Otto-von-Guericke-University Magdeburg
Faculty of Electrical Engineering and Information Technology
Chair for Hardware-Oriented Technical Computer Science

Bachelor Thesis

FPGA-based slow control data concentrator for an MRI setup

Version 1.00, December 2025

submitted: March 1, 2026

by: Daniel Anders (Matrikel Nummer: 242295)
born on November 22, 2003
in Lutherstadt Wittenberg, Germany

Abstract

Slow Control Systems (SCS) play a critical role in long-term monitoring and protection of distributed safety-critical systems for scientific, industrial and medical applications. While software-based solutions provide flexibility and scalability, they suffer from aging and non-deterministic latencies, limiting their ability to respond to detected subsystem failure on the long term. This bachelor thesis proposes an FPGA-centric, resource-efficient and extendable Slow Control Interlock System (SCIS) able to monitor and analyze metrics collected from various devices. A VHDL-based priority-aware data concentrator implemented on the Cologne Chip GateMate™ M1A1 FPGA platform compares metrics to known thresholds, achieves nanosecond-scale interlock responses and schedules collected telemetry data for UDP based transmission using a W5500 Ethernet module towards a main slow control server. A Metric Packet format has been defined, standardizing packets containing sensor values into a 9 byte structure and allowing for unique device identification. An AXI-Stream based infrastructure ensures modularity and enables low-latency processing for high priority packets. Threshold-based interlock assertion occurs deterministically one clock cycle after receiving the Metric Packet's last byte while external interlock assertion is registered after four clock cycles due to glitch filtering. Implementation experiments revealed sufficient maturity of the OSS-CAD-Suite for VHDL design flows and effectiveness in designing custom FIFOs to target native BRAM configurations for efficient physical memory utilization. A measured FPGA reconfiguration time of 36.17 ms favors the FPGA data concentrator over CPU based solutions in high-radiation low-downtime applications such as the long-term monitoring of MRI systems by resolving aging effects. Finally, the proposed data concentrator architecture has been implemented into the slow-control software infrastructure, consisting of a Python-based server collecting UDP Metric Packets, Prometheus as a pull-based time-series database and Grafana for dashboard-based analysis of telemetry. This solution aims to enhance the efficiency, reliability and maintainability of MRI slow control systems while reducing complexity in troubleshooting subsystem failures.

Acknowledgements

I would like to thank M.Sc. Vitalii Burtsev for his invaluable expertise and guidance in the field of Slow Control Systems in high-energy physics experiments, which greatly shaped the direction and depth of this thesis.

I would also like to express my gratitude towards Dr. Daniele Passaretti for his thoughtful advice on academic writing and research, as well as the opportunity to contribute to a conference paper, which proved to be an enriching experience alongside this work.

Original Task Assignment:

Aufgabenstellung für eine Bachelorarbeit

Daniel Anders

Matrikelnummer: 242295

Thema

FPGA-based slow control data concentrator for an MRI setup

Aufgabenstellung

Field Programmable Gate Arrays (FPGAs) are widely used in a variety of applications due to their reconfigurability and flexibility advantages. In Magnetic Resonance Imaging (MRI) systems, various subsystems require precise coordination and data exchange to ensure proper operation. This thesis addresses the challenge of developing a cost-effective and efficient data concentrator for MRI slow control systems using the Cologne Chip FPGA platform and W5500 Ethernet module, providing a streamlined approach to managing communication between different MRI subsystems

The task includes the following:

1. Understand the toolchain flow and development approaches for the Cologne Chip FPGA.
2. Develop HDL code for the Cologne Chip FPGA to initialize the network interface using the W5500 Ethernet module.
3. Develop an AXI-Stream-based infrastructure for transmitting data between components.
4. Develop a round-robin arbitration component for AXI-Stream peripheral interfaces.
5. Validate the capabilities of sending and receiving data.
6. Analyse the reliability of data transmission over time.
7. Document the design, code, setup and configurations for future reference and reproducibility.

Magdeburg, 05.01.2026

Tag der Ausgabe: 05.01.2026

Tag der Abgabe: 30. MRZ. 2026

Erstprüfer: Prof. Dr.-Ing Thilo Pionteck

Zweitprüfer: M. Sc. Vitalii Burtsev

T. Pionteck

Erstprüfer

Prof. Dr.-Ing. Roberto Leidhold
(Vorsitzender des Prüfungsausschusses)

Declaration by the candidate

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been marked.

The work has not been presented in the same or a similar form to any other testing authority and has not been made public.

I hereby also entitle a right of use (free of charge, not limited locally and for an indefinite period of time) that my thesis can be duplicated, saved and archived by the Otto von Guericke University of Magdeburg (OvGU) or any commissioned third party (e. g. *iParadigms Europe Limited*, provider of the plagiarism-detection service "Turnitin") exclusively in order to check it for plagiarism and to optimize the appraisal of results.

Magdeburg, March 1, 2026

Contents

Nomenclature

B	byte, 8 bits
b	bit, binary digit
$0x$	Hexadecimal prefix (e.g., 0xFF)
K	Binary kilo; a multiplier of 2^{10} or 1,024 (e.g., 2K = 2,048)
QM.N	Fixed-point number format; M bits represent integer part; N bits represent fractional part

List of Acronyms

ARP Address Resolution Protocol

AXI Advanced Extensible Interface

AXIS AXI-Stream

BRAM Block Random Access Memory

BSB Block Select Bits

CPE Cologne Programmable Element

CPU Central Processing Unit

CRC Cyclic Redundancy Check

DAQ Data Acquisition System

ECC Error-Correcting Code

FCFS First-Come-First-Serve

FIFO First In - First Out Module

FPGA Field Programmable Gate Array

FSM Finite State Machine

HDL Hardware Description Language

HPF Highest-Priority-First

ICMP Internet Control Message Protocol

ISA Instruction Set Architecture

JTAG Joint Test Action Group

LUT Lookup Table

MRI Magnetic Resonance Imaging

OSI Open Systems Interconnection

PLL Phase Locked Loop

RR Round-Robin

RTL Register Transfer Level

SBC Single Board Computer

SCIS Slow Control Interlock System

SCS Slow Control Systems

SDP Simple Dual Port

SLP Super Low Power

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

TCP Transmission Control Protocol

TDP True Dual Port

TID Total ionizing dosage

TMR Triple-Modular Redundancy

UDP User Datagram Protocol

USB Universal Serial Bus

VHDL Very High Speed Integrated Hardware Description Language

List of Figures

List of Tables

1 Introduction

The long-term monitoring of high-energy physics experiments and large medical devices through the use of Slow Control Systems (SCS) provides protection for distributed and safety-critical hardware [1]. In Magnetic Resonance Imaging (MRI) systems, various subsystems require precise coordination and data exchange to ensure proper operation [2]. In case of a subsystem failure inside complex medical devices, troubleshooting can become time-intensive as the source of the interlock signal is often unknown. Subsystem measurements are beneficial to identifying fault sources, as they provide information about system stability and prevent prolonged downtime or risk escalation [3]. The monitoring of critical parameters such as cryogenic temperatures, liquid helium levels and magnet stability such as instantaneous reactions to interlock signals are required for the MRI machine to be operated safely [3]. As metrics[1] are collected from various distributed sources, a data concentrator system is needed to forward telemetry[2] towards a more central infrastructure for storage and metric analysis.

While traditional software-based SCS solutions offer flexibility, they suffer from software aging and non-deterministic latencies, limiting their long-time reliability and require scheduled reboots leading to non-negligible downtime [4]. FPGAs are widely used in applications exposed to increased amounts of high-ionizing radiation due to their reconfigurability, enabling the implementation of fault-tolerance techniques such as Triple-Modular Redundancy (TMR) using a Majority Voting System [5]. Unlike Central Processing Unit (CPU)-based solutions, FPGAs allow for fast error recovery due to their deterministic reconfiguration time in the range of milliseconds [6].

This thesis addresses the challenge of developing a cost-effective and lightweight data concentrator applicable to an MRI Slow Control Interlock System (SCIS) utilizing the European Cologne Chip GateMate™ M1A1 FPGA, manufactured by Globalfoundries using a 28-nm Super Low Power (SLP) node in Germany [7]. In order to offload the complexity derived by an IP Ethernet stack potentially occupying most of the resources in small FPGAs, the W5500 platform has been chosen to implement lightweight UDP-based Ethernet connectivity on the FPGA fabric. To process, log and visualize metrics measured inside an MRI machine, the setup of a software infrastructure is needed. In addition, this

[1] A measurable and quantifiable system parameter captured at runtime; e.g. temperature, pressure or voltage

[2] Raw data collected from a system, including metrics and events

work aims to evaluate maturity of the open-source toolchain for Very High Speed Integrated Hardware Description Language (VHDL)-driven development on the GateMate™ FPGA platform and the efficiency of targeting Block Random Access Memory (BRAM) primitives to improve memory resource utilization.

1.1 Research Objectives

The primary objective of this thesis is to develop a resource efficient and extendable SCIS utilizing an FPGA data concentrator that overcomes non-deterministic latencies and software aging as found in CPU-based solutions. The data concentrator, described using VHDL, should be able to communicate over a network interface implemented using the W5500 Ethernet module. A deterministic and low-latency logic should monitor incoming metrics and decide on interlock assertion based on metric value comparisons to predefined thresholds. A general round-robin arbitration module for Advanced Extensible Interface (AXI)-Stream peripheral interfaces is needed to make the data concentrator extendable. This work evaluates data sending and receiving capabilities, long-term reliability, extendability and system performance by testing the FPGA hardware design. In addition, the OSS-CAD-Suite workflow maturity regarding GateMate™ FPGA will be discussed for VHDL-based hardware design. An implementation analysis will discuss post-implementation resource utilization and FPGA reconfiguration time on the Cologne Chip platform.

1.2 Thesis Structure

This thesis is organized into eight chapters. Following the introduction, Chapter 2 establishes the theoretical foundation while Chapter 3 reviews related literature. The system design and implementation of the proposed system are described in Chapter 4 and 5. Chapter 6 evaluates overall system performance, hardware utilization and Serial Peripheral Interface (SPI) signal quality as well as a comparison of resource utilization against an Artix-7 implementation. Finally, Chapter 7 will conclude this thesis, followed by a future outlook in Chapter 8.

1.3 Source code

The source code for the data concentrator, W5500 Controller and SPI Master is available on GitHub [8], including a Makefile automating the synthesis, implementation and programming for the Cologne Chip GateMate™ M1A1 FPGA Evaluation Board. In addition, the GitHub repository contains the Prometheus configuration file, the time measurements for FPGA reconfiguration time such as the testing scripts used during system evaluation.

2 Theoretical Background

In order to understand design choices made in this thesis this chapter goes over fundamental concepts and technical constraints shaping this work. More specifically, understanding limitations in software-based monitoring systems and the advantages of using FPGAs in high-field environments should further undermine the transition to an FPGA-based data concentrator.

2.1 Slow Control Systems

Slow Control Systems refer to the non-time critical and long-time monitoring of hardware, including high-voltage modules, cryogenic systems, temperature sensors and pressure gauges [9]. SCS are prominent in industrial applications, large medical devices or high-energy physics experiments, where a large amount of subsystems, often from various hardware vendors, provide metrics in need of monitoring to ensure stable operation of the complete system [10]. Subsystem sensor readings can differ both in frequency and priority. In case of an MRI system, where monitoring of safety-critical parameters such as cryogenic levels or liquid helium temperature is strictly mandatory [3], the handling of safety-critical metrics should be prioritized according to subsystem failure severity (refer to Section 2.6). SCS inherently don't have high bandwidth requirements, as typical sensor update rates in high-energy physics experiments are specified in updates per seconds [11] or even minutes [12]. Individual sensor readings rarely exceed a few bytes of payload, hence the W5500 Ethernet platform (refer to Section 2.5) has been chosen for cost-optimized and resource effective Ethernet connectivity on the Cologne Chip GateMate™ M1A1 FPGA (Section 2.4.1). A SCIS is an extension of regular SCS and adds an immediate safety action, an interlock assertion, to its monitoring duties performed locally by a data concentrator (Section 2.2).

2.2 Data Concentrator

A data concentrator is part of a larger management system and responsible for collecting and managing information from various data sources or Data Acquisition Systems (DAQs). As different subsystems provide telemetry at different volume and priority, the data concentrator needs to temporarily buffer and schedule the processing of received metrics before transporting telemetry, including alerts, towards a more central system for further processing. Data concentrators are well suited for the local implementation of low-latency

parameter monitoring, enabling interlock assertion to prevent risk-escalation by prohibiting system operation in case unsafe operational parameters are detected. Smart grid systems utilize data concentrators, by collecting information from smart meters, that need to relay theirs readings towards more centralized utility servers [13]. To achieve multi-vendor operability, adapters can be used to convert incoming subsystem data streams to be compatible with the interface specification of the data concentrator.

2.3 Software Aging

The tendency of long-running software systems to degrade in performance and become more prone to errors is referred to as software aging [4]. With aging-related problems being often caused by software faults, they can also be the result of classical error propagation and error accumulation. CPU-based solutions typically operate on top of an operating system, where hardware resources need to be shared with other processes. Faults like memory leaks, physical memory fragmentation or data corruption are known to affect software reliability as the system's runtime increases [4]. Though volatile errors like operating system related resource leaks can be resolved by system reboots; this rejuvenation process causes significant downtimes in software-based monitoring of safety-critical parameters.

2.4 FPGA

An FPGA is a user programmable and integrated circuit to develop custom logic and perform digital operations [14]. Its ability to perform parallel data processing sets it apart from Instruction Set Architecture (ISA)-based systems relying on a CPU that executes instructions sequentially. Programming FPGAs is different from general purpose processors, as they don't execute compiled instructions, but require the synthesis and implementation of a desired hardware design. Though options for high-level synthesis of specialized C/C++ code exist for FPGAs by Altera and Xilinx, the GateMate™ M1A1's open-source software stack currently only supports the synthesis of Hardware Description Languages (HDLs) such as Verilog or VHDL.

2.4.1 Cologne Chip GateMate™ M1A1 FPGA

Cologne Chip uses the OSS-CAD-Suite, an open-source binary software distribution used in digital logic design and maintained by YosysHQ [15]. It provides tools for various hardware design languages, Register Transfer Level (RTL) synthesis, formal hardware verification, implementation (place and route) and FPGA programming. More details on the tools used for simulation, synthesis, implementation and bitstream packing will be covered in Section 5.6.

The workflow for the Cologne Chip FPGA Evaluation Board has been documented in a GitLab repository [16] on the example of implementing the FPGA W5500 Controller. Cologne Chip also provides a quickstart guide for usage of the GateMate™ toolchain [17].

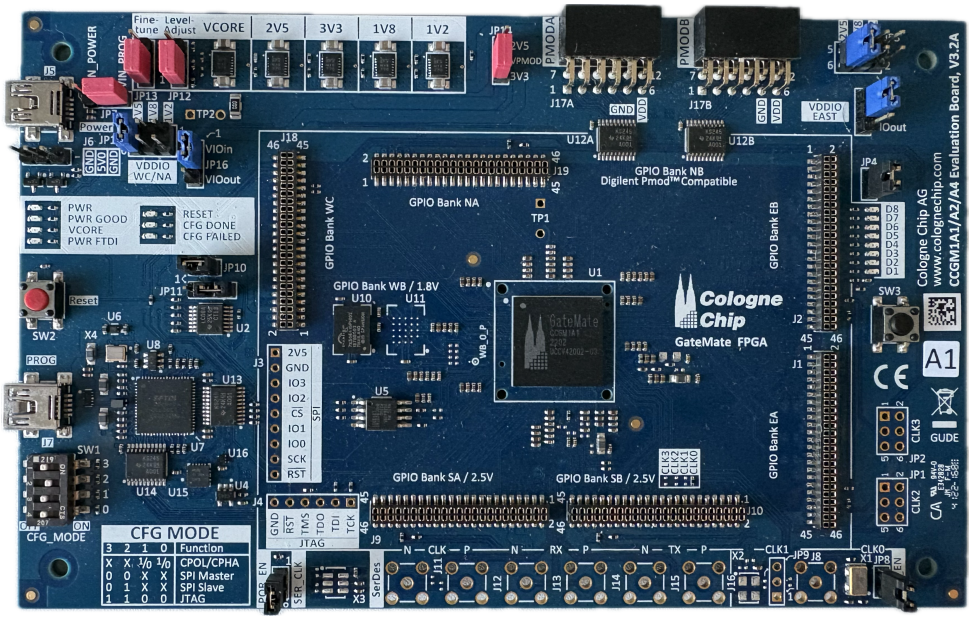


Figure 2.1: Cologne Chip GateMate™ M1A1-E1 Evaluation Board V3.2A

The GateMate™ M1A1 FPGA chip, available on an evaluation board (2.1), is designed by Cologne Chip and manufactured by Globalfoundries using the 28 nm SLP node in Germany [7]. It contains 20480 Cologne Programmable Elements (CPEs) arranged in a 160x128 matrix and 32 dual-port BRAM cells [18]. Each physical 40K BRAM is separable into two independent 20K BRAM blocks, usable in Simple Dual Port (SDP) or True Dual Port (TDP) mode. The GateMate™ Primitives Library [19] lists BRAM configurations, with data widths ranging from single bit values up to 40 bits in TDP mode or 80 bits (SDP). A Phase Locked Loop (PLL) inside the CCGM1A1 FPGA chip provides up to four system clock signals for the implemented design. The CCGM1A1 FPGA Chip can operate in the temperature range of $-40\,°\mathrm{C}$ up to $125\,°\mathrm{C}$, making it even suitable for usage in harsher environmental conditions. Total ionizing dosage (TID) tests at the CERN Highly-Accelerated Mixed Field Facility (CHARM) have shown, that the GateMate™ M1A1 is robust against degradation up to $1\,\mathrm{kGy} = 1000\frac{\mathrm{J}}{\mathrm{kg}}$ of ionizing radiation, which makes it both comparable to other 28 nm-based FPGA architectures and viable for usage in high-radiation environments [20,21]. The Cologne Chip M1A1 Evaluation board has two PMOD ports and multiple GPIO banks allowing for connecting multiple peripheral devices.

The GateMate™ FPGA Datasheet states, that the Evaluation Board is configurable from four different sources listed in the table 2.1 using the `CFG_MODE` switches (seen in Figure 2.1) [7].

Table 2.1: GateMate™ FPGA configuration sources

Configuration Source	Description
SPI Active Mode	FPGA reads autonomously from external flash (SPI master)
SPI Passive Mode	External device programs FPGA via SPI (FPGA is slave)
JTAG	Programming via JTAG test & programming adapter
User Logic	User-configured logic feeds configuration data to FPGA itself

Both active and passive SPI modes are configurable in the clock polarity (`CPOL`) and clock phase (`CPHA`). The FPGA board features an integrated FT2232H Universal Serial Bus (USB) interface chip, which provides dual channel USB JTAG/SPI connectivity and enabling direct bitstream upload from a host computer via a single USB connection. Bitstream uploaded using the SPI mode is written towards a Macronix MX25R6435F 64 Mb flash memory seen as `U5` in Figure 2.1. This memory chip can be used to reconfigure the FPGA in milliseconds, depending on the size of the bitstream and configured SPI mode.

Cologne Programmable Element (CPE)

A CPE (Figure 2.2) is small addressable building block of the larger FPGA fabric and contains a Lookup Table (LUT)-tree, two multiplexer and two flip-flops/latches, such as additional logic for selection and fast signal path routing. Depending on the configuration, a CPE can take the function of either a dual 4 input LUT-tree, a single 8 input LUT-Tree, 6 inputs with a 4 input multiplexer, an 1- or 2-bit full-adder with expandable arrangement or an expandable 2·2-bit multiplier [19].

