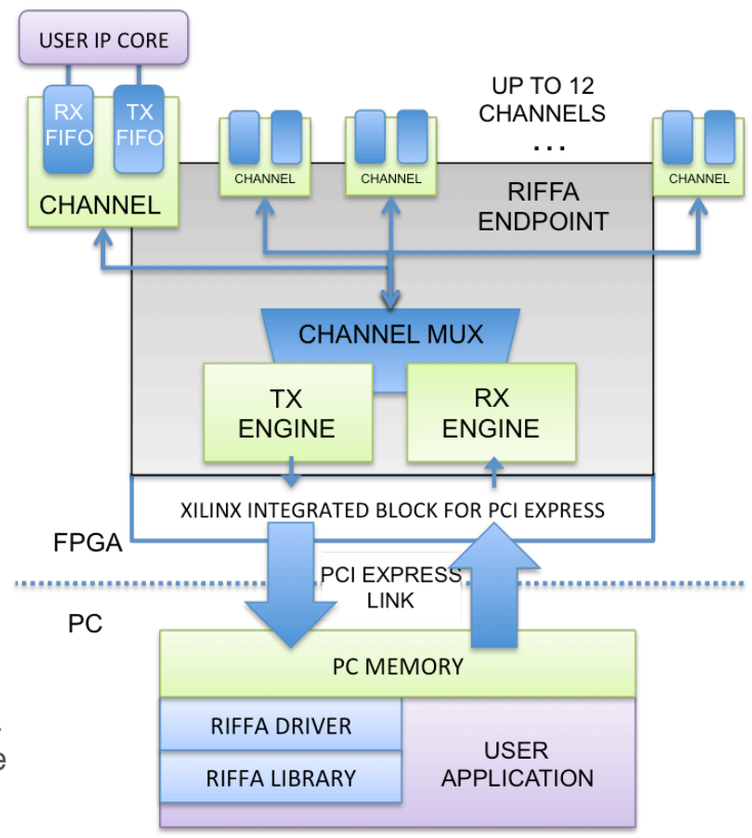
RIFFA: A Reusable Integration Framework For FPGA Accelerators

Architecture

RIFFA 2 is a complete rewrite of the RIFFA 1.0 framework. While both serve the same purpose, their implementations are quite different. A high level architectural diagram of the RIFFA 2 framework is illustrated to the right.

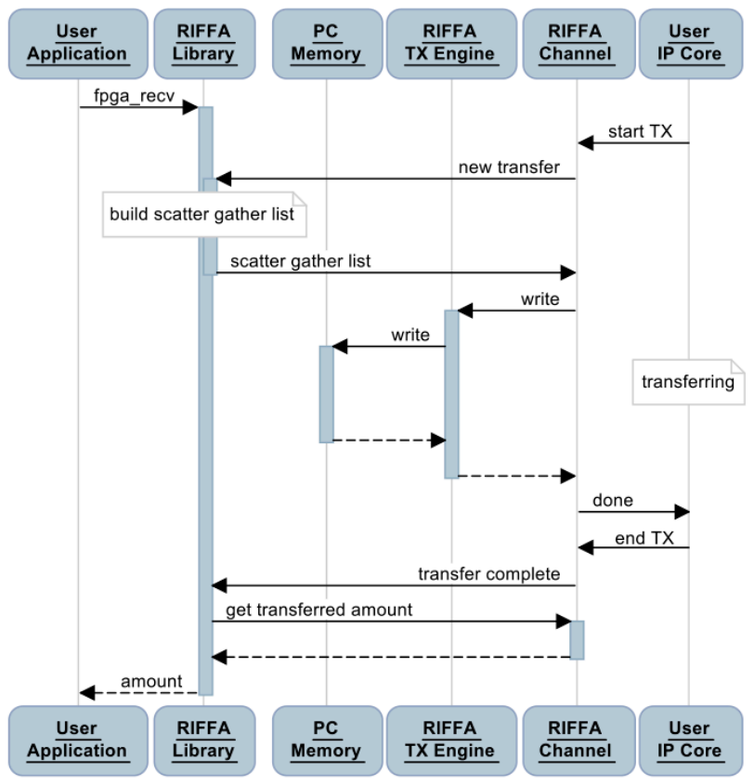


On the hardware side, the interface has been simplified to expose data as a first word fall through FIFO (valid-data-ready interface). The data is transferred by RIFFA's RX and TX DMA engines using scatter gather address information from the workstation. These engines issue and service PCIe packets to and from the PCIe Endpoint. The RIFFA interface supports 32-bit, 64-bit and 128-bit widths, depending on the PCIe link configuration. RIFFA supports all three configurations. Support is planned for the 256-bit interface for PCIe Gen3 endpoints.

