# CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #7 – Applications I

## Quick Points

- HW #2 is out
  - Due Tuesday, September 26 (12:00pm)
  - Includes an instant extension from the 21st



## Recap – Fixed-Point Arithmetic

- Addition, subtraction the same (Q4.4 example):

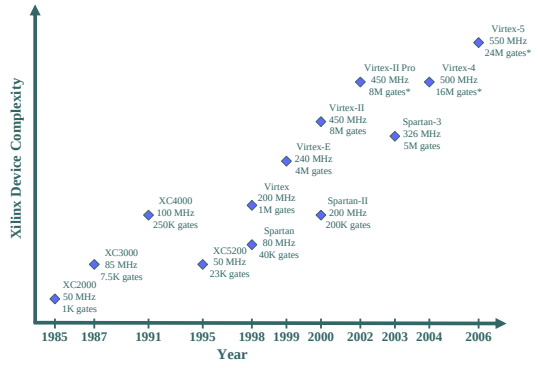
```
  3.6250    0011.1010
+ 2.8125    0010.1101
  6.4375    0110.0111
```

- Multiplication requires realignment:

```
  3.6250              0011.1010
x 2.8125              0010.1101
                       00111010
                      00111010
                     00111010
                    00111010
 10.1953125         1010.00110010
```

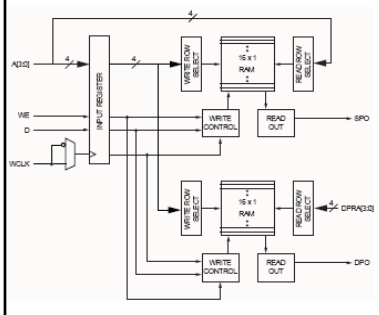


## Recap – Capacity Trends

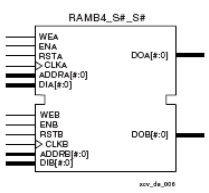





## Recap – FPGA Memory Resources




Distributed LUT-based RAM

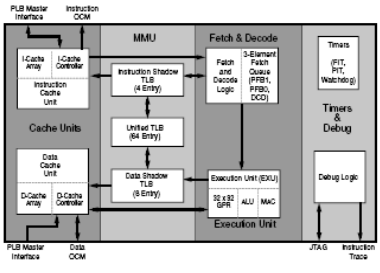


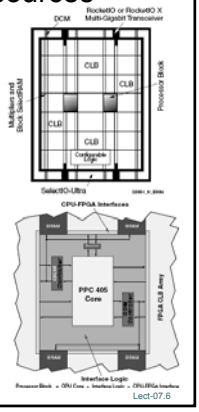

Block Select+RAM



## Recap – FPGA CPU Resources

- Built-in PowerPC 405 processor blocks



## Outline

- Recap
- Splash / Splash 2 System Architecture
- Splash 2 Programming Models
- Splash 2 Applications
  - Text searching
  - Genetic pattern matching
  - Image processing



## Overview

- An early well-known reconfigurable computer was Splash / Splash 2
- Implemented as linear, systolic array
- Developed at Supercomputing Research Center (1990-1994)
- Memory tightly coupled with each FPGA
- Multiple Splash boards can be combined to form larger system

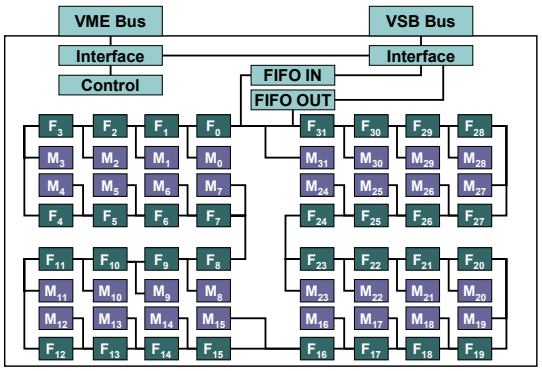


## Splash 1

- Born to solve DNA string matching
- Operational in 1989
- Features
  - 32 Xilinx XC3090 FPGAs (420 Mλ$^2$)
  - 32 128KB SRAMs (600 Mλ$^2$)
  - VMEbus interface (FIFO 1 MHz clock)
- 33 Gλ$^2$ total (not counting interconnect)
- Linear interconnect only



