Buffer and Splitter Insertion for Adiabatic Quantum-Flux-Parametron Circuits

Rongliang Fu, Mengmeng Wang, Yirong Kan, Olivia Chen, Nobuyuki Yoshikawa *Fellow, IEEE*, Bei Yu, and Tsung-Yi Ho *Fellow, IEEE*

Abstract—**The extremely low-bit energy characteristic of the adiabatic quantum-flux-parametron (AQFP) circuit makes it a promising candidate for highly energy-efficient computing systems. However, by contrast with conventional circuit design, general logic synthesis tools can not make sure that the circuit functionality of generated AQFP circuits is correct. AQFP circuits require buffer and splitter insertion for dataflow synchronization at all clock phases of the circuit and multi-fan-out driving. Notably, buffers and splitters inserted take up much area and delay in AQFP circuits, also causing a significant increase in energy dissipation. To address this problem, this paper analyzes in detail why buffer and splitter insertion is necessary for AQFP circuits and proposes a global optimization framework for this purpose. This framework consists of three parts: (i) logic level assignment, (ii) splitter tree generation, and (iii) buffer insertion. An integer linear programming algorithm is proposed for the logic level assignment to estimate the globally optimal number of inserted buffers and splitters. Subsequently, a dynamic programming-based multi-way search tree generation algorithm is proposed to construct an optimal splitter tree for each net of the input circuit. Moreover, three optimization strategies are proposed to further enhance the effectiveness and efficiency of our framework. Experimental results on ISCAS'85 and EPFL benchmarks demonstrate the effectiveness and efficiency of our proposed framework compared with the state-of-the-art, particularly with significant advantages on large circuits.**

Index Terms—**Superconducting electronics, AQFP, buffer and splitter insertion, dynamic programming, integer linear programming**

I. Introduction

The increasing demand for high computational speed and low power consumption has led to a significant challenge in the field of semiconductor integrated circuits. In this context, Josephson junction (JJ)–based superconducting logic circuits have emerged as a promising alternative to traditional complementary metal-oxide-semiconductor (CMOS) technology for future computing systems, owing to their high speed and low-power consumption characteristics. As the active component of superconducting logic circuits, Josephson junctions serve as superconducting switching devices and offer rapid switching capabilities (approximately 1 ps per switch) with remarkably low energy consumption (less than 10^{-19} J per switch) [1]. These devices transmit information through voltage pulses that propagate along superconducting transmission lines almost without loss. Among the superconducting logic families, adiabatic quantum-flux-parametron (AQFP) logic [2] has garnered significant attention due to its highly energy-efficient nature. Derived from quantum-flux-parametron (QFP) logic proposed in 1985 [3], [4], AQFP logic applies the adiabatic switching operations to reduce energy dissipation substantially. Furthermore, AQFP logic utilizes alternating current (AC) for both clock signals and power supplies, thereby reducing the power consumption associated with direct current (DC) bias, while operating at frequencies in the gigahertz range. The study [5] has shown that the switching energy of an AQFP gate ranges from 10^{-20} J to 10^{-21} J at 5 GHz operation.

TABLE I: The comparison between CMOS and AQFP.

Circuit	AQFP	CMOS
Active component	Josephson junction	Transistor
Passive component	Inductor	Capacitor
Information	Current pulse	Voltage level
Clocking scheme	Synchronization (Clock signal)	Asynchronization
Fan-out	1 (Splitter for multiple fan-outs)	≥ 1
Power	Alternating current	Direct current

However, despite these advantages of AQFP logic, the unique characteristics of AQFP logic pose challenges in terms of the direct application of electronic design automation (EDA) tools to AQFP logic, which are typically designed for CMOS technology. TABLE I highlights the fundamental disparities between AQFP and CMOS technologies that make it necessary to develop customized EDA tools for AQFP logic. Firstly, each AQFP logic gate requires a clock signal to release its data signal to its successors and reset its state. That is, AQFP circuits are pipelined in nature. This necessitates the insertion of multiple buffers to ensure that all input data paths to the logic gate from primary inputs (PIs) have an equivalent number of logic gates (namely *logic level*), thereby ensuring correct circuit operation. Secondly, due to the driving limitation in output current, an AQFP logic gate can only drive one output. Addressing this limitation requires the insertion of a specially designed gate, known as a splitter, to enable multiple fan-outs. Therefore, buffer and splitter insertion is a critical step in

The work described in this paper was conducted in the Intelligent Design Automation Lab funded by The Hong Kong Jockey Club Charities Trust.

Rongliang Fu, Bei Yu, and Tsung-Yi Ho are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China. E-mail: {rlfu, byu, tyho}@cse.cuhk.edu.hk.

Mengmeng Wang and Nobuyuki Yoshikawa are with the Department of Electrical and Computer Engineering, Yokohama National University, Yokohama 240-8501, Japan. E-mail: wang-mengmeng-kj@ynu.jp, nyoshi@ynu.ac.jp.

Yirong Kan is with the Division of Information Science, Nara Institute of Science and Technology, Nara 630-0192, Japan. E-mail: kan.yirong@is.naist.jp.

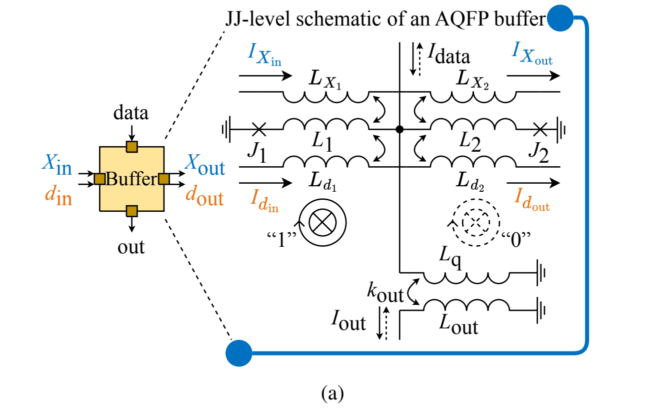
Olivia Chen is with the Department of Advanced Information Technology, Kyushu University, Fukuoka 819-0395, Japan. E-mail: olivia@ait.kyushu-u.ac.jp.

AQFP circuit design, and its legal implementation is essential to ensure the operation of an AQFP circuit is logically correct.

Currently, buffers and splitters still take up over half of the JJ count in the final AQFP circuit. As mentioned above, the JJ is an important component of AQFP circuits. Since its quantity is positively correlated with the area, delay, and energy consumption of AQFP circuits, its quantity serves as a critical metric in the performance estimation of AQFP circuits. So, given the rapid growth of fabrication capabilities for larger and more complex circuits, there is a pressing need for optimization algorithms of buffer and splitter insertion in AQFP circuits. The studies [6] and [7] inserted splitters and buffers separately after conventional logic synthesis, and then used timing-like heuristic algorithms to optimize the number of buffers and splitters inserted. In [8], a heuristic method was applied to implement non-redundant buffer and splitter insertion by scheduling and moving groups of gates, namely chunks. Although this method can reduce the number of buffers and splitters, the process of chunked movement may be endless due to alternating local moves in an upward or downward direction. Additionally, [9] focused on the buffer and splitter insertion problem on a single net. It introduced a dynamic programming-based algorithm that can provide an optimal solution of buffer and splitter insertion for each net within the input circuit after initializing a net delay assignment. However, it lacks a global optimization for the net delay assignment. Furthermore, [10] formulated the buffer and splitter insertion as a scheduling problem and also used a heuristic method to move chunks.

This paper focuses on how to minimize the number of buffers and splitters inserted after logic optimization from the perspective of overall circuit optimization on the basis of ensuring the legality of the AQFP circuit. This paper comprehensively analyzes the operation mechanism of AQFP logic and explains the reason of buffer and splitter insertion. To address this problem, we propose a global optimization framework to address the buffer and splitter insertion problem. It splits the buffer and splitter insertion process into three steps: (i) logic level assignment, (ii) splitter tree generation, and (iii) buffer insertion. First, we propose an integer linear programming (ILP)–based algorithm for the logic level assignment to estimate the globally optimal number of inserted buffers and splitters. Then, we propose a dynamic programming-based multi-way search tree generation algorithm to construct an optimal splitter tree for each net of the input circuit under a given logic level assignment. Finally, we insert the corresponding number of buffers according to the logic level gap on each net. Moreover, we present the following optimization strategies to improve the proposed framework.

- We apply a directed acyclic graph-based longest path algorithm to calculate the minimum logic level of each node as its lower boundary to accelerate solving the proposed ILP model.
- We design a new splitter tree constraint set to speed up the construction of the ILP model. The new design reduces the constraint scale for each splitter tree from exponential to linear.
- We explore the incorporation of circuit depth within the objective function of the proposed ILP model and design a new objective function to jointly optimize the circuit depth and the number of inserted buffers and splitters.



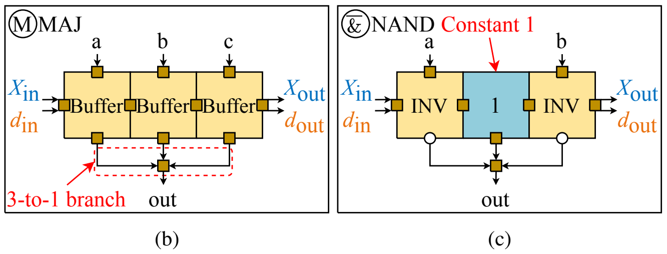


Fig. 1: (a) is a JJ-level schematic of an AQFP buffer. (b) and (c) are schematics of AQFP majority (MAJ) and NAND gates.

Moreover, experimental results on the ISCAS'85 [11] and EPFL [12] benchmarks demonstrate the effectiveness and efficiency of our proposed framework compared with the state-of-the-art, including two methods from ICCAD'21 [9] and DAC'22 [10].

