Exploring FPGA-Accelerated Real-Time Image Processing for the AWAKE Camera Acquisition System

Development and Testing of an Improved Camera Acquisition System for the AWAKE Collaboration

Henrik Sørgård Johannessen

Supervised by: Prof. Mohammad Derawi, NTNU and Cedric Charrondiere, CERN

Department of Electronic Systems,

Norwegian University of Science and Technology (NTNU)

In collaboration with:

Conseil Européen pour la Recherche Nucléaire (CERN)

March 2023

Abstract

This report details the development and validation of an improved image processing and acquisition solution for the current AWAKE (Advanced Wakefield Experiment) data acquisition system at CERN (Conseil Européen pour la Recherche Nucléaire). The project presents a potential upgrade for the current camera system by enabling FPGA-accelerated real-time image processing, eliminating the need for time-consuming post-processing, reducing data storage requirements and improving the overall system performance. The purpose of the system is to capture and analyze the behaviour of plasma wakefields, which are created when a concentrated bunch of protons travel through plasma. The data collected by the camera system will be used to gain a deeper understanding of the physics of wakefield acceleration for future particle acceleration techniques and optimize the alignment of the beamlines connected to the experiment.

The cameras must trigger at a frequency of 9.97 Hz to extract a frame at the exact moment when the proton bunch enters the plasma chamber. To meet these timing requirements, the improved system takes advantage of National Instrument's PXIe-7915 Vision FPGA board to provide real-time image processing between frames. During the development phase, the system's FPGA board is tested to determine its maximum parallel processing capacity and minimal single-frame processing time. The development time is spent solving issues that arise with developing the LabVIEW RT and LabVIEW FPGA code for the hardware and iterating on the system to determine the limitations of the hardware.

The conducted tests show that the proposed system is capable of processing data well within the required timing constraints, and its maximum capacity for parallel processing exceeds the number of cameras currently in use in AWAKE, thereby enabling the addition of more cameras in the future. Ultimately, this project demonstrates the suitability of National Instrument's PXIe-7915 Vision FPGA board for real-time image processing between precise triggering intervals. The results from this report showcase its potential use in AWAKE as well as in other experimental research fields or industrial applications such as manufacturing or diagnostics.

Sammendrag

Denne rapporten detaljerer utviklingen og valideringen av en forbedret bildebehandlings- og anskaffelsessystem for det nåværende AWAKE (Advanced Wakefield Experiment) datainnhentingssystemet ved CERN (Conseil Européen pour la Recherche Nucléaire). Prosjektet presenterer en potensiell oppgradering for det nåværende kamera systemet ved å muliggjøre FPGA-akselerert sanntids bildebehandling, noe som eliminerer behovet for tidkrevende etterbehandling, redusere kravene til datalagring og forbedrer systemets samlede ytelse. Formålet med systemet er å fange og analysere oppførselen til plasma wakefields, som oppstår når en konsentrert bunt med protoner beveger seg gjennom plasma. Dataen samlet inn av kamerasystemet vil bli brukt for å få en dypere forståelse av fysikken bak wakefield akselerasjon for fremtidige partikkelakselerasjonsteknikker og for justeringer av strålelinjene som er tilkoblet eksperimentet.

Kameraene må utløses med en frekvens på 9,97 Hz for å hente ut et bilde i nøyaktige det øyeblikket når protonbunten går inn i plasmakammeret. For å møte disse tidskravene utnytter det forbedrede systemet National Instruments PXIe-7915 Vision FPGA-kort for å gi sanntids bildebehandling mellom utløserpulsene. Under utviklingsfasen blir systemets FPGA-kort testet for å bestemme dens maksimale kapasitet for parallell prosessering og den minimale enkelt-ramme behandlingstiden. Utviklingstiden ble brukt til å løse problemer som oppsto med under utviklingen av LabVIEW RT og LabVIEW FPGA -koden for maskinvaren og til å iterere på systemet for å fastslå begrensningene til maskinvaren.

De gjennomførte testene viser at det foreslåtte systemet er i stand til å behandle data godt innenfor de nødvendige tidsbegrensningene, og dens maksimale kapasitet for parallell prosessering overstiger antall kameraer som for øyeblikket er i bruk i AWAKE, som dermed muliggjør tillegg av flere kameraer i fremtiden. Til syvende og sist viser dette prosjektet egnetheten til National Instruments PXIe-7915 Vision FPGA-kort for sanntids bildebehandling mellom presise utløserintervaller. Resultatene fra denne rapporten viser potensialet for bruken i AWAKE samt i andre eksperimentelle forskningsområder eller industrielle anvendelser, som for eksempel produksjon eller diagnostikk.

Acknowledgements

I would like to express my sincere gratitude to everyone who supported me throughout this project. First and foremost, I would like to thank my CERN supervisor, Cedric Charrondiere and assisting supervisor, Slawomir Sudak, for providing me with invaluable guidance and mentorship. Their expertise and insight were essential in guiding me through the challenges of developing and testing the camera acquisition and processing system.

I would also like to thank the MTA section leader, Odd Øyvind Andreassen, for his continuous support and encouragement throughout my time at CERN. His leadership and guidance were instrumental in helping me integrate into the MTA section and collaborate effectively with the rest of the team.

At NTNU, I am grateful for the support and guidance provided by my course section leader, Halgeir Leiknes, and my project supervisor, Mohammad Derawi. Their feedback and input were critical in shaping my understanding of the theoretical concepts and practical applications of the camera system.

Finally, I want to express my gratitude to my colleagues at the MTA section, who welcomed me warmly and provided me with a stimulating and supportive work environment. Their expertise, insights, and sense of humor made my time at CERN both productive and enjoyable.

Thank you all for your support and encouragement. This project would not have been possible without your help.

Table of Contents

List of Figures

List of Tables

Glossary and Abbreviations

- FPGA: Field-Programmable Gate Array, an integrated circuit that can be programmed and reprogrammed after manufacturing to perform specific logic functions.
- PLD: Programmable Logic Device, an electronic component used to implement digital logic circuits.
- SoC: System-on-Chip, an integrated circuit that integrates multiple components such as microprocessors, memory, and interfaces into a single chip.
- HDL: Hardware Description Language, a specialized programming language used to describe digital circuits and systems.
- HLS: High-Level Synthesis, a design methodology that converts high-level programming languages into hardware description languages.
- ASIC: Application-Specific Integrated Circuit, a customized integrated circuit designed for a specific application.
- DSP: Digital Signal Processor, a specialized microprocessor designed to perform digital signal processing tasks.
- NRE: Non-Recurring Engineering, the one-time costs associated with the design and development of a custom integrated circuit.
- RTL: Register Transfer Level, a level of abstraction in digital circuit design that represents the behavior of the circuit in terms of the transfer of data between registers.
- FIR: Finite Impulse Response, a type of digital filter that has a finite impulse response.
- Decimation: A process of reducing the sampling rate of a signal by discarding some of the samples.
- Interpolation: A process of increasing the sampling rate of a signal by generating new samples from the existing ones.
- Polyphase filtering: A technique used to implement efficient resampling filters using a combination of multiple filters.
- LabVIEW: Laboratory Virtual Instrument Engineering Workbench
- NI: National Instruments
- RTOS: Real-Time Operating System
- VI: Virtual Instrument
- PXI: PCI eXtensions for Instrumentation
- VHDL: VHSIC Hardware Description Language
- VHSIC: Very High-Speed Integrated Circuit
- CMOS: Complementary Metal-Oxide-Semiconductor
- SOPC: System-on-a-Programmable-Chip
- DDR: Double Data Rate
- SDRAM: Synchronous Dynamic Random-Access Memory
- IP: Internet Protocol
- OS: Operating System
- URL: Uniform Resource Locator

1 Introduction

1.1 Background

The AWAKE (Advanced Wakefield Experiment) collaboration is a cutting-edge particle physics project conducted at CERN, the European Organization for Nuclear Research. The primary objective of AWAKE is to explore new particle acceleration techniques that offer the potential for more efficient and compact high-energy particle beams. By harnessing proton-driven plasma wakefield acceleration, AWAKE aims to achieve this acceleration over a shorter distance than conventional methods, which rely on large-scale facilities such as linear accelerators and circular synchrotrons.

The successful acceleration of electrons from an initial energy of 19 MeV to nearly 2 GeV within a 10-meter plasma cell in May 2018[8] marked a significant milestone for the AWAKE collaboration. However, the project's potential for groundbreaking discoveries depends on the accurate collection and analysis of experimental data, a task that is complicated by the vast amounts of information generated during the runtime of the experiment.

To address this challenge, the Measurement, Tests, Analysis, and Electronics section (MTA) has collaborated with AWAKE to develop a system of cameras to collect images of the experiment, which play a crucial role in both data acquisition and beam alignment. These cameras are strategically positioned along the beam line and the plasma cell to capture real-time visual information about the particle interactions and the plasma wakefields created during the acceleration process.[7]

However, attempts at implementing real-time image processing encountered several significant obstacles. The earlier proposed system focused on traditional methods that relied on serialized processing and were unable to provide the required processing power and stability, leading to the loss of crucial data and mismatching frame and trigger timestamps. To address these issues, a trigger management algorithm was developed, but it was ultimately deemed insufficient as it failed to meet the required frequency of 9.97 Hz, often resulting in the loss of more than half of the images.

As a consequence, the existing system has abandoned real-time image processing in favour of post-processing saved data. While this approach has resolved the issue of data loss, it does not exploit the benefits of real-time processing, like real-time monitoring and control, reducing the required bandwidth, and minimising data storage demands. As a result, the current system limits the efficiency and potential effectiveness of the experiments.

1.2 Justification

The need for a more robust and reliable real-time image processing system is clear, as it would allow for more efficient and accurate data collection, potentially leading to new insights and discoveries in the field of particle acceleration physics. A system that can meet the stringent requirements of the AWAKE collaboration would not only benefit the project but could also serve as a blueprint for other large-scale, high-energy physics experiments that rely on real-time data processing and analysis.

In conclusion, the development of a new, more capable real-time image processing system is crucial for the success of the AWAKE collaboration and the broader scientific community.

1.3 Proposing FPGA-acceleration for real-time image processing

The camera acquisition system in place at AWAKE today does not incorporate any storage-saving real-time control methods and serves only as an interim system until an improved camera acquisition system is in place.

This project serves as a collaborative effort between AWAKE and MTA to solve the issues with

the previous solution and propose an improved real-time image processing system for collecting data from the experiment and aligning the beam line. Our proposed solution incorporates Field Programmable Gate Arrays (FPGA) to achieve reliable high-speed real-time processing. The FPGA-based system is potentially fast enough to process multiple high-resolution images in real-time at the required frequency of 9.97 Hz.

This thesis covers the development and evaluation of an FPGA-accelerated real-time image acquisition and processing system. It provides valuable insights into the FPGA hardware and the surrounding supplementary system in preparation for the deployment of an improved camera acquisition system. The contents of this thesis will focus on the development process of the system, and how it is used in validating and testing the new hardware.

1.4 Problem Definition and Research Question

The problem definition chosen for this study is: "Can an FPGA-accelerated real-time data processing system provide sufficiently fast processing speeds and throughput to process the images form all the cameras between each trigger pulse of a 9.97 Hz signal, and can the system accommodate larger camera sensors or additional cameras in the future?"

The research question is: "What are the benefits and limitations of using FPGA-accelerated real-time processing compared to post-processing systems?"

While the problem definition clearly identifies the objective of the research, the research question guides the direction of the study and creates the groundwork for developing the methodology and analysis.

1.5 Scope and Boundaries

This project focuses on the development of an enhanced camera acquisition system for the AWAKE collaboration, emphasizing the application of FPGA technology for real-time image processing. The research objectives include understanding the benefits and limitations of FPGA-based systems for high-speed real-time image processing in data acquisition, designing an improved camera acquisition system tailored to the AWAKE experiments, and assessing the performance of the developed system in relation to the defined problem.

Moreover, the hardware and software will be tested for their potential for future expansion. The components and subsystems will undergo validation in a testbench setup at the MTA section's computer laboratory. The project's scope is confined to the exploration of the NI PXIe-7915 Vision FPGA board combined with the NI PXIe-1095 chassis, while alternative FPGA chips and programming methods will be detailed in the literature review section. The selected constraints are justified by time, budget, and resource limitations. The project aims to address AWAKE's data processing needs within the available hardware and resources without incurring unnecessary additional costs.

The research boundaries establish the framework within which the investigation is conducted, encompassing the following aspects:

- **FPGA (Field Programmable Gate Array)**
- **Real-time image processing**
- **LabVIEW programming**
- **FPGA system design**
- **System determinism**

These boundaries ensure the relevance of the research topics concerning the research question. Subjects that do not demonstrate a justifiable relevance to the field of research will not be explored.

The addressed topics are also constrained by their relevance to the field of electronics engineering. Subjects beyond the justifiable relevance to this research will not be covered.

The report will document the system development process, emphasizing any challenges encountered and the solutions implemented to overcome them. It will also examine the utilized electronics and pertinent technologies.

Through a comprehensive account of the development process and an evaluation of the developed system's performance, this research aims to contribute to the knowledge base on FPGA-based systems for high-speed real-time image processing. The study will also offer insights into the benefits and limitations of LabVIEW programming for FPGA and the use of this technology for data acquisition. FPGA-based image processing systems present a promising method for use in future experimental physics, as they enable faster and more efficient processing of experimental data. The ease of implementation provided by the LabVIEW FPGA and LabVIEW RT platforms also facilitates quicker and simpler development of similar systems, expanding the potential applications beyond niche experimental research and making the technology accessible to a broader audience.

1.6 Structure

The thesis is structured in a way to make it easily digestible. It covers a large project and needs to condense the work so that only the important topics are covered. It tries to follow best practices for conveying information without diverging from the style that is required for an NTNU bachelor's thesis.

The introduction serves three important purposes: to provide an overview of the background for the report's content, to engage the interest and motivate the reader to read on and to present the problem that the research tries to solve.

The literature review is meant to support the reader with answers that may arise during the reading and provide a theoretical foundation for understanding the topics. If the reader has sufficient background knowledge, this section could be skipped or serve only as a summary of the relevant research.

With the prerequisites from the literature review, the methodology breaks down the project and tries to convey the approach taken to achieve the system. A highlight of the most important choices made during development will be explained in detail to provide a better understanding of the end product.

The system design and -implementation section is dedicated to explaining the system in detail, going through the system architecture, as well as showing the software solutions that were developed.

After the project is covered in detail, the results from the testing will be presented. An analysis and discussion section is provided to answer whether the testing and results are sufficient in light of the project's goal and some suggestions for further research will be given.

In the end, the thesis will be concluded with a summary of the key points and a definitive answer to the research problem and question will be provided.

2 Literature Review

The literature review for this thesis aims to provide the reader with essential knowledge about the subject of discussion, serving as the theoretical framework for the project. The review is organized into six main sections: FPGA and Hardware, Comparison with Other Technologies, FPGA design Flow, Software and Tools, Algorithms and Techniques, and Determinism in Real-Time systems. Within each section, relevant topics are discussed in detail, including explanations of abbreviations, acronyms, and technical terms. The overall goal is to equip the reader with the necessary knowledge to easily digest the topics covered in the rest of the thesis.

2.1 FPGA and Hardware

2.1.1 Field-Programmable Gate Arrays (FPGAs)

Field-Programmable Gate Arrays (FPGAs) are integrated circuits that can be programmed and reprogrammed after manufacturing to perform specific logic functions [21]. FPGAs have become an essential part of modern technology due to their flexibility, high performance, and cost-effective solutions to complex digital designs. This subsection provides an overview of FPGAs, their evolution, and their importance in modern technology.

Evolution and development of FPGAs FPGAs were first introduced in the 1980s and have since evolved significantly. The earliest FPGAs consisted of simple programmable logic devices (PLDs) that were used to implement basic combinatorial and sequential logic functions [41]. In the late 1980s and early 1990s, FPGAs with larger capacity and more complex architectures were introduced, allowing for the implementation of more complex digital circuits and system-on-chip (SoC) designs [19]. In the mid-1990s, FPGAs with embedded memory blocks and digital signal processing (DSP) capabilities were introduced, enabling the implementation of more advanced signal processing algorithms [9]. Since then, FPGAs have continued to evolve, with improvements in capacity, speed, power consumption, and reliability.

FPGA design techniques and architectures FPGA design techniques and architectures have also evolved significantly over time. Early FPGAs were programmed using hardware description languages (HDLs) such as VHDL and Verilog [5]. However, as the complexity of FPGA designs increased, new design methodologies and tools were developed. Today, FPGAs are typically programmed using high-level synthesis (HLS) tools that automatically convert software algorithms into hardware implementations [42]. FPGA architectures have also become more complex, with the integration of specialized functional blocks such as embedded processors, DSP blocks, and high-speed serial transceivers [18].

Applications and Use Cases FPGAs have a broad range of applications, including data centers, wireless communication, automotive, aerospace, military, medical devices, and scientific research [28]. For instance, in data centers, FPGAs are used for accelerating compute-intensive workloads such as machine learning, data analytics, and cryptography [34]. In wireless communication, FPGAs are employed for baseband processing, software-defined radio, and wireless backhaul [17].

In the automotive industry, FPGAs play a significant role in advanced driver assistance systems (ADAS) and autonomous driving [35]. They are also used in aerospace and military applications for radar signal processing, image and video processing, and secure communication [27]. In the medical field, FPGAs contribute to digital signal processing and medical imaging [38]. Lastly, in scientific research, FPGAs are utilized for high-performance computing and particle accelerator control systems [43].

FPGAs are also used in the field of finance for high-frequency trading, where speed and low latency

are critical for executing trades in real-time [2]. Additionally, FPGAs are increasingly being used in the field of artificial intelligence, particularly in the training and inference of deep neural networks, due to their ability to accelerate computationally intensive tasks [44]. FPGAs are also utilized in the development of digital audio and video processing systems, where they can efficiently perform tasks such as compression and decompression, filtering, and encryption [17].

Advantages and Disadvantages of FPGAs FPGAs offer numerous advantages over other digital design technologies, such as Application-Specific Integrated Circuits (ASICs) and Digital Signal Processors (DSPs). Some of these advantages include reconfigurability, shorter time-to-market, lower non-recurring engineering (NRE) costs, and the ability to create custom hardware accelerators for specific applications [41]. However, FPGAs also come with some disadvantages, such as higher power consumption, lower performance compared to ASICs, and the need for specialized knowledge and tools for FPGA programming [19].

2.2 Comparison with Other Technologies

2.2.1 FPGAs vs. ASICs vs. DSPs

This subsection presents a more detailed comparison between FPGAs, Application-Specific Integrated Circuits (ASICs), and Digital Signal Processors (DSPs), discussing their respective strengths and weaknesses in different application scenarios.

