An Ethernet-Accessible Control Infrastructure for Rapid FPGA Development

Andrew Heckerling, Thomas Anderson, Huy Nguyen, Greg Price, Sara Siegal, John Thomas

High Performance Embedded Computing Workshop

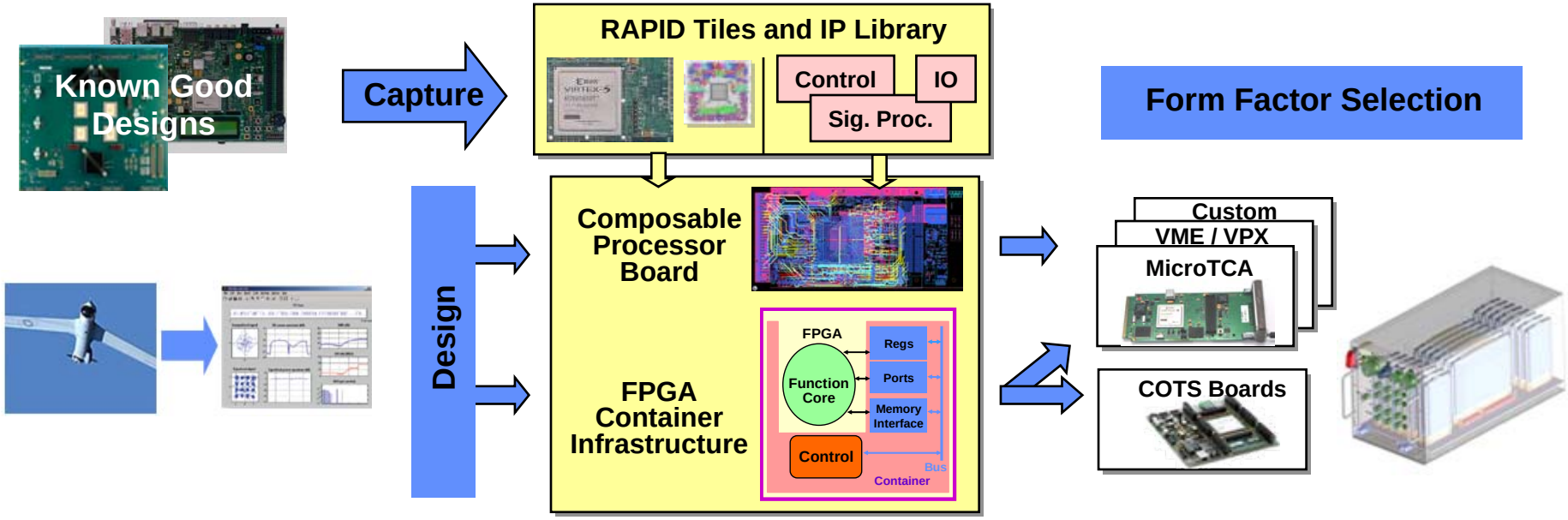
24 September 2008

This work is sponsored by the Department of the Air Force, under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

Outline

- Introduction and Motivation
- Container Infrastructure
  - Concept
  - Implementation
- Example Application
- Summary

Rapid Advanced Processor In Development (RAPID)



Main features of RAPID:

- **Composable processor board**
  - **Custom processor composed from tiles extracted from known-good boards**
    - **Form factor highly flexible**
  - **Tiles accompanied with verified firmware / software for host computer interface**
- **Co-design of boards and IPs**
  - **Use portable FPGA Container Infrastructure to develop functional IPs**
    - **Container has on-chip control infrastructure, off-chip memory access, and host computer interface**
  - **Surrogate board can be used while target board(s) being designed (custom) or purchased (COTS)**

Motivation

Goal:

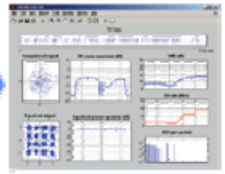
← **Quick Development: 7-12 Months** →

1 - 3 Months

6 - 9 Months

1 - 3 Months

Requirement Analysis



Sys. Des. (Algorithm Architecture)

RAPID Boards

System Dev., Packaging & Verification, Demo

Reduce system development time in half

Airborne Radar System Demo

Receiver Array
4 Channels
20 MHz BW

RAPID Front-end Signal Processor

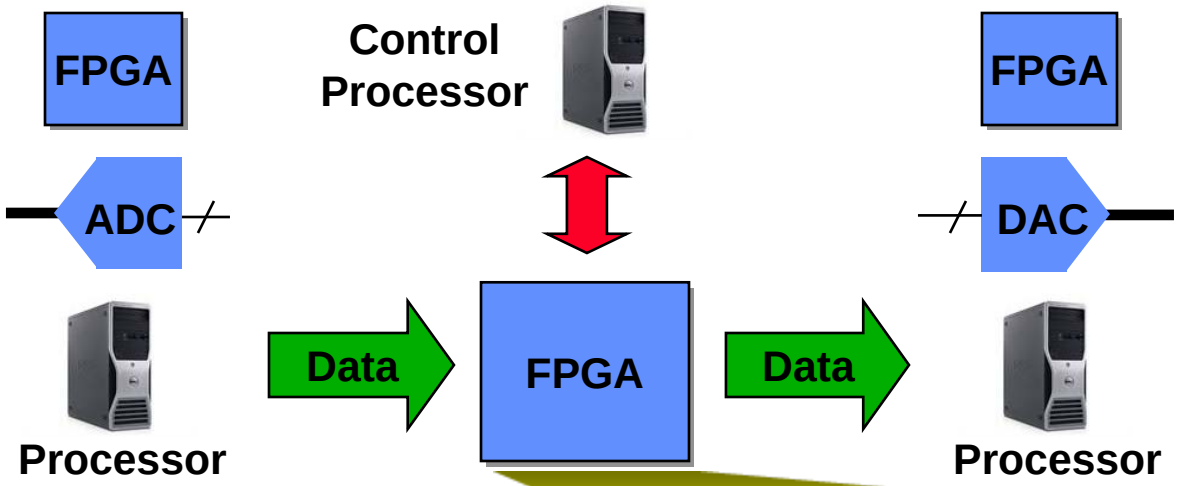
Back-end Processor

ROSA II System Computer

Outline

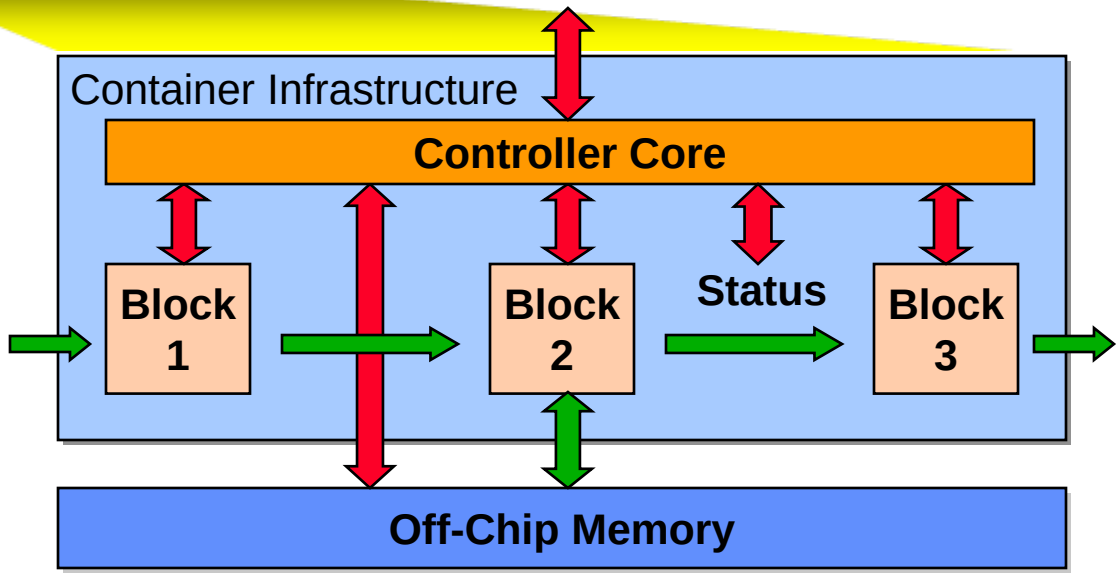
- Introduction and Motivation
- Container Infrastructure
  - Concept
  - Implementation
- Example Application
- Summary

