A High-Performance FPGA Accelerator Framework for Linear Time-Multiplexed Overlays

***Abstract*—Coarse-grained FPGA overlays have shown promising advantages such as software-like programmability and fast compilation to improve the design productivity. However, few of the existing overlays have been developed as complete accelerator systems, suitable for real-world applications. To address this issue, a high-performance overlay system framework is proposed, based on two different memory interfaces, AXI and PCIe. We implement these hardware accelerators on ZedBoard (AXI-based) and VC707 (PCIe-based) platforms respectively, and evaluate their throughput and resource usage for a range of benchmarks. The proposed AXI-Xillybus-Overlay system achieves a much more area efficient implementation in comparison with a state-of-art time-multiplexed (TM) overlay, referred to as VectorBlox MXP, at the cost of around 50% of the throughput which is limited by the 32-bit AXI-ACP interface used by AXI-Xillybus compared to the full 64-bit AXI-HP interface used by VectorBlox MXP. The proposed RIFFA-Overlay system shows the highest performance among all the proposed overlay accelerators (3.6× higher than PCIe-Xillybus-Overlay, and 5.7× better than AXI-Xillybus-Overlay), but at the expense of a larger resource usage on the BRAMs.**

***Index Terms*—FPGA overlay, memory interface, framework**

I. Introduction

Coarse-grained overlays, implemented on top of the conventional fine-grained FPGA, represent a promising solution to the design productivity problem seen in modern FPGA design. This is because coarse-grained architectures enable faster compilation and software-like programmability, compared to the existing fine-grained FPGAs. FPGA overlays can be broadly categorised as spatially configured (SC) or multiplexed. In an SC overlay, a functional unit (FU) is allocated to a single computational operation of the kernel to be accelerated, with FUs connected by a routing network which is essentially static during kernel execution. A multiplexed overlay, on the other hand, shares both the FUs and the interconnect across kernel operations. A linear time-multiplexed (TM) overlay [6] has shown potential for use as an area efficient FPGA accelerator, and significant improvements to the architecture were described in [7] which further improved the overlay performance. The streaming architecture based on feed-forward pipelined datapaths allows for a simple linear interconnect of lightweight functional units, thus minimizing the interconnect requirements. However, to demonstrate the suitability of the overlay as an FPGA accelerator, it is important to develop a complete accelerator system, with an interface between the processor/memory subsystems and the overlay which is able to provide high-bandwidth (and large scale) data transmission. An examination of existing FPGA solutions for memory interfaces, showed that most fall into one of two categories, AXI bus-based solutions for FPGA SoC systems (like in Xilinx Zynq) and PCIe-based solutions for stand-alone systems connected to a host CPU. Among the various frameworks, some of them abstract the interfaces with simple FIFO connections and provide streaming data transfers, which matches the requirements of our proposed linear TM overlay.

In this paper, we implement accelerator solutions for Zynq-based FPGA systems using the AXI interface and for standalone FPGA systems with PCIe connectivity, based on the Xillybus [15], DyRACT [13], and RIFFA [5] communication frameworks. The microarchitecture of the linear TM overlay and the framework of the proposed overlay accelerator systems are presented in detail. We evaluate the performance of the proposed AXI bus-based and PCIe-based overlay accelerators for a range of benchmarks. The proposed AXI bus-based accelerator is compared to a state-of-art TM overlay referred to as VectorBlox MXP [11], in terms of throughput and resource usage. We compare the PCIe-based implementations, i.e. PCIe-Xillybus-Overlay, DyRACT-Overlay and RIFFA-Overlay, in terms of throughput and area utilization. A detailed description of the programming model for DyRACT-Overlay is presented, which makes it very promising for practical usage.

II. Related Work

A. Coarse-grained TM Overlays

A number of overlays have been proposed which share functional units among kernel operations in an attempt to reduce overlay resource requirements [8], [11], [12]. This time-multiplexing of the overlay means that it can change its behavior on a cycle-by-cycle basis while the compute kernel is executing, thus allowing sharing of the limited FPGA resources.

ADRES [9] appeared to be the first integration of a very long instruction word (VLIW) processor tightly-coupled with a coarse-grained reconfigurable matrix. Some of the functional units (FUs) and register files (RFs) are shared by the VLIW processor and the reconfigurable matrix. Thus, there is no need to transfer the data between the processor and the reconfigurable array compared to the tranditional CGRAs. Taras [12] implemented the ADRES as overlays on Intel and Xilinx FPGAs using CGRA-ME modeling framework [1]. The ADRES overlay can achieve up to $1.6\times$ speedup on Fmax and a reduction of 41% LUT usage, with the FPGA architecture optimizations.

QuickDough [8] was proposed to address FPGA design productivity issues. It is comprised of an array of nearest

neighbor interconnected processing elements (PEs), referred to as SCGRA overlay. Application specific SCGRA overlay were implemented on Zynq, achieving a speedup of up to $9\times$ compared to the same application qunning on the Zynq ARM processor. The 250 MHz FU consists of an ALUm multiport data memory (256×32 bits) and customizable depth instruction ROM (Supporting 72-bit instructions) resulting in significant BRAM utilization. Due to the excessive usage of BRAMs, it has limited scalability as a maximum implementation of 5×5 array can fit on Zynq, despite of the frequency degradation caused by the tight placement and route.

