aSoC: A Scalable On-Chip Communication Architecture

Russell Tessier, Jian Liang, Andrew Laffely, and Wayne Burleson
University of Massachusetts, Amherst
Reconfigurable Computing Group

Supported by National Science Foundation Grants CCR-081405 and CCR-9988238

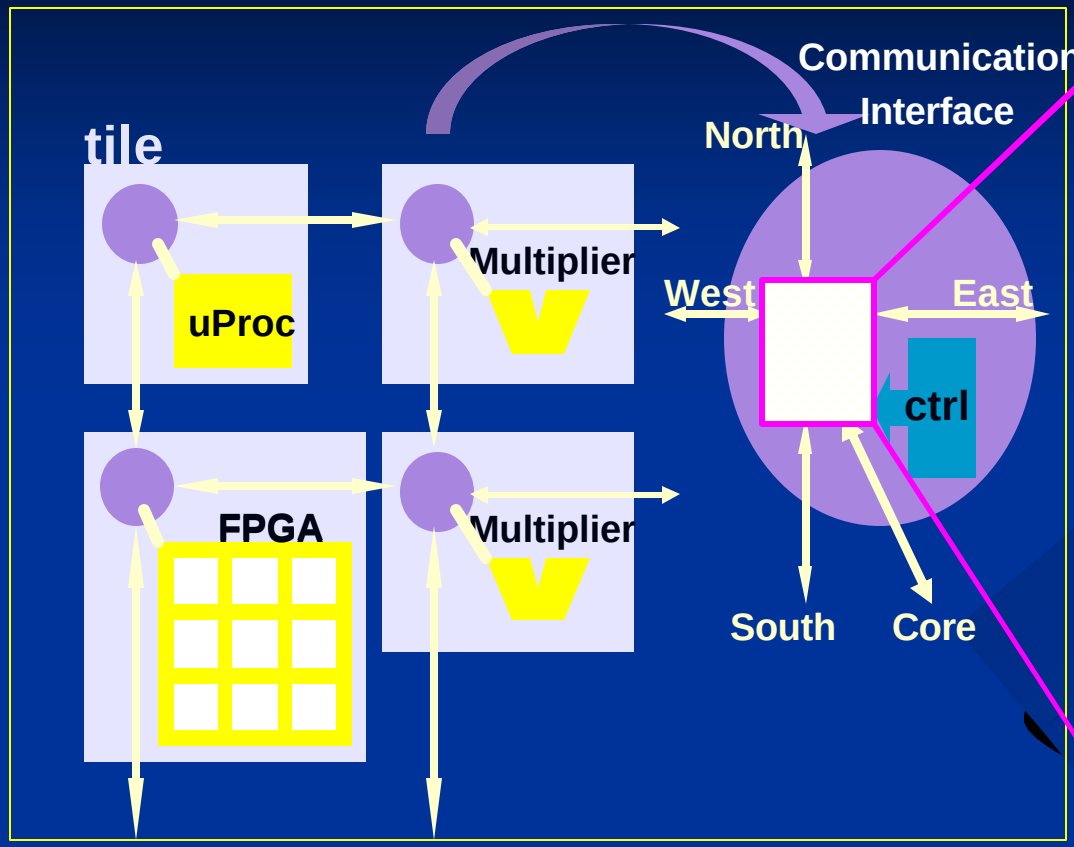
Outline

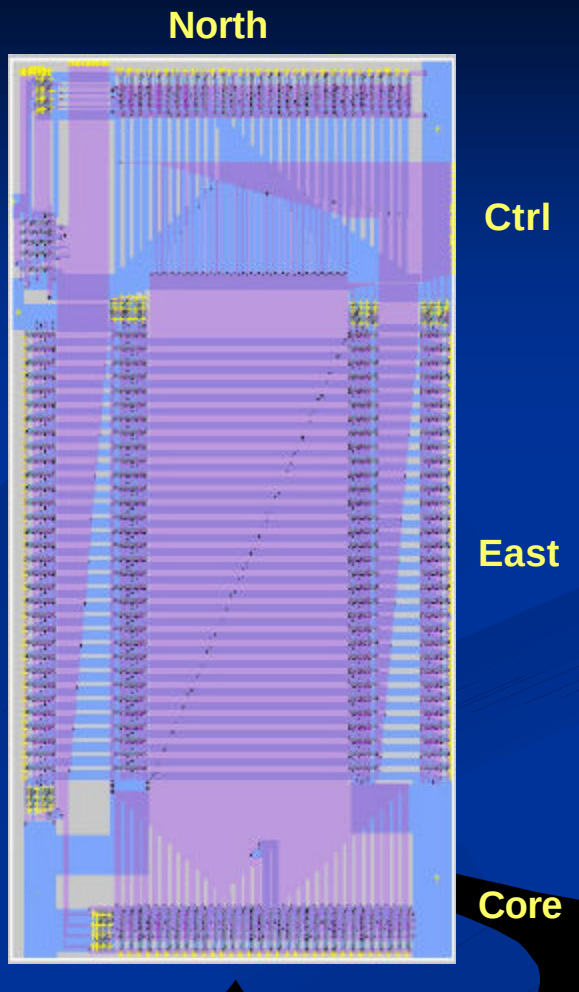
- Design philosophy
- Communication architecture
- Mapping tools / simulation environment
- Benchmark designs
- Experimental results
- Prototype layout

Design Goals / Philosophy

- ***Low-overhead*** core interface for on-chip streams
  - On-chip bus substitute for streaming applications
- Allows for interconnect of heterogeneous cores
  - Differing sizes and clock rates
- Based on static scheduling
  - Support for some dynamic events, run-time reconfiguration
- Development of complete system
  - Architecture, prototype layout, simulator, mapping tools, target applications

