A Study on Management System and Parallel Application Library for Stand-Alone FPGA Clusters

July 2022

Kazuei Hironaka

A Study on Management System and Parallel Application Library for Stand-Alone FPGA Clusters

Kazuei Hironaka

A thesis for the degree of Ph.D. in Engineering
Under the supervision of **Prof. Hideharu Amano**

Graduate School of Science and Technology
Keio University

July 2022

Acknowledgment

First of all, I would like to express my gratitude to my supervisor Prof. Hideharu Amano (Hunga-san) for allowing me to pursue my Ph.D. course at Keio University.

Probably I wasn't a promising Ph.D. student. There were so many interruptions due to my life events and struggling with my research progress. However, he gave me a chance to pursue my research until the end. As a result, I could keep pushing the wall and reaching here even though overdue. Also, I greatly appreciate Prof. Hiroaki Nishi, Prof. Hiroki Matsutani, and Dr. Kentaro Sano. They kindly accepted to review to shape this thesis.

During my Ph.D. research period was struggled to pursue both my job and research. However, it was a fun time, and kindly lab mates supported me. Significantly, thanks to Dr. Hayate Okuhara and Kensuke Iizuka for helping me in lab life, and Dr. Akram Ben Ahmed and Dr. Nguyen Ahh Vu Doan for supporting my research.

I graduated M.A. course in 2012 and came back in 2016. I have been working in a corporate laboratory as a researcher as my profession and doing my academic research as a Ph.D. student on weekends. Many thanks to my company colleagues regarding my Ph.D. activities. Particularly to Mr. Kenta Sato in Hitachi Central Laboratory for giving an idea and sharing knowledge about MPI implementation.

From my decade of experience in the corporate laboratory, I feel the research in corporate tends to be shortsighted, and there is less freedom than in academic research. It is an unavoidable corporate saga; it needs to earn money and return it to the stakeholders. However, a weak collaboration between academia and industry is a critical problem for the whole ICT industry of this nation. Therefore, I want to be a person who breaks the barrier to connecting academia and industry after graduation.

Finally, since 2016, my Ph.D. period might have sacrificed precious family time. I want to thank my wife, Samantha, and my lovely sons, Akihiro and Tomohiro, for understanding my Ph.D. activities.

Kazuei Hironaka (nyacom)
July 2022

Abstract

In recent years, Mobile Edge Computing (MEC) has been gaining attention due to the progress of the expansion of the 5th Generation mobile network (5G), which offers high capacity and low latency wireless communication. MEC is an edge computing method that uses computing resources located in the edge environment of a wireless base station to application task offloading requiring computing power near client devices. Utilizing the 5G network and MEC enables the implementation of various applications that have been difficult to achieve with cloud computing and mobile devices due to network latency.

However, MEC is often installed on power and space restricted environments such as wireless base stations; hence, it is difficult to employ a conventional computing platform used in the data center. Therefore, this thesis focuses on Field-Programable-Gate-Array (FPGA) as a computing platform for MEC, which has high flexibility with programmable logic circuits and achieves high performance with hardware-based data processing. Traditionally, FPGAs have been utilized as hardware devices to realize logic circuits for specific applications. However, the latest FPGAs are multifunctional and utilized for highly efficient general computing acceleration platforms.

As a proof-of-concept to validate the application of FPGA systems to MEC, we have developed a scale-flexible stand-alone multi-FPGA system called FiC and its successor M-KUBOS. A stand-alone multi-FPGA system is an FPGA system consisting of multiple stand-alone FPGA nodes with a high-speed inter-FPGA network, and each FPGA node can operate in both stand-alone and clustered mode. The system scale is flexibly adjustable according to the application and performance requirements by changing the number of FPGA nodes. Utilizing FPGA's high energy efficiency and flexibility, the stand-alone multi-FPGA system is expected to be an ideal computing platform for MEC, achieving space-saving and low power consumption.

On the other hand, FPGA is still a low-layer device compared to CPUs and GPUs. There are challenges in improving platform accessibility, manageability, and application programmability to adopt a stand-alone multi-FPGA system for MEC. For instance, FPGAs are versatile devices and can implement various dedicated

hardware logic, but in order to use them from applications, they generally require dedicated device drivers and middleware. Therefore, to adopt FPGA as a computing platform on MEC, it is necessary to provide a general-purpose platform management interface and protocol that various application platforms can support. In addition, application programmability for stand-alone multi-FPGA systems is a fundamental problem because no standard method has been provided. Hence, providing an application development environment for a stand-alone multi-FPGA system is also required.

To facilitate the use and management of stand-alone multi-FPGA systems from applications, this thesis proposes a platform management system called *FiC-RFC* for stand-alone multi-FPGA systems. With FiC-RFC, MEC applications can access stand-alone multi-FPGA systems directly via simple HTTP access by RESTful APIs, which is a programming fashion widely used in cloud applications. Using RESTful APIs allows direct use of stand-alone multi-FPGA systems from various application platforms and improves platform accessibility and manageability. In a scenario that provisioning FPGA applications by FiC-RFC to 12 nodes FiC system over HTTP access, FiC-RFC achieved scalable management performance and provisioned an application below 20 seconds. The result shows that FiC-RFC improves platform utilization efficiency compared to conventional application provisioning methods such as JTAG, and it supports enough manageability as a computing platform on MEC.

To improve the programmability of applications on stand-alone multi-FPGA systems, we also focus on Message Passing Interface (MPI), a parallel programming environment commonly used in distributed memory architectures, and propose *FiC-MPI*, an MPI-compatible library to facilitate parallel programming in stand-alone multi-FPGA systems. This MPI-compatible library can be directly used in the standard C/C++ HLS FPGA application development flow. It is expected to enable application designers to design parallel programming applications with MPI interface for multi-FPGA nodes easily and improve the programmability of stand-alone multi-FPGA systems. Also, the FiC-MPI provides an application test and debug environment called FiC-MPI Simulator for the MPI applications. It eliminates FPGA implementation of the debugging on the real FPGA platform and improves application productivity dramatically. In order to show the feasibility and usability of FiC-MPI, the thesis ported an MPI-based general numerical benchmark *Himeno Benchmark* (Himeno-BMT) to an application for six nodes M-KUBOS system as a case study. The ported Himeno-BMT implemented with FiC-MPI achieved 178.7 MFLOPS with a single node; scaled to 643.7 MFLOPS with four nodes; and 896.9 MFLOPS with six nodes. The result showed that the FiC-MPI could achieve scalable performance along with the number of nodes, and the case study demonstrated the easiness of developing parallel programs with FiC-MPI on multi-FPGA systems.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

In recent years, the number of mobile devices connecting to the Internet, such as smartphones and the Internet-of-Things (IoT), has increased explosively. According to a study [1] in 2017 by Deloitte, smartphone penetration in developed and developing countries is more than 80%, 4G and other high-speed mobile network coverage reached over 50%. Most U.S. and other developed countries are investing and deploying the 5th generation mobile network (5G) progressively, which has higher capacity and lower latency than the previous 4G network [2].

With the popularization of these mobile devices and the expansion of high-speed mobile networks, edge computing methods known as Multi-access Edge Computing (MEC) [3] have been gaining attention. MEC is an extension of mobile-cloud computing (MCC), which provides data processing services to mobile devices that require real-time and high computational performance, using computing resources installed on the radio base station. MEC is expected to reduce network latency for accessing computing resources; it enables applications that require real-time performance, such as real-time image processing and artificial intelligence (AI) applications; it improves application users' quality of experience (QoE).

The computing platform can provide scalable and sufficient computing power on MEC to meet various application demands is desired. However, the environment of MEC is often strongly restricted for space and power consumption due to installation at radio stations; therefore, deploying high-performance computing equipment like those used in cloud computing is not feasible. Thus, an eligible computing platform

for MEC with excellent space-saving and power efficiency is demanded.

1.2 Motivation and Objectives

Figure 1.1 shows an example of a computing platform proposed for MEC. Currently, embedded server vendors and industry organizations such as the Open Radio Access Network (O-RAN) Alliance [4] has proposed high-density computing platform that integrates servers, storage, and networking in a single unit or rack for the computing platform for MEC.

These existing edge server platforms on the market are versatile, and they can be adapted to a variety of use cases. However, these CPU-based platforms are basically optimized for batch processing of memory data, and they are unsuitable for streaming data processing requiring predictable performance and latency. For example, video analysis is one of the typical applications for processing streaming data [5][6][7] expected to be widely used in applications, such as face detection [8], automatic license plate recognition (ALPR) [7], object recognition for augmented reality [9], etc. Although it is possible to install accelerators such as GPUs on the edge server platform to support these applications, it hardly deploys these systems on a power-constrained and thermal-constrained environment due to GPU's high energy consumption.

Figure 1.1: Converged Platform for 5G Edge Computing by Advantech [10]

To fulfill the computing demands on MEC, this thesis focused on using a multi-FPGA system as a computing acceleration platform for MEC, which can deliver a small footprint and high power efficiency. FPGA is a flexible device that can realize a computing platform with high performance and low power consumption by implementing application-specific hardware. Compared to the GPU, the FPGA is expected to achieve 3-4 times lower power consumption and 10-60 times better energy efficiency [11][12]. Combining multiple FPGAs to structure a multi-FPGA system, it is possible to build a computing platform that can deliver scalable

computing performance and flexible system size to meet application requirements and demands.

As a multi-FPGA system targeted for MEC, the FiC project [13] proposed a stand-alone multi-FPGA system called FiC cluster and its successor M-KUBOS cluster. A stand-alone multi-FPGA system is an FPGA system design that each FPGA node can be used as a stand-alone FPGA host, but also it can construct a clustered multi-node FPGA system. This architectural design is expected to achieve scalable computing performance and system flexibility to fit MEC. The FiC and M-KUBOS clusters employ this stand-alone multi-FPGA design and consist of a high-speed FPGA-to-FPGA network that directly connects cost-effective mid-range FPGAs. Each cluster node operates as an independent FPGA server in stand-alone mode, providing FPGA compute offload services to applications. Depending on the problem, applications can use these nodes as individual or clustered compute resources. However, in order to apply these FiC and M-KUBOS as a computing platform on MEC and to enable FPGA acceleration for MEC applications, the following issues need to be addressed.

FPGA's platform accessibility and manageability issues

For applying FPGA systems to MECs, the FPGAs are devices with low-level access interfaces at the hardware level. The accessibility and manageability for applications to handle FPGA platforms must be improved. In addition, accessibility from remote environments via the Internet should be considered.

Application programmability issues of stand-alone multi-FPGA systems

Currently, many efforts are made by FPGA vendors to improve the development productivity of FPGA applications, such as high-level synthesis (HLS) using C/C++. These efforts have enabled software engineers to develop FPGA applications, and they have dramatically improved the productivity of FPGA applications. However, these FPGA development tools target a single FPGA, and there is no standard way to design bare-metal applications targeting multi-FPGA systems such as FiC and M-KUBOS. Typically, application development on FiC and M-KUBOS requires the implementation of communication protocols specific to each application, and its debugging and testing environment is inadequate. Thus, there is a need to improve the programmability and productivity of applications targeting stand-alone multi-FPGA systems.

The objective of this thesis is to solve the issues of adopting multi-FPGA systems such as FiC and M-KUBOS for MEC and to build an efficient computation platform.

1.3 Thesis Contributions

Development of a platform management system to improve accessibility and manageability of stand-alone multi-FPGA platforms

In order to apply the multi-FPGA platform to MEC, we investigated access methods and management of the multi-FPGA platform from applications, and we focused on Representational State Transfer (REST), a general-purpose programming interface used in cloud applications. By applying this REST concept to the application interface of the multi-FPGA platform, we studied an access method and management architecture that is less dependent on a specific environment and enables the use of the multi-FPGA platform from various application platforms. This study developed FiC Restful FPGA Control (FiC-RFC), a platform management system for FiC and M-KUBOS using RESTful APIs. FiC-RFC enables applications to remotely use and manage FiC and M-KUBOS using a standardized interface provided by the RESTful API, allowing applications on MEC to improve accessibility and manageability of FiC and M-KUBOS.

Development MPI based parallel programming environment for stand-alone multi-FPGA architecture

We focused on the Message Passing Interface (MPI) parallel programming environment commonly used in distributed memory systems to improve the programmability of bare-metal multi-FPGA applications on FiC and M-KUBOS. Designing applications using MPI is expected to improve application programmability by making it possible to design applications using parallel programming on FiC and M-KUBOS application design. In this study, we developed an MPI library, FIC-MPI, which can be directly used in the FPGA application development flow with standard C/C++ and HLS. We also developed the FiC-MPI Simulator to improve the efficiency of application testing and debugging of multi-FPGA applications using FiC-MPI. Compared to traditional methods for developing bare metal multi-FPGA applications with FiC and M-KUBOS, FiC-MPI can significantly improve programmability and the efficiency of the application debugging process by more than 100 times.

1.4 Thesis Outline

The rest of this thesis is organized as shown in Figure 1.2.

Chapter 2: Background and State-of-the-art

This chapter introduces related topics of this thesis and state-of-the-art FPGA, then introduces various multi-FPGA system architectures by use cases.

Chapter 3: Target Systems

This chapter first introduces FiC and M-KUBOS, the stand-alone multi-FPGA systems covered in this thesis, describe their concepts and detailed hardware architecture, and then discusses challenges in applying FiC and M-KUBOS to MEC.

Chapter 4: Platform Management Architecture for Stand-Alone Multi-FPGA Systems

In this chapter, we propose a platform management system, FiC-RFC, which improves the accessibility and manageability of the platform when stand-alone multi-FPGA systems such as FiC and M-KUBOS are applied to MEC. FiC-RFC is a platform management system for FiC and M-KUBOS developed by focusing on HTTP access and RESTful API, which are cloud application methods, to provide remote access and management functions to FPGA platforms. FiC-RFC standardizes the operations required for managing FPGA clusters as a RESTful API, thereby providing a device- and environment-independent application interface. The APIs enable MEC applications and upper-level management systems to remotely access FiC and M-KUBOS for application execution and management using general-purpose HTTP communication.

Chapter 5: Parallel Programming Environment for Multi-FPGA Systems

This chapter proposes an MPI-compatible parallel programming library, FiC-MPI, to improve the programming environment for multi-FPGA systems on FiC and M-KUBOS. FiC-MPI is an MPI library that can be directly used in bare-metal FPGA application development with HLS and was developed to facilitate multi-FPGA parallel application design and improve the efficiency of application testing and debugging tasks. The productivity of application development with FiC-MPI is discussed by comparing it with conventional application development methods. As a case study, we also ported the *Himeno Benchmark*, a general numerical computation benchmark implemented by MPI, to M-KUBOS using FiC-MPI and showed the practicality of FIC-MPI from its performance benchmark and performance scaling results.

Chapter 6: Conclusion and Future Work

This chapter concludes this thesis and outlook for future work.

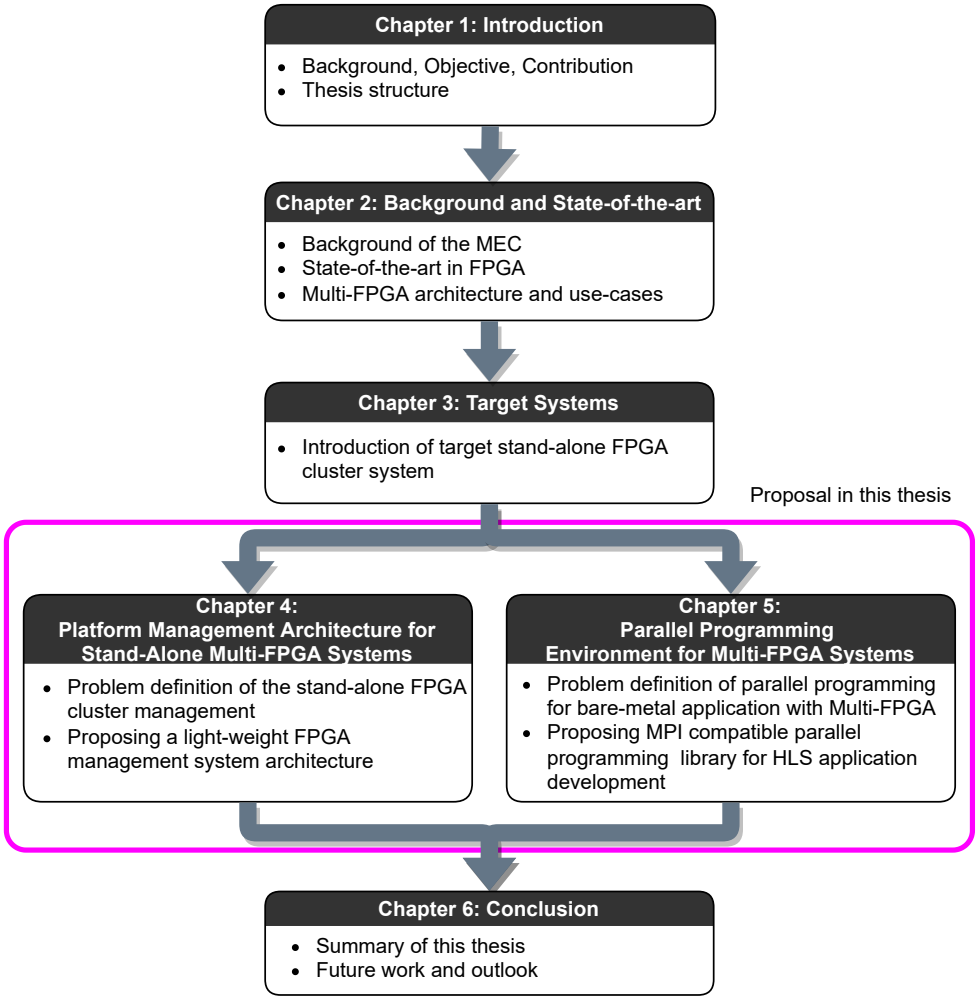


Figure 1.2: Outline of this thesis

2

Background and State of the Art

2.1 Multi-access Edge Computing (MEC)

Multi-access Edge Computing (MEC) [3] is an edge computing method standardized by ETSI (European Telecommunication Standards Institute). Figure 2.1 shows an example MEC architecture and possible IoT use cases. MEC enables edge computing by installing powerful compute nodes (MEC Server) at 5G wireless base stations to take advantage of the 5G's high-speed and low-latency network (Figure 2.2). It realizes various timing-critical IoT use cases, which have been challenging to achieve in the past due to network latency and insufficient computing power on edges. Examples include video streaming and analysis, autonomous driving, real-time drone control, and merged and augmented reality (MR/AR) [5][6][14].

According to iGR [16], the number of MEC installations in the U.S. is expected to reach 563,000 stations in various industries by 2026. Especially in the retail industry, it has the potential of the most significant number of small MEC station operators that may have more than 2,000 locations across the U.S. Such compact MEC stations are typically installed in retail stores, factories, and offices. It provides LTE or 5G mobile network coverage through a specific area.

For the computing platform on the MEC, the following constraints must be taken into account to place computing resources in a MEC station:

1) The platform must be compact in terms of physical and power consumption due to the limitation of power consumption and physical space available in the MEC station. (For example, a MEC station should be small enough to be carried by a person or run on battery power.)

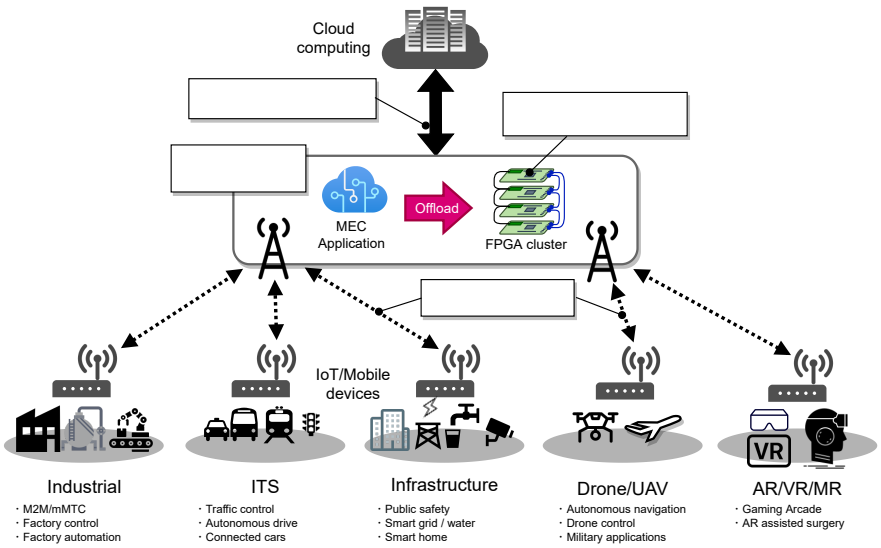


Figure 2.1: Multi-FPGA system on MEC

Figure 2.2: Example private 5G base station by Fujitsu [15]

2) The MEC station may be located in remote areas inaccessible by humans. Hence, the compute node needs to be monitored and managed remotely.

2.1.1 Potentials of Multi-FPGA systems on MEC

A multi-FPGA cluster has the following characteristics suitable for the computing platform on MEC for the following reasons:

1) Data processing is executed in hardware logic using FPGAs. It is possible to achieve low latency than software processing using CPUs. Therefore, it can be used in applications that require real-time performance.

2) Power-efficient hardware logic specialized for specific processing can be realized. Therefore, it consumes less energy than software processing with a CPU and can be installed in small wireless base stations with power constraints.

3) The required system size can be scaled by increasing or decreasing the number of computation nodes through clustering to optimize the footprint.

The Flow-in-Cloud (FiC) [13] is developing a multi-FPGA system for application to compute nodes for the MEC station in order to verify the advantages of multi-FPGA clusters. Chapter 3 describes the detailed system architecture and hardware.

2.2 Field Programmable Gate Array (FPGA)

An FPGA is a device that provides programmable logic circuits for a desired task or application. FPGAs allow users to freely program operations at the boolean logic level and have a high degree of flexibility to realize any desired logic circuit. With the advent of FPGAs, most digital computer architectures can be reproduced on FPGAs, making it easy to realize dedicated custom hardware for specific purposes.

Figure 2.3 shows the CMOS process technology used in FPGAs from representative FPGA vendors Xilinx (now AMD) and Altera (now Intel). Until the mid-2000s, FPGAs were slightly older than the latest process technologies. However, recent years have seen dramatic performance, integration, and price improvements by actively incorporating the latest process technologies.

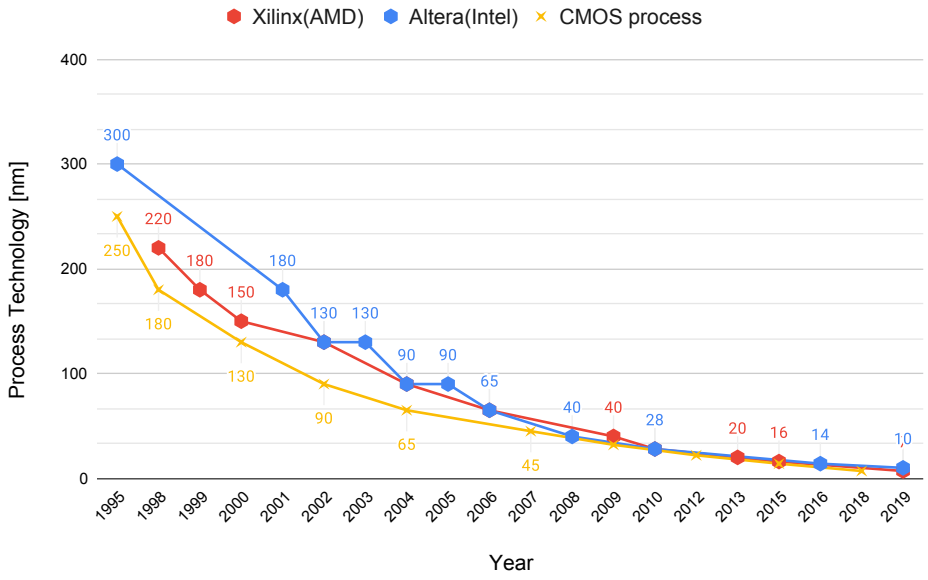


Figure 2.3: Xilinx and Altera's FPGA fabrication Process Rule Trend

Figure 2.4 and Figure 2.5 show the number of logic blocks and RAM capacity implemented in FPGAs. The scale of logic circuits that could be realized on FPGAs in the early days was small compared to ASICs (Application Specific Integrated Circuit). Therefore, these devices' main application was to realize digital circuits, such as substitutes for digital circuits using logic ICs or glue logic to connect digital circuits. However, as you can see in Figure 2.4 and Figure 2.5, FPGAs have been dramatically advanced in their available number of logic blocks and on-chip RAM capacities, according to the semiconductor process improvement.

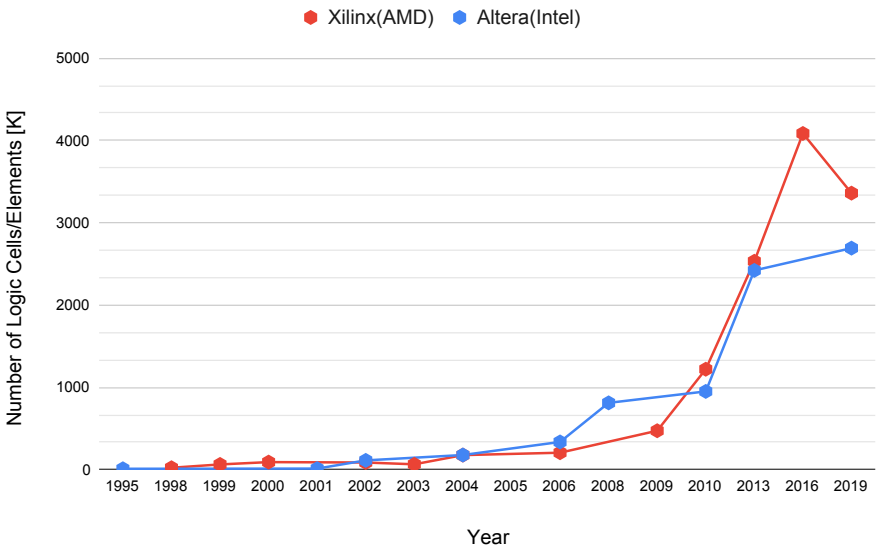


Figure 2.4: Xilinx and Altera's FPGA Logic Cells/Elements Trend

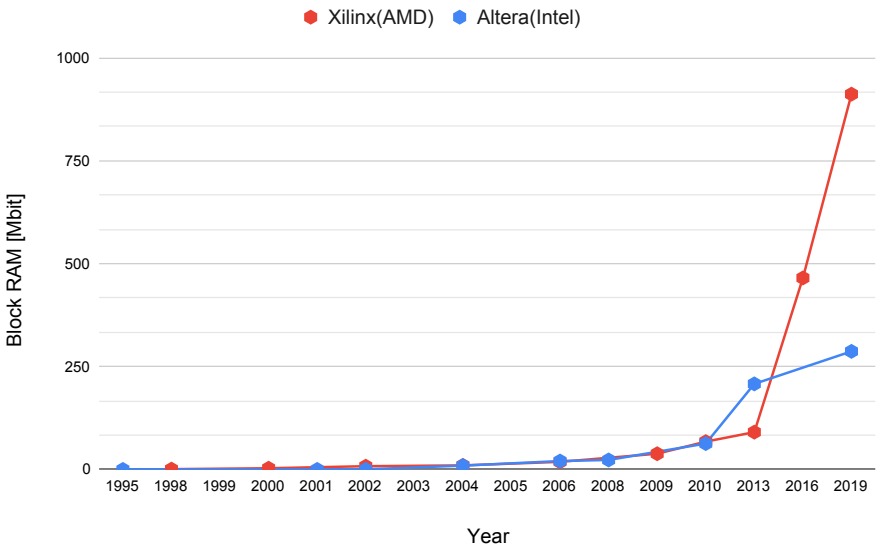


Figure 2.5: Xilinx and Altera's FPGA on-chip RAM Capacity Trend

The world's first FPGA was XC2064 (Figure 2.6) introduced by Xilinx in 1984, which contained 64 logic blocks called Configurable Logic Blocks (CLBs). FPGA users could implement logic circuits by setting up Boolean expressions for these 64

logic blocks and the connections between the logic blocks using a development tool called XACT [17].