Block Random Access Memory (BRAM)

Block RAM refers to dedicated on-chip memory blocks within an FPGA, providing efficient storage for memory-intensive applications such as First In - First Out Modules (FIFOs) or large LUTs, that would otherwise be implemented resource inefficiently using the limited amount of LUTs or flip-flops. The GateMate™ M1A1 FPGA chip has physically embedded and volatile memory cells configurable as either two separate 20 kilobit dual-port Static Random Access Memory (SRAM) cells or a single 40 kilobit S-RAM block [19]. Both BRAM configurations, seen in 2.3 and 2.3, support a TDP mode with the ability to simultaneously access memory using two independent ports, their own clock signal and no read or write restrictions. SDP mode is similar to TDP, but only allows for simultaneous write operations on one hardware port and read operations on the other. For BRAM directly instantiated as `CC_BRAM_20K` or `CC_BRAM_40K` primitives with the appropriate data width configuration (as seen in Table 2.2 and 2.3), the GateMate™ FPGA's built-in

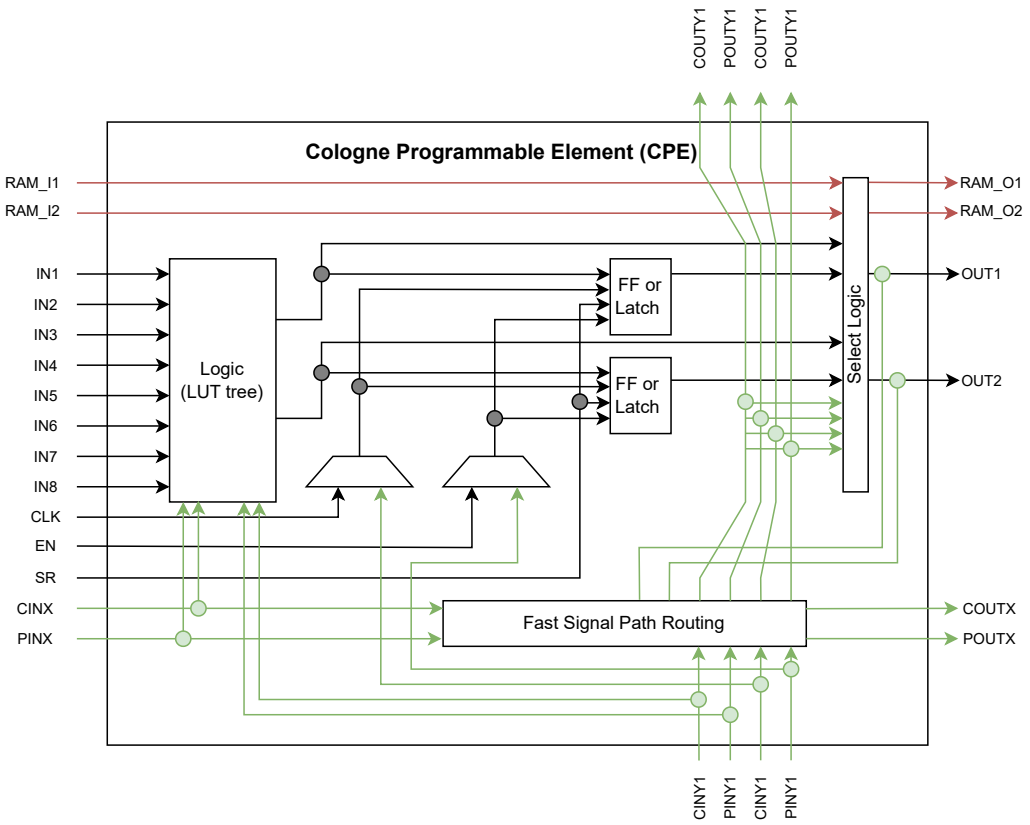


Figure 2.2: Diagram of a Cologne Programmable Element [19]

hardware dedicated for error checking and correcting becomes available. By storing 32 bit words in BRAM cells at a data width of 40 bits, the remaining 8 bits can be used to store Hamming-code parity information, allowing for correction of single bit errors and detection of two bit errors. When Error-Correcting Code (ECC) is enabled, the parity generation and checking is performed automatically by the hardware and delays validity of data by one clock cycle. ECC status flags are provided as dedicated output ports for single-bit and double-bit error detection. However, ECC cannot be inferred from generic behavioral VHDL and becomes unavailable for unsupported data widths.

Cologne Chip provides HDL templates for data- and address-width adjustable BRAM primitives in their primitives library [19], that is first elaborated as a memory block ($mem) by the Yosys synthesis tool and then efficiently mapped to the GateMate™ M1A1's hardware blocks using the nextpnr implementation (place and route) tool.

Table 2.2 and 2.3 show all possible configurations for 20K or 40K RAM Blocks in the GateMate™ M1A1 FPGA chip [19].

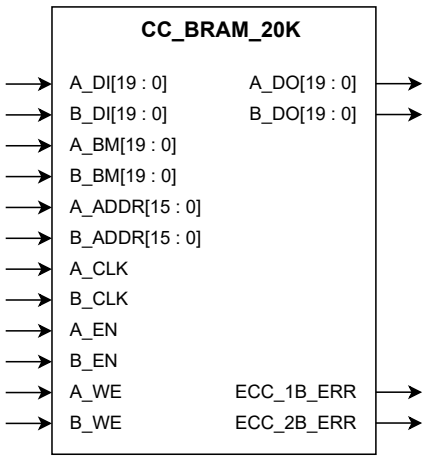


Figure 2.3: 20K BRAM primitive [19]

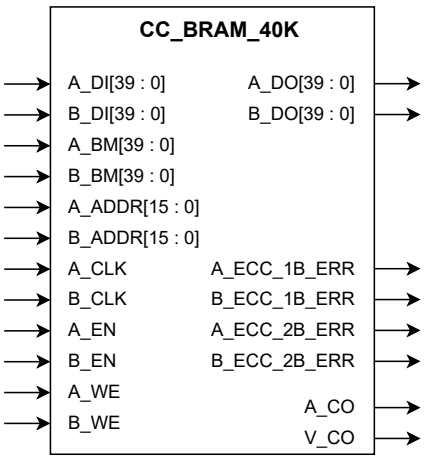


Figure 2.4: 40K BRAM primitive [19]

Table 2.2: RAM configurations per 20K Block

Configuration	TDP	SDP	ECC
16K · 1 bit	✓	✓	×
8K · 2 bit	✓	✓	×
4K · 5 bit	✓	✓	×
2K · 10 bit	✓	✓	×
1K · 20 bit	✓	✓	×
512 · 40 bit	×	✓	✓

Table 2.3: RAM configurations per 40K Block

Configuration	TDP	SDP	ECC
32K · 1 bit	✓	✓	×
16K · 2 bit	✓	✓	×
8K · 5 bit	✓	✓	×
4K · 10 bit	✓	✓	×
2K · 20 bit	✓	✓	×
1K · 40 bit	✓	✓	✓
512 · 80 bit	×	✓	✓

2.4.2 ARM AMBA AXI-Stream

AXI-Stream (AXIS) is a synchronous unidirectional high-throughput streaming interface and part of ARM's AMBA family of on-chip communication protocols. It was chosen as the primary communication protocol due to its ability to support fast, low-latency data transfers through continuous streams. Unlike memory-mapped AXI4, AXIS does not use addresses as it is designed for continuous data transfer from a master to a slave rather than read/write accesses to memory-mapped locations.

A minimal AXIS interface consists of a data bus, a clock signal and the single-bit handshake signals `VALID` and `READY`. The `VALID` signal is controlled by the master and shows, that the payload unit on `DATA` is readable. `READY` is the only signal asserted by the slave and signals the readiness to accept a payload-unit on the data bus. A payload is written to the data bus on the falling edge and read on the rising edge of the clock signal. A payload unit of data is successfully transferred, when the valid and ready signal are high at the rising edge of the clock [22].

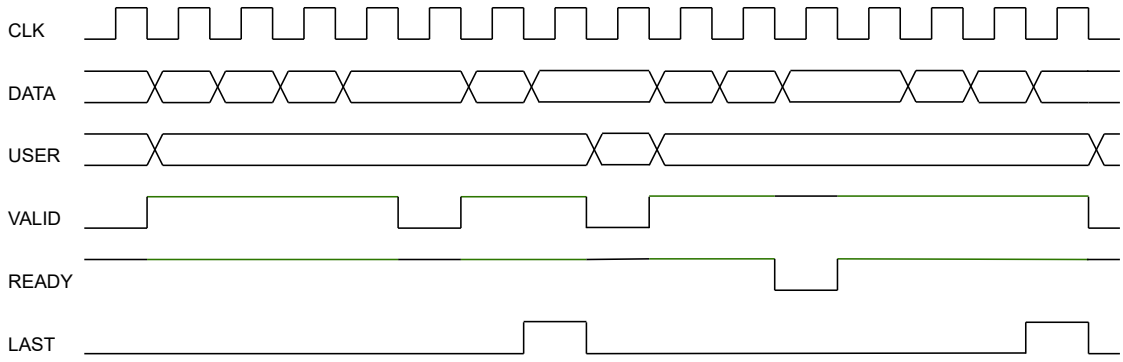


Figure 2.5: AXIS transaction of two 6 byte data packets with USER field and LAST signal; green lines on VALID and READY indicating a successful handshake

This concept can be extended by a `USER`-field and a single-bit `LAST` signal as seen in Figure 2.5. The `USER`-bus carries application specific information alongside every data word while the last bit is asserted at the last word of a data packet.

2.5 W5500 Ethernet Module

The W5500 chip is a hardwired and SPI-based Ethernet controller, that enables internet connectivity by providing an integrated Transmission Control Protocol (TCP)/Internet Protocol (IP) stack. It supports UDP, TCP, IPv4, Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP) while allowing for up to 8 sockets with a total of 32 kilobyte buffer memory [23]. The W5500 offloads Open Systems Interconnection (OSI) Layers 1-4 [24] Physical (Ethernet PHY), Data Link (MAC/ARP), Network (IP/ICMP) and Transport (UDP/TCP) entirely in hardware, allowing controllers to configure UDP/TCP sockets through register interactions instead of running a software TCP/IP stack.
A W5500 Controller written in VHDL already exists for Xilinx FPGA boards such as the PYNQ-Z1, that can push the W5500 to its limit of around 50 Mbit/s on UDP sockets. This hardware design however contains AXIS data FIFO IP-cores with inaccessible source code as they are the intellectual property of Xilinx [25], which is why custom FIFOs had to be designed for the Cologne Chip FPGA. Changes were made to the existing FPGA W5500 Controller, such that it can now utilize all 8 sockets for UDP data exchange and put emphasis on either receiving or transmitting data first (Section 5.2).

With the W5500 chip supporting high-speed SPI signals up to 80 MHz, the finite rise and fall times of digital signals transmitted over interconnecting wiring must be considered, as they determine the effective signal bandwidth rather than serial clock frequency alone [26]. At these speeds, parasitic capacitance and inductance primarily influence signal integrity by limiting edge rates and introducing waveform distortions during signal transitions.

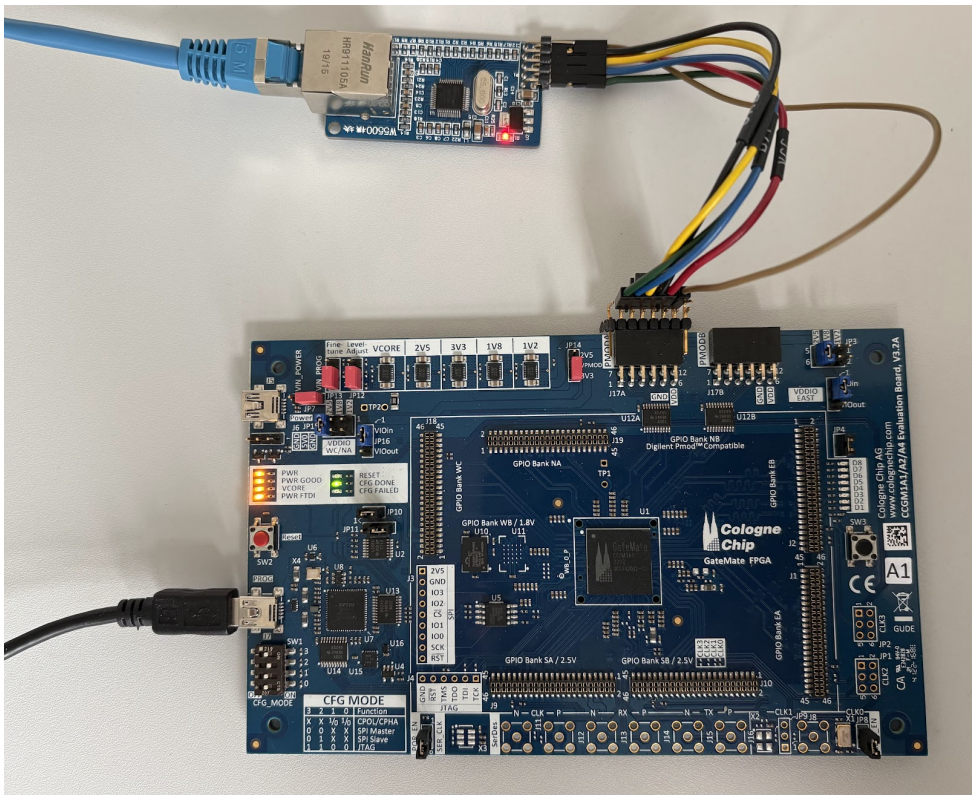


Figure 2.6: A W5500 Ethernet module connected to PMOD-A on the Cologne Chip GateMate™ FPGA Evaluation board; Green glowing LED shows “CGF_DONE” indicating a successfully configured FPGA fabric

Consequently slew rates and drive strengths need to be considered in the constraints file during the implementation stage (Section 5.6.2) to ensure reliable signal transmission.

2.6 Priority-based Scheduling and Arbitration

As subsystem failure severity differs in MRI machines, the order of handling packets containing sensor values is relevant for safety-critical monitoring systems. While scheduling terminology is typically used in the context of task scheduling performed by operating systems, it is applicable to the scheduling of packet processing within a data concentrator. Scheduling in FPGA implementations is different from sequential task scheduling in CPU systems, as some tasks can be parallelized, resulting in increased throughput and reduced need for complex scheduling strategies at the cost of higher FPGA resource utilization. Arbitration and scheduling solutions need to consider throughput limitations and balance FPGA implementation simplicity, fairness among packet sources and processing latency.

In non-preemptive scheduling, once a process starts, it is uninterrupted until completion. This avoids complex-state saving and restoration logic required for preemption in FPGA implementations. The data concentrator has three main responsibilities: collecting

metrics at multiple input channels, low-latency interlock assertion based on parameter monitoring and the scheduling of metrics or alerts for transmission via an Ethernet interface.

First-Come-First-Serve (FCFS) is a strategy optimizing for implementation simplicity and fairness, being used at the data concentrator's input channels. The receiving of data by Threshold Logic performing interlock assertion is designed to operate in parallel.

Round-Robin (RR) arbitration optimizes for fairness among data input-channels by cycling through data sources without favoring one over the other.

While receiving and comparing parameter values to thresholds does not significantly bottleneck data concentrators throughput, W5500-based UDP packet transmission does, as this takes hundreds of clock cycles per telemetry packet, motivating the usage of a simple priority system designed to schedule priority-tagged Metric Packets and alerts for transmission. By using Highest-Priority-First (HPF) scheduling, the average waiting is weighted to favor low latencies for safety-critical telemetry at the expense of low-priority metrics. While HPF favors high-priority packets, it introduces the risk of buffer congestion and starvation, where low-priority metrics are due to a continuous stream of high-priority traffic [27]. In real-time systems, this issue is often addressed through techniques like priority inversion or priority inheritance [28]

2.7 QM.N Fixed Point Values

Fixed point numbers are described using the Q number notation. Like floating point numbers, they approximate real numbers, with the advantage of being less computationally complex. The Q number format is often described using the QM.N notation, where M is the amount of bits storing the integer portion and N the bits describing the fractional part of an approximated number [29]. A mathematical interpretation of the QM.N number format is seen in Figure 2.7.

This will be shown by approximating and storing $x = \pi = 3.14159\ldots$ using a Q4.4 signed fixed-point representation.

A Q4.4 signed fixed-point number gives 8 bits of precision, with the most significant bit encoding the sign. With $N = 4$, the fractional resolution is:

$$\frac{1}{2^N} = \frac{1}{16} = 0.0625$$

$Q_{M.N}$ represents numbers of the form:

$$x = -s \cdot 2^{M-1} + \sum_{i=0}^{M-2} b_i \cdot 2^i + \sum_{j=1}^{N} f_j \cdot 2^{-j}$$

Where:

$s =$ sign bit, $b_i =$ integer bits, $f_j =$ fractional bits.

Figure 2.7: Mathematical interpretation of the QM.N format.

To store x in memory as a fixed point number, a scaling needs to be applied first:

$$x \cdot 2^N = \pi \cdot 16 = 50.2654\ldots$$

Rounding it to the nearest integer gives us the stored integer x_{int}:

$$\text{round}(x) = \text{round}(50.2654\ldots) = 50 = 00110010_{\text{b}}$$

This can be written in QM.N notation:

$$\text{Q4.4 notation: } 0011.0010_{\text{b}}$$

Integer component:

$$0011_{\text{b}} = 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = +3$$

Fractional component:

$$0010_{\text{b}} = 0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} + 0 \cdot \frac{1}{16} = 0.125$$

Thus,

$$3 + 0.125 = 3.125$$

To reconstruct the original number, the stored integer is divided by its scaling:

$$\frac{x_{\text{int}}}{2^N} = \frac{50}{16} = 3.125$$

The rounding error is:

$$|\pi - 3.125| = 0.01659\ldots$$

$$\frac{0.01659\ldots}{\pi} \approx 0.53\%$$

Another advantage of fixed-point numbers is that comparisons in hardware are equivalent to comparing two signed integers. Thus, no additional VHDL libraries are required to check thresholds on fixed-point metric values.

Scaling must be handled by the DAQ constructing Metric Packets and by the Metric Packet Server when converting them to floating-point values. When integer precision is sufficient for sensor readings, scaling can be easily performed in hardware through bit-shifting the integer value by N positions.

3 Related Work

In this chapter prior work related to the development of an FPGA-based data concentrator and interlock system is reviewed. The selected work will also address limitations of software-based monitoring solutions and the radiation tolerance of the target FPGA platform.

Relevant to this implementation is the work presented by the National Synchrotron Radiation Research Center (NSRRC) in Taiwan [30], which describes the development of an FPGA-based front-end interlock safety system. It validates the choice to implement parts of the SCS infrastructure on FPGA fabric due to non-deterministic latencies introduced by software-based solutions.

A related approach is presented by Kolasiński et al. in their development of a High-Performance FPGA Streaming Data Concentrator for GEM Electronic Measurement System for WEST Tokamak [31]. Their work shows the demand for fast, efficient and high-throughput data streaming and data reduction under tight timing conditions on the domain of plasma physics experiments. Although the MRI data concentrator for SCS proposed in this thesis operates under significantly lower bandwidth requirements, both still benefit from pipelined data streaming and early-stage local triggering on the FPGA.

A fundamental challenge regarding software-based architectures has been pointed out by Grottke, Matias and Trivedi [4], who characterize the tendency of long-running software systems to degrade in performance and become increasingly error-prone with time. This issue referred to as software aging, can originate directly from software faults but also from classical error propagation and accumulation over time. CPU-based solutions, which typically operate on top of an operating system layer and must share hardware resources with competing processes, are susceptible to faults like memory leaks, physical memory fragmentation or data corruption. While a system reboot can resolve volatile system faults such as operation system resource leaks, this rejuvenation process introduces significant downtime, raising concerns about the suitability of software-based solutions for the continuous monitoring of safety-critical parameters.

Regarding the suitability of the Cologne Chip GateMate™ M1A1 FPGA chip for usage in radiation-heavy environments, Richard Jung [21] conducted a radiation sensitivity

campaign using proton radiation benchmarks in a Proton Irradiation Facility (PIF) at the Paul Scherrer Institute (PSI). The results indicate that the radiation sensitivity of the CCGM1A1 FPGA chip is comparable to other FPGA microarchitectures using the same 28 nm process technology, suggesting that GateMate™ FPGAs are viable candidates for deployment in the radiation environment of an MRI scanner.

4 System Design

This chapter proposes an FPGA-centric SCIS (Section 4.1) and goes into detail about the data concentrator hardware architecture (Section 4.2). An Ethernet-based approach utilizing a W5500 Ethernet module is shown to implement the data concentrator into SCS. The chapter concludes with an overview on two W5500 configurations in Section 4.3, optimizing for either FPGA resource efficiency or Ethernet throughput.

