MapBuf: Simultaneous Technology Mapping and Buffer Insertion for HLS Performance Optimization

Hanyu Wang, Carmine Rizzi, and Lana Josipović
ETH Zurich, Department of Information Technology and Electrical Engineering, Zurich, Switzerland

***Abstract*—Buffer placement (i.e., pipelining) for frequency regulation is a fundamental step of High-Level Synthesis (HLS). Typical HLS approaches place buffers *before* technology mapping; as the circuit implementation details are unknown, the HLS tool must resort to pre-characterized and conservative delay estimates when deciding on the buffer placement. An alternative is to place buffers *after* technology mapping when the circuit details are known. However, the buffers themselves may invalidate prior mapping assumptions and irreversibly impact the ultimate circuit frequency. In this work, we propose a methodology that simultaneously tackles technology mapping and buffer insertion in HLS-produced dataflow circuits. Our approach achieves a 13.32% and 11.14% average improvement in execution time and area compared to state-of-the-art approaches that handle these problems separately.**

I. Introduction

Pipelining is usually performed during high-level synthesis (HLS), prior to technology mapping: the delay of each computational block is characterized in isolation (e.g., by synthesizing, placing, and routing the standalone component) and registers (also called *buffers*) are placed accordingly [1]–[4]. Unfortunately, this approach fails to recognize the component simplifications and gate-level interactions that will occur during technology mapping: the observed delays are overblown and cause redundant buffer insertion that later optimizations (e.g., retiming [5]) can no longer remove. A dual approach is also possible: technology mapping could be performed first and its result used to determine a less conservative buffer placement [6]. Of course, the circuit needs to be re-mapped afterwards and there is no guarantee that the previous mapping assumptions hold; the ultimate circuit structure can still deviate from the assumed one and its buffer placement may be inadequate.

To overcome these limitations, we present MapBuf, a strategy for *simultaneous* buffer placement and technology mapping based on mixed-integer linear programming. It accounts for the circuit's current mapping as well as possible mapping modifications caused by buffers; it can thus accurately control the circuit's critical path without degrading its implementation. Although our insights apply to any HLS approach, we here focus on optimizing *synchronous dataflow circuits* obtained from C code, as they are known to suffer from poor frequency regulation [2] and can thus significantly benefit from our strategy. On a set of benchmarks obtained from C code, we show that our proposed strategy outperforms a state-of-the-art method [6], achieving on average a 13.32% speedup on benchmark workloads while using 11.14% fewer flip-flops (FFs).

In the rest of the paper, we show an example in Section II to motivate our work. Then, we present background and related work in Section III, and we illustrate our methodology in Section IV and Section V. In Section VI, we depict our entire workflow and evaluate our method.

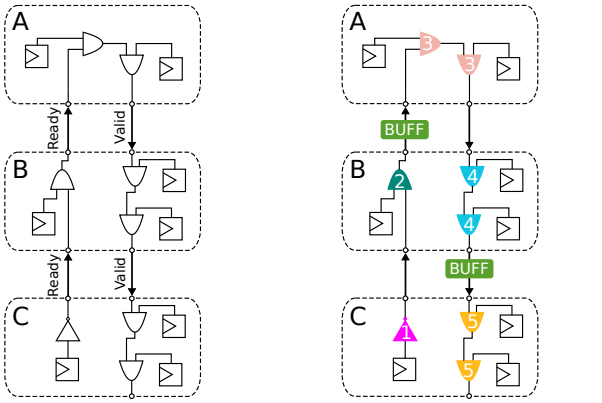
II. Simultaneous Is the Way!

Figure 1a shows a portion of a dataflow circuit with three dataflow units, A, B, and C. The units exchange data via handshake channels, *ready* and *valid*; for simplicity, the data signals are omitted from the figure. To regulate the circuit's critical path, an HLS tool can place buffers on the channels between the units; we here contrast three possible methods to minimize the number of buffers using different timing models. In these models, a look-up table (LUT) represents a unitary delay; the gates forming a single LUT are shown in the same color and annotated with the same LUT ID. In all these cases, the target clock period (CP) corresponds to a combinational delay of two LUTs.

Figure 1b shows an example of performing buffering prior to technology mapping (we refer to this strategy as B-M). Since the delay of each unit is characterized in isolation, the strategy cannot consider any cross-unit optimizations; each unit is assumed to consist of independent LUTs, as shown in the timing model in the figure. To honor the target CP, the HLS tool must place two buffers, ensuring that each combinational path has the length of at most two LUTs.

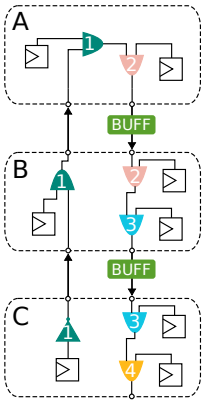
Figure 1c shows the result of performing technology mapping prior to buffering (we refer to this strategy as M-B). Notice that, in this case, a single LUT contains logic from multiple units (e.g., $\texttt{LUT}_1$ contains gates of units A, B, and C). Since buffers can be placed only between the dataflow units, to honor the CP constraint of two LUTs, the HLS tool must place a buffer between A and B (i.e., path $\texttt{LUT}_1$ - $\texttt{LUT}_2$ - $\texttt{LUT}_3$ must be broken into two stages). However, since the HLS tool has no knowledge about the impact of this buffer on the final circuit structure, it must conservatively assume that $\texttt{LUT}_2$ is replicated into two LUTs. For this reason, the lower path of $\texttt{LUT}_2$ - $\texttt{LUT}_3$ - $\texttt{LUT}_4$ will remain in the timing model; this calls for an additional buffer between units B and C.

