# ESE532: System-on-a-Chip Architecture

Day 24: April 17, 2017
Defect Tolerance



## Today

- Reliability Challenges
- Defect Tolerance
  - Memories
  - Interconnect
  - FPGA
- FPGA Variation and Energy



## Message

- At small feature sizes, not viable to demand perfect fabrication of billions of transistors on a chip
- Modern ICs are like snowflakes
  - Everyone is different, changes over time
- Reconfiguration allows repair
  - Finer grain → higher defect rates
  - Tolerate variation → lower energy



## Intel Xeon Phi Offerings

| Part # | Cores |
|---|---|
| 7290F | 72 |
| 7250F | 68 |
| 7230F | 64 |

http://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html



## Intel Xeon Phi Offerings

| Part # | Cores |
|---|---|
| 7290F | 72 |
| 7250F | 68 |
| 7230F | 64 |

Is Intel producing 3 separate chips?

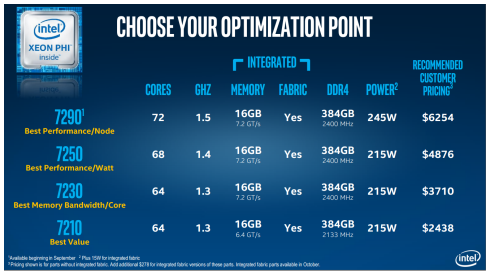


## Preclass 1 and Intel Xeon Phi Offerings

| Part # | Cores |
|---|---|
| 7290F | 72 |
| 7250F | 68 |
| 7230F | 64 |

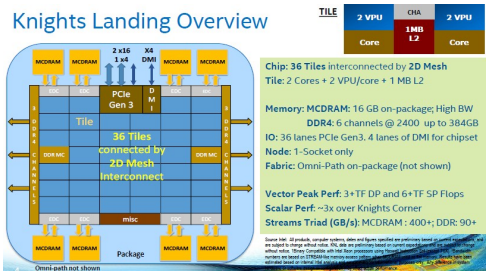
Cost ratio between 72 and 64 processor assuming fixed $mm^2$ per core?

# Intel Xeon Phi Pricing






# Intel Knights Landing

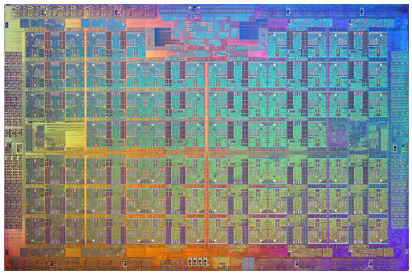



https://www.nextplatform.com/2016/06/20/intel-knights-landing-yields-big-bang-buck-jump/



[Intel, Micro 2016]

# Knights Landing Xeon Phi





[Intel, Micro 2016]

# What's happening?

- Fabricated chip has 76 cores
- Not expect all to work
- Selling based on functional cores
  - 72, 68, 64
- Charge premium for high core counts
  - Don't yield as often, people pay more

https://www.nextplatform.com/2016/08/22/intel-tweaking-xeon-phi-deep-learning/



[Intel, Micro 2016]

# Warmup Discussion

- Where else do we guard against defects today?
  - Where do we accept imperfection today?



# Motivation: Probabilities

- Given:
  - N objects
  - $P_g$ yield probability
- What's the probability for yield of composite system of N items? [Preclass 2]
  - Assume iid faults
  - P(N items good) = $(P_g)^N$

## Probabilities

- $P_{all\_good}(N) = (P_g)^N$
- P=0.999999

| N | $P_{all\_good}(N)$ |
|---|---|
| $10^4$ | |
| $10^5$ | |
| $10^6$ | |
| $10^7$ | |



## Probabilities

- $P_{all\_good}(N) = (P_g)^N$
- P=0.999999

| N | $P_{all\_good}(N)$ |
|---|---|
| $10^4$ | 0.99 |
| $10^5$ | 0.90 |
| $10^6$ | 0.37 |
| $10^7$ | 0.000045 |



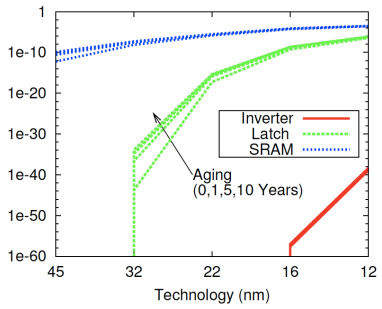
## Simple Implications

- As N gets large
  - must either increase reliability
  - …or start tolerating failures
- N
  - memory bits
  - disk sectors
  - wires
  - transmitted data bits
  - processors
  - transistors
  - molecules

As devices get smaller, failure rates increase
chemists think P=0.95 is good
As devices get faster, failure rate increases



## Failure Rate Increases




[Nassif / DATE 2010]



## Quality Required for Perfection?

- How high must $P_g$ be to achieve 90% yield on a collection of $10^{10}$ devices?

