

Timing-driven Approximate Logic Synthesis Based on Double-chase Grey Wolf Optimizer

Xiangfei Hu¹, Yuyang Ye², Tinghuan Chen^{2,3}, Hao Yan¹, Bei Yu²

¹Southeast University ²CUHK ³CUHK-Shenzhen

Abstract—With the shrinking technology nodes, timing optimization becomes increasingly challenging. Approximate logic synthesis (ALS) can perform local approximate changes (LACs) on circuits to optimize timing with the cost of slight inaccuracy. However, existing ALS methods that focus solely on critical path depth reduction (depth-driven methods) or area minimization (area-driven methods) are inefficient in achieving optimal timing improvement. In this work, we propose an effective timing-driven ALS framework, where we employ a double-chase grey wolf optimizer to explore and apply LACs, simultaneously bringing excellent critical path shortening and area reduction under error constraints. Subsequently, it utilizes post-optimization under area constraints to convert area reduction into further timing improvement, thus achieving maximum critical path delay reduction. According to experiments on open-source circuits with TSMC 28nm technology, compared to the SOTA method, our framework can generate approximate circuits with greater critical path delay reduction under different error and area constraints.

I. INTRODUCTION

Timing optimization is crucial in VLSI design. As the CMOS technology nodes continue to shrink, timing improvements caused by traditional methods, including gate sizing and logic restructure, are limited [1], [2]. In recent years, error-tolerant applications are becoming increasingly popular. Consequently, approximate computing [2], which effectively balances accuracy and performance, has garnered great attention. It can significantly reduce circuit delay, area, and power with the cost of slight computational imprecision.

Recently, approximate logic synthesis (ALS) has been proposed as an automated approximate computing paradigm. It can optimize timing under a relaxed error bound by reducing the depth of critical paths and enhancing the drive strength of gates on critical paths [3]. Based on optimization approaches, existing ALS methods can primarily be divided into two categories: (1) depth-driven methods [4]–[6] and (2) area-driven methods [7]–[10]. Depth-driven methods perform local approximate changes (LACs) to simplify gates on critical paths, providing direct timing improvement. As shown in Fig. 1, LACs are applied to critical paths 1 and 2. By omitting certain gates, both paths become shallower and faster with the cost of a slight error. HEDALS [6] proposes a critical error graph to accelerate critical path depth reduction and strictly control the introduced errors. Area-driven methods select LACs with the best area reduction potential to minimize circuit area. SEALS [8] and VECBEE-SASIMI [9] combine

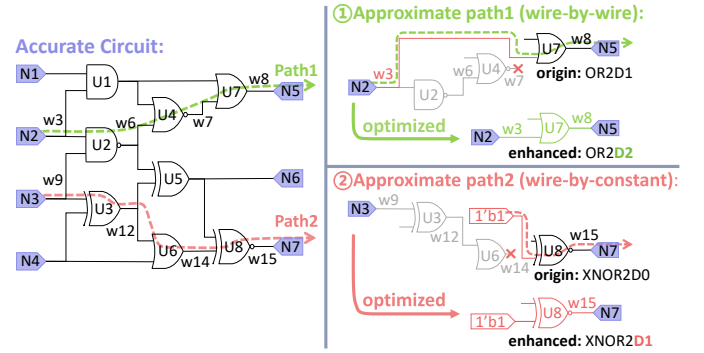


Fig. 1 Optimizing circuit by wire-by-wire (substitute a wire with another wire in circuits) and wire-by-constant (substitute a wire with constant logic value ‘0’/‘1’) LACs. Area reductions are converted into drive strength enhancement of gates.

fast error estimation with greedy algorithms to iteratively select such LACs, efficiently reducing circuit area. Fig. 1 also illustrates that these area reductions can be converted into the enhancement of gate drive strength by post-optimization, leading to further timing improvement.

However, achieving ALS with the greatest potential for timing optimization is challenging for previous methods. Specifically, depth-driven methods inadequately reduce area, leading to difficulties in maximizing the drive strength of gates on critical paths. Area-driven methods simplify many gates on non-critical paths to reduce area, which makes it difficult to obtain the optimal critical path depth. Therefore, it is necessary for timing-driven ALS to simultaneously optimize both critical path depth and area. In this scenario, conventional gradient-based optimizers, including greedy algorithm, genetic algorithm, and traditional grey wolf optimizer (GWO) [11] using a single-chase strategy, cannot finely partition the sampled approximate solutions. Thus, solutions are dispersed in the solution space. This dispersion causes an excessive number of gradients for further optimization. It makes solutions easily move along the gradient with the current fastest critical path depth shortening or area reduction. Finally, traditional optimizers fall into local optima [12], [13].

