Approximate Decomposition of Multi-output LUTs under Acceptable Error Tolerance

Xuechen Zang
The University of Kitakyushu
Fukuoka, Japan
z8dcb401@eng.kitakyu-u.ac.jp

Shigetoshi Nakatake
The University of Kitakyushu
Fukuoka, Japan
nakatake@kitakyu-u.ac.jp

Hiroyuki Kozutsumi
TRL Corp.
Tokyo, Japan
h.kozu@trl.jp

Mitsunori Katsu
TRL Corp.
Tokyo, Japan
katsu@trl.jp

Shoichi Sekiguchi
TAIYO YUDEN Co., LTD
Tokyo, Japan
s-sekiguchi@jty.yuden.co.jp

Abstract— Approximate computing has been widely utilized in logic circuit optimization field to help achieve effective area compression and complexity reduction. This paper proposes a novel methodology for approximate computing with focus on the phase of decomposing large look-up tables (LUTs) into smaller individuals. By asserting reserved bits and finding optimal solutions, the proposed methodology is available to generate approximate LUTs under acceptable error tolerance rate. Experimental results show the decomposition of the 4-bit/8-bit multiplier logic and obtain 2-4 bit reduction within error rate of 5.4%-23.4% / 0-19.4%. The simplicity of the methodology and potential scalability point to several interesting directions for future research.

I. Introduction

Along with the rapid spread of artificial intelligence, approximate computing is becoming one of promising computation techniques which returns a possibly inaccurate result rather than a guaranteed accurate result. Conventionally, this kind of inaccurate computing is allowed for software. However, as growing mobile and embedded devices, the border of software and hardware implementation is no longer strict. A typical hardware implementation contains a logic synthesis, which is the process of converting a high-level description of design into an optimized gate-level representation. Usually, the source of logic synthesis information is hardware description language in RTL, and target is to generate netlist of circuit. After decompositioning, floorplan, placement and routing, the circuit can get mapping with designed logic finally. In recent, a logic is implemented to employ reconfigurable hardware device like FPGA or SRAM-based logic device [1] which owns programmable function to approach designed circuit logic.

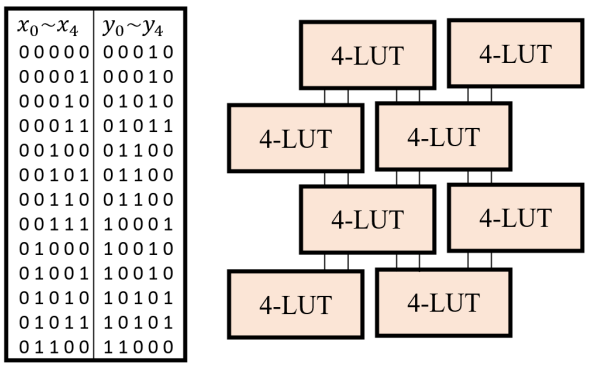
On the other hand, through the functional simulation, an input-and-output relationship of circuit logic can be obtained. As seen in Fig. 1 (a), the input-and-output relationship is regarded as a truth table of the given logic function. Supposing that a memory is enough large to store the truth table, the function can be realized one memory unit. In practical reconfigurable hardware devices, however, we must map the truth table to a set of look-up tables (LUTs) which are wired to adjacent each others as illulstrated in Fig. 1 (b). In general, the input and output size of the given truth table is much larger than the those of LUTs of the device. Therefore, decomposing a larger LUT logic to smaller LUTs logic combinations with remaining relatively low and acceptable error rate is conducive to employ reconfigurable hardware device efficiently and reduce the difficulty of mapping larger LUT logic.

In this paper, we provide a novel idea to generate a set of smaller LUTs whose logic function is the same as a given truth table. An essential procedure of the generation is to decompose a LUT into smaller LUTs which are connected in cascade style. A key idea is to combine efficiently decomposition of a logic and dividing of input by reserving input bits to the latter LUT in the cascade connection. Furthermore, our idea of decomposition has a significant extension, which can generate a set of LUTs corresponding to an approximation logic function under an acceptable error tolerance rate. The simplicity of the methodology and potential scalability point to several interesting directions for future research.

