ESE532:
System-on-a-Chip Architecture

Day 8: September 27, 2017
Spatial Computations

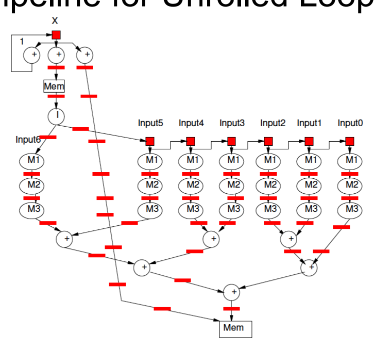
Today

- Accelerator Pipelines
- FPGAs
- Zynq Computational Capacity

Message

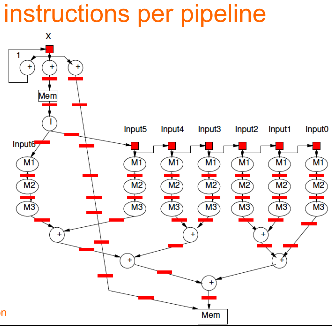
- Custom accelerators efficient for large computations
  - Exploit Instruction-level parallelism
  - Run many low-level operations in parallel
- Field Programmable Gate Arrays (FPGAs)
  - Allow post-fabrication configuration of custom accelerator pipelines
  - Can offer high computational capacity

Pipeline for Unrolled Loop



Preclass 1

- For fully unrolled loop shown, how many instructions per pipeline cycle?
  - Add
  - Mpy
  - Load
  - Store



Spatial Pipeline

- Can compute equivalent of tens of "instructions" in a cycle
- Wire up primitive operators
  - No indirection through RF, memory
- Pipeline for operator latencies
- Any dataflow graph of computational operations

Operators

- Can assemble any custom operators
  - Ones may not have in generic processor
- Processor
  - Add, bitwise-xor/and/or, multiply
  - Maybe: floating-point add, multiply
- Less likely
  - Square-root, exponent, cosine, encryption (AES) step, polynomial evaluate, log-number-system

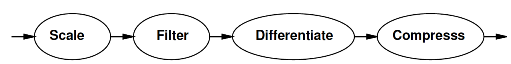
Accelerators

- Compression/decompression
- Encryption/decryption
- Encoding (ECC, Checksum)
- Discrete Cosine Transform (DCT)
- Sorter
- Taylor Series Approximation of function
- Transistor evaluator
- Tensor or Neural Network evaluator

Streaming Dataflow

- Replace operator with custom accelerator
- Stream data to/from it

Streaming Dataflow Example



Application-Specific SoCs

- For dedicated applications may build custom hardware for accelerators
  - Layout VLSI, fab unique chips
  - ESE370, 570
- Tensillica – custom instructions
- Video-encoder – include custom DCT, motion-estimation engines

Customizable Accelerators

- With post-fabrication configurability can exploit without unique fabrication
- Need programmable substrate that allows us to wire-up computations

Field-Programmable Gate Arrays

FPGAs

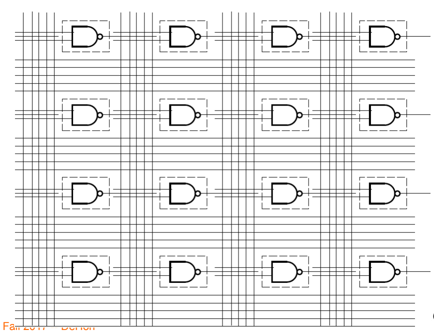
FPGA

- Idea: Can wire up programmable gates in the "field"
  - After fabrication
  - At your desk
  - When part "boots"
- Like a "Gate Array"
  - But not hardwired

Gate Array

- Idea: Provide a collection of uncommitted gates
- Create your "custom" logic by wiring together the gates
- Less layout and masks than full custom
  - Since only wiring together pre-fab gates
  - → lower cost (fewer masks)
  - → lower manufacturing delay

