PAPER

Remote Dynamic Reconfiguration of a Multi-FPGA System FiC (Flow-in-Cloud)

Kazuei HIRONAKA^{†a)}, Kensuke IIZUKA[†], Miho YAMAKURA[†], Akram BEN AHMED[†],
and Hideharu AMANO[†], Members

SUMMARY Multi-FPGA systems have been receiving a lot of attention as a low cost and energy efficient system for Multi-access Edge Computing (MEC). For such purpose, a bare-metal multi-FPGA system called FiC (Flow-in-Cloud) is under development. In this paper, we introduce the FiC multi FPGA cluster which is applied partial reconfiguration (PR) FPGA design flow to support online user defined accelerator replacement while executing FPGA interconnection network and its low-level multiple FPGA management software called *remote PR manager*. With the *remote PR manager*, the user can define the FiC FPGA cluster setup by JSON and control the cluster from user application with the cooperation of simple cluster management tool / library called *ficmgr* on the client host and REST API service provider called *ficwww* on Raspberry Pi 3 (RPi3) on each node. According to the evaluation results with a prototype FiC FPGA cluster system with 12 nodes, using with online application replacement by PR and on-the-fly FPGA bitstream compression, the time for FPGA bitstream distribution was reduced to 1/17 and the total cluster setup time was reduced by 21~57% than compared to cluster setup with full configuration FPGA bitstream.

key words: multi FPGA system, FPGA in cloud computing, interconnection networks, warehouse-scale computing, data centers

1. Introduction

The 5th generation mobile communication (5G) enables much lower latency and higher bandwidth communication than conventional generations. Especially, the delay of a wireless access division is less than 0.5ms and a large number of mobile clients can be connected with a single base station. This technology shift will accelerate MEC (Multi-access Edge Computing) that provides timing critical service for several mobile clients at the base stations near from edges, and it is expected to be used for sophisticated AI (Artificial Intelligence) based applications in factory management, traffic management, and smart city (Fig. 1).

Although high performance and timing critical applications such as voice/image recognition and processing services are demanded in the MEC, the facility and allowable power are limited compared with cloud computing. Under such limitations, the FPGA computing which can accept I/O requests directly and offload data processing with the hard-wired logic has received a lot of attention because of its high energy efficiency and real time data processing capability. FiC (Flow-in-Cloud) [1] is a bare-metal multi-FPGA system

Fig. 1 Multi-access edge computing

with several middle-range economical FPGAs connected with high speed serial links. Each node, called FiC-SW, consists of a mid-range economical Xilinx Kintex Ultrascale FPGA and a tiny Raspberry-Pi3 (RPi3) single board computer unlike traditional multi-FPGA systems with powerful high-end FPGAs like Microsoft's Catapult2/Brainwave [2], [3]. From the High Level Synthesis (HLS) design, multiple FiC-SW's FPGA nodes can be treated as a single monolithic FPGA. Only a host server is provided for system management through onboard RPi3 and high speed data transfer through PCIe. Since requests from multiple edges are supposed to be given to I/O of each FiC-SW, multiple FiC-SW nodes are allocated to multiple users/tasks dynamically. Recently, the usage of the FPGA in the cloud computing environment has been proposed. An FPGA integrator is proposed for *OpenStack* a platform [4] for Infrastructure-as-a-Service (IaaS). As a commercial system, the Amazon EC2 F1 Instance [5] is available, which provides a EC2 compute node with FPGA accelerator. Most of such previous work focused on how to virtualize a single powerful FPGA for multiple users in a cloud. However, in the case of the FiC, a set of jobs each of which is executed on multi-FPGA nodes must be controlled in the system. Moreover, FiC-SW is used as a high performance network switch to connect multiple nodes as well as a user programmable computation nodes.

To deal with such dual-task, the FiC design adopts partial-reconfiguration (PR) design technique and dynamic reconfiguration only for user defined HLS modules. By us-

Manuscript received August 8, 2020.

Manuscript revised February 8, 2021.

Manuscript publicized May 12, 2021.

[†]The authors are with Dept. of ICS, Keio University, Yokohama-shi, 223-8522 Japan.

a) E-mail: fic@am.ics.keio.ac.jp

DOI: 10.1587/transinf.2020EDP7165

ing the PR design technique, the FPGA is divided into two parts: the static part including high speed serial interface, switches, and interface for RPi3 and DRAM, and the PR part which contains the user programmed HLS acceleration modules. The benefits of the PR part are (1) The static part can work continuously when the PR part is changed. (2) The design of the PR part does not influence the operational frequency and resource of the static part. (3) The amount of configuration data for the PR part is smaller than that for the total FPGA, thus the configuration time can be reduced. In order to allocate and distribute the user designed HLS modules with the PR technique, a remote PR manager that configures user defined HLS part of FPGAs from a remote host is essential.

