Pruning-based Trace Signal Selection Algorithm

Kang Zhao, Jinian Bian

EDA Lab, Dept. Computer Science & Technology
Tsinghua University, Beijing 100084, China

Jan 28, 2011

Contents

- Introduction
- Algorithm
- Experiments
- Conclusion

Contents

- Introduction
- Algorithm
- Experiments
- Conclusion

Background

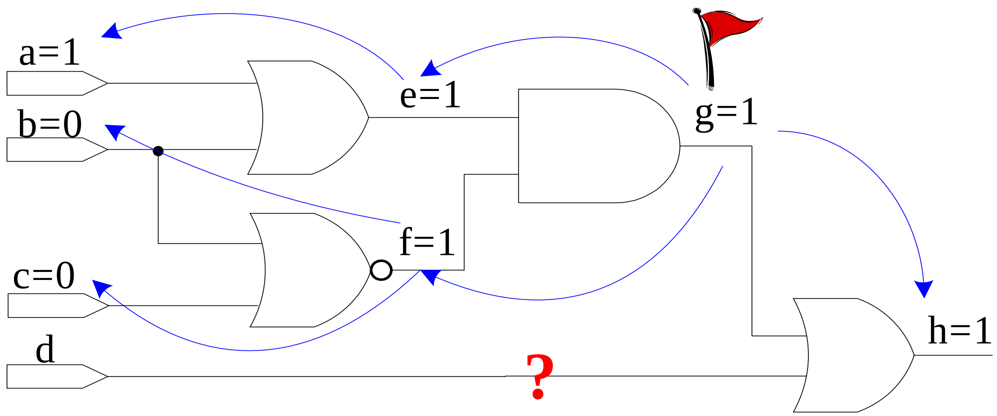
- Silicon debug
  - Pre-silicon validation: functional and timing errors
  - Post-silicon validation: functional and electronic errors
- Motivation
  - Identify the bugs effectively
  - Know each state for each signal in the circuits
  - Enhance the visibility of the circuits

General Method

- Monitor and trace some internal signals
  - The values of those signals are stored in the trace buffer
  - Estimate the values and states of other signals based on an analysis on the trace signals
- Advantage & Limitation
  - Advantage: need not large buffer to store many signals
  - Advantage: low cost and high efficiency
  - Limitation: the number of trace signals are limited due to the trace buffer
- Focus: how to select a limited set of trace signals

Problem

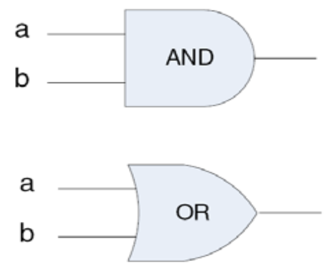
- Problem
  - Given the circuit with *n* FFs, find a subset of trace signals not exceeding *k*, so that the restoration ratio *r* is maximum
- Focus
  - How to select limited trace signals, so we can view more states in the circuit



Related works

- Two restorations
  - Forward: F0/F1
  - Backward: B0/B1
  - Restoration ratio: $(N_{trace}+N_{restore})/N_{restore}$
- Method
  - Sum up all restorations for each FF
  - Select the biggest one as the trace signal
  - The restoration ratio is used to measure the quality of the final results

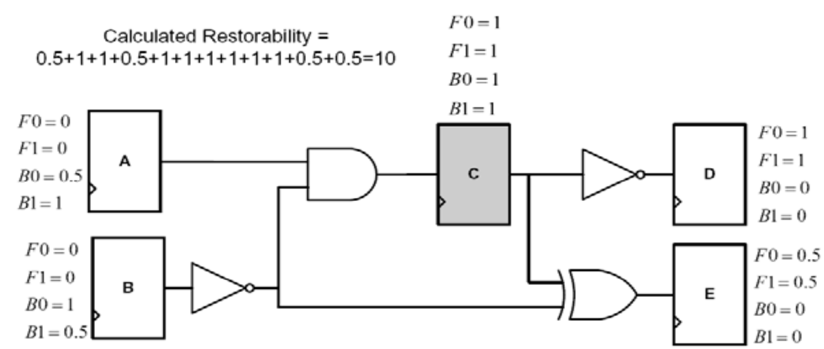
H. F. Ko and N. Nicolici. "Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation". Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), 2008.



