Vol.2021-ARC-243 No.31
Vol.2021-SLDM-193 No.31
2021/1/26

Logic synthesis by looking for good input to gate

Zongjin Zhou[1] Masahiro Fujita[1]

Abstract: *In this paper, we mainly discuss two logic optimization methods. The first one is from logic debugging method called signal selection. This method searches for inputs to a gate instead of searching the new function for some internal node to complete the debugging and optimization tasks. The other one is Set of Pairs of Functions to be Distinguished (SPFD). Base on the flexibility of circuit, we can use this method to optimize the circuit without changing logic function. In our research, we have found that these two methods are mathematically equivalent, but the ways to compute are different, and depending on how to utilize them, more appropriate method can be used in logic optimization. Then we have implemented them and conducted experiments for the comparison between the two method and also experiments in order to evaluate the powerfulness of the two methods*

Keywords: Logic synthesis, Signal Selection, SPFD, equivalent

1. Introduction

Logic Synthesis is the process of transforming a set of Boolean functions from the abstract specification into a network of logic gates in a particular technology. The task of logic synthesis is to transform one presentation of a logic circuit into another acquiring some merits.

In this paper, we discuss two logic optimization techniques which have been developed in different contexts. The first one is from a logic debugging technique called Signal Selection [2]. Debugging logic circuit involves searching for a new logic function for the buggy portions of the design such that the circuit behaves the same as the correct circuit which call specification. It mainly consists of two main procedures: finding the appropriate candidate locations or internal signals for correction and determining the new functions for those candidate signals. Compared to the debug techniques which search for the signals and then find correct logic function, this approach focuses on finding the input to the new function for the selected internal nodes, instead of function itself. There are several advantages in this approach. First it can search for signals which are candidate fanins of the gate even if they are far from the gate. Second is that it can handle much more search space because it searches for the inputs to signal instead of searching signal itself. When the other debugging method searches for signal and determine the correction function, they should handle 2^v variables where v is the number of candidate signals. In some situation, v could be 16 or 32 then 2^v would become very large value and hard to handle of. In Signal Selection, v could reach 100 or more. The definition of Signal Selection will be discussed later.

The other is a logic synthesis method that uses Set of Pairs of Functions to be Distinguished (SPFD) [1]. As we discussed above, the task of logic synthesis is to transform logic circuit into another one with some merits. The basis to complete the transform is “flexibility” derived from various kinds of don’t cares and others. The traditional synthesis methods like compatible observability don’t case (OCDC) and compatible set of permissible functions (CSPFs) can provide enough flexibility to effectively transform the logic circuit. But SPFD can extends the flexibility provided by previous approaches through the target signal must distinguish primary input values.

In this paper, we introduce the definition of these two approaches and discuss the equivalence between them. Then we implement and evaluate them with the same benchmark circuits and show the difference. We also use SPFD to optimize some circuits and show the result.

The rest of this paper is organized as follows. Section 2 and Section 3 overview the definitions and concepts of these two approaches. In section 4, we discuss the equivalence of these two approaches. Section 5 shows the experimental results and section 6 concludes the paper. Section 7 describes the future works.

2. Signal Selection

In this section we give the definition and concepts about the Signal Selection method. This debugging method focuses on combination circuits or the combinational parts of the sequential circuits.

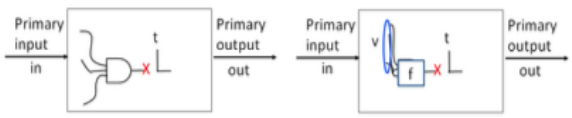


Figure 1. Correctable with t a function of inputs/internal signals. [2]

1 The University of Tokyo, Bunkyo, Tokyo 113-0032, Japan

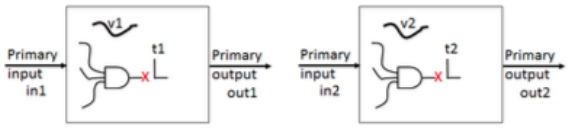


Figure 2. Correctable with t a function of in1 case and in2 case. [2]

Definition 1. Given a buggy circuit **M**, a correctable circuit **S**, primary input **in**, and internal signal of M: **t**. **M** is correct with **t** if

$$\forall in, \exists t. M(t, in) = S(in)$$

From Figure 1, we know that internal signal $t = f(v)$, where f is the logic function mapping internal variable **v** to **t**. We can derive the equation into

$$\forall in, \exists t, \exists v, \exists f. M(t = f(v), in) = S(in)$$

Therefore, the purpose of this approach is to find the internal variables v instead of finding the function f for the internal signal. We know that a variable can choose the value of 0 or 1, so depending on the value of the variable, there are two cases as shown in Figure 2. The definition of in1 case and in2 case are as follows.