The contribution of this paper is as follows: (1) The software of the remote PR manager for a bare metal multi-FPGA cluster is designed, implemented and evaluated on a system with 12 FiC-SW (mk1) FPGA boards, (2) In order to reduce the FPGA configuration time, multi nodes configuration data deployment with data compression is introduced, and (3) The PR design flow of Xilinx's Kintex Ultrascale is shown and evaluated. Although the sophisticated PR control mechanisms [6] have been reported, there are no other papers of the remote PR manager for a multi-FPGA cluster to the best of our knowledge. One reason is that such low-layer software is difficult to be described in a technical paper. However, we believe such a practical case study and evaluation include a constructive contribution for building such multi-FPGA systems.

The rest of the paper is organized as follows: The target multi-FPGA system FiC and its FiC-SW board are introduced in Sect. 2. The FiC PR manager is described in Sect. 3, and the partial-reconfiguration design flow is shown in Sect. 4. The dynamic reconfiguration is evaluated on the real system and shown in Sect. 5. After the comparison with related work in Sect. 6, we conclude this paper in Sect. 7.

2. FiC multi-FPGA Cluster Prototype

The prototype FiC system consists of multiple FiC-SW boards connected with each other and with the I/O board by high-speed interconnect network illustrated in Fig. 2. The system in Fig. 2 shows 24 FiC-SW cluster configuration consisted with 12 mk1 nodes and 12 mk2 nodes which are connected to form a 6x4 torus network for an example, but it also can support various network topologies depending on application with eight 4 channels flat cables connectivity of each node.

2.1 FiC-SW Prototype Board

Figure 3 illustrates our developed FiC-SW prototype board. We employed a middle-class Xilinx's Kintex Ultrascale FPGA mounted on the board which supports up to 64 of GTH high-speed serial transceivers for the FiC interconnect network. To enforce the prototype system's capability, we have been improving the prototype system. Cur-

Fig. 2 Prototype FiC cluster overview

Fig. 3 Prototype FiC-SW board (mk1)

rently we have developed two generations of the FiC-SW prototype boards which are employing different FPGAs. In mk1 board, Kintex Ultrascale XCKU095 is used and in mk2 board, Kintex Ultrascale XCKU115 is used which has more DSP resources than mk1 board. Both generations designed as considered for inter-operability on the interconnect network to constructing FPGA cluster together. Although the GTH serial channel has maximum bandwidth is 16.3Gbps, we regulated the transfer speed to 9.9Gbps for the sake of easy implementation. To provide enough bandwidth, it delivers up to 32 channels per board at the current design, supporting various network topologies connecting hundreds of boards. Each board provides two of 16GB DDR4-SDRAM for data storing for computation. For board management purpose, a Raspberry Pi 3 (RPi3) [7] single board computer is also mounted. The RPi3 is connected to FPGA by GPIO ports to configure the FPGA and communicates to logic of FPGA. Additionally, in the prototype FiC system, a control server connecting to FiC-SW cluster with an I/O board using Xilinx KCU1500. The I/O board is connected to the host by PCI Express Gen3.0 x8 and FiC-SW by two 8 channels serial links connectivity for large size data exchange between FiC-SW cluster and outside. Table 1 summarizes the hardware specifications of the current FiC-SW prototype.

Table 1 Specifications of current prototype FiC system

System Scale	24 FiC-SW boards and an I/O board (KCU1550)
FPGA	Kintex UltraScale XCKU095-FFVB2104 (mk1) XCKU115-FFVB2104 (mk2)
Clock Freq.	100MHz
STDM switch	four 9x9 at maximum
Serial connection	32 channels (8 bundles x 4 channels)
Effective Speed	8.5Gbps (9.9Gbps at a link)
Total exchange bandwidth	272Gbps
Total available throughput	34GBps
Pass through Latency	550 μ sec
Max Latency of the system	1710 μ sec
DRAM	16Gbit DDR4 DRAM (200MHz) x 2
On-board controller	Raspberry Pi3 Model B (BCM2837 ARM Cortex-A53 Quad 1.2GHz)

Fig. 4 Firefly Cable

2.2 Interconnection

For the interconnection among FiC-SWs, we have employed cost efficient *FireflyTM Micro FlyoverTM* system by Samtec [8] (Fig. 4). Each of the Firefly flat cable offers four bi-directional channels to one destination, the design introduces a restriction that four channels must be connected to the same destination together. Hereafter, we call such a set of four bi-directional channels a *bundle*. Logically, the network is shared by Static Time Division Multiplexing (STDM) method for keeping a constant latency and bandwidth between multiple communications [9].

