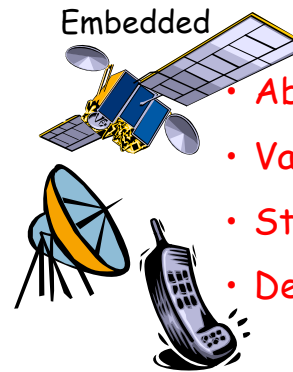


Reconfigurable Computing

Seth Copen Goldstein
CMU

Joint work with
Herman Schmit, Matt Moe, Mihai Budiu,
Srihari Cadambi, Reed Taylor, and Ron Laufer

Application Base Is Changing



- Abundant parallelism
- Variable bit widths
- Streaming data
- Demands high performance

- Compression
- Optical Flow
- Changing Protocols
- Synthetic Aperture Radar
- Filters
- 3D Graphics
- Real time video and Audio
- Encryption
- Compression
- Search

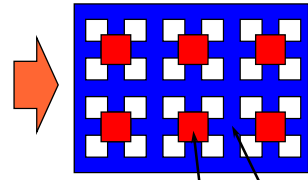
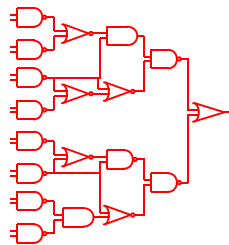
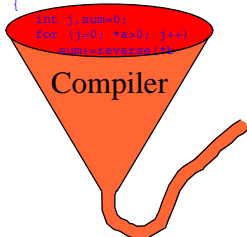
General-Purpose Custom Hardware

General-Purpose

Custom Hardware

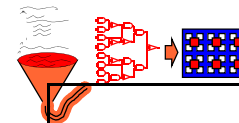
General-Purpose
Custom Hardware

```
int reverse(int x)
{
  int k,r=0;
  for (k=0; k<64; k++)
    r |= x&1;
    x = x >> 1;
    r = r << 1;
}
int func(int* a,int *b)
{
  int i,j,m,n;
  int i=0;
  int j=0;
  int m=0;
  int n=0;
}
```

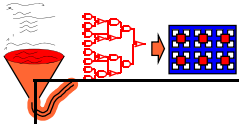


Logic Blocks
Routing Resources

Why Not?



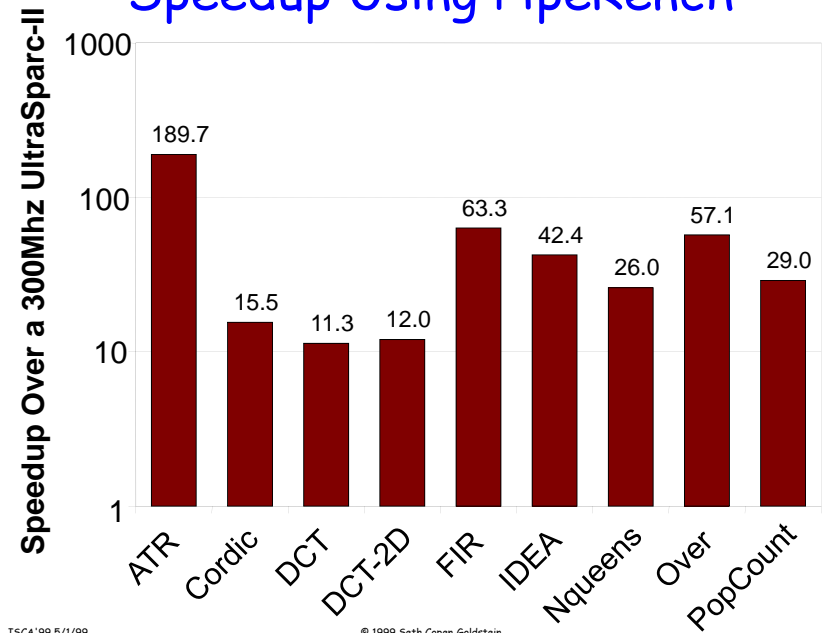
	FPGAs
Design Goal	Glue-logic
Configuration Time	Long (> 100 ms)
Resources	Fixed
Forward Compatibility	NO
Programming Model	None
Development Environment	Poor



Why?

	FPGAs	PipeRench
Design Goal	Glue-logic	Datapaths
Configuration Time	Long (> 100 ms)	Almost zero
Resources	Fixed	Virtual
Forward Compatibility	NO	Yes
Programming Model	None	Pipelines
Development Environment	Poor	Good

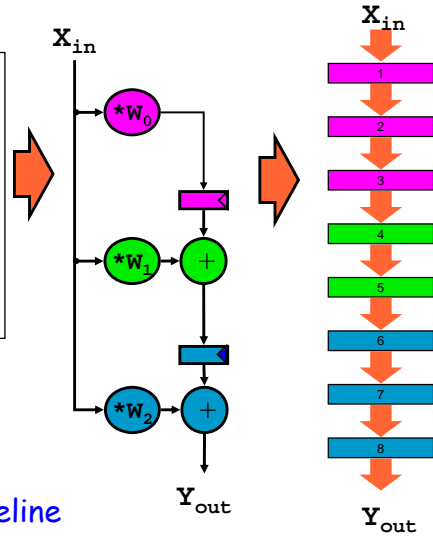
Speedup Using PipeRench



Stream-based Function: FIR

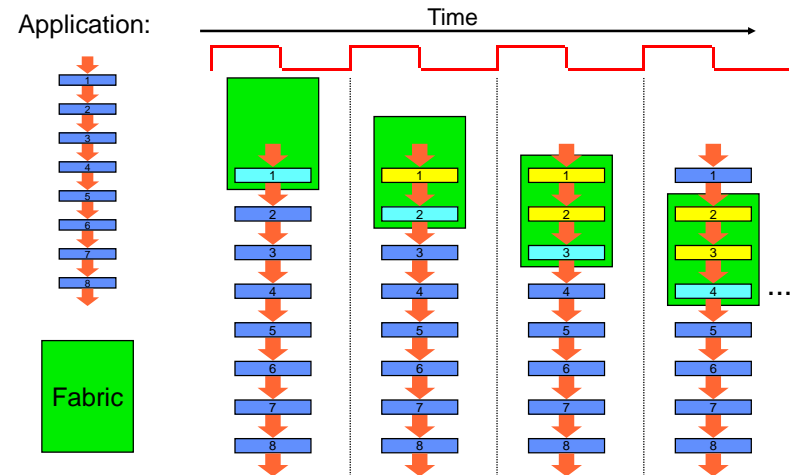
```

for (i=0; i<maxIn; i++)
{
  y[i]=0;
  for (j=0; j<Taps; j++)
  {
    y[i] += x[i+j]*w[j];
  }
}
  
```



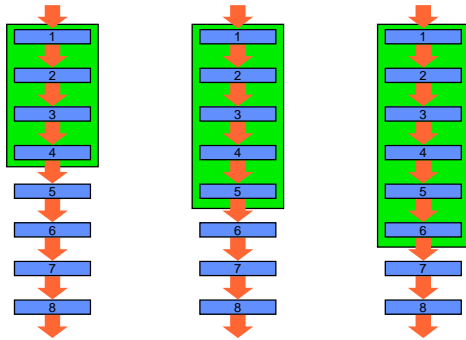
- Pipeline Model
- Constant Propagation
- Local Registers
- A Flexible Vector Pipeline

