Reconfigurable Mapping Functions for Online Architectures

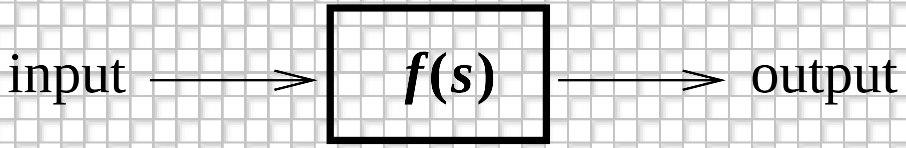
Clemson University

Shyamnath Harinath and Ron Sass

May 15, 2003

Mapping Functions

informally, we're talking about a RC module f


input
$f(s)$
output

with two operations

- add a key,value pair to f
- given a key, find the matching value

Mapping Functions (cont'd)

Closely related to a number of computing concepts...

- *associative memory*
- *content-addressable memory*
- in software, called a *dictionary*, with variety of implementations
  - ❑ hashing
  - ❑ trees
  - ❑ *et al.*

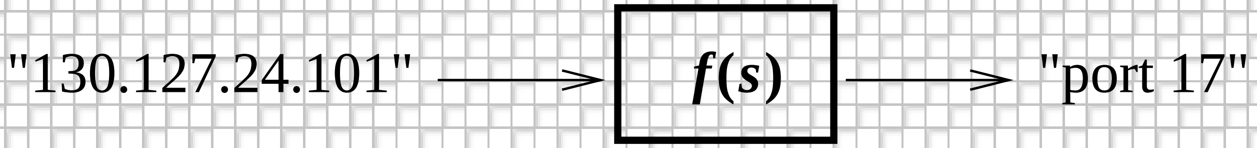
Network Classification

- some applications have stringent constraints (1–5 cycles) on the search operation
- for example, classifying network packets has numerous uses…
  - ❑ IP characterization
  - ❑ flow table and QoS-related router functions
  - ❑ network intrusion detection
- all of these problems require either looking at the packet header (or sometimes payload) and determining

≪back forw≫

Network Classification Example

- for example, a router might want to know ...
  - what outgoing port?



"130.127.24.101" → $f(s)$ → "port 17"

  - has the address/port been legally established?
  - if the payload contain a flagged signature?
- hardware solutions can often be prohibitively expensive

Outline

- ➪ Preliminaries
  - ❑ online architectures
  - ❑ formal problem statement
- ➪ Objective
- ➪ Three RC Implementations
- ➪ Experiments and Results
  - ❑ measures
  - ❑ platform
  - ❑ results

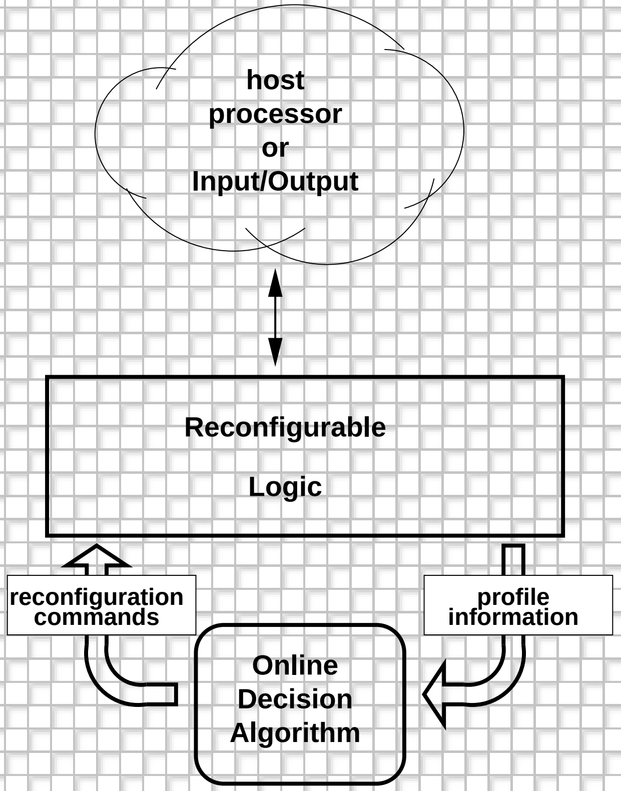
Reconfigurable Computing (RC)

