FPGA Augmented ASICs: The Time Has Come

David Riddoch
Steve Pope
Hardware acceleration is Niche

• (With the obvious exception of graphics for gaming!)

• Even for people with a direct financial incentive to go faster…

• The reason?

  It is enormously hard work!

• I’m going to talk about:
  – Why it is so hard
  – How to make it easier
  – Using online trading applications as an example
  – Industry perspective
• Traders are in a latency race

  Signal ➔ Decision ➔ Action

• Whoever responds to signals fastest, makes money

• These folks have money to spend on technology, and exceptional engineers in-house

• But it is not a case of performance-at-any-cost. Like everyone else, they must balance performance against:
  – Flexibility / speed of deployment
  – Available skills
  – Cost
  – Compatibility
…of course there is lots more that we don’t have time to go into…

But all participants have a financial incentive to reduce latency, and many also have a throughput challenge.
How does the NIC help?

- Low latency cut-through design
- 1024 VNICS per port
- VNIC == Virtual NIC
  - Independent interface for sending and receiving packets
- Flow steering
  - Direct individual flows to specific VNICS
  - Supports scaling and NUMA locality
Kernel networking

- Traditionally the network stack executes in the OS kernel
- Received packets are processed in response to interrupts
- Applications invoke the network via the BSD sockets interface by making system calls
Kernel bypass
Kernel bypass – OpenOnload

- Dedicate a VNIC per application or thread
- TCP/UDP stack as user-level library
- Critical path entirely at user-level
- Reduces per-message CPU time
  - Cuts latency in half
  - Increases message rate by 5x per core
  - Improves scaling
- Fully compatible – no changes to applications needed
Back to the hardware…
What is so hard about FPGA acceleration?

- Let’s assume you want some custom logic
  - Evaluate the available board options
  - (Expensive $\Rightarrow$ low volume $\Rightarrow$ expensive $\Rightarrow$ low volume…)

- FPGA image:
  - Development tools
  - You’ll need some IP blocks:
    - PCIe engine
    - Media access controller (MAC)
    - Memory controller
  - Boilerplate
    - Packet handling: Parsing, demultiplexing, buffering, streaming
    - Protocol handling: Checksums, headers, address resolution, TCP, UDP
    - Managing physical links: Configuration, errors, statistics, flow control

- Host software:
  - Device drivers
  - Control path
  - Fast interface to application (kernel bypass)
What is so hard about FPGA acceleration?

• Let’s assume you want some custom logic
  – Evaluate the available board options
  – (Expensive ➔ low volume ➔ expensive ➔ low volume…)

• FPGA image:
  – Development tools
  – You’ll need some IP blocks:
    • PCIe engine
    • Media access controller (MAC)
    • Memory controller
  – Boilerplate
    • Packet handling: Parsing, demultiplexing, buffering, streaming
    • Protocol handling: Checksums, headers, address resolution, TCP, UDP
    • Managing physical links: Configuration, errors, statistics, flow control

• Host software:
  – Device drivers
  – Control path
  – Fast interface to application (kernel bypass)

We haven’t written a single line of application logic yet!!!
So what is wrong with existing offerings?

- Far too much work needed to create a deployable application
  - Apart from cost and time, the FPGA dev skills just aren’t available
    ➔ Need to provide the boilerplate (at the very least)
    ➔ Need a much simpler host interface

- Hard to deploy incrementally
  - Requires simultaneous changes to multiple components
  - Accelerator network interface can only be used for accelerated traffic
  - Consumes an extra PCIe slot
    ➔ Needs to integrate with existing apps

- Expensive
  - Requires huge benefit to justify investment
    ➔ Need a solution that is widely useful (so we can make lots of them)
    ➔ Need off-the-shelf applications to sell in volume
Solarflare’s Application Onload Engine

• Not an FPGA with an Ethernet interface:

• A full-featured Ethernet adapter with FPGA accelerator:
Solarflare’s Application Onload Engine

- Out of the box it works like a regular Solarflare network adapter
  - Drivers included
  - Works with kernel network stack and kernel bypass (OpenOnload)

- Incremental upgrade
  - Pass-thru by default
  - Accelerate a subset of traffic
  - No new switches, cabling, slot

- Solarflare & 3rd party applications
  - Solve common problems
  - No FPGA expertise required

- FDK (developer kit)
  - Reusable IP blocks to minimise effort for FPGA developers
AOE: Block diagram
Host software interface

• The data path is just *packets*
  – Applications on the host use BSD sockets
    • Via the kernel stack
    • Or via kernel bypass for higher performance

• We also provide a register bus
  – Mastered via a software API or command line tool
  – FPGA applications expose registers and memory
  – Notifications
What are the negatives?