Pipelined Reconfiguration



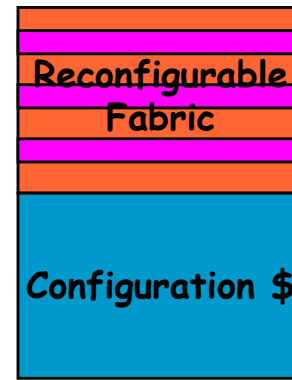
Hardware Virtualization

- More Hardware = More Throughput



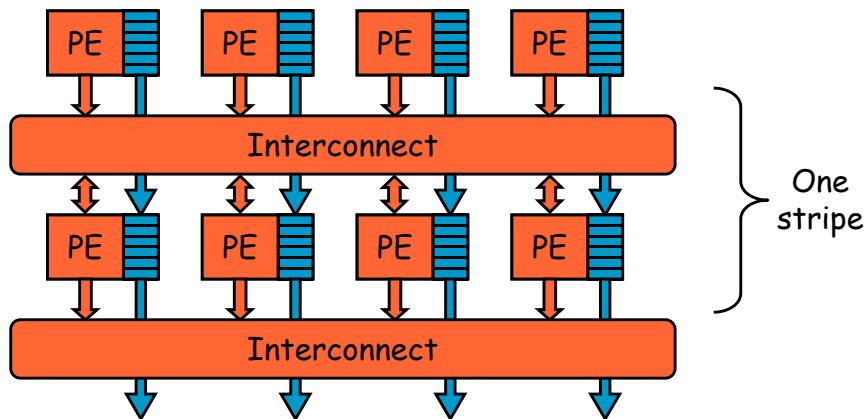
- Forward Compatibility

PipeRench Architecture



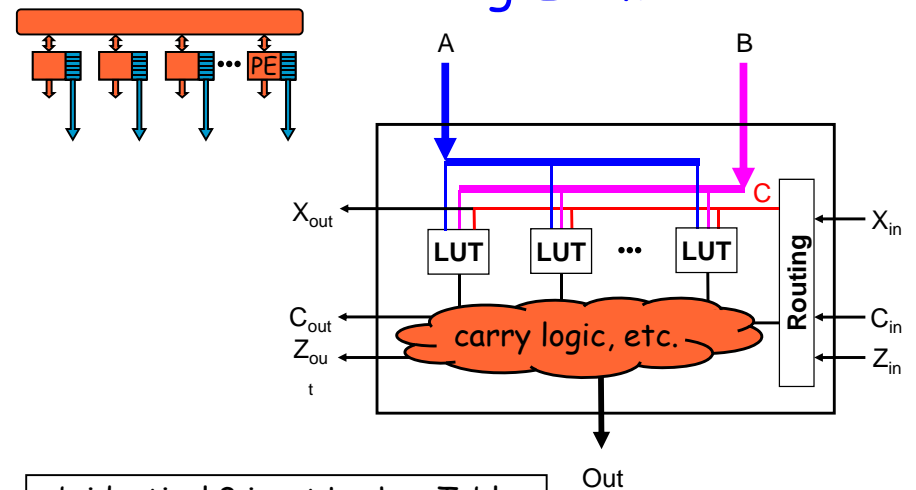
- On chip configuration cache
- Fabric divided into stripes
- Each stripe is 1 pipeline stage
- Configure 1 stripe in 1 cycle
- Apparent configuration load time is zero!

The PipeRench Fabric



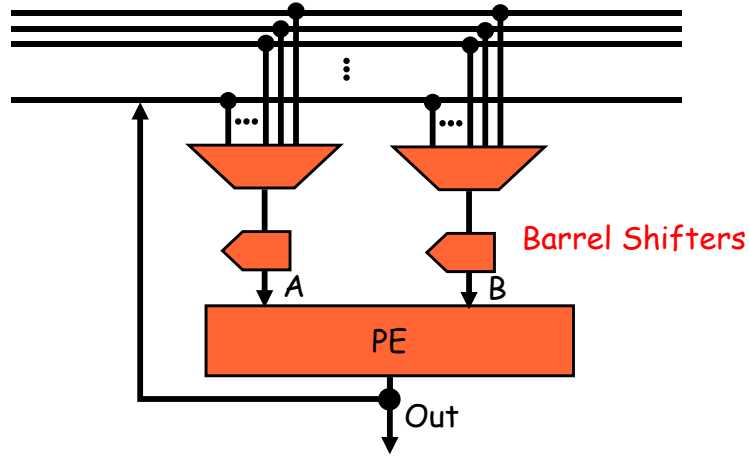
- Word based Processing Element
- Word based interconnect

A Processing Element



- b identical 3 input Lookup Tables
- A & B inputs from interconnect

The interconnect



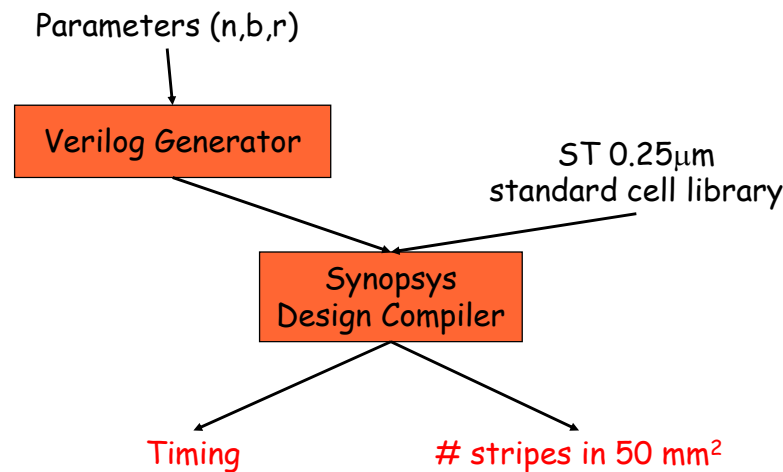
Interconnect composed of n b-bit lines

Evaluating the Design Space

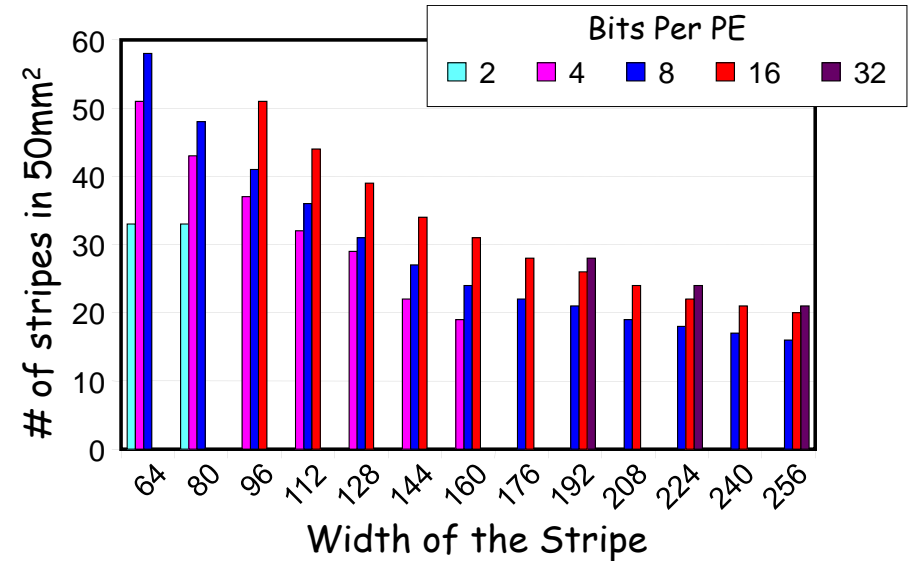
Methodology: Examine the hardware-software interaction

- Parameterize the architecture
 - Stripe is $n*b$ bits-wide $64 \leq n*b \leq 256$
 - PE is b -bits wide $2 \leq b \leq 32$
 - PE has r registers $2 \leq r \leq 16$
- Parameterize the compiler
- Compile kernels for each design point.

