Area Optimization of Circuits Using Approximate Computing

Jiufa Zou, Lunyao Wang*, Zhufei Chu, Yinshui Xia
Faculty of Information Science and Engineering, Ningbo University, Ningbo 315211, China
* wanglunyao@nbu.edu.cn

Abstract—**Approximate computing can be applied to logic synthesis, known as approximate logic synthesis (ALS). ALS converts the original logic function into an approximate logic function with a more compact expression by introducing some acceptable errors, which can significantly reduce area, delay, and power consumption of the circuit. This paper proposes a novel ALS method that uses majority coverage to optimize the circuit area of the input circuit. The method primarily consists of two parts: one is to search for more compressed logical expressions with majority coverage, and the other is to use disjoint sharp products to separate error terms. The proposed method is implemented using the C programming language and tested with MCNC benchmarks. The experimental results show that the proposed method can also reduce the circuit area by 30.07% with an average error rate of 3.43% and that the proposed method is suitable for large function optimizations.**

Keywords—majority coverage; approximate computing; area optimization; logic synthesis

I. Introduction

Approximate computing means that there is a deviation within a certain range between the calculated result and the true result. The application of approximate computing in logic-level circuit design takes the advantage of using the fault-tolerant characteristics of the circuit to simplify the circuit to achieve the purpose of improving circuit performance and optimizing circuit parameters, such as power, area, delay, and so on. Currently, approximate computing has been widely used in fault-tolerant applications, including multimedia processing, sensor signal processing, and other applications.

The approximate computing-based circuit area optimization can be realized at the circuit level and logic level. At the circuit level, people usually focus on the approximate arithmetic circuits, such as the approximate adder[1], multiplier[2], and dividers[3]. By reducing the accuracy, the design space consisting of area, power, delay, and so on is enlarged and further optimization can be achieved. Approximate computing-based logic synthesis[4]-[6] uses an algorithm to find the best approximate circuit version of a given circuit within the error constraints. ALS includes multi-level circuit synthesis and two-level circuit synthesis, as well as combined circuit and approximate circuit. This paper focuses on approximate logic synthesis of two-level circuits.

One primary problem with ALS is selecting appropriate error metrics. Various metrics such as error rate (ER), error margin (EM), and average error margin (AEM) have been proposed for different applications. The ER and EM are widely used in the ALS research community. The ER is defined as the percentage of the number of input combinations that led to the error output in the original function. The EM is defined as the maximum difference between the incorrect output and the correct output. Here we use the ER as a measure. In the published reports, and in this paper, the number of literals is used to measure the circuit area in this paper.

In ALS algorithms, adding or removing certain minterms is a common method for logic optimization. An exhaustive search is always needed to find a minterm complement for maximum literal reduction. In this paper, a novel ALS method for circuit area optimization using majority coverage is proposed. By countering the number of occurrences of each input variable in products, a logic cover named majority cover is found which includes the variables with the larger counter value. The new logic covers containing as many original product terms as possible in a given ER constraint but having fewer literals reduces the circuit area and reduces the number of possible iterations.

The remainder of the paper is organized as follows. Section II explains the motivation and the principle of the majority cover search, and briefly describes the method for ER computing. Section III proposes an algorithm that finds a majority cover. Section IV provides the experimental results and Section V provides the conclusions.

II. Majority Cover Search and ER Calculation

In approximate computing-based optimization, the number of literals is used to represent a circuit area. To obtain a compacted logic expression, A heuristic approach was proposed to select 0 to 1 minterm complements to find an approximate circuit version that had fewer literals for a given ER[4]. The ER was used as a metric, and the number of literals was used to represent the circuit's area[4].

It was that a compacted approximate expression was obtained by converting those "0" minterms with a larger number of adjacent "1" minterms in the K-map of the circuit into Don't care value and expanding the original cover into a larger cover which led to a compacted expression[5].

It was that a logic function was approximately simplified by adding a certain number of minterms. The addition of these additional minterms could put some small products into a large product, thus reducing the number of literals in the expression[6].

978-1-7281-6879-1/20/$31.00 ©2020 IEEE

In those above methods, the approximate functions are achieved by adding or removing some special minterms and expanding the original cover to get a more compacted expression. These are the minterm-based methods. Consideration of the number of minterms increases rapidly with the number of circuit inputs. Therefore, minterm based-methods will be inefficient for large function optimization. In this paper, a majority logic cover-based approximation method is proposed to avoid the rapid increase of minterms in large function optimization.

