# CS294-7: Reconfigurable Computing

## Day 1: Tuesday, January 21

---

- Questionnaire
- Introductions
- Defining Reconfigurable Computing
- Defining this class
- Administrivia
- Course Overview
- Questions

**John Wawrzynek**

Associate Professor

University of California at Berkeley

Computer Science Division

`<http://www.cs.berkeley.edu/~johnw>`


- PhD Caltech (1987)
  - With Carver Mead
  - Thesis on Reconfigurable Architecture
- Original Member of Silicon Compilers, Inc.
- Other Current UCB projects:
  - PI of ONR funded project:
    "Construction of Connectionist Network Supercomputer"
  - T0: Single-Chip Vector Microprocessor
  - IRAM with D. Patterson
- Courses: Digital Design, VLSI Design, Computer Architecture
- Recent Pubs:

  J. Wawrzynek, K. Asanovic, B. Kingsbury, J. Beck, D. Johnson, and N. Morgan. SPERT-II: A Vector Microprocessor System. *IEEE Computer*, March 1996.

  Krste Asanovic, James Beck, Bertrand Irissou, Brian E. D. Kingsbury, Nelson Morgan, and John Wawrzynek. The T0 Vector Microprocessor. In *Proceedings of Hot Chips VII*, August 1995.

# André DeHon

Visiting Postdoc Research Engineer `<http://www.cs.berkeley.edu/~amd>`

- PhD Thesis (MIT, August 1996)
  *Reconfigurable Architectures for General-Purpose Computing*
  Advisor: Dr. Thomas F. Knight, Jr.
- Reinventing Computing Project at MIT
- Architectures
  - DPGA – multicontext FPGA (FPGA'94+'96, FPD'95+'96)
  - TSFPGA – time-switched FPGA
  - MATRIX – coarse-grained reconfigurable (FCCM'96)
  - DPGA-coupled Microprocessor (concept FCCM'94)
- SM Thesis (MIT, January 1993):
  *Robust, High-Speed Networks for Large Scale Multiprocessing*
  MIT Transit Project

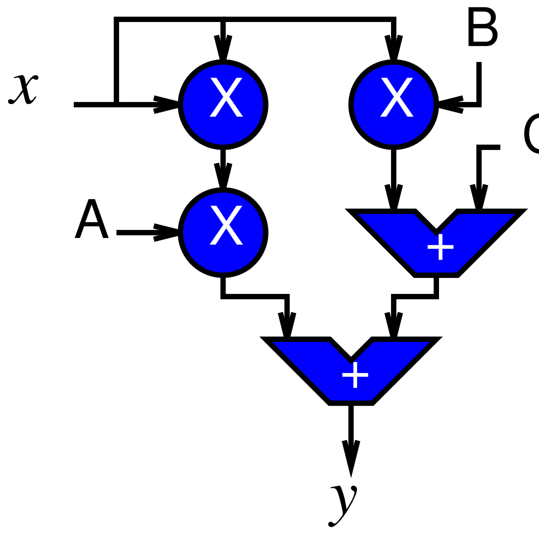
# What is Configurable Computing?

Short answer: Computing via post-fabrication, spatially programmed connection of processing elements.

$$y = Ax^2 + Bx + C$$

Resembles "hardware" computation

"wire" up functional computing blocks



# Fixed v/s “Programmable”

**Fixed Function:**

- Computes one function
  (*e.g.* FP multiplier, divider, DCT )
- Function defined at fabrication

**“Programmable”:**

- Compute “any” computable function
  (*e.g.* Processors, DSPs, FPGAs)
- Function defined after fabrication

# Universallity

"Any" Computation?

$\rightarrow$ Any computation which can "fit" on the "programmable" substrate

Limitations:

- Hold entire computation
- Hold all intermediate data

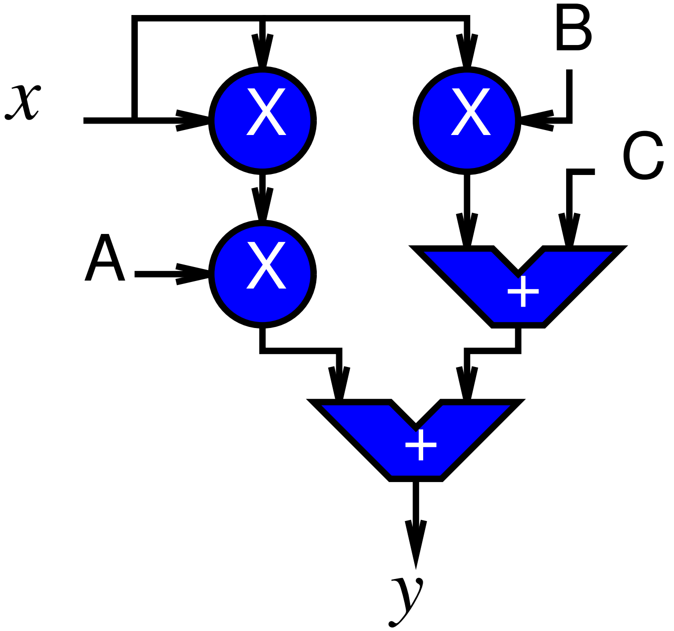
# Instruction Binding Time

When is the behavior of the computing element defined and described to the computational media?

