

Compacted-LUT: Fine-Grained Customizable LUT Architecture via SRAM-MUX Co-Optimization

Mingyang Chen¹, Yunfei Dai¹, Wai-shing Luk¹, Qing He², Yu He³, Lingli Wang^{1,†}

¹State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China

²Tongji University, Shanghai, China

³Phlexing Technology Company Limited, Hangzhou, China

† llwang@fudan.edu.cn

Abstract—Traditional FPGA PLB designs are constrained by the exponential increase in LUT area with the augmentation of inputs. Recent work has explored a pruned LUT based on the non-uniform distribution of Boolean functions in practical benchmarks, designing an 8-input PLB with enhanced functionality and a modest area overhead. Nonetheless, the existing LUT pruning algorithm is prone to local optima and focuses exclusively on SRAM pruning, neglecting lookahead optimization of the MUX tree. In this paper, we propose Compacted-LUT (CLUT), a fine-grained customizable LUT architecture via SRAM-MUX co-optimization. Based on the principle of LUT pruning, we design a novel representation for Boolean functions. This representation directly associates each Boolean function with the number of required SRAMs and MUX-tree transistors. On this basis, a novel evaluation model for the hardware-friendliness of Boolean functions can be formulated. We further design a beam search algorithm to identify an optimal subset of Boolean functions in target benchmarks based on evaluation results. With this subset, the customizable SRAM-MUX co-optimized CLUT architecture can be generated. Furthermore, we propose Asym-CLUT6, a function-diverse 8-input PLB composed of two variant 6-input CLUTs. We evaluate Asym-CLUT6 on VTR and Koios benchmarks. Post-route results show that, compared to the Altera Stratix 10-like architecture and Dual-RLUT6, Asym-CLUT6 reduces the area-delay product by 13.65% and 10.06% on average.

Index Terms—FPGA, Programmable Logic Block, LUT

I. INTRODUCTION

In recent years, the application of field-programmable gate arrays (FPGAs) at the edge has attracted significant attention [1] [2], due to their inherent parallelism, reconfigurability, and ultra-low latency. In such application scenarios, both resource utilization and the delay of FPGAs are highly critical. Therefore, FPGA architecture with multi-input programmable logic blocks (PLBs) is an ideal choice. On the one hand, multi-input PLBs provide powerful functionality, enabling the same logic function to be implemented with fewer PLBs. On the other hand, with the advancement of integrated circuit technology, the main source of delay in FPGAs gradually shifts to interconnections [3]. Multi-input PLBs reduce the number of PLBs along the critical path, thereby decreasing interconnect usage on the critical path and effectively reducing delay.

In modern commercial FPGA architectures, the core of programmable functionality within PLBs is the SRAM-based Look-Up Tables (LUTs). Its full functionality is highly favorable for the design of electronic design automation (EDA) tools. However, the exponential increase in LUT area with the augmentation of inputs fundamentally limits the design of multi-input LUT-based PLBs [4].

Since only a small subset of Boolean functions frequently occurs in practical benchmarks, several non-fully functional PLBs have been proposed to mitigate the area explosion problem of LUTs. COGRE in [5] uses 11 SRAMs and 5 NAND gates to implement 50.6% 6-input Boolean functions. Although COGRE can significantly reduce logic area, it inevitably leads to increased delay and routing area. Moreover, with the adoption of Shannon decomposition in Boolean function characterization, researchers have developed various extended LUT architectures composed of LUTs and logic gates [6] [7]. These extended LUTs achieve a balance between area overhead and performance. To further enhance performance, several heterogeneous PLBs [8] and multi-input PLBs [9] have also been explored. However, these non-fully functional PLB architectures are designed manually based on Boolean functions observed in practical benchmarks. For fast-growing application domains, their applicability and scalability remain limited.

Recently, a fine-grained pruned LUT architecture, named Reduced LUT (RLUT), has been proposed in [10]. It uses the Bayesian algorithm to optimize the SRAMs in LUTs automatically based on target benchmarks. This enables the generation of customized architectures tailored to different requirements, providing enhanced flexibility and scalability. Furthermore, an 8-input PLB (8-PLB) called Dual-RLUT6 [10] composed of two identical 6-input RLUTs has been developed. It can achieve a significant delay improvement with a minor area overhead.

However, the RLUT architecture still has several problems. **Firstly**, only the SRAM optimization is considered by the automatic design algorithm. The SRAM optimization result directly determines the subsequent MUX tree pruning. Nevertheless, the impact on the MUX tree pruning is not accounted by the algorithm. **Secondly**, the algorithm is prone to local optima. Essentially, it resembles a greedy optimization process, where each iteration looks for the best scheme to add an additional SRAM in the existing architecture. Bayesian optimization is merely applied within each iteration to find the best scheme. **Thirdly**, Dual-RLUT6 does not fully exploit the flexibility of RLUTs. The Boolean functions derived from the decomposition of a multi-input Boolean function are typically different. Thus, a PLB composed of complementary RLUTs would provide powerful functionality. Nonetheless, the logic capacity of Dual-RLUT6, composed of two identical 6-RLUTs, is limited.