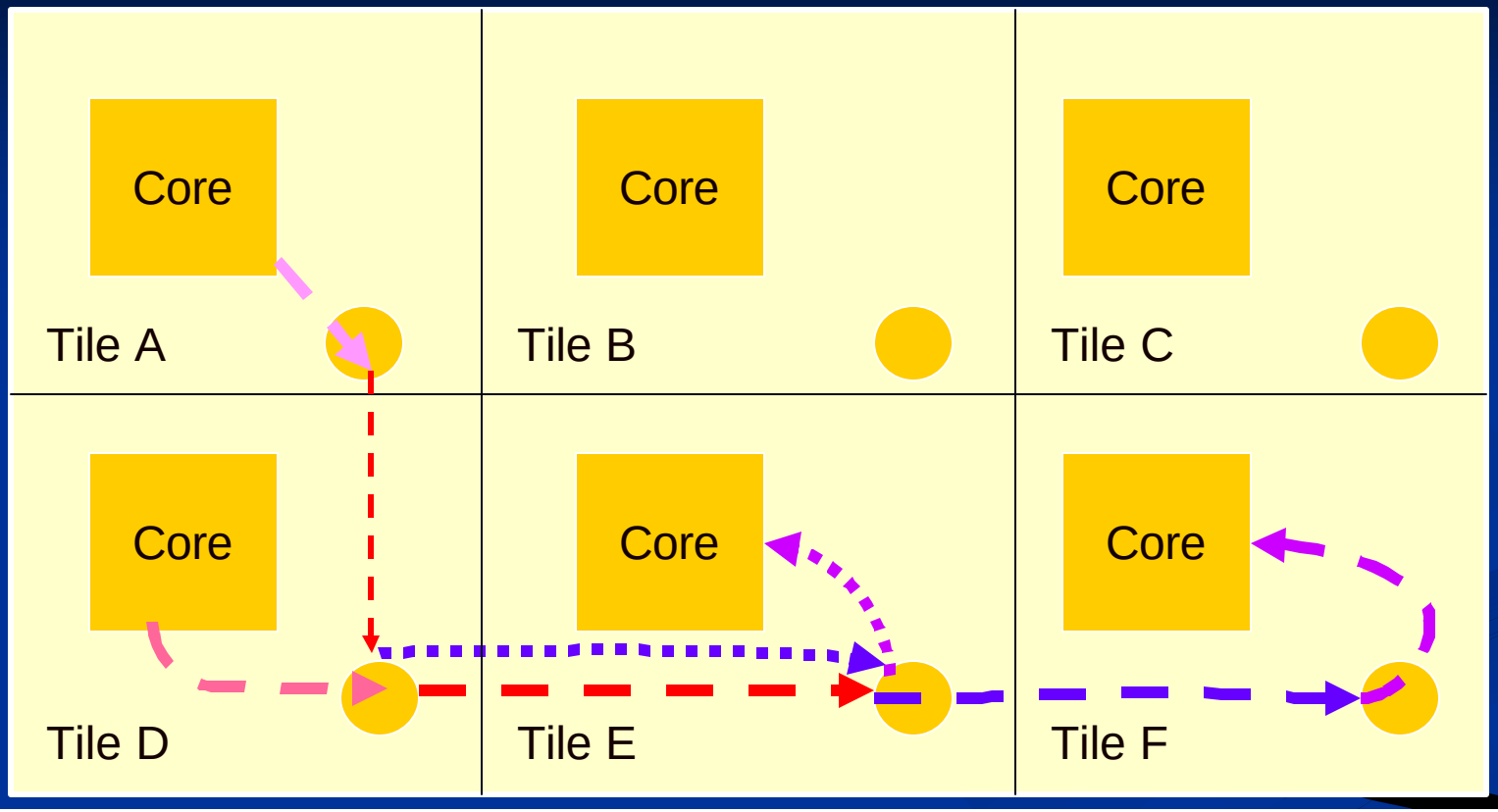
aSoC Architecture





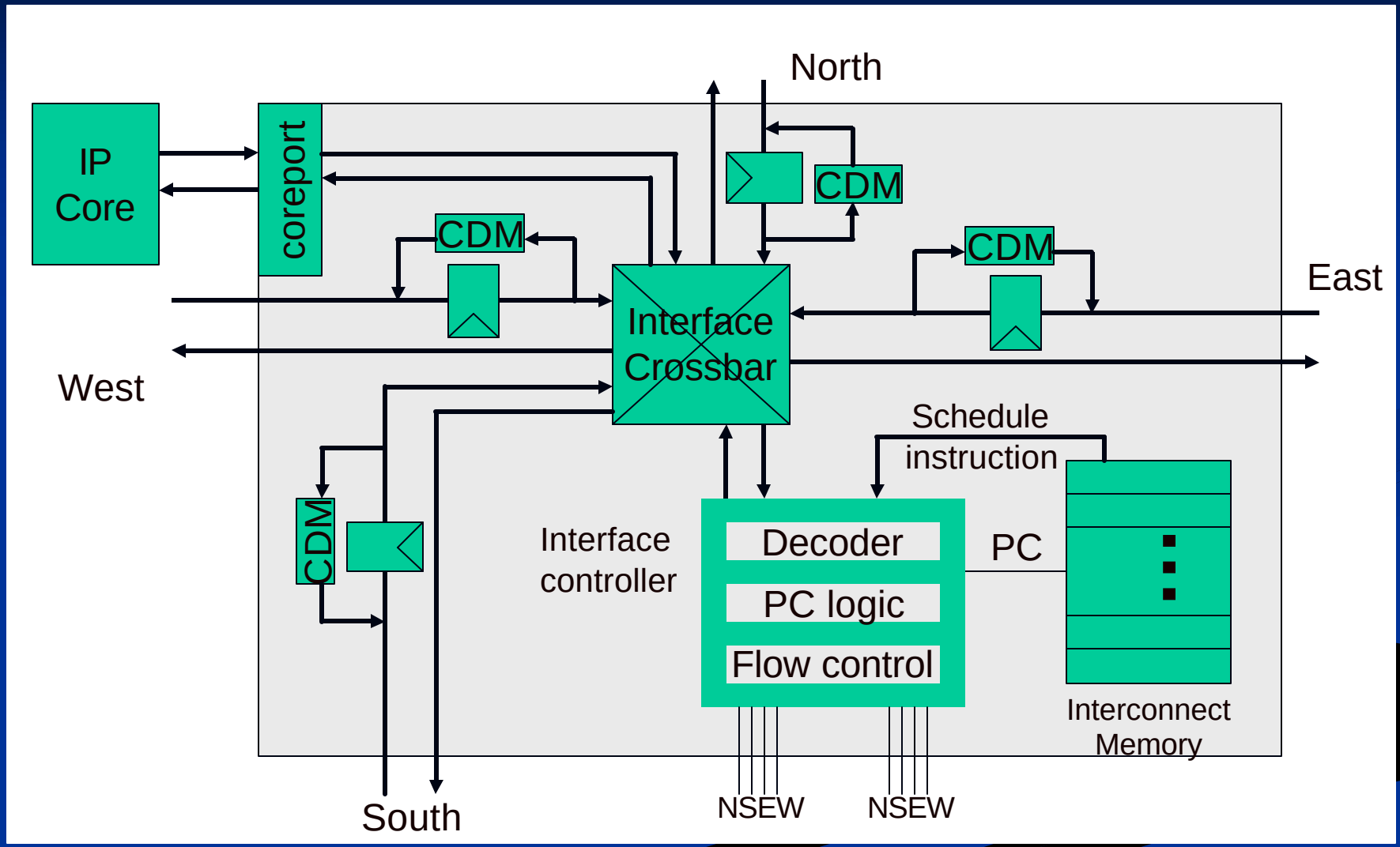
- Heterogeneous Cores
- Point-to-point connections
- **Communication Interface**

Point to Point Data Transfer



- Data transfers from tile to tile on each communication cycle
- Schedule repeats based on required communication patterns

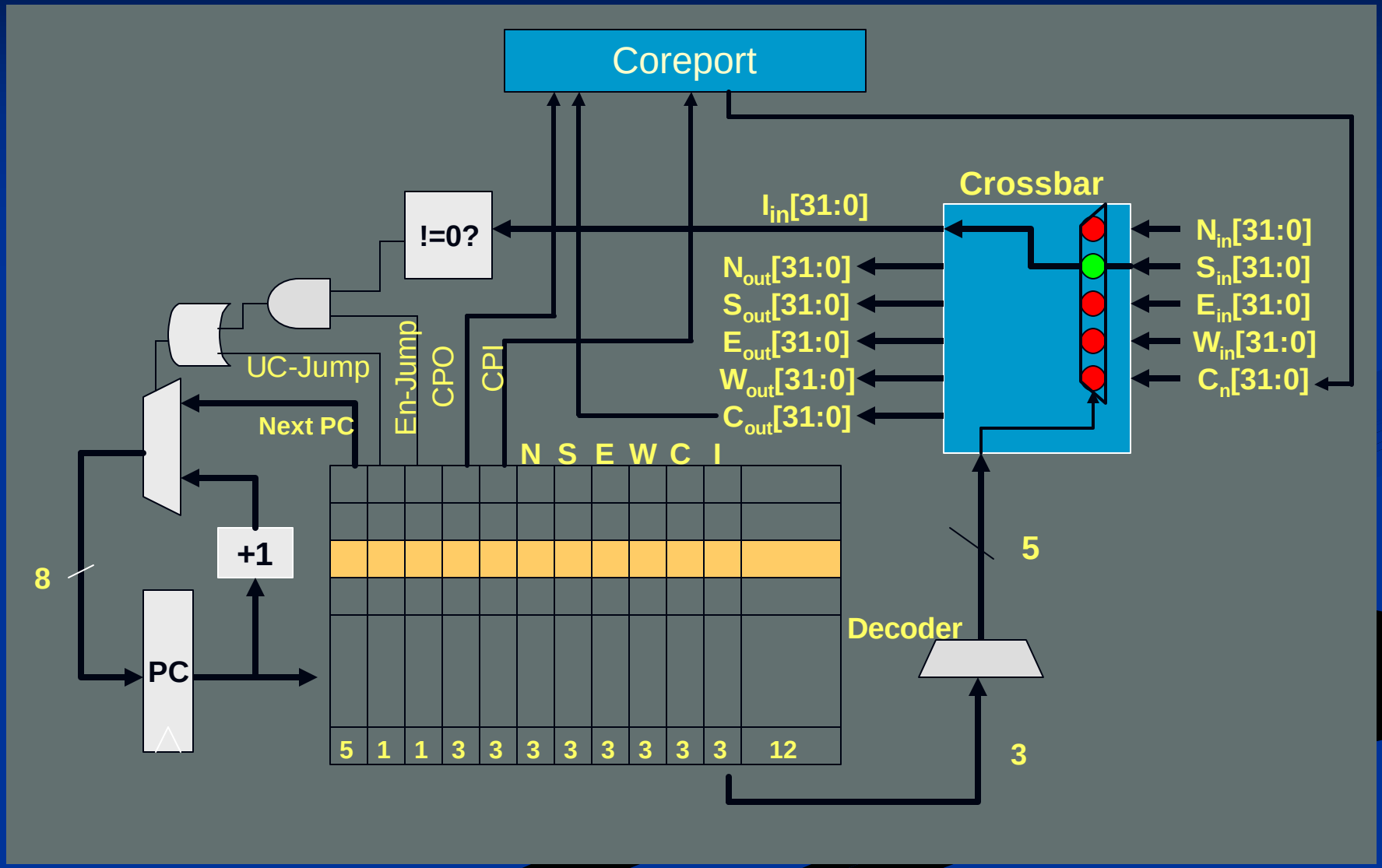
Core and Communication Interface



Communication Interface Overview

- Interconnect memory controls crossbar configuration
  - Programmable state machine
  - Allows multiple streams
- Interface controller manages flow control
  - Supports simple protocol based on single packet buffering
- Communication data memory (CDM) buffers stream data
  - Single storage location per stream
- Coreport provides storage and synchronization
  - Storage for input and output streams
  - Requires minimal support for synchronization

