ANCS 2008

Compact Architecture for High-Throughput Regular Expression Matching on FPGA

Yi-Hua E. Yang, Weirong Jiang and Viktor K. Prasanna
EE, USC

Kyle Wang, NSLAB, Tsinghua University

Outline

- Introduction
- Background
- Basic Architecture on FPGA
- Architectural Optimizations
- Performance Evaluation
- Conclusion and Future Work

Introduction

- Basic regular expression example (**What**)
  - `b*c·(a|b)*[ac]#`
- RE-NFA & RE-DFA (**How**)
  - regular expression matching (REM) architecture based on (Non-)Deterministic Finite Automaton
- Bottleneck for DPI in IDS (**Why**)
  - large number of patterns to scan for and increasing bandwidth of network traffic
- An improved RE-NFA approach on FPGA
  - *n*-state *m*-character input REME (REM Engine) with O(n x $\log_2 m$) constructing time and O(n x m) logic units

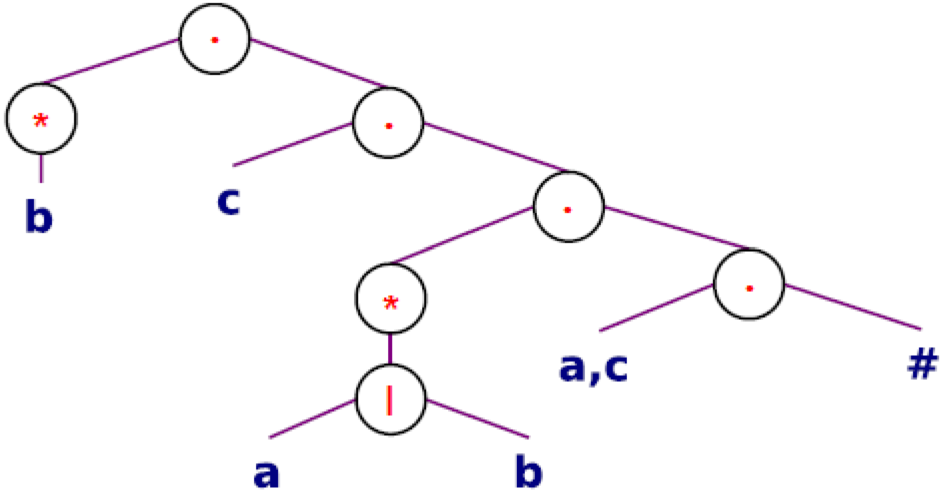
Background

- Challenges to perform REM on hardware
  - processing large numbers of patterns in parallel
  - obtaining high *concurrent throughput*
    - throughput of the input stream processed concurrently by *N* REMEs
    - RE size/complexity, amount of resources, achieved clock frequency
- Prior work upon implementation of RE-NFA on hardware
  - “The Compilation of Regular expressions into Integrated Circuits” (*Floyd et al, J.ACM 1982, Cited by 85*)
  - “Fast Regular Expression Matching Using FPGAs” (*Sidhu et al, FCCM 2001, Cited by 272*)
  - optimizations (e.g. utilizing BRAM)……

Basic Architecture on FPGA

From Regular Expression to NFA(1/3)

- Parse the regular expression into a tree structure

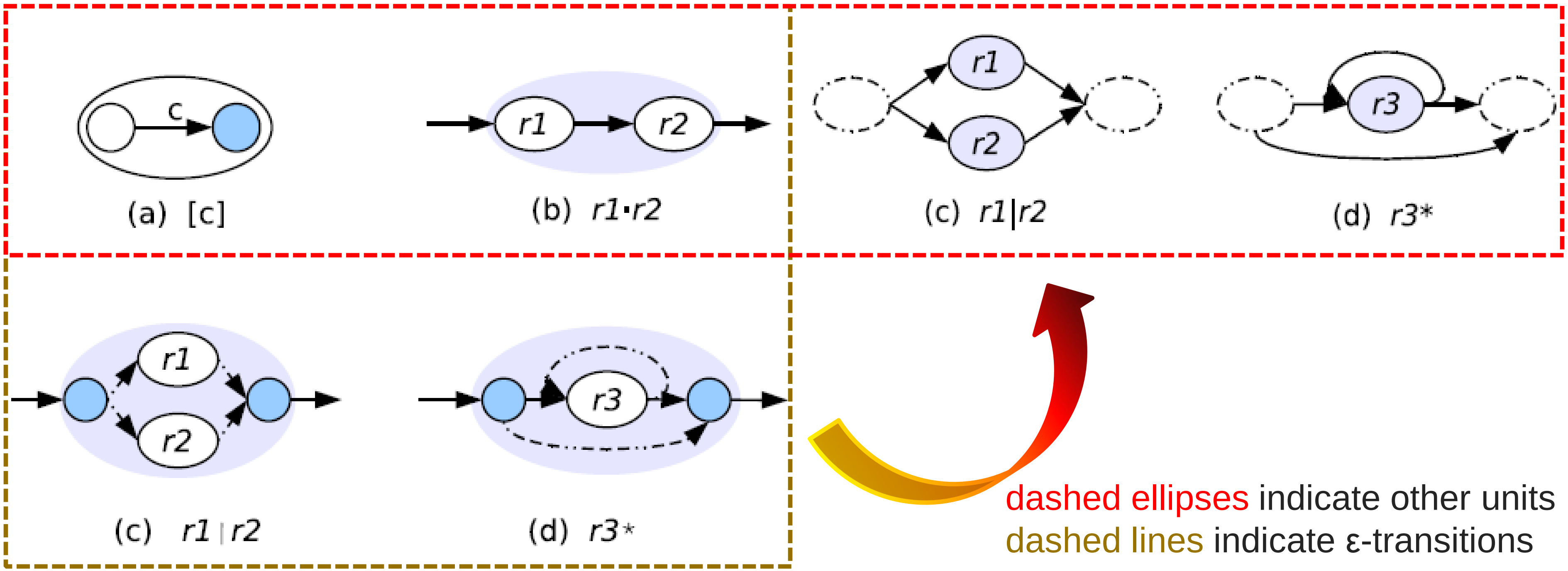


The parse tree for regular expression b*c(a|b)*[ac]#

Basic Architecture on FPGA

From Regular Expression to NFA(2/3)

