# CMPE110 Lecture 23

# Custom Hardware Designs 2

Heiner Litz

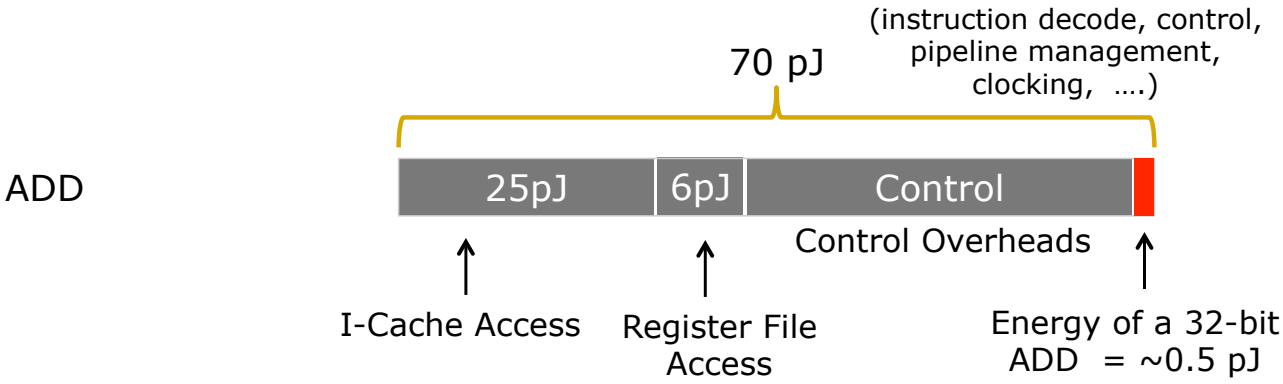
https://canvas.ucsc.edu/courses/12652

# Announcements


- Evaluations
- Next week
  - Monday: Pat will give exam review session 1
  - Wednesday: Heiner will give Zoom lecture on Multicore/GPU
  - Friday: Heiner will give exam review session 2
- Exam Review Session
  - Students to submit concrete questions to cover
  - E.g.: How do you compute the cache tag bits from a memory address?

