DAT480 Reconfigurable Computing

Lecture 3:
Application development
for Reconfigurable Computing

Ioannis Sourdis

Outline of the Lecture:

- Applications development using RC
  - Strengths and Weaknesses
  - Application Characteristics
  - Implementation Strategies

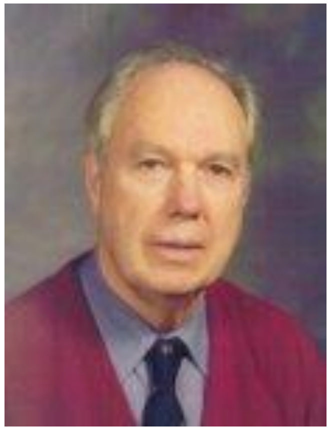
- Network Processing using RC
  - Pattern Matching
  - Regular Expressions matching

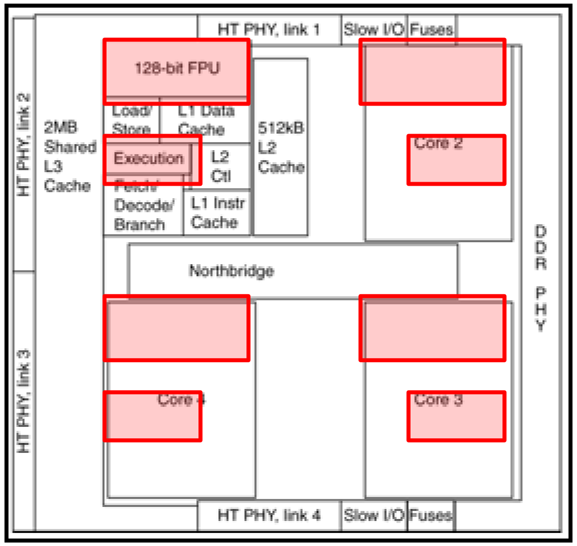
Michael Flynn’s talk @ TU Deft 2009: “Accelerating computations with FPGAs”

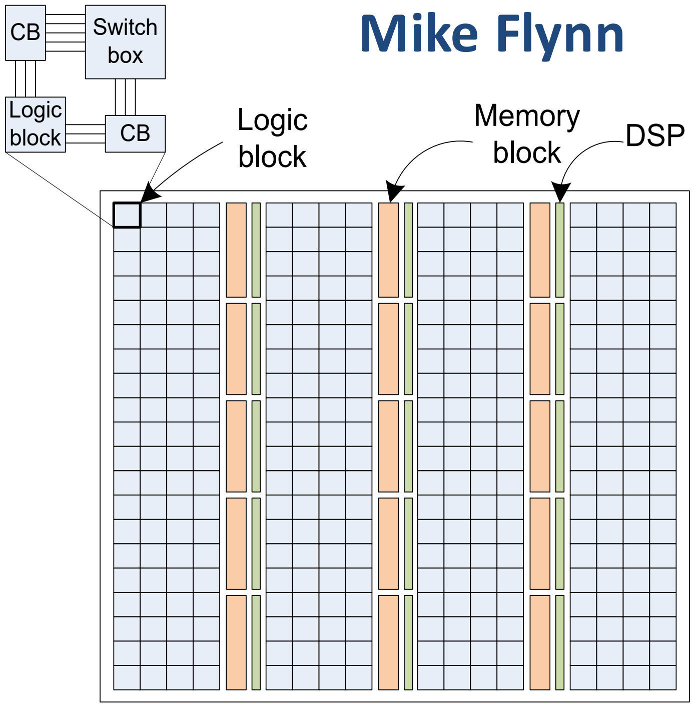
- Work of Michael Flynn and Oskar Mencer @ Maxeler Technologies.
  - Video and slides in Canvas

FPGA the "emulation" technology

- FPGAs have a disadvantage vs. ASICs: **ATP=10^2x-10^3 x**
- Still for some applications they get **10x** better perf./watt!


Mike Flynn





Strengths and Weaknesses

- **Time to Market**
  - **ASICs:** design, verification, fabrication, packadging, device test
  - **FPGAs:** design, verification, (~6 months shorter)
- **Cost (Low vs. High volume)**
  - For low-med volume FPGAs better than ASIC
- **Development Time**
  - HDL take longer than SW,
  - but there are high-level languages: i.e. C-to-HDL
- **Power Consumption**
  - More transistors than ASIC to implement a design
  - vs. Microprocessors ..depends on the case
- **Debug and Verification**
  - An FPGA prototype is faster but also tougher than simulations

Find the first set bit

000000000000000000000**1**0001010101011011111011

In software:

SHIFTs and BRANCHES

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

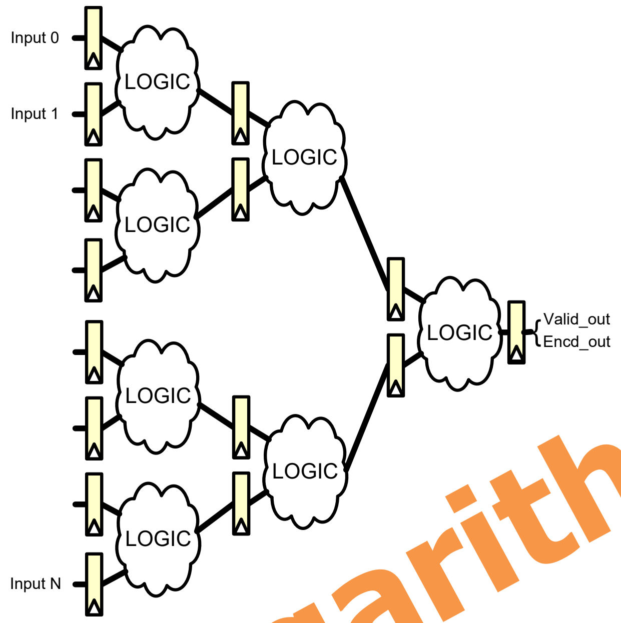
sll bge sll bge sll bge sll bge

sll bge sll bge sll bge sll bge

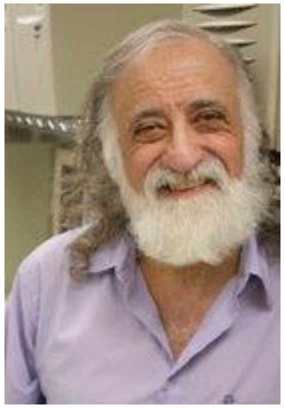
Linear

In hardware

Priority Encoder



Logarithmic


Yale Patt

Application Characteristics

Computational requirements

- Parallelism to compensate Clock Rate

Computational Characteristics:

- Data Parallelism
  - few or no data dependencies
- Data Element Size and Arithmetic Complexity
  - defines circuit complexity and speed
  - Small and low
- Pipelining
  - Can the Appl. tolerate the delay?
- Simple control
  - static vs. dynamic
  - Feed-only-forward vs. Feed-back loops

I/O requirements:

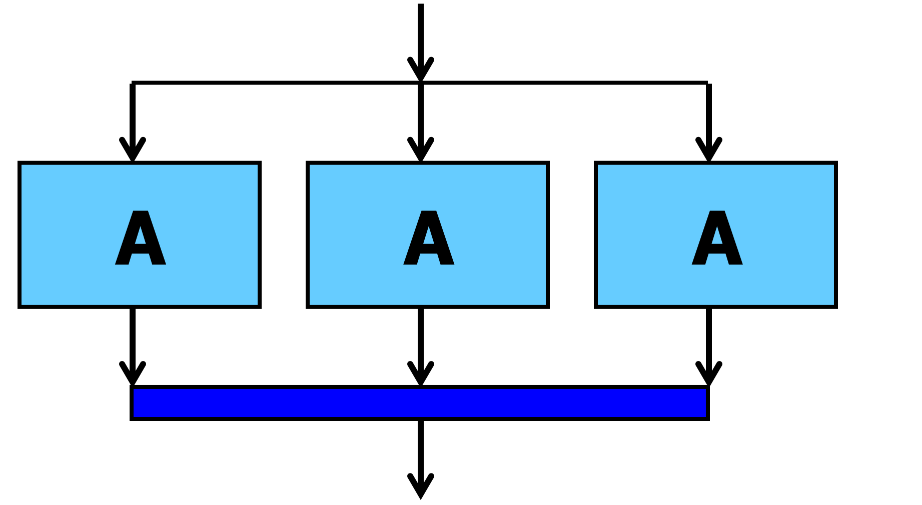
- feed your design with enough data
- I/O vs. Compute bound
- Memory elements to coordinate I/O and computation
  - BRAMs, LUTs, flip-flops
- Memory Size vs. Memory ports (bandwidth)

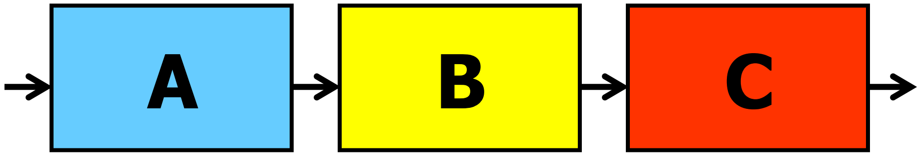
Reconfigurability:

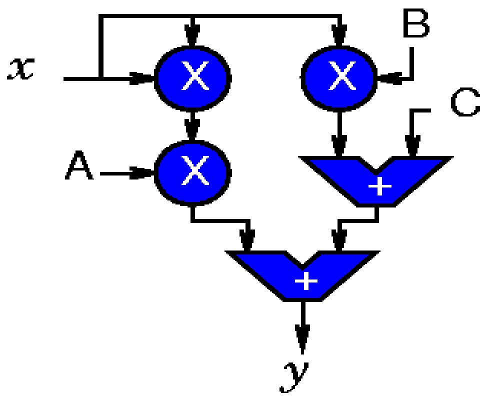
- Changing application requirements

Implementation Strategies

- Exploit **parallelism**, as much as:
  - the application specifications allow,
  - the FPGA resources allow
  - I/O bandwidth
- **Pipelining** (flip-flops are for free)
  - How much latency can the application tolerate?
- **Simple Control**
  - Streaming, Dataflow vs. Complex control



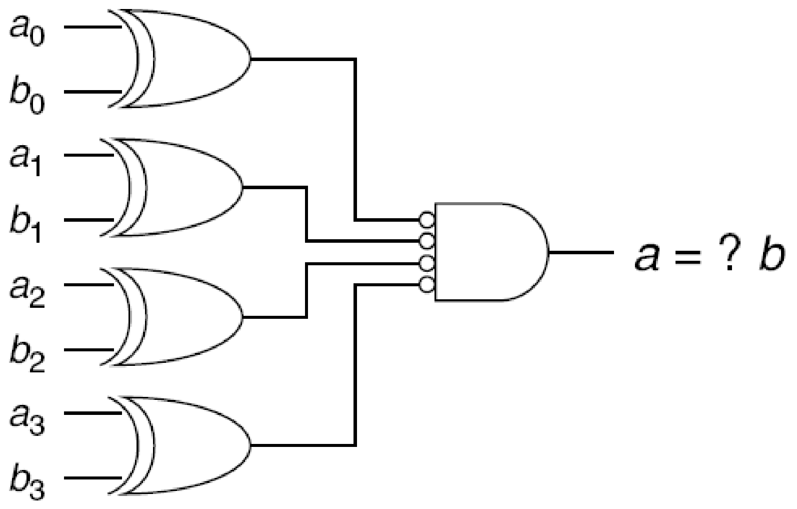




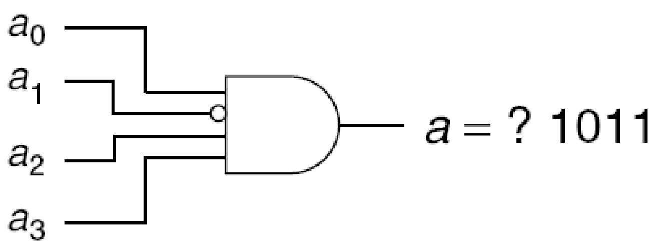
Implementation Strategies

- **Flexibility (to reconfigure):**
  - Adaptation to application requirements (Static or dynamic)
- **Customization:**
  - complete control over Arithmetic schemes, and Number representation
  - Constant folding and data-oriented specialization (e.g. per execution/program)

