High-accuracy Low-power Reconfigurable Architectures for Decomposition-based Approximate Lookup Table

Xingyue Qian[1], Chang Meng[1], Xiaolong Shen[2], Junfeng Zhao[2], Leibin Ni[2], and Weikang Qian[1,3]
[1]University of Michigan-SJTU Joint Institute and [3]MoE Key Lab of AI, Shanghai Jiao Tong University, Shanghai, China
[2]Central Research Institute, 2012 Laboratories, Huawei Technologies Co., Ltd., Shenzhen, China
Emails: {qianxingyue, changmeng}@sjtu.edu.cn, {shenxiaolong3, junfeng.zhao, nileibin}@huawei.com, qianwk@sjtu.edu.cn

***Abstract*—Storing pre-computed results of frequently-used functions into lookup table (LUT) is a popular way to improve energy efficiency, but its advantage diminishes as the number of input bits increases. A recent work shows that by decomposing the target function approximately, the total LUT entries can be dramatically reduced, leading to significant energy saving. However, its heuristic approximate decomposition algorithm leads to sub-optimal approximation quality. Also, its rigid hardware architecture only supports disjoint decomposition and may have unnecessary extra power consumption sometimes. To address these issues, we develop a novel approximate decomposition algorithm based on beam search and simulated annealing, which can reduce 11.1% approximation error. We also propose a non-disjoint approximate decomposition method and two reconfigurable architectures. The first has 10.4% less error using 19.2% less energy and the second has 23.0% less error with same energy consumption compared to the state-of-the-art design.**

I. Introduction

Energy efficiency is of increasing importance in computing systems as transistor size shrinks into nano-scale [1]. Computing with memory is a popular technique for low-energy design [2]. In this method, results of frequently-used functions are pre-computed and stored in a lookup table (LUT) so that they can be retrieved at runtime with low energy consumption [3]. A LUT with 2^n entries is needed to store a function with n input bits. Therefore, as the number of input bits increases, the advantage of computing with memory diminishes. To address this issue, approximate LUT is proposed [4]–[12]. For some error-tolerant applications, hardware cost can be dramatically reduced by carefully introducing errors, while the application-level quality remains almost unaffected.

Many methods design approximate LUT based on Taylor approximation [4]–[8]. Schulte and Stine divide the input into three segments and use their combinations to index two LUTs, whose outputs are added to give the approximate result [4]. Several later works extend the above architecture to more than two LUTs [5]–[8], but they all rely on Taylor approximation, so non-continuous functions are not supported. Another set of works are based on approximate input pattern matching [9], [10]. They store part of all the input-output pairs of a target function in the LUT. When querying an input X, if there exists an input $\hat{X}$ in the LUT close enough to X, the corresponding output $f(\hat{X})$ is returned to approximate $f(X)$. Otherwise, they still require to use conventional computing circuits to obtain $f(X)$. Tian *et al.* propose ApproxLUT that also stores the derivatives f' with the selected input-output pairs [11]. When querying an input X, ApproxLUT matches it with the closest stored input pattern $\hat{X}$ and returns $f(X)$ as $f(\hat{X})+f'(\hat{X})(X-\hat{X})$. However, it is also based on Taylor approximation. Thus, it cannot support non-continuous functions well.

Recently, Meng *et al.* implement approximate LUT from a different angle and propose DALTA, a Boolean decomposition-based approximate LUT architecture [12]. DALTA decomposes the target function into two simpler functions that are stored into two smaller LUTs. It supports both continuous and non-continuous functions and outperforms ApproxLUT in both energy and latency, which shows the potential of Boolean decomposition-based approximate LUT architecture. However, its heuristic approximate decomposition algorithm explores the solution space greedily, so the approximation quality is not optimized. Also, its rigid hardware architecture only supports disjoint decomposition and may have unnecessary extra power consumption sometimes.

In this work, we address the above issues, and our main contributions are as follows:

- We develop a novel approximate decomposition algorithm based on beam search and simulated annealing. It can find decomposition with higher accuracy. The experimental results show that compared to DALTA, it can reduce 11.1% error by using half runtime.
- We extend the disjoint decomposition method to a non-disjoint one, which can give better approximation result.
- We propose two reconfigurable architectures to improve the energy and accuracy and enable accuracy-energy trade-off. Compared to DALTA, the first has 10.4% less error using 19.2% less energy, while the second has 23.0% less error using the same amount of energy.

II. Background

A. Traditional Disjoint Decomposition

Definition 1 *Let f be a single-output Boolean function of n binary inputs $X = (x_n, \ldots, x_1)$. Let $\omega = (A, B)$ be a **partition** on X, i.e., $A \cup B = X$ and $A \cap B = \emptyset$. The function has a **disjoint decomposition** with **free set** A and **bound set** B if there exist functions ϕ and F such that $f(X) = F(\phi(B), A)$.*

Not all Boolean functions have a disjoint decomposition. Ashenhurst gives the condition on its existence as follows [13].

Theorem 1 *A Boolean function has a disjoint decomposition $F(\phi(B), A)$ if and only if there exists a fixed pattern V of 0s and 1s such that all the rows of the 2-dimensional (2D) truth table with its rows and columns defined by A and B, respectively, fit into the following types: 1) a pattern of all 0s; 2) a pattern of all 1s; 3) pattern V; 4) the complement of V.*

The type of each row forms a *type vector* T, while the fixed pattern V is called a *pattern vector*. The functions ϕ and F can be obtained from the vectors V and T, respectively.