1. Fabrication time $\rightarrow$ Fixed function device
2. Beginning of product use $\rightarrow$ Actel/Quicklogic FPGAs
3. Beginning of a usage epoch $\rightarrow$ Reconfigurable FPGAs
4. Every Cycle $\rightarrow$ traditional RISC processors

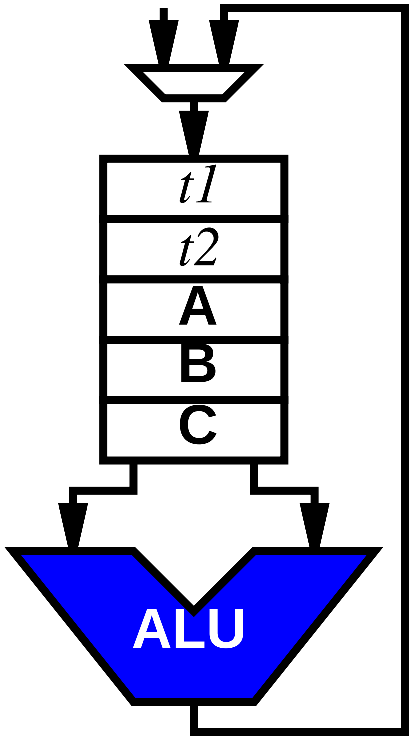
# Spatial v/s Temporal Computing

## Spatial Computation




## Temporal Computation

$t1 \leftarrow x$
$t2 \leftarrow \mathbf{A} \times t1$
$t2 \leftarrow t2 + \mathbf{B}$
$t2 \leftarrow \mathbf{t2} \times t1$
$y \leftarrow \mathbf{t2} + \mathbf{C}$

