2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines

RIFFA: A Reusable Integration Framework for FPGA Accelerators

Matthew Jacobsen, Yoav Freund, Ryan Kastner
Computer Science & Engineering
University of California, San Diego
La Jolla, CA
{*mdjacobs, yfreund, kastner*}*@cs.ucsd.edu*

***Abstract*—We present RIFFA, a reusable integration framework for FPGA accelerators. RIFFA provides communication and synchronization for FPGA accelerated software using a standard interface. Our goal is to expand the use of FPGAs as an acceleration platform by releasing, as open source, a no cost framework that easily integrates software on traditional CPUs with FPGA based IP cores, over PCIe, with minimal custom configuration. RIFFA requires no specialized hardware or fee licensed IP cores. It can be deployed on common Linux workstations with a PCIe bus and has been tested on two different Linux distributions using Xilinx FPGAs.**

***Keywords*-FPGA; CPU; communication; framework;**

I. Introduction

Building FPGA based accelerators requires more skill and often is more difficult than traditional software development but can improve performance by many orders of magnitude. FPGA design is by and large done using hardware design tools similar to ones that would be used to create a custom ASIC. As a result, these tools are ill suited for accelerating software applications. Acceleration frequently requires building considerable components for basic functionality before any application logic can be implemented. Thus, most FPGA applications are designed as standalone systems. Integrated designs require more non-application related framework to be built and would most likely not be reusable.

In contrast, the GPU field has seen substantial development towards accelerating software applications using GPUs. The CUDA and OpenCL languages, and their corresponding tool chains, have enabled near seamless integration into most popular software programming environments. This has made it considerably more attractive to accelerate applications using GPUs despite their rigid computation models, fixed architectures, and generally lower performance [1]. Our goal is to lower the barriers for application acceleration using FPGAs. In doing so, we expect to enable more FPGA accelerated applications and increase the use of FPGAs in traditional software development.

RIFFA is an integration framework for connecting IP cores on an FPGA with software running on a Linux computer. The framework requires a PCIe bus enabled workstation and a FPGA with a PCIe peripheral. RIFFA provides communication and synchronization capabilities with a standard interface for both software and hardware. It is comprised of Verilog and VHDL IP cores, C software libraries, and a Linux device driver, all of which we intend to open source. This represents the main contributions of our work.

This is not the first attempt to integrate FPGAs into traditional software environments. Custom solutions have existed for as long as people have been accelerating applications. However, our goal is a reusable integration framework that is flexible enough to be used in many designs. We would point out that reusable designs also exist.

Vendors such as Impulse Accelerated Technologies and Pico Computing can provide such reusable frameworks. Unfortunately, these solutions are expensive and work only with their platforms which limits wide spread adoption. Microsoft Research's SIRC [2] is an open source solution and has been an inspiration for our own. But while SIRC is free, it requires the Microsoft Windows OS and other proprietary Windows drivers. Furthermore, it uses an Ethernet connection which limits the bandwidth between the host computer and the FPGA. Our approach uses a PCIe bus which offers better performance and is better suited to integrate in supercomputing, desktop application, and other environments where a GPU is used. Intel has developed an integration technology called QuickAssist [3]. Unfortunately, this technology is supported only on a small set of SOC Intel processors. Of course, there are a multitude of FPGA designs that include integrated CPUs. There are also approaches to simplify and allow applications to make better use of IP cores from software running on these integrated CPUs, such as Hthreads [4] and HybridOS [5]. However these solutions utilize custom OS kernels and work only for CPUs running on the FPGA.

The rest of the paper is organized as follows. We describe in detail the RIFFA architecture in Section II. Performance is addressed in Section III. We conclude in Section IV.

II. RIFFA Architecture

RIFFA is a C software library and Linux device driver, on the workstation side, and set of IP cores on the FPGA side. The two are connected via a PCIe bus connection. A diagram of the RIFFA architecture is displayed in Figure 1.