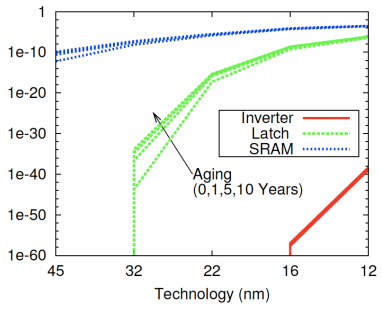
[preclass 4]

$$(P_g)^{10^{10}} > 0.9$$

$$P_g > 1 - 10^{-11}$$



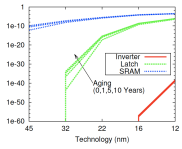
## Failure Rate Increases




[Nassif / DATE 2010]

## Challenge

- Feature size scales down (S)
- Capacity (area) → increases ($1/S^2$)
  - N increase
- Reliability per device goes down
  - $P_g$ decrease
- P(N items good) = $(P_g)^N$





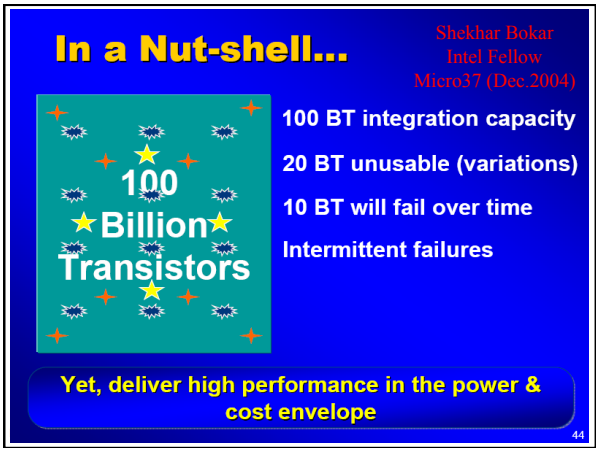

## Defining Problems



## Three Problems

1. **Defects:** Manufacturing imperfection
   - Occur before operation; persistent
     - Shorts, breaks, bad contact
2. **Transient Faults:**
   - Occur during operation; transient
     - node X value flips: crosstalk, ionizing particles, bad timing, tunneling, thermal noise
3. **Lifetime "wear" defects**
   - Parts become bad during operational lifetime
     - Fatigue, electromigration, burnout….
   - …slower
     - NBTI, Hot Carrier Injection






## Defect Rate

- Device with $10^{11}$ elements (100BT)
- 3 year lifetime = $10^8$ seconds
- Accumulating up to 10% defects
- $10^{10}$ defects in $10^8$ seconds
  - →1 new defect every 10ms
- At 10GHz operation:
  - One new defect every $10^8$ cycles
  - $P_{newdefect}=10^{-19}$



## First Step to Recover

Admit you have a problem
(observe that there is a failure)

## Detection

- How do we determine if something wrong?
  - Some things easy
    - ….won't start
  - Others tricky
    - …one **and** gate computes False & True→True
- Observability
  - can see effect of problem
  - some way of telling if defect/fault present



## Detection

- Coding
  - space of legal values << space of all values
  - should only see legal
  - *e.g.* parity, ECC (Error Correcting Codes)
- Explicit test (defects, recurring faults)
  - ATPG = Automatic Test Pattern Generation
  - Signature/BIST=Built-In Self-Test
  - POST = Power On Self-Test
- Direct/special access
  - test ports, scan paths



## Coping with defects/faults?

- **Key idea: redundancy**
- Detection:
  - Use redundancy to detect error
- Mitigating: use redundant hardware
  - Use spare elements in place of faulty elements (defects)
  - Compute multiple times so can discard faulty result (faults)



## Defect Tolerance



## Three Problems

1. **Defects:** Manufacturing imperfection
   - Occur before operation; persistent
     - Shorts, breaks, bad contact
2. **Transient Faults:**
   - Occur during operation; transient
     - node X value flips: crosstalk, ionizing particles, bad timing, tunneling, thermal noise
3. **Lifetime "wear" defects**
   - Parts become bad during operational lifetime
     - Fatigue, electromigration, burnout….
   - …slower
     - NBTI, Hot Carrier Injection