In this work, we propose a timing-driven approximate logic synthesis framework. As shown in Fig. 2, the framework is composed of three steps, including circuit representation, the double-chase grey wolf optimizer (DCGWO), and post-optimization. Firstly, adjacency lists are constructed based

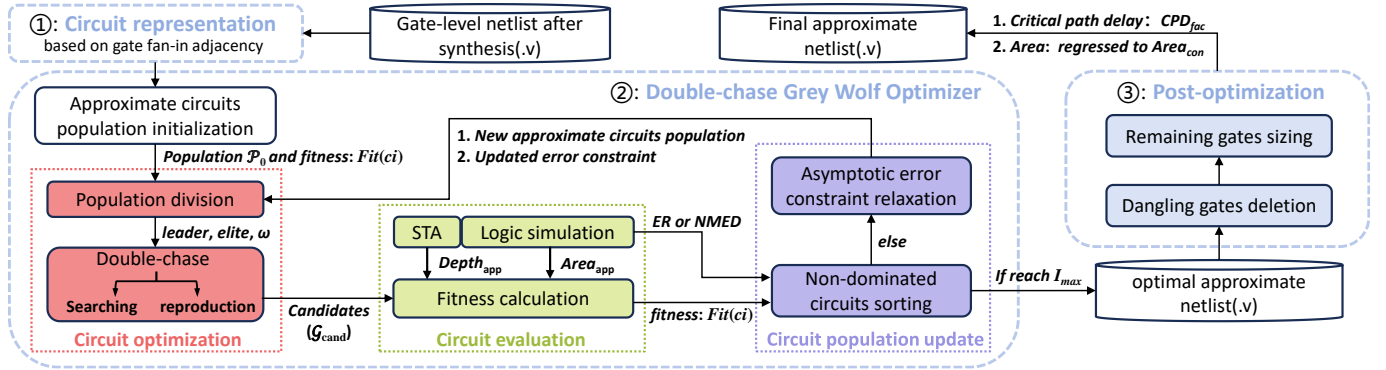


Fig. 2 The overall flow of our timing-driven approximate logic synthesis framework based on double-chase grey wolf optimizer.

solely on gate fan-in relationships to enable fast circuit structure storage and LACs application. Then, DCGWO efficiently optimizes both critical path depth and area under error constraints. Subsequently, post-optimization under area constraints converts the area reduction into further timing optimization. Our contributions are summarized as follows:

- We propose a framework for deeply exploiting timing improvement inherent in the reduction of critical path depth and the enhancement of gate drive strength.
- We represent accurate and approximate circuits based on gate fan-in adjacency lists to improve storage efficiency and accelerate timing optimization.
- We present a DCGWO to effectively select approximate actions for reducing critical path depth and area. Building upon traditional GWO, it divides the generated approximate circuit population into finer hierarchies and precisely formulates appropriate optimization gradients for each hierarchy, improving the efficiency in finding the global optimal approximate circuit.
- The experimental results demonstrate that our framework achieves an average 27.13% and 38.54% critical path delay reduction respectively, under a 5% error rate constraint and under a 2.44% normalized mean error distance constraint, outperforming the state-of-the-art method.

II. PRELIMINARIES

A. Error Metrics

The error metrics used in our framework are error rate (ER) and normalized mean error distance ($NMED$). ER can be used to measure the error of random/control circuits, while $NMED$ can evaluate the error of arithmetic circuits.

For a circuit with m primary inputs and n primary outputs, we assume the probability of input vector I_i occurring is p_i , where $1 \leq i \leq 2^m$. In this case, ER is the probability that the approximate circuit output differs from the accurate circuit output, calculated by Equation (1), where O_i^{app} and O_i^{ori} are output vectors of the approximate circuit and accurate circuit for input vector I_i .

$$ER = \sum_{i=1}^{2^m} (O_i^{app} \neq O_i^{ori}) \times p_i. \quad (1)$$

Error distance is the difference between approximate circuit

output value V_i^{app} and accurate circuit output value V_i^{ori} under input vector I_i . $NMED$ is the mean error distance normalized by the maximum output value, defined in Equation (2).

$$NMED = \sum_{i=1}^{2^m} \frac{|V_i^{ori} - V_i^{app}|}{2^n - 1} \times p_i. \quad (2)$$

B. Problem Formulation

As introduced in Section I, optimizing both critical path depth and area can effectively exploit the potential timing improvement inherent in critical path depth reduction and the enhancement of gate drive strength. Thus, the timing-driven ALS problem can be formulated as follows:

Problem 1 (Timing-driven ALS). *Given a post-synthesis netlist of the accurate circuit with timing, area, and logic information, use an approximate optimizer simultaneously optimizing both critical path depth and area under error constraints to generate the final approximate circuit with maximum critical path delay reduction.*

III. PROPOSED FRAMEWORK

The overall flow of our proposed framework is given in Fig. 2. In step ①, the accurate gate-level netlist is represented by gate fan-in adjacency lists. In step ②, double-chase grey wolf optimizer (DCGWO) simultaneously optimizes critical path depth and area under error constraints. It can efficiently search for optimal approximate circuit through iterative circuit optimization, evaluation and circuit population update. In step ③, by performing dangling gates deletion and remaining gates sizing under area constraint $Area_{con}$ on the generated optimal approximate circuit, the final approximate netlist with maximum critical path delay reduction can be obtained.

