electronics

Article

An Efficient On-Demand Hardware Replacement Platform for Metamorphic Functional Processing in Edge-Centric IoT Applications

Hyeongyun Moon [1] and Daejin Park [1,2,*]

[1] School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Korea; moonhg1209@gmail.com
[2] School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea
* Correspondence: boltanut@knu.ac.kr; Tel.: +82-53-950-5548

Abstract: The paradigm of Internet-of-things (IoT) systems is changing from a cloud-based system to an edge-based system. These changes were able to solve the delay caused by the rapid concentration of data in the communication network, the delay caused by the lack of server computing capacity, and the security issues that occur in the data communication process. However, edge-based IoT systems performance was insufficient to process large numbers of data due to limited power supply, fixed hardware functions, and limited hardware resources. To improve their performance, application-specific hardware can be installed in edge devices, but performance cannot be improved except for specific applications due to a fixed function of an application-specific hardware. This paper introduces an edge-centric metamorphic IoT (mIoT) platform that can use various hardware modules through on-demand partial reconfiguration, despite the limited hardware resources of edge devices. In addition, this paper introduces an RISC-V based metamorphic IoT processor (mIoTP) with reconfigurable peripheral modules. We experimented to prove that the proposed structure can reduce the server access of edges and can be applied to a large-scale IoT system. Experiments were conducted in a single-edge environment and a large-scale environment combining one physical edge and 99 virtual edges. According to the experimental results, the edge-centric mIoT platform that executes the reconfiguration prediction algorithm at the edge was able to reduce the number of server accesses by up to 82.2% compared to our previous study in which the prediction process was executed at the server. Furthermore, we confirmed that there is no additional reconfiguration time overhead even for the large IoT systems.

Keywords: IoT (Internet of Things); hardware reconfiguration; edge computing; metamorphic platform

Citation: Moon, H.; Park, D. An Efficient On-Demand Hardware Replacement Platform for Metamorphic Functional Processing in Edge-Centric IoT Applications. *Electronics* **2021**, *10*, 2088. https://doi.org/10.3390/electronics10172088

Academic Editor: Iouliia Skliarova

Received: 22 July 2021
Accepted: 26 August 2021
Published: 28 August 2021

1. Introduction

The IoT is a system in which diverse social infrastructures such as electronic devices, cars, and buildings are connected through the Internet, and where the connected objects transmit and receive data from each other and operate organically. With the recent universalization of the IoT, the number of things connected to the IoT network has increased rapidly, and, accordingly, the amount of generated data is also increasing rapidly [1,2]. In this paper, both the things that generate data and embedded devices located close to data sources are defined as edges.

Unlike PCs and servers, edges cannot have a stable power supply and sufficient computing resources. As such, in most IoT systems, the edge only handles simple tasks that can be processed with limited computing resources such as data collection, preprocessing, and environmental control. Complex tasks such as inference, situation judgment, and big data processing are processed by sending data to the server. In addition, applications that need to process large numbers of data such as artificial intelligence (AI) and healthcare have been increasing recently [3,4]; however, large numbers of data were transmitted to

the server due to the lack of computing power at the edge. Due to these changes, delays due to network congestion, bandwidth limitations, and server workload limitations are occurring [5,6].

The scope of application of the IoT is also diversifying [7–9]. As the numbers of facilities and objects that are integrated with the IoT—such as smart factories, smart buildings, and smart cars—are increasing, the number of edges that have to process various data and various operations is also increasing. For these requirements, if a general-purpose micro-controller is used, various operations can be performed, but processing performance is very poor. In addition, if a specific application hardware is used, the high-speed processing of specific operations is possible, but it is not possible in other operations. Recently, like ARM NEON, a structure that processes repeated parallel multiply-accumulate (MAC) operations using single instruction multiple data (SIMD) instructions is also used for parallel data processing [10,11]. However, although the architecture that supports SIMD shows some performance improvements, it cannot show more than the parallel architecture hardware designed for specific applications [12]. Therefore, edge-computing-based IoT systems that process data directly at the edge still experience processing delays due to performance limitations [13].

In order to increase the computing performance of the edge, implementing all the necessary hardware is also limited by the environmental constraints of the edge such as limited hardware resources and limited power supply. In general, it is rare that all implemented hardware is active at the same time. In most cases, the hardware has the characteristic that only the necessary parts are partially activated. Therefore, if only the necessary hardware can be implemented in real time, the system will benefit greatly in terms of hardware resources and power [14–16].

A typical example of the above case is a fault tolerance structure or a fault monitor module. The critical section of IoT edge hardware is not fixed but rather changes according to the operation. Therefore, it is necessary to implement data observation functions for various operations of IoT devices [17–19]. However, software-based fault monitoring is inadequate because it has limitations in observing data that change every hour faster than the instruction execution speed and affect the performance of the main application. Therefore, in systems where reliability is important, a hardware-based fault monitor is required. However, implementing all monitors for various critical sections is not appropriate in terms of hardware size, power consumption, and cost due to the characteristic of edge environments. Therefore, it is necessary to efficiently monitor the target system in terms of energy and cost using a system that can implement only the necessary hardware in real time.

This paper proposes a second mIoT platform based on mIoTP in order to efficiently operate hardware in the fault tolerance applications, as shown in Figure 1. The mIoTP is a reconfigurable processor that can reconfigure some peripheral modules on demand. The processor requests the necessary hardware modules to the mIoT server while executing the main application, and the server reconfigures the module in the reconfigurable region by transmitting the requested hardware bitstream to the edge.

It is in the hardware reconfiguration process that the overhead for communication between the edges and the server, the processing of reconfiguration requests in the server, and the reconfiguration of the hardware occur. In a previous work by the authors of this paper, we used a callability-based reconfiguration prediction technique to reduce this overhead, along with a three-layer structure consisting of a main server, edge servers, and edges. The callability is a probabilistic value indicating which hardware will be called in the next processor's operation. Through these techniques, effective reconfiguration management at the server and reduction of reconfiguration time overhead were possible in a large-scale IoT system. However, the bitstream caching algorithm (BCA), which is a callability-based reconfiguration prediction algorithm, was operated on the edge server. Accordingly, the control-flow information of the edge software had to be transmitted to the edge server for reconfiguration prediction. Thus, it was found that unnecessary

server access to transmitting edge operation information occurs excessively, as shown in Table 1. Therefore, in this paper, server access could be further reduced with an edge-centric structure that processes the BCA algorithm at each edge and accesses the server only when prediction fails. Through this structural change, this study was able to reduce server access for the edge hardware reconfiguration by up to 82.2% compared with the author's work. In addition, the changed structure could be applied both to large-scale IoT systems without requiring any additional reconfiguration time overhead and to more dynamic applications by updating the callability look-up table (LUT) at each edge in real time.