2.3 FPGA Logic Design

The FiC-SW FPGA logic is illustrated in Fig. 5. It is divided into two regions: 1) static region (hereafter, we call it *shell*) which includes the STDM network switch and peripheral controller logic, and 2) partial re-configurable (PR) region for user design logic by HLS. In the PR region, several HLS modules are connected with each other using the AXI stream interface, and form a group wrapped with a standard HDL module to connect the static region. Since all HLS modules use the same AXI stream interface to the shell, it can be dynamically replaced without stopping the commu-

Fig. 5 Block diagram of the FiC board

nication between other boards. Only a routing table used in the STDM is changed when new circuits are needed for a new application program. Four ports of the STDM switches are directly connected to the PR region, and 85bit data are transferred with 100MHz clock. That is, each PR region can input and output data with 34Gbps bandwidth in total. To control HLS modules, we defined standard application reset/start/done signals: *ap_rst*, *ap_start*, *ap_done* and data input/output. All signals are mapped to control registers in the shell part, and they are accessible from software on the RPi3. When a HLS module uses DRAM, a full AXI-4 bus is optionally provided in the static region to connect the PR region and the DDR-4 DRAM controller.

2.4 STDM Switch

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. Compared with high speed packet switching networks [10], latency is larger especially under light traffic load. Thus, there have been only a few examples to be adopted for an interconnection network of parallel computer systems [11], [12]. By using the STDM, the communication time is divided into slots, and therefore, data from one board to another can be transmitted only at fixed time slots. Figure 6 shows the outline of the STDM switch with 3 ports and 3 slots. The 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.

Broadcasting can be easily done by sending data in an input FIFO to multiple different output registers. If the HLS module for a given target application has fixed communication partners and computation time, we can select an optimal number of time slots and routing table setting for each board [13].

Interconnection between nodes, each FiC-SW board has 32 GTH serial channels and bundled every 4 channels into a bundle as we mentioned in Sect. 2.2, so it is 8 bundles per a node. Interchange between channels in a bundle is

Fig. 6 The STDM switch with 3 ports and 3 slots

not so useful since a bundle destination is always the same. Thus, we provide four independent STDM switches, one for each bundle and each board provides four 9x9 switches (8 bundles + one internal port from/to on-chip HLS modules) in total at the largest configuration. Since 8 bundles can be too many for small-scale systems, we have developed a parameterizable switching generator which can generate the HDL description of a switch ranging from 3x3 to 9x9 for various system configurations and maximize the partial reconfigurable region.

A 9.9 Gbps serial data link, including Error Correction Coding (ECC) scheme, is transferred via Xilinx Aurora IP and then converted into 85 bit data for each 100MHz system clock cycle. This relatively slow system clock is adopted for easy implementation of the HLS modules. In other words, the effective data transfer rate of each lane is 8.5 Gbps, and the total exchange throughput of the four switches is 272 Gbps. The latency required to pass through a board is 55 clock cycles, thus 550 μ sec. Most of the clock cycles are spent on the serial-parallel conversion by the Aurora-IP. For now, the scale of the experimental system is small (2x4 torus), and the diameter is only 3. In most of the HLS designs, the slot number can be set to be two. Therefore, the largest latency of the system is 1710 μ sec independently of the traffic load.

2.5 Applications on the FiC-SW Multi-FPGA Cluster

Several practical applications for the FiC cluster have been developed. For example, [14] proposes an implementation of recurrent neural networks (RNN) on FiC-SW cluster. [15] proposes an implementation of LeNet network on FiC-SW cluster with all-to-all network communication among nodes. [16] proposes an implementation of an accelerator to sequential pattern matching for RNA sequences.

3. Multi-FPGA Management System

The FiC management system is consisted of two layers as shown in Fig. 7. The upper layer, Flow-OS [1] accepts a job assigned into available multiple boards according to the

Fig. 7 The current operational environment of FiC

scheduling policy. In order to improve the flexibility for assignment, it makes the use of virtualization mechanism provided in the FiC board. Now, it is under development and the function of Flow-OS is done manually. The lower layer is FiC PR manager on which we focus in this paper. It is consisted of the following components: A) A cluster configuration tool called *ficmgr* which runs on the host server or user client, and B) an on-board FPGA management middleware called *ficwww* which runs on the RPi3 of FiC-SW to provide FPGA control via HTTP based APIs, and C) a remote FPGA JTAG for debugging.

3.1 Cluster Setup

For executing a job on multiple FiC-SW nodes, at first, the configuration data are transferred to each FPGA through the on board RPi3. As shown in the next section, only the configuration data for the PR region are transferred. Although individual bitstream is needed to be transferred to each FPGA, a single bitstream is often broadcasted to the multiple nodes. Then, the routing tables of the STDM switch are distributed. After the application starts, the I/O data must be transferred. *ficmgr* described in Python manages all of them on the host server by calling APIs implemented on RPi3.

To handle the cluster setup sequences, the user defines the cluster configuration for multiple nodes by JSON format text file. *ficmgr* reads it and firstly, it configures each FiC-SW node simultaneously via HTTP. An example of the JSON file is shown in Fig. 8. A key “fic08” means the target cluster node for configuration, entries under “fpga” are specifying FPGA configuration settings, entries under “switch” are specifying STDM switch configuration of the FPGA shell part, and entries under “option” are specifying after configuration behaviors such as automatically starts an application of HLS part after the FPGA configuration, runs an application on the RPi3, etc. After the FPGA configuration is completed, the *ficmgr* distributes the routing table for the STDM switch network to each FiC-SW node. The routing table can be automatically generated from the interconnection topology of HLS modules assigned into each FiC-SW with the algorithm and tools provided in [13].

