DyRACT: A Partial Reconfiguration Enabled Accelerator and Test Platform

Kizheppatt Vipin, Suhaib A. Fahmy
School of Computer Engineering
Nanyang Technological University, Singapore
{vipin2,sfahmy}@ntu.edu.sg

***Abstract*—Integrating FPGAs with a general purpose computer remains difficult, but recent efforts have resulted in open frameworks that offer a software API and hardware interface to allow easier integration. However, such systems only support static FPGA designs. With the addition of partial reconfiguration (PR) support, such frameworks can enable more effective use of FPGAs. Now, designers can incorporate hardware accelerators within their software applications, and these can be loaded dynamically as required. We present a PR-enabled FPGA platform that allows user modules to be loaded onto the FPGA, inputs to be applied, results obtained, and functions to be swapped at runtime. The interface and PR management logic are part of the static region, while multiple accelerators can be loaded using high level functions provided by the API. Reconfiguration and data transfer are both managed over the PCIe interface from the host PC, with communication throughput of more than 1.5 GB/s (75% of peak PCIe bandwidth) and reconfiguration of a large accelerator in 20 ms.**

I. Introduction

Field Programmable Gate Arrays (FPGA) offer improved performance over software for many applications, while maintaining a higher level of flexibility compared to dedicated hardware. As a result, they have been used in a wide variety of application domains, including bioinformatics [1], radio systems [2], and computer vision [3]. Though the development time for designing FPGA systems is much improved over ASICs, validating FPGA applications on real hardware remains challenging. A key reason is that managing interfaces to the FPGA and the flow of data is cumbersome and typically addressed in an ad-hoc manner, precluding re-use. This difficulty in integrating FPGAs within general computing systems is also one of the key barriers to the use of FPGA accelerators within larger software systems, where the efficiency of this coupling must be maximised.

Recently, researchers have developed open source frameworks to enable easier interfacing between a host PC and FPGA boards [4], [5], [6]. These frameworks offer an API that abstracts the interface, enabling FPGA designs to be accessed efficiently from software on the host. However, these platforms only support static FPGA designs, and in some cases require a system reboot in order to change the FPGA design. Where multiple designs, or alternative variations of a design, need to be tested, this can lengthen test iterations, adding to development effort.

Ideally, users would like to use FPGAs within a host PC for acceleration of compute intensive tasks within the context of a larger software application [7]. In addition to requiring an abstracted interface as provided by the aforementioned frameworks, such a scenario makes sense when multiple different accelerators can be configured as application requirements change during runtime. Existing frameworks require full FPGA reconfiguration to be managed through an external interface like JTAG, requiring additional driver support, and representing a challenge in settings where non-standard connections are not supported. The long time required to reconfigure also means an "on demand" approach to loading accelerators is infeasible.

We propose that both FPGA configuration and data transfer be managed over a single PCIe interface to the host, resulting in high throughput data transfer and fast reconfiguration. This requires partial reconfiguration (PR), with communication infrastructure and reconfiguration management placed in the *static* region, allowing the physical link to be maintained during reconfiguration of the accelerator(s) in the reconfigurable region.

When using an FPGA to host accelerators, it is essential that reconfiguration time be minimised as the reconfiguration overheads can obliterate any hardware acceleration benefit [8]. Again, reconfiguration through JTAG is unsuitable due to its long latency in the order of seconds. Reconfiguration from external non-volatile memory is another option but is severely limited by a storage capacity of just a few bitstreams. The dedicated internal configuration access port (ICAP) on Xilinx FPGAs can enable reconfiguration within a few milliseconds. Hence, enabling reconfiguration over the PCIe interface, by way of the ICAP also opens up the possibility for regular reconfiguration from software, with practically unlimited bitstream storage capacity.

A further benefit of this approach is that FPGA designs that themselves use partial reconfiguration can also be systematically validated, without the need to develop low-level reconfiguration management infrastructure. As PR finds wider use in domains such as software defined radio [9] and audio/video processing [10], providing a simple way to test such systems becomes more important.

In this paper, we present the open source DyRACT framework which offers a Dynamically Reconfigured Accelerator Testbed, interfacing an FPGA board with a host PC over PCIe, using PR to manage reconfiguration. It allows:

1) Loading of a partial bitstreams and passing of data in and out of accelerators.
2) Incorporation of hardware accelerators in software applications with minimal reconfiguration overhead.