- Modified McNaughton-Yamada construction



(a) [c]

(b) r1·r2

(c) r1|r2

(d) r3*

(c) r1 | r2

(d) r3*

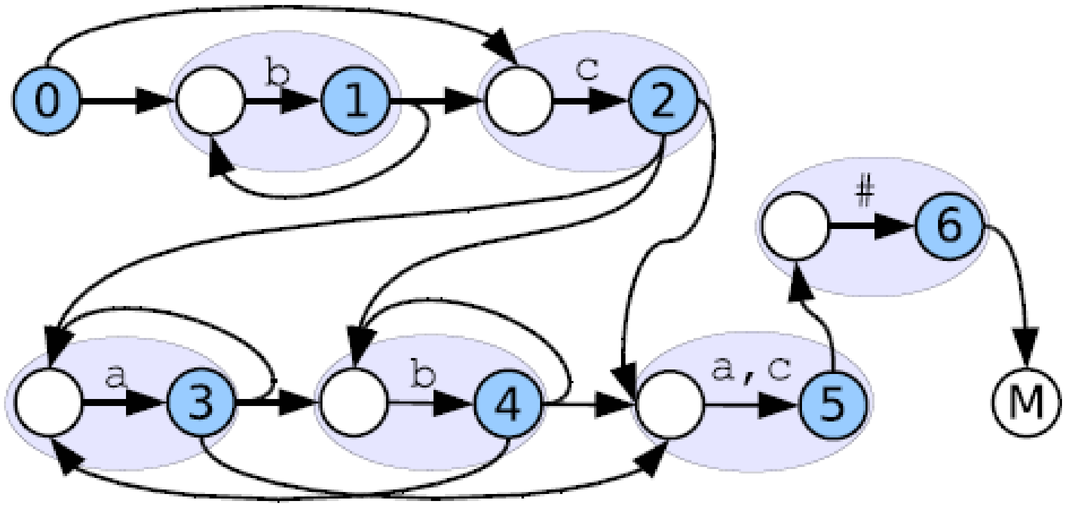
dashed ellipses indicate other units

dashed lines indicate ε-transitions

Basic Architecture on FPGA

From Regular Expression to NFA(3/3)

- Constructing the "modular" NFA using the modified McNaughton-Yamada rules

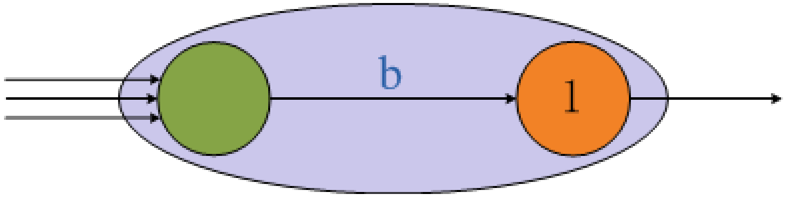


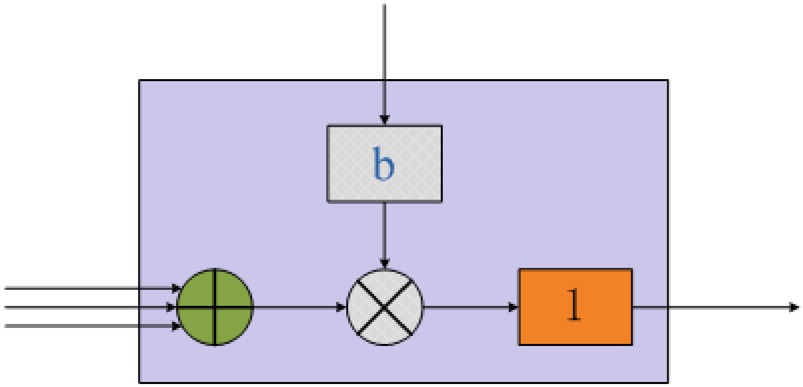
The RE-NFA for regular expression b*c(a|b)*[ac]#

Basic Architecture on FPGA

From RE-NFA to HDL(1/2)