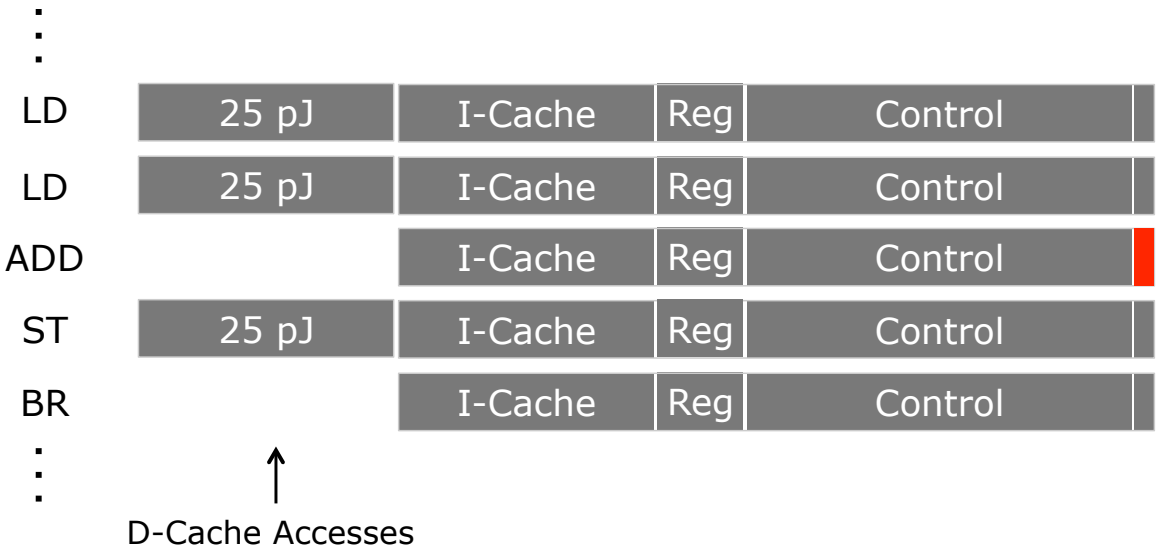
# Review

# Anatomy of a RISC Instruction



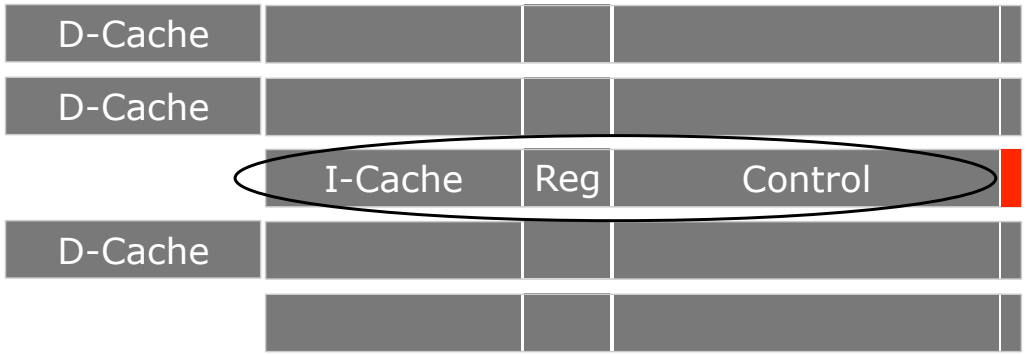

* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

# How Do We Reduce This Waste?



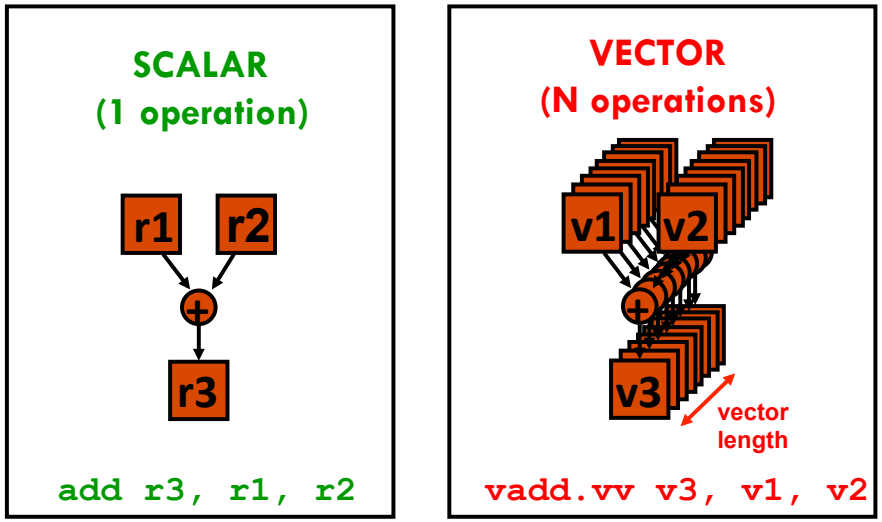

* Assuming a typical 32-bit embedded RISC in 45nm @ 0.9V technology

# Reducing This Waste



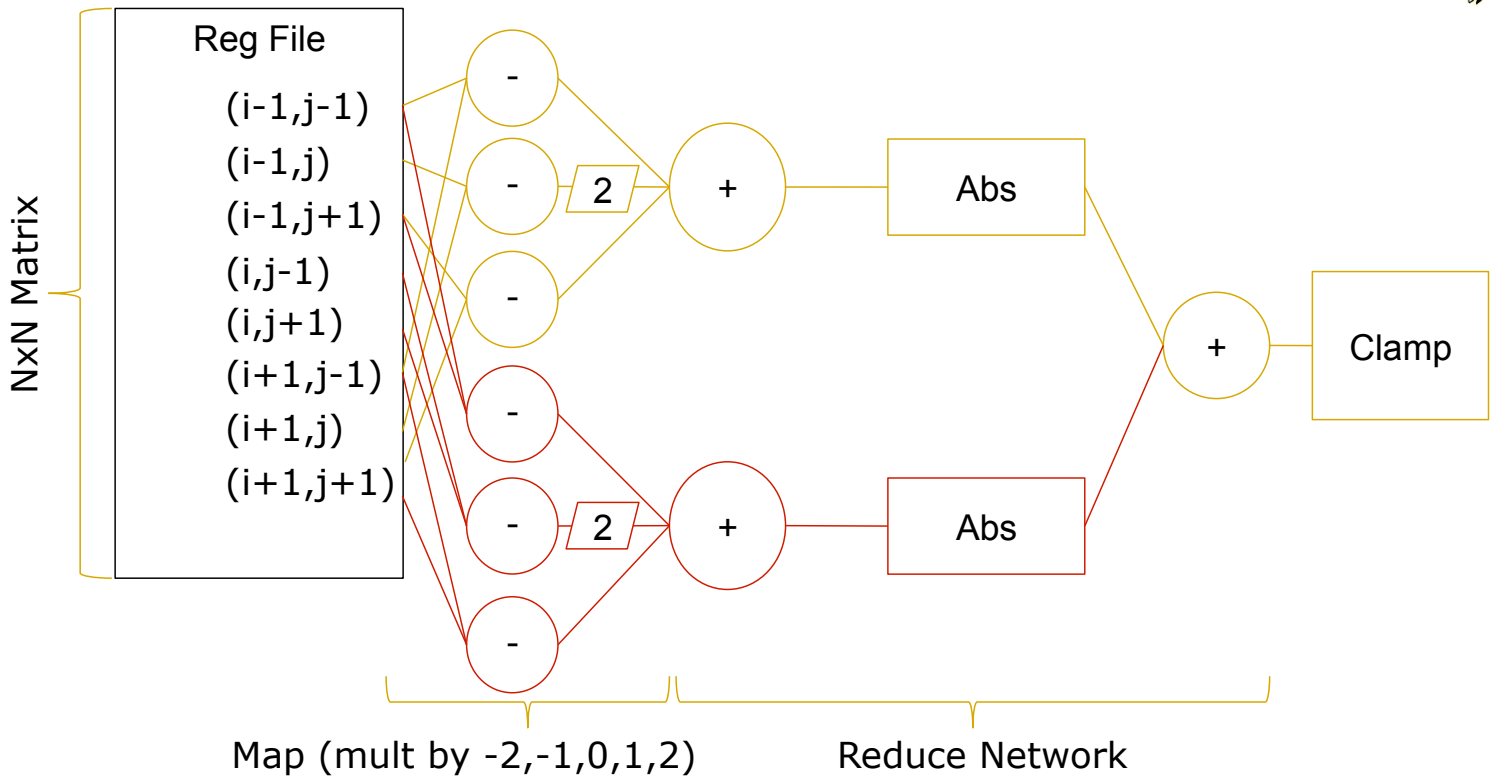

**1. Perform a large number of operations per instruction**

# Data-level Parallelism (Vectors)




- Scalar instructions operate on single numbers (scalars)
- Vector instructions operate on vectors of numbers
  - Linear sequences of numbers
  - Stored in a vector register file

# Domain Specific HW



# Memory Concerns: Bandwidth Matching


- Can only operate as fast as we can feed it
  - Memory too slow: Accelerator must stall