Gate Array

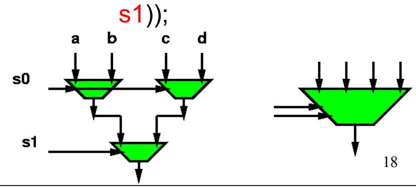


GA→FPGA

- Remove the need to even fabricate the wiring mask
- Make "customization" soft
- Key trick:
  - Use reprogrammable configuration bits
  - Typically: static-RAM bits
    - Like SRAM cells or latches
    - Hold a configuration value

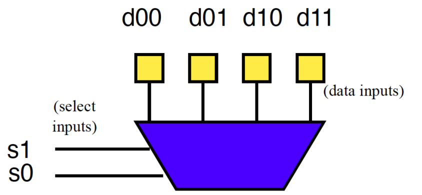
Mux with configuration bits = programmable gate

- bool mux4(bool a, b, c, d, s0, s1) {
  return(mux2(mux2(a,b,s0),
  mux2(c,d,s0),
  s1));
  }



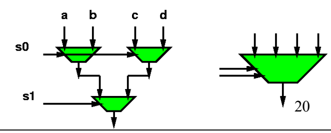
Preclass 2a

- How do we program to behave as and2?



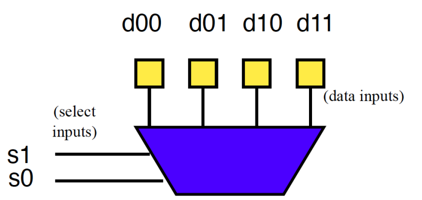
Mux as Logic

- bool and2(bool x, y)
  {return (mux4(false,false,false,true,x,y));}



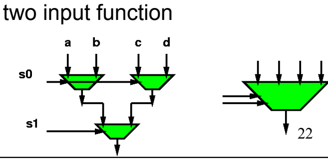
Preclass 2b

- How do we program to behave as xor2?



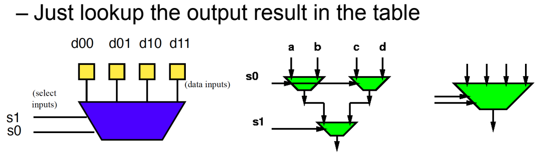
Mux as Logic

- bool and2(bool x, y)
  {return (mux4(false,false,false,true,x,y));}
- bool xor2(bool x, y)
  {return (mux4(false,true,true,false,x,y));}
- Just by “configuring” data into this mux4,
  - Can select **any** two input function



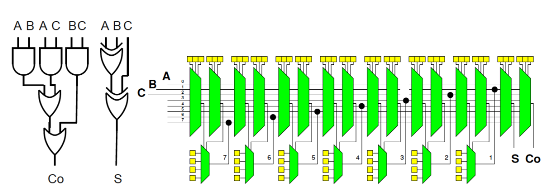
LUT – LookUp Table

- When use a mux as programmable gate
  - Call it a LookUp Table (LUT)
  - Implementing the Truth Table for small # of inputs
    - # of inputs =k (need mux-2^k)
  - Just lookup the output result in the table



Preclass 3

- How do we program full adder?



FPGA

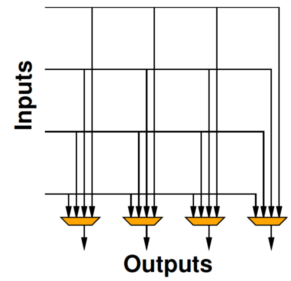
- Programmable gates + wiring
  - (both built from muxes w/ config. bits)
- Can wire up any collection of gates
  - Like a gate array

Penn ESE532 Fall 2017 -- DeHon

25

Crossbar Interconnect

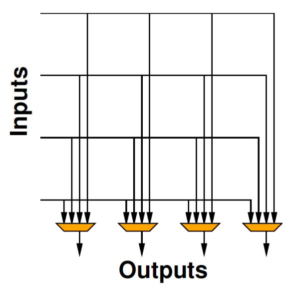
- I-inputs
- O-outputs
- Can connect any input to any output
- Functionally equivalent to
  - I-input Mux for each output



Penn ESE532 Fall 2017 -- DeHon

Crossbar Scaling

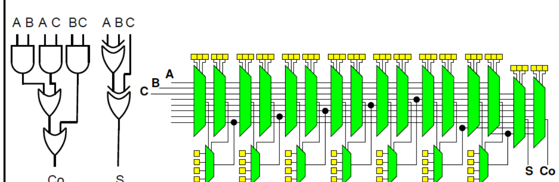
- How many 2-input muxes to build an I-input mux?
- How does crossbar scale with I, O?



