{280}{81}n_f^2\right),\\
\gamma_3 &= \frac{1}{(4\pi)^8}\left(-\frac{4603055}{81}-\frac{271360}{27}\zeta_3+17600\zeta_5+\left(\frac{183446}{27}+\frac{68384}{9}\zeta_3-1760\zeta_4-\frac{36800}{9}\zeta_5\right)n_f\right.\\
&\qquad\left.+\left(-\frac{10484}{243}-\frac{1600}{9}\zeta_3+\frac{320}{3}\zeta_4\right)n_f^2+\left(+\frac{664}{243}-\frac{128}{27}\zeta_3\right)n_f^3\right).
\end{aligned} \tag{10.57}$$

The vector and axial-vector coefficients are of course zero: $\gamma \equiv 0$, while for the tensor operator, the following results have been obtained [115]:

$$\begin{aligned}
\gamma_0 &= \frac{1}{(4\pi)^2}\left(\frac{8}{3}\right),\\
\gamma_1 &= \frac{1}{(4\pi)^4}\left(\frac{724}{9}-\frac{104}{27}n_f\right),\\
\gamma_2 &= \frac{1}{(4\pi)^6}\left(\frac{105110}{81}-\frac{1856}{27}\zeta_3-\left(\frac{10480}{81}+3209\zeta_3\right)n_f-\frac{8}{9}n_f^2\right).
\end{aligned} \tag{10.58}$$

For the quark field the anomalous coefficients are [109]:

$$
\begin{aligned}
\gamma_0 &= 0, \\
\gamma_1 &= \frac{1}{(4\pi)^4}\left(\frac{134}{3} - \frac{8}{3}n_f\right), \\
\gamma_2 &= \frac{1}{(4\pi)^6}\left(\frac{20729}{18} - 79\zeta_3 - \frac{1100}{9}n_f + \frac{40}{27}n_f^2\right), \\
\gamma_3 &= \frac{1}{(4\pi)^8}\left(\frac{2109389}{81} - \frac{565939}{162}\zeta_3 + \frac{2607}{2}\zeta_4 - \frac{761525}{648}\zeta_5\right. \\
&\qquad + \left(-\frac{324206}{81} - \frac{4582}{27}\zeta_3 - 79\zeta_4 - \frac{320}{3}\zeta_5\right)n_f \\
&\qquad \left. + \left(\frac{7706}{81} + \frac{320}{9}\zeta_3\right)n_f^2 + \frac{280}{243}n_f^3\right).
\end{aligned}
\tag{10.59}
$$

Chapter 11

Renormalization Results

In this chapter we present the parameters used for the calculation our renormalization factors, evaluate a number of different schemes and show chiral extrapolations leading to our final renormalization factors.

11.1 Our Parameters

We calculated renormalization factors for 7 dynamical $16^3 \times 32$ lattices of $n_f = 2$ at various

lattice	β	am_0	a/fm	a/GeV^{-1}	am_{AWI}	m_{AWI}/MeV	N_{cfg}	$N_{\mathrm{cfg}}^{\mathrm{renorm}}$
A50	4.70	-0.050	0.15032(112)	0.7619(57)	0.03027(8)	39.73(10)	200	10
A66	4.70	-0.066	0.13487(95)	0.6835(48)	0.00589(40)	8.62(57)	200	10
B60	4.65	-0.060	0.14980(89)	0.7591(45)	0.02356(13)	31.04(16)	325	8
B70	4.65	-0.070	0.14055(104)	0.7123(53)	0.00836(23)	11.74(32)	200	5
C64	4.58	-0.064	0.15831(127)	0.8023(64)	0.02995(20)	37.33(25)	200	11
C72	4.58	-0.072	0.15051(115)	0.7627(58)	0.01728(16)	22.65(21)	200	11
C77	4.58	-0.077	0.14487(95)	0.7342(48)	0.01054(19)	14.35(26)	300	10

Table 11.1: Parameters of our dynamical CI lattices

lattice	β	am_0	a/fm	a/GeV^{-1}
A	4.70	-0.06987	0.1311(15)	0.6644(76)
B	4.65	-0.07576	0.1352(21)	0.6851(106)
C	4.58	-0.08377	0.1380(17)	0.6993(86)

Table 11.2: Parameters of our chirally extrapolated dynamical CI lattices

lattice spacings and quark masses using the Lüscher-Weisz gauge action and the CI Dirac operator. For their parameters, see table 11.1 and for the data from chirally extrapolated lattices, table 11.2.

In these tables, lattice spacings have been calculated in [98] using the method proposed by Sommer [82, 83] where the Sommer scale was chosen to be $r_0 = 0.48\,\mathrm{fm}$. AWI-masses have been determined in [98].

In order to convert renormalization factors between schemes we need $\Lambda_{\overline{\mathrm{MS}}}$ (see formula 10.47). For these $n_f = 2$ configurations we used

$$\Lambda_{\overline{\mathrm{MS}}}^{n_f=2} = 243(24)\,\mathrm{MeV} \tag{11.1}$$

determined from $r_0 = 0.5\,\mathrm{fm}$ using the results of [116].

In order to learn about the scale dependence of the renormalization factors, we calculated the factors for the momenta listed in table 11.3. We tried to keep the momenta as diagonal as possible in oder to reduce discretization artifacts.

space diagonal	at	ax	ay	az	$(ap)^2$	μ^2/GeV^2
	2	0	0	2	0.7035948450	1.438594708
yes	2	1	1	1	0.7807011294	1.596248923
	2	1	1	2	0.8578074138	1.753903137
	3	0	1	1	0.9349136982	1.911557352
yes	3	1	1	1	1.166232551	2.384519996
	3	1	1	2	1.243338836	2.542174210
	3	1	2	2	1.397551404	2.857482640
yes	3	2	2	2	1.705976542	3.488099498
	3	2	2	3	1.860189111	3.803407927
yes	4	1	1	1	2.322826817	4.749333215
	4	1	1	2	2.554145670	5.222295858
yes	4	2	2	2	2.631251955	5.379950073
	5	1	1	3	2.862570808	5.852912717
	5	1	2	2	3.016783377	6.168221146
	5	1	2	3	3.093889661	6.325875361
	5	2	2	3	3.325208514	6.798838004
	6	1	2	2	3.787846220	7.744763292
yes	6	3	3	3	5.792609614	11.84377287
yes	7	4	4	4	9.570817549	19.56882939
yes	8	4	4	4	10.18766782	20.83006310
yes	9	5	5	5	15.04536374	30.76227862
yes	10	5	5	5	15.81642658	32.33882077

Table 11.3: Values of momenta used for obtaining renormalization factors at various scales. The values given in GeV are with respect to the lattice spacing of lattice C.

We investigated the lattice configurations and momenta used by the authors of [117] and were able to reproduce their results in a qualitative manner. Additionally, we

- employed higher statistics,
- used more quark masses,
- thus were able to perform a full chiral extrapolation
- did not use cubic spline interpolations for renormalization factors between scales, but a linear one instead.

We plot most of our results in the following sections. The data points have been connected by dashed lines to guide the eye.

When we calculate expectation values for matrix elements like in formula 10.2 we need to fix to a specific gauge. As mentioned earlier, we use the Landau gauge. At this point one can run into the problem of gauge fixing (Gribov) ambiguities. This has been addressed in [106, 118, 119, 120] for similar parameters, but no significant effects were found, therefore, no explicit check was performed in this work.

11.2 Evaluation of the Reduced RI'-MOM Scheme

Due to elementary symmetries of the continuum action we have conserved vector charge and conserved axial charge (Vector Ward Identity and Axial Ward Identity).

$$\mathcal{O}_V = j_V^\mu(x) = \psi(x)\gamma^\mu\bar{\psi}(x) \tag{11.2}$$

$$\mathcal{O}_A = j_A^\mu(x) = \psi(x)\gamma^\mu\gamma^5\bar{\psi}(x) \tag{11.3}$$

This implies, that both related Noether currents are protected against renormalization, and hence that their anomalous dimensions vanish. On the lattice we still have an exact vector Ward identity. The axial Ward identity is modified, but for Ginsparg Wilson type fermions it can be shown [121] that their anomalous dimension is still zero. For CI fermions we have an approximate Ginsparg Wilson symmetry, so we expect $Z_A \approx 1$, with a very mild scale dependence. In figures 11.1 and 11.2 we show a comparison of the scale dependence of the RI' renormalization factors of the axial and vector current operators j_A and j_V for the reduced and the original schemes, where we chose to determine the quark renormalization factor by the free CI reference propagator 10.18. A comparison of the schemes for the quark renormalization factors can be found in the next section.

However, as can been seen in figures 11.1 and 11.2, the renormalization factors of the

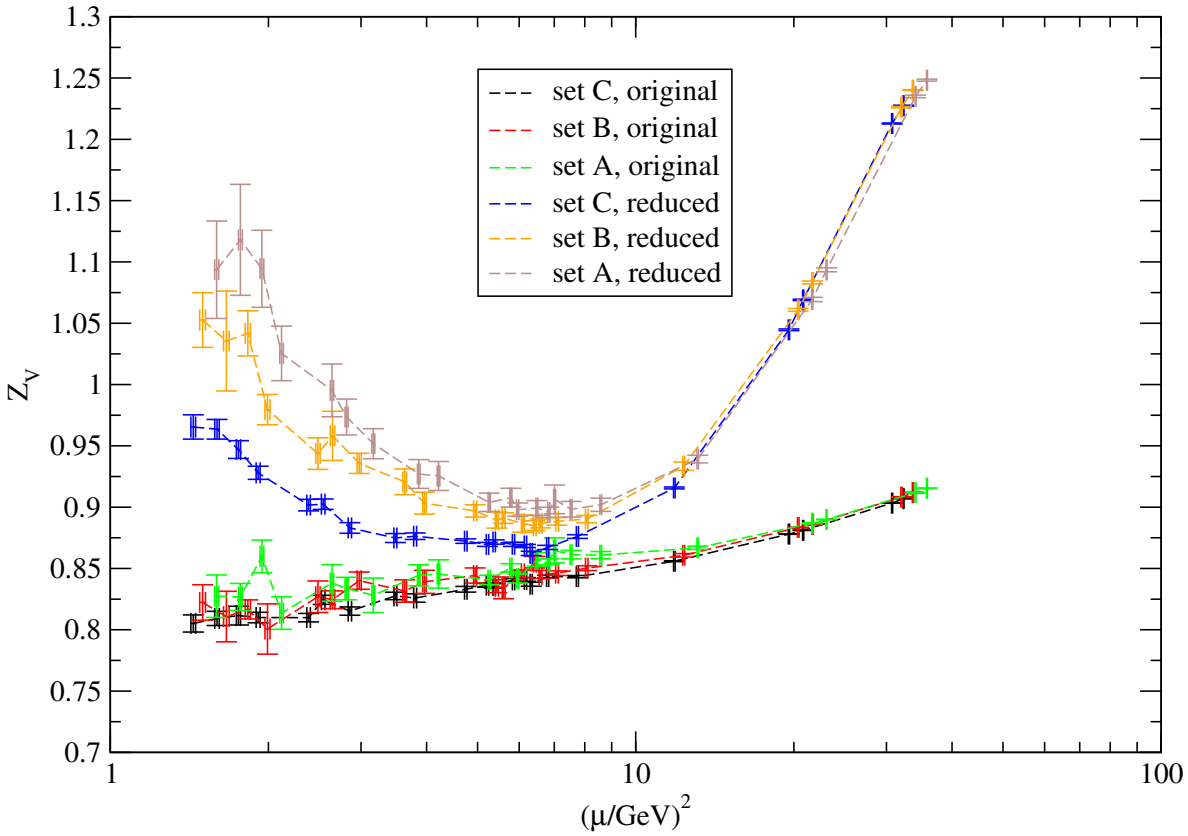


Figure 11.1: Comparison of the scale dependence of the chirally extrapolated RI' renormalization factor of the operator j_V for the reduced and the original schemes. Here

reduced schemes behave much worse than the original schemes with respect to our expectations in scale independence. We thus conclude that the propagator reduction 10.38 interferes heavily with the symmetries of the action: While this reduction was introduced to milden the continuum extrapolation, for each lattice spacing (or β) it worsens wanted good properties:

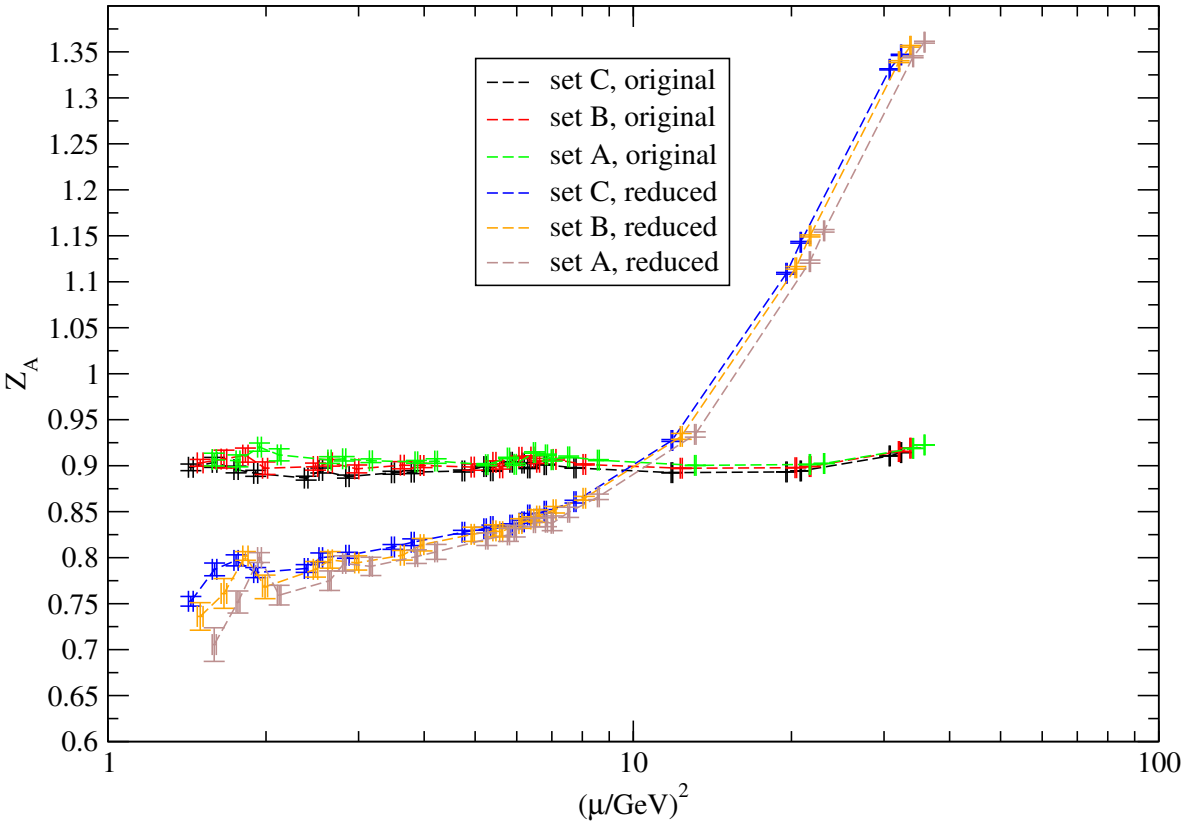


Figure 11.2: Comparison of the scale dependence of the chirally extrapolated RI' renormalization factor of the operator j_A for the reduced and the original schemes.

for example chiral symmetry. For this reason we abandon the reduced scheme at this point of this thesis and focus on the original schemes which keep the lattice-symmetries intact and thus simplify the extraction of a renormalization factor at a specific scale.

11.3 Evaluation of the Quark Renormalization Schemes

In section 10.4 we introduced the RI'-scheme in 4 different variants. All of them coincide in the continuum limit, but may differ heavily while at finite lattice spacing. Since any operator renormalization factor contains a factor of Z_ψ and since we want to have predictive data already at finite lattice spacing, it is significant to find the scheme with the mildest continuum extrapolation behavior.

$$\mathcal{O}^{\text{ren}} = \lim_{a\to 0} Z_{\mathcal{O}}^{\mathcal{S}_1}(a)\mathcal{O}^{\mathcal{S}_1}(a) = \lim_{a\to 0} Z_{\psi}^{\mathcal{S}_1}(a)\frac{12\mathcal{O}^{\mathcal{S}_1}(a)}{\text{Tr}\left(\Lambda_{\mathcal{O}}(a)\left(\Lambda_{\mathcal{O}}^{\text{Born}}\right)^{-1}\right)} = \lim_{a\to 0} Z^{\mathcal{S}_2}(a)\mathcal{O}^{\mathcal{S}_2}(a) \quad (11.4)$$

Since all factors involving $\mathcal{O}$ are independent of the quark renormalization scheme we find

$$\lim_{a\to 0} \frac{Z_\psi^{\mathcal{S}_1}(a)}{Z_\psi^{\mathcal{S}_2}(a)} = 1 \quad (11.5)$$

and are able to expand with respect to the lattice spacing around the continuum limit $a = 0$:

$$\frac{Z_\psi^{\mathcal{S}_1}(a)}{Z_\psi^{\mathcal{S}_2}(a)} = c_0 + c_1 a + c_2 a^2 + \mathcal{O}(a^3), \quad (11.6)$$

where $c_0 = 1$. In order to judge the continuum extrapolation a bit more quantitative we may fit the parameters c_i for example at a scale of $\mu = 2\,\mathrm{GeV}$. We plot the relevant data points in figure 11.3 but do not attempt such a fit since our lattice spacings are too close to each other to obtain a reasonable prediction at the size of the errors involved. Nevertheless we can make out a trend towards 1 for the ratios of the quark renormalization factors when $a \to 0$.