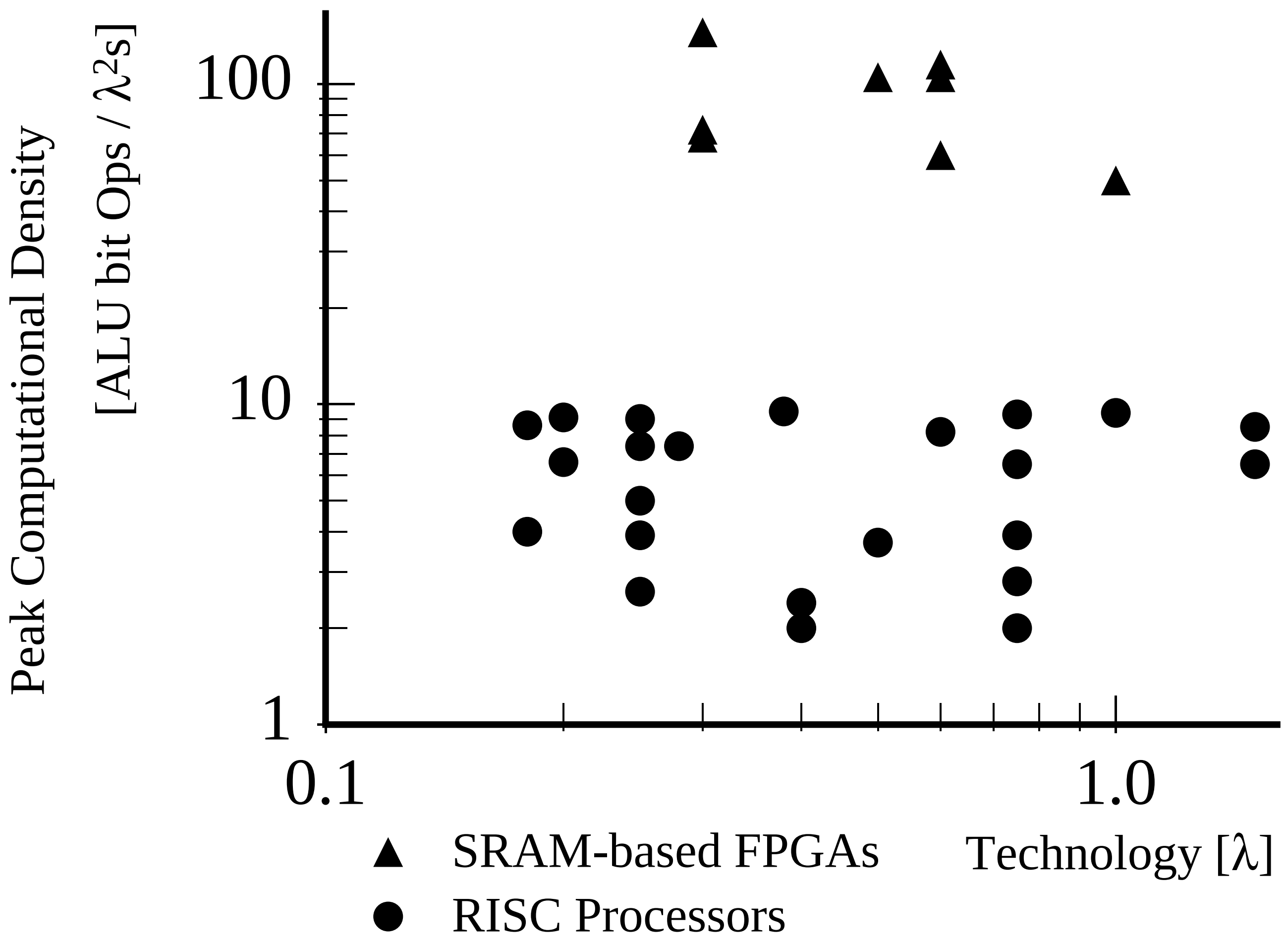


## Benefits of a Stored Program Processor Over Fixed Function Devices

1. Non-permanent customization and application development after fabrication
   - more important than ever
   - large fixed design costs, economies of scale
   - time-to-market, evolving req. and standards

2. *Reuse* same active computing resources heavily to support "large" computations on "small" amount of hardware
   - very important in the pre- and early VLSI days
   - less important as we approach 100G$\lambda^2 \rightarrow$1T$\lambda^2$/IC
   - many problems fit without heavy reuse
   - density advantage for spatial implementation

# Density of Reconfigurables






BRASS

# Reconfigurable Benefits

- $10\times$ raw density benefit
  - reduction of control, instruction distribution, and instruction and data storage
- potential for fine-grained yield
  - feasible to control/deploy computational units at bit level

# Reconfigurable Drawbacks

- Each computing resource is dedicated to a single function.
- Must dedicate spatial computing resources for every computational subtask required.
- Infrequently needed portions of a computation sit idle most of the time

$\rightarrow$ inefficient resource utilization

# Where is RC most interesting?

**Regular applications** – need to perform the same basic operations repeatedly (reuse resources primarily to perform the same operation)

**High concurrency** – large number of operations which can occur simultaneously

(for Traditional FPGAs)

**Fine-grained data** – small operand data widths

# Typically Mixed

Typical Devices and Systems mix:

