Mapping-Aware Constrained Scheduling for LUT-Based FPGAs

Mingxing Tan, Steve Dai, Udit Gupta, Zhiru Zhang

School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{mingxing.tan, hd273, ug28, zhiruz}@cornell.edu

ABSTRACT

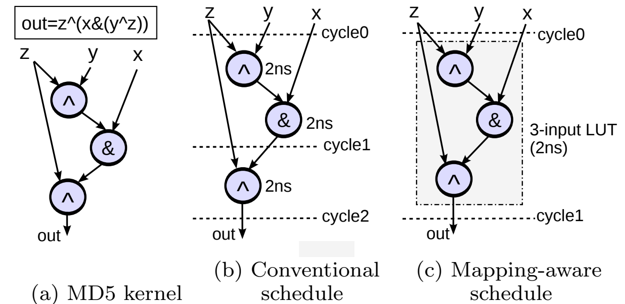
Scheduling plays a central role in high-level synthesis, as it inserts clock boundaries into the untimed behavioral model and greatly impacts the performance, power, and area of the synthesized circuits. While current scheduling techniques can make use of pre-characterized delay values of individual operations, it is difficult to obtain accurate timing estimation on a cluster of operations without considering technology mapping. This limitation is particularly pronounced for FPGAs where a large logic network can be mapped to only a few levels of look-up tables (LUT).

In this paper, we propose MAPS, a mapping-aware constrained scheduling algorithm for LUT-based FPGAs. Instead of simply summing up the estimated delay values of individual operations, MAPS jointly performs technology mapping and scheduling, creating the opportunity for more aggressive operation chaining to minimize latency and reduce area. We show that MAPS can produce a latency-optimal solution, while supporting a variety of design timing requirements expressed in a system of difference constraints. We also present an efficient incremental scheduling technique for MAPS to effectively handle resource constraints. Experimental results with real-life benchmarks demonstrate that our proposed algorithm achieves very promising improvements in performance and resource usage when compared to a state-of-the-art commercial high-level synthesis tool targeting Xilinx FPGAs.

1. INTRODUCTION

As modern-day field-programmable gate arrays (FPGAs) integrate billions of transistors to meet the ever-increasing design complexity, high-level synthesis (HLS) is becoming a major player in improving design productivity and reducing the overall verification effort for large-scale FPGA-based designs [11]. HLS raises the level of design abstraction from register-transfer-level (RTL) modeling to high-level software programming by automatically transforming untimed behavioral descriptions into optimized cycle-accurate hardware implementations. One of the most important steps in HLS is scheduling, which analyzes the parallelism in the input behavioral model and intelligently assigns time steps to operations based on the design constraints. As one would expect, scheduling dictates the clock frequency, latency, and throughput of the synthesized design; it also largely influences the area and power of the final hardware circuit.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA'15, February 22–24, 2015, Monterey, California, USA.
Copyright © ACM 978-1-4503-3315-3/15/02 ...$15.00.
http://dx.doi.org/10.1145/2684746.2689063.



(a) MD5 kernel (b) Conventional schedule (c) Mapping-aware schedule

Figure 1: Reducing latency with mapping-aware scheduling for an MD5 kernel – Target clock period is $5ns$; each logic operation or LUT incurs a $2ns$ delay. Symbols $\wedge$ and & represent `XOR` and `AND` operations, respectively.

When determining the clock boundaries (or placement of registers) given a target clock period, existing scheduling techniques typically rely on delay estimates from the pre-characterization of the RTL building blocks (e.g., logic gates, adders, multiplexers, etc.). The timing analysis on a cluster of operations is usually carried out by simply summing up the estimated delay values of the individual operations on the longest path. While such additive delay models are not unreasonable for FPGA designs dominated by arithmetic operations that use dedicated carry chains and DSP blocks, the estimated delay is often too pessimistic for logic-operation-intensive applications where a large logic network can be mapped to only a few levels of look-up tables (LUTs). Without considering the impact of technology mapping, the overestimation of logic delays during scheduling will easily result in suboptimal performance and resource usage for the synthesized circuit.

Figure 1(a) illustrates the data-flow graph (DFG) of a simple kernel from the MD5 message-digest algorithm, a widely used cryptographic hash function [27]. With a $5ns$ clock period constraint, conventional scheduling algorithms would simply compute the critical path delay of the DFG and insert registers as shown in Figure 1(b) to meet the target clock period constraint, assuming that each logic gate incurs a $2ns$

delay. As a result, the synthesized design requires a latency of two cycles and two LUTs. On the other hand, it is evident from the DFG that all operations can be scheduled combinationally and realized together with a single 3-input LUT, as illustrated in Figure 1(c). The importance of accounting for technology mapping during operation scheduling is obvious from this simple example. An accurate delay estimation of a cluster (or subgraph) of operations should not be a simple function that adds up operation delays on the critical path, but should instead factor in the level of LUTs needed to cover the subgraph. Otherwise, a suboptimal register placement from HLS would unnecessarily increase the latency of the synthesized design and in turn limit the scope of optimizations in the downstream CAD toolflow, including technology mapping.

In this paper, we propose **MAPS**, a **MAP**ping-aware constrained **S**cheduling algorithm for LUT-based FPGAs, to address the inherent inaccuracy of delay estimation in HLS due to the lack of information of LUT mapping. By employing a novel labeling approach that simultaneously produces the time steps of operations as well as their depths in LUT levels, MAPS generates a minimum-latency schedule in the perspective of both the scheduler and technology mapper. Our experiments with a collection of real-life benchmarks demonstrate that MAPS can significantly improve the performance and reduce the register and LUT usage compared to a state-of-the-art commercial high-level synthesis tool targeting Xilinx FPGAs. More specifically, our main contributions are as follows:

1. To our knowledge, we are the first to propose a truly integrated scheduling and mapping algorithm in HLS for FPGAs to fundamentally address and exploit the interdependence between scheduling and LUT mapping for optimizing the performance and area of the synthesized circuits.

2. We present a novel and scalable constrained scheduling algorithm using relaxation-based labeling that is able to achieve a latency-optimal schedule, while supporting a variety of design timing requirements expressed in a system of difference constraints.

3. We also propose an efficient incremental scheduling algorithm to optimize the schedule under resource constraints.

We emphasize that the proposed mapping-aware scheduling algorithm goes beyond a simple extension of traditional technology mapping techniques on a high-level dataflow graph [8, 9]. Technology mapping requires a pre-determined register placement, while scheduling determines such a register placement. We consider technology mapping during scheduling to address this interdependence between scheduling and mapping and optimally place registers such that the timing is met and the latency is minimized. In fact, our approach is analogous to a generalization of retiming-based technology mapping [23, 24]. However, unlike traditional or retiming-based technology mapping techniques, our algorithm starts with an untimed design and is able to handle a rich set of scheduling constraints. In addition, traditional and retiming-based technology mapping are not able to propagate registers into a cycle of a circuit, and do not solve the mapping-aware scheduling problem.

The rest of the paper is organized as follows: Section 2 reviews previous work on scheduling and mapping; Section 3 provides background on scheduling constraints and mapping techniques; Section 4 presents the MAPS algorithm; Section 5 reports experimental results, followed by additional discussions in Section 6; We conclude the paper in Section 7.