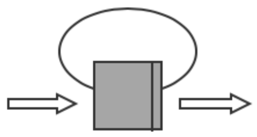
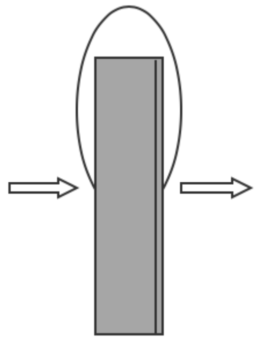
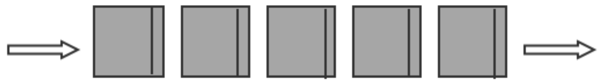
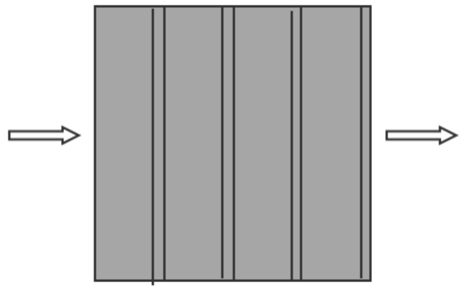
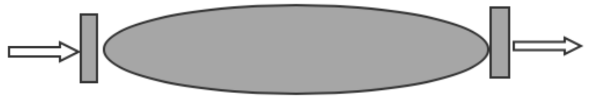
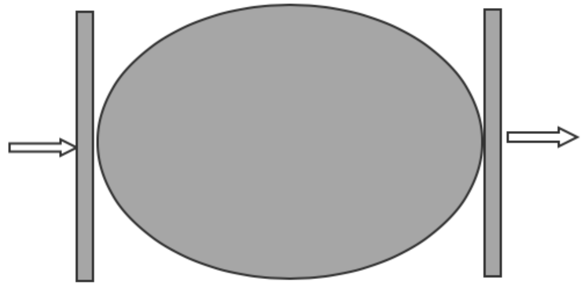
Without Constant folding



With Constant folding



Arithmetic: Implementation Styles

	Digit-Serial	**Digit-Parallel**
Sequential - loop x times		
Pipelined - loop unrolled		
Combinational - loop unrolled - no registers - logic min		

Implementation Strategies: **Reconfiguration**

- Reconfiguration: how often?
  - **Configure-once** (static, before execution)
    - or infrequently: every week, month or year
  - **Runtime Reconfiguration**, (dynamic, during execution)
    - **Global** (the entire device)
      - Distinct temporal application phases
    - **Partial** (part of the device)
    - Functional vs. temporal partitioning

- Can my design fit in the FPGA?
  - **If not:** can I break it in parts and swap them in and out?
- Reconfiguration overhead
  - Can I stall to reconfigure?
  - Can I prefetch my new design before RT reconfiguration?
    - Is there enough space and time

RunTime Reconfiguration example: Runtime Reconfigured Artificial Neural Network (RRANN)

Global RTR

- Back-propagation alg. (training of neural network): divided in 3 phases:
  - 3 exclusive configurations
  - Loaded one after the other
  - 5x better density eliminating idle circuits

Partial RTR

- Shared resources between the 3 phases
  - Only part of the device needs to change in each phase
  - Common circuit configured only once

Design Patterns for Reconfigurable Computing

- Design pattern: <u>a solution to a recurring problem</u>, **example:**
  - **Problem:** Design often too big to fit in the available HW
  - **Solution:** time multiplex the large design onto the limited HW,
  - <u>*Time multiplexing*</u> is a design pattern
- André DeHon et. al. **"Design Patterns for Reconfigurable Computing",** FCCM 2004
  - Organized design patterns in types/classes based on the problems they address

Coarse-grain time-multiplexing Design Pattern

- **When:** the fixed device capacity smaller than what the design requires
- **Can Apply,** when computational flow graph has:
  - no feedback loops (Acyclic dataflow graph), or
  - feedback loops can be contained within the capacity of the device, or
  - very slow feedback loops
- Reducing Reconfiguration overhead:
  - Long run lengths
  - Partial reconfiguration
  - Bitstream compression
  - Parallel on-chip memories for reconfiguration
  - Prefetching
- Implementation:
  - Task partitioning (manually or automated)
  - **control algorithm** coordinates the reconfiguration to instantiate the various blocks (static or dynamic shceduling).

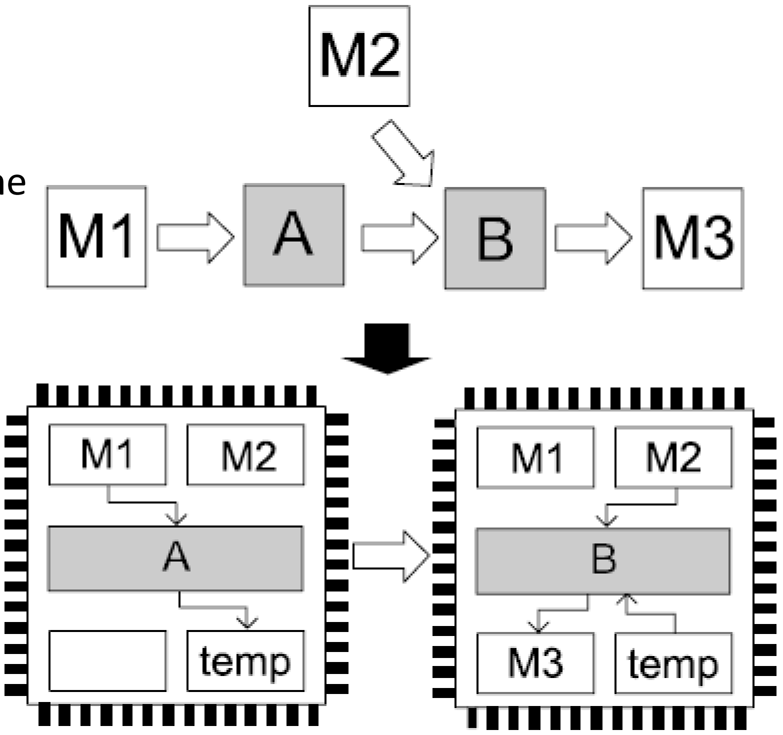
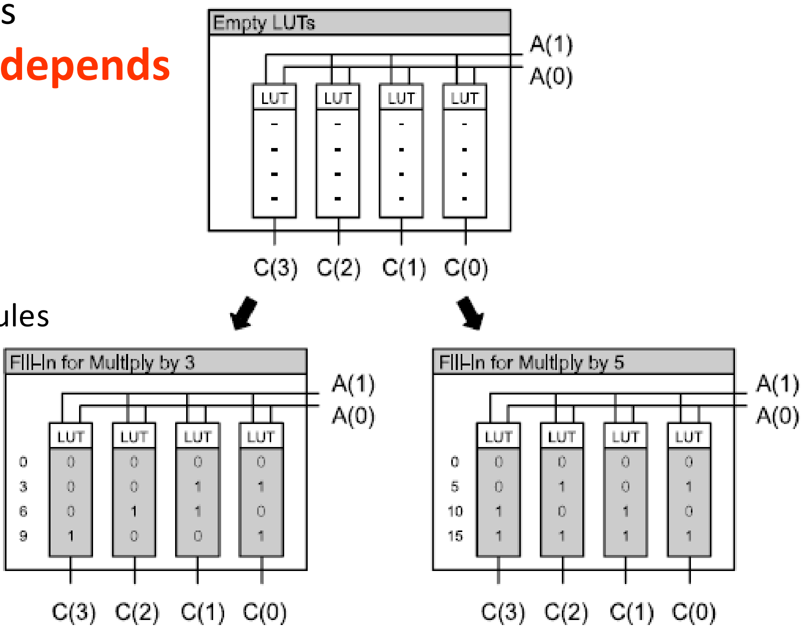


Figure 1: Time Multiplexing Pattern

André DeHon **"Design Patterns for Reconfigurable Computing",** FCCM 2004

Template Specialization Design Pattern

- **Specialization pattern:**
  - Reduce space/time for a computation
  - Early-bound data can be folded in the implementation
- **Template specialization:**
  - minimize (re)generation time
  - e.g. Keep interconnects fixed and change only LUT contents
- **Tradeoffs:** template specialization vs. Full specialization **(depends on the case)**:
  - Faster to change
  - Less flexibility (i.e. Fixed interconnect)
  - more resources in order to be generic.
    - Sometime a template is a superset of some fully specialized modules
- **Examples:**
  - Multiply by Constant
  - String Matching/ Content Addressable Memory
- **Implementation:**
  - LUTs with a direct write
    - **XC6000:** a controller can write to config. Memory
    - **Virtex**: change bitstream or use SRL16

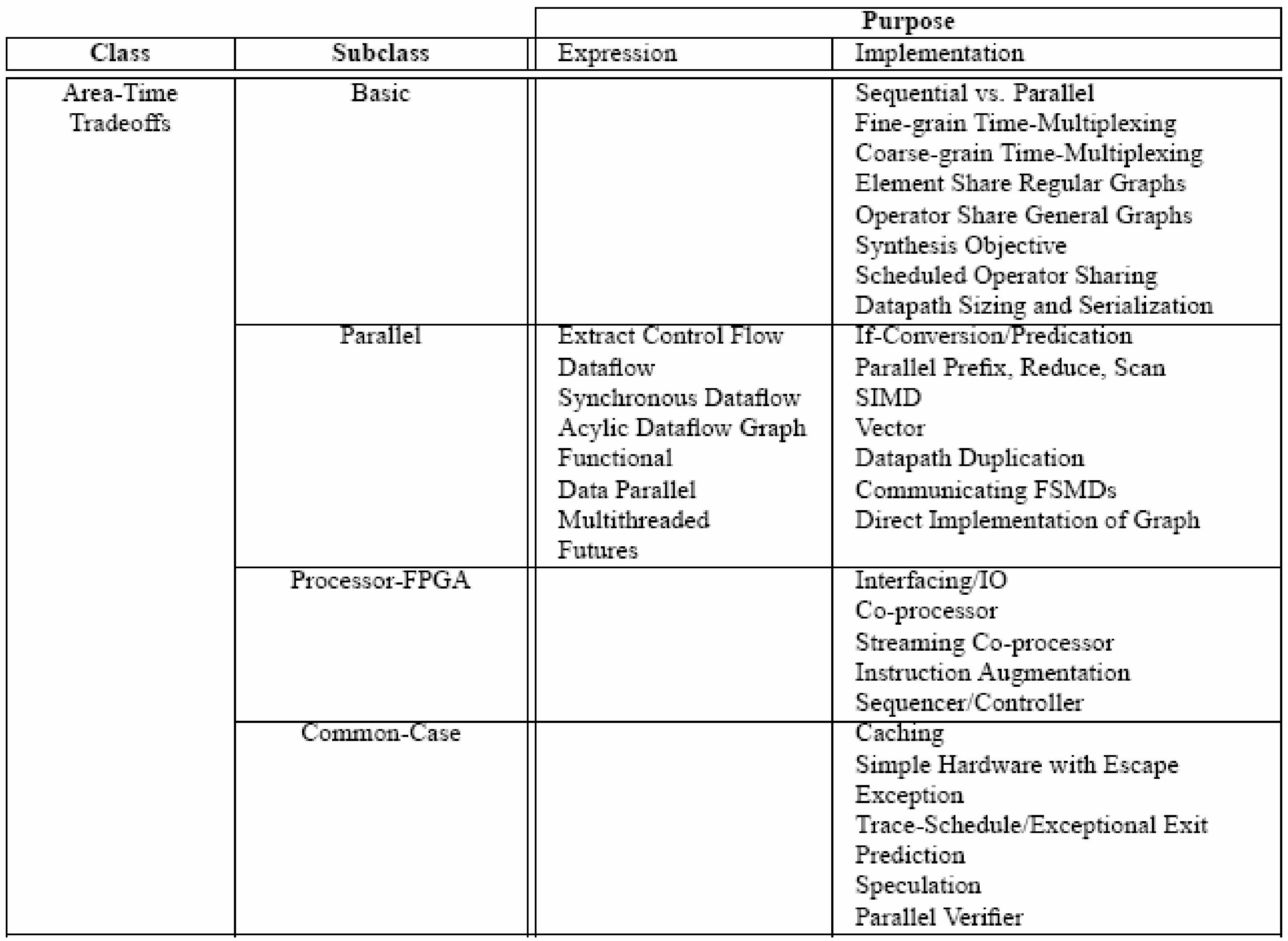


(Shown with 2-LUTs for simplicity)