## Two Models

- Disk Drives (defect map)
- Memory Chips (perfect chip)

## Disk Drives

- Expose defects to software
  - software model expects defects
    - Create table of good (bad) sectors
  - manages by masking out in software
    - (at the OS level)
    - Never allocate a bad sector to a task or file
  - yielded capacity varies



## Memory Chips

- Provide model in **hardware** of perfect chip
- Model of perfect memory at capacity X
- Use redundancy in hardware to provide perfect model
- Yielded capacity fixed
  - discard part if not achieve

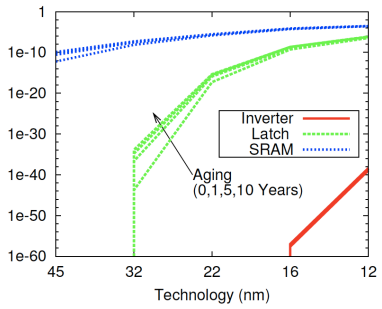


## Example: Memory

- Correct memory:
  - N slots
  - each slot reliably stores last value written
- Millions, billions, etc. of bits…
  - have to get them all right?



## Failure Rate Increases




[Nassif / DATE 2010]



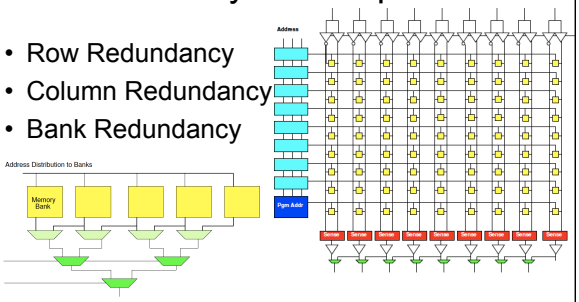
## Memory Defect Tolerance

- Idea:
  - few bits may fail
  - provide more raw bits
  - configure so yield what looks like a perfect memory of specified size



## Memory Techniques

- Row Redundancy
- Column Redundancy
- Bank Redundancy



# Yield M of N

- Preclass 5: Probability of yielding 3 of 5 things?
  - Symbolic?
  - Numerical for $P_g$=0.9?



# Yield M of N

- P(M of N) = P(yield N)
  + (N choose N-1) P(exactly N-1)
  + (N choose N-2) P(exactly N-2)…
  + (N choose N-M) P(exactly N-M)…
  [think binomial coefficients]



# M of 5 example

- $1*P^5 + 5*P^4(1-P)^1+10P^3(1-P)^2+10P^2(1-P)^3+5P^1(1-P)^4 + 1*(1-P)^5$
- Consider P=0.9
  - $1*P^5$ 0.59 M=5 P(sys)=0.59
  - $5*P^4(1-P)^1$ 0.33 M=4 P(sys)=0.92
  - $10P^3(1-P)^2$ 0.07 M=3 P(sys)=0.99
  - $10P^2(1-P)^3$ 0.008
  - $5P^1(1-P)^4$ 0.00045
  - $1*(1-P)^5$ 0.00001

Can achieve higher system yield than individual components!



# Possible Yield of 76 cores@ P=0.9

| Processors Yield | Prob Exact | Prob at least |
|---|---|---|
| 76 | 0.001 | 0.001 |
| 75 | 0.004 | 0.005 |
| 74 | 0.016 | 0.020 |
| 73 | 0.041 | 0.061 |
| 72 | 0.079 | 0.140 |
| 71 | 0.119 | 0.259 |
| 70 | 0.148 | 0.407 |
| 69 | 0.156 | 0.562 |
| 68 | 0.141 | 0.704 |
| 67 | 0.112 | 0.816 |
| 66 | 0.079 | 0.895 |
| 65 | 0.050 | 0.945 |
| 64 | 0.028 | 0.973 |



# Possible Yield of 76 cores@ P=0.9

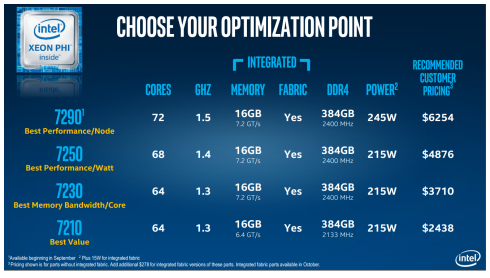
| Processors Yield | Prob at least |
|---|---|
| 76 | 0.001 |
| 75 | 0.005 |
| 74 | 0.020 |
| 73 | 0.061 |
| 72 | 0.140 |
| 71 | 0.259 |
| 70 | 0.407 |
| 69 | 0.562 |
| 68 | 0.704 |
| 67 | 0.816 |
| 66 | 0.895 |
| 65 | 0.945 |
| 64 | 0.973 |

Out of 100 chips, how many?