Performance and Flexibility FPGAs offer a high degree of flexibility and reconfigurability compared to ASICs, which are custom-built for a specific application and cannot be reprogrammed. While ASICs generally provide higher performance and lower power consumption, FPGAs enable rapid prototyping and shorter time-to-market. DSPs, on the other hand, are specialized processors designed for signal processing tasks, offering a balance between the flexibility of FPGAs and the performance of ASICs.

Cost and Development Time ASICs typically have high non-recurring engineering (NRE) costs and long development times, making them more suitable for high-volume applications where the initial investment can be amortized over a large number of units. FPGAs, with their lower NRE costs and faster design cycles, are better suited for low-to-medium volume applications or when rapid prototyping is a priority. DSPs generally have lower development costs compared to ASICs, but may not provide the same level of customization as FPGAs.

2.3 FPGA Design Flow

2.3.1 Overview of the FPGA Design Process

The FPGA design process consists of several stages, from initial design entry to final bitstream generation. This subsection provides a brief overview of these stages, giving the reader a better understanding of the overall FPGA design flow.

Design Entry Design entry is the first stage in the FPGA design process, where the designer describes the desired digital system using a hardware description language (HDL) like VHDL or Verilog, or a high-level programming language such as C, C++, or OpenCL. The choice of language depends on the designer's preferences and the level of abstraction they wish to work with.

Synthesis During synthesis, the design description is converted into a gate-level netlist that represents the digital system as a set of interconnected logic gates. This stage involves optimizing

the design for area, performance, and power consumption, based on the designer's constraints and objectives.

Placement and Routing Placement and routing is the process of assigning the digital system's components to specific locations on the FPGA device and connecting them using programmable routing resources. This stage aims to minimize resource usage and interconnect delays, ensuring that the design meets its performance and power requirements.

Bitstream Generation Finally, the completed design is converted into a bitstream, which is a binary file that configures the FPGA device to implement the desired digital system. The bitstream is loaded onto the FPGA, allowing it to perform the specified functions.

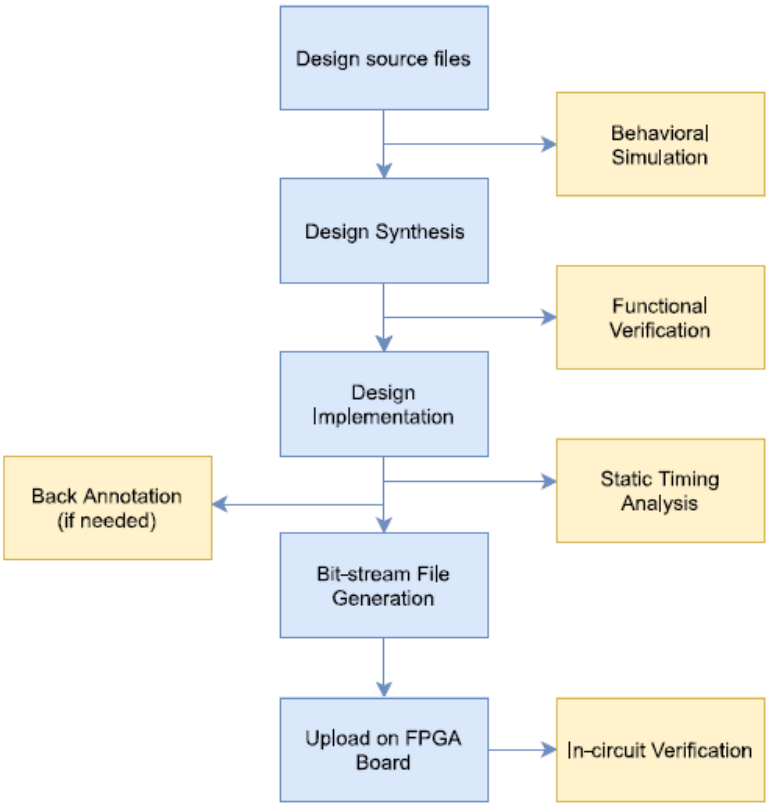

Figure 1: Diagram depicting the FPGA design flow[16]

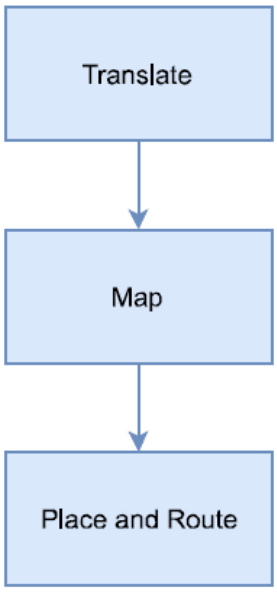

Figure 2: Diagram depicting the implementation process within the FPGA design flow[16]

Figure1. Shows a diagram-depiction of the design flow for a generic FPGA chip. The design implementation process is depicted in Figure 2, and includes producing the physical layout for the FPGA. The 3-stage process translates the previously written netlists, divides them into sub-blocs and maps them to specific locations on the physical FPGA chip, and routed the connections between the blocks. This process takes into consideration the architecture of the FPGA chip to ensure efficient space utilization, i.e., placing I/O processing near I/O pins to save space and time.

2.4 Software and Tools

2.4.1 High-Level Synthesis (HLS) Tools

High-Level Synthesis (HLS) tools play a crucial role in modern FPGA design flow, allowing designers to convert high-level programming languages such as C, C++, and OpenCL into hardware description languages (HDLs) like VHDL and Verilog [42]. These tools help designers create complex digital systems more efficiently and with a faster time-to-market compared to traditional HDL-based design methods [5]. This subsection discusses the importance, features, and popular HLS tools available in the industry.

Importance of HLS tools The increasing complexity of digital systems has made manual HDL-based design methods time-consuming and error-prone. HLS tools help overcome these challenges by automating the hardware design process and allowing designers to work with higher levels of abstraction, making the design process more intuitive and efficient [42]. HLS tools also enable rapid design space exploration, which helps designers identify and optimize the most suitable hardware architectures for a given application [5].

Features of HLS tools HLS tools typically provide a range of features to facilitate the hardware design process. Some of these features include automatic pipelining, loop unrolling, resource sharing, and memory partitioning [42]. These optimizations can significantly improve the performance, power consumption, and resource utilization of FPGA designs. HLS tools also often provide application-specific libraries, such as those for image processing and machine learning, which can further accelerate the design process [5].

Popular HLS Tools Several HLS tools are available in the industry, catering to different design requirements and user preferences. Some of the most popular ones include Xilinx Vivado HLS, Intel FPGA SDK for OpenCL, and Catapult C Synthesis [42]. Each tool offers its unique features and optimizations, making it essential for designers to carefully evaluate the most suitable option for their specific application.

2.4.2 FPGA Programming Environments

In addition to HLS tools, various programming environments are available to help designers implement their FPGA designs. These environments typically include support for hardware description languages (HDLs), simulation tools, and synthesis tools, among others. This subsection discusses some of the most widely used FPGA programming environments in the industry.

Xilinx Vivado Design Suite The Xilinx Vivado Design Suite is a comprehensive FPGA design environment that supports the entire design flow, from RTL design and synthesis to implementation and debugging [21]. Vivado includes support for both VHDL and Verilog HDLs, as well as the Vivado HLS tool for high-level synthesis. It also offers advanced features such as constraint-driven design, logic optimization, and power analysis to help designers create efficient and high-performance FPGA designs.

Intel Quartus Prime Intel Quartus Prime is another popular FPGA design environment that supports Intel FPGA devices [11]. Like Vivado, Quartus Prime provides support for both VHDL and Verilog HDLs, as well as the Intel FPGA SDK for OpenCL for high-level synthesis. Quartus Prime also includes advanced features such as design partitioning, incremental compilation, and power-driven layout to help designers optimize their FPGA designs for performance, power, and cost.

Lattice Diamond Lattice Diamond is an FPGA design environment specifically designed for Lattice Semiconductor FPGA devices [36]. It supports both VHDL and Verilog HDLs and includes a range of features to help designers create efficient and high-performance FPGA designs. Some of these features include design exploration, power estimation, and an integrated design flow that streamlines the design process.

LabVIEW FPGA LabVIEW FPGA is a graphical programming environment developed by National Instruments for designing and implementing FPGA designs on NI FPGA hardware platforms [22]. The environment is based on the LabVIEW programming language, which uses dataflow programming and a graphical syntax to represent the structure and behavior of digital systems. The LabVIEW FPGA compilation process involves several stages, including design entry, synthesis, place-and-route, and bitstream generation. Designers create their FPGA designs using LabVIEW's graphical editor, which allows for efficient representation of complex logic and parallel processing structures. Once the design is complete, it is synthesized into an intermediate netlist representation, which is then optimized for the target FPGA device. The place-and-route stage determines the physical layout of the design on the FPGA, assigning specific FPGA resources to each logic element and interconnecting them based on the netlist. Finally, a bitstream is generated, which can be loaded onto the FPGA to configure it with the implemented design. LabVIEW FPGA also includes simulation and debugging tools, as well as integration with NI's hardware platforms for seamless system development and deployment.[3, 40, 29]

2.5 Algorithms and Techniques

2.5.1 Digital Signal Processing (DSP) Algorithms

Digital Signal Processing (DSP) algorithms play a critical role in a wide range of applications, including communications, audio and video processing, and sensor data processing. FPGAs, with their parallel processing capabilities and customizable hardware, are well-suited for implementing DSP algorithms. This subsection provides an overview of common DSP algorithms and their FPGA implementations.

Finite Impulse Response (FIR) Filters Finite Impulse Response (FIR) filters are a fundamental building block in digital signal processing and are commonly used for filtering, smoothing, and signal detection tasks [33]. FPGA implementations of FIR filters can take advantage of the parallel processing capabilities of FPGAs to achieve high-performance and low-latency filtering operations [17].

A FIR filter can be represented mathematically as a convolution between the filter coefficients h[n] and the input signal x[n]:

$$y[n] = \sum_{k=0}^{M-1} h[k]x[n-k] \tag{1}$$

where M is the filter length, h[k] are the filter coefficients, and x[n] is the input signal. This equation computes the output signal y[n] at each time index n.

Resampling Techniques Resampling techniques are essential in digital signal processing for various applications, including audio and image processing, where signals need to be converted between different sampling rates or resolutions. Common resampling techniques include decimation, interpolation, and polyphase filtering, which can be efficiently implemented on FPGAs due to their inherent parallelism and high computational capabilities [13].

Decimation is the process of reducing the sampling rate of a signal by removing samples or averaging adjacent samples, while interpolation increases the sampling rate by inserting new samples between existing ones. Polyphase filtering is a more advanced resampling technique that allows for efficient implementation of both decimation and interpolation, by dividing the input signal into multiple sub-bands and processing each sub-band independently. FPGAs can be utilized to implement high-performance resampling algorithms by parallelizing the processing of sub-bands, leading to faster and more efficient designs [17].

Decimation is a process of reducing the sampling rate of a signal by a factor of L. It can be achieved by first low-pass filtering the signal and then downsampling it:

$$y[n] = x[Ln] \tag{2}$$

where x[n] is the input signal, and y[n] is the downsampled output signal.

Interpolation is a process of increasing the sampling rate of a signal by a factor of M. It can be achieved by inserting M-1 zero-valued samples between each input sample, followed by low-pass filtering:

$$y[nM] = \sum_{k=0}^{L-1} x[k]h[nM-k] \tag{3}$$

where x[k] is the input signal, h[n] is the interpolation filter, and y[nM] is the interpolated output signal.

Polyphase filtering is a more advanced resampling technique that allows for efficient implementation of both decimation and interpolation. It involves dividing the input signal into multiple sub-bands and processing each sub-band independently. The output is then reconstructed by combining the sub-band outputs using a polyphase filter.

Histogram Creation and Pixel Brightness Level Processing In image processing, histograms are often used to analyze the distribution of pixel brightness levels in an image. Histograms can provide valuable insights into the image's characteristics, such as contrast and dynamic range, and can be used for various image enhancement techniques, including histogram equalization and adaptive contrast enhancement [15].

FPGAs can be employed to efficiently compute histograms and process pixel brightness levels in real-time, taking advantage of their parallel processing capabilities and customizable hardware resources. One common approach to implementing histogram creation on FPGAs involves dividing the image into multiple sub-regions and processing each sub-region in parallel. This parallelism allows for faster computation of histograms and enables real-time analysis of large images or video streams [37].

In addition to computing histograms, FPGAs can also be utilized for implementing various image enhancement techniques based on pixel brightness levels. For example, histogram equalization is a technique that redistributes the pixel brightness levels to achieve a more uniform histogram, resulting in improved contrast and brightness in the output image [15]. FPGAs can efficiently implement histogram equalization algorithms by parallelizing the processing of pixel brightness levels, leading to faster and more efficient designs suitable for real-time applications [26].

A histogram of an image can be represented as a function h(i) that counts the number of pixels with intensity level i. It can be computed using the following equation:

$$h(i) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \delta(i - I(x, y)) \tag{4}$$

where N and M are the image dimensions, I(x, y) is the intensity value of the pixel at position (x, y), and δ is the Dirac delta function.

Histogram equalization is a technique that modifies the pixel intensities to obtain a more uniform histogram. It can be achieved by computing the cumulative distribution function (CDF) of the image histogram and then mapping the pixel intensities to their corresponding CDF values:

$$s = T(r) = \sum_{j=0}^{r} p(j) \tag{5}$$

where s is the new pixel intensity, r is the old pixel intensity, and p(j) is the probability of intensity level j in the image histogram. This mapping function can be efficiently implemented on an FPGA by parallelizing the processing of pixel brightness levels.

2.6 Determinism in Real-Time Systems

Determinism is the ability of a system to perform a specific task within a predictable and well-defined time frame. This is a critical requirement in real-time systems, which are designed to execute tasks within strict timing constraints.

In real-time systems, timing predictability is essential, and any unexpected delay or variability in the timing of system operations can lead to system failures, and even safety-critical issues.

Thus, a system is considered deterministic if it can execute a set of operations within a specific time frame, with predictable and repeatable results. A deterministic system is essential for applications such as aviation, automotive, and medical systems, where safety is critical.

2.6.1 Importance of Determinism in Real-Time Systems

The importance of determinism in real-time systems cannot be overstated. Real-time systems must respond to external stimuli within strict time constraints, with reliable and predictable results.

For example, in an automotive system, the response time of the airbag system must be precise and predictable, responding to a crash within milliseconds. Any delay in the response could lead to severe injury or loss of life.

In addition to safety-critical systems, determinism is also essential in industrial automation, robotics, and multimedia systems, where real-time processing is required.[20, 39, 10, 4]

2.6.2 Impact of Non-Determinism in Real-Time Systems

Non-determinism, or unpredictable behavior, in real-time systems can have severe consequences, such as system crashes, data loss, or even injury or loss of life in larger industrial or vehicular controll systems. Non-determinism can occur due to various factors, such as system overload, software errors, or hardware faults.

For instance, a system overload due to excessive load on the processor or memory can lead to unpredictable system behavior, leading to system crashes or incorrect results. Similarly, a software error, such as a race condition, can cause unpredictable behavior in the system, leading to incorrect results or system crashes.[20, 39, 10, 4]

2.6.3 Achieving Determinism in Real-Time Systems

Achieving determinism in real-time systems requires careful consideration of the system architecture, hardware, and software design.

Hardware factors, such as processor speed, cache size, and memory size, can significantly impact the system's determinism. Additionally, the system's software design must ensure that the system's operations are executed within a specific time frame.

To achieve determinism, real-time systems often use techniques such as preemptive scheduling, priority-based scheduling, and resource reservation. These techniques ensure that system resources are allocated and used efficiently, without impacting the system's determinism. [20, 39, 10, 4]

3 Methodology

This section outlines the approach taken to address the specific problem and research question mentioned earlier, which centres on determining the benefits and limitations of employing an FPGA-accelerated processing approach for achieving real-time image processing for the AWAKE project. The methodology comprises several subsections that combined to clarify the approach taken to achieve the results. Firstly, the hardware selection for the project will be justified, focusing on the FPGA board and the advantages it serves over other methods. This is followed by discussing the rest of the system hardware and detailing the reasons for the selection. Lastly, the development tools and methods used to evaluate the system's performance will be outlined as well as touching on potential error sources related to the test procedure.

3.1 FPGA for Real-Time Image Processing

FPGAs (Field-Programmable Gate Arrays) are increasingly used for real-time image processing applications due to their hardware parallelism, flexibility, and high-performance capabilities. In AWAKE, high-resolution images are captured from several cameras at 10 FPS (Frames Per Second) continuously throughout the run of the experiment. Employing a real-time image processing system at AWAKE could prove to limit the strain on the network and database, as well as improve the performance and usability of the system. [6]

3.1.1 Advantages of FPGA

FPGA boards offer several advantages over traditional processors. FPGAs can perform multiple operations in parallel, which is crucial for processing large amounts of data in real time. They can also process data with very low latency, which is important in real-time image processing applications where even small delays can cause significant issues. In addition, FPGA boards can be customized for specific applications, making them ideal for logic implementations. Finally, FPGA boards are power-efficient and can be used in applications where power consumption is a concern.[30, 14]

3.1.2 Comparison to Other Approaches

FPGA boards offer a unique combination of parallelism, low latency, customization options, and power efficiency for real-time image processing applications. Other real-time processing hardware include: GPUs (Graphics Processing Units), which are highly customizable and can perform many operations simultaneously; and DSPs (Digital Signal Processors), which are specialized processors designed for handling digital signal processing tasks, such as filtering, modulation, demodulation, and other operations involved in processing signals like audio, video, and images. While DSPs do not have the same flexibility or parallelism as FPGAs and GPUs, FPGAs feature dedicated DSP blocks or circuits that can perform processing operations in hardware. De Kock et al. (2014)[25] conducted a comparative study on accelerating real-time image processing on FPGAs, GPUs, and DSPs. They concluded that FPGAs offer a unique combination of parallelism, low latency, customization options, and power efficiency for real-time image processing applications, and that FPGAs outperform GPUs in speed and throughput due to the dedicated DSP blocks.

3.2 Hardware Selection

The AWAKE project's camera acquisition system in place today consists of several cameras placed along the beamlines of the wakefield experimental setup. These cameras are connected to a National Instruments (NI) rack-mounted server, the PXIe-1095 chassis. This server is responsible for acquiring the images from all the cameras and sending them to storage for post-processing.

This project functions as an addition to the current system and employs the same PXIe-1095 chassis with the addition of the NI PXIe-7915 Vision FPGA board for FPGA-accelerated processing. The selection of the FPGA board is based on several factors related to compatibility and performance capabilities. The PXIe-7915 board takes advantage of the high-speed, low-latency communication link with the PXIe-1095 to provide hardware-level parallelism for processing multiple high-resolution image operations simultaneously. The combination of FPGA-based processing and the PXIe-1095 chassis ensures deterministic real-time performance, which is critical for systems where timing and synchronization are important.

