# CprE / ComS 583 Reconfigurable Computing

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

Lecture #8 – Reconfigurable Networking

## Recap – 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

$$d_{0,0} = 0$$
$$d_{i,0} = d_{i-1,0} + \text{Delete}(s_i)$$
$$d_{0,j} = d_{0,j-1} + \text{Insert}(t_j)$$
$$d_{i,j} = \min \begin{cases} d_{i-1,j} + \text{Delete}(s_i) \\ d_{i,j-1} + \text{Insert}(t_j) \\ d_{i-1,j-1} + \text{Substitute}(s_i, t_j) \end{cases}$$

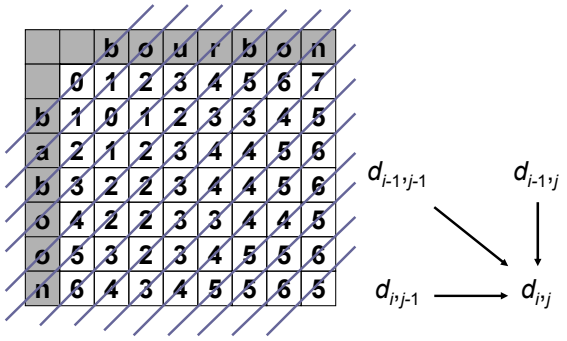
- Distance(Source, Target) = $d_{m,n}$



## 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

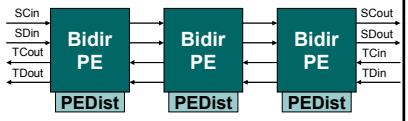
endif

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



## Outline

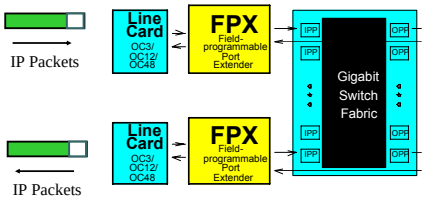
- Recap – Pattern Matching on Splash-2
- The Field-Programmable Port Extender (FPX)
- FPX Architecture
- FPX Programming Model
- FPX Applications
  - Pattern Matching
  - Packet Classification
  - Rule Processing



## Application – Network Processing

- Networking applications well-suited for reconfigurable hardware
  - Target signatures change often
  - Massive quantities of stream-based data
  - Repetitive operations
- Connecting up to a realistic networking environment is hard
  - Washington University experimental setup one of the best
  - Shows importance of both memory and processing capability
- Numerous experiments performed over the past five years

## Network Routing with the FPX



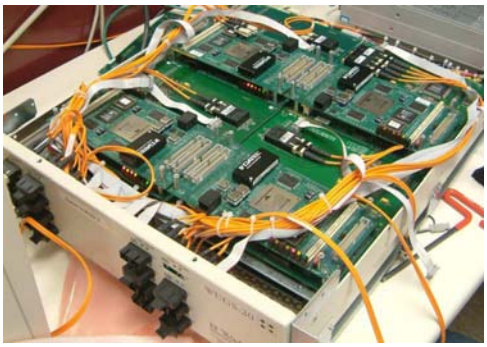

- FPX Modules distributed across each port of a switch
- IP packets (over ATM) enter and depart line card
- Packet fragments processed by modules
- Advantages:
  - New protocols implemented directly in silicon
  - Easy to upgrade in the field



## FPX Hardware Device

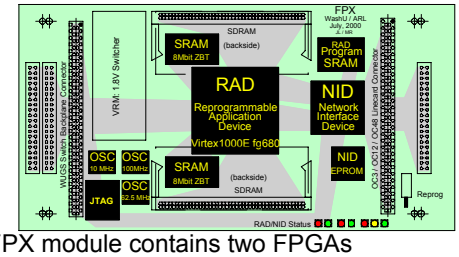


## FPX Hardware in a WUGS-20 Switch





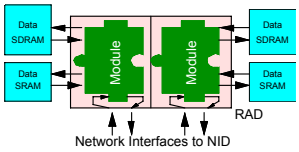
## FPGA-based Router




- FPX module contains two FPGAs
- NID – network interface device
  - Performs data queuing
- RAD – reprogrammable application device
  - Specialized control sequences



