Low-Level RF to Controls Interface: Future Perspectives

(or, What's the difference between hardware and software?)

Lawrence R. Doolittle, LBNL, Berkeley, CA 94720,

recycle.lbl.gov

LLRF 2005, Geneva, Switzerland

This work is supported by the Director, Office of Science, Office of Basic Energy Sc
the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The SN
is being carried out by a collaboration of six US Laboratories: Argonne National
tory (ANL), Brookhaven National Laboratory (BNL), Thomas Jefferson National Ac
Facility (TJNAF), Los Alamos National Laboratory (LANL), E. O. Lawrence Ber
tional Laboratory (LBNL), and Oak Ridge National Laboratory (ORNL). SNS is ma
UT-Battelle, LLC, under contract DE-AC05-00OR22725 for the U.S. Department of

Old joke:

What's the difference between hardware and software?

[1] I attribute it to Rick Cochran in about 1982, but it may be older than tha

Old joke.

What's the difference between hardware and software?

- Hardware keeps getting cheaper, faster, and smaller!

[1] I attribute it to Rick Cochran in about 1982, but it may be older than tha

Old joke:

What's the difference between hardware and software?

- Hardware keeps getting cheaper, faster, and smaller!

Outline:

• Philosophical differences between hardware and software (What *is* the d between hardware and software?)

• How the FPGA fundamentally alters that landscape

• How boundaries are drawn between and within hardware and software

• Where technology is leading those boundaries

• How these ideas apply to accelerator design

• Practical choices for LLRF to controls interface hardware

• Communicating both data and meta-data to the world

[1] I attribute it to Rick Cochran in about 1982, but it may be older than tha

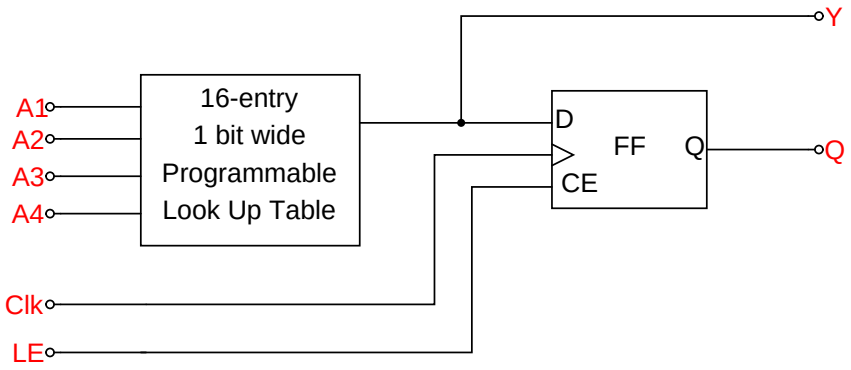
What's the difference between hardware and software

• by design and fundamental principles, software has unbounded complex
Turing machine, halting problem).

• As long as hardware is seen as an implementation of a finite feature set
processor), technology advances (Moore's law) and competition push its perf
up and cost down.

• For a fixed function, as hardware speeds increase, more of the work can
"in software." So people take the penalty of making general-purpose hardw
specialize function by its programming. The bare hardware gets less "comp
better understood. This reduces risk, improves schedule, and saves money.

Core FPGA architecture



• modern devices have up to 200,000 of these primitives, plus the routing to i
nect them