The system also composes a selection of complementary hardware, necessary for the operation of the system which will be discussed in the following sections.

3.2.1 PXIe-1095 chassis

The PXIe1095 chassis serves as the main hardware component. It is responsible for tying all of the elements together, initializing the cameras and FPGA, receiving and logging data, and running test procedures. The software for this is written on a Windows-based host computer and remotely deployed to the Linux-RT-based operative system. Built on the LabVIEW RT programming software, the system is both scalable and easily integrated with the FPGA. The rack-mounted chassis interacts with the cameras and the FPGA, by delivering images from the camera to the FPGA. It then takes the processed images from the FPGA, does some calculations and logs these to a file.

The PXIe (PCI eXtensions for Instrumentation Express), an enhanced version of the PXI (PCI eXtensions for Instrumentation) standard, is a Linux RTOS (Real-Time Operative System) based server designed to hold and power PXI Express modules. It is compact and reliable, making it suitable for space-constrained industrial environments, and making it easy to integrate into existing equipment or facilities like AWAKE. The chassis provides 18 expansion slots with up to 32 GB/s system bandwidth, enabling fast data transfers and reduced latency which makes it ideal for high-performance test and measurement applications.[24, 31]

3.2.2 PXIe-7915 FPGA Board

The FPGA board is the focus of this system, it is responsible for image processing and is what this project is trying to evaluate. It gets the images from the server and processes them in real-time using the programmable logic cells and dedicated Digital Signal Processor (DSP) blocks. The processed images are then sent back to the server for storage and further analysis. The FPGA is programmed using the LabVIEW FPGA extension. This allows the specific processing operations to be deployed to the FPGA by a drag-and-drop selection.

The PXIe-7915 FPGA board is specifically designed to work in conjunction with the PXIe-1095 chassis and takes advantage of its high-speed communication links to provide hardware-level parallelism. It uses a Xilinx Virtex-7 FPGA for high-speed processing and has onboard DDR3(Double Data Rate 3) SDRAM (Synchronous Dynamic Random Access Memory), two high-speed serial transceivers, and a Gigabit Ethernet port for communication with other devices. The FPGA has a high number of logic cells, distributed RAM, and DSP slices that enable it to perform complex parallel signal processing tasks, making it especially well-suited for real-time image processing applications. [23, 32]

3.2.3 NI VirtualBench VB-8012

The 9.97 Hz trigger signal is responsible for synchronising the cameras to the precise timing of the particle beams in the experimental area of AWAKE. This signal is derived from the CERN Synchronization Frequency (CSF) which is responsible for the operation order for the different particle experiments at CERN. At AWAKE it is responsible for a set of operations including capturing an image at the exact time the particles enter the frame.

NI's VirtualBench VB-8012 is a compact, digital benchtop instrument featuring an oscilloscope, function generator, digital multimeter and more. In this project, it is used as a function generator to provide a 10 Hz square wave signal to the cameras to simulate the 9.97 Hz trigger signal. Simulating the 9.97 Hz signal as a 10 Hz square wave is adequate because the tests performed during the development focus on the time it takes to process a frame and is not reliant on the exact frame rate.

3.2.4 Basler ace GigE

The cameras employed in this system are of the same type used in the current AWAKE cam-acq system. Being placed very close to the beamlines, the cameras are exposed to high amounts of radiation during the run of the experiment. Due to the number of cameras that AWAKE employs in hi-rad (high radiation) areas, it is preferable to use inexpensive replaceable cameras as opposed to radiation-resistant cameras that are often very expensive.

The Basler ace GigE(Gigabit Ethernet) cameras is a family of high-performance, high-quality, and low-cost area scan cameras designed for use in industrial and scientific applications. They offer Gigabit Ethernet connectivity, and a dedicated BNC connector for trigger signal input, which makes them well-suited for this project.[1]

A number of different cameras are used at awake to cover specific needs. To ensure that this system reliably accepts the different cameras at AWAKE, a variety of camera types are employed. The cameras vary mostly in resolution and framerate. Since the trigger signal for the cameras will not exceed 10 Hz, a varying framerate above 10 does not impact the performance. The resolution of the images does however contribute significantly to the processing speed and is hence a factor that needs to be considered when developing the system. Below is a list of the different camera types employed in this project, with naming:

Table 1: Showing the camera types employed in this project sorted by name

Name	Type	Resolution (pixels)	Frame Rate (fps)
Cam 01	acA2500-20gm	2592 x 2048	21
Cam 02	acA2500-20gm	2592 x 2048	21
Cam 03	acA2500-20gm	2592 x 2048	21
Cam 04	acA1920-40gm	1920 x 1200	42
Cam 05	acA1600-60gm	1600 x 1200	60
Cam 06	acA1600-60gm	1600 x 1200	60
Cam 07	acA1600-60gm	1600 x 1200	60
Cam 08	acA1600-60gm	1600 x 1200	60
Cam 09	acA1600-60gm	1600 x 1200	60
Cam 10	acA1600-60gm	1600 x 1200	60

3.2.5 EX-6006 PoE Injectors

Each camera is connected to the server through a Power Over Ethernet (POE) injector that supplies them with power.

The EX-6006 PoE Injectors are a type of PoE injector designed for use in industrial and commercial applications. The EX-6006 PoE Injectors provide a simple and cost-effective solution for supplying power to Ethernet-enabled devices, such as the Basler ace GigE cameras. These injectors are compact and easy to install, providing reliable power delivery over Ethernet cables.

3.3 Software Integration and Processing Algorithms

Both the PXIe-7915 FPGA board and the PXIe-1095 chassis are compatible with NI's graphical programming software LabVIEW. This simplifies the development process by utilising a large library of modular code blocks to build the system. These code blocks are called VI's and stand for Virtual Instrument, which is a self-contained software module that represents a set of programmatic tasks as a diagram much like an electrical schematic. This allows for a more intuitive way of interpreting code for people with limited backgrounds in software architecture. A VI consists of a front panel user interface and a block diagram that contains the underlying code. This allows faster implementation of the image processing algorithms, making rapid iterative testing possible. It also ensures the expandability and maintenance of the software throughout its lifecycle after development.

LabVIEW Real-Time (RT) is an add-on module for developing deterministic, time-sensitive data acquisition applications on real-time targets like the PXI chassis. LabVIEW FPGA allows for the programming of reconfigurable digital circuits without the use of traditional hardware description languages (HDLs) like VHDL or Verilog. These two add-on modules enable the whole system to be developed on a Windows-based host computer, and then deployed to the PXI target.[12, 40]

The processing algorithms applied in this system are specifically chosen for the requirements for AWAKE. They include resampling the image to reduce its size and calculating the pixel brightness distribution as a histogram. For this system, resampling is a process for reducing the resolution of an image while still retaining the information. Histograms are graphical representations of the distribution of values in a dataset. In this system, the histogram depicts the distribution of pixel brightness in the image which helps with identifying the beam behaviour.

3.4 Performance Evaluation and Error Analysis

The primary focus of the performance evaluation is finding the limitations of the FPGA board to determine its suitability for AWAKE. To evaluate the real-time image processing capabilities of the FPGA board, the time it takes to process a single frame needs to be established.

3.4.1 Data Logging and Visualization

To ensure quality measurements a timestamping system is integrated. The timestamp is created by the camera at the moment of capture and is stored in the metadata of the frame. The timestamp is then read and saved to a file. In LabVIEW, TDMS (Technical Data Management Streaming) files are often used for being able to store large amounts of data, including analogue and digital signals. This project however relies on storing data as fast as possible and a more lightweight file type was preferred. Text files (.txt) were therefore chosen as the logging solution. The timestamps for each frame are saved to the file together with the camera number, frame number, and processing time. The exact processing time is established by measuring the number of ticks or clock cycles that have elapsed before and after the image has been processed.

To parse the dataset stored in the .txt file and visualize the processing time for each camera, a Python script is created. The script processes the data and generates a scatter plot with the time on the x-axis and the processing time on the y-axis for each camera.

The script can be divided into three main parts: parsing the file, processing the data, and plotting the data.

Parsing the file: The parse_file function reads the content of the .txt file and extracts the relevant information for each camera entry. It creates a dictionary, data, with the camera number as the key and a list of tuples containing the acquisition timestamp and processing time as the value. For each entry in the .txt file, the camera name, acquisition timestamp, and processing time are extracted and added to the data dictionary.

Processing the data: The plot_data function first finds the minimum timestamp in the entire dataset. This minimum timestamp will be used to calculate the relative time for each data point in the plot. Then, for each camera in the dataset, the relative times and processing times are calculated and stored in separate lists. The relative time is the difference between the acquisition timestamp of each data point and the minimum timestamp.

Plotting the data: Using the matplotlib.pyplot library, a scatter plot is created for each camera in the dataset. The x-axis represents the relative time (in seconds and milliseconds), and the y-axis represents the processing time. Each camera is assigned a unique label, and a legend is added to the plot to identify the data points corresponding to each camera. The plot is then displayed with appropriate axis labels and a title.

Initial testing of the logging solution showed that the process of writing to the .txt file was too slow to keep up with logging from multiple cameras, and introduced errors in the measurements. The errors materialized as repeating patterns in the plot and made the data unusable. A real-time plotting function in LabVIEW was therefore implemented to visualize the processing time for each frame and the file logging functionality was abandoned.

Comparing the two solutions in Figure3 and Figure**??**, they both provide a clear visualization of the processing time for each frame and the real-time plotting solution does not remove important visual information.

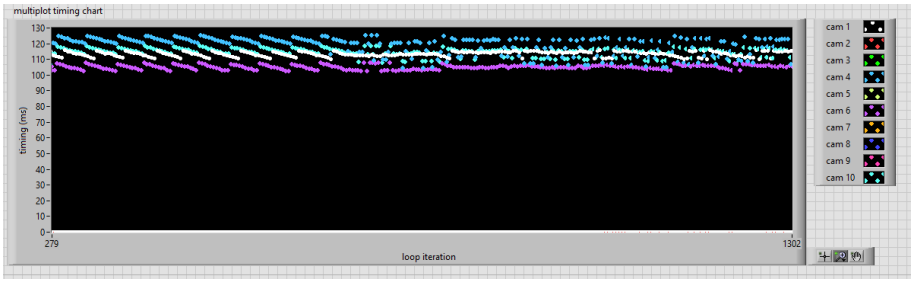

Figure 3: Screenshot of the real-time plot

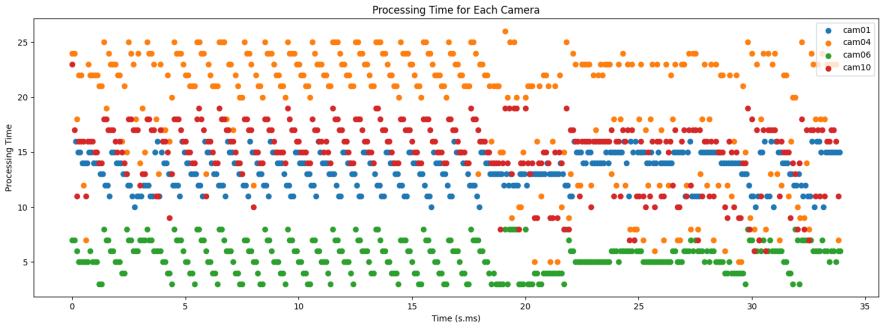

Figure 4: Screenshot of the Python script logfile parser

To evaluate the limitations of the FPGA chip, the system is scaled to accommodate additional cameras. The processing time of a single frame gives an idea of how many images the system can process sequentially between each trigger pulse. After this, the processing loop on the FPGA is multiplied to evaluate the maximum number of parallel processes. When these factors are evaluated, the full capabilities of the system can be calculated and tested.

While it's important to note that tick count timing is based on the computer's internal clock, which can vary slightly based on system load and hardware configurations, it is accurate enough to evaluate the performance of a 10 Hz image acquisition system. For higher-performance systems,

more precise timing methods, such as dedicated hardware timers or external clock signals, should be employed.

The aim of the project is to evaluate whether the proposed system is suitable for the AWAKE cam-acq system by evaluating the performance of the provided FPGA board. However, the absolute limitations of the FPGA board are left to speculation due to a limit in the number of cameras available for testing. While this could be calculated based on the tested system, this could be a potential error source.

4 System Design and Implementation

This section delves into the design and implementation of the finished system and details the choices made along the way. The objective of the system is to assess the PXIe-7915's capabilities and determine whether the integrated solution can achieve the required processing speed for AWAKE's current data acquisition needs, as well as accommodate potential future expansions. The following subsections encompass a comprehensive exploration of the system architecture, design principles, implementation, integration and testing processes, and an evaluation of the performance and scalability aspects pertaining to the enhanced system.

4.1 System Architecture

The system architecture is split into two sections detailing the hardware and software architecture of the system. The aim of this section is to get familiar with the overall structure of the system before delving into the details of the program. Since the selection of hardware is mentioned in the Methodology section, 3.2, the hardware section serves to create an understanding of the interconnections between each component and what function each part serves. Likewise, the software section details the structure of the system as well as the connections and role of each program component.

4.1.1 Hardware

The improved camera acquisition and processing system for AWAKE is designed to receive images from multiple cameras and process them in real-time using a vision-specific FPGA board. The overall hardware architecture of the system is shown in Figure 5. The hardware is composed of five key components:

- Basler ace GigE cameras:
  - acA2500-20gm
  - acA1920-40gm
  - acA1600-60gm
- PoE Injectors
  - EX-6006
- NI VirtualBench VB-8012 Signal Generator
- PXIe-7915 FPGA Board
  - Kintex UltraScale +
- PXIe1095 Chassis

The system components are connected to each other via ethernet or coaxial cabling, or a direct PCI bus connection for the modules inside the PXIe target. The cameras connect to the PXIe with Gigabit ethernet cabling, ensuring a versatile connection and the ability to connect through a PoE injector. The PoE injector applies a voltage over a set of dedicated strands in the ethernet cable for the output to the cameras. The cameras also feature a BNC connector for a trigger pulse signal, which in this project is provided by the VirtualBench. Inside the PXIe-1095 are four GbE expansion cards that receive the signal from the cameras, as well as the FPGA board.

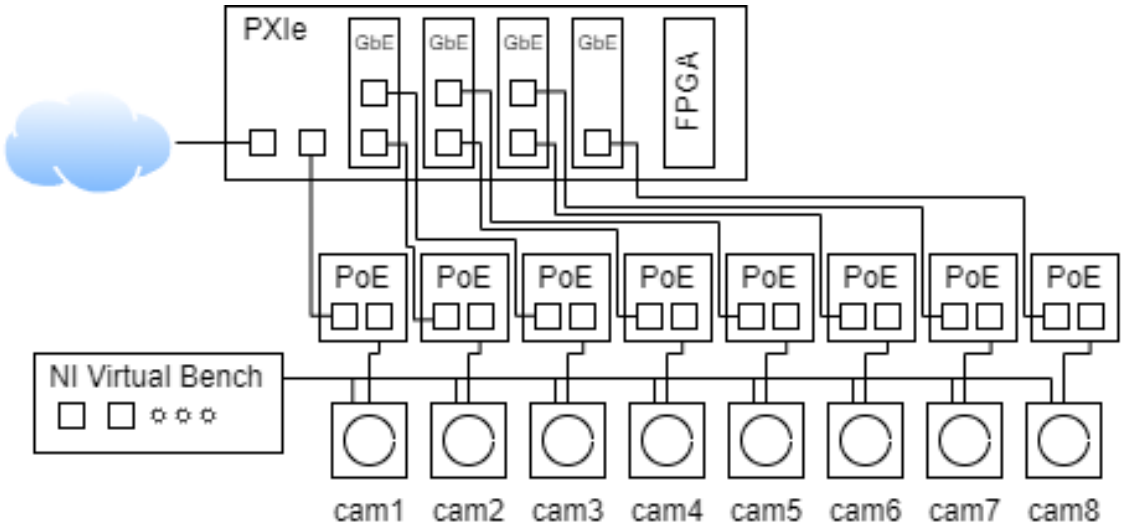

Figure 5: Diagram showing the AWAKE camera acquisition system hardware architecture

4.1.2 Software

The software of the camera acquisition system can be broken down into three subsystems: The RT-target code, which ties the system together and communicates with the hardware; the FPGA's processing code; and the communication with a remote server that is responsible for storing the data and making it accessible for the rest of the CERN network. However, the make of a fully deployable acquisition system is not relevant to the scope of this research and some parts are hence excluded from this thesis. This section focuses instead on the testbench setup and replaces the CMW (CERN Middleware) communication VI with a logger VI. The logger is responsible for logging the data to a file, as well as graphing the processing time for each camera on the front panel.

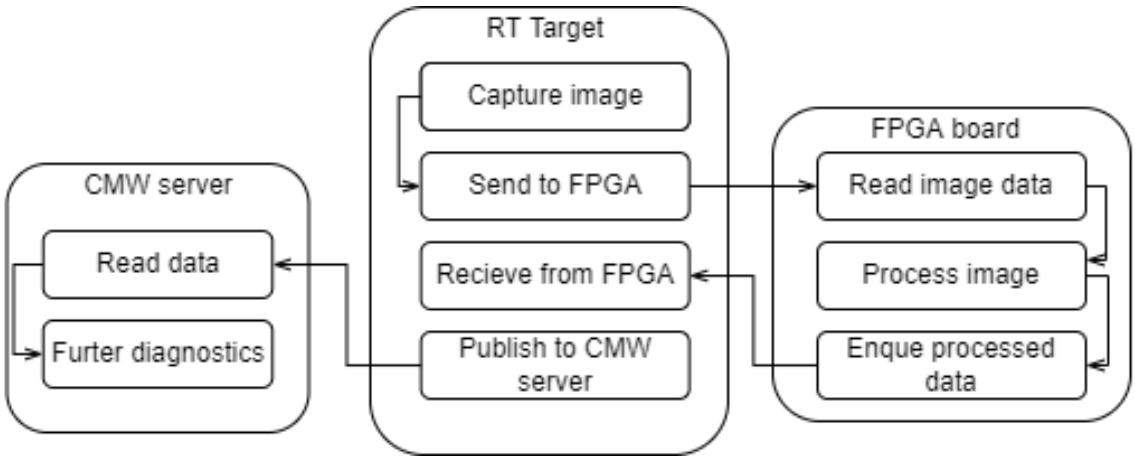

Figure 6: Diagram showing the AWAKE camera acquisition system software architecture

The software architecture diagram above shows the main actions of the software that continuously loop. In addition to these loops, the software consists of an initiation and termination -phase. The RT-target code consists of a main VI that contains the rest of the code blocks shown below, in Figure7. Firstly, this VI is responsible for starting the initiation and the other loops. It then goes idle by repeating a loop that checks the status of a stop button. Pressing the stop button sends a notification to the rest of the subsystems telling them to terminate. Lastly, a termination sequence frees all resources by deleting any references or data contained in memory.