- Mapping state directly into logic module

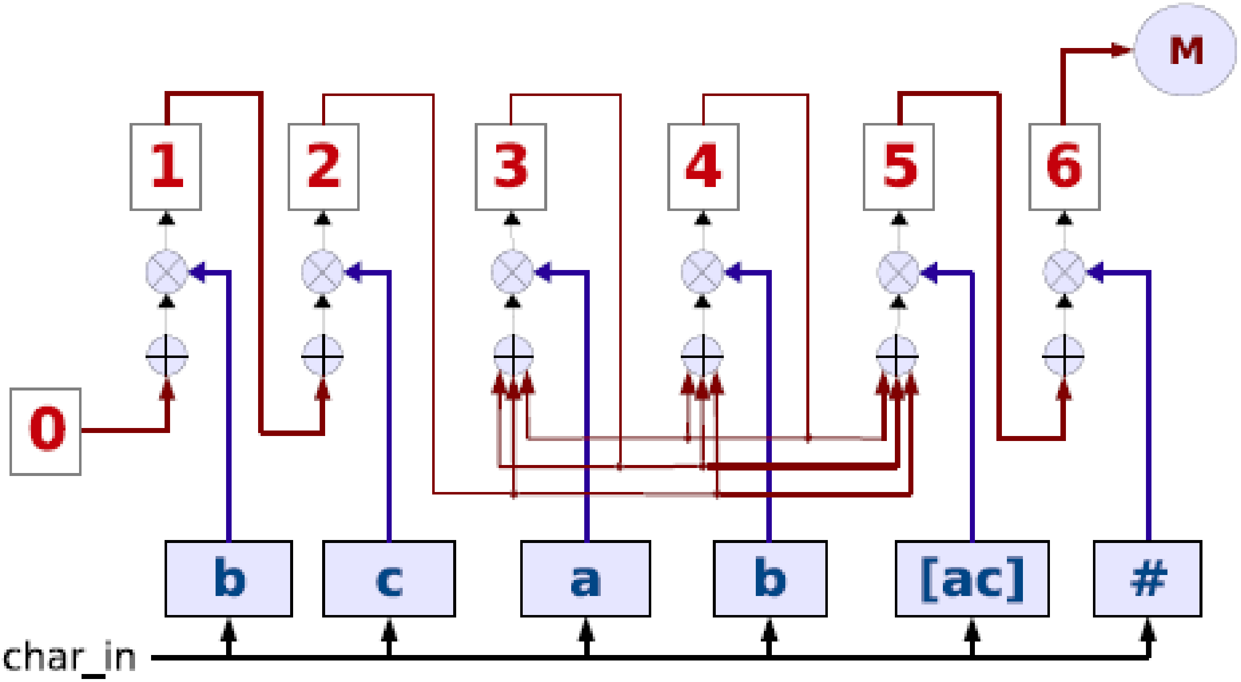




Basic Architecture on FPGA

From RE-NFA to HDL(2/2)

- Mapping NFA directly to HDL



Matching circuit for regular expression b*c(a|b)*[ac]#

Basic Architecture on FPGA

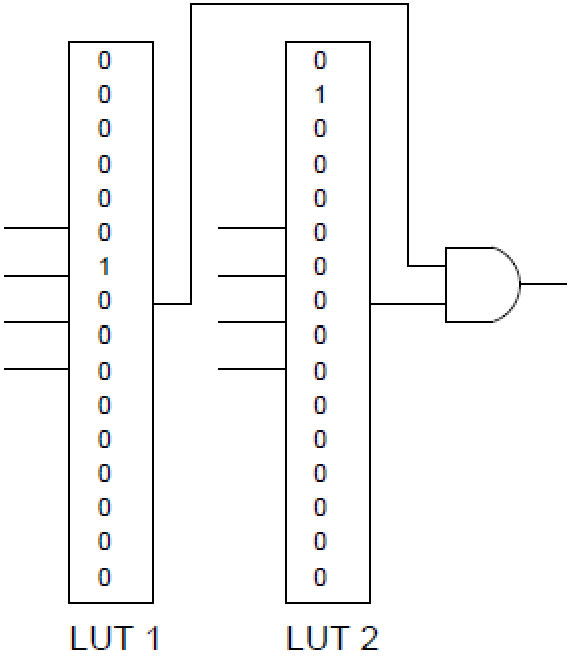
BRAM-based Character Classification(1/2)

- Character classification unit
  - 256 bits for specifying all 8-bit characters
    - e.g. character a can be classified by $\underbrace{\overbrace{000\cdots01}^{91\#}00\cdots0000}_{256\#}$
    - max 256 x *n* bits BRAM usage for *n*-state RE-NFA
    - Less if two states use the same character
  - cons.
    - redundancy in BRAM
    - max 2^{256} possible character classes (e.g. simple [ac])
    - optimization is necessary

Basic Architecture on FPGA

BRAM-based Character Classification(2/2)

- Dual-4bit selector
  - 2*16 bits
  - only ‘and gate’ delay
  - why not use?
    - logic slices *vs.* BRAM
    - clock frequency influence
- For character ‘a’
  - 0110 0001
  - if ‘.’ ?



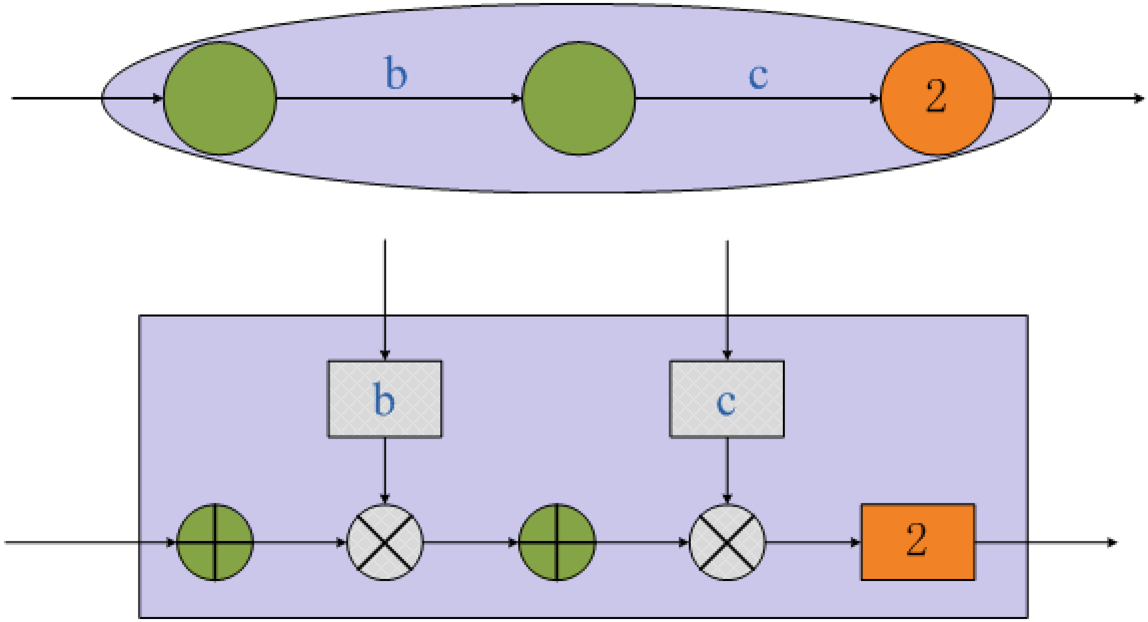
Architectural Optimizations

- Three available optimizations to improve the basic design above
  - Multi-Character Input Matching
  - Centralized Character Classification
  - Staging and Pipelining
- These techniques are unique to our design and take advantage of the modularity of the proposed architecture.
  - Scalable pattern matching for high speed networks(multi-character)
  - Regular Expression Matching for Reconfigurable Packet Inspection(centralized)
  - Assisting Network Intrusion Detection with Reconfigurable Hardware(pipeline)

