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 datacenters. 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
decoding/extracting the bitstream information required for PR. An arriving PR request packet triggers the PS to begin the process. The Ethernet PS 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 notifies the PS to begin the decoding/extracting of the bitstream information required for PR.

**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.
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):** 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. 2. Sequence of events for PS frame decode and PCAP PR flow.](image)

![Fig. 3. Variation in reconfiguration triggered over the network interface.](image)

The benefit of offloading packet decoding to the Ethernet bridge can be seen when the PS executes the decoding task

### Table I: Network PR Experiment Results

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

**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**