• SNS LLRF accomplished with ∼3000 such cells; includes

  ◦ 4 12x12 multipliers and some adders to make up DSP feedback path

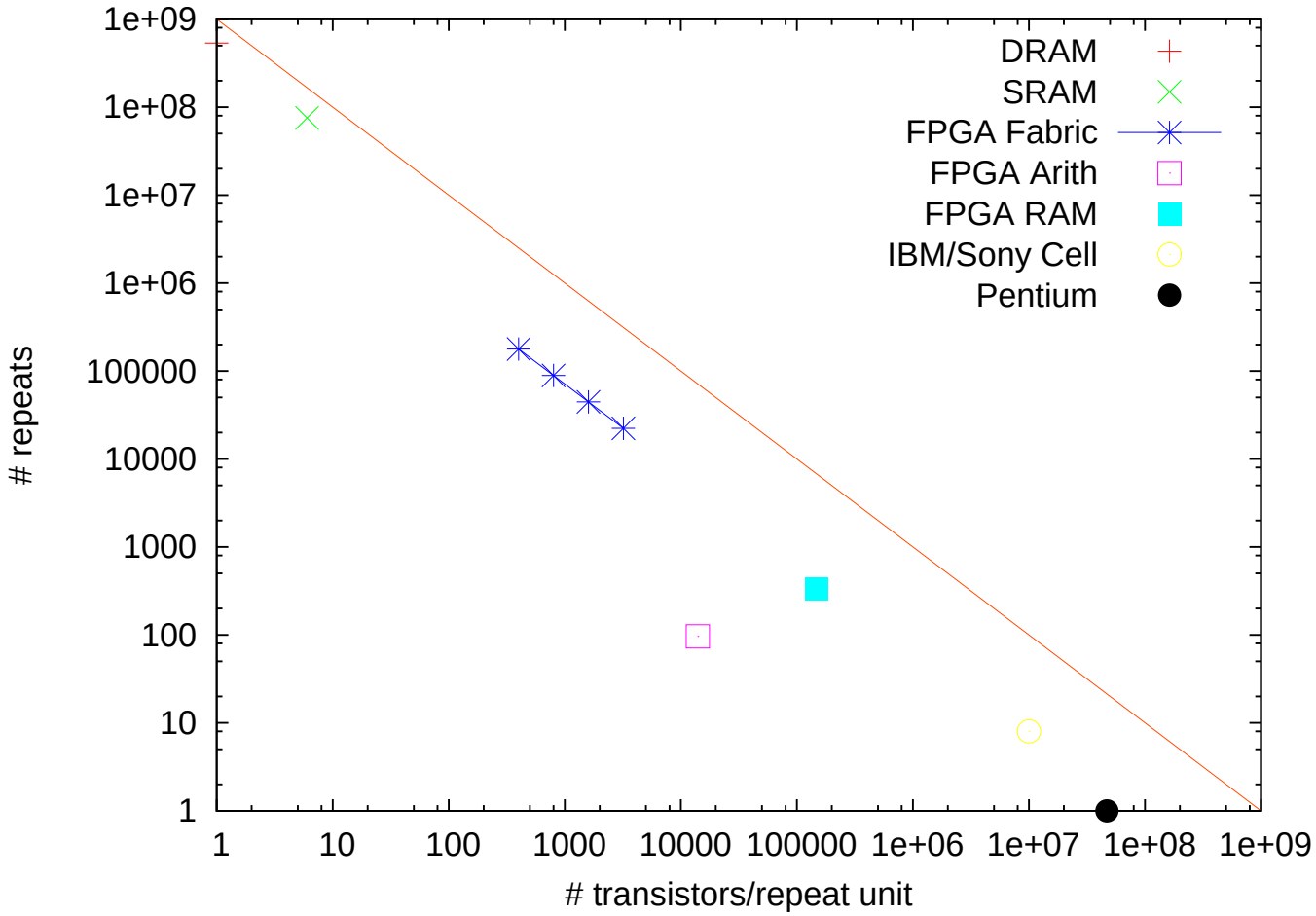
  ◦ 1 15-bit accurate pipelined CORDIC

  ◦ 4 programmable boxcar averagers

  ◦ a timing engine, host interface, and SPI adapter

• I choose to label FPGA programming “software,” since both the configura
and the corresponding source code are bits that can be moved across the net

Programmable device spectrum



Lessons

- Communication is hard, even on the chip level.