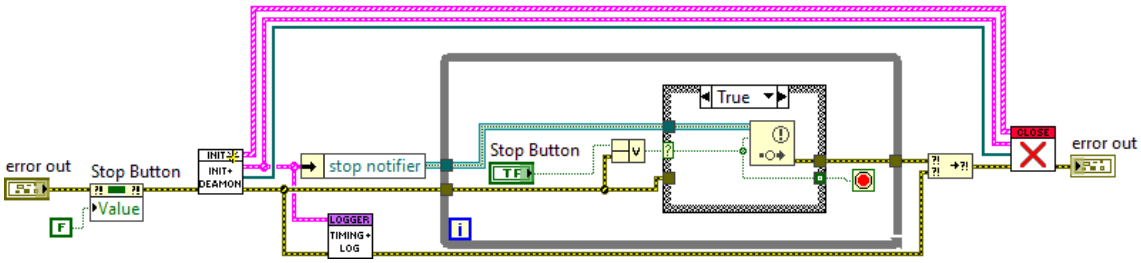

Figure 7: Block diagram of the rt_main.vi

4.2 Detailed System Design

This section is provided to discuss the software in more detail and highlight the sub-VIs that compose the system. It tries to explain the key components, their role in the system, their interactions, and the design approach. The code is structured as a tree with each code block containing another set of code blocks. Each subsection features images of the piece of code discussed with the VI icon to make it easy to place the code block within the system. Many of the code blocks feature case structures that do not show in the images, these are therefore provided in the appendix.

Firstly, the initiation phase is explained. Followed by detailing the different processes that continually run during the operation on the RT-target and the FPGA. Lastly, a rundown of the termination procedure is added.

4.2.1 Initiation

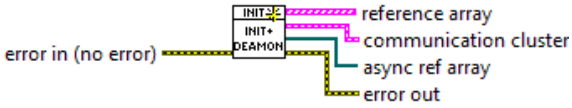

Figure 8: Image showing the init.vi icon

The initiation procedure for the system is a major part and consists of a set of sequential initiation operations followed by deploying the acquisition loop for each camera. Following the code from left to wright, the code starts by establishing a communication reference, allowing running loops to share information with each other.

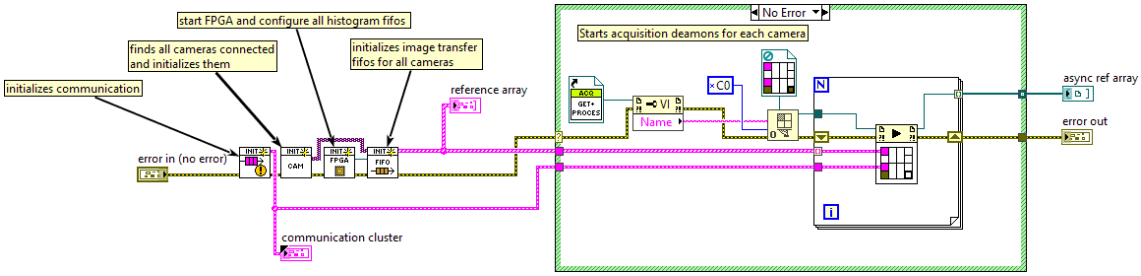

Figure 9: Image showing the init.vi block diagram

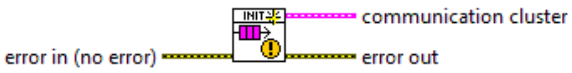

Figure 10: Image showing the init_communication.vi icon

To communicate between running loops, the loops must both have access to a shared piece of memory. The communication methods initiated here are a notifier and two queues. The notifier is a single datatype memory allocation that signals if the stop button has been pressed. The queues are a system where data can be stored from multiple sources, and accessed by a dequeuer. This offers a tidy way of processing tasks in the order they were created.

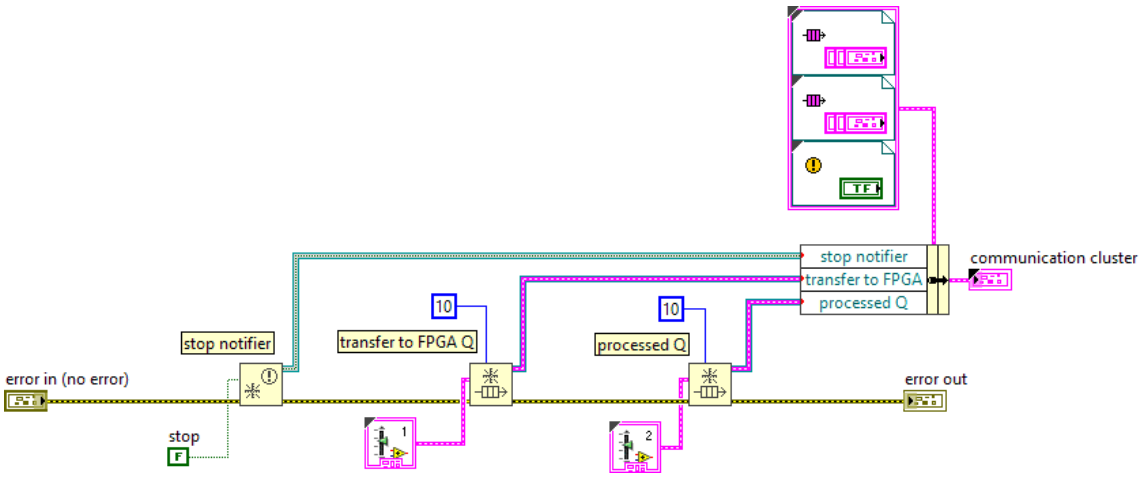

Figure 11: Image showing the init_communication.vi block diagram

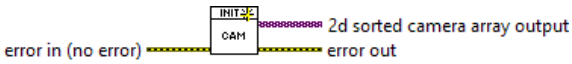

Figure 12: Image showing the init_camera.vi icon

The cameras are initiated in sequence in the order they are connected. The initiation applies a set of settings for the camera and starts listening to the output. When all the cameras are initiated, an array of camera elements is returned from the for-loop. The array then goes through a custom sorting algorithm that places cameras of equal resolution together in groups. The group size is dependent on the number of cameras that can be processed by a single FPGA loop and is therefore determined by the processing speed for a single frame.

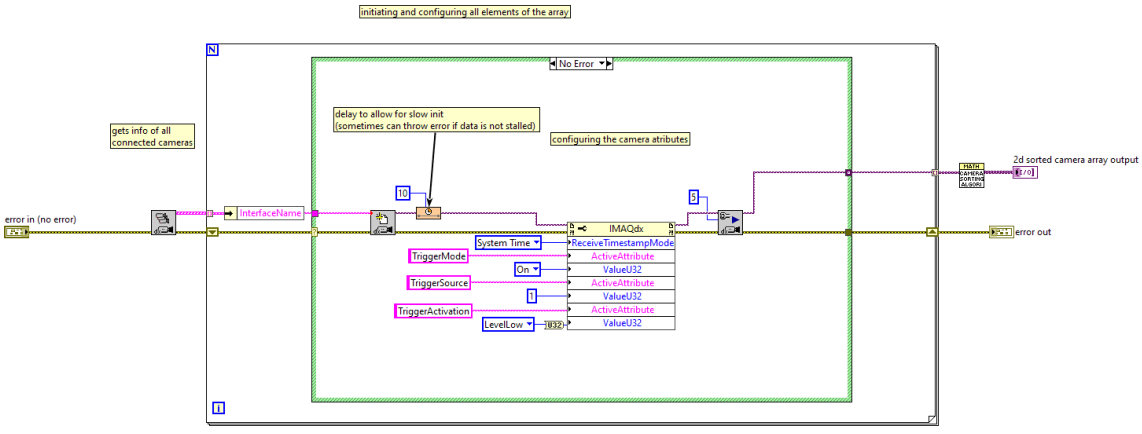

Figure 13: Image showing the init_camera.vi block diagram

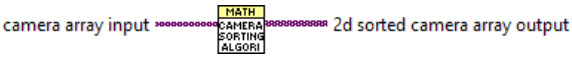

Figure 14: Image showing the camera_sorting.vi icon

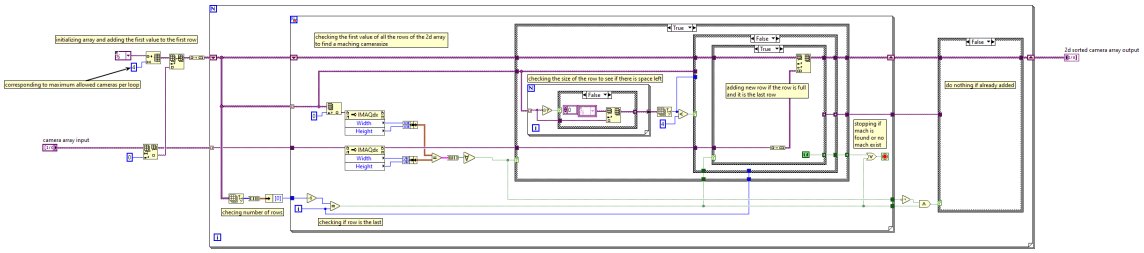

Figure 15: Image showing the camera_sorting.vi block diagram

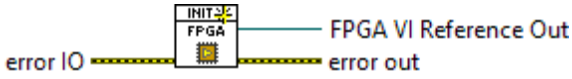

Figure 16: Image showing the init_FPGA.vi icon

The FPGA code is applied to the FPGA and the device is started. A set of FIFO (First In First Out) queues, referred to as just FIFOs, are then established for receiving the histogram data from the FGPA.

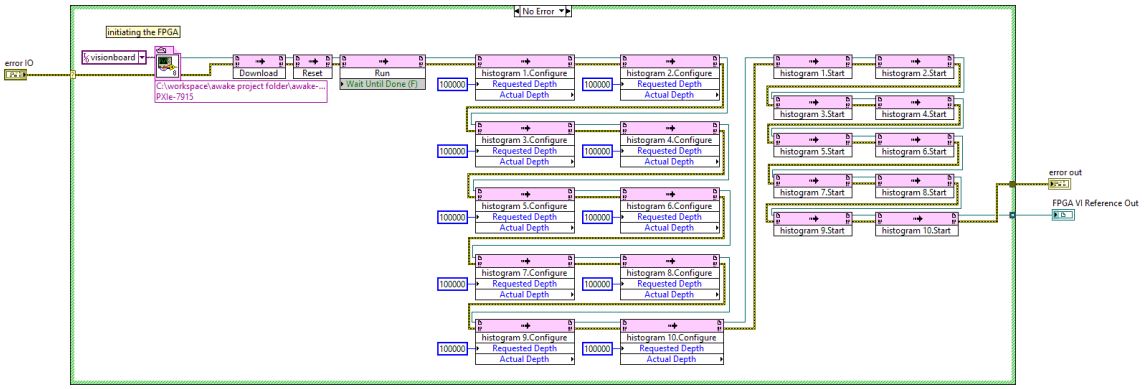

Figure 17: Image showing the init_FPGA.vi block diagram

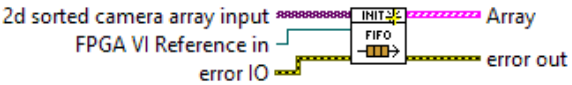

Figure 18: Image showing the init_configure_FIFO.vi icon

A set of FIFOs are created and configured for each camera resolution group. A group of four cameras share a pair of FIFOs, one for queuing images TO the FPGA and one for dequeuing the processed image FROM the FPGA. The FIFOs are shared between a group because the configuration sets the size of the FIFOs based on the camera resolution. Since one of the processing operations on the FPGA is resampling, the receiving FIFO must have the size of the resampled image. This is set by dividing the horizontal and vertical length by the resampling factor. A set of memory locations are allocated for each camera for holding the image before and after it passes through the FPGA. The data from the cameras are written to the FIFO in four parallel processes, meaning the pixel data could get mixed up. To avoid this a queuing system is employed to block the FIFO resource while one camera is writing to it. The references are then packaged together with each camera in an array of all the cameras.

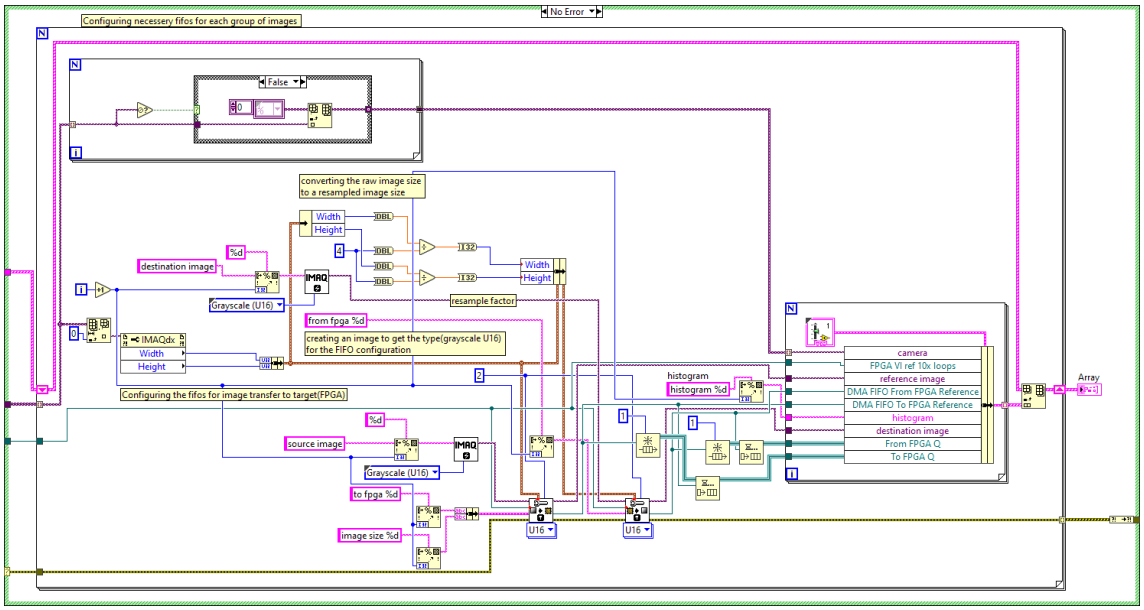

Figure 19: Image showing the init_configure_FIFO.vi block diagram

4.2.2 Acquisition daemon

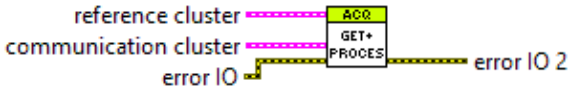

Figure 20: Image showing the acquisition_deamon.vi icon

A parallel image acquisition loop is created for each camera element and runs continuously in the background of the program. The "daemons" stores the image from the camera in memory and waits for the FIFO to be available. When the FIFO is available, the loop takes it from the queue making it unavailable for the other cameras in the group. The image is then queued for the FPGA processing. A few milliseconds after, a resampled image and a histogram are received. The FIFO is then made available again for the next camera in the group. The resampled image and histogram are queued for the logger.vi together with measurements of the processing time.

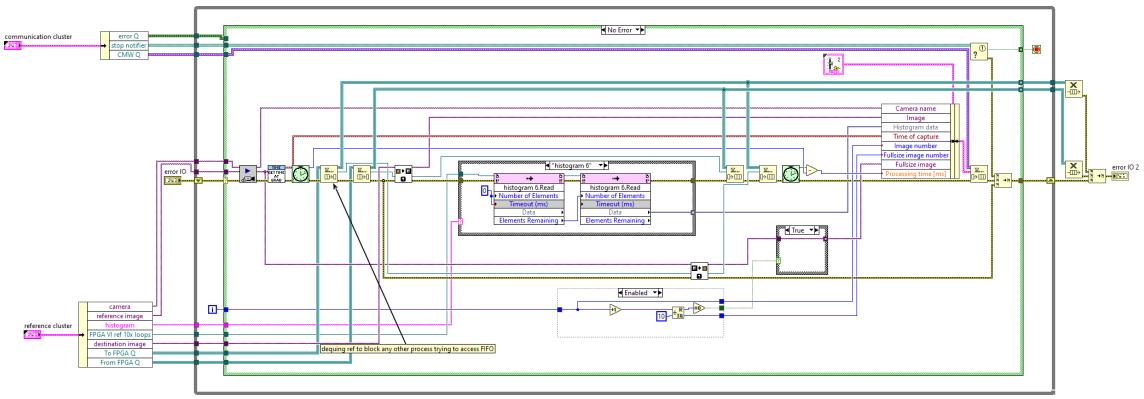

Figure 21: Image showing the acquisition_daemon.vi block diagram

4.2.3 FPGA code

The block diagram representation of the FPGAs code consists of a number of loops running in parallel. This section focuses on the code within each loop. Displayed below is the block diagram of the initial test code, it consists of a single loop and is capable of processing the images from four cameras sequentially between each trigger pulse. Each parallel process of the acquisition daemon is paired with one loop on the FPGA. It pipelines the processing of the image with communication signals between each task to synchronize the data flow. The synchronization signals, shown in green in Figure22 are a set of flags that tell the previous process when it can receive new data. It performs two processes, resampling the image, and creating a histogram of the brightness level distribution of the dataset.

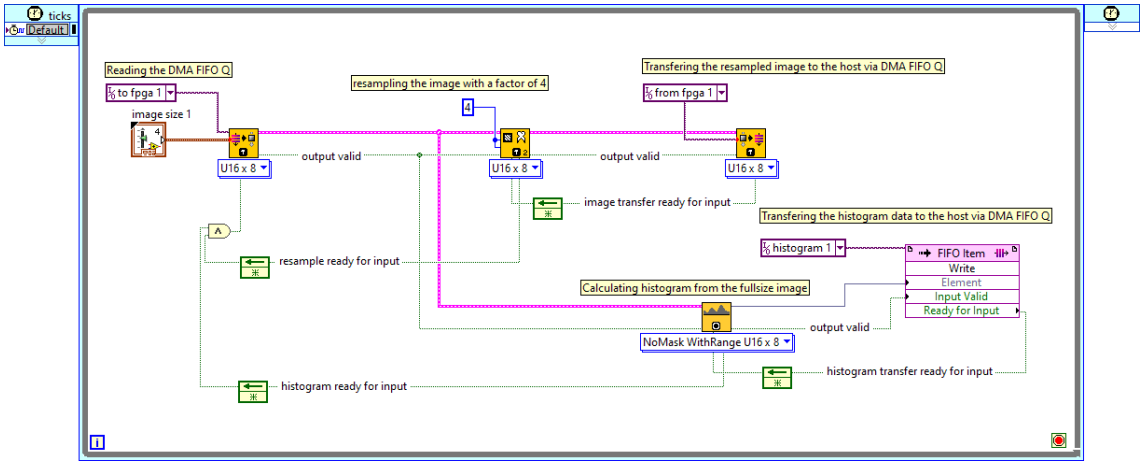

Figure 22: Image showing the FPGA single loop block diagram code

4.2.4 Logger

Figure 23: Image showing the logger.vi icon

The logger.vi is a set of loops responsible for displaying the processing time and image stream; logging the image stream, with a timestamp, camera name, frame count, etc.; and logging system errors that occur during the runtime of the system.

