CS184a:
Computer Architecture (Structure and Organization)

Day 21: March 2, 2005

Time Multiplexing

Previously

- Saw how to pipeline **architectures**
  - specifically interconnect
  - talked about general case
- Including how to map to them
- Saw how to **reuse** resources at maximum rate to do the *same* thing

Today

- Multicontext
  - Review why
  - Cost
  - Packing into contexts
  - Retiming requirements
- [stuff saw in overview week 2-3, now can dig deeper into details]

How often is **reuse** of the *same* operation applicable?

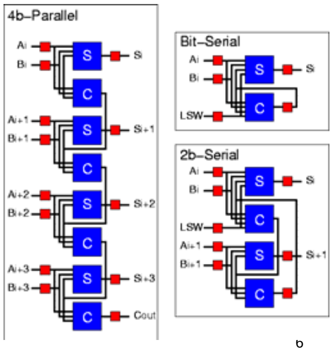
- In what cases can we exploit high-frequency, heavily pipelined operation?
- …and when can we not?

How often is **reuse** of the *same* operation applicable?

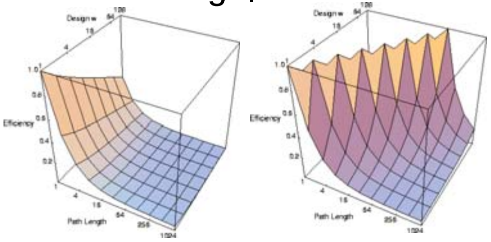
- Can we exploit higher frequency offered?
  - High throughput, feed-forward (acyclic)
  - Cycles in flowgraph
    - abundant data level parallelism [C-slow]
    - no data level parallelism
  - Low throughput tasks
    - structured (*e.g.* datapaths) [serialize datapath]
    - unstructured
  - Data dependent operations
    - similar ops [local control -- next time]
    - dis-similar ops

Structured Datapaths

- Datapaths: same *pinst* for all bits
- Can serialize and reuse the same data elements in succeeding cycles
- example: adder

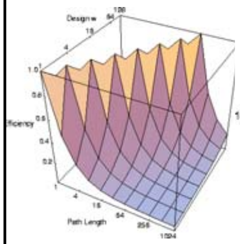


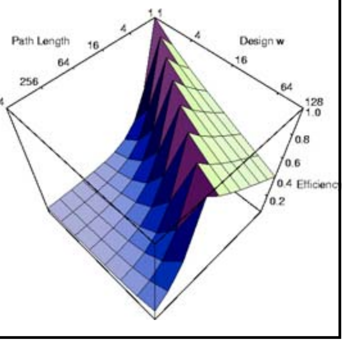
Throughput Yield



FPGA Model -- if throughput requirement is reduced for wide word operations, serialization allows us to reuse active area for same computation

Throughput Yield





Same graph, rotated to show backside.

Remaining Cases

- Benefit from multicontext as well as high clock rate
- *i.e.*
  - cycles, no parallelism
  - data dependent, dissimilar operations
  - low throughput, irregular (can't afford swap?)

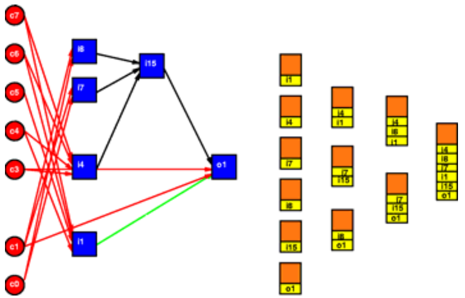
Single Context

- When have:
  - cycles and no data parallelism
  - low throughput, unstructured tasks
  - dis-similar data dependent tasks
- Active resources sit idle most of the time
  - Waste of resources
- Cannot reuse resources to perform **different** function, only **same**

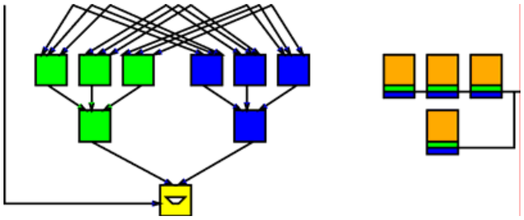
Resource Reuse

- To use resources in these cases
  - must direct to do different things.
- Must be able tell resources how to behave
- => separate instructions (*pinsts*) for each behavior