4.1 Slow Control Interlock System Infrastructure

The SCIS illustrated in Figure 4.1 is primarily intended to relay telemetry collected at different subsystems referred to as Metric Packet sources towards a central host computer, such that subsystem information can be stored, processed, monitored and further analyzed. Metrics can originate from multiple data sources or DAQs, referred to as Metric Sources, providing their sensor readings on various communication interfaces. Because the on-chip data concentrator only supports AXIS as an input interface (Section 5.3.1), communication interfaces like SPI, I^2C or UART require a peripheral controller and potentially an adapter, that not only constructs Metric Packets containing a unique device identifier (shown in Figure 5.6), but also provides these Metric Packets on a compatible AXIS interface.

Both the peripheral controllers and data concentrator are implemented on the Cologne Chip GateMate™ M1A1 FPGA. The data concentrator unit compares metric values from different input channels to predefined thresholds and buffers them temporarily, until a dedicated Transceiver Ethernet Interface, implemented using the W5500 Ethernet platform offloading the TCP/IP stack from FPGA fabric, becomes available to send Metrics Packets as UDP packets towards the software architecture shown in Figure 4.1.

The Metric Packet Server listens for UDP packets sent via an Ethernet interface on multiple ports and exposes an HTTP endpoint, that lists current metric values to be accessible by Prometheus [32], a pull-based time-series database that scrapes metrics at a predefined regular time interval. Grafana is a platform to visualize and analyze data using dashboards [33], being configured to automatically query Prometheus by making API calls to the Prometheus HTTP API. The Grafana dashboard engine, configured to visualize metrics collected from Prometheus, is accessible through a web browser indicated by the clients in Figure 4.1.

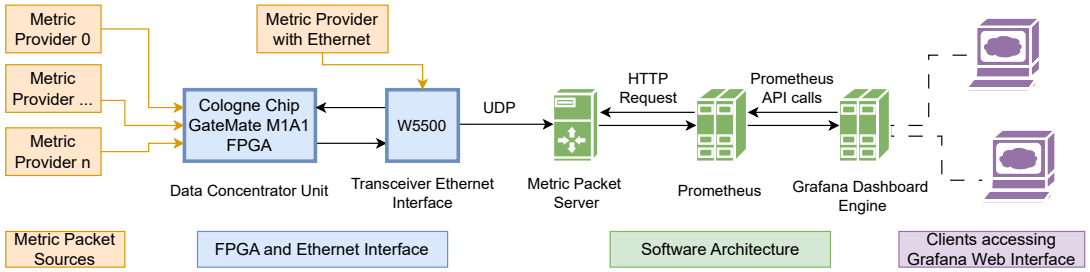


Figure 4.1: Overview of Slow Control Infrastructure, showing metric sources (orange) and the Cologne Chip GateMate™ M1A1 being programmed with bitstream containing the data concentrator and peripheral controllers

To make the overall design extendable, each component has been designed with modularity in mind. The peripheral controller are completely agnostic to the inner workings of the data concentrator and software architecture components communicate via HTTP access points, allowing for distributed hosting of the Metric Packet Server, Prometheus and Grafana.

4.2 Data Concentrator Unit

Figure 4.2 shows the proposed data concentrator design and comes with a configurable amount of AXIS based input channels accepting Metric Packets using peripheral interfaces, with one of them being an Ethernet Interface. Data is first checked by a Threshold Logic for immediate comparisons to local threshold values and interlock assertion if sensor readings exceed safety-critical parameter boundaries. Components indicated in red functionally belong to the interlock handling system.

Processed Metric Packets are forwarded to a Metric Packet Manager, primarily buffering and scheduling the transmission of Metric Packets based on their priority. Within the Metric Packet Manager, a round-robin arbiter cycles through all available Metric Packet input streams and reads the packet priority and configures the data path to lead towards the correct Priority FIFO. Metric Packets are stored in their corresponding Priority FIFO wait until an Ethernet module is ready to send them towards a Metric Packet Server.

Telemetry always flows upstream, meaning that Metric Packets entering the data concentrator and interlock alerts can only be emitted by the Telemetry Sender 4.2, allowing for separation of TX and RX data handling for the Ethernet Interface. This allows for the usage of a separate Ethernet module at the data concentrator's Ethernet Interface TX side specifically dedicated for transmission of UDP packets, effectively increasing the overall bandwidth the data concentrator can operate at. As W5500 Ethernet modules

where chosen to implement Ethernet connectivity, two configurations optimizing for FPGA resource efficiency or Ethernet throughput will be shown in Section 4.3 utilizing an FPGA W5500 Ethernet controller and SPI Master. The data concentrator accepts external interlock signals, removes signal glitches using a digital glitch filter and outputs a global interlock signal for further processing.

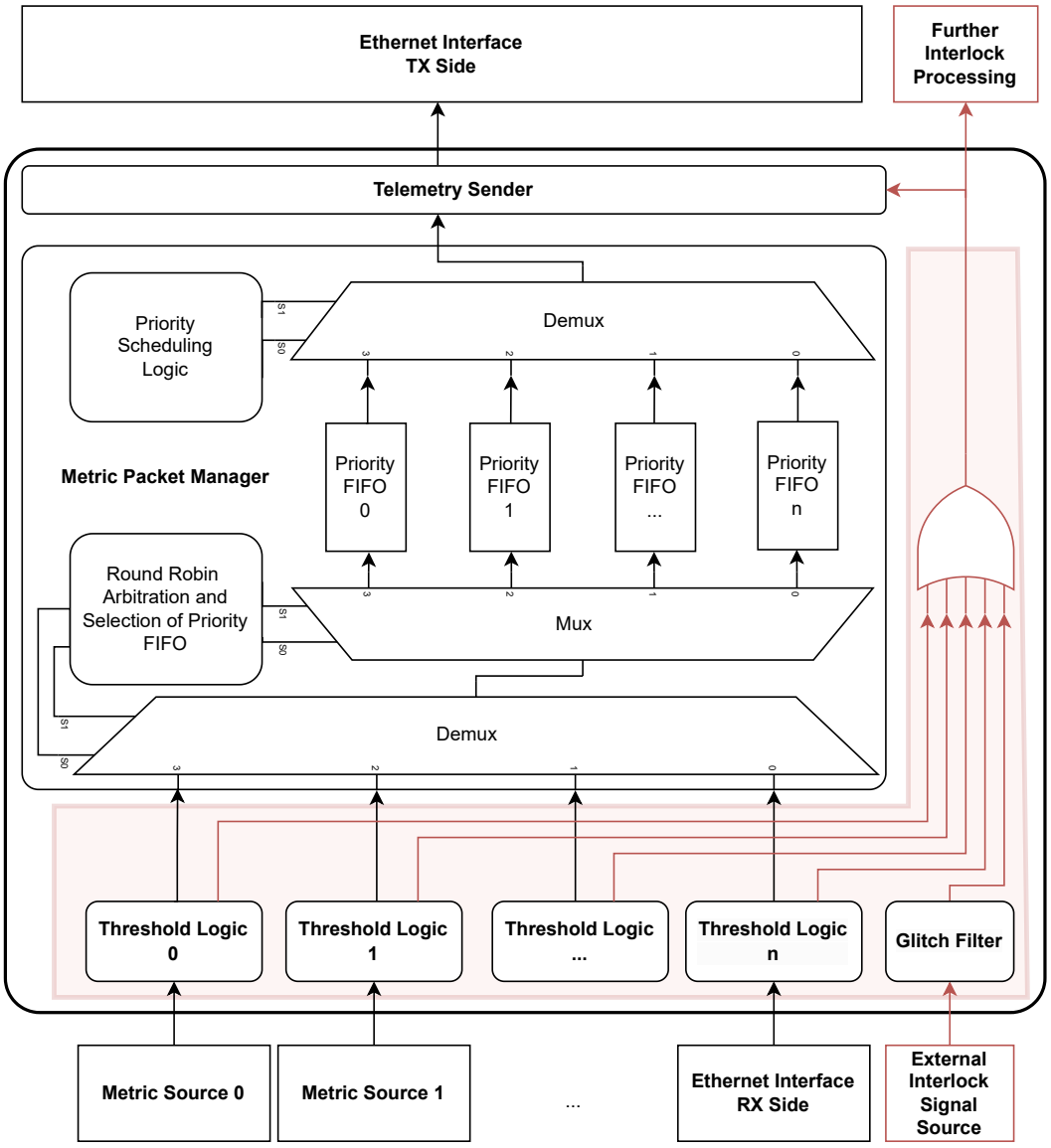


Figure 4.2: Proposed data concentrator hardware design showing data flow of Metric Packets and interlock logic indicated in red; 8 bit data width AXIS used to transport Metric Packets; Red arrows indicate interlock signals

4.3 W5500 Configuration Options

The data concentrator supports two W5500 configurations. In bandwidth and resource constrained scenarios, a single W5500 Ethernet module handles both TX and RX communication with shared throughput (first configuration in Figure 4.3). For higher throughput requirements, a setup for two W5500 modules can be deployed for dedicated TX/RX responsibilities (second configuration in Figure 4.3), effectively doubling the available Ethernet bandwidth. As scheduling between TX and RX duties can become a latency critical factor for interlock assertions, the W5500 Controller supports two scheduling modes to prioritize either receiving or transmission of data. This is implemented and further explained in Section 5.2. For interlock latency aware usage with a single W5500 module, `receive_first` scheduling comes at the advantage of checking all 8 RX socket buffers for available data first, before scheduling the transmission of UDP packets towards the Metric Packet Server.

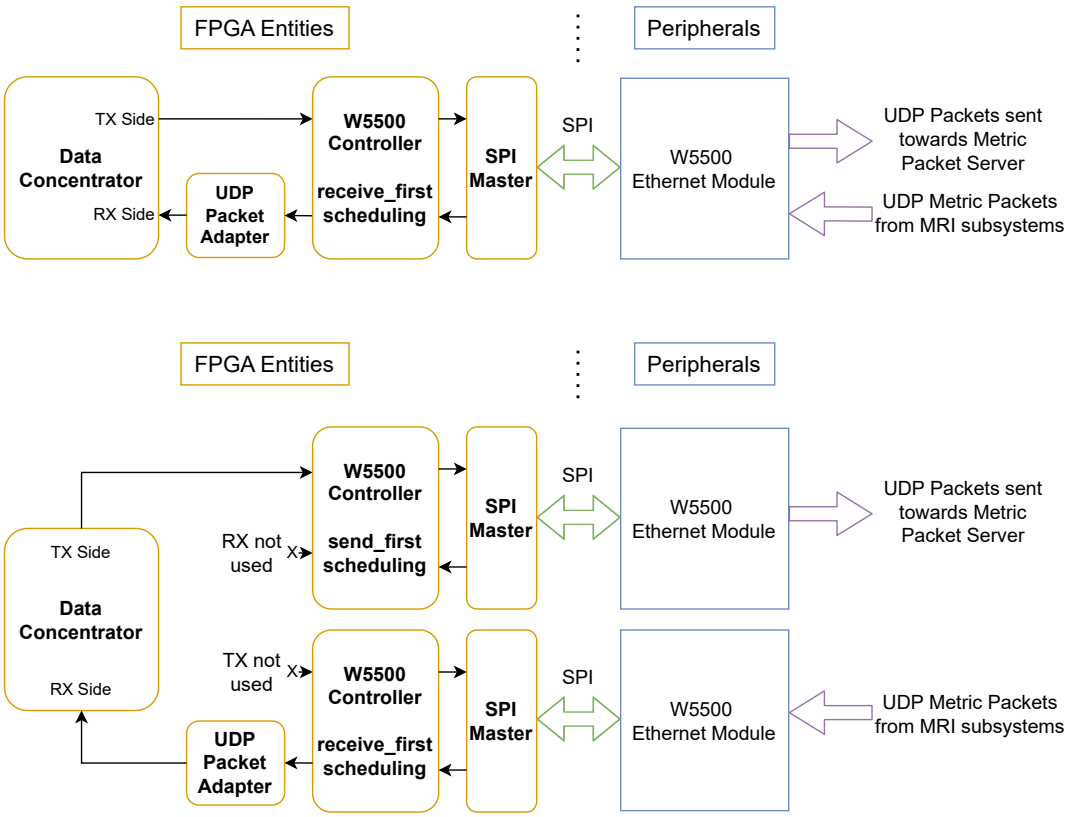


Figure 4.3: Diagram showing two possible W5500 setups for the FPGA data concentrator; First setup shows a single W5500 Ethernet module with sending and receiving responsibilities and shared throughput; Second Setup utilizes two W5500 modules with split TX/RX responsibilities for higher Ethernet throughput; `X` indicates not used

5 Implementation

This chapter details the implementation of the data concentrator proposed in the previous Chapter 4 on the Cologne Chip GateMate™ M1A1 FPGA and the supporting software infrastructure. It begins with an overview of the development workflow and design methodology (Section 5.1), followed by the VHDL implementation of core FPGA components such as the FPGA W5500 Controller and its SPI peripheral controller (Section 5.2), the Metric Packet structure (Section 5.3), the AXI-Stream designed to transport Metric Packets (Section 5.3.1), the UDP Packet Adapter (Section 5.3.2), the Threshold Logic (Section 5.4.1), the Metric Packet Manager (Section 5.4.2) and the Telemetry Sender in Section 5.4.3. The implementation of a digital glitch filter will be shown in Section 5.4.4 and synthesis constraints are covered in the Section 5.6.1. This chapter concludes with the host-sided software infrastructure including the Metric Packet Server (Section 5.7.1), Prometheus as a time series database (Section 5.7.2) and the Grafana dashboard engine in Section 5.7.3.

5.1 Overview and Development workflow

To implement the system design proposed in the previous chapter using VHDL, a development environment is needed, that supports synthesis of VHDL and can target the CCGM1A1 FPGA chip. For this purpose, the OSS-CAD-Suite [15] provides a collection of open-source tools enabling a VHDL/Verilog workflow for the Cologne Chip FPGA. Design principles and methodology for system implementation are covered in Section 5.1.2.

5.1.1 Development Workflow

The adopted design flow follows the current reference workflow for the GateMate ™ FPGA family and combines open-source tools such as Yosys, nextpnr and openFPGAloader as recommended by the device vendor. The OSS-CAD-Suite provides a toolchain, that enables rapid development iterations with full visibility into resource utilization and timing analysis. All data concentrator components including the existing FPGA W5500 Controller and SPI Master are described using VHDL, a hardware description language that is not natively supported by the synthesis tool Yosys [34]. GHDL [35], a VHDL 2008 simulator and analyzer, was used for VHDL testbench simulations and is compatible with GTKwave [36], an included waveform viewer. Yosys has support for a GHDL plugin [37], that adds compatibility for GHDL elaborated designs. To perform logic synthesis

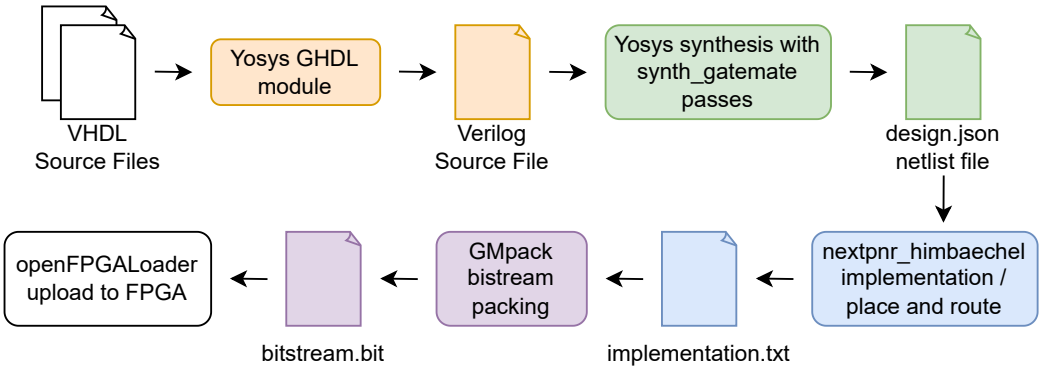


Figure 5.1: Flow chart showing VHDL workflow using the OSS-CAD-Suite

that targets the M1A1 FPGA chip and generate a netlist, Yosys has to be inferred with the `synth_gatemate` command, a script that contains a list of Yosys passes including logic optimizations, dead-code elimination and technology mapping targeting GateMate™ FPGAs. Nextpnr [38], an open-source place and route tool, requires Yosys to emit a JSON netlist file. Nextpnr-himbaechel is a modified version of nextpnr incorporating the work of project peppercorn [39], that allows the synthesized design to be mapped towards the GateMate™ M1A1 FPGA chip. As the implementation tool only outputs the implemented design as a human-readable file, it needs to be packed as a bitstream using GMpack before programming the Cologne Chip Evaluation Board seen in Figure 2.1 using openFPGALoader [40].

A Makefile, that follows the GateMate™ toolchain quickstart guide [17], is used to automate this workflow and included in project's root directory [8].

5.1.2 Design Methodology

To ensure extendability, each component is designed as an independent module with clearly defined responsibilities and standardized interfaces. The FPGA W5500 Controller (Section 5.2) operates independently from the data concentrator's implementation described in Section 5.4, communicating solely through AXIS interfaces. Finite State Machines (FSMs) are employed for sequential control logic, where data is processed in multiple steps or a control flow is required. Examples include the Threshold Logic (Section 5.4.1), where Metric Packets need to be sequentially parsed before any sensor value is compared to its known thresholds and the packet is buffered for further processing in the data concentrator.

All designs follow an active-high synchronous reset strategy to ensure deterministic initialization and simplify timing analysis.

For design simplicity, the entire system implemented on the FPGA chip operates on a single clock domain, eliminating complexity of clock domain crossing logic and associated synchronization circuitry. Data pipelining, employed in both the W5500 Controller and data concentrator, relaxes timing constraints by breaking combinatorial paths into multiple clock cycles. This allows for a higher clock frequency and improves timing closure during implementation.

VHDL development followed a testbench-driven approach, where modules were verified using GHDL simulations prior to system integration. GTKwave was used for waveform analysis to validate AXIS handshakes and state machine behavior in behavioral VHDL simulations.

5.2 FPGA W5500 Controller

An existing W5500 Controller and SPI Master, originally developed for the PYNQ-Z1 FPGA board, were migrated from the Vivado Design Suite to the Cologne Chip GateMate™ M1A1 FPGA platform using OSS-CAD-Suite. This required special attention to sensitivity lists and latching logic, as it was observed, that Vivado's simulation tools operated under more relaxed VHDL rules, potentially utilizing a more heuristics based approach while GHDL defaults to stricter interpretations of VHDL-based designs if not configured otherwise using command-line arguments. The original hardware design utilized an AXIS data FIFO, an Intellectual Property (IP) core with inaccessible source code and owned by AMD/Xilinx [25]. This made it necessary to design a custom VHDL AXIS FIFO for the GateMate™ M1A1 FPGA chip, that utilizes the FPGA's BRAM blocks. While the GateMate™ primitives guide [19] lists a CC_FIFO_40K IP core that would be ready to use with an additional AXI-Stream wrapper, the W5500 Controller has no requirement for FIFOs of 40 bit data width. As most data inside the W5500 Controller is processed byte-wise, a $2K \cdot 10$ bit TDP configuration, as shown in Table 2.2, is sufficient to store an 8 bit payload unit and the AXIS `LAST` bit inside a line of memory and will be synthesized towards a more resource efficient CC_BRAM_20K primitive seen in Figure 2.3. This packing of 8 bit `DATA` with a `LAST` bit in a 10 bit line of memory is shown in Figure 5.2.

