rrBox: Remote Dynamically Reconfigurable Middlebox using NetFPGA

Tze Hon Tan*, Chia Yee Ooi†, M. N. Marsono*
*Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
†MJIIT, Universiti Teknologi Malaysia Kuala Lumpur, Jalan Semarak, 54100 Kuala Lumpur, Malaysia
Email: thtan5@live.utm.my, ooichiayee@ic.utm.my, nadzir@fke.utm.my

***Abstract*—This paper presents a remote dynamically reconfigurable middlebox using NetFPGA 10G development board. The packet forwading algorithm in this middlebox can be updated remotely through the 1Gbps Ethernet connection without a host computer. The proposed architecture uses a customized reconfiguration controller and the Internal Configuration Access Port (ICAP) available in the reconfigurable device. Functional update is important to patch design flaws and bugs, to optimize design performance, and to cope with the changing of execution unit's functional requirement. Based on the experimental result, the implemented middlebox achieved roughly 352Mbps reconfiguration throughput.**

***Index Terms*—Remote dynamic reconfiguration, Partial reconfiguration, Self-reconfiguration, NetFPGA 10G, Middlebox.**

I. INTRODUCTION

In reconfigurable computing, dynamic reconfiguration is a vital feature that enables updates in the field, improves the area utilization and allows defect compensation [1]. Dynamic reconfiguration provides a solution to update a system so it can be adapted to changes in both functional and performance requirement. The trend in network processing shows that high performance requirement is needed, in the network devices especially the throughput to cope with the growing bandwidth demands [2]. Flexibility is another important requirement of network application to update its functionalities [3] as some of the execution requirements were unknown during the design time. Based on these requirements, reconfigurable device like FPGA is a good option to implement network application as FPGA has both performance advantages in ASIC solution and flexibility advantage in software solution [2].

Most network application especially middleboxes are required to operate in high throughput and distributed. Additionally, middleboxes are required to remain active and operational. The cost to update such system is very high and this problem can be solved by utilizing the dynamic reconfiguration feature found in the reconfigurable device for remote functional update. However, the utilization of dynamic reconfiguration feature is not straightforward and requires proper methodology in the design process. Therefore, a good framework to efficiently reconfigure the reconfigurable device is required for network applications.

In this work, a remote dynamically reconfigurable middlebox architecture is proposed. The proposed architecture is implemented and tested experimentally using NetFPGA 10G development board. This works aims to enable remote dynamic reconfiguration in NetFPGA 10G for functional updates purposes. In order to achieve a single-chip solution, the proposed architecture utilizes a customized reconfiguration controller together with Internal Configuration Access Port (ICAP) without relying on a General Purpose Processor or a host computer to handle remote dynamic reconfiguration processes. Furthermore, the 1Gbps Ethernet port attached to the NetFPGA 10G improved the partial bitstream transmission speed.

II. RELATED WORK

Naous et al. [4] developed the NetFPGA development board to enable fast prototyping of network equipment. Antichi et al. [5] presented the path to migrate existing 1G design into the new NetFPGA 10G development board. The framework of NetFPGA is versatile and an official web repository is provided to store the open source projects.

By using NetFPGA, Yin et al. [6] demonstrated customizable virtual network by using partial reconfiguration. In this work, the dynamic virtual network allocation relies on the dynamic reconfiguration feature in FPGA. The authors used JTAG interface to load the partial bitstream dynamically into the FPGA device. Based on [7], loading partial bitstream with JTAG is less efficient compared to using ICAP.

Similarly, Pontarelli et al. [8] used NetFPGA to develop a FPGA-based Network Intrusion Detection System (NIDS). In order to improve the utilization of logic resources, the authors exploit the dynamic reconfiguration feature in FPGA. Additionally, the implemented system is capable to work at wire speed. However, this work also uses JTAG that is inefficient [7].