Architectural Optimizations

Multi-Character Input Matching(1/2)

- 2-character input (2-Stride Accelerating)

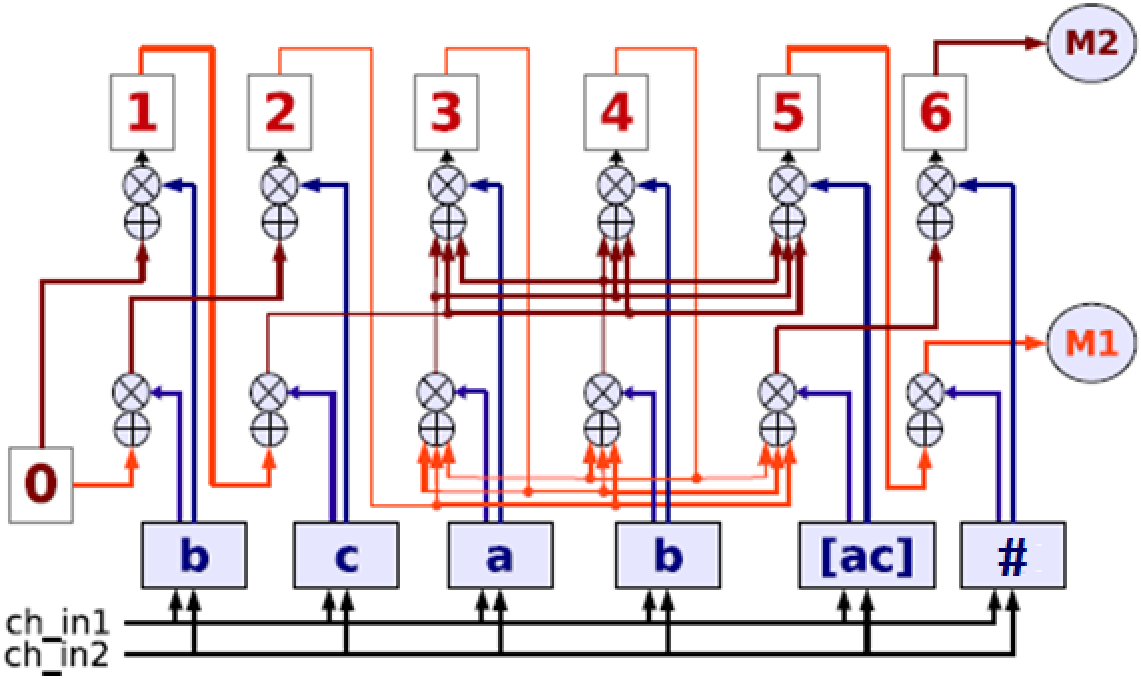


Multiple outputs may be at ‘and gate’ or ‘state register’

Architectural Optimizations

Multi-Character Input Matching(2/2)

- O(n x $\log_2 m$) time Alg. for m-character input



2-input matching circuit for regular expression b*c(a|b)*[ac]#

Architectural Optimizations

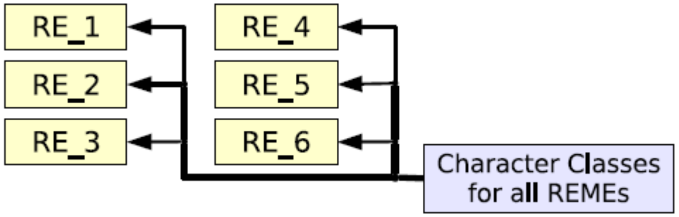
Centralized Character Classification(1/2)

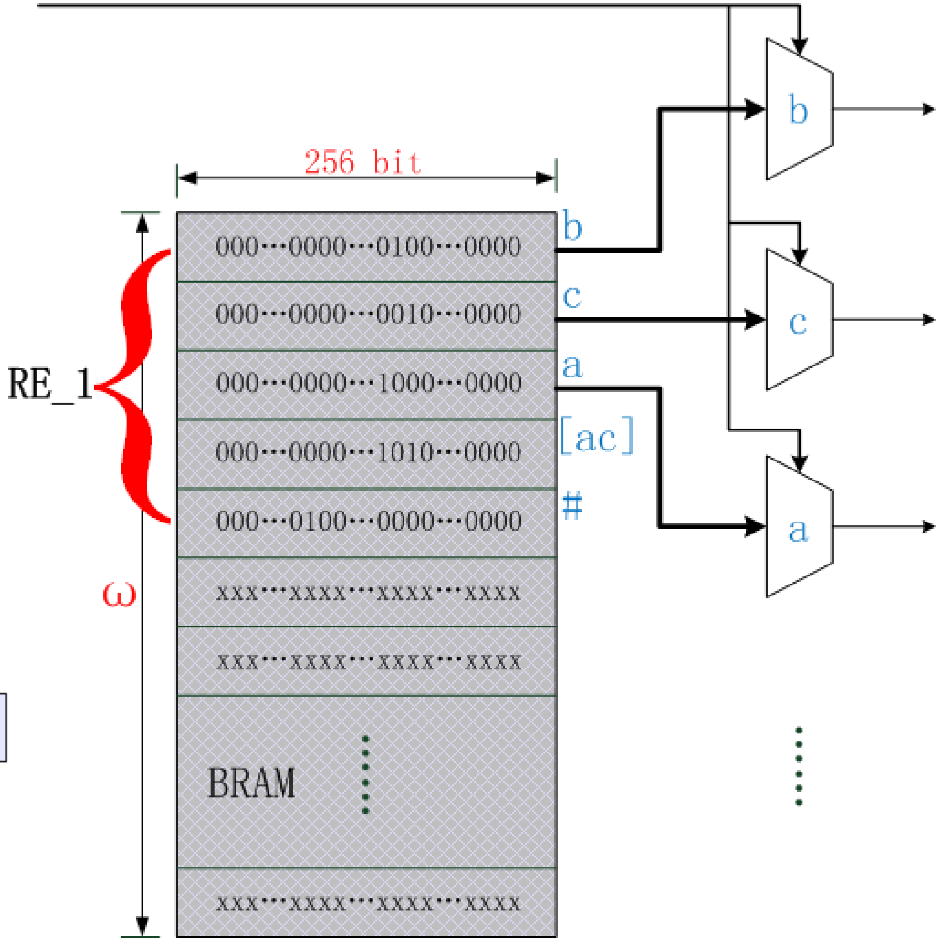
- Call function to examine and compare each state's character class to the character class entries (CCE) in BRAM so far
  - if new: a new 256-bit entry is added into BRAM
  - if old: proper connection is made from the BRAM output of the previous CCE to the input of the current state
  - O(n x ω) time complexity
    - ω is the number of distinct character classes among the n state, in the worst case, ω could be linear in n

Architectural Optimizations

Centralized Character Classification(2/2)

- BRAM storage
- How about the fan-out of the selector?