A. Circuit Representation

We construct adjacency lists storing the circuit structure based solely on fan-in relationships between gates. By discarding all wire information, the LACs used in our framework, including wire-by-wire [14] and wire-by-constant [15] replacements (shown in Fig. 1), can be easily implemented by changing the gate fan-in adjacency. This operation mode enables us to efficiently assess the impact of LACs and generate corresponding approximate netlist. To check for circuit loop violations, we further label each gate with a unique integer ID. Fig. 3

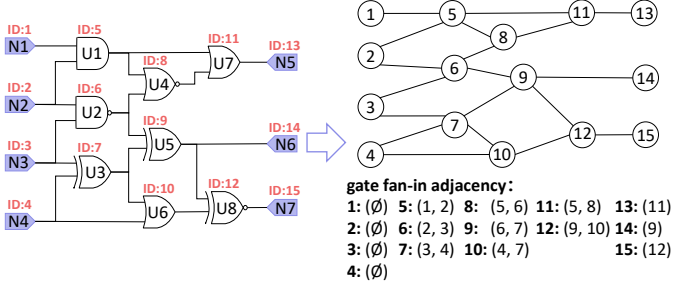


Fig. 3 Circuit representation based on gate fan-in adjacency.

shows an example of circuit representation, the circuit on the left is stored as fan-in adjacency lists on the right.

To accommodate this circuit representation method, we update the related definitions of above two LACs: the gate to be changed is called target gate, while the gate used for change (constant ‘0/1’ are also treated as gates) is called switch gate.

B. Double-chase Grey Wolf Optimizer

In DCGWO, we first generate the initial approximate circuits population $\mathcal{P}_0: \{\forall c_i \in \mathcal{P}_0\}$ by performing LACs on randomly selected target gates of the accurate circuit. Each approximate circuit in \mathcal{P}_0 is evaluated for fitness (defined in section III-B as function Fit), which is composed of critical path depth and area. Circuits with higher fitness values indicate better quality.

Circuit Optimization. As shown in Fig. 2, in each iteration, we perform the double-chase strategy to optimize approximate circuits in the population. The preliminary work for double-chase involves the population division shown in Fig. 4. It divides the population into leader circuit c_l , elite circuits \mathcal{G}_e , and ω circuits group \mathcal{G}_ω based on their fitness values. Specifically, the leader circuit c_l is the approximate circuit with the highest fitness. It guides the elite circuits with fitness ranks 2, 3, and 4 in Chase 1. The elite circuits guide ω circuits group in Chase 2. For Chase 1 and 2, we design two approximate actions: **circuit searching** and **circuit reproduction**. They are used alternately to generate new approximate circuits along suitable optimization gradients.

The circuit searching essentially uses wire-by-wire and wire-by-constant to shorten critical paths. Specifically, we first use PrimeTime [16] to obtain the critical paths with maximum propagation time from primary input (PI) to primary output (PO). Then, for each critical path, all gates on it are stored in the targets set \mathcal{T}_c and undergo uniform (0,1) distribution sampling. All fan-ins of sampled gates with a probability greater than 0.5 are also stored in \mathcal{T}_c . The target gate is randomly selected from \mathcal{T}_c . To limit introduced error, switch gate is selected based on similarities, i.e., the percentage of cycles when output of target gate holds the same value with output of each gate in its transitive fan-in (TFI) or the constant logic value ‘0’, ‘1’. The gate or constant logic value with the highest similarity is selected to substitute the target gate.

Fig. 5 shows circuit searching examples. For obtaining c_{s1}

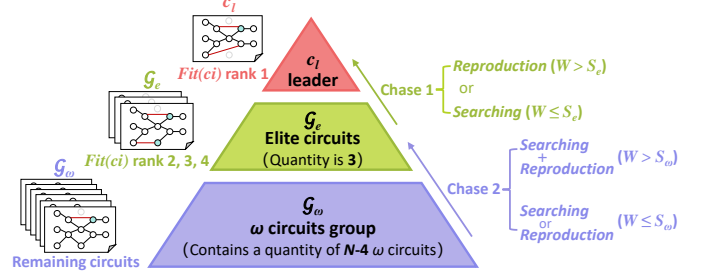


Fig. 4 Population division. Population is divided into leader c_l , elite circuits \mathcal{G}_e , and ω circuits group \mathcal{G}_ω based on fitness, with each hierarchy engaging in distinct chase operations.

from c_{p1} through wire-by-constant searching, Path1 is the critical path. Thus we select ID8 gate (outputs: 14 cycles of ‘0’ and 2 cycles of ‘1’) as the target gate, and constant logic value ‘0’ with the highest similarity 0.875 as the switch gate. In this case, the fan-in adjacency of the ID11 gate is changed from (5, 8) to (5, con0), greatly decreasing the Path1 depth. Similarly, for obtaining c_{s2} from c_{p2} , the fan-in adjacency of ID15 PO is changed from 12 to 10, decreasing the Path3 depth.