978-0-7695-4699-5 2012

DOI 10.1109/FCCM.2012.44

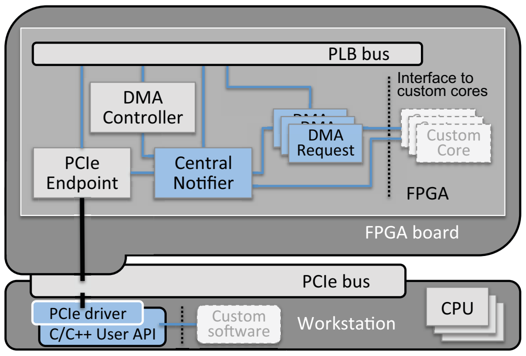

Figure 1. Architecture of RIFFA framework. Application acceleration cores interface with the DMA Request and Central Notifier cores.

In designing RIFFA we sought to expose interfaces general enough for most applications, to support high communication throughput with low latency, and to be compatible with off the shelf workstations and FPGAs. For these reasons, we built our framework to use a PCIe bus. PCIe buses are common in most workstations and increasingly so in embedded systems. They offer high bandwidth connections with extremely low latency. Many FPGA boards come equipped with PCIe connections and chip makers are combining FPGAs with CPUs, connected by PCIe, in the same package or on the same die[1]. We chose Linux because it is an open source platform with wide adoption and well suited for high performance application execution. Our initial version has targeted only Xilinx FPGAs. Future versions may include support for FPGAs from other vendors.

A. Software Interface

On the software end, we developed a PCIe Linux device driver and a set of software libraries. The device driver probes for the FPGA at boot time and assigns addresses within the workstation's PCIe address space for the PCIe Endpoint on the FPGA. During this process kernel address space is reserved for communicating with the FPGA. Once address space is assigned, the driver can access the PCIe Endpoint. In order to enable access outside of the kernel, the driver creates a virtual device file in the *dev* filesystem. This virtual device file can then be opened, read from, written to, or memory mapped by any application in user space. In this way, we expose the FPGA to user space. Accessing this virtual device file executes PCIe read or write transactions over the PCIe bus. On the FPGA, the PCIe Endpoint services these requests by translating them to Processor Local Bus (PLB) requests via address translation. This gives applications on the workstation the ability to access individual IP cores on the FPGA using file operations or memory assignments.

[1]Intel ECx5C Series and Xilinx Zync platforms.

To provide signaling of events, the driver establishes an interrupt channel between the workstation and PCIe Endpoint on the FPGA. Received interrupt vectors identify which IP core has signaled the interrupt. Our driver acknowledges interrupts and exposes individual interrupts by creating a set of numbered virtual files in the $proc$ filesystem. When an application attempts to read or poll[2] any of these files, the driver returns the number of interrupts received from the corresponding IP core. If no interrupts have yet been received, the driver sleeps the calling thread and wakes it up when the appropriate interrupt is received. This design exposes many logically distinct interrupt channels using the single PCIe device interrupt in a thread efficient manner. RIFFA currently supports up to 16 interrupt channels. One drawback to this design is that interrupts must be acknowledged by the driver before another interrupt vector can be sent by the FPGA. We mitigate this problem by AND'ing pending interrupt requests on the FPGA so that a single PCIe interrupt received on the workstation can trigger multiple logical interrupt channels.

Even with a fast processor, we found that writing 32 bits of data at a time via PCIe transactions is inefficient for sending more than a few words of data. Additionally, there is no standard software facility for sending interrupts to PCIe devices from the workstation. We therefore added DMA transfer support and workstation-to-FPGA interrupts (so called "doorbells") using PCIe write transactions. Software initiated writes to a controller IP core signal the request for a DMA transfer and/or an interrupt to a specific IP core. This makes it convenient for applications to have input data DMA transferred to the appropriate IP core then have the core interrupt signaled. These IP core doorbells, manifest as line pulses to the receiving IP cores.