The original W5500 Controller was modified such that its MAC address, the IP addresses and UDP ports are now configurable as a generic in the VHDL top file. Additionally the W5500 Controller's functionality has been extended to make use of all 8 sockets available on the W5500 Ethernet module. A VHDL generic determines the amount of sockets opened as UDP sockets and checked for received data. Each of the eight sockets is configured with 2 KB of memory for both the TX buffer and 2 KB for the RX buffer by

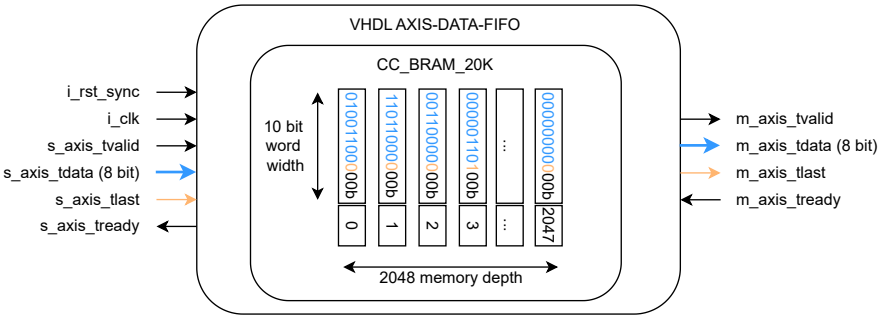


Figure 5.2: AXI-Stream data FIFO for the W5500 Controller and SPI Master utilizing the Cologne Chip's 20K BRAM primitive; blue indicates the eight data bits, orange indicates the last-bit

default, resulting in full utilization of 32 KB of available W5500 buffer memory.

During the SPI header phase, the Block Select Bits (BSB) behave similar to upper memory address bits and select one of the 8 possible sockets [41]. As only the BSB need to be modified for a socket switch, the W5500 FSM only had to be extended by scheduling related states. The two round-robin-based scheduling modes `send_first` (Figure 5.3) and `receive_first` (Figure 5.4) allow the W5500 Controller's control graph to favor either sending or receiving of UDP packets.

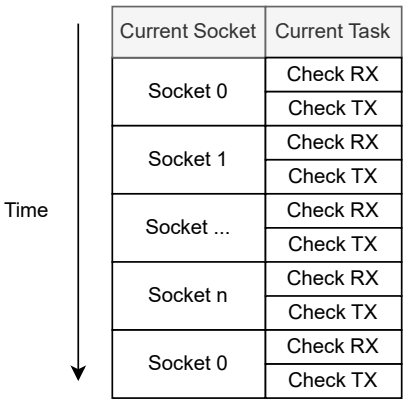


Figure 5.3: `send_first` W5500 scheduling

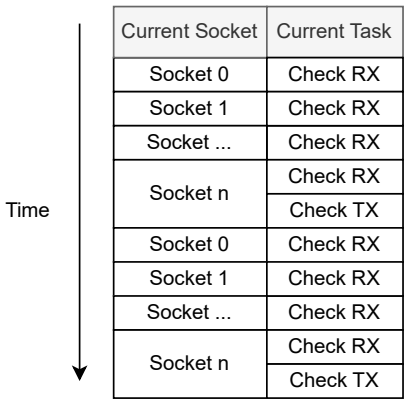


Figure 5.4: `receive_first` W5500 scheduling

Determining if there is data to send for the W5500 is as simple as checking if the external `TVALID` signal is high, while checking if data has been received takes up to 90 clock cycles per socket. In the `send_first` configuration, the frequency at which available data to send is checked is eight times higher than during the `receive_first` configuration, where data from all eight socket RX buffers is read first, before any data can be transmitted by the W5500.

To ensure proper support of 8 simultaneously open UDP sockets and lay foundations for a future priority system, a 3 bit `USER` field encoding the socket index has been added to the AXI-Streams carrying payload data towards and from W5500 socket buffers. The `USER` field can be seen in Figure 5.7. The `ext_payload_tuser` signal selects the port, that the UDP packet should be sent from and `ext_payload_ruser` tells the data concentrator which socket the UDP packet has been received from.

One advantage of using UDP packets with the W5500 Ethernet chip is that each packet includes an 8 bit UDP header as seen in Figure 5.5 rather than just the received payload. This allows for basic integrity checks and early discarding of Metric Packets, as the UDP packet adapter reads the UDP payload length before the data concentrator has to parse the Metric Packet (refer to Section 5.3.2).

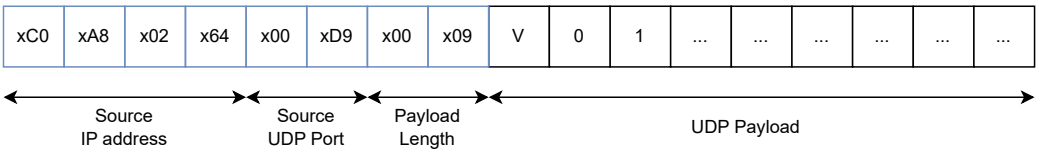


Figure 5.5: W5500 UDP packet structure, beginning with 8 byte UDP header (blue) and followed by 9 bytes of UDP payload

5.3 Metric Packets

A Metric Packet starting with protocol code `V01` (HEX: 56 30 31) is followed by an identifier, ends with a numeric value and has a length of 9 bytes as shown in Figure 5.6. Metric Packets always need to start with an ASCII "V" and are followed by two digits represented in ASCII to be parsed by the Threshold Logic. Currently only protocol code `V01` is supported, which expects two identifier bytes and a four-byte big-endian signed Q22.10 fixed point number (refer to Section 2.7), coming at a fractional precision of 0.0009765625 and an integer range from -2097152 to 2097151. This scaling has been chosen, because its fractional precision allows for measurements to capture fine-grained sensor variations in the millivolt range while maintaining a sufficiently large dynamic range for typical operation conditions.

As Metric Packets are streamed and received byte-wise, the protocol code serves as another simple integrity test and lets the data concentrator skip over unsupported packets.

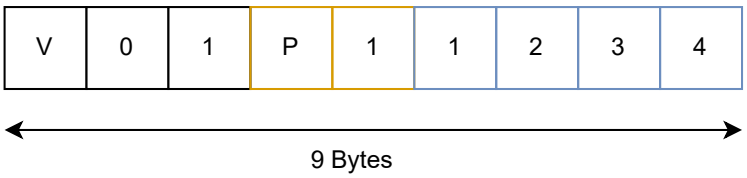


Figure 5.6: Metric packet with ASCII protocol code `V01` (black), two byte identifier (yellow) and four byte metric value (blue)

While 16 identifier bits can encode up to unique $2^{16} = 65536$ metric packet sources, only 1024 pairs of threshold values are supported per input channel of the data concentrator (5.10). This limitation is introduced by the Threshold Logic (5.4.1) and the need for efficient use of BRAM resources inside the M1A1 FPGA chip. In other words, $2^6 = 64$ devices specified by the 6 least significant identifier bits can share the same threshold pair in BRAM.

5.3.1 Metric Packet AXI-Stream and FIFO

The data concentrator accepts AXIS interfaces with an 8 bit `DATA` width, a 3 bit `USER` field width and a `LAST` signal at its input channels as shown in Figure 5.7. This is structurally identical to the W5500 Controller's data interface. Due to the FPGA W5500 Controller being designed with modularity in mind, its `USER` field encoding the socket number will be interpreted by the data concentrator as the Metric Packet's priority. The `LAST` bit is asserted at the last byte of a Metric Packet.

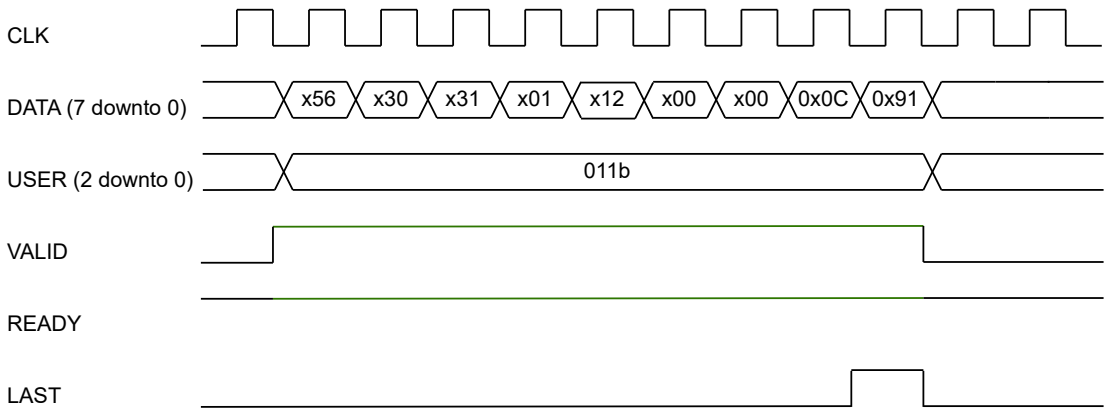


Figure 5.7: AXIS transaction of a Metric Packet accepted by the data concentrator; The RX-side W5500 received the UDP packet on socket 3

To buffer Metric Packets inside of the data concentrator and UDP packet adapter Metric Packet FIFOs are introduced, a modification of the AXIS-data-FIFO used in the W5500 Controller that additionally stores the `USER`-bits in each memory line. The Metric Packet

FIFO is designed to only uses a single CC_20K_BRAM primitive by utilizing data packing as seen in Figure 2.3.

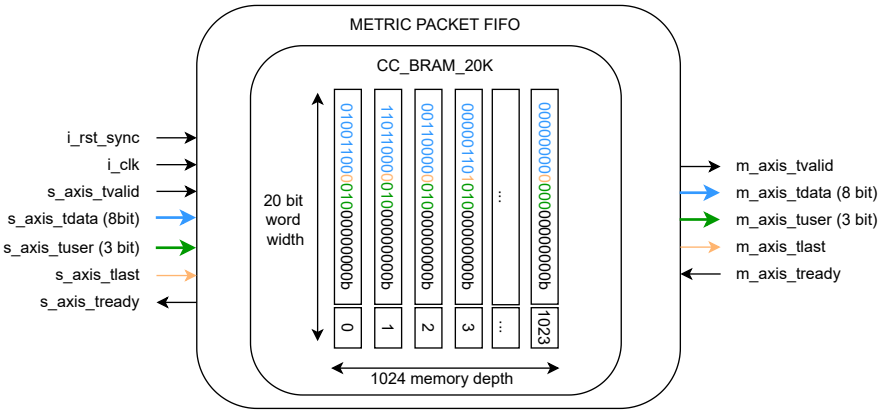


Figure 5.8: Metric Packet AXI-Stream data FIFO utilizing a single CC_BRAM_20K primitive; blue indicates the eight data bits, orange indicates the last-bit and green the three user bits; Remaining bits stay unused but could serve as parity bits in the future

5.3.2 UDP Packet Adapter

The purpose of an adapter is to allow compatibility between two communication interfaces, visualized in Figure 5.9. The W5500 outputs UDP packets at an AXIS interface starting with a UDP header, shown in Figure 5.5, that needs to be parsed and removed from the data stream. Because a data concentrator compatible Metric Packet is expected to start with an ASCII “V” (HEX: x56), this is checked before the UDP payload is written into a Metric Packet Buffer FIFO indicated in Figure 5.8. The RX FIFO shown in Figure 5.9 must be emptied before the W5500 Controller’s FSM can return to its default routine. For this reason, the buffer FIFO inside the UDP packet adapter also is also responsible for immediately draining the RX FIFO, preventing the W5500’s control flow from stalling.

5.4 Data Concentrator Implementation

This section on the data concentrator will go over the implementation of its four components, the Threshold Logic 5.4.1, the Metric Packet Manager 5.4.2, the Telemetry Sender 5.4.3 and a Glitch Filter design for the external interlock signal 5.4.4.

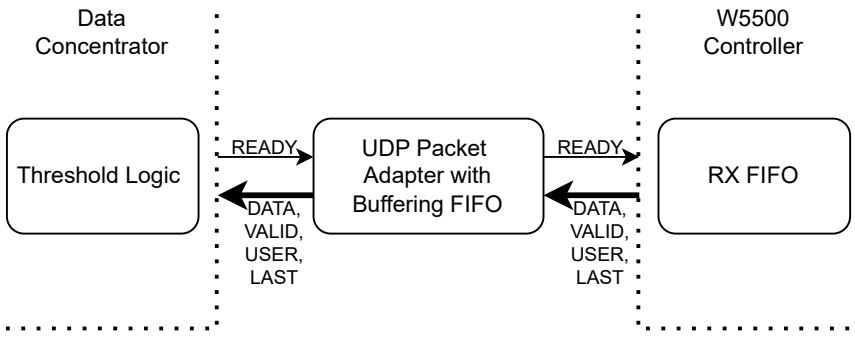


Figure 5.9: UDP packet adapter in between a W5500 Controller and data concentrator ensuring compatibility between both subsystems

5.4.1 Threshold Logic

The Threshold Logic expects Metric Packets as a byte-wise AXI-Stream and checks compatibility with the Metric Packet's protocol code first, as illustrated in the FSM A.2. Packets with unsupported protocol codes aren't dismissed, instead they are passed through the Threshold Logic without affecting interlock assertion. With the identifier being the second part of the Metric Packet starting with a with protocol code `V01`, the most significant 10 bit of the two identifier bytes (Figure 5.10) are used as an address to look up a pair of threshold values in a pre-initialized BRAM. 10 bit addresses can be used to look up from 40 bit words in 1024 lines of memory as seen in the possible configurations for the CC_BRAM_40K primitive (Figure 2.4).

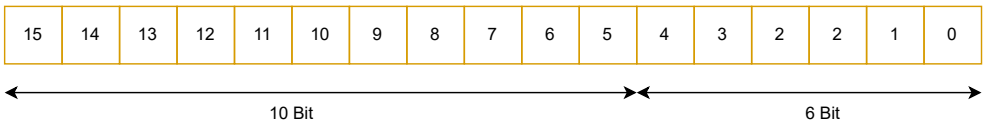


Figure 5.10: Closer look at the 16 identifier bits of a Metric Packet; the most significant 10 bits encoding the address for threshold lookup from BRAM

In the current implementation stored numeric threshold values are only 32 bits in length, which means that the 8 most significant bits in each line of memory stay unused. As threshold values come in pairs, a single bit appended to the 10 bit address to encode the location of the lower and upper threshold values. Because $2048 \cdot 32$ bit $= 65536$ bit exceeds the storage capacity of 20K or 40K BRAM primitives (Section 2.4.1), the synthesis tool Yosys has to infer the memory block using multiple BRAM cells, that operate in parallel. In theory four CC_BRAM_20K or two CC_BRAM_40K cells are needed to satisfy memory requirements of a fully utilized lookup BRAM. However, because the memory content is known at compile-time and thresholds are constant, synthesis optimization passes can result in fewer BRAM instances than expected, when the theoretical storage

capacity of threshold values is not fully utilized (Section 5.6.1). The theoretical memory structure implemented inside each Threshold Logic unit to look up pairs of 32 bit numeric threshold values using two CC_BRAM_40K cells is shown in Table 5.1.

Table 5.1: Table showing memory structure storing 1024 pairs of thresholds using two CC_BRAM_40K cells configured for 10 bit address width and 40 bit words each

address(10 downto 1)	**address(0) = 0b Lower Threshold**	**address(0) = 1b Upper Threshold**
0000000000b = 0	0x0000000000	0x000000003A
0000000001b = 1	0x0000000F3C	0x000000F000
0000000010b = 2	0x0000000012	0x0000000ABC
...	...	...
1111111111b = 1023	0x0000000000	0x000000F000

Metric values, that exceed the specified safe area of operation pull up the Threshold Logic's interlock signal, highlighted in Figure 4.2 as red arrows. This information is forwarded to the Telemetry Sender and queues an alert packet for a W5500 module to transmit as a UDP packet with highest priority on socket 0 (refer to Section 5.4.3). The Thresholding Logic's interlock signal is kept high until the `deassert_interlock` signal is pulled high by mechanically pressing the button on the Cologne Chip M1A1 Evaluation Board seen in Figure 2.6. Unless the safe operation area for each identifier is explicitly specified, the BRAM used in the Threshold Logic comes pre-initialized with zeros for both lower and upper thresholds. Non-zero parameter values will result in a protective interlock assertion by default. As MRI systems are medical devices with safety-relevant sensor readings, this defensive programming ensures, that all monitored devices operate within known and safe boundaries.

Checked Metric Packets are then fully written into a FIFO acting as a buffer, such that they are ready for further processing in the Metric Packet Manager (refer to Section 5.4.2) as the Threshold Logic becomes available to accept the next Metric Packet.

A diagram of the Threshold Logic entity described using VHDL is shown in Figure 5.11.

5.4.2 Metric Packet Manager

The diagram in Figure 5.12 shows a Metric Packet Manager with a variable amount of inputs at the bottom, where each input is a Metric Packet AXI-Stream (refer to Section 5.7). Metric Packets stored inside the Threshold Logic's buffering FIFOs are provided at the Metric Packet Manager's input channels, while a RR arbiter selects one input channel

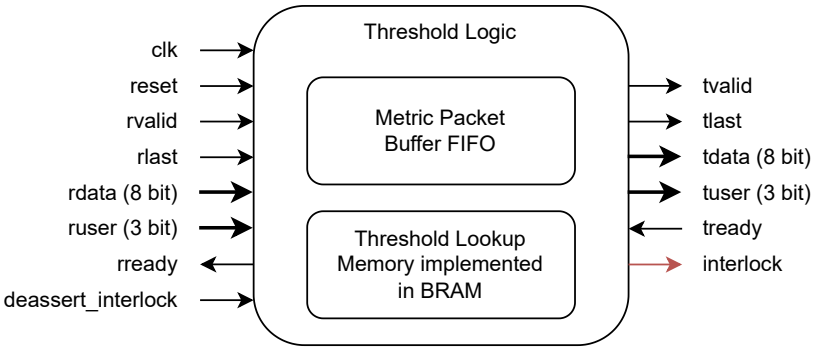


Figure 5.11: Diagram of Threshold Logic showing the entity ports, the included Metric Packet Buffer FIFO and the Threshold Lookup Memory

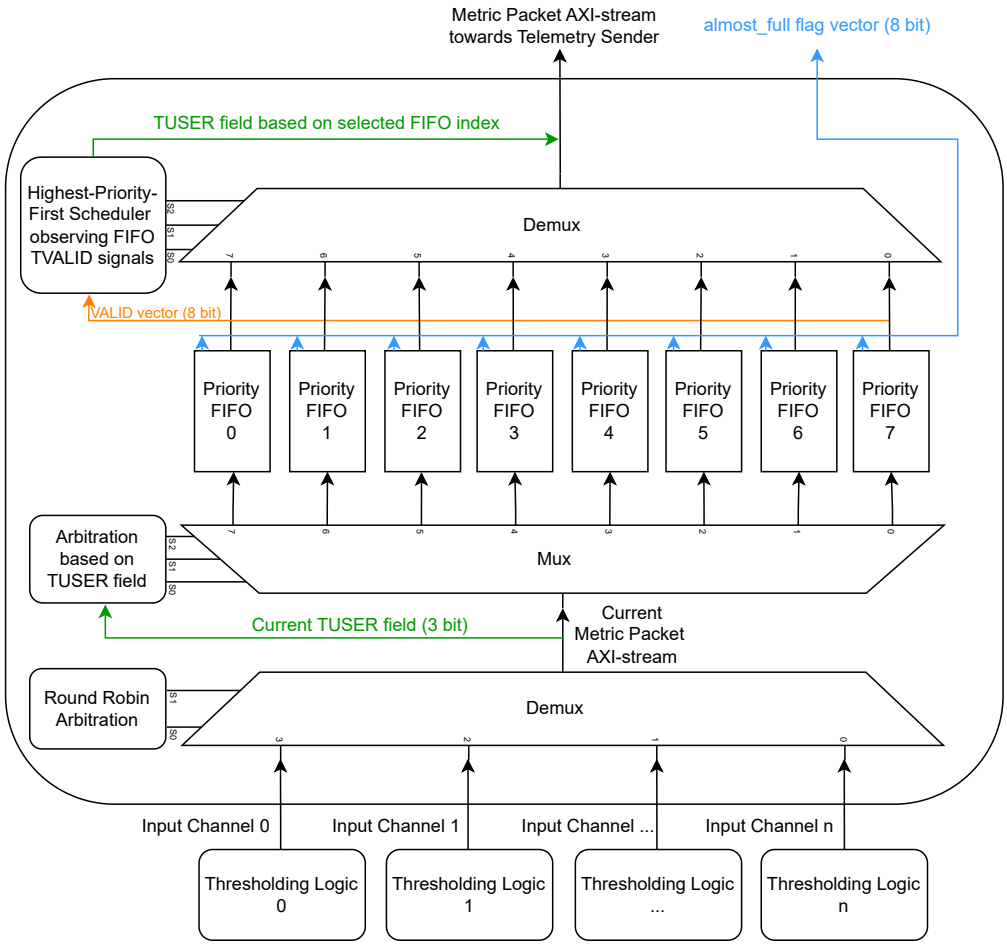


Figure 5.12: Diagram of Metric Packet Manager showing the arbitration of Metric Packets towards their corresponding priority FIFO according to the current input's `TUSER` field (green) and the Highest-Priority-First Scheduler with respect to the FIFOs outgoing valid signals (orange); `almost_full` flags (blue) being forwarded towards the Telemetry Sender

after another, indicated by a demultiplexer (Demux) shown in Figure 4.2. When valid data is observed on the currently selected incoming AXIS interface, the Metric Packet's priority is read from the 3 bit AXIS `TUSER` field, serving as the select signal for a multiplexer (Mux) based design directing the packet to its corresponding data path. This allows Metric Packets to be streamed into and temporarily stored within a designated Priority FIFO. These FIFOs are implemented as an AXIS-Data-FIFOs (refer to Figure 5.2) extended by an `almost_full` flag. This flag indicates congestion when more than 2000 bytes are stored, approaching their maximum storage capacity of 2048 bytes. Eight Priority FIFOs were chosen due to the W5500 Ethernet chip supporting up to 8 simultaneously open UDP sockets.