- ➪ using FPGAs, RC realizes a digital hardware circuit (a ***configuration***) at run-time
- ➪ SRAM-based FPGAs can be reprogrammed repeatedly
- ➪ modern (larger) FPGAs support partial reconfigurable
- ➪ form the basis of run-time reconfigurable (RTR) systems where multiple circuits are cycled through during a single application

Online Architectures

further refinement of RTR systems is an ***online architecture*** where

- sequence of configurations not known *a priori*
- configuration not known *a priori*
- an online algorithm decides the next change at run-time



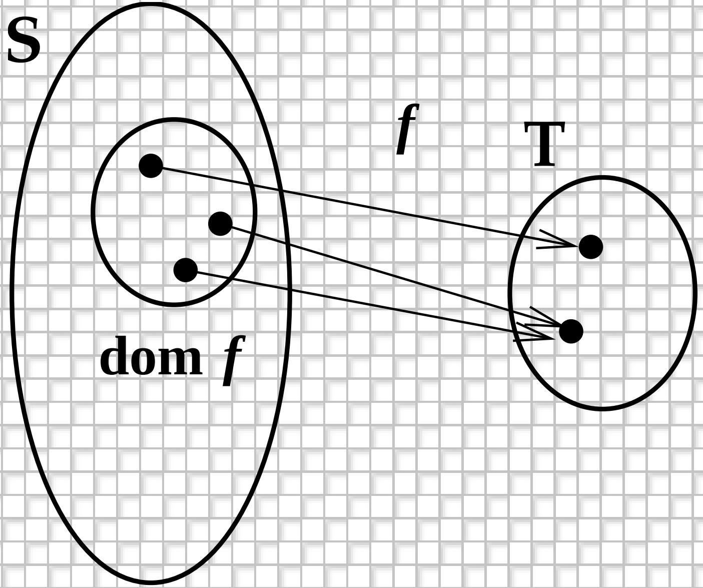
Arbitrary Mapping Function

We are interested in realizing mapping functions in online architectures; formally our mapping function is

- a partial function

  $$f : \mathbf{S} \nrightarrow \mathbf{T}$$

  where …



≪back forw≫

Arbitrary Mapping Function (cont'd)

where

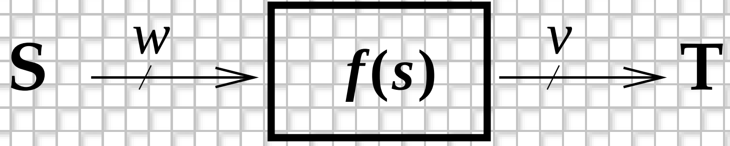
- the source set **S** $= \{s : \mathbb{Z} \mid 0 \leq t < 2^w \bullet s\}$

  where w is input width (in bits)

- the target set **T** $= \{t : \mathbb{Z} \mid 0 \leq t < 2^v \bullet t\}$

  where v is output width (in bits)

- the capacity, n, is $\lceil \log_2 n \rceil \approx v$



Arbitrary Mapping Function (cont'd)

➪ two operations; assuming $s \in \mathbf{S}$ and $t \in \mathbf{T}$

- ❑ SEARCH— given s find t; i.e. calculate $f(s)$
- ❑ ASSOC— given (s, t) and f make a new f' such that

$$f = f' \text{ except } f'(s) = t$$

Content-Addressable Memory (CAM)

- ➪ hardware device that
  - ❑ standard memory mode
  - ❑ match mode
- ➪ collections of small CAMs are used in
  - ❑ TLB (virtual-to-physical memory translations)
  - ❑ main memory hierarchies (caches)