3) Testing of partially reconfigurable systems with an abstracted management API.
4) A shorter iteration cycle for testing multiple FPGA hardware modules.

The rest of this paper is organised as follows: Section II discusses related work, Section III presents the DyRACT architecture and the functional descriptions, Section IV presents the implementation details, performance characterisation and a case study, and Section V concludes the paper.

II. Related Work

Numerous approaches to interfacing FPGAs with host PCs have been proposed to support hardware accelerators in software applications. A PCI-X based interface was described in [11] achieving a throughput of up to 667 MB/s. SIRC [4] is interfaces a Windows host PC and an FPGA board over Ethernet but fails to offer the throughput capabilities often required for such software-hardware systems. Recently, frameworks that interface over higher throughput PCI Express (PCIe) links, such as RIFFA [5], [12] and OCPI, have emerged. These frameworks enable static FPGA designs to be accessed through an abstracted software API on the host, including hooks for different programming languages. In previous work, we introduced an open source framework with extended FPGA interface support including PCIe, DRAM and Ethernet [6]. Up to four bitstreams are stored in the on-board non-volatile memory and the FPGA is configured through an external reconfiguration interface. In some cases, these frameworks require a host reboot when the FPGA application is reconfigured, or they rely on PCIe features that can be unreliable to support hot-reconfiguration. Furthermore, loading new user logic requires complete re-implementation of the full design, including fixed communication infrastructure. This can lead to issues with timing closure for large designs, and consumes considerable time. Hence, the use of these frameworks in the context of dynamically reconfigurable accelerators is not possible.

Apart from verifying static FPGA designs, there are few mature approaches for testing partially reconfigurable systems. A functional approach was proposed in [13], however a real hardware test using bitstreams requires considerable effort in implementing the full communication and reconfiguration architecture. PR also has benefits in testing static designs, and has been demonstrated as an effective technique for fault injection to study the effects of radiation and single event upsets (SEUs) [14], [15]. Supporting PR through an abstracted interface can hence offer benefits in the test of both PR and non-PR designs.

Reconfiguration time remains an important aspect in any PR system or framework. If not managed efficiently, reconfiguration can consume a considerable proportion of overall system execution time [16], [17]. While PR is supported through the JTAG interface, internal controllers such as the ICAP offer much better performance, as they provide direct access to the configuration memory from the FPGA fabric. Some have even suggested alternative external interfaces like RS-232, though the achievable throughput is clearly limited [18]. Vendor-provided ICAP controllers offer poor throughput (∼4–10 MB/s) when bitstreams are pre-stored in off-chip memory [19], though the ICAP interface itself is capable of much higher throughput. As a result, a number of custom reconfiguration controllers have been proposed that offer higher throughput (∼400 MB/s) [20], [21]. However, in a number of cases bitstreams are assumed to be in either internal block memories or off-chip memory, without any systematic way of managing bistreams.

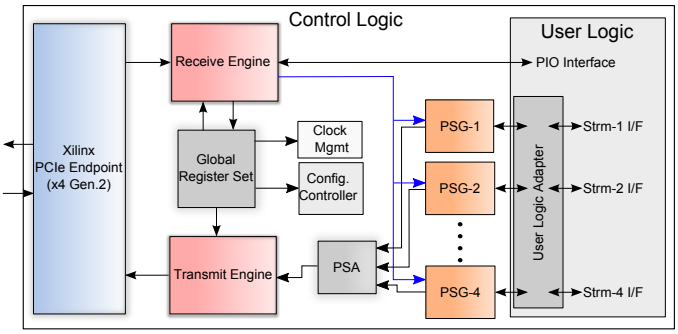

Fig. 1. DyRACT hardware architecture.

In this paper, we propose to load (partial) bitstreams over the PCIe interface that is also used for data communication. This has been demonstrated by Xilinx, but with a low reconfiguration speed of 30MB/s and supporting only a one-time reconfiguration [22]. This approach was improved in [23], enabling multiple reconfigurations, but the reconfiguration speed remained low due to the lack of DMA support.