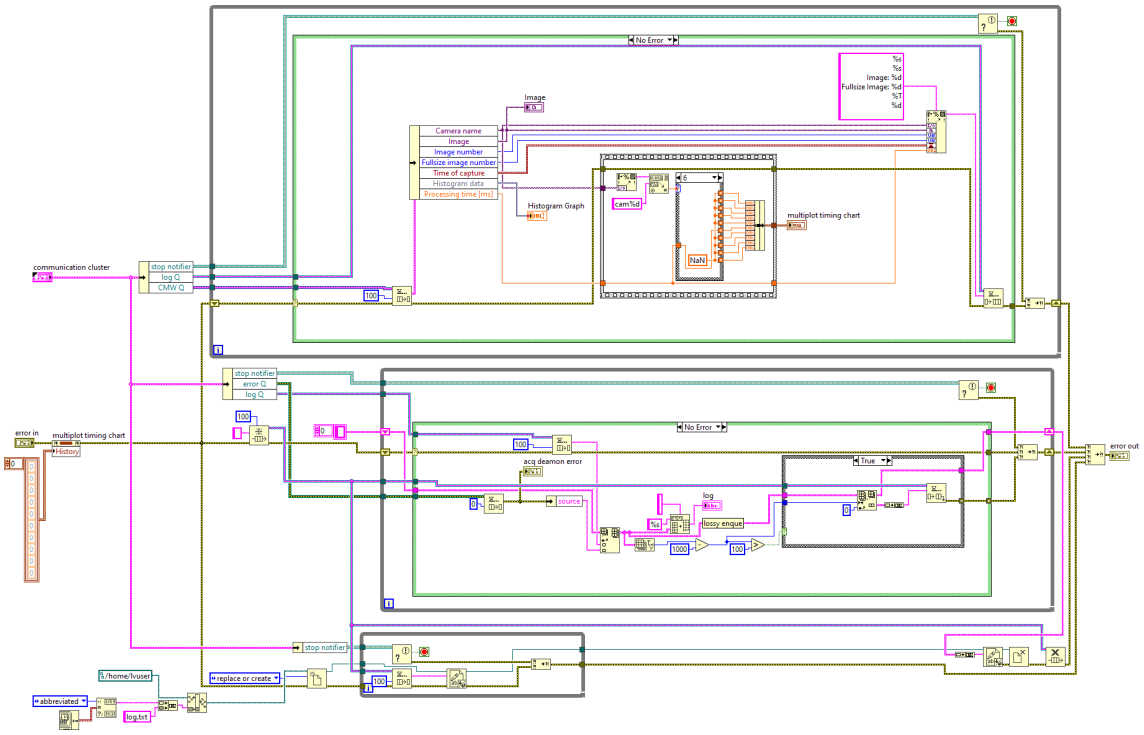

Figure 24: Image showing the logger.vi block diagram

4.2.5 Close

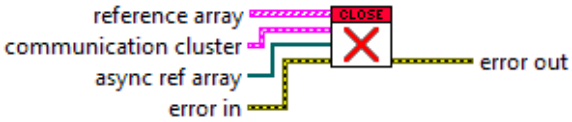

Figure 25: Image showing the close.vi icon

This VI runs at the end of the program after the other processes are terminated. The close.vi is responsible for freeing up memory at the end of the program runtime. This means closing all the references and deleting queue elements that remain in the queue.

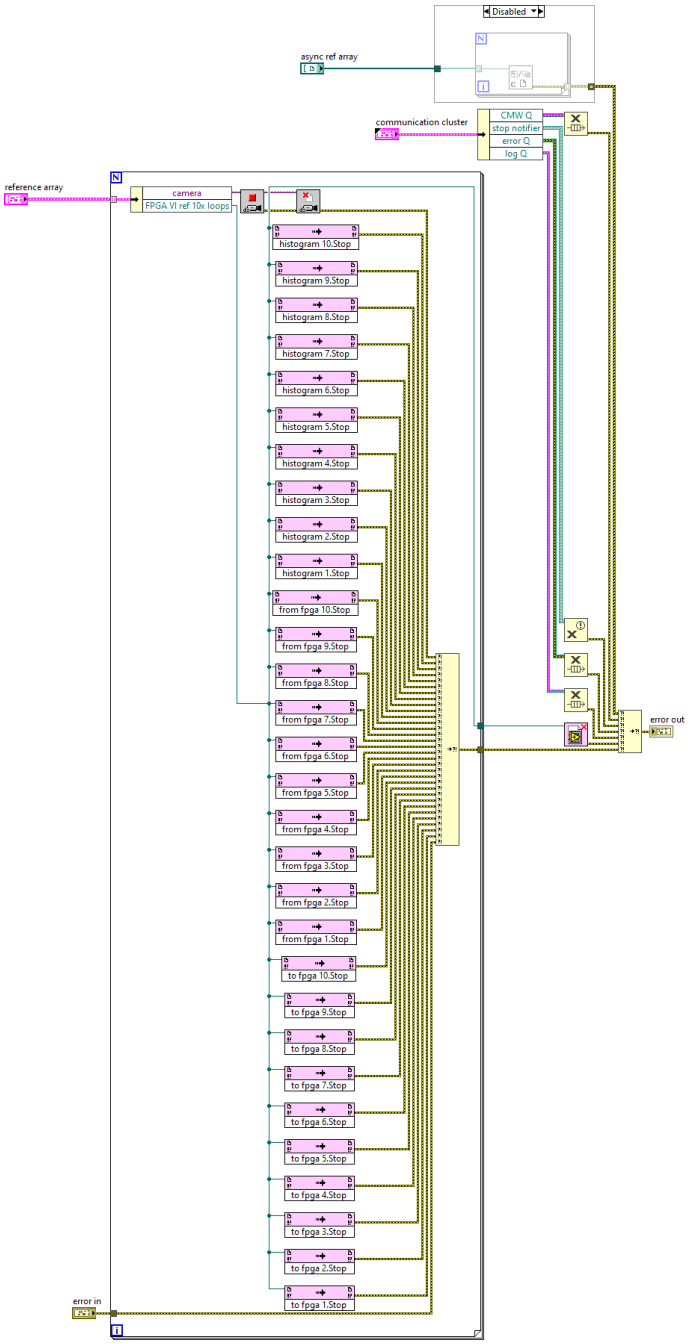

Figure 26: Image showing the close.vi block diagram

4.3 Design Considerations and Choices

The development process is a crucial part of any project, where trade-offs are made that affect the final product. This section delves into the design process and the reasoning behind certain choices made during the development of a camera system. Additionally, it delves into the workarounds that were implemented to overcome the challenges encountered during the development of the camera system.To ensure the functionality of the system certain challenges had to be worked around. Through this exploration, we aim to provide insights into the design process and the importance of being adaptable when working on complex engineering projects.

4.3.1 Bandwidth Limitations for the PoE network switch

The way the cameras are connected to the PXIe-1095 chassis at AWAKE is through a PoE network switch instead of a standalone PoE injector. This allows the network switch to connect to the PXIe through a single ethernet cable with the downside of it increasing the bandwidth required for the signal. During early testing of the system, we found that the bandwidth of the PoE switch available at the MTA sections computer-laboratory proved to be too low, resulting in partial data loss. This manifested itself as black stripes in the images where the data had been lost. Below is a screen snippet of the received image from the PoE switch.

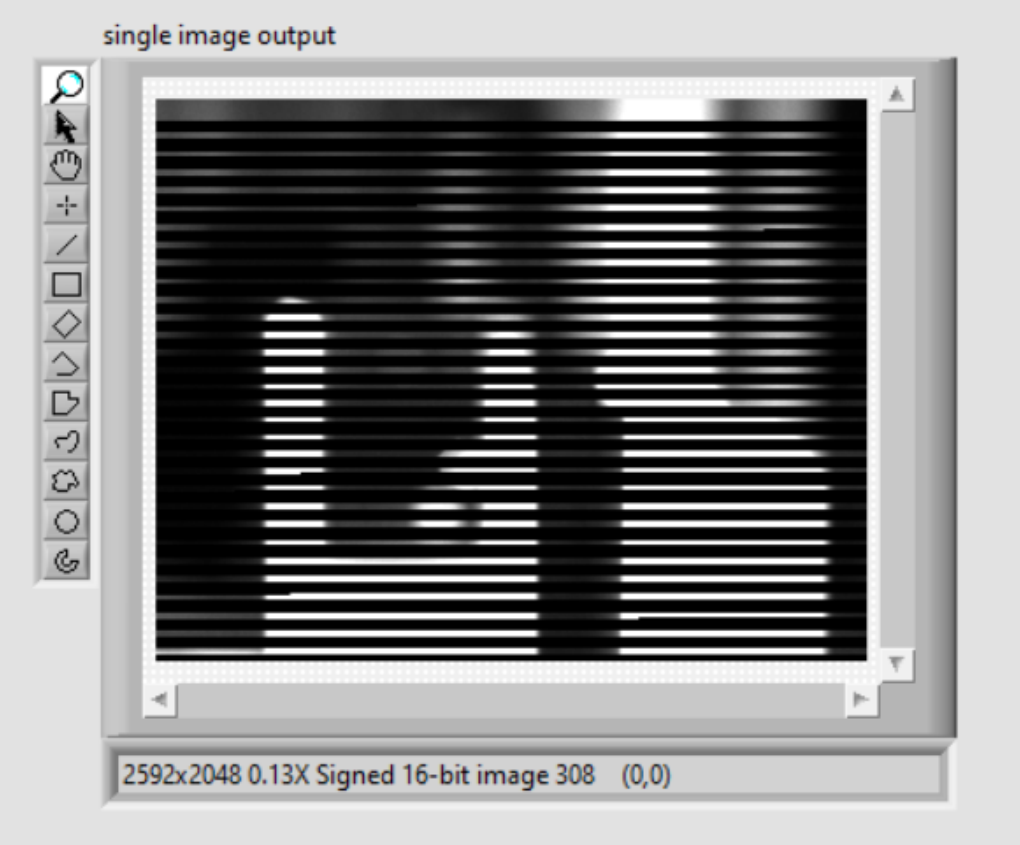

Figure 27: Partial data loss during image transfer resulting in black stripes over image

The solution to this was to bypass the switch by connecting the cameras directly to the PXI. For this, four gigabit-ethernet expansion modules were installed into the PXI, providing a total of nine GbE ports. This configuration supports up to eight cameras, with one port reserved for internet access. This solution eliminated the error but put a limit on the number of cameras that could be tested at one time. Bypassing the PoE switch also made it necessary to employ separate PoE devices for supplying power to the cameras.

Figure 28: Photograph depicting the installed GbE expansion cards with 6 cameras connected via ethernet cables

4.3.2 FPGA task synchronization error

During the development of the FPGA code, a lot of new concepts had to be employed to ensure efficient image processing. One way of increasing the efficiency of code on the FPGA is to pipeline the processing. LabVIEW FPGA already employs this principle in many of their premade code blocks. However, the different code blocks are not synchronized with each other. Before realising this, a lot of time was spent debugging an issue with scrambled data from the FPGA, shown in Figure 29. The issue was resolved by synchronizing the different processes with a series of communication wires that communicate whether a process is ready to receive or transmit data.

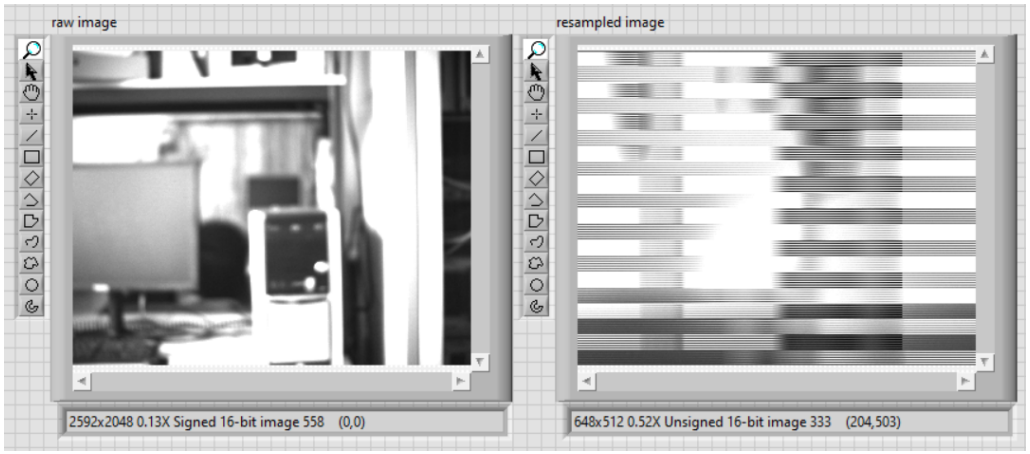

Figure 29: Screenshot showing a comparison of the transmitted and received image from the FPGA

—

5 Results

In this section, we present the findings of our study on FPGA-accelerated real-time data processing. The study focuses on whether the proposed solution for the AWAKE camera acquisition system is able to provide sufficiently fast processing to reliably process images from all the cameras at 9.97 Hz. And if the system can accommodate additional cameras in the future. Firstly, we provide our preliminary findings, followed by our findings on the processing speed for each camera. Then we look at the expandability of our proposed system. Finally, we analyse the findings.

5.1 Preliminary testing

A series of preliminary tests had to be performed to ensure the reliability of the system as well as to find ways of improving the system. Firstly, we needed to determine whether the FPGA-accelerated real-time image processing task can be done reliably in under 100 milliseconds. This was tested by measuring the processing time of every frame for each camera size group. Unsurprisingly we found a clear correlation between the image size and processing time, with the largest image being the slowest to process. Below is a selection of the test results from each size group of cameras:

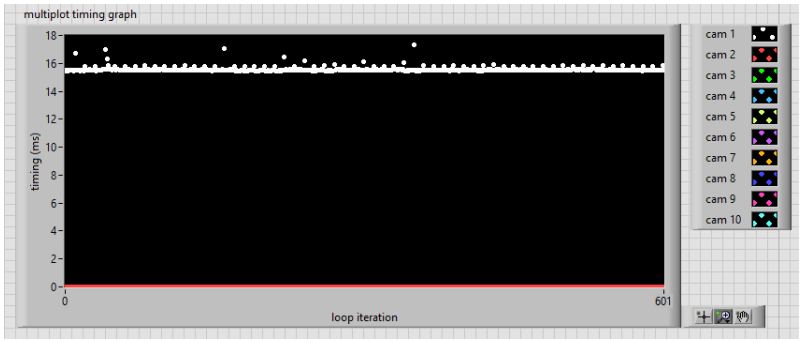

Figure 30: Image showing the processing time in [ms] for camera 1 acA2500-20gm Resolution: 2592 x 2048, over a period of 1 minute

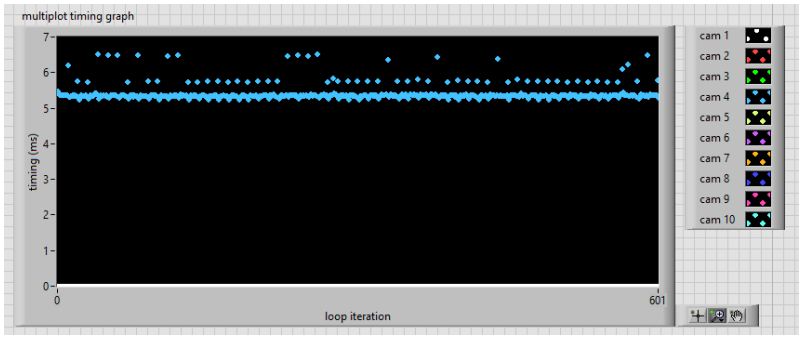

Figure 31: Image showing the processing time in [ms] for camera 4 acA1920-40gm Resolution: 1920 x 1200, over a period of 1 minute

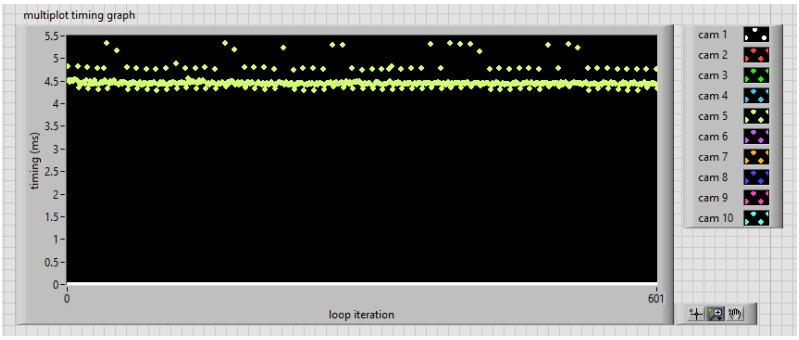

Figure 32: Image showing the processing time in [ms] for camera 5 acA1600-60gm Resolution: 1600 x 1200, over a period of 1 minute

The processing time for each frame determines the number of frames that a single FPGA loop can process within the 100 ms window of our 10 Hz signal. The 18 ms processing time for camera 1 suggested that each loop could handle frames from five cameras at the same time. However, outliers of frames that take longer than 18 ms to process made the system lose frames which made the system unstable. This instability led to unreasonably unreliable processing times and eventually resulted in a crash, shown in Figure 33. A more modest group size of four was therefore selected and used for the rest of the testing.