## Splash 1 Architecture



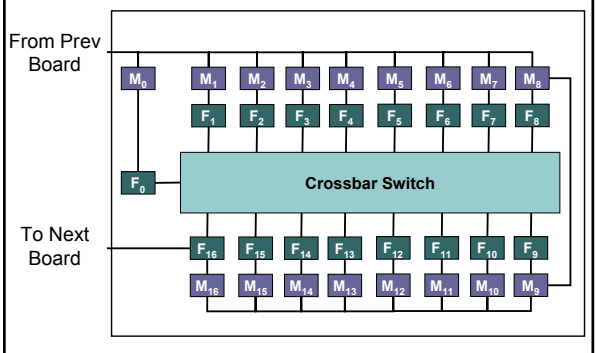



## Splash 2

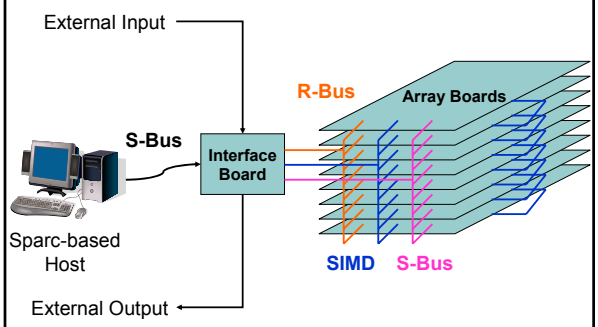
- Operational in 1992
- Attached processor system
- Features
  - 17 Xilinx XC4010 FPGAs (500 Mλ$^2$)
  - 17 512KB SRAMs (2 Gλ$^2$)
  - 9 TI SN74ACT8841s (16 port, 4b crossbar)
  - Sbus interface (< 100MB/s)
- 43 Gλ$^2$ total (not counting interconnect)
- Supported 2 programming models:
  - Systolic
  - SIMD



## Splash 2 Architecture



## Splash 2 Architecture (cont.)






## Comparing Splash 1 and 2

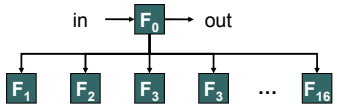
| | Splash 1 | Splash 2 |
|---|---|---|
| **Year** | 1989 | 1992 |
| **FPGA** | XC3090 | XC4010 |
| **4-LUTs / Board** | 10,240 | 13,600 |
| **Max Input Rate** | 1 MB/s | 100 MB/s |
| **Interconnect** | Linear Array | Linear Array<br>Broadcast<br>Crossbar |
| **Memory / FPGA** | 128 KB | 512 KB |
| **Area** | 33 Gλ² | 43 Gλ² |



## Splash 2 Programming Models

- Linear (systolic) array
  - All near-neighbor communication, pipelined
  - Very fast (at the time) speed of 20-30MHz achieved
  - All FPGAs run the same program


- SIMD array
  - Instructions fanned out to all processing elements
  - Data across all elements collected at the end






## Programming Splash 2

- Languages
  - VHDL
  - dbC – a C-like language capable of SIMD instructions
- How many programs?
  - Host
  - Interface board (XL and XR)
  - Splash array board
    - X0
    - X1 – X16
    - TI crossbar chips
  - **Five** programs!
- Requires intimate knowledge of the architecture



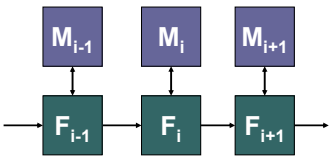
## Application #1 – Dictionary Search

- Search through dictionary of words for data hit
- Applicable to internet search engines / databases
- Opportunities for search parallelism
- Splash implementation uses systolic communication



