Simplifying FPGA Design for SDR with a Network on Chip Architecture

Matt Ettus

<matt@ettus.com>

Ettus Research

GRCon13

Ettus Research

Outline

Ettus Research

USRP FPGA Capability

	Gen 1	Gen 2	Gen 3 E300	Gen 3 X310
FPGA	Cyclone I	Spartan 3	Zynq	Kintex 7
Logic Cells	12K	53K	85K	406K
Memory	26KB	252KB	560KB	3180KB
Multipliers	NONE!	126	220	1540
Clock Rate	64 MHz	100 MHz	200 MHz	250 MHz
Total RF BW	8 MHz	50 MHz	128 MHz	640 MHz
Free space	none	~50%	~75%	85+%

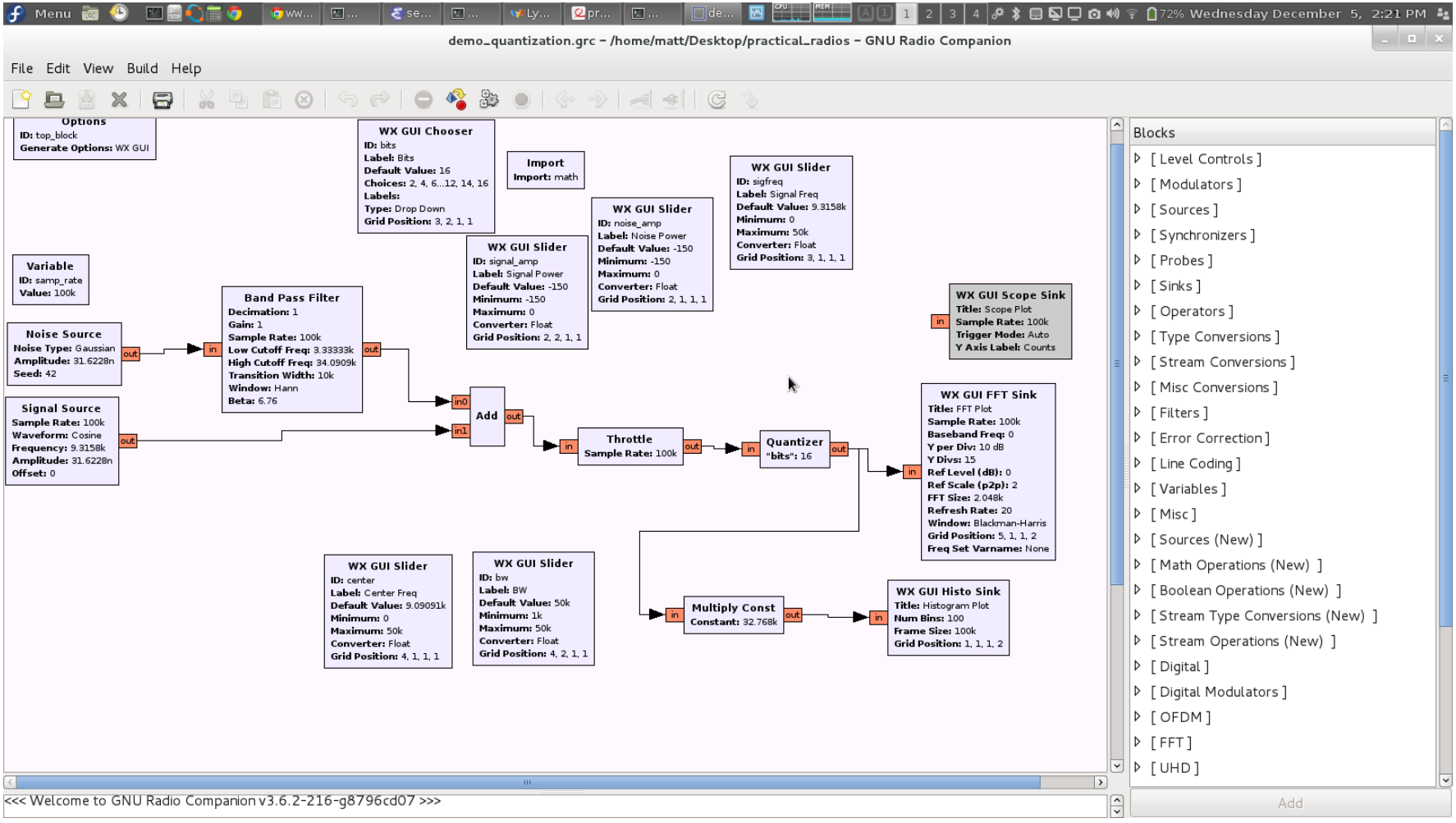
The Challenges

- FPGA computational capability scales with Moore's Law but our ability to use it does not
- Domain experts in algorithms not necessarily skilled in FPGA design, networking issues, verilog, or the internals of a particular radio system
- Poor reusability of computational elements across multiple designs due in part to lack of standard interfaces
- Difficulty migrating CPU code to FPGA
- FPGA reconfigurability is on a very slow time scale
  - Seconds to load a configuration
  - Minutes to hours to compile a new configuration
  - Hours to months to develop a new design
- Difficulty scaling designs beyond one FPGA
- Highly interconnected large designs make it difficult to meet timing requirements and slow down compile cycles

The Opportunities

- Massive new FPGAs allow a lot of freedom
  - Don't have to design like a "rewritable ASIC"
- Considering and valuing *composability* of blocks and code reuse early in the process can lead to large productivity gains
- Be able to take advantage of high level design languages and tools
  - can provide a better way for algorithm experts to express computation than in Verilog or VHDL
  - LabVIEW FPGA, DSP Designer, VivadoHLS, Coregen, Simulink, etc.