Definition 2. $in1\ case$ is defined as the case when the input $in1$ corrects the circuit only if $t = 0$. If $t = 1$, the circuit becomes buggy. This definition can be expressed as: $M(t = 0, in1) = S(in1) \wedge M(t = 1, in1) \neq S(in1)$.

Definition 3. $in2\ case$ is defined as the case when the input $in2$ corrects the circuit only if $t = 1$. If $t = 0$, the circuit becomes buggy. This definition can be expressed as: $M(t = 1, in2) = S(in2) \wedge M(t = 0, in2) \neq S(in2)$.

With Definition 2 and Definition 3, we can derive the following theorem by combining them: if M is correctable with t and internal signals v, then the followings must be satisfied:

$\forall t1, t2. M(t1 = 0, v1, in1) = S(in1) \wedge M(t1 = 1, v1, in1) \neq S(in1) \wedge M(t2 = 1, v2, in2) = S(in2) \wedge M(t2 = 0, v2, in2) \neq S(in2) \rightarrow v1 \neq v2.$

We can transform the former equation into the SAT equation below:

$\forall t1, t2. M(t1 = 0, v1, in1) = S(in1) \wedge M(t1 = 1, v1, in1) \neq S(in1) \wedge M(t2 = 1, v2, in2) = S(in2) \wedge M(t2 = 0, v2, in2) \neq S(in2) \wedge v1 = v2.$

If the equation above is UNSAT, then we have found that the set of internal signal v, are sufficient as the fanins. Here we give a simple proof: if two different inputs $in1$ and $in2$ result in the same $v1$ and $v2$ $(v1 = v2)$, that means we don't have two different circuits for one of the inputs is correct and the other one incorrect. It is a contradiction. By iteratively solving this SAT equation and check the sufficiency of these candidates, we can finally get the solution or no solution. The detail of calculation can be checked in [2].

3. SPFDs

In this section, we will discuss the concepts and definition of Set of Pairs of Functions to be Distinguish (SPFDs). The key point to understand SPFDs is that how a logic function **"distinguishes"** the pairs of logic functions.

To understand "distinguish", we may first introduce "include".

Definition 1. For two logic functions f and g. If $f(x)$ becomes 1 for all the input x where $g(x)$ becomes 1, we said that f include g. It can be written as $g \leq f$.

Definition 2. For a pair of logic functions g_1 and g_2, f can distinguish them if either one of the following conditions is satisfied.

$$g_1 \leq f \leq \overline{g_2}$$
$$g_2 \leq f \leq \overline{g_1}$$

The g_1 could be seen as one of the ON-Set and g_2 is one of the OFF-Set.

Definition 3. SPFD is a set of pairs of functions to be distinguished. It can be expressed like the following.

$$\{(g_{1a}, g_{1b}), (g_{2a}, g_{2b}), \ldots, (g_{na}, g_{nb})\}$$

Then if a function f can satisfy a SPFD, it means that f can distinguish all the g_{ia} and g_{ib} in the SPFD like that:

$[(g_{1a} \leq f \leq \overline{g_{1b}}) + (g_{1b} \leq f \leq \overline{g_{1a}})] \wedge \cdots \wedge [(g_{na} \leq f \leq \overline{g_{nb}}) + (g_{nb} \leq f \leq \overline{g_{na}})]$

Besides, we must know that not only nodes but also wires have their SPFD. The SPFD of node N_i can be expressed as $SPFD_i$. The SPFD of the wire $C_{[i,j]}$ can be expressed as $SPFD_{[i,j]}$.