- In terms of performance, our proposed framework with optimizations yields effective results on ISCAS'85 and simple arithmetic benchmarks [11], with an average reduction of 12.71% and 5.07% in the total number of inserted buffers and splitters compared with the methods in ICCAD'21 [9] and DAC'22 [10], respectively. Additionally, our framework can produce smaller or equal circuit depths than others for all generated circuits. Moreover, our proposed framework with optimizations is 417.01 times the speed of our framework without optimizations [13] while exhibiting a marginal 0.13% reduction in the number of inserted buffers and splitters.
- In terms of scalability, the experimentation extends to larger-scale circuits sourced from the EPFL benchmark [12]. Notably, our proposed framework achieves an average reduction of 32.06% and 9.13% in the total number of inserted buffers and splitters over the baselines, respectively. Meanwhile, our framework excels in efficiency, showcasing an average runtime reduction of 83.20% and 27.04% over the baselines, respectively.

The rest of this paper is organized as follows. Section II provides an overview of AQFP circuits and delves into the necessity of buffer and splitter insertion within AQFP circuits. Section III introduces the problem formulation. Section IV

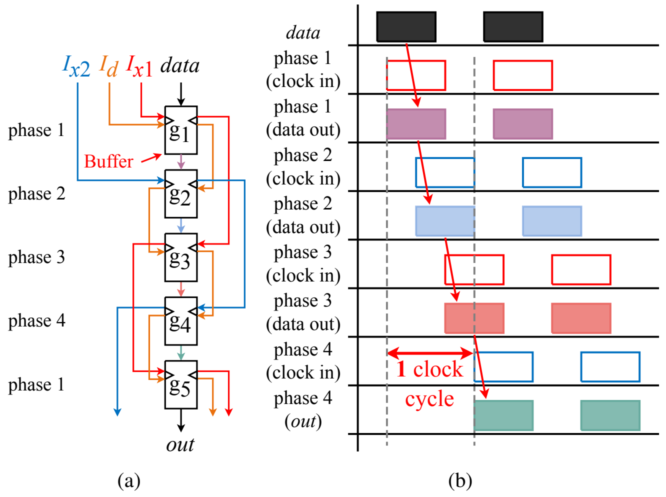


Fig. 2: (a) is an AQFP buffer chain driven by the 4-phase clocking scheme, where the latency between two gates is a quarter cycle of AC excitation currents I_{x_1} and I_{x_2}. (b) shows data propagation between adjacent clock phases in (a).

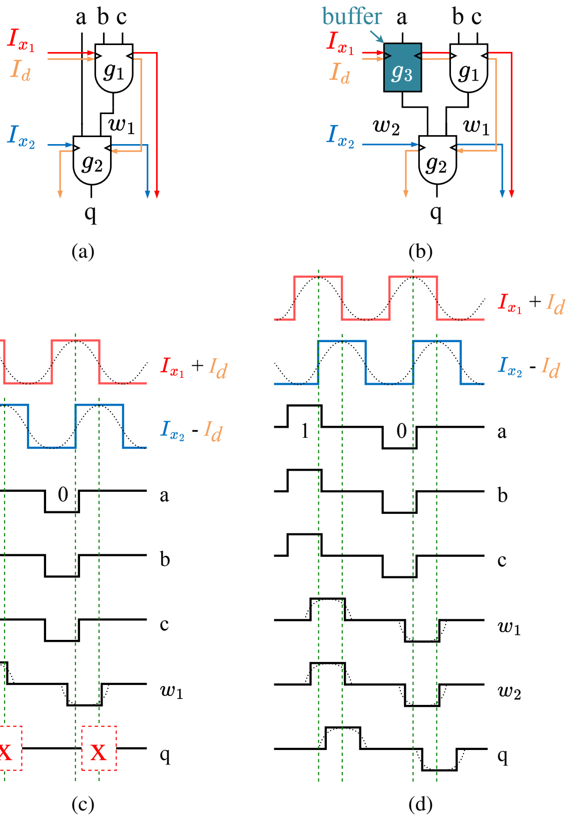


Fig. 3: A gate-level schematic shows the necessity of buffer insertion for correct operation in the AQFP circuit with the function $q = abc$, where the 4-phase clocking scheme is applied. (a) and (b) are schematics before and after buffer insertion, respectively, where I_{x1} and I_{x2} are AC clock signals with a phase separation of 90°, and I_d is the DC input, which applies an offset flux of half an SFQ to each logic gate. Besides, a, b, and c are three data inputs, and q is a data output; (c) and (d) are timing schematics of (a) and (b), respectively.

describes our proposed buffer and splitter insertion framework. Section V presents our optimizations for the proposed framework. Section VI demonstrates the effectiveness and efficiency of our framework. Section VII summarizes this paper and discusses potential avenues for further optimizing AQFP circuit design.

II. Preliminaries

A. Adiabatic Quantum-Flux-Parametron Logic

AQFP logic employs adiabatic superconductors to achieve the operation with energy dissipation levels approaching the thermodynamic and quantum limits [14], making it a promising option for constructing highly energy-efficient computing systems. Typically, AQFP logic cells are constructed by integrating four fundamental components: Buffer, Inverter (INV), Constant, and Branch. The buffer has two JJs J_1 and J_2, and its JJ-level schematic is illustrated in Fig. 1(a), where L_1 and L_2 are inductors, and $I_{d_{in}}$ is the DC. Initially, a small input current I_{in} is applied to the buffer, followed by the supply of AC excitation current $I_{X_{in}}$ to the inductors L_{X_1} and L_{X_2}, which are magnetically coupled with L_1 and L_2 respectively. Consequently, a single flux quantum (SFQ) is stored in either the left loop ('1') or the right loop ('0'). The output current I_{out} is produced by the signal transformer, and its direction indicates the logic state of the buffer. This signal transformer contains two inductors L_q and L_{out}, which are coupled by a coupling factor k_{out}. Notably, if k_{out} is positive, this cell operates as a buffer; Conversely, it operates as an inverter.

The design of AQFP logic cells is exemplified through AQFP majority (MAJ) and NAND gates. Fig. 1(b) showcases an AQFP majority gate with the function $M(a, b, c) = ab + ac+bc$, comprising three buffers and a 3-to-1 branch. The presence of the majority gate makes the majority-inverter graph (MIG) a natural fit for AQFP circuit design. By fixing input c to Constant **1** (**0**), an MAJ gate can become an OR (AND) gate with the function $M(a, 1, b) = a + b$ ($M(a, 0, b) = ab$). So, the 3-input majority gate, the 2-input AND gate, and the 2-input OR gate all have 6 JJs. Moreover, since the buffers and inverters in AQFP logic are interchanged by adjusting k_{out}, the AQFP NAND gate can be designed by replacing two buffers in the OR gate with two INVs, i.e., $M(\bar{a}, 1, \bar{b}) = \bar{a} + \bar{b} = \overline{ab}$, as shown in Fig. 1(c).

B. Clock-synchronized Data Propagation of AQFP Logic

Since the operation of all AQFP gates is synchronized with an AC power source that serves as both an excitation current and a clock signal, the multi-phase clocking scheme is commonly employed in AQFP circuit design. This scheme utilizes multiple clock sources with phase differences to facilitate data propagation in AQFP circuits, where the clock signal should arrive after the data signal during a certain period. Specifically, this enables data transmission across various logic levels during a substantial overlap of adjacent clock phases. Take the 4-phase clocking scheme [15] as an example. Fig. 2(a) shows an AQFP buffer chain with the 4-phase clocking scheme. I_{x1}

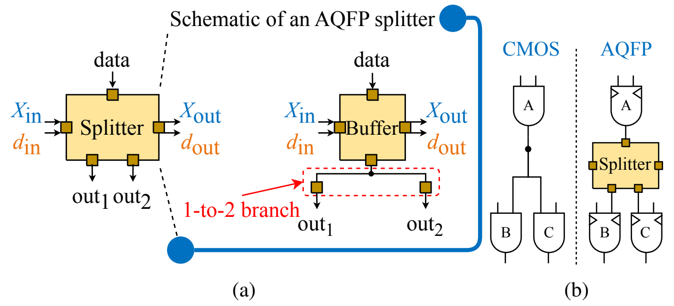


Fig. 4: (a) The schematic of a 1-to-2 splitter. (b) The multi-fan-out comparison between CMOS logic and AQFP logic.

and I_{x2} are two AC clock signals with a phase separation of $90°$, and I_d is the DC input for an offset flux supply to each logic gate. The input *data* in Fig. 2(a) propagates phase by phase following clock signals, as shown in Fig. 2(b).

To ensure the operation of the AQFP circuit is correct, all inputs to the AQFP gate must be in the same logic level. To achieve this requirement, buffer insertion is essential. The example shown in Fig. 3 illustrates the necessity of buffer insertion in an AQFP circuit with the function $q = abc$. Two inputs to AND gate g_2 in Fig. 3(a) have different logic levels. When the first rising edge of clock signal I_{x_2} arrives at AND gate g_2, a significant delay relative to input a results in a lack of effective overlap between data signal a and clock signal I_{x_2}. Consequently, the output 'x' of g_2 in Fig. 3(c) may be indeterminate [16]. Following the insertion of a buffer g_3 in Fig. 3(b), two inputs to gate g_2 are now harmonized to the same logic level, enabling gate g_2 to output '1' correctly, as shown in Fig. 3(d).

C. Fan-out Limitation of AQFP Logic

In contrast to CMOS circuits, AQFP gates exhibit a low output driving ability, typically only capable of driving a single output. Consequently, the AQFP circuit design necessitates a special cell, known as a splitter, to facilitate the realization of multiple fan-outs. The splitter consists of a buffer and a 1-to-X branch, typically with $2 \leq X \leq 4$. Fig. 4(a) shows the schematic of a 1-to-2 splitter, where $I_{X_{in}}$ is the AC clock signal and d_{in} is the DC input. In an AQFP circuit, when the output signal of a logic gate (A) needs to be transmitted to multiple gates (B and C), splitters must be positioned at its output to enable multiple fan-outs, as shown in Fig. 4(b). Since the splitter requires a clock signal, its insertion for multi-fan-out implementation can cause a delay increase in the data path. This can impact buffer insertion for clock-synchronized data propagation. Notably, diverse splitter tree configurations for large fan-out implementation can result in varying numbers of inserted buffers and splitters, as detailed in our prior research [13].