A. The Principle of majority Cover Based Optimization

For an n variables logic function f, let C_{on}, C_{off}, and C_{dc} represent the set of logic "1" (known as $ONSET$), logic "0" (known as $OFFSET$), and “don't cares” (known as $DCSET$) respectively. And C is the universal set of f which can be expressed as:

$$C = C_{on} \cup C_{off} \cup C_{dc} \tag{1}$$

Suppose C^m is a majority cover of C; C_{poff} is a sub-cover of C^m and the products in C_{poff} is included by C_{off}, namely $C_{poff} = C^m \cap C_{off}$. Suppose C_{pon} is a sub-cover of C_{on}, and for any product p_i, $p_i \in C_{pon}$, there exits $p_i \cap C^m \neq \emptyset$. The C_{pon} can be expressed as the following.

$$C_{pon} = C^m \oplus C' \oplus C_{poff} \tag{2}$$

Where: $C' \subseteq C_{pon}, C' \cap C^m = \emptyset$.

Let $C_{pon} \rightarrow \sum_{i=1}^{h} p_i$; $C' \rightarrow \sum_{l}^{u} p_l$; $C_{poff} \rightarrow \sum_{j=1}^{r} p_j$; $C^m \rightarrow p^{n-m}$, then Eq (2) can be rewritten as:

$$\sum_{i=1}^{h} p_i = p^{n-m} \oplus \sum_{l=1}^{u} p_l \oplus \sum_{j=1}^{r} p_j = p^{n-m} \oplus \sum_{s=1}^{v} p_s \tag{3}$$

Where $v = u + r$. Therefore, if $h > v + 1$, the number of products in $(P^{n-m} \oplus \sum_{s=1}^{v} p_s)$ will be less than that of $\sum_{i=1}^{h} p_i$. If the “EXOR” operator in (3) is replaced by “OR”, then the minterms in $\sum_{s=1}^{v} p_s$ will lead error outputs, and $\sum_{s=1}^{v} p_s$ is the error set. The minterms in error set will lead error outputs. To find that the sub-cover C^m which meets the inequality $h > v + 1$, it is specified that the C^m is considered as a good candidate cover for approximate optimization when the constraint shown in (4) is satisfied.

$$N \geq M \times 2^m \tag{4}$$

$x_1x_2 \backslash x_3x_4$	00	01	11	10
00		1	F	
01		1	1	
11				1
10				

Figure. 1. The example TB simplification using ALS.

In (4), N is the number of minterms in C_{on}. M is a proportional coefficient. In order to make the inequality $h > v + 1$ easier to hold, we specify $M \geq 3/4$. Namely, most of minterms in C^m take the value "1" or DC. Therefore C^m is called as majority cover.

The idea of logic synthesis using majority cover was first proposed by Tran which was based on minterms[7]. Minterms based majority cover optimization was easy to find C^m, easy to separate C^m from C, and easy to obtain the sub-expression $\sum_{s=1}^{v} p_s$ in (3). But it is not suitable for large function optimization.

An improved majority cover optimization was proposed using products instead of minterms[8]. the majority cover method was used for large mixed polarity Reed-Muller function optimization. To reduce the computational complexity, the products were converted into disjointed products before the majority cover searching[8]. Considering that the number of products of a function is usually far less than the number of minterms, therefore, the product-based majority cover is very suitable for large function optimization.

B. Circuit Area Optimization Based on Majority Coverage

Fig.1 shows an example of 4-input logic function area optimization using majority cover and approximate computing techniques. The function is $f = \overline{x_1x_3}x_4 + \overline{x_1}x_2x_4 + x_1x_2x_3\overline{x_4}$ which has 10 literals. Fig.1 is its K-map. For a 4-variable logic function, there are 16 kinds of input combinations. Each input combination can be represented by a minterm. In Fig.1, the value of 4 minters is 1, and the other 12 minterms are 0.

When the value of a minterm is intentionally changed from 0 to 1 or vice versa, it means the related output is changed, too. Namely, an error is introduced. Approximate computing technique based area optimization is to find such minterms(or products), by changing their values, get a more compacted function with some error outputs.