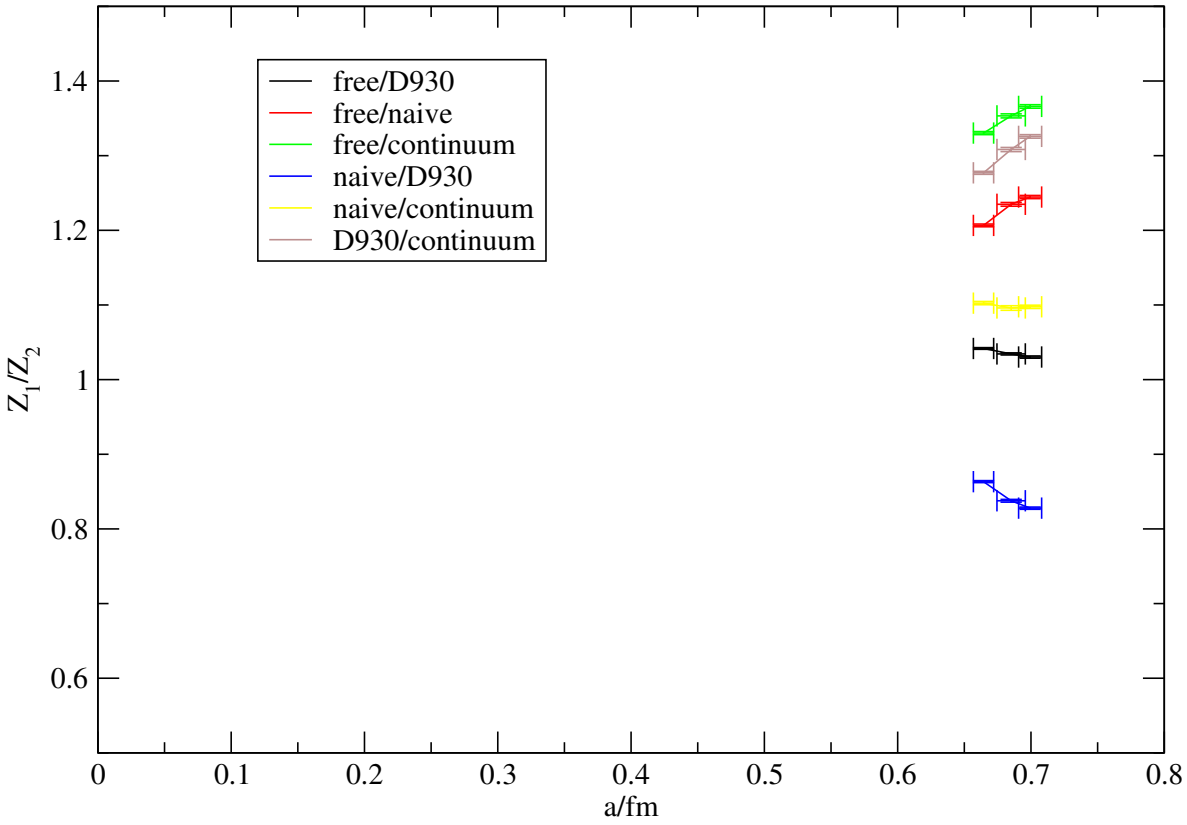
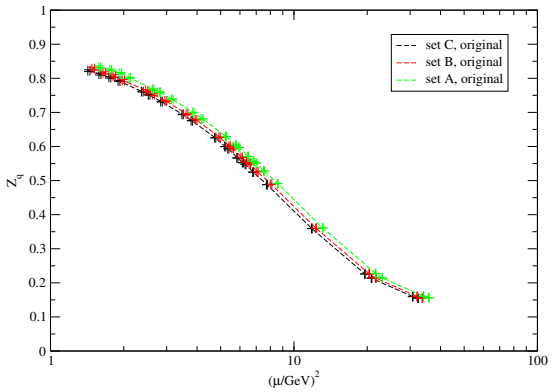
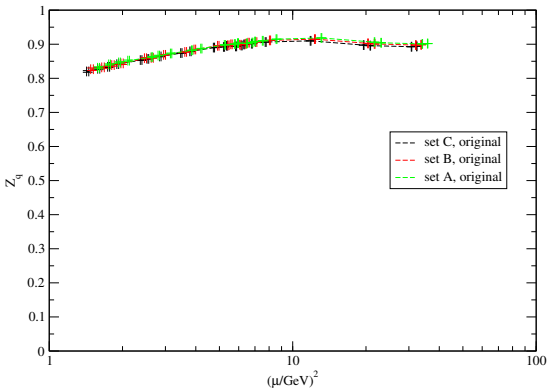


Figure 11.3: Ratios of quark renormalization factors plotted against the lattice spacing for the chirally extrapolated lattices A, B and C. With our 4 variants of quark renormalization factors goes a total of 6 interesting conversion factors which need to go to 1 in the continuum limit $a \to 0$, which is at the left end of the figure.

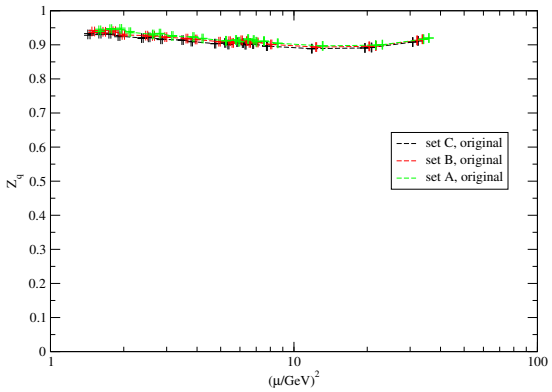
Since the method explained above does not provide any conclusions for our data, we now try to judge the quality of the data from a look at the scale dependence of the renormalization factors: For each lattice spacing available, we plot the scale dependence of the chirally extrapolated RI'-renormalization factors in figure 11.4. All except one show a rather strong dependence on the scale. When we convert them to the RGI scheme in order to see whether anomalous dimensions match the perturbative ones (see figure 11.5), we find a plateau for type c only. We are therefore led to believe, that type c is rather close to the continuum where the RI' $\to$ RGI renormalization factors are correct. For this reason we drop all other quark renormalization schemes at this point of this thesis.



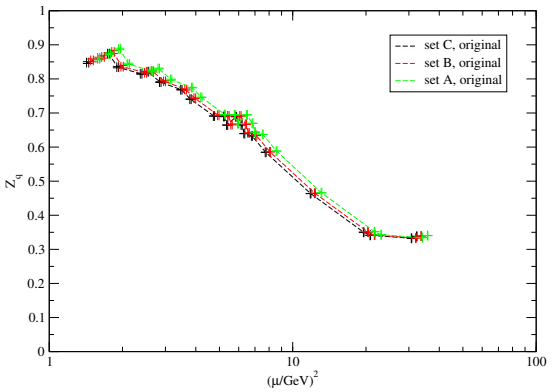
(a) The continuum propagator in 10.12.



(b) The free CI propagator in 10.16 with coefficients optimized for interacting quarks.

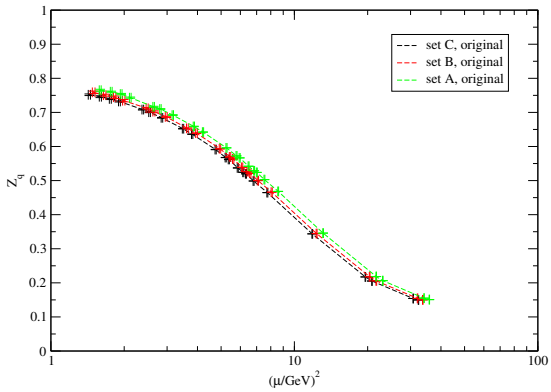


(c) The free CI propagator in 10.16 with coefficients optimized for free quarks.

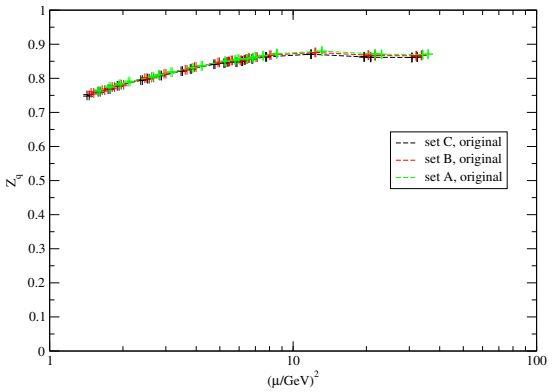


(d) The naive lattice propagator in 10.13.

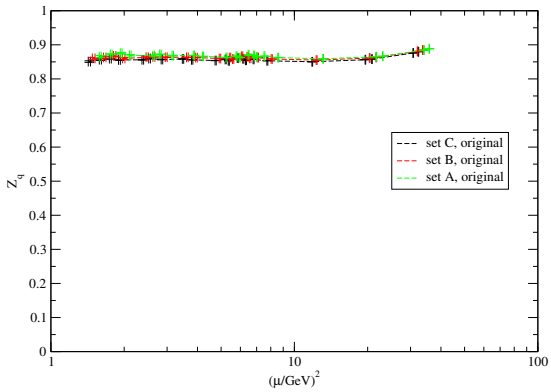
Figure 11.4: Scale dependence of the chirally extrapolated RI' quark renormalization factor determined from a comparison of ...



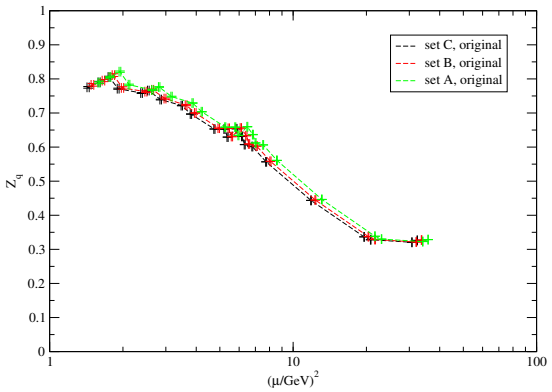
(a) The continuum propagator in 10.12.



(b) The free CI propagator in 10.16 with coefficients optimized for interacting quarks.



(c) The free CI propagator in 10.16 with coefficients optimized for free quarks.



(d) The naive lattice propagator in 10.13.

Figure 11.5: Scale dependence of the chirally extrapolated RGI quark renormalization factor determined from a comparison of ...

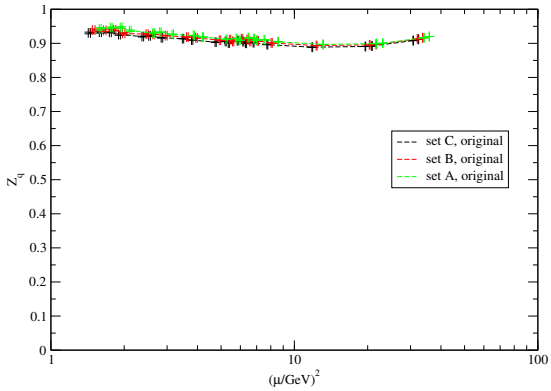
11.4 Results for Different Schemes

In this subsection we show results for the chirally extrapolated renormalization factors in the RI', RGI and $\overline{\text{MS}}$ schemes for all simple currents with respect to the scale at our three different lattice spacings. Since our lattice spacings are pretty close to each other, so are the renormalization factors. Figure 11.6 shows results for the RI' scheme. More interesting is the scaling behavior in the RGI scheme in figure 11.7, as we can make out regions of the scale μ where discretization effects are small and the perturbative expansion becomes valid. This is the case, where RGI curves become flat such that a window as in formula 10.3 is obvious. The renormalization factors for the quark, scalar, axial-vector and pseudo-scalar show mild scale dependence in a region[1] around $\mu = 2\,\text{GeV}$, such that we can interpret these values as plateau values. Unfortunately vector and tensor show a strong scale dependence, such that no window is obvious and a reliable extraction of renormalization factors is hardly possible. Additionally we plot the ratios of axial-vector over vector and pseudo-scalar over scalar renormalization factors

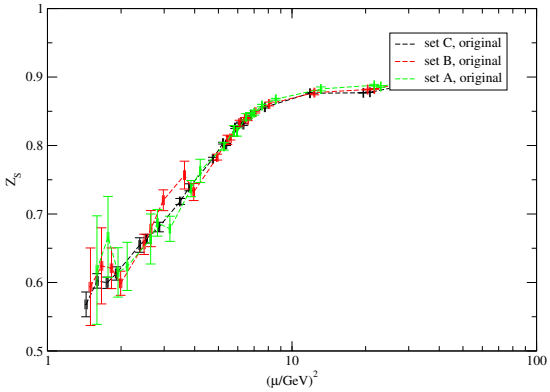
$$Z_{A/V} = \frac{Z_A}{Z_V}, \quad Z_{P/S} = \frac{Z_P}{Z_S} \tag{11.7}$$

For these ratios most discretization effects should cancel [44].

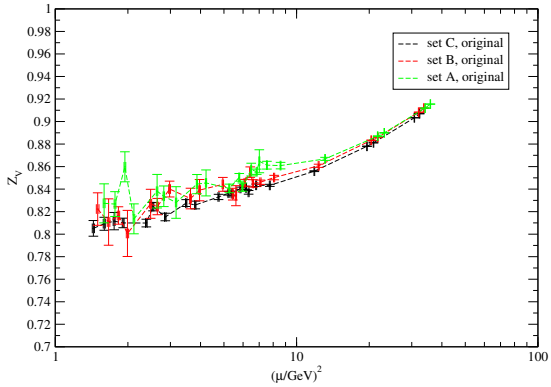
[1] A lucky accident.



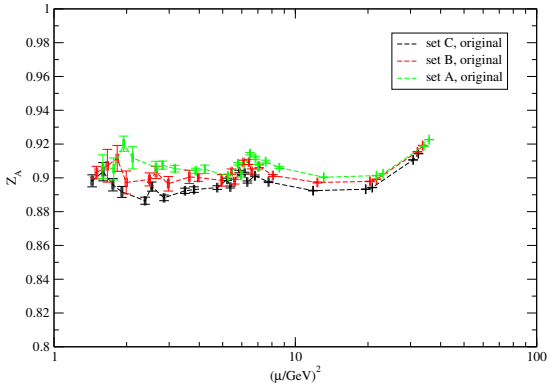
(a) The quark renormalization factor.



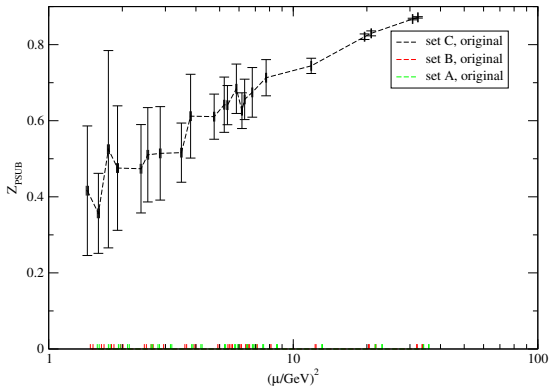
(b) The scalar renormalization factor.



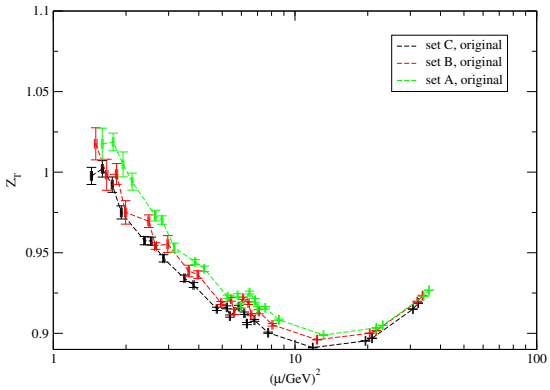
(c) The vector renormalization factor.