Figure 2: Template Pattern

André DeHon **"Design Patterns for Reconfigurable Computing"**, FCCM 2004

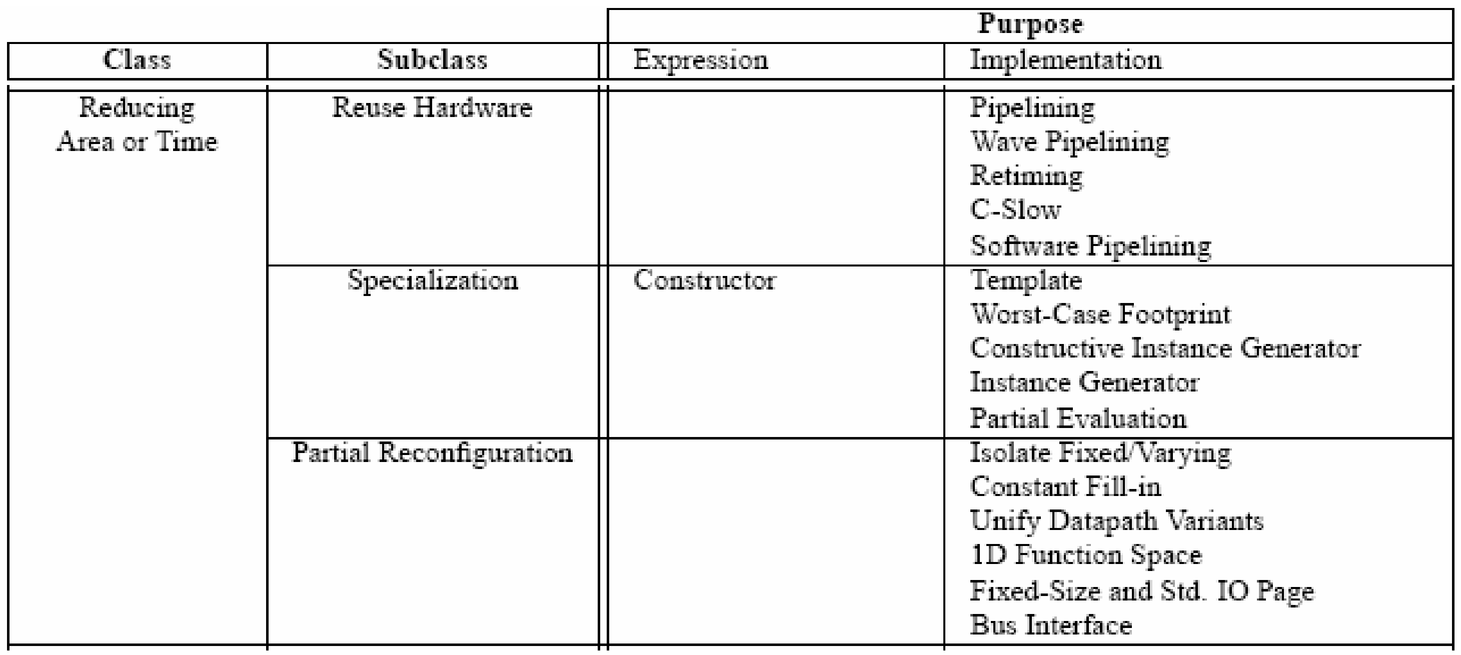
Some Design Patterns for Reconf. Computing



		Purpose	
Class	Subclass	Expression	Implementation
Area-Time Tradeoffs	Basic		Sequential vs. Parallel Fine-grain Time-Multiplexing Coarse-grain Time-Multiplexing Element Share Regular Graphs Operator Share General Graphs Synthesis Objective Scheduled Operator Sharing Datapath Sizing and Serialization
	Parallel	Extract Control Flow Dataflow Synchronous Dataflow Acylic Dataflow Graph Functional Data Parallel Multithreaded Futures	If-Conversion/Predication Parallel Prefix, Reduce, Scan SIMD Vector Datapath Duplication Communicating FSMDs Direct Implementation of Graph
	Processor-FPGA		Interfacing/IO Co-processor Streaming Co-processor Instruction Augmentation Sequencer/Controller
	Common-Case		Caching Simple Hardware with Escape Exception Trace-Schedule/Exceptional Exit Prediction Speculation Parallel Verifier

André DeHon **"Design Patterns for Reconfigurable Computing",** FCCM 2004

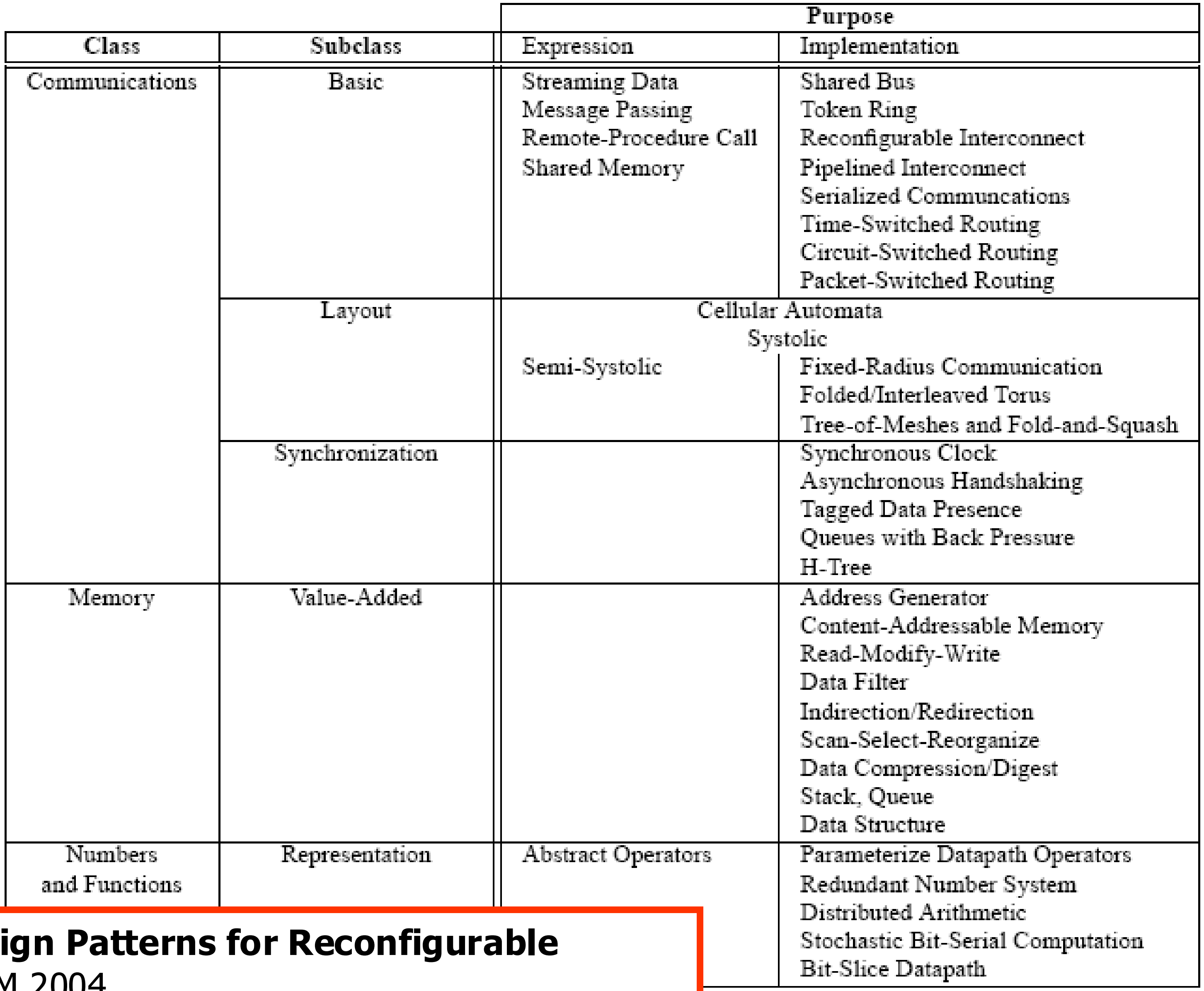
Some Design Patterns for Reconf. Computing



		Purpose	
Class	Subclass	Expression	Implementation
Reducing Area or Time	Reuse Hardware		Pipelining Wave Pipelining Retiming C-Slow Software Pipelining
	Specialization	Constructor	Template Worst-Case Footprint Constructive Instance Generator Instance Generator Partial Evaluation
	Partial Reconfiguration		Isolate Fixed/Varying Constant Fill-in Unify Datapath Variants 1D Function Space Fixed-Size and Std. IO Page Bus Interface

André DeHon **"Design Patterns for Reconfigurable Computing",** FCCM 2004

Some Design Patterns for Reconf. Computing



Class	Subclass	Purpose: Expression	Purpose: Implementation
Communications	Basic	Streaming Data Message Passing Remote-Procedure Call Shared Memory	Shared Bus Token Ring Reconfigurable Interconnect Pipelined Interconnect Serialized Communcations Time-Switched Routing Circuit-Switched Routing Packet-Switched Routing
	Layout	Cellular Automata	
		Systolic	
		Semi-Systolic	Fixed-Radius Communication Folded/Interleaved Torus Tree-of-Meshes and Fold-and-Squash
	Synchronization		Synchronous Clock Asynchronous Handshaking Tagged Data Presence Queues with Back Pressure H-Tree
Memory	Value-Added		Address Generator Content-Addressable Memory Read-Modify-Write Data Filter Indirection/Redirection Scan-Select-Reorganize Data Compression/Digest Stack, Queue Data Structure
Numbers and Functions	Representation	Abstract Operators	Parameterize Datapath Operators Redundant Number System Distributed Arithmetic Stochastic Bit-Serial Computation Bit-Slice Datapath

André DeHon **"Design Patterns for Reconfigurable Computing"**, FCCM 2004

Application Use Cases: **Network Processing using Reconfigurable HW**

Network Processing in Reconfigurable HW

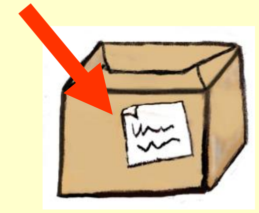
- Deep Packet Inspection
  - Check the packet header
  - Check the packet content

  “understand” the content of each packet

- Network Routing
  - Check Dest IP address to find route

High-Speed Network Processing

- Analyze traffic:
  - **Packet Header**



  - **Packet Content**

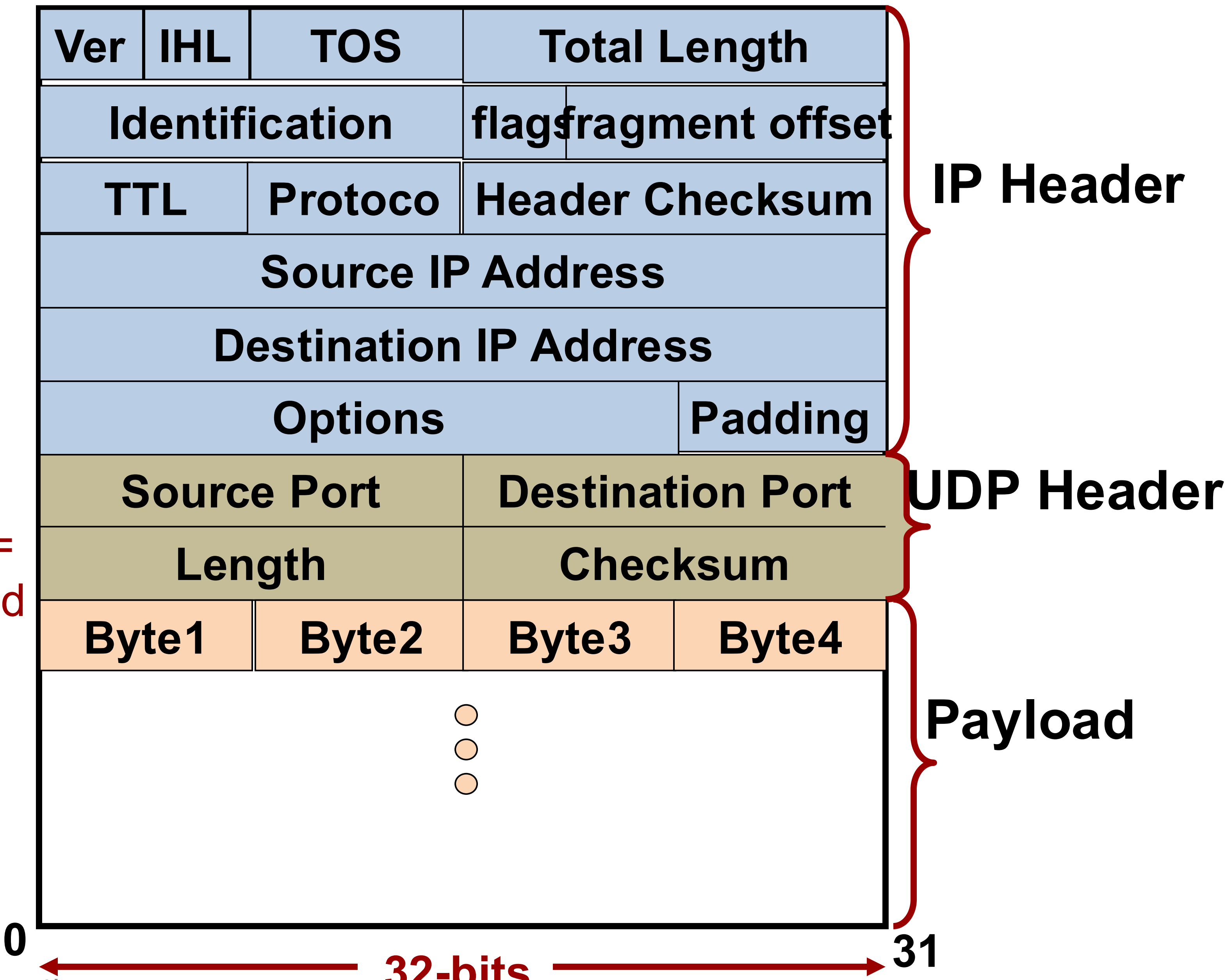
- Performance requirement:
  - **10’s of Gigabits/sec**
  - **e.g. 100 Gbps**