Sell with 72:
Sell with 68:
Sell with 64:
Discard:



# Intel Xeon Phi Pricing



## Repairable Area

- Not all area in a RAM is repairable
  - memory bits spare-able
  - io, power, ground, control not redundant



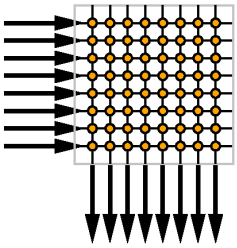
## Repairable Area

- P(yield) = P(non-repair) * P(repair)
- P(non-repair) = $P^{Nnr}$
  - $N_{nr} << N_{total}$
  - $P > P_{repair}$
    - e.g. use coarser feature size
    - Differential reliability
- P(repair) ~ P(yield M of N)



## Consider a Crossbar

- Allows us to connect any of N things to each other
  - E.g.
    - N processors
    - N memories
    - N/2 processors
    - + N/2 memories

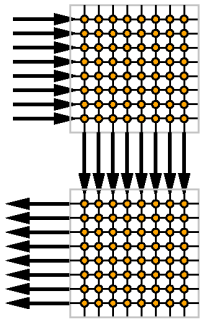




## Crossbar Buses and Defects

- Two crossbar multibus
- Wires may fail
- Switches may fail

- How tolerate
  - Wire failures between crossbars?
  - Switch failures?

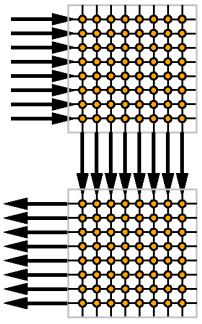




## Crossbar Buses and Defects

- Two crossbars
- Wires may fail
- Switches may fail

- Provide more wires
  - Any wire fault avoidable
    - M choose N

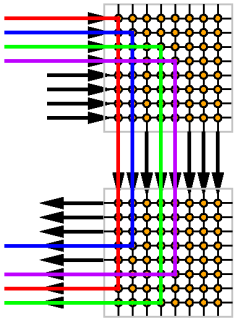




## Crossbar Buses and Defects

- Two crossbars
- Wires may fail
- Switches may fail

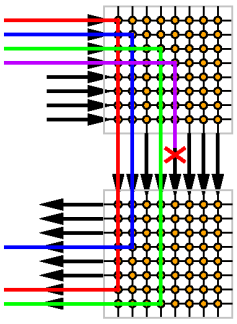
- Provide more wires
  - Any wire fault avoidable
    - M choose N



## Crossbar Buses and Faults

- Two crossbars
- Wires may fail
- Switches may fail

- Provide more wires
  - Any wire fault avoidable
    - M choose N

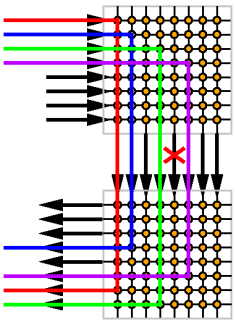




## Crossbar Buses and Faults

- Two crossbars
- Wires may fail
- Switches may fail

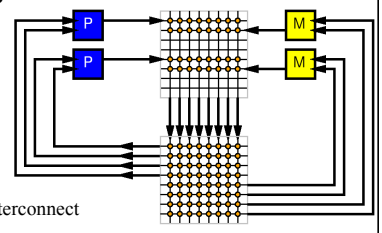
- Provide more wires
  - Any wire fault avoidable
    - M choose N
  - Same idea





## Simple System

- P Processors
- M Memories
- Wires

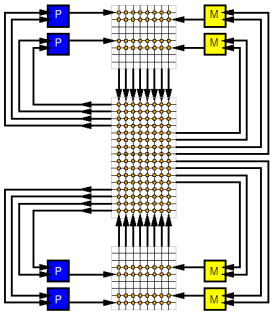



Memory, Compute, Interconnect



## Simple System w/ Spares

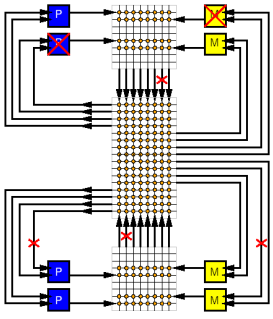
- P Processors
- M Memories
- Wires
- Provide spare
  - Processors
  - Memories
  - Wires