In this paper, we present the DyRACT framework, incorporating a hardware design and associated software interface for loading designs into an FPGA over a static PCIe interface, then allowing high throughput communication between the host and accelerator. DyRACT supports the use of dynamically reconfigured hardware accelerators within applications running on the host PC. This also enables easier hardware validation of designs including iterative testing. Finally, it is possible to use this framework to test PR systems without the need for building significant infrastructure. Reconfiguration and data throughput are shown to be close the the theoretical maximum supported by the PCIe interface. We are releasing DyRACT to the community to enable easier validation and implementation of FPGA-based designs.

III. Hardware-Software Architecture

DyRACT is comprised of the *control logic* and *user logic*, as shown in Fig. 1. The control logic implements interface management, reconfiguration control, and clock management, while the user logic implements the custom hardware accelerator. The control logic is implemented in the *static* region, while the user logic is implemented in a partially reconfigurable region (PRR), which can be reconfigured at runtime over PCIe. Some of the low level components are adapted from our previous driver [6], though the DRAM and Ethernet interfaces have been removed. The following subsections describe each block in detail.

A. PCIe Endpoint block

We use the Xilinx *PCIe Integrated Endpoint Block*, configured for PCIe Gen 2 ×4 link width as the physical interface.

Theoretically this gives a maximum throughput of 2 GB/s per direction in full-duplex mode, and this is portable across all Virtex-6 and Virtex-7 FPGAs. The backend of the PCIe block is a 64-bit wide AXI4-Stream interface clocked at 250 MHz.

B. PCIe transaction layer

The Endpoint block is directly interfaced with the *receive* and *transmit* engines, which together act as the *PCIe transaction layer*. Here, transaction layer packets (TLPs) are generated and consumed, representing the unit of communication for PCIe. The receive engine decodes received TLPs and routes them to the appropriate target. The transmit engine generates memory read TLPs during DMA operations to fetch data from host memory, memory write TLPs to transmit data from the FPGA to host memory, and completion TLPs in response to read requests from the host.

C. Global Register Set

This module implements all the control and status registers required for interface, communication and configuration management. The *control register* is used to initiate DMA operations between the host and FPGA, as well as to trigger reconfiguration operations. The *status register* is updated after each DMA or reconfiguration operation, allowing the host to ascertain operation completion. Separate address and length registers are used to enable multiple concurrent DMA operations between the host and user stream interfaces.

D. PCIe Stream Generator (PSG)

The PCIe Stream Generators (PSG) act as the DMA controllers between the host and user stream interfaces. Our framework supports a configurable number of PSGs, with each one managing a single user stream interface. The present implementation supports up to 4 concurrent streaming interfaces to user logic.

Since a single read request cannot be larger than 4KB (as per the PCIe protocol), a PSG has to make multiple read requests to the host during DMA write operations (from host to FPGA). When an endpoint device makes multiple outstanding read requests, the completion packets may return out of order. Typically, the endpoint is forced to make a new request only after receiving the data for the previous request. This can severely degrade performance since there can be a large latency between a memory read request and its completion. To achieve full bandwidth during DMA write operations, the FPGA must be able to issue back to back read requests to the host while managing out of order completions.

We exploit the *tag* number field in the PCIe packet headers to implement *virtual channels*, which enable multiple outbound read requests. The tag management is implemented with the help of one FIFO and a unique tag number for each channel. Multiple outbound read requests are generated up to the number of virtual channels. When packets are received in response to the read requests, logic checks the tag number and routes the data to the appropriate FIFO. Later, a *read sequencer* is used to reorder the data by reading sequentially from the FIFOs.

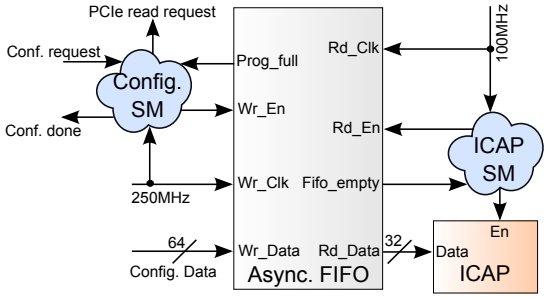

Fig. 2. Configuration controller showing interface connections.