  - Accelerator too slow: Add more accelerators (or make yours bigger!)
- Match performance with available memory bandwidth

# Memory Overheads

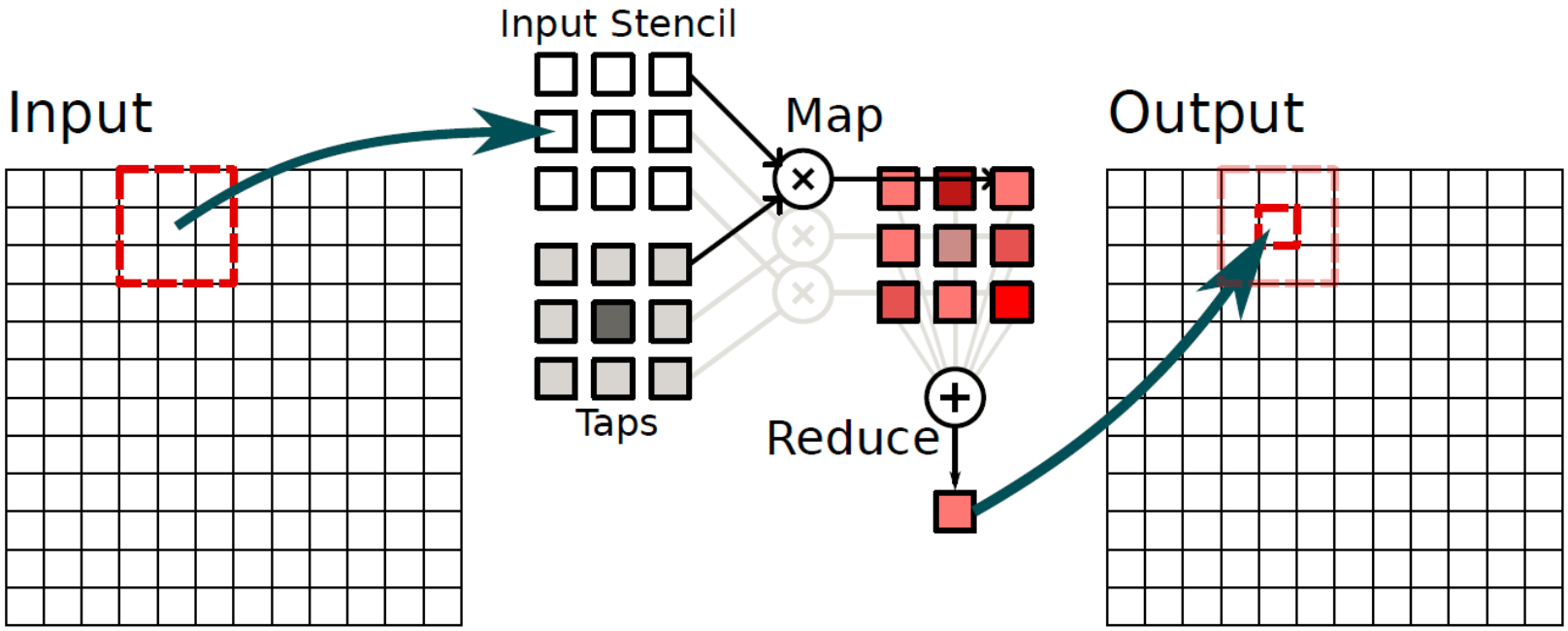

| Operation | Energy | Scale |
|---|---|---|
| 8-bit add | 0.03 pJ | 1 |
| 32-bit add | 0.10 pJ | 3 |
| 32-bit FP mult | 4.00 pJ | 133 |
| **RISC instruction** | **70.00 pJ** | **2,300** |
| 8KB cache access | 10.00pJ | 300 |
| **1MB cache access** | **100.00 pJ** | **3,000** |
| **DRAM access** | **2,000.00 pJ** | **60,000** |

# Minimizing Memory Access


- Getting data to and from accelerators (and cores) is expensive
  - Especially if that data is in DRAM
- To minimize energy
  - Minimize read/writes by buffering operands and intermediates locally
  - Maximize data reuse (where do we find reuse in Sobel?)