II. Background

A. Approximate Computing

Approximate computing is defined as a computing technique to generate results with possible inaccuracy, by relaxing the exact equivalency requirements between provided specifications and generated results. Some manual design techniques for approximate computing have been



(a) Truth Table: 5-input and 5-output

(b) Mappint to 4–LUTs: 4-input and 4-output

Fig. 1. Mapping to LUTs

proposed in recent years, mainly focus on approximate logic synthesis (ALS) and optimized reconstruction of circuit structure. It has emerged as a new design paradigm that exploits the error resilience of a wide range of application domains by allowing hardware implementations to forsake exact Boolean equivalence with algorithmic specifications.

B. Related Work

There have been a number of previous works focused on the utilization of approximate computing in the Electronic Design Automation (EDA) field. One class of the researchs mainly explores the structural characteristics and the importance of their output bits of specific circuits, and obtains performance improvements through manual approximation, like [2] and [3].

Another class, the automation becomes necessary as circuits have increasingly complex functionality. [4] proposes a heuristic methodology based on the principle of Karnaugh map optimization. [5] proposes a framework to automatically synthesize approximate circuits for a given error constraint. [6] proposes a method to synthesize approximate circuits using Boolean matrix factorization. In addition, the use of emerging machine learning technologies is also worth noting, like reinforcement learning to enhance ALS [7] [8] and deep learning in ALS optimization [9].

III. Approximate Decomposition for LUT

The LUT is a form of table that saves predefined information in form of array-like entries that are easily accessible. In Boolean logic, an N-bit LUT can encode a N-input Boolean function by storing the corresponding truth table. In the case of N bits, the LUT has 2^N rows, each row corresponds to one possible bit pattern. The inputs to the Boolean function drive the LUT to access the value of the corresponding output stored in the array.

A. Definition of Question

A universal LUT optimization definition is given as follows: for a given LUT and its approximation LUT, the Boolean logic of them are F and F'. The functions of both logics are represented as $Y = F(X)$ and $Y' = F'(X)$ (We label approximation relative variables by the apostrophe, lso in later). For the same input set $X = \{x_0, x_1, \ldots, x_n\}$, there are different output set $Y = \{y_0, y_1, \ldots, y_n\}$ and $Y' = \{y'_0, y'_1, \ldots, y'_n\}$. To evaluate the performance of approximation results, we define the correct rate R_c, which is defined as follows:

$$C_i = \begin{cases} 0, y_i \neq y'_i \\ 1, y_i = y'_i \end{cases} \tag{1}$$

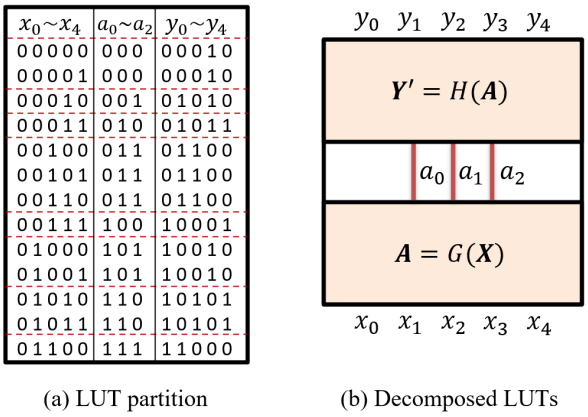
$$R_c = \frac{\sum_{i=1}^{n} C_i}{n} \tag{2}$$

For Boolean logic, an N-bit input Boolean truth table can represented by an N-bit LUT completely. While for a lot of actual Boolean logics, some different inputs do not always map different output values but the same ones, which generates repeated output values and provides the space of optimization. By counting the amount of the same outputs individuals to understand their distribution, we can efficiently decompose LUT and generate approximation within an acceptable error tolerance rate. To decompose LUT with large size effectively, we utilize the methodology of block dividing: For each non-repeat output value, we divide the inputs corresponding to the same output value into the same group and count the amount of output groups. Obviouly, for N-bit LUTs with d non-repeat output values, d is definitely not bigger than 2^N, which reveals potential optimization space. An the example shown in Fig. 2, a decomposed 4-bit LUT are combined of two parts. The first part $A = G(X)$ works like an encoder, which encodes input X to intermediate variables A. And the second part $Y' = H(A)$ works like a decoder, decoding intermediate variables A to Boolean output Y'. For the whole process, there is $Y' = F'(X) = H(G(X))$.