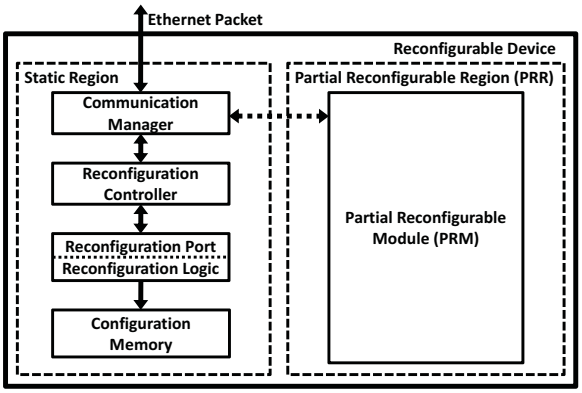

Figure 1. Framework architecture

Lastly, Zhang et al. [9] designed a security system using NetFPGA development board and Virtex5 device. In the proposed work, Virtex5 device is responsible to analyze captured attacks and forward the result to NetFPGA for updates. However, the remote reconfiguration on the NetFPGA is not standalone and requires control from the host PC for bitstream transmission and translation.

The works in [10], [11], [12], [13], [14] are various designs and implementations of dynamically reconfigurable platform. Most of these works provide reconfiguration at low throughput.

III. PLATFORM OVERVIEW

Figure 1 illustrates the framework architecture for remote dynamically reconfigurable platform. The platform mainly consists of the Static Region and Partial Reconfigurable Region. The Static Region includes the Communication Manager to handle Ethernet packet transmission and Reconfiguration Controller to handle the loading of partial bitstream. In Xilinx FPGAs, the Reconfiguration Port is instantiated with the ICAP primitive, while the Reconfiguration Logic and Configuration Memory are not visible to the designer. In the Partial Reconfigurable Region, the Partial Reconfigurable Module is linked to the Communication Manager for internal communication.

IV. PLATFORM IMPLEMENTATION

The NetFPGA 10G development board is chosen for implementation as the NetFPGA 10G development board comes with a Virtex 5 reconfigurable device that supports dynamic reconfiguration and SFP+ cages that support gigabit Ethernet module. Additionally, NetFPGA 10G framework shown in the Figure 2 provides the flexibility for functional extension. Explicitly, extended functionalities can be implemented in the form of Xilinx IP core and are integrated into the system using Xilinx

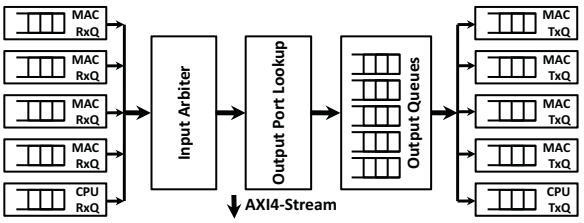

Figure 2. NetFPGA 10G Reference Pipeline [5]

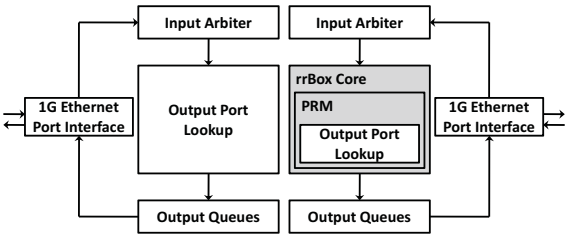

Figure 3. Top level architecture of NetFPGA 10G (left) and rrBox (right)

Platform Studio. IP cores are linked using the AMBA AXI4-Stream Interconnect that allows high bandwidth and unidirectional data transfers. Essentially, NetFPGA 10G development board is developed for the purpose of network applications prototyping.

By using the NetFPGA 10G framework as the basis for the implementation, Figure 3 shows the top level block diagram of both NetFPGA 10G and rrBox, while Figure 4 shows the functional block diagram of rrBox. By utilizing well established Ethernet frameworks and 1Gbps Ethernet ports on NetFPGA 10G development board, the remote reconfiguration becomes more reliable and partial bitstream transmission is faster. In this work, the NetFPGA 10G is functioning in standalone mode and host computer is not required. All hardware designs were behaviorally described using Verilog Hardware Description Language and synthesized using Xilinx ISE DS 13.4. Furthermore, all hardware designs were verified using ModelSim simulation before tested experimentally in the NetFPGA 10G development board.

A. Partial Reconfiguration Design Flow

Xilinx provides several design flows for partial reconfiguration: modular method, difference-based method, small bit manipulation method, early access method and partition-based method. This research uses the partition-based method because the other methods does not support Xilinx Virtex 5. Partition-based method is similar but less complex than the early access method.