Penn ESE532 Fall 2017 -- DeHon

Crossbar Interconnect

- How would crossbar interconnect scale with number of gates N?



Penn ESE532 Fall 2017 -- DeHon

28

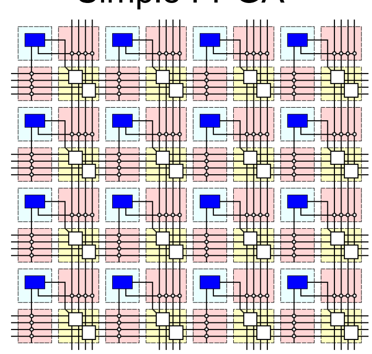
Crossbar Interconnect

- Crossbar interconnect is too expensive
  - And not necessary
- Want
  - To be able to wire up gates
  - Economical with wires and muxes
    - …and configuration bits
  - Exploit locality (keep wires short)

Penn ESE532 Fall 2017 -- DeHon

29

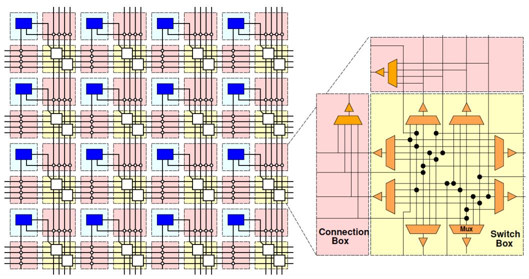
Simple FPGA



Penn ESE532 Fall 2017 -- DeHon

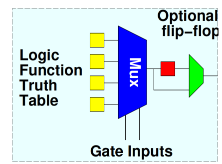
30

Simple FPGA

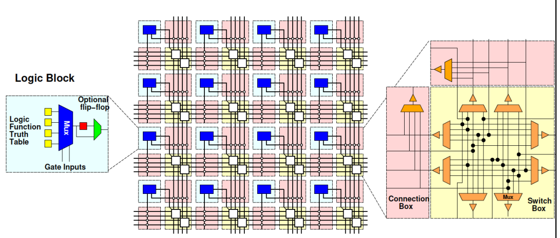


Flip-Flops

- Want to be able to pipeline logic
- …and generally hold state
  - E.g. implement hold Input-N in preclass 1
- Add optional flip-flop on each gate



Simple FPGA



FPGA Design

- Raises many architectural design questions
  - How big (many inputs) should the gates have?
    - Are LUTs really the right thing…
  - How rich is the interconnect?
    - Wires/channel
    - Wire length
    - Switching options

Modern FPGAs

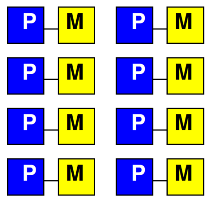
- Logic Blocks
  - hardwired fast-carry logic
    - Can implement adder bit in single "LUT"
  - Speed optimized: 6-LUTs
  - Energy, Cost optimization: 4-LUTs
  - Clusters many LUTs into a tile
- Interconnect
  - Mesh, segments of length 4 and longer

More than LUTs

- Should there be more than LUTs in the "array" fabric?
- What else might we want?