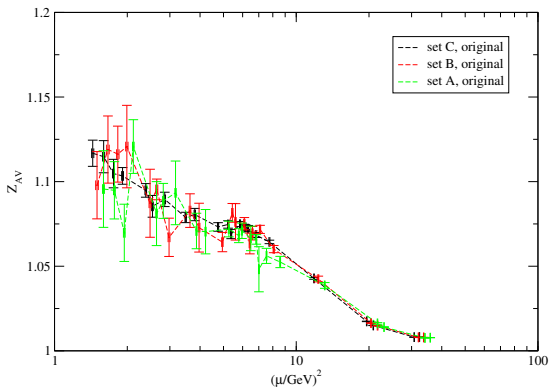
(d) The axial-vector renormalization factor.



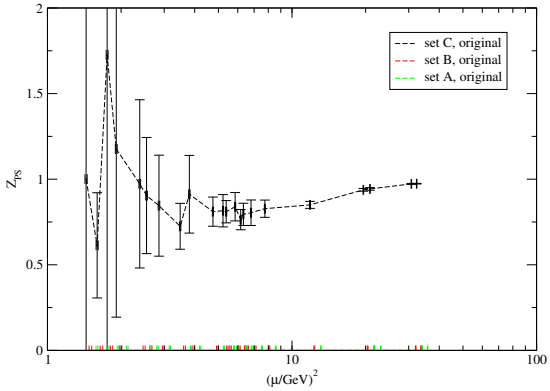
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.

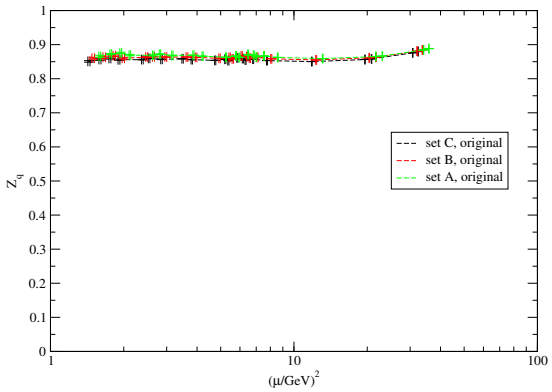


(g) The axial/vector renormalization factor.

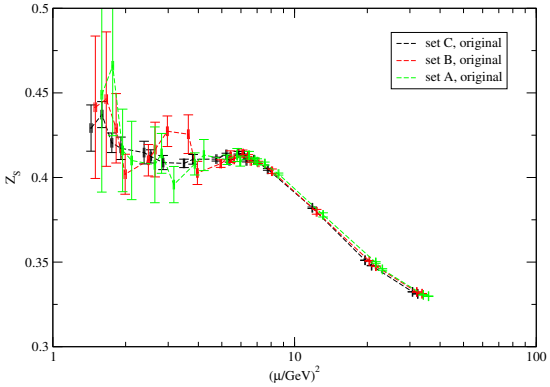


(h) The pseudo-scalar/scalar renormalization factor.

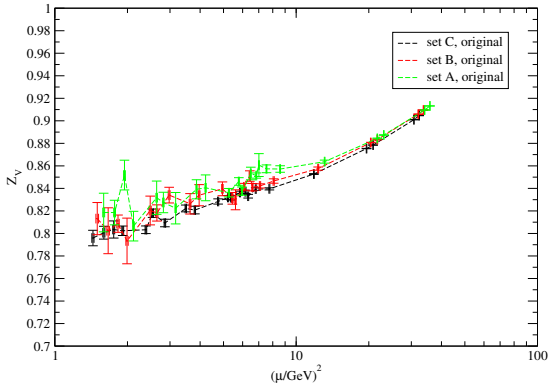
Figure 11.6: Scale dependence of various chirally extrapolated RI' renormalization factors. Plots differ in vertical scales.



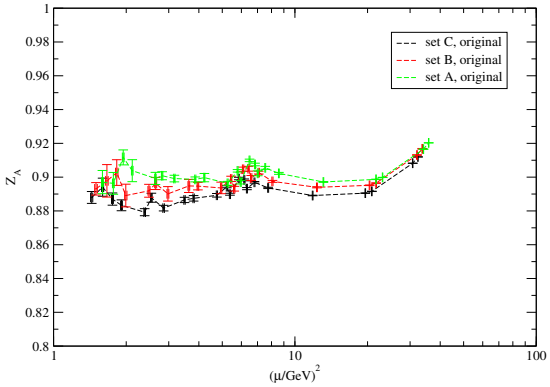
(a) The quark renormalization factor.



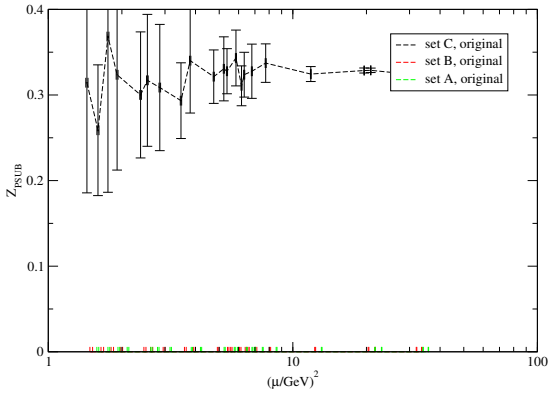
(b) The scalar renormalization factor.



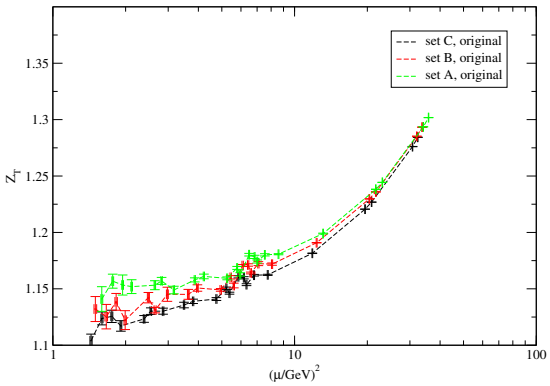
(c) The vector renormalization factor.



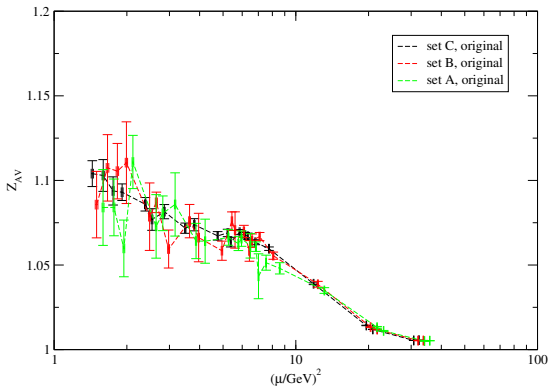
(d) The axial-vector renormalization factor.



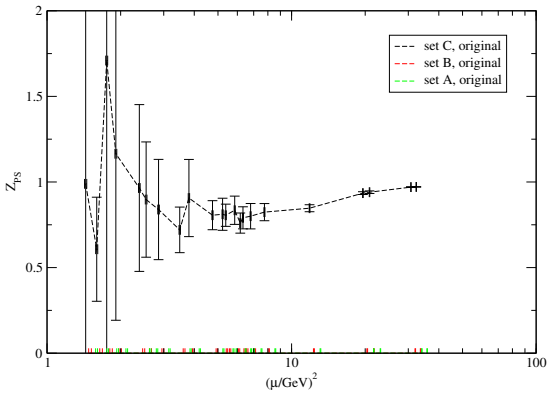
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.

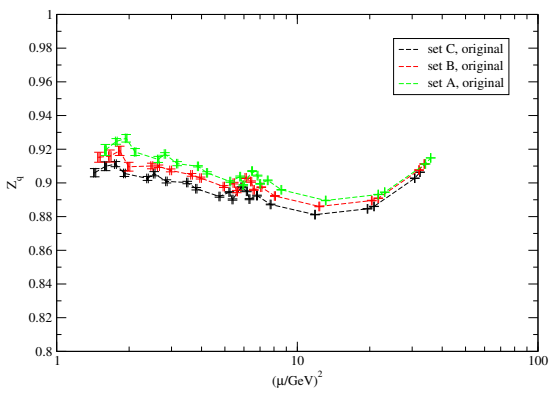


(g) The axial/vector renormalization factor.

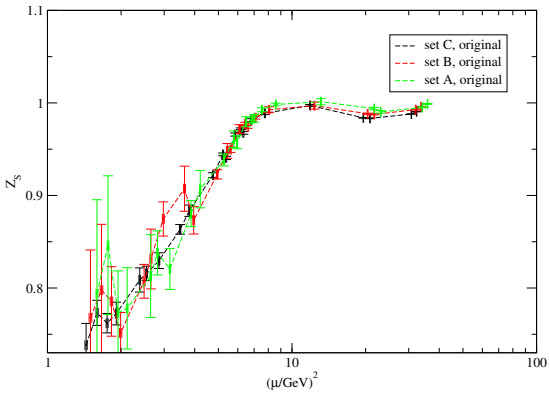


(h) The pseudo-scalar/scalar renormalization factor.

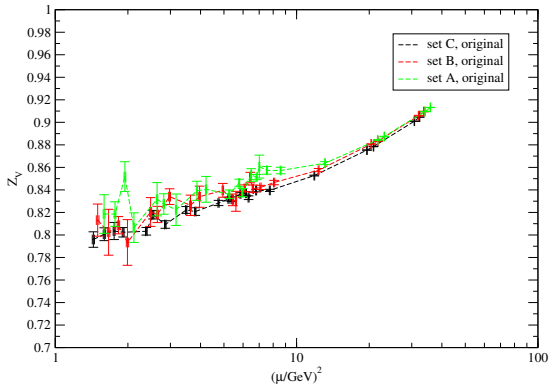
Figure 11.7: Scale dependence of various chirally extrapolated RGI renormalization factors. Plots differ in vertical scales.



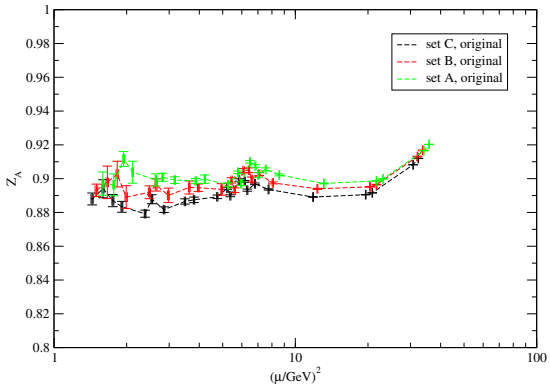
(a) The quark renormalization factor.



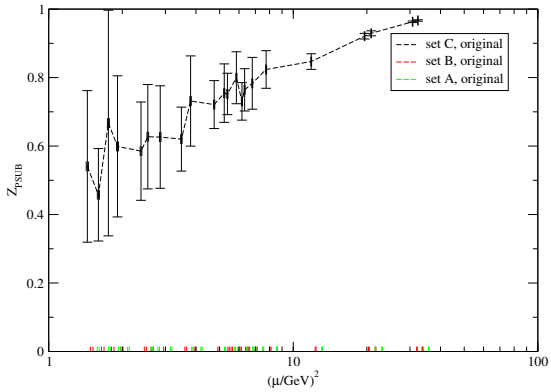
(b) The scalar renormalization factor.



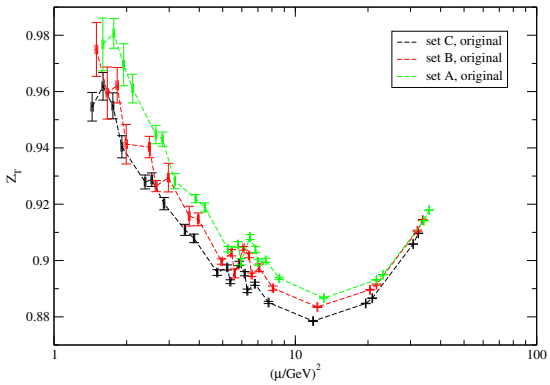
(c) The vector renormalization factor.



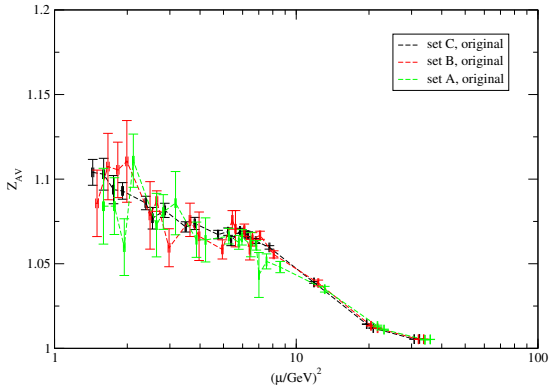
(d) The axial-vector renormalization factor.



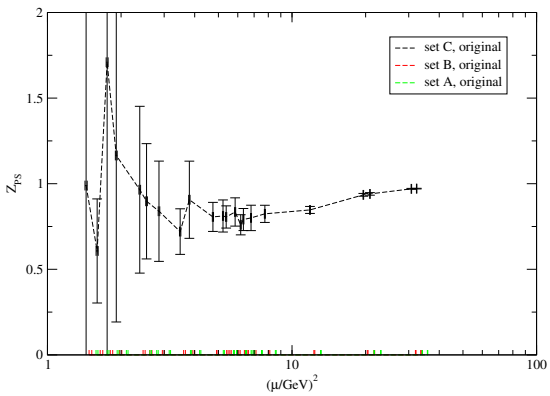
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.



(g) The axial/vector renormalization factor.



(h) The pseudo-scalar/scalar renormalization factor.

Figure 11.8: Scale dependence of various chirally extrapolated $\overline{\mathrm{MS}}$ renormalization factors. Plots differ in vertical scales.

11.5 Chiral Extrapolation of Renormalization Results

Although there is partially quenched data available for a number of set of lattice configurations, we only show fully dynamical data, i.e., set the valence quark mass equal to the sea quark mass. Extrapolating this quark mass down to physical values is not only interesting because of phenomenological aspects, but essential since all the scheme conversions employed hold for massless quarks only. For this reason we extrapolate quark masses down to zero within our RI' data set, then convert them to other schemes like RGI or $\overline{\text{MS}}$. For each available lattice spacing and scale μ we extrapolate the renormalization factors of any operator $\mathcal{O}$ in the AWI quark masses down to $m_{\text{AWI}} = 0$.

11.5.1 Simple Factor Extrapolations

All simple renormalization factors (except the pseudo-scalar) can be Taylor expanded around $m_{\text{AWI}} = 0$. At the quark masses employed a linear extrapolation is enough:

$$Z_{\mathcal{O}}(m) = Z_{\mathcal{O}}^{(0)} + Z_{\mathcal{O}}^{(1)} m + \mathcal{O}(m^2) \tag{11.8}$$

This is also true for the renormalization factor ratio $Z_{A/V}$, although, as obvious from later plots, a quadratic fit is necessary since linear terms cancel heavily for this specific ratio.

11.5.2 Pseudo-scalar Extrapolations

The pseudo-scalar density, however, couples to the Goldstone boson channel, so we expect $\mathcal{O}(m_{\text{AWI}}^{-1})$ contributions to the pseudo-scalar propagator in the chiral limit. As discussed in [103, 122, 123, 124, 125] the inverse renormalization factor can be expanded as

$$\frac{1}{Z_P} = \frac{A}{m} + \frac{1}{B} + Cm + \mathcal{O}(m^2). \tag{11.9}$$

So, if we subtract the pole contribution by defining

$$\frac{1}{Z_P^{\text{SUB}}} \equiv \frac{1}{Z_P} - \frac{A}{m} = \frac{1}{B} + Cm + \mathcal{O}(m^2), \tag{11.10}$$

this will extrapolate down for zero quark mass to

$$Z_P^{\text{SUB}}(0) = B. \tag{11.11}$$

This is also true for the renormalization factor ratio

$$Z_{P/S} = \frac{Z_P}{Z_S} = \frac{1}{Z_S^{(0)} + Z_S^{(1)} m + \mathcal{O}(m^2)} \frac{1}{\frac{A}{m} + \frac{1}{B} + Cm + \mathcal{O}(m^2)} \tag{11.12}$$

$$= \frac{1}{\frac{Z_S^{(0)} A}{m} + \left(\frac{Z_S^{(0)}}{B} + Z_S^{(1)} A \right) + \left(Z_S^{(0)} C + \frac{Z_S^{(1)}}{B} \right) m + \mathcal{O}(m^2)} \tag{11.13}$$

since an analytical function $Z_S(m)$ in the denominator does not change the expansion structure.

11.5.3 Fitting Procedure

We fit the parameters $Z_{\mathcal{O}}^{(0)}$ and $Z_{\mathcal{O}}^{(1)}$, or in the pseudo-scalar case(s) A, B and C, to our lattice results using the described functions. This can be easily done for lattice C, where there are 3 sea quark masses available. For lattices A and B we do only have 2 sea quark masses at our

disposal such that a determination of pseudo-scalar renormalization factor Z_P^{SUB} is impossible using this method. We estimate the errors of the fitted variables using the jacknife method.