FPGA Processing Application



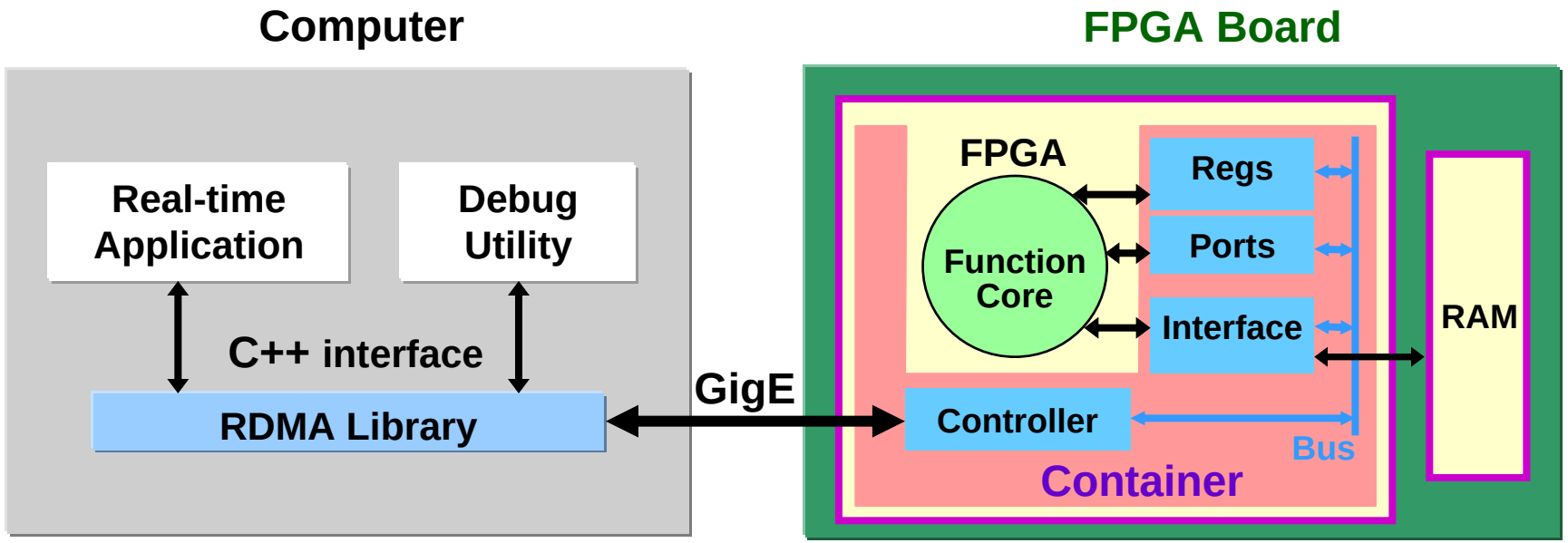
- FPGA processes high-speed streaming data from various sources
- Control processor initializes, monitors, and configures FPGA

- Control Processor gains visibility into FPGA via Controller Core
- Controller Core provides monitoring and control of memories and functional blocks
  - Set parameters, load memory, monitor status



FPGA Container Infrastructure

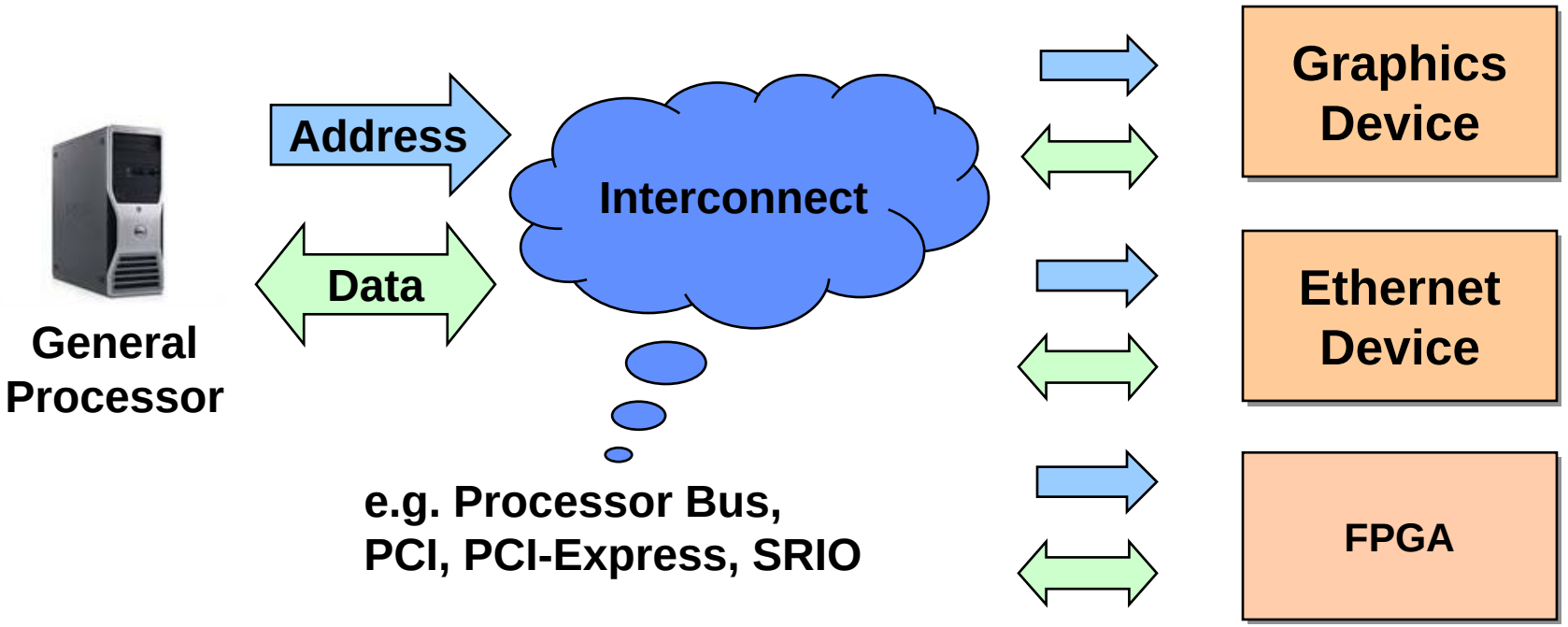
- FPGA Function Core development can be accelerated with infrastructure provided by Container: host computer interface, on-chip registers, ports, and memory interface
- Real-time application or debug utility can access any address (registers, ports, and memories) on the FPGA
- Message formatting and data transfer operations are supported through Remote Direct Memory Access (RDMA) library