E. PCIe Stream Arbitrator (PSA)

Arbitration logic is required to fairly serve the requests from multiple PSGs accessing the transaction layer. Our design is scalable with the current implementation supporting up to 256 stream interfaces with round-robin arbitration among them.

F. Configuration Controller

The configuration controller is a key feature of this framework. It manages partial reconfiguration of the user logic. It is an improved version of our previous open source PR controller designed to operate in processor-less systems [24]. The controller instantiates the internal configuration access port (ICAP) as shown in Fig. 2. Two independent state machines, the configuration state machine (CSM) and the ICAP state machine (ISM), manage low-level reconfiguration. The configuration operation is triggered by the control register after the host configures the partial bitstream size and its location in the host memory in the global register set. The CSM generates memory read requests to the host to receive the partial bitstream. Received bitstream data is stored in an 8KB asynchronous asymmetric FIFO, with a 64-bit write port clocked at 250 MHz. The CSM generates a new read request only when all data corresponding to the previous request is received and there is sufficient space in the FIFO. Unlike PSGs, the configuration controller does not implement virtual channels since the maximum reconfiguration speed supported by the ICAP is only 400 MB/s so they are not needed.

The ICAP state machine (ISM) constantly monitors the read port of the FIFO for bitstream data. The FIFO read port is 32 bits wide, clocked at 100 MHz – the maximum clock frequency supported by the ICAP. As soon as the FIFO empty signal is de-asserted, the ISM fetches data from the FIFO and writes it to the ICAP. Since the FIFO depth is double the maximum PCIe read request size, the bitstream read from host memory can overlap with ICAP transactions, maximising reconfiguration throughput.

G. Clock Management

One restriction in PR-based designs is that the reconfigurable region cannot contain any clock modifying logic such as mixed mode clock managers (MMCMs). This means the required user logic clock frequency must be provided from the static region. By default, all user stream interfaces run at the PCIe interface clock frequency (250 MHz). But it is possible that some user logic implementations cannot achieve this

TABLE I. FRAMEWORK USER APIS.

API Name	Description
fpga_send_data(channel, data, len, block)	Initialize a DMA transfer between the host and *channel* of array *data* of length *len* *channel*: USERPCIE1..4 *block*: blocking/non-blocking selection when target is USER interface
fpga_recv_data(channel, data, len, block)	Similar to `fpga_send_data()` but to read data from the host
fpga_reconfig(bitstream)	Reconfigure the FPGA with the specified bitstream
fpga_wait_interrupt (channel)	Synchronization function for data transfers.
fpga_reg_wr(addr,data)	Write single 32-bit register
fpga_reg_rd(addr)	Read single 32-bit register
user_set_clk(frequency)	Set the clock frequency to the user logic. (250, 200, 150 and 100 MHz)

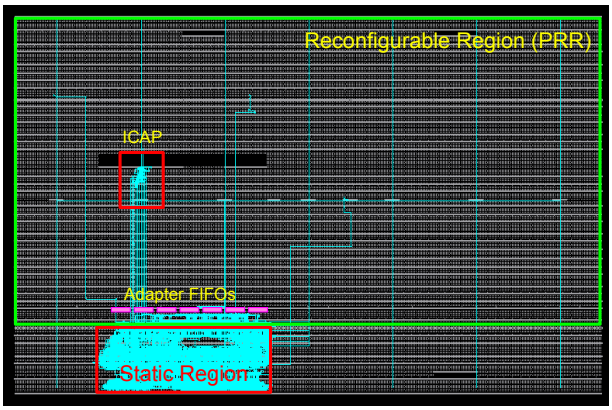

Fig. 3. Virtex-7 floorplan for DyRACT.

frequency. Lowering interface clock frequency compromises throughput for all user logic implementations, so instead, we allow the user to modify interface clock frequency at runtime. This is supported through our software API, which uses the dynamic reconfiguration port (DRP) of the MMCM.

H. User Logic Adapter

One major challenge associated with PR designs is achieving timing closure. Since the the routing of the static design does not change with each different PR bitstream, it is essential that the static logic achieve timing closure even for very large user logic. To preserving the routing between the static and reconfigurable regions, the tools automatically instantiate *proxy logic* on each region boundary (static ↔ reconfigurable) crossing net. Proxy logic is implemented in LUTs that act as pass-through for routing preservation.