In this paper, we propose Compacted-LUT (CLUT), a fine-grained customizable LUT architecture via SRAM-MUX co-

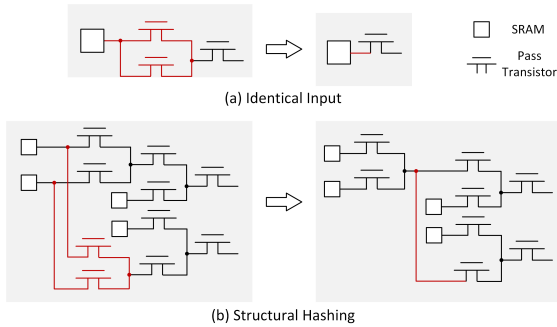


Fig. 1. Two situations of MUX tree optimization [10].

optimization. Based on the principle of LUT pruning in RLUT, we transform the truth table of each Boolean function into a novel two-dimensional representation, referred to as the bit assignment array (*BAA*). The distribution of elements in these arrays directly reflects the number of SRAMs and MUX tree transistors. On this basis, a novel evaluation model for the hardware-friendliness of Boolean functions can be formulated. We further design a beam search algorithm to identify an optimal subset of Boolean functions in target benchmarks based on evaluation results. With this subset, the customizable SRAM-MUX co-optimized CLUT architecture can be designed. Moreover, we propose Asym-CLUT6, a function-diverse 8-input PLB (8-PLB) composed of two variant 6-input CLUTs (6-CLUTs). We extend COFFE2 [11] to accurately model the area and delay of CLUT, use Boolean matching [12] to verify the logic capacity of CLUT, and modify the EDA flow of VTR [13] to accommodate CLUT. We evaluate Asym-CLUT6 on VTR [13] and Koios [14] benchmarks. Post-route results show that, compared to Altera Stratix 10-like architecture [15] and Dual-RLUT6 [10], Asym-CLUT6 reduces the area-delay product (ADP) by 13.65% and 10.06% on average.

The contributions of this paper include:

- We propose Compacted-LUT, a SRAM-MUX co-pruning fine-grained customizable LUT architecture.
- We introduce a beam search algorithm for the generation of CLUT architecture, which is based on the hardware-friendliness evaluation results of Boolean functions.
- We design a novel function-diverse 8-PLB composed of two variant 6-CLUTs named Asym-CLUT6 with significant improvements in ADP by the extended VTR flow.

The rest of this paper is organized as follows. Section II covers the background. In Section III, the architecture of CLUT and its automatic design algorithm are described in detail. Section IV introduces the EDA flow of CLUT. The experimental results are provided in Section V, followed by the conclusion in Section VI.

II. BACKGROUND

A. Principle of LUT Pruning

For a K -input Boolean function, there are 2^K possible input combinations. Therefore, to ensure full functionality, a K -LUT requires 2^K SRAMs. However, the distribution of Boolean functions in practical benchmarks is uneven, with the vast majority of Boolean functions rarely occurring. Thus,

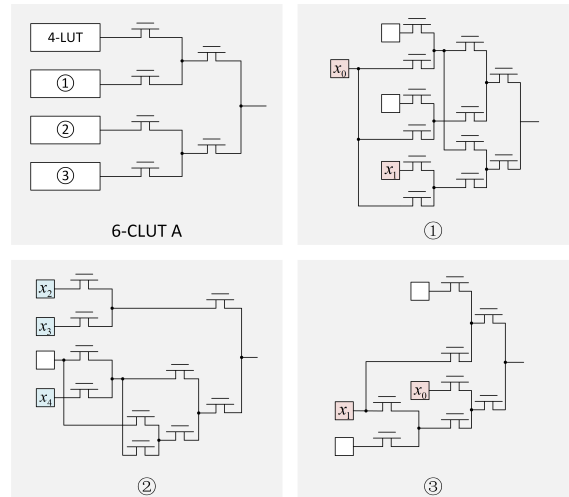


Fig. 2. The schematic of a typical 6-CLUT with 29 SRAMs (23 in CLUT and 6 in PINVs) and 66 pass transistors. Two red SRAMs x_0 represent the same SRAM, and the same applies to the SRAMs x_1 . Three blue SRAMs x_2 , x_3 , and x_4 also appear in the 4-LUT.

the required number of SRAMs is significantly less than 2^K . For the K -input Boolean functions f_0, f_1, \dots, f_n occurring in practical benchmarks, if there are two input combinations X_0 and X_1 satisfying that:

$$\forall i, f_i(X_0) = f_i(X_1), \quad (1)$$

two SRAMs originally used to store the results of X_0 and X_1 in the LUT can be pruned into one [10]. In particular, for the LUT after SRAM pruning, if the situations shown in Fig. 1 occur, its MUX tree can also be further optimized. The automatic design algorithm for RLUT focuses solely on SRAM optimization, neglecting its effects on the MUX tree. In CLUT, we design an evaluation model for the hardware-friendliness of Boolean functions. This model evaluates the Boolean functions from both the SRAM and MUX tree optimization perspectives, enabling the co-optimization of them in CLUTs.