We show plots of fits of such chiral extrapolations exemplary for lattice C (fully dynamical) in figure 11.11 and for lattice C77 (partially quenched) in figure 11.10. Further we show plots of results of such chiral extrapolations for lattice C in figure 11.12, for lattice B in figure 11.13 and for lattice A in figure 11.14.

In the pseudo-scalar cases, the operator product expansion ensures that pole contributions are suppressed for large values of μ. This feature can be checked in figures 11.9, where we plot the fit parameter $A\mu^2$ against the scale. As explained in [126], this coefficient is at one loop level expected to be

$$A\mu^2 = \frac{4\pi}{3}\alpha_S\langle q\bar{q}\rangle, \tag{11.14}$$

In one-loop partially quenched chiral perturbation theory the condensate includes a logarithmic term that leads to a divergence in the chiral limit [127]. With only 4 valence quark masses available for most of our partially quenched lattices, such an additional logarithmic term cannot be disentangled from the $1/m$ pole term, so we ignore it. Coefficient A shows a rather universal behavior at low scales with an estimated uncertainty of a few percent for $\langle q\bar{q}\rangle$, which indicates a consistent pion-pole removal over the whole range of lattice configurations. At a scale of 2 GeV we are just at the border of validity. At higher scales discretization effects in the data points obviously invalidate the extrapolation function used for the fit.

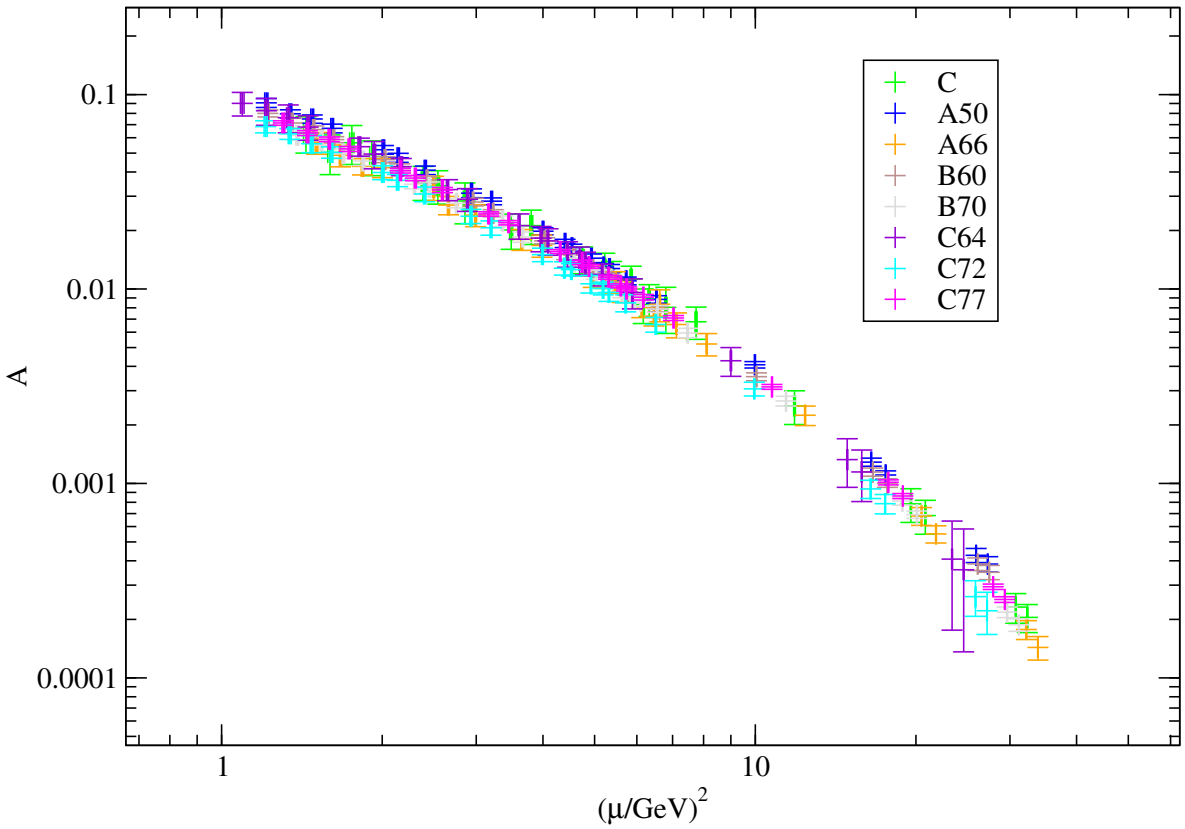
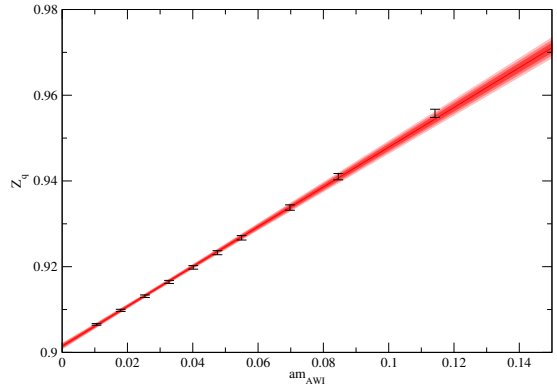
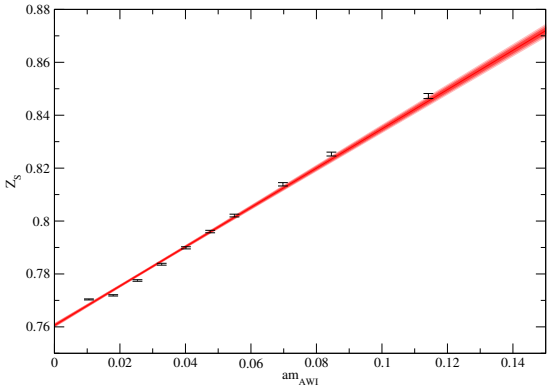


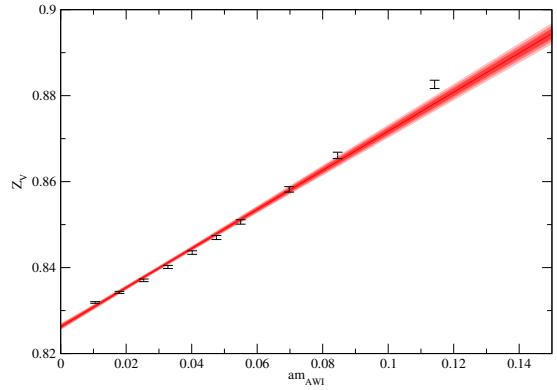
Figure 11.9: Pseudo-scalar fit parameter $A\mu^2$ against the scale for all lattices where a fit was possible.



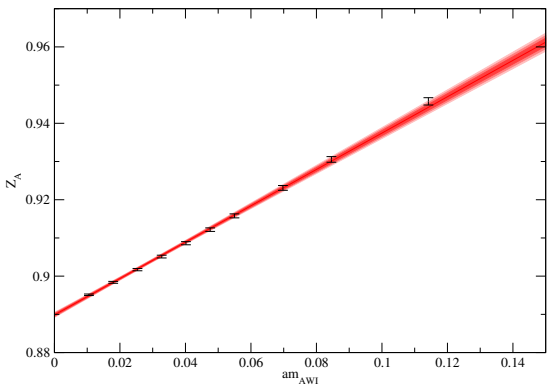
(a) the quark renormalization factor.



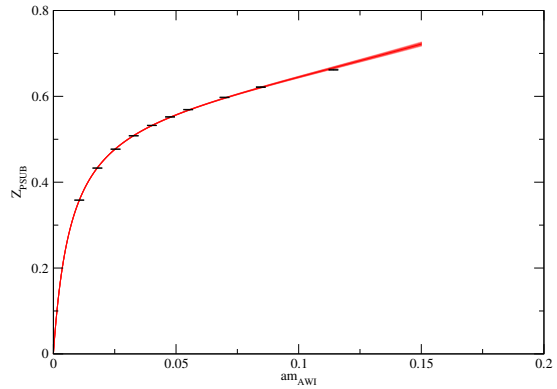
(b) the scalar renormalization factor.



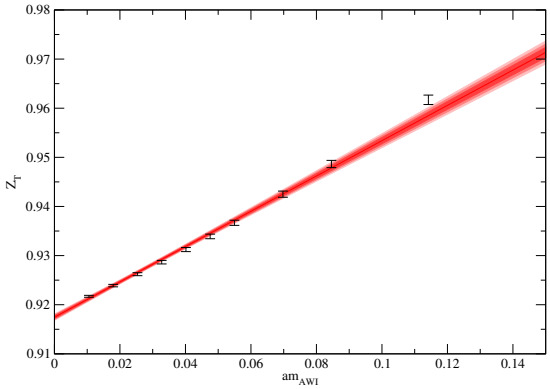
(c) the vector renormalization factor.



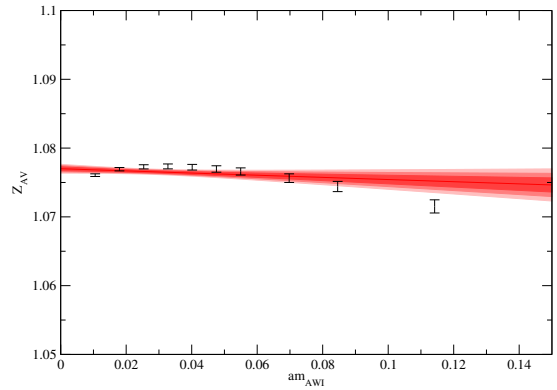
(d) the axial-vector renormalization factor.



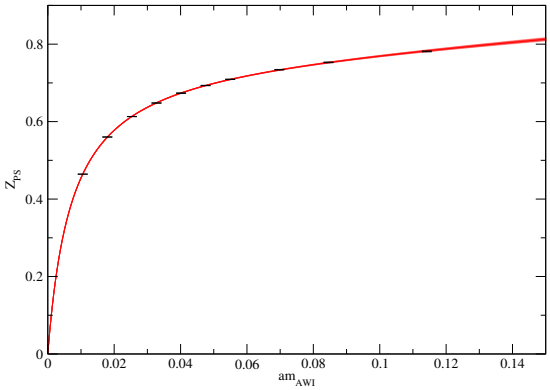
(e) the pseudo-scalar renormalization factor.



(f) the tensor renormalization factor.

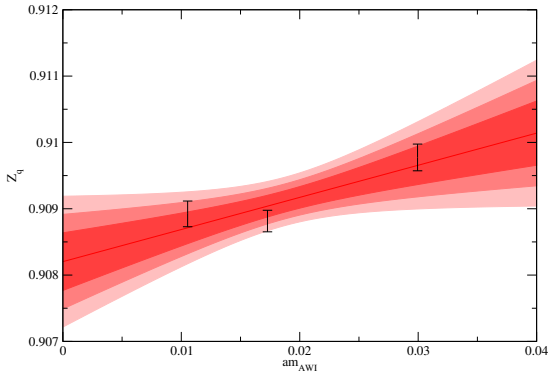


(g) the axial-vector/vector renormalization factor.

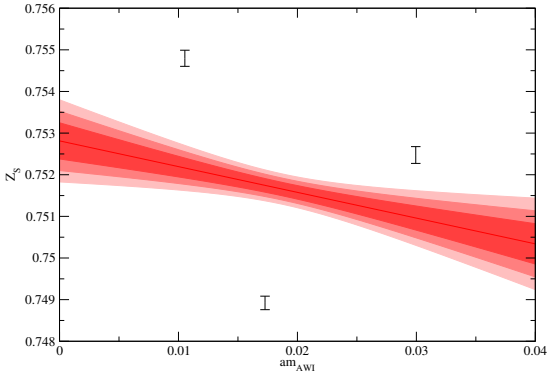


(h) the pseudo-scalar/scalar renormalization factor.

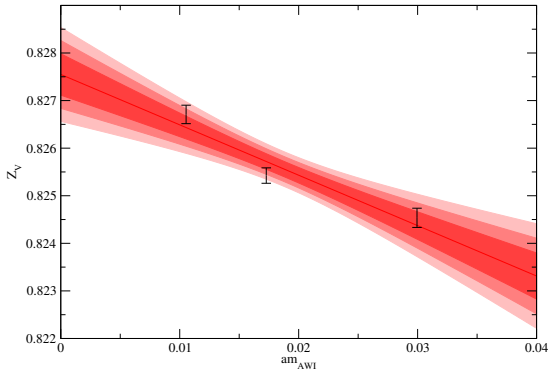
Figure 11.10: Chiral extrapolations of RI' renormalization factors at a scale of $\mu = 2\,\mathrm{GeV}$ for results on lattice C77 ($\beta = 4.58$, partially quenched) of various renormalization factors. Filled bands represent 1σ, 2σ and 3σ confidence bands. Plots differ in vertical scales.



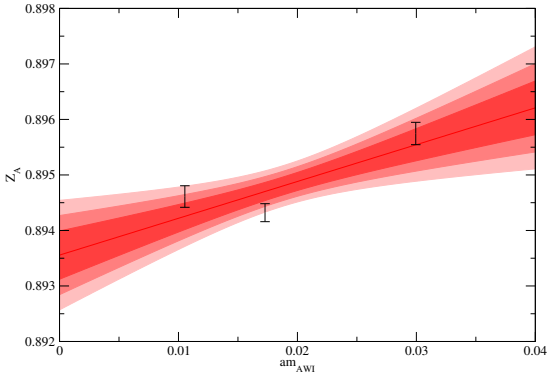
(a) the quark renormalization factor.



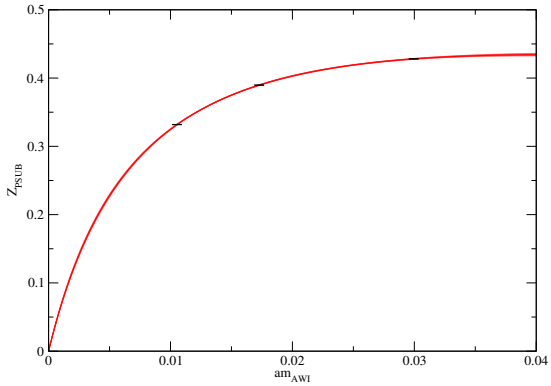
(b) the scalar renormalization factor.



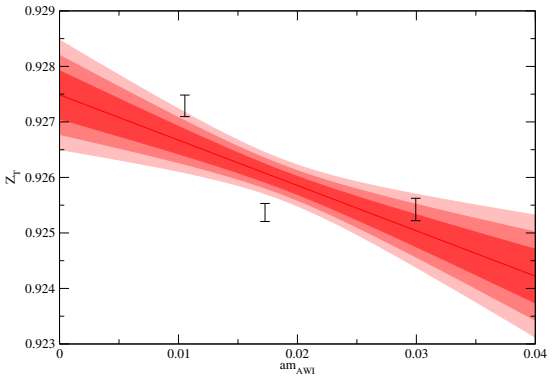
(c) the vector renormalization factor.



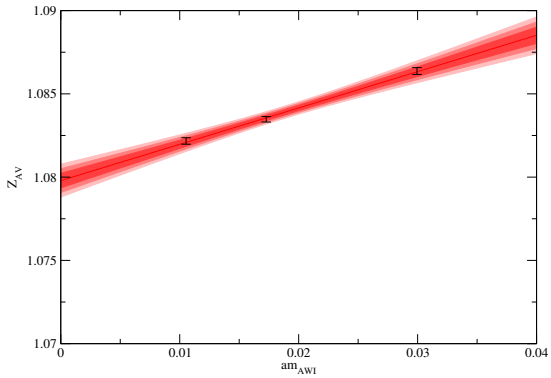
(d) the axial-vector renormalization factor.



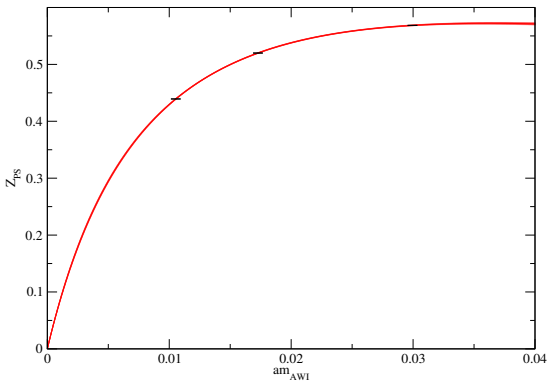
(e) the pseudo-scalar renormalization factor.



(f) the tensor renormalization factor.