RIFFA 2 relies on a Vendor PCIe Endpoint core to drive the transceivers. These are lowest-level interface that FPGA vendors provide. We have tested with the following Xilinx and Altera Endpoint cores:

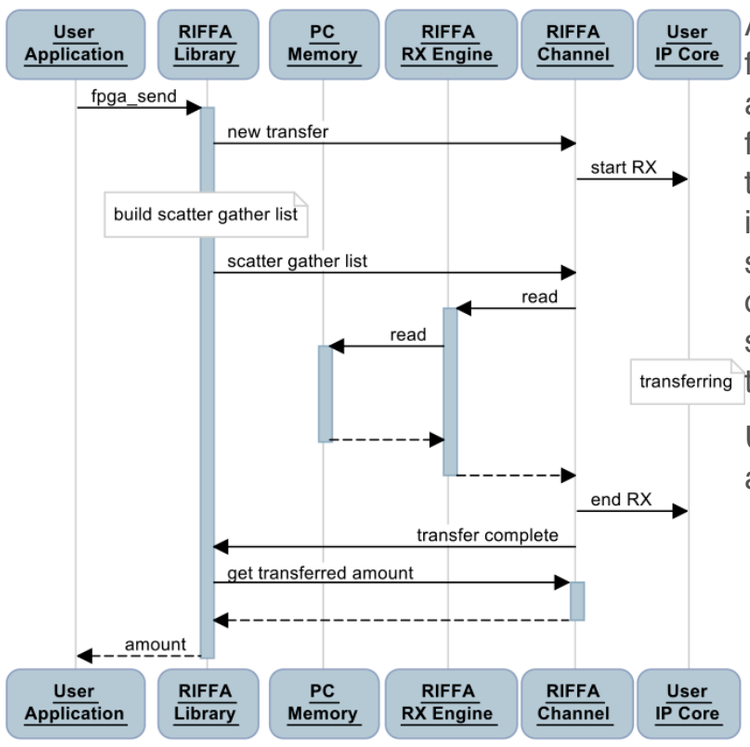
- Xilinx Spartan 6 Integrated Block for PCI Express ver. 2.4
- Xilinx Virtex 6 Integrated Block for PCI Express ver. 2.5
- Xilinx 7 Series Integrated Block for PCI Express vers. 1.6, 1.8, 2.1
- Altera IP Compiler for PCI Express (Stratix IV, Cyclone IV)
- Altera HardIP For PCI Express (Stratix V)

A sequence diagram for an upstream transfer is shown to the right An upstream transfer is initiated by the FPGA. However, they will not begin until the user application calls the user library function fpga_recv. Upon doing so, the thread enters the kernel driver and begins the pending upstream request. If the upstream request has not yet been received, the thread waits for it to arrive (bounded by the timeout parameter). On the diagram, the user library and device driver are represented by the single node labeled "RIFFA Library".



Servicing the request involves building a list of scatter gather elements which identify which pages of physical memory correspond to the receptacle byte array. The scatter gather elements are written to a shared buffer. This buffer location and content length are provided to the FPGA. Each page enumerated by the scatter gather list is pinned to memory to avoid costly paging. The FPGA reads the scatter gather data then issues write requests to memory for the upstream data. If more scatter gather elements are needed, the FPGA will request additional elements via interrupt. Otherwise, the kernel driver waits until all the data is written. The FPGA provides this notification, again via an interrupt.

After the upstream transaction is complete, the driver reads the FPGA for a final count of data words written. This is necessary as the scatter gather elements only provide an upper bound on the amount of data that is to be written. This completes the transfer and the function call returns to the application with the final count.



A similar sequence exists for downstream transfers. The left figure illustrates this sequence. In this direction, the application initiates the transfer by calling the library function fpga_send The thread enters the kernel driver and writes to the FPGA to initiate the transfer. Again, a scatter gather list is compiled, pages are pinned, and the FPGA reads the scatter gather elements. Each of the elements results in one or more read requests by the FPGA. The read requests are serviced and the kernel driver is notified only when more scatter gather elements are needed or when the transfer has completed.

Upon completion, the driver reads the final count read by the FPGA. In error free operation, this value should always be the length of all the scatter gather elements. The final count is returned to the application.