B. Overall Decomposition VS Reserved Bit Decomposition

About how to decompose a LUT, we propose two different ways to compare: overall decomposition (OD) and reserved bit decomposition (RBD). For more intuitive, a 4-bit LUT optimization comparison is showed in Fig. 3. The original LUT is of 4-bit inputs and 4-bit outputs, representing it simply as $(4, 4)$. In Fig. 3 (a), it has been decomposed to two new LUTs, $(4, 3)$ and $(3, 4)$ respectively. And in Fig. 3 (b), LUTs are of size $(3, 2)$ and $(3, 4)$. And the input x_3 is the reserved bit, which connects the sub-LUT, $Y' = H(A)$.

An attempt of decomposing a 4-bit LUT are shown in Fig. 3 to make a brief explanation. In Fig. 3(a) using



(a) LUT partition (b) Decomposed LUTs

Fig. 2. Example: Decompose a 5-bit LUT

the OD method, the condition for lossless decomposition is that the amount of non-repeat individuals of the output value set Y_1' is not over the maximum possibilities that the digits a_0 a_2 can express, that is, $2^3 = 8$. In Fig. 3(b) using the RBD method, the condition for lossless decomposition is that for each value (0 or 1) of the reserved bit x_3, the number of non-repeat individuals of output value set Y_2' is not over the maximum possibilities that the digits a_0 a_1 can express $2^2 = 4$. In Fig. 3(c), the output bits of the first sub-LUT $A = G(X)$ are further reduced on the basis of 3(b). In actual situations, this step will be executed if there is still rest optimization space after the RBD is completed. If not, this step will be skipped. Therefore, it is clear that: If an N-bit LUT can be decomposed losslessly by RBD, it can also be decomposeed losslessly by OD. On the contrary, if an N-bit LUT can be decomposeed losslessly by OD, it may not be able to be decomposed losslessly by RBD. Adding reserved bit reduces rest optimization space. To classify reserved bits does reduce the size of decomposed LUTs, while add restrictions to LUT decomposition.

In a short summary, if the result of OD is within the error tolerance, then further try to use RBD. If the RBD decomposition succeeds to obtain a feasible solution, further attempt will be made to reduce the intermediate transition output bits of the sub-LUT, so as to reduce the number of terminals of the sub-LUT and reduce the subsequent mapping cost.

C. LUT Decomposition Algorighm

Although it actually imports extra cost, the method of decomposeing larger LUTs to smaller ones make it easier to use reconfigurable hardware device to approach designed complex circuit logic. The decomposition cost can be evaluated from two indicators: one is the total additional size increment of the decomposed LUTs compared to the original LUTs, and another is R_c, the accuracy of the output of the decomposed circuit compared to the original LUTs. For the given and fixed LUTs logic and its decomposition, two factors are related to the size of the extra cost. The amount of non-repeat output values d and configuration of reserved bits. The two factors determine how many bits transition input and output $(a_0 - a_2)$ can be reduced in the intermediate process of decomposition, the size of the extra LUT size cost and the final accuracy result, R_c.

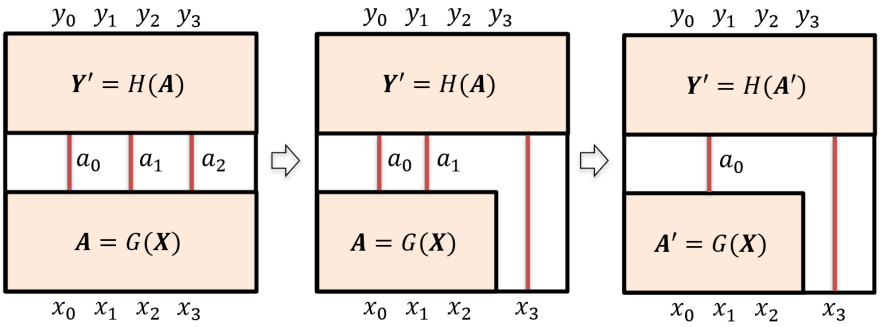
In OD process, it is not difficult to calculate the R_c from counting the non-repeat output value individuals. Assuming the output value set Y, having d different non-repeat output values. For an N-bit LUT and $2^N > d > 2^{N-1}$, the decomposition loss will be the $(d - 2^{N-1})$ groups which owns the least entries. Thus, R_c is available to be calculated. For RBD, it is unavailable to obtain the R_c of the whole LUT due to the existing of reserved bits. Because it is possible that same output values in the same divided group, while correspond unequal reserved bits values. Therefore, we propose a methodology of divide-and-conquer, dividing the whole LUT into separate blocks and count the total loss to calculate the R_c of the whole LUT. The process of dividing LUT according to reserved bits has been shown in Fig. 4.