In Fig.1, we modify the value of minterm $\overline{x}_1\overline{x}_2x_3x_4$ (marked F in K-map) from 0 to 1. That means when input is (0011), the output will not 0 but 1. After doing so, we can get an approximate function. It can be expressed as $f_{app} = \overline{x_1}x_4 + x_1x_2x_3\overline{x_4}$. Compare to the original function, the approximate function has 6 literals. Literals are reduced by 4 which means circuit area saving. But at the same time, the output error is introduced because of the modification of minterms value. In Fig.1, there are 16 kinds of inputs, and one of them has an error output, so the ER is 1/16.

The output difference between the outputs of the approximation function and the original function is the errors of the approximation function. The difference can be calculated by the disjointed sharp product between the logic covers of approximate function and its original function. Here the symbol “$\otimes$ ” represents the disjointed sharp product operator. For cover C_a and cover C_b, let C_{dis} is the result of the disjoint sharp product operation between C_a and C_b, then

$$C_{dis} = C_a \otimes C_b = C_a - C_a \cap C_b \tag{5}$$

Equation (5) shows the result of $C_a \otimes C_b$ is equal to the rest after removing the common part of C_a and cover C_b from C_a. Therefore, let C_{app} is the cover of the approximate function and C_{org} is the cover of the original function, C_{er} is the difference between C_{app} and C_{or} , then

$$C_{er} = \left(C_{app} \otimes C_{org}\right) \cup \left(C_{org} \otimes C_{app}\right) \qquad (6)$$

Where $\left(C_{app} \otimes C_{org}\right)$ stands for the part of the cover that belongs to C_{app} but not to C_{org}, and $\left(C_{org} \otimes C_{app}\right)$ stands for the part of the cover that belongs to C_{org} but not to C_{app}. The union of $\left(C_{app} \otimes C_{org}\right)$ and $\left(C_{org} \otimes C_{app}\right)$ is the difference between C_{app} and C_{org}. Furthermore, the products in C_{er} are disjointed which makes it very easy to count the number of minterms in C_{er}[8].

The advantage of using the disjoint sharp product operation for error computing is that the operation is based on covers(or products), not minterms, and it is fast and suitable for large functions.

The ER is defined as the ratio of the number of input combinations that cause output errors to the total number of input combinations of the original function. The ER can be expressed as:

$$ER = \frac{Nums(C_{er})}{Nums(C_{org})} \times 100\% \qquad (7)$$

Where $Nums(C_{er})$ and $Nums(C_{org})$ represents for the minterms number in C_{er} and C_{org}, respectively.

III. Approximate Logic Optimization Using Majority Cover

Majority cover based approximate optimization can be divided into two parts. One is the majority cover C^m searching and the other is the ER computing.

The C^m search can be achieved using the ONSET table shown in Fig.3. The onset table consists of three parts. They are T_{up},T_m and T_l respectively. And the part T_l is initialized with "-1" at the beginning.

In T_{up}, variables are in the first row, and the other rows are the products of the original function. For a variable x_j in a row,$x_j \in (0,1,-)$, means the complement of x_j, x_j itself, and x_j doesn't appear respectively.

$x_1x_2 \backslash x_3x_4$	00	01	11	10
00		1	F	
01		1	1	1
11	1	F	1	
10		1	1	

Figure. 2. The K-map corresponding to f_1.

		x_1	x_2	x_3	x_4
T_{up}		0	-	0	1
		0	1	1	-
		1	0	-	1
		1	1	0	0
		1	1	1	1
T_m	T_0	2	1	2	1
	T_1	3	3	2	3
	T_{dc}	0	1	1	1
T_l		-1	-1	-1	1

Figure. 3. The $ONSET$ table of f_1.

		x_1	x_2	x_3	x_4
T_{up}		0	-	0	1
		0	1	1	-
		1	1	0	0
		1	1	1	1
T_m	T_0	2	0	2	1
	T_1	2	3	2	2
	T_{dc}	0	1	0	1
T_l		-1	1	-1	1

Figure. 4. For the variable x_2 split $ONSET$ table.

In T_m, each column is used to store the number of "0" or "1" in the corresponding column in T_{up}. The first row of T_m labeled "T_0" is used to store the number of "0" in each column, and the number of "1" is stored in the second row labeled "T_1" and the number of "-" is stored in the bottom row labeled "T_{dc}".

The part T_l is for the product P^{n-m} generation by following steps.

Step1: initialize the T_l with -1s;

Step2: let $T_0' = T_0 + T_{dc}$,$T_1' = T_1 + T_{dc}$, and find the largest number in T_0' and T_1', if the largest number is in T_0', then reset the bit at the same column of T_l; if the largest number is T_1', then set the bit at the same column of T_l; otherwise mark "2" at the same column T_l;

Step3: convert T_l into P^{n-m}. The rule is "0" and "1" for the complement of a variable and the variable itself, and "-1" or "2" means the variable does not appear in P^{n-m}.

