SPFD-Based Delay Resynthesis

Andrea Costamagna, Chang Meng, Giovanni De Micheli
Integrated Systems Laboratory, EPFL, Lausanne, Switzerland

***Abstract*—We propose novel algorithms for post-mapping delay optimization, referred to as *delay resynthesis*. The state-of-the-art approach optimizes delay by rewriting with optimal local substructures. However, it is limited to local transformations, does not exploit *don't cares*, and relies on a basic decomposition heuristic to handle subnetworks with more than four inputs. Our method overcomes these limitations by employing modern resynthesis techniques for non-local optimization and integrating *don't care* conditions. In addition, we introduce a more powerful decomposition strategy that extends beyond prior methods. Central to our approach is the *set of pairs of functions to be distinguished* (SPFD), a Boolean function representation that captures functional dependencies between nodes for finer-grained logic restructuring. Experimental results on EPFL benchmarks show a 5.70% delay improvement over the state of the art.**

***Index Terms*—Resynthesis, Delay, SPFD**

I. Introduction

WHILE the demand for higher computing performance continues to grow, CMOS technology has entered the deep nanometer regime, where financial and physical constraints limit further speed improvements through transistor scaling alone. This challenge necessitates greater emphasis on design-level optimization of digital systems.

This challenge can be addressed at the logic synthesis level by enhancing delay-oriented optimization. Several techniques have been proposed to perform rewriting aimed at reducing the critical path of combinational circuits [1]. The *theory of equioptimizable arrival time patterns* (TEAP) underpins the state of the art in delay optimization [2]. TEAP enables one to construct databases storing several circuits for each function, ensuring optimal delay for specific arrival time profiles.

However, TEAP-based approaches are constrained by memory limitations, restricting databases to 4-input functions and relying on simple decomposition heuristics to extend subcircuit sizes. Additionally, these methods use only local structural information and do not exploit *don't cares*.

This paper addresses these limitations by integrating TEAP within a modern resynthesis engine that enables non-local and *don't care*-aware database-based rewriting. Furthermore, inspired by recent works on delay-driven LUT-mapping [3], we enhance the decomposition heuristic for scaling up subcircuit replacements with a novel *don't care*-aware synthesis technique for networks of four input look-up tables (4-LUT). Applied to the EPFL benchmarks after delay-oriented optimization and mapping, our approach reduces delay by 5.70% compared to the state-of-the-art, demonstrating effectiveness.

II. Background

A. Logic Synthesis Basics

A *Boolean network* is a directed acyclic graph (DAG) in which nodes represent logic gates and edges represent wires. A *mapped network* is a Boolean network where each node corresponds to a cell from a technology library. The arrival time of a node x, t_x^A, is the earliest time at which its output stabilizes. The required time, t_x^R, is the latest time at which a signal x can change without increasing the circuit delay.

Each node x in a combinational circuit implements a Boolean function, referred to as its *global node functionality*. A *dependency cut* is a subset $\mathcal{C} = (x, \mathcal{L})$, where $\mathcal{L}$ consists of nodes that do not lie on any path from x to a circuit output. The Boolean function f, called the cut functionality, expresses the global functionality of x in terms of $\mathcal{L}$, i.e., $x = f(\mathcal{L})$.

A dependency cut $\mathcal{C} = (x, \mathcal{L})$ is a *structural cut* if the circuit contains a sub-network that implements the cut functionality. A *reconvergence-driven cut* is a structural cut constructed using heuristics that maximize the number of reconvergent paths from the leaves $\mathcal{L}$ to the root x [4].

B. Boolean Basics

We represent an incompletely specified Boolean function $f : \mathbb{B}^n \to \{0, 1, *\}$ as two completely specified Boolean functions:

1) The *onset* $\tau : \mathbb{B}^n \to \mathbb{B}$ $\tau_M = 1/0 \Leftrightarrow f_M = 1/0$
2) The *careset* $\mu : \mathbb{B}^n \to \mathbb{B}$ $\mu_M = 0 \Leftrightarrow f_M = *$.

Where f_M is the value of the function f at minterm $M \in \mathbb{B}^n$.