- Compared to host-attached-FPGA boards:
  - AOE has higher latency between FPGA and host
  - AOE has no direct access to host from FPGA

- Harder for FPGA apps to access host memory
  - Software on critical path
  - Much higher latency

- FPGA can’t master other devices
Alternative architectures?

- Bearing in mind we didn’t want to change the ASIC

- PCIe attached FPGA

  - Pros:
    - Fast access to host from FPGA
    - Better latency for pass-thru

  - Cons
    - Increased complexity
      - FPGA interacts with NIC ASIC via descriptor rings
      - Need new interface between FPGA and host
    - PCIe core in FPGA
      - (Less space for other things)
Alternative architectures?

- Bearing in mind we didn’t want to change the ASIC
- PCIe attached FPGA

**Pros:**
- Fast access to host from FPGA
- Better latency for pass-thru

**Cons**
- Increased complexity
  - FPGA interacts with NIC ASIC via descriptor rings
  - Need new interface between FPGA and host
- PCIe core in FPGA
  - (Less space for other things)
- Significantly worse latency between host and wire
Bearing in mind we didn’t want to change the ASIC

Add PCIe interface to FPGA

Pros:
  - Fast access to host from FPGA

Cons
  - Increased complexity
    • Need new interface between FPGA and host
  - PCIe core in FPGA
    • (Less space for other things)
  - Slightly worse latency through NIC
Alternative architectures?

• And if we could change the ASIC?

• Add fast bus for host access

• Pros:
  – Fast access to host from FPGA

• Cons
  – Increased complexity
  – New interface between FPGA and host
    • But at least we’re backwards compatible, so optional
Example 1: Dual-line arbitration

- Market data is published as a pair of redundant feeds
- Traders often subscribe to both
  - For reliability
  - To get lowest latency
- Line arbitration converts the pair of streams into a single feed
Example 1: Dual-line arbitration
Example 1: Dual-line arbitration

- Line arbitration in the FPGA accelerator
  - Application sees a single stream

- Application gets the benefits of dual-line arbitration with half the data rate

- More likely to keep up
  - Reduces queuing delays
  - Reduces likelihood of unrecoverable loss due to buffer overflow

- No changes to software!
Example 2: Symbol splitting

- Market data packets contain messages for multiple securities (symbols)
- Single or few packet streams
- Distributing load is a problem
  - May only be interested in a subset of symbols
  - Or may want to distribute load over multiple processes or threads
  - Must process messages in order (per symbol)
- Demultiplex in software is inefficient
Example 2: Symbol splitting
Example 2: Symbol splitting

- Split market data stream into per-symbol streams
- NIC distributes streams across processes, threads, cores
  - More efficient because we’ve eliminated thread/cache interactions
- Much higher throughput possible
- Lower latency
- Discard symbols we don’t care about
  - Reduce throughput and queuing delays
Next we show how the symbol splitter is implemented in the FPGA

Using reusable components

Connected by a streaming packet bus
- Based on Altera’s Avalon-ST streaming interface
- Carries packets and/or messages
- Meta-data words are interleaved within packets

Components connected by packet bus may:
- Inspect packets and add meta-data
- Mutate packet data and meta-data
- Pass-thru meta-data they don’t recognise
- Take actions based on meta-data
  - Manipulate state (lookup-tables, databases)
  - Routing decisions
- Buffering (FIFOs, off-chip memory)
Symbol splitter implementation

Diagram showing the flow of data through various components such as MAC, IP PAR, LU, MSG PAR, MSG SPL, and LU, with additional components like DEM, MSG STI, and ARB.
Single meta-data word a start of each packet
Parse headers, add meta-data
Lookup steam, add meta-data
Parse eligible packets into messages
Pass-thru other packets

- Errors during parsing also lead to pass-thru
Split packets at message boundaries

- Original headers are discarded at this point
• Assign integer ID for each output stream
Symbol stream-ID selects FIFO
Arbitrate amongst symbol streams

Arbiter

Selects input based on packet meta-data and/or fill-level.

Can be custom logic.

Trade-off between packet rate and latency
Stitch messages back into packets

- For minimum latency at low rates: One message per packet
- Packet rate limit per stream forces multiple messages per packet
...and deliver to host
Custom apps: How much work?

- FPGA image
  (some standard blocks)
- Device drivers
- App/FPGA interface
- App integration
Custom apps: How much work?
Custom apps: How much work?

- FPGA business logic
- App integration
• Solarflare’s Application Onload Engine
  – Practical acceleration for network applications
  – Much less work to offload custom business logic
  – Supports incremental deployment
  – Shipping now
  – First deployments being used for enterprise messaging and market data