Proxy logic can deteriorate timing performance and their placement can exacerbate this problem. In our experiments, we found the Xilinx implementation tools place proxy logic inefficiently, causing large net delay and thus failing to achieve the 250 MHz user interface clock frequency. One possible solution for this is to constrain the locations of all proxy logic close to the interface boundary. Since the user interface contains hundreds of signals, this is not practical.

The user logic adapter instead instantiates AXI4-Stream FIFOs in between each PSG and its corresponding user stream interface. These FIFOs reside in the reconfigurable region and their locations are manually constrained close to the region boundary. This causes the implementation tools to place the proxy logic close to the interface boundary and thus helps with timing closure. This adapter is automatically added to the design by our development infrastructure and users do not have to consider it when designing their user logic.

I. Software Infrastructure

The software component of DyRACT consists of a PCIe driver and a *user library* supported on Linux. The low level PCIe driver is an extensively modified version of the RIFFA driver with the user library providing different communication APIs as listed in Table I. The `fpga_send_data()` and `fpga_recv_data()` functions are used for DMA transfer between the host and user stream interfaces. The specific user stream interface number is provided as an argument to these APIs. The `fpga_reconfig()` function is used to initiate a reconfiguration operation by specifying the partial bitstream corresponding to required user logic.

The send and receive operations can act in both *blocking* and *non-blocking* modes. In blocking mode, the API call returns only after the DMA operation is complete while in non-blocking mode the API returns immediately after initiating a transfer. Non-blocking operations are supported only for data transfers below 4 MB, since data transfers above this size require two buffers and interrupt based buffer management. Non-blocking mode transfer enables overlapping DMA operations to multiple stream interfaces and overlapped read-write operations providing better throughput for small data transfers. Non-blocking transfers must be synchronised with the `fpga_wait_interrupt()` function before starting a new DMA operation to the same channel.

IV. IMPLEMENTATION AND CHARACTERISATION

In this section we discuss the implementation details of DyRACT on multiple target FPGAs. Detailed communication and reconfiguration performance numbers are reported. We also present an example application built using the proposed platform to demonstrate its functionality.

A. Implementation

Xilinx ISE and PlanAhead 14.6 were used for implementation. The proposed platform was implemented and hardware validated on a Xilinx ML605 evaluation board containing a Virtex-6 XC6VLX240T FPGA and on a VC707 evaluation board containing a Virtex-7 XC7VX485T FPGA. The static and reconfigurable regions are area constrained and interface FIFOs are location constrained as shown in Fig. 3. Without the location constraints, the implementation cannot achieve timing closure on the Virtex-6 for region crossing signals, although this is not an issue on the Virtex-7 due to its higher fabric speed.

The resource utilisation, with 4 user stream interfaces enabled, is shown in Table II. On the Virtex-6 FPGA the platform consumes about 6% of both logic and BRAMs. On the Virtex-7, logic consumption is about 3% and BRAM utilisation is about 2.5%. Although some unused resources cannot be used

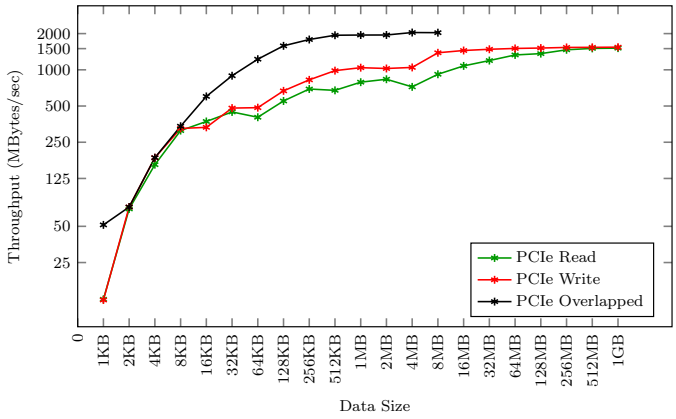

Fig. 4. PCIe communication bandwidth (PCIe Gen 2 ×4 configuration).

by the user logic due to the rectangular shape requirement for the reconfigurable region (as per Xilinx guidelines), about 80% of the FPGA area is available for user logic implementation.