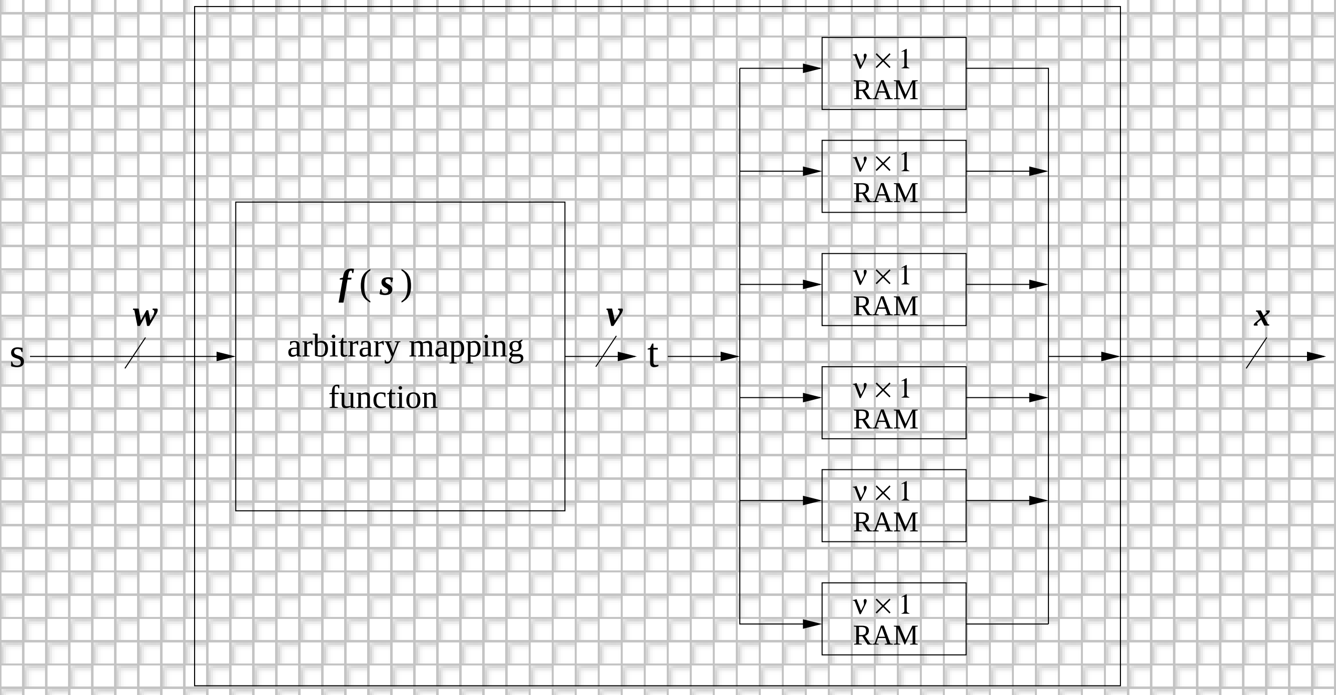
CAM device v. Arbitrary Mapping Function

- ➪ CAMs are similar to arbitrary mapping functions except that a CAM's capacity is usually not related to $|\mathbf{S}|$ or $|\mathbf{T}|$
- ➪ recall in our problem, we assume capacity

$$n \ll |\mathbf{S}| \qquad n \approx |\mathbf{T}|$$

- ➪ mapping functions can be easily extended to CAMs with a bank of $v \times 1$ RAMs

Content-Addressable Memory (cont'd)



Bank of RAMs (*x* times ν × 1)

Objective

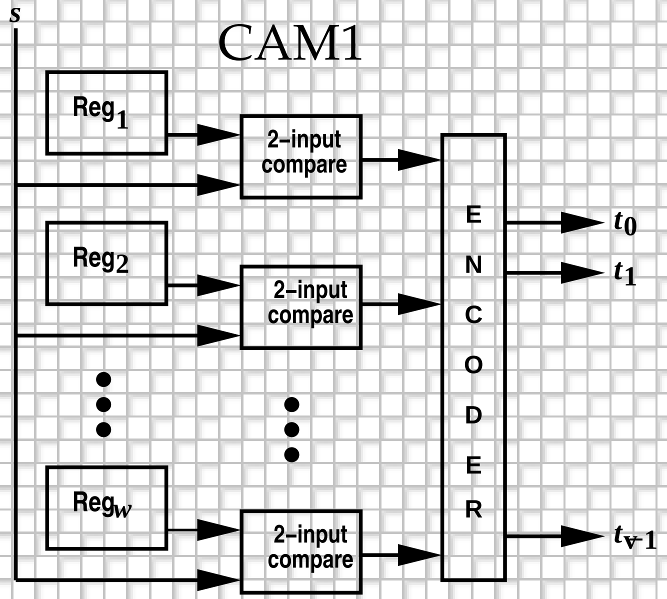
- ➪ replace CAMs and other dictionary structures with mapping functions implemented in online architectures
- ➪ for our motivating applications, we can assume
  - ❑ SEARCH is very common and must be very fast
  - ❑ ASSOC is less frequent and can be more costly
- ➪ The Question: While meeting the tight timing constraints of the SEARCH operation, what implementation maximizes capacity and minimizes reconfiguration time (ASSOC operation)?