B. NPN-equivalence-based Functionality Extension of LUT

The classification based on Negation-Permutation-Negation (NPN) equivalence is a commonly used method for categorizing Boolean functions. Two Boolean functions are considered NPN-equivalent if one can be transformed into the other by input negation, input permutation, and output negation [16].

To further extend the logic capacity of CLUT, similar to RLUT [10], input negation, input permutation, and output negation are supported by additional programmable inverters (PINVs) at the inputs, wire switching in routing, and inversions of SRAM configuration values. Therefore, as long as the CLUT can implement one Boolean function in an NPN equivalence class, it can implement all Boolean functions in the same class.

III. PROPOSED ARCHITECTURE: CLUT

A. Overall Architecture

A typical 6-CLUT architecture is shown in Fig. 2. It retains the basic architecture of SRAMs and pass-transistor-based MUX tree from conventional LUTs, while performing extensive pruning of both. Compared to RLUT, the MUX tree of CLUT

Truth Table									
Input	Output	1	0	0	0	1	0	0	0
00	0	0	1	1	1	1	0	0	0
01	1	0	1	1	1	0	1	1	1
10	1	0	1	1	1	0	1	1	1
11	1	0	1	1	1	0	1	1	1

Fig. 3. The M (center) and BAA (right) of 2-input Boolean function or .

is more compact. In addition, since more than 60% of LUTs in circuits after technology mapping have 4 or fewer inputs [10], we retain an embedded conventional 4-LUT in CLUT, as shown in Fig. 2. This embedded LUT enables CLUT to implement all Boolean functions with 4 or fewer inputs, ensuring that CLUT can support the realization of any circuit.

B. Evaluation Model for Boolean Functions

As a customized architecture, CLUT is generated based on Boolean functions observed in practical benchmarks. The occurrence frequency of these Boolean functions in the benchmarks varies, and the hardware resources required to implement them also differ. To make full use of hardware resources, we need to filter out Boolean functions that occur infrequently and are hardware-unfriendly. Therefore, we develop an evaluation model for the hardware-friendliness of Boolean functions, based on SRAM and MUX tree pruning in CLUT.

According to Eq. (1), the key to LUT pruning lies in the equivalence relationship of the Boolean function outputs under different inputs. We design a matrix M to store this relationship. For a K -input Boolean function, its possible input combinations include X_0, \dots, X_{2^k-1} , the elements of M are:

$$M_{ij} = \begin{cases} 1, & \text{if } f(X_i) = f(X_j) \\ 0, & \text{if } f(X_i) \neq f(X_j) \end{cases} \quad (2)$$

where M_{ij} represents the element in the i -th row and the j -th column of M , while $f(X_i)$ is the output with the input combination X_i of the target function f . Based on Eq. (2), M is symmetric with respect to the main diagonal. Furthermore, for $\forall i$, $M_{ii} = 1$ always holds. Thus, to further reduce complexity, only the triangular array above the main diagonal of M is necessary, which is defined as the bit assignment array (BAA):

$$BAA_{ij} = \begin{cases} 1, & \text{if } f(X_i) = f(X_{j+1}) \\ 0, & \text{if } f(X_i) \neq f(X_{j+1}) \end{cases} \quad (3)$$

where $0 \leq i \leq j < 2^k - 1$, while K is the input number of the target Boolean function. We take the 2-input Boolean function or as an example. Its truth table is [0, 1, 1, 1]. Therefore, its corresponding M and BAA are shown in Fig. 3.

With BAA , for a given set of Boolean functions $F_n = \{f_0, f_1, \dots, f_n\}$, we can quickly compute all input combinations that satisfy Eq. (1) by the logic *and* operation ($\&$). The result, referred to as the compacted array (CA), is calculated by:

$$CA_{ij}(F_n) = BAA_{ij}^{f_0} \& BAA_{ij}^{f_1} \& \dots \& BAA_{ij}^{f_n}, \quad (4)$$

where BAA^f is BAA of f . If $CA_{ij} = 1$, for any Boolean function f in F_n , $f(X_i) = f(X_{j+1})$ holds. Therefore, the outputs $f(X_i)$ and $f(X_j)$ can be stored in the same SRAM.

With CA and BAA , we formulate an evaluation model for the hardware-friendliness of Boolean functions, which includes both the SRAM-friendliness and the MUX tree-friendliness.

1) *SRAM-Friendliness Evaluation*: The geometric distribution of elements in the CA exhibits a mathematical relationship with the number of SRAMs in CLUT. For $\forall j$, if $\exists i$, $CA_{ij} = 1$, $f(X_{j+1})$ can share SRAM with $f(X_i)$. Otherwise, $f(X_{j+1})$ cannot share the SRAM with $f(X_0) \sim f(X_j)$, and a new SRAM must be added. Thus, each column where all elements are 0 in CA corresponds to an SRAM in CLUT on hardware.