Example: Serial Evaluation

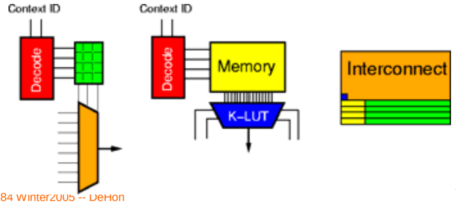


Example: Dis-similar Operations

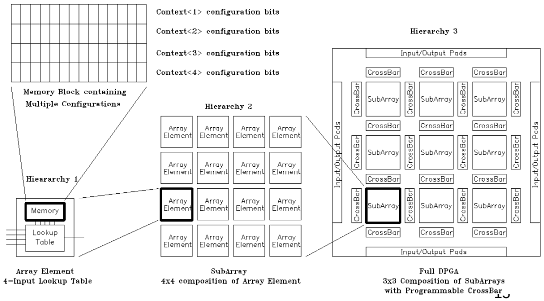


Multicontext Organization/Area

- $A_{ctxt} \approx 80K\lambda^2$
  - dense encoding
- $A_{base} \approx 800K\lambda^2$
- $A_{ctxt} : A_{base} = 1:10$

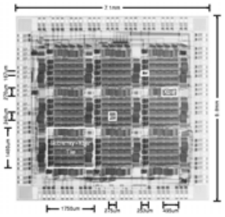


Example: DPGA Prototype


Context<1> configuration bits
Context<2> configuration bits
Context<3> configuration bits
Context<4> configuration bits
Memory Block containing
Multiple Configurations
Hierarchy 1
Memory
Lookup Table
Hierarchy 2
Array Element
Hierarchy 3
Input/Output Pads
CrossBar
SubArray
Array Element
4–Input Lookup Table
SubArray
4x4 composition of Array Element
Full DPGA
3x3 Composition of SubArrays
with Programmable CrossBar

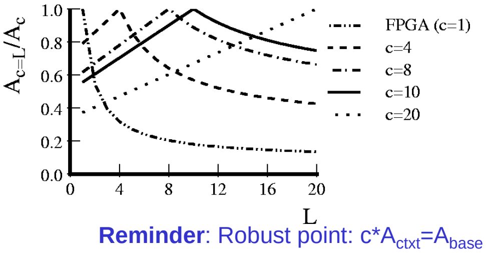
Example: DPGA Area

Process	1.0μ CMOS
Chip	7.1mm×6.8mm
AEs	144
Contexts	4
AE Area	$640K\lambda^2$
A_{base}	$544K\lambda^2$
A_{ctx}	$24K\lambda^2$
$A_{base} : A_{ctx}$	20+:1
(nominal delay)	9ns



Multicontext Tradeoff Curves

- Assume Ideal packing: $N_{active} = N_{total}/L$



Reminder: Robust point: $c{*}A_{ctxt} = A_{base}$

In Practice

Limitations from:

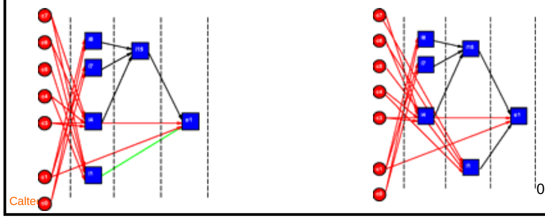
- Scheduling
- Retiming

Scheduling Limitations

- N_A (**active**)
  - – size of largest stage
- **Precedence**:
  - can evaluate a LUT only after predecessors have been evaluated
  - →cannot always, completely equalize stage requirements

Scheduling

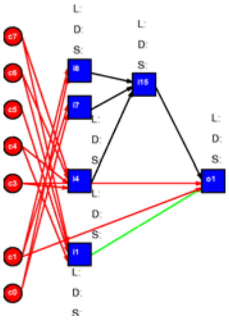
- Precedence limits packing freedom
- Freedom do have
  - – shows up as slack in network



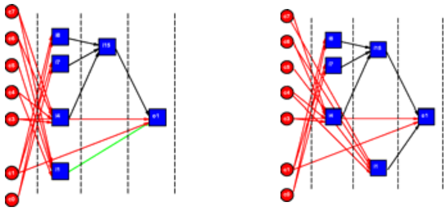
Scheduling

- Computing Slack:
  - – ASAP (As Soon As Possible) Schedule
    - propagate depth forward from primary inputs
      - – depth = 1 + max input depth
  - – ALAP (As Late As Possible) Schedule
    - propagate distance from outputs back from outputs
      - – level = 1 + max output consumption level
  - – Slack
    - slack = L+1-(depth+level) [PI depth=0, PO level=0]