$$F0(c) = \max\{F0(a), F0(b)\}$$
$$F1(c) = (F1(a) + F1(b))/2$$
$$B0(a) = (\max\{B0(c)\} + F1(b))/2$$
$$B1(a) = \max\{B1(c)\}$$

$$F0(c) = (F0(a) + F0(b))/2$$
$$F1(c) = \max\{F1(a), F1(b)\}$$
$$B0(a) = \max\{B0(c)\}$$
$$B1(a) = (\max\{B1(c)\} + F0(b))/2$$

F1/F0 --- the probability of restoring data 1/0 of a node through forward propagation
B1/B0 --- the probability of restoring data 1/0 of a node through backward justification



(a) Circuit-under-debug

Clock Cycle

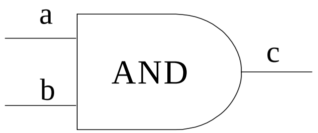
	0	1	2	3	4
A	1	1	X	X	X
B	0	0	X	X	X
C	0	1	1	0	X
D	X	1	0	0	1
E	X	1	0	X	X

(b) Restored data in flip flops

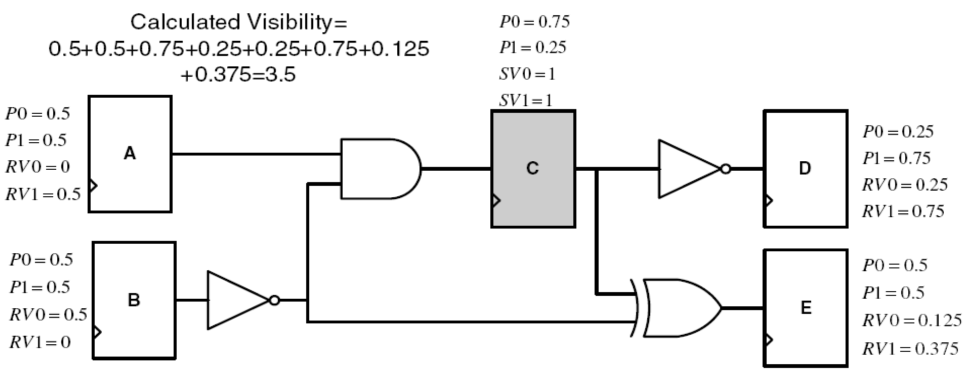
$$Restoration\ Ratio = \frac{14}{4} = 3.5$$

Related works

- Accurate restorations
  - *RV* is used to replace the *F* and *B*
  - $RV = P*(F+B-F*B)$
  - P means the probability in the functional mode
- Method
  - Sum up all restorations
  - Select the biggest one as the trace
- Advantage
  - The conditional probability can consider the two directions together
  - *P* can reduce the additional effect of the initial values



RV0(c) = [V0(a)+V0(b)-V0(a)*V0(b)]/[1-P1(a)*P1(b)]
RV1(c) = [V1(a)*V1(b)]/[P1(a)*P1(b)]
RV0(a) = R0(c)*V1(b)
RV1(a) = V1(c)/P1(a)



X. Liu and Q. Xu. “Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation”. Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), 2009.

Limitations

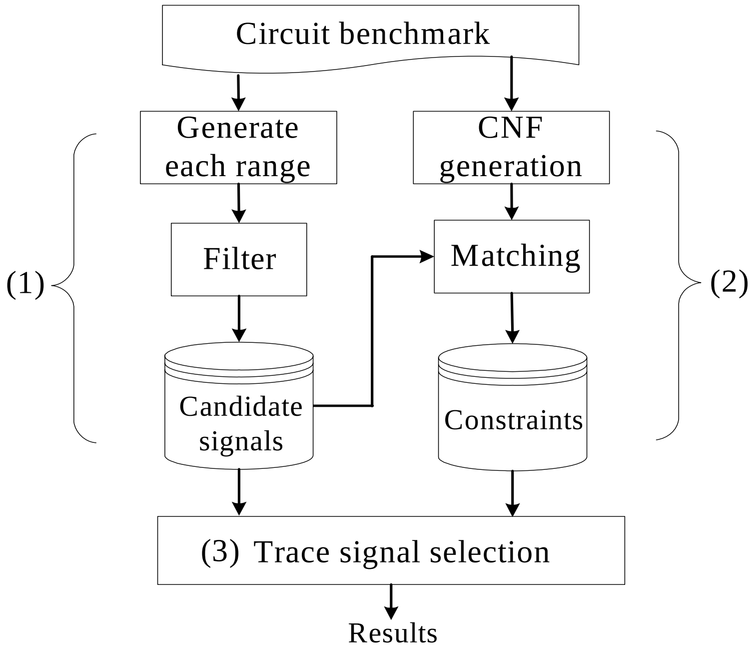
- Greedy trace signal selection
  - Fist select the signal ***a*** which can get the biggest restoration ratio
  - Then, fix the signal ***a***; select another signal ***b*** which can get another biggest restoration ratio
  - …
- Limitation
  - The signal a may be not the best selection although it is the best choice for the first iteration
  - The previous choice will affect the following iterations
  - The greedy strategy can only get a better result