The number of such columns is referred to as N_{sc} . Thus, the SRAM-friendliness evaluation of Boolean functions is based on their impact on the probability of an increase in N_{sc} . For a set of Boolean functions $F_n = \{f_0, \dots, f_n\}$, we can rewrite Eq. 4:

$$CA_{ij}(F_n) = CA_{ij}(F_{n-1}) \& BAA_{ij}^{f_n}. \quad (5)$$

On this basis, if $CA_{ij}(F_{n-1}) = 0$, f_n has no effect on the probability of an increase in N_{sc} of $CA(F_n)$. Otherwise, if $BAA_{ij}^{f_n} = 0$, the probability is inversely correlated with the number of elements that have a value of 1 in the j -th column of $CA(F_{n-1})$. Therefore, from the perspective of minimizing the probability of an increase in N_{sc} , the SRAM-friendliness evaluation result R_s of f_n is calculated by:

$$R_s^{f_n} = \sum \left(\frac{1}{q} \& CA_{ij}(F_{n-1}) \& BAA_{ij}^{f_n} \right), \quad (6)$$

where q is the number of elements with a value of 1 in the j -th column of $CA(F_{n-1})$. $R_s^{f_n}$ represents an estimation of the f_n impact on the probability of an increase in N_{sc} . For f_n with larger R_s , the probability of adding new SRAMs for the CLUT generated with it is less.

2) *MUX tree-friendliness*: There are two methods to optimize MUX tree transistors based on SRAM pruning, as shown in Fig. 1. To simplify the evaluation, this paper focuses solely on the optimization method known as *identical input*, as shown in Fig. 1(a).

For the K -input LUT, the MUX tree is used to select the output corresponding to adjacent inputs. In $CA(F_n)$, this corresponds to $CA_{(2i)(2i)}(F_n) = 1$. Thus, according to Eq. 5, the MUX tree-friendliness evaluation result R_m of f_n is:

$$R_m^{f_n} = \sum (CA_{(2i)(2i)}(F_{n-1}) \& BAA_{(2i)(2i)}^{f_n}), \quad (7)$$

$R_m^{f_n}$ directly reflects the impact of f_n on the number of MUX tree transistors in CLUT. For f_n with larger R_m , the number of MUX tree transistors in the CLUT generated with it is fewer.

In conclusion, the evaluation result R of the Boolean function f_n on the SRAM-friendliness and the MUX tree-friendliness is:

$$R^{f_n} = (R_s^{f_n} + \alpha R_m^{f_n}) \cdot \beta, \quad (8)$$

where β is the occurrence frequency of f_n , and α is the parameter. Based on the area of the pass transistor and the SRAM, α is set to 1.68 to better optimize the MUX tree.

C. Automatic Design Algorithm

Based on the hardware-friendliness evaluation of Boolean functions, we design an automatic algorithm to select a suitable

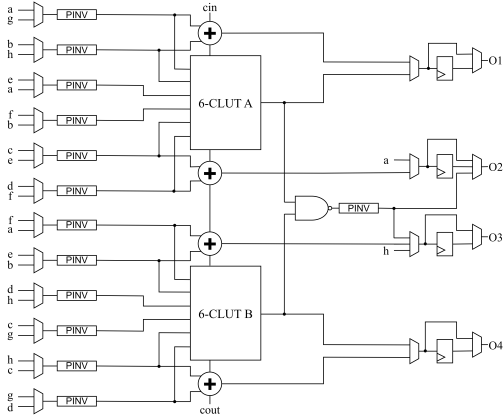


Fig. 4. The schematic of Asym-CLUT6.

set of Boolean functions to generate a CLUT. According to Section II-B, CLUT supports NPN-equivalence-based functional extension. Therefore, the objective of the algorithm is to:

- Select a suitable set of NPN equivalence classes.
- Select a Boolean function in each NPN equivalence class.

If the optimal set of functions is determined by exhaustive search, the complexity is impractical. To reduce complexity, we design a beam search algorithm. Beam search is an improved version of greedy search. In each step, it retains several of the best current results for the next step to achieve a globally better result. Firstly, we ignore NPN classes with extremely low occurrence frequency to reduce the complexity. Secondly, all functions and their corresponding *BAA*s in each NPN class are computed. Thirdly, beam search [17] is used to generate the optimized *CA*. In each iteration, the best set $\{CA_1, \dots, CA_{n_1}\}$ is retained. All functions in the library are evaluated with these *CAs* in the next iteration. For the best evaluated function, we remove all functions $\{f_1, \dots, f_{n_2}\}$ that are NPN equivalent to it from the library. Subsequently, we evaluate $\{f_1, \dots, f_{n_2}\}$ with $\{CA_1, \dots, CA_{n_1}\}$. We retain the top n_1 evaluation results to generate a new set of *CAs*. The iteration stops when the total frequency of the best function in each iteration exceeds the constraint. We generate the CLUT based on the obtained *CA* in the last iteration. For the first iteration, to support the embedded 4-LUT, we use a special *CA*, where the elements in the first 15 columns are 0, and the other elements are 1.