UDP/IP Packet Format

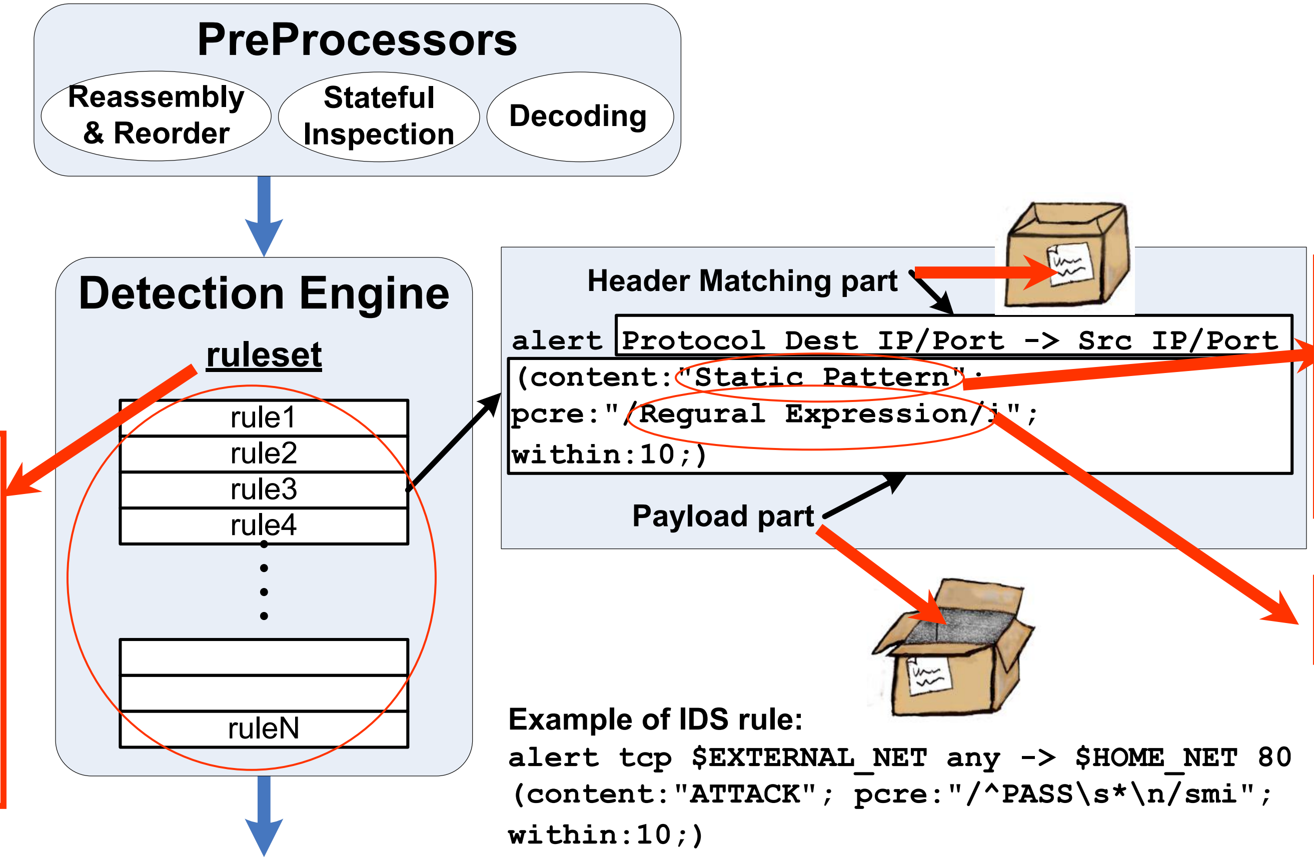
Note: flags 3 bits

UDP Protocol = 17

UDP length (bytes) = UDP header+payload



DPI for Network Intrusion Detection



10k rules to be matched in parallel against each incoming packet

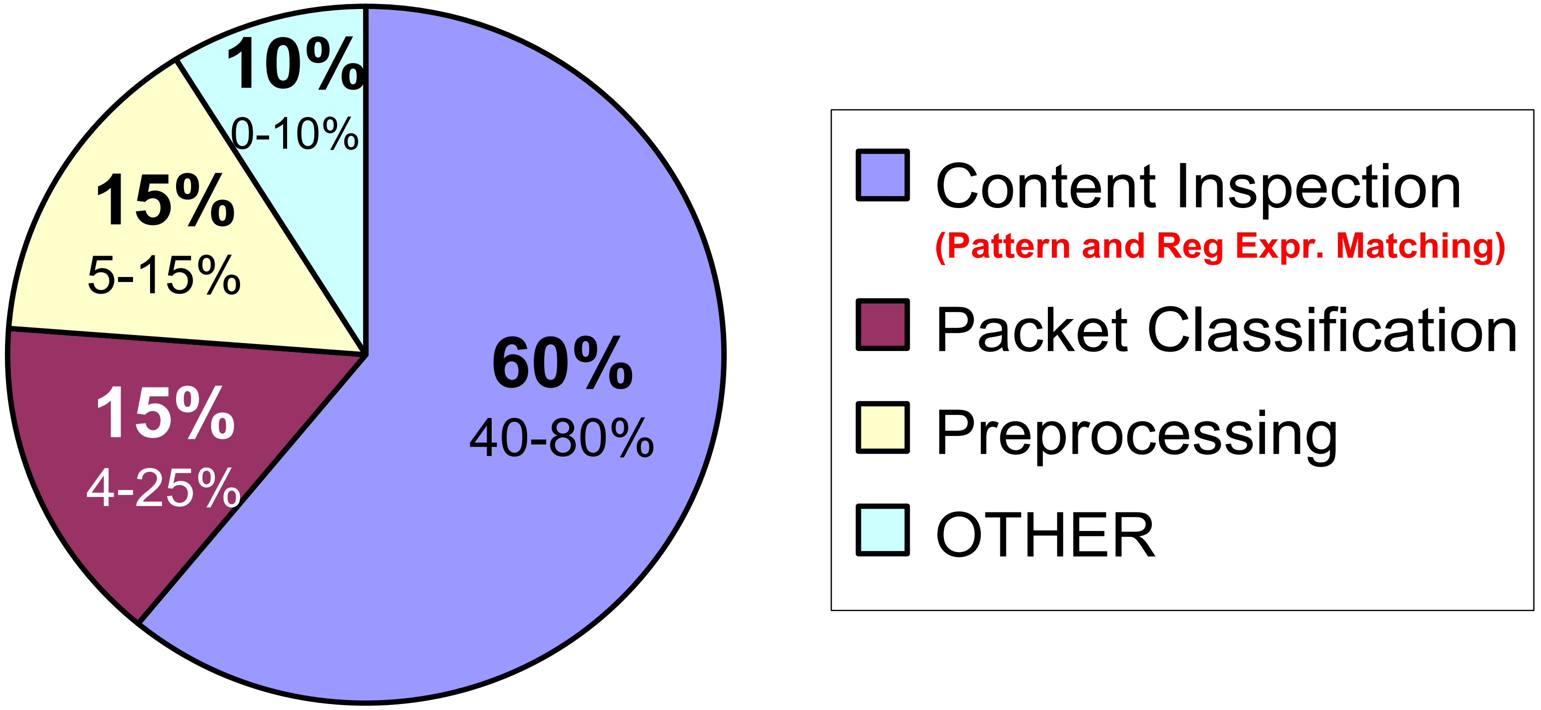
3-4k patterns 10 characters each on average