Figure 1d shows a circuit obtained via simultaneous buffer placement and technology mapping. Unlike the previous case, the HLS tool can now account for the mapping change caused by the placement of a buffer between A and B; concretely, it is

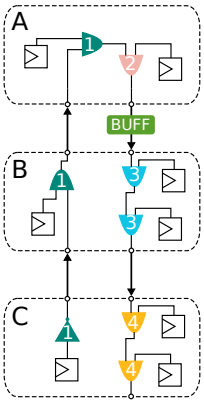


(1a) Dataflow circuit

(1b) Buffering before mapping ("*B-M*")



(1c) Mapping before buffering ("*M-B*")



(1d) Simultaneous mapping and buffering (ours, i.e., MapBuf)

Fig. 1: Comparison of different buffer placement techniques for dataflow circuits. In all figures, gates forming a single LUT are shown in the same color and annotated with the same LUT ID. Both Figure 1b and Figure 1c separate technology mapping and buffer insertion: the former executes buffer placement before technology mapping, and the latter applies them in the reversed order. On the other hand, our strategy (Figure 1d) considers both problems simultaneously to find the solution with the best performance and lowest number of buffers.

aware that the gates of unit B now map to a new $\texttt{LUT}_3$ and the gates of unit C to a new $\texttt{LUT}_4$. A single buffer is sufficient to honor the CP target; the resulting circuit is smaller (i.e., fewer buffers) and, possibly, achieves better throughput (if a buffer is omitted from a throughput-determining path) than the solutions of Figure 1b and Figure 1c.

This example points to the need to perform buffer placement and technology mapping simultaneously to obtain smaller and faster circuits. The rest of this paper illustrates our methodology to perform this task to achieve high-throughput and low-area dataflow circuits obtained from C code and targeting an Field-Programmable Gate Array (FPGA) implementation.

III. Background

The following section provides a background on dataflow circuits. It explains the concepts of buffer insertion and technology mapping and discusses the relationship between these circuit design aspects.

A. Dataflow Circuit and Buffer Insertion

A *dataflow circuit* (also called *elastic circuit*) is a digital circuit that uses a handshake communication protocol [7], [8]. Researchers have recently studied the application of dataflow circuits in HLS [9] since dataflow circuits can adjust their scheduling dynamically. Meanwhile, traditional HLS tools generate a finite-state machine (FSM) to determine the scheduling of the circuits statically at compilation time [10]. Due to these characteristics, dataflow circuits outperform state-of-the-art static HLS circuits when the input circuit contains irregular memory accesses or irregular control flow.

These circuits are composed of *dataflow units*, which communicate through *channels*. A *buffer* is the equivalent of a standard synchronous register: it can be inserted between dataflow units and is used to break combinational paths (and, thus, controls the operating frequency). It uses the same communication protocol to interact with adjacent units and thus can be placed on any channel without affecting circuit functionality. This is fundamentally different from traditional pipelines, where it is necessary to balance registers insertion [10].

Buffer insertion is one of the key optimization steps in dataflow circuit synthesis flow, as it defines the circuit's operating frequency and throughput, thus determining its overall execution time. Previous works have extensively studied this problem and formulated timing models through Mixed-Integer Linear Programming (MILP) [1], [2]. This formulation describes buffer insertion to regulate frequency while simultaneously selecting the number of buffer slots (i.e., the number of data items that a buffer can hold) to maximize throughput, as we show in Section IV-E.

However, these works [1], [2] would pre-characterize the delays of each dataflow unit and integrate this information into the timing model. As shown in Figure 1b, this technique does not account for the output circuit generated by technology mapping achieving sub-optimal results.

B. Subject Graph and Technology Mapping

A *subject graph* is a directed acyclic graph (DAG) composed of abstract logic operations (not actual gates) [11]. The nodes of the subject graph with no incoming and no out-coming edge are called *primary inputs* (PIs) and *primary outputs* (POs), respectively. *Combinational inputs* (CIs) are the union of PIs and register outputs; *combinational outputs* (COs) are the union of POs and register inputs. Commonly used subject graphs in the literature include AND-Inverter Graphs (AIG) [12] and XOR-AND-Inverter Graphs (XAG) [13].

Technology mapping is an essential step of any CAD flow that maps the subject graph to a technology-dependent network

composed of macro-cells. Macro-cells in FPGAs include *look-up tables* (LUTs), *carry chains* and *digital signal processors* (DSPs). Most technology mapping algorithms utilize *cuts* to map subject graph nodes to macro-cells [11], [14], [15]. A cut C of node n is a set of nodes (called *leaves*) such that every path from any combinational input to n traverses at least one leaf of C. A cut is K–feasible if the number of leaves does not exceed K. In FPGA mapping, a K–feasible cut implies a K–input LUT.

C. Relationship between Technology Mapping and Frequency Regulation

Previous works have analyzed the relationship between technology mapping and frequency regulation [6], [16], [17]. Rizzi et al. [6] have proposed an iterative approach for mapping-aware buffer insertion in dataflow circuits. As already shown in Figure 1c, it is essential to consider the effect of buffer insertion on technology mapping to avoid sub-optimal buffer placement. This methodology attempts to mitigate this effect by executing multiple iterations. However, the buffer placed in each iteration cannot guarantee a minimal buffer placement. Tan et al. [17] and Pan et al. [16] propose methods that simultaneously execute technology mapping and frequency regulation (the former by inserting registers, the latter through retiming them). However, these methods only consider latency optimization and ignore throughput optimization, which is the most common optimization in HLS [3]. Indeed, their algorithms cannot size the buffer slots for throughput improvement. Moreover, they cannot be directly applied to our scenario because, in our problem, buffers can be placed only between dataflow units [2] and not on any edge of the subject graph.

IV. MILP Formulation for Performance Optimization

We demonstrated in Section II the necessity of tackling technology mapping and buffer insertion problems concurrently. This section illustrates how MapBuf defines these steps in the same linear programming system for critical path regulation and throughput optimization.