Inspired by a crossover in genetic algorithm [17], circuit reproduction is designed to aggregate well-optimized path sets with low errors from two selected approximate circuits, generating a reproduced circuit with better quality. Specifically, we first divide each selected circuit according to the POs and corresponding TFI. Then, for each PO, we use its maximum arrival time T_a and the error generated on it $Error$ to form the PO-TFI pair evaluation function $Level$ in Equation (3), where w_t and w_e are the weights of T_a and $Error$ respectively.

$$Level(PO_i) = w_t \times \frac{1}{T_a(PO_i)} + w_e \times \frac{1}{Error(PO_i)}. \quad (3)$$

Subsequently, we choose PO-TFI pairs with higher $Level$ from two selected circuits to form the reproduced circuit. Some gates are shared by different PO-TFI pairs. Thus, gates in the reproduced circuit only accept adjacency information from the first write-in. Taking circuits c_{p1} and c_{p2} in Fig. 5 as an example, by comparing their $Level$, we select PO2-TFI, PO3-TFI pairs from c_{p1} , and PO1-TFI pair from c_{p2} , to form circuit c_{r1} . Since gates with IDs 8, 10 and 12 are not in any PO-TFI pair, to ensure the completeness of c_{r1} , their information is selected from c_{p1} and c_{p2} to write in c_{r1} .

Fig. 4 illustrates that approximate circuits at different hierarchies consult their adjacent higher-hierarchy circuits for circuit searching and reproduction. Therefore, we design the fitness distance D , decision parameter W and decision threshold S for both elite circuits \mathcal{G}_e and ω circuits group \mathcal{G}_ω . D is calculated by Equation (4), where r_c is defined as a random value between $[0, 2]$. Since \mathcal{G}_e reference the leader circuit c_l for Chase 1, D for elite circuits in \mathcal{G}_e include the fitness of leader circuit $Fit(c_l)$. Similarly, \mathcal{G}_ω reference \mathcal{G}_e for Chase 2. Thus, D for ω circuits in \mathcal{G}_ω include the average fitness of elite circuits in \mathcal{G}_e . W provides a dynamic correction to D by

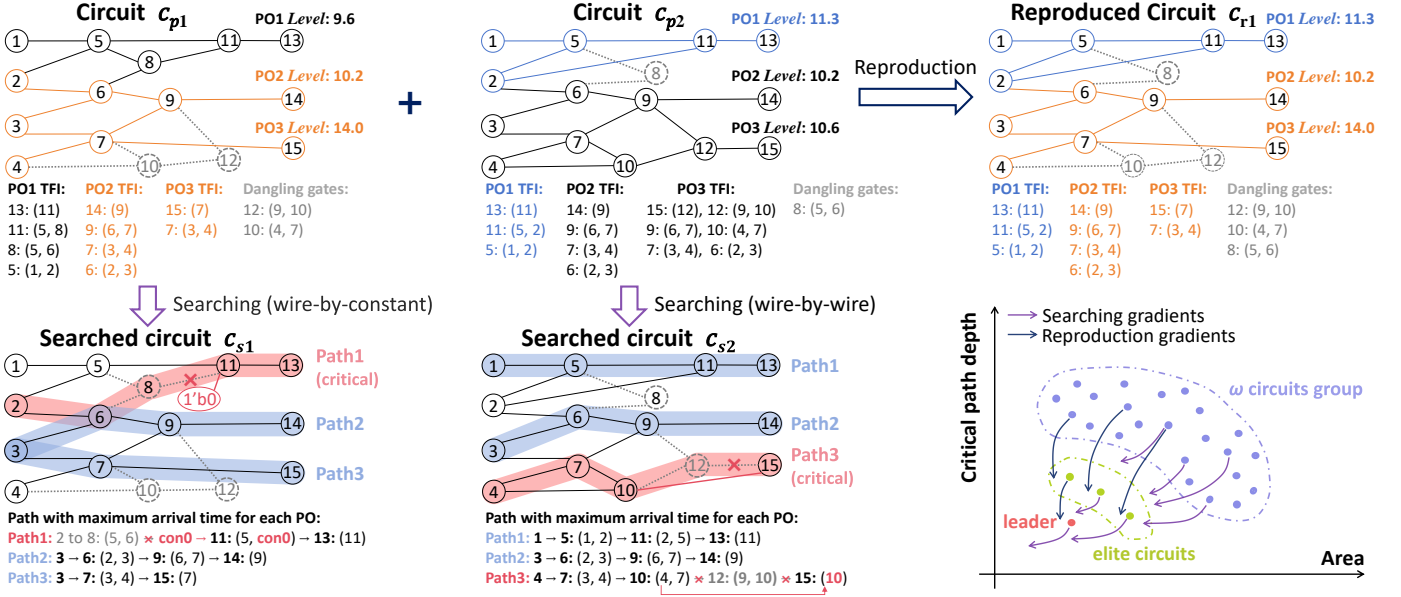


Fig. 5 Illustrations of the circuit searching, circuit reproduction, and optimization gradients guided by them in double-chase.

adding the encircling coefficient A .

