CS294-059

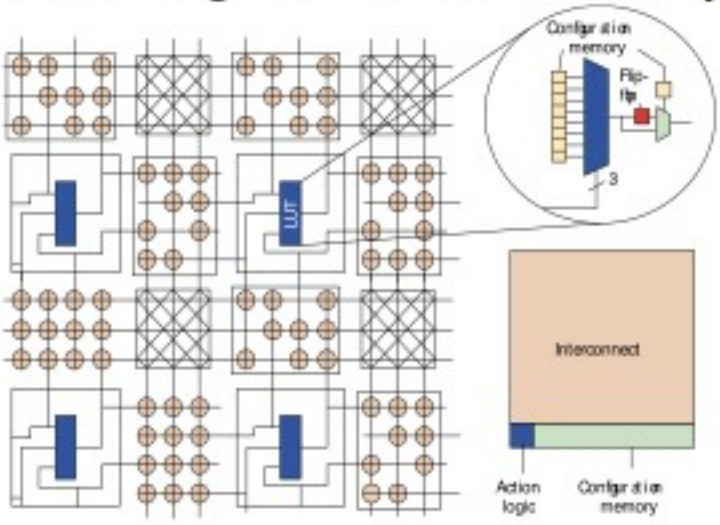
Advanced Topics in Hardware Design

The Technology and Business of FPGAs

Fall Semester 2010
John Wawrzynek
with
John Lazzaro

Typical Simple Field Programmable Gate Array

- Two-dimensional array of simple logic- and interconnection-blocks.
- LUTs implement any function of n-inputs (n=3 in this case).
- Optional Flip-flop with each LUT.



- Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Most FPGAs include special circuits to accelerate adder carry-chain.

Why are FPGAs Interesting?

- Technical viewpoint:
  - For hardware/system-designers, like ASICs only better! "Tape-out" new design every few minutes/hours.
  - Does the "reconfigurability" or "reprogrammability" offer other advantages over fixed logic?
  - Dynamic reconfiguration? In-field reprogramming? Self-modifying hardware, evolvable hardware?

Why are FPGAs Interesting?

- Questioning our assumptions about computing:

"Are we making copies in sub-micron CMOS VLSI of copies in NMOS of copies in TTL of early vacuum tube computer designs?"

A. DeHon

- 100000x increase in single-chip silicon capacity changes the underlying design costs.
  - Von Neumann architectures were designed to heavily time-multiplex the expensive ALU resource.

General-purpose computing machines don't have to look like processors.

Why are FPGAs Interesting?

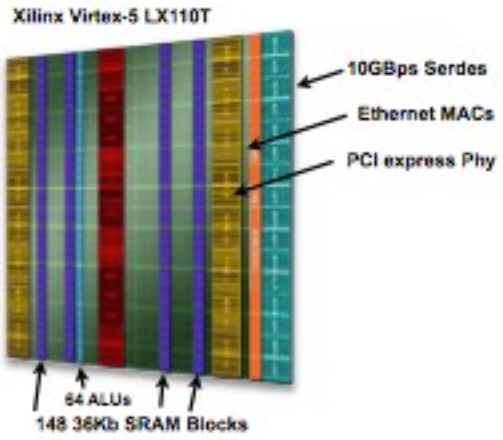
- Staggering logic capacity growth (10000x):

Year Introduced	Device	Logic Cells	"logic gate equivalents"
1985	XC2064	128	1024
2011	XC7V2000T	1,954,560	15,636,480

- FPGAs have tracked Moore's Law better than any other programmable device.

Why are FPGAs Interesting?

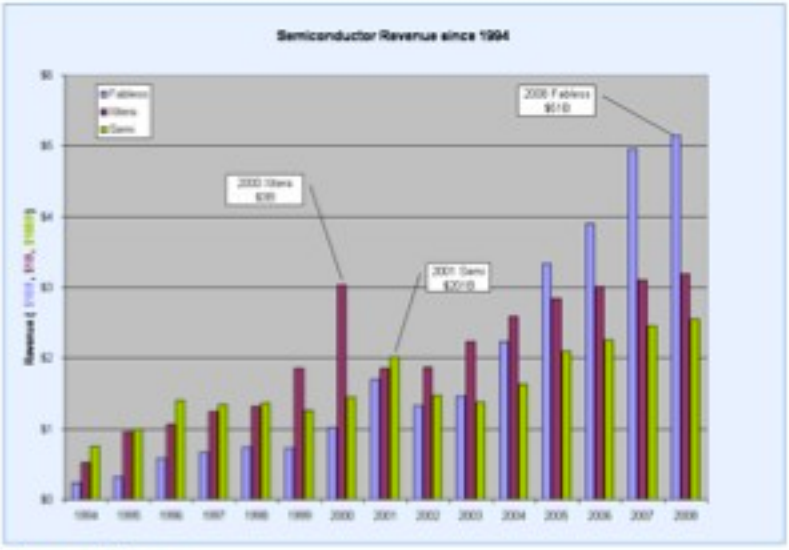
- Logic capacity now only part of the story: on-chip RAM, high-speed I/Os, "hard" function blocks, ...
- Modern FPGAs are "reconfigurable systems"



But, the heterogeneity erodes the "purity" argument. Mapping is more difficult. Introduces uncertainty in efficiency of solution.

Why are FPGAs Interesting?

- Have been an archetype for the semiconductor industry as a whole:



from: mattrhodes.net

Putting the FPGA Business in Perspective. How large is it compared to others?