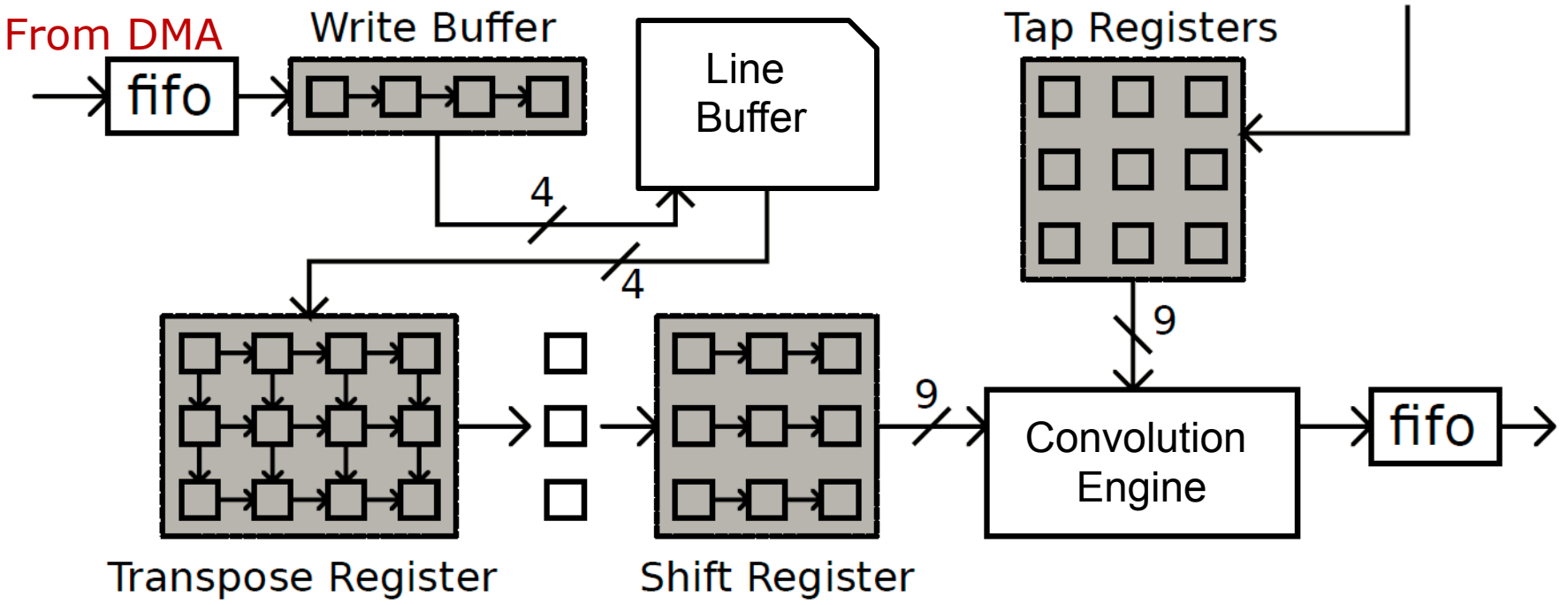
*Read and write once, reuse often!*

# Convolution Engine




- How many rows of data do we need to buffer?
- How much new data do we need per pixel?

# Convolution Datapath




- How many pixels per cycle?

# Managing the Accelerator


- Think of the accelerator as an I/O device
- Software manages the accelerator
  - Initiates Sobel computation
  - Provides input, receives output
  - Deals with errors

# Moving Data Between CPU & Accelerator


- Direct Memory Access (DMA)
  - Streaming memory transfer mechanism
  - Takes in a set of source memory addresses
  - Takes in a MMIO target address
  - Streams source memory words to target

# Writing to and from Custom HW


- Processor gathers image addresses
  - Ideally in a contiguous physical chunk
- Transmits addresses to DMA
- DMA writes data to accelerator
  - Signals when complete
- Accelerator computes on data
- Accelerator DMAs results back to allocated memory space

# When To Build Custom?


- Custom can offer better performance or energy
- Why don't we build it for everything?