The on-board RPi3 is connected to the dedicated management network via Ethernet for FPGA bitstream distribution (Fig. 9). For the current FiC-SW nodes, each FPGA

```

{
  "fic08": {
    "fpga": {
      "bitstream": "fic_top.bin",
      "progmde": "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"
    }
  },
  ...
}

```

Fig. 8 An example of JSON description

Application	Partial	Clear	Total	Gzip
lchk	-	-	34.18	2.32
LeNet	-	-	34.18	2.85
sort	-	-	34.18	2.32
lchk_small_pr	12.31	0.81	13.11	1.01
LeNet_small_pr	12.31	0.81	13.11	1.57
sort_small_pr	12.31	0.81	13.11	1.01
lchk_max_pr	26.64	1.82	28.46	1.23
LeNet_max_pr	26.64	1.82	28.46	1.84
sort_max_pr	26.64	1.82	28.46	1.20

configuration bitstream size is around 30~40 MB, it takes a few seconds to distribute FPGA configuration bitstream to each FiC-SW every time even if the configuration data for the PR region is smaller than that for the total configuration. This FPGA bitstream distribution time may become a problem regarding the system installed into the MEC and used by several applications. Therefore, the *ficmgr* uses gzip compression on the HTTP transaction to suppress the distribution time. In our preliminary evaluation of gzip compression for the FPGA bitstream file, the compression ratio is reached to around ~1/15 (Table 2). This result means the FPGA configuration bitstream is “sparse” in terms of information entropy, and application of gzip compression is efficient as shown later. After the data is distributed to each RPi3, it is written into the target FPGA with SelectMAP x8 (parallel) mode via RPi’s GPIO port.

3.2 Run Application on the Cluster

Application on the FiC cluster consists of client-side program and FPGA (HLS) side program as like as accelerators commonly used. The client-side program is mainly pro-

Fig. 9 Operation environment of FiC cluster

grammed by Python or C code to write an application and kernel code by HLS is offloaded to the FPGA.

The user can define the cluster setup including FPGA bitstream programming and interconnection network switch table configuration for multiple nodes by JSON setup configuration file, and set up the cluster with the configuration by cluster management tool called *ficmgr* (Fig. 9 (1)). The *ficmgr* deploys the FPGA bitstream to each node by HTTP request and setups the interconnection network switch table (Fig. 9 (2)). After the FPGA bitstream transferred to every node, the on-board RPi3 downloads the FPGA bitstream to the FPGA (Fig. 9 (3)). Once all required nodes are configured, the client program sends the data for computation to each node (Fig. 9 (4)). To exchange data between client and the FiC FPGA cluster, on-board RPi3 or FiC I/O board is used. The on-board RPi3 is directly connected to FPGA through RPi3’s GPIO port. It can exchange small-sized data like parameters needed for execution between host and FPGA or onboard DDR4 DRAM through FPGA. Moreover, the client program can access the data on onboard DDR4 DRAM through HTTP request after FPGA configuration. Moreover, the FiC I/O board connected with the PCIe of the host server is also available when the application requires large-sized data. The FiC I/O board offers 8 channels 9.9Gbps serial links similar to the cluster interconnection network and it provides 256bit x 512 block data transfer between the client host and the FiC FPGA cluster. After the data transferred, the client program initiates the FPGA application run (Initiates *ap_start* signal on all nodes) and waits until application execution finished signal (*ap_done*) by HLS asserted on all nodes (Fig. 9 (5) (7)). While running the application on the cluster, nodes are communicated by high-speed interconnection (Fig. 9 (6)). After the application execution finished, the client program reads out the results on each node (Fig. 9 (7)).

3.3 ficwww: Web API Provider for FiC Management

Representational State Transfer (REST) is a software architecture style currently widely used in web applications. In

Table 3 REST APIs for prototype FiC system

API	Method	Function
/fpga	POST	Configure FPGA with posted 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
/ddrwrite	POST	Write data on onboard DDR through FPGA
/ddrread	POST	Read data on onboard DDR through FPGA

Fig. 10 fiC-SW cluster Web dashboard

the RESTful API design, all API calls are represented as a corresponding URL, and it is callable by simple HTTP requests (POST / GET / DELETE / PUT etc.) from clients. Considering the FiC usage as an accelerator on the MEC, web API is advantageous for making it accessible from various applications. In *fiC-SW* running on RPi3 on each FiC-SW defines the minimum mandatory API sets shown in Table 3. From the host server, *ficmgr* controls its operation by calling the API sets.