The *information* contained in a Boolean function is its capability to distinguish two minterms (M, K) when $\tau_M \mu_M \neq \tau_K \mu_K$. The set $\Upsilon_f = \{(\tau, \mu)\}$ compactly encodes this information, often called *set of pairs of functions to be distinguished* (SPFD) or *information graph* (IG) [5].

Let $x_i : \mathbb{B}^n \to \mathbb{B}$ be a Boolean function with SPFD Υ_{x_i}. The *SPFD difference* $\Upsilon_f - \Upsilon_{x_i} = \{(\tau, \mu x_i), (\tau, \mu x_i')\}$ generates another SPFD, where the information shared by x_i and f is removed. Each subset (τ, μ_i) identifies a disjoints set of minterms not yet distinguished after using the information in x_i. This operation can be iterated: $\Upsilon_f - \Upsilon_{x_i} - \Upsilon_{x_j} = \{(\tau, \mu x_i x_j), (\tau, \mu x_i' x_j), (\tau, \mu x_i x_j'), (\tau, \mu x_i' x_j')\}$. $\{..(\tau_i, \mu_i)..\}$ contains the same information as $\{..(\tau_i', \mu_i)..\}$.

The set $\mathcal{C} = (x, \mathcal{L})$ is a *dependency cut* if and only if the SPFDs of the global functions of x and $\mathcal{L} = \{x_i\}_{i=1}^{L}$ satisfy

$$\Upsilon_x - \sum_{x_i \in \mathcal{L}} \Upsilon_{x_i} = \varnothing \tag{1}$$

In this paper we propose a timing-aware algorithm for selecting non-structural *dependency cuts* using Eq. (1).

C. Exact Delay Resynthesis

Let us define the main problem in delay optimization:

This work was supported in part by Synopsys.

EXACT DELAY SYNTHESIS

Given: 1) A Boolean function $f : \mathbb{B}^n \mapsto \mathbb{B}^m$;
2) A set of input arrival times $\mathbf{t} = (\mathbf{t}_i)_{i=1}^{n}$;
3) A technology library $\ell = (\ gate_i\ )_{i=1}^{n_L}$.

Find a circuit $\mathcal{N}$ synthesizing f, and minimizing the output(s) arrival time $\min_{t^A(\mathcal{N})}(f, \mathbf{t}, \ell)$.

This problem is intractable, so its optimum solution can only be found for small values of n. Hence, larger circuits are first represented as sub-optimal networks, and optimized through *resynthesis*, which amounts to replacing sub-optimal circuits in a cut with an optimum alternative from a database.

Since arrival time pattern are real valued vectors $\mathbf{t} \in \mathbb{R}^n$, defining a complete delay-based database was considered an unfeasible task until the *theory of equioptimizable arrival patterns* (TEAP) [2] enabled defining finite-size databases.

D. Theory of Equioptimizable Arrival Patterns

Let d_{MUX} be the delay of a multiplexer from the gate library L. Using Shannon decomposition, we can prove that the exact delay circuit of an n-input function must satisfy $\min_{t^A(\mathcal{N})}(f, \mathbf{t}, \ell) \leq \Delta(n, \ell) + \max(\mathbf{t}) = n \cdot d_{MUX} + \max(\mathbf{t})$.

Using this observation, Amarù et al. [2] devised the compression strategy sketched in Fig. 1, which maps an arrival time pattern $\mathbf{t}$ to $\mathbf{t}' = \Gamma(\mathbf{t})$, while ensuring that $\mathbf{t}$ and $\mathbf{t}'$ share the same exact delay circuit. This compression strategy maps

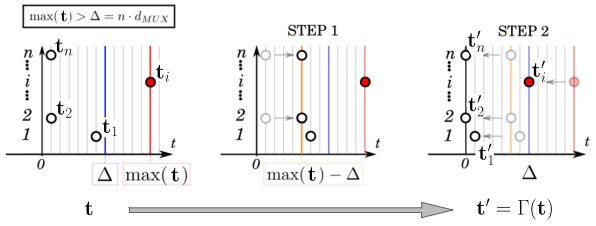

Fig. 1. Arrival time lossless compression $\mathbf{t}' = \Gamma(\mathbf{t}) \in [0, \Delta(n, L)]^n$.

infinitely many input arrival patterns to $(\Delta(n, \ell)+1)^n$ classes, enabling resynthesis through database-based rewriting.