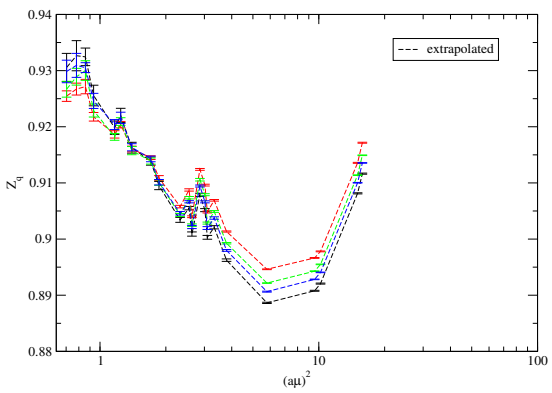


(g) the axial-vector/vector renormalization factor.

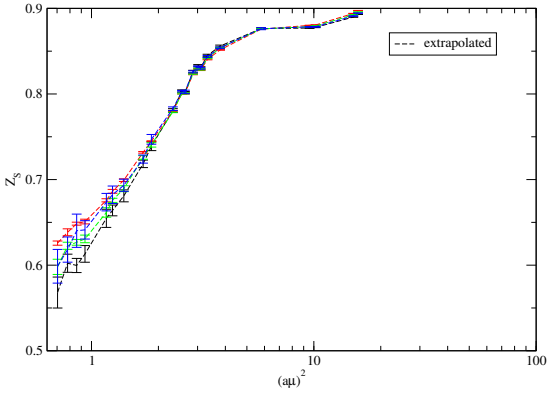


(h) the pseudo-scalar/scalar renormalization factor.

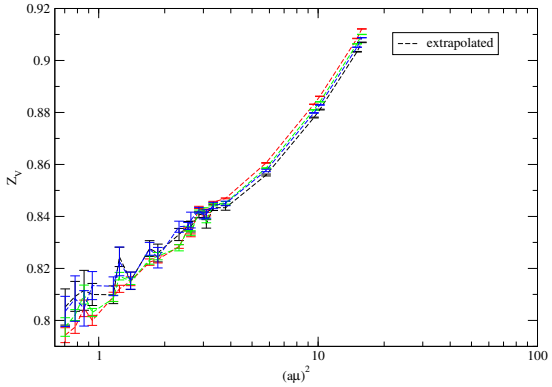
Figure 11.11: Chiral extrapolations of RI' renormalization factors at a scale of $\mu = 2\,\text{GeV}$ for results on lattice C ($\beta = 4.58$) of various renormalization factors. Filled bands represent 1σ, 2σ and 3σ confidence bands. Plots differ in vertical scales.



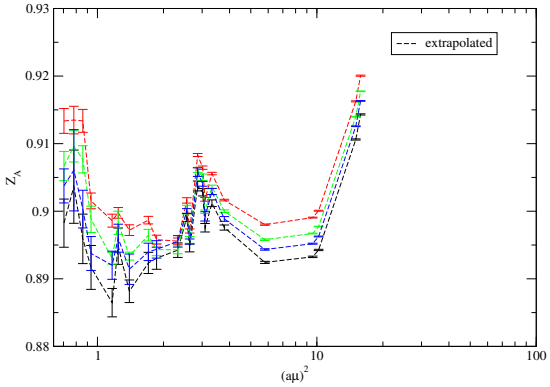
(a) The quark renormalization factor.



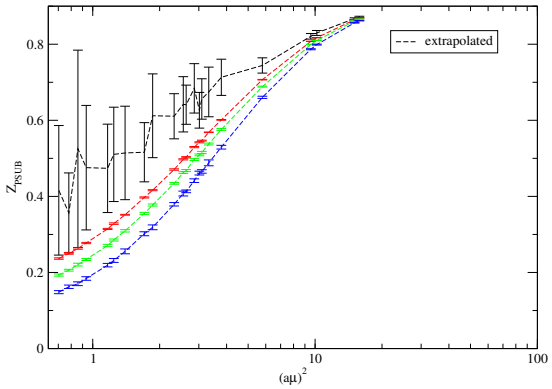
(b) The scalar renormalization factor.



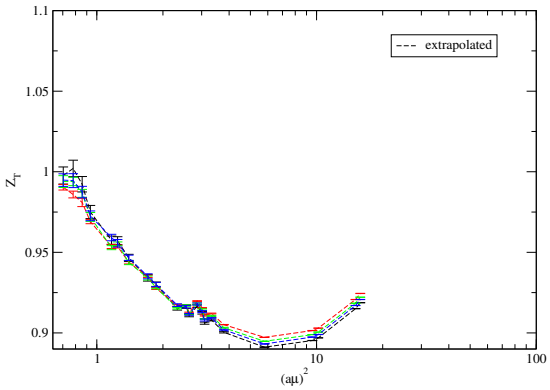
(c) The vector renormalization factor.



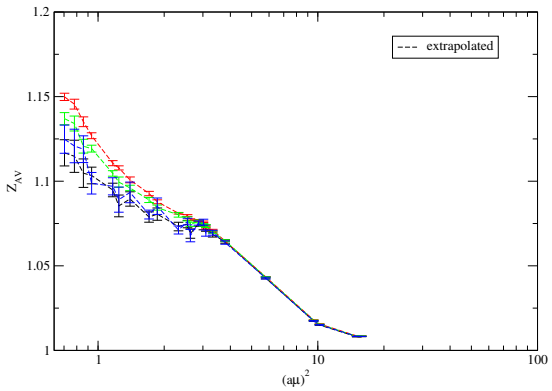
(d) The axial-vector renormalization factor.



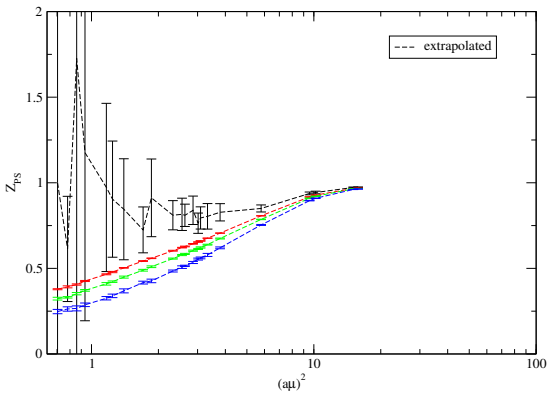
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.

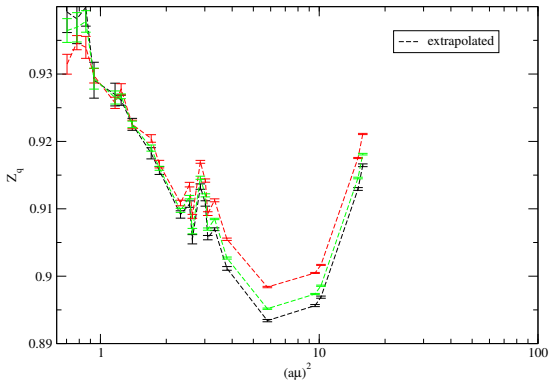


(g) The axial-vector/vector renormalization factor.

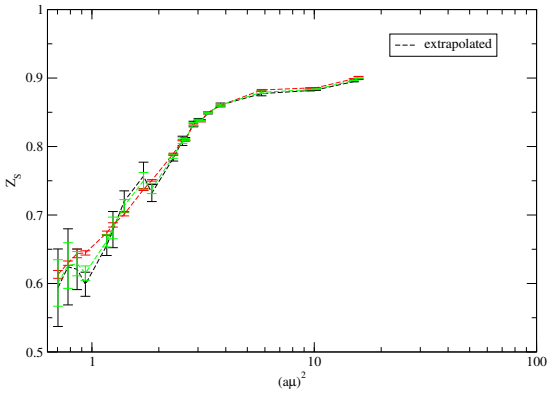


(h) The pseudo-scalar/scalar renormalization factor.

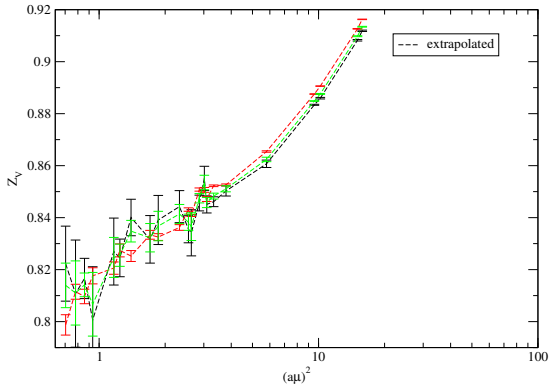
Figure 11.12: Scale dependence of the chiral extrapolation of RI' renormalization factors of lattice C ($\beta = 4.58$) of various renormalization factors. Plots differ in vertical scales.



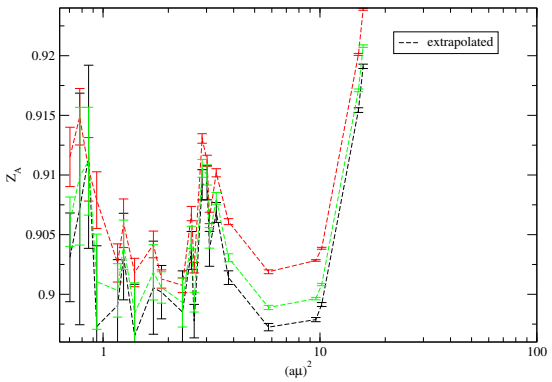
(a) The quark renormalization factor.



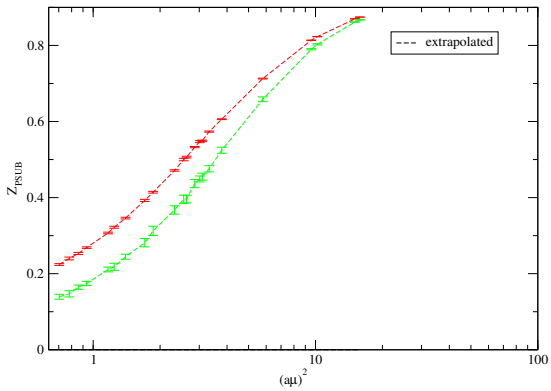
(b) The scalar renormalization factor.



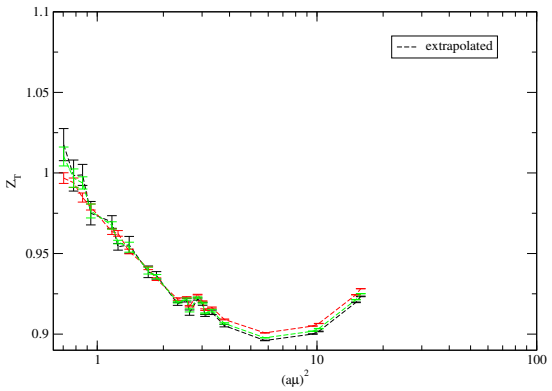
(c) The vector renormalization factor.



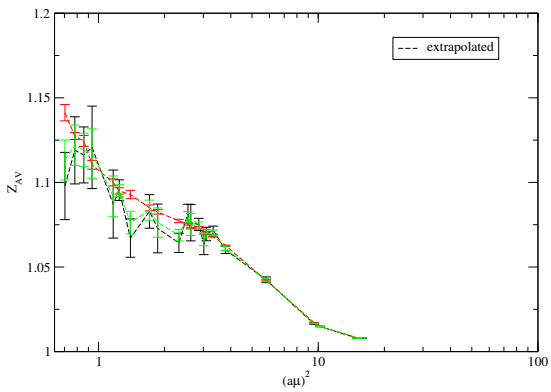
(d) The axial-vector renormalization factor.



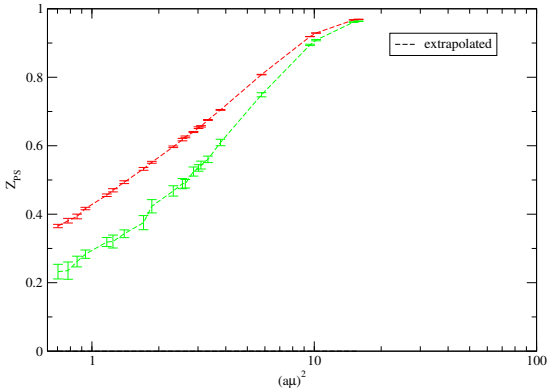
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.

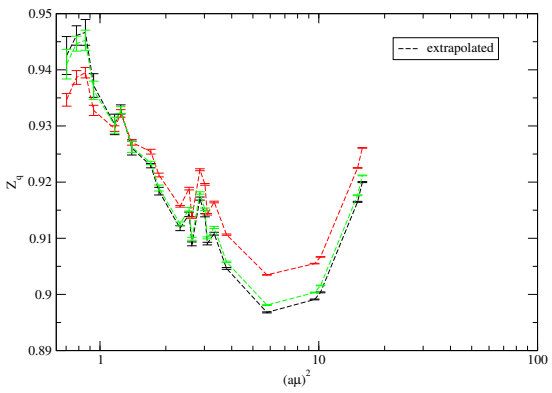


(g) The axial-vector/vector renormalization factor.

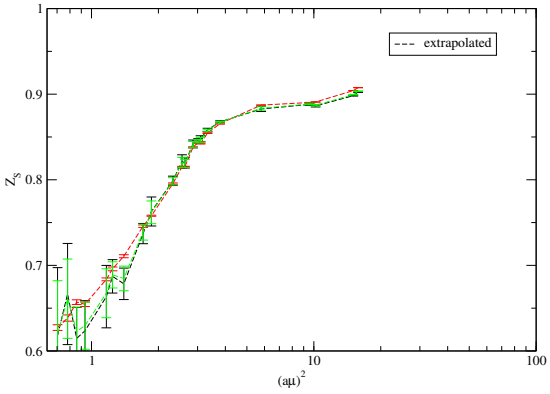


(h) . . .the pseudo-scalar/scalar renormalization factor.

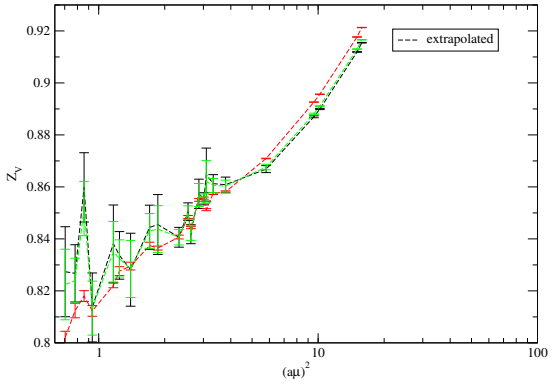
Figure 11.13: Scale dependence of the chiral extrapolation of RI' renormalization factors of lattice B ($\beta = 4.65$) of various renormalization factors. Plots differ in vertical scales.



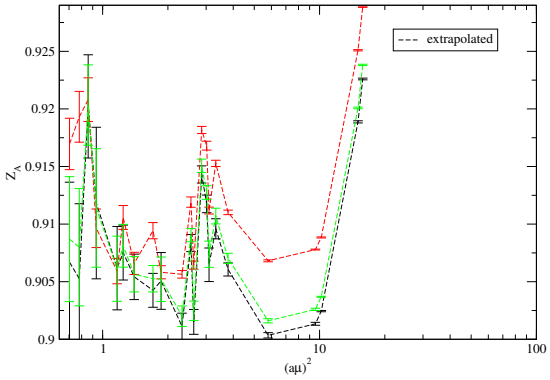
(a) The quark renormalization factor.



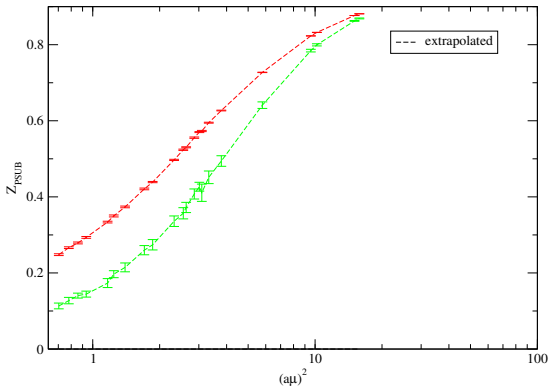
(b) The scalar renormalization factor.



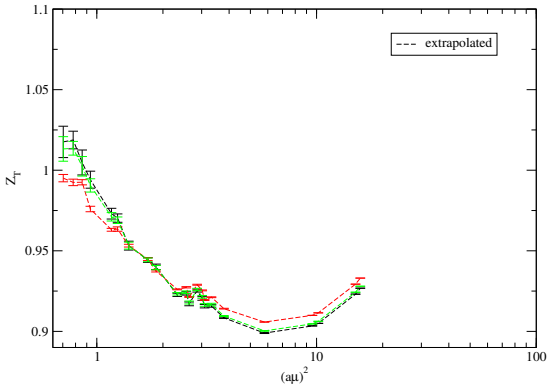
(c) The vector renormalization factor.