Although the batch-style job control supported by *ficmgr* is convenient to control a large number of FPGA boards, a more user-friendly interface is needed especially at the program development stage. FiC-dashboard (Fig. 10) is provided to manage and monitor the FiC cluster through the web browser. With the dashboard, users can monitor and configure the cluster operations such as controlling power of the nodes, resetting of cluster nodes, configuration of FPGA, switch configuration, monitoring the switch network status, and checking FPGA application status by web browser. The

Fig. 11 Debugging FiC-SW overview

FiC-dashboard is hosted by Apache HTTP daemon on the control server, and displays multiple FiC-SW boards information on a display screen. Each *fiC-SW* WebGUI Uniform Resource Locator (URL) is mapped to sub directory of the control server URL by HTTP server proxy function, where the user can access to each *fiC-SW* from the dashboard directly.

3.4 Remote FPGA Debugging

Even for the application development with HLS, the hardware-level debugging is often required. The FiC RP manager supports remote hardware debugging with the Integrated Logic Analyser (ILA) of Xilinx FPGA. To support this feature, we use Xilinx Virtual Cable (XVC) [17] protocol. The XVC is an open-source TCP/IP based protocol by Xilinx which acts like a JTAG cable to access the target FPGA device remotely. The benefit of using XVC is the protocol acting just like a cable, it means the XVC is just transferring JTAG operation issued by hardware server (hw_server) in localhost to the target device in remote, therefore the XVC is independent of vendor-specific JTAG commands and it allows us to use ILA with various JTAG hardware. Figure 11 shows the current FiC system remote debugging environment overview. In the FiC system, we connected FTDI's FT232H USB UART chip to the USB port on RPi3 as a JTAG hardware and running the XVC server process on the RPi3. This configuration allows users to access the JTAG port of each FPGA and debugging with ILA from remote.

4. Partial Reconfiguration on FiC-SW

4.1 Partial Reconfiguration Design Flow

Figure 12 shows the design flow to enable partial reconfiguration. Firstly, we make a first design with modules of shell (static region) and dummy HLS module (PR region) to connecting shell and PR region with AXI buses, then synthesize the design and store it in a checkpoint. Here, the first HLS module is synthesized and saved in another checkpoint. After setting the PR region, the checkpoint of the first HLS module is read into the PR region, then the common implement operations (opt_design, place_design, and

Fig. 12 Partial reconfiguration design flow

route.design) are applied on this design, and three configuration files (the total configuration data, the data to clear the configuration, and the PR configuration data) are generated. The data to clear the configuration are needed to clear the PR area before the partial reconfiguration. Only Kintex Ultrascale requires such a process for the PR. These configuration files are for the first HLS module.

Here, the PR region was changed into the black box, the wiring of the static region was locked, and the design was stored into the checkpoint for the template. Once the template is created, the partial reconfiguration design can be performed only by synthesizing, placing, and routing other HLS modules unless the PR region size is changed. All these steps are executed with Vivado tcl mode, and the designer can generate their PR configuration data just by executing the tcl script.

The problem is how to fix the PR region. For using the maximum resources in the PR region, the PR region must be as large as possible. The size of the PR is depending on the type of FPGA. In the case of XCU095 used in the FiC, the total FPGA consists of 5x8 tiles (X0Y0 - X4Y7). Among them, the bottom most row (X0Y0 - X4Y0), and top most row (X0Y7 - X4Y7) cannot be used for the PR. Fortunately, in the FiC, the tiles can be efficiently used for the interfaces and switches. For the design of the current prototype with four 5 x 5 switches can be placed in this area. The maximum area for the PR region assigned into the Pblock X1Y0-X4Y6 as shown in Fig. 13 (a). Figure 13 (b) shows the result of place-and-route for the design of LeNet shown in later. The total amount of configuration data is not dependent on the design in the PR region but related to the region size. Thus, in order to reduce the configuration time, we provided a smaller region as shown in Fig. 14 (a). It uses mainly 9 tiles (X1Y1 - X3Y3) and partly 3 tiles (X1Y4 - X3Y4) for keeping enough DSPs. This small PR area was enough for all designs shown in the evaluation. Figure 14 (b) shows the result of place-and-route for the design of LeNet.

4.2 Resources in the PRs and Configuration data size

In this paper, we used Vivado Version 2019.1. As shown before, FiC uses XCKU095-FFVB2104 (mk1 board). The target frequency for both static and the PR part is set to be 100MHz, while the interface region of serial links uses

a) Max. PR region pblock

b) Max. PR region after P&R

Fig. 13 The floor-plan of the maximum PR

a) Small PR region pblock

b) Small PR region after P&R

Fig. 14 The floor-plan of the small PR

Table 4 The amount of resources in the PR region

Resource	FPGA Total	Max. PR	Small PR
LUT as logic	537600	354240 (65.9%)	132640 (24.7%)
LUT as memory	76800	48960 (63.7%)	16720 (21.8%)
Registers	1075200	708480 (65.9%)	265280 (24.7%)
BRAM	1680	1224 (72.9%)	456 (27.1%)
DSP	768	544 (70.8%)	152 (19.8%)
Conf.Data	35843476	27935580(77.9%)	16099796(44.9%)
Conf.Clear	-	1913396(5.3%)	845876(2.3%)

150MHz clock. The hardware resource in the PRs is shown in Table 4. **Conf.Data** and **Conf.Clear** represent amount of configuration data used for the partial reconfiguration and clearing the reconfiguration area. The max. PR can use about 65% of the total LUTs available in the FPGA. Other resources are almost similar, that is, the balance in the resources is kept. The small PR occupies about 25%, but unfortunately, the amount of configuration data is not reduced.