D. Circuit Design of AQFP Logic

Given these unique characteristics of AQFP logic, the design of AQFP circuits diverges from that of CMOS circuits. Existing methods typically separate the AQFP circuit design process into two key phases, including logic optimization and the subsequent insertion of buffers and splitters. Specifically, early studies [17], [6] have leveraged the Yosys synthesis suite [18], an open-source logic synthesis tool, to generate a CMOS-like gate-level netlist. This netlist is then subject to buffer and splitter insertion to align with AQFP circuit design requisites. Yosys tool usually employs AND-OR-inverter (AOI)-based logic optimization techniques to refine the given function and subsequently can map the resultant intermediate netlist into a custom AQFP standard cell library. However, owing to the native majority cell in AQFP logic, which can have a more compact representation for logic function with the same physical overhead as the AND/OR cell, MIG-based logic synthesis approaches [7], [19], [20] have been introduced for AQFP circuits.

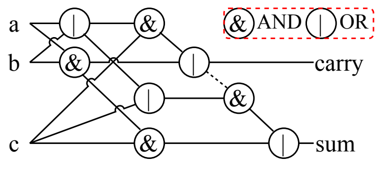
For instance, considering a 1-bit full adder, Fig. 5(a) shows an AOI-based implementation with 8 gates and circuit depth of 5, followed by MIG-based logic optimization to minimize gate count and circuit depth, resulting in an MIG-based 1-bit full adder with 3 gates and circuit depth of 2 as shown in Fig. 5(b). Subsequent buffer and splitter insertion for Fig. 5(b) yields the final legal 1-bit AQFP full adder with 4 buffers and 4 splitters, as shown in Fig. 5(c). Furthermore, the latest study [21] of AQFP logic synthesis integrates these two phases and employs Bayesian optimization to explore the best MIG structure concerning the actual AQFP cost after buffer and splitter insertion for AQFP circuit design requisites.

III. Problem Formulation

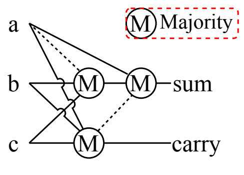
A. Terminology

An AQFP circuit can be represented by a directed acyclic graph $G(V, E)$, where V is the node set comprising logic gates, and E is the edge set comprising nets. The node set $V = I \cup O \cup C$ consists of the set I of primary inputs, the set O of primary outputs (POs), and the set C of logic gates. For a node $v \in V$, $E_i(v)$ is the set of its input edges, and $E_o(v)$ is the set of its output edges. The edge set $E = E_u \cup E_m$ consists of the set E_u of 2-pin nets and the set E_m of over-2-pin nets. For an edge $e \in E$, e_s is the source of the edge e, and e_t is the set of sinks of the edge e. If the edge e is a 2-pin net, i.e., $e \in E_u$, then $|e_t| = 1$, otherwise $|e_t| > 1, e \in E_m$. Following buffer and splitter insertion, an extended graph $G'(V', E')$ can be derived, where $V' = V \cup B \cup S$, with B and S denoting the sets of buffers and splitters, respectively. The maximum fan-out of the splitter is denoted as X, where $X > 1$. For a node $v \in V$, $\text{FI}(v)$ and $\text{FO}(v)$ represent the set of its fan-in nodes and its fan-out nodes, respectively. Additionally, for a PI $i \in I$, the set of its fan-in nodes is empty. For a PO $o \in O$, the set of its fan-out nodes is also empty.

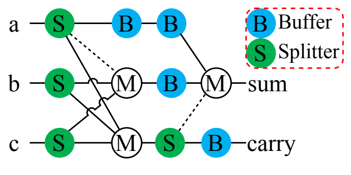
Furthermore, for uniformity in figures, all nodes are categorized into multiple columns in terms of their logic level, as shown in Fig. 6. So, for any node $v \in V_i$ in the i^{th} column, its logic level is i, i.e., $L(v) = \max_{u \in \text{FI}(v)} L(u) + 1 = i, v \in V_i$. This paper assumes that all primary inputs arrive at the same clock phase and that all primary outputs are also produced at the same clock phase. That is, all primary inputs are allocated to the first column, and all primary outputs are allocated to the



(a)



(b)



(c)

Fig. 5: The example of a 1-bit AQFP full adder circuit design. (a) An AOI-based 1-bit full adder. (b) MIG-based optimized result of (a). (c) Legal 1-bit AQFP full adder circuit after buffer and splitter insertion for (b).

last column. Therefore, the logic level of each primary input is set to 0, while the logic level of each primary output is the maximum logic level l_{max}, corresponding to the circuit depth $d = l_{max} - 1$. To minimize the circuit depth, the logic level of primary outputs should be minimized.

B. Problem Formulation

The primary focus of this paper is the buffer and splitter insertion for AQFP circuits. Given that the JJ number of a buffer is the same as that of a splitter, the minimization of the total JJ number caused by buffer and splitter insertion can be equivalent to that of the total number of inserted buffers and splitters. Our objective is to minimize the number of inserted buffers and splitters while meeting the requirements of AQFP circuit design. Therefore, the buffer and splitter insertion problem for AQFP circuit design can be formulated as follows:

- Input:
  1) A given circuit $G(V, E)$,
  2) The maximum fan-out X of the splitter.
- Output:
  An extended circuit $G'(V', E')$, where all inputs to each logic gate have the same logic level and all gate outputs have a single fan-out.
- Constraints:
  1) Fan-out limitation:
  $$\forall e \in E', |e_t| = 1, \tag{1}$$
  which ensures that each edge in circuit G' has only two pins, including one source and one sink.
  2) Path balance:
  $$\forall u \in \mathrm{FI}(v), v \in V', L(v) = L(u) + 1, \tag{2}$$
  which ensures that all inputs to each node in circuit G' have the same logic level, thus enabling circuit G' to meet the prerequisite for clock-synchronous data propagation.
  3) PI alignment:
  $$\forall i \in I, L(i) = 0, \tag{3}$$
  which makes primary inputs arrive at the same clock phase.
  4) PO alignment:
  $$\forall o \in O, L(o) = \max_{v \in V'} L(v) + 1, \tag{4}$$
  which makes primary outputs occur at the same clock phase.

  So, Equations (1) and (2) guarantee the final circuit is legal after buffer and splitter insertion.
- Goal:
  $$\min_{G'} |S \cup B|, \tag{5}$$
  which means minimizing the number of buffers and splitters inserted into circuit G' while maintaining the functional structure of the original circuit G.

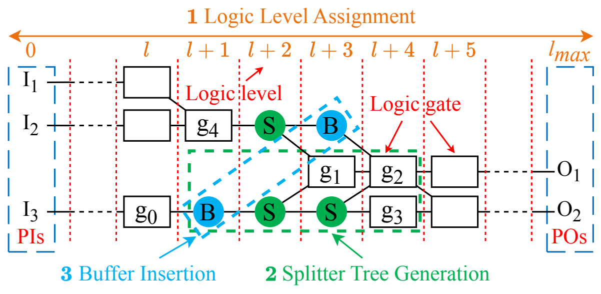


Fig. 6: The flow of the proposed method contained logic level assignment, splitter tree generation, and buffer insertion.

IV. Buffer and Splitter Insertion

This section proposes a global optimization framework for buffer and splitter insertion, as shown in Fig. 6. This framework divides the process of buffer and splitter insertion into three phases. First, an integer linear programming model is proposed to assign the logic level for each logic gate. Then, based on a predefined logic level assignment, splitter trees are generated for all over-2-pin nets by an optimal multi-way search tree generation method. Finally, buffers are inserted for all 2-pin nets to ensure the path balance.

The assignment of logic levels to all logic gates within circuit G has a significant impact on the result of its buffer and splitter insertion. Fig. 7 showcases two distinct solutions for buffer and splitter insertion of the input circuit shown in Fig. 7(a). It is assumed that inserting buffers and splitters is optimal after assigning the logic level of each logic gate. Fig. 7(b) heuristically determines the logic levels of logic gates, i.e., $L(g_1) = 3$, $L(g_2) = 3$, $L(g_3) = 2$, $L(g_4) = 4$, and $L(g_5) = 5$. After inserting buffers and splitters, the result requires eight buffers and three splitters, as shown in Fig. 7(c). By contrast, Fig. 7(d) makes an optimal logic level assignment for all logic gates, i.e., $L(g_1) = 2$, $L(g_2) = 3$, $L(g_3) = 3$,

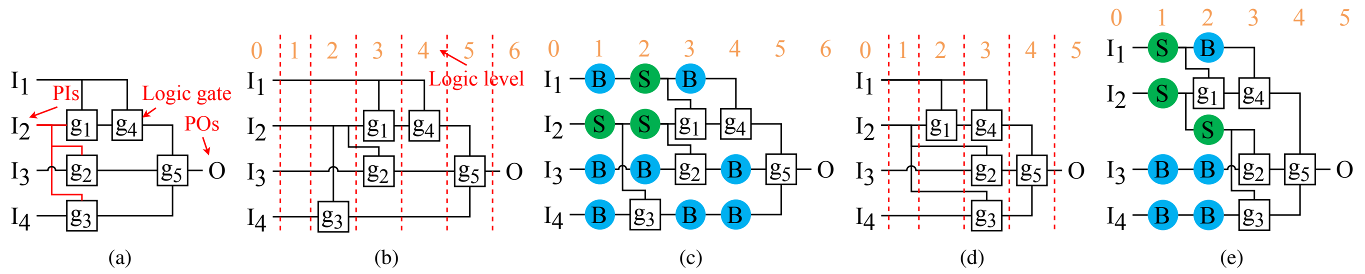


Fig. 7: The example of buffer and splitter insertion under various logic level assignments when a maximum fan-out of the splitter of 2. (a) is the initial input circuit. (b) and (d) show its two distinct logic level assignment solutions, where the orange number at the top of each column denotes the logic level of each node within the respective column. Subsequently, (c) and (e) show the results after buffer and splitter insertion corresponding to the configurations in (b) and (d), respectively.