Let $G = (V_G, E_G)$ be the initial dataflow graph before buffer insertion, where V_G is the set of dataflow units and E_G is the set of channels. Let $H = (V_H, E_H)$ be the subject graph corresponding to G for technology mapping, where V_H is the set of technology-independent nodes and E_H is the set of edges. Given G and H, MapBuf utilizes the variables shown in Table I to formulate the problem. The domains of our variables include integer, real number, and binary numbers, which make the problem a mixed integer linear programming problem.

An example of a dataflow graph is shown in Figure 2a, and its corresponding subject graph is shown in Figure 2b. The dataflow graph G has two dataflow units: $V_G = \{A, B\}$, and two channels: $E_G = \{Ready, Valid\}$. In this example, we use XAG as the subject graph and map gates a, b, c, and d directly to nodes in the subject graph. The square boxes are registers,

Input parameters		
D_{lut}	$\mathbb{R}_+$	Delay of a LUT
CP	$\mathbb{R}_+$	Target clock period
Output variables		
R_c	$\{0,1\}$	Indicates if channel $c, c \in E_G$, is buffered
N_c	$\mathbb{Z}_{\geq 0}$	Number of slots of the buffer on channel c
Internal variables		
R_e	$\{0,1\}$	Indicates if edge $e, e \in E_H$, is buffered
T_e^{in}	$\mathbb{R}_{\geq 0}$	Timing var. for edge e input, $e \in E_H$
T_e^{out}	$\mathbb{R}_{\geq 0}$	Timing var. for edge e output, $e \in E_H$
S_n^{γ}	$\{0,1\}$	Indicates if cut γ of node n is selected

TABLE I: Variable declaration for the MILP formulation in Section IV.

the outputs of n_1 to n_5 are the CIs, and inputs of n_6 and n_7 are the COs, as defined in Section III.

We allocate *timing variables* to the endpoints of edges in the subject graph to depict the delay at these points. An edge e in the subject graph has two timing variables, denoted by T_e^{in} and T_e^{out}. They represent the propagation delay entering and exiting the edge. For example, $T_{e_6}^{in}$ in Figure 2b is the propagation delay entering edge e_6, which is equal to the departure delay at a's output. $T_{e_6}^{out}$ is equal to the delay at the input of node c.

We use *timing constraints* to express the relationship between timing variables and their interaction between technology mapping and buffer insertion. They examine the feasibility and ensure the solution honors the target clock period. In the remaining part of this section, we explain different timing constraints that we employ in our MILP.

A. Clock Period Constraints

The *clock period constraints*, as shown in Equation (1) and Equation (2), are timing constraints that enforce the clock period target as the upper bound of all timing variables.

$$T_e^{in} \leq CP, \forall e \in E_H \tag{1}$$

$$T_e^{out} \leq CP, \forall e \in E_H \tag{2}$$

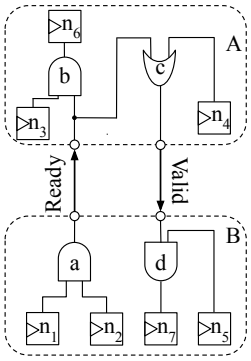
where CP is the user-specified clock period target (see Table I).

B. Buffer Insertion Variables and Channel Constraints

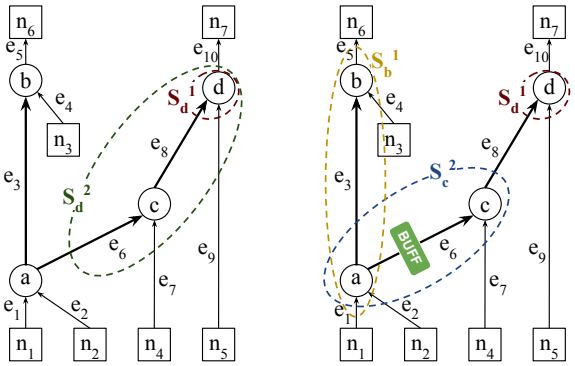
We use binary *buffer insertion variables*, denoted by R_e, to represent whether edge e is buffered. Buffer insertion variable R_e interacts with two timing variables on e as shown in Equation (3).

$$T_e^{out} - T_e^{in} + CP \cdot R_e \geq 0, \forall e \in E_H. \tag{3}$$

If $R_e = 0$, i.e., no buffer is inserted, the delay propagates from edge input to edge output with no increase. Otherwise, if $R_e = 1$, the equation becomes $T_e^{out} \geq T_e^{in} - CP$, where T_e^{in} can be at most equal to CP due to Equation (1). The largest value of the right-hand side is 0 which becomes a redundant constraint since T_e^{out} is a non-negative value.



(2a) Dataflow graph



(2b) Subject graph (2c) Selection conflict

Fig. 2: Example of simultaneous buffer insertion and 3-LUT mapping. Figure 2b shows the subject graph obtained from the dataflow graph in Figure 2a. Figure 2c presents an example of a buffer and selection variable conflict; we use it to illustrate Equation (6) in Section IV-D.

Besides channels, edges in the subject graph can correspond to internal edges inside dataflow units. This is the case for edges e_1 and e_2 in Figure 2b, which connect the register n_1 and n_2 to gate a, all within the dataflow unit B. Thus, no buffer can be inserted on these edges; therefore, we assign $R_e = 0$ if e is internal. In contrast, edges e_6 and e_8 correspond to the two channels *Ready* and *Valid*, respectively. Thus, R_{e6} and R_{e8} are free binary variables and could be either 1 or 0.

Moreover, one channel can correspond to multiple subject graph edges. For instance, channel *Ready* in Figure 2a corresponds to edges e_3 and e_6. The R_e variables of these two edges are equivalent (i.e., $R_{e_3} = R_{e_6}$). Therefore, we add these equality constraints to ensure that the edges corresponding to the same channel are either all buffered or none of them are buffered.