## Dictionary Search (cont.)

- Each FPGA used to look into local memory
- Longer data words "hashed" into 18 bit address
- Valid bit in memory indicates if data value is currently stored
- Could be stored in several locations



## Example Hash Function

```
Shift amount: 7 bits
Hash function: 1100 1000 1010 0011

00 0000 0000 0000 0000 0000     Clear hash register
01 1010 0001 1101 00            Input the letters "th"
--------------------------------
10 1000 0011 0101 1100 0000     Temporary Result


10 0000 0101 0000 0110 1011     Result for "th"
00 0000 0001 1001 01            Input for letters "e_"
----------------------------------------
01 0010 0110 0001 1110 1011     Temporary result


10 0101 1010 0100 1100 0011     Result for "the_"
```

- XOR two character value with temp result and hash function
- Rotate result
- Different hash function for each FPGA



## Dictionary Search (cont.)

- Distribute dictionary in parallel to all memories
- Collect word values in FIFOs
- Distribute words two characters at a time across all devices
- Perform local hashing and lookup in parallel
- Collect "hit" result at end
- Splash 2 implementation results
  - 25 MHz
  - Three phases needed
    - Fetch 2 bit-sliced characters
    - Perform hash
    - Table look-up
  - Takes advantage of both systolic and SIMD modes



## Genetic Pattern Matching

- Comparing strings by edit distance
- Motivation: The Human Genome Project
  - Do two genetic strings match?
  - How are they related?
- When biologists characterize a new sequence, they want to compare it to the (growing) database of known sequences
- Abstraction:
  - What is the cost of transforming *s* into *t*
  - Given – costs for insertion, deletion, substitution



## Alphabet and Costs

- Alphabet
  - Letters in the string. For DNA, there are four:
    - A (Adenine)
    - C (Cytosine)
    - T (Thymine)
    - G (Guanine)
- Transformation Costs
  - Insert:1, Delete:1, Substitute:2, match:0
- Type of comparison
  - One target to many sources
  - One target to one source



## Substitution Example

| Word | Move | Cost |
|---|---|---|
| baboon | Delete 'o' | 1 |
| bab\|on | Substitute 'o' | 2 |
| bobon | Insert 'u' | 1 |
| boubon | Insert 'r' | 1 |
| bourbon | Match? | 0 |
| **bourbon** | Total cost: | 5 |



## Dynamic Programming Solution

- Source sequence: $s_1, s_2, \ldots s_m$
- Target sequence: $t_1, t_2, \ldots t_n$
- $d_{i,j}$ = distance between subsequence $s_1, s_2, \ldots s_i$ and subsequence $t_1, t_2, \ldots t_j$, where

$$d_{0,0} = 0$$

$$d_{i,0} = d_{i-1,0} + \text{Delete}(s_i)$$

$$d_{0,j} = d_{0,j-1} + \text{Insert}(t_j)$$

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + \text{Delete}(s_i) \\ d_{i,j-1} + \text{Insert}(t_j) \\ d_{i-1,j-1} + \text{Substitute}(s_i, t_j) \end{cases}$$

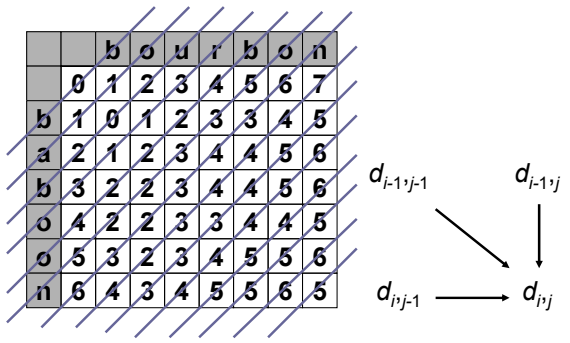
- Distance(Source, Target) = $d_{m,n}$

## Dynamic Programming Example

| | | b | o | u | r | b | o | n |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| b | 1 | | | | | | | |
| a | 2 | | | | | | | |
| b | 3 | | | | | | | |
| o | 4 | | | | | | | |
| o | 5 | | | | | | | |
| n | 6 | | | | | | | |