## Simple System w/ Defects

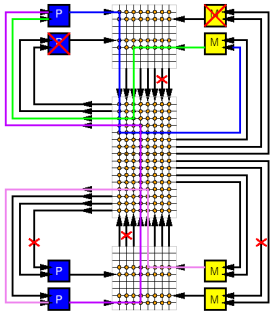
- P Processors
- M Memories
- Wires
- Provide spare
  - Processors
  - Memories
  - Wires
- ...and defects





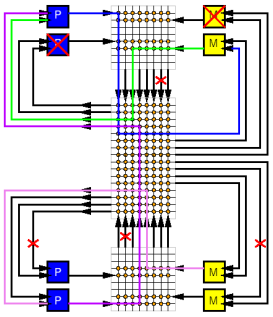
## Simple System Repaired

- What are the costs?
  - Area
  - Energy
  - Delay



## Simple System Repaired

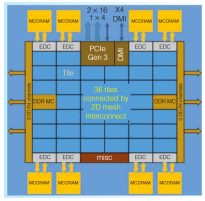
- P Processors
- M Memories
- Wires
- Provide spare
  - Processors
  - Memories
  - Wires
- Use crossbar to switch together good processor and memories

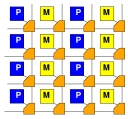





## In Practice

- Crossbars are inefficient
- Use switching networks with
  - Locality
  - Segmentation
- …but basic idea for sparing is the same








## FPGAs



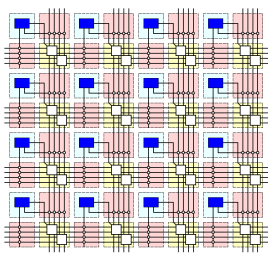
## Modern FPGA

- Has 10,000 to millions of LUTs
- Hundreds to thousands of
  - Memory banks
  - Multipliers
- Reconfigurable interconnect



## XC7Z020

- 6-LUTs: 53,200
- DSP Blocks: 220
  - 18x25 multiply, 48b accumulate
- Block RAMs: 140
  - 36Kb
  - Dual port
  - Up to 72b wide





## Modern FPGA

- Has 10,000 to millions of gates
- Hundreds to thousands of
  - Memory banks
  - Multipliers
- Reconfigurable interconnect

- If a few resources don't work
  - avoid them

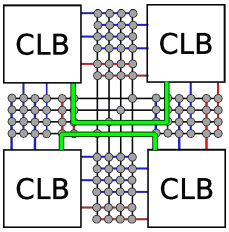
## Granularity

- How do transistors compare between 6-LUT and 64b processor core?
  - [qualitative or ballpark]
- Yield of 6-LUTs vs. processor cores?
- Resources lots per transistor defect?



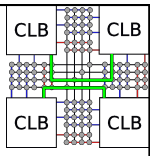
## Interconnect Defects

- Route around interconnect defects

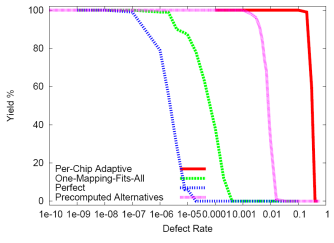





## Defect-Level Viable with FPGAs




- Fine-grained repair
- Avoiding routing defects
  - Tolerates >20% switch defects

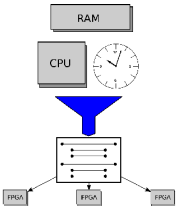


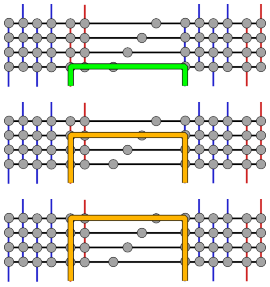

[Rubin/unpublished]



## Choose Your own Adventure

- **Idea:** Precompute Alternate mappings
  - Still just one bitstream



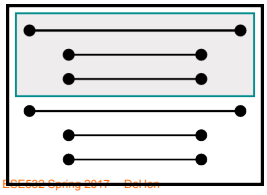
[Rubin, FPGA 2008]