$$D(c_i) = \begin{cases} r_c \times Fit(c_i) - Fit(c_i) & \forall c_i \in \mathcal{G}_e \\ \frac{r_c}{3} \sum_{c_j \in \mathcal{G}_e} Fit(c_j) - Fit(c_i) & \forall c_i \in \mathcal{G}_w \end{cases}, \quad (4)$$

$$W(c_i) = A \times D(c_i), \quad (5)$$

where A is calculated based on the scaling factor a as:

$$A = (2 \times r_1 - 1) \times a, \quad (6)$$

where r_1 is a random value between $[0, 1]$. The scaling factor a balances the global search and local convergence of the population during the iterative process. As shown in Equation (7), a decreases with the increase of iteration $iter$ until $iter$ reaches the upper limit of iteration I_{\max} .

$$a = 2 - \frac{2 \times iter}{I_{\max}}. \quad (7)$$

As shown in Fig. 4, the approximate actions are decided by the relationship between decision parameter W and decision thresholds S . Decision thresholds used for \mathcal{G}_e and \mathcal{G}_w are S_e and S_w , respectively. For circuit c_i in \mathcal{G}_e , if $W(c_i) > S_e$, it executes circuit reproduction with another circuit of superior fitness to generate a reproduced circuit. Otherwise, it uses circuit searching to reduce its critical path depth and area. Meanwhile, for circuit c_i in \mathcal{G}_w , if $W(c_i) > S_w$, it performs both circuit searching and reproduction. Otherwise, it randomly selects either circuit searching or reproduction. When the double-chase is completed, the leader c_l conducts circuit searching to ensure its variability. Then, approximate circuits before and after double-chase are stored in the candidates' group \mathcal{G}_{cand} for further evaluation and update.

The lower right corner of Fig. 5 demonstrates that the double-chase strategy can effectively guide the entire population to move along the appropriate gradient with simultaneous critical path depth and area reductions.

Circuit Fitness Evaluation. The fitness function is composed of two optimization objectives: critical path depth and area. The depth-related information, including the maximum critical path depth of each approximate circuit $Depth_{app}$ and the depth of the longest path in corresponding accurate circuit $Depth_{ori}$, are obtained through static timing analysis using PrimeTime [16]. Since circuit searching and reproduction change the connection relationship between gates, some gates become dangling due to their inability to connect to any PO. Therefore, the area of each approximate circuit $Area_{app}$ is the area of accurate circuit $Area_{ori}$ minus the area of these dangling gates.

The fitness function Fit of approximate circuit c_i is defined in Equation (8), where w_d and $w_a = 1 - w_d$ respectively denote the weights assigned to the critical path depth and area. Circuits with higher fitness values indicate better quality.

$$Fit(c_i) = w_d \times \frac{Depth_{ori}(c_i)}{Depth_{app}(c_i)} + w_a \times \frac{Area_{ori}(c_i)}{Area_{app}(c_i)}. \quad (8)$$

Circuit Population Update. To select high-quality approximate circuits under error constraints, we perform non-dominated sorting [18] on the evaluated candidates' group \mathcal{G}_{cand} . It is achieved based on Pareto dominance between circuits determined by two functions: depth function $f_d = \frac{Depth_{ori}}{Depth_{app}}$ and area function $f_a = \frac{Area_{ori}}{Area_{app}}$. Firstly, we remove circuits exceeding the specified error constraint from \mathcal{G}_{cand} . Then, we maintain the dominated list \mathcal{L}_d for each remaining circuit. For approximate circuits c_i and c_j , if c_i is not inferior to c_j in two functions, and is superior in at least one of them, then c_i dominates c_j and is added to \mathcal{L}_d of c_j . Circuits with empty \mathcal{L}_d are considered Pareto-optimal circuits. We place them into the 0-ranked Pareto set while removing them from \mathcal{G}_{cand} and the \mathcal{L}_d of other circuits. Subsequently, new Pareto-optimal circuits with empty \mathcal{L}_d emerge, forming the 1-ranked Pareto set, and undergo the same removal process. This will repeat until \mathcal{G}_{cand} is empty.

We further calculate the crowding distance $Dist$ of approximate circuits in each Pareto set. With higher $Dist$, circuits are less likely to overlap in the objective function space, resulting in better optimization efficiency. In the k -ranked Pareto set, approximate circuits are sorted separately based on f_d and f_a . The $Dist$ of the circuits at the beginning and end of these two sorted lists are set to $+\infty$. For approximate circuit c_i , circuits adjacent to c_i in each sorted list are c_{i-1} and c_{i+1} . In this case, $Dist$ is calculated by Equation (9).

$$Dist(c_i) = \sum_{x=d,a} \frac{f_x(c_{i-1}) - f_x(c_{i+1})}{\max_k(f_x) - \min_k(f_x)}. \quad (9)$$

Based on Pareto set partition and crowding distance calculation, we sort the approximate circuits within each Pareto set in descending order of their $Dist$. Subsequently, starting from the 0-ranked Pareto set, we sequentially select N approximate circuits to form a new population for the next iteration.

After the non-dominated circuits sorting, the asymptotic error constraint relaxation is employed. We design a quadratic function scheme (i.e., $Error_{cons.}^{iter} = b \times iter^2 + Error_{cons.}^0$) to gradually increase the error constraint $Error_{cons.}^{iter}$ as the iteration $iter$ rises, ultimately relaxing it to the user-specified maximum error constraint by setting appropriate empirical parameter b . This operation prevents the population from quickly moving to the maximum error constraint boundary and getting trapped in local optima.

