Network Enabled Partial Reconfiguration for Distributed FPGA Edge Acceleration

Alex R. Bucknall*, Shanker Shreejith†, Suhaib A. Fahmy*
*School of Engineering, University of Warwick, Coventry, UK
†Department of Electronic and Electrical Engineering, Trinity College Dublin, Ireland

Abstract—**Partial reconfiguration supports virtualisation of applications on FPGAs, enabling compute to dynamically adapt to workloads in distributed infrastructure and datecenters. While the latter often makes use of the PCIe interface and supporting infrastructure to allocate and load compute kernels via a host CPU, FPGAs are becoming increasingly popular as standalone resources in edge-computing, requiring them to manage accelerators autonomously. This paper presents a platform that supports the managing of accelerator bitstreams over the network interface on a Xilinx Zynq device without intervention by the Arm processor. We compare against traditional vendor provided PR management for both library accelerators and custom accelerators and show that we achieve a 29% decrease in reconfiguration trigger latency using this approach.**

I. Introduction

The use of FPGAs within data centres, has enabled efficient acceleration of operations such as data/feature extraction, dataflow management and information analysis [1], for a range of applications including image recognition [2], in-network security [3] and machine learning [4], among others. However this virtualisation introduces considerable overheads in latency, bandwidth and resource consumption with respect to the volume of data being processed [5]. Non-traditional edge architectures such as distributed reconfigurable heterogeneous system-on-chip (SoC) platforms can be deployed to provide high throughput parallel processing nearer to data sources. The benefits of these SoCs stem from the tight coupling of Arm-based processing systems (PS) for general compute and programmable logic (PL) for custom accelerators, as well as allowing the PS to manage tasks such as networking, while data processing can be offloaded to the PL.

Software managing the networking stack can lead to control prioritisation issues and non-deterministic packet latency, due to the nature of task driven processing seen within typical scheduler systems. Higher priority tasks may override control events and lead the PS to miss critical tasks such as reconfiguration. Hence for connected systems, there is a desire to move functionality into the network interface to enable a more predictable and lower latency response to networking events [6]. In previous work, we demonstrated the benefits of direct processing of network data in the PL on the Xilinx Zynq without the involvement of the PS [7].

In this paper we propose a novel approach to managing partial reconfiguration (PR) via the network interface on the Xilinx Zynq SoC platform to enable low latency network controlled accelerator management. We compare performance between traditional methods of initiating PR to a method of loading control packets for reconfiguration in the PL directly, bypassing the PS Ethernet driver, reducing latency and non-determinism. We explore decoding incoming Ethernet frames within the FPGA, directing them into the PL to process custom frame headers for PR commands.

II. Background and Related Work

In addition to compute parallelisation, FPGA-based acceleration provides the ability to dynamically reconfigure the parts of the hardware for the task at hand. This technique, called partial reconfiguration (PR), enables the FPGA to retain functionality in the static fabric logic, whilst configuring predefined partially reconfigurable regions [8], [9]. PR has several advantages over standard reconfiguration, including use of the static regions whilst reconfiguring, power savings [10], and the ability to retain logical state whilst under reconfiguration.

As FPGA acceleration within data centres has increased in popularity, various strategies have been proposed to simplify both individual and cluster addressing of attached accelerators. The Microsoft SIRC platform presents a PCIe interface API for managing synchronisation of an attached reconfigurable accelerator [11]. Approaches like DyRACT [12] extended this approach by utilising custom PR controllers and drivers to deliver PR bitstreams at high throughput. [13] demonstrated virtualization of FPGA accelerators initiated via the network interface using full configuration. For time critical applications the software networking stack adds additional non deterministic latency to the existing delay from network propagation. With increasing demand to distribute the compute capability closer to the source, the task of reconfiguration must be shifted away from centralised servers in traditional accelerators, requiring the FPGA to manage its own reconfiguration autonomously.