D. Asym-CLUT6

To enhance the functionality of the PLB and fully exploit routing resources, we propose an 8-PLB consisting of two variant 6-CLUT named Asym-CLUT6, as shown in Fig. 4. In Asym-CLUT6, CLUT *A* and CLUT *B* are partially complementary in functionality. CLUT *A* is generated first. Subsequently, we remove several Boolean functions from the function library that can be implemented by CLUT *A* and have a relatively low occurrence frequency. With the modified library, CLUT *B* is designed. CLUT *B* generated by this method is partially functionally complementary to CLUT *A*. Only very few Boolean functions cannot be implemented by either of them. Since Boolean functions obtained from the decomposition of a multi-input Boolean function are generally different, Asym-

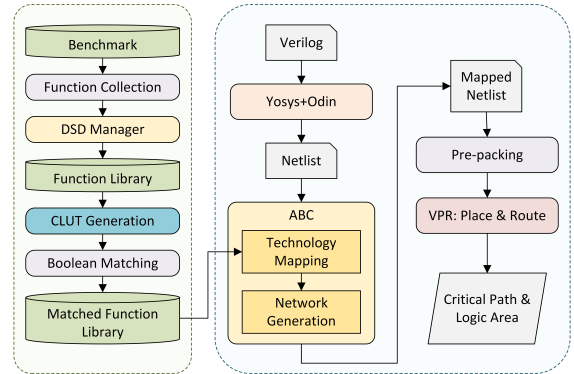


Fig. 5. The EDA flow of CLUT.

CLUT6 has greater logic capacity compared to Dual-RLUT6 [10], which is composed of two identical RLUT6s.

Due to the asymmetry of the CLUT inputs, when operating in fracturing mode, the interconnection for the shared inputs between the Asym-CLUT6 must be more complex. Asym-CLUT6 has 8 inputs, while two 6-CLUTs have 12 inputs. The least common multiple of 8 and 12 is 24. Therefore, in Asym-CLUT6, we added 12 2-to-1 MUXes at each 6-CLUT input to improve the flexibility of Asym-CLUT6 in the fracturing mode, as shown in Fig. 4. Moreover, due to the NAND gate, Asym-CLUT6 cannot negate the output by SRAM inversion. Thus, a PINV is added at the output to ensure that it can support NPN-equivalence-based functional extension.

IV. EDA FLOW OF CLUT

The EDA flow for CLUT is based on VTR flow, as shown in Fig. 5. Compared to conventional VTR flow, the main improvements include the construction of the Boolean function library, Boolean matching, and pre-packing.

A. Construction of Function Library

In ABC [18], the technology mapping for FPGA is represented by the command *if*. This command includes a cut-enumeration process that can be used to collect Boolean functions from the benchmark to build the function library. During this process, if a Boolean function is identified as the best cut, it is likely to appear in the final mapped netlist. Consequently, in the function library, its occurrence count increases by one. Moreover, a novel representation of Boolean functions in terms of their disjoint-support decomposition (DSD) [19] is used to make the function library more compact.

B. Boolean Matching

Unlike conventional LUTs, for the CLUT with incomplete functionality, Boolean matching [12] must be involved to verify whether a function can be implemented by CLUT. The Boolean matching problem is whether a Boolean function f with K inputs can be realized in a programmable circuit G with M inputs and l programmable bits. Its QBF [20] representation is:

$$H = \exists L_1 \dots L_l \forall x_1 \dots x_K \exists z_1 \dots z_o (G \equiv f), \quad (9)$$

where $x_1 \dots x_K$ and $L_1 \dots L_l$ represent input variables and programmable bits, and $z_1 \dots z_o$ are any auxiliary variable in G . We use the Boolean representation [21] to derive G and solve it with the QSAT solver [22].

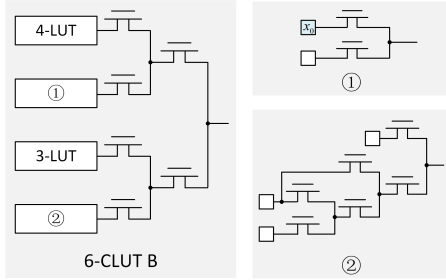


Fig. 6. The schematic of 6-CLUT B with 34 SRAMs (28 in CLUT and 6 in PINVs) and 58 pass transistors. The blue SRAM x_0 also appears in the 4-LUT.

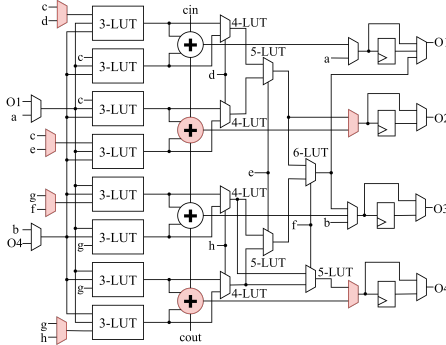


Fig. 7. The schematic of enhanced Stratix 10 ALM [15].