B. DyRACT Development Framework

Xilinx supports PR based system design through PlanAhead [25], which requires the designer to follow specific implementation steps, that non-expert developers find difficult to follow. As part of DyRACT, we provide command-line scripts which automate these PR design steps. With slight modifications, the scripts can support accelerator implementation in multiple reconfigurable regions with independent management of data transfer between each region and the host. The scripts use pre-synthesised netlists and placed and routed control logic for the static region, while automatically incorporating the specific user logic into the design to generate the full and partial bitstreams. This provides several advantages:

- The pre-routed control logic has already achieved timing closure using specific location constraints.
- Designers are not required to undertake manual floorplanning.
- Using pre-routed control logic considerably reduces overall tool execution time.
- Since the specific IP-cores are already routed, issues related to software version differences are avoided.

If designers are interested in additional exploration of the platform, they can completely reimplement the control logic using the corresponding HDL design files and modify the user constraints file (UCF).

TABLE II. DyRACT Resource utilisation

FPGA	**Virtex-6 LX240T**			**Virtex-7 VX485T**		
Module	Regs	LUTs	BRAMs	Regs	LUTs	BRAMs
PCIe Core	791	738	4	1402	923	4
Transaction layer	1058	727	0	1069	613	0
DMA Control	2711	2809	12	2564	2519	12
Config. Control	451	328	2	298	261	2
Clock management	85	84	0	85	73	0
User Adapter	1556	791	8	1556	792	8
Total	6652	5477	26	6974	5181	26

At first initialisation, a full bitstream is stored in on-board flash memory to configure the FPGA at system boot-up time – this contains the full static system and empty user logic. The provided reconfiguration API function (`fpga_reconfig()`) can be used in the user's application C code to reconfigure the FPGA with the specified user partial bitsreams, as generated by the scripts. These calls can be incorporated within a larger software application to load an accelerator, pass data to it, and retrieve the results. Since reconfiguration is managed over the same interface, and accelerators can be loaded without breaking the interface, multiple accelerators can be loaded in succession. The API also enables designers wanting to test a dynamically reconfigurable system to do so without the need for building any PR management circuitry.

C. Characterisation

The host machine for the performance validation was an HPZ420 workstation with an Intel Xeon E5-1650 3.2 GHz CPU, the Intel C600/X79 series chipset, and 16 GB of DRAM, running Ubuntu 12.04 LTS. A stream based user logic design capable of sourcing and sinking an infinite amount of data, is used to determine data throughput. The performance measurements were done with the help of *Performance Application Programming Interface* (PAPI) [26]. Overheads such as DMA controller configuration, interrupt latencies and the interrupt service routines are included in all measurements.

Fig. 4 shows the PCIe communication throughput between the host and FPGA. Write performance peaks at 1542 MB/s and read performance peaks at 1513 MB/s, which is more than 75% of the theoretical maximum PCIe throughput. Further performance improvement is difficult due to packet overheads, host machine limitations and limited packet buffering in the FPGA. It can be seen that for transfers above 4 MB in size, both read and write performance improve due to the double buffering scheme used in the host machine. The benefits of non-blocking data transfer for overlapped read-write operations is also demonstrated. Using this method, it is possible to achieve a throughput of up to 2.1 GB/s for data transfers below 8 MB in size.

We also measured the latency for accessing individual registers implemented in the global register set and in the user logic. A single register write operation takes 133 ns and a single register read takes 1448 ns. Latency for write operation is much lower than that of read operation since the host can post a write operation without waiting for the data to be physically written into the register but is blocked in the read case until the valid data is received.

Meanwhile, reconfiguration over the PCIe interface achieves a throughput of up to 365 MB/s, which is more than 91% of the maximum supported ICAP throughput. For the target Virtex-6, with presented area constraints, the uncompressed partial bitstream size is 7.036 MB, which can be configured in 20.6 ms. For the Virtex-7, the partial bitstream size is 16.85 MB and it can be reconfigured in 46 ms. JTAG based reconfiguration would take about 11 and 21 seconds for the Virtex-6 and Virtex-7 FPGAs respectively. The ML605 also has a 16 MB platform flash which can store a single bitstream with reconfiguration taking about 100 ms. The VC707 has a 128 MB BPI flash which can store up to 4 bitstreams and

TABLE III. PERFORMANCE COMPARISON FOR SOFTWARE, HARDWARE AND HARDWARE-SOFTWARE IMPLEMENTATIONS.