This work is supported by the National Key R&D Program of China under grant number 2020YFB2205501. Corresponding author: Weikang Qian.

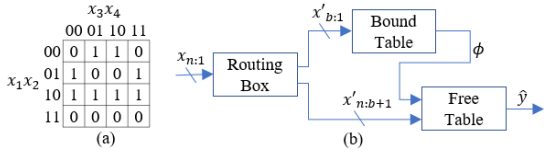

Fig. 1. (a) An example 2D truth table; (b) Approximate single-output LUT.

Example 1 *Fig. 1(a) shows the 2D truth table of a function f. Its rows and columns are respectively defined by the free set $A = \{x_1, x_2\}$ and the bound set $B = \{x_3, x_4\}$. With $V = (0, 1, 1, 0)$, the 4 rows of the truth table are of types 3, 4, 2, and 1, respectively. Thus, f has a disjoint decomposition $f = F(\phi(B), A)$. The corresponding type vector is $T = (3, 4, 2, 1)$. The function ϕ is given by the vector V as $\phi(x_3, x_4) = \bar{x}_3x_4 + x_3\bar{x}_4$. The function F is given by the vector T as $F(\phi, x_1, x_2) = \phi\bar{x}_1\bar{x}_2 + \phi\bar{x}_1x_2 + x_1\bar{x}_2$.*

B. DALTA

DALTA is a decomposition-based approximate lookup table architecture for an n-input m-output Boolean function $Y = G(X)$, where $X = (x_n, \ldots, x_1)$ and $Y = (y_m, \ldots, y_1)$ with $x_i, y_k \in \{0, 1\}$ [12]. G can be expressed as $G = (g_m, \ldots, g_1)$, where each component function g_k is an n-input single-output Boolean function that outputs y_k. DALTA approximates G by approximating each component function respectively, *i.e.*, $\hat{Y} = \hat{G}(X) = (\hat{g}_m(X), \ldots, \hat{g}_1(X))$, where $\hat{g}_k$ has a disjoint decomposition uniquely defined by its variable partition ω_k, pattern vector V_k, and type vector T_k.

DALTA uses an architecture shown in Fig. 1(b) to implement the approximate disjoint decomposition $\hat{g}_k$. It is called *approximate single-output LUT*, which has X as input and outputs one bit of the approximate output $\hat{Y}$, $\hat{y}_k$. The architecture has three main parts: routing box, bound table, and free table. Routing box shuffles the inputs X into X'. The last b bits of X', $x'_b, ..., x'_1$, form the bound set B and the other bits, $x'_n, ..., x'_{b+1}$, form the free set A. The bound table is implemented by a b-input LUT and realizes the function ϕ. The free table is implemented by an $(n - b + 1)$-input LUT and realizes the function F. To store the accurate component function g, a LUT of 2^n entries is needed. However, only $(2^b + 2^{n-b+1})$ LUT entries are needed to store $\hat{g}_k$, so the hardware cost is reduced dramatically. To approximate the m-bit output Y, m approximate single-output LUTs are implemented, and the outputs $\hat{y}_k$'s are combined as $\hat{Y}$.

DALTA applies a heuristic algorithm to derive a good approximate disjoint decomposition $\hat{g}_k$ with a small error. The error is measured by *mean error distance* (*MED*), defined as $MED(G, \hat{G}) = \sum_X p_X \cdot \left| Bin\left(G(X)\right) - Bin\left(\hat{G}(X)\right) \right|$, where p_X is the occurrence probability of input X, and $Bin(W)$ gives the binary number encoded by the Boolean vector W.

The algorithm optimizes the setting of each $\hat{g}_k$, *i.e.*, ω_k, V_k, and T_k, from the most significant bit (MSB) to the least significant bit (LSB) in turn for R rounds. When optimizing $\hat{g}_k$, the algorithm fixes all the other approximate component functions to their accurate versions if they have not been optimized yet in the first round, or their latest optimized versions otherwise. Since the number of all possible partitions grows exponentially with the number of input bits, to be practical, when optimizing $\hat{g}_k$, the algorithm randomly generates P candidate partitions ω_k's for consideration and greedily picks the one with the smallest MED. For a given partition ω_k, a function *OptForPart*$(G, \hat{G}, k, \omega_k)$ is used to optimize the MED and get the corresponding pattern vector V_k and type vector T_k. Specifically, in *OptForPart*, two 2D truth tables are first generated based on G and $\hat{G}$ respectively according to the partition ω_k. Then, Z initial pattern vectors V_{ini}'s are randomly generated. For each V_{ini}, a local optimal (V, T) pair is reached by alternatively fixing one element of (V, T) and optimizing the other. Finally, the best of the Z local optimums is returned as the best (V, T) pair for the partition ω_k.

III. Improved Approximate Decomposition Algorithm

DALTA's approximate decomposition algorithm explores the solution space greedily, so the approximation quality is not optimized. Here, we propose an improved algorithm based on beam search and simulated annealing, *BS-SA*, which can find a better decomposition quickly. We first present the beam searched-based overall flow. Then, we show an important improvement in the first round of the flow, that is, when optimizing an approximate component function, how to set the other unknown approximate component functions. Finally, we describe our simulated annealing-based algorithm that finds an optimized setting for each approximate component function.

A. Beam Search-based Overall Flow

We call $s = (E, \omega, V, T)$ a decomposition setting of an approximate component function $\hat{g}$, where E is the MED, ω is the partition, and V and T are the pattern vector and type vector, respectively. Since s can uniquely define $\hat{g}$, the aim of the approximate decomposition algorithm is to find an optimized setting sequence, *i.e.*, $S^* = (s^*_m, ..., s^*_1)$ that defines an approximate function $\hat{G}$ minimizing $MED(G, \hat{G})$.