C. Cut Selection Variables and Delay Propagation Constraints

Before formulating the constraints, we run cut enumeration and prepare a set of cuts for each node n in the subject graph, denoted by Γ_n. Each cut γ, $\gamma \in \Gamma_n$, is a set of leaf nodes in the subject graph. We define *cut selection variables*, S_n^γ, as a set of binary variables indicating if the cut γ is selected by node n. We use *cut selection constraints*, as shown in Equation (4), to enforce the selection of one cut per node. This will allow us to propagate delays across nodes, as we will illustrate in the remainder of this section.

$$\sum_{\gamma \in \Gamma_n} S_n^\gamma = 1, \forall n \in V_H. \tag{4}$$

The delay propagation per node differs with respect to the cut selected for the node because the set of input leaves (and, consequently, input delays) changes depending on the chosen cut. For this reason, delay propagation equations are replicated per leaf for each cut of all the nodes in the subject graph. Equation (5) shows the *delay propagation constraints* considering the leaf l of the cut γ of node n of the subject graph, where ne is the output edge of node n and le is the output edge of leaf l.

$$CP \cdot (1 - \sum_{\delta \in \Delta_l} S_n^\delta) + T_{ne}^{in} \geq T_{le}^{out} + D_{lut}, \forall l \in \gamma, \forall \gamma \in \Gamma_n, \forall n \in V_H, \tag{5}$$

where D_{lut} is the delay that we assume for one LUT level, and Δ_l is the subset of cuts of node n which contain as leaf l (i.e., $\Delta_l \subseteq \Gamma_n \wedge \forall \delta \in \Delta_l \mid l \in \delta$). In the rest of this section, we use examples to illustrate Equation (5).

Consider node d in Figure 2b. It has two possible candidate cuts $\{c, n_5\}$ and $\{a, n_4, n_5\}$ whose selection variables are S_d^1 and S_d^2, respectively. In all the equations of node d, ne is edge e_{10}. Considering leaf a, le is edge e_6, and the set Δ_a is composed only by the second cut since the first one does not include a as a leaf. Equation (5) would become

$$CP \cdot (1 - S_d^2) + T_{e_{10}}^{in} \geq T_{e_6}^{out} + D_{lut}.$$

If the second cut is selected, $S_d^2 = 1$, the delay of edge e_6 is propagated until edge e_{10} with a delay increase of D_{lut}. This means that nodes covered by this cut are substituted by a LUT during technology mapping, and the delay of the output of this LUT is equal to the delay of one of its leaves (in this case a) increased by the delay of a LUT. If the second cut is not selected, the equation could be approximated to $T_{e_{10}}^{in} \geq 0$, which becomes a redundant constraint ignoring the delay propagation from edge e_6 to e_{10}.

If we consider the leaf n_5, le is edge e_9, and the set Δ_{n_5} is composed by both cuts since they contain n_5 as a leaf. In this case, the previous equation would be

$$CP \cdot \left(1 - (S_d^1 + S_d^2)\right) + T_{e_{10}}^{in} \geq T_{e_9}^{out} + D_{lut}.$$

If any of the two cuts are selected, the output delay of edge e_9 is propagated to the input of edge e_{10} increased by D_{lut}. Since these are the only cuts of node d, one of them must be selected, and Equation (4) will enforce this delay propagation.

D. Cut Selection Conflicts

As mentioned in Section II, buffer insertion also affects technology mapping. A cut (and its corresponding LUT) covers an edge in the subject graph if this edge belongs to at least one path from the root to any leaf of the cut. After placing a buffer on an edge, a LUT can no longer cover it

since LUTs cannot implement the logic of multiple sequential stages as shown in Figure 1c. Subsequently, it is not possible to select the cut that represents this LUT. For this reason, buffer insertion excludes the possibility of choosing particular cuts. We account for this effect using the conflict constraints in Equation (6). This equation is replicated for all the edges and cuts where the cut γ covers edge e.

$$R_e + S_n^\gamma \leq 1, \tag{6}$$

where R_e is the buffer insertion variable for edge e and S_n^γ is the cut selection variable for node n and cut γ.

Figure 2c shows an example of cut selection conflict. If $R_{e_6} = 1$, then $S_c^2 = 1$ is invalid. A buffer insertion variable can affect multiple cuts. For instance, if $R_{e_6} = 1$, S_b^1 cannot be selected since edges e_6 and e_3 represent the same channel in the dataflow graph and, consequently, $R_{e_6} = R_{e_3}$ as discussed in Section IV-B. Also, a cut can be affected by more than one buffer insertion variable if the cut covers multiple channels. We precompute these interactions and encode the corresponding equalities into the MILP.

E. Objective Function

Our objective function, shown in Equation (7), is similar to the one used in previous work [1].

$$\text{max.} \quad \text{throughput} - \alpha \cdot \sum_c (N_c), c \in E_G, \tag{7}$$

where N_c is the number of buffer slots on channel c of the dataflow graph and α is a user-defined parameter where $\alpha \ll 1$. N_c depends on R_c which represents the presence of a buffer on channel c. In particular, R_c variables are a subset of the R_e variables where e is the edge of a channel c. If R_e is set to 1 on a channel, then R_c is equal to 1 which means that there is at least one buffer slot on channel c (e.g., $N_c \geq 1$)

V. Specialized Cut Enumeration

The complexity of our MILP depends on the number of cuts included in the formulation. A higher number of cuts per node would determine a larger design space exploration and our MILP solver would, potentially, not be able to find the optimal solution within the pre-defined timeout. Therefore, we apply a state-of-the-art cut enumeration method with priority cuts to restrict the number of cuts [18]. Contrary to previous cut ranking and pruning which mainly focus on area recovery [19]–[21], we specialize our cut enumeration for buffer placement. We rank cuts using the following criteria and prune those with a lower ranking.

Criteria 1. MapBuf selects the cuts for delay propagation and places buffers to satisfy the clock period constraints as indicated in Equations (3) and (5). A lower propagation delay corresponds to a lower number of buffers; for this reason, it selects cuts to minimize the number of logic levels. We implement a heuristic based on cutless FPGA mapping to achieve this goal [20]: the main difference is that we select multiple top-ranking cuts per node instead of a single one.