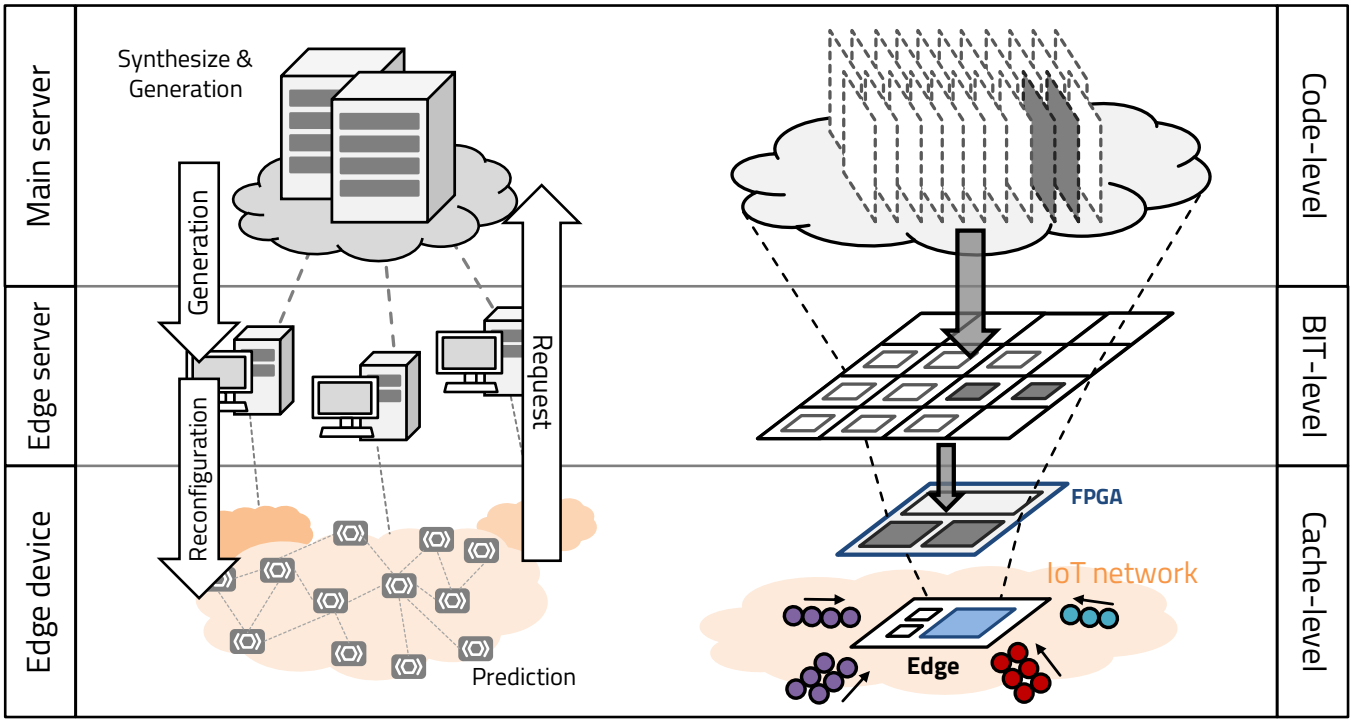


Figure 1. Overall structure of mIoT platform.

Table 1. Type of communication between edge and server.

Type	L1-Miss	Information Transfer	Pre-Reconfiguration
Composition	>0.1%	82%	17.9%
Time(s)	0.8	>0.001	0.8

This paper is organized as follows: Section 1 (Introduction) contains the necessity of the study and a brief description of the study, Section 2 (Background) explains the basic knowledges of the study, Section 3 (Edge-Centric mIoT Platform) explains the proposed structure in detail, Section 4 (implementation) explains the implementation of the proposed structure by applying it to the fault-tolerance case, Section 5 (Experiment) presents an experiment to verify the proposed structure, Section 6 (Discussion) considers the research results in terms of power consumption and reconfiguration time overhead, and finally Section 7 (Conclusions) explains the research conclusion.

2. Background

2.1. Dynamic Partial Reconfiguration (DPR)

The biggest advantage of the field-programmable gate array (FPGA) is its re-programmability. With this advantage, the designer can continuously update the design. However, as with updating the system software firmware, there is a disadvantage in that an operation gap occurs when reconfiguring the entire FPGA logic using a new bit-

stream [20,21]. However, with the DPR technology provided by Xilinx, it is possible to overcome this disadvantage and design more flexible systems. DPR is a technology that can partially reconfigure only the reconfigurable region while ensuring continuous operation of the static region by separating the design into a static logic and a reconfigurable logic during the design flow, as shown in Figure 2 [22,23]. In general, the entire hardware is not activated at the same time. Therefore, if a module with less probability of being used could be replaced with another module, it would be possible to perform more diverse operations with smaller hardware resources [24–26].

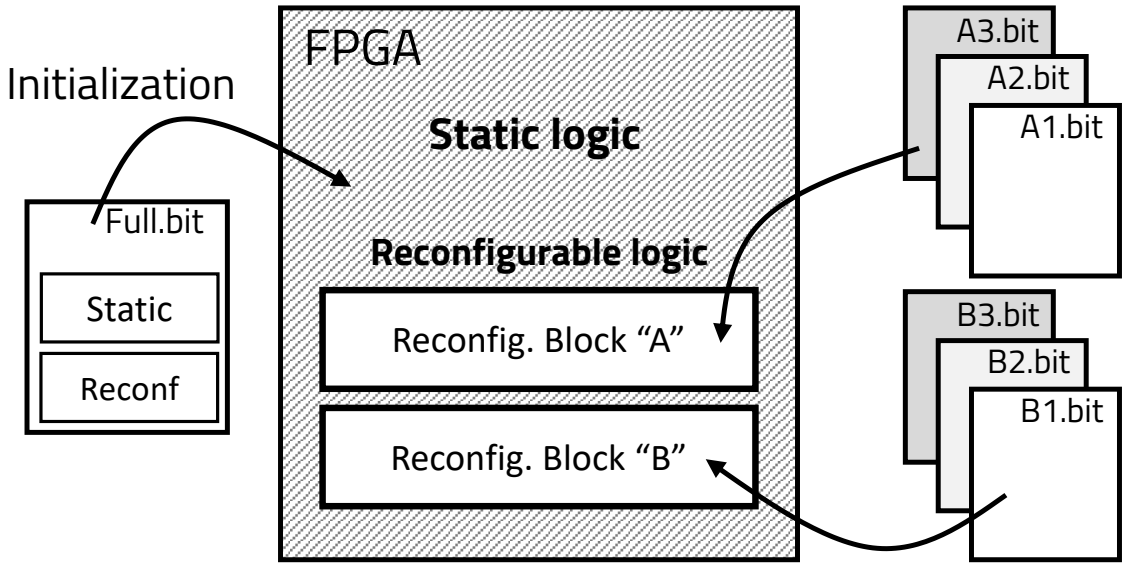


Figure 2. Dynamic partial reconfiguration.

DPR techniques can be classified depending on whether or not an external flash memory is used to store partial bitstreams. If the partial bitstream to be used is stored in an external flash memory, the reconfiguration can be quickly processed. However, there are two drawbacks to this. First, an external memory, such as the SPI-flash memory, is required. In general, external memories are known to consume 100 times more power than an internal memory logic, which is inappropriate in IoT edge environments where the power supply is insufficient. Second, an additional internal logic is needed to read partial bitstream from an external memory and reconfigure the FPGA itself [27]. In addition, software to access the external DRAM memory and several Xilinx's modules such as partial reconfiguration controller (PRC) and internal configuration access port (ICAP) are required. Therefore, the method of using external memory did not fit the purpose of the present research to implement only the necessary internal logic. In addition, since only the partial bitstream pre-stored in an external memory can be used, the operation flexibility of edge devices is limited by the size of the external memory. As a result, DPR techniques that do not use an external memory in IoT edge environments are effective in terms of power consumption, flexibility of operation, and the utilization of internal logic.

2.2. Metamorphic IoT Platform (mIoT)

In general, cloud-based platforms such as Hardware-as-a-Service (HaaS) have been widely adopted for IoT systems in order to overcome the insufficient computing power of the edge [28,29]. However, due to the universalization of the IoT, the number of edges that are connected to servers is increasing, and bottlenecks are occurring due to constraints on network bandwidth [30]. This problem can be partly improved by the recent development of wireless communication technologies such as 5G and Bluetooth, and wired communication technologies such as optical fiber. However, as the volume of data processed by the

server increases, server workload problems occur due to the limited server resources [30]. Thus, the edge cannot reduce processing delays without reducing network and server dependency. On the other hand, if all calculations are processed on an edge, as mentioned before, processing delays also occur due to the poor performance and flexibility of the edge [31,32]. Therefore, this paper proposes a metamorphic IoT platform that can reduce server workload and network dependency compared to the cloud-based system and increase the processing performance and flexibility compared to the edge-based IoT system.

