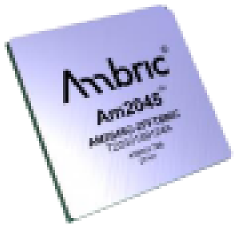


A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

Mike Butts

mike@ambric.com

Ambric Introduction

- Fabless Semiconductor Company
- Founded in 2003 in Beaverton, Oregon
- Veteran team – 60+ employees and growing
- Production silicon – August 2007
- Product releases: Chip, IDE, applications, board – January 2008

Ambric Objectives

- Maximum possible performance and performance/watt for embedded and accelerated applications
  - streaming media, image processing, networking, software radio
  - superior to FPGAs, DSPs, multicores, even approaching ASICs
- Reasonable and reliable application development
  - write software not hardware, with reliable reuse
- Hardware and software scalability to track Moore's Law
  - future silicon processes
  - development productivity

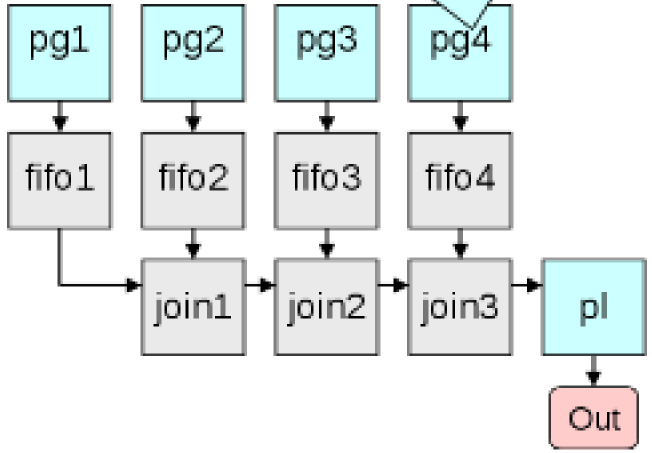
A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

What’s in a Good Programming Model?

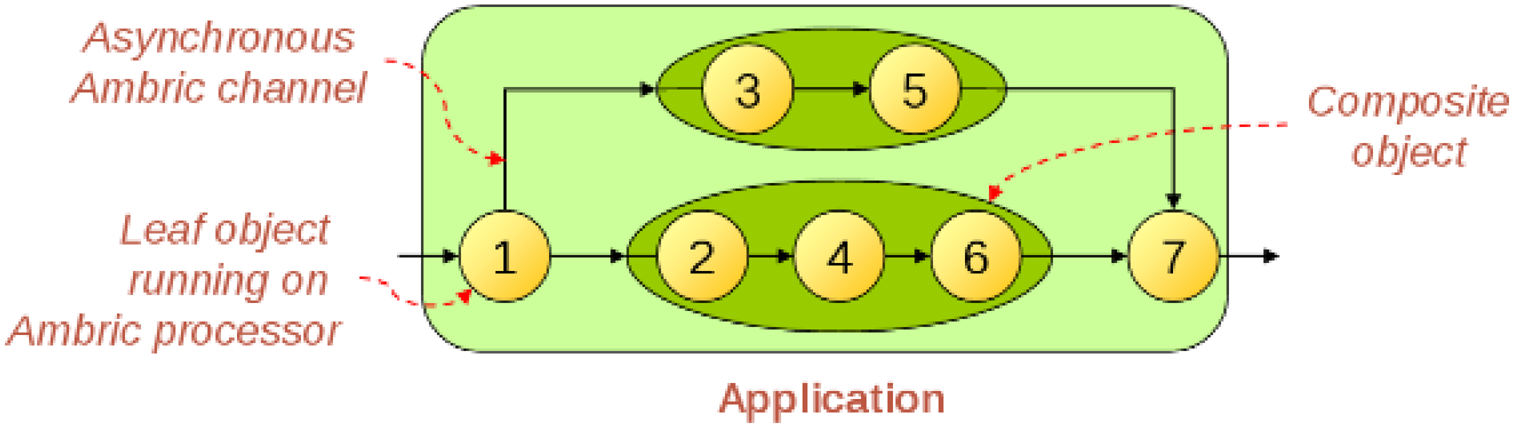
- What should be in a programming model?
  - What is familiar, productive, scalable to any size and speed?
- Software languages (C, Java, ...) are familiar and productive
  - Array of sequential processors
- Block diagrams are familiar, scalable, encapsulated and hierarchical
  - Reconfigurable interconnect
- Strict encapsulation and hierarchy with standard interfaces enables strong design reuse, necessary for scalable development cost.

```
for (int c = min; c <= max; c += 2*inc) {
    int fac;
    for (fac = 3; fac <= max; fac += 2) {
          if (c % fac == 0) break; }
    if (c == fac) {
          primes.write(c);  }
    else primes.write(0);
}
```

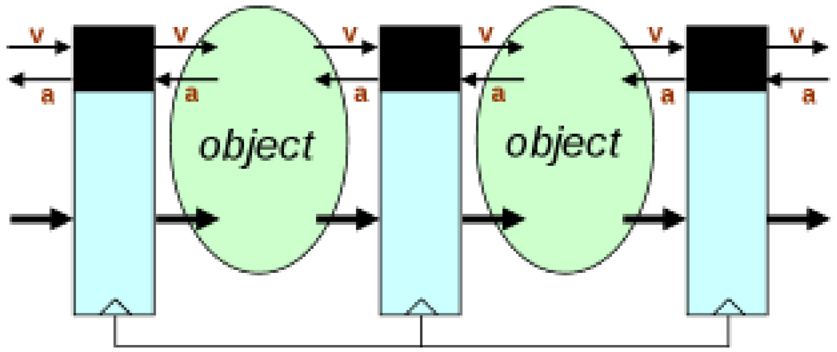


Structural Object Programming Model

- Objects are software programs running concurrently on an asynchronous array of Ambric processors and memories
- Objects exchange data and control through a structure of self-synchronizing asynchronous Ambric channels
- Objects are mixed and matched hierarchically to create new objects, snapped together through a simple common interface
- Easier development, high performance and scalability

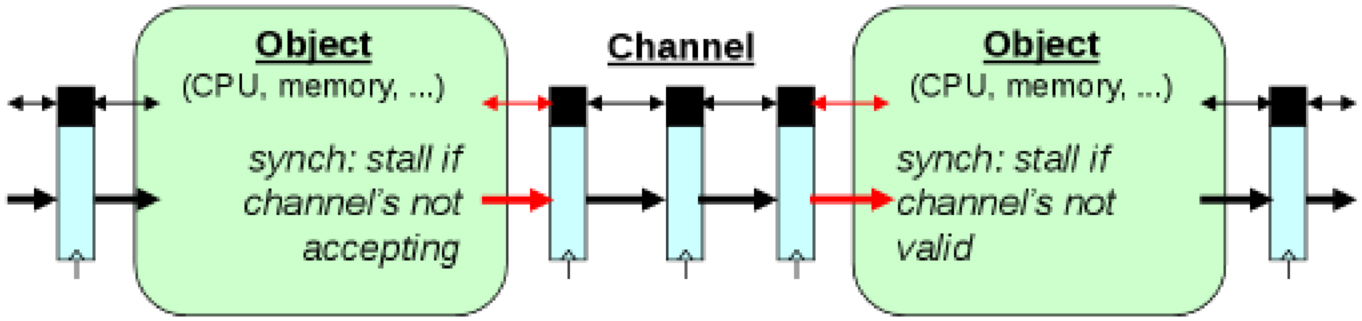

Asynchronous Ambric channel
Leaf object running on Ambric processor
Composite object
1
2
3
4
5
6
7
Application

Ambric Channels



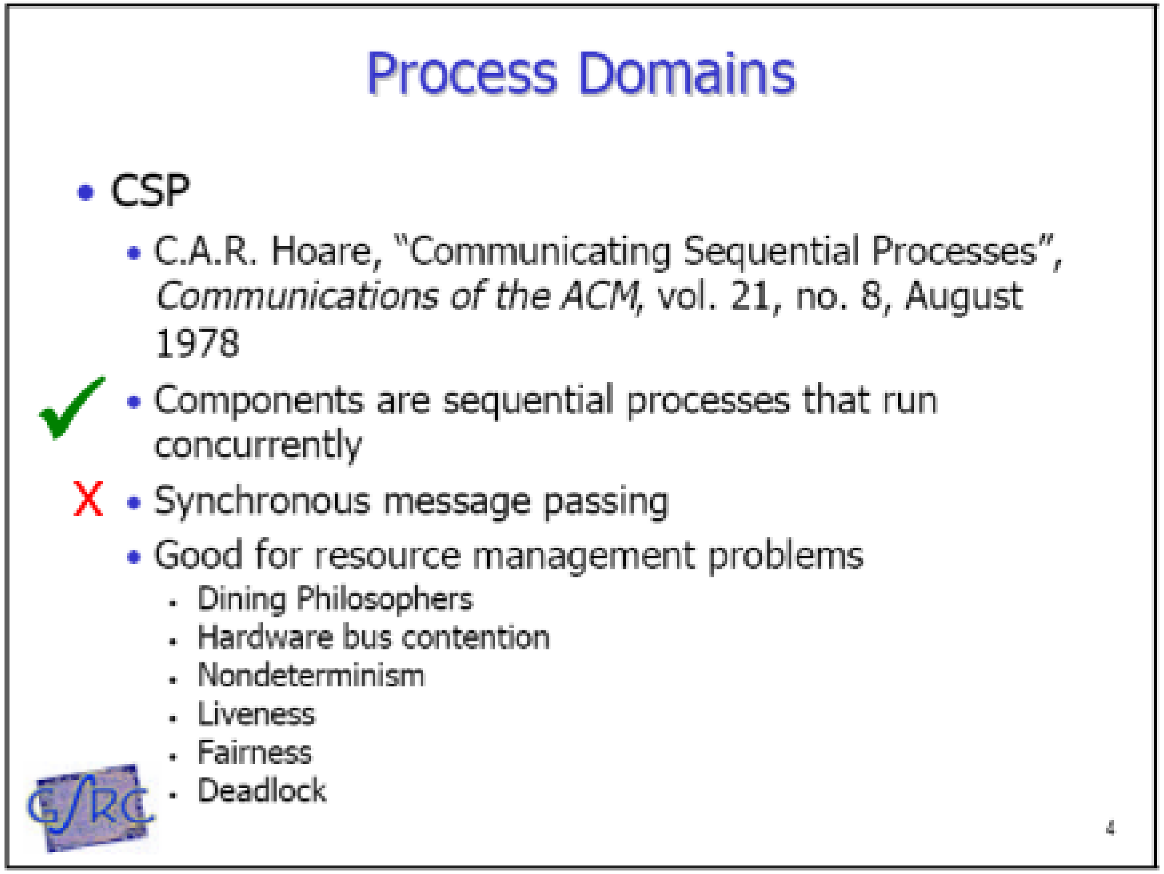
- Chains of Ambric registers form Ambric channels
  - Word-wide, unidirectional, point-to-point, strictly ordered
  - Inter-stage object throttles its channels with Ambric protocol
    - *v* downstream, *a* upstream
  - Fully encapsulated, fully scalable for control and data between objects
- Objects linked through channels are asynchronous to each other
  - Each operates when it can, *on its own*, according to its channels
  - Objects are synchronized with one another *only* through channels
- Globally Asynchronous Local Synchronous (GALS) clocking
  - Physically scalable, no low-skew long wires