Algorithm 1: Beam search-based flow.

Input: G: accurate function; R: iteration round; N_{beam}: number of beams.
Output: $S^* = (s^*_m, ..., s^*_1)$: an optimized setting sequence.

```
1  Get a random sequence S; Set of top sequences 𝕊_b ← {S};
2  for output bit k ← m to 1 do
3      𝕊 ← ∅;
4      for S = (s_m, ..., s_k, ..., s_1) ∈ 𝕊_b do
5          Ĝ ← GetApproxFunction(S);
6          𝔹_s ← FindBestSettings(G, Ĝ, k, N_beam);
7          for s ∈ 𝔹_s do
8              𝕊 ← 𝕊 ∪ {(s_m, ..., s_{k+1}, s, s_{k-1}, ..., s_1)};
9      𝕊_b ← FindTops(𝕊, N_beam);
10 S* = (s*_m, ..., s*_1) ← best sequence in 𝕊_b;
11 for round ← 2 to R do
12     for output bit k ← m to 1 do
13         Ĝ ← GetApproxFunction(S*);
14         {s} ← FindBestSettings(G, Ĝ, k, 1);
15         S* ← (s*_m, ..., s*_{k+1}, s, s*_{k-1}, ..., s*_1);
16 return S*;
```

The flow of our proposed approximate decomposition algorithm is shown in Algorithm 1. Lines 1–10 correspond to the first round, while Lines 11–15 correspond to the later rounds. DALTA greedily picks the best setting for each $\hat{g}_k$ in turn in each round. However, we observed that the only greedy choice at each output bit narrows down the search space of

the rest bits significantly, which may lead to a sub-optimal result. Besides, we observed that the overall approximation performance is largely determined in the first round. Therefore, in the first round, instead of greedily picking the best setting for each $\hat{g}_k$, we apply beam search [14], which extends the greedy search by maintaining the top $N_{beam} > 1$ best settings.

Specifically, in the first round, Line 1 puts an arbitrary setting sequence S into the set of top decomposition setting sequences $\mathbb{S}_b$. Since we propose a model to be introduced in Section III-B to predict the LSBs that have not been approximated in the first round, the choice of S does not affect the behavior of the algorithm. Line 2 traverses the output bit from the MSB to the LSB. When approximating output bit k, Line 4 further traverses each sequence S in $\mathbb{S}_b$. For a sequence S, Line 5 first obtains the corresponding approximate function $\hat{G}$. Then, Line 6 calls the function *FindBestSettings* to generate a set $\mathbb{B}_s$, which contains the top N_{beam} best decomposition settings of the current $\hat{g}_k$. *FindBestSettings* is a simulated annealing-based algorithm and will be introduced in Section III-C. Finally, for each top decomposition setting s in $\mathbb{B}_s$, Line 8 sets the setting of k-th output bit in S, s_k, to be s, and puts the updated S into $\mathbb{S}$. After all the sequences in $\mathbb{S}_b$ are checked, Line 9 updates $\mathbb{S}_b$ with N_{beam} sequences with the least errors in $\mathbb{S}$, which will be used when approximating the next output bit. After the last bit is approximated, Line 10 sets S^* to be the best sequence in $\mathbb{S}_b$, and the later rounds further optimize it. For later rounds (Lines 11–15), we obtain $\hat{G}$ using the best sequence S^*, and only the best decomposition setting s for the k-th ($1 \le k \le m$) output bit is obtained and used to update the setting of that bit.

B. Predictive Model for the LSBs

In the algorithm when optimizing $\hat{g}_k$, the other approximate component functions are assumed to have been known. However, the LSBs, *i.e.*, $\hat{g}_{k-1}, \ldots, \hat{g}_1$ are unknown in the first round. DALTA solves this issue by setting them as the accurate version [12]. However, in reality, the optimization algorithm tends to minimize the overall error. Thus, a better model is to predict their behavior in minimizing the error. For this purpose, we introduce a predictive model that sets the unknown bits to the ones that minimize the error. Specifically, assume that we are optimizing the k-th output bit. At this moment, by the optimization order shown at Line 2 in Algorithm 1, for each input X, the output bits $\hat{y}_m, \ldots, \hat{y}_{k+1}$ are known, while $\hat{y}_{k-1}, \ldots, \hat{y}_1$ are unknown. For a fixed choice of the k-th output bit $\hat{y}_k \in \{0, 1\}$, we define $\hat{Y}_M = \sum_{j=k}^{m} 2^{j-1}\hat{y}_j$. Based on the target output bits $y_m, \ldots, y_k$ for the input X, we define $Y_M = \sum_{j=k}^{m} 2^{j-1} y_j$. In order to minimize the error distance $|Y - \hat{Y}|$ for the input X and the choice $\hat{y}_k$, we set the unknowns $\hat{y}_{k-1}, \ldots, \hat{y}_1$ by distinguishing the following three cases:

1) The case where $\hat{Y}_M > Y_M$. In this case, to minimize $|Y - \hat{Y}|$, we should let $\hat{y}_j = 0$ for $1 \le j \le k-1$.
2) The case where $\hat{Y}_M < Y_M$. In this case, to minimize $|Y - \hat{Y}|$, we should let $\hat{y}_j = 1$ for $1 \le j \le k-1$.
3) The case where $\hat{Y}_M = Y_M$. In this case, to minimize $|Y - \hat{Y}|$, we should let $\hat{y}_j = y_j$ for $1 \le j \le k-1$.