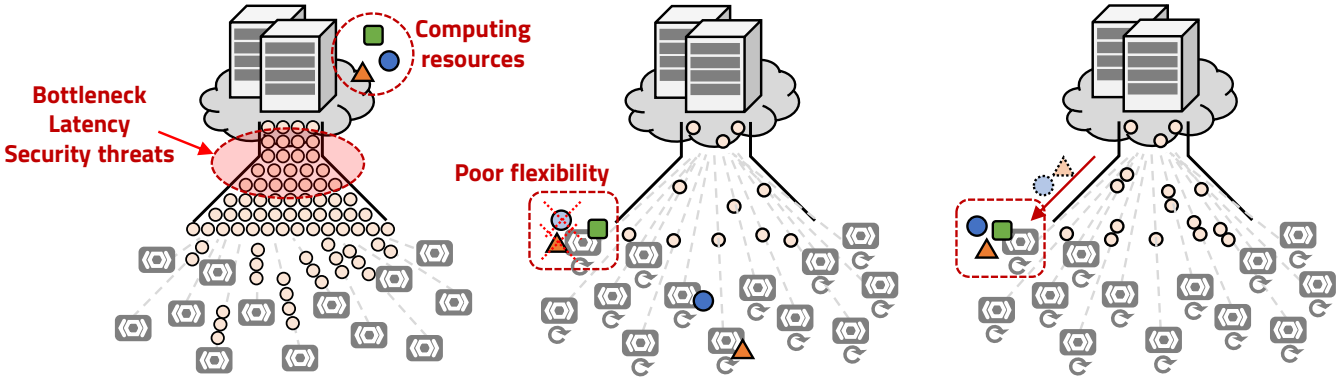
The mIoT platform is an IoT platform that has metamorphism that can change the hardware configuration of the edge in real time depending on the situation [33]. The dictionary definition of metamorphism is "(of rock) changed into a new form and structure by very great heat and pressure;" in other words, its structure and shape are changed by the external environment. The proposed mIoT can be reconfigured with appropriate hardware according to the external environment of the IoT operation and the operation state of the application software.

The proposed platform can overcome network congestion and the delays caused by server resource overload, which are the main problems of the server-centric IoT platform, and improve the lack of performance and flexibility, which are the main problems of the edge-centric IoT platform. The mIoT platform is a three-layer structure composed of edges, edge servers, and main servers and the server reconfigures the edge's FPGA when the edge requests hardware from the server. In other words, the edge requests the hardware from the edge server in a timely manner, and the edge server sends the bitstream to the edge if it has the requested hardware bitstream. If the edge server does not have the requested bitstream, however, the edge server requests the main server to create the bitstream. The edge of the mIoT platform is an FPGA-based device equipped with DPR technology, and the edge logic is designed by dividing it into static logic and reconfigurable logic at the FPGA's design flow. The server reconfigures the requested logic in the reconfigurable region according to the edge operation, while the edge server processes the reconfiguration operation according to the reconfiguration request of the edge, and the main server is responsible for hardware synthesis and bitstream generation. This type of platform structure allows diverse hardware modules and accelerators to be used on the edge devices despite the limited hardware resources while enhancing the performance and flexibility of the edge devices.

The proposed mIoT platform consists of edge devices with an embedded FPGA that can process various data and server structures for reconfiguring the FPGA with low reconfiguration time overhead. To reconfigure an FPGA, network connections are required because the bitstream must be transmitted from the server, but the system is safe from security threats and attackers because data are not transmitted to the outside; however, if reconfiguration is not needed, the data can be processed directly at the edge regardless of the network connection. Thus, the proposed platform offers better security, network independence, and flexibility than existing cloud-centric (CC) and edge-centric (EC) IoT systems. Thus, the CC-based platform shown in Figure 3a can perform powerful and diverse operations by transferring data from the edge to the cloud, but bottlenecks, latency, and security threats can occur; whereas the EC-based platform shown in Figure 3b can address the cloud problem by directly processing data at the edge, although various computing operations are not possible in this case. Finally, as shown in Figure 3c, the mIoT system can not only mitigate the disadvantages of the CC by directly processing data at the data source but can also mitigate the disadvantages of the EC by receiving various hardware functions from the server.

In edge environments, power consumption also needs to be minimized. Edge device FPGAs use dynamic partial reconfiguration (DPR) to reconfigure the hardware, and most DPR-based platforms use external memory to store various bitstreams. However, external memories are unsuitable for edge environments because they require additional power consumption. Therefore, the proposed platform can minimize edge's power consumption by adopting the on-demand reconfiguration method without requiring an external memory. As with power consumption, the reconfiguration overhead should be minimized. To reduce

the FPGA reconfiguration overhead, this study devised a three-layer structure composed of the main server-edge server-edge device and callability-based prediction and prefetching techniques.



(a) Cloud-centric (CC) IoT system (b) Edge-centric (EC) IoT system (c) mIoT-based IoT system

Figure 3. Comparison between CC, EC, and mIoT-based IoT systems.

As edge devices also require a processor to work in FPGAs, it is necessary to design the optimal processor according to the operating environment due to the constraints of the edge environment. Therefore, this study adopted a Chisel (Constructing Hardware In a Scala Embedded Language)-based processor design technique that can easily redesign the optimal processor by parameterizing the hardware module. Finally, a fault-safe reconfigurable platform was built to verify the overall mIoT, and a large-scale simulation with real devices was conducted to verify that the server can manage reconfiguration tasks with low time overhead, even in large-scale applications.

The first mIoT proposed in the author's previous study was a server-centric mIoT platform [33]. The edges transmitted all the operational information needed to predict the next reconfiguration to the edge server, and all the reconfiguration tasks, such as next reconfiguration predictions and callability LUT updates, were processed by the edge server. Thus, excessive communication occurred because large-scale edge devices were managed by each edge server. To solve this problem, this paper proposes a second mIoT platform, namely an edge-centric mIoT platform. The edge-centric structure is a structure that accesses the edge server only when a prediction failure occurs by executing all reconfiguration predictions at each edge. As a result, it is possible to reduce network load and unnecessary communication. In addition, the mIoT structure can also be applied to dynamic applications through real-time callability LUT updates, which are the basis for BCA operations. The edge of the previous study was an ASIC-FPGA co-design structure that implemented static regions such as ASICs and dynamic regions as FPGAs. However, in this study, the entire reconfigurable processor was integrated into the FPGA to simplify the design and make updates easier, while the reconfigurable and static regions were designed separately to prevent the reconfigurable region update process from affecting the operation of the static region.

3. Edge-Centric mIoT Platform

3.1. Metamorphic IoT Processor (mIoTP)

The mIoTP proposed in this paper is the RISC-V based on an on-demand reconfigurable processor that was designed in the FPGA. During the design process, the processor was designed by dividing the logic into static and reconfigurable regions. Thus, the RISC-V core and the basic peripheral modules were implemented in the static regions, while the peripheral modules, which could change depending on the processor operation, were implemented in the reconfigurable regions. In the previous research, the static region was designed as an ASIC, and only the reconfigurable module was implemented in the FPGA.

However, in this paper, the static and reconfigurable regions were integrated into the FPGA in order to simplify the design and make updates easier.

The overall edge hardware structure and operating principles are shown in Figure 4. The RISC-V processor in the FPGA was designed by dividing it into static and reconfigurable regions, and the reconfigurable region can have space to implement one or more modules. Xilinx's DPR technology can determine the number of reconfigurable modules in the implementation stage of the FPGA's design flow. Two or more reconfiguration spaces are similar to the cache in the computer architecture, and it is a structure to increase the accuracy of callability based reconfiguration prediction techniques to reduce reconfiguration overhead, because the number of modules in the reconfiguration region and the probability of having the necessary modules are proportional. Furthermore, the edge was mounted with an external SPI-flash memory for storing the application binaries to be executed by the processor.

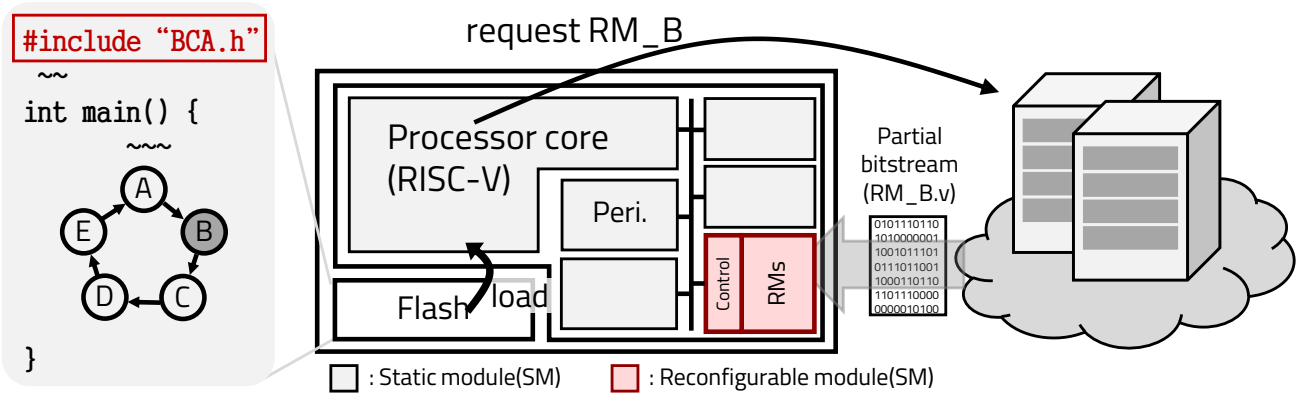


Figure 4. mIoT edge device structure.