2. RELATED WORK

A large number of scheduling algorithms have been proposed to optimize for various design metrics related to performance, area, and power. Constrained scheduling is in general NP-hard and thus usually relies on heuristic algorithms. Examples of classical scheduling heuristics include force-directed scheduling [25] and list scheduling [1]. Recently, the application of system of difference constraints (SDC) [12, 15] has enabled an efficient and scalable linear programming approach to the constrained scheduling problem that encapsulates a rich set of realistic scheduling models including chained operations, multi-cycle operations, frequency constraints, and relative timing constraints in I/O protocols. SDC can be extended to handle loop pipelining [30, 4], an important scheduling optimization in the presence of loop-carried dependence. We note that our MAPS framework can also efficiently handle these SDC constraints.

In addition to optimizing HLS for classic design objectives such as performance [28, 16] and area [30], there is a growing trend in integrating upstream scheduling with downstream physical implementation. For example, Cong et al. [10] evaluate metrics for quantifying interconnect optimization opportunity during scheduling and envision a scheduling approach that generates layout-friendly RTL architecture. Most recently, Zheng et al. [31] propose an HLS flow that performs scheduling and place-and-route iteratively. By back-annotating more realistic post-place-and-route delay estimates after each iteration, the HLS tool is able to compute an improved schedule for timing closure. Obviously, an efficient back-annotation flow can also benefit our MAPS approach by providing more accurate delay estimates. In this work, we further address the limitation of the additive delay model assumed by the existing HLS tools.

In relation to scheduling, mapping is a downstream step in the design implementation flow and can affect the final quality of results (QoR) even with an optimized schedule [6]. There has been an extensive amount of research on mapping with optimization objectives ranging from LUT depth [8] and area [14, 5] traditionally to reliability [13] and even security [3] more recently. Prominent mapping techniques include FlowMap [8], CutMap [9], and DaoMap [5].

Apart from integrating mapping with upstream logic synthesis [7] and downstream placement [21], Pan et al. propose a retiming-based technology mapping technique that considers mapping for register repositioning to achieve the minimum clock period [23, 24]. While it is able to achieve the optimal clock period, the retiming-based mapping approach is limited to circuits with an initial register placement and cannot perform actual scheduling on an untimed circuit. In Section 6, we will show that MAPS is a generalization of the retiming-based mapping.

3. PRELIMINARIES

Scheduling is the problem of assigning time steps to operations from an untimed behavioral description to synthe-

size a cycle-accurate RTL model. Technology mapping (or mapping for short) is the process of transforming a graph of technology-independent logic elements into technology-dependent logic cells, such as LUTs, DSP blocks, and memories, on the target FPGA device. In this paper, we focus on the LUT mapping problem of logic operations. Operations that are not mapped to LUTs (e.g., memory operations) are referred to as *black-box operations.*

3.1 Scheduling Constraints

Constraints are an important ingredient of the scheduling formulation. Commonly encountered constraints in HLS include *dependence constraints* which arise from data and control dependences in the control-data flow graph (CDFG), *latency and relative timing constraints* which define the required maximum or minimum number of cycles between two operations (e.g., user-specified I/O protocols), and the *clock period constraint* which ensures that the critical path meets the target clock period.

It has been shown that all of the aforementioned constraints can be precisely represented and efficiently solved in the form of system of difference constraints, or SDC [15, 30]. Under the SDC formulation, dependence constraints can be represented in the form of $s_u - s_v \leq 0$, where s_u and s_v denote the control steps of operations u and v. Latency and relative timing constraints are represented in the form $s_u - s_v \leq d$, where d is the minimum schedule time difference between u and v. SDC-based scheduling makes use of the *constraint graph* during feasibility checking and optimization. In the constraint graph, each edge $u \rightarrow v$ with a weight d represents an SDC constraint $s_u - s_v \leq d$. The constraint graph can be easily constructed from the CDFG by keeping the same set of nodes on CDFG and adding edges for each SDC constraint [30].

It is important to note that although MAPS is designed to conveniently handle various SDC constraints, it purposely avoids clock period difference constraints. Such modeling of the clock period constraints assumes additive delay among chained operations as in conventional scheduling approaches and is a reason for sub-optimal QoR.

3.2 Technology Mapping

Mapping is typically performed on a directed acyclic graph (DAG) of logic gates, abstractly represented as nodes, for the combinational paths between register boundaries. Let C_v denote a cone rooted at node v, defined as the sub-graph of v and some of its predecessors such that there exists a path from any node in C_v to v that is entirely contained within C_v. The cone C_v is K-feasible if there are no more than K nodes outside C_v with edges pointing to nodes in C_v. A cut of C_v, denoted as $CUT(C_v)$, is defined to be the set of input nodes of C_v. $CUT(C_v)$ is K-feasible if C_v is a K-feasible cone. In Figure 1(c), for example, the three nodes form a 3-feasible cone rooted at the bottom node.

In LUT-based FPGAs, any K-feasible cones can be implemented with a K-input LUT (or K-LUT), so the mapping problem reduces to the problem of optimally covering the input graph with K-feasible cones [8, 23]. Such mapping framework generally consists of cut enumeration, cut ranking, cut selection, and final mapping generation. Cut enumeration explores all K-feasible cuts rooted at each node, while cut ranking evaluates these cuts based on the optimization objective. Cut selection typically follows a reverse topological order to select the most optimal cut for each node based on the previous ranking information to generate the final mapping solution. In Section 4.1, we will discuss our customized cut enumeration algorithm for MAPS.

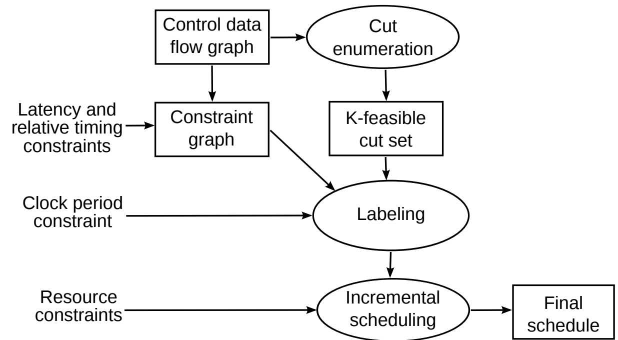


Figure 2: Overall design flow of MAPS.

3.3 MAPS Problem Formulation

We formulate the mapping-aware scheduling problem as an optimization problem formally stated as follows:

Given: (1) A CDFG G that captures data and control dependence constraints; (2) A target clock period T_{cp}; (3) A set of additional scheduling constraints C, including latency and relative timing constraints expressed in the form of SDC, and resource constraints; (4) A target FPGA device using K-input LUTs.

Goal: Find a minimum-latency schedule for G so that no constraints in C are violated, and within each cycle, there exists a feasible K-LUT mapping that meets T_{cp}.