# Custom Design Costs


- Design costs
  - RTL and verification
  - Driver software
- Area costs
  - Only handles one algorithm
  - Potentially limited use/reuse

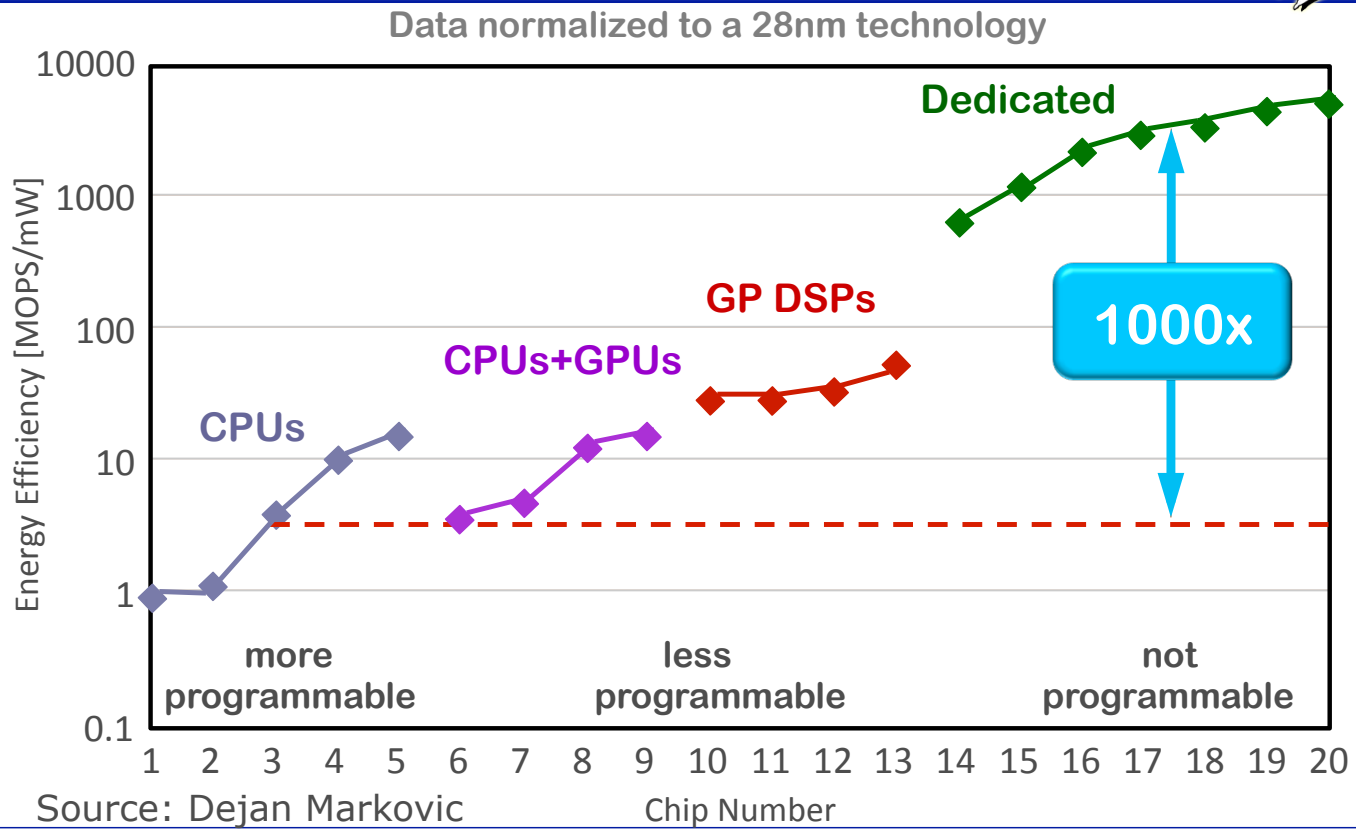
# Is it worth it?


- How often will it be used?
  - May be able to generalize the accelerator at minor extra cost
- How much math does it do?
  - Are there enough ops to justify sending data to an accelerator?
  - Is the math lower/different precision?
- How much control flow does it have?
  - Control dominated applications may need processor-like flexibility
- What do memory accesses look like?
  - Are we spending most of our energy on DRAM accesses anyway?

# Flexibility vs Efficiency


- Processors are programmable
  - Flexible – can target a large variety of problems
  - Inefficient – energy overheads of programmability
- Custom hardware is efficient
  - Efficient – does not incur overheads related to programmability
  - Inflexible – only does one thing, cannot be "reprogrammed" in a general way
- Can we do both?