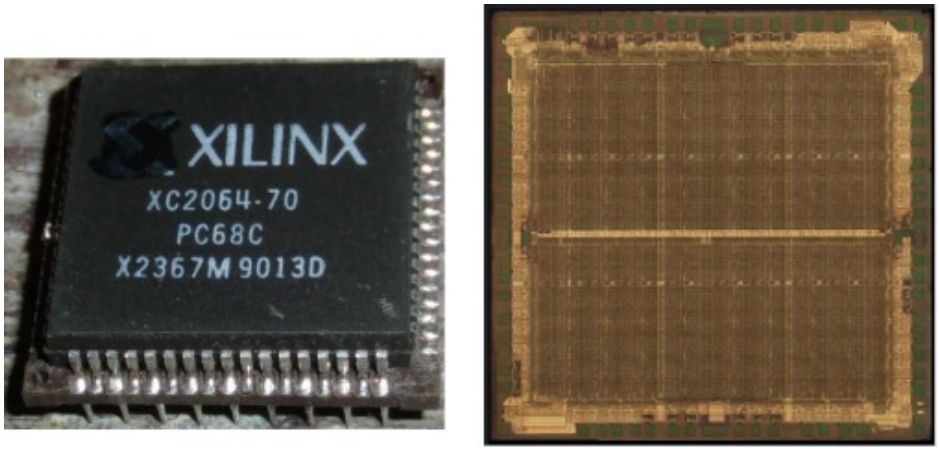


Figure 2.6: Xilinx's XC2064 Package and Die [18]

Later, in the 1990s, as the number of logic blocks increased according to Moore's Law, development tools (EDA tools) for FPGAs became more automated. FPGA users designed logic circuits using Hardware Description Language (HDL). EDA tools enabled it to automate logic block allocation (placement) and interconnection between them on FPGAs. FPGAs becomes more multifunctional in the 2000s. Various functional blocks and interfaces incorporated logic blocks to realize system LSIs using only FPGA.

This multifunctionality of FPGAs has brought various function blocks to the architecture, such as large on-chip memory capacity, multiply-and-add (DSP) suitable for digital signal processing operations, microprocessors, and multifunctional I/O and transceivers. Today, the number of logic blocks in the latest FPGAs has reached 4 million. Hardware advances in FPGAs are making steady progress in response to application requirements. Such as Intel Statrix 10 MX with high-speed and high-capacity on-chip memory (HBM) inside the FPGA enables to implementation of memory-required applications on FPGA, and Xilinx Versal with dedicated arithmetic circuit blocks to accelerate tensor calculations commonly is used in AI applications.

Development tools for FPGAs have also advanced to keep pace with the increasing sophistication of FPGA hardware. The latest development tools for FPGAs now support High-Level Synthesis (HLS) as a standard feature, enabling hardware logic generation from high-level languages such as C/C++. In addition, OpenCL, an open heterogeneous computing programming standard, supports FPGAs. Thus, an environment is being developed in which FPGAs can be used

as high-performance computation platforms.

2.2.1 FPGA Architecture

This section describes the basic architecture of FPGAs. The basic idea of FPGA architecture is using programmable logic blocks called Configurable Logic Block (CLB) (Xilinx), Adaptive Logic Module (ALM) or Logical Element (LE) (Altera) are connected with arranged vertically and horizontally via programmable connection elements called Connection Block (CB), and Switch Block (SB) to realize arbitrarily logic circuits by combining them. Figure 2.7 shows the basic structure of an island-style FPGA architecture.

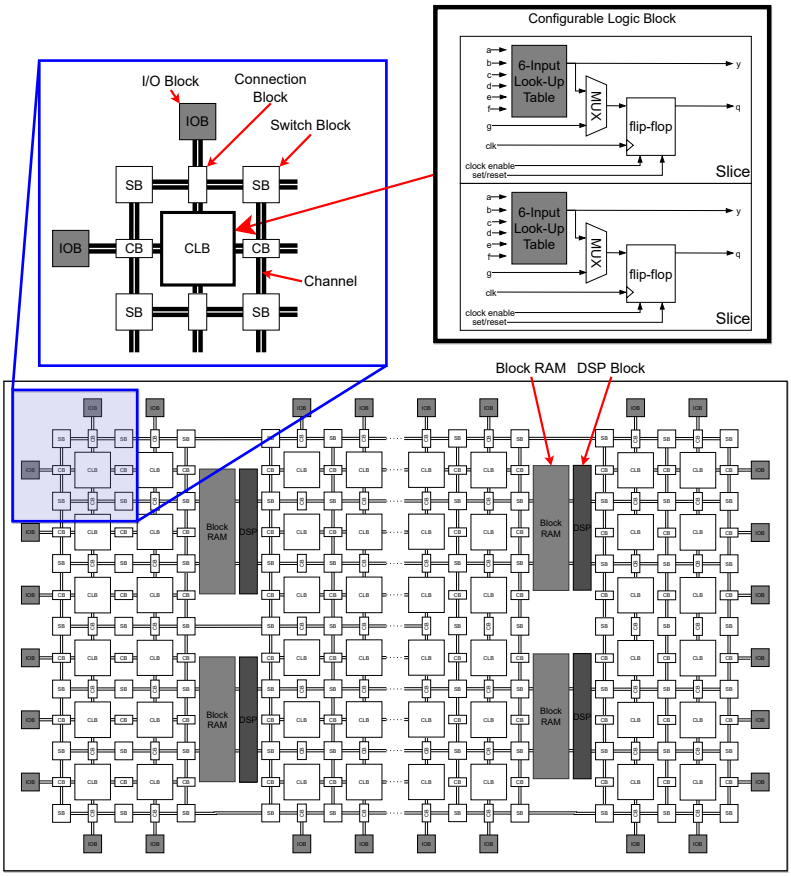


Figure 2.7: Typical Island-style FPGA Architecture

Configuration Logic Block (CLB)

A Configuration Logic Block (a CLB in Xilinx and an ALM or LE in Altera) is a programmable logic block unit that realizes a combinational circuit. The basic CLB configuration consists of multiple programmable logic blocks called Slices within the CLB.

Each Slice contains a multiplexer circuit that can implement an arbitrary N-input Boolean function called a look-up table (LUT). The number of LUT inputs is generally 4 or 6. The truth table of the LUT is held in SRAM, so the settings are volatile when power is lost. Therefore, it is necessary to perform a configuration to set the contents of the SRAM when power is turned on. The SRAM of the LUT is also used as a small-capacity SRAM. The output of the LUT is connected to a flip-flop or register via a programmable multiplexer. The circuit output can be synchronized to a clock to form a synchronous circuit.

Connection Block (CB)

The Connection Block (CB) is a programmable switch element that connects the CLB and the routing channel.

Switch Block (SB)

Switch blocks (SBs) are programmable switch elements located in routing channels that intersect vertically and horizontally in the FPGA. Together with CBs, they set the paths to form connections between blocks such as CLBs.

Input/Output Block (IOB)

Input/Output Block (IOB) is a functional block that provides an interface between the FPGA's internal circuitry and the package pins. Each IOB can be programmatically set to a signal level such as TTL (1.2V) or CMOS (2.5V) and the direction of input/output such as INPUT, OUTPUT, or Bi-directional.

DSP

A Digital Signal Processor (DSP) is a functional block that provides a variety of functions commonly used in signal processing, such as multiply-add, multiply-accumulate, counter functions, and bit-width boolean operations. These operations can also be performed by combining CLBs, but the number of CLBs required is large and inefficient. Therefore, a DSP is placed as a dedicated functional block to perform these operations. Xilinx's FPGAs typically include a 48 bits DSP module called DSP48E blocks. Intel's Arria 10 [19] has a floating-point arithmetic

unit built into the DSP module, enabling the efficient implementation of applications that require numerical calculations.

Block RAM

Block RAM is a functional block that provides random access memory (RAM). The SRAM in the LUT of each CLB can also be used as a distributed RAM. However, the capacity is small and inefficient to be used as a RAM to store large capacity data, so the BRAM is provided as a dedicated RAM block. Recent Xilinx FPGAs typically have a capacity of 36 kbits per block. Each BRAM is implemented as dual-port RAM. In FPGA logic design, the typical use of BRAM is to implement FIFOs and data queues.

Also, Xilinx's UltraScale+ series has the UltraRAM (URAM) block [20], which provides large-capacity RAM. UltraScale+'s URAM is implemented as a 72 bits wide dual-port RAM with a capacity of 288 kbits per block. URAM can basically be used in the same way as BRAM. But some FPGAs have URAM with a capacity as large as 360 Mbits, which is very useful for applications that require fast and large-capacity RAM.

Embedded Microprocessor

Some FPGAs have a microprocessor as a functional block. These are system-on-chip (SoC)-type devices that implement the FPGA fabric and microprocessor (CPU) on the same package. Xilinx's Zynq-7000 series [21] released in 2012 is implementing Xilinx's Virtex 7 medium-sized FPGA with a dual-core 1 GHz ARM Cortex-A9 processor on a single chip. The FPGA fabric part of Zynq is called Programmable Logic (PL), and the CPU part is called Processing System (PS). The PL and PS are connected by the Advanced eXtensible Interface (AXI) bus, which allows the PS's ARM processor to access the logic on the PL's FPGA.

Also, Intel announced Xeon Scalable processor with integrated FPGA (Xeon Gold 6138P) [22] in 2018. This CPU is Intel's Skylake-SP generation 28-core Xeon processor (14nm process) of Intel's Skylake-SP generation and their Arria 10 GX 1150 FPGA (20nm process) on a single chip. Xeon and Arria 10 have one Ultra Path Link (UPL) with 9.6 GT/s bandwidth on the chip, and two PCIe 3.0 x8 connections allow access to the FPGA directly from the processor via a high-bandwidth, low-latency bus.

SoC-type FPGAs can realize applications that combine the high flexibility of CPUs with the high processing efficiency of FPGAs, especially on embedded applications. For example, highly efficient systems can be easily built by offloading inefficient CPU processing to FPGAs. Thus, SoC-type FPGAs have characteristics that make them suitable platforms for MEC and edge computing.

2.2.2 FPGA Application Development

This subsection describes development techniques for implementing applications on FPGAs.

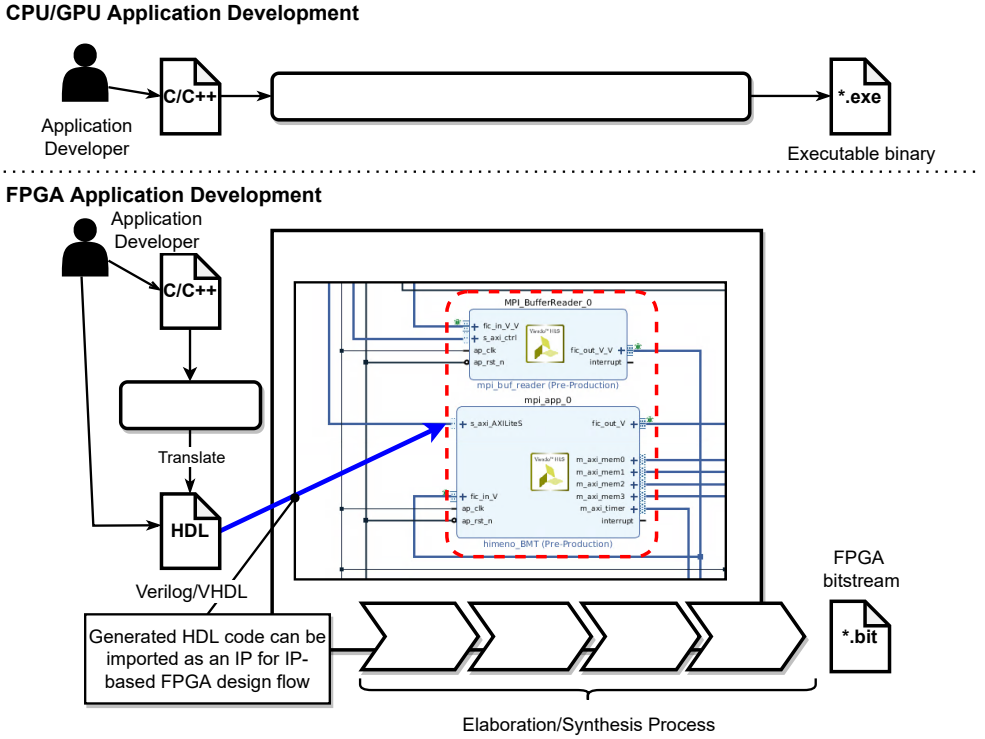


Figure 2.8: CPU/GPU and FPGA's Application Development Flow

Figure 2.8 shows the comparison of application development flow between CPU/GPU and FPGA. Typically, the application development flow for CPUs and GPUs is implemented by a compiler. The compiler converts the programming language description written by the application designer into a binary code that the CPU or GPU can execute. On the other hand, in general FPGA application development, applications are implemented as hardware logic using HDL (Hardware Description Language) such as Verilog and VHDL. Implemented hardware logic description is converted to the target FPGA device application using the Electric Design Automation (EDA) tool provided by the FPGA vendor in the following steps:

Logic synthesis

Logic synthesis synthesizes the logic circuit for the target FPGA device from the HDL description. This logic synthesis process involves the interpretation and optimization (elaboration) of the HDL description into logic circuits.

Technology mapping

After the *Logic synthesis*, *Technology mapping* is performed. This process maps the generated logic circuits using target FPGA elements such as logic and functional blocks.

Placement and Routing

After the *Technology mapping*, *Placement and Routing* is performed to connect the mapped logic circuit to the FPGA elements using routing resources, such as CBs, SBs, and channels.

Timing analysis and Design rule check (DRC)

Timing analysis and *DRC* analyze whether the implemented logic circuit meets physical restrictions such as operational clock frequency.

Bitstream generation and Programming

Finally, the FPGA application is implemented by generating the FPGA programming bitstream file. FPGA user programs FPGA device with the bitstream before use.

IP-based Logic Design

Although IP-based design is available on the FPGA logic design, application designers need to implement hardware in HDL when implementing a specific application logic that cannot achieve with IPs. It is still presenting programmability challenges in FPGA logic design. To address the issue, FPGA vendor has provided a High-Level Synthesis (HLS) FPGA development environment since around 2013. The HLS can be expected to improve the programmability of FPGA applications. Although the current HLS has some limitations, it frees application designers from low-level hardware implementation with HDL. Instead, they can implement FPGA hardware in high-level languages like C/C++ and System-C.

The typical HLS compiler, such as Xilinx's Vivado HLS and Intel's (Altera) HLS Compiler, converts C/C++ code into HDL with Verilog or VHDL. This converted code can be imported as an IP in the FPGA design tool. HLS allows the implementation of FPGA applications by application designers who do not have hardware design knowledge in HDL. Thus, the HLS is one of the solutions for FPGA programmability challenges. However, the FPGA design flow with HLS simply extends the conventional FPGA design flow. There is still not eliminated the complexity of the FPGA design process, such as the time-consuming logic synthesis and the effort required in debugging and verification.

Overlays

An overlay is a method of implementing an application on FPGA by implementing a specialized architecture for specific processing on FPGA. In this method, the application designer implements an application for the implemented architecture on FPGA. This specific architecture implemented on the FPGA is called the *Overlay*.

Since the overlay virtualizes the FPGA device, application designers can design FPGA applications with less effort than conventional FPGA hardware design flow, and it can eliminate the complexity of the application development on FPGA. A typical example of an Overlay is a soft-core processor implemented on an FPGA. FPGA vendor often provides soft-core processors as an IP. For example, Xilinx's MicroBraze [23] and OpenRISC [24] are well known. Using these soft-core processors, the application designer can easily implement their application on FPGA while utilizing the rich functional blocks of FPGAs. On the other hand, the efficiency of the application is dominated by the efficiency of the overlay used.

For Multi-FPGA Environment

Application development on multi-FPGA systems that use multiple FPGAs is fundamentally challenging because typical FPGA design tools target a single FPGA. Therefore, two typical methods can be considered to design applications on multi-FPGA systems. The first option is using FPGAs indirectly by overlay, then implementing multi-node applications on the overlay. Another option is using a heterogeneous computing framework that can support multiple FPGAs such as OpenCL and Xilinx SDAccel.

For the specific multi-FPGA architecture, application development environments that support parallel processing with multiple FPGAs have been proposed. For example, the related research [25] and [26] proposed a multi-node FPGA application development environment for specific multi-FPGA architecture. These environments implement special code converters to extract Message Passing Interface (MPI) operations from the user code and generate HLS code for each FPGA node. These approaches deserve attention because they reduce the complexity of application development on a multi-FPGA environment while taking advantage of FPGAs' high parallelism and efficiency rather than using abstractions such as overlay virtualization and frameworks.

2.3 FPGA for General Computing

The essential advantage of FPGAs over general-purpose CPUs and GPUs is their potential to achieve high performance and efficiency through purpose optimized hardware-based data processing. In recent years, there has been a strong interest

in computing platforms that can process large amounts of data with higher efficiency due to the growing demand for big data processing and AI processing. Therefore, there have been attempts to utilize FPGAs in various computing domains.

From the next section onward, we will focus on specific computing systems that utilize FPGAs, with particular attention to notable examples and those mainly related to this study.

2.3.1 Big-Data Analysis

Microsoft Catapult v1 and v2

Microsoft Catapult v1 [27] is known as a notable example of large-scale FPGA use in the back-end of a commercial Web service. The system is a large-scale deployment of FPGA-equipped computing nodes in a data center to accelerate the Page Ranking process of Microsoft's Bing search engine.

Catapult v1 system achieves a 1.95x throughput and 28% processing latency reduction than the software-only PageRank implementation. The v1 system consists of 17 racks of compute nodes, each equipped with a 12-core SandyBridge generation Intel Xeon server and Stratix V FPGAs, Intel's high-end FPGAs, for a total of 1632 nodes in a large-scale system. Each FPGA mounted on the server constitutes a 6x8 torus network that enables direct communication between FPGAs on other compute nodes using a 10Gbps SAS cable and SerialLite III (SL3) [28] protocol. PageRank is implemented to perform pipeline processing on 8 FPGA nodes using this local FPGA network. Each FPGA board is equipped with 8GB of DDR3 SDRAM and connected to the server of the compute node via PCIe Gen3 x8. The FPGA design separates the Role, which implements the processing kernel part, and the Shell part, which implements other fundamental functions, such as PCIe connection with the host server and SL3 switches. Therefore, the Role part can be replaced according to the application, enabling a general-purpose design that can be used for applications other than PageRank.

Catapult v2 [29] is the successor system developed based on the results of the v1 system, which is a more extensive system than the v1 system with 5,760 nodes. In order to solve the scalability and flexibility issues identified in the v1 system, the v2 system allows servers and FPGAs to be interconnected to a standard Ethernet network in the data center instead of the rack-closed FPGA network used in the v1 system. For this architecture, the v2 system uses a high-density blade server with an FPGA daughter board with Intel Stratix V D5 high-end FPGA and 4GB of DDR4 SDRAM connected to the PC server via two PCIe Gen3 x8. Another architectural uniqueness of the v2 system is that the two 40GbE interfaces on the FPGA board and the PC server's NIC is directly connected to one port on the FPGA board, and the other port on the FPGA board is connected to the rack's top-of-rack (TOR) switch. PC server

communication always goes through the programmable switch implemented on the FPGA; FPGA can provide offloading features intervening in the middle of the PC server's communication path, such as network encryption and compression. Since the FPGAs are connected to the PC server locally via PCIe and the data center network via 40GbE, a compute node can offload a process to a local FPGA connected via PCIe and offload to remote FPGAs in the other compute nodes connected via 40GbE. Therefore, in the v2 system, PC servers and FPGAs on compute nodes can be used as independent computing resources, and the architecture can improve operational efficiency in the data center.

2.3.2 Artificial Intelligence

Microsoft Project Brainwave

Microsoft's Project Brainwave [30] uses the Catapult v2 architecture deployed in data centers to build an architecture that can execute deep neural network (DNN) models at high speed and low latency. Brainwave's architecture is based on the Catapult v2 architecture with a soft-core accelerator called Brainwave NPU deployed as an overlay on FPGA. The Brainwave NPU architecture is based on the Neural Functional Unit (NFU), an accelerator core for matrix-vector operations with a dedicated instruction set (ISA) for DNN model execution, and Altera's Nios soft-core processor IP for controlling the NFUs. Brainwave achieves more than ten times the DNN model execution performance and less than one-tenth the processing latency of CPU-based implementations.

2.3.3 Cloud Computing

Galapagos

Galapagos [31] is a project to build a large-scale CPU and FPGA heterogeneous computing environment for use in cloud computing is being developed at the University of Toronto. Galapagos uses OpenStack [32], an open-source orchestration tool, to manage a multi-FPGA cluster system built using x86 servers and compute nodes connected via PCIe or SoC-type FPGA nodes with ARM. The FPGA hardware is virtualized using Galapagos Hypervisor, a Shell that contains essential functions such as communication interfaces with outside FPGAs and local DRAM controllers. FPGA applications are implemented as Roles on this hypervisor. Users can define applications and networks that combine CPUs and FPGAs, and Galapagos selects the CPUs and FPGAs available in the cluster and connects them according to the resource and network requirements defined in the application to provide a heterogeneous computing environment combining CPUs and FPGAs.

Amazon EC2 F1 Instance

Amazon EC2 F1 Instance [33] is an FPGA-equipped instance available on Amazon's EC2 cloud computing service. The F1 instance is a breakthrough in that cloud computing and FPGA-based acceleration can work seamlessly. The F1 instance consists of a single server node with an Intel Broadwell E5 2686 v4 processor and 976 GB of memory, up to eight Xilinx UltraScale+ VU9P FPGAs connected via PCIe x16. The eight FPGAs are provided with 64 GB DDR4 local memory with ECC and connected by a 400 Gbps bidirectional ring bus, allowing users to implement their protocols on the FPGAs to connect them in a high-bandwidth, low-latency network [34].

IBM cloudFPGA platform

IBM's cloudFPGA [35] platform is a high-density FPGA cluster system for cloud computing developed by IBM Research Zurich. The unique feature of this system is that each FPGA node on the cloudFPGA platform is directly connected to the data center network; unlikely typical FPGA systems that use FPGAs connected to PC servers via internal buses such as PCIe. A high-density FPGA cluster is built in a 2U chassis containing two backplanes with 40GbE Ethernet switches, each containing 32 FPGA boards with Xilinx Kintex UltraScale XCKU060 equipped with 10GbE. The FPGA design on the cloudFPGA architecture uses Shell and Role design approach. The Shell is a pre-designed FPGA design for implementing fundamental functions to use FPGA, and the Role is a user-designed FPGA design for an application. The Shell of the cloudFPGA implemented the Ethernet, TCP/IP stacks, and HTTP server for RESTful API provider to manage the FPGA node. Since the FPGA nodes are directly exposed to the data center network, the applications that want to use FPGA acceleration can access the FPGA remotely through HTTP access by RESTful APIs. The cloudFPGA architecture and its management system have many similarities with FiC and M-KUBOS, which are the target of this research. However, the cloudFPGA system aims to build a large-scale FPGA infrastructure in a data center, and its purpose and scale are different.

2.3.4 Edge Computing

Zedwulf

Zedwulf [36] was developed at the Nanyang Technological University (NTU) Singapore. It is a small-scale multi-FPGA system using Xilinx's Zynq MP-SoC FPGA. The architecture consisted of 32 Zynq-Z7020 FPGA nodes with a 1 Gbps Ethernet network. Each node has Xilinx's Zynq-Z7020 MPSoC, an SoC-type FPGA with dual-core 32-bit ARMv7 architecture and FPGA fabric, and 512 MB onboard

DRAM. The communication between the FPGA nodes is done by ARM optimized version of Open MPI or MPICH, an open-source MPI implementation. Each node installed the MPI and Xillinux 1.3 on Zynq's PS, enabling them to offload the application kernel implemented on the PL from PS. With the 32-node graphing problem benchmark, the Zedwulf cluster system achieved 1.5x the performance of the Intel Core i7-4770K (4 cores, 8 threads) with nearly the same power efficiency.

3

Target Systems

This chapter introduces target stand-alone multi-FPGA systems in this thesis. The chapter is organized as follows: Section 3.1 is the preface of the chapter. Section 3.2 introduces the architecture of the FiC multi-FPGA cluster. Section 3.3 presents the architecture of the M-KUBOS multi-FPGA cluster. Section 3.4 explains the STDM network implementation used in the inter-FPGA network for FiC and M-KUBOS clusters.

3.1 Multi-FPGA System for MEC Environment

In the edge environments with power restrictions and space constraints such as MEC, introducing conventional typical server-based FPGA systems is inefficient. This research focused on the stand-alone multi-FPGA systems for MEC that realize a small footprint and system scale flexibility according to application demands on edge computing. The stand-alone multi-FPGA system constructs independent FPGA nodes with an inter-FPGA network to connect nodes. Each node can be used as an independent FPGA node, and it also can be used as a clustered multi-FPGA system with the inter-FPGA network.

The FiC cluster and M-KUBOS clusters have been developed for proof-of-concept of the stand-alone multi-FPGA systems for MEC. The FiC cluster is a multi-FPGA system built with cost-efficient FPGA nodes that combine Xilinx's mid-range FPGA Kintex UltraScale and Raspberry Pi 3B single-board computer. And the successor M-KUBOS cluster is a multi-FPGA system using Zynq UltraScale+ MPSoC, addressing the weaknesses of the FiC cluster. Both systems are

equipped with an inexpensive, high-speed inter-FPGA network directly connecting FPGAs. This inter-FPGA network can flexibly change the network's topology and the number of connected nodes according to the system requirements and scale of the application demands on the edge environment. The following sections describe the architecture and hardware systems of the FiC and M-KUBOS systems.

3.2 Flow-in-Cloud (FiC) Multi-FPGA Cluster

The prototype Flow-in-Cloud (FiC) multi-FPGA cluster consists of multiple *FiC-SW* FPGA boards connected by an inter-FPGA network that uses high-speed serial links illustrated in Figure 3.1. The system is the first generation of the stand-alone multi-FPGA system that we developed to apply to the MEC environment. In the FiC system, four FiC-SW nodes are mounted in a 4U height dedicated cabinet that can fit on a standard 19-inch server rack.