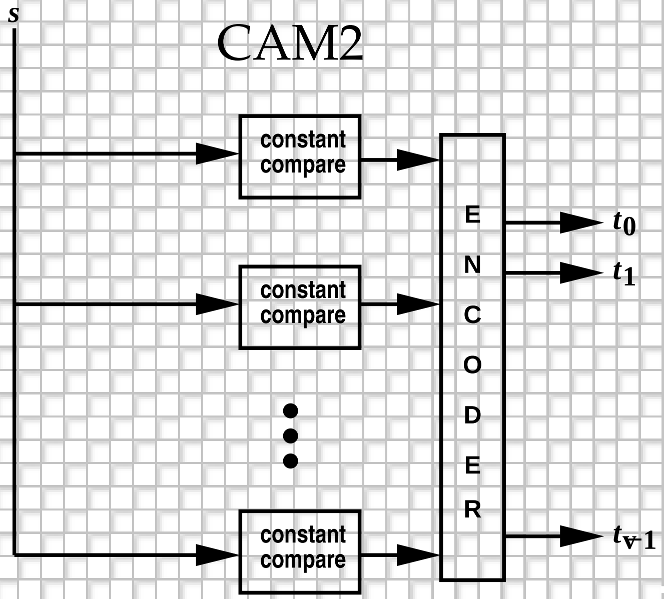
Three Implementations

- CAM1/CAM2
- RCAM
- QM-Tab

CAM1/CAM2

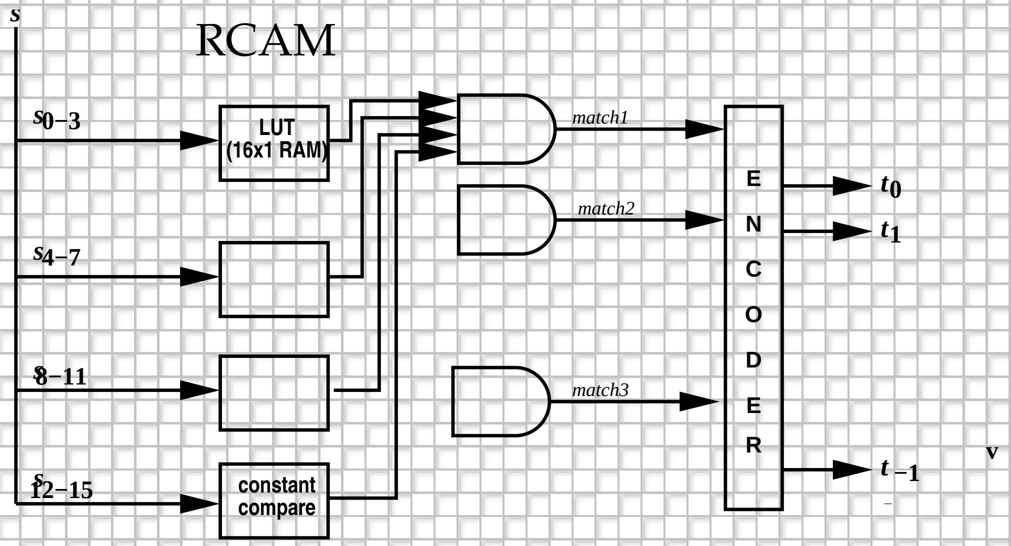
- CAM1 — registers each $s \in \mathbf{S}$ separately, feeds a two-input comparator
- CAM2 — configures constant comparators for each $s \in \mathbf{S}$





RCAM

- as described previously[*] aim for a single match (nomatch) signal using LUTs to store data and matching function