VectorBlox MXP [11] is a commercial soft vector processor overlay targetting Altera or Xilinx FPGAs via the Avalon or AXI interfaces. Data transfer is handled by a double-buffered vector scratchpad and a dedicated DMA engine communicating with the scalar host processor via the AXI HP port. MXP can operate at a maximum frequency of 110 MHz on a Xilinx XC7Z020 device with 16 vector lanes. It demonstrates up to $1000\times$ speedup over MicroBlaze. The programming model is user-friendly as it is a combination of ANSI-C and VectorBlox C extensions. However, it is difficult to implement the applications which cannot be easily vectorized.

B. Memory Interface Solutions

Two of the more common solutions for interfacing FPGA based hardware accelerators are AXI bus-based solutions [10], [14], [15] for FPGA SoC systems (like in Xilinx Zynq) and PCIe-based solutions [2], [5], [13], [15] for stand-alone systems connected to a host CPU.

ZyCAP [14] was developed as an open source soft DMA controller for high performance partial reconfiguration on the Xilinx Zynq. It provides two interfaces, an AXI-Lite interface connected to general purpose (GP) port and another AXI4 interface connected to the high performance (HP) port, and achieves a maximum throughput of 382 MB/s between the external memory and the partial reconfigurable region (PRR).

An infrastructure to evaluate the data processing bandwidth and the energy consumption between the ARM processor subsystem and the DRAM on Zynq using the AXI HP/ACP interface was developed in [10]. The implementation achieved a maximum full-duplex data processing bandwidth of over 1.6 GB/s for a single HP/ACP port, running at 125 MHz on a Zynq XC7Z020-1C device.

Xillybus [15] is a representative commercial solution which support both AXI (baseline) and PCIe (Revision XL) interfaces while providing portability across different FPGA devices. While Xillybus is provided free of charge for research and teaching purposes, users have full access to customize the complete Xillybus project, including the hardware design, the device driver and the software applications, with good documentation provided. The baseline Xillybus has a data rate of $\sim$300 MB/s running at 100 MHz on a Zynq-based ZedBoard, while a maximum data rate of $\sim$3.5 GB/s for Gen2x8 PCIe interface is achieved by Revision XL.

There are a number of academic PCIe-based solutions, such as DyRACT [13] and EPEE [2]. DyRACT mainly focuses on dynamic partial reconfiguration over the PCIe Gen2x4 interface, along with a configuration controller and clock management, while EPEE was developed as a general purpose PCIe communication library, targeting a wide range of FPGA devices. RIFFA [5] is an open source reusable framework for the integration of FPGA accelerators with workstations supporting PCIe Gen 2 and Gen3 protocols. A scatter-gather DMA-based design bridges the vendor-specific PCIe Endpoint core and multiple communication channels for user defined IP cores. The latest release of RIFFA (version 2.2.2) achieves a unidirectional maximum bandwidth of 3.6 GB/s for the Gen2x8 configuration on the Xilinx VC707 platform and 3.5 GB/s for the Gen3x4 configuration on the Terasic DE5-Net.

Though TM overlays have shown great potential to impove the FPGA design productivity, few of them have been developed as full accelerator systems except for the existing work investivated above. The PCIe-based memory soluctions provides a perfect bridge between the host processor and the FPGA-based accelerators, for their high bandwidth and low latency. Among the existing implementations, Xillybus, RIFFA, and DyRACT appear to be the most promising solutions due to their availability, ease-of-use and portability. In subsequent sections we develop high performance overlay-based accelerators designs using these solutions and analyse the overlay capabilities with different memory interfaces.

III. Linear TM Overlay

A. Architecture Description

We have proposed to build a linear array of FUs as a time-multiplexed (TM) overlay [7] where each FU can be time multiplexed among operations present in a single scheduling stage of a directed acyclic graph (DAG). The 32-bit linear TM overlay is comprised of a quasi-streaming data interface made up of two FIFO channels implemented using Block RAMs, which transmit the data through daisy-chained fully pipelined time-multiplexed FUs, as shown in Figure 1. By eliminating the fully flexible routability of the CGRA-like overlays, the linear TM overlay can achieve a very area efficient design, representing less than 6% of the logic and DSP resources on Zynq. The initiation interval (II) can be significantly reduced by making some minor architectural enhancements, such as adding a rotating register file and replicating the data stream. These changes result in a peak throughput of 1.8 GOPS. Adding write-back capabilities to the FU design reduces the overlay depth requirement by allowing multiple nodes on the critical path to be combined. This also eliminates the need to reconfigure the overlay whenever the application kernel changes, making the overlay suitable for more general purpose applications.