Here is an example to illustrate the process of majority cover search. Given a 4-input logic function $f_1 = \overline{x_1}\,\overline{x_3}x_4 + \overline{x_1}x_2x_3 + x_1\overline{x_2}x_4 + x_1x_2\overline{x_3}\,\overline{x_4} + x_1x_2x_3x_4$, Fig.2 is the K-map of f_1. Fig 3 is the ONSET table of f_1. Considering f_1 consists of 5 products, thus there are 5 rows in T_{up}. For example, the product

$p_0 = \overline{x}_1\overline{x}_3x_4$ of f_1 expressed as "0-01" in the first row. The number of times that each variable appears in T_{up} is listed in T_m. After summing T_0, T_{dc} and T_1, T_{dc},it can get $T_0^{'} = \{ T_0 + T_{dc} \} = \{ 2,2,3,2 \}$, $T_1^{'} = \{ T_1 + T_{dc} \} = \{ 3,4,3,4 \}$. The largest number is in column x_4 and x_2 in T_1'. Then set the bit of column x_4 in T_l shown as Fig.3. If there are more than one of the largest numbers in T_0' and T_1', then whichever is arbitrarily selected. For $T_l = \{-1,-1,-1,1\}$ in Fig.3, then $P^{n-m} = x_4$. The cover of P^{n-m} (or C^m) is the red dotted circle in Fig.2. In the circle, there are 8 minterms, and among them, 6 minterms take "1" which makes $M \geq 3/4$ in (4) hold. Therefore the red dotted circle is a majority cover. By using x_4 to replace those products covered by x_4 in Fig.2, we can get the approximation function $f_{1app} = x_4 + \overline{x_1}x_2x_3 + x_1x_2\overline{x_3x_4}$. Compared to f_1, f_{1app} has a more compact form and saves 9 literals. However, if we set the bit of column x_2 in T_l, for $T_l = \{-1,1,-1,-1\}$, then $P^{n-m} = x_2$. The products represented by T_l do not meet the majority cover constraint shown in (4). After that, further research for the majority coverage can be done based on P^{n-m}. For example, based on Fig.3, we can decompose the ONSET table by removing those rows in T_{up} whose bit in column x_2 is "0". The remaining rows form a new ONSET table shown in Fig.4, which then generates a new T_0' and T_1'. By searching for the largest number in T_0' and T_1' and setting or resetting the corresponding column in T_l, a new T_l is obtained. In Fig.4, we set the bit in column x_4 to be T_l and $T_l = \{-1,1,-1,1\}$, with the corresponding $P^{n-m}$is x_2x_4. The blue dotted circle in Fig.2 is for x_2x_4, which is a majority cover.

Considering the circuit's area can be measured by the number literals of a function, therefore, the main goal for the area optimization algorithm in this paper is to reduce the number of laterals in a function. The proposed algorithm described in pseudo-code-named TB_Area_App is shown below.

Algorithm_opt: TB _ Area _ App (C_{pla}, C_{app}, ER_{th})

Input: The original products set C_{pla} and threshold for error rate ER_{th}.

Output: The approximate products set with ER less than ER_{th}.

Initialize: current ER=0.

```
Do{  Cpla= Classify _output (Cpla); // step 1
      Capp= Majority _ cover _App (Cpla); //step2
      Cerr= Disjoin _Error ( Cpla, Capp ); //step 3
      ER= Error _rate _ Calculation ( Cpla, Cerr ); //step 4
   } while { ER ≤ ERth }
Print _result ( Capp, ER );
```

In the TB_Area_App, the logic function is described in PLA format. C_{pla} and C_{app} are the sets of products of a function before and after optimization, respectively.

In step1, the products in PLA is classified according to the output, and the products with equal outputs are put together.

In step2, searching the majority cover using *ONSET* table.

In step3, using the disjoint sharp product operation described in (5), and calculating the difference between C_{app} and C_{pla}. The result is stored in C_{err}.

In step 4, calculating the ER using (7). If the ER is less than ER_{th} which is the pre-set threshold of ER, then replace those products which are completely covered by the majority cover, otherwise discard the majority cover and search another one till no more majority cover can be found.