Memory constraints limit databases to 4-input functions. However, less local rewriting using structural cuts with size $L > 4$ can optimize delay further. The authors address this issue using decomposition. First, they attempt 2-operands disjoint support decomposition on the latest arriving variable: $f(\mathcal{S}) = x_i \odot h(\mathcal{S} \backslash x_i)$. In case of success, the problem reduces to synthesizing a $(L-1)$-input function. Otherwise, Shannon decomposition reduces the problem to the synthesis of two $(L-1)$ input functions: $f(\mathcal{S}) = x_i' f_0(\mathcal{S} \backslash x_i) + x_i f_1(\mathcal{S} \backslash x_i)$. When the support size is 4, the database is used. In this paper, we propose a more powerful decomposition using SPFDs.

III. SPFD-Based Delay Resynthesis

A. Delay-Oriented Resynthesis

Algorithm 1 outlines the resynthesis engine, whose goal is to reduce the arrival time of the nodes on the combinational critical paths. For each of these nodes, a window is built from a reconvergence-driven cut, and a set of candidate dependency cuts is obtained by both structural cut enumeration and non-structural cut selection, as discussed in Sec. III-D.

Each candidate cut has size at most k, which is a parameter of the engine. Based on this value, the cut functionalities are decomposed into 4-LUT networks using the SPFD-based decomposition discussed in Sec. III-E. Next, we traverse the 4-LUT networks in topological order. For each 4-LUT we identify the patterns never appearing at its inputs, that are local *don't cares* that we exploit during database look-up. This gives us a pair (τ, μ) for each 4-LUT, where $\tau, \mu : \mathbb{B}^4 \rightarrow \mathbb{B}$.

We enumerate all the acceptable functions by assigning the *don't care* minterms to the *onset* or to the *offset*, and select the database entry yielding the smallest arrival time. After synthesizing the entire 4-LUT network, if the output arrival time is lower than the target node, we substitute it.

This approach introduces several key innovations:

- Section III-B presents a systematic method for selecting arrival time patterns to populate the database.
- Section III-C details our strategy for constructing databases with up to 6-input gates.
- Section III-D introduces our algorithm for selecting non-structural cuts, extending optimization beyond the locality constraints of structural-based state-of-the-art approaches.
- Section III-E describes our SPFD-based heuristic for synthesizing 4-LUT networks.

Algorithm 1: Delay-Oriented Resynthesis

```
Data: circuit G, database D, support size L
Result: A new mapped circuit optimized for delay
for v ∈ G on critical paths do
    Build and simulate a window for node v;
    cands ← {Delay-aware dependency cuts of size L};
    cands ← {Enumeration of structural cuts of size L};
    N_best ← ∅, t^A_best = t^A_x;
    for (x, L, f) ∈ cands do
        N ← ∅;
        4LUTs ← SPFD-decomposition(f);
        for τ ∈ 4-LUTs do
            μ ← Compute Satisfiability don't cares;
            N_i ← delay-match with don't cares from D;
            N ← add N_i;
        if t^A(N) < t^A_best then
            (N_best, t^A_best) ← (N, t^A(N));
    if t^A_best < t^A_x then
        x ← Substitute with N_best;
return the optimized circuit;
```

B. Lossy Quantization of the Arrival Time Patterns

Although finite, the number of arrival pattern classes identified by TEAP, $(\Delta(n, \ell)+1)^n$, is too large to be stored. Lossy quantization is needed to identify tractable delay sets [2].