Figure 3.2 shows current 24 FiC-SW nodes configuration diagram. The nodes are connected to form a 6x4 torus network, for example. Each of the nodes is connected with cost-efficient *Firefly™ Micro Flyover™* flat cables by Samtec [37] (Figure 3.3). Each flat cable offers four bidirectional lanes to one destination, so this design restricts the four lanes connected to the same destination. Hereafter, we call such a set of four bidirectional lanes a *channel*. The high-speed network between nodes uses a Static Time Division Multiplexing (STDM) data communication scheme to keep a constant latency and bandwidth between multiple communications [38]. Each FiC-SW node equips Xilinx's mid-range FPGA Kintex UltraScale and cost-efficient Raspberry Pi3 single-board computer (SBC). Each node works as an independent FPGA node, and FPGAs are directly connected with the inter-FPGA network. We call this multi-FPGA architecture style for *stand-alone multi-FPGA* architecture.

The current prototype FiC cluster has one cluster management server connected to the inter-FPGA network of the FiC cluster nodes. The management server equips a Xilinx KCU1500 PCIe attached FPGA board. This FPGA board is connected to the server by PCI Express Gen3.0 x8 and is connected to the cluster via two serial channels (4 lanes x 2 channels) for data exchange with the FiC cluster.

Figure 3.1: Prototype 24 nodes FiC Cluster

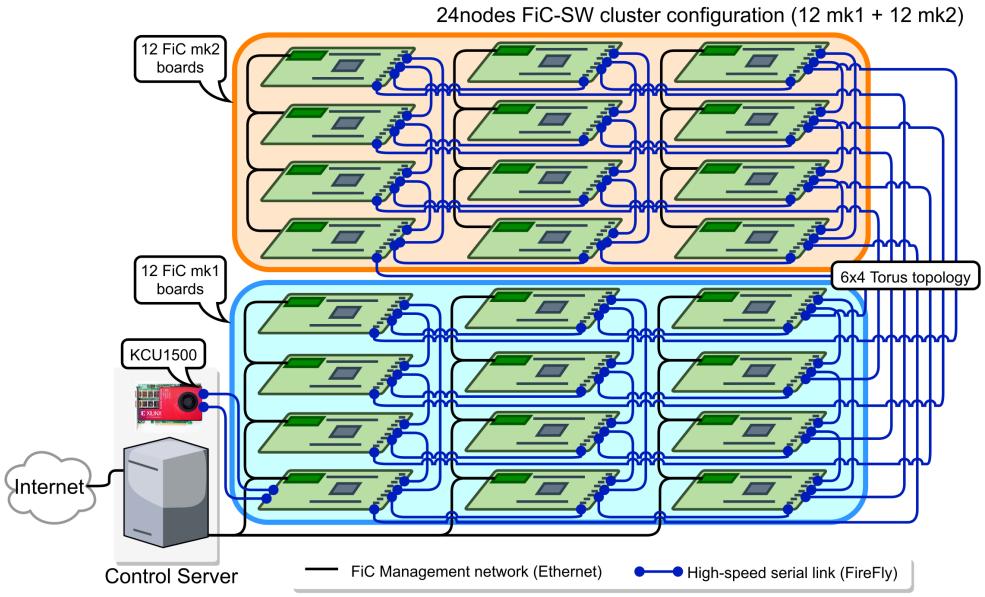


Figure 3.2: Prototype 24 nodes FiC Cluster Diagram

3.2.1 FiC-SW Prototype FPGA Board

Figure 3.4 shows the FiC-SW prototype board. We employ a mid-range Xilinx's Kintex Ultrascale XCKU095 FPGA for the first generation and XCKU115 FPGA for the second generation of the FiC-SW board. Both FPGAs support up to 64 GTH high-speed serial transceivers for the inter-FPGA network. Although the maximum bandwidth of the GTH serial ports is 16.3 Gbps, we regulated the transfer speed to 9.9 Gbps to avoid implementation complexity. The inter-FPGA network connectivity delivers up to 32 lanes (4 lanes x 8 channels) per board at the current design to provide enough bandwidth. It can support various network topologies connecting hundreds of cluster nodes.

Each board equips two 16 GB DDR4 SDRAM for storing data of FPGA computation, and a Raspberry Pi3 single-board computer is mounted on the board as an FPGA node controller. This RPi3 and FPGA are connected via GPIO port to configure and control the FPGA from the RPi3. Table 3.1 summarizes the detailed hardware specifications of the current 24 nodes FiC cluster and FiC-SW FPGA board.

Table 3.1: Specifications of current 24 nodes FiC cluster

System Scale	24 FiC-SW board and an I/O board (KCU1550)
FPGA	Kintex UltraScale XCKU095-FFVB2104 (ver 1) XCKU115-FFVB2104 (ver 2)
Clock Freq.	100 MHz
STDM Switch	four 9x9 at maximum
Serial Links Effective Speed Total Exchange Bandwidth Total Available Throughput	32 channels bundled into 8 lanes 8.5 Gbps (9.9 Gbps at a link) 272 Gbps 34 GBps
Pass through Latency Max Latency of the System	550μsec 1710μsec
DRAM	16Gbit DDR4 DRAM (200 MHz) x 2
On-board Controller	Raspberry Pi3 Model B (BCM2837 ARM Cortex-A53 Quad 1.2 GHz)

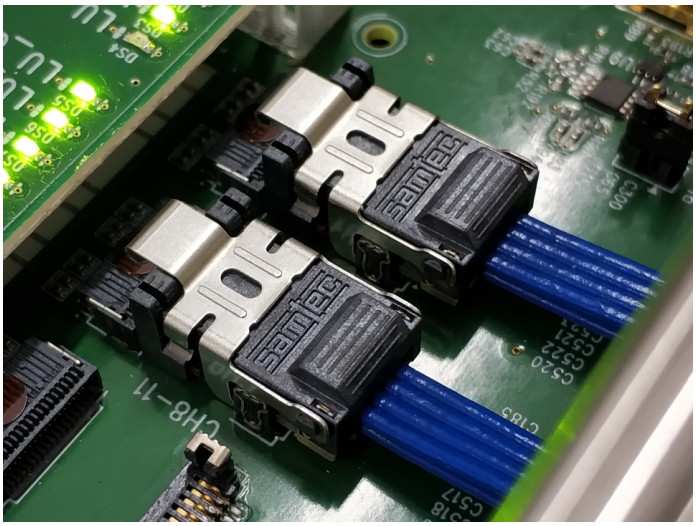


Figure 3.3: Samtec Firefly Flat Cable

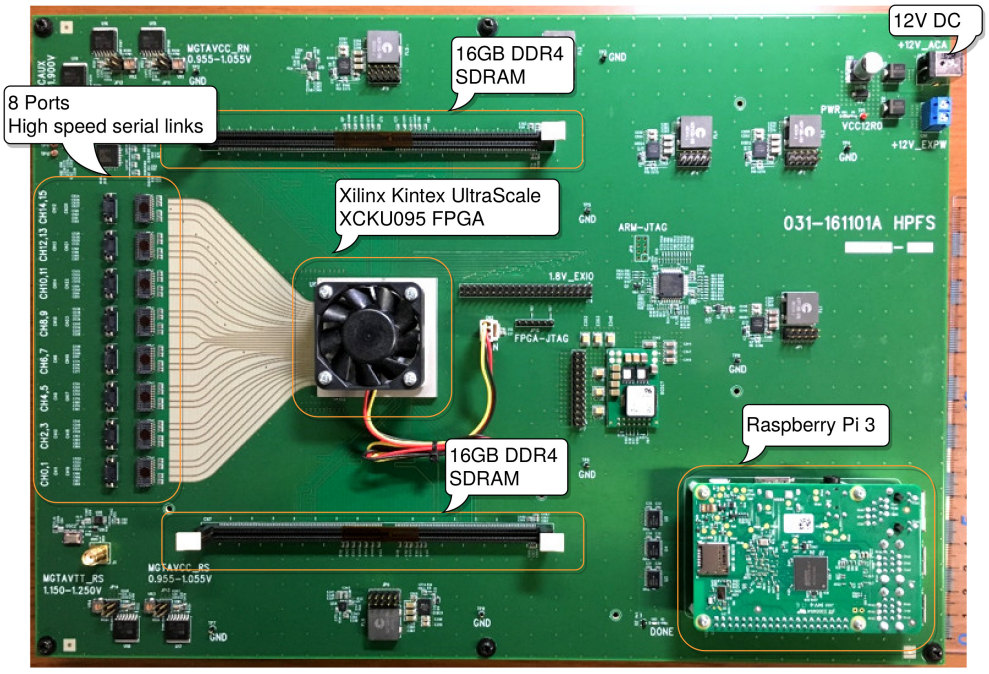


Figure 3.4: FiC-SW FPGA board

3.2.2 FPGA Logic Design on the FiC-SW board

Figure 3.5 shows FPGA logic design diagram of the FiC-SW board. To design the FPGA logic for the FiC-SW board, we employed *Shell and Role* FPGA design approach. The Shell is a static and pre-redesigned region of the FPGA design, which contains fundamental logic for user application. The Role is a dynamic region for user application logic implementation.

Figure. 3.6 shows *Shell and Role* FPGA design flow of the FiC-SW. The application designer uses a Shell for the base FPGA design and implements application logic into the Role. Then, it imports the Role to the Shell design as an IP to connect them on the Vivado's block design. This FPGA design flow allows the application user rapidly design FPGA logic and improve productivity. The Shell is provided as a pre-designed Vivado project, and the Role can be designed with IP-based FPGA design flow using HDL (Verilog/VHDL) or HLS with C/C++ (Xilinx Vivado HLS [39]). The Shell of the current FiC-SW contains Xilinx's Aurora serial transceivers IP [40] for GTH serial link, STDM switch for inter-FPGA network, DRAM controller IP (Xilinx's MIG [41]) for onboard DDR4 DRAM, and GPIO interface IP for communication to the onboard Raspberry Pi 3. Currently, the Shell provides four in and out ports of the STDM switch to the Role, and the Role can communicate with other FPGA nodes through this STDM switch. Each STDM switch port is implemented as a 170 bits AXI stream bus interface on the block design.

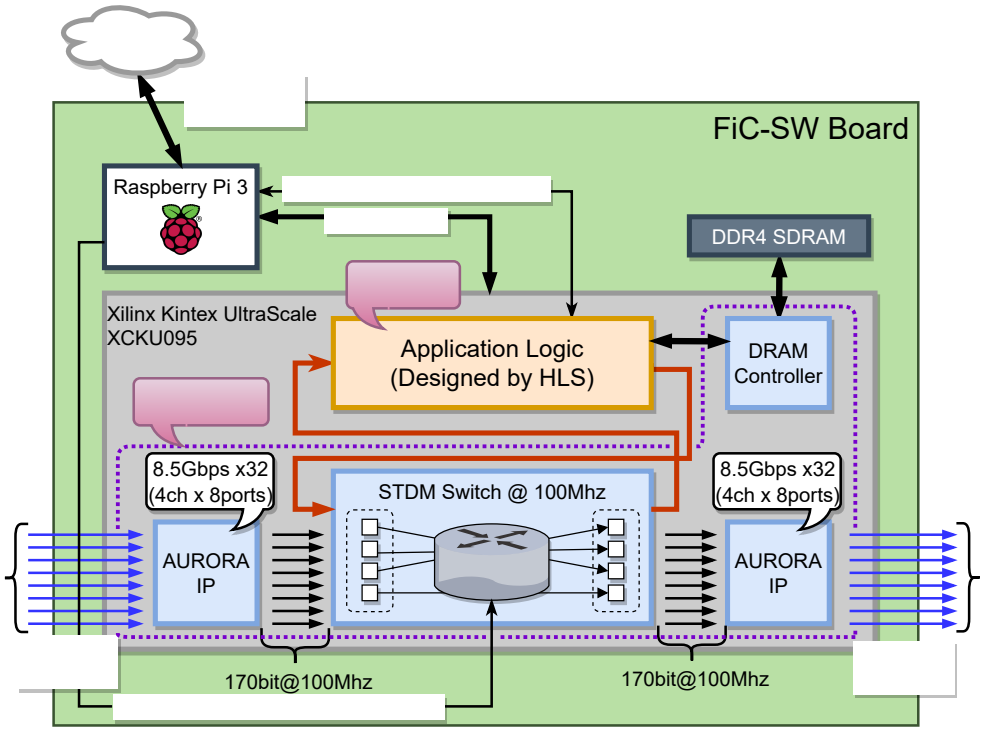


Figure 3.5: Block Diagram of the FiC-SW's FPGA Design

3.3 M-KUBOS Multi-FPGA Cluster

3.3.1 Introduction of M-KUBOS Multi-FPGA Cluster

The M-KUBOS cluster system is a successor stand-alone multi-FPGA system to the FiC cluster introduced in Section 3.2. Figure 3.8 shows the current six nodes M-KUBOS cluster system. The M-KUBOS cluster consists of the M-KUBOS board and a high-speed inter-FPGA network. The M-KUBOS board employs Zynq UltraScale+ MPSoC XCZU19EG, the current highest-ranked System-on-Chip (SoC) FPGA from Xilinx. Currently, the cluster is constructed with a simple ring topology with six M-KUBOS nodes. The interconnect between M-KUBOS nodes uses the STDM network the same as the FiC cluster. Therefore, the M-KUBOS and FiC clusters can operate each other for system expansion. In the SoC FPGA, CPU and FPGA are tightly connected by a high-speed and large capacity SoC bus, which can avoid communication capacity drawbacks on the FiC-SW board architecture. An application environment that can easily access FPGA is realized by employing SoC FPGA for the FPGA nodes.

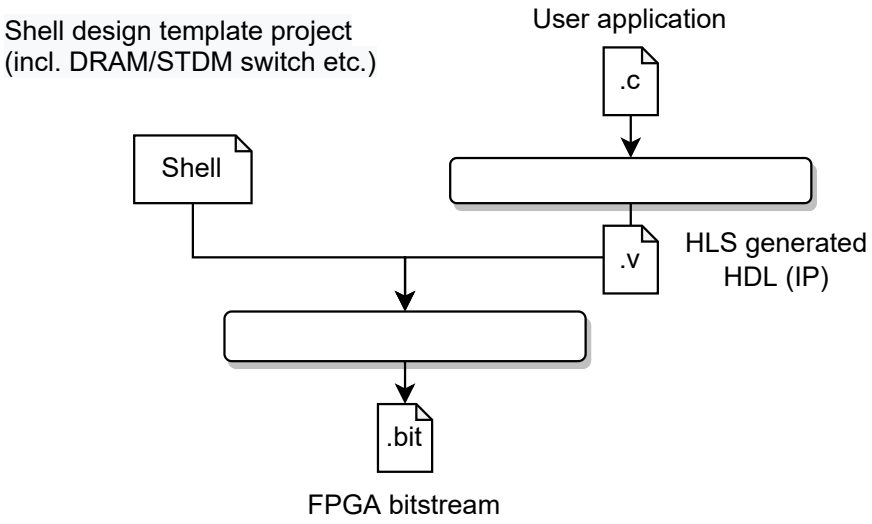


Figure 3.6: FPGA Design Flow on the FiC-SW

3.3.2 M-KUBOS Zynq FPGA board

Figure 3.9 shows the M-KUBOS board. The M-KUBOS board is a Zynq SoC FPGA board that can build FPGA clusters, commercialized by PALTEK [42] based on the concept of FiC [43]. Table 3.2 shows the detailed hardware specifications of the M-KUBOS board. The Zynq UltraScale+ MPSoC XCZU19EG has a processing system (PS) based on ARM Cortex A53 Quad-core and Cortex R5 Quad-core. This PS is coupled with the PL (Processing Logic) of the UltraScale+ FPGA fabric. The PL (Processing Logic) has 1143K logic cells, 70.6 Mb BRAM/URAM, and 1948 DSPs. The scale of PL resources is roughly equivalent to the Kintex UltraScale FPGA used in the FiC-SW board. The board equips two sets of onboard DDR4 SDRAM (4 GB): the one is accessible from both PS and PL, and the other is PL-only. As shown in Figure 3.2, the board is equipped with a variety of standard I/Os to support a wide range of applications. For the inter-FPGA network, the board equips eight sets of GTH and four sets of GTY high-speed serial links. This serial link has compatibility with the FiC cluster.

3.3.3 Introducing PYNQ to M-KUBOS

For the cluster node management of the stand-alone multi-FPGA systems, an individual FPGA node management capability is required. For this purpose, the M-KUBOS has employed PYNQ (Python productivity for Zynq) [44] a Linux distribution for the PS of Zynq SoC. PYNQ is a Linux distribution based on Ubuntu

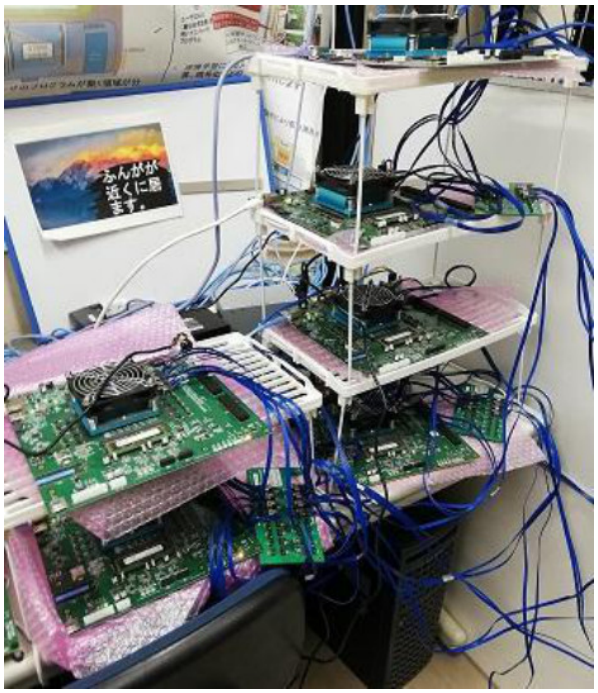


Figure 3.7: M-KUBOS Cluster

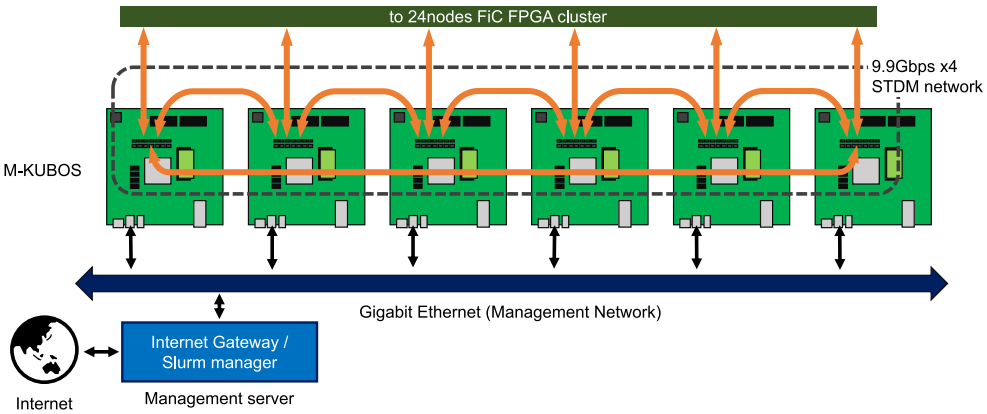


Figure 3.8: Six nodes M-KUBOS Cluster Diagram

which includes device drivers and Python libraries for controlling the PL of Zynq SoC. By introducing the PYNQ environment, the PL can be controlled and used from Python scripts running on the PS. For example, the configuration information of the PL can be set as an *overlay* from a Python script on the PS. The logic circuits on the PL can be accessed using MMIO (Memory Mapped IO) or DMA (Direct

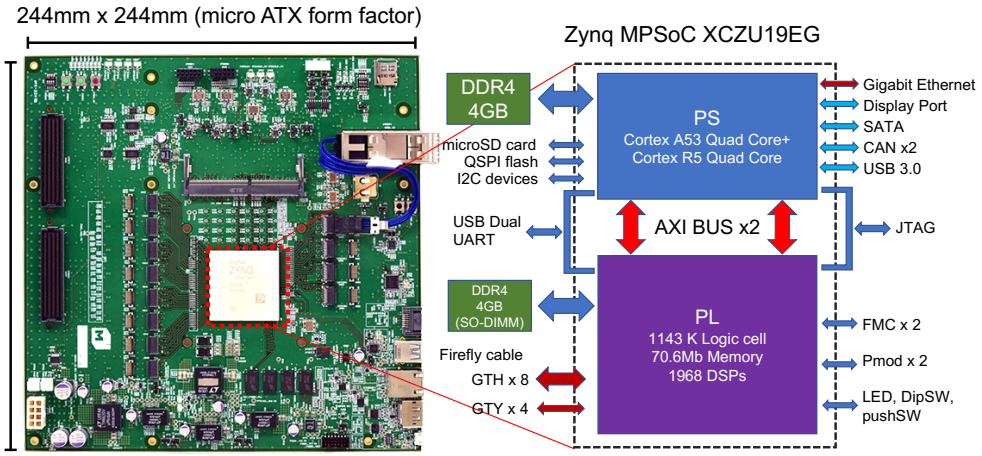


Figure 3.9: M-KUBOS board

Table 3.2: Detailed Specification of the M-KUBOS board

Item	Specification
Form Factor	244mm x 244mm (microATX)
FPGA	XCZU19EG-2FFVC1760
Memory	PS: 4 GB DDR4-2400 PL: 1x 4 GB DDR4-2400 SODIMM Socket
I/O	4x GTY 4TX/4RX (max 28. 125 Gbps) 4x GTH 8TX (max 16. 3 Gbps) 4x GTH 8RX (max 16. 3 Gbps) USB3.0 $\times$ 1 USB-UART $\times$ 1 1 Gb Ether(RJ45) DisplayPort 1.2

Memory Access) from the PS by integrated device drivers on the PYNQ. The PL logic can be designed using standard IP-based block design flow for FPGA design by Xilinx Vivado. Introducing the PYNQ environment for the M-KUBOS improves FPGA node manageability and enhances the accessibility of the FPGA from the applications.

3.3.4 FPGA Logic Design on M-KUBOS board

The FPGA logic design flow of the M-KUBOS board uses *Shell and Role* design flow approach, the same as the FiC-SW board. The difference between the FiC-SW

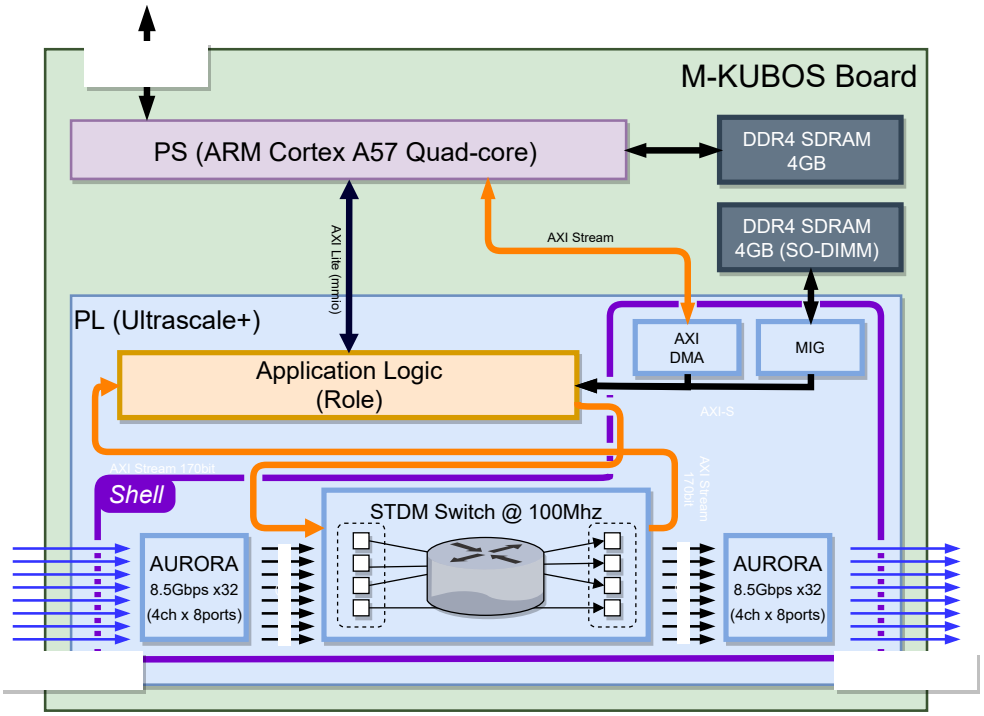


Figure 3.10: Block Diagram of M-KUBOS Board

board's design and the M-KUBOS board's design, the M-KUBOS uses Zynq SoC FPGA and equips two onboard DRAMs. In the M-KUBOS board, one DRAM is connected to PS and PL and shared with them. PS uses this DRAM for running Linux. Another DRAM is dedicated to the PL. The application user can choose a few design options to exchange data to/from PS and PL with these DRAMs. Table 3.3 shows the currently available Shell for user application design for the M-KUBOS board.

Figure 3.11 shows the resource utilization of each Shell. Overall, Shell's resource overhead is within 10%. This leaves sufficient resources for user-designed IP. However, in M and F, due to the large STDM switch buffers, the BRAM resource usage is as large as 20% and 25%, thus leaving room for BRAM resource optimization of the STDM switch design. The amount of BRAM consumed by the current Shells may not cause an extreme shortage of BRAM resources in the practical implementation of user designs. Moreover, the URAMs available in UltraScale+ generation FPGAs are not used by the Shell. Therefore, the all amount URAM resources can be used in the user design.

Table 3.3: Shell design template for M-KUBOS logic design

Shell Type	Description
S (Single)	The simplest shell with only the connection interface to the PS, including DMA controller, control registers
SwD (Single with DRAM)	**S** with an additional DRAM controller (MIG)
M (Multiple)	**S** plus Network Interface IP (Aurora) Multi-board connection is possible
F (Fully embedded)	Shell equipped with all of the above. The most resource-intensive

.