The major design flow in the partition-based method includes modeling and synthesis on all functional modules, defining partial reconfigurable module, defining de-

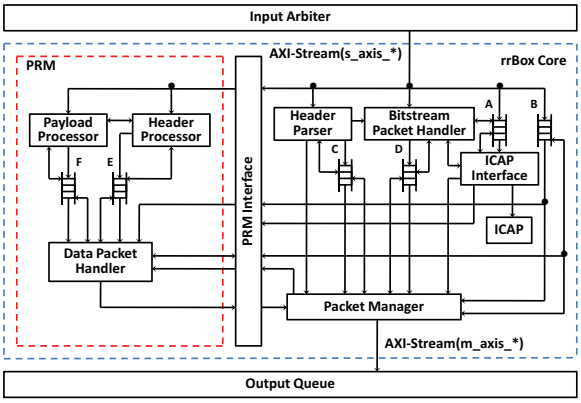

Figure 4. Functional block diagram of rrBox

Table I
THE LIST OF FIFOS

FIFO Name	Role in the platform
A: Bit_fifo	Storing extracted partial bitstream for loading into reconfiguration port
B: In_fifo	Storing received Ethernet packets for forwarding to destination port
C: Hdr_fifo	Storing the extracted packet header informations
D: Bit_stat_fifo	Storing the extraction status of partial bitstream
E: Dst_port_fifo	Storing the destination port to forward the received packets

sign constraints and generating bitstream. The constraint file defines the area and location of the Partial Reconfigurable Region, the timing constraints of the implemented design and the location of physical pins. The PlanAhead supports the partition-based method and provides user-friendly graphical user interface to implement designs with partial reconfiguration.

B. FIFOs

Table I shows the list FIFOs and their role in the platform. These FIFOs are implemented using the on-chip BlockRAM in the FPGA device and functioning as the interface between blocks.

C. Header Parser

The Header Parser is responsible to parse the header of received packets and store the results in the Hdr_fifo. The header information extracted from the received packets includes the source and destination MAC address, source and destination IP address, UDP port number and segment number of bitstream packet. Besides that, this block differentiates bitstream packet from data packet based on the UDP port number and the unique identifier in the packet payload. Each platform has a unique device ID in the packet payload for identification purposes and to enable mass deployment.

D. Bitstream Packet Handler

The Bitstream Packet Handler is responsible to extract partial bitstream from the bitstream packets. The extracted partial bitstream is temporarily stored in the Bit_fifo waiting for the acknowledgement before loading into the ICAP. This block also handles the bitstream packet verification and keeping track on the storage status of the partial bitstream. Any failure arises in storing the partial bitstream (due to the full FIFO) will result in unmarked bitstream packets. A Unmarked bitstream packet is the acknowledgement to request retransmission from rrBox Client. Each received bitstream packets is verified by the subsequent bitstream packet, while the bitstream termination packet will verify the last segment of the partial bitstream. The verification mechanism is implemented to ensure the partial bitstream is stored and loaded into the ICAP in a proper sequence.

E. ICAP Interface

The ICAP Interface is responsible to handle the loading of the partial bitstream into the ICAP. Due to the verification mechanism, the ICAP Interface begins by loading the first segment of the partial bitstream into the ICAP upon the arrival of second segment. The loading of each partial bitstream segment takes several clock cycles because the ICAP bus width is only 32-bit. The ICAP Interface is essential because proper signals assertion and control are required to load the partial bitstream to the ICAP.

F. Header Processor

The Header Processor is responsible to process data packets based on the header information. For example, the Header Processor of a switch controls the data packet forwarding based on the source and destination MAC address in the packet. Additionally, the Header Processor can be designed to drop the data packet based on predefined conditions. The packet forwarding algorithm in middlebox is implemented in the Header Processor. The Header Processor is designed and customized based on the application requirements.

G. Data Packet Handler

The Data Packet Handler is responsible to forward the data packet to the Packet Manager based on the result from the Header Processor. The Data Packet Handler can be designed to do alteration in the data packet based on the application requirements. Additionally, the Data Packet Handler can be extended to support flow managment by adding additional FIFO to store data packets.

H. Packet Manager

The Packet Manager acts as the coordinator for the rrBox Core. In general, the Packet Manager manages major processes in the rrBox Core by offloading processes to handlers. Thus, either the Bitstream Packet Handler or Data Packet Handler will be activated by the Packet Manager based on the process demand. Once the handler is activated, Packet Manager will wait for the handler to complete the offloaded task before proceeding to the next task. During the dynamic reconfiguration, the Packet Manager handles data packets on behalf of the Data Packet Handler. Additionally, the Packet Manager is responsible to initialize the Partial Reconfigurable Module after the remote reconfiguration process has been completed. In order to reduce the logic resources used, the communication protocol used for partial bitstream transmission is the User Datagram Protocol (UDP).

