

From Forest to Tree: Prioritizing the Maximum Additional Delay in AQFP Circuit Design

Yinuo Bai¹, Mingjia Fan², Tsung-Yi Ho³, Zhou Jin^{4,*}

¹Department of Mathematics, Tufts University, Medford, MA, USA

²Department of Computer Science, China University of Petroleum (Beijing), Beijing, China

³Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR

⁴College of Computer Science and Technology, Zhejiang University, Hangzhou, China

*Corresponding author: z.jin@zju.edu.cn

Abstract—This paper presents a fast and scalable algorithm for buffer and splitter insertion in AQFP circuits. The method maps each wire to a homeomorphic graph, constructs an additional-delay-free multi-ary forest, and merges it into an optimal tree under delay and fanout constraints. The formulation guarantees per-wire optimality in terms of maximum additional delay, total additional delay, and internal node count. A circuit-level refinement further reduces redundant insertion by identifying and adjusting critical wires. On standard AQFP benchmarks, the proposed approach achieves $2.72\times$, $525.70\times$, and $1.33\times$ speedups over [1], [2], and [3], respectively, while maintaining comparable insertion counts and logic depths.

Index Terms—Superconductor-based logic, Adiabatic quantum-flux-parametron (AQFP), Buffer and splitter insertion

I. INTRODUCTION

Adiabatic Quantum-Flux-Parametron (AQFP) logic offers ultra-low static power dissipation and high switching speed, making it a promising platform for cryogenic and energy-efficient computing [4]–[6]. Due to fundamental differences from CMOS in device behavior, gate libraries, and multi-phase clocking, traditional CMOS-oriented EDA techniques cannot be directly applied to AQFP circuits [7], [8].

A major challenge in AQFP design is the pervasive use of buffers and splitters. Buffers are required to enforce strict input-delay alignment, while limited output current necessitates splitters to support fanout. Although indispensable for timing correctness, these elements significantly affect circuit depth, latency, and power consumption, and buffer–splitter insertion quickly becomes a runtime bottleneck in large-scale designs [9].

Existing solutions can be broadly classified into heuristic and global optimization approaches. Heuristic scheduling-based methods [10], [11] scale well but lack global optimality, while global optimization techniques [3], [12], [13] improve solution quality at the cost of high computational complexity. A recent divide-and-conquer formulation [1] captures AQFP timing semantics via lexicographic objectives, yet its dynamic-programming search exhibits exponential growth.

In this work, we present a fast and structure-aware buffer–splitter insertion algorithm that preserves lexicographic optimality while avoiding exhaustive search, enabling scalable synthesis of large AQFP circuits.

Algorithm 1 Construction of the Optimal Tree

Require: OD_{\max} , wire gates G

Ensure: T_{opt}

```

1:  $F \leftarrow \text{CONSTRUCTFOREST}(G)$ 
2: if  $F$  is a tree then
3:   return  $F$ 
4: end if
5:  $T_1 \leftarrow \text{MERGEFOREST}(F)$ 
6: if  $T_1$  satisfies the tight-capacity condition then
7:   return  $T_1$ 
8: end if
9:  $T_2 \leftarrow \text{RECONSTRUCTTREE}(T_1)$ 
10:  $T_3 \leftarrow \text{PRUNETREE}(T_2)$ 
11: return  $T_3$ 

```

II. PROPOSED ALGORITHM

We now present our buffer and splitter insertion algorithm. For each AQFP wire (a gate and the gates it drives), we map its insertion structure to a rooted multi-ary tree under two constraints: (i) buffers have an out-degree of 1 and splitters have an out-degree of $[2, OD_{\max}]$, where OD_{\max} denotes the maximum fanout supported by a splitter; (ii) each gate $G[i]$ has a least delay requirement $G[i].\text{Delay}$, and every path from the root to $G[i]$ must contain at least $G[i].\text{Delay}$ internal nodes. In this model, internal nodes represent buffers/splitters and leaves represent logic gates. We seek a tree T_{opt} that minimizes, in lexicographic order, the maximum additional delay, total additional delay, and number of internal nodes.

Our algorithm constructs T_{opt} in three steps: (1) build an additional-delay-free forest F ; (2) merge F into an initial tree T_1 ; and (3) refine T_1 by reconstruction and pruning to obtain the final tree. This constructive process avoids the exponential-state dynamic programming in [1] while preserving the same lexicographic objective.