Outline

- Introduction and Motivation
- Container Infrastructure
  - Concept
  - Implementation
- Example Application
- Summary

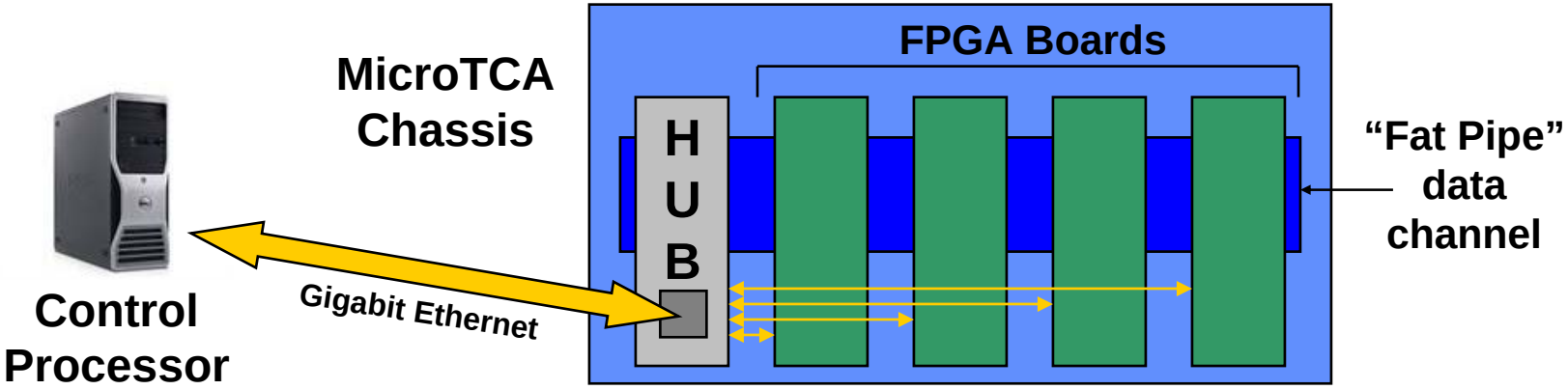
Motivation for Memory-Mapped Control



- Memory-Mapped Control Means
  - Device control via simple reads/writes to specific addresses
  - Processor and interconnect not specific to any device
    - With proper software, processor can control any device
- Container Infrastructure extends concept to FPGA control

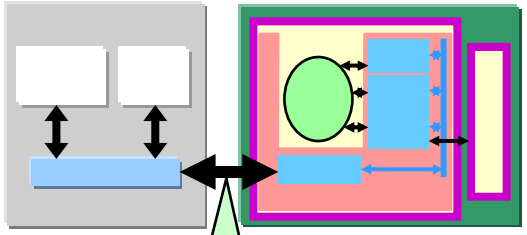
Interconnect

- Interconnect Choices
  - Ethernet, Serial RapidIO, PCI Express, etc.
- Platform-Specific Considerations
  - MicroTCA has Gigabit Ethernet channel to all payload slots, separate from higher-speed data channels



- Advantages of using Gigabit Ethernet
  - Ubiquitous
  - Wide industry support
  - Easy to program

Memory-Mapped Control Protocol



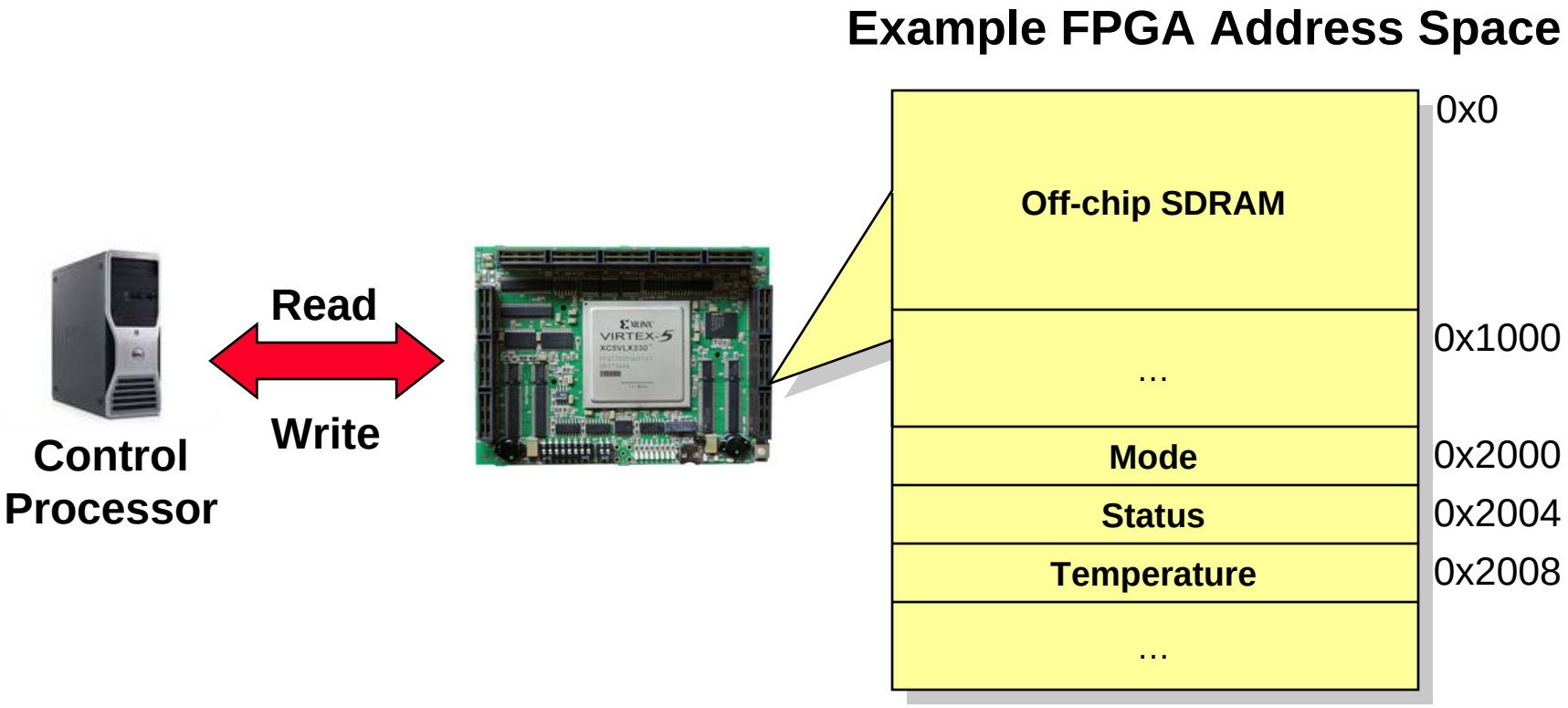
- Stateless “request/response” protocol
- Reads and writes an address range (for accessing memory) or a constant address (for accessing FIFO ports)
- Presently implemented on top of UDP and Ethernet

Message Format

0	magic	version
4	node number	
8	command	
12	address	
16	length	
20	flags	
24	message tracking id	
28	data (optional)	
…		

Command	Purpose
READ	Request read data
WRITE	Request write data
DATA	Response to READ
ACK	Response to WRITE
NACK	Response to READ/WRITE (command failed)