B. Overlay Control

A back-pressure control circuit is built around the input FIFO channel to manage the functionality of the proposed overlay, as shown in Figure 2. There are three control signals which indicate the duration for instruction load, overlay setup and data write respectively, referred to as *inst_load*, *reg_wren*,

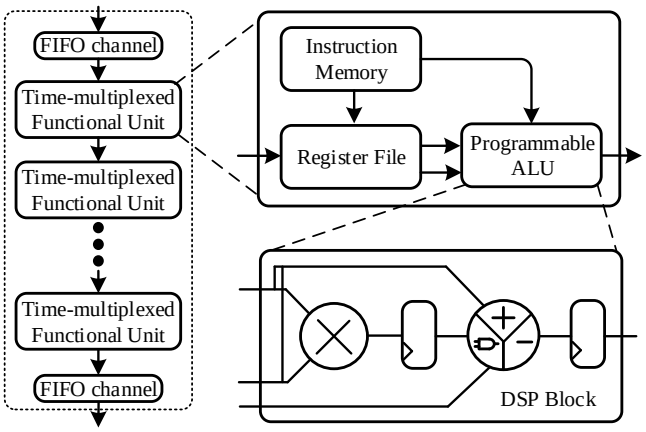


Fig. 1: A linear TM overlay.

and *data_wren*. Initially, FU instructions are read from the memory and streamed through the daisy-chained FUs. During instruction load (when the *inst_load* is high), both the write enable port of the FIFO and the valid signal (*valid_out*) for data output are disabled. After instruction load, two integers are written to the back-pressure control circuit. The first represents the number of data words to be input to the first FU for a specific compute kernel while the other is equal to the II minus one (II-1) and determines the interval between data loads. These values are written into the controller when the *reg_wren* signal is active.

The dashed box on the LHS of Figure 2 acts as the control module for the write enable port of the FIFO, while the other dashed box on the RHS contains the logic to control the read enable port of the FIFO. Data is written into the FIFO when *wr_en* is high. The read enable signal (*rd_en*) for the FIFO is generated from a counter which determines the number of cycles needed to load input data to the FU. The counter starts counting from 0 when the *empty* signal goes low (indicating that data is available in the FIFO). The counter counts up until the count value equals II-1, at which point it rolls over back to zero. The *rd_en* signal is valid only while the counter is less than the initial number loaded into the back-pressure circuit, limiting the amount of data to be loaded into the first FU. Once the counter value is greater than or equal to the data load number (in the back-pressure circuit), the *rd_en* signal is forced low and the input data is buffered in the FIFO.

C. What is Needed for a Full Working Implementation?

The linear TM overlay has shown its advantages to implement an area efficient design with fast context switch and comparable high throughput. However, to demonstrate the suitability of the overlay as an FPGA accelerator, it is important to develop a memory interface between the processor/memory subsystems and the overlay which is able to provide high-bandwidth data transmission. As discussed in the previous section, two stream interfaces are required to send the instructions and the input data from the host side and receive

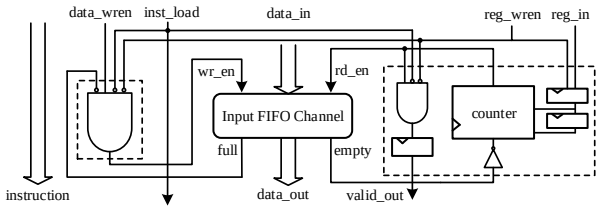


Fig. 2: Back-pressure control circuit.

the output data from the overlay accelerator. Additionally, it is necessary to set up memory arrays or registers so that the user has access to control the overlay, i.e. the instruction load, overlay setup, and data write.

IV. Overlay Accelerator Framework

A block diagram of the proposed overlay accelerator system, based on an array of linear TM overlays [7], which supports both Zynq-based SoC and standalone PCIe accelerators, is shown in Figure 3. The FPGA memory subsystem provides the interface between the overlay on the FPGA fabric and the ARM processor on a Zynq SoC via the AXI bus (or directly to the memory via the PCIe link). For the AXI bus-based overlay, two 32-bit FIFOs are used to connect a single 32-bit linear TM overlay with the memory subsystem. For the PCIe-based overlay, we propose replicating four 32-bit linear TM overlays to make full use of the 128-bit data bandwidth. The four overlay instances can implement multiple compute kernels at runtime and thus reduce the II to a quarter of that of a single overlay. Data transfers between the internal memory subsystem, the DDR SDRAM and the offchip DRAM are under the control of a scatter-gather DMA engine.

A. Xillybus-Overlay

Xillybus is a portable, easy to use DMA-based data transfer solution which provides a simple abstraction of the AXI/PCIe interfaces. One side of the Xillybus IP core is connected to the host processor, while the other side communicates with one standard FIFO, providing six signals to control the data flow. The host processor can be either the ARM processor of the Xilinx Zynq SoC (AXI-Xillybus: for an AXI bus-based solution) or a PC based x86 CPU (PCIe-Xillybus: for a PCIe-based solution). Xillybus provides a simple data loopback demo design, along with its driver for the default device files and the software code for testing purpose. When connecting Xillybus to the linear TM overlay, two standard FIFOs need be used to buffer the input/output data.