Because utilizing this communication and event signaling framework would require understanding of its implementation, we created a high level API for user applications. This library is written in C. Many of the core functions and their descriptions are listed in Table I. A use case example is shown as a call diagram in Figure 2. This diagram illustrates the relationship between software function calls on the workstation and hardware signaling on the FPGA. A typical user application would initialize the FPGA connection using $fpgaMapMemory$. Data can then be read and written or DMA transferred using the provided functions. For each interrupt it wishes to wait for, the application must call $fpgaInterruptOpen$. Calls to $fpgaInterruptWait$ block the calling thread until an interrupt is received. Because there are 16 separate channels, multiple threads can wait on different channels without interference. No user level synchronization primitives are needed. When no longer needed, the FPGA connection can be closed using $fpgaUnmapMemory$ and interrupt

[2]The *poll* operation is used for asynchronous I/O.

Table I
RIFFA CORE SOFTWARE FUNCTIONS.

Function	Description
fpgaMapMemory	Opens the FPGA virtual device file and maps the PCIe address ranges into memory.
fpgaUnmapMemory	Unmaps all memory mapped address and closes the FPGA virtual device file.
fpgaInterruptOpen	Gets the IP core interrupt file descriptor.
fpgaInterruptClose	Releases the IP core interrupt file descriptor.
fpgaInterruptWait	Causes the calling thread to wait until the corresponding IP core fires an interrupt.
fpgaFireInterrupt	Fires an interrupt to the specified IP core.
fpgaRequestDma	Triggers a DMA transfer between the FPGA and workstation memory.
fpgaReadWord	Reads a 32 bit word from FPGA memory.
fpgaWriteWord	Writes a 32 bit word to FPGA memory.

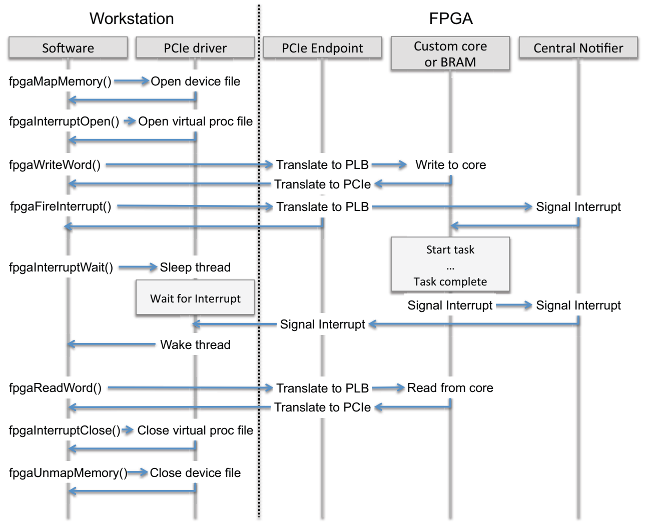

Figure 2. Example usage of RIFFA from a user application.

notifications can be terminated using $fpgaInterruptClose$. All the communication details and kernel structures are hidden from the user application. Additional APIs are being developed to provide support for common application tasks.

One of the drawbacks of this design is that the kernel driver must be configured with an address space large enough to support the maximum amount of contiguous response data expected from any IP core. We currently set this to 8 MB for our example applications, but plan to address this constraint in future versions.

B. Hardware Interface

The key hardware components in RIFFA are the PCIe Endpoint, DMA Controller, Central Notifier, and DMA Request cores as pictured in Figure 1.