C. Pre-packing with QSAT Solver

Due to the asymmetric inputs of CLUT, before placement in VPR [13], pre-packing with the QSAT solver [22], is required to determine actual pin assignments. Besides, for two functions implemented in the fracturing mode of Asym-CLUT6, we must further verify whether one can be implemented by 6-CLUT A, and the other by 6-CLUT B. Otherwise, new functions need to be reassigned for the fracturing mode of Asym-CLUT6.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

We evaluate Asym-CLUT6 with the 19 largest VTR [13] and the 20 largest Koios [14] benchmarks, which are also used to build the function library. Based on these benchmarks, the proposed design algorithm generates 6-CLUT A and 6-CLUT B, as shown in Fig. 2 and Fig. 6. Asym-CLUT6 consists of these two CLUTs. We compare Asym-CLUT6 with Dual-RLUT6 [10] and enhanced Stratix 10 ALM with a carry chain, as shown in Fig. 7, where the red part highlights its difference from conventional Stratix 10 ALM. It also supports *extended LUT* mode, which can implement a subset of 8-input functions, and the fracturing mode [23]. Besides, we keep routing-related architectures the same to avoid disturbances of non-PLB parts.

Our experimental flow is shown in Fig. 5. Firstly, the input Verilog file is converted into a BLIF netlist by Yosys [24] and ODIN. Subsequently, ABC [18] performs technology mapping with the function library and Boolean matching results. We modify the *dsd_filter* command in ABC, which can remove Boolean functions from the library according to Boolean matching results. Consequently, we invoke the command *if -k* to perform technology mapping under the given function library.

TABLE I
THE AREA MODELING RESULTS OF CLB BY COFFE2 IN 22NM.

Area (MWTA)	Stratix 10 ALM [15]	Dual-RLUT6 [10]	Asym-CLUT6
10 PLBs	10168.0	13338.1	12082.0
40 FFs	1449.9	1449.9	1449.9
Input mux	8958.3	9407.2	9407.2
Out mux	1516.2	1516.2	1516.2
Carry	3109.5	3109.5	3109.5
Total CLB	25201.9	28821.0	27564.8

TABLE II
THE DELAY MODELING RESULTS IN 22NM.

Delay (ps)	Stratix 10 ALM [15]	Dual-RLUT6 [10]	Asym-CLUT6
6-LUT	186.12	164.63	164.63
NAND	N/A	13.55	13.55
PINV	N/A	19.84	19.84
Total	186.12	198.02	198.02

The mapped result is verified by *cec* command in ABC. With the modeling results, the mapped netlist is pre-packed and passed to VPR [13] for placement and routing. The routing channel width is set to 300, since the benchmark *LU64PEEng* needs a minimum routing channel width of 232, and a 1.3 \times minimum channel width is adopted.

B. Area and Delay Modeling

The area and delay modeling for Asym-CLUT6 is based on COFFE2 [11] in 22nm. We extend COFFE2 to support the modeling of CLUT. The area modeling results are shown in Table I, where *MWTA* is the minimum-width transistor area. Compared to Dual-RLUT6, Asym-CLUT6 has a more compact MUX tree, resulting in a 9.42% reduction in PLB area. However, due to PINVs, the Configurable Logic Blocks (CLB) area of Asym-CLUT6 is 9.38% larger than that of Stratix 10 ALM. Moreover, compared to Stratix 10 ALM, there are additional 2-to-1 MUXes at both Dual-RLUT6 and Asym-CLUT6 inputs. The *Input mux* in Table I accounts for this.

We generate the transistor-level netlist and derive the CLB delays through HSPICE simulations. The results, shown in Table II, include the delay of LUTs, as well as those of the NAND gate and PINVs. Since the delay of different input pins to the LUT is different, the average delay of 6-LUT is given. The impact of input sharing with MUXes at inputs on delay is considered separately in the architecture file. Since the pruning of the MUX tree does not affect the critical path, the delays of Dual-RLUT6 and Asym-CLUT6 are identical. Compared to Stratix 10 ALM, 6-CLUT in Asym-CLUT6 does not have to support fracturability, resulting in fewer internal buffers and a lower delay. However, the NAND gate and PINVs in Asym-CLUT6 introduce additional delay. Therefore, the average CLB delay of Asym-CLUT6 increases by 11.90 ps.

C. Boolean Function Coverage

The coverage of Boolean functions intuitively reflects the logic capacity of the PLB. The results of the functional coverage analysis for Stratix 10 ALM, RLUT, and CLUT are summarized in Table III, where *Total* refers to the total number of NPN equivalent classes, while *nClass* and *occur%* are the number of implementable classes and their total occurrence frequencies. Compared to 6-RLUT with 31 SRAMs and 90 pass transistors, 6-CLUT A with 29 SRAMs and 66 pass transistors has a greater logic capacity. This demonstrates that the design

TABLE III
FUNCTION COVERAGE OF STRATIX 10 ALM, RLUT, AND CLUT.