IV. Experimental Results and Analysis

The proposed algorithm is implemented in C and run on a PC with Windows 10, 3GHz CPU clock, and 8G memory. TABLE I shows the tested results with MCNC benchmarks. Where "i/o" is the number of inputs and outputs. Products_org is the number of product terms of the benchmark after optimized using ESPRESSO, Lts_org is the number of literals of benchmarks after optimized using ESPRESSO, and Lts_app represents the number of literals after optimized using approximation computing technique based on majority cover. Area_opt is for the improvement of area optimization which defined as *Area_opt={ (Lts_org-Lts_app) / Lts_org}*100%.*

TABLE I. Logic Function Optimization Results Based on Approximate Computing

circuits	i\o	Products_org	Lts_org	Lts_app	ER(%)	Area_opt (%)	times (ms)
5xp1	7\10	65	260	205	5.50	21.15	1
rd84	8\4	255	1774	1498	4.12	14.10	4
b12	15/9	43	149	97	6.58	34.90	1
b9	16\5	119	754	568	5.80	24.67	16
table5	17\15	158	1895	1808	0.27	4.5	<0.01
s1238	33\33	855	7266	3405	<0.01	53.13	15761
x1	51\35	275	1858	848	1.60	54.36	571
x7dn	66\15	538	4062	2913	<0.01	36.70	836
x3	135\99	656	3803	3528	5.10	7.23	1044
i7	199\67	264	861	448	5.06	47.96	261
Avg.					3.43	30.07	

From TABLE I, it can be concluded that by introducing error outputs, the circuit area can be further optimized compare to ESPRESSO. With an average ER of 3.43%, the number of literals can be reduced by 30.07%.

TABLE I also shows the CPU time required for each function. It can be found that the number of input variables has little effect on the speed of the algorithm. The algorithm speed is closely related to the number of products. Those functions with more products usually require more time to deal with. Considering the proposed algorithm is based on products, and the number of products usually far less than minterms, and it is suitable for large function optimization. In TABLE I, the largest function has 199 inputs.

V. Summary

In this paper, a novel circuit area optimization method using approximated computing is proposed. The approximate computing technique consists of majority cover search and ER calculation. Unlike the reported minterm-based methods, the method in this paper is based on products and has higher efficiency in dealing with large circuits. The ER calculation is also novel which is realized by checking the difference between the logic covers before and after optimization. The sharp disjoint operation is implemented in logic difference checking. The experimental results show that the proposed method is suitable for large function optimization and with an average ER of 3.43%, the number of literals in the function is reduced by 30.07%.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation of China under Grant 61471211, 61501268, U1709218, the Natural Science Foundation of Zhejiang Province under Grant LY19F040004 and the Natural Science Foundation of Ningbo under Grant 2019A610077.

REFERENCES

[1] M. Ramasamy, G. Narmadha and S. Deivasigamani, "Carry based approximate full adder for low power approximate computing," 2019 7th International Conference on Smart Computing & Communications (ICSCC), Sarawak, Malaysia, Malaysia, pp. 1-4, June 2019.

[2] S. Kim and Y. Kim, "High-performance and energy-efficient approximate multiplier for error-tolerant applications," 2017 International SoC Design Conference (ISOCC), Seoul, South Korea, pp. 278-279, November 2017.

[3] L. Chen, J. Han, W. Liu, P. Montuschi and F. Lombardi, "Design, Evaluation and Application of Approximate High-Radix Dividers," in IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 3, pp. 299-312, July-Sept. 2018.

[4] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, pp. 957-960, March 2010.

[5] S. Salamat, M. Ahmadi, B. Alizadeh and M. Fujita, "Systematic approximate logic optimization using don't care conditions," 2017 18th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, pp. 419-425, May 2017,.

[6] M. Fujita, "An approach to approximate computing: Logic transformations for one-minterm changes in specification," 2017 IEEE International High Level Design Validation and Test Workshop (HLDVT), Santa Cruz, CA, USA, pp. 91-94, December 2017.

[7] A. Tran and E. Lee, "Generalisation of tri-state map and a composition method for minimisation of Reed-Muller polynomials in mixed polarity," in IEE Proceedings E - Computers and Digital Techniques, vol. 140, no. 1, pp. 59-64, Jan. 1993.

[8] Wang L Y, Xia Y S, Chen X X, "Two-level MPRM functions optimization based on majority cubes," in Dianzi Yu Xinxi Xuebao(Journal of Electronics and Information Technology), vol.34, no. 4, pp. 986-99, 2 April 2012.