SOPM Realized in Silicon


Object
(CPU, memory, ...)
synch: stall if channel's not accepting
Channel
Object
(CPU, memory, ...)
synch: stall if channel's not valid

- Objects exchange data and control thru a structure of Ambric channels
  - Each stage has forward and backward flow control, and buffering
- Standard interface between objects
  - Encapsulation, reuse
- Self-synchronizing on each transfer
  - Asynchronous system
- Channels can be any length or speed: no scheduling, no timing closure
  - Easier on tools, easier to program, easier to debug, reliable

Model of Computation: not quite CSP



Process Domains

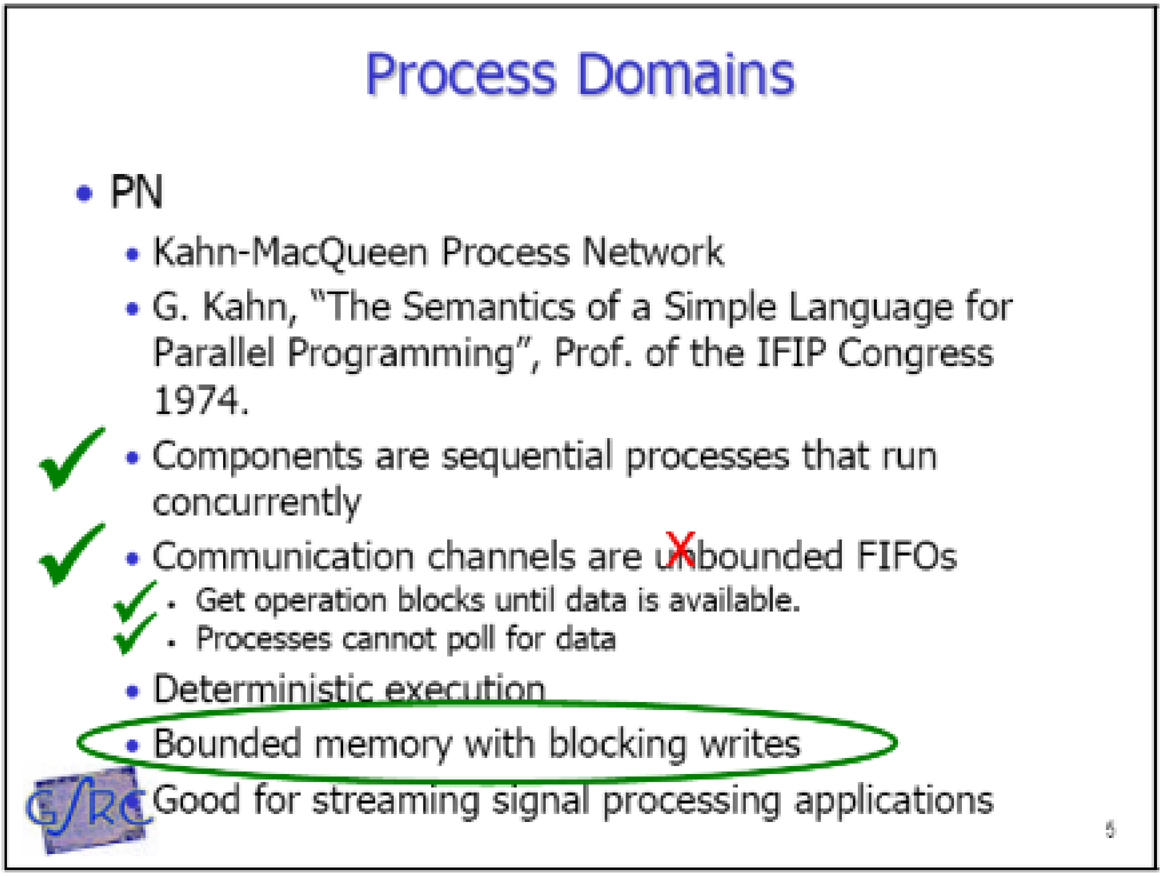
- CSP
  - C.A.R. Hoare, "Communicating Sequential Processes", *Communications of the ACM*, vol. 21, no. 8, August 1978
  - ✓ Components are sequential processes that run concurrently
  - X Synchronous message passing
  - Good for resource management problems
    - Dining Philosophers
    - Hardware bus contention
    - Nondeterminism
    - Liveness
    - Fairness
    - Deadlock

4

slides from "A Brief Tutorial on Models of Computation"
The MESCAL Team, UC Berkeley, Fall 2001

- Ambric MoC was inspired by CSP, but is not quite CSP.
  - Message passing is buffered, not strictly synchronous.

Model of Computation: Process Network


Process Domains

- PN
  - Kahn-MacQueen Process Network
  - G. Kahn, "The Semantics of a Simple Language for Parallel Programming", Prof. of the IFIP Congress 1974.
  - ✓ Components are sequential processes that run concurrently
  - ✓ Communication channels are ~~un~~bounded FIFOs
    - ✓ Get operation blocks until data is available.
    - ✓ Processes cannot poll for data
  - Deterministic execution
  - Bounded memory with blocking writes
  - Good for streaming signal processing applications

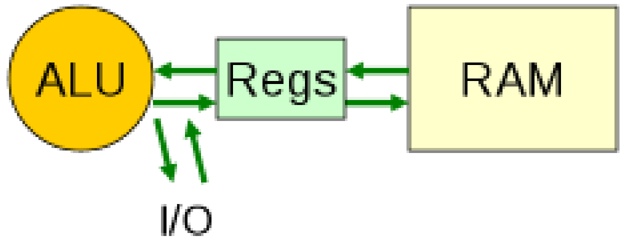
5

slides from "A Brief Tutorial on Models of Computation"
The MESCAL Team, UC Berkeley, Fall 2001

- Ambric MoC is a Process Network with bounded FIFOs.
  - FIFO-like primitive register, streaming RAMs for bigger FIFOs.
  - Channels carry data and control, and strictly preserve sequence.

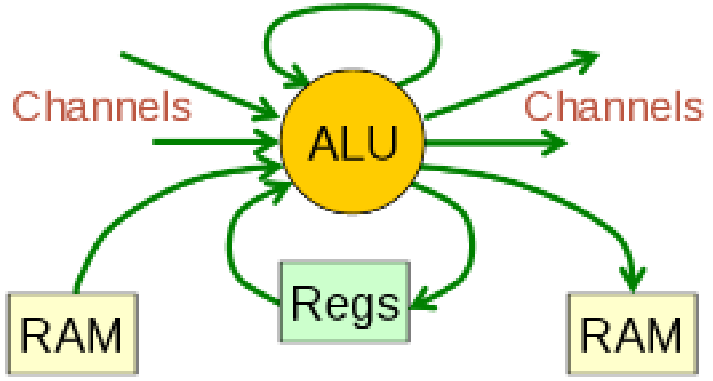
Traditional vs. Ambric Processors

- **Traditional processor architecture**
  - Primary: register-memory hierarchy
  - Secondary: communication



- **Ambric processor architecture**
  - Primary: communicate through channels
  - All data goes through channels
    - Memory
    - Registers
    - Inter-processor streams
    - Instruction streams

    to reduce local storage
  - Channels synchronize all events

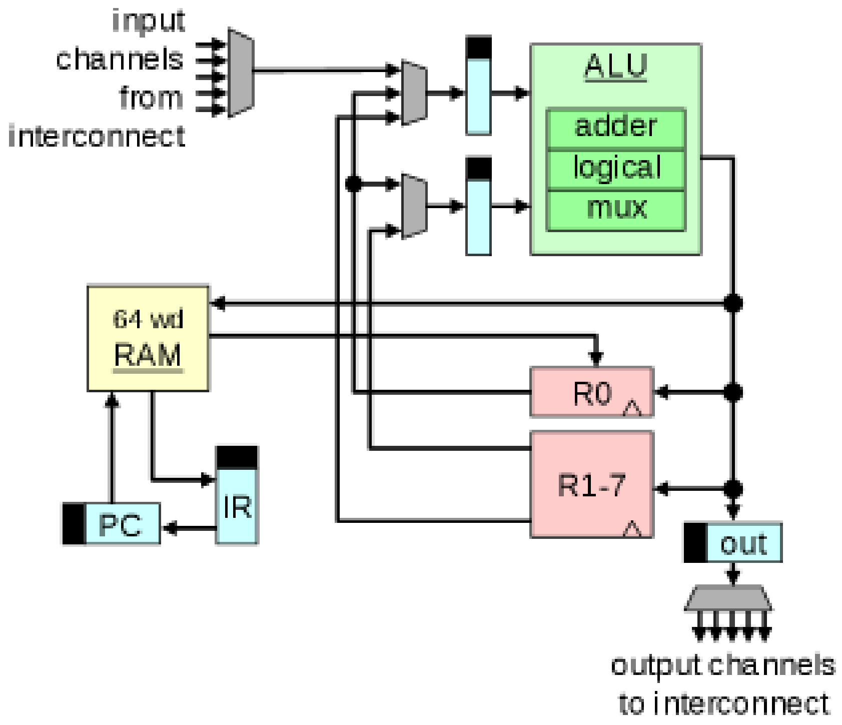


A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

Ambric SR Processor

- Simple 32-bit Streaming RISC
- Mainly for fast small utility objects:
  - complex addressing, complex fork/join, pack/unpack, serialize/deserialize
- Ambric channels
  - 1 input, 1 output per instruction
  - Instruction fields select inputs, outputs just like selecting registers
- One ALU: 32b or dual 16b ops
- 8 general registers
- 16 bit instructions for code density
  - Zero-overhead looping
- 64 word local code/data RAM
  - 128 instructions
- Three-stage Ambric channel datapath