The PCIe Endpoint drives the PCIe slot on the FPGA board. It also functions as a PLB to PCIe bridge so that address space can be mapped between the two buses. This IP core is provided free of charge by Xilinx. It is configured with two 4 MB IPIF-to-PCIe base address register (BAR) mappings and one 8 KB PCIe-to-IPIF BAR mapping. IP interface (IPIF) refers to the PLB side of the PCIe bridge. The IPIF-to-PCIe BARs translate IP core accesses using PLB addresses to workstation PCIe memory accesses using workstation PCIe addressing. The PCIe-to-IPIF BARs work in exactly the opposite direction. These BARs support writing to the FPGA IP cores and receiving responses. This core is also configured to send MSI style interrupts to the Linux device driver. Xilinx provides the source for this core with its developer tools (a version of which is free). The core we use in our designs is the Xilinx plbv46_pcie ver. 3.0.0.a.

The DMA Controller is also a Xilinx IP core with similar no-fee licensing. It provides DMA transfer capabilities over the PLB bus. We configure this core to have a FIFO depth of 48 along with a read and write PLB burst size of 16. This core fires interrupts upon completion of DMA transfers. The core we use in our designs is the xps_central_dma ver. 2.0.1.b. Xilinx also provides the source for this core.

The Central Notifier is the heart of the RIFFA framework. It aggregates interrupt requests from IP cores and sends them to the PCIe Endpoint. It also handles DMA and interrupt requests using PLB mapped registers. The register space is partitioned into blocks which support DMA transfers and interrupts for the 16 possible IP core channels. Writes to registers in each block initiate a DMA transfer by setting the source address, destination address, transfer length, and interrupt flag. Upon receiving the length value, the Central Notifier queues a DMA transfer request into a FIFO to be issued to the DMA Controller. This reduces resources by requiring only one DMA Controller and leverages PLB arbitration to avoid request collisions. After the DMA transfer is complete, an interrupt/doorbell may be fired to the appropriate target (workstation or core) if the interrupt flag was set. Interrupts/doorbells may also be initiated by setting the interrupt flag and requesting a transfer length of zero.

The Central Notifier is also responsible for initializing the PCIe Endpoint. This is accomplished by writing to PCIe Endpoint registers over the PLB. The IPIF-to-PCIe BARs cannot be fully configured until the workstation boots and the Linux kernel assigns an address range. This value may change between reboots. Thus after each boot the driver transmits the kernel assigned address to the Central Notifier which then finishes the IPIF-to-PCIe BAR configuration.

The interface to custom IP cores is comprised of doorbell in, interrupt out, and DMA request/acknowledgment signals. The Central Notifier provides the interrupt signals for up to 16 different applications. DMA signaling is handled by the DMA Request core. This core exports a simplified set of signals for requesting DMA transfers and receiving completion information using a simple assert and pulse interface. The combined signals present an interface that simplifies the task of receiving events from software and responding with data. All interface functionality is also accessible via reads/writes from the PLB. Thus on-chip processors or any PLB master can make use of RIFFA as well.

If a core requires parameter data, currently, it must allocate BRAM or memory channel cores on the PLB. Software on the workstation can then write or DMA transfer parameter data to the BRAM using the software interface. We intend to integrate support for temporary parameter data into future versions of RIFFA.

III. Performance

High bandwidth and low latency are among the criteria for this communications framework. We have profiled RIFFA running on Fedora 12 and Ubuntu 10.04 (Linux kernels 2.6.31-32). The FPGA designs were implemented using ISE and XPS 11.5 on a Xilinx ML506 board with a Virtex 5 XC5VSX50T running at 125 MHz. The ML506 board supports a single lane PCIe Gen 1 connection and was connected to a Dell Optiplex 745. The Dell has dual core Intel 2.4 GHz processors and 4 GB of RAM.

Latency times and bandwidths of key operations are listed in Table II. Latencies were measured using cycles counted on the FPGA. The interrupt latency is the time from the FPGA signaling of an interrupt until the Linux device driver receives it. The read latency measures the round trip time of a request from the workstation to BRAM and the returned response (over the PCIe bus). Compared to the alternative freely available framework [2], the round trip latency is 36 times faster. The time to resume a user thread after it has been woken by an interrupt is the only latency that stands out. At 10.4 μs it represents the longest delay and is wholly dependent on the Linux kernel implementation.