A lossy quantization procedure identifies a characteristic subset of all possible input arrival time patterns. We employ k-means clustering, which aims to partition a set of observations

into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

We enumerate structural cuts over the EPFL benchmarks after delay-oriented technology mapping. The arrival time patterns were re-normalized using the transformation Γ and sorted by arrival time to induce an ordering. Next, we identify the k-means, quantize them onto intervals of duration d_{MUX}. With this procedure we select the set of input arrival time patterns $\{0,0,0,0\}$, $d_{MUX}\{3,3,0,0\}$, and $d_{MUX}\{3,0,0,0\}$. The enumeration of all possible 11 permutations generates the input arrival patterns used to populate the database.

C. Database Construction

We construct a database using circuit enumeration with gates from the $7nm$ `asap7` technology library. Unlike prior works with simplified libraries and uniform gate delays, we incorporate precise pin-to-pin delays from the `.genlib` file.

For each arrival time pattern, we construct D_2, a database of circuits using 1 and 2-input gates. Arrival times are discretized using a quantization time step, chosen to be smaller than the smallest pin-to-pin delay in L. The database is built level by level by sweeping over pairs of existing sub-circuits in order of arrival, evaluating each gate according to pin-to-pin delays, and adding a new sub-circuit if it implements a unique function and its arrival time aligns with the current quantization step.

This method produces a near-optimal database in minutes. Minor arrival time fluctuations occur due to quantization errors and pin-to-pin delay variations, which prevent a strict ordering.

Since exhaustive enumeration with larger gates is infeasible, we employ an incremental approach, in which we use resynthesis to optimize a database, while accepting some sub-optimality. After generating D_2, we construct D_3 (up to 3-input gates) by considering triplets of the earliest N arriving functions and substituting entries based on arrival time comparisons. This process continues until D_6, which is a database of 4-input functions using up to 6-input gates.

D. Non-Structural Cut Selection Using Delay Information

Our algorithm supports rewriting with nonstructural cuts, enabling nonlocal optimizations by uncovering nontrivial dependencies between a node and distant parts of the circuit that are overlooked during technology mapping or structural-cut-based rewriting [6]. This section describes how these essential dependencies are identified.

Algorithm 2: Delay-Aware Dependency Cut Selection

Data: A node x, a set of candidate nodes $\mathcal{D} = \{x_i\}_{i=0}^{N}$
Result: A dependency cut $\mathcal{C} = (x, \mathcal{L})$ with $|\mathcal{L}| \leq L$

$t^A_{max} \leftarrow t^A_x - \langle d \rangle$, $t^A_{min} \leftarrow \min_{x_i \in \mathcal{D}} t^A_{x_i}$;
for $t = [1, N]$ **do**
  for $x_j \in \mathcal{D}_t = \{x_i : t^A_{x_i} < t^A_{min} + t\frac{t^A_{max} - t^A_{min})}{N}\}$ **do**
    $\Upsilon \leftarrow \Upsilon - \Upsilon_{x_j}$ $\mathcal{L} \leftarrow \{x_j\}$;
    while $\Upsilon \neq \varnothing \wedge |\mathcal{L}| < k$ **do**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \arg\min_{x_l \in \mathcal{D}_t}(|\Upsilon - \Upsilon_{x_l}|)$
    if $\Upsilon = \varnothing$ **then**
      return $\mathcal{C} = (x, \mathcal{L})$;

Let $\Upsilon = \{(\tau_i, \mu_i)\}_{i=1}^{P}$ be an SPFD. The number of remaining minterms to be distinguished can be computed as $|\Upsilon| \doteq \sum_{i=1}^{P} |\tau_i \mu_i|_1 |\tau'_i \mu_i|_1$. This metric enables the design of algorithms for solving Eq. (1), as the one in Algorithm 2.

Given a node x, we identify a set of candidate nodes $\mathcal{D}$ by constructing a subnetwork (*window*) from a reconvergence-driven cut rooted in x [4]. We evaluate the functionality of each node through exhaustive simulation of the sub-network. We identify the earliest arrival time t^A_{min} and estimate the latest arrival time as the difference between the arrival time of the target node and the average cell delay in the database $\langle d \rangle$.