	Q3 2009	Q4 2009	Q/Q Growth	Q1 2010 (Guidance)
Broadcom	$1,194,745	$1,283,434	7.4%	Up 0 to 5%
Marvell	$803,098	<TBD>		
Nvidia	$903,206	$982,500	8.8%	Flat
Xilinx	$414,950	$513,300	23.7%	down 1% to up 3%
Altera	$286,612	$365,000	27.3%	up 5 - 10%
TI	$2,880,000	$3,005,000	4.3%	down 2% - up 6%
INTEL	$9,389,000	$10,600,000	12.9%	down 5-10%
AMD	$1,396,000	$1,646,000	17.9%	down "seasonally"
Qualcomm	$1,699,000	$1,608,000	-5.4%	Flat
Atheros	$156,641	$185,700	18.6%	up 5%
Silicon Labs	$125,913	$127,200	1.0%	Flat to down 5%
Average			5.5%	

from: mattrhodes.net

Why are FPGAs Interesting?

- Have attracted an huge amount of investment for new ventures:

Why are FPGAs Interesting?

- Have attracted an huge amount of investment for new ventures:
  - Most startups have failed. Why?
  - Business dominated by "Xitera"

Worldwide FPGA/PLD vendor revenues and rankings, 2007-2008

Rank 2007	Rank 2008	Company	Revenue ($M) 2007	Revenue ($M) 2008	Revenue Change 2007-2008	Market Share 2008
1	1	Xilinx	1,809	1,906	5.4%	51.2%
2	2	Altera	1,216	1,323	8.8%	35.5%
3	3	Lattice Semiconductor	229	222	-3.1%	6.0
4	4	Actel	196	218	11.2%	5.9%
6	5	QuickLogic	28	23	-17.9%	0.6%
5	6	Cypress Semiconductor	32	21	-34.4%	0.6%
7	7	Atmel	14	9	-35.7%	0.2%
8	8	Chengdu Sino Microelectronics System	4	3	-25.0%	0.1%
		Others	0	0	NM	0.0%
		Total Market	**3,528**	**3,725**	**5.6%**	**100.0%**

Source: Gartner

Why are FPGAs Interesting?

- FPGAs at the leading edge of IC processing:
  - Xilinx V7 out next year with 28nm TSMC processing
  - Foundaries like FPGAs - regularity help get process up the "learning curve"
  - High-volume commitment gets interest of foundry
  - (Gives FPGAs a competitive edge over ASICs, which usually are built on an older process.)

Why are FPGAs Interesting?

- FPGAs have been wildly successful even though they are inefficient in silicon area, energy, and performance :
  - "Measuring the Gap Between FPGAs and ASICs", Ian Kuon and Jonathan Rose, FPGA'06
  - Versus ASICs: area 40X, delay 3-4X, power 12X
- How can this be? Is there something more important than silicon efficiency?

Why this course?

- *For computer architects*, FPGA are a great prototyping platform. The "killer app" (RAMP).
- *Reconfigurable computing research.* Is their a better (more efficient) way to build general purpose computing devices? Can RC help the power problem? The parallel programming problem?
- *FPGAs are here to stay* and have a bright future. Fewer and fewer ASIC starts. Process variability and proposed new processing technologies favor regularity at the chip architecture level.
- *Industry needs our help* with their roadmap - applications, chip architecture, tools, ...
- *For us future entrepreneurs: What* are the lessons from the 50+ startups?

Course Structure

- Discussion based. Assigned reading material, followed by class discussion.
- Website has topics and schedule (note reading for next time).
- Partial list of guest speakers:

Steve Trimberger, Xilinx Fellow
Mike Hutton, Director of Hardware Architecture at Tabula
Altera Representative, TBD
Jonathan Greene, Actel Fellow
Rajit Manohar, Founder and Chief Scientist at Achronix
Carl Ebeling, Former CTO M2000/Abound

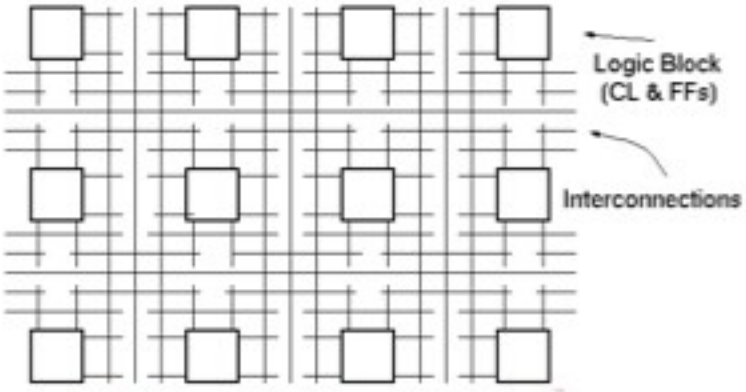
- Grading: ~30% Reading Summaries and Class Participation, ~70% project
- Unit option? "No-project" option?