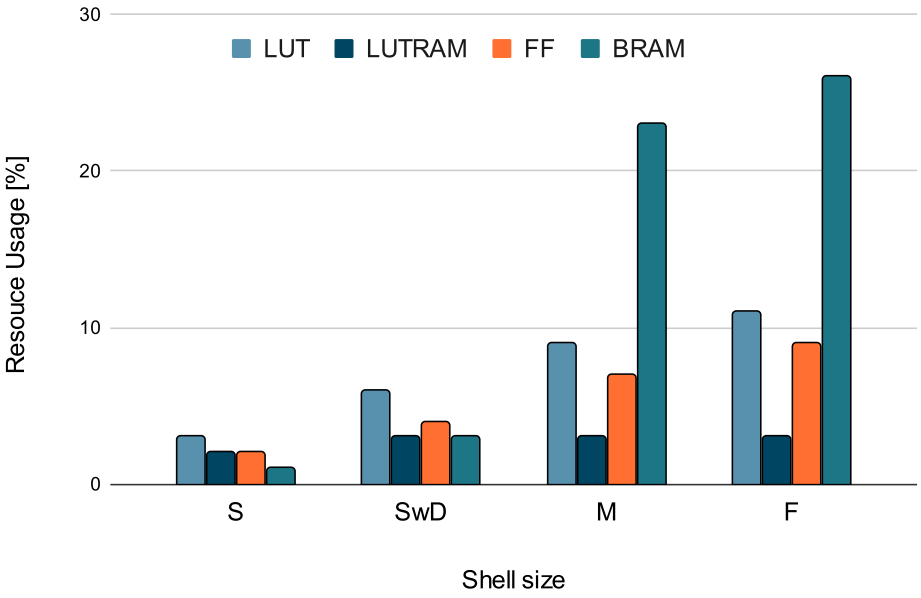


Figure 3.11: Resource Utilization of each Shell for M-KUBOS

3.4 Statically Time Division Multiplexing (STDM) network

Both FiC and M-KUBOS clusters use the statically time-division multiplexing (STDM, a.k.a. TDM) network scheme for node-to-node intercommunication. The STDM is a technique enabling several communications to share a single communication lane and establish a circuit between a source node and a destination node. Figure 3.12 shows comparison between STDM network (Figure 3.12(a)) and

packet switching network (Figure 3.12(b)). In the STDM network, communication time is divided into slots. The data from one board to another can be transmitted only at a fixed time slot. Compared with high-speed packet switching networks [45], communication latency is more expensive under light traffic load, but communication latency and bandwidth are predictable by a number of time slots. Since the STDM is a relatively simple communication multiplexing method, it simplifies the network switch and router implementation. Hence, it is beneficial for reducing the hardware resource overhead of the FiC and M-KUBOS's Shell.

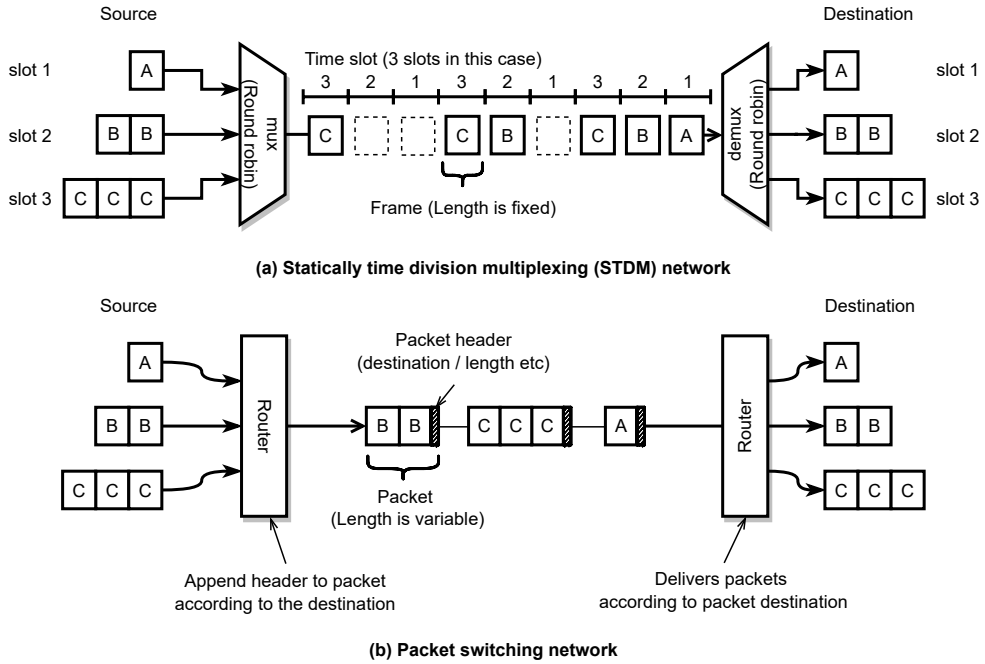


Figure 3.12: STDM and Packet Switching

However, the STDM network has some drawbacks on performance degradation when the number of communication slots that share the same channel is increased. To mitigate this drawback, Hu et al. [46] proposed an algorithm that generates an optimal number of STDM slots and routing table on a specific topology and application requirement. The STDM network has been only a few examples to be adopted for an interconnection network of parallel computer systems [47][48]. However, we believe the STDM is a more straightforward mechanism than packet switching, and its characteristic is suitable for timing critical application of the MEC. Figure 3.13 shows the packet format used in the FiC / M-KUBOS STDM network. In the FPGA design, the interface of the STDM switch is implemented as a 170 bits AXI stream

interface. The total packet data length is 170 bits. The lower 128 bits field is the data payload, and the middle 25 bits is a command field used for application-specific usages, such as burst payload transfer control and address. The upper 17 bits are header fields containing: a valid packet flag (1 bit), the packet's destination board ID (8 bits), and the packet's slot ID.

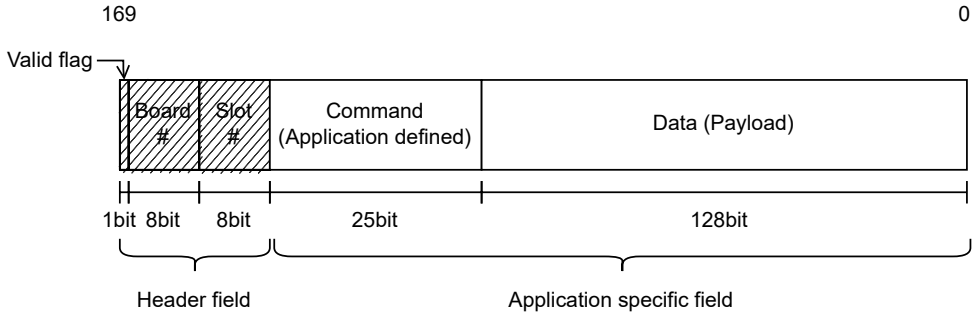


Figure 3.13: Packet Format of the FiC's STDM Network

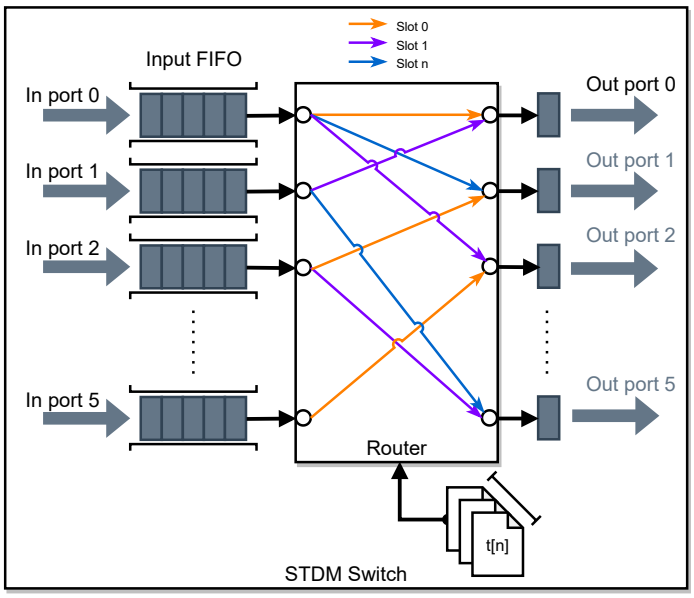


Figure 3.14: STDM switch with 6 Ports and 6 Slots

Figure 3.14 shows the outline of the FiC's STDM network switch with six ports and six slots. Incoming data stored into input FIFO is transferred to the output port at a given slot according to the header and the routing table set before the application

starts. As mentioned in FiC-SW and M-KUBOS hardware implementation, we employed economical *Firefly* flat cables for the inter-FPGA network. Each FiC-SW and M-KUBOS board provides eight channels of flat cables. The cable bundles four bidirectional lanes, connecting the lane to the same destination. Therefore, we provide four independent STDM network switches for each channel. Each board offers four 9x9 switches (eight channels + one port from/to on-chip HLS modules) at the most extensive configuration. Since eight channels can be too many for small-scale systems, we developed a parameterized switching generator that can generate the HDL description of a switch, ranging from 3x3 to 9x9 for various system configurations to maximize the partial reconfigurable region. For more high-performance data transfer capability is required, the related work [49] proposed lane aggregation with four lanes and slot distribution of the STDM network.

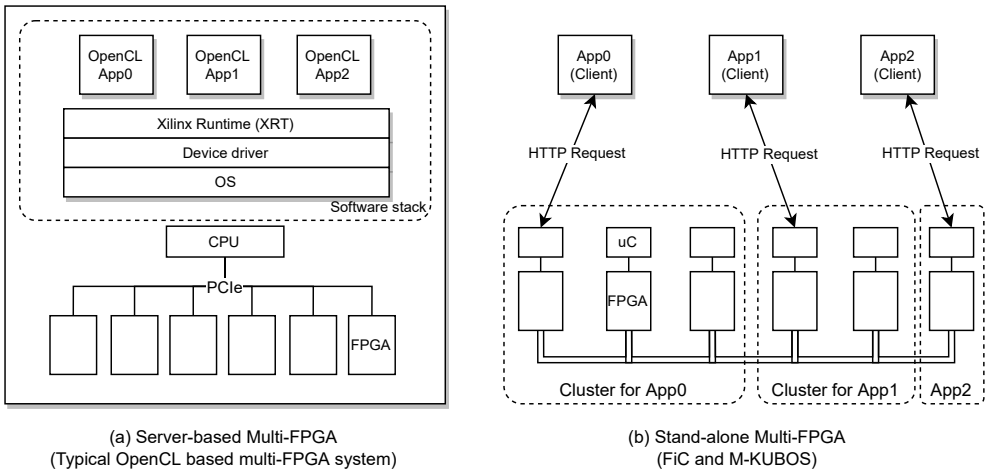
The inter-FPGA network implementation of the FiC-SW and M-KUBOS uses a 9.9 Gbps high-speed serial link, including Error Correction Coding (ECC) scheme. For this implementation, the FiC-SW and M-KUBOS used Xilinx Aurora IP. Transfer data is converted into 85 bits data for each 100 MHz system clock cycle. This relatively slow system clock is adopted to avoid the implementation difficulty of the user's HLS modules. In other words, each channel's effective data transfer rate is approximately 8.5 Gbps (1.065 GB/s). Therefore, the total data exchange throughput of the four switches is 272 Gbps (34 GB/s). The data throughput is divided number of STDM slots, so the actual data throughput is 1/n. The maximum communication latency can be expected as follows:

$$2S(L-1) + (60 + 2S)HC$$

Here, the parameters are: S is a number of slots, L is a number of packets, H is a number of network hops, and C is clock cycles. In the inter-FPGA network evaluation on the FiC-SW system, the network latency required to pass through a board is 55 clock cycles, thus 550 μsec. The Aurora-IP spends most of the clock cycles on the serial-parallel conversion. Related studies [49] and [50] have performed more detailed analysis of this STDM network.

3.5 Challenges in Applying FiC and M-KUBOS as a Computing Platform on MEC

3.5.1 Problems on stand-alone multi-FPGA FPGA platform accessibility and manageability from applications



(a) Server-based Multi-FPGA (Typical OpenCL based multi-FPGA system)

(b) Stand-alone Multi-FPGA (FiC and M-KUBOS)

Figure 3.15: Multi-FPGA Cluster Configurations

Generally, a multi-FPGA system is constructed by FPGAs installed on a server and connected to CPUs via an internal bus such as PCI Express (PCIe). Figure 3.15(a) shows this server-attached multi-FPGA system configuration. FPGAs are tightly attached to a host CPU as an acceleration device like GPUs in this configuration. This typical configuration is supported by heterogeneous computing environments such as OpenCL [51] and FPGA vendor's provided OpenCL environments such as Xilinx SDAccel [52] and Intel SDK for OpenCL [53]. Hence, the framework has guaranteed the accessibility of FPGAs from applications that offload tasks to FPGA by a device driver and middleware provided by the framework. For example, SDAccel provides application runtime (Xilinx Runtime (XRT)), device drivers, and FPGA shell design for OpenCL applications. The OpenCL applications can access and manage FPGAs by application runtime.

By contrast, a stand-alone multi-FPGA system like FiC and M-KUBOS (Figure 3.15(b)) is not supported by a standardized FPGA acceleration environment like OpenCL. Therefore, to provide task offloading service with stand-alone FPGAs on MEC, establishing a standard access interface for the FPGA platform that can support various client application environments and programming languages is required. In particular, the computing platform of MEC may be installed in a physically inaccessible environment. It is necessary to ensure the remote accessibility

and manageability of the platform. Moreover, since FPGA is a low-level device compared to other accelerator devices like GPUs, primitive device management tasks like resetting a device and configuring an FPGA are sometimes required. For example, provisioning an FPGA cluster application to the FiC and M-KUBOS from a client application is an essential task. This task requires multiple device management operations targeting multiple stand-alone FPGAs efficiently. Thus, a universal and environment-independent stand-alone multi-FPGA platform management interface for FiC and M-KUBOS is also required.

3.5.2 Problem on application programmability for FiC and M-KUBOS

Most FPGA vendors offer high-level synthesis (HLS) on FPGA application development, which dramatically improves design productivity. HLS allows application designers to design FPGAs using C/C++ with certain restrictions and use directives to optimize their designs. This allows software engineers unfamiliar with hardware design to use FPGAs in various application areas. However, most FPGA development tools such as HLS from these vendors are designed for a single FPGA system; there is no established way to design multi-FPGA applications for stand-alone multi-FPGA systems such as FiC and M-KUBOS. For example, in application development for FiC and M-KUBOS, application designers have to implement an application-specific multi-FPGA communication protocol to communicate among FPGAs [54][55][56].

Moreover, no application test environment is provided except for debugging on the real system; the application designer needs many trials and errors on the real system implementation to test the application. This less application programmability of FiC and M-KUBOS hinders the opportunity to adopt multi-FPGA systems on MEC. Thus, providing a parallel programming environment that can support stand-alone multi-FPGA systems and improving application programmability is required.

4

Platform Management Architecture for Stand-Alone Multi-FPGA System

This chapter presents a management architecture for the stand-alone multi-FPGA, inspired by cloud service methodology. The management architecture is implemented in the FiC cluster system and evaluated by the scenario of the cluster setup remotely. The chapter is organized as follows: Section 4.1 describes the proposed platform management system concept for stand-alone multi-FPGA. Section 4.2 presents the implementation of the proposed management system FiC-RFC. Section 4.3 evaluates the FiC-RFC. Section 4.4 introduces related research on parallel programming for multi-FPGA systems. Finally, we summarize the chapter in Section 4.5.

4.1 Concept Overview

In order to use FiC and M-KUBOS in MEC, it is necessary to have a platform management system that can support various application environments by taking advantage of the features of the stand-alone multi-FPGA architecture that can flexibly change the system configuration. For example, in order for a MEC application to be able to use FiC and M-KUBOS, it is necessary to have an interface that allows the MEC application to perform management operations on multiple FiC and M-KUBOS

nodes. Furthermore, it is important to ensure accessibility to remote management functions on the computing platform of MEC.

In this research, we focused on RESTful interface, which is an application programming interface widely used in cloud applications, as an interface for managing and accessing the FIC and M-KUBOS platforms from applications. RESTful interface [57] is an application programming interface that uses HTTP access. HTTP access provides better remote accessibility and is less dependent on the platform environment, ensuring accessibility from various applications and platforms. In RESTful API, an associated uniform resource identifier (URI) is provided to call API. The application can invoke these APIs by sending a specified JavaScript Object Notation (JSON) or eXtensible Markup Language (XML) message to the associated URI using HTTP GET and POST requests. And the API responses can obtain as JSON or XML messages. Since RESTful API is accessible by simple HTTP requests, applying a RESTful interface for FPGA management tasks improves the accessibility from various client application environments or programming languages.

In addition, since various applications are executed on-demand in MEC in response to requests from clients, it is expected that the use of FPGA systems such as FIC and M-KUBOS from MEC applications will be on-demand and spot-used. Therefore, it is desired to realize an execution environment in which FPGA can be used without being aware of the FPGA platform used by the application. This research focuses on the serverless architecture method used in cloud applications. We examined the realization of an FPGA usage environment in which MEC applications can manage FPGA platforms and execute FPGA application kernels in a Function-as-a-Service (FaaS) manner. FaaS is a category of cloud computing services that provides an on-demand computation service to client applications that allows running specific tasks as a function call style. This FaaS approach can abstract the hardware platform behind them from the application use. We applied this FaaS fashion service provision method for multi-FPGA platform management from the client applications. With the FaaS-based FPGA usage environment, MEC applications can manage the FPGA platform and execute FPGA applications with only simple API calls. It anticipates that it will facilitate the applications' FPGA platform usability.

To examine these concepts, we developed an FPGA platform management middleware called *FiC-RFC* (Restful FPGA Controller) for FiC and M-KUBOS systems. FiC-RFC implemented essential platform management tasks for the stand-alone multi-FPGA system from the MEC application and made them accessible by RESTful APIs. The following sections introduce the FiC-RFC architecture implementation on the FiC-SW multi-FPGA system.

4.2 FiC-RFC Implementation for the FiC cluster

4.2.1 Platform management operations for FPGA task offloading

For considering a platform management system for MEC applications to utilize FiC and M-KUBOS, extracting the required platform management operations from typical FPGA platform usage scenarios is necessary. This section reviews task offload scenarios from MEC applications to FiC clusters and describes the required platform management tasks.

(a) FPGA node allocation
This task allocates the required FPGA nodes as requested by the client application. Resource manager systems such as *FiC-RM* and *MEC-platform* can perform this assignment task manually or automatically, taking into account the inter-FPGA network topology of FIC and M-KUBOS.

(b) FPGA setup (FPGA programming)
This task sets up the FPGA of each FiC. Since the FPGA in the initial stage has no logical configuration information (bitstream) programmed, it must perform the programming task before use. In FiC, this task performs by the onboard microcontroller (RPi3).

(c) Inter-FPGA network setup (STDM network)
This task sets up the routing table of the STDM switch for the inter-FPGA network. The STDM switch is implemented as a Shell of the FPGA design, so this operation must perform after the operation (b).

(d) Application data transfer
This task transfers application data to compute on FPGA from the client application. The onboard RPi3 can transfer data to the FPGA's BRAM or DRAM.

(e) Application kernel control
This task controls the application kernel implemented on the FPGA, such as starting, resetting, or setting parameters.

(f) Application result data transfer
This task transfers application results to the client application. The onboard RPi3 can fetch data from the FPGA's BRAM or DRAM.

(g) FPGA debugging from a remote environment
This task is to debug the FPGA from a remote environment. Generally, on the FPGA debugging, JTAG and embedding debugging IPs such as Integrated Logic Analyzer (ILA) [58] are used. The FiC establishes a JTAG connection to

the FPGA with a USB JTAG cable and Xilinx Virtual Cable (XVC) [59] server on RPi3. It allows debugging from Xilinx Vivado on a remote environment.

4.2.2 Management System Architecture on FiC cluster

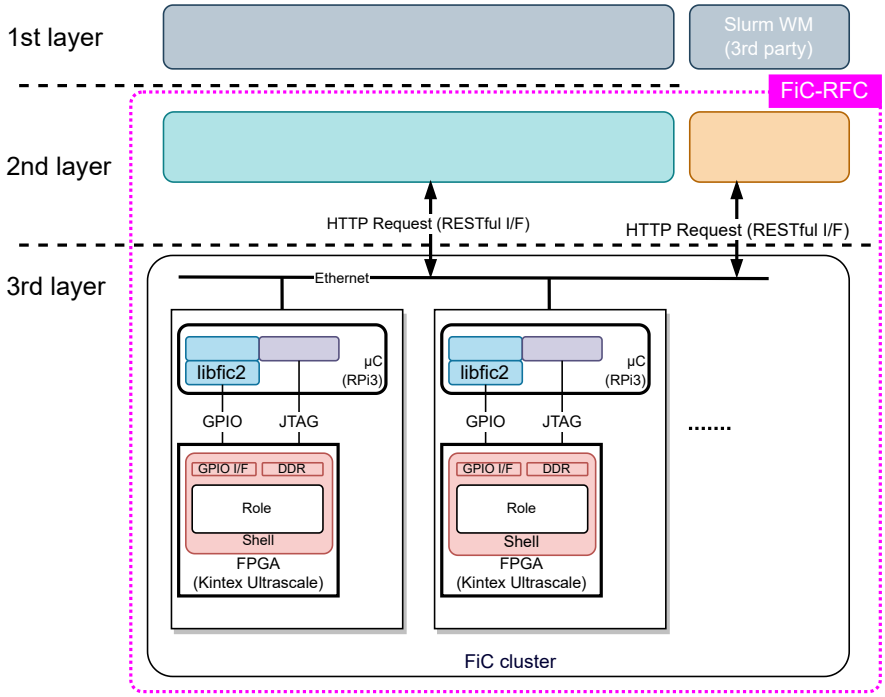


Figure 4.1: Management System Architecture and FiC-RFC Scope

The management architecture of FiC and M-KUBOS implements each management layer of the platform management function as an independent module and combines them with a general-purpose application interface (RESTful API). In this way, we aimed to realize a highly versatile management architecture by reducing the dependency between specific environments and each management layer. By adopting a highly independent architecture at each management layer, we have realized an architecture design that can use and manage FiC and M-KUBOS in various environments and system configurations. For example, if an advanced job management system is not required, it is possible to use only the FPGA platform from the application.

The current FiC's cluster management system consists of three layers, as shown in Figure 4.1. The first layer provides FPGA virtualization, resource management of the FPGA cluster, and job scheduling. This layer depends on the deployment

environment or purpose of the FPGA system. For example, the MEC uses the *MEC platform* to allocate computing resources to the MEC application [60]. The current FiC cluster uses the FiC resource manager (FiC-RM) [61] for this resource management task which is a lightweight job and resource manager developed in related work. The FiC-RM manages FiC-SW nodes as computational resources. It automatically assigned accepted jobs to available nodes according to the scheduling policy.

The first layer can also apply a general-purpose job scheduler such as Slurm Workload Manager [62] or LSF [63]. These are often used in HPC clusters and other clustering computing systems. We currently employ Slurm Workload Manager when combining the FiC cluster and its successor M-KUBOS cluster instead of FiC-RM.

The second and third layers are the FiC-RFC architecture's main scope; the second layer is the cluster node management layer, which provides primitive FPGA management operations such as configuring cluster nodes, data I/O, and monitoring nodes. The *ficmgr* is a CLI-based multi-FPGA management utility that setup the multi-FPGA cluster with JSON description. And *libficmgr* is an application programming library version of *ficmgr* that the client application can use. Both *ficmgr* and *libficmgr* communicate FPGA nodes by RESTful interface.

The third layer is the device layer. FPGA management middleware called *ficwww* and *libfic2* and XVCd are running on each RPi3 of the FPGA nodes. The ficwww is a RESTful API provider for FPGA node management functions, and libfic2 is an FPGA hardware driver for accessing *Shell* on the FPGA device. The XVCd is a JTAG server for debugging FPGA implemented for Xilinx Virtual Cable [59] protocol to communicate Xilinx Vivado's hardware server. The FPGA design of the FiC-SW board employs *Shell* and *Role* style. The Shell on the FPGA device is an FPGA logic design template that contains glue logic for fundamental functions such as GPIO interface to the libfic2, onboard DRAM controller, and STDM switch. And the user application is implemented in the *Role*.

The following subsections describe the details of each management component.

ficmgr/libficmgr

The *ficmgr* and *libficmgr* are cluster management tools and libraries for user applications running on the client side. Both *ficmgr* and *libficmgr* communicate each FiC-SW board by RESTful APIs. The *ficmgr* provides cluster management operation from the command line interface on the user client. The application user can define the cluster configuration by JSON style setup files without the job scheduler on the first layer. The ficmgr is useful when FPGA cluster assignment is predetermined, and they can make the management architecture simple for the embedded environment. The *libficmgr* is a cluster management API binding for Python language, and it can be used in user applications such as Jupiter notebook.

On-board Microcontroller for Cluster Node Management

As we described it in Section 3.2, we employed *Raspberry Pi3* (RPi3) single-board computer as an onboard management controller for the FiC-SW board. The RPi3 is one of the successful single-board computing platforms, available at a low-cost and well-supported complete package of Linux distributions. In the FiC-SW board, the onboard RPi3 is connected to the management network via the Ethernet (Wi-Fi or Wired). It is also connected to the FPGA via RPi3's GPIO port for FPGA configuration and communicates to FPGA logic from the RPi3.

libfic2

A *libfic2* is a low-level communication driver for communicating RPi3 and FPGA via GPIO port, It provides FPGA programming and communication capabilities of the FPGA logic. The libfic2 is developed with C language for small footprint and high performance, and it is supported with Xilinx's SelectMap x16 or x8 FPGA configuration modes for the FPGA configuration.

The RPi3's GPIO port is not the best design option for high-speed communication. However, the RPi3 did not have any versatile high-speed IOs at the time, so we had to use GPIO for FPGA communication on the FiC cluster.

Shell

Shell refers to the pre-designed and pre-implemented FPGA logic design containing fundamental glue logic for FPGA applications. It enables the enhancement of hardware re-usability and FPGA application productivity. FiC-SW's Shell contains GPIO interface for communicating on-board RRi3, DRAM memory controller for onboard DDR and STDM switch for inter-FPGA network. The kernel of the user's FPGA application is implemented on the *Role*. Role refers to the dynamic region of the FPGA logic design, which can independently reconfigure at the run time with the partial reconfiguration technique [64].

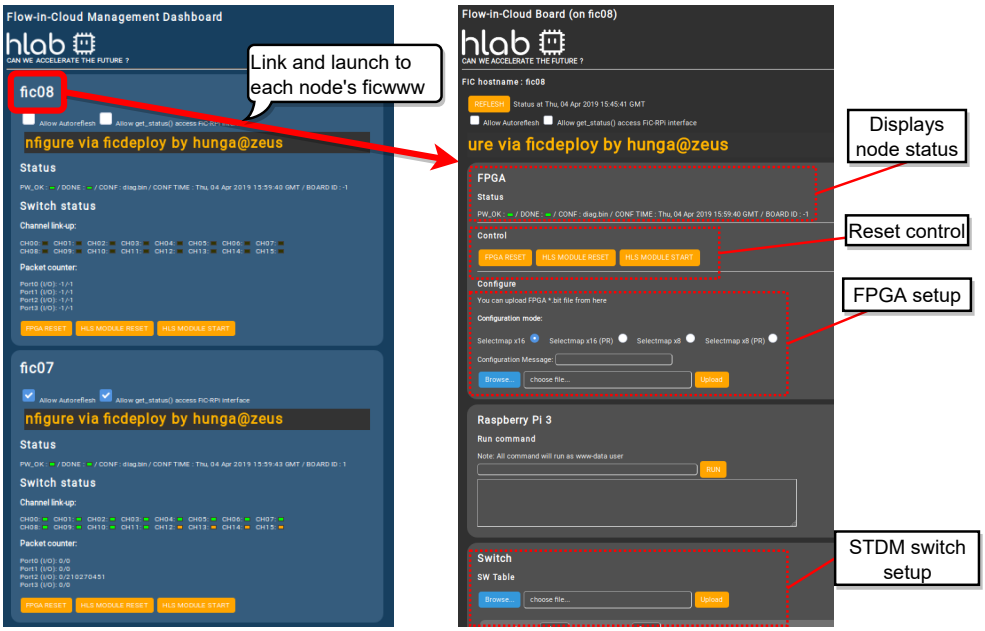
ficwww and ficdash

Although the batch-style CLI-based multi-FPGA management utility *ficmgr* is provided, a more user-friendly interface is needed, especially at the program development stage.