Criteria 2. MapBuf must be able to break every channel with a buffer; that way, it can explore a wide variety of buffering solutions in the search for high-quality ones. If all the cuts of a node cover a channel c, it would be impossible to place a buffer on c due to Equation (6). To this end, for each node, we preserve all cuts with at least one leaf on any channel. For instance, in Figure 2b, we keep the cuts S_d^1 and S_d^2 for node d since leaves c and a have outputs on channels *Valid* and *Ready*, respectively. In this way, it would be possible to break channel *Ready* and set cut $S_d^2 = 1$.

VI. Evaluation

In this section, we present our entire workflow and illustrate the effectiveness of MapBuf by comparing it with two recent optimization strategies that run technology mapping and buffer insertion separately: B-M that runs buffer insertion before technology [2], as shown in Figure 1b and M-B that executes mapping before buffering [6], as illustrated in Figure 1c. MapBuf lies in the middle of the design space and tackles mapping and buffering simultaneously, as depicted in Figure 1d.

A. Workflow

Our workflow is shown in Figure 3. The inputs are a dataflow graph generated by an open-source dynamically scheduled HLS tool [9], the target FPGA architecture, and the target clock period. The output is a buffered dataflow graph. Given the dataflow circuit, we run logic synthesis and retrieve the subject graph. We then perform cut enumeration to prepare a set of cuts for each node in the subject graph using the strategy of Section V. We employ our MILP formulation from Section IV and solve it using the Gurobi solver [22] with a 40-minute timeout. We extract the buffer placement from the MILP solution to obtain the buffered dataflow circuit. We run logic synthesis and technology mapping on this circuit using ODIN-II 8.1.0 with Yosis [23] and ABC 1.01 (applying the command "`if -K 6`") [24]. We evaluate the number of clock cycles and verify the functional correctness of the circuit by running behavioral simulations in Modelsim 2021.2 [25]. We evaluate the maximum achievable clock period by parsing the post-layout setup timing report of VPR 8.1.0 [26] using a modified VTR version of the Stratix-IV architecture [26]. Finally, we read the FPGA utilization report from VPR and get the number of LUTs and FFs required in the implementation.

Notice that we do not use the LUT mapping solution (i.e., the cut selection variables) to implement the LUTs directly but run ODIN-II and ABC on the buffered dataflow graph instead. On one hand, MapBuf models only the depth of the logic network to regulate frequency. On the other hand, ABC executes area recovery heuristics which improves the number of LUTs, thereby, the implementation area. Note that this aspect is orthogonal to our optimization and only an additional optimization that our baseline strategies benefit from as well.

We use a consistent evaluation flow on the buffered dataflow graph of all three buffering methods we compare. We analyze the same HLS kernels evaluated from a recent work that explores buffer placement in dataflow circuits [6]. Additionally, we introduce a benchmark implementing a single forward path of a Convolutional Neural Network (CNN) [27] to illustrate

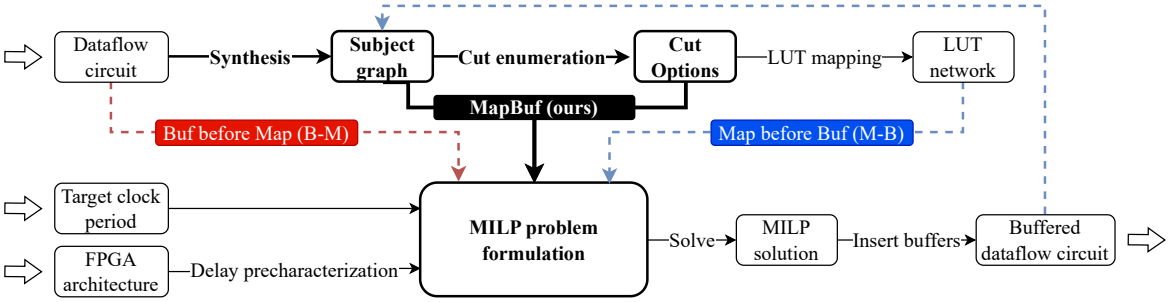


Fig. 3: MapBuf workflow. All three approaches that we consider (i.e., M-B, B-M, and MapBuf) input a dataflow circuit description, the target clock period, and FPGA architecture information, with the goal of producing a buffered dataflow circuit by solving an MILP. B-M [2] relies on precharacterized delays and entirely omits LUT mapping details (red dashed line). M-B [6] iterates between circuit mapping and the MILP (dashed blue lines). The novelty of MapBuf is in describing both buffer insertion and technology mapping simultaneously inside a single MILP formulation (highlighted with bold black lines).

the effectiveness of our approach on large workloads. We set our target clock period to $CP = 4.2$ ns consistently with our baselines [2], [6].

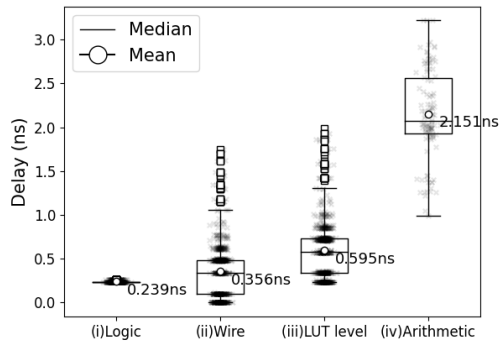


Fig. 4: Delay characterization. We show the delay value distribution measured in our flow for the following constructs: (i) inside a single LUT, (ii) on the wire between two LUTs, (iii) the effective LUT level delay (i.e., the sum of the single LUT delay and the surrounding wiring), and (iv) an arithmetic unit (including the wire delays to and from the arithmetic units). The average delay of a LUT level (D_{lut}) is 0.595 ns, consisting of an average logic delay of 0.239 ns and a wire delay of 0.356 ns; we include this average value in our timing model. For a 4.2 ns clock period target, the longest combinational path should contain at most 7 LUT levels. Besides, we employ the average arithmetic unit delay, $D_{arith} = 2.151$ ns.