$L(g_4) = 3$, and $L(g_5) = 4$. Following inserting buffers and splitters, the final result only requires five buffers and three splitters, as shown in Fig. 7(e). Moreover, the circuit depth of Fig. 7(e) is also one smaller than that of Fig. 7(c). Therefore, the key to the buffer and splitter insertion problem is to handle the logic level assignment from the global perspective to minimize the number of buffers and splitters required.

A. *Logic Level Assignment*

Definition 1 (Multi-way Tree)**.** *A multi-way tree is a tree data structure where each node has multiple children. For instance, all nodes in a X-way tree have at most X children.*

Definition 2 (Complete Multi-way Tree)**.** *A complete multi-way tree is a multi-way tree in which all the levels are completely filled except the lowest-level nodes, which are filled from as far left as possible. For instance, all nodes in a complete X-way tree that are not in the last level have X children.*

Definition 3 (Path Sum)**.** *The path sum of a node v in a multi-way tree is the index of the layer where node v is located, i.e. the depth of node v in this multi-way tree.*

In this paper, we denote the aggregate sum of all leaves' path sums as the all path sum of the multi-way tree.

Lemma 1. *Given a complete X-way tree with n leaves, its tree depth is $h(n, X) = \lceil \log_X n \rceil$.*

Lemma 2. *Given a complete X-way tree with n leaves, the number of its non-leaf nodes is $\beta(n, X) = \lceil \frac{n-1}{X-1} \rceil$.*

Lemma 3. *Given a complete X-way tree with n leaves, the number of nodes in the i^{th} level is $\ell(i, X) = X^i, i \in [0, h(n, X))$, and the number of nodes in the last level is $\ell(h(n, X), X) = \lceil \frac{n - \ell(h(n,X)-1, X)}{X-1} \rceil + n - \ell(h(n, X) - 1, X)$.*

Lemma 4. *Given a complete X-way tree with n leaves, its all path sum is $f(n, X) = \ell(h(n, X), X) * h(n, X) + (n - \ell(h(n, X), X)) * (h(n, X) - 1)$.*

The assignment of the logic level for each logic gate can be achieved by estimating the minimum number of buffers and splitters necessary for the circuit. Since the delay between two connected gates is contingent upon the gap of their logic levels, the total delay of a net e can be calculated as $p(e) = \sum_{t \in e_t} (L(t) - L(e_s) - 1)$. In the case of a 2-pin net, the number of buffers inserted is equal to its delay, while for an over-2-pin net, the number of inserted buffers is less than its delay due to the presence of inserted splitters. However, the dynamic nature of the splitter tree structure for an over-2-pin net poses challenges in accurately calculating the minimum number of buffers and splitters inserted for this net. To address this challenge, the utilization of the complete multi-way tree is advocated in the logic level assignment process to evaluate the number of buffers and splitters required by an over-2-pin net e. That is because the complete multi-way tree has the minimum number of nodes and the minimum tree depth for the same number of leaves according to Theorems 1 and 2, thereby requiring that all path sum $f(|e_t|, X)$ of the splitter tree is no less than that of the complete multi-way tree with $|e_t|$ leaves, namely, the fan-out limitation constraint.

Theorem 1. *Given a tree with n leaves where each node has at most X children, it has the minimum number of nodes when it is a complete X-way tree.*

Proof. Assume that there exists a tree A with n leaves and each non-leaf node having at most X children that has fewer nodes than a complete X-way tree B with n leaves. So, tree A must have fewer non-leaf nodes than tree B. According to Definition 2, each non-leaf node in a complete X-way tree has exactly X children, except for the nodes at the last level. So, non-leaf nodes in tree A with more than X children exist, which contradicts the assumption. Therefore, a complete multi-way tree has the minimum number of nodes. □

Theorem 2. *Given a tree with n leaves where each node has at most X children, it has the minimum tree depth when it is a complete X-way tree.*

Proof. Assume that there exists a tree A with n leaves and each non-leaf node having at most X children that has a shallower depth than a complete X-way tree B with n leaves.

So, tree A must have fewer levels than tree B. According to Definition 2, each level within tree B is completely filled with the maximum possible number of nodes as many as possible. In tree A, there can be levels with fewer nodes than the maximum possible number of nodes. This means that tree A requires more levels to accommodate the same number of leaves as tree B, which contradicts the assumption. Therefore, a complete multi-way tree has the minimum tree depth. $\square$

Specifically, a complete X-way tree with $|e_t|$ leaves is first built, enabling the computation of its all path sum denoted as $f\left(|e_t|, X\right)$ and the number of its non-leaf nodes denoted as $\beta\left(|e_t|, X\right)$. Subsequently, the residual shared delay from all leaf nodes can be fulfilled by a buffer chain, the length of which can be approximated to $\frac{p(e)-f(|e_t|,X)}{|e_t|}$. Consequently, the number of buffers and splitters required by the net e can be approximated to $\frac{p(e)-f(|e_t|,X)}{|e_t|}+\beta\left(|e_t|, X\right)$, that is, the sum of the length of the buffer chain and the number of non-leaf nodes in the complete X-way tree. For instance, assuming a maximum fan-out of 2 for the splitter in Fig. 6, the $(3+1)$–pin net between the source g_0 and the sinks (g_1, g_2, and g_3) can consist of a buffer chain with only one buffer and a complete binary tree with two splitters as non-leaf nodes. Therefore, the number of buffers and splitters required by the circuit can be formulated as:

$$\sum_{e\in E_u} p(e) + \sum_{e\in E_m}\left(\frac{p(e)-f\left(|e_t|, X\right)}{|e_t|}+\beta\left(|e_t|, X\right)\right). \quad (6)$$

According to Lemmas 1 to 4, $f\left(|e_t|, X\right)$ and $\beta\left(|e_t|, X\right)$ are constants for a specified edge e and a maximum fan-out of X. Therefore, the problem of minimizing the number of splitters and buffers can be formulated as an integer linear programming problem as follows:

$$\min \quad \sum_{e\in E}\left(\frac{1}{|e_t|} * \sum_{v\in e_t} L(v) - L(e_s)\right), \quad (7)$$

$$\text{s.t.} \quad \forall v \in I, L(v) = 0, \quad (8)$$

$$\forall v, u \in O, L(v) = L(u), \quad (9)$$

$$\forall v \in O \cup C, \forall u \in \mathrm{FI}(v), L(v) \geq L(u) + 1, \quad (10)$$

$$\forall e \in E_m, \forall \boldsymbol{s} \subseteq e_t, \sum_{t\in \boldsymbol{s}} \left(L(t) - L(e_s) - 1\right) \geq f\left(|\boldsymbol{s}|, X\right). \quad (11)$$

Considering two examples in Figs. 7(b) and 7(d), their values corresponding to Equation (7) are $\frac{115}{6}$ and $\frac{97}{6}$, respectively, thereby demonstrating the capability of our ILP model in distinguishing the superior logic level assignment. Equations (8) and (9) align PIs and POs so that all PIs and all POs are on the first and last levels, respectively. To limit the fan-out number of the edge e, as shown in Equation (11), all path sum of the multi-way tree constructed from any subset $\boldsymbol{s}$ of its sinks e_t must be no less than that of the complete multi-way tree with $|\boldsymbol{s}|$ leaves, that is because the complete multi-way tree can meet the fan-out requirement with the minimum all path sum. For Equation (11), it is imperative to enumerate all potential subsets of sinks associated with each edge. To enumerate all subsets of sinks e_t for the edge e, the following characteristic function $\chi_{\boldsymbol{s}}(x)$ is employed to identify whether the sink $x \in e_t$ exists $\boldsymbol{s}$.

$$\chi_{\boldsymbol{s}}(x) = \begin{cases} 1 & \text{if } x \in \boldsymbol{s}, \\ 0 & \text{if } x \notin \boldsymbol{s}. \end{cases} \quad (12)$$

$\{\chi_{\boldsymbol{s}}(x) \mid x \in e_t\}$ can be represented by a 0-1 sequence. For example, considering the sinks $e_t = \{x_1, x_2, x_3, x_4, x_5\}$, $\{x_1, x_2, x_4\}$ can be encoded as 11010. Consequently, all possible sink combinations within e_t, i.e., all subsets of sinks e_t, can be generated by full permutations of the 0-1 sequence of length $|e_t|$. Then, the sinks corresponding to the index of the '1' element in each permutation form a subset.

However, there are $2^{|e_t|}$ subsets for the edge e, which renders the exhaustive enumeration of all subsets infeasible for the edge with a substantial number of sinks. While increasing the number of enumerated subsets may enhance the adequacy of splitter tree constraints, it is essential to consider the trade-off between effectiveness and efficiency. Therefore, for edges with over 30 sinks, an approximate approach is adopted where $|e_t|$ rank sequences are initially generated via the random shuffle algorithm, each sequence determining the specific sink order. Subsequently, for the sink set ordered by each sequence, subsets ranging in length from 2 to $|e_t|$ are selectively chosen from the left, resulting in a total of $|e_t| * (|e_t| - 1)/2$ subsets. Through this method, a total of $|e_t| * |e_t| * (|e_t| - 1)/2$ subsets are effectively selected to govern the maximum out-degree of all nodes within the multi-way tree. Given the complexity arising from the multitude of decision variables and constraints, determining the appropriate range for decision variables becomes challenging. Consequently, an integer linear programming solver incorporating a linear programming-based branch-and-bound algorithm is leveraged to address and resolve the optimization problem outlined in Equation (7), ultimately facilitating the determination of the logic level for each logic gate.