B. RIFFA-Overlay

RIFFA has a more complicated framework which generally consists of three layers. On the bottom layer, an RX engine and a TX engine are connected to the vendor-specific PCIe Endpoint IP core, which bridges the connection between RIFFA and the host CPU. The middle layer supports up to

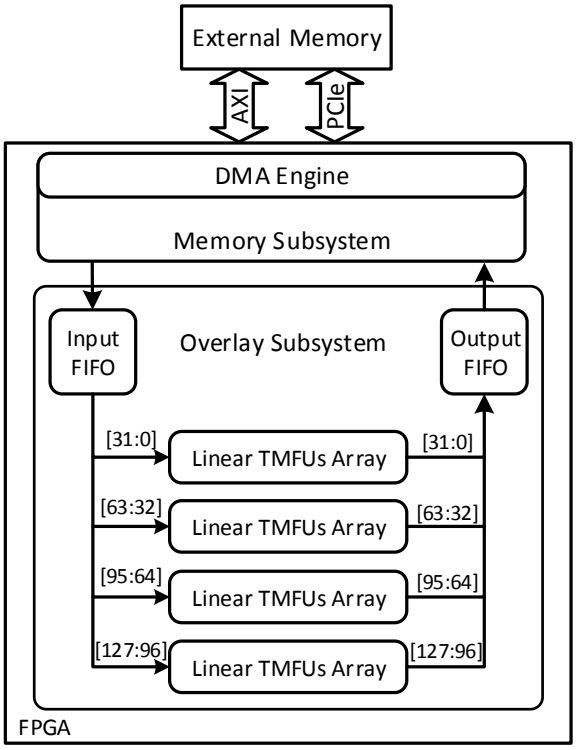


Fig. 3: The proposed overlay accelerator system.

12 channels which send packets to the TX engine and receive packets from the RX engine through a scatter-gather DMA engine. The overlay which connects to the RX/TX FIFOs from the communication channels, is located on the top layer. An example design of one channel for data transfer, along with driver and software code, is provided. Integration of the linear TM overlay with RIFFA is quite similar to that of Xillybus, except that the standard FIFOs are replaced with first word fall through (FWFT) FIFOs to meet the timing closure.

C. DyRACT-Overlay

The original version of DyRACT communication infrastructure enables PCIe-based high-speed communication between a host computer and FPGA-based user logic with support for partial reconfiguration (PR). It provisions multiple AXI4-stream backend interface, enabling seamless integration with vendor-supported IP cores. We have modified the original version to make it lite-weight and tailored to support the proposed overlay architecture. Since the present implementation does not exploit PR, the reconfiguration control logic is disabled, and a single backend AXI4-stream interface is enabled. The stream interface width is configurable through width-conversion FIFOs. A new AXI-Lite interface is added to support command-data interface between the host and the overlay. The software architecture follows similar structure of that of RIFFA. The low-level communication protocols and interrupts are managed by the driver and the user library provides APIs for integration with application programmes.

D. Programming Model

A user-friendly programming model is developed for the overlay accelerator. Once the overlay bitstream has been generated and downloaded into the FPGA, users have access to send the instructions and streaming data from the host PC to the FPGA using dedicated APIs. Take DyRACT-Overlay as an example, there are three register files developed for overlay control purpose. One register file (0x30) is written with a 4-bit tag, which is followed by a register file (0x34) to restore the instruction with its corresponding FU. Another register file (0x38) is used to store the number of data words to be input to the first FU for a specific comptuter kernel (based on the instructions) and the value of (II-1), when the instruction load has been done. Afterwards, a bunch of user-defined data can be transferred to the FPGA, and will be send back to the host PC after the processing of the overlay. A snippet example code shows how to load the instructions, setup the overlay, and write the data to the FPGA can be found in Table I.

TABLE I: Example code.

```
fpga_reg_wr(0x30,0x0); //Tag of FU0
fpga_reg_wr(0x34,0x3033D080); // Instruction 0

fpga_reg_wr(0x30,0x1); //Tag of FU1
fpga_reg_wr(0x34,0x8852000); // Instruction 1

fpga_reg_wr(0x38,5); // No. of input data
fpga_reg_wr(0x38,5); // (II-1)

dyract_send_data((unsigned char *)mydata,
    sendSize*sizeof(int)); //Send data
dyract_recv_data((unsigned char *) recvdata,
    recvSize*sizeof(4)); //Receive data
```

The 32-bit instructions are written in hexadecimal style in Table I, which are used as configurations for the DSP-based FUs. A description of the instruction format, using *ADD R3, R5 (WB)* operation as an example, is shown in Table II. From this table, it can be seen that a 32-bit instruction has four sections, the 18-bit ALU control, the 2-bit input map multiplexing, two 5-bit source operand addresses, with the remaining 2-bit reserved for future use.

V. Experimental Evaluations

We conducted experiments for both the AXI bus-based and PCIe-based overlay accelerators. For the AXI bus-based system, the experimental software runs on the ARM processor of the Xilinx ZedBoard. For the PCIe-based accelerators (implemented using PCIe-Xillybus, RIFFA and DyRACT respectively), the experiments are run on an HP Z420 workstation (six 3.5 GHz Intel Xeon E5-1650 cores) with a Xilinx VC707 evaluation board plugged into the PCIe Gen2 slot of the motherboard.