Here we give an easy example the SPFD of an AND gate has been shown in Fig 3.

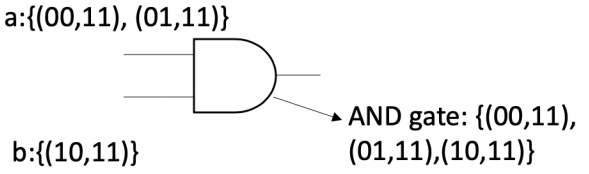


Figure 3. The SPFD of AND gate

In Fig 3. We all know the logic function of AND gate is $f = a \cdot b$. Then in this situation, the ON-Set of the SPFDs is (11), that OFF-Set of the SPFD is (00, 10, 11). Combine the ON-Set and OFF-Set, we can get the SPFDs about the AND gate is {(00,11), (01, 11), (10, 11)}. For the input wires a and b. The SPFD of wire a can define as following, we look at the pairs of function in SPFD of AND gate, we find that fanin wire a will have different value in these pairs (00, 11) and (01, 11), then we can assign these two pairs to the wire a. And the rest is assigned to wire b.

Given a SPFD $\{(g_{1a}, g_{1b}), (g_{2a}, g_{2b})\}$, we can define the logic function is $f = \mathrm{A} \cdot g_{1a} + \bar{A} \cdot g_{1b} + \mathrm{B} \cdot g_{2a} + \bar{B} \cdot g_{2b}$. By controlling the variable of A and B, we can get different logic function f which satisfy the SPFD and can be fixed to the node. This is "flexibility" provided by SPFD that we use to transform the logic circuit.

4. Equivalence

4.1. Explanation of Equivalence between the two approaches

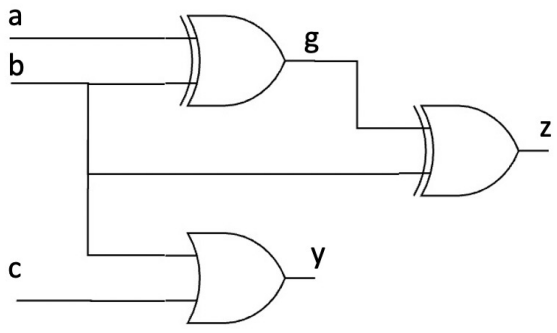
In this section, we want to discuss the equivalence of these two methods.

Recall the Debugging method Signal Selection. The goal of this method is to find the different value of internal signal v. Let's see again this equation:

$M(t1 = 0, v1, in1) = S(in1) \wedge M(t1 = 1, v1, in1) \neq S(in1) \wedge M(t2 = 1, v2, in2) = S(in2) \wedge M(t2 = 0, v2, in2) \neq S(in2) \rightarrow v1 \neq v2$. Where $t = f(v)$.

We can easily find that these in1 case and in2 case can be seen as the ON-Set and OFF-Set of SPFD. In in1 case, circuit will be correctable if $t = 0 = f(v1)$, and in in2 case, circuit will be correctable if t $= 1 = f(v2)$.

We use the SPFD to express this condition like we have many candidate pair of $(in1, in2)$, and the function f can distinguish these pairs means that these variables would be the correct input to this node. From the explanation before, the definition of Signal Selection can be easily transformed to the SPFD form. This prove that they are equivalent in theoretically.



a	b	c	g	y	z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	1	0	1	1

Figure 4. simple circuit and it's truth table

Fig. 4. Show a simple circuit and its truth table. We assume there is buggy exist node g. From the truth table, we can get in1 case with input (a, b) is (00, 11), in2 case is (01, 10), if we compute the $in1\ case \times in2\ case$ = {(00, 10), (11, 01), (01, 00), (10, 11)}, it's the same as the SPFD of node g.

4.2. Meaning of Equivalence

Two approaches can used for logic synthesis. In [2], authors implement an experiment with the same circuits with Signal Selection. Since there is no bug in the circuit to be corrected, it would become logic optimization approach. It would find fewer number of inputs to the functions compared to the original number of circuit and find other inputs to the functions. That means if the current inputs are not desirable, we can use Signal Selection to find more desirable inputs. For SPFD, we can actually replace the input to the functions. Like the above example, from the truth table, we know that SPFD of z is {(11, 01), (11, 00), (01, 10), (00, 10)}, SPFD of g is {(00, 10), (11, 01), (01, 00), (10, 11)}, wire (g, z) can distinguish both SPFD of g and z, so it's SPFD would be (11, 01) and (00, 10). We can easily find that wire a can also distinguish wire (g, z), then we can find other input to gate z is primary input a.