FPGA vendors have developed proprietary reconfiguration managers for in-device reconfiguration. The Xilinx Internal Configuration Access Port (ICAP) is exposed to software through the Processor Configuration Access Port (PCAP). PCAP uses a blocking PS driver to handle reconfiguration, restricting the processor from executing other compute tasks during reconfiguration, while the standalone ICAP interface requires users to design controller infrastructure around it. A variety of high-speed PR controllers such as RT_ICAP [14] provide the advantage of a real time PR manager but do not support a PS interface for hybrid FPGA platforms. ZyCAP

[15] offers near peak throughput for PR, by utilising the PS direct memory access (DMA) controller to pre-load a target bitstream for transfer into the ICAP and can be controlled through software. We extend ZyCAP as the PR manager in our custom architecture. Despite the developments in PR, deterministically triggering PR from the network is non-trivial, especially on SoCs that rely on the PS for network/packet management whilst scheduling other processes. Our approach to enabling network PR, utilises a strategy for remapping the network path from the PS into the PL.

III. Architecture

Traditional Approach: Off-the-shelf FPGA SoC boards integrating the Xilinx Zynq, typically attach the Ethernet PHY directly to the PS-EMAC cores, allowing for complete TCP/IP networking stacks within the OS on the Arm cores. To communicate with the Ethernet controller, a PS driver must be initialised at setup; this allocates a continuous range of n 64-bit memory locations, for the RX/TX buffer queue, mapped to the addresses on the controller.

The driver looks up the location of the RX buffer descriptors, transfers the frame to memory and sets the queue status. The PS Ethernet driver will then alert the application that a frame has arrived via interrupt or polling. The Ethernet Direct Memory Access (DMA) then copies the frame to a buffer and resets the status. Using the Ethernet PS driver requires that the PS is involved in all RX and TX transactions. Traditionally, an arriving PR request packet triggers the PS to begin the decoding/extracting the bitstream information required for PR.

DMA Proxying: Initiating PR from a network packet, requires the PS to decode and extract the PR command from an incoming frame. This may result in PR taking place under non-deterministic latency if the PS is occupied handling high priority tasks. An alternative to this flow control would be to perform frame decoding/encoding in hardware. The architecture presented in this paper utilises a method of indirectly forwarding packets into the PL known as DMA proxying [16]. This maps Ethernet buffer locations, stored within DRAM, to alternative locations such as in the PL BRAM. This enables the DMA to move data in/out of the mapped locations as well as to the PS, if required. While this strategy reduces the latency introduced by the PS waiting to receiving an incoming frame, it blocks the PS while unpacking, processing and decision making. It is possible to implement a complete Ethernet stack into the PL and perform the features of the Ethernet PS Driver within hardware, however this increases design complexity significantly.

Network Partial Reconfiguration: In our proposed architecture, received Ethernet frames are proxied (DMA) into an Ethernet bridge within the PL, as shown in fig. 1. The Ethernet bridge implements the packet handling and decoding logic that is usually performed by a driver on the PS. The bridge is initialised from the PS at system start, by writing into an internal configuration stack; post-initialisation, the bridge can monitor, analyse and redirect incoming packets based on sniffed frame header content and/or data segments, reducing the latency and non-determinism associated with packet reception and decoding within the software stack.

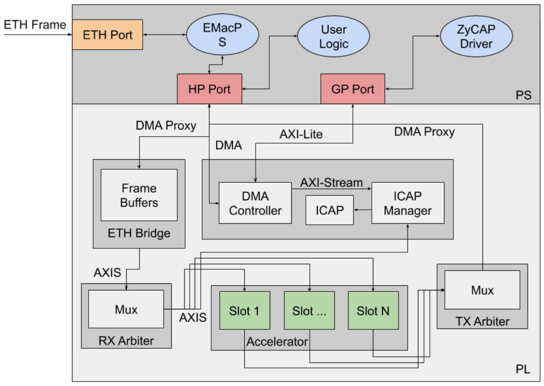

Fig. 1. Network enabled PR architecture.

The configuration stack is a memory mapped bank of registers that can be configured to match OSI layer 2/3 addresses, specific data segment patterns, byte offsets, packet types and etc. Based on the match, the bridge configures the egress path for the packet by configuring the RX arbiter's multiplexer, such as directing it into accelerator slots, the PS, the ZyCAP manager for PR or to be ignored/dropped. The incoming packet is buffered into the ping-pong RX FIFO within the bridge, allowing packets to be received and processed simultaneously. During the packet buffering, inline sniffing logic extracts data from packet headers and compares it against the register stack. A matched packet header, redirects the packet into an accelerator slot or back to the PS DRAM via PL DMA. The entire packet is moved to the address/offset stored in the PS register stack, mimicking a ring-buffer DMA seen in typical PS Ethernet stacks.