The mIoTP reads the main application binary from SPI-flash memory and executes it, and the edge requests the necessary hardware bitstream to the edge server to reconfigure some peripheral modules. Since the reconfiguration process should not affect the core operation of the static region, the DPR technique provided by Xilinx was applied to the design. Thus, the mIoTP can reduce the on-demand reconfiguration overhead through reconfigurable regions divided into one or two spaces and DPR techniques, while the processor can ensure the continuity of static logic operation regardless of the reconfiguration process.

3.2. Callability-Based Bitstream Caching Algorithm

The mIoT platform uses probability-based reconfiguration prediction techniques to minimize the reconfiguration overhead required for the on-demand hardware reconfiguration of edge devices. The probability values for the prediction techniques are called callability. Through the control flow analysis of the software, one can represent a probabilistic value concerning which function will be executed next according to the previous execution history. This probabilistic value is the callability. The mIoT platform minimizes the reconfiguration overhead by pre-implementing modules with high callability on the edge's FPGA before the request. Based on this method, a bitstream caching algorithm (BCA) was proposed with the mIoT platform structure, which is a three-layer structure.

The previous first mIoT platform executed the BCA on each edge server. All of the edges sent their application operation information to the edge server, and the edge server managed the control flow history of all the edge devices. The edge server performed tasks such as saving edges application execution history, updating callability LUTs, and predicting the next module of each edge. The purpose of the mIoT platform was to reduce both network congestion and server workload compared to cloud-based IoT platforms. This purpose was also achieved in the author's first research. Because, the mIoT platform

reconfiguration task on the server required only 8-bit application control flow data for next module prediction. Therefore, only 8-bit information was transmitted from the edge to the server instead of transmitting a large number of data, and the server only had to reconfigure the edge hardware instead of complex data processing. However, unnecessary server access occurred in the process of transmitting the operation information of each edge when large-scale edges were connected to the server. These additional communications occurred to record edge operation history, update callability LUT, and predict next reconfiguration module in the edge server. Thus, if the BCA algorithm operates at the edge, it should be possible to reduce these unnecessary accesses and the network load. Therefore, in this mIoT platform, the next modules were predicted at each edge by managing their own operation information and updating their callability without transmitting operation information to the server. As a result, since the edge accesses the edge server only in situations where reconfiguration is necessary, unnecessary access to the edge server can be reduced.

BCA in edge devices is shown in Algorithm 1. First, a BCA library to replace the server's BCA operation is added to the main application of the edge, as shown in Figure 4. Since the increased size of the main application binary due to the BCA library is approximately 0.3 KB, the additional increase in the size due to the BCA library is small compared to the sizes of the MB flash memory and the 4 KiB instruction cache. The edges update the callability by recording the control flow history in real time. Based on this callability, the reconfigurable peripheral module of the processor is reconfigured by predicting the hardware prior to execution of the function requiring reconfiguration. In this process, if the prediction is hit when a function requiring hardware reconfiguration is called, the edge's processor can operate immediately without any pause and additional reconfigurations. This situation is represented as an L1-hit and the reconfiguration overhead as almost zero. If this prediction is a miss, an additional hardware reconfiguration request must be sent to the edge server, and this situation is represented as an L1-miss. When an L1-miss occurs, the operation of the edge device pauses until the reconfiguration operation is completed. In order to reduce the reconfiguration overhead due to the L1-miss, the divided reconfiguration region case is also presented in previous our work. In the previous work, it was confirmed that the reconfiguration overhead decreases as the number of divided reconfiguration regions increases. In this case, the L1-hit ratio can be increased by implementing the hardware with the highest callability and that with the second highest callability in the reconfiguration region of the edge. This principle is similar to the locality of the cache in computer architecture, and the reconfiguration overhead can be greatly reduced by increasing the L1-hit ratio.

3.3. Metamorphic IoT Server

The mIoT platform server structure consists of edge servers and main servers. The edge servers receive and process the reconfiguration requests of the edges. If the requested hardware bitstream exists in the edge server repository, it immediately sends the bitstream to the edges and finishes the reconfiguration task. This situation is called an L2-hit. However, if the requested hardware bitstream does not exist in the edge server repository, the edge server requests bitstream synthesis and generation to the main server to generate the requested bitstream. This situation is expressed as an L2-miss and requires the largest reconfiguration overhead compared to other cases, as shown in Figure 5. The main server sends the requested hardware bitstream to the edge server through the logic synthesis, place and route (P&R), and bitstream generation process.

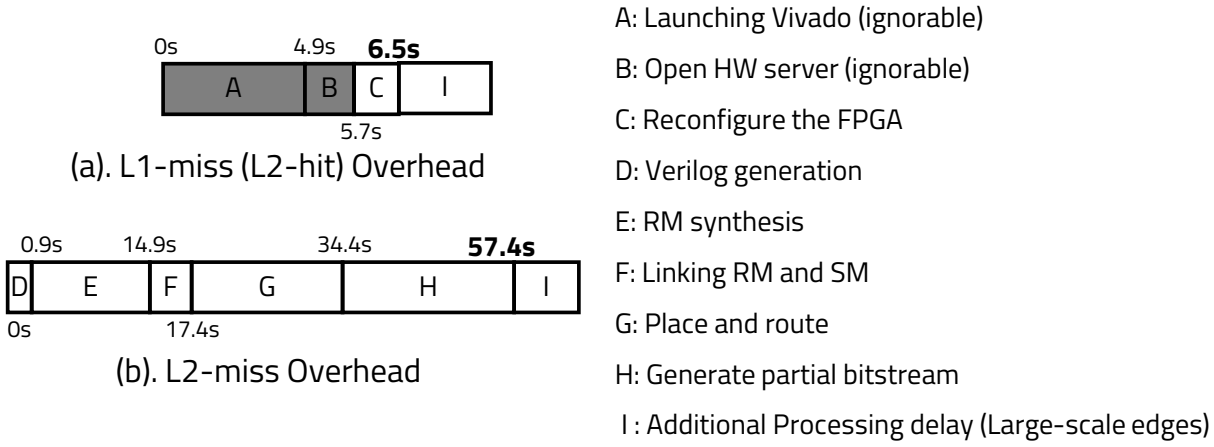


Figure 5. Overhead composition for each case.

Algorithm 1: Edge bitstream caching algorithm.

Goal : Predict and request reconfiguration
BIT_{req} : Partial bitstream requested from the edge
BIT_{prog} : Pre-programmed partial bitstream
E_{recon} : Reconfiguration event at the edge
$FUNC_{RM}$: Programed RM in FPGAs

if E_{recon} **then**
    Check control flow history
    Find the highest callability
    Determine BIT_{req}
    Request BIT_{req}
    Reconfigure BIT_{req}
if *Run* $FUNC_{RM}$ **then**
    Pause processor
    if $BIT_{req} = BIT_{prog}$ **then**
        break // L1-hit
    else
        Request BIT_{req}
        Reconfigure BIT_{req}
    Re-run processor

The mIoT platform server structure is as shown in Figure 6. The first version of the mIoT consisted of a structure in which the BCA was integrated into the reconfiguration processing engine (RPE) of the edge servers. However, in this paper, since the BCA is executed at each edge, the edges only access the edge server when an L1-miss occurs. With this method, the number of server accesses, as well as server workload and network load, was reduced. To summarize the entire server structure, the edge requests the required hardware bitstream according to its application execution, the edge server reconfigures the edge's reconfigurable region using the RPE, and the main server synthesizes and generates the requested hardware bitstreams.

$$t_{total\_avg} = \frac{\sum^{N} t_{recon}}{\sum^{N} E_{total\_recon}} \quad (1)$$

where:
N = Number of edges of the entire system

t_{total_avg} = Reconfiguration time overhead average of all edges
t_{recon} = Reconfiguration time overhead of one edge
E_{total_recon} = Number of reconfiguration events of one edge

$$t_{recon} = t_{miss} + t_{trans} + t_{pre} \quad (2)$$

where:
t_{miss} = Reconfiguration time overhead due to L1-miss
t_{trans} = Reconfiguration time overhead due to edge's information transfer
t_{pre} = Reconfiguration time overhead due to pre-reconfiguration