## Reprogrammable Application Device

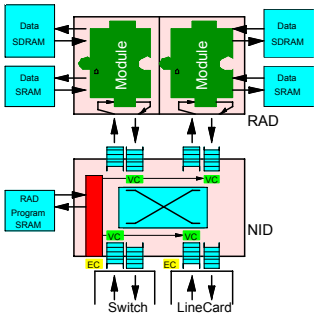



- Spatial Re-use of FPGA Resources
  - Modules implemented using FPGA logic
  - Module logic can be individually reprogrammed
- Shared Access to off-chip resources
  - Memory Interfaces to SRAM and SDRAM
  - Common Datapath to send and receive data

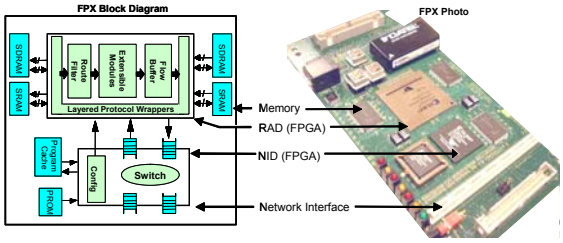


## Architecture of the FPX

- **RAD**
  - Large Xilinx FPGA
  - Attaches to SRAM and SDRAM
  - Reprogrammable over network
  - Provides two user-defined Module Interfaces
- **NID**
  - Provides Utopia Interfaces between switch & line card
  - Forwards cells to RAD
  - Programs RAD



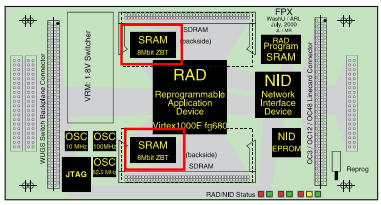
## Architecture of the FPX (cont.)






## FPX SRAM

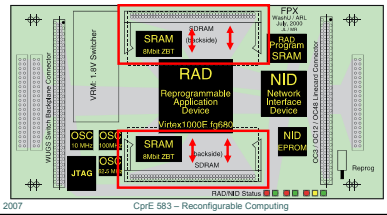
- Provide low latency for fast table-lookups
- Zero Bus Turnaround (ZBT) allows back-to-back read / write operations every 10ns
- Dual, Independent Memories
- 36-bit wide bus






## FPX SDRAM

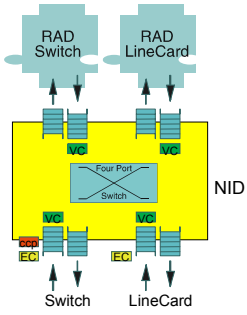
- Dual, independent SDRAM memories
- 64-bit wide, 100 MHz
- 64Mb / Module : 128 Mb total [expandable]
- Burst-based transactions [1-8 word transfers]
- Latency of 14 cycles to Read/Write 8-word burst





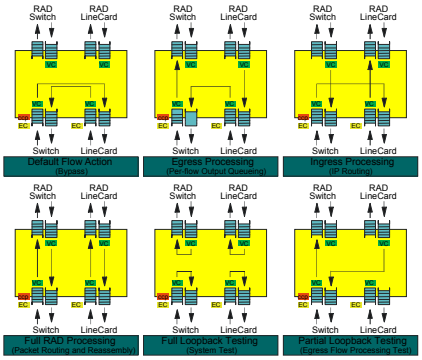

## Routing Traffic Flows

- Traffic flows routed among
  - Switch
  - Line Card
  - RAD.Switch
  - RAD.Linecard
- Functions
  - EC Check packets for errors
  - CCP Process commands
    - Control, status, & reprogramming
  - VC Implement per-flow forwarding





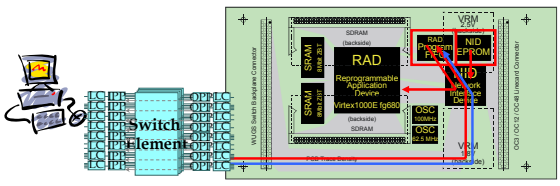

## Typical Flow Configurations






## Reprogramming Logic

- NID programs at boot from EPROM
- Switch Controller writes RAD configuration memory to NID
  - Configuration file for RAD arrives transmitted over network via control cells
- Switch Controller issues {Full/Partial} reconfigure command
- NID reads RAD config memory to program RAD
  - Performs complete or partial reprogramming of RAD



## FPX Interfaces Provides

- Well defined Interface
  - Utopia-like 32-bit fast data interface
  - Flow control allows back-pressure
- Flow Routing
  - Arbitrary permutations of packet flows through ports
- Dynamically Reprogrammable
  - Other modules continue to operate even while new module is being reprogrammed
- Memory Access
  - Shared access to SRAM and SDRAM
  - Request/Grant protocol