When a PR request is decoded, the inline logic extracts the bitstream name and raises an interrupt on the PS. The PS reads the bitstream name from the bridge register and uses the ZyCAP driver to decode, locate and transfer the bitstream to the PL. Upon DMA set up, the driver releases control to the PS, enabling the PR event while PS continues scheduling.

Our framework also supports remote-PR, allowing for a incoming bitstream to be loaded over the network. Such a request is decoded by matching the frame header and remote-PR keyword within the payload. Upon receiving this command, the bridge records the information about the request, such as the number of packets and bitstream size. The bridge configures the RX arbiter to forward the packet to the PR controller, while also initialising the ICAP manager to receive packets from the arbiter instead of the DMA. At each subsequent remote packet, the bridge monitors and updates the frame and byte count; upon completion, the bridge interrupts the PS.

On a match of a data only packet, the bridge instructs the RX arbiter to set the destination to the target accelerator slot. The bridge may also be configured to direct the packet into accelerator slots via the arbiter, which sets multiple write paths in response. The arbiter follows a strict first-in-first-out (FIFO)

TABLE I
NETWORK PR EXPERIMENT RESULTS.

Experiment	RX (ns)	Decode (ns)	PR (ns)	Tot. T'put (MB/s)
PCAP (PS)	53238	297	6303840	122
ZyCAP (PS)	53246	294	1955382	368
ZyCAP (PL)	37605	N/A	1953795	391

system to ensure that incoming packets are correctly ordered and limits stalling by implementing a time-out mechanism that allows a packet to be dropped if the destination is unable to accept it. An interrupt is then raised and the PS can issue a retransmission request, using the stored packet information.

For transmission, the TX arbiter manages output data paths and packs/delivers frames onto the network via reverse proxy.

IV. EXPERIMENTS

To evaluate PR over the network, we simulate delivering Ethernet frames that request a cryptographic scheme to be loaded into an accelerator slot to process a subsequent stream of encrypted data. We implement the case study utilising PRESENT [17] and AES-256 [18] accelerators on the PYNQ-Z1 platform. This mimics an IoT system where a sensor array with limited compute capability and/or power budget for onboard encryption, offloads the computation to an edge accelerator [19]. We compare our architecture against PS decoding using locally cached bitstreams. A single accelerator slot is considered for simplicity; a PR bitstream size of 799,584 bytes was generated for both cores.

An Ethernet frame was crafted, composing of payload segments to denote the packet type, e.g. PR command, data, etc. We compare a PS driven PR command using Xilinx's PCAP PR flow, a PS driven PR command flow using ZyCAP in PL and our custom network PR architecture which implements packet handling within the PL. In the case of a PR command frame, the frame has a specific layer-2 format and a data segment that denotes the frame as a PR command and the name of the bitstream. The generated Ethernet frame is looped back at the PHY for precise event timing. For remote-PR request, the data header specifies the request command, size of the bitstream and a counter denoting the sequence number. Table I highlights results for bitstreams cached in the DRAM.

Frame Decoding in PS (PCAP): This case presents the typical scenario with vendor tools, where the PS handles all aspects of networking, decoding and PR. The PS RX buffers were configured to DMA transfer the received frames into set locations in DRAM and subsequently interrupt the PS when a frame has been received. The interrupt handler decodes the frame and if found to be a PR request, triggers a software task that decodes the requested mode name, looks up its location in DRAM and initialises the PR event via PCAP. We observed the total time taken for the task to complete from the RX interrupt was 6.357 ms with 53.54 µs to trigger PR and 6.304 ms for PR itself. This meets expected PCAP performance of approx. 145 MB/s [20].