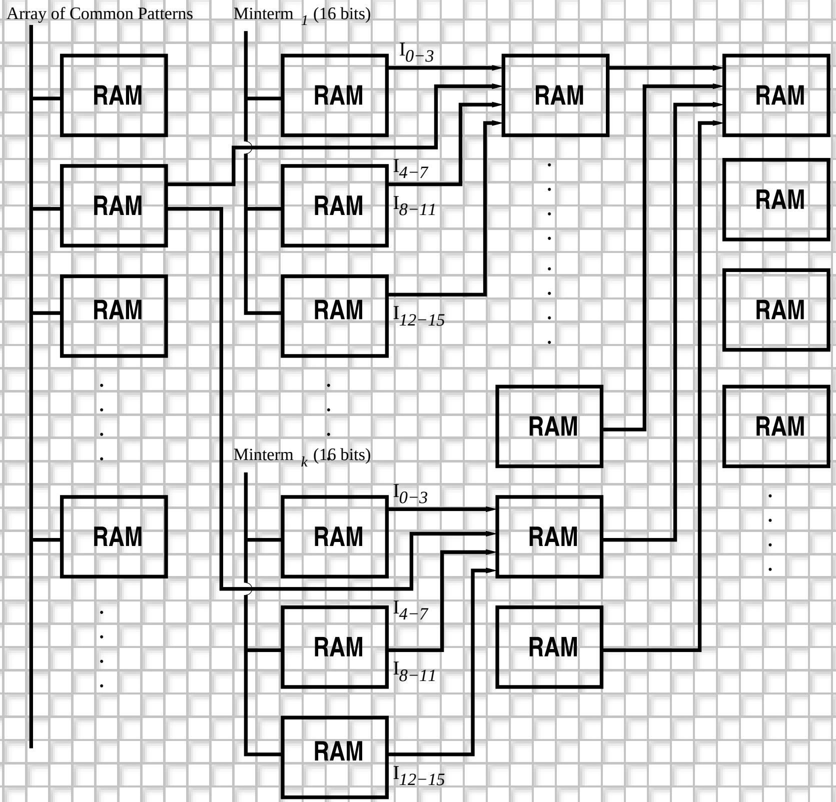


[*] Steve Guccione, Delon Levi, and Daniel Dows, "A Reconfigurable Content Addressable Memory," RAW 2000

QM-Tab

- applies the multibit Quine-McClusky tabulation method to the function f to reduce the number of gates and then maps to LUTs
    - ❑ each output bit in t has a separate function (all functions share common minterms)
    - ❑ shared minterms lead to larger capacity
    - ❑ ASSOC is a significant software process
    - ❑ the number of cascaded LUTs increases very slowly with capacity
- unlike the others, performance is highly dependent on data set

QM-Tab (cont'd)


Array of Common Patterns
Minterm $_1$ (16 bits)
Minterm $_k$ (16 bits)
RAM
I_{0-3}
I_{4-7}
I_{8-11}
I_{12-15}

Experiments

- first-order effects of technical choices
- use Java for software processes
- use JHDL for hardware description
- use Xilinx tool chain to generate bitfiles
- target hardware: XC4085XLA (no partial reconfiguration)
- important measures: space, SEARCH latency, ASSOC compute time

Measures

- ➪ space — $s(n, w)$ measured in CLBs (imprecise)
- ➪ SEARCH latency — measured in cycles
- ➪ ASSOC— total time is reconfigure time plus time to determine next configuration
  - ❏ for 4085 (no partial reconfiguration), reconfigure time is constant
  - ❏ we simply report software time-to-compute next configuration
- ➪ some implementations depend on the data; for the others we summarize
- ➪ ultimately, we'd like to know for each implementation a range of w and capacities n for a device

