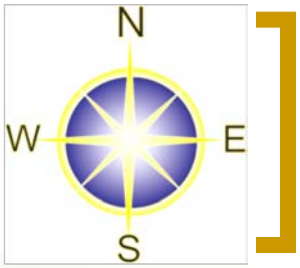
FCCM 2001

Fast Regular Expression Matching using FPGAs

Reetinder Sidhu and Viktor K. Prasanna
EE, USC

Kyle Wang, NSLAB, Tsinghua University

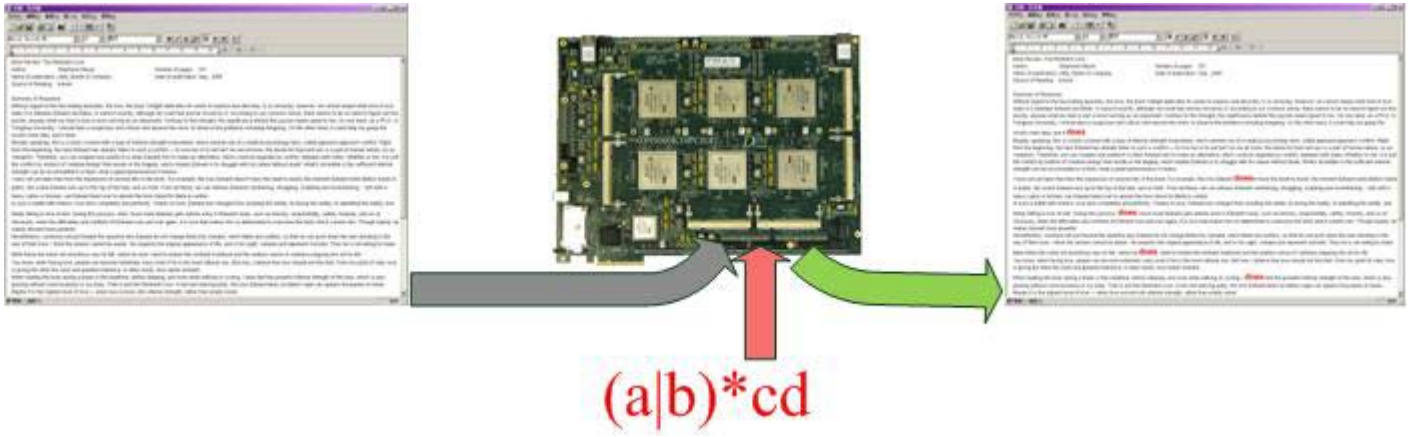
Outline



- Introduction
- NFA Construction using FPGA
- NFA Construction using SRGA
- Performance Evaluation
- Conclusion