TABLE II: Instruction format.

FU part	ALU Control								Input Map Control		RF Control		Reserved
Signals	*NDF*	*WB*	alumode	inmode	opmode	cea2	ceb2	usemult	split	immop	src1	src2	
No. of bits	1	1	4	2	7	1	1	1	1	1	5	5	2
Locations	[30]	[29]	[28:25]	[24:23]	[22:16]	15	14	13	12	11	[10:6]	[5:1]	[31][0]
ADD R3, R5 (*WB*)	0	1	0000	00	0110011	1	1	0	1	0	00011	00101	0 0

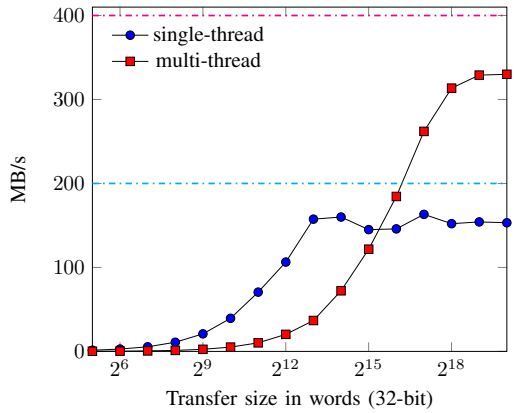


Fig. 4: AXI-Xillybus loopback test evaluation.

TABLE III: DFG characteristics of benchmark set.

No.	Benchmark Name	Characteristics				
		I/O nodes	graph edges	op nodes	graph depth	graph width
1.	chebyshev	1/1	12	7	7	1
2.	mibench	3/1	22	13	6	3
3.	qspline	7/1	50	25	8	6
4.	fft	6/4	24	10	3	4
5.	kmeans	16/1	39	23	5	8
6.	mm	16/1	31	15	4	8
7.	spmv	16/2	30	14	3	8
8.	stencil	15/2	30	14	5	6

A. *AXI-Xillybus System*

The AXI-Xillybus system is using one 32-bit wide AXI ACP port for data transmission, running at a frequency of 100 MHz. Hence the theoretical total bandwidth of this system is 800 MB/s, and the maximum streaming throughput is 400 MB/s in a full-duplex system.

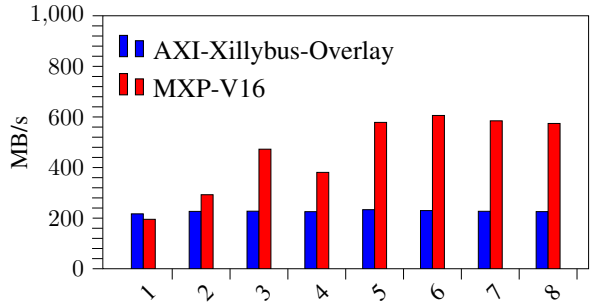
1) AXI-Xillybus Loopback Test: Before the accelerator system can be developed, we first ensure that the demonstration bundle provided by the Xillybus vendor operates correctly on our Zynq-based system. We evaluate the performance of Xillybus by testing the round-trip data transmission throughput using the default settings, using sequential and parallel (multi-threading) programming for *read* and *write* functions respectively.

The bandwidth of the loopback test for AXI-Xillybus using different data block sizes is shown in Figure 4. As seen from this diagram, the single-threaded loopback test saturates at a throughput of around 150 MB/s (maximum of 160 MB/s) when the transfer size exceeds 8K words, while the multi-thread loopback test saturates (with a maximum throughput of 330 MB/s) when the transfer size exceeds 500K words. The two dashed lines indicate the theoretical throughput of the 32-bit AXI-ACP port running at 100 MHz. The throughput of the multi-thread test surpasses that of the single-thread test for data block sizes exceeding 64K words, and for large block sizes achieves approximately twice the throughput. Creating Pthreads introduces a significant overhead for smaller transfer sizes, thus the multi-threading version is only applicable for large data transfers.

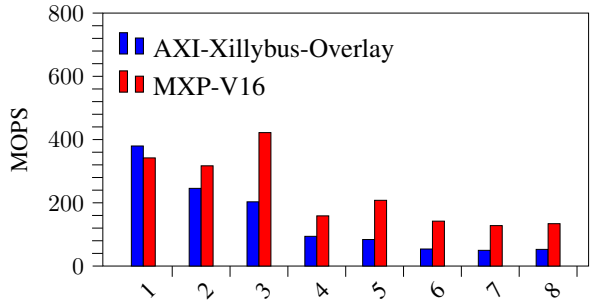
2) AXI-Xillybus based Overlay Accelerator: As previously discussed, Xillybus provides multiple streaming device files and seekable devices files which have access to the memory array. In our design, two 32-bit streaming device files are instantiated for data acquisition and one 32-bit seekable device file is responsible for the instruction load and overlay setup. The performance of the proposed single DSP V3 overlay [7] based accelerator (referred to as AXI-Xillybus-Overlay) is evaluated and compared to VectorBlox MXP [11], in terms of bandwidth and area consumption. To perform this comparison, we use the set of kernels shown in Table III, which are extracted from compute intensive applications available from [3], [4]. Both AXI-Xillybus-Overlay and MXP are running on Xillinux-2.0, which is the latest Linux distribution released for ZedBoard (kernel 4.4), with AXI-Xillybus using the Pthread multi-threading technique with a data block transfer size of 1M words.