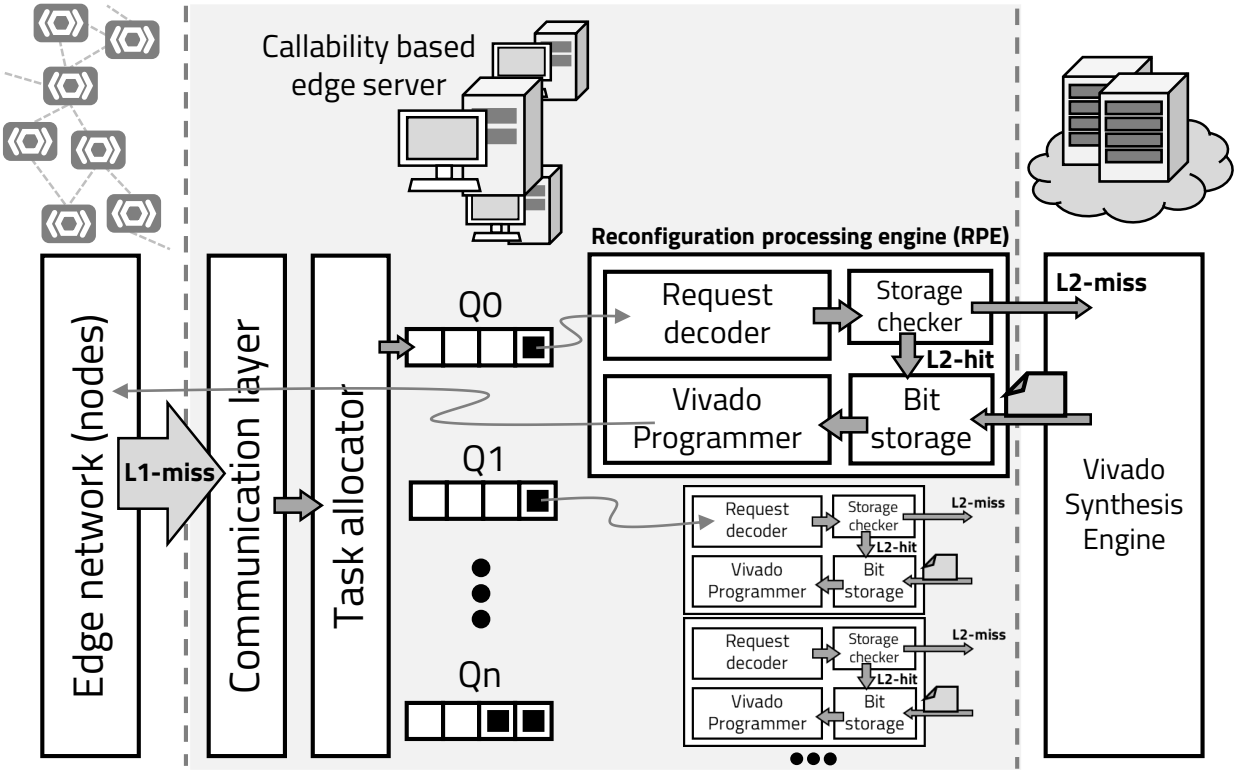


Figure 6. mIoT server structure.

4. Implementation

This paper proposes a metamorphic fault-safe processor (mFSP) that applied the proposed mIoT structure to fault monitoring application. IoT edges mainly operate in an unstable external environment. Since the edges are connected to each other and operate organically, malfunctions generated from a single edge can affect the entire IoT system through the propagation between the edges [34,35]. Therefore, a structure that allows stable operation of the IoT edges is needed [36].

Several techniques for operating the IoT edges safely have been presented in the past. First, techniques to compare processing results using the duplicate and comparison circuits for critical modules in the system or to add redundancy circuits in the system for voting techniques to maintain reliability are mainly used [37–39]. However, these techniques have problems in that both their logic area and power consumption increase due to redundant circuits. The second is the addition of monitoring functions using software or hardware [40,41]. However, software monitoring techniques cannot be applied to clock-level defects, and they also affect the main application performance, because the code for data monitoring is inserted into the main application. In addition, since a general hardware monitoring circuit can only monitor the point specified in the initial design process, it cannot protect the various modules and various data paths. Furthermore, there

is a disadvantage in that the logic area increases as the observing point is added. Therefore, this paper proposes a fault-safe mIoT platform that can reconfigure the monitoring circuit according to the processor's operation in order to overcome these constraints. Since the proposed structure uses a hardware monitoring circuit, the system can be monitored by clock-levels, and since only the necessary monitoring circuits are implemented on demand, the logic area can be reduced.

The mFSP presented in this paper is an RISC-V-based processor that can reconfigure peripheral modules on demand, and its detailed structure is shown in Figure 7. The processor was implemented in Xilinx's Arty FPGA and Freedom E300 SoC platform environments. The processor was designed based on RV32IMAC ISA and has a 16 KiB tightly coupled data memory (TCDM) and a 4 KiB instruction cache. In the processor, the static peripheral modules include UART, SPI, and JTAG, while the reconfigurable modules include system monitoring modules. A main application binary is stored in the external SPI-flash memory, and it is executed as an execute-in-place (XiP). Unlike the previous mIoT, it is possible to minimize server access for the FPGA reconfiguration of the edges by running the application, including the BCA library, on the edges.

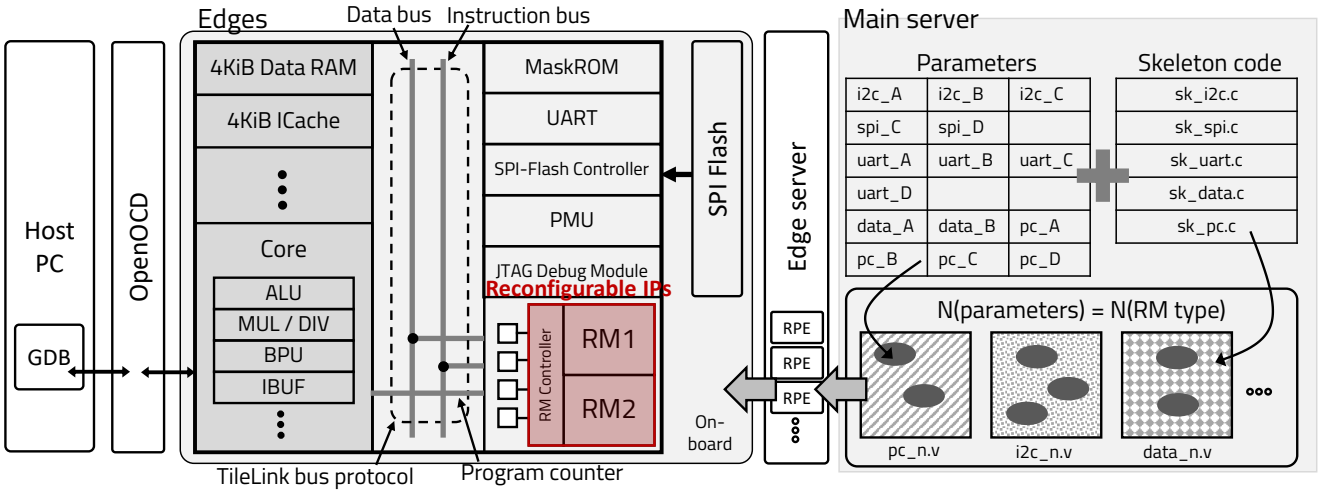


Figure 7. mFSP structure.

The reconfiguration requests generated while executing the applications on the mFSP are transmitted to the edge servers. In the edge servers, the C language and the TCL-based RPE process the reconfiguration requests. If the partial bitstream requested from the edge exists in the edge server, the partial bitstream is immediately transmitted to the edge. If the requested bitstream does not exist, the edge servers request synthesis and bitstream generation by the main server. The main server generates Verilog HDL codes by combining the parameters and the skeleton codes prepared for each application using C language and TCL and performs synthesis, P&R, and bitstream generation, as shown in Figure 7. Finally, the generated bitstream is delivered to the edge through the edge server.