C. Simulated Annealing-based FindBestSettings

When DALTA optimizes $\hat{g}_k$, it randomly picks P partitions from all possible choices for consideration, which leads to a sub-optimal result [12]. In our BS-SA algorithm, we use simulated annealing (SA) in *FindBestSettings* to find the best settings for $\hat{g}_k$. The general process is shown in Algorithm 2.

Algorithm 2: SA-based *FindBestSettings*.

Input: G: accurate function; $\hat{G}$: approximate function; k: output bit; N_{beams}: number of beams.
Output: $\mathbb{B}_s$: the set of top N_{beams} best decomposition settings of $\hat{g}_k$.
Parameter: P: variable partition limit; N_{nb}: number of neighbours; τ_0: initial temperature; α: temperature decrease factor.

1 Randomly generate variable partition ω; $\tau \leftarrow \tau_0$;
2 $(E_\omega, V, T) \leftarrow$ *OptForPart*$(G, \hat{G}, k, \omega)$; Best error $E^* \leftarrow E_\omega$;
3 Set of visited partitions $\Phi \leftarrow \{\omega\}$; $\mathbb{B}_s \leftarrow \{(E_\omega, \omega, V, T)\}$;
4 **while** $|\Phi| < P$ **do**
5 Best neighbour error $E^*_{nb} \leftarrow \infty$; $\Omega \leftarrow$ *GenNeib*(ω, N_{nb});
6 **for** $\omega_{nb} \in \Omega$ **do**
7 **if** $\omega_{nb} \notin \Phi$ **then**
8 $(E_{\omega_{nb}}, V, T) \leftarrow$ *OptForPart*$(G, \hat{G}, k, \omega_{nb})$;
9 $\Phi \leftarrow \Phi \cup \{\omega_{nb}\}$; Store $E_{\omega_{nb}}$ for ω_{nb};
10 **if** $E_{\omega_{nb}} < E^*$ **then** $E^* \leftarrow E_{\omega_{nb}}$;
11 $\mathbb{B}_s \leftarrow \mathbb{B}_s \cup \{(E_{\omega_{nb}}, \omega_{nb}, V, T)\}$;
12 **if** $|\mathbb{B}_s| > N_{beam}$ **then**
13 Delete the entry in $\mathbb{B}_s$ with the largest E_ω;
14 **else** Fetches stored $E_{\omega_{nb}}$ for ω_{nb};
15 **if** $E_{\omega_{nb}} < E^*_{nb}$ **then** $(\omega^*_{nb}, E^*_{nb}) \leftarrow (\omega_{nb}, E_{\omega_{nb}})$;
16 **if** $E^*_{nb} \le E_\omega$ **then** $(\omega, E_\omega) \leftarrow (\omega^*_{nb}, E^*_{nb})$;
17 **else if** *rand*$() < \exp(\frac{E_\omega - E^*_{nb}}{\tau E^*})$ **then** $(\omega, E_\omega) \leftarrow (\omega^*_{nb}, E^*_{nb})$;
18 $\tau \leftarrow \alpha\tau$;
19 **if** *no change of* Φ *in* 3 *successive iterations* **then break**;
20 **return** $\mathbb{B}_s$;

Lines 1–3 obtain an optimized setting of a randomly-generated initial partition ω by calling the function *OptForPart* introduced in Section II-B, where E_ω denotes the corresponding error for ω. In addition, Line 1 sets the temperature τ as an initial value τ_0. Lines 4–19 are the main loop, in which the current partition ω may be updated. Within the loop, Line 5 first generates N_{nb} neighbours of ω. For a variable partition ω with free set A and bound set B, another partition ω_{nb} with free set A_{nb} and bound set B_{nb} is called its *neighbour* if A and A_{nb} differ in one element.

Lines 6–15 check each neighbour ω_{nb} to get the best one, ω^*_{nb}. If ω_{nb} is a new partition, Line 8 calls *OptForPart* to obtain an optimized setting $(E_{\omega_{nb}}, V, T)$ for it. Then, Line 9 adds ω_{nb} into the set of visited partitions, Φ, and stores the associated error $E_{\omega_{nb}}$. If $E_{\omega_{nb}}$ is less than the global best error E^*, Line 10 updates E^* by $E_{\omega_{nb}}$. Lines 11–13 update the set $\mathbb{B}_s$, which contains the top N_{beam} global best settings. If ω_{nb} has already been visited, Line 14 fetches its associated error $E_{\omega_{nb}}$ directly. Then, Line 15 updates the best neighbour partition ω^*_{nb} and its associated error E^*_{nb} with the current neighbour ω_{nb} and its associated error $E_{\omega_{nb}}$, respectively, if $E_{\omega_{nb}} < E^*_{nb}$.

If the error of the best neighbour is smaller than that of the current partition, Line 16 updates the current partition ω as the best neighbour. Otherwise, Line 17 does the same update with a probability related to the normalized error difference and the current temperature. Line 18 reduces the current temperature τ with a scaling factor $\alpha \in (0, 1)$. If at least P partitions are visited (Line 4) or the set of visited partitions Φ does not change for 3 successive iterations (Line 19), the process ends.

IV. Reconfigurable Hardware Architectures

This section presents two proposed reconfigurable architectures. They both support the basic disjoint decomposition mode same as DALTA, which we call the *normal mode*. In addition, the first architecture supports a *bound-table-only* (*BTO*) mode to further reduce power consumption, while the second supports both the BTO mode and a *non-disjoint decomposition* (*ND*) mode to improve the accuracy.