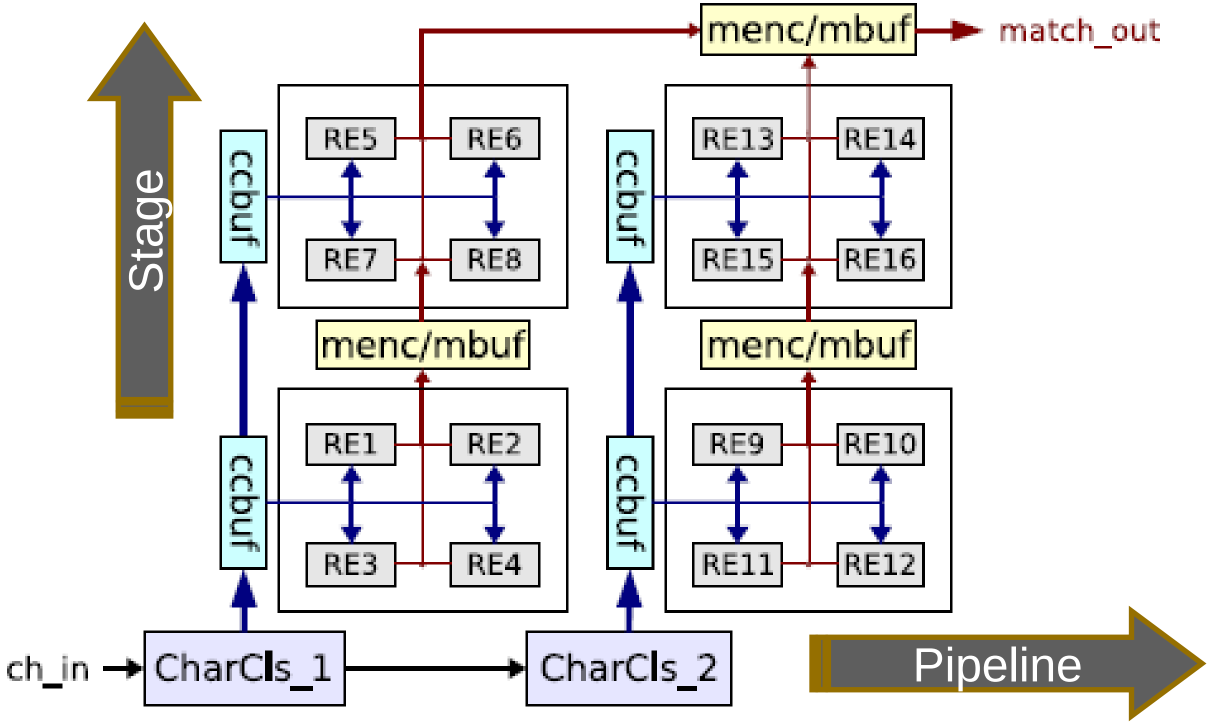


Architectural Optimizations

Staging and Pipelining

- Against the decline in achievable clock frequency with larger numbers of REMEs

fixed delay:
p (pipeline)
+ *s* (stage)
clock cycles



Performance Evaluation

Hardware Complexity of Regular Expression

- Metrics to quantify the complexity of RE
  - **area requirement**
    - **State count:** Total number of states needed by REME
  - **logic complexity**
    - **State fan-in:** maximum number of states that can immediately transition to any state in the REM
    - **State fan-out:** maximum number of states to which any state in the REME can immediately transition
  - **routing complexity**
    - **Loop size:** total number of transitions within a loop of state transitions
    - **Branch-size delta:** difference in number of transitions between two state transition paths with the same first and final states

Performance Evaluation

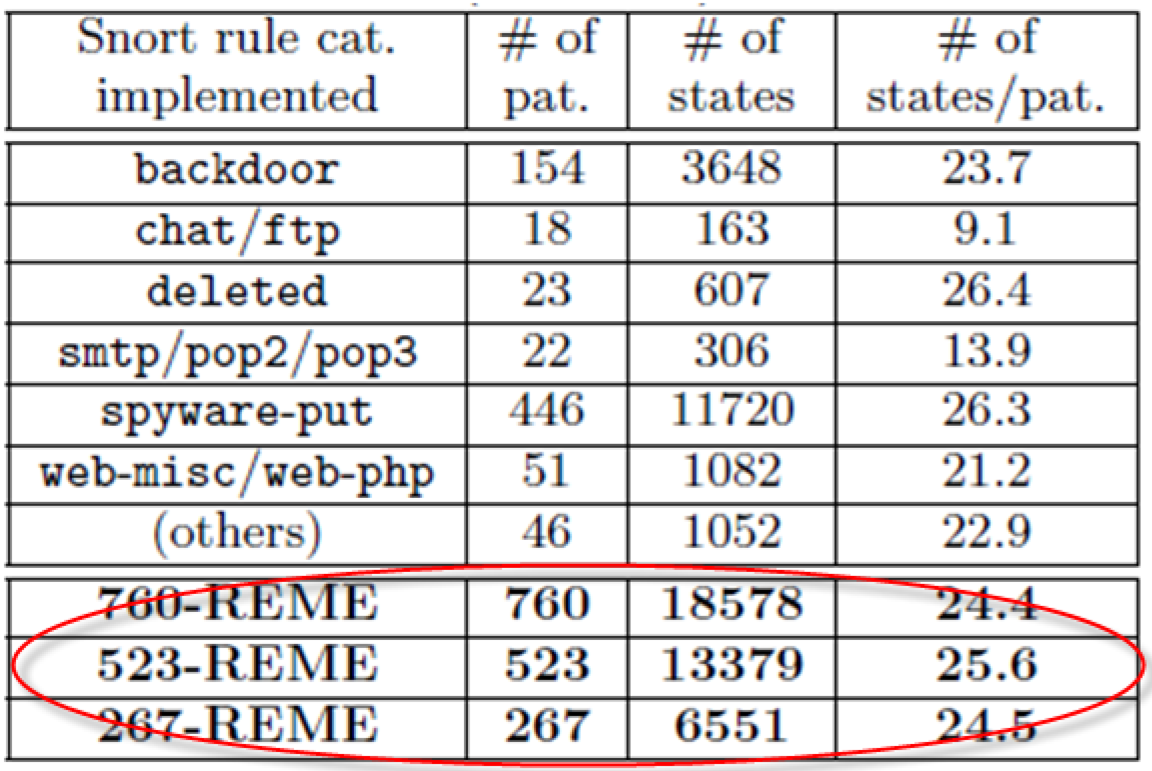
Implemented Regular Expressions(1/3)