B. Splitter Tree Generation

After assigning the logic level of each logic gate, the primary concern shifts to the buffer and splitter insertion problem for a single net. For a 2-pin net, the number of required buffers is equivalent to its delay, without the need for any splitters. For an over-2-pin net, the task involves creating a splitter tree composed of buffers and splitters. This process poses a huge challenge due to the dynamic nature of the splitter tree structure. To address this problem, we regard the splitter tree constructed for each single net as a multi-way search tree, where the delay of each node serves as its key, and the number of non-leaf nodes indicates the number of buffers and splitters required. The following terms are defined to solve the splitter tree generation problem of a $(n+1)$-pin net e.

- The nodes of the splitter tree: The leaf nodes of the splitter tree represent the sinks e_t of the net e, where $n = |e_t|, n \geq 1$. The out-degree of each non-leaf node must be in the range $[1, X]$. A non-leaf node is a buffer if its out-degree is 1; otherwise, it is a splitter. Besides, the delay of the leaf node v is denoted as $v.delay$.

```
Algorithm 1: Optimal splitter tree generation
  Input: An over-2-pin net $e$ with a source $s$ and $n$ sinks
         $\mathbf{t}$, and the maximum fan-out $X$ of the splitter
  Output: A generated splitter tree
1  $\mathbf{t}_i.delay = L(\mathbf{t}_i) - L(s) - 1, i \in [1, n]$
2  Reorder t in ascending order based on their delays
3  $H = \lceil \log_X n \rceil + 1$
4  $D = \max_{i \in [1,n]} \mathbf{t}_i.delay + H$
5  $\mathbf{dp} = \{+\infty, +\infty, +\infty\}^{n \times n \times X \times (D+1)}$
6  $\mathbf{pt} = \{-1, -1\}^{n \times n \times X \times (D+1)}$
7  // Initialization
8  for $s \in [1, \min\{n, X\}]; l \in [1, n - s + 1]$ do
9      $D_c = \mathbf{t}_l.delay + H$
10     $r = l + s - 1$
11     if $s == 1$ then
12         for $d = 0$ to $D_c$ do
13             $\delta = |d - \mathbf{t}_l.delay|$
14             $\mathbf{dp}_{l,l,1,d} = d > \mathbf{t}_l.delay ? \{\delta, \delta, 0\} : \{0, 0, \delta\}$
15     else
16         $\mathbf{dp}_{l,r,s,D_c} = \mathbf{dp}_{l,r-1,s-1,D_c} + \mathbf{dp}_{r,r,1,D_c}$
17 // Calculate the minimum cost
18 for $len \in [2, n]; l \in [1, n - len + 1]$ do
19     $D_c = \mathbf{t}_l.delay + H$
20     $r = l + len - 1$
21     for $d = D_c - 1$ to $0$; $s \in [1, \min\{len, X\}]$ do
22         if $s == 1$ then
23             $\mathbf{dp}_{l,r,s,d} = \min_{k \in [1, \min(len, X)]} \mathbf{dp}_{l,r,k,d+1} + \{0, 0, 1\}$
24             $\mathbf{pt}_{l,r,s,d} = \{-1, k\}$
25         else
26             $\mathbf{dp}_{l,r,s,d} = \min_{\substack{k \in [l+u-1, r-v] \\ s=u+v, u \in [1,s)}} \mathbf{dp}_{l,k,u,d} + \mathbf{dp}_{k+1,r,v,d}$
27             $\mathbf{pt}_{l,r,s,d} = \{k, u\}$
28 return a splitter tree built by backtracking using pt
```

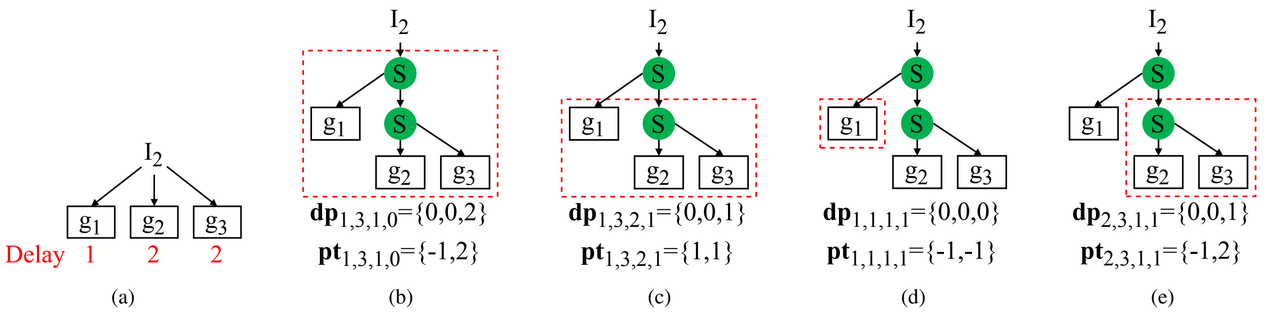
- The generation cost of the splitter tree: The number of inserted buffers and splitters in the generated splitter tree must be minimized. Furthermore, in order to uphold the integrity of the logic level assignment outcome, two extra costs are introduced. First, since the delay of each leaf is fixed based on assigned logic levels, we should minimize the increase in it to avoid the impact on other nodes. Second, we also minimize the total extra delay to prevent adverse repercussions on the result of the logic level assignment. Therefore, for a generated splitter tree, its cost [9] $\{ed, ted, nn\}$ consists of three parts, including maximum extra delay $ed = \max_{t \in e_t} d(t) - t.delay$ of leaf nodes, total extra delay $ted = \sum_{t \in e_t} \{d(t) - t.delay\}$ of leaf nodes, and the number of non-leaf nodes nn of generated tree. Moreover, we define the addition operation of two costs $\{ed_1, ted_1, nn_1\}, \{ed_2, ted_2, nn_2\}$ as $\{\max\{ed_1, ed_2\}, ted_1 + ted_2, nn_1 + nn_2\}$. The comparison operation between two costs is performed item by item, with the priority decreasing from left to right.

Inspired by the optimal multi-way search tree [22], an optimal splitter tree generation algorithm based on dynamic programming is proposed, as shown in Algorithm 1. This algorithm can ensure the optimality of buffer and splitter insertion for a single net under a specified logic level assignment. Firstly, the delay of each leaf node is calculated (line 1). Then, leaf nodes $\mathbf{t}$ are reordered in ascending order regarding their delays (line 2), and the maximum tree depth D is calculated (line 4). Since an X-way tree with $n = |\mathbf{t}|$ leaf nodes has the minimum height $H = \lceil \log_X n \rceil + 1$ when it is a complete X-way tree, there exists a splitter tree with n leaf nodes, where the extra delay of its leaf node with the maximum delay is at most H, meaning that the ed of the optimal splitter tree is not greater than H. **dp** and **pt** are two four-dimensional arrays. $\mathbf{dp}_{l,r,s,d}$ records the cost of s splitter subtrees with leaf nodes $\mathbf{t}_l, \mathbf{t}_{l+1}, ..., \mathbf{t}_r$, whose root nodes' depth is d. $\mathbf{pt}_{l,r,s,d} = \{m, s'\}$ records two substructures: one $\{l, m, s', d\}$ with s' root nodes and $m - l + 1$ leaf nodes $\mathbf{t}_l, \mathbf{t}_{l+1}, ..., \mathbf{t}_m$, and the other $\{m+1, r, s-s', d\}$ with $s - s'$ root nodes and $r - m$ leaf nodes $\mathbf{t}_{m+1}, \mathbf{t}_{m+2}, ..., \mathbf{t}_r$. When $m = -1$, the current substructure has only one root node with s' fan-outs. Lines 5-16 initialize **dp** and **pt**. When the substructure $\{l, r, s, d\}$ with leaf nodes $\mathbf{t}_l, \mathbf{t}_{l+1}, ..., \mathbf{t}_r$ has only one root node (i.e., $s = 1$) in depth d, a buffer is inserted (lines 22-24); otherwise, it is divided into two parts, and a splitter is inserted (lines 25-27). After obtaining the total cost $\mathbf{dp}_{1,n,1,0}$, an optimal splitter tree can be constructed by the backtracking method using **pt** (line 28). The time complexity and the space complexity of Algorithm 1 are $O(Xn^3\lceil \log_X n \rceil)$ and $O(Xn^2\lceil \log_X n \rceil)$, respectively.

Fig. 8 shows an example of exploring an optimal splitter tree for the $(3+1)$-pin net, which is marked by the red color in Fig. 7(a). First, the delay of each leaf node is calculated, and then all leaf nodes are sorted, as shown in Fig. 8(a). Fig. 8(b) show the generated splitter tree and its final cost $\mathbf{dp}_{1,3,1,0}$. That is, this splitter tree consists of only two splitters and has no extra delay. When we start to construct this tree by the backtracking method, $\mathbf{pt}_{1,3,1,0} = \{-1, 2\}$ indicates that the substructure marked by the red box has one root node with two fan-outs, thus turning into and visiting $\mathbf{pt}_{1,3,2,1}$. $\mathbf{pt}_{1,3,2,1} = \{1, 1\}$ indicates that the substructure marked by the red box in Fig. 8(c) needs to be divided into two parts, including one with the leaf node g_1, as shown in Fig. 8(d) and the other with the leaf nodes g_2 and g_3, as shown in Fig. 8(e). Since the substructure $\{1, 1, 1, 1\}$ in Fig. 8(d) has only one leaf node, the backtracking process can end. For the substructure $\{2, 3, 1, 1\}$ in Fig. 8(e), the above process needs to be repeated.