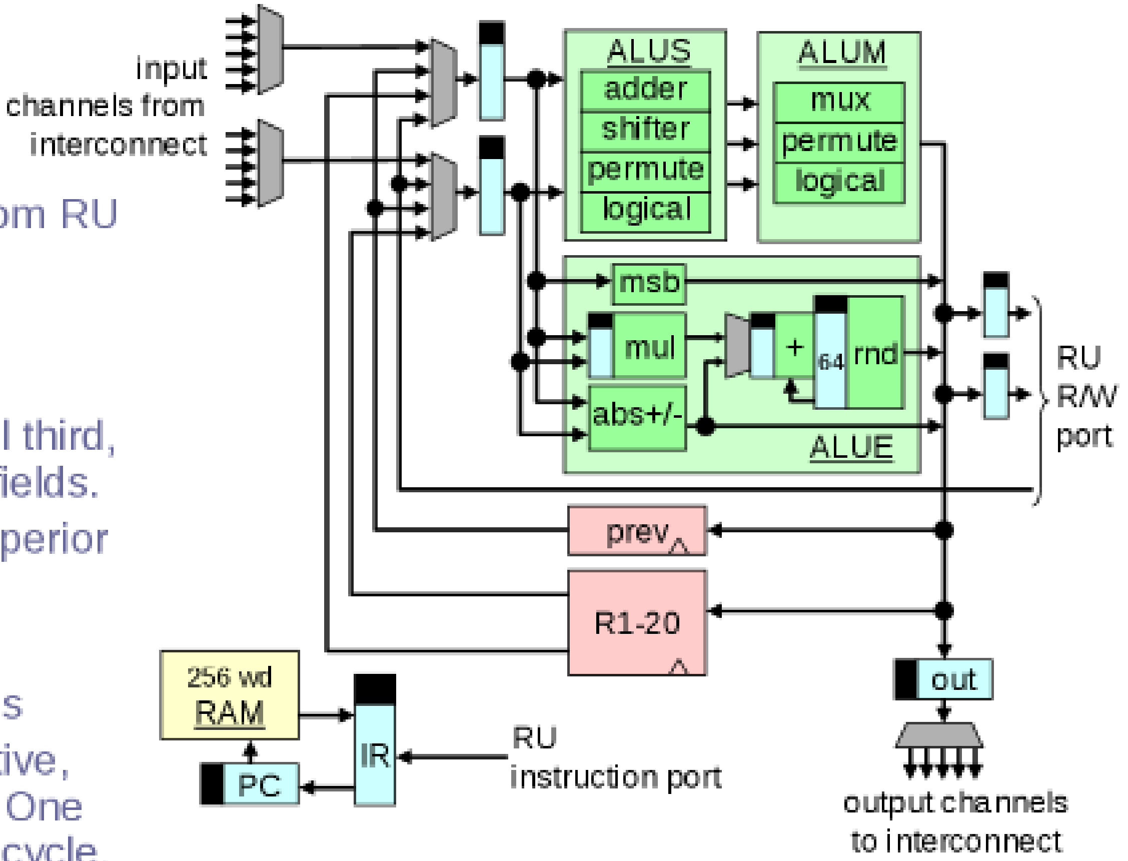
input channels from interconnect
ALU
adder
logical
mux
64 wd RAM
R0
R1-7
IR
PC
out
output channels to interconnect

16 bit instructions

ALU op	src1	src2	out	dest
ALU op	src1	opcode	out	dest
mem r/w	src1	src2	address	
branch		condition	offset	
loopstart		count	offset	

Ambric SRD Processor

- Streaming RISC with DSP extensions
- 32 bit instructions
  - 256 in local RAM, more from RU
  - Zero-overhead looping
- Multiple ALUs capture instruction-level parallelism
  - 2 ALUs in series, a parallel third, with individual instruction fields.
  - For stream processing, superior code density to VLIW
- 3 ALUs
  - 32b, dual 16b, quad 8b ops
  - 3rd ALU alongside is iterative, pipelined, for MAC, SAD. One 32b*8b or two 16b*8b per cycle, 64-bit accumulator, rounding
- RU read-write channels
- 3-stage Ambric channel datapath



32 bit instructions

ALUS op	ALUM op	src1	src2	out	dest
ALUE op	src1	src2	out	dest	
mem r/w	base	src1	src2	address	
branch	condition	offset			
loopstart	count	offset			

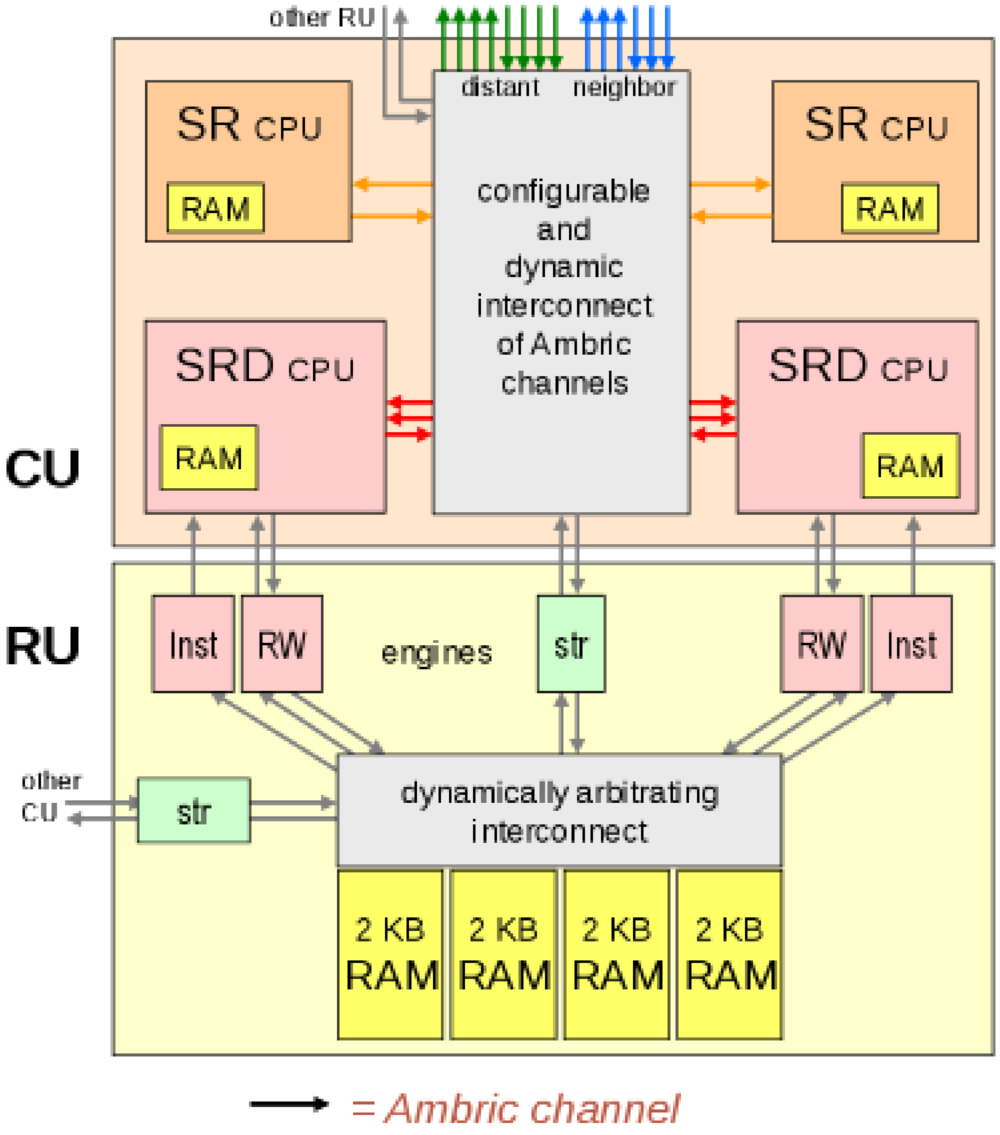
Ambric Compute Unit, RAM Unit

Compute Unit (CU)

- Two SRD 32-bit CPUs
- Two SR 32-bit CPUs
- Channel interconnect
  - CPU-CPU is dynamic under instruction control
  - CU-Neighbor, CU-Distant are statically configured