A. Reconfigurable Architecture with BTO and Normal Mode

We observe that for some output bits in some functions, the minimum error when all rows in the 2D truth table are fixed to type 3 is close to the original minimum error achieved by choosing the best type from type 1 to type 4 for each row. In this case, the approximate function $F(\phi(B), A)$ is independent of the free set A, and it reduces to $\phi(B)$. This implies that in the approximate single-output LUT shown in Fig. 1(b), we can turn off the free table and directly use the output of the bound table, ϕ, thus saving the power consumption. This mode is called the BTO mode. To get the best pattern vector under the BTO mode, we only need to modify the function *OptForPart* by restricting the type vector T as a vector of all 3s.

Example 2 *For the example function shown in Fig. 2(a), it can be accurately decomposed with* $V = (1, 1, 1, 0)$ *and* $T = (3, 2, 3, 3)$. *However, if we restrict that all the rows should be of type 3, then the closest approximate decomposition has its pattern vector as* $(1, 1, 1, 0)$, *corresponding to the function* $\phi(x_3, x_4) = \bar{x}_3\bar{x}_4 + \bar{x}_3x_4 + x_3\bar{x}_4$. *It is very close to the accurate decomposition: only the red cell in Fig. 2(a) is wrong. Meanwhile, it does not need to use the free table.*

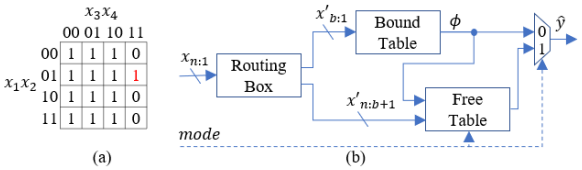

Fig. 2. (a) An example 2D truth table; (b) BTO-Normal architecture.

To support both the power-saving BTO mode and the more accurate normal mode, we propose a reconfigurable architecture as shown in Fig. 2(b). We call it *BTO-Normal*. When the signal *mode* $= 0$, the architecture operates in the *BTO* mode, where the free table is turned off by setting its enable and clock signals to zero, hence saving the dynamic power and the total power consumption. When *mode* $= 1$, the architecture operates in the normal mode and the free table is activated as usual. The approximate output bit $\hat{y}$ is selected from the outputs of the bound and free tables by the signal *mode* through a MUX.

When using BTO-Normal to implement a function, we also need to decide the operating mode of each output bit. For this purpose, at Line 14 of Algorithm 1, besides obtaining the best setting s for the normal mode, we also obtain the best setting s_{BTO} when using the BTO mode. Let the MEDs of the settings s and s_{BTO} be E and E_{BTO}, respectively. For the current output bit, we choose the BTO mode if $E_{BTO} < (1 + \delta)E$, where $\delta > 0$ is a factor controlling the mode selection; otherwise, we choose the normal mode.

B. Reconfigurable Architecture with BTO, Normal, and ND Modes

This section presents the architecture with the BTO, normal, and ND modes denoted as *BTO-Normal-ND*. We first show the approximate non-disjoint decomposition algorithm and architecture. Then, we show the reconfigurable architecture.

1) Approximate Non-disjoint Decomposition: The approximation based on disjoint decomposition is restrictive and sometimes, can lead to a large error. In order to reduce the error, we resort to non-disjoint decomposition. A non-disjoint decomposition can be represented as $f(X) = F(\phi(B), A, C)$, where (A, B) form a partition of X and C is a subset of B. We limit to the case where C contains a single input in set B so that the hardware cost is not increased too much. Let that input be $x_s \in B$. Then, we can rewrite the non-disjoint decomposition as $F(\phi(B), A, x_s)$. For $j = 0, 1$, define $F_j(\phi, A) = F(\phi, A, j)$. By distinguishing between $x_s = 0$ and $x_s = 1$, we can prove:

$$f(X) = F(\phi(B), A, x_s) = \bar{x}_s F_0(\phi(B), A) + x_s F_1(\phi(B), A). \quad (1)$$

Based on Eq. (1), we can realize a non-disjoint decomposition by an architecture shown in Fig. 3(a), which serves as a foundation for our proposed reconfigurable architecture that will be introduced next. The architecture consists of a bound table that realizes ϕ, a free table (Free Table 0) that realizes F_0, a free table (Free Table 1) that realizes F_1, and a MUX that uses x_s to select from F_0 and F_1.

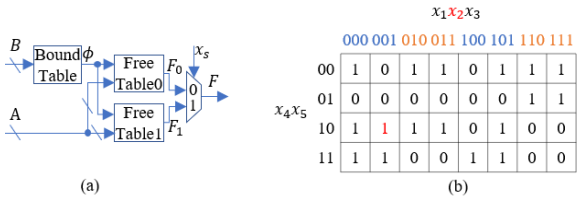

Fig. 3. (a) An architecture for realizing a non-disjoint decomposition; (b) An example of approximate non-disjoint decomposition.

Next, we illustrate given a single-output target function $t(X)$ and a variable partition $\omega = (A, B)$, how to obtain a single-output approximate function $f(X)$ with a non-disjoint decomposition in the form of $F(\phi(B), A, x_s)$ so that $MED(t, f)$ is minimized. Note that in the actual situation, we try to find an approximate non-disjoint decomposition for the k-th ($1 \le k \le m$) LSB of G so that $MED(G, \hat{G})$ is minimized. The solution to this actual problem shares the same basic idea as the solution to the above problem, but due to space limit, we omit the details.