C. Buffer Insertion

After generating splitter trees for all over-2-pin nets, only 2-pin nets exist in the circuit. The buffers must be inserted for all 2-pin nets E' to achieve the path-balancing constraint in Equation (2). Since each net e has only two pins, the number of



I2
g1
g2
g3
Delay 1 2 2
(a)
S
$\mathbf{dp}_{1,3,1,0}$={0,0,2}
$\mathbf{pt}_{1,3,1,0}$={-1,2}
(b)
$\mathbf{dp}_{1,3,2,1}$={0,0,1}
$\mathbf{pt}_{1,3,2,1}$={1,1}
(c)
$\mathbf{dp}_{1,1,1,1}$={0,0,0}
$\mathbf{pt}_{1,1,1,1}$={-1,-1}
(d)
$\mathbf{dp}_{2,3,1,1}$={0,0,1}
$\mathbf{pt}_{2,3,1,1}$={-1,2}
(e)

Fig. 8: The example of exploring substructures. (a) shows a $(3+1)$-pin net from Fig. 7(a), and the red number at the bottom of each leaf represents its delay. (b) – (e) show different substructures and their corresponding costs.

its required buffers is equal to its delay, i.e., the logic level gap between its source e_s and its sink e_t. However, the extra delay may exist after the splitter tree generation, thereby causing a logic level change for certain nodes. To address this problem, the ILP model proposed in Section IV-A can be simplified as the following to re-calculate the final logic level of each node.

$$\min \quad \sum_{e \in E'} (L(e_t) - L(e_s)), \tag{13}$$

$$\text{s.t.} \quad \text{Equations (8) to (10).} \tag{14}$$

After solving this simplified ILP model using the ILP solver, the final logic level of each node is determined. Following the insertion of the corresponding number of buffers for all nets, a legal AQFP circuit can be obtained. That is, the buffer and splitter insertion process is completed.

V. Optimizations

Although the above buffer and splitter insertion framework has achieved effective improvements in reducing the number of inserted buffers and splitters, as evidenced in Section VI, it is essential to note that this framework comes with a significant runtime overhead. Upon conducting a thorough analysis and comparison of the three phases within the proposed framework, it became evident that the bottleneck lies in the logic level assignment process described in Section IV-A. Specifically, the construction and solution of the ILP model take up a lot of time and act as the main contributors to the runtime. Therefore, to enhance the efficiency of our proposed framework, we re-optimize the design of the ILP model. Furthermore, a re-evaluation of the objective function shown in Equation (7) has led to a more comprehensive formulation to further improve the effectiveness of our proposed framework.

A. Lower Boundary Constraints

The logic level of each node is determined by the constraints imposed by its neighboring nodes, as illustrated in Equations (8) to (11). Notably, a node's logic level has not been restricted by its own characteristics. That is, it has no lower and upper boundary. Given that the boundary setting for new variables can benefit the solution of the ILP model, this section will introduce how to calculate the lower boundary of each node's logic level.

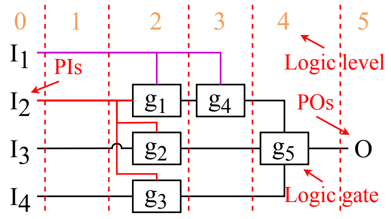


Fig. 9: The example of the lower boundary of Fig. 7(a).

As illustrated in Fig. 7(a), the minimum logic level gap between all directly connected nodes is set to 1 in the current framework. Considering the edge with multiple sinks, the insertion of at least one splitter between its source and sinks is necessary. Consequently, the logic level gap between its source and any of its sinks must be no less than 2. Based on this observation, the constraint in Equation (10) can be re-formulated as

$$\forall e \in E, \forall v \in e_t, L(v) \geq L(e_s) + 1 + (|e_t| > 1 \,?\, 1 : 0)\,. \tag{15}$$

For instance, Fig. 9 provides an update about Fig. 7(a). Since the red net has three sinks (g_1, g_2, and g_3), the logic level gap between the source I_2 and these three sinks must be at least 2. This means that the logic levels of these three sinks are at least 2. Furthermore, since the purple net has two sinks (g_1 and g_4), the logic level gap between the source I_1 and these two sinks must also be at least 2. However, a 2-pin net exists between the source g_1 and the sink g_4. The logic level gap between these two nodes must be at least 1. Consequently, the logic level of g_4 is determined to be at least 3, which is the lower boundary of g_4.

Algorithm 2 summarizes the flow of the logic level lower boundary calculation. For a given circuit $G(V, E)$, lower boundaries $\mathbf{lb}$ of all nodes' logic levels are initialized to 0 (line 2). Subsequently, a topological order V' is established from PIs to POs for all nodes (line 3). Proceeding through each node u in topological order (lines 4-10), the algorithm iterates over its respective sinks. For the sink v, when the edge between u and v has multiple fan-outs, the connection weight w between u and v is set to 2; otherwise, this weight w is set to 1 (line 6). Finally, if the sum of the lower boundary $\mathbf{lb}_u$ of node u and the weight w is greater than the current lower boundary $\mathbf{lb}_v$ of node v (line 8), an update is made to

Algorithm 2: Logic level lower boundary calculation

```
Input: A given circuit $G(V, E)$, where $n = |V|$
Output: Lower boundaries of all nodes' logic levels
1  $m = 0$
2  $\mathbf{lb} = \{0\}^n$
3  Create a topological order $V'$ from PIs to POs.
4  for $u \in V'$ do
5      for $e \in E_o(u)$ do
6          $w = 1 + (|e_t| > 1 ? 1 : 0)$
7          for $v \in e_t$ do
8              if $\mathbf{lb}_u + w > \mathbf{lb}_v$ then
9                  $\mathbf{lb}_v = \mathbf{lb}_u + w$
10                 $m = \max(m, \mathbf{lb}_v)$
11 for $o \in O$ do
12     $\mathbf{lb}_o = m$
13 return lb
```

the lower boundary $\mathbf{lb}_v$ of node v (line 9), i.e., $\mathbf{lb}_v = \mathbf{lb}_u + w$. Meanwhile, the current maximum logic level lower boundary m is recorded (line 10). Furthermore, given that all POs are aligned at the last level, their logic level lower boundaries are uniformly set to m (lines 11-12).

B. Splitter Tree Constraints

For the construction of the ILP model, the constraint addressing fan-out limitation as presented as Equation (11) is pretty complex, and its scale exhibits an exponential growth with respect to the number of fan-outs. This significantly extends the construction time of the ILP model, especially for circuits containing nets with large fan-outs. The complexity of this constraint stems from the exhaustive enumeration of all possible combinations for the sink set. Consequently, efforts are directed toward mitigating the enumeration burden by reducing the number of subsets enumerated while upholding the effectiveness of the constraint.

Given that the similarity in the delay can impact the depth of leaves in the final splitter tree, we propose a strategy where the fan-out limitation constraint works on the subsets composed of sinks with similar delays instead of all subsets. Notably, this strategy can directly benefit from the logic level lower boundary calculated in the previous subsection. Specifically, for an over-2-pin net e, its sinks e_t are first rearranged in terms of their logic level lower boundaries. Subsequently, a sliding window of size n is used to select the sink subset. This window moves on re-ordered sinks from left to right with a step size of 1, where each movement selects n sinks within the window as a subset, and then this subset will be subject to the fan-out limitation constraint detailed in Equation (11). Fig. 10 shows an example of sliding window movement. In this way, under the given window size n, only $|e_t| - n + 1$ subsets are chosen, and the number of constraints for each subset is also constant, meaning the constraint complexity is reduced from exponential to linear. The selection of the window size involves a trade-off between effectiveness and efficiency. That is because a larger

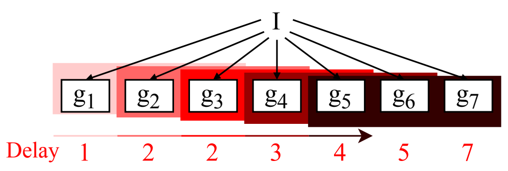


Fig. 10: A sliding window of size 3 is applied to an ordered $(7+1)$-pin net for the fan-out limitation constraint construction, where this window can move four times from left to right, and each movement contains three sinks.

window size can accommodate more sinks, thereby enhancing the constraints between sinks, while the constraint complexity of the subset chosen by the window grows exponentially with the window size.

C. Objective Function Optimization

Upon the implementation of the two optimization strategies previously proposed, a notable acceleration in the construction and solution of the proposed ILP model can be observed, up to 417 times evidenced in TABLE II. In addition, an extra novel strategy is introduced to further decrease the number of inserted buffers and splitters. Within the original ILP model, the objective in Equation (6) aims to estimate the total number of inserted buffers and splitters. After the simplification, the final objective in Equation (7) actually transforms into the sum of the average delays of all nets. This streamlined formulation, while straightforward, has demonstrated superior outcomes compared to the methods from ICCAD'21 [9] and DAC'22 [10], with an average reduction of 12.61% and 4.8% on the number of inserted buffers and splitters, as evidenced in the "`ASPDAC'23 [13]`" part of TABLE II.

Nevertheless, this objective primarily concentrates on the number of inserted buffers and splitters and neglects the circuit depth. Hence, a novel objective is devised, which represents the weighted aggregation of the sum of all nets' average delays and the circuit depth, formulated as

$$\frac{\alpha}{|E|} * \sum_{e \in E} \left(\frac{1}{|e_t|} * \sum_{v \in e_t} L(v) - L(e_s) \right) + \frac{(1-\alpha)}{|O|} * \sum_{o \in O} L(o). \tag{16}$$

The weight $\alpha \in [0, 1]$ represents the contribution to the sum of all nets' average delays. Increased complexity in constructing splitter trees is observed with higher maximum fan-out numbers in a circuit. Consequently, a higher value of α is recommended in such cases, whereas a lower value of α is preferred otherwise.

VI. Experimental Results

The proposed global optimization framework for buffer and splitter insertion of AQFP logic has been implemented using the C++-17 language. The experiments were conducted on the machine with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and 256.0 GB memory running Ubuntu 22.04. The Gurobi [23] was chosen as the integer linear programming solver due to its notable performance. This paper selected two categories

TABLE II: The experimental results on the ISCAS'85 and simple arithmetic benchmarks.