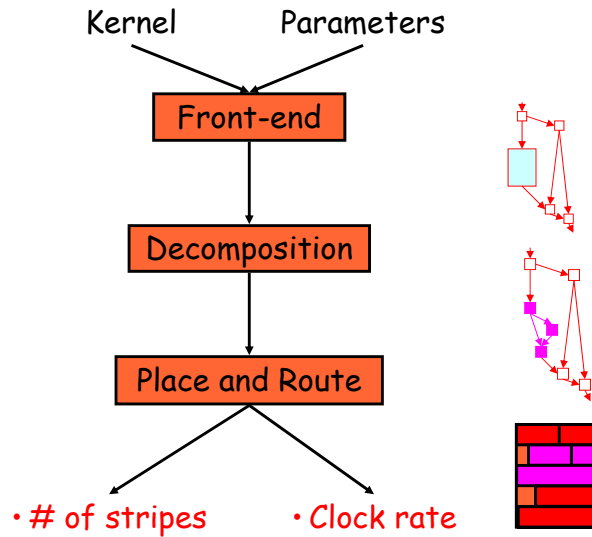
Hardware Synthesis Flow



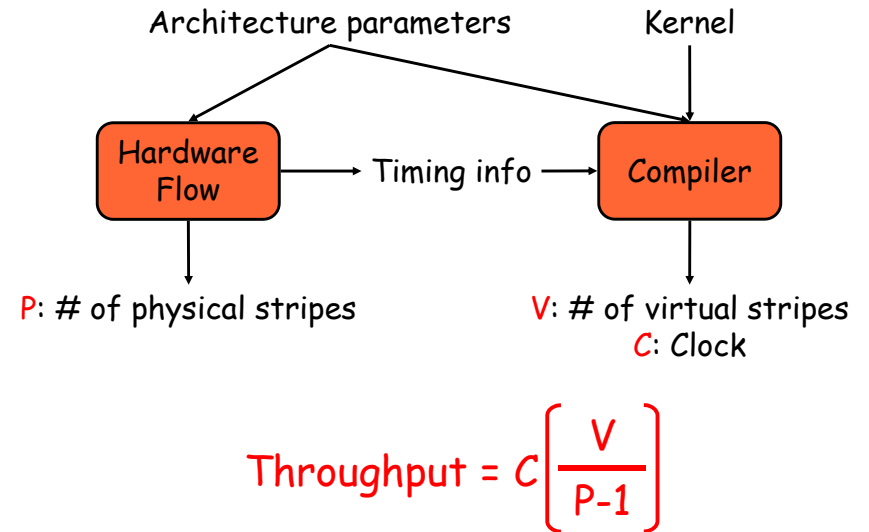
The Number of Stripes in 50mm²



The Compiler



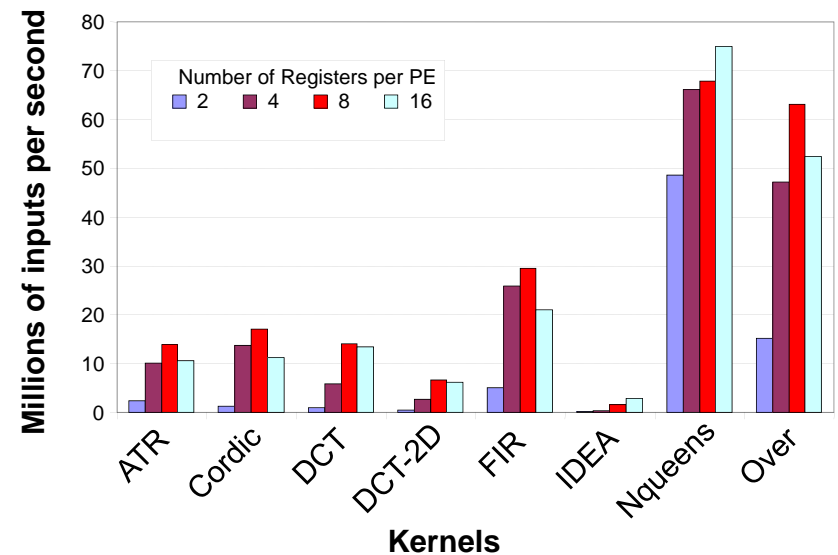
Throughput



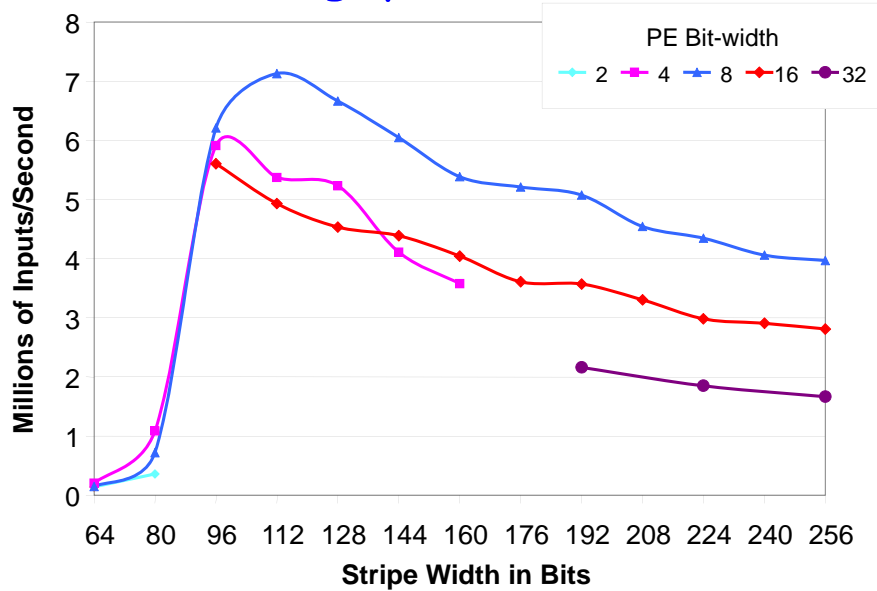
PE width tradeoffs

	Narrow	Wide
Utilization	High	Low
Carry Chain Speed	Slow	Fast
Configuration Size	Big	Small
Interconnect Flexibility	More	Less
Ease of Compilation	Easy	Hard

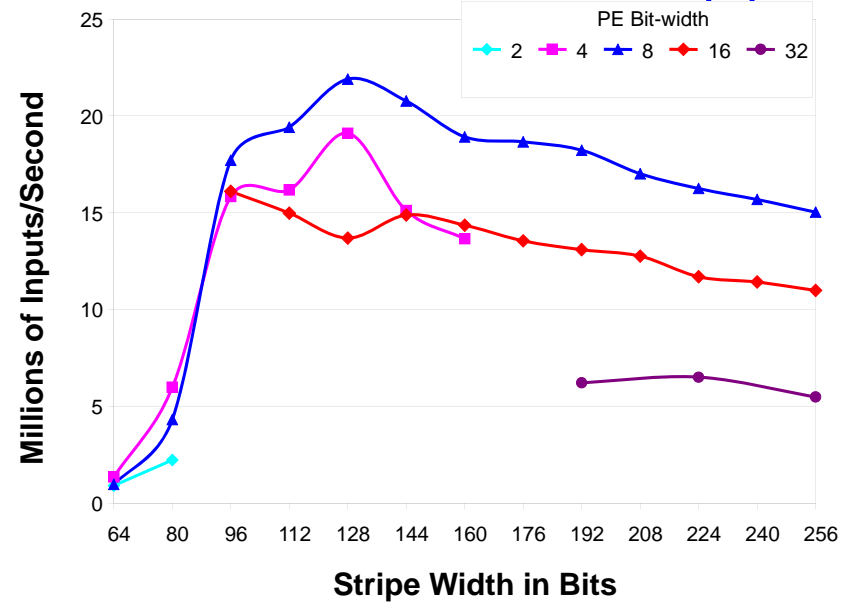
How Many Registers?



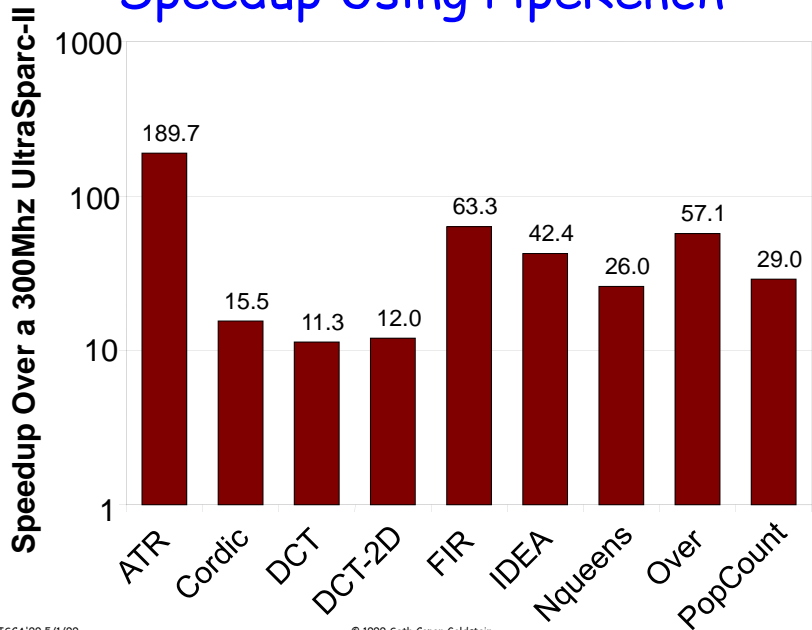
Throughput for IDEA



Harmonic Mean of Throughput

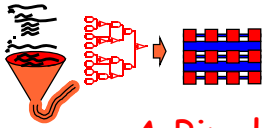


Speedup Using PipeRench



Sources of Performance

- Exploit multiple levels of parallelism
 - MIMD, SIMD
 - ILP
 - Pipeline
 - bit-level
- Custom function units
 - Custom sizes
 - Specialized functions
- Improved memory performance
- Data dependent hardware generation