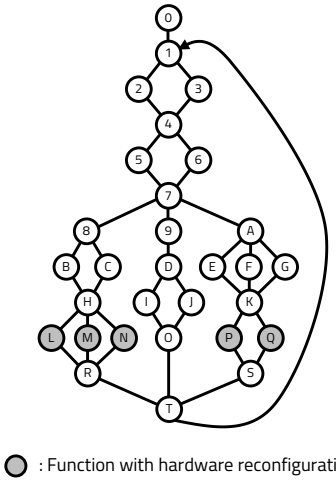
5. Experiment

In order to verify the proposed mFSP platform, the authors built the following experimental environment after designing the edge mFSP based on Xilinx Arty-7 35T FPGA. The RISC-V processor and the reconfigurable peripheral modules were designed in the internal logic of the FPGA, and the main application was stored in the external SPI-flash memory. The edge server was implemented on a laptop, while the main server was implemented on a PC. The server's performance was not critical because most of the tasks are FPGA reconfiguration. All of the server environments used C and TCL-based scripts, Vivado 2018.01 version in Windows environment. In the proposed platform structure, the edge and the server were wirelessly connected to manage the reconfiguration processes.

Several studies have already been published in which FPGAs are programmed and reconfigured wirelessly [42,43]. Therefore, in the actual commercialization stage, the method of wireless programming proposed in other research studies will be adopted, but in the present paper, JTAG-based programming was used for the experiment.

The main application of the edge for measuring performance changes the monitoring spot of the monitoring module according to the situation, and has a control flow, as shown in Figure 8a. In the control flow, a gray circle such as L means a function that requires hardware operation of the reconfigurable region, and a white circle such as A means a function that does not require the hardware operation. To compare the reconfiguration overhead and the server access frequency, the application used in the previous mIoT study was used, and the callability of the application was obtained, as shown in Figure 8b. The callability was obtained through 1000 iterations of main application of the edge in advance, and unlike the first mIoT platform, the callability was updated in real time during the execution of the program.

(a)



(b)

Function			Callability
Stage 1	Stage 2	Stage 3	
5	B	L	0
		M	0.37
		N	0.63
	C	L	0
		M	0
		N	0
	E	P	1.00
		Q	0
	F	P	1.00
		Q	0
	G	P	1.00
		Q	0

Function			Callability
Stage 1	Stage 2	Stage 3	
6	B	L	0
		M	0.15
		N	0.85
	C	L	0.67
		M	0.33
		N	0
	E	P	0
		Q	1.00
	F	P	0.73
		Q	0.27
	G	P	0.63
		Q	0.38

Figure 8. Application control flow and extracted callability.

The result of reducing server access in the edge-centric mIoT platform is shown in Figure 9 and Table 2. The main application of the edge was repeatedly executed 1000 times, and the edge server used three RPEs, and as a result, the number of server accesses was compared with the results of previous research. In the edge-centric structure, the single RM structure with a single reconfiguration space reduced server access by 69.7% compared to the server-centric mIoT. In addition, server access could be reduced by 82.2% in the double RMs structure with two reconfiguration spaces. Futhermore, this reduction had the effect of reducing the power consumed by communications between the edges and the servers. Since the edge's mFSP has an instruction cache size of 4 KiB, designers should also pay attention to the instruction code size of the edge. Therefore, this study can confirm that the increase in binary size due to the inclusion of the BCA in the edge application is 352 bytes, which is about 8.6% of the total size of the instruction cache. Therefore, although the BCA operation was added in the edge, it was confirmed that the proposed structure could operate without any additional reconfiguration overhead. In Figure 10, 100 nodes could be managed with an average of 0.21 s reconfiguration time overhead using three RPEs.

Table 2. Edge-centric mIoT platform experiment results.

Type	[33] (Double RMs)	EC mIoT (Single RM)	EC mIoT (Double RMs)
# Server access	2716	824	484
# Server access (relative)	1.000	0.303	0.178
BCA Lib size	-	352 byte	352 byte
BCA Lib size/ICache size	-	8.6%	8.6%

EC = Edge-centric; BCA Lib size = BCA library binary size = Increased edge's application binary size; BCA Lib size/ICache size = BCA library binary size versus edge's instruction cache size (4 KiB).

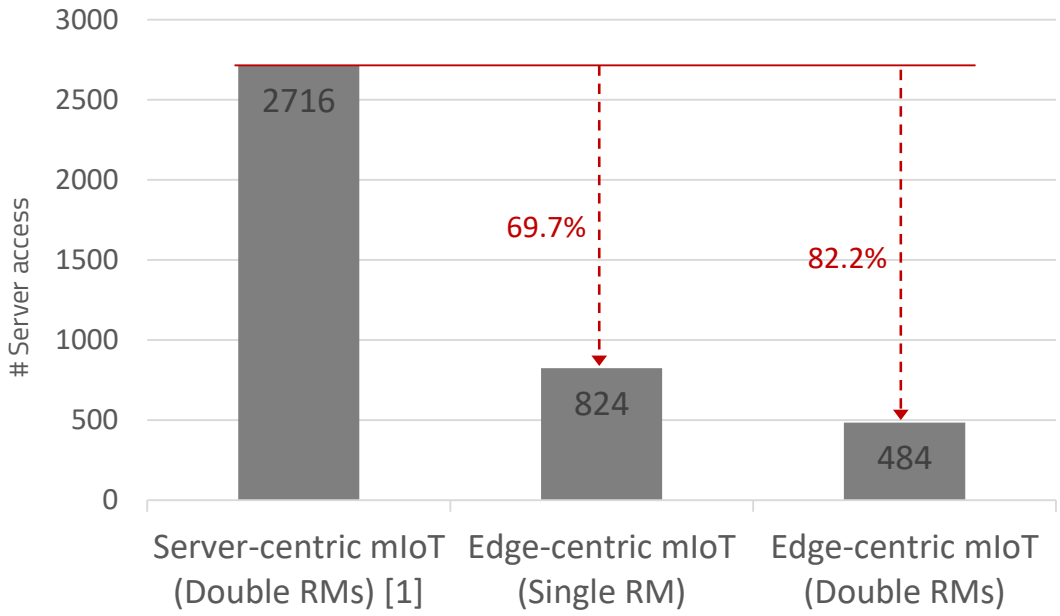


Figure 9. Edge-centric mIoT platform experiment results.

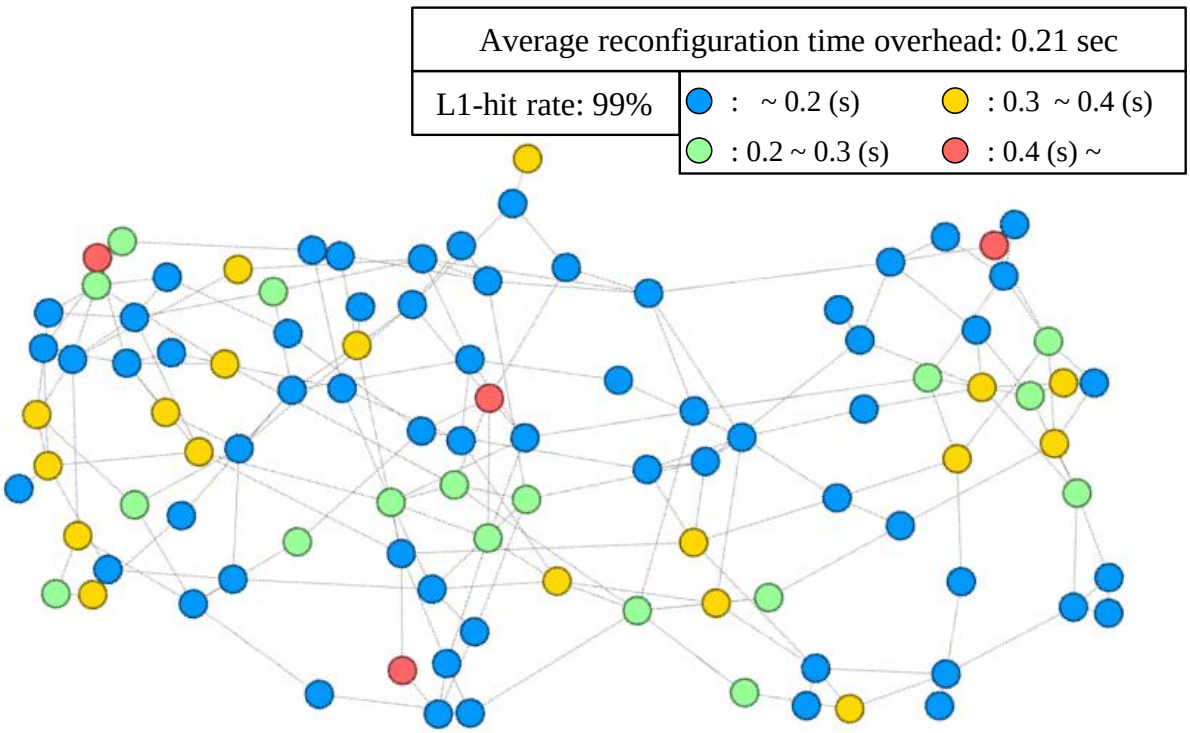


Figure 10. Large-scale simulation results with reconfiguration overheads.

