

ECE 497NC: Unconventional Computer Architecture

Lecture 7: Reconfigurable Computing 7: Purely-Reconfigurable Systems

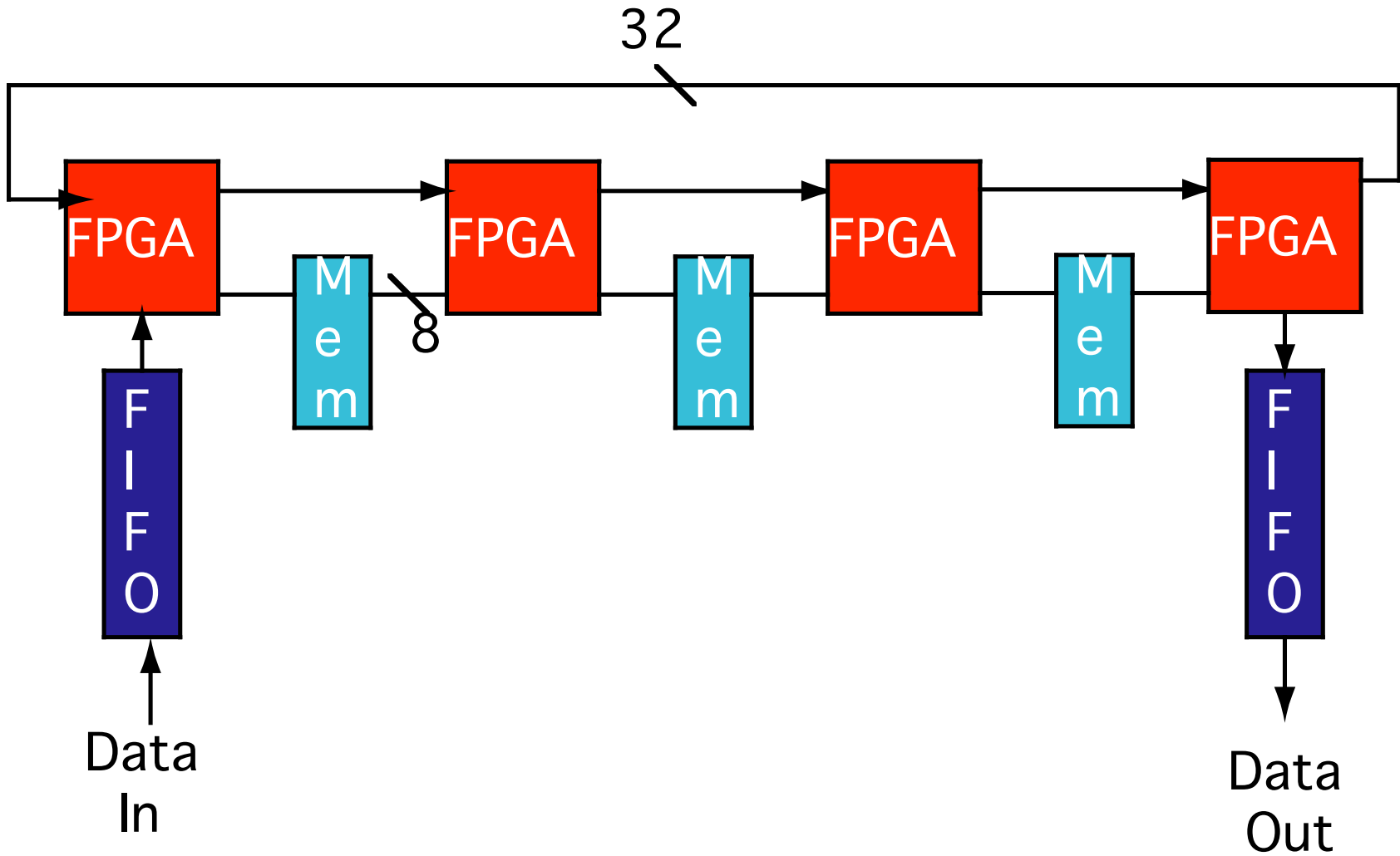
Outline

- Splash -- computing systems based on conventional FPGAs
- PipeRench -- a system that uses reconfigurable “stripes” of small ALUs
- Teramac -- using reconfigurable logic to implement defect-tolerant computing