Introduction

- Regular Expression Matching (REM)
  - Unix text search program —— grep



Input
Text File

Input
Regular Expression

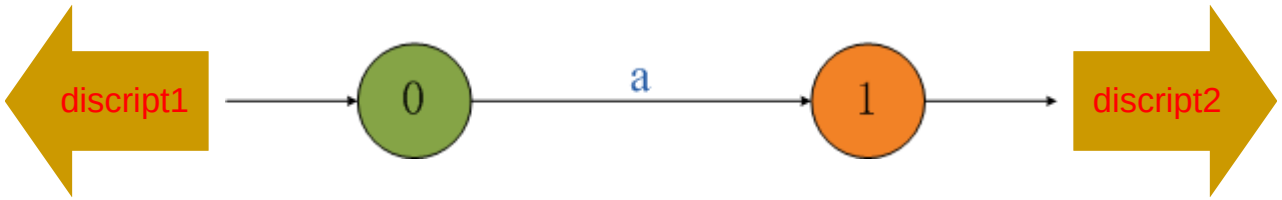
Output
Matching Stings

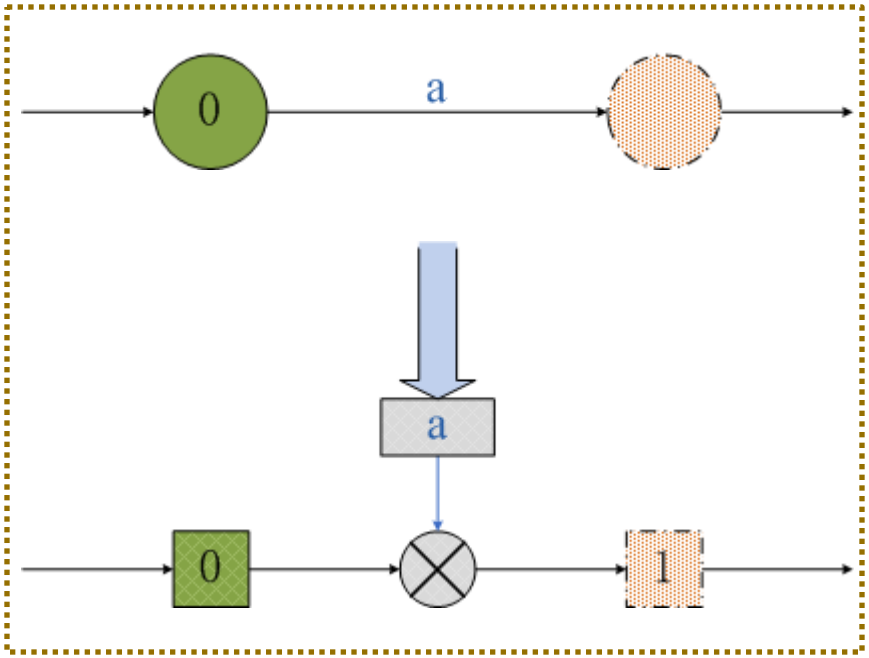
Introduction

- REM Algorithm Complexity
  - *n*-length RE, *m*-length text

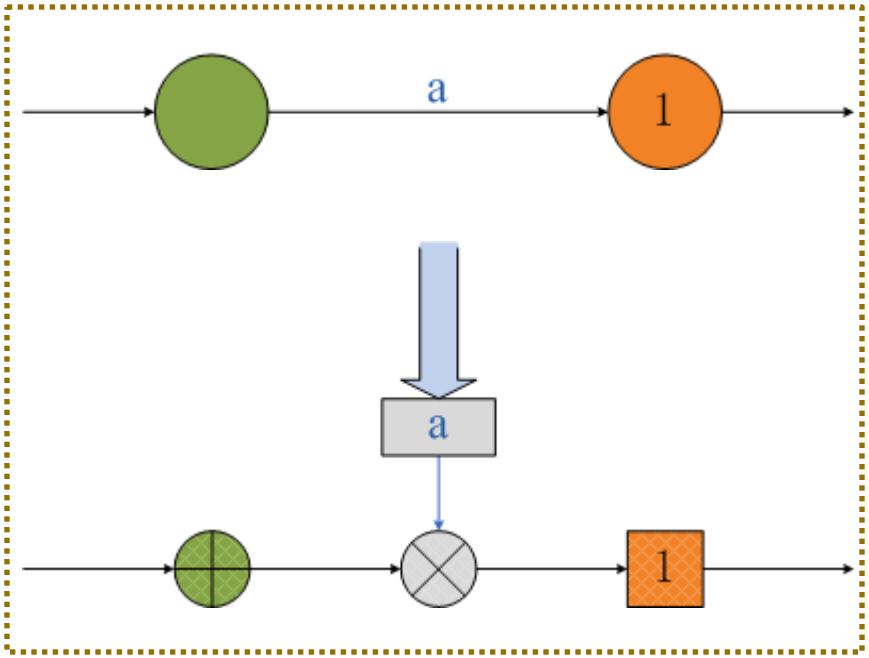
Approach	Memory	Constructing Time	Processing Time
DFA	$O(2^n)$	$O(2^n)$	$O(m)$
NFA	$O(n)$	$O(n)$	$O(nm)$
Proposed	$O(n^2)$	$O(n)$	$O(m)$

NFA Construction using FPGA





Vs.

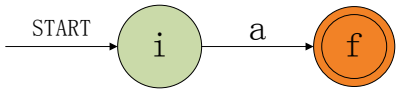


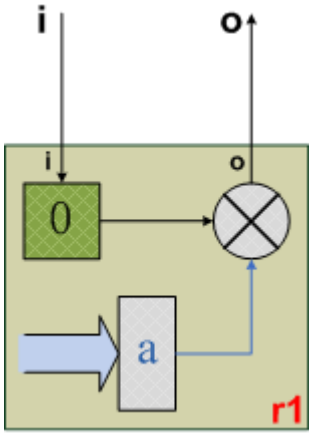
Sidhu's approach

Yang's approach

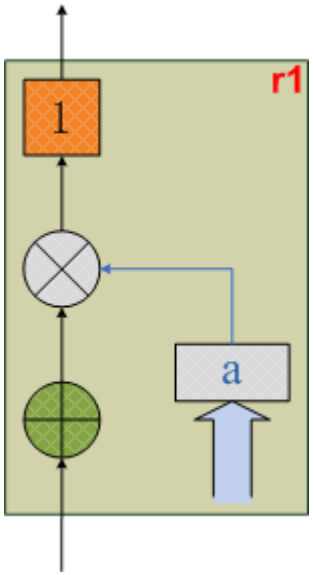
NFA Construction using FPGA

NFA(*a*)





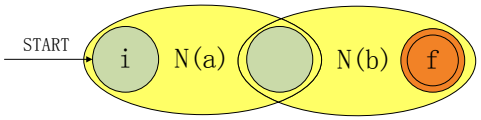
Sidhu's approach

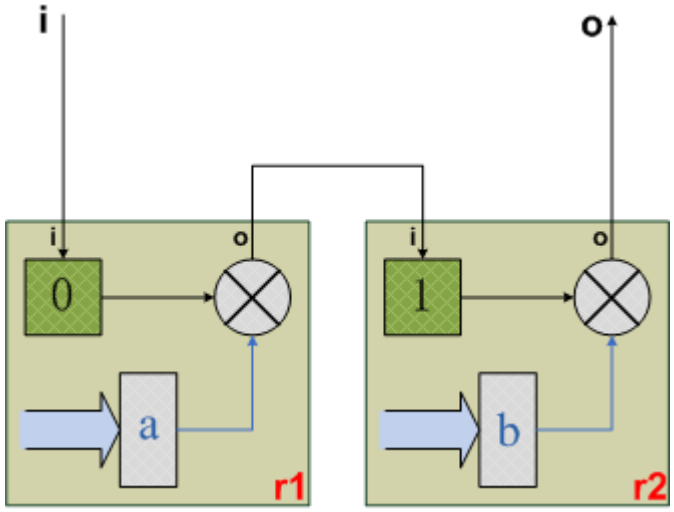


Yang's approach

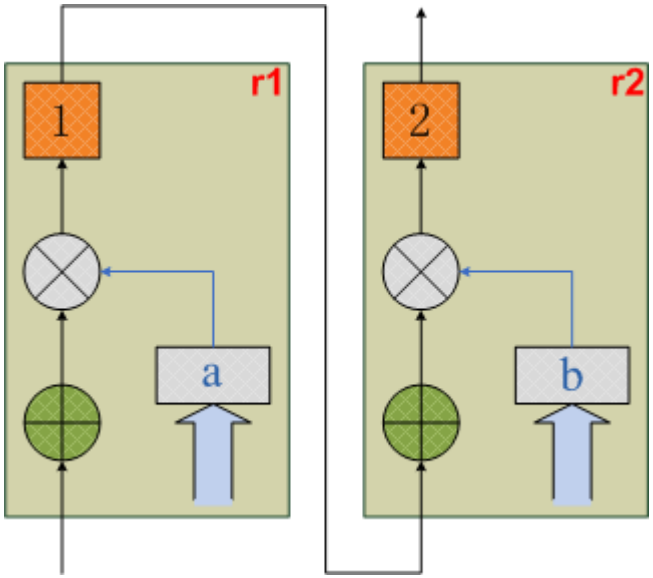
NFA Construction using FPGA

NFA(*ab*)