Figure 5 shows the performance of a single 32-bit linear TM overlay (AXI-Xillybus-Overlay) and a 16-lane 32-bit MXP (MXP-V16) implemented on a ZedBoard for the benchmarks given in Table III. As seen from the figure, MXP-V16 outperforms AXI-Xillybus-Overlay over all benchmarks (with the exception of 'chebyshev'). Table IV shows the hardware resource usage of AXI-Xillybus-Overlay and MXP-V16. As seen from this table, AXI-Xillybus-Overlay is much more area efficient, and consumes only 19.8% LUTs, 26.1% FFs, 7.6% BRAMs and 7.1% DSP blocks, compared to MXP-V16.

The MXP memory system is based on one 64-bit AXI HP port running at a frequency of 110 MHz while AXI-Xillybus is based on a single 32-bit AXI ACP port running at a frequency of 100 MHz. This corresponds to a theoretical maximum throughput of 880 MB/s for MXP-V16 and 400 MB/s for AXI-Xillybus-Overlay. As seen from Figure 5, MXP-V16 achieves a throughput of 460.6 MB/s (52.3% of theoretical maximum) on average while AXI-Xillybus has a throughput of 226.6 MB/s (56.6% of theoretical maximum) on average. The throughput of AXI-Xillybus is about half of that of MXP-V16,



(a) Data processing throughput in MB/s



(b) Throughput in MOPS

Fig. 5: Performance comparison for the benchmarks using a transfer size of 1M words.

TABLE IV: Area overhead of AXI bus-based systems.

System	Resource Usage			
	LUTs	FFs	BRAMs	DSPs
AXI-Xillybus-Overlay	5,747	5,798	5	8
MXP-V16	28,974	22,174	66	112
Available	**53,200**	**106,400**	**140**	**220**

mainly due to AXI-Xillybus using only 32-bit of the available 64-bit ACP port while MXP-V16 uses the full 64-bit HP port.

Although the AXI bus-based Xillybus provides an area efficient interface solution for the Xilinx Zynq SoC, the throughput is still far below the theoretical maximum shown in Table **??**. This is mainly due to not making good use of the available HP/ACP ports (Xillybus uses only one 32-bit port which is operating at a frequency of 66.7% of the theoretical maximum that the AXI bus on Zynq can support).

B. PCIe System

Our PCIe system is based on the 8-lane Gen2 PCIe interface running at a frequency of 250 MHz, which corresponds to a maximum bandwidth of 8000 MB/s and hence a maximum streaming throughput of 4000 MB/s for a full-duplex system.

1) PCIe-Xillybus Loopback Test: Xillybus has been upgraded to a new version (PCIe-Xillybus) to support high performance PCIe interface communication since 2015. The block diagram of PCIe-Xillybus for data loopback is almost the same as that of AXI-Xillybus, except that the 32-bit FIFO is replaced with a 128-bit FIFO and module *Xillybus.v*

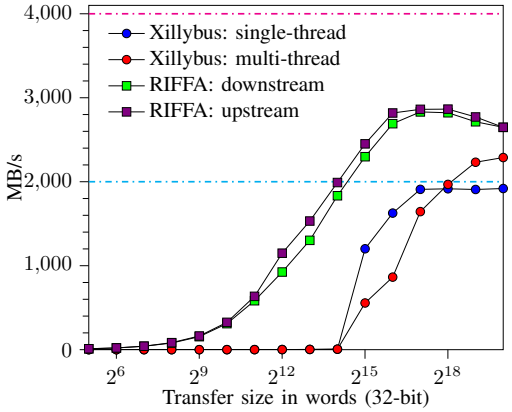


Fig. 6: PCIe system loopback test evaluation.

(including the Xillybus IP core and a Xilinx 7 series integrated block for PCIe) is connected to an HP Z420 workstation via the PCIe link. Similar to AXI-Xillybus, we developed two host programs to evaluate the round-trip bandwidth in both half-duplex and full-duplex mode.

The bandwidth of the loopback test for PCIe-Xillybus using different data block sizes is shown in Figure 6. As seen from this diagram, the data transmission is very slow when the transfer size is less than 32K words. This is due to large system overhead for a data buffer size of less than 128 KBytes [15]. The single-threaded loopback test saturates at a throughput of 1900 MB/s when the transfer size exceeds 128K words, while the multi-thread loopback test saturates (with a maximum throughput of 2300 MB/s) when the transfer size exceeds 500K words. The two dashed lines indicate the theoretical bandwidth of the Gen2x8 PCIe interface running at 250 MHz. The throughput of the multi-thread test surpasses that of the single-thread test for data block sizes exceeding 256K words, and achieves around 1.2× higher bandwidth when transferring 1M words. Although the throughput of the multi-thread test is not as expected (twice that of the single-thread test), it shows better performance for the transmission of large data blocks.