I. rrBox Client

The rrBox Client is responsible to read the generated partial bitstream and send it through the Ethernet to the NetFPGA. Since the size of the partial bitstream is large, the transmission of the partial bitstream to the NetFPGA requires a number of packets. In rrBox Client, the user has the option to choose the size of each bitstream packet transmitted to the NetFPGA. The rrBox Client will resend any unmarked or timed out bitstream packets to ensure the partial bitstream is extracted and stored properly. Upon receiving the acknowledgement in the last segment of partial bitstream, the rrBox will send the bitstream termination packet to the NetFPGA for verification.

V. RESULT & DISCUSSION

Table II lists the logic resources required to enable remote dynamic reconfiguration in the NetFPGA 10G development board. XC5VTX240T FPGA has 37440 slices, in which only 8154 are occupied (21.78% logic utilization). Besides that, the utilization on the BlockRAM of the platform is approximately 23.76%. The full system includes the rrBox Core and the fundamental components in the NetFPGA framework, which are the Input Arbiter, Output Queues and Ethernet Interfaces. Therefore, there are sufficient slices and BlockRAM available for the implementation of the Partial Reconfigurable Module.

Based on the implemented platform, various sizes of partial bitstreams have been generated to obtain the platform performance. The generated partial bitstream sizes are 255KByte, 334KByte, 512KByte and 684KByte. The reconfiguration time for each partial bitstream was recorded and plotted in Figure 5. It shows that reconfiguration time increases linearly with partial bitstream sizes. The reconfiguration throughput is around

Table II
LOGIC UTILIZATION

Logic Resources	rrBox Core	Full System	Available
Number of Slice Registers	9822	16446	149760
Number of Slice LUTs	9705	16441	149760
Number of occupied Slices	5114	8154	37440
Number of BlockRAM	14	77	324
Number of ICAP	1	1	2
Number of BUFG	2	10	32

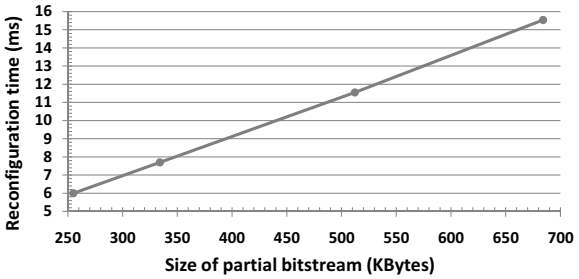

Figure 5. Reconfiguration time for various size of partial bitstream

350Mbps. The reconfiguration time is the total time required for the partial bitstream transmission through the Ethernet and the time required to load the partial bitstream into the configuration memory through the ICAP. In this platform, the reconfiguration frequency is 50MHz and the ICAP bus width is 32 bit. In addition, the platform is operating at 100MHz and the AXI4-Stream bus width is 64 bit.

Table III shows the comparison between this work and other previous works. From the table, the reconfiguration frequency of this work is the lowest among previous works [10], [11], [12], [13], [14] but the achieved reconfiguration throughput is the highest. This is because the reconfiguration controller is implemented using custom logic instead of using General Purpose Processor. Additionally, the 1Gbps Ethernet connection speeds up the transmission of partial bitstreams. Although the reconfigurable device used can have impact on the reconfiguration throughput, the reconfiguration throughput is still very dependent on the design architecture to achieve high efficiency in transmission and the loading of partial bitstream. In fact, achieving high reconfiguration throughput and high reconfiguration efficiency are a major milestone for the practical application of the developed platform.

Based on the functional block diagram of rrBox, the packet forwarding algorithms were implemented as the Partial Reconfigurable Module. Thus, the packet forwarding algorithm can be updated remotely from time to time for optimization and customization purposes. In order to test the functionalities of the implemented

Table III
COMPARISON WITH PREVIOUS WORK

Research work	FPGA family	Bitstream storage or transmission	Reconf. frequency (MHz)	Reconf. bus width (bit)	Size of partial bitstream (KByte)	Reconf. time (ms)	Reconf. throughput (Mbps)
[10]	Virtex 4	DDR-SDRAM	100	32	-	-	226.24
[11]	Virtex 2 Pro	Ethernet	100	8	200	-	80.00
[12]	Virtex 2	SRAM	66	8	290.838	56.8	40.96
[13]	Spartan 3	SRAM	-	8	-	-	35.50
[14]	Spartan 3	SDRAM	65	8	335	166	16.14
Proposed	Virtex 5	Ethernet	50	32	684	15.54	352.12

platform experimentally, packet forwading algorithms such as switch, hub and loopback were loaded into the platform remotely at run-time. The changes on the packet forwarding algorithms can be observed in client computers using the Wireshark packet analyzer.