Another factor influencing dividing process and final performance is the choice of reserved bits, because different bit patterns have different weight of influence on internal logic. For an N-bit LUT, the choice space of k reserved bit is C_N^k, and the total space $\{C_N^1 + ... + C_N^N\}$. To find optimal solutions, we use depth-first search in the total space. The pseudo Algorithm. 1 is given to explain the procedure.

Algorithm 1 Decompose LUT by chosen reserved bits

Input: Y: the output of N-bit LUT L, R_{cm}: required minimal correct rate
Output: RES: LUT decomposition result

```
RES = {}
for k in range(0, N − 1) do
    space = Permutation(C_N^k) /* Search space: the permutation of reserved bit selection */
    length = len(space) /* the length of search space */
    for i in range(0, length) do
        res_i = Decompose(Y, space[i]) /* Decompose LUT L by selected reserved bits */
        R_c = CalculateLoss(res_i) /* Calculate the correct rate R_c */
        if R_c > R_cm then
            RES.append(res_i)
        end if
    end for
end for
return RES
```



(a) Overall decomposition (OD)

(b) Reversed bit decomposition(RBD)

(c) Lower Complexity RBD

Fig. 3. Progressive LUT decompositions

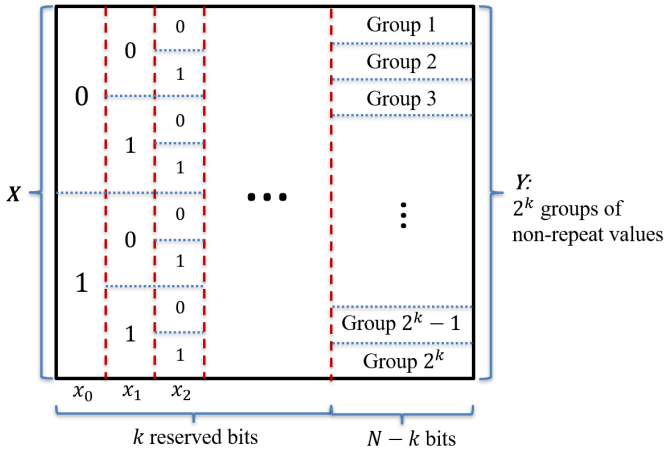


Fig. 4. LUT decomposition process with reserved bits

TABLE I
4-BIT/8BIT MULTIPLIER LOGIC DECOMPOSITION

Case	Method	LUT Size	Reserved Bit	Correct/Total
4-bit MUL	OD	(8,6)+(6,8)	N/A	215/256
	RBD	(7,6)+(7,8)	x_3	242/256
		(6,5)+(7,8)	x_3, x_7	229/256
		(5,4)+(7,8)	x_2, x_3, x_7	196/256
		(4,3)+(7,8)	x_1, x_2, x_3, x_7	183/256
8-bit MUL	OD	(16,14)+(14,16)	N/A	63288/65536
	RBD	(15,14)+(15,16)	x_7	65536/65536
		(15,13)+(14,16)	x_7	52784/65536
		(14,12)+(14,16)	x_7, x_{15}	48711/65536
		(13,12)+(15,16)	x_6, x_7, x_{15}	57611/65536

IV. Experiments

In this section, we test the LUT decomposition methodology on cases of basic arithmetic multiplier logic, using an 4-bit multiplier logic and a 8-bit multiplier logic respectively. For the 4-bit multiplier logic, the input part consists of two parts $i_1 = x_0x_1x_2x_3 \& i_2 = x_4x_5x_6x_7$, each part as two 4-bit binary numbers. And the output $i_1 \times i_2$ is an 8-bit binary number. The 8-bit multiplier logic is similar and includes two 8-bit input and one 16-bit output. The error rate limitation is set as 30% and the results beyond it will be droped. The operation result of the LUT decomposition methodology under 4-bit and 8-bit multiplier logic are shown in Tab. I.