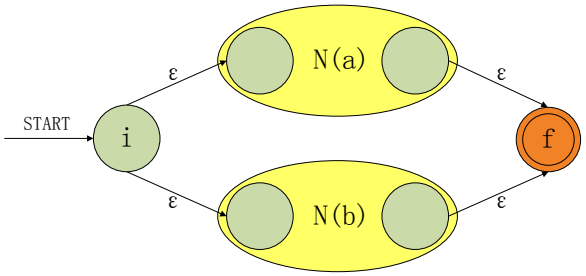
Sidhu's approach

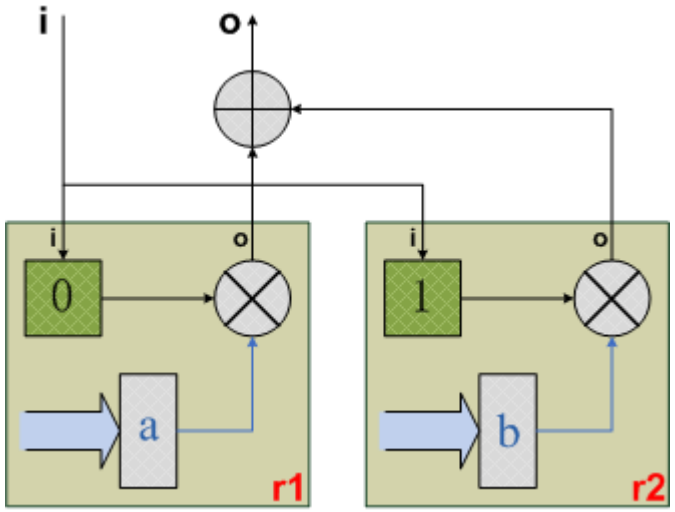


Yang's approach

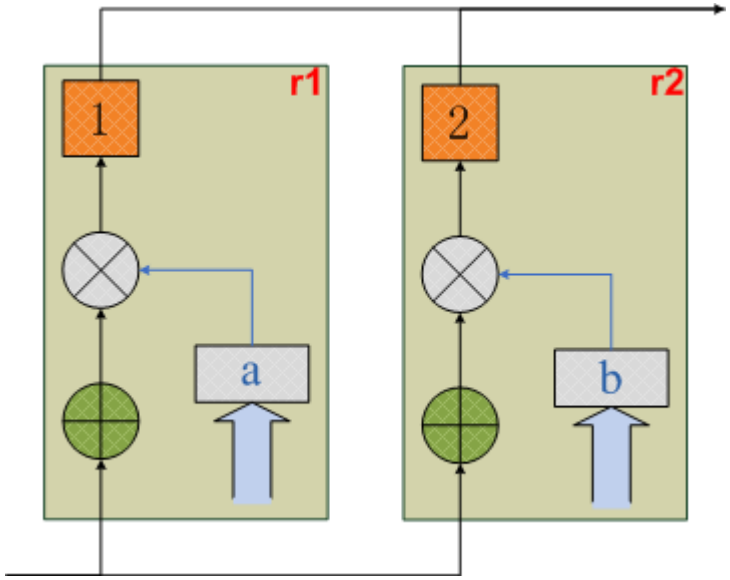
NFA Construction using FPGA

NFA(*a*|*b*)