- For any fixed function, FPGAs and processors are many times worse (slower bigger) than an equivalent ASIC would be – if you could afford the developme schedule, and risk. Since the accelerator community is predominantly ma 'bottom feeders' we need to work with the general-purpose chips designed f (especially telecom)

Moore's Law

Larry Doolittle, 04 Apr 2003, comp.arch.fpga:

> "I have said since 2000 that Moore's law ends in 2005, not for techn
> reasons, but for economic ones. In concrete terms, we are now close enc
> to that threshold that I predict the industry will burn so much money tr
> to make 65 nm fab lines work, and the market for the resulting chips
> be weak enough, that those fabs will never turn a profit. The industr
> a whole will therefore not be financially willing or able to make a ser
> attempt at any following generations, or if they try, it will be at a m
> slower pace than we are used to."

Side effect: end of the line for bloat. *Hip hip hooray, and don't trip on your*

What 65 nm chips will we see in the next few years that are interesting to t
erator community?

- FPGAs
- Pentiums and Athlons

Both process huge amounts of data, more than enough juice to run all know
applications. What interconnect scheme makes it easy to attach an FPGA t
modity computer?

Interface trends:

- fewer pins
- higher speed
- lower voltage
- differential
- point-to-point instead of bus
- more “communication channel” less “address/data/control”
- block transfer aids throughput, but hurts latency

Examples:

- TTL → ECL → LVDS
- ISA → PCI → PCI-X
- GPIB → Ethernet
- Address-Data-Control→ HyperTransport

Specific hardware

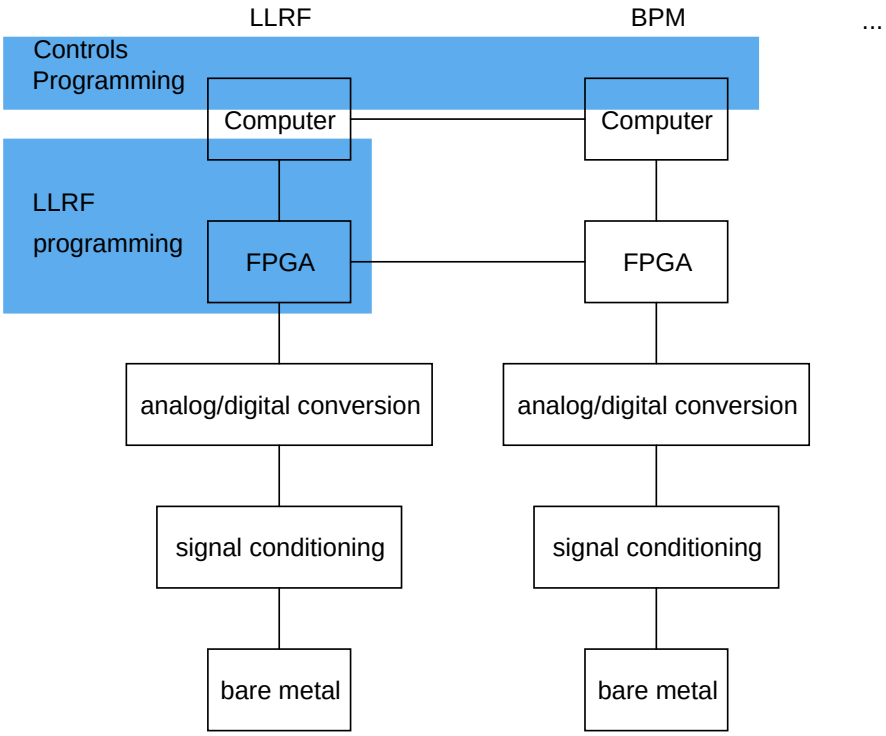
- 100-Mbit Ethernet
  - theoretical 12 MB/s
  - SLAC Coldfire
  - nanoEngine measured 3 MB/s with US$700 hardware
- USB-2.0
  - theoretical 60 MB/s
  - CY7C68013 - measured 33 MB/s with US$15 hardware
- 1000-Mbit Ethernet
  - theoretical 120 MB/s
  - direct attachment of PHY chip to FPGA could approach theoretical spe

full-speed, in-fabric TCP/IP is painful to think about. Consider UDP/IP Ethernet protocol like ATA over Ethernet (AoE).