A *ficwww* is a device-side RESTful API provider on the FiC-SW board. ficwww is developed with Python and lightweight web framework PyFlask + JINJA2 [65], hosted with *lighttpd* HTTP daemon on the RPi3. The ficwww also provides a simple WebGUI (Figure 4.2(b)). It allows the users to do FPGA management operations

such as FPGA's reset/set, programming, STDM switch configuration, and monitoring the FPGA status and network link via a web browser.

A *ficdash* is a cluster management web dashboard for the FiC cluster (Figure 4.2(a)). This web dashboard is hosted on the control server. It manages and monitors multiple cluster nodes' information in one web dashboard. Users can monitor the multiple cluster node status via a web browser, such as node power and STDM network status.



(a) ficdash
FiC Management Dashboard
(Control Server)

(b) ficwww
WebGUI (Each node)

Figure 4.2: ficdash and ficwww WebGUI

4.2.3 FPGA Configuration Bitstream Distribution over HTTP

The FPGA configuration bitstream is distributed via the management network by the HTTP request for the cluster setup process. This FPGA configuration bitstream data is sized around 30∼40 MB for each node depending on the FPGA logic size, and the data size will be inflated due to encoded in BASE64 text data in RESTful request. This FPGA configuration bitstream distribution time may become a problem regarding the system installed into the edge and used by several applications.

Therefore, the *ficmgr* and ficwww use gzip compression on the HTTP transport to suppress the configuration data size and distribution time. Table 4.1 shows our

preliminary evaluation of gzip compression for three different design for FiC-SW. In Table 4.1, Application name ending with *pr* means the configuration bitstream for the partial reconfiguration. The column *Config* is a raw binary FPGA configuration bitstream size. The column *Clear* is a clear configuration bitstream for partial reconfiguration required to clear the FPGA logic region after the full configuration.

The compression ratio is reached at around 1/15, and the result shows that compression is very effective for FPGA bitstream data.

Table 4.1: Configuration Size (in MB)

#	Application	Config	Clear	Total	Gzip
1	lchk	34.18	-	34.18	2.32
2	lenet	34.18	-	34.18	2.85
3	sort	34.18	-	34.18	2.32
4	lchk_small_pr	12.31	0.81	13.11	1.01
5	lenet_small_pr	12.31	0.81	13.11	1.57
6	sort_small_pr	12.31	0.81	13.11	1.01
7	lchk_max_pr	26.64	1.82	28.46	1.23
8	lenet_max_pr	26.64	1.82	28.46	1.84
9	sort_max_pr	26.64	1.82	28.46	1.20

4.2.4 FiC-RFC's RESTful APIs

Representational State Transfer (REST) [57] is a software architecture style currently widely used in web applications. In the RESTful API design, all API calls are represented as a corresponding URI, and it is callable by simple HTTP requests (POST / GET / DELETE / PUT, etc.) from clients. Considering the FiC cluster usage as an accelerator on the MEC, web API is advantageous for making it accessible from various applications and programming languages. ficwww provides the essential node management API sets shown in Table 4.2.

Table 4.2: FiC-RFC's RESTful APIs

API	Method	Function
/fpga	POST	Configure FPGA with bitstream
	GET	Obtain FPGA configuration status
	DELETE	Reset and flush FPGA configuration
/switch	POST	Configure switch routing table
/hls	POST	User HLS module start/reset control or Data Send / Receive from HLS module
/status	GET	Obtain FiC board status information
/regread	POST	Read out data from specified memory address on FiC-SW
/regwrite	POST	Transfer data to specified memory address on FiC-SW
/runcmd	POST	Invoke specified command on RPi3 controller

4.2.5 Setting up an application on the FiC cluster

Executing an FPGA application on multiple FiC-SW nodes needs configuration data deployed to each FPGA at first. Individual FPGA configuration bitstream is needed to be transferred to each FPGA. In the FiC cluster, this process has done by HTTP data transport by RESTful APIs. After FPGAs have been configured, the routing table for the STDM switch on each node is distributed and configured.

To handle the multiple FiC-SW nodes setup sequences, a cluster provisioning tool called *ficmgr* setups multiple nodes simultaneously with cluster configuration JSON file. An example of the cluster setup JSON file is shown in Figure 4.3. A key `fic08` means the target cluster node for configuration. Entries under `fpga` specify FPGA bitstream configuration settings. Entries under `switch` specify STDM switch configuration. And entries under "option" specify after configuration behaviors such as automatically starting the FPGA application after the FPGA configuration, running

a custom command on the RPi3, etc.

```
{
 "fic08":{
     "fpga":{
         "bitstream": "fic_top.bin",
         "progmode": "sm8",
         "msg" : "Top module"
     },
     "switch": {
         "slots": 1,
         "ports": 4,
         "outputs": 1,
         "table": {
             "output0": {
                 "port0": {
                     "slot0": 0
                 },
                 "port1": {
                     "slot0": 0
                 },
                 "port2": {
                     "slot0": 0
                 },
                 "port3": {
                     "slot0": 0
                 }
             }
         }
     },
     "option": {
         "auto_hls_reset_start": true,
         "auto_runcmd": "cat /proc/cpuinfo"
     }
},
...
```

Figure 4.3: An example of JSON description

4.2.6 Using FiC cluster from client application

The *libficmgr* provides cluster operation capabilities from the client application such as Jupiter Notebook or Python. Figure 4.4 shows an example user application snippet in Python. Since the library communicates to the cluster with RESTful APIs, the client application can access the cluster from the Internet and offloads the workload.

```
1  import os
2  import sys, traceback
3  import libficmgr
4  #------------------------------------------------------------
5  MK1_FPGA_BITSTREAM = 'mk1_ficbd_ddr4_20201017.bin'
6  STDM_TABLE_FILE    = 'table.json'
7  DDR_DATA_FILE      = '1G.dat'
8  DATA_SIZE          = 1024*1024*128
9  TARGETS            = ('fic00', 'fic01', 'fic02', 'fic03')
10 #------------------------------------------------------------
11 if __name__ == '__main__':
12     ficmgr = libficmgr.libficmgr()
13
14     for target in TARGETS:
15         # ---- Configure target FPGA via HTTP request -----
16         ficmgr.fic_prog(target, 'sm8', True,
17                         MK1_FPGA_BITSTREAM, "Run_example_app")
18
19         # ---- Configure routeing table STDM switch
20         with open(STDM_TABLE_FILE, 'rt') as f:
21             ficmgr.fic_set_switch(target, f.read())
22
23         # ---- Reset FPGA and Start HLS ----
24         ficmgr.fic_hls_cmd(target, 'reset')
25         ficmgr.fic_hls_cmd(target, 'start')
26
27         # ---- Transfer data to DDR DRAM ----
28         with open(DDR_DATA_FILE, 'rb') as f:
29             ficmgr.fic_hls_ddr_write(target, f.read(DATA_SIZE), 0)
30
31         # ---- Set application parameter / command ----
32         ficmgr.fic_hls_send(target, [0x01, 0x01])
33
34     for target in TARGETS:
35         # ---- Wait until computation done signal assert ----
36         while ficmgr.fic_read(target, 0xfff0) == 0:
37             # ---- Read data file and transfer to DDR DRAM ----
38             # Readout 1MB from DDR memory
39             ret = ficmgr.fic_hls_ddr_read(target, DATA_SIZE, 0)
40             print(target, len(ret['data']))
```

Figure 4.4: An example user application in Python with libficmgr

4.2.7 Remote FPGA Debug

Even for application development with HLS, hardware-level debugging is often required. The FiC-RFC supports remote hardware debugging with the Integrated Logic Analyzer (ILA) [58] of Xilinx FPGA. We use Xilinx Virtual Cable (XVC) [59] protocol to support this feature. The XVC is an open-source TCP/IP protocol by Xilinx, which acts like a JTAG cable to remotely access the target FPGA device.

The benefit of using XVC is the protocol acts just like a cable, which means the XVC is just transferring the JTAG operation issued by the hardware server (hw_server) on localhost to the target device in the remote. Therefore, the XVC is independent of vendor-specific JTAG commands, and it allows us to use ILA with various JTAG hardware.

Figure 4.5 shows the current FiC system remote debugging environment overview. In the FiC-SW node, we use FTDI's FT232H [66] based JTAG bridge to the USB port on RPi3 and run an open-source XVCd server [67] on the RPi3. Using this configuration allows users to access the JTAG port of each FPGA and debug with ILA from the remote environment.

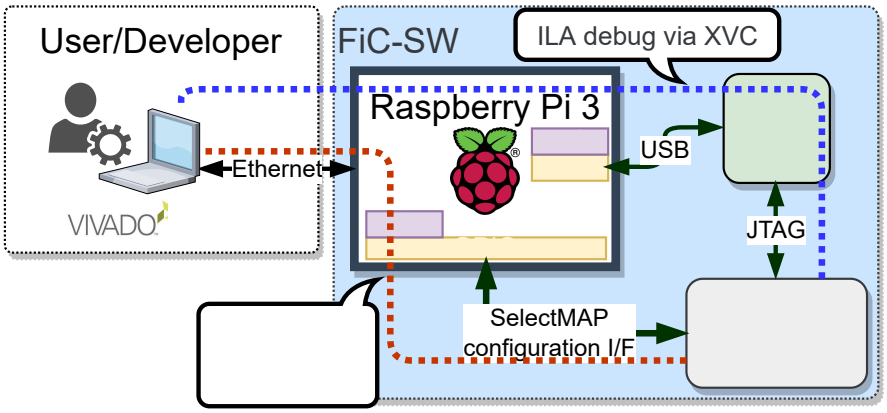


Figure 4.5: Debugging FiC-SW over XVC

4.3 Evaluation

4.3.1 Evaluation environment setup

The provisioning time of an application to a cluster is an important performance metric for a platform management system. Therefore, in this section, we evaluated the provisioning time for FPGA applications using FiC-RFC, the proposed multi-FPGA platform management system. The evaluation uses a 12-node FiC-SW mk1 board and three different designs shown in Table 4.1, and evaluates the performance of each part related to application provisioning shown in Figure 4.6.

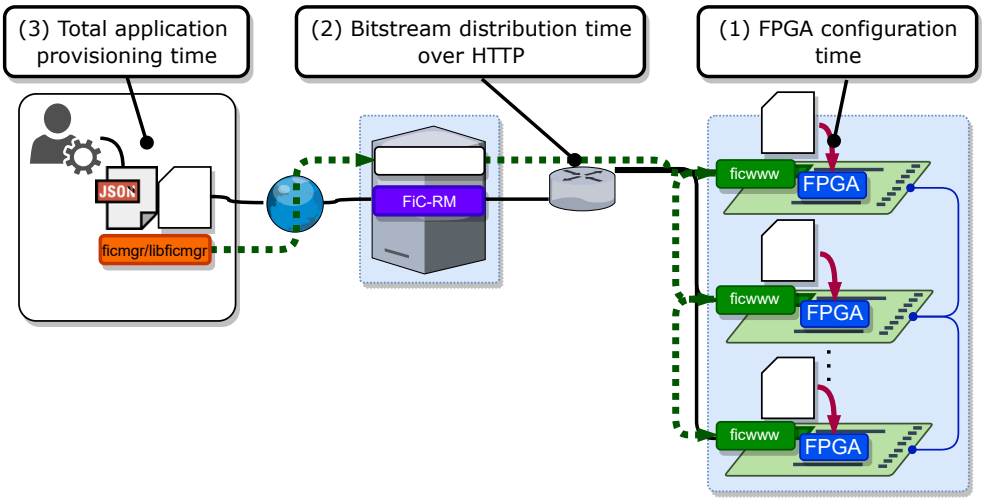


Figure 4.6: FiC-RFC evaluation setup

4.3.2 FPGA configuration time of the FiC-SW board

Since the FiC cluster is a bare-metal and disaggregated multi-FPGA system for MEC base stations, many applications are expected to set up and use FPGAs on demand. Therefore, short FPGA reconfiguration time is essential to serve computation services for multiple users or applications as many as possible. Table 4.3 shows a comparison of a single FPGA configuration time for Kintex Ultrascale XCKU095 on the FiC-SW board by several configuration methods.

(a) RPi3 - Configuring FPGA with SelectMap 8 bit parallel mode by RPi3 and libfic2 library with three-speed modes.

(b) DLC9LP - Configuring FPGA with USB Xilinx Platform Cable (DLC9LP) by Vivado hardware manager.

(c) XVC - Configuring FPGA with USB JTAG bridge (FT232H) and Xilinx Virtual Cable (XVC) by Vivado hardware manager.

Table 4.3: Comparison of FPGA Configuration Time

#	Method	Mode	Time	Bitrate
1	RPi(Safe)	SMAPx8	0:31 s	8.80 MB/s
2	RPi(Normal)	SMAPx8	0:10 s	25.76 MB/s
3	RPi(Fast)	SMAPx8	0:01 s	202.6 MB/s
4	DLC9LP	JTAG	1:17 s	2.71 MB/s
5	XVC	JTAG	4:26 s	1.05 MB/s

The speed mode differences between Safe/Normal/Fast in the RPi3 are signal hold time differences for the SelectMap 8 bit parallel port driven by GPIO port. Safe mode is the most conservative configuration mode; it holds the signal longer for a stable FPGA configuration. Normal mode holds it shorter than Safe mode. The *Fast* mode optimizes the GPIO bit-bang operation to reduce the number of parallel port signal toggling. It can reduce the number of PIO operations by CPU. However, due to variations in electrical signal characteristics in FiC-SW board and RPi3 manufacturing, we observed that the Fast mode is not always stable on some nodes, so we use the Normal mode typically. The result shows that the RPi3(Normal) mode achieved 10 seconds to configure the FPGA. Compared to other FPGA programming methods, it is 6.4x faster than configuring by genuine Xilinx's platform JTAG cable DLC9LP.

As a comparison with other systems, the IBM's cloud FPGA [68][69] achieved 8.264$\sim$12.215 MB/sec of FPGA configuration throughput. This FPGA system uses Xilinx's Kintex Ultrascale XCKU60 FPGA of the same series as FiC and configures

FPGA via HTTP access. The studies [70][71][72][73] are reported up to 10 MB/s of FPGA configuration bitrate with 8 bit ICAP (Integrated Configuration Access Port) on the Xilinx's Spartan, Virtex-2, and Zynq-7000 MPSoC systems. Thus, FiC-SW with RPi3 achieves 1.1-2.1x FPGA configuration performance compared to other systems, which is sufficient to support FPGA application provisioning over HTTP.

4.3.3 FPGA bitstream distribution time over HTTP

FPGA bitstream distribution time is also an essential metric for the platform management system. In the FiC cluster, each FPGA bitstream distributes by RPi3's 100Base-TX Ethernet network over HTTP (RESTful API). Since FiC-SW's Ultrascale FPGA is a middle-range FPGA, the size of each configuration bitstream is around 35MB (Table 4.1). Figure 4.7 shows the comparison between the FPGA configuration bitstream transfer time of the LeNet design 1∼12 of FiC-SW nodes. In Figure 4.7, the **w/o gzip** is FPGA bitstream transfer time without gzip compression, and **w/gzip** is FPGA bitstream transfer time with gzip compression.

For FPGA configurations via the RESTful API, the FPGA bitstream (binary format) is converted to BASE64 encoding in REST messages (JSON). Therefore, even though the total size of messages over HTTP has increased by about 1.3 times, the results show that applying gzip compression reduced the bitstream transfer time to 1/14. This result shows that applying gzip compression can reduce the overhead of BASE64 conversion, as the FPGA configuration bitstream binaries can be effectively compressed by gzip (Table 4.1). From a scalability point of view, when gzip compression is not applied, the FPGA bitstream distribution time tends to increase as the number of target nodes increases, which is 1.24 times that of 12 nodes compared to 1 node. This is because the REST message that operates each node becomes huge without gzip compression, which increases the load on the management network to which RPi3 is connected. On the other hand, when gzip compression was applied, no increase in FPGA bitstream distribution time was observed even for 12 nodes, and stable performance was achieved. These results show that applying to gzip compression to the FPGA bitstream contributes to shortening and improving the efficiency of FPGA bitstream distribution time. In addition, in the evaluation of 12-node FiC clusters, the FPGA bitstream distribution time was within 10 ms, which was a negligible percentage of the total application provision time.

4.3.4 Total application provisioning time with FiC-RFC

Figure 4.8 through Figure 4.10 show the total application provisioning time of each number of FiC-SW nodes (up to 12). In this evaluation, we are using RPi3 (Normal) configuration mode (method #2 in Table 4.3) with gzip bitstream compression for HTTP transfer, and three different designs in Table 4.1 including w/ and w/o partial

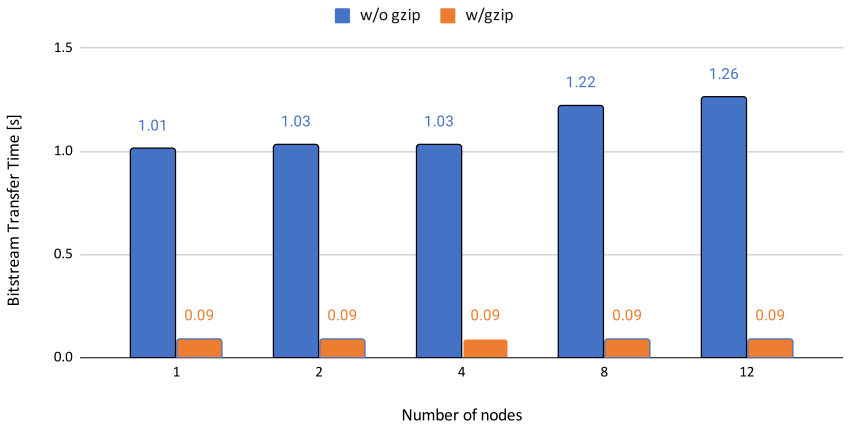


Figure 4.7: FPGA Bitstream Distribution Time over HTTP

reconfiguration. We have evaluated each condition ten times and compared it to the shortest total setup time, which can be considered our current management system's achievable maximum setup performance. The total application provisioning time includes a bitstream distribution time, FPGA configuration time from RPi3 to FPGA via GPIO port, HTTP request processing time including bitstream data encoding (BASE64 and compression, etc.), and client's provisioning process synchronization time.

Results in Figure 4.8 through Figure 4.10 show total application provisioning time is not significantly increased according to the number of nodes. The approximately increased time overhead per 4 nodes is 2%. Thus, the result shows the application provisioning performance of the FiC-RFC has good enough scalability to support up to 24 nodes, which is the current FiC cluster scale.

Comparing application provisioning time between full configuration (**Full**) (avg. 16.80s) in Figure 4.8, partial reconfiguration in maximized PR area configuration (**Max PR**) (avg. 14.02s) in Figure 4.9, and minimized PR area configuration (**Small PR**) (avg. 8.00s) in Figure 4.10, the total time is reduced 16.54% in the **Max PR** case and 52.38% reduced in the **Small PR**, according to the configuration bitstream size.

Comparing the bitstream file size of LeNet application between **Full** (34.18 MB), **Max PR** (28.46 MB), and **Small PR** (13.11 MB), the **Small PR** size is 62% smaller than **Full**, but the total time reduction is 57% only. On the other hand, the **Max PR** size is 17% smaller than **Full**, but the total time reduction is 21%. The partial reconfiguration requires 2 phases of FPGA programming with the *clear* bitstream before programming the PR bitstream. However, the result suggests that the penalties caused by 2 phases configuration are negligible, and a bigger PR region design is

more efficient than a small PR region design in terms of the application provisioning time.

Figure 4.11 and Figure 4.12 show total provisioning time variations of LeNet application in every ten iterations. In both full configuration and PR cases, the result shows that the worst provisioning time is longer, along with the increasing number of target nodes. This result is caused by the issuable numbers of HTTP requests by the cluster provisioning client application regulated. Comparing the full configuration case and the PR case, the worst setup time in the full configuration case is more stable than the PR case because the setup process is only once per node. But in the PR case, the worst setup time is longer because it did two-phase configurations per node. Thus, the result suggests that the setup time stability must be considered if applying to larger-scale systems.

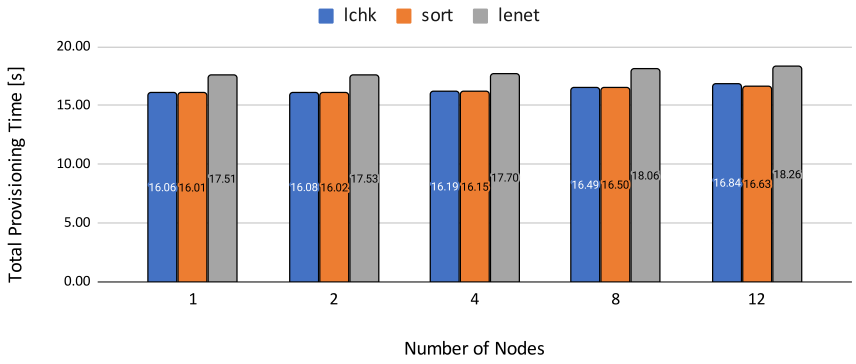


Figure 4.8: Total Application Provisioning Time (Full configuration)

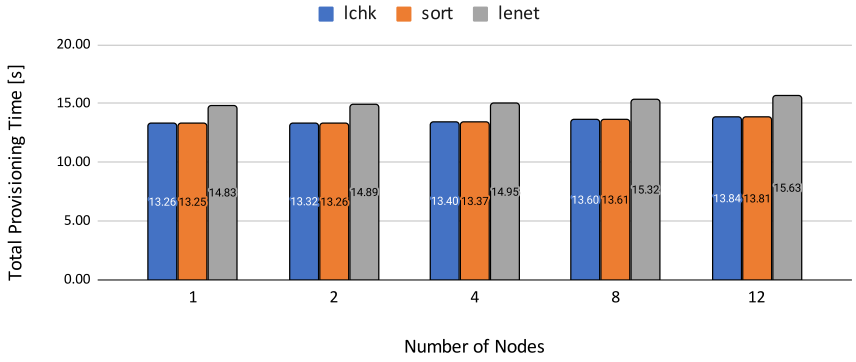


Figure 4.9: Total Application Provisioning Time (Max PR area)

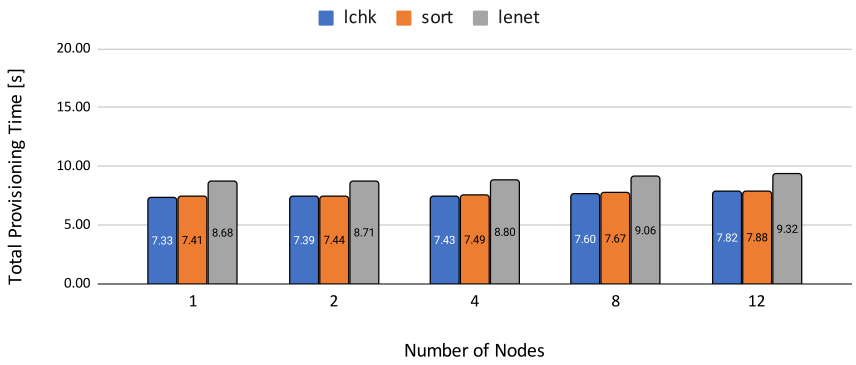


Figure 4.10: Total Application Provisioning Time (Small PR area)

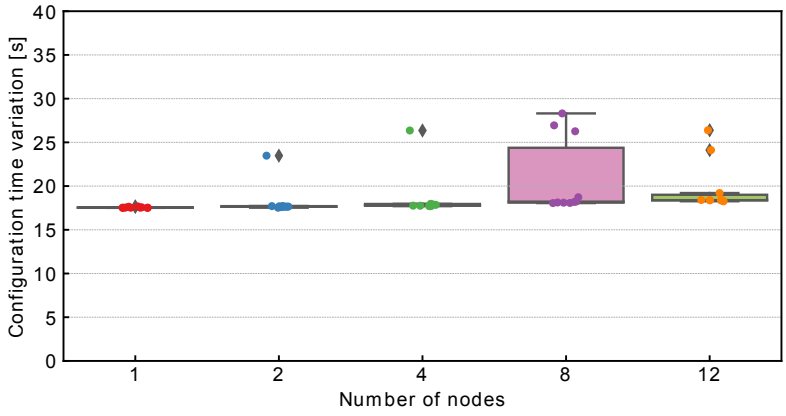


Figure 4.11: Setup Time Variation (Full Configuration)

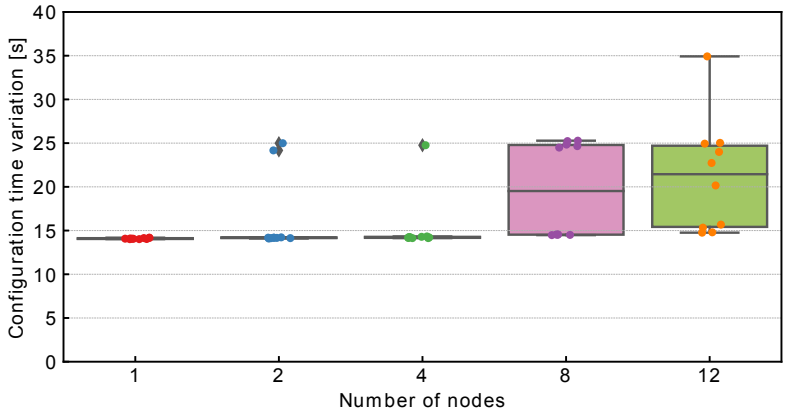


Figure 4.12: Setup Time Variation (Partial Reconfiguration)

4.4 Multi-FPGA Applications on FiC and M-KUBOS cluster using FiC-RFC

In the FiC project, many neural-network-based image recognition applications for multi-FPGA have been implemented with FiC/M-KUBOS and using the FiC-RFC multi-FPGA platform management system.

For example, a related work FiC-RNN [54] implements recurrent neural networks (RNN) [74] on the FiC cluster using up to four FiC-SW nodes controlled by the FiC-RFC. It achieved 31-61x speed-up compared to Intel Xeon E5 CPU and achieved 3.5-2.8x energy efficiency. And a related work [55] implements LeNet [75] on the eight nodes FiC cluster. It achieved 149.6 GOPS, 12.6x performance compared to Intel Xeon E5-2667. For the M-KUBOS cluster, a related work [56] implements ResNet-18 [76] on M-KUBOS cluster. It achieved 158 GOPS, 1.16x performance, and 6.56x energy efficiency compared to AMD Ryzen 3960X implementation.