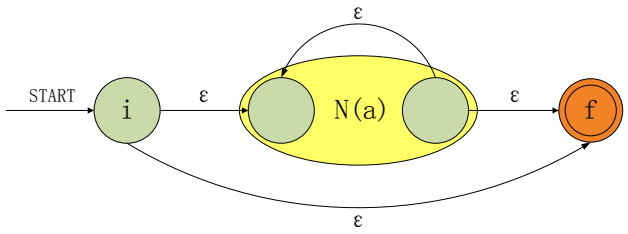
Sidhu's approach

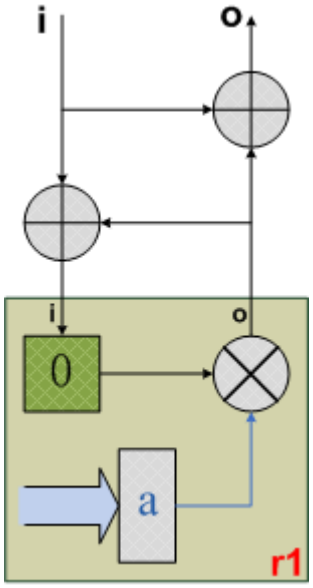


Yang's approach

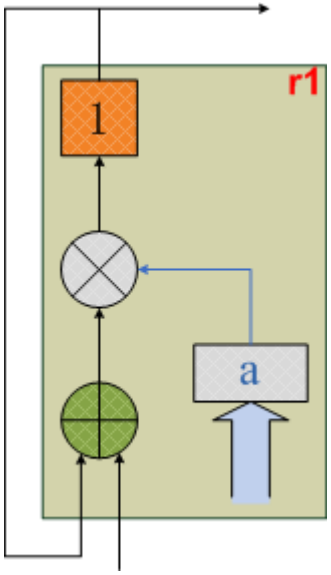
NFA Construction using FPGA

NFA(*a)**



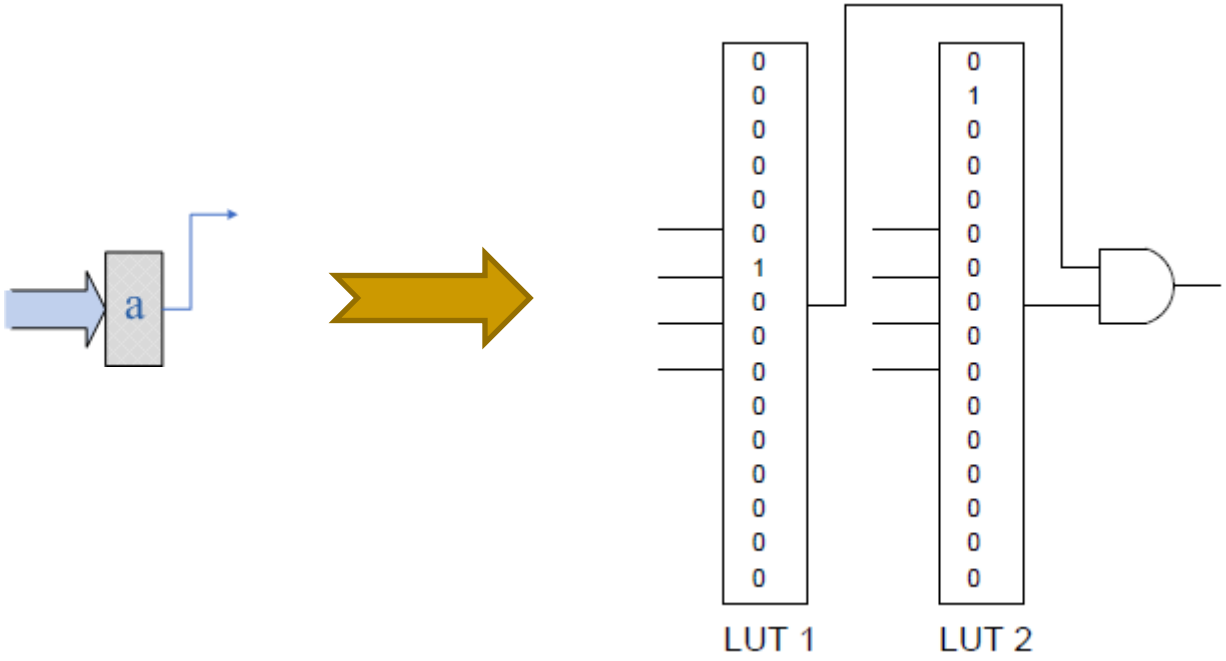


Sidhu's approach

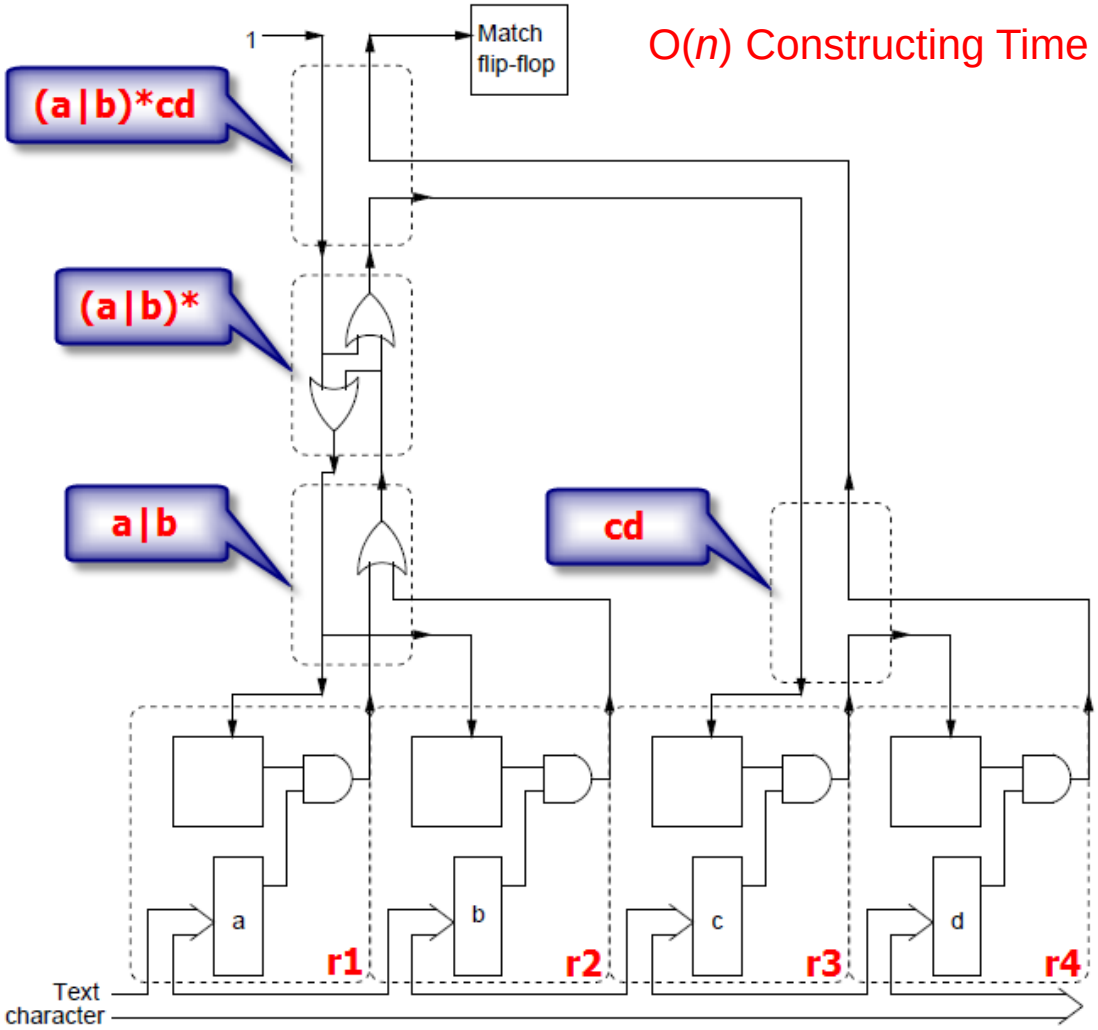


Yang's approach

NFA Construction using FPGA



NFA Construction using FPGA



O(*n*) Constructing Time

```
for(i=0; i<regexp_len; ++i)
{
  switch(regexp[i])
  {
    case char: place_char(regexp[i], &p);
               push(p);

    case     |: place_|(&p);
               pop(&p1);
               route1(p, p1);
               pop(&p2);
               route2(p, p2);
               push(p);

    case     .: place_.(&p);
               pop(&p1);
               route1(p, p1);
               pop(&p2);
               route2(p, p2);
               push(p);

    case     *: place_*(&p);
               pop(&p1);
               route1(p, p1);
               push(p);
  }
}
pop(&p);
route_input_high(p);
route_output_ff(p);
```