4. MAPS ALGORITHM

In this section we present MAPS, a mapping-aware constrained scheduling algorithm to address and exploit the interdependence between scheduling and LUT mapping to minimize the latency of the synthesized design. Figure 2 shows the overall flow for MAPS. MAPS takes the CDFG along with a variety of scheduling constraints as inputs, and generates a minimum-latency schedule. The algorithm consists of three major steps: (1) *Cut enumeration* generates all K-feasible cuts for each node in the CDFG using a work-list-based approach; (2) *Relaxation-based labeling* computes for each operation its minimum time step and minimum level (in terms of LUT depth) allowed by the design constraints; (3) *Incremental scheduling* legalizes the schedule for resource-constrained operations to avoid resource conflicts based on the results from labeling.

4.1 Cut Enumeration

MAPS generates all K-feasible cuts at each node as part of the process for determining the minimum schedule latency. Unlike traditional cut enumeration algorithms that typically operate on an acyclic DAG with logic operations, MAPS deals with both logic operations and *black-box operations* that cannot be mapped to LUTs, such as multiplications and memory reads/writes. In addition, MAPS needs to handle cycles that arise from loop-carried dependences on CDFG.

Typically, the K-feasible cut set for each node v can be computed based on its inputs on the CDFG. Suppose v's inputs are u_1, u_2, ..., u_p and their own cut set are $CUTS_{u_1}$, $CUTS_{u_2}$, ..., $CUTS_{u_p}$, where $CUTS_{u_i}$ is a collection of cuts

and each cut $c \in CUTS_{u_i}$ is a K-feasible cut. The K-feasible cut set for v can be computed as follows:

- If v is a black-box operation that does not map to LUTs, the only legal cut for v is itself, which is a *trivial cut*, i.e., $CUTS_v = \{\{v\}\}$, as block boxes will never be packed together with any other operation.
- If v is a logic operation, the K-feasible cuts for v can be computed by merging the K-feasible cuts for all its inputs as follows:

$$\begin{aligned} &CUTS_v = mergeCuts(u_1, ..., u_p) = \\ &\{C' = C_1 \cup ... \cup C_p | C_i \in CUTS_{u_i}, |C'| < K\} \end{aligned} \tag{1}$$

Without considering cycles, we can easily compute the cut set for each node by traversing the graph in topological order using conventional cut enumeration approaches [9]. However, a simple topological traversal is not enough when the graph contains cycles. To handle the cycles on CDFG, our cut enumeration algorithm iteratively applies Equation (1) to each node until convergence, when all K-feasible cuts are obtained. More specifically, we maintain a work list for nodes that need to be updated. Initially, the work list contains all operations, and the cut set for each node is the trivial cut. For each node in the work list, we apply Equation (1) to compute the new cut set. If a new cut is added for a node, we update its cut set and add all its successors to the work list. We remove a node from the work list each time it is visited. The algorithm terminates when the work list becomes empty. As suggested by previous studies [14], cut enumeration is an exponential algorithm with respect to K. Nevertheless, the actual runtime for cut enumeration is typically fast for $K \leq 6$.

Our cut enumeration algorithm can be applied on both word-level or bit-level dependence graphs. By decomposing the word-level CDFG into bit-level dependence graph [29], we are able to capture the exact bit-level inputs for each operation, but such graph decomposition would complicate our scheduling and mapping. To make our algorithm more scalable and efficient, we currently implement our cut enumeration algorithm on word-level CDFG. In this case, we cannot simply compute the inputs of each node based on word-level values, because each output bit may depend on multiple input bits of a single word-level value. For example, given an arithmetic operation $C[1:0] = A[1:0] + B[1:0]$, the highest output bit C_1 depends on four input bits A_1, A_0, B_1, and B_0, which come from only two word-level values A and B on CDFG.

To handle such bit-level dependence, our cut enumeration algorithm performs bit-level dependence analysis based on CDFG for different types of operations: (a) For bitwise logic operations such as `AND`, `OR`, and `XOR`, each bit is independent of the other bits, so we can compute the word-level dependence on the CDFG without considering the bits of each value; (b) For other bitwise operations such as `SHIFT`, `ZEXT`, `SEXT`, and `TRUNC`, each output bit depends on at most a single input bit, but not necessarily the one at the same position. In this case, we annotate each input value with bit positions to distinguish different bits of the same word-level value. (c) For arithmetic operations such as `ADD` and `SUB`, each output bit can be dependent on multiple input bits. The highest output bit is dependent on the largest number of input bits, and we always examine this bit to consider the worst case. As with (b), we annotate each input value with the bit positions it depends on. For the given example $C[1:0] = A[1:0] + B[1:0]$, our analysis will determine that C has four inputs A_0, A_1, B_0, and B_1.

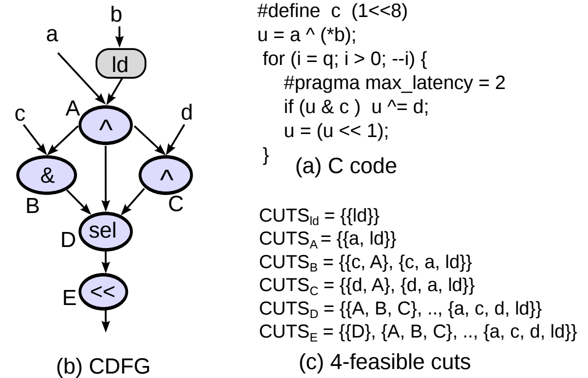


Figure 3: Cut enumeration for a CRC kernel – The memory read `ld` is a black-box operation, while `A-E` are logic operations; $\wedge$ = `XOR`; & = `AND`; $<<$ denotes a left shift; *sel* denotes a `MUX` operation.

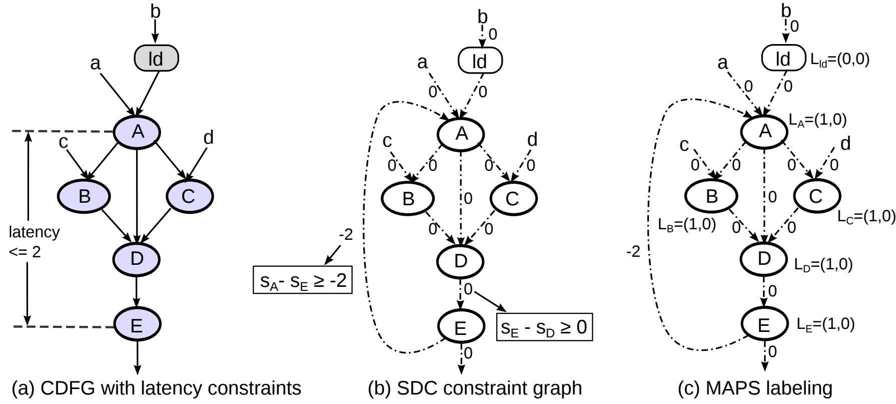
Figure 3 demonstrates the cut enumeration for a cyclic redundancy check (CRC) kernel. Based on the source code (a) and the corresponding CDFG (b), the cut set for each node are listed in (c). For simplicity, we only consider one iteration of the loop. As shown in this figure, the black-box operation `ld` only has a trivial cut, but other operations can have multiple K-feasible cuts. Note that the `AND` between u and a special constant $c = (1 << 8)$ is used to test the highest bit of u. For this special case, our bit-level dependence analysis will determine that the output value only depends on a single input bit.