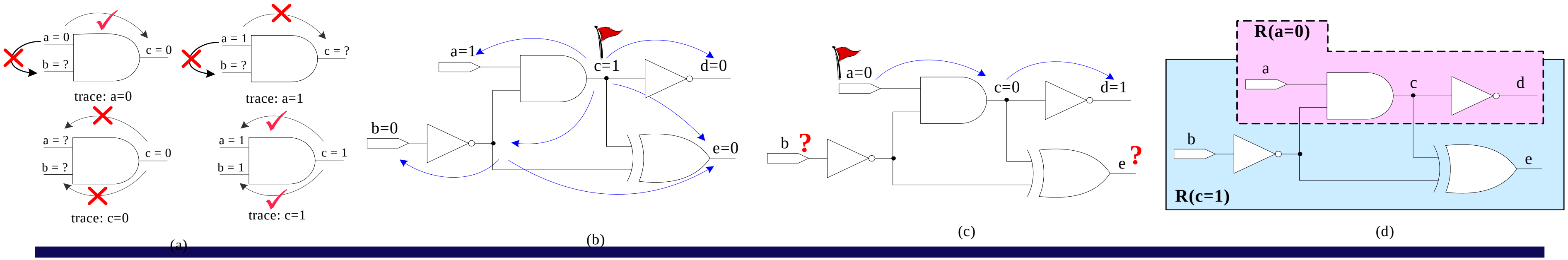
Contents

- Introduction
- Algorithm
- Experiments
- Conclusion

Strategy

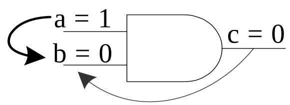
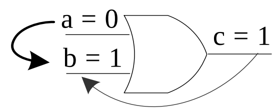
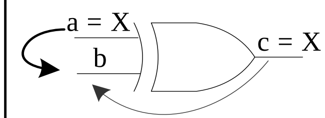
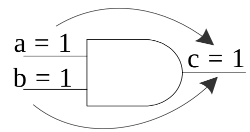
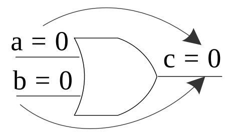
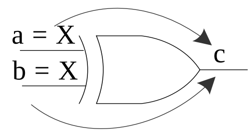
- Three steps
  - Compute the restoration range for each signal
  - Generate the constraints based on the candidate trace signals
  - Compare the effects based on different combinations, find the better one
- Restoration range
  - R(x)={y | y can be restored by x}





Candidate generation – restoration range

- **Theorem:** Let R(a) and R(b) are two different ranges. If there is no conflict, R(a, b)⩾R(a)∪R(b)
  - E.g. if *a*=1 and *c*=0 for the AND gate, the input *b* will be restored
  - For each gate, if the values of other signals satisfy the constraints in the figure below, the value of last signal will be restored

AND	OR	XOR
 trace: a=1, c=0	 trace: a=0, c=1	 trace: a=X, c=X
 trace: a=1, b=1	 trace: a=0, b=0	 trace: a=X, b=X
Input=1 Output=0	Input=0 Output=1	Input=X Output=X

Candidate generation – range unit

- ***Add* function:** is used to compute the restoration range, especially for the unit of ranges
  - Range(S0+S1)=S0+Add(S0, S1, D)
  - D denotes the number of invisible ports
  - Once a port is restored, D=D-1 and this process will stop until D=1

Add(Set S_0, Set S_1, Degree $\&D$)