Considering that, using small PRs is not a so efficient way to reduce the configuration data.

5. Evaluation

5.1 Evaluation Environment

In this section, we evaluated the multi-FPGA management system for dynamic FPGA reconfiguration. For the evaluation, we used 12 FiC-SW mk1 board nodes and 3 different designs shown in Table 2 to make sure all nodes are same conditions with FPGA device and configuration method to see management scalability. In Table 2, an application name ending with “pr” means the configuration file for the partial reconfiguration. Columns Partial, Clear, Total and Gzip are showing configuration file size.

5.2 FPGA Configuration Time

Since FiC is a bare-metal multi-FPGA system, the cluster setup time is an essential metrics to support multiple users or applications. Specifically, FPGA configuration time is a dominant of the setup time. Table 5 shows a comparison of single FPGA configuration time for the FiC-SW mk1 board (Kintex Ultrascale XCKU095) by several configuration methods.

- **RPi3+libfic2** is the configuration using RPi3 GPIO port to configure FPGA by SMAPx8 interface.
- **RPi3+libfic2(Opt)** is the configuration using the RPi3+libfic2 with optimized GPIO operation.
- **Xilinx DLC9LP** is configuration using USB Xilinx Platform Cable (DLC9LP) by Vivado hardware manager.
- **FT232+XVC** is configuration using USB serial converter chip FT232 and Xilinx Virtual Cable (XVC) by Vivado hardware manager.

Table 5 shows **RPi3+libfic2** method has dramatically reduced FPGA configuration time compared to other methods. Especially, with optimization has achieved 64 times faster than configuring by official platform JTAG cable **Xilinx DLC9LP**. This optimization is for reduce required number of GPIO operation in RPi3 during FPGA configuration by using an FPGA bitstream data characteristic which contains sequential same data patterns. Although the FPGA connected by RPi3’s GPIOs which are not designed as high speed data transactions, the result shows the GPIO achieved good enough performance for FPGA configuration without additional costly hardware.

Table 5 Comparison of FPGA configuration time

Method	Mode	Time	Throughput
1 RPi3+libfic2	SMAPx8	0:31 s	1.10 MB/s
2 RPi3+libfic2(Opt)	SMAPx8	0:01 s	25.32 MB/s
3 Xilinx DLC9LP	JTAG	1:17 s	339.84 KB/s
4 FT232+XVC	JTAG	4:26 s	131.59 KB/s

5.3 FPGA Bitstream Distribution Time

FPGA bitstream distribution time is also an essential metrics in the bare-metal multi-FPGA system. In the FiC-SW cluster, each FPGA bitstream distributes by moderate speed RPi3’s 100Base-TX Ethernet network. Since FiC-SW’s Ultrascale FPGA is a middle range FPGA, the size of each configuration bitstream is around 35MB (Table 2). However, Table 2 shows that the FPGA configuration bitstream is effectively compressed. In this evaluation, we evaluate how bitstream compression is effective when distributing FPGA bitstream by HTTP request to each node.

Figure 15 shows the comparison between FPGA configuration bitstream transfer time of the LeNet design 1~12 of FiC-SW nodes. In Fig. 15, **raw** is non-compressed FPGA bitstream transfer time to each FiC-SW node and **gzip** is FPGA bitstream transfer time with compressed bitstream to each FiC-SW node. The result shows **raw** bitstream transfer time is barely increased according to the number of nodes. By contrast, **gzip** result shows greatly suppressed bitstream distribution time to 1/14 compared to **raw** and it is achieved constant performance if the number of nodes is increased.

Thus, applying bitstream compression can reduce bitstream distribution time dramatically and avoid management network congestion during the cluster setup process, also the result suggests the bitstream distribution time is not a big concern with the current cluster scale.

5.4 System Scalability

Figure 16 through Fig. 18 show the total cluster setup time of each number of mk1 FiC-SW node (up to 12) in SMAPx8 FPGA configuration mode (method #1 in Table 5) from RPi3 with gzip bitstream compression and 3 different designs in Table 2 including w/ and w/o partial reconfiguration. In this evaluation, we have evaluated 10 times each condition and compare the shortest total setup time which can be regarded as an achievable maximum setup performance of our current management system. The total setup time is including a bitstream distribution time, FPGA configuration time from RPi3 to FPGA, HTTP request processing time including bitstream data encoding (base64 and compression, etc.) time, and process synchronization time among nodes.