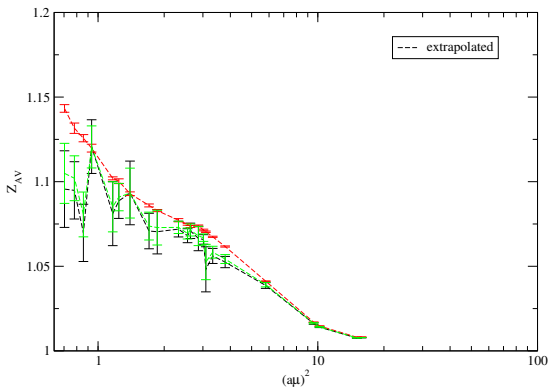
(d) The axial-vector renormalization factor.



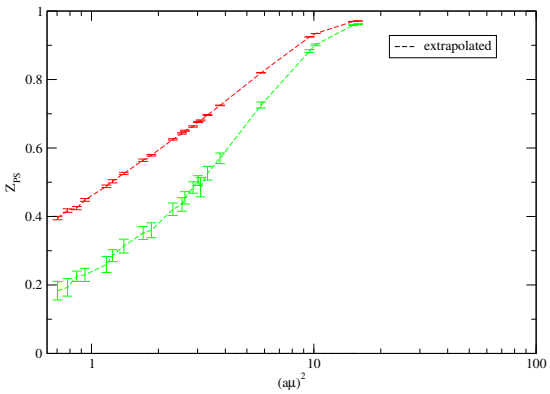
(e) The pseudo-scalar renormalization factor.



(f) The tensor renormalization factor.



(g) The axial-vector/vector renormalization factor.



(h) The pseudo-scalar/scalar renormalization factor.

Figure 11.14: Scale dependence of the chiral extrapolation of RI' renormalization factors of lattice A ($\beta = 4.70$) of various renormalization factors. Plots differ in vertical scales.

11.6 Final Results

We show our final results in tables 11.4, 11.5, 11.6, 11.7, 11.8, 11.9, 11.10 and 11.11. For lattices A, B and C we extrapolated in the sea quark mass. All other lattices were extrapolated with respect to the valence quark mass, i.e., these are partially quenched results. Values of 0 indicate, that an extrapolation was not possible due to a lack of data points, i.e., one would need more mass values.

lattice	$Z_q^{\mathrm{RI'}}$	$Z_q^{\overline{\mathrm{MS}}}$	Z_q^{RGI}
A	0.9212(5)	0.9083(5)	0.8678(5)
B	0.9154(6)	0.9027(6)	0.8624(5)
C	0.9082(7)	0.8955(7)	0.8555(6)
A50	0.9010(2)	0.8885(2)	0.8489(2)
A66	0.9162(6)	0.9035(6)	0.8632(6)
B60	0.8995(5)	0.8870(5)	0.8474(5)
B70	0.9093(2)	0.8966(2)	0.8565(2)
C64	0.8920(4)	0.8796(4)	0.8404(4)
C72	0.8959(4)	0.8835(4)	0.8440(4)
C77	0.9009(2)	0.8883(2)	0.8486(2)

Table 11.4: Quark renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_S^{\mathrm{RI'}}$	$Z_S^{\overline{\mathrm{MS}}}$	Z_S^{RGI}
A	0.7472(138)	0.8907(165)	0.4101(76)
B	0.7341(121)	0.8751(144)	0.4029(67)
C	0.7481(45)	0.8918(54)	0.4108(25)
A50	0.7741(11)	0.9228(13)	0.4248(6)
A66	0.7451(3)	0.8882(35)	0.4089(16)
B60	0.7717(9)	0.9199(11)	0.4235(5)
B70	0.7550(12)	0.9002(15)	0.4147(7)
C64	0.7799(13)	0.9296(15)	0.4280(7)
C72	0.7664(17)	0.9135(2)	0.4206(9)
C77	0.7587(4)	0.9046(5)	0.4167(2)

Table 11.5: Scalar renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_V^{\text{RI'}}$	$Z_V^{\overline{\text{MS}}}$	Z_V^{RGI}
A	0.8449(97)	0.8397(96)	0.8397(96)
B	0.8393(93)	0.8342(93)	0.8342(93)
C	0.8275(32)	0.8224(32)	0.8224(32)
A50	0.8241(8)	0.8191(8)	0.8191(8)
A66	0.8348(4)	0.8297(4)	0.8297(4)
B60	0.8225(11)	0.8175(11)	0.8175(11)
B70	0.8306(9)	0.8256(9)	0.8256(9)
C64	0.8177(11)	0.8128(11)	0.8128(11)
C72	0.8189(17)	0.8139(17)	0.8139(17)
C77	0.8228(5)	0.8178(5)	0.8178(5)

Table 11.6: Vector renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_A^{\text{RI'}}$	$Z_A^{\overline{\text{MS}}}$	Z_A^{RGI}
A	0.9046(19)	0.8991(18)	0.8991(18)
B	0.9001(23)	0.8946(23)	0.8946(23)
C	0.8933(15)	0.8879(15)	0.8879(15)
A50	0.8907(5)	0.8853(5)	0.8853(5)
A66	0.9018(19)	0.8963(19)	0.8963(19)
B60	0.8890(7)	0.8836(7)	0.8836(7)
B70	0.8957(6)	0.8902(6)	0.8902(6)
C64	0.8852(7)	0.8798(7)	0.8798(7)
C72	0.8859(9)	0.8805(9)	0.8805(9)
C77	0.8889(4)	0.8835(4)	0.8835(4)

Table 11.7: Axialvector renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_P^{\text{RI'}}$	$Z_P^{\overline{\text{MS}}}$	Z_P^{RGI}
A	0.0000()	0.0000()	0.0000()
B	0.0000()	0.0000()	0.0000()
C	0.6117(991)	0.7293(1183)	0.3361(547)
A50	0.6383(83)	0.7609(1)	0.3503(46)
A66	0.5786(262)	0.6898(312)	0.3176(144)
B60	0.6253(7)	0.7454(83)	0.3432(38)
B70	0.6008(13)	0.7164(124)	0.3300(57)
C64	0.6332(416)	0.7547(496)	0.3475(228)
C72	0.5924(217)	0.7061(259)	0.3251(119)
C77	0.6104(42)	0.7277(5)	0.3353(23)

Table 11.8: Pseudoscalar renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_T^{\text{RI'}}$	$Z_T^{\overline{\text{MS}}}$	Z_T^{RGI}
A	0.9426(16)	0.9208(15)	1.1591(19)
B	0.9357(23)	0.9140(22)	1.1507(28)
C	0.9268(15)	0.9052(14)	1.1394(18)
A50	0.9134(4)	0.8922(4)	1.1232(5)
A66	0.9373(15)	0.9156(14)	1.1526(18)
B60	0.9116(9)	0.8904(8)	1.1209(11)
B70	0.9253(5)	0.9038(5)	1.1374(6)
C64	0.9027(5)	0.8817(5)	1.1101(6)
C72	0.9084(7)	0.8873(7)	1.1171(9)
C77	0.9155(4)	0.8942(4)	1.1254(5)

Table 11.9: Tensor renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_{AV}^{\text{RI'}}$	$Z_{AV}^{\overline{\text{MS}}}$	Z_{AV}^{RGI}
A	1.0707(115)	1.0642(114)	1.0642(114)
B	1.0725(141)	1.0659(14)	1.0659(14)
C	1.0790(34)	1.0724(34)	1.0724(34)
A50	1.0805(7)	1.0740(7)	1.0740(7)
A66	1.0794(26)	1.0728(26)	1.0728(26)
B60	1.0807(9)	1.0741(8)	1.0741(8)
B70	1.0790(9)	1.0724(9)	1.0724(9)
C64	1.0824(1)	1.0758(9)	1.0758(9)
C72	1.0818(17)	1.0752(17)	1.0752(17)
C77	1.0802(4)	1.0736(4)	1.0736(4)

Table 11.10: Axialvector/Vector renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

lattice	$Z_{PS}^{\text{RI'}}$	$Z_{PS}^{\overline{\text{MS}}}$	Z_{PS}^{RGI}
A	0.0000()	0.0000()	0.0000()
B	0.0000()	0.0000()	0.0000()
C	0.8900(1962)	0.8845(195)	0.8845(195)
A50	0.8049(88)	0.8000(87)	0.8000(87)
A66	0.7761(413)	0.7713(411)	0.7713(411)
B60	0.7915(73)	0.7867(73)	0.7867(73)
B70	0.7751(123)	0.7704(122)	0.7704(122)
C64	0.7979(389)	0.7930(386)	0.7930(386)
C72	0.7619(246)	0.7573(245)	0.7573(245)
C77	0.7842(48)	0.7794(48)	0.7794(48)

Table 11.11: Pseudoscalar/Scalar renormalization factors for the chirally extrapolated dynamical CI lattices to various schemes.

Chapter 12

Conclusions & Perspectives

12.1 QPACE

QPACE was a huge success for the University of Regensburg, IBM and all partners within the Sonderforschungsbereich (SFB). The project is likely to trigger a number of follow-up projects. For example, the iDataCool prototype of the University of Regensburg, a Hot-Water-Cooled IBM iDataPlex cluster, is one of them. Further, SuperMUC, a 3 PFLOP/s supercomputer of the Leibniz-Rechenzentrum (LRZ) near Munich, expected for 2012, is based [128] on Hot-Water-Cooling technology invented for QPACE, Aquasar and iDataCool.

12.2 Lattice QCD

In this PhD thesis we explained methods for the determination of hadron masses, as well as matrix elements and their non-perturbative renormalization using nearly chiral numerical simulations of QCD. Our results coincide very well with phenomenology in the axial charge of the nucleon ground state. For the excited nucleon states of positive and negative parity we found interesting predictions with small errors.

Other authors obtained useful predicitions of the excited nucleon axial charges from the non-relativistic quark model [129, 130]: $g_A^{0-} = -1/9$ and $g_A^{1-} = 5/9$. As shown by our results, these values may point into the right direction, but the model needs further improvement to efficiently explain physical values. From our calculations however, we can not be sure which of the negative parity nucleon states is the lower one, but the results from the non-relativistic quark model suggest that the negative parity nucleon ground state has a near 0 axial charge, while the first excited negative parity nucleon state has an axial charge near 1.

Other authors obtain different results. For example, Takahashi et al. [39] obtain significantly lower axial charges for their excited nucleons (by a factor of 2). This may be attributed to the fact that they employ Wilson type fermions, see section 8.1.2. This type of discretization breakes chiral symmetry explicitly on the lattice and this symmetry may only be recovered very late in continuum extrapolation. Chiral symmetry, and is spontaneous breaking, is argued to be especially important for the nucleon spectrum:

The occurrence of parity doublets in the upper light baryon spectrum (i.e., above 2 GeV) has been the motivation to suggest an effective restauration of chiral symmetry in high lying hadrons [131]. For the nucleon ground state no chiral partner can be identified, indicating that the symmetry breaking plays a major role in its mass generation, where a significant part of its mass can be shown to stem from the quark condensate.

The lightest positive parity excited nucleon, known as the Roper resonance, is $N^+(1440)$. The lightest negative parity states are $N^-(1535)$ and $N^-(1650)$. These are believed to be linear

combinations of a three quark state N_{3q} and possibly a Lambda-Kaon molecule state $N_{\lambda K}$ [132]. Here, the three quark state is believed to be the chiral partner of the Roper resonance. It is shown in [133] that parity doubling because of effective chiral symmetry restauration leads to vanishing axial charges for the doublet. For the positive parity states we find decreasing axial charges for more excited states, see table 9.25. This matches the hypothesis of effective chiral symmetry restauration. For the negative parity states we may see the same behavior if the state labeled 0 turns out to be lighter than the state labeled 1 for increasing statistics and at the physical point in AWI quark mass: As evident from figure 9.18 this means that the states would have to cross below our highest simulation point. In this case, axial charges may mix as well.

There are a number of interesting opportunities that may improve our results:

- While we obtain low noise interpolators for the positive parity ground state for any source smearing used, the negative parity ground state remains rather noisy. For large smearing radii, one can see a heavy degradation in the signal. One may argue that the radial part of the negative parity wavefunction is rather small or compact. When using more narrow sources we see improved signals for the ground, as well as for excited states. For this reason one may argue that the negative parity states have a different spatial distribution and that therefore different type of smearings may improve the overlap of sources and state and therefore the signal. Derivative sources have heavily improved lattice determinations of excited mesons properties, see [134, 135]. Such calculations may be very interesting for baryons as well.

- A continuum extrapolation is currently impossible for CI fermions because of the lack of simulation results at different lattice spacings. Although simulations were carried out at 3 different β, all of them came out to be at very similar lattice spacings (and volumes). However, for the negative parity states, we have seen that wider sources did degrade the signal, so we estimate that our lattices were large enough to carry these states.

- For the same reasons an infinite volume extrapolation could not be performed.

- We use the RI'-MOM scheme to non-perturbatively determine renormalization factors for a number of current operators. We perturbatively convert them to the $\overline{\text{MS}}$ scheme and remove known scale dependencies in the custom RGI scheme, where factors become flat in the perturbative sector. In order to estimate systematic errors in scheme conversions, a different intermediate scheme may be employed. For example the $\widetilde{\text{MOM}}$gg has been employed in a number of publications [136, 137].

- At high lattice momenta discretization artifacts seem to dominate the renormalization factors. Since different volumina are unavailable for CI fermions, extrapolations need to be taken to a later stage. A different method that was successfully applied to Wilson fermions in order to remove discretization artifacts is a lattice-perturbative subtraction [136, 137]. For CI fermions the action is more complicated and such calculations have not been attempted.

- Further systematical errors may be induced though the ignorance of higher order terms in scheme conversion and errors inferred by the uncertainties in the QCD scale parameter Λ_{QCD} and the Sommer scale r_0.

Appendix A

Notations, Conventions and Coefficients

A.1 Notations and Conventions

As common practice in high energy physics we use units where $\hbar = c = 1$ and the Einstein summation convention $a^\mu b_\mu = \sum_\mu a^\mu b_\mu$. Minkowski indices run $\mu \in \{0, 1, 2, 3\}$ where 0 is time. Further the totally antisymmetric Levi-Civita tensor is normalized such that $\epsilon_{0123} = +1$. We use the metric $g = \mathrm{diag}(+1, -1, -1, -1)$. The Dirac algebra is defined by the relations

$$\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu}\mathbb{1}, \qquad \gamma_0\gamma_\mu^\dagger\gamma_0 = \gamma_\mu \tag{A.1}$$

$$\gamma^5 \equiv i\gamma^0\gamma^1\gamma^2\gamma^3 = \gamma_5, \qquad \{\gamma^5, \gamma^\mu\} = 0, \qquad \gamma_5^\dagger = \gamma_5, \qquad (\gamma_5)^2 = \mathbb{1} \tag{A.2}$$

We define Charge Conjugation by

$$\mathcal{C}\psi(x)\mathcal{C}^{-1} = \eta_c C\bar{\psi}(x)^T \tag{A.3}$$

$$C\gamma_\mu C^{-1} = -\gamma_\mu^T \qquad \Rightarrow \qquad C^T = -C \tag{A.4}$$

In every hermitian representation we can choose C such that

$$C^{-1} = C^\dagger = -C^* \Rightarrow \qquad C\gamma_5 C^{-1} = \gamma_5^T \tag{A.5}$$

A.2 Euclideanization

The process of Euclideanization is an essential step in rendering the path integral real valued in order to make numerical calculations feasible. To achieve this we replace any dependence on the actual time t by $-i\tau$: Let $O_{\mathrm{M}}(x)$ be some standard Minkowskian operator. Then the Euclidean version of this operator is defined by:

$$O(x) := O(x^1, x^2, x^3, x^4 = \tau) = O_{\mathrm{M}}(-i\tau, x^1, x^2, x^3) \tag{A.6}$$

Now we define for any Minkowski vector operator A_{M}^μ its Euclidean partner by

$$A_i := A^i := A_{\mathrm{M}}^i = -A_i^{\mathrm{M}}, \qquad A_4 := A^4 := iA_{\mathrm{M}}^0 = iA_0^{\mathrm{M}}, \tag{A.7}$$

implicitly setting the new metric to $g = \mathbb{1}$. We go on looking at vector products and obtain the simple rule

$$A_{\mathrm{M}}^\mu B_\mu^{\mathrm{M}} = A_{\mathrm{M}}^0 B_{\mathrm{M}}^0 - A_{\mathrm{M}}^i B_{\mathrm{M}}^i = i^2 A^4 B_4 - A^i B_i = -A^\mu B_\mu, \tag{A.8}$$

which means that for example the action, which is a contraction of various vectors transforms as simple as $L_{\mathrm{M}} = -L$, which we can use to evaluate

$$iS_{\mathrm{M}} = i\int d^4x_{\mathrm{M}} L_{\mathrm{M}} = -i\int d(-i\tau)d^3xL = -\int d^4xL = -S \tag{A.9}$$

Further looking at the covariant derivative $D^{\mathrm{M}}_{\mu} = \partial^{\mathrm{M}}_{\mu} - igA^{\mathrm{M}}_{\mu}$ we obtain its Euclideanized version from

$$D^{\mathrm{M}}_0 = D^0_{\mathrm{M}} = i\partial_4 - ig(-i)A_4 = i(\partial_4 + igA_4) = iD_4 = iD^4 \tag{A.10}$$

$$-D^{\mathrm{M}}_i = D^i_{\mathrm{M}} = -\partial_i - igA_i = -D_i = -D^i \tag{A.11}$$

Now that we set the metric to unity, the gamma matrices will have to follow the rule

$$\{\gamma^{\mu}, \gamma^{\nu}\} = 2\delta^{\mu\nu}\mathbb{1} \tag{A.12}$$

and that the Euclidean gamma matrices γ corresponding to the Minkowskian gamma matrices γ_{M} can be calculated by

$$\gamma^i = -i\gamma^i_{\mathrm{M}}, \qquad \gamma^4 = \gamma^0_{\mathrm{M}}, \qquad \gamma^5 = -\gamma^5_{\mathrm{M}} \tag{A.13}$$

A.3 Chirally Improved Dirac Operator Coefficients

The D_{CI} operator has been introduced and defined in the papers [80, 81, 138]. We provide coefficients used throughout this work.