We partition $\mathcal{D}$ into N subsets based on arrival time intervals $[t^A_{min}, t_{min} + i(t^A_{max} - t^A_{min})/N]$, $i \in [1, N]$, prioritizing early-arriving variables to aid delay-reducing rewriting. For each subset, we seek a dependency cut of size k, initializing the search by enforcing one node in the solution—capturing cases missed by greedy methods. We then proceed greedily.

E. SPFD-Based Decomposition

Let $f : \mathbb{B}^L \rightarrow \{0, 1, *\}$ be the cut functionality of a dependency cut $\mathcal{C} = (x, \mathcal{L})$. We decompose f as a network of 4-LUTs, to be replaced with database entries. We devise the delay-aware SPFD-based decomposition reported in Algorithm 3. Fig. 2 illustrates the recursive step of the procedure.

Let $\mathcal{S} = \{x_i\}_{i=1}^{L} \subseteq \mathcal{L}$ be the functional support of f, sorted so that $\mathbf{t}_i \geq \mathbf{t}_{i+1}$. The termination condition of the recursion occurs when $|\mathcal{S}| \leq 4$: a 4-LUT is obtained by interpolating the support functions $\{x_i\}_{i=0}^{L-1}$ with the target f.

If the termination check fails, we delay the latest arriving variables x_0 and x_1, placing them in the support of the top 4-LUT h. The SPFD difference $\Upsilon - \Upsilon_{x_0} - \Upsilon_{x_1} = \{(\tau, \mu_0), (\tau, \mu_1), (\tau, \mu_2), (\tau, \mu_3)\}$ lists minterm pairs that x_0 and x_1 cannot distinguish, necessitating two new functions, p and q. We derive candidates for p and q by enumerating functions that retain unsynthesized information from the SPFD difference. Fig. 2 shows the enumeration of the 28 candidates:

1) $p = (\tau\mu_0, \mu_0)$ $\quad q = (\tau(\mu_1 + \mu_2 + \mu_3), \mu_1 + \mu_2 + \mu_3)$
⋮
28) $p = (\tau'\mu_0 + \tau\mu_2, \mu_0 + \mu_2)$ $\quad q = (\tau'\mu_1 + \tau\mu_3, \mu_1 + \mu_2 + \mu_3)$

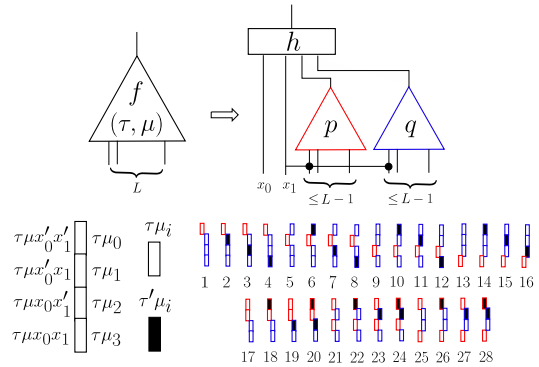

Fig. 2. Decomposition of a k-LUT into a 4-LUT, and two $(L-1)$-LUTs and SPFD-enumeration of candidates p and q.

We iterate through the candidate functions p and q, extract their functional supports, and select the most suitable ones

based on two criteria: (i) the earliest arriving variables and (ii) a measure of total support size to favor compact synthesis.

This decomposition procedure inherently includes both Shannon decomposition and two-operand disjoint support decomposition. Explicitly, the functions $p_{17} = (\tau x_0, \mu x_0)$ and $q_{17} = (\tau x_0', \mu x_0')$ identify the Shannon decomposition.

Algorithm 3: SPFD-decomposition

Data: $f = (\tau, \mu)$, support $\mathcal{S} = \{x_i\}_{i=0}^{L-1}$ sorted by $\mathbf{t}$
Result: A 4-LUT network implementing f