Interface Control Circuitry



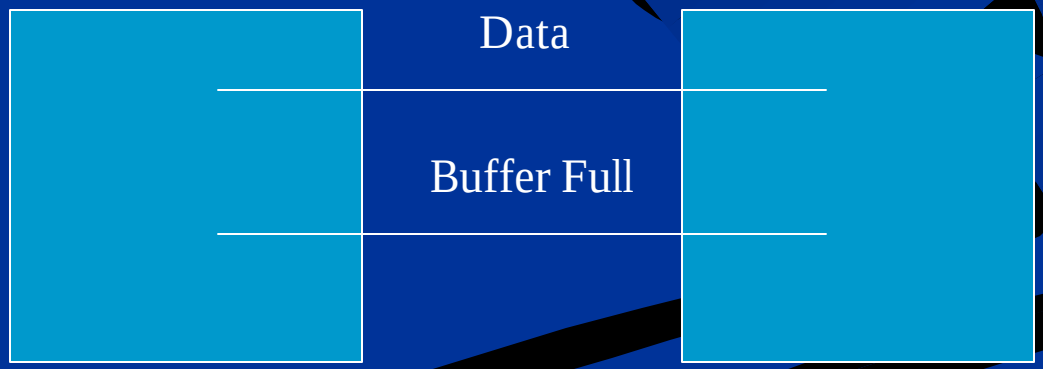
Data Dependent Stream Control

- Two types of branches
  - Unconditional branch – end of schedule reached
  - Conditional branch – test data value to modify schedule sequence
- Provides minimal support for reconfiguration
- Requires core interface support

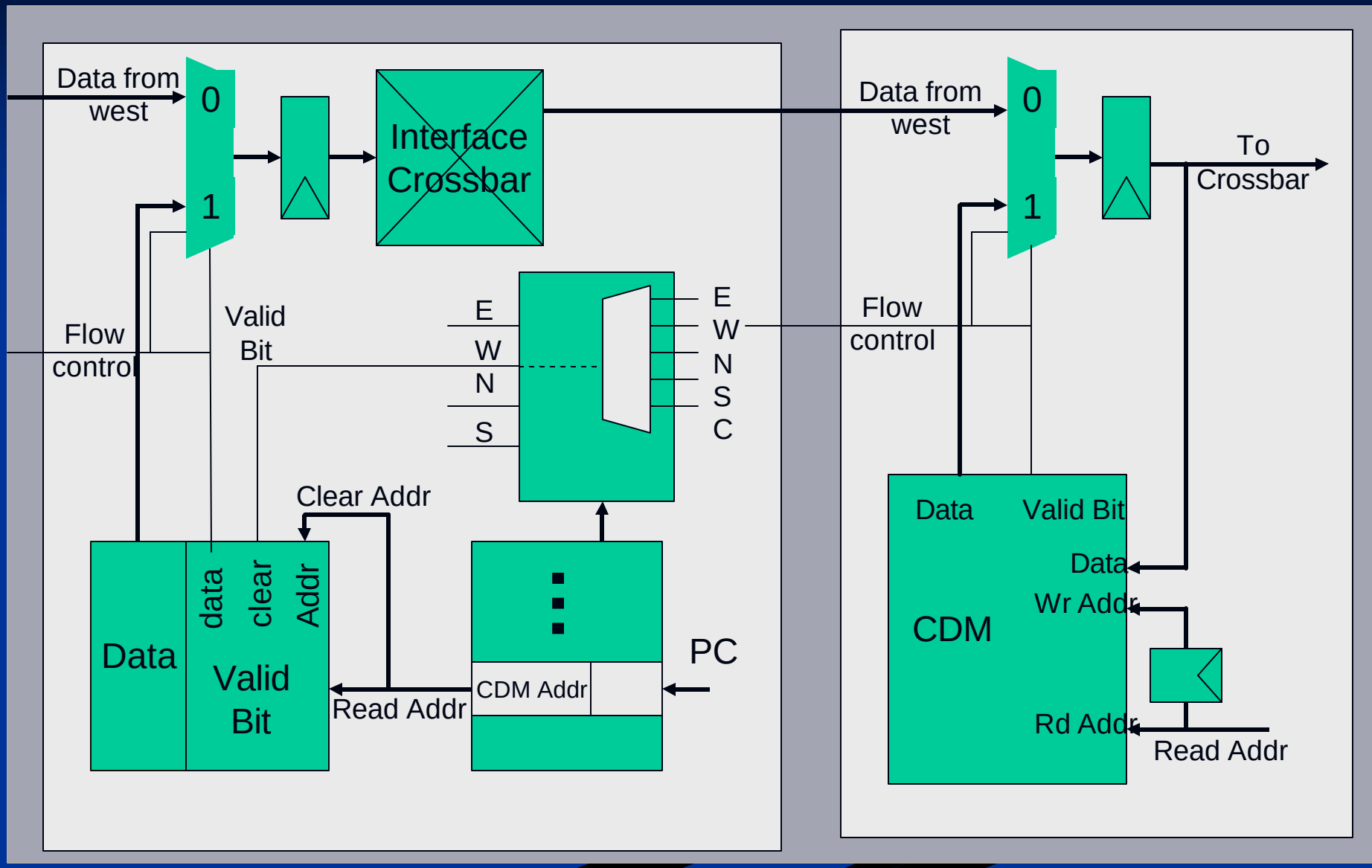
Instruction #	Required Communications	NextPC	*EnJump*	UcJump	Comment
0	coreport to south	-	0	0	*data transfer south*
1	coreport to I_{out}	0x0	1	0	*test count value*
2	coreport to east	-	0	0	*data transfer east*
3	coreport to I_{out}	0x2	1	0	*test count value*
4	coreport to west	-	0	0	*data transfer west*
5	coreport to I_{out}	0x4	1	0	*test count value*
6	go to 0x0	0x0	0	1	*restart transfer sequence*

Inter-tile Flow Control / Buffer

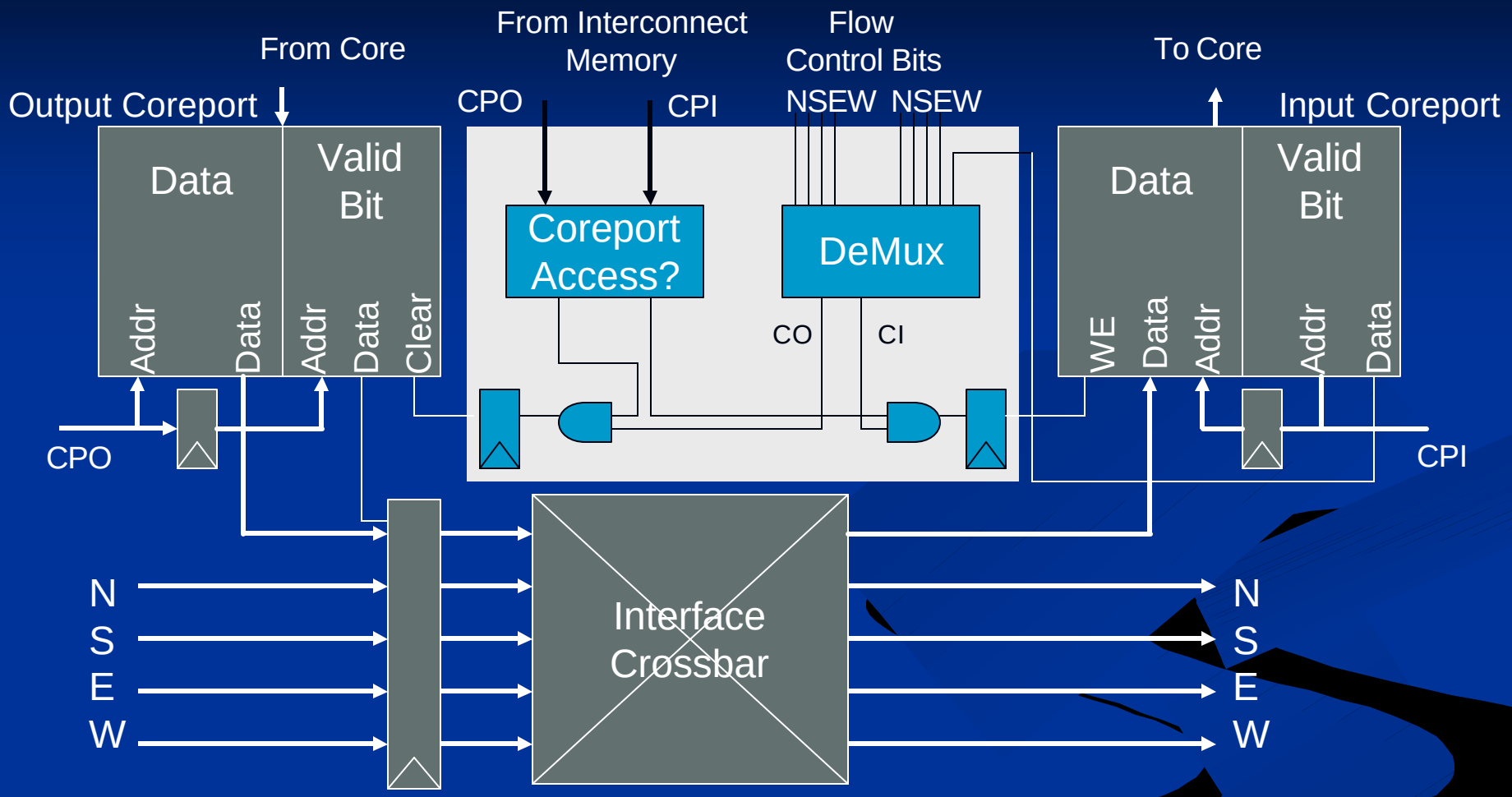
- Provide minimum amount of storage per stream at each node (1 packet)
- First priority: transfer data from storage
- Send and acknowledge simultaneously
- Can't send same stream on consecutive cycles