According to these related work and results, the FiC-RFC allows accessible and manageable stand-alone multi-FPGA systems such as FiC and M-KUBOS cluster on the MEC by the applications, and the system can be expected to achieve better performance and energy efficiency than the CPU in image recognition applications, which is expected to be one of the primary tasks in MEC.

4.5 Related Work

Deploying orchestration software is a common approach to managing configurations for multiple compute nodes. OpenStack [32] is an open-source orchestration software that allows users to deploy user-defined configuration templates for relatively large IT infrastructures in data centers. For example, in a related study [31], an FPGA system applicable to a cloud environment is orchestrated using OpenStack.

Related researches [68][69][77] propose the concept of the cloudFPGA. It is similar to the FiC concept, which provides FPGA acceleration as a cloud service. Like the FiC cluster, they propose FPGA systems and management systems that allow direct access to FPGAs from applications using a REST-based API over HTTP.

However, these architectures are different in purpose and scale from FiC. The system was designed for applying high-density stand-alone FPGA systems to data centers. Therefore, the cloudFPGA implements a management function that provides a RESTful interface on the FPGA itself, which is a suitable architecture for designing a high-density FPGA system. However, because this design shares the same FPGA between the management function and the application, the implemented application may affect the management function's behavior and the platform's reliability. For this reason, an architecture like FiC in which the FPGA and management functions are implemented separately is considered suitable for use in an edge environment where

platform reliability is essential.

4.6 Summary

In this chapter, we introduced a platform management system for a stand-alone multi-FPGA system which improves the accessibility of applications and the manageability of the stand-alone multi-FPGA system. To prove the concept, we developed a management system architecture called FiC-RFC and applied it to the FiC cluster, stand-alone multi-FPGA systems intended for MEC environments.

The FiC-RFC enables the management and controlling of the multi-FPGA system from a remote environment through a RESTful API, a web-based general-purpose programming interface. And it allows supporting various application environments to use the multi-FPGA cluster for task offloading.

We evaluated the time required for each part of a 12-node FiC cluster to provision an application by HTTP access. The result showed the practicality of FiC-RFC on the system scale of the current FiC cluster.

5

Parallel Programming Environment for Multi-FPGA System

This chapter presents a parallel application programming environment for the multi-FPGA system and implements an MPI-compatible programming library for bare-metal FPGA application development with HLS. The chapter is organized as follows: Section 5.1 describes the concept and implementation details of FiC-MPI. Section 5.2 describes the multi-FPGA application development with FiC-MPI. Section 5.3 describes the performance evaluation of FiC-MPI and the porting work of the Himeno Benchmark using FiC-MPI as a case study and evaluates its benchmark performance. Section 5.5 introduces related research on parallel programming for multi-FPGA systems. Finally, we summarize the chapter in Section 5.6.

5.1 FiC-MPI: Message Passing Interface library for HLS

5.1.1 Concept Overview

The design concept of the FiC-MPI is to enable parallel programming for multi-FPGA systems such as FiC and M-KUBOS clusters in standard MPI and HLS C/C++ languages. It eliminates efforts and burdens for multi-FPGA application development and debugging. The FiC-MPI provides a standard MPI interface to handle necessary node communications and synchronizations for multi-node parallel

programming. And the library is designed for utilizing FPGA hardware efficiently. It allows the application programmer can be managed the required FPGA hardware resources for MPI hardware implementation according to the scale of the problem.

For improvement of application productivity, the library provided a function-level multi-node MPI simulation environment called *FiC-MPI simulator*. The environment enhanced debugging productivity and agility of multi-FPGA application development.

5.1.2 Design and Limitations

The FiC-MPI library is designed to be available with standard C/C++ supported by Xilinx's Vivado HLS, without a special code transformer before compilation, unlike [25][68]. The library is written in pure C++ in Vivado HLS and is designed to be synthesizable into HDL in the Vivado HLS while maintaining compatibility with standard MPI libraries. All APIs are available from the user's HLS code with the library header (ficmpi.h). The library is also designed to manage the required hardware resources in FPGA and is adaptable to the STDM network used on FiC and M-KUBOS clusters. Since the restriction on HLS and hardware resources is unavoidable, the library currently supports a subset of MPI primitives to support commonly well-used APIs in MPI applications.

5.1.3 Supported MPI APIs

Table 5.1 shows supported MPI APIs by the FiC-MPI. The library covers minimum primitives for peer-to-peer and collective communication of the MPI specification. Due to HLS's execution model restriction, the FiC-MPI only supports blocking communication APIs.

5.1.4 Feature for Efficient Hardware Synthesis

Figure 5.1 shows an example of simple multi-node π calculation using FiC-MPI with Vivado HLS C/C++ code. As a unique feature of the FiC-MPI library, APIs can be invoked with user-specified data widths for efficient hardware synthesis. For example, `MPI_Bcast()<uint32_t>` in line 41 of Figure 5.1 specifies the data type to generate hardware for 32 bits unsigned integer type data. This feature allows the programmer to implement appropriate bit-width hardware synthesized by the HLS compiler.

Arbitrary vector types in the general MPI library are also supported. This arbitrary vector type can be defined by `MPI_Type_vector()`, and the vector type can be used in the MPI data type specification (MPI_Type) argument. This capability makes it possible to handle only a part of the data specified in an array. For example, when a user wants to exchange a part of a huge size data array

Table 5.1: APIs provided by FiC-MPI library

Type	API	Description
Program control	MPI_Init	MPI program init.
	MPI_Finalize	MPI program termination
	MPI_Comm_rank	Get current process rank
	MPI_Comm_size	Get process size
Peer-to-Peer communication	MPI_Send	P2P data send (blocking)
	MPI_Recv	P2P data receive (blocking)
Collective communication	MPI_Bcast	Broadcasting data send/receive
	MPI_Scatter	Scattering data send/receive
	MPI_Gather	Gathering data
	MPI_Allgather	All gathering data
	MPI_Reduce	Reduce data
	MPI_Allreduce	All reduce data
	MPI_Alltoall	Exchange data to all ranks
Misc	MPI_Type_vector	Define vector type
	MPI_Cart_create	Define Cartesian topology
	MPI_Cart_get	Get current location on Cartesian topology
	MPI_Cart_shift	Shift location on Cartesian topology

with another rank, the custom vector type can be selected for a part of the array to be exchanged. It contributes to reducing the required hardware resource and communication efficiency.

5.1.5 Integration with STDM Network

To make FiC-MPI compatible with STDM networks, the node identification number on MPI communication, *Rank*, is mapped to the slot number on STDM communication. As shown in line 23 of Figure 5.1, the application can recognize its rank and available slot number by giving the identification number of the node corresponding to the rank as an argument when the MPI library is initialized in `MPI_Init(node_id)`. The given `node_id` sets this identification number on the board or gives an arbitrary ID before the application is executed. Figure 5.2 shows an example when communicating three nodes with Rank0, Rank1, and Rank2 in the STDM network.

- At Slot0, Rank0 is a sender, and Rank1 and Rank2 are receivers. Rank0 and Rank2 are not physically connected, so the Rank0 data directly transfers to Rank1, and Rank2 receives the data via Rank1 STDM switch.

```
#include <ap_int.h>
#include "ficmpi.h"

#define  MPI_WORLD_SIZE 4

void mpi_app(
    uint8_t bid, uint32_t iter,
    float result[1],
    ficbus_T fic_in[128],   // STDM network in
    ficbus_T fic_out[128]   // STDM network out
)
{
#pragma HLS INTERFACE s_axilite port=bid
#pragma HLS INTERFACE s_axilite port=iter
#pragma HLS INTERFACE s_axilite port=result
#pragma HLS INTERFACE s_axilite port=return
#pragma HLS INTERFACE axis port=fic_in
#pragma HLS INTERFACE axis port=fic_out

    MPI_Status p_st;
    int rank, size;

    MPI_Init(bid);  // Init MPI
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int interval[1];
    int n[1];

    interval[0] = iter;
    n[0] = 0;

//-----------------------------------------------------
// Usage:
// int MPI_Bcast<DATA_T>(
//      DATA_T *sendbuf, DATA_T *recvbuf,
//      const int count, MPI_Datatype datatype,
//      int root_rank, MPI_Comm comm,
//      ficbus_T *fic_in, ficbus_T *fic_out)
//-----------------------------------------------------
    MPI_Bcast<uint32_t>(
        interval, n, 1, MPI_INT32_T, 0, MPI_COMM_WORLD,
        fic_in, fic_out
    );

    const double h = 2.0 / n[0];

    double sum[1];
    double pi[1];
    sum[0] = 0.0;
    pi[0] = 0.0;

    for (int i = rank; i < n[0]; i += size) {
        const double x0 = i * h - 1.0;
        const double x1 = (i + 1) * h - 1.0;
        const double f0 = 2.0 / (1.0 + x0 * x0);
        const double f1 = 2.0 / (1.0 + x1 * x1);
        sum[0] += 0.5 * (f0 + f1) * (x1 - x0);
    }

//-----------------------------------------------------
// Usage:
// int MPI_Reduce<DATA_T>(
//      DATA_T *sendbuf, DATA_T *recvbuf, const int count,
//      MPI_Datatype datatype,
//      MPI_Op op, int root_rank, MPI_Comm comm,
//      ficbus_T *fic_in, ficbus_T *fic_out)
//-----------------------------------------------------
    MPI_Reduce<double>(
        sum, pi, 1,
        MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD,
        fic_in, fic_out
    );

    if (rank == 0) {
        result[0] = pi[0];
    }

    return;
}
```

Figure 5.1: An example HLS application with FiC-MPI library

- At Slot1, Rank1 is a sender, and Rank0 and Rank2 are receivers. Both Rank0 and Rank2 are directly connected to Rank1 so that the Rank1 data directly transfers to Rank0 and Rank2.
- At Slot2, Rank2 is a sender, and Rank0 and Rank1 are receivers. Rank2 and Rank0 are not physically connected like Slot0, so Rank2 data directly transfers to Rank1, and Rank2 receives data via Rank1 STDM switch, respectively.

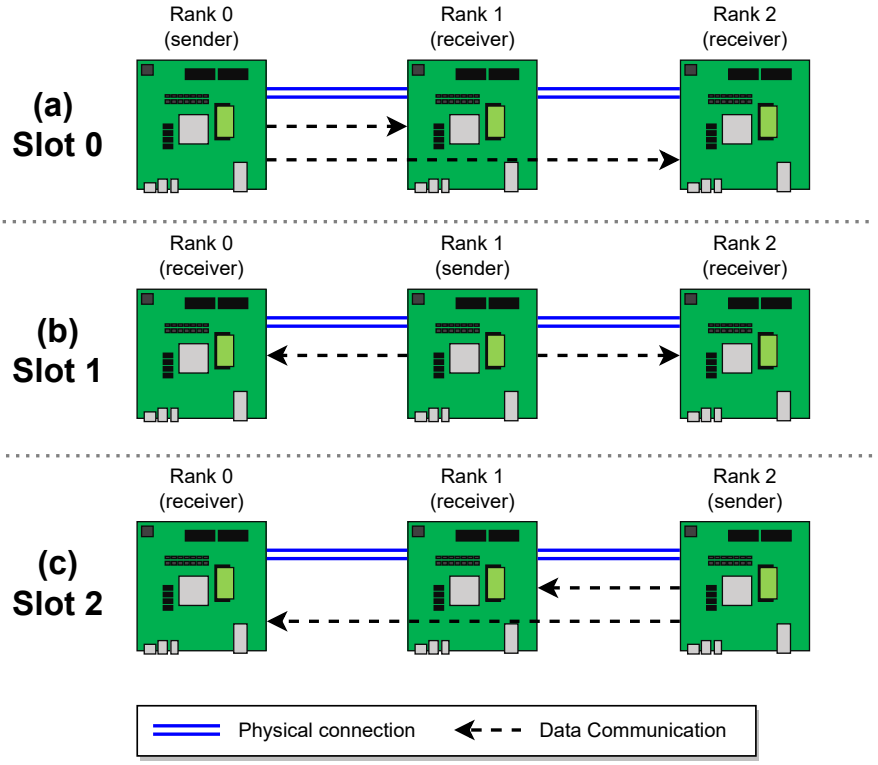


Figure 5.2: Three nodes MPI communication on STDM network

The routing table of the STDM switch must be set up before executing an application. The most straightforward routing table setup that allows different ranks to send and receive data to and from each other is *broadcasting*.

Figure 5.3 shows this broadcasting configuration in the STDM switch of Rank1 node in Figure 5.2 at Slot1. In this way, Rank1's output (HLS logic out) is sent to other ranks, and all other ranks receive the data. This broadcasting can be achieved by using the multicast function of STDM switches, and the current FiC-MPI implementation assumes this broadcast method as the default network.

In the STDM network, increasing the number of slots has a side effect of increasing communication delay. Hence, broadcasting is unnecessary if it is

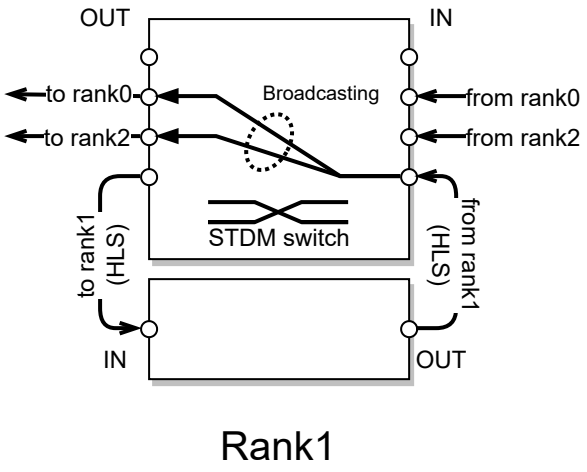


Figure 5.3: Broadcasting of the STDM Switch in the Rank1 at Slot1 in Figure 5.2

obvious that arbitrary ranks will not communicate with each other. Moreover, the communication delay on the STDM network can be optimized by optimization algorithm [46] to find the optimal number of slots for communication.

5.1.6 MPI Communication with STDM Network

The transferred data corruption with the STDM network is protected by the ECC scheme provided by Xilinx Aurora IP. However, the MPI library needs to handle burst data transfer and error handling such as data miss-delivery and non-reachable.

Figure 5.4 shows a typical MPI communication sequence in `MPI_Send()`. First, the sender rank sends a synchronization request to the receiver rank, and then the receiver rank prepares for data receive and returns the sync acknowledge packet to the sender rank. After synchronization, the sender rank sends a header and data packets to the receiver rank. After finishing data packets transferred, the library checks all data that have been received and sends a fin packet to the sender rank to end the communication.

Since the STDM network is not guaranteed to transfer data in order if they use different slots, the MPI library needs to handle correct incoming data orders from the multiple ranks. Figure 5.5 shows the STDM network packet format used in the FiC-MPI. All packets commonly have message sequence ID (Seq ID) and source rank (Src rank) in the command field; the incoming data sequence is checked with a data sequence number of the sequence ID by the library.

The first packet (Seq ID=0) of the MPI communication is dedicated to the header packet. This packet contains information such as the total number of data of the

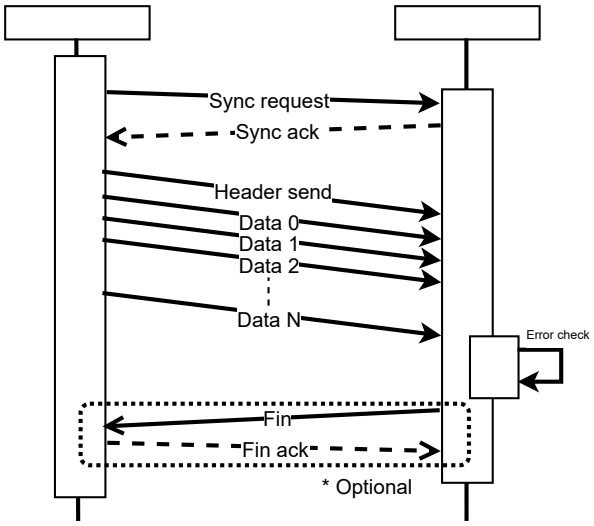


Figure 5.4: Communication Sequence of MPI_send()

message (data count), MPI data type, MPI tag ID, request ID (an ID determining the message is a response for which request), destination rank, source rank, and message type.

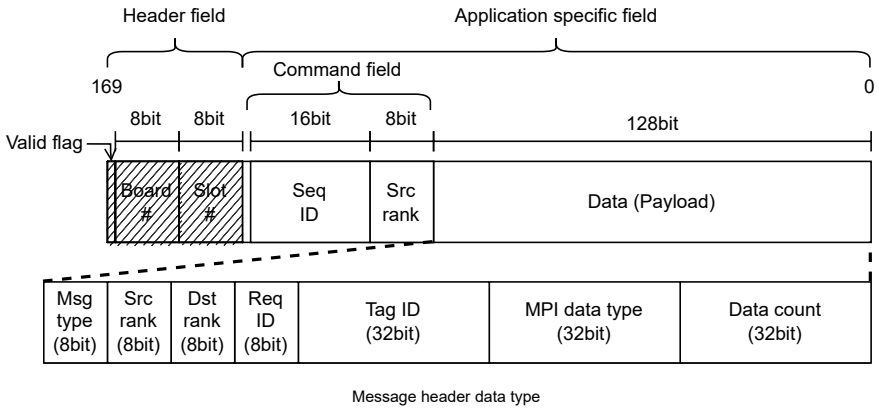


Figure 5.5: FiC-MPI Packet Format

Figure 5.6 shows the reordering process for an incoming packet to guarantee incoming data order. A dedicated ordering buffer of each rank is prepared like a mailbox, and the incoming packet is stored in the ordering buffer according to Seq ID and source rank (Src rank). According to the consumer pointer, the library reads out the valid packets from the target rank's ordering buffer and waits if expecting

incoming packet has not come yet. By this mechanism, the FiC-MPI achieves reliable message exchange on the STDM network.

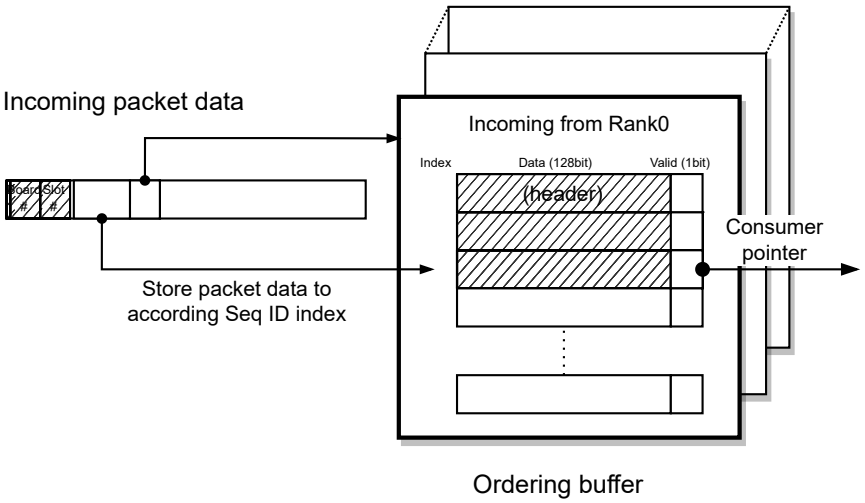


Figure 5.6: Structure of the Packet Re-ordering Buffer

Compatibility Features

To enhance the portability of MPI applications, it also provides minimal support for Cartesian topology functions supported by standard MPI libraries. This feature can generate Cartesian topologies up to three dimensions to limit the number of hardware resources. In addition, since FiC-MPI currently assumes the broadcast network as the default network mentioned before, the mapping rank to Cartesian topology in ascending order of node number. Thus, there is room for optimal mapping based on the cluster's physical topology in the future.

5.2 Application Development with FiC-MPI

5.2.1 Debugging Multi-FPGA Application with FiC-MPI Simulator

In typical FiC and M-KUBOS application development, designers had to spend a lot of time debugging and testing. As for the debugging environment of FPGA applications in HLS using conventional C/C++, the C/RTL co-simulation environment is provided by the vendor tool, but these are targeted at a single FPGA. In particular, in a multi-FPGA application that uses multiple FPGAs, there is no environment for functional level debugging and testing, and it is always necessary to implement on the actual FPGA platform before debugging and testing. For this reason, the application designer is forced to debug the FPGA hardware design more than the debugging of the application itself, which poses a problem in the productivity of the application. Therefore, in order to support parallel programming on a multi-FPGA environment using HLS, FiC-MPI has developed a simulation environment FiC-MPI Simulator that can perform application debugging and testing at the functional level. This FiC-MPI Simulator provides an environment for executing and simulating multi-FPGA applications developed with HLS and FiC-MPI on a PC environment to overcome the challenges of multi-FPGA application productivity. Using this simulation environment makes it possible to test and debug multi-FPGA applications without implementing them on the actual FPGA, which is expected to improve application productivity. Figure 5.7 shows multi-FPGA application development flow with FiC-MPI. In the application development flow with FiC-MPI, the designer first tests and debugs an application with FiC-MPI Simulator. To test an application on FiC-MPI Simulator, compiling application code with a standard C++ compiler (e.g., GCC, LLVM/Clang) with a FiC-MPI simulation wrapper. In the FiC-MPI Simulator environment, the application is implemented as a multiprocess application that communicates with IPC socket. This simulation environment allows an application designer to debug an application with standard application debugger tools (e.g., GDB). It is possible to improve multi-FPGA application debuggability significantly. After testing and debugging with the FiC-MPI Simulator environment, the application designer implements an application with standard FPGA implementation flow.

5.2.2 Comparison between Existing Application Development Method

In conventional multi-FPGA application development on FiC and M-KUBOS clusters, there are many hurdles application designers need to override described in the previous section. In this section, we will discuss multi-FPGA application development productibility, comparing the existing approach and with the FiC-MPI. Figure 5.8 shows a comparison between the conventional multi-FPGA design flow and the flow with the FiC-MPI. Generally, the application design flow for FiC and

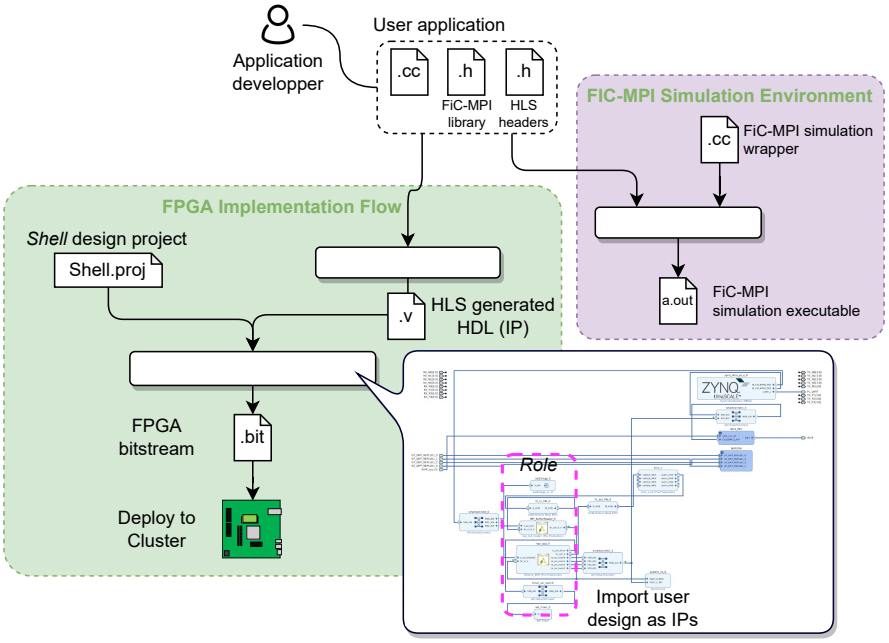


Figure 5.7: Multi-FPGA Application Development Flow with FiC-MPI

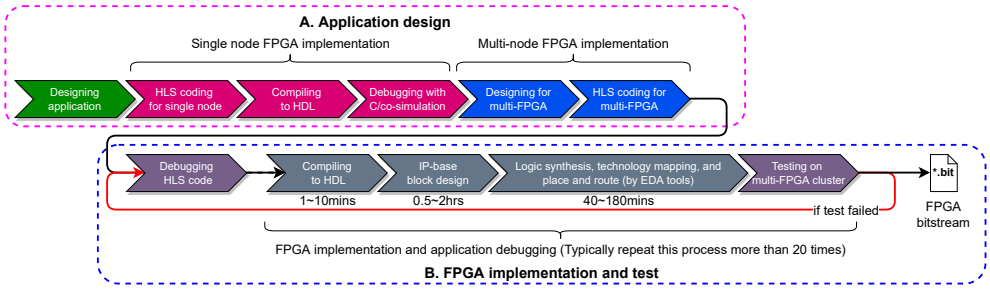
M-KUBOS's multi-FPGA applications is the following steps: (A) Application design and (B) FPGA implementation and test.

(A) is a step for designing the multi-FPGA application kernel. In the conventional flow, an application designer first creates the application for a single FPGA using HLS with C/C++. After the single FPGA application is created, they test with HLS application debug environments, such as functional simulation and co-simulation. Once the single FPGA application is debugged, the application designer modifies the application for multi-FPGA. In this step, the application designer must design specific communication protocols among FPGAs to implement a multi-FPGA application apart from the application kernel. Moreover, the application designer can not test the multi-FPGA application at this step because there is no debugging environment supporting multi-FPGA communication provided. By contrast, in the flow with the FiC-MPI, the application designer can design the application as an MPI application that supports parallel programming from the beginning. The designer can test and debug the application with the FiC-MPI Simulator at a function level without implementing the application into a real FPGA hardware platform.

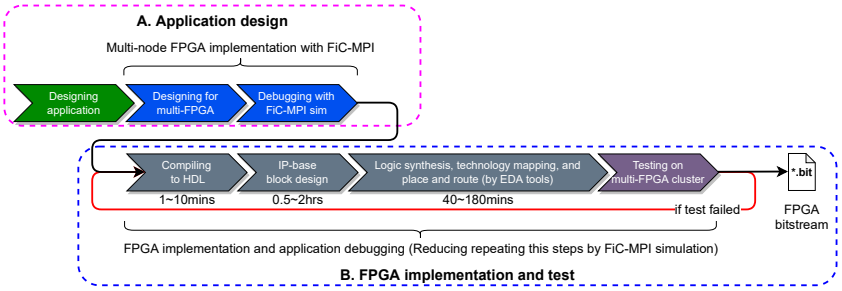
(B) is a step for FPGA implementation with the FPGA EDA tools. The designer implements the application with IP-based FPGA design flow and logic synthesis for the target FPGA. Typically, the application implementation process on the FPGA requires a long lead time. For example, logic synthesis typically takes a few ten

minutes to hours. After the implementation process for the FPGA, the designer deploys the application to the cluster and tests them.