Memory-Mapped Control on FPGA

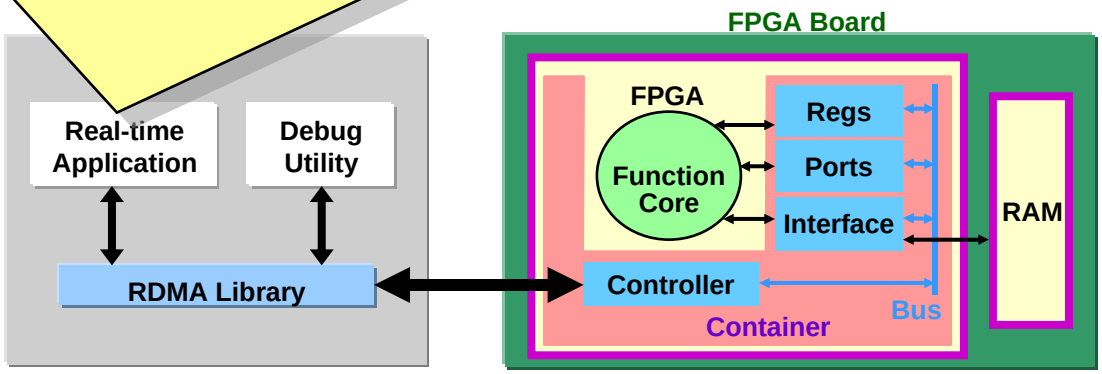


- Each device or core has an address within the FPGA
- Control processor refers to these addresses when reading from or writing to the FPGA

Real-Time Application Example

- Real-Time Application uses simple C++ methods to communicate with FPGA
- C++ interface portable to other interconnects (SRIO, PCIe)

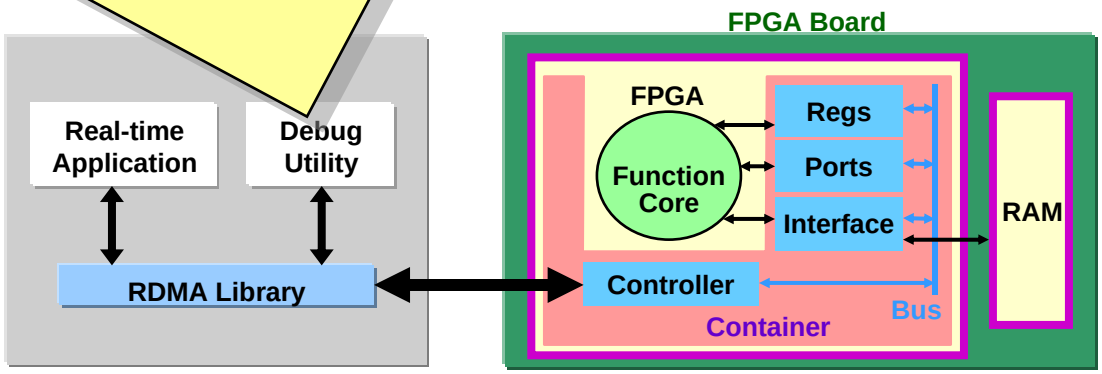
```
// Create an FPGA access object
FpgaUdpReadWrite fpga("fpga-network-address", FPGA_UDP_PORT);

// Send input data from myBuffer to the FPGA
fpga->write(FPGA_INPUT_DATA_ADDR, INPUT_DATA_LENGTH, myBuffer);

// Read back the output data
fpga->read(FPGA_OUTPUT_DATA_DDR, OUTPUT_DATA_LENGTH, myBuffer);
```



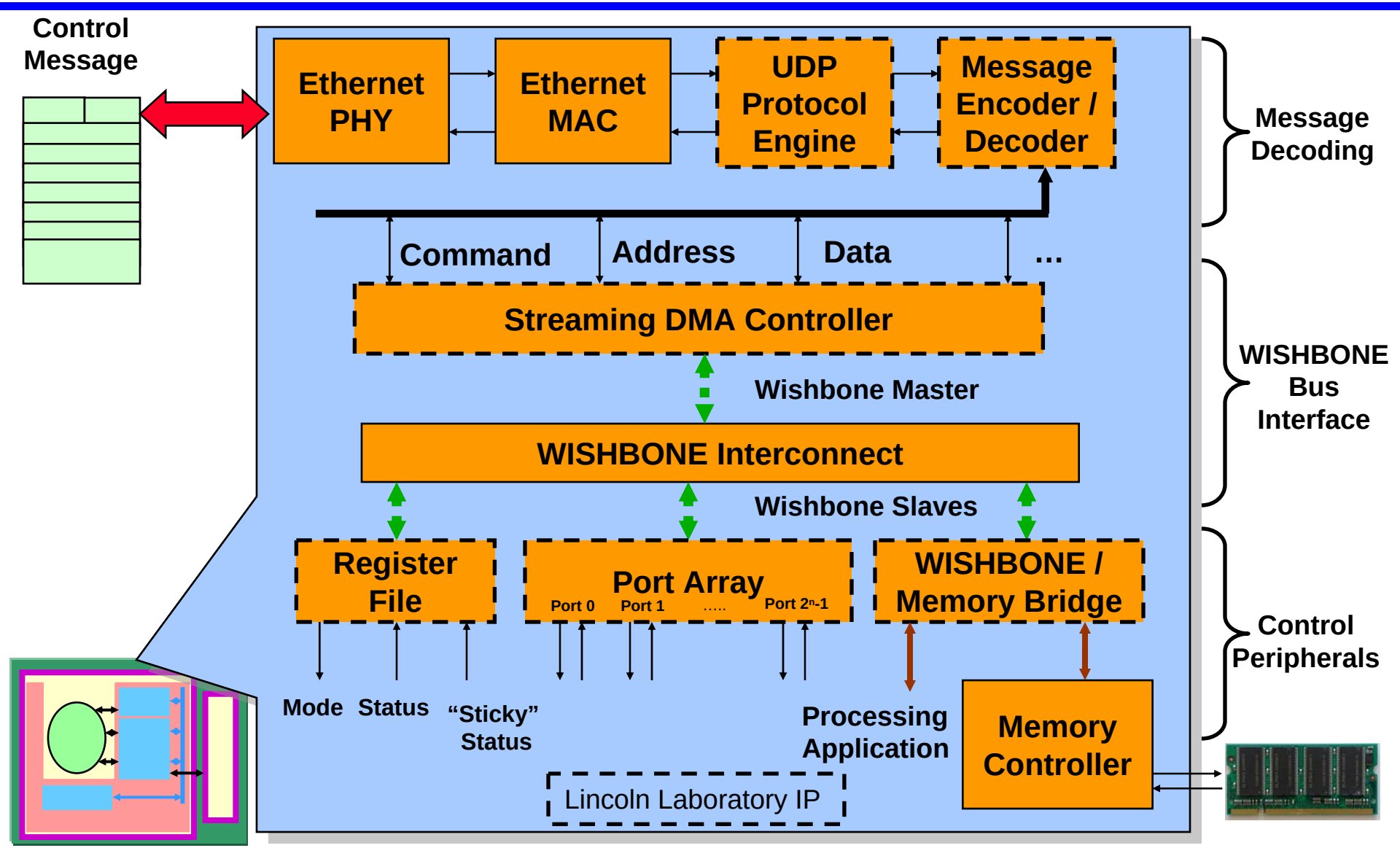
Command-Line Example

- Command-line and scripting interface provides debug access to FPGA container
- Function core can be tested before final software is written