First, we assume that the shared bit is fixed as $x_s \in B$. Define $\mathbb{B} = B \backslash \{x_s\}$ and $\mathbb{X} = X \backslash \{x_s\}$. We rewrite $t(X)$ as $t(\mathbb{X}, x_s)$, $f(X)$ as $f(\mathbb{X}, x_s)$, and $\phi(B)$ as $\phi(\mathbb{B}, x_s)$. For $j = 0, 1$, define $t_j(\mathbb{X}) = t(\mathbb{X}, j)$, $f_j(\mathbb{X}) = f(\mathbb{X}, j)$, and $\phi_j(\mathbb{B}) = \phi(\mathbb{B}, j)$. We use $\mathbb{P}(\Omega)$ to denote the probability of event Ω. By definition, we have

$$\begin{aligned} MED(t, f) &= \sum_X \mathbb{P}(X)|t(X) - f(X)| \\ &= \sum_{\mathbb{X}} \mathbb{P}(\mathbb{X}, x_s = 0)|t_0(\mathbb{X}) - f_0(\mathbb{X})| \\ &+ \sum_{\mathbb{X}} \mathbb{P}(\mathbb{X}, x_s = 1)|t_1(\mathbb{X}) - f_1(\mathbb{X})| \end{aligned} \quad (2)$$

$$= \mathbb{P}(x_s = 0) \sum_{\mathbb{X}} \mathbb{P}(\mathbb{X}|x_s = 0)|t_0(\mathbb{X}) - f_0(\mathbb{X})|$$
$$+ \mathbb{P}(x_s = 1) \sum_{\mathbb{X}} \mathbb{P}(\mathbb{X}|x_s = 1)|t_1(\mathbb{X}) - f_1(\mathbb{X})|.$$

Since $f(X) = F(\phi(B), A, x_s)$, we have that for $j = 0, 1$, $f_j(\mathbb{X}) = F(\phi_j(\mathbb{B}), A, j) = F_j(\phi_j(\mathbb{B}), A)$. Clearly, $(\mathbb{B}, A)$ forms a partition on the variable set $\mathbb{X}$. Thus, $F_j(\phi_j(\mathbb{B}), A)$ $(j = 0, 1)$ is a disjoint decomposition. By Eq. (2), to minimize $MED(t, f)$, it is equivalent to finding two disjoint decompositions $f_0(\mathbb{X}) = F_0(\phi_0(\mathbb{B}), A)$ and $f_1(\mathbb{X}) = F_1(\phi_1(\mathbb{B}), A)$ to minimize $MED(t_0, f_0)$ and $MED(t_1, f_1)$, respectively, under the corresponding conditional probability distribution on $\mathbb{X}$, which can be obtained from the probability distribution on X. We can solve these two disjoint decompositions by calling the function *OptForPart*. After that, we obtain F_0, F_1, and $\phi(B) = \bar{x}_s\phi_0(\mathbb{B}) + x_s\phi_1(\mathbb{B})$. Finally, we can get the non-disjoint decomposition according to Eq. (1).

Example 3 *Fig. 3(b) shows an example for a single-output function t on five inputs under the variable partition $A = \{x_4, x_5\}$ and $B = \{x_1, x_2, x_3\}$. Assume that we want to get an approximate non-disjoint decomposition $f(X) = F(\phi(B), A, x_2)$ that minimizes $MED(t, f)$. For the shared bit x_2, t_0 and t_1 are defined by the blue and orange columns in the table, respectively. Assume that the inputs are uniformly distributed. We first obtain the disjoint decompositions $f_0 = F_0(\phi_0(x_1, x_3), x_4, x_5)$ that minimizes $MED(t_0, f_0)$ and $f_1 = F_1(\phi_1(x_1, x_3), x_4, x_5)$ that minimizes $MED(t_1, f_1)$. We get $\phi_0(x_1, x_3) = \bar{x}_1\bar{x}_3 + x_1x_3$, $F_0(\phi, x_4, x_5) = \phi\bar{x}_4\bar{x}_5 + \phi x_4\bar{x}_5 + x_4x_5$, $\phi_1(x_1, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_1x_3$, and $F_1(\phi, x_4, x_5) = \bar{x}_4\bar{x}_5 + \phi\bar{x}_4x_5 + \phi x_4\bar{x}_5$. Finally, we get the non-disjoint decomposition $F(\phi(B), A, x_2)$ as $F(\phi, x_4, x_5, x_2) = \bar{x}_2F_0(\phi, x_4, x_5) + x_2F_1(\phi, x_4, x_5)$ and $\phi(x_1, x_2, x_3) = \bar{x}_2\phi_0(x_1, x_3) + x_2\phi_1(x_1, x_3)$.*

Note that in the above illustration, we assume that the share bit x_s is known. If real case, it is unknown. To solve this problem, we enumerate all the bits in B and finally pick the one with the least MED.

2) BTO-Normal-ND Architecture: The proposed reconfigurable BTO-Normal-ND architecture is shown in Fig. 4, which is based on the non-disjoint decomposition architecture in Fig. 3(a). Since the input goes through a routing box, we can always rearrange the bound set B to let the selected shared bit x_s be x'_b. The mode of the architecture is determined by the signals $mode_1$ and $mode_2$. When $(mode_2, mode_1) = (0, 0)$, both free tables are off, and the architecture is in the BTO mode. When $(mode_2, mode_1) = (0, 1)$, Free Table 1 is off, Free Table 0 is on, and the architecture is in the normal mode. When $(mode_2, mode_1) = (1, 1)$, both free tables are on. In this case, the block connection is the same as that shown in Fig. 3(a), and hence, the architecture is in the ND mode.