Slack Example



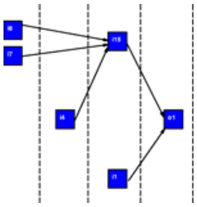
Allowable Schedules



Active LUTs (N_A) = 3

Sequentialization

- Adding time slots
  - – more sequential (more latency)
  - – add slack
    - allows better balance



L=4 →N_A=2 (4 or 3 contexts)

Multicontext Scheduling

- "Retiming" for multicontext
  - **goal**: minimize peak resource requirements
    - resources: logic blocks, retiming inputs, interconnect
- NP-complete
- list schedule, anneal

Multicontext Data Retiming

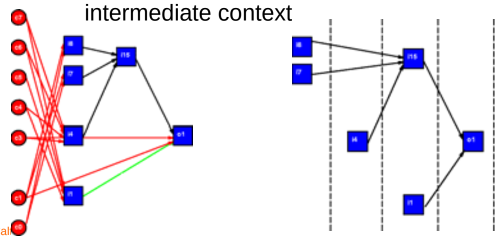
- How do we accommodate intermediate data?
- Effects?

Signal Retiming

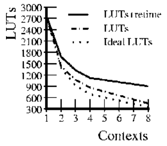
- Non-pipelined
  - hold value on LUT Output (wire)
    - from production through consumption
  - Wastes wire and switches by occupying
    - for entire critical path delay L
    - not just for 1/L'th of cycle takes to cross wire segment
  - How show up in multicontext?

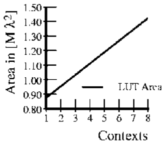
Signal Retiming

- Multicontext equivalent
  - need LUT to hold value for each intermediate context

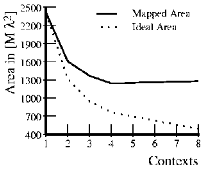


DES Latency Example

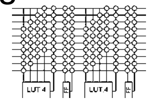




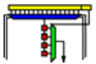
Single Output case

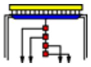


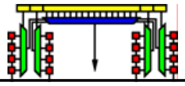
Alternate Retiming

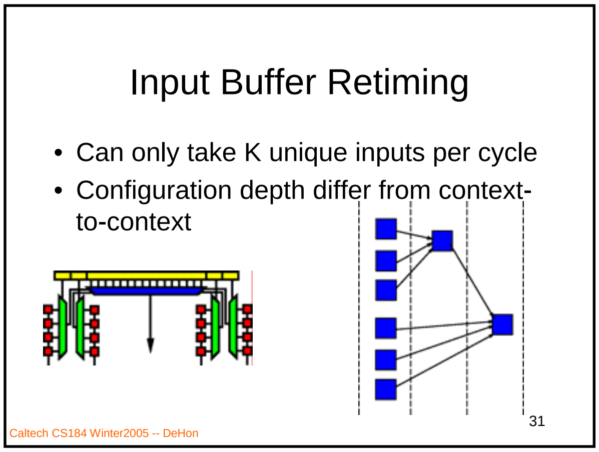


- Recall from last time (Day 20)
  - Net buffer
    - smaller than LUT
  - Output retiming
    - may have to route multiple times
  - Input buffer chain
    - only need LUT every depth cycles