2) RIFFA Loopback Test: Similarly, a loopback design using RIFFA was also implemented. A first word fall through (FWFT) FIFO is used for data loop back, and the RIFFA IP core is acting as the bridge between the vendor-specific PCIe Endpoint and the FWFT FIFO. Data transmission is under the control of a finite state machine (FSM), which executes the upstream transfers and downstream transfers in different processing states. Thus, it is not possible to achieve full-duplex data transmission.

The bandwidth for RIFFA data loopback is shown in Figure 6. As seen from this diagram, both the upstream transfers (read) and downstream transfers (write) saturate at around 2.85 GB/s when the transfer size is equal to 128K sample words, corresponding a total bandwidth of 2.85 GB/s, as read and write cannot happen simultaneously. The bandwidth degrades slightly when the transfer size exceeds 128k words due to the overutilization of BRAMs. As mentioned previously,

TABLE V: Area overhead of PCIe-based systems.

System	Resource Usage			
	LUTs	FFs	BRAMs	DSPs
PCIe-Xillybus-Overlay	16,027	12,488	14.5	32
RIFFA-Overlay	13,657	18,581	289.5	32
DyRACT-Overlay	17,344	16,464	10	32
Available	**303,600**	**607,200**	**1,030**	**2,800**

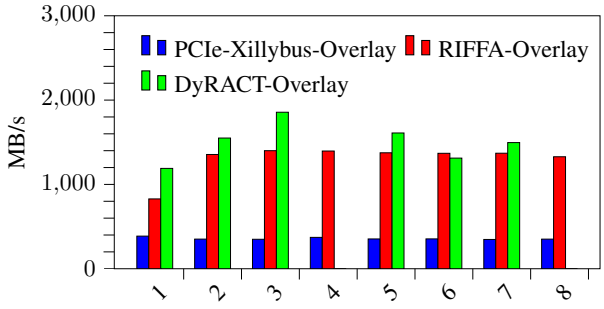
the RIFFA hardware was not developed for full-duplex data transmission. Hence the *fpga_send* function and *fpga_recv* function will execute sequentially and it is useless to use multi-threading techniques as was done for Xillybus.

3) PCIe-Xillybus based Overlay Accelerator: In the PCIe-Xillybus based overlay accelerator system, we propose replicating four 32-bit linear TM overlays, interfacing with the 128-bit FIFOs to make full use of the data width. Thus, two 128-bit streaming device files are instantiated for data acquisition and one 128-bit seekable device file is responsible for instruction load and overlay setup.

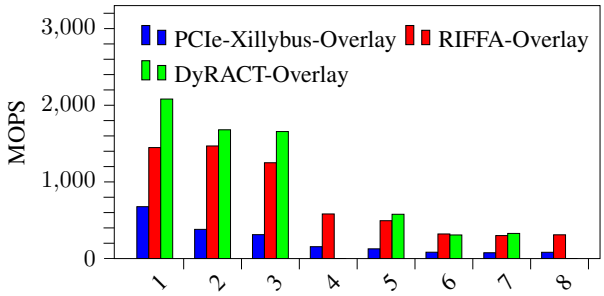
4) RIFFA based Overlay Accelerator: Integration of the linear TM overlay with RIFFA is quite similar to that of Xillybus, except that the standard FIFOs are replaced with FWFT FIFOs. Although RIFFA has no access to the FPGA RAMs directly, it is possible to load the instructions and registers by adding a new state to the FSM control. The RX (view from the integrated overlay side) is divided into two processes: one for overlay initialization, and the other one for data transmission.

Figure 7 shows the performance of four 32-bit linear TM overlays integrated with PCIe-Xillybus (PCIe-Xillybus-Overlay) and RIFFA (RIFFA-Overlay), respectively. AXI-Xillybus uses a block size of 1M words with the Pthread multi-threading technique, while RIFFA uses a block size of 128K words. Both PCIe-Xillybus-Overlay and RIFFA-Overlay are running on an Ubuntu 14.04.5 workstation (six 3.5 GHz Intel Xeon E5-1650 cores) with a Xilinx VC707 evaluation board plugged into the PCIe Gen2 slot of the motherboard. As can be seen from Figure 7, RIFFA-Overlay achieves an average throughput of 1300 MB/s (32.5% of theoretical maximum), which is around 3.6× better than PCIe-Xillybus-Overlay with an average throughput of 358 MB/s (9% of theoretical maximum). The ratio of the throughput in Mega-operations per second (MOPS) is proportional to that of data processing throughput in MB/s.

Table V shows the hardware resource usage of RIFFA-Overlay and PCIe-Xillybus-Overlay. RIFFA-Overlay consumes 15% fewer LUTs, 49% more FFs, 19× more BRAMs and the same DSP blocks compared to PCIe-Xillybus-Overlay. While the logic and DSP resources used are comparable, there is a big difference in the BRAM utilization. This is due to the half-duplex mode of the RIFFA interface, resulting in a significant BRAM consumption to implement the two large depth FWFT FIFOs. Developing a full-duplex RIFFA based system would minimize the BRAM resource usage.