A. Construction of a Delay-Free Forest F

Let G be the set of gates on a wire, and let $L[i]$ be the number of gates with least delay requirement i . We construct a multi-ary forest F in a bottom-up manner. Let $i_{\max} = \max_x G[x].\text{Delay}$ and define the number of internal nodes $I[i]$ at each level by

$$I[i_{\max}] = 0, \quad I[i] = \left\lceil \frac{L[i+1] + I[i+1]}{OD_{\max}} \right\rceil$$

for $i = i_{\max} - 1, \dots, 0$. Level i of F thus contains $I[i]$ internal nodes and $L[i]$ leaves. If $I[0] + L[0] = 1$, the structure is already a tree; otherwise, it is a forest.

By construction, all leaves in F appear exactly at their least delay level, so both the maximum and total additional delays are zero. One can further show that F uses the minimum possible number of internal nodes among all forests (or trees) satisfying these delay constraints. Hence F is optimal under zero additional delay and serves as the tightest possible “delay-free envelope” for subsequent merging.

B. Root-Node Integration: Forest F to Initial Tree T_1

To integrate the multiple roots of F into a single tree, we treat the roots as leaves of an additional multi-ary prefix. Let $x_0 = I[0] + L[0]$ be the number of roots of F and iterate

$$x_{i+1} = \left\lceil \frac{x_i}{OD_{\max}} \right\rceil$$

until $x_k = 1$. This yields a prefix of k levels with x_k, x_{k-1}, \dots, x_1 nodes. We regard all nodes in this prefix as internal and attach level 1 of F at level $k + 1$, obtaining a merged tree T_1 .

If, for every level i , the out-degree capacity of internal nodes exactly matches the number of children,

$$OD_{\max} I_1[i] = I_1[i + 1] + L_1[i + 1],$$

then T_1 already minimizes the number of internal nodes given its maximum additional delay k , and is in fact T_{opt} . Otherwise, T_1 attains the optimal maximum additional delay but may contain redundant internal nodes, which motivates further refinement.

C. Refinement by Reconstruction and Pruning

We refine T_1 in two stages. First, we reconstruct a tree T_2 by greedily reassigning nodes level by level. At each level i , we:

- place leaves whose least delay is exactly satisfied at level i , ensuring no leaf exceeds the optimal maximum additional delay;
- fill remaining capacity with internal nodes taken from deeper levels of T_1 , so that later levels retain enough internal-node capacity to remain feasible;
- if capacity still remains, place additional leaves (within their delay bounds) to reduce total additional delay.

This reconstruction preserves the optimal maximum additional delay while reducing the total additional delay as much as possible under the capacity and delay constraints. However, T_2 in general still inherits the internal-node count of T_1 and therefore may not yet be optimal in the third objective.

In the second stage, we prune T_2 to remove redundant internal nodes and obtain T_3 . Starting from the deepest levels and moving upward, we repeatedly:

- test whether removing an internal node at level j keeps the tree feasible by promoting up to OD_{\max} of its children from level $j + 1$;

- when feasible, delete the node and move eligible leaves upward (within their delay constraints), thereby reducing both internal-node count and additional delay;
- stop at level j when no further removal is possible without violating delay or out-degree constraints.

This process terminates when all levels are processed or no further internal nodes can be removed. The resulting tree T_3 has the same maximum and total additional delay as T_2 but a reduced number of internal nodes. One can show that T_3 coincides with T_{opt} for the given wire. Algorithm 1 summarizes the overall flow.

III. EXPERIMENTAL

A. From Wire-Level Optimization to Circuit-Level Design

Following [1], we extend per-wire optimization to the whole circuit by applying the algorithm to each wire iteratively. To better capture global interactions, we distinguish *cold wires*, where all sinks with the same delay requirement are aligned at the same level, from *hot wires*, which violate this property. Hot wires are the main source of global suboptimality. In our implementation, we prioritize nodes on hot wires, especially those that also serve as sources of other wires, during scheduling and insertion. This simple heuristic significantly reduces redundant buffers and splitters at the circuit level while preserving the wire-level optimality guarantees.