PipeRench

A Pipelined Reconfigurable Fabric

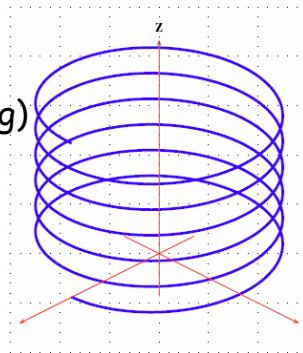
- Virtualizes the hardware
- Supports forward compatibility
- Oriented towards datapaths
- Effective configuration time of zero.
- Simple programming model
- Easy compilation target

Conclusions

- PipeRench matches Future Workloads
- Hardware Virtualization
 - Forward compatibility
 - Good performance
- Pipelined Reconfiguration
 - Simple programming model
- Hardware-Software Interaction
 - PE bit-width $b = 8$
 - Stripe Width $n*b = 128$
 - Registers per PE $r = 8$
- Space is Time

What Computer Architects Do

- Given Constraints of:
 - Technology
 - Application
- Use Essential Themes:
 - Exploit locality (AKA caching)
 - Prediction / Speculation
 - Pipelining
 - Parallelism
 - Virtualization / Indirection
 - Specialization
- But, there is progress



What We Need To Do

- Decrease design costs
- Improve design verification
- Manufacturing Cost
 - Eliminate mask costs
 - Decrease fab plants costs
 - Increase yield
- Keep compilation time constant
- Invent a new technology

Goal

- Programmable Logic Device with
 - $\geq 10^{10}$ gate-equivalents/ CM^2
 - ≤ 1 Watt/ CM^2
 - \leq nanocents per gate
 - $\geq 10^{11}$ ops/sec
- Replace ASIC

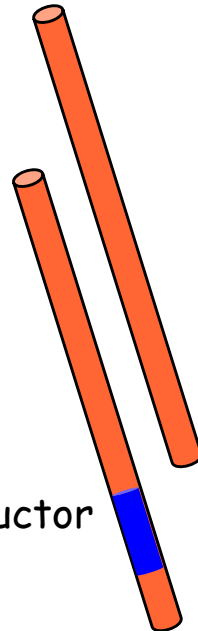
Less than $100nm^2$ for each gate and all its associated routing

Electronic Nanotechnology

- 1-30 nanometer features
- Uses electrons, currents, voltages to accomplish computing
- Can build the components we need (diodes, wires, resistors, latches, switches, and maybe transistors)
- Fabrication by **directed self-assembly**
- Fabrication method implies
 - defects
 - non-deterministic placement

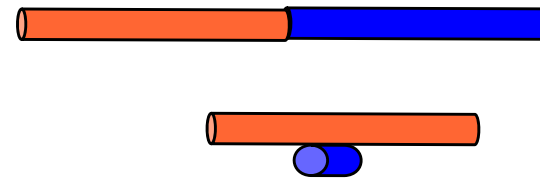
Wires

- 2 - 30 nm in diameter
- < 2000 nm in length
- good conductors
- excellent current densities
- can be built from
 - organics (carbon nantubes)
 - metals
- Can be both metal or semiconductor



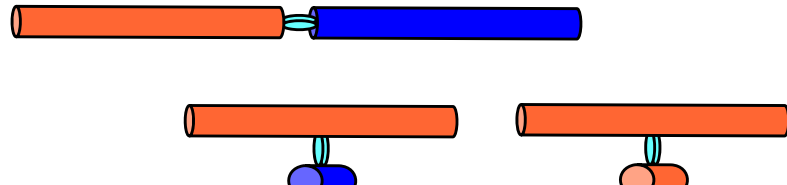
Diodes

- Conducts current in only one direction
- reasonable on/off ratios
- it has only 2 terminals
- Results from contact between 2 different materials



Configurable molecules

- Molecule that can be
 - conducting (ON)
 - insulating (OFF)
- Turn on with $V_{dd}+V_{config}$ Voltage drop
- Turn it off with $-V_{config}$



The switch holds its own state!

Synthesis primitives

- Wires can be grown
- Molecules can be made

What is missing:

- No end-to-end alignment of wires
- Nothing more complex than a 2-d mesh
- Gain
- No complexity!

Is it enough?

Underlying Technology

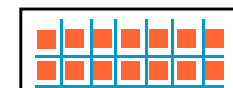
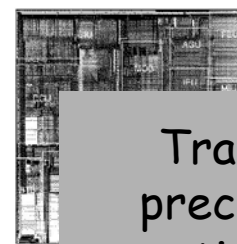
- Lots of wires
- Lots of switches
- All programmable

Scale implies:

- Defects
- Randomness

A case for Reconfigurable Devices

Where Complexity?



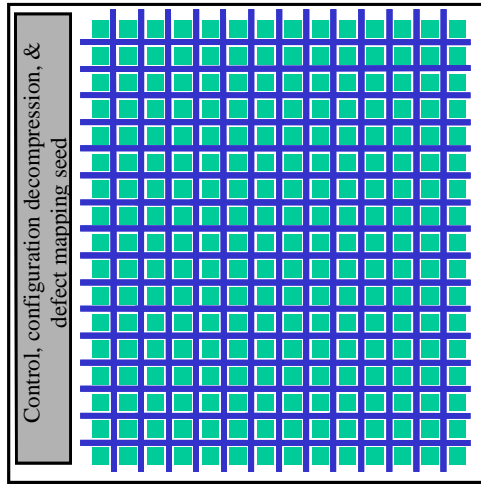
Tradeoff complexity (and precision) at manufacturing time for complexity at compilation time.

Complex fixed chip
+
Program

Regular, tileable structures
+
Configuration

The NanoFabric

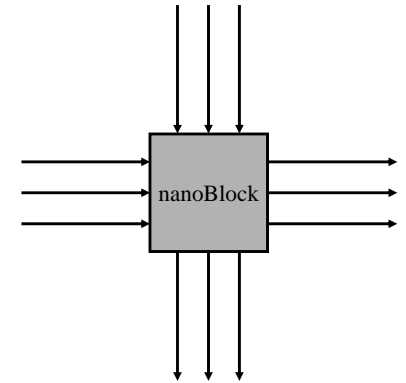
- nanoscale layer on top of CMOS
- Highly regular
- $\sim 10^6$ clusters
- $\sim 10^8$ long lines



Boring: **Just what we want!**

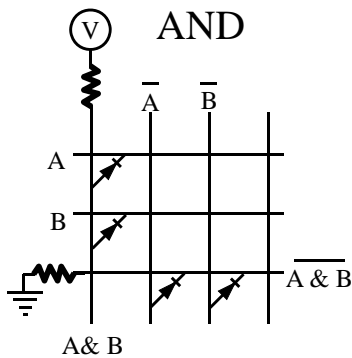
The NanoBlock

- Basic unit of logic
- Small # of I/Os
- Has an orientation
 - south-west
 - north-east
- Internally, a matrix of configurable diodes.
- Entire block is on a lithographic scale