# Flexibility vs Efficiency




Source: Dejan Markovic

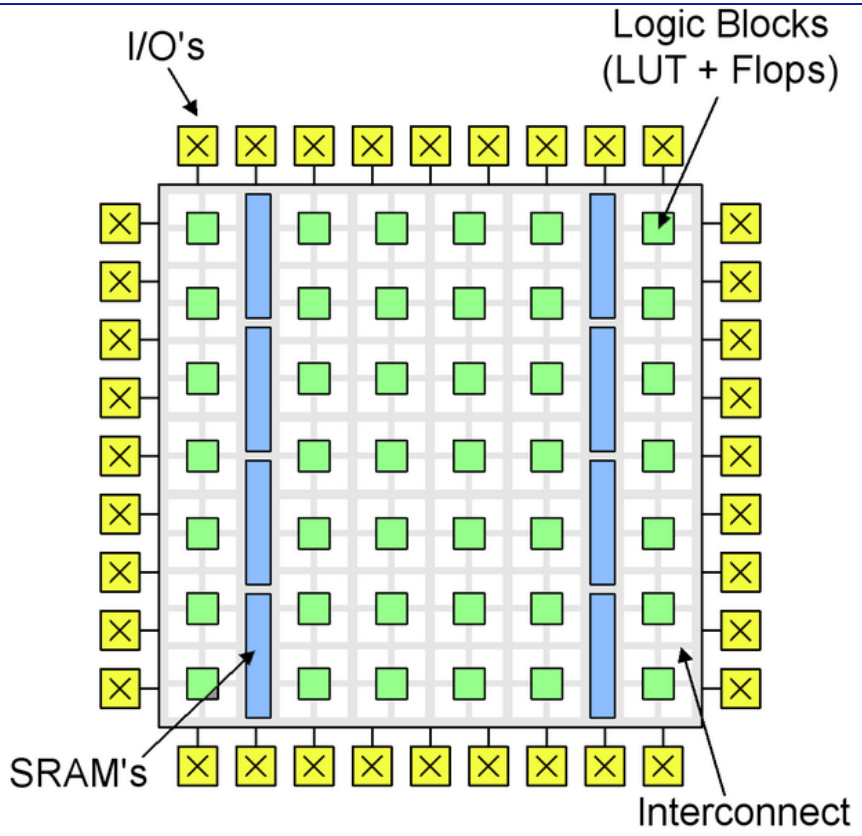
# Reconfigurable Architectures


- Architecture composed of reconfigurable building elements
  - Compute, Memory, Interconnect
- Building blocks laid out as a 1-D or 2-D array on the chip
  - Hence the name "Spatial Architectures"
- Configuration bits controls operations of each element
  - Usually stored in dedicated SRAM cells that drive a MUX select signal to control a reconfigurable element
- Different configurations implement different "custom hardware"
  - Same underlying architecture, different config bits

# Example - FPGAs

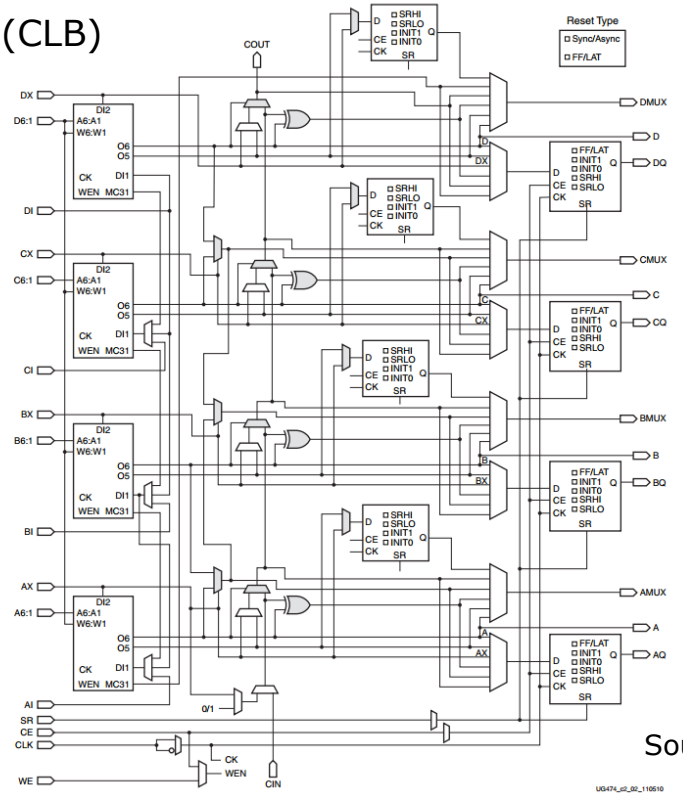

- Field-programmable Gate Array
- 2-dimensional array of reconfigurable logic elements
  - Look-up Tables (LUTs), Flip-flops, Adders, Block RAM (BRAMs), DSP blocks
- Static Programmable Interconnect
  - Switch box, connection box
- Input/Output interface at chip boundary
- Uses:
  - Digital circuit emulation
  - Custom hardware accelerators

# Example - FPGAs



# Reconfigurable Compute

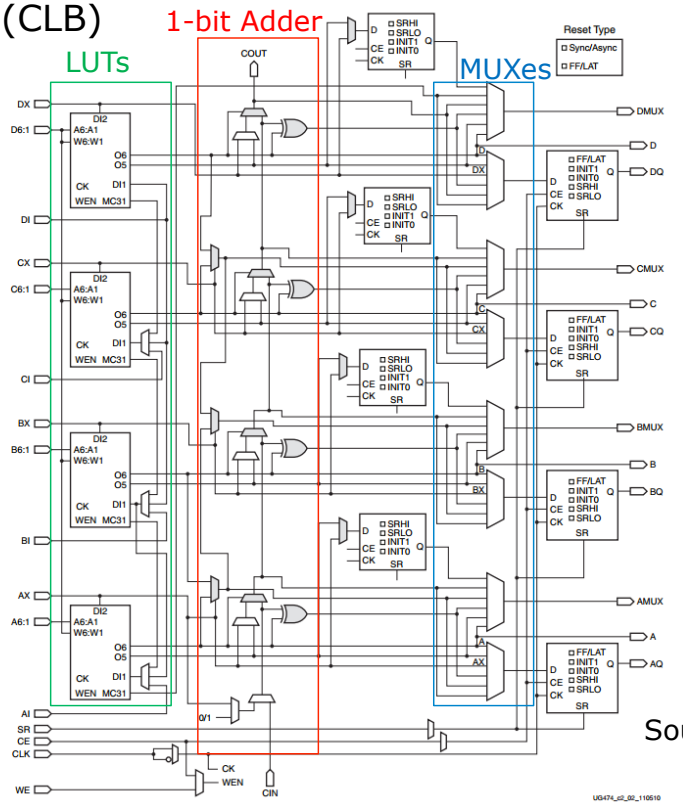
Xilinx Configurable Logic Block (CLB)