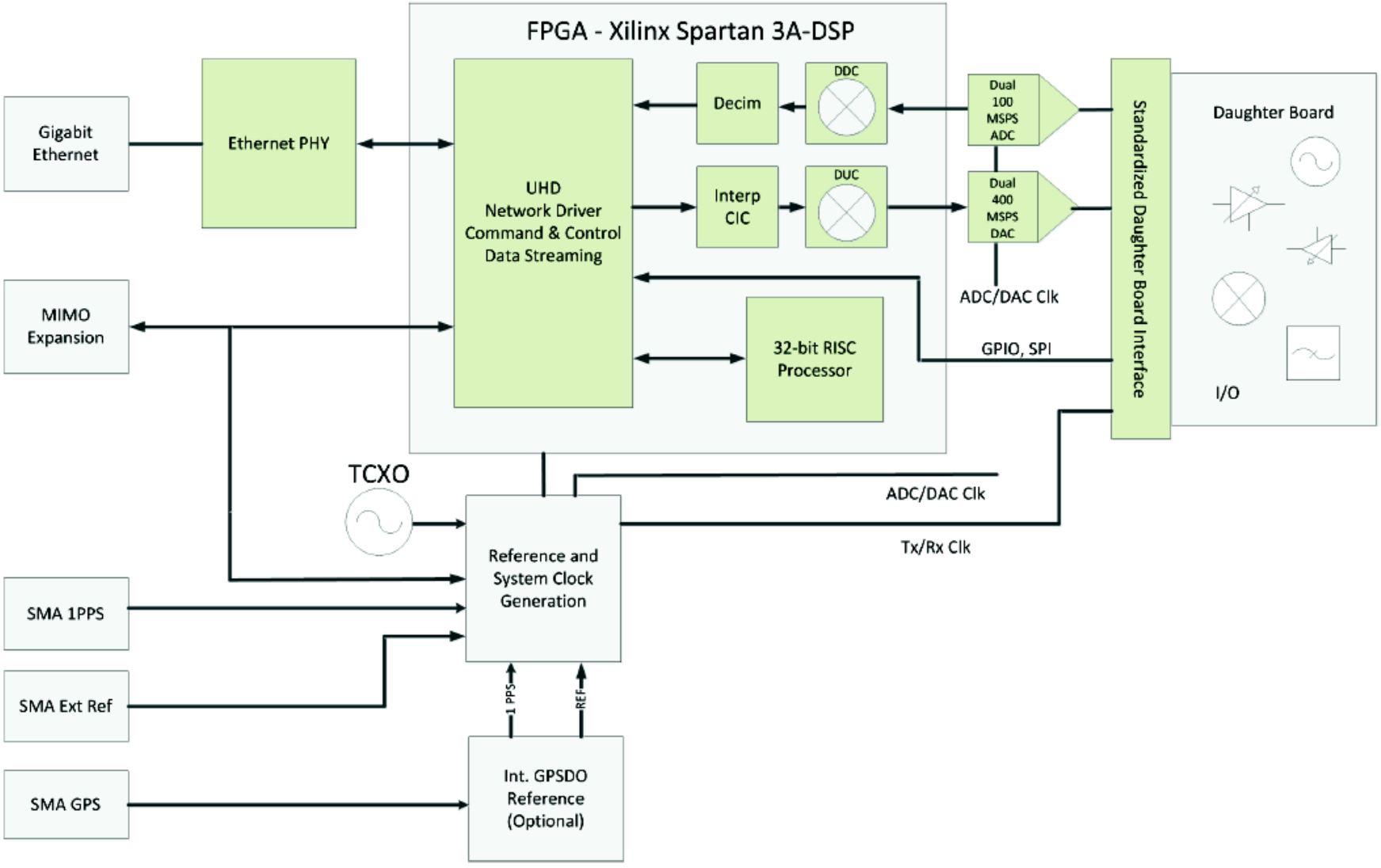
The Goal



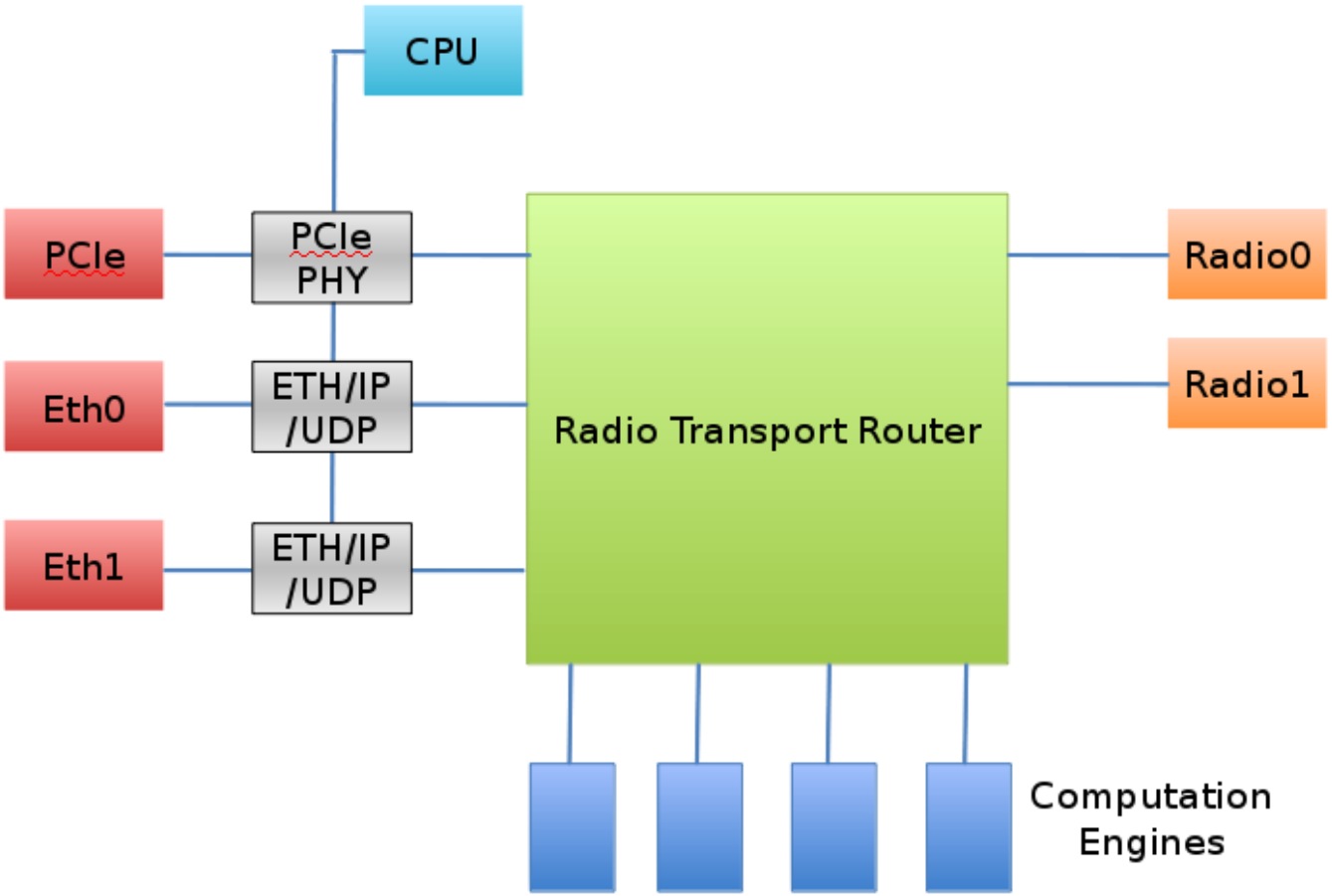
Ettus Research

Outline

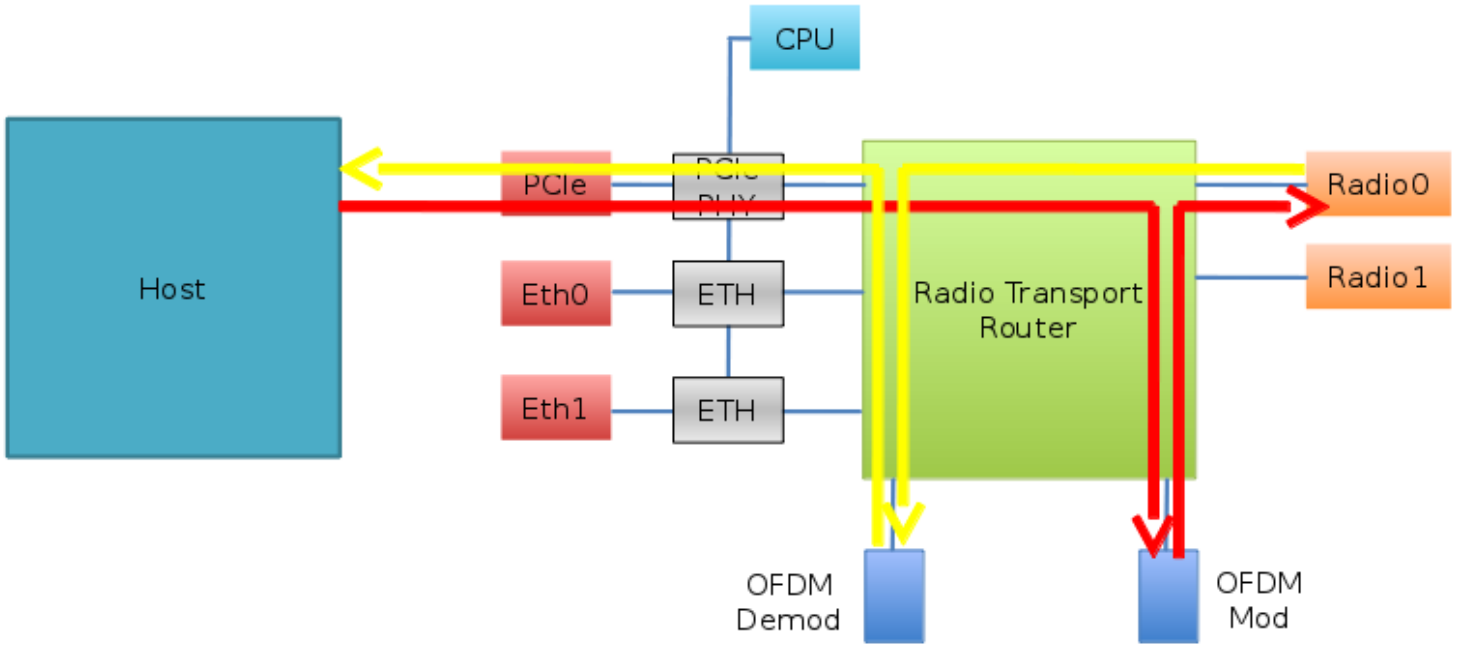
Traditional FPGA Data Flow



RF NoC


CPU
PCIe
PCIe PHY
Eth0
ETH/IP /UDP
Eth1
ETH/IP /UDP
Radio Transport Router
Radio0
Radio1
Computation Engines

RF NoC Example Application



RF NoC Principles

- Very simple Interface/API
  - 64-bit AXI FIFO in and out
  - AXI DDR3 interface for scratch space if necessary
  - VITA-49 standard packet formats
- All communication is packet-oriented over FIFO interfaces
- Data (baseband samples, symbols, packets, etc.) and Control are carried over the same path
- Data and Control carried in the same packet format
- All endpoints are created equal
  - Any-to-any communications
  - No “host” is necessary
- All communication is flow-controlled (both fine and coarse)
- Each block can be in its own clock domain

Radio Blocks

- Contain everything which must happen in the sample clock domain
  - All precise timekeeping and sequencing
  - All control of radio hardware
  - All sample-rate DSP (DUC, DDC, IQ Balance, most sample rate conversion)
- Produces and consumes packets of baseband samples at a fixed rate

Computation Engines

- Can perform arbitrary processing on both data and control
  - Generic FIR engine
  - FFT machine
  - OFDM sync
  - Turbo decoding
  - Frequency hopping state machine
  - AGC state machine
  - Protocol processing
  - Upper layer stacks
  - CPU (i.e. Microblaze)
  - Large memory buffers
  - etc.