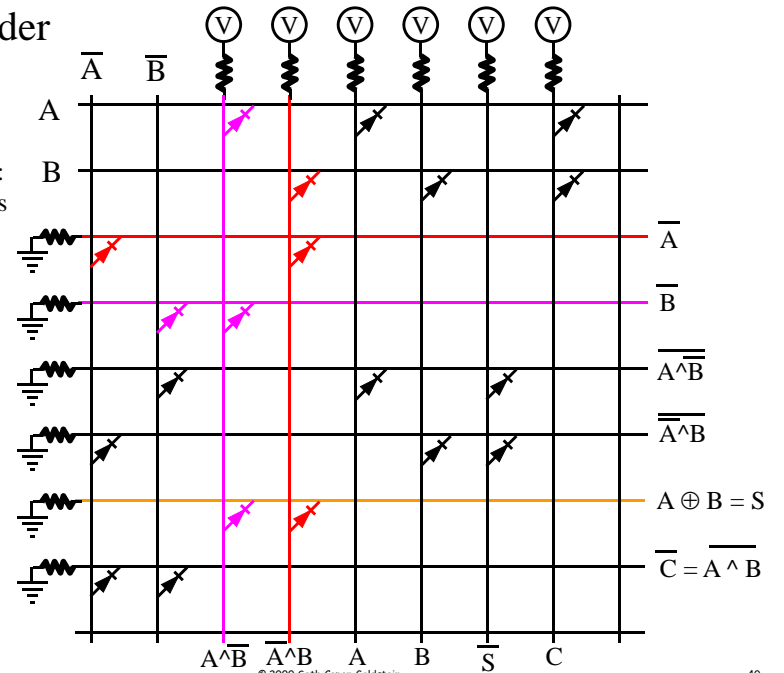
Computing with a nanoBlock

- The active component is a diode.
- Use diode-resistor logic



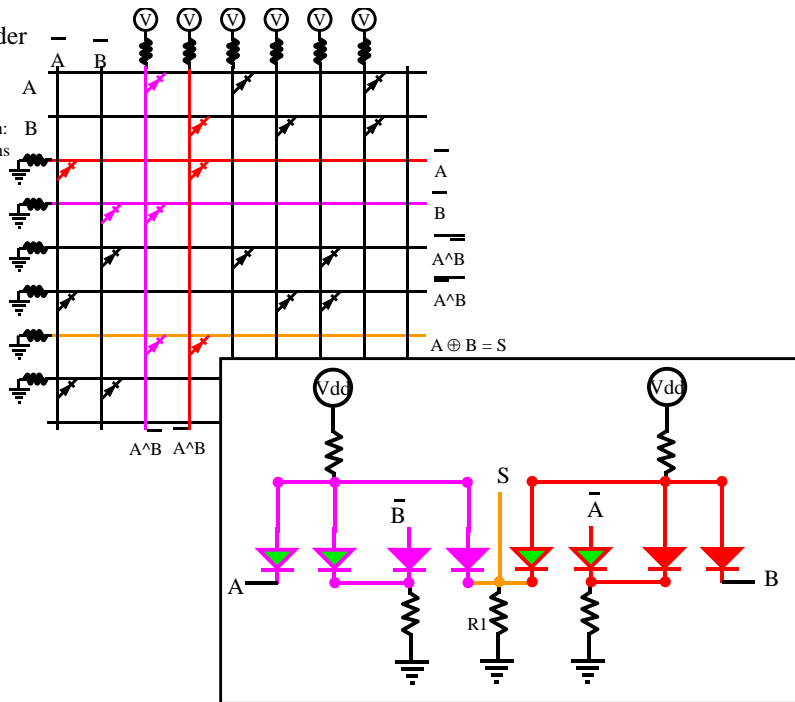
Half adder

$S = A \oplus B$
 $C = A \wedge B$
 critical path:
 S has 3 turns



Half adder

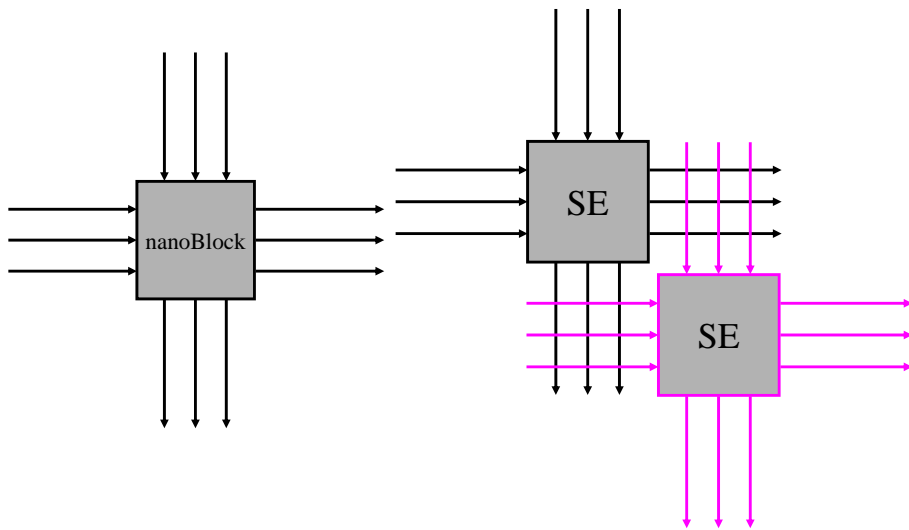
$S = A \oplus B$
 $C = A \wedge B$
 critical path:
 S has 3 turns



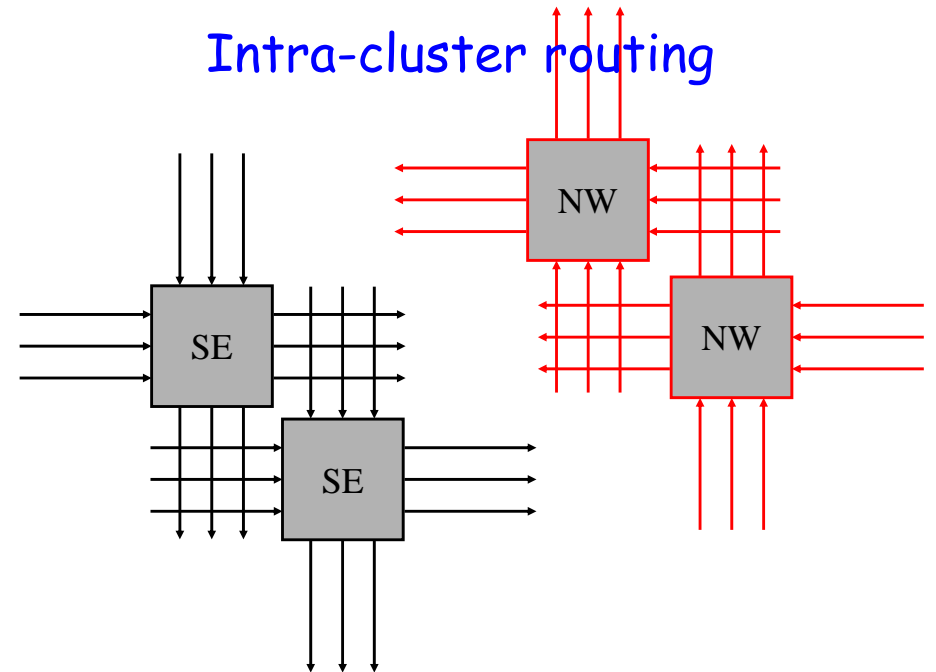
Requirements for the Matrix

- Good diodes
 e.g., low voltage drop
- Different resistor values
 - e.g., R & 10R, where $R \sim 10^9$ Ohms
 - low R wires
- reasonable current densities
- Using Spice (and guesses!):
 - switching at 100Mhz-1Ghz
 - nanoWatts