To efficiently implement the Priority FIFO with respect to physical BRAM utilization, each FIFO receiving a Metric Packet AXI-Stream as input intentionally does not store the `USER`-field information inside its memory, such that the design leverages the more resource efficient $2K \cdot 10$ bit configuration shown in Table 2.2 and Figure 5.2. This approach minimized unused padding bits within the BRAM cells while providing sufficient storage to store the 8 bit `TDATA` information alongside the `TLAST` bit. Instead of storing the `TUSER` field in memory, the Metric Packet's priority is reconstructed from the Priority FIFO's index, restoring the original Metric Packet AXI-Stream when packets are being forwarded towards the Telemetry Sender.

An eight bit long vector is constructed by concatenating all master-sided `TVALID` signals originating from the Priority FIFOs giving information on whether the FIFO contains data to be transmitted. All valid signals are evaluated within the same clock cycle, allowing the scheduler to select a data path using combinatorial logic based on current FIFO occupancy information and according to a HPF arbitration scheme [42]. This configuration allows for direct data passthrough of Priority FIFO content towards the Telemetry Sender, ensuring that critical Metric Packets are prioritized for transmission with minimal latency. However, this simple strategy is susceptible to a problem called starvation as explained in the chapter Theoretical Background 2.6, where low priority tasks, or in this case low priority Metric Packets, can theoretically wait for an indefinite amount of time until being further processed.

Considering the possibility of low priority FIFOs being filled up faster than being emptied and to avoid stalling, the `almost_full` bit vector shown in Figure 5.12 is forwarded to the Telemetry Sender, such that a non critical low priority alert UDP packet acting as a notification can be send to the Metric Packet Server for debugging purposes. In practice, FIFO congestion only becomes a problem when the amount of incoming Metric Packets exceeds the output bandwidth that the TX-side W5500 can handle or when the

data concentrator receives an exceeding amount of high-priority Metric Packets, blocking the emptying of low priority FIFOs. Both the potential issue of starvation and FIFO congestion will be discussed in Section 6.5.

5.4.3 Telemetry Sender

The Telemetry Sender shown in Figure 5.13 is the entity in the data concentrator, that interfaces directly with a W5500 Controller, such that telemetry data can be sent as UDP packets over Ethernet. It receives both a Metric Packet AXI-Stream (refer to Section 5.3.1) coming from the Metric Packet Manager and the global interlock signal generated by an OR-operation of all available interlock signals as shown in Figure 4.2. By default, the Metric Packets stored in the Priority FIFOs are forwarded towards a W5500 Controller, whose input AXIS interfaces for UDP payloads matches the output interface of the Telemetry Sender. In case a rising edge is detected on the global interlock signal, the Telemetry Sender's FSM (refer to Figure A.3) temporarily halts transmission of Metric Packets and prioritizes sending a 9 byte "INTERLOCK" ASCII sequence as a high priority UDP packet on socket 0 towards the Metric Packet Server. This enables the Metric Packet Server to time-stamp the assertion of a interlock with millisecond-level precision [43]. Additionally, the `almost_full` flag vector presented by the Metric Packet Manager is observed to notify the Metric Packet Server of too much load on Priority FIFOs within the data concentrator, hinting a slowdown or priority adjustment for devices providing Metric Packets. In this case, a 10 Byte UDP packet "ALMOSTFULL" is queued for the TX-side W5500 to sent on the lowest priority UDP port.

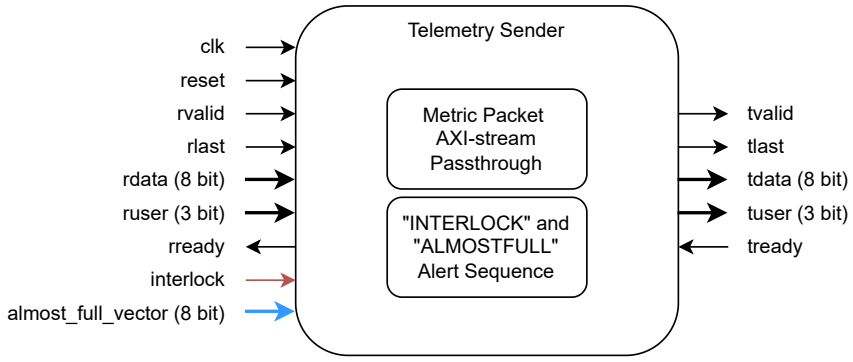


Figure 5.13: Diagram of the Telemetry Sender showing entity ports, AXIS passthrough of data from the Priority FIFOs towards the W5500 and storage of the alert sequence "INTERLOCK" and "ALMOSTFULL"

5.4.4 Interlock Glitch Filter

Glitches refer to temporary and unwanted distortions in a digital signal [44]. Though external interlock inputs such as the one provided to the data concentrator in the proposed

system design shown in Figure 4.2 are inherently asynchronous to the FPGA system clock, they can be synchronized by inserting an additional buffer at the input pin, specified using constraints during the implementation stage (Section 5.6.2). Slow signal transitions on the external interlock input or a mechanical contact bounce can lead to unwanted transitions of the signal sampled by the FPGA.

The Interlock Glitch Filter is implemented using a 4-stage shift register sampling the external interlock signal on every system clock cycle. Logic observing the shift register outputs a logical "1", when all four values are equal to "1". Analogous to this, the glitch filter's output is pulled low, when the shift register's content is equal to `0000b` as demonstrated in Figure 5.14. This behavior introduces a digital hysteresis characteristic, as the filtered output does not change state in response to intermediate signal values. Though

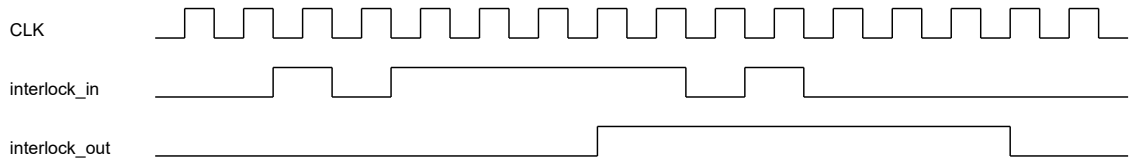


Figure 5.14: Interlock Glitch Filter hysteresis behavior shown on positive and negative glitch on an incoming external interlock signal

this approach adds a minimum latency of four clock cycles to the propagation of a state switch on the interlock signal, it adds resilience to short unwanted signal pulses in the form of positive or negative glitches and preventing the Telemetry Sender implemented in Section 5.4.3 to send multiple "INTERLOCK" alert UDP packets.

A diagram of the Interlock Glitch Filter's VHDL entity and its ports is shown in Figure 5.15, with the `interlock_in` signal being the external interlock signal and `interlock_out` being the filtered signal. The observation logic refers to the conditional logic, that checks for all shift register bits being "1" or "0" before driving the output interlock signal.

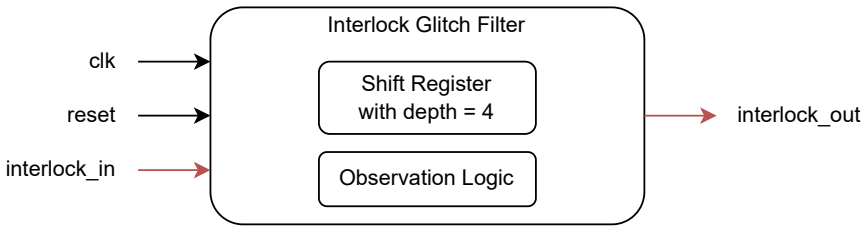


Figure 5.15: Diagram of the Interlock Glitch Filter showing entity ports, the 4 stage shift register and observation logic controlling the output signal

5.5 Phase-locked loop CC_PLL

A PLL refers to the onboard analog circuit generating the FPGA's system clock frequencies. The CC_PLL primitive needs to be instantiated in the VHDL top file. The Cologne Chip GateMate™ M1A1 evaluation board is equipped with a 10 MHz oscillator, that is configured as the PLL's reference clock parameter (`REF_CLK`). The PLL's low jitter mode is enabled and the clock generator is set to its high performance "SPEED" mode (`PERF_MD`). Though the output clock frequency (`CLK_OUT`) can theoretically be chosen freely, it is primarily limited by the timing estimations and routing during the hardware design's implementation stage.

The generated PLL output clocks are routed through CC_BUFG global clock buffers primitives before being distributed across the FPGA fabric [19].

5.6 Synthesis and Implementation process

A complete list of commands used to run synthesis and implementation of the proposed system is available in a Makefile in the root directory of its GitLab repository [8]. The 2026-01-21 release build of OSS-CAD-Suite [15] was used during all steps.

5.6.1 Synthesis

To perform synthesis on the existing VHDL-2008 source code, Yosys is used with the GHDL plugin imported as a module. The VHDL frontend GHDL is configured for relaxed VHDL-2008 synthesis and Synopsis IEEE compatibility. After importing all relevant VHDL source code files and specifying the top entity, synth_gatemate is executed, a command within Yosys, that runs the complete synthesis flow for Cologne Chip GateMate™ FPGAs. This applies a sequence of Yosys passes that transform RTL code into a technology mapped netlist of GateMate™ specific primitives [45], ready for place and route implementation. When synthesizing designs for the GateMate™ FPGA, it's mandatory to invoke synth_gatemate with the "-luttree" and "-nomx8" flags [17].

The `luttree` flag enables technology mapping towards dedicated LUT-tree cells within the Cologne Programmable Elements (2.2), making efficient use of the GateMate™ FPGA's logic structure. The `nomx8` flag disables Yosys passes, that map the existing hardware design towards 8 input multiplexers (CC_MX8), as these primitives are not supported by the nextpnr_himbaechel place and route implementation tool in the available versions of OSS-CAD-Suite. The write_json command outputs a final and synthesized design as a JSON file for further processing (5.6.2). A post-synthesis resource utilization for a single-input channel data concentrator configured for two W5500 Ethernet modules, one

UDP packet adapter, two W5500 Controller and two SPI Master is listed in Table 5.2.

Table 5.2: Post-synthesis resource utilization; Hardware design contains data concentrator, UDP Packet adapter, two W5500 Controller and two SPI Master and fully utilized threshold lookup memory

Synthesis Statistics					
Number of wires	7437	Number of public wires	1550		
Number of wire bits	25257	Number of public wire bits	14506		
Number of memories	0	Number of processes	0		
Number of memory bits	0	Number of cells	9276		
Number of ports	11	Number of port bits	18		
Cell Breakdown					
`$scopeinfo`	51	CC_L2T4	2435	CC_LUT2	967
CC_ADDF	1403	CC_L2T5	745	CC_MX2	930
CC_BRAM_20K	18	CC_LUT1	144	CC_MX4	3
CC_BRAM_40K	2	CC_OBUF	13	CC_PLL	1
CC_DFF	2558	CC_TOBUF	1	CC_IBUF	4
CC_BUFG	2				

With the exception of "$scopeinfo" only containing metadata, the remaining cells refer to technology-mapped primitives that have a hardware equivalent in the GateMate™ FPGA fabric. Notable cell usage includes the 1403 full adders (CC_ADDF), 2558 D-type flip-flops (CC_DFF), 18 CC_BRAM_20K, two CC_BRAM_40K primitives and 930 dual input multiplexers. The BRAM cells show, that the threshold lookup memory (Section 5.4.1) has been synthesized into two 20K BRAM cells, while the eight priority FIFOs (Section 5.4.2), the buffer FIFOs inside the Threshold Logic (Section 5.4.1) and UDP packet adapter (Section 5.3.2) such as the AXIS-data-FIFOs for two W5500 Controller and SPI Master (Figure 5.2) each have been mapped towards 20K BRAM cells.

Underutilizing threshold lookup memory, either by accessing only a fraction of the available address space or storing numerically small positive threshold values, allows Yosys to map the design to fewer physical BRAM primitives. Because threshold values are constant at compile-time, Yosys can perform memory optimization passes able to detect unused address regions or unnecessary Most Significant Bits (MSBs) in stored values, enabling more resource efficient technology mapping towards smaller BRAM configurations (refer to Section 2.4.1) as long as the hardware design stays functionally equivalent [46].

For example, when less than 512 threshold pairs are utilized in total and they are stored in a contiguous block within the address range of 0 to 1023, a single CC_BRAM_40K primitive in SDP or TDP mode is sufficient to functionally substitute the intended dual CC_BRAM_40K design while maintaining original design behavior.

5.6.2 Implementation

Nextpnr_himbaechel, an open-source place and route tool, is configured to target the "CCGM1A1" device and consume the previously Yosys-generated `design.json` netlist as input.

A constraints file is required to map top level signal towards physical FPGA IO ports. The proposed design exposes SPI signals, an interlock output signal, an external interlock input and a deassert_interlock input signal. The interlock output signal is mapped towards an LED, deassert_interlock is controlled by the mechanical push button on the Cologne Chip GateMate™ M1A1 Evaluation Board (Figure 2.1) and all remaining signals are mapped towards pins on the two PMOD headers.

When driving high-speed SPI signals, parasitic wire capacitance significantly affects signal integrity by degrading rise and fall times at higher serial clock frequencies. To improve edge quality the PMOD header pins driving `MOSI`, `SCLK` and `CS` are implemented using output buffer flip-flops (`FF_OBF=true`) and are configured for both fast slew rates and a driving strength of 9 mA.

Enabling `FF_OBF` places a flip-flop in the I/O block directly before output buffer, ensuring that the output signal is registered at the I/O pin and reducing timing variation and jitter. The `DRIVE=9` constraint increases the output drive strength to lower the effective impedance and allow charging and discharging capacitive loads more quickly. Together, these settings result in faster rise and fall times on the output pins.

The input pin receiving the external interlock signal has the FF_IBF option enabled, such that the Glitch Filter implemented in Section 5.4.4 receives a preregistered signal to prevent metastability issues from asynchronous signal state switching.

During the implementation process, nextpnr incrementally improves placement and routing of logic resources on the FPGA while targeting the output clock frequency specified in the PLL parameters (Section 5.5). Setting `time_mode` to "worst" through command-line arguments to configure nextpnr_himbaechel to analyze timing using worst-case conditions (highest temperature, lowest voltage, slowest process corner) [17], ensures the design meets timing requirements (no negative slack) under all conditions. Nextpnr_himbaechel allows the `fpga_mode` configuration to be set to "speed", favoring higher frequencies over reduced FPGA power consumption. This can also be configured using a PLL unit parameter (`PERF_MD`) in the VHDL top file. Using command line arguments, the routing algorithm "router2" has been selected, as mandated by the GateMate™ quickstart guide [17]. This is

extended by using the `-router2-tmg-ripup` flag that enables timing-driven rip-up yields in the `router2` algorithm. This approach tries to re-balance routes based on timing slack and critical paths to improve timings.

Nextpnr outputs a human readable text file, that needs to be packed into a bitstream (Section 5.6.3) before being programmed to the FPGA. An analysis of the post implementation resource utilization and a comparison to an Artix-7 implementation will be covered in Section 6.2.

5.6.3 Bitstream packing and uploading

GMpack 1.9-26 is a bitstream generation tool for GateMate™ FPGAs, developed by Cologne Chip and included in the OSS-CAD Suite [15]. It accepts text files emitted by place and route tool nextpnr and outputs a binary bitstream file for FPGA configuration. Newer versions of GMpack support options to enable the reconfiguration interface `-reconfig`, background reconfigurations in flash mode `-background` and Cyclic Redundancy Check (CRC) checking `-crcmode=check` [17]. The `spimode` option is set to "quad", as this allows for faster reconfiguration times when the bitstream is stored on flash memory on the Cologne Chip GateMate™ Evaluation Board.

OpenFPGAloader, an open-source utility for programming FPGAs, provides vendor-agnostic programming of FPGA devices from manufactures such as Xilinx/AMD, Altera/Intel, Lattice, GOWIN, Efinix and Cologne Chip [40]. The tool supports volatile SRAM configuration for fast iterative development and non-volatile flash memory programming for persistent deployment. To program the FPGA with the generated bitstream file, the project's Makefile provides two upload options using openFPGALoader: Joint Test Action Group (JTAG)-based programming (`-b=gatemate_evb_jtag`) and SPI-based bitstream upload (`-b=gatemate_evb_spi`). Both are using the USB interface the FT2232H chip exposes (as explained in Section 2.4.1) for Bitstream uploads.

5.7 Slow Control Software Infrastructure

Software infrastructure refers to an architecture consisting of a Metric Packet Server, Prometheus and Grafana running on Linux host computers and communicating via HTTP interfaces as shown in Figure 4.1. For simplicity, all components were implemented on a single host computer running Fedora Linux (Kernel 6.18).

Subsection 5.7.1 goes into detail about the Metric Packet Server receiving UDP Metric Packets transmitted by a data concentrator instance such as the usage of the Prometheus client library. The Prometheus setup and pull configurations will be covered in Subsec-

tion 5.7.2. The visualization of metrics in Grafana, that were previously pulled from Prometheus is shown in Subsection 5.7.3. The Metric Packet Server's source code such as the configuration file for Prometheus can be found on GitHub [8].

5.7.1 Metric Packet Server

Written in Python 3.12, the Metric Packet Server is a script that listens on 8 UDP ports, parses received Metric Packets and prepares them to be listed at an HTTP access point to be pulled by Prometheus (5.7.2).

Because Metric Packets with protocol code `V01` arrive with their metric value encoded as a signed Q22.10 fixed point number, their numeric value needs to be converted to a floating point value for further processing. This is done by interpreting the four bytes containing the metric value as a signed `i32` integer, dividing it by the scaling factor $2^{10} = 1024.00$ (Section 2.7) and storing the reconstructed value as an `f32`-floating point value.

The official `prometheus_client` Python instrumentation library [47] was used to expose metrics via an HTTP endpoint at the host computer's network interface at port 8000 under the path `/metrics`. Metrics are displayed using the Prometheus exposition format [48], enabling Prometheus to scrape them and store values in its database.

Prometheus supports the usage of labels, additional metadata that is exposed and stored alongside the metric values. Because the original 16 bit device identifier found in the Metric Packet (refer to Section 5.3) is not natively compatible with the Prometheus naming convention [48], the Metric Packet Server needs to translate it into a hex-encoded identifier string. Both the Metric Packet's hex-encoded device identifier and source UDP port are exposed as labels to ensure uniqueness across different telemetry sources and allow for efficient filtering in Grafana. The string literal `data_concentrator_[x]` serves as the base metric name, allowing for separation of data when multiple data concentrators are in use while keeping each telemetry stream as a unique time-series in the database. This results in the following exposition format:

data_concentrator_[x]{identifier=[HEX String],port=[PORT]}

For example: data_concentrator_0{identifier="051C",port="9219"}

In the rare event, that a UDP interlock alert packets is sent by the Telemetry Sender implemented in Section 5.4.3, the latest interlock alert packet will be timestamped as a 64-bit UNIX time value with millisecond precision at arrival of the UDP packet and exposed as a gauge. The arrival of an "ALMOSTFULL" UDP packet is not timestamped,

instead it is just logged for debugging purposes, hinting a reduction in frequency some Metric Packets are provided to the data concentrator or a reevaluation of packet priorities.