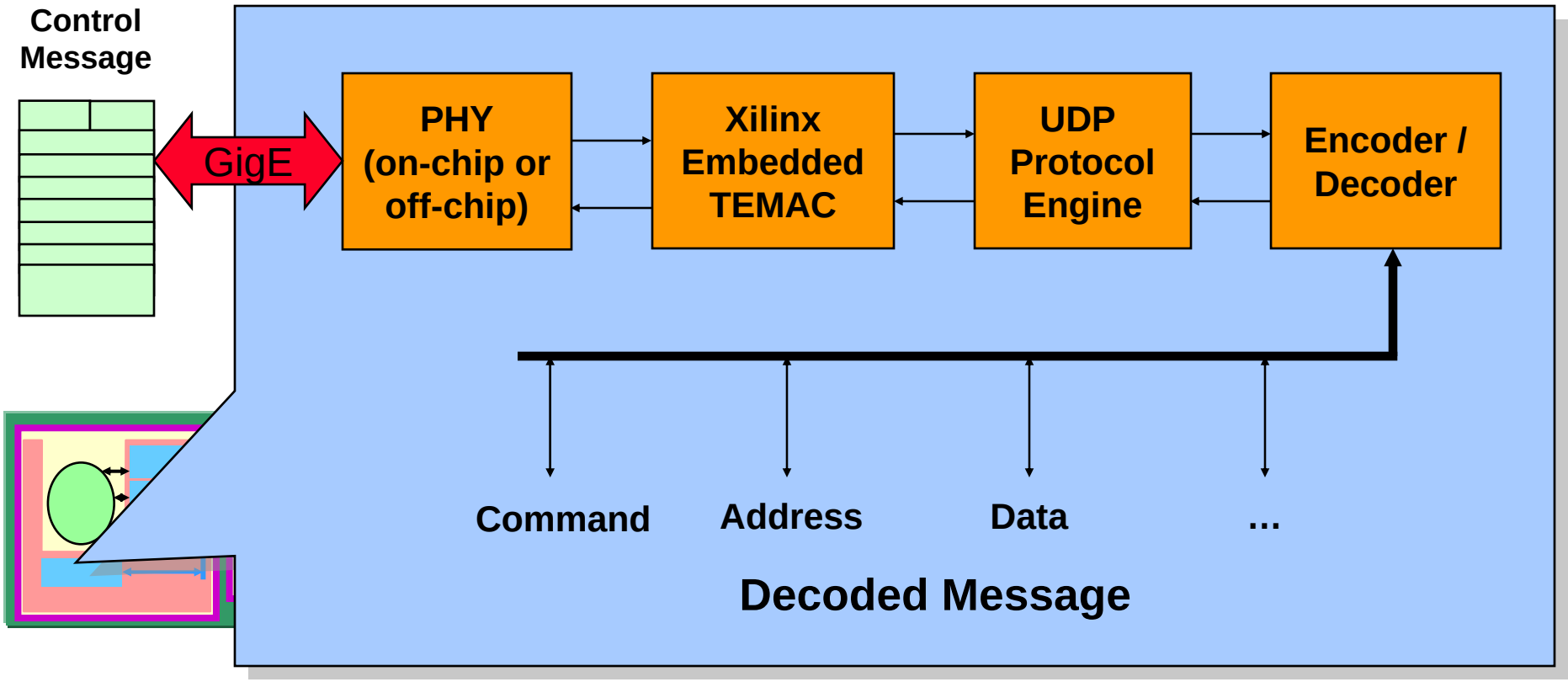
```
# Send input data to the FPGA
w 192.168.0.2 1001 0x0 sample_input_data.bin

# One-second delay (in ms)
P 1000

# Read back the output data
r 192.168.0.2 1234 0x10000000 0x8000 result_data.dat
```



Integrated Container System

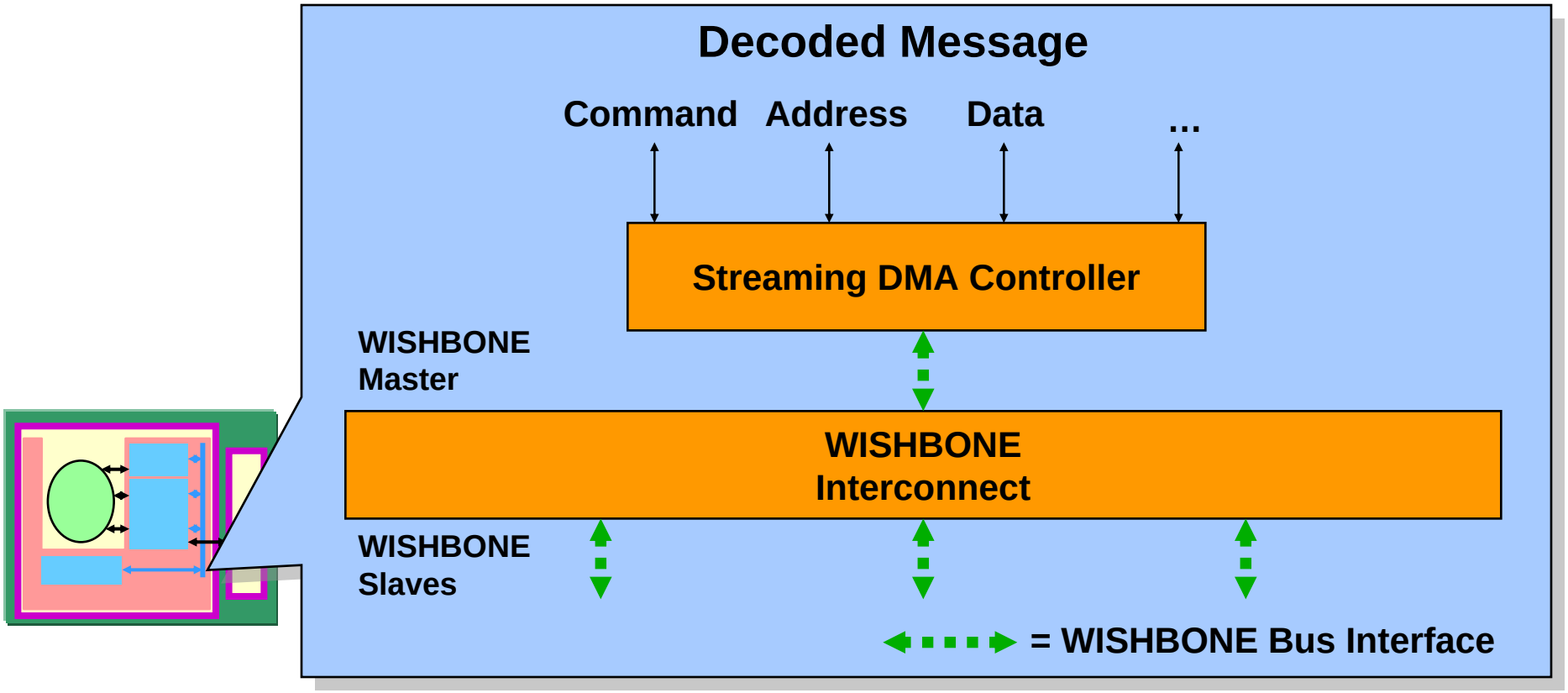


Message Decoding



- Inside the FPGA, the control message is decoded into a memory-mapped read or write command
- Can mix and match components to implement different protocols