Testcase	Original Circuit			ICCAD'21 [9]				DAC'22 [10]				ASPDAC'23 [13]				Ours			
	Gates	Depth	MFO	BS	JJs	Depth	T(s)	BS	JJs	Depth	T(s)	BS	JJs	Depth	T(s)	BS	JJs	Depth	T(s)
adder1	7	4	2	16	74	8	0.15	16	74	8	0.01	16	74	8	0.02	16	74	8	0.03
adder8	77	17	3	404	1270	33	0.08	371	1204	33	0.01	371	1204	33	0.03	371	1204	33	0.03
alu32	1513	100	128	15244	39566	171	17.39	14742	38562	173	0.48	13976	37030	169	1792.05	13991	37060	169	2.16
multiplier8	439	35	9	1907	6448	70	0.21	1861	6356	71	0.04	1681	5996	70	0.18	1681	5996	70	0.16
counter16	29	9	4	99	372	17	0.05	65	304	17	0.01	66	306	17	0.02	65	304	17	0.02
counter32	82	13	4	201	894	23	0.04	155	802	23	0.01	156	804	23	0.03	155	802	23	0.03
counter64	195	17	4	412	1994	30	0.05	352	1874	30	0.02	351	1872	30	0.04	349	1868	30	0.04
counter128	428	22	4	836	4240	38	0.10	760	4088	38	0.04	755	4078	38	0.05	751	4070	38	0.05
c17	6	3	2	17	70	5	0.05	12	60	5	0.01	12	60	5	0.02	12	60	5	0.02
c432	121	26	10	889	2504	37	0.07	886	2498	39	0.01	829	2384	37	0.16	843	2412	37	0.04
c499	387	18	8	1238	4798	29	0.11	1270	4862	31	0.03	1173	4668	29	0.07	1173	4668	29	0.06
c880	306	27	9	1702	5240	40	0.11	1665	5166	41	0.03	1536	4908	40	0.07	1536	4908	40	0.06
c1355	389	18	9	1240	4814	29	0.11	1271	4876	31	0.03	1186	4706	29	0.07	1186	4706	29	0.06
c1908	289	21	14	1405	4544	35	0.12	1344	4422	37	0.03	1253	4240	34	0.13	1246	4226	34	0.06
c2670	368	21	32	2014	6240	28	0.24	2092	6392	30	0.04	1869	5954	28	816.85	1876	5964	28	0.68
c3540	794	32	38	3419	11606	53	0.63	2281	9326	56	0.18	1961	8690	52	225.85	1964	8696	52	0.60
c5315	1302	26	41	6016	19922	40	1.08	5989	19790	42	0.46	5485	18860	40	869.07	5531	18952	40	0.64
c6288	1870	89	17	9476	30172	179	2.62	9016	29252	180	0.64	8832	28884	179	275.80	8832	28884	179	3.41
c7552	1394	33	170	8090	24560	58	15.81	8598	25560	66	0.43	6756	21892	58	9421.72	6740	21860	58	1.93
sorter32	480	15	2	510	3900	30	0.12	480	3840	30	0.04	480	3840	30	0.05	480	3840	30	0.05
sorter48	880	20	3	915	7110	35	0.17	880	7040	35	0.11	896	7072	35	0.09	880	7040	35	0.07
Ave. ratio				1.17	1.08	1.00	2.51	1.06	1.03	1.03	**0.48**	1.00	1.00	1.00	417.01	**1**	**1**	**1**	1

for the benchmark circuits. The first category includes smaller circuits from the ISCAS'85 and simple arithmetic benchmark [11], while the second category comprises larger circuits from the EPFL benchmark [12]. Furthermore, all benchmark circuits used in this paper are composed of buffers with 2 JJs, splitters with 2 JJs, 3-input majority gates with 6 JJs, 2-input AND with 6 JJs, and 2-input OR with 6 JJs. For consistency in comparison with the baselines, the maximum fan-out of the splitter is set to 4. Additionally, the size of the sliding window is set to 8.

To ensure more precise experimental results, several modifications were applied to the benchmark circuits. Given the interchangeability of buffers and inverters in AQFP logic, the inverter connected to the output port of logic gates can be transferred to their input ports. For instance, $\overline{M(x,y,z)} = M(\overline{x},\overline{y},\overline{z})$ for an MAJ gate, $\overline{x \,\&\, y} = \overline{x} \,|\, \overline{y}$ for an AND gate, and $\overline{x \,|\, y} = \overline{x} \,\&\, \overline{y}$ for an OR gate. Moreover, if a PI is directly connected to a PO through an inverter, this inverter can be retained. The number of logic gates (Gates), original circuit depth (Depth), and the maximum fan-out (MFO) for all benchmark circuits are summarized in the "`Original Circuit`" part of TABLEs II and III.

This paper adopted the buffer and splitter insertion methods from ICCAD'21 [9], DAC'22 [10], and ASPDAC'23 [13] as the baselines, where the method from ASPDAC'23 is our proposed framework without optimizations. The evaluation metrics considered the number of inserted buffers and splitters, as well as the circuit depth. Additionally, to ensure the consistency of all experimental methods concerning path-balancing and fan-out handling for all PIs and POs of the circuit, the buffers and splitters are inserted into the PIs and POs of the circuit such that all PIs have a logic level of 0, all interconnecting nets have a fan-out of 1, and all POs maintain the same logic level.

TABLE II shows the experimental results on the ISCAS'85 and simple arithmetic benchmarks [11]. "BS" denotes the number of inserted buffers and splitters; "JJs" denotes the total number of JJs in the circuit; "Depth" denotes the circuit depth; while "T(s)" denotes the runtime in seconds. Comparative analysis with the method proposed in ICCAD'21 reveals that our proposed framework achieves an average reduction of 12.71% and 6.94% in the number of inserted buffers and splitters and the total JJs, respectively, while exhibiting a 36.07% decrease in runtime. When compared with the heuristic method from DAC'22, our proposed framework achieves an average reduction of 5.07% in the number of inserted buffers and splitters and 3.05% in the total JJs, with a comparable runtime. As for the method from ASPDAC'23, our framework demonstrates a speed improvement of 417.01 times compared to it, while reducing the number of inserted buffers and splitters by 0.13% on average. This finding further highlights the runtime drawback of the ILP-based method before optimization. Moreover, our proposed framework consistently yields circuit designs with smaller or equal circuit depths compared with the baseline methods. Notably, our proposed framework exhibits notable advantages over the baseline methods, particularly in circuits with substantial fan-outs. For instance, in the case of the `c7552` circuit with a maximum fan-out of 170, our framework realizes significant improvements in the number of inserted buffers and splitters compared with ICCAD'21 and DAC'22, achieving reductions of up to 16.69% and 21.61%, respectively.

TABLE III: The experimental results on larger-scale circuits from the EPFL benchmark.

Testcase	Original Circuit			ICCAD'21 [9]			DAC'22 [10]			Ours		
	Gates	Depth	MFO	BS	JJs	Depth	BS	JJs	Depth	BS	JJs	Depth
div	57247	4372	370	3660404	7664410	8618	3117173	6577828	8703	2785942	5915486	8601
hyp	214335	24801	255		**N/A**		13912571	29111152	41464	12642718	26571446	41292
log2	32060	444	253	268569	729498	779	188374	569108	788	158122	508604	773
multiplier128	27062	274	145	315282	792936	529	134413	431198	531	121620	405612	526
sin	5416	225	84	34185	100866	354	29140	90776	360	25999	84494	352
sqrt	24618	5058	126	1761312	3670332	8224	1453718	3055144	8288	1416653	2981014	8199
square	18484	250	75	125312	361528	412	102638	316180	414	97268	305440	409
Ave. ratio				1.58	1.38	1.01	1.10	1.07	1.01	**1**	**1**	**1**

* **N/A** means that the method exceeds the memory limit when running on the configured machine.
* **Ave. ratio** does not include the **N/A** items.

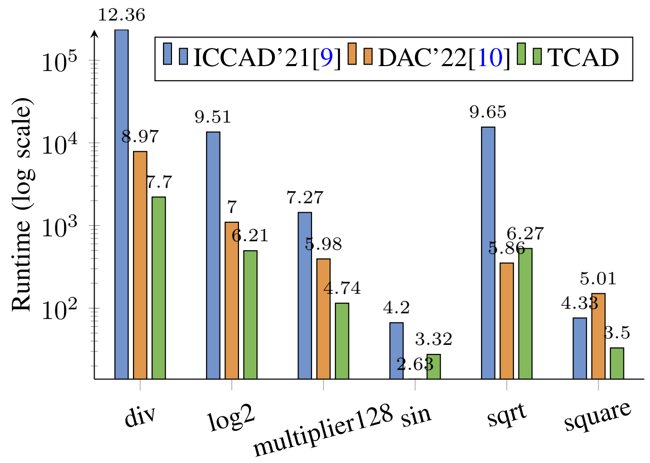


Fig. 11: The runtime comparison on the larger-scale circuits from the EPFL benchmark in seconds, with the ordinate recording the log scale of the runtime.

In order to demonstrate the scalability of our proposed framework, large circuits sourced from the EPFL benchmark are employed for evaluation purposes, whose results are shown in TABLE III. It is noteworthy that our framework attains significant effectiveness while maintaining high efficiency. Specifically, our proposed framework achieves an average reduction of 32.06% and 9.13% in the number of inserted buffers and splitters, simultaneously with smaller circuit depths, compared with ICCAD'21 and DAC'22, respectively. Moreover, our proposed framework also has 29.98 and 2.52 times the speed of these two baselines, respectively, as shown in Fig. 11. Additionally, the method presented in ASPDAC'23 exceeds the memory limit of the configured machine, thereby preventing any feasible solution for these benchmark circuits under the configured machine. This finding further highlights the memory drawback of the ILP-based method before optimization. In the case of the `hyp` circuit, the method from ICCAD'22 also fails to provide a solution on the configured machine, while our proposed framework can quickly generate a high-quality solution. Compared with the method from DAC'22, our solution requires 9.13% fewer buffers and splitters and only costs about one-third of its runtime.