There are some small differences. In Signal Selection method, we didn't know the logic function f. The $v1$ and $v2$ will be computed from iterative solving the SAT problem. After completing the searching for fanin signals, the logic function f can be fixed. But for the SPFDs method, we force the fuction f must distinguish all the pairs in the SPFD, so the function f can easily computed with computing the SPFD. Also, the Signal Selection method is based on solving the SAT problem, it can compute the fanin of the gate directly from nothing. But for SPFD of a node or wire, it needs to compute from primary output to primary input in a reverse topological order. It will cost much more space and time, especially the computing of SPFD usually use

BDD form.

They are mathematically equivalent and have similar effect on logic optimization, but the ways to compute are different. Depend on how to utilize them, we can use appropriate method in different logic optimization problem.

5. Experiment Result

In this section, we implement two experiments. The first one is to compare the performance of these two methods. The second one is to implement the SPFD to optimize the circuits.

We mainly use the ITC'99 benchmarks (combinational version) [4] and some LGSynth'91 benchmarks [5] for our experiment. The experiments are both implemented on the platform with CPU processor Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz and 500 GB memory running Linux kernel 5.5.4.

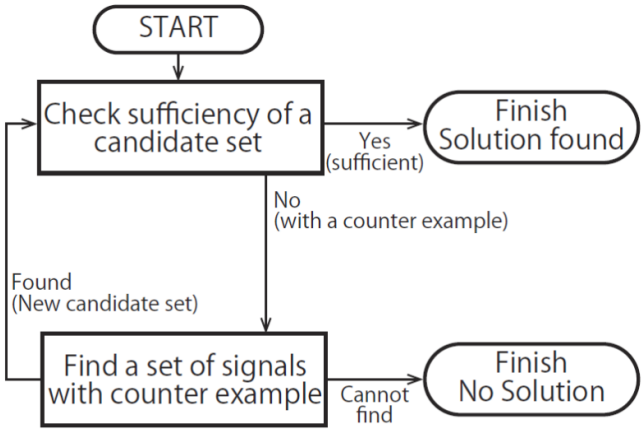


Figure 5 Overflow of iterative calculation of Signal Selection [2]

5.1 Experiment of comparing two methods

In this experiment, we focus on the runtime of two approaches. For SPFD, we compute the SPFD of circuit using BDD (Binary Decision Diagram). The computation is starting from primary output to primary input. We suppose the SPFD of primary output is given, so the process is:

(1). At each node N_i, the $SPFD_i$ is distributed to each input wires, creating the wire SPFDs.

(2). Once all the fanout wires SPFDs are computed, the node SPFD is computed as the union of the fanout wire SPFDs.

For the Signal Selection, because it is proposed by our former assistant researcher in our lab, we have the execute program to implement directly. The main computation process has shown in Figure 5. We choose the same circuit as both specification and implementation circuit. It is the basic computation to show efficiency of Signal Selection. Since the computation of SPFD need compute the whole circuit's node at once, we would try to find the inputs for all the internal nodes in Signal Selection.

Table 1 Result of Experiment 1

circuit	internal gates	runtime #1(s)	runtime #2 (s)	#iter (#2)
b01	48	0.224	0.308	2
b02	28	0.201	0.550	5
b03	157	1.017	0.695	6
b04	727	48.819	18.056	54
b05	998	11.937	1.866	12
b06	55	0.102	0.305	2
b07	441	23.672	1.353	9
b08	175	1.170	0.761	7
b09	170	0.120	1.108	10
b10	196	2.390	1.184	10
b11	764	3.824	0.829	7

The experiment result has shown in Table 1. Column 1 is the circuit name. The circuit used in this experiment come from ITC'99 benchmarks. Column 2 represent the number of the internal node in each circuit. Column 3 represent the runtime of computing SPFD of circuit. Column 4 represent the runtime of computing Signal Selection and column 5 is the number of iteration number of searching the fanins signal.