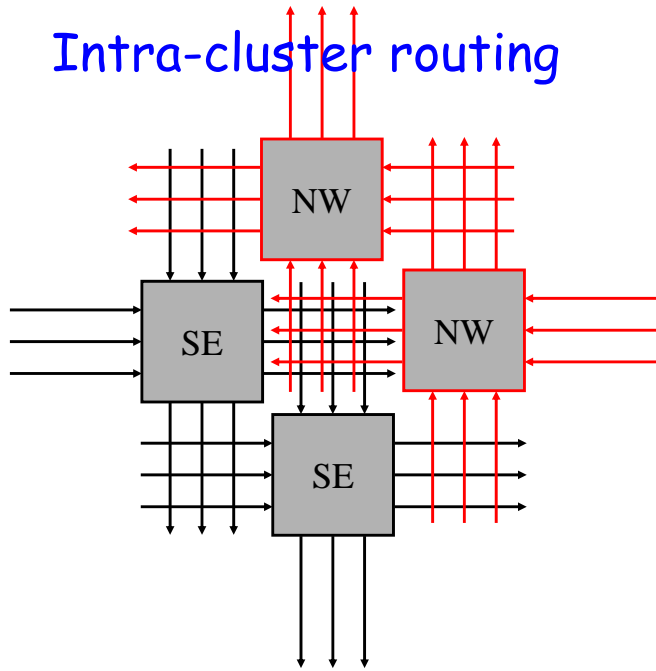
Intra-cluster routing



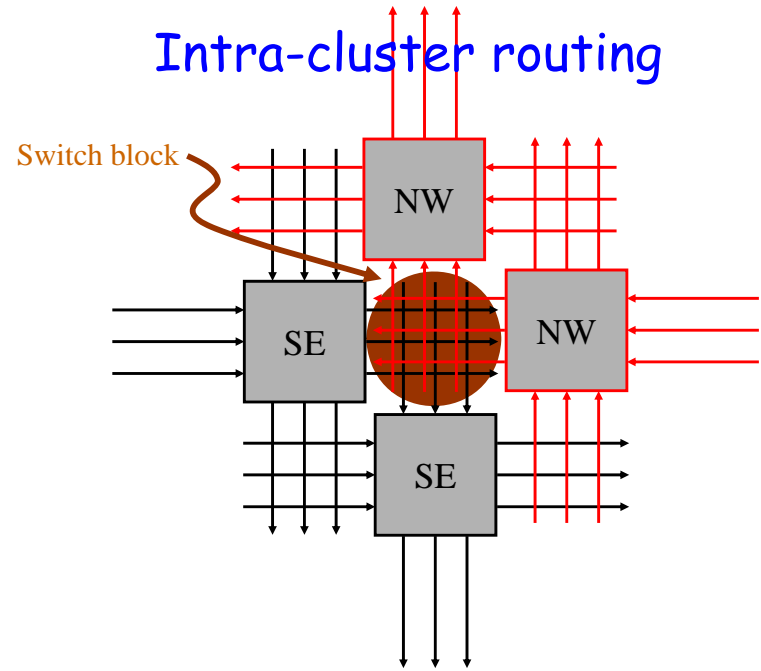
Intra-cluster routing



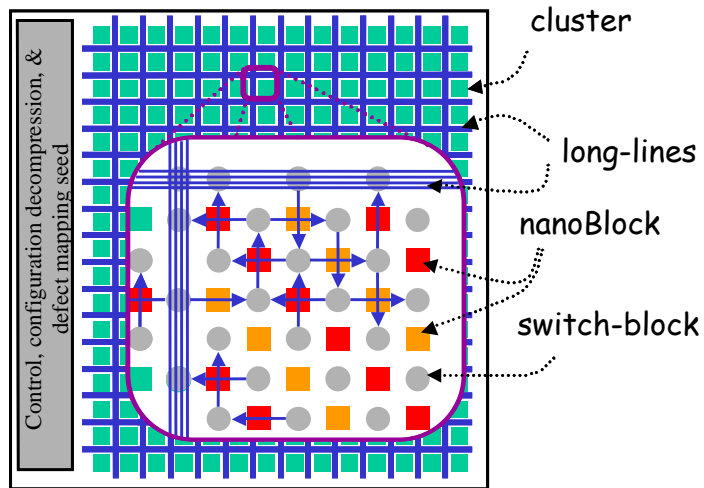
Intra-cluster routing



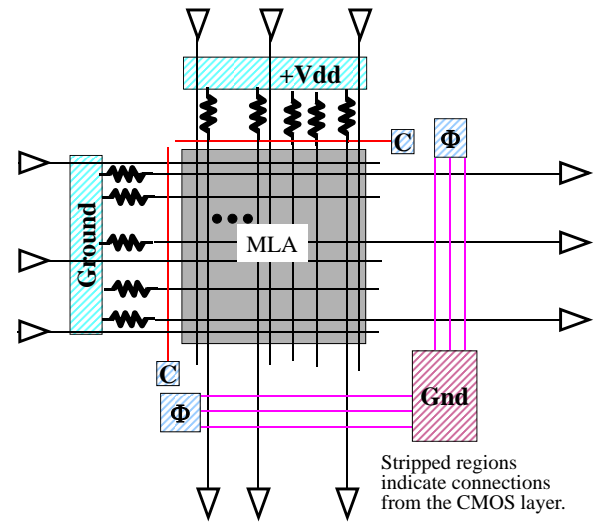
Intra-cluster routing



The NanoFabric



NanoBlock Buildable



Densities

- Assuming,
 - 100nm CMOS process
 - 20nm centers for nanowires
 - Blocks are 3 input, 8 internal wires
 - 20 long-lines per routing channel
 - Pretty poor diodes
- Blocks $1.6 \times 10^8 / \text{Cm}^2$
- Configuration bits $4.5 \times 10^{10} / \text{Cm}^2$
- Avg Configuration Time 200ms /Cm²
- Power (@10MHz) 2 Watt /Cm²

Now what

- Defect discovery
- Compilation

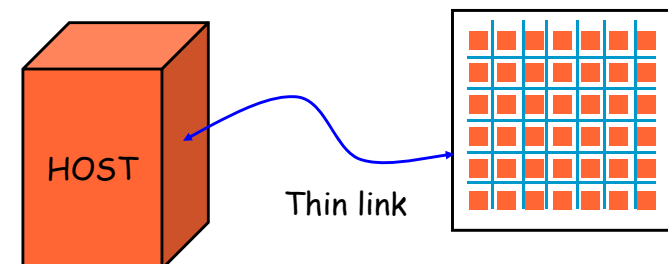
What Is a Defect?

- Defects in Manufacturing
 - broken wires
 - crossed wires
 - stuck-at switches
 - etc.
- Defects in Knowledge
 - unknown device
 - unknown characteristics of device

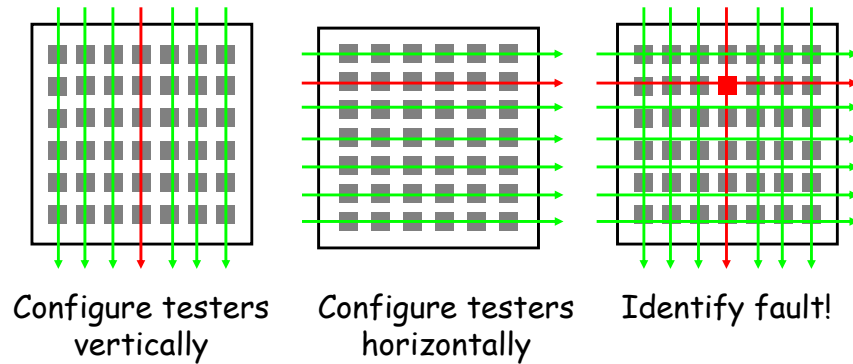
Think of this as Capability Mapping

Mapping the Device

- Need to discover the characteristics of the individual components, but
- Can't selectively stimulate or probe the components
- Download test machines



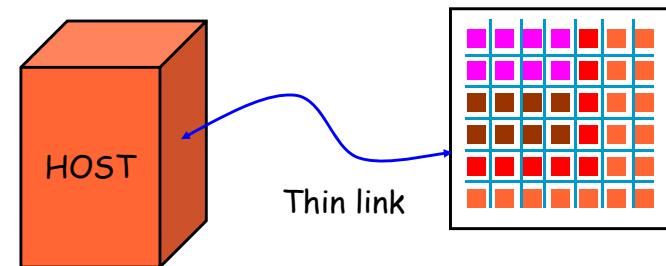
Built-In Self-Test



Configure the device to test itself!