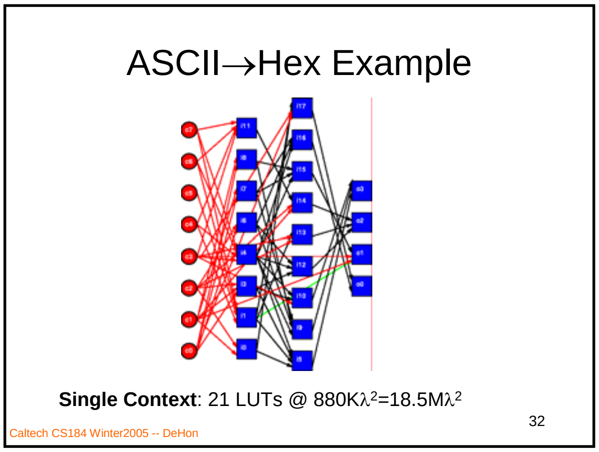






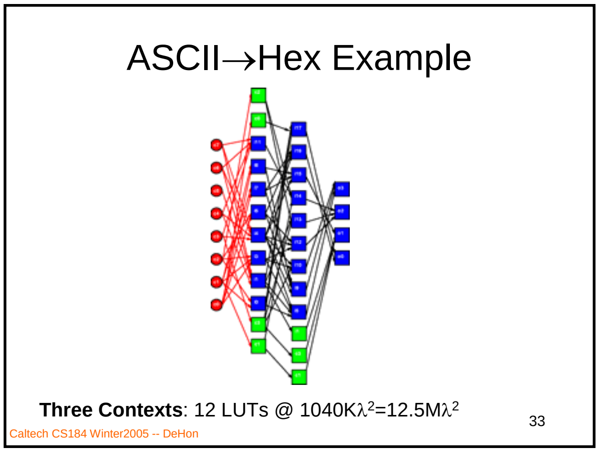
Input Buffer Retiming

- Can only take K unique inputs per cycle
- Configuration depth differ from context-to-context



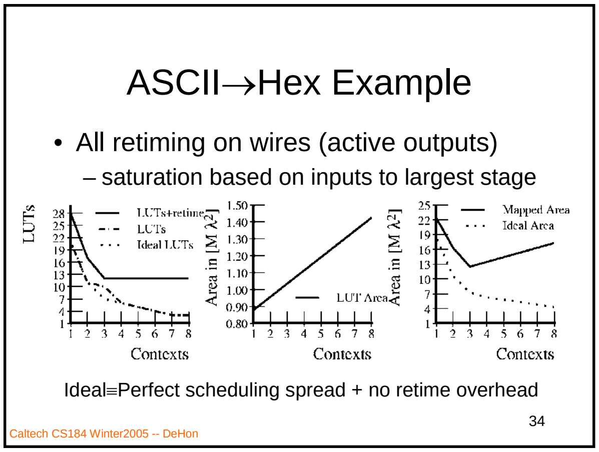
ASCII→Hex Example

Single Context: 21 LUTs @ 880Kλ^2=18.5Mλ^2



ASCII→Hex Example

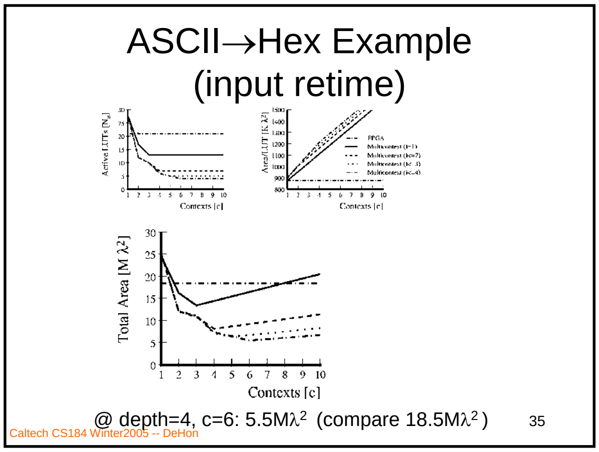
Three Contexts: 12 LUTs @ 1040Kλ^2=12.5Mλ^2



ASCII→Hex Example

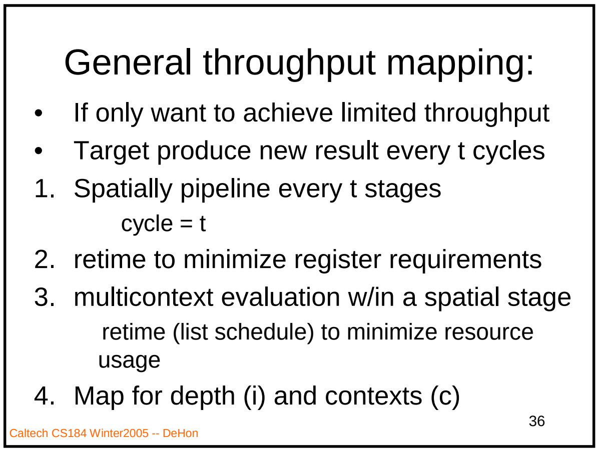
- All retiming on wires (active outputs)
  - – saturation based on inputs to largest stage

Ideal≡Perfect scheduling spread + no retime overhead



ASCII→Hex Example (input retime)

@ depth=4, c=6: 5.5Mλ^2 (compare 18.5Mλ^2)



General throughput mapping:

- If only want to achieve limited throughput
- Target produce new result every t cycles