FiC-MPI allows application designers to design their applications as multi-FPGA applications with MPI from the beginning. Therefore, it is considered that the design cost and the design time of the application can be reduced. Also, when testing and debugging an application, FiC-MPI Simulator allows the application designer to perform functional tests of the application without using the actual FPGA platform. Therefore, the number of implementations required for testing and debugging on the actual FPGA platform can be reduced. According to Google's related research on software productivity [78], average C/C++ developers are executing application builds more than 200 times per month. For example, if 10% of 200 builds need to run tests after coding, the application designer will need to repeat the FPGA implementation more than 20 times. Without the FiC-MPI Simulator, the application designer would need to spend an average of 40 minutes (for FiC and M-KUBOS) or more on the FPGA implementation process (e.g., logic synthesis, etc.) for each test. In contrast, the FiC-MPI Simulator can create the test environment in about 5 seconds using a C/C++ compiler. This significantly reduces the lead time required to test and debug applications. Thus, FiC-MPI significantly increases the productivity of multi-FPGA applications by more than 100 times.



(a) Conventional multi-FPGA application development with HLS and IP-based FPGA design flow



(b) Multi-FPGA application development with FiC-MPI, HLS, and IP-based FPGA design flow

Figure 5.8: Comparison between Multi-FPGA Application Development Flow

5.3 Evaluation

5.3.1 FiC-MPI Performance on M-KUBOS Cluster

This subsection describes the evaluation of the communication performance of FiC-MPI. In this evaluation, we used a single STDM link on the six nodes M-KUBOS cluster shown in Figure 3.8, and evaluated the communication throughput and latency at each data transfer length. The evaluation tested the commonly used one-to-one and collective communication patterns by MPI applications.

One-to-one communication pattern: PingPong

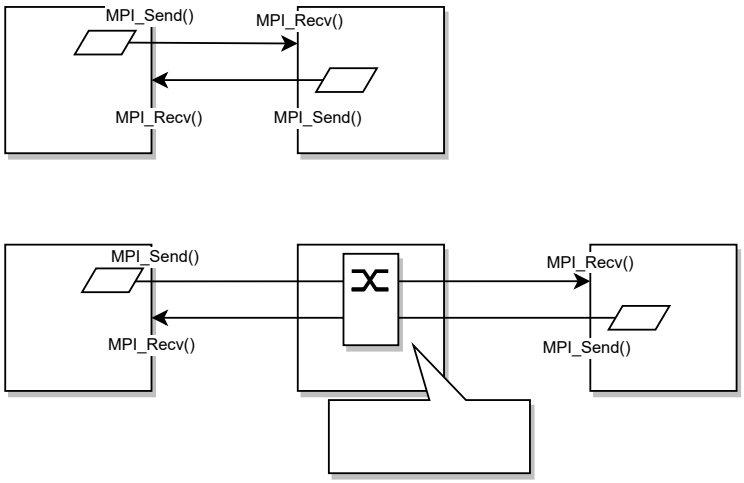


Figure 5.9: PingPong Communication Example

PingPong communication pattern is shown in Figure 5.9. PingPong is a one-to-one communication pattern between two ranks, in which data is sent from one rank by `MPI_send()`, received by the other rank by `MPI_recv()`, and then sent back to the original rank by `MPI_send()`. The throughput for each data transfer length is shown in Figure 5.10 and the latency is shown in Figure 5.11.

The throughput of one-to-one communication in FiC-MPI reaches a ceiling of approximately 299 MB/s at a data transfer length of 1 MB, which is considered the upper limit of the current FiC-MPI data transfer capability. The theoretical transfer capacity per lane of the STDM network in M-KUBOS is approximately 1.1 GB/s, and the practical data transfer throughput is 510 MB/s with two slots measured in related work [49]. The current payload efficiency is 75% as shown in Figure 5.5, and the estimated maximum transfer throughput is approximately 383 MB/s. Based on the

measured throughput, the transfer efficiency of the current FiC-MPI implementation is 78% of the estimated maximum network transfer performance.

Since the current implementation of FiC-MPI uses a broadcast on the STDM network, the effective bandwidth that can be used per node is decreased along with the increase of the slots number. Comparing two nodes (2 slots), four nodes (4 slots), and six nodes (6 slots) cases, the transfer performance decreases to 55% (4 slots) and 38% (6 slots) at 1 MB data transfer length. The result shows those huge performance penalties by increasing the number of slots. It needs to consider applying the more sophisticated network usage instead of broadcast or using optimization techniques to reduce the required number of slots proposed in [46]. Figure 5.11 shows that the latency increases in proportion to the transfer length, exceeding 1 ms around 64 kB and reaching 18 ms at 1 MB.

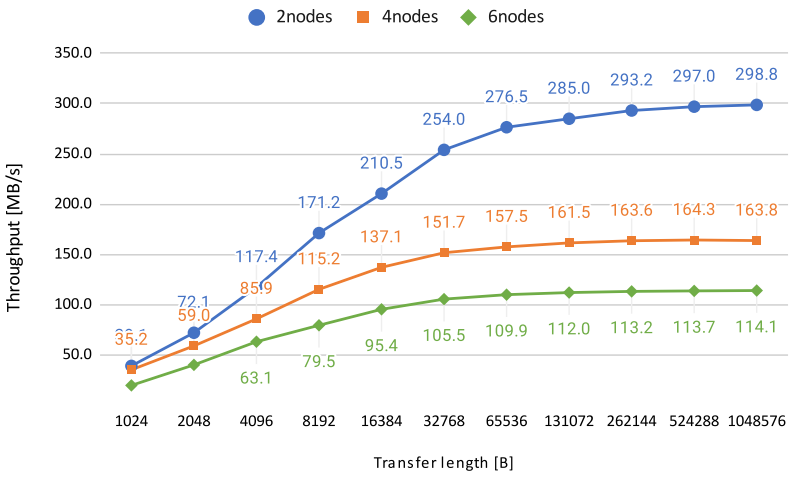


Figure 5.10: PingPong Throughput at a Transfer Size

Figure 5.12 shows that PingPong communication latency from Rank0 to Rank1∼Rank5 with six nodes configuration at each transfer data length. The result shows the communication latency increases with the number of nodes (hops). In the 0 to 5 case, where the communication passes through 4 nodes, the latency increased by about 2 percent compared to the 0 to 1 case.

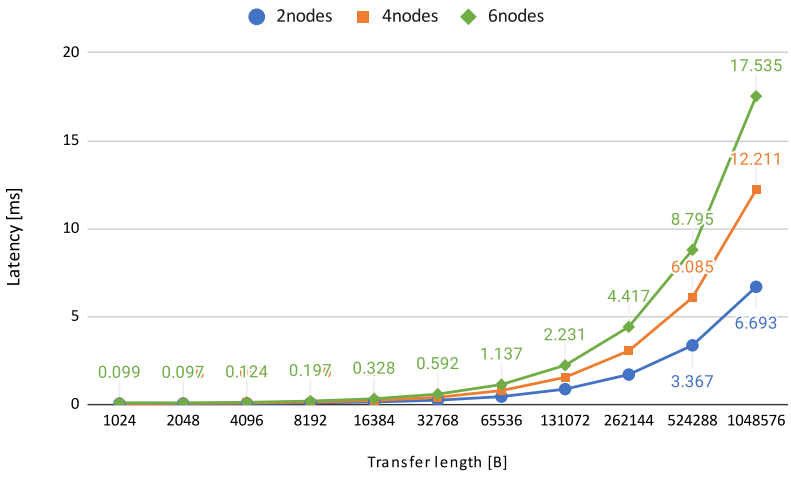


Figure 5.11: PingPong Latency at a Transfer Size

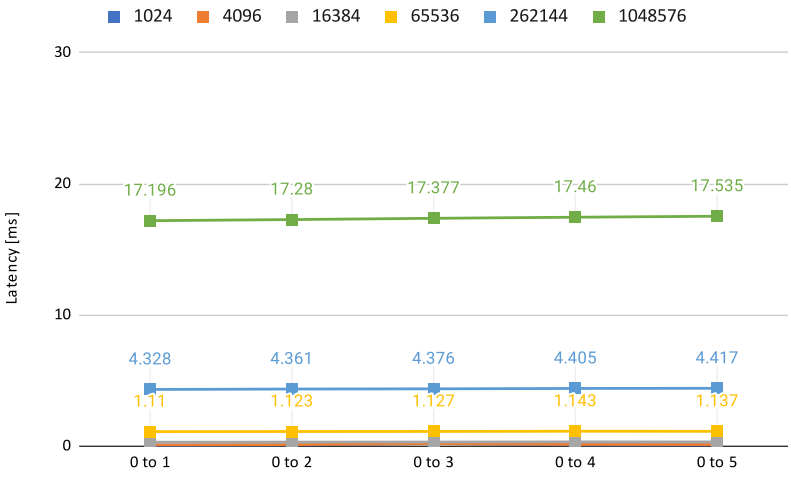


Figure 5.12: PingPong Communication Latency at a Different Hops

Collective communication pattern: AllReduce

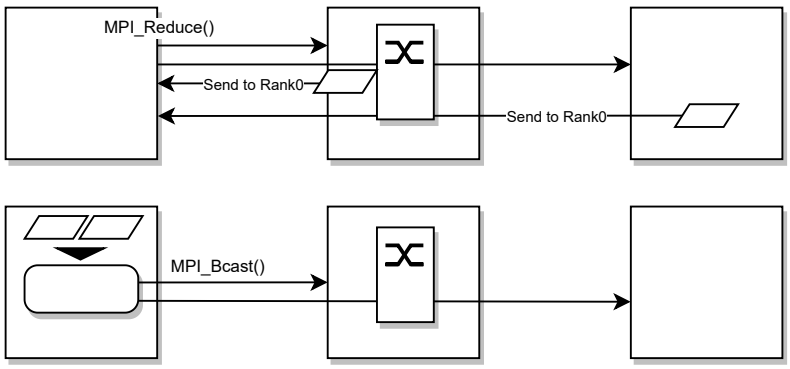


Figure 5.13: AllReduce Communication Example

AllReduce communication pattern is shown in Figure 5.13. It is a collective communication that collects data from all ranks, performs an aggregate operation and sends back the result to all ranks. The implementation of FiC-MPI's `MPI_AllReduce()` aggregates all rank's data to Rank0 by `MPI_Reduce()` operation, then performs the aggregate operation such as accumulations on Rank0, and then broadcasts the results to all ranks by `MPI_Bcast()` (broadcasting). Figure 5.14 shows the observed throughput and Figure 5.15 shows its latency.

Compared to communication throughput in PingPong, AllReduce achieved similar throughput until 16 kB data length with two nodes (2 slots) communication. However, the throughput is saturated at 215 MB/s in more than 32 kB data length due to the AllReduce requiring additional clock cycles to a reduction operation specified by `MPI_Op` argument of the `MPI_AllReduce()` during the data transfer. When the number of nodes increases, the throughput decreases to 47% compared to the two-nodes case in the six-nodes configuration. Latency was also up to 6.4x longer in the six-nodes configuration compared to the two-nodes configuration. This throughput degradation and longer latency are possibly caused due to an increasing number of shared slots and required times of communication synchronization. The current node synchronization algorithm in the FIC-MPI is being serialized, so further design review can mitigate this communication overhead.

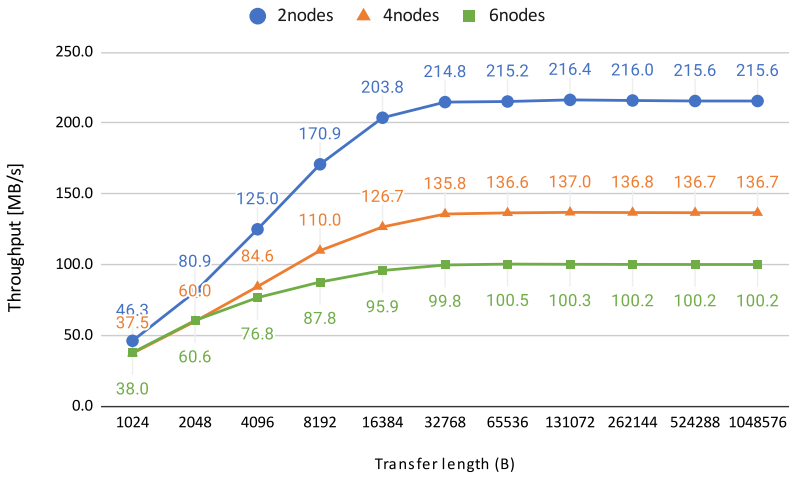


Figure 5.14: AllReduce Throughput at a Transfer Size

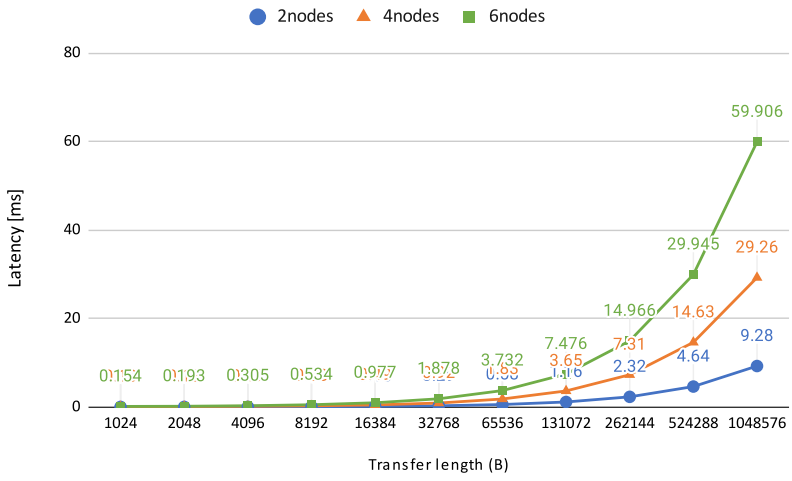


Figure 5.15: AllReduce Latency at a Transfer Size

5.3.2 Case study: Porting Himeno Benchmark with FiC-MPI

Himeno Benchmark

The Himeno Benchmark (Himeno-BMT) [79] is a general-purpose numerical benchmark designed to evaluate the performance of the incompressible fluid analysis code published by RIKEN. It measures the processing speed of the main kernel loop when solving the Poisson equation by Jacobi's iterative method. In this case study, we tried to port the Himeno benchmark with FiC-MPI to the M-KUBOS cluster and evaluate its porting work practicality, performance, and scalability.

Porting the Himeno-BMT to Multi-FPGA Application

The Himeno benchmark is written in Fortran or C with OpenMP and MPI parallelization methods. The porting work is based on the MPI version code written in C (cc_himenoBMTxp_mpi) and modifying the code that needed to be supported for HLS and FiC-MPI. The original C code statically allocates the matrices for arithmetic operations in the stack area using C arrays, so it exceeds the amount of BRAM/URAM resources available in the FPGA and cannot be ported with HLS. Therefore, in this implementation, these matrices are statically allocated using DRAM instead of BRAM, and the part required for kernel calculation is loaded on demand. In addition, MPI calls in the original code were changed to the corresponding API of FiC-MPI. The rest of the arithmetic kernel is original, without explicit optimization for FPGA.

Debugging in this porting work was done using the FiC-MPI simulator. We first build the HLS kernel code with the FiC-MPI simulator for the PC application. Then, we checked the HLS code correctness by comparing the original MPI code results. After that, we implemented the HLS kernel for the real M-KUBOS cluster system and checked the calculation correctness. This method reduced the debugging times on the real M-KUBOS cluster hardware and eliminated the burden on the application porting process. In this porting, we use Xilinx Vivado HLS 2020.1 for HLS and Xilinx Vivado 2020.1 for FPGA logic synthesis. The FPGA design is based on M-KUBOS's fully embedded (F) Shell design described in Section 3, and we obtained the FPGA design running at 100 MHz.

Performance and Scalability

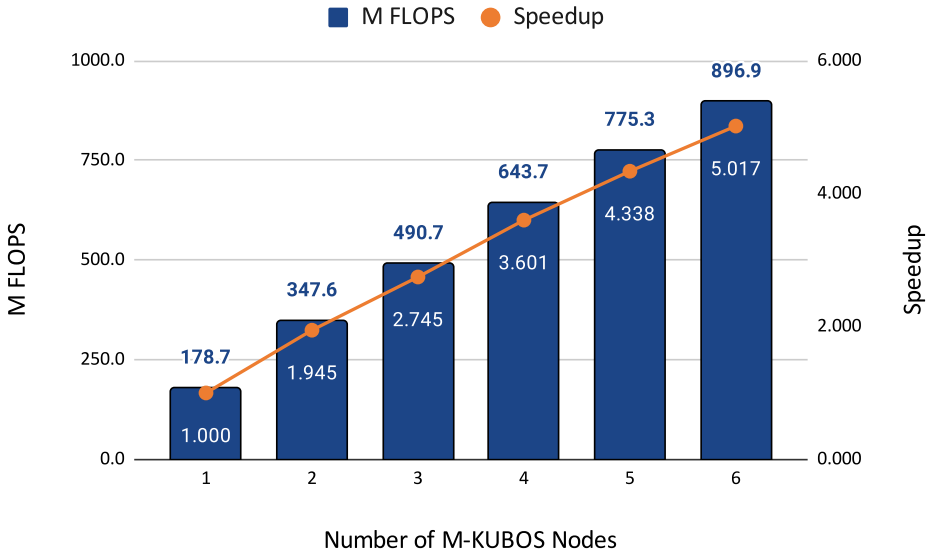


Figure 5.16: Himeno benchmark performance on M-KUBOS cluster

The problem size in the Himeno benchmark used in the evaluation is M (128x128x256), and the number of rotation loops (nn) is nn=50, which is stable in performance. The floating-point precision is 32bit which was originally used in the Himeno Benchmark. Figure 5.16 shows the performance measured by the number of nodes from one to six. The bar graph shows the performance values of MFLOPS, and the line graph shows the performance increase rate based on one node. The horizontal axis is the number of execution nodes. A single M-KUBOS node achieved 178.7 MFLOPS. This performance is 2.2 times greater than the 82.84 MFLOPS, a reference performance value with the Himeno Benchmark on Intel Pentium III 600 MHz, and 1.2 times greater than 148.5 MFLOPS, a performance value on the PS (ARM Cortex-A53 1.2 GHz with gcc 7.3.0 -O3 optimization).

This result is no surprise due to the following reasons: 1) M-KUBOS runs at only 100 MHz, which is x10∼x20 times lower operation frequency than state-of-art CPUs. 2) The application kernel is not well optimized for FPGA architecture for highly parallelized. In fact, the design used only 4% DSPs and 18% LUTs compared to occupying 84% of BRAMs. The low DSP/LUT resource utilization indicates room for improvement in implementation efficiency. 3) The Himeno Benchmark is a memory performance-intensive benchmark.

Therefore, we believe that performance can be dramatically improved by optimizing the HLS design to take full advantage of its parallelism, by using such

as an optimized 3D array computation [80], and arbitrary precision fixed-point data types. We observed the cluster performance increment according to the increase of M-KUBOS nodes. With four nodes, the performance was 643.7 MFLOPS (3.6x compared to a single node) instead of 531.4 MFLOPS (3.5x compared to single-core) using Open MPI v2.1.1 on four cores Cortex-A53 of the PS. The result shows that the FiC-MPI achieved similar performance scalability to conventional MPI implementation. The maximum performance was 896.9 MFLOPS with six nodes, and the performance increment was 5.0x.

Processing Time Breakdown

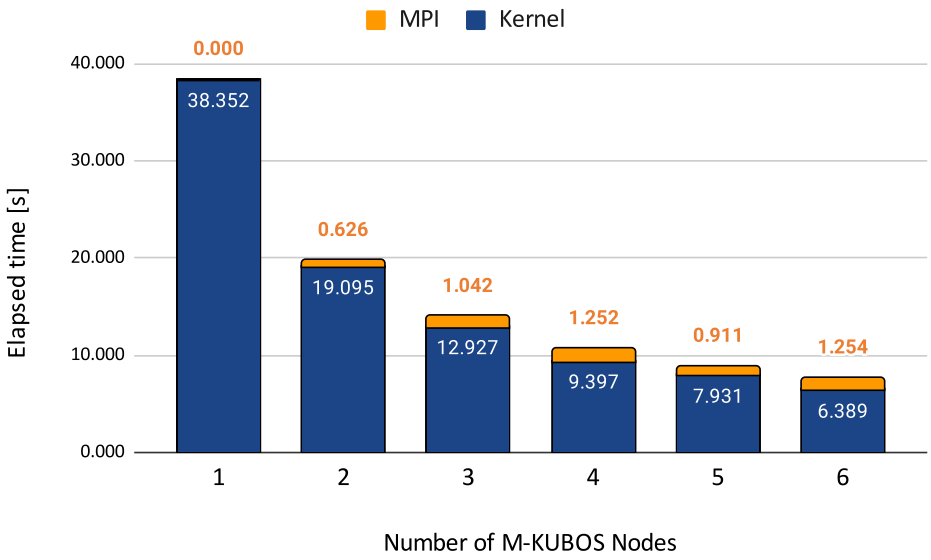


Figure 5.17: Processing time breakdown of the Himeno Benchmark

Figure 5.17 shows the breakdown of kernel processing time and communication processing by MPI when running the Himeno Benchmark on each number of nodes. The processing time measuring the arithmetic kernel (Jacobi iteration method) and the MPI processing part was obtained by the AXI Timer IP in the PL that synchronized with the operating frequency (100 MHz). Figure 5.17 shows that the processing time of the MPI part increases along with the number of nodes increases. The communication delay of the STDM causes this processing time to increase network along with an increasing number of nodes, and it increases the latency of the node synchronization process performed by FiC-MPI. The percentage of MPI processing time is approximately 16% at six nodes.

This result suggests that the performance scalability of the M-KUBOS cluster

is limited due to communication delays caused by the increase in the number of slots on the STDM network. This limitation can be managed by applying network optimization, such as optimizing the required number of slots according to the application and reducing the number of slots between nodes that do not need to communicate. Although the performance and scalability obtained from the Himeno Benchmark are not high, the case study shows that even applications written for CPUs can be ported and executed as applications for a multi-FPGA cluster like the M-KUBOS easily by using FiC-MPI.

5.4 Related Work

Today, a typical example of a parallel programming environment for multi-FPGAs is OpenCL [51]. OpenCL is a specification for parallel computing developed by the Khronos Group that supports heterogeneous environments combining CPUs with GPUs and FPGAs. For FPGAs, OpenCL is supported by Xilinx's SDAccel (Vitis unified software platform) [52] and Intel's FPGA SDK for OpenCL [53]. They target CPU-centric multi-FPGA environments with x86-based CPUs, and FPGA cards are connected via PCIe.

Message Passing Interface (MPI) is a well-known parallel programming environment for distributed memory architecture. In related research [81] proposes a parallel programming environment in which a software processor is deployed on an FPGA and implemented MPI on its soft CPU core. Also, in related research [36] constructs a multi-FPGA system that can be used with MPI. This system employs SoC-type FPGA Xilinx Zynq-7020 for the FPGA node and runs Open MPI [82] and MPICH [83] on an embedded Linux environment running on the Zynq's ARM core.

As a bare-metal approach to MPI implementation for multi-FPGA systems that do not use soft-core CPUs or embedded CPUs, related researches [25][26] are proposes a compiler-based code transpiration approach. In this method, a special code transpiler or transformer is used to analyze MPI code to extract MPI kernel code and interpret it for generating a synthesizable HLS code for each FPGA node.

The FiC-MPI developed in this study is a toolless and straightforward approach compared to related research. Therefore, it is more compatible with the standard HLS application development workflow, which allows application designers to implement multi-FPGA applications using only a standard FPGA development toolchain.

5.5 Summary

In this chapter, we considered the use of MPI, a parallel programming environment commonly used in distributed memory system environments, to improve the application development environment in stand-alone multi-FPGA systems such as FiC and M-KUBOS. In this study, we have developed the MPI library FiC-MPI, which can be used in the FPGA application development environment using C/C++ with standard HLS. FiC-MPI enables application designers to implement FPGA applications with MPI parallel programming, which greatly reduces the application design and implementation process on a stand-alone multi-FPGA environment.

In addition, the FiC-MPI Simulator environment was developed to provide an application testing and debugging environment for the multi-FPGA environment on FiC and M-KUBOS. The FiC-MPI Simulator environment allows application designers to test and debug MPI applications without using actual FPGA platforms. Compared to conventional HLS application development methods on FiC and M-KUBOS, the FiC-MPI Simulator environment can improve the efficiency of application debugging by more than 100 times, showing that it can significantly improve application productivity.

To demonstrate the practicality of FiC-MPI, as a case study, we ported a general-purpose numerical benchmark implemented by MPI, *Himeno Benchmark* (himeno-BMT), as a multi-FPGA application for M-KUBOS using FiC-MPI. Through this case study, we showed that existing MPI applications could be ported and executed as multi-FPGA applications, demonstrating application development's practicality with FiC-MPI.

6

Conclusion and Future Work

6.1 Conclusion

Multi-access Edge Computing (MEC) is a new methodology of edge computing in the 5G mobile network era that utilizes computing resources on edge. In MEC, latency-sensitive and computing power required applications, such as artificial intelligence (AI) and image recognition, are expected to be one of the primary tasks. Since MEC is a power and space restricted environment, there are demands for a compact, flexible, and power-efficient computing platform to support such various applications.

This thesis focused on a stand-alone multi-FPGA system as a computing platform for MEC. An FPGA achieves high flexibility and power efficiency through reconfigurable architecture and hardware-based data processing; the latest high-performance and feature-rich FPGAs are expected to be used as accelerators for general-purpose computing.

The FiC project developed the FiC and its successor M-KUBOS cluster, a stand-alone multi-FPGA system aimed at applying on MEC. A stand-alone multi-FPGA system is constructed with directly connected FPGA nodes; it is expected to realize a flexible and energy-efficient computing platform compared to traditional server-based systems in power and space constrained environments. The FiC and M-KUBOS clusters aim to prove that a multi-FPGA system delivers scalable computing power and high energy efficiency on edge computing. To enhance the usability of these stand-alone multi-FPGA systems in MEC applications, ensuring the platform manageability and programmability from the various application

platforms and programming environments is essential. Thus, this thesis contributed to establishing a base architecture that improves the platform manageability and programmability of the multi-FPGA systems such as FiC and M-KUBOS.

The first proposal of this thesis is a cloud-computing-inspired multi-FPGA hardware platform management system called *FiC-RFC*. FiC-RFC enables a multi-FPGA system to be accessible and manageable from MEC applications; it can be supported by various resource management platforms such as *FiC-RM* and applications by using RESTful APIs, a programming semantics widely used in cloud computing. Several AI-based image recognition applications have been implemented with FiC-RFC for the FiC and M-KUBOS clusters; they achieved 2-6 times better energy efficiency and 1.1-12 times better performance than the state-of-art CPU implementation.

The second proposal in this thesis is to provide a programming environment for multi-FPGA architecture. Since FiC and M-KUBOS clusters employ a *bare-metal* approach programming model, applications (an offloading kernel for FPGA) need to design with the FPGA vendor toolchain. Such default development tool for FPGA does not support parallel programming, which is becoming a hurdle for application development on the stand-alone multi-FPGA architecture. In this proposal, we focused on Message Passing Interface (MPI), which is often used in parallel programming of distributed memory architectures, and developed a tool-independent MPI library for HLS, called *FiC-MPI*, to improve application programmability in stand-alone multi-FPGA architectures. FiC-MPI is an MPI library that can be directly used in FPGA kernel development with C/C++ in HLS; it eliminates the difficulty of implementing parallel programming applications using multi-FPGA architecture. Moreover, the FiC-MPI simulator environment provides a test and debug environment for multi-FPGA applications. This environment eliminates the FPGA implementation process for testing and debugging using the real FPGA platform and improves the productivity of the multi-FPGA application. FiC-MPI library provides compatible APIs in standard MPI implementations and allows for porting existing MPI-based applications to the multi-FPGA architecture. To demonstrate the feasibility of the FIC-MPI, ported the Himeno Benchmark, a general-purpose numerical benchmark application written in MPI, to the M-KUBOS system with FiC-MPI as a case study.