Fig. 15 FPGA bitstream distribution time

Fig. 16 Setup time (Full configuration)

Fig. 17 Setup time (Max PR area)

Fig. 18 Setup time (Small PR area)

Results in Fig. 16 through Fig. 18 show total setup time is not significantly increased according to the number of nodes, the approximately increased time overhead per 4 nodes is only 2%. Thus, the simultaneous setup process execution for multiple nodes of the PR manager is well working and it has good enough scalability to support up to 24 nodes which is the current FIC-SW cluster scale.

Comparing setup time between full configuration (**Full**) (avg. 16.80s) in Fig. 16, partial reconfiguration in maximized PR area configuration (**Max PR**) (avg. 13.23s) in Fig. 17 and minimized PR area configuration (**Small PR**) (avg. 7.21s) in Fig. 18, the total setup time is reduced 21.25% in **Max PR** case and 57.08% reduced in **Small PR**, according to the configuration bitstream size. Considering 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** bitstream size is 62% smaller than **Full** but setup time reduction is 57% only. On the other hand, the

Fig. 19 Setup time variation (Full configuration)

Fig. 20 Setup time variation (Partial reconfiguration)

Max PR bitstream size is 17% smaller than **Full** but setup time reduction is 21%.

The partial reconfiguration requires to program the FPGA in 2 phases with the “clear” bitstream before programming the PR bitstream individually. But the result suggests that the penalties according to 2 phases configuration are negligible and bigger PR region design is more efficient rather than small PR region design in terms of setup time.

Figure 19 and Fig. 20 are showing total setup time variations of LeNet application in every 10 iterations.

In both full configuration and PR cases, the result shows that the worst total setup time is longer according to the increasing number of setup nodes. This caused by the issuable numbers of HTTP requests for the setup process was regulated in the client host, so it causes unstable HTTP response time. Comparing the full configuration case and the PR case, the worst setup time in 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 2 phase configurations per node. Thus, the result suggests the current management system needs to be considered for setup time stability improvement to apply for more large scale system.

6. Related Work

Recently, FPGA-in-cloud [18] is commercially accessible and various techniques have been proposed to manage them.

[4] proposes an FPGA integration for cloud computing platform. It manages a large FPGA divided into independent regions with partial reconfiguration and shared by multiple

users.

In [19], multiple regions in an FPGA can be used by different tasks. Although the above works focused on sharing the resource on a single large FPGA, some recent works were extended to manage multi-FPGA systems [6], [20]. Galapagos [6] uses the abstraction layer and Galapagos HW stack to solve the problem of managing large clusters of FPGA accelerators. The FPGA is abstracted and provided in the form of a stack. [20] provides an interface that allows users to use software and hardware as a Virtual Network Functions, for making effective use of multiple FPGAs in the data center. The system in [21] tries to integrate multiple FPGAs directly attached to the data-center network as an accelerator fabric.

Most of the above use multiple FPGAs each of which is directly connected to host a server which forms a cluster for the data center. Since the software for cloud computing can be used directly, the low level FPGA manager can be simple and differently implemented as the PR manager.

ZedWulf [22] is a Zynq based ARM + FPGA cluster similar to FiC in the aspect that both are bare-metal FPGA clusters. However, in ZedWulf, each Zynq tends to execute individual jobs, while multiple FPGAs in FiC execute a single job as if there were a large single monolithic virtual FPGA.

7. Conclusion

In this paper, we have introduced our FiC multi-FPGA cluster system, its low layer of multiple FPGA management software called remote PR manager, and its design flow of the FPGA application with partial reconfiguration (PR). By using the PR, the design is useful for bare-metal FPGA cluster because the shell part which includes the cluster interconnect and switch can run continuously when the application design implemented in the core part is replaced. The PR manager provides primitives of FPGA cluster management functions by the cooperation of *ficmgr* on the client host and *ficwww* on RPi3 in every node, it provides FPGA and I/O management and control capability of multiple FPGAs with a user-defined JSON file, or intuitive web GUI. For efficient cluster setup, we applied PR FPGA design flow and on-the-fly FPGA bitstream compression/decompression in PR Manager, the configuration distribution time was reduced to 1/17 and the total cluster setup time was reduced to 19~58%. Currently, management functions on the upper layer like job scheduling, job assignment, and FPGA virtualization are done manually by the user with a simple dashboard on the website for arbitration. But when the Flow-OS which now under development is available, it will reduce such a burden on users. The development of the PR manager and the results in this paper are useful to understand the trade-off in the design of Flow-OS.