- Patterns selected for RE-NFA evaluation
  - avoid patterns too short or too simple
    - can be dealt with pattern-level optimizations
  - count identical patterns in different rules as one
    - should not be used to inflate the REME numbers
  - avoid patterns with long quantified repetitions
    - can be easily dealt with shift-register
  - avoid patterns requiring backreference
    - requires capability beyond regular languages
    - is beyond the scope of this paper

Performance Evaluation

Implemented Regular Expressions(2/3)

- Snort rule categories (partially implemented)

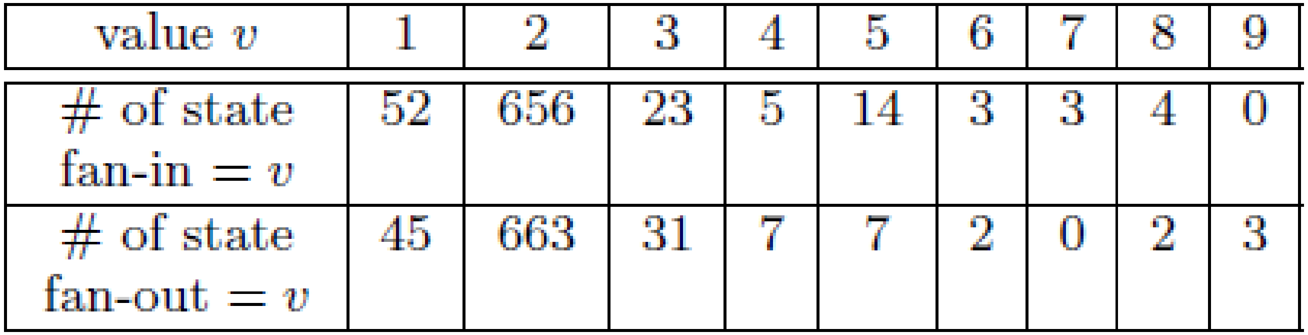


Snort rule cat. implemented	# of pat.	# of states	# of states/pat.
backdoor	154	3648	23.7
chat/ftp	18	163	9.1
deleted	23	607	26.4
smtp/pop2/pop3	22	306	13.9
spyware-put	446	11720	26.3
web-misc/web-php	51	1082	21.2
(others)	46	1052	22.9
760-REME	760	18578	24.4
523-REME	523	13379	25.6
267-REME	267	6551	24.5

Performance Evaluation

Implemented Regular Expressions(3/3)