NFA Construction using FPGA

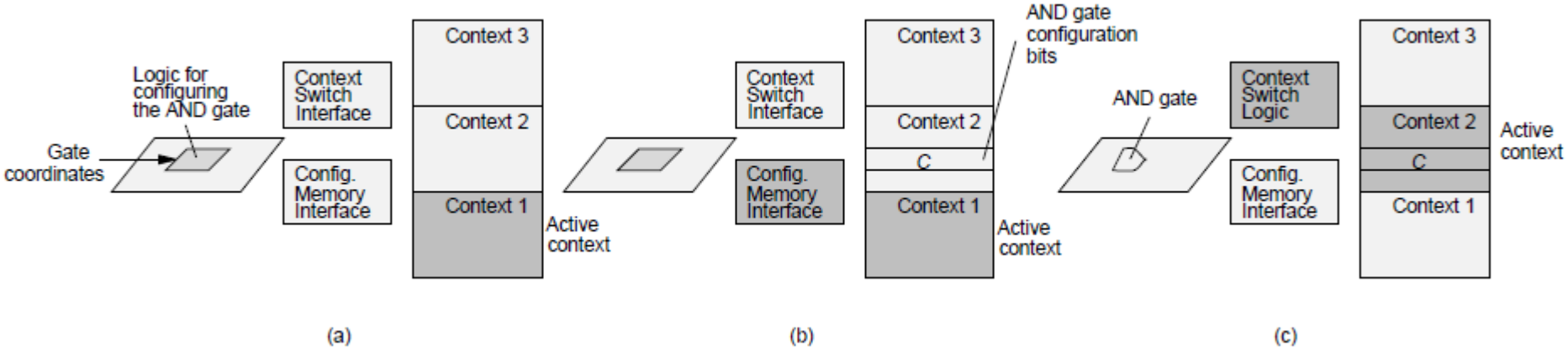
- Yang's approach vs. Sidhu's approach
  - allow latency
    - state register occurs after, rather than before, the character matching
  - easy to construct
    - uniform cell structure for combination with only routing, rather than routing and adding logic
  - not minimize circuit logic at HDL level
    - same transition signal can be produced by different OR gates

NFA Construction using SRGA

- *Self-Reconfiguration* Gate Array
  - **WHAT**
    - device can generate configuration bits at runtime and use them to modify its own configuration
      - single cycle context switching
      - single cycle random access to configuration memory
  - **WHY**
    - reduce NFA mapping time
      - of NFA construction
      - of configuration bits generation for NFA logic
      - of device configuration with generated bits

NFA Construction using SRGA

- *Self-Reconfiguration* Gate Array
  - **HOW**
    - "A Self-Reconfigurable Gate Array Architecture" (*R. Sidhu, FPLA 2000*)