The equation for calculating the reconfiguration time overhead t_{total_avg} is shown in Equation (1). Since each IoT edge operation changes depending on the situation, the average value of the entire system is calculated to obtain the average reconfiguration time overhead. Therefore, the sum of the reconfiguration time overhead t_{recon} consumed at every edge is divided by the sum of the number of reconfiguration events E_{total_recon} occurring at every edge [33]. This is about 5% faster than 0.22 s, which was the result obtained using the same configuration in the previous server-centric mIoT, but it cannot be clearly established that the reconfiguration overhead decreased due to the characteristic of large-scale edges operating independently and accessing the server irregularly. Depending on the order of server access by multiple edges and the execution of programs at each edge, the edge server's reconfiguration processing operation may be slightly different. Therefore, it is concluded that the edge-centric mIoT platform can greatly reduce unnecessary server access; and even when a reconfiguration operation is added to the edges, it can be applied without any additional reconfiguration overhead in a large-scale configuration.

6. Discussion

Power consumption: The results of this study showed that server access can be reduced by up to 82.2%. The reason for this result was that the edge operation information transmitted to the server was reduced by processing the reconfiguration prediction task at the edge. External communication and internal processing always show a trade-off relationship in terms of power consumption, but it was confirmed that external communication consumes more power than simple pre-processing, as in the author's previous study [44]. Thus, although simple prediction operations were added to the edge, server access could be greatly reduced, making it reasonable to expect a reduction in power consumption.

Reconfiguration time overhead: This study confirmed that there was no significant difference in the reconfiguration time overhead, although the server access for reconfiguration was drastically reduced. This is because the task that takes a large portion of the communication between edges and servers and the task that takes a long processing time are different, as shown in Table 1 and Equation (2). Therefore, it is important to reduce the pre-configuration time overhead, because pre-configuration requires the same time as an L1-miss but takes a large portion of the communication. Based on our previous work, if we store the partial bitstream at the edge and reconfigure it directly on the edge, the edge will be able to complete the reconfiguration in a very short time without server access [45]. Therefore, t_{pre} of Equation (2) can be converged to almost 0, and the total reconfiguration time overhead can be close to 0 depending on the accuracy of the bitstream caching prediction.

7. Conclusions

This paper proposed a reconfigurable processor-based edge hardware that can partially reconfigure peripheral modules on demand and an IoT platform that can efficiently manage the reconfiguration process of the edges. It also proposes a three-layer structure consisting of edges, edge servers, a main server, and a callability-based BCA in order to reduce the overhead required for on-demand reconfiguration of the edges. Finally, it proposes an edge-centric mIoT platform that can reduce unnecessary server access by operating the BCA at each edge, and a reconfigurable processor structure in which reconfigurable peripheral modules and a static RISC-V core are integrated. With the proposed structure, server access can be reduced by up to 82.2%, even when the code size increases by 352 bytes (8.6% of the instruction cache's size), as the BCA was added to the edge application. In addition, the proposed structure could be applied equally to a large-scale edge configuration without any additional reconfiguration overhead. In this study, we reduced the number of server accesses of the edge by inserting BCA into the edge, but the reconfiguration time overhead could not be significantly reduced because the server access for pre-reconfiguration could not be reduced. However, by analyzing the reconfiguration overhead based on the experimental results, we found that if bitstream

storage memory and hardware logic for self-configuration were added to edge, the reconfiguration time overhead could be significantly reduced by reducing pre-configuration overhead. Therefore, in future works, we will reduce the reconfiguration time overhead by adding a self-reconfiguration circuit and bitstream storage memory to the mIoT device structure. Through this, various functions will be able to operate without pause on the minimum hardware.

Author Contributions: H.M. wrote the entire manuscript and designed core architecture and performed the software/hardware implementation; D.P. proposed main concept of the proposed architecture and designed system software architecture and also devoted his role as principle investigator and the corresponding author. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (4199990113966, 10%), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2019R1A2C2005099, 10%), and Ministry of Education (NRF-2018R1A6A1A03025109, 10%, NRF-2020R1I1A1A01072343, 10%). This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 60%).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Zhu, C.; Leung, V.C.M.; Shu, L.; Ngai, E.C. Green Internet of Things for Smart World. *IEEE Access* **2015**, *3*, 2151–2162. [CrossRef]
2. Wen, Z.; Yang, R.; Garraghan, P.; Lin, T.; Xu, J.; Rovatsos, M. Fog Orchestration for Internet of Things Services. *IEEE Internet Comput.* **2017**, *21*, 16–24. [CrossRef]
3. De Vito, L.; Picariello, E.; Picariello, F.; Rapuano, S.; Tudosa, I. A Dictionary Optimization Method for Reconstruction of ECG Signals after Compressed Sensing. *Sensors* **2021**, *21*, 5282. [CrossRef]
4. Porkodi, V.; Yuvaraj, D.; Khan, J.; Karuppusamy, S.A.; Goel, P.M.; Sivaram, M. A Survey on Various Machine Learning Models in IOT Applications. In Proceedings of the 2020 International Conference on Computing and Information Technology (ICCIT-1441), Tabuk, Saudi Arabia, 9–10 September 2020; pp. 1–4. [CrossRef]
5. Yousefpour, A.; Ishigaki, G.; Gour, R.; Jue, J.P. On Reducing IoT Service Delay via Fog Offloading. *IEEE Internet Things J.* **2018**, *5*, 998–1010. [CrossRef]
6. Liu, J.; Luo, K.; Zhou, Z.; Chen, X. ERP: Edge Resource Pooling for Data Stream Mobile Computing. *IEEE Internet Things J.* **2019**, *6*, 4355–4368. [CrossRef]
7. Perera, C.; Liu, C.H.; Jayawardena, S.; Chen, M. A Survey on Internet of Things From Industrial Market Perspective. *IEEE Access* **2014**, *2*, 1660–1679. [CrossRef]
8. Baker, S.B.; Xiang, W.; Atkinson, I. Internet of Things for Smart Healthcare: Technologies, Challenges, and Opportunities. *IEEE Access* **2017**, *5*, 26521–26544. [CrossRef]
9. Wollschlaeger, M.; Sauter, T.; Jasperneite, J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Ind. Electron. Mag.* **2017**, *11*, 17–27. [CrossRef]
10. Meloni, P.; Capotondi, A.; Deriu, G.; Brian, M.; Conti, F.; Rossi, D.; Raffo, L.; Benini, L. NEURAghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs. *ACM Trans. Reconfigurable Technol. Syst.* **2018**, *11*, 1–24. [CrossRef]
11. Zhong, G.; Dubey, A.; Tan, C.; Mitra, T. Synergy: An HW/SW Framework for High Throughput CNNs on Embedded Heterogeneous SoC. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–23. [CrossRef]
12. Lee, K.; Kong, J.; Kim, Y.G.; Chung, S.W. Memory streaming acceleration for embedded systems with CPU-accelerator cooperative data processing. *Microprocess. Microsyst.* **2019**, *71*, 102897. [CrossRef]
13. Vogler, M.; Schleicher, J.M.; Inzinger, C.; Dustdar, S. Optimizing Elastic IoT Application Deployments. *IEEE Trans. Serv. Comput.* **2018**, *11*, 879–892. [CrossRef]
14. Ma, L.; Sham, C.W.; Sun, J.; Valencia Tenorio, R. A Real-Time Flexible Telecommunication Decoding Architecture Using FPGA Partial Reconfiguration. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 2149–2153. [CrossRef]
15. Samir, N.; Gamal, Y.; El-Zeiny, A.N.; Mahmoud, O.; Shawky, A.; Saeed, A.; Mostafa, H. Energy-Adaptive Lightweight Hardware Security Module using Partial Dynamic Reconfiguration for Energy Limited Internet of Things Applications. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–4. [CrossRef]