Course Project Requirement

- Examples:
  - Take one the the (failed) startups, dig into their technology, marketing, and business plans in general. Do an analysis and report (called "due diligence" by the investors). Now with the benefit of hindsight, go a step further and make recommendations on how to fix the plan.
  - Propose a new FPGA or FPGA-ecosystem company from scratch, including a business plan and pitch. The rest of us, then perform the "due diligence".
- Open to proposals

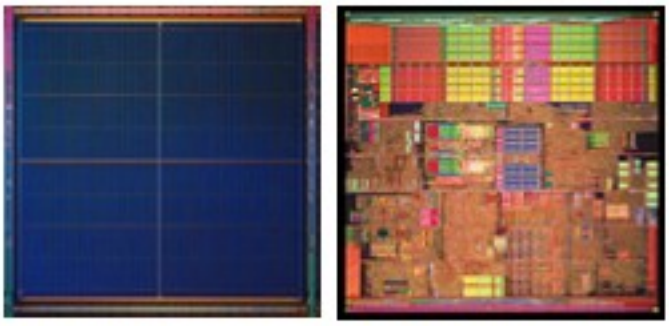
FPGA Overview

- Basic idea: two-dimensional array of logic blocks and flip-flops with a means for the user to configure (program):
  1. the interconnection between the logic blocks,
  2. the function of each block.



Simplified version of FPGA internal architecture:

Die Photos: Virtex FPGA vs. Pentium IV



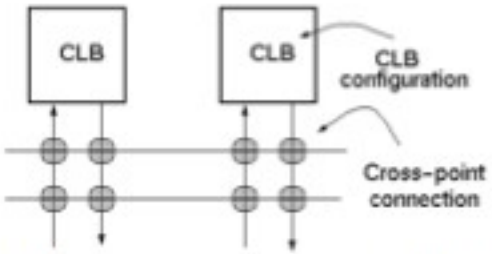
- FGPA Vertex chip looks remarkably structured
  - Very dense, very regular structure
- "Full-Custom" Pentium chip somewhat more random in structure
  - Large on-chip memories [caches] are visible

FPGAs are in widespread use

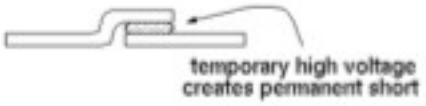
Far more designs are implemented in FPGA than in custom chips.

FPGA Variations

- Families of FPGA's differ in:
  - physical means of implementing user programmability,
  - arrangement of interconnection wires, and
  - the basic functionality of the logic blocks.
- Most significant difference is in the method for providing flexible blocks and connections:



- Anti-fuse based (ex: Actel)

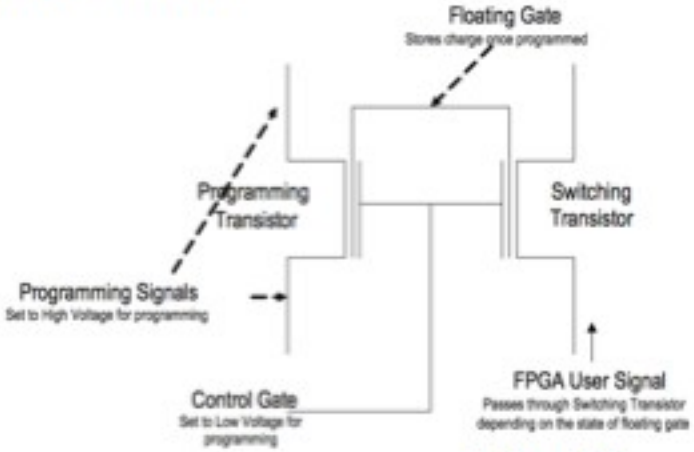


+ Non-volatile, relatively small
– fixed (non-reprogrammable)

• Several "floating gate" or eprom style approaches have been used. One now by Actel.

FPGA Variations

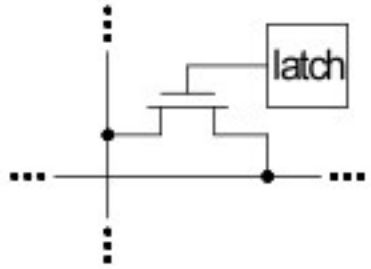
- "Floating-gate" / EPROM / FLASH based (ex: Actel, others)



+ Non-volatile
+ reprogrammable
– larger size then anti-fuse
– requires special process

User Programmability

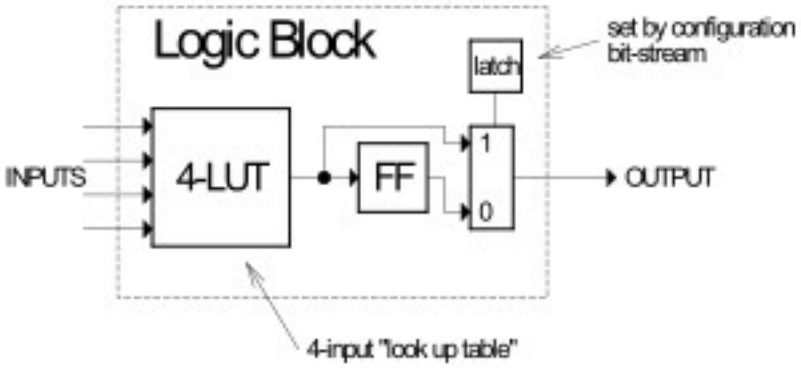
- Latch-based (Xilinx, Altera, ...)



+ reconfigurable
– volatile
– relatively large.