4.2 Relaxation-Based Labeling

The labeling step aims to compute a minimum-latency schedule while considering mapping and respecting all SDC constraints for dependence, relative timing, latency, and clock period. Unlike conventional SDC scheduling algorithms, our technique considers LUT mapping and computes both the time step and the LUT level within the time step for each operation. Our technique also differs fundamentally from technology mapping algorithms, which operate on an acyclic DAG with pre-defined register boundaries; instead, it operates on an untimed CDFG along with user-specified timing constraints which may result in additional cycles. To this end, we introduce a relaxation-based labeling algorithm, which to our knowledge is the first to achieve a latency-optimal solution for both time step and LUT level under timing constraints captured in the SDC form.

To jointly represent the time step and LUT depths of each operation, we define the *L-value* of node v, $L_v = (s, l)$, where s denotes the time step of v in cycles, and l denotes the arrival time of v within a time step. Given a target clock period T_{cp} in LUT levels, it follows that $s > 0$ and $1 \leq l < T_{cp}$. For simplicity we assume that the delay of each operation is quantized to LUT levels, and is less than T_{cp} (that is, no combinational multi-cycle operations). However, the proposed approach can easily be generalized to handle real-valued delays and to multi-cycle operations.

We define the following operations for L-values below. Here $L_1 = (s_1, l_1)$, $L_2 = (s_2, l_2)$, *Delay* is a delay value



(a) CDFG with latency constraints (b) SDC constraint graph (c) MAPS labeling

Figure 4: MAPS labeling for CRC – (a) CDFG with a latency constraint for the loop body shown in Figure 3; (b) SDC constraint graph that captures dependence constraints (e.g., $s_E - s_D \geq 0$ where s_D and s_E denote the time step of D and E, respectively.), and a latency constraint (i.e., $s_A - s_E \geq -2$); (c) MAPS labeling results for the constraint graph based on the cut information from Figure 3. Each label $L_v = (s, l)$ represents the L-value of node v, where s denotes the time step of v in cycles, and l denotes the level of v in LUT depth.

of a combinational operation quantized to LUT levels, and Lat is a time step difference value imposed by user-defined relative timing and latency constraints. The first two operations are used to update the L-value for each node based on different constraints, while the following operations are used to facilitate the description of our labeling algorithm.

$$L_1 + (0, Delay) = \begin{cases} (s_1, l_1 + Delay) & \text{if } (l_1 + Delay) < T_{cp} \\ (s_1 + 1, Delay) & \text{otherwise} \end{cases}$$

$$L_1 + (Lat, 0) = (s_1 + Lat, 0)$$

$$L_1 < L_2 = \begin{cases} true & \text{if } (s_1 < s_2) \text{ or } (s_1 = s_2 \text{ and } l_1 < l_2) \\ false & \text{otherwise} \end{cases}$$

$$\min(L_1, L_2) = \begin{cases} L_1 & \text{if } L_1 < L_2 \\ L_2 & \text{otherwise} \end{cases}$$

$$\max(L_1, L_2) = \begin{cases} L_2 & \text{if } L_1 < L_2 \\ L_1 & \text{otherwise} \end{cases}$$

In order to obtain a minimum-latency schedule while considering a rich set of scheduling constraints, our labeling algorithm computes the *optimal L-values*, i.e. the minimum L-values among all legal schedules, based on the constraint graph of SDC [26] constructed from the CDFG. A straightforward idea for computing the optimal L-value of each node is to propagate the L-values from primary inputs of the CDFG to the primary outputs if the constraint graph is acyclic. However, additional constraints such as maximum relative timing constraints may result in cyclic dependences. To overcome this challenge, we introduce a relaxation-based labeling algorithm, which maintains a lower bound on the L-value of each node and successively relaxes this lower bound to meet each of the scheduling constraints for the node.

Algorithm 1 details the proposed relaxation-based labeling. Let $Delay_v$ denote the delay value of node v quantized to the number of LUT levels, $Lat_{u \to v}$ denotes the weight of the edge from u to v on the SDC constraint graph, which also represents the minimum time step difference between u and v. As shown in lines $1-2$, the L-value of each node v is initialized to $(0, Delay_v)$, which is a trivial lower bound on v's L-value without any constraints. The algorithm then iteratively updates the lower bound on each node's L-value (line 3-19) by propagating both mapping and scheduling

Algorithm 1: $Labeling(CG, CUTS)$

input : CG – constraint graph with nodes V and edges E; $CUTS$ – cut set for all nodes on CDFG.
output: L – labels for each node.

```
1  foreach node v in V do
2      L_v ← (0, Delay_v)
   // Tighten the labels by at most B iterations
3  for i ← 1 to B do
4      updated ← false
5      foreach node v in V do
           // Mapping constraints
6          f_v ← (∞, ∞)
7          foreach cut C in CUTS_v do
8              L' ← (0, 0)
9              foreach node u in cut C do
10                 L' ← max{L', L_u + (0, Delay_v)}
11             f_v ← min(f_v, L')
           // Latency & relative timing constraints
12         g_v = (0, 0)
13         foreach edge u → v in E do
14             g_v ← max{g_v, L_u + (Lat_{u→v}, 0) + (0, Delay_v)}
15         if L_v < max{f_v, g_v} then
16             L_v ← max{f_v, g_v}
17             updated ← true
18     if updated = false then
19         return SUCCESS
20 return FAILURE
```

constraints according to Equations (2) and (3):

$$L_v \geq f_v = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{L_u + (0, Delay_v)\} \quad (2)$$

$$L_v \geq g_v = \max_{\forall u \to v \in E} \{L_u + (Lat_{u \to v}, 0) + (0, Delay_v)\} \quad (3)$$

Here f_v and g_v are two lower bounds of node v's L-value required by mapping constraints and SDC scheduling constraints, respectively. Following Equation (2), lines $6-11$ in Algorithm 1 deal with mapping constraints, and calculate the the new f_v of node v by selecting the best cut for v. Similarly, lines $12-14$ calculate the new g_v of node v based on various SDC scheduling constraints on the constraint graph. If any of f_v and g_v is greater than the original L_v, we update $L_v = \max\{f_v, g_v\}$ via relaxation in lines $15-17$.

LEMMA 1. *For each iteration of Algorithm 1, L_v is always a lower bound of node v's L-value, i.e., L_v is less than or equal to v's L-value in any legal schedule that satisfies the mapping and scheduling constraints.*

PROOF. Let S be a legal schedule, $\overline{L_v}$ be the L-value for each node v in S, $\overline{f_v}$ and $\overline{g_v}$ be the lower bounds of v's L-value computed according to Equations (2) and (3) for S:

$$\overline{f_v} = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{\overline{L_u} + (0, Delay_v)\}$$

$$\overline{g_v} = \max_{\forall u \rightarrow v \in E} \{\overline{L_u} + (Lat_{u \rightarrow v}, 0) + (0, Delay_v)\}$$