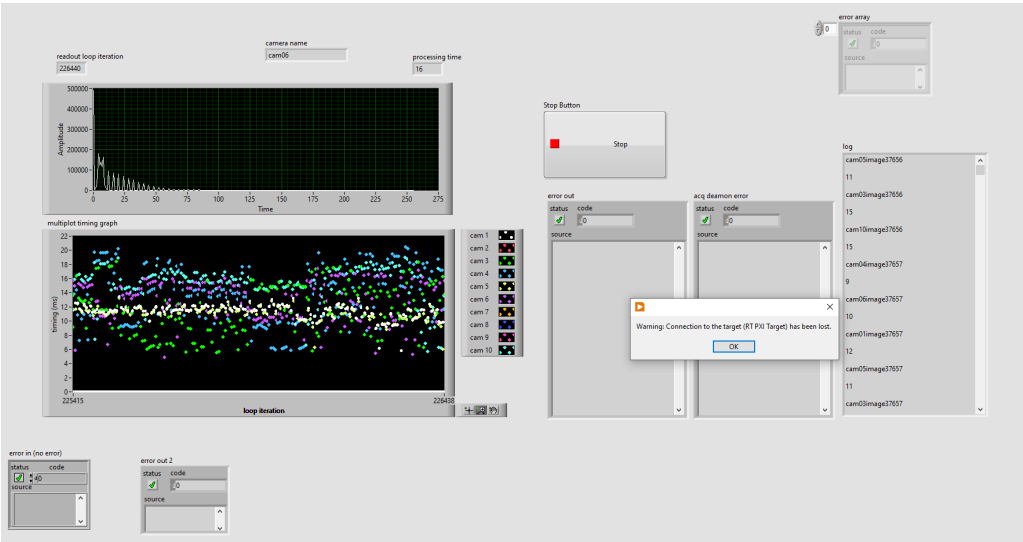

Figure 33: Screenshot showing a system crash error message and an unreasonable variation in processing time

5.2 Processing Speeds and Throughput

To effectively address the project's problem definition, it was crucial to evaluate the system's performance under increased workloads, achieved by connecting multiple cameras simultaneously. Due to the limited Ethernet connectivity of the PXIe, we were only able to test the system with a maximum of eight cameras at a time. This more than what AWAKE currently employs, and is thus sufficient for determining whether the system fulfils the requirements for AWAKE's application. However, this meant that we could not conduct a thorough test of the outer limitations of the FPGA. We, therefore, have to result to educated speculation about the system's stability at higher loads.

Our tests conducted with eight cameras demonstrated that while the single-frame processing speed increased for all cameras, the system was capable of reliably handling the additional traffic gener-

ated.

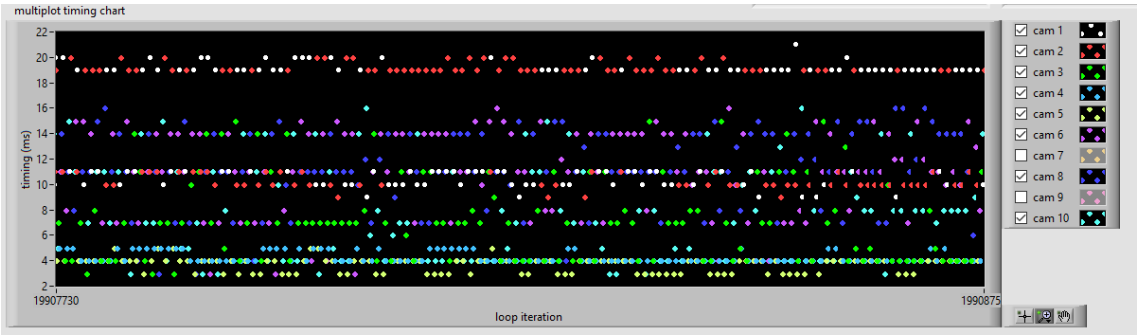

Figure 34: Image showing the processing time for each camera in [ms]

In addition to the processing time evaluation, we also assessed the system's resource consumption during operation in order to determine the performance of the RT-target side. Figure 35 illustrates the resource consumption of the RT-target at runtime.

System Resources

Total Physical Memory	7.72 GB			
Free Physical Memory	5.59 GB			
Total Virtual Memory	7.64 GB			
Free Virtual Memory	5.70 GB			
Primary Disk Capacity	468 GB			
Primary Disk Free Space	466 GB			
CPU Model	Intel(R) Xeon(R) CPU E3-1515M v5 @ 2.80GHz			
CPU Cores	4			
CPU Logical Processors	4			
CPU Total Load	67%	66%	67%	65%
CPU Interrupt Load	17%	15%	16%	14%

Figure 35: Image showing the resource consumption of the RT-target at runtime

Analyzing the processing time and resource consumption data allowed us to gain insights into the system's efficiency and throughput. It also helped identify potential bottlenecks and areas for improvement on the RT-target side, such as optimizing resource utilization or finding memory leaks.

5.3 Scalability

In order to analyze the scalability of the system, we carried out a systematic assessment of the FPGA's capacity to handle increasing numbers of loops. This was achieved by compiling progressively larger FPGA code until the system could no longer be compiled. The maximum number of loops that the FPGA could accommodate was found to be 10. Given that each FPGA loop can support four high-resolution cameras, this means that our system can theoretically handle up to 40 high-resolution cameras simultaneously.

Table 2, illustrates the device utilization for the FPGA code employing 10 loops. It can be observed that the total slices used is 91.4% of the available resources. This indicates that it may be possible to add an 11^{th} loop by optimizing slice utilization. However, this enhancement was not realized within the scope of this project.

Table 2: Showing the device utilization for the FPGA code at 10 loops

Device Utilization	Used	Total	Percent
Total Slices	37878	41460	91.4
Slice Registers	170381	663360	25.7
Slice LUTs	203950	331680	61.5
Block RAMs	444	1080	41.1
DSP48s	160	2760	5.8

The potential scalability of the system extends even further when considering the fact that most cameras employed by AWAKE are of lower resolution than the high-resolution cameras used in our tests. Consequently, the FPGA loops could process more than four frames within the 100 ms timeframe, allowing for a greater number of lower-resolution cameras to be supported.

In addition to the FPGA's inherent scalability, the PXIe-1095 chassis also provides room for additional FPGA modules, which further expands the system's capacity. By fully utilizing the available space within the PXIe chassis, it is theoretically possible to scale the system to support over 100 cameras. This opens up the possibility of accommodating larger and more complex experimental setups, enhancing the overall flexibility and adaptability of the system.

6 Analysis and Discussion

This section discusses the findings in light of the methodology and research question. The findings of our study suggest that the proposed FPGA-accelerated real-time data processing system fits the requirements for AWAKEs camera acquisition system and that it is scalable enough to handle additional requirements in the future. The system is able to process images from eight cameras simultaneously at a rate that meets the required 9.97 Hz, and the system can accommodate a theoretical limit of 40 high-resolution cameras without system modifications.

6.1 Importance of Findings

The findings contribute to the understanding of FPGA-accelerated real-time data processing, especially in the context of National Instruments hardware and LabVIEW FPGA programming. The project incorporates new methods of implementing the system and has contributed to further developing the LabVIEW FPGA code library. The system provides empirical evidence of the system's processing capabilities and scalability, which is valuable for future research and development of similar systems. This also proves valuable for the continued development of the AWAKE cam-acq system and could aid in future image acquisition systems for CERN.

Demonstrating the Feasibility of FPGA-accelerated Real-time Data Processing: The successful implementation of FPGA-accelerated real-time data processing for the AWAKE camera acquisition system demonstrates the feasibility of using FPGA technology in high-performance scientific applications. This opens up possibilities for further exploration of FPGA-based solutions in other scientific fields, such as medical imaging, astrophysics, and other high-energy particle physics experiments.

Advancing the LabVIEW FPGA Code Library: The development of this system has led to the refinement of the LabVIEW FPGA code module library by highlighting an issue with the IP architecture for one of the processing VIs that was used during the project. The VI did not support x1 pixel buses on the Kintex UltraScale+ FPGA targets. While the VI did not change in time to be included in this system, the improved module can be utilized in future projects, potentially saving development time.

Enhancing System Scalability: The scalability of the proposed system ensures that it can adapt to future requirements without significant modifications. As the number of cameras and resolution requirements increase, the FPGA-accelerated data processing system can be expanded accordingly, offering a cost-effective and efficient solution for evolving experimental setups.

Providing a Benchmark for Future Research: The empirical data gathered from the performance analysis of the FPGA-accelerated real-time data processing system offers a benchmark for future research in this area. Researchers and engineers can use this information to optimize their FPGA designs and compare the performance of their solutions against the established benchmark.

Informing the Development of Future Image Acquisition Systems: The successful implementation of the FPGA-accelerated real-time data processing system for the AWAKE camera acquisition system provides valuable insights that can be applied to the development of similar systems in the future. By understanding the challenges and solutions presented in this research, engineers and researchers can make informed decisions when designing image acquisition systems for other large-scale scientific experiments.

6.2 Exploring Potential Causes of Varying Processing Time

Looking at the multi-plot for the processing timing in Figure34. A consistent pattern in processing time for each camera size group is apparent. However, the timing for each individual frame varies widely in two distinct ways. Focusing on cam 01, white and cam 02, red, the timing is consistent around the 10 ms and 20 ms mark, with a deviation of ± 1 ms.

System performance is primarily determined by throughput and maximum single-frame processing time. The variation in timing between individual frames does not significantly affect the overall limitations of the system. As a result, the focus has shifted away from determining the root cause of this variation, and no definitive solution is currently available. However, it is still essential to explore potential causes for these variations and identify areas for future improvement. Understanding the possible reasons for these variations can help improve the system's overall performance and efficiency for extensions to the system.

Some potential causes for varying processing time include:

Inefficient resource use: The physical processing layout of the FPGA chip implemented in the system might not be optimized for the FPGA architecture, leading to inefficient use of available resources. An unoptimized layout results in the system demanding more resources which could make it run slower than expected, or crash under heavy loads. Researching more efficient algorithms for netlist creation for the specific FPGA architecture could address this issue.

System load: The FPGA system may underperform under varying loads, which could be caused by fluctuations in the input data or other system tasks. This could be addressed by implementing load-balancing mechanisms, buffering input data, or prioritizing critical tasks to ensure consistent performance.

Thermal throttling: If the FPGA chip overheats due to inefficient resource usage or inadequate cooling, it may experience thermal throttling. This results in reduced performance and can cause the system to slow down or crash. To mitigate this issue, sufficient cooling or down-clocking methods could be implemented to ensure the chip's temperature remains within acceptable limits.

Clock frequency and timing violations: The system's clock frequency might be too low, or the design may have timing violations that cause the system to operate incorrectly. Timing violations can lead to unpredictable behaviour or system crash. While timing violations are checked during the compilation of the FPGA code, varying system load could cause the system to fall behind, leading to timing violations.

Data Dependencies: The processing algorithm might have inherent data dependencies that cause the execution time to vary based on the input data. Certain data patterns or input characteristics might trigger specific parts of the algorithm that take longer to process. Consequently, the processing time for each frame can vary depending on the content of the frame.

Memory Bandwidth Limitations: Inefficient memory access patterns or memory bottlenecks can result in varying processing times. If the algorithm relies heavily on memory access, the available memory bandwidth might not be sufficient to sustain the required throughput. This can lead to stalling of the system as it waits for memory accesses to complete, resulting in longer processing times for some frames.

6.3 Improvements for Future Development

While the FPGA-accelerated real-time data processing system is well within the requirements for AWAKE, some improvements could be done to the current system to improve its capabilities. Based on our findings, the following set of improvements could be implemented:

Netlist Optimization: The place-and-route stage of the compilation for the FPGA code on LabVIEW FPGA takes a sort of brute-force approach to determine the netlist for the physical layout of the FPGA. Inefficiencies in the physical layout result in inefficient resource utilization and could impact system performance. Research into optimizing the netlist creation could prove significant for the performance of the system. Not only would this provide faster processing with less energy loss, but it would also be less space-demanding on the physical chip, allowing for additional processing loops on the FPGA.

Implementation of Load Balancing and Buffering: Currently, the system relies on a fixed number of cameras per FPGA loop to ensure reliable processing. However, this approach can be limiting as it does not account for variations in camera resolution or processing times. Load balancing can be implemented to distribute the processing workload more evenly among the FPGA loops, ensuring that no loop is overloaded while others are idle. Buffering can also be implemented to ensure that input data is buffered, and ready for processing by the FPGA loop as soon as it becomes available. These improvements can increase the system's overall throughput and reduce the likelihood of system crashes due to overloaded or timing violations. Additionally, load balancing and buffering can be used to improve the system's scalability, allowing it to accommodate more cameras or handle higher-resolution images without significant modifications.

Exploration of Dynamic Voltage and Frequency Scaling (DVFS): Implementing DVFS can be a significant improvement to the current FPGA-accelerated system. By adjusting the operating voltage and frequency based on the system load, the system can achieve better power efficiency and thermal management. DVFS can also improve system performance by allowing the FPGA to operate at a higher frequency when the system load is low and reducing the frequency when the system load is high. This can lead to faster processing times for some frames and more consistent processing times overall, addressing the issue of varying processing times discussed earlier. However, implementing DVFS requires careful consideration of the FPGA architecture, the processing algorithm, and the system's power management features. Therefore, a thorough analysis and testing must be performed to determine the optimal power settings and frequency scaling factors for the system.

Installing Additional Hardware: Another potential improvement to the FPGA-accelerated real-time data processing system for the AWAKE camera acquisition system is to install additional hardware into the PXIe chassis. The current system utilizes one PXIe-1095 chassis, which supports up to 18 expansion modules. Installing additional FPGA modules could increase the number of cameras that the system can accommodate. This could be particularly useful for larger experimental setups that require more cameras, or for experiments that involve higher resolution cameras. Additionally, installing additional hardware could also provide redundancy and fault tolerance for the system, reducing the risk of data loss or system crashes. However, implementing this improvement would require additional investment in hardware, as well as modifications to the RT-target code to support the increased number of cameras and FPGA modules.

Investigation of Alternative FPGA Platforms: The effectiveness of this system is by no means proof of the superiority of LabVIEW FPGA or NIs PXIe-7915 FPGA board. Future research could therefore investigate the effectiveness of alternative FPGA platforms in implementing real-time data processing systems. This can lead to the development of more cost-effective and efficient solutions. Not relying on National Instruments can even provide open-source code for achieving FPGA-accelerated image processing.

7 Conclusion

In conclusion, this study has presented the findings of our investigation into the feasibility of FPGA-accelerated real-time data processing for the AWAKE camera acquisition system. The results show that the proposed solution meets the requirements of the AWAKE system and is scalable enough to handle additional cameras and future requirements. Moreover, this study contributes to the advancement of FPGA-based solutions in high-performance scientific applications and the development of the LabVIEW FPGA code library.

The preliminary testing of the system revealed the correlation between processing time and image size, which set the sequentially processed group size limitations. Moreover, the instability caused by outlier frames that take longer than 18 ms to process showed that a group size of four cameras per FPGA loop had to be used for a stable system and was hence used for the rest of the testing. The test results showed that the system can handle an increased workload by running the system with up to eight cameras at a time, with reliable processing speed for each camera. The system's scalability was also established by determining the maximum number of loops the FPGA can accommodate by compiling the code with a total of 10 loops before it reached its limit. The FPGA-accelerated data processing system can process four high-resolution frames per FPGA loop within the 100 ms timeframe, allowing for a theoretical limit of 40 high-resolution cameras to be connected at a time. Additional improvements to the system could explore different ways of upscaling its capabilities as well as providing a more stable system.

The findings of this study have important implications for the scientific community, demonstrating the feasibility and scalability of FPGA-accelerated real-time data processing. The study provides empirical evidence of the system's processing capabilities and offers a benchmark for future research in this area. The study also advances the LabVIEW FPGA code library and offers valuable insights that can be applied to the development of similar systems in the future.

While exploring potential causes of varying processing times, several possible areas for future development and improvement of the proposed system were identified. Optimizing the code would improve the overall system performance, decrease load times during startup and increase system reliability. Implementing load-balancing mechanisms would make better use of the system's resources and could avoid issues like timing violations. Implementing dynamic voltage and frequency scaling to manage the system load could help in thermal management to ensure that the chip's temperature remains within acceptable limits, improving the system's reliability and longevity. Additionally, future hardware expansion could improve the system's capacity and processing speed, allowing the system to meet timing requirements well beyond the need for awake.

Overall, this study has presented an innovative solution for FPGA-accelerated real-time data processing and demonstrated its feasibility and scalability in the context of the AWAKE camera acquisition system. The findings have important implications for the scientific community and offer valuable insights that can be applied to the development of similar systems in the future.

Bibliography

[1] Basler AG. *Basler ace GigE User's Manual*. Basler AG. 2018.

[2] Ankit Agarwal et al. 'Real-time Trading Using FPGA'. In: *Communications of the ACM* 61.11 (2018), pp. 53–61.

[3] J. Aguilar et al. 'A Comparative Study of High-Level Synthesis Tools for FPGAs: Vivado HLS, Catapult C, and LabVIEW FPGA'. In: *IEEE Access* 8 (2020), pp. 217181–217197.

[4] Twan Basten, Henk Corporaal and Rainer Ernst. *Embedded systems design: A unified hardware/software introduction*. Springer, 2011.

[5] J. Bhasker. *A VHDL Primer*. 3rd. Prentice Hall, 1998. ISBN: 9780130965756.

[6] Bruno Miguel Gomes Carrasqueira. 'Accelerating Online Data Processing at LHCb using Field Programmable Gate Arrays'. MA thesis. Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2019. URL: https://run.unl.pt/bitstream/10362/93698/1/Carrasqueira_2019.pdf.

[7] Cedric Charrondiere, Thomas Zillio, Slawomir Sudak, Henrik Johannessen. *[AWK-Cam] AWAKE Camera Acquisition System*. https://readthedocs.web.cern.ch/display/MTA/%5BAWK-Cam%5D+AWAKE+Camera+Acquisition+System. CERN. 2023.

[8] CERN. *AWAKE successfully accelerates electrons*. CERN. 18th Feb. 2021. URL: https://home.cern/news/news/experiments/awake-successfully-accelerates-electrons (visited on 22nd Mar. 2023).

[9] Jian Chen, Dawei Wu and Yanzhao Qiu. 'FPGA-based Digital Signal Processing'. In: *International Journal of Computer Theory and Engineering* 4.6 (2012), pp. 1012–1017. DOI: 10.7763/ijcte.2012.v4.570.

[10] Jing-Jia Chen and John A Stankovic. 'Real-Time Systems'. In: *Handbook of Computer Networks*. Springer, 2008, pp. 1–38.

[11] Intel Corporation. *Intel Quartus Prime Pro Edition Handbook*. 2021. URL: https://www.intel.com/content/www/us/en/programmable/documentation/lro1543472477946.html.

[12] National Instruments Corporation. *LabVIEW Real-Time Module User Manual*. Aug. 2020. URL: https://www.ni.com/pdf/manuals/376567c.pdf.

[13] Ronald E. Crochiere and Lawrence R. Rabiner. *Multirate Digital Signal Processing*. Prentice Hall, 1983.

[14] M David, F Yao and W Luk. 'FPGA-based Real-Time Image Processing for the ALICE Experiment at CERN'. In: *IEEE Transactions on Nuclear Science* 61.4 (2014), pp. 1884–1892.