B. Delay Characterization

Our model needs delays of different macro-cells; we here describe how we obtain them.

We determine the LUT level delay (D_{lut}) and carry chain delay (D_{arith}) by evaluating all benchmarks using the workflow from Section VI-A; our results are plotted in Figure 4. We incorporate the average of the measured delay values into our timing model. Other special nodes like DSPs are pipelined, and there is no combinational delay from input to output.

D_{arith} is the pre-characterized delay for *carry chains*. The edges leaving and entering the carry chain unit still respect the Equations (1), (2) and (3). However, as the carry chain units are not implemented using LUTs, we replace Equation (5) with the following equation for delay propagation.

$$T_{e2}^{in} \geq T_{e1}^{out} + D_{arith},$$

where $e1$ and $e2$ are the arithmetic unit's input and output edges.

C. Results: Performance Evaluation

Table II demonstrates the number of clock cycles, achieved clock period, logic levels, the execution time (i.e., product of clock period and cycles) achieved by the three approaches we consider (i.e., B-M, M-B, and our MapBuf). Results show that MapBuf's buffered circuit requires fewer clock cycles than B-M on **all benchmarks** since pre-characterization overestimates the propagation delay. As a result of reducing the clock cycles and shortening the clock period, we reduce the execution time by up to 43%, as depicted in the "Speedup" column.

Compared with M-B, MapBuf's mapping and LUT-level prediction are more flexible. During one iteration, M-B method assumes a fixed mapped LUT network when running buffer insertion and a static buffer configuration when running another technology mapping. Therefore, the iterative method selects a locally optimal solution. Meanwhile, MapBuf explores both buffer insertion variables and cut selection variables simultaneously. As a result, MapBuf's throughput and, thus, execution time, is higher than M-B on **all benchmarks**, resulting in the average speedup of 13.32%.

We currently assume a fixed LUT delay plus average wire delay and ignore the wire delay variability. As shown in Figure 4, the longest wire delay is around 2.0 ns, which is larger than MapBuf's wire delay assumption of 0.35 ns. This explains why, despite the fact that we accurately honor the LUT level target, the clock period results on some benchmarks, e.g., "insertion sort", are higher than the target. In

	Cycles			Clock period (ns)			LUT level			Execution time (ns)			Speedup	
	B-M	M-B	**ours**	B-M	M-B	**ours**	B-M	M-B	**ours**	B-M	M-B	**ours**	B-M	M-B
gaussian	5050	4481	**3354**	5.35	4.75	**4.58**	4	5	5	27018	21285	**15371**	43%	28%
covariance	179494	179465	**176862**	4.68	4.49	**4.29**	5	4	5	840032	805798	**758561**	10%	6%
insertion sort	232	219	**199**	5.11	**5.02**	5.13	6	6	6	1186	1099	**1021**	14%	7%
gemver	9622	8632	**6652**	5.92	**5.31**	6.02	5	4	6	56962	45836	**40052**	30%	13%
gsumif	5271	4342	**4227**	5.02	4.79	**4.18**	5	5	5	26460	20798	**18350**	31%	12%
gsum	5368	4450	**4326**	4.69	**4.02**	4.27	6	4	5	25176	19002	**18684**	26%	2%
matrix	101515	70828	**67662**	4.64	4.77	**4.40**	5	6	6	471030	337850	**297577**	37%	12%
mvt	20115	20072	**20048**	3.83	4.53	**3.72**	4	5	4	77040	90926	**74599**	3%	18%
stencil 2d	30674	28300	**23592**	4.97	**4.65**	4.98	5	5	5	152450	131595	**117441**	23%	11%
CNN	657990	626092	**440552**	5.98	5.2	**5.11**	7	5	5	3934780	3255678	**2251221**	43%	31%
Avg. impr. % B-M	-	10.49%	**20.92%**	-	4.46%	**6.53%**				-	14.36%	**25.92%**		
Avg. impr. % M-B	-	-	**11.50%**	-	-	**1.57%**				-	-	**13.32%**		

TABLE II: Performance comparison of B-M (buffering before mapping, see Figure 1b), M-B (mapping before buffering, see Figure 1c), and MapBuf (indicated as *ours*). We measure the clock cycles, clock period, and LUT levels, and calculate the execution time as the product of the clock period and cycles. For each parameter, we present the arithmetic mean of improvements compared to B-M ("Avg. impr. % B-M") and M-B ("Avg. impr. % M-B"). MapBuf systematically honors the LUT level target, *always* reduces the number of clock cycles with respect to both baselines, and typically achieves a lower clock period. Consequently, MapBuf achieves execution time speedups with respect to *both methods on all benchmarks* (see rightmost column "Speedup"), with an average speedup of 25.92% compared to B-M and 13.32% compared to M-B.

	#FFs			#LUTs		
	B-M	M-B	**ours**	B-M	M-B	**ours**
gaussian	808	801	**648**	1302	1241	**1083**
covariance	1616	**1306**	1381	2653	**2285**	2373
insertion sort	2230	1867	**1828**	3528	2903	**2799**
gemver	5129	5177	**3109**	7207	7281	**4786**
gsumif	917	820	**813**	1513	**1284**	1373
gsum	649	**514**	567	1084	**858**	971
matrix	966	891	**714**	1455	1397	**1145**
mvt	1607	1317	**1178**	2420	2117	**1902**
stencil 2d	1567	1599	**1192**	2396	2386	**1917**
CNN	2355	2462	**2226**	3996	4122	**3848**
Avg. impr. % B-M	-	8.60%	**19.79%**	-	8.50%	**16.77%**
Avg. impr. % M-B	-	-	**11.14%**	-	-	**8.11%**

TABLE III: FPGA utilization comparison. We show the LUT and FFs usage in the FPGA implementation; "Avg. impr. % B-M" and "Avg. impr. % M-B" represent the arithmetic mean of area reduction compared to B-M and M-B, respectively. In addition to the significant performance improvements shown in Table II, MapBuf generally uses fewer FFs and LUTs than prior approaches.

the future, we intend to include the wire delay variability in the timing model to further improve the accuracy of our model and, consequently, circuit performance. Yet, even with this discrepancy, MapBuf systematically outperforms both M-B and B-M, which points to the relevance of considering buffer insertion and technology mapping simultaneously.