The proposed multi-FPGA platform management system and application programming environment enhance the feasibility of multi-FPGA systems adoption on MEC, and the systems support the deployment of many services and expand the use cases of MEC.

6.2 Directions for Future Work

Future work includes the following considerations:

6.2.1 Improvement for FiC-RFC architecture

Support for real-time application control model

RESTful is non-interactive communication with HTTP requests such as POST and GET method. Therefore, it isn't easy to use for realtime-ness applications such as drone control and autonomous drive control. These applications are expected to do real-time data camera image analysis with FPGAs, and support for real-time communication is essential. For the supporting realtime-ness communication with the HTTP request, employing the Websocket for the ficwww is considered.

Support for remote debugging via HTTP

In the current FiC-RFC system, RESTful communication is not fully supported for FPGA debugging. Employing Websocket communication via HTTP, it can encapsulate XVC communication between JTAG and hardware server in the remote location, which enhances the usability of debugging the system.

6.2.2 Improvement for FiC-MPI

Improvement of payload efficiency of FiC-MPI

The current packet format payload used by FiC-MPI is approximately 75%, and this payload efficiency can be improved by reviewing the design.

Optimizing communication flow of FiC-MPI

The current FiC-MPI communication control is not fully optimized, resulting in a significant decrease in data transfer throughput when the number of nodes increases in group communication. Therefore, optimizing communication control flow, such as synchronization between nodes, is necessary.

Communication performance improvement

The current FiC-MPI communicates using only one lane, so the hardware performance is not fully utilized. Therefore, there is still room for increasing communication performance with supporting lane aggregation, which uses multiple lanes simultaneously.

Bibliography

[1] Deloitte, “Global mobile consumer trends: Second edition”, Analysis, 2017.

[2] Isla McKetta, *Ookla Insights Articles - Growing and Slowing: The State of 5G Worldwide in 2021*, 2021. [Online]. Available: https://www.ookla.com/articles/state-of-worldwide-5g-2021.

[3] Yun Chao Hu and Miran Patel and Dario Sabella and Nurit Spereher and Valerie Young, “Mobile Edge Computing A Key Technology towards 5G”, ESTI(European Telecommunication Standards Institute), White Paper, 2015.

[4] u. s. O-RAN ALLIANCE e.V.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A Survey on Mobile Edge Computing: The Communication Perspective”, *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017. DOI: 10.1109/COMST.2017.2745201.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile Edge Computing: A Survey”, *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018. DOI: 10.1109/JIOT.2017.2750180.

[7] J. Shashirangana, H. Padmasiri, D. Meedeniya, and C. Perera, “Automated license plate recognition: A survey on methods and techniques”, *IEEE Access*, vol. 9, pp. 11 203–11 225, Dec. 2020. DOI: 10.1109/ACCESS.2020.3047929.

[8] NEC Corporation, *Press Release - NEC provides face recognition demo system utilizing MEC to DOCOMO 5G Open Lab OKINAWA*, Jan. 2019. [Online]. Available: https://www.nec.com/en/press/201901/global_20190116_02.html.

[9] X. Li, Y. Tian, F. Zhang, S. Quan, and Y. Xu, “Object detection in the context of mobile augmented reality”, *CoRR*, vol. abs/2008.06655, 2020. arXiv: 2008.06655. [Online]. Available: https://arxiv.org/abs/2008.06655.

[10] Advantech, *Advantech 5G Edge Servers for vRAN & MEC | Private 5G Networks for IIoT | 5G Edge Computing*. [Online]. Available: https://www2.advantech.com/nc/spotlight/5G/.

[11] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-Window Applications", in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12, Monterey, California, USA: Association for Computing Machinery, 2012, 47âĂŞ56, ISBN: 9781450311557. DOI: 10.1145/2145694.2145704. [Online]. Available: https://doi.org/10.1145/2145694.2145704.

[12] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs Suitable for Edge Computing?", in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/biookaghazadeh.

[13] K. Musha, T. Kudoh, and H. Amano, "Deep learning on high performance FPGA switching boards: Flow-in-cloud", English, in *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, Germany: Springer Verlag, Jan. 2018, pp. 43–54, ISBN: 9783319788890. DOI: 10.1007/978-3-319-78890-6_4.

[14] P. Ranaweera, A. Jurcut, and M. Liyanage, "MEC Enabled 5G Use Cases: A Survey on Security Vulnerabilities and Countermeasures", *ACM Computing Surveys*, Jul. 2021.

[15] Fujitsu Limited, *Press releases | Fujitsu Launches Japan's First Commercial Private 5G Network*, 2020. [Online]. Available: https://www.fujitsu.com/global/about/resources/news/press-releases/2020/0327-01.html.

[16] "The Business Case for MEC in Retail: A TCO Analysis and its Implications in the 5G Era", iGillottResearch, Inc., White Paper, 2017.

[17] Ken Sherriff, *Reverse-engineering the first FPGA chip, the XC2064*. [Online]. Available: http://www.righto.com/2020/09/reverse-engineering-first-fpga-chip.html.

[18] John McMaster, *xc2064-70*. [Online]. Available: https://siliconpr0n.org/archive/doku.php?id=mcmaster:xilinx:xc2064-70.

[19] Intel Corporation, *Intel Arria 10 FPGAs & SoCs*. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html.

[20] Xilinx, Inc., *UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices*. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp477-ultraram.

[21] Xilinx, Inc, *Zynq-7000 SoC*, 2018. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[22] Wikichip.org, *Xeon Gold 6138P - Intel*. [Online]. Available: https : / / en . wikichip.org/wiki/intel/xeon_gold/6138p.

[23] Xilinx Inc., *MicroBlaze Soft Processor Core*. [Online]. Available: https://www.xilinx.com/products/design-tools/microblaze.html.

[24] OpenRISC Community. [Online]. Available: https://openrisc.io/.

[25] N. Eskandari, N. Tarafdar, D. Ly-Ma, and P. Chow, "A Modular Heterogeneous Stack for Deploying FPGAs and CPUs in the Data Center", in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19, Seaside, CA, USA: Association for Computing Machinery, 2019, 262âĂŞ271, ISBN: 9781450361378. DOI: 10.1145/3289602.3293909. [Online]. Available: https://doi.org/10.1145/3289602.3293909.

[26] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "Programming Reconfigurable Heterogeneous Computing Clusters Using MPI With Transpiration", in *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2020, pp. 1–9. DOI: 10.1109/H2RC51942.2020.00006.

[27] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services", in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ser. ISCA '14, Minneapolis, Minnesota, USA: IEEE Press, 2014, pp. 13–24, ISBN: 978-1-4799-4394-4.

[28] Intel Corporation, *Serial Lite III Streaming IntelÂő FPGA IP User Guide*. [Online]. Available: https : / / www . intel . com / content / www / us / en / docs / programmable/683330/21-3-19-3-0/quick-reference.html.

[29] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture", in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783710.

[30] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI", in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 1–14. DOI: 10.1109/ISCA.2018.00012.

[31] N. Tarafdar, N. Eskandari, V. Sharma, C. Lo, and P. Chow, "Galapagos: A Full Stack Approach to FPGA Integration in the Cloud", *IEEE Micro*, vol. 38, no. 6, pp. 18–24, 2018. DOI: 10.1109/MM.2018.2877290.

[32] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack", in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 109–116. DOI: 10.1109/FCCM.2014.42.

[33] Amazon Web Services, Inc., *Amazon EC2 F1 Instance*, https://aws.amazon.com/jp/ec2/instance-types/f1/.

[34] Jeff Barr, *AWS News Blog - Developer Preview - EC2 Instances (F1) with Programmable Hardware*, https://aws.amazon.com/jp/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/, 30.

[35] F. Abel, J. Weerasinghe, C. Hagleitner, B. Weiss, and S. Paredes, "An FPGA Platform for Hyperscalers", in *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, 2017, pp. 29–32. DOI: 10.1109/HOTI.2017.13.

[36] P. Moorthy and N. Kapre, "Zedwulf: Power-Performance Tradeoffs of a 32-Node Zynq SoC Cluster", in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 68–75.

[37] Samtec, Inc., *Micro Flyover On-Board Optical Engine, FireFly*. [Online]. Available: https://www.samtec.com/optics/optical-cable/mid-board/firefly.

[38] K. Hironaka, N. A. V. Doan, and H. Amano, "Towards an Optimized Multi FPGA Architecture with STDM Network: A Preliminary Study", in *Applied Reconfigurable Computing. Architectures, Tools, and Applications - 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings*, 2018, pp. 142–150. DOI: 10.1007/978-3-319-78890-6_12. [Online]. Available: https://doi.org/10.1007/978-3-319-78890-6_12.

[39] Xilinx, Inc., *Vivado High Level Synthesis*. [Online]. Available: https://www.xilinx.com/video/hardware/vivado-high-level-synthesis.html.

[40] ——, *Aurora 64B/66B*. [Online]. Available: https://www.xilinx.com/products/intellectual-property/aurora64b66b.html.

[41] ——, *Memory Interface*. [Online]. Available: https://www.xilinx.com/products/intellectual-property/mig.html.

[42] PALTEK, *FPGA computing platform M-KUBOS*. [Online]. Available: https://www.paltek.co.jp/design/original/m-kubos/.

[43] K. Tomohiro, T. Ryosei, A. Hideharu, K. Michihiro, M. Hiroki, H. Toshihiro, I. Tsutomu, S. Kuniyasu, T. Akira, A. Erio, N. Shu, and K. Taura, "Flow in Cloud: A dataflow centric cloud system of heterogeneous engines", *TECHNICAL REPORT OF IEICE vol. 117 no. 153 CPSY2017-16*, pp. 1–5, 2017.

[44] Xilinx, Inc, *PYNQ | Python productivity for Zynq*, 2019. [Online]. Available: http://www.pynq.io/.

[45] J. Sheng, C. Yang, T. Wang, and M. C. Herbordt, "High Performance Dynamic Communication on Reconfigurable Clusters", in *26th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2018, Boulder, CO, USA, April 29 - May 1, 2018*, 2018, p. 219. DOI: 10.1109/FCCM.2018.00053. [Online]. Available: https://doi.org/10.1109/FCCM.2018.00053.

[46] Y. Hu, T. Kudoh, and M. Koibuchi, "A Case of Electrical Circuit Switched Interconnection Network for Parallel Computers", in *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2017, pp. 276–283. DOI: 10.1109/PDCAT.2017.00052.

[47] R. Stefan and K. Goossens, "Multi-path routing in time-division-multiplexed networks on chip", in *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, 2009, pp. 109–114. DOI: 10.1109/VLSISOC.2009.6041339.

[48] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems", in *2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Copenhagen, Denmark, 9-11 May, 2012*, 2012, pp. 152–160. DOI: 10.1109/NOCS.2012.25. [Online]. Available: https://doi.org/10.1109/NOCS.2012.25.

[49] K. Ito, K. Iizuka, K. Hironaka, Y. Hu, K. Koibuchi, and H. Amano, "Implementing a Multi-ejection Switch and Making the Use of Multiple Lanes in a Circuit-switched Multi-FPGA System", in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, 2020, pp. 211–217.

[50] K. Azegami, K. Musha, K. Hironaka, A. B. Ahmed, M. Koibuch, Y. Hu, and H. Amano, "A STDM (Static Time Division Multiplexing) Switch on a Multi-FPGA System", in *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2019, pp. 328–333. DOI: 10.1109/MCSoC.2019.00053.

[51] Khronos Group Inc., *OpenCL | Open Standard for Parallel Programming of Heterogeneous Systems*. [Online]. Available: https://www.khronos.org/opencl/.

[52] Xilinx, Inc., *SDAccel Development Environment*. [Online]. Available: https://www.xilinx.com/products/design-tools/legacy-tools/sdaccel.html.

[53] Intel Corporation, *Intel FPGA SDK for OpenCL Software Technology*. [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html.

[54] Y. SUN and H. AMANO, "FiC-RNN: A Multi-FPGA Acceleration Framework for Deep Recurrent Neural Networks", *IEICE Transactions on Information and Systems*, vol. E103.D, no. 12, pp. 2457–2462, 2020. DOI: 10.1587/transinf.2020PAP0003.

[55] Y. Yamauchi, A. B. Ahmed, K. Hironaka, K. Iizuka, and H. Amano, "Horizontal division of deep learning applications with all-to-all l communication on a multi-FPGA system", in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, 2020, pp. 277–281. DOI: 10.1109/CANDARW51189.2020.00060.

[56] Y. Fukushima, K. Iizuka, and H. Amano, "Parallel Implementation of CNN on Multi-FPGA Cluster", English, in *2021 IEEE 14th International Symposium on Embeded Multicore/Manu-core Systems-on-Chip (MCSoC)*, Dec. 2021.

[57] R. T. Fielding, "Architectural styles and the design of network-based software architectures", Publication, University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[58] Xilinx, Inc., *Integrated Logic Analyzer (ILA)*. [Online]. Available: https://www.xilinx.com/products/intellectual-property/ila.html.

[59] ——, *Xilinx Virtual Cable*. [Online]. Available: https://github.com/Xilinx/XilinxVirtualCable.

[60] ESTI, *ESTI GS MEC 003 v3.1.1 (2022-03) - Multi-access Edge Computing (MEC) Framework and Reference Architecture*. 2022.

[61] M. YAMAKURA, R. Takano, A. AHMED, M. Sugaya, and H. AMANO, "A Multi-Tenant Resource Management System for Multi-FPGA Systems", *IEICE Transactions on Information and Systems*, vol. E104.D, pp. 2078–2088, Dec. 2021. DOI: 10.1587/transinf.2021PAP0005.

[62] *Slurm Workload Manager*. [Online]. Available: https://slurm.schedmd.com/.

[63] *IBM Spectrum LSF Suites*. [Online]. Available: https://https://www.ibm.com/products/hpc-workload-management/.

[64] M. Yamakura, K. Hironaka, K. Azegami, K. Musha, and H. Amano, "The Evaluation of Partial Reconfiguration for a Multi-Board FPGA System FiCSW", in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, ser. HEART 2019, Nagasaki, Japan: Association for Computing Machinery, 2019, ISBN: 9781450372558. DOI: 10.1145/3337801.3337805. [Online]. Available: https://doi.org/10.1145/3337801.3337805.

[65] Armin Ronacher, *Flask (A Python microframework)*. [Online]. Available: http://flask.pocoo.org/.

[66] Future Technology Devices International Ltd., *FT232H - Hi-Speed Single Channel USB UART/FIFO IC*. [Online]. Available: https://www.ftdichip.com/old2020/Products/ICs/FT232H.htm.

[67] Felix Domke (tmbinc), *xvcd - Xilinx Virtual Cable Daemon*. [Online]. Available: ttps://github.com/tmbinc/xvcd.

[68] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "System Architecture for Network-Attached FPGAs in the Cloud using Partial Reconfiguration", in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 293–300. DOI: 10.1109/FPL.2019.00054.

[69] B. Ringlein, F. Abel, D. Diamantopoulos, B. Weiss, C. Hagleitner, M. Reichenbach, and D. Fey, "A Case for Function-as-a-Service with Disaggregated FPGAs", in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 333–344. DOI: 10.1109/CLOUD53861.2021.00047.

[70] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-Fast Downloading of Partial Bitstreams through Ethernet", in *Architecture of Computing Systems – ARCS 2009*, M. Berekovic, C. Müller-Schloer, C. Hochberger, and S. Wong, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 72–83, ISBN: 978-3-642-00454-4.

[71] J. Vidal, F. de Lamotte, G. Gogniat, J.-P. Diguet, and P. Soulard, "UML Design for Dynamically Reconfigurable Multiprocessor Embedded Systems", in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, Dresden, Germany: European Design and Automation Association, 2010, 1195âĂŞ1200, ISBN: 9783981080162.

[72] Muhammed Al Kadi and Patrick Rudolph and Diana Göhringer and Michael Hübner, "Dynamic and partial reconfiguration of Zynq 7000 under Linux", *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1–5, 2013.

[73] D. Wanta, W. T. Smolik, J. Kryszyn, P. Wróblewski, and M. Midura, "A Run-Time Reconfiguration Method for an FPGA-Based Electrical Capacitance Tomography System", *Electronics*, vol. 11, no. 4, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11040545. [Online]. Available: https://www.mdpi.com/2079-9292/11/4/545.

[74] J. L. Elman, "Finding Structure in Time", *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. DOI: https://doi.org/10.1207/s15516709cog1402_1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[75] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[77] R. Polig, J. Weerasinghe, and C. Hagleitner, "RESTful Web Services on Standalone Disaggregated FPGAs", Jan. 2021. DOI: 10.36227/techrxiv.13560317.v1. [Online]. Available: https://www.techrxiv.org/articles/preprint/RESTful_Web_Services_on_Standalone_Disaggregated_FPGAs/13560317.

[78] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' Build Errors: A Case Study (at Google)", in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, Hyderabad, India: Association for Computing Machinery, 2014, 724âĂŞ734, ISBN: 9781450327565. DOI: 10.1145/2568225.2568255. [Online]. Available: https://doi.org/10.1145/2568225.2568255.

[79] *Himeno benchmark*. [Online]. Available: https://i.riken.jp/en/supercom/documents/himenobmt/.

[80] Brown, Nick and Dolman, David, "It's All About Data Movement: Optimising FPGA Data Access to Boost Performance", in *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2019, pp. 1–10. DOI: 10.1109/H2RC49586.2019.00006.

[81] M. Saldana and P. Chow, "TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs", in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–6. DOI: 10.1109/FPL.2006.311233.

[82] *Open MPI: Open Source High Performance Computing*. [Online]. Available: https://www.open-mpi.org/.

[83] *MPICH | High-Performance Portable MPI*. [Online]. Available: https://www.mpich.org/.

Publications

Related Papers

Journal Papers

[1] Kazuei HIRONAKA, Takumi INAGE, Kensuke IIZUKA, Kohei ITO, and Hideharu AMANO, “Development of M-KUBOS an FPGA Cluster System for Multi-access Edge Computing", IEICE Trans. Information and Systems ,Vol.J105-D, No.10, Oct. 2022. (In press) (in Japanese)

[2] Kazuei HIRONAKA, Kensuke IIZUKA, Miho YAMAKURA, Akram BEN AHMED, and Hideharu AMANO, “Remote dynamic reconfiguration of a multi-FPGA system FiC (Flow-in-Cloud)”, *IEICE Trans. Information and Systems*, Vol.E104-D, No.8, pp.1321-1331, Aug. 2021.

International Conference Papers

[3] Kazuei Hironaka, Kensuke Iizuka, Hideharu Amano, “Implementing VTA, a tensor accelerator on Flow-in-Cloud”, In *Proc. of the 8th International Conference on Applied Computing and Information Technology, (ACIT 2021)*, pp. 46-50, July 2021.

[4] Kazuei Hironaka, Kensuke Iizuka, Akram Ben Ahmed, M. M. Imdad Ullah, Yugo Yamauchi, Yuxi Sun, Miho Yamakura, Aoi Hiruma, and Hideharu Amano, “Demonstration of flow-in-cloud: A multi-FPGA system”, In *Proc. of the 29th International Conference on Field-Programmable Logic and Applications, (FPL 2019)*, pp. 417-418, September 2019.

[5] Kazuei Hironaka, Akram Ben Ahmed, and Hideharu Amano, “Multi-FPGA Management on Flow-in-Cloud Prototype System”, In *Proc. of the 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2019)*, pp. 443-448, July 2019.
(**Best Student Presenter Award**)

[6] Kazuei Hironaka, Ng Anh Vu Doan, Hideharu Amano, “Towards an optimized multi FPGA architecture with STDM network: A preliminary study”, In *Proc. of the 14th International Symposium on Applied Reconfigurable Computing, (ARC 2018)*, pp. 142-150, May 2018.

Domestic Conference Papers and Technical Reports

[7] 弘中和衛, 飯塚健介, 天野英晴, “FPGA システム Flow-in-Cloud における TVM テンソルアクセラレータ (VTA) の実装”, 信学技報, vol. 120, no. 435, CPSY2020-69, pp. 115-120, 2021 年 3 月.

[8] 稲毛琢己, 弘中和衛, 飯塚健介, 天野英晴, “M-KUBOS を用いた PYNQ クラスタの構築”, 信学技報, vol. 120, no. 338, CPSY2020-41, pp. 107-112, 2021 年 1 月.

[9] 弘中和衛, 山倉美穂, 天野英晴, “マルチ FPGA システムにおける部分再構成の実際”, 信学技報, vol. 120, no. 36, RECONF2020-16, pp. 85-90, 2020 年 5 月.

Other Papers

Journal Papers

[10] Kohei ITO, Kensuke IIZUKA, Kazuei HIRONAKA, Yao HU, Michihiro KOIBUCHI, and Hideharu AMANO, "Improving the Performance of Circuit-Switched Interconnection Network for a Multi-FPGA System", IEICE Trans. Information and Systems, Vol.E104-D, No.12, pp.2029-2039, December 2021.

International Conference Papers

[11] Takumi Inage, Kazuei Hironaka, Kensuke Iizuka, Kohei Ito, Yasuyu Fukushima, Mitaro Namiki, and Hideharu Amano, “M-KUBOS/PYNQ Cluster for multi-access edge computing”, In *Proc. of the 2021 Ninth International Symposium on Computing and Networking (CANDAR)*, pp. 95-101, November 2021.

[12] Takumi Inage, Kazuei Hironaka, Kensuke Iizuka, and Hideharu Amano, “Software management system of the PYNQ cluster”, In *Proc. of the 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOLCHIPS24)*, Poster 11, April 2021.

[13] Tomoki Shimizu, Kohei Ito, Kensuke Iizuka, Kazuei Hironaka, and Hideharu Amano, "Hybrid Network of Packet Switching and STDM in a Multi-FPGA System", In *Proc. of the 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOLCHIPS24)*, pp. 234-239, April 2021.

[14] Kohei Ito, Kensuke IIzuka, Kazuei Hironaka, Yao Hu, Michihiro Koibuchi, and Hideharu Amano, "Implementing a Multi-ejection Switch and Making the Use of Multiple Lanes in a Circuit-switched Multi-FPGA System" , In *Proc. of the 2020 Eighth International Symposium on Computing and Networking Workshop (CANDARW)*, pp. 211-217, November 2020.

[15] Yugo Yamauchi, Akram Ben Ahmed, Kazuei Hironaka, Kensuke IIzuka, and Hideharu Amano, "Horizontal division of deep learning applications with all-to-all communication on a multiFPGA system" , In *Proc. of the 2020 Eighth International Symposium on Computing and Networking Workshop (CANDARW)*, pp. 277-281, November 2020.

[16] Kensuke Iizuka, Kohei Ito, Kazuei Hironaka, and Hideharu Amano, "A Method of Partitioning Convolutional Layer to Multiple FPGAs" , In *Proc. of the 17th International SoC Design Conference (ISOCC)*, pp.25-26, October 2020.

[17] Tomoki Shimizu, Kohei Ito, Yugo Yamauchi, Kauei Hironaka, and Hideharu Amano, "Implementation of a Packet-Switching Router on a Mukti-FPGA System", In *Proc. of the 2020 IEEE Symposium in Low-Power and High-Speed Chips (COOLCHIPS23)*, Poster 3, Tokyo, Japan, April 2020.

[18] Kohei Ito, Kensuke Iizuka, Yugo Yamauchi, Kazuei Hironaka, Yao Hu, Michihiro Koibuchi, and Hideharu Amano, "Implementation of an Application Utilized Multi-Switch on a Multi-FPGA System" , In *Proc. of the 2020 IEEE Symposium in Low-Power and High-Speed Chips (COOLCHIPS23)*, Poster 4, Tokyo, Japan, April 2020.

[19] Hideharu Amano, Akram Ben Ahmed, Kazuei Hironaka, Kensuke Iizuka, Yugo Yamauchi, M.M.Imdad Ullah, Yuxi Sun, Miho Yamakura, Aoi Hiruma, Tomotaka Shimizu, and Kohei Ito, "Flow-in-Cloud: a Scalable multi-FPGA system for HPC" , In *Proc. of the European Network on High-performance Embedded Architecture and Compilation 2020 (HiPEAC2020) EuroEXA workshop*, Jan 2020, Bologna, Italy, January 2020.

[20] Keita Azegami, Kazusa Musha, Kazuei Hironaka, Akram Ben Ahmed, Michihiro Koibuch, Yao Hu, and Hideharu Amano, "A STDM (Static Time Division Multiplexing) Switch on a Multi-FPGA System" , In *Proc.*

of the 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2019), pp.328-333, Singapore, October 2019.

[21] Miho Yamakura, Kazuei Hironaka, Keita Azegami, Kazusa Musha, and Hideharu Amano, “The Evaluation of Partial Reconfiguration for a Multi-board FPGA System FiCSW”, In *Proc. of the International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2019)*, pp.1-4, Nagasaki, Japan. June 2019.

[22] Miho Yamakura, Kazuei Hironaka, Keita Azegami, Kazusa Musha, and Hideharu Amano, ”Partial Reconfiguration Technique for a Multi-board FPGA System FiCSW”, In *Proc. of the 2019 IEEE Symposium in Low-Power and High-Speed Chips (COOLCHIPS22)*, Poster 9, Yokohama, Japan, April 2019. (COOL Chips 22 Featured Poster Award)

Domestic Conference Papers and Technical Reports

[23] 伊藤光平, 飯塚健介, 山内脩吾, 弘中和衛, 胡曜, 鯉渕道紘, 天野英晴,“マルチ FPGA における複数スイッチを使用した際の性能評価”, 信学技報, vol. 119, no. 372, CPSY2019-58, pp. 37-42, 2020 年 1 月.

[24] 清水智貴, 伊藤光平, 飯塚健介, 山内脩吾, 弘中和衛, 天野英晴,“ルータのマルチ FPGA システムへの実装と性能評価”, 信学技報, vol. 119, no. 372, CPSY2019-57, pp. 31-36, 2020 年 1 月.

[25] 天野英晴, 弘中和衛, 飯塚健介,“M-KUBOS ボードを用いた次世代 FiC”, 信学技報, vol. 120, no. 121, CPSY2020-9, pp. 55-60, 2020 年 7 月.

[26] 畔上佳太, 武者千嵯, 弘中和衛, Akram Ben Ahmed, 天野英晴,“マルチ FPGA ボード間通信を行うスイッチの開発”, 信学技報, vol. 118, no. 215, RECONF2018-29, pp. 55-59, 2018 年 9 月.

[27] 弘中和衛, 川口智大,“大規模ストレージシステムを対象としたデータ重複排除見積もりシステムの試作”, 信学技報, vol. 117, no. 44, CPSY2017-10, pp. 51-54, 2017 年 5 月.