[15] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson Education, 2006.

[16] HardwareBee. *Understanding FPGA Programming and Design Flow*. 2018. URL: https://hardwarebee.com/understanding-fpga-programming-and-design-flow/#gallery-1 (visited on 12th Apr. 2023).

[17] David Harris and Sarah Harris. *Digital Design and Computer Architecture*. 2nd. Morgan Kaufmann Publishers, 2009.

[18] Reiner Hartenstein and Jürgen Becker. *Configurable Computing: Technology and Applications*. 1st. Springer, 2001. ISBN: 9781461530325.

[19] Scott A. Hauck and André DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. 1st. Morgan Kaufmann, 2007. ISBN: 9780123705228.

[20] José Luis Hernández-Valverde, David Juárez-Romero and Luis J de la Cruz Llopis. 'Real-time operating systems: a review from the deterministic scheduling perspective'. In: *Journal of Systems Architecture* 55.5 (2009), pp. 332–353.

[21] Xilinx Inc. 'FPGA vs. ASIC Comparison'. In: (2022). URL: https://www.xilinx.com/support/documentation/white_papers/wp272-FPGA-vs-ASIC-Comparison.pdf.

[22] National Instruments. *LabVIEW FPGA Compilation Process*. https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YHVTCA4&l=en-NO. 2021.

[23] National Instruments. *NI PXIe-7915 User Manual*. National Instruments. 2018.

[24] National Instruments. *PXI Platform Specification*. National Instruments. 2020.

[25] E. de Kock, J. van der Veen and W. Luk. 'Accelerating Real-time Image Processing on FPGAs, GPUs, and DSPs: A Comparative Study'. In: *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Sept. 2014, pp. 1–8. DOI: 10.1109/FPL.2014.6927459.

[26] Boris Kovac et al. 'Hardware acceleration of image processing algorithms using FPGA devices'. In: *Proceedings of the 35th International Convention MIPRO*. IEEE. 2012, pp. 1049–1054.

[27] Rakesh Kumar, Pawan Kumar and Ravi Kumar. 'FPGA-Based Radar Signal Processing System'. In: *Proceedings of the 2012 International Conference on Communication, Information & Computing Technology (ICCICT)*. 2012, pp. 16–21. DOI: 10.1109/ICCICT.2012.6398669.

[28] Ian Kuon and Jonathan Rose. 'Measuring the Gap Between FPGAs and ASICs'. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.6 (2008), pp. 1007–1018. DOI: 10.1109/TCAD.2008.923715.

[29] Wenjing Liu, Shiyuan Wang and Jinyu Luo. 'A survey on FPGA design techniques'. In: *Journal of Systems Architecture* 96 (2019), pp. 102–116. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.04.005. URL: https://www.sciencedirect.com/science/article/pii/S1383762118303479.

[30] DK Mishra, SS Rout and S Mishra. 'Real-time Image Processing using FPGA and its Applications: A Review'. In: *International Journal of Computer Applications* 149.2 (2016), pp. 34–40.

[31] National Instrument. *PXIe-1095 Specifications*. https://www.ni.com/docs/en-US/bundle/pxie-1095-specs/page/specs.html. National Instrument. 2023.

[32] National Instrument. *PXIe-7915 Specifications*. https://www.ni.com/docs/en-US/bundle/pxie-7915-specs/page/specs.html. National Instrument. 2023.

[33] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 1999.

[34] Andrew Putnam et al. 'A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services'. In: *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*. 2014, pp. 13–24. DOI: 10.1109/ISCA.2014.6853198.

[35] Sergio Saponara et al. 'FPGAs for Automotive Applications: A Comprehensive Survey'. In: *IEEE Transactions on Industrial Informatics* 15.3 (2019), pp. 1438–1453. DOI: 10.1109/TII.2018.2875886.

[36] Lattice Semiconductor. *Lattice Diamond User Guide*. 2021. URL: https://www.latticesemi.com/-/media/LatticeSemi/Documents/UserManuals/EI/LatticeDiamondUserGuide.ashx.

[37] Ali Sharifi Shirazi, Hossein Zayyani and Mehdi Mokhtarzade. 'A Review of Image Denoising Algorithms Based on Partial Differential Equations'. In: *Signal, Image and Video Processing* 12.2 (2018), pp. 227–244.

[38] Dharmendra Singh and Prashant Patil. 'FPGA-Based Digital Signal Processing for Medical Imaging: A Review'. In: *Journal of Medical Systems* 42.8 (2018), p. 139. DOI: 10.1007/s10916-018-1006-7.

[39] David B Stewart and Lawrence D Paulson. 'Achieving determinism in real-time systems'. In: *Proceedings of the IEEE* 82.1 (1994), pp. 55–63.

[40] S. Trimberger. 'FPGAs and Reconfigurable Computing: History and Future'. In: *Proceedings of the IEEE* 103.3 (Mar. 2015), pp. 407–425. DOI: 10.1109/JPROC.2015.2395220.

[41] Stephen Trimberger. 'FPGAs: Instant Access'. In: *IEEE Spectrum* 35.9 (1998), pp. 50–55. DOI: 10.1109/6.709616.

[42] John D. Villasenor and John Wawrzynek. 'High-Level Synthesis for FPGAs: From Prototyping to Deployment'. In: *Proceedings of the IEEE* 98.2 (2010), pp. 332–352. DOI: 10.1109/JPROC.2009.2037634.

[43] Bogdan Wojtsekhowski et al. 'FPGA-based Data Acquisition and Control System for Nuclear Physics Experiments'. In: *IEEE Transactions on Nuclear Science* 61.4 (2014), pp. 2227–2233. DOI: 10.1109/TNS.2014.2334402.

[44] Ruochen Zhang et al. 'Survey on FPGA-based Deep Learning Accelerators'. In: *ACM Computing Surveys* 54.2 (2021), pp. 1–27. DOI: 10.1145/3446875.

Appendix

camera_processing_time_visualization.py

```
import re
from collections import defaultdict
import matplotlib.pyplot as plt
from datetime import datetime

def parse_file(file_path):
    data = defaultdict(list)

    with open(file_path, 'r') as file:
        content = file.read()
        entries = content.split('\n')

        for i in range(0, len(entries), 6):
            if i + 5 >= len(entries):
                break

            camera_name = entries[i].strip()
            camera_number = int(re.search(r'\d+', camera_name).group())
            acquisition_timestamp = datetime.strptime(entries[i + 4].strip(), "%I:%M:%S.%f %p %m/%d/%Y")
            processing_time = int(entries[i + 5].strip()) - 100

            data[camera_number].append((acquisition_timestamp, processing_time))

    return data

def plot_data(data):
    plt.figure(figsize=(10, 6))

    min_timestamp = min(t for values in data.values() for t, _ in values)

    for camera_number, values in sorted(data.items()):
        if not values:  # Skip if there are no values for the camera
            continue

        # Calculate relative time (seconds + milliseconds) for the x-axis
        relative_times = [(t - min_timestamp).total_seconds() for t, _ in values]
        processing_times = [pt for _, pt in values]
        plt.scatter(relative_times, processing_times, label=f'cam{camera_number:02d}')

    plt.xlabel('Time (s.ms)')
    plt.ylabel('Processing Time')
    plt.title('Processing Time for Each Camera')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    file_path = r"C:\Users\henri\Documents\VS-Code\NTNU del 1.5\ReadDataFromTXT\processingtimelog.txt"
    data = parse_file(file_path)
    plot_data(data)
```