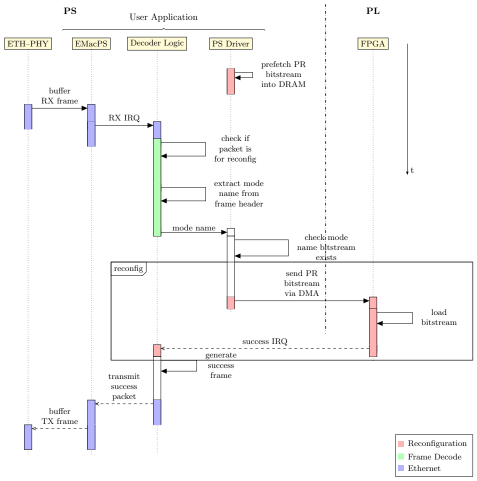

Fig. 2. Sequence of events for PS frame decode and PCAP PR flow.

Frame Decoding in PS (ZyCAP): To evaluate performance gains with custom PR controllers, we integrate the ZyCAP PR manager and driver to manage reconfiguration. In this experiment, the software tasks are still required to decode the frame however once the mode name is known, control is passed to the ZyCAP manager, releasing the PS from the PR task. Fig. 2 shows the arrival of the PR packet at the Ethernet PHY to the completion of PR. The PS is freed from the bitstream transfer and allowed return to event processing; the ZyCAP driver marks the completed task by raising an interrupt, syncing the PS. We observed the total time consumed for PR from the reception of frame in the PS to completion of the PR task reduced to 2.01 ms. This is due to the improved PR speed offered by the custom PR manager, completing PR in 1.955 ms with an overall throughput of 368 MB/s, in line with reported results for ZyCAP [15]. Notably, the driver releases PS control after PR set up; however, the time to trigger PR is still limited at 53.54 µs.

Frame Decoding in PL (ZyCAP): To reduce the overhead of PS frame decoding, we utilise the proposed PR architecture to offload packet decoding to the PL. The frame is redirected into the PL using DMA proxying, decoded inline by the logic and triggers PR, allowing the PS to be bypassed. Fig. 4 captures the events from PR frame arrival at the Ethernet PHY. Decoding the packet within the PL allows the PR driver to receive a PL interrupt, read the requested mode and initiate a PR request via DMA, releasing the PS. This reduces the time to trigger PR to 37.61 µs, performing the PR task in 1.992 ms.

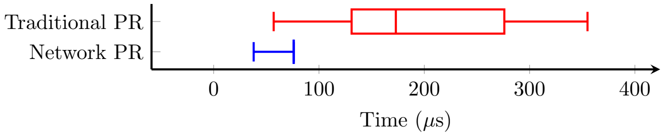

Fig. 3. Variation in reconfiguration triggered over the network interface.

The benefit of offloading packet decoding to the Ethernet bridge can be seen when the PS executes the decoding task while also handling non-preemptive critical tasks. We emulate the case where PS is periodically performing a compute task of high priority during packet decoding. This task reads a register set with variable execution time, while the packet decoding event attempts to decode the frame, perform mode lookup and trigger PR. We observe the variation in latency across a varying number of simulated critical tasks (up to a max. of 10000), performed 25 times and repeated using our proposed PL decoding, shown in fig. 3. The variation between the upper and lower quartiles of the proposed network PR approach is only 1μs compared to the 178μs seen in the vendor flow; a vast improvement over the PS only approach.

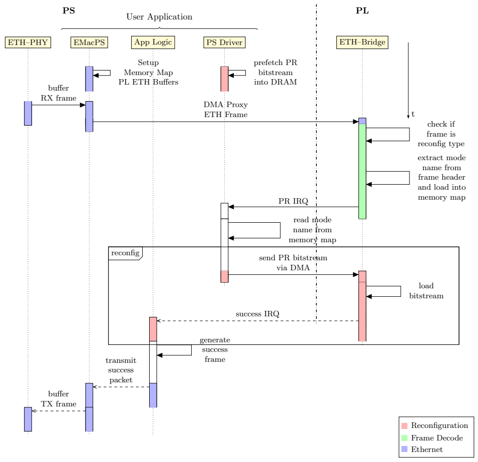

Fig. 4. Sequence of events for PL frame decode and ZyCAP PR flow.

Bitstream Over Network: Bypassing the PS decoding also enables receipt and loading of bitstreams over the network into the PL, without PS oversight. Although cached bitstreams offer fast turnaround, storing all modes locally results in huge memory utilisation and overhead.