Inter-tile Flow Control

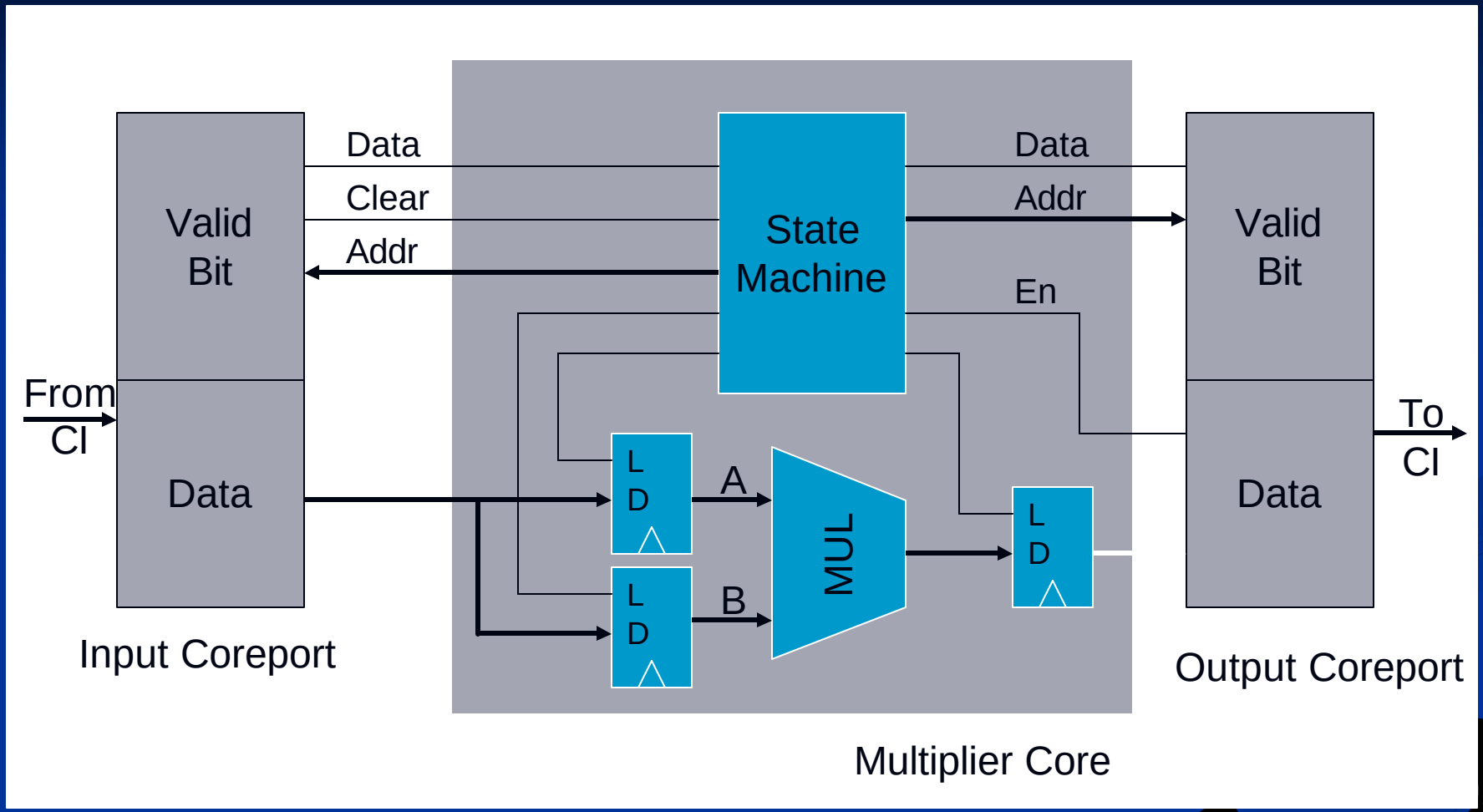


Coreport Interface to Communication



- **Data buffer provides synchronization with flow control**
- **Stream indicators (CPO, CPI) provide access to flow control bits**

Adapting the IP Core

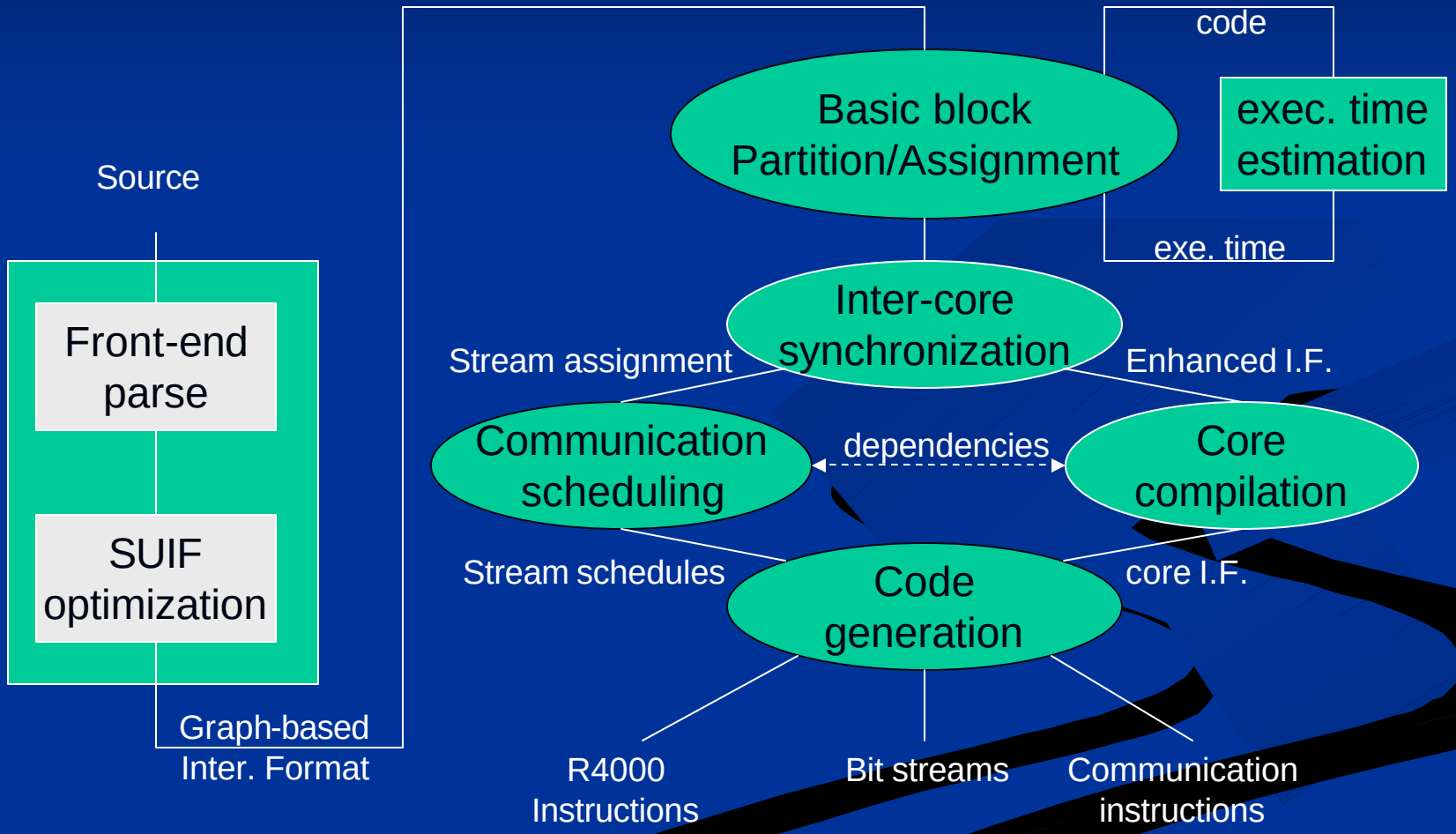


- Multiplier example
  - State machine sequencer

Design Mapping Tool Flow

- Support multiple core clock speeds and design formats
- Automate scheduling/routing
- Allow feedback between core characteristics and mapping decisions
- Generate both core and communication programming information
- Lots of room for improvement (StreamIt, HW/SW partitioning, estimators)

Design Mapping Tools



Design Mapping Tool Front End

- Current system isolates computation into basic blocks
  - Stream-oriented front-end (e.g. StreamIt) more appropriate.
- Front-end preprocessing
  - Built on SUIF
  - Performs standards optimizations
- Intermediate form used for subsequent partitioning placement, and scheduling (routing)
- User interface allows for interaction and feedback

Partitioning and Assignment

- Clustering used to collect blocks based on cost function:

$$\text{cost} = x * T_{compute} + y * \frac{1}{T_{overlap}} + z * c_{total}$$

- Cost function takes both computation and communication into account
  - $T_{compute}$ = estimate overall compute time
  - $T_{overlap}$ = estimate overall time of overlapping communication
  - c_{total} = estimate overall communication time
- Swap-based approach used to minimize cost across cores based on performance estimates.