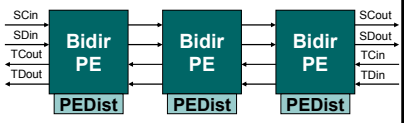
## Parallelism on the Anti-Diagonal






## Bidirectional PE Example

If (SCin != 0) and (TCin != 0)
  PEDist ← min { PEDist + Substitute(SCin, TCin); TDin + Delete(SCin); SDin + Insert(TCin) }
else-if (SCin != 0)
  PEDist ← SDin
else-if (TCin != 0)
  PEDist ← TDin
endif
SCout ← SCin
TCout ← TCin
SDout ← PEDist
TDout ← PEDist






## Bidirectional Summary

- 16 CLBs/PE
- 384 PEs/Board
- 2,100 Million Cells/sec
- Requires 2*(*m*+*n*) PEs
- Uses only half the processors at any one time
- Must stream both source and target for each comparison
  - Makes comparison against large DB impractical



## Genetic Search Performance

- Nearly linear scaling in cell updates per second (CUPS)
- Need to reuse array for large patterns

| Hardware | CUPS | λ | Area | CUP/λ²s |
|---|---|---|---|---|
| Splash 2 x16 | 43,000M | 0.60*μ* | 500Mλ²x17x16 | 0.32 |
| Splash 2 | 3,000M | 0.60*μ* | 500Mλ²x16 | 0.38 |
| Splash 1 | 370M | 0.60*μ* | 420Mλ²x32 | 0.028 |
| P-NAC (34) | 500M | 2.0*μ* | 7.8Mλ²x34 | 1.9 |
| CM-2 (64K) | 150M | ? | | |
| CM-5 (32) | 33M | ? | | |
| SPARC 10 | 1.2M | 0.40*μ* | 1.6GMλ² | 0.00075 |
| SPARC 1 | 0.87M | 0.75*μ* | 273Mλ² | 0.0032 |



## Application #3 – Image Processing

- Reconfigurable computers well suited to image processing due to high parallelism and specialization (filtering)
- Algorithms change sufficiently fast such that ASIC implementations become outdated
- Examine two issues with Splash
  - Image compression
  - Image error estimation
- Parallelize across array in SIMD and systolic fashion

## Gaussian Pyramid Operations

- Gaussian Pyramid
  - Down sample image to compress image size for communication

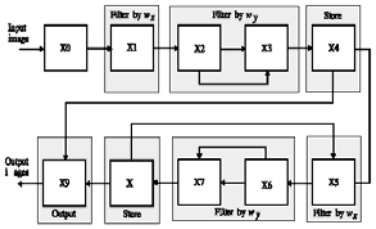
$$g_k(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n)\, g_{k-1}(2i+m, 2j+n)$$

  - Average over a set of points to create new point

- Laplacian Pyramid
  - Determine error found from Gaussian Pyramid
  - Expand contracted picture and compare with original



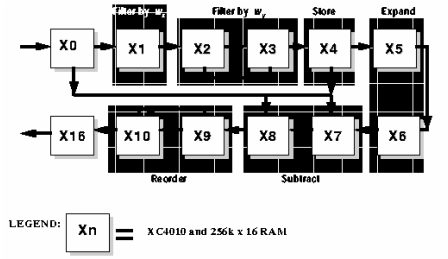
## Gaussian Pyramids




- Systolic array in which each device performs a separate function
- Limited by clock rate of slowest device



## Pyramid Flow




- Generates both Gaussian and Laplacian pyramid for 512 x 480 image in 22.7 ms at 15.7 MHz



## Other Image Processing

- Target recognition
- Break image into “chips”
- Each chip passed through linear array in attempt to match with stored image
- Images can be rotated, mirrored
- Zoom in if suspicious object found



## Summary

- Splash 2 effective due to scalability and programming model
- Parameterizable applications benefit that are regular and distributed
- High bandwidth effective for searching/signal processing
- Challenges remain in software development