To demonstrate, we split the PR bitstream into frames loaded over the network (Ethernet) instead of locally cached. On receiving the PR request, the bridge verifies the frame, extracts the mode and initialises the ZyCAP manager. PR data frames are directed into the PR module until the last frame is received and then loaded into an available slot. We observed the network provisioning at 53.59 ms, measured from the arrival of the PR request at the PL to the completion of PR. We do not compare this to other experiments as PL decoding time is only a fraction of the time taken to transfer the bitstream across the network.

V. Conclusion

We have presented a strategy for managing partial reconfiguration over a network interface, bypassing the PS in FPGA SoCs. This strategy enables high performance, low latency PR for streaming applications that utilise custom PL accelerators such as in-network cryptography. We demonstrate how this approach significantly improves the time to reconfigure, compared to traditional PR methods that require PS processing of Ethernet frames. Our study demonstrates that the PS bypass approach achieves a 29.76% decrease in PR latency over traditional control flows, as well as freeing the PS and reducing time variation for frame handling.

Acknowledgement

This work was supported by the UK Engineering and Physical Sciences Research Council, grant EP/N509796/1.

References

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[2] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Int. Conf. Field Programmable Logic and Applications*, 2009, pp. 32–37.

[3] S. Soliman, M. A. Jaela, A. M. Abotaleb, Y. Hassan, M. A. Abdelghany, A. T. Abdel-Hamid, K. N. Salama, and H. Mostafa, "FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping," *Integration*, vol. 68, pp. 108–121, 2019.

[4] A. Upegui, C. A. Pena-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and microsystems*, vol. 29, no. 5, pp. 211–223, 2005.

[5] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[6] S. Shreejith and S. A. Fahmy, "Smart network interfaces for advanced automotive applications," *IEEE Micro*, vol. 38, no. 2, pp. 72–80, 2018.

[7] S. Shreejith, R. A. Cooke, and S. A. Fahmy, "A smart network interface approach for distributed applications on Xilinx Zynq SoCs," in *Int. Conf. Field Programmable Logic and Applications*, 2018, pp. 186–190.

[8] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 72:1–72:39, 2018.

[9] *UG909: Vivado Design Suite User Guide: Partial Reconfiguration*, Xilinx Inc., Jul. 2019, v2019.1.

[10] A. Nafkha and Y. Louet, "Accurate measurement of power consumption overhead during FPGA dynamic partial reconfiguration," in *Int. Symp. Wireless Communication Systems*, 2016, pp. 586–591.

[11] K. Eguro, "SIRC: An extensible reconfigurable computing communication API," in *Int. Symp. Field-Programmable Custom Computing Machines*, 2010, pp. 135–138.

[12] K. Vipin and S. A. Fahmy, "DyRACT: A partial reconfiguration enabled accelerator and test platform," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2014.

[13] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack," in *Int. Symp. Field-Programmable Custom Computing Machines*, 2014, pp. 109–116.

[14] L. Pezzarossa, M. Schoeberl, and J. Sparsø, "A controller for dynamic partial reconfiguration in FPGA-based real-time systems," in *Int. Symp. Real-Time Distributed Computing (ISORC)*, 2017, pp. 92–100.

[15] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, 2014.

[16] M. Geier, F. Pitzl, and S. Chakraborty, "GigE Vision data acquisition for visual servoing using SG/DMA proxying," in *Symp. Embedded Systems For Real-time Multimedia (ESTIMedia)*, 2016.

[17] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Int. Works. Cryptographic Hardware and Embedded Systems*, 2007, pp. 450–466.

[18] Secworks, "secworks/aes256," Feb 2014. [Online]. Available: https://github.com/secworks/aes

[19] N. Samir, Y. Gamal, A. N. El-Zeiny, O. Mahmoud, A. Shawky, A. Saeed, and H. Mostafa, "Energy-Adaptive Lightweight Hardware Security Module using Partial Dynamic Reconfiguration for Energy Limited Internet of Things Applications," in *Int. Symp. Circuits and Systems (ISCAS)*, 2019.

[20] *UG585: Zynq-7000 SoC All Programmable SoC Technical Reference-Manual*, Xilinx Inc., Jul. 2018, v1.12.2.