Because S is a legal schedule, each node v must satisfy both mapping and scheduling constraints, i.e., $\overline{L_v} \geq \max\{\overline{f_v}, \overline{g_v}\}$. Considering Algorithm 1, let L_v^i denote node v's L-value after iteration i ($i \geq 0$). We prove by induction that for each iteration i:

$$L_v^i \leq \overline{L_v}, \quad \forall \text{ node } v \tag{4}$$

Base case: When $i = 0$, our algorithm initializes $L_v^0 = (0, Delay_v)$, which is a trivial lower bound because every operation cannot finish before its own operation delay; therefore, it is evident that $L_v^0 \leq \overline{L_v}$.
Induction step: Suppose (4) is true for $i = k$, i.e., $L_u^k \leq \overline{L_u}$ for each node u. Considering $i = k+1$, Algorithm 1 would update L_v^{k+1} according to Equations (2) and (3):

$$f_v = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{L_u^k + (0, Delay_v)\} \leq \overline{f_v}$$

$$g_v = \max_{\forall u \rightarrow v \in E} \{L_u^k + (Lat_{u \rightarrow v}, 0) + (0, Delay_v)\} \leq \overline{g_v}$$

$$L_v^{k+1} = \max\{f_v, g_v\} \leq \max\{\overline{f_v}, \overline{g_v}\} \leq \overline{L_v}$$

It is clear for $f_v \leq \overline{f_v}$ and $g_v \leq \overline{g_v}$ by substitute every L_u^k with $\overline{L_v}$ and applying the inductive assumption $L_u^k \leq \overline{L_u}$. Evidently, Equation (4) is also true for $i = k+1$. □

Algorithm 1 iteratively relaxes L-values until convergence. SInce L-values are always lower bounds in each iteration, Algorithm 1 is guaranteed to converge to a legal schedule with optimal L-values if the problem is feasible.

LEMMA 2. *All nodes on the constraint graph would reach their optimal L-values within $B = D \cdot (|V|-1) \cdot |V|$ iterations if the scheduling problem is feasible.*

Here $|V|$ denotes the total number of nodes on the constraint graph, while D denotes the maximum delay in terms of LUT levels for any edge on the constraint graph. Lemma 2 is relatively obvious by considering the worst-case mapping. In the worst case, each node is mapped to a distinct LUT (i.e., only trivial cuts are chosen). Since any simple path on the constraint graph can include at most $|V| - 1$ nodes, the L-value of any node can be at most $D \cdot (|V| - 1)$. As shown in Algorithm 1, at least one of L-values is increased by at least one in each iteration; otherwise, all nodes would have obtained the optimal L-values and the algorithm would successfully exit. Therefore, after at most $B = D \cdot (|V|-1) \cdot |V|$ iterations, all nodes should have settled at the optimal L-values if the scheduling problem is feasible.

THEOREM 1. *Algorithm 1 returns a legal schedule with optimal L-value for each node in pseudo-polynomial time, and thus is able to achieve a minimum-latency schedule with pseudo-polynomial complexity.*

Algorithm 2: *IncrementalScheduling*$(CG,\ C_r,\ L)$

```
   input  : CG – constraint graph; C_r – resource constraints;
            L – initial labels from the labeling step.
   output: S – final schedule.
1  s ← 0 // Time step
2  while more resource-constrained nodes to schedule do
3      foreach unscheduled resource-constrained node v with
       time step s, in ascending priority order do
4          if no resource conflict for v at s then
5              schedule v at s
6              update resource scoreboard
7          else
8              L_v ← (s + 1, 0)
9              update the labels for v's successors on CG
10             if labeling is infeasible then
11                 report failure and exit
12     s ← s + 1;
```

If the scheduling problem is feasible, Algorithm 1 returns successfully after all nodes settle at their lower bounds while satisfying all dependence, relative timing, and clock period constraints (lines 17-18). Otherwise, it reports failure (line 19). As the L-value L_v essentially represents the number of time steps plus the number of LUT levels from primary inputs to node v, we have shown that the L-values computed by Algorithm 1 are optimal L-values, and our schedule is thus minimum-latency.

Figure 4 illustrates the labeling process for the CRC kernel described in Figure 3. Given the CDFG and a user-specified latency constraint in Figure 3(a), our labeling algorithm first constructs the constraint graph as shown in Figure 4(b). The constraint graph contains the same set of nodes on the CDFG but includes a different set of weighted edges, with each edge representing an SDC constraint. For each edge on the CDFG, we add a zero-weighted edge accordingly to the constraint graph to represent a dependence constraint (e.g., $A \rightarrow ld$ captures $s_A - s_{ld} \geq 0$). For the latency constraint, we add an additional edge from E to A with a weight of -2, representing the latency constraint $s_A - s_E \geq -2$. As shown in Figure 4(b), the additional latency constraint can introduce a new cycle on the constraint graph. Based on the constraint graph, we compute the L-value for each node in topological order of the original data flow graph using Algorithm 1. Here we assume that the black-box operation `ld` takes a full cycle. Therefore, node `A` has a L-value of (1,0), indicating that the earliest time step of `A` is one and the minimum LUT depth is zero. Our algorithm computes the L-values for logic operations `A-E` based on their cut information. For example, the best cut for node `E` is `{a,c,d,ld}`, suggesting an optimal L-value of (1,0). The final labeling results are shown in Figure 4 (c).

4.3 Incremental Scheduling

In this section we introduce an incremental scheduling algorithm to handle the resource constraints on black-box operations (e.g., memory port limits). Given that resource-constrained scheduling is NP-hard in general, MAPS employs a heuristic method which legalizes an initial schedule from the previous labeling step by incrementally rescheduling the operations that cause resource conflicts.

Algorithm 2 lists the pseudo-code for our incremental scheduling algorithm. We start from an initial schedule obtained from the labeling step where all nodes are labeled

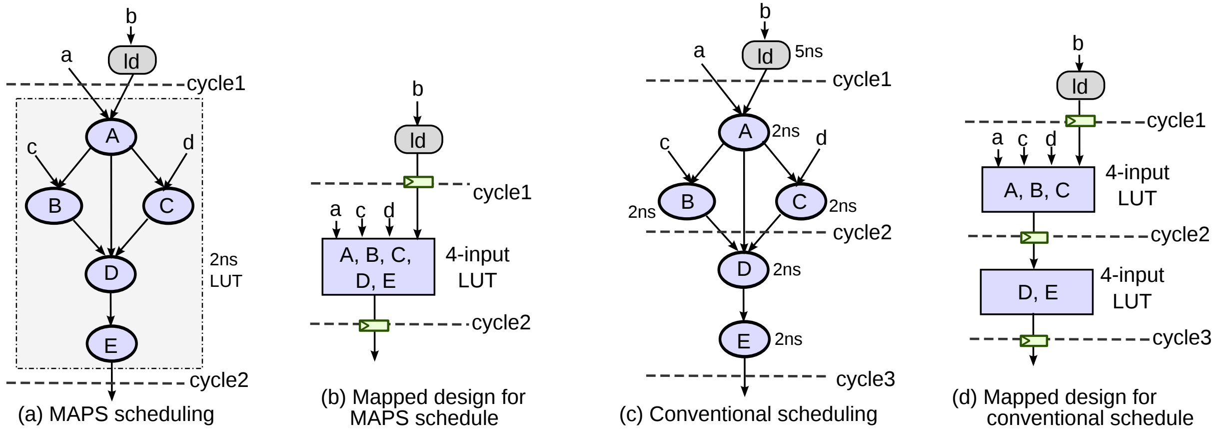


Figure 5: MAPS scheduling for CRC – (a) MAPS scheduling for CDFG nodes based on their L-values from Figure 4; (b) Mapped design for (a), which results in a 2-cycle latency with one LUT and two registers; (c) Conventional schedule for CRC based on pre-characterized delay values of individual operations; (d) Mapped design for (c), which results in a 3-cycle latency with two LUTs and three registers. Here we assume that the target clock period is $5ns$; each LUT or logic operation takes $2ns$; the black-box operation `ld` takes a full cycle.