5.7.2 Prometheus

As a pull-based time-series database and monitoring application, Prometheus 3.6.0 [32] scrapes every configured HTTP endpoint every 60 seconds by default. A YAML file found in the projects GitHub repository [8] is used to configure Prometheus to request telemetry data from the Metric Packet Server at a `scrape_interval` of 10 seconds, being the effective temporal resolution of the Slow Control System. While Prometheus does not specify a minimum scrape interval, each scrape iteration is an HTTP that retrieves potentially thousands of parameter values from the Metric Packet Server. For this reason, one second should be considered as a practical minimum for this time interval. Since the SCIS is designed to operate for an extended duration of time, a higher temporal resolution significantly increases the host's CPU load and database storage requirements over time.

Prometheus will act as a data source for Grafana (implemented in Subsection 5.7.3) and requires exposure of its HTTP API at a network-accessible port, by default 9090.

5.7.3 Grafana

Grafana (Version 12.3.1) is an open-source, web-accessible platform for visualizing and analyzing time-series data using interactive dashboards. To ensure ease of deployment, it is implemented within a Docker container and configured with a persistent volume mounted to store configuration data and user data. This allows for state preservation across system restarts and container upgrades. While Grafana is configured to interface with Prometheus via its HTTP API, it notably supports a variety of backends including InfluxDB, OpenTelemetry and AlertManager. Once added as a data source, metrics can be retrieved and visualized using various panel types including bar graphs, line charts, gauges or tables, allowing for interactive dashboard visualization of time-series data as shown in Figure 5.16.

Because the Prometheus database does not store unit information by design, they need to be configured in Grafana after selecting a metric to query. The web interface is configured to be accessible on HTTP port 3000 by default. Although Grafana can query the Prometheus HTTP API at any chosen interval, the achievable temporal resolution is effectively constrained by Prometheus' scrape interval (Section 5.7.2).

Figure 5.16 shows the Grafana web interface accessed through a web browser and showcasing a configured dashboard with various metrics displayed as diagrams.

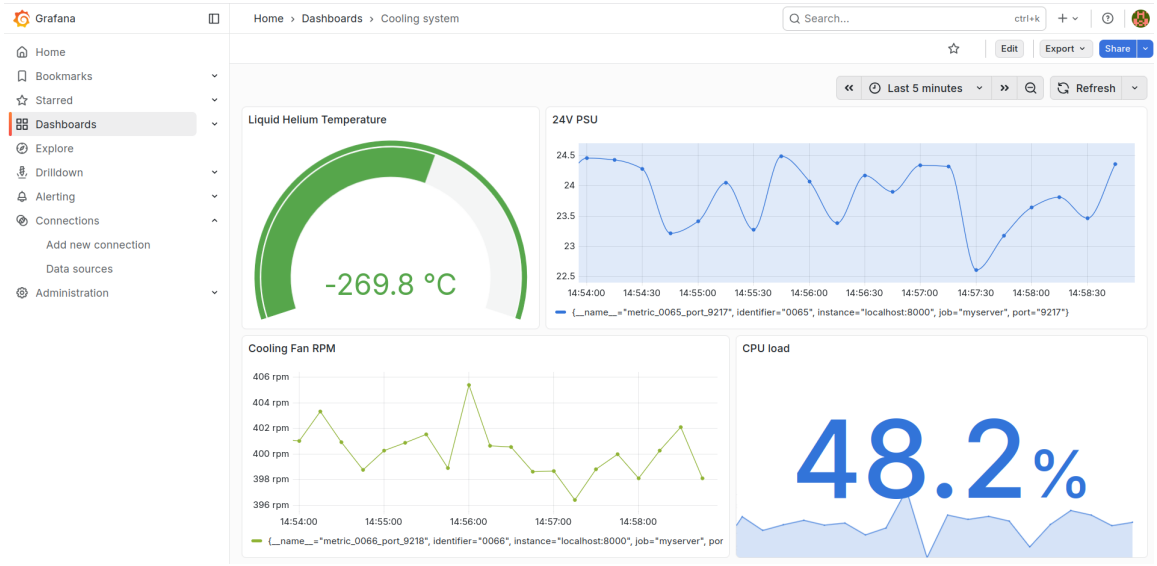


Figure 5.16: Grafana dashboard configured to access synthetic metric data scraped by a Prometheus instance; Dashboard shows different diagrams, configured units and line interpolation

6 System Evaluation

In this chapter the implemented system's performance, reliability and potential bottlenecks is evaluated. A post-implementation analysis in Section 6.2 provides insight into the resource utilization of the data concentrator infrastructure on the Cologne Chip GateMate™ M1A1 platform and compares utilization to a comparable Artix-7 implementation. This is followed by an assessment of system extendability in Section 6.1 and SPI signal quality in Section 6.3 under the chosen hardware and constraint settings. Section 6.4 goes into detail about data concentrator latency measurements of interlock assertion and Metric Packet round-trip times. Finally, FPGA reconfiguration times are measured and discusses in the context of long-term system reliability and device aging. The utilized Python-based testing scripts can be found in the project's GitHub repository [8].

6.1 System Configuration and Extendability

To validate the dual W5500 configuration proposed in Section 4.3, two W5500 Ethernet modules were connected to the PMOD headers of the GateMate™ evaluation board as shown in Figure 6.1. Additionally, an ESP32 microcontroller module was connected to the FPGA board to drive an external interlock signal, testing the data concentrators ability to react to external interlock assertions. Both FPGA W5500 Controller leave flexibility for configuration of MAC-addresses, static IPv4 addresses, the amount of used W5500 sockets and UDP ports using generics within the VHDL top file. This eases reconfiguring the system for different network setups. To connect both W5500 Ethernet modules to a host computer running Fedora 43 (Linux Kernel 6.18), a DGS-105 network switch by D-Link shown in Figure 6.2 was used to route network packets to their destination IP address with negligible latency. A Lenovo USB 3.0 Gigabit Ethernet adapter was used as the host's network interface and configured with a static IP address. Experiments with the ESP32 module asserting interlock signals were successful and resulted in the transmission of the "INTERLOCK" UDP packet towards the Metric Packet Server. The Interlock Glitch Filter (Section 5.4.4) was effective at preventing single pulse state switches triggering the accidental transmission of another alert packet.

Adding data acquisition systems to the data concentrator as a source for Metric Packets requires the setup of an additional Threshold Logic units. These units operate in parallel, which is beneficial for systems where interlock assertion latency is critical. While increasing the number of Threshold Logic units can reduce the average waiting time for some Metric

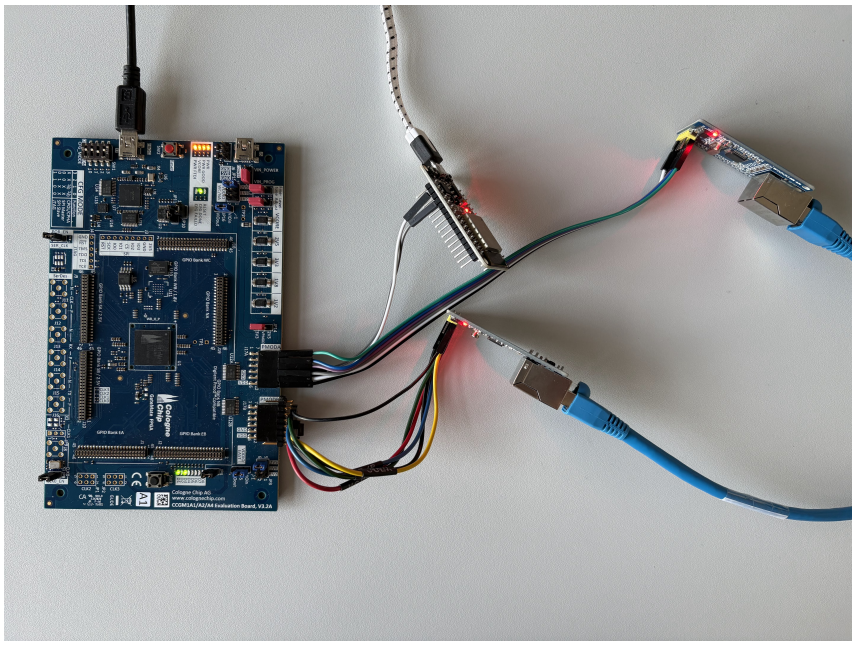


Figure 6.1: Two W5500 Ethernet Modules and an ESP32 module asserting external interlock signals connected to PMOD headers of the Cologne Chip GateMate™ M1A1-E1 Evaluation board

Figure 6.2: D-Link DGS-105 used as a network switch for UDP Metric Packets

Packets, it also increases FPGA resource utilization and ultimately gets bandwidth limited by the simple round-robin scheduling of the Metric Packet Manager reading from one buffer FIFO at a time (refer to Section 5.4.1 and 5.4.2) and bandwidth limitations introduced by the W5500 Ethernet module dedicated for TX duties. The cost of adding W5500 modules to the data concentrator's input channels to extend Ethernet throughput for incoming UDP packets is discussed in the following Section 6.2.

Analog sensor readings can be incorporated into the Ethernet-based data concentrator infrastructure by using a microcontroller such as the RP2040. Its integrated Analog-Digital

Converter (ADC) converts analog signals to digital values and constructs UDP Metric Packets, that are sent to a W5500 module acting as a receiving endpoint for Metric Packets over Ethernet. Although threshold values must be stored as signed Q22.10 fixed point numbers at dedicated 11 bit addresses in a pre-initialized BRAM memory, a Python script found in the project's GitHub repository [8] was created to automatically generate memory initialization lines based on the 2 byte identifier such as the lower and upper threshold value. It accepts floating point values as input, converts them into signed Q22.10 format and generates VHDL compatible initialization lines to be inserted in the Lookup BRAM definition.

6.2 Post Implementation Analysis

The hardware design successfully implemented onto the GateMate™ FPGA fabric is visualized on a placement map (Figure 6.3) generated by nextpnr and containing the data concentrator hardware design, two W5500 Controller, two SPI Master and one UDP packet adapter. Grey spaces show unused FPGA resources and indicating extension capabilities for additional peripheral devices, DAQs or larger storing buffers.

In Table 6.1 the post-implementation FPGA resource utilization is shown for a system design with two W5500 Controller, two SPI Master, one UDP packet adapter and a fully utilized threshold lookup memory.

Table 6.1: Post-implementation FPGA device utilization of a system design configured for two W5500's, containing one UDP Packet Adapter and a full-size threshold lookup memory inside the Threshold Logic

Resource	**Used**	**Total**	**Utilization (%)**
USR_RSTN	0	1	0
CPE_COMP	0	20480	0
CPE_CPLINES	110	20480	0
IOSEL	18	162	11
GPIO	18	162	11
CLKIN	1	1	100
GLBOUT	1	1	100
PLL	1	4	25
CFG_CTRL	0	1	0
SERDES	0	1	0
CPE_LT	9224	40960	22
CPE_FF	2558	40960	6
CPE_RAMIO	1240	40960	3
RAM_HALF	22	64	34

The post-implementation FPGA device utilization table shows, that the current data concentrator is lightweight at a total LUT usage of 22% and a BRAM usage of 34%, with each W5500 Controller and SPI Master contributing about 6% of LUT and 5 Half-BRAM blocks. 22 `RAM_HALF` instances correspond to 11 physical BRAM-tiles on the FPGA fabric, visible as vertical lines in the placement map shown in Figure 6.3. We can observe that only 6% of available flip-flop resources within the matrix of CPE cells and 25% of the PLL's functionality was utilized, due to only using the non-phase shifted `clk0` signal. A utilization comparison to an Artix-7 XC7A35T will be discussed in the following Subsection 6.2.1.

During the routing process, the worst slack estimates are caused by the combinatorial arbitration of AXI-Streams carrying telemetry data and by the look-ahead logic within AXIS data FIFOs inside the scheduling stage of the data concentrator. Adding further pipelining-stages could improve timings at the cost of increased data transmission latency within the data concentrator and slightly higher resource utilization. The hardware design implemented onto the GateMate™ FPGA fabric operates at a system clock frequency of 40 MHz under worst timing assumptions. An estimation of achievable throughput based on FPGA system clock frequency using the W5500 Controller and SPI Master will be given Section 6.5.

Table 6.2 lists each component's usage of Half-BRAM blocks for the implemented design at a fully utilized lookup BRAM inside the Threshold Logic.

Table 6.2: Half BRAM block utilization by system components; Design implemented with fully utilized lookup BRAM inside Threshold Logic

Entity Name	**Half BRAM Usage**	**Entity Name**	**Half BRAM Usage**
SPI Master 1	2	UDP Packet Adapter	1
SPI Master 2	2	Threshold Logic	5
W5500 Controller 1	2	Metric Packet Manager	8
W5500 Controller 2	2		

6.2.1 Comparisons to Artix-7 Implementation using Vivado

The open-source OSS-CAD-Suite toolchain proved sufficiently mature for productive VHDL development on the GateMate™. To evaluate synthesis and implementation efficiency compared to mature commercial toolchains, the hardware design was implemented on a comparable AMD/Xilinx FPGA using Vivado 2025.1. After removing the Cologne Chip PLL primitive and setting the FPGA system clock frequency to 100 MHz, the full

hardware design was implemented onto an AMD/XILINX Artix-7 XC7A35TCPG236-3 FPGA chip, which offers comparable LUT resources to the CCGM1A1 [49]. The 100 MHz clock frequency is significantly higher than the 40 MHz that was achievable using nextpnr on the GateMate™ FPGA, effectively pushing the design to limits induced by W5500 SPI communication. Table 6.3 comparing post-implementation device utilization between the GateMate™ and Artix-7 FPGA shows, that overall logic on the GateMate™ FPGA fabric utilization is noticeably more resource-intensive than the Artix-7 implementation. This is expected due to differing architectures, as the Artix-7 utilizes a 6-input LUT structure, whereas the GateMate™ FPGA utilizes more fine-grained approach with its Cologne Programmable Elements (CPE) containing two 4-input LUTs. Comparing the post-routing placement visuals generated by nextpnr (Figure 6.3) and Vivado 2025.1 (Figure 6.4) shows, that logic on the CCGM1A1 is placed around BRAM columns [7] while logic on the AMD/Xilinx FPGA is placed in blocks distributed throughout two FPGA tiles.

Table 6.3: Post-implementation resource utilization comparison

	GateMate™ M1A1 FPGA			**AMD Artix-7 XC7A35T**		
Resource	**Used**	**Total**	**Util. (%)**	**Used**	**Total**	**Util. (%)**
GPIO	18	162	11,1	18	106	16,9
SERDES	0	1	0	0	2	0
LUT	9224	40960	22,5	1984	20800	9,5
FF	2558	40960	6,2	2065	41600	4,9
BRAM	22	64	34,3	9	50	18

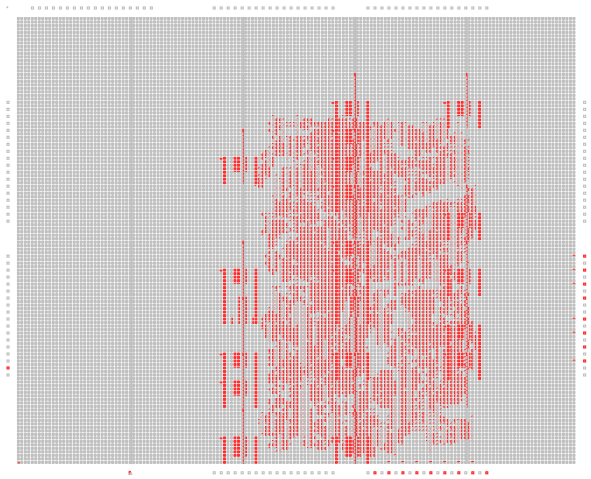


Figure 6.3: Post-routing placement visualization of FPGA data concentrator implementation for GateMate M1A1 exported by nextpnr; Logic placed around BRAM columns

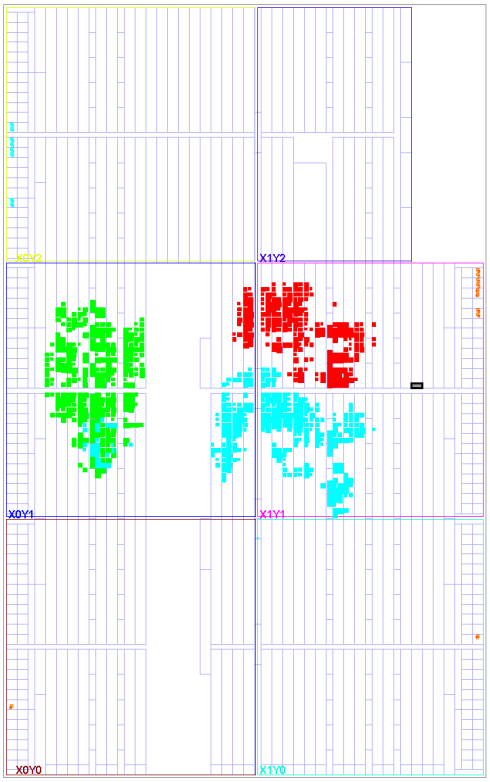


Figure 6.4: AMD Artix-7 XC7A35T placement visual; Green: data concentrator usage; Red: TX W5500 side; Blue: RX W5500 side

6.3 SPI Signal Quality

SPI signal measurements were conducted using an SIGLENT SDS2104X digital oscilloscope (DSO) to evaluate the rising and falling edge quality of digital signals transmitted via the PMOD header interface. To achieve this, an existing SPI Master's `SCLK` signal was cloned onto a second PMOD header pin and both signals were measured simultaneously using oscilloscope probes. During testing, the W5500 such as the associated jumper wires were disconnected to eliminate loading effects and to isolate the influence of additional pin constraints specified during the implementation stage (Section 5.6.2). Measurements were taken at an `SCLK` frequency of 20 MHz.

The oscilloscope capture shown in Figure B.3 presents signal voltage measurements on both a PMOD pin with default constraints (`SLEW=fast` and `DRIVE=3`) on channel 1 (blue) and a header pin configured at a fast slew rate (`SLEW=fast`), an increased drive strength of 9 mA (`DRIVE=9`) and added output flip-flops (FF_OBF=true) on channel 2 (green). It can be observed, that both signals show correct logic-level transitions between their intended '0' and '1' states, the output pin with a fast slew rate and higher drive strength demonstrates

significantly reduced rising and falling edge times. At 20 MHz `SCLK` frequency, the signal on channel 2 closely approximates an ideal square wave, while the signal on channel 1 exhibits characteristics similar to a triangle waveform due to slower edge transitions. Both signals show typical ringing characteristics found in switching circuits, primarily caused by resonance of reactance components and an impedance mismatch between the driver and load [50].

In conclusion, the constraints settings chosen during the implementation stage produce a stable 20 MHz square wave at the `SCLK` output pin with significantly reduced rising and falling edge times. These improved edge characteristics improve signal integrity and allow for a more robust SPI communication compared to default pin configurations.

6.4 Latency and Time measurements

This section aims to evaluate round-trip-time (RTT) measurements for the FPGA data concentrator implementation by sending Metric Packets as UDP packets towards a W5500 and locally measuring time until they return. UDP packets were intentionally sent to only a single UDP socket, as this creates a worst-case scenario for the W5500 Controller's `receive_first` scheduling (refer to Figure 5.4), where the delay in between checks of the same RX buffer is at its maximum. These results will be compared to a CPU-based data concentrator setup implemented as a Python script on a Pine A64 Single Board Computer (SBC) running Armbian 25.8.1 [51]. In the second part of this section, interlock assertion and latencies will be discussed.