1.5k RegExpr

Example of IDS rule:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80
(content:"ATTACK"; pcre:"/^PASS\s*\n/smi";
within:10;)
```

Intrusion Detection Systems Profiling



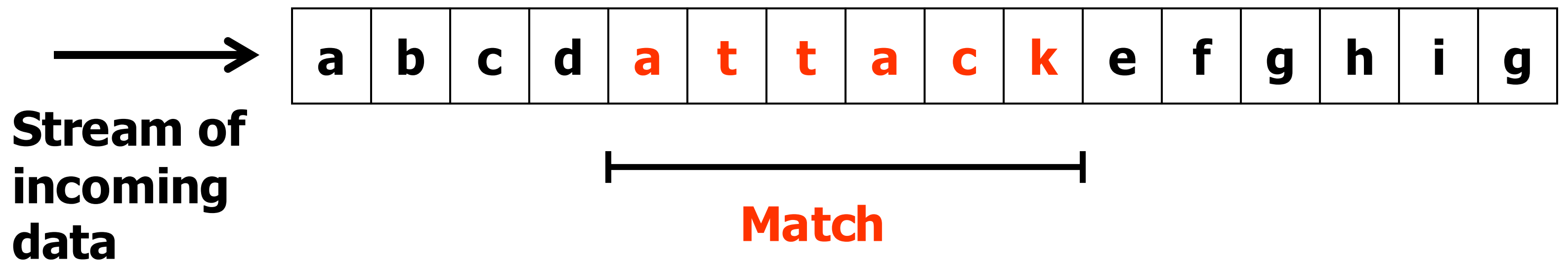
DPI Design Alternatives

Alternatives:

- **Software:**
  - Very Flexible
  - Very slow (a few tens or Hundreds **Mbps**)
- **ASIC**
  - Too expensive (NRE cost)
  - NOT Flexible
  - Very Fast (**10+ Gbps**)
  - **Memory-based design** to gain flexibility -> limits design decisions
- **FPGAs/Reconfigurable HW**
  - Flexible (can reconfigure)
  - Relatively fast (**2-10 Gbps**)

Content Inspection:

Pattern Matching and Regular Expressions



Set of search Patterns:

P1: attack
P2: virus
P3: intrusion
P4: get pwd

Set of search Reg Exprs:

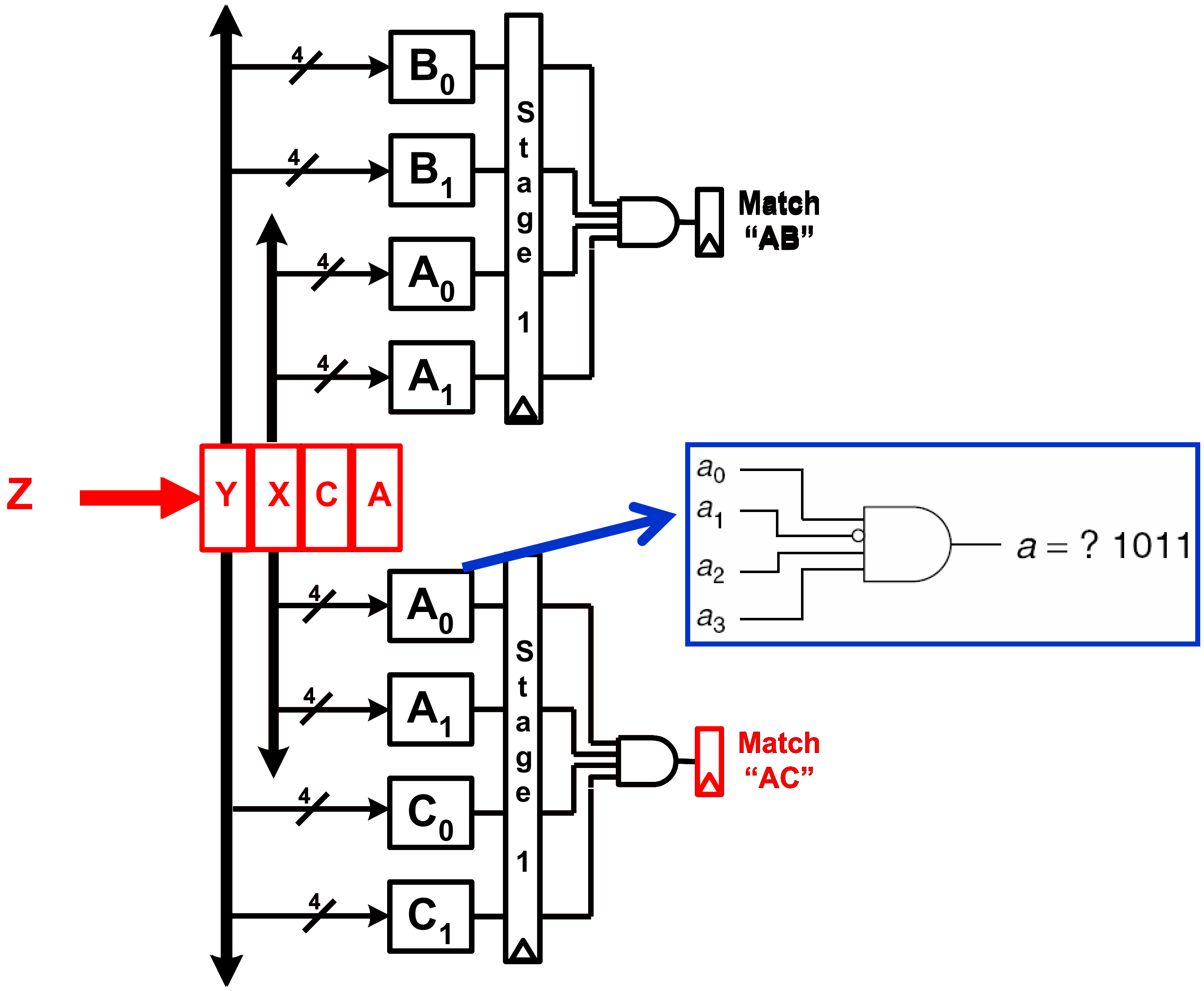
R1: at{2}ack
R2: get(^\n){4,}

DPI Content Inspection: Characteristics of the Application

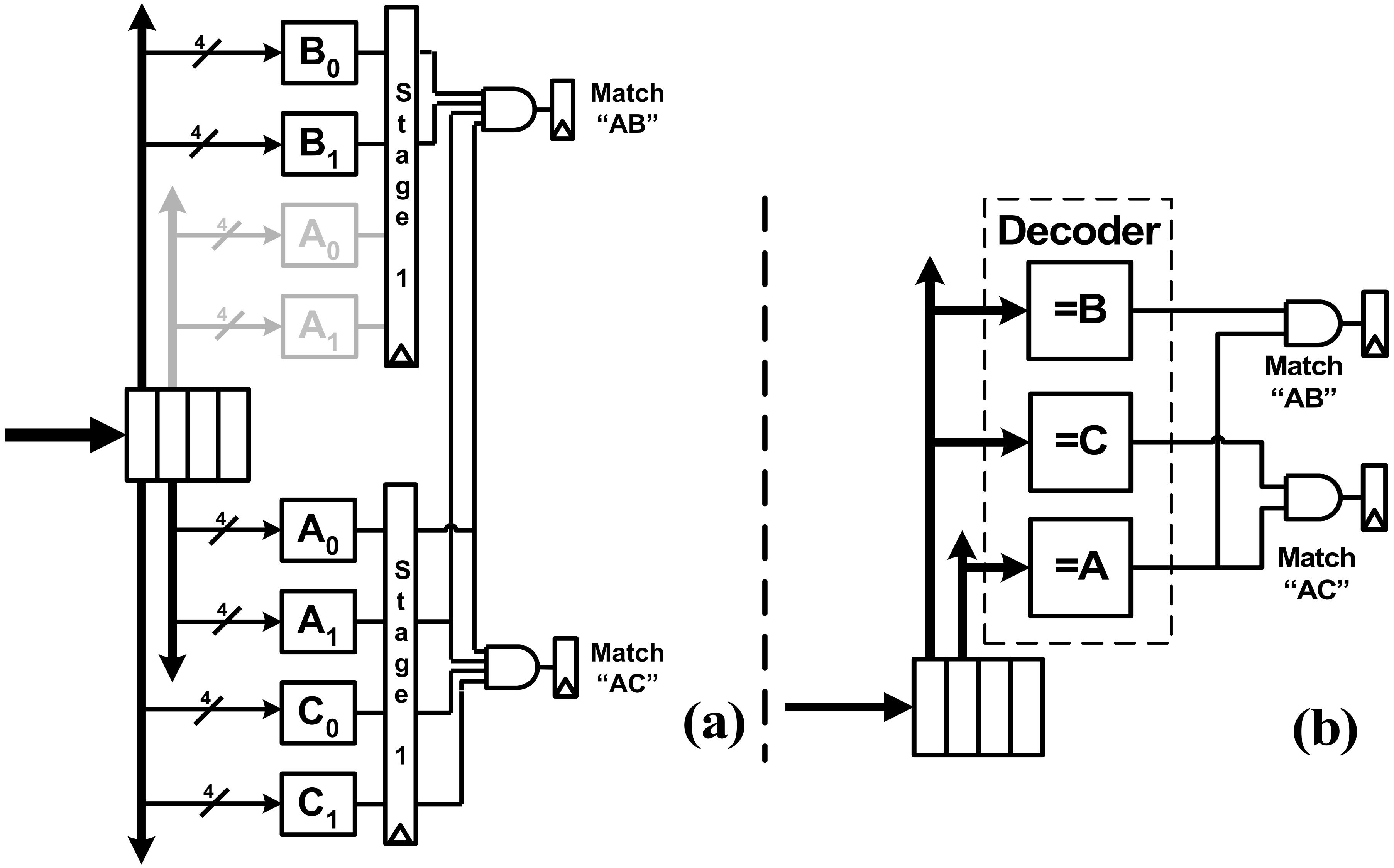
- **Performance Requirements:**
  - multiple Gbps throughput
- **Parallelism:**
  - 1,000's of contents to be matched in parallel against a stream of data
- **Data element Size**:
  - 1-char=1 byte
- **Computation complexity:**
  - bit-wise comparisons ++
- **Control:**
  - Simple: start/stop or bop/eop, it's a streaming application
- **Pipelining:**
  - as fine-grain as possible (can tolerate latency upto ~100's of nsec)
- **I/O:**
  - stream of data at Gbps rates
- **Reconfigurability:**
  - ruleset changes every few weeks: **configure-once strategy**

Perfect candidate for Reconfigurable HW

Pattern Matching

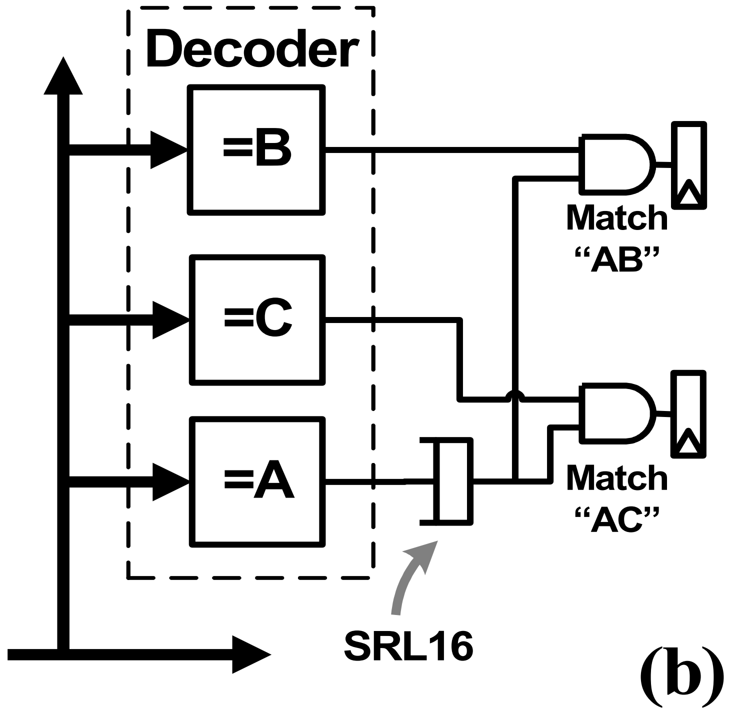


Optimization of Basic CAM Structure