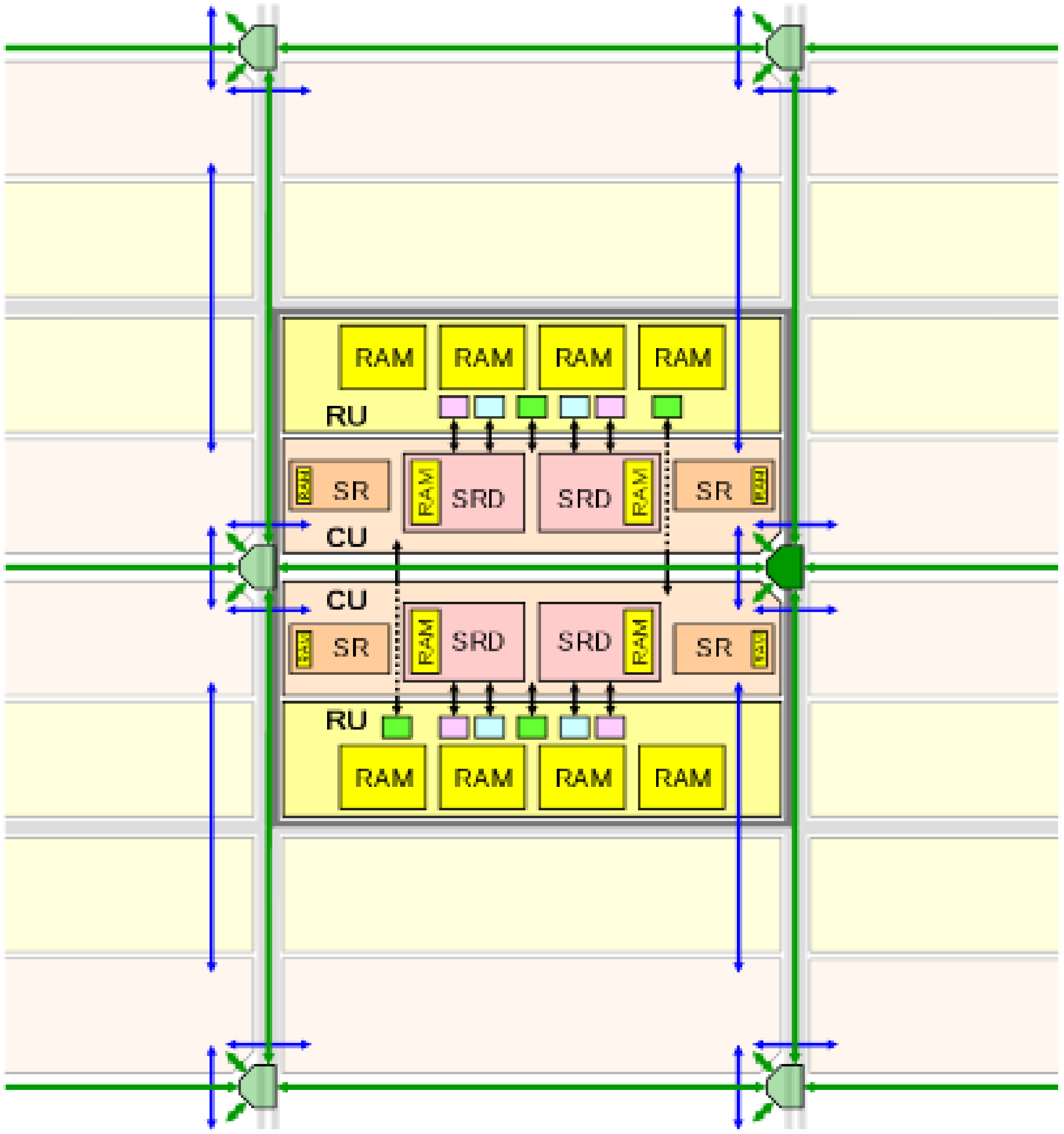
RAM Unit (RU)

- Four 2KB RAM banks
- RU engines turn RAM regions into channels
  - FIFO and random access
- Engines dynamically connect to banks through channels with arbitration



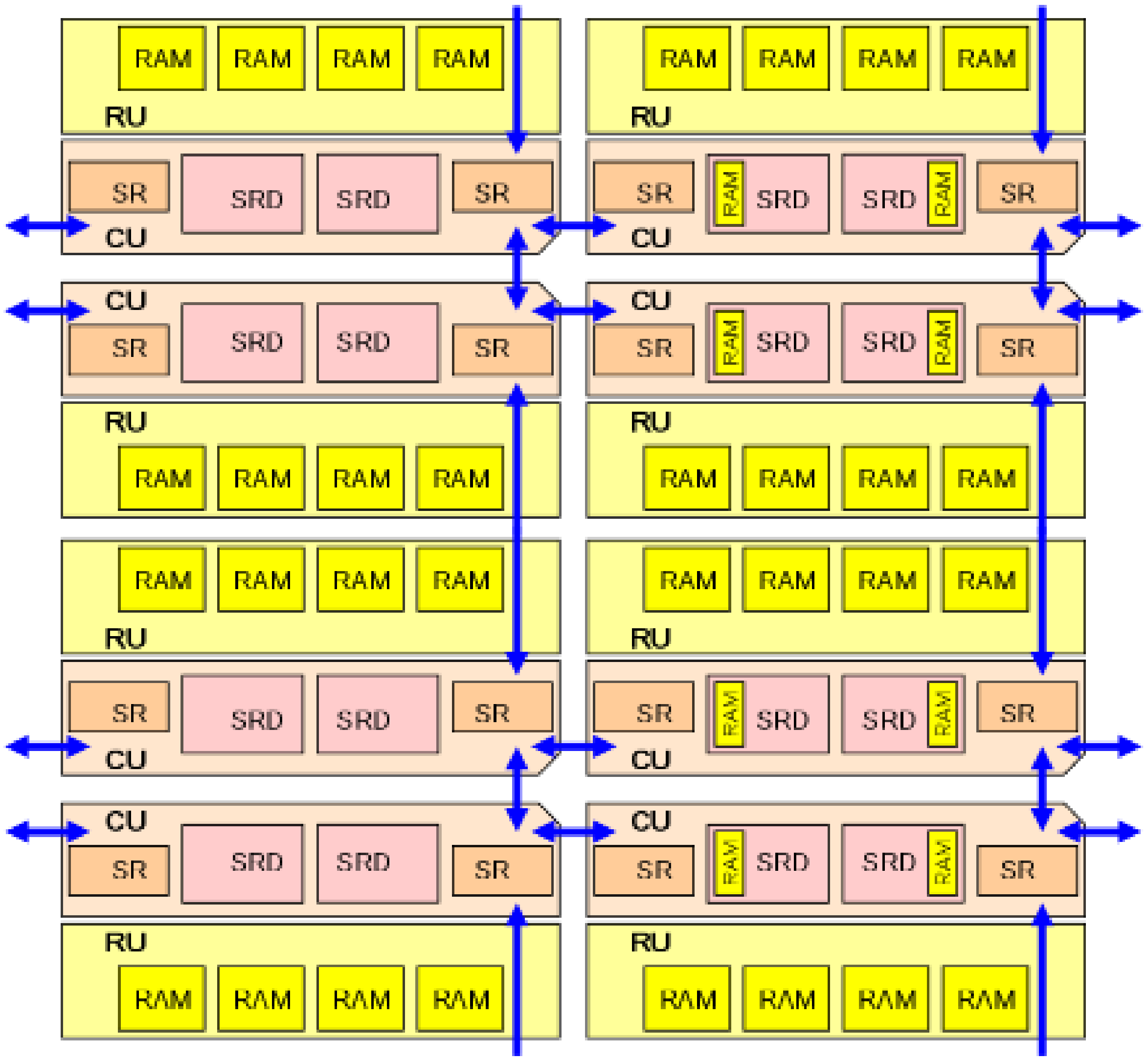
Bric and Interconnect

- Bric is the silicon building block
  - Two Compute Units (CUs)
  - Two RAM Units (RUs)
- Core is an array of brics
- Hierarchical Interconnect
  - Neighbor CU-CU channels
  - Switched inter-bric network
  - No wires longer than a bric



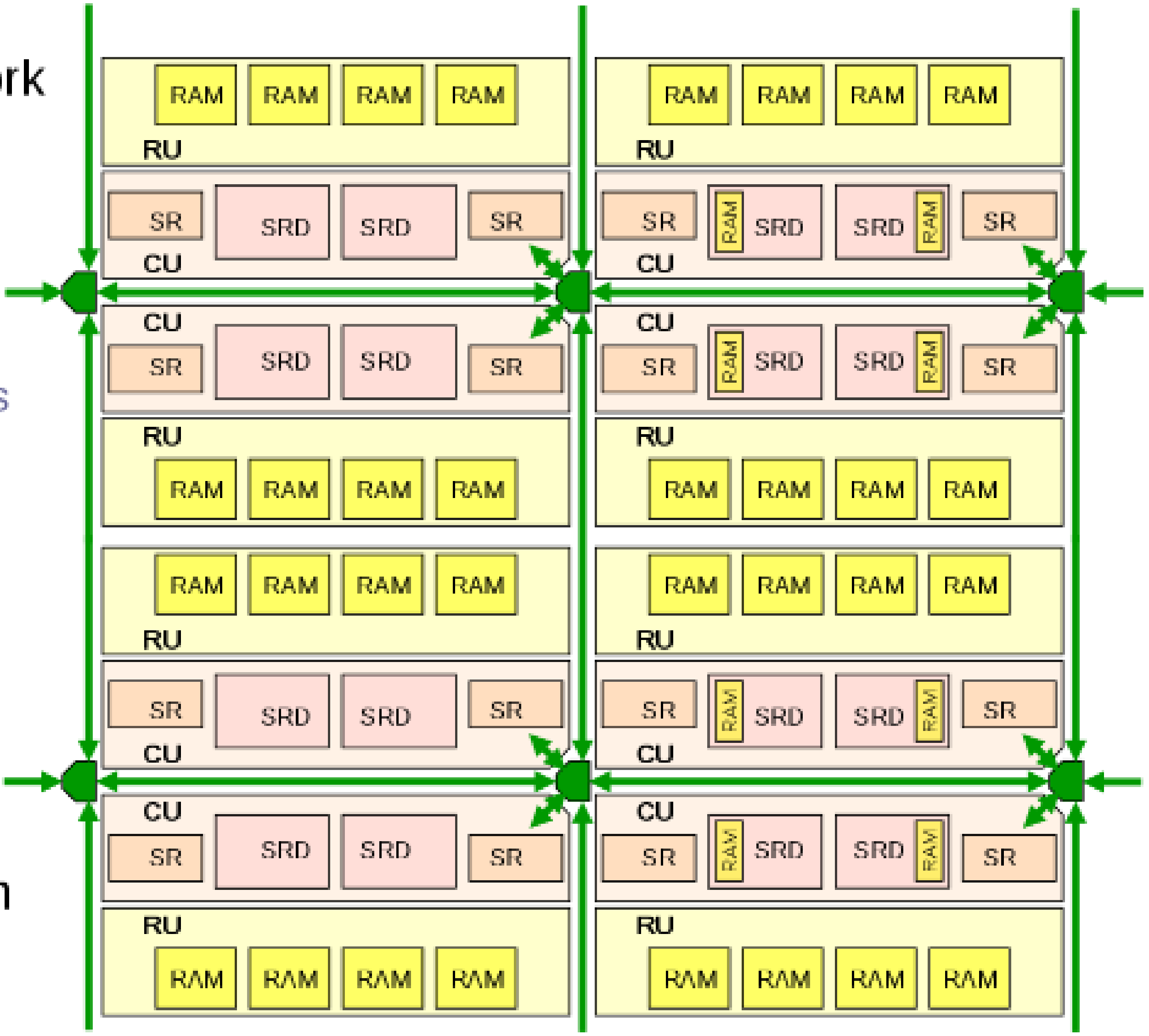
Neighbor Channels

- Neighbor channels connect neighbor CUs N/S/E/W
  - 1 channel each way
- Each channel is 32-bits wide @ up to 9.6 Gbps