Splash 1, 2

- Basic idea: build FPGA cards that plug into the standard I/O bus of Sun computers.
 - Offload compute-intensive portions of programs onto the FPGA boards to improve performance
- Splash 1 -- 32 FPGA chips on a VMEbus board
- Splash 2 -- 17 (higher-performance) FPGAs on a SBus board
 - Higher bandwidth to host processor
 - Better on-board communication
 - Different memory organization

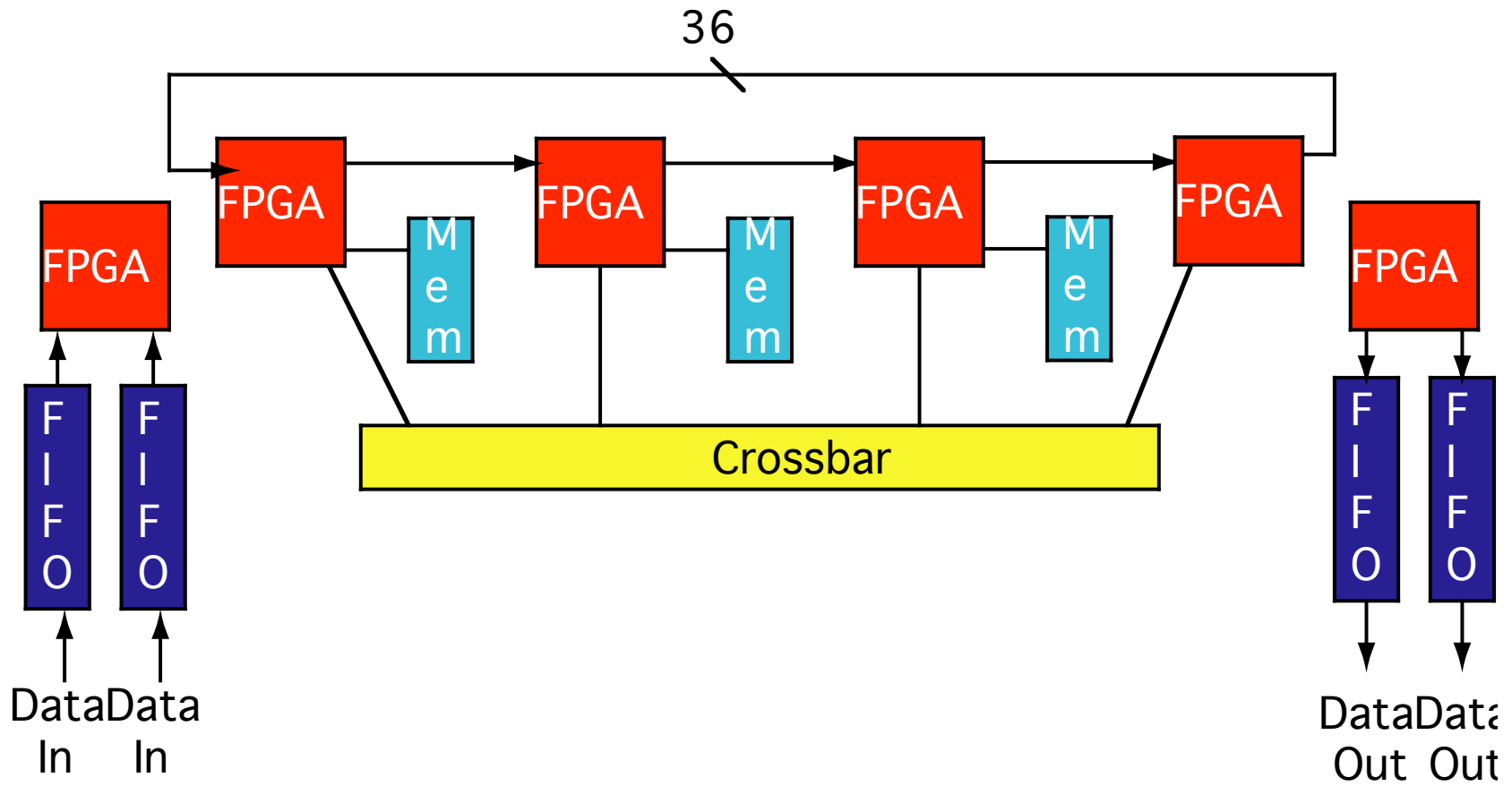
Splash 1



Programming Splash 1

- Hand-designed circuits for the FPGAs using Xilinx editors or schematic capture tools.
- Very limited communication capabilities -- data can only flow from one FPGA to the next one on the ring.