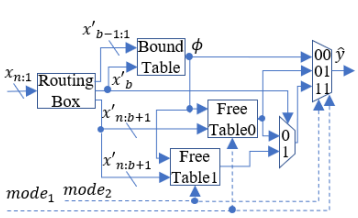

Fig. 4. BTO-Normal-ND architecture.

The mode selection is similar to that of BTO-Normal. At Line 14 of Algorithm 1, besides obtaining the best setting s for the normal mode, we obtain the best setting s_{BTO} when using the BTO mode and the best setting s_{ND} when using the ND mode. Let the MEDs of the settings s, s_{BTO}, and s_{ND} be E, E_{BTO}, and E_{ND}, respectively. For the current output bit, we choose the BTO mode if $E_{BTO} < (1 + \delta)E$ and $E_{ND} > (1 - \delta')E$, where δ and δ' are two parameters satisfying that $0 < \delta < \delta' < 1$. Otherwise, if $E_{ND} < (1 - \delta)E$, we choose the ND mode. For the rest case, we choose the normal mode.

V. Experimental Results

This section shows the experimental results. We implement the algorithms in C++ and perform experiments on a computer with 48-core 2.4GHz Intel 4214R processors and 64GB RAM. All the architectures are implemented in Verilog. LUTs are implemented by RAMs consisting of D flip-flops. The architectures are further synthesized with Synopsys Design Compiler (DC) [15] using the Nangate 45nm standard cell library [16]. DC is also used for measuring areas and delays. The functionality is verified by Synopsys VCS [15], and the power is evaluated by Synopsys PrimeTime [15].

We use the same benchmarks as DALTA [12] in Table I. The 6 continuous functions are from [11], whose inputs and outputs are both quantized into 16 bits. The 4 non-continuous functions are from AxBench [17], whose 16-bit inputs are stitched by two 8-bit operands of the original functions. We assume that all inputs are uniformly distributed.

Table I. Benchmarks used in the experiments.

Continuous	Domain	Range	Non-continuous	#input	#output
$\cos(x)$	$[0, \frac{\pi}{2}]$	$[0, 1]$	Brent-Kung	16	9
$\tan(x)$	$[0, \frac{2\pi}{5}]$	$[0, 3.08]$	Forwardk2j	16	16
$\exp(x)$	$[0, 3]$	$[0, 20.09]$	Inversek2j	16	16
$\ln(x)$	$[1, 10]$	$[0, 2.30]$	Multiplier	16	16
$\text{erf}(x)$	$[0, 3]$	$[0, 1]$			
$\text{denoise}(x)$	$[0, 3]$	$[0, 0.81]$			

A. Performance of BS-SA Algorithm

This section studies the performance of the BS-SA algorithm by comparing it with the DALTA algorithm. Since DALTA only supports the normal mode, we only test this mode of BS-SA. Both DALTA and BS-SA spend most of their runtime in calling the function *OptForPart*. To reduce the runtime, we call the function for different partitions in parallel with 44 threads for both algorithms. In order to sufficiently use these threads, in the actual implementation of BS-SA, for each function call of *FindBestSettings*, we run 10 SA-based processes simultaneously that share a common set of visited partitions Φ. For both DALTA and BS-SA, we set the bound set size $b = 9$, the iteration round $R = 5$, and the number of initial patterns $Z = 30$, which are same as those in [12]. In BS-SA, we set the beam number $N_{beam} = 3$, the number of neighbours $N_{nb} = 5$, the initial temperature $\tau_0 = 0.2$, and the decrease factor $\alpha = 0.9$. Since the SA algorithm can choose more promising variable partitions, we set the partition limit P for BS-SA as 500, while that for DALTA is 1000.

We run both DALTA's algorithm and BS-SA for 10 times and obtain the minimum, average, and standard deviation of MED and the average runtime, as listed in Table II. Note that the minimum MEDs for DALTA's algorithm for several benchmarks are much smaller than those reported in [12]

because they are the best results in 10 runs. Comparing the geometric means of the metrics, BS-SA can reduce 11.1% minimum error and 97.1% standard deviation over DALTA by using half of its runtime.

Table II. Comparison of DALTA's algorithm and BS-SA.

	DALTA				BS-SA			
	MED			Time (s)	MED			Time (s)
benchmark	Min	Avg	Stdev	Avg	Min	Avg	Stdev	Avg
cos	9.47	10.50	0.88	424	8.66	8.80	0.14	202
tan	3.13	3.61	1.32	352	3.09	3.38	0.16	168
exp	9.06	15.92	8.04	433	8.76	8.87	0.05	189
ln	10.39	13.94	6.48	442	10.03	10.10	0.04	190
erf	13.97	46.68	18.99	391	10.93	10.99	0.10	185
denoise	9.49	29.01	10.10	446	8.77	8.84	0.14	199
Brent-Kung	0.09	0.85	0.70	148	0.06	0.31	0.21	76
Forwardk2j	586.9	870.9	199.0	650	581.0	624.9	64.58	327
Inversek2j	325.9	425.0	116.4	643	299.8	300.8	1.94	264
Multiplier	414.2	584.2	146.1	682	318.5	319.2	0.74	274
GEOMEAN	17.39	34.04	11.07	429	15.46	18.53	0.32	195

B. Performance of Reconfigurable Hardware Architectures

To show the performance of BTO-Normal and BTO-Normal-ND, we compare them with DALTA and two rounding-based architectures, *RoundOut* and *RoundIn*. *RoundOut* rounds off the q LSBs of the output and keeps the rest. We adjust q for each benchmark so that the resulting MED is larger than that of DALTA. *RoundIn* rounds off w bits of the input and uses the rest to look up for the approximate output. Specifically, we partition the inputs into blocks of 2^w adjacent ones and use the median output in each block as the approximate output of all the inputs in the block. We set $w = 6$ to obtain a comparable MED.