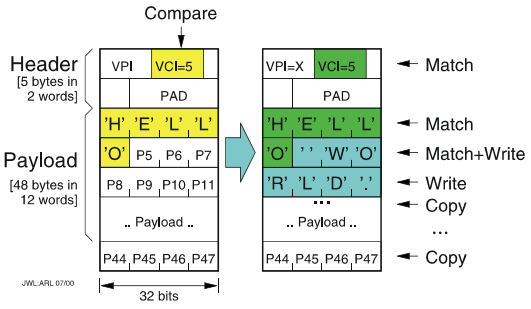


## Pattern Matching using the FPX

- Use Hardware to detect a pattern in data
- Modify packet based on match
- Pipeline operation to maximize throughput

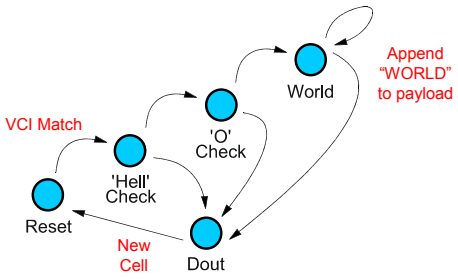


## "Hello, World" Module Function



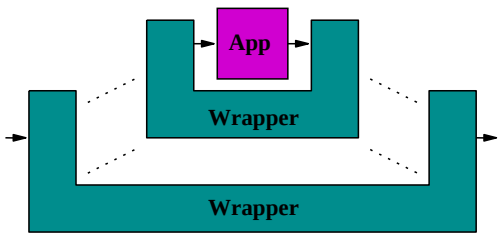



## Logical Implementation



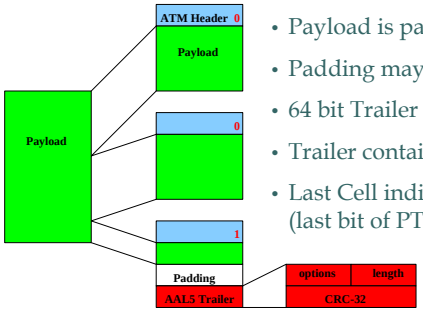



## The Wrapper Concept



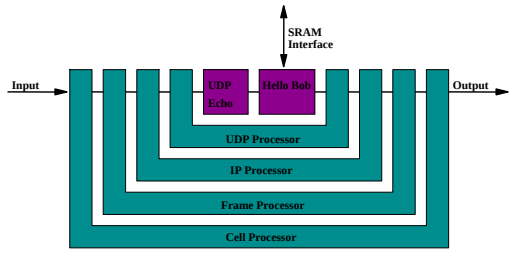



## AAL5 Encapsulation




- Payload is packed in cells
- Padding may be added
- 64 bit Trailer at end of cell
- Trailer contains CRC-32
- Last Cell indication bit (last bit of PTI field)

## HelloBob Module





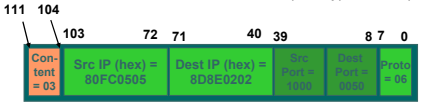

## Results: Performance

- Operating Frequency: 119 MHz.
  - 8.4ns critical path
    - Well within the 10ns period RAD's clock.
    - Targeted to RAD's V1000E-FG680-7
- Maximum packet processing rate:
  - 7.1 Million packets per second.
    - (100 MHz)/(14 Clocks/Cell)
    - Circuit handles back-to-back packets
- Slice utilization:
  - 0.4% (49/12,288 slices)
    - Less than one half of one percent of chip resources
- Search technique can be adapted for other types of data matching and modification
  - Regular expressions
  - Parsing image content …



## CAM-based Packet Matching

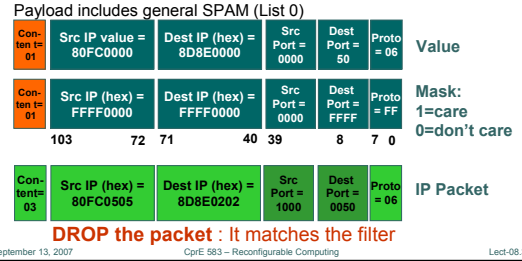
- Sample Packet:
  - Source Address = 128.252.5.5 *(dotted.decimal)*
  - Destination Address = 141.142.2.2 *(dotted.decimal)*
  - Source Port = 4096 *(decimal)*
  - Destination Port = 50 *(decimal)*
  - Protocol = TCP (6)
  - Payload = "Consolidate your loans. CALL NOW"
    - Payload Lists = { General SPAM (0), Save Money SPAM (1) }
    - Content Vector = "00000011" *(binary)* = x"03" *(hex)*