Summary Results

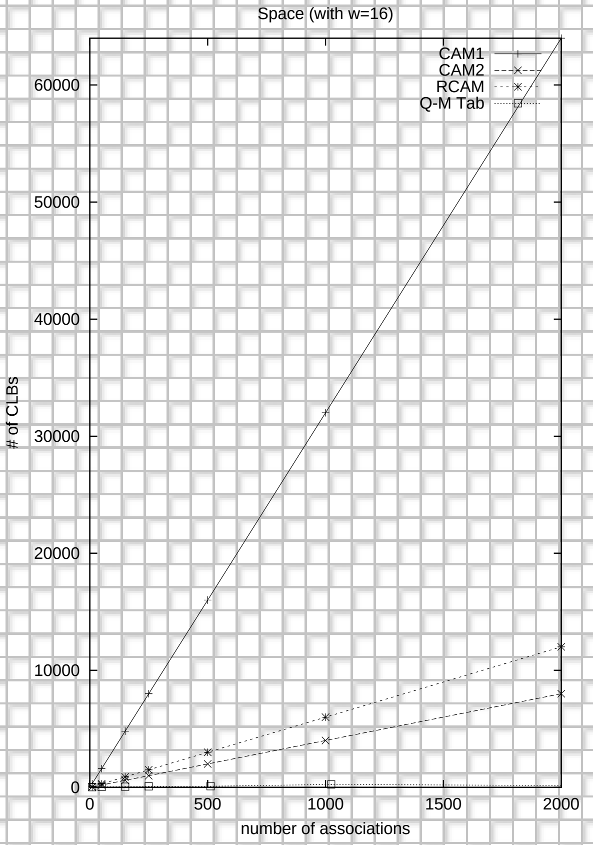
Implementation	$S(n;\ w)$	cycles	fixed routing?	$T_1(n)$
CAM1	$3 \times n \times \lceil w/2 \rceil$	3	y	none
CAM2	$2 \times n \times \lceil w/8 \rceil$	2	y	$O(1)$
RCAM	$3 \times n \times \lceil w/8 \rceil$	2	n	$O(n)$
QM Tabulation	varies	$O(1)^\dagger$	n	$O(2^n)$

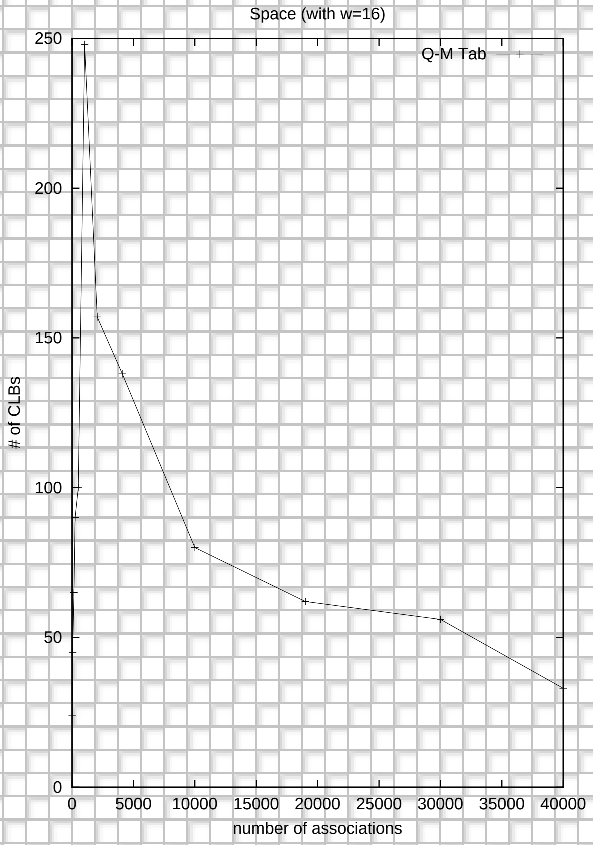
† usually 2–3

Data Dependent Measures

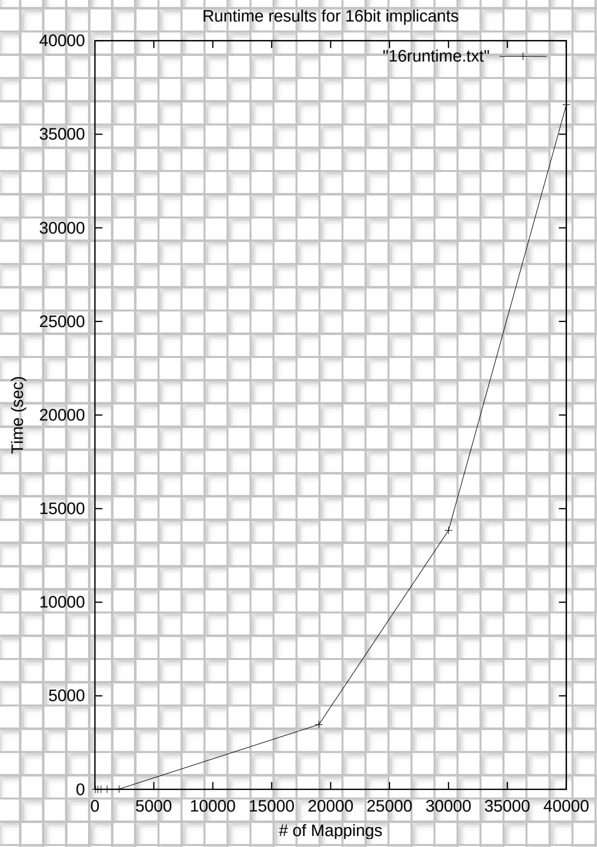
- to complete the comparison, we need data samples for for QM-Tab ASSOC operation
  - ❑ randomly select various size sets of $s \in \mathbf{S}$ elements
  - ❑ for each assign a random output t
- worked with the range of capacities and a fixed input width w for QM-Tab ASSOC operation