As we can see, with small circuit, the runtime of these two approaches don't have big difference. When the number of internal gates is small, both methods can get result immediately. When the number of node increase, the computation time of SPFD would also increase. That is because the computation of SPFD is based on BDD form. If the node's number is increasing, the number of BDD will be larger and larger. Then it will spend much more space to store the BDD information and much time to calculate the BDD operation. However, runtime of Signal Selection didn't increase obviously. We can assume that the runtime of two approaches not only concern the number of internal node but also the complexity of the internal logic mapping. In b04, both SPFD and signal selection will cost much time to get the result. It would due to the redundant gate in b04 circuit.

From the table, we can get some information. When you are planning to do some logic optimization on small circuit, both approaches can handle the task. If you want to deal with some large circuit like more than 1,000 or 10,000 internal nodes, the SPFD would perform poorly even that it can't get the result because the limitation of BDD. In this situation, we can choose Signal Selection to do the optimization.

Since they are equivalent, we can have some free choices to use both approaches to optimize logic circuit in different situations.

5.2 Logic synthesis by implement SPFDs

In this experiment, we want to use the SPFDs to optimize the circuit. The benchmark circuit would be transformed to AIG (And-inverter Graph). It can reduce the size of circuit which would make the SPFD computing more easier.

The optimization procedure can be conclude following:

Procedure optimizationCircuit:

(1) For each node N_i and wire $C_{[i,j]}$, compute the logic function f_i, the node $SPFD_i$ and the wire $SPFD_{[i,j]}$.

(2) Check each wire $C_{[i,j]}$, if the wire SPFD is null, remove the wire $C_{[i,j]}$, then go to step 4. Otherwise go to step 3.

(3) If there is a node N_k, its logic function f_k can satisfy the $SPFD_{[i,j]}$, then we replace the $C_{[i,j]}$ with $C_{[k,j]}$ and go to stpe 4. Otherwise go to step 2.

(4) If we change the fanins of the node N_j, we would modify the new logic funtion of N_j, then go to step 2.

(5) If all wires are selected, the procedure halt. Then check the internal node, if some node $N_i's$ fanout becomes empty, remove node N_i.

Table 2 Result of Experiment 2

circuit	initial wire	initial node	optimized wire	ratio (w)	optimized node	ratio (n)
i1	101	31	99	0.980	29	0.935
i10	4522	1772	TO	-	TO	-
k2	2946	1051	2936	0.997	1041	0.990
alu2	862	352	721	0.836	211	0.599
alu4	1562	638	1407	0.901	483	0.757
x3	1616	601	1507	0.933	492	0.819
des	8686	3500	TO	-	TO	-
apex6	1430	593	1383	0.967	546	0.921
apex7	440	167	415	0.943	142	0.850
cc	121	42	118	0.975	39	0.929
avg				0.942		0.850

Through the optimization procedure, we would use some other wires which satisfy the wire SPFD replace the original input wires. After the procedure, we can actually reduce the number of wires and nodes in the logic circuit. The experiment result has shown in Table 2.

Like Table 1, Column 1 is the circuit name. The circuit used in this experiment come from LGSynth'91 benchmarks. Column 2 and 3 represent the number of initial wire and initial node. Column 4 and 6 is the number of optimized wires and nodes. Ratio (w) is the optimization ratio of wire, ratio (n) is the optimization ratio of node. In i10 and des, we can find TO. It means “time out”, that is more than 30 minutes although we didn’t put the runtime on the table.

From the experiment result, we first talk about the “TO” cases. As mentioned before, SPFD can’t handle large circuit. i10 and des has more than 1500 node and 4000 wires in the circuit. “TO” tells us that the circuit of this size has exceeded SPFD scope.

For the remaining result, we find that after the optimization, the number of wires and nodes has decreased in different degree. It proves that optimization is correct although we didn’t implement accurate optimization calculation. The average wire reduction is 0.942. In some small circuit or large circuit, the reduction ratio is not ideal. We think it is because small circuit has less flexibility for used to optimize and large circuit has complex internal architecture.