C. Post-Optimization

Post-optimization is performed on the optimal approximate circuit generated by DCGWO. It can further convert the area reductions into timing performance improvements by enhancing the drive strength of gates.

We first delete dangling gates produced by circuit searching and reproduction from the optimal approximate circuit. In this process, we traverse the entire circuit, identifying and removing gates with empty transitive fan-out (TFO). For each fan-in of the removed gates, we similarly perform identification and removal operations until no gates with empty TFO remain. Subsequently, for the processed optimal approximate circuit, we use Design Compiler [19] to resize its remaining gates without adjusting any circuit structure under area constraints $Area_{con}$. Consequently, the final approximate circuits with minimum critical path delay CPD_{fac} are obtained.

IV. EXPERIMENTAL RESULTS

Our proposed framework is implemented in Python. We set up the experimental environment on the Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs in parallel with 128GB memory. The benchmarks listed in TABLE I are from ISCAS'85 [20] and EPFL [21]. Each circuit is synthesized into gate-level netlist by Design Compiler [19] under TSMC 28nm technology. Among these benchmarks, random/control circuits are optimized under ER constraints, while arithmetic circuits are optimized under $NMED$ constraints. For the generated approximate circuits, their timing-related information is obtained through static timing analysis performed by PrimeTime [16]. The circuit error and the similarities between outputs

TABLE I The benchmark statistics. CPD_{ori} (ps) and $Area_{ori}$ (μm^2) respectively represents the **critical path delay** and area of accurate circuit.

Type	Circuit	#gate	#PI/PO	CPD_{ori}	$Area_{ori}$	Description
Random/ Control	Cavlc	573	10/11	186.35	450.31	Coding Cavlc
	c880	322	60/26	185.34	177.67	8-bit ALU
	c1908	366	33/25	235.14	223.34	16-bit SEC/DED circuit
	c2670	922	233/140	218.40	288.71	12-bit ALU and controller
	c3540	667	50/22	293.09	459.42	8-bit ALU
	c5315	2595	178/123	122.25	1129.55	9-bit ALU
Arithmetic	c7552	1576	207/108	282.13	939.33	32-bit adder/comparator
	Int2float	198	11/7	127.02	194.63	int to float converter
	Adder16	269	32/17	58.92	288.41	16-bit adder
	Max16	154	32/16	131.78	91.43	16-bit 2-1 max unit
	c6288	1641	32/32	847.79	687.08	16 × 16 multiplier
	Adder	1639	256/129	1394.7	495.78	128-bit adder
	Max	2940	512/120	2799.8	954.03	128-bit 4-1 max unit
	Sin	10962	24/25	701.03	4367.27	24-bit sine unit
Sqrt	13542	128/64	67929.3	6262.10	128-bit square root unit	

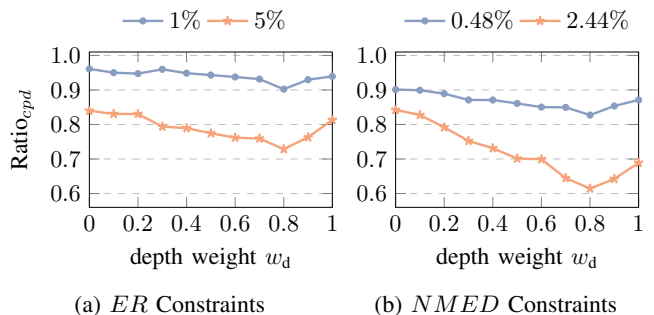


Fig. 6 Average critical path delay ratios $Ratio_{cpd}$ generated by our framework using different depth weight w_d under the tightest and loosest ER and $NMED$ constraints.

of gates are obtained using VECBEE based on Monte Carlo simulation [9]. By setting the number of sampled input vectors to 1×10^5 , this method can achieve fast error and similarities evaluation with nearly no deviation.

A. Parameter Setting

The parameters of our framework are set as follows. The population size N is 30 and the upper limit of iterations I_{max} is 20. For PO-TFI pair evaluation function $Level$, w_t is $0.9 \times CPD_{ori}$ under both error constraints, while w_e is respectively 0.1 and 0.2 under ER and $NMED$ constraints. For circuit fitness Fit , we determine the optimal weights based on **critical path delay ratios** of the final approximate circuits over the accurate circuits (i.e., $Ratio_{cpd} = \frac{CPD_{fac}}{CPD_{ori}}$). Fig. 6 illustrates that the minimum $Ratio_{cpd}$ are achieved under both the tightest and loosest error constraints when w_d is 0.8 and $w_a = 1 - w_d$ is 0.2. Therefore, we follow this setting.

B. Optimization Performance

Since our framework focuses on timing optimization, we compare the performance of our framework, including final **critical path delay ratios** $Ratio_{cpd} = \frac{CPD_{fac}}{CPD_{ori}}$ and **runtime**, with: (1) area-driven methods: VECBEE-SASIMI [9]; (2) depth-driven methods: VaACS [5], HEDALS [6]; (3) traditional GWO (single-chase). Approximate circuits generated by these works experience post-optimization (in Section III-C) under **area constraints** $Area_{con}$ to convert area reduction into further critical path delay reduction by Design Compiler [19].