Distant Channels

- Configurable network for longer routes
  - one bric per hop
- Each bric has a switch
  - connected to CUs
- Switches are interconnected by bric-long channels
  - four channels each way
  - up to 9.6 Gbps
- Total interconnect bisection bandwidth is 792 Gbps

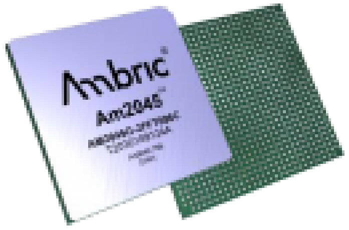


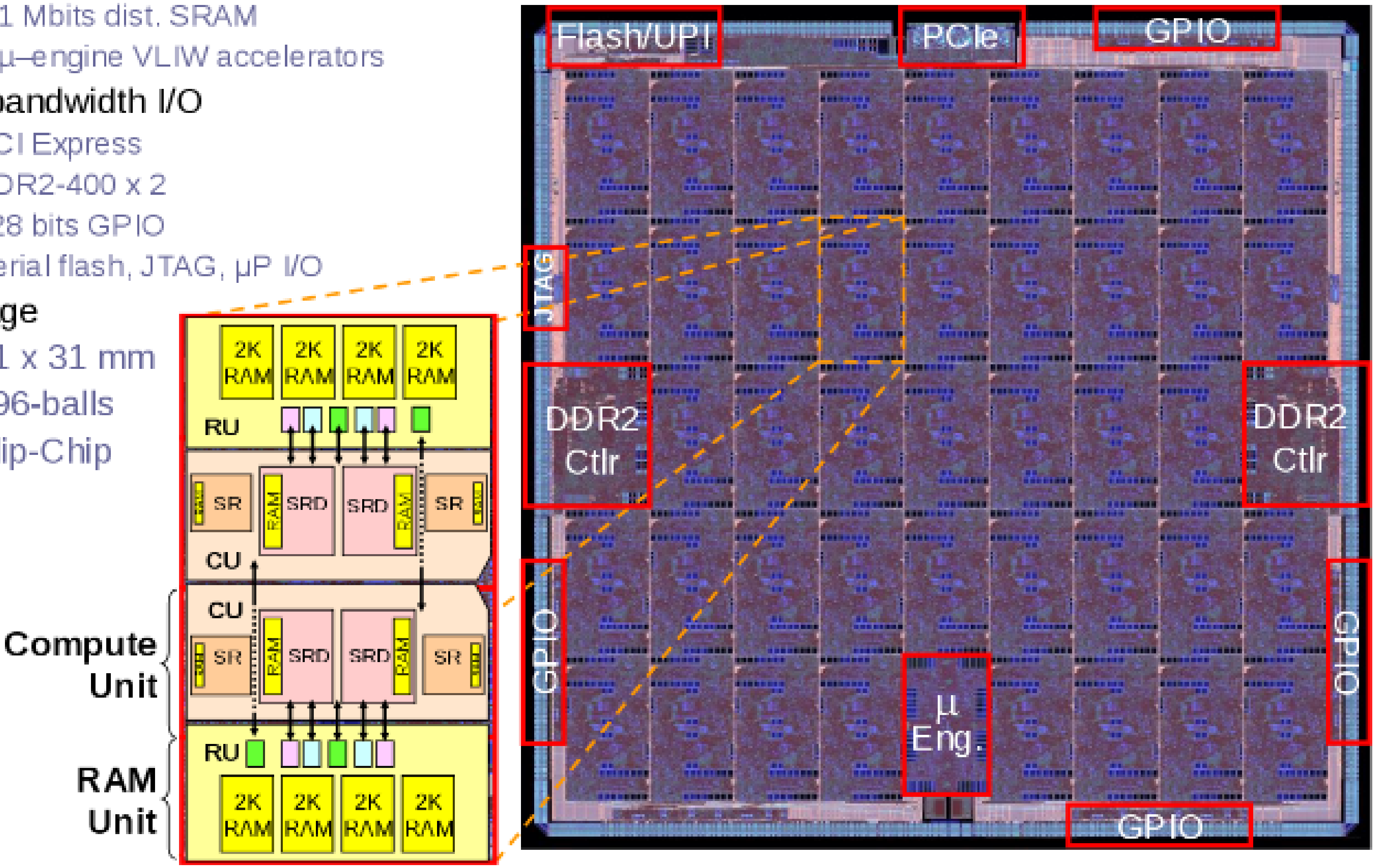
A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

Am2045 Chip

- 130nm standard-cell ASIC
  - 180 million transistors
- 45 brics, 1.03 teraOPS
  - 336 32-bit processors
  - 7.1 Mbits dist. SRAM
  - 8 μ–engine VLIW accelerators
- High-bandwidth I/O
  - PCI Express
  - DDR2-400 x 2
  - 128 bits GPIO
  - Serial flash, JTAG, μP I/O
- Package
  - 31 x 31 mm
  - 896-balls
  - Flip-Chip





Performance Metrics

- **Am2045 @ 300 MHz:**
  - 1.03 trillion operations per second (8-bit, 16-bit Sum of Abs. Diff.)
    - 60 GMACS (16x16, 32 bit sum)
  - 792 Gbps interconnect bisection bandwidth
  - 26 Gbps DRAM + 16 Gbps high-speed serial + 13 Gbps parallel

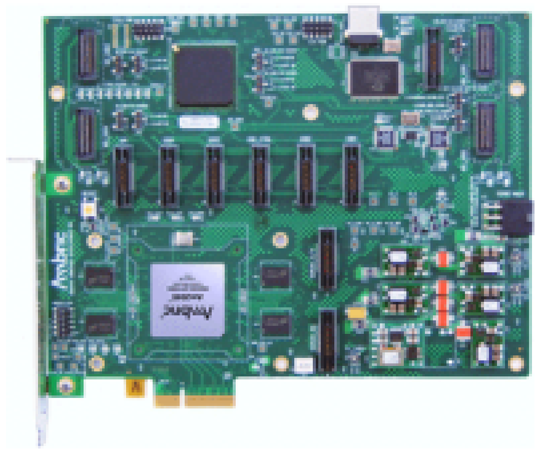
Kernel	*Instances @ Rate Each*	
32-tap FIR filters	168 @ 4.7 Msps ... 5 @ 223 Msps	16 bit data
Dot Product	168 @ 200 Msps	16 bits in, 32 bit sum
Maximum Value	168 @ 343 Msps	n=100, 16 bits, 2 wide
Saturators	168 @ 600 Msps	signed 16 to unsigned 8, 2 wide
Viterbi ACS	336 @ 600 Mbps	16-bit
1K point FFT	84 @ 8.8 Msps	complex 16-bit radix-2
AES	56 @ 181 Mbps 7 @ 1.1 Gbps	feedback modes non-feedback modes

Ambric

Ambric Development Boards

- **Am2045 Software Development Board**
  - 1 production Am2045 + SDRAM
  - PCI Express interface to host
  - For rapid software development and application acceleration

- **Am2045 Integrated Development Board**
  - 1 production Am2045 + SDRAM
  - 4 32-bit GPIO connectors, USB
  - Stand-alone capable on the benchtop, or in a PCIe slot with PC cover off
  - Serial Flash, power connector
  - For embedded development