(a) for free fermions		(b) for the all dynamical $n_f = 2$ lattices	
value	coefficient	value	coefficient
$+0.1409870061 \cdot 10^{+1}$	s_1	$+0.1481599252 \cdot 10^{+1}$	s_1
$-0.4063348276 \cdot 10^{-1}$	s_2	$-0.5218251439 \cdot 10^{-1}$	s_2
$-0.1328179378 \cdot 10^{-1}$	s_3	$-0.1473643847 \cdot 10^{-1}$	s_3
$-0.1707793316 \cdot 10^{-2}$	s_5	$-0.2186103421 \cdot 10^{-2}$	s_5
$+0.1707277975 \cdot 10^{-2}$	s_6	$+0.2133989696 \cdot 10^{-2}$	s_6
$-0.2995931667 \cdot 10^{-2}$	s_8	$-0.3997001821 \cdot 10^{-2}$	s_8
$-0.4097715677 \cdot 10^{-3}$	s_{10}	$-0.4951673735 \cdot 10^{-3}$	s_{10}
$-0.7711930549 \cdot 10^{-3}$	s_{11}	$-0.9836500799 \cdot 10^{-3}$	s_{11}
$+0.6542013926 \cdot 10^{-2}$	s_{13}	$+0.7529838581 \cdot 10^{-2}$	s_{13}
$+0.2526693368 \cdot 10^{+0}$	v_1	$+0.1972229309 \cdot 10^{+0}$	v_1
$+0.4483311559 \cdot 10^{-2}$	v_2	$+0.8252157565 \cdot 10^{-2}$	v_2
$+0.3493344361 \cdot 10^{-2}$	v_4	$+0.5113056314 \cdot 10^{-2}$	v_4
$+0.1077099799 \cdot 10^{-2}$	v_5	$+0.1736609425 \cdot 10^{-2}$	v_5
$-0.7464002396 \cdot 10^{-1}$	t_1	$-0.8792744664 \cdot 10^{-1}$	t_1
$-0.1947456954 \cdot 10^{-2}$	t_2	$-0.2553055577 \cdot 10^{-2}$	t_2
$0.1702447555 \cdot 10^{-2}$	t_3	$+0.2093792069 \cdot 10^{-2}$	t_3
$-0.4273892564 \cdot 10^{-2}$	t_5	$-0.5567377075 \cdot 10^{-2}$	t_5
$-0.2924376836 \cdot 10^{-2}$	t_{15}	$-0.3427310798 \cdot 10^{-2}$	t_{15}
$-0.6927076246 \cdot 10^{-2}$	p_1	$-0.8184103136 \cdot 10^{-2}$	p_1

Table A.1: Coefficients used for the CI Dirac operator

Clifford generator	generating path	coefficient
$\mathbb{1}$	$\langle\rangle$	s_1
$\mathbb{1}$	$\langle 1\rangle$	s_2
$\mathbb{1}$	$\langle 1,2\rangle$	s_3
$\mathbb{1}$	$\langle 1,1\rangle$	s_4
$\mathbb{1}$	$\langle 1,2,3\rangle$	s_5
$\mathbb{1}$	$\langle 1,1,2\rangle$	s_6
$\mathbb{1}$	$\langle 1,2,1\rangle$	s_7
$\mathbb{1}$	$\langle 1,2,-1\rangle$	s_8
$\mathbb{1}$	$\langle 1,1,1\rangle$	s_9
$\mathbb{1}$	$\langle 1,2,3,4\rangle$	s_{10}
$\mathbb{1}$	$\langle 1,2,-1,3\rangle$	s_{11}
$\mathbb{1}$	$\langle 1,2,3,-1\rangle$	s_{12}
$\mathbb{1}$	$\langle 1,2,-1,-2\rangle$	s_{13}
$\vdots$	$\vdots$	$\vdots$
γ_μ	$\langle 1\rangle$	v_1
γ_μ	$\langle 1,2\rangle$	v_2
γ_μ	$\langle 1,1,1\rangle$	v_3
γ_μ	$\langle 1,2,3\rangle$	v_4
γ_μ	$\langle 2,1,3\rangle$	v_5
γ_μ	$\langle 1,2,2\rangle$	v_6
γ_μ	$\langle 1,1,2\rangle$	v_7
γ_μ	$\langle 1,2,2\rangle$	v_8
γ_μ	$\langle 1,2,1\rangle$	v_9
$\vdots$	$\vdots$	$\vdots$
$\gamma_\mu\gamma_\nu$	$\langle 1,2\rangle$	t_1
$\gamma_\mu\gamma_\nu$	$\langle 1,2,3\rangle$	t_2
$\gamma_\mu\gamma_\nu$	$\langle 1,3,2\rangle$	t_3
$\gamma_\mu\gamma_\nu$	$\langle 1,1,2\rangle$	t_4
$\gamma_\mu\gamma_\nu$	$\langle 1,2,-1\rangle$	t_5
$\gamma_\mu\gamma_\nu$	$\langle 1,2,3,4\rangle$	t_6
$\vdots$	$\vdots$	$\vdots$
$\gamma_\mu\gamma_\nu$	$\langle 1,2,-1,-2\rangle$	t_{15}
$\vdots$	$\vdots$	$\vdots$
$\gamma_\mu\gamma_\nu\gamma_\rho$	$\langle 1,2,3\rangle$	a_1
$\gamma_\mu\gamma_\nu\gamma_\rho$	$\langle 1,2,3,4\rangle$	a_2
$\vdots$	$\vdots$	$\vdots$
γ_5	$\langle 1,2,3,4\rangle$	p_1
$\vdots$	$\vdots$	$\vdots$

Table A.2: List of the paths corresponding to the lowest coefficients of the CI Dirac operator.

References

[1] C. Eames and R. Eames, "**A Computer Perspective: Background to the Computer Age, Catalog of a unique computer history exhibit at IBM headquarters in 1971**", *Harvard University Press* (1973).

[2] Wikipedia, "**Tabulating Machine**", `wiki/Tabulating_machine`.

[3] Wikipedia, "**Supercomputer**", `wiki/Supercomputer`.

[4] Argonne National Laboratory, "**IBM Blue Gene/P supercomputer at the Argonne Leadership Computing Facility. Photo, courtesy of Argonne National Laboratory**", `wiki/File:IBM_Blue_Gene_P_supercomputer.jpg`.

[5] Top500.org, "**The Top 500 List - November 2010**", (2010) `2010/11`.

[6] H. Markram, "**The Blue Brain Project**", *Nature Reviews Neuroscience* **7** (2006) 153–160.

[7] H. Markram, "**Henry Markram on the Blue Brain Project**", *The Desgin and the Elastic Mind Symposium* **7** (2008) `www.youtube.com/watch?v=1fU4mK7ERfw`.

[8] E. P. DeBenedicti, "**Reversible Logic for Supercomputing**", *Proceedings of the 2nd conference on Computing frontiers* (2005) `portal.acm.org/citation.cfm?id=1062325`.

[9] G. Bilardi *et al.*, "**The Potential of On-Chip Multiprocessing for QCD Machines**", *HiPC 2005* **LNCS vol. 3769** (2005).

[10] F. Belletti *et al.*, "**QCD on the Cell Broadband Engine**", *PoS* **LAT2007** (2007) 039, `0710.2442`.

[11] G. Goldrian *et al.*, "**QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine**", *Comput. Sci. Eng.* **10** (2008) 26.

[12] S. F. Schifano, "**Computing Systems for Theoretical Physics**", *Young Investigators Symposium* (2008) `symposium08/presentations/s_schifano.pdf`.

[13] F. Belletti *et al.*, "**Computing for LQCD: apeNEXT**", *Comput. Sci. Eng.* **8** (2006) 18.

[14] P. Boyle *et al.*, "**Overview of the QCDSP and QCDOC computers**", *IBM Journal of Research & Development* **49** (2005) 351.

[15] D. Pleiter, "**Lattice QCD on the Cell BE**", *Power Architecture Developer Conference* (2007) `07/Session_Downloads/PADC07_Pleiter.pdf`.

[16] H. Baier *et al.*, "**Status of the QPACE Project**", *PoS* **LATTICE2008** (2008) 039, 0810.1559.

[17] H. Baier *et al.*, "**QPACE – a QCD parallel computer based on Cell processors**", *PoS* **LAT2009** (2009) 001, 0911.2174.

[18] Y. Nakamura *et al.*, "**Lattice QCD Applications on QPACE**", 1103.1363.

[19] L. Biferale, F. Mantovani, M. Pivanti, M. Sbragaglia, A. Scagliarini, S. Schifano, F. Toschi, and R. Tripiccione, "**Lattice Boltzmann fluid-dynamics on the QPACE supercomputer**", *Procedia Computer Science* **1** (2010), no. 1, 1075 – 1082, science/article/B9865-506HM1Y-46/2/dec46e0f79d4dcd0560908c8b4c740f0. ICCS 2010.

[20] Green500.org, "**The Green 500 List - November 2009**", (2009) 2011/09/top/list.php.

[21] Green500.org, "**The Green 500 List - June 2010**", (2010) 2010/06/top/list.php.

[22] Xilinx, "**Virtex-5 FPGA User Guide**", **v5.3** (2011) user_guides/ug190.pdf.

[23] I. Ouda and K. Schleupen, "**Application Note: FPGA to IBM Power Processor Interface Setup**", (2008).

[24] IBM, "**Device Control Register Bus, Architecture Specifications**", **v3.5** (2006) $file/DcrBus.pdf.

[25] S. Solbrig, "**Synchronization and Error Reporting on QPACE**", (2010) STRONGnet2010/solbrig.pdf.

[26] M. Pivanti, S. F. Schifano, and H. Simma, "**An FPGA-based Torus Communication Network**", *PoS* **LATTICE2010** (2010) 038, 1102.2346.

[27] IEEE, "**802.3™: CSMA/CD (Ethernet) ACCESS METHOD**", (1983-2011) 802/802.3.html.

[28] Wikipedia, "**OSI Model**", (March 7, 2011) wiki/OSI_model.

[29] Xilinx, "**Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide**", (2011) user_guides/ug194.pdf.

[30] Xilinx, "**Virtex-5 FPGA Embedded Tri-Mode Ethernet MAC User Guide**", (2011) sw_manuals/xilinx11/sim.pdf.

[31] S. Heybrock, "**A high-performance network controller for the QPACE architecture**", *Diploma Thesis* **University of Regensburg** (2008).

[32] J. Corbet, A. Rubini, and G. Kroah-Hartman, "**Linux Device Drivers**", **v3** (2005).

[33] **QCDSF/UKQCD** Collaboration, F. Winter *et al.*, "**Baryon Axial Charges and Momentum Fractions with $N_f = 2 + 1$ Dynamical Fermions**", *PoS* **LATTICE2010** (2010) 163, 1102.3407.

[34] **QCDSF/UKQCD** Collaboration, D. Pleiter *et al.*, "**Nucleon form factors and structure functions from N(f)=2 Clover fermions**", *PoS* **LATTICE2010** (2010) 153, 1101.2326.

[35] **ETM** Collaboration, C. Alexandrou *et al.*, "**Axial Nucleon form factors from lattice QCD**", *Phys. Rev.* **D83** (2011) 045010, 1012.0857.

[36] D. B. Renner, "**Status and prospects for the calculation of hadron structure from lattice QCD**", *PoS* **LAT2009** (2009) 018, 1002.0925.

[37] T. T. Takahashi and T. Kunihiro, "**Lattice QCD study of $g_A^{N^*N^*}$ with two flavors of dynamical quarks**", *ECONF* **C070910** (2007) 297, 0711.1961.

[38] K.-S. Choi, W. Plessas, and R. F. Wagenbrunn, "**Axial charges of the nucleon and N* resonances**", *Phys. Rev.* **C81** (2010) 028201, 0908.3959.

[39] T. T. Takahashi and T. Kunihiro, "**Two-flavor lattice QCD study of the axial charges of N(1535) and N(1650)**", *Nucl. Phys. Proc. Suppl.* **186** (2009) 113–116, 0911.2543.

[40] M. E. Peskin and D. V. Schroeder, *An Introduction to Quantum Field Theory*. Westview Press, 1995.

[41] A. V. Manohar, "**An introduction to spin dependent deep inelastic scattering**", hep-ph/9204208.

[42] B. E. Tipton, "**Measurement of Polarized Parton Distributions with Spin-Dependent Deep-Inelastic Scattering**",.

[43] **Particle Data Group** Collaboration, K. Hagiwara *et al.*, "**Review of particle physics**", *Phys. Rev.* **D66** (2002) 010001.

[44] **the RIKEN-BNL-Columbia-KEK** Collaboration, S. Sasaki, K. Orginos, S. Ohta, and T. Blum, "**Nucleon axial charge from quenched lattice QCD with domain wall fermions**", *Phys. Rev.* **D68** (2003) 054509, hep-lat/0306007.

[45] H. J. Rothe, *Lattice Gauge Theories: An Introduction (Third Edition)*. World Scientific, 2005.

[46] K. G. Wilson, "**Confinement of quarks**", *Phys. Rev.* **D10** (1974) 2445–2459.

[47] K. G. Wilson, *New Phenomena In Subnuclear Physics*. Plenum Press, New York, 1977.

[48] B. Sheikholeslami and R. Wohlert, "**Improved Continuum Limit Lattice Action for QCD with Wilson Fermions**", *Nucl. Phys.* **B259** (1985) 572.

[49] M. Lüscher, S. Sint, R. Sommer, P. Weisz, and U. Wolff, "**Non-perturbative O(a) improvement of lattice QCD**", *Nucl. Phys.* **B491** (1997) 323–343, hep-lat/9609035.

[50] R. G. Edwards, U. M. Heller, and T. R. Klassen, "**The effectiveness of non-perturbative O(a) improvement in lattice QCD**", *Phys. Rev. Lett.* **80** (1998) 3448–3451, hep-lat/9711052.

[51] J. B. Kogut and L. Susskind, "**Hamiltonian Formulation of Wilson's Lattice Gauge Theories**", *Phys. Rev.* **D11** (1975) 395.

[52] L. Susskind, "**Lattice Fermions**", *Phys. Rev.* **D16** (1977) 3031–3039.

[53] H. S. Sharatchandra, H. J. Thun, and P. Weisz, "**Susskind Fermions on a Euclidean Lattice**", *Nucl. Phys.* **B192** (1981) 205.

[54] H. Kluberg-Stern, A. Morel, O. Napoly, and B. Petersson, "**Flavors of Lagrangian Susskind Fermions**", *Nucl. Phys.* **B220** (1983) 447.

[55] G. W. Kilcup and S. R. Sharpe, "**A tool kit for staggered fermions**", *Nucl. Phys.* **B283** (1987) 493.

[56] M. Lüscher and P. Weisz, "**On-Shell Improved Lattice Gauge Theories**", *Commun. Math. Phys.* **97** (1985) 59.

[57] G. Curci, P. Menotti, and G. Paffuti, "**Symanzik's improved Lagrangian for lattice gauge theory**", *Phys. Lett.* **B130** (1983) 205.

[58] G. P. Lepage and P. B. Mackenzie, "**On the viability of lattice perturbation theory**", *Phys. Rev.* **D48** (1993) 2250–2264, `hep-lat/9209022`.

[59] M. G. Alford, W. Dimm, G. P. Lepage, G. Hockney, and P. B. Mackenzie, "**Lattice QCD on small computers**", *Phys. Lett.* **B361** (1995) 87–94, `hep-lat/9507010`.