D. Results: FPGA Utilization Evaluation

All three methods minimize the total number of buffer slots as part of their objective function to reduce the area consumption of the final circuit. The results in Table III demonstrate the effectiveness of MapBuf regarding area optimization. With the help of cut selection variables, MapBuf satisfies the target logic level while using, on average, 19.79% fewer FFs compared to the B-M method and 11.14% compared to the M-B method. Since MapBuf inserts less FFs, it breaks fewer combinational logic paths and reduces the number of LUTs consequently. In contrast, the other two methods place unnecessary buffers, as shown in Section II. For this reason, MapBuf generates circuits with 16.77% fewer LUTs compared to the B-M method and 8.11% compared to the M-B method. Together with the results in Section VI-C, we conclude that MapBuf inserts buffers less aggressively than the other two methods, but in more appropriate locations: our circuits are typically faster and cheaper than prior solutions. Hence, MapBuf achieves Pareto-optimal design points that were not possible with prior techniques.

E. Results: LUT Level Estimation Accuracy

As mentioned in the Section VI-A, after MapBuf outputs the buffered dataflow circuit, we use ABC to run technology mapping and proceed with clock period evaluation based on ABC's mapping result. The LUT level derived by MapBuf's MILP constraints may deviate from the ultimate value after ABC. We are here interested in evaluating these effects and analyze possible discrepancies. Therefore, we use a benchmark suite of combinational logic networks with different sizes [28] to evaluate MapBuf's performance. We change (only in this evaluation) the objective function to minimize the LUT level and read the objective function value after optimization. Figure 5 shows the results. MapBuf achieves almost the same results as ABC, indicating the high quality of LUT level calculation in MapBuf.

F. Results: MILP Runtime

MapBuf combines buffer insertion and technology mapping into a MILP formulation which is, naturally, more complex than approaches that handle these optimizations separately (such as the two approaches we discussed in this section). We here investigate the runtime of MapBuf and its ability to achieve near-optimal results within an acceptable time frame.

Our runtime experiment considers CNN, as it contains the most complex loop nest (i.e., a single loop encapsulating three loops with up to six levels of nesting) among our benchmarks.

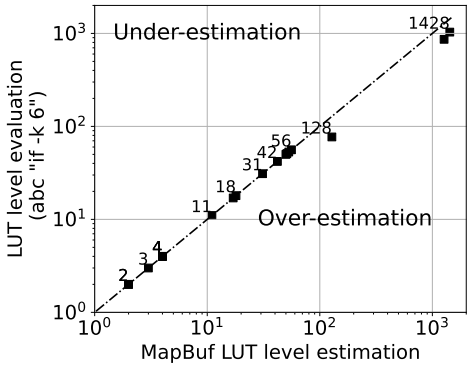


Fig. 5: Accuracy of MapBuf's LUT level estimation. x-axis is the solution of our MILP solver with 200 seconds timeout, and y-axis is the LUT level after running ABC's script "`if -K 6`". Both axes are of log scale, and the dashed line $y = x$ indicates that MapBuf achieves the same LUT level as ABC. Points above and below the diagonal line indicate an underestimation and an overestimation of LUT levels by MapBuf, respectively. MapBuf achieves the depth-optimal LUT level precisely the same as ABC's mapping results and slightly overestimates networks deeper than 128 LUT levels.

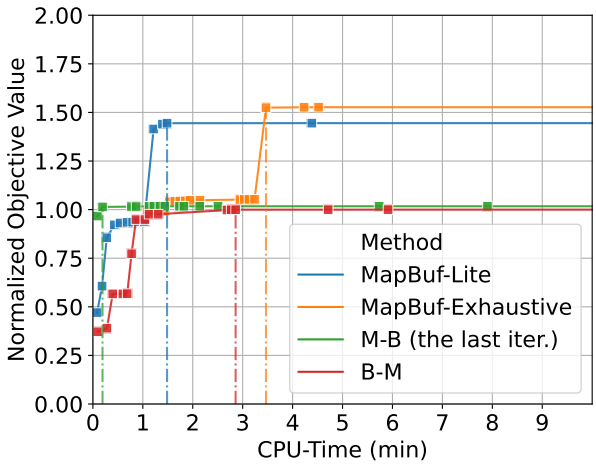


Fig. 6: MILP objective function value with respect to CPU time of benchmark "CNN". The squares represent updates of the objective value. We plot two runs of MapBuf with different numbers of precomputed cuts (MapBuf-Exhaustive which and MapBuf-Lite) and two runs of the baselines B-M and M-B. We normalize the y-axis using B-M's final objective value. The objective function values in all four runs gradually converge but at different speeds. Results show that MapBuf's CPU time is comparable to the other two methods and achieves better objective function values, which points to its scalability and ability to achieve high-quality results.

We consider four optimization techniques: B-M, M-B (in particular, the last round of the iterative method), MapBuf with an exhaustive cut enumeration that allows up to 100 cuts per node (MapBuf-Exhaustive), and MapBuf with one cut per node in addition to the cuts preserved by criteria 2 from Section V (MapBuf-Lite). All four techniques target the optimization function from Equation (7); we can, thus, directly compare the objective function value that each technique is able to achieve within a given CPU runtime.