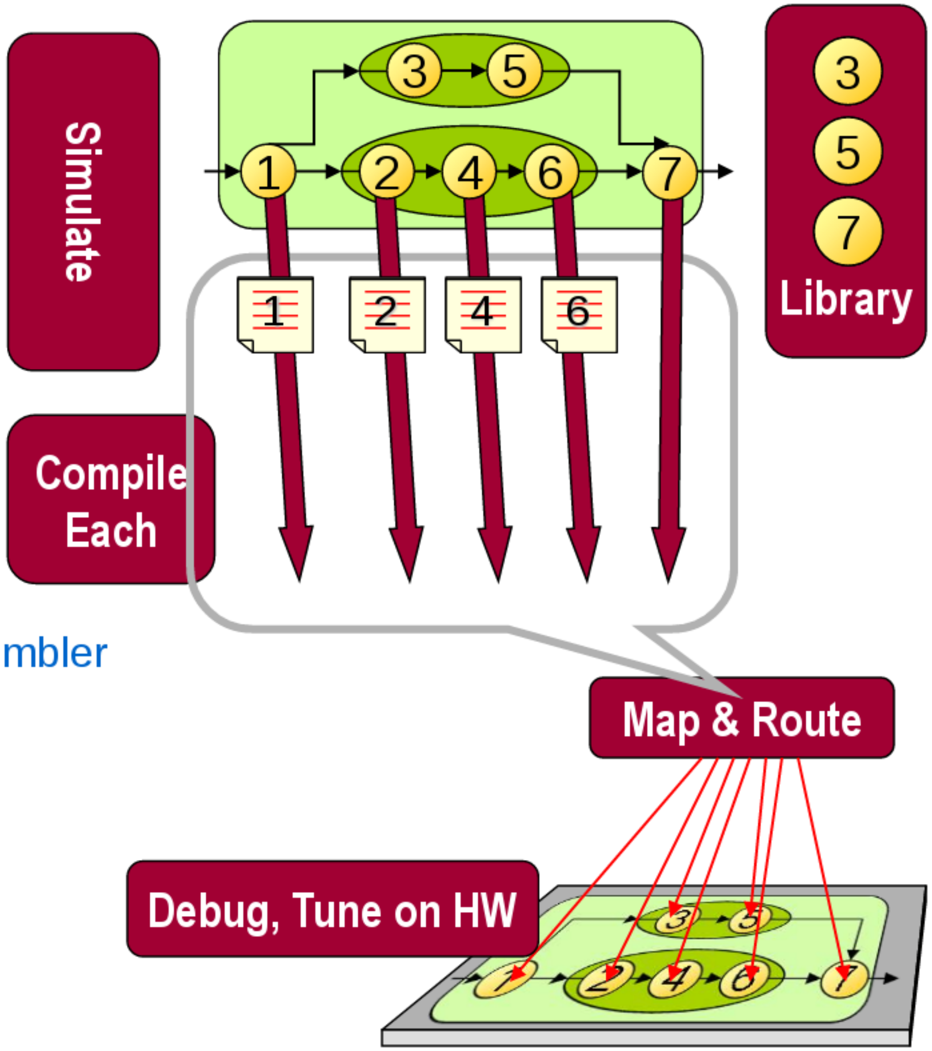


A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

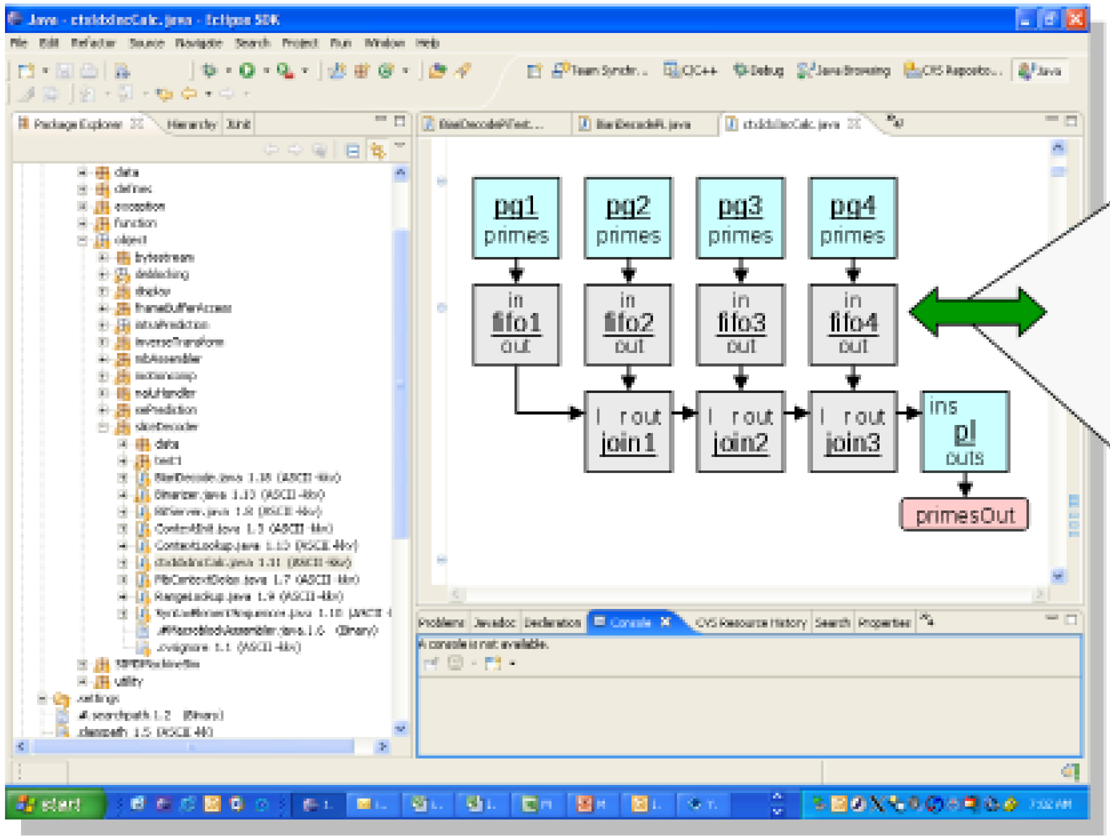
- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

Ambric Tool Chain

- Eclipse IDE (Integrated Design Env.)
  - All tools in the open IDE
- Structure
  - Conceive your application as a structure of objects and the messages they exchange
  - Divide-and-conquer using hierarchy
- Reuse
  - Encapsulated library objects
- Code and Test
  - Write your new objects in Java or Assembler
  - Verify with functional simulation
- Realize on HW
  - Compile each object separately
  - Run mapper-router, configure chip
  - Debug, profile and tune performance



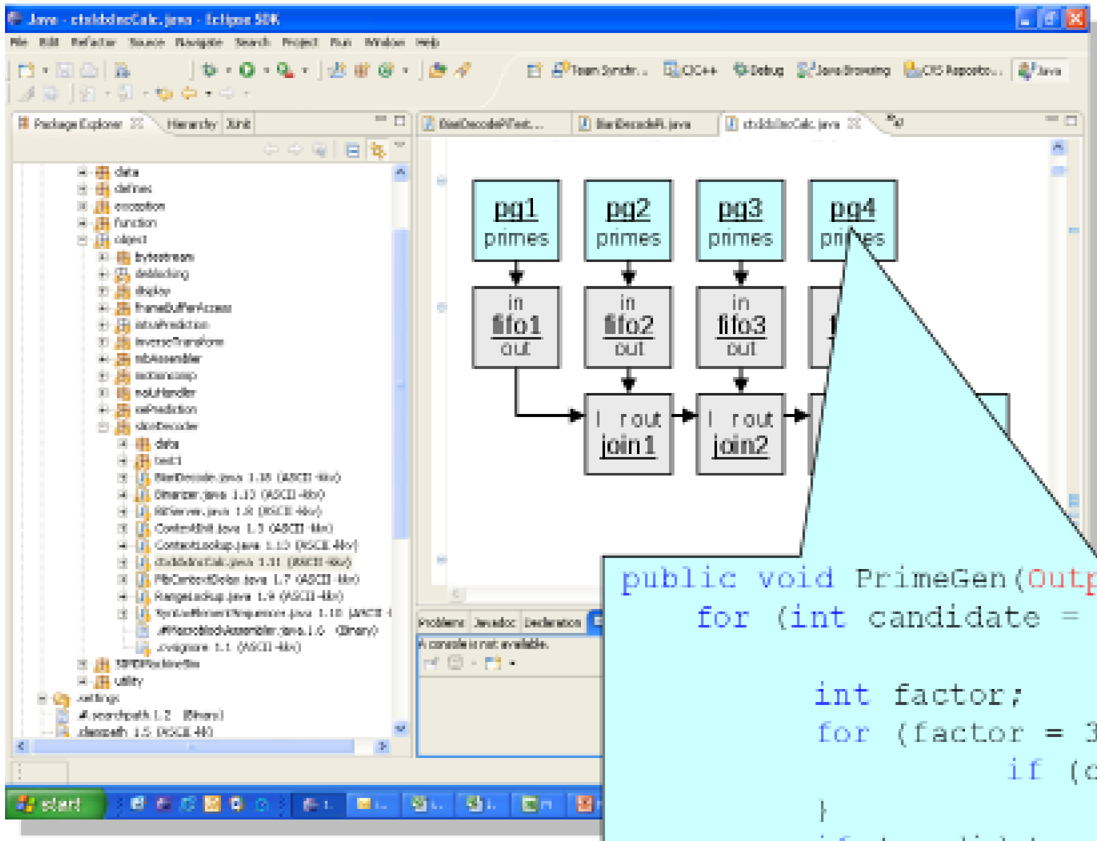
Structural Programming in aStruct



```
binding PrimeMakerImpl
    implements PrimeMaker {
  PrimeGen pg1 = {min = 3, increment =
    4, max = IPrimeMaker.max};
  PrimeGen pg2 = {min = 5, increment =
    4, max = IPrimeMaker.max};
  PrimeGen pg3 = {min = 7, increment =
    4, max = IPrimeMaker.max};
  PrimeGen pg4 = {min = 9, increment =
    4, max = IPrimeMaker.max};
  Fifo fifo1 = {max_size = fifoSize};
  Fifo fifo2 = {max_size = fifoSize};
  Fifo fifo3 = {max_size = fifoSize};
  Fifo fifo4 = {max_size = fifoSize};
  AltWordJoin join1;
  AltWordJoin join2;
  AltWordJoin join3;
  PrimeList pl;
  channel
    c0 = {pg1.primes, f1.in},
    c1 = {pg2.primes, f2.in},
    c2 = {pg3.primes, f3.in},
    c3 = {pg4.primes, f4.in},
    c4 = {f1.out , j1.l},
    c5 = {f2.out , j1.r},
    c6 = {j1.out , j2.l},
    c7 = {f3.out , j2.r},
    c8 = {j2.out , j3.l},
    c9 = {f4.out , j3.r},
    c10 = {j3.out , pl.ins},
    c11 = {pl.outs , primesOut};
}
```

- Graphical or textual entry
  - menus or text (not shown) defines channel interfaces, object parameters
- Hierarchical, modular

Standard Language for Objects



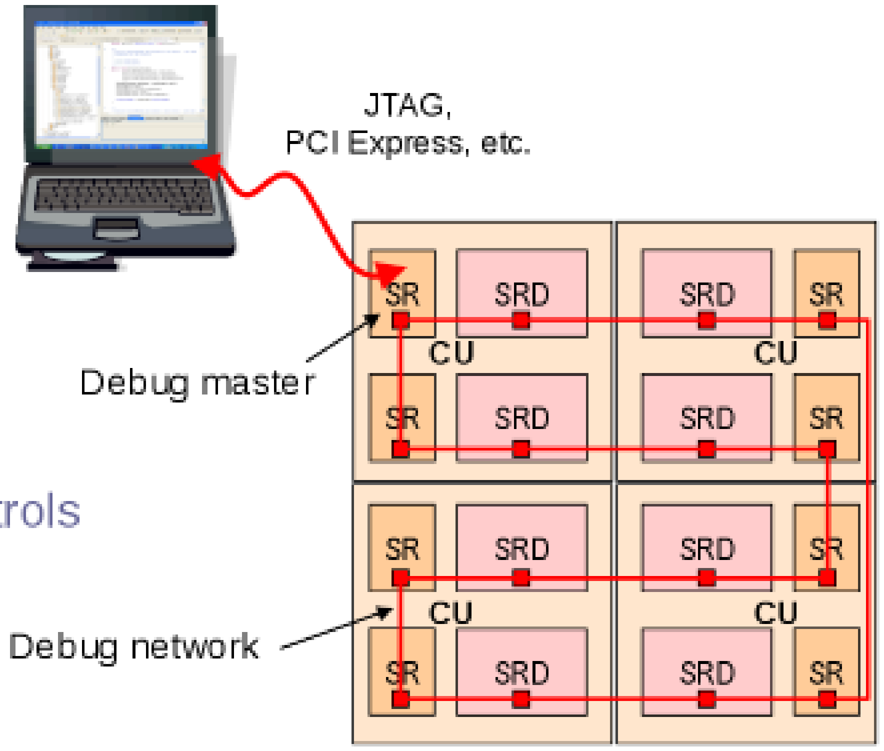
```
public void PrimeGen(OutputStream<Integer> primes) {
    for (int candidate = min; candidate <= max;
                          candidate += 2*increment) {
        int factor;
        for (factor = 3; factor <= max; factor += 2) {
                if (candidate % factor == 0) break;
        }
        if (candidate == factor) {// is prime
                primes.write(candidate); // write out
        }
        else primes.write(0);
    }
}
```