$A \Leftarrow \emptyset$; $Queue \Leftarrow S_1 - S_0$;
while($Queue$ is not empty){
  $x \Leftarrow$ the top element of $Queue$;
  Remove the top element of $Queue$;
  for(each gate i connecting with the node x){
    if(i is not dead and x is potential for i){
      $D[i] \Leftarrow D[i] - 1$;
      if($D[i] = 1$){
        $A \Leftarrow A \cup$R(y); //y is the last port of i
        Add R(y) to the end of $Queue$;
        Sign the gate i to be dead;
      }//end if
    }//end if
  }//end for
}//end while
return $A \cup S_1 - S_0$;

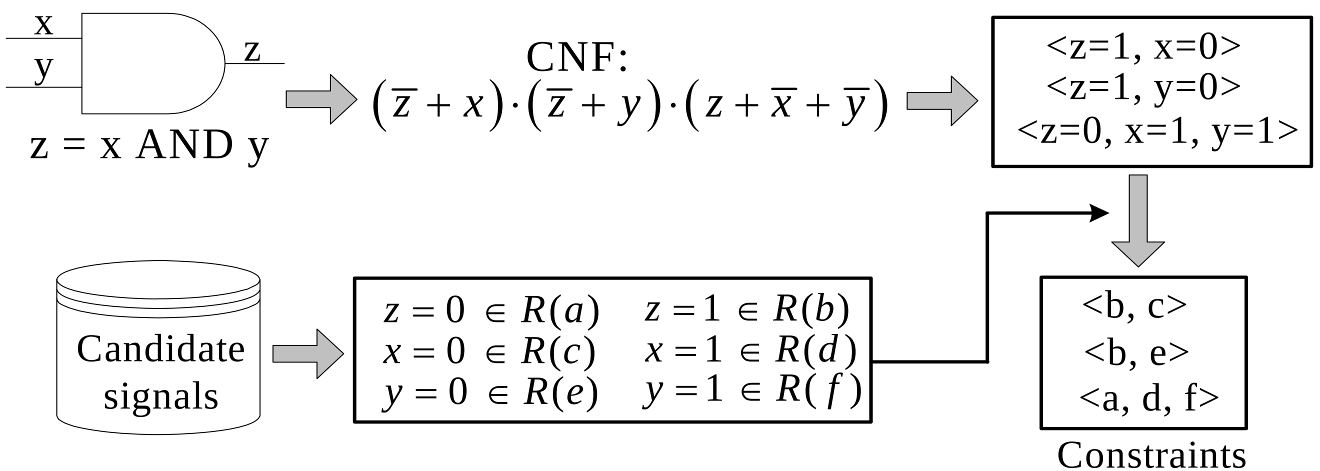
Candidate generation – filter

- **Theorem:** let *a* and *b* are two different signals and there is no NOT gate between them. Then, we can get:
  - $b \in R(a) \rightarrow a \notin R(b)$
  - $b \in R(a) \rightarrow R(b) \subset R(a)$
- Filter
  - If $b \in \mathrm{R}(a)$, $\mathrm{R}(a)$ must be larger or equal to $\mathrm{R}(b)$, and *b* can be replaced by *a*
  - If there is NOT gate between *a* and *b*, one signal can be removed

```
Filter( )
C ⇐all signals in the circuit;
for(each two signals i, j in the circuit)
    if(i, j are connected with a NOT gate)
        C ⇐ C − {R(j)};
    else if(j ∈R(i))
        C ⇐ C − {R(j)};
    end if
end for
return C;
```

Constraints

- Conjunctive normal form (CNF)
  - Each Boolean circuit can be repressed by CNF
  - "*z*=*x* AND *y*" can be repressed by: $(\overline{z}+x)(\overline{z}+y)(z+\overline{x}+\overline{y})$
- CNF → constraints
  - The constraints are generated via negative operation on each item of CNF
  - E.g. <*z*=1, *x*=0>, <*z*=1, *y*=0>, and <*z*=0, *x*=1, *y*=1>