The average node reduction ration is 0.850. It seems a good result. But it shows the same situation as wire reduction ratio. It seems only suitable for medium size circuits.

The average ratio tells us this optimization procedure is effective but still have room for improvement. We didn’t implement the accurate calculation for the reduction just replace the input wire with other wires. If we can compute the gain after replacing, maybe we would get better optimized result in both wire and node reduction.

6. Conclusion

In this paper, we mainly discuss two logic optimization method, Signal Selection and SPFDs. From the definition of two approach, we find that they are equivalent in mathematically. It can give us more choice in logic synthesis in different situation.

Then we implement two experiment. The first one is to compare the performance of two approach and the second one is to verify the effectiveness of SPFDs. The comparative experiment demonstrated the characteristics of two approaches by comparing their runtime. The runtime of SPFD would increase with the circuit size increased and Signal Selection would not change obviously. According the result, we find that both approaches can handle small circuits. But in large circuit, we would like to choose Signal Selection instead of SPFDs. In the second experiment, we verify the effectiveness of SPFD optimization. Because this experiment is just looking for some other wire can satisfy the original wire SPFD and replacing the original input wire, the reduction ratio is not so ideal. It still provides enough “flexibility” to transform the original circuit to another with the area reducing.

7. Future work

Even the SPFD is proposed in 1990s, it still have potential in logic synthesis fields. In the second experiment, we have optimized the circuit using the flexibility provided by SPFD. Although he result is not so good and can just handle middle size circuit. We can focus on computing the gain after replacing the original wires. It would improve the performance of wire reduction ratio. The Signal Selection is original logic debugging technique but it can apply for different fields. We know there are always multiple target occurs in nowadays synthesis or debugging process. It needs optimization method can handle multiple target in the same time. Now the SPFDs and Signal Selection are calculated one by one. It means that it is hard to handle multiple targets. About the Signal Selection, we sucessfully improved to calculate multiple targets in one time [6]. Since they are equivalent, we can refer to the method they took and extend the SPFDs calculation into multiple nodes. That would make SPFDs handles more different situations.

Reference

[1] S. Yamashita, H. Sawada, and A. Nagoya. A New Method to Express Functional Permissibilities for LUT based FP- GAs and Its Applications. In International Conference on Computer Aided Design, pages 254–261, November 1996.

[2] Amir Masoud Gharehbaghi, Masahiro Fujita: A New Approach for Debugging Logic Circuits without Explicitly Debugging Their Functionality. ATS 2016: 31-36

[3] Amir Masoud Gharehbaghi, Masahiro Fujita: A new approach for selecting inputs of logic functions during debug. ISQED 2017: 166-173

[4] https://github.com/squillero/itc99-poli

[5] https://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.pdf

[6] Yusuke Kimura, “Automatic Rectification Methods for Logic Debugging and ECO”, phd-thesis, 2020.

[7] S. Yamashita, H. Sawada and A. Nagoya, "SPFD: A new method to express functional flexibility," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 19, no. 8, pp. 840-849, Aug. 2000.

[8] https://people.eecs.berkeley.edu/~alanmi/abc/

[9] A. Mishchenko and R. K. Brayton, "SAT- based complete don't-care computation for network optimization," Design, Automation and Test in Europe, Munich, Germany, 2005, pp. 412- 417 Vol. 1

[10] Robert K. Brayton. Understanding SPFDs: A New Method for Specifying Flexibility. In IWLS, 1997.

[11] A. Mishchenko, J. S. Zhang, S. Sinha, J. R. Burch, R. Brayton and M. Chrzanowska-Jeske, "Using simulation and satisfiability to compute flexibilities in Boolean networks," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 5, pp. 743-755, May 2006.

[12] Masahiro Fujita: Automatic correction of logic bugs in hardware design: Partial logic synthesis. Procedia Computer Science 125: 790-800, 2018.

[13] H. Sato, Y. Yasue, Y. Matsunaga and M. Fujita, "Boolean resubstitution with permissible functions and binary decision diagrams," 27th ACM/IEEE Design Automation Conference, Orlando, FL, USA, 1990, pp. 284-289.