Mapping the Device

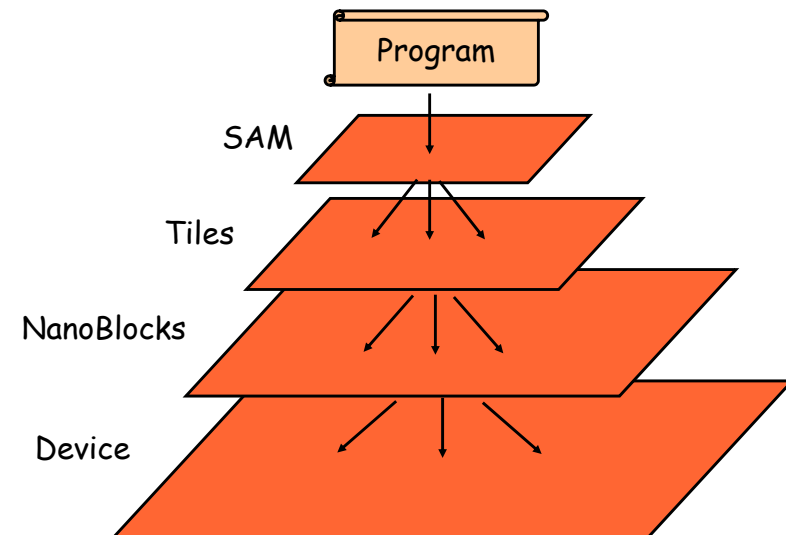
- Download signature generators
- Then, also detectors
- Then, replicators
- Will scale (at worst) linearly
- For proposed device ~1 day to map



Requirements to support defect tolerance

- Reconfigurable individual components
- Knowledge of
 - defect free architecture
 - potential characteristics of components
- Digital detection of Characteristics
- Localization of faults

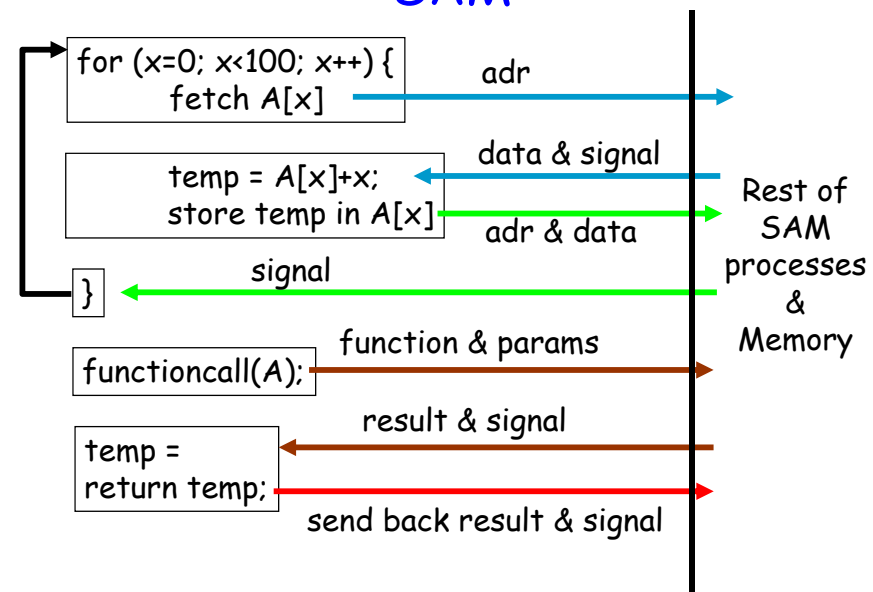
Architectural Abstractions



Split-phase Abstract Machine

- A collection of simple cooperating processes
- All potentially long-latency operations are split-phase
 - procedure calls
 - memory operations

SAM

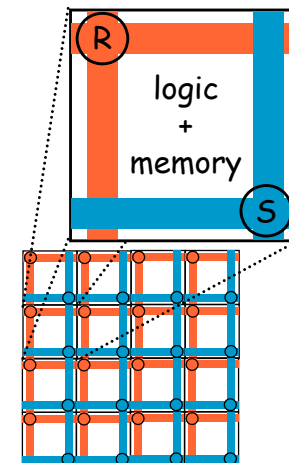


SAM

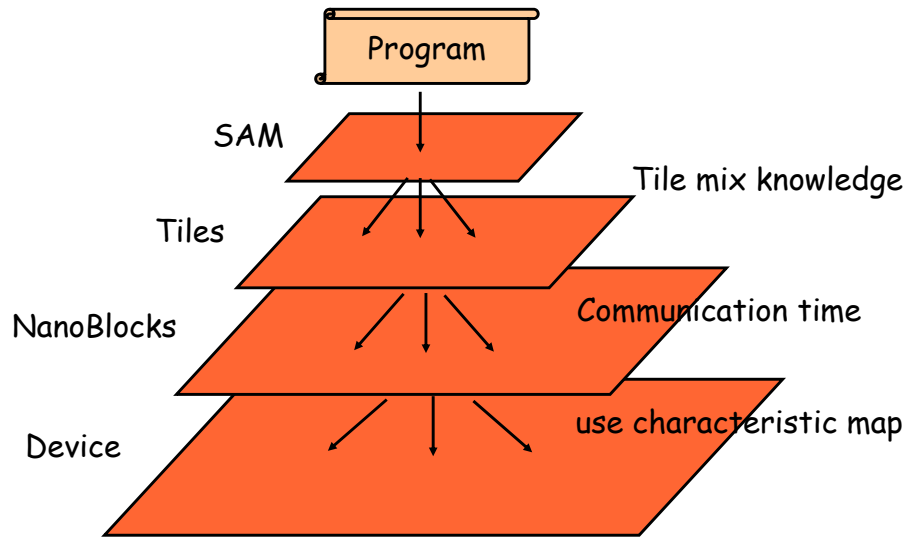
- **Allows compilers to avoid global picture.**
- Eliminates need for central control.
- Supports long wires.
- **Supports memory hierarchy.**
- Can be extended to deal with single-event upsets.

Tile Machine

- Map each SAM process to a simple "processor"
 - simple FSM
 - customized circuit
 - local memory
 - static network switch
 - packet network router



Architectural Abstractions



Compilation

- Separate out into SAM processes
- Localize memory
- Partition
- Map to tiles (local place & route)
- Place and route tiles
- Map tiles to NanoBlocks

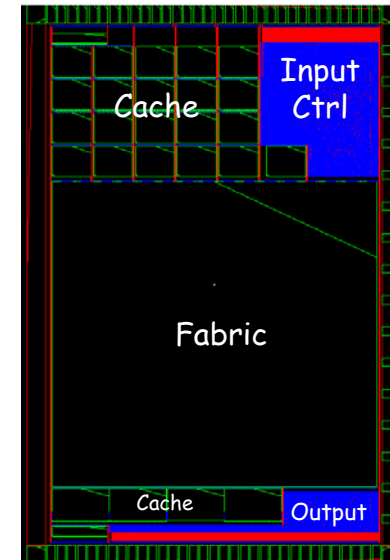
It is sufficient!

- Matrix components:
 - Diodes
 - different resistors
 - different length wires
- Gain provided by latches
- Connections to CMOS wires
- <5% defect rate
- **Reconfigurable**

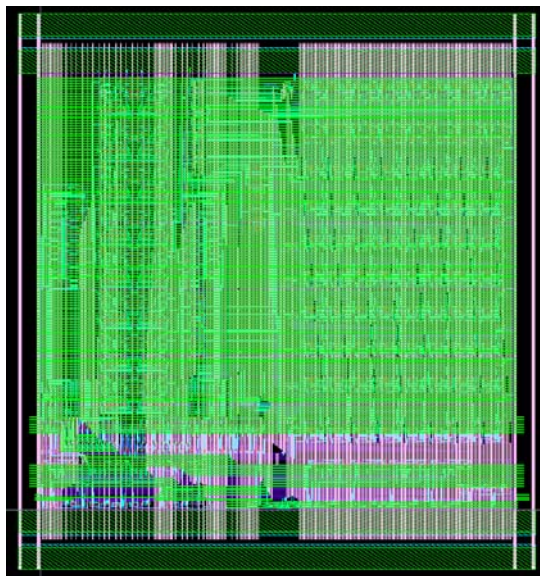
Conclusions

- Can build regular structures
- Exploits reconfigurability to provide
 - Built-in self test
 - Custom circuits
- nanoFabric is scalable
 - Low power
 - MHz
 - Tera-components