## Sample Filter

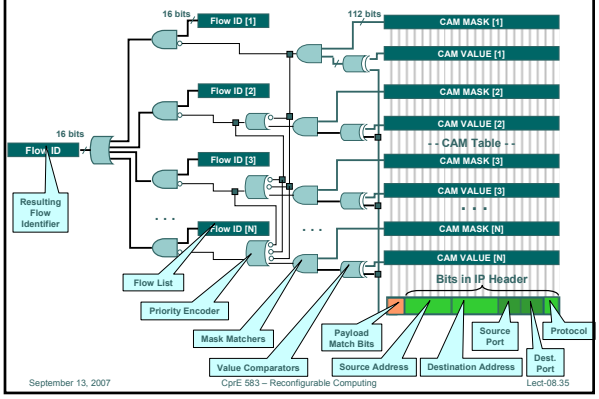
- Source Address = 128.252.0.0 / 16
- Destination Address = 141.142.0.0 / 16
- Source Port = Don't Care
- Destination Port = 50
- Protocol = TCP (6)
- Payload includes general SPAM (List 0)




**DROP the packet** : It matches the filter



## Packet Classifier with FlowID

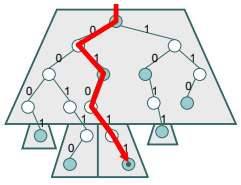




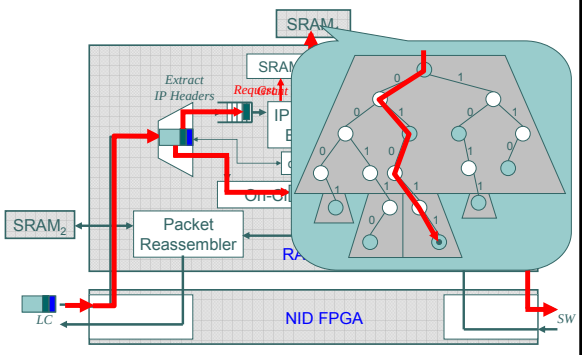

## Fast IP Lookup Algorithm

- Function
  - Search for best matching prefix using Trie algorithm

| Prefix | Next Hop |
|---|---|
| * | 4 |
| 01* | 7 |
| 10* | 2 |
| 110* | 9 |
| 0001* | 1 |
| 1011* | 0 |
| 00110* | 5 |
| 01011* | 3 |

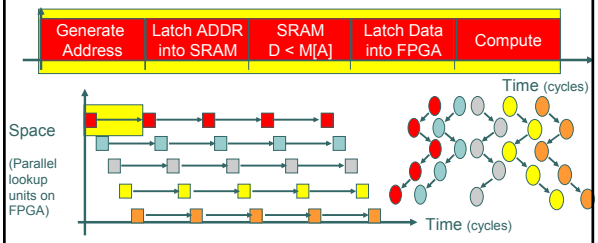


## Hardware Implementation in the FPX






## Pipelined FIPL Operations




- Throughput : Optimized by interleaving memory accesses
  - Operate 5 parallel lookups
  - t_pipelined_lookup = 550ns / 5 = 110 ns
  - Throughput = 9.1 Million packets / second



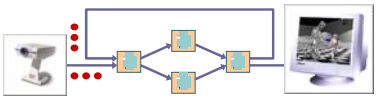
## Other Modules Implemented

- IPv6 Tunneling Module
  - Tunnels IPv6 over IPv4
- Statistics Module
  - Event counter
- Traffic Generator
  - Per-flow mixing
- Video Recoder
  - Motion JPEG
- Embedded Processor
  - KCPSM
- IPv4 CAM Filter
  - 104 Bit header matching
- Fast IP Lookup (FIPL)
  - Longest Prefix Match
  - MAE-West at 10M pkts/second
- Packet Content Scanner
  - Reg. Expression Search
- Data Queueing
  - Per-flow queue in SDRAM



## Summary

- Field Programmable Port Extender (FPX)
  - Network-accessible Hardware
  - Reprogrammable Application Device
- Module Deployment
  - Modules implement fast processing on data flow
  - Network allows Arbitrary Topologies of distributed systems



- Project Website
  - http://www.arl.wustl.edu/arl/projects/fpx/