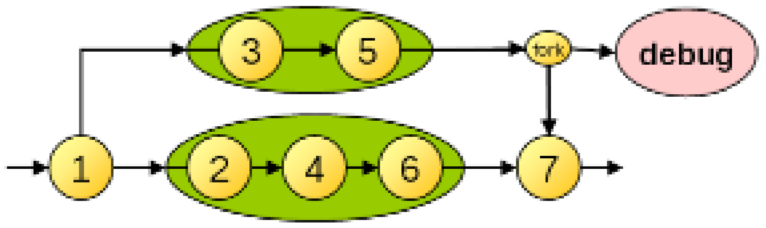
- Program primitive objects in Java
  - Strict subset of standard Java
    - static memory
  - Classes define the channels

- Language-agnostic
  - C, etc. to follow

Debugging a Massively-Parallel Application

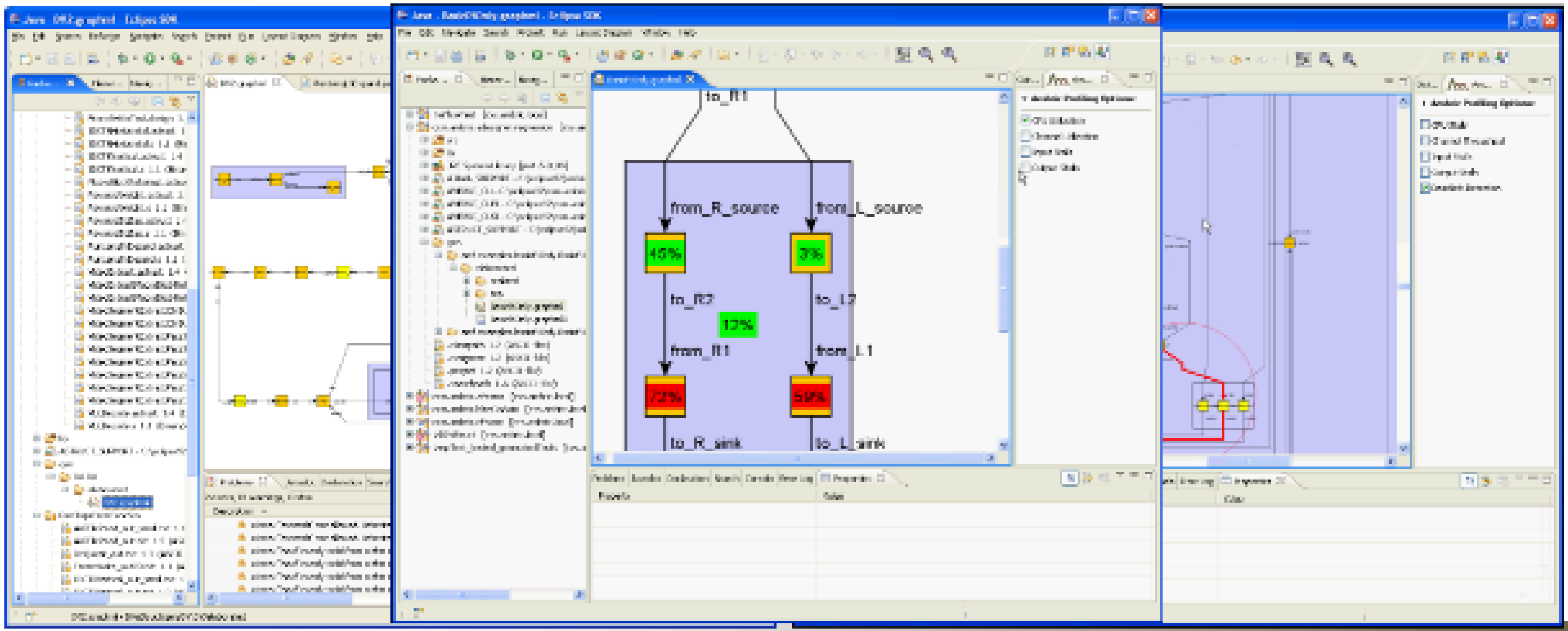
- Debugger/Profiler in Eclipse IDE
  - Integrated with source code
- Debug network in silicon
  - Strictly separate network that can't deadlock
  - Transparently observes and controls processors and channels
  - Every processor can trap, run a watchdog timer
- Add debug objects
  - Program unused processors and RAMs for debug tools
  - No effect on function or performance, thanks to Ambric channels





Source & Graphical Debugging

- Profile
  - Identify processing bottlenecks, deadlock and utilization problems
- Find
  - Seamlessly access debugger features from graphical view
  - Symbolic references, single step and breakpoints
  - Special features like ‘channel monitoring’



A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

Library Objects

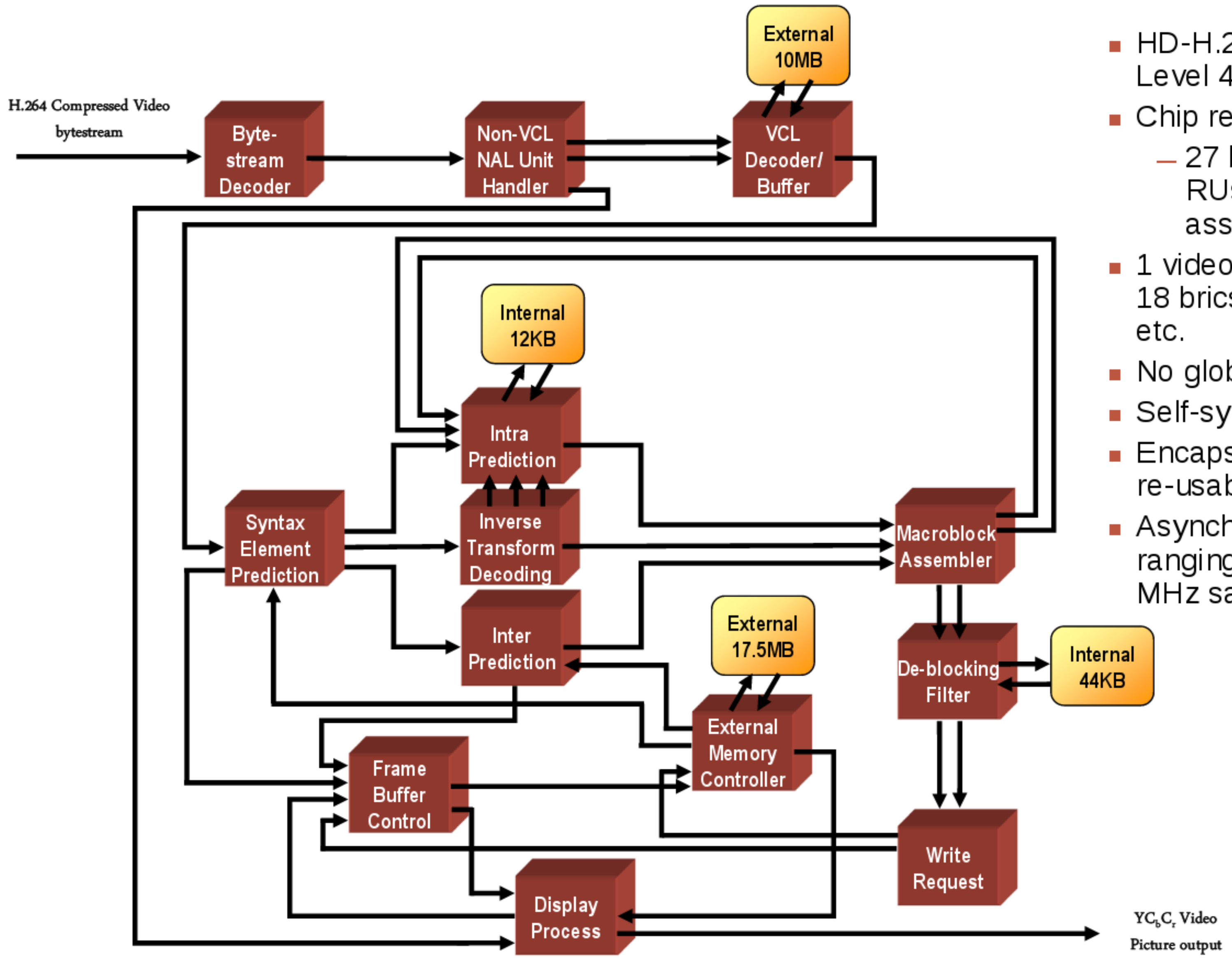
- Video Compression
  - Motion Estimation
    - Full-search
    - Hierarchical-search
  - H.264 I-frame Decoder Module
    - Deblocking Filter
    - Inverse Transform
    - Intra-prediction
    - Macroblock assembler
    - Motion Compensation
    - Cache-controller
    - CABAC decode
  - DV Decoder Module
  - DVCPRO-HD modules
    - Variable length codec
    - Forward & Inverse DCT

- Signal Processing
  - FFT radix-2, radix-4
  - FIR, IIR Filters
  - Vector Saturation
  - Maximum value search
  - Dot Product
  - Matrix Math

- Communications
  - Turbo CTC
  - Viterbi
  - AES encryption
  - Regular expression search