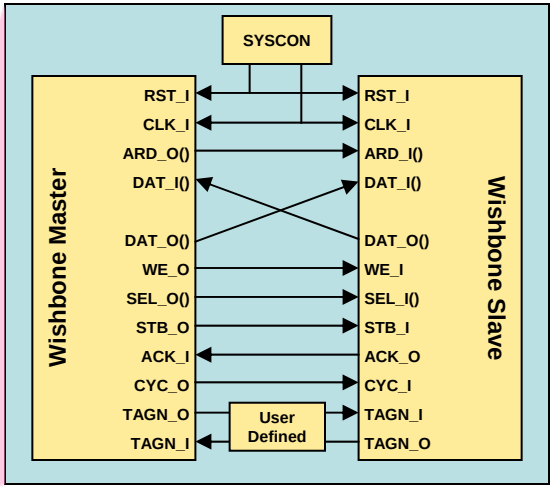
WISHBONE Bus Interface



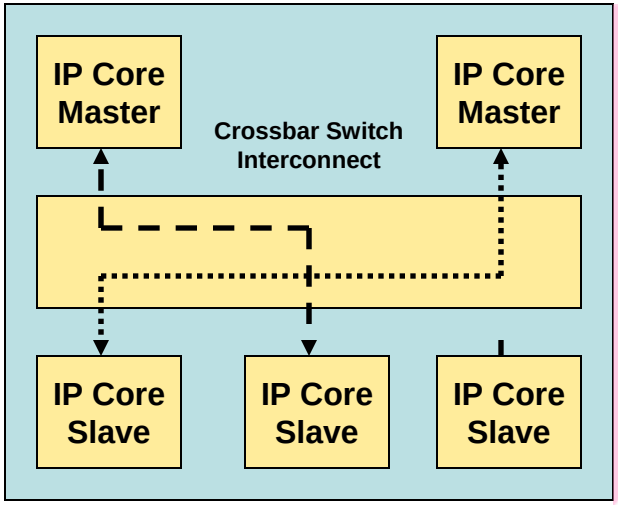
- Streaming DMA Controller (SDMAC) handles read/write commands by generating WISHBONE bus cycles
- WISHBONE Interconnect routes transactions to destinations based on memory map
- Transaction block sizes range from one word (four bytes) to 8k bytes

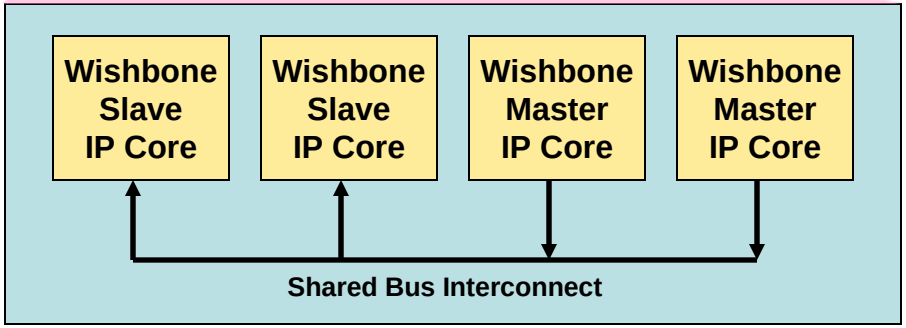
WISHBONE Bus

- **WISHBONE is a flexible, open-source bus for system-on-chip designs**
  - **Specifies a logical (not electrical) interface between IP peripherals**
  - **WISHBONE peripherals and interconnect hubs are available on the OpenCores web site**



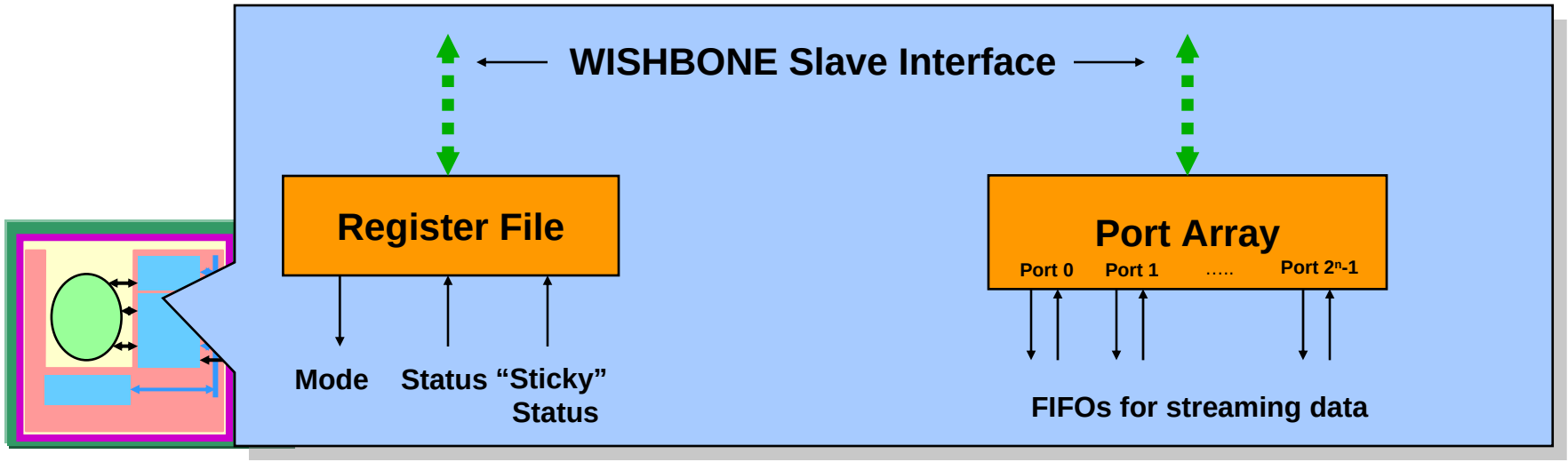
FPGA





Diagrams and specification: http://www.opencores.org/

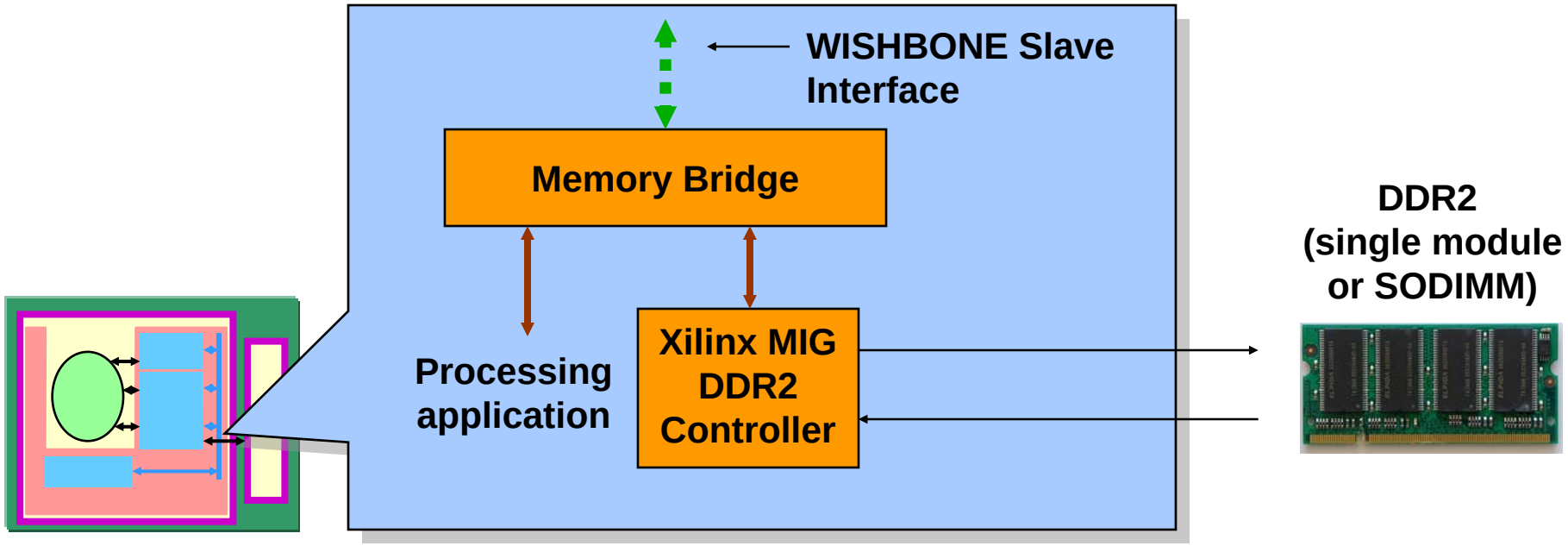
Control Peripherals: Register File and Port Array