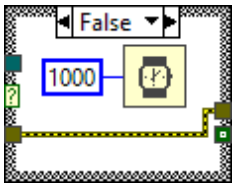

Figure 36: rt_main.vi false case

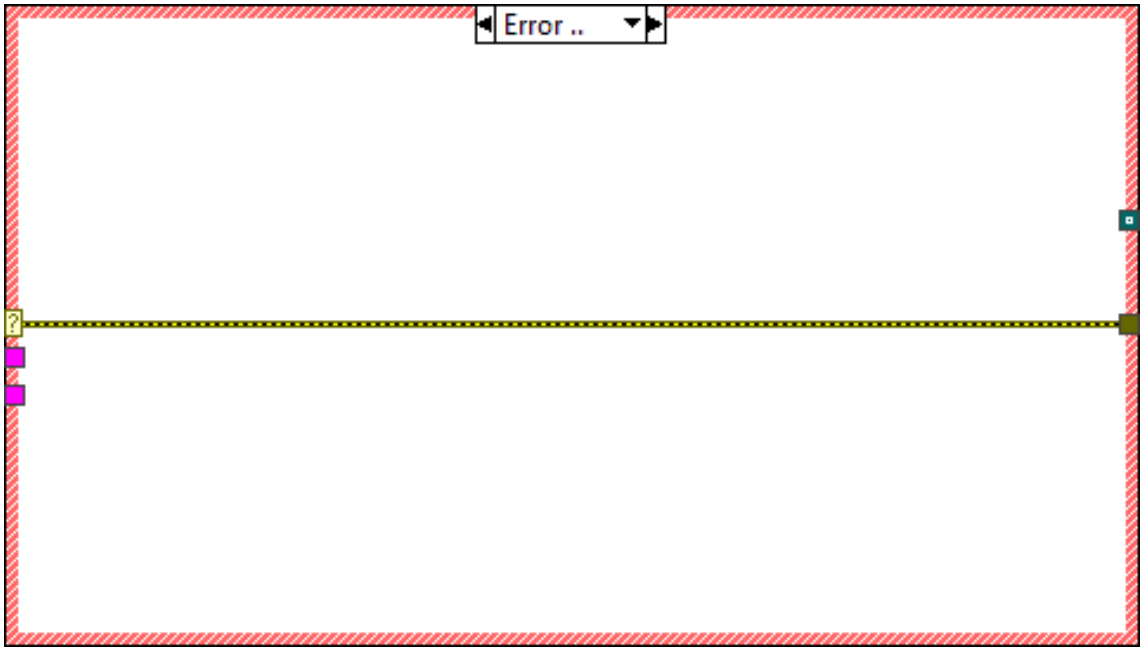

Figure 37: init.vi error case

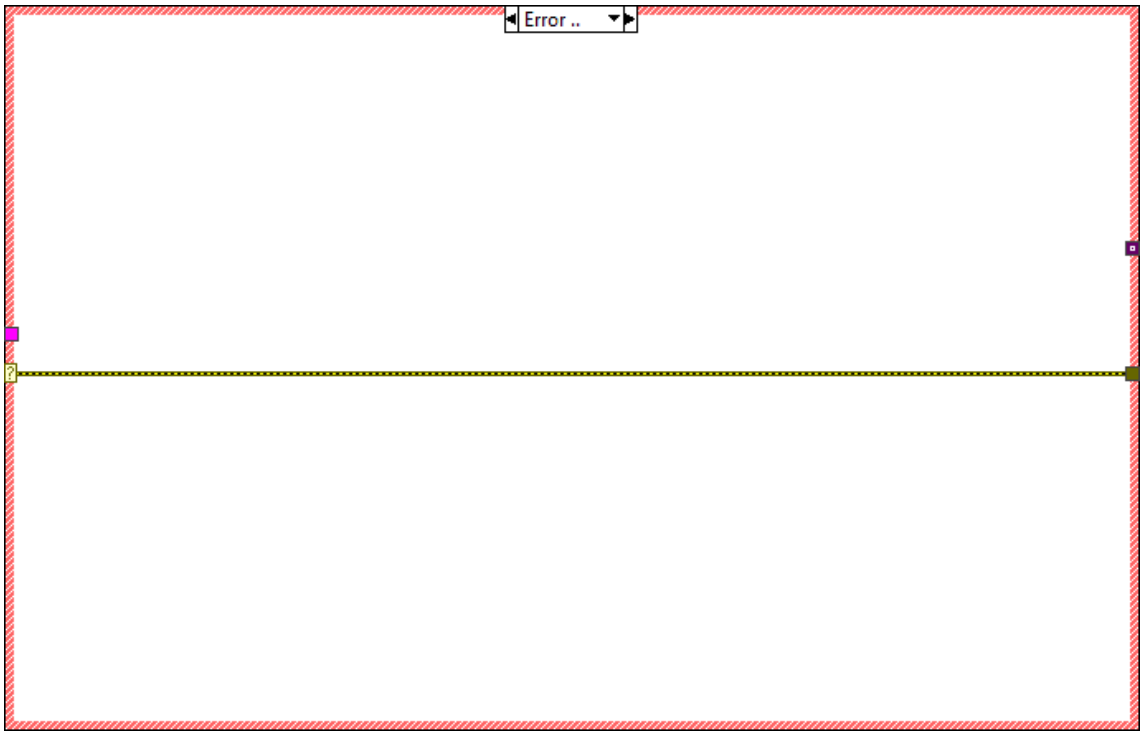

Figure 38: init_camera.vi error case

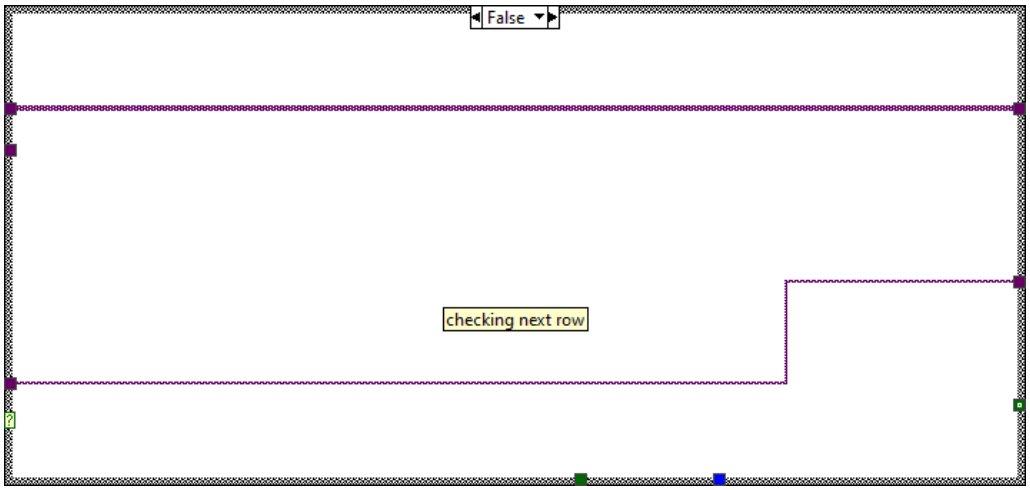

Figure 39: camera_sorting.vi case 1

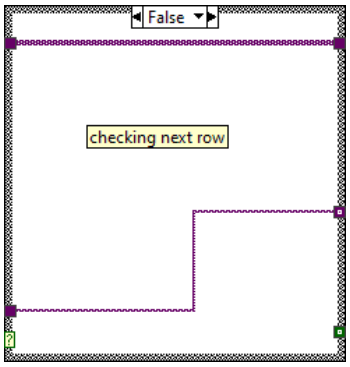

Figure 40: camera_sorting.vi case 2

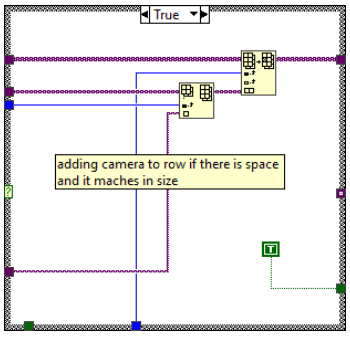

Figure 41: camera_sorting.vi case 3

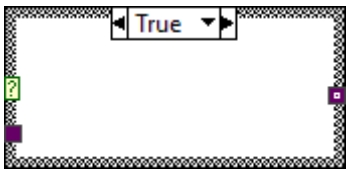

Figure 42: camera_sorting.vi case 4

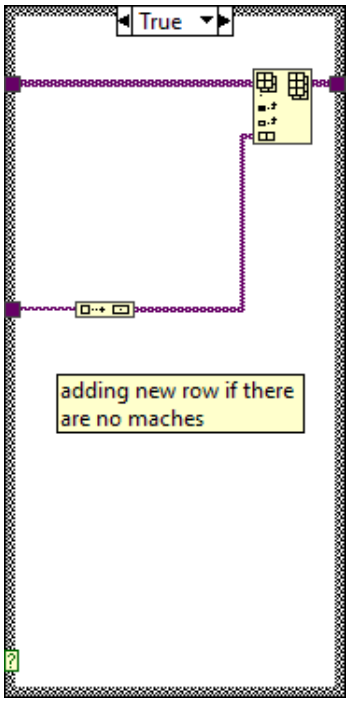

Figure 43: camera_sorting.vi case 5

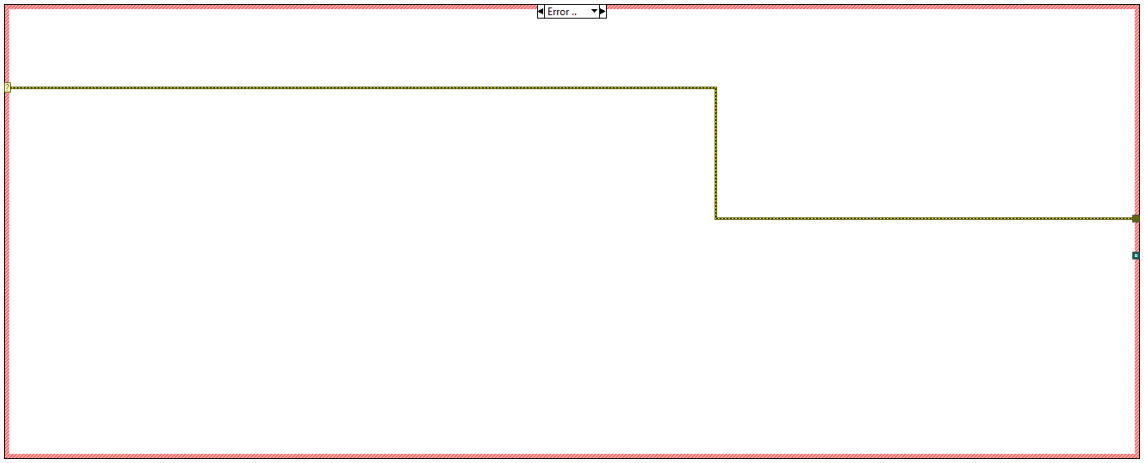

Figure 44: init_FPGA.vi error case

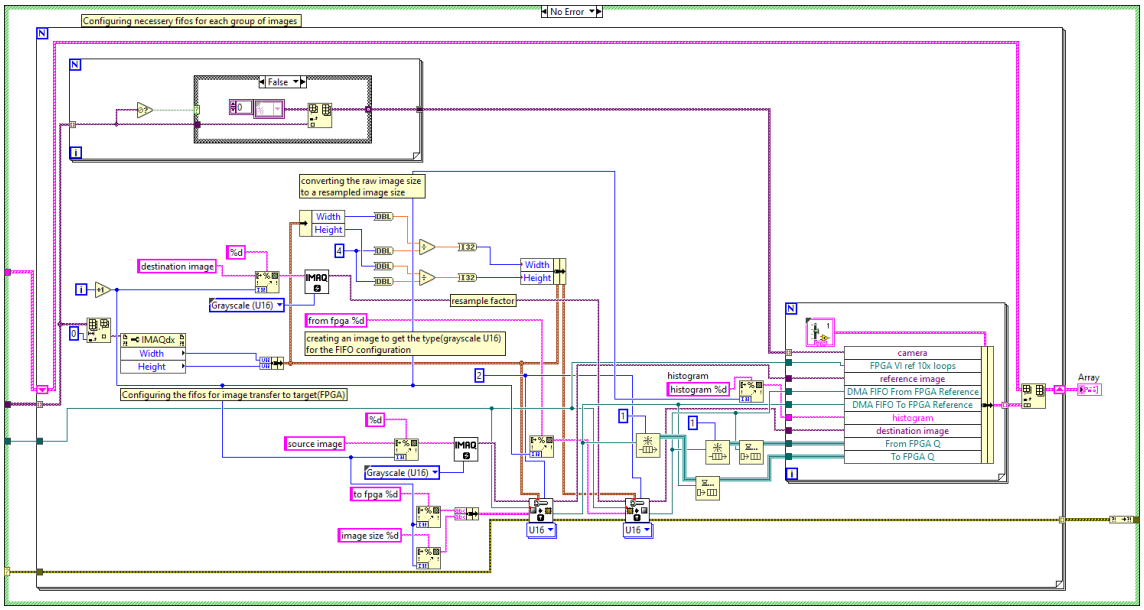

Figure 45: init_FIFO.vi error case

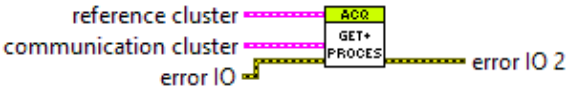

Figure 46: acquisition_deamon.vi error case

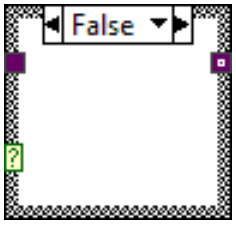

Figure 47: acquisition_deamon.vi case 2

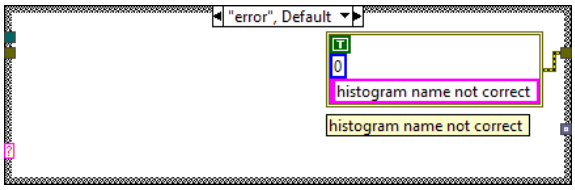

Figure 48: acquisition_deamon.vi case 3

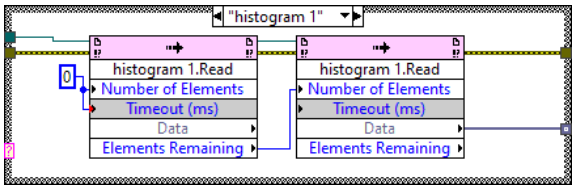

Figure 49: acquisition_deamon.vi case 4

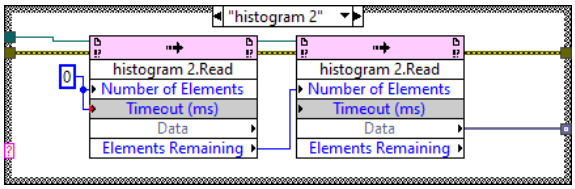

Figure 50: acquisition_deamon.vi case 5

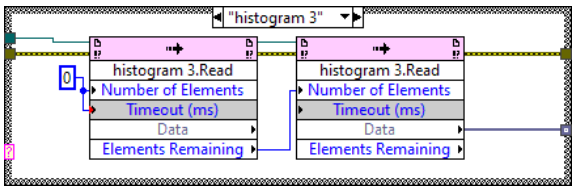

Figure 51: acquisition_deamon.vi case 6

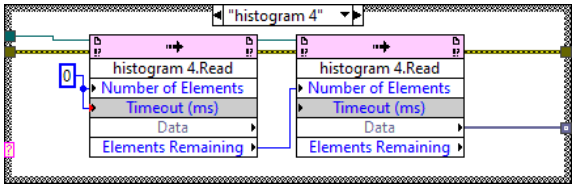

Figure 52: acquisition_deamon.vi case 7

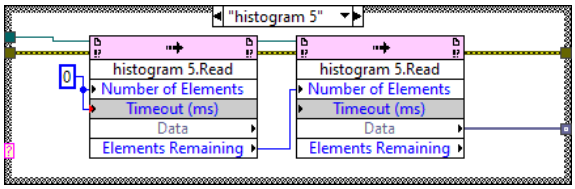

Figure 53: acquisition_deamon.vi case 8

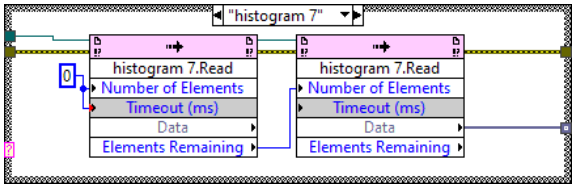

Figure 54: acquisition_deamon.vi case 9

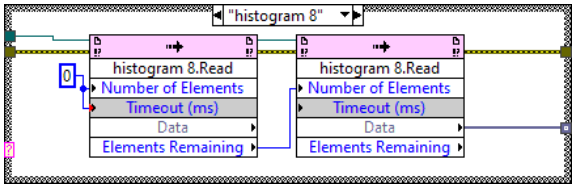

Figure 55: acquisition_deamon.vi case 1,0

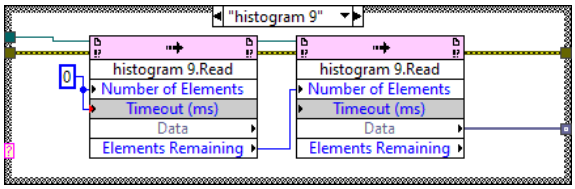

Figure 56: acquisition_deamon.vi case 1,1

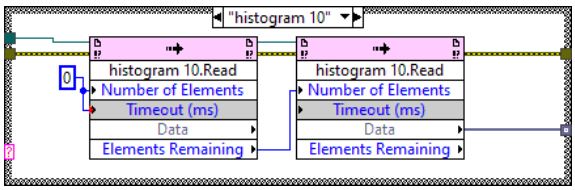

Figure 57: acquisition_deamon.vi case 1,2

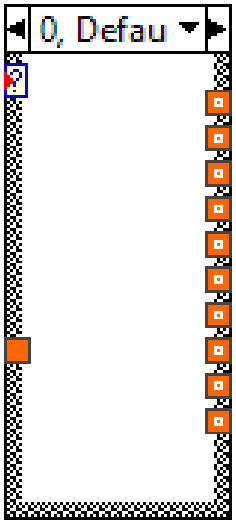

Figure 58: logger.vi case 1

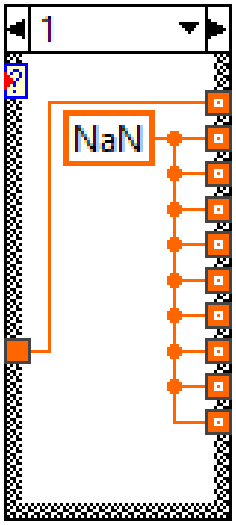

Figure 59: logger.vi case 2

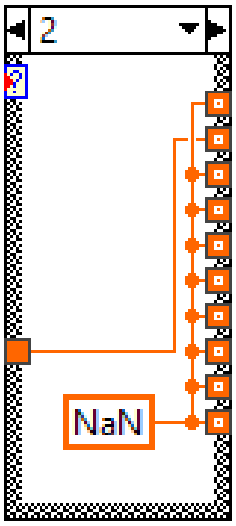

Figure 60: logger.vi case 3

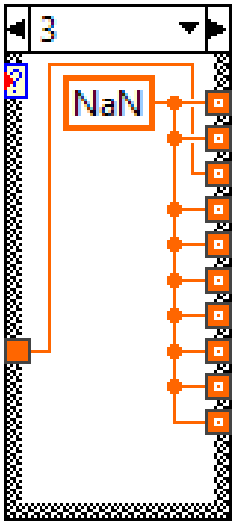

Figure 61: logger.vi case 4

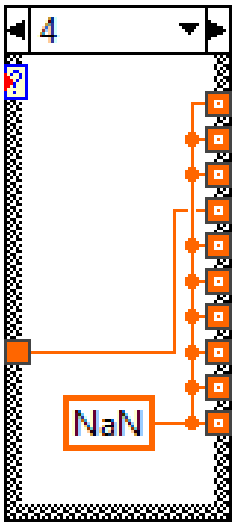

Figure 62: logger.vi case 5

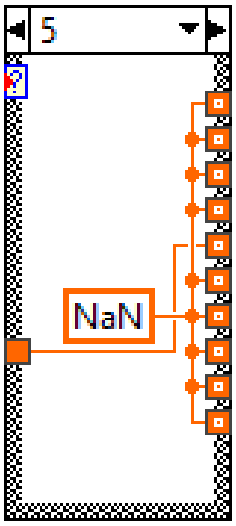

Figure 63: logger.vi case 6

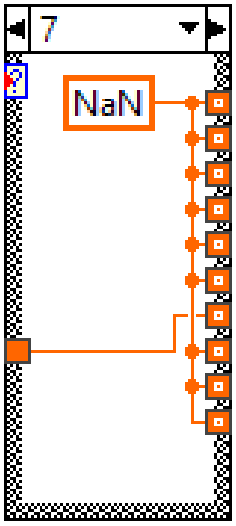

Figure 64: logger.vi case 7

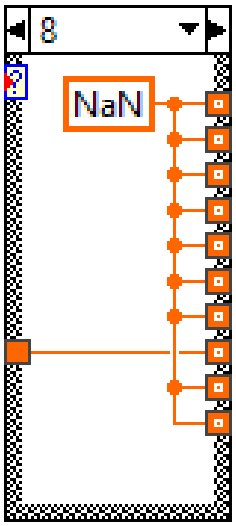

Figure 65: logger.vi case 8

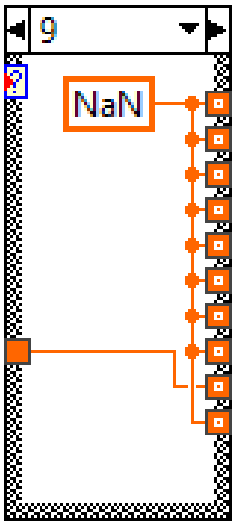

Figure 66: logger.vi case 9

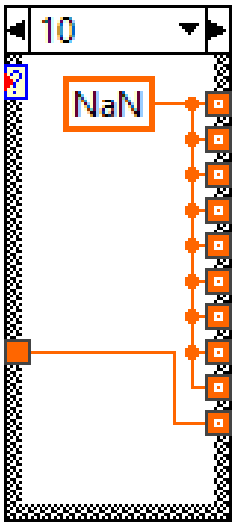

Figure 67: logger.vi case 10

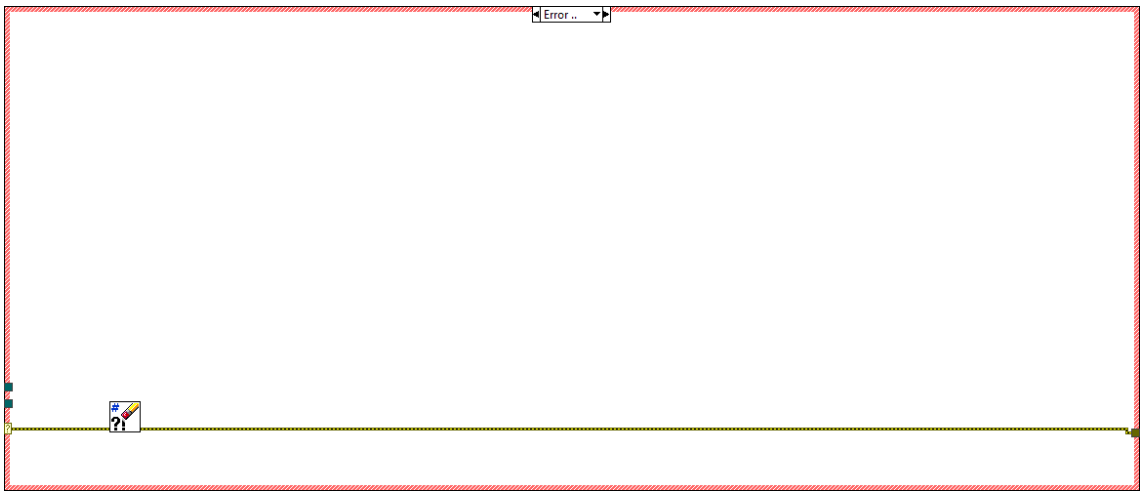

Figure 68: logger.vi case 11

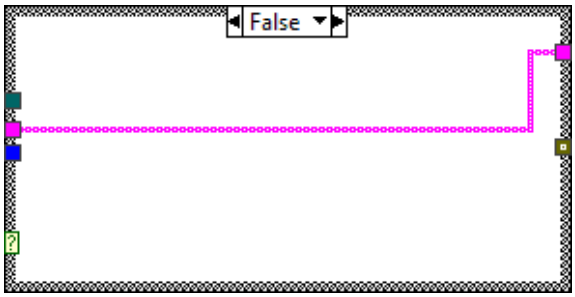

Figure 69: logger.vi case 12

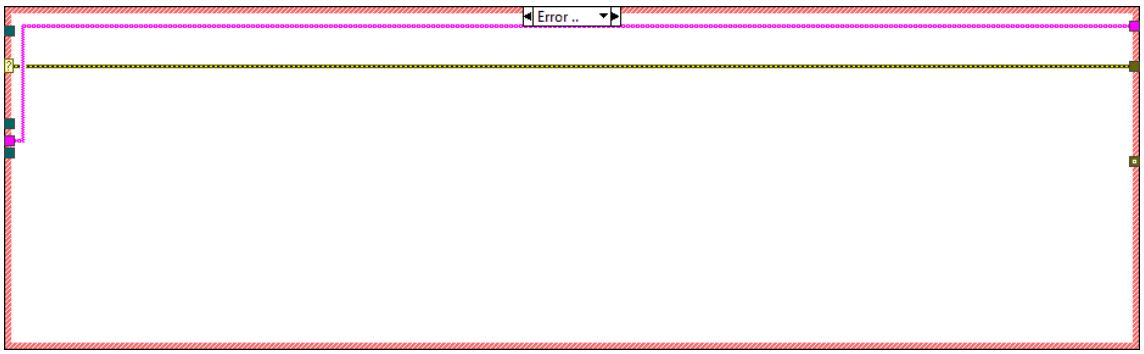

Figure 70: logger.vi case 13

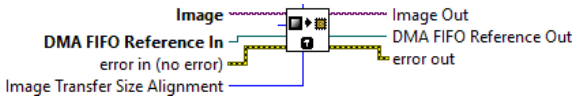

Figure 71: IMAQ_FPGA_Image_Transfer_to_Target_U16.vi icon

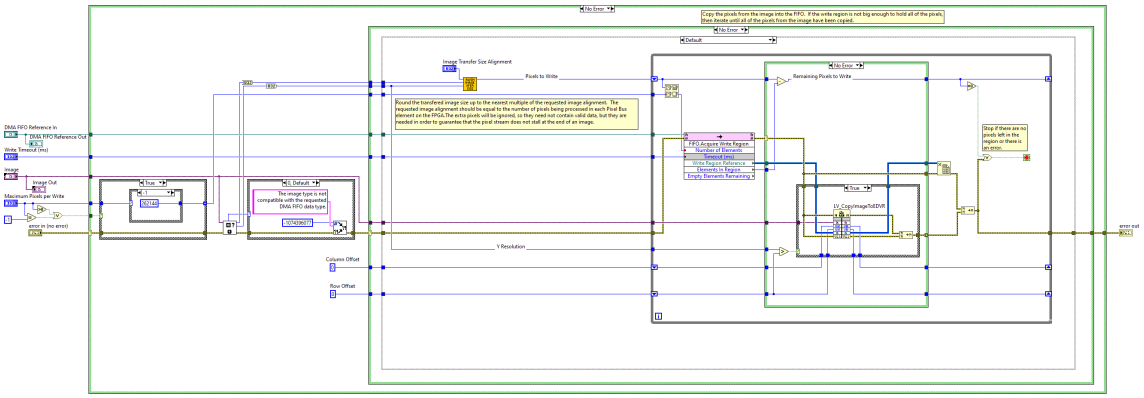

Figure 72: IMAQ_FPGA_Image_Transfer_to_Target_U16.vi block diagram

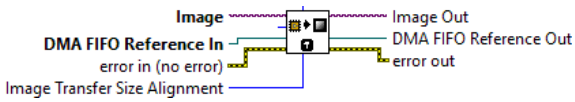

Figure 73: IMAQ_FPGA_Image_Transfer_from_Target_U16.vi icon

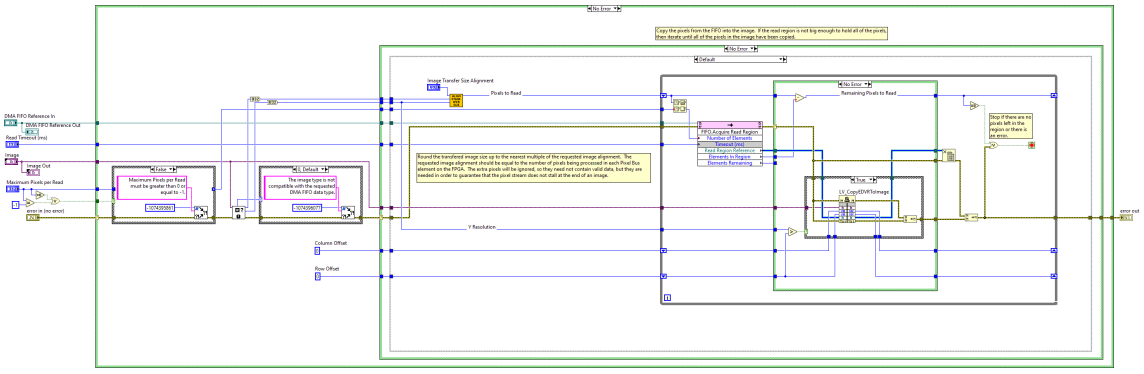

Figure 74: IMAQ_FPGA_Image_Transfer_from_Target_U16.vi block diagram