The LUT decomposition methodology optimizes the two indexes under the error rate tolerance: the amount and selection of reserved bits, and the another is the maximum decomposition accuracy of the remaining part, the sub-LUT. The benefits and additional costs under each approximate solution are also shown in Tab. II. Experimental results show the decomposition of the 4-bit/8-bit multiplier logic and obtain 2-4 bit reduction within error rate of 5.4%-23.4% / 0-19.4%.

Operation time: 4-bit 2.271 s, 8-bit about 44min. Operated under the hardware environment of Intel i7-6700HQ and 16GB RAM.

V. Conclusion

We have introduced a divide-and-conquer methodology focusing how decompose a large truth table into smaller LUTs and combine them to the approximation of original truth table. The experiments of universal multiplier logic have been conducted and the result shows that it is available to generate approximate truth tables under an acceptable error tolerance by reasonable selection of reserved bits and decomposition accuracy. This reduces the difficulty of mapping large-scale LUT logic to finite input/output programmable logic circuits effectively.

As the data results shown in Section IV, there are multiple approximate LUT decomposing solutions for the same

TABLE II
Decomposition Performance Summary

Case	LUT Size	Method	Total Bit Reduction	Correct Rate	Additional Size Cost
4-bit MUL	(8,6)+(6,8)	OD	2	0.840	25%
	(7,6)+(7,8)	RBD	2	0.946	50%
	(6,5)+(7,8)		3	0.895	25%
	(5,4)+(7,8)		4	0.766	12.5%
	(4,3)+(7,8)		5	0.715	6.25%
8-bit MUL	(16,14)+(14,16)	OD	2	0.966	100%
	(15,14)+(15,16)	RBD	2	1.0	50%
	(13,12)+(15,16)		4	0.879	12.5%
	(15,13)+(14,16)		3	0.806	50%
	(14,12)+(14,16)		4	0.744	25%

single LUT complexity reduction, and corresponding to different error rate and additional cost of LUT scale selection. On the other hand, because of the limitation of decomposition and exhaustive search in computational efficiency, when facing large scale LUTs. This will also be one of our future directions of exploring the way to search optimal decomposition solutions of larger size truth tables more efficiently.

References

[1] S. Wang, Y. Higami, H. Takahashi, M. Sato, M. Katsu, S. Sekiguchi, Testing of Interconnect Defects in Memory Based Reconfigurable Logic Device, IEEE 26th Asian Test Symposium (ATS), 2017.

[2] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, and K. Roy. IMPACT: Imprecise adders for low-power approximate computing. In Proc. ISLPED 2011, pages 409 –414, Aug. 2011.

[3] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In Proc. VLSI Design, pages 346 –351, Jan. 2011.

[4] Shin, D., & Gupta, S. K. (2010). Approximate logic synthesis for error tolerant applications. Proceedings -Design, Automation and Test in Europe, DATE, 957–960. https://doi.org/10.1109/date.2010.5456913

[5] Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K., & Raghunathan, A. (2012). SALSA: Systematic logic synthesis of approximate circuits. Proceedings - Design Automation Conference, 796–801. https://doi.org/10.1145/2228360.2228504

[6] Hashemi, S., Tann, H., & Reda, S. (2018). BLASYS: Approximate logic synthesis using boolean matrix factorization. Proceedings - Design Automation Conference, Part F1377(June). https://doi.org/10.1145/3195970.3196001

[7] Pasandi, G., Nazarian, S., & Pedram, M. (2019). Approximate Logic Synthesis: A Reinforcement Learning-Based Technology Mapping Approach. Proceedings - International Symposium on Quality Electronic Design, ISQED, 2019-March, 26–32. https://doi.org/10.1109/ISQED.2019.8697679

[8] Hosny, A., Hashemi, S., Shalan, M., & Reda, S. (2020). DRiLLS: Deep Reinforcement Learning for Logic Synthesis. Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, 2020-Janua, 581–586. https://doi.org/10.1109/ASP-DAC47756.2020.9045559

[9] Haaswijk, W., Collins, E., Seguin, B., Soeken, M., Kaplan, F., Susstrunk, S., & De Micheli, G. (2018). Deep Learning for Logic Optimization Algorithms. Proceedings - IEEE International Symposium on Circuits and Systems, 2018-May. https://doi.org/10.1109/ISCAS.2018.8351885