- Each register has an address
- Registers enable / disable features, trigger processes, report condition and events
- Multiple registers can be used in any combination of types

- Port Array translates memory-mapped WISHBONE operations to data streams
- Useful for testing computational blocks that expect data to arrive in a FIFO-like fashion

Control Peripherals: DDR2 SDRAM Controller Interface



- High-speed processing application and lower-speed WISHBONE interface share access to DDR2 memory
  - Used to preload data into external memory for use by the processing application or for debugging
- Xilinx memory controller interfaces to memory

Resource Usage on Virtex-5 SX95T

Component	LUTs	FFs	BRAM Kbytes	Clock rate
Controller Core Functions	3,172 (5.4%)	3,853 (6.5%)	83.25 (7.6%)	125 MHz
Register File	132 (0.2%)	200 (0.3%)	0 (0%)	125 MHz
Port Array	396 (0.7%)	531 (0.9%)	0 (0%)	125 MHz
DDR2 Bridge / Memory Controller	2,309 (3.9%)	2,275 (3.9%)	31.5 (2.9%)	125 MHz 200 MHz
Total	6,009 (10.2%)	6,859 (11.6%)	114.75 (10.5%)	

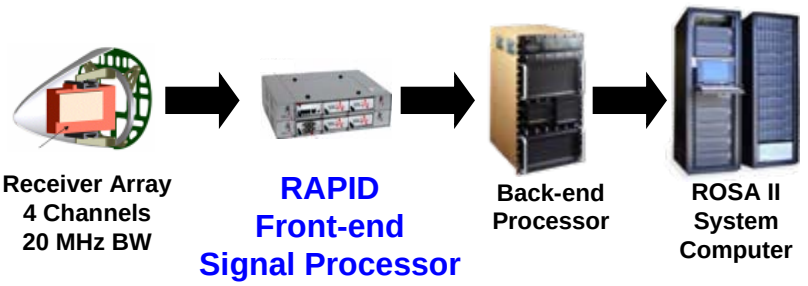
- Container infrastructure consumes 7-12% of Virtex-5 SX95T depending on functionality used
- Resource usage is constant as FPGA size increases

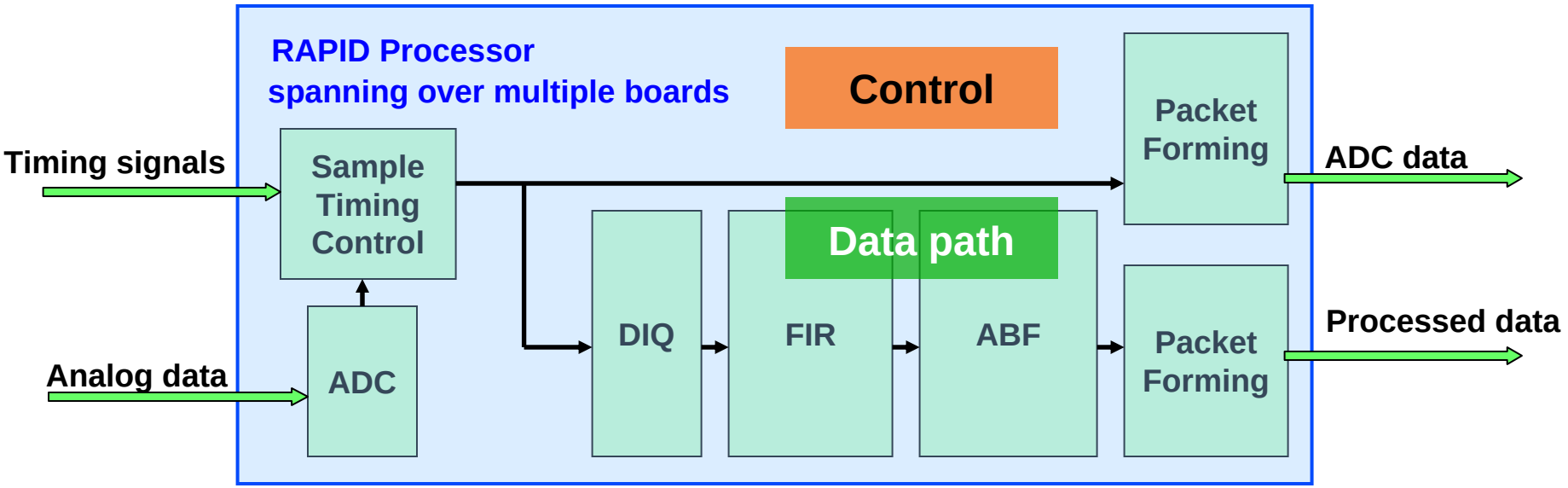
Outline

- Introduction and Motivation
- Container Infrastructure
  - Concept
  - Implementation
- Example Application
- Summary