VII. Conclusion

This paper introduced the buffer and splitter insertion problem of AQFP circuits. It first illustrated the importance of buffer and splitter insertion for AQFP circuit design in meeting the path-balancing requirement and fan-out limitation. Then, a global optimization framework was proposed to minimize the number of inserted buffers and splitters while legalizing AQFP circuits. This framework divided the buffer and splitter insertion process into three phases, including logic level assignment, splitter tree generation, and buffer insertion. Firstly, an integer linear programming model was built to assign the logic level of each logic gate so that the delay of all nets can be determined. Then, an optimal splitter tree could be generated for each net using the optimal multi-way search tree method based on dynamic programming. Finally, the logic level of all logic gates was updated, and buffers were inserted into all nets to achieve path balance. Furthermore, three optimization strategies were introduced to enhance the proposed framework's effectiveness and efficiency. Experimental results on benchmark circuits demonstrated the effectiveness and efficiency of the proposed framework, outperforming state-of-the-art methods in terms of the number of inserted buffers and splitters, circuit depth, and runtime, particularly on large circuits. Specifically, on large circuits from the EPFL benchmark, the proposed method reduced the number of inserted buffers and splitters by an average of 32.06% and 9.13%, and the runtime by an average of 83.20% and 27.04%, respectively, compared with the methods in ICCAD'21 and DAC'22.

Furthermore, from the perspective of the whole design flow for AQFP circuits, the insertion of buffers is imperative not solely during the logic synthesis phase but also throughout the physical design phase [24], [25]. Additionally, buffers inserted in both stages are to meet the delay requirement of the AQFP circuit design. Therefore, the integration of these two stages may be beneficial in minimizing the number of inserted buffers, the circuit latency, and even the energy consumption. In addition, the derivative technologies of AQFP logic, especially reversible quantum-flux-parametron (RQFP) [26], have also attracted much attention. RQFP is implemented by three AQFP splitters and AQFP majority gates. Hence, buffer and splitter insertion is also required in RQFP logic circuit design [27], [28] to meet its design constraints.

References

[1] D. S. Holmes, A. L. Ripple, and M. A. Manheimer, “Energy-efficient superconducting computing—power budgets and requirements,” *IEEE Transactions on Applied Superconductivity (TASC)*, vol. 23, no. 3, pp. 1 701 610–1 701 610, 2013.

[2] 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, 2013.

[3] K. Loe and E. Goto, “Analysis of flux input and output josephson pair device,” *IEEE Transactions on Magnetics (TMAG)*, vol. 21, no. 2, pp. 884–887, 1985.

[4] M. Hosoya, W. Hioe, J. Casas, R. Kamikawai, Y. Harada, Y. Wada, H. Nakane, R. Suda, and E. Goto, “Quantum flux parametron: a single quantum flux device for josephson supercomputer,” *IEEE Transactions on Applied Superconductivity (TASC)*, vol. 1, no. 2, pp. 77–89, 1991.

[5] N. Takeuchi, T. Yamae, C. L. Ayala, H. Suzuki, and N. Yoshikawa, “An adiabatic superconductor 8-bit adder with $24k_bt$ energy dissipation per junction,” *Applied Physics Letters*, vol. 114, no. 4, p. 042602, 2019.

[6] C. L. Ayala, R. Saito, T. Tanaka, O. Chen, N. Takeuchi, Y. He, and N. Yoshikawa, “A semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors,” *Superconductor Science and Technology*, vol. 33, no. 5, p. 054006, 2020.

[7] R. Cai, O. Chen, A. Ren, N. Liu, N. Yoshikawa, and Y. Wang, “A buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits,” in *IEEE International Conference on Computer Design (ICCD)*, 2019, pp. 429–436.

[8] S.-Y. Lee, H. Riener, and G. De Micheli, “Irredundant buffer and splitter insertion and scheduling-based optimization for AQFP circuits,” 2021.

[9] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021, pp. 1–8.

[10] S.-Y. Lee, H. Riener, and G. De Micheli, “Beyond local optimality of buffer and splitter insertion for AQFP circuits,” in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 445–450.

[11] ISCAS’85 and simple arithmetic benchmarks. [Online]. Available: https://github.com/lsils/SCE-benchmarks/tree/main/ISCAS

[12] L. Amarù, P.-E. Gaillardon, and G. De Micheli, “The EPFL combinational benchmark suite,” in *IEEE/ACM International Workshop on Logic Synthesis (IWLS)*, 2015.

[13] 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 *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2023, pp. 769–774.

[14] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, “Energy efficiency of adiabatic superconductor logic,” *Superconductor Science and Technology*, vol. 28, no. 1, p. 015003, 2014.

[15] N. Takeuchi, S. Nagasawa, F. China, T. Ando, M. Hidaka, Y. Yamanashi, and N. Yoshikawa, “Adiabatic quantum-flux-parametron cell library designed using a 10 kA cm^{-2} niobium fabrication process,” *Superconductor Science and Technology*, vol. 30, no. 3, p. 035002, 2017.

[16] Q. Xu, C. L. Ayala, N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, “HDL-based modeling approach for digital simulation of adiabatic quantum flux parametron logic,” *IEEE Transactions on Applied Superconductivity (TASC)*, vol. 26, no. 8, pp. 1–5, 2016.

[17] Q. Xu, C. L. Ayala, N. Takeuchi, Y. Murai, Y. Yamanashi, and N. Yoshikawa, “Synthesis flow for cell-based adiabatic quantum-flux-parametron structural circuit generation with HDL back-end verification,” *IEEE Transactions on Applied Superconductivity (TASC)*, vol. 27, no. 4, pp. 1–5, 2017.

[18] C. Wolf, “Yosys open synthesis suite,” https://yosyshq.net/yosys/.

[19] E. Testa, S.-Y. Lee, H. Riener, and G. De Micheli, “Algebraic and boolean optimization methods for AQFP superconducting circuits,” in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2021, pp. 779–785.

[20] G. Meuli, V. Possani, R. Singh, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage, P. Vuillod, L. Amaru, S. Chase, J. Kawa, and G. De Micheli, “Majority-based design flow for AQFP superconducting family,” in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2022, pp. 34–39.

[21] R. Fu, J. Huang, M. Wang, Y. Nobuyuki, B. Yu, T.-Y. Ho, and O. Chen, “BOMIG: A majority logic synthesis framework for aqfp logic,” in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2023, pp. 1–2.

[22] L. Gotlieb, “Optimal multi-way search trees,” *SIAM Journal on Computing (SICOMP)*, vol. 10, no. 3, pp. 422–433, 1981.

[23] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2024. [Online]. Available: https://www.gurobi.com

[24] Y.-C. Chang, H. Li, O. Chenht, Y. Wang, N. Yoshikawa, and T.-Y. Ho, “ASAP: an analytical strategy for AQFP placement,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

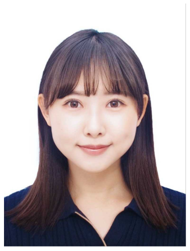
[25] R. Fu, O. Chen, B. Yu, N. Yoshikawa, and T.-Y. Ho, “DLPlace: A delay-line clocking-based placement framework for AQFP circuits,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–8.

[26] N. Takeuchi, Y. Yamanashi, and N. Yoshikawa, “Reversible logic gate using adiabatic superconducting devices,” *Scientific Reports*, vol. 4, p. 6354, 2014.

[27] R. Fu, O. Chen, N. Yoshikawa, and T.-Y. Ho, “Exact logic synthesis for reversible quantum-flux-parametron logic,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–9.

[28] R. Fu, R. Wille, and T.-Y. Ho, “RCGP: An automatic synthesis framework for reversible quantum-flux-parametron logic circuits based on efficient cartesian genetic programming,” in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

Rongliang Fu received his BS degree in software engineering from the Northwestern Polytechnical University, Xi’an, China, in 2018 and his MS degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently studying for his Ph.D degree in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation and computer architecture.



Mengmeng Wang received her BS degree in communication engineering from Chongqing University, Chongqing, China, in 2016 and her MS degree in electrical and computer engineering from Yokohama National University, Yokohama, Japan, in 2024. Her research interests include superconducting electronics and approximate computing.

Yirong Kan received the B.E. degree from the Chengdu University of Technology, Chengdu, China, in 2015, the M.E. degree from Yunnan University, Kunming, China, in 2019, and the Ph.D. degree from the Nara Institute of Science and Technology, Ikoma, Japan, in 2022. He has been an Assistant Professor with the Division of Information Science, Nara Institute of Science and Technology since 2022. His current research interests include reconfigurable computing, computer architecture, digital IC design, and emerging computing technologies.

Olivia Chen is an associate professor in the Department of Advanced Information Technology at Kyushu University, Japan, where she is also a distinguished researcher within the JST PRESTO and FOREST program. She earned her PhD degree in ECE from Yokohama National University, Japan, in 2017. Her research is focused on cutting-edge topics, including superconducting electronics, energy-efficient computing, approximate computing, deep learning hardware accelerators, and design automation for superconducting VLSI implementation. Her research findings have been published in major applied superconductivity journals, such as IEEE TAS and SUST, and presented at top tier Architecture/EDA conferences including ISCA, MICRO, DAC, DATE, ICCD, and ICCAD.

Nobuyuki Yoshikawa (F'24) is a professor at the Institute of Advanced Sciences (IAS) at Yokohama National University (YNU), where he leads the superconductivity electronics group. He earned his Ph.D. in Electrical and Computer Engineering (ECE) from YNU in 1989 and has since been affiliated with YNU's ECE Department. His research primarily centers on superconductive devices and their integration into digital and analog circuits. Presently, his focus is on developing highly energy-efficient superconducting digital circuits, particularly those that employ Adiabatic Quantum-Flux Parametron (AQFP) and Single Flux Quantum (SFQ) logic, with an aim towards high-performance computing applications. In 2023, he was honored with the IEEE Council on Superconductivity (CSC) Award for Continuing and Significant Contributions in the field of Applied Superconductivity. He is also a Fellow of the IEEE.

Bei Yu (M'15-SM'22) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Associate Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 & 2013, ASPDAC 2021 & 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and many other awards, including DAC Under-40 Innovator Award (2024), IEEE CEDA Ernest S. Kuh Early Career Award (2022), and Hong Kong RGC Research Fellowship Scheme (RFS) Award (2024).

Tsung-Yi Ho (F'24) is a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.