Space for a fixed $w = 16$

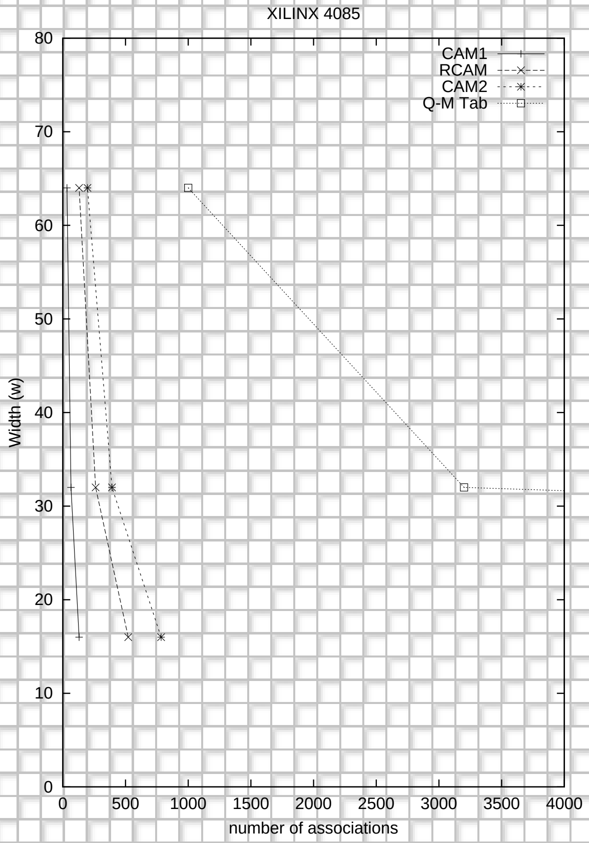




Time to Determine Next Configuration



Capacity XC4085



≪back forw≫

Analysis : Resource Utilization

- clearly, CAM1/CAM2 are very wasteful of resources, severely limiting capacity
- RCAM lies in between CAM2 and QM-Tab but is much closer to CAM2
- QM-Tab — clear winner in capacity

Analysis : Structure

- ➪ the real story behind time-to-reconfigure is the circuit's structure
  - ❑ CAM1/CAM2 — simple structure with easily accessed (s, t) pairs (none or almost no software process needed)
  - ❑ RCAM — slightly more complicated structure but the re-programming is not free and may require re-routing
  - ❑ QM-Tab — total loss of structure and we are unaware of any incremental algorithms

Conclusion

- all implementations meet SEARCH cycle requirements
- CAM1/CAM2 and RCAM are of limited use due to their relatively small capacity
- QM-Tab delivers desired capacity but existing time-to-reconfigur algorithm
  - is costly
  - not intended to be incremental
- *results suggest that an incremental algorithm similar to multi-bit QM-Tab is possible that approaches QM-Tab's capacity but with less reconfiguration cost*

Thank You!

For more information, please visit the Parallel Architecture Research Lab web page:

http://www.parl.clemson.edu/

or email me...

rsass@clemson.edu

List of Slides

Content-Addressable Memory (CAM)

- hardware device that
  - standard memory mode
  - match mode
- collections of small CAMs are used in
  - TLB (virtual-to-physical memory translations)
  - main memory hierarchies (caches)