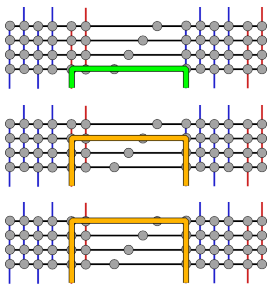


## Select Alternatives at Load Time

Examine path

- Configure
- Test
  - Good: skip to next net
  - Bad: grab next path



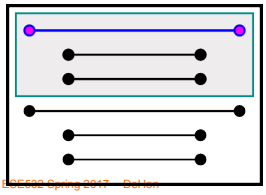


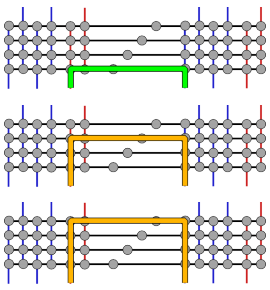


## Loading

Examine path

- **Configure**
- Test
  - Good: skip to next net
  - Bad: grab next path

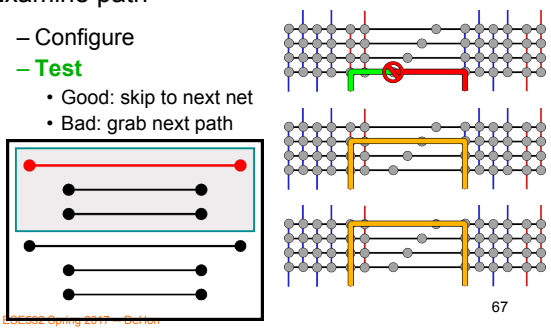




## Loading

Examine path

- Configure
- **Test**
  - Good: skip to next net
  - Bad: grab next path

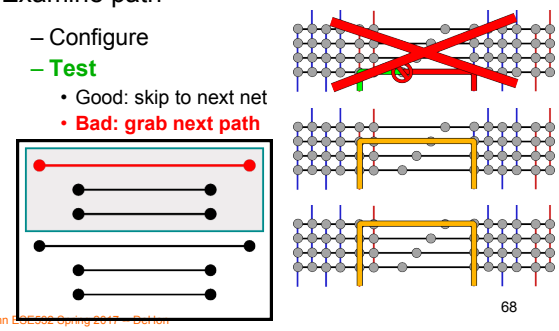




## Loading

Examine path

- Configure
- **Test**
  - Good: skip to next net
  - **Bad: grab next path**

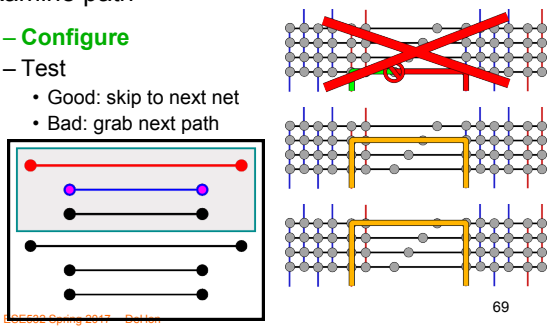




## Loading

Examine path

- **Configure**
- Test
  - Good: skip to next net
  - Bad: grab next path

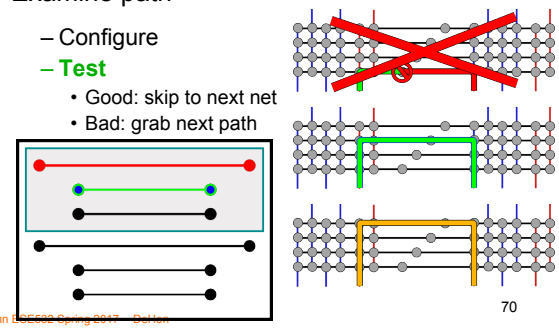




## Loading

Examine path

- Configure
- **Test**
  - Good: skip to next net
  - Bad: grab next path

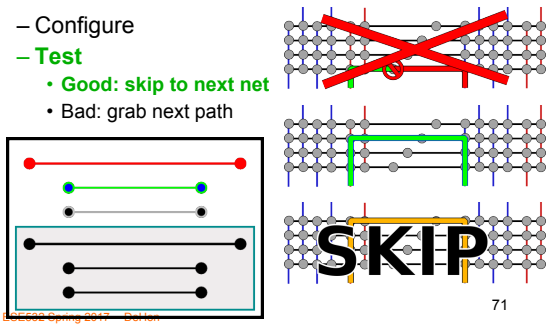




## Loading

Examine path

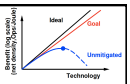
- Configure
- **Test**
  - **Good: skip to next net**
  - Bad: grab next path