with the optimal L-values, i.e., scheduled as-soon-as-possible (ASAP), and legalize the solution by gradually postponing resource-constrained nodes to later time steps based on a priority function. For each time step s, we check if a resource conflict exists for any resource-constrained nodes that are initially assigned to s. If so, we postpone the ones with the least priorities, which to some extent is similar to how list scheduling operates. The priority of each node is determined based on its ASAP label (i.e., the L-values obtained from Algorithm 1) and its as-late-as-possible (ALAP) label that can be computed in a similar manner using backward propagation. More specifically, we compute the ALAP label for each node by iteratively updating the upper bound of its L-value. Initially, the ALAP label of each node v is set to the maximum L-value, the upper bound L-value without any constraints; afterwards, for each cone C rooted at r that contains v, we tighten the upper bound of v's L-value by checking every successor of r; meanwhile, we also refine the upper bound of v's L-value by checking each outgoing edge from v on the constraint graph to ensure that all SDC timing constraints are satisfied. We repeatedly tighten the upper bound of L-value for each node until convergence. Otherwise, the problem is infeasible and we will stop after at most $B = D \cdot (|V| - 1) \cdot |V|$ iterations similar to the previous ASAP labeling process.

After obtaining both ASAP and ALAP labels, we can calculate the priority of each node based on its *mobility*, which is defined to be the difference between the ALAP and ASAP labels. Intuitively, nodes with a lower mobility are assigned with a higher priority. If we increase the label for a resource-constrained node v, we also incrementally update the labels for all the successors of v using the relaxation-based labeling process introduced in the previous section.

Figure 5 illustrates the MAPS scheduling results for the CRC kernel. Since `ld` is the only black-box operation, there is no resource conflict for this example. The scheduled design will be further mapped to LUTs and registers as shown in Figure 5(b). For comparison, Figure 5 (c) and (d) illustrate the conventional schedule and the corresponding mapped design. Clearly, our MAPS algorithm is able to reduce the latency by mapping multiple operations into a single LUT, and at the same time reduce the LUT and register usage by maximizing the utilization of each LUT.

Overall Time Complexity – It is important to notice that the L-values always increase monotonically during both relaxation labeling and incremental scheduling. According to Lemma 2, the upper bound of any L-value is $D \cdot (|V| - 1)$, so the maximum number of iterations involved during relaxation labeling and incremental scheduling will be bounded by $D \cdot (|V| - 1) \cdot |V|$. In each iteration, L-values are updated optimally according to each edge in the constraint graph, and the complexity for each iteration is bounded by $O(|E|)$. As a result, the total time complexity for the relaxation and incremental scheduling is $O(D \cdot |V|^2 \cdot |E|)$. Note that since the maximum latency value D is usually a very small constant, our algorithm would converge in polynomial time $O(|V|^2 \cdot |E|)$.

5. EXPERIMENTAL RESULTS

Our experiments are conducted based on a widely used, state-of-the-art commercial HLS tool, which takes a behavioral C/C++ program as an input and outputs RTL code in VHDL or Verilog based on the LLVM compiler infrastructure [19]. To our knowledge, the commercial HLS tool makes use of pre-characterized delay values of individual operations to perform scheduling and relies on the underlying logic synthesis tool to perform technology mapping after the HLS stage (to which we refer as the baseline approach).

We have implemented the MAPS algorithm in C++ as an LLVM pass, which takes the LLVM IR from the original HLS tool as input, reorders instructions based on our proposed mapping-aware constrained scheduling algorithm, and generates tool-specific intrinsics (i.e., wait statements) to specify clock boundaries so that the HLS tool will preserve our scheduling results. We push the scheduling results through the same HLS engine and RTL back end to perform resource binding and RTL code generation. The generated Verilog RTL design is implemented by Xilinx Vivado 2013.4 targeting a Virtex-7 FPGA device. Notably, both the baseline approach and our MAPS approach rely on the same downstream CAD toolflow to perform technology mapping. Nevertheless, our MAPS approach can generate schedules

Table 1: Description for MAPS benchmarks.

Design	Source	Application Domain	# of LLVM IR Operations	Description
XORR	Kernel		773	XOR reduction on a bit vector
GFMUL	Kernel		85	Galois field multiplication
CLZ	Kernel		450	Couting the number of leading zeros in a 64-bit word
CRC	MiBench [17]	Communication	58	Cyclic redundancy check for error detection
MD5	MiBench [17]	Cryptography	996	Message-digest algorithm
AES	CHStone [18]	Cryptography	806	Advanced encryption standard
SHA	CHStone [18]	Cryptography	476	Secure hash algorithm
DFADD	CHStone [18]	Scientific computing	838	Double-precision floating-point addition
MT	[22]	Scientific computing	1171	Mersenne twister 32-bit random number generator
RS	[2]	Communication	840	Reed-Solomon decoder
DR	[20]	Machine learning	262	Digit recognition based on K-nearest neighbors

that are more friendly for LUT mapping, resulting in higher quality of results.

We have evaluated the proposed technique with a broad range of benchmarks used in a variety of application domains, such as cryptography, scientific computing, communication, and machine learning. Table 1 briefly describes these benchmarks. For each benchmark, we apply the same set of HLS front-end optimizations such as loop unrolling and array partition for both the baseline approach and our approach. Our proposed algorithm focuses on mapping optimization for logic operations (e.g. `AND`, `OR`, and `XOR`), bitwise operations (e.g. `ZEXT`, `SEXT`, `SHIFT`, and `TRUNC`), and narrow-bit-width arithmetic operations (e.g. `ADD`, `SUB`, and `CMP`), while all other operations are treated as black-box operations. We model the delay of each black-box operation using pre-characterized delay values parsed from the schedule report generated by the commercial HLS tool; the delay of non-black-box operations are modeled based on the delay of a single LUT.

5.1 Results for Representative Kernels

We first present the results for three representative application kernels, i.e. `CLZ, XORR, GFMUL`, which provide insights for the advantages of MAPS. In these kernels, most of the operations are logic operations, which can be aggressively chained and mapped to a few LUTs. Table 2 shows the detailed timing, latency, and resource usage for each kernel.

Figure 6 lists the C code snippets for all kernels. `XORR` applies a simple `XOR` operation over a bit vector and forms a balanced `XOR` reduction tree with tree-depth of 10. Given the $5ns$ target clock period, the original HLS tool can chain at most six levels of `XOR` operations into a single cycle based on the pre-characterized delay values, resulting in a two-cycle latency. By considering technology mapping, our maps algorithm is able to map all the `XOR` operations into five levels of 6-input LUTs, which can achieve a zero-latency combinational design while still meeting the target clock period constraint. Similarly, `GFMUL` applies a set of `AND/XOR/SHIFT` operations to a few inputs, while `CLZ` consists of a number of `AND/OR` operations to compute the number of leading zeros in a 64-bit word. Since all operations in these kernels are logic or bitwise operations, MAPS can effectively pack all operations of GFMUL to form a combinational circuit, and reduce the latency of `CLZ` to one.