nInputs	Total	6-RLUT [10]		6-CLUT <i>A</i>		6-CLUT <i>B</i>		Stratix 10 ALM [15]		Dual-RLUT6 [10]		Asym-CLUT6	
		nClass	occur%	nClass	occur%	nClass	occur%	nClass	occur%	nClass	occur%	nClass	occur%
5	10196	9469	98.44%	8844	98.44%	9142	98.54%	10196	100.0%	10196	100.0%	10196	100.0%
6	68330	11504	82.17%	12363	83.57%	26324	83.74%	68330	100.0%	66555	99.95%	67954	99.99%
7	217074	N/A	N/A	N/A	N/A	N/A	N/A	83742	57.96%	151297	94.95%	190979	99.74%
8	388723	N/A	N/A	N/A	N/A	N/A	N/A	35199	19.59%	79222	70.05%	209305	92.08%

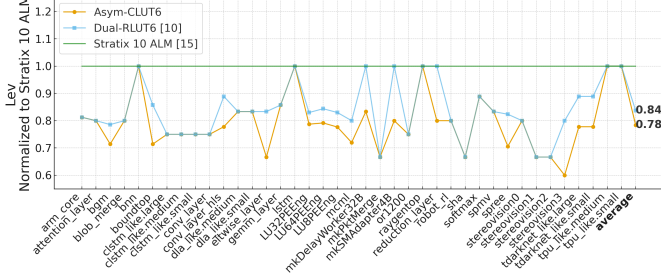


Fig. 8. Normalized results of *Lev* with respect to Stratix 10 ALM.

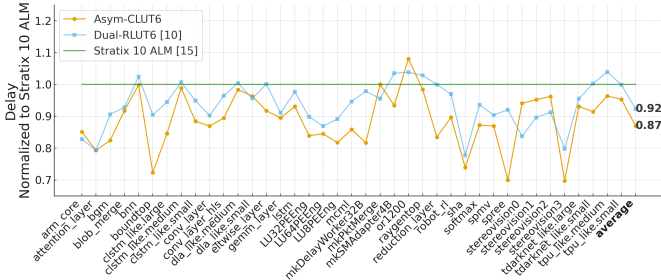


Fig. 9. Normalized results of *Delay* with respect to Stratix 10 ALM.

algorithm of CLUT can co-optimize the SRAMs and MUX trees. Moreover, a beam-search-based design algorithm can obtain better results compared to the design algorithm of RLUT.

Although the logic capacity of CLUT *A* and CLUT *B* is slightly higher than that of RLUT, the logic capacity of Asym-CLUT6 is significantly greater than that of Dual-RLUT6. This is due to the more diverse combination of Boolean functions of two 6-CLUTs with partially complementary functionality, where only 9.76% 6-input Boolean functions cannot be implemented by either of them. Compared to Stratix 10 ALM, although Asym-CLUT6 cannot implement all 6-input functions, it covers a much larger portion of functions with 7 or 8 inputs, thereby reducing logic levels and the number of logic blocks.

D. Post-Synthesis and Post-Route Results

The post-synthesis and post-route results are shown in Fig. 8~11. *Lev* represents the number of PLBs on the critical path, while *Delay* refers to the post-route delay of the critical path. Compared to Stratix 10 ALM, Asym-CLUT6 reduces *Lev* by 21.63%. Therefore, although the PLB delay of Asym-CLUT6 is 6.39% higher than that of Stratix 10 ALM, *Delay* is reduced by 13.07%. *nCLB* is the number of CLBs in the post-route circuit, and *Area* represents the post-route logic area, including the CLB, BRAM, and MUX areas. Since the number of PLBs in the synthesized netlist of ABC does not account for the fracturing of PLBs, we evaluate the number of logic blocks with *nCLB*. Compared to Stratix 10 ALM, although Asym-CLUT6 has 9.38% more CLB area, *Area* still decreases by 0.67% due to a reduction of 8.94% in *nCLB* on average.

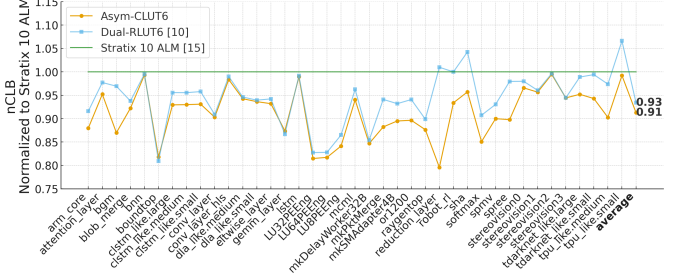


Fig. 10. Normalized results of *nCLB* with respect to Stratix 10 ALM.

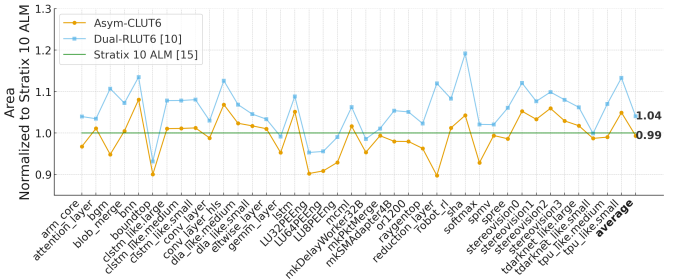


Fig. 11. Normalized results of *Area* with respect to Stratix 10 ALM.