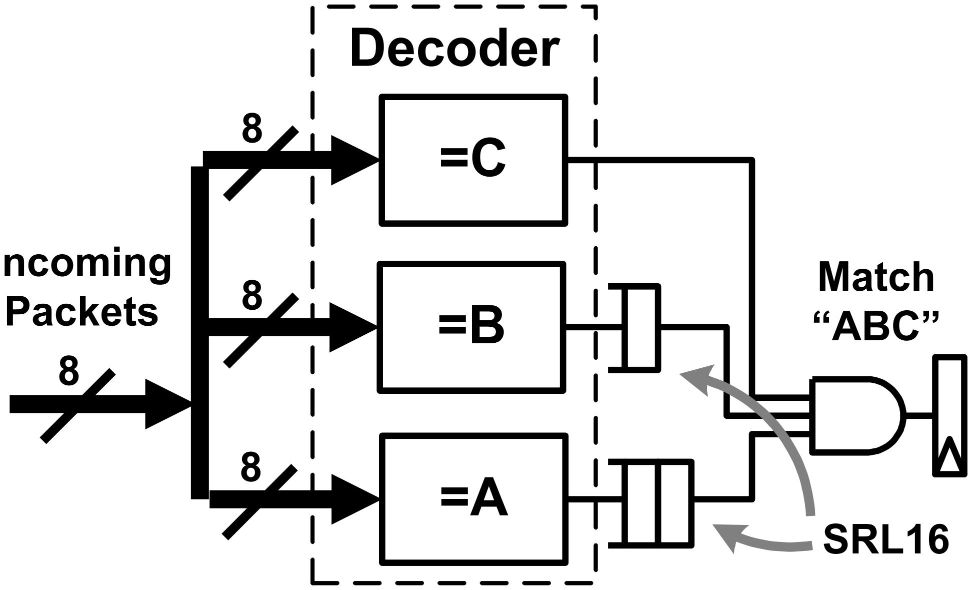


Optimized CAM =>DCAM

- Centralized decoders for incoming data comparison
- Previous Shifter: 8 bit
- Here, 1 bit
- Proper shifting of decoded data using Xilinx SRL16 cell
  - Share the outputs of shift registers
  - SRL16 can fit in one Logic Cell, and has width up to 16

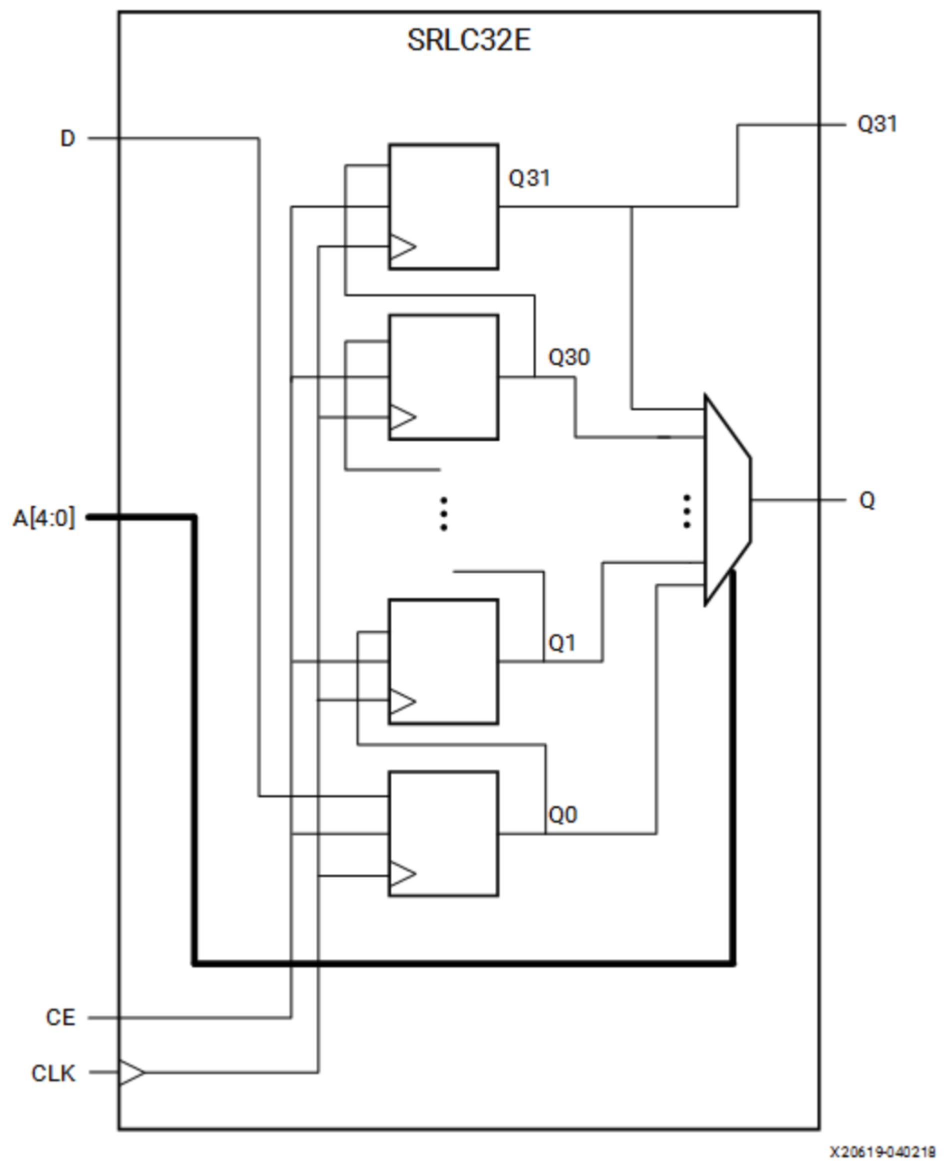


DCAM Structure

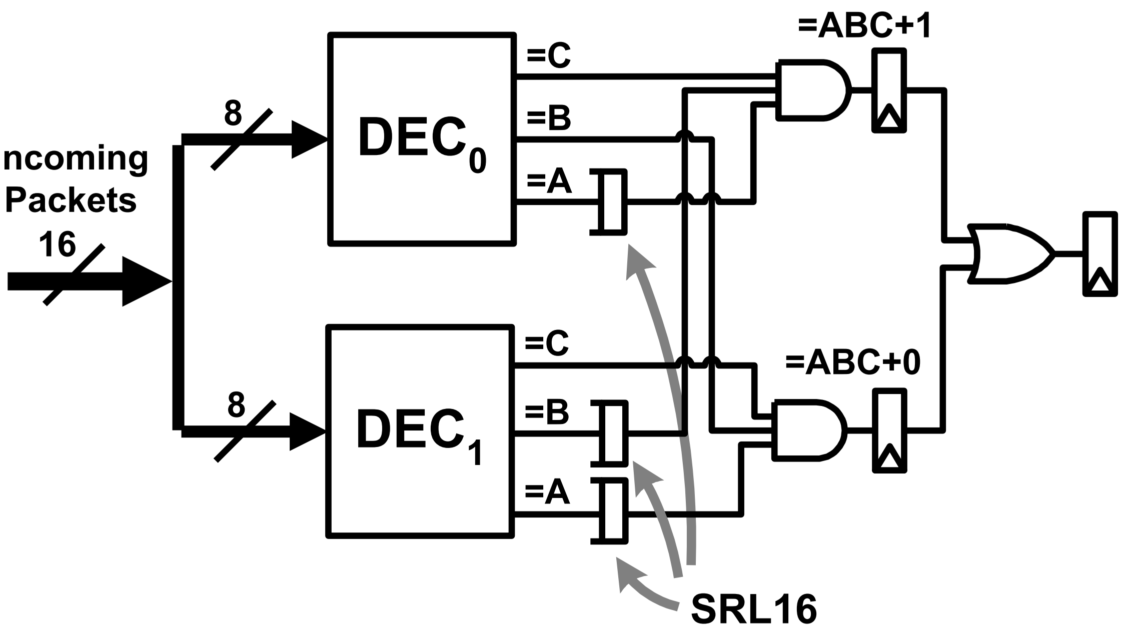


SRL

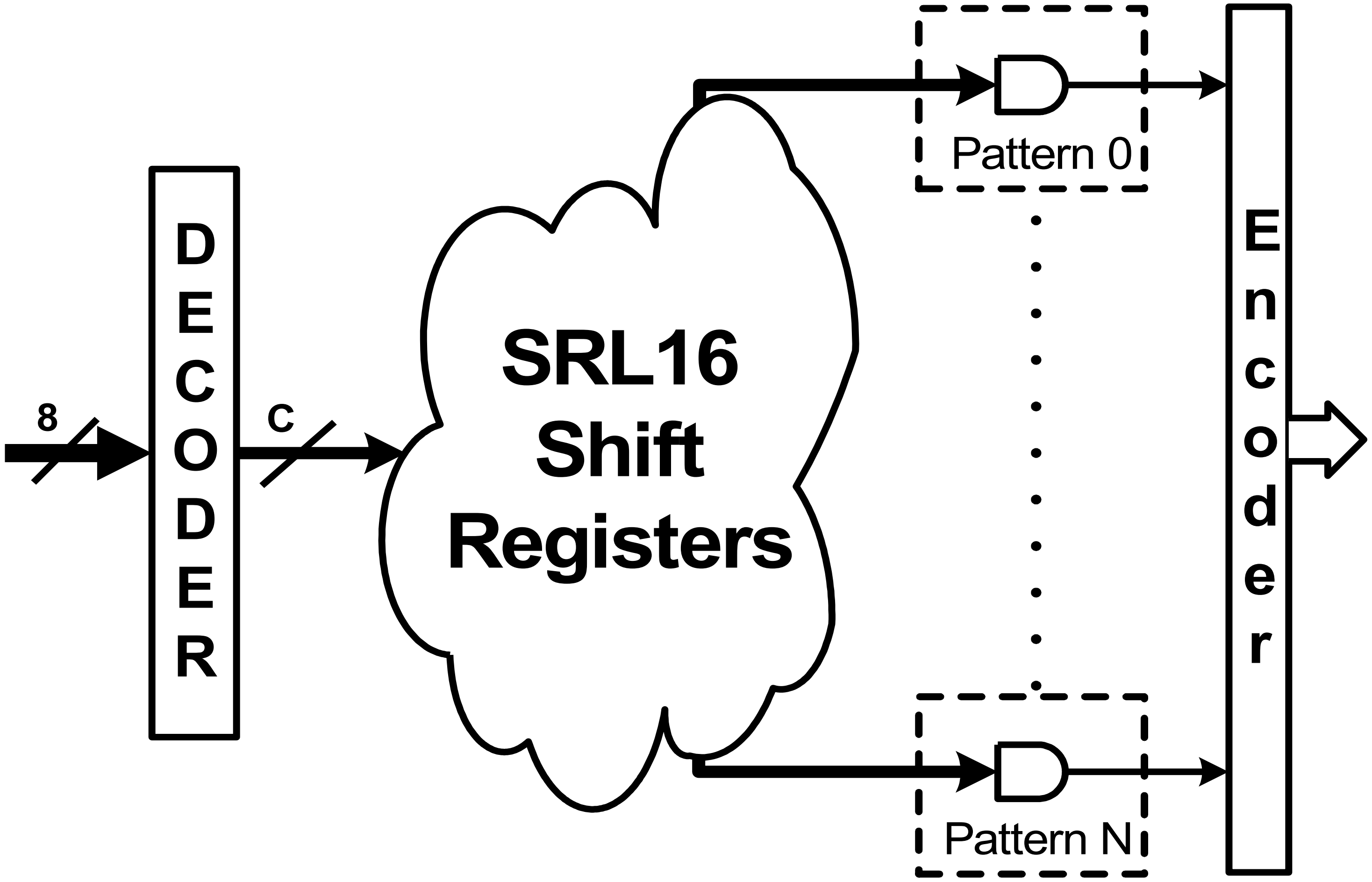
Figure: SRL Shift Register Primitive



DCAM Structure with Parallelism

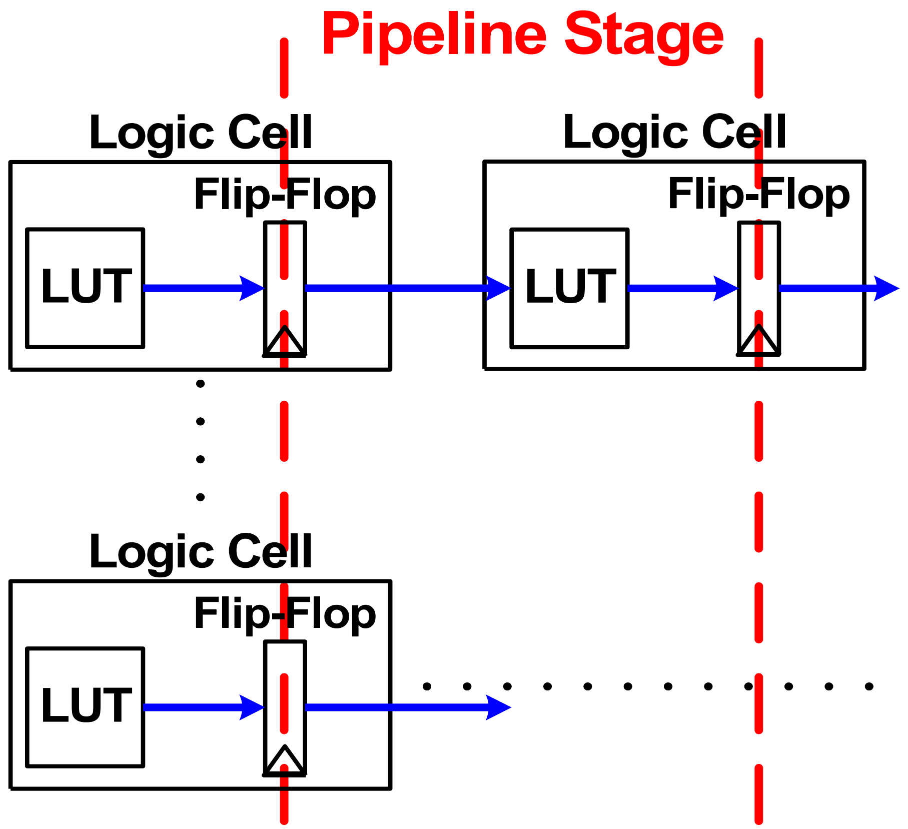


Top-Level DCAM



DCAM: Pipelining and Parallerlism

- **Fine-grain Pipeline**: High frequency
  - Each pipeline stage fits in 1-level 4-input LUTs and their corresponding registers
  - Cycle time = 1 * LogicCellDelay + WireDelay
- **Parallelism**
  - Process *N* incoming packet bytes per c.c.
  - *N* parallel comparators for every pattern.



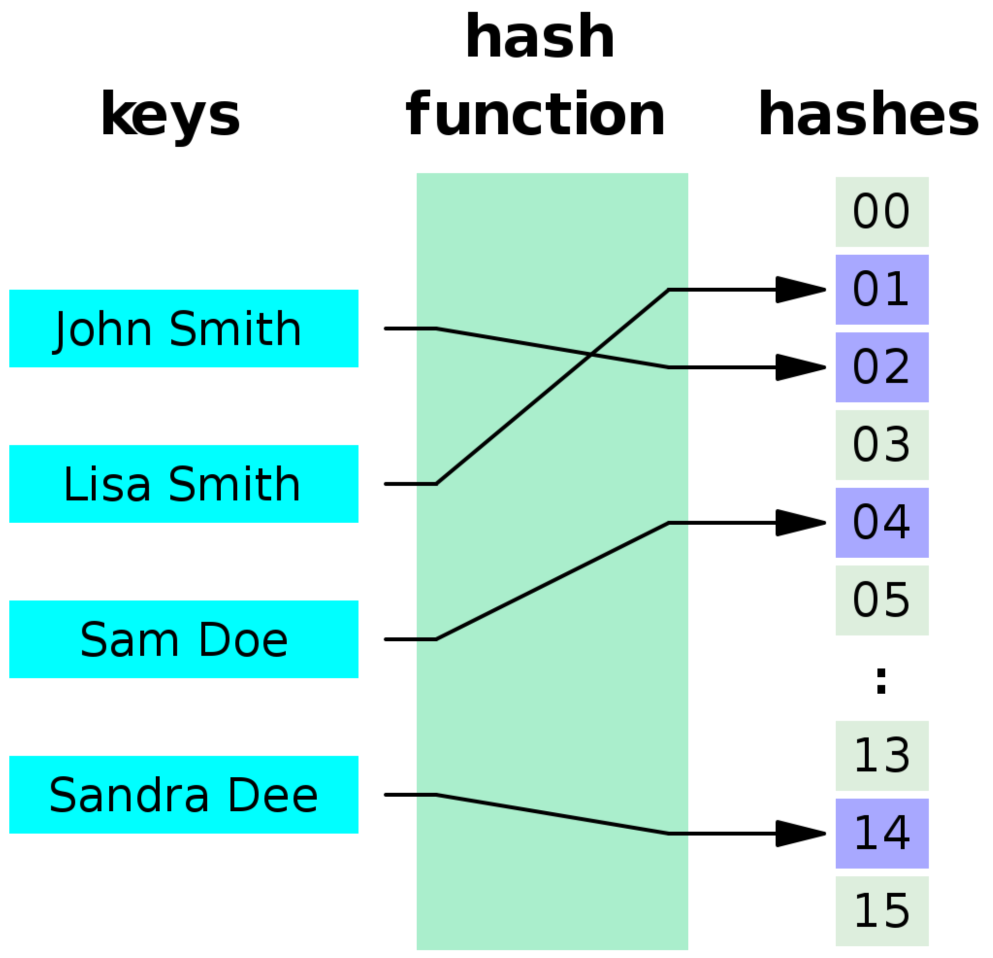
DCAM Results

- 2.5-10 Gbps processing throughput (Virtex2 technology)
- 0.7-2.7 logic cells per matching character
  - Today FPGAs have upto 700k LCs …

Software: a few 100's Mbps throughput

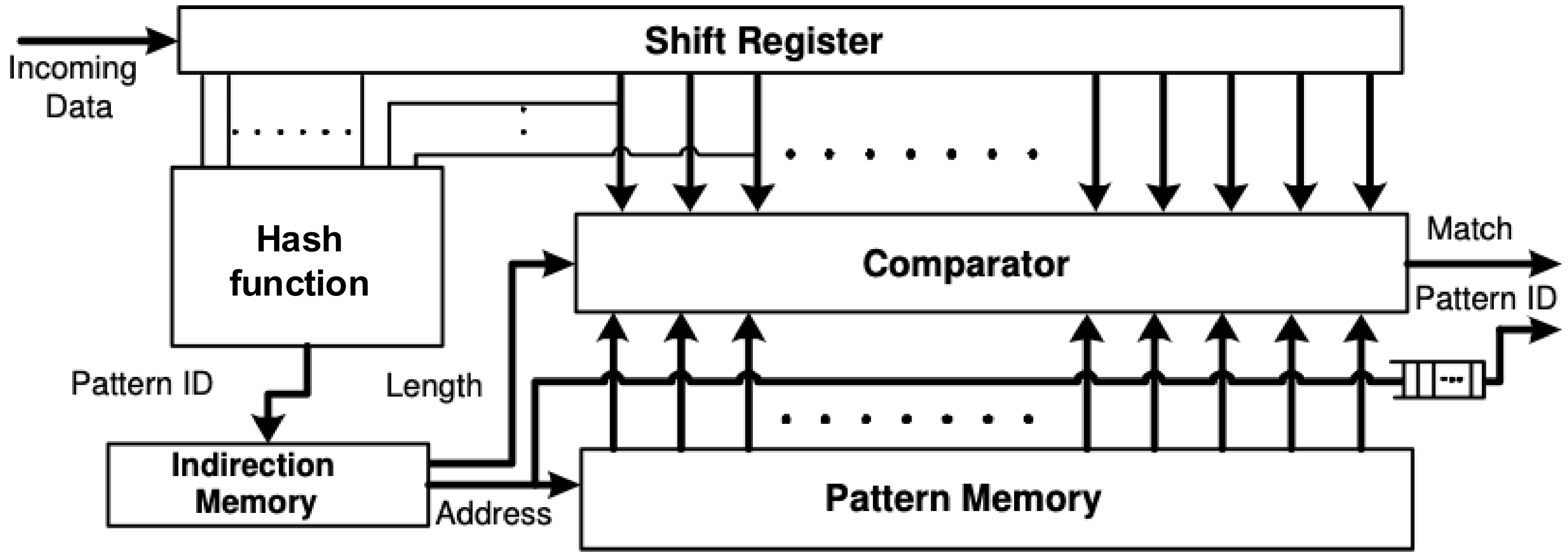
Pattern matching with hashing

- Choose a hash function that gives a different output per search pattern



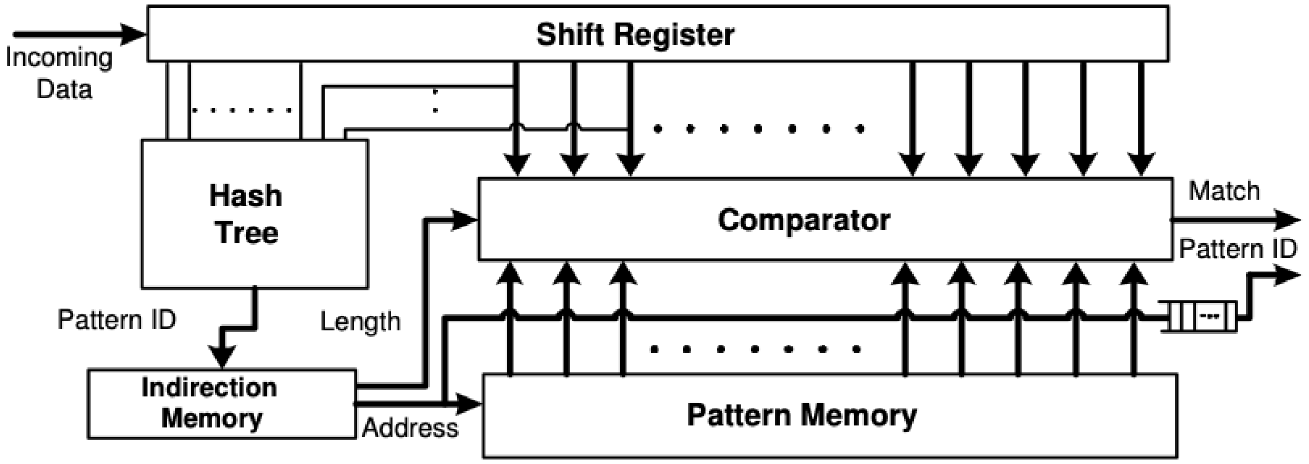
Pattern matching with hashing

- Hash incoming traffic and use the hash result as index to find the potential matching pattern
- Compare incoming traffic with the potential matching pattern



Pattern matching with hashing

- Challenges
  - Hard to find a perfect hash function
    - CRC, Bloom filters, Sourdis et al.