By packing more logic operations together and enabling more aggressive operation chaining, our MAPS algorithm also provides added benefits by reducing register usage due to the shorter latency, and at the same time, reducing LUT

```
out = 0
for (i = 0; i < n; i++) {
   #prgma hls unroll
   out ^= in[i]
}
```

(a) XORR

```
for (i = 0; i < n; i++) {
   #pragma hls unroll
   if (b & (1<<i))
      out ^= a << i;
}
```

(b) GFMUL

```
zero = 1;  out = 0;
for (i = 0; i < 64; i++) {
   #pragma hls unroll
   if (x[i]) zero = 0;
   out += zero;
}
```

(c) CLZ

Figure 6: C code snippets for three kernels.

usage by increasing the utilization of each individual LUT. For example, MAPS completely removes all registers (or FFs) for `XORR` and `GFMUL`. For the `CLZ` kernel, MAPS can reduce the number of FFs by 78% and LUTs by 40% with a $5ns$ target clock period.

5.2 Results for Real-Life Applications

We have also evaluated our approach on a number of real-life applications selected from a broad range of domains. Table 2 shows the latency and resource usage comparisons for these designs. We observe MAPS can significantly reduce the latency over the commercial HLS tool (up to 60%) while still meeting timing for all applications. These results provide further evidence that MAPS is able to generate more efficient solutions with shorter latency and hence higher performance by better utilizing the clock period through chaining more operations in each cycle.

From Table 2, we also observe that our approach can effectively reduce the usage of FFs and LUTs by eliminating unnecessary registers across clock boundaries and maximizing the utilization of each individual LUT. On average, MAPS reduces the number of FFs by 25% and the number of LUTs by 9% with the $5ns$ target clock period. However, it is worth noting that since our current approach mainly focuses on latency optimization, it may lead to LUT duplication and thus increase resource usage in a few cases (e.g. `SHA` and `MD5`).

5.3 Efficiency Analysis for MAPS

To understand the efficiency of MAPS, we have evaluated the execution time for the HLS process with different scheduling algorithms. Table 3 indicates that all benchmarks finish in several seconds, and the additional runtime overhead for MAPS is negligible. While the worst-case time complexity of MAPS is relatively high due to iterative labeling, our experimentation reports that the labeling algorithm converges within only a few iterations for all designs.

Table 3 also lists the number of operations per cycle for different scheduling algorithms. Compared to the baseline approach, MAPS can pack more operations together by con-

Table 2: Timing and resource usage comparison with target clock periods of 8ns and 5ns – `CP` = achieved clock period; `LAT` = latency in # of cycles; `LUT` = # of lookup-tables; `FF` = # of flip-flops; `AVERAGE` = the geometric mean of improvements for the eight real-life applications. Values in parentheses are percentage of increase (+) or decrease (-) over the baseline approach used in a state-of-the-art commercial HLS tool.

		Target Clock Period = 5ns				Target Clock Period = 8ns			
Design	Approach	CP(ns)	LAT	LUT	FF	CP(ns)	LAT	LUT	FF
XORR	baseline	2.88	1	133	17	4.38	1	124	3
	MAPS	2.28	0 (-100%)	120 (-10%)	0 (-100%)	2.19	0 (-100%)	120 (-3%)	0 (-100%)
GFMUL	baseline	2.93	2	50	27	4.38	1	50	18
	MAPS	1.68	0 (-100%)	43 (-14%)	0 (-100%)	1.64	0 (-100%)	43 (-14%)	0 (-100%)
CLZ	baseline	2.93	11	177	169	4.43	7	139	121
	MAPS	2.93	1 (-91%)	107 (-40%)	38 (-78%)	4.38	1 (-86%)	87 (-37%)	17 (-86%)
CRC	baseline	2.93	161	57	310	4.43	129	52	249
	MAPS	2.93	65 (-60%)	41 (-28%)	126 (-59%)	4.43	65 (-50%)	41 (-21%)	126 (-49%)
MD5	baseline	4.39	126	9175	6747	5.83	67	9316	4952
	MAPS	4.24	95 (-25%)	8812 (-4%)	8417 (+25%)	5.94	48 (-28%)	8570 (-8%)	6626 (+34%)
AES	baseline	4.78	197	4895	5855	5.74	141	4322	4459
	MAPS	4.44	133 (-32%)	3989 (-19%)	3540 (-40%)	5.90	109 (-23%)	4007 (-7%)	3369 (-24%)
SHA	baseline	4.21	561	2916	3196	5.10	321	3720	2331
	MAPS	3.87	421 (-25%)	3032 (+4%)	3263 (+2%)	5.97	241 (-25%)	3146 (-15%)	2466 (+6%)
DFADD	baseline	4.81	11	5950	2735	6.44	8	5282	1671
	MAPS	4.80	10 (-9%)	5528 (-7%)	2106 (-23%)	6.35	7 (-12%)	4353 (-18%)	1527 (-9%)
MT	baseline	3.96	146	3617	4630	6.31	59	7652	2857
	MAPS	4.03	130 (-11%)	3447 (-5%)	2295 (-50%)	6.37	57 (-3%)	7850 (+3%)	2060 (-28%)
RS	baseline	4.23	124370	1710	974	4.95	105351	1502	875
	MAPS	4.30	79222 (-36%)	1546 (-10%)	828 (-15%)	5.61	76040 (-28%)	1598 (+6%)	820 (-6%)
DR	baseline	3.70	520021	625	432	5.06	340021	483	236
	MAPS	3.80	400021 (-23%)	630 (+1%)	427 (-1%)	4.64	260021 (-24%)	480 (-1%)	214 (-9%)
AVERAGE			-29%	-9%	-25%		-25%	-8%	-14%

Table 3: Runtime and schedule statistics.

	Runtime (seconds)		Operations/cycle	
	baseline	MAPS	baseline	MAPS
XORR	56.0	64.7	387	773
GFMUL	4.3	11.1	28	85
CLZ	24.0	29.7	38	225
CRC	3.9	11.8	10	19
MD5	15.6	28.8	8	10
AES	20.5	61.9	16	24
SHA	8.9	19.6	16	22
DFADD	9.3	11.1	25	27
MT	36.5	193.5	8	11
RS	23.0	24.6	8	9
DR	44.5	50.5	9	11

sidering mapping, thus creating the opportunity for more aggressive operation chaining to minimize the schedule latency and area.

6. DISCUSSIONS

In this section, we will draw the distinction between MAPS and other integrated scheduling techniques and outline the relationship among MAPS, SDC scheduling, and retiming.