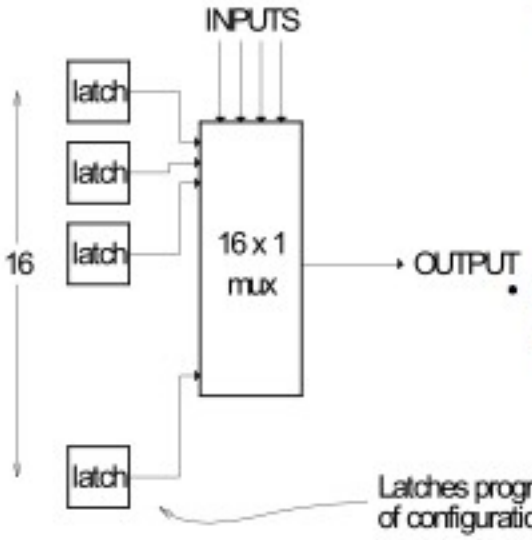
- Latches are used to:
  1. control a switch to make or break cross-point connections in the interconnect
  2. define the function of the logic blocks
  3. set user options:
     - within the logic blocks
     - in the input/output blocks
     - global reset/clock
- "Configuration bit stream" is loaded under user control

Idealized FPGA Logic Block



- 4-input look up table (LUT)
  – implements combinational logic functions
- Register
  – optionally stores output of LUT

4-LUT Implementation



- n-bit LUT is implemented as a 2^n x 1 memory.
  - inputs choose one of 2^n memory locations.
  - memory locations (latches) are normally loaded with values from user's configuration bit stream.
  - Inputs to mux control are the CLB inputs.
- Result is a general purpose "logic gate".
  - n-LUT can implement any function of n inputs!

LUT as general logic gate

- An n-lut as a direct implementation of a function **truth-table**.
- Each latch location holds the value of the function corresponding to one input combination.

Example: 2-lut

INPUTS	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

• • •

Implements *any* function of 2 inputs.

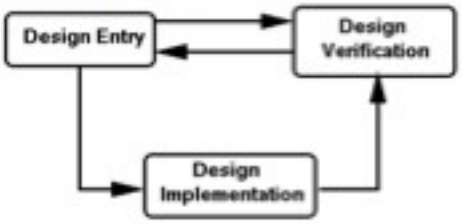
How many of these are there?

How many functions of n inputs?

Example: 4-lut

INPUTS	
0000	F(0,0,0,0) ← store in 1st latch
0001	F(0,0,0,1) ← store in 2nd latch
0010	F(0,0,1,0) ←
0011	F(0,0,1,1) ←
0011	
0100	•
0101	•
0110	•
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

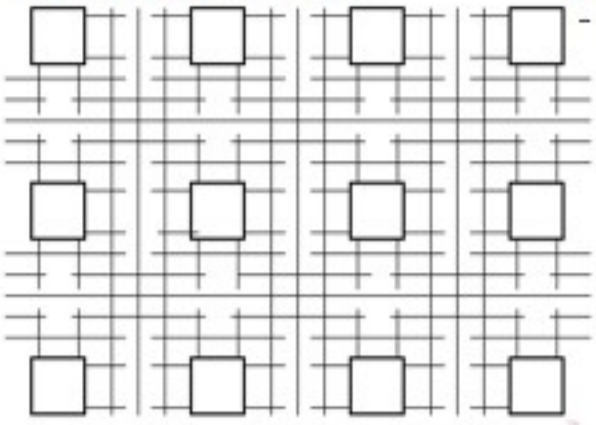
FPGA Generic Design Flow



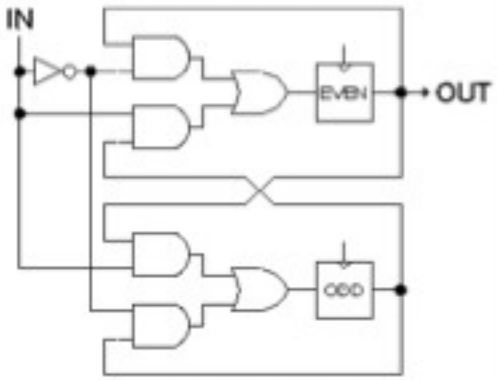
- Design Entry:
  - Create your design files using:
    - schematic editor or
    - HDL (hardware description languages: Verilog, VHDL)
- Design Implementation:
  - Logic synthesis (in case of using HDL entry) followed by,
  - Partition, place, and route to create configuration bit-stream file
- Design verification:
  - Optionally use simulator to check function,
  - Load design onto FPGA device (cable connects PC to development board), optional "logic scope" on FPGA
    - check operation at full speed in real environment.

Example Partition, Placement, and Route

- Idealized FPGA structure:



- Example Circuit:
  - collection of gates and flip-flops



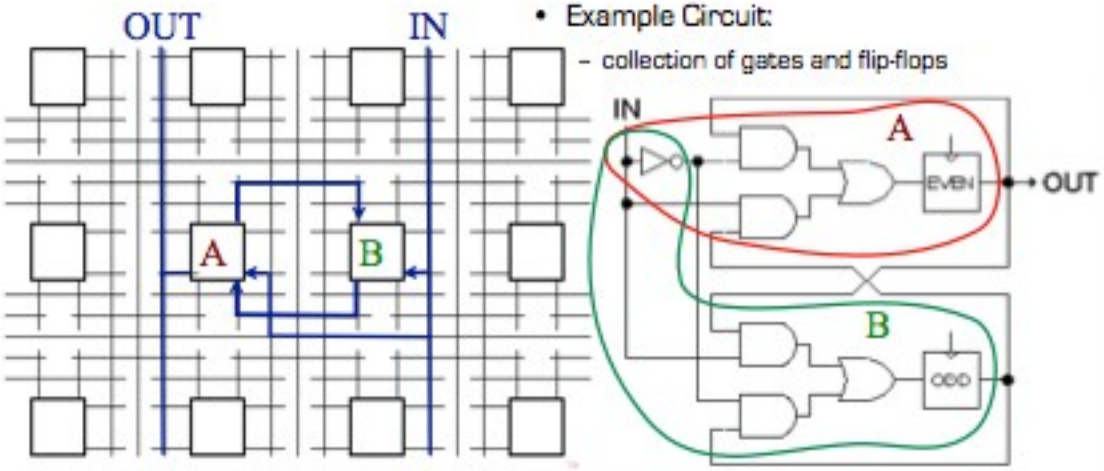
Circuit combinational logic must be "covered" by 4-input 1-output LUTs.