Scheduled Routing

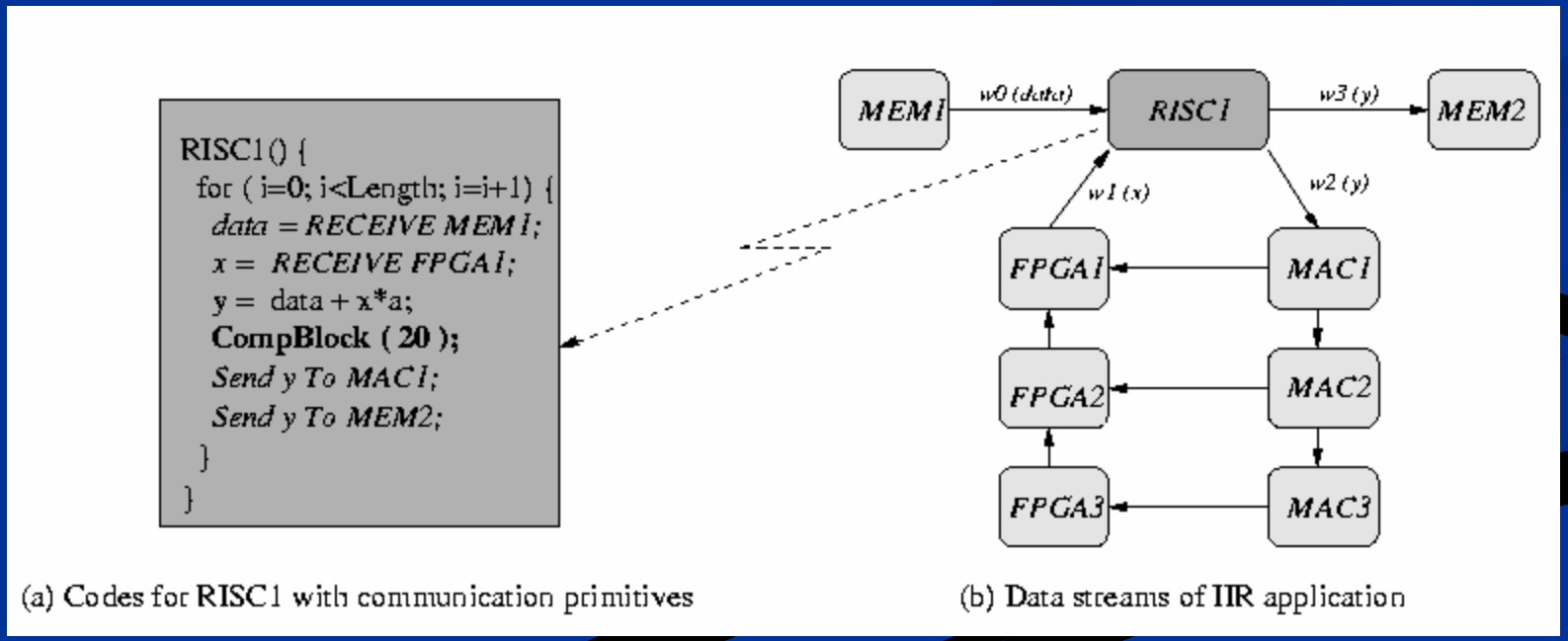
- Number and locations of streams known as a result of scheduling
- Stream paths routed as a function of required path bandwidth (channel capacity)
- Basic approach
  - Order nets by Manhattan length
  - Route streams using Prim's algorithm across time slices based on channel cost
  - Determine feasible path for all streams
  - Attempt to "fill-in" unused bandwidth in schedule with additional stream transfers

Back-end Code Generation

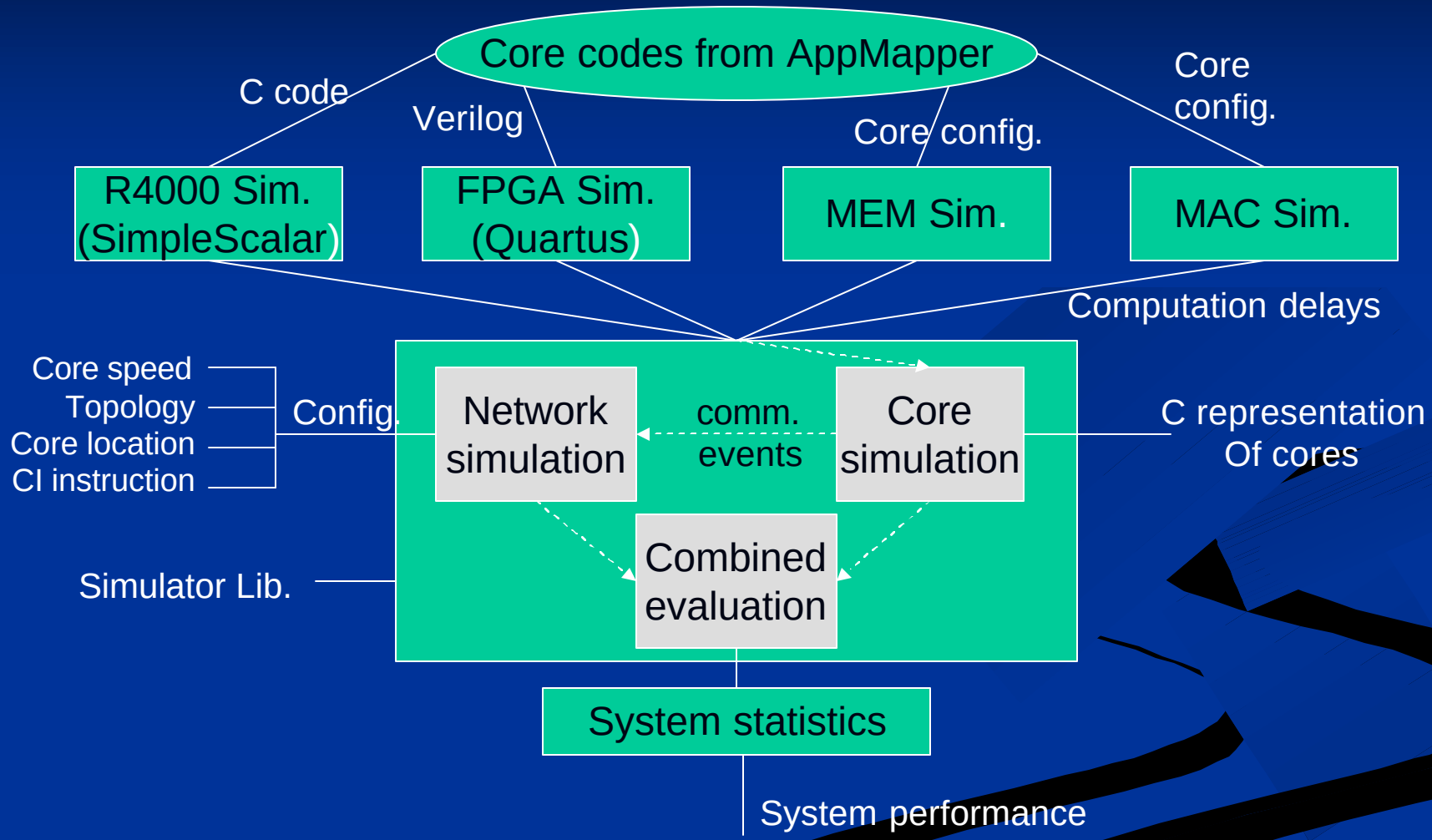
- C code targeted to R4000 cores
  - Subsequently compiled with gcc
- Verilog code for FPGA blocks
  - Synthesized with Synopsys and Altera tools
- Interconnect memory instructions for each interconnect memory
  - Limited by size of interconnect memory

Simulation Overview

- Simulation takes place in two phases
  - Core simulator determines computation cycles between interface accesses
  - Cycle accurate interconnect simulator determines data transfer between cores taking core delay into account.