B. Experimental Result

Experimental results on ISCAS’85 and arithmetic benchmarks show that the proposed algorithm achieves competitive JJ counts and circuit depths compared with three state-of-the-art methods [1]–[3]. Across all benchmarks, the resulting circuit depths are identical to or within one level of prior methods, while JJ counts remain comparable.

More importantly, the proposed method consistently delivers substantial runtime improvements: $2.72\times$ over ICCAD’21, $525.70\times$ over ASPDAC’23, and $1.33\times$ over TCAD’25 on average. The speedup becomes more pronounced on large and deep circuits, where dynamic-programming-based and SMT-based approaches incur excessive runtime overhead.

IV. CONCLUSION

This work presents an efficient and scalable algorithm for buffer and splitter insertion in AQFP circuits. By analytically constructing an additional-delay-free forest and transforming it into an optimal multi-ary tree through structured merging, reconstruction, and pruning, the proposed method eliminates the exhaustive search required by dynamic programming while preserving lexicographic optimality in maximum additional delay, total additional delay, and internal-node count.

The resulting forest-to-tree framework is fast, theoretically grounded, and practically scalable, making it well suited for integration into full-stack AQFP design flows, especially in scenarios where runtime budgets dominate design iteration cycles.

REFERENCES

- [1] C.-Y. Huang, Y.-C. Chang, M.-J. Tsai, and T.-Y. Ho, "An optimal algorithm for splitter and buffer insertion in adiabatic quantum-flux-parametron circuits," in *International Conference On Computer Aided Design (ICCAD)*, 2021.
- [2] R. Fu, M. Wang, Y. Kan, N. Yoshikawa, T.-Y. Ho, and O. Chen, "A global optimization algorithm for buffer and splitter insertion in adiabatic quantum-flux-parametron circuits," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023.
- [3] R. Fu, M. Wang, Y. Kan, O. Chen, N. Yoshikawa, B. Yu, and T.-Y. Ho, "Buffer and splitter insertion for adiabatic quantum-flux-parametron circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 3, pp. 975–988, 2025.
- [4] N. Takeuchi, D. Ozawa, Y. Yamanashi, and N. Yoshikawa, "An adiabatic quantum flux parametron as an ultra-low-power logic device," *Superconductor Science and Technology*, vol. 26, no. 3, p. 035010, jan 2013.
- [5] O. A. Mukhanov, "Energy-efficient single flux quantum technology," *IEEE Transactions on Applied Superconductivity*, vol. 21, no. 3, pp. 760–769, 2011.
- [6] Y. Bai, E. Yi, W. Xing, B. Yu, and Z. Jin, "Unleashing the potential of aqfp logic placement via entanglement entropy and projection," in *Design Automation Conference (DAC)*, 2024.
- [7] E. Testa, S.-Y. Lee, H. Riener, and G. De Micheli, "Algebraic and boolean optimization methods for aqfp superconducting circuits," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021.
- [8] G. Pasandi and M. Pedram, "A dynamic programming-based, path balancing technology mapping algorithm targeting area minimization," in *International Conference On Computer Aided Design (ICCAD)*, 2019.
- [9] S. Razmkhah, R. S. Aviles, M. Li, S. Gupta, P. A. Beereel, and M. Pedram, "Challenges and unexplored frontiers in electronic design automation for superconducting digital logic," in *Design, Automation and Test in Europe Conference (DATE)*, 2024.
- [10] R. Saito, C. L. Ayala, and N. Yoshikawa, "Buffer reduction via n-phase clocking in adiabatic quantum-flux-parametron benchmark circuits," *IEEE Transactions on Applied Superconductivity*, vol. 31, no. 6, pp. 1–8, 2021.
- [11] B.-H. Wu and W.-K. Mak, "Optimization for buffer and splitter insertion in aqfp circuits with local and group movement," in *International Symposium on Physical Design*, 2024, p. 255–262.
- [12] S.-Y. Lee, H. Riener, and G. D. Micheli, "Irredundant buffer and splitter insertion and scheduling-based optimization for aqfp circuits," *International Workshop on Logic and Synthesis (IWLS)*, 2021.
- [13] —, "Beyond local optimality of buffer and splitter insertion for aqfp circuits," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022.