Figure 6 plots the evolution of the objective function value of the four techniques with CPU runtime. A higher value indicates a circuit with better performance. We can identify which term is updated by the slope of the line since throughput improvements increase the objective function more significantly than reducing buffers (due to the value of α). We use dashed lines to indicate the last update of throughput and, thus, the CPU time required to acquire the final performance (without accounting for possible improvements in the area). We observe the following from the figure: (1) M-B and B-M quickly converge to the same objective function value, which does not further improve with time. (2) MapBuf-Exhaustive takes 21% longer to converge than the B-M method, but then achieves a higher objective function value than M-B and B-M; this is in line with our observations from Section VI-C, which demonstrates the superior performance of MapBuf. (3) MapBuf-Lite converges faster than MapBuf-Exhaustive and even 49% faster than B-M, with only a minor decrease in objective function value; this indicates the ability of our heuristic from Section V to effectively reduce MILP runtime without significant performance degradation. All of these point to MapBuf's ability to achieve high-performance design points that prior techniques were not able to discover, as well as its scalability and broad applicability.

VII. Conclusion

Buffer insertion for performance optimization is a critical design step of high-level synthesis (HLS); however, its effectiveness is hindered by the inability of HLS to account for the effects of technology mapping on the circuit's combinational delays. This paper proposes MapBuf, an optimization strategy for HLS-produced dataflow circuits that formulates technology mapping and buffer insertion into a joint mixed-integer linear programming (MILP) problem. By simultaneously exploring mapping solutions and buffer configurations, MapBuf can accurately insert buffers to maximize circuit throughput and regulate its frequency. In addition to the exact MILP formulation, we propose a heuristic cut ranking algorithm to specialize cut enumeration—it enables MapBuf to efficiently and scalably explore cuts during mapping. We demonstrate that our method places buffers less aggressively and more accurately: it outperforms two recent optimization methods by achieving 25.92% and 13.32% average speedup while employing 19.79% and 11.14% fewer FFs, respectively. The fact that MapBuf systematically achieves Pareto-optimal design points that were unattainable by prior methods points to its relevance in making HLS-produced circuits efficient and suitable for various FPGA architectures.

REFERENCES

[1] L. Josipović, S. Sheikhha, A. Guerrieri, P. Ienne, and J. Cortadella, "Buffer placement and sizing for high-performance dataflow circuits," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Seaside, CA, Feb. 2020, pp. 186–96.

[2] C. Rizzi, A. Guerrieri, P. Ienne, and L. Josipović, "A comprehensive timing model for accurate frequency tuning in dataflow circuits," in *Proceedings of the 32nd International Conference on Field-Programmable Logic and Applications*, Aug. 2022, pp. 375–83.

[3] *Vivado High-Level Synthesis*, Xilinx Inc., 2018. [Online]. Available: http://www.xilinx.com/products/ design-tools/vivado/integration/esl-design.html

[4] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2, pp. 24:1–24:27, Sep. 2013.

[5] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[6] C. Rizzi, A. Guerrieri, and L. Josipović, "An iterative method for mapping-aware frequency regulation in dataflow circuits," in *Proceedings of the 60rd ACM/IEEE Design Automation Conference*, Jul. 2023, to appear.

[7] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–76, Sep. 2001.

[8] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, San Francisco, CA, 2006, pp. 657–662.

[9] L. Josipović, R. Ghosal, and P. Ienne, "Dynamically scheduled high-level synthesis," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterery, CA, Feb. 2018, pp. 127–136.

[10] Z. Zhang and B. Liu, "SDC-based modulo scheduling for pipeline synthesis," in *Proceedings of the 32nd International Conference on Computer-Aided Design*, San Jose, CA, Nov. 2013, pp. 211–18.

[11] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[12] A. Mishchenko, S. Chatterjee, R. K. Brayton, X. Wang, and T. Kam, "Mapping with boolean matching , supergates and choices," in *ERL Technical Report*, EECS Dept., UC Berkeley, Mar. 2004.

[13] I. Háleček, P. Fišer, and J. Schmidt, "Are XORs in logic synthesis really necessary?" in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, Dresden, Germany, 2017, pp. 134–139.

[14] D. Chen and J. Cong, "Daomap: a depth-optimal area optimization mapping algorithm for FPGA designs," in *IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, Nov. 2004, pp. 752–759.

[15] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in LUT-based FPGA mapping," in *Proceedings of the 3rd ACM international symposium on Field-programmable gate arrays*, 1995, pp. 68–74.

[16] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," in *Proceedings of the 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, Feb. 1998, p. 35–42.

[17] M. Tan, S. Dai, U. Gupta, and Z. Zhang, "Mapping-aware constrained scheduling for LUT-based FPGAs," in *Proceedings of the 23rd ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, Feb. 2015, pp. 190–9.

[18] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2007, pp. 354–361.

[19] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proceedings of the 7th ACM/SIGDA international symposium on Field programmable gate arrays*, Monterey, CA, 1999, pp. 29–35.

[20] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Cutless FPGA mapping," ERL Technical Report, EECS Dept., UC Berkeley, Tech. Rep., 2007.

[21] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs," in *Proceedings of the 14th ACM/SIGDA international symposium on Field programmable gate arrays*, Monterey, CA, 2006, pp. 41–49.

[22] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022. [Online]. Available: https://www.gurobi.com

[23] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin II–an open-source Verilog HDL synthesis tool for CAD research," in *Proceedings of the 18th IEEE Symposium on Field-Programmable Custom Computing Machines*, Charlotte, NC, May 2010, pp. 149–56.

[24] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool," in *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.

[25] Intel, "ModelSim," 2021. [Online]. Available: https://www.intel.com/

[26] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High performance CAD and customizable FPGA architecture modelling," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 1–55, Jun. 2020.

[27] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers design efficient FPGA accelerators," in *Proceedings of the 29th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Virtual Event, USA, Feb. 2021, p. 147.

[28] L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis*, Mountain View, CA, Jun. 2015.