Specific hardware (continued)

- HyperTransport
  - 200 MHz to 1400 MHz double data rate, 2, 4, 8, 16, or 32-bits in each d
(100 to 1600 MB/s peak)
  - "software compatible with PCI"
- PCI-Express
  - theoretical 250 MB/s per lane, 1, 2, 4, 8, 16, or 32 lanes.
  - "software compatible with PCI"
- Xilinx MGT
- Advanced TCA
  - compact PCI plus uncommitted differential pair routing, up to 1250 MB
  - SLAC controls group really attracted to its claimed 99.999% uptime

Hardware and software partitioning



- Hardware and software boundaries are different
- Custom computer hardware can't compete in price/performance with co processor boards
- Low latency, hard-real-time communication among FPGAs is not conside but is something to watch.

Another view of partitioning

- Application layer (beam physics)
- Controls network (archiving, alarms, knobs, high-level autonomous behav
spans subsystems)
- Programmable instrumentation hardware (includes both FPGA and compu
for calibration, linearization, demodulation, gain control, fast feedback loops
- Real Hardware - signal conditioning, ADC, power dissipation

Instrumentation and Control people, formerly working with pure hardware
need to transfer their expertise to designs using programmable hardware, g
both an FPGA (using Verilog or VHDL) and a computer.

Aside: all current languages suck. Still looking for a good language for this h
work that is designed to represent domain-specific knowledge in a way tha
used (and simulated for testing and development) efficiently, independent of th
of FPGA *vs.* CPU.

Complication: sharing a computer

The global controls group and the instrumentation group need to come to ter
the sharing of the "IOC" or "FEC."

- *Good fences make good neighbors.*
- Difficult to write an interface that makes sense to both sides
- We need to develop the expectation of rigorous testing:
  - test hardware with software, software with hardware.
  - automated regression tests
  - in-situ testing
  - bench fixtures
  - configure FPGA to emulate real hardware (carefully! don't want anyon
confused which mode is engaged!)

SNS LLRF has had some success with the first couple of items above. Ev
everything shakes out in simulation, glitches still hit in the field.

Hardware as a component of a meta object

Each hardware component needs the following support:

• Physical attachment and data communication to a computer

• Hardware-specific software to perform functions not considered suitable fo
compensating for hardware quirks, turning raw signals into more abstract
like bits to Volts, or I, Q to R, θ. Refer to Markus Hoffmann's parameter ex

• Namespace and device configuration entries in a data base. Mapping to a
physical coordinates, serial number, revision level, maintenance history, etc.

Coordination needed between controls software, instrumentation software, ins
tation firmware, and hardware. This is part of what controls experts me
they use the word "frameworks," where they focus on coordinating multiple
software.

Conclusions

• Moving functionality from hardware to software should reduce risk and u
Hardware engineers still have to design it and debug it.

• Managing complexity is hard, whether it is encapsulated in traditional sof
FPGA programming. Our community needs to build tools, techniques, and ex
to handle the software scope that now includes FPGAs (e.g., Warsaw's IID).

• All our hardware designs start to look alike after a while. More inter-lab coo
will help focus attention on the difficult parts, like accelerator-specific interfa
FPGA programming.

• Key hardware interface is the attachment of a custom FPGA board to a co
computer.

• Can't claim system success until you demonstrate a full-scale hardware and
test isolated from the billion-dollar accelerator.

Conclusions

• Moving functionality from hardware to software should reduce risk and u
Hardware engineers still have to design it and debug it.

• Managing complexity is hard, whether it is encapsulated in traditional sof
FPGA programming. Our community needs to build tools, techniques, and ex
to handle the software scope that now includes FPGAs (e.g., Warsaw's IID).

• All our hardware designs start to look alike after a while. More inter-lab coo
will help focus attention on the difficult parts, like accelerator-specific interfa
FPGA programming.

• Key hardware interface is the attachment of a custom FPGA board to a co
computer.

• Can't claim system success until you demonstrate a full-scale hardware and
test isolated from the billion-dollar accelerator.

Merci beaucoup!
Thank you!