Implementation	Reconfig. Time (ms)	Processing time/frame (ms)	Throughput (frames/sec)
Software	0	1.023	976
Hardware	0	0.153	6510
Software-Hardware	3.698	0.355	2812

reconfiguration takes 130 ms. Storing bitstreams in the flash is a time consuming operation taking up to 20 minutes on ML605 and 30 minutes on the VC707. Hence, none of these other reconfiguration approaches makes sense for dynamically reconfiguring the user logic, especially if this is done for an accelerator within a larger software program, where such latency can nullify any performance advantage. The provided function calls make it possible to incorporate fast reconfiguration seamlessly into a user application. Reconfiguration performance can also be improved by enabling bitstream compression with the amount of compression dependent on the logic in the bitstream. For the case study below, containing 1577 registers and 1464 LUTs, the partial bitstream size was reduced to 1.29 MB using the Xilinx-supported compression method, allowing the accelerator to be reconfigured in under 3.6 ms.

D. Case Study

To demonstrate the effectiveness of DyRACT in the context of a software application with hardware accelerators, an example video processing application was implemented and tested. The application implements several filters: a thresholder, inverter, Gaussian filter, Laplace filter and Sobel edge detector. These were implemented in user logic running at 250MHz with a 64-bit streaming interface. The application processes a continuous stream of 640×480 greyscale video frames with 8 bits per pixel resolution, to produce a continuous stream of output data. As a streaming application, the hardware latency is purely a function of the pipeline. Identical filters are implemented in software (using C) for performance comparison.

Partial bitstreams corresponding to each of these filters are stored in a bitstream library in the host machine. Data is sent and received from the FPGA board using the API functions in non-blocking fashion (streaming). The software application can change from one filter to another, triggering a reconfiguration each time. Table III presents a performance comparison for the inversion filter when implemented as a standalone hardware module, as a pure software implementation, and as a hardware accelerator within software code.

A standalone hardware implementation clearly provides the highest performance. Meanwhile, integrating a hardware accelerator using the proposed framework increases performance by nearly 3× compared to pure software. The reduction in the performance compared to pure hardware is attributed to the communication latency between the host and the FPGA and the overheads associated with DMA and interrupt management. This underlines the importance of the communication between the FPGAs and the host machines when FPGAs are used to implement accelerators in larger software applications. Improved performance is expected when the framework is implemented on newer FPGAs that support higher PCIe throughput. Reconfiguration over PCIe allows a new filter to be reconfigured in under 4 ms (due to bitstream compression), corresponding to the processing time for 10 frames. Hence, as expected, overall performance improves as the processing time increases compared to reconfiguration time. The case study demonstrates a fully functional integration of accelerators reconfigured over PCIe and integrated within a software application.

V. CONCLUSIONS AND FUTURE WORK

We have presented DyRACT, a platform that uses partial reconfiguration to provide a full API for implementing hardware accelerators within software applications, and enables testing of multiple hardware modules in quick succession. The DMA based streaming architecture achieves more than 75% of the theoretical PCIe interface bandwidth, hence allowing high throughput data transfer between host and accelerator. Partial reconfiguration over PCIe reduces reconfiguration time to a few milliseconds compared to several seconds for JTAG. The unified communication and reconfiguration infrastructure avoids the need for proprietary software drivers and dedicated external wiring to the JTAG port. By combining high data throughput and fast reconfiguration, it becomes feasible to implement software applications with hardware accelerators. A video processing cast study was presented, demonstrating the reconfiguration, and data throughput capabilities of DyRACT.

We intend to further improve performance by supporting FPGAs with faster PCIe interfaces. Reconfiguration performance can be further improved by over-clocking the ICAP port, and safe methods are under investigation. We are also working on support for multiple FPGA boards in the same host machine, and abstracted management of reconfigurable regions.

Finally we are publicly releasing this platform and development framework to encourage more people to explore integration of hardware accelerators in software applications, and to ease testing of multiple hardware designs [27].