The FPGA data concentrator is used with two separate W5500 Ethernet modules shown in Figure 6.1 splitting receiving and transmission of UDP packets to reduce potential system latency and leverage W5500 Controller scheduling (Figure 5.3 and 5.4) prioritizing sending or receiving of UDP packets. All tests were conducted using a USB 3.0 Gigabit Ethernet Adapter connected to a computer running Linux and a DGS-105 D-Link Gigabit Ethernet network switch shown in Figure 6.2. Time measurements are notably influenced by the scheduling behavior of the NAPI network stack inside the Linux kernel [52]. A hybrid interrupt-driven and polling mechanism aims to reduce CPU interrupt overhead under high load by batching network events using a strategy called interrupt coalescing [53], where hardware interrupts are intentionally held back until the next polling cycle. As a result, CPU overhead decreases at the cost of increased packet latency and jitter due to batching.

During testing, coalescing was configured with `$ethtool -C eth0 rx-users 0` to disable the receive coalescing timer, thereby minimizing batching effects introduced by the network

interface controller (NIC). To furthermore reduce scheduling variability, the measurement script and Python based data concentrator hosted on the Pine A64 SBC were executed with real-time priority and pinned to a dedicated logical CPU core using `$taskset -c 4 chrt -f 80 python main.py`.

The histogram in Figure 6.5 shows 65536 time measurements taken between the event of a UDP Metric Packet being send towards the FPGA setup of the data concentrator running at an FPGA system clock frequency of 40 MHz (20 MHz SPI speed) and the moment the UDP packet is received again. This test is repeated with a Pine A64 SBC hosting the CPU implementation of the data concentrator and shown in Figure 6.6. The descriptive statistics tables are given in Table 6.4 and 6.5.

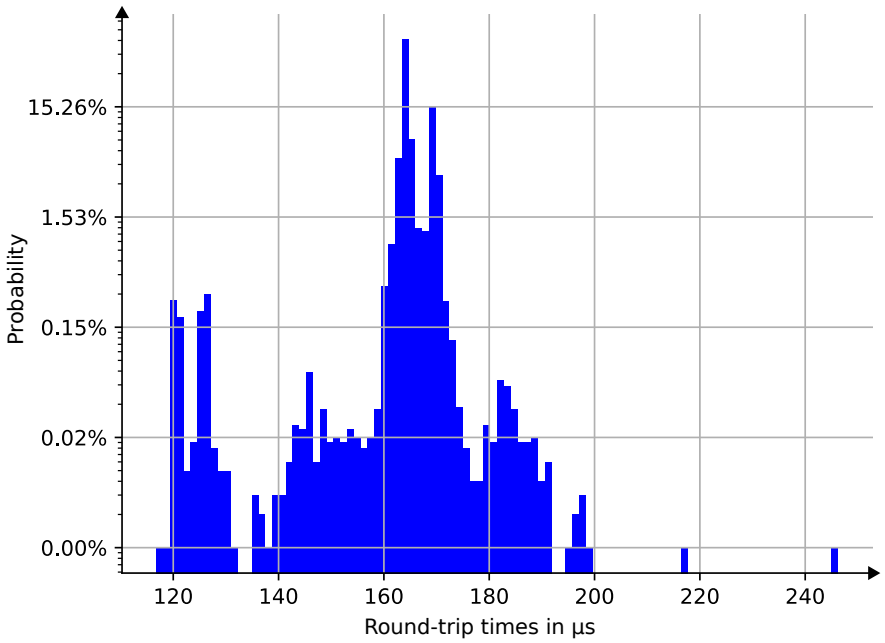


Figure 6.5: Round-trip time (RTT) measurements of FPGA data concentrator implementation shown as a 100 bin histogram of 65536 transmitted and received Metric Packets

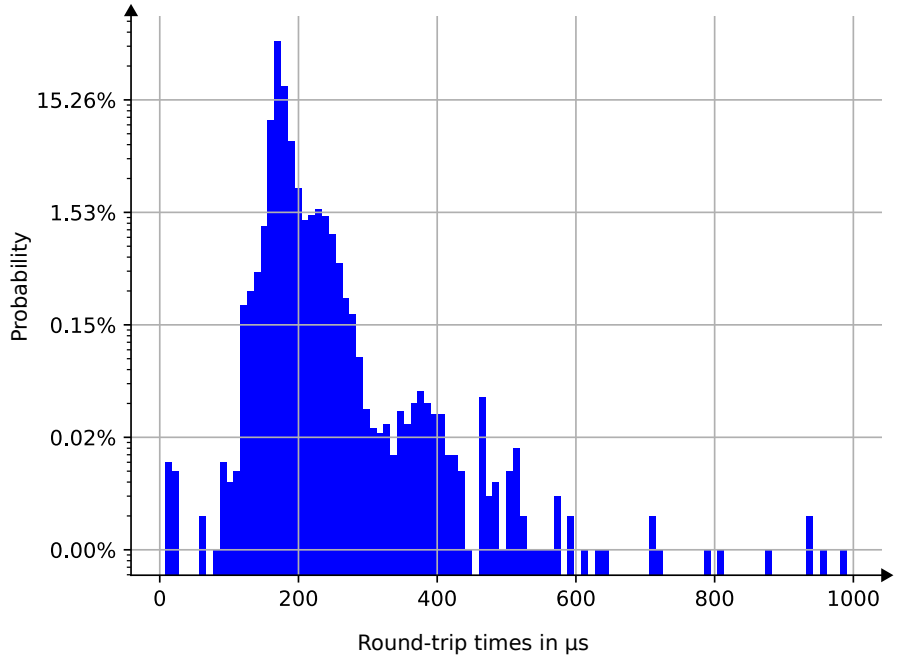


Figure 6.6: Round-trip time (RTT) measurements of CPU data concentrator shown as a 100 bin histogram of 65536 transmitted and received Metric Packets.

The bin containing the most frequent round-trip times for the FPGA data concentrator implementation is seen at around 164 µs and contains 62.53 % of collected values, while for the CPU implementation the probability of the time measurement being taken at 172 µs is at 50.35 %. At a standard deviation of 4.99 µs, the FPGA implementation shows less introduces jitter than the CPU implementation at 25.69 µs. This increased variance is expected, as the CPU implementation has a Python interpreter overhead, interrupt handling and relies on the Linux kernel's scheduler introducing non-deterministic delays from context switches, even with real-time priority. Jitter around smaller batches seen in the histogram shown in Figure 6.5 is caused in part by the internal socket scheduling within the two W5500 Controller. The FPGA implementation operating without an operating system layer achieves a more deterministic timing that reduces overall scheduling uncertainty for the data concentrator. With the maximum measured round-trip time for the FPGA data concentrator being 246.25 µs, the total delay introduced by the FPGA data concentrator and two W5500 Ethernet modules remains well below one millisecond, while the CPU implementation has time measurements exceeding 1 millisecond in rare occasions.

Bottlenecks for the FPGA data concentrator include the simple round-robin scheduling selecting an input stream for the Metric Packet Manager, the scheduling of Metric Packets slowing down processing of low priority packets and internal scheduling of the W5500 Controller.

Table 6.4: FPGA data concentrator implementation (40 MHz sys_clk) round-trip time (RTT) statistics

Metric	Value
Mean RTT	164.76 µs
Median RTT	164.25 µs
Minimum RTT	116.75 µs
Maximum RTT	246.25 µs
Standard deviation	4.99 µs

Table 6.5: CPU data concentrator implemented on Pine A64 SBC round-trip time (RTT) statistics

Metric	Value
Mean RTT	178.23 µs
Median RTT	172.00 µs
Minimum RTT	7.75 µs
Maximum RTT	991.00 µs
Standard deviation	25.69 µs

6.4.1 Interlock Time Measurements

Two critical timing metrics characterize interlock assertion, the latency from Metric Packet reception until interlock assertion by the Threshold Logic unit and the end-to-end delay from UDP alert packet transmission until its reception by the Metric Packet Server.

Threshold Logic Interlock Latency

Interlock latency measurements for the Threshold Logic were determined using a GHDL simulation of a data concentrator testbench. The interlock assertion is deterministic and occurs one clock cycle after the Threshold Logic received the final byte of a 9 byte Metric Packet. In an ideal scenario with direct data transfer, this would require 10 clock cycles in total.

However, the actual latency is influenced by SPI communication characteristics introduced by the W5500 Ethernet platform. The SPI Master communicating with a W5500 module not only operates at an SPI clock frequency of half the FPGA system clock frequency, but needs to deserialize data read on the MISO SPI wire. This effectively slows down transmission speeds towards the data concentrator by a factor of 16, resulting in an expected interlock decision after 130 clock cycles after the first valid byte of the Metric Packet is accepted by the Threshold Logic. At an FPGA system clock frequency of 40 MHz, 130 clock cycles are equivalent to a latency of 3.25 µs.

A state switch issued on the external interlock input signal is registered after a minimum of 4 clock cycles. This latency is introduced by the Interlock Glitch Filter (Section 6.4.1), which implements hysteresis behavior to reduce unwanted state transitions.

End-to-End Alert Latency

Upon detection of a rising edge on the global interlock signal, the FPGA data concentrator suspends the transmission of scheduled metric packets and immediately issues a UDP alert packet. The end-to-end latency is primarily influenced by the W5500 Controller's availability to handle UDP packet, Linux task scheduling and the interrupt coalescence configuration of the Linux host's network interface.

Capturing the UDP packet's time of arrival by the Metric Packet Server is a viable option when millisecond precision is sufficient to determine the moment of interlock assertion. Achieving this precision requires proper configuration of the Linux host's network interrupt scheduling and process priority settings.

6.5 Throughput Limitations and Scheduling Evaluation

The data concentrator's throughput is fundamentally constrained by the W5500 Ethernet module bandwidth limitations and the Metric Packet Manager's scheduling strategy. This section discusses these constraints under both the single and dual W5500 configuration as proposed in Section 4.3 and examines the practical implications of highest-priority-first scheduling.

6.5.1 W5500 Module Throughput Constraints

With a W5500 Ethernet Module being the peripheral device sending telemetry towards the Metric Packet Server as shown in Figure 4.1, three primary bottlenecks introduced by the W5500 chip need to be considered. More specifically, bandwidth limitations introduces by the serialization and deserialization of data by the SPI Master (refer to Figure 4.3), throughput dependencies based on the achievable SPI clock frequency [43] and per-socket buffer size constraints [41]. Each of the 8 UDP sockets within the W5500 chip is allocated with 2 KB for TX and RX buffers each. Filling up an RX buffer faster than the W5500 Controller and UDP Packet Adapter (Section 5.3.2) can drain it should be avoided to not risk a memory overflow. In this case, the W5500 will start to discard incoming UDP packets, effectively resulting in packet loss without notification of the sender.

Using the `receive_first` W5500 Controller default routine (Section 5.2) combats this issue by prioritizing RX buffer polling over TX operations, ensuring that all 8 sockets are checked for available data fore initiating any transmission, reducing the probability of buffer overflow at the cost of increased TX latency.

6.5.2 Single vs Dual W5500 Performance

In the single W5500 configuration, one Ethernet module handles TX and RX duties sequentially. The W5500 Controller's FSM must complete either a receive (reading from RX socket buffer) or transmit operation (writing to TX socket buffer) before switching between these two. This creates a bottleneck, where the W5500 Controller reads received UDP packets while the data concentrator accumulates Metric Packets or alerts waiting for transmission. In another case, when the W5500 Controller focuses on transmission, UDP packets may fill up the W5500's limited RX socket buffers. Despite this limitation, the single W5500 configuration remains FPGA resource efficient and a viable option, when non-latency critical Metric Packets arrive over Ethernet in low volume. When RX socket buffer overflow is unlikely either due to low incoming packet rates or when occasional packet loss is tolerable, the `send_first` scheduler reduces TX latency by prioritizing transmission of telemetry queued by the data concentrator over RX buffer polling. Though FPGA resource effective, the single W5500 configuration is severely throughput limited compared to a dual W5500 configuration (refer to Section 4.3) as the simple SPI based W5500 interfacing managed by a W5500 Controller handles TX and RX duties one after another. This introduces additional latency in transmitting Metric Packets or alerts towards the Metric Packet Server.

A dual W5500 configuration, though requiring more FPGA resources, addresses throughput bottlenecks by splitting TX and RX duties onto two separate W5500 Ethernet Modules. This approach allows one W5500 Controller to continuously drain UDP packets while another simultaneously transmits processed Metric Packets from the Priority FIFOs. The result of two W5500 Controller with scheduling configured as in Figure 4.3 is a near-doubled effective Ethernet bandwidth and the reduction of transmission latencies towards further slow control software infrastructure.

At an FPGA system clock frequency of 40 MHz the achievable W5500 Ethernet transmission speeds are estimated at up to 18 Mbit/s per W5500 Module based on previous evaluations of the FPGA W5500 Controller implementation [43]. At a pessimistic combined finite state machine and SPI communication overhead of 30 %, this results in handling an estimated 175000 9-byte UDP packets per second. The data concentrator's input bandwidth over Ethernet can be extended by connecting another W5500 Controller, SPI Master and UDP packet adapter at the cost of 7% LUT resources and 3 physical BRAM tiles.

6.5.3 Highest-Priority-First Scheduling Analysis

The Metric Packet Manager implements a highest-priority-first scheduler, that selects data from Priority FIFOs in descending order of importance (Section 5.4.2). While this scheduling approach is theoretically susceptible to starvation of low priority tasks [42], experimental evidence suggests that this concern does not practically occur in the the implemented system. First, the temporal scale of operation differs significantly from theoretical worst case scenarios. Metric Packets typically arrive at intervals measured in seconds, while the scheduler processes and transmits packets in microseconds to milliseconds. Three orders of magnitude timing differences provide substantial buffering headroom. Second, the system's critical safety function, interlock assertion, occurs at the Threshold Logic stage (Section 5.4.1) immediately upon receiving a Metric Packet, independent of transmission scheduling within the Metric Packet Manager. With a capacity of over 200 Metric Packets (protocol code `V01`), the buffering FIFO within the Threshold Logic allows continuous processing of Metric Packets even when the Metric Packet Manager's round-robin arbitration is not immediately ready to accept them. That means the interlock signal continuous to be deterministically asserted one clock cycle after the last byte is accepted by the data concentrator, regardless of Priority FIFO congestion or transmission backlog inside the Metric Packet Manager.

Starvation of low priority Metric Packets would only occur under sustained conditions, where incoming packet rates exceed the system's transmission capacity. In such scenarios, Priority FIFO buffers would approach their 2 KB storage limits and the detection of a rising edge on any `almost_full` flag would trigger the transmission of an "ALMOSTFULL" alert packet to the Metric Packet Server (Section 5.4.3). This notification mechanism servers as a diagnostic indicator, that system load is improperly balanced, suggesting either the reduction of sensor polling rates or packet priorities to be re-evaluated to reflect actual operational criticality.

The occurrence of "ALMOSTFULL" alerts indicates a configuration issue rather than a fundamental system limitation. As MRI SCS monitor quasi-static parameters such as temperatures, pressure and voltage levels that change gradually, genuine high priority events that requiring microsecond level interlock assertion are exceedingly rare. The system's design philosophy prioritizes deterministic interlock assertion over guaranteed packet delivery order, aligning with the safety-critical requirements of SCIS (Section 2.1).

6.6 FPGA Reconfiguration Time

Reconfiguration time refers to the duration between de-assertion of the `RESET` signal on the CCGM1A1 FPGA chip and the rising edge on the `CFG_DONE` signal, indicating

successful loading of bitstream and FPGA fabric configuration. This metric is relevant for long-term system reliability, recovery from configuration errors or radiation induced upsets in radiation harsh environments.

The GateMate™ M1A1 FPGA supports autonomous reconfiguration from SPI flash memory, either after pressing the evaluation board's reset button or programmatically using the `CC_CFG_CTRL` primitive [19]. After release of the reset signal, the FPGA enters configuration mode and reads the bitstream stored in the onboard Macronix MX25R6435F 64 Mb flash memory using SPI when "SPI-Active" is configured using the `CFG_MODE` switches on the GateMate™ M1A1-E1 Evaluation Board shown in Section 2.4.1.

Both the default single wire SPI mode such as the quad SPI mode (four simultaneous data lines) were measured 30 times and averaged. GMpack's `-spimode` parameter controls which mode is encoded in the generated bitstream (refer to Section 5.6.2). With the single wire `-spimode=single` utilizing only a single MOSI line, reconfiguration time was measured at an average of 145.8 ms (Figure B.1). Using the Quad SPI mode reduced this value down to an average of 36.17 ms (Figure B.2), a speedup by a approximate factor of 4. Observations indicate that the FPGA reconfiguration time scales with the generated bitstream size and therefore with the complexity of the implemented design. CRC error checking enabled via the GMpack bitstream packing tool had no measurable effect on reconfiguration times.

The sub-40 ms reconfiguration capability in quad-SPI mode compares favorably to alternative CPU based embedded systems running for an extended amount of time. For comparison, even on highly optimized Linux system, a typical cold boot on industrial hardware results in more than a second of downtime [54] while FPGA-based systems allow for rapid recovery and minimize interruptions during long-term use.

7 Conclusion

This thesis presents a scalable and lightweight FPGA-centric SCIS solution, that addresses deterministic latency and software aging limitations of traditional software-based SCS architectures while maintaining its flexibility. Implemented on the GateMate™ CCGM1A1 FPGA, the proposed data concentrator design utilizing two W5500 Ethernet modules achieves sub-millisecond round-trip times and a deterministic nano-second scale interlock response using threshold-based logic.

Implementation experiments revealed that OSS-CAD-Suite's maturity for productive VHDL development on the GateMate platform, despite higher resource utilization and lower achievable FPGA system clock frequencies compared to commercial toolchains like Vivado 2025.1. The implemented hardware design consuming 34% of available BRAM and 22% of LUT resources leaves sufficient headroom for implementation of additional peripheral controllers. Designing custom FIFOs with depths and word widths aligned to native GateMate™ BRAM configurations proved efficient in the implementation of memory-intensive components.

By utilizing quad-SPI mode, the FIFO reconfiguration time was reduced to 36.18 ms, minimizing system downtime in long-term monitoring setups. This rapid recovery favors the FPGA-based data concentrator over traditional CPU-based solutions, which suffer from significantly longer rejuvenation periods. These results demonstrate, that the GateMate platform provides a viable and low-cost solution for safety-critical monitoring and protection.

8 Future Outlook

The lightweight data concentrator architecture proposed in this thesis allows for expansion beyond currently tested UDP-based Metric Packets. The Cologne Chip GateMate™ M1A1 Evaluation Board provides the necessary hardware for additional connectivity through available PMOD header pins, six GPIO banks and high-speed SERDES connectors.

As only protocol code `V01` has been implemented, the Threshold Logic and Metric Packet Server can be extended to support further protocol codes, enabling different identifier lengths, support for other number formats and adjusted numerical dynamic ranges for metric values. Furthermore, a control unit could be implemented within the data concentrator allowing for dynamically configurable threshold values stored in BRAM or the software-based deassertion of interlock signals. To improve data integrity, future protocol codes could require self-checking error detection to be added to Metric Packets, ranging from simple parity bits to more robust methods such as a CRC. Considering usage in radiation heavy environments, certain safety-critical components such as the Threshold Logic units could be instantiated with TMR to create a Majority Voting System capable of combating accidental interlock assertions caused by bit-flips. Exploring the use of ECC BRAM configuration for the threshold lookup memory would provide an additional layer of protection against radiation-induced bit-flips.

Future work should evaluate alternative synthesis and implementation strategies to further optimize the system. This includes the impact of different passes within the synth_gatemate script in Yosys to evaluate how specific mapping configurations affect resource utilization. Additionally, various place and route strategies should be tested to minimize signal slack and maximize the operational FPGA system clock frequency.

Given the rising popularity of microscaling formats like MXFP4 in hardware acceleration for Artificial Intelligence (AI) applications, future research should evaluate their implementation on the GateMate™ FPGA fabric. Since the FPGA lacks dedicated Digital Signal Processing (DSP) slices, such explorations would determine implementation efficiency of low-precision arithmetic using CPEs.