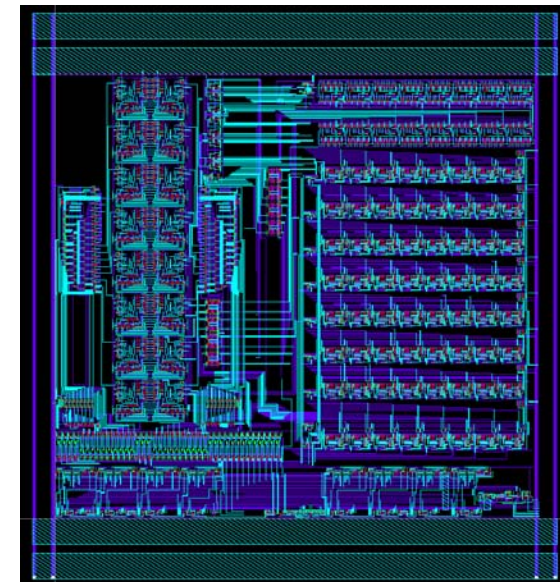
Chip Layout



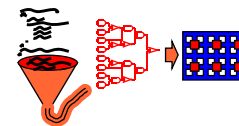
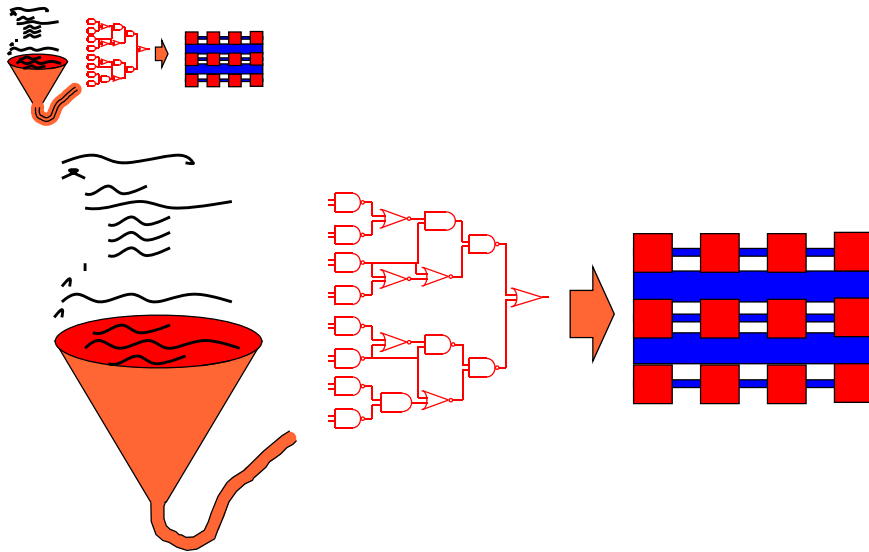
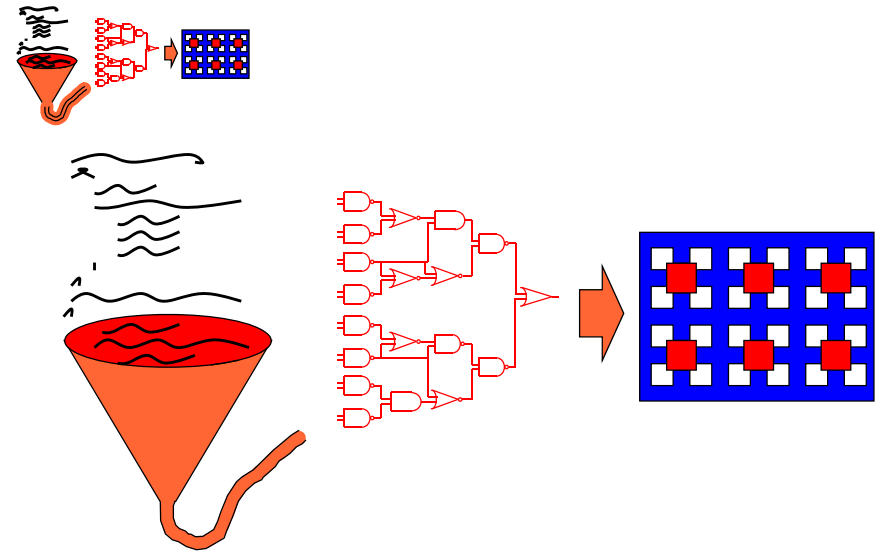
PipeRench Tile



Without M3 or M4



Rest of slides should be ignored

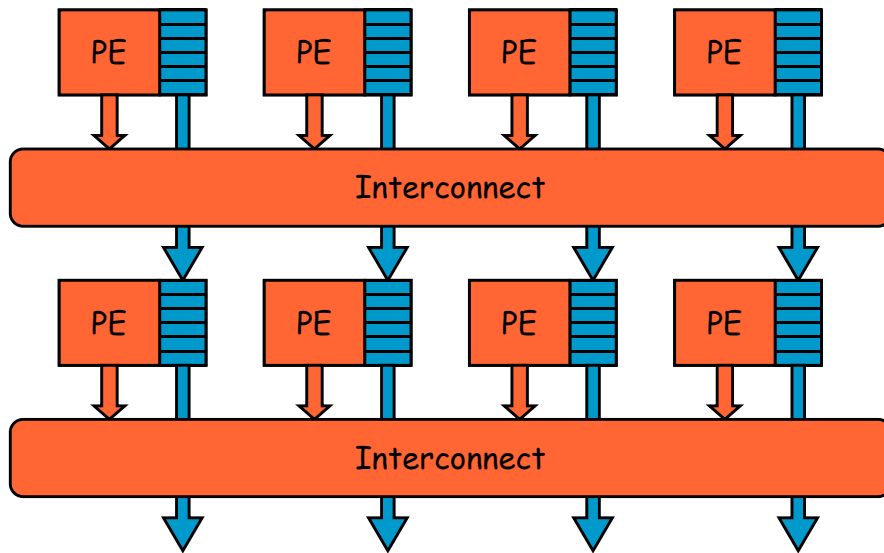


Why Not?

Field Programmable Gate Arrays are not suited for general-purpose custom hardware.

- Fixed resources
- No forward compatibility
- Oriented towards glue-logic
- Long configuration times
- No programming model
- Poor Development Environment

The PipeRench Fabric



ISCA'99 5/1/99

© 1999 Seth Copen Goldstein

73

Why Not?

	FPGAs	PipeRench
Orientation	Glue-logic	Datapaths
Configuration Time	Long (> 100 ms)	Almost zero
Resources	Fixed	Virtual
Forward Compatibility	NO	Yes
Programming Model	None	Pipelines
Development Environment	Poor	Good

ISCA'99 5/1/99

© 1999 Seth Copen Goldstein

74

Future Requirements

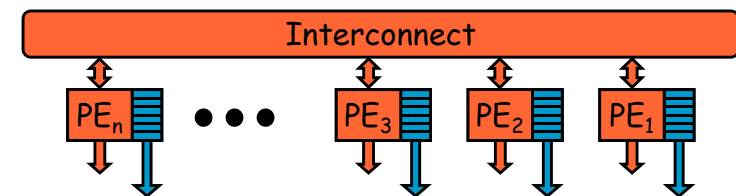
- Embedded Computing
 - COTS part
 - Easy programming model
 - Field programmable
 - Hardware performance
 - General-Purpose Computing
 - Increased Performance
 - Respect the memory bandwidth gap
 - Both
 - Variable bit width operations
 - Exploit all levels of parallelism
 - Use replication
- ↓ Manufacturing Cost
 ↓ Design Cost
 ↑ Flexibility

ISCA'99 5/1/99

© 1999 Seth Copen Goldstein

75

Finding the Best Instance



- Number of PEs per stripe n
- Bit width of each PE b
- The number of registers per PE p
- The type of interconnect
- The internals of a PE

ISCA'99 5/1/99









© 1999 Seth Copen Goldstein

76

Space is Time

- Larger configurations
 - ⇒ more virtualization
 - ⇒ lower throughput
- Tradeoff between circuit delay and circuit size is direct
- E.g., Faster carry chains
 - ⇒ wider additions per clock cycle
 - ⇒ fewer stripes
 - ⇒ better performance

PE width tradeoffs

	Narrow	Wide
Utilization		
Carry Chain Speed		
Configuration Size		
Interconnect Flexibility		
Ease of Compilation	