- Programs often limited by I/O off of the board.
 - Depends a lot on whether the program matches the communication system. If so, I/O often bottleneck. If not, performance may be bad.

Splash 2



Splash 2

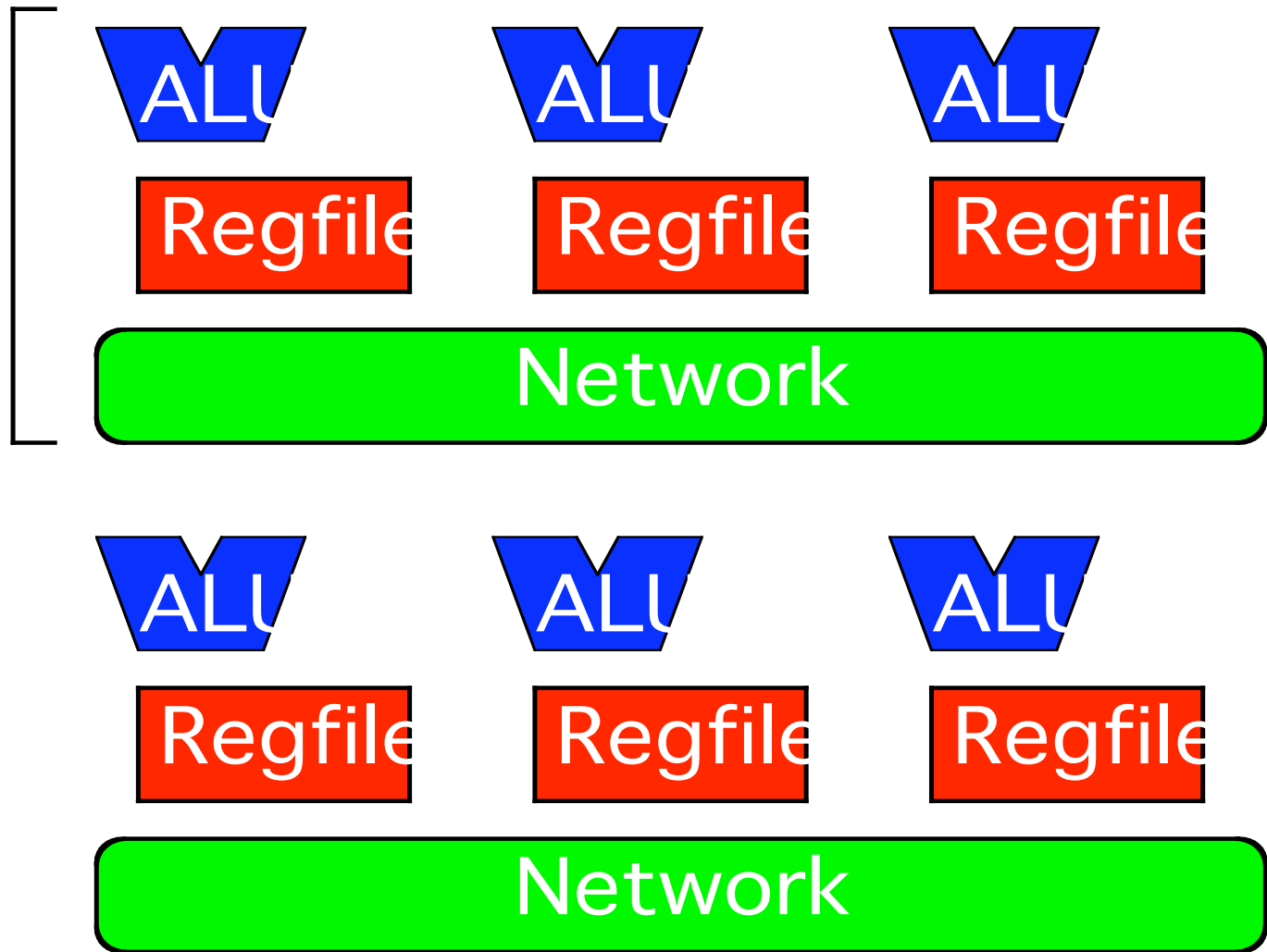
- Programmed in VHDL
 - Need to program the computing FPGAs, chips that control the crossbars, I/O control FPGAs
- Two input and two output queues
- Programmable crossbar allows more complex communication patterns
- Supports multiple boards in a system
- Keyword search application expected to be performance-limited by bandwidth to host, DNA matching should exceed supercomputer performance by two orders of magnitude.