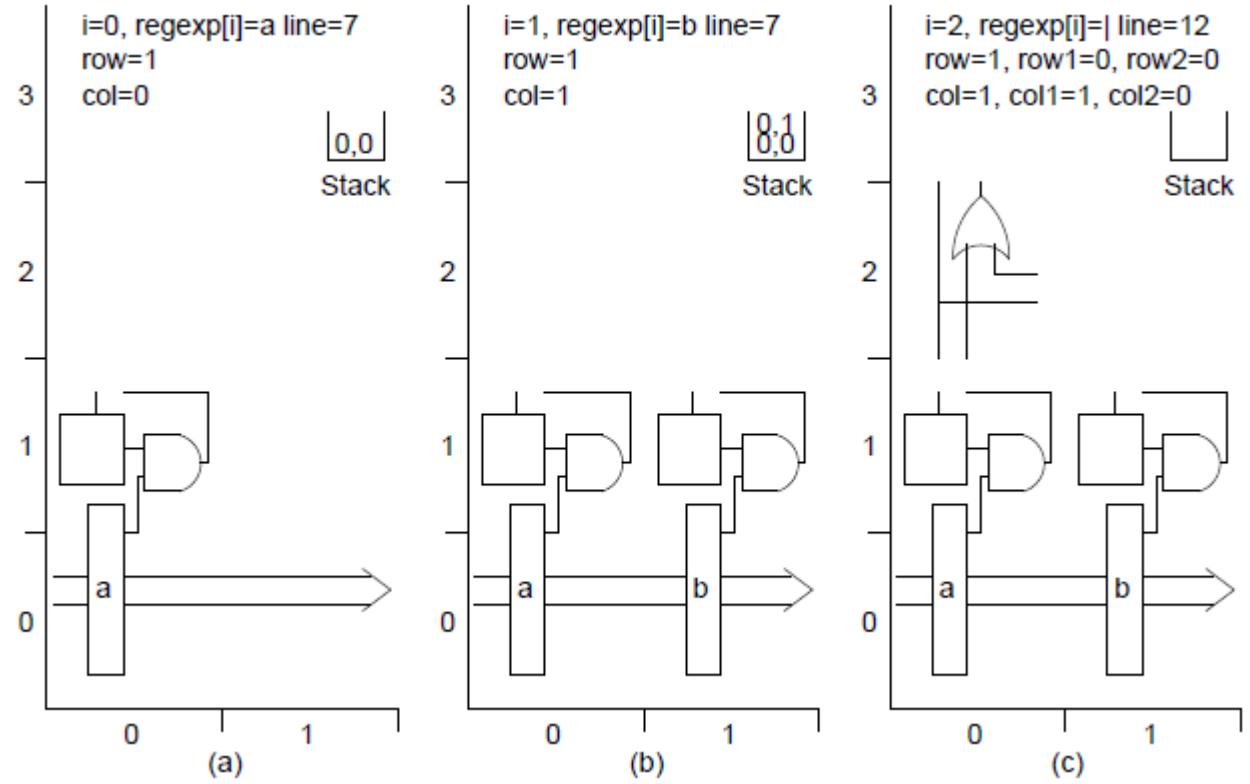


Configure an AND Gate into SRGA

NFA Construction using SRGA

postorder traversal

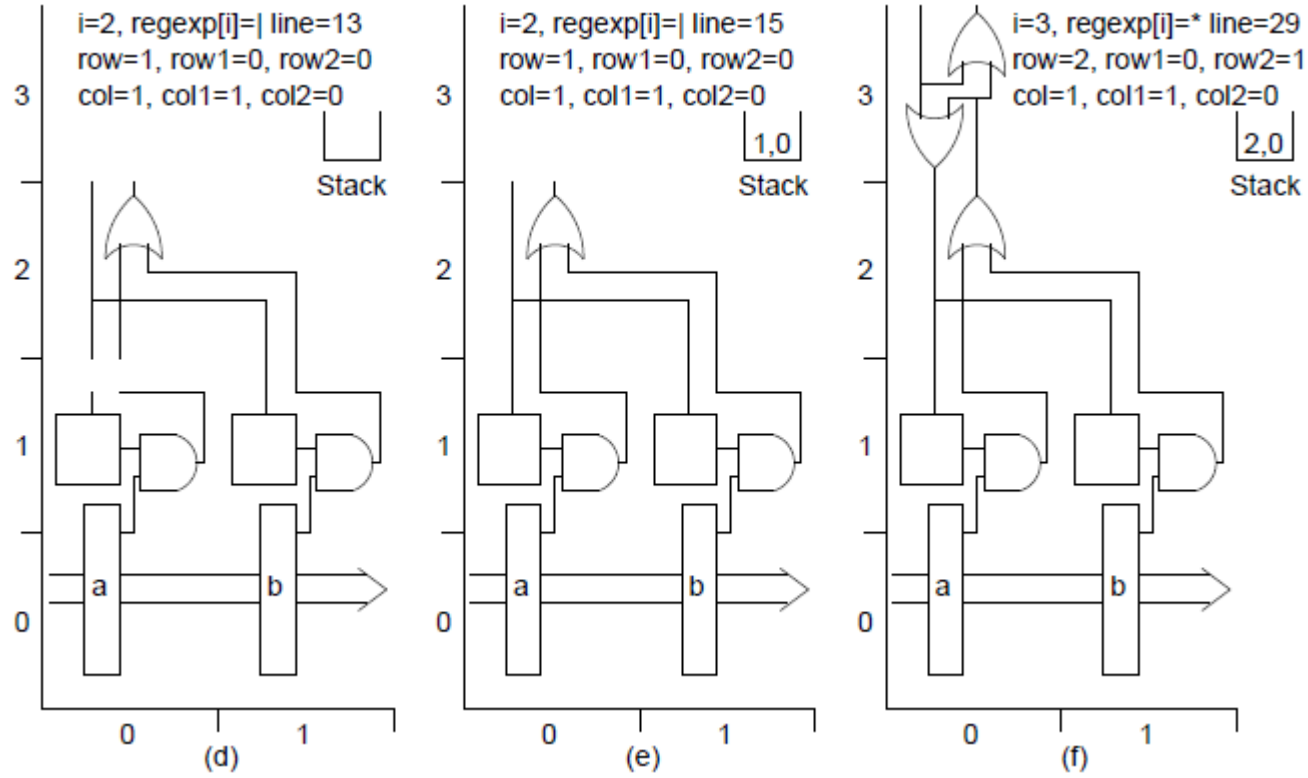
- ab|*cd··




```
 1 row=1; col=0; i=0;                                  [1]
 2 while(i<regexp_len)
 3 {
 4   switch(regexp[i])
 5   {
 6     case char: place_char(regexp[i], col); [46]
 7                push(0, col);               [1]
 8                ++col;                      [0]
 9
10     case    |: pop(&row1, &col1);          [1]
11                pop(&row2, &col2);          [1]
12                place_|(row, col2);         [22]
13                route_row(col2, col1);      [10]
14                route_col(row, row2);       [10]
15                push(row, col2);            [1]
16                ++row;                      [0]
17
19     case    .: pop(&row1, &col1);          [1]
20                pop(&row2, &col2);          [1]
21                place_.(row, col2);         [14]
22                route_row(col2, col1);      [10]
23                route_col(row, row2);       [10]
24                push(row, col2);            [1]
25                ++row;                      [0]
26
27     case    *: pop(&row2, &col2);          [1]
28                place_*(row, col2);         [26]
29                push(row, col2);            [1]
30                ++row;                      [0]
31
32   }
33   ++i;
34 }
35 pop(&row, &col);                                  [1]
36 route_input_high(row, col);                       [3]
37 route_output_ff(row, col);                        [3]
```