TABLE II Comparison of performance between our framework and others under 5% ER constraints. All final generated circuits experience post-optimization under area constraints $Area_{con}$ to convert area reduction into further critical path delay reduction.

Circuit	$Area_{con}$ (μm^2)	VECBEE-S [9]		VaACS [5]		HEDALS [6]		GWO (single-chase)		Ours	
		Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)
Cavlc	450.00	0.9219	60.03	0.8745	356.89	0.9071	194.43	0.8963	407.25	0.8602	310.42
c880	177.00	0.9026	43.11	0.9221	227.13	0.8913	104.00	0.9183	201.51	0.8399	193.86
c1908	223.00	0.8679	65.32	0.5166	235.68	0.3372	310.42	0.5021	307.56	0.3865	202.79
c2670	288.00	0.6708	308.16	0.8101	477.92	0.7589	250.28	0.7703	313.99	0.6314	339.63
c3540	459.00	0.9670	391.42	0.9729	435.26	0.9203	373.26	0.9224	479.88	0.8732	324.59
c5315	1129.00	0.9113	1857.32	0.8599	1963.55	0.8270	1662.08	0.8165	1655.07	0.8034	1449.37
c7552	939.00	0.9262	1726.27	0.9133	1336.64	0.7391	1315.85	0.8877	1420.32	0.7063	1279.18
Average	523.57	0.8811	635.94	0.8385	719.01	0.7687	601.47	0.8162	683.65	0.7287	585.69

TABLE III Comparison of performance between our framework and others under 2.44% $NMED$ constraints. All final generated circuits experience post-optimization under $Area_{con}$ to convert area reduction into further critical path delay reduction.

Circuit	$Area_{con}$ (μm^2)	VECBEE-S [9]		VaACS [5]		HEDALS [6]		GWO (single-chase)		Ours	
		Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)	Ratio _{cpd}	runtime(s)
Int2float	194.00	0.9331	71.23	0.5047	151.73	0.7649	32.68	0.6010	178.30	0.4496	132.12
Adder16	288.00	0.9973	67.20	0.5295	173.85	0.4513	47.30	0.5216	189.01	0.4275	167.03
Max16	91.00	0.7087	93.17	0.4209	189.73	0.4470	105.97	0.3928	277.38	0.3708	208.55
c6288	687.00	0.9663	4410.29	0.8696	3279.62	0.6368	2563.41	0.9079	2991.00	0.8313	2103.88
Adder	495.00	0.7814	1697.37	0.8133	2083.15	0.7110	1362.70	0.8008	1550.03	0.6917	1193.71
Max	954.00	0.8809	2600.78	0.8933	3397.50	0.8355	2992.08	0.7517	3121.44	0.6799	2035.62
Sin	4367.00	0.9187	5391.68	0.8326	3872.31	0.7945	3380.52	0.8722	4392.77	0.7603	3176.46
Sqrt	6262.00	0.7993	33117.12	0.8011	20160.76	0.7437	11242.29	0.7803	17894.50	0.7058	9950.11
Average	1667.25	0.8732	5931.11	0.7081	4163.58	0.6731	2715.87	0.7035	3824.30	0.6146	2370.94

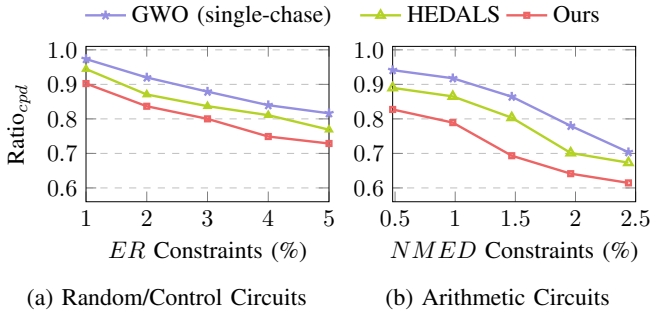


Fig. 7 Average critical path delay ratios $Ratio_{cpd}$ generated by our framework, HEDALS [6] and traditional GWO under different ER and $NMED$ constraints.

For random/control circuits, the performance comparison of all works under the loosest 5% ER constraint is detailed in TABLE II. According to the comparison results, by setting the same area constraints, our framework maximizes the average critical path delay reduction to 27.13% with shorter runtime under the 5% ER constraint. Similarly, for arithmetic circuits, the performance comparison of all works under the loosest 2.44% $NMED$ constraint is detailed in TABLE III. The comparison results indicate that our framework maximizes the average critical path delay reduction to 38.54% with shorter runtime under the 2.44% $NMED$ constraint.