Besides, compared to Dual-RLUT6, Asym-CLUT6 reduces *Lev* and *nCLB* by 6.56% and 2.36%. With 4.36% less CLB area, *Delay* and *Area* of Asym-CLUT6 are reduced by 5.79% and 4.54%. Therefore, the ADP of Asym-CLUT6 is reduced by 10.06% and 13.65% compared to Dual-RLUT6 [10] and Stratix 10 ALM [15]. These improvements primarily stem from the optimized area and stronger logic capacity of Asym-CLUT6. These show that Asym-CLUT6 achieves significant improvements in both delay and area, which demonstrates its potential for high-performance applications.

VI. CONCLUSION

This paper proposes a fine-grained customized LUT architecture CLUT, which is optimized through SRAM-MUX co-pruning. It is automatically designed based on target benchmarks with a novel function evaluation model and beam search algorithm. Furthermore, an 8-input PLB, Asym-CLUT6, composed of two 6-CLUTs with partially complementary functionality, is proposed. We evaluate CLUT on VTR and Koios benchmarks. Experimental results show that 6-CLUT with fewer SRAMs and pass transistors than that of 6-RLUT, has a larger logic capacity. Moreover, Asym-CLUT6 has a logic capacity comparable to that of an 8-LUT, with 4.36% less CLB area compared to Dual-RLUT6. Therefore, Asym-CLUT6 can reduce the average ADP after routing by 10.06% and 13.65%, compared to Dual-RLUT6 and Stratix 10 ALM.

VII. ACKNOWLEDGMENT

This work is supported by National Science and Technology Major Project (2021ZD0114701).

REFERENCES

- [1] C. Xu, S. Jiang, G. Luo, G. Sun, N. An, G. Huang, and X. Liu, "The case for fpga-based edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2610–2619, 2020.
- [2] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference (DAC) 2019*, 2019, pp. 1–6.
- [3] L.-C. Lu, "Physical design challenges and innovations to meet power, speed, and area scaling trend," in *Proceedings of the 2017 ACM on International Symposium on Physical Design (ISPD)*, 2017, pp. 63–63.
- [4] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, 2002.
- [5] Y. Okamoto, Y. Ichinomiya, M. Amagasaki, M. Iida, and T. Sueyoshi, "COGRE: A configuration memory reduced reconfigurable logic cell architecture for area minimization," in *2010 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2010, pp. 304–309.
- [6] J. H. Anderson, Q. Wang, and C. Ravishankar, "Raising FPGA logic density through synthesis-inspired architecture," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 3, pp. 537–550, 2011.
- [7] J. H. Anderson and Q. Wang, "Area-efficient FPGA logic elements: Architecture and synthesis," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2011, pp. 369–375.
- [8] I. Ahmadpour, B. Khaleghi, and H. Asadi, "An efficient reconfigurable architecture by characterizing most frequent logic functions," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–6.
- [9] W. Feng, J. Greene, and A. Mishchenko, "Improving FPGA performance with a S44 LUT structure," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, pp. 61–66.
- [10] M. Yang, C. Zeng, K. Zhu, and L. Wang, "RLUT: A Reduced LUT Architecture with Fine-Grained Scalability and Its Automatic Design Flow for Large Frequent Functions," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 18, no. 3, pp. 1–32, 2025.
- [11] S. Yazdanshenas and V. Betz, "COFFE 2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–27, 2019.
- [12] J. Cong and Y.-Y. Hwang, "Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1077–1090, 2001.
- [13] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker *et al.*, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 2, pp. 1–55, 2020.
- [14] A. Arora, A. Boutros, D. Rauch, A. Rajen, A. Borda, S. A. Damghani, S. Mehta, S. Kate, P. Patel, K. B. Kent *et al.*, "Koios: A deep learning benchmark suite for FPGA architecture and CAD research," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 355–362.
- [15] M. Eldafrawy, A. Boutros, S. Yazdanshenas, and V. Betz, "Fpga logic block architectures for efficient deep learning inference," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 3, pp. 1–34, 2020.
- [16] D. Chai and A. Kuehlmann, "Building a better Boolean matcher and symmetry detector," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*. IEEE, 2006, pp. 1–6.
- [17] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.
- [18] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2010, pp. 24–40.
- [19] A. M. R. Brayton, "Faster Logic Manipulation for Large Designs," in *Proceedings of the International Workshop on Logic and Synthesis (IWLS)*, vol. 13, 2013.
- [20] M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke, "Solving QBF with counterexample guided refinement," *Artificial Intelligence*, vol. 234, pp. 1–25, 2016.
- [21] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 2002.
- [22] A. C. Ling, D. P. Singh, and S. D. Brown, "FPGA PLB architecture evaluation and area optimization techniques using boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 7, pp. 1196–1210, 2007.
- [23] Intel Corporation, "Logic array blocks and adaptive logic modules user guide," 2017.
- [24] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free Verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, vol. 97, 2013.