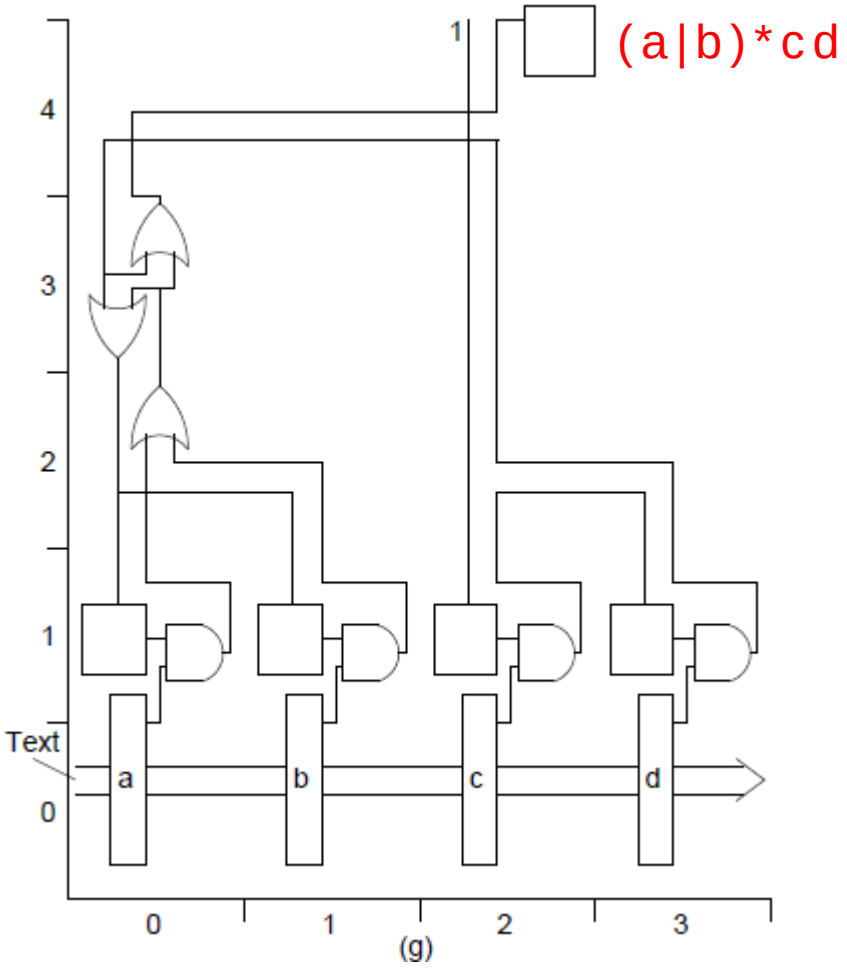
NFA Construction using SRGA

- ab|*cd··




```
 1 row=1; col=0; i=0;                                  [1]
 2 while(i<regexp_len)
 3 {
 4   switch(regexp[i])
 5   {
 6     case char: place_char(regexp[i], col);          [46]
 7                push(0, col);                        [1]
 8                ++col;                               [0]
 9
10     case    |: pop(&row1, &col1);                   [1]
11                pop(&row2, &col2);                   [1]
12                place_|(row, col2);                 [22]
13                route_row(col2, col1);              [10]
14                route_col(row, row2);               [10]
15                push(row, col2);                     [1]
16                ++row;                               [0]
17
19     case    .: pop(&row1, &col1);                   [1]
20                pop(&row2, &col2);                   [1]
21                place_.(row, col2);                 [14]
22                route_row(col2, col1);              [10]
23                route_col(row, row2);               [10]
24                push(row, col2);                     [1]
25                ++row;                               [0]
26
27     case    *: pop(&row2, &col2);                   [1]
28                place_*(row, col2);                 [26]
29                push(row, col2);                     [1]
30                ++row;                               [0]
31
32   }
33   ++i;
34 }
35 pop(&row, &col);                                    [1]
36 route_input_high(row, col);                        [3]
37 route_output_ff(row, col);                         [3]
```

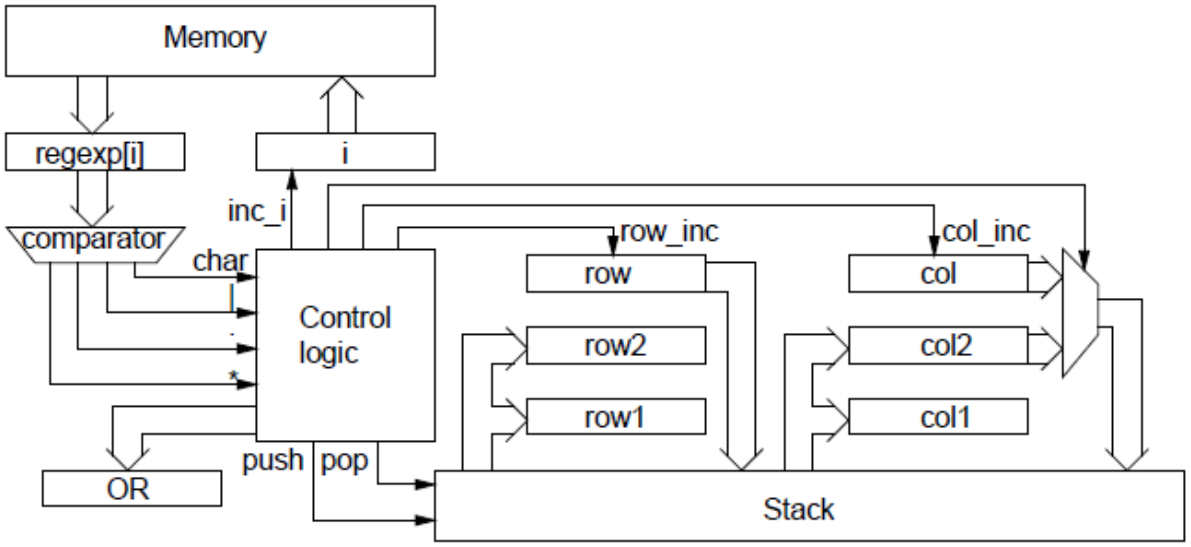
NFA Construction using SRGA




```
 1 row=1; col=0; i=0;                                [1]
 2 while(i<regexp_len)
 3 {
 4   switch(regexp[i])
 5   {
 6     case char: place_char(regexp[i], col); [46]
 7                push(0, col);               [1]
 8                ++col;                      [0]
 9
10     case    |: pop(&row1, &col1);          [1]
11                pop(&row2, &col2);          [1]
12                place_|(row, col2);         [22]
13                route_row(col2, col1);      [10]
14                route_col(row, row2);       [10]
15                push(row, col2);            [1]
16                ++row;                      [0]
17
19     case    .: pop(&row1, &col1);          [1]
20                pop(&row2, &col2);          [1]
21                place_.(row, col2);         [14]
22                route_row(col2, col1);      [10]
23                route_col(row, row2);       [10]
24                push(row, col2);            [1]
25                ++row;                      [0]
26
27     case    *: pop(&row2, &col2);          [1]
28                place_*(row, col2);         [26]
29                push(row, col2);            [1]
30                ++row;                      [0]
31
32   }
33   ++i;
34 }
35 pop(&row, &col);                            [1]
36 route_input_high(row, col);                 [3]
37 route_output_ff(row, col);                  [3]
```