if $L \leq 4$ **then**
  return $h \leftarrow$interpolate from $\{x_i\}_{i=0}^{L-1}$ and f;
$\{(p_i, q_i)\}_{i=1}^{28} \leftarrow \Upsilon_f - \Upsilon_{x_1} - \Upsilon_{x_0}$;
$t^A_{best} \leftarrow t^A_{x_1}$ $i^* \leftarrow 17$ $P^* \leftarrow (L-1)^2$;
for $i \in [1, 28]$ **do**
  $\mathcal{S}_p \leftarrow$functional support p_i sorted by arrival time;
  $\mathcal{S}_q \leftarrow$functional support q_i sorted by arrival time;
  if $\mathcal{S}_{p,0} \neq x_0 \wedge \mathcal{S}_{q,0} \neq x_0$ **then**
    $t^A_{worst} = \max\{\max_{x_i \in \mathcal{S}_p}(t^A_{x_i}), \max_{x_i \in \mathcal{S}_q}(t^A_{x_i})\}$;
    if $t^A_{worst} < t^A_{best}$ **then**
      $(t^A_{best}, i^*) \leftarrow (t^A_{worst}, i)$;
    else if $(t^A_{worst} = t^A_{best}) \wedge (|\mathcal{S}_p||\mathcal{S}_q| \leq P^*)$ **then**
      $(P^*, i^*) \leftarrow (|\mathcal{S}_p||\mathcal{S}_q|, i)$;
$p, q \leftarrow p_{i^*}, q_{i^*}$;
$y_p \leftarrow$SPFD-decomposition$(p, \mathcal{S}_p)$;
$y_q \leftarrow$SPFD-decomposition$(q, \mathcal{S}_q)$;
return $h \leftarrow$interpolate from $\{x_0, x_1, y_p, y_q\}$ and f;

IV. Experiments

We compare the state-of-the-art engine (`EDR`) [2] with our engine (`IDR`) under two baseline conditions. In both cases, we begin with the following `ABC` flow [7]: first, we apply `resyn2rs` twice to remove redundancies and reduce the number of nodes. Finally, we minimize the circuit depth with SOP-balancing (`if -g`), and we do mapping to the `asap7` technology library, followed by delay resynthesis. For `EDR`, we use a database containing up to 3 input gates, which is better than any database achievable with enumeration. For `IDR`, we employ the database constructed as described in Section III-C.

Table I presents the results for the EPFL benchmarks. In the first experiment, we perform resynthesis after area-oriented mapping using $L = 4 \rightarrow 5 \rightarrow 6 \rightarrow 6$, with 8 structural cuts. This setup investigates high-effort delay-optimization in a region of the design space with abundant optimization opportunities. Under comparable area overhead, `IDR` achieves a 7.19% delay improvement over `EDR`.

In the second experiment, we perform delay-oriented mapping to investigate improvements beyond initial delay-driven optimization and mapping. We enumerate 12 structural cuts and perform resynthesis with $L = 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$, obtaining a delay improvement of 5.70% over the state of the art.

V. Conclusions

This paper discusses new algorithmic techniques for delay-oriented optimization after technology mapping. We address the limitations of a state-of-the-art engine, achieving substantial improvements, with an average delay reduction of 5.70%. Future works will use this engine for design space exploration.

TABLE I
Comparison with state-of-the-art delay resynthesis [2]

Area-Oriented Mapping

benchmark	area			delay			runtime	
	baseline	EDR[%]	IDR[%]	baseline	EDR[%]	IDR[%]	EDR[s]	IDR[s]
adder	70.40	104.60	91.52	3050.05	−32.08	−66.22	0.15	1.74
bar	129.61	−0.00	10.45	198.66	0.00	−0.26	0.03	0.23
div	2758.48	0.47	0.44	51598.82	−0.21	−0.28	9.44	8.68
hyp	13930.18	11.23	35.75	326123.66	−3.68	−11.33	278.24	1108.98
log2	1578.50	4.53	3.92	5531.19	−12.40	−14.47	147.92	100.28
max	152.58	151.39	14.43	1794.88	−4.54	−18.36	0.82	0.78
mult	1387.33	5.02	7.28	3641.84	−13.62	−28.52	42.31	41.07
sin	296.15	16.94	20.28	2683.45	−13.55	−18.13	19.44	19.44
sqrt	1254.19	58.49	133.07	87101.96	−14.46	−24.09	50.24	162.33
square	1089.26	5.31	5.52	3292.83	−29.89	−53.64	1.82	4.06
arbiter	467.82	5.21	4.40	851.30	−0.73	−1.16	0.43	0.81
cavlc	32.78	0.15	10.92	228.92	−1.29	−9.35	0.01	0.10
ctrl	5.85	8.38	16.75	125.05	−2.27	−12.32	0.00	0.16
dec	27.06	5.28	8.02	86.33	−4.04	−7.41	0.02	0.06
i2c	56.73	4.78	5.82	246.74	−19.48	−23.34	0.01	0.05
int2float	10.77	0.00	4.18	175.88	0.00	−2.22	0.00	0.15
memctrl	2063.17	0.68	0.65	1386.03	−6.97	−9.61	2.25	2.71
priority	29.40	66.05	99.05	1527.32	−35.16	−34.12	0.05	0.48
router	9.03	16.06	27.35	319.40	−17.59	−21.44	0.01	0.05
voter	548.25	1.56	1.42	1003.74	−4.41	−3.97	4.86	5.08
		23.31%	25.06%		**−10.82%**	**−18.01%**	27.91s	72.86s