(a) Data processing throughput in MB/s



(b) Throughput in MOPS

Fig. 7: Performance comparison for the benchmarks.

VI. Discussion and Conclusions

In this paper, we have proposed overlay accelerator systems based on two different memory interfaces, AXI and PCIe. The AXI-Xillybus-Overlay represents a very area efficient implementation compared with VectorBlox MXP-V16, but with about half of the throughput. AXI-Xillybus has only half the theoretical bandwidth of MXP, as Xillybus uses just 32-bits of an ACP port while MXP uses the full 64-bits of an HP port. If Xillybus were modified to use a 64-bit ACP port and two parallel V3 overlays were implemented, it would achieve a throughput close to MXP-V16, but with significantly fewer hardware resources (approximately 20% of the LUTS and 8% of the hard macros (BRAM and DSP) required by MXP on Zynq. Thus, the linear TM overlay represents a relatively efficient implementation when FPGA resources are limited, as would be the case when an accelerator was used with other major subsystems in an FPGA based SoC design.

The PCIe-Xillybus-Overlay and RIFFA-Overlay were also proposed for more high-performance centric accelerator systems. The RIFFA-Overlay has a 3.6× speed improvement when compared to the PCIe-Xillybus-Overlay, and a 5.7× better throughput than the AXI-Xillybus-Overlay, achieving a throughput of 1300 MB/s on average. The RIFFA-Overlay appears to be the most promising overlay accelerator for high computing purposes, however, there is a need to develop a full-duplex system, which will not only reduce the BRAM utilization, but will also double the theoretical throughput.

We open source the DyRACT-Overlay framework for future research and it can be downloaded from https://github.com/louislxw/linear_tm_overlay.

ACKNOWLEDGMENT

This work is supported by the Singapore Ministry of Education (MOE) Academic Research Fund Tier 2 under grant MOE2017-T2-1-002.

REFERENCES

[1] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson. CGRA-ME: a unified framework for CGRA modelling and exploration. In *2017 IEEE 28th Int. Conf. Application-specific Syst., Archit. and Processors (ASAP)*, pages 184–189, 2017.

[2] J. Gong, T. Wang, J. Chen, H. Wu, F. Ye, S. Lu, and J. Cong. An efficient and flexible host-FPGA PCIe communication library. In *Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL)*, pages 1–6, 2014.

[3] S. Gopalakrishnan, P. Kalla, M. B. Meredith, and F. Enescu. Finding linear building-blocks for RTL synthesis of polynomial datapaths with fixed-size bit-vectors. In *Int. Conf. On Comput. Aided Design (ICCAD)*, pages 143–148, 2007.

[4] C.-H. Hoy, V. Govindarajuz, T. Nowatzki, R. Nagaraju, Z. Marzecy, P. Agarwal, C. Frericks, R. Cofell, and K. Sankaralingam. Performance evaluation of a DySER FPGA prototype system spanning the compiler, microarchitecture, and hardware implementation. In *Int. Symp. on Performance Analysis of Syst. and Software (ISPASS)*, pages 203–214, 2015.

[5] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner. RIFFA 2.1: A reusable integration framework for FPGA accelerators. *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)*, 8(4):22, 2015.

[6] X. Li, A. Jain, D. Maskell, and S. A. Fahmy. An area-efficient FPGA overlay using DSP block based time-multiplexed functional units. In *Proc. 2nd Int. Workshop on Overlay Archit. for FPGAs (OLAF)*, 2016.

[7] X. Li, A. Jain, D. Maskell, and S. A. Fahmy. A time-multiplexed FPGA overlay with linear interconnect. In *Proc. Conf. Design, Autom. and Test in Europe (DATE)*, pages 1075–1080, 2018.

[8] C. Liu, H.-C. Ng, and H. K.-H. So. QuickDough: a rapid FPGA loop accelerator design framework using soft CGRA overlay. In *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, pages 56–63, 2015.

[9] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *Proc. 13rd Int. Conf. Field Program. Logic Appl. (FPL)*, pages 61–70. 2003.

[10] M. Sadri, C. Weis, N. Wehn, and L. Benini. Energy and performance exploration of accelerator coherency port using Xilinx ZYNQ. In *Proc. 10th Conf. FPGAworld*, page 5, 2013.

[11] A. Severance and G. G. Lemieux. Embedded supercomputing in FPGAs with the VectorBlox MXP matrix processor. In *Proc. Int. Conf. Hardware/Software Codesign and Syst. Synthesis (CODES+ ISSS)*, pages 1–10, 2013.

[12] I. Taras and J. H. Anderson. Impact of FPGA architecture on area and performance of CGRA overlays. In *Proc. 27th Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, pages 87–95, 2019.

[13] K. Vipin and S. A. Fahmy. DyRACT: A partial reconfiguration enabled accelerator and test platform. In *Proc. 24th Int. Conf. Field Program. Logic Appl. (FPL)*, pages 1–7, 2014.

[14] K. Vipin and S. A. Fahmy. ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq. *IEEE Embedded Syst. Letters*, 6(3):41–44, 2014.

[15] Xillybus Ltd. IP core product brief.