Source: 7 Series FPGAs CLB User Guide

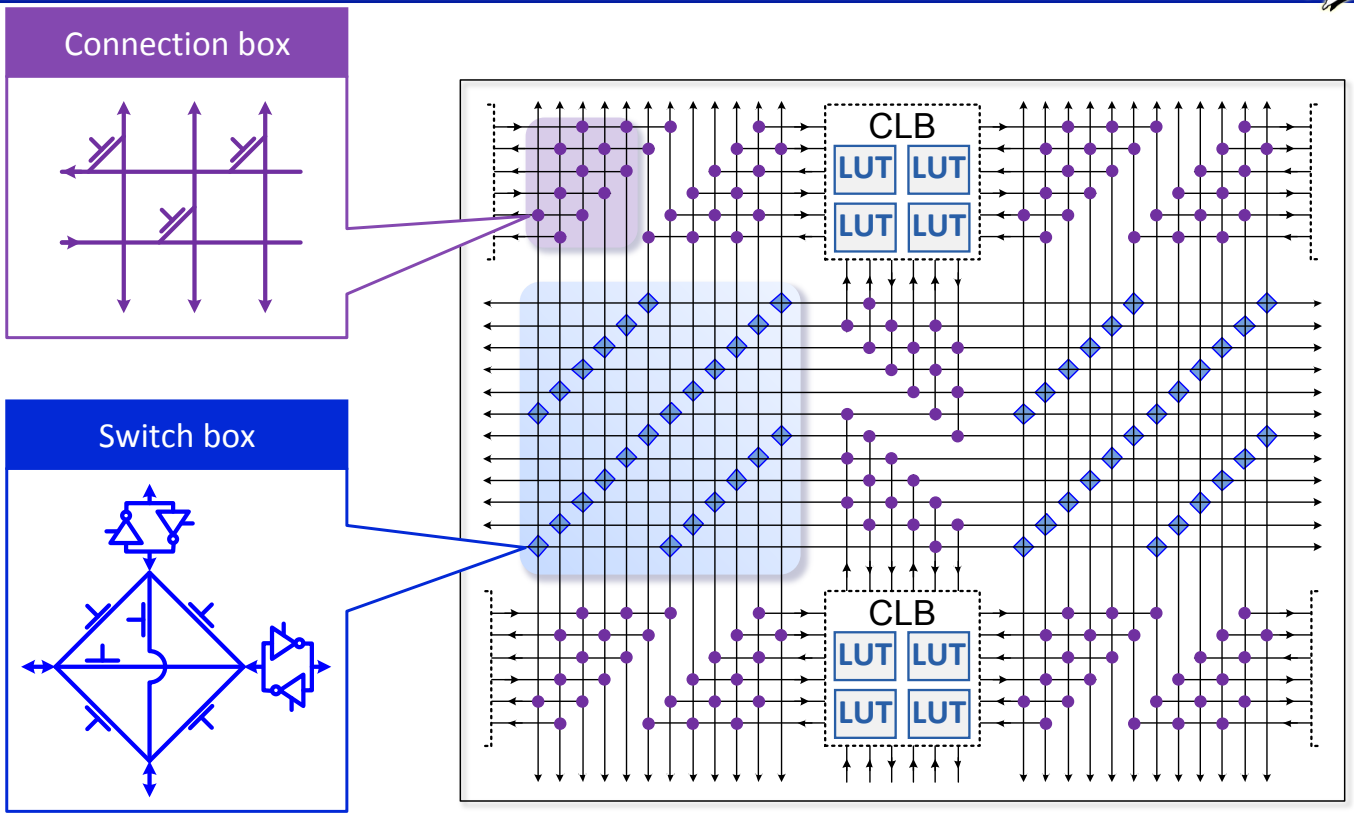
# Reconfigurable Compute

Xilinx Configurable Logic Block (CLB)