Embedded Memory

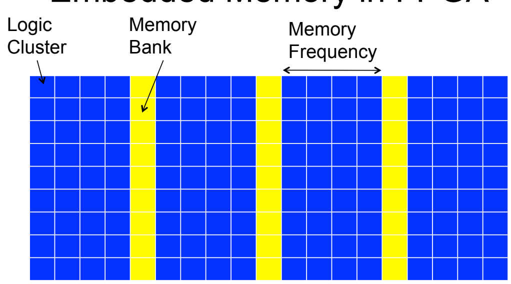
- One flip-flop per LUT doesn't store state densely
- Want memory close to logic



Embed Memory in Array

- Replace logic clusters
- Convenient to replace columns
  - Since area of memory may not match area of logic cluster

Embedded Memory in FPGA

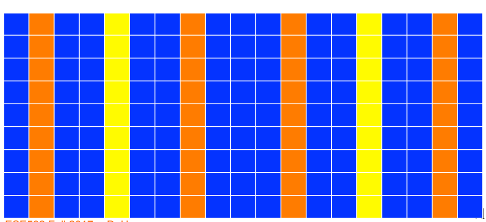


Hardwired Multipliers

- Can build multipliers out of LUTs
  - Just as can implement multiplies on processor out of adds
- But, custom multiplier is smaller than LUT-configured multiplier
  - …and multipliers common in signal processing, scientific/engineering compute

Multiplier Integration

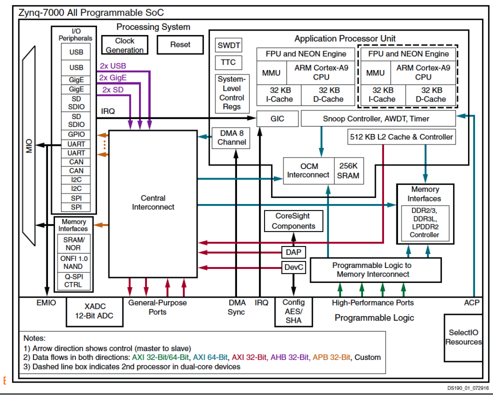
- Integrate like memories
  - Replace columns



More FPGA Architecture Design Questions

- Size of Memories? Multipliers?
- Mix of LUTs, Memories, Multipliers?
- Add processors? Floating-point?
- Other hardwired blocks?
- How manage configuration?

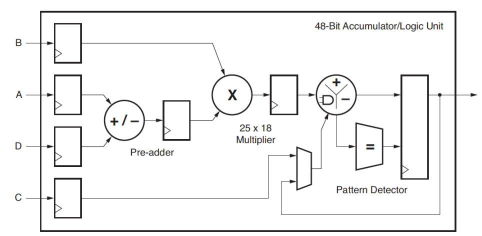
Zynq



XC7Z020

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

DSP48



Xilinx UG479 DSP48E1 User's Guide

Preclass 4

Resoruces	Cycle	per second
50,000 adder bits	0.2 GHz	
128×2 adder bits	1.0 GHz	
200 multiply-accumulates	0.2 GHz	
8×2 multiply-accumulates	1.0 GHz	

Compute Capacity

- How compare between ARM/NEON and FPGA array?
  - Adder-bits/second?
  - Multiply-accumulators/second?

Capacity → Density

- Says Zynq has high computational capacity in FPGA
- More broadly
  - FPGA can have more compute/area than processor
    - E.g., more adder bits in some fixed area
  - SIMD can have more compute/area than processor
    - How wide SIMD can you exploit?

FPGA Potential

- FPGA Array has high raw capacity
- Exploitable when computation has high regularity
  - Uses the same computation over-and-over
  - High throughput on a computation
  - Build customized accelerator pipeline to match the computation
- Low-hanging fruit
  - Operator/function takes most of the compute time

90/10 Rule (of Thumb)

- Observation that code is not used uniformly
- 90% of the time is spent in 10% of the code
- Knuth: 50% of the time in 2% of the code
- Opportunity
  - Build custom datapath in FPGA (hardware) for that 10% (or 2%) of the code

Big Ideas

- Custom accelerators efficient for large computations
  - Exploit Instruction-level parallelism
  - Run many low-level operations in parallel
- Field Programmable Gate Arrays (FPGAs)
  - Allow post-fabrication configuration of custom accelerator pipelines
  - Can offer high computational capacity

Admin

- Reading for Day 9 on canvas
- HW4 due Friday
- No homework due 10/6 (Fall Break)
- HW5 out
  - Due 10/13
  - SDSoC synthesis at end slow (plan for it)