[60] H. B. Nielsen and M. Ninomiya, "**Absence of Neutrinos on a Lattice. 1. Proof by Homotopy Theory**", *Nucl. Phys.* **B185** (1981) 20.

[61] H. B. Nielsen and M. Ninomiya, "**Absence of Neutrinos on a Lattice. 2. Intuitive Topological Proof**", *Nucl. Phys.* **B193** (1981) 173.

[62] P. H. Ginsparg and K. G. Wilson, "**A Remnant of Chiral Symmetry on the Lattice**", *Phys. Rev.* **D25** (1982) 2649.

[63] S. Capitani, M. Göckeler, R. Horsley, P. E. L. Rakow, and G. Schierholz, "**Operator improvement for Ginsparg-Wilson fermions**", *Phys. Lett.* **B468** (1999) 150–160, `hep-lat/9908029`.

[64] H. Neuberger, "**Exactly massless quarks on the lattice**", *Phys. Lett.* **B417** (1998) 141–144, `hep-lat/9707022`.

[65] H. Neuberger, "**More about exactly massless quarks on the lattice**", *Phys. Lett.* **B427** (1998) 353–355, `hep-lat/9801031`.

[66] R. Narayanan and H. Neuberger, "**A Construction of lattice chiral gauge theories**", *Nucl. Phys.* **B443** (1995) 305–385, `hep-th/9411108`.

[67] D. B. Kaplan, "**A Method for simulating chiral fermions on the lattice**", *Phys. Lett.* **B288** (1992) 342–347, `hep-lat/9206013`.

[68] J. Callan, Curtis G. and J. A. Harvey, "**Anomalies and Fermion Zero Modes on Strings and Domain Walls**", *Nucl. Phys.* **B250** (1985) 427.

[69] Y. Shamir, "**Chiral fermions from lattice boundaries**", *Nucl. Phys.* **B406** (1993) 90–106, `hep-lat/9303005`.

[70] V. Furman and Y. Shamir, "**Axial symmetries in lattice QCD with Kaplan fermions**", *Nucl. Phys.* **B439** (1995) 54–78, `hep-lat/9405004`.

[71] P. Hasenfratz and F. Niedermayer, "**Perfect lattice action for asymptotically free theories**", *Nucl. Phys.* **B414** (1994) 785–814, hep-lat/9308004.

[72] T. A. DeGrand, A. Hasenfratz, P. Hasenfratz, and F. Niedermayer, "**The Classically perfect fixed point action for SU(3) gauge theory**", *Nucl. Phys.* **B454** (1995) 587–614, hep-lat/9506030.

[73] U. J. Wiese, "**Fixed point actions for Wilson fermions**", *Phys. Lett.* **B315** (1993) 417–424, hep-lat/9306003.

[74] W. Bietenholz, R. Brower, S. Chandrasekharan, and U. J. Wiese, "**Perfect lattice actions for staggered fermions**", *Nucl. Phys.* **B495** (1997) 285–305, hep-lat/9612007.

[75] P. Hasenfratz, "**Lattice QCD without tuning, mixing and current renormalization**", *Nucl. Phys.* **B525** (1998) 401–409, hep-lat/9802007.

[76] P. Hasenfratz *et al.*, "**The construction of generalized Dirac operators on the lattice**", *Int. J. Mod. Phys.* **C12** (2001) 691–708, hep-lat/0003013.

[77] P. Hasenfratz, "**Prospects for perfect actions**", *Nucl. Phys. Proc. Suppl.* **63** (1998) 53–58, hep-lat/9709110.

[78] **BGR** Collaboration, C. Gattringer *et al.*, "**Quenched spectroscopy with fixed-point and chirally improved fermions**", *Nucl. Phys.* **B677** (2004) 3–51, hep-lat/0307013.

[79] **Bern-Graz-Regensburg** Collaboration, P. Hasenfratz, K. J. Juge, and F. Niedermayer, "**New results on cut-off effects in spectroscopy with the fixed point action**", *JHEP* **12** (2004) 030, hep-lat/0411034.

[80] C. Gattringer, "**A new approach to Ginsparg-Wilson fermions**", *Phys. Rev.* **D63** (2001) 114501, hep-lat/0003005.

[81] C. Gattringer, I. Hip, and C. B. Lang, "**Approximate Ginsparg-Wilson fermions: A first test**", *Nucl. Phys.* **B597** (2001) 451–474, hep-lat/0007042.

[82] R. Sommer, "**A New way to set the energy scale in lattice gauge theories and its applications to the static force and alpha-s in SU(2) Yang-Mills theory**", *Nucl. Phys.* **B411** (1994) 839–854, hep-lat/9310022.

[83] **ALPHA** Collaboration, M. Guagnelli, R. Sommer, and H. Wittig, "**Precision computation of a low-energy reference scale in quenched lattice QCD**", *Nucl. Phys.* **B535** (1998) 389–402, hep-lat/9806005.

[84] R. Gupta, T. Bhattacharya, and G. Kilcup, "**Comparison of inversion algorithms for Wilson fermions on the CM5**", hep-lat/9605029.

[85] V. Bernard, N. Kaiser, and U.-G. Meissner, "**Chiral dynamics in nucleons and nuclei**", *Int. J. Mod. Phys.* **E4** (1995) 193–346, hep-ph/9501384.

[86] S. Weinberg, "**Phenomenological Lagrangians**", *Physica* **A96** (1979) 327.

[87] S. Weinberg, "**Effective chiral Lagrangians for nucleon - pion interactions and nuclear forces**", *Nucl. Phys.* **B363** (1991) 3–18.

[88] M. Lüscher, "**Selected topics in lattice field theory**",. Lectures given at Summer School 'Fields, Strings and Critical Phenomena', Les Houches, France, Jun 28 - Aug 5, 1988.

[89] G. P. Lepage, "**The Analysis of Algorithms for Lattice Field Theroy**",. Invited lectures given at TASI'89 Summer School, Boulder, Colorado, Jun 4-30, 1989.

[90] M. Lüscher and U. Wolff, "**How to calculate the elastic scattering matrix in two-dimensional quantum field theories by numerical simulation**", *Nucl. Phys.* **B339** (1990) 222–252.

[91] B. Blossier, G. von Hippel, T. Mendes, R. Sommer, and M. Della Morte, "**Efficient use of the Generalized Eigenvalue Problem**", *PoS* **LATTICE2008** (2008) 135, 0808.1017.

[92] T. Burch, C. Hagen, C. B. Lang, M. Limmer, and A. Schafer, "**Excitations of single-beauty hadrons**", *Phys. Rev.* **D79** (2009) 014504, 0809.1103.

[93] **Particle Data Group** Collaboration, W. M. Yao *et al.*, "**Review of particle physics**", *J. Phys.* **G33** (2006) 1–1232.

[94] **Bern-Graz-Regensburg** Collaboration, D. Brommel *et al.*, "**Low lying nucleons from chirally improved fermions**", *Nucl. Phys. Proc. Suppl.* **129** (2004) 251–253, hep-lat/0309036.

[95] T. Burch *et al.*, "**Excited hadrons on the lattice: Baryons**", *Phys. Rev.* **D74** (2006) 014504, hep-lat/0604019.

[96] **UKQCD** Collaboration, C. R. Allton *et al.*, "**Gauge invariant smearing and matrix correlators using Wilson fermions at Beta = 6.2**", *Phys. Rev.* **D47** (1993) 5128–5137, hep-lat/9303009.

[97] C. Best *et al.*, "**Pion and rho structure functions from lattice QCD**", *Phys. Rev.* **D56** (1997) 2743–2754, hep-lat/9703014.

[98] G. Engel, C. B. Lang, M. Limmer, D. Mohler, and A. Schäfer, "**Chirally improved lattice spacings and AWI-masses**", *private communication, paper in prepatation.*

[99] C. Morningstar and M. J. Peardon, "**Analytic smearing of SU(3) link variables in lattice QCD**", *Phys. Rev.* **D69** (2004) 054501, hep-lat/0311018.

[100] A. Hasenfratz and F. Knechtli, "**Flavor symmetry and the static potential with hypercubic blocking**", *Phys. Rev.* **D64** (2001) 034504, hep-lat/0103029.

[101] G. P. Engel, C. B. Lang, M. Limmer, D. Mohler, and A. Schafer, "**Some results on excited hadrons in 2-flavor QCD**", *PoS* **LATTICE2010** (2010) 103, 1010.2366.

[102] S. Capitani, "**Lattice perturbation theory**", *Phys. Rept.* **382** (2003) 113–302, hep-lat/0211036.

[103] M. Gockeler *et al.*, "**Nonperturbative renormalisation of composite operators in lattice QCD**", *Nucl. Phys.* **B544** (1999) 699–733, hep-lat/9807044.

[104] G. Martinelli, C. Pittori, C. T. Sachrajda, M. Testa, and A. Vladikas, "**A General method for nonperturbative renormalization of lattice operators**", *Nucl. Phys.* **B445** (1995) 81–108, hep-lat/9411010.

[105] H. Suman and K. Schilling, "**A Comparative study of gauge fixing procedures on the connection machines CM2 and CM5**", hep-lat/9306018.

[106] C. Gattringer, M. Göckeler, P. Huber, and C. B. Lang, "**Renormalization of bilinear quark operators for the chirally improved lattice Dirac operator**", *Nucl. Phys.* **B694** (2004) 170–186, hep-lat/0404006.

[107] V. Maillart and F. Niedermayer, "**A specific lattice artefact in non-perturbative renormalization of operators**", hep-lat/0807.0030.

[108] T. van Ritbergen, J. A. M. Vermaseren, and S. A. Larin, "**The four-loop beta function in quantum chromodynamics**", *Phys. Lett.* **B400** (1997) 379–384, hep-ph/9701390.

[109] K. G. Chetyrkin and A. Rétey, "**Renormalization and running of quark mass and field in the regularization invariant and MS-bar schemes at three and four loops**", *Nucl. Phys.* **B583** (2000) 3–34, hep-ph/9910332.

[110] J. A. Gracey, "**Three loop anomalous dimension of non-singlet quark currents in the RI' scheme**", *Nucl. Phys.* **B662** (2003) 247–278, hep-ph/0304113.

[111] J. A. Gracey, "**Three loop anomalous dimension of the second moment of the transversity operator in the MS-bar and RI' schemes**", *Nucl. Phys.* **B667** (2003) 242–260, hep-ph/0306163.

[112] **QCDSF** Collaboration, M. Gockeler, R. Horsley, D. Pleiter, P. E. L. Rakow, and G. Schierholz, "**A lattice determination of moments of unpolarised nucleon structure functions using improved Wilson fermions**", *Phys. Rev.* **D71** (2005) 114511, hep-ph/0410187.

[113] K. G. Chetyrkin and A. Rétey, "**Three-loop three-linear vertices and four-loop MOM beta functions in massless QCD**", hep-ph/0007088.

[114] J. A. M. Vermaseren, S. A. Larin, and T. van Ritbergen, "**The 4-loop quark mass anomalous dimension and the invariant quark mass**", *Phys. Lett.* **B405** (1997) 327–333, hep-ph/9703284.

[115] J. A. Gracey, "**Three loop MS-bar tensor current anomalous dimension in QCD**", *Phys. Lett.* **B488** (2000) 175–181, hep-ph/0007171.

[116] M. Göckeler *et al.*, "**A determination of the Lambda parameter from full lattice QCD**", *Phys. Rev.* **D73** (2006) 014513, hep-ph/0502212.

[117] P. Huber, "**Renormalization factors of quark bilinears using the DCI operator with dynamical quarks**", *JHEP* **11** (2010) 107, 1003.3496.

[118] M. L. Paciello, S. Petrarca, B. Taglienti, and A. Vladikas, "**Gribov noise of the lattice axial current renormalization constant**", *Phys. Lett.* **B341** (1994) 187–194, hep-lat/9409012.

[119] L. Giusti, M. L. Paciello, C. Parrinello, S. Petrarca, and B. Taglienti, "**Problems on lattice gauge fixing**", *Int. J. Mod. Phys.* **A16** (2001) 3487–3534, hep-lat/0104012.

[120] L. Giusti, S. Petrarca, B. Taglienti, and N. Tantalo, "**Remarks on the gauge dependence of the RI/MOM renormalization procedure**", *Phys. Lett.* **B541** (2002) 350–355, hep-lat/0205009.

[121] S. Aoki and Y. Taniguchi, "**One loop renormalization for the axial Ward-Takahashi identity in domain-wall QCD**", *Phys. Rev.* **D59** (1999) 094506, hep-lat/9811007.

[122] J.-R. Cudell, A. Le Yaouanc, and C. Pittori, "**Pseudoscalar vertex, Goldstone boson and quark masses on the lattice**", *Phys. Lett.* **B454** (1999) 105–114, hep-lat/9810058.

[123] J. R. Cudell, A. Le Yaouanc, and C. Pittori, "**Large pion pole in Z(S)(MOM)/(Z(P)(MOM) from Wilson action data**", *Phys. Lett.* **B516** (2001) 92–102, hep-lat/0101009.

[124] L. Giusti and A. Vladikas, "**RI/MOM renormalization window and Goldstone pole contamination**", *Phys. Lett.* **B488** (2000) 303–312, hep-lat/0005026.

[125] D. Becirevic *et al.*, "**Renormalization constants of quark operators for the non-perturbatively improved Wilson action**", *JHEP* **08** (2004) 022, hep-lat/0401033.

[126] J. Noaki *et al.*, "**Non-perturbative renormalization of bilinear operators with dynamical overlap fermions**", *Phys. Rev.* **D81** (2010) 034502, 0907.2751.

[127] M. F. L. Golterman and K.-C. Leung, "**Applications of Partially Quenched Chiral Perturbation Theory**", *Phys. Rev.* **D57** (1998) 5703–5710, hep-lat/9711033.

[128] I. R. . Development, "**Supercomputers in a sugar cube; Energy efficiency and miniaturization thanks to water-cooled chips**", events/gruenes_rechenzentrum/CeBIT_2011_Energy_Efficiency_ExecSummary_MK.pdf.

[129] L. Y. Glozman, "**QCD symmetries in excited hadrons**", 0710.0978.

[130] J. C. Nacher *et al.*, "**Chiral unitary approach to the N* N* pi, N* N* eta couplings for the N*(1535) resonance**", *Nucl. Phys.* **A678** (2000) 187–211, nucl-th/9906018.

[131] L. Y. Glozman, "**Parity doublets and chiral symmetry restoration in baryon spectrum**", *Phys. Lett.* **B475** (2000) 329–334, hep-ph/9908207.

[132] D. Jido, "**Lambda(1405) and kaonic few-body states in chiral dynamics**", 1103.2592.

[133] L. Y. Glozman, "**Alternative experimental evidence for chiral restoration in excited baryons**", *Phys. Rev. Lett.* **99** (2007) 191602, 0706.3288.

[134] C. Gattringer, L. Y. Glozman, C. B. Lang, D. Mohler, and S. Prelovsek, "**Derivative sources in lattice spectroscopy of excited mesons**", *Phys. Rev.* **D78** (2008) 034501, 0802.2020.

[135] C. Gattringer, L. Y. Glozman, C. B. Lang, D. Mohler, and S. Prelovsek, "**Meson spectroscopy with derivative quark sources**", *PoS* **LAT2007** (2007) 123, 0709.4456.

[136] M. Gockeler *et al.*, "**Renormalisation of composite operators in lattice QCD: perturbative versus nonperturbative**", *PoS* **LATTICE2010** (2010) 228, 1010.1360.

[137] M. Gockeler *et al.*, "**Perturbative and Nonperturbative Renormalization in Lattice QCD**", *Phys. Rev.* **D82** (2010) 114511, 1003.5756.

[138] C. Gattringer, M. Göckeler, P. E. L. Rakow, S. Schaefer, and A. Schäfer, "**A comprehensive picture of topological excitations in finite temperature lattice QCD**", *Nucl. Phys.* **B618** (2001) 205–240, hep-lat/0105023.

Acknowledgements

During the past three and a half years that I spent on this Thesis I was able to gain lots of experience in a number of interesting fields in research. I looked into supercomputing, designed, built, interleaved hardware, logic and software and also enjoyed working in this exciting ever-growing field of technology. I would like to thank my professors Andreas Schäfer and Tilo Wettig for giving me the opportunity to be part of the big enterprise called QPACE. The success of this project was only possible due to a great industrial partner. I would like to thank all the people at IBM that worked so hard to make this project become true.

Special thanks go to my University coworkers Dirk Pleiter, Nils Meyer, Stefan Solbrig and many more, since they made the hard work that we put into this project enjoyable.

I want to say thanks to Tommy Burch, Meinulf Göckeler and Christian Lang for helpful advice and interesting discussions on the Variational Method, as well as Lattice Renormalization and results.

Finally, I want to say the greatest thanks to my girlfriend Elli for her patience each and every time I was distracted from life.

Regensburg, May 2011
Thilo Maurer