Flip-flops from circuit must map to FPGA flip-flops.
(Best to preserve "closeness" to CL to minimize wiring.)

Best placement in general attempts to minimize wiring.

Vdd, GND, clock, and global resets are all "prewired".

Example Partition, Placement, and Route

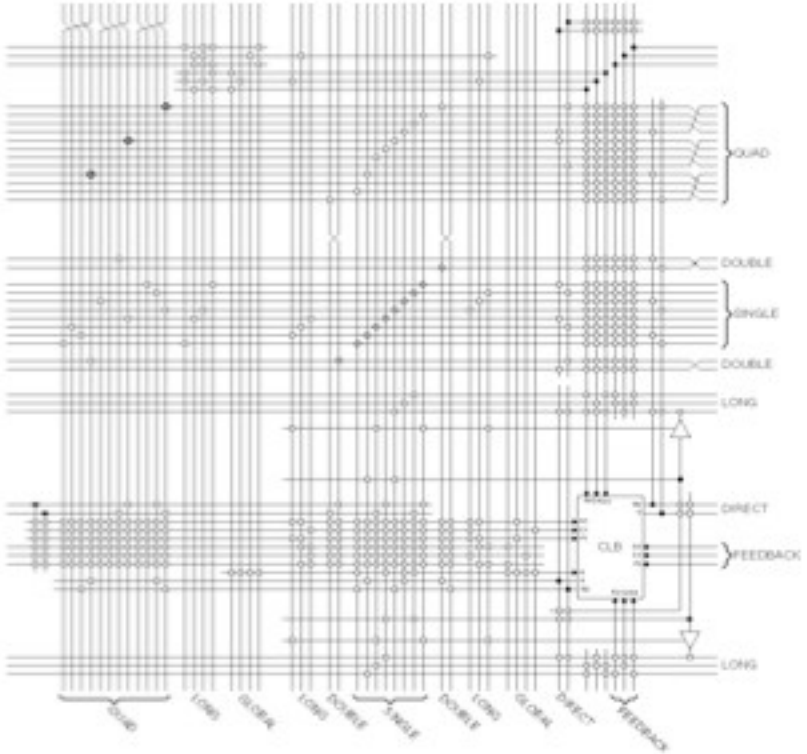


- Example Circuit:
  - collection of gates and flip-flops

Two partitions. Each has single output, no more than 4 inputs, and no more than 1 flip-flop. In this case, inverter goes in both partitions.

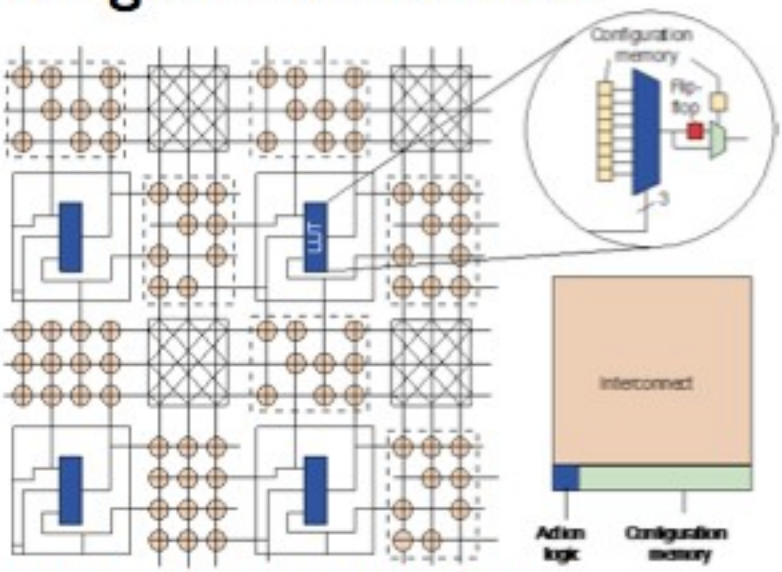
Note: the partition can be arbitrarily large as long as it has not more than 4 inputs and 1 output, and no more than 1 flip-flop.

Xilinx FPGAs (interconnect detail)



Field Programmable Gate

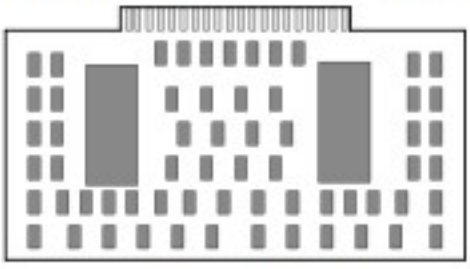
- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: LUTs implement any function of n-inputs (n=3 in this case).
- Optional Flip-flop with each LUT.