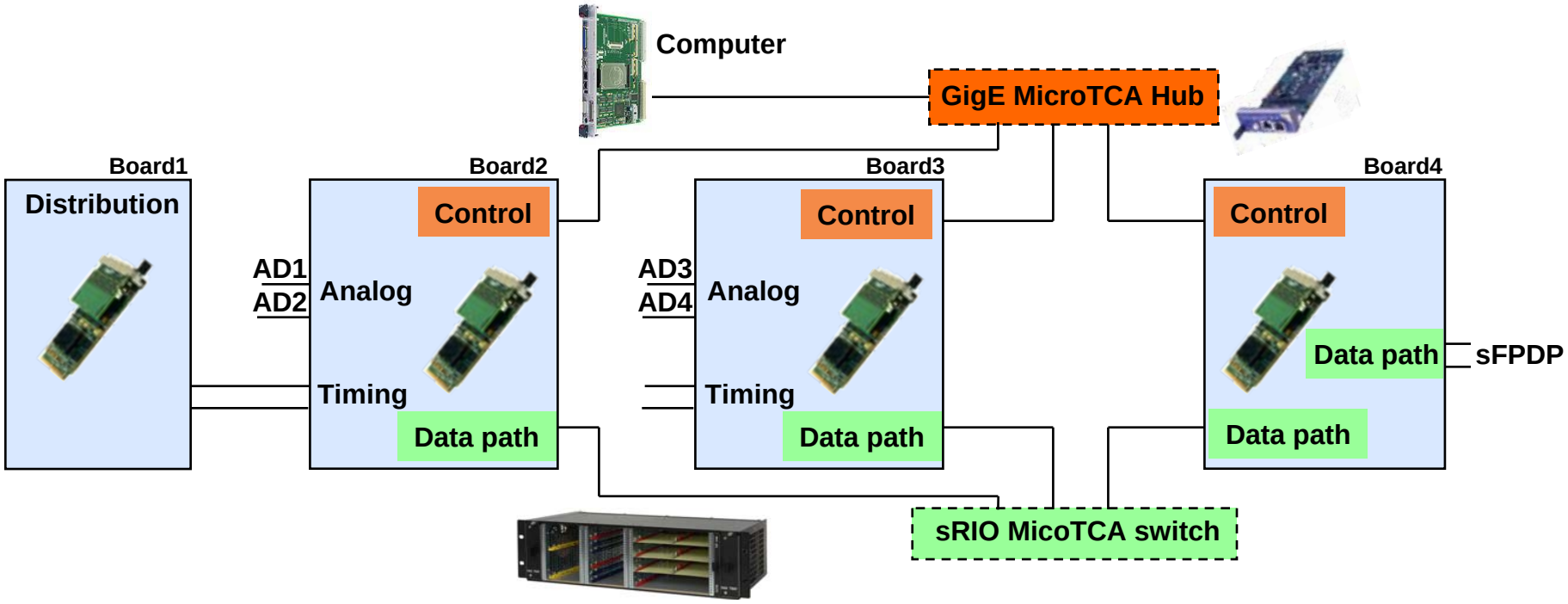
ROSA II Front-End RAPID Processor

- ROSA II
  - Open architecture for putting a radar system together quickly
  - Interfaces and protocols are defined for subsystems
- Front-end Processor
  - Developed with RAPID process
  - Performs Digital IQ, FIR, and Adaptive Beamforming





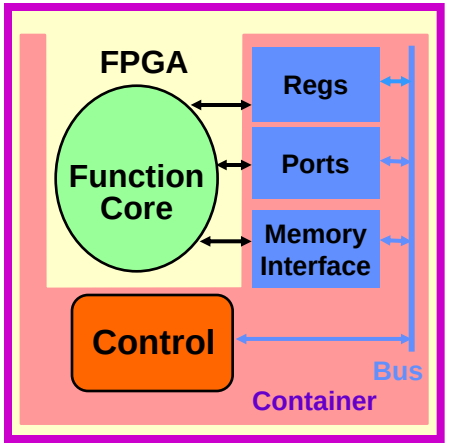
RAPID Front-End Processor System



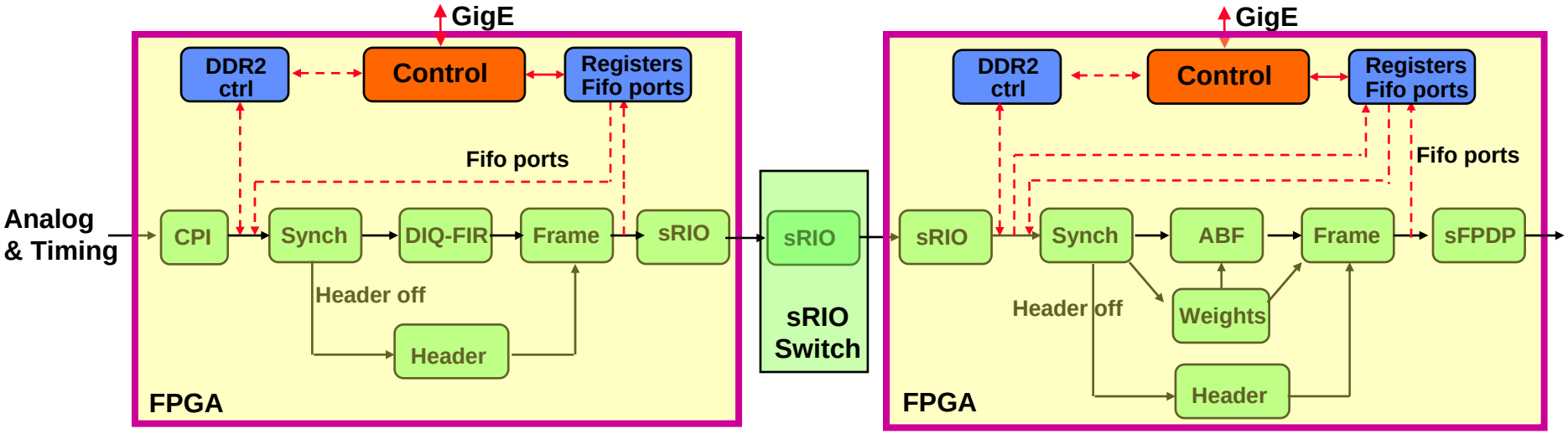
- Processor is mapped to a MicroTCA system
  - Separate Control and Datapath on one Hub card
  - GigE base channel 0 is used for system control (~1 Gb/sec)
  - Serial RapidIO fat-pipe is used for datapath (~10 Gb/sec)
- Container Infrastructure allows access to each FPGA via Gigabit Ethernet
  - High observability and controllability

Development with FPGA Container Infrastructure

- Container provides host computer access and on-chip control structure
  - Helps development of custom function cores
  - Flexible script-based method for sending test data and reading back response



- Facilitates system-level testing with multiple data-path source and destination options
  - Data-path source and destination set via mode registers
  - Raw data from memory, FIFO ports, or stream input. Processed result to memory, FIFO ports, or stream output

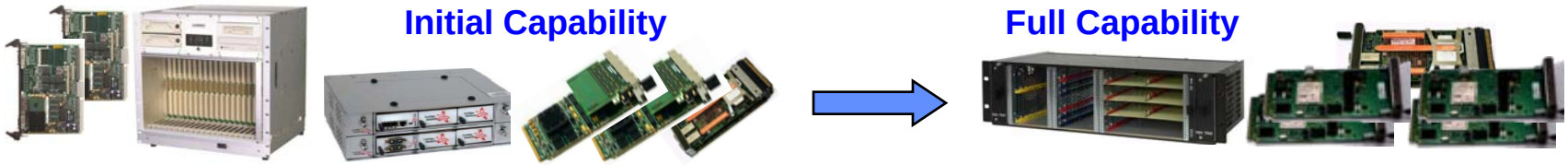


System-Level Benefits

- **Eases development of Function Cores**
  - Script interpreter on host computer allows easy sending of test data and reading of results
  - *Incremental* system integration tests with multiple data sources and output destinations
  - Estimated saving in system integration test: 2 months

- **Enables development on surrogate system(s)**
  - Highly portable Container Infrastructure allows early development

  - 2 month head start while waiting for COTS system (initial capability)
  - 6-9 month head start with custom boards (full capability)



Summary

- Presented a Container Infrastructure for FPGA development
  - Memory-mapped control protocol for accessing FPGA registers, FIFO ports, and external DDR2 memory
- Container Infrastructure enables fast system development
  - Helps development of FPGA function cores
  - Facilitates incremental system integration
  - Allows early FPGA development on surrogate boards
- Future work
  - Extend framework to non-Gigabit-Ethernet channels
  - Ensure high portability and interoperability with COTS boards
  - Extend the container concept for high-speed data co-processing

Acknowledgements

RAPID Team

- Ford Ennis
- Michael Eskowitz
- Albert Horst
- George Lambert
- Larry Retherford
- Michael Vai

UDP protocol engine

- Timothy Schiefelbein