NFA Construction using SRGA

- Datapath and Control Logic
- O(n^2) area




```
 1 row=1; col=0; i=0;                                  [1]
 2 while(i<regexp_len)
 3 {
 4   switch(regexp[i])
 5   {
 6     case char: place_char(regexp[i], col);          [46]
 7                push(0, col);                        [1]
 8                ++col;                               [0]
 9
10     case    |: pop(&row1, &col1);                   [1]
11                pop(&row2, &col2);                   [1]
12                place_|(row, col2);                 [22]
13                route_row(col2, col1);              [10]
14                route_col(row, row2);               [10]
15                push(row, col2);                     [1]
16                ++row;                               [0]
17
19     case    .: pop(&row1, &col1);                   [1]
20                pop(&row2, &col2);                   [1]
21                place_.(row, col2);                 [14]
22                route_row(col2, col1);              [10]
23                route_col(row, row2);               [10]
24                push(row, col2);                     [1]
25                ++row;                               [0]
26
27     case    *: pop(&row2, &col2);                   [1]
28                place_*(row, col2);                 [26]
29                push(row, col2);                     [1]
30                ++row;                               [0]
31
32   }
33   ++i;
34 }
35 pop(&row, &col);                                    [1]
36 route_input_high(row, col);                        [3]
37 route_output_ff(row, col);                         [3]
```

Performance Evaluation

- Test RE
  - (a|b)*a(a|b){k}
  - k varies from 8 to 19
- Test platform
  - PC
    - 800MHz Pentium III Xeon processor, 2GB RAM
    - GNU grep v2.4, Red Hat 6.2
  - FPGA
    - Virtex XCV100 (450MHz Pentium III, 20 x 30 CLBs)
    - proposed approach

Performance Evaluation

- Text search and DFA construction
  - worst case and best case

k	Text file size (bytes)	CPU time	Maximum memory
8	2560	0.01 s	1 MB
9	5632	0.05 s	1 MB
10	12288	0.15 s	1.9 MB
11	26624	0.50 s	2.2 MB
12	57344	2.22 s	3.0 MB
13	122880	16.11 s	4.4 MB
14	262144	82.88 s	7.5 MB
15	557056	345.33 s	13 MB
16	1179648	1383.55 s	26 MB
17	2490368	5499.60 s	54 MB
18	5242880	21900.36 s	111 MB
19	11010048	87309.38 s	229 MB

k	Text file size (bytes)	CPU time	Maximum memory	Time per character
8	2560	0.00 s	580 KB	–
9	5632	0.00 s	580 KB	–
10	12288	0.00 s	580 KB	–
11	26624	0.00 s	580 KB	–
12	57344	0.00 s	580 KB	–
13	122880	0.005 s	580 KB	–
14	262144	0.01 s	580 KB	–
15	557056	0.03 s	580 KB	53.86 ns
16	1179648	0.04 s	580 KB	33.91 ns
17	2490368	0.08 s	580 KB	32.12 ns
18	5242880	0.17 s	580 KB	32.42 ns
19	11010048	0.34 s	580 KB	30.88

Performance Evaluation

- Software performance

k	DFA size	Construction time
8	420 KB	0.01 s
9	420 KB	0.05 s
10	1.32 MB	0.15 s
11	1.62 MB	0.50 s
12	2.42 MB	2.22 s
13	3.82 MB	16.11 s
14	6.42 MB	82.87 s
15	12.42 MB	345.30 s
16	25.42 MB	1383.51 s
17	53.42 MB	5499.52 s
18	110.42 MB	21900.19 s
19	228.42 MB	87309.04 s

Performance Evaluation

- FPGA performance

k	Configuration bit generation	FPGA configuration	NFA construction
8	20 ms	1 ms	21 ms
9	38 ms	1 ms	39 ms
10	31 ms	1 ms	32 ms
11	33 ms	1 ms	34 ms
12	30 ms	1 ms	31 ms
13	28 ms	1 ms	29 ms
14	32 ms	1 ms	33 ms
15	33 ms	1 ms	34 ms
16	33 ms	1 ms	34 ms
17	36 ms	1 ms	37 ms
18	36 ms	1 ms	37 ms
19	30 ms	1 ms	31 ms
28	38 ms	1 ms	39 ms

k	NFA area	Construction time	Time per text character
8	10 × 7 CLBs	21 ms	10.70 ns
9	11 × 8 CLBs	39 ms	11.68 ns
10	12 × 8 CLBs	32 ms	11.99 ns
11	13 × 9 CLBs	34 ms	12.17 ns
12	14 × 9 CLBs	31 ms	12.69 ns
13	15 × 10 CLBs	29 ms	12.32 ns
14	16 × 10 CLBs	33 ms	12.70 ns
15	17 × 11 CLBs	34 ms	11.89 ns
16	18 × 11 CLBs	34 ms	12.55 ns
17	19 × 12 CLBs	37 ms	13.06 ns
18	20 × 12 CLBs	37 ms	13.24 ns
19	21 × 13 CLBs	31 ms	14.98 ns
28	30 × 16 CLBs	39 ms	17.42 ns

Performance Evaluation

- SRGA implementation
  - conservatively estimation
  - constructing time
    - 259+176k clock cycles
    - 100ns clock period

19=2+1+2*8

k	NFA size	NFA construction time
8	19 × 22	166.7 μs
9	21 × 24	184.3 μs
10	23 × 26	201.9 μs
11	25 × 28	219.5 μs
12	27 × 30	237.1 μs
13	29 × 32	254.7 μs
14	31 × 34	272.3 μs
15	33 × 36	289.9 μs
16	35 × 38	307.5 μs
17	37 × 40	325.1 μs
18	39 × 42	342.7 μs
19	41 × 44	360.3 μs

Conclusion

- Show how to efficiently perform REM using FPGAs
  - propose the approach
  - both efficient in time and space
- Show how NFA construction can be performed very quickly using self-reconfiguration
  - propose the approach
  - dramatically reduce the NFA constructing time
- Overcome exponential blowup of DFA in time and space

Thanks for your attention!