- Fuses, EPROM, or Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Modern FPGAs include special circuits to accelerate adder carry-chain and many

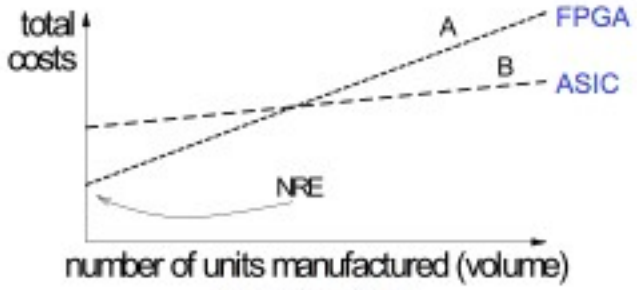
Why FPGAs?

- By the early 1980's most of the logic circuits in typical systems where absorbed by a handful of standard large scale integrated circuits (LSI).
  - Microprocessors, bus/IO controllers, system timers, ...
- Every system still had the need for random "glue logic" to help connect the large ICs:
  - generating global control signals (for resets etc.)
  - data formatting (serial to parallel, multiplexing, etc.)
- Systems had a few LSI components and lots of small low density SSI (small scale IC) and MSI (medium scale IC) components (used as "glue logic").



Why FPGAs?

- Custom ICs where sometimes designed to replace the large amount of glue logic:
  - reduced system complexity and manufacturing cost, improved performance.
  - However, custom ICs are relatively very expensive to develop, and delay introduction of product to market (time to market) because of increased design time.
- Note: need to worry about two kinds of costs:
  1. cost of development, sometimes called non-recurring engineering (NRE)
  2. cost of manufacture
  - A tradeoff usually exists between NRE cost and manufacturing costs

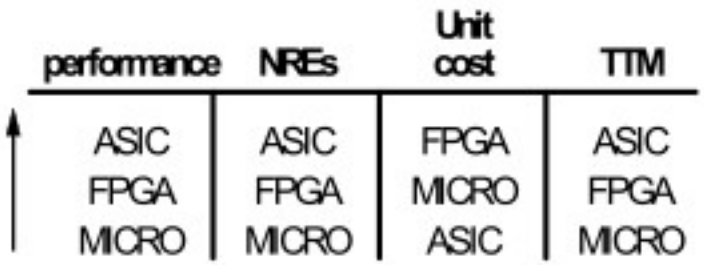


Why FPGAs?

- Therefore the custom IC approach was only viable for products with very high volume (where NRE could be amortized), and which were not time to market (TTM) sensitive.
- FPGAs were introduced as an alternative to custom ICs for implementing glue logic:
  - improved density relative to discrete SSI/MSI components (but not as good as custom ICs)
  - with the aid of computer aided design (CAD) tools circuits could be implemented in a short amount of time (no physical layout process, no mask making, no IC manufacturing), relative to ASICs.
    - lowers NREs
    - shortens TTM
- Because of Moore's law, the density (gates/area) of FPGAs continued to grow through the 80's and 90's to the point where major data processing functions can be implemented on a single FPGA.

Why FPGAs?

- FPGAs continue to compete with custom ICs for special processing functions (and glue logic) but now also compete with microprocessors in dedicated and embedded applications.
  - Performance advantage over microprocessors because circuits can be customized for the task at hand. Microprocessors must provide special functions in software (many cycles).
  - The flexibility of a software processor. The efficiency of an ASIC (well sort of!).

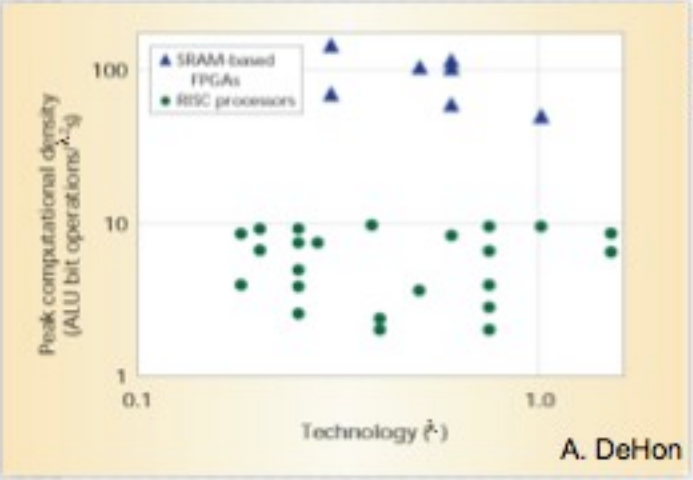


Implementation Alternatives

Full-custom:	All circuits/transistors layouts optimized for application.
Standard-cell:	Arrays of small function blocks (gates, FFs) automatically placed and routed.
Gate-array (structured ASIC):	Partially prefabricated wafers customized with metal layers or vias.
FPGA:	Prefabricated chips customized with loadable latches or fuses.
Microprocessor:	Instruction set interpreter customized through software.
Domain Specific Processor:	Special instruction set interpreters (ex: DSP, NP, GPU).

By "ASIC", most people mean "Standard-cell" based implementation.