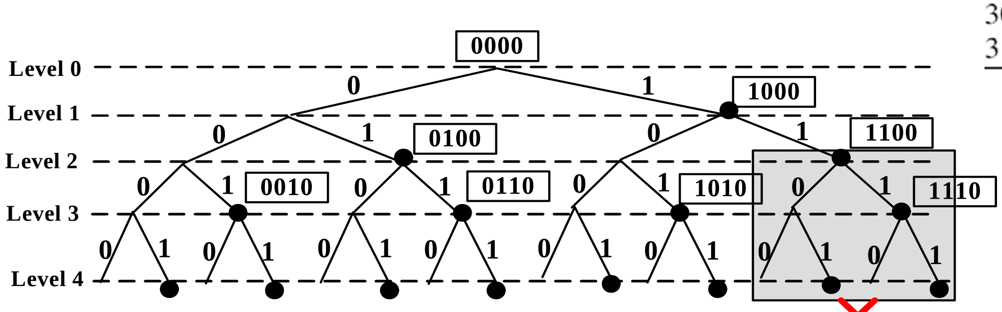
The flow of constraints generation process

Selection Algorithm

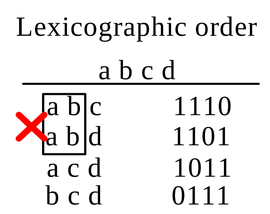
Idea: enumerate the combination with the lexicographic order. If violate the constraint, the subtree will be omitted

- (1) initialization, line 1-7
- (2) different combinations are enumerated with the lexicographic order, line 8-12
- (3) once the point exceeds the scope, the exploration stop and pop the stacks, line 13-22
- (4) otherwise, next signal will be put into the stacks, line 23-29

```
Selection(Set C_trace, Size n, Constraints)
1.  V ⇐ Schedule(C_trace, Constraints);
2.  Generate a new array A, and its length is n;
3.  A.push_back(V.begin); p ⇐1; //p: a pointer to V
4.  Generate two empty arrays X and Y;
5.  for(each gate i in the circuit)
6.     D[i] ⇐the total port number of i; //max degree
7.  for(each i ∈ A) R_push(A[i], X, Y, D);
8.  while(A is not empty And V is not full){
9.     if(A is full and its range is larger than ever){
10.       Record the combination in result;
11.    }
12.    if(A is full){A.pop_back(); R_pop(X, Y, D);}
13.    else if(V is full){
14.       A.pop_back();
15.       if(no violation during previous iteration){
16.          A.pop_back(); R_pop(X, Y, D);
17.       }
18.       if(A is not empty){
19.          p ⇐ the next position in V for A.end;
20.          A.pop_back(); R_pop(X, Y, D);
21.       }//if
22.    }//else if
23.    else{
24.       A.push_back(V[p]);
25.       if(A violates the constraints){
26.          A.pop_back();    ++p;    continue;
27.       }
28.       else{R_push(V[p], X, Y, D);    ++p;}
29.    }//else
30. }//end while
31. return the better result;
```



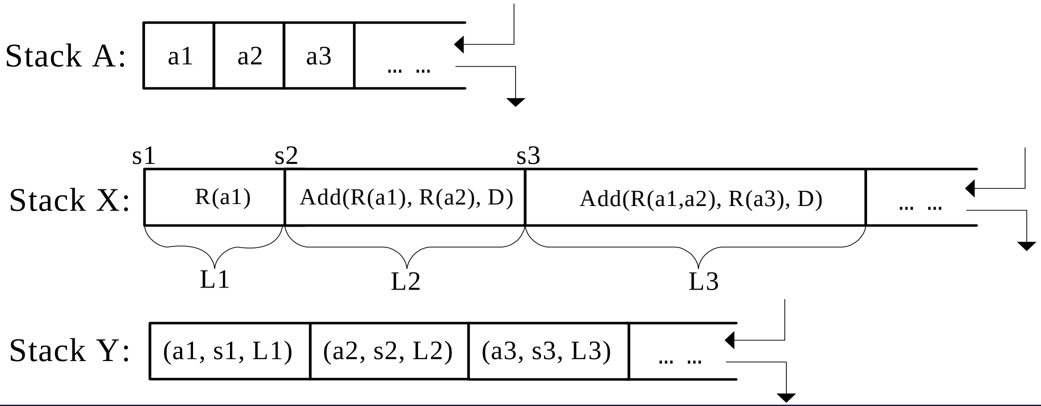
(a)



(b)