We further compare the average $Ratio_{cpd}$ achieved by our work with HEDALS [6] and traditional GWO under 5 different ER constraints (1%, 2%, 3%, 4%, 5%) and 5 different $NMED$ constraints (0.48%, 0.98%, 1.47%, 1.96%, 2.44%). According to results in Fig. 7, as the ER or $NMED$ constraint tightens, our framework consistently achieves greater critical path delay reductions than others. Fig. 8 illustrates

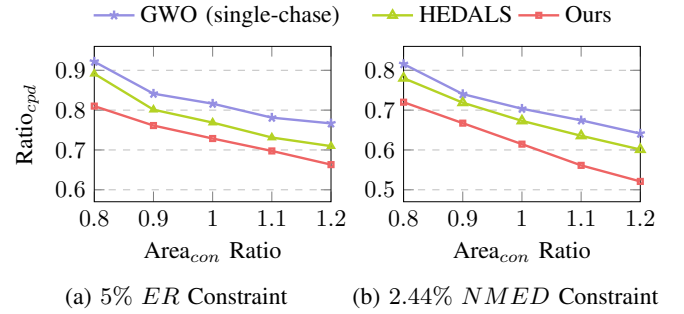


Fig. 8 Average critical path delay ratios $Ratio_{cpd}$ generated by our framework, HEDALS [6] and traditional GWO under different area constraints ($Ratio \times Area_{con}$).

how the average $Ratio_{cpd}$ varies with different area constraints ($0.8 \times \sim 1.2 \times Area_{con}$) under the loosest ER and $NMED$ constraints. The results indicate that our framework outperforms other works in timing optimization across all area constraints. These achievements demonstrate that our framework can generate approximate circuits with superior performance while meeting diverse accuracy and area requirements.

In summary, by leveraging carefully designed approximate actions and the powerful search capabilities of DCGWO, our framework can better exploit the timing improvement inherent in critical path shortening and the enhancement of gate drive strength. Additionally, compared to traditional GWO, using the double-chase strategy to further formulate the optimization gradients indeed helps the optimizer find better solutions. Benefiting from the fast implementation of LACs and the inherent parallelism of GWO, our framework maintains low time consumption despite using PrimeTime [16] for accurate timing analysis.

V. CONCLUSION

In this work, we propose a timing-driven approximate logic synthesis framework based on DCGWO to effectively optimize circuit timing under ER or $NMED$ constraints. Its main idea involves using DCGWO to optimize both critical path depth and area to achieve precise and efficient optimal approximate circuit generation and utilizing post-optimization under area constraints to convert area reduction into further timing improvement. According to the experimental results on open-source designs, under the same error and area constraints, our framework can achieve more critical path delay reduction than existing methods within an acceptable time consumption.

REFERENCES

- [1] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.
- [2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.
- [3] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Timing-driven technology mapping approximation based on reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.
- [4] K. Balaskas, G. Zervakis, H. Amrouch, J. Henkel, and K. Siozios, "Automated design approximation to overcome circuit aging," *IEEE Transactions on Circuits and Systems I*, vol. 68, no. 11, pp. 4710–4721, 2021.
- [5] K. Balaskas, F. Klemme, G. Zervakis, K. Siozios, H. Amrouch, and J. Henkel, "Variability-aware approximate circuit synthesis via genetic optimization," *IEEE Transactions on Circuits and Systems I*, vol. 69, no. 10, pp. 4141–4153, 2022.
- [6] C. Meng, Z. Zhou, Y. Yao, S. Huang, Y. Chen, and W. Qian, "Hedals: Highly efficient delay-driven approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 3491–3504, 2023.
- [7] C. Meng, W. Qian, and A. Mishchenko, "Alsrac: Approximate logic synthesis by resubstitution with approximate care set," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [8] C. Meng, X. Wang, J. Sun, S. Tao, W. Wu, Z. Wu, L. Ni, X. Shen, J. Zhao, and W. Qian, "SEALS: Sensitivity-driven efficient approximate logic synthesis," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 439–444.
- [9] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 5085–5099, 2022.
- [10] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Approximate logic synthesis by genetic algorithm with an error rate guarantee," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2023, pp. 146–151.
- [11] S. Mirjalili, S. Saremi, S. M. Mirjalili, and L. d. S. Coelho, "Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization," *Expert sys. appl.*, vol. 47, pp. 106–119, 2016.
- [12] S. Zheng, L. Zou, S. Liu, Y. Lin, B. Yu, and M. Wong, "Mitigating distribution shift for congestion optimization in global placement," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.
- [13] Y. Zhao, L. Wang, K. Yang, T. Zhang, T. Guo, and Y. Tian, "Multi-objective optimization by learning space partitions," *arXiv preprint arXiv:2110.03173*, 2021.
- [14] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE)*. IEEE, 2013, pp. 1367–1372.
- [15] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and applications of approximate circuits by gate-level pruning," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 25, no. 5, pp. 1694–1702, 2017.
- [16] Synopsys, "Primetime user guide," <https://www.synopsys.com/cgi-bin/imp/pdfdla/pdf1.cgi?file=primetime-wp.pdf>, 2023.
- [17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.
- [18] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to nondominated sorting for evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 201–213, 2014.
- [19] Synopsys, "Design compiler user guide," <https://www.synopsys.com/zh-cn/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html>, 2023.
- [20] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
- [21] EPFL, "The epfl combinational benchmark suite," <https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/>, 2019.