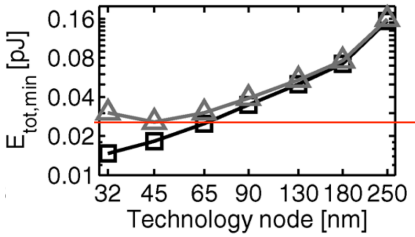





## FPGAs
## Variation and Energy

## Variation threatens E/Op reduction



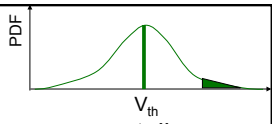

Black nominal
Grey with variation

Min-Energy for multiplication (typically subthreshold)

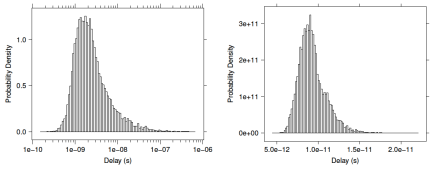
[Bol *et al.*, IEEE TR VLSI Sys 17(10):1508—1519]



## Driven by Tails




- High margins driven by uncommon tails
  - Most devices much better
- Large device count → sample further into tails

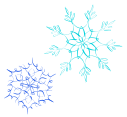


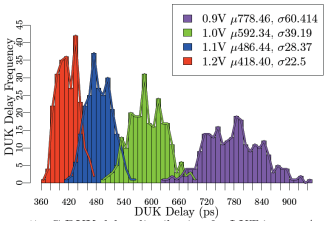

Mehta 2012: 22nm PTM LP – 10,000 samples



## Variation

- Modern ICs are like Snowflakes
  - Each one is different

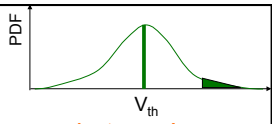




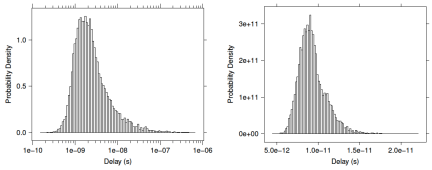

[Gojman, FPGA2013]



## Driven by Tails




- Given high defect tolerance, what can I do with the devices in the tails?




Mehta 2012: 22nm PTM LP – 10,000 samples

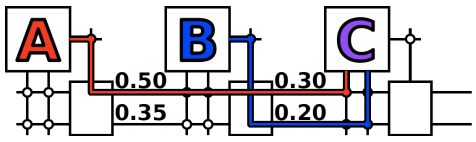


## Variation Tolerance

- **Idea:** assign resources, post fabrication to compensate for variations
- **Opportunity:**
  - Balance fast paths and slow paths
  - Assign slow resources to non-critical paths
  - Avoid devices in uncommon tails
  - Scale voltage down more aggressively
- Fixed design limited to worst-case path
  - Must scale voltage up so path meets timing
- **Paradigm shift:** Component-specific mapping

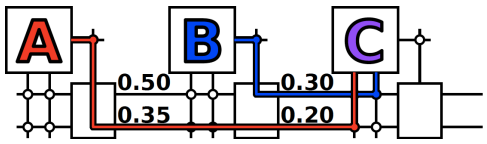


## Variation Challenge




- Use of high $V_{th}$ resource forces high supply voltage ($V_{dd}$) to meet timing requirement
- Delay: CV/I and I goes as $(V_{dd}-V_{th})^2$

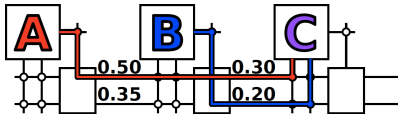
## Component-Specific



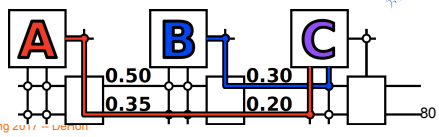

- Avoid high $V_{th}$ resource
- Allow lower supply voltage ($V_{dd}$) to meet timing requirement
- Delay: CV/I and I goes as $(V_{dd}-V_{th})^2$