  - Different length of patterns
  - Complex to process more than one byte (character)
    - 1 byte/cycle => 2.5Gbps (@250MHz)
    - 4 bytes/cycle => 10Gbps
    - 100 Gbps needs 40 bytes

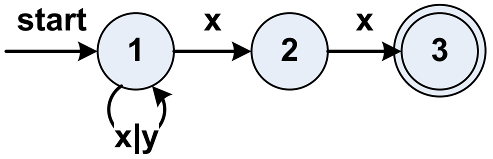


Regular Expression Matching

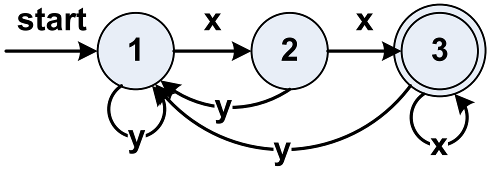
NFAs vs. DFAs

- **NFAs:** Non-Deterministic Finite Automata
  - Multiple active states at a time
  - Very bad performance in SW
- **DFAs:** Deterministic Finite Automata
  - Only one active state at a time
  - Good perormance in SW
  - State explosion
  - High memory requirements

NFA

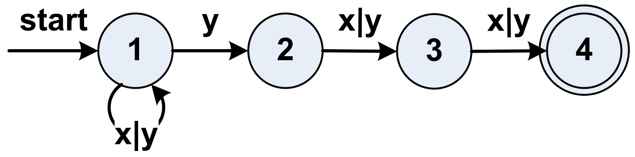


DFA

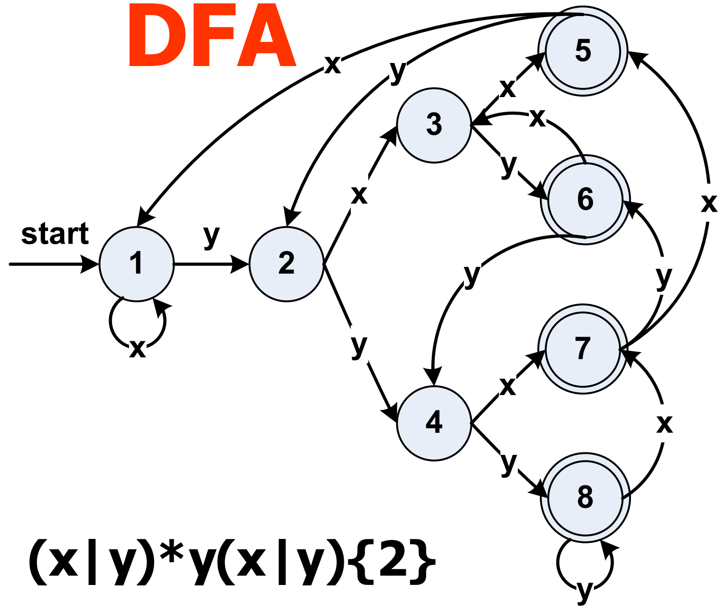


(x|y)*x{2}

NFA

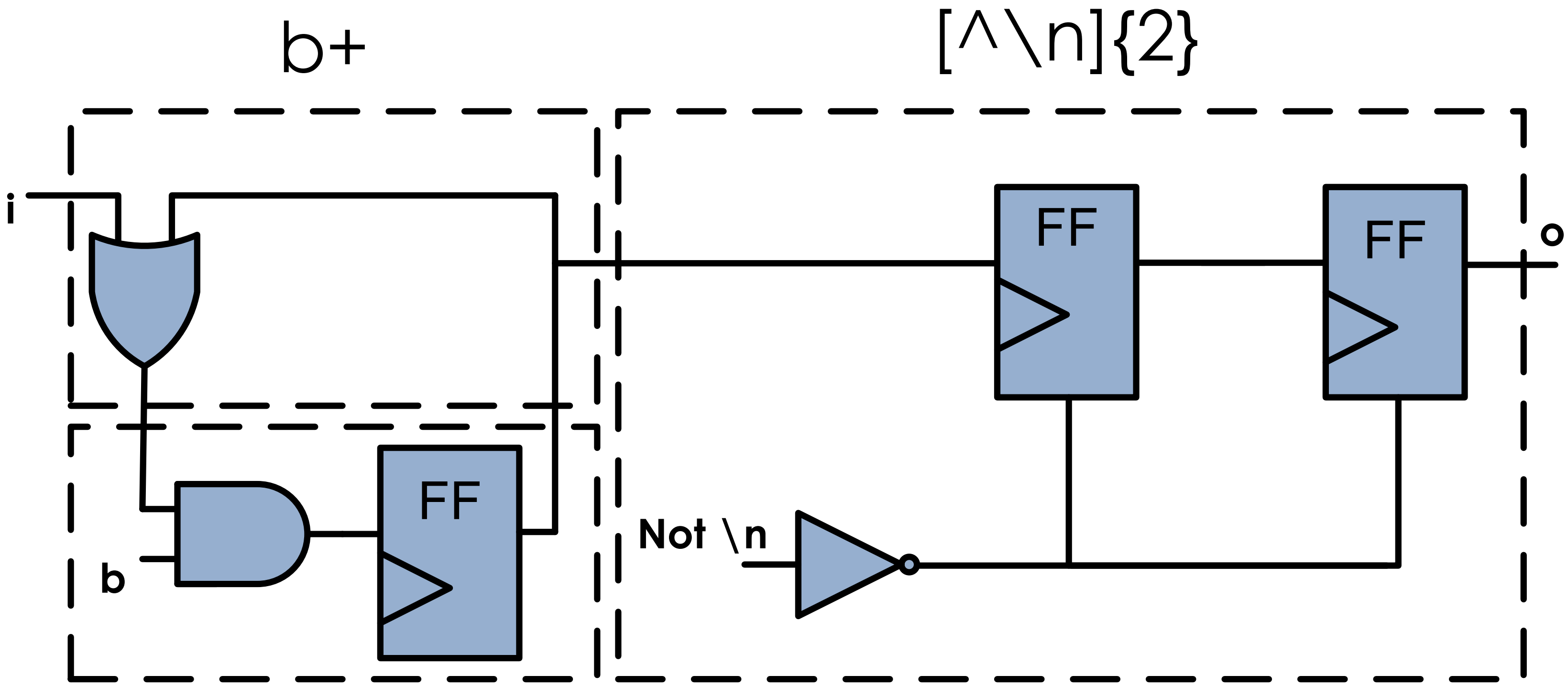


DFA



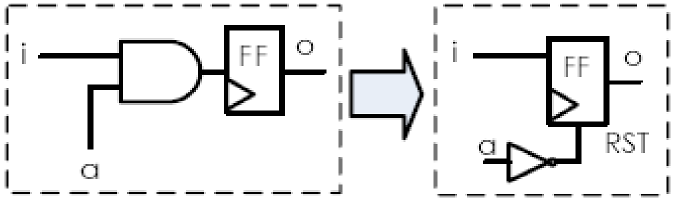
(x|y)*y(x|y){2}

NFA Reconfigurable HW implementation

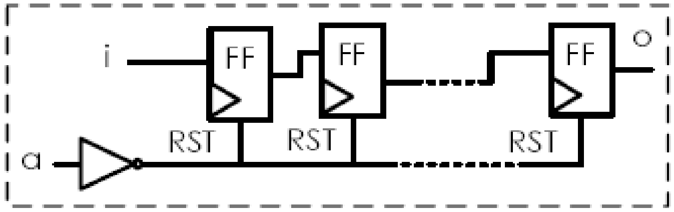


Regular Expression: **b+[^\n]{2}**

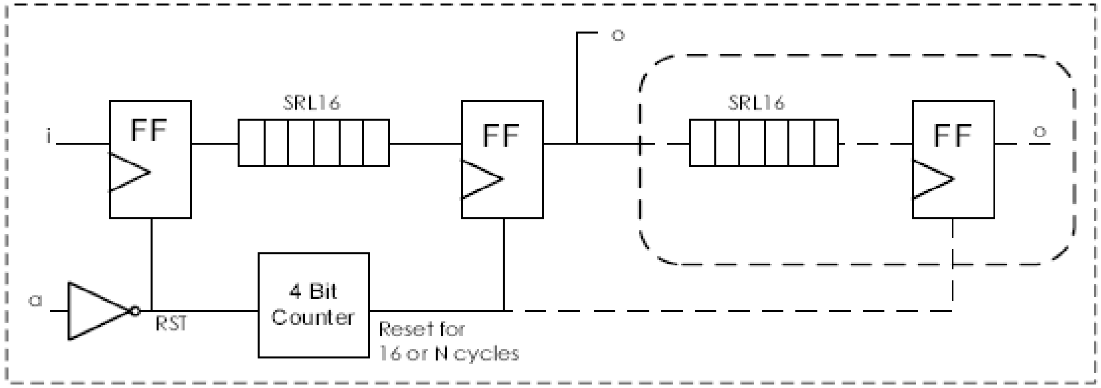
Constraint Repetitions



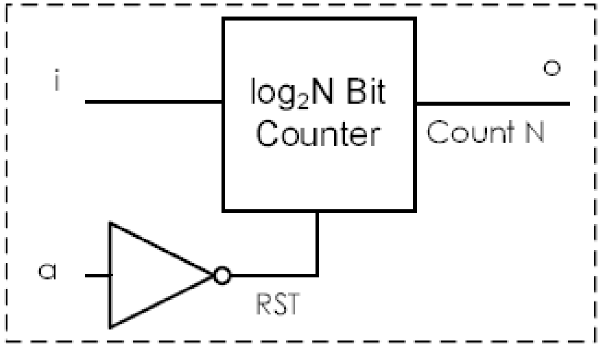
a{1} = a



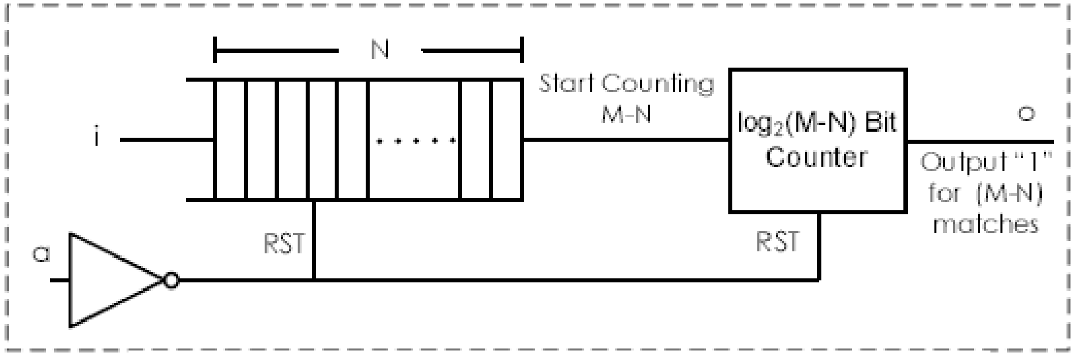
a{N} = aa...a, N times



The proposed Exactly block: $a\{N\}$. Successive flip-flops and SRL16s with a reset mechanism.

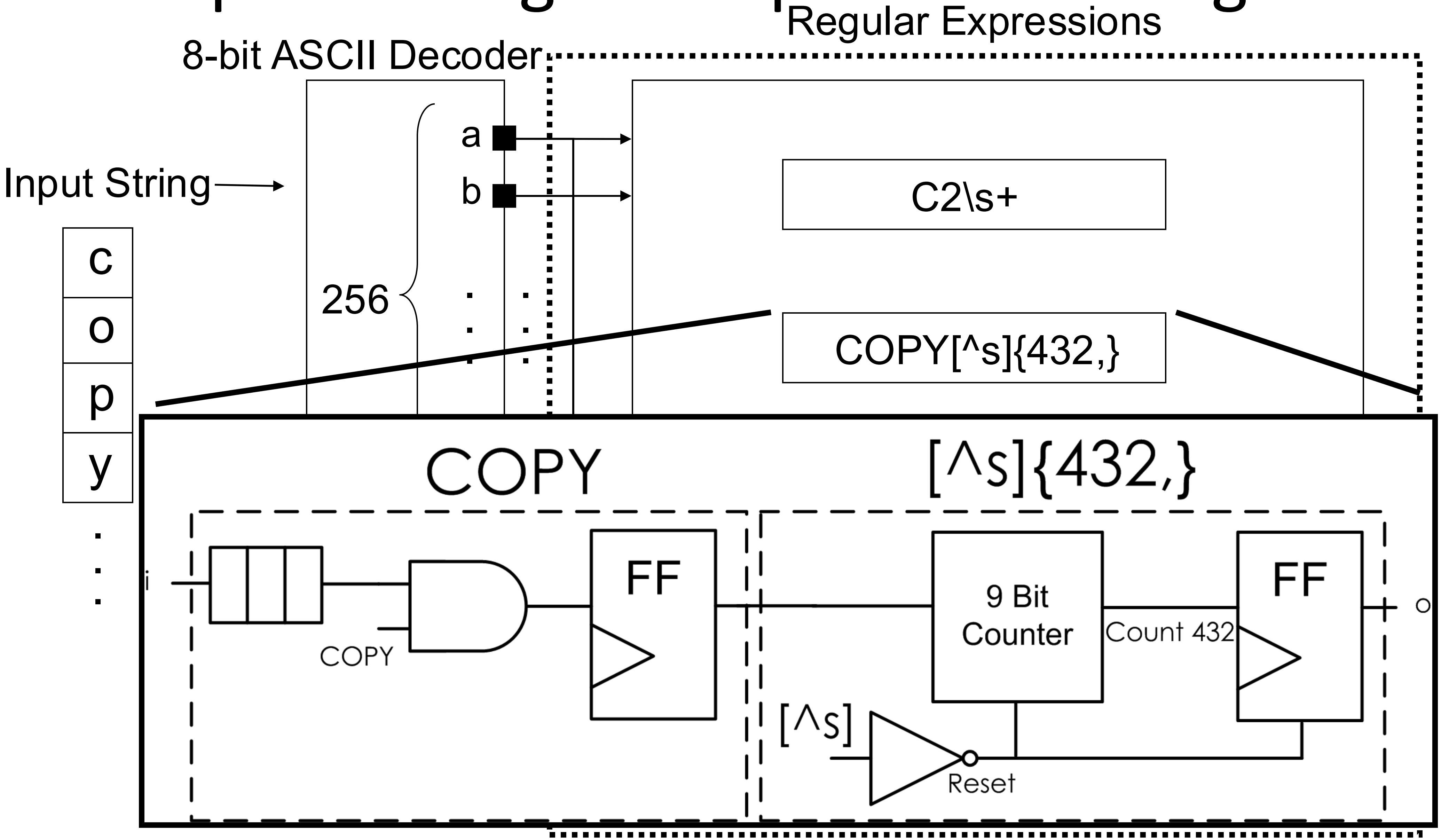


The AtLeast block: $a\{N,\}$.



The Between block: $a\{N, M\} = a\{N\}a\{0, M-N\}$.

Top-level Regular Expressions design



Results: Regular Expressions in Reconf. HW

NFAs in Reconfig. HW

- 1,500 Reg Expr, 70k characters
- Throughput: 2.1 Virtex2, 3 Gbps Virtex4
- Area 50k logic cells

DFAs in ASIC:

- 315 Reg Expr, 11k chars
- Throughput: 16 Gbps

NFAs in Software

- Only 20-220 Reg Expr