16. Rihani, M.A.; Nouvel, F.; Prevotet, J.C.; Mroue, M.; Lorandel, J.; Mohanna, Y. Dynamic and partial reconfiguration power consumption runtime measurements analysis for ZYNQ SoC devices. In Proceedings of the 2016 International Symposium on Wireless Communication Systems (ISWCS), Poznan, Poland, 20–23 September 2016; pp. 592–596. [CrossRef]
17. Kim, H.; Lee, D.; Cho, J.; Park, D. Software execution freeze-safe microcontroller using power profile tracking for IoT-driven connected services. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; pp. 237–240. [CrossRef]
18. Duarte, R.P.; Neto, H.C. Stochastic Processors on FPGAs to Compute Sensor Data Towards Fault-Tolerant IoT Systems. In Proceedings of the 2018 IEEE Conference on Dependable and Secure Computing (DSC), Kaohsiung, Taiwan, 10–13 December 2018; pp. 1–8. [CrossRef]
19. Wang, C.; Vo, H.T.; Ni, P. An IoT Application for Fault Diagnosis and Prediction. In Proceedings of the 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, Australia, 11–13 December 2015; pp. 726–731. [CrossRef]
20. Sadeghi, M.; Razavi, S.A.; Zamani, M.S. Reducing Reconfiguration Time in FPGAs. In Proceedings of the 2019 27th Iranian Conference on Electrical Engineering (ICEE), Yazd, Iran, 30 April–2 May 2019; pp. 1844–1848. [CrossRef]
21. Xie, J.; Wang, Y.; Chen, L.; Wang, J.; Wang, Y.; Lai, J.; Tong, J. Fast configuration architecture of FPGA suitable for bitstream compression. In Proceedings of the 2009 IEEE 8th International Conference on ASIC, Changsha, China, 20–23 October 2009; pp. 126–130. [CrossRef]
22. Lie, W.; Wu, F.Y. Dynamic Partial Reconfiguration in FPGAs. In Proceedings of the 2009 Third International Symposium on Intelligent Information Technology Application, Nanchang, China, 21–22 November 2009; Volume 2, pp. 445–448. [CrossRef]
23. Vipin, K.; Fahmy, S.A. ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq. *IEEE Embed. Syst. Lett.* **2014**, *6*, 41–44. [CrossRef]
24. Paulino, N.M.C.; Ferreira, J.C.; Cardoso, J.M.P. Dynamic Partial Reconfiguration of Customized Single-Row Accelerators. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2019**, *27*, 116–125. [CrossRef]
25. Tej, B.; Hannachi, M.; Abdelali, A.B. Partial Dynamic Reconfiguration for efficient Adaptive Implementation of a Video Shot Boundary Detection System. In Proceedings of the 2020 4th International Conference on Advanced Systems and Emergent Technologies, Hammamet, Tunisia, 15–18 December 2020; pp. 327–331. [CrossRef]
26. Youssef, E.; Elsemary, H.A.; El-Moursy, M.A.; Khattab, A.; Mostafa, H. Energy Adaptive Convolution Neural Network Using Dynamic Partial Reconfiguration. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 9–12 August 2020; pp. 325–328. [CrossRef]
27. Daoud, L.; Hussein, F.; Rafla, N. Real-time Bitstream Decompression Scheme for FPGAs Reconfiguration. In Proceedings of the 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), Windsor, ON, Canada, 5–8 August 2018; pp. 1082–1085. [CrossRef]
28. Botta, A.; de Donato, W.; Persico, V.; Pescape, A. Integration of Cloud computing and Internet of Things: A survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684 – 700. [CrossRef]
29. Stanik, A.; Hovestadt, M.; Kao, O. Hardware as a Service (HaaS): Physical and virtual hardware on demand. In Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, Taipei, Taiwan, 3–6 December 2012; pp. 149–154. [CrossRef]
30. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C.T. Edge of Things: The Big Picture on the Integration of Edge, IoT and the Cloud in a Distributed Computing Environment. *IEEE Access* **2018**, *6*, 1706–1717. [CrossRef]
31. Ferdian, R.; Aisuwarya, R.; Erlina, T. Edge Computing for Internet of Things Based on FPGA. In Proceedings of the 2020 International Conference on Information Technology Systems and Innovation (ICITSI), Bandung, Indonesia, 19–23 October 2020; pp. 20–23. [CrossRef]
32. Gomes, T.; Pinto, S.; Gomes, T.; Tavares, A.; Cabral, J. Towards an FPGA-based edge device for the Internet of Things. In Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), Luxembourg City, Luxembourg, 8–11 September 2015; pp. 1–4. [CrossRef]
33. Lee, D.; Moon, H.; Oh, S.; Park, D. mIoT: Metamorphic IoT Platform for On-Demand Hardware Replacement in Large-Scaled IoT Applications. *Sensors* **2020**, *40*, 3337. [CrossRef]
34. Diaz, M.; Martin, C.; Rubio, B. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *J. Netw. Comput. Appl.* **2016**, *67*, 99 – 117. [CrossRef]
35. Roman, R.; Najera, P.; Lopez, J. Securing the Internet of Things. *Computer* **2011**, *44*, 51–58. [CrossRef]
36. Esposito, C.; Castiglione, A.; Pop, F.; Choo, K.R. Challenges of Connecting Edge and Cloud Computing: A Security and Forensic Perspective. *IEEE Cloud Comput.* **2017**, *4*, 13–17. [CrossRef]
37. Ostanin, S.; Matrosova, A.; Butorina, N.; Lavrov, V. A fault-tolerant sequential circuit design for soft errors based on fault-secure circuit. In Proceedings of the 2016 IEEE East-West Design Test Symposium (EWDTS), Yerevan, Armenia, 14–17 October 2016; pp. 1–4. [CrossRef]
38. Thati, V.B.; Vankeirsbilck, J.; Boydens, J. Comparative study on data error detection techniques in embedded systems. In Proceedings of the 2016 XXV International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 12–14 September 2016; pp. 1–4. [CrossRef]

39. Veljkovi, F.; Riesgo, T.; de la Torre, E. Adaptive reconfigurable voting for enhanced reliability in medium-grained fault tolerant architectures. In Proceedings of the 2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Montreal, QC, Canada, 15–18 June 2015; pp. 1–8. [CrossRef]
40. Choi, K.; Park, D.; Cho, J. SSCFM: Separate Signature-Based Control Flow Error Monitoring for Multi-Threaded and Multi-Core Environments. *Electronics* **2019**, *8*, 166. [CrossRef]
41. An, J.; Seok, M.G.; Park, D. Automatic on-chip backup clock changer for protecting abnormal MCU operations in unsafe clock frequency. *IEICE Electron. Express* **2016**, *13*, 20160808. [CrossRef]
42. Gao, T.; Xu, X.; Zhang, H.; Yang, H. A highly-integrated wireless configuration circuit for FPGA chip. In Proceedings of the 2014 International Symposium on Integrated Circuits (ISIC), Singapore, 10–12 December 2014; pp. 260–263. [CrossRef]
43. Adly, I.; Ragai, H.; Shehata, K.; Al-Henawy, A. Wireless Configuration Controller Design for FPGAs in Software Defined Radios. *Online J. Electron. Electr. Eng. OJEEE* **2010**, *2*, 293.
44. Kwak, J.; Lee, S.; Cho, J.; Park, D. Lightweight Polygonal Approximation-Based ECG Signal Processing Platform. In Proceedings of the 2019 IEEE International Conference on Dependable, Autonomic and Secure Computing, Fukuoka, Japan, 5–8 August 2019; pp. 819–824. [CrossRef]
45. Moon, H.; Cho, J.; Park, D. Reconfigurable Fault-Safe Processor Platform Based on RISC-V for Large-Scaled IoT-Driven Applications. In Proceedings of the 2019 IEEE International Conference on Dependable, Autonomic and Secure Computing, Fukuoka, Japan, 5–8 August 2019; pp. 627–632. [CrossRef]