Bibliography

[1] S. Yousaf Shah, A. Tsirou, P. G. Verdini, F. Hartmann, L. Masetti, G. H. Dirkes, R. Stringer, and M. Fahrer, "The cms tracker detector control system," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 604, no. 1, pp. 281–283, 2009, pSD8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900209001739

[2] C. Westbrook. (2012) Mr system layout. [Online]. Available: https://mriquestions.com/mr-system-layout.html

[3] Medicines and Healthcare products Regulatory Agency (MHRA), "Safety guidelines for magnetic resonance imaging equipment in clinical use," 2021, sections 5.3, 8.2 discuss incident investigation, system monitoring requirements, and engineered safeguards for subsystem failures. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/958486/MRI_guidance_2021-4-03c.pdf

[4] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)*. IEEE, 2008, pp. 1–6.

[5] H. Sandberg, "Radiation hardened system design with mitigation and evaluation on sram-based fpgas," Master's thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2016, m.Sc. thesis, demonstrates TMR/majority voting reduces SEU failure rates in commercial SRAM FPGAs. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1052529/FULLTEXT01.pdf

[6] W. Huang and E. J. McCluskey, "A memory coherence technique for online transient error recovery of FPGA configurations," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays (FPGA 2001)*, Monterey, California, USA, 2001.

[7] C. Chip. Gatemate (tm) fpga datasheet. Accessed: 2025-12-30. [Online]. Available: https://colognechip.com/docs/ds1001-gatemate1-datasheet-latest.pdf

[8] HTI, Otto von Guericke University Magdeburg. Fpga-centric-scis. Accessed: 2026-01-12. [Online]. Available: https://github.com/HTI-OVGU/FPGA-centric-SCIS

[9] MIDAS. Slow control system - documentation. Accessed: 2025-11-30. [Online]. Available: https://daq00.triumf.ca/MidasWiki/index.php/Slow_Control_System

[10] P. V. Chumakov, D. S. Egorov, R. V. Nagdasev, and V. B. Shutov, "Multi-purpose detector (mpd) slow control system, historical background, status and plans," in *Proceedings of the XXVI International Symposium on Nuclear Electronics & Computing (NEC'2017)*. Becici, Budva, Montenegro: CEUR-WS, 2017, pp. 288–292, veksler and Baldin Laboratory of High Energy Physics, Joint Institute for Nuclear Research.

[11] D. Thers *et al.*, "Micromegas as a large microstrip detector for the compass experiment," *Nucl. Instrum. Meth. A*, vol. 469, pp. 133–146, 2001.

[12] M. Collaboration, "Data acquisition and slow control interface for the mu2e experiment," Fermi National Accelerator Laboratory, Batavia, IL, USA, Tech. Rep. FERMILAB-PUB-21-750-PPD-SCD, 2021, mu2e internal/public technical report. [Online]. Available: https://lss.fnal.gov/archive/2021/pub/fermilab-pub-21-750-ppd-scd.pdf

[13] T. I. P. Prakash. Data concentrators: The core of energy and data management. Accessed: 2025-11-30. [Online]. Available: https://www.ti.com/lit/wp/spry248a/spry248a.pdf?ts=1764468936054

[14] L. Semiconductors. What is an fpga? Accessed: 2025-12-01. [Online]. Available: https://www.latticesemi.com/en/What-is-an-FPGA

[15] YosysHQ. Oss cad suite. Accessed: 2025-12-02. [Online]. Available: https://github.com/YosysHQ/oss-cad-suite-build

[16] Documentation for oss-cad-suite cologne chip fpga. Accessed: 2025-12-03. [Online]. Available: https://code.ovgu.de/iikt-hti/research/projects/colognechip

[17] CologneChip. Gatemate toolchain: Quickstart guide. Accessed: 2025-12-21. [Online]. Available: https://colognechip.com/programmable-logic/gatemate/gatemate-toolchain-quickstart/

[18] C. Chip. Gatematetm fpga evaluation board. Accessed: 2025-12-01. [Online]. Available: https://colognechip.com/programmable-logic/gatemate-evaluation-board/

[19] ——, *GateMateTM FPGA User Guide*, 2025, accessed: 2025-12-02. [Online]. Available: https://www.colognechip.com/docs/ug1001-gatemate1-primitives-library-latest.pdf

[20] P. Urban. A european 28nm fpga with an open-source toolchain and radiation qualification results. Accessed: 2025-12-02. [Online]. Available: https://indico.cern.ch/event/1587509/contributions/6690211/attachments/3140818/5574499/gatemate-fdf2025.pdf

[21] R. Jung, "Radiation qualification of the cologne chip gatemate a1 fpga," Master Thesis, Fachhochschule Dortmund, 2023.

[22] Amba axi-stream documentation. Accessed: 2025-12-04. [Online]. Available: https://documentation-service.arm.com/static/64819f1516f0f201aa6b963c

[23] WizNet. W5500. Accessed: 2025-12-03. [Online]. Available: https://wiznet.io/products/ethernet-chips/w5500

[24] CloudFlare. Glitches - timing issues of digital circuits. Accessed: 2025-12-21. [Online]. Available: https://www.cloudflare.com/learning/network-layer/what-is-a-protocol/

[25] Xilinx. Axi4-stream data fifo. Accessed: 2025-12-03. [Online]. Available: https://docs.amd.com/r/en-US/pg085-axi4stream-infrastructure/AXI4-Stream-Data-FIFO?tocId=9BStd4X7q~gctuNfFQO5Ug

[26] MeetOptics. Rise & fall times. Accessed: 2025-12-28. [Online]. Available: https://www.meetoptics.com/academy/rise-fall-times

[27] geeksforgeeks. Starvation and aging in operating systems. Accessed: 2026-01-09. [Online]. Available: https://www.geeksforgeeks.org/operating-systems/starvation-and-aging-in-operating-systems/

[28] GeeksforGeeks. (2023) Difference between priority inversion and priority inheritance. [Online]. Available: https://www.geeksforgeeks.org/operating-systems/difference-between-priority-inversion-and-priority-inheritance/

[29] RareSkills. Q number format. Accessed: 2025-12-08. [Online]. Available: https://rareskills.io/post/q-number-format

[30] A. I. Wawrzyniak *et al.*, "SOLARIS Interlock System Based on FPGA," in *Proc. PCaPAC'18*, ser. Personal Computers and Particle Accelerator Controls, no. 13. JACoW Publishing, Geneva, Switzerland, 01 2019, paper THP13, pp. 117–119. [Online]. Available: https://jacow.org/pcapac2018/papers/thp13.pdf

[31] P. Kolasiński, K. T. Poźniak, A. Wojeński, P. Linczuk, G. Kasprowicz, M. Chernyshova, D. Mazon, T. Czarski, J. Colnel, K. Malinowski, and D. Guibert, "High-performance fpga streaming data concentrator for gem electronic measurement system for west tokamak," *Electronics*, vol. 12, no. 17, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/17/3649

[32] Prometheus. Accessed: 2026-01-03. [Online]. Available: https://prometheus.io/

[33] Grafana. Accessed: 2026-01-03. [Online]. Available: https://grafana.com/grafana/

[34] YosysHQ. Yosys. Accessed: 2025-12-21. [Online]. Available: https://github.com/YosysHQ/yosys

[35] Ghdl. Accessed: 2025-12-03. [Online]. Available: https://github.com/ghdl/ghdl

[36] Gtkwave. Accessed: 2025-12-21. [Online]. Available: https://github.com/gtkwave/gtkwave

[37] Yosys - ghdl plugin. Accessed: 2025-12-21. [Online]. Available: https://github.com/ghdl/ghdl-yosys-plugin

[38] Nextpnr. Accessed: 2025-12-21. [Online]. Available: https://github.com/ghdl/ghdl-yosys-plugin

[39] Project peppercorn. Accessed: 2025-12-21. [Online]. Available: https://github.com/YosysHQ/prjpeppercorn

[40] Openfpgaloader. Accessed: 2025-12-21. [Online]. Available: https://github.com/trabucayre/openFPGALoader

[41] WizNet. W5500 datasheet. Accessed: 2025-12-10. [Online]. Available: https://docs.wiznet.io/pdf-viewer?file=%2Fassets%2Ffiles%2FW5500_ds_v110e-226ffec190c588b69f88d629789585e1.pdf

[42] tutorialspoint. Operating system - priority scheduling algorithm. Accessed: 2026-01-09. [Online]. Available: https://www.tutorialspoint.com/operating_system/os_priority_scheduling_algorithm.htm

[43] D. Anders, "Low-cost ethernet implementation on fpga for iot application," Research Project, Otto von Guericke University Magdeburg, 2025, unpublished project report. [Online]. Available: https://code.ovgu.de/iikt-hti/teaching/student-work/research-projects/w5500-daniel-anders/-/blob/main/w5500_with_errata.pdf

[44] RealDigital. Glitches - timing issues of digital circuits. Accessed: 2025-12-20. [Online]. Available: https://www.realdigital.org/doc/ff991122c4d78b897d96b4091f65eb31

[45] YosysHQ. synth_gatemate - synthesis for cologne chip gatemate fpgas. Accessed: 2025-12-26. [Online]. Available: https://yosyshq.readthedocs.io/projects/yosys/en/0.47/cmd/synth_gatemate.html

[46] YosysHQ, *Memory Handling*, YosysHQ, 2024, yosys Documentation. [Online]. Available: https://yosyshq.readthedocs.io/projects/yosys/en/latest/using_yosys/synthesis/memory.html

[47] client_prometheus. Accessed: 2026-01-03. [Online]. Available: https://prometheus.github.io/client_python/

[48] Prometheus. Querying basics. Accessed: 2026-02-28. [Online]. Available: https://prometheus.io/docs/prometheus/latest/querying/basics/

[49] Xilinx, Inc., *7 Series FPGAs Data Sheet: Overview (DS180)*, v2.6.1 ed., Xilinx, Inc., San Jose, CA, USA, September 2020, available at: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[50] TechWeb. What is ringing? Accessed: 2026-01-07. [Online]. Available: https://techweb.rohm.com/trend/glossary/16530/

[51] Armbian. Pine a64 and lts armbian 25. Accessed: 2026-01-10. [Online]. Available: https://www.armbian.com/pine64/

[52] Napi. [Online]. Available: https://docs.kernel.org/networking/napi.html

[53] IBM. Interrupt coalescing - ibm docs. Accessed: 2026-01-11. [Online]. Available: https://www.ibm.com/docs/en/aix/7.1.0?topic=options-interrupt-coalescing

[54] M. Åsberg, T. Nolte, M. Joki, J. Hogbrink, and S. Siwani, "Fast linux bootup using non-intrusive methods for predictable industrial embedded systems," in *2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2012, pp. 1–8.

A Diagrams and Finite State Machines

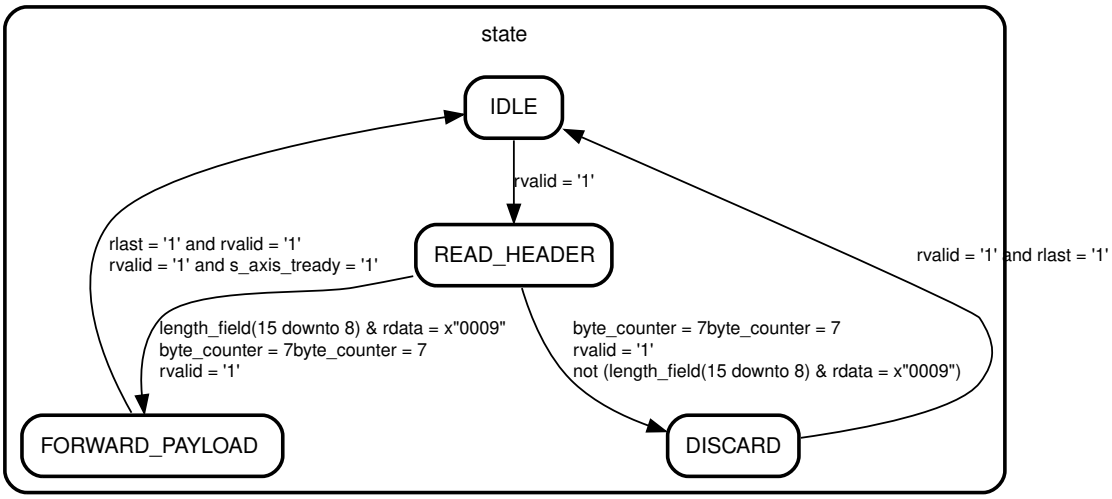


Figure A.1: UDP Packet Adapter - VHDL finite state machine

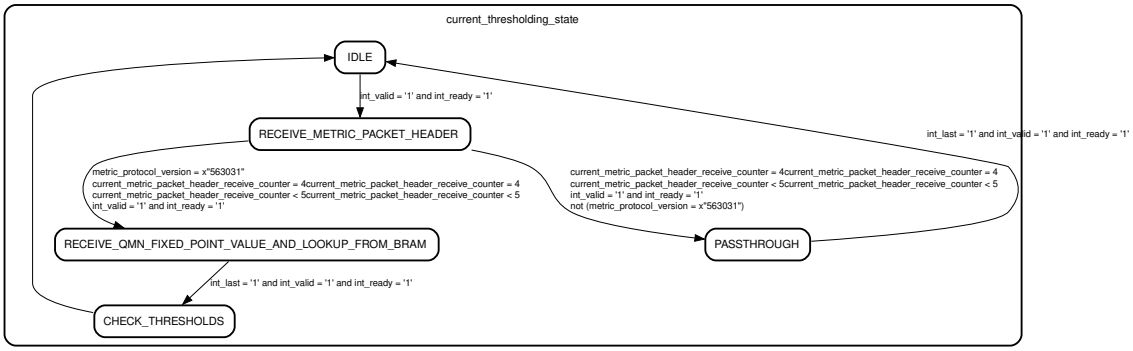


Figure A.2: Threshold Logic - VHDL finite state machine

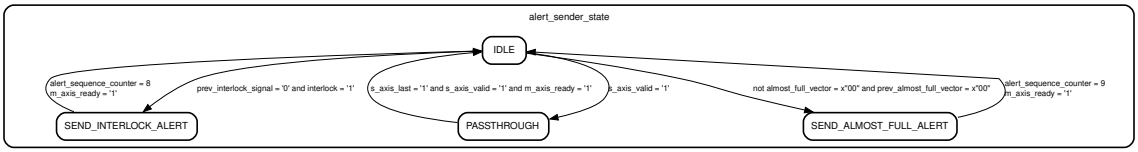


Figure A.3: Telemetry Sender - VHDL finite state machine

B Oscilloscope captures

This chapter lists screencaptures and measurements taken on the SIGLENT SDS2104X digital storage oscilloscope.

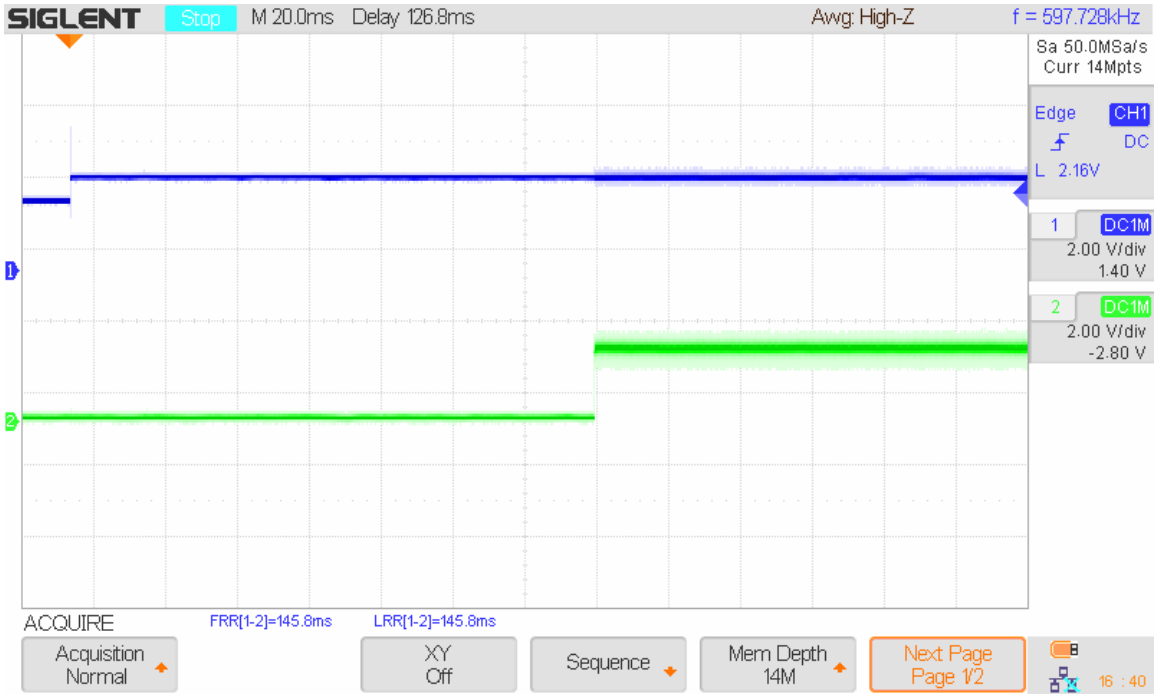
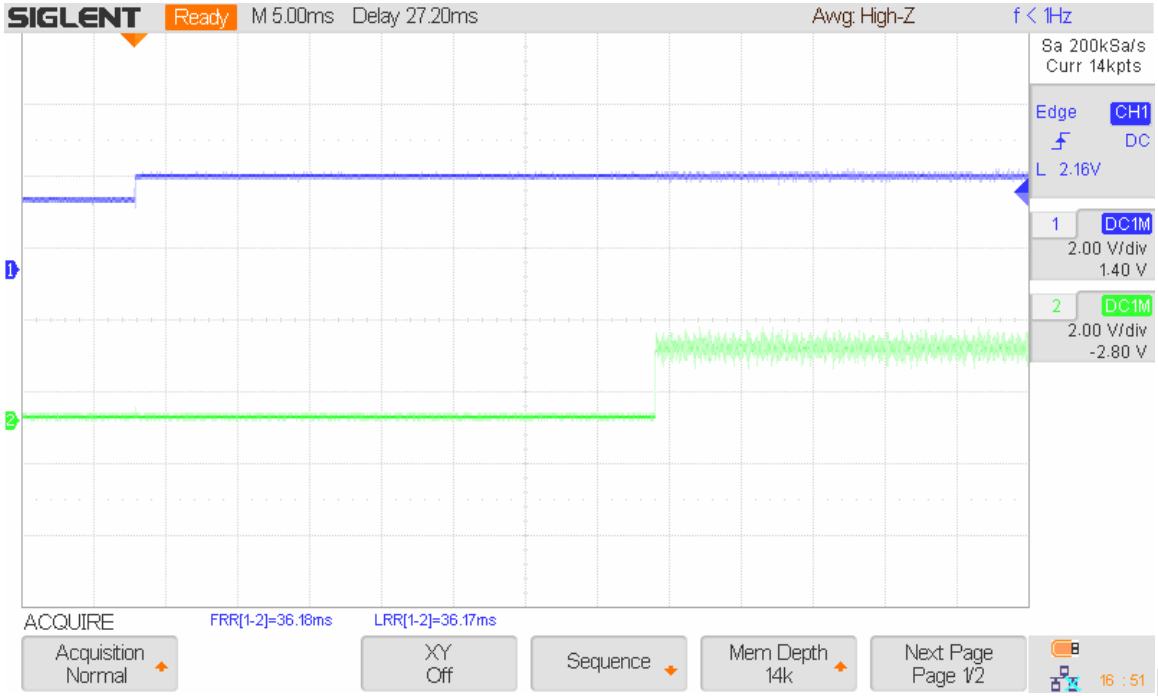


Figure B.1: Measured reconfiguration time of 145.8 ms between the release of the reset button and rising edge of the CFG_DONE signal using `spimode=single`



[!ht]

Figure B.2: Measured reconfiguration time of 36.18 ms between the release of the reset button and rising edge of the CFG_DONE signal using `spimode=quad`

Figure B.3: 20 MHz SCLK signal measurement; Probe connected to PMOD header pins and no load connected; Green signal with `FF_OBUF=true`, `SLEW=fast` and `DRIVE=9` pin constraints; Blue signal without additional pin constraints