- Throughput: 1-56 Mbps

DFAs in SW:

- A few tens of Reg Expr
- Throughput: 0.6-1.6 Gbps
- Memory: several Mbytes

References

Books:

- **“Reconfigurable computing, the theory and practice of FPGA-based computing”**, by Scott Hauck, André DeHon, **Chapters: 21**

Papers:

- **“Design Patterns for Reconfigurable Computing”,** André DeHon, Joshua Adams, Michael DeLorimier, Nachiket Kapre, Yuki Matsuda, Helia Naeimi, Michael Vanier, and Michael Wrighton. In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, April 2004.
- **"Regular Expression Matching in Reconfigurable Hardware",** Ioannis Sourdis, João Bispo, João M.P. Cardoso and Stamatis Vassiliadis, *Int. Journal of Signal Processing Systems for Signal, Image, and Video Technology (Springer), Volume 51, Number 1, April 2008.*
- **"Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching",** Ioannis Sourdis and Dionisios Pnevmatikatos, *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), pp. 258-267, Napa CA, USA, April 2004,*
- **"A Reconfigurable Perfect-Hashing Scheme for Packet Inspection** ", Ioannis Sourdis, Dionisios Pnevmatikatos, Stephan Wong and Stamatis Vassiliadis, 15th International Conference on Field Programmable Logic and Applications (FPL'05), pp. 644-647, Tampere, Finland, August 2005.

Links:

- Michael Flynn’s talk **“Accelerating computations with FPGAs”** (M. Flynn and O. Mencer @ Maxeler Technologies) [Video, Slides]

Summary

- Applications development using RC
  - Strengths and Weaknesses
  - Application Characteristics
  - Implementation Strategies

- Network Processing using RC
  - Pattern Matching
  - Regular Expressions matching

- Book chapter 21