REFERENCES

[1] M. C. Herbordt, T. VanCourt, Y. Gu, B. Sukhwani, A. Conti, J. Model, and D. DiSabello, "Achieving high performance with FPGA-based computing," *Computer*, vol. 40, no. 3, p. 50, 2007.

[2] J. Lotze, S. A. Fahmy, J. Noguera, L. Doyle, and R. Esser, "An FPGA-based cognitive radio framework," in *Proceedings of IET Irish Signals and Systems Conference*, 2008, pp. 138–143.

[3] S. Jin, J. Cho, X. Dai Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "FPGA design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.

[4] K. Eguro, "SIRC: An Extensible Reconfigurable Computing Communication API," in *Proc IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 135–138.

[5] M. Jacobsen, Y. Freund, and R. Kastner, "RIFFA: A Reusable Integration Framework for FPGA Accelerators," in *Proc. IEEE International Symposium on Field-Programmable Custom Computing Machines*, Apr. 2012, pp. 216–219.

[6] K. Vipin, S. Shreejith, D. Gunasekera, S. Fahmy, and N. Kapre, "System-level FPGA device driver with high-level synthesis support," in *International Conference on Field Programmable Technology (FPT)*, 2013, pp. 128–135.

[7] S. Ludwig, R. Slous, and S. Singh, "Implementing photoshop filters in Virtex," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 1999, pp. 233–242.

[8] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, 2014.

[9] J. Delahaye, J. Palicot, C. Moy, and P. Leray, "Partial reconfiguration of FPGAs for dynamical reconfiguration of a software radio platform," in *Proceedings of IST Mobile and Wireless Comms. Summit*, 2007.

[10] S. Bouchoux, E. Bourennane, and M. Paindavoine, "Implementation of JPEG2000 arithmetic decoder using dynamic reconfiguration of FPGA ," in *International Conference on Image Processing (ICIP)*, 2004.

[11] R. D. Chamberlain, B. Shands, and J. White., "Achieving real data throughput for an FPGA co-processor," in *Proc. Workshop on Building Block Engine Architectures for Computers and Networks*, 2004.

[12] M. Jacobsen and R. Kastner, "RIFFA 2.0: A reusable integration framework for FPGA accelerators," in *Proc. International Conference on Field-Programmable Logic*, Sep. 2013.

[13] L. Gong and O. Diessel, "Resim: A reusable library for RTL simulation of dynamic partial reconfiguration," in *Proceedings of International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[14] L. Antoni, R. Leveugle, and B. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2002, pp. 245–253.

[15] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 965–970, Aug 2007.

[16] K. Bondalapati and V. K. Prasanna, "Dynamic precision management for loop computations on reconfigurable architectures," in *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 1999, pp. 249–258.

[17] J. Tripp, , H. Mortveit, A. Hansson, and M. Gokhale, "Metropolitan road traffic simulation on FPGAs," in *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2005, pp. 117–126.

[18] R. Fong, S. Harper, and P. Athanas, "A versatile framework for FPGA field updates: an application of partial self-reconfiguration," in *Proceedings of International Workshop on Rapid Systems Prototyping*, 2003, pp. 117–123.

[19] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial selfreconfiguration." in *Proceedings of IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2007.

[20] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 498–502.

[21] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller," Microsoft Research, Tech. Rep. MSR-TR-2009- 150, Sept. 2009.

[22] S. Tam and M. Kellermann, "Xapp883: Fast configuration of PCI express technology through partial reconfiguration," Xilinx Inc., Tech. Rep., Nov. 2010.

[23] P. Ostler, M. Wirthlin, and J. Jensen, "FPGA bootstrapping on PCIe using partial reconfiguration," in *Proceedings of International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2011, pp. 380–385.

[24] K. Vipin and S. Fahmy, "A high speed open source controller for FPGA partial reconfiguration," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 61–66.

[25] *PlanAhead User Guide*, Xilinx Inc., Jun. 2013.

[26] Performance application programming interface (PAPI). University of Tennessee. [Online]. Available: http://icl.cs.utk.edu/papi/docs/

[27] DyRACT repository. [Online]. Available: https://github.com/archntu/dyract