Simulation Environment



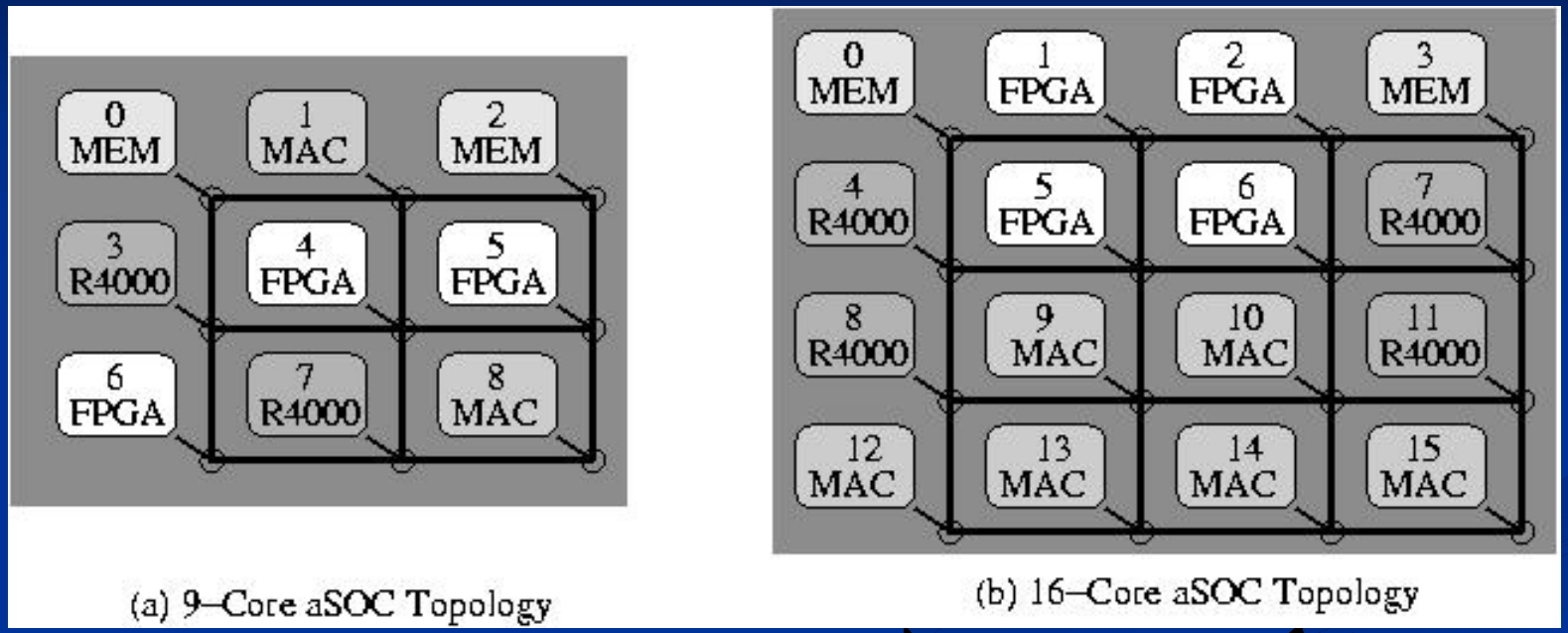
Core Simulators

- Simplescalar (D. Burger/T. Austin – U. Wisconsin)
  - Models R4000-like architecture at the cycle level
  - Breakpoints used to calculate cycle counts between communication network interaction
- Cadence Verilog –XL
  - Used to model 484 LUT FPGA block designs
  - Modeled at RTL and LUT level
- Custom C simulation
  - Cycle counts generated for memory and multiply accumulate blocks
- Simulators invoked via scripts

Interconnect Simulator

- Based on NSIM (MIT NuMesh Simulator – C. Metcalf)
- Each tile modeled as a separate process
- Interconnect memory instructions used to control cycle-by-cycle operation
- Core speeds and flow control circuitry modeled accurately.
- Adapted for a series of on-chip interconnect architectures (bus-based architectures)

Target Architectural Models

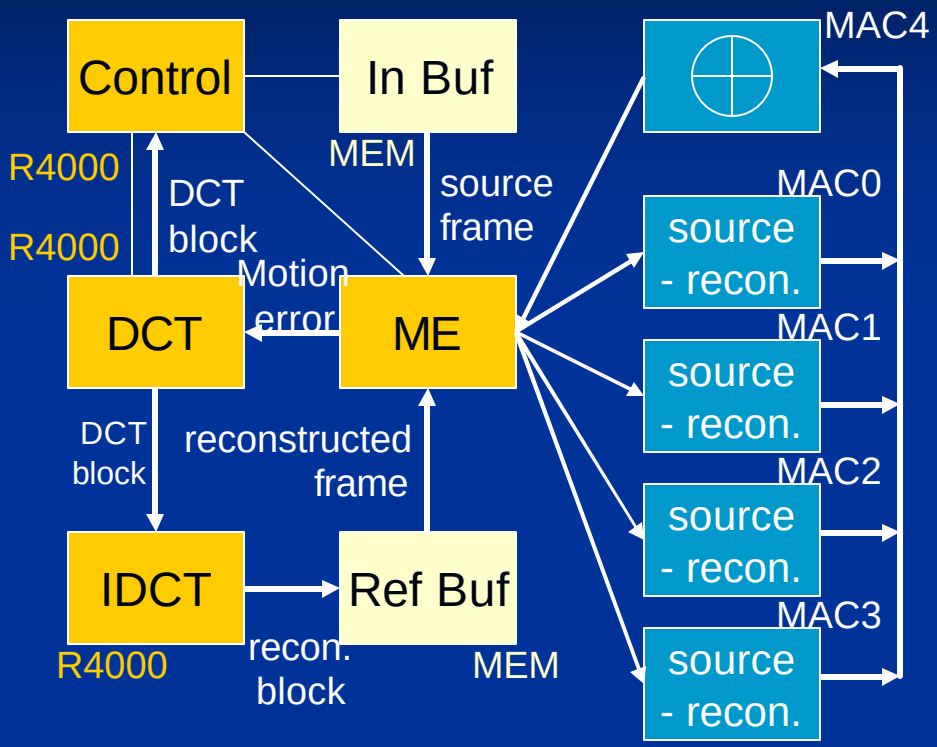


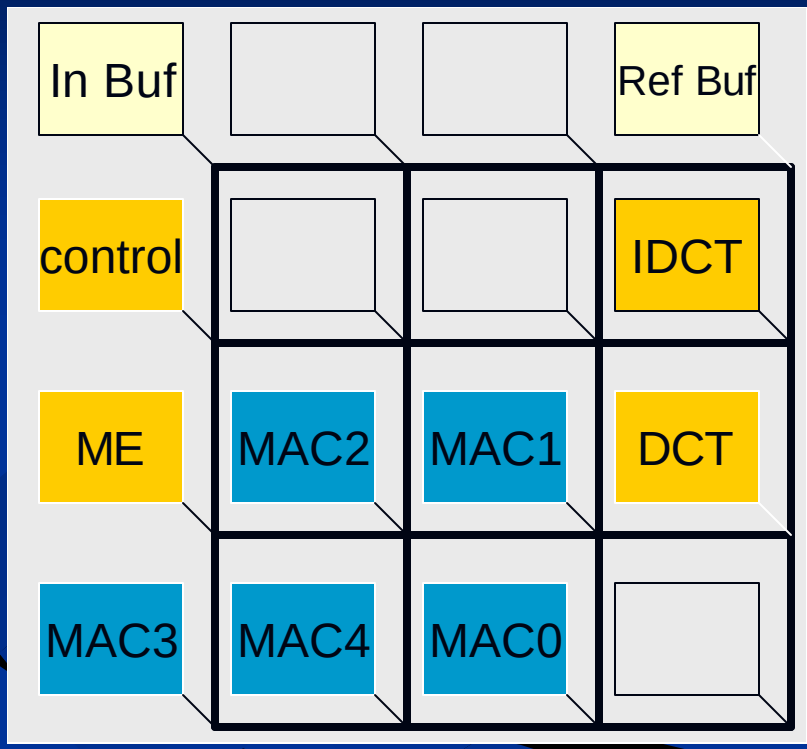
(a) 9–Core aSOC Topology

(b) 16–Core aSOC Topology

- FPGA blocks contain 121 4-LUT clusters
- Custom MAC and 32Kx8 SRAM (Mem) blocks
- Same configurations used for all benchmarks

Example: MPEG-2





- Design partitioned across eleven cores
- Other applications: IIR filter, image processing, FFT

Core Parameters

	Speed	Area ($?^2$)
Comm. Interface	2.5 ns	2500 x 3500
MIPs R4000	5 ns	4.3×10^7 **
MAC	5 ns	1500 x 1000
FPGA	10 ns	30000 x 30000
MEM (32Kx8)	5 ns	10000x 10000

- Communication interface, MAC, FPGA, and MEM sizes determined through layout (TSMC 0.18um)
- ** R4000 size from MIPs web page

Mapping Statistics

Design	No. Cores	No. Streams	Max CI Instruct.	Max Streams Per CI	Max CPort Mem. Depth
IIR	9	11	2	5	5
IIR	16	20	2	5	5
IMG	9	8	2	3	3
IMG	16	15	4	4	4
FFT	16	25	6	7	7
MPEG	16	19	4	8	8

- Number of Interconnect Mem instructions (CI Instruct) deceptively small
- Likely need to better fold streams in schedule