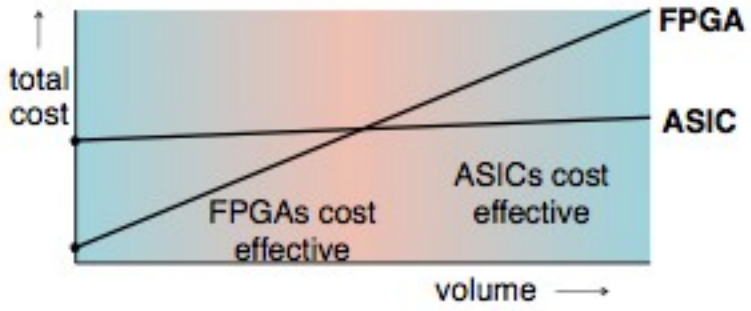
Advantages of RC versus Processors



- Conventional processors have several sources of inefficiency:
  - Heavy time-multiplexing of Function Units (ALUs).
  - Instruction issue overhead.
  - Memory hierarchy to deal with memory latency.
  - Operator mismatch

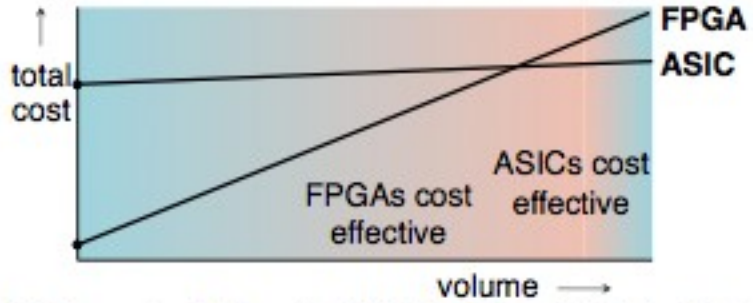
Peak (raw) performance

Traditional FPGA versus ASIC



- **ASIC:** High NRE costs ($2M for 0.35um chip). Relatively Low cost per die.
- **FPGAs:** Very low NRE costs. Relatively low silicon efficiency ⇒ high cost per part.
- **Cross-over volume** from cost effective FPGA design to ASIC in the 10K range.

Cross-over Point is Moving Right

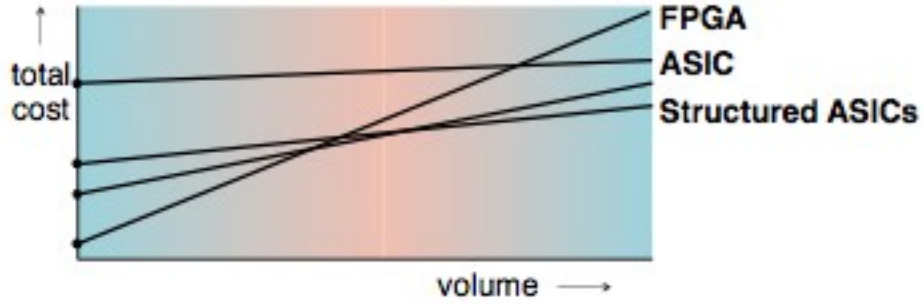


- **ASIC:** Increasing NRE costs ($40M for 90nm chip[1]) (verification, mask costs[2], etc.)
  - Fewer **silicon designs** becomes inevitable.
- **FPGAs:** Move in to fill the need, furthermore, FPGAs better able to follow Moore's Law, relatively cheaper to test.
- Cross-over volume now >100K range.

[1] Vahid Manian, VP manufacturing and operations, Broadcom Corp.

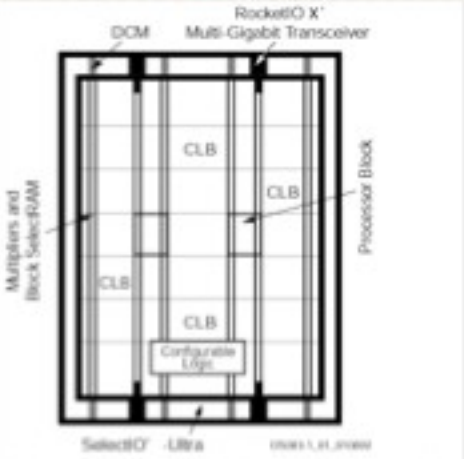
[2] Roger Minear, Agere Systems Inc, 30- 35- layer mask set ≈$650,000 for 130nm and $1.4M for 90nm.

Post-fabrication Customization



- **Gate Array** like devices (structured ASICs) return to fill the gap. Post-fab customization with limited mask layers or direct-write e-beam.
  - Lower NREs than ASICs, more silicon efficiency than FPGAs.

FPGA turned into CSOC (configurable system on a chip)



Virtex-II Pro X Generic Architecture Overview

Xilinx Virtex-II Pro 100+

- ~100K logic blocks, each with 4-LUT and Flip-flop (8 Million "system" gates)
- 1 MBytes SRAM bits
- 444 18x18bit dedicated multipliers
- 20 10-Gbit/s serial communication links

  Good for board to board connections.
- ~1000 user I/Os (most with LVDS 600 Mb/s signaling) Good from inter-chip communication and memory interface.
- 2 embedded hard PowerPC cores
- 10-20 GFLOPs (single precision) sustained, Itanium 2 1.6GFLOPs
- ~200 (16-bit) GOPS

FPGAs are **Re**configurable

Seemingly obvious point but …

1. Volume/cost graphs don't accurately capture the potential real costs and other advantages.
2. Commercial applications have not taken advantage of reconfigurability
   - Xilinx/Altera haven't done much to help.
   - Methodologies/tools nearly nonexistent.

Reconfiguration uses:

- Field upgrades ⇒ product life extension, changing requirements.
- In system board-level testing and field diagnostics.
- Tolerance to faults.
- Risk-management in system development.
- Runtime reconfiguration ⇒ higher silicon efficiency.
  - Time-multiplexed pre-designed circuits take maximum use of resources.
  - Runtime specialized circuit generation.