The bandwidth measurements are for a single direction transfer between BRAM on the PLB and CPU main memory. We tested using DMA transfer sizes of 8 KB - 256 KB, by a factor of 2. The bandwidth in the direction of the FPGA to the workstation is sustained at 72% of the theoretical maximum for a single lane PCIe Gen 1 channel. The bandwidth is consistent across this range largely due to DMA pipeline depth. However the bandwidth in the opposite direction is comparatively quite poor. We are currently investigating this bottleneck.

Despite the relatively poor performance of the workstation to FPGA bandwidth, we feel the PCIe based connection is superior to the Ethernet connection used in [2]. Newer FPGAs support additional PCIe lanes which will increase bandwidth further. Even with a single lane, the maximum sustained transfer rate is 1.5 times higher than what is possible over Gigabit Ethernet.

We did not test performance between off chip FPGA DRAM and CPU main memory as there are many configuration variables that affect the performance of off chip DRAM that do not affect the PCIe link.

Resource usage for RIFFA is listed in Table III. By far the largest utilization is from the PCIe Endpoint core. This high utilization is specific to the Virtex 5 family of FPGAs. With newer Virtex 6 and Spartan 6 FPGAs, the slice register and slice LUT utilization is considerably lower due to additional PCIe interface hard macros. Virtex 6 FPGAs use only 2505 slice registers and 3763 slice LUTs for the same core. Similarly, the Spartan 6 family uses only 2208 slice registers and 2868 slice LUTs. This is consistent with the trend of increasing PCIe adoption for FPGA connectivity.

Table II
RIFFA key latencies and bandwidths.

Description	Value	
FPGA to PC interrupt time	3 μs	$\pm$ 0.06
PC read from FPGA round trip time	1.8 μs	$\pm$ 0.09
PC thread wake after interrupt time	10.4 μs	$\pm$ 1.16
FPGA to PC bandwidth	181 MB/s	$\pm$ 3.14
PC to FPGA bandwidth	25 MB/s	$\pm$ 1.22
Theoretical max 1x PCIe Gen 1 bandwidth	250 MB/s	

Table III
RIFFA resource utilization.

Core Name	Slice Regs	Slice LUTs	BRAMs	DSP48Es
Central Notifier	1051	1080	2	0
PCIe Endpoint	7899	8741	10	0
DMA Controller	577	782	0	0
DMA Request	245	215	0	0

IV. Conclusion

We presented RIFFA, a reusable integration framework for FPGA acceleration. The framework is a high performance, no cost, open source alternative to expensive, proprietary integration frameworks. It offers a consistent, generally applicable interface to hardware and software with minimal custom configuration. RIFFA can help expand the use of FPGAs for application acceleration by letting designers focus on application logic instead of building communication and synchronization infrastructure. We plan on improving this framework to include more common operations, overcome performance bottlenecks, and response length constraints. The RIFFA source code, example code, and project documentation can be http://cseweb.ucsd.edu/~mdjacobs/.

References

[1] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *SASP*. IEEE, 2008, pp. 101–107.

[2] K. Eguro, "SIRC: An extensible reconfigurable computing communication API," in *FCCM*, R. Sass and R. Tessier, Eds. IEEE Computer Society, 2010, pp. 135–138.

[3] http://www.intel.com/technology/platforms/quickassist.

[4] W. Peck, E. K. Anderson, J. Agron, J. Stevens, F. Baijot, and D. L. Andrews, "Hthreads: A computational model for reconfigurable devices," in *FPL*. IEEE, 2006, pp. 1–4.

[5] J. H. Kelm and S. S. Lumetta, "Hybridos: runtime support for reconfigurable accelerators," in *FPGA*. New York, NY, USA: ACM, 2008, pp. 212–221.