We use the best results in the 10 runs, *i.e.*, the ones that give column 2 in Table II, to configure DALTA. For BTO-Normal and BTO-Normal-ND, we run BS-SA only once thanks to its high stability. We set the mode selection factor $\delta = 0.01$ and $\delta' = 0.1$. We set the same delay constraint for *RoundIn*, DALTA, BTO-Normal, and BTO-Normal-ND during synthesis so that they have similar latency. For each benchmark, we measure the energy for 1024 read operations and record their average. For MED, area, latency, and energy, we take their geometric means over all 10 benchmarks and normalize the result to that of DALTA as shown in Fig. 5. We can see that the three decomposition-based architectures, *i.e.*, DALTA, BTO-Normal, and BTO-Normal-ND, have smaller error and energy than the two rounding-based ones. Compared to DALTA, BTO-Normal has 10.4% less error due to the use of BS-SA and 19.2% less energy due to the use of BTO mode for some output bits. BTO-Normal-ND improves DALTA's error by 23.0% due to the availability of the ND mode. It has almost the same energy as DALTA, but it requires 29% more area due to the extra free tables.

C. Accuracy-Energy Trade-off of BTO-Normal-ND

We perform a case study on the widely-used cosine function using BTO-Normal-ND. By selecting different modes for each output bit, we can obtain configurations with accuracy-energy trade-off as shown in Fig 6. We label the number of output bits in each mode, *i.e.*, $(\#BTO, \#Normal, \#ND)$, for two configurations. All the 6 configurations between these two inclusively have both less error and less energy than DALTA.

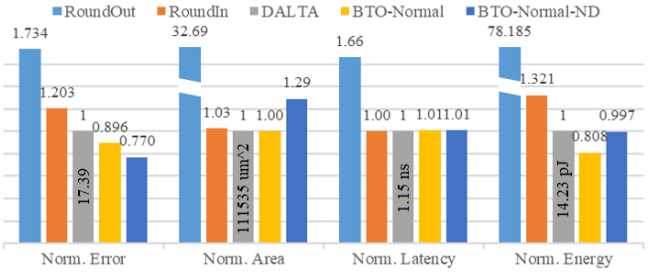

Fig. 5. Performance comparison of hardware architectures.

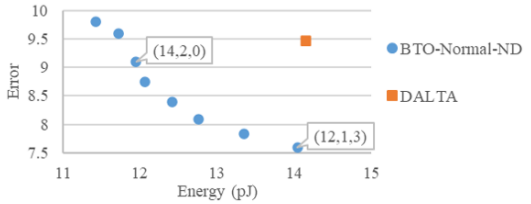

Fig. 6. Accuracy-energy trade-off of the function $\cos(x)$.

VI. Conclusion

In this work, we propose two high-accuracy low-power reconfigurable architectures for decomposition-based approximate LUT. We also develop a novel approximate decomposition algorithm based on beam search and simulated annealing, which can quickly give optimized configurations. The experiments show that the proposed architectures outperform the state of the art.

References

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, 2016.
[2] P. T. P. Tang, "Table-lookup algorithms for elementary functions and their error analysis," in *ARITH*, 1991, pp. 232–236.
[3] J. Cong *et al.*, "Energy-efficient computing using adaptive table lookup based on nonvolatile memories," in *ISLPED*, 2013, pp. 280–285.
[4] M. J. Schulte and J. E. Stine, "Symmetric bipartite tables for accurate function approximation," in *ARITH*, 1997, pp. 175–183.
[5] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *VLSI signal proc.*, vol. 21, no. 2, pp. 167–177, 1999.
[6] J.-M. Muller, "A few results on table-based methods," *Reliab. Comput.*, vol. 5, no. 3, pp. 279–288, 1999.
[7] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Trans Comput*, vol. 54, no. 3, pp. 319–330, 2005.
[8] S. Hsiao *et al.*, "Hierarchical multipartite function evaluation," *IEEE Trans Comput*, vol. 66, no. 1, pp. 89–99, 2017.
[9] A. Rahimi, "Approximate associative memristive memory for energy-efficient GPUs," in *DATE*, 2015, pp. 1497–1502.
[10] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *DATE*, 2016, pp. 1327–1332.
[11] Y. Tian *et al.*, "ApproxLUT: A novel approximate lookup table-based accelerator," in *ICCAD*, 2017, pp. 438–443.
[12] C. Meng *et al.*, "DALTA: A decomposition-based approximate lookup table architecture," in *ICCAD*, 2021, pp. 1–8.
[13] R. L. Ashenhurst, "The decompositions of switching functions," in *ISTSF*, 1959, pp. 74–116.
[14] C. Sammut, *Beam Search*. Springer US, 2017, pp. 120–120.
[15] Synopsys, Inc., "Synopsys softwares," 2021. [Online]. Available: http://www.synopsys.com
[16] Nangate, Inc., "Nangate 45nm open cell library," 2021. [Online]. Available: https://si2.org/open-cell-library/
[17] A. Yazdanbakhsh *et al.*, "AxBench: A multi-platform benchmark suite for approximate computing," in *IEEE Des. Test*, 2016.