Source: 7 Series FPGAs CLB User Guide

# Static Programmable Interconnect

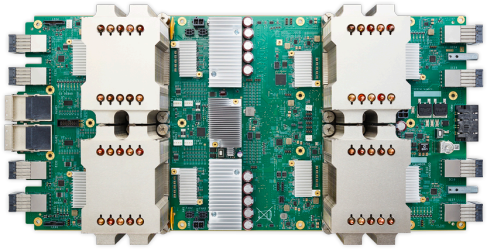


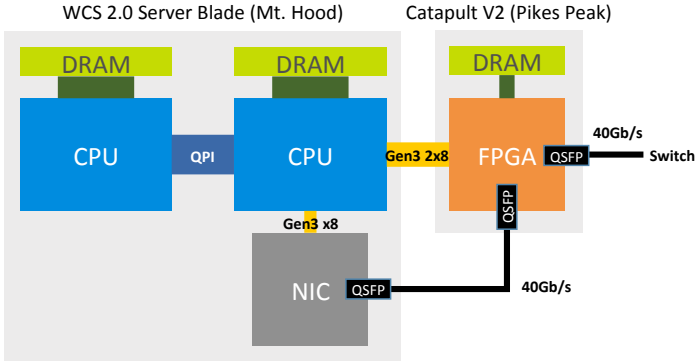
# FPGAs – Pros and Cons


- Pros
  - More flexible than custom hardware
  - Cheaper than custom hardware – same chip is reused across designs
  - Can be more efficient than a general-purpose processor, especially Perf/W
  - Great for circuit emulation, validation
- Cons
  - Less efficient than custom hardware
  - Less programmable than general purpose processor
  - Slow programming toolchain (compare compile times for lab2 and lab3)

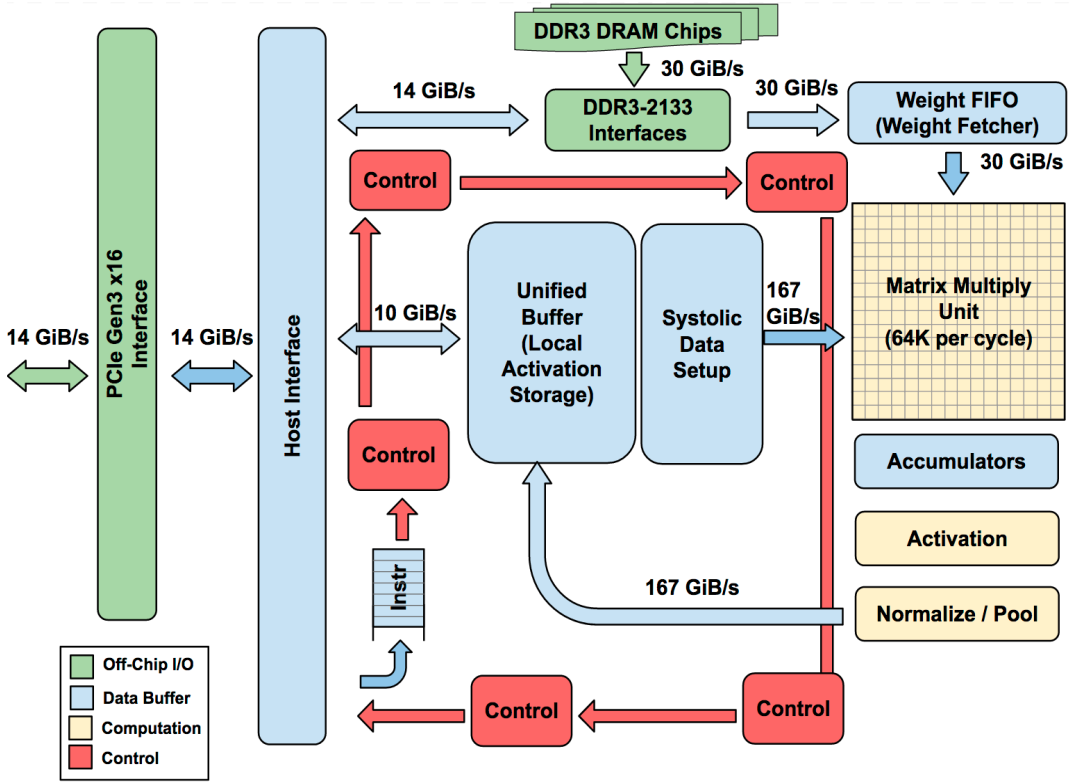
# Acceleration in Industry

- Cloud
  - Google TPU & Cloud TPU
  - Amazon F-1 FPGA instances
  - Azure FPGA
- Mobile
  - Most chips include accelerators
  - Image, video, audio, ML, …





# Google TPU


DDR3 DRAM Chips
30 GiB/s
14 GiB/s
DDR3-2133 Interfaces
30 GiB/s
Weight FIFO (Weight Fetcher)
30 GiB/s
Control
Control
14 GiB/s
PCIe Gen3 x16 Interface
14 GiB/s
Host Interface
10 GiB/s
Unified Buffer (Local Activation Storage)
Systolic Data Setup
167 GiB/s
Matrix Multiply Unit (64K per cycle)
Accumulators
Activation
Normalize / Pool
Control
Instr
167 GiB/s
Control
Control
Off-Chip I/O
Data Buffer
Computation
Control

# Summary


- Programmable hardware → generality, design reuse
- Custom hardware → efficiency

- How to achieve both
  - Use both programmable & custom hardware
  - Generalize custom hardware for a domain or class of algorithms