The targeted practical application for the developed platform is as a network middlebox. With remote dynamic reconfiguration feature proposed in the NetFPGA 10G development board, the implemented network application can be functionally updated. Network protection is one of the network applications that can benefit from the remote functionalities updates as some of the security vulnerabilities and execution requirements are not known during the design time. Network traffic management is another network application that requires remote dynamic reconfiguration. In specific, there is concept drift issue in network traffic and the traffic classifier requires updates to maintain its accuracy.

VI. CONCLUSION

In this paper, a remote dynamically reconfigurable middlebox has been implemented using the NetFPGA 10G development board. The developed middlebox supports remote dynamic reconfiguration for applications updates through the Ethernet connection. Additionally, the developed middlebox provides flexibility to customize and optimize the packet forwarding algorithm. Network applications can be integrated into the platform by modifying the Partial Reconfigurable Modules, which are the Data Packet Handler, Header Processor and Payload Processor. In short, the developed middlebox aims to provide a feature to customize application after the deployment. Lastly, the developed platform achieves 352.12Mbps of reconfiguration throughput. In near future, several case studies in network protection and traffic management will be included to benchmark the developed platform.

VII. ACKNOWLEDGEMENT

This work is supported in part by the Ministry of Science, Technology & Innovation of Malaysia (MOSTI) under ScienceFund Grant 01-01-06-SF1222 (UTM Vote No. 4S095).

REFERENCES

[1] A. Schallenberg, *Dynamic partial self-reconfiguration: Quick modeling, simulation, and synthesis*. Germany: Suedwestdeutscher Verlag fuer Hochschulschriften, 2010.

[2] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable network packet processing on the field programmable port extender (FPX)," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, Feb 2001.

[3] J. W. Lockwood, "An open platform for development of network processing modules in reprogrammable hardware," in *IEC DesignCon'01*, Santa Clara, CA, USA, Jan 2001.

[4] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "NetFPGA: Reusable router architecture for experimental research," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, Aug 2008.

[5] G. Antichi, M. Shahbaz, S. Giordano, and A. Moore, "From 1G to 10G: code reuse in action," in *First Workshop on High Performance and Programmable Networking 2013*, Jun 2013.

[6] D. Yin, D. Unnikrishnan, Y. Liao, L. Gao, and R. Tessier, "Customizing virtual networks with partial FPGA reconfiguration," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, Sep 2010.

[7] M. N. Krifa, B. Ouni, and A. Mtibaa, "Exploring the self reconfiguration of FPGA: Design flow, architecture and performance," *International Journal on Computer Science and Engineering*, vol. 3, no. 4, Apr 2011.

[8] S. Pontarelli, C. Greco, E. Nobile, S. Teofili, and G. Bianchi, "Exploiting dynamic reconfiguration for FPGA based network intrusion detection systems," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, Sep 2010.

[9] K. Zhang, X. Ding, K. Xiong, B. Yu, and S. Dai, "RSS: A reconfigurable security system designed on NetFPGA and Virtex5-LX110T," in *1st European NetFPGA Developers Workshop*, Sep 2010.

[10] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, Apr 2010.

[11] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-fast downloading of partial bitstreams through ethernet," in *Architecture of Computing Systems–ARCS 2009*. Springer Berlin Heidelberg, Mar 2009.

[12] L. Braun, K. Paulsson, H. Kromer, M. Hubner, and J. Becker, "Data path driven waveform-like reconfiguration," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Sep 2008.

[13] E. Cantó, M. López, F. Fons *et al.*, "Self reconfiguration of embedded systems mapped on Spartan-3," in *4th International Workshop on Reconfigurable Communication Centric SoCs (ReCoSoC 2008)*, Jul 2008.

[14] I. Gonzalez, E. Aguayo, and S. Lopez-Buedo, "Self-reconfigurable embedded systems on low-cost FPGAs," *Micro, IEEE*, vol. 27, no. 4, Jul/Aug 2007.