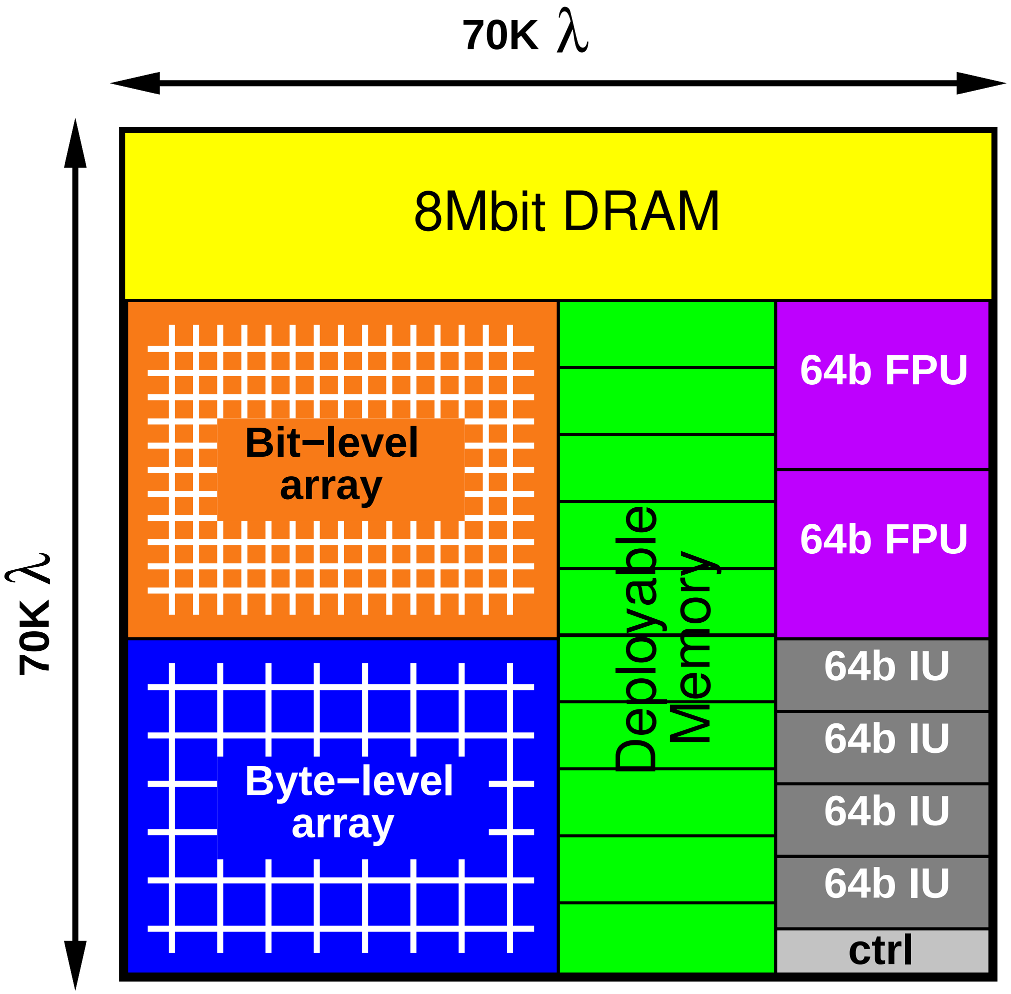
- fixed
- programmable
- *configurable*

*e.g.*

- Modern processors – Programmable ALU, Fixed FPU
- Multimedia engines – Programmable ALU, fixed motion estimation hardware
- ASIC/ASIPs – Processor core + fixed function

# Future IC Systems

- bulk DRAM
- reconfigurable
  - fine-grained
  - coarse-grained
- fast RAM
- fixed units
- dedicated control



# Reconfigurable Design Relevant to

- FPGA architects
- “General-Purpose” Processor architects
- ASIC/ASP architects
- System designers who may use any of the above

# Reconfigurable Computing: State-of-the-Art

- Conventional FPGAs weren’t really designed for reconfigurable computing
- Discovery and remapping for FPGA is not an automated process
- No “standard” design flow
- Small pool of design expertise

# Challenges

- Expand understanding of where to exploit
- Automate (parts of) the discovery process
- Develop more “Computing Friendly” architectures
- Develop tactics for managing reconfigurable resources
- Understand how to mix “reconfigurability” into systems with fixed and programmable functional units.

# This Class

- Architectures for late-binding computing systems
- Emphasis on spatial computing
- Heavy look at architecture
  - Instructions
  - Interconnect
  - Compute
  - Data retiming

# Admin

- Two lectures / week
- Typically 2-3 papers to read for each lecture
- Student Requirements / Grading:
  - scribe lectures (25%)
  - small spatial computing project (25%)
  - reconfigurable computing research project (50%)

# Intro

Today $\rightarrow$ February 11

1. Motivation: What's RC and Why is it interesting?
2. Computing requirements, VLSI Area, scaling, capacity
3. Instructions and GP architecture space
4. FPGA intro/tutorial
5. Dig into one or two CCMs (Perle-1, Cheops)
6. Dig into an application (SPLASH sequence matching)
7. Comparing computing devices

# Spatial Computing

February 13 $\rightarrow$ 27

1. Systolic Architectures and Algorithms
2. Sampling of Systolic Structures
3. Cellular Automata
4. Bit Serial Processing

# Interconnect

March 4 $\rightarrow$ 18

1. Interconnect (relative area, delay, capacity, growth, Rent, Sea-of-Gates)
2. Traditional FPGA Interconnect (Xilinx, Altera, Brown+Rose, Lewis)
3. VLSI interconnect structures
4. Partitioning and Placement
5. Routing

# Compute

March 20 $\rightarrow$ April 8

1. Memory as a programmable element;
   LUTs+interconnect as an alternate to one big memory.
2. LUT Mapping
3. Alternatives to LUTs: ULMs, PLAs
4. ALU bits, fast carry logic, and conventional CLBs

# Retiming

April 10 $\rightarrow$ 17

1. Fine-grained retiming – heavy pipelining, delay minimization, c-slow
2. Coarse-grain – bit-serial, fifo, video stream; VLSI structures and costs for retiming (*e.g.* flip-flops, shift-registers, register files, fifos, SRAM, DRAM)
3. Retiming as behavioral optimization (minimize function usage, memory intermediates)

# Other Issues

April 22 $\rightarrow$ May 1

1. Control (where data affects the instruction (operation performed) and/or istream) – FSMs
2. Binding time and specialization
3. Programming styles (single lecture overview)
4. Reuse/sharing