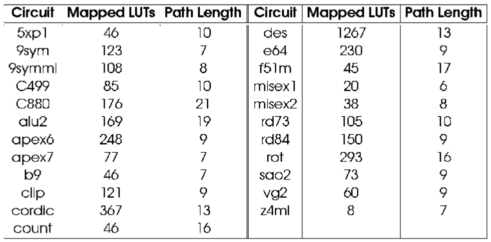
1. Spatially pipeline every t stages

   cycle = t
2. retime to minimize register requirements
3. multicontext evaluation w/in a spatial stage

   retime (list schedule) to minimize resource usage
4. Map for depth (i) and contexts (c)

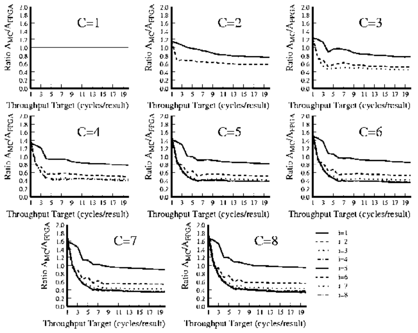
Benchmark Set

- 23 MCNC circuits
  - area mapped with SIS and Chortle

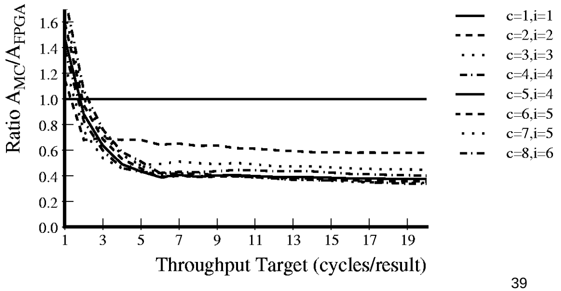


Circuit	Mapped LUTs	Path Length	Circuit	Mapped LUTs	Path Length
5xp1	46	10	des	1267	13
9sym	123	7	e64	230	9
9symml	108	8	f51m	45	17
C499	85	10	misex1	20	6
C880	176	21	misex2	38	8
alu2	169	19	rd73	105	10
apex6	248	9	rd84	150	9
apex7	77	7	rot	293	16
b9	46	7	sao2	73	9
clip	121	9	vg2	60	9
cordic	367	13	z4ml	8	7
count	46	16			

Multicontext vs. Throughput

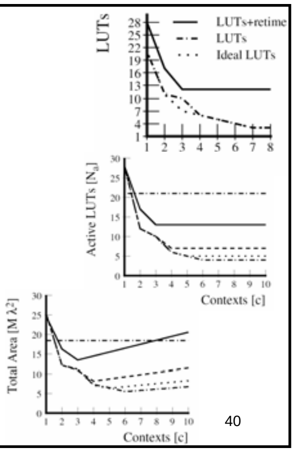


Multicontext vs. Throughput



General Theme

- Ideal Benefit
  - *e.g.* Active=N/C
- Precedence Constraints
- Resource Limits
  - Sometimes bottleneck
- Net Benefit
- Resource Balance

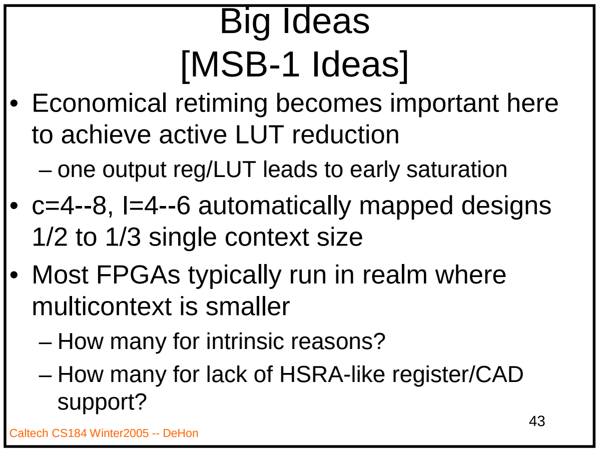


Admin

- Altera recruiting on campus Friday, March 4th

Big Ideas [MSB Ideas]

- Several cases cannot profitably reuse same logic at device cycle rate
  - cycles, no data parallelism
  - low throughput, unstructured
  - dis-similar data dependent computations
- These cases benefit from more than one instructions/operations per active element
- A_{ctxt}<< A_{active} makes interesting
  - save area by sharing active among instructions



Big Ideas [MSB-1 Ideas]

- Economical retiming becomes important here to achieve active LUT reduction
  - one output reg/LUT leads to early saturation
- c=4--8, I=4--6 automatically mapped designs 1/2 to 1/3 single context size
- Most FPGAs typically run in realm where multicontext is smaller
  - How many for intrinsic reasons?
  - How many for lack of HSRA-like register/CAD support?