FPGAs are **Re**configurable

Seemingly obvious point but …

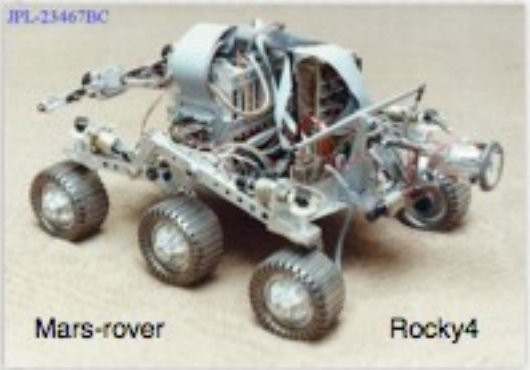
1. V … :ial
   r
2. C
   r

Rec

- Risk-management in system development.
- Runtime reconfiguration ⇒ higher silicon efficiency.
  - Time-multiplexed pre-designed circuits take maximum use of resources.
  - Runtime specialized circuit generation.

Multi-modal Computing Tasks

A premier application for reconfigurable devices is one with constrained size/ weight, need multiple functions at near ASIC performance.



- **Mini/Micro-UAVs**
  - One piece of silicon for all of sensor processing, navigation, communications, planning, logging, etc.
  - At different times different tasks take priority and consume higher percentage of resources.
- Other example: **hand-held** multi-function device with GPS, smart image capture/analysis, communications.

Multiple ASICs too expensive/big. Processor too slow.
Fine-grained reconfigurable devices has the flexibility to efficiently match task parallelism over a wide variety of tasks – deployed as needed and reconfigured as needed.

Sounds great, what's the catch?

- Lack of programming model with convenient and effective tools.
- Most successful computing applications using reconfigurable devices involve substantial "hand mapping". Essentially circuit design.
- Contributed to limited success at DARPA, demise of many startups, continues to be the challenge.
- C-to-gates startups might help: Synfora, Auto-ESL, ...

Fine-grained Reconfigurable Fabrics

- Some work in RC has evolved to course-grain (processor based) arrays:
  - Broadcom Calisto/silicon-spice/matrix. MIT/RAW, UCB/ MIT DSA architecture, Ambric.
- Many-core architecures now happening.
- Will homogeneous fine-grained (CLB based) arrays be more important in the future?
  1. Will a single array-of-processors type architecture be efficient for more than a particular class of apps (the general purpose parallel machine architecture problem).
     - Parallel machines one application might probably yield low computational density on other problems.

Fine-grained Reconfigurable Fabrics

2. Homogeneous fine-grained arrays are maximally flexible:
   a. Admit a wide variety of computational architectures models: arrays of processors, hybrid approaches, hard-wired dataflow, systolic processing, vector processing, etc.
   b. Admit a wide variety of parallelism modes: SIMD, MIMD, bit-level, etc. Resources can be deployed to lower-latency when required for tight feedback loops (not possible with may parallel architectures that optimize for throughput).
   c. Supports many compilation/resource management models: Statically compiled, dynamically mapped.

Safe bet as a standard device.

Fine-grained Reconfigurable Fabrics

- Xilinx and Altera doing a great job on current vector, but:
  - Tools are weak
    - No help on programming model issue.
    - No architecture synthesis, retiming.
    - No runtime support or operating system.
  - Reconfiguration is slow.
  - Power consumption much higher than it needs to be.
  - No defect tolerance

Rapid Runtime Reconfiguration

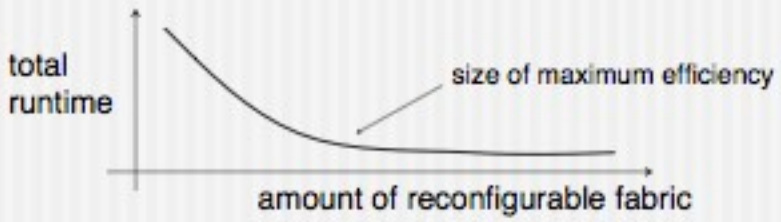
- Might permit even higher efficiency through hardware sharing (multiplexing) and on the fly circuit specialization.
  - Largely unexploited (unproven) to date.
  - A few research projects have explored this idea.
  - Need to be careful – multiplexing adds cost.
  - Remember the "Binding Time Principle"

    Earlier the "instruction" is bound, the less area & delay required for the implementation.

Rapid Runtime Reconfiguration

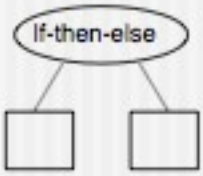
Why dynamic reconfiguration?

1. Time-multiplexing resources allows more efficient use of silicon (in ways ASICs typically do not):
   a. Low-duty cycle or "off critical path" computations time share fabric while critical path stays mapped in:

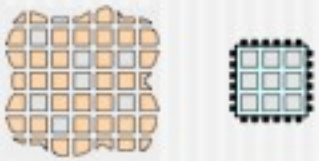


Rapid Runtime Reconfiguration

b. Course **data-dependent control** flow maps in only useful dataflow:
c. Allowable task foot-print may change as other tasks come and go or faults occur.



Fabric **virtualization** allows automatic migration up and down in device sizes and eases application development.



Rapid Runtime Reconfiguration

2. Runtime Circuit Specialization:
   - Example: fixed coefficient multipliers in adaptive filter changing value at low rate.
   - Aggressive constant propagation (based perhaps on runtime profiling), reduces circuit size and delay.
   - Could use "branch/value/range prediction" to map most common case and fault in exceptional cases.
   - Can be template based – "fill in the blanks", but better if we put PPR in runtime loop!
   - New work using array assisted place and route may make it possible.