Two features

- Pruning
  - Each level in the binary tree represents the current signal is selected or not
  - Since the enumeration uses the lexicographic order, once the combination violates the constraints, there will be no necessary to explore its subtree
- Stack structure
  - Since the neighboring candidates have the similar segment, the intermediate results can be reused. We used the stack structure to settle this issue
  - A: store the candidate signals; X: stores the restoration range of current signal; Y: records the additive length in X for each change



Contents

- Introduction
- Algorithm
- Experiments
- Conclusion

Experimental Setup

- Setup
  - Program with C++ on a Linux machine
  - Intel Xeon 3GHz CPU and 4GB memory
  - OS: Red Hat Enterprise Linux AS release 3
  - Gcc 3.4.6 with option -03
- Front-end
  - Benchmarks: some circuits from ISCAS'89
  - We have implemented a random 0/1 generator, using C++ function *rand()*
  - Trace buffer: 8*4k, 16*4k
- Back-end
  - Execution time: C++ function *clock*() and the macro CLOCKS_PER_SEC
  - A simulator computes the restoration ratio *r*

Results

- Comparison
  - Ko's selection algorithm in DATE'2008, referred as "gradual"; Xiao's algorithm in DATE'2009, referred as "greedy"; Ko's similar method in TCAD'09, referred as "cover".

The results for the buffer 8*4k

		gradual[6]			**greedy**[4]			**cover**[7]			**pruning**(proposed)			$\triangle r$ (×)		
name	**#DFF**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**[6]**	**[4]**	**[7]**
s382	21	51987	2.625	2.90	55977	2.99	2.64	59986	3.49	0.42	40356	11.09	1.51	4.22	3.71	3.18
s641	19	43997	2.38	12.01	56107	4.51	7.96	50995	7.37	3.73	36321	10.08	1.71	4.24	2.24	1.37
s1196	18	39664	2.24	6.18	39713	2.24	6.78	52164	5.34	1.24	33692	9.42	8.50	4.21	4.21	1.76
s1238	18	39664	2.24	5.88	39713	2.24	4.15	52164	5.34	1.10	33692	9.42	9.13	4.21	4.21	1.76
s1494	6	0	1.00	5.21	0	1.00	7.75	12000	2.00	3.01	10090	3.52	12.66	3.52	3.52	1.76
s1423	74	262087	9.19	226.22	266530	14.33	152.20	262081	9.19	29.07	144415	37.10	105.32	4.04	2.59	4.04

The results for the buffer 16*4k

		gradual[6]			**greedy**[4]			**cover**[7]			**pruning**(proposed)			$\triangle r$ (×)		
name	**#DFF**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**#RS**	r	**time(s)**	**[6]**	**[4]**	**[7]**
s382	21	19997	1.31	4.31	55977	2.99	2.37	59986	3.49	0.39	40356	11.09	1.01	8.47	3.71	3.18
s641	19	11999	1.19	15.99	56107	4.51	7.04	50995	7.37	2.25	36321	10.08	1.13	8.47	2.24	1.37
s1196	18	7997	1.12	7.22	7999	1.12	8.39	52164	5.35	0.96	33692	9.42	7.08	8.41	8.41	1.76
s1238	18	7997	1.12	9.72	7996	1.12	7.45	52164	5.34	1.10	33692	9.42	13.73	8.41	8.41	1.76
s1423	74	230318	4.59	357.53	246605	21.55	90.88	258111	8.17	27.29	144415	37.10	80.39	8.08	1.72	4.54

Results

- Two tables
  - The results when 8 nodes and 16 nodes are selected as trace signals
  - “#TN” means the number of selected trace signals; “#RS” is the number of restoration states
  - “r” is the restoration ratio and “time” is execution time
- Results
  - The proposed algorithm can bring a higher restoration ratio than the previous methods, about 2 - 4 times
  - When the number of the trace nodes increases, our improvement on the ratio may be higher
- Limitation
  - Although pruning technique is used to reduce the huge search space, the executing time will still rise rapidly when the circuit size increases

Contents

- Introduction
- Algorithm
- Experiments
- Conclusion

Conclusion

- Conclusion
  - This paper proposes a pruning-based trace signal selection algorithm to improve the restoration ratio for the data acquisition in the post-silicon validation
  - The algorithm generate the visible restoration range for each FF, and explore different enumerations to find the better one. To accelerate the efficiency, it proposes the CNF-based constraints for the exhaustive pruning
- Future work
  - How to enhance the efficiency during the trace signal selection will be our emphasis

Thanks