Delay-Oriented Mapping

benchmark	area			delay			runtime	
	baseline	EDR[%]	IDR[%]	baseline	EDR[%]	IDR[%]	EDR[s]	IDR[s]
adder	82.67	19.41	73.70	1712.20	−3.92	−65.67	0.22	3.13
bar	183.25	−0.00	−0.00	158.12	0.00	0.00	0.49	4.05
div	3480.78	0.44	1.90	29100.80	−0.17	−0.24	165.28	97.54
hyp	23548.87	0.11	0.36	182533.34	−0.01	−0.02	303.47	425.21
log2	2405.75	0.36	0.61	3061.95	−0.03	−0.10	513.72	475.41
max	187.24	13.70	113.11	1150.23	−5.80	−16.79	3.32	11.98
mult	2089.54	0.43	1.59	2063.06	−1.06	−15.85	75.29	89.57
sin	526.48	0.09	0.60	1464.55	0.00	−0.01	176.86	180.27
sqrt	3471.26	1.84	3.91	44043.51	−0.00	−0.37	158.86	325.34
square	1133.42	2.23	3.78	1583.30	−10.84	−32.29	2.55	4.89
arbiter	448.25	1.89	0.56	500.84	−0.04	−0.76	0.95	1.30
cavlc	39.10	−0.00	0.26	160.48	0.00	0.00	0.04	0.34
ctrl	7.72	−0.00	11.01	91.42	0.00	0.00	0.01	0.15
dec	27.44	0.00	25.15	66.15	0.00	−0.45	0.04	0.95
i2c	62.73	−0.00	3.78	133.82	0.00	−1.11	0.03	0.25
int2float	12.29	0.41	10.09	139.55	−0.08	−1.38	0.03	0.39
memctrl	2267.93	0.09	0.74	800.37	−0.03	−0.23	3.37	6.62
priority	35.15	−0.00	7.74	564.25	0.00	−0.70	0.08	0.32
router	14.39	1.18	1.18	177.79	0.00	0.00	0.03	0.17
voter	1313.12	0.12	0.73	631.59	0.00	−0.00	13.97	21.78
		2.11%	13.04%		**−1.10%**	**−6.80%**	70.93s	82.48s

Acknowledgements

The authors thank Alessandro Tempia Calvino and Alan Mishchenko for the insightful discussions.

References

[1] W. Yang, L. Wang, and A. Mishchenko, "Lazy man's logic synthesis," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 597–604.

[2] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P.-E. Gaillardon, J. Olson, R. Brayton, and G. De Micheli, "Enabling exact delay synthesis," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 352–359.

[3] A. T. Calvino, G. De Micheli, A. Mishchenko, and R. Brayton, "Enhancing delay-driven lut mapping with boolean decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[4] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 395–402.

[5] A. Costamagna, A. T. Calvino, A. Mishchenko, and G. De Micheli, "Area-oriented resubstitution for networks of look-up tables," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[6] ——, "Area-oriented optimization after standard-cell mapping," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 1112–1119.

[7] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.