References

- [1] T. Kudoh, R. Takano, H. Amano, M. Koibuchi, H. Matsutani, T. Hanawa, T. Ikegami, K. Suzuki, A. Tanaka, E. Akanuma, S. Namiki, 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.
- [2] 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," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.1–13, 2016.
- [3] 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," ISCA'18, pp.1–14, June 2018.
- [4] S. Byma, J.G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, pp.109–116, May 2014.
- [5] Amazon Web Services, Inc., "Amazon EC2 F1 Instance." <https://aws.amazon.com/jp/ec2/instance-types/f1/>
- [6] 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, Nov. 2018.
- [7] Raspberry Pi Foundation, "Raspberry Pi." <https://www.raspberrypi.org/>
- [8] Samtec, Inc., "Micro Flyover On-Board Optical Engine, FireFly." <https://www.samtec.com/optics/optical-cable/mid-board/firefly>
- [9] K. Hironaka, N.A.V. Doan, and H. Amano, "Towards an optimized multi FPGA architecture with STDM network: A preliminary study," Applied Reconfigurable Computing. Architectures, Tools, and Applications - 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings, vol.10824, pp.142–150, 2018.
- [10] J. Sheng, C. Yang, T. Wang, and M.C. Herbordt, "High performance dynamic communication on reconfigurable clusters," 26th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2018, Boulder, CO, USA, April 29 - May 1, 2018, p.219, 2018.
- [11] R. Stefan and K. Goossens, "Multi-path routing in time-division-multiplexed networks on chip," 2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC), pp.109–114, Oct. 2009.
- [12] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki, "A statically scheduled time-division-multiplexed network-on-chip for real-time systems," 2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NoCS), Copenhagen, Denmark, 9-11 May, 2012, pp.152–160, 2012.
- [13] Y. Hu, T. Kudoh, and M. Koibuchi, "A case of electrical circuit switched interconnection network for parallel computers," 2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp.276–283, Dec. 2017.
- [14] 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.
- [15] Y. Yamauchi, A.B. Ahmed, K. Hironaka, K. Iizuka, and H. Amano, "Horizontal division of deep learning applications with all-to-all communication on a multifpga system," 2020 Eighth International Symposium on Computing and Networking Workshop (CANDARW), pp.277–281, Nov. 2020.
- [16] M.M.I. Ullah, A.B. Ahmed, and H. Amano, "Implementation of fm-index based pattern search on a multi-fpga system," Applied Reconfigurable Computing. Architectures, Tools, and Applications - 16th International Symposium, ARC 2020, Toledo, Spain, April 1-3, 2020, Proceedings [postponed], ed. F. Rincón, J. Barba, H.K.

So, P.C. Diniz, and J. Caba, *Lecture Notes in Computer Science*, vol.12083, pp.376–391, Springer, 2020.

- [17] Xilinx, Inc., “XilinxVirtualCable.” <https://github.com/Xilinx/XilinxVirtualCable>
- [18] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmailzadeh, 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,” *Proceeding of the 41st Annual International Symposium on Computer Architecture, ISCA '14*, Piscataway, NJ, USA, pp.13–24, IEEE Press, 2014.
- [19] V.V. Skvortsov, M.I. Zvyagina, and A.A. Skitev, “Sharing resources in heterogeneous multitasking computer systems based on FPGA with the use of partial reconfiguration,” *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pp.370–373, Jan. 2018.
- [20] N. Tarafdar, T. Lin, N. Eskandari, D. Lion, A. Leon-Garcia, and P. Chow, “Heterogeneous virtualized network function framework for the data center,” *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp.1–8, Sept. 2017.
- [21] J. Weerasinghe, R. Polig, F. Abel, and C. Hagleitner, “Network-attached fpgas for data center applications,” *2016 International Conference on Field-Programmable Technology (FPT)*, pp.36–43, Dec. 2016.
- [22] P. Moorthy and N. Kapre, “Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster,” *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp.68–75, May 2015.

Miho Yamakura received a B.E. degree in 2019 from Keio University. She is currently a master's student in the Department of Information and Computer Science, Keio University. Her research interest is reconfigurable architectures.

Akram Ben Ahmed received his M.S.E. and Ph.D. degrees in Computer Science and Engineering from the University of Aizu, Japan, in 2012 and 2015, respectively. He later joined Keio University, Japan, as a postdoctoral researcher. He is currently a Research Scientist at the National Institute of Advanced Industrial Science and Technology (AIST), Japan. His research interests include on-chip interconnection networks, reliable and fault-tolerant systems, and low-power embedded systems.

Hideharu Amano received Ph.D. degree from the Department of Electronic Engineering, Keio University, Japan in 1986. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the area of parallel architectures and reconfigurable systems.

Kazuei Hironaka received B.E. degree in Tokai University in 2010, M.E. degree from the Keio University in 2012. He is currently a Ph.D. student in the Department of Information and Computer Science, Keio university, He has been also a reseracher in the Central Research Laboratory of Hitachi, Ltd. since 2012. His research interests are reconfigurable architecture and data storage systems.

Kensuke Iizuka received B.E. degree in 2018 and M.E. degree in 2020 from Keio University. He is currently a Ph.D. student in the Department of Information and Computer Science, Keio University. His research interests are Deep learning accelerators and reconfigurable architectures.