- State fan-in and fan-out values

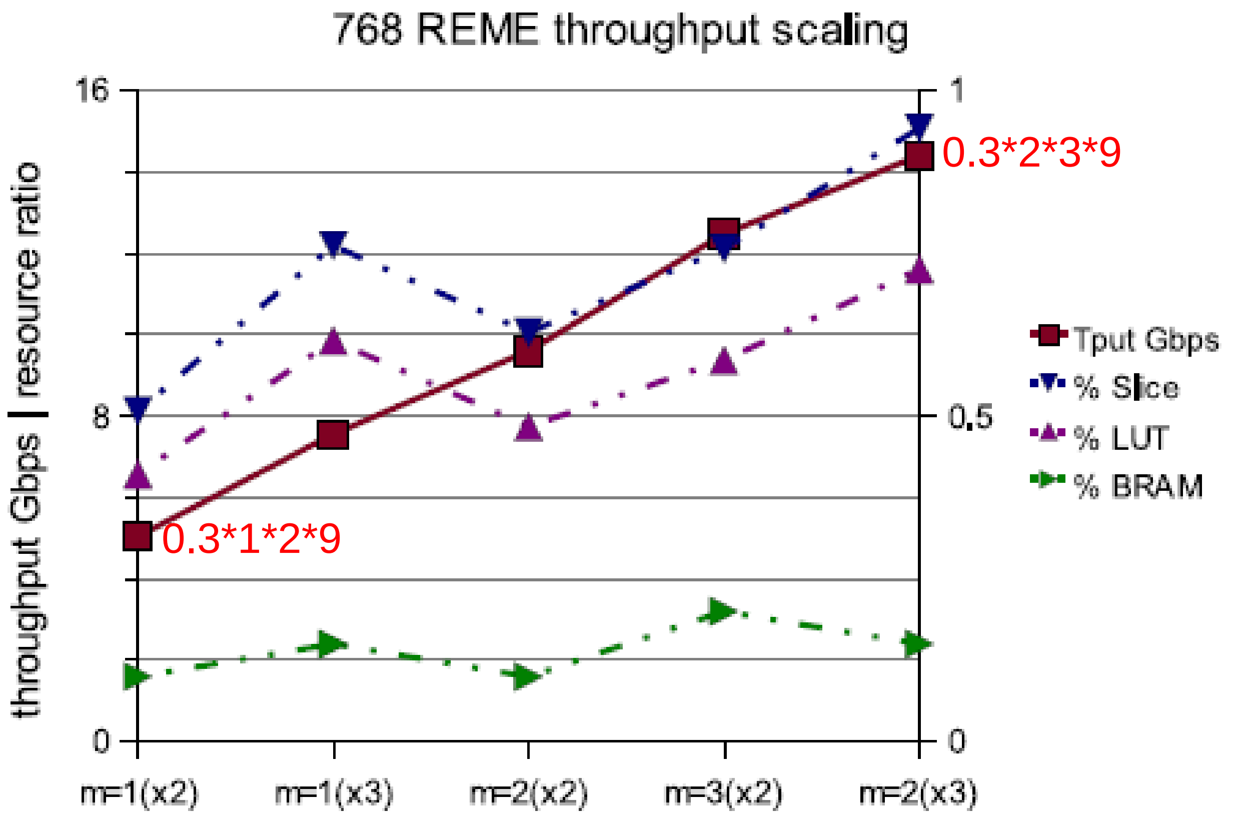


value v	1	2	3	4	5	6	7	8	9
# of state fan-in $= v$	52	656	23	5	14	3	3	4	0
# of state fan-out $= v$	45	663	31	7	7	2	0	2	3

- Although a few states reach 8 or 9, the slowest REME determines the overall clock frequency when implementing in the same clock domain

Performance Evaluation

Implementations Results(1/3)



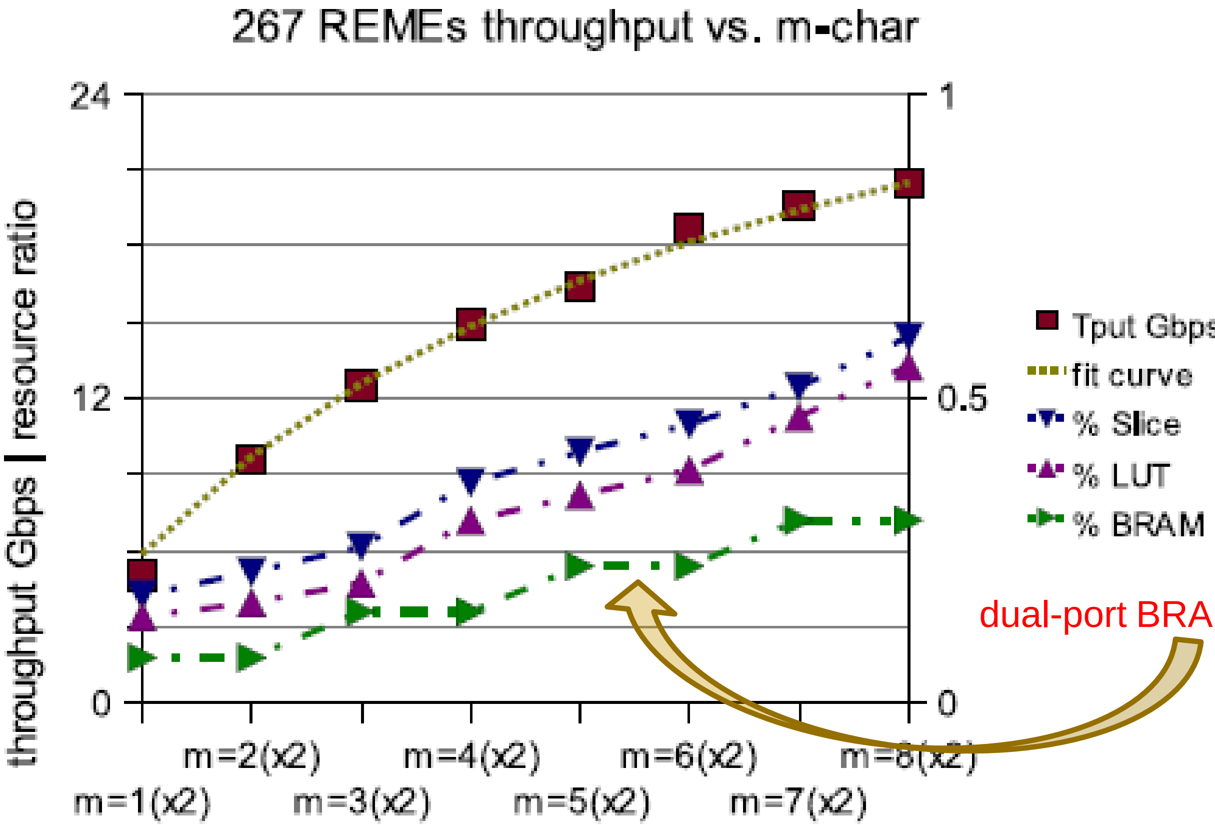
0.3*2*3*9

0.3*1*2*9

Throughput scaling of 760 REMEs on Xilinx Virtex 4 LX-100-12

Performance Evaluation

Implementations Results(2/3)