Comparison to IBM CoreConnect

	9 Core Model		16 Core Model			
Execution Time (ms)	IIR	IMG	IIR	IMG	FFT	MPEG
R4000	0.049	327.0	0.350	327.0	0.79	152
CoreConnect	0.012	22.0	0.016	30.5	0.12	173
Coreconnect (burst)	0.012	18.9	0.015	24.3	0.12	172
aSoC	0.006	9.6	0.006	7.3	0.09	83
aSoC Speed-up vs. burst	2.0	2.3	2.5	3.3	1.3	2.1
Used aSoC Links	8	8	33	27	41	26
aSoC max. link usage	10%	8%	37%	28%	2%	25%
aSoC ave. link usage	7%	7%	22%	25%	2%	5%
CoreConnect busy (burst)	91%	100%	100%	99%	32%	67%

- Still work to do on mapping environment to boost aSoC link utilization

Comparison to Hierarchical CoreConnect

	9-core Model		16-Core Model			
Execution Time *(ms)*	IIR	IMG	IIR	IMG	FFT	MPEG
Hier CoreConnect	0.013	26.0	15.7	37.4	0.15	178
aSoC	0.006	9.6	7.0	7.3	0.09	83
aSoC speedup	2.1	2.7	2.2	5.1	1.6	2.2

- Multiple levels of arbitration slows down hierarchical CoreConnect

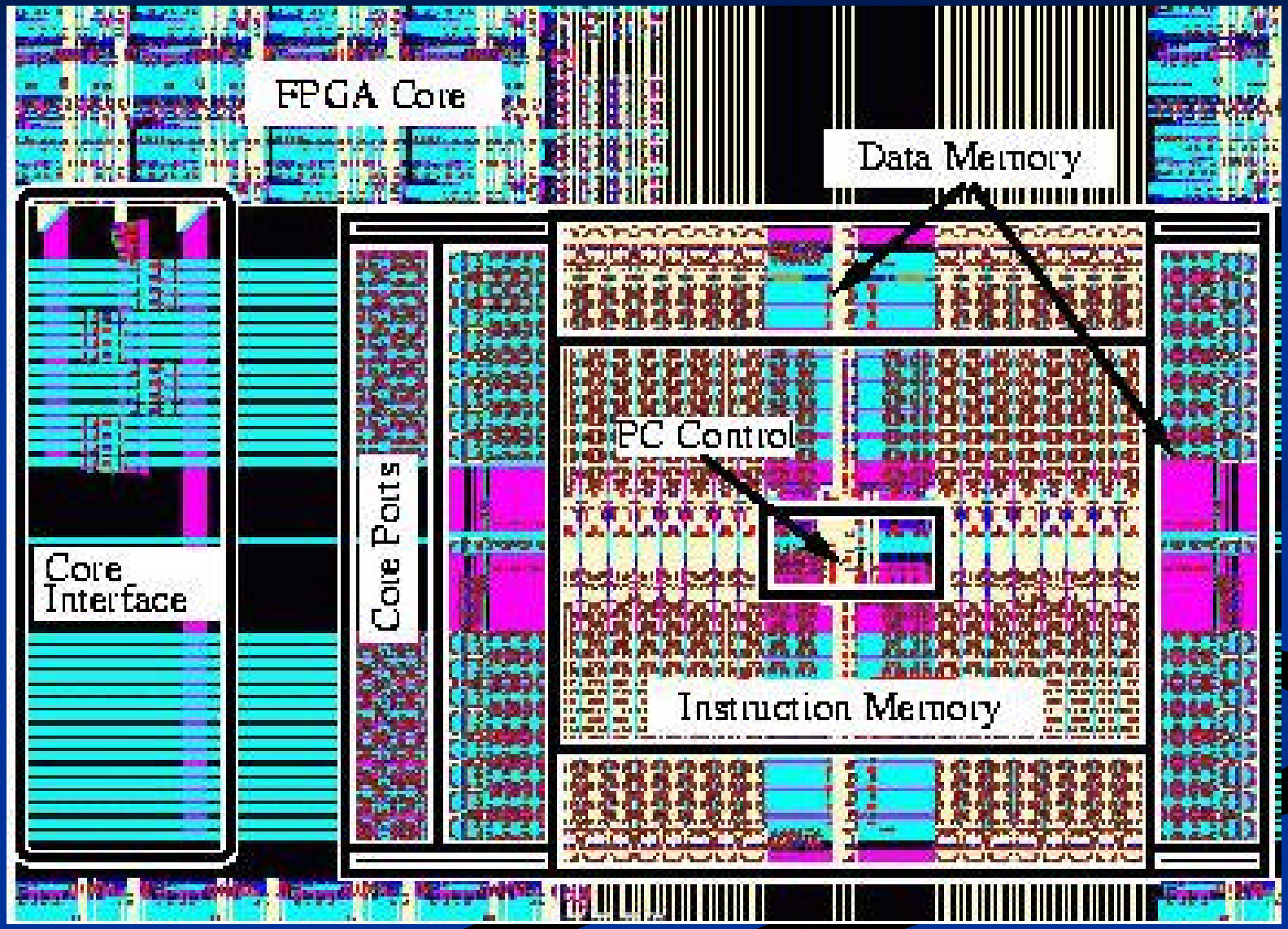
aSoC Comparison to Dynamic Network

- Direct comparison to oblivious routing network [1]

	9-Core Model		16 Core Model		
Execution Time (ms)	IIR	IMG	IIR	IMG	MPEG
Dynamic Routing	0.008	14.4	8.7	9.7	162.0
aSoC	0.006	6.1	7.0	7.3	82.5
aSoC Speedup	1.3	2.4	1.3	1.3	2.0

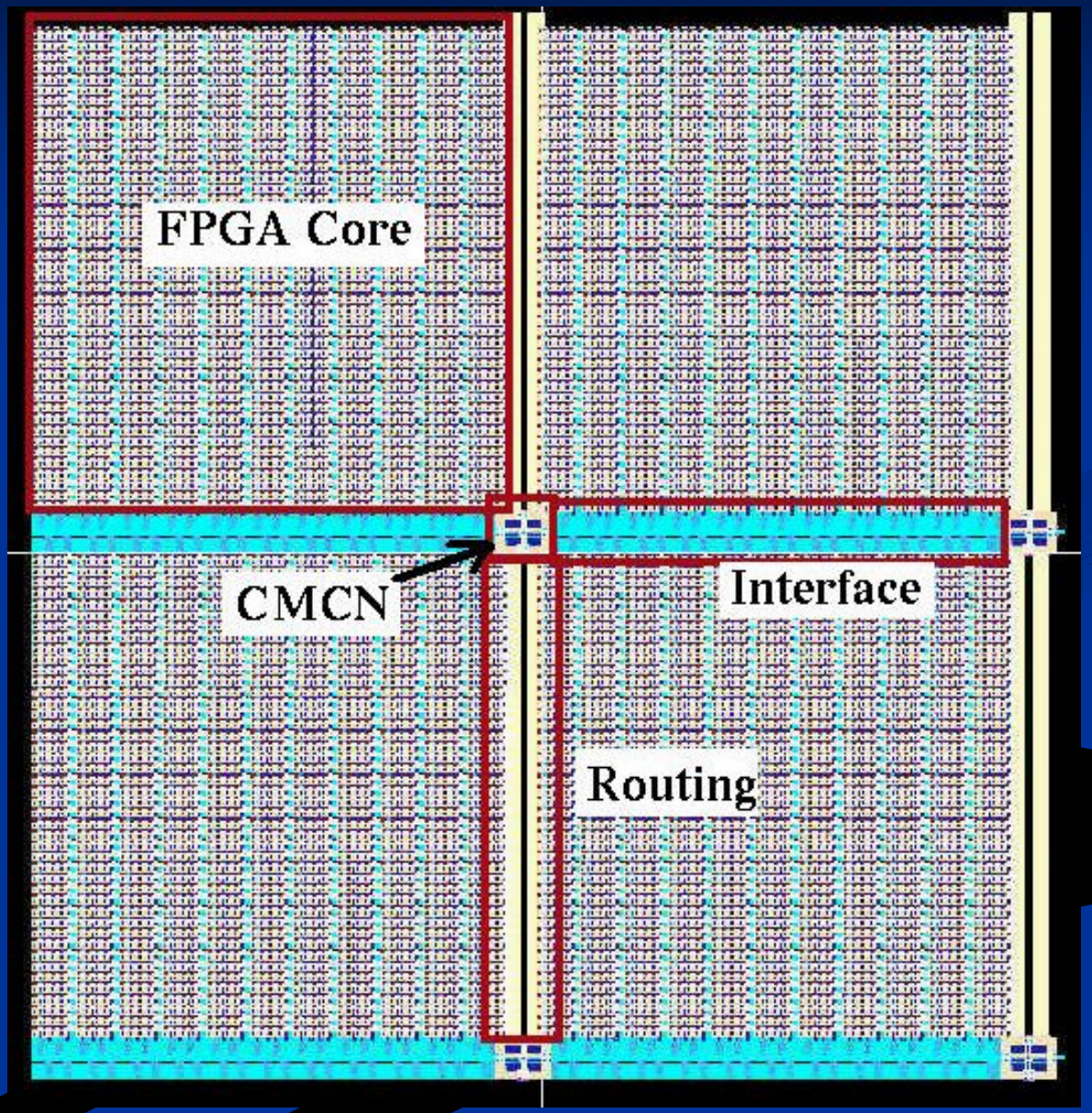
1. W. Dally and H. Aoki, “Deadlock-free Adaptive Routing in Multi-computer Networks Using Virtual Routing”, IEEE Transactions on Parallel and Distributed Systems, April 1993