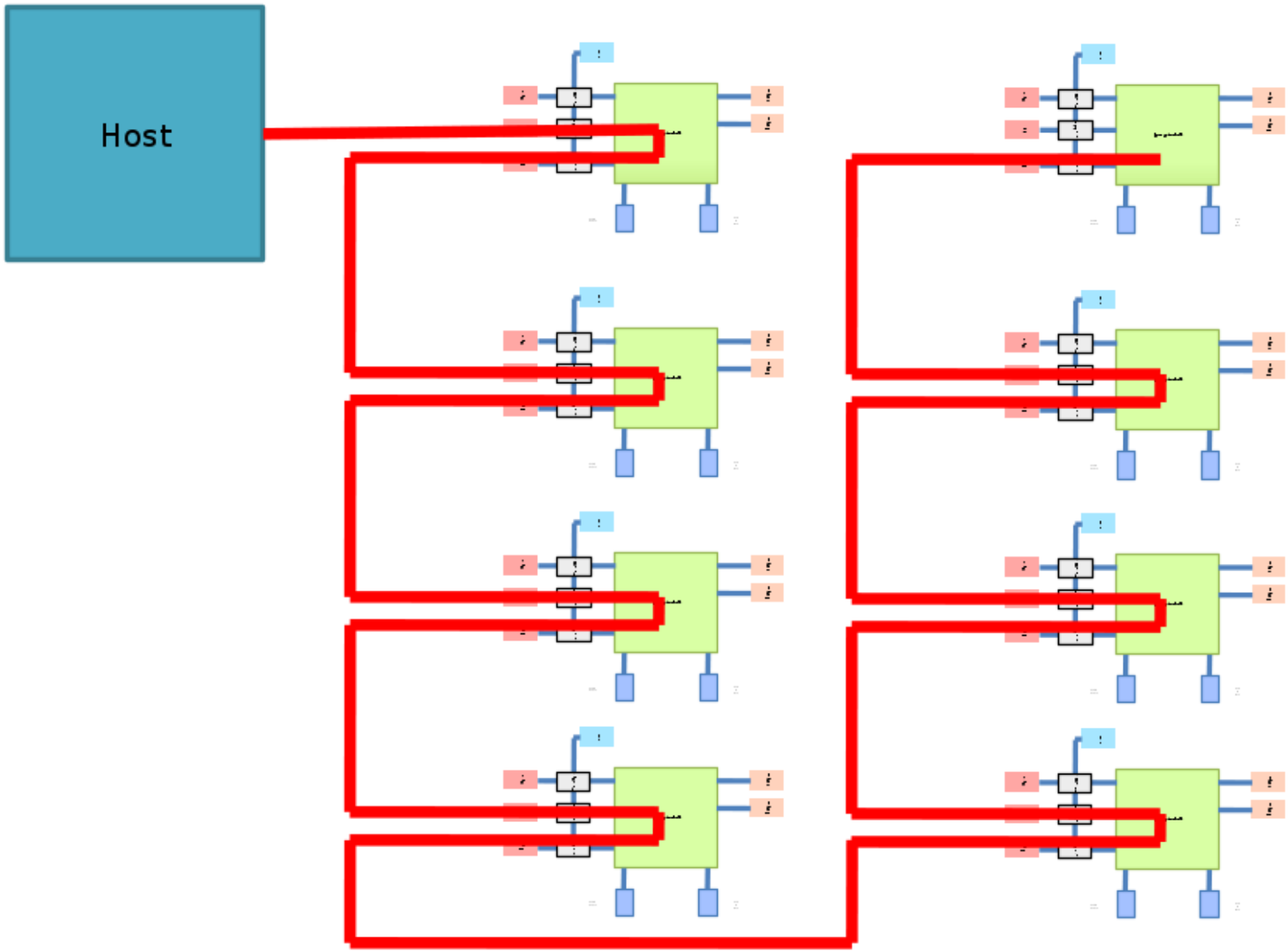
External Interfaces

- Packets entering or leaving FPGA via
  - 1G/10G Ethernet
  - PCIe
  - USB3
  - AXI interfaces to ARM and System RAM in Zynq
  - Adaptation layer to other processing paradigms (i.e. massive multicore, GPU, etc.)
- Filters out non-RFNoC traffic and passes it to the control processor
  - e.g. ARP, Ping
  - Out-of-band control like setting up the network router, misc. hardware controls, etc.
  - Network diagnostics
- Strips off other protocol layers and compresses headers on ingress, reverse operation on egress
- Maps RFNoC address to external protocol address (i.e. MAC and IP addresses and UDP port)