## Component-Specific Assignment

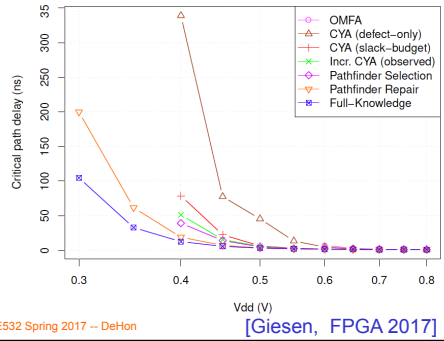



- Could come out other way
  - Best mapping unique to component



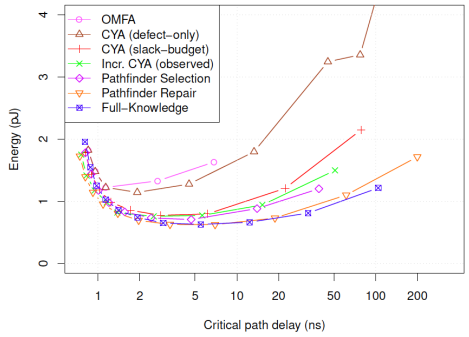



## Knowledge Mapping and Voltage-Delay




 [Giesen, FPGA 2017] 81

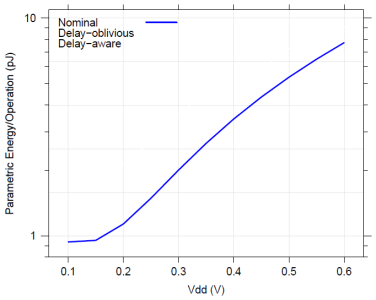
## Energy vs. Delay




 [Giesen, FPGA 2017] 82

## Energy vs $V_{dd}$ (des)

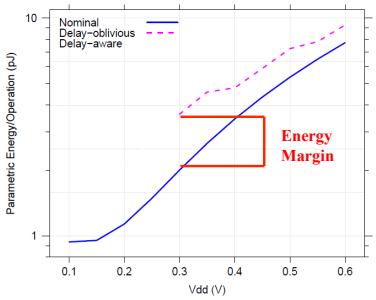
- Nominal uses minimum size




 [Mehta, FPGA 2012] 83

## Energy vs $V_{dd}$ (des)

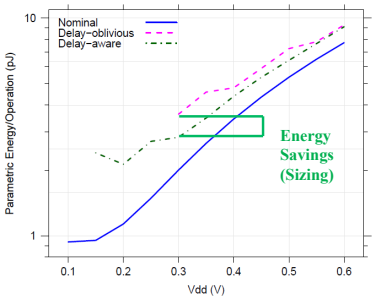
- Nominal uses minimum size




 [Mehta, FPGA 2012] 84

## Energy vs $V_{dd}$ (des)

- Nominal uses minimum size
- Delay-aware routing reduces energy margins
  1. Smaller sizes

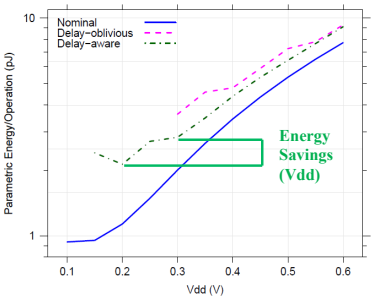



 [Mehta, FPGA 2012]



## Energy vs $V_{dd}$ (des)

- Nominal uses minimum size
- Delay-aware routing reduces energy margins
  1. Smaller sizes
  2. Lower voltages

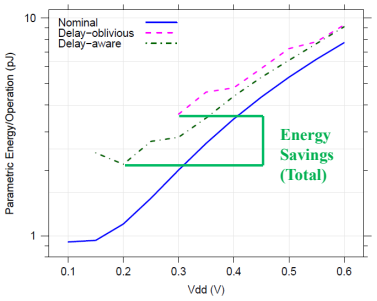



 [Mehta, FPGA 2012]



## Energy vs $V_{dd}$ (des)

- Nominal uses minimum size
- Delay-aware routing reduces energy margins
  1. Smaller sizes
  2. Lower voltages
  3. Less Leakage

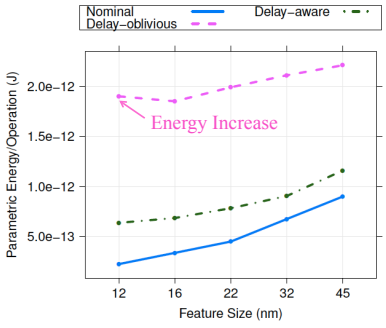



 [Mehta, FPGA 2012]



## Minimum Energy vs Technology

- Delay-oblivious scales to 16nm
- Delay-aware scales to 12nm at least….
- Extend useful life of Silicon technology generation




 [Mehta PhD thesis 2012 (refined from FPGA 2012)]

## Big Ideas

- At small feature sizes, not viable to demand perfect fabrication of billions of transistors on a chip
- Modern ICs are like snowflakes
  - Everyone is different, changes over time
- Reconfiguration allows repair
  - Finer grain → higher defect rates
  - Tolerate variation → lower energy





## Admin

- Final Project Report
  - Due Friday