MAPS jointly performs scheduling and mapping to enable more aggressive operation chaining that is otherwise not possible with conventional scheduling approaches. The MD5 example in Figure 1(c) demonstrates that even the most accurate delay estimates obtained using the post-place-and-route back-annotation approach proposed in Zheng et al. [31] fails to reveal that mapping all operations into a single LUT and scheduling them within the same cycle leads to the shortest latency. In fact, such an approach would quickly converge to a sub-optimal solution, like the one in Figure 1(b). The failure stems from the reliance on the additive delay model commonly assumed throughout HLS tools, which simply sums up the delay of the critical path to determine the necessary clock boundaries to meet timing. Such an additive delay model is inaccurate in the perspective of downstream physical implementation because it does not consider mapping optimizations that are able to cluster multiple operations into a single LUT. The fact that the flow proposed in [31] focuses on the accuracy of delay estimates, while MAPS emphasizes the fundamental property of LUT mapping, [31] serves as a complement to the MAPS framework.

Although MAPS is designed to efficiently handle various difference constraints, the problem it addresses cannot be easily solved by the SDC scheduling approach [30]. Specifically, the task of computing and using delay information in the presence of mapping is non-trivial. MAPS determines the minimum L-value of each node based on cut enumeration, which selects one of the possible cones rooted at the node. Such either-or constraints cannot be optimally handled by SDC without relying on a heuristic transformation.

On the other hand, MAPS bears similarity to the retiming-based mapping technique [24], but is in fact

a generalization of the retiming-based mapping problem. Retiming-based mapping starts with a timed circuit with an initial register placement and aims to reposition these registers optimally such that the timing is met. MAPS instead takes in an untimed behavioral description and optimally add registers in between operations to achieve the minimum latency while meeting timing. An intuitive attempt to reduce the MAPS scheduling problem to the retiming-based mapping problem is to add different numbers of registers at the end of the input untimed circuit and allow the retiming-based algorithm to optimally propagate the registers into the circuit. While this approach simply requires a binary search on the number of registers, it is not optimal in the presence of cycles. The retiming-based mapping approach is unable to propagate the register into a cycle and will result in an infeasible mapping. It is possible to experiment with different initial register placements, but the complexity will be exponential, rendering the approach impractical.

7. CONCLUSIONS

In this paper, we propose a mapping-aware scheduling algorithm, which can efficiently incorporate mapping information into scheduling in order to generate much more efficient solutions for LUT-based FPGAs. Unlike conventional scheduling algorithms, our proposed algorithm jointly performs cut-based mapping and relaxation-based scheduling while respecting a variety of dependence, relative timing, latency, and resource constraints. We show that our algorithm can efficiently compute an optimized scheduling solution with reasonable execution time. Experimental results demonstrate that our proposed technique can achieve very promising improvements in performance and resource usage by enabling more aggressive operation chaining and maximizing the utilization of each LUT compared to the state-of-the-art commercial HLS tools.

8. ACKNOWLEDGMENTS

This work was supported in part by NSF Award CCF-1337240 and a research gift from Xilinx, Inc. We thank Dr. Peichen Pan at Xilinx for sharing his helpful insights on the problem of joint retiming and technology mapping.

9. REFERENCES

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson. A Comparison of List Schedules for Parallel Processing Systems. *Communications of the ACM*, 17(12):685–690, Dec 1974.

[2] A. Agarwal, M. C. Ng, and Arvind. A Comparative Evaluation of High-Level Hardware Synthesis Using Reed–Solomon Decoder. *IEEE Embedded Systems Letters*, 2(3):72–76, 2010.

[3] T. Beyrouthy and L. Fesquet. An Asynchronous FPGA Block with its Tech-Mapping Algorithm Dedicated to Security Applications. *International Journal of Reconfigurable Computing*, 2013.

[4] A. Canis, S. Brown, and J. Anderson. Modulo SDC Scheduling with Recurrence Minimization in High-Level Synthesis. *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2014.

[5] D. Chen and J. Cong. DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs. *Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 752–759, 2004.

[6] D. Chen, J. Cong, and P. Pan. FPGA Design Automation: A Survey. *Foundations and Trends in Electronic Design Automation*, 1(3):139–169, 2006.

[7] G. Chen and J. Cong. Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 48–55, 2001.

[8] J. Cong and Y. Ding. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 13(1):1–12, 1994.

[9] J. Cong and Y.-Y. Hwang. Simultaneous Depth and Area Minimization in LUT-based FPGA Mapping. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 68–74, 1995.

[10] J. Cong, B. Liu, G. Luo, and R. Prabhakar. Towards Layout-Friendly High-Level Synthesis. *Int'l Symp. on Physical design (ISPD)*, pages 165–172, 2012.

[11] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 30(4):473–491, Apr 2011.

[12] J. Cong, B. Liu, and Z. Zhang. Scheduling with Soft Constraints. *Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 47–54, 2009.

[13] J. Cong and K. Minkovich. LUT-based FPGA Technology Mapping for Reliability. *Design Automation Conf. (DAC)*, pages 517–522, 2010.

[14] J. Cong, C. Wu, and Y. Ding. Cut Ranking and Pruning: Enabling a General and Efficient FPGA Mapping Solution. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 29–35, 1999.

[15] J. Cong and Z. Zhang. An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation. *Design Automation Conf. (DAC)*, pages 433–438, Jul 2006.

[16] S. Dai, M. Tan, K. Hao, and Z. Zhang. Flushing-Enabled Loop Pipelining for High-Level Synthesis. pages 1–6, 2014.

[17] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. *Int'l Symp. on Workload Characterization (IISWC)*, pages 3–14, 2001.

[18] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii. CHStone: A Benchmark Program Suite for Practical C-Based High-Level Synthesis. *Int'l Symp. on Circuits and Systems (ISCAS)*, pages 1192–1195, 2008.

[19] C. Lattner and V. Adve. LLVM: a Compilation Framework for Lifelong Program Analysis & Transformation. *Int'l Symp. on Code Generation and Optimization (CGO)*, pages 75–86, 2004.

[20] Y. Lee. Handwritten Digit Recognition Using K Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks. *Neural computation*, 3(3):440–449, 1991.

[21] J. Y. Lin, D. Chen, and J. Cong. Optimal Simultaneous Mapping and Clustering for FPGA Delay Optimization. *Design Automation Conf. (DAC)*, pages 472–477, 2006.

[22] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[23] P. Pan, A. K. Karandikar, and C. Liu. Optimal Clock Period Clustering for Sequential Circuits with Retiming. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 17(6):489–498, 1998.

[24] P. Pan and C.-C. Lin. A New Retiming-Based Technology Mapping Algorithm for LUT-based FPGAs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 35–42, 1998.

[25] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 8(6):661–678, Jun 1989.

[26] V. Pratt. Two Easy Theories Whose Combination is Hard. Technical report, MIT, 1977.

[27] R. Rivest. The MD5 Message-Digest Algorithm. 1992.

[28] M. Tan, B. Liu, S. Dai, and Z. Zhang. Multithreaded Pipeline Synthesis for Data-parallel Kernels. *Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 718–725, 2014.

[29] J. Zhang, Z. Zhang, S. Zhou, M. Tan, X. Liu, X. Cheng, and J. Cong. Bit-level optimization for high-level synthesis and FPGA-based acceleration. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 59–68, 2010.

[30] Z. Zhang and B. Liu. SDC-Based Modulo Scheduling for Pipeline Synthesis. *Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 211–218, 2013.

[31] H. Zheng, S. T. Gurumani, K. Rupnow, and D. Chen. Fast and Effective Placement and Routing Directed High-Level Synthesis for FPGAs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 1–10, 2014.