aSoC Layout


FPGA Core
Data Memory
Core Interface
Core Ports
PC Control
Instruction Memory

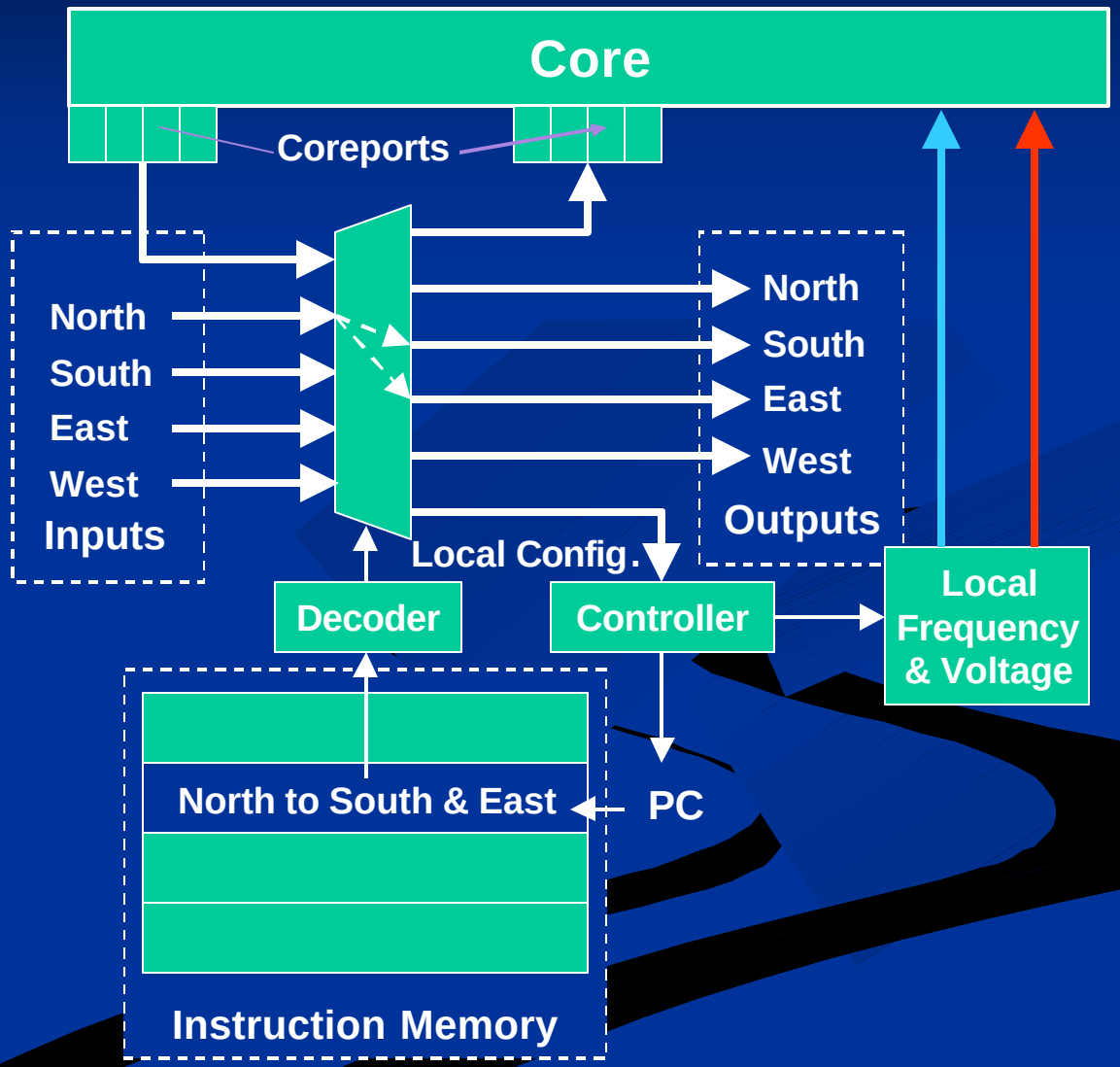
aSoC Multi-core Layout

- Comm. Interface consumes about 6% of tile
- Critical path in flow control between tiles
- Currently integrating additional cores



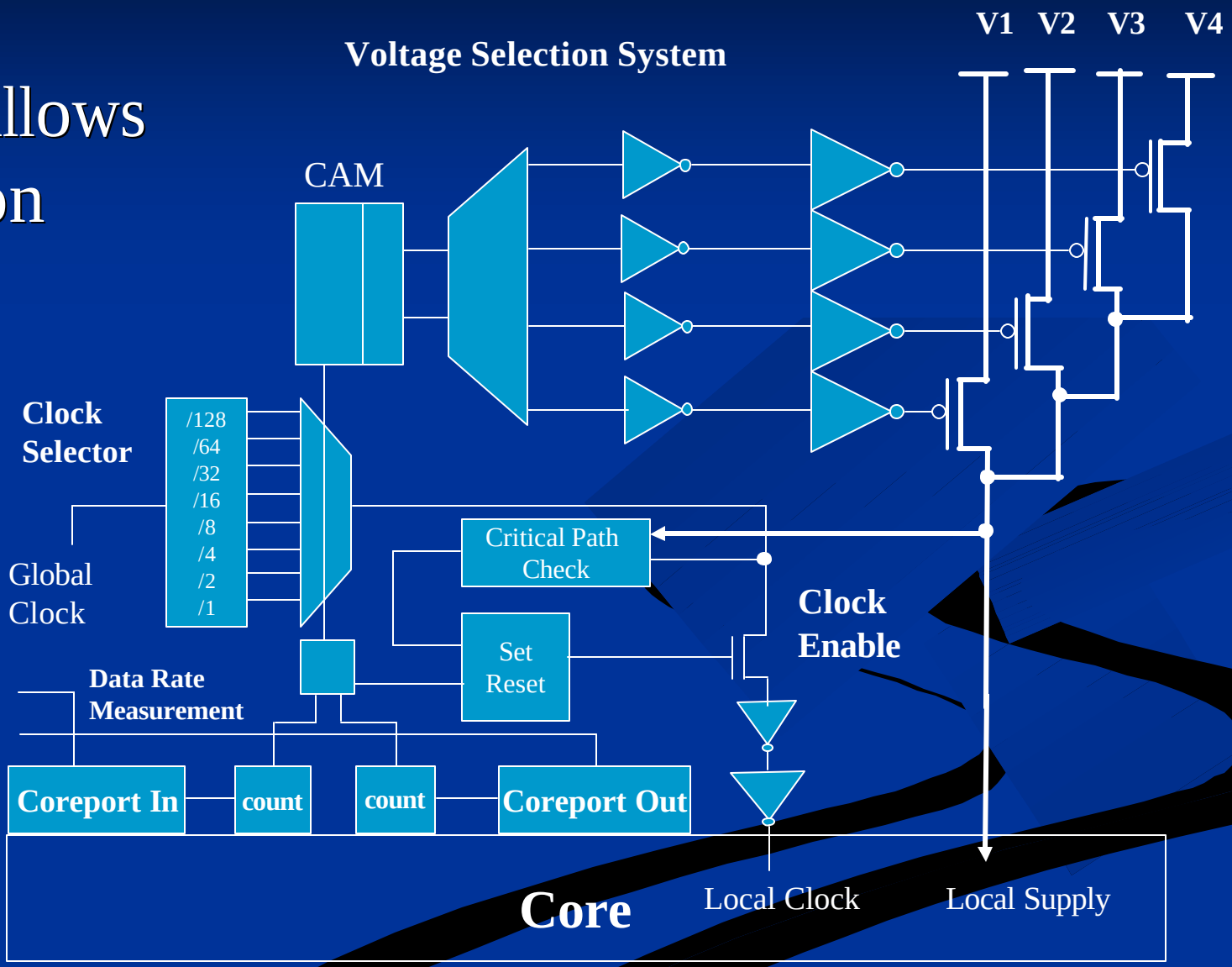
Future Work: Dynamic Voltage Scaling

- Data transfer rate to/from core used to control voltage and clock
- Counter and CAM used to select sources
- May be software controlled



Future Work: Dynamic Voltage Scaling

- CAM allows selection



Future Work

- Improved software mapping environment
- Integration of more cores
- Mapping of substantial applications
  - Turbo codes
  - Viterbi decoder
- More integrated simulation environment

Summary

- Goal: Create low-overhead interconnect environment for on-chip stream communication
- IP core augmented with communication interface
- Flow control and some stream reconfiguration included in the architecture
- Mapping tools and simulation environment assist in evaluating design
- Initial results show favorable comparison to bus and high-overhead dynamic networks.