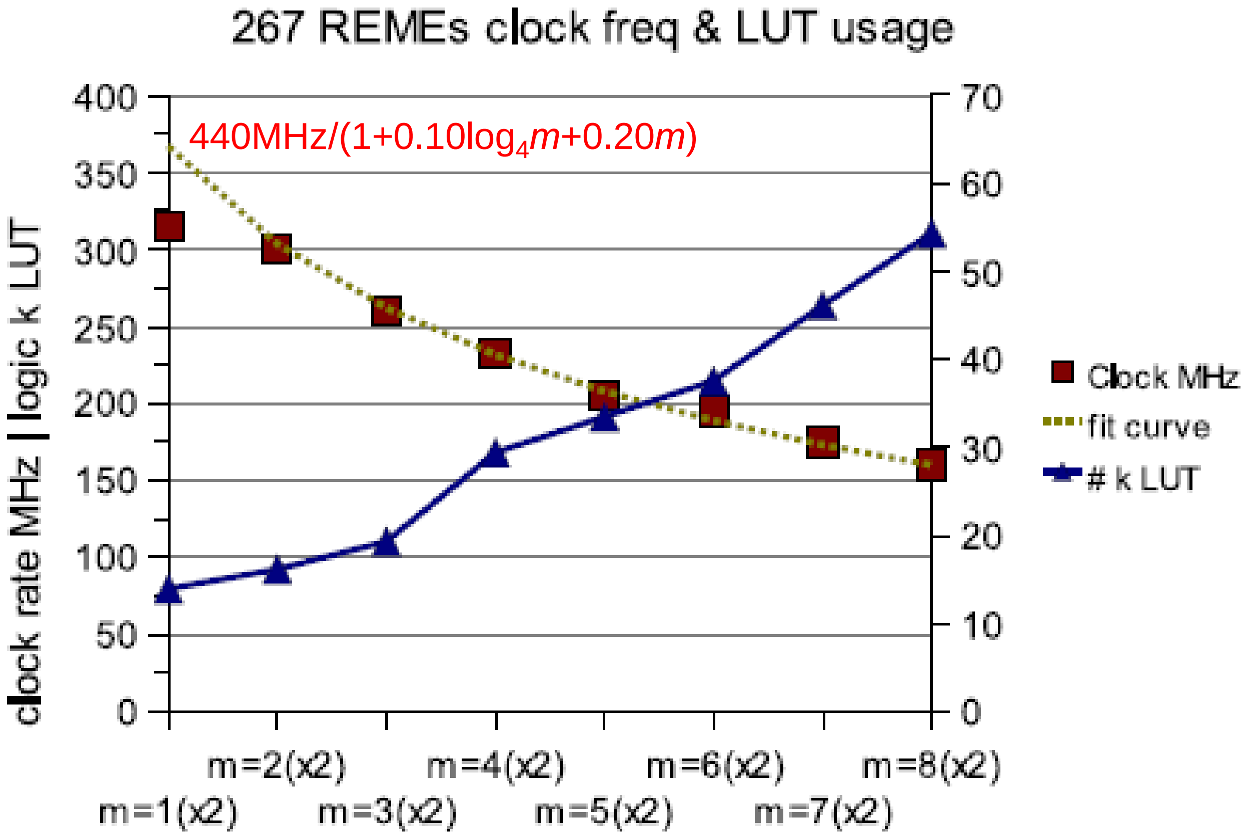


dual-port BRAM for 2-input

Throughput scaling of 267 REMEs on Xilinx Virtex 4 LX-100-12

Performance Evaluation

Implementations Results(3/3)

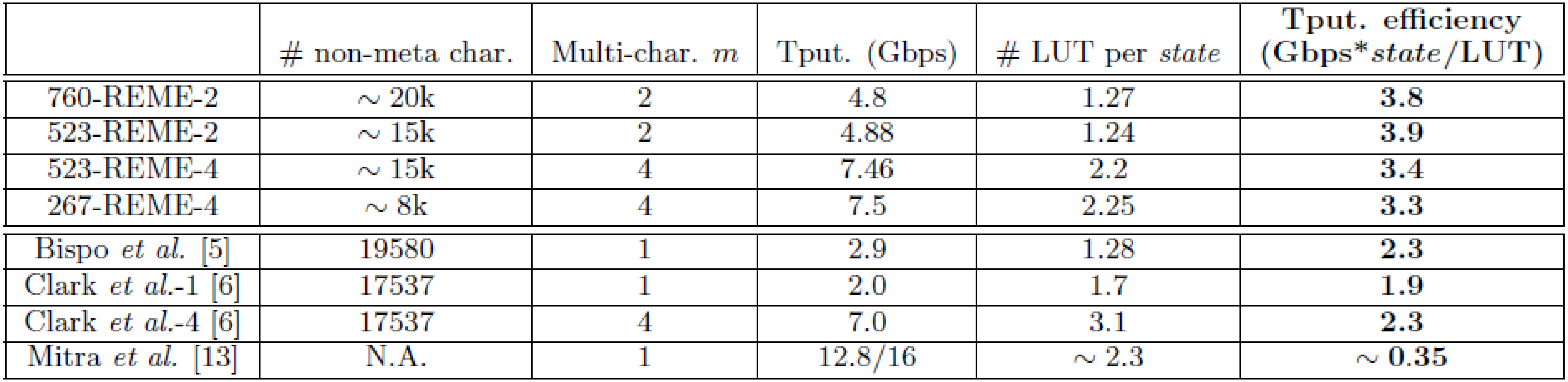


Clock rate and LUT usage of 267 REMEs on Xilinx Virtex 4 LX-100-12

Performance Evaluation

Performance Comparison

- Throughput efficiency
  - concurrent throughput of the circuit divided by the number of LUTs the circuit uses per state
  - a metric to evaluate performance in various conditions



	# non-meta char.	Multi-char. m	Tput. (Gbps)	# LUT per *state*	Tput. efficiency (Gbps**state*/LUT)
760-REME-2	~ 20k	2	4.8	1.27	**3.8**
523-REME-2	~ 15k	2	4.88	1.24	**3.9**
523-REME-4	~ 15k	4	7.46	2.2	**3.4**
267-REME-4	~ 8k	4	7.5	2.25	**3.3**
Bispo *et al.* [5]	19580	1	2.9	1.28	**2.3**
Clark *et al.*-1 [6]	17537	1	2.0	1.7	**1.9**
Clark *et al.*-4 [6]	17537	4	7.0	3.1	**2.3**
Mitra *et al.* [13]	N.A.	1	12.8/16	~ 2.3	**~ 0.35**

NOTE: The FPGA platform is not the same, but the throughput efficiency can still illustrate something

Conclusion and Future Work

- Conclusion
  - compact and easy-mapping architecture
  - stack character spatially to match multiple input
  - utilize block memory (BRAM) available
  - 2D staging and pipelining organization
- Future Work
  - group REMEs into stages intelligently (Fang Yu)
    - exploit pattern-level properties such as the common prefix extraction
  - separate simple and complex REs into different clock domains
    - supposedly a pipeline of simple REMEs can be clocked higher than complex REMEs

Thanks for your attention!