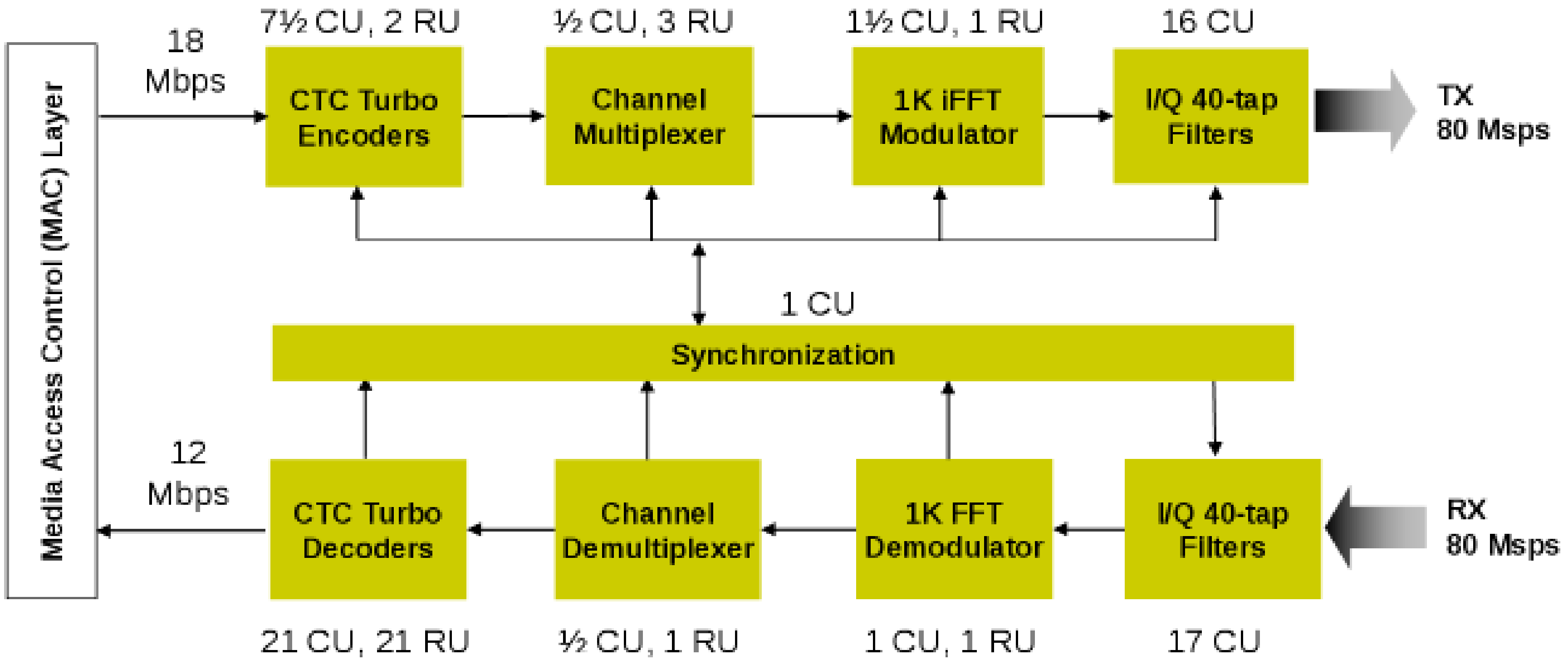
- Pixel Processing
  - HD Video scaler
  - HD De-interlacer

HD H.264 Decoder Structure



- HD-H.264, High Profile - Level 4
- Chip resources required
  - 27 brics with 100% of RUs used and 80% of associated CUs used
- 1 video stream per chip with 18 brics left over for scaling, etc.
- No global state machine
- Self-synchronizing
- Encapsulated, modular, re-usable code
- Asynchronous operation ranging from 100 MHz - 333 MHz saves power

WiMAX Base Station Block Diagram



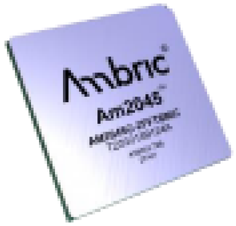
Total: 67 CU, 31 RU = 34 brics = 80% Am2045

A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program

Ambric University Program

- Ambric offers development tools, documentation, hardware, and limited technical support, at no cost, to University Program partners.
- Benefits to University
  - Access to a real massively-parallel embedded-systems architecture
  - Very high performance execution with energy efficiency
  - Easier and faster software-only development effort
  - Real execution target for research tools, languages, etc.
- Benefits to Ambric
  - Real development experience with more application areas
  - Promote innovative tools, methodologies, libraries
  - Get to know the best future graduates
- Current Members as of January 2008
  - U. Wash EE Dept.: Prof. Scott Hauck
  - Portland State U. ECE Dept.: Prof. Dan Hammerstrom
  - Halmstad U. CERES (Sweden): Prof. Bertil Svenson



www.Ambric.com

Publications

- "Synchronization through Communication in a Massively Parallel Processor Array", Mike Butts, IEEE Micro, Sept/Oct 2007.
- "A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing", Mike Butts, Anthony Mark Jones, Paul Wasson, IEEE Symposium on Field-Programmable Custom Computing Machines, April 2007.

A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing

- Structural Object Programming Model
- Architecture
- Chip
- Tools
- Applications
- University Program
- Additional Background Material

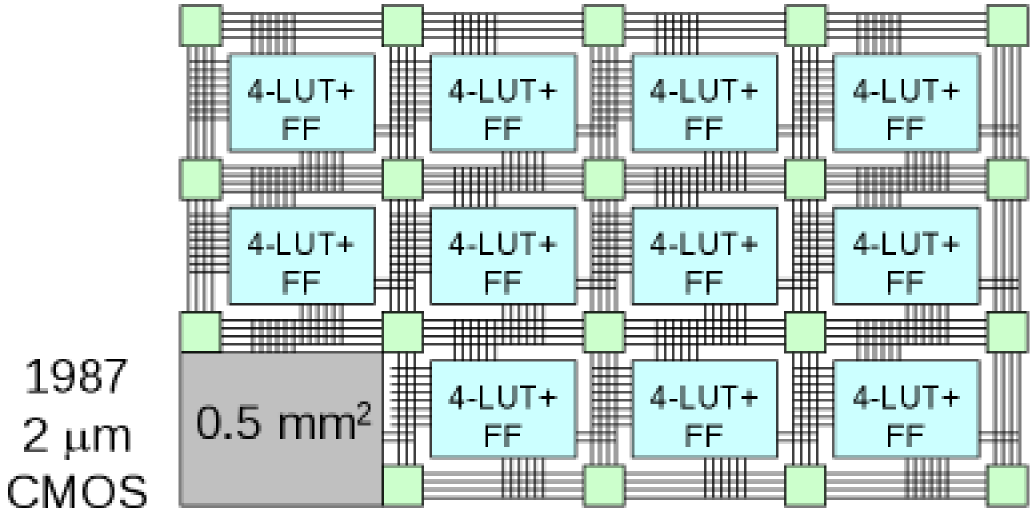
Ambric

The Energy Efficiency of Parallelism

- Strict power budgets at all levels limit power scalability
  - 1W handheld, 10W portable, 100W desktop/server
- Minimize energy per operation
  - Lower voltage: slower but far less power
- Get performance back with parallelism
  - This makes the most power-efficient use of silicon area
- Example:
  - One cool processor: 75% speed, 42% power*
  - Two in parallel: 150% speed, 84% power
- The catch is making parallelism practical. Ambric's programming model opens this door.
  - High performance made scalable

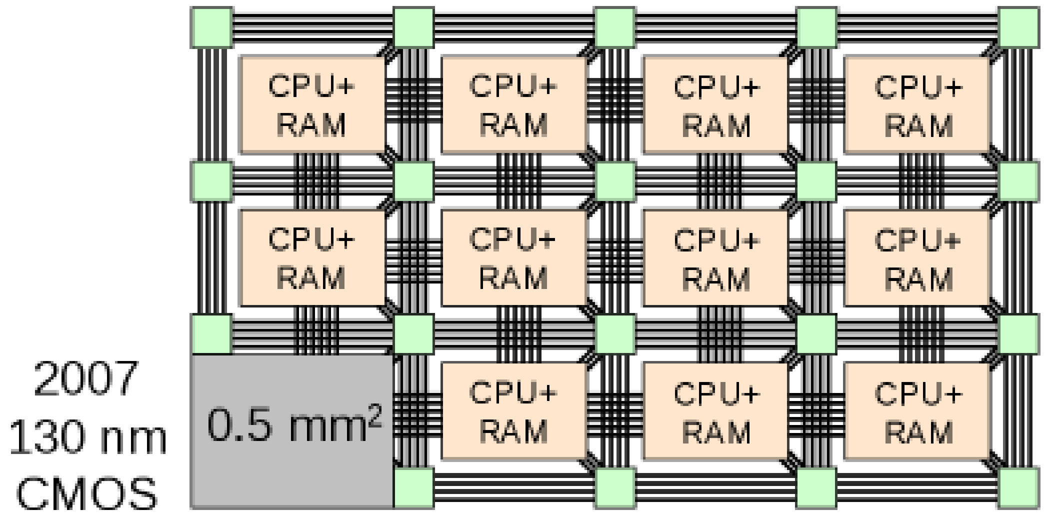
*power $\propto V^2f$, $f \propto V$ $\Rightarrow$ power $\propto V^3$

20th Century Reconfigurable Computing



- In 1987, a great new way to spend 0.5 mm^2 of silicon was:
  - A 4-LUT, a flip-flop, and reconfigurable wires
- But the FPGA was never an ideal computing platform:
  - RTL productivity is not scaling with Moore's Law
  - High-level synthesis has had limited success
  - Developer must be mindful of HW issues such as timing closure
- RC developer must be application expert, SW and HW engineer

21st Century Reconfigurable Computing



- What is the best way to use 0.5 mm² of silicon today?
  - 32-bit CPU, several KB of RAM, and reconfigurable buses

- So just fab a chip with CPUs and buses, and throw it over the wall at the programmers. Not likely to succeed!

- Pick a good **programming model** for reconfigurable computing ***first***. Then build silicon and tools to implement that model.