PipeRench

- Goal: Design reconfigurable hardware to support computing applications
 - Provide small-bit computations rather than individual logic operations
 - Low configuration time to support run-time reconfiguration
 - Support programs that don't fit in the reconfigurable fabric
 - Allow compatibility between product generations
 - Fast compilation time

PipeRench Stripes

Stripe



Pipeline Virtualization in PipeRench

- Problem: Conventional FPGAs fail catastrophically when program size exceeds capacity of the FPGA.
 - Other problem: No code compatibility between different generations of a device.
- Approach: Organize hardware into “stripes” that can be reconfigured in a single clock cycle.
 - If hardware provides as many stripes as the pipeline has stages, configure each stripe before the first data reaches it, then produce one result/cycle.
 - If hardware does not provide enough stripes, “wrap around” when reach bottom of the array, reconfiguring the stripes just before data reaches them
 - Limits communication paths through array
 - V-stage pipeline on device with p stripes has throughput proportional to $(p-1)/v$
 - Program performance improves automatically as size of hardware increases, until $p = v$.

PipeRench Performance

- Varies with number of registers/computing element, though not monotonically
 - Consider area impact of registers on number of computing elements per unit area
- With 8 registers/element, get speedups of 10-100x over 300MHz UltraSparc on compute kernels (assumes 100MHz PipeRench)
- Simulating entire program performance using kernels on PipeRench gives much lower improvements -- 7-12%
 - Amdahl's Law hurts a lot.

Teramac -- Fast Computing With Broken Hardware

- Issue: Standard IC and system designs require perfect fabrication -- can't tolerate even a single defect
 - Slight exaggeration -- memory arrays usually designed with redundancy
- FPGAs have potential to tolerate large numbers of faults
 - Implemented out of large numbers of identical sub-components
 - Potential to reconfigure around failures
- Teramac: Configurable supercomputer implemented using FPGAs at all levels
 - FPGAs as computing units
 - FPGAs implement network (programmable crossbars)

Fault-Tolerance In Teramac

- Many FPGAs in system, organized as multiple boards, each with four multi-chip modules.
 - Wiring in multi-chip modules, connections between boards implemented using cheap but error-prone technologies.
- First program run was hardware “mapper” that determined how chips were connected, where defects were
 - Only a small fraction of the hardware on each chip was critical.
 - Algorithms generate stream of pseudo-random bits that get sent to each neighbor on the chip.
 - Defects located by row, column of incorrect results.

Results

- Appx. 75% of the chips used in Teramac had at least one defect
 - Big cost savings -- normally those 75% would have been destroyed by manufacturer.
 - Chips designed to limit the amount of critical hardware and to make that hardware less failure-prone
- About 3% of the total “resources” (logic cells, wires, etc.) in Teramac were defective
 - Total of 220,000 defects
- In spite of this, they were able to achieve extremely high performance on a set of applications by mapping around defects

How Does This Relate to Nanotechnology?

- Silicon fabrication gives extremely low defect rates per device
 - Defect-intolerant design requires that defect rate improve at same rate as number of devices/chip
- Many alternatives to silicon VLSI are expected to have much higher defect rates
 - As high as a few percent for some technologies
 - Basically means that we'll get zero defect-free chips
- Architectures like Teramac offer a way to get correct operation out of defective chips.