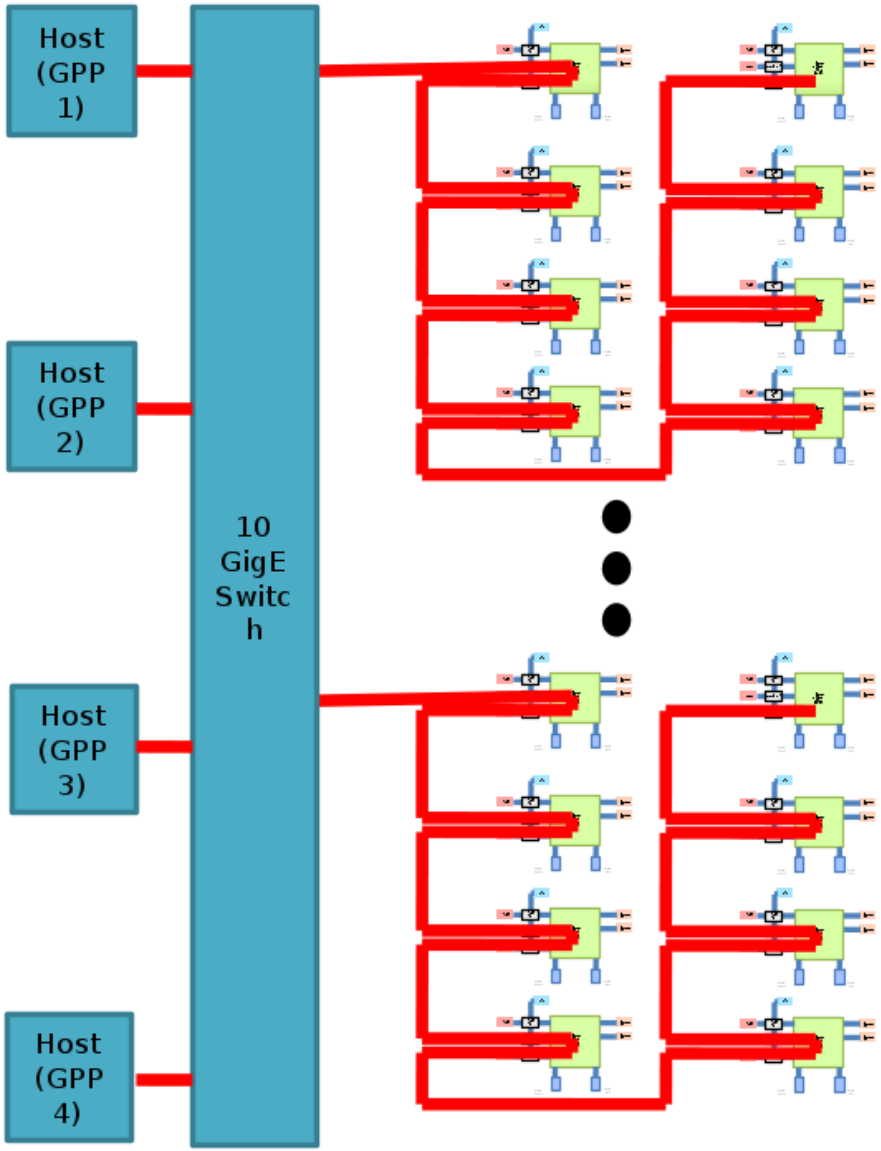
Network Fabric

- Full crossbar to route packets between blocks
- Routing tables are loaded out of band by the control processor, and are set up by the application
- All routing decisions are made entirely based on the first line of every packet
- No packet is allowed into the crossbar until it is complete
  - Prevents slow packets from causing congestion
  - Ensures data flows through at full rate
- Flow control assures that any packet entering the router can also exit because there is buffer space guaranteed on the other side

Simple MIMO Example



Massively Scalable MIMO



End to End Flow Control

- Uses coarse-grained (packet-oriented) flow control
  - Data consumers report consumption of data packets back to producers with special flow control packets, inband
  - No producer may send data to a consumer unless it knows there is enough buffer at the consumer to hold it
    - Guarantees no blocking in the routing fabric
- If any block in a chain originating from a timed source (i.e. radio receiver) can't keep up, samples back up to the source
  - Source reports overflow into the chain when its output buffers fill
- If any block in a chain ending at a timed sink (i.e. radio transmitter) can't produce data fast enough, the pipe empties
  - Destination reports underflow back up the chain towards the source when its input buffers are empty

Outline

Ettus Research

Status

- Working implementation of radios, network fabric, external interfaces, and flow control across multiple product lines with a single code base
- A small number of simple computation engines have been implemented
- To Do
  - Implement more interesting computation engines
  - Demonstrate multi-FPGA flowgraphs
  - Automate the setup and routing process based on a GNU Radio flowgraph
  - Automatically migrate processing from host to computation engines
  - Automatically build FPGA image with user selected computation engines
    - Take advantage of partial reconfiguration
  - Produce skeleton reference computation engines in various design environments
    - Verilog, VHDL, MyHDL, LabVIEW FPGA, Simulink, VivadoHLS, BlueSpec, etc.

Contact

- Matt@Ettus.com
- @EttusResearch
- http://ettus.com